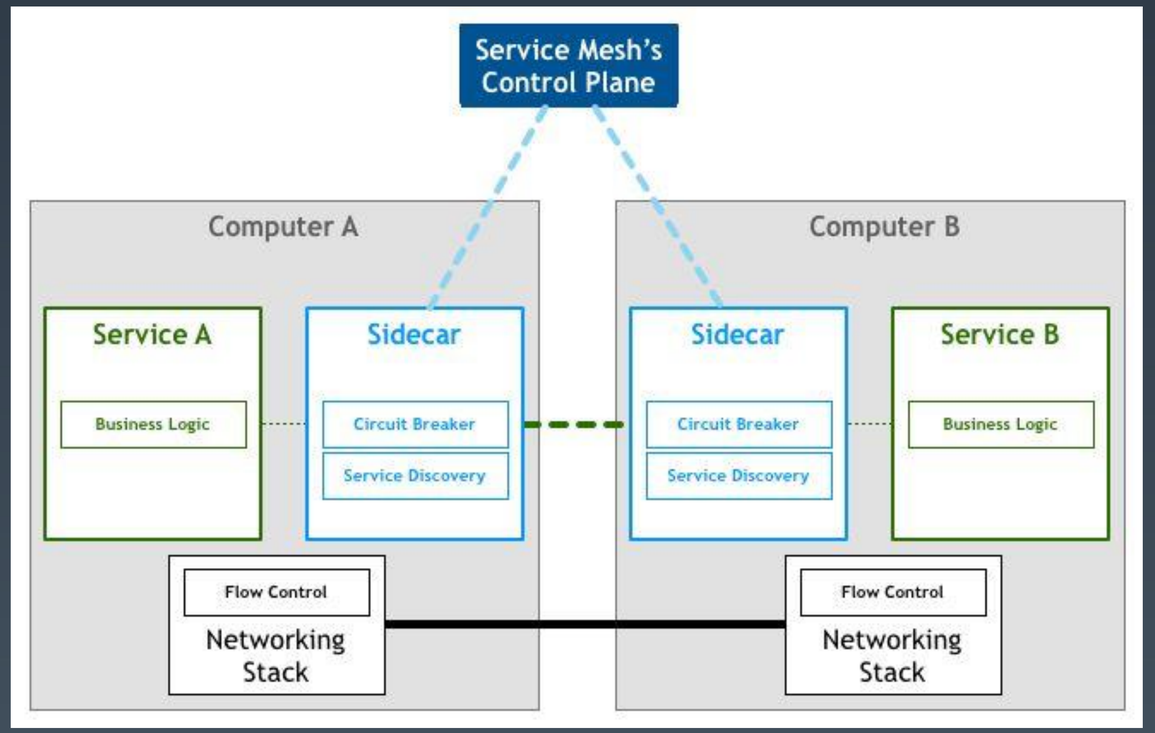
Proxyless Service Mesh

毛剑

Service Mesh 的优势和困局

[1] Service Mesh 提供了微服务化开发的新思路，核心思想是构建一个代理转发网络并结合控制和转发分离的做法来对成千上万个微服务间做流量、策略、安全等管理，利于整体统一管控和升级。在 Service Mesh 出现之前，包括我们当前都是基于 SDK 来治理微服务（部分中间件我们是以 Proxy 形式存在，即 Mesh 化）。



