OSC 源创会

# DerbySoft 微服务架构实践

## 2017 开源中国源创会（重庆）

朱攀

# 背景介绍

- 朱攀，2007年加入德比软件，负责公司数据对接平台的基础设施和基础服务建设。

- 德比软件为全球酒店集团及其分销渠道提供数据服务，客户包括全球重要地区的顶级分销渠道，在线旅行社，垂直搜索引擎，批发商以及众多大型旅游经销商（如： Booking.com，Expedia，Google， Facebook，Ctrip 等）。

- 在 AWS 全球 16 个可用区里有 2000 + 个服务节点，每天处理 10 亿 + API 调用；

# 内容大纲

- 什么是微服务
- 德比软件实施微服务思考和解决的问题详解
- 实施微服务的收益总结
- 实施微服务的挑战总结

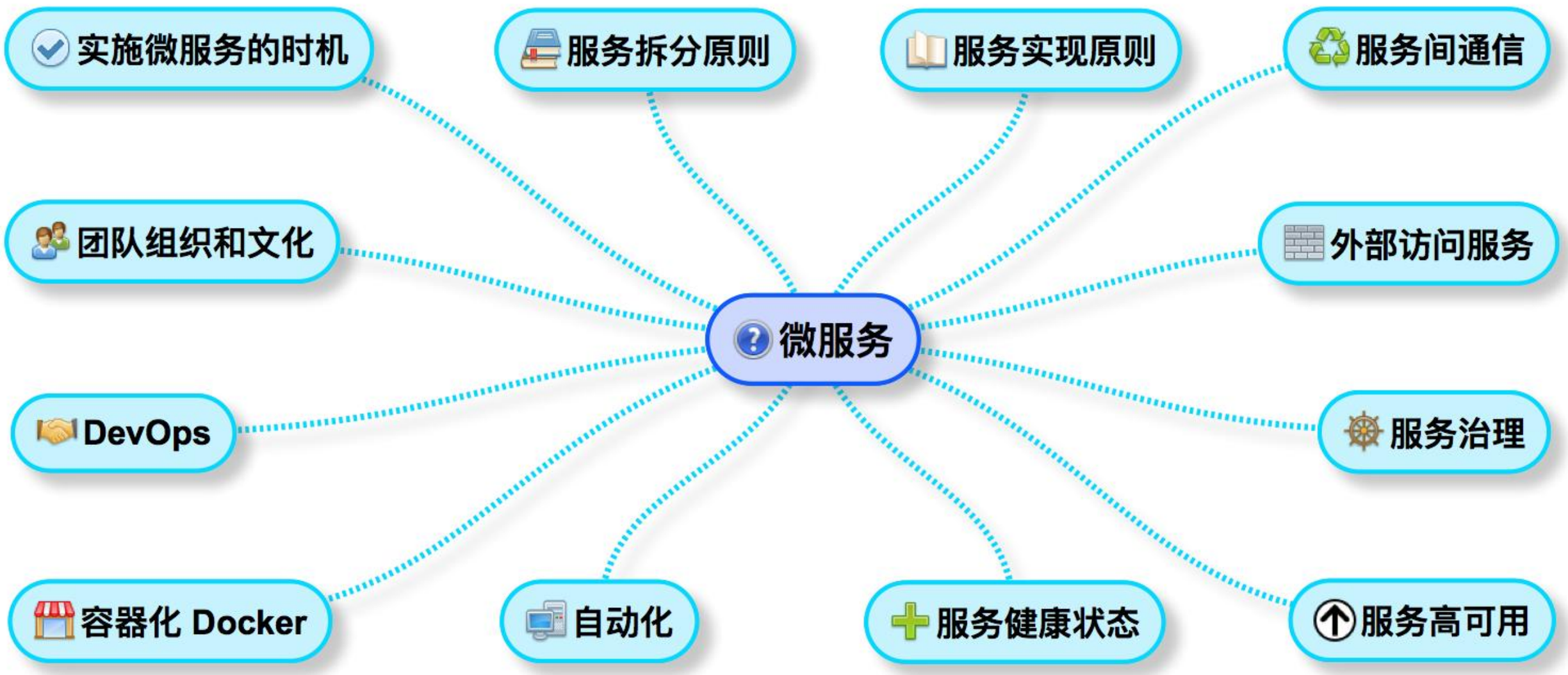
# 什么是微服务

- 分布式系统
- 面向服务架构
- 粒度
- DevOps

# 我们为什么需要微服务架构

- 系统太大难以维护？
- 系统可用性差？想借助微服务提高系统可用性？
- 为了更快的响应变化？提高需求交付的频率？
- 为了降低开发成本？
- 团队出于对新技术的向往？需要灵活的技术栈？

