

Service Mesh 高可用在企业级生产中的实践

罗广明

百度高级研发工程师

- **罗广明、百度高级工程师**
- ServiceMesher 社区 (servicemesher.com) 治理委员会核心成员
- 云原生社区 (cloudnative.to) 联合创始成员
- 百度云智学院认证讲师
- 目前在「百度云云原生团队」负责微服务治理与相关中间件研发
- 对云原生架构与技术、研发流程、团队文化有深入研究，对 Spring Cloud、Service Mesh 等微服务治理框架有丰富实践经验

/01 Service Mesh 与
Spring Cloud 应
用的互通、共治

/02 注册中心与
高可用方案

/03 通过治理策略
保证服务高可用

/01

Service Mesh 与 Spring Cloud 应用的互通、共治

Spring Cloud 的优缺点

优点

- 微服务架构的集大成者
- 轻量级组件
- 开发灵活、简便
- 社区生态强大、活跃度高

缺点

- 仅适用于 JAVA 应用、Spring Boot 框架
- 侵入性强
- 升级成本高、版本碎片化严重
- 内容多、门槛高
- 治理功能仍然不全

Service Mesh 的优缺点

优点

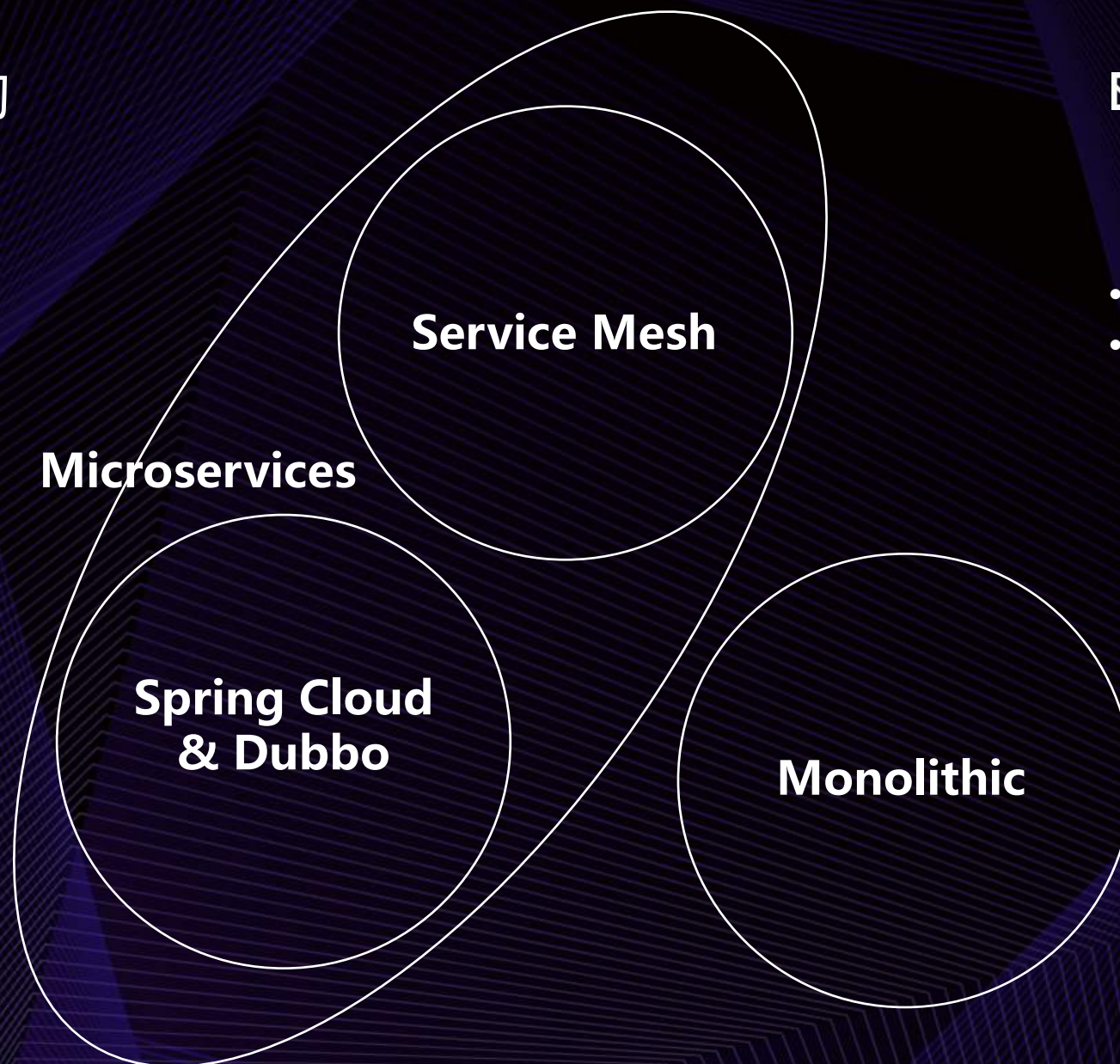
- 微服务治理与业务逻辑解耦
- 异构系统的统一治理
- 三大技术优势：
 - 可观察性
 - 流量控制
 - 安全

