Pivotal

微服务实践之路

刘凡
资深云平台架构师
13661145645

Microservices vs Monoliths

Monoliths

Almost all the successful microservice stories have started with a monolith that got too big and was broken up.

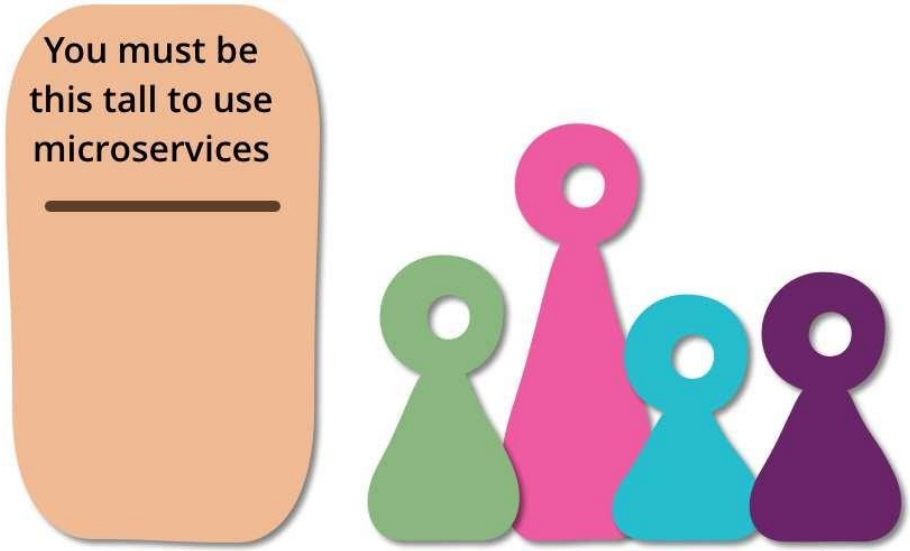
Microservices system

Almost all the cases where I've heard of a system that was built as a microservice system from scratch, it has ended up in serious trouble.

Source: http://martinfowler.com/bliki/MonolithFirst.html Martin Fowler

Pivotal

Are you tall enough?



Source: martinfowler.com/bliki/MicroservicePrerequisites.html

Microservice Architecture Challenges

云原生 / 微服务架构的复杂性

配置管理

服务注册发现

负载均衡

全链路跟踪

弹性伸缩

优雅降级

流量控制

API网关

日志集中管理

应用服务故障自愈

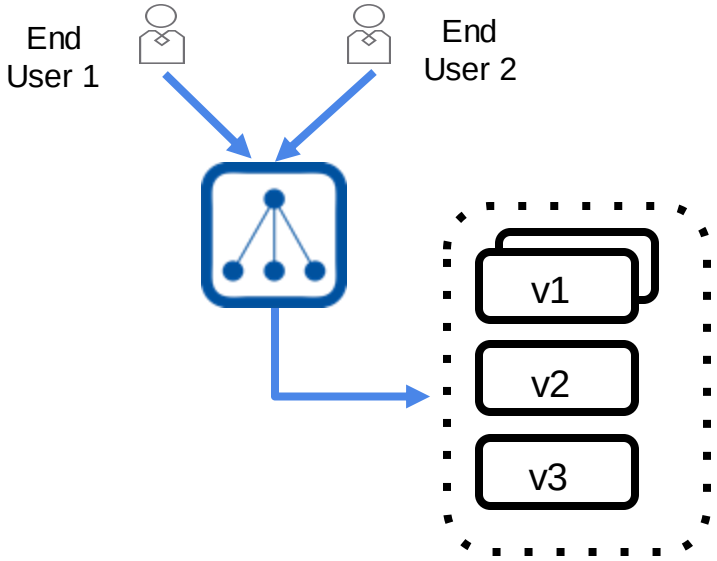
打包，部署，调度

应用监控度量

安全

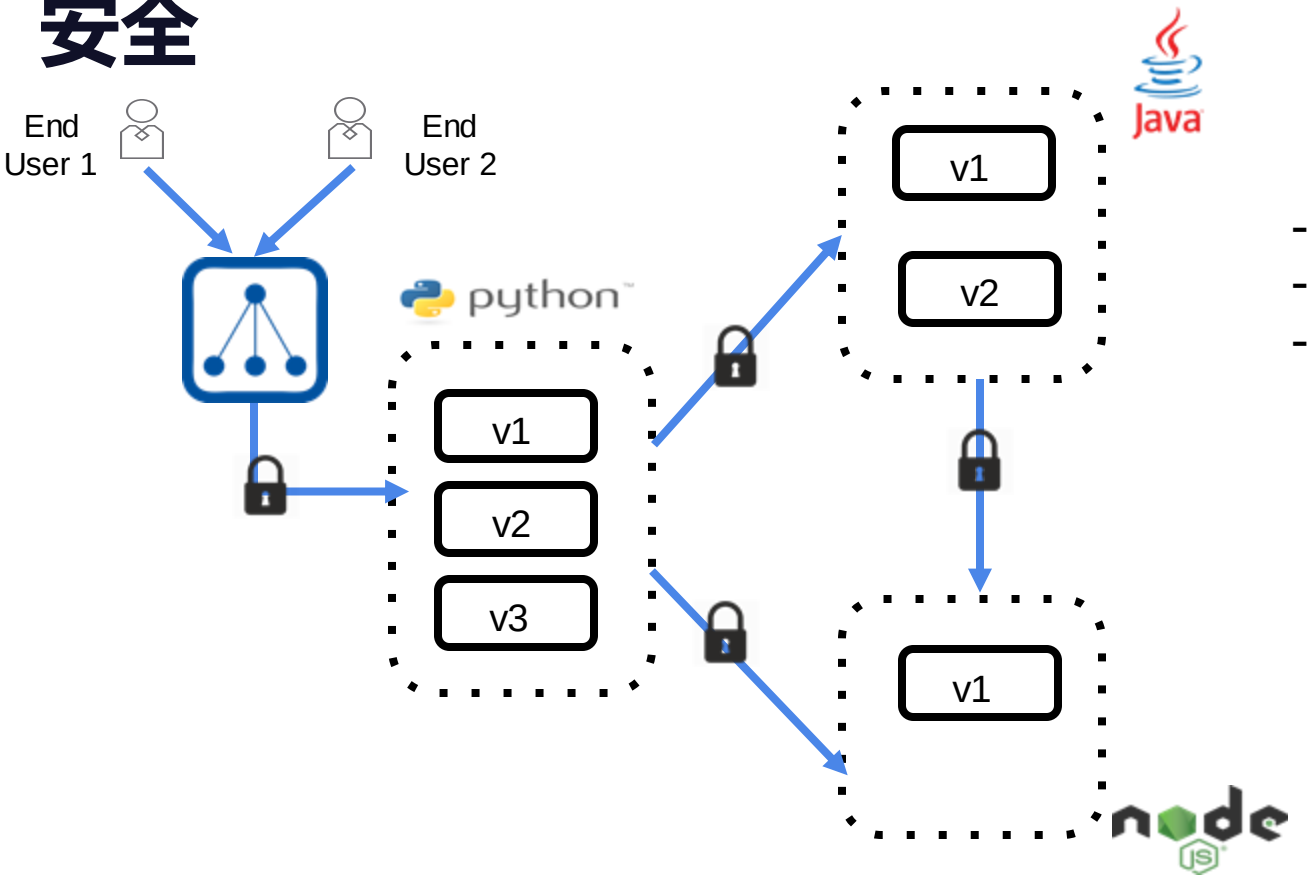
Pivotal

微服务生命周期



- 支持多实例
- 滚动升级到新版本
- A/B Testing测试新功能
- 跨云支持

安全



- 认证和鉴权
- 零信任
- 深度防御

微服务运行时的挑战

- 开发人员关心
  - 负载均衡
  - 流量管理
  - 熔断机制
  - 客户端重试机制
  - 。。。
- 运维人员关心
  - 加密通信机制
  - 认证，鉴权以及安全审计
  - 应用度量和可观察性
  - 安全策略
  - 。。。

如何应对微服务运行时挑战?

