



GOTC 2023

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE, INTO THE FUTURE

「GOTC Cloud Native Summit」专场

本期议题：基于 KiteX Proxyless 和 Istio 的云原生微服务实践

胡文 2023年05月28日

CONTENT

目录

01.

Kitex 与 Istio

02.

Kitex Proxyless 实现原理

03.

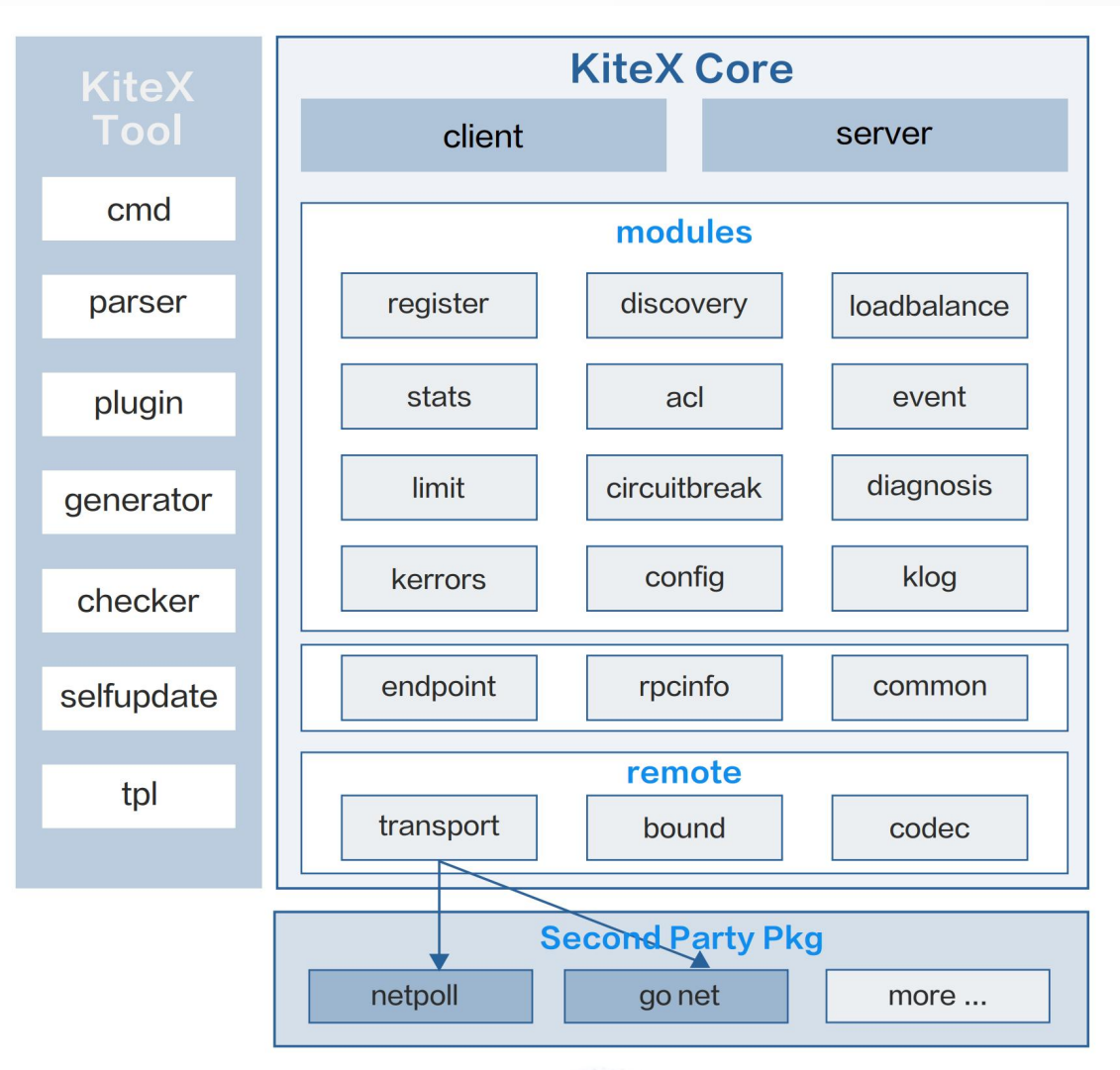
基于 Kitex Proxyless 的全链路泳道实践

01

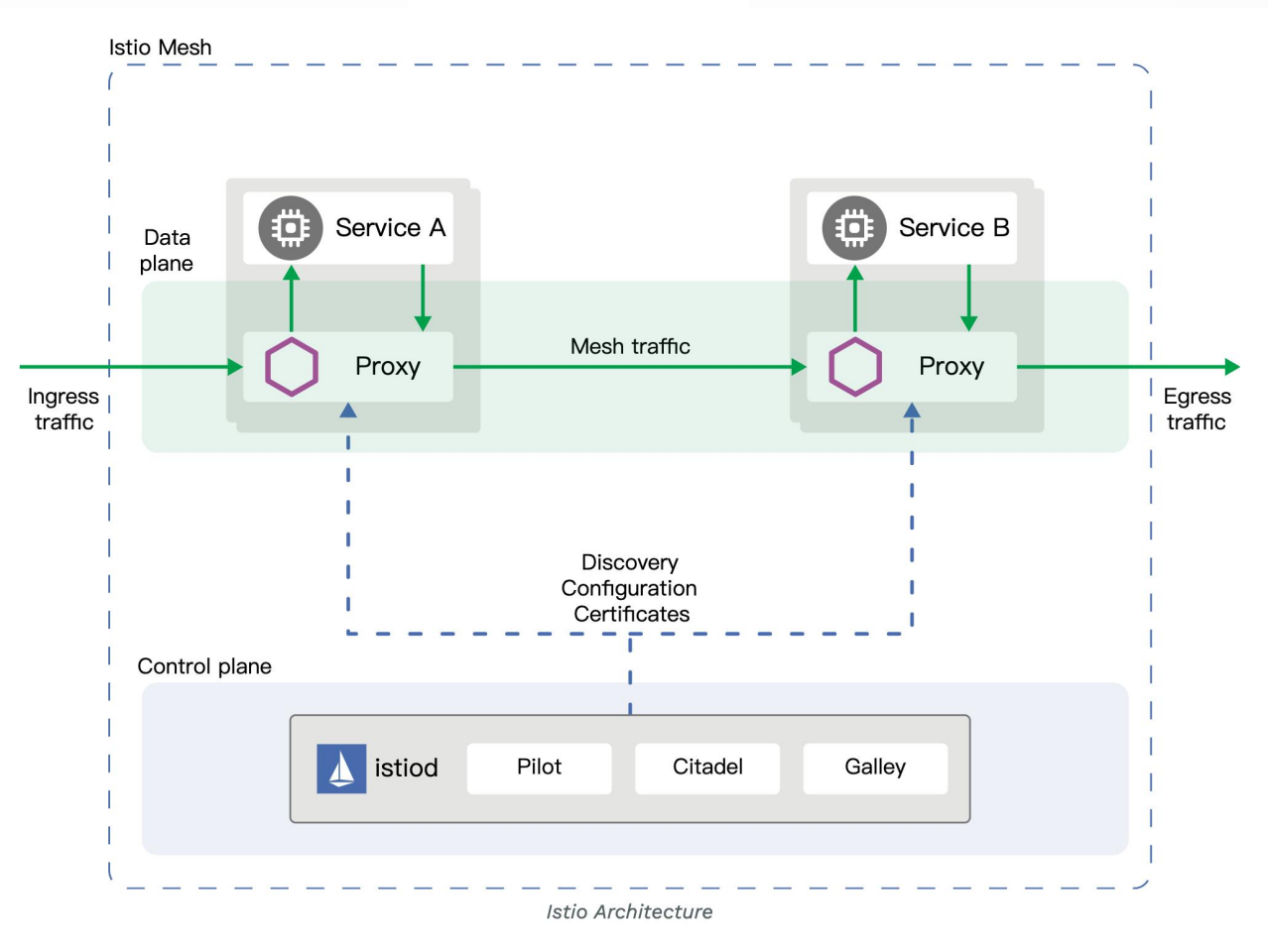
Kitex 与 Istio

Kitex 是字节开源的一款 Golang RPC 框架，具备高性能、高拓展的特点，在字节内部广泛使用。

- 使用自研的高性能网络库 Netpoll，性能相较 go net 具有显著优势
- 提供了较多的扩展接口以及默认扩展实现
- RPC 消息协议默认支持 Thrift、Kitex Protobuf、gRPC
- 支持的传输协议有 THeader、HTTP2
- 支持 PingPong、Oneway、双向 Streaming
- 内置大量服务治理拓展，使用者可按需集成
- 内置代码生成工具，可支持生成 Thrift、Protobuf 以及脚手架代码

