蚂蚁金服 ANT FINANCIAL | 金融科技 FINANCIAL TECHNOLOGY

ServiceMesh发展趋势（续）

棋到中盘路往何方

蚂蚁金服 敖小剑

标题中续字的缘由

5月底，我在Cloud Native Meetup上做了一个“ServiceMesh发展趋势：云原生中流砥柱”的演讲，当时主要讲了三块内容：

- Service Mesh产品动态
- Service Mesh发展趋势
- Service Mesh与云原生

今天的内容可以视为是上次演讲部分内容的深度展开，如社区关心的Mixer v2，以及我个人看到的一些业界新的技术方向，如web assembly技术，还有产品形态上的创新，如google traffic director对servicemesh的虚拟机形态的创新支持。

在ServiceMesh出道四年之际，也希望和大家一起带着问题来对ServiceMesh未来的发展进行一些深度思考。

轻松一刻：最近流行的梗，各种灵魂拷问

网约车司机：你清楚你的定位吗?

快递小哥：你是什么东西?

算命先生：你算什么东西?

上海垃圾分拣阿姨：你是什么垃圾?

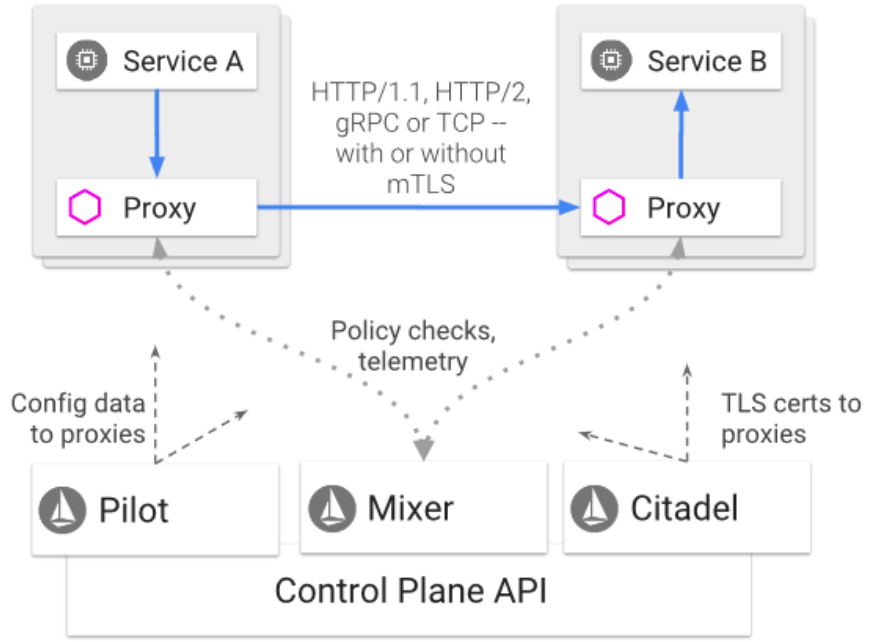
配钥匙师傅：你配吗?

小区保安：你是谁，你从哪里来，你要去哪里?

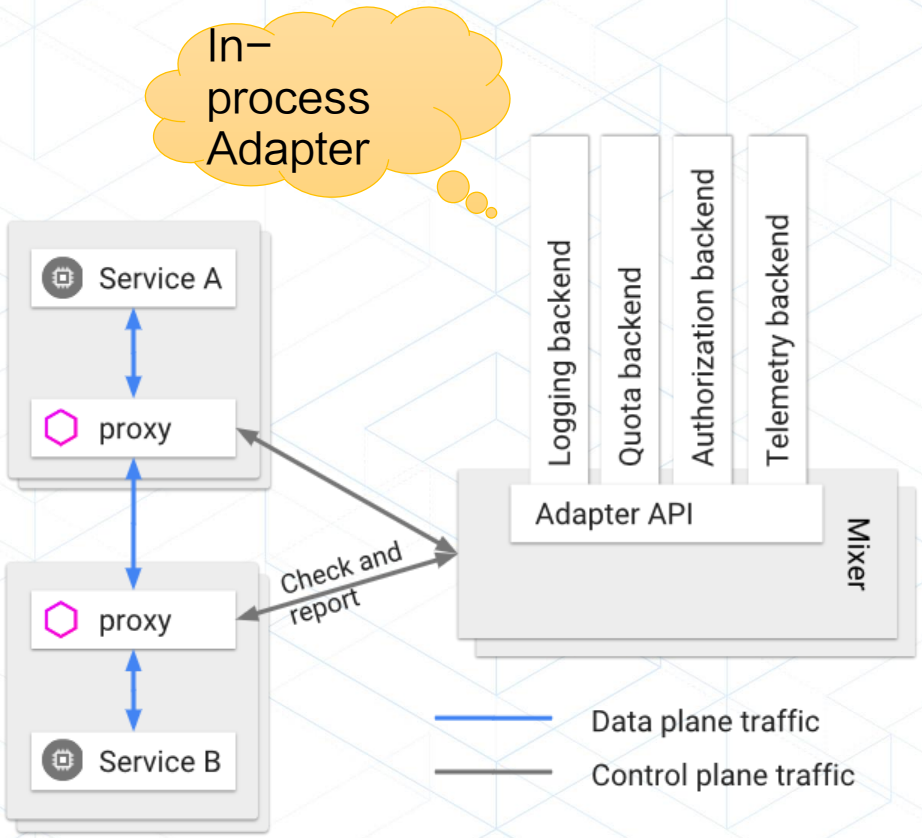
让我们来对ServiceMesh做一次灵魂拷问，首先：

要架构，还是要性能?