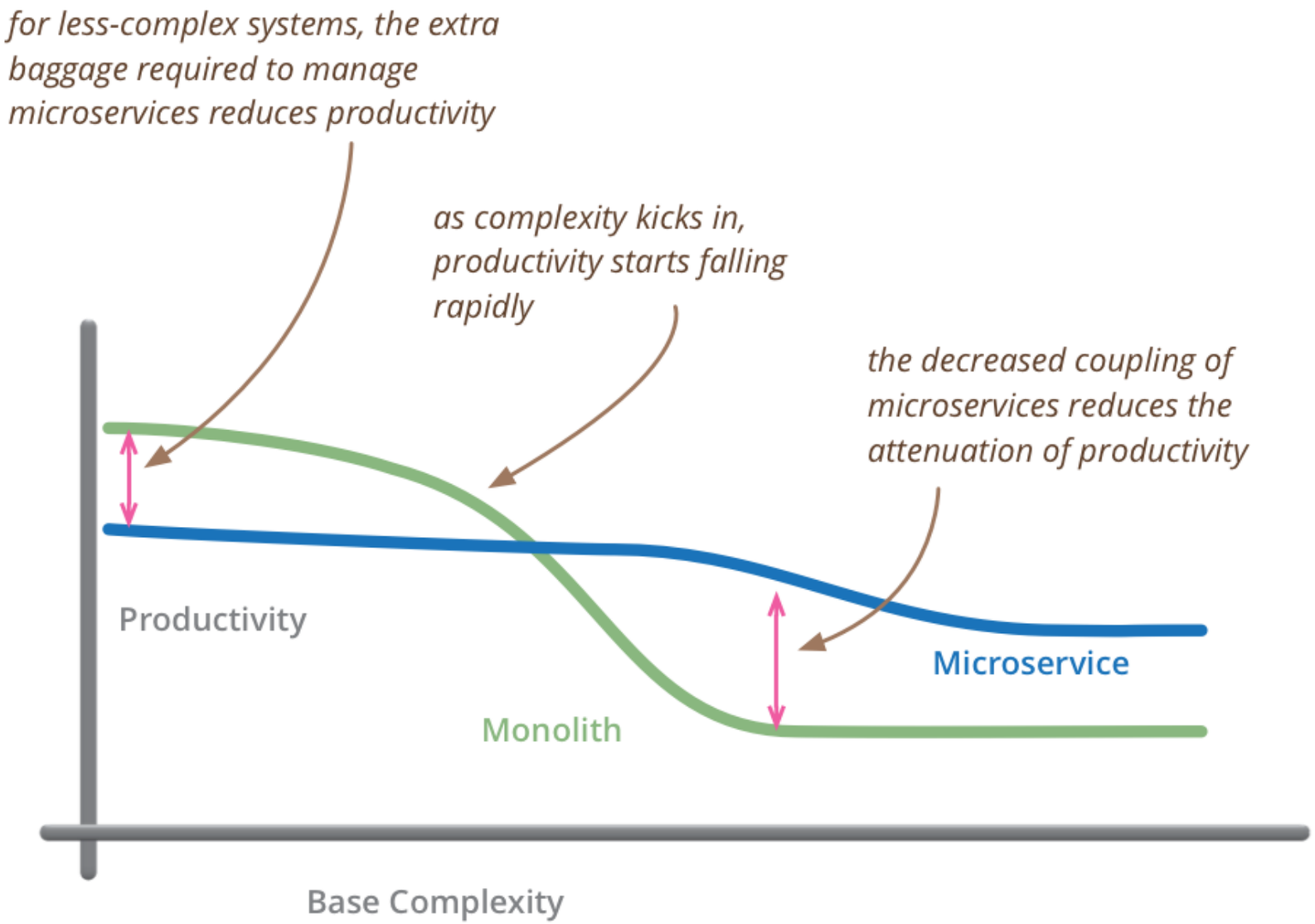
# 实施微服务需要思考的十二大类问题



# 实施微服务架构的时机

- 是不是每个系统都适合用微服务架构？
- 系统在什么阶段适合实施微服务？

# 实施微服务架构的时机




生产率和复杂度关系图

*but remember the skill of the team will outweigh any monolith/microservice choice*

