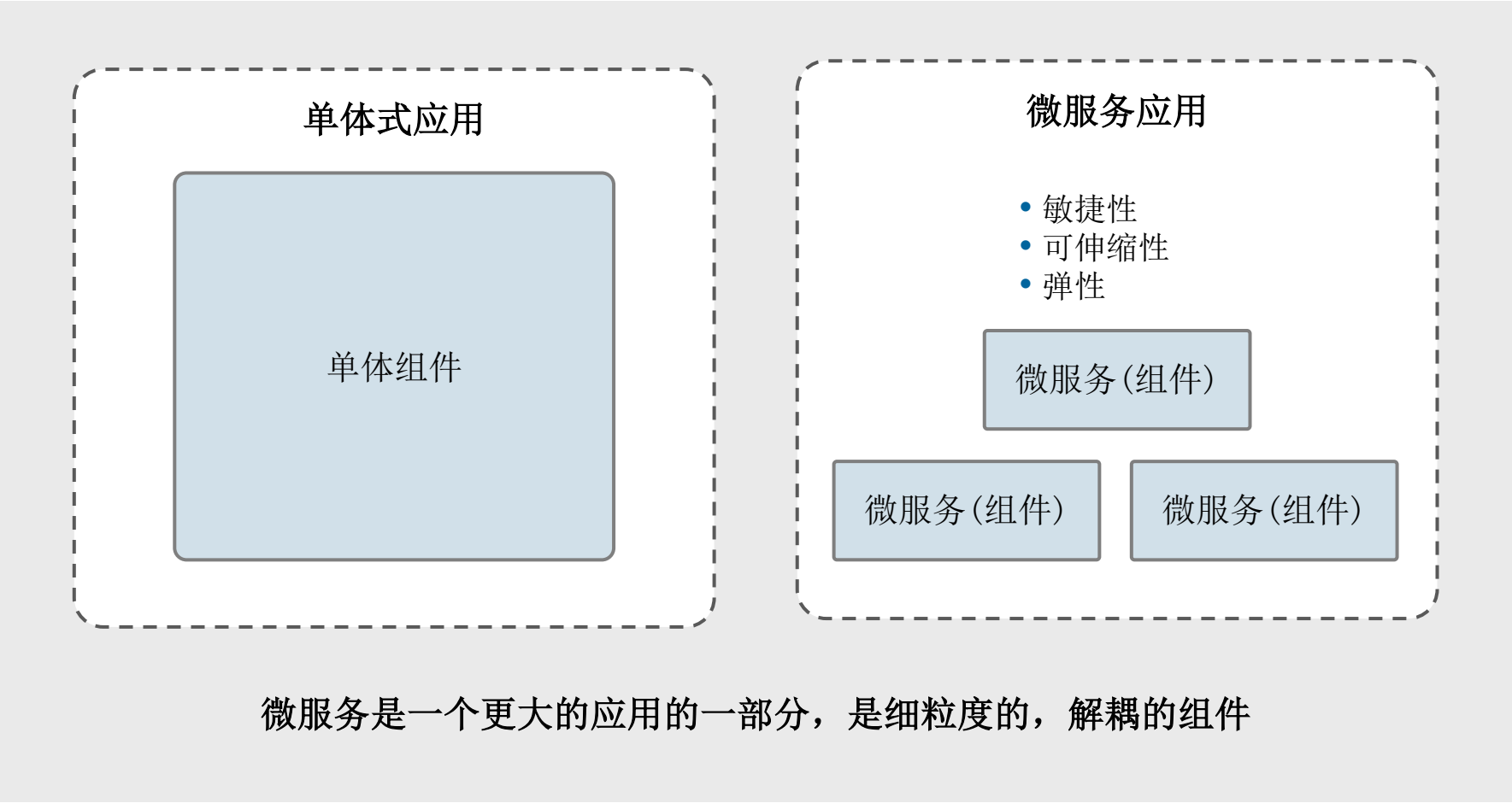
微服务架构
从概念到实践

目录

微服务架构

微服务架构是将大的单体应用分解为更易于管理的、全面解耦的模块



微服务是一个更大的应用的一部分，是细粒度的，解耦的组件

微服务架构原则

- 将大的单体应用分解为多个小的服务
  - 一个服务一个容器 one service per container
- 为单一功能优化的服务
  - 单一责任原则 single responsibility principle
- 通过REST API或消息中间件通信
- 为每个服务应用CI/CD (持续集成/持续交付)
- 为每个服务应用高可用性 (HA, high availability)和集群策略