Istio的回答：架构优先，性能暂放一边

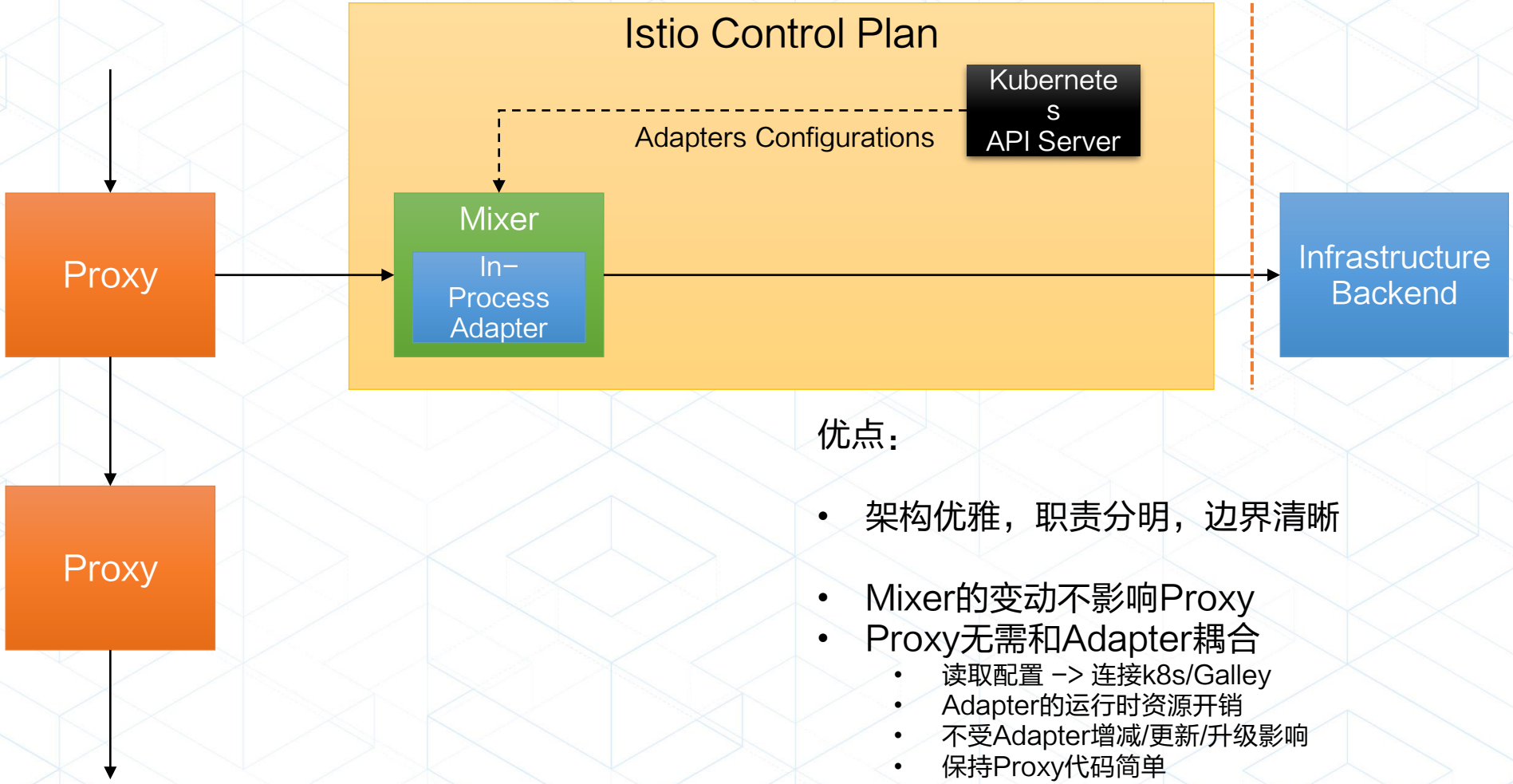


Istio 0.1 – 1.0



Mixer 0.1 – 1.0

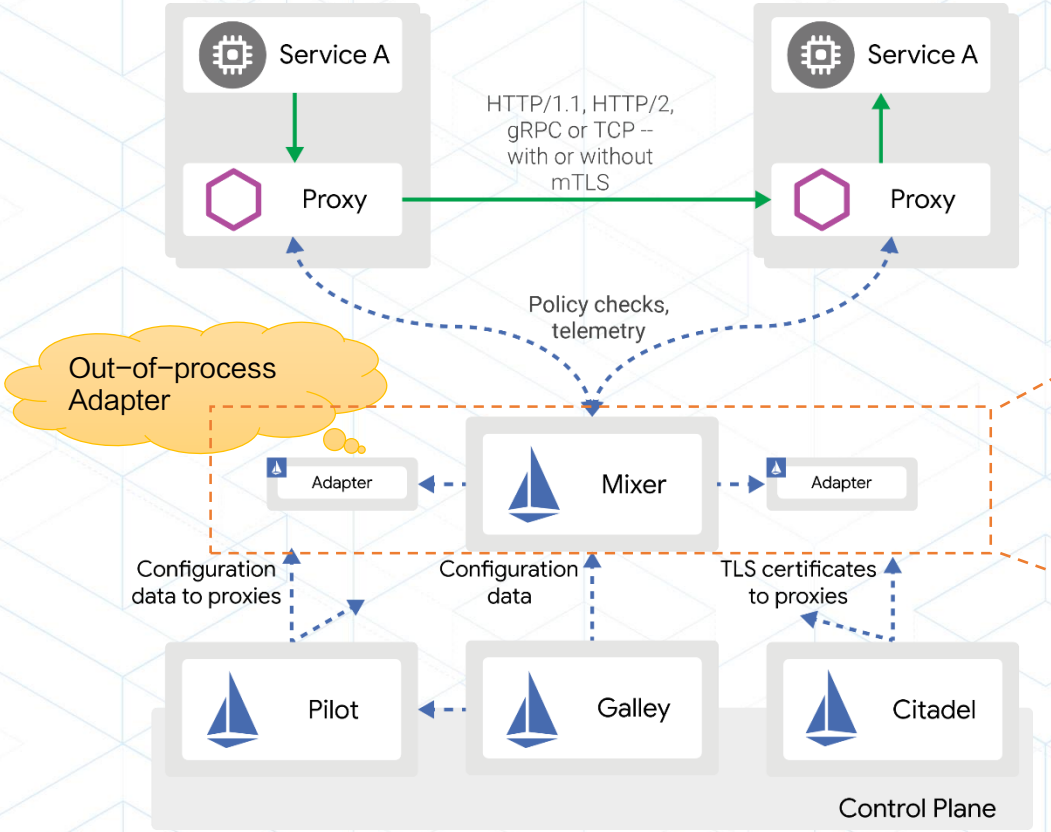
为什么Istio选择Mixer和Proxy分离的架构?



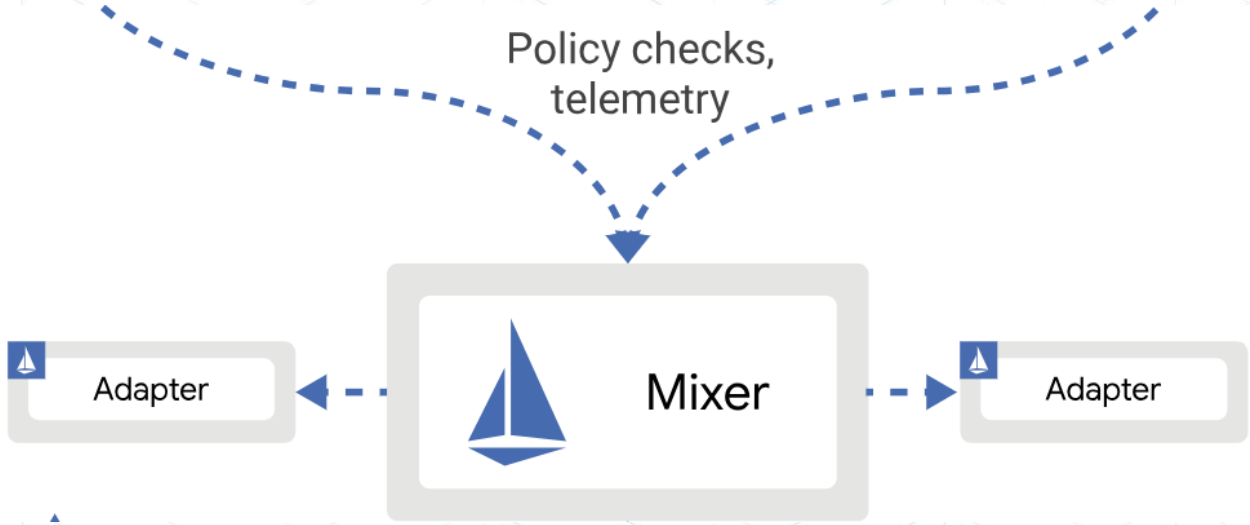
优点：

- 架构优雅，职责分明，边界清晰
- Mixer的变动不影响Proxy
- Proxy无需和Adapter耦合
  - 读取配置 -> 连接k8s/Galley
  - Adapter的运行时资源开销
  - 不受Adapter增减/更新/升级影响
  - 保持Proxy代码简单
- 保持Proxy代码简单
- 数据平面可替换原则

Istio的新回答：架构继续优先，性能继续放一边

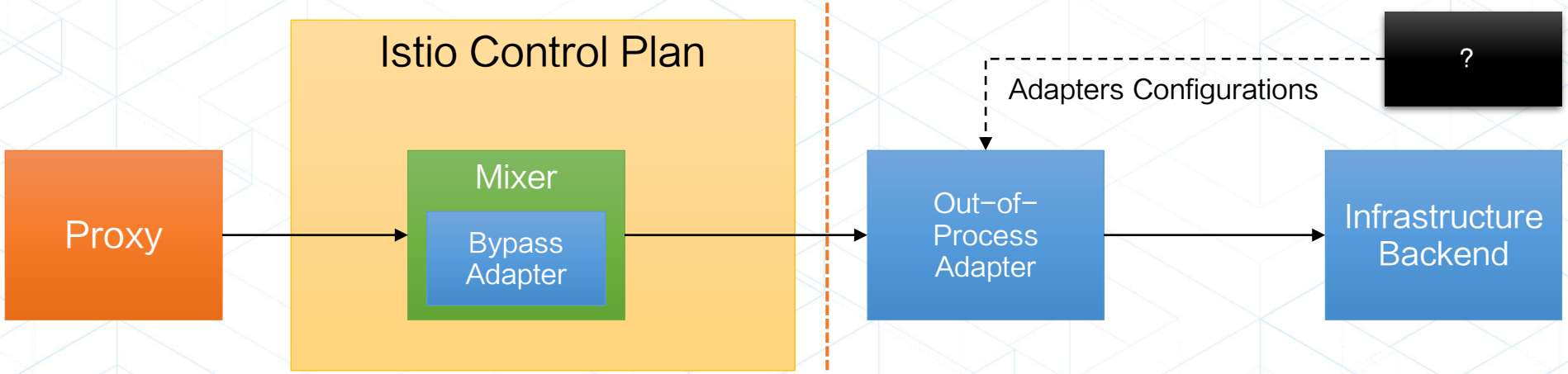


Istio 1.1 – 1.2



Mixer 1.1 – 1.2

为什么Istio选择Out-of-Process Adapter?

