微服务演进之道

关于我

华为软件工程技术专家

ThoughtWorks首席咨询师

Sybase Tech Leader

- 微服务、DevOps、持续交付有丰富的实践经验
- 《微服务架构与实践》作者
- 《DevOps实践指南》译者
- 《消费者驱动契约测试-Pact》译者
- 中国首批EXIN DevOps Master教练
- 西安DevOps Meetup 联合发起人
- 《使用SpringBoot/Cloud构建微服务》视频作者(StuQ)

- 微服务架构的核心
- 微服务架构生态系统
- 微服务实践应用案例

Why microservices?

容错性

快速上线

复杂度增加

高可用性

需求快速响应

流量不确定

可管理性

独立发布

创新与尝试

灰度发布

可测试性

技术多样性

组织扩张

Speed

快速上线能力

需求快速响应

不断创新与尝试

独立发布

技术多样性

Safety

可用性

容错性

用户体验

易管理性

易测试性

Speed

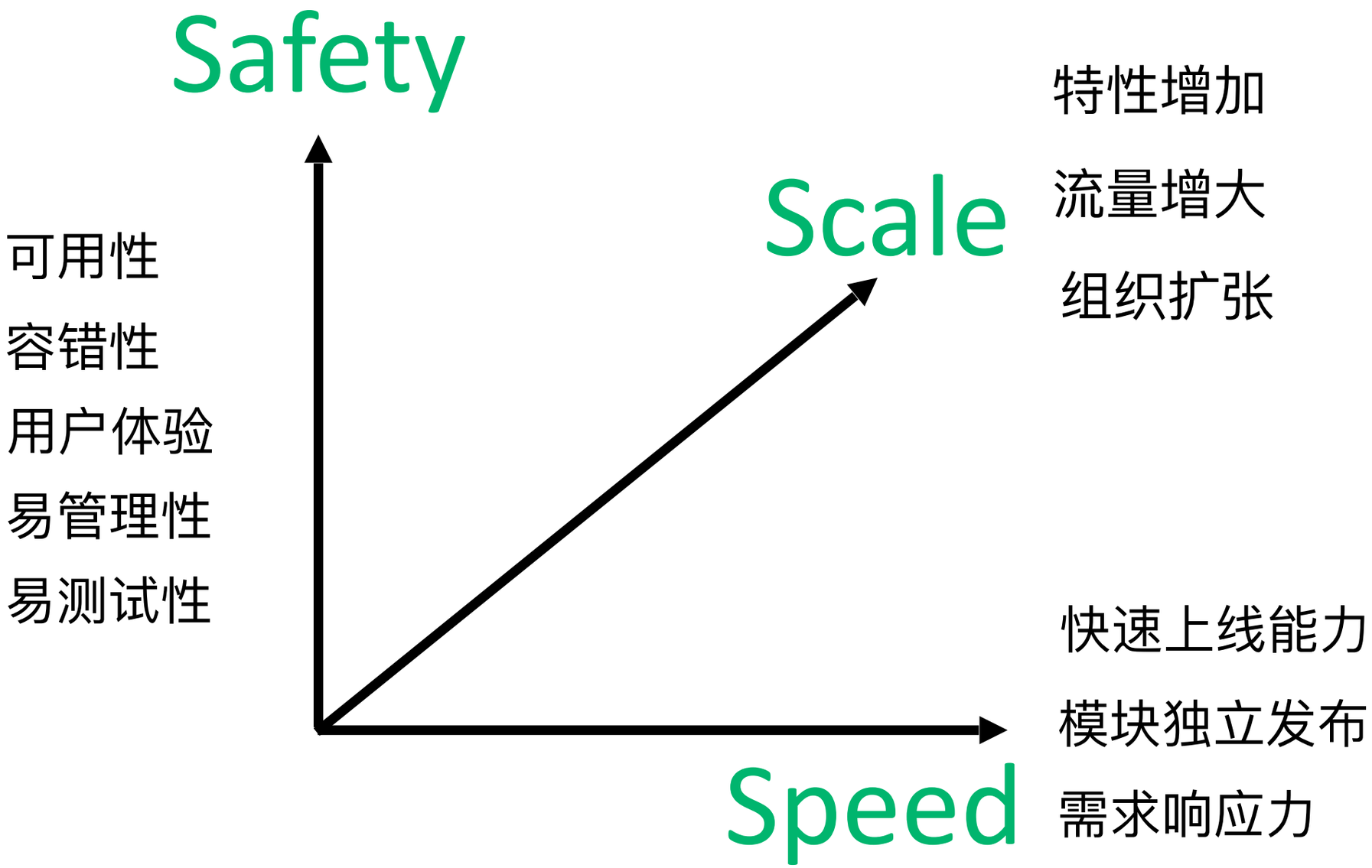
快速上线能力

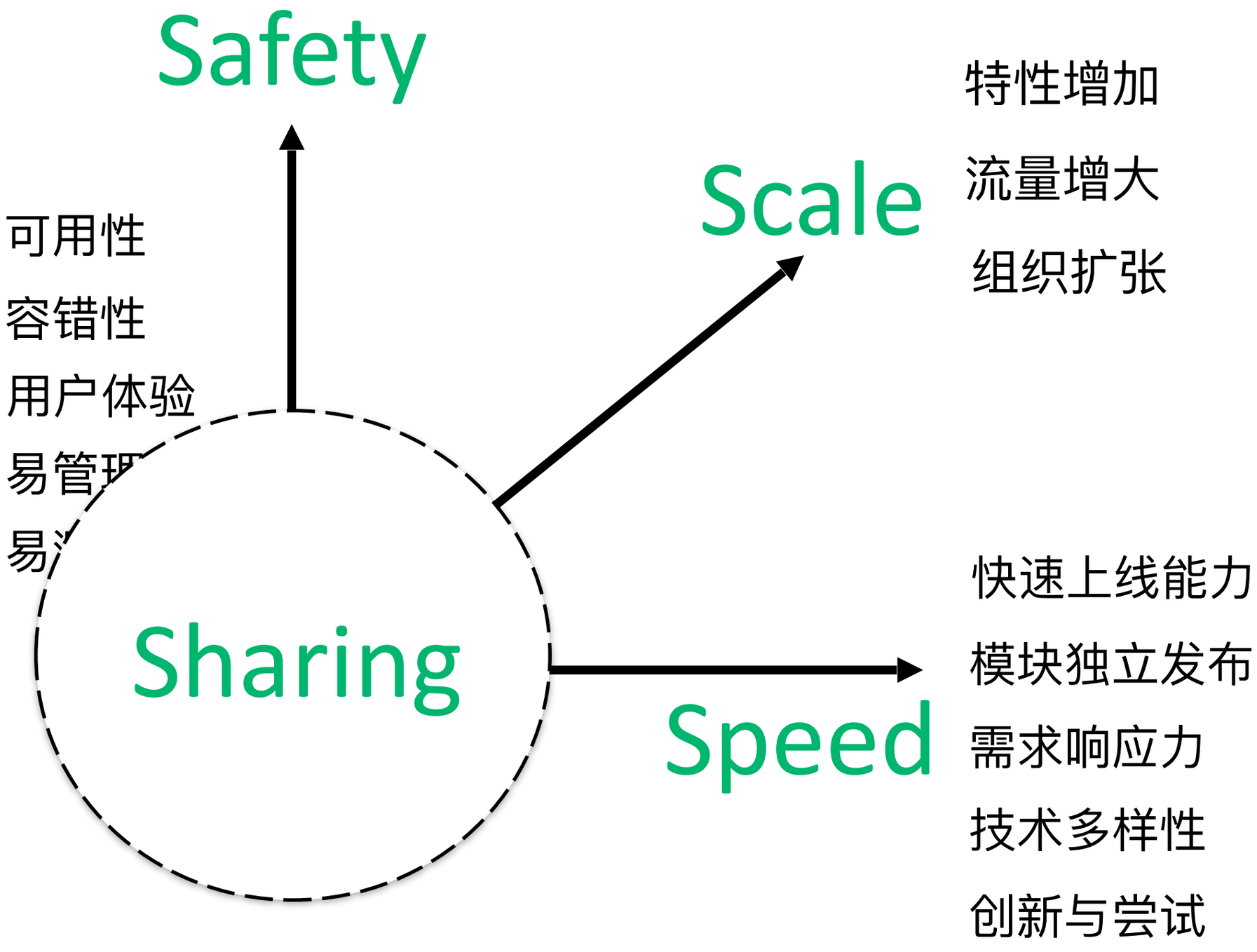
需求快速响应

不断创新与尝试

独立发布

技术多样性


Safety
可用性
容错性
用户体验
易管理性
易测试性
Scale
特性增加
流量增大
组织扩张
Speed
快速上线能力
模块独立发布
需求响应力
技术多样性
创新与尝试


Safety
可用性
容错性
用户体验
易管理
易运
Scale
特性增加
流量增大
组织扩张
Sharing
Speed
快速上线能力
模块独立发布
需求响应力
技术多样性
创新与尝试

什么是微服务架构

微服务架构

Microservices - the new architectural style

Martin Fowler, Mar 2014

微服务架构是一种架构模式，它提倡将单一应用程序划分成**一组小的服务**，服务之间互相协调、互相配合，为用户提供最终价值。

每个服务运行在其**独立的进程中**，服务与服务间采用**轻量级的通信机制**互相协作（通常是基于HTTP协议的RESTful API）。

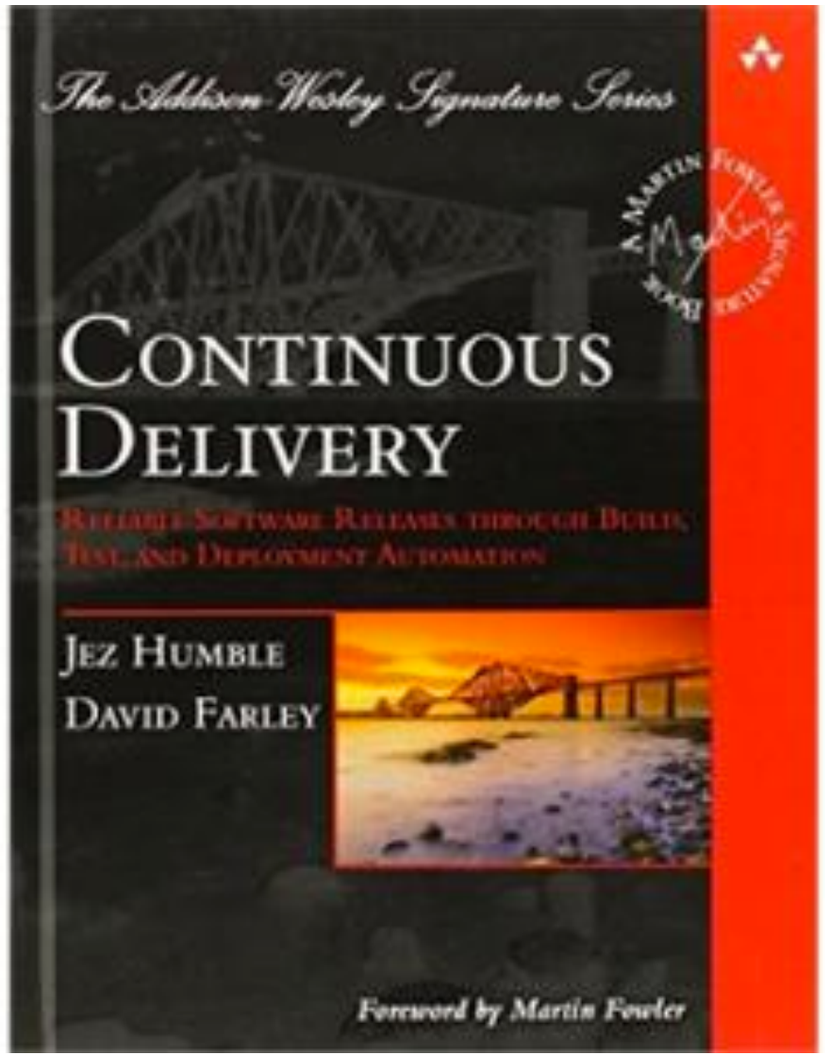
每个服务都围绕着具体业务进行构建，并且能够**被独立的部署**到生产环境、类生产环境等。

以缩短缩短交付周期为核心
基于DevOps
的演进式架构

A thousand Hamlets in a thousand people’s eyes.