推动微服务开发的因素

- 开发团队如何组织
  - 微服务开发由一个小团队负责具有单一功能的一个模块的完整生命周期
  - 通过最小化团队间沟通协作、消除变更的范围和风险加速交付

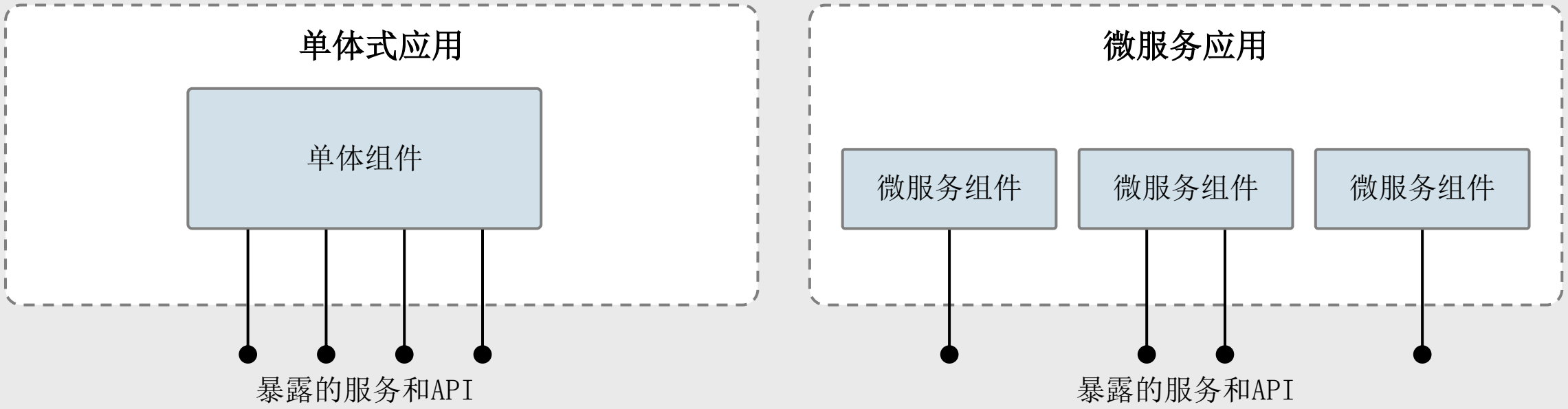
- 应用如何构建
  - 使用标准化的云环境，包括好处和坏处

- 应用如何交付运行
  - 每个服务一个容器，标准化的打包封装

微服务与SOA/API

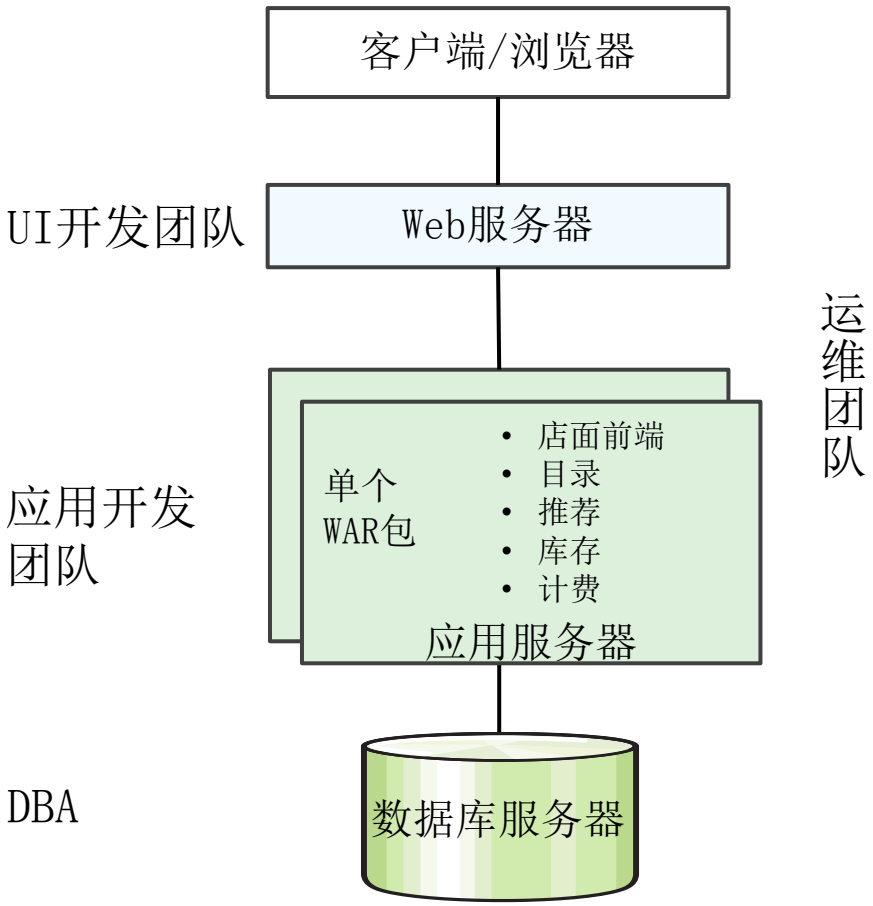
~~微服务是更细粒度的web服务~~
~~API就是微服务~~

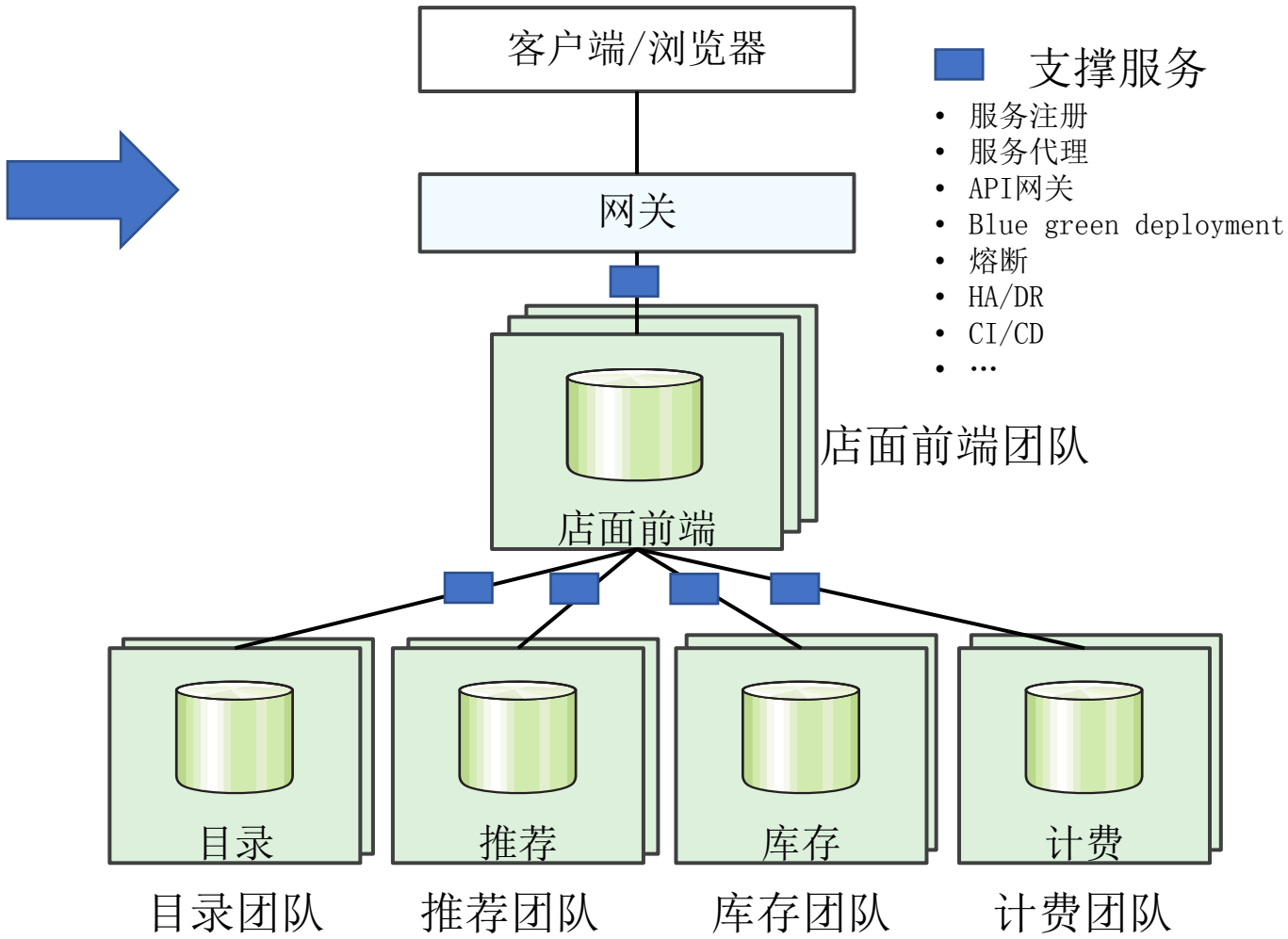
“微”指的是组件的粒度，
而不是暴露出来的接口的粒度



“微服务架构”其实是
“微组件架构”？

单体应用 vs. 微服务应用





云原生应用

- 使用标准云环境，利用它带来的好处，也需要它带来的新的挑战
  - 弹性伸缩
  - 不可变更的部署
  - 可抛弃的实例
  - 不可预测的基础设施

- 与底层操作系统保持明确的协议确保可迁移性
- 可以弹性伸缩而不需要在工具、架构、开发实践等方面做显著变更
- 对来自基础设施层和应用层不可避免的故障保持弹性
- 全面洞察技术和业务问题
- 使用云服务，例如存储、消息队列、缓存等
- 快速、可重复地部署，保持敏捷性
- 自动化准备工作，减少新开发人员加入所需的时间和成本

微服务架构的好处

- 一句话：独立！
- 两句话：使开发更高效！

- 独立的团队
  - 小团队，所有团队成员都可以清楚的理解整个项目
- 独立的开发
  - 有限的、明确的依赖关系
- 独立的时间表
  - 独立发布新版本
- 最合适的语言
  - 每个服务可以选择不同的语言
- 独立拥有数据
  - 独立选择最合适的技术，最合适的schema
- 独立伸缩
  - 隔离问题

微服务架构兴起的技术原因

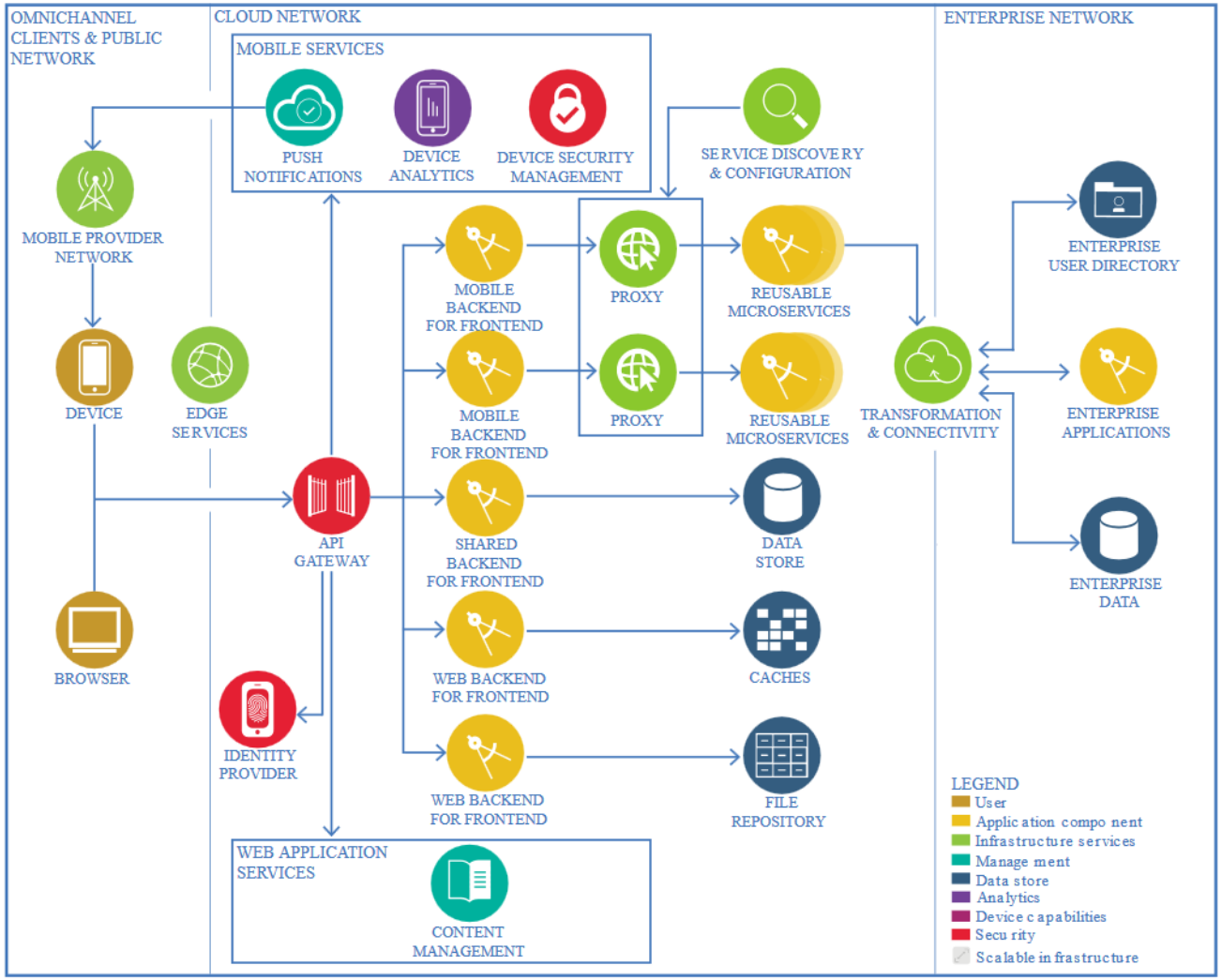
- 分布式计算变得容易、可行
  - 网络变得成熟、稳定、普遍
  - REST API和轻量级消息引擎变得简单
- 应用托管变得容易、简单
  - 轻量级运行时出现，例如node.js, liberty
  - 基础设施简化，包括操作系统虚拟化、容器化、IaaS服务，工作负载虚拟化
  - PaaS的兴起
- 敏捷开发方法与工具的成熟
  - IBM Bluemix Garage方法，极限编程，Scrum, 测试驱动开发，CI/CD
  - 标准化DevOps工具与实践的成熟

微服务架构的挑战

- 增加运维复杂性
- 要求开发人员具备熟练的DevOps技能
- 分布式系统带来的额外复杂性
  - 网络延迟
  - 容错
  - 串行化
- 解耦的非交易型系统设计
- 如何定位实例？
- 维护分区数据的可用性和一致性
- 端到端的测试

