



基于Service Mesh的服务治理探索和实践

TOP100Summit

全球软件案例研究峰会

时间：11月15~17日

地点：北京国际会议中心

100个年度最值得学习案例

MPD工作坊（深圳站）

时间：9月21~22日

地点：深圳博林圣海伦酒店

20个3小时大时段沙盘课程



100



MPD工作坊（北京站）

时间：7月06~07日

地点：北京国家会议中心

20个3小时大时段沙盘课程

MPD工作坊（上海站）

时间：10月26~27日

地点：上海

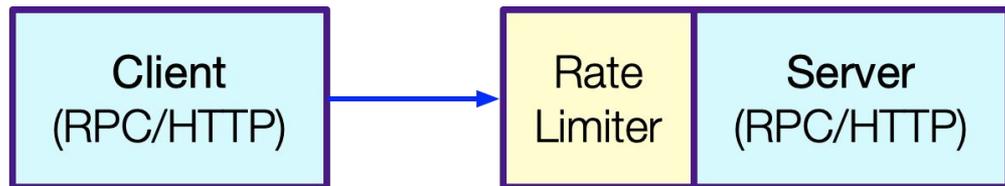
20个3小时大时段沙盘课程

- 概述
 - 从微服务到Service Mesh
- Mesh落地
 - 整体架构
 - 实现方案
 - 关键问题
- 服务治理
 - 访问控制
 - 动态过载保护
 - 染色系统
- 未来展望
 - 规划

概述-案例分享一：服务端限流

语言&协议支持

1. golang、python、C++、java、rust
2. RPC & HTTP
3. python UWSGI多进程模型无法支持



图：服务端限流模型

限流策略

1. 单实例限流
2. 服务整体限流(quota)
3. 丢弃策略：保证高权重请求

问题：从开发到上线需要多久？策略调整了怎么办？

概述-案例分享二：服务安全



为什么需要内部服务安全

数据越来越多，服务信息越来越敏感

用户隐私意识逐渐增强

内网不可信假设

问题: 强制性安全策略? 接入成本?

内网服务安全的需求

身份认证 Authorization

访问授权 Authentication

防窃听加密 Encryption

访问历史审计 Audit

概述-微服务及云环境下服务框架的新挑战



火龙果·整理
uml.org.cn

语言/组件不断增长

基础功能重复建设
策略一致性问题
功能新增、升级、维护困难

复杂的调用关系

精细化的流量调度策略
访问安全机制

云环境部署

更好的服务治理：
更快速响应变更、负载均衡

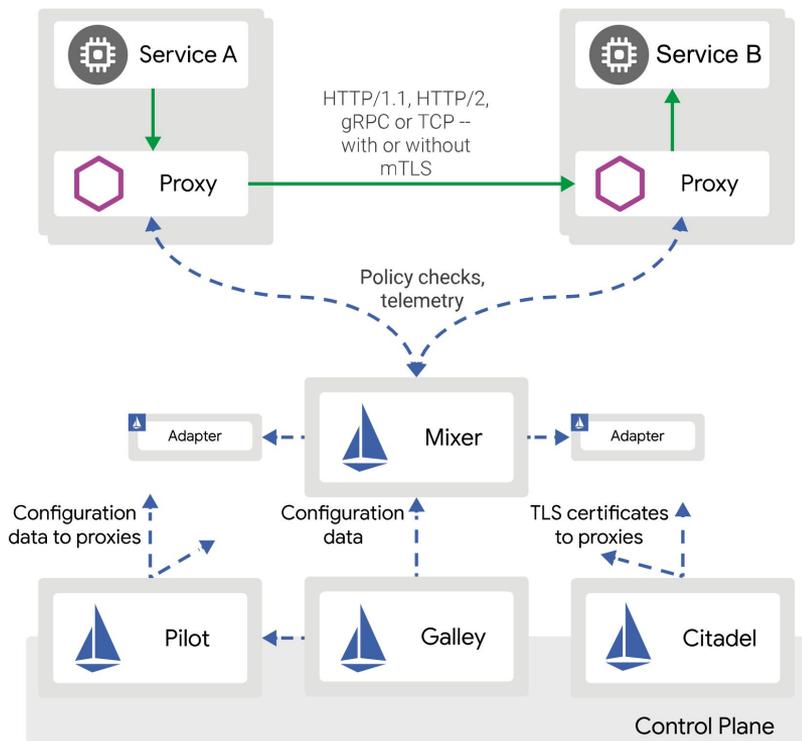
概述 - Service Mesh



火龙果·整理
uml.org.cn

核心思想

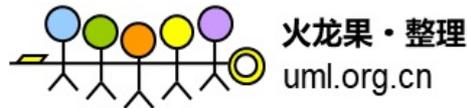
1. 代理层接管所有流量
2. 中心化控制
3. 云平台原生集成



Istio架构(来源: istio官方)