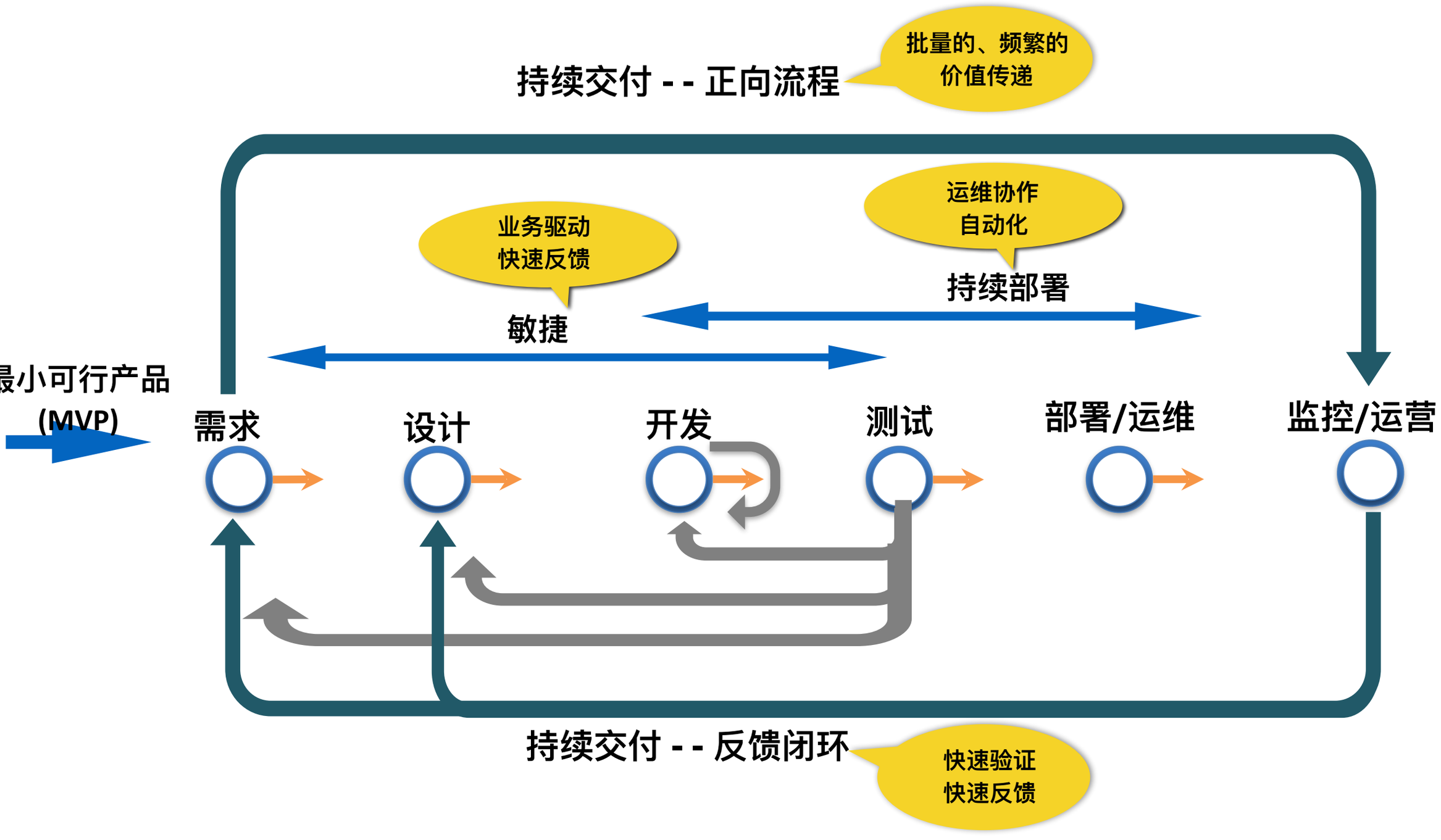
Schakespear

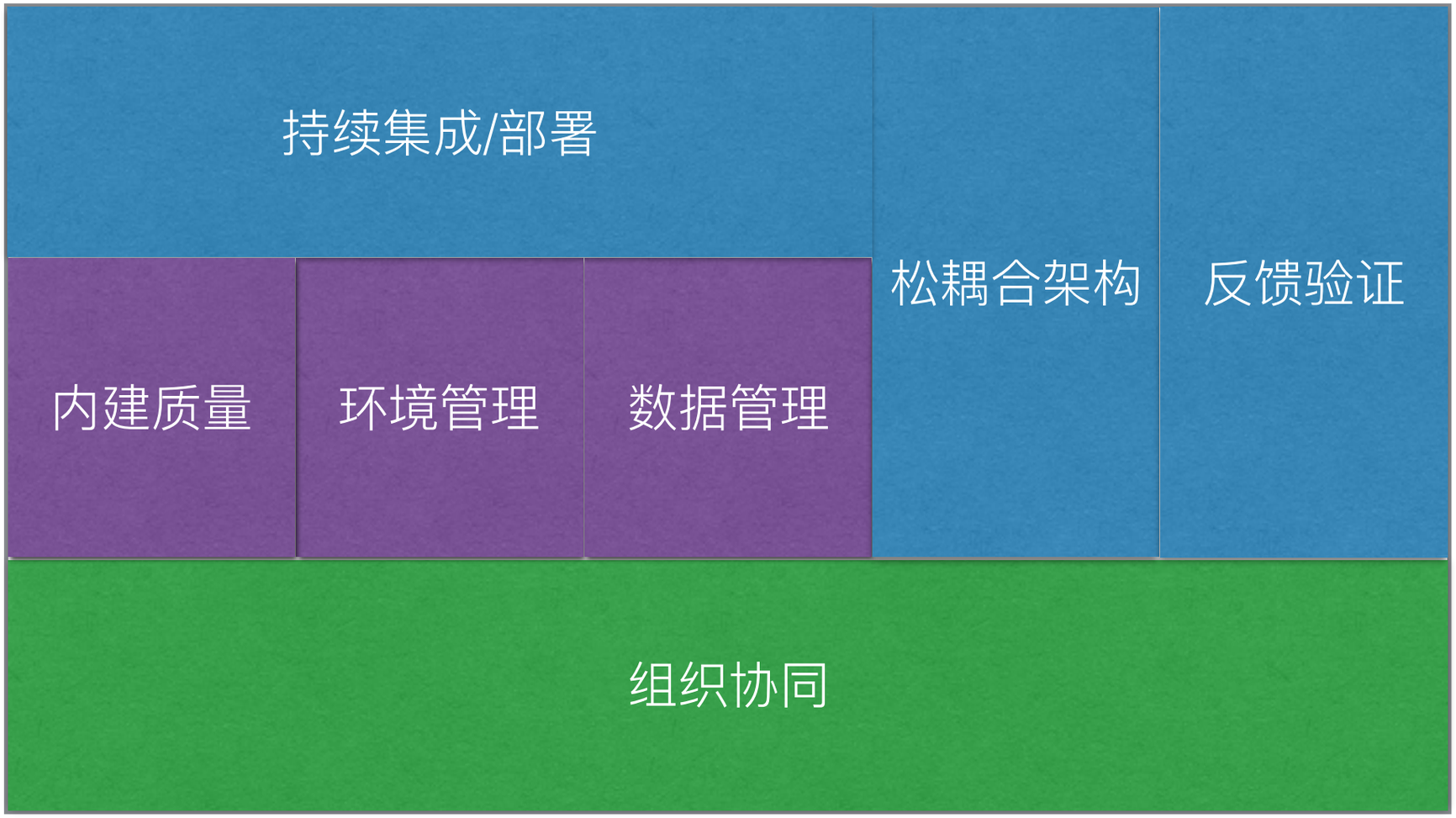
为什么以缩短交付周期为核心?



- *Faster time to market*
- *Lower risk release*
- *Higher quality*
- *Lower costs*
- *Better products*

https://www.continuousdelivery.com/

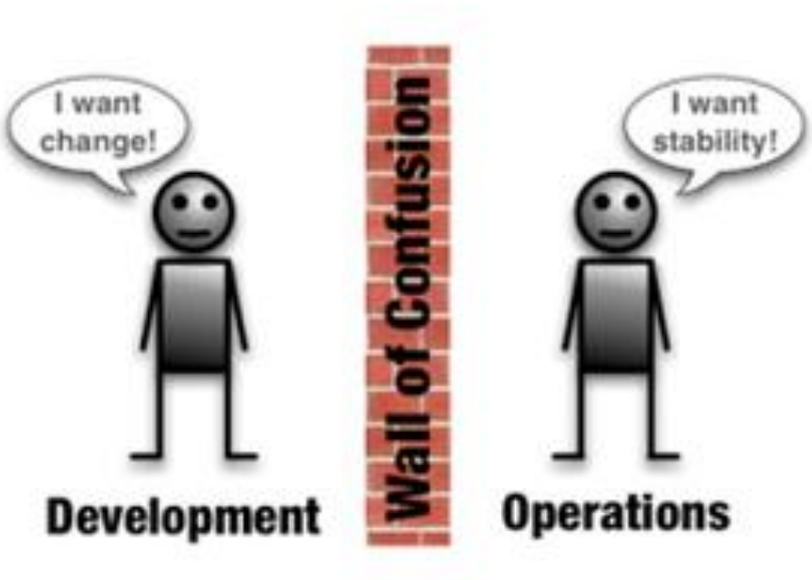

持续交付 - - 正向流程
批量的、频繁的
价值传递
业务驱动
快速反馈
敏捷
运维协作
自动化
持续部署
最小可行产品
(MVP)
需求
设计
开发
测试
部署/运维
监控/运营
持续交付 - - 反馈闭环
快速验证
快速反馈



微服务架构是持续交付体系中
松耦合架构的一种实现

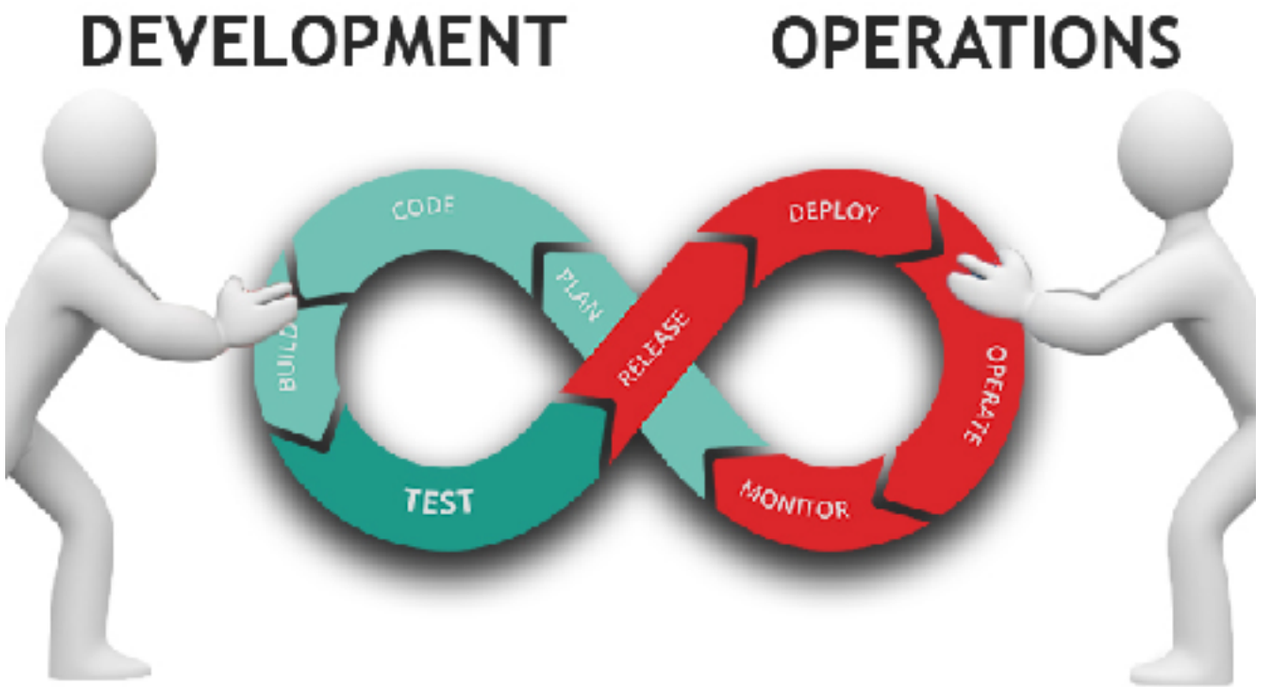
为什么基于DevOps?