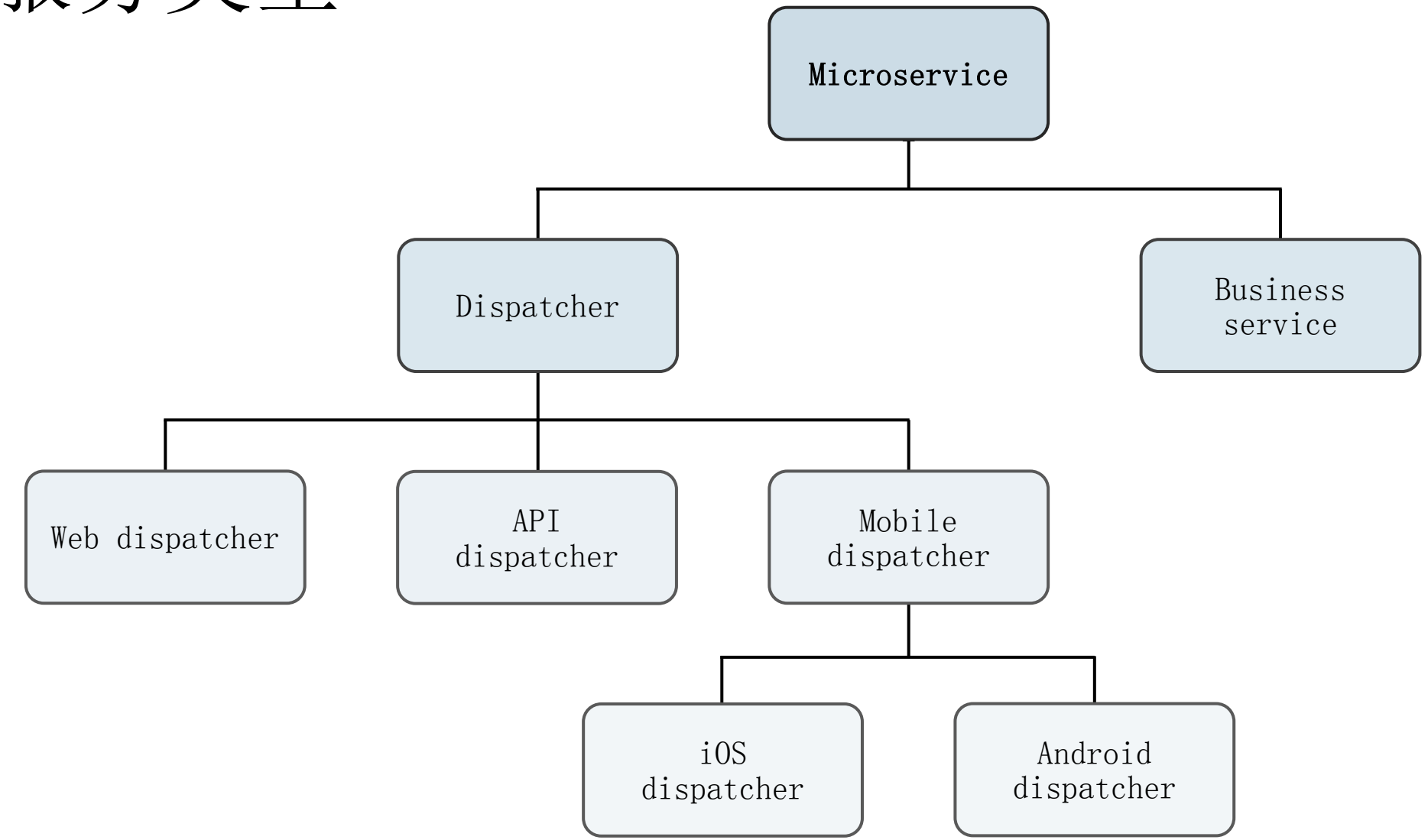
目录

微服务应用参考架构



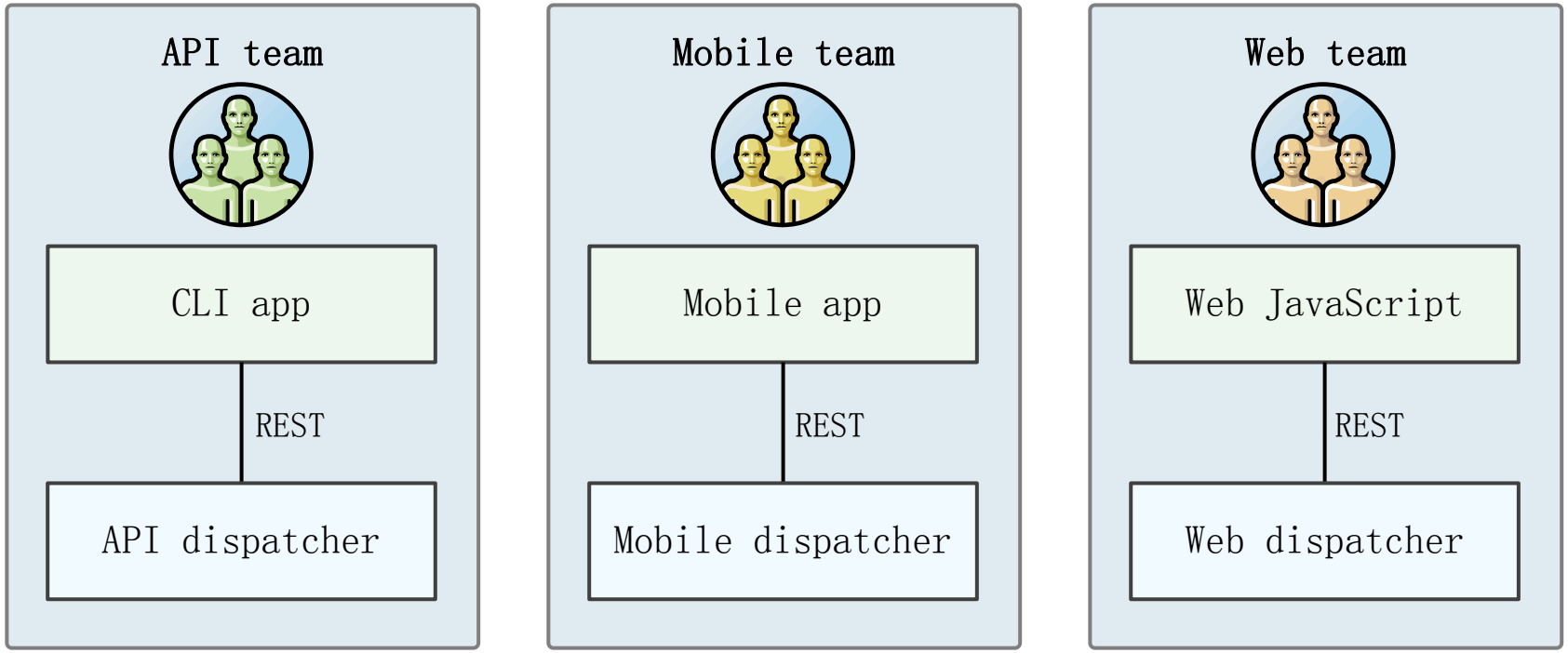
https://www.ibm.com/devops/method/content/architecture/omnichannelArchitect

微服务类型


Microservice
Dispatcher
Business service
Web dispatcher
API dispatcher
Mobile dispatcher
iOS dispatcher
Android dispatcher

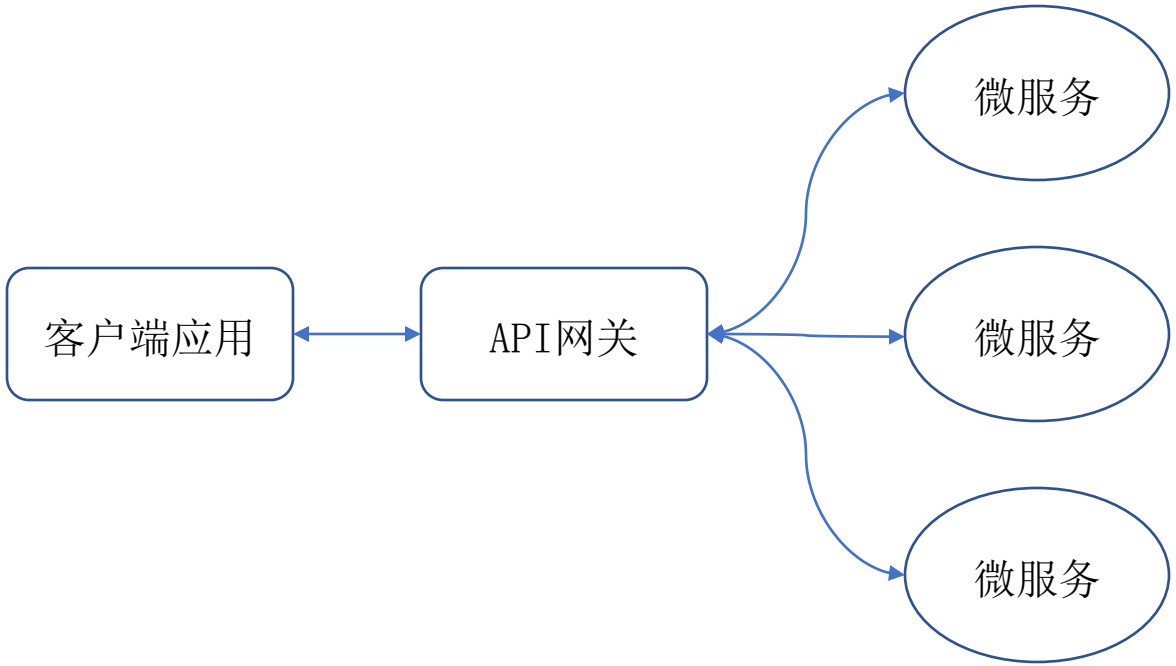
BFF (Back ends for front ends)模式

- 每一个分配器作为一个外部前端的后端
- 由相同的团队开发前端和后端
- 采用类似或兼容的语言



“Pattern: Backends For Frontends” by Sam Newman
http://samnewman.io/patterns/architectural/bff/

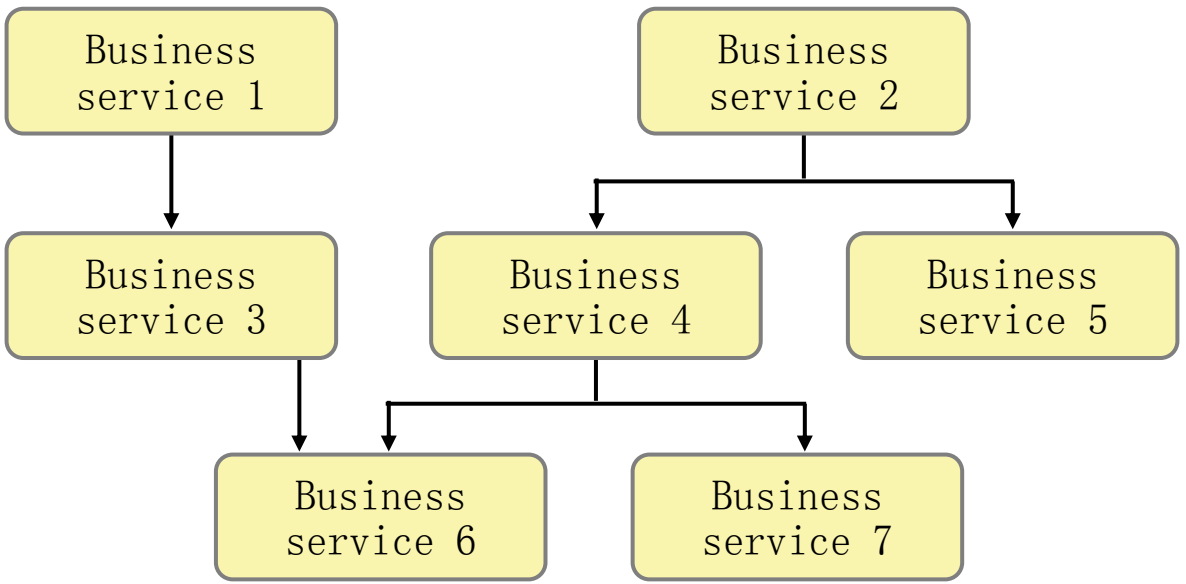
API网关模式



API网关对客户端应用与内部微服务之间的通信进行抽象，使得微服务可以聚合满足客户端需要

业务服务

- 业务服务可以调用其他业务服务，但需要避免循环依赖
- 保持每个服务完成独立完整的任务，而不是分层

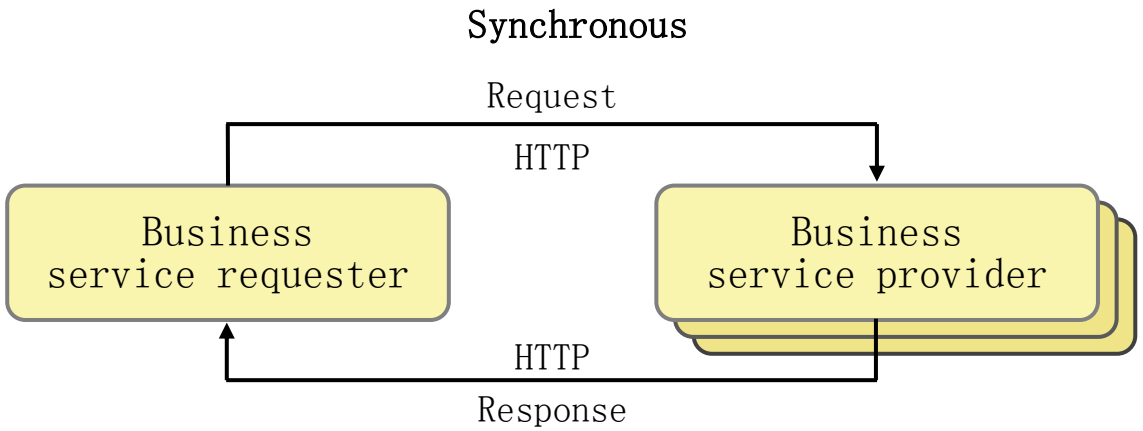


微服务编程语言选择

- Dispatcher编程语言选择
  - 与客户端一致性
    - 例如JavaScript前端往往可能选择node.js作为后端运行时语言
    - iOS前端也可以选择Swift作为后端运行时语言
  - I/O密集型，能够支持大规模并发客户端访问
- 业务服务编程语言选择
  - CPU计算密集型
  - 方便与外部系统集成对接
  - Java在很多时候依然是一个好的选择

