WEEK #1 初识微服务

[StuQ-1160] 跟我做一个Java微服务项目

讲师：刘俊强 https://junq.io

StuQ 斯达克学院微信公众号

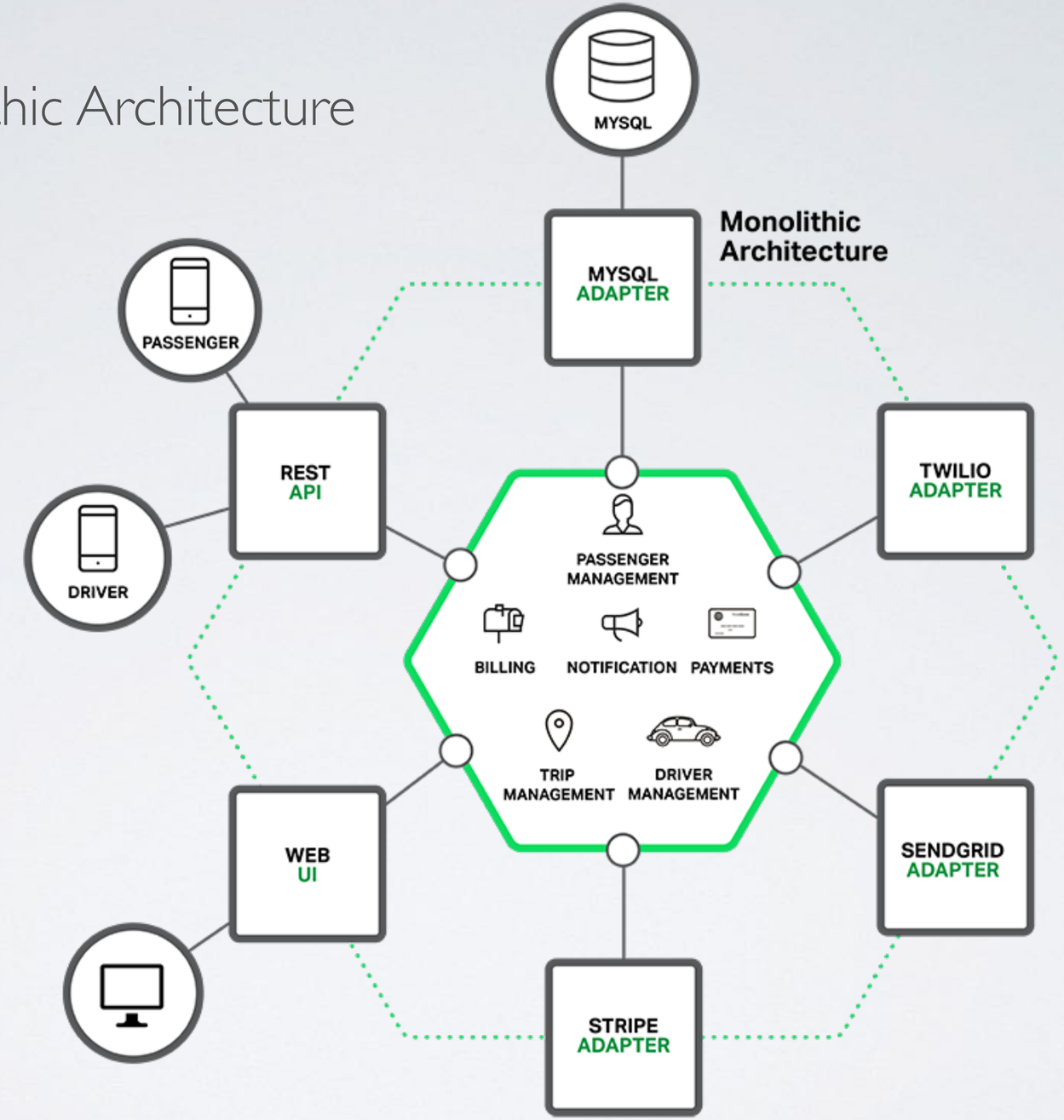
我的个人微信公众账号

AGENDA

1. 服务架构设计发展
2. 微服务简介
3. 自动化环境
4. 示例项目eMall介绍

