



全栈服务网格 - Aeraki Mesh 助你在 Istio 服务网格中管理任何七层流量

赵化冰@腾讯云





讲师简介

赵化冰

腾讯云技术专家


Aeraki Mesh 开源项目创始人

 @zhaohuabing

 @zhaohuabing

 @zhaohuabing

 @zhaohuabing

 <https://zhaohuabing.com>





目录

- ❑ Service Mesh 中的七层流量管理能力
- ❑ 如何在服务网格中管理 Dubbo、Thrift, 以及私有协议
- ❑ Aeraki Mesh 的实现原理
- ❑ MetaProtocol 七层代理框架介绍
- ❑ Aeraki Mesh 的产品落地案例
- ❑ Aeraki Mesh 的开源生态





Service Mesh

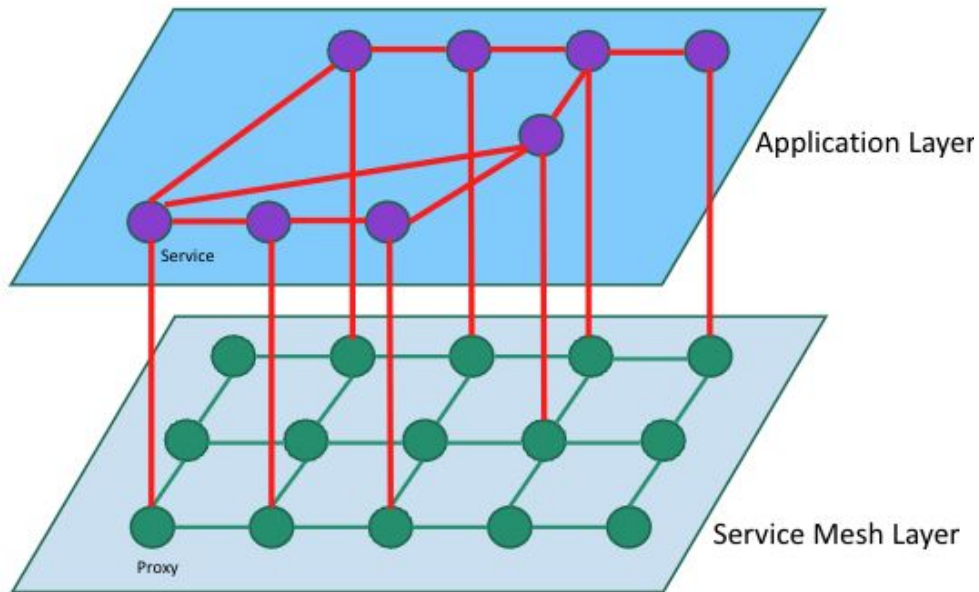
处理服务间通信(七层通信)的云原生基础设施层:

Service Mesh 将各个服务中原来使用 SDK 实现的七层通信相关功能抽象出来, 使用一个专用层次来实现, Service Mesh 对应用透明, 因此应用可以无需关注分布式架构带来的通信相关问题, 而专注于其业务价值。

流量控制: 服务发现、请求路由、负载均衡、灰度发布、错误重试、断路器、故障注入

可观察性: 遥测数据、调用跟踪、服务拓扑

通信安全: 服务身份认证、访问鉴权、通信加密



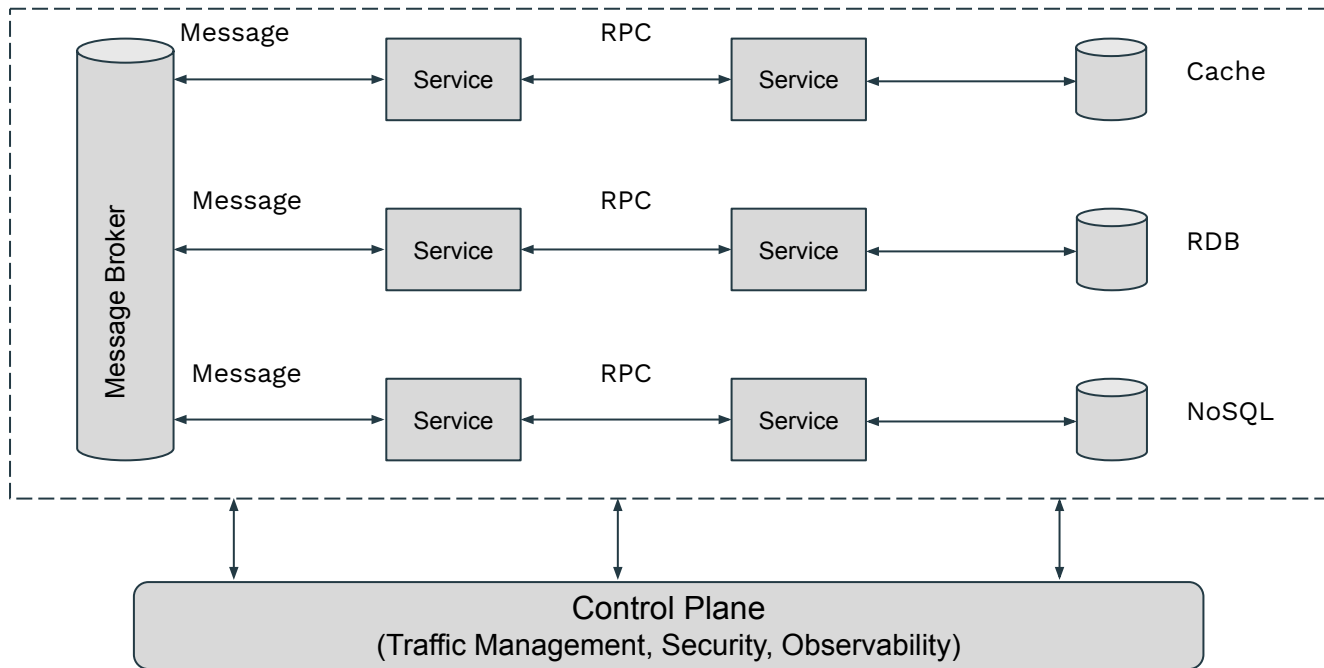
微服务中的常见七层协议

微服务中除 HTTP 之外的的常见七层协议：

- RPC: Thrift, Dubbo, Private RPC Protocols ...
- Messaging: Kafka, RabbitMQ ...
- Cache: Redis, Memcached ...
- Database: MySQL, PostgreSQL, MongoDB ...