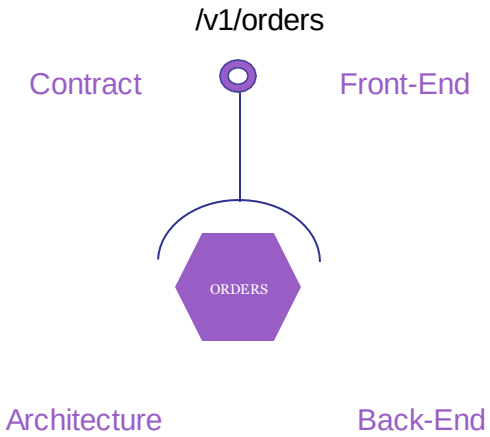
- 微服务治理
  - API Gateway
  - Spring Cloud家族
- 分布式事务
- 微服务生命周期管理
  - CI/CD
  - 云平台

如何应对微服务运行时挑战?

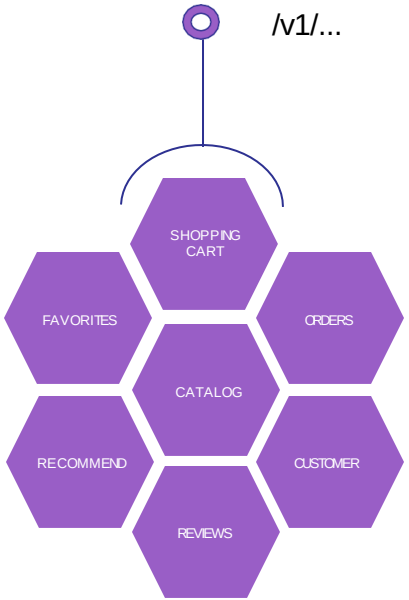
- 微服务治理
  - API Gateway
  - Spring Cloud家族
- 分布式事务
- 微服务生命周期管理
  - CI/CD
  - 云平台

Why API Gateway?

Microservice vs API



在微服务架构体系中，API Gateway是必不可少的

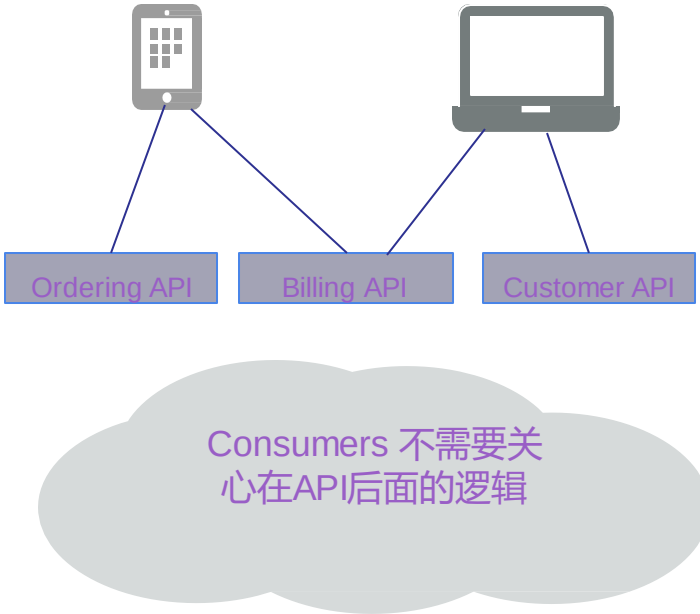


Microservices eCommerce System

如果没有API Gateway，那么?

- 点对点的服务调用会非常混乱
- 非统一的接口和协议来使用服务
- 客户端直接与服务通信，重构微服务将是一项挑战
- 如果客户端的需求与每个微服务公开的细粒度API之间存在不匹配，则客户端必须进行多次调用才能获得所需的内容。在复杂的应用程序中，这可能意味着数百个服务调用。

API使消费者免受微服务复杂性的影响



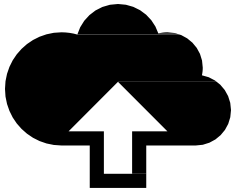
API网关的作用

- 智能路由：限流，降级，负载均衡，弹性伸缩，服务发现等
- 监控：不光资源监控，还有利于分析和洞察业务指标
- 发布：金丝雀/蓝绿发布/AB Testing
- 安全：authN, authZ，访问权限控制
- 灵活性：巨石应用的绞杀式改造，协议转换，聚合编排，简化客户端代码

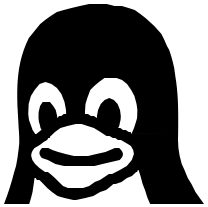
网关的类型

Appliance

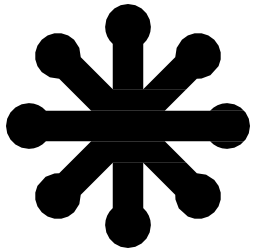
SAAS



Web Server



Mesh



Developer Oriented

Spring Cloud Gateway

Spring Cloud Gateway主要特性

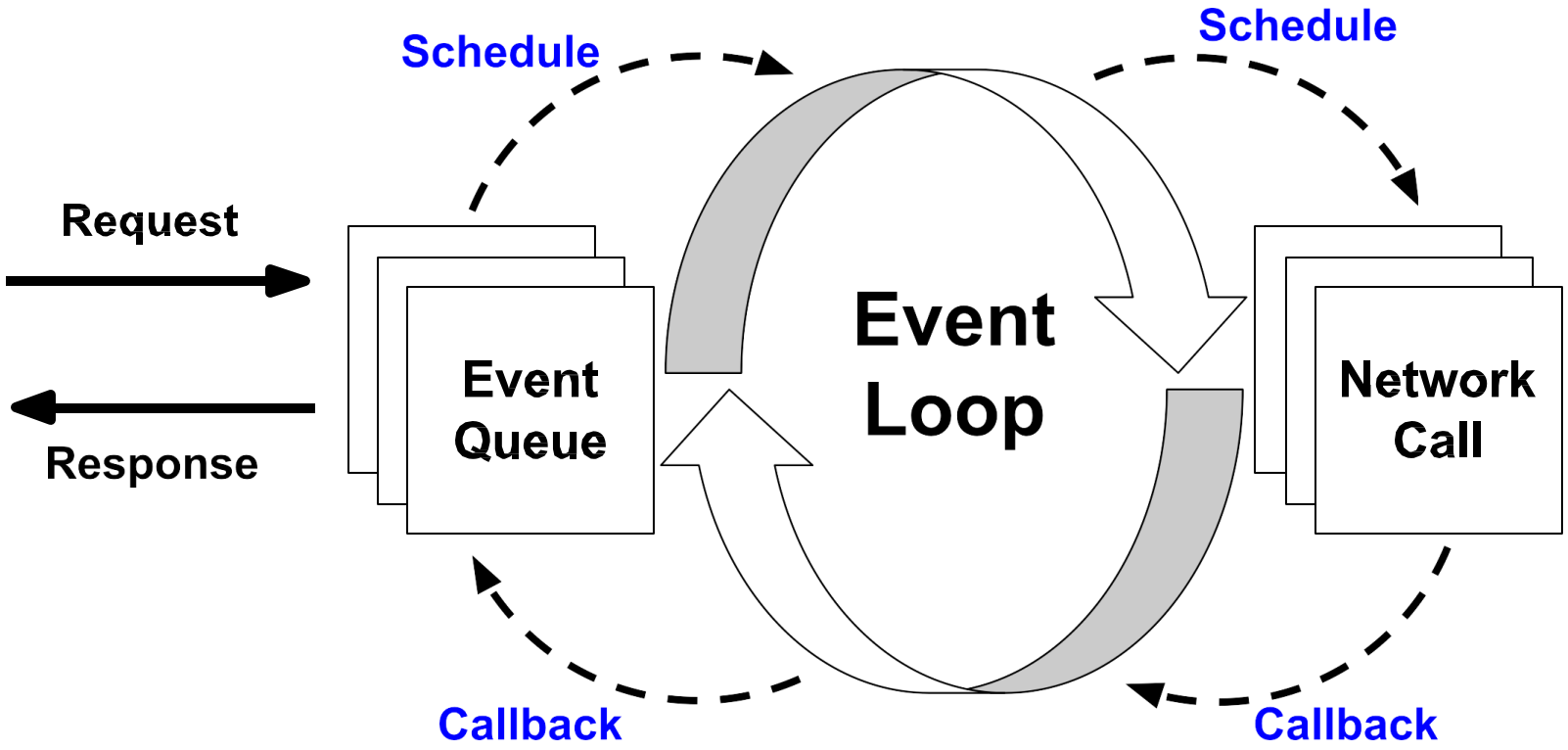
- 基于 Spring Framework 5，Project Reactor 和 Spring Boot 2.0
- 可以基于请求的属性进行匹配的动态路由
- Predicates 和 Filters 可作用于特定路由
- 集成了 Hystrix 断路器
- 集成了 Spring Cloud DiscoveryClient
- 易于编写的 Predicates 和 Filters
- 限流
- 路径重写

