LINUXCON
containercon
CLOUDOPEN
CHINA 2017
HUAWEI

微服务的设计原则与生态系统

王磊

关于我

华为公司技术专家

ThoughtWorks首席咨询师

Sybase Tech Leader

- 《微服务架构与实践》作者
- 《DevOps Handbook》中文译者之一
- 国内较早倡导和实践微服务的先行者
- 对于自动化测试、持续交付、DevOps有丰富的实践经验
- 西安DevOps Meetup 联合发起人

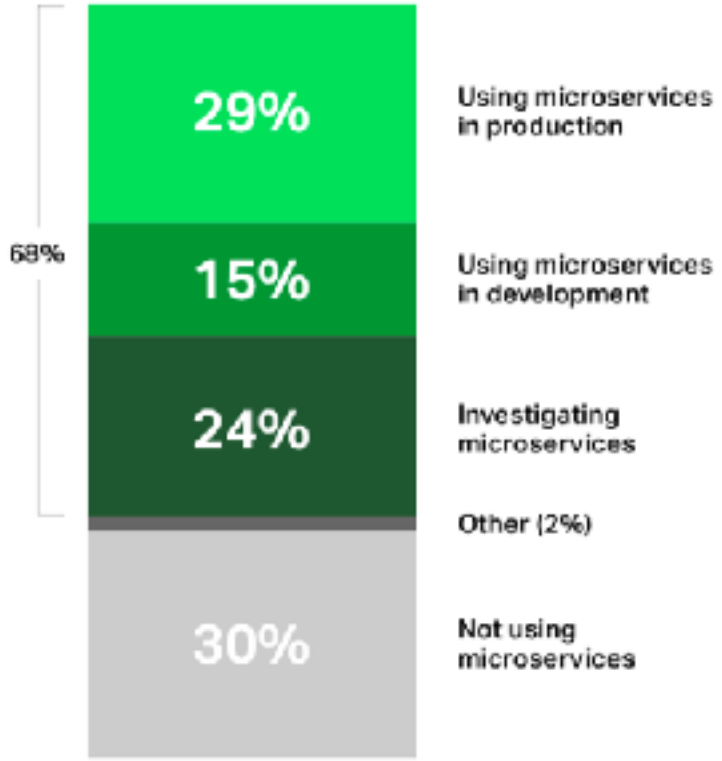
议题

- 微服务架构的核心
- 微服务架构设计原则
- 微服务架构生态系统

Are you using microservices?

Microservices are entering mainstream

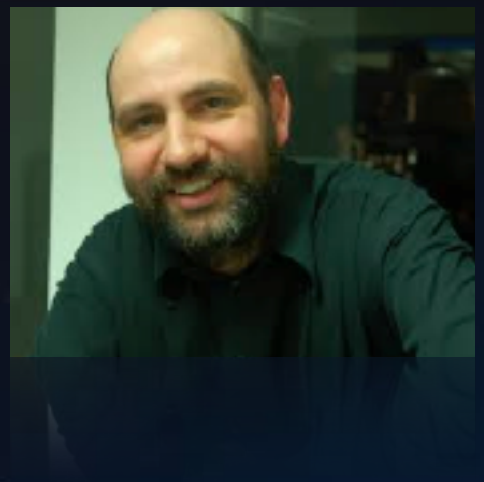
68% of organizations are using or investigating microservices



SURVEY QUESTION Which statement "best" defines how your organization is currently using microservices?

https://www.nginx.com/resources/library/app-dev-survey/

什么是微服务架构?



Microservices – the new architectural style.

Martin Fowler,Mar 2014

- 微服务架构是一种架构模式，将单体应用划分成**一组小的服务**，服务之间互相协作，为用户提供最终价值
- 每个服务运行在其**独立的进程中**，服务间采用**轻量级的通信机制**协作（通常是基于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

Fred George

Micro (u)Services Architecture - small, short lived services rather than SOA. *2012/3*

Adrian Cockcroft

Loosely coupled service oriented architecture with bounded contexts. *2014/11*

Neal Ford

Microservices are the first post DevOps revolution architecture. *2015/10*

.....

以缩短交付周期为核心

基于DevOps

的演进式架构

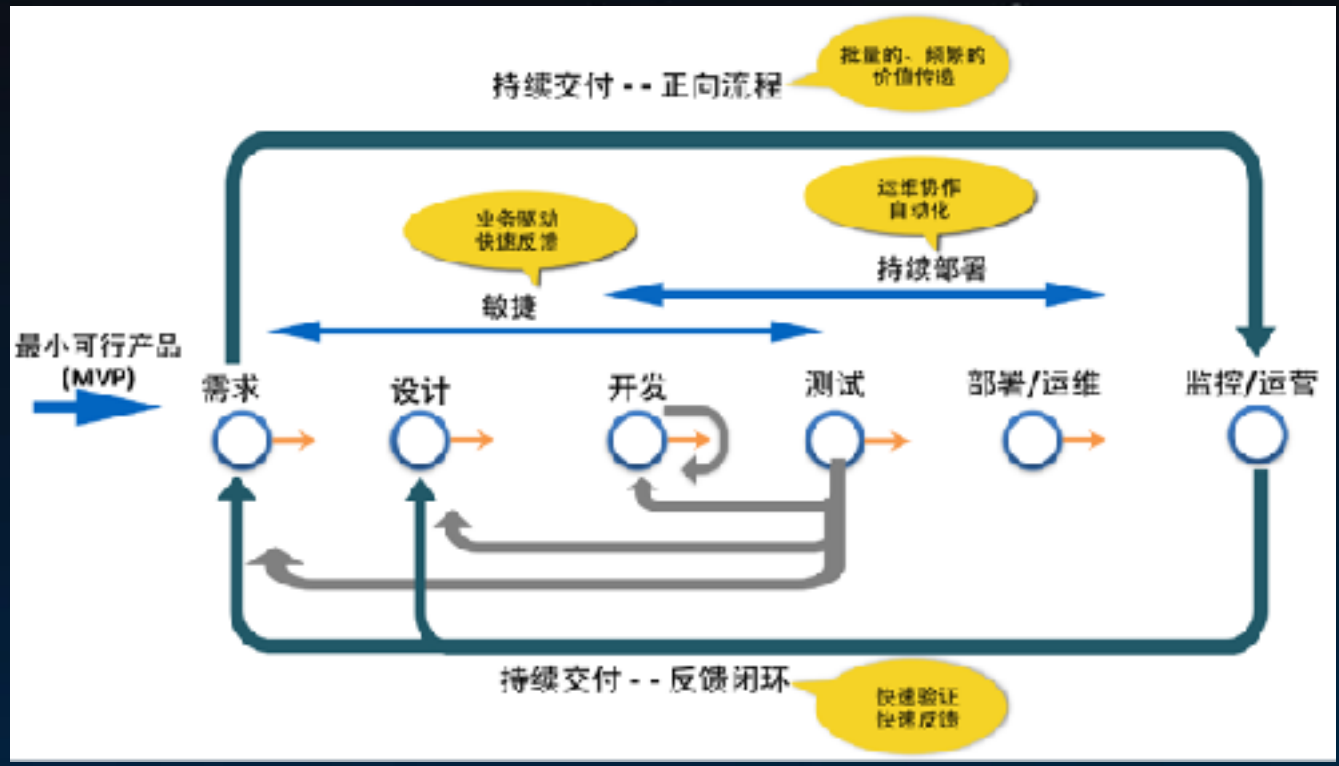
为什么以缩短交付周期为核心?

We can't deliver new business value as fast as the business needs it. It is absolutely clear that old approaches are showing down the company's growth and we are at risk of losing out competitive advantages.

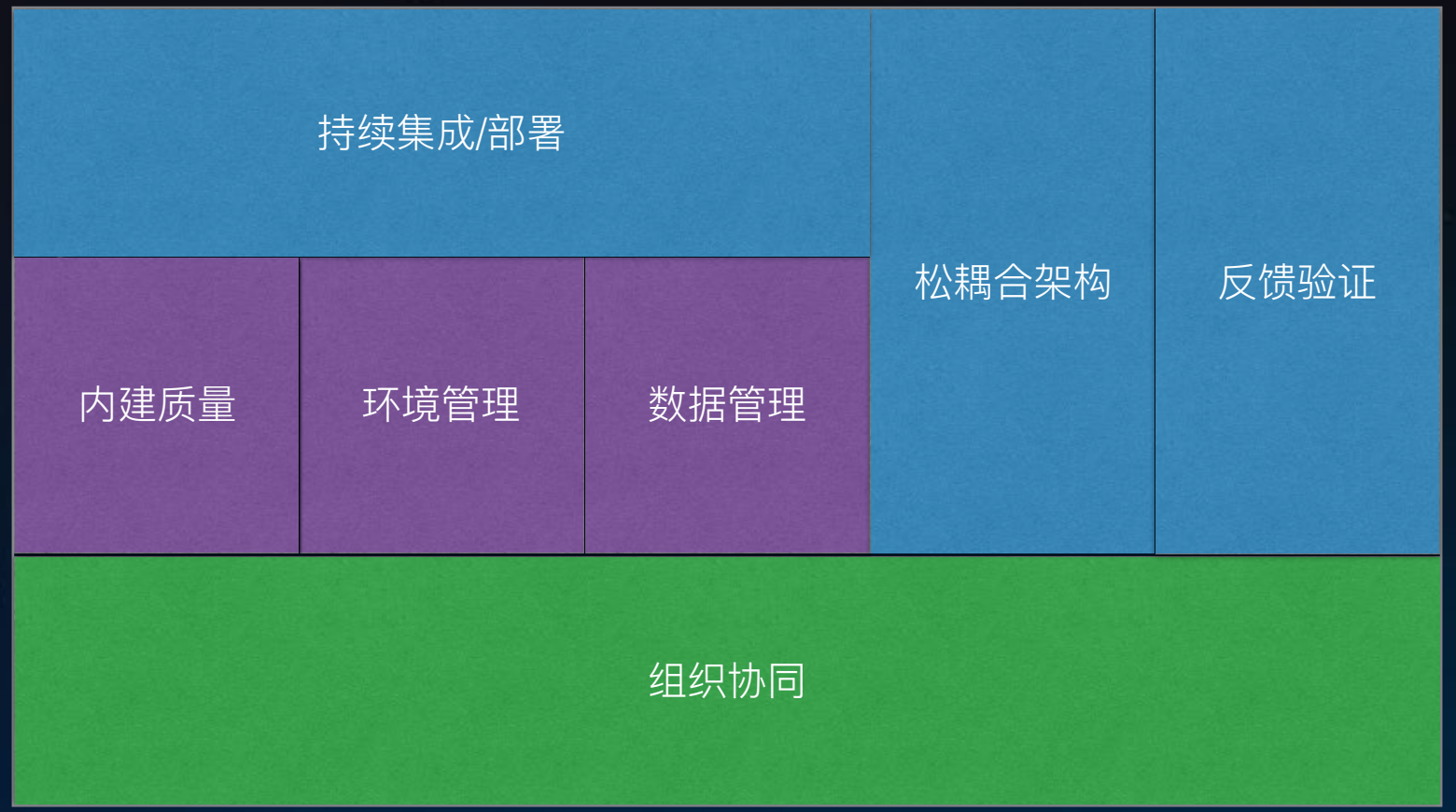
我们交付特性的速度已经无法满足业务变化需要。旧的交付方式阻碍了组织的发展，而我们也因此正在丧失竞争力。

