**浪潮信息边缘计算缘脑平台 V2.0**

**技术白皮书**

目录

[1 前言 1](#_Toc179877221)

[2 缘脑平台概述 3](#_Toc179877222)

[2.1 产品定位 3](#_Toc179877223)

[2.2 产品架构 3](#_Toc179877224)

[2.2.1 技术架构 3](#_Toc179877225)

[2.2.2 功能架构 6](#_Toc179877226)

[3 缘脑平台基础技术原理 6](#_Toc179877227)

[3.1 容器核心技术与原理 6](#_Toc179877228)

[3.1.1 容器 6](#_Toc179877229)

[3.1.2 容器编排系统Kubernetes 9](#_Toc179877230)

[4 缘脑平台功能特色和实现原理 15](#_Toc179877231)

[4.1 边缘资源池和边缘节点接入 15](#_Toc179877232)

[4.1.1 功能描述 15](#_Toc179877233)

[4.1.2 技术原理 16](#_Toc179877234)

[4.1.3 约束 17](#_Toc179877235)

[4.2 边缘自治 17](#_Toc179877236)

[4.2.1 功能描述 17](#_Toc179877237)

[4.2.2 技术原理 18](#_Toc179877238)

[4.2.3 约束 19](#_Toc179877239)

[4.3 云边单向网络隧道 19](#_Toc179877240)

[4.3.1 功能描述 19](#_Toc179877241)

[4.3.2 技术原理 20](#_Toc179877242)

[4.3.3 约束 21](#_Toc179877243)

[4.4 容器镜像服务 21](#_Toc179877244)

[4.4.1 功能描述 21](#_Toc179877245)

[4.4.2 技术原理 23](#_Toc179877246)

[4.4.3 约束 23](#_Toc179877247)

[4.5 应用空间管理 24](#_Toc179877248)

[4.5.1 功能描述 24](#_Toc179877249)

[4.5.2 技术原理 24](#_Toc179877250)

[4.5.3 约束 24](#_Toc179877251)

[4.6 工作负载管理 25](#_Toc179877252)

[4.6.1 功能描述 25](#_Toc179877253)

[4.6.2 技术原理 27](#_Toc179877254)

[4.6.3 约束 27](#_Toc179877255)

[4.7 网络管理 28](#_Toc179877256)

[4.7.1 服务 28](#_Toc179877257)

[4.7.2 功能描述 28](#_Toc179877258)

[4.7.3 技术原理 29](#_Toc179877259)

[4.7.4 约束 30](#_Toc179877260)

[4.8 存储管理 30](#_Toc179877261)

[4.8.1 多种存储类型对接 30](#_Toc179877262)

[4.8.2 功能描述 30](#_Toc179877263)

[4.8.3 技术原理 31](#_Toc179877264)

[4.8.4 约束 32](#_Toc179877265)

[4.9 存储卷管理 33](#_Toc179877266)

[4.9.1 功能描述 33](#_Toc179877267)

[4.9.2 技术原理 34](#_Toc179877268)

[4.9.3 约束 34](#_Toc179877269)

[4.10 配置中心 35](#_Toc179877270)

[4.10.1 功能描述 35](#_Toc179877271)

[4.10.2 技术原理 36](#_Toc179877272)

[4.10.3 约束 36](#_Toc179877273)

[4.11 运营服务 37](#_Toc179877274)

[4.11.1 统一用户认证 37](#_Toc179877275)

[4.11.1.1 功能描述 37](#_Toc179877276)

[4.11.1.2 技术原理 37](#_Toc179877277)

[4.11.1.3 约束 38](#_Toc179877278)

[4.11.2 访问控制 38](#_Toc179877279)

[4.11.2.1 功能描述 38](#_Toc179877280)

[4.11.2.2 技术原理 39](#_Toc179877281)

[4.11.2.3 约束 40](#_Toc179877282)

[4.11.3 双因子认证 40](#_Toc179877283)

[4.11.3.1 功能描述 40](#_Toc179877284)

[4.11.3.2 技术原理 40](#_Toc179877285)

[4.11.3.3 约束 41](#_Toc179877286)

[5 缘脑平台安全可靠设计 41](#_Toc179877287)

[5.1 多维度安全体系设计 41](#_Toc179877288)

[5.2 可靠性设计 42](#_Toc179877289)

[5.2.1 容器资源池可靠性 43](#_Toc179877290)

[5.2.2 云管理层可靠性 46](#_Toc179877291)

[5.2.2.1 总体介绍 46](#_Toc179877292)

[5.2.2.2 数据库高可靠 47](#_Toc179877293)

[5.2.2.2.1 服务介绍 47](#_Toc179877294)

[5.2.2.2.2 产品架构 48](#_Toc179877295)

[5.2.2.2.3 功能特性 48](#_Toc179877296)

[5.2.2.3 消息队列高可靠 48](#_Toc179877297)

[5.2.2.3.1 服务介绍 48](#_Toc179877298)

[5.2.2.3.2 产品架构 49](#_Toc179877299)

[5.2.2.3.3 功能特性 49](#_Toc179877300)

[5.2.2.4 微服务组件高可靠 50](#_Toc179877301)

[5.2.2.4.1 服务介绍 50](#_Toc179877302)

[5.2.2.4.2 产品架构 51](#_Toc179877303)

[5.2.2.4.3 功能特性 51](#_Toc179877304)

# 前言

文档用途

本文档用于描述缘脑平台功能特色、技术原理及适用场景。

适用范围

本文档为公司内部、外部了解缘脑平台技术实现的参考文档。

读者对象

本文档提供给以下相关人员使用：

* 客户代表
* 产品经理
* 售前工程师
* 系统维护工程师
* 研发工程师

安全声明

您购买的产品在业务运营或故障定位的过程中可能会获取或使用用户的某些个人数据（如告警邮件接收地址、IP地址），因此您有义务根据所适用国家或地区的法律法规制定必要的用户隐私政策并采取足够的措施以确保用户的个人数据受到充分的保护。

浪潮信息将一如既往的严密关注产品与解决方案的安全性，为客户提供更满意的服务。浪潮信息已全面建立产品安全漏洞应急和处理机制，确保第一时间处理产品安全问题。若您在本产品使用过程中发现任何安全问题，或者寻求有关产品安全漏洞的必要支持，请直接联系浪潮信息客户服务人员。

# 缘脑平台概述

浪潮信息致力于成为中国领先的边缘计算解决方案供应商，提供面向未来智慧边缘计算的整体解决方案，凭借浪潮信息边缘服务器、边缘盒子、边缘计算操作系统、信息安全技术为客户打造领先的边缘计算基础架构，全面支撑各种边缘计算业务场景。浪潮信息缘脑平台基于开放架构，兼容多元异构平台，满足不同应用场景需求。缘脑平台以开源Kubernetes为核心，直接作用于异构复杂硬件基础设施之上，能够对数据中心和边缘基础设施资源池进行管理和调度、并支撑各类企业业务应用高效、安全、可靠和稳定运行。

## 产品定位

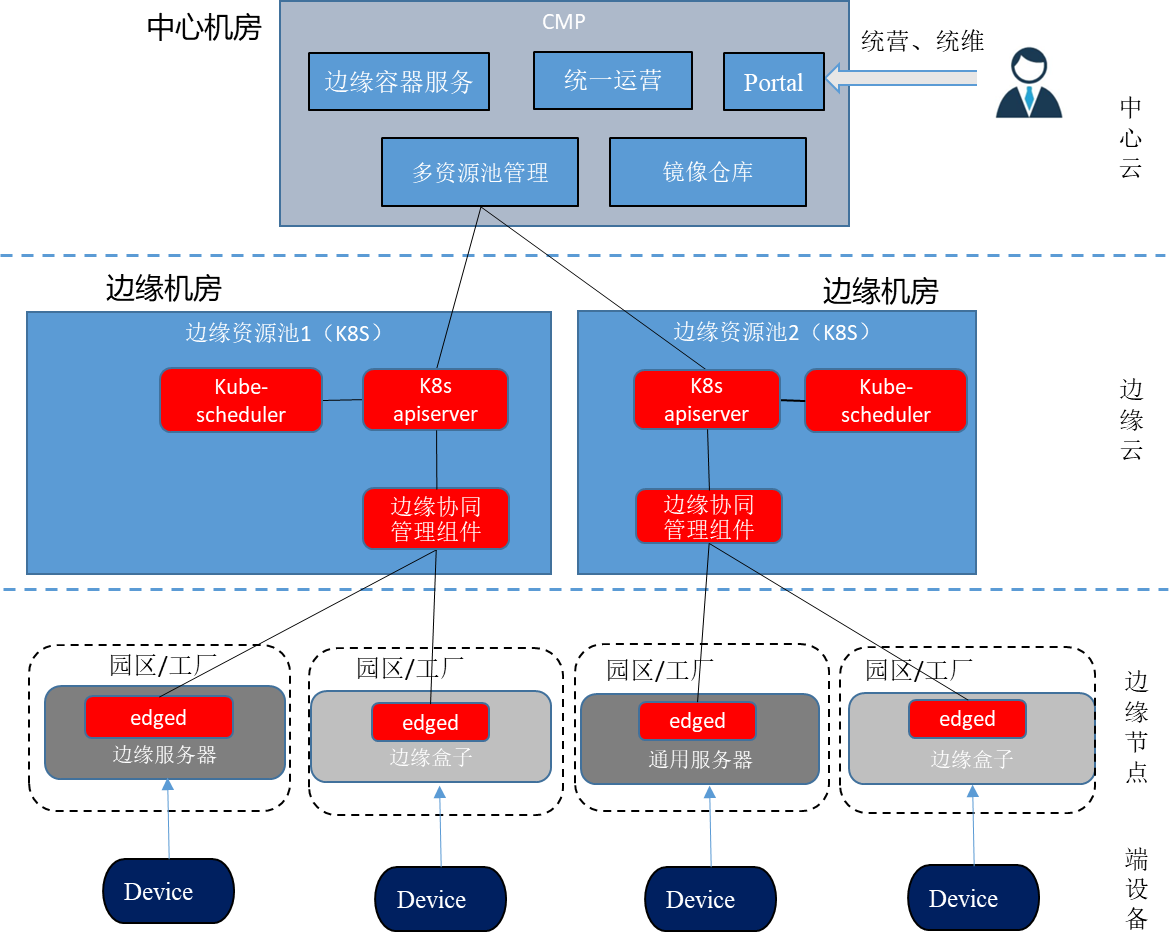
浪潮信息缘脑平台采用业界先进的架构设计理念和技术架构，驱动产品在可靠性、稳定性、性能、易用性等方面的全面提升。依托浪潮信息在边缘服务器和边缘盒子的优势计算能力，将计算和服务能力下沉到边缘侧，构建软硬一体的边缘计算平台。为终端用户提供从中心云到边缘云，再到现场的边缘节点各个层次和场景的计算服务，支撑用户业务持续创新。缘脑平台提供基于容器的各类应用的生命周期管理，通过统一的运营运维门户，保证一致性的用户体验。

## 产品架构

### 技术架构

缘脑平台采用服务化的架构设计理念，利用精简化的服务组件构建一个个子系统，各个子系统之间采用REST API进行交互，每个子系统可以独立运行，从而取代大集成式的传统基础架构（基于重量级、非标准化的平台）。

缘脑平台中，每一种服务被拆分成最小的组件，各组件可做到高内聚低耦合，彼此独立，因此每项服务都能独立扩展，并能根据业务需求，快速灵活的做到精简部署和配置，从而满足各种复杂的业务场景。同时，独立化组件服务具备故障隔离功能，可将各种服务问题（例如计算服务故障或存储服务故障）的影响限制在一定范围内，以确保不对其他服务产生影响。