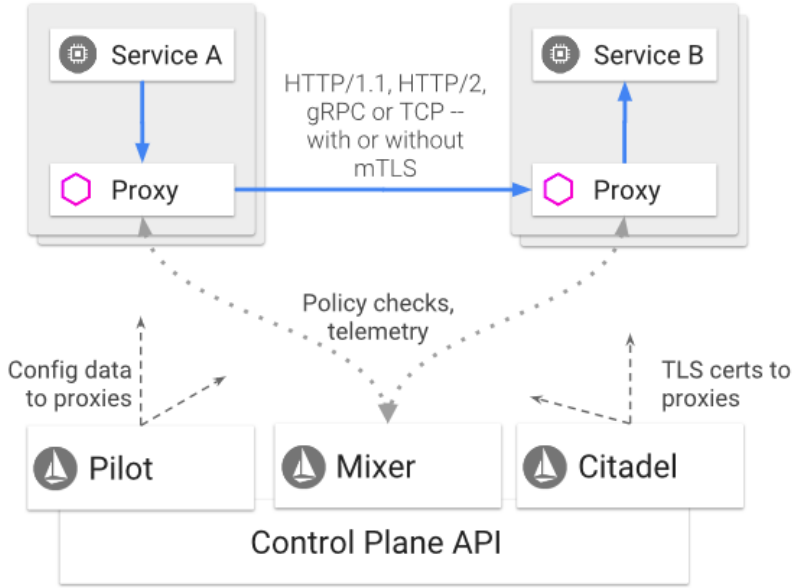


优点：

- 架构更优雅，职责更分明，边界更清晰
- Out-of-Process Adapter不再是Istio的组成部分
  - 安装部署
  - 配置
  - 维护

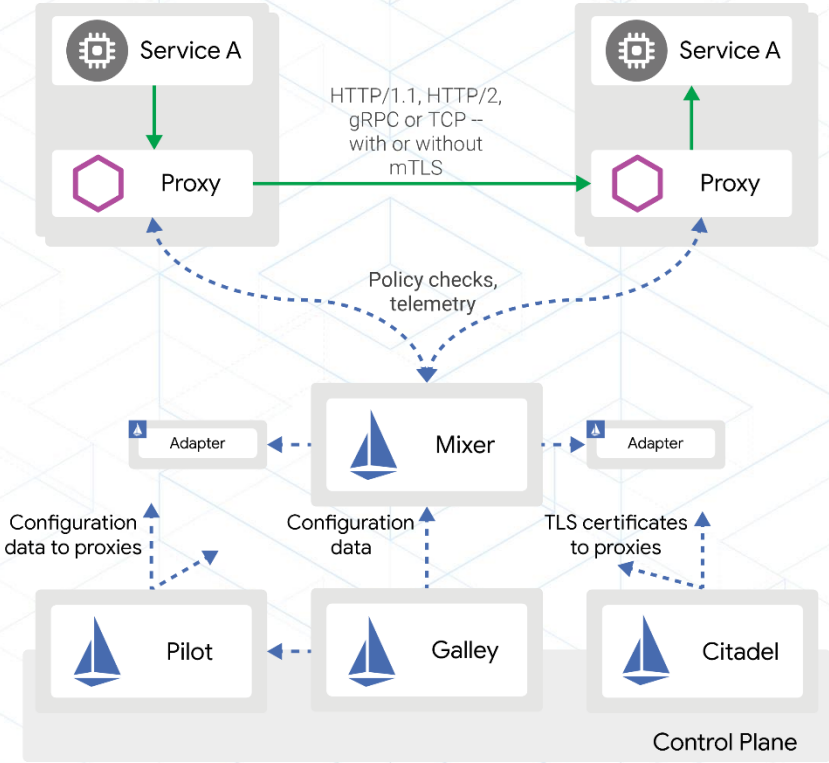
Mixer v1 架构的优点

- 集中式服务：
  - 提高基础设施后端的可用性
  - 为前提条件检查结果提供集群级别的全局2级缓存
- 灵活的适配器模型，使其以下操作变得简单：
  - 运维添加、使用和删除适配器
  - 开发人员创建新的适配器（超过20个适配器）

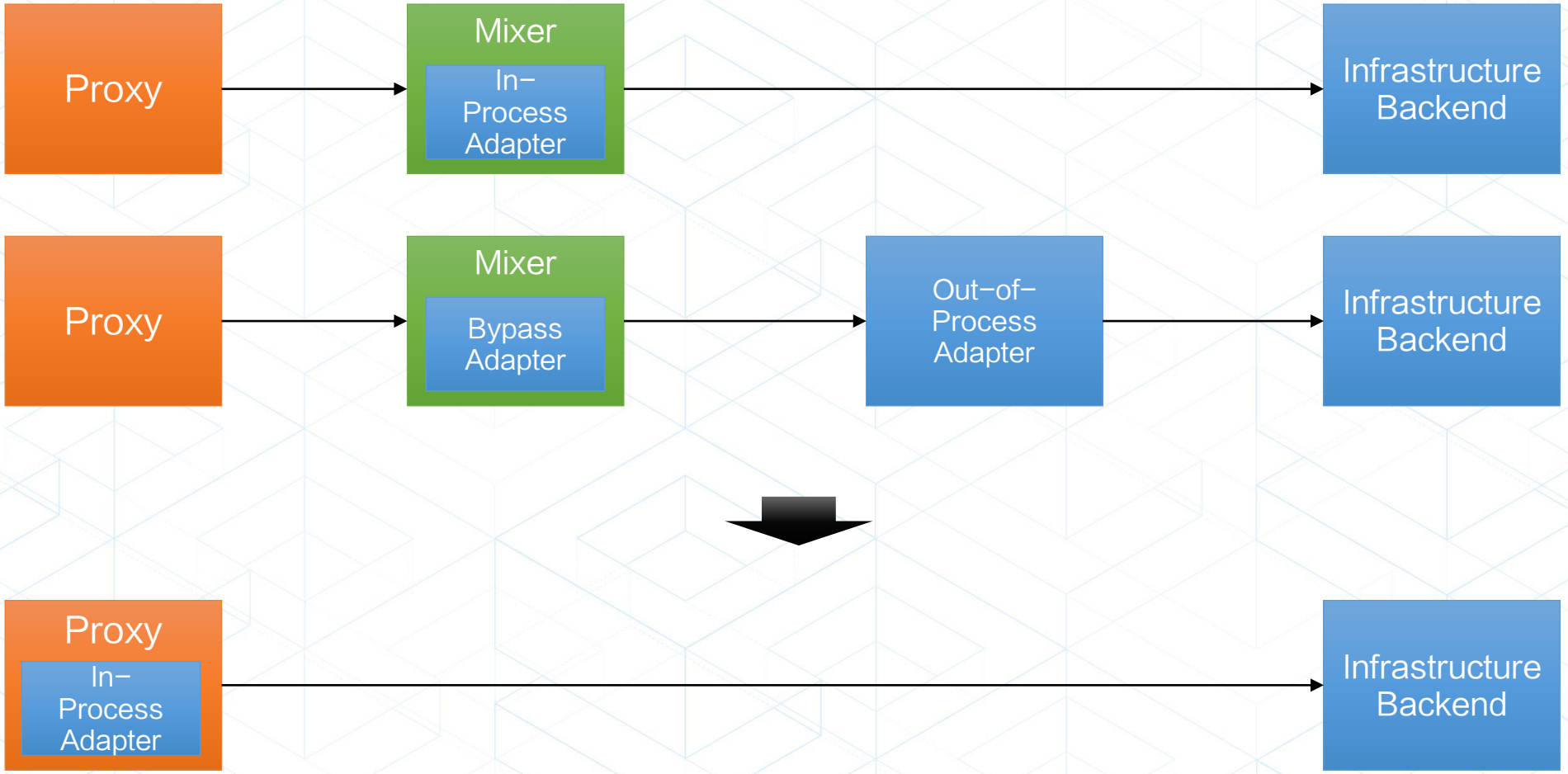


Mixer v1 架构的缺点

- 管理开销
  - 管理Mixer是许多客户不想负担的
  - 进程外适配器强制运维管理适配器，增加此负担
- 性能
  - 即使使用缓存，在数据路径中同步调用Mixer也会增加端到端延迟
  - 进程外适配器进一步增加了延迟
  - 授权和认证功能是天然适合mixer pipeline的，但是由于mixer 设计的延迟和SPOF（单点故障）特性，导致直接在Envoy中实现(Envoy SDS)
- 复杂性
  - Mixer使用一组称为模板的核心抽象，来描述传递给适配器的数据。这些包括“metrics”，“logentry”，“tracepan”等。这些抽象与后端想要消费的数据不匹配，导致运维需要编写一些手动配置，以便在规范的 Istio 样式和后端特定的样式之间进行映射。原本期望这种映射可以在适配器中实现很大程度上的自动化，但是最终还是太复杂并需要手动配置。



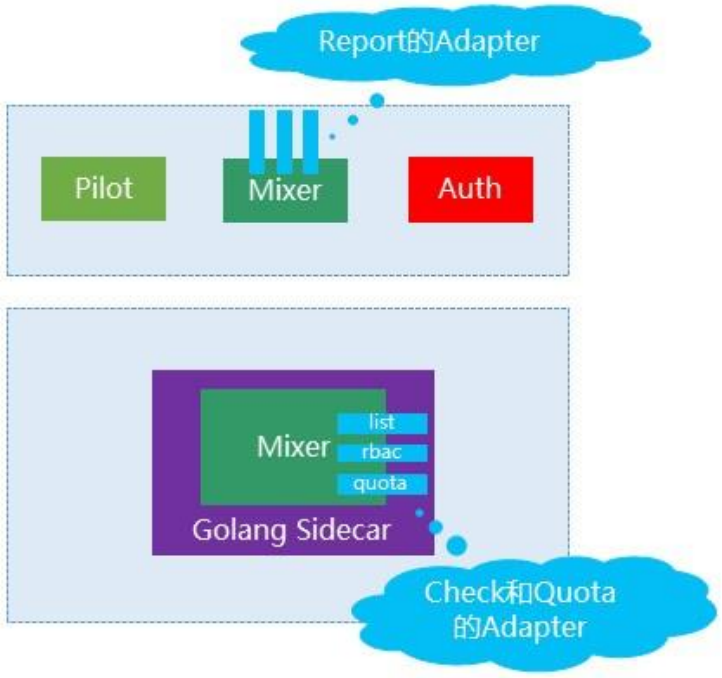
如果要性能，该怎么做?