<https://istio.io/docs/concepts/what-is-istio/arch.svg>

概述 - Service Mesh 收益



代理层收益

1. 基础功能收敛
2. 策略一致性
3. 与业务松耦合的功能升级
4. 强制性安全策略
5. 非C++语言服务的性能平滑提升

控制面收益

1. 精细化的流量调度策略
2. 可定制的服务治理功能

核心收益

微服务框架成为可独立迭代的产品

概述 - Service Mesh 风险

性能与稳定性

proxy带来延迟/CPU消耗

proxy稳定性成为关键路径

中心化控制面带来稳定性风险

开发运维模式

问题排查多一层

架构组成为线上问题的汇聚中心

不适用场景

技术栈稳定, 无服务治理/监控等强迭代需求

私有场景下需要长期技术/运维投入

- 概述
 - 从微服务到Service Mesh
- Mesh落地
 - 整体架构
 - 实现方案
 - 关键问题
- 服务治理
 - 访问控制
 - 动态过载保护
 - 染色系统
- 未来展望
 - 规划

ByteMesh整体架构

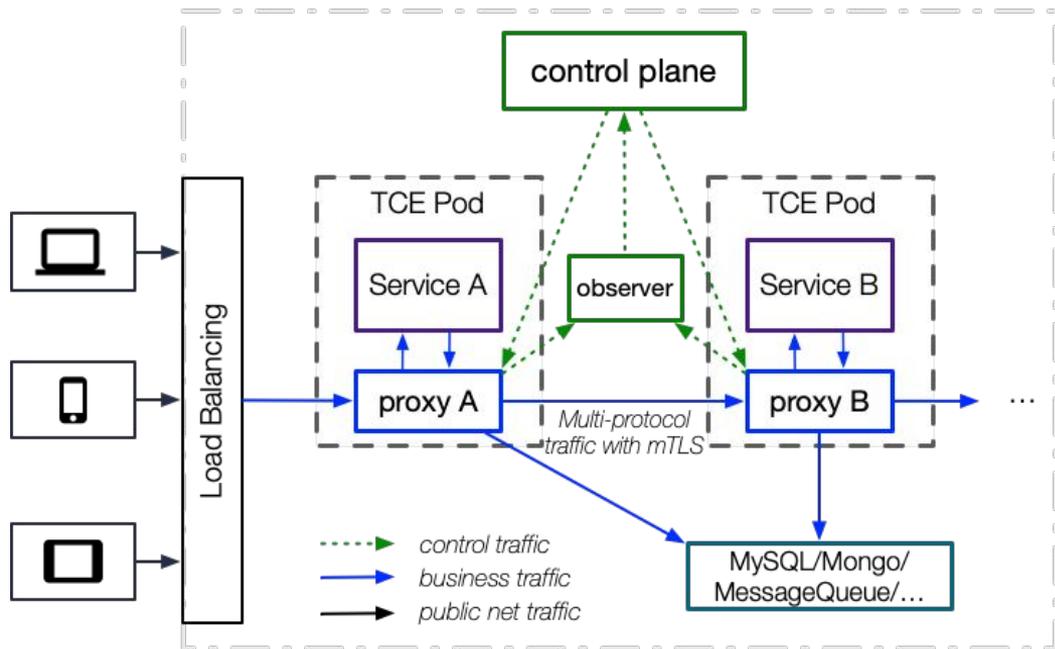


实现方式

1. 代理层接管所有流量，执行路由和服务治理策略，屏蔽语言/协议差异
2. 服务治理中心根据请求信息，动态决定路由策略
3. Cloud Native, 无感知的快速升级

主要特点

1. 基于envoy, 支持私有RPC协议, 支持MySQL, Mongo协议
2. 自研控制面, 集成服务发现和配置
3. mixer使用公司基础设施
4. 统一裸物理机、容器云, 一键升级



ByteMesh实现方案

1. 流量代理(proxy)

- 基于元信息的流量代理
- 多协议, 统一接入规范
- 性能(吞吐/延迟)

2. 控制面(control panel)

- 流量调度和服务治理功能支持
- 高可用
- 扩展能力

3. 云平台集成

- 统一物理机和容器, 与k8s解耦, 原生集成
- 灰度升级和容灾方案
- 大规模proxy平滑、稳定升级

一句话描述:

实现一个高性能多协议的代理和一个灵活可扩展的控制服务, 将他们与云平台原生集成, 最终通过轻量级的RPC框架来输出能力。

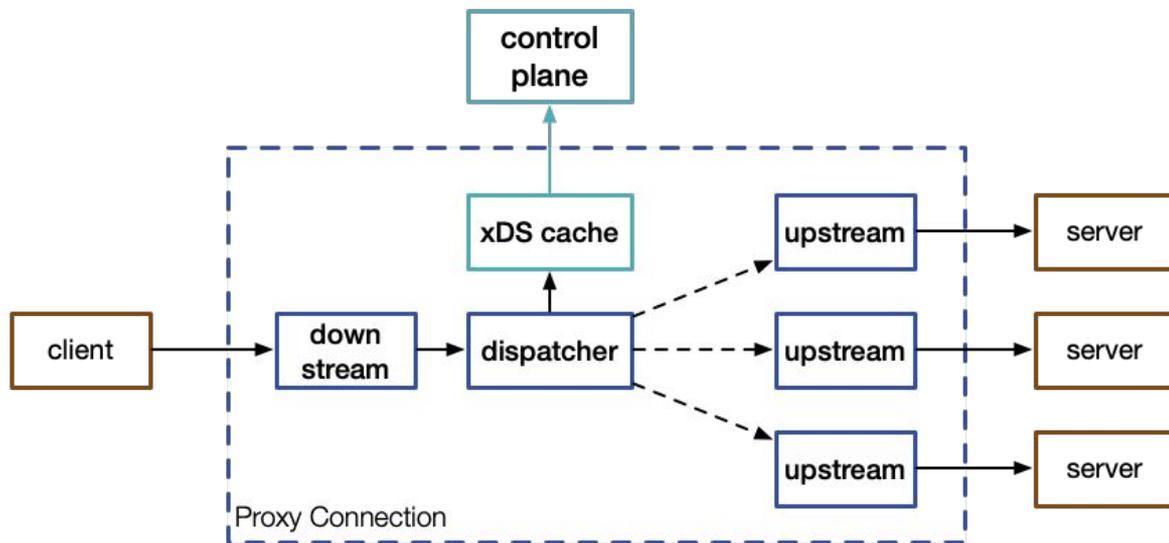
ByteMesh Proxy 事件模型与数据流

基于envoy的流量代理

每个线程对应一个事件循环
每个事件循环对应多个连接
连接隔离

实现

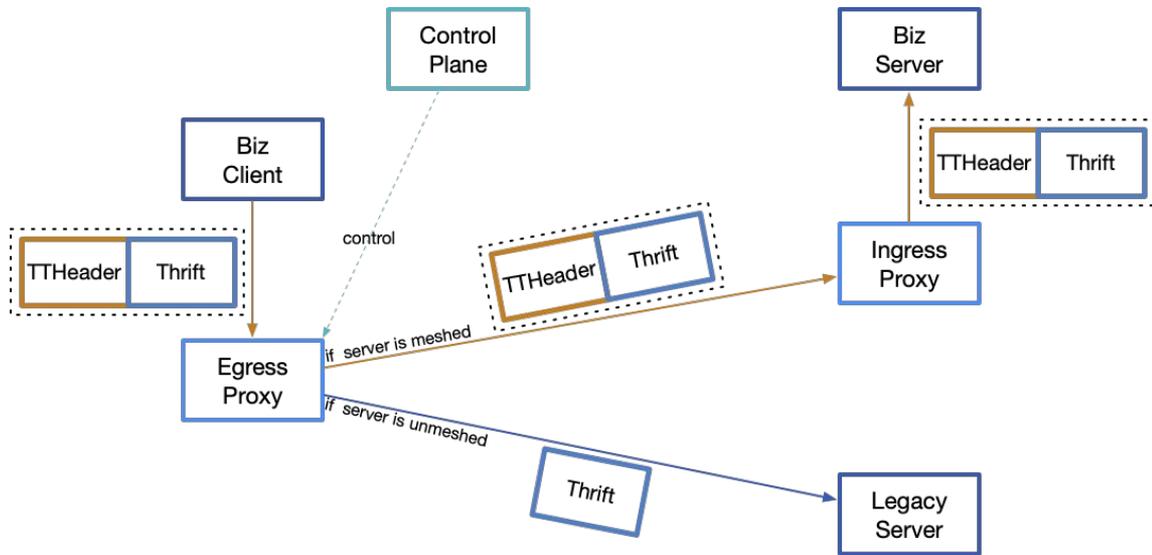
出口/入口分离
长短连接转换
支持thrift、HTTP、MySQL、MongoDB等协议