缺点

- 增加了复杂度
 - 整体链路的复杂度
 - 操作运维的复杂度
- 需要更专业的运维技能
- 带来延迟
- 平台的适配

Istio-Handbook : Service Mesh 概述

混合微服务架构



- 微服务 & 单体
- Spring Cloud & Service Mesh

运行时支撑服务

- 服务注册中心
- 服务网关
- 配置中心

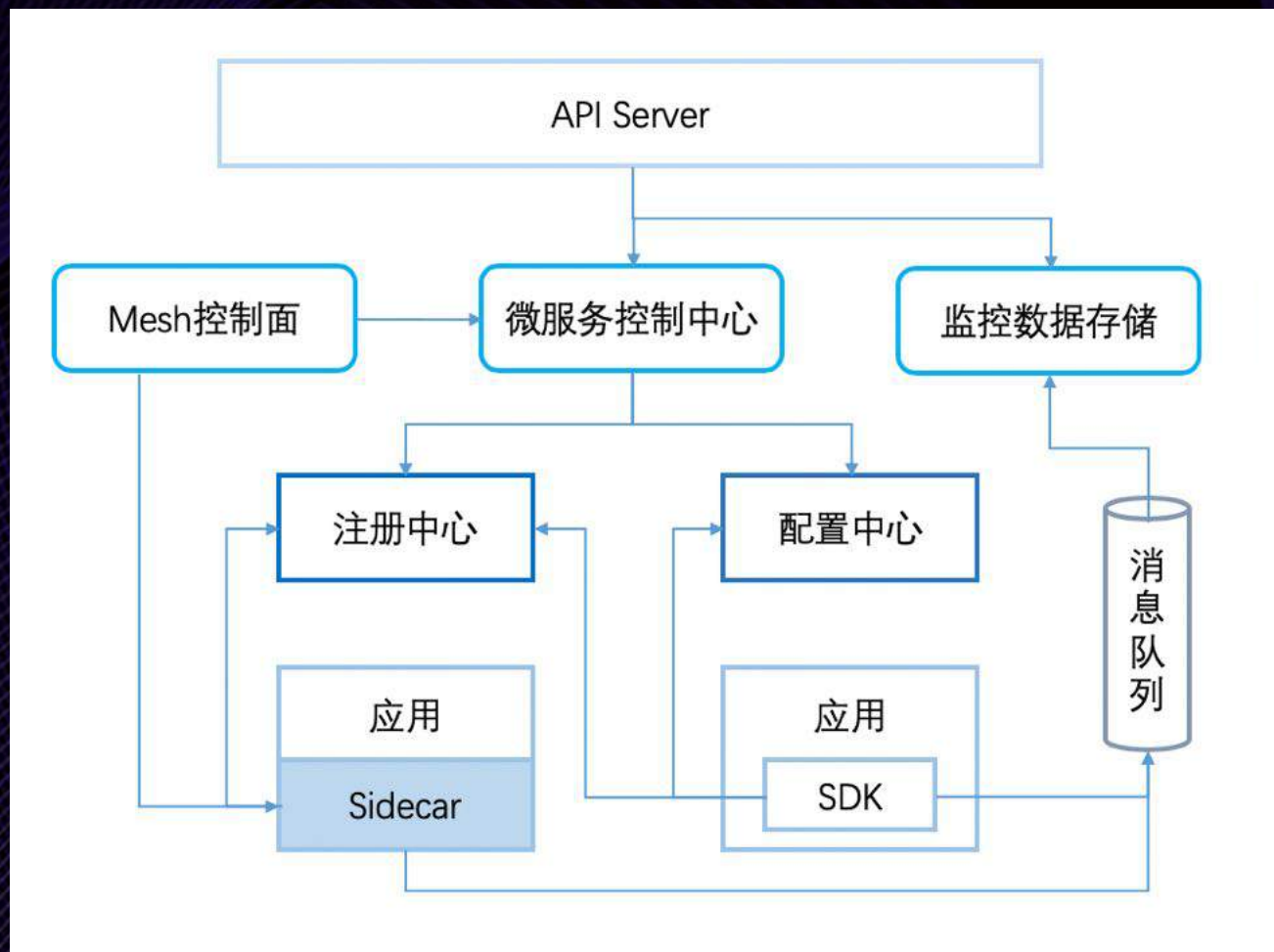
目标

- 互联互通
- 平滑迁移
- 灵活演进

环境

- 虚拟机
- Kubernetes

混合微服务的互联互通



- Spring Cloud
- Service Mesh

百度智能云 CNAP 混合微服务架构图

/02

