

极客时间 Java 进阶训练营

第 20 课

分布式服务-Spring Cloud 与微服务架构



KimmKing

Apache Dubbo/ShardingSphere PMC

个人介绍

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

目录

1. 微服务架构发展历程*
2. 微服务架构应用场景*
3. 微服务架构最佳实践*
4. Spring Cloud 技术体系*
5. 微服务相关框架与工具
6. 总结回顾与作业实践

1. 微服务架构发展历程

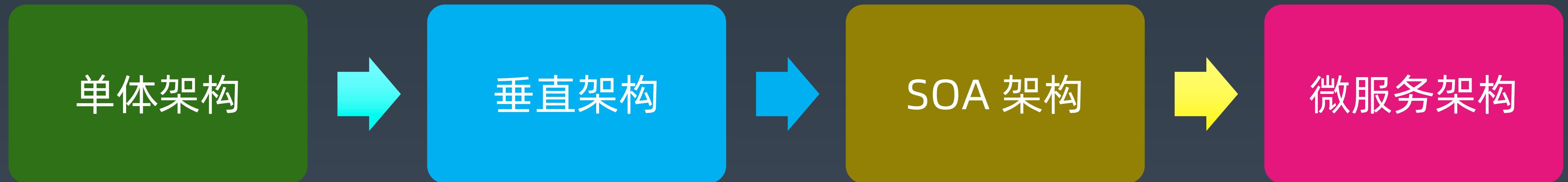
微服务发展历程



不管是互联网，还是银行、证券、保险，业务越来越复杂，数据越来越多。

系统对性能、稳定性，一致性，可用性，扩展性，可维护性，要求越来越高。

微服务发展历程



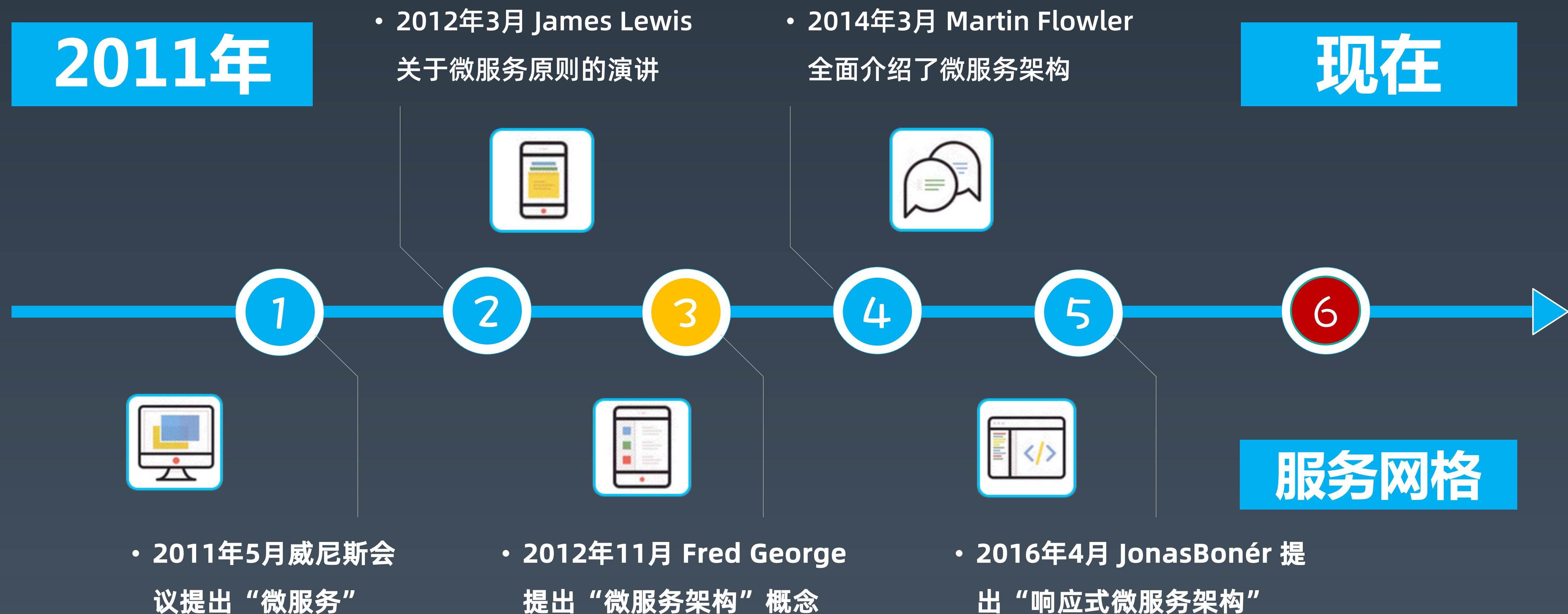
简单单体模式是最简单的架构风格，所有的代码全都在一个项目中。这样研发团队的任何一个人都可以随时修改任意的一段代码，或者增加一些新的代码。

分层是一个典型的对复杂系统（不仅仅是软件）进行结构化思考和抽象聚合的通用性办法，也符合金字塔原理。MVC是一个非常常见的3层（3-Tier）结构架构模式。

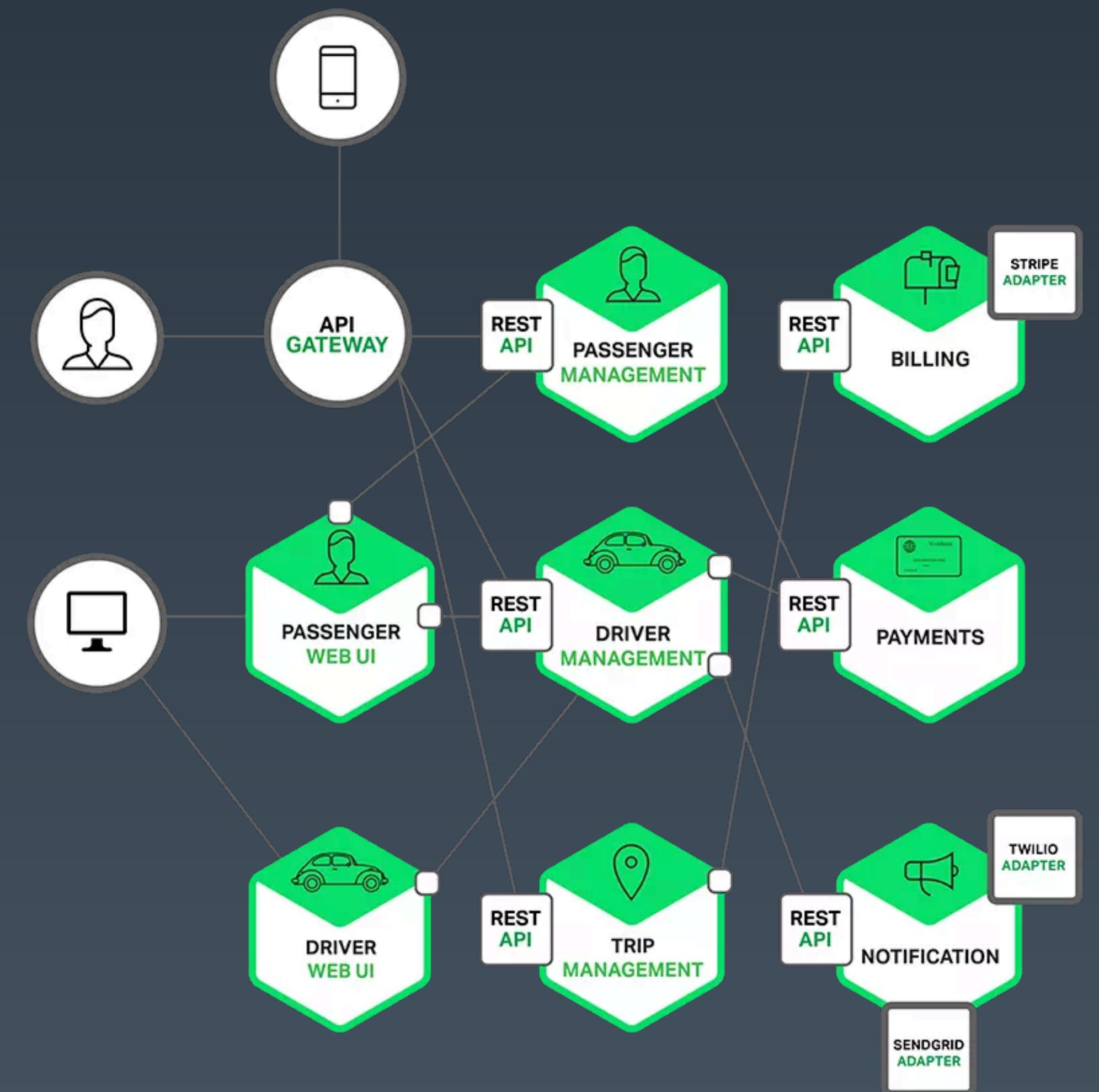
面向服务架构（SOA）是一种建设企业IT生态系统的架构指导思想。SOA的关注点是服务。服务最基本业务功能单元，由平台中立性的接口契约来定义。