Reactive

Project

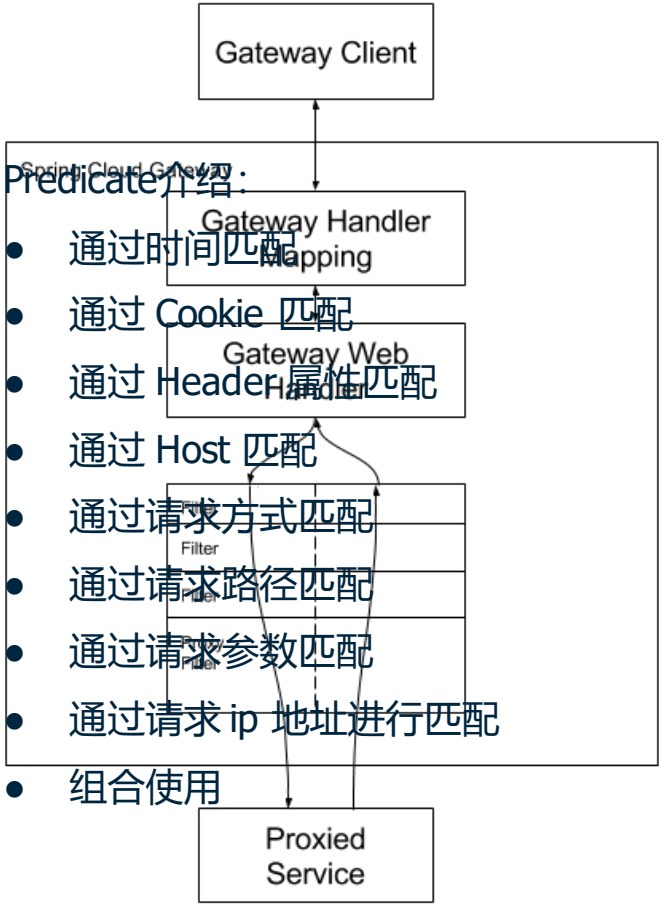
Reactor

Non-blocking



基本概念和工作原理

- Route（路由）：这是网关的基本构建块。它由一个 ID，一个目标 URI，一组断言和一组过滤器定义。如果断言为真，则路由匹配。
- Predicate（断言）：这是一个 Java 8 的 Predicate。我们可以使用它来匹配来自 HTTP 请求的任何内容，例如 headers 或参数。
- Filter（过滤器）：我们可以使用它修改请求和响应。



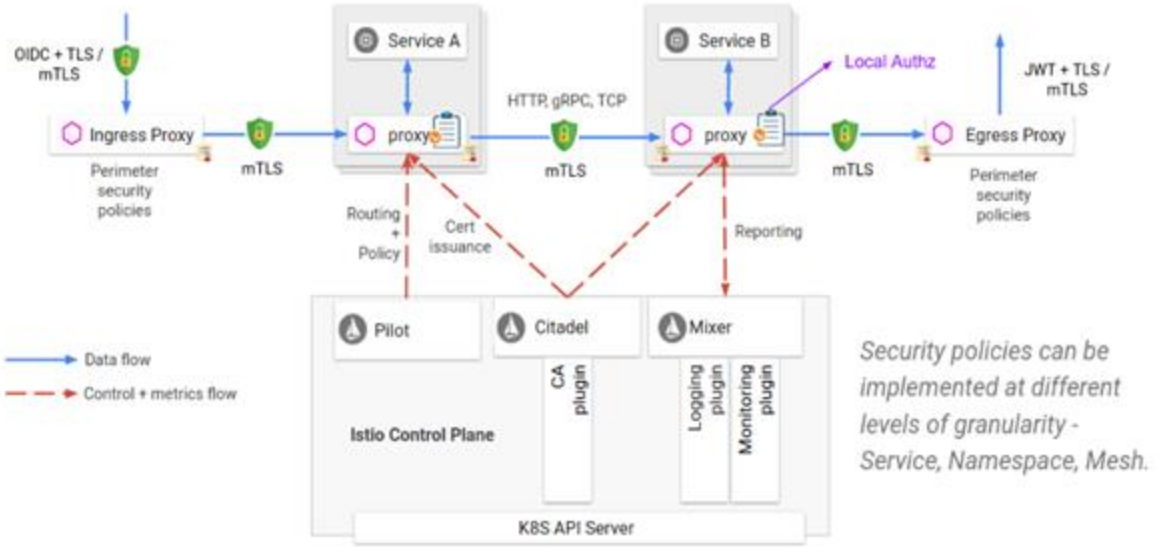
Predicate介绍：

- 通过时间匹配
- 通过 Cookie 匹配
- 通过 Header 属性匹配
- 通过 Host 匹配
- 通过请求方式匹配
- 通过请求路径匹配
- 通过请求参数匹配
- 通过请求 ip 地址进行匹配
- 组合使用

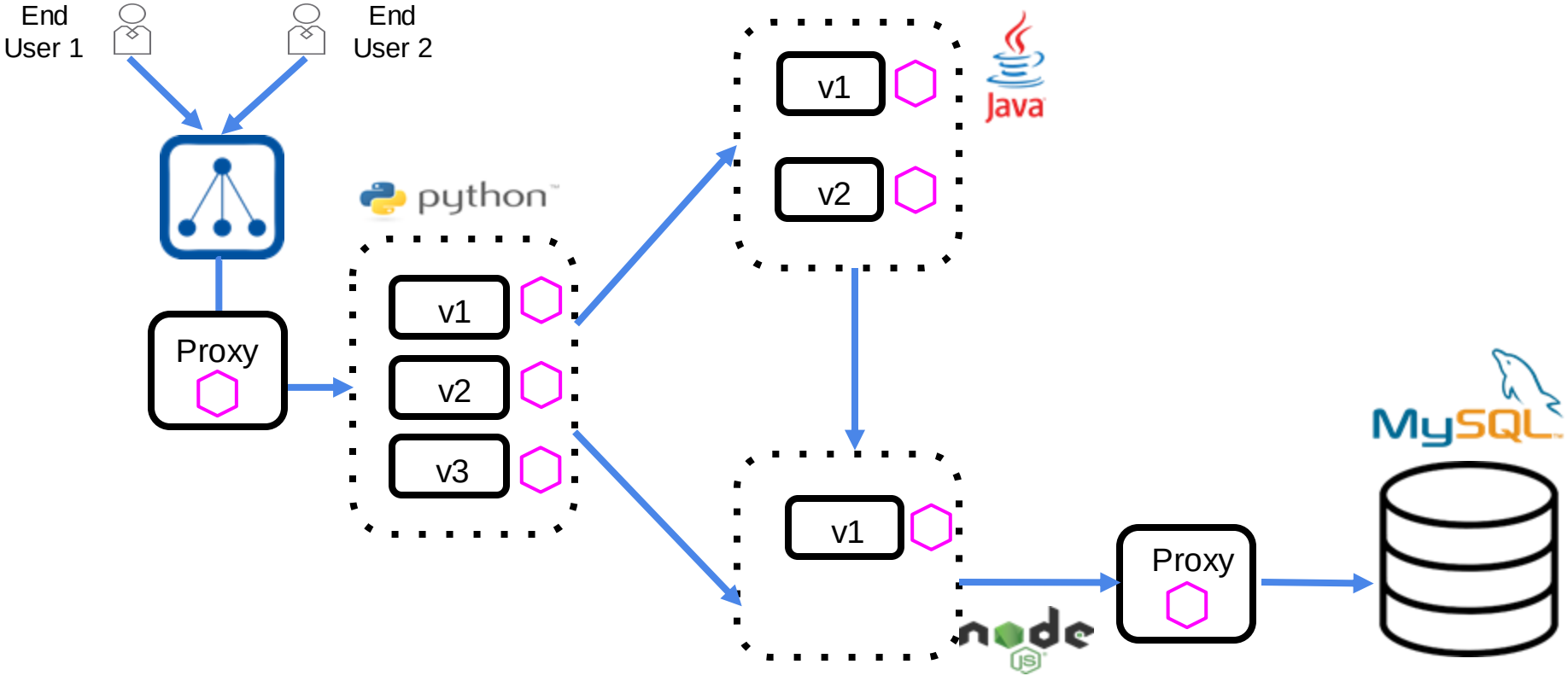
Istio is an open framework for connecting, securing, managing and monitoring services