注册中心与高可用方案

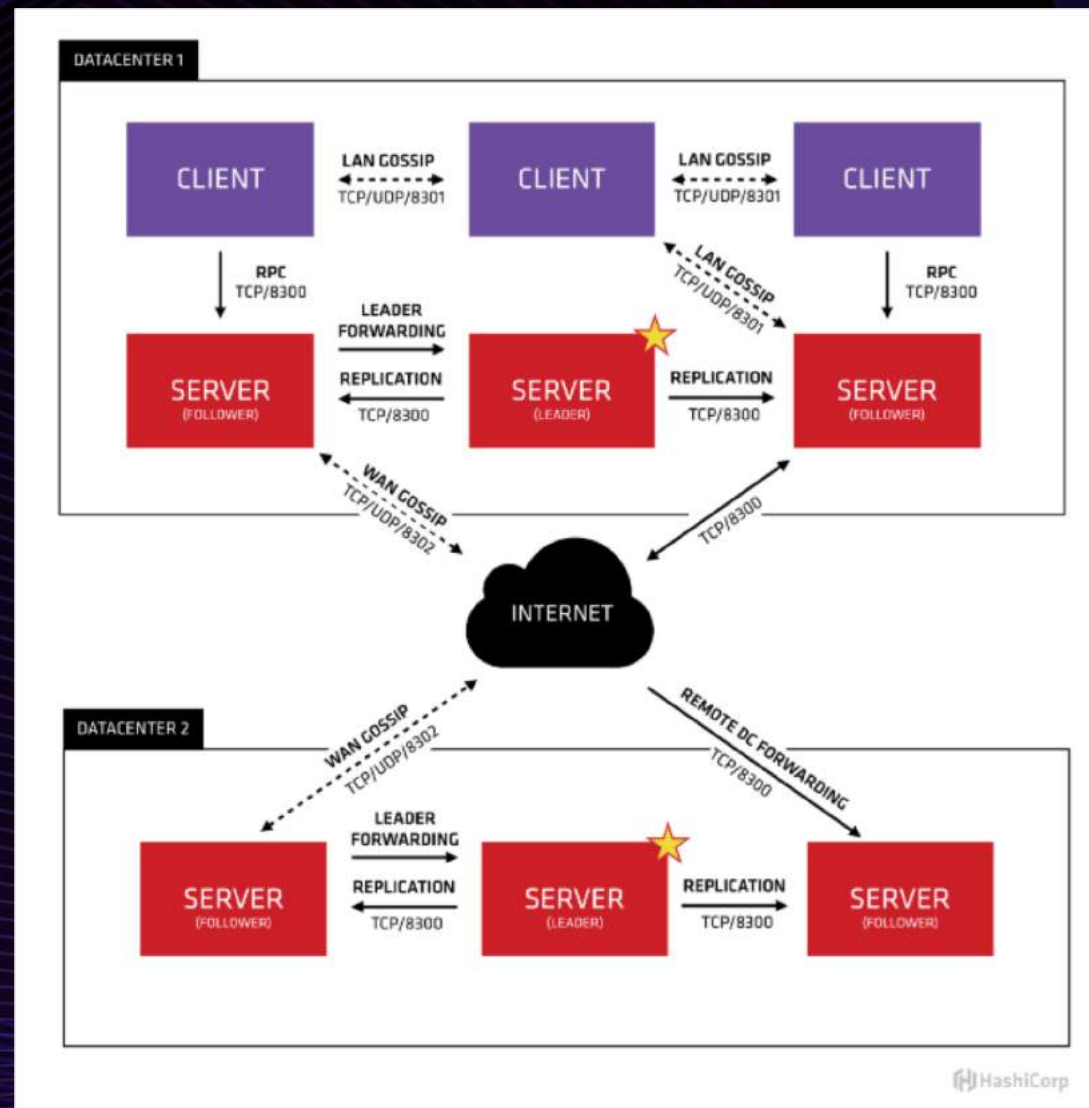
注册中心 - Consul

- Consul is a tool for service discovery and configuration. Consul is distributed, highly available, and extremely scalable.
- Consul provides several key features:
 - Service Discovery
 - Health Checking
 - Service Segmentation/Service Mesh
 - Key/Value Storage
 - Multi-Datacenter