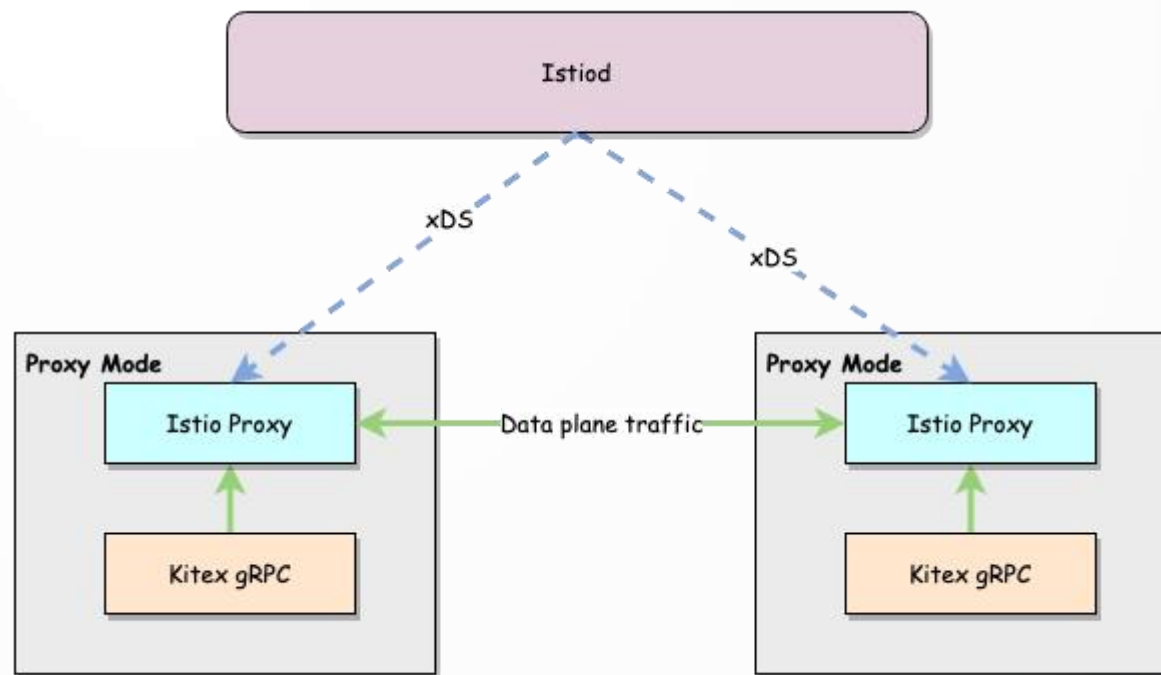


Istio 是一个开源服务网格，它透明地分层到现有的分布式应用程序上。Istio 强大的特性提供了一种统一和更有效的方式来保护、连接和监视服务。Istio 是实现负载平衡、服务到服务身份验证和监视的路径——只需要很少或不需要更改服务代码



Kitex 支持 gRPC 协议，能够以最标准的 Istio Sidecar 模式对接

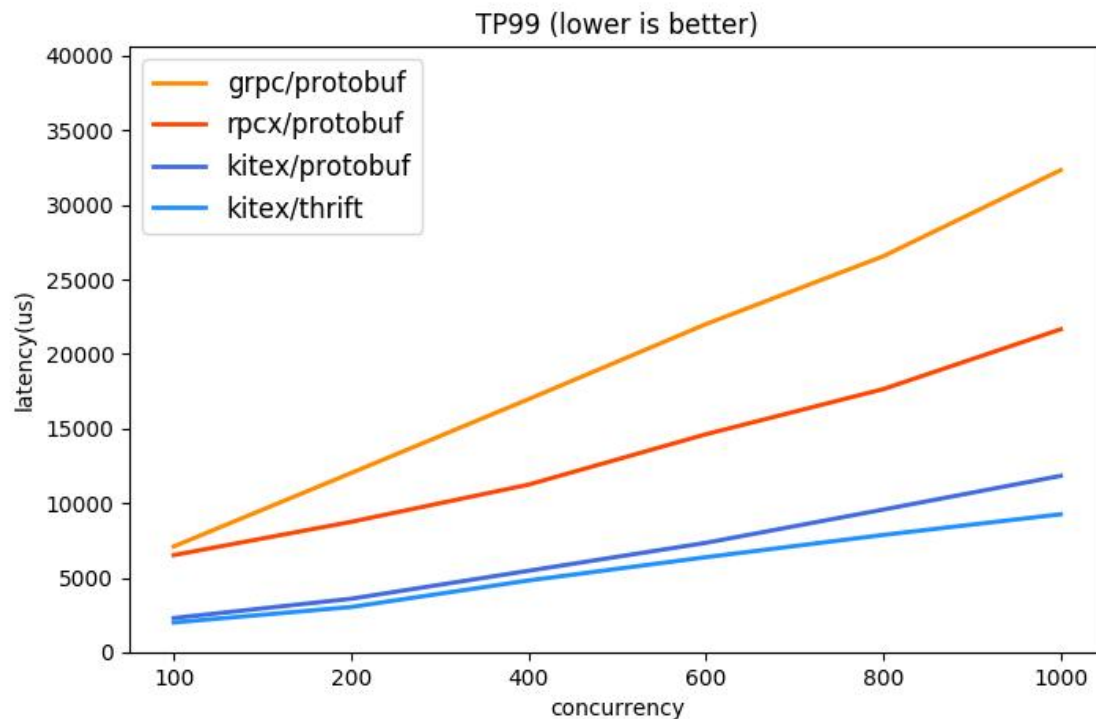
- 标准 Istio 模式，支持的特性比较全面，数据面也能够最大化复用 Envoy 能力
- 业务侵入小
- 支持平滑升级



Kitex 与 Istio — Thrift 协议优化

Kitex 默认采用 Thrift 协议，并在 Thrift 协议上做了很多优化

- 结合自研 netpoll 能力，能够直接暴露底层连接的 buffer
- 基于 SIMD 优化 Thrift 编码
- 使用 inline 来内联函数，减少函数调用栈开销
- 基于 buffer 管理、string / binary 零拷贝、预计算、struct 字段重排等优化，减少内存操作开销
- 开源了高性能 Thrift 编解码器 Frugal，Frugal 具有无需生成代码、高性能（在多核场景下，Frugal 的性能可以达到传统编解码方式的 5 倍）和稳定性等特点



那我们能不能继续保持使用 Thrift 协议来接入网格呢？

Thrift 协议对接 Istio 痛点

- Istio 移除了对 Thrift 的原生支持，需要借助 Envoyfilter 来配置 Thrift 协议的路由规则，使用体验不太佳
- 更为重要的是，传统 Sidecar 模式的资源利用率和性能损耗等通病，在需要追求极致性能的场景可能会尤为凸显，否

则 Kitex-gRPC + Istio 标准 Sidecar 模式可能会是更好的选择。

02 Kitex Proxyless 实现原理

Kitex 直接基于 xDS 协议对接 Istiod 控制面，无需再借助 sidecar 来实现与控制面交互，可以实现 Proxyless 模式的方式纳入到网格中统一管理。

优势

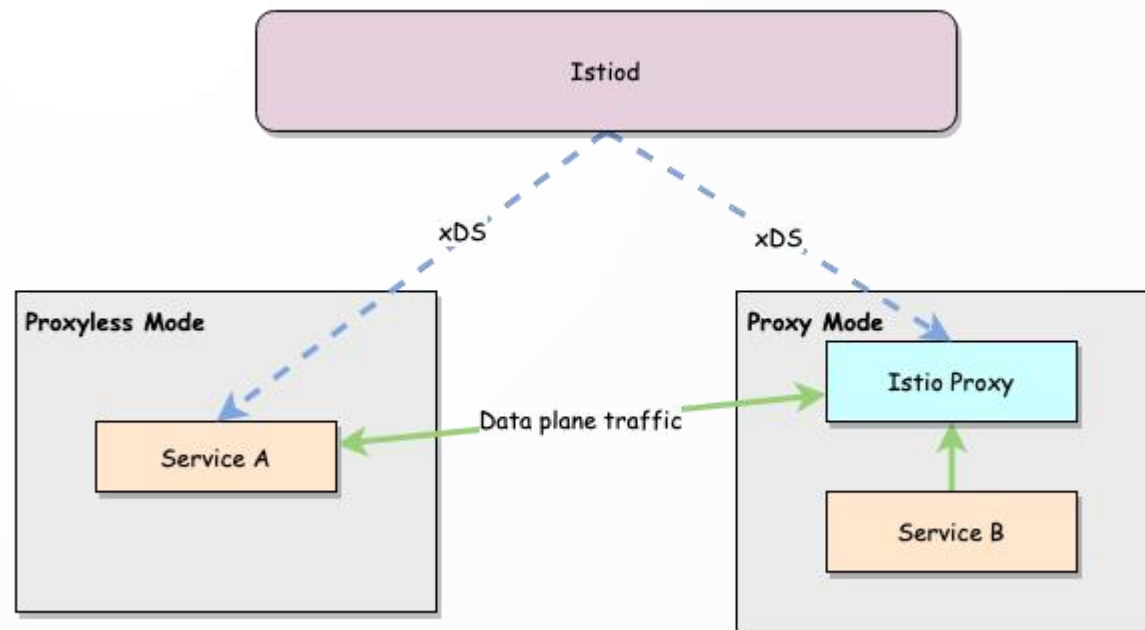
- 能够复用 Thrift 协议，且框架直接通信，避免 Sidecar Proxy 中转损耗，更适合性能敏感场景
- 架构比较简洁，少了 Sidecar 组件本身带来的运维复杂度及其故障率
- 多了一种选择，用户可以灵活根据具体场景来选择 Proxy 模式或者 Proxyless 模式