O'Reilly Software Architecture Conference 2017.4

- 缩短交付周期
- 降低发布风险
- 质量内嵌

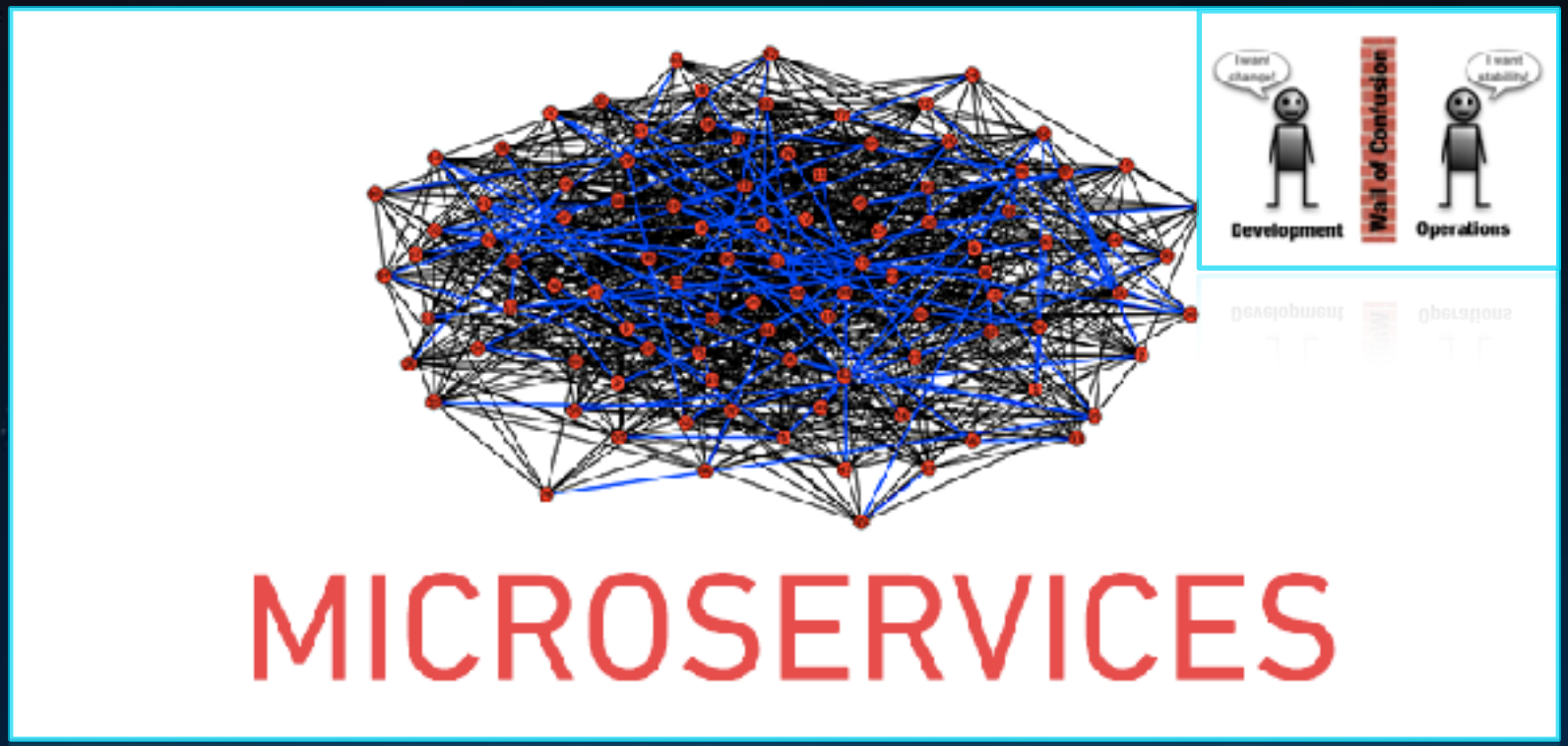


https://www.continuousdelivery.com/

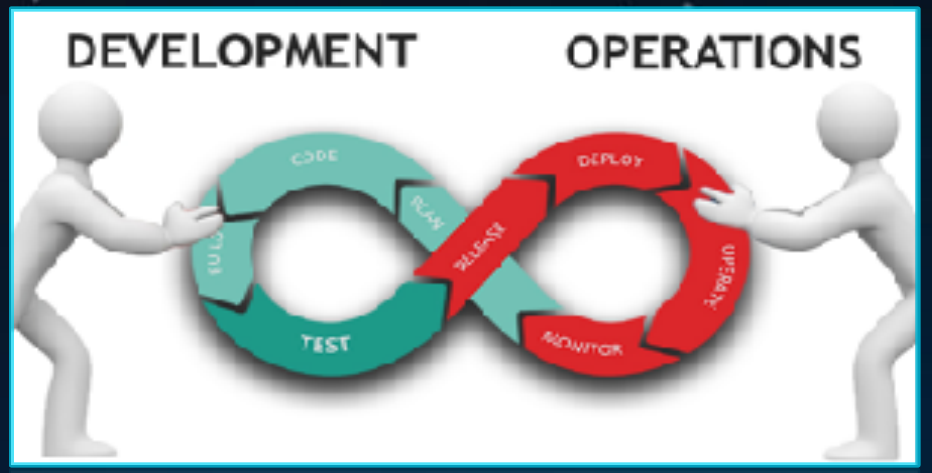


微服务架构是松耦合架构机制的一种实现

为什么基于DevOps?



为什么基于DevOps?



- Communication
- Automation
- Measuring
- Sharing

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

什么是演进式架构?

架构一旦确定，很难改变

什么是演进式架构?

支持增量式变更作为第一原则

- 演进是动态平衡
- 痛苦的事情提前做
- 运维意识是关键

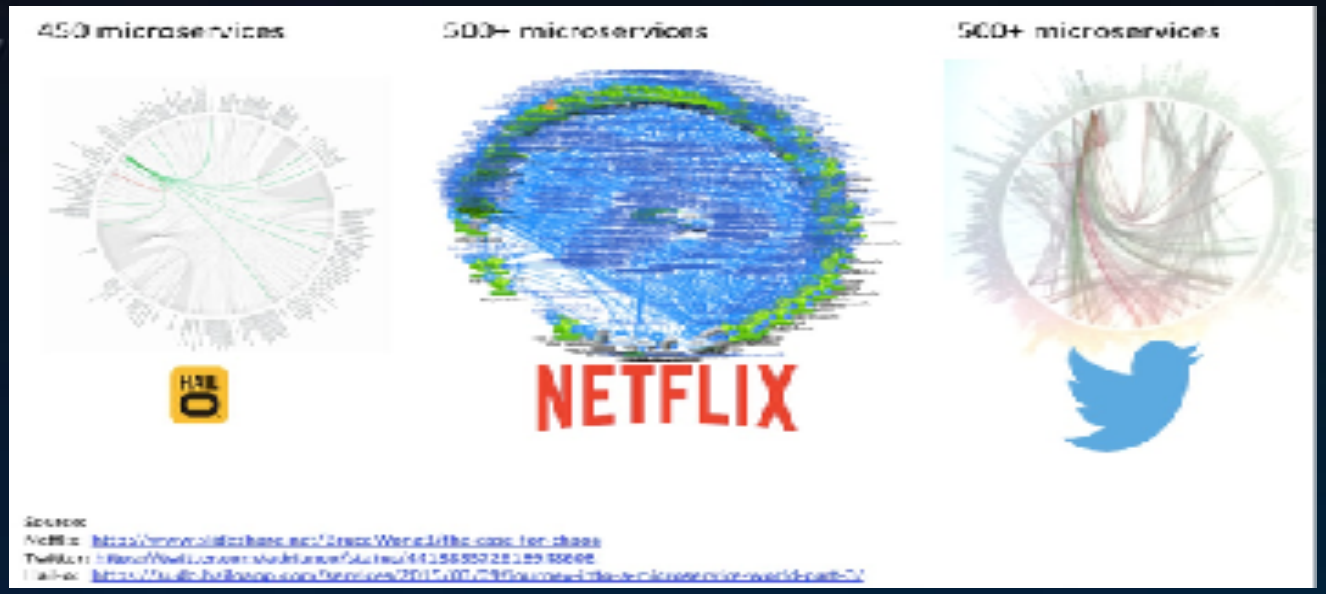
演进式架构 - 动态的平衡

架构的演进基于业务、技术和团队的平衡

演进式架构 - 痛苦的事提前做

识别问题并用自动化的手段消除痛苦

演进式架构 - 运维意识是关键



架构只是抽象，直到真正投入运维产生价值

议题

- 微服务架构的核心
- 微服务架构设计原则
- 微服务架构生态系统

It is about architecture,
BUT not only architecture……

微服务架构的设计原则

- 竞争能力平衡
- 围绕业务构建
- “去”中心化
- 自动化“一切”

没有完美的架构，只有适合的架构

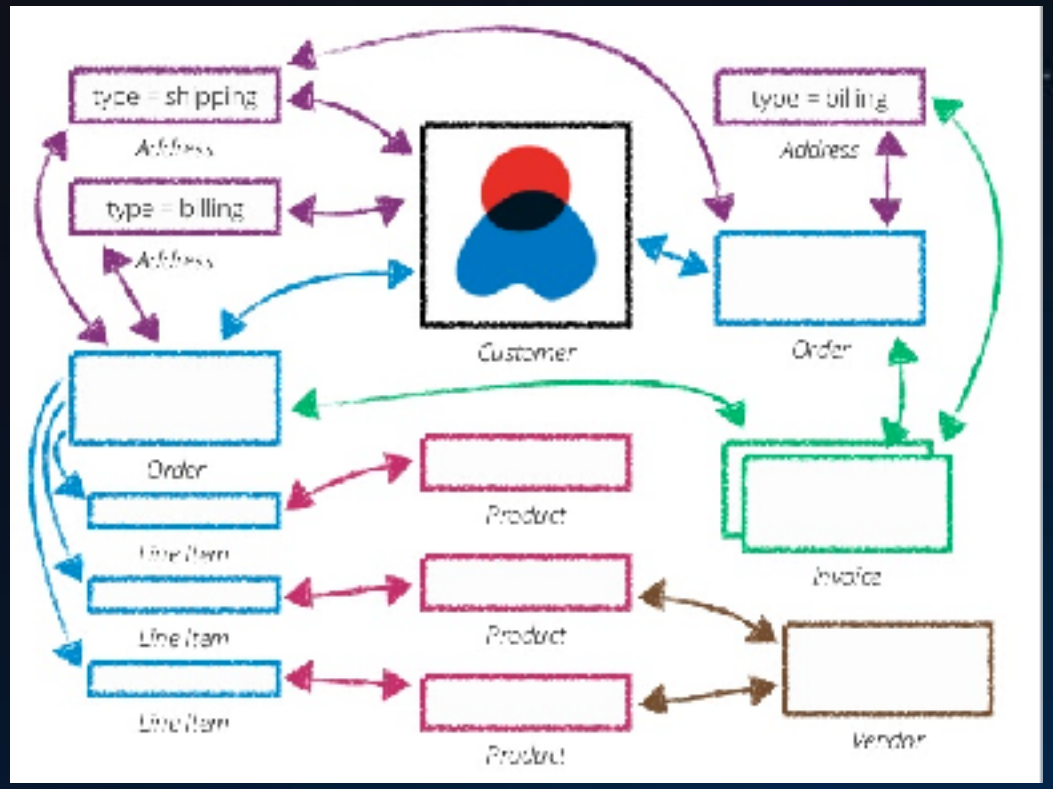
- Velocity
- Scalability
- Availability
- Agility

- Operation-ability
- Complexity
- Performance
- Testability

微服务架构的设计原则

- 竞争能力平衡
- 围绕业务构建
- “去”中心化
- 自动化“一切”