服务架构设计发展

单体架构 Monolithic Architecture



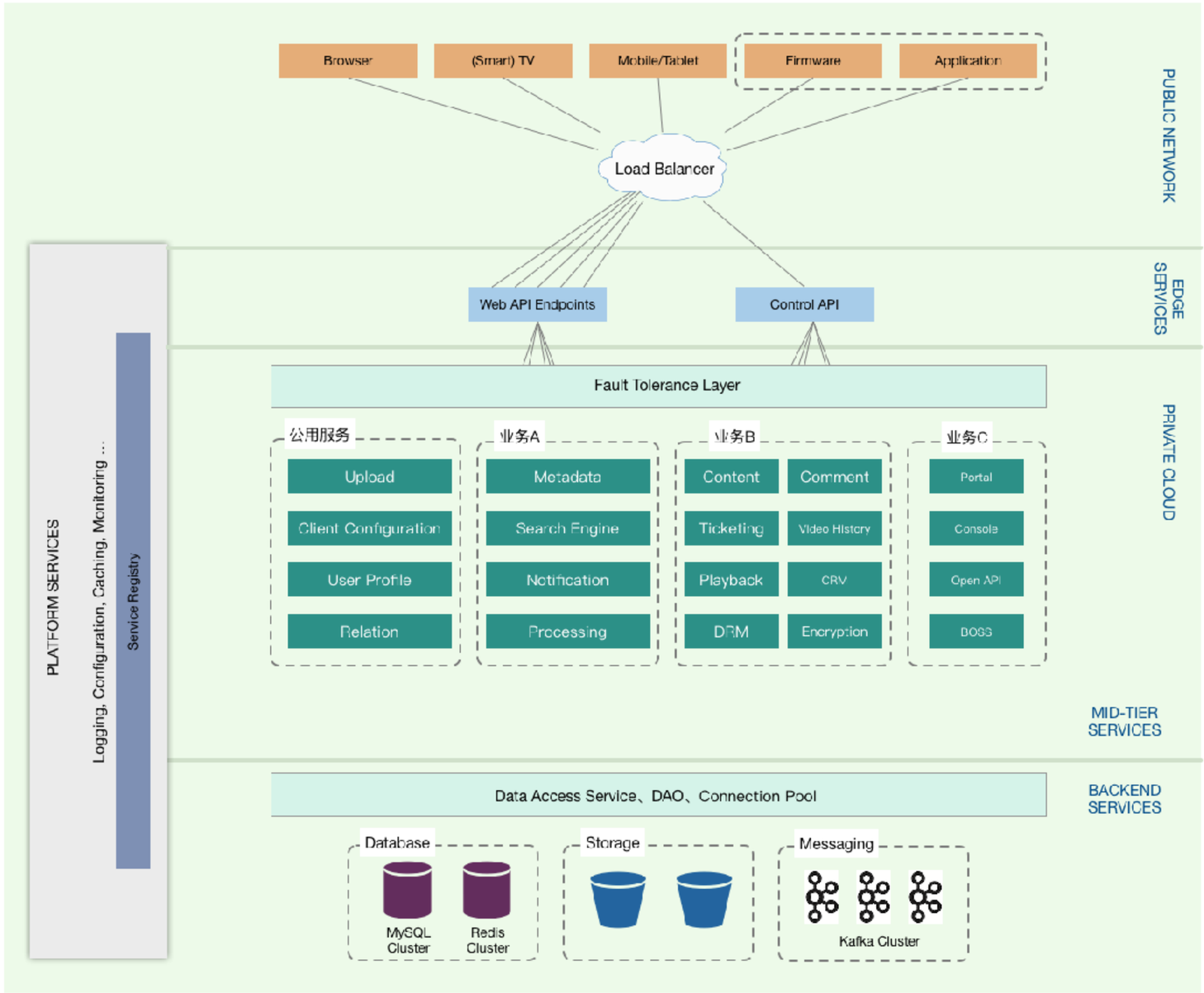
单体架构特点及好处

- 单一代码库、IDE友好
- 容易部署，单一WAR包
- 开发模型简单，一份代码库进行编码、构建和部署
- 简单可伸缩性，在负载均衡器后运行多份应用副本
- 技术栈单一