缺点

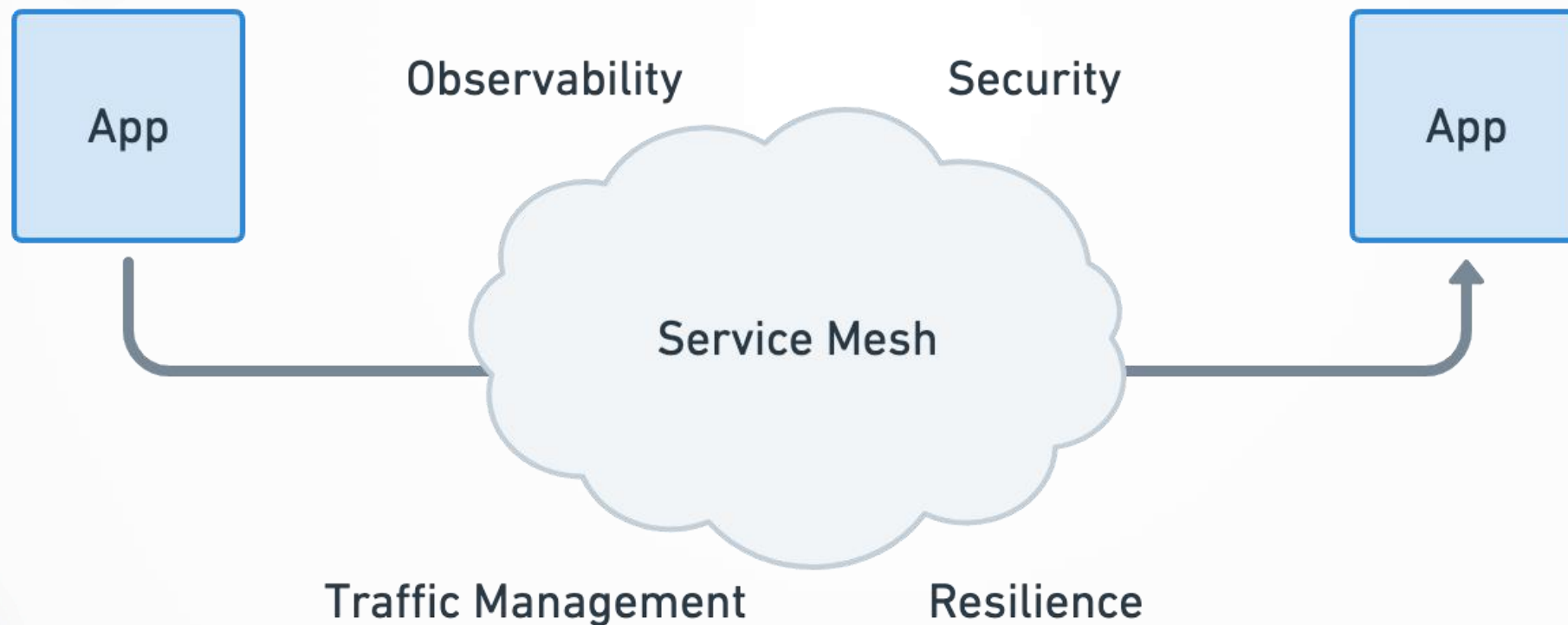
- 多语言支持成本高
- 升级成本较高

疑问

- Proxyless 是不是背离了 ServiceMesh 的初衷？

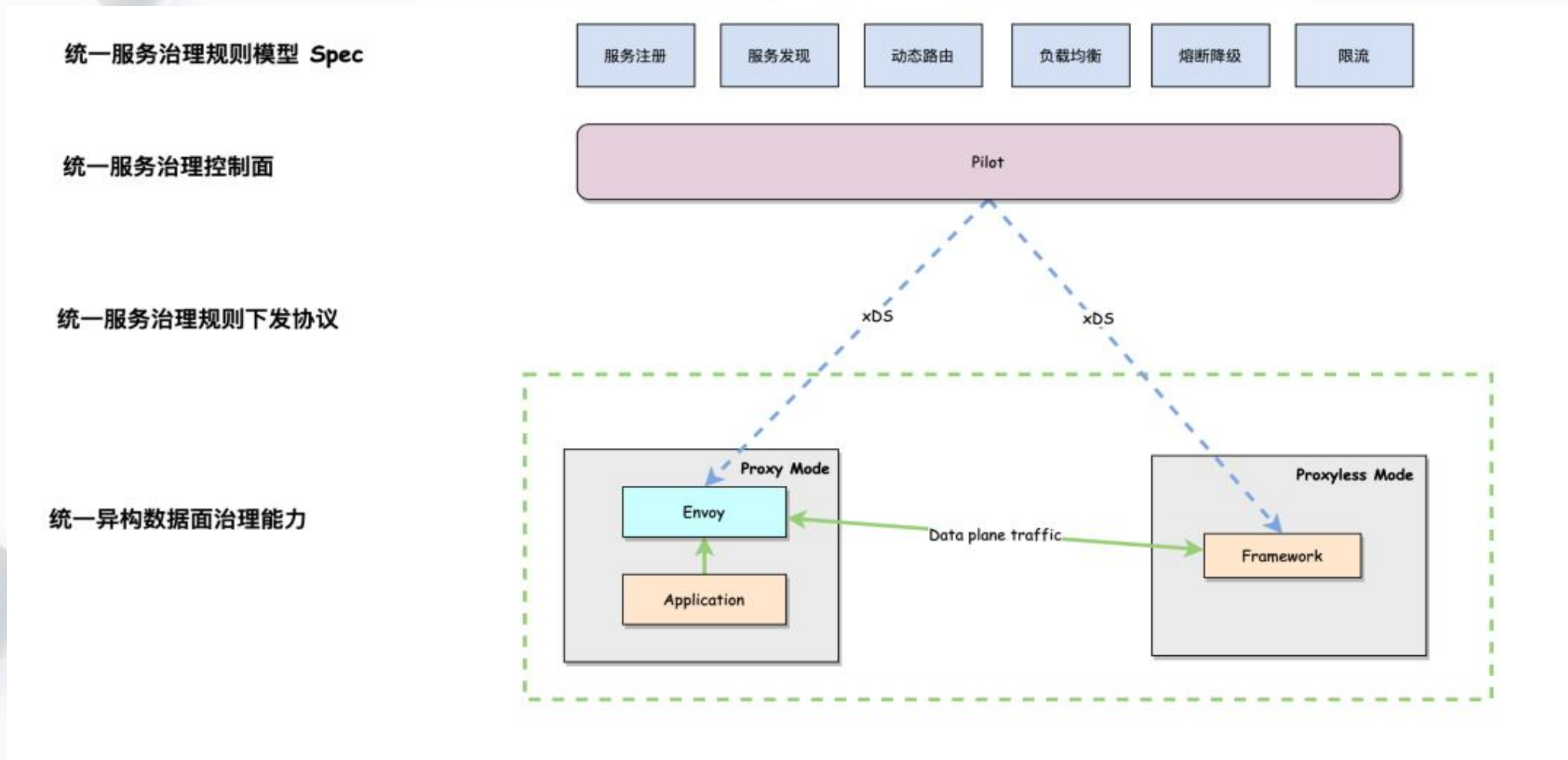


ServiceMesh — 无关模式，服务间通信基础设施标准抽象

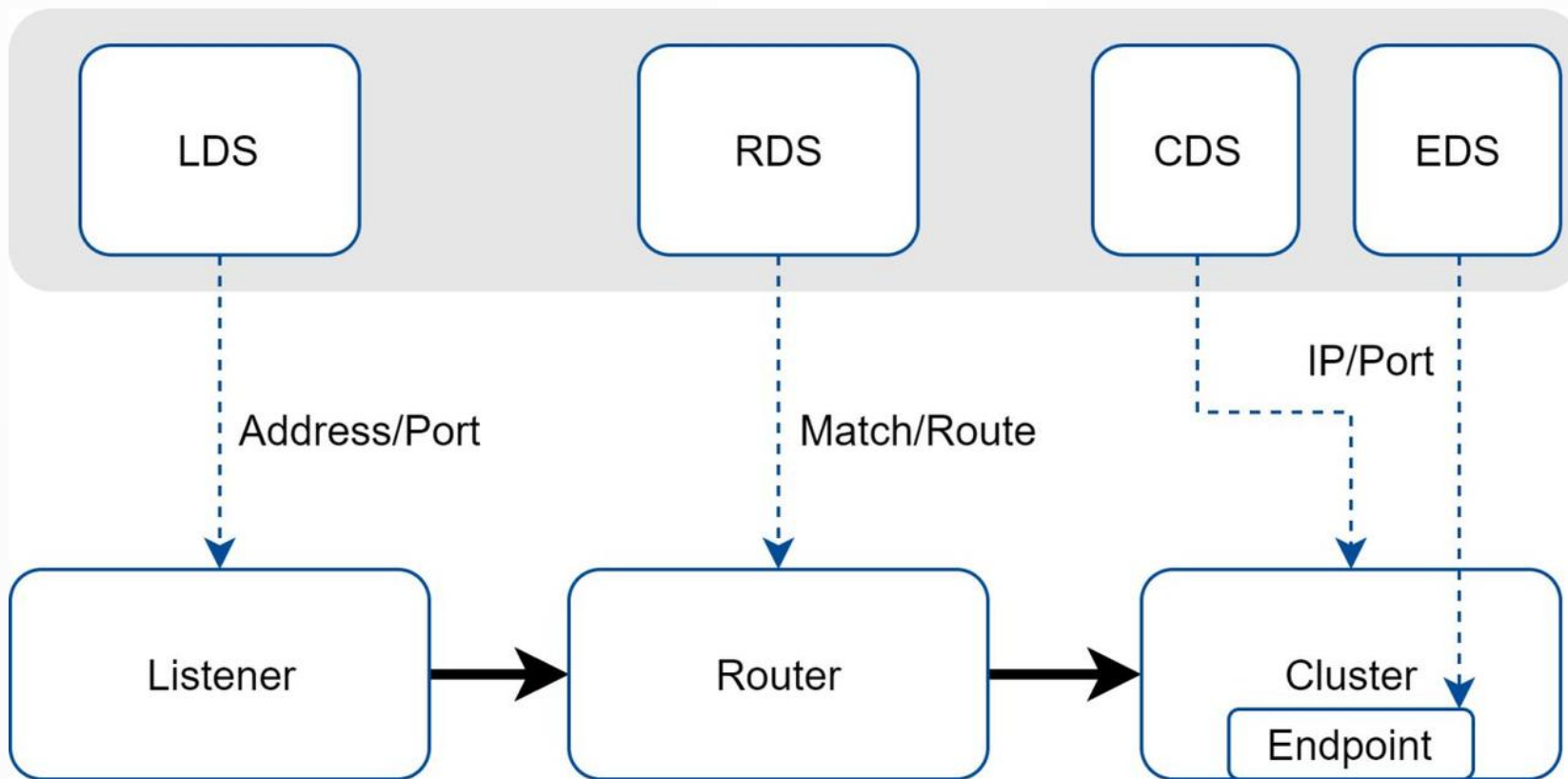


图片来自于: <https://isovalent.com/blog/post/cilium-service-mesh/>

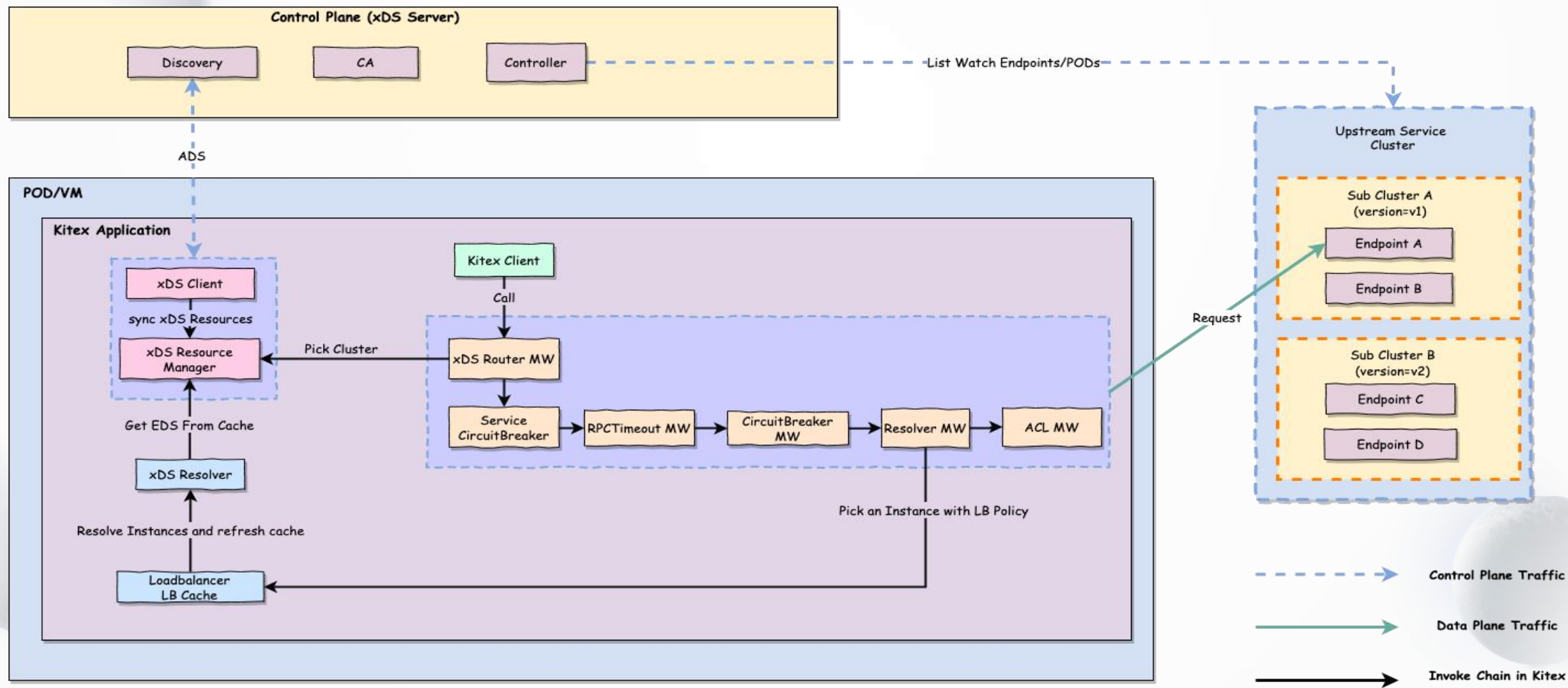
标准统一、兼容开放，在最适合的场景选择最适合的模式即可



xDS 协议是 envoy 制定的一套标准控制面规范。xDS 并非特指某一种数据协议，而是一组不同数据源的发现服务协议的总称



Kitex Proxyless — 整体架构设计



Kitex Proxyless — Kitex 可拓展性

Kitex 具备非常高的可拓展性

- 我们可以通过外部扩展 [kitex-contrib/xds](https://github.com/kitex-contrib/xds) 的形式对 xDS API 进行了支持
- 并且基于 Kitex 提供的拓展接口，适配了流量路由、超时控制、服务发现、负载均衡等治理模块，实现了动态治理规则下发并应用能力

