

Tencent Music's service mesh practice with Istio and Aeraki (Istio + Aeraki 在腾讯音乐的服务网格落地)

Huabing Zhao@Tencent Cloud, ChengQiang Wang@Tencent Music



Huabing Zhao


Software Engineer @ Tencent Cloud
Creator of Aeraki Mesh

 @zhaohuabing

 @zhaohuabing

 @zhaohuabing

 @zhaohuabing

 <https://zhaohuabing.com>



#IstioCon



目录

- ❑ Service Mesh 中的七层流量管理能力
- ❑ 如何在 Istio 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 的开源生态
- ❑ Aeraki Mesh 在腾讯音乐的落地实践



Service Mesh

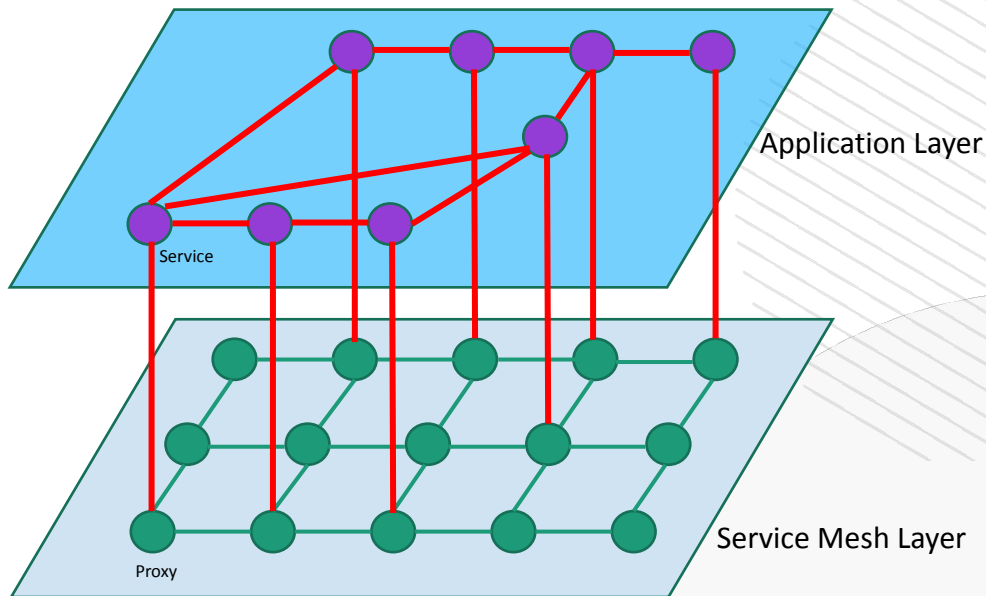
处理服务间通信(七层通信)的云原生基础设施层:

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



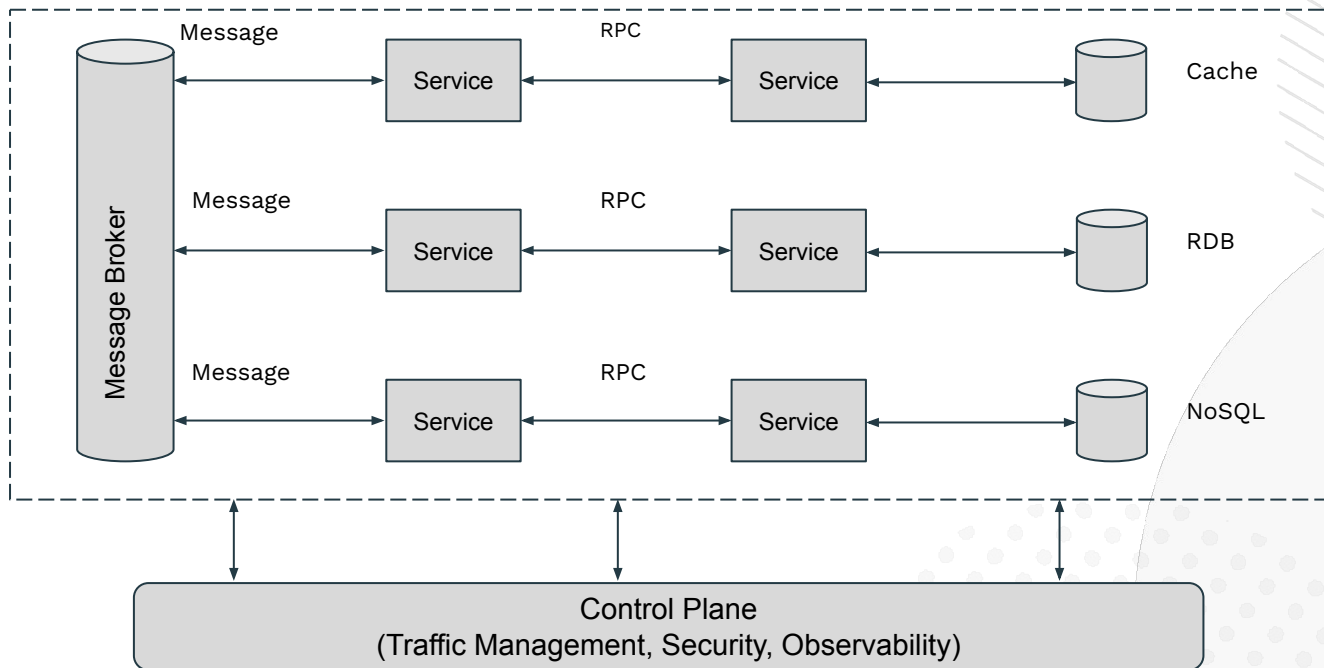
微服务中的常用七层协议

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:七层服务治理

- 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

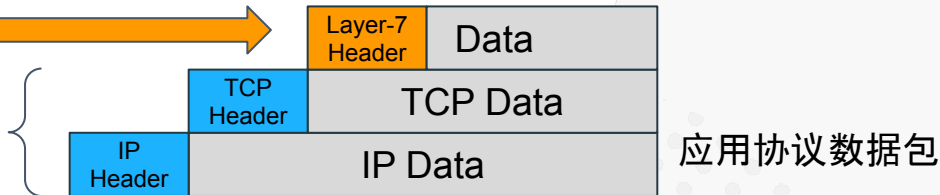
我们得到的网格能力:三/四层服务治理

- 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



Istio 协议扩展:控制面和数据面需要进行的改动

```
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
```



Istio 代码改动

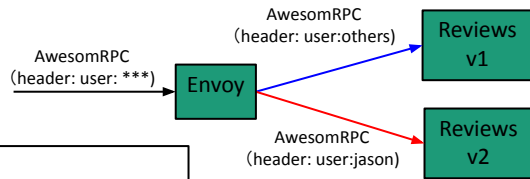
- 解析 CRD
- 生成 xDS 配置下发

```
{
  "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",
          },
        },
      ]
    }
  ],
}
```



AwesomeRPC Filter

- Decoding/encoding
- Parsing header
- Routing
- Load balancing
- Circuit breaker
- Fault injection
- Telemetry collecting



困难:

- Istio 目前缺少一个良好的协议扩展机制
- Istio 需要理解 Envoy filter 中协议特定的知识
- Istio 代码中维护众多七层协议的代价较大