Proxy
Mixer
In-Process Adapter
Infrastructure Backend
Proxy
Mixer
Bypass Adapter
Out-of-Process Adapter
Infrastructure Backend
Proxy
In-Process Adapter
Infrastructure Backend

蚂蚁先行一步 ☺

最大的改变：合并部分Mixer功能

- ✓ Mixer三大功能：
  - Check – 同步阻塞
  - Quota – 同步阻塞
  - Report – 异步批量
- ✓ 合并Check和Quota
- ✓ Report暂时保留在Mixer中



大规模微服务架构下的Service Mesh探索之路

2018-06-30

Istio社区的Proposal

Mixer V2 Architecture

https://docs.google.com/document/d/1QKmtem5jU_2F3Lh5SqLp0IuPb80_70J7aJEYu4_gS-s/edit#heading=h.hvvcgepdykro

（摘要翻译：https://skyao.io/learning-istio/mixer/design/v2.html ）

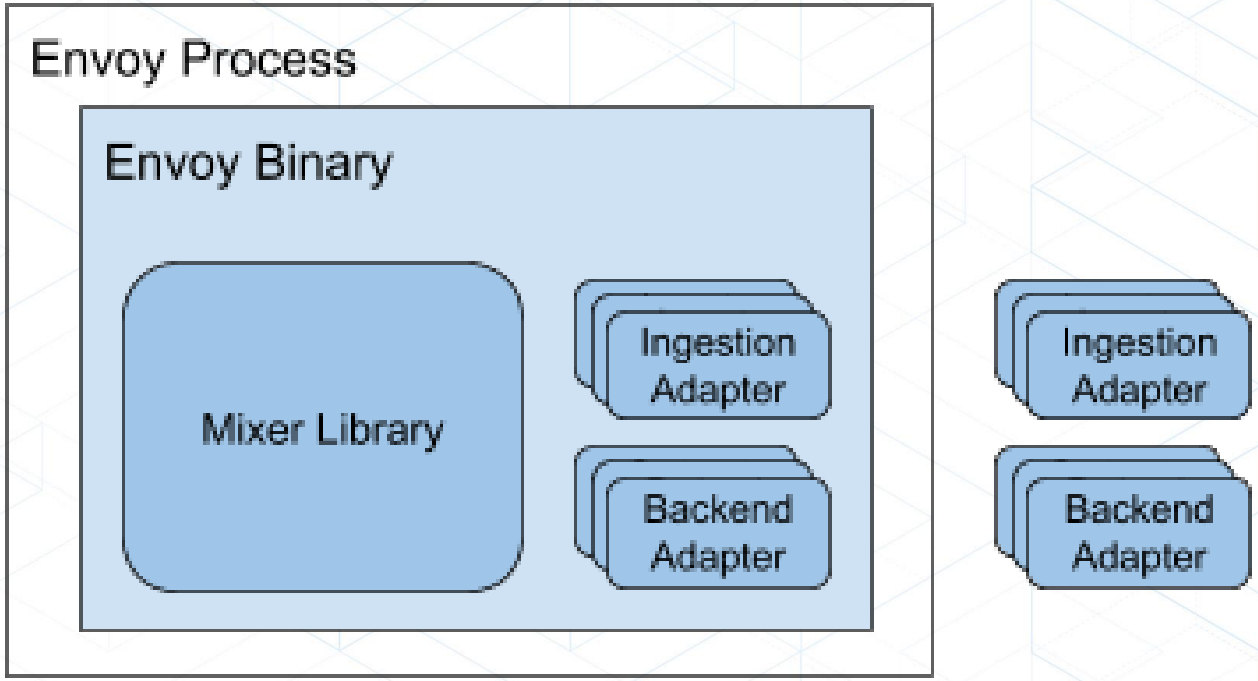
OWNER: MTAIL@GOOGLE.COM
WORK-GROUP: POLICIES AND TELEMETRY
SHORT SELF LINK:
REVIEWERS: XXXX[ ], XXX []

STATUS: WIP | IN REVIEW | APPROVED | OBSOLETE
CREATED: 12/18/2018
RELEASE VERSION: N/A
APPROVERS: XXX [], XXX []

Mixer v2 Proposal的核心

Mixer-In-Proxy. Mixer will be rewritten in C++ and directly embedded in Envoy. There will no longer be any stand-alone Mixer service. This will improve performance and reduce operational complexity.

Mixer-In-Proxy/Mixer合并进Proxy。 Mixer 将用C ++重写并直接嵌入到Envoy。 将不再有任何独立的 Mixer 服务。 这将提高性能并降低运维复杂性。



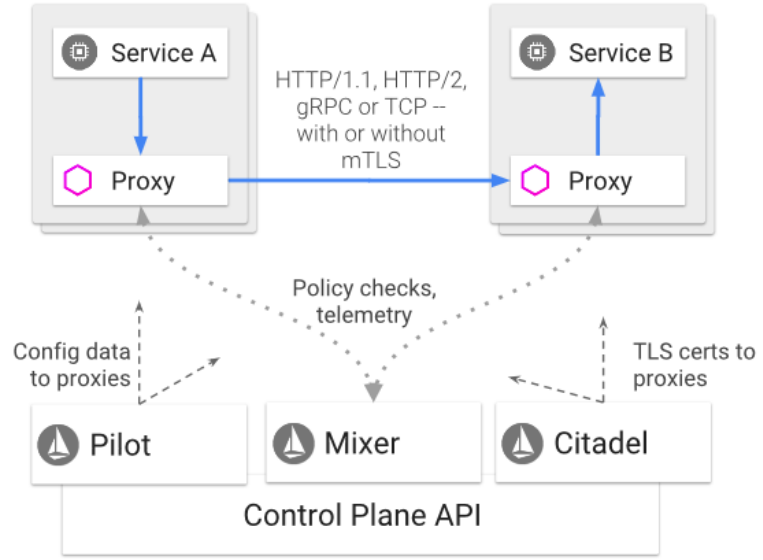
Mixer合并到Sidecar之后

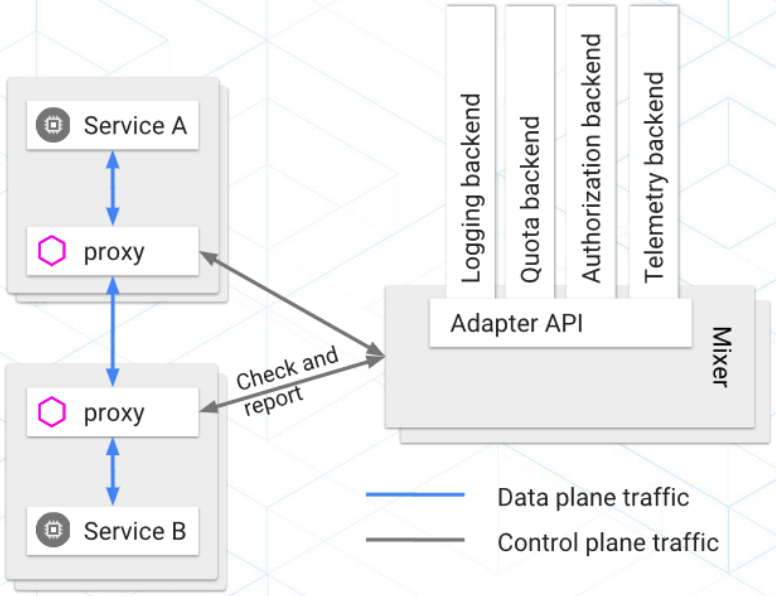
性能有了，架构怎么办?

Mixer v1的优点不应该成为Mixer v2的缺点

优点：

- 架构优雅，职责分明，边界清晰
- Mixer的变动不影响Proxy
- Proxy无需和Adapter耦合
  - 读取配置 -> 连接k8s/Galley
  - Adapter的运行时资源开销
  - 不受Adapter增减/更新/升级影响
  - 保持Proxy代码简单
- 保持Proxy代码简单
- 数据平面可替换原则





- 集中式服务：
  - 提高基础设施后端的可用性
  - 为前提条件检查结果提供集群级别的全局2级缓存
- 灵活的适配器模型，使其以下操作变得简单：
  - 运维添加、使用和删除适配器
  - 开发人员创建新的适配器（超过20个适配器）

合并没问题，如何合并才是问题

Envoy在设计上是可扩展的

扩展点：

- L4/L7 filters
- Access loggers
- Tracers
- Health checkers
- Transport sockets
- Retry policy
- Resource monitors
- Stats sink

扩展方式：

- C++
- Lua(目前仅限于HTTP Traffic)
- Go extensions (beta, for Cilium)

