《云原生入门级开发者认证》学习笔记之云原生架构总览

原创

山河已无恙 已于 2022-04-02 20:49:04 修改 770 收藏

分类专栏： CKA&云原生相关笔记 文章标签： 云原生

于 2022-04-02 20:46:34 首次发布

本文链接：https://blog.csdn.net/sanhewuyang/article/details/123928649

版权

写在前面

- 嗯，报了华为云的一个《云原生入门级开发者认证》，这里整理课堂笔记记忆，感兴趣小伙伴也可去试试
- 学习的原因：
  - 虽然考了CKA,了解了一些K8s相关的知识，但是对云原生一直都很模糊
  - 希望对云原生有一个基本的认识,云原生入门
  - 博文主要是课堂笔记
  - 转载需要写URL，所以改正了原创

傍晚时分，你坐在屋檐下，看着天慢慢地黑下去，心里寂寞而凄凉，感到自己的生命被剥夺了。当时我是个年轻人，但我害怕这样生活下去，衰老下去。在我看来，这是比死亡更可怕的事。--------王小波

一 云原生架构总览

云原生技术的发展

- 2001年，VMware发布了第一个针对x86服务器的虚拟化产品ESX和GSX，即ESX-i的前身。
- 2006年10月，以色列的创业公司Qumranet在完成了虚拟化Hypervisor基本功能、动态迁移以及主要的性能优化之后，正式对外宣布了KVM的诞生。2009年4月，VMware推出业界首款云操作系统VMware vSphere。
- 2006年，AWS推出首批云产品Simple Storage Service (S3)和Elastic Compute Cloud(EC2)，使企业可以利用AWS的基础设施构建自己的应用程序。
- 2010年7月，Rackspace Hosting和NASA联合推出了一项名为OpenStack的开源云软件计划。
- 2011年，Pivotal推出了开源版PaaS Cloud Foundry，作为Heroku PaaS的开源替代品，并于2014年底推出了Cloud Foundry Foundation。
- 2008年，LXC（Linux Container）容器发布，这是一种内核虚拟化技术，可以提供轻量级的虚拟化，以便隔离进程和资源。LXC是Docker最初使用的具体内核功能实现。
- 2013年，Docker发布，组合LXC，Union File System和cgroups等Linux技术创建容器化标准，docker风靡一时，container逐步替代VM，云计算进入容器时代。
- 2015年7月，Google联合Linux基金会成立了CNCF组织，kubernetes成为CNCF管理的首个开源项目。
- 2018年3月，Kubernetes从CNCF毕业，成为CNCF第一个毕业项目。

云原生的定义

> 云原生定义-Pivotal早期观点

Pivotal公司的Matt Stine于2013年首次提出云原生的概念,并推出了Pivotal CloudFoundry和Spring系列开发框架,是云原生的探路者。

2015年,云原生刚推广时,Matt Stine在《迁移到云原生架构》一书中定义了符合云原生架构的几个特征

- 符合12因素应用(12 Factors Application)
- 面向微服务架构(Microservices)
- 自服务敏捷集成设施(Self Service Agile Infrastructure)
- 基于API的协作(API-Based Collaboration)
- 抗脆弱性(Antifragility)

> 云原生定义-Pivotal当前论述

Pivotal官方网站对云原生最新论述如下:

- 云原生是一种构建和运行应用程序的方法,它利用了云计算交付模型的优势;
- 云原生关注如何创建和部署应用程序,而不是在何处(云计算);
- 虽然现在公有云影响了几乎每个行业的基础设施投资思想,但类似云的交付模式并不仅限于公有云环境,它适用于公有云和私有云;
- 云原生结合了DevOps、持续交付、微服务和容器的概念;
- 当公司以云原生方式构建和运营应用程序时,它们可以更快地将新想法推向市场并更快地响应客户需求;

2019年，Pivotal被vmware收购，成为其子公司。

> 云原生定义-`CNCF早期观点`

`云原生计算基金会`(以下简称`CNCF`)是一个开源软件基金会,成立于2015年7月,隶属于`Linux基金会`。致力于云原生(Cloud Native)技术的普及和可持续发展。。CNCF是GitHub上许多增长最快的项目的提供者的中立家园，其中包括`Kubernetes`，`Prometheus`和`Envoy`等

起初,`CNCF`对`云原生`的定义包含以下三个方面:

- `应用容器化` (Software stack to be Containerized)
- `面向微服务架构` (Microservices Oriented)
- `应用支持容器的编排调度` (Dynamically Orchestrated)

到2018年,随着社区对`云原生`理念的广泛认可和云原生生态的不断扩大,还有CNCF项目和会员的大量增加,起初的定义已经不再适用。

> 云原生定义-`CNCF当前定义`.(2018年更新后的定义论述如下:)

云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中,构建和运行可弹性扩展的应用。云原生的代表技术包括`容器`、`服务网格`、`微服务`、`不可变基础设施`和`声明式API`。

可变服务器基础架构:服务器会不断更新和修改。使用此类基础架构的工程师和管理员可以通过SSH连接到他们的服务器,手动升级或降级软件包,逐个服务器地调整配置文件,以及将新代码直接部署到现有服务器上。换句话说,这些服务器是可变的;它们可以在创建后进行更改。由可变服务器组成的基础设施本身可称为可变传统或(贬低)手工艺。

不可变基础架构:其中服务器在部署后永远不会被修改。如果需要以任何方式更新,修复或修改某些内容,则会根据具有相应更改的公共映像构建新服务器以替换旧服务器。经过验证后,它们就会投入使用,而旧的则会退役.