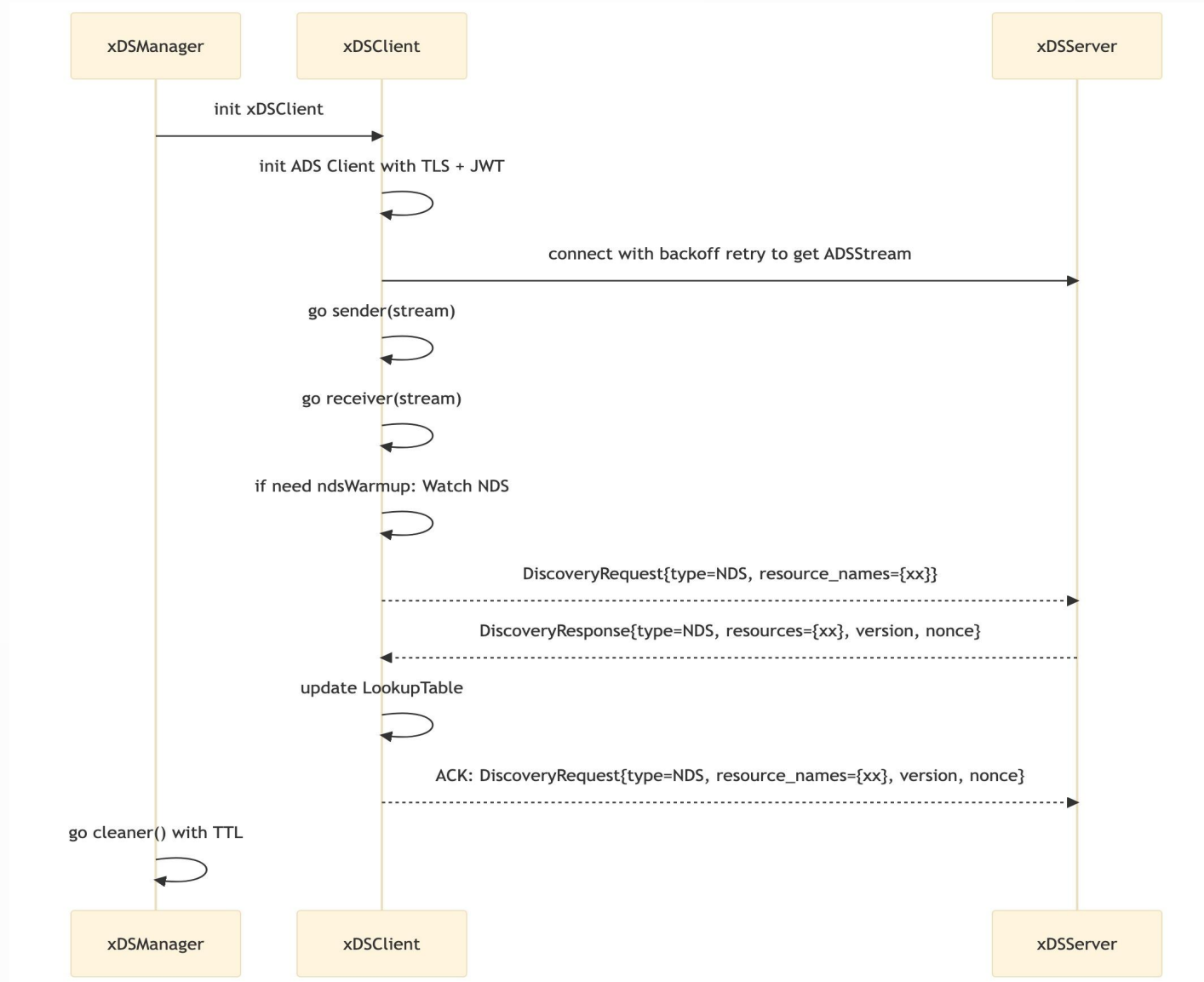
```
if opts.EnableXDS {
    if err := xdsmanager.Init(xdsmanager.WithXDSServerAddress(opts.XDSAddr)); err != nil {
        klog.Fatal(err)
    }
    return reviewsservice.NewClient(
        opts.Endpoint, // use svc fqdn
        kclient.WithSuite(tracing.NewClientSuite()), 开启 opentelemetry
        kclient.WithXDSSuite(xds.ClientSuite{          开启 kitex xds (proxyless)
            RouterMiddleware: xdsuite.NewXDSRouterMiddleware(
                xdsuite.WithRouterMetaExtractor(metadata.ExtractFromPropagator),
            ),
            Resolver: xdsuite.NewXDSResolver(),
        }),
    )
}
```