ByteMesh Proxy 平滑升级

优秀RPC协议的3特征

1. 明确的消息边界
2. 连接多路复用
3. 请求路由能力

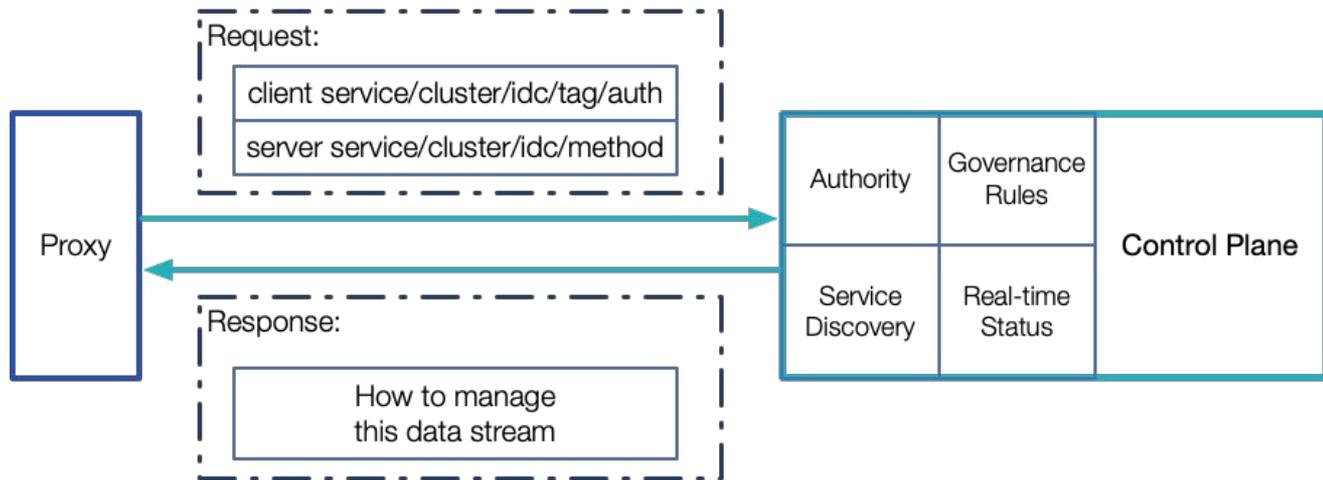


	thrift(buffered)	TTHHeader
消息边界	否	是
连接多路复用	否	是
请求路由能力	基于内容	基于元信息

thrift与TTHHeader协议对比

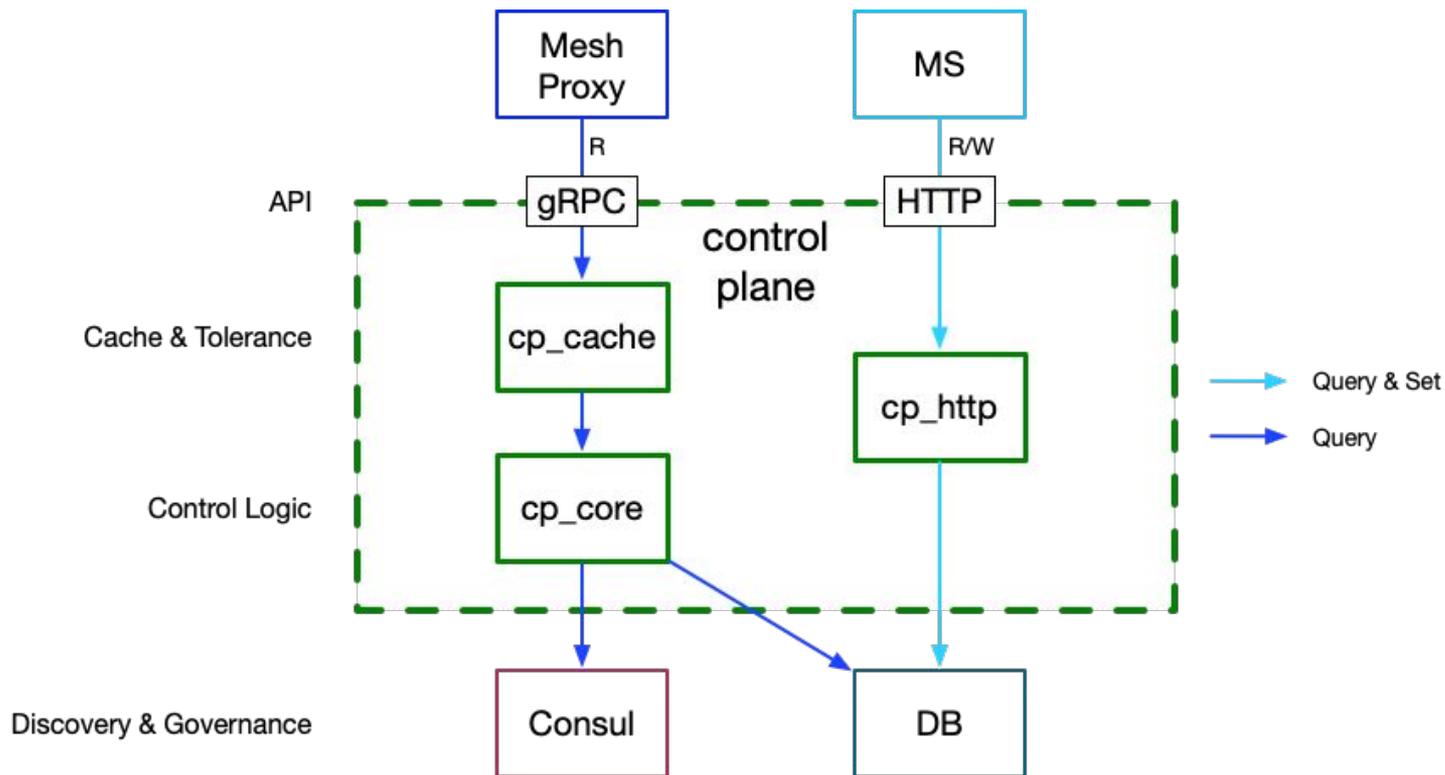
特征

1. 灵活的需求调度
2. 多种服务治理策略
3. 开放式接入协议
4. 屏蔽服务发现和配置



控制面请求处理流程

ByteMesh控制面Arch

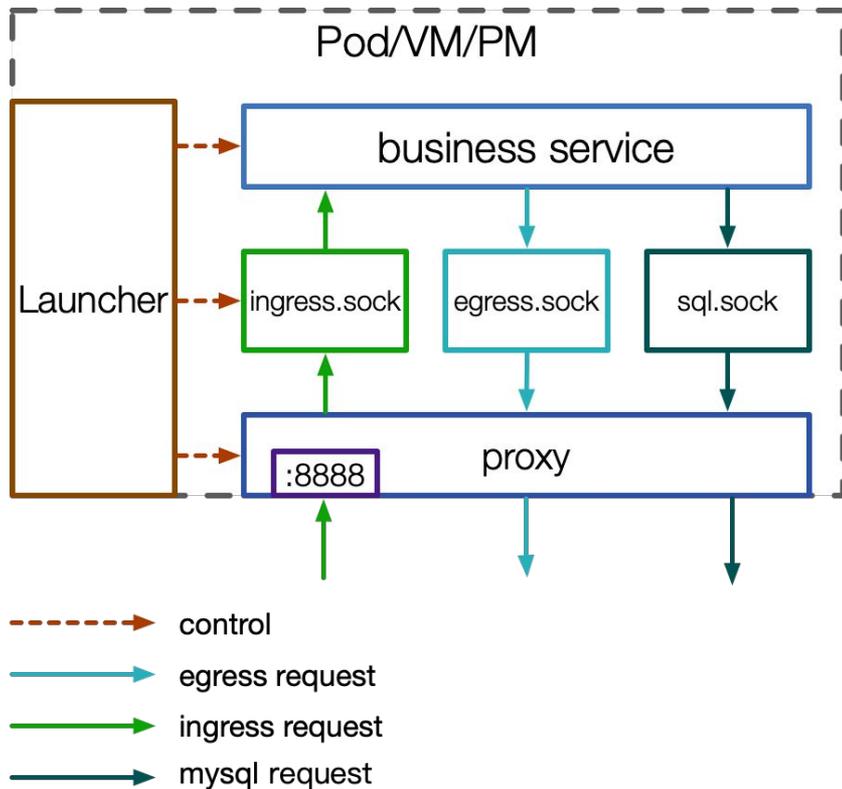


ByteMesh流量劫持



特点

1. proxy以sidecar形式部署
2. 显式接管流量
3. 可独立接管出、入、mysql等流量类型
4. proxy无感知升级



ByteMesh 部署方案

ByteMesh实现总结



1. 制定THeader协议规范
 - a. 实现元信息与用户数据分离, 基于元信息进行路由
 - b. 统一各语言的接入标准
 - c. 支持平滑升级
2. 重点关注proxy性能和稳定性
3. 控制平台面
 - a. 可扩展的插件架构
 - b. 多维tag支持灵活的调度需求
 - c. 多级缓存+分集群部署保证高可用
4. 平台集成
 - a. 显式接管流量
 - b. 通过mesh agent屏蔽底层环境差异