Istio基本架构

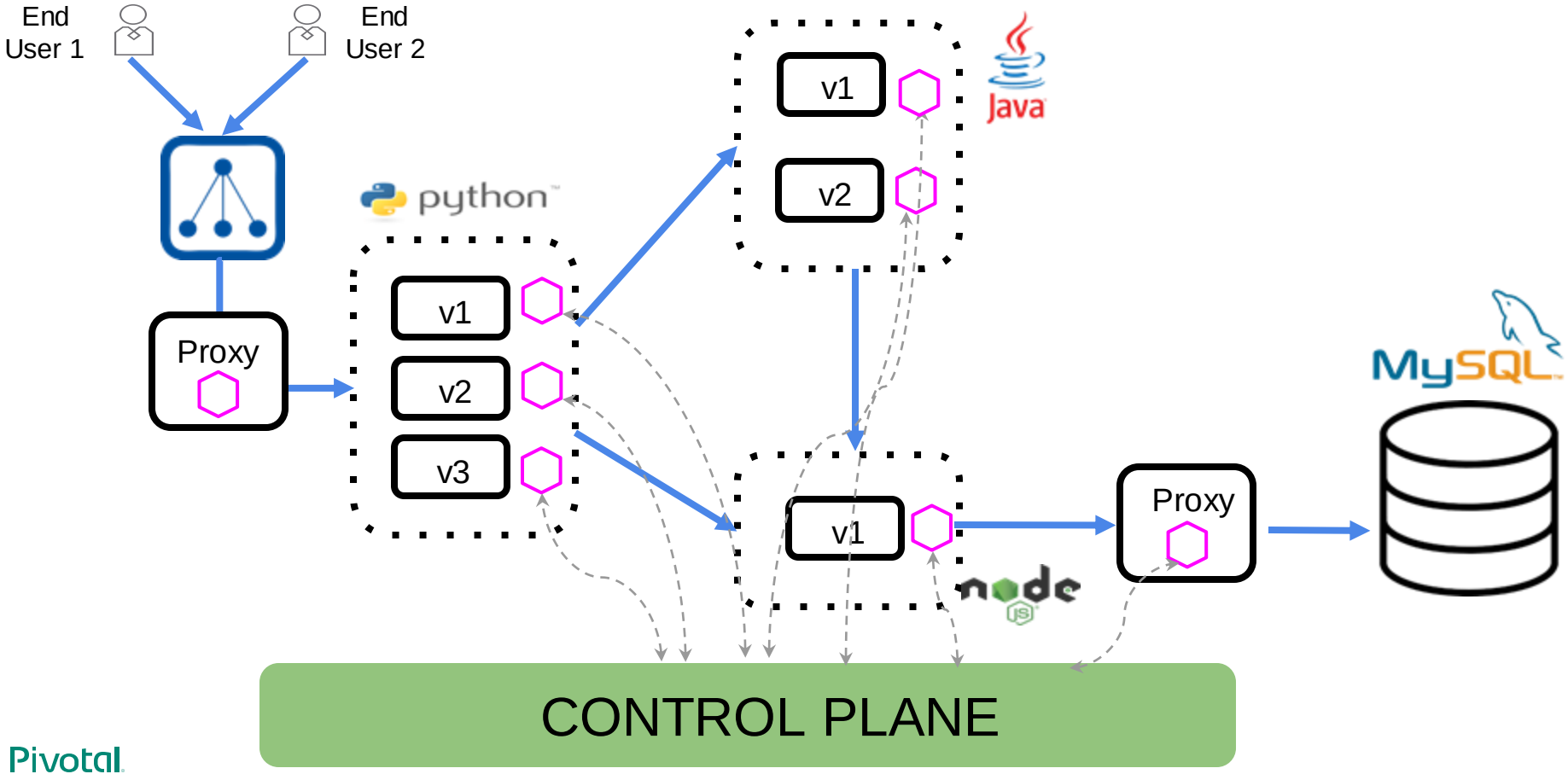
- **Pilot**
  - 控制层面配置和推送安全策略
- **Citadel**
  - 采用mTLS进行服务间认证和鉴权
  - 内置用户凭据管理
- **Mixer**
  - 插件模式支持外部策略提供者进行扩展



Sidecars 提供了一种通用的框架



Service Mesh: Sidecars + Control Plane



如何应对微服务运行时挑战?

- 微服务治理
  - API Gateway
  - Spring Cloud家族
- 分布式事务
- 微服务生命周期管理
  - CI/CD
  - 云平台

Spring Cloud家族

微服务支持

Spring Cloud Netflix

Spring Cloud Netflix provides Netflix OSS integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms. With a few simple annotations you can quickly enable and configure the common patterns inside your application and build large distributed systems with battle-tested Netflix components. The patterns provided include Service Discovery (Eureka), Circuit Breaker (Hystrix), Intelligent Routing (Zuul) and Client Side Load Balancing (Ribbon)..

- Eureka 服务注册、发现
- Hystrix + Turbine 断路器
- Ribbon 负载均衡
- Feign Java HTTP客户端
- Zuul 智能路由

Pivotal

Spring Cloud家族组件

- **微服务治理**
  - **Spring Cloud Netflix** Integration with various Netflix OSS components (Eureka, Hystrix, Zuul, Archaius, etc.).
  - **Spring Cloud Alibaba** With Spring Cloud Alibaba, you only need to add some annotations and a small amount of configurations to connect Spring Cloud applications to the distributed solutions of Alibaba, and build a distributed application system with Alibaba middleware.
  - **Spring Cloud Commons** Common classes used in different Spring Cloud implementations
  - **Spring Cloud Consul** Service discovery and configuration management with Hashicorp Consul.
  - **Spring Cloud Zookeeper** Service discovery and configuration management with Apache Zookeeper.
  - **Spring Cloud Config** provides server-side and client-side support for externalized configuration in a distributed system.
  - **Spring Cloud Bus** An event bus for linking services and service instances together with distributed messaging. Useful for propagating state changes across a cluster (e.g. config change events).
  - **Spring Cloud Cluster** Leadership election and common stateful patterns with an abstraction and implementation for Zookeeper, Redis, Hazelcast, Consul.
  - **Spring Cloud Sleuth** Distributed tracing for Spring Cloud applications, compatible with Zipkin, HTrace and log-based (e.g. ELK) tracing.
  - **Spring Cloud Security** Provides support for load-balanced OAuth2 rest client and authentication header relays in a Zuul proxy.
  - **Spring Cloud Contract** support for Consumer-driven Contracts and service schemas in Spring applications, covering a range of options for writing tests, publishing them as assets, and asserting that a contract is kept by producers and consumers — for both HTTP and message-based interactions.
- **Spring Cloud和不同云平台对接的Connector**
  - **Spring Cloud for Cloud Foundry** Integrates your application with Pivotal Cloudfoundry. Provides a service discovery implementation and also makes it easy to implement SSO and OAuth2 protected resources, and also to create a Cloudfoundry service broker.
  - **Spring Cloud Kubernetes** Spring Cloud common interface implementations that consume Kubernetes native services.
  - **Spring Cloud for Amazon Web Services** Easy integration with hosted Amazon Web Services. It offers a convenient way to interact with AWS provided services using well-known Spring idioms and APIs, such as the messaging or caching API. Developers can build their application around the hosted services without having to care about infrastructure or maintenance.
  - **Spring Cloud Connectors** Makes it easy for PaaS applications in a variety of platforms to connect to backend services like databases and message brokers (the project formerly known as "Spring Cloud").
- **微服务数据流及编排**
  - **Spring Cloud Data Flow** A cloud native programming and operating model for composable data microservices on a structured platform.
  - **Spring Cloud Stream** Messaging microservices with Redis, Rabbit or Kafka. Simple declarative model to send and receive messages in a Spring Cloud app.
  - **Spring Cloud Stream Modules** Spring Cloud Stream Modules can be used with Spring Cloud Stream to create, build, and scale message-driven data microservices.
- **Starter和工具**
  - **Spring Cloud Starters** Spring Boot-style starter projects to ease dependency management for consumers of Spring Cloud. (Discontinued as a project and merged with the other projects after Angel.SR2.)
  - **Spring Cloud CLI** Spring Boot CLI plugin for creating Spring Cloud component applications quickly in Groovy