Envoy最新的扩展方式：Web Assembly

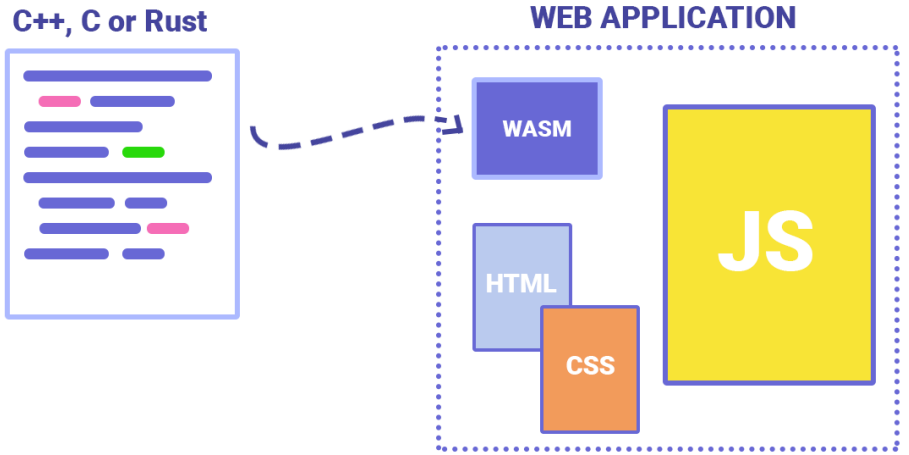
- Support WebAssembly (WASM) in Envoy
  - https://github.com/envoyproxy/envoy/issues/4272
  - 计划在1.12版本提供（1.11版本发布于7月12日）
- envoy-wasm项目：Playground for Envoy WASM filter
  - https://github.com/envoyproxy/envoy-wasm

Web Assembly介绍

Mozilla官方定义：

WebAssembly是一种新的编码方式，可以在现代的网络浏览器中运行 － 它是一种低级的类汇编语言，具有紧凑的二进制格式，可以接近原生的性能运行，并为诸如C / C ++等语言提供一个编译目标，以便它们可以在Web上运行。它也被设计为可以与JavaScript共存，允许两者一起工作。

WebAssembly不是一门编程语言，而是一份字节码标准。
WebAssembly字节码是一种抹平了不同CPU架构的机器码，
WebAssembly字节码不能直接在任何一种CPU架构上运行，
但由于非常接近机器码，可以非常快的被翻译为对应架构的机器码，因此WebAssembly运行速度和机器码接近。（类比Java bytecode）



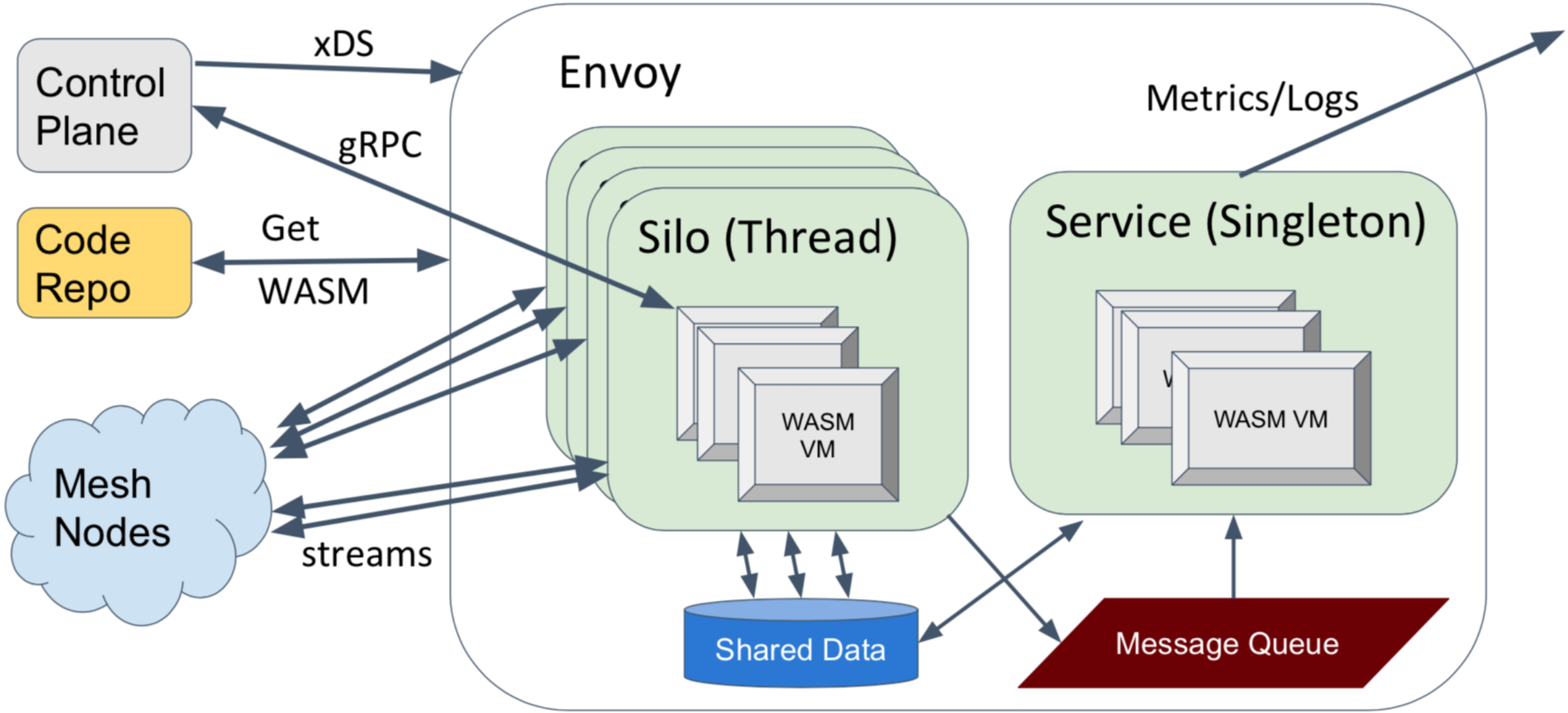
（摘录自http://blog.enixjin.net/webassembly-introduction/）

使用Web Assembly扩展Envoy的好处

- 避免修改Envoy
- 避免网络远程调用（check & report）
- 通过动态装载（重载）来避免重启envoy
- 隔离性
- 实时A/B测试

（内容来自”Extending Envoy with WebAssembly”）

Envoy支持Web Assembly的架构



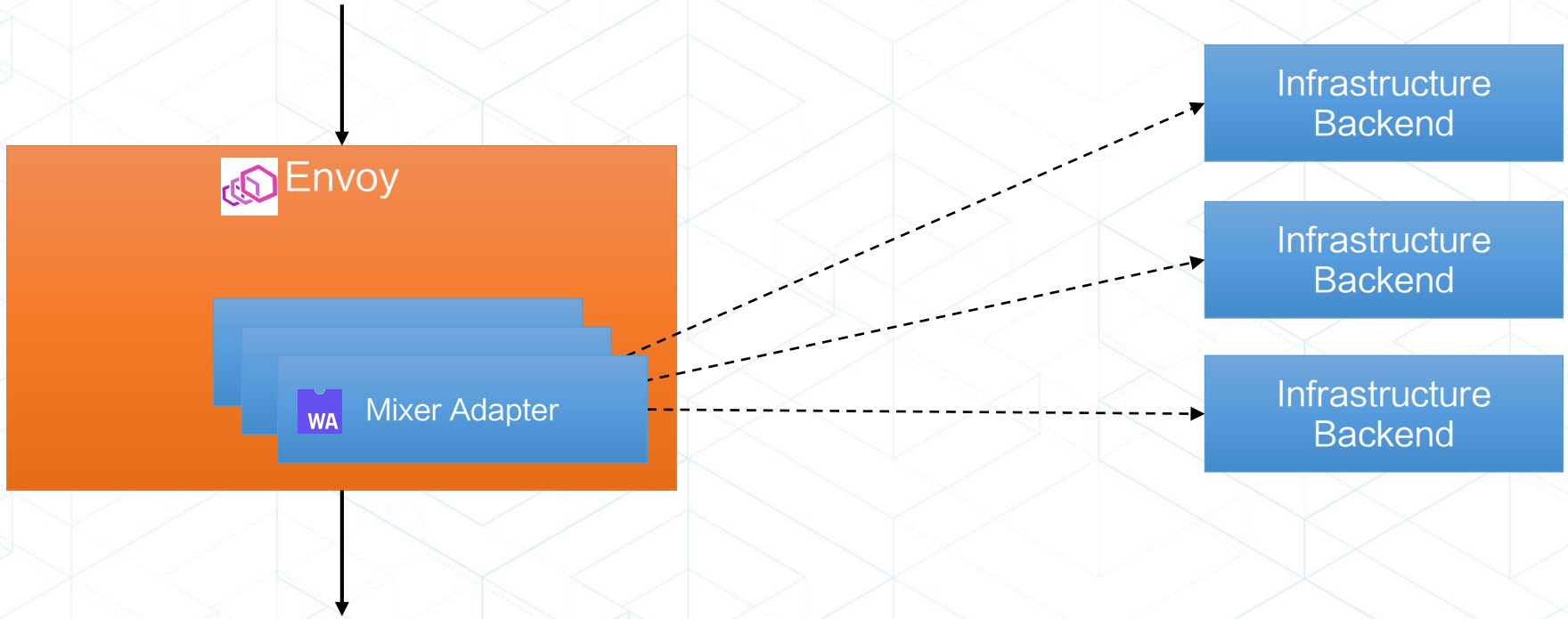
（图片来自”Extending Envoy with WebAssembly”）

Envoy支持的Web Assembly VM

- WAVM (https://github.com/WAVM/WAVM)
- V8(https://v8.dev/)
- Null Sandbox (use the API, compile directly into Envoy)

（内容来自”Extending Envoy with WebAssembly”）

支持Web Assembly扩展的Mixer v2：终极目标形态



望穿秋水，路阻且长

Envoy对Web Assembly的支持预计有希望在3-6个月内实现

https://github.com/envoyproxy/envoy/issues/4272

- 2018年8月28日，提出Issue
- 2018年10月开始动手
- 2019年5月poc完成，创建envoy-wasm项目
- 目前放在envoy的下一个milestone 1.12中

Mixer v2从提出到现在8个月了，依然是In Review状态

- Istio能否接受Mixer v2?
- 如果接受，什么时候开工?
- 如果开工，什么时候完工?
- 如果完工，什么时候稳定?