xDSManager 负责管理 xDS 相关资源缓存

- 支持 TLS 及 JWT 认证
- 基于 goroutine 去实现一个生产消费模型
- 支持基于 backoff 的重连机制

并且值得注意的是，由于我们不在依赖 pilot agent，因此需要自己解决 FQDN -> ClusterIP 的解析问题

针对这个问题，我们的解决方案是基于 NDS (NameTable) 构建了一个本地的 LookupTable，用于 FQDN 解析



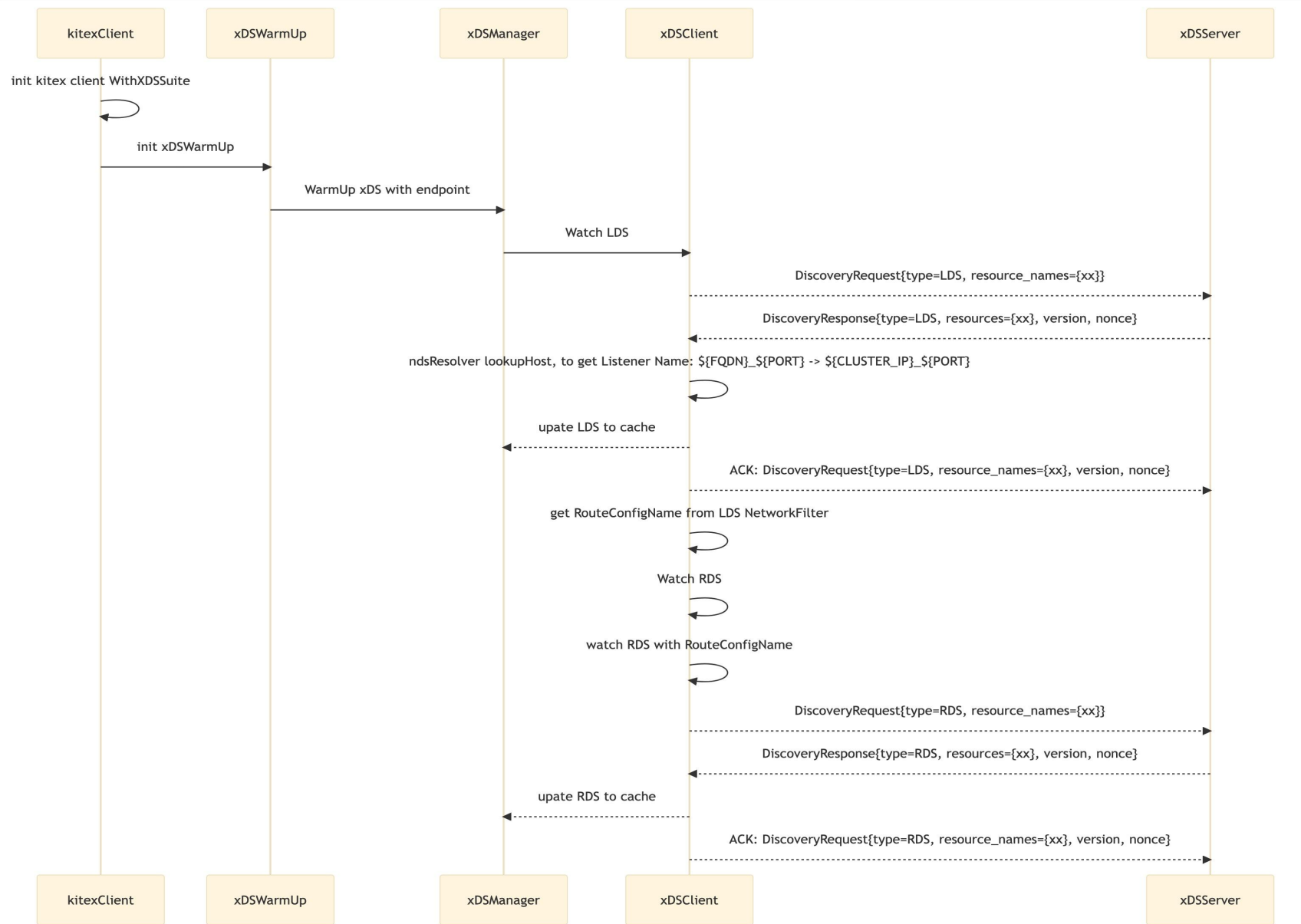
Kitex Proxyless — xDS 资源预热

xDS 资源预热主要用于服务启动能够提前将所依赖服务相关的 xDS 资源缓存在本地，这样请求的时候直接从缓存读取即可

值得注意的是，proxyless 模式已经天然具备懒加载特质，因为实例化 kitex Client 的时候就已经明确声明了依赖的服务

在此基础上，我们资源预热的范围也会非常明确，只需要缓存依赖服务相关的 xDS 资源即可。（这个其实也是 proxyless 模式的优势之一）

另外，在构建 LDS 缓存的时候，我们会基于之前构建的本地 LookupTable 来完成一个 FQDN 到 ClusterIP 的解析。



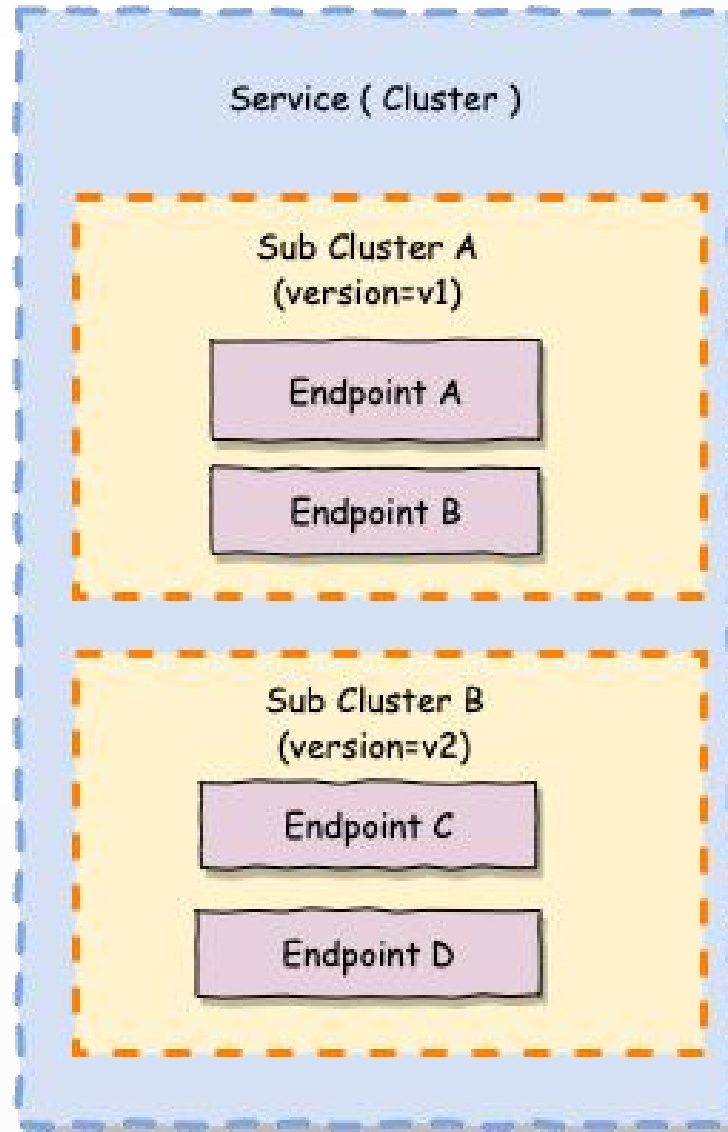
Kitex Proxyless — 服务模型

Istio 的服务模型可以简化理解成右图

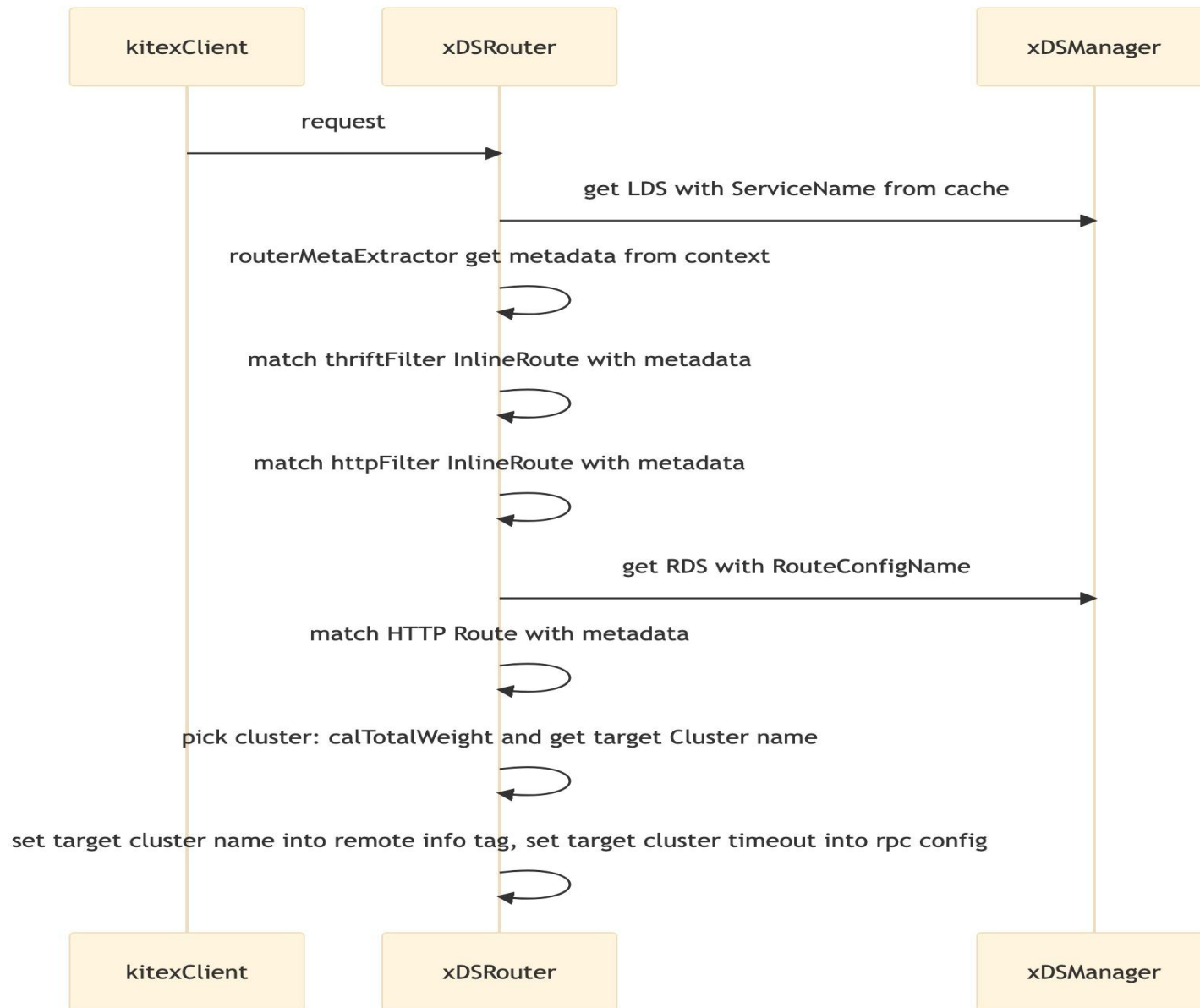
- Service 会作为一个总的 Cluster
- 每个 Service 可以根据 Label 拆分成多个 subset，而每个 subset 都会作为一个独立的 Cluster 存在
- 每个 sub cluster 又是一组 endpoints 的集合，每个 endpoint 代表一个具体的服务实例

可以看到，一个 Service 级别的 Cluster 会包含多个 Sub Cluster

而所谓动态路由，其实就是 Pick Sub Cluster 的过程



1. 增加一个 xDS Router MW 来负责 Pick Cluster (路由), watch 目标服务的 LDS 及 RDS
2. 获取 LDS, 并提取目标服务的 LDS 中的 Filter Chain 及其 inline RDS
3. 获取 RDS, 根据 VirtualHost 和 ServiceName 来匹配 (支持前缀、后缀、精确、通配), 获取目标服务的路由配置
4. 遍历处理匹配到的 RDS 中的路由规则, 路由规则主要分为两部分:
 - 1) Match (支持前缀、后缀、精确、通配等), 目前版本我们支持以下两种即可:
 - Path (必须项): 从 rpcinfo 提取 Method 进行匹配
 - HeaderMatcher (可选项): 从 rpcinfo 中提取 Tag 进行匹配
 - 2) Route:
 - Cluster: 标准 Cluster
 - WeightedClusters (权重路由): MW 内根据权重来选择 cluster
5. 将选择到的 Cluster 写入 EndpointInfo.Tag, 用于之后的服务发现。