微服务架构风格，以实现一组微服务的方式来开发一个独立的应用系统的方法。其中每个小微服务都运行在自己的进程中，一般采用HTTP资源API这样轻量的机制相互通信。

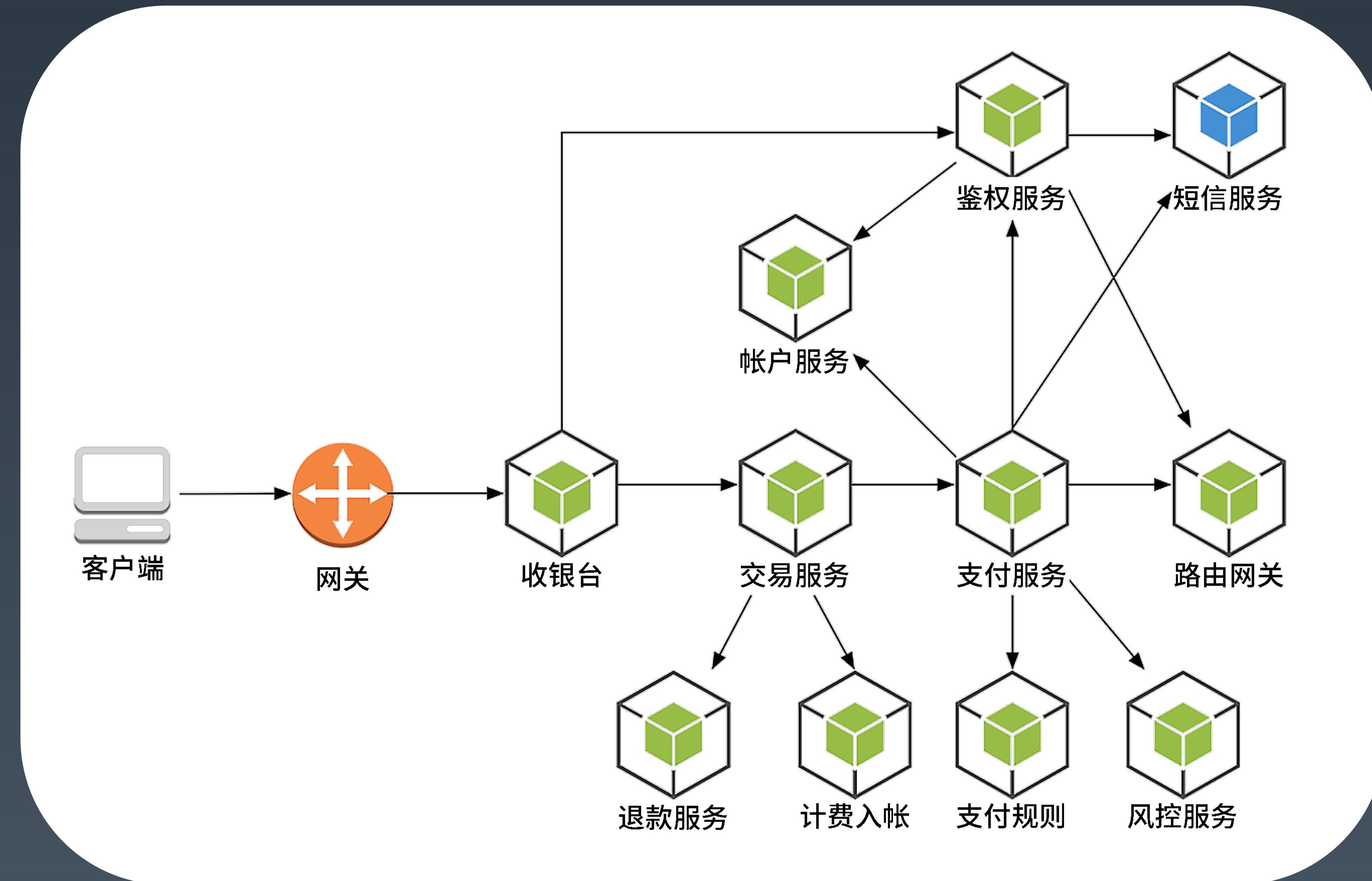
微服务发展历程



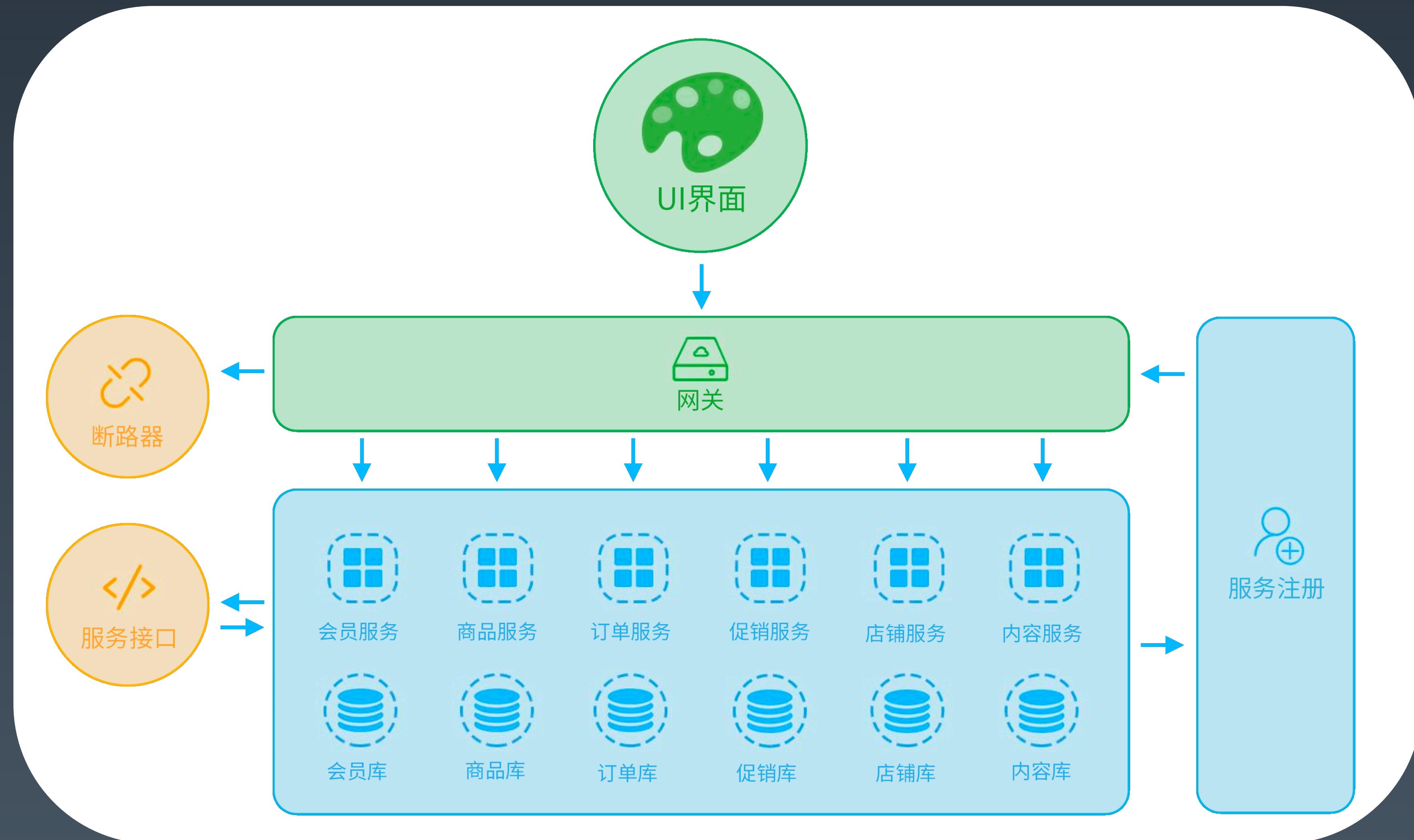
微服务架构



微服务架构



微服务架构



微服务发展历程--1.响应式微服务

- 响应式编程是一个专注于数据流和变化传递的异步编程范式。 -- 维基百科

响应式宣言

即时响应性

可恢复性

弹性

消息驱动

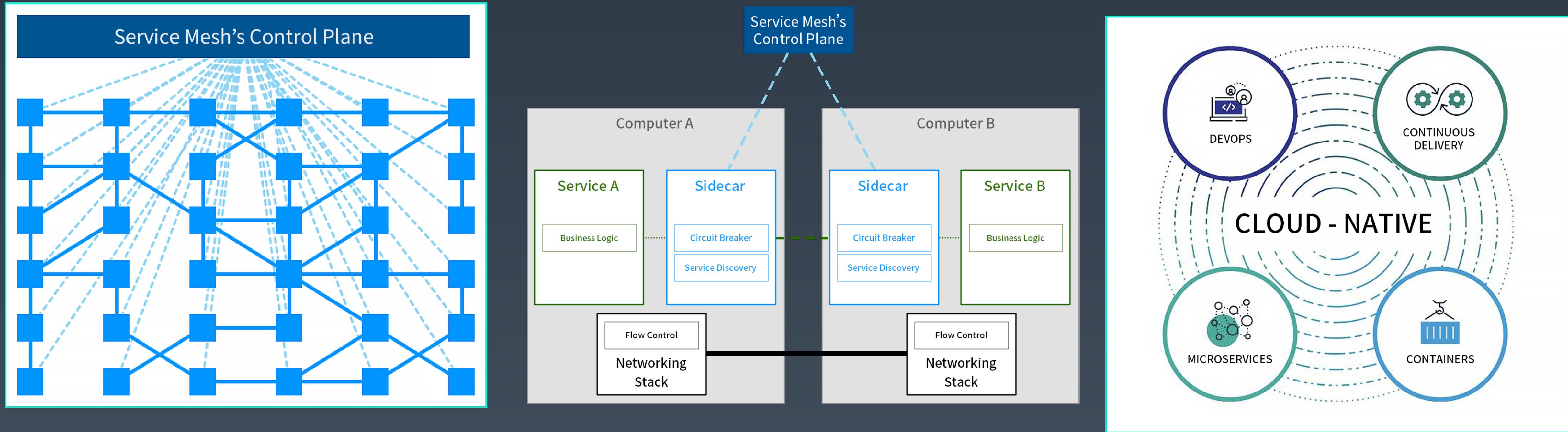
自治性

异步性

伸缩性

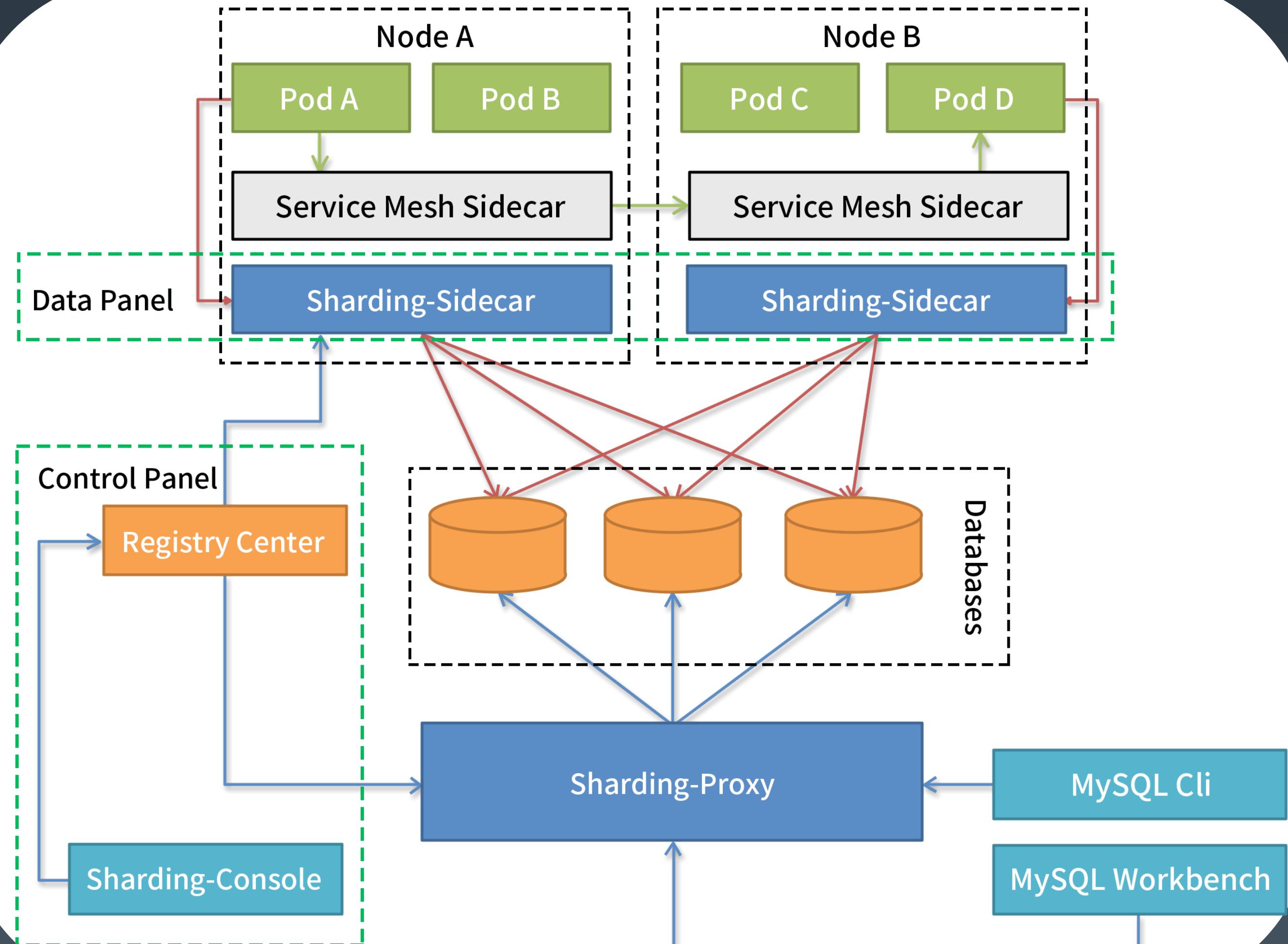
回弹性

微服务发展历程--2.服务网格与云原生



将服务间的网络通信层及其控制策略下沉到基础设施，就形成了所谓的“服务网格”技术。
通过微服务、容器化、持续交付、Devops 等技术，组成了所谓的“云原生”体系。

微服务发展历程--3.数据库网格



Level 6: Sharding-Engine (6.x+)

Level 5: Sharding-Sidecar (5.x+)

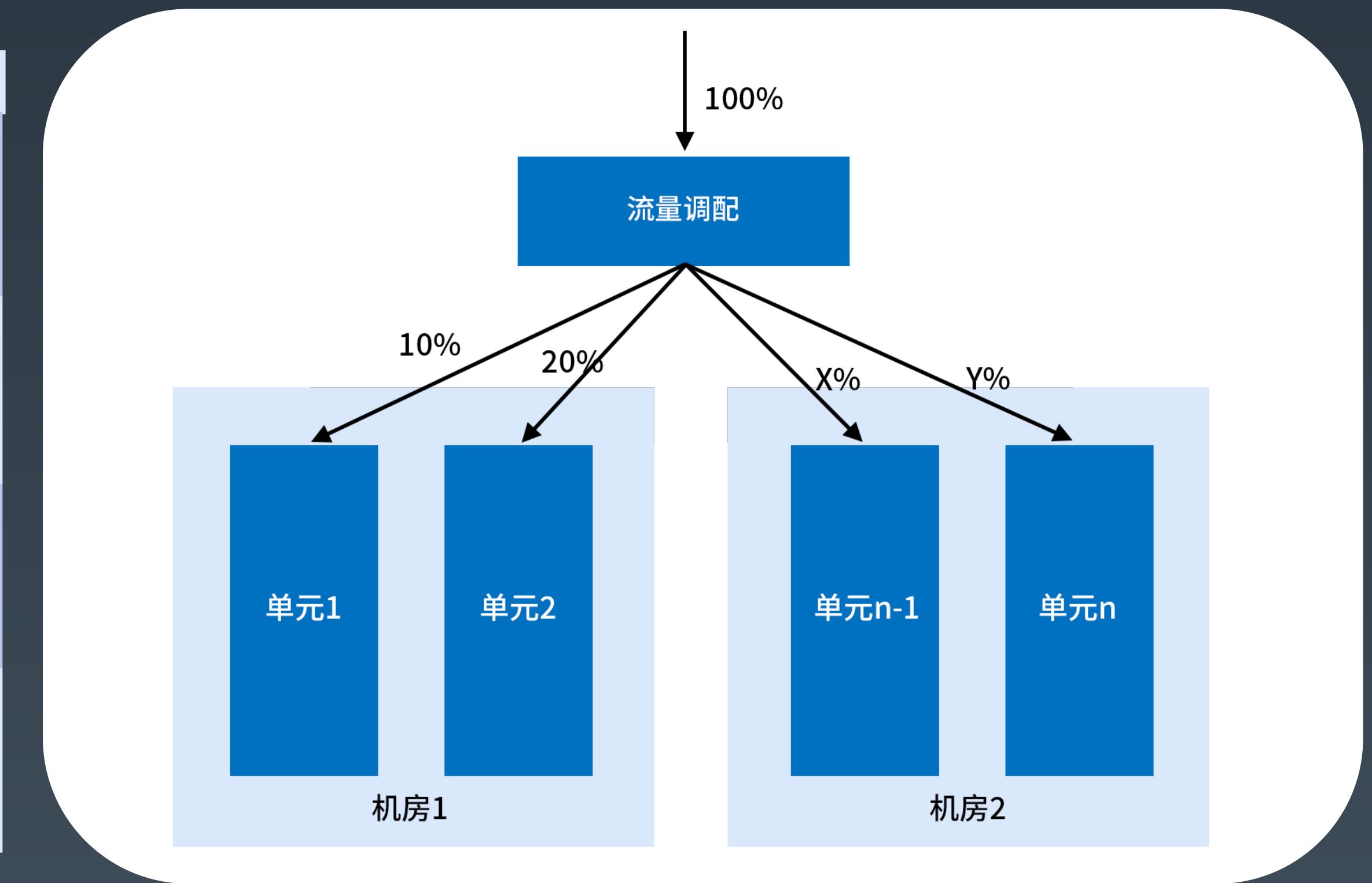
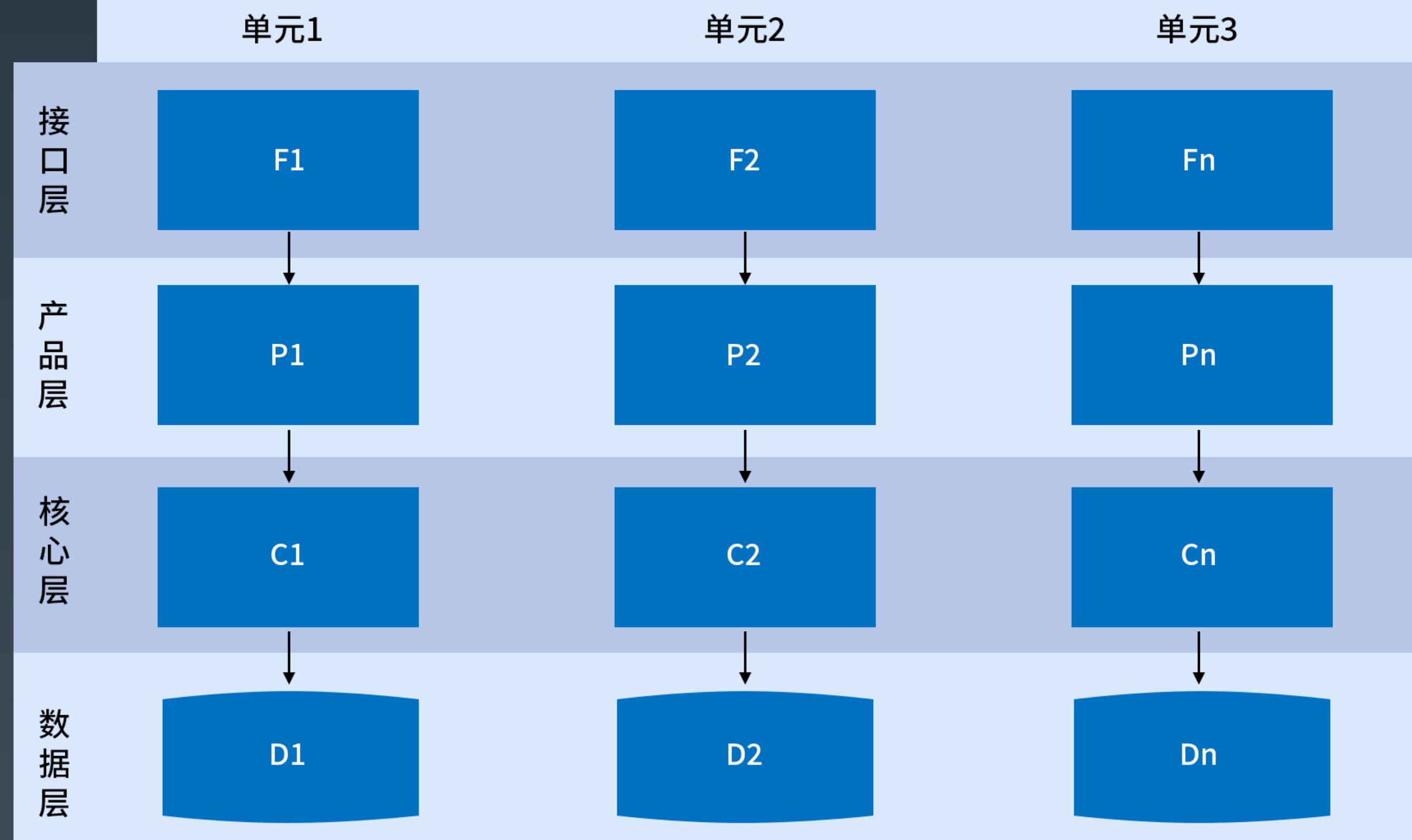
Level 4: Sharding-Scaling (4.x+)

