



# SOFT130091.01

## 云原生软件技术

### 1. 概述



复旦大学软件学院  
沈立炜

[shenliwei@fudan.edu.cn](mailto:shenliwei@fudan.edu.cn)

# 现代软件行业的需求



## 现代应用的上线交付时间要求越来越短

需要一个新的软件架构以及开发方式来确保软件应用能够满足上线交付时间越来越短的要求



## 现代软件需要支持的设备形式越来越多样化

需要一个更加完善的应用程序设计理念以及更加强大、灵活的计算基础设施来满足设备形式越来越多样化的要求



## 现代应用对服务的可靠性要求越来越高

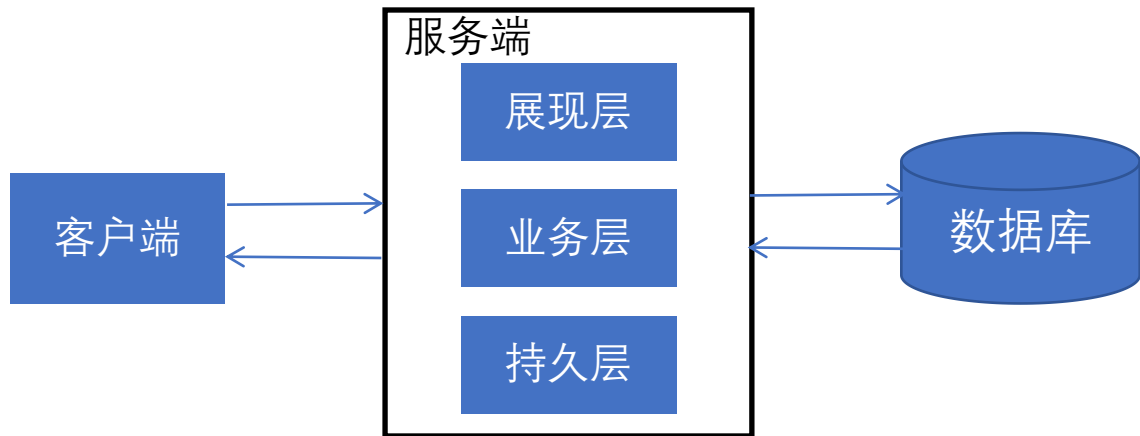
需要一个容错性好、易于管理和便于观察的松耦合系统来满足服务可靠性越来越高的要求

# 软件架构的变化

- 从软件架构的变化发展过程中，可以发现软件架构不是永远一成不变的，随着时间的推移，技术的不断创新与发展，新的技术会被应用，软件架构会被重新打磨、迭代改进与完善。
- 软件架构的发展
  - ✓ 集中式架构
  - ✓ 分布式架构
  - ✓ 云原生架构

# 软件架构的变化 - 集中式架构

- 集中式架构，也称为单体架构，就是系统的所有功能被整体部署到同一个进程中。
- 比较经典的标准三层分层设计：展现层、业务层、持久层
- 随着时间的推移，当系统规模继续增长以及业务场景越来越复杂后，系统部署维度等方面就越来越困难。

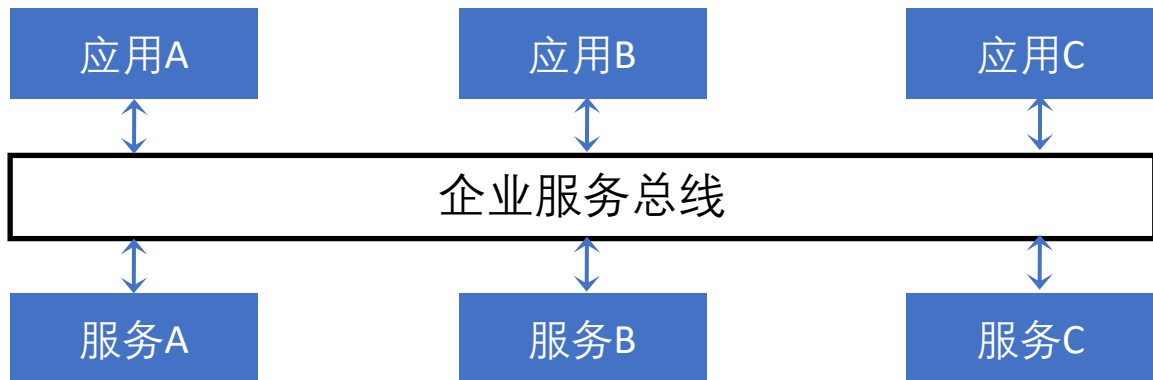


## 软件架构的变化 - 分布式架构

- 分布式架构是将系统内的功能进行划分，分散到不同的服务中，业务由这些不同的服务提供。
- 面向服务的架构（SOA）：将应用程序中的不同功能单元拆分成不同的服务，这些不同的服务通过接口或者规定的协议进行通信，最终提供一系列功能。
- SOA是一种理念
  - ✓ ESB（企业服务总线）是SOA的一种实现方式
  - ✓ 微服务架构是SOA的另一种实现方式

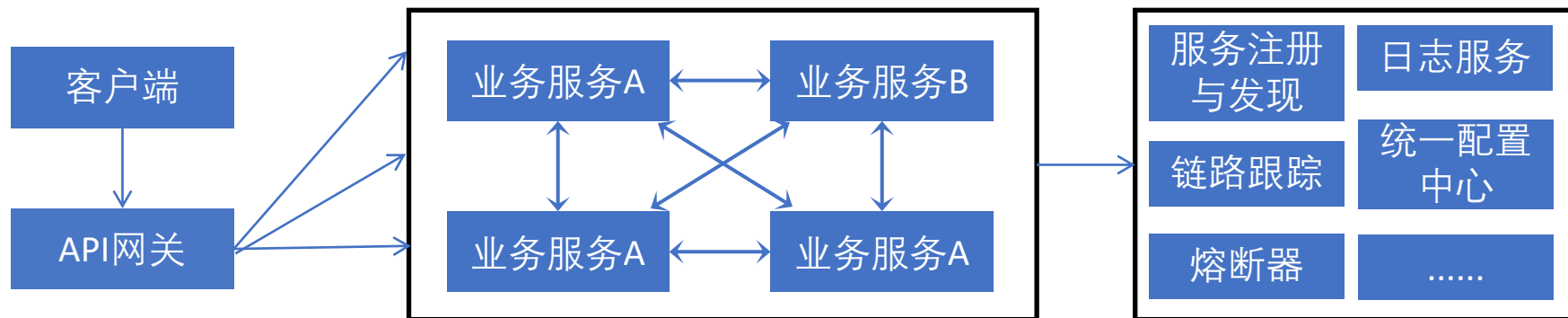
## 软件架构的变化 - 分布式架构

- 企业服务总线的主要作用是提供网络中最基本的连接中枢，是一种中心化的形式。
- 随着时间的增加以及应用规模的增长，企业服务总线可能会成为影响系统瓶颈的重要因素，同时也可能成为影响整个系统正常运行的故障点。
- 如果一个服务出现问题，可能造成企业服务总线的阻塞。