Pivotal

Spring Cloud Kubernetes

- Configmap -> Config Server
- K8S Service -> Eureka
- Ingress -> Zuul
- Ribbon -> Spring Cloud Kubernetes Ribbon

如何应对微服务的主要挑战

- 微服务治理
  - API Gateway
  - Spring Cloud家族
- 分布式事务
- 微服务生命周期管理
  - CI/CD
  - 云平台

分布式事务的挑战

分布式事务的挑战

ACID

?

1. 单体应用中，通过数据库的回滚保证事务的一致性
2. 没有Foreign Key
3. 但分布式事务很脆弱
4. 加上分布式系统的复杂性
5. 当微服务规模变得很大时，很难解决复杂的分布式事务问题

分布式事务的挑战

BASE

- 基本可用（**Basically Available**）

  是指分布式系统在出现故障的时候，允许损失部分可用性，即保证核心可用。
- 软状态（**Soft State**）

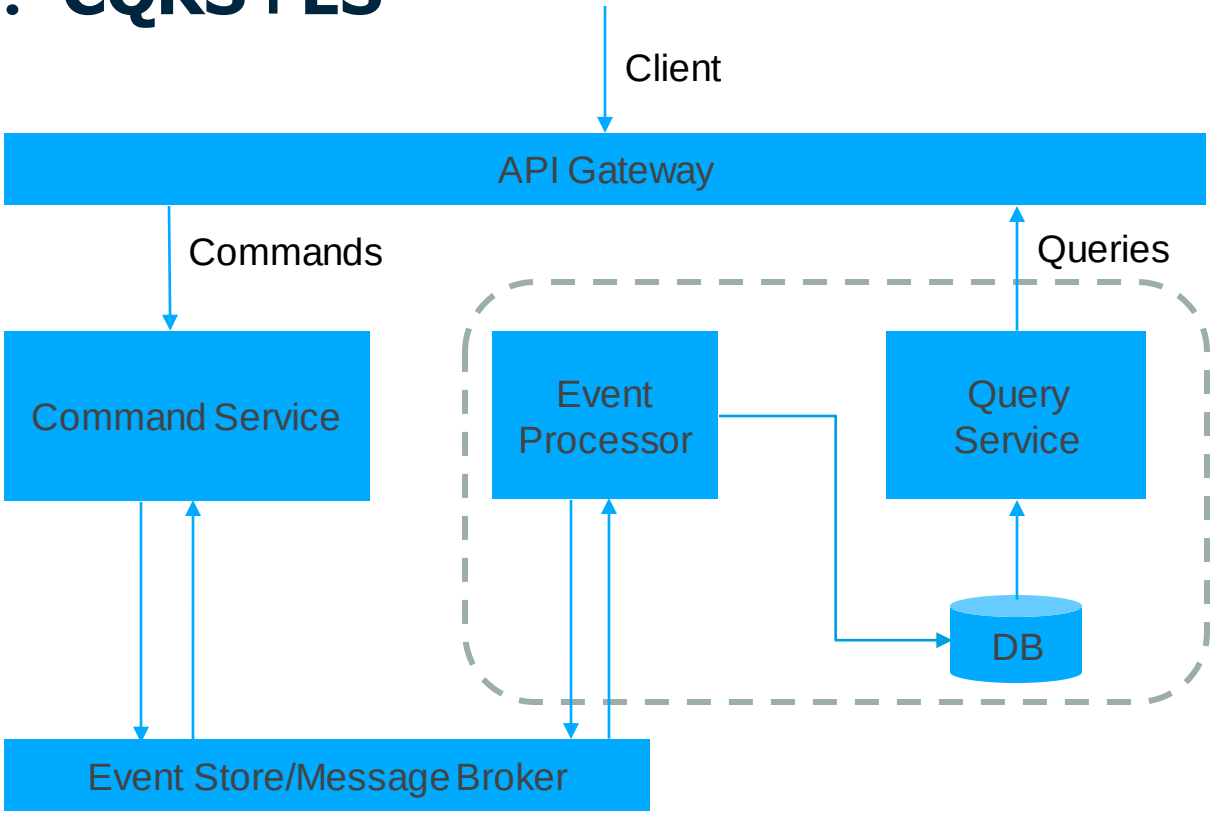
  是指允许系统存在中间状态，而该中间状态不会影响系统整体可用性。分布式存储中一般一份数据至少会有三个副本，允许不同节点间副本同步的延时就是软状态的体现。
- 最终一致性（**Eventual Consistency**）