**Martin Fowler**

# 实施微服务架构的时机

- 实施微服务架构的决定性因素应该是单个应用已过于庞大和复杂，以至于难以修改和部署。

- 软件架构和开发方法都可能会影响团队的组织形式，做好打造合适的开发团队和组织的心理准备。

# 服务的拆分原则

- 服务规划与设计（类比城市规划）
- 职责单一，功能完备
- 粒度适中，2 pizza 团队可开发和维护
- 服务分级（核心服务，非核心服务）

# 服务的实现原则

- 语言无关（服务支持多种编程语言调用）
- 能够独立部署和运行
- 考虑非功能职责的隐性要求（响应时间，TPS ，容量等）
- 演进的过程中能保持服务 API 的兼容性

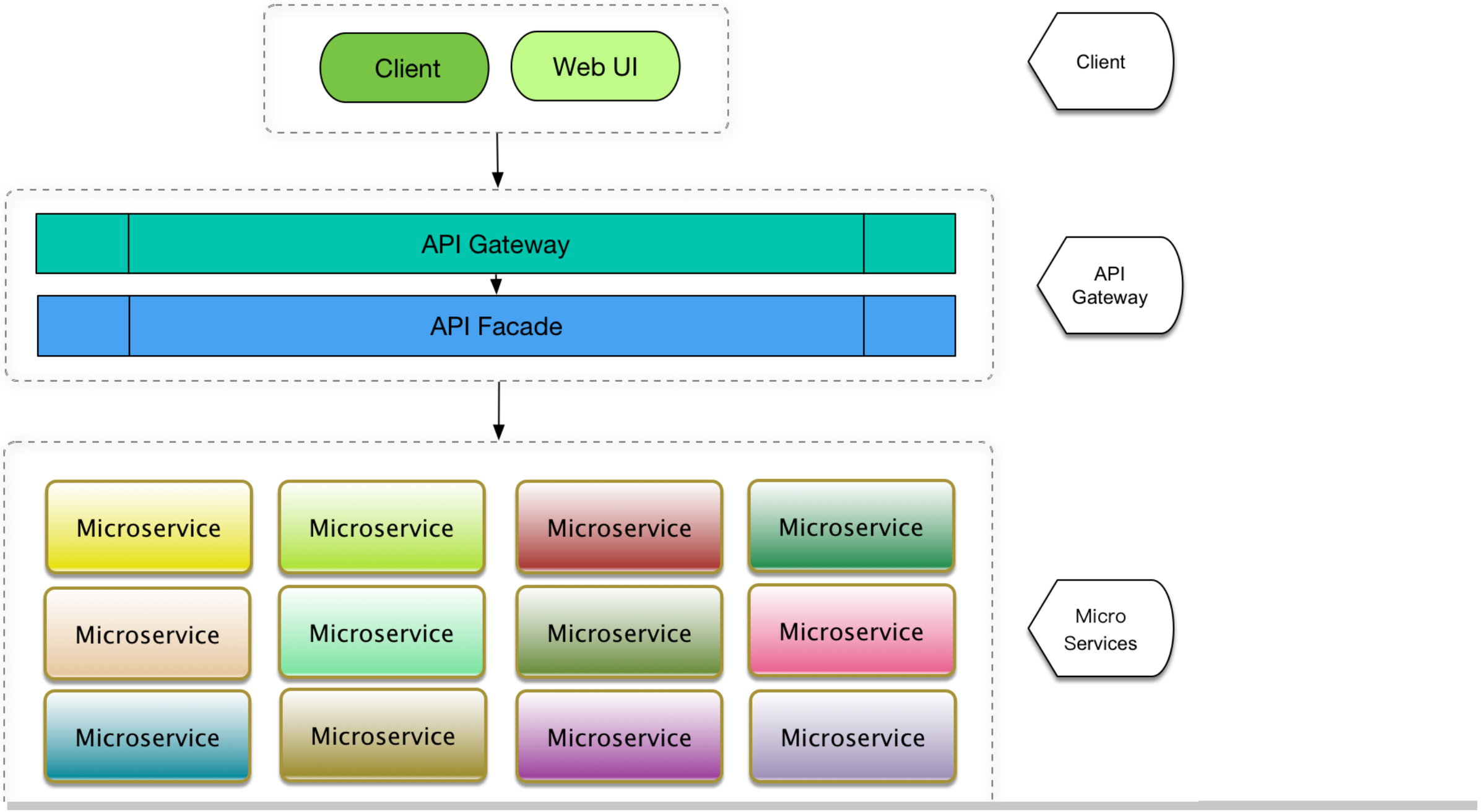
# 微服务之间的通信方式

- 同步调用（REST/RPC，gRPC，Thrift，Finagle，Dubbo）
- 异步调用（RPC，Akka）
- 异步消息（MQ，Kafka）
- 消息序列化方式（XML，Json，Protobuf，Thrift）
- 服务框架（DerbySoft-RPC，TCP，Protobuf，多语言实现）