- 概述
 - 从微服务到Service Mesh
- Mesh落地
 - 整体架构
 - 实现方案
 - 关键问题
- 服务治理
 - 访问控制
 - 动态过载保护
 - 染色系统
- 未来展望
 - 规划

Service Mesh下的服务治理新思路



火龙果·整理
uml.org.cn

快速的迭代

精细化治理

定制化能力

标准化接入

ByteMesh 访问控制(ACL)

服务安全的需求

认证 Authorization

授权 Authentication

加密 Encryption

审计 Audit



ByteMesh 访问控制 - 授权

授权粒度

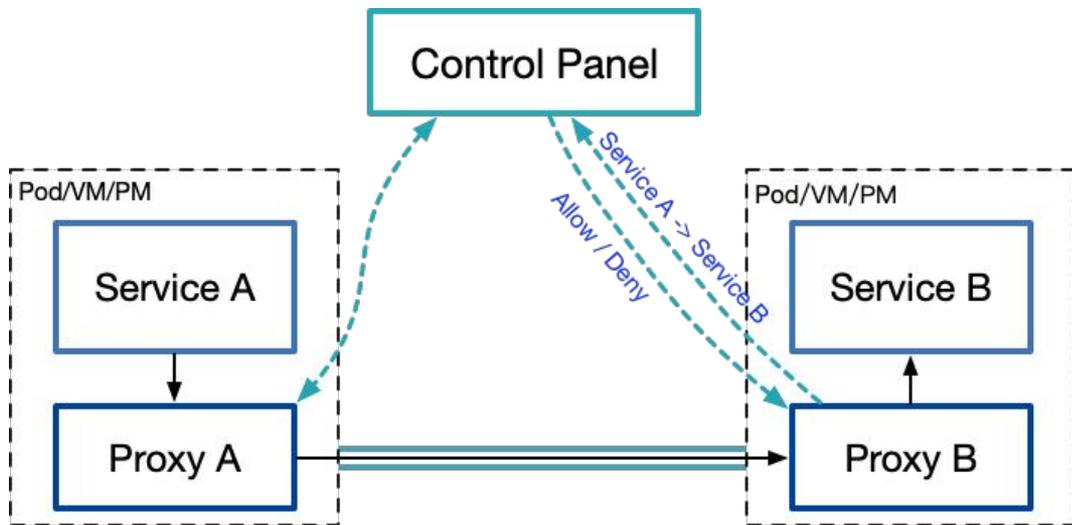
源头: 个人、**服务** ✓、应用 (App)

目标: 服务、**接口** ✓

授权模式

黑名单 - 松散授权(默认)

白名单 - 严格授权(可选)



