现代软件架构介绍

I - 微服务

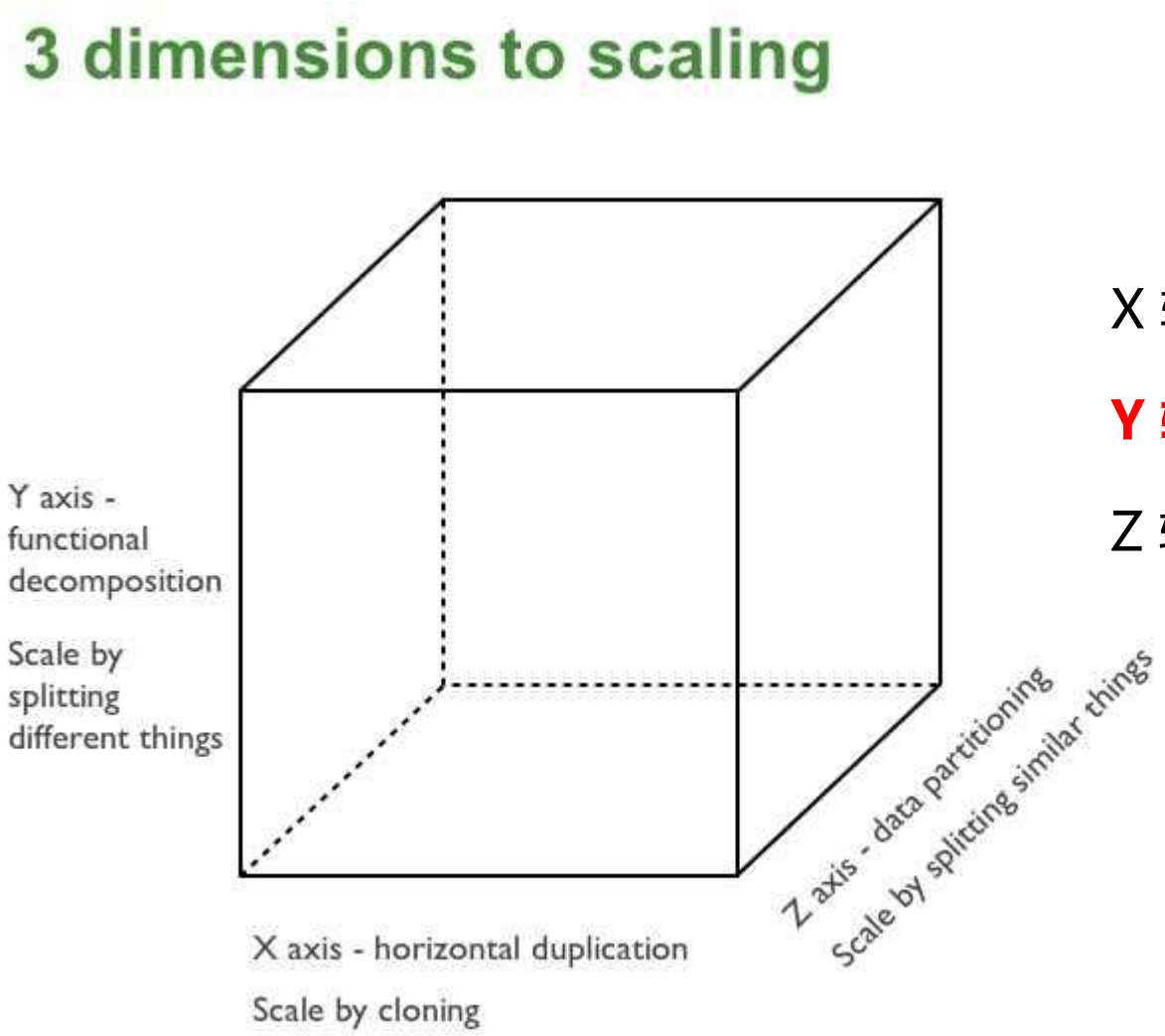
Microservices

现代软件架构群——从设计到实现

- 微服务（Microservices）及其模式
- 3 Dementions to Scaling
- 领域驱动设计（Domain Driven Design）
- 六边形架构（Hexagonal Architecture）
- RESTful Web services
- Spring Cloud + Spring Data
- Kubernetes based DevOps
- Native Cloud Application
- 面向对象设计（OOD）的模式