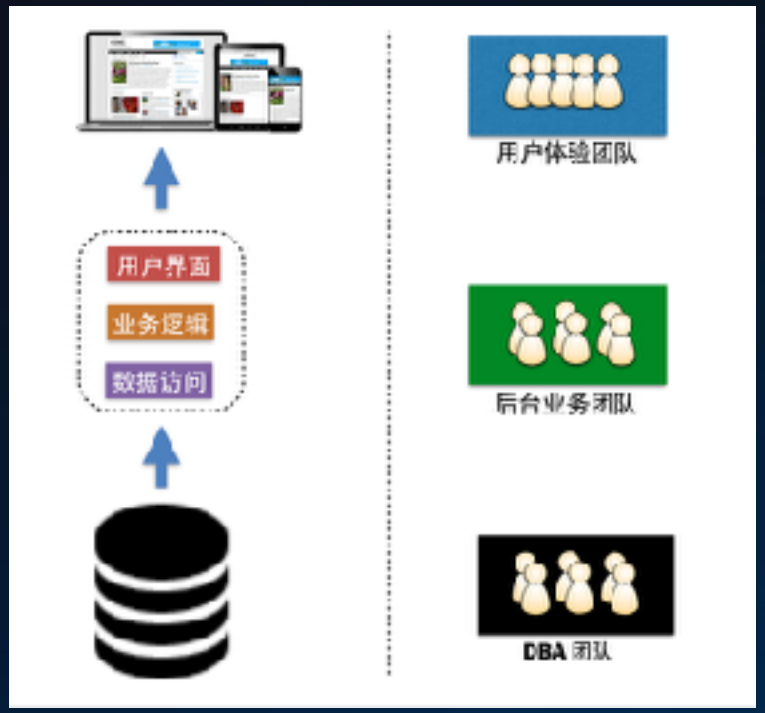
围绕业务-构建团队



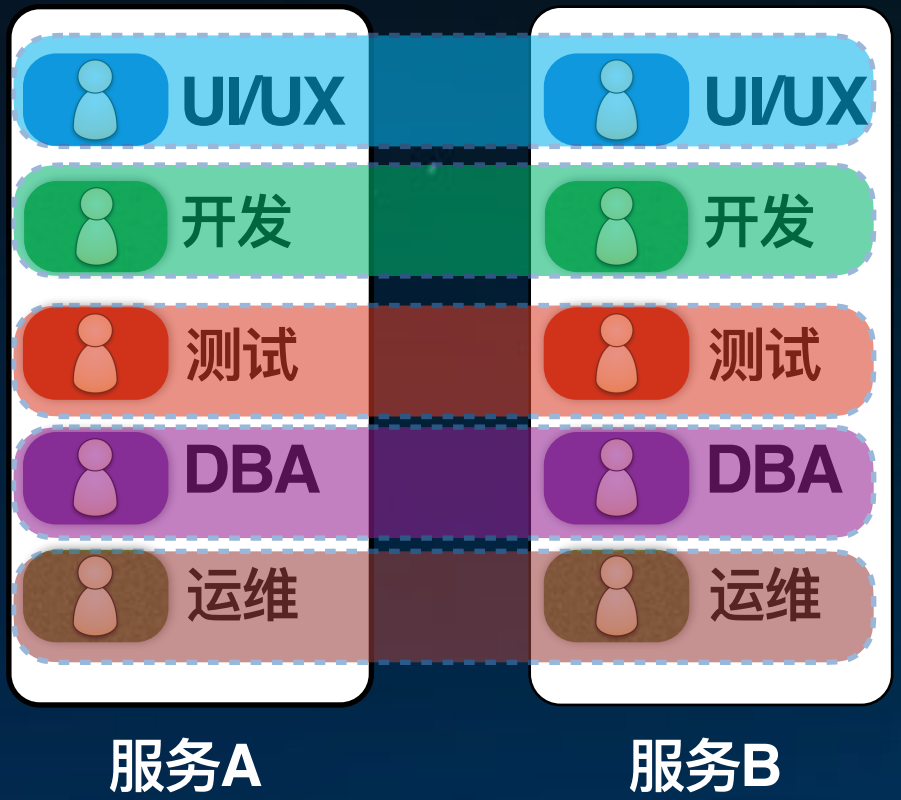
- 业务领域模型(Domain)
- 通用语言(Ubiquitous language)
- 界限上下文(Bundle context)

围绕业务-构建团队

康威定律：组织产生的设计成果等同于组织内的协作结构



VS

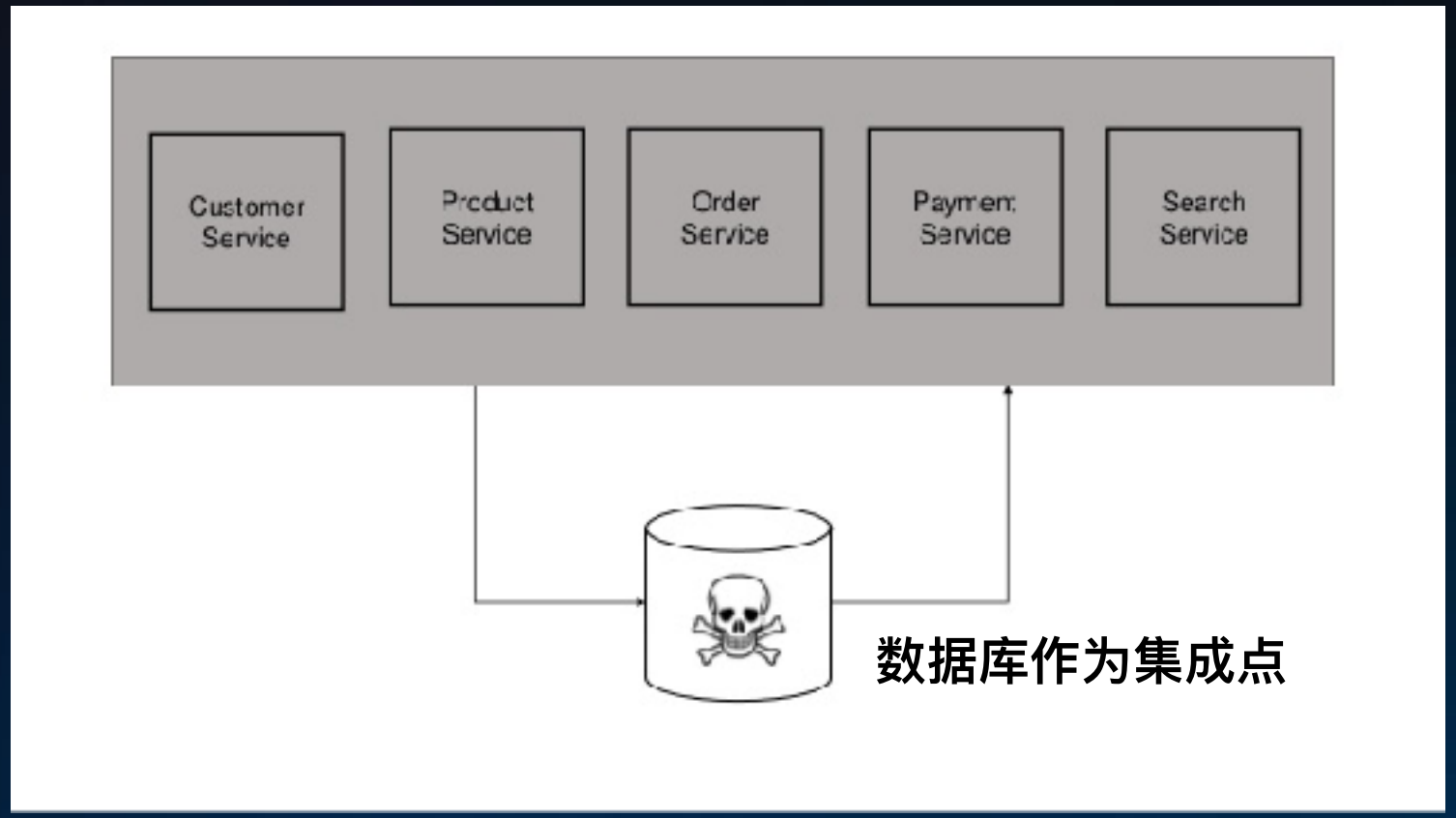


微服务架构的设计原则

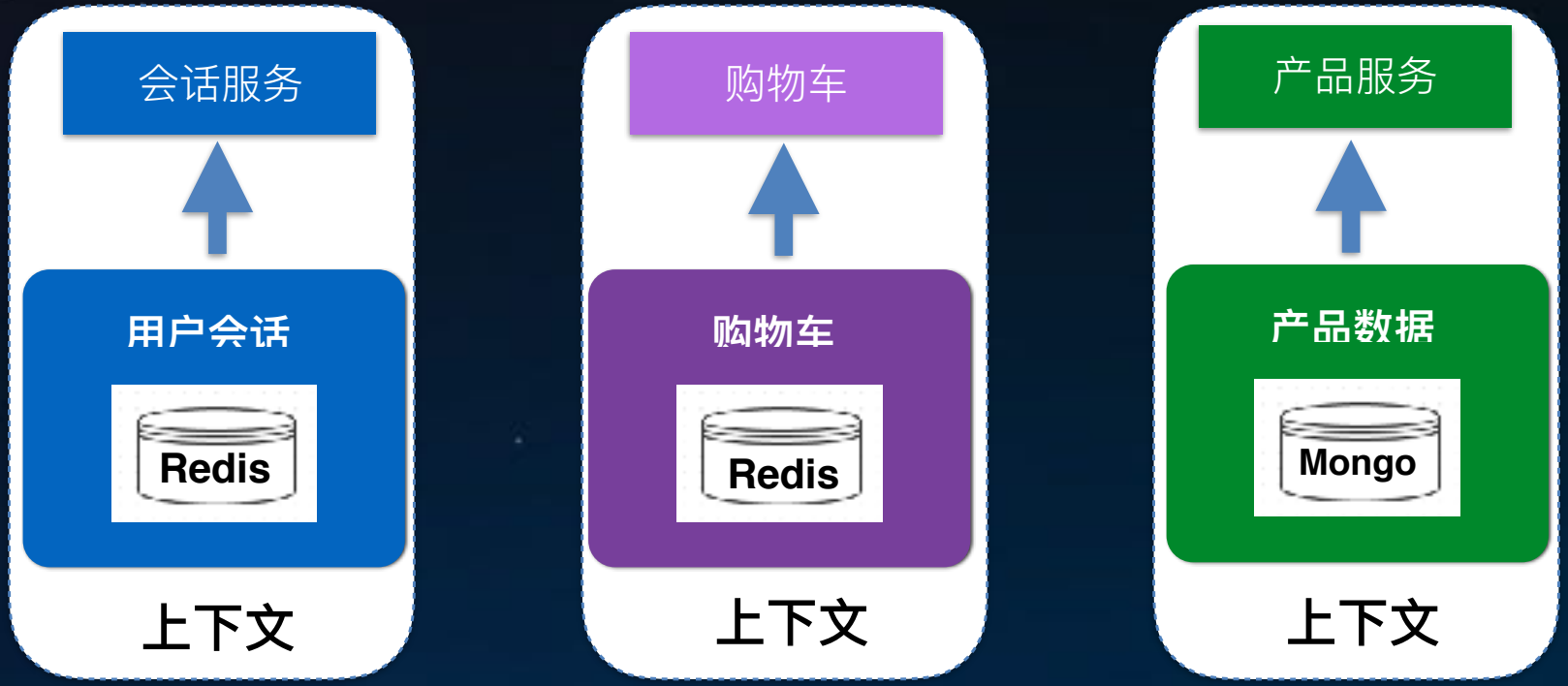
- 竞争能力平衡
- 围绕业务构建
- “去”中心化
- 自动化“一切”