  是指系统中的所有数据副本经过一定时间后，最终能够达到一致的状态。弱一致性和强一致性相反，最终一致性是弱一致性的一种特殊情况。

使用Event-Driven架构来实现分布式系统

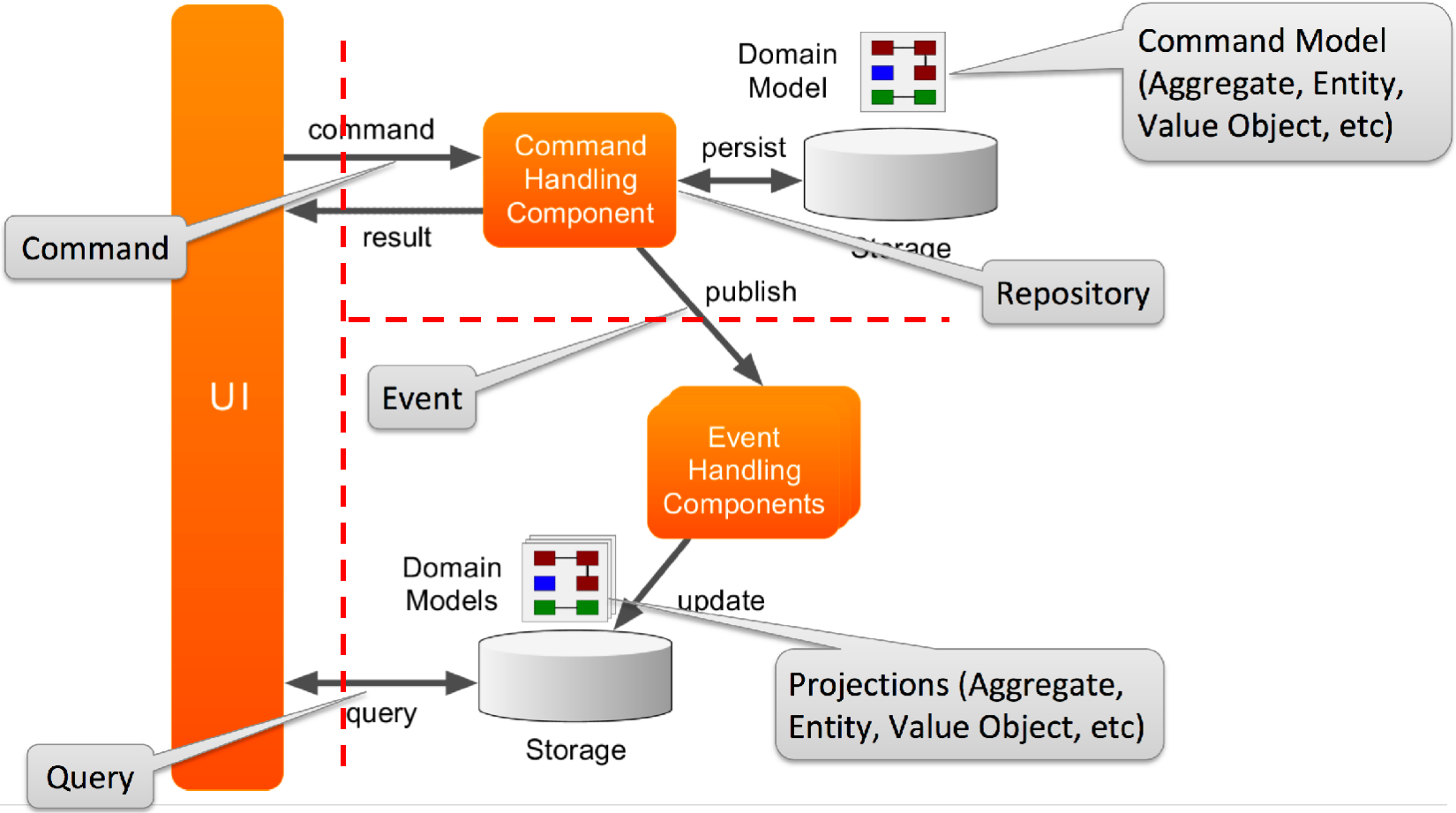
Event-Driven Microservices

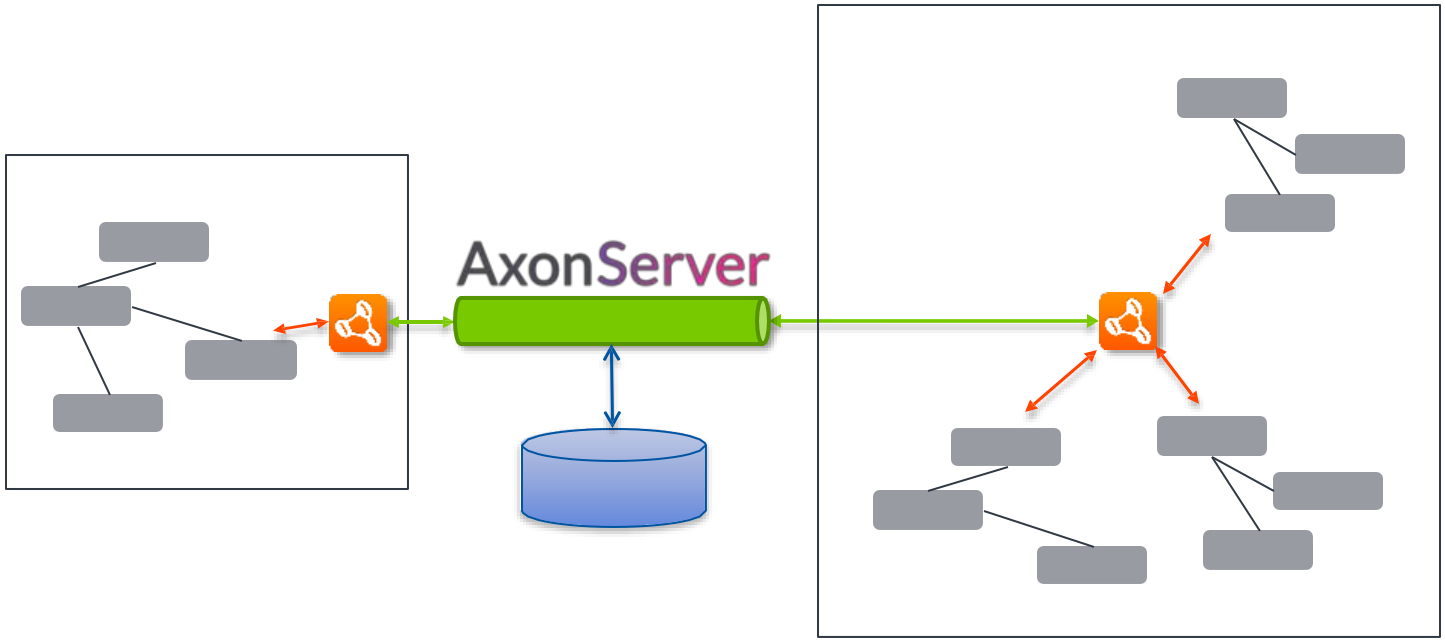
最佳实践：CQRS+ES

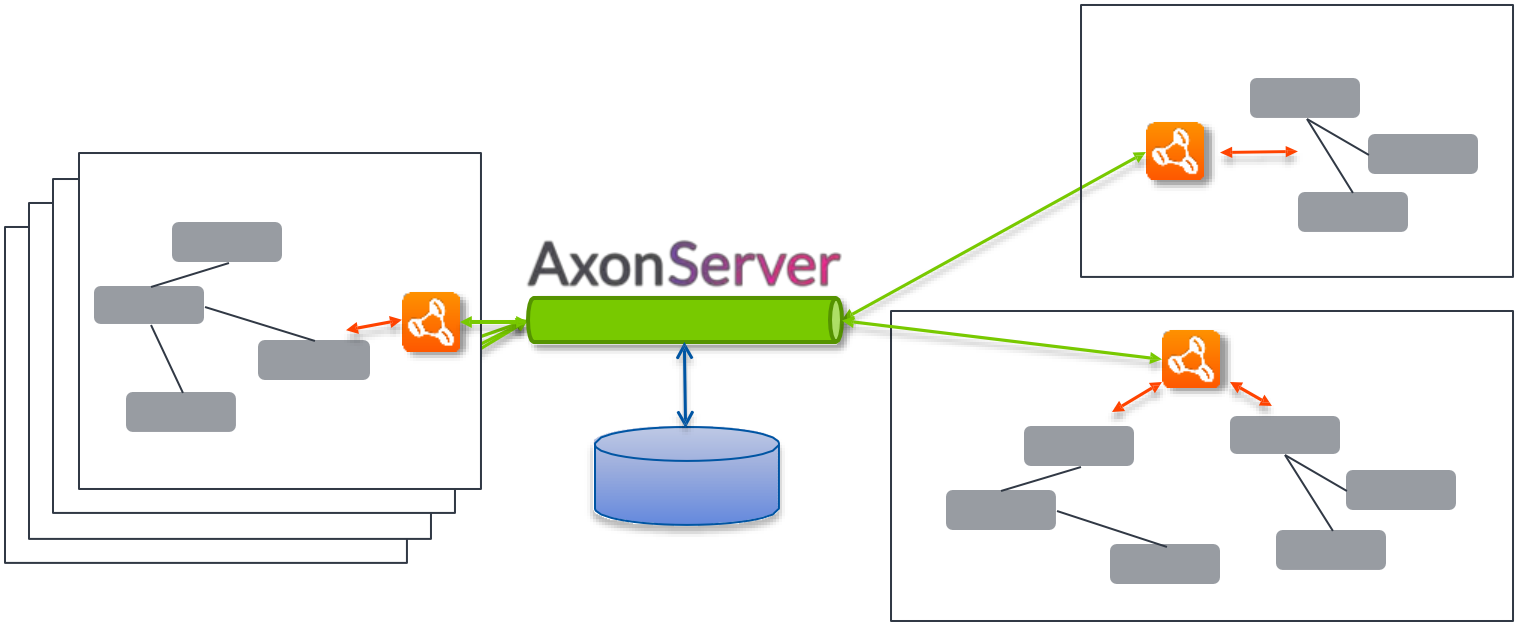

Client
API Gateway
Commands
Queries
Command Service
Event Processor
Query Service
DB
Event Store/Message Broker