Level 3: Sharding-Proxy中间件 (3.x+)

Level 2: Sharding-JDBC框架 (1.x+)

Level 1: MySQL数据库提供的能力

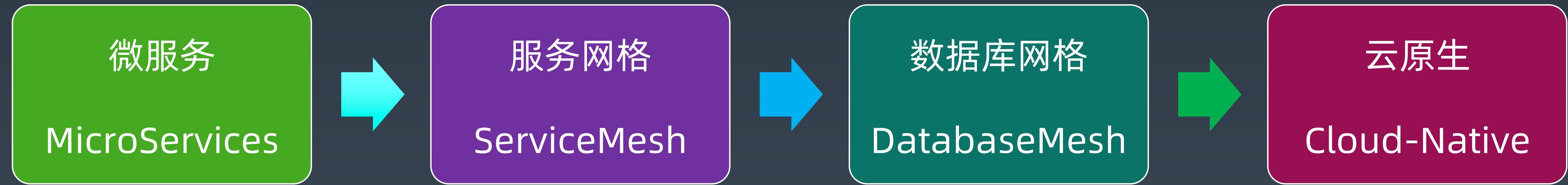
微服务发展历程--4.单元化架构



以单元为组织架构，以单元化部署为调度单位。

每个单元，是一个五脏俱全的缩小版整站，它是全能的，因为部署了所有应用；但它不是全量的，因为只能操作一部分数据。能够单元化的系统，很容易在多机房中部署，因为可以轻易地把几个单元部署在一个机房，而把另外几个部署在其他机房。通过在业务入口处设置一个流量调配器，可以调整业务流量在单元之间的比例。

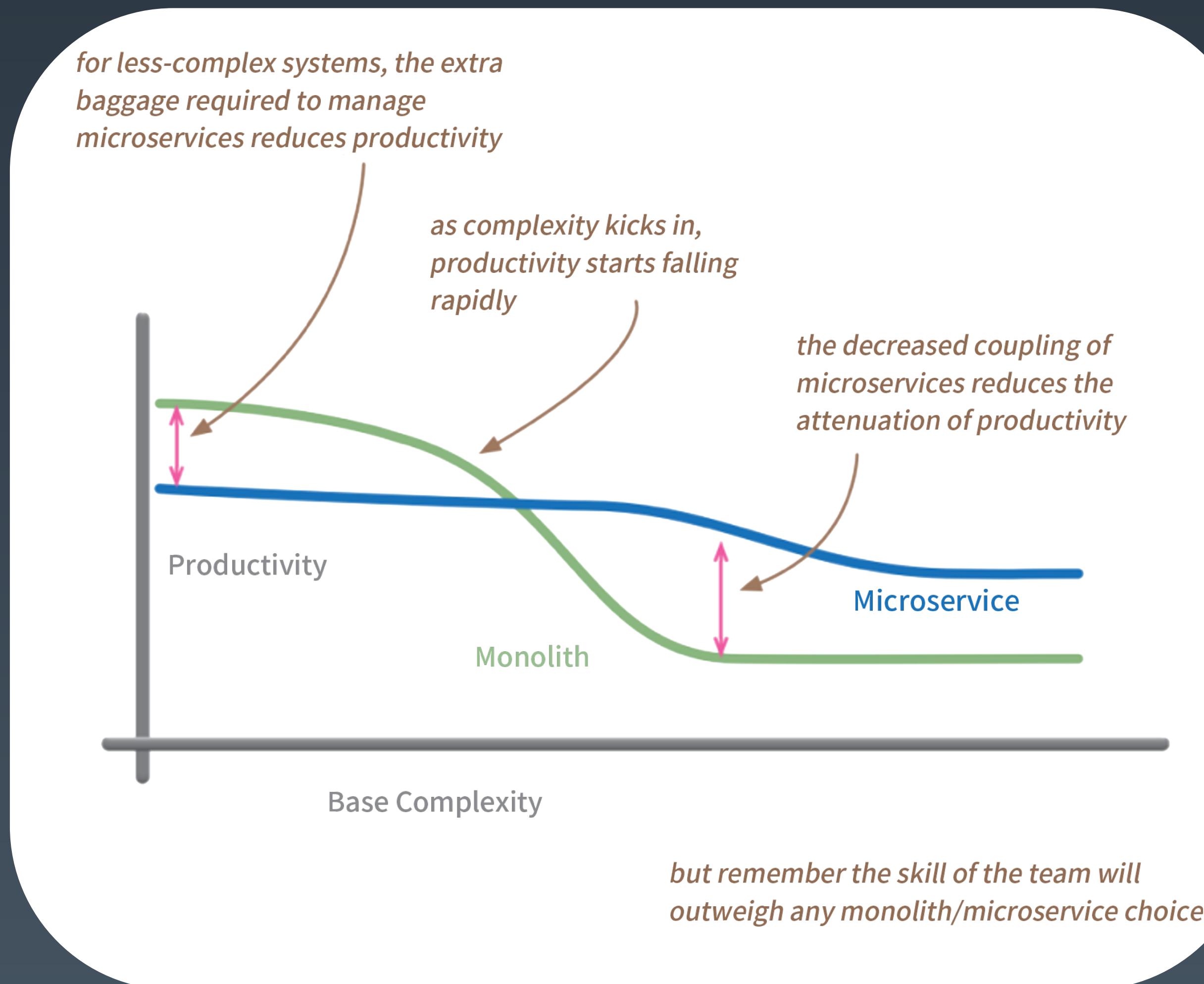
微服务发展历程



单元化架构
Cell Architecture

2. 微服务架构应用场景

什么时候用微服务呢？



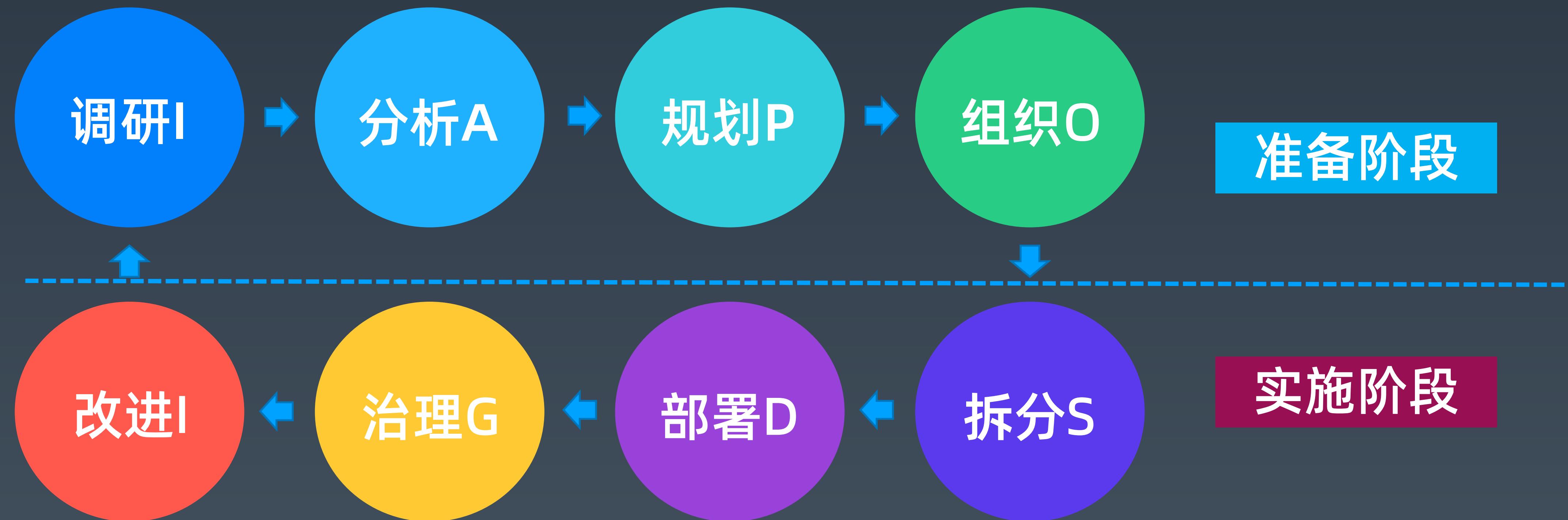
- ✓ 微服务应用在复杂度低的情况下，生产力反而比单体架构低
- ✓ 在复杂度高的地方，情况恰恰相反
- ✓ 随着复杂度升高，单体架构的生产力快速下降，而微服务相对平稳，为什么？
拆分本身是非常复杂的。