# 软件架构的变化 - 分布式架构

- 在微服务架构下，系统中的每一个服务都是一个独立的可部署单元，各个服务之间相互解耦并且通过远程通信协议进行通信，是一种去中心化的形式。
- 由于分布式系统的内生复杂性，也就是服务通信与服务治理等方面的复杂性，还需考虑服务注册与发现、统一配置中心、链路追踪等方面问题。



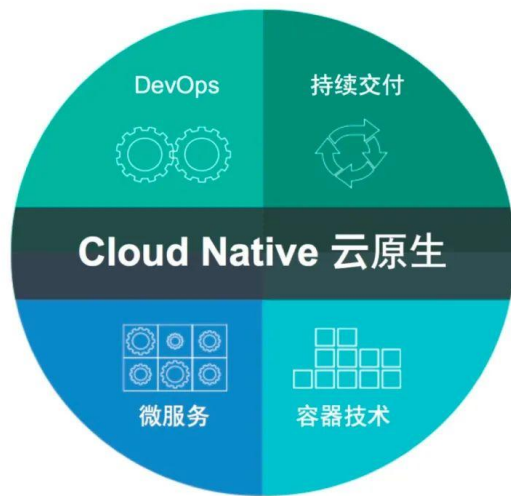
## 软件架构的变化 - 微服务架构往云原生架构的发展

- 一个由业务微服务与非业务的基础服务组件组成的微服务架构才能称为一个较为完善的微服务架构。
- 随着时间的推移和技术的不断发展，特别是容器、容器编排等技术的推进，同时随着云计算的落地实现，在微服务架构的基础上不断进行交付周期、服务可靠性等方面的提升。
- 在使用微服务架构的过程中，非业务层面的服务治理等方面的问题已被逐渐解决，软件架构的发展也开始演化为云原生架构。



# 软件架构的变化 - 云原生架构

- 云原生架构是一整套技术体系与方法论。
- 在云原生架构中，应用都是面向云进行架构设计的，生在云上、长在云上。
- 在服务发生故障时可以实现应用服务能力的自我恢复，并且依托于容器等技术，可以实现毫秒级的弹性伸缩等
- 可以依靠云原生架构来实现现代软件行业的需求，即缩短上线交付周期、支持多样化的设备以及提高服务可靠性。



# 云原生概念的起源

## CLOUD (云) + NATIVE (土生土长的)

- 2013年Pivotal公司的马特.斯坦提出了云原生的概念，它是一个思想集合，包括DevOps，持续交付，微服务，敏捷基础设施，康威定律等
  - ✓ 云原生 (CloudNative) 是一个组合词，Cloud+Native。Cloud 表示应用程序位于云中，而不是传统的数据中心
  - ✓ Native 表示应用程序从设计之初即考虑到云的环境，原生为云而设计，在云上以最佳姿势运行，充分利用和发挥云平台的弹性 + 分布式优势。

# 云原生发展历程

“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。这些技术能构建容错性好、易于管理和便于观察的松耦合系统，并结合可靠的自动化手段使工程师轻松对系统做出改进和可预测的重大变更

2001年  
虚拟化

vmware

2006 IaaS

aws

2009  
公有云PaaS

HEROKU

2010  
开源IaaS

openstack

2011  
开源PaaS

CLOUD FOUNDRY

2013  
Docker  
云原生概念

docker

2014  
Kubernetes



2015  
CNCF

CLOUD NATIVE  
COMPUTING FOUNDATION

2018  
CNCF重定义

CLOUD NATIVE  
COMPUTING FOUNDATION

以资源为核心

以应用为核心

# 云原生计算基金会 - CNCF

CNCF 致力于通过培养和维持一个开源、供应商中立的项目生态系统来推动云原生技术的广泛采用，进而实现让云原生无处不在的愿景。CNCF 成立 5 年多来，开源为云原生技术带来了前所未有的发展浪潮，极大的加速了云原生在全球范围内快速应用和发展。云原生技术生态也日趋完善，细分项目不断涌现。相较于早年的云原生技术生态主要集中在容器、微服务、DevOps 等技术领域，现如今的技术生态已扩展至底层技术、编排及管理技术、安全技术、监测分析技术、大数据技术、人工智能技术、数据库技术以及场景化应用等众多分支，初步形成了支撑应用云原生化的全生命周期技术链

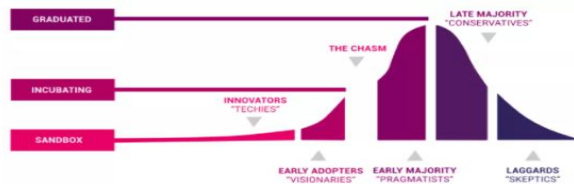


## 2015年，在谷歌倡导下成立

CNCF是一个厂商中立的基金会，致力于快速成长的开源云原生技术的推广和普及，  
1) 容器化封装；2) 通过中心编排系统的动态资源管理3) 面向微服务。



## CNCF的项目成熟度模型



## 项目分布

开始只有K8s一个项目，几年来快速发展，开始向原来不擅长的领域快速发展，例如大数据、人工智能、高性能计算、边缘计算等。

# 云原生理念

- 解耦
- 封装
- 高可扩展的分布式
- 敏捷开发，持续部署
- 抗脆弱性

## 解耦

- 从大型机、服务器到虚机（VM）、再到容器，资源分配的颗粒度越来越小，启动速度越来越快，资源重建的代价越来越小，都是因为解耦
- 云原生有很多思想，关键点就是“分而治之”，就是“解耦”，解耦的原则是“高内聚、低耦合”；而单体应用是把很多功能耦合在一起，没有协同，很难交付
- 将应用分解为高内聚、松耦合的模块，模块之间通过远程API协作

# 封装

- 应用和环境封装在一起，同时发布
- 敏捷基础设施指使用脚本或文件配置计算基础设施环境，而不是手动配置环境的方法
  - ✓ 敏捷基础设施也可称为基础设施即代码（Infrastructure as Code）或者可编程基础设施（Programmable Infrastructure），基础设施即代码可以将基础设施配置完全当作软件编程来进行。
  - ✓ 开始让编写应用和创建其运行环境之间的界限变得逐渐模糊。
  - ✓ 应用可能包含用于创建和协调其自身虚拟机或容器的脚本。

- 不可变基础设施是一种基础设施模式，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，就先对公共镜像进行修改，然后用镜像构建新服务器来替换旧服务器。经过验证后，新服务器投入使用，旧的服务器就会下线。
- 不可变基础设施的好处是在基础设施中有更多的一致性和可靠性。



## 高可扩展的分布式

- 分布式系统看上去就像是一台机器在工作，但其实它由一组独立的机器组成，他们之间通过网络相连接
- 分布式系统可以将计算任务分配到不同的机器上，这种分配任务的能力使得应用具有可扩展性、可靠性，同时更经济
- 高度可扩展，弹性伸缩、动态调度、优化资源利用