推荐框架

- Axon(axoniq.io)
- Eventuate.io

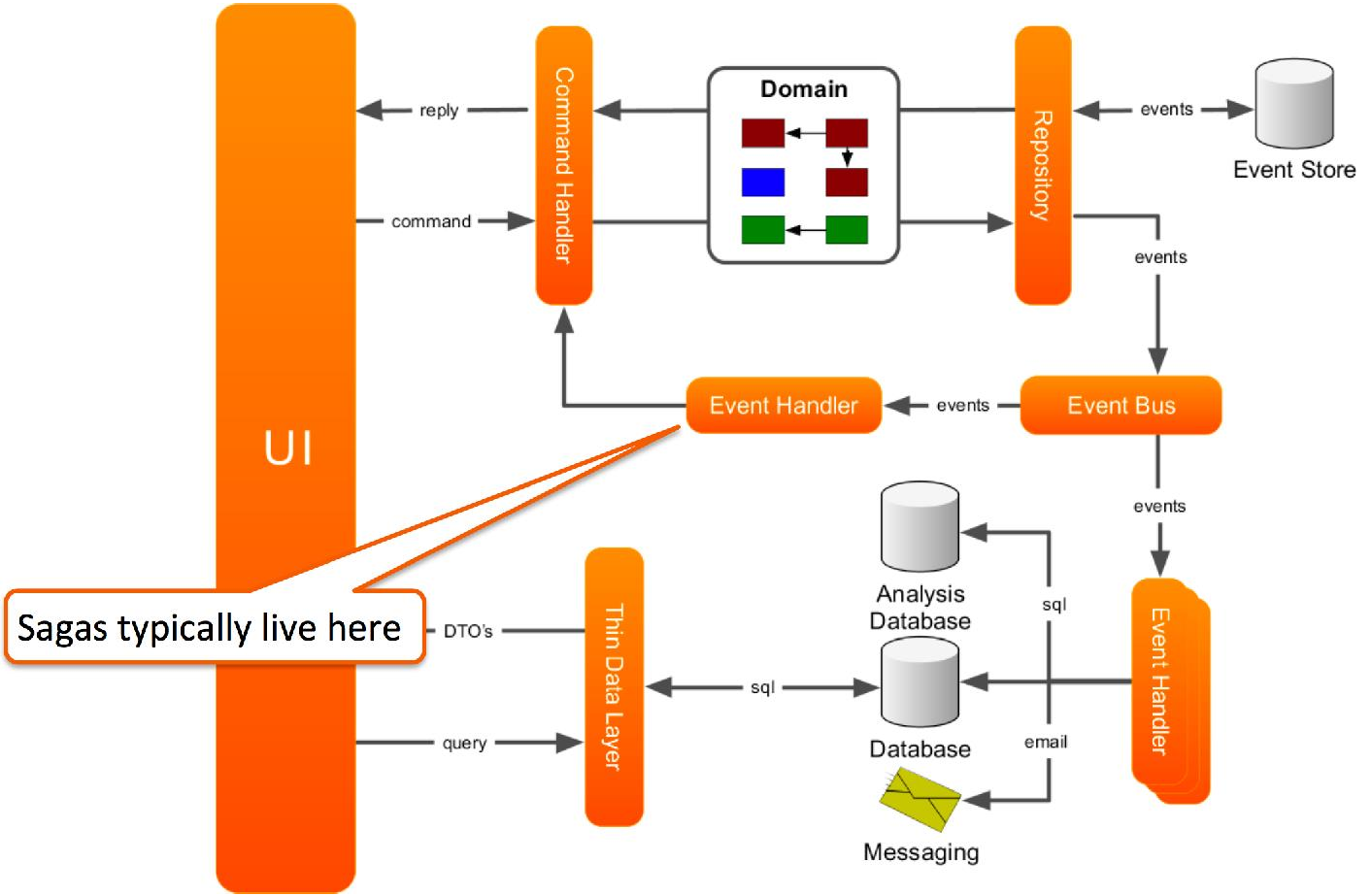

Domain
Model
Command Model
(Aggregate, Entity,
Value Object, etc)
command
Command
Handling
Component
persist
Command
result
Repository
publish
UI
Event
Event
Handling
Components
Domain
Models
update
Projections (Aggregate,
Entity, Value Object, etc)
query
Storage
Query


AxonServer


AxonServer

Saga

- 用来管理BASE事务
- 基于事件做出相应
- 对以下的活动进行协调
  - Bounded contexts
  - Aggregates

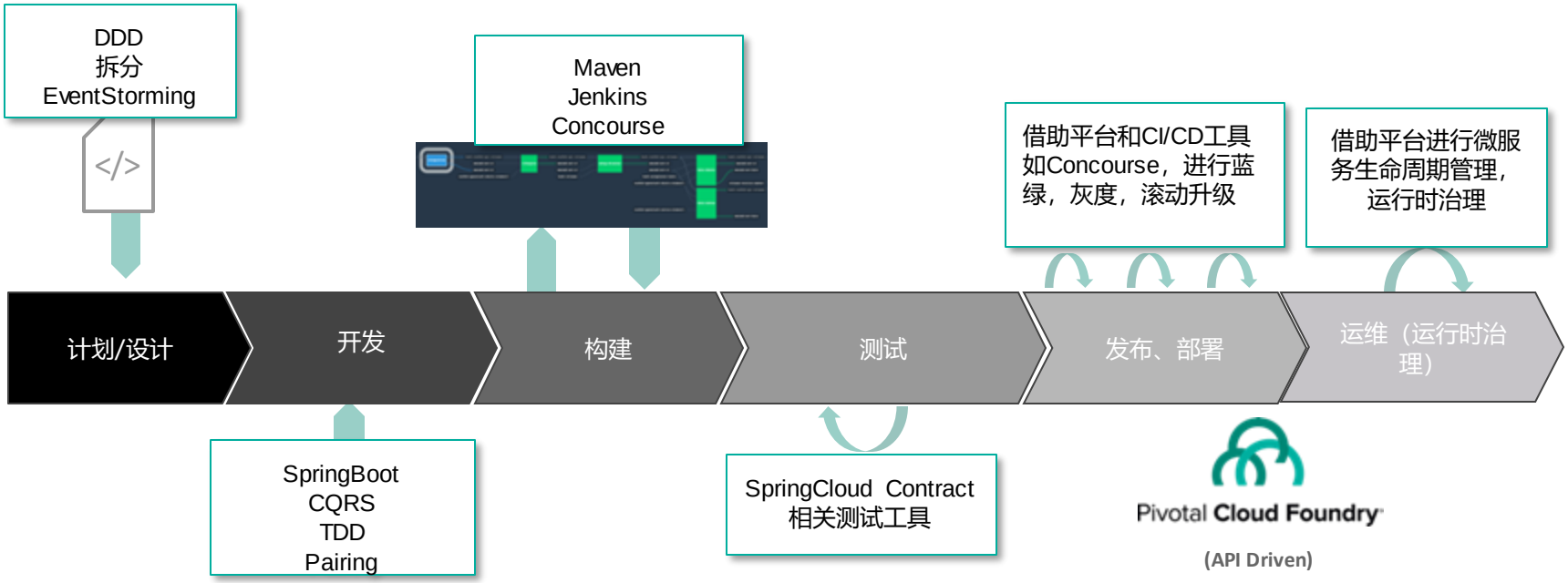

UI
reply
command
Command Handler
Domain
Repository
events
Event Store
events
Event Handler
events
Event Bus
events
Sagas typically live here
DTO's
query
Thin Data Layer
sql
Analysis Database
Database
sql
email
Event Handler
Messaging

如何应对微服务运行时挑战?