如上图所示，缘脑平台包括CMP（云统一管理系统平台）和边缘容器服务大核心模块。

CMP层采用微服务架构的设计理念，基于K8S+SpringBoot的架构模式。每个服务节点可分布式独立部署，UI通过Rest API网关调用功能服务,对外可提供符合RESTful规范的API接口供第三方集成。云管平台微服务框架可实现多服务节点间服务的负载均衡、服务路由、集群容错等机制，使系统具有很强的连通性、健壮性、伸缩性和扩展性，CMP层主要包括统一资源管理、运营管理等核心功能特征。

**统一资源管理：**

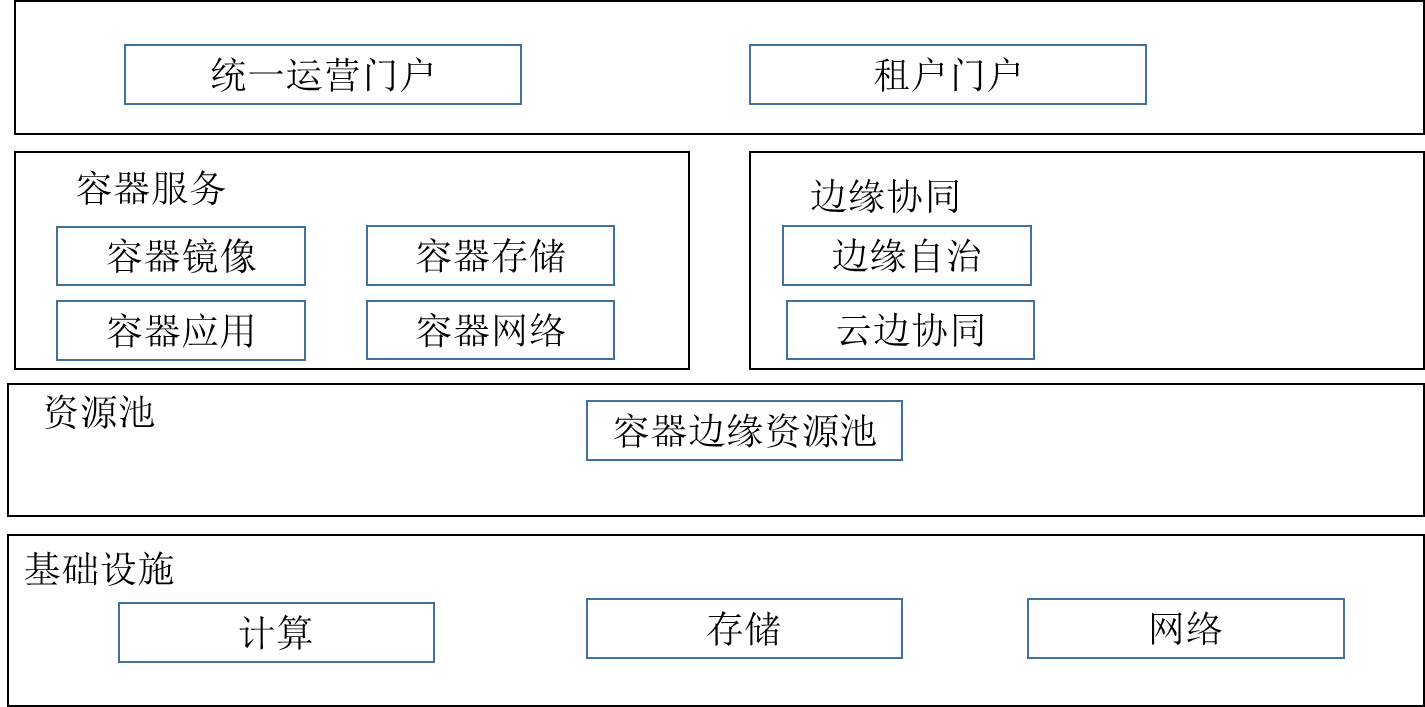
* 定义统一的资源管理模型-VDC（虚拟数据中心），包括统一的配额管理，资源与组织模型（用户、角色等）的关系等，将K8S的namespace在cmp层进行一致的资源管理，同时可作为第三方上云接入的资源管理的标准模型。

**运营管理：**定义统一用户角色模型并实现统一的多租户管理、申请审批、工单管理、资源计量计费等，为用户提供运营管理员、运维管理员、租户管理员和租户用户等多种角色，满足用户多种运营模式需求。

容器云服务以容器应用管理为中心，该模块基于开源Kubernetes架构实现，并进行增强和扩展，向提供标准的符合Kubernetes规范要求的REST API，以声明式API+控制器为核心设计理念，通过CRD可以轻松定制和扩展云原生能力。

容器云服务基于插件式模式对接容器的运行时、存储以及网络，提供容器应用的计算、存储和网络资源，通过CRI（容器运行时接口）对接实现了shim层的各种容器运行时，包括docker、containerd等，通过CSI（容器存储接口）对接实现了插件层的存储系统，云端通过CNI（容器网络接口）对接各种网络插件，包括flannel、calico等，边端通过edgemesh实现云边、边边通信。

### 功能架构



**边缘服务：**包括边缘资源池管理、容器应用、容器存储、容器配置中心、容器网络以及容器镜像服务等。

**边缘协同**：包括边缘自治和云边协同，能够在离线的时候应用无影响，能够在中心统一下发应用到边缘节点。

**运营服务：**实现对统一用户权限管理、VDC管理、操作审计、计量计费、流程审批等功能，实现云服务的统一运营，云平台统一管理、维护和配置。

# 缘脑平台基础技术原理

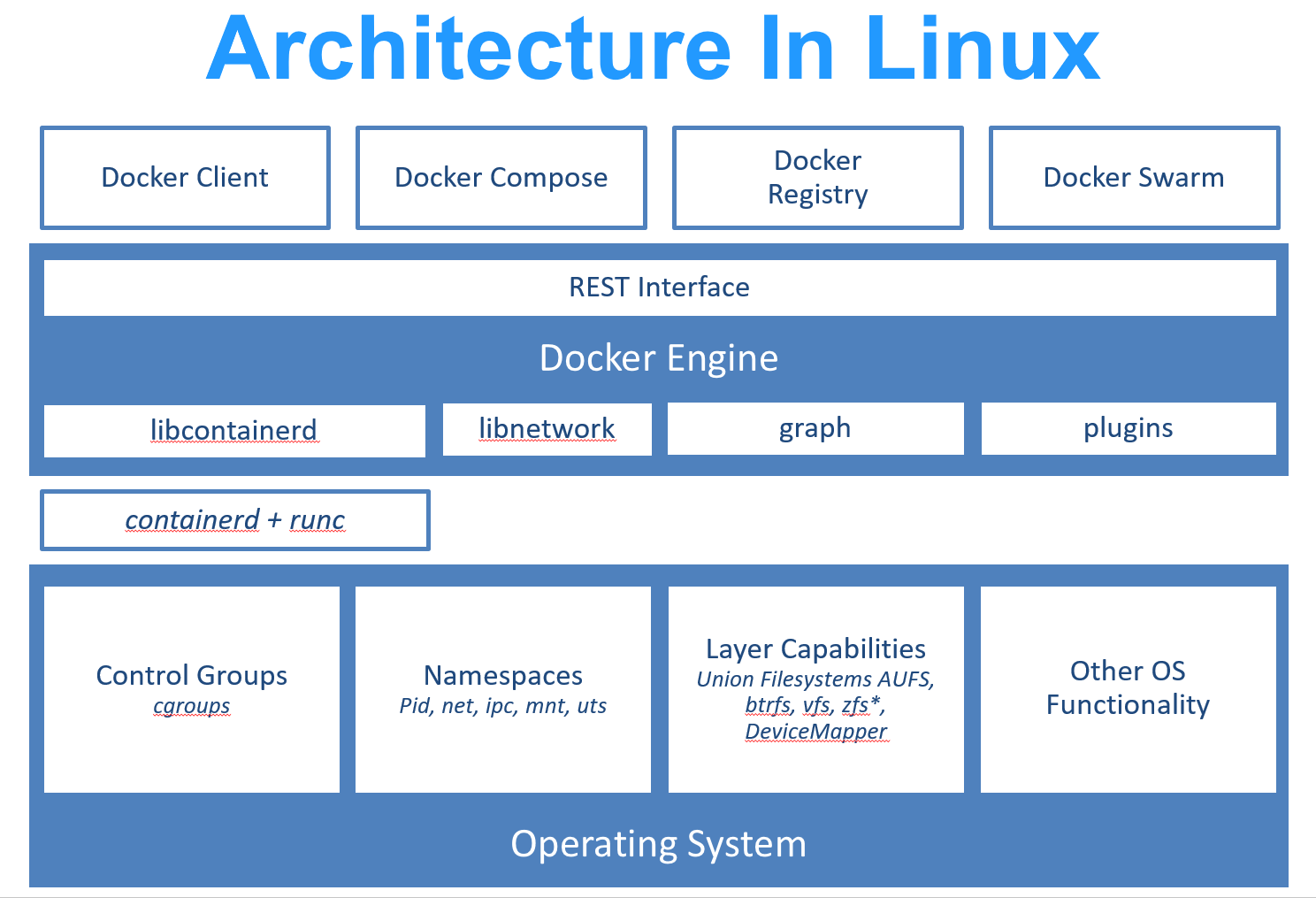
## 容器核心技术与原理

### 容器

Docker最初是dotCloud公司创始人Solomon Hykes在法国期间发起的一个公司内部项目，它是基于dotCloud公司多年云服务技术的一次革新，并于2013年3月以Apache 2.0授权协议开源，主要项目代码在GitHub上进行维护。Docker项目后来还加入了Linux基金会，并成立推动开放容器联盟（OCI）。

Docker自开源后受到广泛的关注和讨论，至今其GitHub项目已经超过5万2千个星标和一万多个fork。甚至由于Docker项目的火爆，在2013年底，dotCloud公司决定改名为Docker。Docker最初是在Ubuntu 12.04上开发实现的；RedHat则从RHEL6.5开始对Docker进行支持；Google也在其PaaS产品中广泛应用Docker。

Docker使用Google公司推出的Go语言进行开发实现，基于Linux内核的cgroup，namespace，以及AUFS类的Union FS等技术，对进程进行封装隔离，属于操作系统层面的虚拟化技术。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。最初实现是基于LXC，从0.7版本以后开始去除LXC，转而使用自行开发的libcontainer，从1.11开始，则进一步演进为使用runC和containerd。



Docker在容器的基础上，进行了进一步的封装，从文件系统、网络互联到进程隔离等等，极大的简化了容器的创建和维护。使得Docker技术比虚拟机技术更为轻便、快捷。作为一种新兴的虚拟化方式，Docker跟传统的虚拟化方式相比具有众多的优势。

更高效的利用系统资源：由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销，Docker对系统资源的利用率更高。无论是应用执行速度、内存损耗或者文件存储速度，都要比传统虚拟机技术更高效。因此，相比虚拟机技术，一个相同配置的主机，往往可以运行更多数量的应用。

**更快速的启动时间：**传统的虚拟机技术启动应用服务往往需要数分钟，而Docker容器应用，由于直接运行于宿主内核，无需启动完整的操作系统，因此可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。

**一致的运行环境：**开发过程中一个常见的问题是环境一致性问题。由于开发环境、测试环境、生产环境不一致，导致有些bug并未在开发过程中被发现。而Docker的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现「这段代码在我机器上没问题啊」这类问题。

**持续交付和部署：**对开发和运维（DevOps）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。使用Docker可以通过定制应用镜像来实现持续集成、持续交付、部署。开发人员可以通过Dockerfile来进行镜像构建，并结合持续集成（Continuous Integration）系统进行集成测试，而运维人员则可以直接在生产环境中快速部署该镜像，甚至结合持续部署（Continuous Delivery/Deployment）系统进行自动部署。而且使用Dockerfile使镜像构建透明化，不仅仅开发团队可以理解应用运行环境，也方便运维团队理解应用运行所需条件，帮助更好的生产环境中部署该镜像。