# 敏捷开发，持续部署

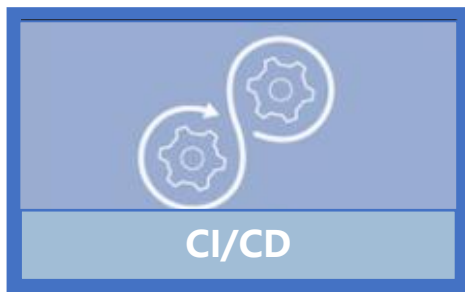
快速、频繁地升级应用程序，而不影响业务服务



关注于...  
**开发过程**

强调的是...  
**变化**

效果是  
**加速开发**



关注于...  
**软件定义的软件工作**

强调的是...  
**工具**

效果是  
**自动化**



关注于...  
**文化**

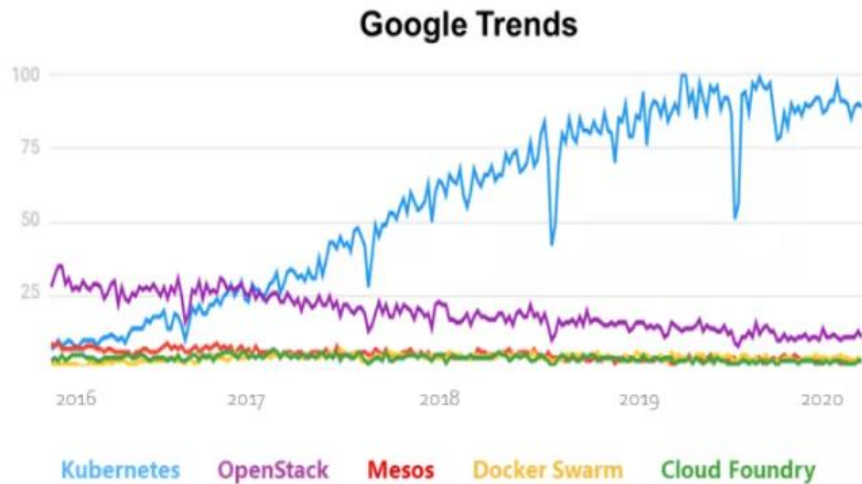
强调的是...  
**角色**

效果是  
**快速响应能力**

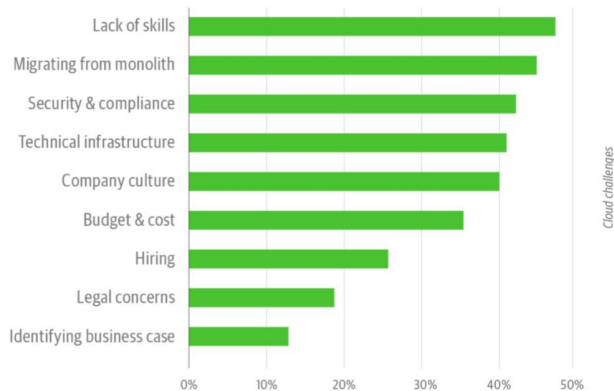
# 抗脆弱性

- Netflix大胆提出反脆弱架构理念，为了让系统更加健壮，不是将它们严格保护起来，而是通过架构处理让它们能适应各种“脆弱”
- Netflix从周一到周五，上午9点到下午3点，混乱猴子军团会随机杀生产实例，这个叫Chaos Monkey，还有增加延迟的Latency Monkey
- 云原生应用需要有能容忍故障的设计，尽可能地优化各种故障场景的响应，这将让团队时刻想到服务故障的情况下用户的体验
- 云原生的微服务框架提供了大量故障检测和恢复手段，例如断路器、熔断机制、观测手段、度量工具。所以，云原生应用通常实时监控各个层次的指标，检测架构元素（每秒数据库接收的请求数）和业务指标（每分钟接收的订单数），以提供一种可视化的早期监控指标。

# 云原生在企业界的发展趋势



What are the biggest challenges your organization is facing after adopting cloud native infrastructure? (Select all that apply)



过去几年中，云原生关键技术正在被广泛采纳，CNCF 调查报告显示，超过 8 成的用户已经或计划使用微服务架构进行业务开发部署等

# 云原生技术体系的四大组成部分

## 容器

- 应用与依赖封装在一起，微服务发布的最佳载体
- 轻量化，启动快，高效低耗
- 非常适合引擎进行编排

- Docker
- Kubernetes (K8S)
- 容器存储、容器网络

## 微服务

- 单一职责的业务模块，通过远程网络调用协作
- 每个服务可被独立部署、更新、扩容、重启
- 最大限度支持低成本的可扩展、可演进

- Spring Cloud
- Service Mesh

## DevOps

- 关注整体价值交付的效率，而不是单独的开发或运维
- 打通专业壁垒
- 度量驱动开发（MDD），强调可视化、量化、自动化

- Docker
- Kubernetes
- Jenkins

## 持续交付

- 把系统看待成产品而不是项目
- 快速发布、快速反馈、频繁发布、打通业务-IT闭环
- 自动化部署（蓝绿、金丝雀、灰度）

- Jenkins
- Git
- Ansible

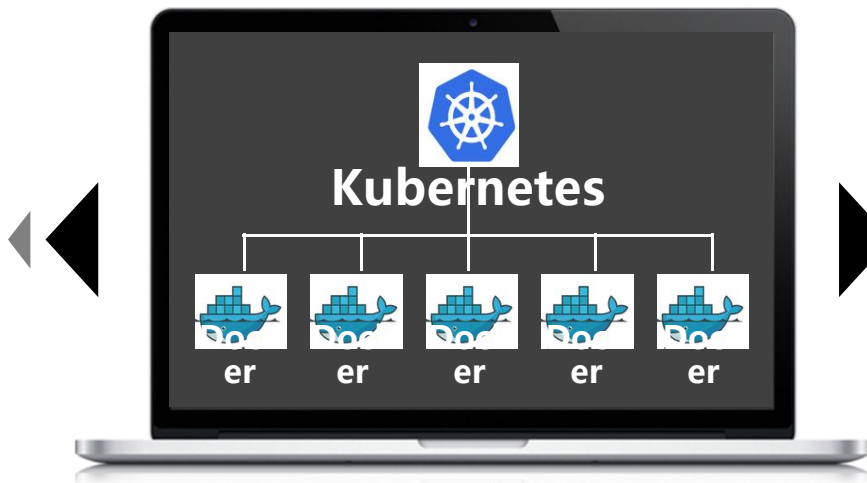
...

# 容器是云原生的计算环境

## Docker

云原生最佳封装器

- 轻量化虚拟技术，操作系统内部的应用隔离
- 启动快，小于1秒
- 程序、环境、依赖项、系统库全部封装在容器内，以容器镜像为单元部署，对环境无依赖
- 较好的隔离性
- 较高的部署密度