不可变基础架构的好处:基础架构中更高的一致性和可靠性,以及更简单,更可预测的部署过程。它可以缓解或完全防止可变基础架构中常见的问题,例如配置漂移和雪花服务器。但是,有效地使用它通常包括全面的部署自动化,云计算环境中的快速服务器配置,以及处理状态或短暂数据(如日志)的解决方案

云原生核心理念

总得来说有以下几大核心理念:

- 利用 容器 和 服务网格 等技术,解耦 软件开发 ,提高了 业务开发部署 的灵活性和易维护性
- 以 Kubernetes 为核心的多层次、丰富的开源软件栈,被各大厂商支持,用户选择多,避免厂商绑定
- 以 Kubernetes 为核心的松耦合平台架构,易扩展,避免侵入式定制Kubernetes已被公认是platform for platform
- 中心式编排 ,对应用和微服务进行统一的动态管理和调度,提高工作效率和资源利用率

云原生的技术版图

底层:主要是在运行容器化服务之前,需要为容器准备标准化的基础环境,比如用于 自动化部署和配置容器运行平台和环境 ,代表性工具和厂商包括 Ansible 、 Chef 、 Puppe , OpenStack 等。容器镜像库。凭据管理主要用于在整个容器平台中进行秘钥管

Runtime(运行时) :容器的整个运行环境,是云原生底层技术中最核心的部分它包括了 运行时 、 存储 、 网络 三大块, Runtime 提供容器的运行环境(Docker)。 存储 要解决容数据器持久化的问题。 网络 也是非常核心且非常复杂多变的一块内容,常见方案有 Calico 、 Flannel , open Switch 等。

编排调度层,主要负责容器平台的编排和调度,包括 服务的发现和治理 , 远程调用服务代理 , 微服务治理 等组件。

最上层 就是跟应用定义与开发相关的内容,主要是一些技术或工具来支撑。

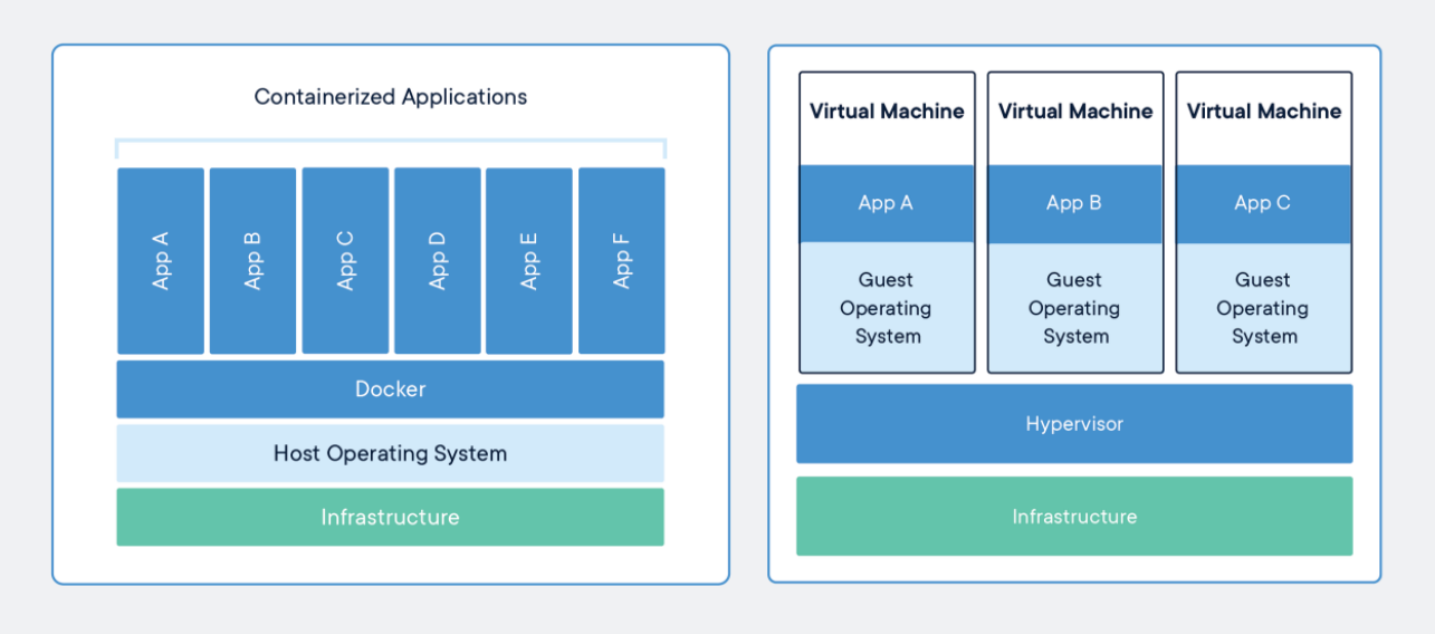
横向上 云原生还包括众多的经过认证的平台供应商。应用运维层,包含了大量用于对平台进行监控(Prometheus-Nagios - Grafana 、 Zabbix 等)、日志(Fluentd , ElasticSearch , Logstash)、以及追踪(Jaeger)的工具。最后还有一块是关于无服务器架构 serverless 的内容