微服务是什么

软件可伸缩性——Scaling

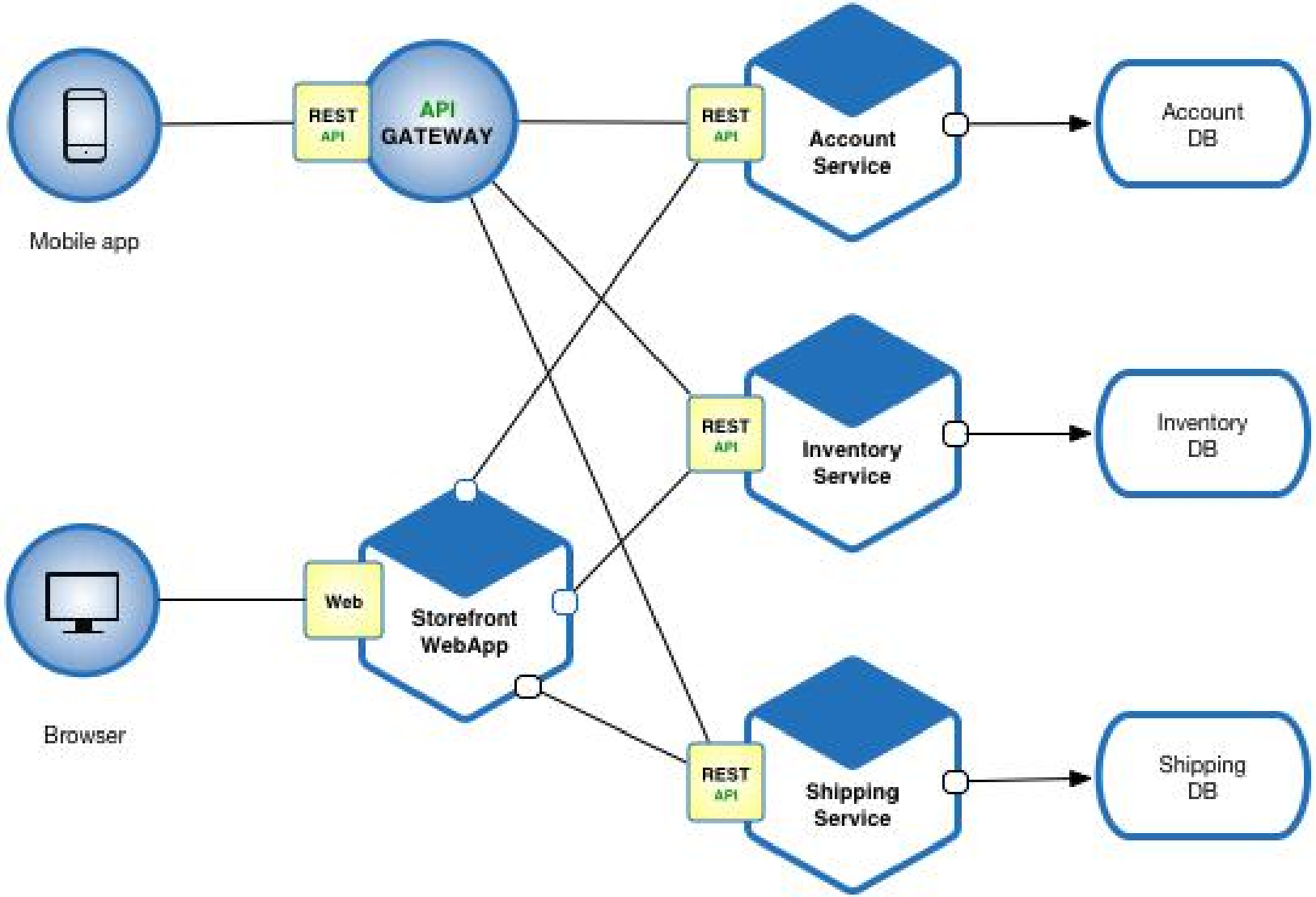


X 轴 - 应用程序多份拷贝，节点之间负载均衡

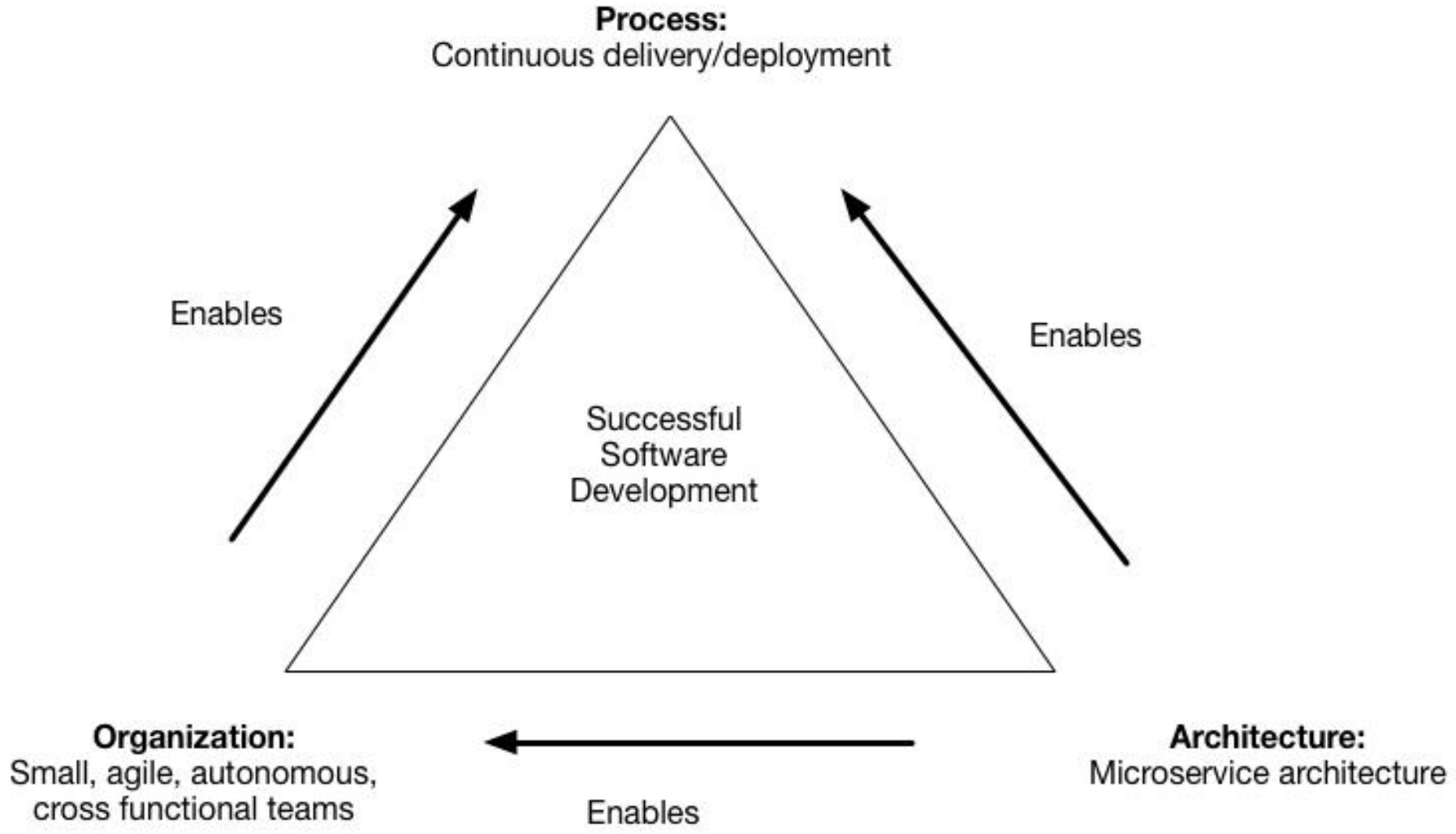
Y 轴 - 将应用程序切分为多个部分，多个服务

Z 轴 - 数据分片，各自处理不同数据子集

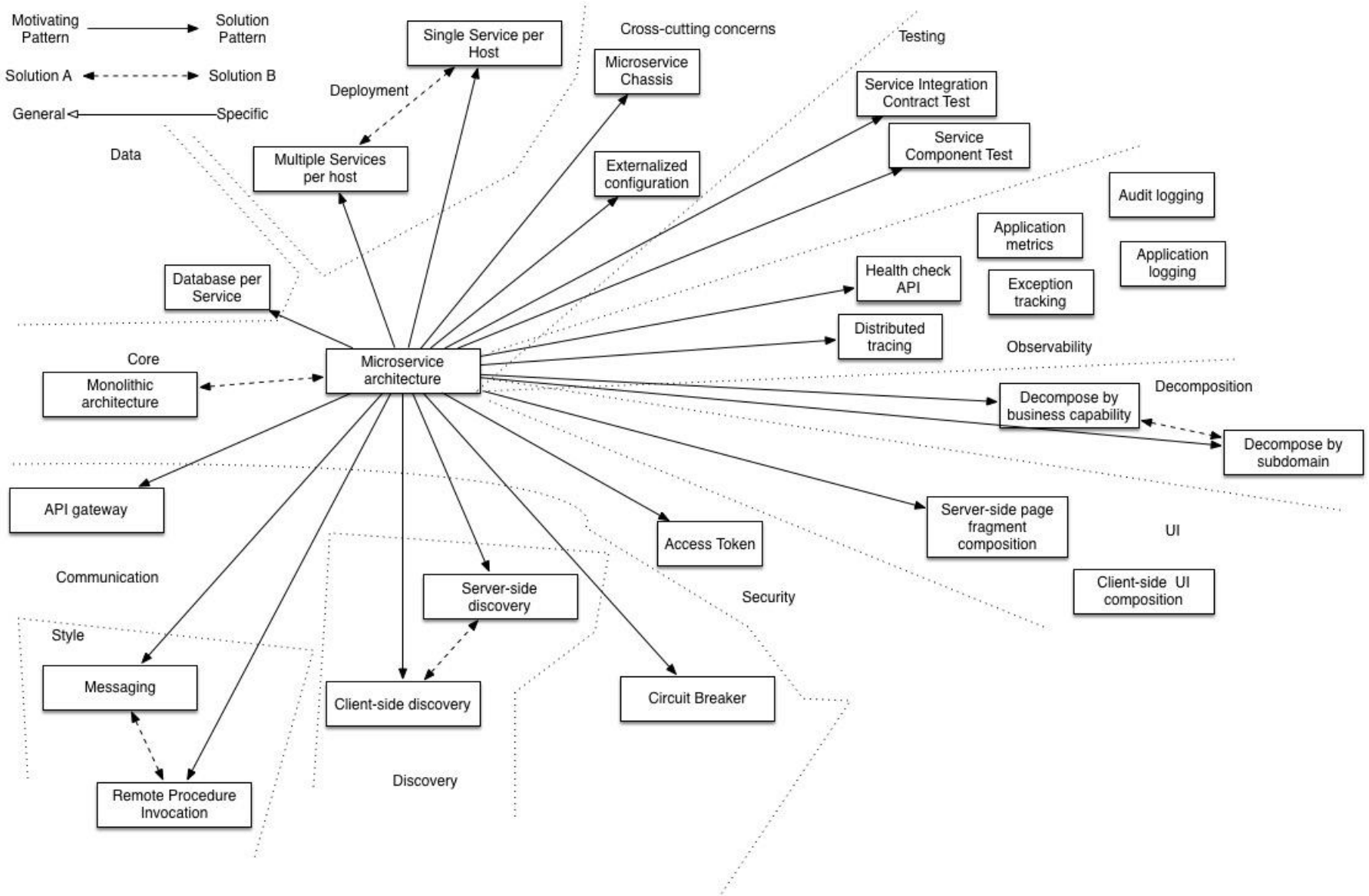
微服务——Y轴伸缩性范例


Mobile app
REST API
API GATEWAY
REST API
Account Service
Account DB
REST API
Inventory Service
Inventory DB
Browser
Web
Storefront WebApp
REST API
Shipping Service
Shipping DB