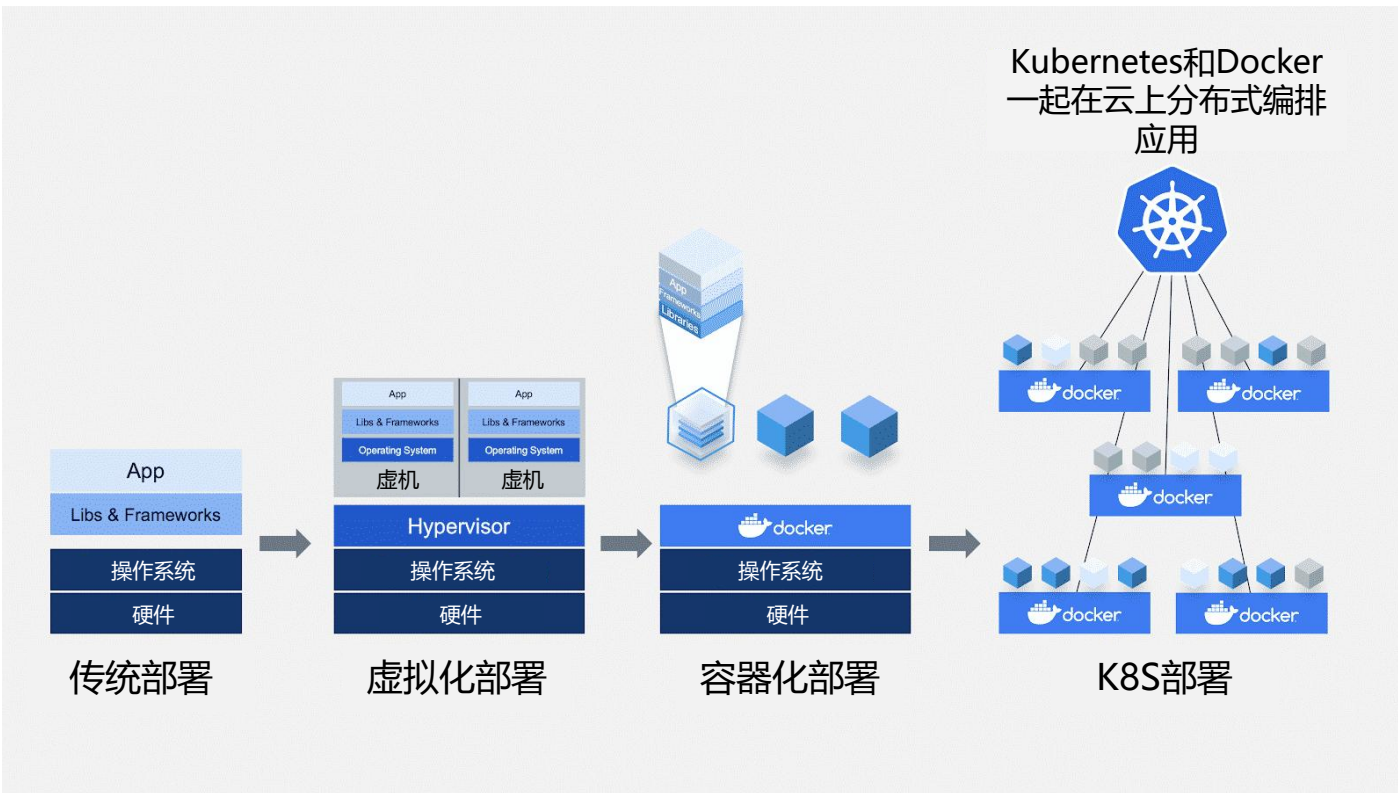
容器技术不仅仅是Docker和K8S  
容器存储、容器网络、容器GPU、容器HPC、  
容器边缘计算等等

## Kubernetes(K8S)

容器编排引擎

- 在节点上创建和部署容器
- 资源管理，即把容器部署到有足够资源的节点上，或者当节点超出限额时转移到别的节点上
- 监控容器运行，出现故障时重启或重新编排
- 在集群内对容器自动扩容/缩容
- 为容器提供网络映射服务
- 在集群内为容器提供负载均衡服务

# 容器是云原生的计算环境



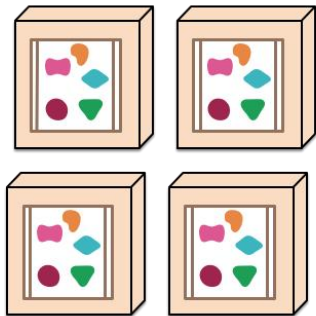
# 微服务：具有独立业务价值的单元，通过网络协议合作

- 围绕业务功能分解为小型的、松耦合的各种服务，由独立团队开发和管理，每个服务都专注于一个特定的任务，由一个小团队负责开发和运营。
- 每个服务被视为一个独立的应用，有独立的团队、测试、开发、数据和部署。这些服务在独立的进程中运行，相互之间通过同步或异步消息的API进行通信

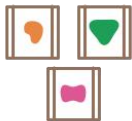
A monolithic application puts all its functionality into a single process...



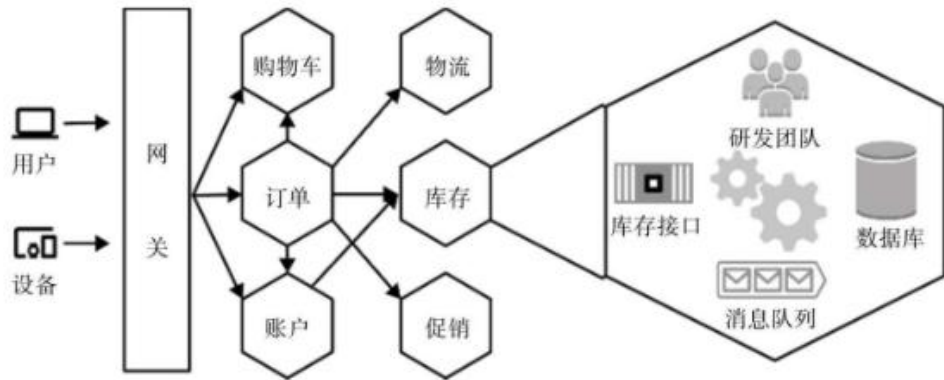
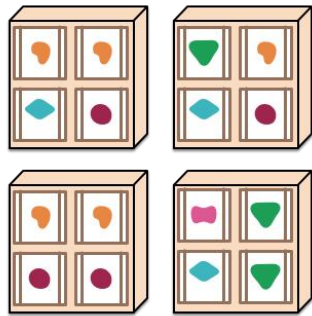
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...