**更轻松的迁移：**由于Docker确保了执行环境的一致性，使得应用的迁移更加容易。Docker可以在很多平台上运行，无论是物理机、虚拟机、公有云、私有云，甚至是笔记本，其运行结果是一致的。因此用户可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。

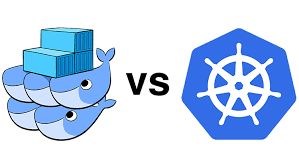
**更轻松的维护和扩展：**Docker使用的分层存储以及镜像的技术，使得应用重复部分的复用更为容易，也使得应用的维护更新更加简单，基于基础镜像进一步扩展镜像也变得非常简单。此外，Docker团队同各个开源项目团队一起维护了一大批高质量的 官方镜像，既可以直接在生产环境使用，又可以作为基础进一步定制，大大的降低了应用服务的镜像制作成本。

### 容器编排系统Kubernetes

容器在服务运维上提供了较大的便利，但是对于大规模的情况，显得有些无力。Kubernetes基本上是这两年最热门、最被人熟知的技术了，它为软件工程师提供了强大的容器编排能力，模糊了开发和运维之间的边界，让我们开发、管理和维护一个大型的分布式系统和项目变得更加容易。

作为一个目前在生产环境已经广泛使用的开源项目Kubernetes被定义成一个用于自动化部署、扩容和管理容器应用的开源系统；它将一个分布式软件的一组容器打包成一个个更容易管理和发现的逻辑单元。

Kubernetes是希腊语『舵手』的意思，它最开始由Google的几位软件工程师创立，深受公司内部Borg和Omega项目的影响，很多设计都是从Borg中借鉴的，同时也对Borg的缺陷进行了改进，Kubernetes目前是Cloud Native Computing Foundation (CNCF)的项目并且是很多公司管理分布式系统的解决方案。



在Kubernetes统治了容器编排这一领域之前，其实也有很多容器编排方案，例如compose和Swarm，但是在运维大规模、复杂的集群时，这些方案基本已经都被Kubernetes替代了。

1、Kubernetes设计理念：

（1）声明式

声明式（Declarative）的编程方式一直都会被工程师们拿来与命令式（Imperative）进行对比，这两者是完全不同的编程方法。我们最常接触的其实是命令式编程，它要求我们描述为了达到某一个效果或者目标所需要完成的指令，常见的编程语言Go、Ruby、C++其实都为开发者了命令式的编程方法，在Kubernetes中，我们可以直接使用YAML文件定义服务的拓扑结构和状态：

apiVersion: v1

kind: Pod

metadata:

name: rss-site

labels:

app: web

spec:

containers:

- name: front-end

image: nginx

ports:

- containerPort: 80

- name: rss-reader

image: nickchase/rss-php-nginx:v1

ports:

- containerPort: 88

这种声明式的方式能够大量地减少使用者的工作量，极大地增加开发的效率，这是因为声明式能够简化需要的代码，减少开发人员的工作，如果我们使用命令式的方式进行开发，虽然在配置上比较灵活，但是带来了更多的工作。

Kubernetes中声明式的API其实指定的是集群期望的运行状态，所以在出现任何不一致问题时，它本身都可以通过指定的YAML文件对线上集群进行状态的迁移，就像一个水平触发的系统，哪怕系统错过了相应的事件，最终也会根据当前的状态自动做出做合适的操作。

（2）显式接口

第二个Kubernetes的设计规范其实就是——不存在内部的私有接口，所有的接口都是显示定义的，组件之间通信使用的接口对于使用者来说都是显式的，我们都可以直接调用。

当Kubernetes的接口不能满足工程师的复杂需求时，我们需要利用已有的接口实现更复杂的特性，在这时Kubernetes的这一设计就不会成为自定义需求的障碍。

（3）无侵入性

为了尽可能满足用户（工程师）的需求，减少工程师的工作量与任务并增强灵活性，Kubernetes为工程师提供了无侵入式的接入方式，每一个应用或者服务一旦被打包成了镜像就可以直接在Kubernetes中无缝使用，不需要修改应用程序中的任何代码。

Docker和Kubernetes就像包裹在应用程序上的两层，它们两个为应用程序提供了容器化以及编排的能力，在应用程序内部却不需要任何的修改就能够在Docker和Kubernetes集群中运行，这是Kubernetes在设计时选择无侵入带来最大的好处，同时无侵入的接入方式也是目前几乎所有应用程序或者服务都必须考虑的一点。

（4）可移植性

在微服务架构中，我们往往都会让所有处理业务的服务变成无状态的服务，以前在内存中存储的数据、Session等缓存，现在都会放到缓存数据库、ETCD等数据库中存储，微服务架构要求我们对业务进行拆分并划清服务之间的边界，所以有状态的服务往往会对架构的水平迁移带来障碍。

然而有状态的服务其实是无可避免的，我们将每一个基础服务或者业务服务都变成了一个个只负责计算的进程，但是仍然需要有其他的进程负责存储易失的缓存和持久的数据，Kubernetes对这种有状态的服务也提供了比较好的支持。

Kubernetes引入了PersistentVolume和PersistentVolumeClaim的概念用来屏蔽底层存储的差异性，目前的Kubernetes支持下列类型的PersistentVolume：

这些不同的PersistentVolume会被开发者声明的PersistentVolumeClaim分配到不同的服务中，对于上层来讲所有的服务都不需要接触PersistentVolume，只需要直接使用PersistentVolumeClaim得到的卷就可以了。

2、Kubernetes特性

（1）自动包装：根据资源需求和其他约束自动放置容器，同时不会牺牲可用性，混合关键和最大努力的工作负载，以提高资源利用率并节省更多资源。

（2）横向缩放：使用简单的命令或UI，或者根据CPU的使用情况自动调整应用程序副本数。

（3）自动部署和回滚：Kubernetes逐渐部署对应用程序或其配置的更改，同时监视应用程序运行状况，以确保它不会同时终止所有实例。如果出现问题，Kubernetes会为您恢复更改，利用日益增长的部署解决方案的生态系统。

（4）存储编排：自动安装您所选择的存储系统，无论是本地存储，如公有云提供商GCP或AWS，还是网络存储系统NFS、iSCSI、Gluster、Ceph、Cinder,或Flocker。

（5）自我修复：重新启动失败的容器，在节点不可用时，替换和重新调度节点上的容器，对用户定义的健康检查不响应的容器会被中止，并且在容器准备好服务之前不会把其向客户端广播。

（6）服务发现和负载均衡：不需要修改您的应用程序来使用不熟悉的服务发现机制，Kubernetes为容器提供了自己的IP地址和一组容器的单个DNS名称，并可以在它们之间进行负载均衡。

（7）秘钥和配置管理：部署和更新密钥和应用程序配置，不会重新编译您的镜像，不会在堆栈配置中暴露密钥（secrets）。

（8）批处理：除了服务之外，Kubernetes还可以管理批处理和CI工作负载，如果需要，替换出现故障的容器。

# 缘脑平台功能特色和实现原理

## 边缘资源池和边缘节点接入

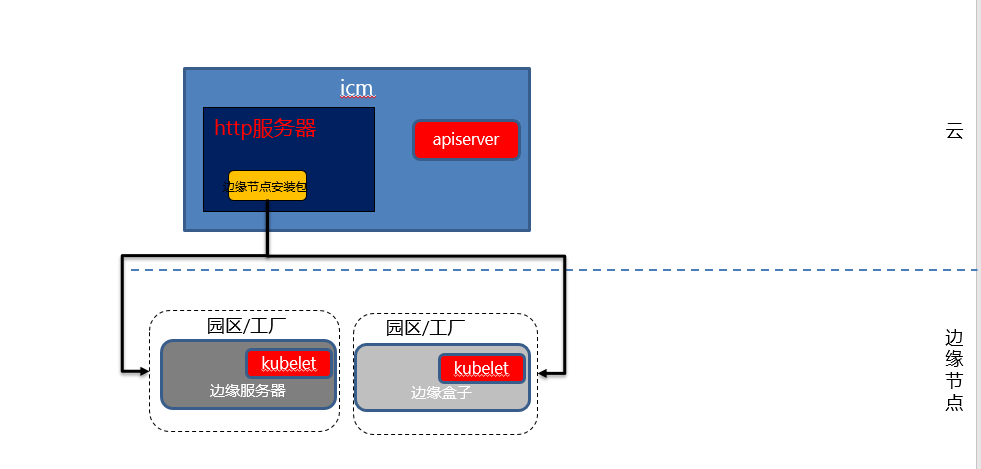
### 功能描述

边缘资源池为一个K8S容器集群，边缘节点为K8S集群里的工作节点，承载具体的容器应用的部署。

集群服务功能包括：

* 集群接入管理：支持通过证书配置、用户名密码两种形式接入、修改、移除不同CPU架构的集群。
* 集群概览信息采集及展示：包括基本信息、节点状态统计、集群管理组件状态统计、CPU/内存/存储等各类资源使用情况。
* 集群节点管理：基本信息查看、运行状态、容器组列表、资源用量及历史曲线、是否可调度管理、标签管理、维护模式管理等功能。
* 虚拟数据中心绑定信息：支持与当前集群绑定的虚拟数据中心资源配额、使用量展示。
* 边缘节点接入：支持将不同架构的边缘节点一键接入。

### 技术原理



1. 首先需要将容器集群添加到边缘资源池，添加完成后资源池为可用状态。
2. 在边缘资源池的节点详情页面点击新增边缘节点按钮，弹出框可输入边缘节点名称，点击生成部署命令按钮，此时，调用指定的http接口返回包含集群token、边缘节点安装包地址等信息的Linux命令。
3. 登录边缘节点Linux服务器，复制上面Linux命令执行后，即可接入边缘节点。其中，通过EdgeHub与云端进行通信，EdgeHub中有两类client，分别是httpclient以及websocket/quic client，前者用于与EdgeCore与CloudCore通信所需证书的申请，后者负责与CloudCore的日常通信（资源下发、状态上传等）

Edged管理的node、pod的状态信息，先报到MetaManager，再传到EdgeHub，最终把数据同步到云上。这样就实现了node、pod状态的上报。当断网重启后，

### 约束

无。

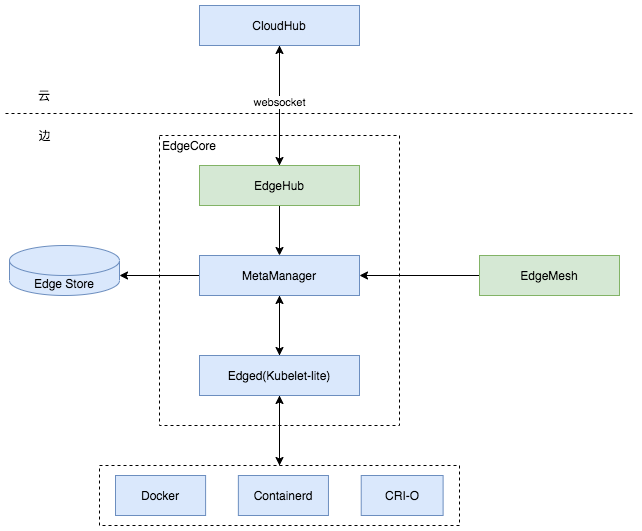
## 边缘自治

### 功能描述

对于标准的Kubernetes，如果节点断网失联后，节点上的Pod将不能正常提供服务；网络长时间断连，云端控制器将对业务容器进行驱逐；节点重启后，内存数据丢失，容器也不会被拉起。

而云端边缘自治就是为解决上述问题的。当边缘节点断网失联后能正确提供服务；边缘节点发生异常重启后，需要借助本地缓存数据实现Pod能被重新拉起并正常运行，所有的微服务可以被正常访问；长时间断连后网络恢复时，边缘和云端数据能快速同步，最终保持一致性。