OWNER: MTAIL@GOOGLE.COM
WORK-GROUP: POLICIES AND TELEMETRY
SHORT SELF LINK:
REVIEWERS: XXXX[ ], XXX []

STATUS: WIP | IN REVIEW | APPROVED | OBSOLETE
CREATED: 12/18/2018
RELEASE VERSION: N/A
APPROVERS: XXX [], XXX []

有了高大上的容器/k8s/云原生，

还要不要支持土里土气的

虚拟机?

ServiceMesh主流产品对虚拟机的支持情况

Linkerd 1.*

天然支持

Envoy

天然支持

Istio

开始准备支持，后来实质性取消，基本只有k8s好用

Linkerd 2.0

只支持k8s

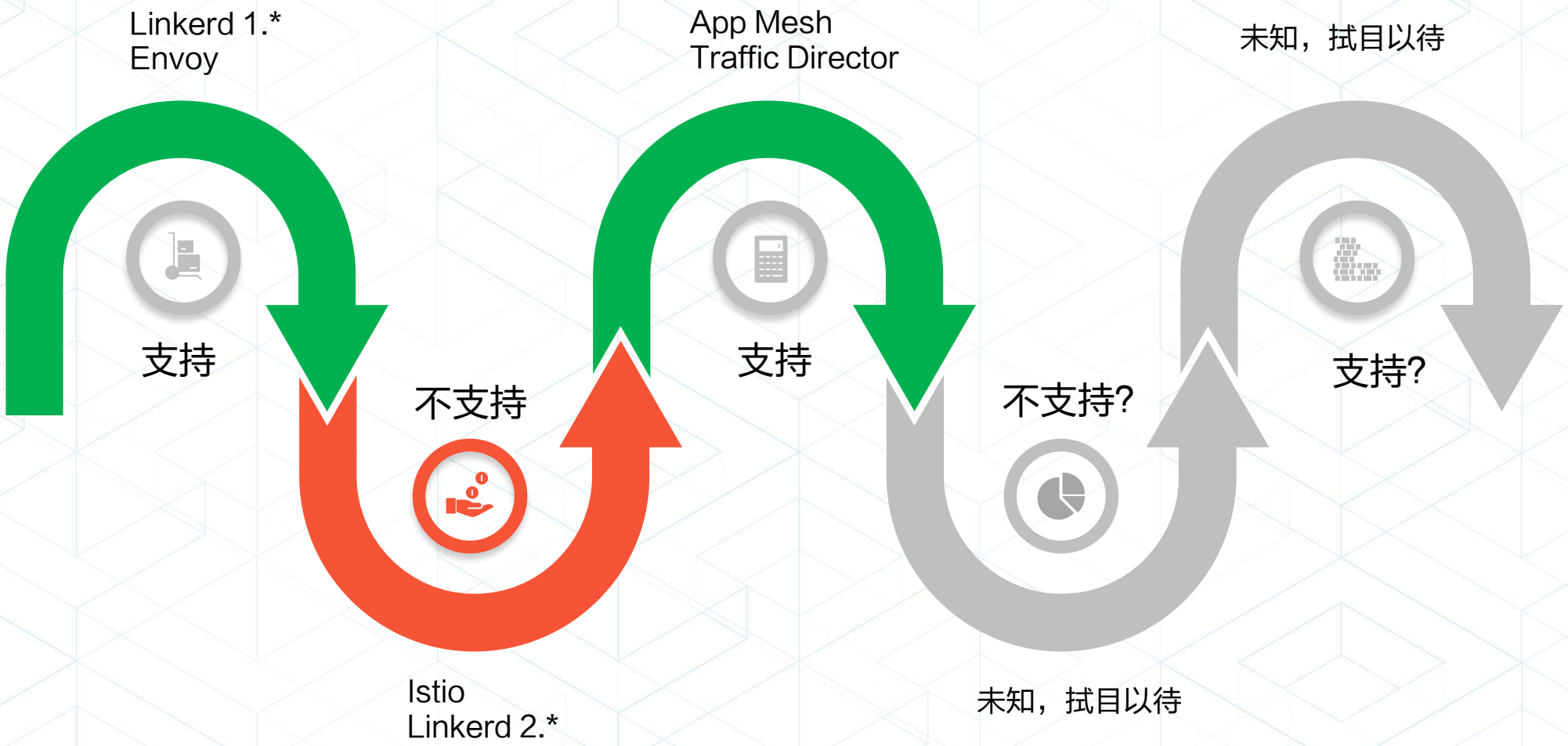
AWS app mesh

支持虚拟机，而且可以虚拟机和容器相互访问

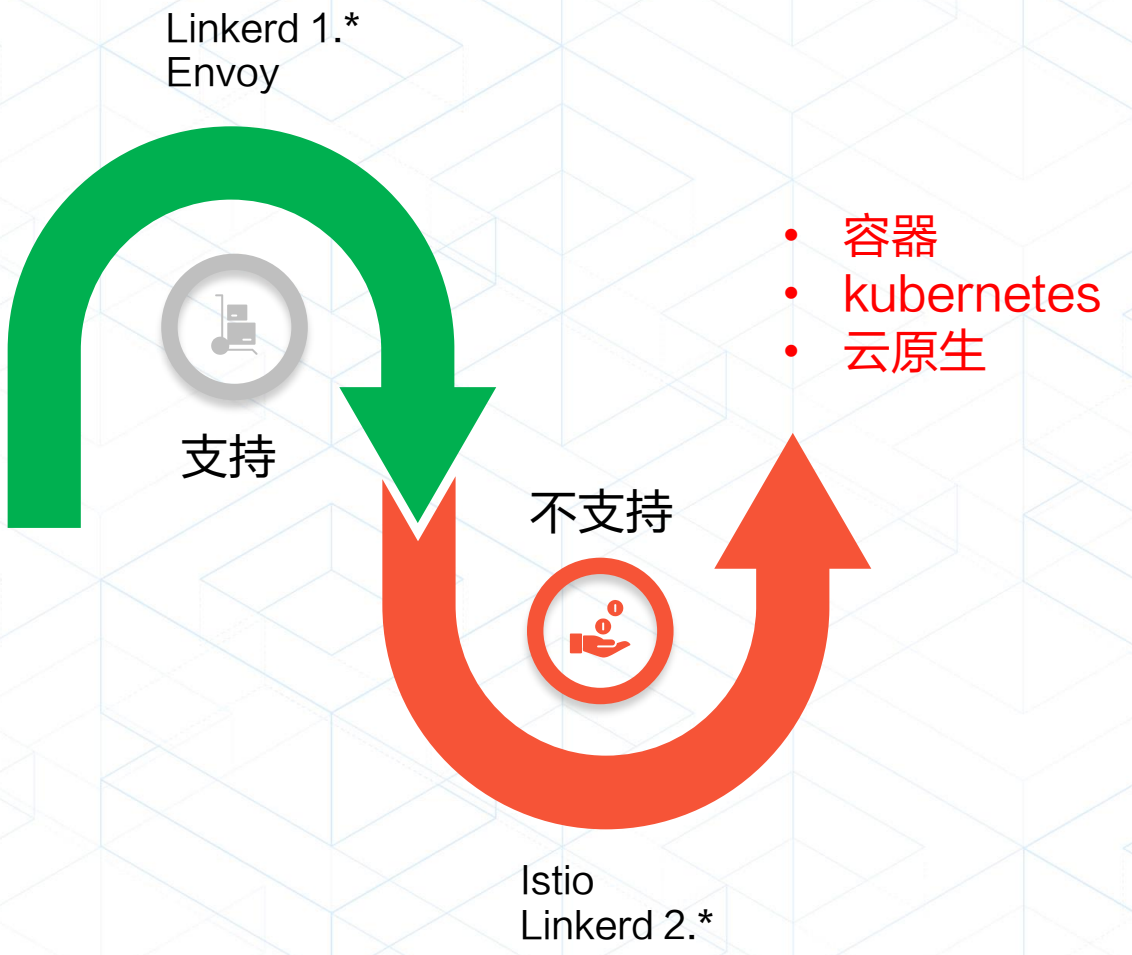
Google Traffic Director

支持虚拟机，而且可以虚拟机和容器相互访问

历史总是惊人的相似，螺旋式上升?