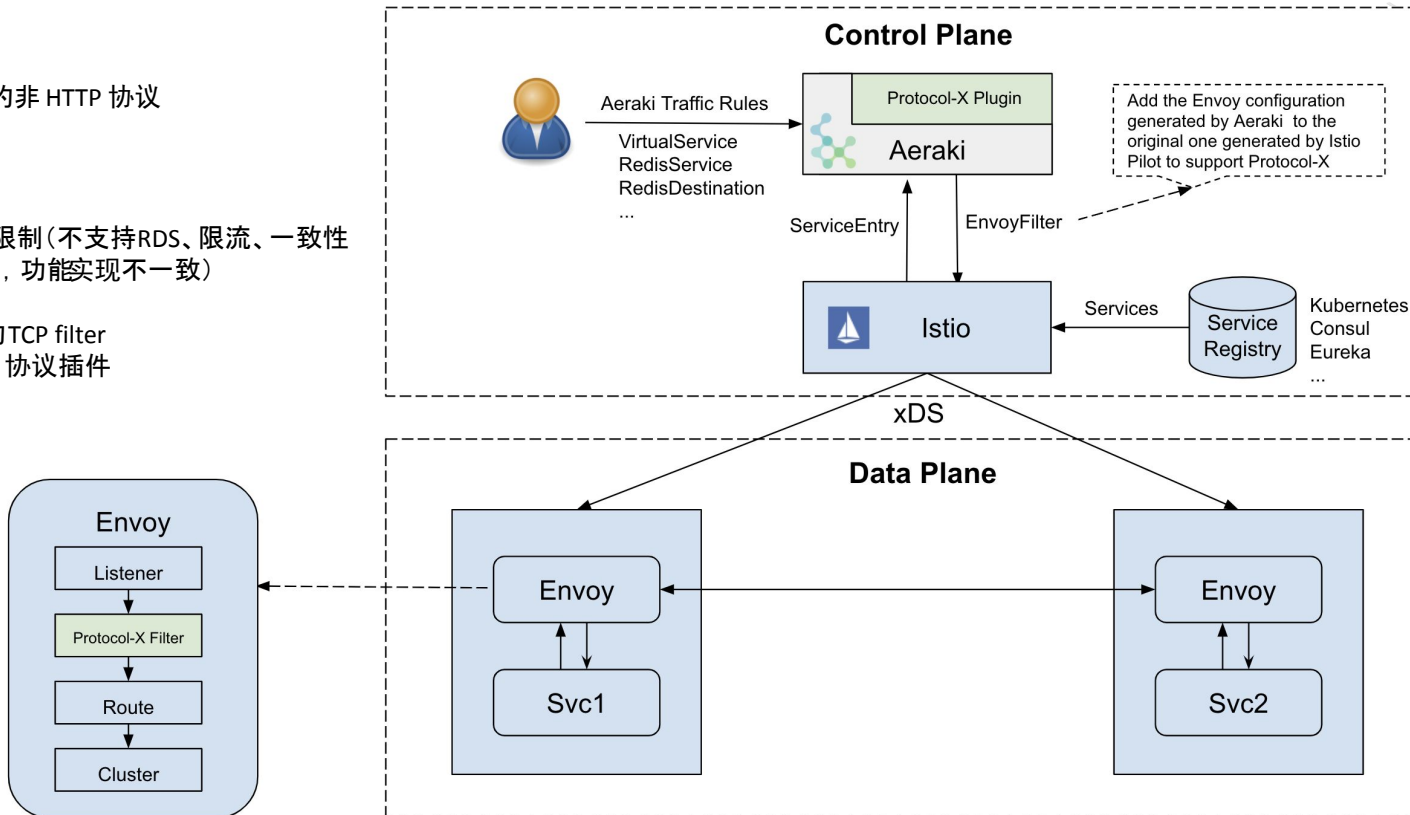
Aeraki 第一版架构

能力:

- 和 Istio 无缝集成, 对 Istio 无侵入
- 可以支持 Envoy 中已经提供实现的非 HTTP 协议 (Dubbo, Thrift, Redis等)

问题:

- Envoy Dubbo, Thrift Filter 的功能限制 (不支持RDS、限流、一致性 Hash、流量镜像、调用跟踪等能力, 功能实现不一致)
- 支持一个新的协议工作量很大
 - 数据面需要编写一个完整的TCP filter
 - 控制面需要开发一个 Aeraki 协议插件