去“中心化” - 用合适的技术解决问题

去“中心化” - 业务持有数据

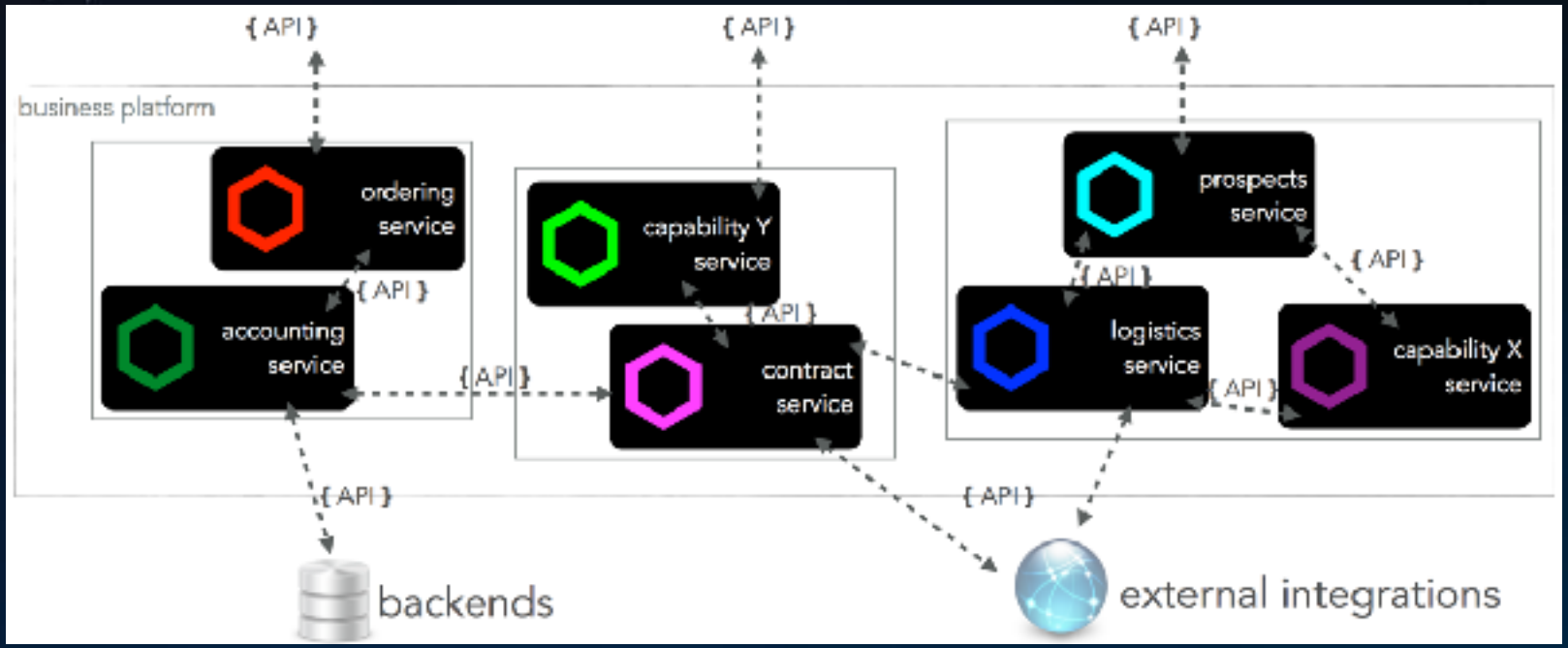

Customer Service
Product Service
Order Service
Payment Service
Search Service
数据库作为集成点

去“中心化” - 业务持有数据



去“中心化” - 智能服务而非智能总线

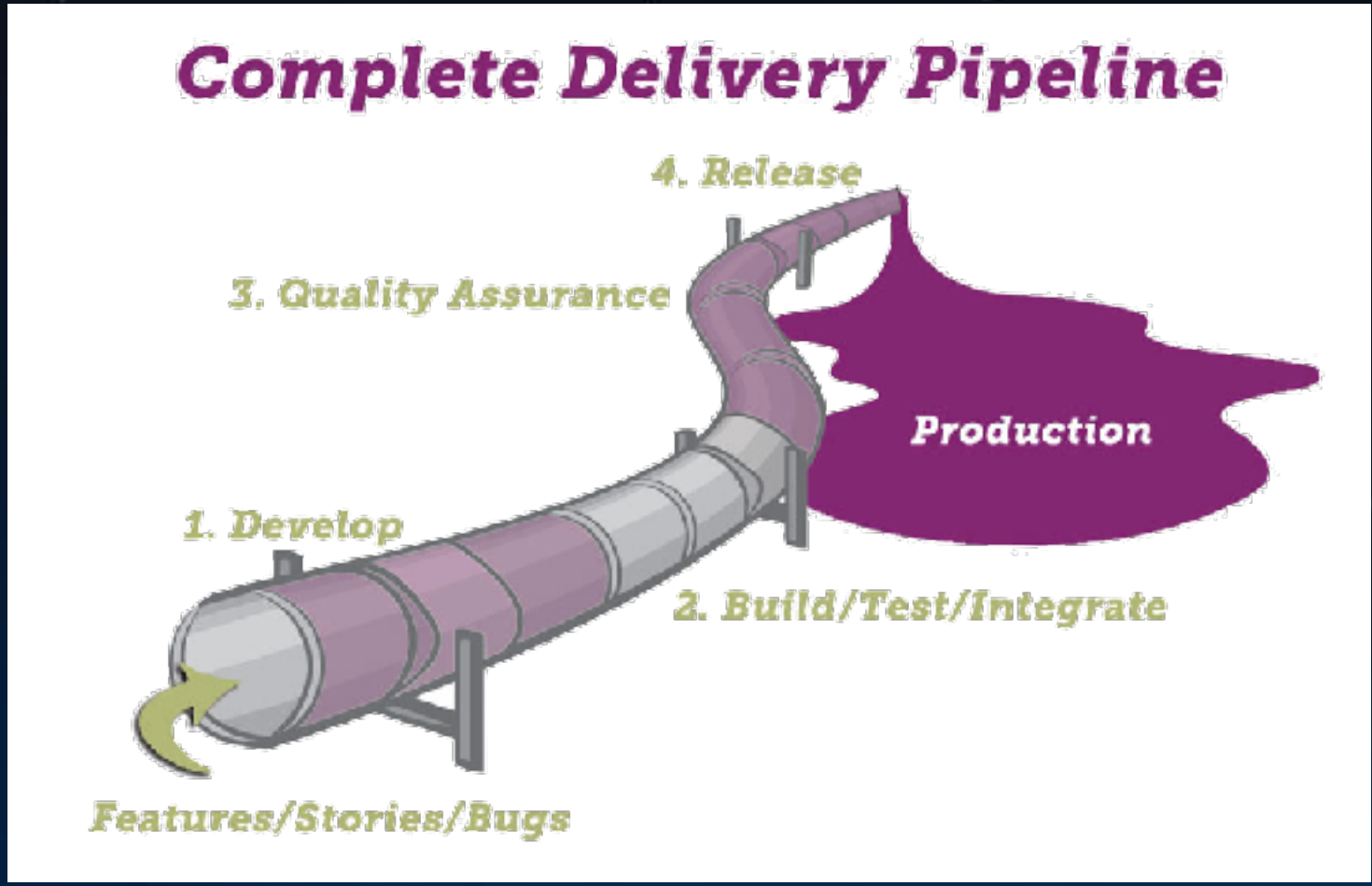
去“中心化” - 智能服务而非智能总线



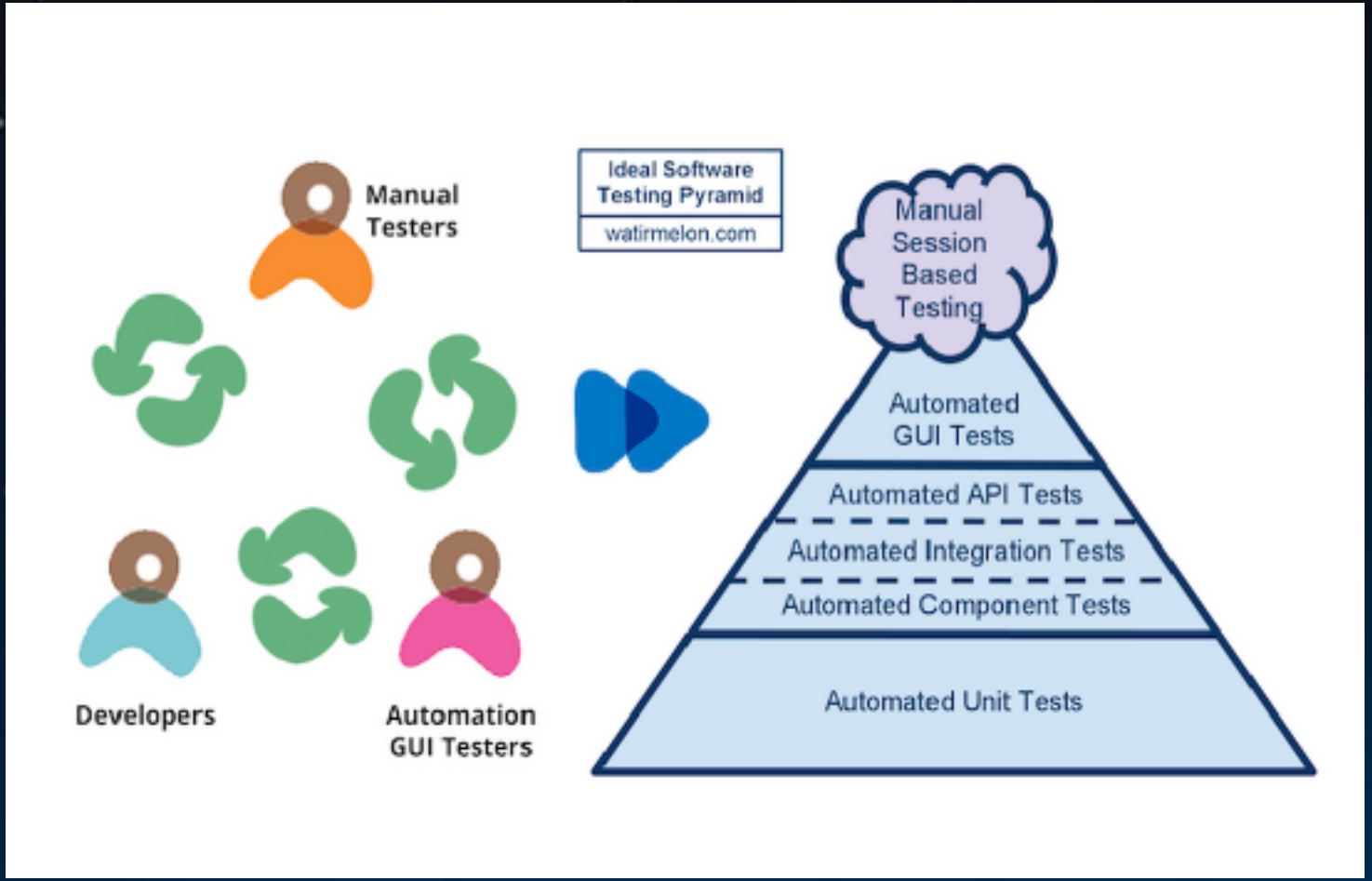
微服务架构的设计原则

- 竞争能力平衡
- 围绕业务构建
- “去”中心化
- 自动化“一切”

自动化“一切”- 交付流水线


Complete Delivery Pipeline
4. Release
3. Quality Assurance
Production
1. Develop
2. Build/Test/Integrate
Features/Stories/Bugs

自动化“一切”- 测试策略与应用



自动化“一切” - 基础设施与部署流程

应用部署自动化（包/映像/容器）

包部署　映像部署　容器部署

议题

- 微服务架构的核心
- 微服务架构设计原则
- 微服务架构生态系统

为什么需要生态系统?