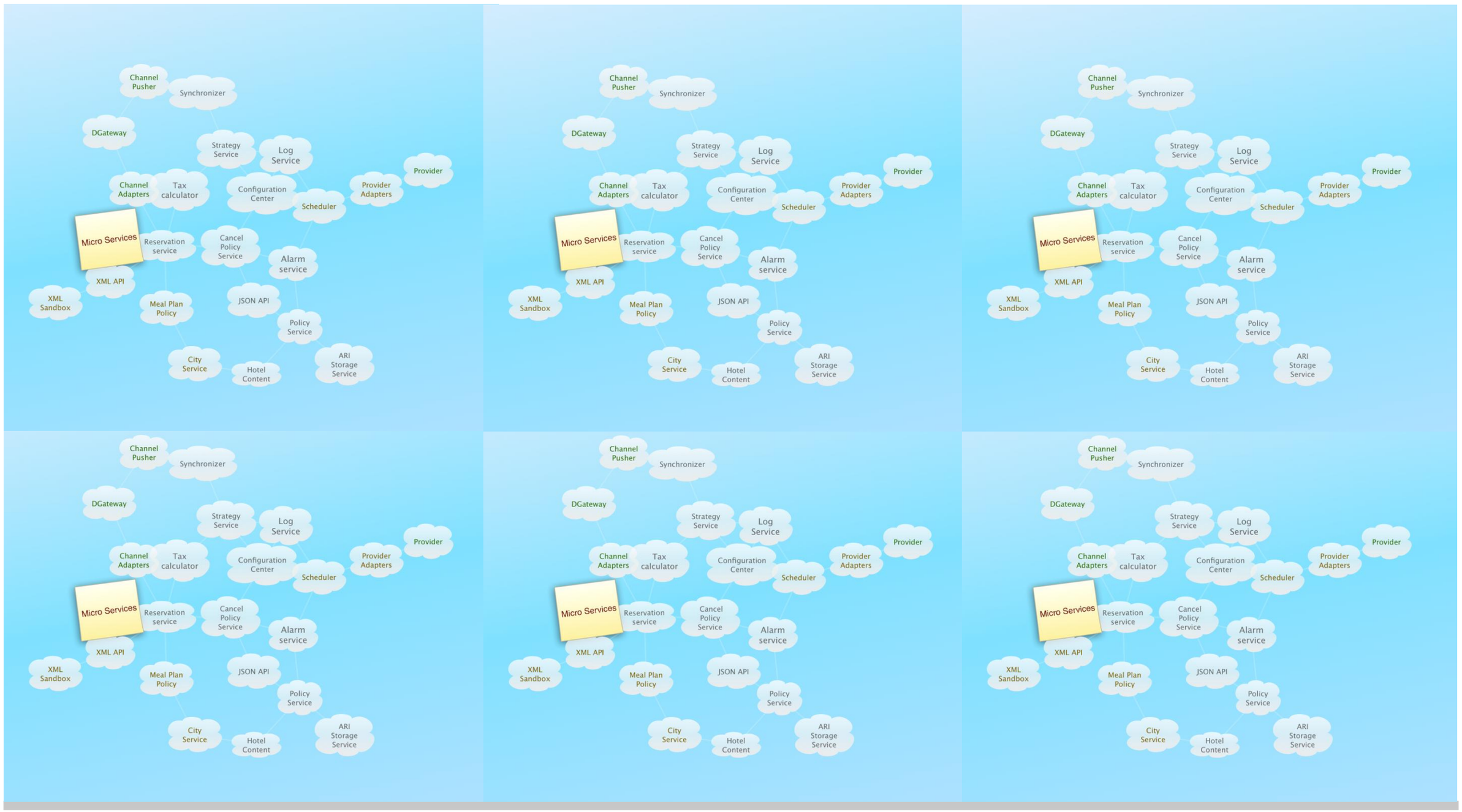
# 外部如何访问服务

- API Facade (整合后台的服务，提供更符合需求的 API)
- API Gateway (统一服务入口，提供安全认证和授权，流控等 API 管理功能)

# 外部如何访问服务


Client
Web UI
API Gateway
API Facade
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice
Microservice
Client
API Gateway
Micro Services