容器技术-提高应用可移植性,提升业务敏捷

容器可以将应用本身及其依赖打包,使得应用可以实现“ 一次封装,到处运行 ”。容器也可以理解成一种沙盒技术, 沙盒 在计算机安全领域中是一种 安全机制 ,为运行中的程序提供的隔离环境。

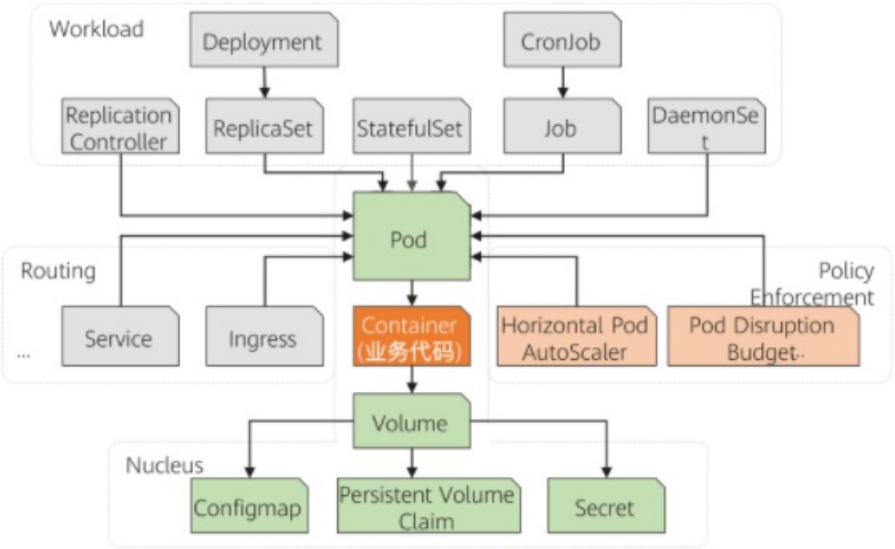
容器核心价值可移植性:

- 环境标准化,应用随处运行敏捷:
- 创建速度快,秒级资源弹性提高生产力:
- 消除跨服务依赖性和冲突



主流的容器技术,如`Docker`,它是通过`内核虚拟化技术(namespace以及cgroups等)`来提供容器的资源隔离与安全保障。由于Docker通过操作系统层的虚拟化实现隔离,所以`Docker容器`在运行时,不需要类似虚拟机额外的操作系统开销,提高资源利用率。同时,Docker能够帮助你快速地测试、快速地编码、快速地交付,并且缩短从编码到运行应用的周期,从而使得企业实现业务敏捷。

Kubernetes的声明式API-面向开发者提供全新分布式原语



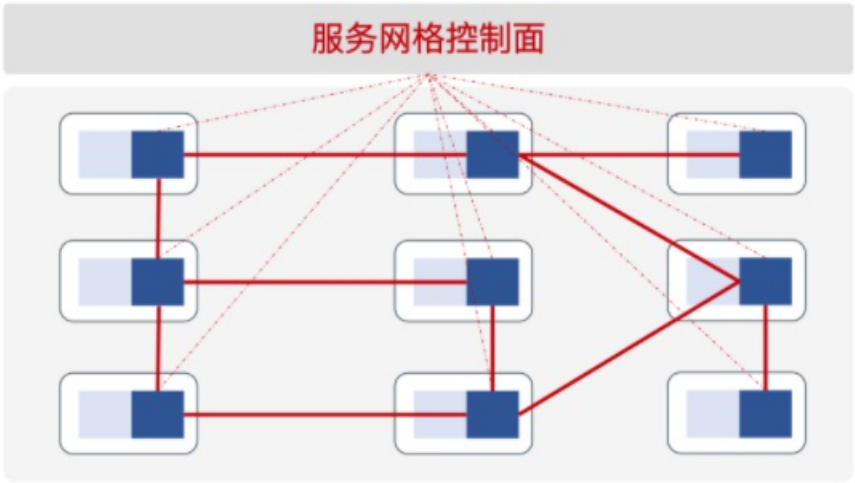
针对期望状态结果给出声明,而不是过程

对于我们使用`Kubernetes API`对象的方式,一般会编写对应`API对象`的`YAML`文件交给`Kubernetes`(而不是使用一些命令来直接操作API)。

所谓“`声明式`”,指的就是我只需要提交一个定义好的`API`对象来“声明”(这个YAML文件其实就是一种“声明”)表示所期望的最终状态是什么样子就可以了。而如果提交的是一个个命令,去指导怎么一步一步达到期望状态,这就是“命令式”了。可以说,声明式API是Kubernetes项目编排能力“赖以生存”的核心所在。