系统化的工程

- 分布式系统复杂性
- 服务的治理与维护
- 测试策略与契约测试
- 持续交付流水线

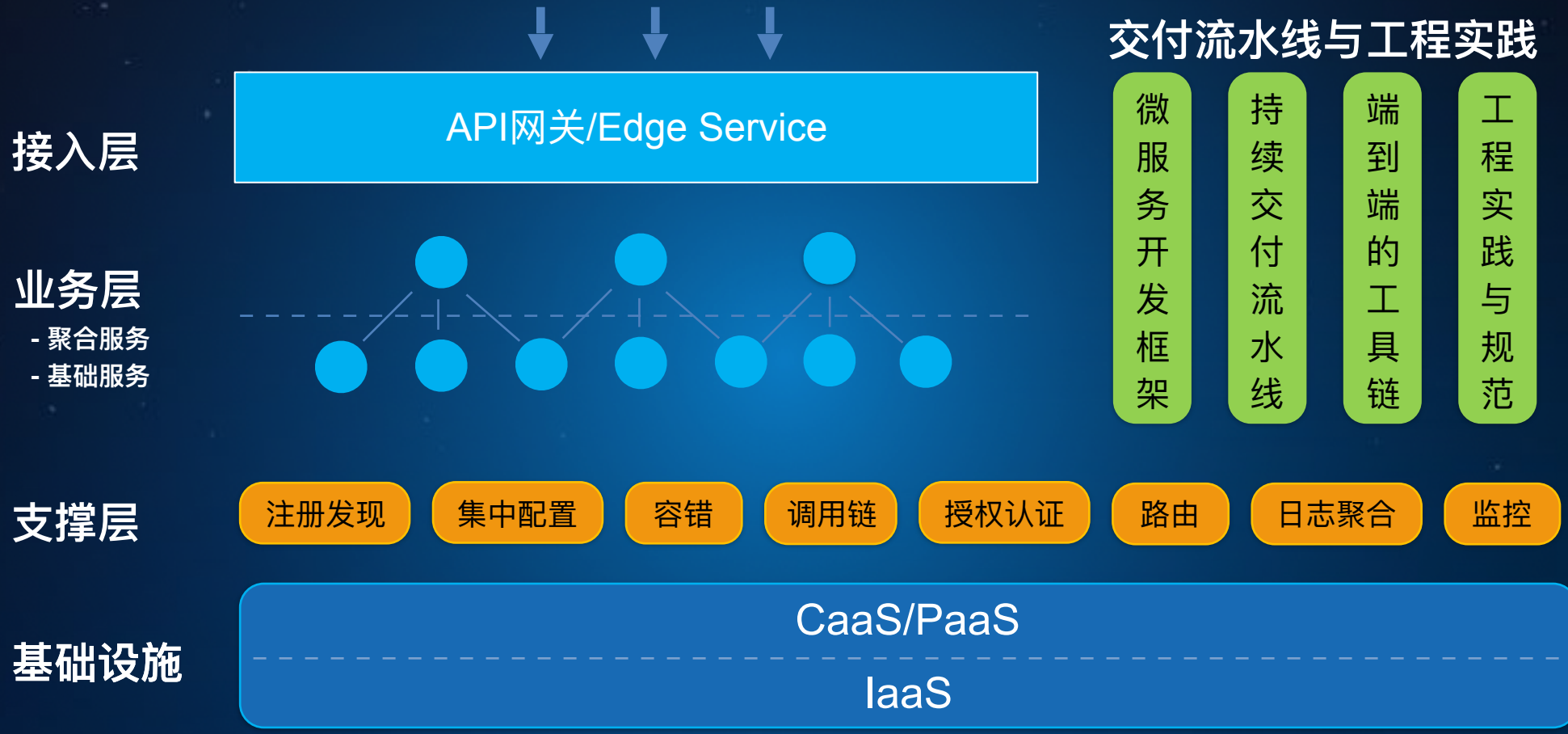
框架层出不穷

- 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

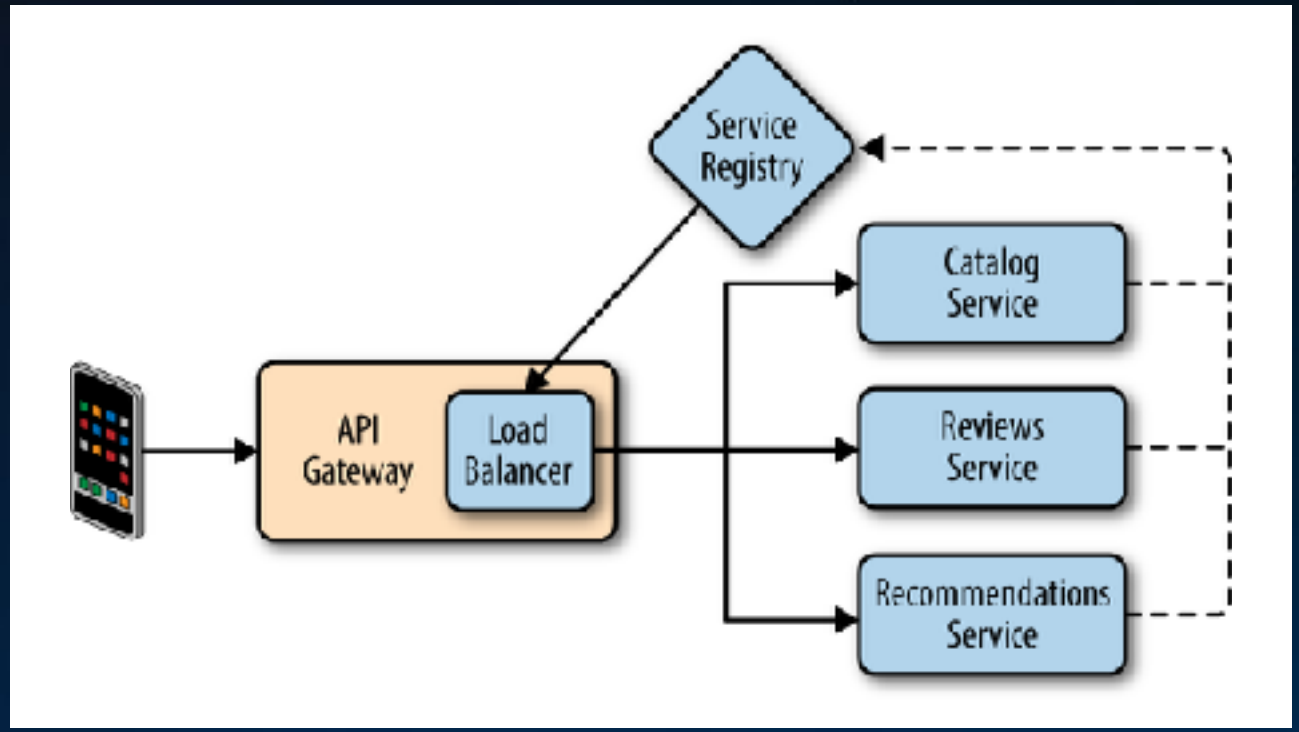
- 基础设施(私有云/公有云)
- 持续集成/持续部署流水线
- 团队的敏捷/工程化实践
- 端到端的工具链