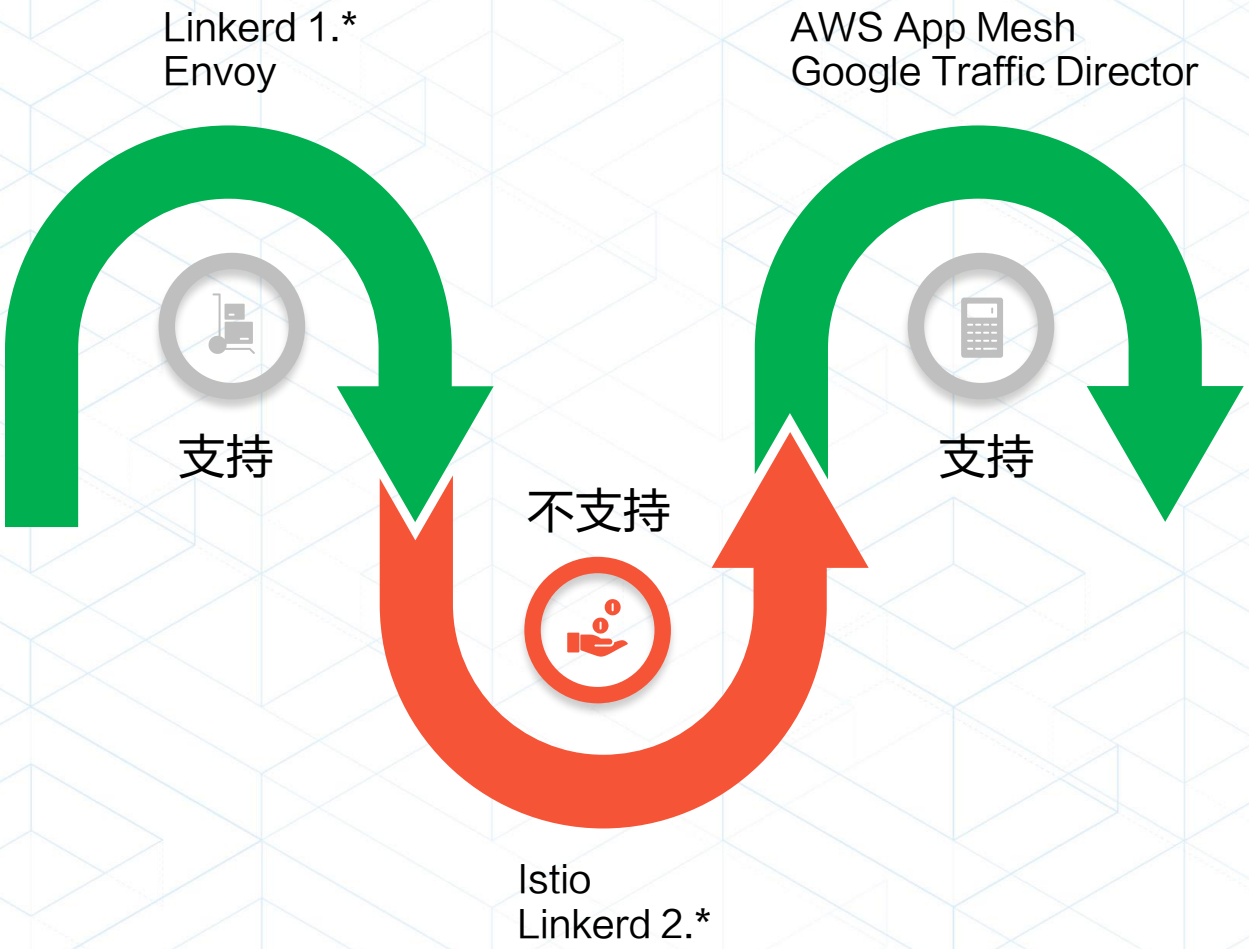


第一个转折容易理解：相比虚拟机，k8s提供了太多便利



- 容器
- kubernetes
- 云原生

第二个转折做何解?



App Mesh：我原以为只有我这长相的会背叛革命，想不到你这浓眉大眼的家伙也背叛革命了。

理想（云原生普及）和现实（虚拟机大量存在）的差距

[DreamMesh抛砖引玉(2)-CloudNative]

2018-02-10

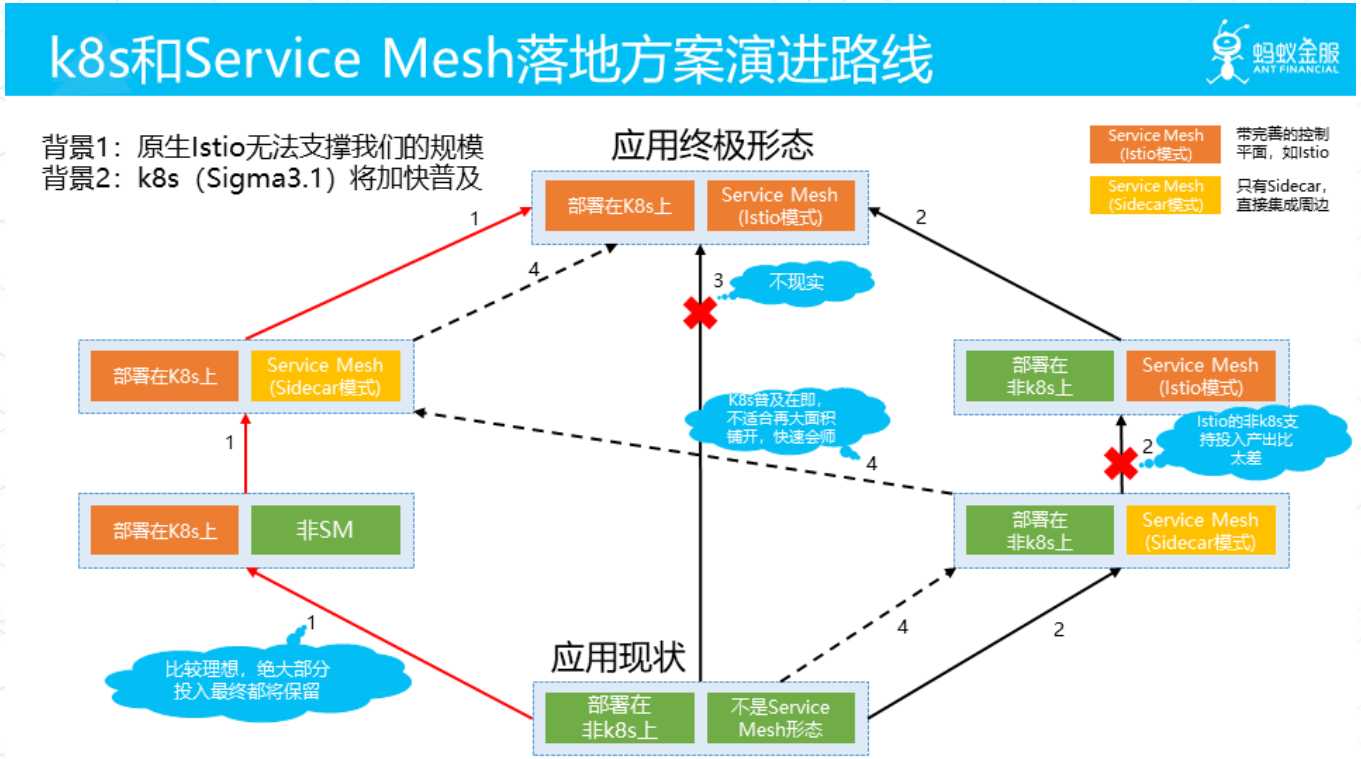
理想很丰满，现实很骨感。Cloud Native虽然令人向往，然而现实中，有多少企业是真的做好了Cloud Native的准备?

问题：到底该先容器/k8s，再上微服务/servicemesh；还是先微服务/servicemesh，再上容器/k8s?