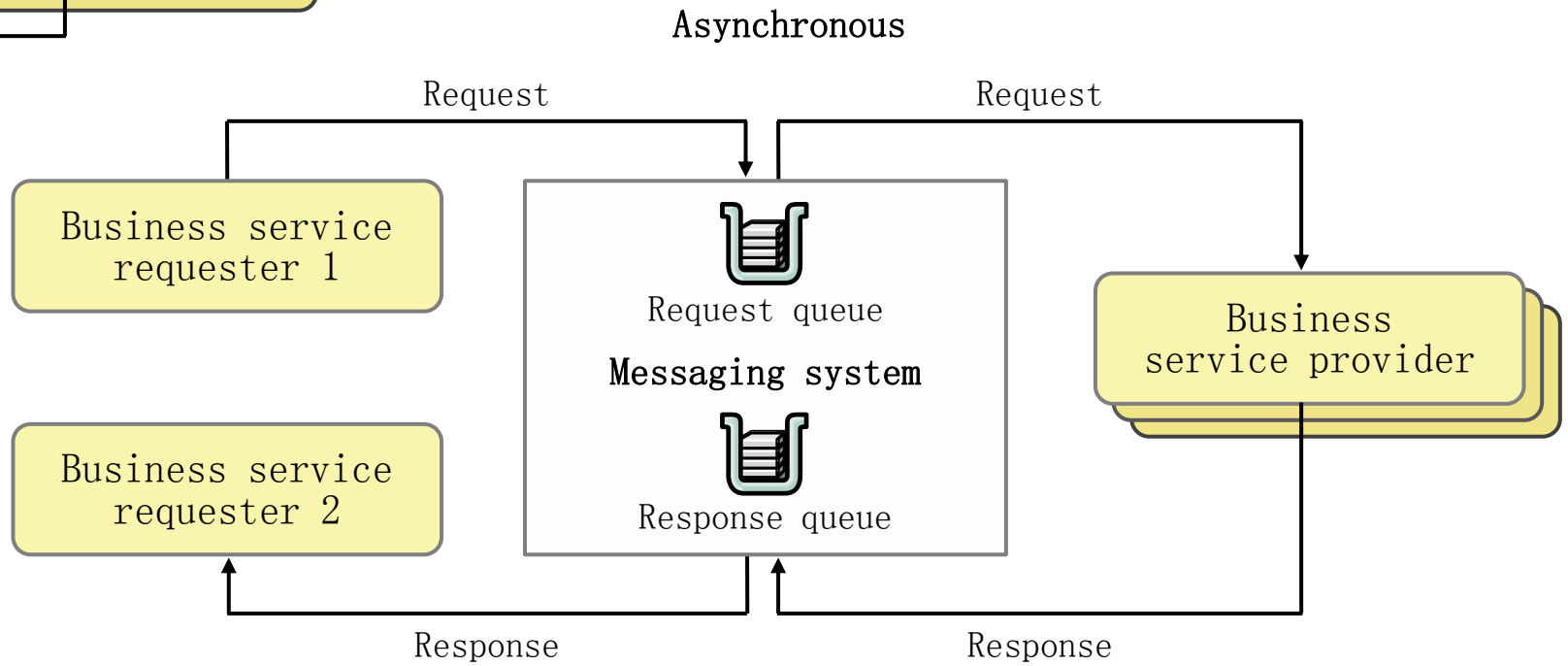
- 当然，最终是开发团队选择开发语言☺
  - 选择最合适完成工作的语言
  - 选择开发团队熟练掌握，或者具备技能的开发人员比较容易获得的语言

微服务集成——同步 vs. 异步



异步通信使微服务更强壮

- 请求者不需要在提供者运行计算时阻塞
- 请求响应可以由不同的请求者实例处理
- 消息系统可以处理执行和结果



服务之间的通信

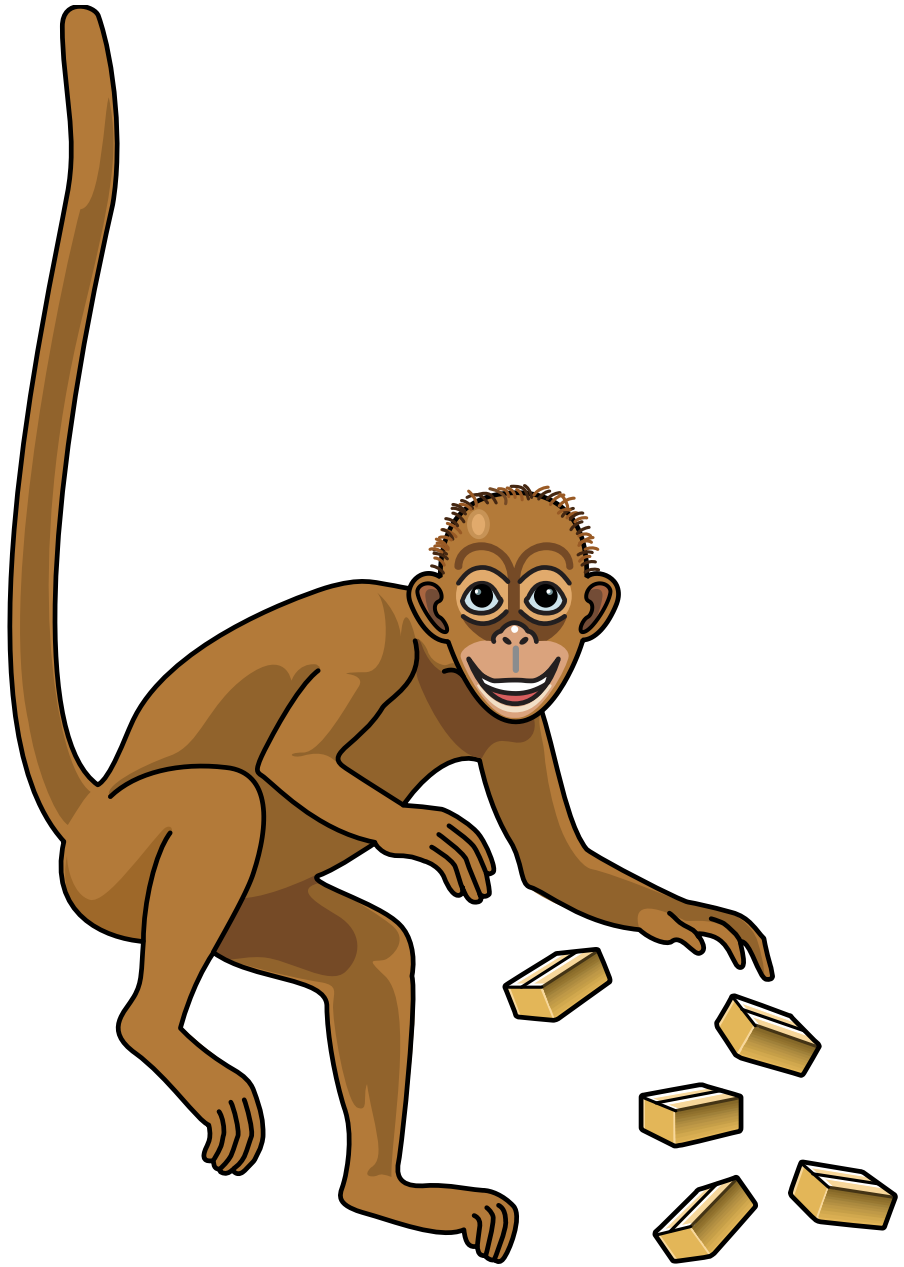
- 服务之间通信方式的选择应该是语言无关的
  - 服务可以使用不同的编程语言实现

- 最常用的同步协议
  - HTTP REST
  - JSON
- 异步协议
  - 消息系统，例如AMQP, Kafka…Message Hub, MQ Light, RabbitMQ…
  - JSON消息负载

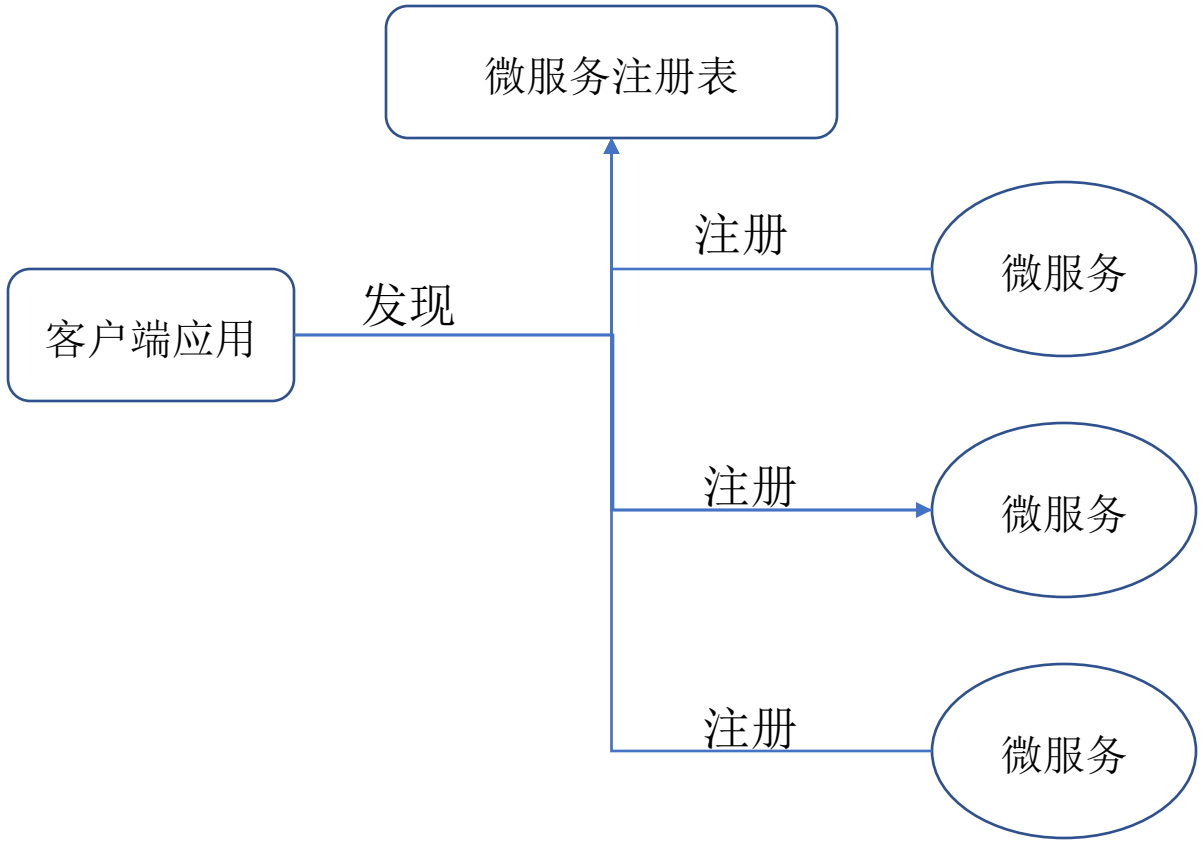
- 目标是完全解耦
  - 尽可能使用消息机制
  - 使用服务注册和发现服务
  - 使用负载均衡
  - 熔断模式
- 避免同时使用多种调用风格