微服务的影响度


Process:
Continuous delivery/deployment
Enables
Enables
Successful
Software
Development
Organization:
Small, agile, autonomous,
cross functional teams
Enables
Architecture:
Microservice architecture

微服务的设计模式


Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Data
Deployment
Single Service per Host
Multiple Services per host
Cross-cutting concerns
Microservice Chassis
Externalized configuration
Testing
Service Integration Contract Test
Service Component Test
Audit logging
Application metrics
Application logging
Health check API
Exception tracking
Distributed tracing
Observability
Database per Service
Core
Monolithic architecture
Microservice architecture
Decomposition
Decompose by business capability
Decompose by subdomain
API gateway
Server-side page fragment composition
UI
Client-side UI composition
Access Token
Security
Communication
Server-side discovery
Style
Messaging
Client-side discovery
Circuit Breaker
Discovery
Remote Procedure Invocation

微服务的设计模式

- Decomposition patterns
  - Decompose by business capability
  - Decompose by subdomain
- The Database per Service pattern describes how each service has its own database in order to ensure loose coupling.
- The API Gateway pattern defines how clients access the services in a microservice architecture.
- The Client-side Discovery and Server-side Discovery patterns are used to route requests for a client to an available service instance in a microservice architecture.
- The Messaging and Remote Procedure Invocation patterns are two different ways that services can communicate.
- The Single Service per Host and Multiple Services per Host patterns are two different deployment strategies.
- Cross-cutting concerns patterns:
  - Microservice chassis pattern and Externalized configuration
- Testing patterns:
  - Service Component Test and Service Integration Contract Test
- Circuit Breaker
- Access Token
- Observability patterns:
  - Log aggregation
  - Application metrics
  - Audit logging
  - Distributed tracing
  - Exception tracking
  - Health check API
  - Log deployments and changes
- UI patterns:
  - Server-side page fragment composition
  - Client-side UI composition

微服务的特性

微服务

- English Article
- 中文翻译
- “Building Microservices” Sam Newman, 2015

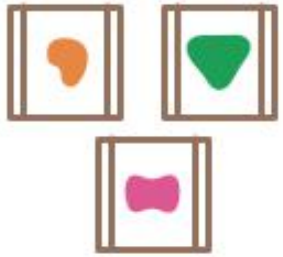
James Lewis

Martin Fowler

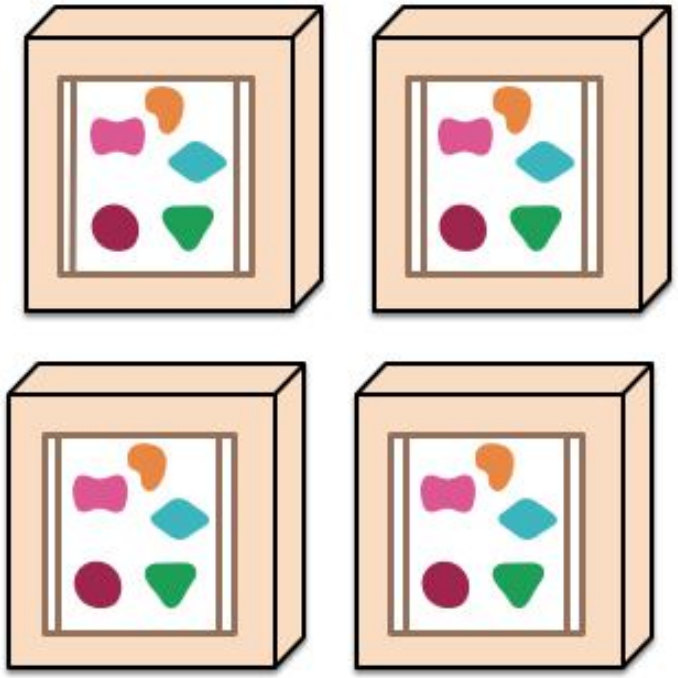
过去的单体风格——Monolithic style

一个单体应用程序把它所有的功能放在一个单一进程中...

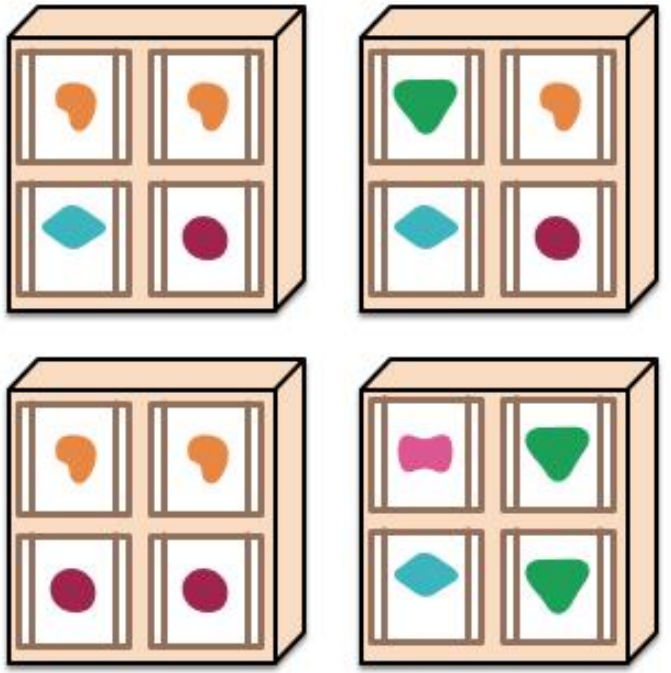
一个微服务架构把每个功能元素放进一个独立的服务中...