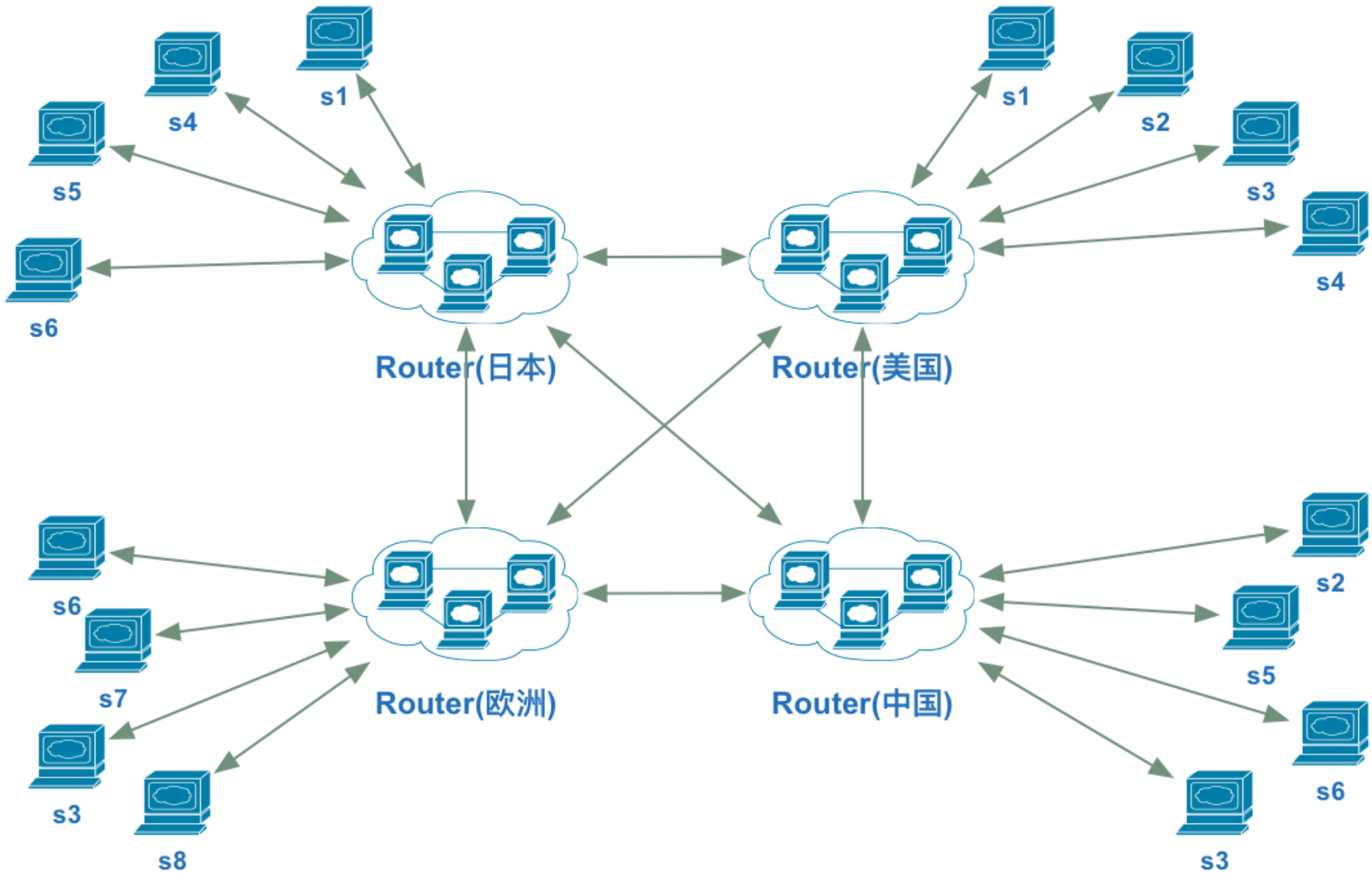
# 服务治理


Channel Pusher
Synchronizer
DGateway
Strategy Service
Log Service
Channel Adapters
Tax calculator
Configuration Center
Scheduler
Provider Adapters
Provider
Micro Services
Reservation service
Cancel Policy Service
Alarm service
XML API
XML Sandbox
Meal Plan Policy
JSON API
Policy Service
City Service
Hotel Content
ARI Storage Service