### 技术原理



1. 通过EdgeHub与云端进行通信，EdgeHub中有两类client，分别是httpclient以及websocket/quic client，前者用于与EdgeCore与CloudCore通信所需证书的申请，后者负责与CloudCore的日常通信（资源下发、状态上传等）
2. Edged管理的node、pod的状态信息，先报到MetaManager，再传到EdgeHub，最终把数据同步到云上。这样就实现了node、pod状态的上报。当断网重启后，Edged从本地数据库中拿到相应数据进行服务恢复
3. CloudHub下发事件到EdgeHub，MessageDispatcher组件将请求下发到不同的模块，如果是pod的元数据，就推给MetaManager再转给Edged去拉起相应容器；同时将相应的数据保存在数据库中。对于下发的事件加了消息完整性校验机制，只有边缘收到数据且回复完整性校验通过信息，云端才认为发送成功，反之则进行重试，针对未发送的消息实时检查合并去重，避免网络震荡。
4. 长时间断连后网络恢复时，云端将同步来自CloudCore的最新消息更新本地元数据，最终保持数据一致性。

### 约束

无。

## 云边单向网络隧道

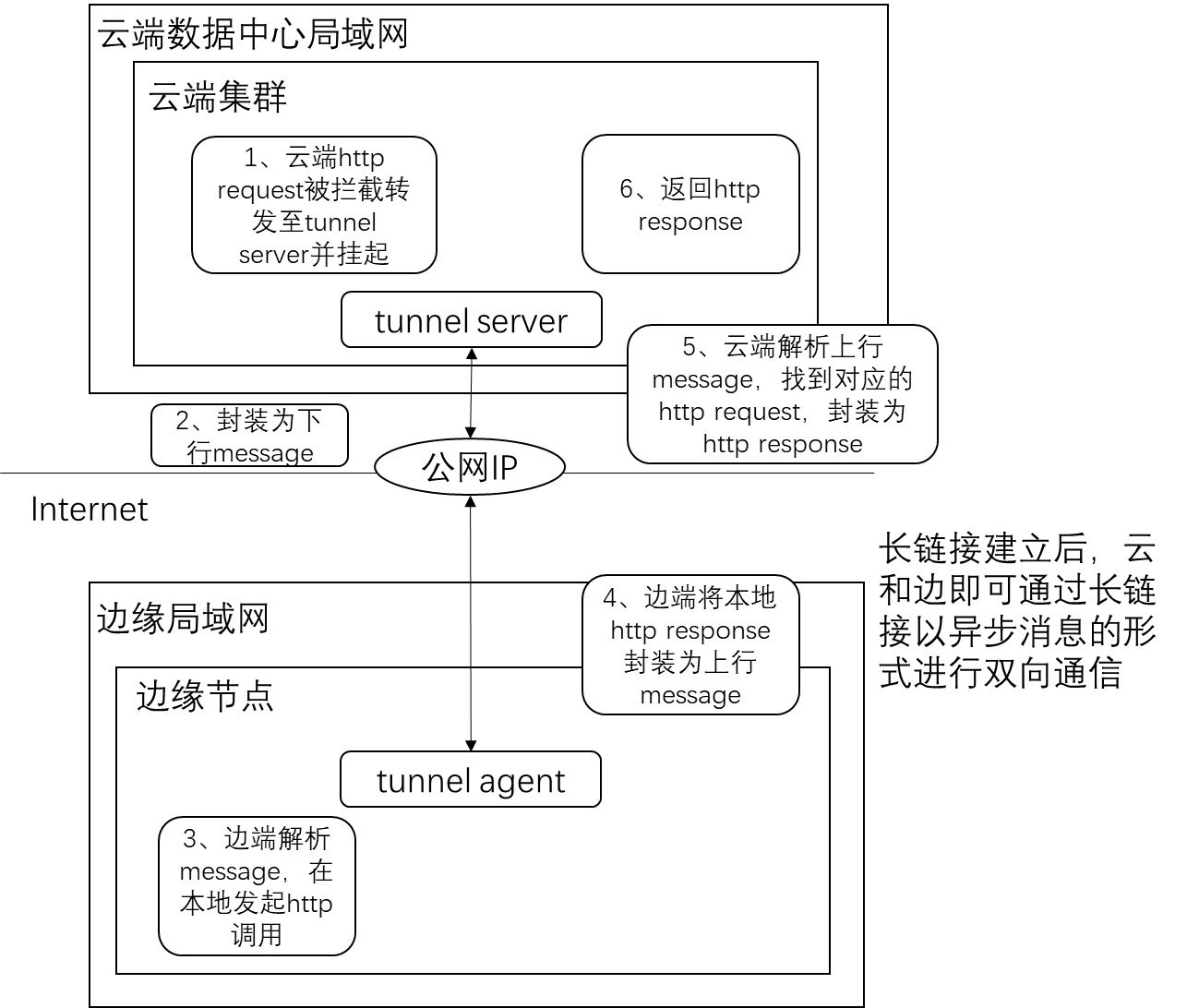
### 功能描述

在边缘计算场景中，k8s集群的边缘节点常位于本地专有网络中，云端管控节点位于公网中，边缘节点可以通过公网ip访问云端控节点，但是云管控节点访问边缘节点无法访问网络不通，云端与边缘之间网络单向连通；这样就出现云端管控节点无法访问边缘节点问题。

而单向网络通信隧道技术就是为解决上述问题的。边缘场景下，云端管控节点请求无法正常通过网络请求到边缘节点；在云端管控节点和边缘节点之间增加一个网络隧道，云端节点的请求全部通过这个网络隧道正常到达边缘节点，所有的请求正常访问，最终保证云端请求顺利的下发到边缘。

### 技术原理

组件物理拓扑图



云边通信隧道原理

1. 边缘云端：云端配置路由拦截规则，将指向边端的http请求拦截到云端的tunnel server上；或者由云端的controller主动调用云端tunnelserver接口，下发请求。
2. 边缘节点：实现tunnel client，主动与tunnelserver建立长连接，将自身信息上报给云端。同时实现message解析逻辑，从云端下发的message中解析出目标地址、目标协议，在本地发起调用。
3. tunnel client与tunnel server两个组件间的长链接协议支持websocket和quic两 种，其中websocket方案较为成熟，quic性能较好
4. 云端controller和tunnelserver都封装为一个cloudcore组件，二者之间的调用为进程内协程间的message通信。
5. 云边控制流和运维流通过两个长链接通道实现，技术原理一致，由于长链接隧道是顺序读写，这样可以防止双方流量互相影响；控制流包括云端对象的创建、变更等消息下发到边端、边端资源状态变化上报到云端；运维流包括云端主动执行的exec、metrics、logs等k8s-apiserver主动向边端发起调用的流量；

### 约束

无。

## 容器镜像服务

### 功能描述

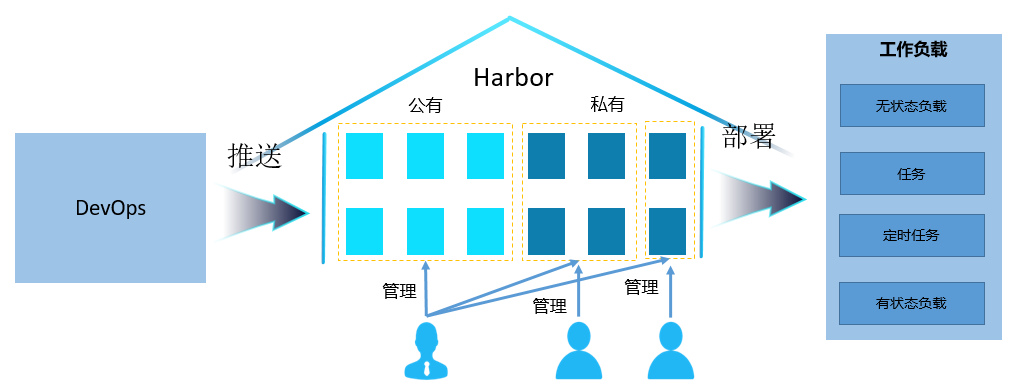
镜像仓库服务是一种基于harbor实现的镜像服务，支持镜像全生命周期管理的服务包含上传、拉取、推送及删除等镜像操作 ，提供简单易用、安全可靠的镜像管理功能，帮助租户快速部署容器化服务；支持基于角色控制的企业级容器镜像管理能力，提供以虚拟数据中心为粒度的多租户镜像隔离服务，将镜像分为公有镜像和私有镜像，公有镜像可以被所有租户使用，私有镜像被对应虚拟数据中心的租户、对应域的域管理员和超级管理员使用，保障了镜像仓库的租户隔离和租户间的安全访问；多租户镜像仓库服务还支持第三方标准harbor仓库的接入和使用与多处理器架构镜像的统一管理的能力；同时，镜像仓库服务还提供了镜像漏洞扫描的服务来保障云平台的安全。

镜像仓库服务功能包含：

* 支持多租户镜像的统一管理：将镜像分为公有镜像和私有镜像，公有镜像可以被所有租户可以查询和使用，私有镜像是以虚拟数据中心为粒度对镜像进行的租户权限控制，超级管理员上传、删除、扫描公有镜像和私有镜像，域管理员与虚拟数据中心管理员只能上传、删除和扫描所在域和所在虚拟数据中的私有镜像。
* 第三方仓库管理：支持第三方harbor仓库的增加、删除、修改、仓库列表查看、镜像列表查看、镜像详情、镜像拉取的查看等功能。
* 镜像管理：支持上传镜像：使用UI上传镜像、推送镜像、拉取镜像、删除镜像、删除镜像版本、查看镜像列表、查看镜像详情的功能。
* 分类管理：支持分类类别的增加、删除、修改的功能。
* 镜像漏洞扫描：支持全库漏洞扫描，定时漏洞扫描与上传自动漏洞扫描的功能，并且支持全架构镜像无差别漏洞扫描，同时也支持了限制高危镜像的拉取操作的功能。

### 技术原理

容器云平台通过与Harbor的交互及扩展实现容器镜像管理功能，整体架构如下所示：



* 容器云平台调用Harbor REST API，实现镜像的分类管理、镜像信息查询、推送或拉取。
* 容器云平台支持与Harbor账户对接，通过账户与Harbor内部项目关联，实现镜像公有私有权限设置。