...并且通过在多个服务器上复制这个单体进行扩展



...并且通过跨服务器分发这些服务进行扩展，只在需要时才复制.



同单体应用相对照，微服务是什么

- 将一个单一应用程序开发为一组小型服务的方法，每个服务运行在自己的进程中，服务间通信采用轻量级通信机制(通常用HTTP资源API)。
- 这些服务围绕业务能力构建并且可通过全自动部署机制独立部署。
- 这些服务共用一个最小型的集中式的管理，服务可用不同的语言开发，使用不同的数据存储技术。

微服务架构的九大特性

- 特性一：通过服务组件化（Componentization via Services）
- 特性二：围绕业务能力组织（Organized around Business Capabilities）
- 特性三：是产品不是项目（Products not Projects）
- 特性四：智能端点和哑管道（Smart endpoints and dumb pipes）
- 特性五：去中心化治理（Decentralized Governance）
- 特性六：去中心化数据管理（Decentralized Data Management）
- 特性七：基础设施自动化（Infrastructure Automation）
- 特性八：为失效设计（Design for failure）
- 特性九：进化式设计（Evolutionary Design）

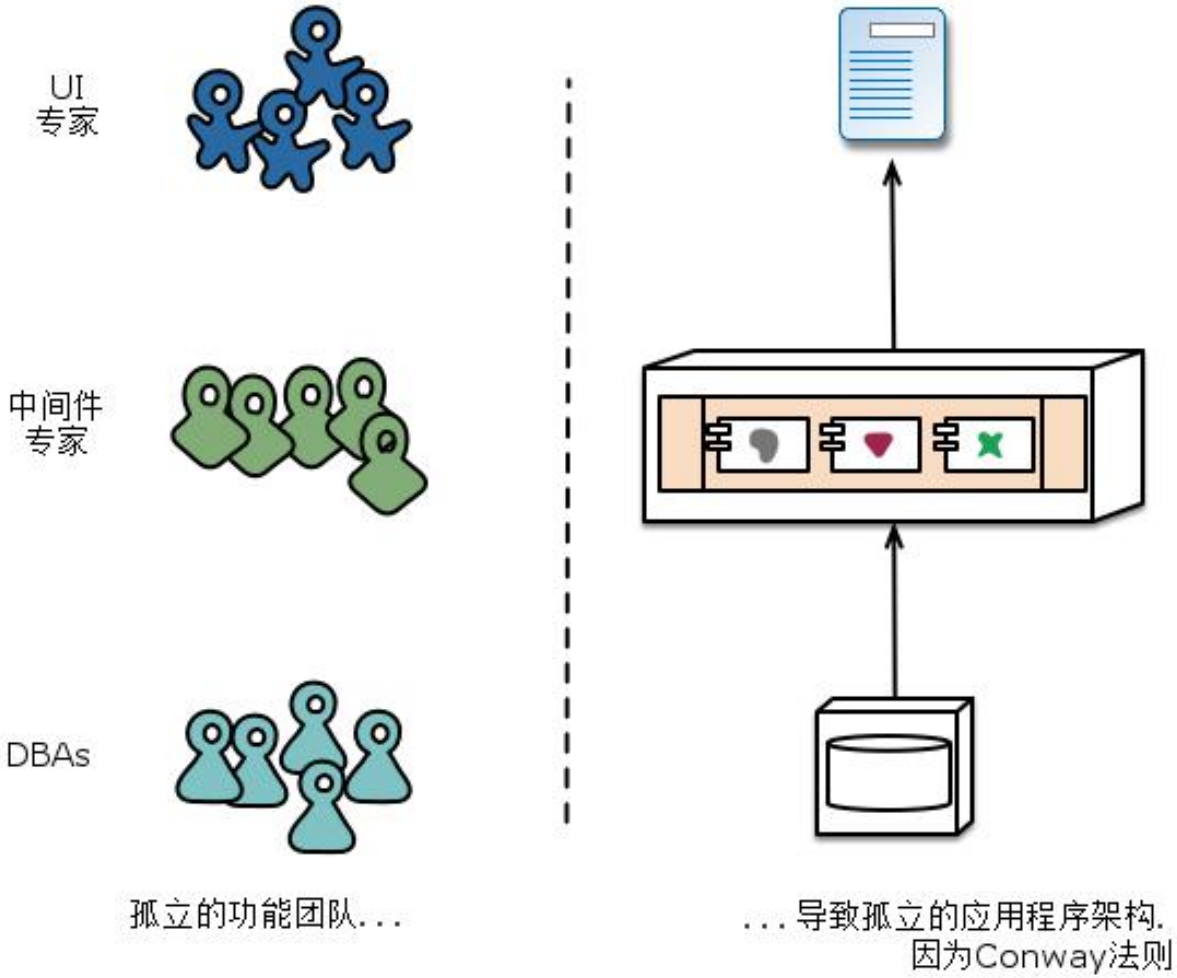
特性一：通过服务组件化

- 组件是一个可独立替换和独立升级的软件单元。
- 微服务架构组件化软件的主要方式是分解成服务。我们把库定义为链接到程序并使用内存函数调用来调用的组件，而服务是一种进程外的组件，它通过web服务请求或rpc机制通信。