许多企业引入的原因，总结下来：

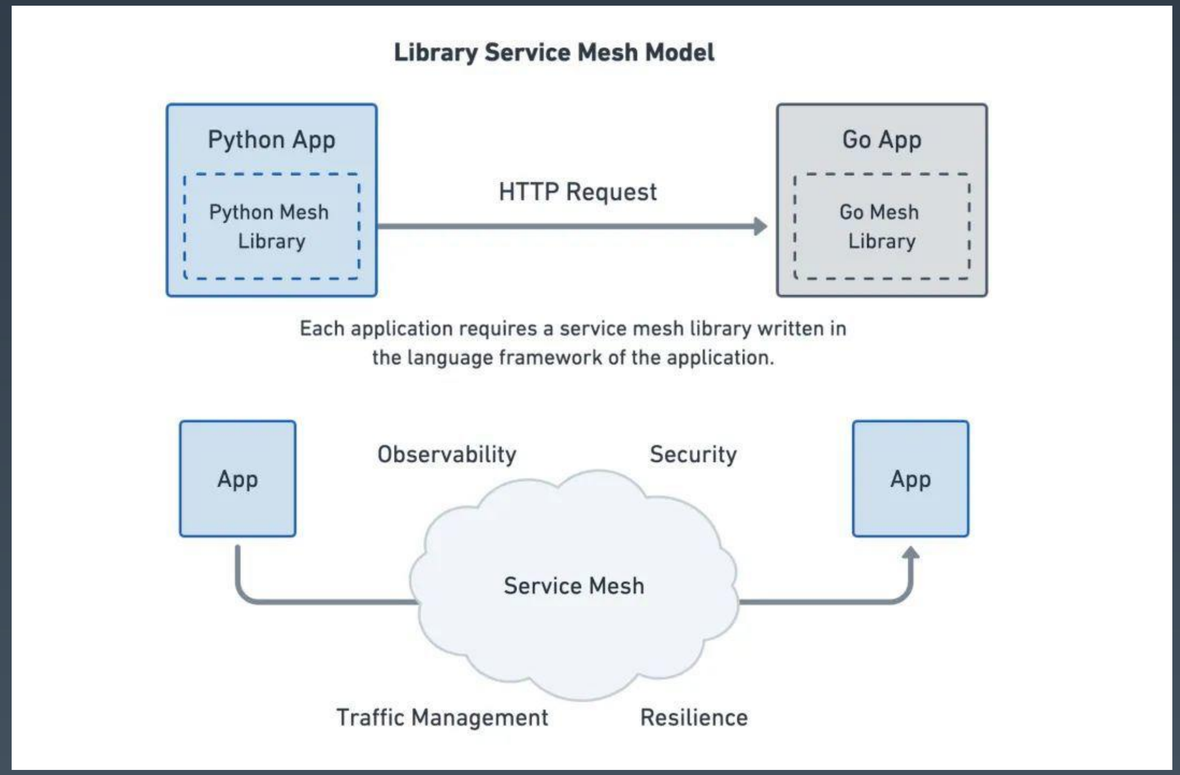
- *多语言带来的 SDK 开发门槛；*
- *SDK 版本升级和兼容性；*
- *历史技术债务导致缺乏服务治理能力；*

种种复杂原因催生了服务之间通讯层抽象，它不和应用代码耦合。把完整的治理能力下沉到 Proxy 中。

Service Mesh 的优势和困局

纵观今天服务网格的功能，可以总结几点：

- *弹性连接：服务与服务之间的通信必须能够跨越边界，如云、集群和场所。通信必须是有弹性的和容错的；*
- *L7 流量管理：支持多种通信协议（HTTP、gRPC、Thrift、WebSocket 等），以及在 Endpoint 之间的负载均衡策略。针对流量的速率控制；*
- *安全性：发送和接收服务都必须能够根据身份验证对方；*
- *可观察性和跟踪：追踪和指标形式的可观察性对于理解、监控和排除应用程序的稳定性、性能和可用性至关重要；*
- *透明：该功能必须以透明的方式提供给应用程序，即不需要改变应用程序代码；*