服务网格-剥离业务代码和分布式框架

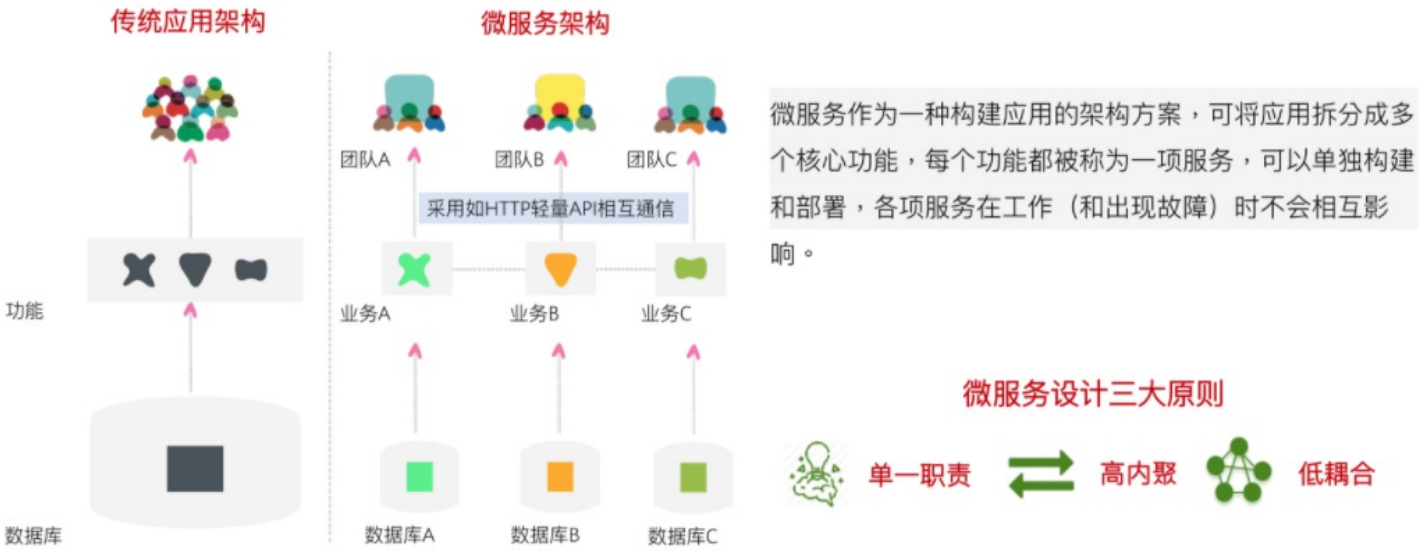
- 非侵入式接管应用服务通信
- 细粒度流量治理:灰度发布、故障注入、可观测性支持
- 平台团队聚焦框架层的开发和调优
- 业务团队聚焦业务本身的开发



- `Service Mesh` -词最早由开发 Linkerd 的Buoyant公司提出,并于2016年9月29日第一次公开使用了这一术语
- 服务网格通过非侵入式的方式接管应用的服务通信。对于每个业务单元/模块来说他们甚至不需要对网络通信、负载均衡等有任何的感知,。
- 服务网格提供细粒度流量治理,包括灰度发布、故障注入、可观测性支持等能力,挺高了业务应用的易维护性。
- 对于企业开发者来说,服务网格可以很好地帮助他们`剥离业务代码和分布式框架`。

微服务-加速企业应用架构升级

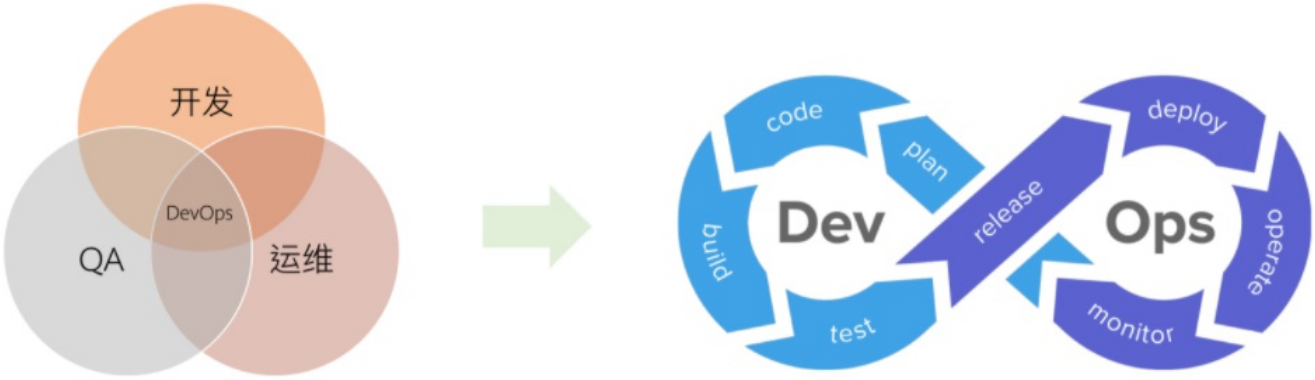
在`CNCF`的定义中,`微服务`也是作为一种代表性的技术,而实际上,`微服务更侧重于描述软件架构`,这种软件架构相比单体架构,更加能够发挥云原生相关的技术优势



微服务是一种用于构建应用的架构方案,它是松散耦合的分布式架构框架,因此一个团队的更改不会破坏整个应用。使用微服务的好处是,开发团队能够快速构建应用的新组件,以满足不断变化的业务需求。

微服务架构有别于更为传统的单体式方案,可将应用拆分成多个核心功能。每个功能都被称为一项服务,可以单独构建和部署,这意味着各项服务在工作(和出现故障)时不会相互影响。比如你在线购物时,使用搜索栏来找产品,这个搜索功能就是一项服务,同时你也看到了相关产品推荐,这些推荐也是来自于另外一项服务,还有购物车等,都是一项一项的服务

DevOps -促进开发运维一体化



DevOps=开发(Development) +运维(Operations) ,是打通开发与运维之间的壁垒,促进开发、运营和质量保障(QA)等部门之间的沟通协作,以便对产品进行小规模、快速选代式地开发和部署,快速响应客户的需求变化。它强调的是开发运维一体化,加强团队间的沟通和快速反馈,达到快速交付产品和提高交付质量的目的。