特色功能：

多处理器架构镜像的统一管理：支持X86、MIPS、ARM、ALPHA、POWER等多种处理架构镜像的统一管理

### 约束

无。

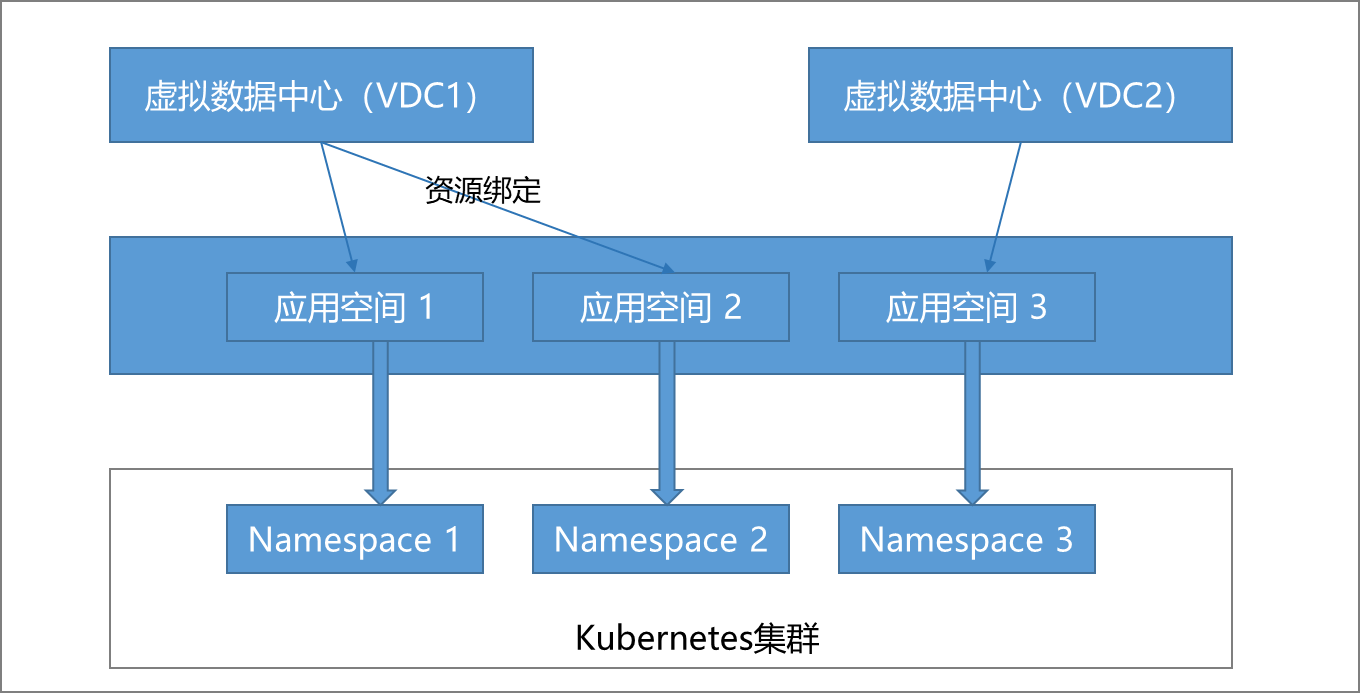
## 应用空间管理

### 功能描述

应用空间管理用于管理Kubernetes底层namespace资源，支持应用空间创建、删除以及绑定虚拟数据中心。支持通过该功能进行底层namespace与上层VDC绑定，可通过此功能进行应用空间之间网络策略设置等。

### 技术原理

应用空间与Kubernetes底层namespace为一对一对应关系，通过kube-apiserver进行namespace创建与删除。



### 约束

无。

## 工作负载管理

### 功能描述

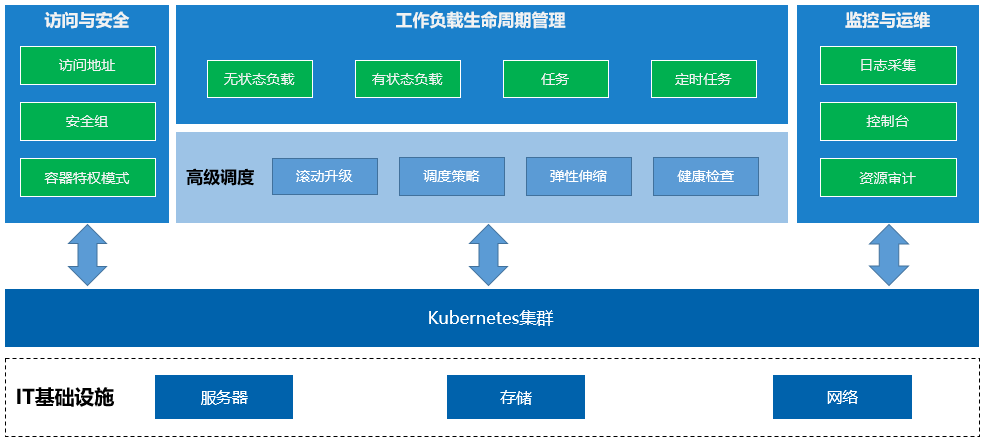
工作负载是Kubernetes的抽象资源，包括无状态负载、有状态负载、任务、定时任务等类型。工作负载管理提供多种工作负载的管理，支撑工作负载的状态展示及变更、升级及回滚、访问地址管理、安全组管理、调度策略设置、日志及控制台、容器组及容器信息查看等功能。

工作负载管理整合容器的状态管理、升级与回滚、访问控制、调度策略、资源监控、日志、控制台等功能，并能够支持可视化设置副本数、自动弹性伸缩、健康检查等特性，能够最大程度的实现应用软件的快速部署与升级、智能监控、自动运维等一系列操作，保障应用软件服务的可持续性、高可用性，提高资源利用率，减少用户在运维管理上的人力与时间成本。

工作负载服务功能包括：

* 无状态负载管理：Kubernetes无状态负载抽象资源管理，支持修改、升级、回滚、开启、关闭、删除、运行状态查看、容器组及容器管理、事件查看、环境变量查看、访问地址管理、安全组管理、调度策略管理、控制台日志及采集日志展示、容器控制台、资源审计、资源使用历史曲线等功能。
* 有状态负载管理：Kubernetes有状态负载抽象资源管理，支持修改、升级、回滚、开启、关闭、删除、运行状态查看、容器组及容器管理、事件查看、环境变量查看、访问地址管理、安全组管理、调度策略管理、控制台日志及采集日志展示、容器控制台、资源审计、资源使用历史曲线等功能。
* 任务：Kubernetes任务抽象资源管理，支持修改、存储卷配置、重新执行、删除、运行状态查看、容器组及容器管理、事件查看、环境变量查看、安全组管理、调度策略管理、控制台日志及采集日志展示、容器控制台、资源审计等功能。
* 定时任务：Kubernetes定时任务抽象资源管理，支持修改、存储卷配置、启用、停用、删除、运行状态查看、任务记录、事件查看、环境变量查看、安全组管理、调度策略管理、控制台日志及采集日志展示、容器控制台、资源审计等功能。
* 自动水平弹性伸缩：云平台提供工作负载水平弹性伸缩能力，能够在工作负载的CPU/内存使用量达到设置好的阈值时自动增加副本数，在使用量过低时自动减少副本数，帮助应用自动应对波峰波谷，保证工作负载高可用的前提下提高资源使用率。
* 多维度健康检查：云平台提供了一套完整的工作负载健康检查及重启机制，保障服务的持续运行，包括进程级的健康检查及业务级健康检查，能够自动定期检查容器进程、业务请求是否正常，并在异常情况下重新启动容器。
* 灵活的调度策略：支持客户根据自身的实际需求为工作负载设置灵活的节点调度策略，包括指定节点、按照节点标签选择、根据与其他工作负载亲和关系选择等，云平台根据设置的策略将容器调度到符合要求的节点上。
* 可控制的特权模式容器：针对某些特殊场景，例如监控类、安全类应用，云平台允许创建特权模式的工作负载。在特权模式下，Docker容器将完全拥有内核的管理权限。但与此同时也会有安全隐患，因此管理员在使用特权模式时，一定要对应用有足够的了解。
* 零停服的升级策略：云平台支持工作负载滚动更新的升级策略，在一次完整的发布过程中，合理地分成多个批次，每次发布一个批次，成功后再发布下一个批次，直到把所有旧的批次全部更新到新版本。在整个滚动过程期间，保证始终有可用的副本在运行，从而平滑的发布新版本，实现零停服(without an outage)、用户零感知。
* 全面的日志采集及展示：云平台支持容器控制台日志及自定义文件日志的采集，并通过直观、便捷的操作支撑日志的展示和查询，帮助运维人员快速查找定位应用异常问题。

### 技术原理



工作负载管理服务提供基于Kubernetes集群的抽象资源管理能力，支撑工作负载的生命周期管理、访问与安全设置、监控与运维，并具备滚动升级、调度策略设置、弹性伸缩、健康检查等自动运维能力。

### 约束

无。

## 网络管理

网络管理服务提供应用或工作负载的访问地址创建、修改、删除、变更内外部访问、设置外部访问方式、配置域名信息等一系列功能，提供应用或工作负载的安全组的创建、配置安全组规则、关联或移除组件等一系列功能。

### 服务

### 功能描述

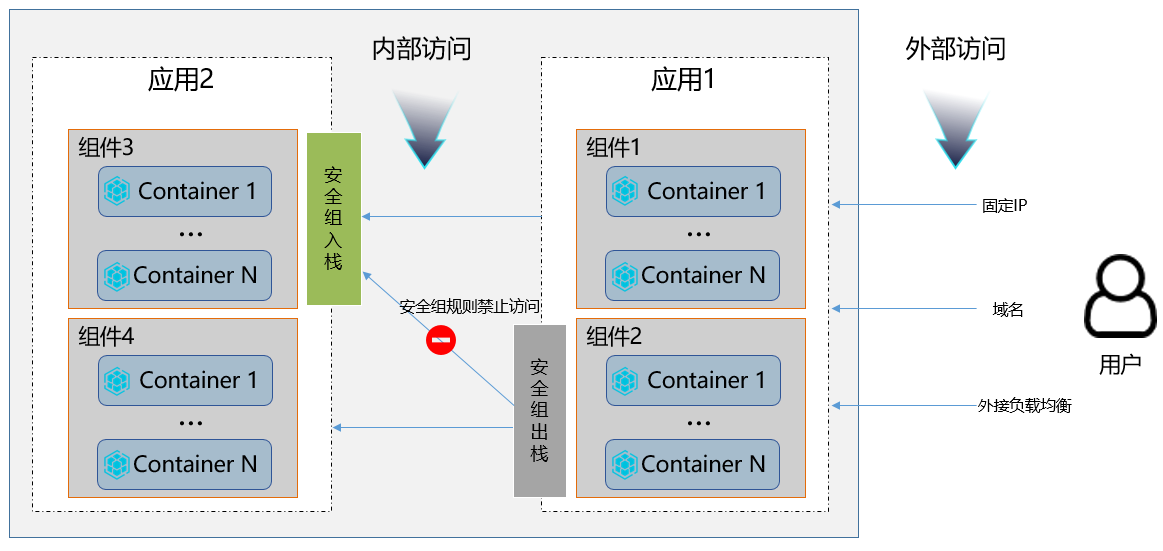
服务是应用组件可以对外访问或提供对外访问以及访问限制等一系列属性的集合，在实际使用场景中对应应用或工作负载的服务与安全组功能。

服务整合多类型访问地址包括内部访问、外部固定IP访问、外接负载均衡访问、域名访问的创建，并支持访问地址的类型转换，同时支持详细地址的增删改，支持实时获取容器端口进行相应的端口映射关系修改，可以配置Http与https类型的域名，能够最大程度减少用户配置应用的访问入口的时间成本。网络管理服务还提供应用加入安全组以实现限制应用访问的功能，安全组规则设置支持方向类型、端口号、授权组件、端口协议的配置，安全组关联组件多个组件批量加入，在保证应用安全的前提下最大程度的节约了用户在运维管理上的人力与时间成本。

服务功能包括：

* 多类型的服务创建：支持创建内部访问类型、外部固定IP访问类型、外接负载均衡访问类型的访问地址，满足客户不同场景下使用不同类型访问地址的需求。
* 域名访问网络调优：域名访问服务支持域名链路调优设置，满足用户不同的网络场景，网络链路不同的需求配置，支持调优的参数有连接超时时间、请求超时时间、响应超时时间和数据传输大小，为用户提供定制化的网络链路服务，充分利用网络资源。
* 服务的创建、修改、删除、修改描述、修改会话亲和。
* 增加、修改、删除详细地址。
* 支持添加、修改、删除域名，并支持七层负载均衡连接超时时间、请求超时时间、响应超时时间、上传大小限制的功能。
* 开启关闭固定IP访问、开启关闭外接负载均衡访问。
* 修改外接负载均衡IP地址。
* HTTP与HTTPS域名配置。