大多数 Service Mesh 实现都不能在七层上处理这些协议

- 主要关注 HTTP
- 其他协议的流量被作为 TCP 看待



我们希望从服务网格中获得这些协议的哪些治理能力

我们期望的网格能力:七层流量管理能力

- 服务发现(基于服务的逻辑名称, 如 Host, Service)
- 七层负载均衡、基于应用协议的错误码进行重试和熔断
- 基于七层协议头的路由(RPC协议中的调用服务名、方法名等)
- 故障注入(RPC 协议层的错误码)
- 七层请求 Metrics(调用次数, 调用失败率等)
- 调用跟踪

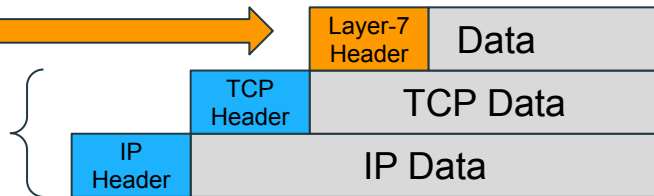
我们得到的网格能力:三/四层流量管理能力

- 服务发现(基于 VIP 或者 Pod IP:DNS 只用于解析得到 IP, 不能被 Envoy 感知)
- 四层负载均衡、基于四层链接错误的重试和熔断
- 基于四层的路由(IP + Port)
- 基于四层的 Metrics(TCP收发包数量等)

我们期望的网格能力



我们得到的网格能力



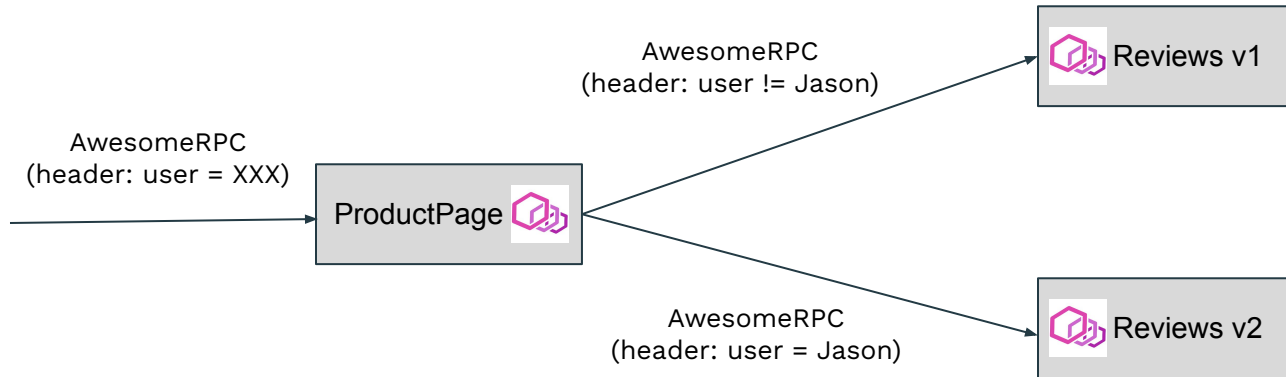
应用协议数据包



如何在 Istio 服务网格中对非 HTTP 协议进行管理

以 Istio 的 BookInfo 为例：假设采用一个私有的RPC协议：AwesomeRPC

如何在 Istio 中实现基于 AwesomeRPC 协议的 Traffic split 用例？





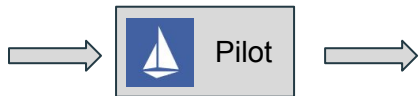
如何在 Istio 服务网格中对非 HTTP 协议进行管理

Istio Config

```

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews-route
spec:
  hosts:
  - reviews.prod.svc.cluster.local
  awesomeRPC:
  - name: "canary-route"
    match:
    - headers:
      user:
        exact: Jason
    route:
    - destination:
        host: reviews.prod.svc.cluster.local
        subset: v2
  - name: "default"
    route:
    - destination:
        host: reviews.prod.svc.cluster.local
        subset: v1

```



控制面侧改动:

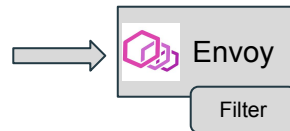
- 支持为 AwesomeRPC 配置路由规则
- 生成 AwesomeRPC 相关的数据面配置

Envoy Config

```

{
  "virtual_hosts": [
    {
      "name": "reviews.default.svc.cluster.local:9080",
      "services": [
        "reviews.default.svc.cluster.local",
        "reviews"
      ],
      "routes": [
        {
          "name": "canary-route"
          "match": {
            "headers": [
              {
                "name": ":user",
                "exact_match": "Jason"
              }
            ]
          },
          "route": {
            "cluster": "outbound|9080||reviews.default.svc.cluster.local | v2",
          },
        },
        {
          "name": "default"
          "route": {
            "cluster": "outbound|9080||reviews.default.svc.cluster.local | v1",
          },
        }
      ]
    }
  ],
}

```



数据面侧:

AwesomeRPC Filter

- Decoding/Encoding
- Routing
- Load balancing
- Circuit breaker
- Fault injection
- Stats
- ...

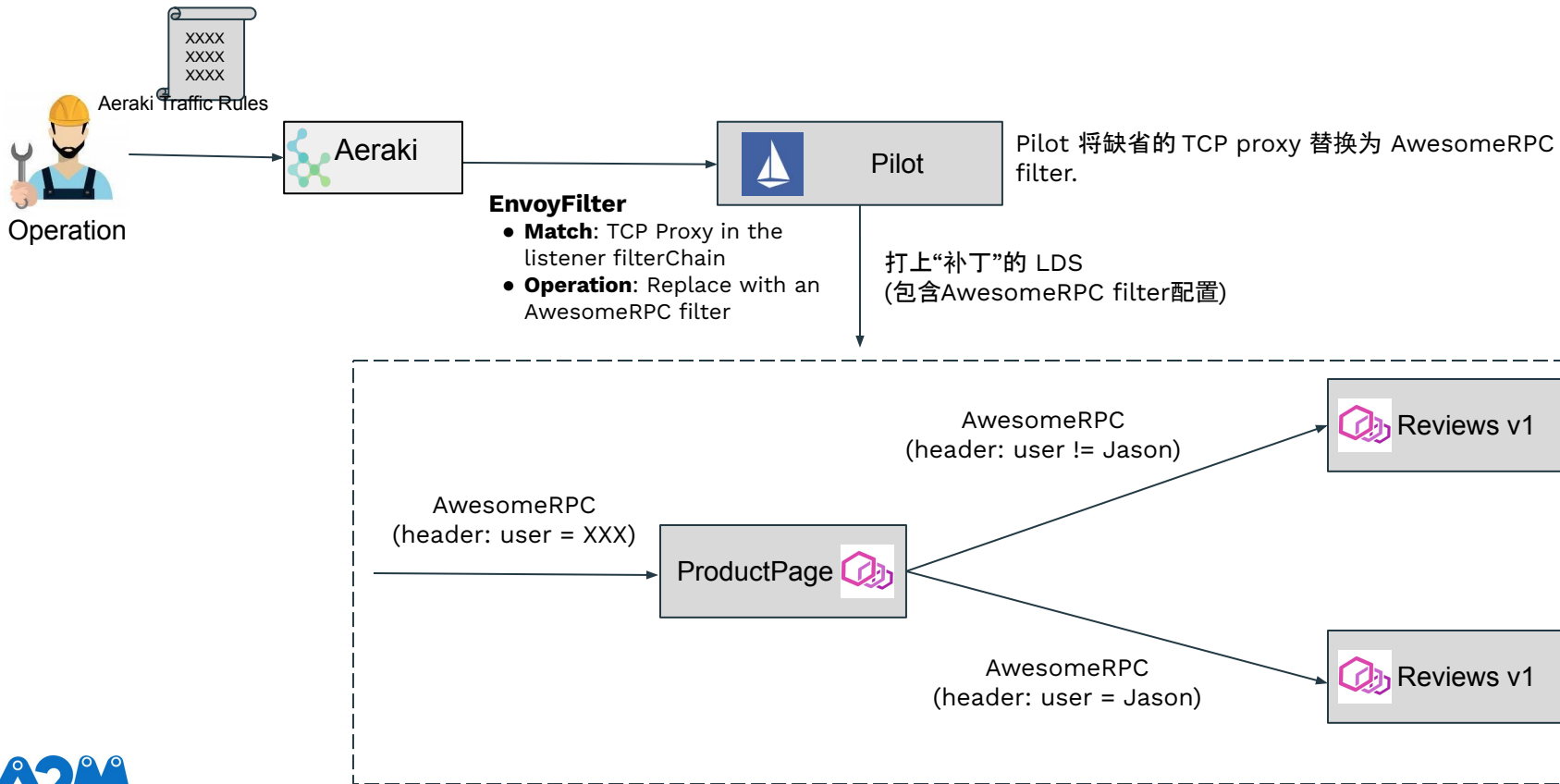
该方案的问题:

- 控制面: 需要 fork 并维护一个 Istio 私有分支-维护代价大, 升级困难
- 数据面: 编写一个 Envoy 插件来实现私有协议的处理: 工作量非常大, 需要对 Envoy 有源码级的定制能力



控制面解决思路: Istio EnvoyFilter CRD

EnvoyFilter 是一个 Istio 的配置 CRD。可以使用 EnvoyFilter 向 Istio 生成的 Envoy 配置打一个“补丁”。



数据面解决思路：一个通用的七层代理框架

大部分七层协议的路由、熔断、负载均衡等能力的实现逻辑是类似的，没有必要每个协议都全部从头实现，重复造轮子。

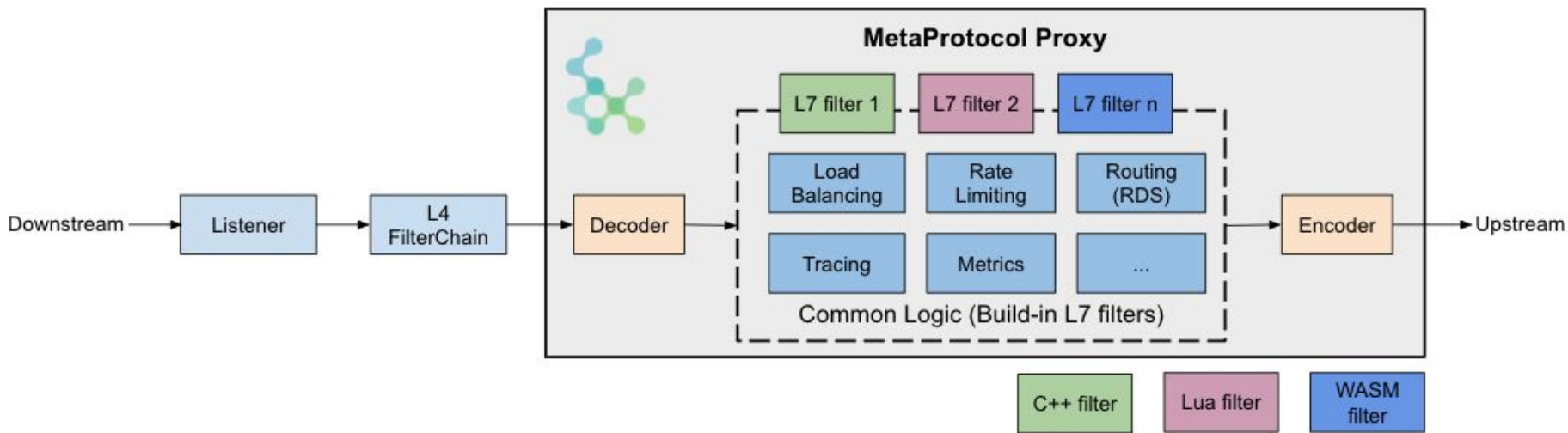
Protocol	Destination service	Parameters could be used for routing
HTTP 1.1	host	host, path, method headers
HTTP 2	pseudo header: authority	pseudo header: authority, path, method, headers
gRPC	HTTP 2 path	Request-Headers(Delivered as HTTP2 headers)
TARS	ServantName	ServantName, FuncName, Context
Dubbo	service name	service name, service version, service method
Any RPC Protocol	service name in message header	some key:value pairs in message header





MetaProtocol: 基于 Envoy 的七层协议框架

- MetaProtocol Proxy 中实现七层协议的通用逻辑：负载均衡、熔断、动态路由、消息头修改、本地\全局限流、请求指标上报、调用跟踪等。
- 基于 MetaProtocol 实现一个自定义协议时，只需要实现 Decode 和 Encode 扩展点的少量代码（数百行代码）。
- 提供基于 C++、WASM、Lua 的 L7 filter 扩展点，用户可以实现一些灵活的自定义协议处理逻辑，例如认证鉴权等。



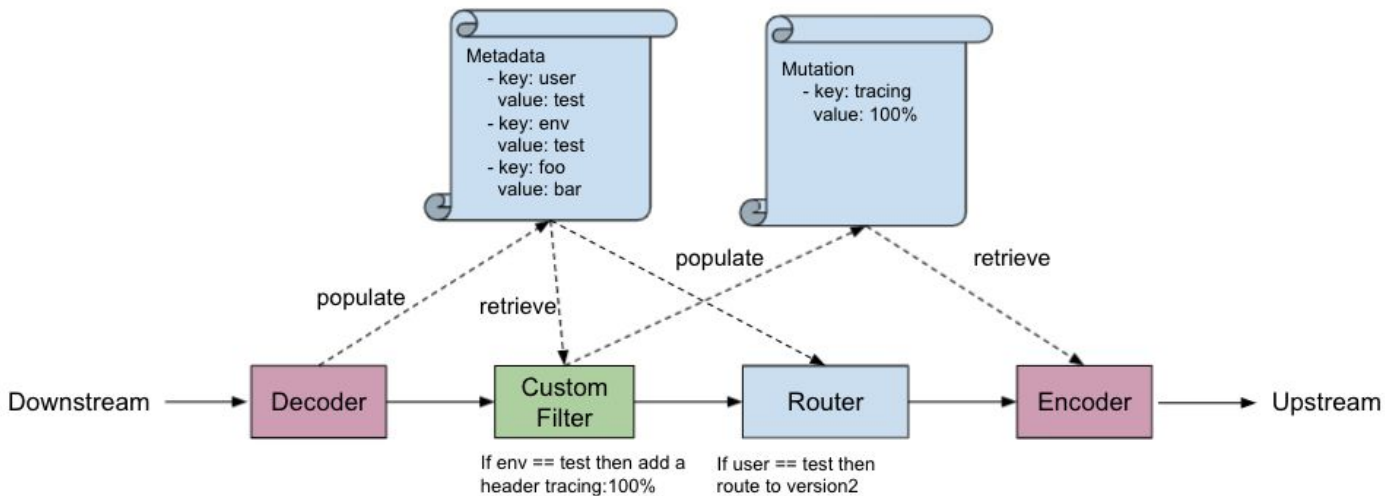
MetaProtocol: 请求处理路径

处理流程:

1. Decoder 解析 Downstream 请求, 填充 Metadata
2. L7 filter 从 Metadata 获取所需的数据, 进行请求方向的业务处理
3. L7 filter 将需要修改的数据放入 Mutation 结构中
4. Router 根据 RDS 配置的路由规则选择 Upstream Cluster
5. Encoder 根据 Mutation 结构封包
6. 将请求发送给 Upstream

L7 filter 共享数据结构:

- Metadata: decode 时填充的 key:value 键值对, 用于 L7 filter 的处理逻辑中
- Mutation: L7 filter 填充的 key:value 键值对, 用于 encode 时修改请求数据包



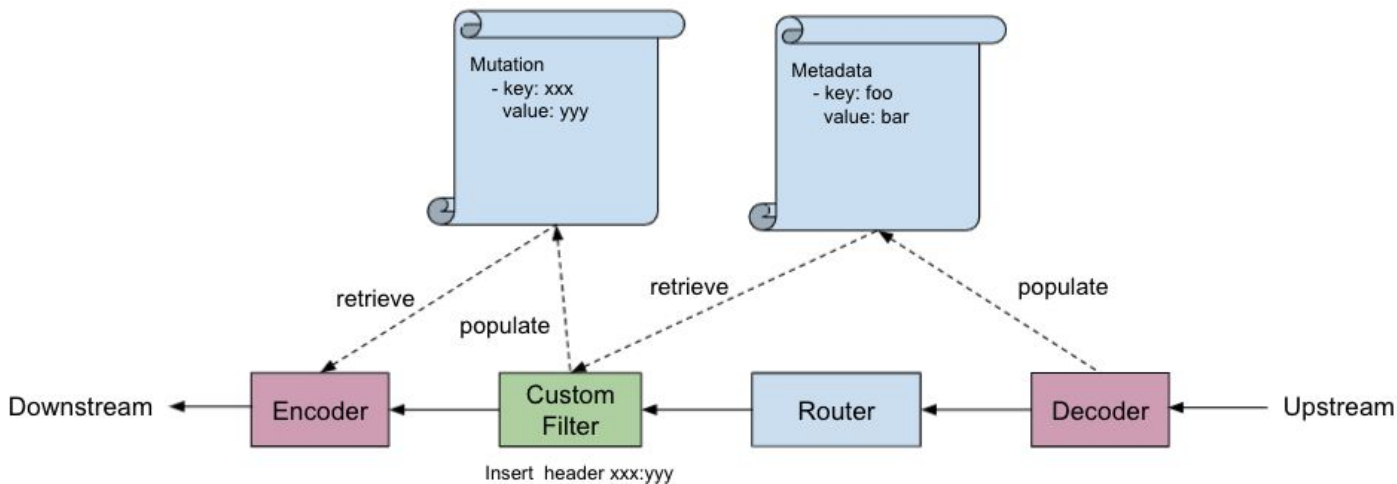
MetaProtocol: 响应处理路径

处理流程:

1. Decoder 解析 Upstream 的响应, 填充 Metadata
2. Router 根据 connection/stream 对应关系找到响应的 Downstream 连接
3. L7 filter 从 Metadata 获取所需的数据, 进行响应方向的业务处理
4. L7 filter 将需要修改的数据放入 Mutation 结构中
5. Encoder 根据 Mutation 结构封包
6. 将响应发送到 Downstream

L7 filter 共享数据结构:

- Metadata: decode 时填充的 key:value 键值对, 用于 L7 filter 的处理逻辑中
- Mutation: L7 filter 填充的 key:value 键值对, 用于 encode 时修改响应数据包

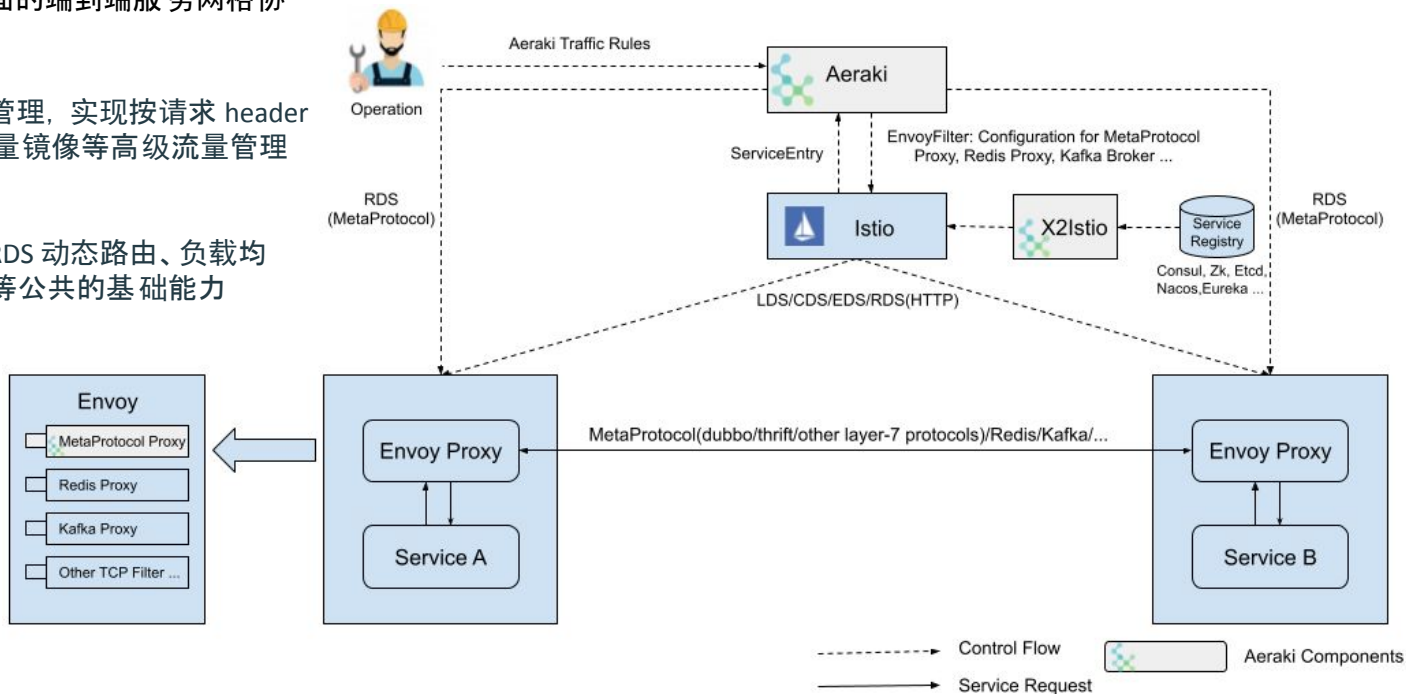




Aeraki Mesh 架构(Aeraki + MetaProtocol 双剑合璧)