- “耐心”不足
- 忽略“真实世界”



- 注重“稳”
- 对“变化”很谨慎

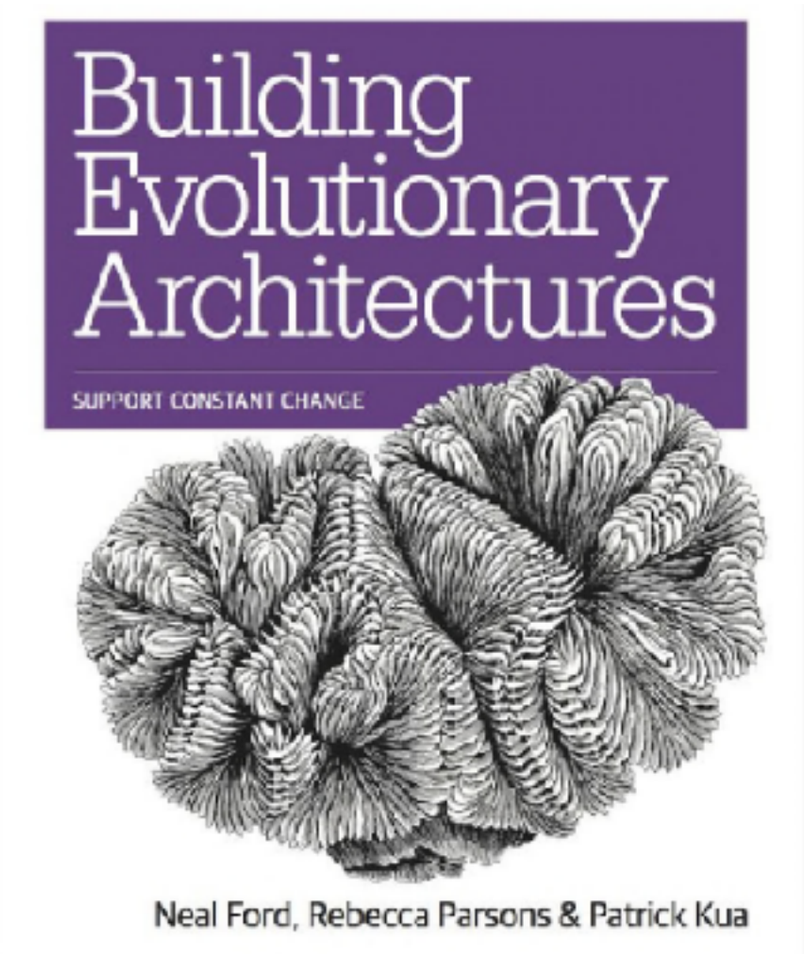
为什么基于DevOps?



- **C**ommunication
- **A**utomation
- **L**ean
- **M**easuring
- **S**haring

https://www.supinfo.com/articles/single/3652-what-is-devops

什么是演进式架构?



《演进式架构》
O’Reilly 2017 11

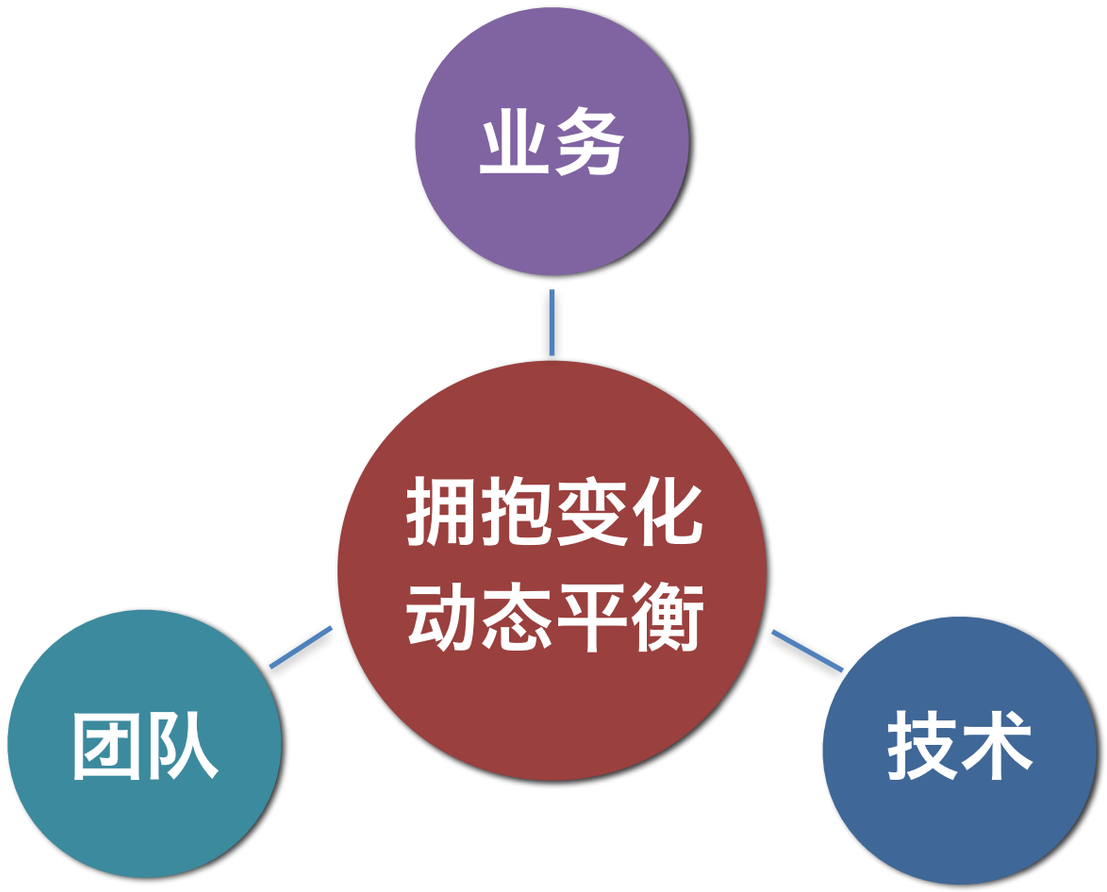
什么是演进式架构?

- 支持增量式变更作为第一原则

- 持续的动态演进
- 运维意识是关键
- 痛苦的事情提前做

演进式架构的核心

- **动态演进**

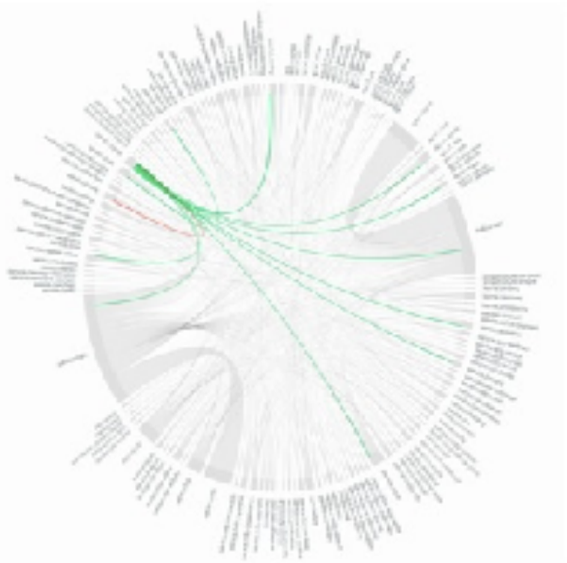


基于业务、技术和团队的**动态平衡与演进**

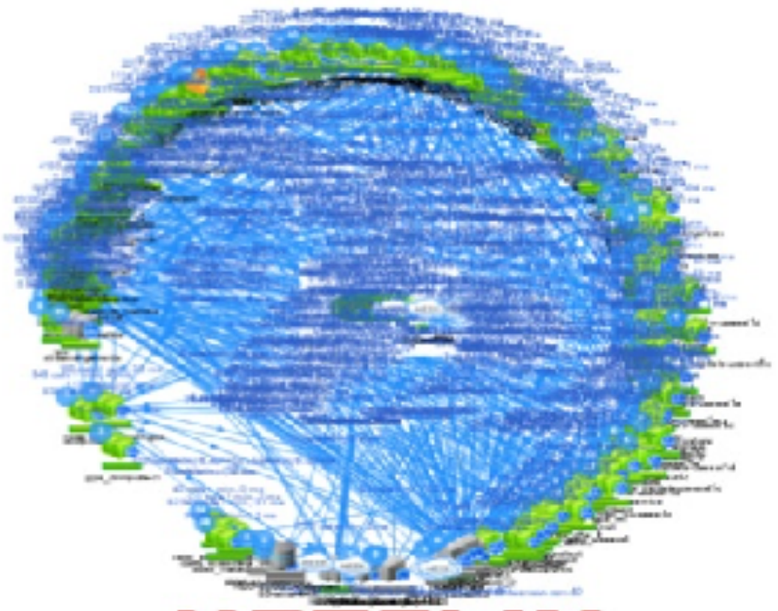
演进式架构的核心

- **架构师的运维意识**
  - 架构只是抽象，直到**真正上线**并投入运维**产生价值**
  - 软件世界不断的变化，而架构只是**演进过程的快照**

450 microservices



500+ microservices



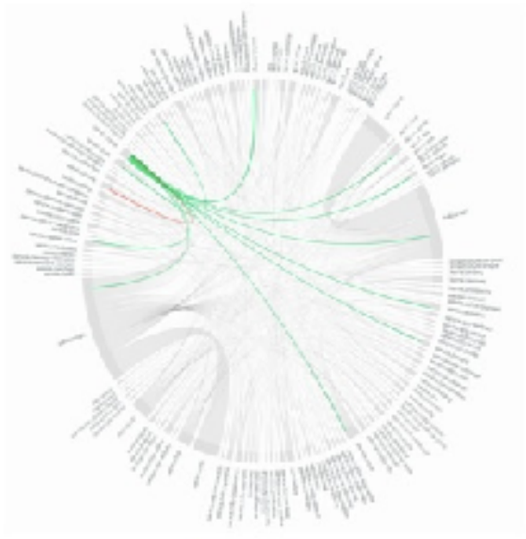
500+ microservices

演进式架构的核心

- **痛苦的事提前做**
  - **不断**识别问题并用**自动化**的手段消除痛苦
  - 持续集成、持续部署、基础设施即代码

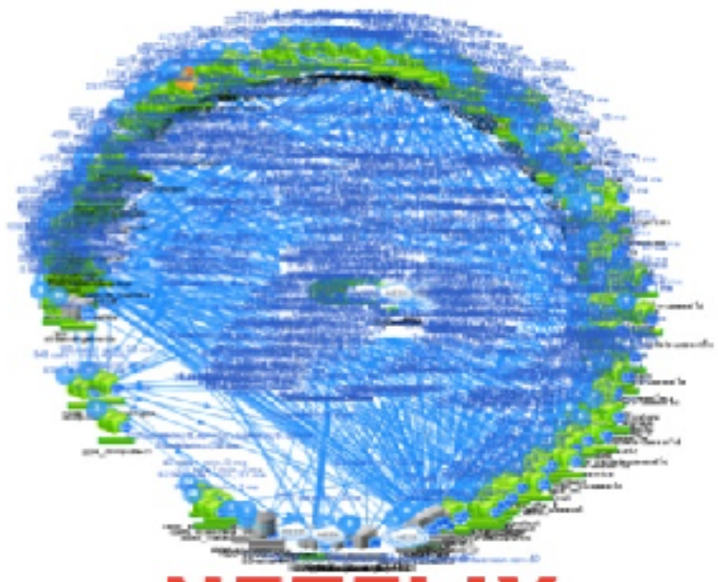
微服务架构是一种演进式架构

450 microservices



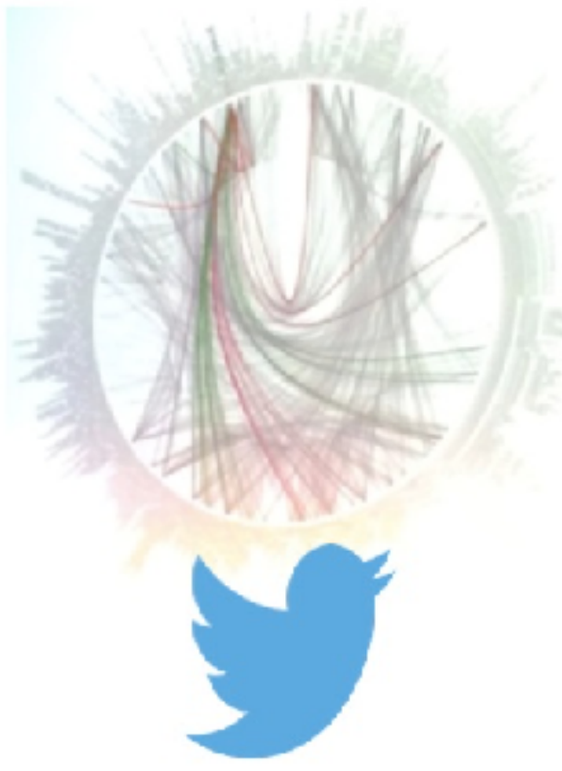
HAIL O

500+ microservices



NETFLIX

500+ microservices



Source:
Netflix: http://www.slideshare.net/BruceWong3/the-case-for-chaos
Twitter: https://twitter.com/adrianco/status/441883572618948608
Hail-o: https://sudo.hailoapp.com/services/2015/03/09/journey-into-a-microservice-world-part-3/

以缩短**交付周期**为核心

基于**DevOps**

构建的**演进式架构**

I am not perfect in my walk but I want to do the right thing.