### 技术原理



容器资源池通过多样的访问地址设置及灵活的安全组功能，为用户提供细粒度的访问控制管理。

访问方式包括：

* 内部访问：应用通过Kubernetes service访问应用，该模式下service是ClusterIP类型，通过集群的内部IP暴露服务，服务只能够在集群内部可以访问。
* 对外固定IP访问：应用通过NodePort方式在每个Node上分配一个端口作为外部访问入口，然后通过平台的集群VIP方式对外提供访问。
* 域名形式访问：通过Kubernetes ingress实现基于域名的七层负载均衡访问[Ingress](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#ingress-v1beta1-networking-k8s-io)公开了从集群外部到集群内[服务](https://kubernetes.io/zh/docs/concepts/services-networking/service/)的HTTP和HTTPS路由，可以提供负载均衡、SSL终结和基于域名的虚拟托管。
* 负载均衡IP访问：容器集群外部以设置IP形式暴露，使用固定IP直接访问，并支持为服务设置会话亲和性。

在裸金属上部署的Kubernetes集群，是无法使用LoadBalancer类型的Service的，因为Kubernetes本身没有提供针对裸金属集群的负载均衡器,容器资源池采用MetalLB来实现Kubernetes LoadBalancer类型的Service，用软件的方式实现裸金属模式的负载均衡。

### 约束

无。

## 存储管理

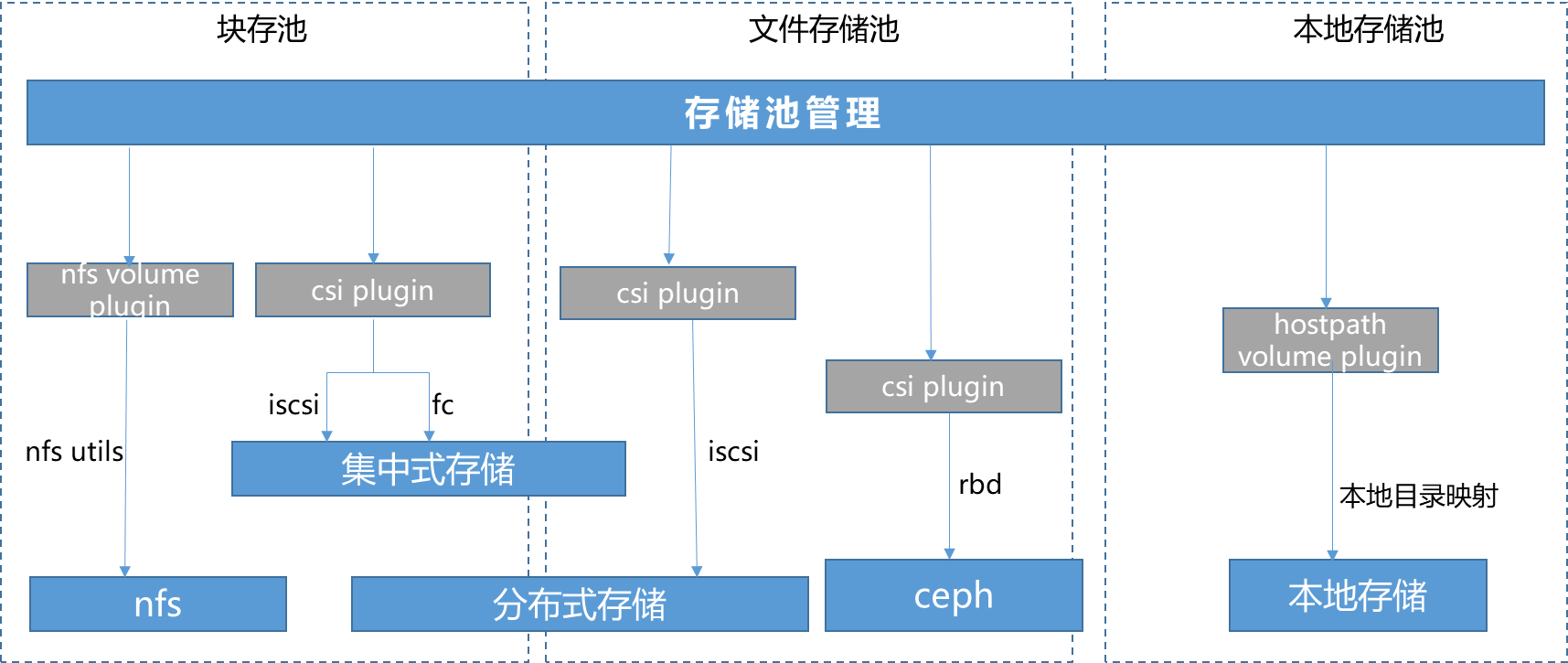
### 多种存储类型对接

### 功能描述

随着企事业单位信息化的发展，信息化系统用户规模不断扩大，不但需要满足用户存储大量文本、图像和视频等要求，还要保证数据的稳定性和安全性，因此对于容器平台纳管多种类型存储要求越来越高。

缘脑平台容器服务存储池功能支持对接多种类型的存储，支持本地存储、分布式存储、集中式存储，目前仅支持本地存储。

### 技术原理



集中式存储，从概念上可以看出来是具有集中性的，也就是整个存储是集中在一个系统中的。但集中式存储并不是一个单独的设备，是集中在一套系统当中的多个设备。

分布式存储系统是大量普通PC服务器通过Internet互联，对外作为一个整体提供存储服务。分布式存储系统具有如下几个特性：

* 可拓展：分布式存储系统可以扩展到几百台甚至几千台的集群规模，随着集群规模的增长，系统整体性能表现为线性增长。
* 低成本：分布式存储系统的自动容错、自动负载均衡机制使其可以构建在普通PC机之上。线性扩展能力也使得增加、减少机器非常方便，可以实现自动运维。
* 高性能：无论是针对整个集群还是单台服务器，都要求分布式存储系统具备高性能。
* 易用：分布式存储系统需要能够提供易用的对外接口，同时要求具备完善的监控、运维工具，并能够方便地与其他系统集成。

缘脑平台容器服务将支持接入的多种类型的存储按照存储提供的形式，分为块存储、文件存储、本地存储。

块存储是一种主流存储类型，主要是基于Block块的模式，缘脑平台容器服务支持通过插件将任意大小的块设备挂载到容器组内，为容器组提供持久化存储空间。目前支持两种主流形态，一种是基于SAN的传统集中式存储，另一种是基于IP的分布式存储，比如浪潮信息AS13000，经过多方位的大规模和性能测试，对上述两种主流形态的支持度很高，同时也支持通过Kubernetes标准插件对接第三方存储。

文件存储也是主流存储类型，提供了可共享访问、弹性扩展、高可靠以及高性能的共享文件存储服务。缘脑平台容器服务支持将文件存储同时挂载给多个容器组，允许同时进行访问避免了大量的数据拷贝与同步成本，达到真正的资源共享。

本地存储做为远端存储系统的一种补充，支持挂载缘脑平台容器集群中服务器文件系统上文件或者目录到容器组中的容器。缘脑平台 容器服务管理的本地存储支持单节点读写访问模式，不支持卷扩容。

### 约束

无。

## 存储卷管理

### 功能描述

存储卷管理提供存储卷的全生命周期管理，支撑存储卷的创建、挂载、卸载、扩容、存储卷共享、卷模板、查看事件、资源审计、监控使用量、使用率、历史曲线等一系列功能。

缘脑平台容器服务提供可视化的存储卷管理功能，能够屏蔽不同厂商、不同类型的存储产品复杂的设置及对接过程。

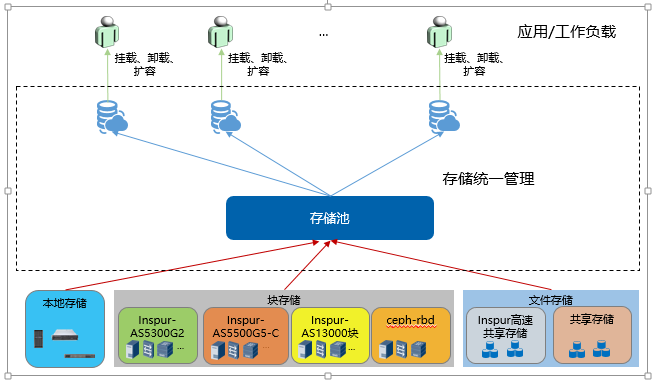
根据存储文件系统的管理方式以及卷的特性，将存储分为块存储、文件存储、本地存储，更加方便用户查找使用。

支持应用挂载、卸载存储卷，以满足不同数据持久化功能。支持存储卷扩容功能，满足不断增加的应用数据需求。支持存储卷使用量、使用率动态的展示，用户直观的了解存储卷基本概况。

存储卷管理功能包括：

* 存储卷生命周期管理：创建、修改、删除、挂载、卸载。
* 存储池生命周期管理：存储池创建、修改、删除。
* 应用信息全方位展示：基本信息、存储卷使用量、存储卷使用率、资源审计、存储池使用量、使用率信息。
* 卷模板：支持使用卷模板为有状态负载应用动态创建存储卷。
* 主动卸载、挂载卷：存储卷对应用主动发起挂载、卸载动作。
* 存储卷在线扩容：支持未挂载、已挂载的存储卷扩容，支持通过修改存储卷容量，实现在不影响业务系统以及持久化数据情况下，快捷的实现存储卷扩容。扩容时业务不中断，数据不丢失，满足高可用业务需求。
* 存储卷使用率、使用量、历史曲线：展示存储卷使用量、使用率，方便用户实时的查询当前存储的使用情况，分析存储使用情况。
* 存储卷共享：支持一个存储卷同时为多个应用提供数据存储服务

### 技术原理



缘脑平台容器集群通过存储类型、存储卷的管理功能，屏蔽底层不同厂商、不同类型的存储设备的对接和使用，为上层工作负载提供卷的挂载、卸载、扩容、使用量监控等功能。

### 约束

* 在同一个虚拟数据中心中，不可存在相同名称的存储卷。
* 存储卷在被应用组件使用的时候，无法被删除，只有未被使用的存储卷才可以被删除。
* 访问模式为单节点读写的存储卷不能挂载给不同的应用组件。
* 应用组件实例数大于1，不能挂载访问模式单节点读写存储卷。
* 应用组件开启弹性伸缩或者升级策略为滚动升级时不能挂载访问模式单节点读写存储卷。

## 配置中心

### 功能描述

配置中心是Kubernetes的两种特殊类型的存储卷的集合，包括普通配置和加密配置，是从集群外部向Pod内部的应用注入配置信息的方式。

普通配置实现向容器中提供配置文件或环境变量来实现不同配置，从而实现了镜像配置与镜像本身解耦，使容器应用做到不依赖于环境配置。

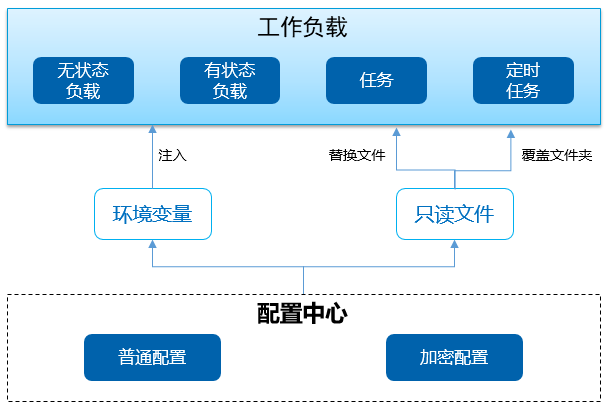
加密配置在普通配置的基础上解决了密码、token、密钥等敏感数据的配置问题。

配置中心为工作负载提供了最基础的资源配置功能，能够最大程度的方便用户添加和修改配置文件，减少用户在运维管理上的人力与时间成本。

配置中心服务功能包括：

* 普通配置：Kubernetes普通配置，支持手动输入添加和导入文件添加，修改和删除等功能。在工作负载中，支持将普通配置的键值对配置到容器环境变量中，以及支持通过替换文件和覆盖文件的方式将普通配置挂载到容器上。
* 加密配置：Kubernetes加密配置，包含两种加密类型：普通类型、TLS（安全传输层协议），支持手动输入添加和导入文件添加，修改和删除等功能。多用于配置密码、Token、私钥等敏感数据，在工作负载中，支持将普通配置的键值对配置到容器环境变量中，以及支持通过替换文件和覆盖文件的方式将普通配置挂载到容器上。
* 支持用户手动输入或者文件导入的方式，来定义配置信息。
* 支持将配置信息中想要使用的键值对，通过覆盖文件或替换文件的方式添加到容器的挂载卷上。
* 通过覆盖文件添加的挂载卷支持热加载功能，无需重启工作负载，即可生效。