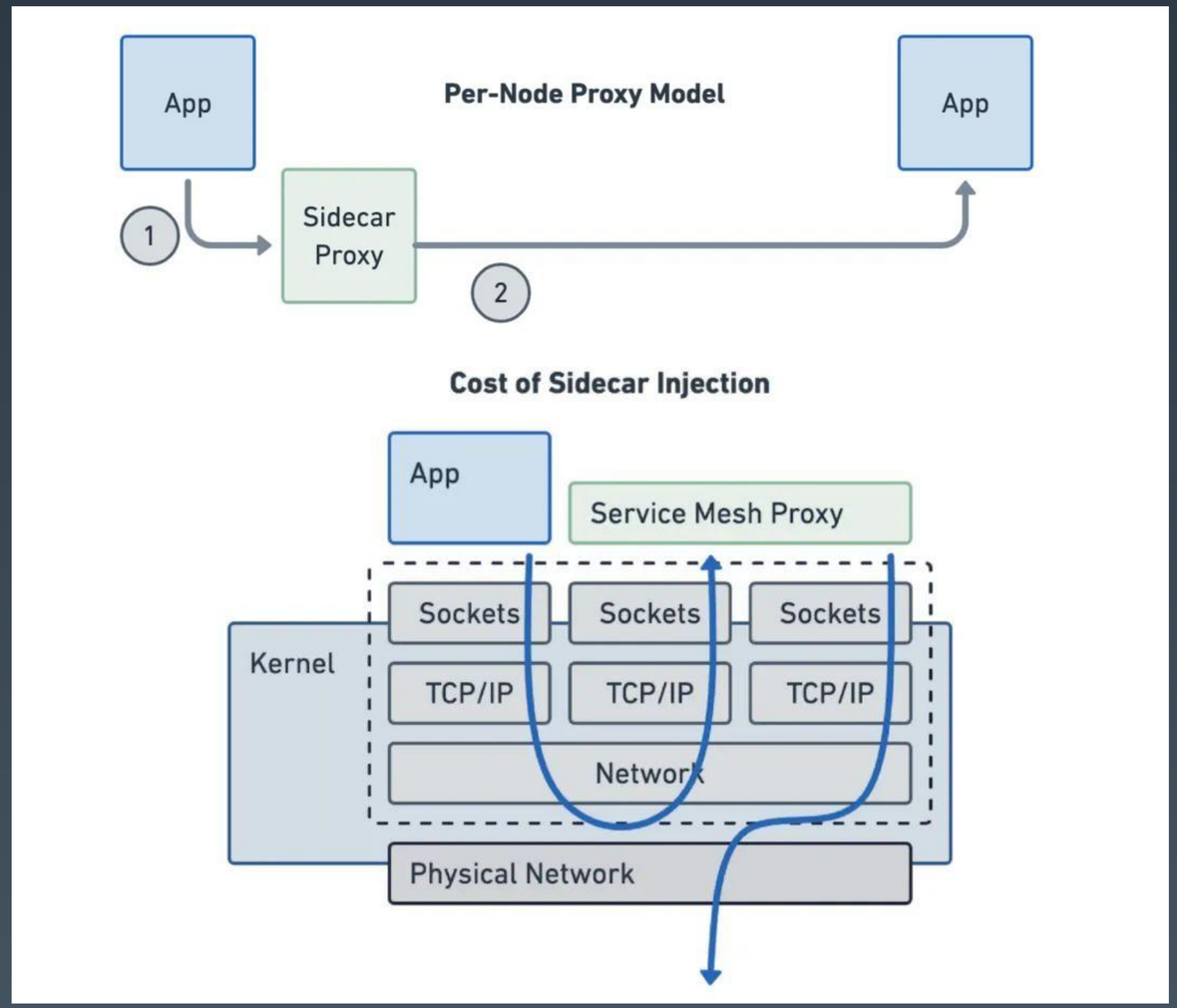


Service Mesh 的优势和困局

Kubernetes 服务网格解决方案要求你在每一个应用 pod 上添加一个代理 sidecar 容器 [2]，如 Envoy [3] 或 Linkerd-proxy [4]。这是正确的：即使在一个非常小的环境中，比如说有 20 个服务，每个服务运行五个 pod，分布在三个节点上，你也有 100 个代理容器。无论代理的实现多么小和有效，这种纯粹的重复都会耗费资源。

Latency & Cost：

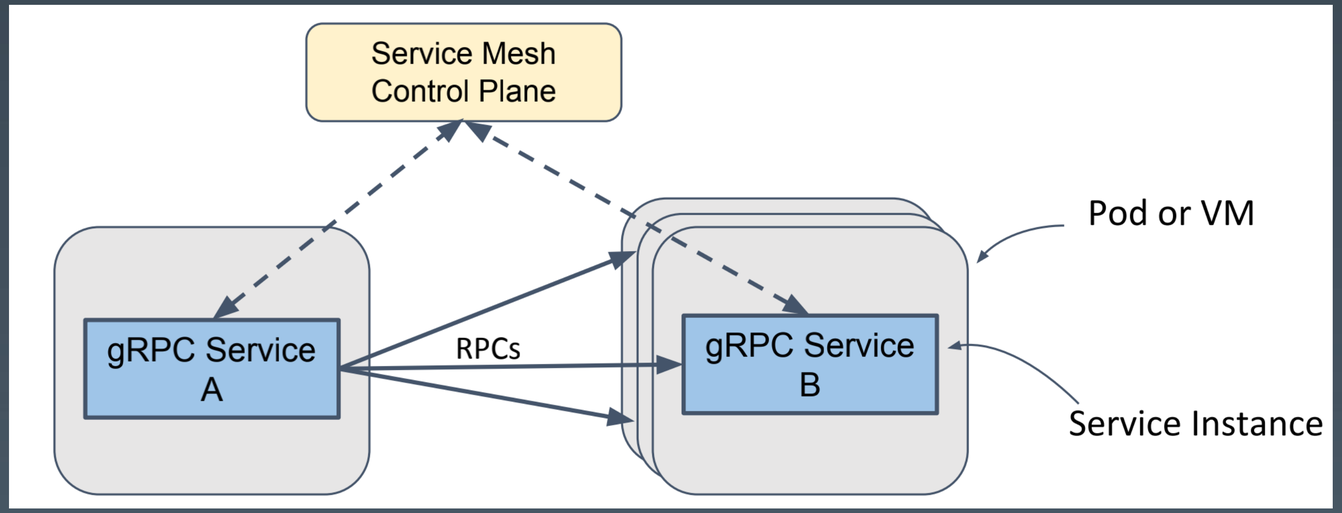
- *[5] Proxy Kernel Performance Improve;*
- *Pre-Node Proxy Model: Both -> Client Side ;*
- *[6] Kubernetes VPA;*
- *[7] eBPF;*
- *Proxyless Service Mesh;*