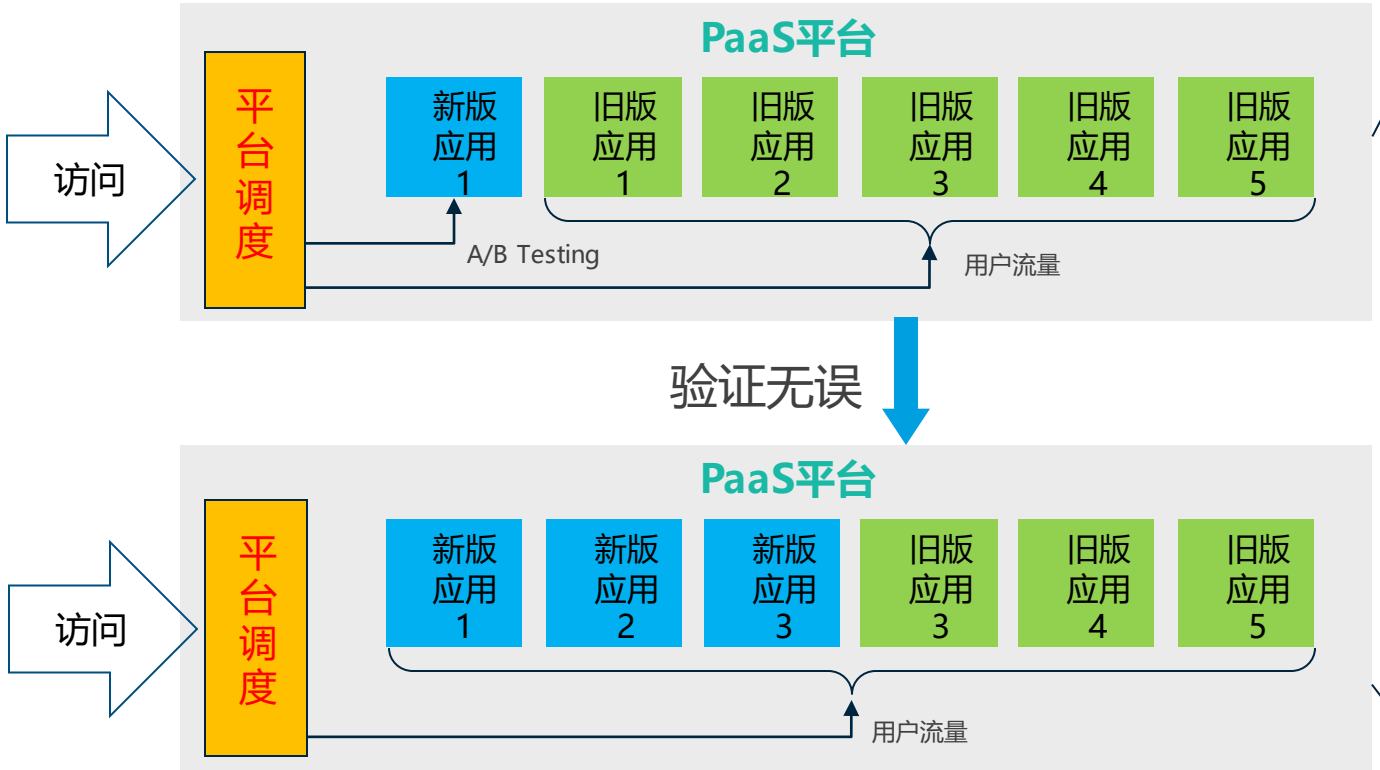
- 微服务治理
  - API Gateway
  - Spring Cloud家族
- 分布式事务
- 微服务生命周期管理
  - 持续集成/持续交付
  - 云平台

微服务生命周期管理

微服务应用整个生命周期



PaaS平台应用场景—应用更新发布

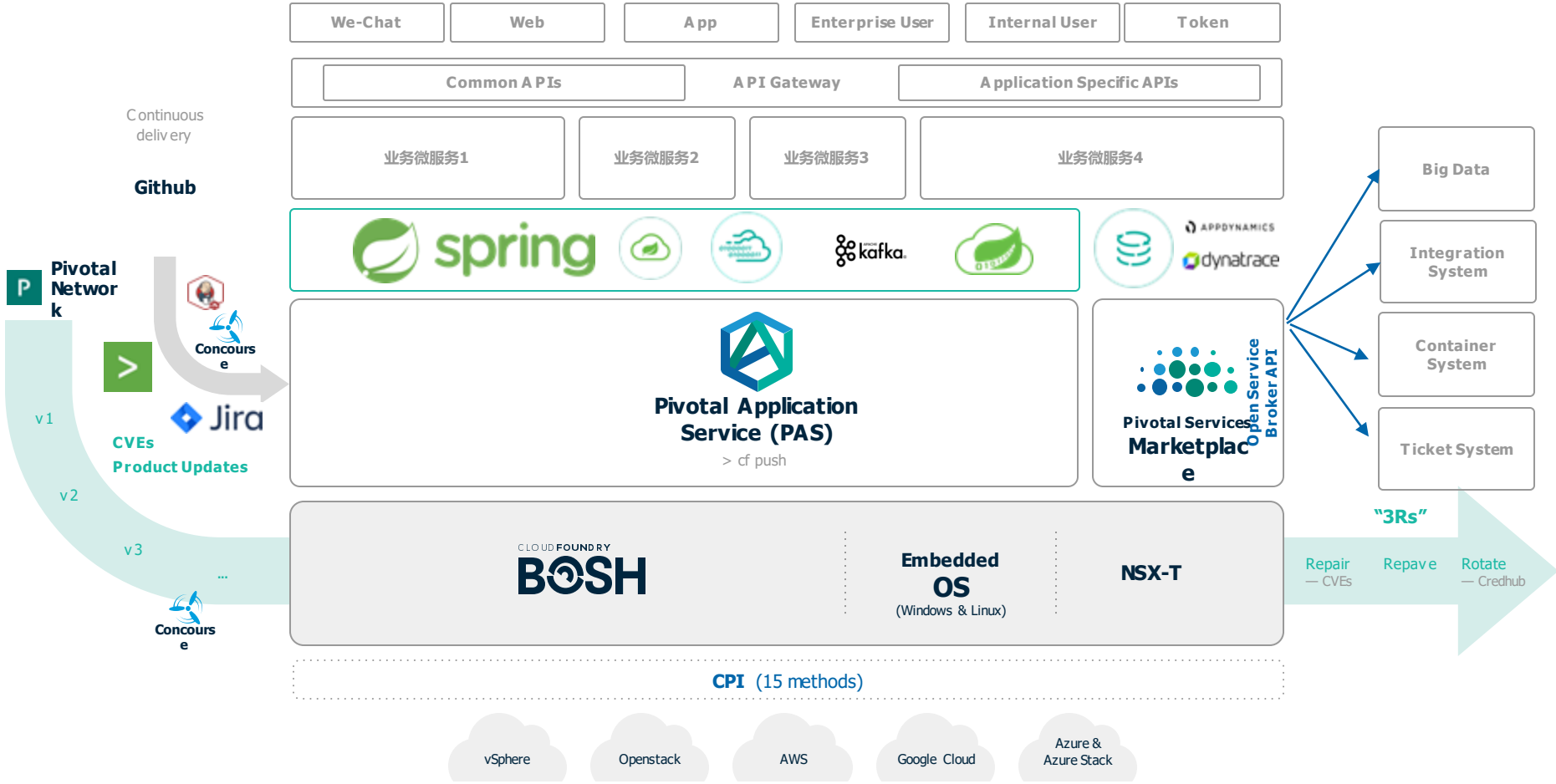


第一步
灰度发布/
金丝雀发布

- 发布策略内置，由PaaS平台自动执行，除非有需要人工确认或者干预的管理类流程，其余情况均不需人工参与；
- PaaS平台调度进行访问流量分配，保证用户无感知；
- 旧应用实例优雅停机、销毁，新应用实例均全新启动；
- 整个过程应用实例总量不变

第二步
滚动发布

一种基于PCF的微服务全生命周期管理的治理的实现



Pivotal

Transforming How The World Builds Software