微服务生态系统

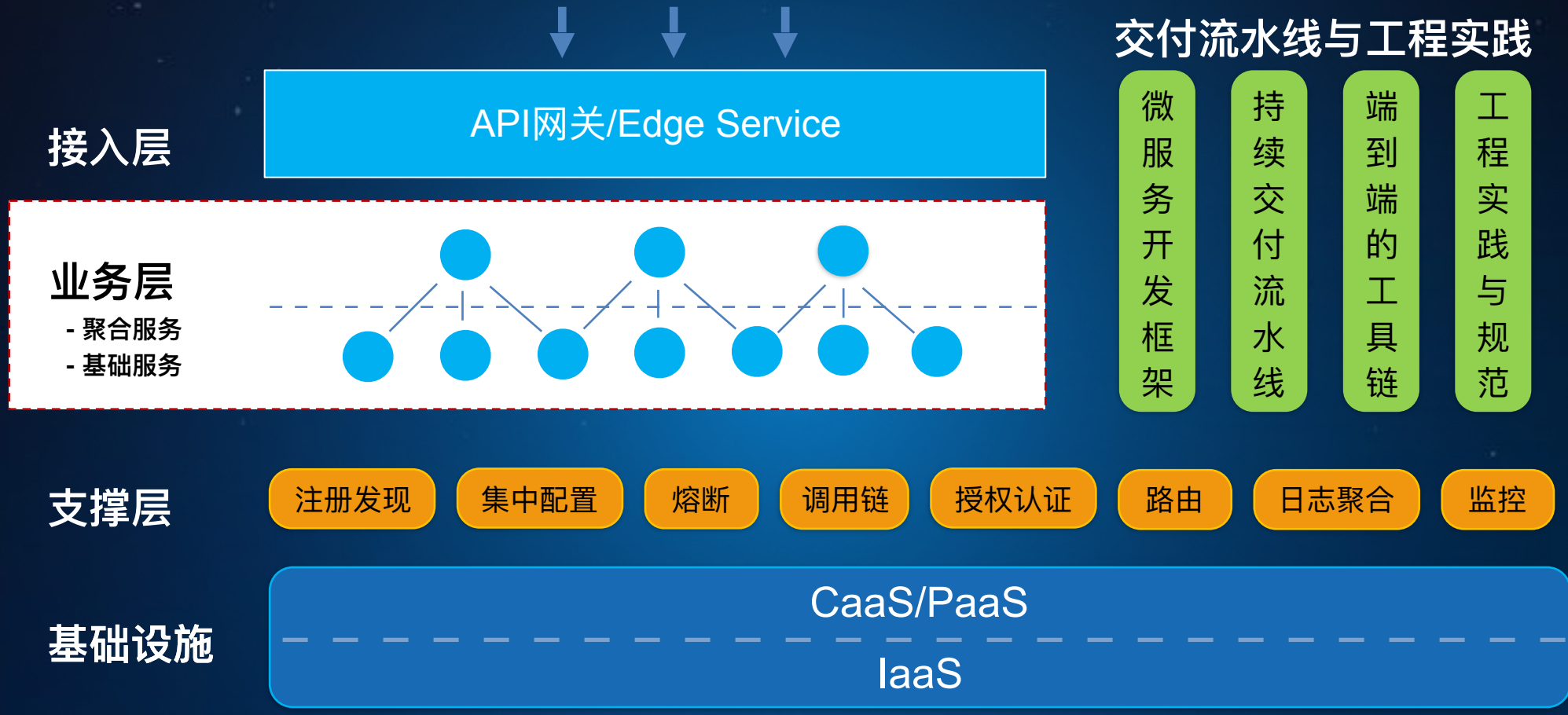


API网关

- 提供统一接口，封装内部变化
- 协议转换
- 流量限制
- 调用统计
- 安全认证

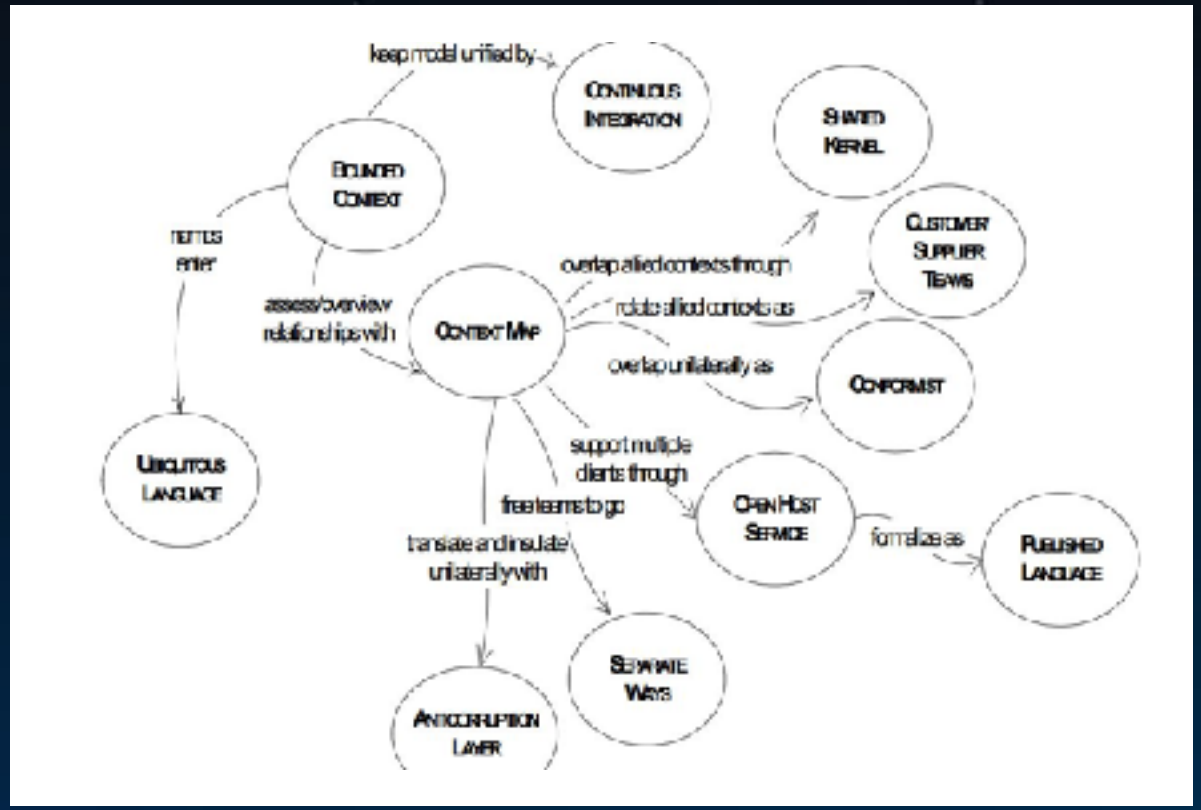


微服务生态系统



服务设计与拆分

- 面向对象设计(名词/动词)
- 可重用的逻辑
- 资源密集型部分
- 领域驱动设计
- 数据访问方式



基础服务实现

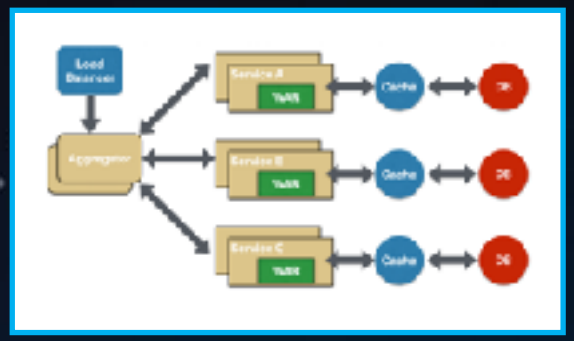
Node.js

- Actionhero - Multi-transport Node.js API server with integrated cluster capabilities and delayed tasks.
- Baucis - To build and maintain scalable HATEOAS:Level 3 REST APIs.
- Express - Fast, unopinionated, minimalist web framework for Node.js
- Graft - Full-stack javascript through microservices
- Hapi - A rich framework for building applications and services.
- Hudson Taylor - Set of libraries for building automatically documented, we
- Koa - Next generation web framework for Node.js
- Loopback - Node.js framework for creating APIs and easily connecting to
- Micro - Asynchronous HTTP microservices.
- Micro-Valhalla - A simple, fast framework for writing microservices in Node.
- Restify - Node.js module built specifically to enable you to build correct RE
- Seneca - A microservices toolkit for Node.js
- Serverless - Build and maintain web, mobile and IoT applications running
  known as JAWS!

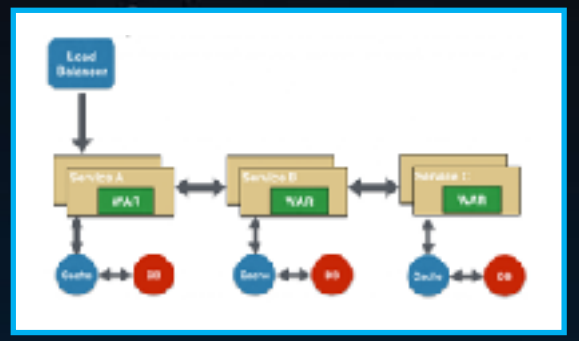
Java

- Airlift - Framework for building REST services in Java.
- Disruptor - High-performance inter-thread messaging library.
- Dropwizard - Java framework for developing ops-friendly, high-performance, RESTful web services.
- HTTP Remoting - Libraries for defining and creating RESTish/RPC servers and clients based on Feign or Retrofit as a client and Dropwizard/Jersey with JAX-RS service definitions as a server.
- Jersey - RESTful services in Java. JAX-RS reference implementation.
- MSF4J - High throughput & low memory footprint Java microservices framework.
- QBit - Reactive programming library for building microservices.
- Ratpack - Set of Java libraries that facilitate fast, efficient, evolvable and well tested HTTP applications. specific support for the Groovy language is provided.
- Restlet - Helps Java developers build web APIs that follow the REST architecture style.
- Spark - A micro-framework for creating web applications in Java 8 with minimal effort.
- Spring Boot - Makes it easy to create stand-alone, production-grade Spring based applications.

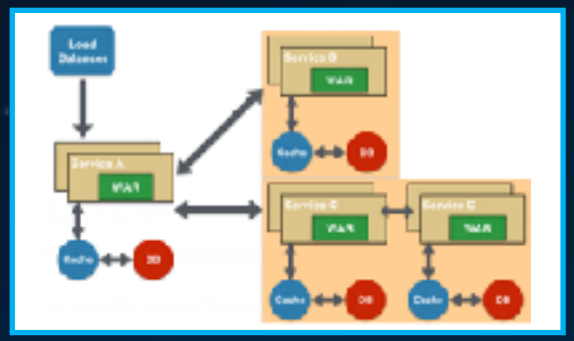
聚合服务实现



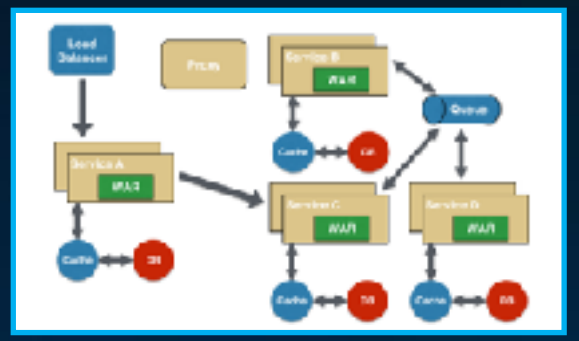
Proxy



Chained



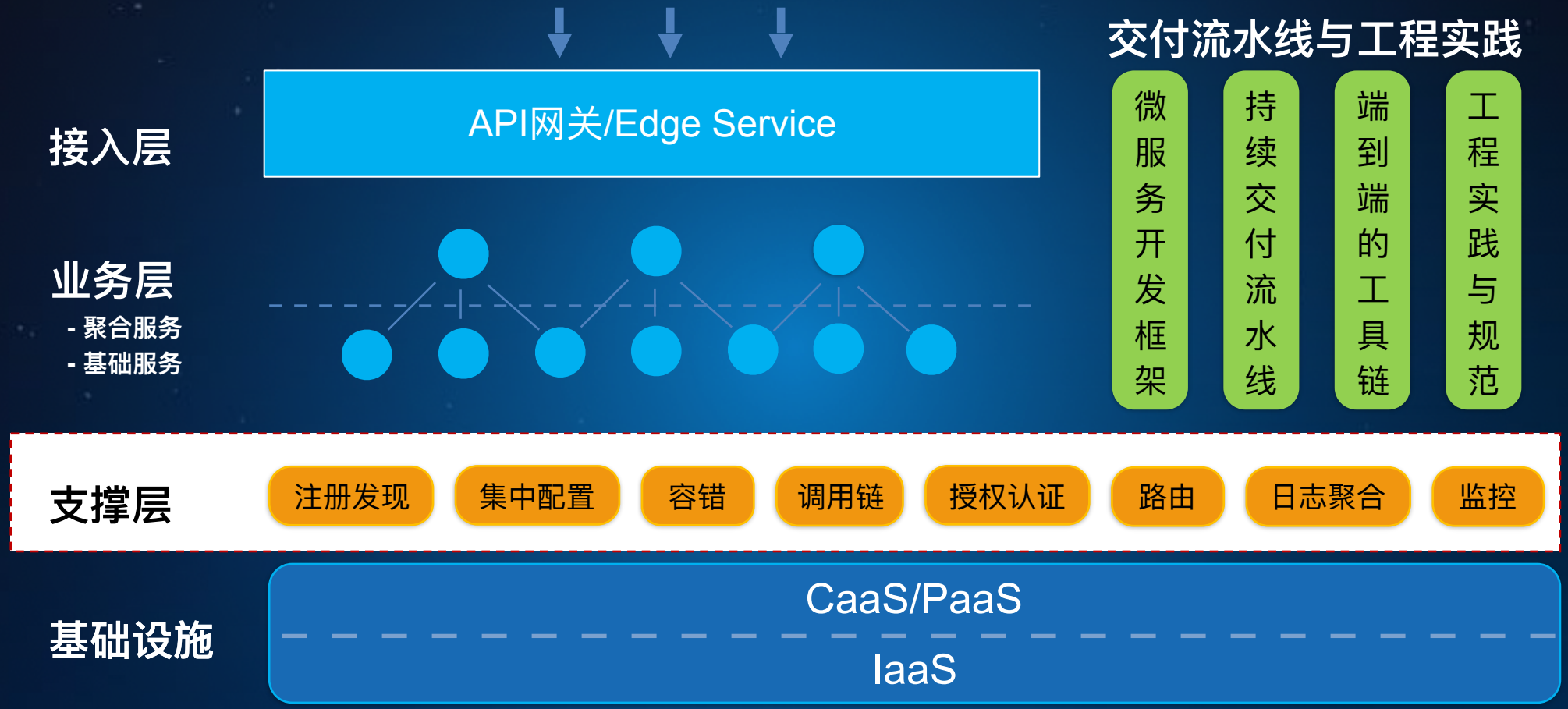
Shared Cache



Asynchronous Messaging

https://dzone.com/articles/microservice-design-patterns

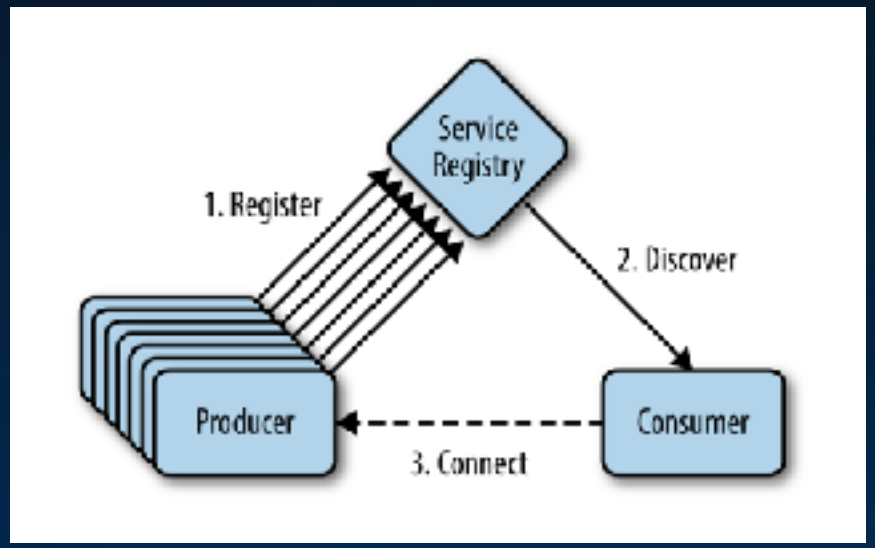
微服务生态系统



注册发现

为什么要注册发现

- 服务重启/升级后的IP地址变化
- 水平伸缩后服务的实例数量变化
- 同一个结点运行多个服务(端口不同)