单体架构的问题

- 庞大的代码库，关系错综复杂
- 编译构建时间很长，与持续集成配合麻烦
- 部署臃肿，影响部署速度与频次
- 扩展能力与弹性受限
- 新技术与工具框架使用会受限

服务化架构 SOA - Service Oriented Architecture



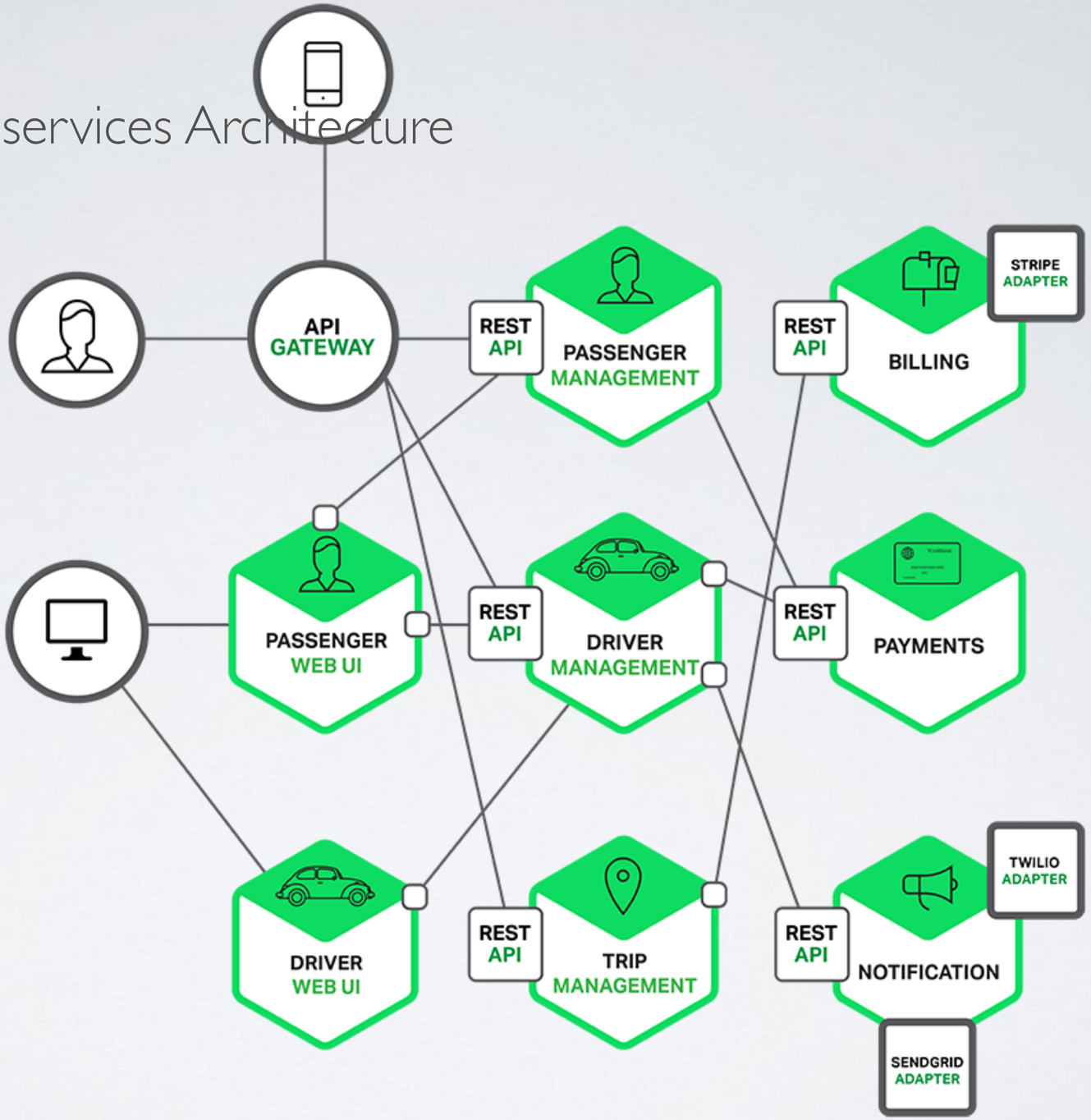
服务化架构

- 对业务进行分层，通常表现层（前端）、公共服务、业务逻辑服务、数据访问层等
- 对业务进行解耦，通过Pub-Sub或RPC进行服务间调用关系解耦
- 服务独立性，多数服务可以进行独立打包、发布
- 通常每个服务的技术栈单一
- 部署简单，具备可伸缩性

服务化架构的问题

- 对于部分服务而言，代码库依然巨大
- 打包、发布、部署流程不足够好
- 可伸缩性变强，但依然不够好
- 维护团队间沟通受阻，技术经验有效传递不够
- 服务增多对开发人员不够友好

微服务架构 Microservices Architecture



架构设计发展

MVC  SOA  Microservices

架构设计发展

MVC  SOA  Microservices

视图、业务逻辑、前后端间分层

架构设计发展

MVC → SOA → Microservices

视图、业务逻辑、前后端间分层

大型系统分层、解耦，标准接口调用，分布式系统

架构设计发展

MVC → SOA → Microservices

视图、业务逻辑、前后端间分层

大型系统分层、解耦，标准接口调用，分布式系统

云计算时代产物，关注敏捷交付和部署速度、频次

微服务简介

In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler

In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler

- suit of small services：由一系列小服务组成，没错，“微”即是小；
- running in its own process：每个服务运行于自己的独立进程；
- built around business capabilities：围绕着业务功能进行建模；
- independently deployable：每个服务可进行独立部署；
- bare minimum of centralized management：最低限度的集中管理。

微服务的特征

- 每个微服务都是业务完整的
  - 接口及界面呈现、业务逻辑、数据管理
- 每个微服务仅对一个业务负责，如
  - 产品服务、评价服务、支付服务、订单服务
- 每个微服务接口明确定义
  - 且保持不变，接口消费者只关注接口，对微服务不具备依赖
- 独立部署、升级及伸缩