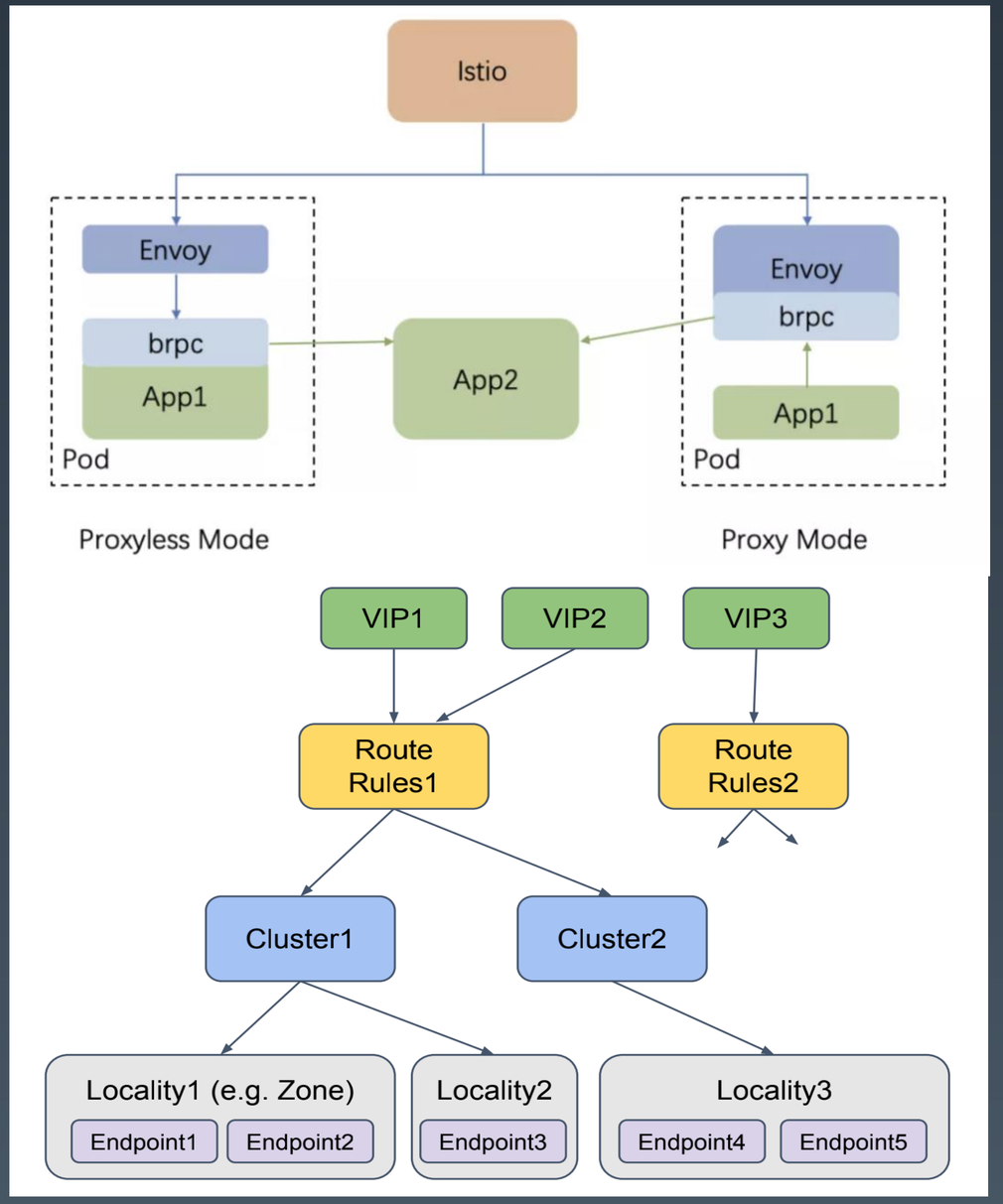


基于 xDS 的智能客户端

xDS 是一类发现服务的总称，包含 LDS， RDS，CDS， EDS 以及SDS。 Envoy 通过 xDS API 可以动态获取 Listener（监听器），Route（路由）， Cluster（集群）， Endpoint（集群成员）以及Secret（证书）配置。

Envoy 通过查询文件或管理服务器来动态发现资源。概括地讲，对应的发现服务及其相应的 API 被称作 xDS。Envoy 通过订阅（subscription）方式来获取资源，如监控指定路径下的文件、启动 gRPC 流或轮询 REST-JSON URL。[8]