什么时候用微服务呢？

- 大规模复杂业务系统的架构升级与中台建设



怎么应用微服务架构-161



3. 微服务架构最佳实践

六大最佳实践

遗留系统改造

自动化管理

恰当粒度拆分

分布式事务

扩展立方体

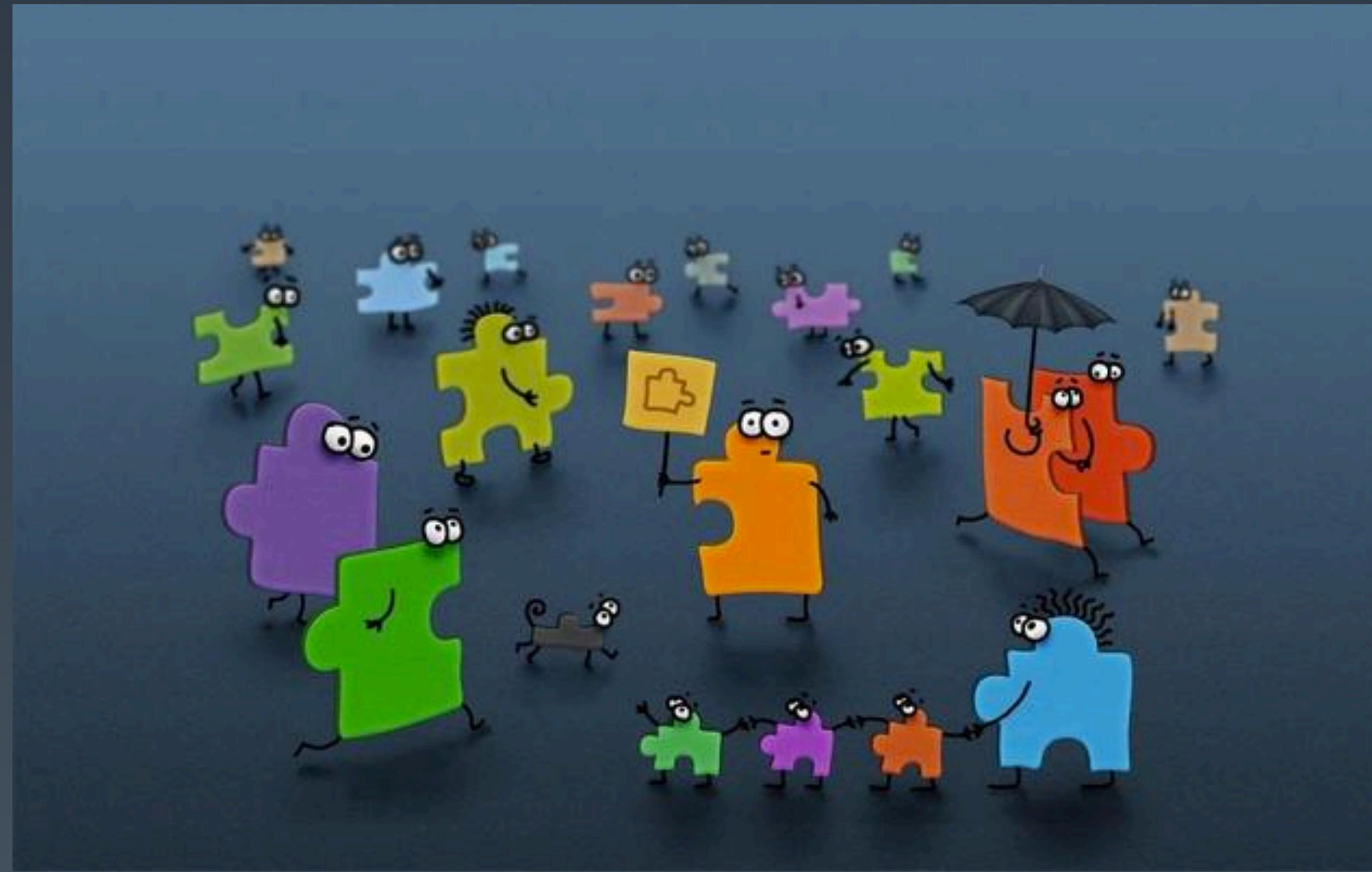
完善监控体系

最佳实践-01



- ①功能剥离、数据解耦
- ②自然演进、逐步拆分
- ③小步快跑、快速迭代
- ④灰度发布、谨慎试错
- ⑤提质量线、还技术债

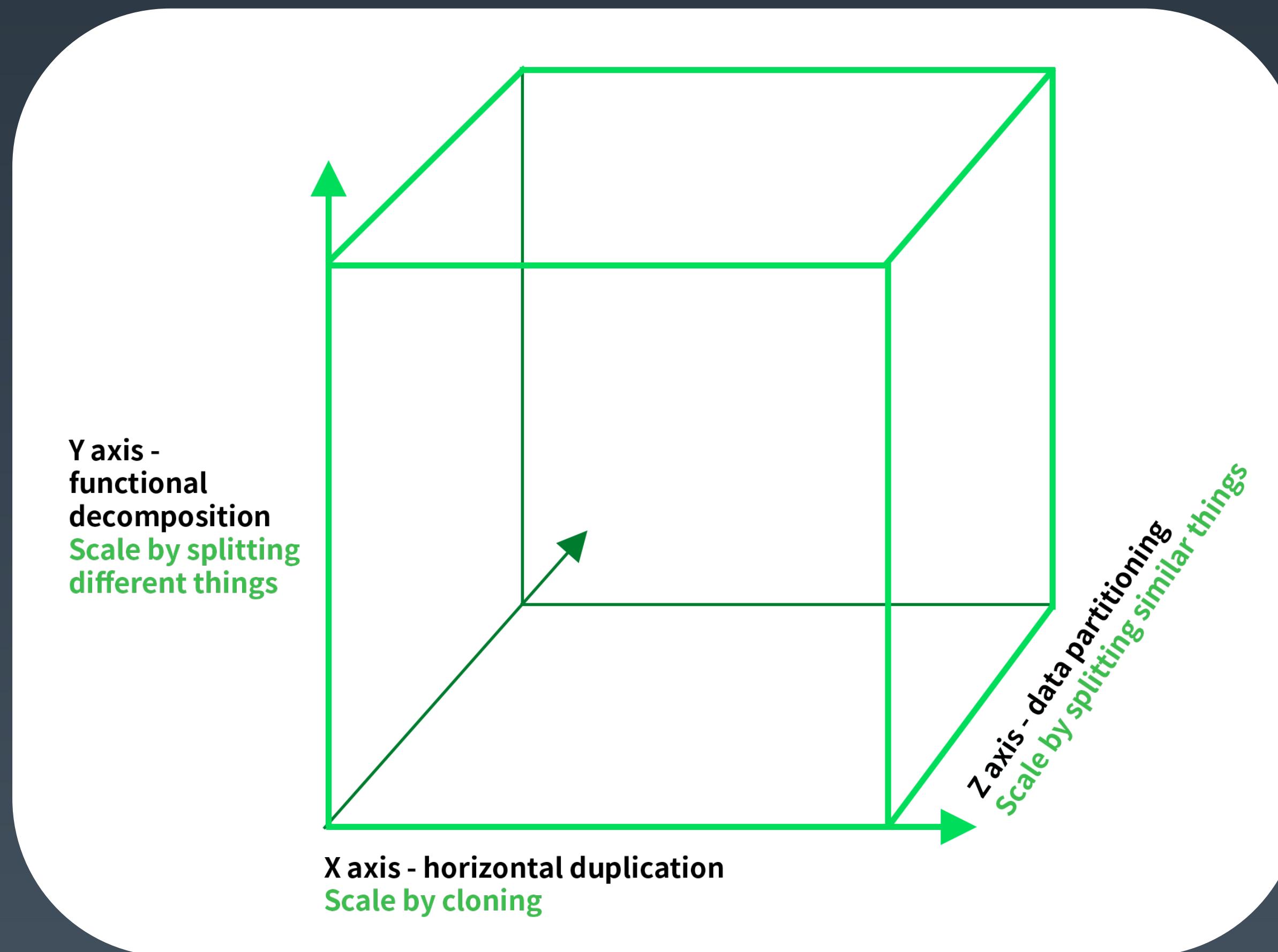
最佳实践-02



拆分原则：

1. 高内聚低耦合
2. 不同阶段拆分要点不同

最佳实践-03



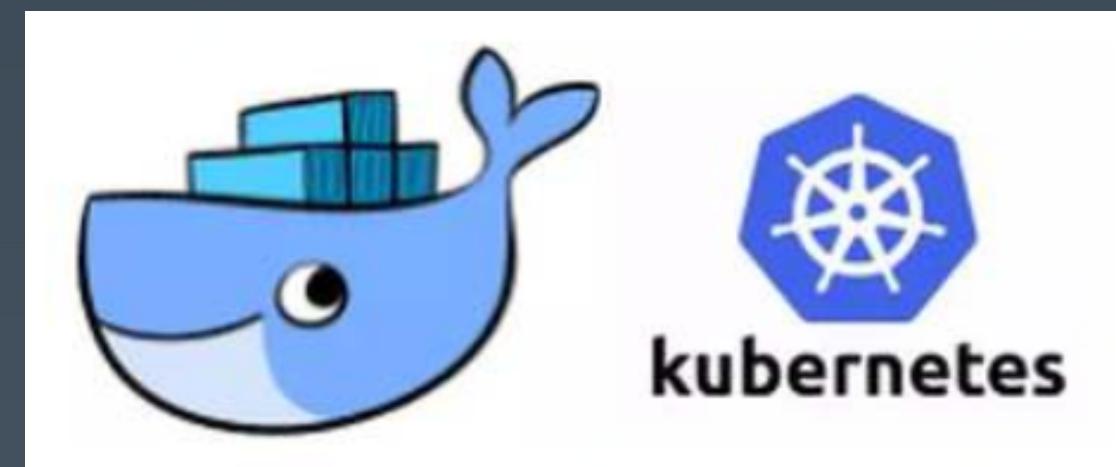
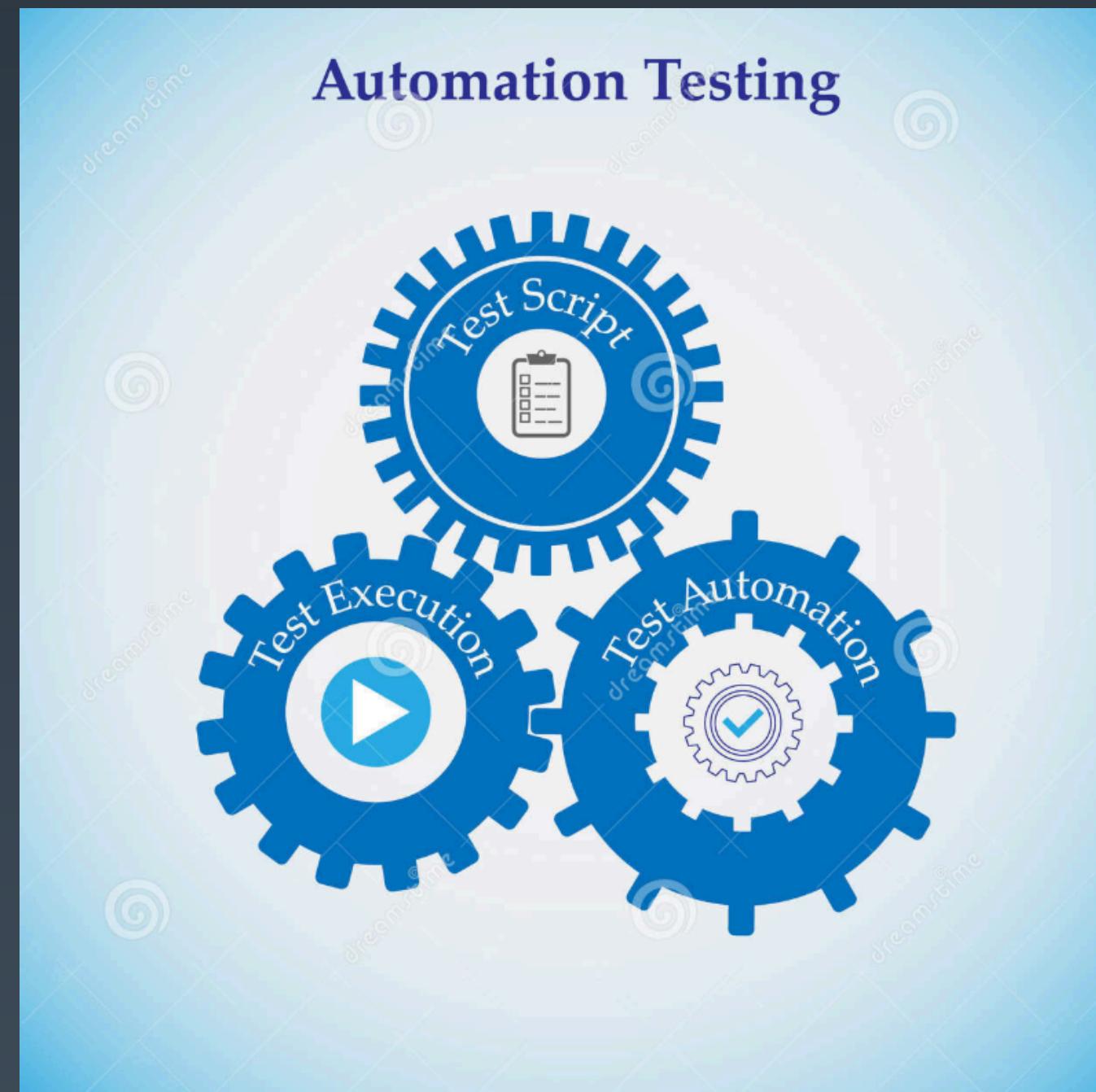
扩展立方体：