注册中心 - Consul

- Consul 架构



- 注册中心容灾

服务端容灾

- 部分节点宕机
- 网络分区

客户端容灾

- 注册中心完全不可用

注册中心 - Consul

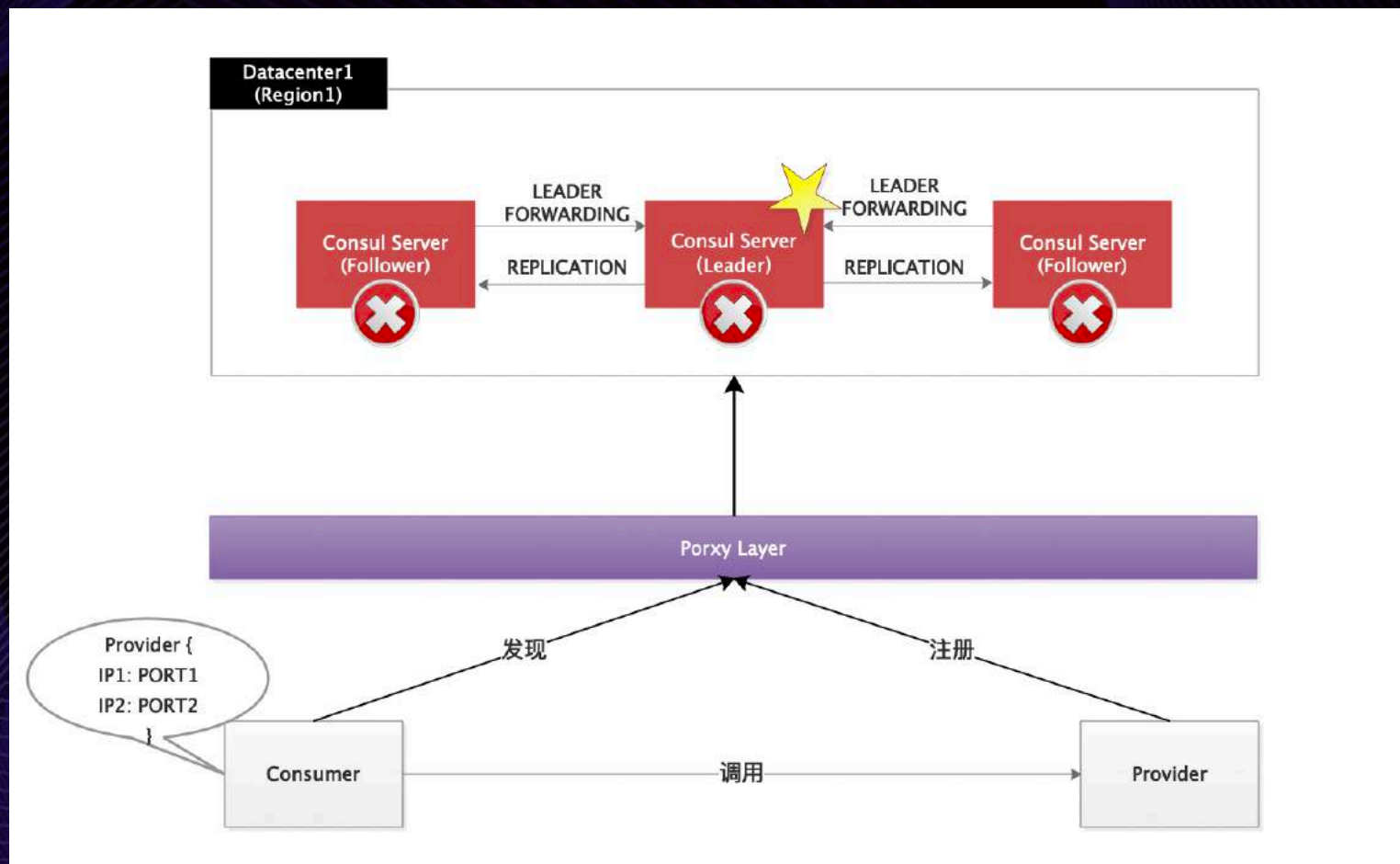
- 服务端容灾与节点数

Servers	Quorum Size	Failure Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

注册中心 - Consul

- 客户端容灾

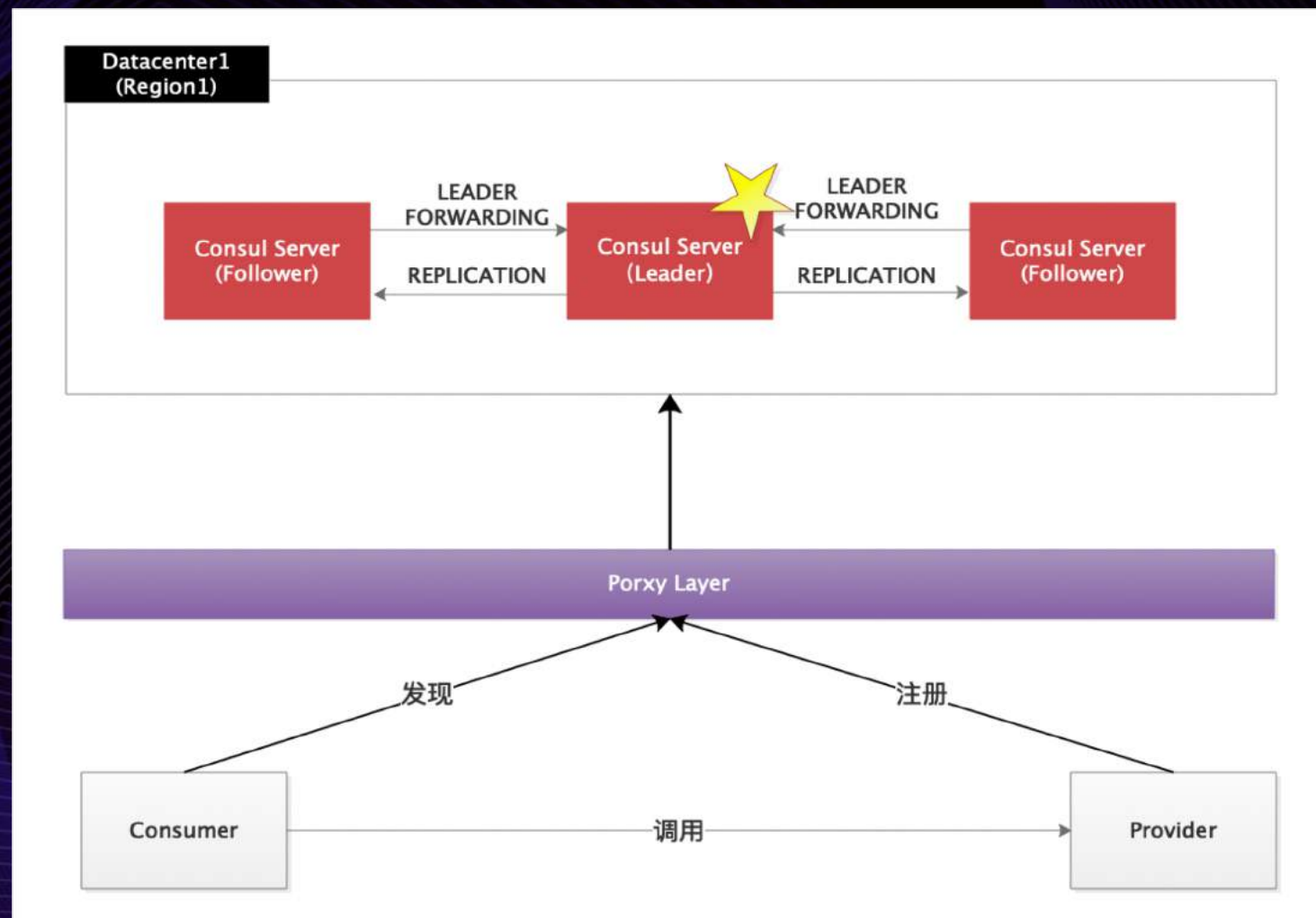
客户端缓存 !



注册中心 - Consul

- 架构设计

- 多地域？
- 多租户？



/03

通过治理策略保证服务高可用

治理策略 & 高可用

描述	N个9	可用性级别	年度停机时间
基本可用	2个9	99%	87.6小时
较高可用	3个9	99.9%	8.8小时
具备故障自动恢复能力可用	4个9	99.99%	53分钟
极高可用	5个9	99.999%	5分钟