Design for failure

- 任何服务都可能出故障
- 应用必须保持工作，保持响应
- 保持应用弹性的设计模式
  - 服务注册表 Service Registry
  - 熔断 Circuit Breaker
  - 隔离舱 Bulkhead
  - 命令 Command
- 故障测试
  - 有目的的，随机地制造故障
  - 猴子军团框架 Simian Army framework



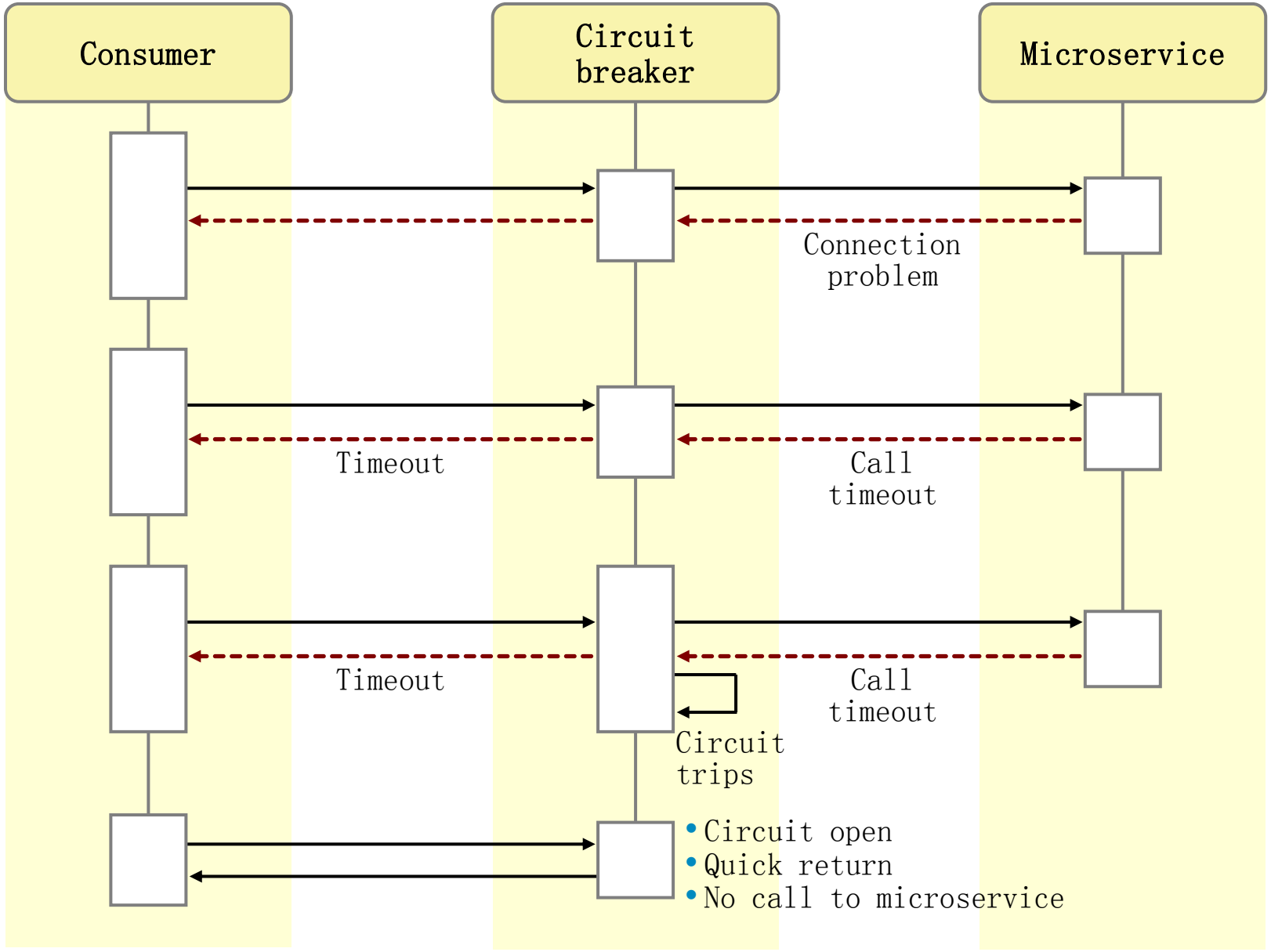
微服务注册与发现



- 客户端如何发现服务提供者
  - 每个服务提供者都是一个微服务实例
  - 客户端可以使用相同微服务的任何一个实例
  - 微服务实例池和他们的位置会发生变化，例如出于以下需要：
    - 弹性伸缩
    - 健康管理
    - 负载分布
- 微服务注册表
  - 服务实例启动以后就会出现在微服务注册表清单中
  - 注册表知道每个服务实例的位置
  - 注册表可以在服务实例之间分散负载
- 示例
  - Eureka
  - Consul
  - Cloud Foundry router

熔断 Circuit Breaker

- 调用不可靠的服务实例
  - 服务可能发生错误
  - 服务可能超时
  - 网络可能发生故障
- Circuit Breaker
  - 消费者可以可靠地调用服务
  - 被调用的服务是不可靠的
  - 在被调用服务发生故障时快速返回