Kirk Cameron

- 微服务架构的核心
- 微服务架构生态系统
- 微服务实践应用案例

为什么需要生态系统?

系统化的工程

- 分布式系统
- 服务的治理与维护
- 测试策略与自动化
- 持续交付流水线

框架层出不穷

- API网关
- 服务开发框架
- 测试验证框架
- 部署运维工具

多维度互相依赖

- 基础设施(私有云/公有云)
- 持续集成/持续部署流水线
- 团队的敏捷/工程化实践
- 端到端的工具链

多维度相互依赖，互相制约

微服务生态系统

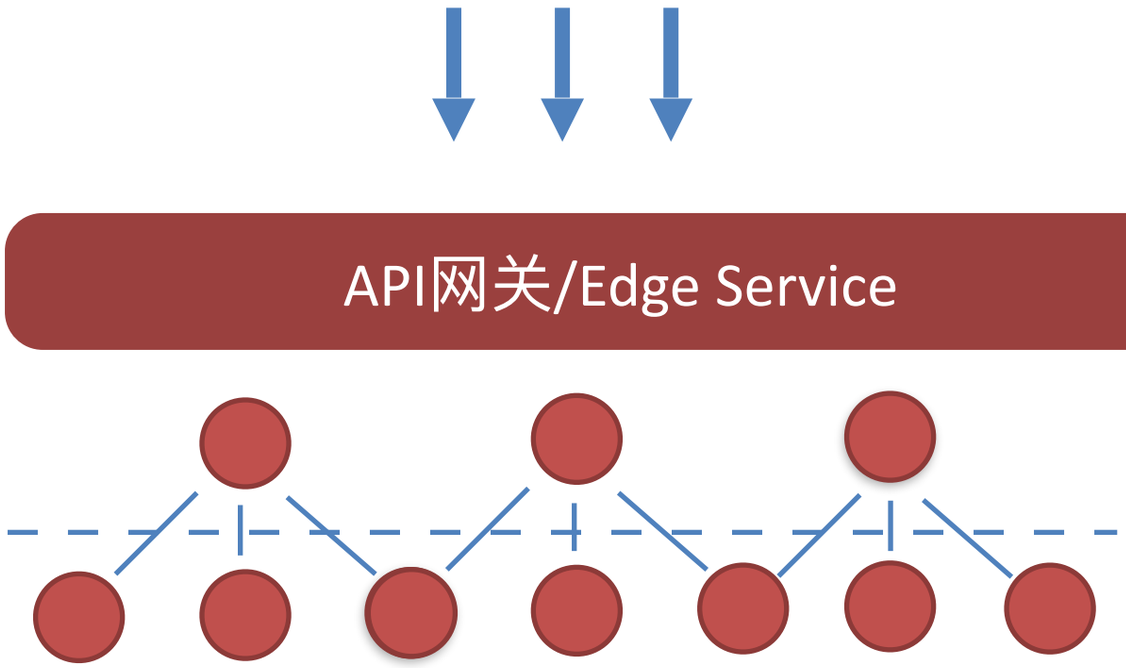
接入层

业务层

- 聚合服务
- 基础服务

支撑层

基础设施



交付流水线与工程实践

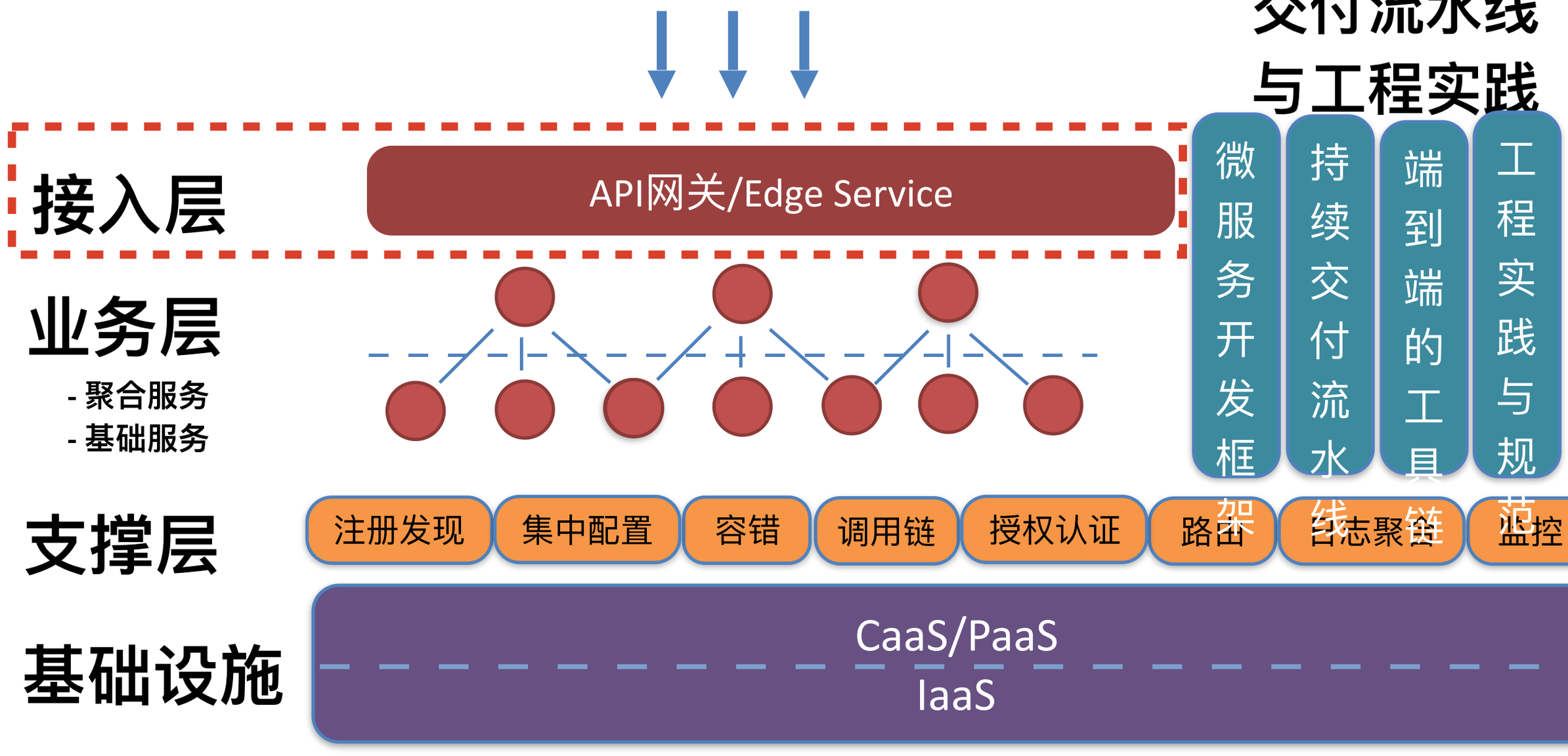
- 微服务开发框
- 持续交付流水
- 端到端的工具
- 工程实践与规

注册发现 | 集中配置 | 容错 | 调用链 | 授权认证 | 路由 | 日志聚合 | 监控

CaaS/PaaS

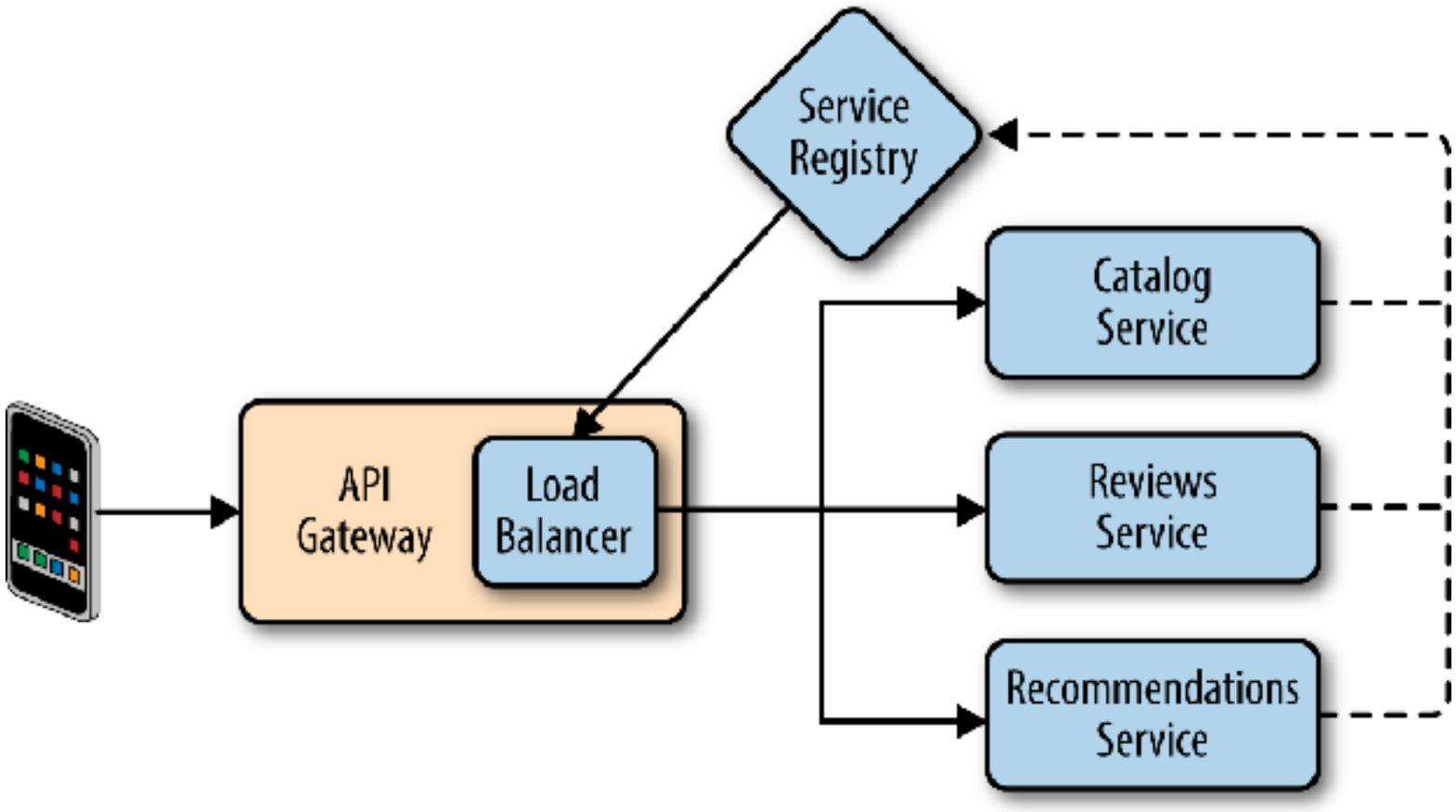
IaaS

微服务生态系统

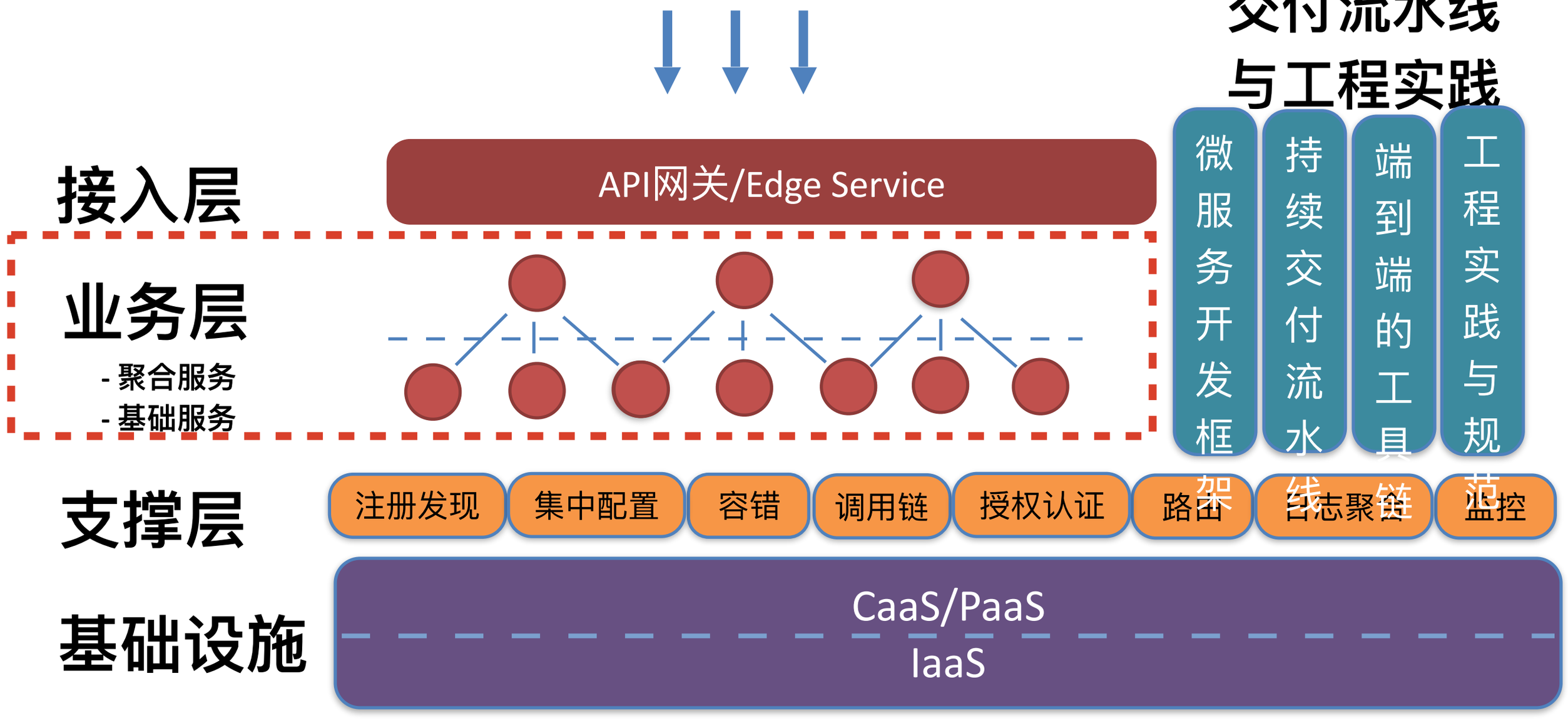

交付流水线
与工程实践
接入层
API网关/Edge Service
业务层
- 聚合服务
- 基础服务
支撑层
注册发现
集中配置
容错
调用链
授权认证
路由
日志聚合
监控
基础设施
CaaS/PaaS
IaaS
微服务开发框架
持续交付流水线
端到端的工具链
工程实践与规范