ByteMesh 访问控制 - 鉴权/加密/审计

鉴权 Authentication

混合云环境下的Token注入

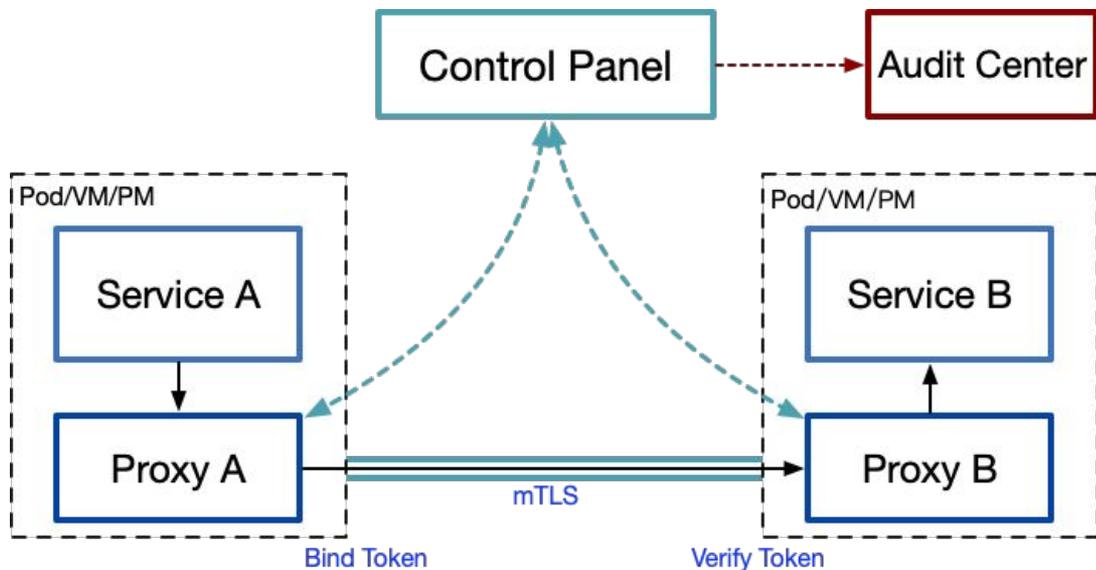
- Kubernetes - Secrets
- VM/PM 机器注入

加密 Envryption

mTLS

审计 Audit

分析调用情况历史



ByteMesh 动态过载保护

判断过载状态: 队列时间

T1: 从请求接收到服务开始处理的时间

决定期望拒绝比例

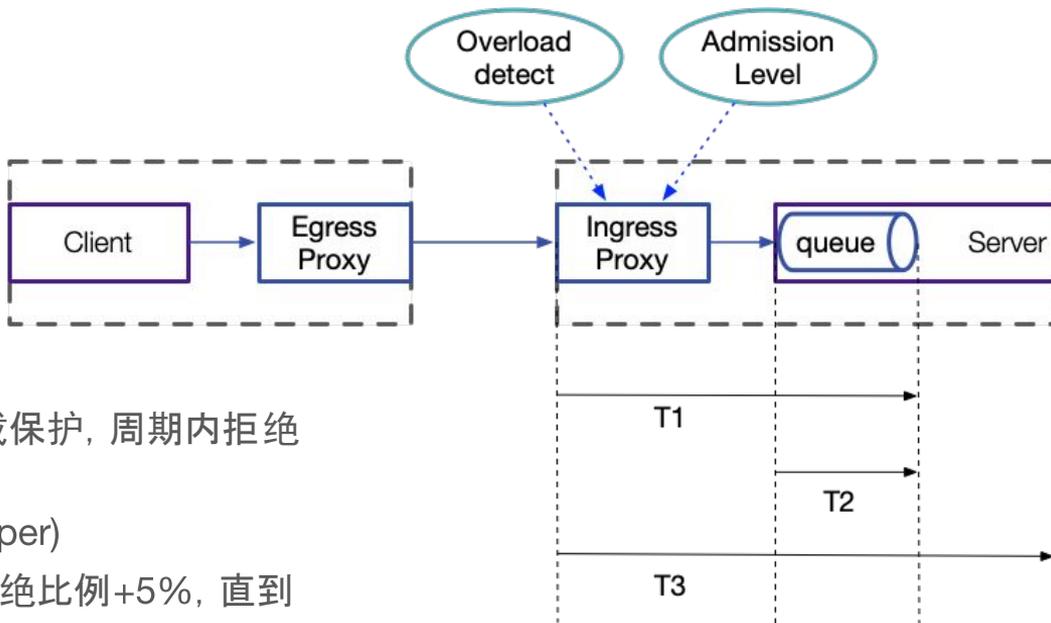
期望拒绝比例指为了完成服务的自动过载保护, 周期内拒绝掉的请求占实际总请求量的比例。

首先设立稳定区间(thres_lower, thres_upper)