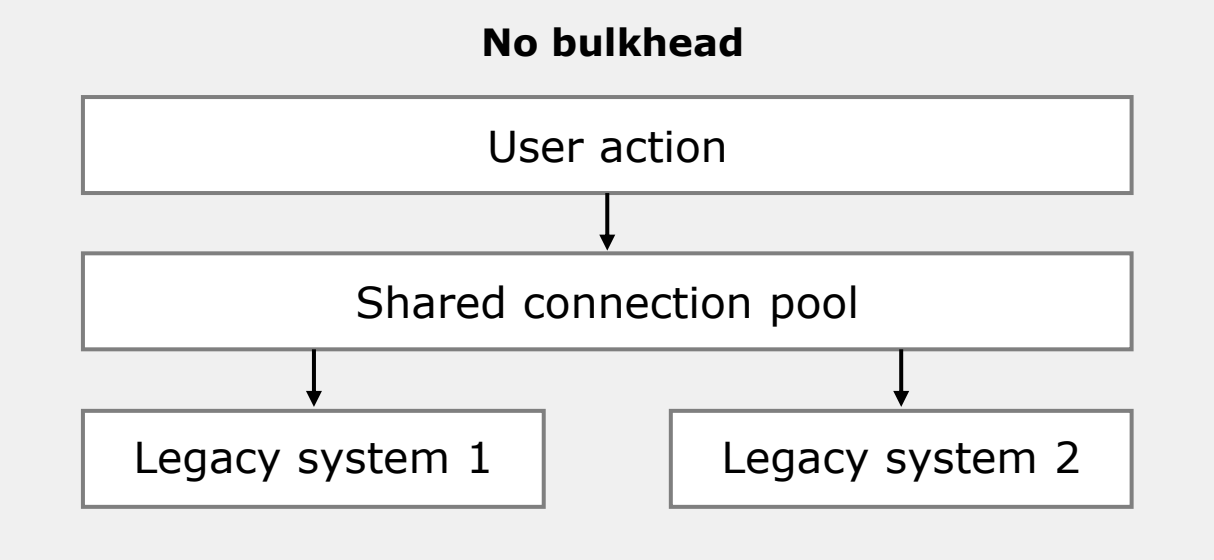
隔离舱 Bulkhead

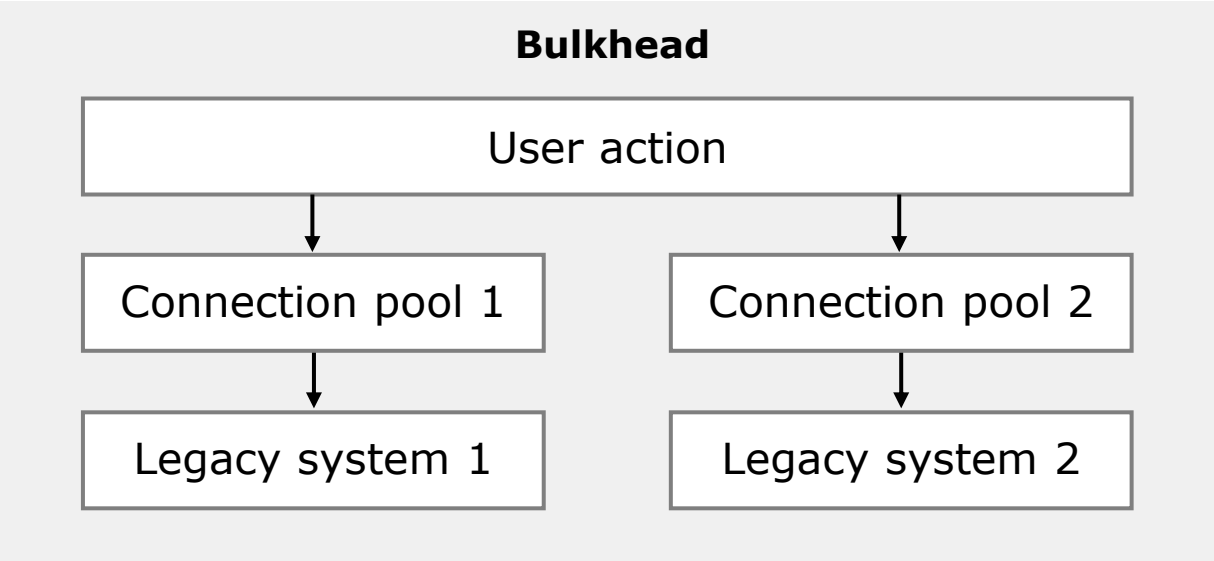
没有隔离舱：

- 单个连接池共享用于连接多个外部系统
- 如果其中一个外部系统阻塞了连接
  - 所有连接都会在这个系统上阻塞
  - 无法连接其他正常工作的外部系统

有隔离舱：

- 为每个外部系统设置独立的连接池
- 如果其中一个外部系统阻塞了连接
  - 所有连接这个系统的连接被阻塞
  - 连接其他外部系统的连接可以正常工作





命令和熔断

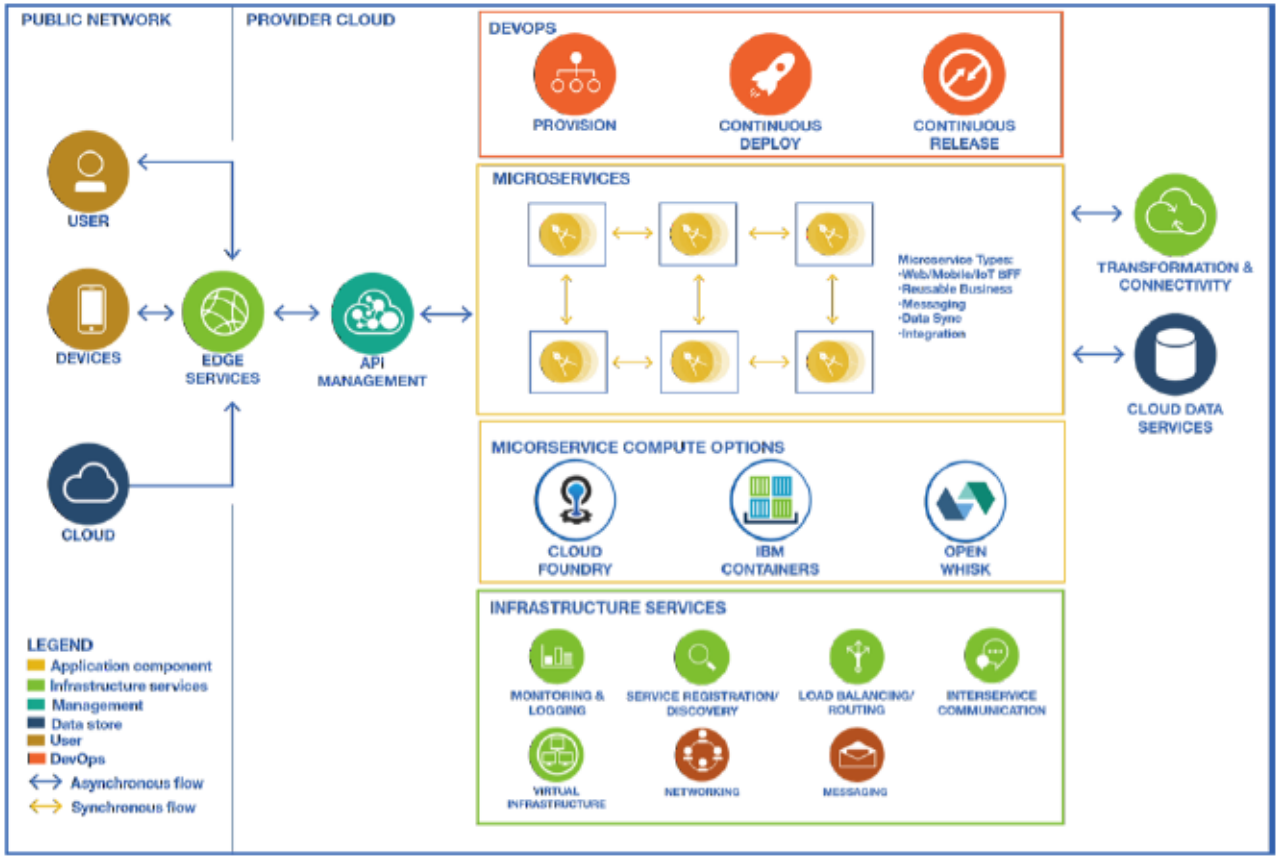
- 熔断模式通常和命令对象一起使用
- 当熔断失败时
  - 如何重试连接不同的服务实例？
  - 不应该要求每个消费者自己实现重试的逻辑

- 命令对象
  - 将服务请求封装为一个命令对象
  - 失败的命令会重试
  - 命令可以放在队列中实现节流
  - 可以收集度量指标用于监控

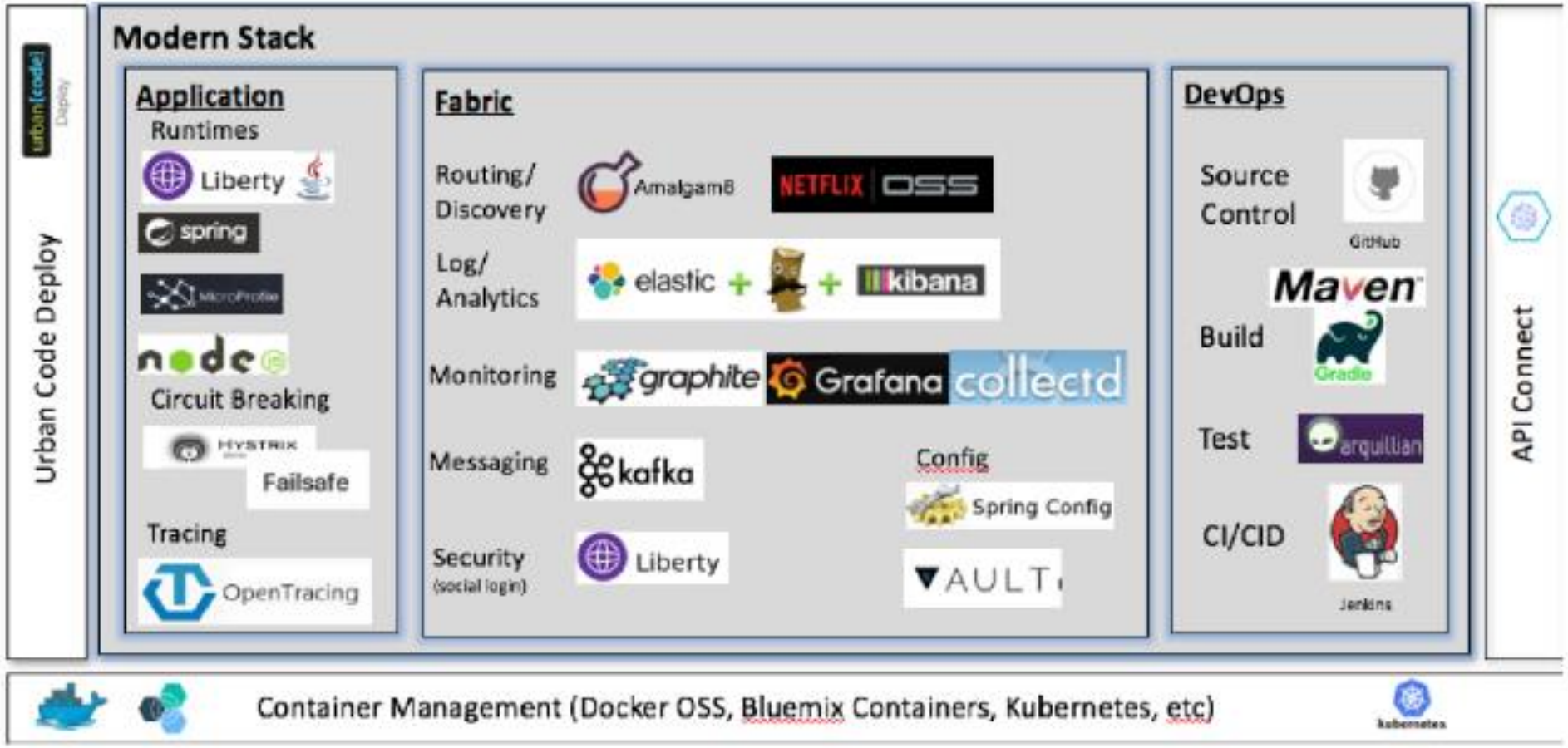
- 示例：Netflix Hytrix
  - 微服务调用容错框架
  - 提供实现熔断、隔离舱、命令对象等模式的代码
  - 提供仪表板监控服务可用性和性能