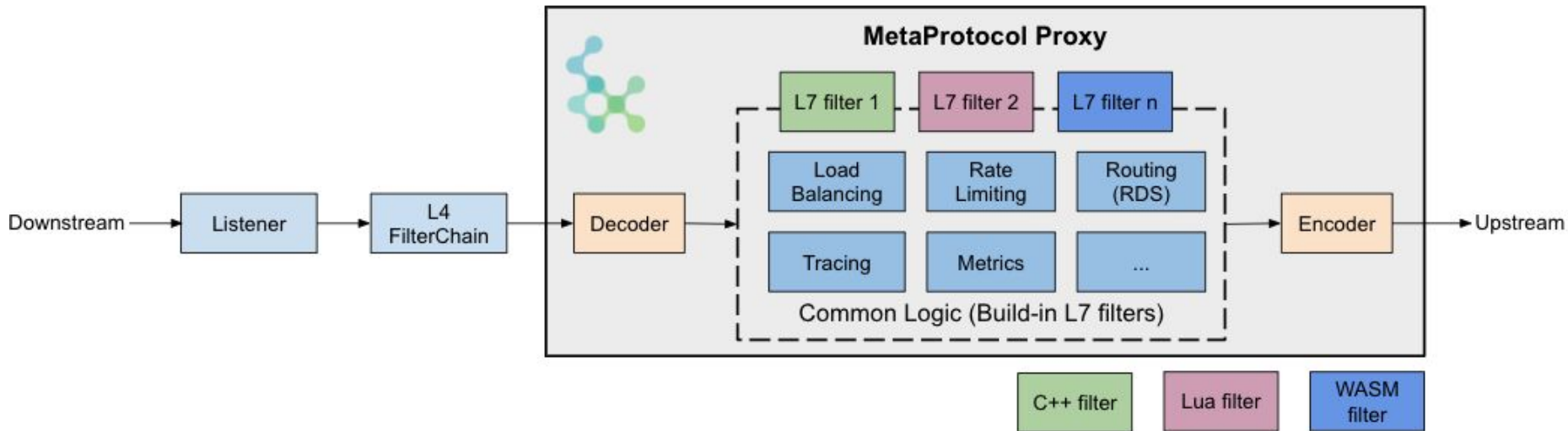
重复造轮子？常见七层协议流量管理的相似之处

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，例如认证授权等。



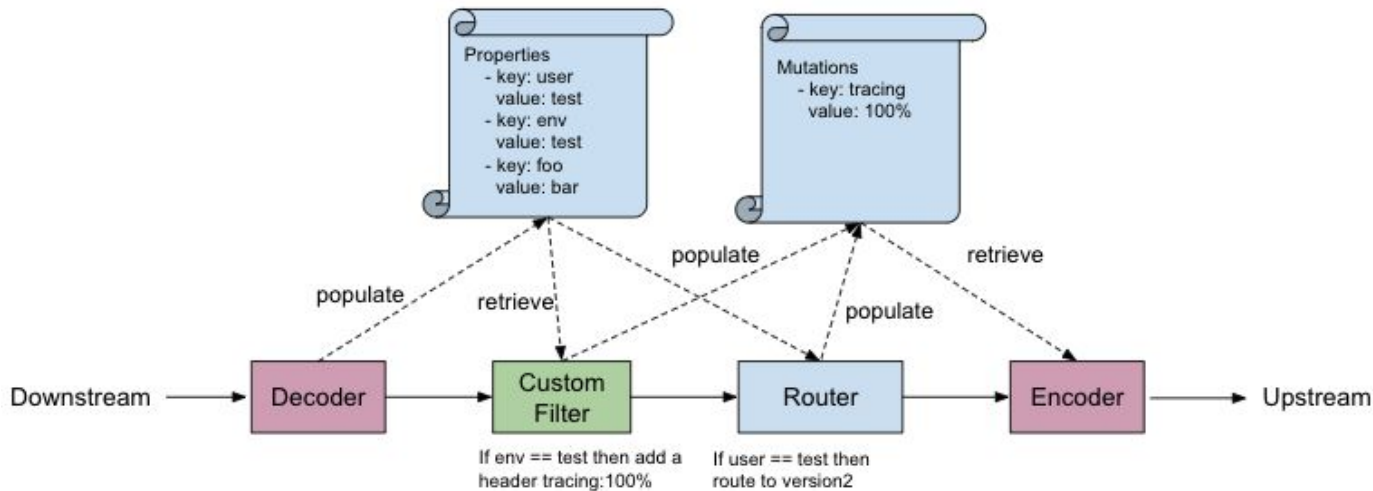
MetaProtocol: 请求处理路径