服务发现

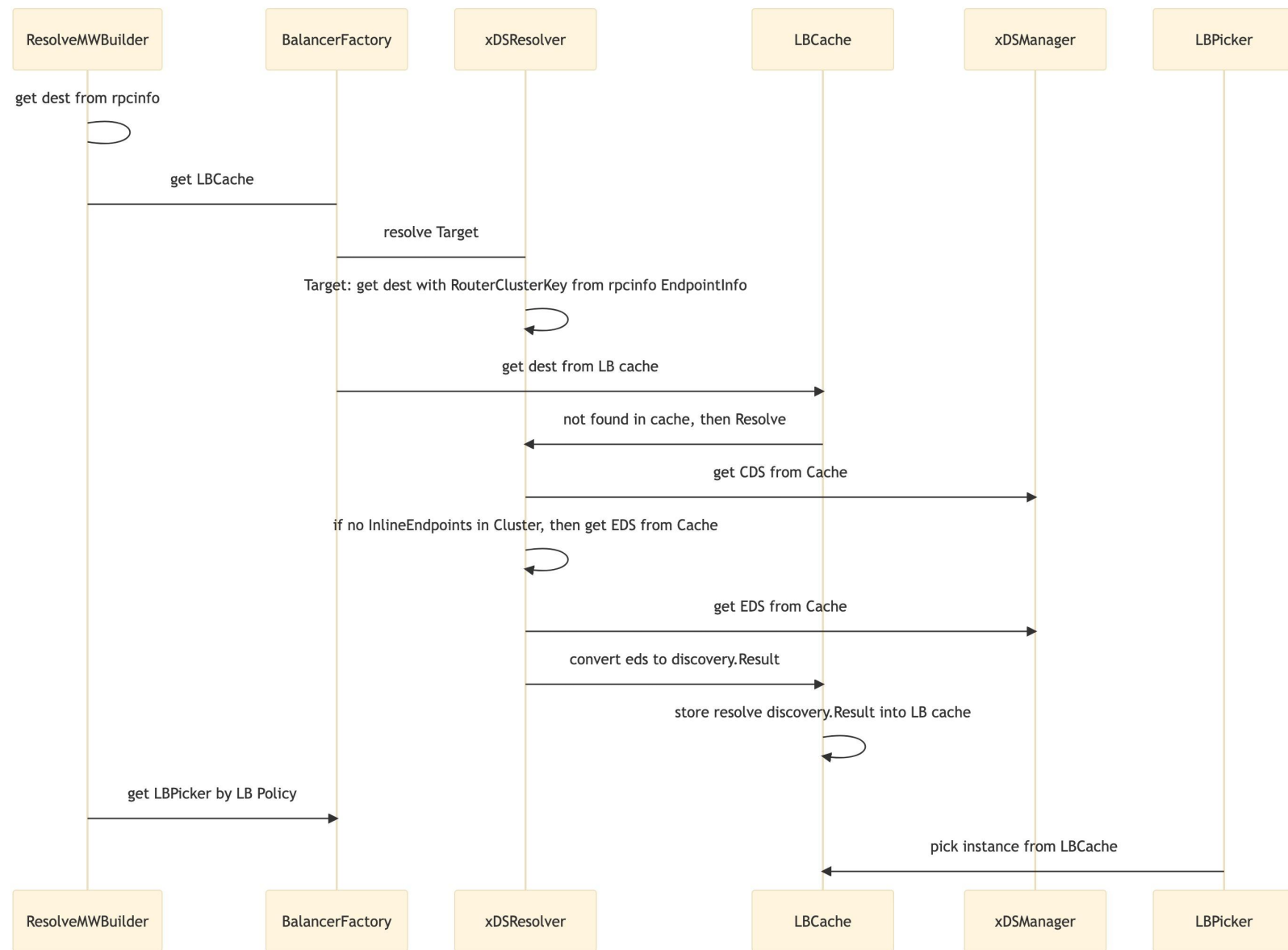
基于 Kitex 的 Resolver 接口，拓展一个 XDSResolver 来进行服务发现。

- 根据 RDS 找到对应的 CDS (Pick Cluster) 之后，进入 Resolve 阶段
- 通过根据 CDS 获取对应的 Cluster 配置，包括负载均衡策略、熔断等
- 之后拉取 EDS 资源以获取实例信息，并根据对应的 LB 策略，pick 出最终的一个 Endpoint

负载均衡

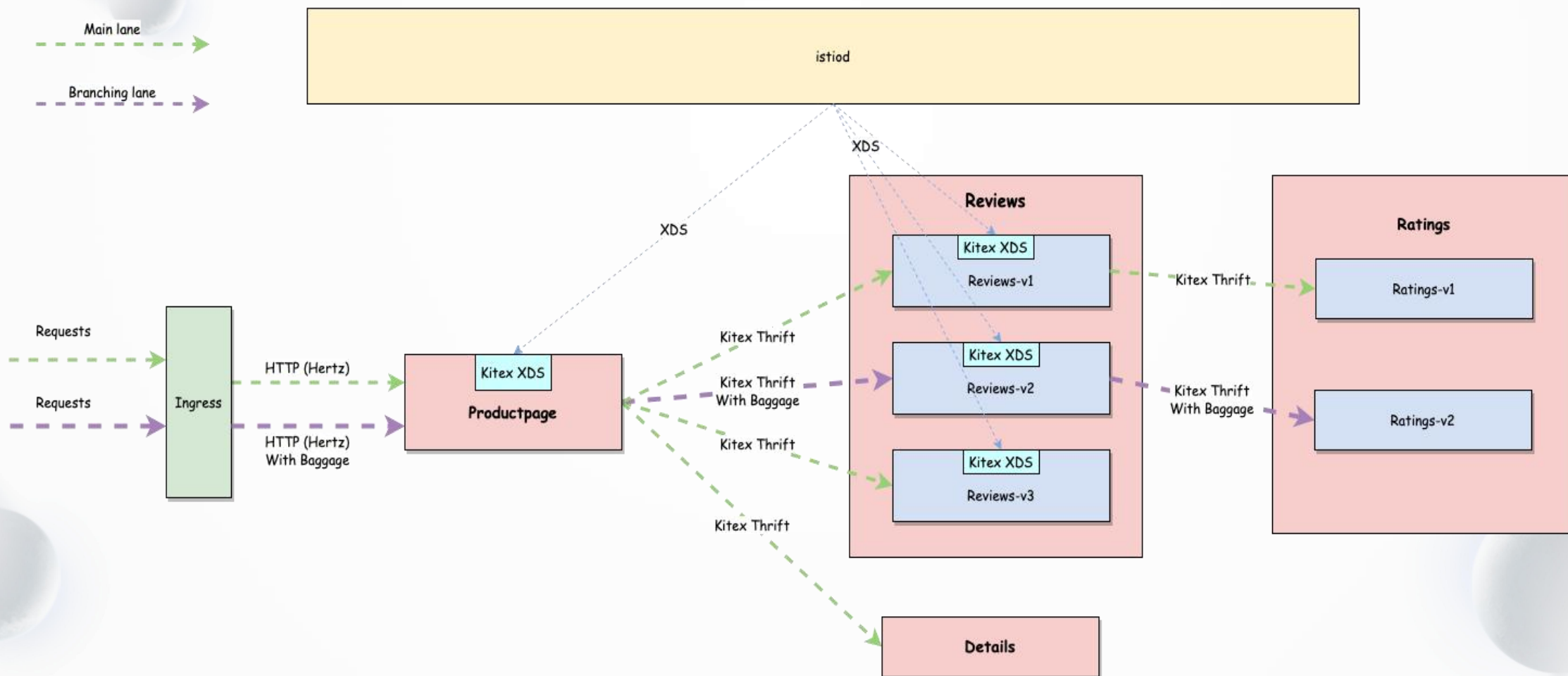
Lb policy 包含在 CDS 内，所以 lb 策略的变化是在动态路由选定 cluster 之后获取并应用的。

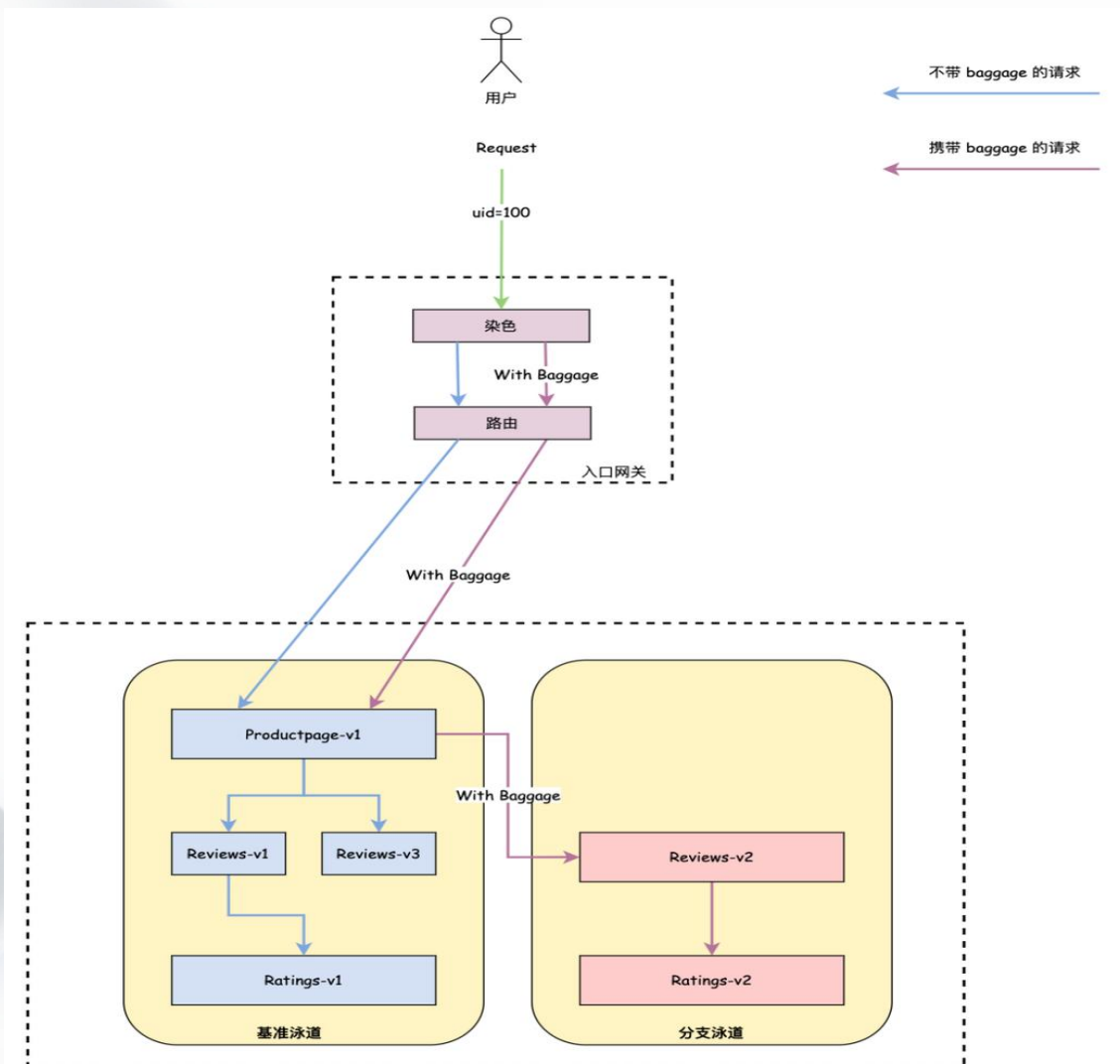
Kitex 支持自定义 LoadBalancer，实际上就是根据 lb 策略，从服务发现获得的实例列表内选取一个实例进行调用。