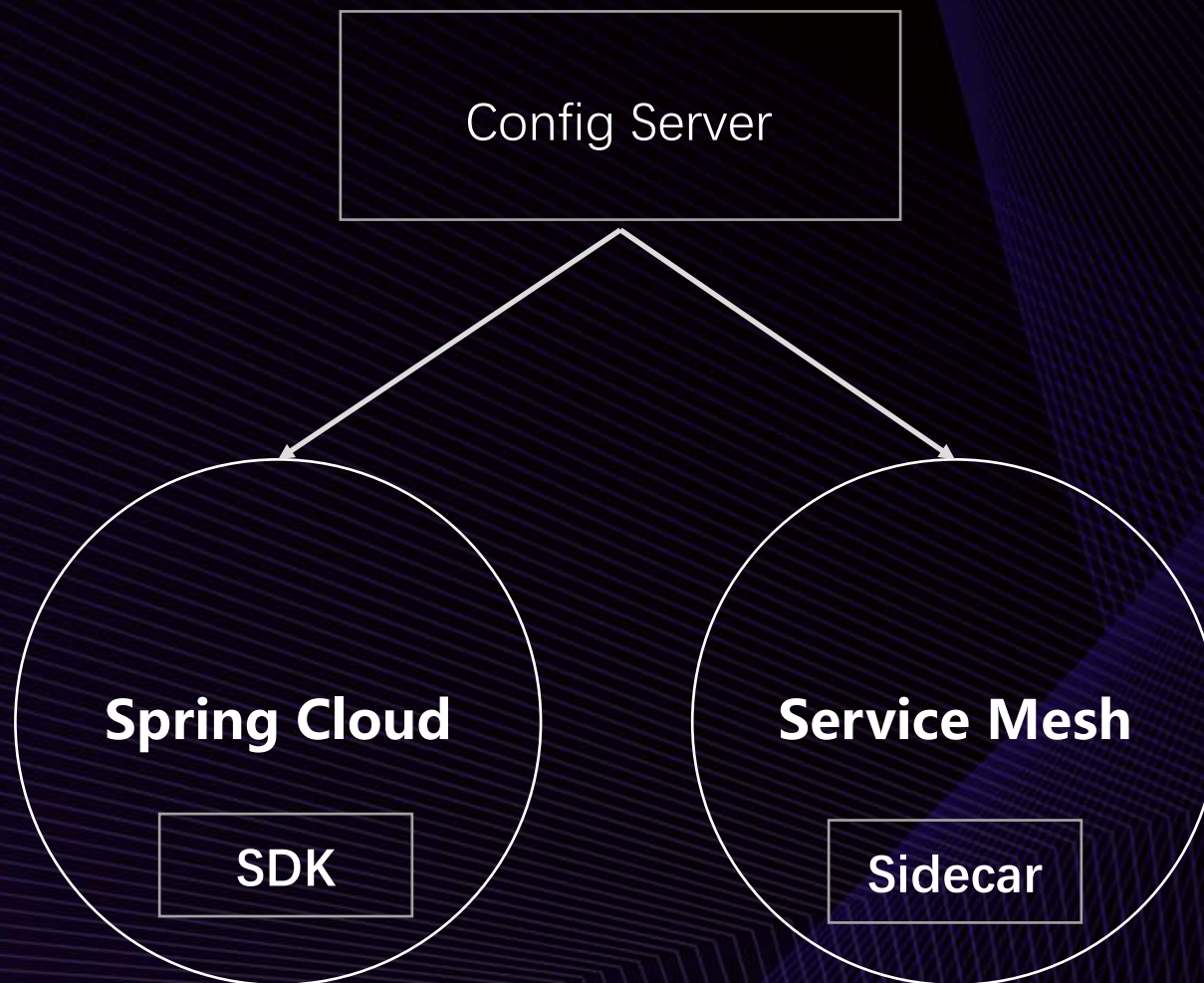


- 微服务高可用设计手段



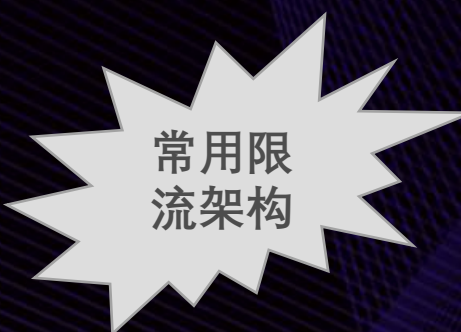
治理策略 & 高可用

- 微服务高可用设计手段
 - 限流
 - 熔断
 - 负载均衡+实例容错



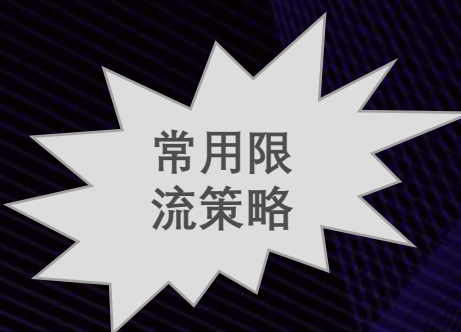
- 微服务高可用设计手段 - 服务限流
 - 对于一个应用系统来说一定会有极限并发/请求数，即总有一个TPS/QPS阈值，如果超过了阈值则系统就会不响应用户请求或响应的非常慢，因此我们最好进行过载保护，防止大量请求涌入击垮系统。
 - 服务限流其实是指当系统资源不够，不足以应对大量请求，即系统资源与访问量出现矛盾的时候，我们为了保证有限的资源能够正常服务，因此对系统按照预设的规则进行流量限制或功能限制的一种方法。
 - 限流的目的是通过对并发访问/请求进行限速或者一个时间窗口内的的请求进行限速来保护系统，一旦达到限制速率则可以拒绝服务或进行流量整形。
 - 限流无非就是针对超过预期的流量，通过预先设定的限流规则选择性的对某些请求进行限流“熔断”。