目录

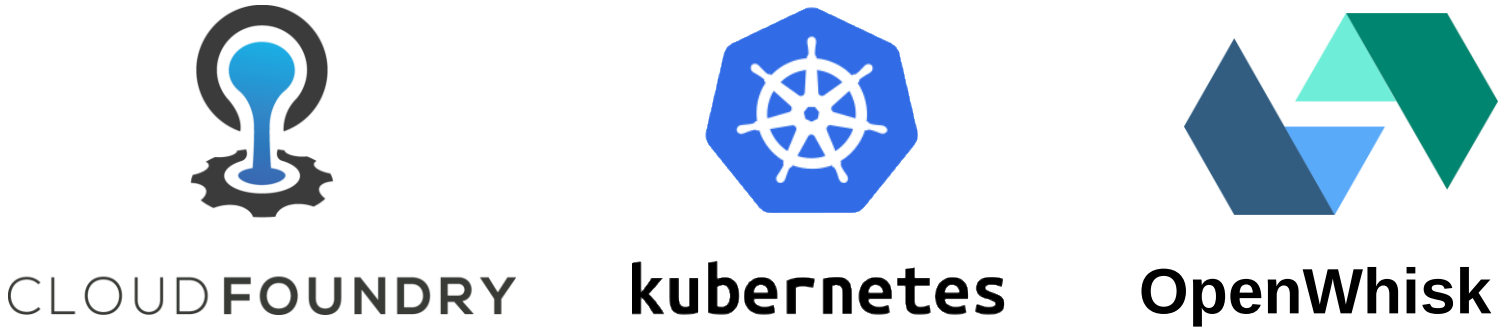
采用微服务架构应具备的能力



微服务架构技术堆栈

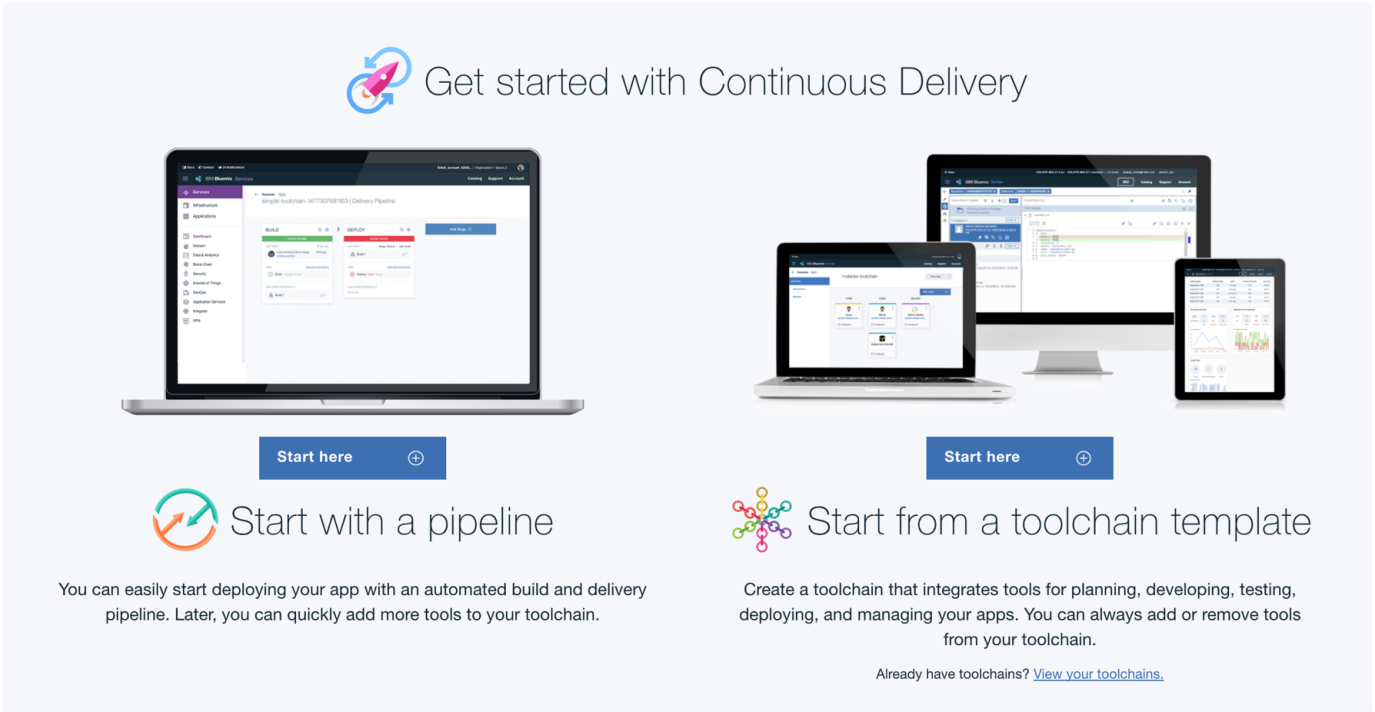

Urban Code Deploy
Modern Stack
Application
Runtimes
Liberty
spring
MicroProfile
node
Circuit Breaking
HYSTRIX
Failsafe
Tracing
OpenTracing
Fabric
Routing/
Discovery
Amalgam8
NETFLIX OSS
Log/
Analytics
elastic
kibana
Monitoring
graphite
Grafana
collectd
Messaging
kafka
Config
Spring Config
Security
(social login)
Liberty
VAULT
DevOps
Source
Control
GitHub
Maven
Build
Gradle
Test
arquillian
CI/CID
Jenkins
API Connect
Container Management (Docker OSS, Bluemix Containers, Kubernetes, etc)
kubernetes