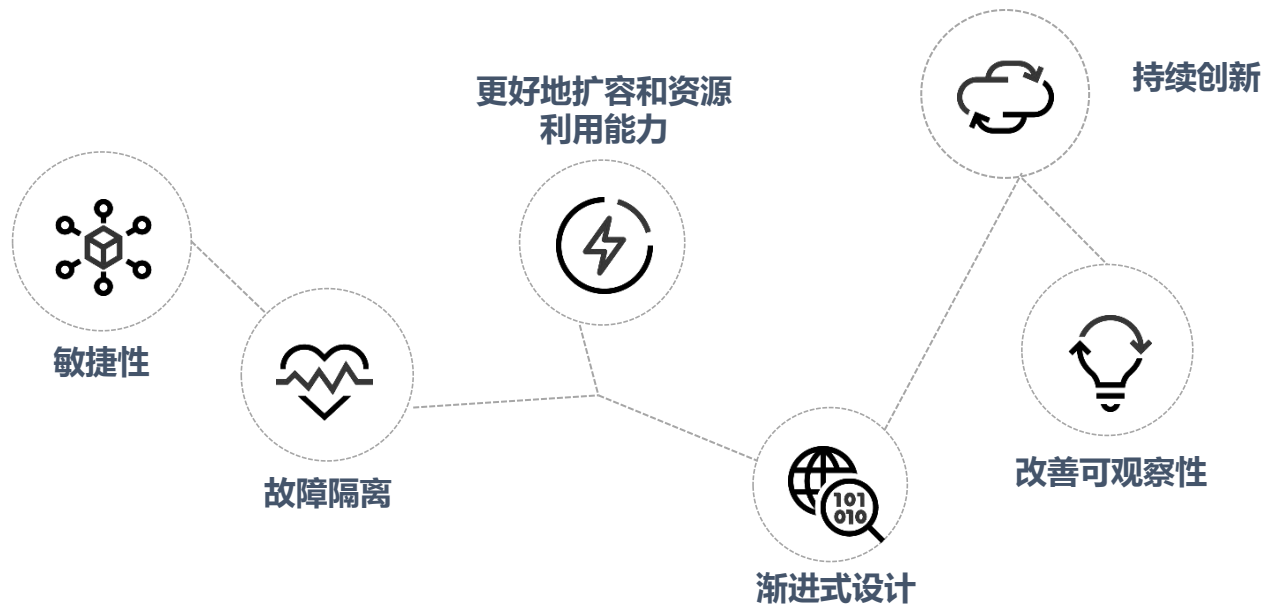


... and scales by distributing these services across servers, replicating as needed.



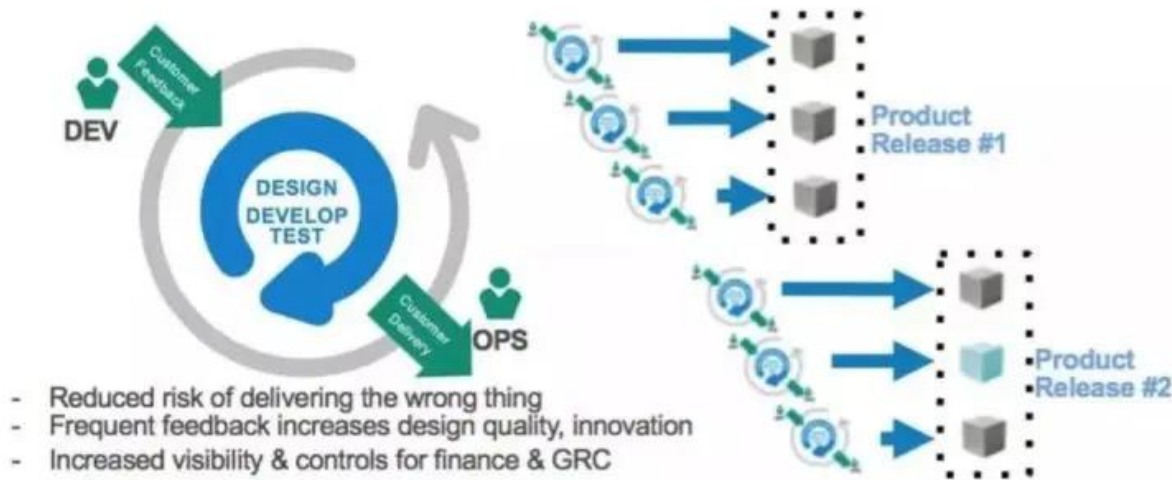


# 微服务的优势



正确实施微服务架构有助于提升大型应用的发布频率，使得企业能够更快地为客户提供更可靠的服务

# DevOps和持续交付的理念



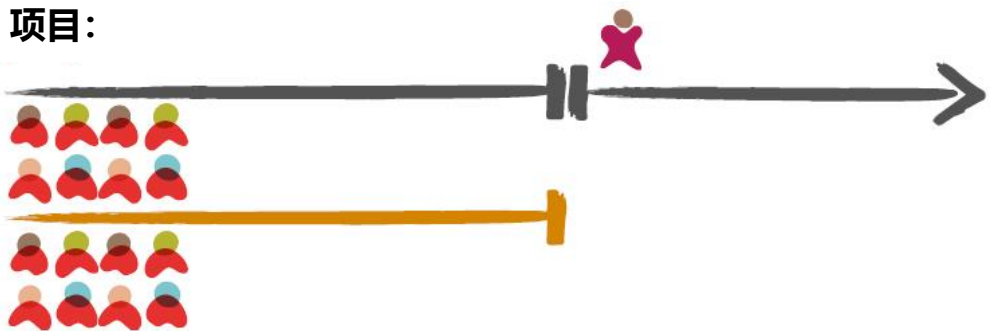
**用小步快跑的方式，打破瀑布式开发流程**

**DevOps:** Dev+Ops, 就是开发和运维合体，避免开发和产品，开发和运维的专业壁垒和冲突；DevOps 是一个敏捷思维，沟通文化，也是组织形式，为云原生提供持续交付能力打好文化基础。

**持续交付:** 持续交付是不误时开发，不停机更新，小步快跑，反传统瀑布式开发模型，要求开发版本和稳定版本并存，需要流程和工具支撑。

# 运维模式的区别：传统应用 vs 云原生应用

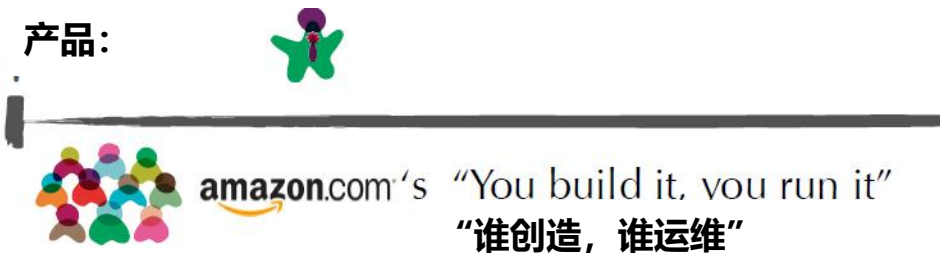
项目：



是产品，不是项目

- 传统应用的开发模式：提供一些被认为是完整的软件，一旦开发完成，软件将移交给维护部门，然后，开发组就可以解散掉了
- 云原生应用认为开发组应该负责产品的整个生命周期，“谁建造，谁运维”的理念，它要求开发团队对软件产品的整个生命周期负责，这要求开发者每天都关注软件产品的运行情况，并与用户联系的更紧密，同时承担一些售后支持
- 成熟的产品会与业务功能进行绑定，除了把软件看成既定的功能的集合之外，会进一步关心“软件如何帮助用户实现业务功能”这样的问题

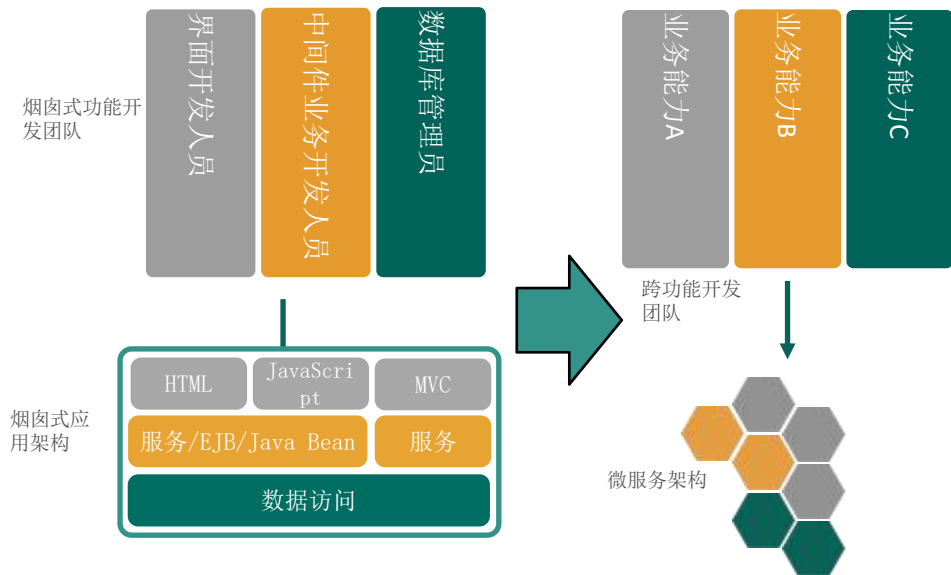
产品：



amazon.com's "You build it, you run it"  
“谁创造，谁运维”