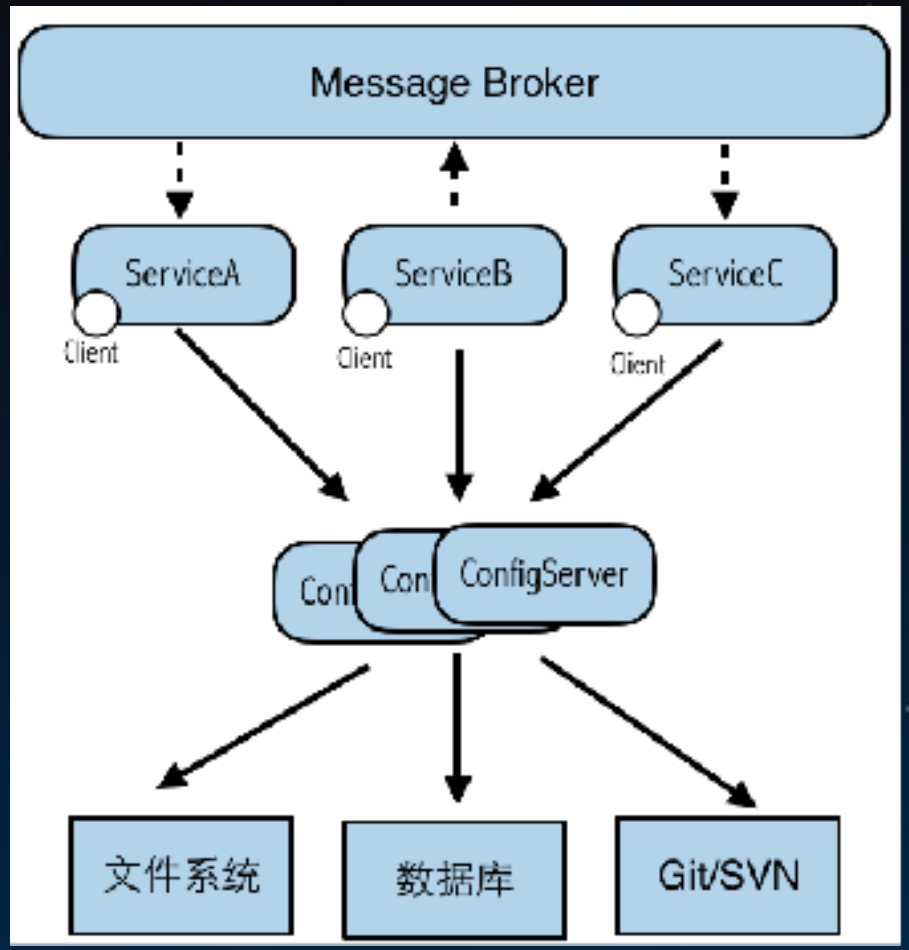
集中配置管理

配置信息的管理

- 与服务在同一个包中
- 使用隔离的配置文件
- 使用环境变量管理

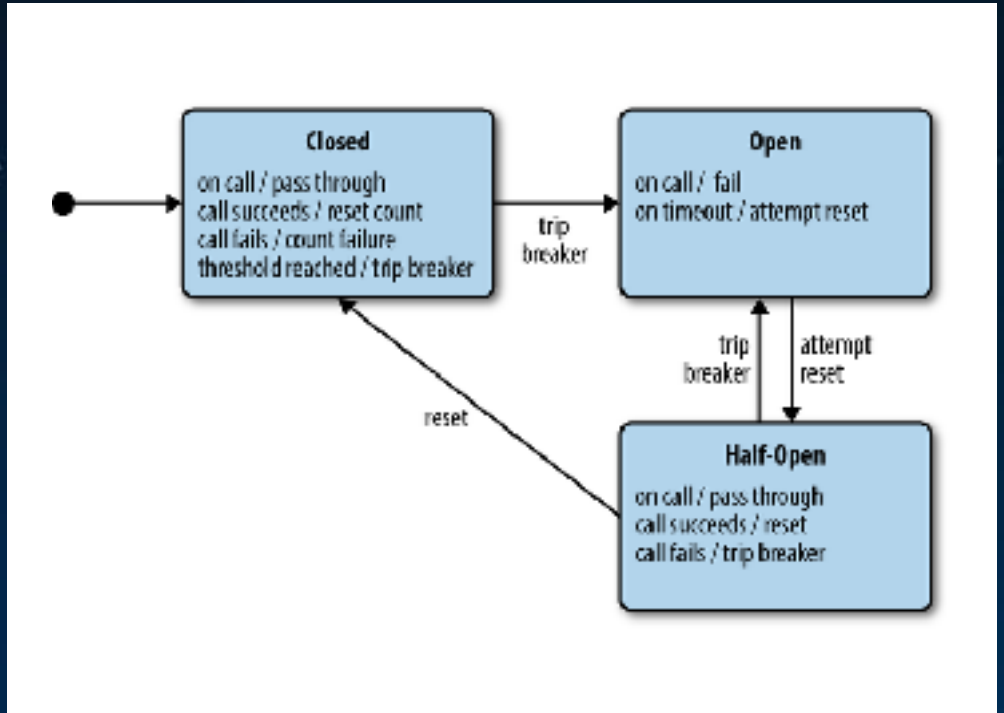
配置信息管理的挑战

- 动态更新配置信息
- 多实例间的同步
- 配置信息的版本管理

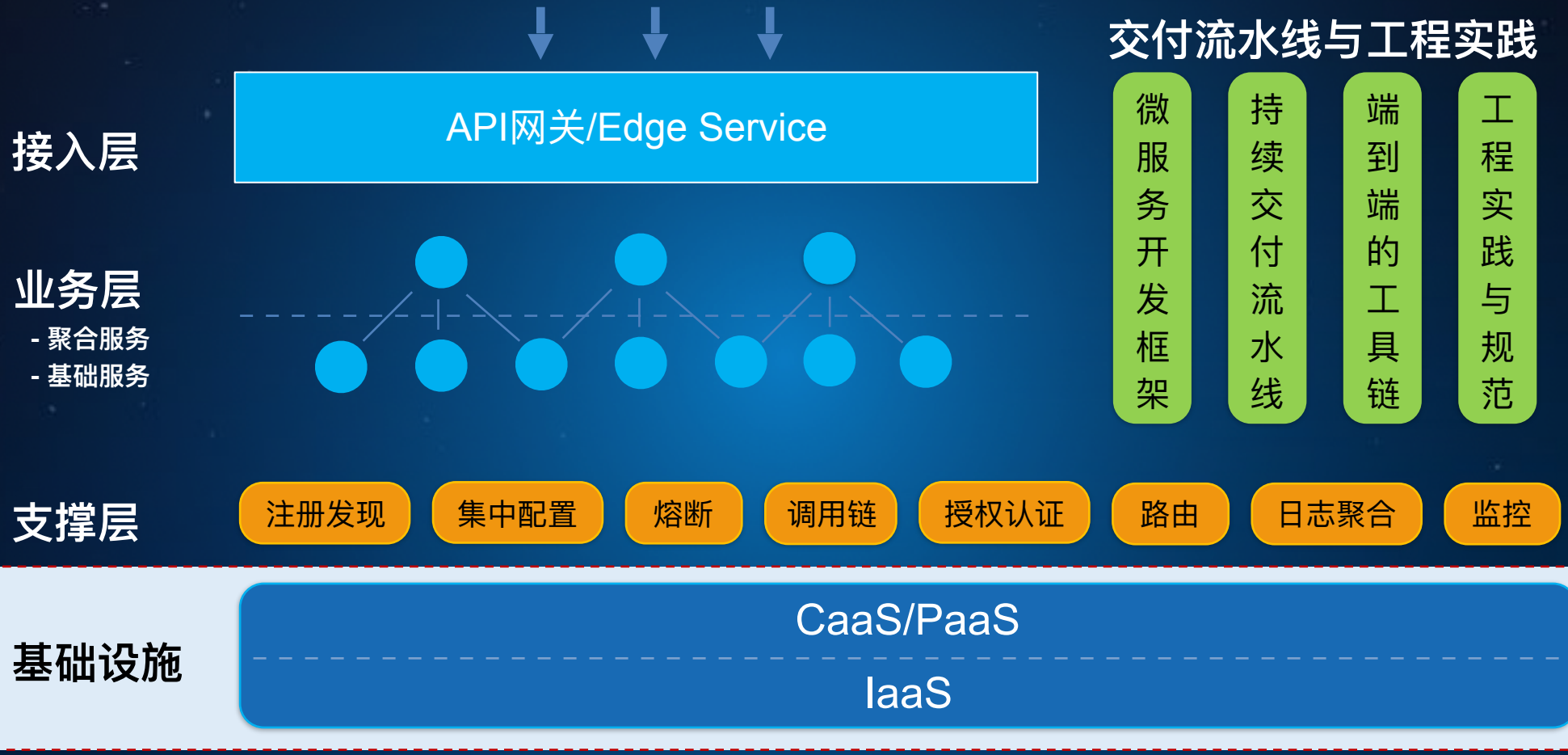


容错(Resilient)

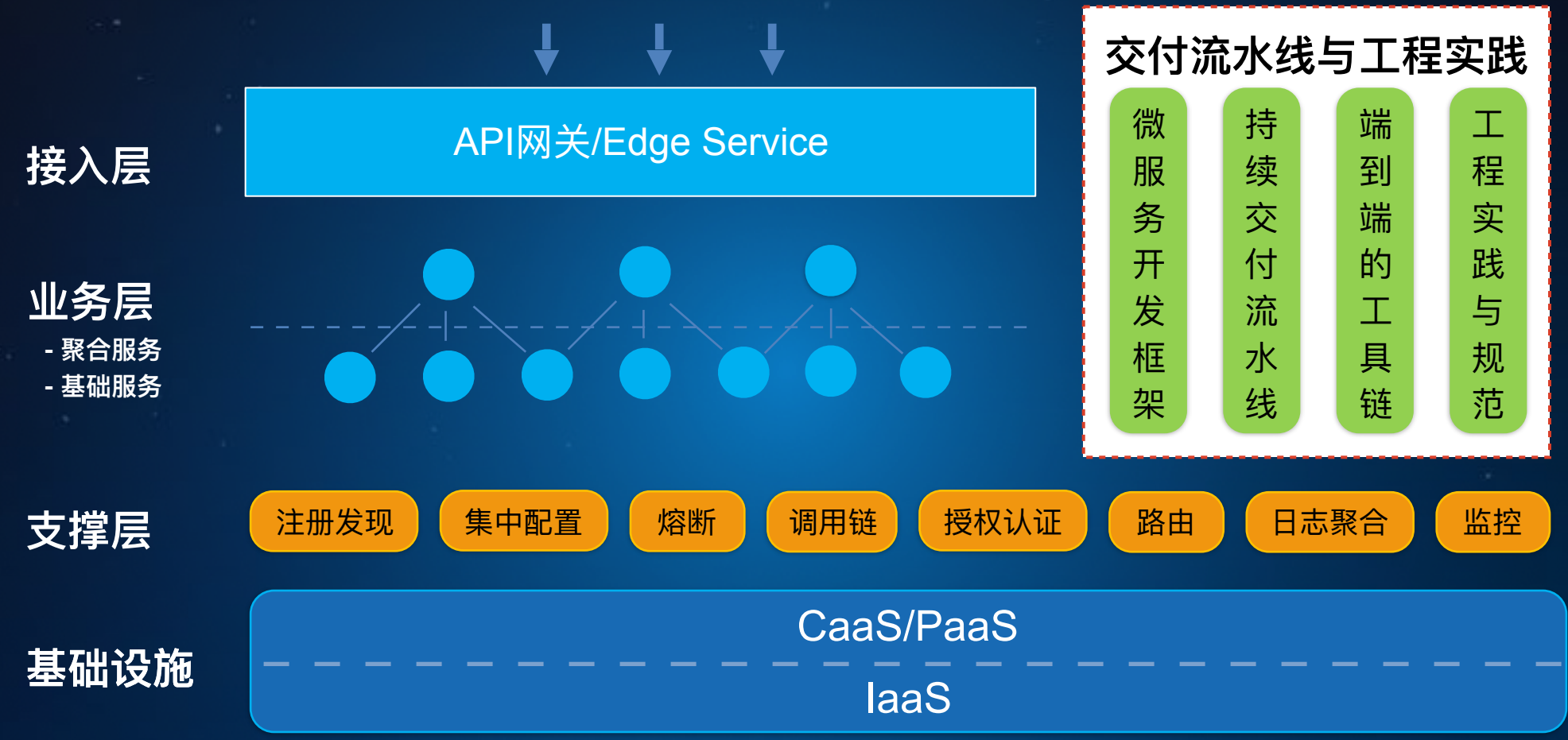
- 限流 - 超过处理能力，采用适当策略丢弃
- 降级 - 关闭非核心业务，保证核心业务可用
- 熔断 - 避免某个服务不可用导致的故障蔓延