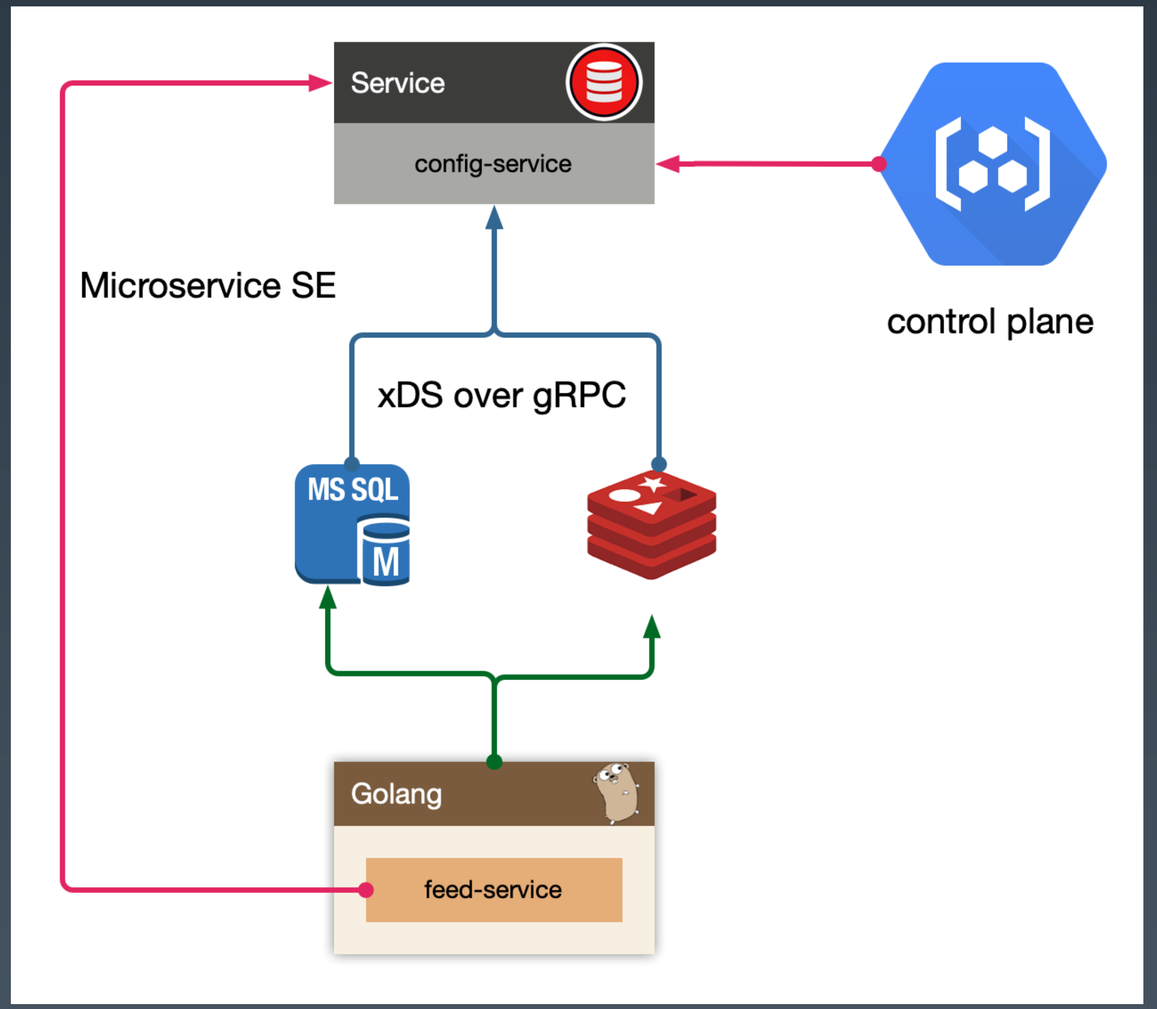


基于 xds 的智能客户端

一个业务应用通过暴露 Endpoint 具备服务化能力，同时一个业务应用本身依赖基础设施来完成读写数据，例如：消息队列、缓存、数据库等。我们如果具备了流量 gRPC 的治理能力，同样的中间件也需要治理能力。一个完整的应用由如下部分组成：