API网关

- 集中接口，封装路由
- 协议转换
- 流量限制
- 调用统计
- 安全认证

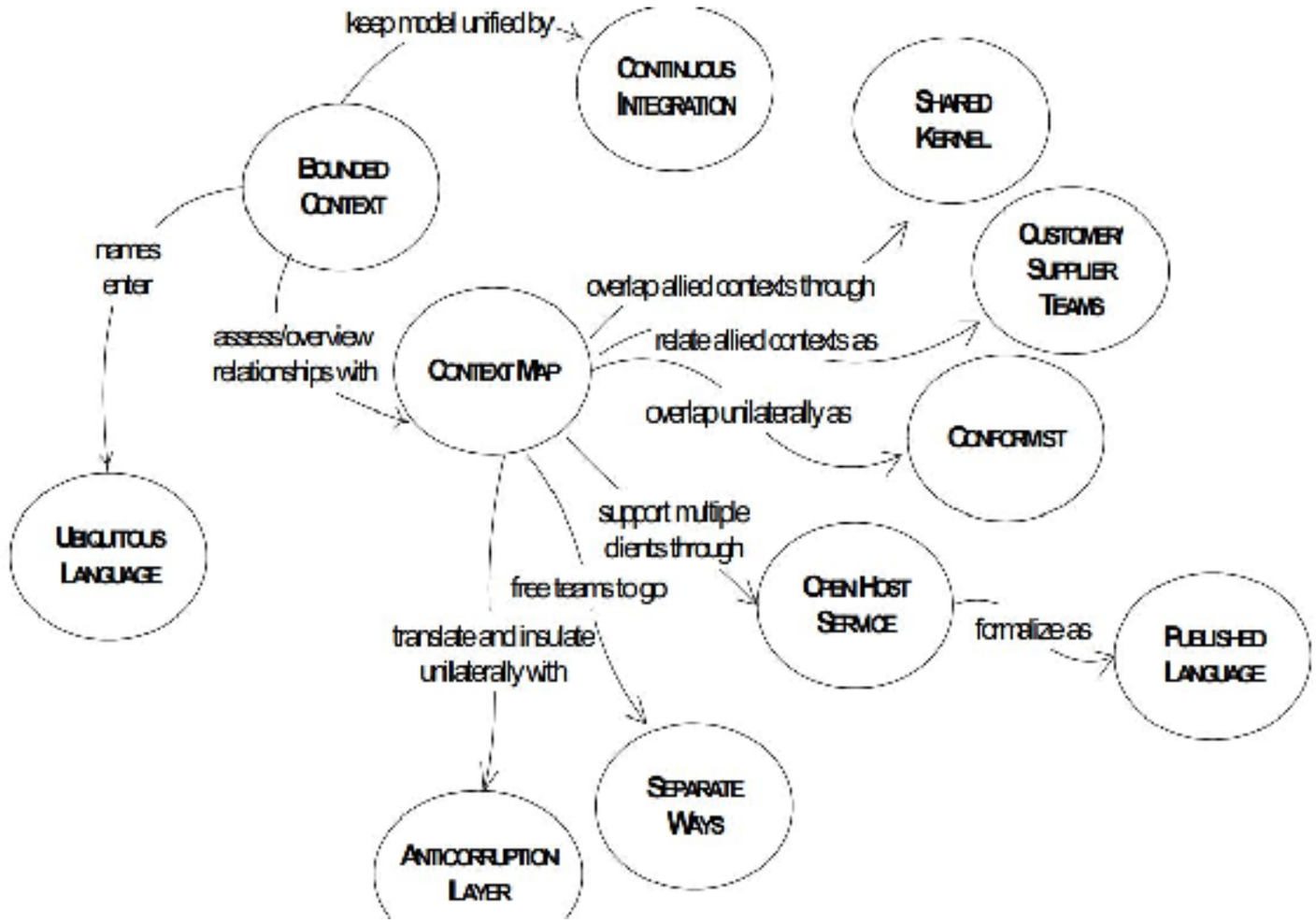


微服务生态系统

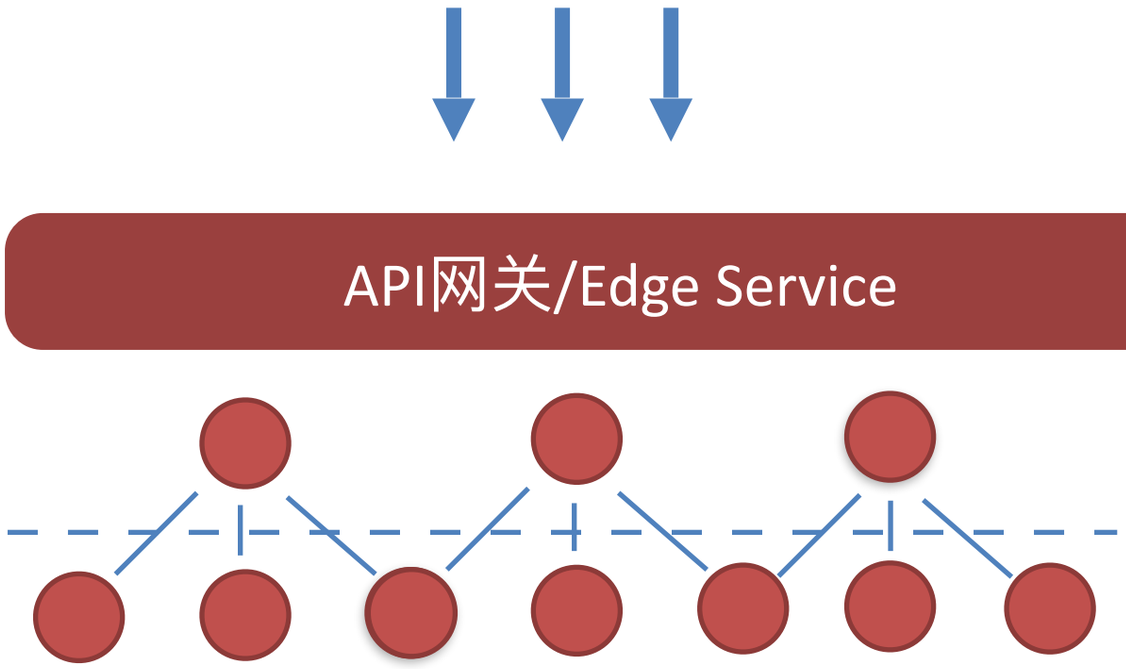

接入层
API网关/Edge Service
业务层
- 聚合服务
- 基础服务
支撑层
注册发现
集中配置
容错
调用链
授权认证
路由
日志聚合
监控
基础设施
CaaS/PaaS
IaaS
交付流水线与工程实践
微服务开发框架
持续交付流水线
端到端的工具链
工程实践与规范

服务设计(拆分)