服务的独立性与自主性

微服务的独立性与自主性

- 微服务间的独立性是关键
  - 代码库独立
  - 技术栈独立
  - 可伸缩性、可扩展性独立
  - 还有业务功能等

独立的代码库

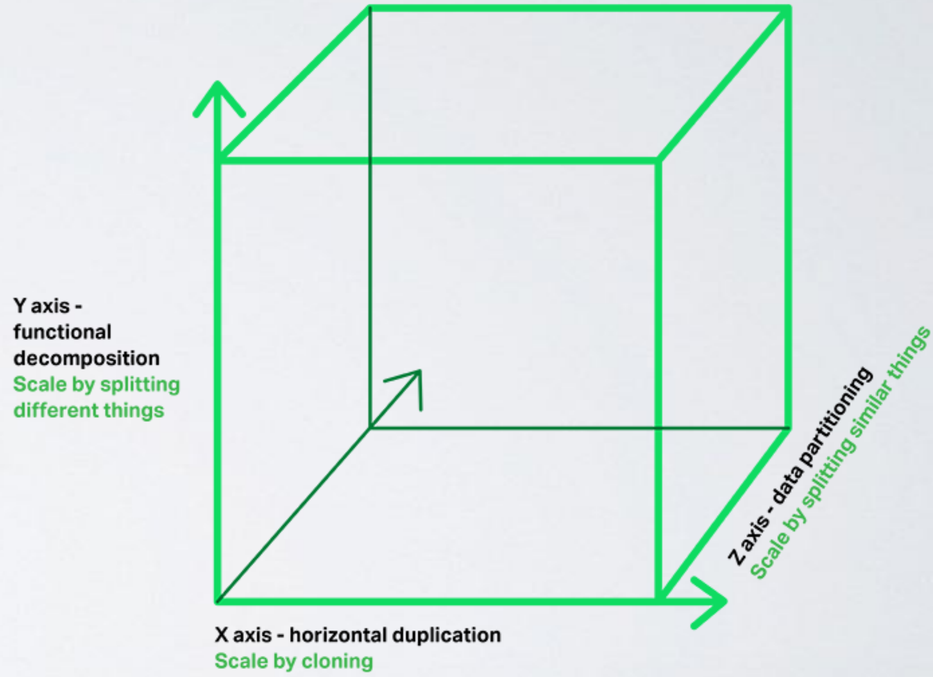
- 每个微服务具备自己的代码仓库
  - 由对应团队开发者维护
  - 编译、打包、发布及部署都很快
  - 服务启动迅速
  - 在各个服务的代码库间没有交叉依赖

技术栈独立

- 每个微服务都有自己独立的技术栈实现
  - 根据业务实现需求来选择最合适的技术栈
  - 团队可以尝试新的技术、工具或框架
  - 所选技术栈一般来说都很轻量级
  - 不需要同一标准化技术栈的选择，这样
    - 无需针对技术选型而纠结，关注于业务实现
    - 无需引入未被使用的技术或库

独立的可伸缩性

- 每个微服务都可以独立地伸缩
  - 可以更加直观定位性能瓶颈
  - 数据库分片可以根据需求来



业务功能独立

- 每个微服务可以在不影响其他微服务的情况下进行功能扩展
  - 例如，更新新版本界面，不需要更新整个系统
  - 可以进行整个业务功能的重写，并替换之
- **注意：保证接口明确定义且稳定**