1. 水平复制：复制系统
2. 功能解耦：拆分业务
3. 数据分区：切分数据

单元化架构的基础

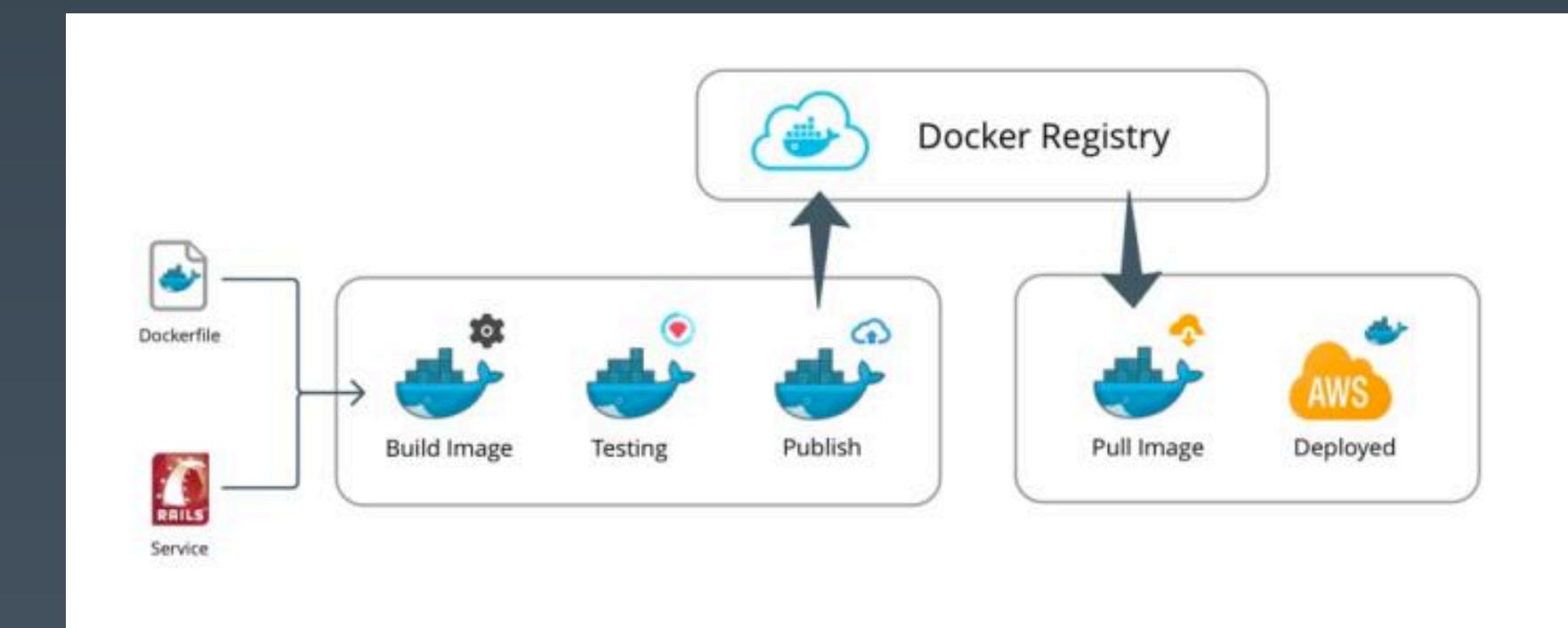
+特性开关 +容错设计

最佳实践-04

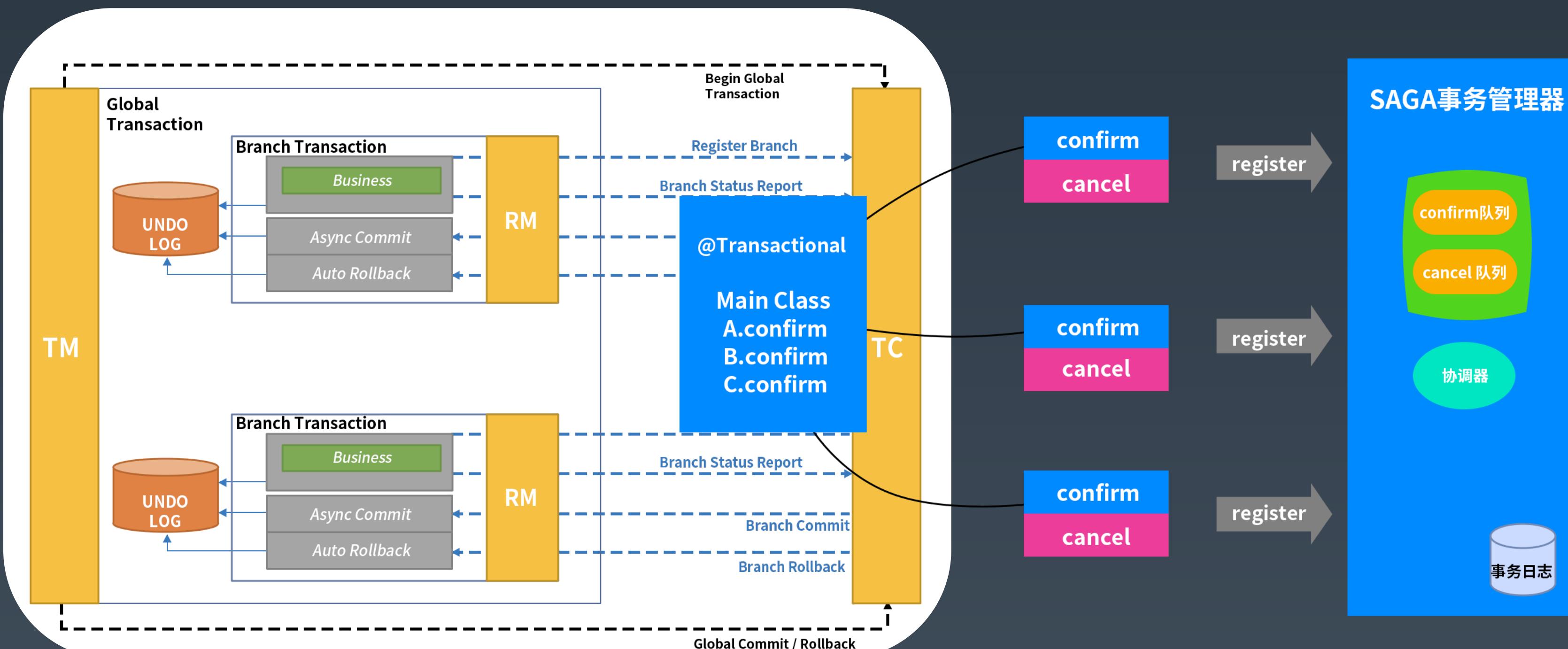


- 自动化测试
- 自动化部署
- 自动化运维

降低服务拆分带来的复杂性
提升测试、部署、运维效率

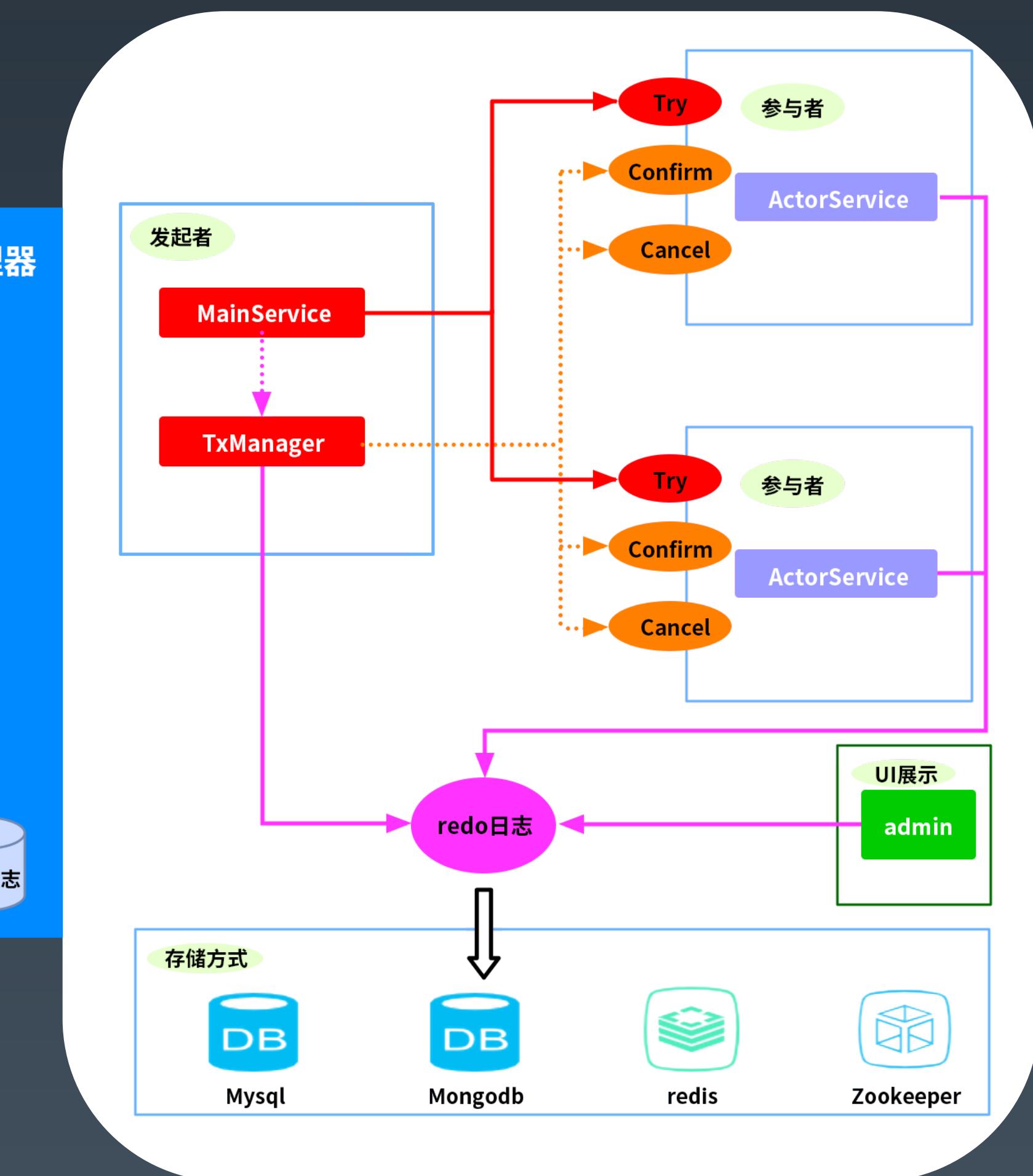


最佳实践-05

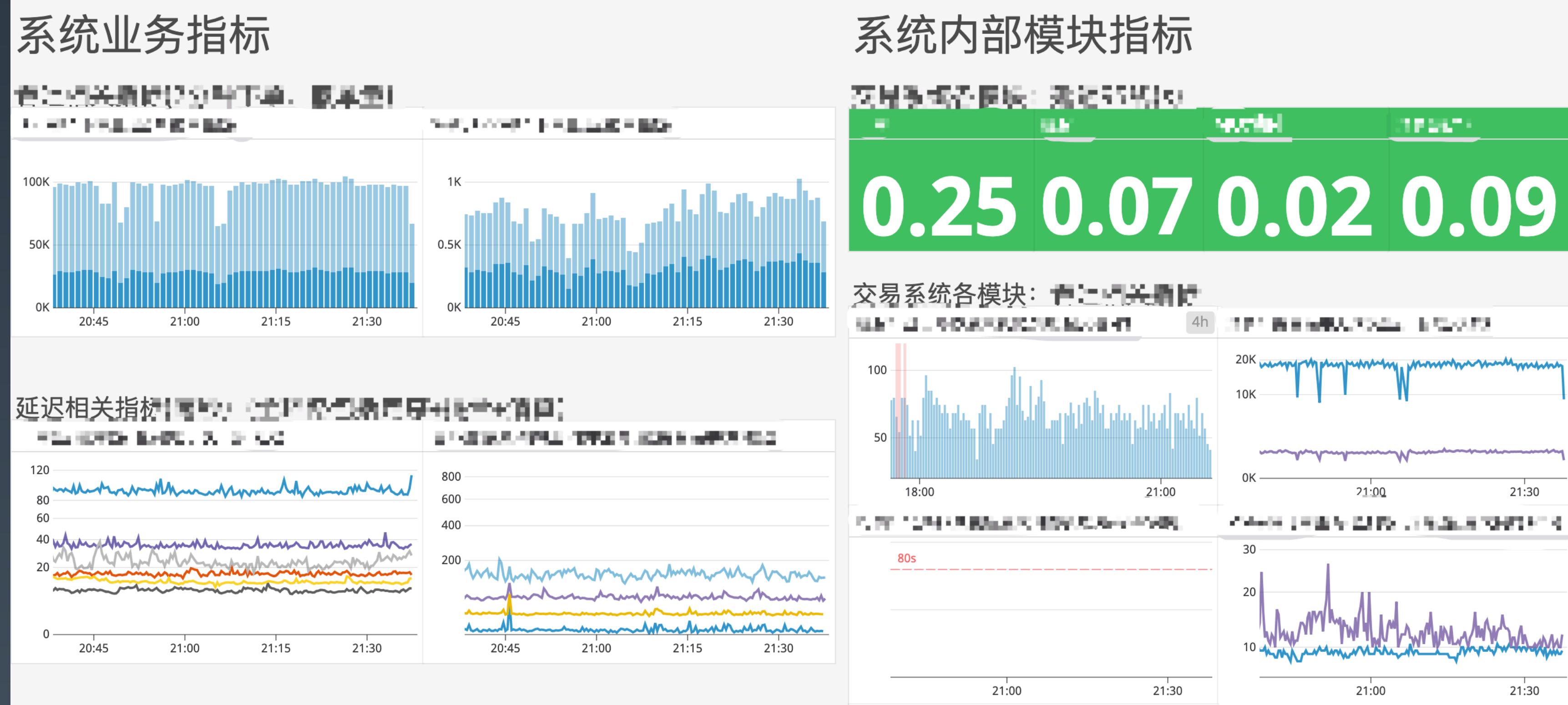


幂等/去重/补偿

慎用分布式事务!