Aeraki [发音 Air-rah-ki]: 是希腊语中“微风”的意思, 希望这一缕清风助力 Istio 在云原生的海洋中航行得更快更远。
Aeraki Mesh 提供了数据面+控制面的端到端服务网格协议扩展解决方案。

- 控制面: Aeraki + Istio 提供控制面管理, 实现按请求 header 路由、灰度发布、地域感知 LB、流量镜像等高级流量管理能力。
- 数据面: MetaProtocol Proxy 实现 RDS 动态路由、负载均衡、熔断、Metrics 和 Tracing 上报等公共的基础能力





Aeraki Mesh 组件

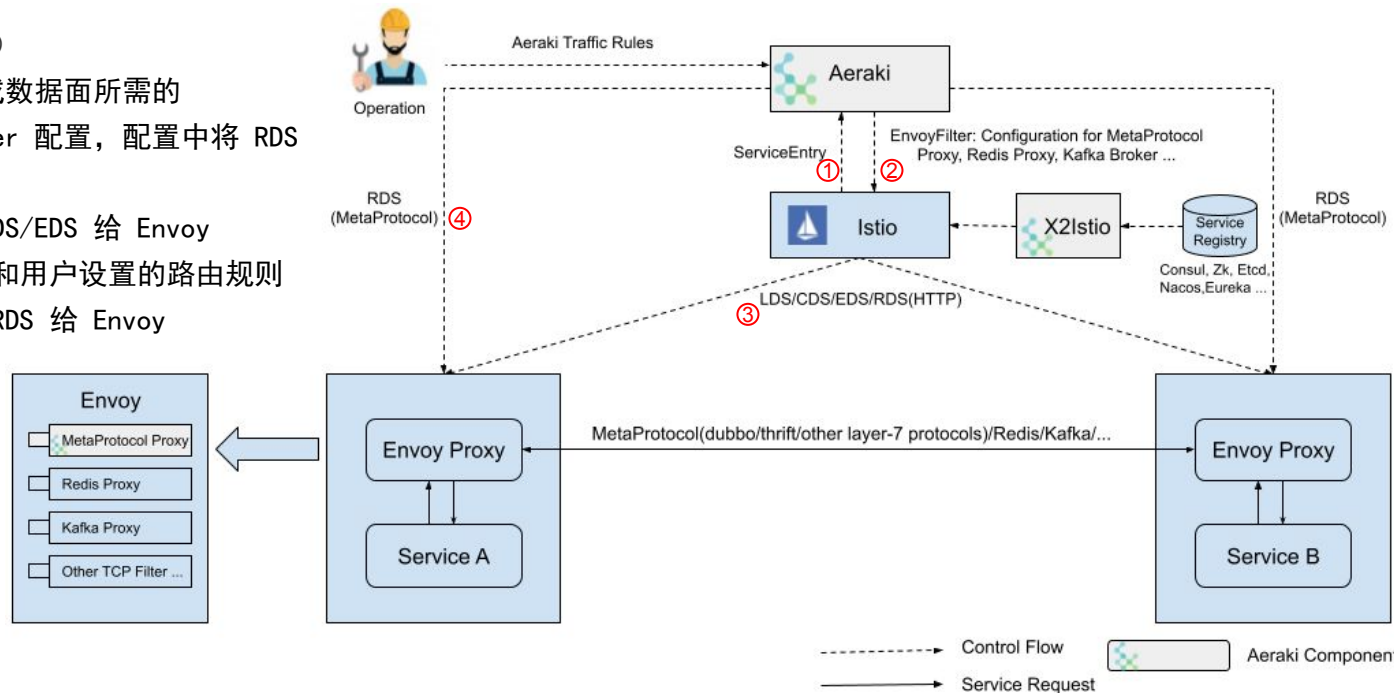
- **Aeraki:**
 - 提供面向运维的, 用户友好的七层流量管理配置规则。
 - 将运维配置的流量规则翻译为 envoy 配置, 通过 Istio 的 EnvoyFilter API 将配置下发到数据面的 sidecar 代理。
 - 为数据面的 MetaProtocol Proxy 提供 RDS (路由发现服务)。不同于专注于 HTTP 的 Istio RDS, Aeraki 提供的 RDS 服务旨在提供为七层协议提供一个通用动态路由发现服务, 可以用于所有基于 MetaProtocol 之上的应用协议。
- **MetaProtocol Proxy:**
 - 提供通用的七层流量治理能力, 包括负载均衡、断路、重试、路由、限流、故障注入、可观察性数据上报等等。
 - 提供编解码接口, 只需要实现该接口即可基于 MetaProtocol Proxy 实现一个应用协议。
 - 提供七层 filter 接口能力, 可基于 MetaProtocol 开发自定义七层 filter, 加入应用的特有处理逻辑。
- **X2Istio** (Consul2Istio, Dubbo2Istio: etcd, zk, nacos): 将第三方注册表连接到 Istio。



控制流

通过 Aeraki 实现控制面的流量管理规则下发：

1. Aeraki 从 Istio 中获取 ServiceEntry, 通过端口命名判断协议类型 (如 tcp-metaprotocol-thrift)
2. 为 MetaProtocol 服务生成数据面所需的 MetaProtocol Proxy Filter 配置, 配置中将 RDS 指向 Aeraki
3. Istio 下发 LDS (Patch)/CDS/EDS 给 Envoy
4. Aeraki 根据服务发现数据和用户设置的路由规则动态生成路由规则, 通过 RDS 给 Envoy





基于 MetaProtocol 接入一个私有协议

1、实现 Codec 接口(约数百行代码)

可参考 dubbo 和 thrift 的实现:

https://github.com/aeraki-mesh/meta-protocol-roxy/tree/master/src/application_protocols

自定义协议项目模板:

<https://github.com/aeraki-mesh/meta-protocol-a-wesomercp>

```
class Codec {
public:
    virtual ~Codec() = default;

    /*
     * decodes the protocol message.
     *
     * @param buffer the currently buffered data.
     * @param metadata saves the meta data of the current message.
     * @return DecodeStatus::DONE if a complete message was successfully consumed,
     * DecodeStatus::WaitForData if more data is required.
     * @throws EnvoyException if the data is not valid for this protocol.
     */
    virtual DecodeStatus decode(Buffer::Instance& buffer, Metadata& metadata) PURE;

    /*
     * encodes the protocol message.
     *
     * @param metadata the meta data produced in the decoding phase.
     * @param mutation the mutation that needs to be encoded to the message.
     * @param buffer save the encoded message.
     * @throws EnvoyException if the metadata or mutation is not valid for this protocol.
     */
    virtual void encode(const Metadata& metadata, const Mutation& mutation,
                       Buffer::Instance& buffer) PURE;

    /*
     * encodes an error message. The encoded error message is used for local reply, for example, envoy
     * can't find the specified cluster, or there is no healthy endpoint.
     *
     * @param metadata the meta data produced in the decoding phase.
     * @param error the error that needs to be encoded in the message.
     * @param buffer save the encoded message.
     * @throws EnvoyException if the metadata is not valid for this protocol.
     */
    virtual void onError(const Metadata& metadata, const Error& error, Buffer::Instance& buffer) PURE;
};
```





基于 MetaProtocol 接入一个私有协议

2、在控制面采用 ApplicationProtocol CRD 定义七层协议

```
apiVersion: metaprotocol.aeraki.io/v1alpha1
kind: ApplicationProtocol
metadata:
  name: dubbo
spec:
  codec: aeraki.meta_protocol.codec.dubbo
  protocol: dubbo
```





流量管理示例

权重路由

```
apiVersion: metaprotocol.aeraki.io/v1alpha1
kind: MetaRouter
metadata:
  name: test-metaprotocol-dubbo-route
  namespace: meta-dubbo
spec:
  hosts:
  - org.apache.dubbo.samples.basic.api.demoservice
  routes:
  - name: traffic-split
    match:
      attributes:
        interface:
          exact: org.apache.dubbo.samples.basic.api.DemoService
        method:
          exact: sayHello
        foo:
          exact: bar
    route:
  - destination:
      host: org.apache.dubbo.samples.basic.api.demoservice
      subset: v1
      weight: 20
  - destination:
      host: org.apache.dubbo.samples.basic.api.demoservice
      subset: v2
      weight: 80
```

本地限流

```
apiVersion: metaprotocol.aeraki.io/v1alpha1
kind: MetaRouter
metadata:
  name: test-metaprotocol-thrift-route
  namespace: meta-thrift
spec:
  hosts:
  - thrift-sample-server.meta-thrift.svc.cluster.local
  localRateLimit:
    tokenBucket:
      fillInterval: 60s
      maxTokens: 5
      tokensPerFill: 5
    conditions:
  - tokenBucket:
      fillInterval: 10s
      maxTokens: 2
      tokensPerFill: 2
    match:
      attributes:
        method:
          exact: sayHello
```



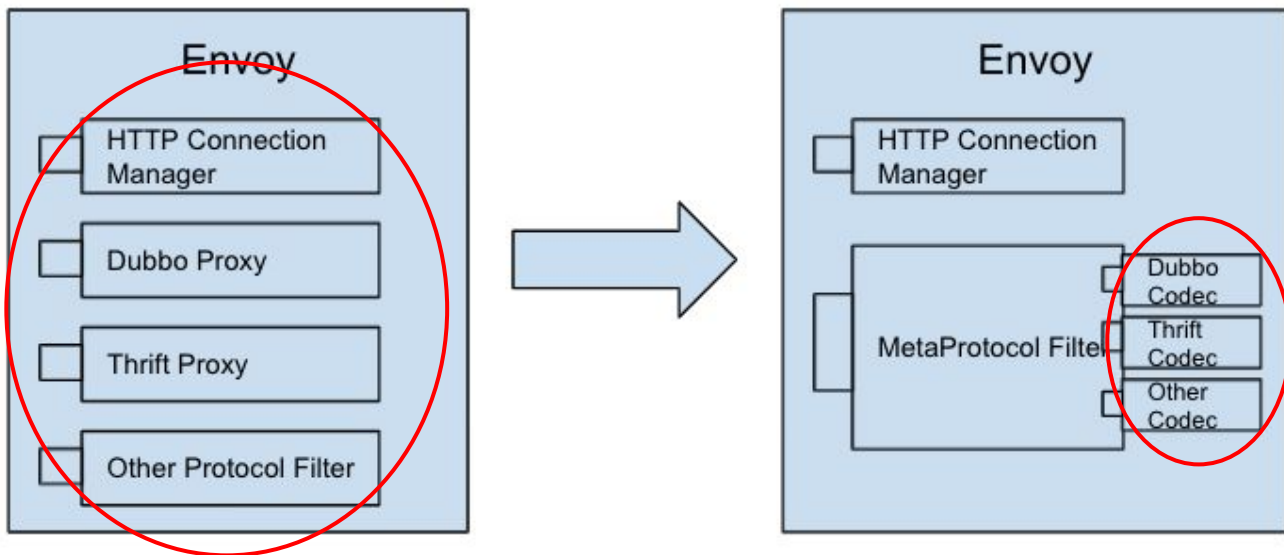
协议接入工作量对比(采用/不采用 Aeraki Mesh)

数据面的工作量:

- 不采用 Aeraki Mesh: **巨大的工作量**: 需要编写一个完整的 Envoy L4 filter
- 采用 Aeraki Mesh: **很少的工作量**: 只需要实现 codec 接口 (通常数百行代码)

控制面的工作量:

- 不采用 Aeraki Mesh: **巨大的工作量**: 需要专门为该协议编写一个控制面 (基于 Istio 魔改或者从头编写)
- 采用 Aeraki Mesh: **工作量为零**: Aeraki 可作为任何基于 MetaProtocol 的协议的控制面



服务网格产品治理能力对比

协议	Istio	Linkerd	Aeraki Mesh + Istio
Http/gRPC	支持	支持	同 Istio
Dubbo	不支持	不支持	支持(七层)
Thrift	不支持	不支持	支持(七层)
Redis	不支持	不支持	支持(七层)
私有协议	不支持	不支持	支持(七层)



Aeraki Mesh 的优势

- 只需实现编解码接口少量代码即可快速接入新的七层协议
- 完整的 Mesh 数据面+控制面闭环解决方案
- MetaRouter 路由匹配条件非常灵活, 理论上可以支持采用协议包中的任意字段路由、限流等治理
- 和 Istio 无缝集成, 可充分利用 Istio 已有能力, 快速跟随上游社区迭代。
- 可以基于 Aeraki Mesh + Istio 构建一个管理 HTTP 和 Dubbo、Thrift, 以及私有协议流量的全栈服务网格。
- 对于无法纳入 MetaProtocol 管理的特殊协议, Aeraki 也有对应的插件进行支持 (Redis、Kafka、Zookeeper等)