云原生应用

> 云原生应用程序是 专为云模型构建 的。这些应用程序由小型专用功能团队快速构建和部署到一个平台,可提供轻松的 横向扩展和硬件解耦 -为组织提供跨云环境的更高灵活性,弹性和可移植性。”- Pivotal

> 云原生应用是独立的 小规模松散耦合服务的集 合,旨在提供备受认可的业务价值,例如快速融合用户反馈以实现持续改进。简而言之,通过云原生应用开发,可以加速构建新应用,优化现有应用并将这些应用全部组合在一起。其目标是以企业需要的速度满足应用用户的需求。”
> - RedHat

云原生应用理解

- 基于云原生 的相关技术,设计运行在云上的,充分发挥云优势的应用。
- 一般采用 容器 的打包、分发、部署的形式,应用内(间)采用 微服务 的架构,充分利用云提供的组件服务,采用 DevOps 的组织架构和方法,通过 CI/CD工具链 ,实现产品和服务的持续交付。

传统应用与云原生应用的区别

云原生应用	传统应用
可预测。 云原生应用符合旨在通过可预测行为最大限度提高弹性的框架或“合同”	不可预测。 通常构建时间更长，大批量发布，只能逐渐扩展，并且会发生更多的单点故障
操作系统抽象化	依赖操作系统
资源调度有弹性	资源冗余较多，缺乏扩展能力
团队借助DevOps更容易达成协作	部门墙导致团队彼此孤立
敏捷开发	瀑布式开发
微服务各自独立，高内聚，低耦合	单体服务耦合严重
自动化运维能力	手动运维
快速恢复	恢复缓慢

云原生应用的12-Factors要素

`Heroku`于2012年提出`12因素`,告诉开发者如何利用云平台提供的便利来开发更具可靠性和扩展性、更加易于维护的云原生应用。

第一基准代码。一份代码库对应多份部署,所有部署的基准代码相同,但每份部署可以使用不同的版本

第二,依赖。显式声明依赖关系,通过依赖清单确切的声明所有依赖项,这一做法会统一应用到生产和开发环境。

第三,配置。在环境中存储配置,推荐将应用的配置存储于环境变量中,环境变量可以非常方便地在不同的部署间做修改,却不动一行代码。与配置文件不同,不小心把它们迁入代码库的概率微乎其微,与一些传统的解决配置问题的机制,比如Java的属性配置文件相比,环境变量、语言和统计无关。

第四,后端服务。把后端服务当作附加资源,每个不同的后端服务是一份资源,例如个mysql数据库是一个资源,两个mysql数据库被当做两个不同的资源,云原生应用将这些数据库都视作附加资源,这些资源和他们附属的部署保持松耦合。

第五,构建发布运行云原生应用,需严格区分构建、发布、运行这三个步骤。举例来说,直接修改处于运行状态的代码是非常不可取的做法,因为这些修改很难再同步回构建步骤

第六,进程。以一个或多个无状态进程运行应用,在运行环境中,应用程序通常是以个或多个进程运行的

第七,端口绑定。通过端口绑定来提供服务。

第八,并发。通过进程模型进行扩展,在12-factor应用中,进程是一等公民。12Factor应用的进程主要借鉴于unix守护进程模型。开发人员可以运用这个模型去设计应用架构,将不同的工作分配给不同的进程类型。例如,HTTP请求可以交给web进程来处理,而常驻的后台工作则交由worker进程负责

第九,易处理。快速启动和优雅终止和最大化健壮性,这有利于快速弹性的伸缩应用迅速部署变化的代码或配置文件的部署应用。

第十,开发环境与线上环境等价,尽可能的保持开发预发布线上环境相同。

十一,日志。把日志当做事件流,日志应该是事件流的汇总,将所有运行中的进程和后端服务的输出流,按照时间顺序收集起来。

十二,管理进程。后台管理任务当做一次性进程运行,一次性管理进程应该和正常的常驻进程使用同样的环境,这些管理进程和任何其他的进程一样,使用相同的代码和配置,基于某个发布版本运行,后台管理代码应该随其他应用程序代码一起发布,从而避免同步问题

云原生架构原则及常用模式

云原生架构演进原则

弹性:微服务采用无状态设计,支持按需使用、自动水平伸缩;实例快速启动,并在不影响业务的前提下优雅中止。这一点可以充分利用云的弹性的特征,利用云环境提供的镜像、监控、资源动态编排和调度服务。设计应用程序时,不绑定特定基础资源使其能够自由伸展,根据需要增删实例。

分布式:更多强调解耦。应用侧,则是业务逻辑和数据解耦、业务逻辑和会话解耦,数据分布式,每个服务拥有自己的数据库,服务不能直接访问其他服务的数据库,只能通过服务接口访问其他服务的数据。

高可用,高可用的概念范畴比较广,云原生应用的设计特征,`Design For Failure,即“为失败而设计”`,这里主要强调基于不可靠的基础设施资源来设计高可用系统,并且在应用实例失效的情况下,系统能快速发现并恢复。高可用的设计的主要原则有可观测、可灰度、可回滚等。实现的方式有很多种,比如,通过k8s实现POD状态的监测和维护,通过灰度发布、蓝绿部署等手段来保证升级、回滚时系统的高可用。