最佳实践-06



+Devops +SRE

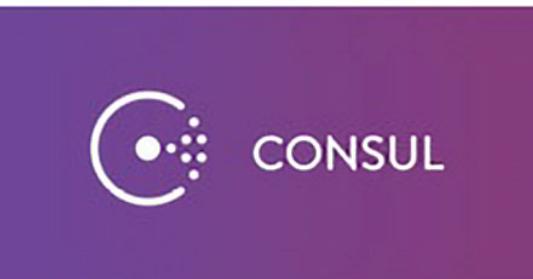
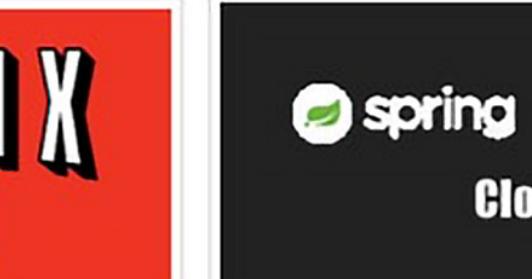
- 监控与运维：
1. 业务监控
 2. 系统监控
 3. 容量规划
 4. 报警预警
 5. 运维流程
 6. 故障处理

稳定性建设

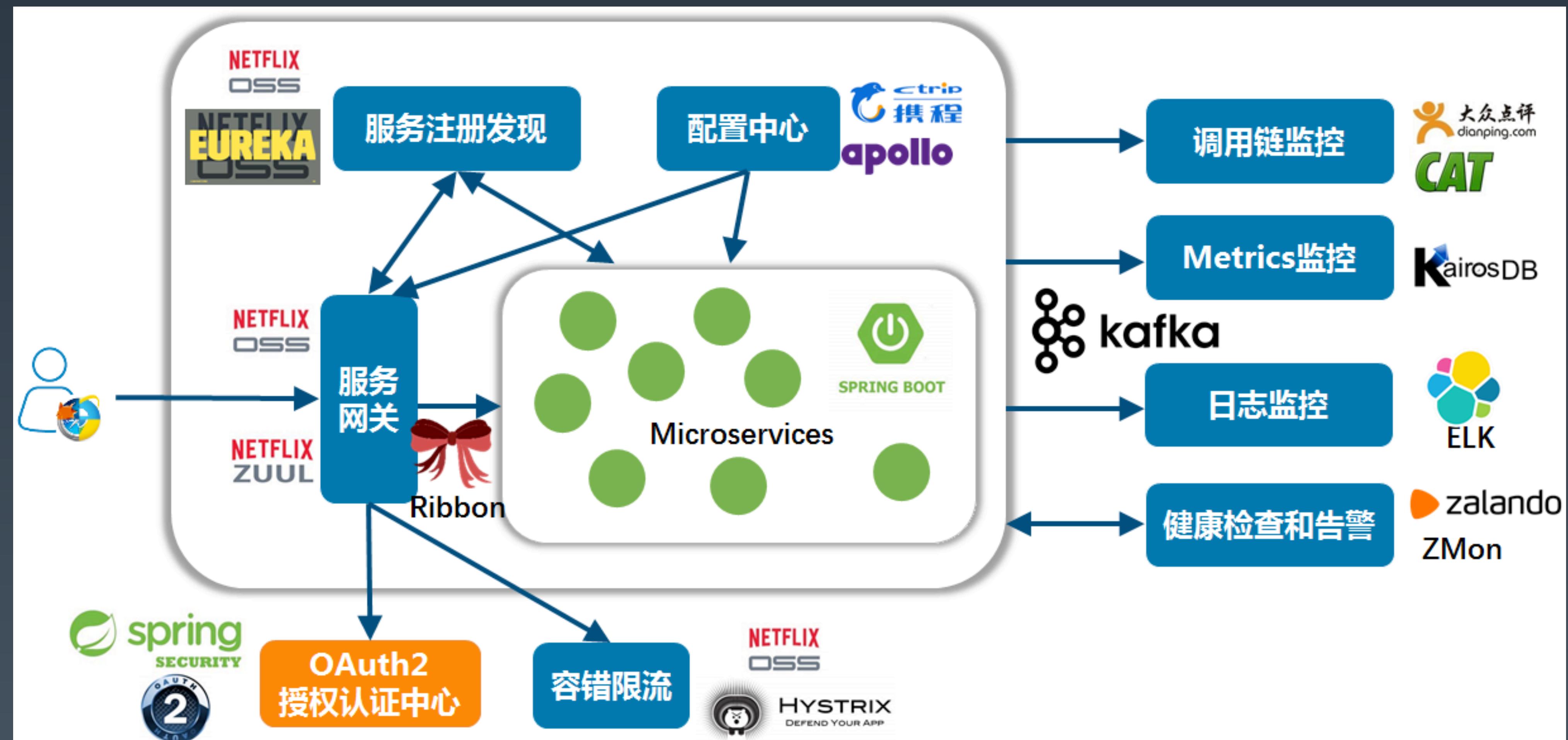
4. Spring Cloud 技术体系

Spring Cloud

Spring Cloud生态

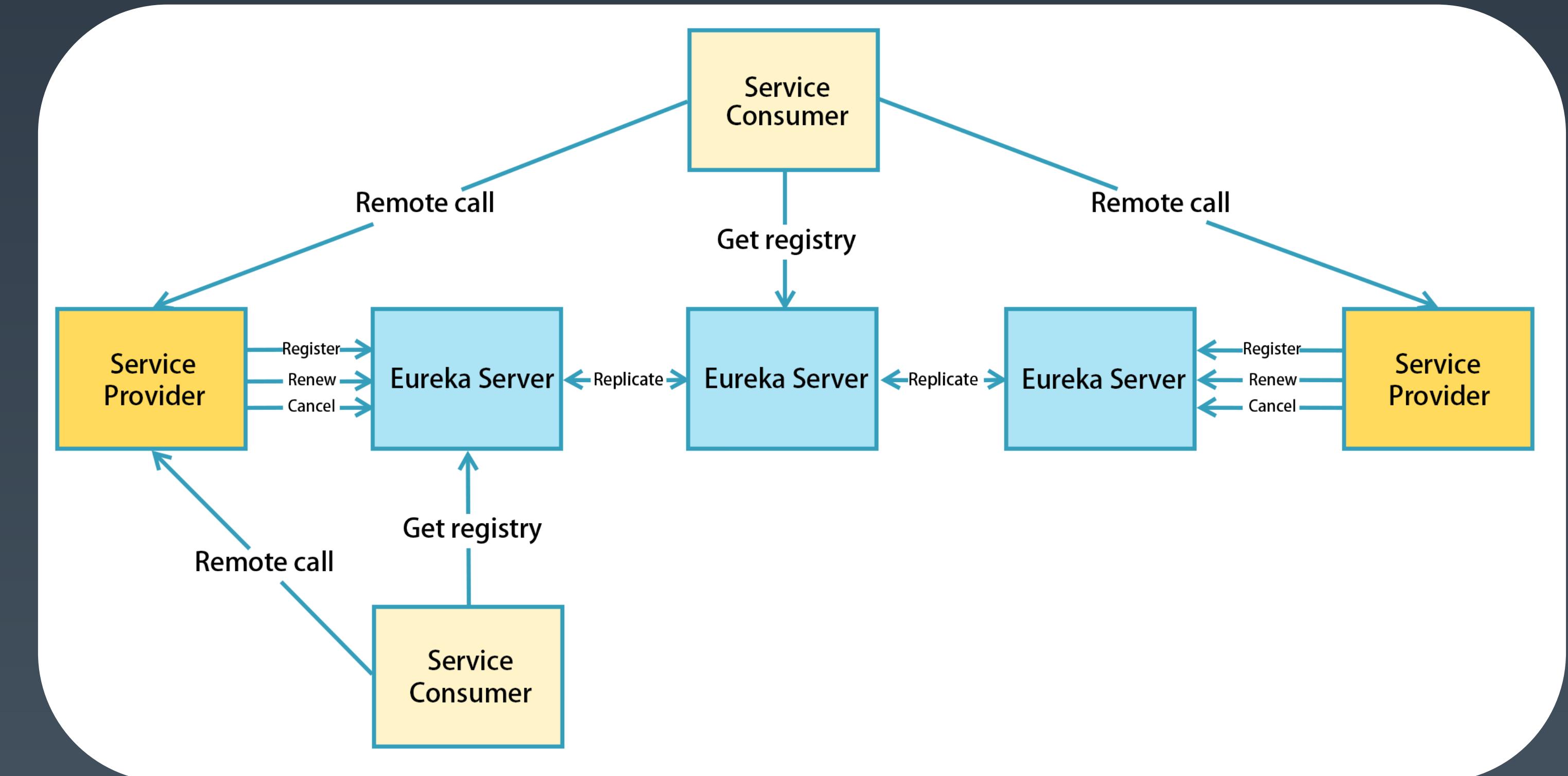
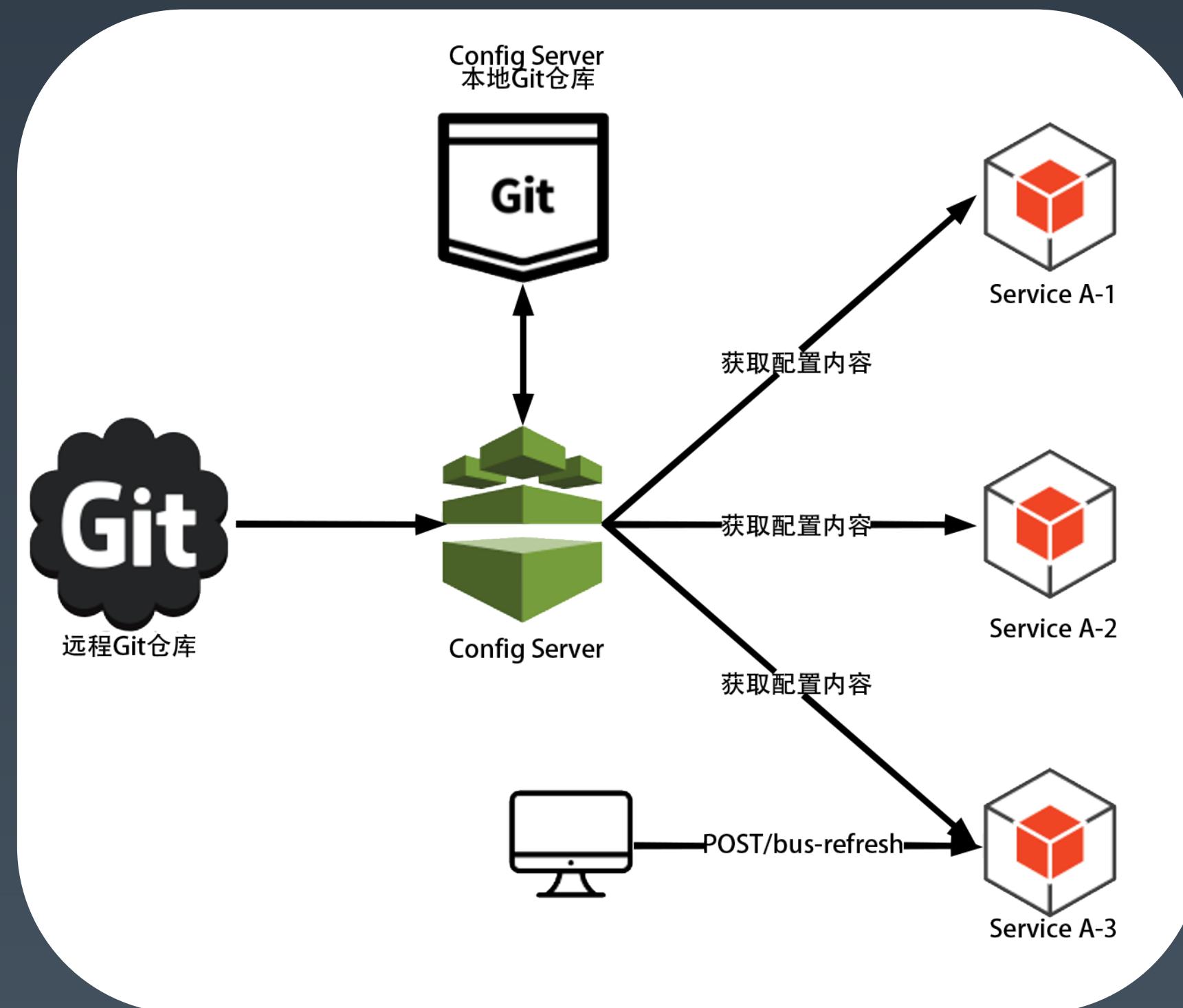
 Spring Cloud Config Spring 配置管理工具包, 让你可以把配置放到远程服务器, 集中化管理集群配置, 目前支持本地存储、Git以及Subversion。	 Spring Cloud Bus Spring 事件、消息总线, 用于在集群(例如, 配置变化事件)中传播状态变化, 可与Spring Cloud Config联合实现热部署。	 Eureka Netflix 云端服务发现, 一个基于REST的服务, 用于定位服务, 以实现云端中间层服务发现和故障转移。	 Hystrix Netflix 熔断器, 容错管理工具, 旨在通过熔断机制控制服务和第三方库的节点, 从而对延迟和故障提供更强大的容错能力。	 Zuul Netflix Zuul是在云平台上提供动态路由, 监控, 弹性, 安全等边缘服务的框架。Zuul相当于是设备和Netflix流应用的Web网站后端所有请求的前门。	 Archaius Netflix 配置管理API, 包含一系列配置管理API, 提供动态类型化属性、线程安全配置操作、轮询框架、回调机制等功能。	 Spring Cloud Starters Pivotal Spring Boot式的启动项目, 为Spring Cloud提供开箱即用的依赖管理。
 Consul HashiCorp 封装了Consul操作, consul是一个服务发现与配置工具, 与Docker容器可以无缝集成。	 Spring Cloud Sleuth Spring 日志收集工具包, 封装了Dapper和log-based追踪以及Zipkin和HTrace操作, 为SpringCloud应用实现了一种分布式追踪解决方案。	 Spring Cloud Zookeeper Spring 操作Zookeeper的工具包, 用于使用zookeeper方式的服务发现和配置管理。	 Spring Cloud Stream Spring 数据流操作开发包, 封装了与Redis, Rabbit, Kafka等发送接收消息。	 Ribbon Netflix 提供云端负载均衡, 有多种负载均衡策略可供选择, 可配合服务发现和断路器使用。	 Feign OpenFeign Feign是一种声明式、模板化的HTTP客户端。	 Spring Cloud Cluster Spring 提供Leadership选举, 如:Zookeeper, Redis, Hazelcast, Consul等常见状态模式的抽象和实现。