在IBM Bluemix上实现微服务架构



计算选项

IBM Bluemix DevOps持续交付服务



集成的DevOps工具链

通过自动化的交付管道持续交付

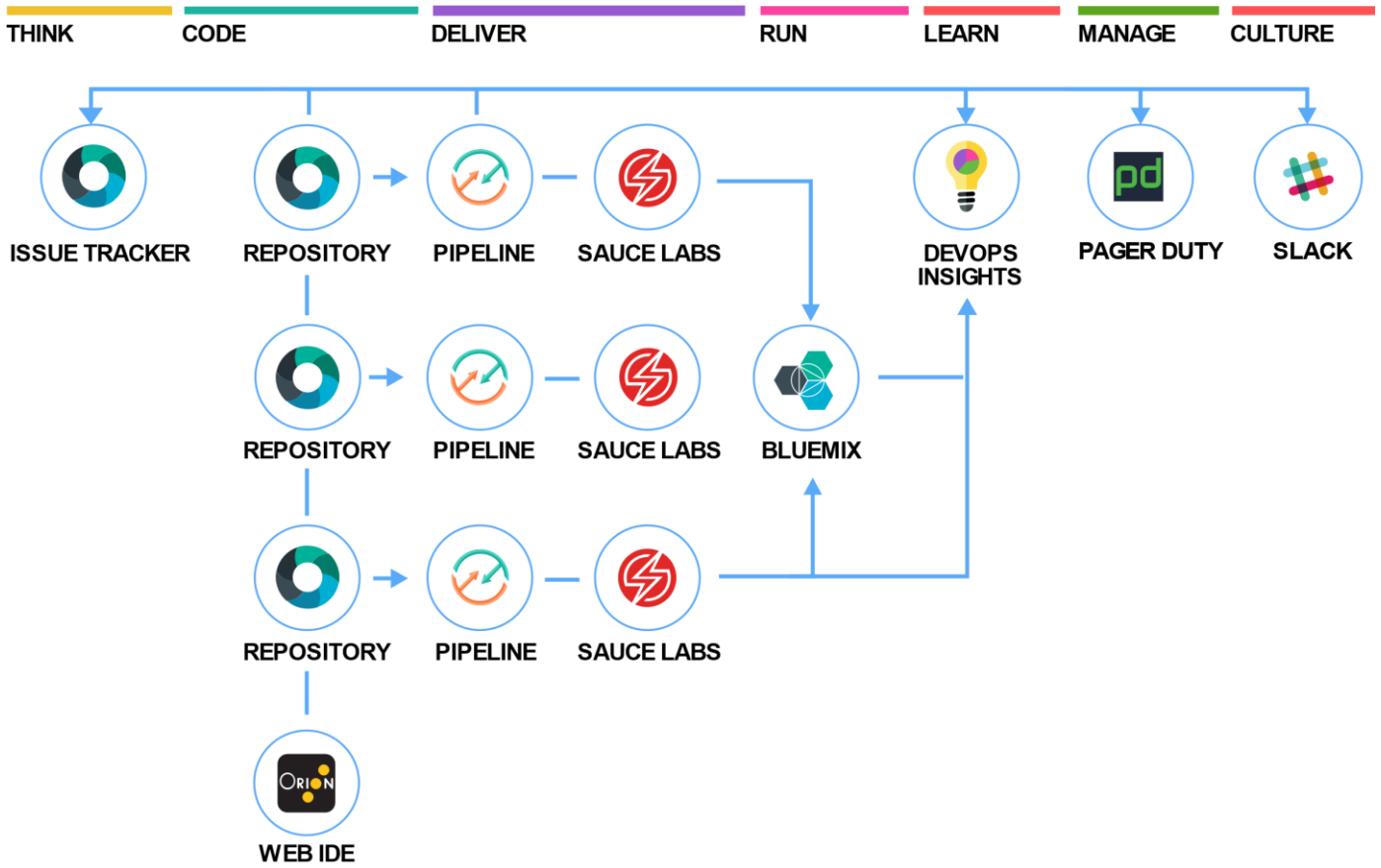
DevOps Insight持续改进质量

WebIDE可以随时随地编辑代码

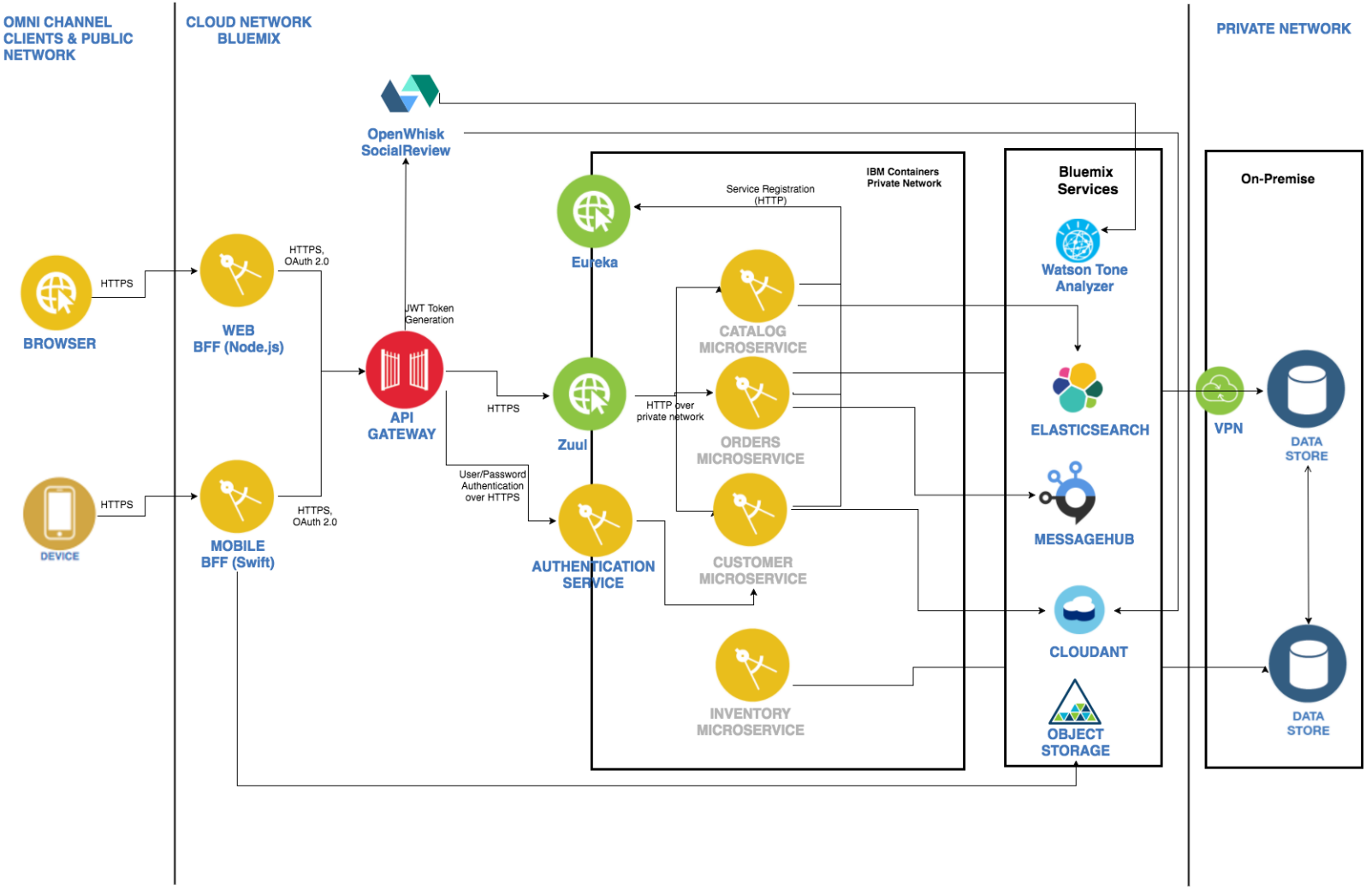
Git Repos and Issue Tracking

To learn more visit:
https://bluemix.net/devops/

IBM Bluemix DevOps微服务工具链

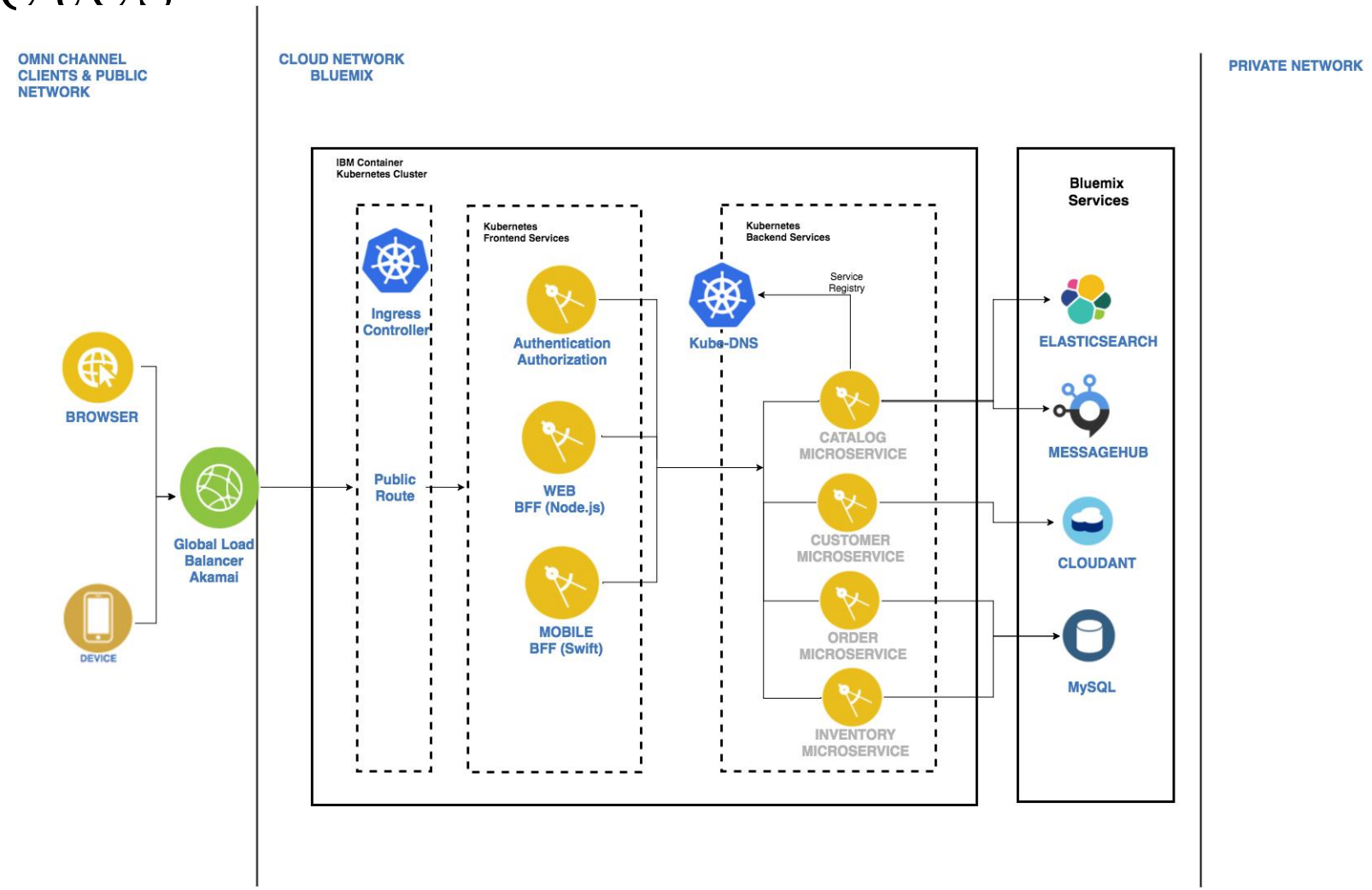


IBM Bluemix微服务应用架构示例



https://github.com/ibm-cloud-architecture/refarch-cloudnative

IBM Bluemix微服务应用架构示例 - Kubenetes



Message Hub

1

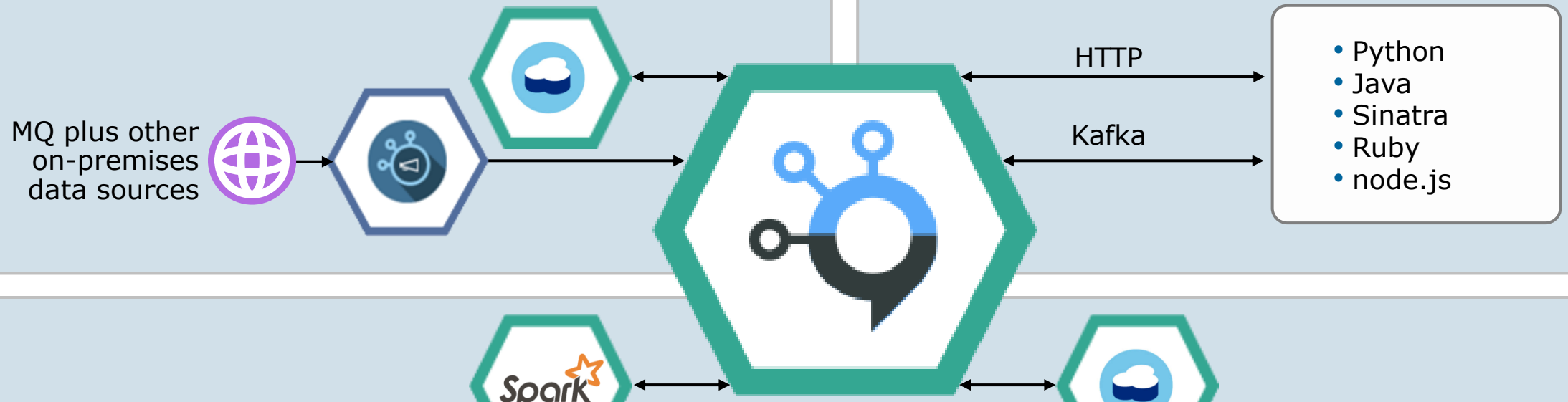
Hub for asynchronously connecting services inside Bluemix or beyond

Applications that are connected to events that happen in other Bluemix services, or from beyond the cloud

2

Microservices allow applications to evolve

Open protocols support polyglot run times, application-controlled behavior, and reactive scale



3

Insights from the data you already have

Data needs to be streamed from anywhere to one or many analytics engines

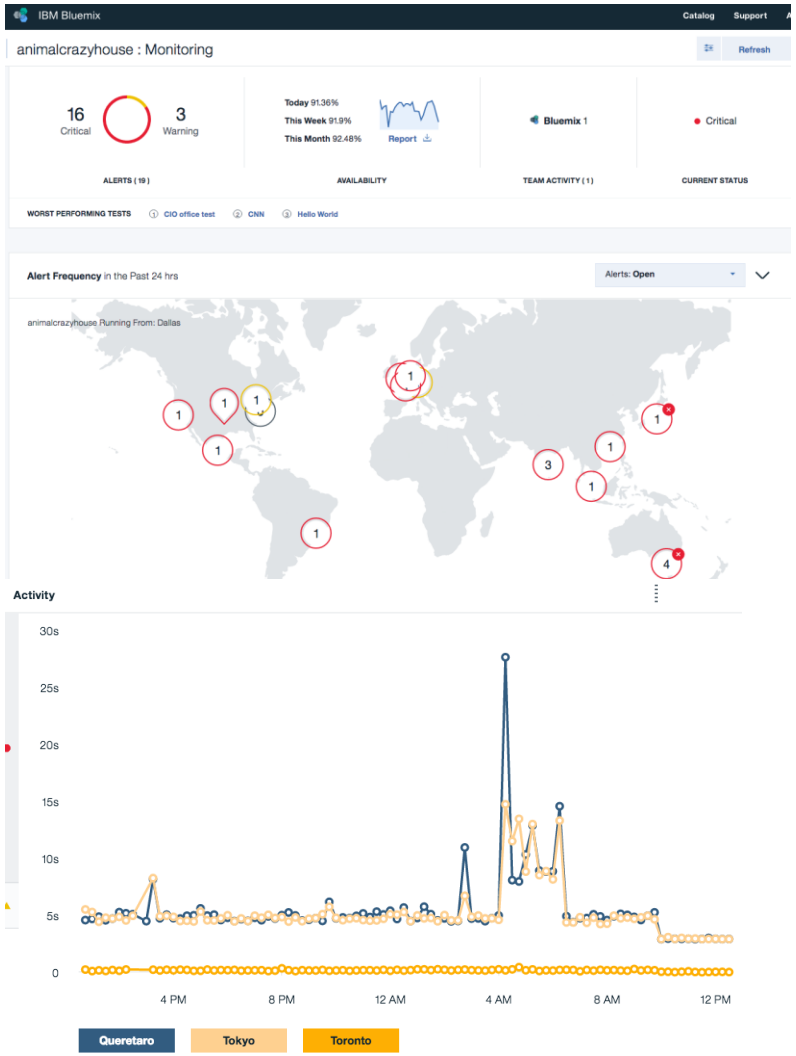
Auto-scaling

- 自动增加或减少应用程序的计算能力。应用程序实例数会根据定义的 Auto-Scaling 策略动态调整。

- 动态缩放
  - 自动添加或除去资源以匹配当前工作负载。
- 定制缩放策略
  - 定义感兴趣的度量值策略。
- 度量值统计信息
  - 可视化性能度量值的当前和历史记录值。
- 缩放历史记录
  - 根据状态、时间和类型查询缩放活动。

可用性监视

- 通过监视快速检测、隔离和诊断应用程序问题



目录

资源和参考

- IBM微服务参考架构
  - https://www.ibm.com/devops/method/content/architecture/omnichannelArchitecture
- IBM微服务基础系列培训课程
  - https://www-03.ibm.com/services/learning/ites.wss/zz-en?pageType=page&c=K356575R37836F54
- IBM微服务POV
  - https://developer.ibm.com/cloudarchitecture/docs/microservices/understanding-microservices-guide/
- IBM微服务决策指导
  - https://developer.ibm.com/cloudarchitecture/docs/microservices/microservices-decision-guide/
- IBM微服务系列红皮书
  - Evolve the Monolith to Microservices with Java and Node
  - Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach
  - Microservices Best Practices for Java
- IBM微服务参考架构实现
  - https://github.com/ibm-cloud-architecture/refarch-cloudnative
  - https://github.com/ibm-cloud-architecture/refarch-cloudnative-netflix
  - https://github.com/ibm-cloud-architecture/refarch-cloudnative-kubernetes/tree/kube-int