### 技术原理



配置中心服务提供基于Kubernetes集群的基础配置功能，配置可以通过环境变量的方式注入到工作负载的容器中，也可以通过替换文件或覆盖文件夹的形式挂载到容器上。

### 约束

无。

## 运营服务

### 统一用户认证

#### 功能描述

缘脑平台中引入了IAM（Identify and Access Management，身份识别与访问管理系统）体系，使用标准化的认证体系替换自研体系。

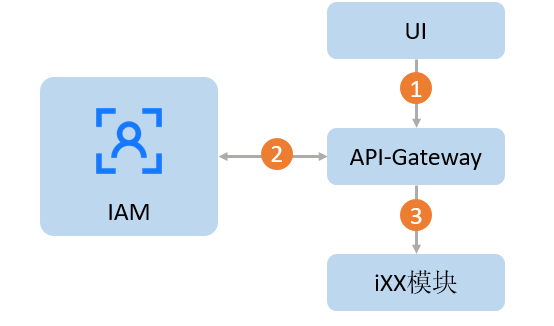
#### 技术原理

缘脑平台 中的统一用户认证体系基于开源的KeyCloak。KeyCloak依托于Redhat，是RedHat SSO的上游项目。作为一套开源的、独立的认证授权服务器，KeyCloak主要基于OAuth2.0协议和JWT（Json Web Token）规范实现，提供了多种语言的集成，例如Louketo、Spring Boot插件、NodeJS插件等，不同语言都可以通过插件适配KeyCloak，从而提供了快速集成、快速适配上线的能力，便于缘脑平台或其他系统对接接入。

缘脑平台基于KeyCloak提供的原生的功能进行了二次开发和扩展，如下



目前基于KeyCloak的IAM体系已经完全融入缘脑平台 中，取代了原有的用户体系，其交互流程如下：



1. UI访问后台需要经过API网关；
2. 访问API网关时，API网关通过IAM验证认证信息是否正确，如果正确，则继续；
3. 请求路由到iXX模块，完成具体的功能执行。

#### 约束

无

### 访问控制

#### 功能描述

随着企业规模的不断扩大，企业使用系统的人员越来越多，所需要的权限多样性也在上升，需要灵活、直观的角色权限访问控制系统。

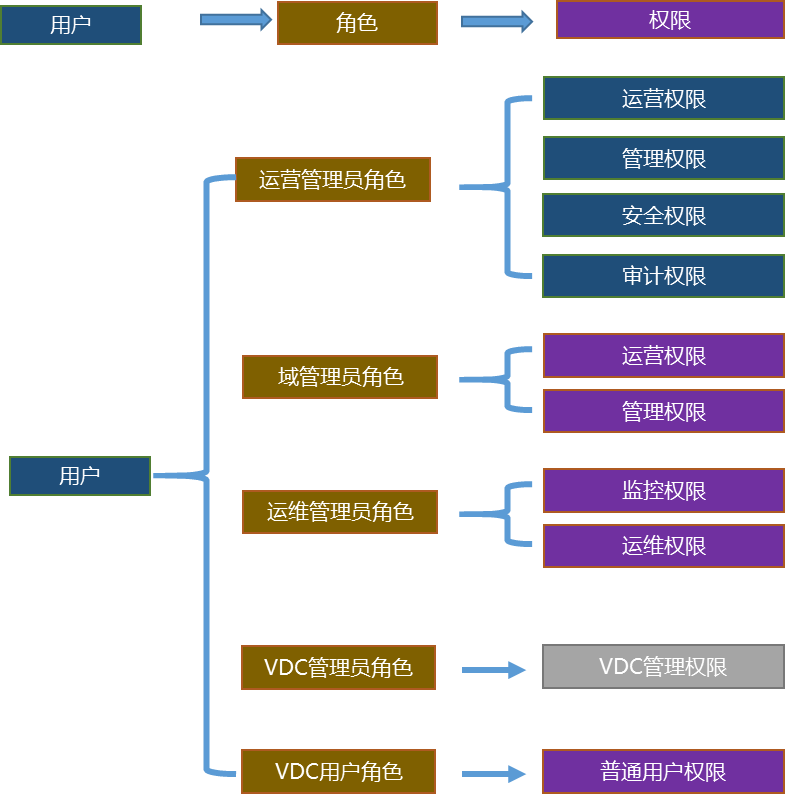
平台站在用户的角度，基于RBAC，为客户提供了可自定义的角色权限访问控制功能，通过角色自定义功能，我们可以创建具有不同权限的角色，并赋予系统中不同的用户，从而实现对于系统资源访问的有效控制，提升了系统的安全性和可用性。

#### 技术原理

自定义角色是以系统预置角色为基础的权限集合，通过权限分配“树”的自定义勾选，可实现所有可选权限的有效控制和细粒度的分配。同时，也实现了权限和角色多对多的控制和操作。



系统中有五种角色类型，分别为运营管理员、域管理员、虚拟数据中心管理员、虚拟数据中心用户。每一种角色有一个内置角色。内置角色具有该角色类型所有的权限。自定义角色时，选择角色类型，然后通过勾选权限树，就可以实现自定义角色功能。将角色赋予实际用户，该用户就具备了此角色的权限。



#### 约束

无。

### 双因子认证

#### 功能描述

双因子认证服务，是一种基于证书安全认证技术，除了常规的用户名和密码的认证之后，增加了一种基于证书的身份认证方式，该方式的认证使得系统的安全性更高。当系统开启了双因子认证之后，即使用户的用户名密码被盗取了，没有证书仍然无法登录系统，这样将大大提高系统的安全性。

用户若要访问系统，应该首先获取证书。系统管理员作为系统的证书管理员，需要线下给用户颁发证书，该证书由用户自行妥善保管。颁发的证书形式包含两种：一种是电子版的数字证书，管理员线下给到用户后，用户需要自行保存，并安装到客户端；另一种是Ukey形式，管理员通过Ukey厂商提供的工具将证书写入到Ukey中，交由用户使用，用户妥善保管Ukey。后续用户访问系统时，浏览器自动读取客户端的数字证书（不区分是否装在Ukey中）用于登录系统。

#### 技术原理

双因子认证功能，主要包括证书管理和证书认证。证书管理主要是负责证书的颁发、删除，证书认证是负值对证书的可信认证。



1. 系统管理员线下颁发证书给对应的用户；
2. 用户访问云平台时，浏览器自动携带证书访问Ingress；
3. Ingress进行证书的双向认证，验证证书可信性；
4. 证书认证成功，客户端与服务端建立安全通信，否则请求被拒绝；

#### 约束

* 使用该功能的前提条件是当前系统开启了双因子认证功能。
* 当前系统开启了双因子认证服务之后，所有用户都需要安装证书之后才能登录。
* 部分版本不支持该功能。

# 缘脑平台安全可靠设计

## 多维度安全体系设计

* 容器安全
  + 容器安全上下文：支持容器安全上下文配置，包括是否以特权模式运行、文件系统读写权限、运行容器进程的用户/用户组；
  + 容器镜像保护：提供镜像数字签名验证功能，确保容器镜像未被篡改。支持上传时启用镜像签名功能，可在镜像详情查看开启状态，在应用容器启动时验证签名信息；
  + 容器镜像漏洞扫描：基于漏洞数据库对镜像版本进行安全扫描，用户可以快速的扫描容器镜像版本，以及查看镜像存在的漏洞详情。
* 安全管理中心
* 身份认证：支持本地用户认证和域用户认证两类认证方式，支持双因子认证；
* 权限管理：支持基于角色的访问控制模型（RBAC），将管理员按职责划分成不同角色并赋予最小权限，并支持用户自定义角色以及权限分域功能；
* 账号密码保护：采用安全通道和密码技术实现用户账号密码信息的安全传输和加密存储，支持密码复杂度校验、登录失败锁定、会话管理等多种密码安全策略；
* 系统日志审计：支持系统操作日志与运行日志的记录和查看，仅允许授权用户查看和管理系统日志，避免产生抵赖风险；
* 黑白名单：支持配置访问系统的IP地址黑名单和白名单，限制非法访问的IP地址和地址段。

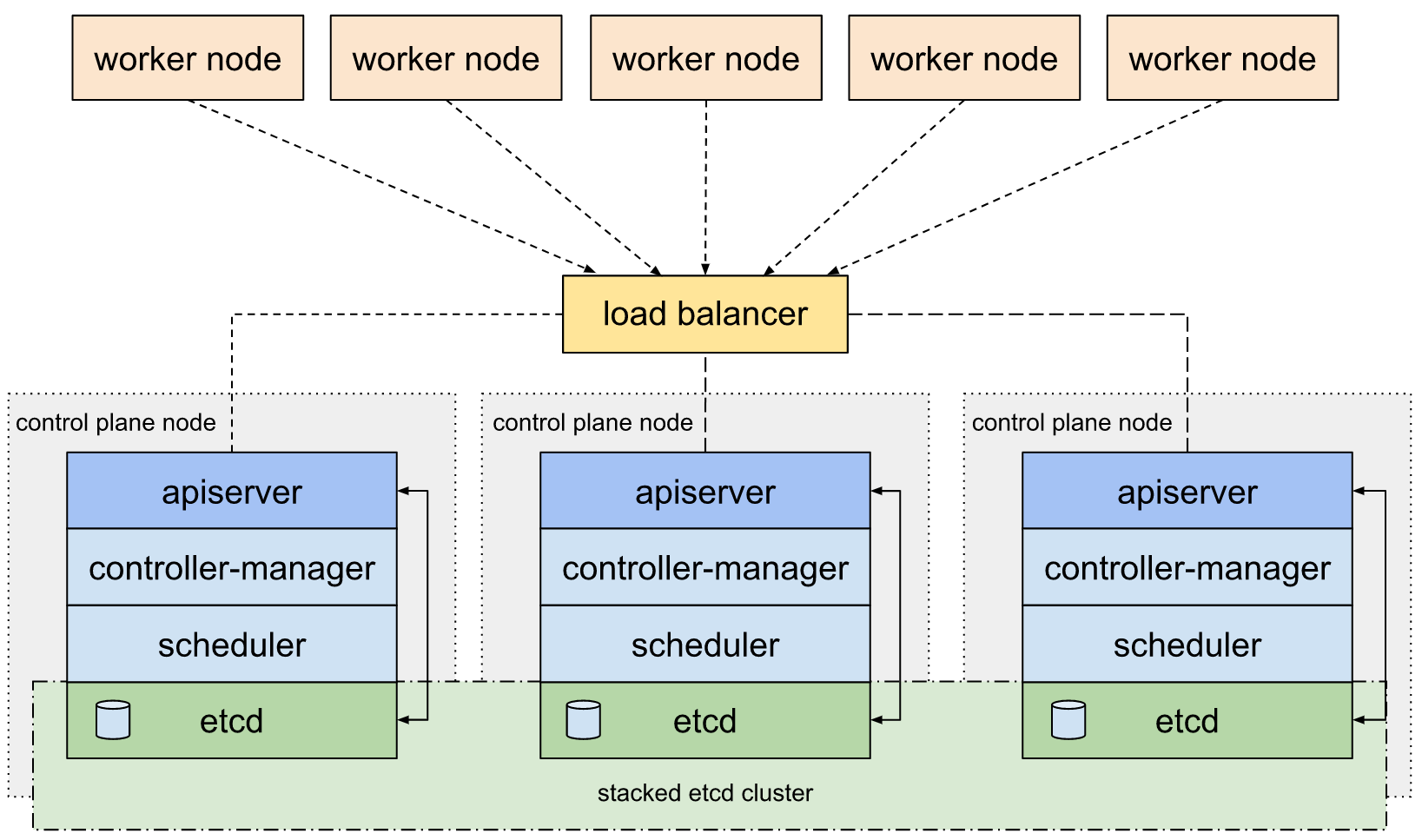
## 可靠性设计

缘脑平台包括容器资源池和云管理平台两大组件，容器资源池是企业级的K8S版本，云管理平台都是以容器的supervisor K8S版本为底座进行部署，所以缘脑平台的可靠性设计是继承延续了K8S的可靠性设计。