- 面向对象分析
- 可重用的逻辑
- 资源密集型的业务
- 领域驱动设计



微服务生态系统



接入层

业务层

- 聚合服务
- 基础服务

交付流水线与工程实践

- 微服务开发框架
- 持续交付流水线
- 端到端的工具链
- 工程实践与规范

支撑层

注册发现 | 集中配置 | 容错 | 调用链 | 授权认证 | 路由 | 日志聚合 | 监控

基础设施

CaaS/PaaS

IaaS

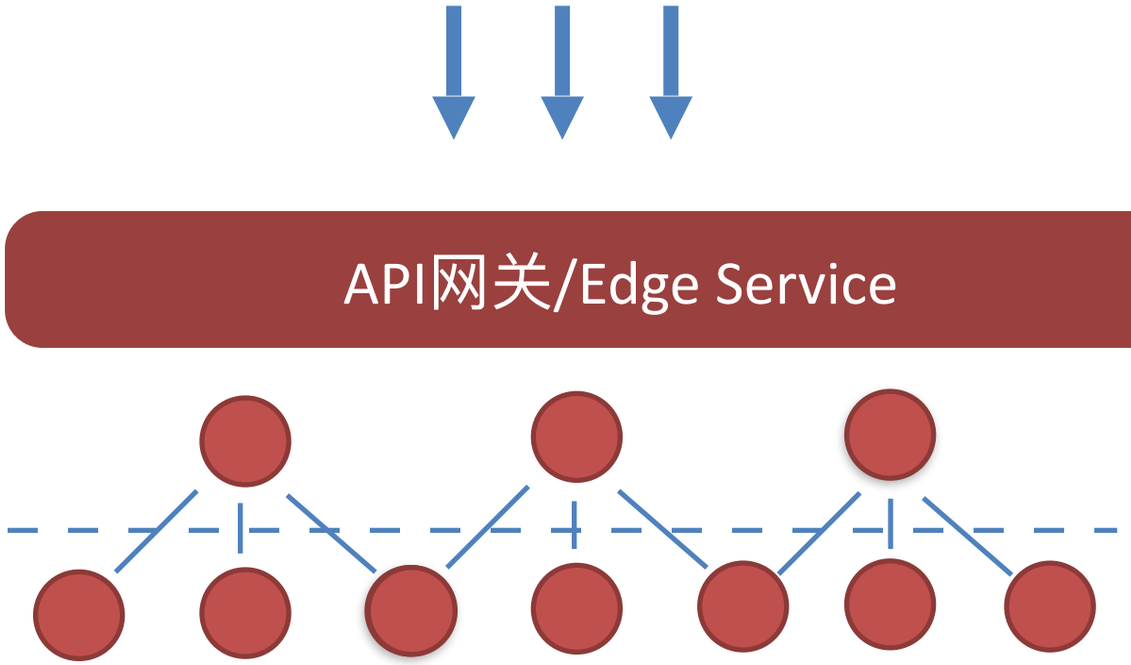
微服务生态系统

接入层

API网关/Edge Service

业务层

- 聚合服务
- 基础服务



支撑层

注册发现 | 集中配置 | 容错 | 调用链 | 授权认证 | 路由 | 日志聚合 | 监控

基础设施

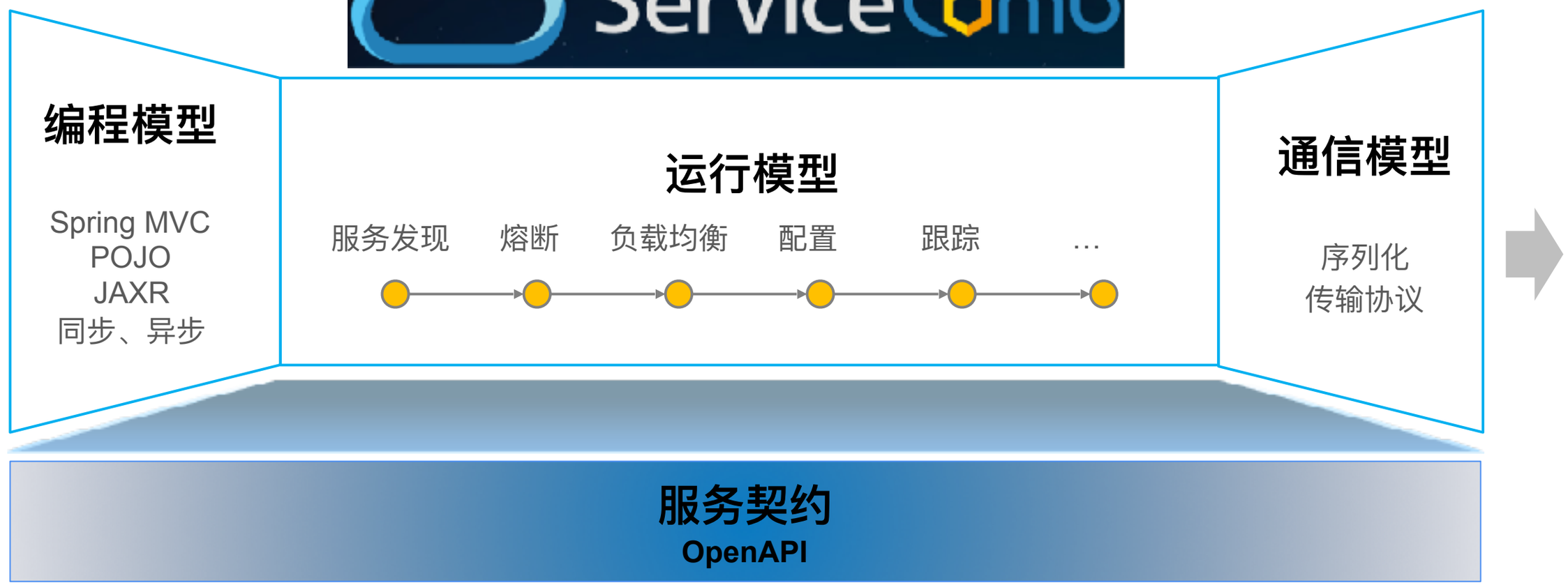
CaaS/PaaS

IaaS

交付流水线与工程实践

- 微服务开发框
- 持续交付流水
- 端到端的工具
- 工程实践与规

微服务开发框架



编程模型：支持Spring MVC、POJO、JAXR以及异步的方式

运行模型：服务发现、熔断、负载均衡、集中配置、动态调用链

通信模型：支持REST、RPC的传输机制

服务契约：基于OpenAPI，定义服务间契约

- 微服务架构的核心
- 微服务架构生态系统
- 微服务实践应用案例

XXX云背景介绍

- ✓ **客户：** 一线 MKT/行销人员
- ✓ **定位：** “以营促销”转型，支撑精准“看病” 快速“开方”
- ✓ **现状**：70W+代码，团队50+ 发布周期2~3个月

  修改需要重新构建整个应用

  测试时间长，部署时间上，代码耦合度高

  新员工上手时间长，维护成本高

期望交付速度2~3周，支撑项目创新

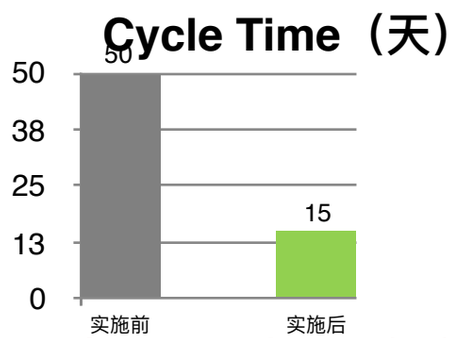
-

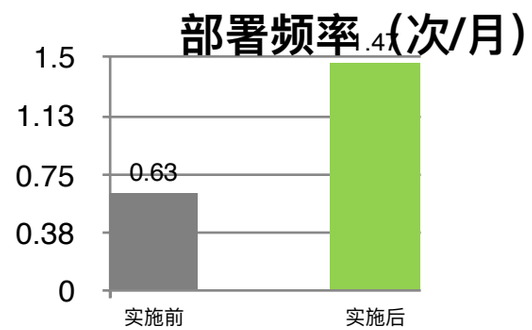
应用一系列实践

3个月后.....

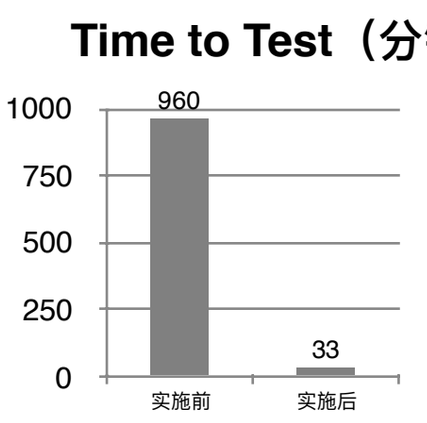
应用实践效果

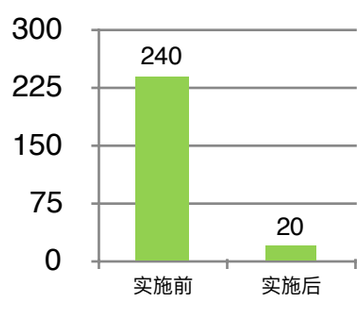
一、结果类度量是核心指标，反映了服务实施过程中端到端的交付效率

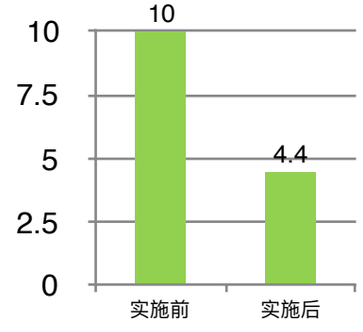


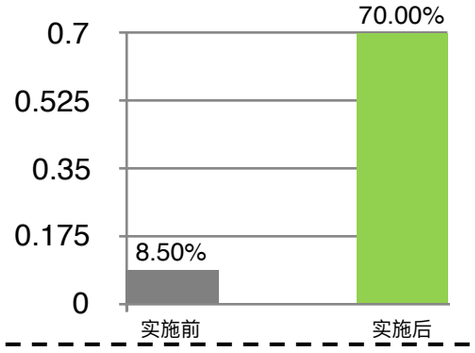


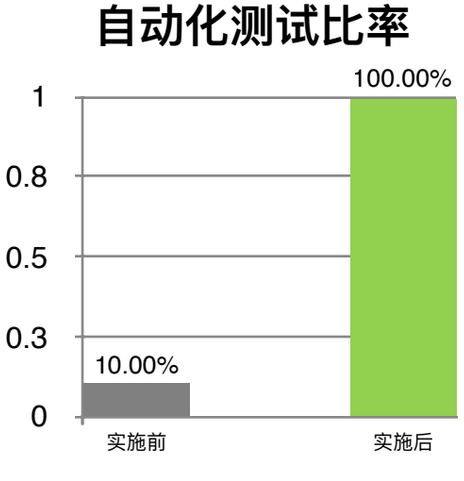
二、过程类度量指标是可选项，反映了服务实施过程中局部的优化效率

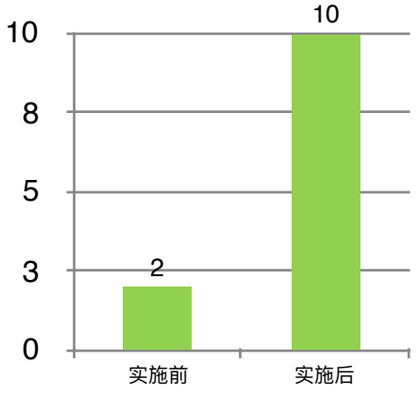


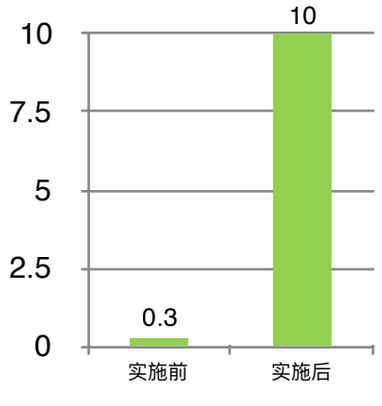


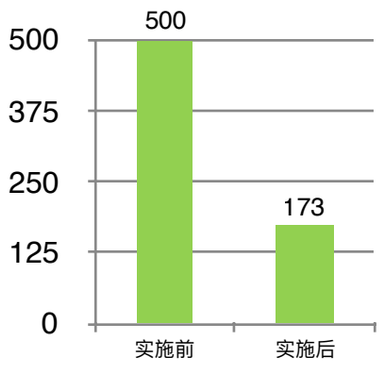






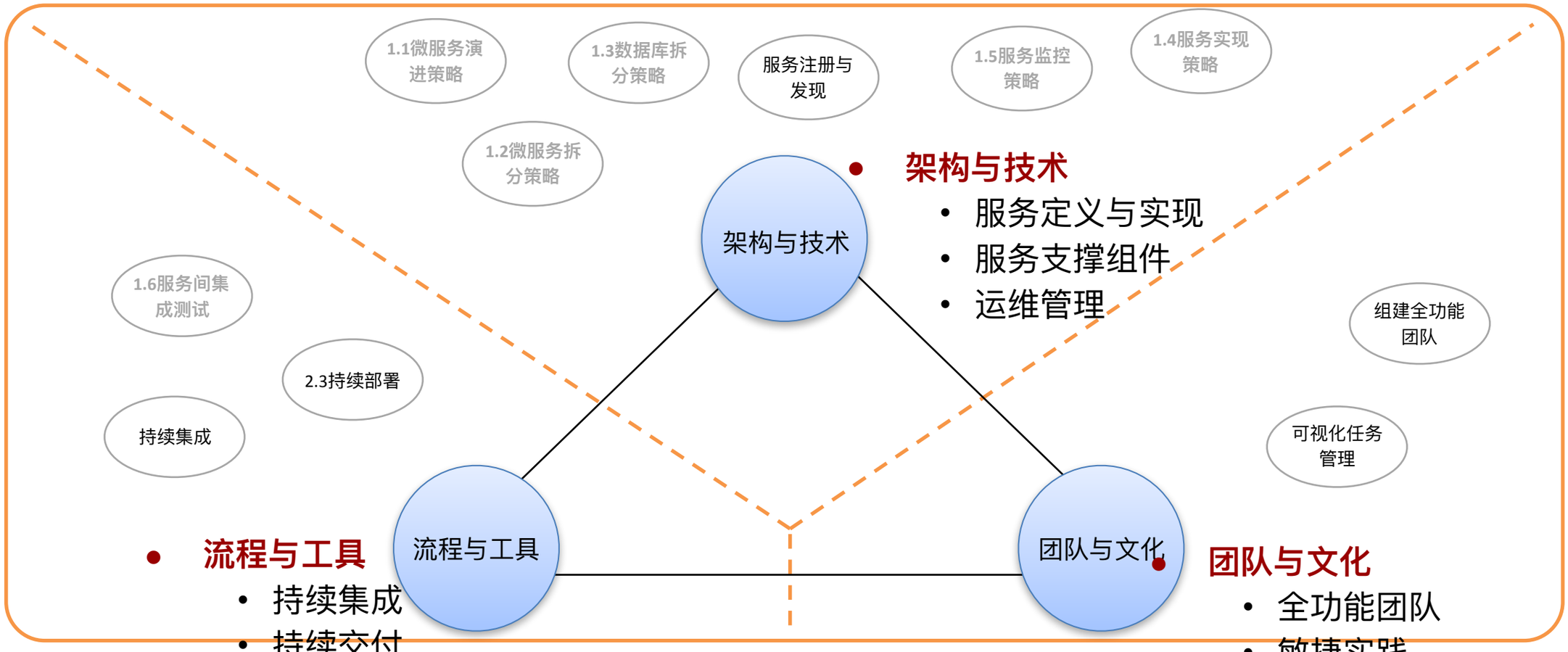






落地20+项实践，试点服务的交付周期从50天降低到14天。

从3个维度进行服务化架构演进



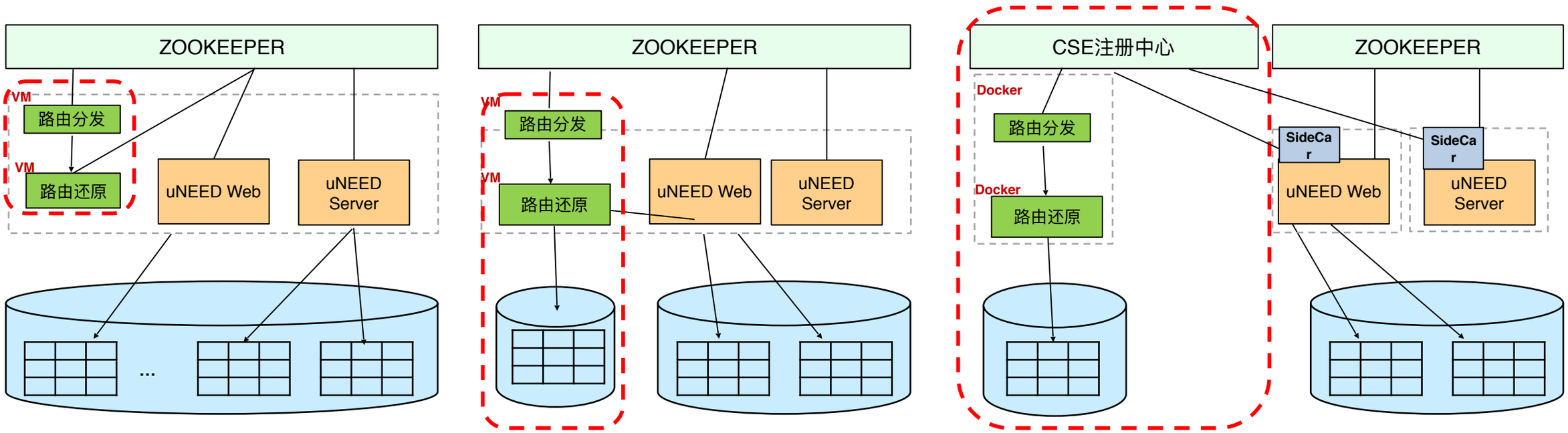
实践 1: 如何演进?