Spring Cloud



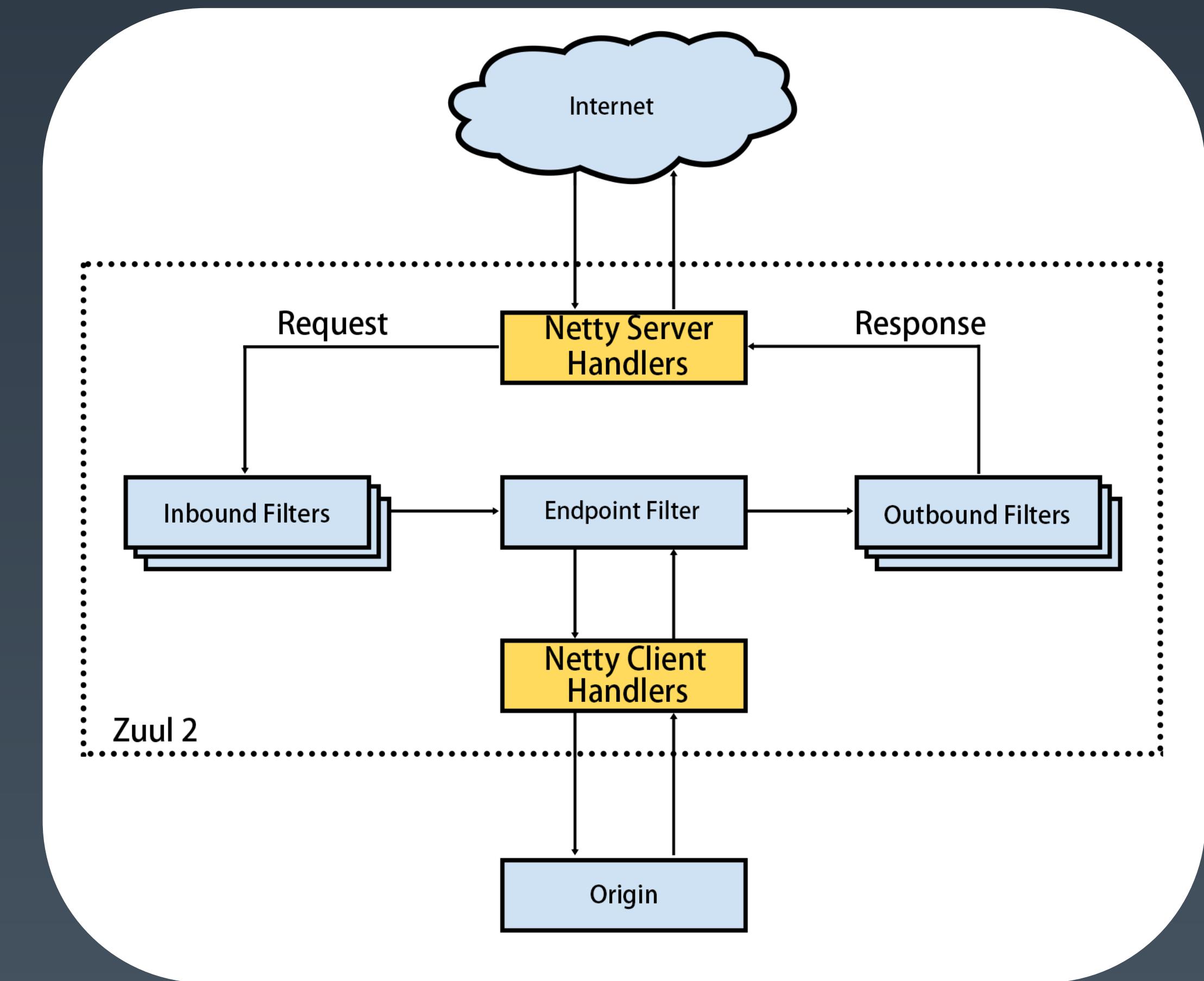
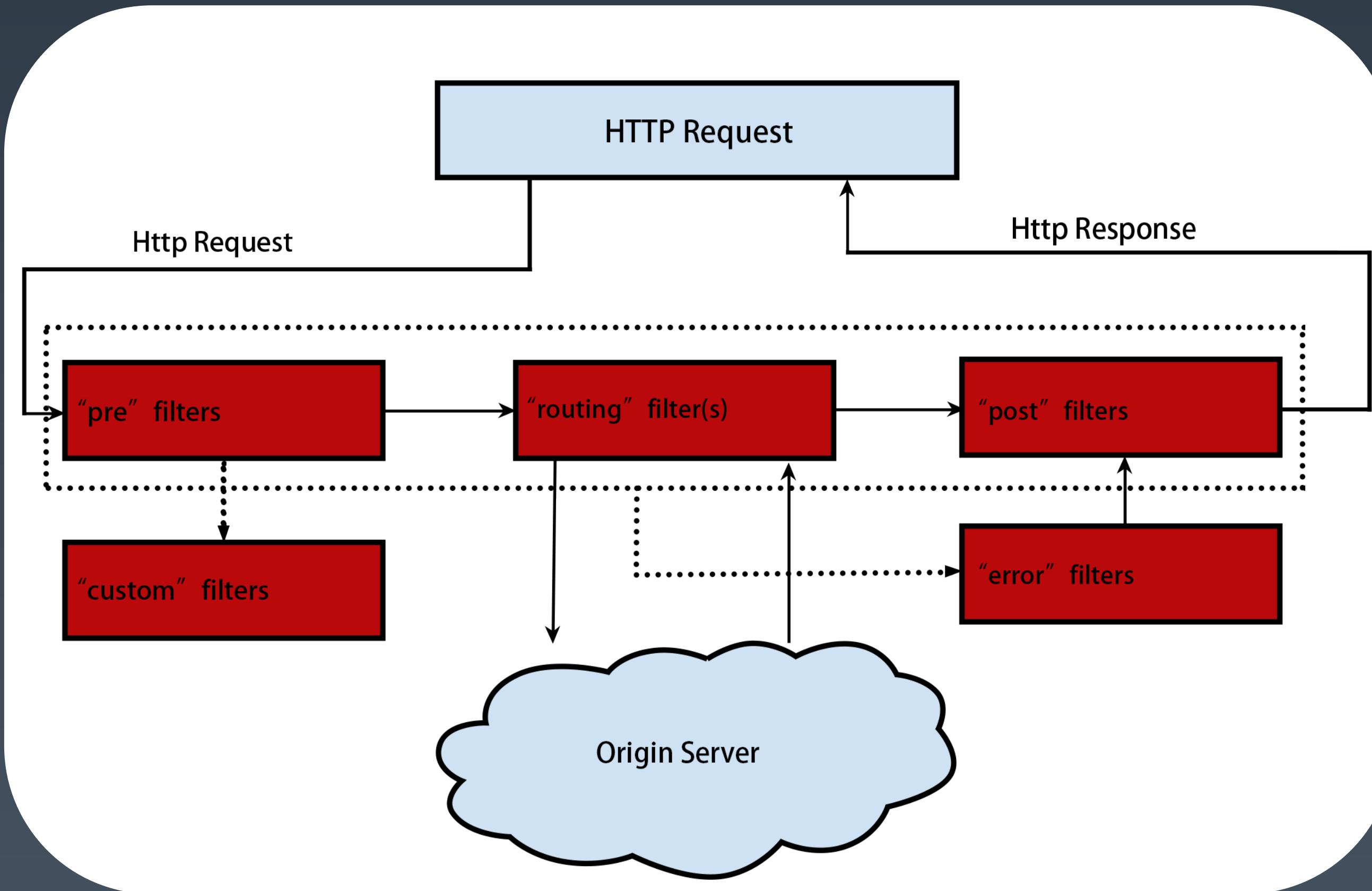
Spring Cloud

- Config/Eureka/Consul



Spring Cloud

- Zuul/Zuul2/Spring Cloud Gateway



Spring Cloud

- Feign/Ribbon

Feign 的核心功能就是，作为 HTTP Client 访问 REST 服务接口。

优势在于：

1、全都基于注解，简单方便

2、跟 XXTemplate 一样，内置了简化操作，OOP

3、跟其他组件，ribbon，hytrix 联合使用

Ribbon 是用于云环境的一个客户端内部通信（IPC）库。

特性：

1、负载均衡

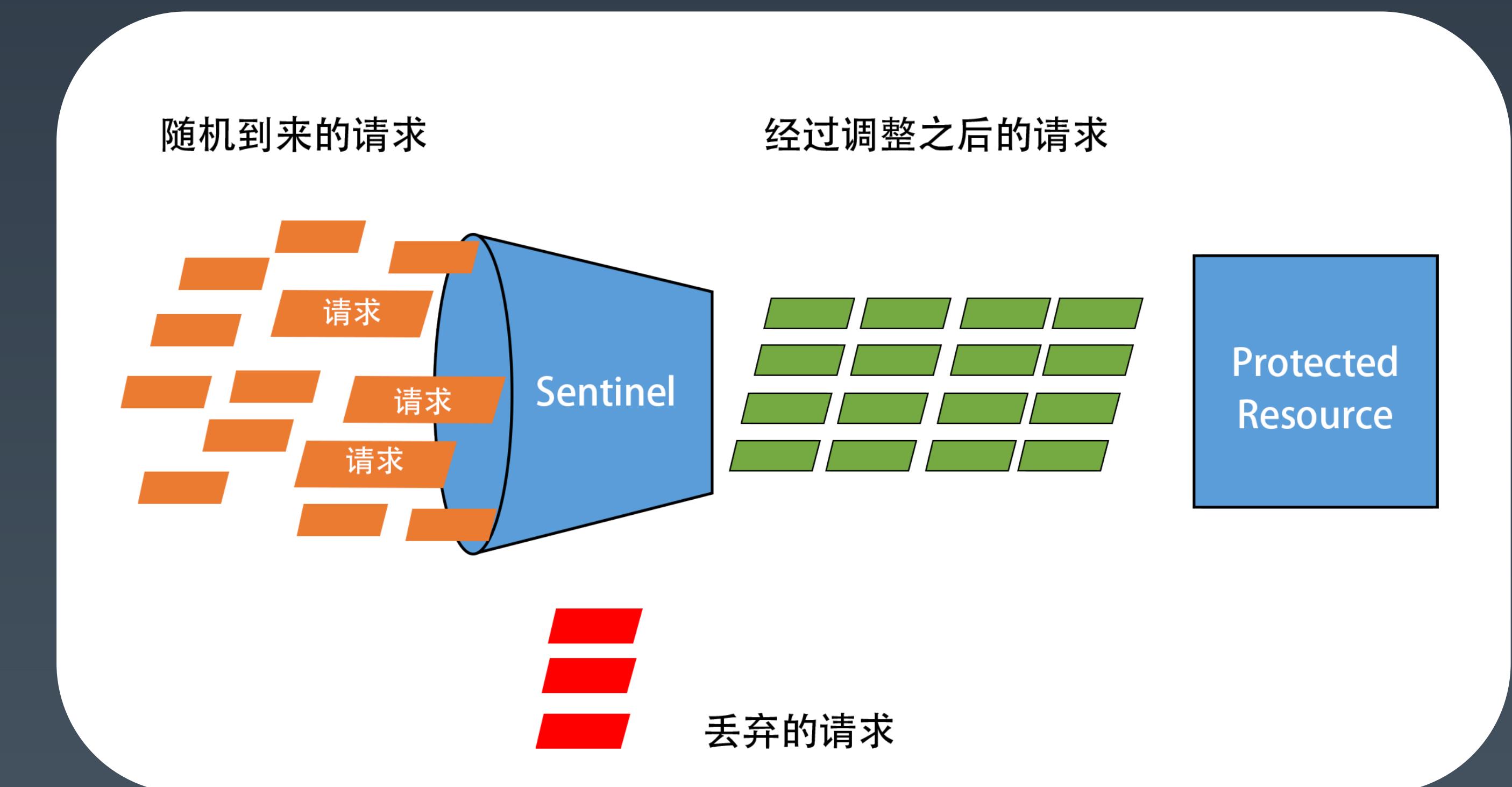
2、容错

3、多协议支持（HTTP, TCP, UDP），特别是异步和反应式下

4、缓存和批处理

Spring Cloud

- Hystrix/Alibaba Sentinel/Reslient4j



5. 微服务相关框架与工具

相关工具-APM

APM：应用性能监控

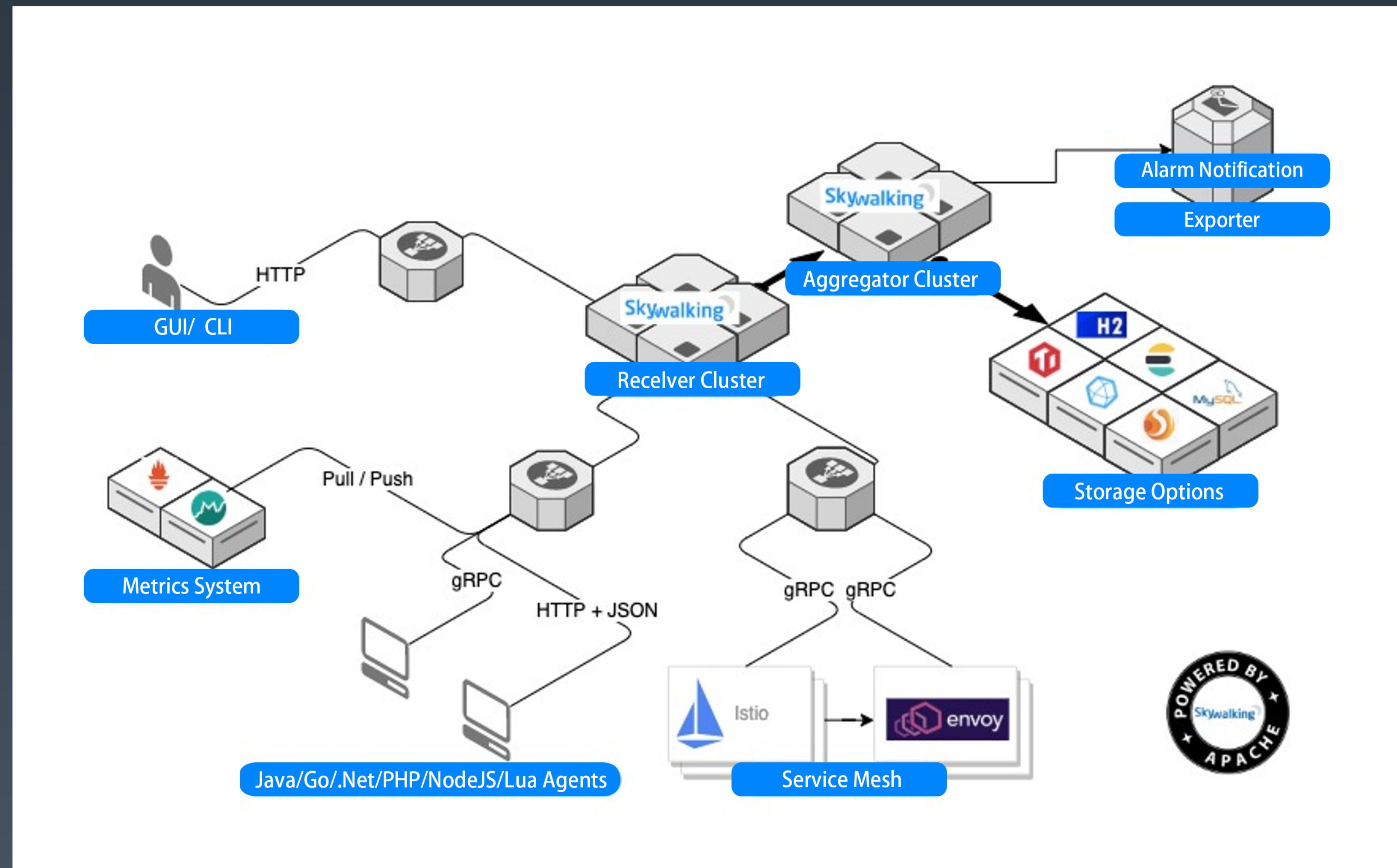
- Apache Skywalking
- Pinpoint
- Zipkin
- Jaeger