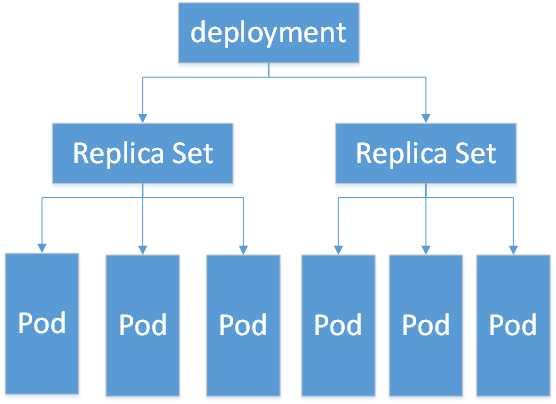
缘脑平台是基于K8S的实现版本，该版本中所有组件都实现了pod化，由K8S进行pod组件的生命周期管理与调度，每个组件都具备K8S的特性，总是维持在环境中定义的正常运行的副本机制，支持水平快速扩展，支持故障转移/重启，具有健康检查机制，确保每个组件都能正常对外提供服务。

### 容器资源池可靠性

容器资源池集群采用副本机制维护集群组件的高可用，控制平面至少由3台服务器组成，其上运行容器资源池的控制组件，避免单点组件，架构如下图所示。



* loadblancer高可用：使用haproxy实现，分别运行在3台服务器上组成主备集群，通过keepalived组件提供vip，vip所在的节点为api的请求入口，请求被负载到后端状态正常的apiserver组件上。当所在节点的haproxy出现问题时，keepalived将把vip切换到其他节点。控制平面节点放置在haproxy TCP后面接受负载请求。负载均衡器将流量分配给目标列表中所有运行状况良好的控制平面节点。
* kube-apiserver高可用：在Node节点上使用代理对多个Master做负载均衡。
* kube-controller-manager与kube-scheduler高可用：这两项服务是Master节点的一部分，他们的高可用相对容易，仅需要运行多份实例即可。这些实例会通过向apiserver中的Endpoint加锁的方式来进行leader election， 当目前拿到leader的实例无法正常工作时，别的实例会拿到锁，变为新的leader。
* etcd高可用：使用独立的etcd集群，使用3台或者5台服务器只运行etcd，独立维护和升级。甚至可以使用CoreOS的update-engine和locksmith，让服务器完全自主的完成升级。这个etcd集群将作为基石用于构建整个集群。采用这项策略的主要动机是etcd集群的节点增减都需要显式的通知集群，保证etcd集群节点稳定可以更方便的用程序完成集群滚动升级，减轻维护负担。
* Worker node高可靠：Worker node是容器集群中为pod提供运行环境的资源池，其上的pod都由控制平面控制，控制平面中etcd维护者所有容器资源的元数据、状态等信息，controller-manager通过list/watch机制查看etcd里存储的数据来决定pod的生命周期与调度。当某个worker node出现异常状况后，其上的pod资源会得到合理的处理，始终维持着组件可靠运行所需的pod个数。因为虚拟化资源池与云管理层都以workload pod的形式部署在容器之上，所以虚拟化资源池与云管理层组件的可靠性受益于成熟的容器高可靠设计，加之虚拟化资源池、云管理层自身的高可靠设计，使得虚拟化资源池、云管理层的运行更加稳定可靠。
* 业务POD高可靠：使用Deployment来创建Replica Set。Replica Set在后台创建pod。确保Kubernetes中有指定数量的pod在运行。如果少于指定数量的pod，会创建新的pod，反之则会删除掉多余的以保证pod数量不变。



* pod健康检测：当pod不健康，运行出错或者无法提供服务时，由于某种原因挂掉或崩溃。ReplicaSet会立刻用这个Pod的模版新启一个Pod来替代它。
* 弹性伸缩：在业务高峰或者低峰期的时候，可以动态的调整pod的数量来提高资源的利用率。平台定时自动监控平台pod的整体资源使用情况，并基于预设的策略做到自动伸缩pod数量。
* 滚动升级：滚动升级为一种平滑的升级方式，通过逐步替换的策略，保证整体系统的稳定，在初始化升级的时候就可以及时发现和解决问题，避免问题不断扩大。
* 回滚：当升级pod镜像或者相关参数的时候发现问题，可以回滚，回滚到上一个稳定的版本或者指定的版本。每一次对Deployment的操作，都能保存下来，给予后续的回滚使用。

### 云管理层可靠性

#### 总体介绍

云平台底层基于计算存储融合的安全可靠服务器以及网络交换设备的部署，通过虚拟化技术整合，实现计算资源池、存储资源池和网络资源池构建。上层通过云资源管理平台建设实现云资源管理、系统管理、安全管理、智能监控、自动化运维等功能。通过云平台提供的虚拟化以支撑上层业务系统相关模块的部署和运行。

云平台的功能模块高可用设计包括下面几个方面：

* 数据库高可用：通过组建数据库高可用集群实现了数据库的高可用，当单台数据库服务器出现硬件故障时，另外的备用节点会承接数据库服务，保证了业务连续性和数据安全性；
* 数据库分库分表：通过将不同业务使用的数据库进行分库和分表操作，即减轻了单个数据库的压力，同时也保证了在单一数据库出现故障时，将影响范围降到最小；
* 服务高可用：将每个服务安装到多台物理服务器里，上层通过负载均衡进行请求分担，既可以减轻单台服务器的压力，又可以在单台服务器出现故障时还有备用节点可以承担相应的服务，实现了服务的高可用；
* 消息队列高可用：采用分布式消息队列，防止在消息量过大时发成消息拥塞的情况，提高了消息队列的稳定性，实现了消息队列的高可用功能；
* 故障监控：监控系统组件的进程健康状态，当出现组件服务异常时，可以按照之前既定的策略进行自维护和告警，包括服务自动重启，服务器故障域迁移等操作；
* 状态恢复：重启故障的系统组件，快速恢复系统正常运行。

通过整体高可用设计，可以使云平台从如下方面保证平台的稳定、提高业务连续性：

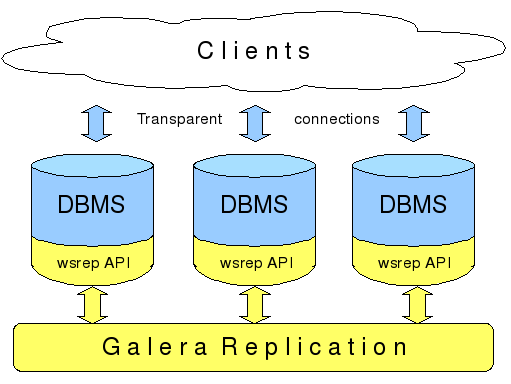
* 数据库无单点；
* API服务无单点；
* 消息队列无单点；
* 保障了整个平台无单点故障；
* 管理进程监控和故障自动恢复；
* 提高了系统的整体可靠性，降低了系统维护复杂度。

#### 数据库高可靠

##### 服务介绍

使用mariadb作为后端数据存储的数据库，使用galera实现mariadb数据库的复制集群。Galera Cluster是一个高可用集群解决方案，它不是一个分布式数据库，没有进行分库、分表的操作。

##### 产品架构



数据库高可用

##### 功能特性

* 同步复制
* 双活多主
* 可对集群中任一节点进行数据读写
* 自动成员控制，故障节点自动从集群中移除
* 自动节点加入
* 真正并行的复制，基于行级
* 直接客户端连接，原生的 MySQL 接口

#### 消息队列高可靠

##### 服务介绍

通过RabbitMQ内置集群功能实现消息队列的高可用。Rabbitmq内置有Cluster集群功能，同一个Cluster的节点会共享topic、exchange、binding和queue等元信息，但是对于真正的queue消息数据是要依赖于Mirror Queue机制来实现消息的HA的，而且组成Cluster建议至少3个节点，否则网络分区发生的时候也不好做决策。所以采用Cluster+Mirror Queue的Rabbitmq高可用方案。

##### 产品架构



消息队列高可用架构设计

##### 功能特性

在镜像集群模式下，创建的queue，无论元数据还是queue里的消息都会存在于多个实例上，就是说，每个RabbitMQ节点都有这个queue的一个完整镜像，包含queue的全部数据的意思。然后每次你写消息到queue的时候，都会自动把消息同步到多个实例的queue上。

在RabbitMQ后台新增了一个镜像集群模式的策略，指定的时候是可以要求数据同步到所有节点的，也可以要求同步到指定数量的节点，再次创建queue的时候，应用这个策略，就会自动将数据同步到其他的节点上去了。

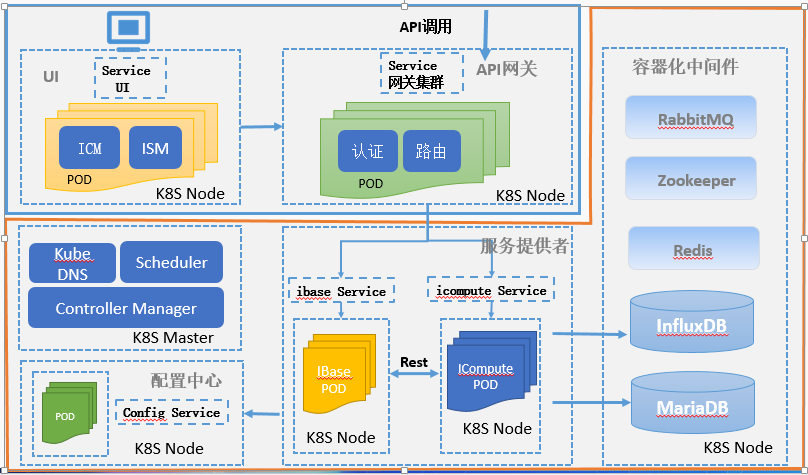
#### 微服务组件高可靠

##### 服务介绍

平台采用微服务架构的设计理念，基于K8S+SpringBoot、springcloud的架构模式，每个服务节点可分布式独立部署。平台微服务架构可实现多服务节点间的服务的负载均衡、服务路由、集群容错等机制，使系统具有很强的连通性，健壮性，伸缩性和扩展性。

平台采用全方位的高可用设计，平台中的每个微服务都可以多副本部署，在一个节点因为故障停止服务时，请求会自动分配到其他可用节点，并且K8S会维护节点的副本数，保证系统中每个微服务的可用性。系统内置健康检查机制，定时扫描各个服务是否健康，一旦某个服务鉴定为非健康状态，那么该服务会重启部署，保证节点的高可用。

##### 产品架构



微服务产品架构

##### 功能特性

* **多副本：**系统的各个组件以pod集群的形式运行在K8S容器环境下，K8S可以配置每个pod的副本数量，当某个pod由于某种原因宕掉时，K8S会自动重启一个新的pod，保持正在运行的pod数量和其副本数量一致。
* **VIP：**K8S的Service VIP机制为同一个组件的pod提供了统一的对外访问IP，实现pod服务集群内的访问负载均衡。
* **服务熔断：**Hystrix实现熔断器确保某个服务故障时系统依然可用。
* **服务间调用：**各模块服务独立部署于各自pod集群，模块服务间用feign实现内部调用，UI与网关、网关与各个服务模块之间通过REST API的方式调用。