- 优点：
  - 服务可独立部署
  - 更加明确的组件接口，松耦合
- 缺点：
  - 远程调用消耗大
- 解决方案：
  - 远程API被设计成粗粒度

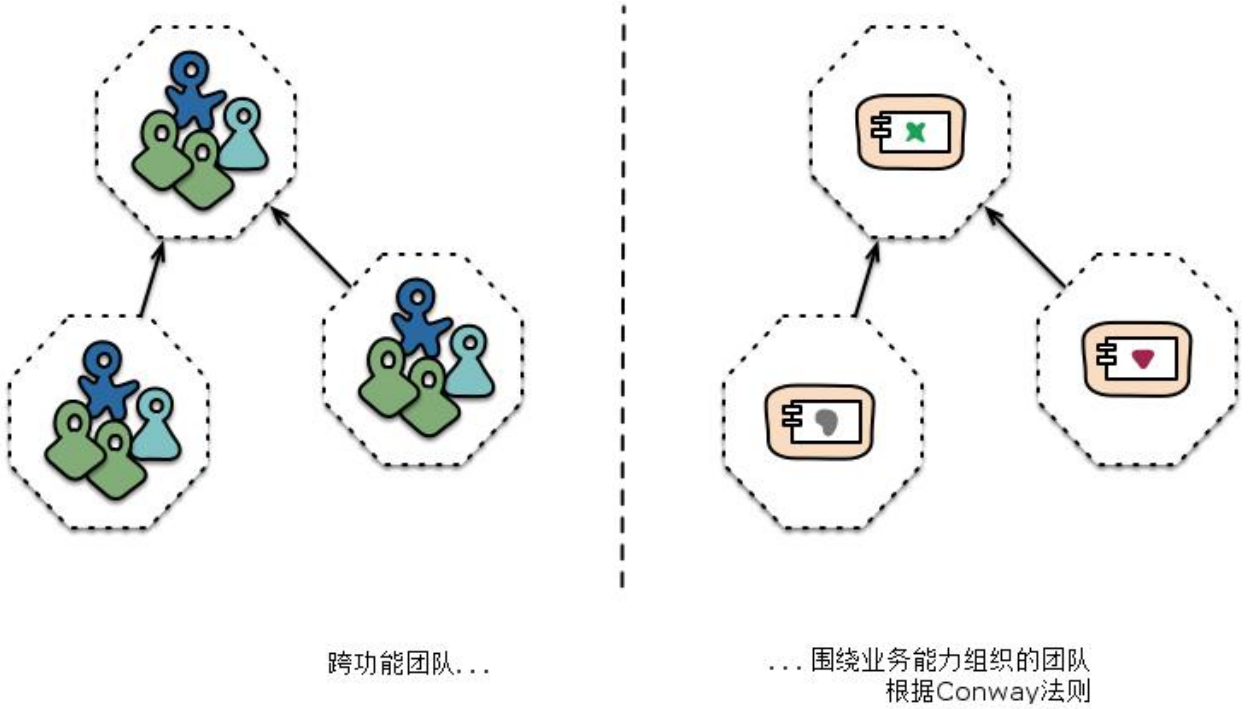
特性二：围绕业务能力组织

- 功能导向的组织结构匹配单体应用
- 即使是简单的更改也会导致跨团队的时间和预算审批



特性二：围绕业务能力组织

- 业务能力导向的组织结构匹配微服务架构
- 团队都是跨职能的，包括开发需要的全方位技能：用户体验、数据库、项目管理
- 服务组件所要求的更加明确的分离，使得它更容易保持团队边界清晰



特性三：是产品不是项目

- 通常的项目模式：目标是交付将要完成的一些软件。完成后的软件被交接给维护组织，然后它的构建团队就解散了。

- 微服务架构认为一个团队应该负责产品的整个生命周期。对此一个共同的启示是亚马逊的理念 “you build, you run it”，开发团队负责软件的整个产品周期。
- 产品思想与业务能力紧紧联系在一起。要持续关注软件如何帮助用户提升业务能力，而不是把软件看成是将要完成的一组功能。

特性四：智能端点和哑管道

- 企业服务总线(ESB)，在ESB产品中通常为消息路由、编排(choreography)、转化和应用业务规则引入先进的设施。

- 智能端点：尽可能的解耦和尽可能的内聚 - 他们拥有自己的领域逻辑，他们的行为更像经典UNIX理念中的过滤器 - 接收请求，应用适当的逻辑并产生响应。使用简单的REST风格的协议来编排他们，而不是使用像WS-Choreography或者BPEL或者通过中心工具编制(orchestration)等复杂的协议。
- 哑管道：在轻量级消息总线上传递消息。即像RabbitMQ或ZeroMQ这样只充当消息路由器的消息机制。