监控

- ELK
- prometheus+Grafana
- MQ+时序数据库
(InfluxDB/openTSDB 等)

	pinpoint	zipkin	jaeger	skywalking
OpenTracing兼容	否	是	是	是
客户端支持语言	java、php	java,c#,go,php等	java,c#,go,php等	java,NET Core,NodeJS and PHP
存储	hbase	ES, mysql,Cassandra, 内存	ES, kafka,Cassandra, 内存	ES, H2, mysql, TiDB, sharding sphere
传输协议支持	thrift	http,MQ	udp/http	gRPC
ui丰富程度	高	低	中	中
实现方式-代码侵入性	字节码注入,无侵入	拦截请求, 侵入	拦截请求, 侵入	字节码注入, 无侵入
扩展性	低	高	高	中
trace查询	不支持	支持	支持	支持
告警支持	支持	不支持	不支持	支持
jvm监控	支持	不支持	不支持	支持
性能损失	高	中	中	低

相关工具-APM: Apache Skywalking



相关工具-APM

可观测性：

- Logging
- Tracing
- Metrics

标准：

- OpenTracing
- OpenSensus
- **OpenTelemetry**



OPEN TRACING

- One “vertical” (Tracing)
Users want only one dependency
- One “layer” (API)
- “Looser” coupling (small scope)
- Lots of languages (~12)
- Broad adoption



OpenCensus

- Many “verticals” (Tracing, Metrics)
- Many “layers” (API, impl, infra)
- “Tighter” coupling (framework-y)
Users and vendors want flexibility
- Many languages (5 in beta)
- Broad adoption



OPEN TRACING

- 370 contributors
- 5360 stars on GitHub
- 100s of supported integrations
with OSS libraries and
frameworks



OpenCensus

- 390 contributors
- 3392 stars on GitHub
- Backed by Google, Microsoft,
Omnition, Postmates,
Dynatrace, Shopify (soon)

相关工具-权限控制

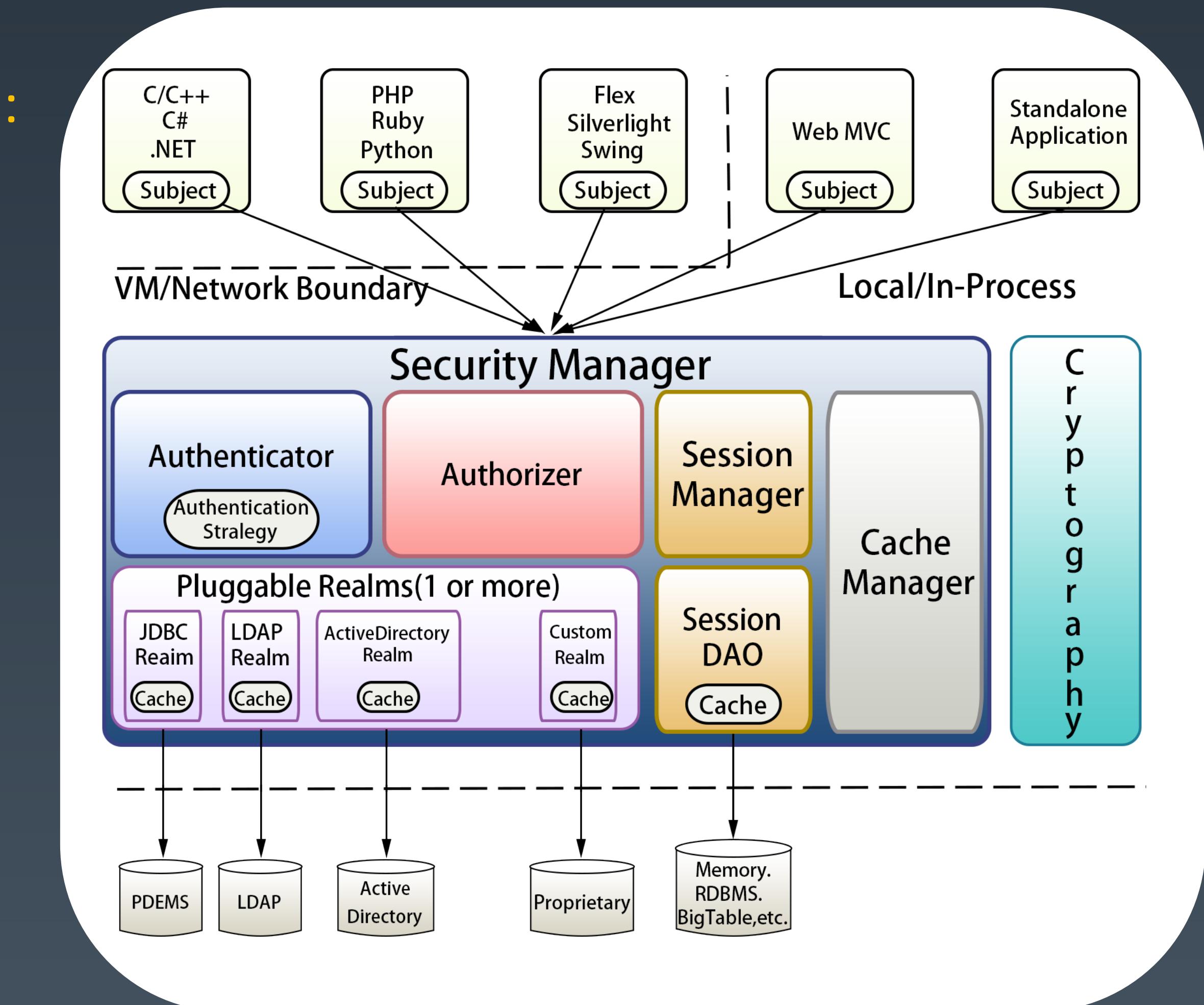
最核心的3A（其他：资源管理、安全加密等）：

- Authc: Authentication
- Authz: Authorization
- Audit

CAS+SSO(TGT、ST)

JWT/Token, OAuth2.0

SpringSecurity, Apache Shiro



相关工具-数据处理

1、读写分离与高可用 HA:

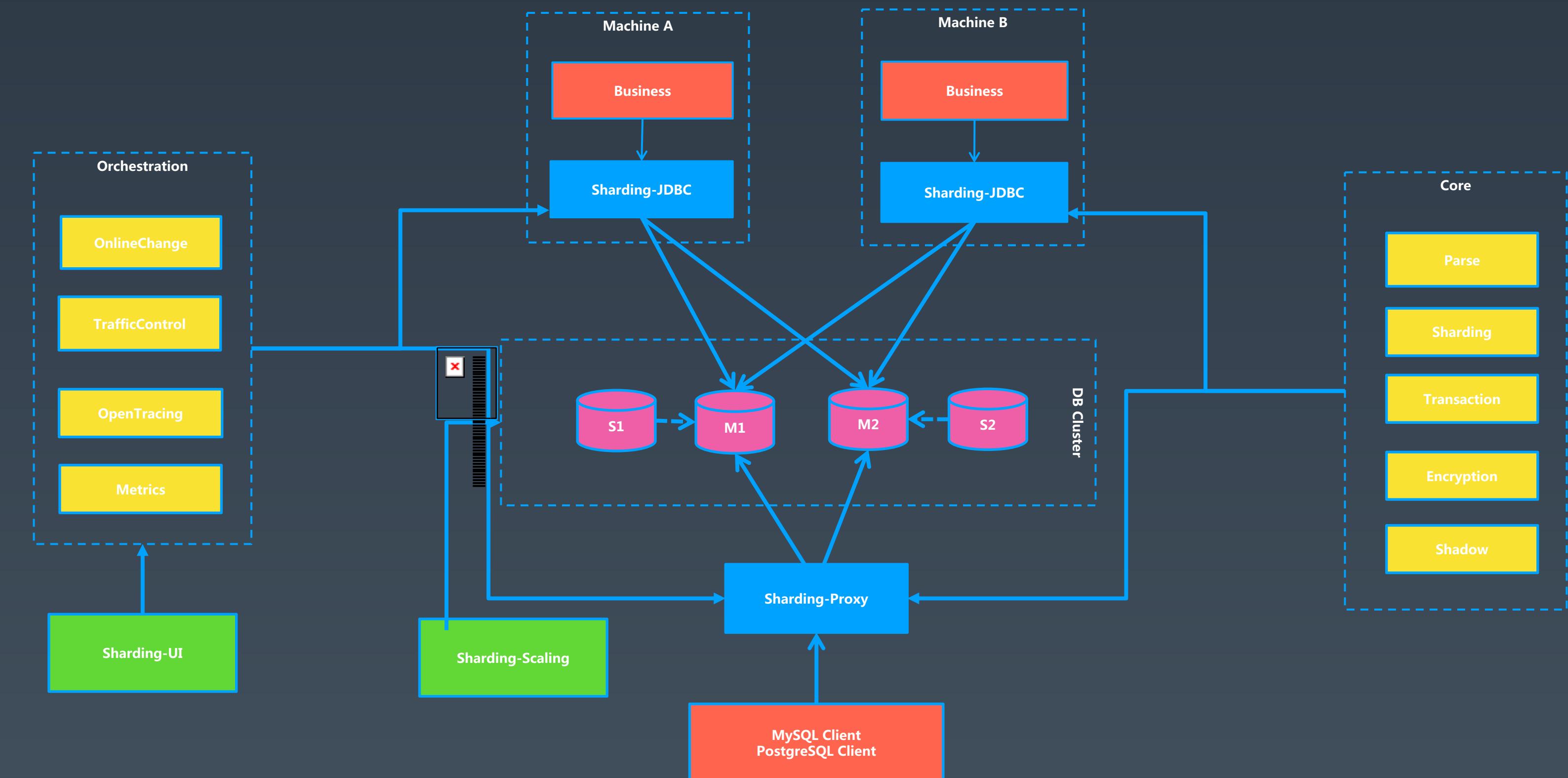
2、分库分表 Sharding:

3、分布式事务 DTX:

4、数据迁移 Migration:

5、数据集群扩容 Scaling:

6、数据操作审计 Audit:

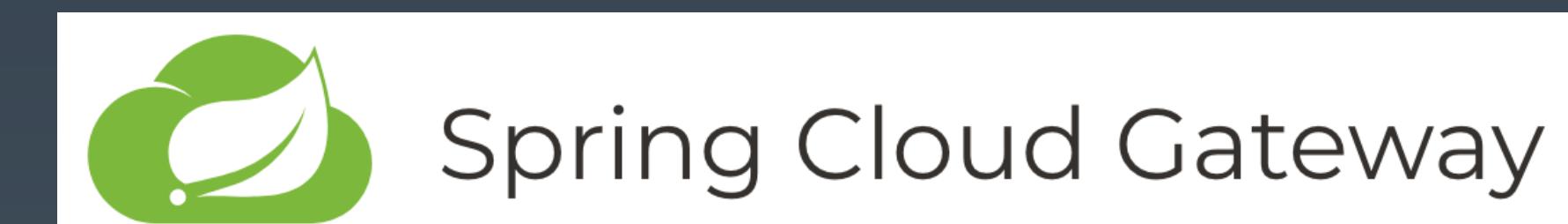


相关工具-网关与通信

1、流量网关与 WAF (Nginx/OR/Kong/Apisix)

2、业务网关 (Zuul/Zuul2/SCG/Soul)

3、REST 与其他协议之争 (websocket/actor/rsocket/mq...)



6. 总结回顾与作业实践

第 20 课总结回顾

微服务架构发展历程

微服务使用场景与最佳实践

Spring Cloud 技术体系

微服务相关技术与工具

第 20 课作业实践

- 1、（选做）进度快的，把前几次课的选做题做做。
- 2、（选做）进度慢的，把前几次课的必做题做做。
- 3、（选做）学霸和课代表，，考虑多做做挑战题。
- 4、（挑战☆）对于不断努力前行的少年：
 - 1) 把你对技术架构演进的认识，做一个总结。
 - 2) 把你对微服务的特点，能解决什么问题，适用于什么场景，总结一下。
 - 3) 画一个脑图，总结你能想到的微服务相关技术框架和中间件，想想都有什么作用。
 - 4) 思考（☆☆）：微服务架构是否能应用到你最近接触或负责的业务系统，如何引入和应用，困难点在什么地方。
 - 5) 研究（☆☆☆）：学习和了解 Spring Cloud 体系，特别是 Netflix 和 Alibaba 套件，画出他们的体系结构图。