微服务生态系统



微服务生态系统



华为微服务开发框架

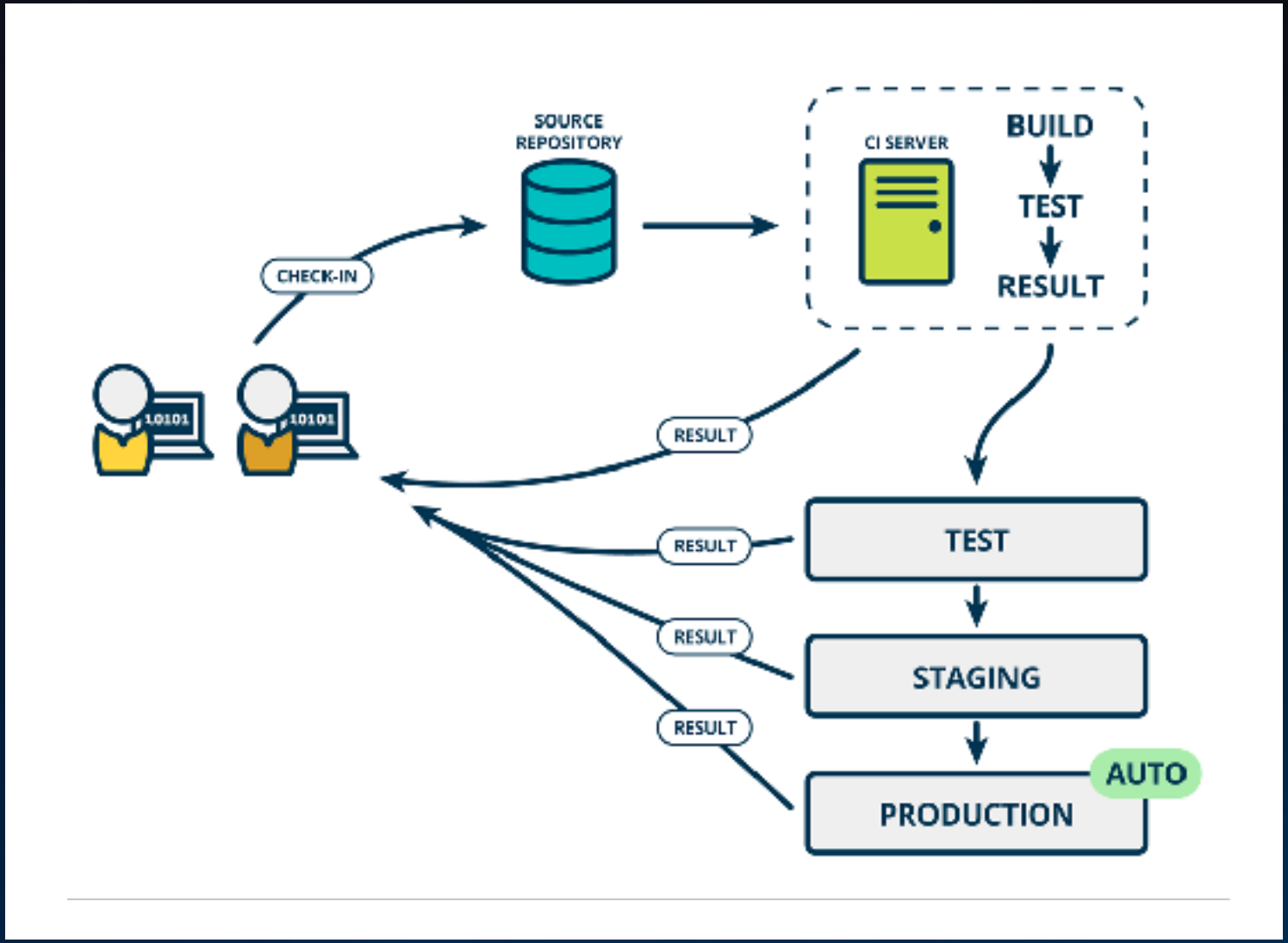
华为ServiceComb

编程模型 - 使用OpenAPI的方式定义契约

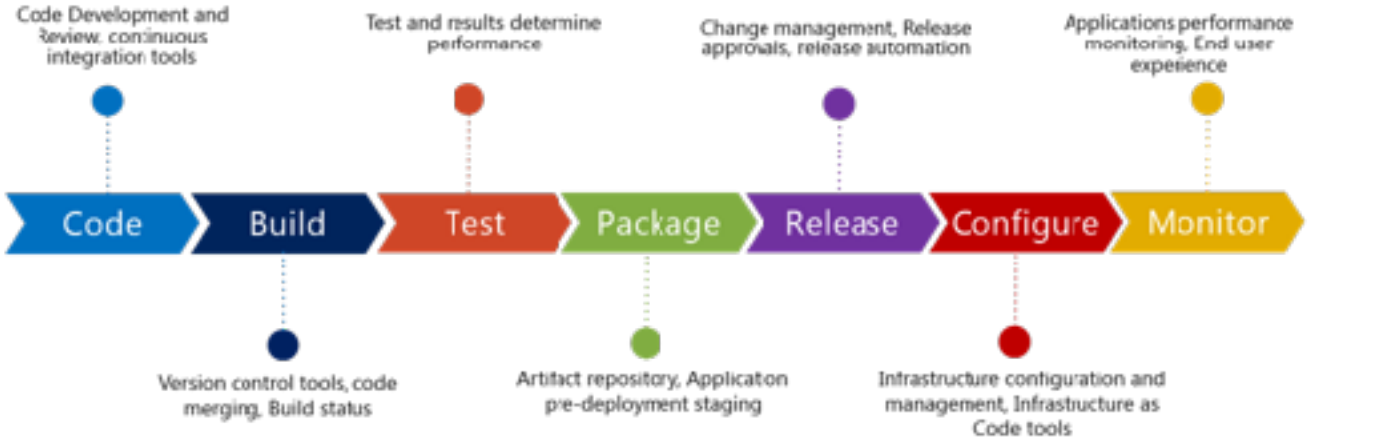
运行模型 - 服务发现/熔断/负载均衡/跟踪

通信模型 - 高效的序列化机制/传输协议

持续交付流水线



端到端工具链


Code Development and Review, continuous integration tools
Test and results determine performance
Change management, Release approvals, release automation
Applications performance monitoring, End user experience
Code
Build
Test
Package
Release
Configure
Monitor
Version control tools, code merging, Build status
Artifact repository, Application pre-deployment staging
Infrastructure configuration and management, Infrastructure as Code tools

工程化实践

- 独立代码库
- 服务说明文件
- 易于本地运行

- 监控
- 告警
- 日志聚合

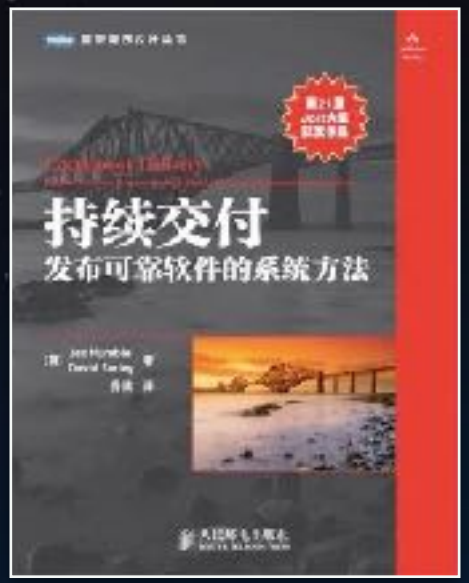
- 测试金字塔
- 契约测试
- 组件测试

- 特性开关
- 蓝绿部署
- 金丝雀发布

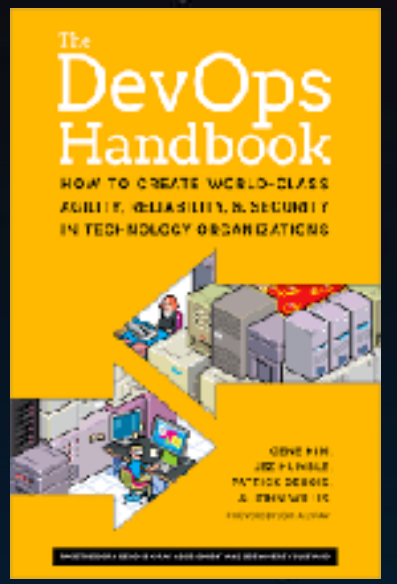
- 基础设施即代码
- 一键部署(环境/方式)

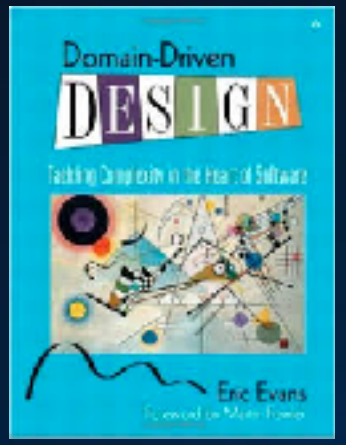
总结

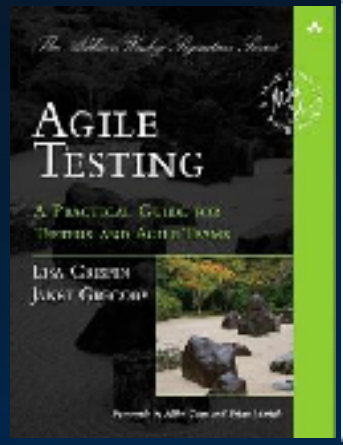
- 微服务架构的核心
- 微服务架构设计原则
- 微服务架构生态系统


持续交付
发布可靠软件的系统方法


O'REILLY
Building Microservices
Sam Newman


The DevOps Handbook
HOW TO CREATE WORLD-CLASS


Domain-Driven
DESIGN
Eric Evans


AGILE
TESTING


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


微服务
架构与实践

Thank You

- Github：https://github.com/ServiceComb
- 官网：https://www.servicecomb.io