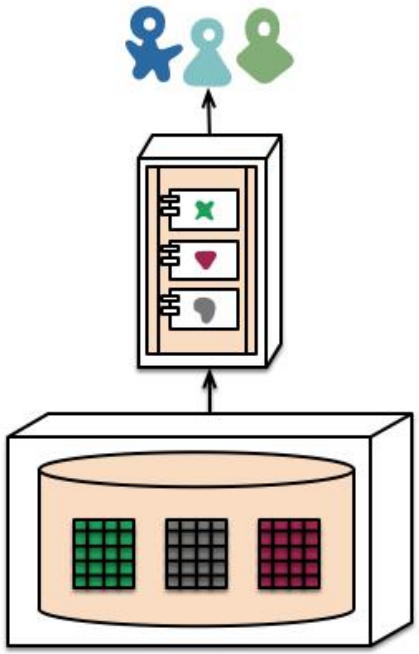
特性五：去中心化治理

- 单体应用倾向于用一套标准化的技术平台解决所有问题
- 类似于ESB的中心标准的复杂性超出控制
- * SOA与微服务，前者通过ESB强化了中心化的依赖，与微服务有本质的区别

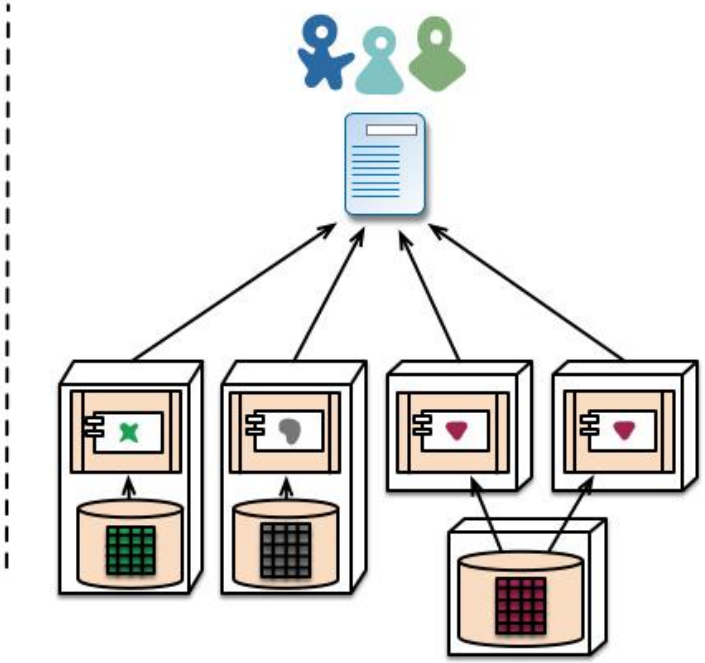
- 微服务架构下可以根据不同的服务边界采用更合适的技术
- 服务之间的松耦合、明确的服务接口降低了“中心化”依赖
- 粗粒度通信代替细粒度通信
- 采用Decompose by business capability和Decompose by subdomain模式来划分服务边界

特性六：去中心化数据管理

- 数据库供应商授权许可的商业模式导致尽量少的数据库现状
- 微服务更倾向于让每个服务管理自己的数据库，或者同一数据库技术的不同实例，或完全不同的数据库系统 - 这就是所谓的混合持久化
- 2 phase-commit不是可选方案
- 微服务架构强调服务间的无事务协作，对一致性可能只是最终一致性和通过补偿操作处理问题