- *Dataset；*
- *Configuration；*
- *Application；*

过去我们依托配置中心读写配置，依赖 MySQL、Redis-Cluster 等完成读写的读写和加速。把整个治理层面从流量扩展到所有组件，统一管理。



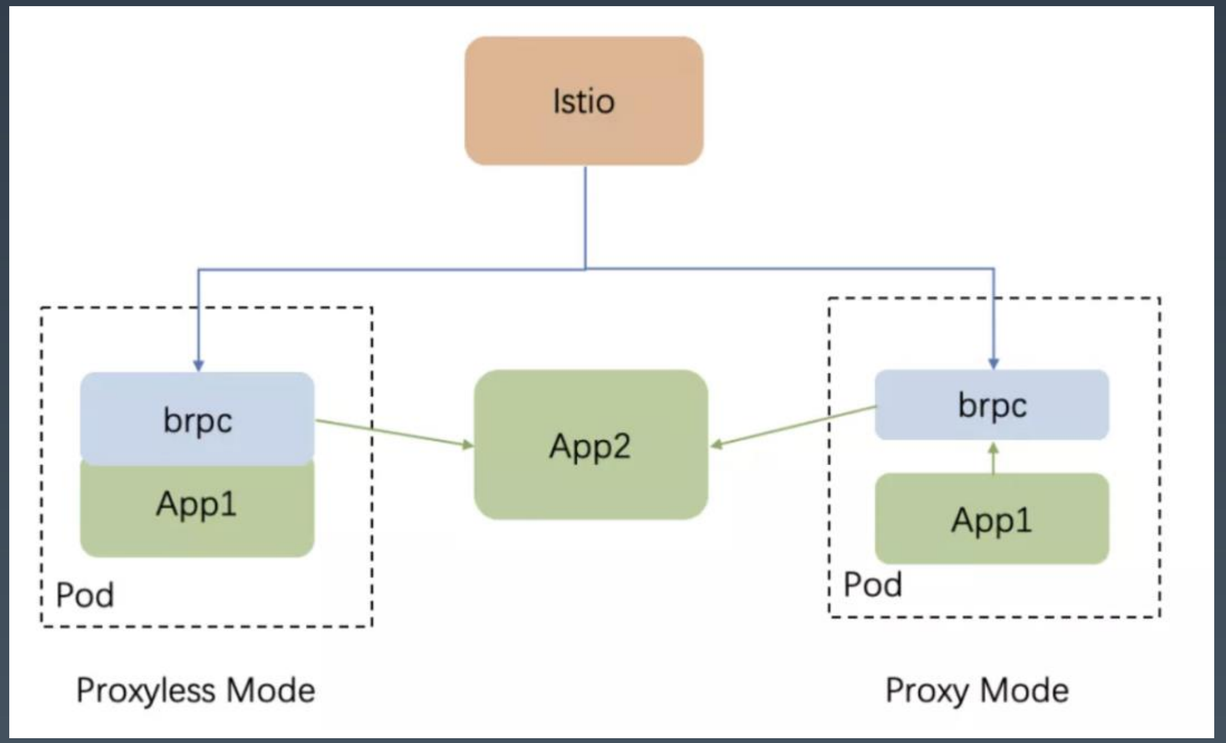
xds 带来的标准化思考

gRPC 通过标准化解决了跨语言的问题，即遵循同一套标准和协议。同样的 xds 基于一套标准协议，把控制面协议标准化了，可以自己基于协议实现通信管控的 API。这样最大的优势在于一套标准处处实现，所有的代码都可以在 Proxy 和 Proxyless 模式下复用。

从企业业务场景上来说：

- *大流量型业务必然会具备复杂且可靠的治理能力，治理能力趋于稳定，标准化解决新老应用的兼容问题；*
- *历史债务业务，通过 proxy 模式可以具备治理能力，参考2-8原则；*
- *需要 new feature 的业务一定乐意尽快升级；*