自动化:业务/服务的颗粒度更小,交付部署更频繁,迫切需要系统能够自动化部署同时要增强对服务以及所部署的软硬件环境的全方位监控、评估能力。

自服务:自服务强调服务可被其他应用或开发者自助发现,自助按需获取,自助使用并计量,自助服务管理。自服务的前提是高度自治,同时,从易用性的角度,暴露友好的交互方式(Web界面、命令行、SDK...),使能应用开发者简单、高效地使用其提供的功能

云原生应用架构思考:

单体架构的局限性

单体架构的问题不在于不可拆分上,在于无法隔离和自治。应用规模越大,局限性越明显

云原生架构模式: 微服务架构

微服务架构 就是其中一种实现方式。它实现了服务彻底拆分,各服务可以独立打包、独立部署和独立升级,对开发者而言,摆脱开发语言的束缚。每个微服务负责的业务比较清晰,利于后期扩展和维护。微服务之间可以采用REST和RPC协议进行通信。同时,微服务架构可以和其他云原生技术完美结合,充分发挥云的优势。

微服务独立性和敏捷性更好,架构持续演进更容易,更适合云原生应用

云原生架构模式: Serverless架构

Serverless (无服务器架构) 指的是由开发者实现的服务端逻辑运行在无状态的计算容器中,它由事件触发,完全被第三方管理,Serverless是在传统容器技术和服务网格上发展起来,更侧重让 使用者只关注自己的业务逻辑即可 。

Serverless方案 业务价值更轻量化 :

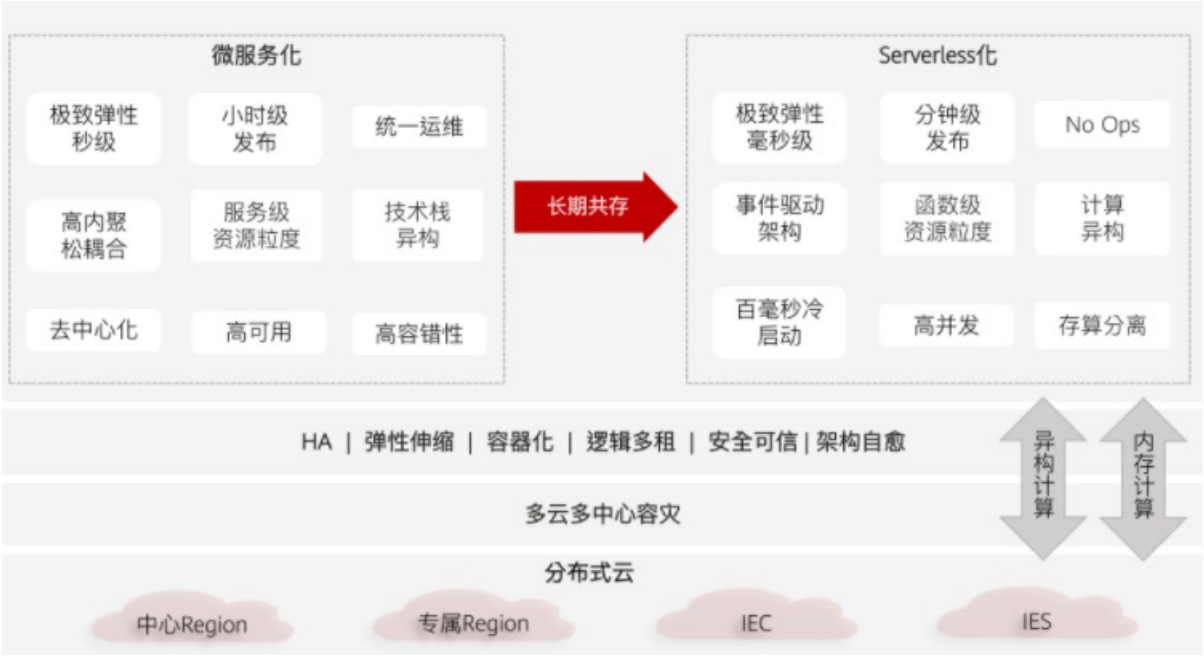
- 用户专注于业务创新和代码开发,代码运行环境由云平台提供,无需管理基础设施资源。
- 更快弹性:根据请求的并发数量自动调度资源运行函数,毫秒级弹性伸缩,高效应对业务峰值。
- 更低成本:根据函数调用次数、运行时长和节点转换次数计费,函数不运行时不产生费用,更加节省成

适用场景:

- 短时运行处理:闲时报表处理、定时日志分析、小程序后端
- 事件驱动处理:实时图片处理、实时数据流处理、IoT规则/事件处理
- 显著波峰波谷:视频转码、视频直播、热点事件推送

Serverless与微服务的关系:

微服务向Serverless演进,并长期共存



华为云云原生解决方案

华为云长期投入云原生技术与产业,是全球云原生领域领导者

华为云基于擎天架构

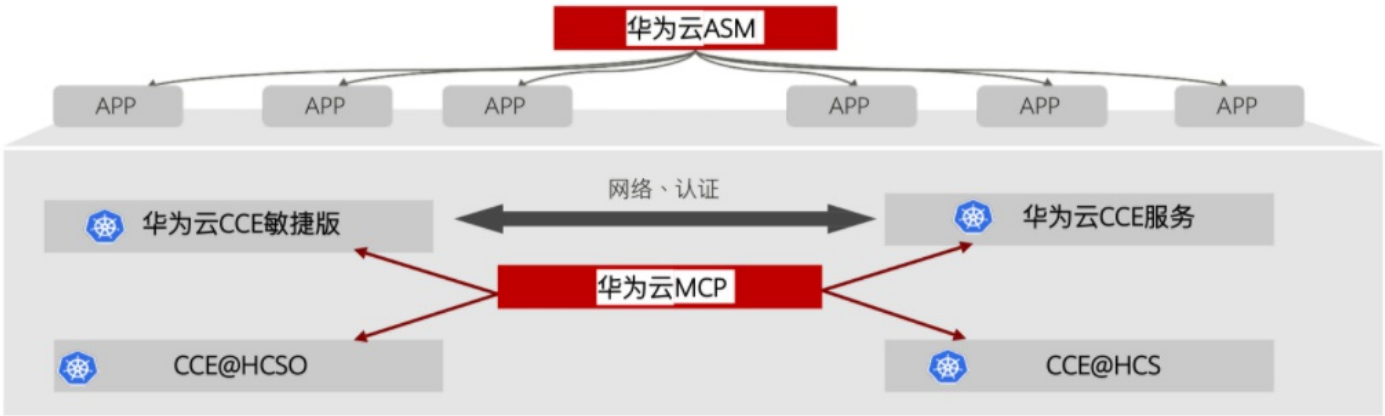
云原生基础设施:在云原生基础设施方面,华为云基于擎天架构实现了基于应用SLA来灵活调度算力,根据应用IO的不同,动态分配网络带宽,根据应用粒度大小,自动分配不同的存储。云原生应用与基础设施不再割裂,而是相互可感知,从而为业务提供更高性价比、资源高效的容器服务。

极致弹性、极致性能的云原生基础设施底座

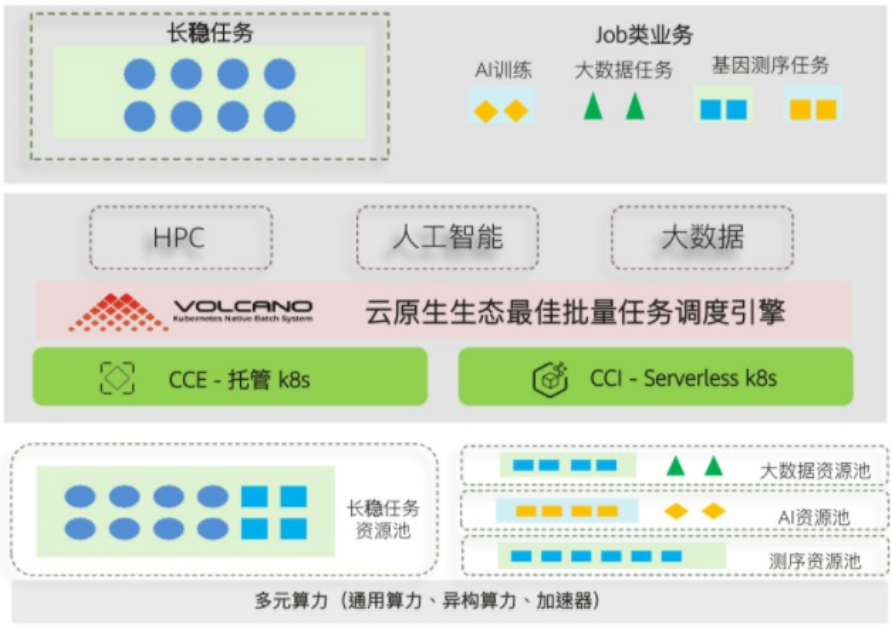
极致轻量	快速并发	硬件直通	动态队列	Warm Pool	独立VF	POSIX卸载	NAS卷直挂
硬件全卸载，资源“0”损耗	百容器启动缩短30%	性能提升40%	PPS提升2倍	端到端打通<2s	时延降低50%	单进程节省30M内存	效率提升30%

基于云原生基础设施的多云管理解决方案

- 多集群、多区域、多云统一应用管理能力，实现更大规模业务支撑能力，更灵活的弹性与容灾能力
  - 多云容器平台（MCP）：基于多集群联邦技术完成多个不同区域、不同云的K8s集群与应用统一管理
  - 应用服务网络（ASM）：完成多个不同区域、不同云的Kubernetes集群上所部属应用的全局治理