- 微服务高可用设计手段 - 服务限流
 - 接入层限流
 - 调用外部限流服务限流
 - 切面层/代理层限流



常用限流架构

- 微服务高可用设计手段 - 服务限流
 - 拒绝策略
 - 延迟处理
 - 特权处理



常用限流策略

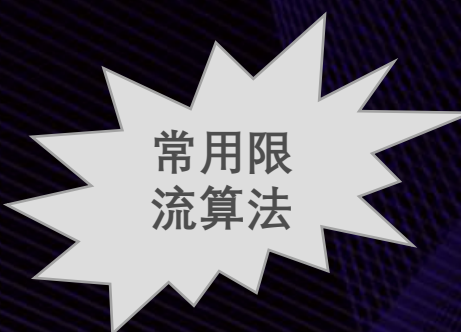
- 微服务高可用设计手段 - 服务限流

- 固定、滑动时间窗口限流

- 适合微服务接口
 - 选定的时间粒度上限流

- 令牌桶、漏桶限流

- 适合阻塞限流
 - 超过最大访问频率后，请求阻塞等待或者直接拒绝（等待时间=0）



常用限流算法

- 微服务高可用设计手段 - 服务限流

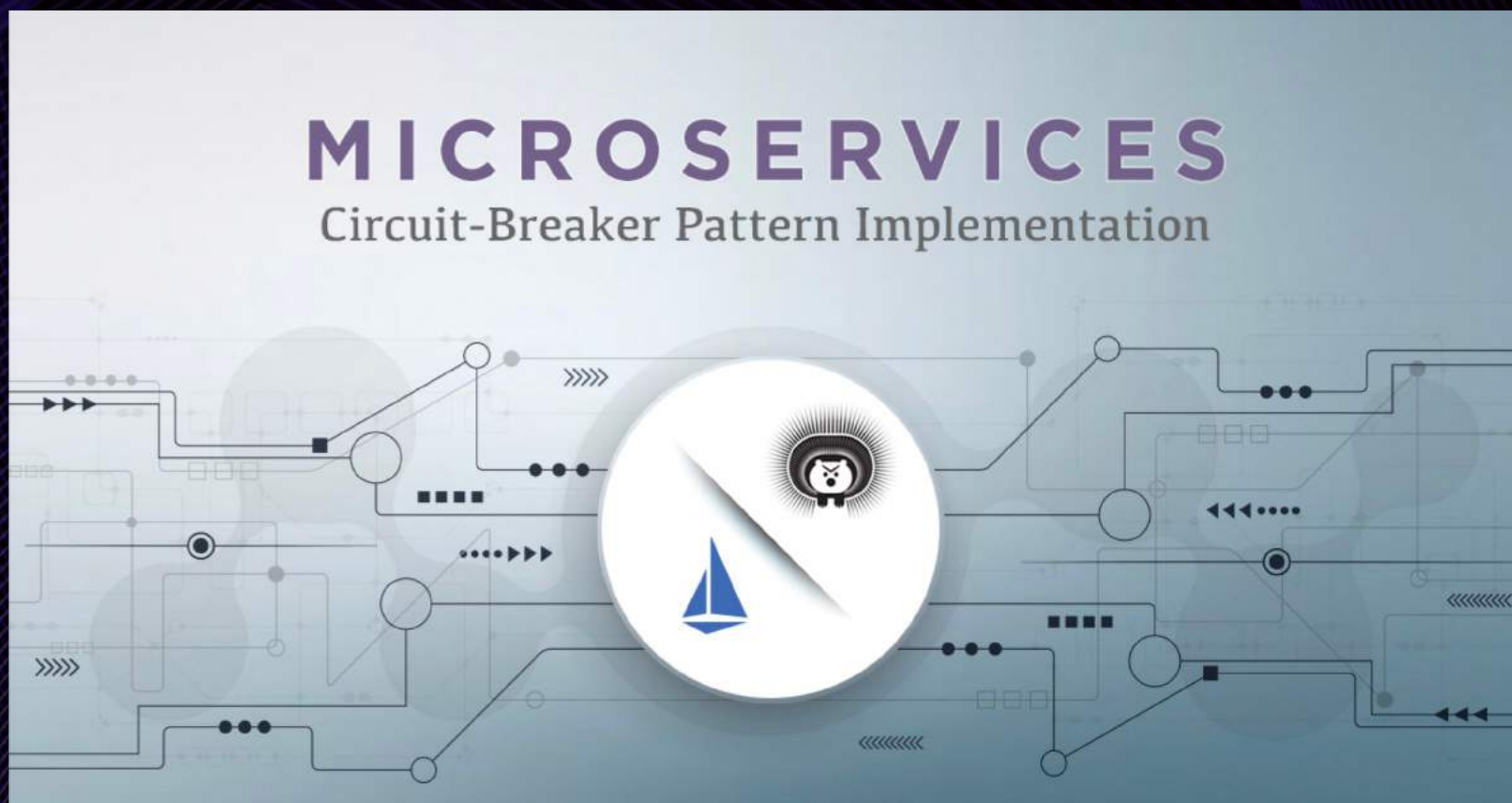
- 基于来源限流 (from)
 - 系统标签
 - 自定义标签
- 基于标签匹配 (to)
 - 系统标签
 - 自定义标签