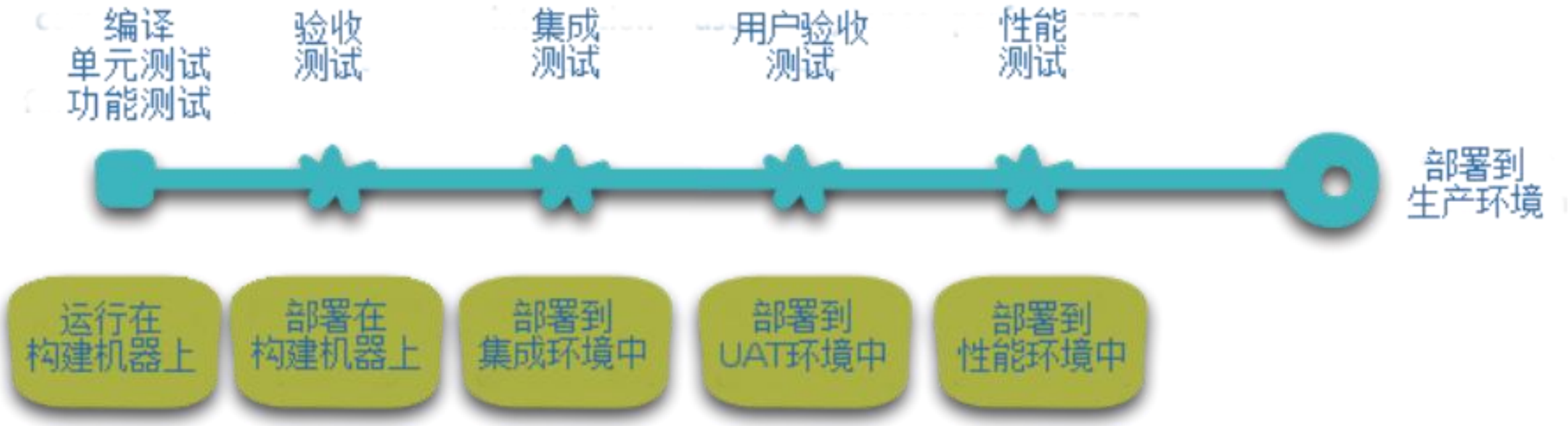


单体 – 单一数据库



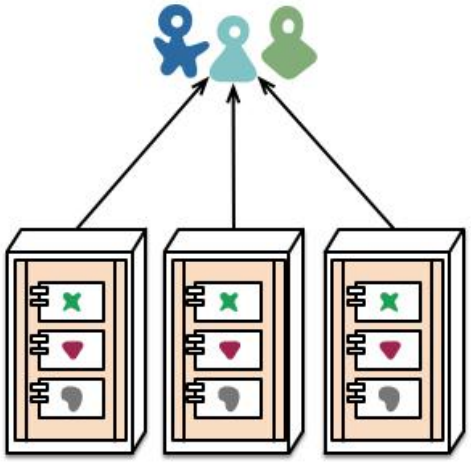
微服务 – 应用程序数据库

特性七：基础设施自动化

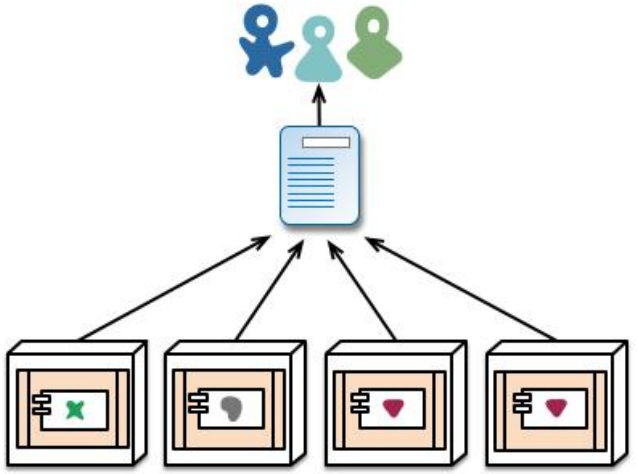

编译
单元测试
功能测试
验收
测试
集成
测试
用户验收
测试
性能
测试
部署到
生产环境
运行在
构建机器上
部署在
构建机器上
部署到
集成环境中
部署到
UAT环境中
部署到
性能环境中

特性七：基础设施自动化

- 一旦为单体投入了自动化生产之路，那么部署更多的应用程序似乎也不会更可怕
- 持续部署的目标之一是使部署枯燥，所以无论是一个或三个应用程序，只要它的部署仍然枯燥就没关系
- 作为持续交付和持续部署的一个后果，增加自动化的一个副作用是创造有用的工具，以帮助开发人员和运营人员。



单体 – 多个模块在同一个进程中



微服务 – 每个模块运行在不同的进程中

特性八：为失效设计

- 单体应用中的组件都是进程内的，组件失败的影响度较小，而跨进程的组件连接性故障更容易产生，并导致严重后果
- * 同步调用被认为是有害的，在服务间有大量的同步调用，将遇到停机的乘法效应。简单地说，就是系统的停机时间与各个组件停机时间的乘积。面临一个选择，让调用变成异步或者管理停机时间。微服务风格更倾向于前者

- 使用服务作为组件的一个结果是，应用程序需要被设计成能够容忍服务失效。
- 与单体应用设计相比这是一个劣势，因为它引入额外的复杂性来处理它。
- 快速发现故障，尽可能自动恢复服务，语义监测（业务/非业务访问指标）可以提供一套早期预警系统，触发开发团队跟进和调查。
- 每个单独的服务需要完善的监控和日志记录，断路器状态、当前吞吐量和时延...

特性九：进化式设计

- 通过服务分解来控制变更而不是减缓变更，用正确的态度和工具，可以频繁、快速且控制良好的改变软件
- 组件的关键特性是独立的更换和升级的理念 - 这意味着我们要找到这样的点，我们可以想象重写组件而不影响其合作者
- 系统中很少变更的部分应该和正在经历大量扰动的部分放在不同的服务里。如果你发现你自己不断地一起改变两个服务，这是它们应该被合并的一个标志
- 更细粒度的发布，只需要重新部署修改的服务，可以简化和加速发布过程
- 通过设计成对服务的提供者的变化尽可能的宽容来应对服务变更导致的阻断

微服务的优劣

优势	劣势
强化的模块边界 Strong Module Boundaries	分布式 Distribution
自治系统，便于部署 Independent Deployment	最终一致性 Eventual Consistency
实现技术多样化 Technology Diversity	运维复杂 Operational Complexity

参考资料

https://martinfowler.com/articles/microservices.html
http://microservices.io/index.html
http://dddcommunity.org/
https://www.infoq.com/minibooks/domain-driven-design-quickly
http://www.infoq.com/cn/minibooks/domain-driven-design-quickly
http://static.olivergierke.de/lectures/ddd-and-spring/
http://tpierrain.blogspot.hk/2016/04/hexagonal-layers.html
http://fideloper.com/hexagonal-architecture
http://12factor.net/
http://microservices.io/articles/scalecube.html
http://theartofscalability.com/
http://microservices.io/patterns/microservices.html