从人性层面看到基础设施（框架）升级。



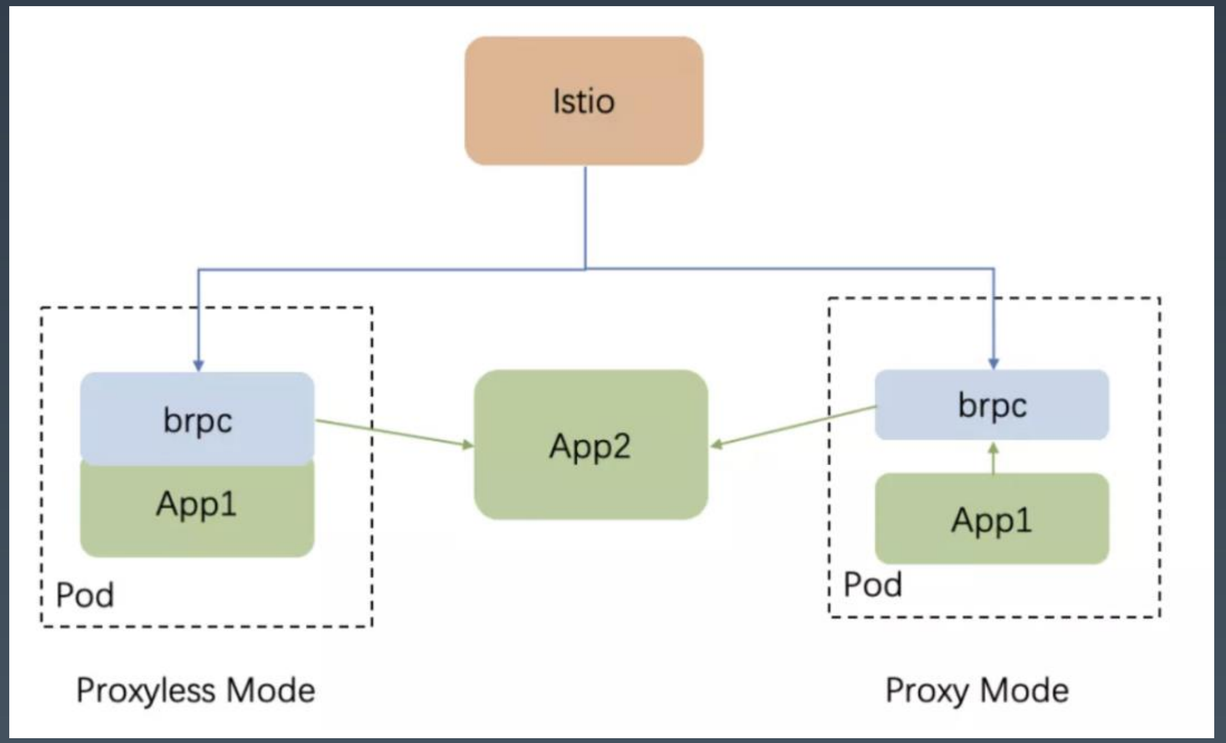
xds 带来的标准化思考

gRPC 通过标准化解决了跨语言的问题，即遵循同一套标准和协议。同样的 xds 基于一套标准协议，把控制面协议标准化了，可以自己基于协议实现通信管控的 API。这样最大的优势在于一套标准处处实现，所有的代码都可以在 Proxy 和 Proxyless 模式下复用。

从企业业务场景上来说：

- *大流量型业务必然会具备复杂且可靠的治理能力，治理能力趋于稳定，标准化解决新老应用的兼容问题；*
- *历史债务业务，通过 proxy 模式可以具备治理能力，参考2-8原则；*
- *需要 new feature 的业务一定乐意尽快升级；*

从人性层面看待基础设施（框架）升级。

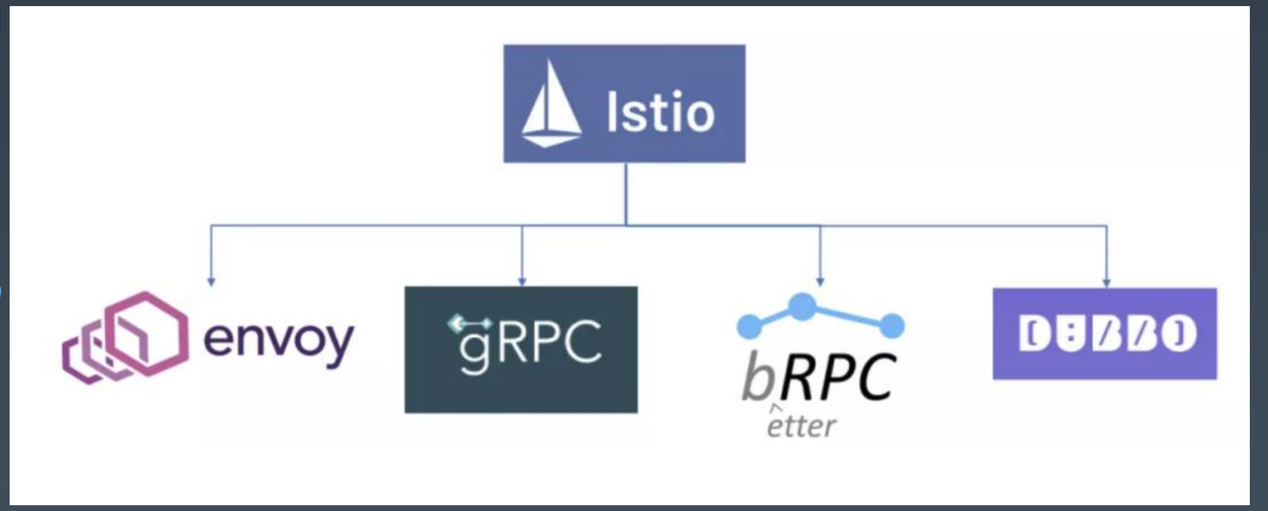


xds 带来的标准化思考

无论参考 Google 代码库的 Depend on Head（stable）还是基于 Mono-repo 的大仓强更新兼容性问题，都可以保证在兼顾兼容的情况下强制升级（部分场景要考虑 feature toggle/flag 控制风险）。通过容器编排平台，内部的 PaaS 走一轮可控的 Rolling Update，也可以完成整体业务的灰度和升级。

从目前趋势看来，Istio 仍然会作为 Service Mesh 控制平面的首选。Envoy 仍然会作为 Proxy 模式的主流选择，但各 RPC 框架也不甘于只做瘦客户端，将会继续发展自己的 Proxyless 方案，让 Service Mesh 能落地到更多的业务场景之中。