03

基于 Kitex Proxyless 的全 链路泳道





流量染色

在网关层进行流量染色，根据原始请求中的元数据，来进行一定规则（条件、比例）转换成对应的染色标识

基准泳道

未被染色的流量会被路由到基准泳道中

分支泳道

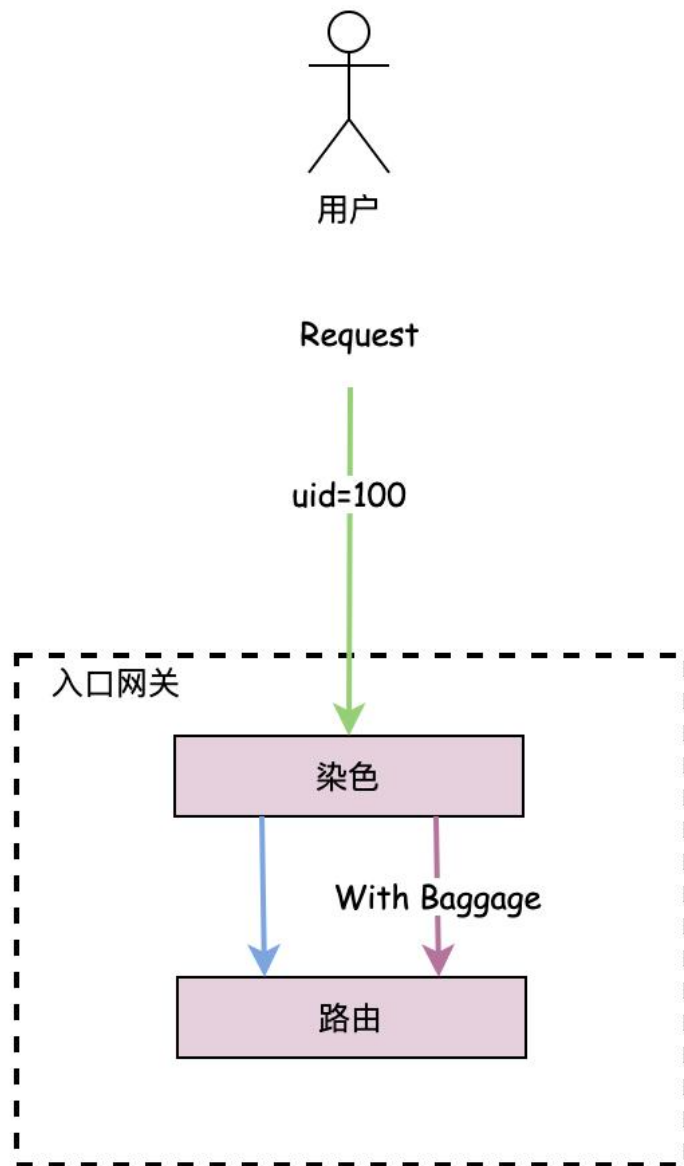
被染色的流量会被路由到
reviews-v2 -> ratings-v2 的分支泳道中

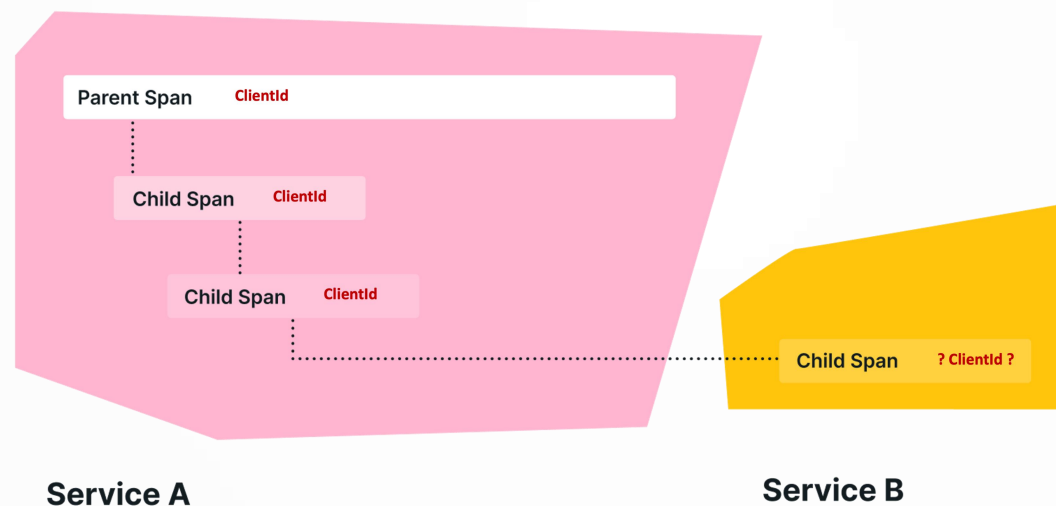
全链路泳道 — 流量染色

我们可以在网关层进行流量染色，通常会根据原始请求中的元数据，来进行一定规则（条件、比例）转换成对应的染色标识。

- 按条件染色：当请求元数据满足一定条件之后，就给当前请求打上染色标识，如：请求头中 uid=100、cookie 匹配等等。
- 按比例染色：按照一定比例，给请求打上染色标识。

有了一套统一的流量染色机制之后，我们配置路由规则的时候，就不需要关心具体的业务属性标识了，只需要根据 染色标识来配置即可。





要实现流量染色标识在全链路透传，我们通常会借助 Tracing Baggage 机制，在全链路中传递对应染色标识，大部分 Tracing 框架都支持 Baggage 概念机能力，如：OpenTelemetry、Skywalking、Jaeger 等等。

有了一套通用的全链路透传机制，业务方就只需要接入一遍 tracing 即可，无需每次业务属性标识发生变化就配合改造一次。

全链路泳道 — 染色标透传 (Baggage)

Kitex 已经原生支持了 opentelemetry, 开启 opentelemetry 即可自动透传 Baggage

```
if opts.EnableXDS {  
    if err := xdsmanager.Init(xdsmanager.WithXDSServerAddress(opts.XDSAddr)); err != nil {  
        klog.Fatal(err)  
    }  
    return reviewsservice.NewClient(  
        opts.Endpoint, // use svc fqdn  
        kclient.WithSuite(tracing.NewClientSuite()),  
        kclient.WithXDSuite(xds.ClientSuite{  
            RouterMiddleware: xdssuite.NewXDSRouterMiddleware(  
                xdssuite.WithRouterMetaExtractor(metadata.ExtractFromPropagator),  
            ),  
            Resolver: xdssuite.NewXDSResolver(),  
        }),  
    )  
}
```

开启 opentelemetry 即可自动透传 baggage

全链路泳道 — 泳道效果（基准泳道）

入口流量请求头中不带 uid=100 的请求，会自动路由到基准泳道服务，reviews v1 和 v3 服务间轮询，展示的效果是评分为 0 或 1 随机。



全链路泳道 — 泳道效果（分支泳道）

入口流量请求头中携带了 uid=100 的请求，会被精准路由到分支泳道



THANKS