当T1超过thres_upper, 加强限制, 期望拒绝比例+5%, 直到期望拒绝100%。

当T1低于thres_lower, 放松限制, 期望拒绝比例-1%, 直到期望拒绝0%。

当T1在稳定区间内, 维持期望拒绝比例。



动态过载保护中的队列时间计算

ByteMesh 动态过载保护



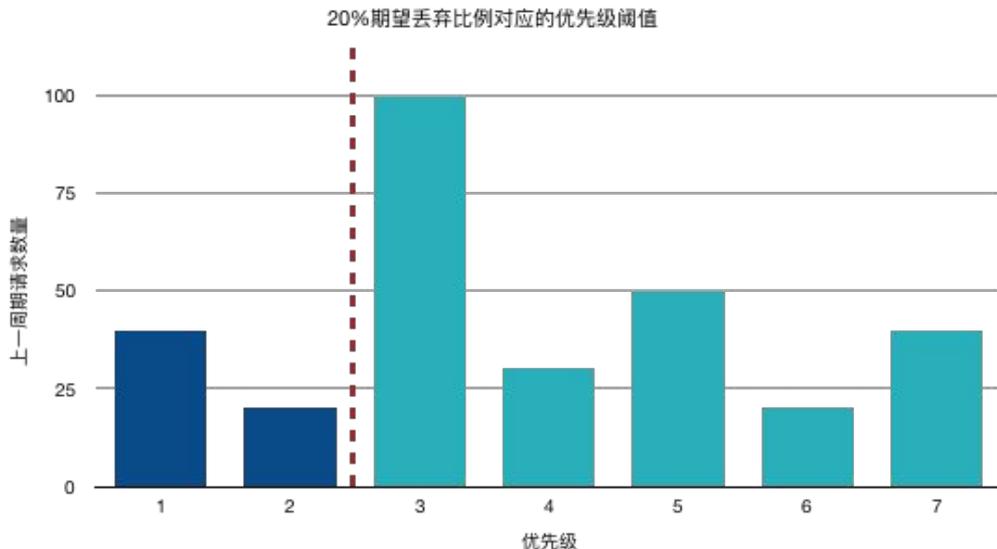
统计优先级直方图

Ingress proxy 统计**每个周期**的请求优先级累积直方图。

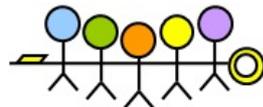
确立拒绝优先级

根据当前的**期望拒绝比例**，按照上一周期的直方图划线，找出**优先级阈值**。

本周期，优先级低于阈值的请求被拒绝，高于阈值的请求可通过。



ByteMesh 染色系统



火龙果·整理
uml.org.cn

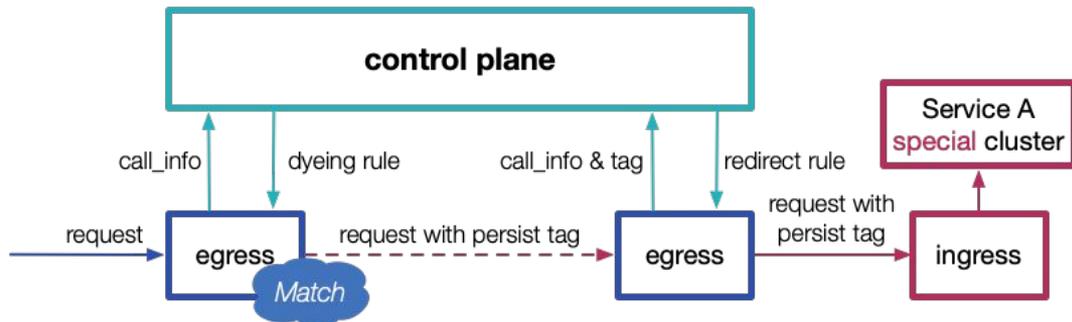
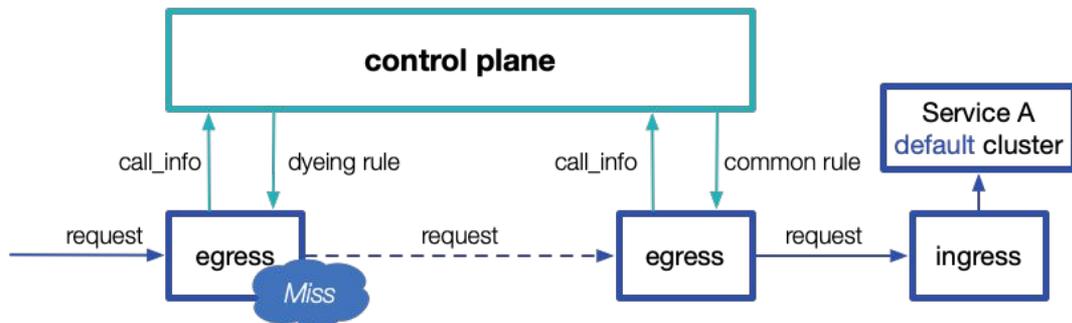
请求染色

根据请求信息进行分组打标签

灵活调度与治理

服务治理平台支持染色 tags

所有策略扩展至逻辑划分



请求染色及路由应用

ByteMesh 服务治理新功能



火龙果·整理
uml.org.cn

1. 动态负载均衡
2. 基于Deadline的Timeout
3. 流量录制
4. 故障注入和容灾演练

- 概述
 - 从微服务到Service Mesh
- Mesh落地
 - 整体架构
 - 实现方案
 - 关键问题
- 服务治理
 - 访问控制
 - 动态过载保护
 - 染色系统
- 未来展望
 - 规划

未来展望



火龙果·整理
uml.org.cn

1. 更广泛的接入场景
2. 持续优化proxy层性能
3. 探索更多的服务治理应用和问题追查系统
4. 构建覆盖全公司的流量视图和流量体系



火龙果·整理
uml.org.cn

THANKS

 ByteDance 字节跳动