# 服务治理

- 服务注册，服务发现
- 服务依赖关系管理
- 服务路由
- 版本管理
- 服务隔离

# 服务治理-路由服务


s1
s4
s5
s6
Router(日本)
Router(美国)
s1
s2
s3
s4
s6
s7
s3
s8
Router(欧洲)
Router(中国)
s2
s5
s6
s3

# 服务的高可用策略

- 负载均衡
- 容错机制
- SLA 策略
- 超时控制

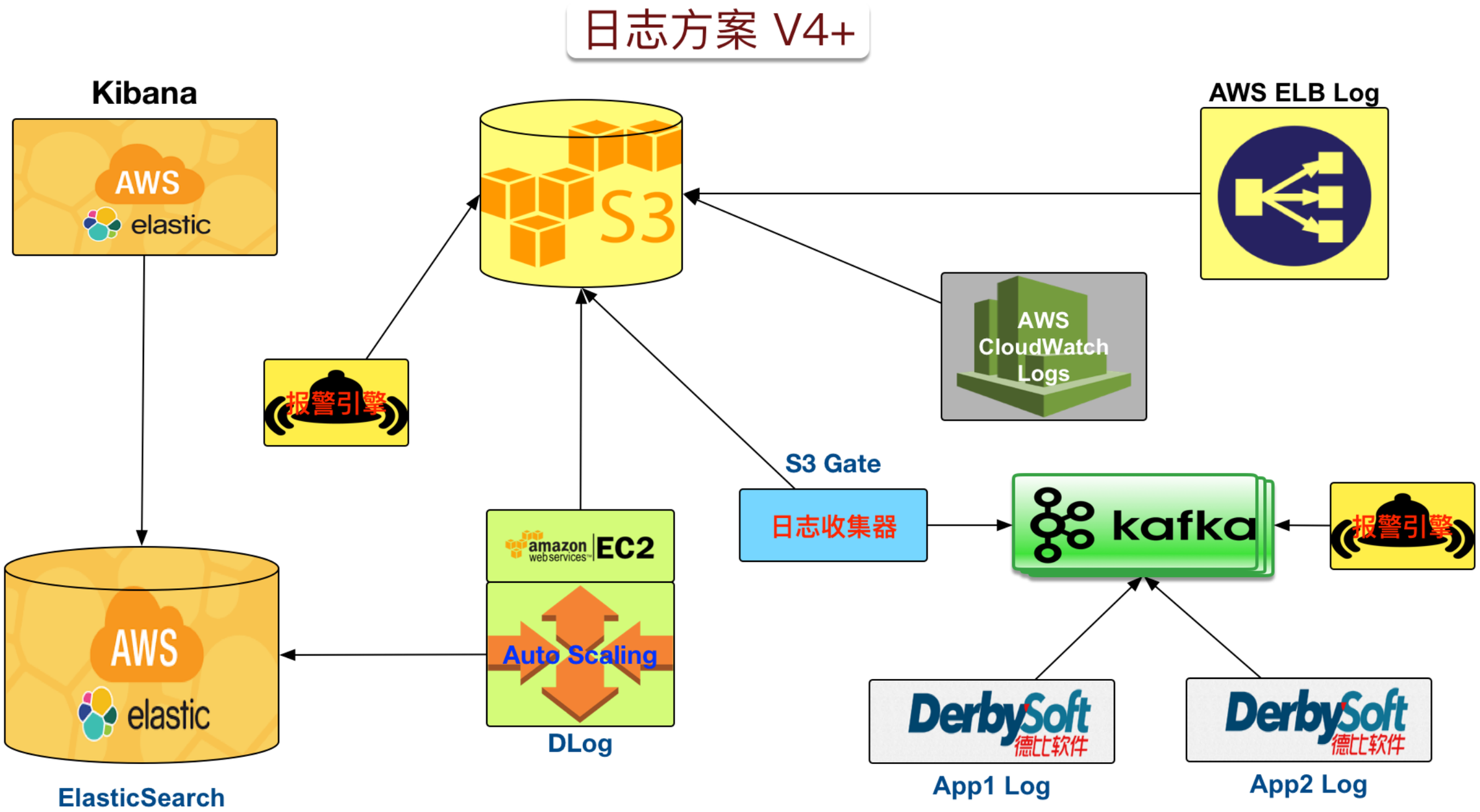
# 服务的高可用策略

- 服务多可用区冗余部署（N+2）
- 服务限流
- 熔断机制
- 服务降级

# 服务健康状态

- 监控报警（硬件&系统层监控、服务层监控、业务层监控）
- 日志（ELK）
- 调用链追踪（分布式Trace，Google Dapper，Zipkin，Jaeger）