Netflix没有技术CTO职位，只有首席产品CPO，工程团队和产品团队的VP都向CPO汇报，这样做是为了产品导向，便于技术和产品沟通合作，避免两边扯，避免业务驱动还是技术驱动的悖论，大家都是产品驱动

# 云原生的团队的组织形式

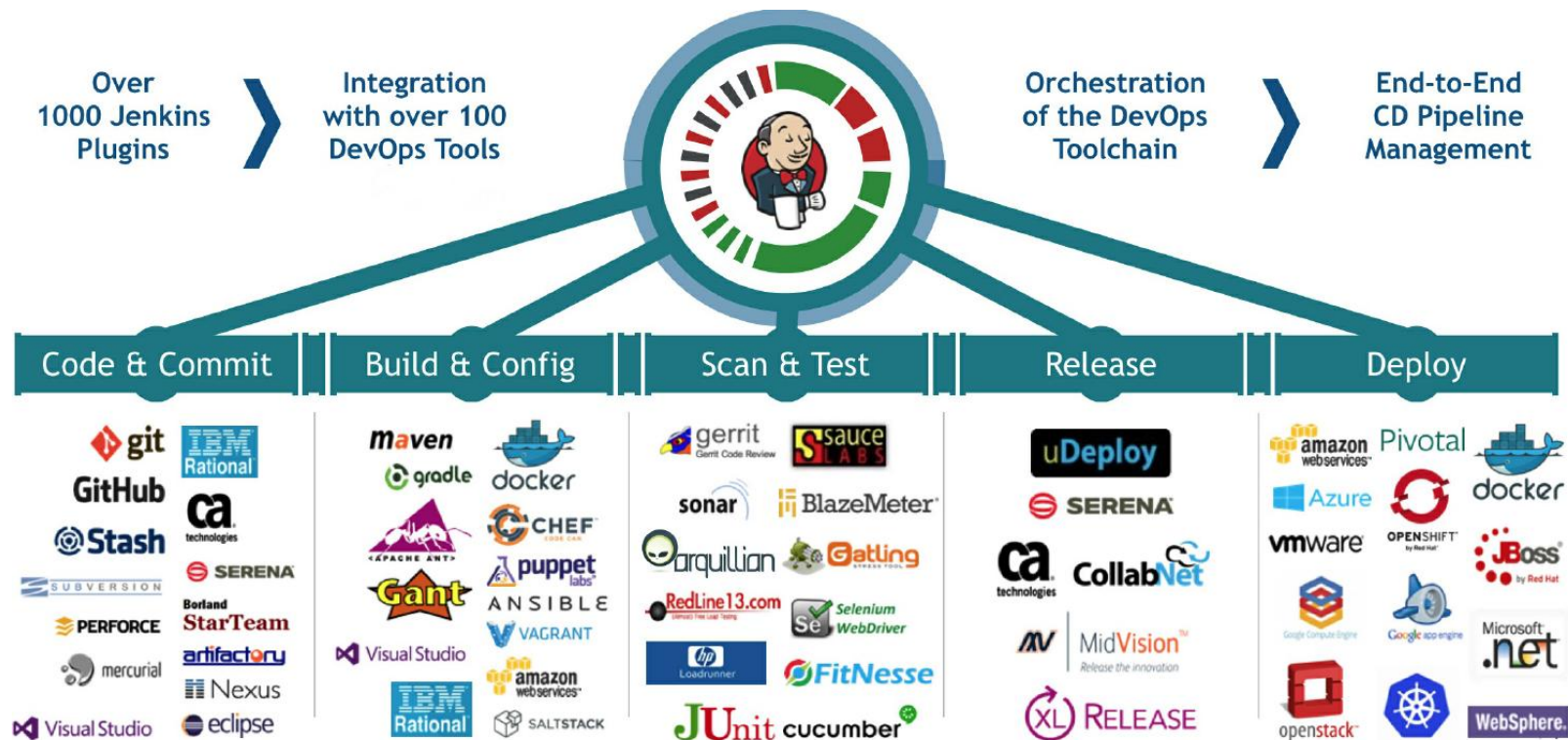


## 围绕业务能力进行组织

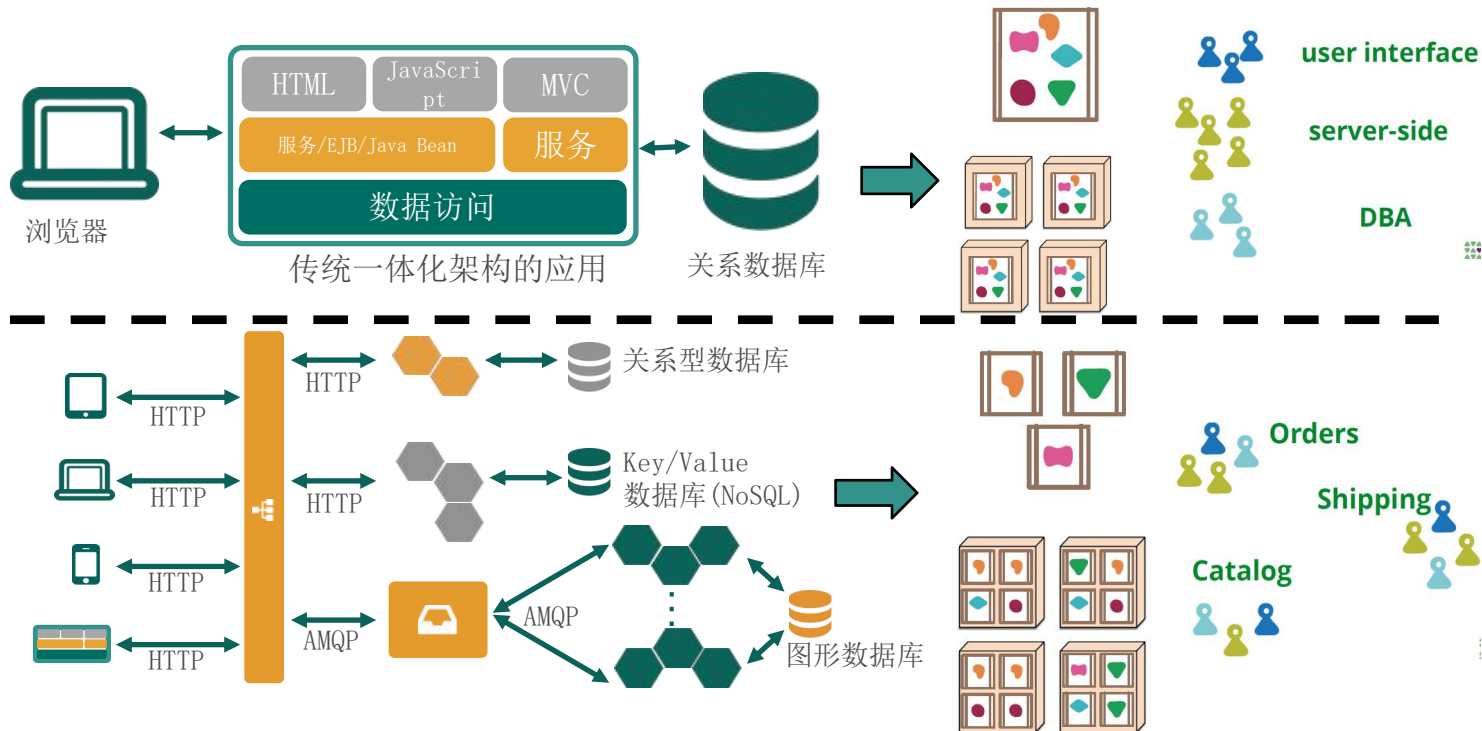
微服务架构团队有如下特点:

- 每个服务制作好一件事，更加专注和简单
- 用合适的工具来做合适的事情
- 服务之间是松耦合的，独立部署
- 服务的团队之间是相互独立的
- 单一功能的改变只需要重新构建、部署相应的服务即可

# DevOps的工具链生态



# 云原生应用 vs 传统应用的概念模型



# 云原生应用 vs 传统应用的区别

传统应用（单体应用）	云原生应用
需求比较固定	需求是持续发展的
是一个项目，完成后交给运维	是一个产品，持续发展
应用模块牵一发而动全身	应用由多个微服务组成，松耦合
垂直扩展；硬件定义可靠性	水平扩展，应用设计消除了对基础设施的依赖
很难根据负载自动扩展	自动，水平扩展
应用的恢复是靠人工的	应用的恢复是自动化的
应用系统的物理环境错误导致业务停顿	应用的物理环境出错是可以接受的
对操作系统和虚机有察觉和依赖	通过容器抽象于操作系统
物理机宕机是重大事故，不遗余力地保障物理机的运转	物理机对应用不是那么重要
积极备份数据以便应用环境出错时恢复	设计时要尽量避免数据恢复的必要
升级时候不睡觉	灰度发布、发布回滚

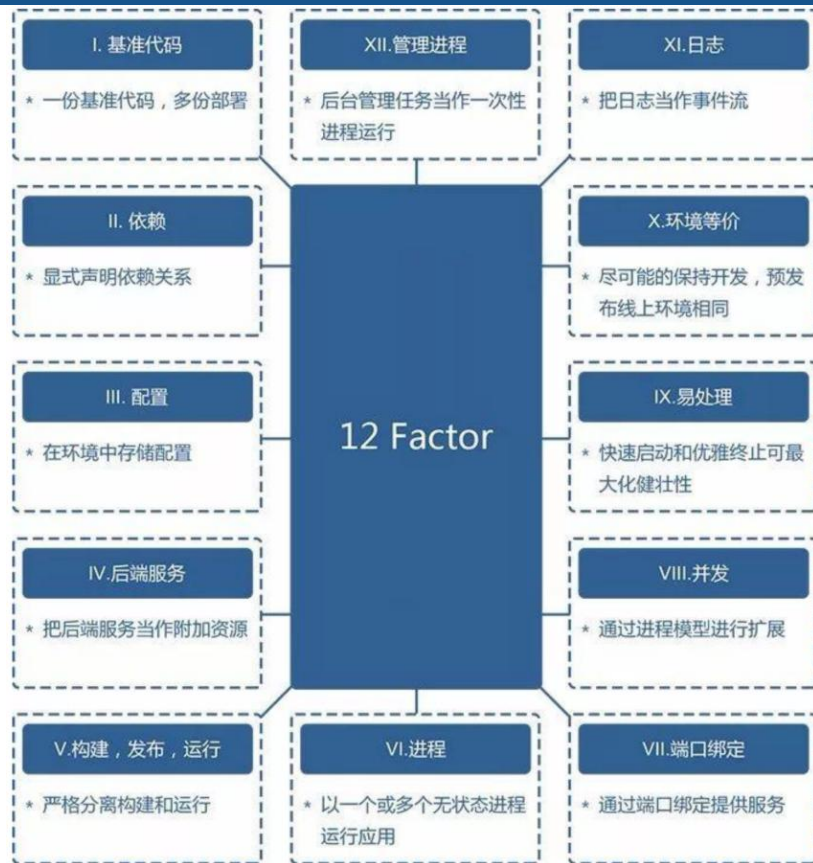
# 两种上云的区别：IaaS 相对于容器

虚拟机	容器
可能需要停机更新	云原生应用始终是最新的，需支持频繁变更，持续交付，蓝绿部署
无法动态扩展，需要冗余的资源以抵抗流量高峰	弹性自动伸缩，通过共享降成本
对网络资源，比如ip，端口等有依赖，甚至硬编码	对网络和存储都没有这种限制
通常人肉部署手工运维	自动化的
通常依赖系统环境	不会硬连接到任何系统环境，依赖抽象的基础架构，良好移植性
单体（巨石）应用	纵向划分服务，模块化更合理
虚拟机的启动需要整体操作系统，花费启动时间会更长	秒级启动
虚拟机镜像包含操作系统，占用几个GB大小	容器镜像小得多
虚拟机纵向扩容（增加更多资源）	横向扩容，主要是增加新的实例，快速启动，快速响应
虚拟机额外开销较多	轻量级虚拟化，部署密度大



# 云原生应用应该什么样？ - 12要素宣言

- 12要素是Heroku工程师从云端应用开发的最佳实践中总结出来的，定义了一个优雅的现代应用在设计过程中需要遵循的基本原则
- 现代化应用特点：
  - 自恢复性：拥抱错误而不是尝试杜绝错误；
  - 敏捷性：快速开发、快速迭代；可运维性：深入应用内部运维而不是依赖应用外围的系统；
  - 可观测性：随时随地查看应用的状态



## 基准代码

- 基准代码是日常工作中存放在版本控制系统中的对应代码库。
- 一份基准代码多份部署，这个代码可以部署到多个环境中，比如开发、测试和生产环境。
- 应用与基准代码之间应该是一一对应的关系。

# 依赖关系

- 依赖关系的声明和隔离在开发云原生应用程序中很重要，许多问题是由于缺少依赖关系或依赖关系的不同版本造成的，根源在于内部部署环境和云环境之间的差异。
- 显式声明依赖关系，即应用应该显式地声明自己的依赖项。
  - ✓ 有利于管理依赖
  - ✓ 当有新的开发人员进入项目组时，只需一个构建命令即可安装所有依赖项解决环境配置问题
- 目前容器技术的使用已经大大减少了依赖性引起的问题，因为依赖关系在Dockerfiles 中声明并且已经打包到容器中。