协议支持

- MetaProtocol-Dubbo
- MetaProtocol-Thrift
- MetaProtocol-tRPC(腾讯内部RPC协议)
- MetaProtocol-腾讯音乐私有协议
- MetaProtocol-腾讯融媒体私有协议
- MetaProtocol-腾讯游戏私有协议(接入中)
- Redis (Envoy 原生 Filter)
- Kafka (Envoy 原生 Filter)
- ZooKeeper (Envoy 原生 Filter)

功能特性

- 七层(请求级别)负载均衡(支持一致性哈希/会话粘滞)
- 请求熔断保护
- 基于 Metadata 的灵活 RDS 动态路由
- 流量拆分蓝绿部署/灰度发布
- 本地/全局限流
- 消息头更改
- 请求级指标(平均/Pxx 请求时延, 错误统计等)
- 流量镜像 — 开发中
- 调用跟踪 — 开发中

产品落地

- 腾讯融媒体/冬奥会视频直播
- 腾讯音乐
- 小红书
- 某腾讯游戏项目(协议接入中)
- 某大型连锁超市(协议接入中)
- 某政府采购平台(灰度测试中)



Aeraki Mesh 典型案例：在冬奥会直播中的应用

产品介绍：承担国内外大型赛事、政治新闻、明星热点、热播影视剧在移动端向全球中文网民的稳定输出





Aeraki Mesh 典型案例:应用项目背景

业务角度:业务复杂

语言、协议、框架、基础组件、开发团队繁多,异构化严重

- 开发语言:C++/Java/Go/Python/Rust等
- 协议类型:HTTP/tRPC/Videopacket等

业务角度:难以维护

- 模体量大:靠传统运维手段,已经难以支撑如此大的业务体量
- 热点事件突发:突发流量存在业务宕机隐患
- 无流量管控:缺乏分布式场景流量调度、管控能力

技术选型:决策因素

- 业务无侵入:业务开发时间紧张,无多余人力投入治理需求
- 主流技术:Istio:Service Mesh 领域事实标准
- 多协议支持:Aeraki Mesh:基于 Istio 支持七层协议扩展