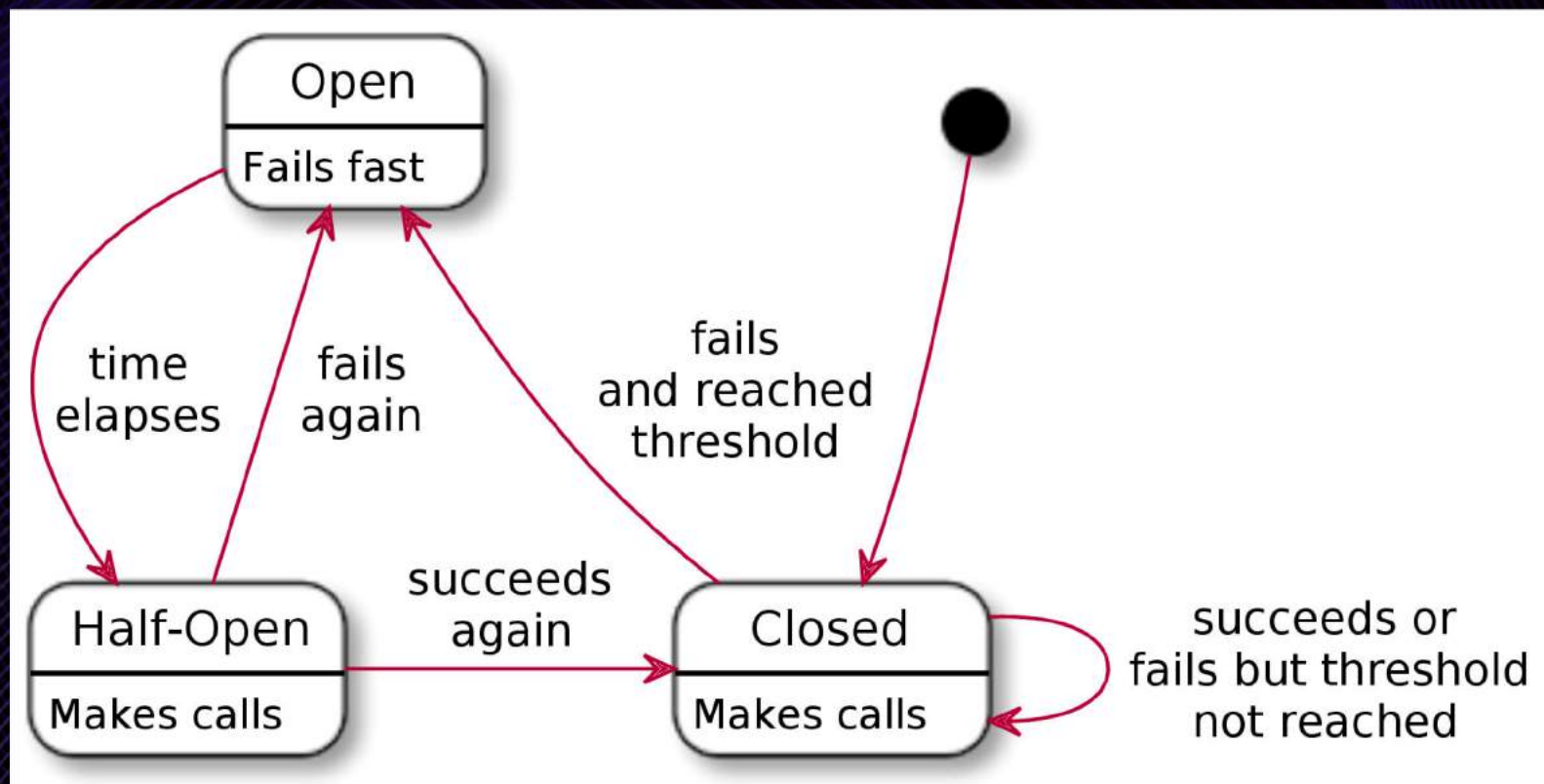
matchLabels

Exclusion
matching

- 微服务高可用设计手段 - 熔断

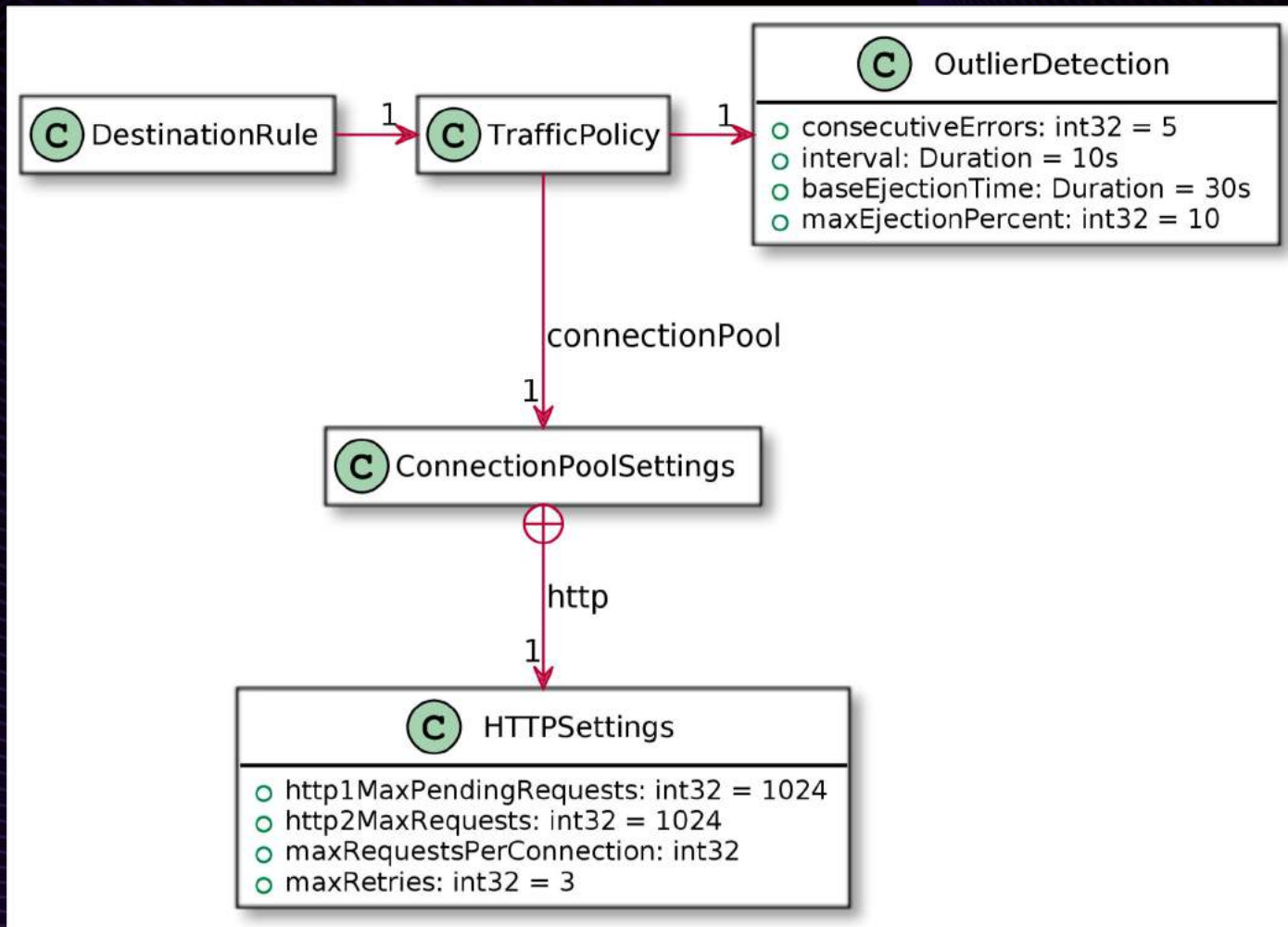


- 微服务高可用设计手段 – 熔断断路器



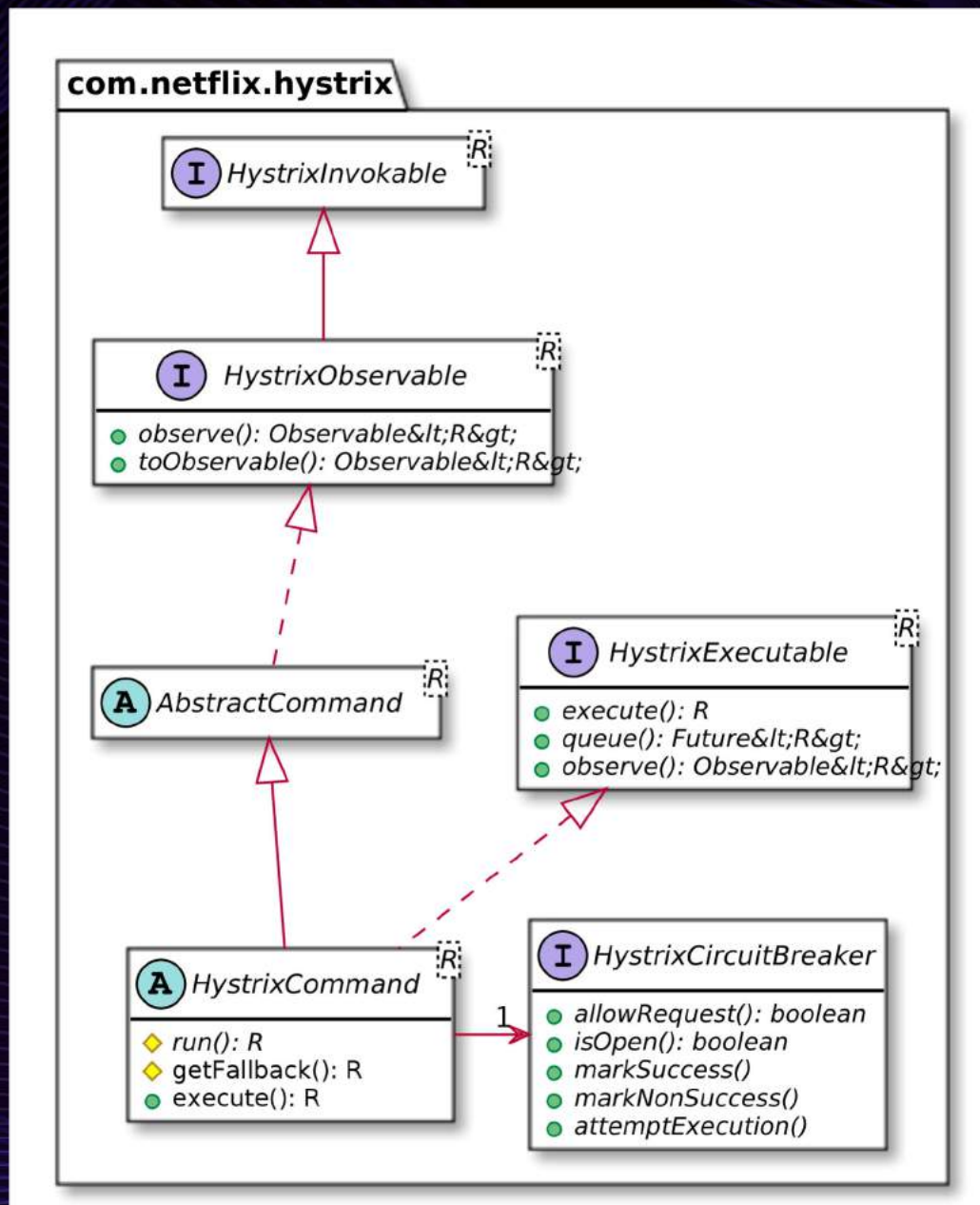
治理策略 & 高可用

- 微服务高可用设计手段
 - Istio 断路器



治理策略 & 高可用

- 微服务高可用设计手段
 - **Hystrix** 断路器



- 微服务高可用设计手段：Istio vs **Hystrix**

	Istio	Hystrix
运作方式	黑盒方式	白盒方式
熔断粒度	实例级别	实例级别+方法级别
半开状态	不支持	支持
最大熔断比例	支持	不支持
fallback	不支持	支持

- 微服务高可用设计手段：Istio vs **Hystrix**

	Istio	Hystrix
运作方式	黑盒方式	白盒方式
熔断粒度	实例级别	实例级别+方法级别
半开状态	不支持	支持
最大熔断比例	支持	不支持
fallback	不支持	支持

- 微服务高可用设计手段 - 熔断
 - 自动熔断 vs 手动熔断
 - Fail-fast vs Fallback



常用熔断用法

治理策略 & 高可用

- 微服务高可用设计手段
 - 负载均衡
 - 随机 (Random)
 - 轮询 (RoundRobin)
 - 响应时间权重 (WeightedResponseTime)
 - 环哈希 (Ring Hash)
 - 实例容错
 - Fail-fast
 - Failover
 - Failresnd

治理策略 & 高可用 – 总结

- 从手段看高可用
- 从架构看高可用
- 从硬件看高可用
- 从软件看高可用
- 从治理看高可用

本质

写在最后 – 开源与社区

- 什么是开源？
 - 事物规划为可以公开访问的，因此人们可以修改并分享。
 - 也泛指一组概念：开源的方式！
- 如何参与开源？
- 开源能带来哪些好处？



- Service Mesh 概述
- Consul作为注册中心在云环境的实践与应用
- 有了这三个锦囊，再也不用担心微服务治理了
- 一文理解微服务高可用的常用手段
- 微服务断路器模式实现：Istio vs Hystrix

感谢聆听



欢迎关注，获取最新分布式架构内容



关注服务网格，关注 ServiceMesher