## 配置

- 应用中用到的信息（数据库地址、账号、密码等）放在配置文件中，文件中的内容一般称为配置。
- 在环境中存储配置，将应用配置信息存储在环境变量中。
- 配置和代码应该严格分开，这样就可以很容易地配置不同的环境。
  - ✓ 例如，测试环境中，有测试配置文件，应用程序部署到生产环境中，替换为生产环境中的配置文件即可。

## 后端服务

- 后端服务是应用运行时所需的通过网络调用的各种服务，例如MySQL、Redis等。
- 将后端服务作为附加资源使用，当部署应用时就按需伸缩这些资源。
  - ✓ 比如云原生中的缓存服务和数据库即服务是后端服务。在访问这些后端服务时，建议通过外部配置系统获取这些服务的配置信息来减少耦合。

## 构建、发布和运行

- 构建是将应用代码进行打包的过程。
- 发布是在相应的环境中部署应用代码包。
- 运行是在相应的环境中启动应用程序。
- 严格分离构建和运行阶段，不要直接跳过构建、发布这两个阶段而直接去修改运行状态的代码。
- 运营团队使用持续集成和持续部署来完全自动构建和发布应用程序。

# 进程

- 进程是正在运行的应用程序实例。
- 使用一个或多个无状态进程运行应用程序，即在日常开发中应用程序的运行一般是以一个或多个进程的形式存在。
- 云中的应用程序应该是无状态的，需要持久化的任何数据都应该存储在外部，这样才能实现云计算的弹性。

## 端口绑定

- 端口绑定指可以通过IP+端口的方式来访问服务。
- 当应用程序启动后，应用程序会监听指定端口的请求。一般情况下都是通过域名访问服务，但是在向域名发送请求时，请求都会被路由到绑定端口的进程中。



## 并发

- 并发指应用程序与计算单元不是一一对应关系，一个程序可以有多个计算单元。
- 通过进程模型进行扩展，应用应采取多进程的运行方式按需启动，并通过进程的水平扩展增大并发能力

## 易处理

- 易处理指应用进程可以快速启动与停止。
- 快速启动和优雅终止可最大化健壮性，即应用可以通过快速启动和停止来确保应用的稳定性。
- 当停止应用时应该妥善处理正在运行的任务，比如将该任务退回至后端队列服务中。

## 环境对等

- 开发环境与线上环境等价指应该减少各个环境、环节之间的差异。
- 尽可能保持开发、预发布、线上环境相同，即应用依赖的基础环境、开发人员等尽可能保持一致。
- 不论是测试、生产等环境下部署的后端服务的版本应该是一致的。
- 利用容器技术将服务所需的依赖性打包，从而减少不一致环境的问题。

## 日志

- 日志用来记录日期、时间、操作者及动作等相关信息。
- 将日志看作是事件流，即日志应该是按照时间顺序汇总的事件流。
- 应用一般会有多个进程，也就会产生多份日志，可以采用日志索引分析系统以便于对日志进行搜索、分析与展示等。

## 管理进程

- 管理进程是执行的一些管理与维护应用的任务。
- 将后台管理任务作为一次性进程运行，即应该一次性运行需要执行的管理与维护应用任务。
- 执行管理与维护应用的任务的环境应该与应用的环境保持一致。

# 云原生：理念+技术体系+方法论+最佳实践

**设计：**领域设计

**开发：**敏捷、DevOps













**封装：**容器

**资源编排：**Kubernetes

**功能编排：**微服务（Spring Cloud, Service Mesh）

**维护：**可观测、DevOps、持续交付、智能运维

**治理：**大规模微服务的复杂性管理

	Development Process	Application Architecture	Deployment & Packaging	Application Infrastructure
~ 1980	Waterfall 	Monolithic 	Physical Server 	Datacenter 
~ 1990				
~ 2000	Agile 	N-Tier 	Virtual Servers 	Hosted 
~ 2010	DevOps 	Microservices 	Containers 	Cloud 

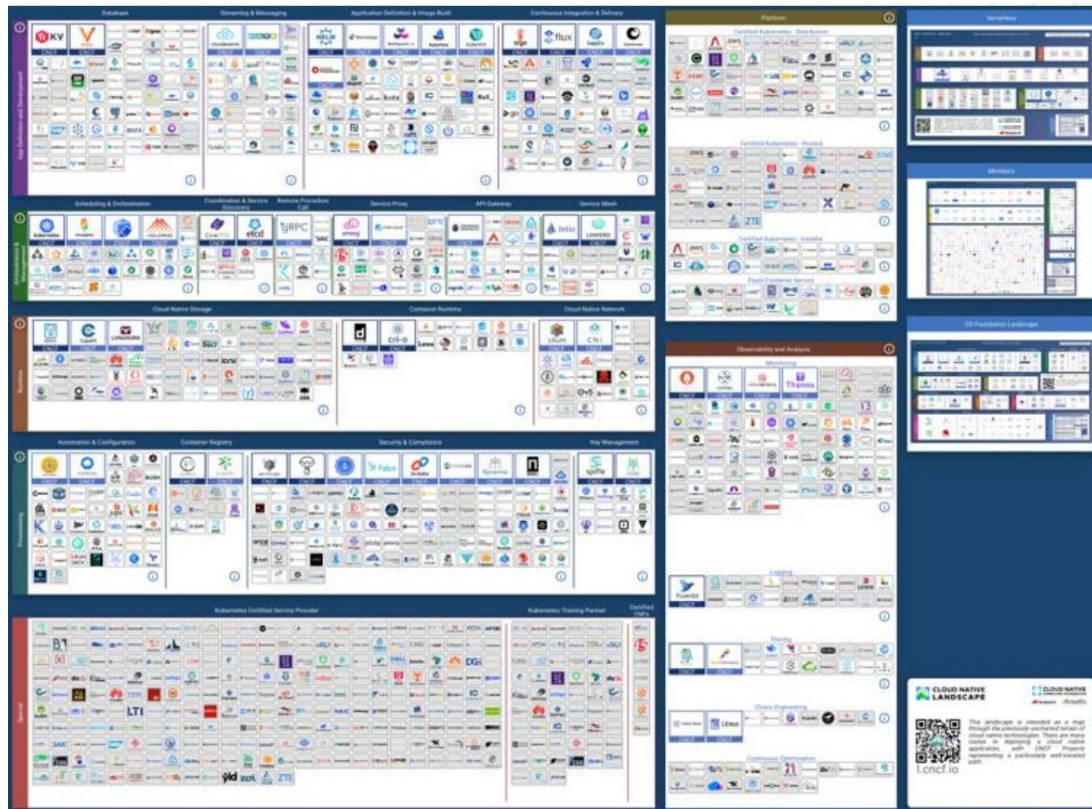
# 云原生全景图

## 从底向上:

- ✓ Provisioning
- ✓ Runtime
- ✓ Orchestration and Management
- ✓ Application
- ✓ Definition and Development

## 贯穿所有:

- ✓ Observability and Analysis
- ✓ Platform



<https://landscape.cncf.io/>

## 小结

- 软件架构的变化
- 云原生的概念和理念
- 云原生技术体系
- 云原生应用12要素



# SOFT130091.01

## 云原生软件技术

**End**

### 1. 概述