Aeraki Mesh 典型案例: 服务过载保护

依托 Aeraki Mesh 的限流和熔断对服务提供过载保护

本地限流接入
100%

本地限流保护服务不被瞬时突增流量击穿，所有服务均已配置。

推荐使用方式：

压测平台压测不同应用 -> 根据容量模型预测容量 -> 配置本地限流阈值

全局限流接入
3条业务线

全局限流保护某条请求链路的整体稳定性。

熔断接入
100%

熔断保护客户端总是拿到健康的服务端实例节点。





Aeraki Mesh 典型案例: 基于限流指标的HPA

基于 Aeraki Mesh 提供的限流频次指标进行服务水平扩容

- 相对于 CPU/内存 等资源指标, 限流次数指标更贴合服务的实际业务压力
- 可以在保证已有服务稳定的情况下有效地 应对突发请求流量

扩缩容

方式 手动 自动

伸缩策略

禁止缩容

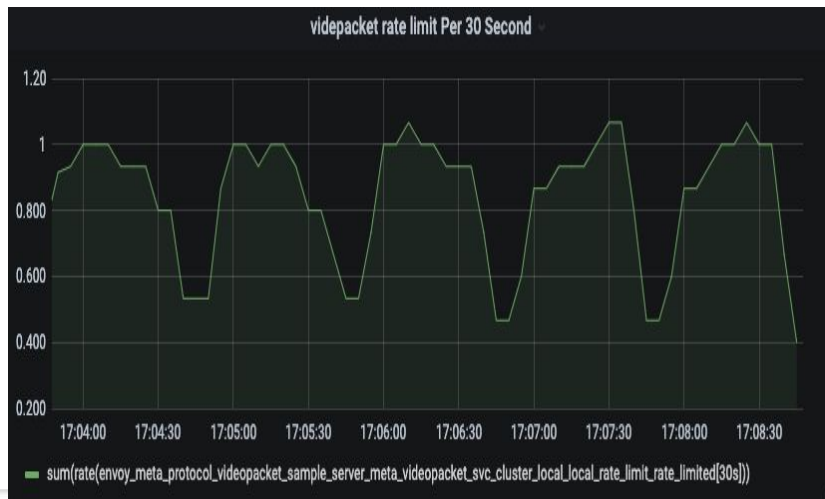
禁止扩容

* 触发策略 每30秒限流频次 次

伸缩范围

* 弹性伸缩范围

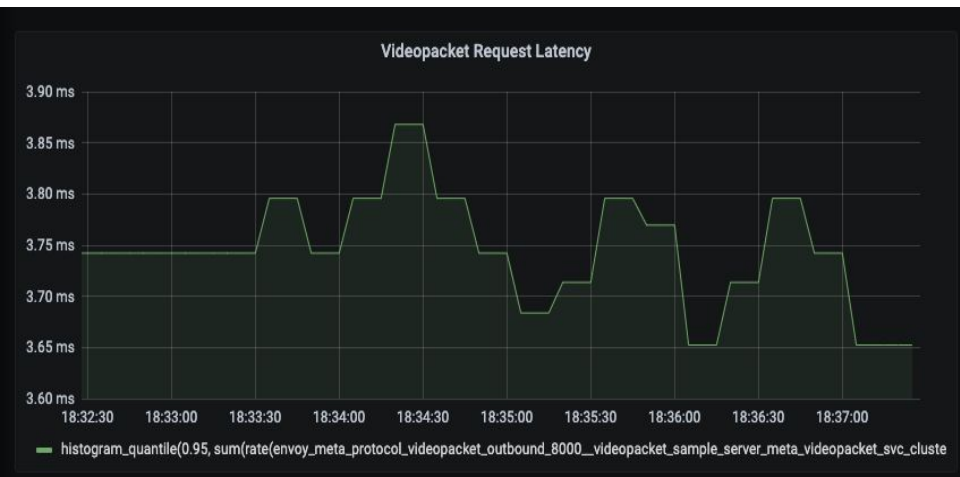
- CPU使用量
- 内存使用量
- 网络入带宽
- 网络出带宽



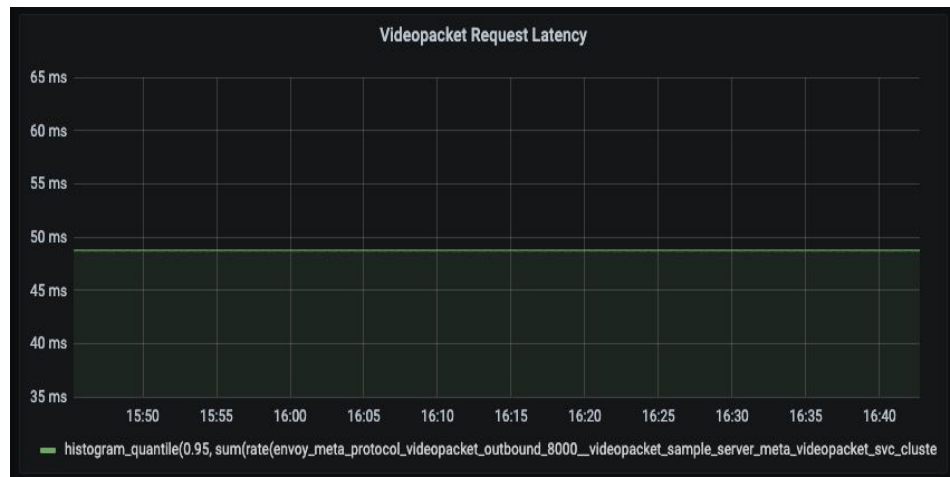


Aeraki Mesh 典型案例: Sidecar 时延

20字节请求耗时



12万字节请求耗时



Videopacket协议请求耗时:

- 1、20字节代理层增加耗时: 1~6ms
- 2、12万字节代理层增加耗时: 15~30ms

注意:

- 1、单纯谈请求耗时并没有意义, 需要看业务占比
- 2、耗时跟协议的编解码性能有关, 性能优的协议耗时相对少
- 3、跟请求体的大小正相关, 这一点从图中也可以看出来








Aeraki Mesh 开源生态

Istio ecosystem integration 项目



providers pro services integrations

Istio is a vibrant part of the cloud native stack. These are some of the projects and software that integrate with Istio to enable added functionality.

 <p>Aeraki extends Istio to manage traffic for any layer-7 protocols.</p> <p>Learn more</p>	 <p>Ambassador Edge Stack and Istio can be deployed together on Kubernetes.</p> <p>Learn more</p>	 <p>Apigee lets you centrally govern or manage APIs, providing centralized API publishing, visibility, governance, and usage analytics.</p> <p>Learn more</p>
--	--	--

CNCF 云原生全景图 Service Mesh 项目
















CNCF Cloud Native Interactive Landscape

The Cloud Native Trail Map (png, pdf) is CNCF's recommended path through the cloud native landscape. The cloud native landscape (png, pdf), serverless landscape (png, pdf), and member landscape (png, pdf) are dynamically generated below. Please open a pull request to correct any issues. Greyed logos are not open source. Last Updated: 2022-03-02 06:07:47Z

You are viewing 15 cards with a total of 72,732 stars, market cap of \$3.8T and funding of \$187.6M.

[Landscape](#)
[Card Mode](#)
[Members](#)
[Serverless](#)
[Wasm](#)
[Tweet](#) (1794)

Orchestration & Management - Service Mesh (15)

 <p>Aeraki Mesh ★ 571 Tencent Cloud MCap: \$526.7B</p>	 <p>AWS App Mesh ★ 171 Amazon Web Services MCap: \$1.9T</p>	 <p>Consul ★ 24,350 HashiCorp MCap: \$8.8B</p>	 <p>EaseMesh ★ 390 MingFase, Inc.</p>	 <p>GLASNOSTIC ★ 144 Glasnostic Funding: \$2.1M</p>	 <p>Gloo Mesh ★ 144 Solo.io Funding: \$175.5M</p>
 <p>Grey Matter ★ 21,633 GreyMatter.io MCap: \$1.8T</p>	 <p>Istio ★ 2,619 Google MCap: \$1.8T</p>	 <p>Kuma ★ 1,245 Cloud Native Computing Foundation (CNCF) Funding: \$3M</p>	 <p>LINKERD ★ 1,245 Cloud Native Computing Foundation (CNCF) Funding: \$3M</p>	 <p>MESHERY ★ 1,245 Cloud Native Computing Foundation (CNCF) Funding: \$3M</p>	 <p>Open Service Mesh ★ 2,329 Cloud Native Computing Foundation (CNCF) Funding: \$3M</p>
 <p>Service Mesh Interface (SMI) ★ 307 Cloud Native Computing Foundation (CNCF) Funding: \$3M</p>	 <p>Service Mesh Performance ★ 211 Cloud Native Computing Foundation (CNCF) Funding: \$3M</p>	 <p>Traefik Mesh ★ 1,615 Traefik Labs Funding: \$11.1M</p>			





Demo 及教程

快速开始

Get Aeraki up and running in less than 5 minutes!

请参照下面的步骤来安装、运行和测试 Aeraki:

1. 从 github 下载 Aeraki。

```
git clone https://github.com/aeraki-mesh/aeraki.git
```

2. 安装 Aeraki, Istio 和 demo 应用。

```
make demo
```

请注意: Aeraki 要求启用 [Istio DNS 代理](#)。如果你在一个正在运行的 Istio 部署上安装 Aeraki, 请确保打开了 Istio DNS 代理功能; 你可以直接使用 `make demo` 命令来在一个全新的 K8s 集群上从头安装 Aeraki 和 Istio, `make demo` 会对 Istio 进行正确的配置。

3. 安装 aerakitl

aerakitl 脚本工具封装了一些常用的 debug 命令, 我们将在后续的教程中使用这些命令来查看应用程序和代理的信息。

```
git clone https://github.com/aeraki-mesh/aerakitl.git ~/aerakitl;source ~/aerakitl/aerakitl.sh
```

4. 在浏览器中打开下面的网页, 来查看 Aeraki 部署的应用程序和上报的请求指标数据。

- Kaili `http://{istio-ingressgateway_external_ip}:20001`
- Grafana `http://{istio-ingressgateway_external_ip}:3000`
- Prometheus `http://{istio-ingressgateway_external_ip}:9090`

在线 demo:

<http://aeraki.zhaohuabing.com:3000/d/pgz7wp-Gz/aeraki-demo?orgId=1&refresh=10s&kiosk>

使用教程: <https://www.aeraki.net/zh/docs/v1.0/tutorials/>





如何参与社区？

参与社区会议：<https://www.aeraki.net/zh/community/#community-meetings>

Community meetings

Aeraki community don't hold meetings on a regular basis. An ad-hoc meeting will be proposed when the community have some technical topics that need to be discussed.

For phone-in information, the date of the next meeting, and minutes from past meetings, see [Aeraki community meeting](#).

Tencent Meeting

Join contributors and maintainers [online](#).

Meeting doc

For meeting details, consult the [Aeraki community meeting document](#).

YouTube

Missed a meeting? No problem. See the [Aeraki channel](#) for meeting videos.

参与微信群：请微信联系 zhaohuabing 进群

Aeraki Mesh 官网：<https://www.aeraki.net>

Aeraki Mesh Github：<https://github.com/aeraki-mesh>





微信扫码加入 Aeraki 讨论群

感谢聆听！

网站：<https://www.aeraki.net>

Github：<https://github.com/aeraki-mesh>