基于云原生基础设施的高性能计算解决方案



高性价比

- 统一计算平台，业务混合部署，集群利用率 平均提升30%
- 昇腾，鲲鹏 - 高性价比异构算力

极致性能

- Volcano智能调度，批量任务效率 平均提升30%
- 30秒发放1000容器，满足极速弹性要求

高效运维

- Serverless服务，基础设施NoOps
- 自动+自定义指标弹性伸缩

基于云原生基础设施的边云协同解决方案

云原生边缘架构

- 以标准化的容器在中心云和边缘云一致性地交付应用已成为业界共识，K8S成为云边统一的云原生底座。

极轻极简

- 支持容器和函数两种轻量化运行时，最小可支持128MB内存的边缘设备。

边云协同

- 云端训练、边缘推理，使能40+边缘AI算法、IoT、时序数据库、流计算等延伸到边缘；
- 可与10+云服务进行连接协同，实现边云协同。

异构算力

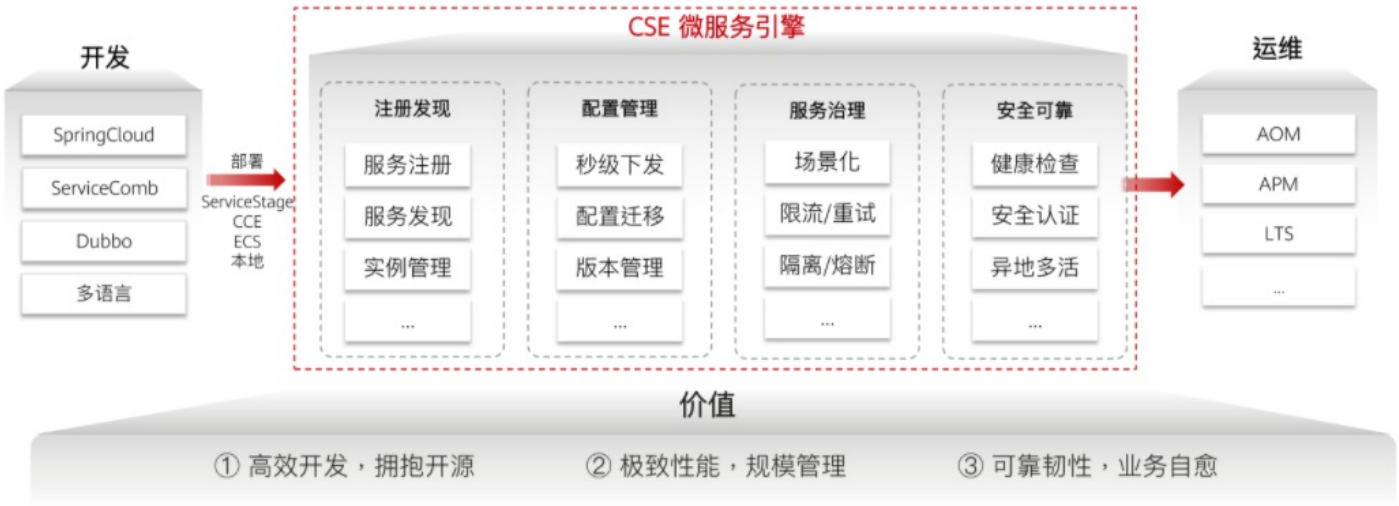
- 除X86、GPU之外，与华为高性能鲲鹏+昇腾基础设施深度整合，提供高性能、低成本的边缘AI推理算力，性价比提升30%。

`KubeEdge`是华为捐献给`CNCF`的第一个开源项目,也是全球首个基于`Kubernetes`扩展的,提供`云边协同能力`的开放式边缘计算平台。KuberEdge就是依托Kubernetes的容器编排和调度能力,实现云边协同、计算下沉、海量设备接入等。

KubeEdge的边缘相当于把Node节点拉远,并进行一些优化从而实现边缘自治; Kubernetes master运行在云端,用户可以直接通过kubectl命令行在云端管理边缘节点、设备和应用,使用习惯与Kubernetes原生的完全一致,无需重新适应。边缘端部署轻量级进程,并支持边缘节点的离线运行

华为云DevSecOps让应用开发更安全...

企业级微服务管理平台,加速微服务应用开发和高可用运维



云原生未来发展趋势

Kubernetes编排统一化:编排对象不断扩展延伸

`Kubernetes 的编排对象持扩展`

- 以容器为基础编排对象逐渐延展至虚拟机、函数等,理论上所有可编程、有API、可抽象成资源的对象,都在成为Kubernetes的编排对象。

`应用侧围绕Kubernetes生态加速演`

- 以Kubernetes为核心的云原生技术栈将推广到更多的应用场景。在大数据领域,Spark和Kubernetes的集成已经非常普遍;机器学习方面,用Kubernetes去编排机器学习的工作流以取得业界的广泛共识。

服务治理Mesh化:加速传统应用转型

`Istio`、 `Consul`,`Linkerd`是`Service Mesh`领域最受欢迎的三大解决方案。

`Mesh`化是传统应用转型云原生的关键路径

- 服务治理与业务逻辑解耦：将服务通信及相关管控功能从业务程序中分离并下层到基础设施层,使其和业务系统完全解耦,使开发人员更加专注于业务本
- 异构系统的统一治理：通过服务网格技术将主体的服务治理能力下沉到基础设施,可方便地实现多语言、多协议的统一流量管控、监控等需求。

传统应用架构中`业务和功能`耦合度较高,无法充分发挥云的效能,传统应用中用于`治理服务的中间件服务通常与应用强绑定部署`,治理能力被植入每个应用,重复造轮子现象严重。

`Mesh`化加速业务逻辑与非业务逻辑的解耦。将非业务功能从客户端SDK中分离出来放入独立进程,利用Pod中容器共享资源的特性,实现用户无感知的治理接管。服务治理的Mesh化为传统应用轻量化改造提供了前提,也为云平台沉淀通用服务治理能力,加速中间件下沉为基础设施提供了可能。

应用服务Serverless化:更加聚焦业务的核心价值

`Serverless`将进一步释放云计算的能力,将安全、可靠、可伸缩等需求交由基础设施实现,使用户仅需关注业务逻辑而无需关注具体部署和运行,极大地提高应用开发效率。同时这个方式促进了社会分工协作,云厂商可以进一步通过规模化、集约化实现计算成本大幅优化。