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:

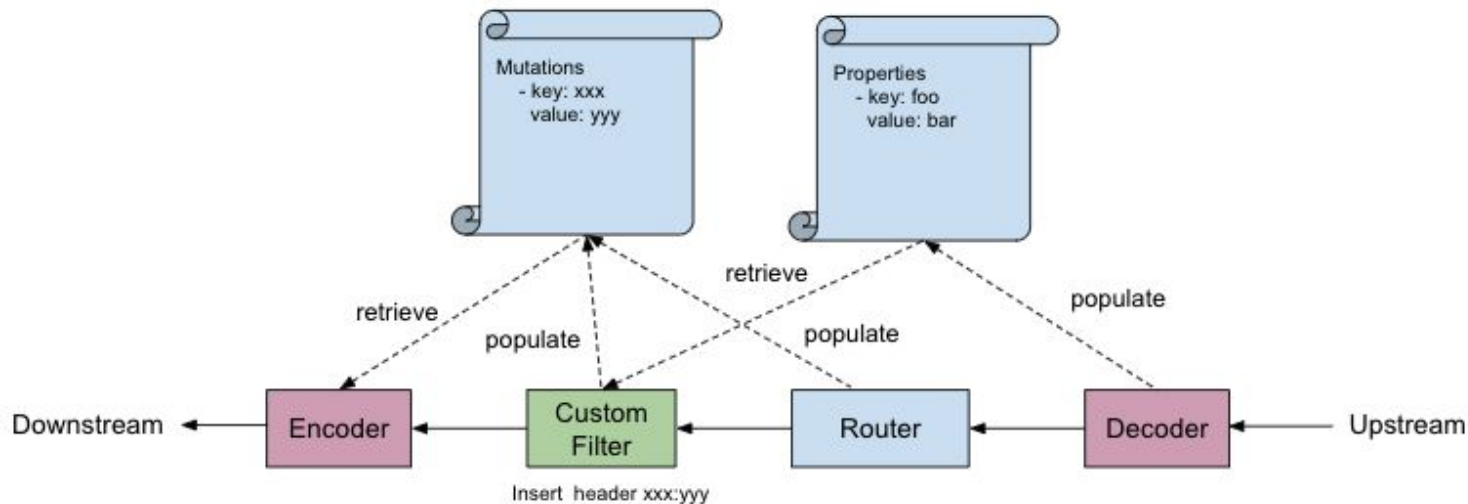
- Metadata: decode 时填充的 key:value 键值对, 用于 L7 filter 的处理逻辑中
- Mutation: L7 filter 填充的 key:value 键值对, 用于 encode 时修改请求数据包