每个公司都会有自己的实际情况和选择。

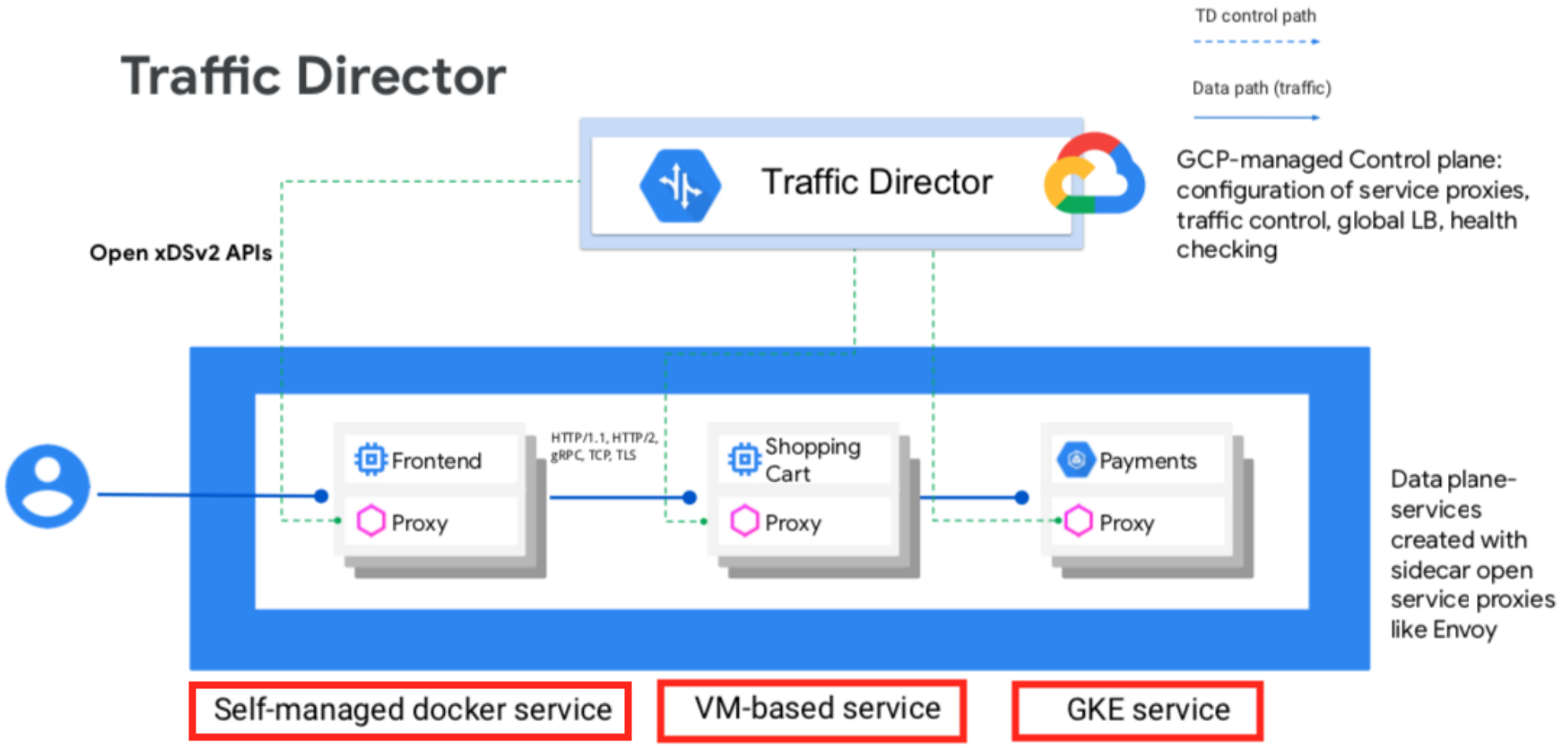
[蚂蚁金服Service Mesh渐进式迁移方案]

2018-11-25



Service Mesh 和 k8s 落地可能的多种演进路线

Google Traffic Director的选择：ServiceMesh先行



Traffic Director 官方文档如是说：“按您的节奏进行现代化改造”

托管式实例组：效仿容器和k8s的方式来管理虚拟机

	容器/k8s	虚拟机（托管式实例组）
管理实例副本（伸缩）	k8s HPA或者serverless	通过实例模版设置自动伸缩
管理实例副本（固定）	k8s replicaset	通过实例模版设置实例数
应用版本升级	更改镜像文件	更改实例模版
启动应用	启动业务容器	按照**实例模版**启动虚拟机+应用
创建应用	镜像文件	**自动启动脚本**
操作系统	镜像文件的基础镜像	实例模版的操作系统配置
硬件	容器的硬件配置	实例模版的硬件配置

Traffic Director：效仿k8s/Istio的方式来管理服务

	k8s	Traffic Director
服务调用	流量劫持+ Sidecar（envoy）	流量劫持+ Sidecar（envoy）
灰度发布	通过Istio支持	通过托管式实例组支持
服务发现	通过Istio Pilot 下发（xDS协议）	通过 Traffic Director（xDS协议）
健康检查	k8s进行健康检查	通过托管式实例组配置健康检查
服务注册	通过k8s自动注册	手工关联服务和托管实例组
创建服务	通过 k8s	在控制台手工创建
服务定义	k8s service	Google Traffic Director 服务

创新思路：补齐虚拟机的短板，向容器看齐，维持一致的用户体验

Traffic Director 将对虚拟机的支持提升到新的高度

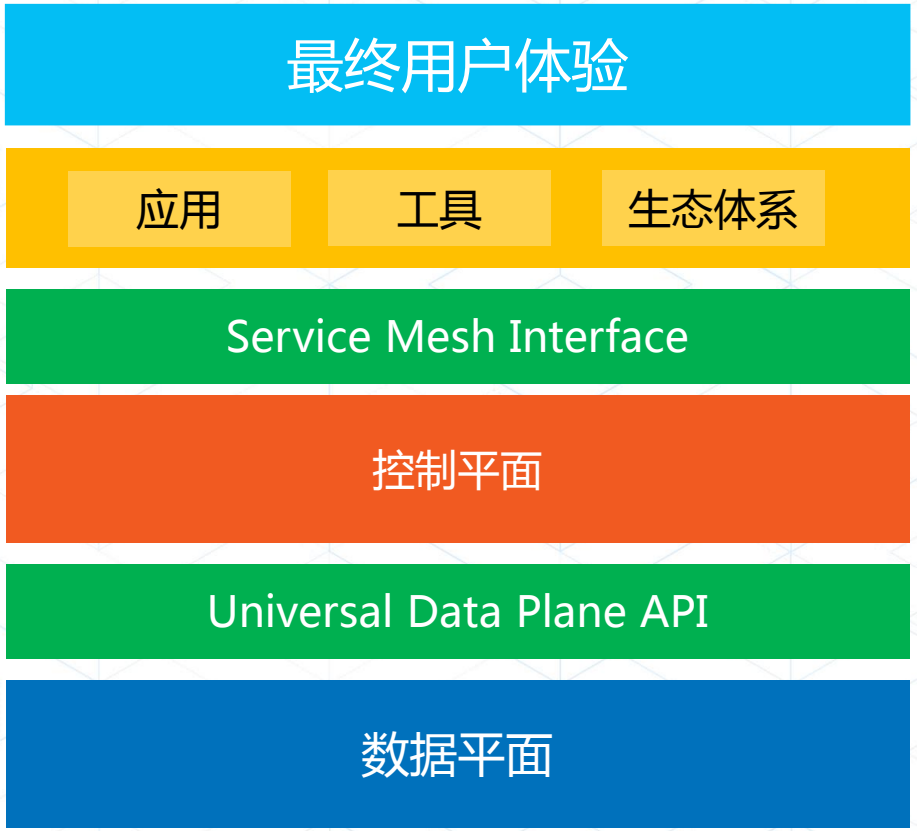
@Google：

说好的供应商不锁定呢?

SMI带来的美好愿景

“SMI 是在 Kubernetes 上运行服务网格的规范。它定义了由各种供应商实现的通用标准。这使得最终用户的标准化和服务网格供应商的创新可以两全其美。SMI 实现了灵活性和互操作性。”

“SMI API的目标是提供一组通用的，可移植的Service Mesh API，Kubernetes用户可以以供应商无关的方式使用这些API。通过这种方式，可以定义使用Service Mesh技术的应用程序，而无需紧密绑定到任何特定实现。”



Linkerd 2.4.0: 开始支持SMI

Linkerd 2.4.0: 发布于2019-07-11

This release adds traffic splitting functionality, support for the Kubernetes Service Mesh Interface (SMI), graduates high-availability support out of experimental status, and adds a tremendous list of other improvements, performance enhancements, and bug fixes.

Linkerd’s new traffic splitting feature allows users to dynamically control the percentage of traffic destined for a service. This powerful feature can be used to implement rollout strategies like canary releases and blue-green deploys. Support for the Service Mesh Interface (SMI) makes it easier for ecosystem tools to work across all service mesh implementations.

Google在ServiceMesh标准化上表现异常：而标准化是供应商不锁定的基石

SMI出来之前：

- Istio迟迟未贡献给CNCF
- Istio API 是私有API，未见有标准化动作
- Envoy xDS v2 API是社区事实标准，但这是Envoy
- 统一数据平面API，感觉更像是Envoy在推动

SMI出来之后：

- ServiceMesh玩家除Istio之外几乎都参与
- 未见Istio有积极回应

AWS：我原以为只有我这长相的会背叛革命，想不到你这浓眉大眼的家伙也背叛革命了。

SMI阵营：汇集几乎所有玩家，唯独AWS和Google缺席

Service Mesh 出道四年，后面会怎么发展

路往何方？

灵魂拷问

要架构，
还是要
性能？

落地，落地，落地

性能有了，
架构
怎么办？

优化，创新

每一个问题和答案，
都会深刻影响未来几
年Servicemesh的
走向，请密切关注

要不要
支持
虚拟机？

落地，创新

说好的
供应商
不锁定呢？

标准化，生态共建

关注 **ServiceMesher** 微信公众号
获取社区最新信息

关注 金融级分布式架构 微信公众号
获取 **SOFAStack** 最新信息

ServiceMesher 社区是由一群拥有相同价值观和理念的志愿者们共同发起，
于 2018 年 4 月正式成立，致力于成为 Service Mesh 技术在中国的布道者和领航者。
社区官网：https://www.servicemesher.com

ServiceMesher