# 服务健康状态-日志


日志方案 V4+
Kibana
AWS
elastic
S3
AWS ELB Log
AWS CloudWatch Logs
报警引擎
S3 Gate
日志收集器
kafka
报警引擎
amazon web services EC2
Auto Scaling
DLog
AWS
elastic
ElasticSearch
DerbySoft 德比软件
App1 Log
DerbySoft 德比软件
App2 Log

# 自动化

- 测试自动化（单元测试、功能测试、性能测试等）
- 发布自动化（配置管理、版本管理等）
- 运维自动化

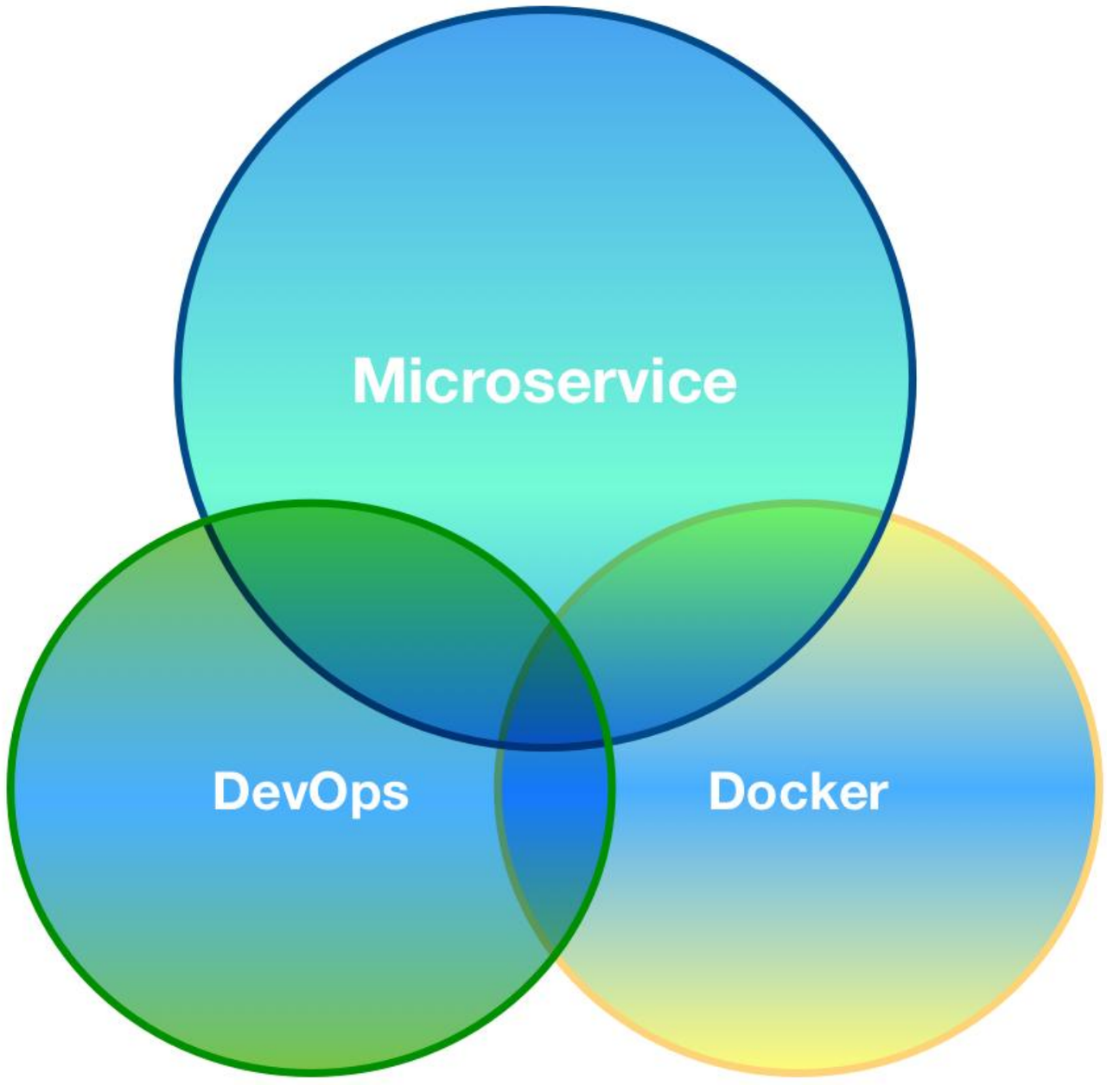
# 容器化

- Docker
- Kubernetes / Mesos（集群管理 & 调度和编排）