如何由一个单体架构逐步演进为全微服务架构？--服务演进策略：**双模IT，新老系统共存，抽取服务/创建新服务模式。**

1、拆服务：对服务进行拆分，数据库保持不变。

2、拆数据：服务管控自己数据的全生命周期。

3、架构优化：引入微服务框架及Docker容器。

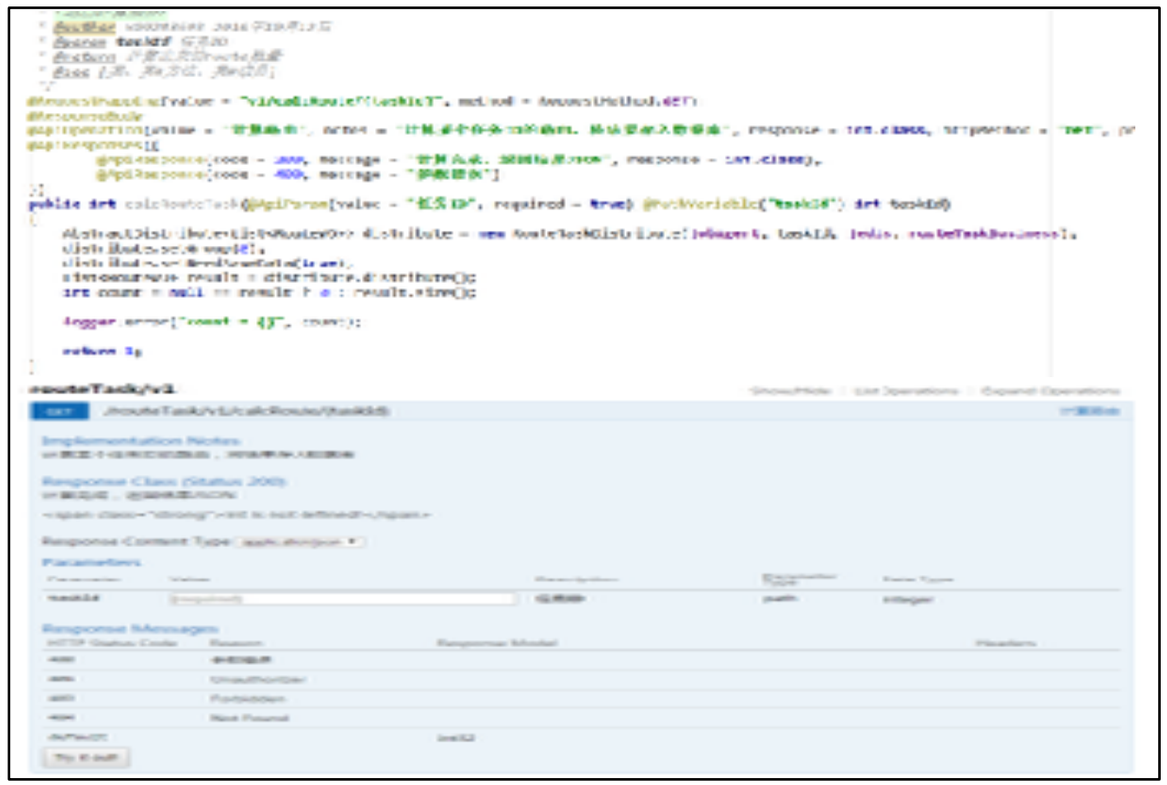


实践 3: 如何标准化?

使用**服务自描述文档（One Page ReadME）**对微服务进行全面描述。
通过**Swagger**对接口进行描述，使用SwaggerUI生成接口描述静态页面。

1、完整的服务自描述文档

2、使用标签标注接口信息并使用静态页面展示接口信息

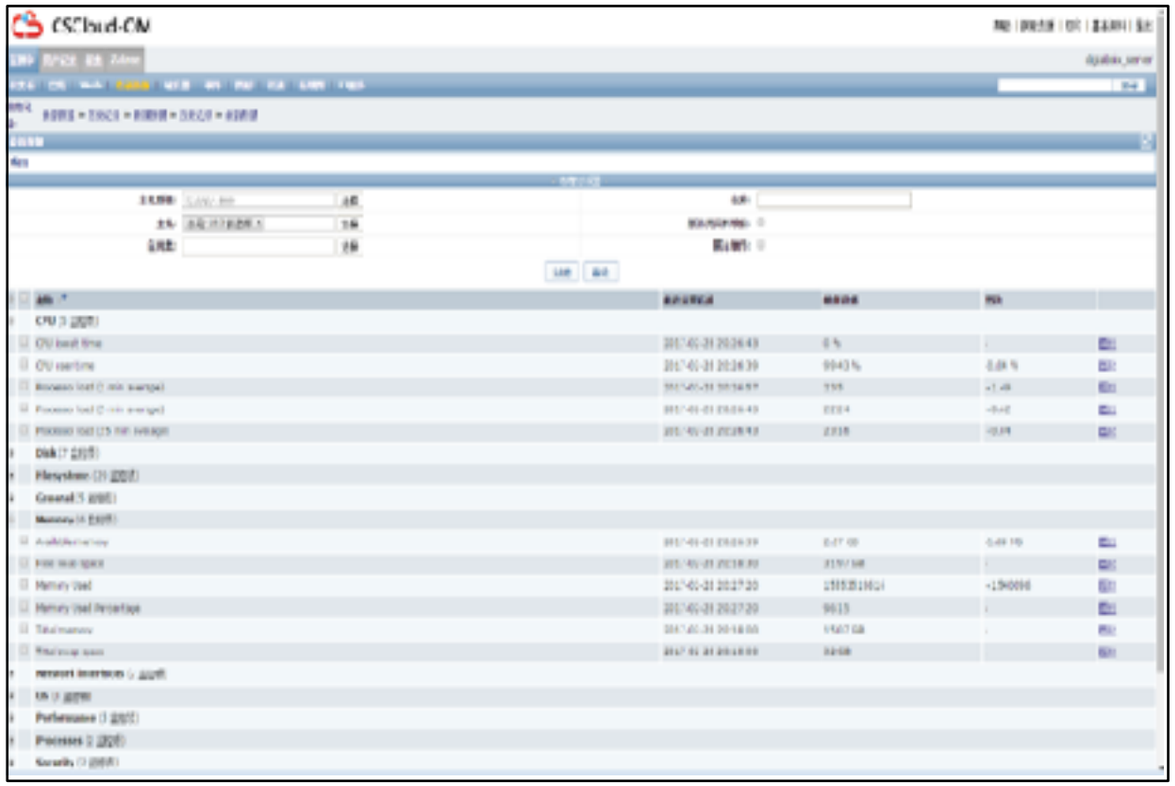


优点：帮助团队直观、快速了解服务功能、依赖、环境、测试、CI地址等。

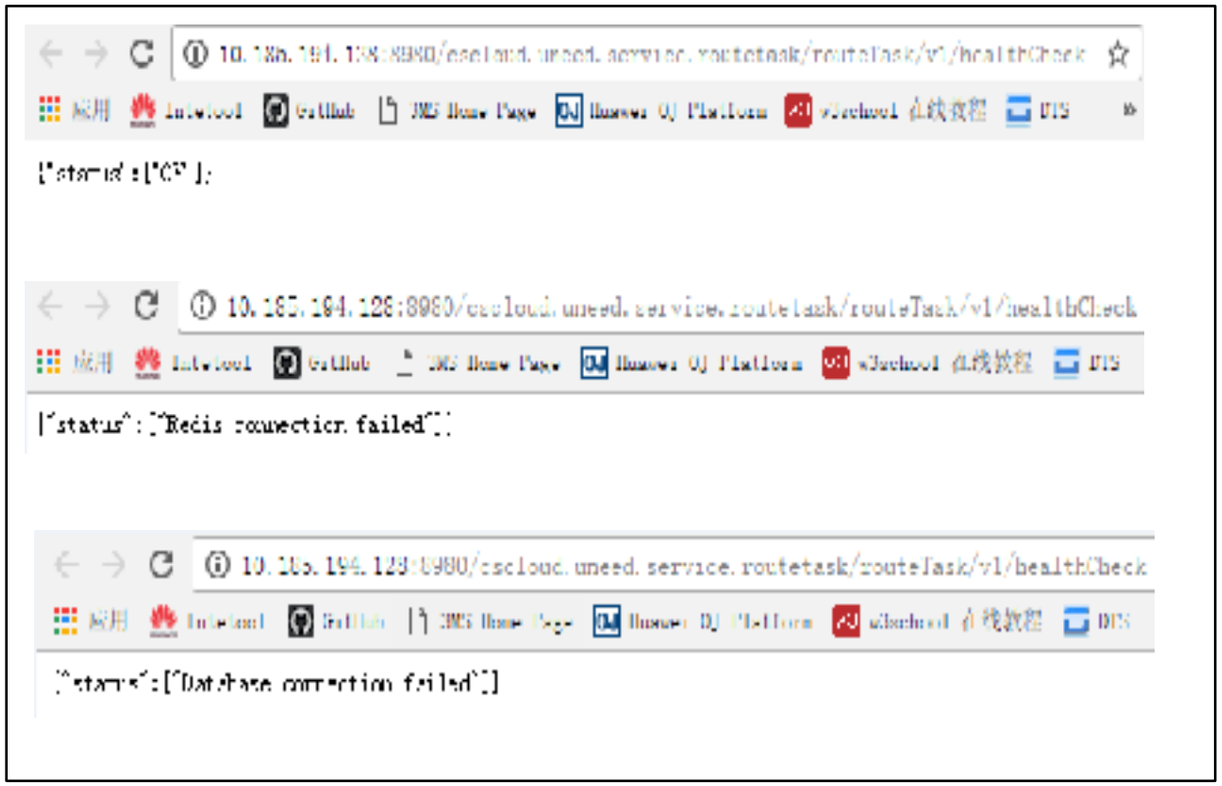
实践 4: 如何监控?

如何第一时间知道服务的运行状态？--使用Zabbix进行虚拟机**系统资源监控**，在微服务中增加服务**健康性检查**接口。

1、基于Zabbix监控虚拟机资源，包括CPU、内存等。



2、在微服务中增减健康性检查接口，检查该微服务依赖的DB、Cache等资源是否准备就绪。

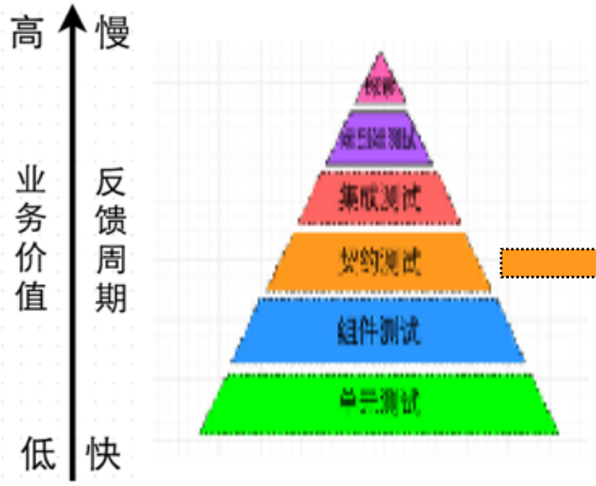


优点：You build it，You run it. 团队第一时间能够监控服务并处理异常。

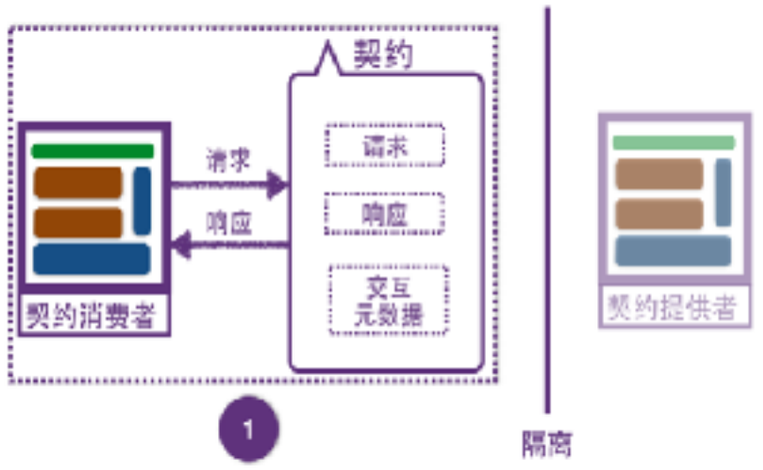
实践 5： 如何测试?