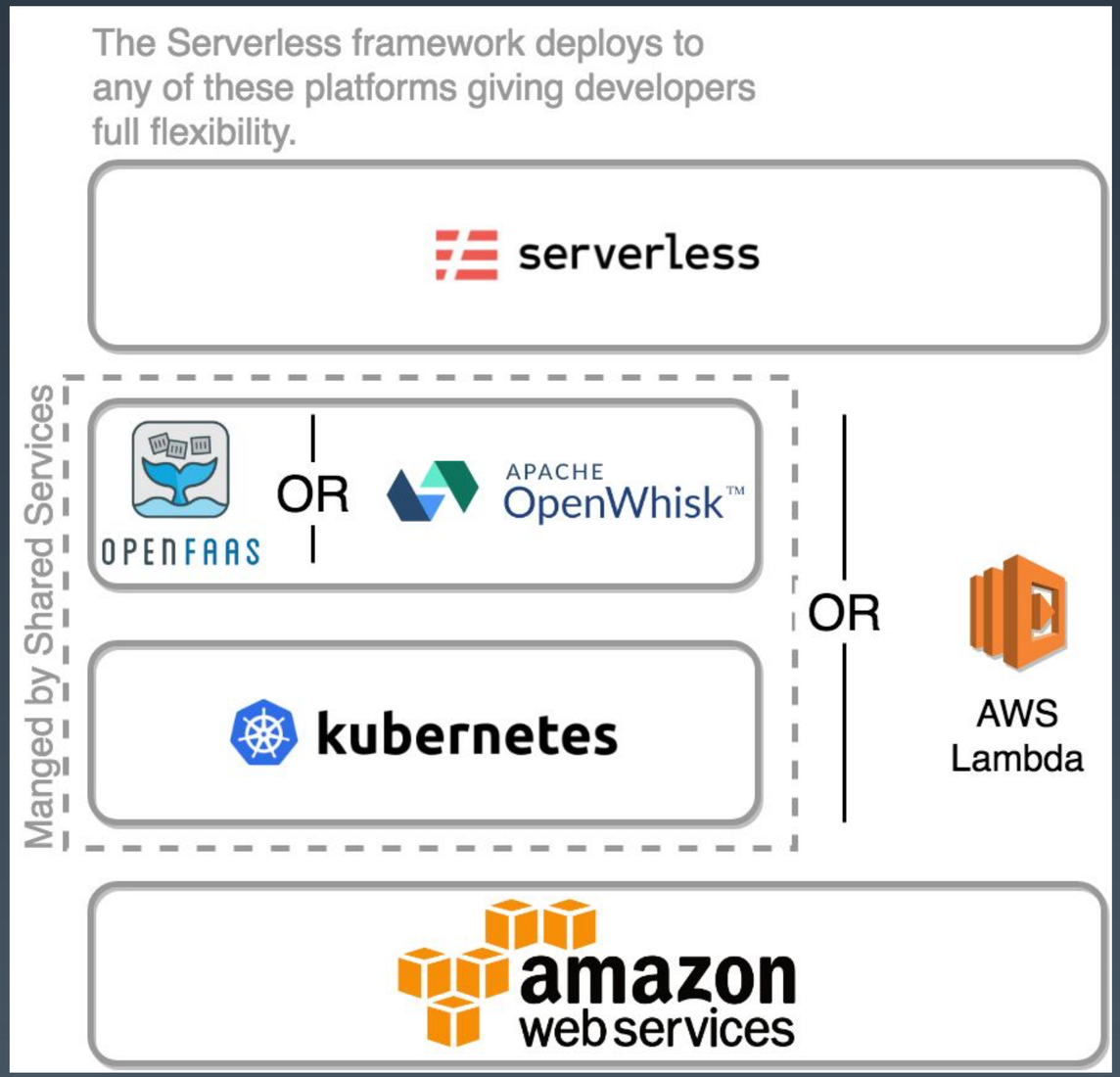
xds 实际上是定义了 Service Mesh 的能力标准。



xds 带来的标准化思考

我感觉 Proxyless Service Mesh 的终极形态类似 FaaS，或者叫 App Engine。

用户只需要编写业务代码，通过代码的接口契约来进行开发。在编译层面注入各类实现。后续只需要升级 App Engine Runtime，做一次 Rolling Update 即可更新能力。



References

[1] 告别 Sidecar - 使用 eBPF 解锁内核级服务网格
https://cloud.tencent.com/developer/article/1917877

[2] eBPF 如何简化服务网格 https://cloud.tencent.com/developer/article/1895605

[3] Envoy https://www.envoyproxy.io/

[4] Linkerd-proxy https://linkerd.io/

[5] Proxyless Service Mesh在百度的实践与思考
https://blog.csdn.net/u011537073/article/details/121915218

[6] 基于 k8s 的纵向扩容介绍 VPA 介绍

https://zhuanlan.zhihu.com/p/45646562

[7] How eBPF will solve Service Mesh

https://isovalent.com/blog/post/2021-12-08-ebpf-servicemesh

[8] Proxyless Service Mesh with gRPC

Q&A