# DevOps

- Development & Operations
- 将运维作为需求整合到开发流程中去
- 开发自运营，运营自开发

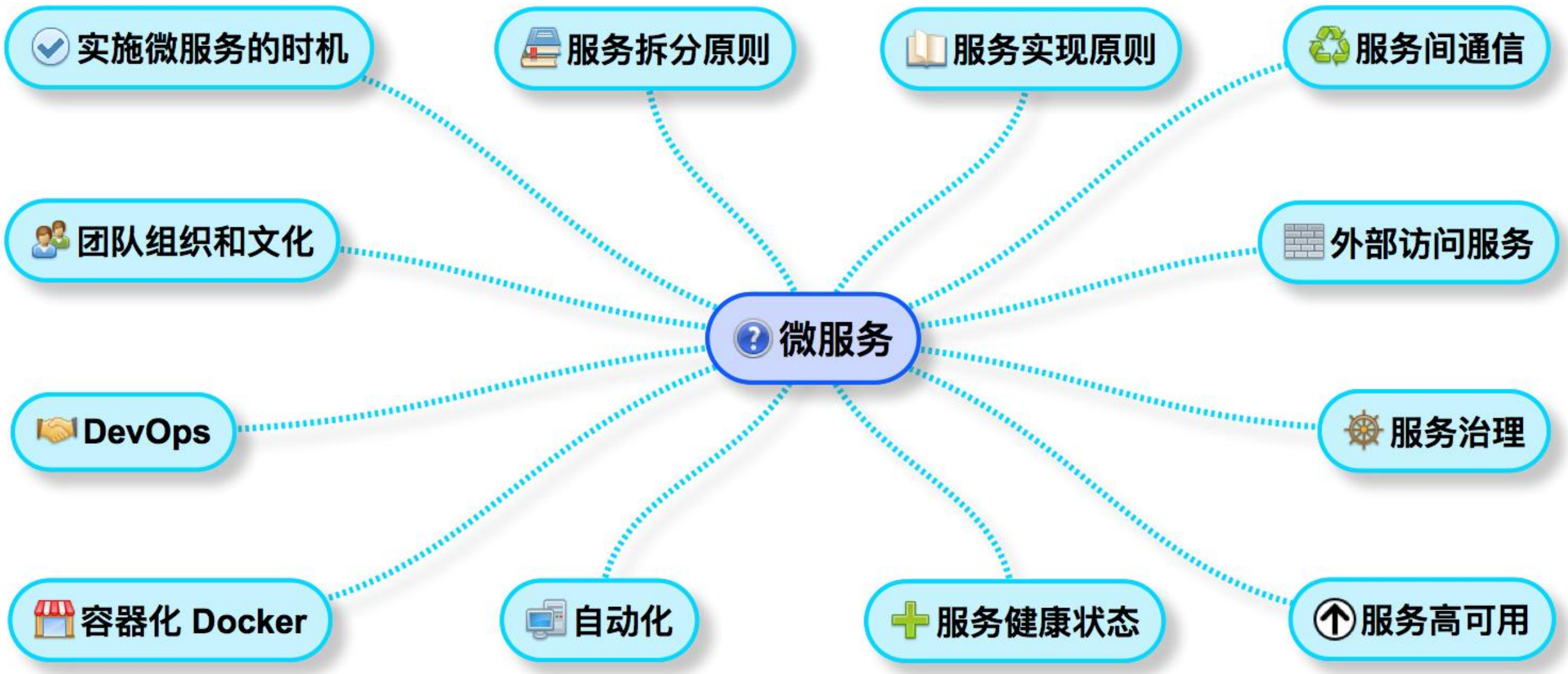
# 微服务 & Docker & DevOps



# 微服务架构对团队的影响

- 开发工程师角色的变化（个人能力要求更高，介入运维）
- 运维工程师角色的变化（介入开发）
- 更加紧密合作的全栈自治小团队，对团队提供的服务负责
- 技术团队结构组织和文化的影响

# 实施微服务要考虑的问题总结



# 实施微服务的收益

- 单个服务简单可控，系统易修改和更新
- 系统拥有足够的弹性与扩展能力，更有效的利用硬件资源
- 实现敏捷开发和部署
- 灵活的技术栈，高效的技术团队

# 实施微服务的挑战

- 数据一致性
- 服务治理
- 测试
- 部署和运维
- 监控
- 故障追踪