服务的弹性与容错性

服务的弹性和容错性

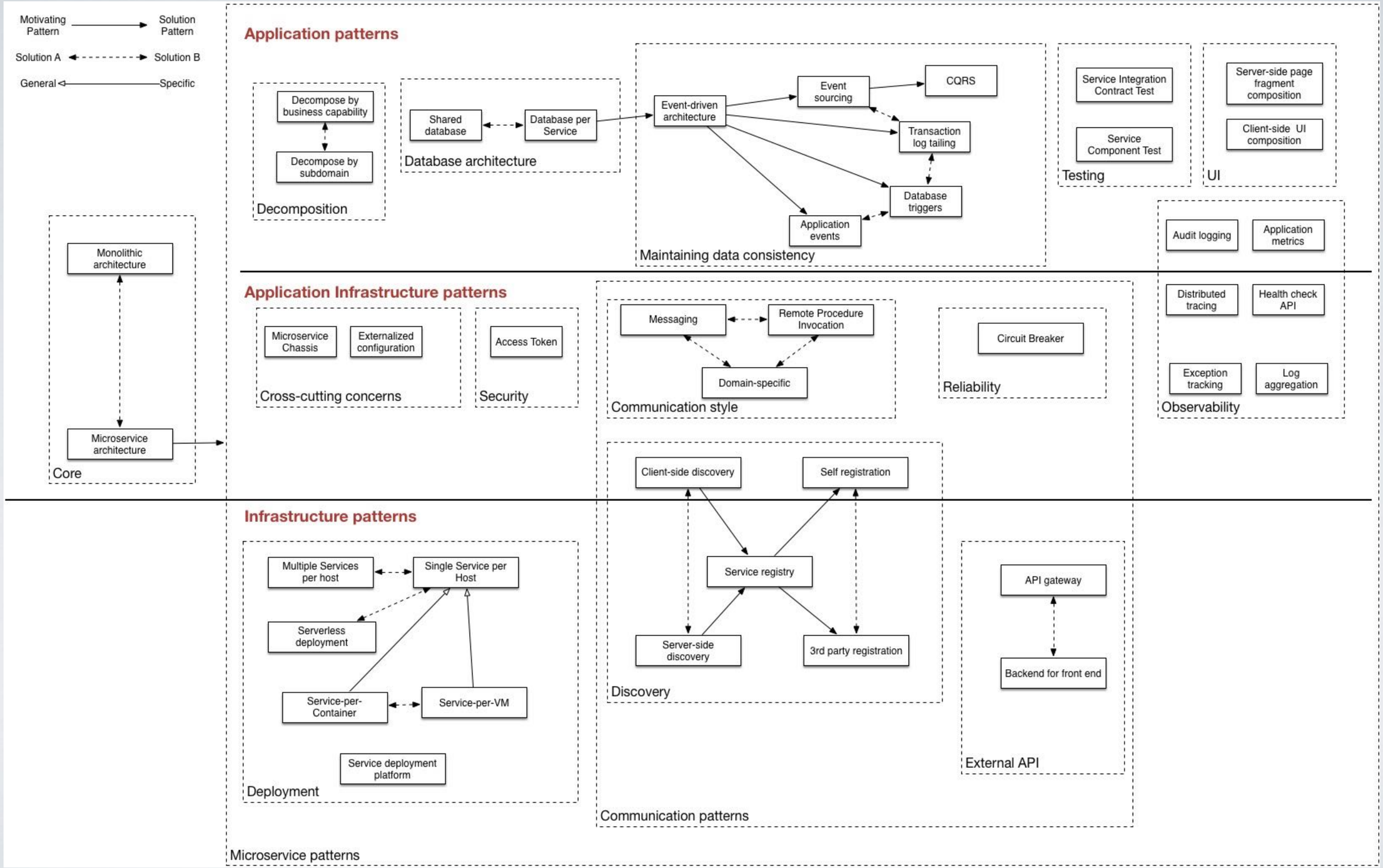
- 稳定的接口、标准化的通信是前提
  - 一般使用HTTP(S)、REST、JSON
- 微服务带来的运维复杂度由自动化环境来降低
- API Gateway来简化多个微服务API使用
- 使用断路器增强可靠性
- 服务发现机制

自动化环境

自动化环境

- 将微服务部署至海量环境时，需准备的有：
  - 持续部署（Continuous Delivery）
  - 健康监测及监控
  - 服务恢复机制

示例项目EMALL


Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Application patterns
Decompose by business capability
Decompose by subdomain
Decomposition
Shared database
Database per Service
Database architecture
Event-driven architecture
Event sourcing
CQRS
Transaction log tailing
Database triggers
Application events
Maintaining data consistency
Service Integration Contract Test
Service Component Test
Testing
Server-side page fragment composition
Client-side UI composition
UI
Audit logging
Application metrics
Distributed tracing
Health check API
Exception tracking
Log aggregation
Observability
Monolithic architecture
Microservice architecture
Core
Application Infrastructure patterns
Microservice Chassis
Externalized configuration
Cross-cutting concerns
Access Token
Security
Messaging
Remote Procedure Invocation
Domain-specific
Communication style
Circuit Breaker
Reliability
Infrastructure patterns
Multiple Services per host
Single Service per Host
Serverless deployment
Service-per-Container
Service-per-VM
Service deployment platform
Deployment
Client-side discovery
Self registration
Service registry
Server-side discovery
3rd party registration
Discovery
API gateway
Backend for front end
External API
Communication patterns
Microservice patterns

EMALL架构示意图