如何测试服务间的调用是否正确?
--**契约测试**属于集成测试的一种，它提供了一种机制，帮助**尽早验证**消费者和提供者在协作过程中**接口是否发生变化**。

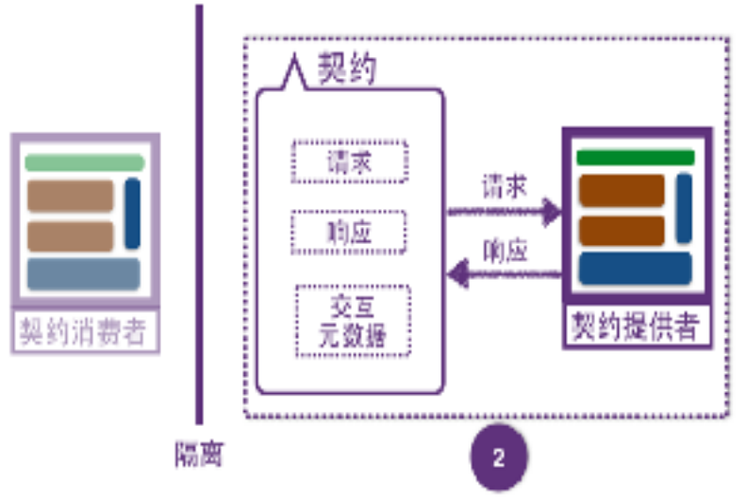
优化测试策略



1、服务消费者定义请求体/响应体，生成契约文件

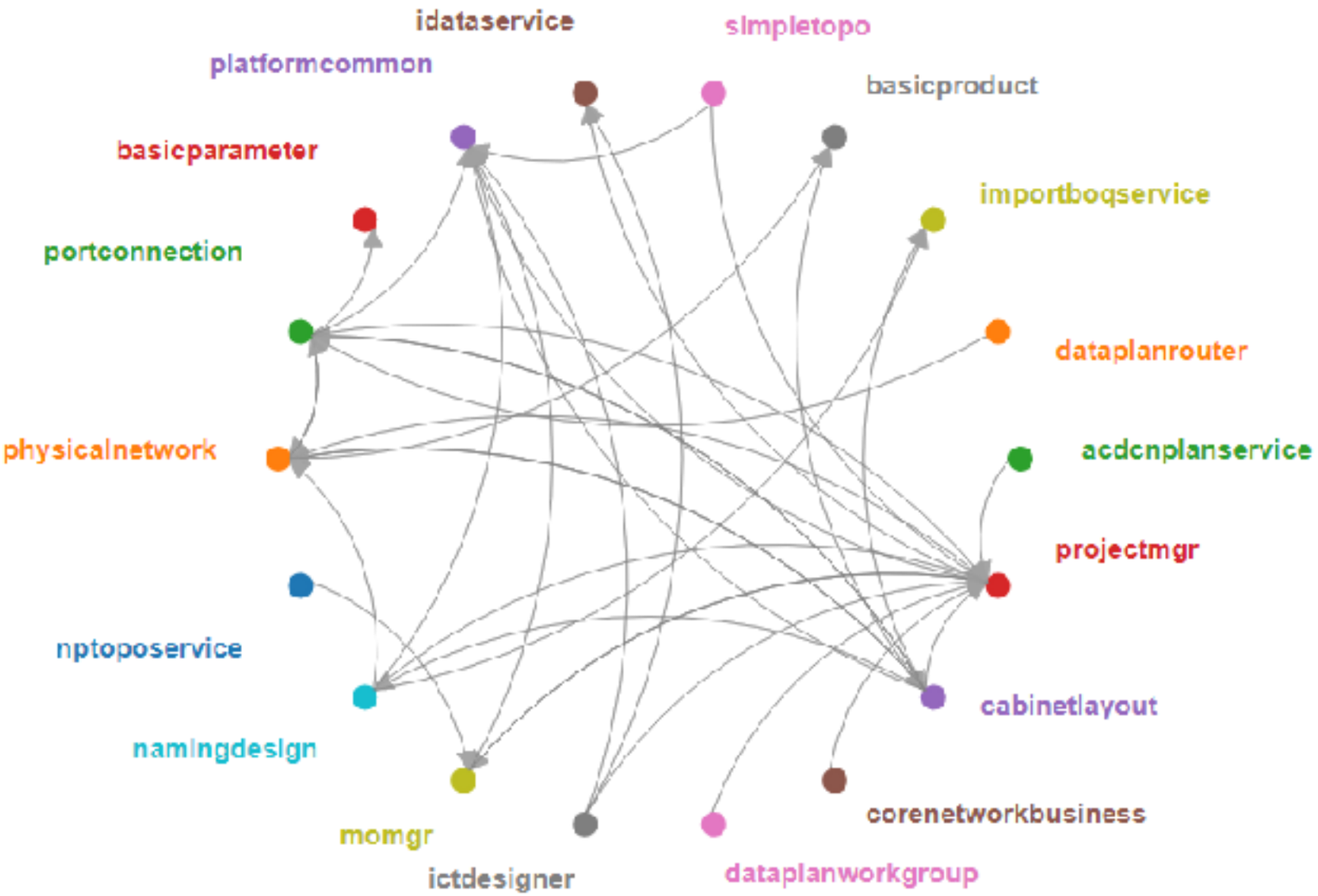


2、服务提供者按照契约中所定义的用例进行测试，验证所提供的服务是否满足需求



实践 5：消费者驱动的契约（PACT）测试

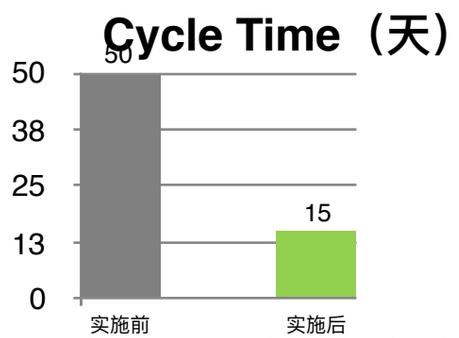
基于契约生成的服务依赖图

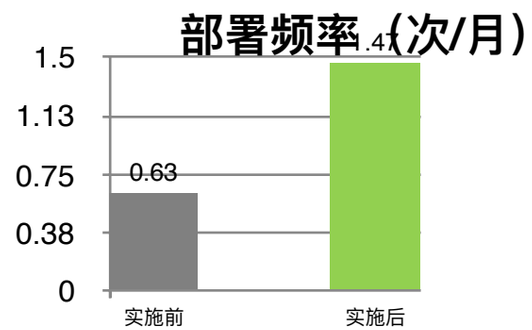


http://pact.doczh.cn/

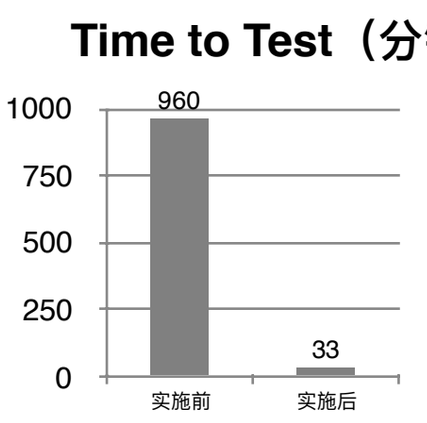
应用实践效果

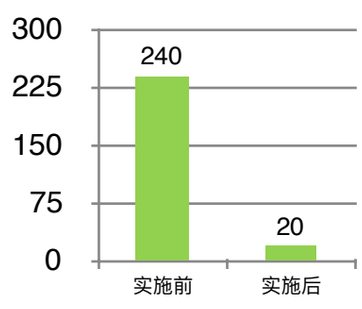
一、结果类度量是核心指标，反映了服务实施过程中端到端的交付效率

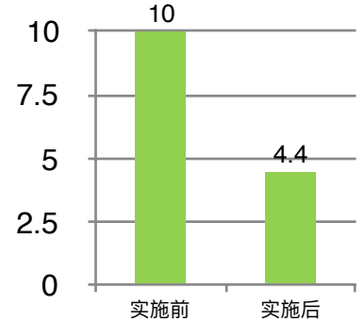


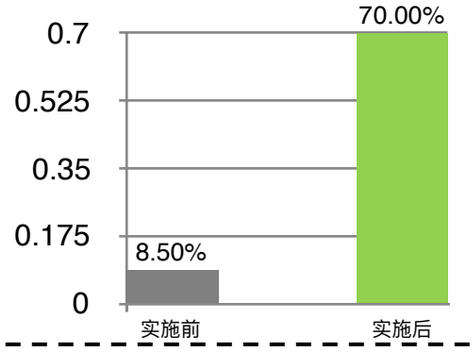


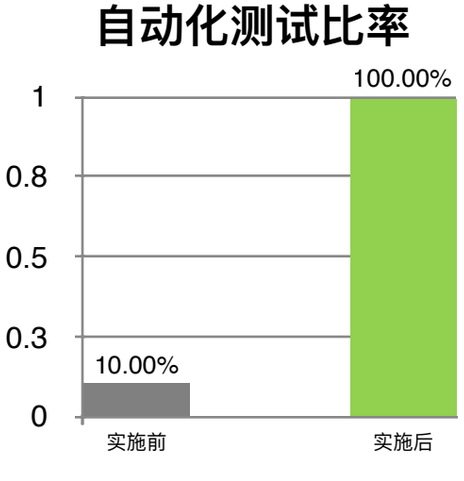
二、过程类度量指标是可选项，反映了服务实施过程中局部的优化效率

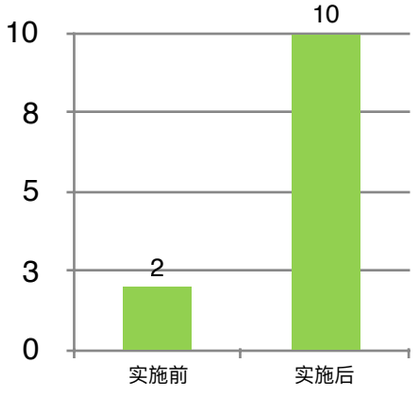


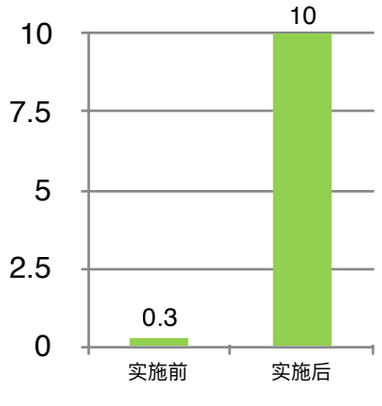


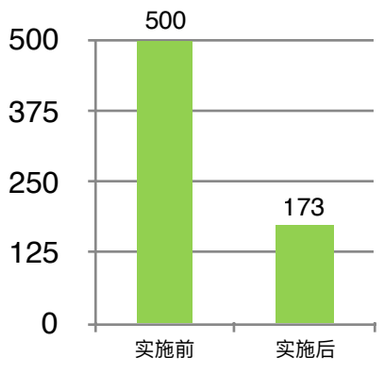










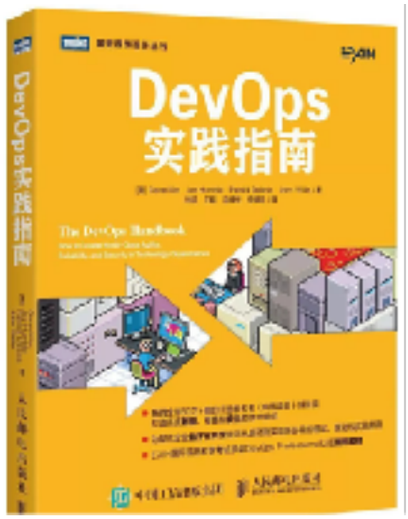


落地20+项实践，试点服务的交付周期从50天降低到14天。

谢谢

推荐书籍


持续交付
发布可靠软件的系统方法


DevOps
实践指南


Release It!
Design and Deploy
Production-Ready Software
Michael T. Nygard


O'REILLY
Building
Microservices
Sam Newman


微服务
架构与实践