MetaProtocol: 响应处理路径

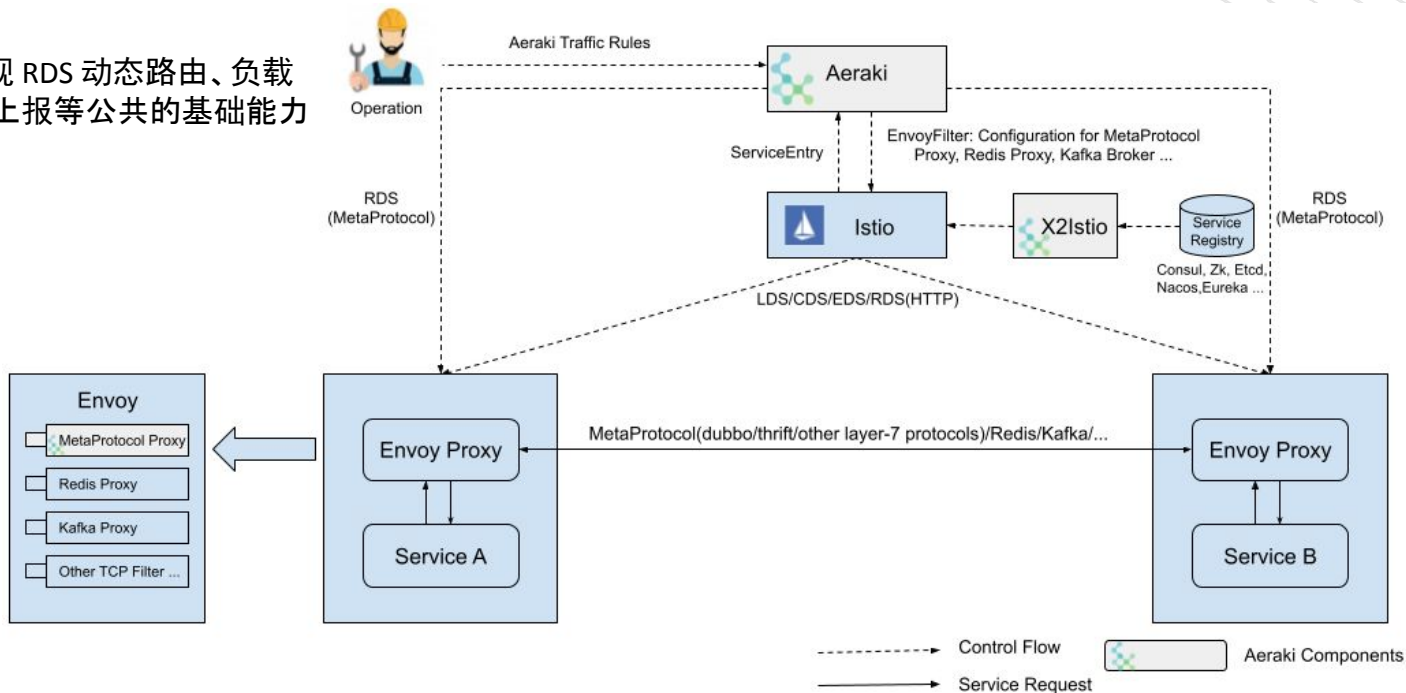
1. 处理流程:
2. Decoder 解析 Upstream 的响应, 填充 Metadata
3. Router 根据 connection/stream 对应关系找到响应的 Downstream 连接
4. L7 filter 从 Metadata 获取所需的数据, 进行响应方向的业务处理
5. L7 filter 将需要修改的数据放入 Mutation 结构中
6. Encoder 根据 Mutation 结构封包
7. 将响应发送到 Downstream

- 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 最新架构 (Aeraki + MetaProtocol Proxy 双剑合璧 → Aeraki Mesh)

- 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



如何基于 Aeraki Mesh 开发一个自定义协议？

1. 实现编解码接口 codec (约数百行代码)
2. 定义一个 ApplicationProtocol

```
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 to indicate
     * can't find the specified cluster, or there is no healthy
     *
     * @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) PURE;
};
```

```
apiVersion: metaprotocol.aeraki.io/v1alpha1
kind: ApplicationProtocol
metadata:
  name: my-protocol
  namespace: istio-system
spec:
  protocol: my-protocol
  codec: aeraki.meta_protocol.codec.my_protocol
```

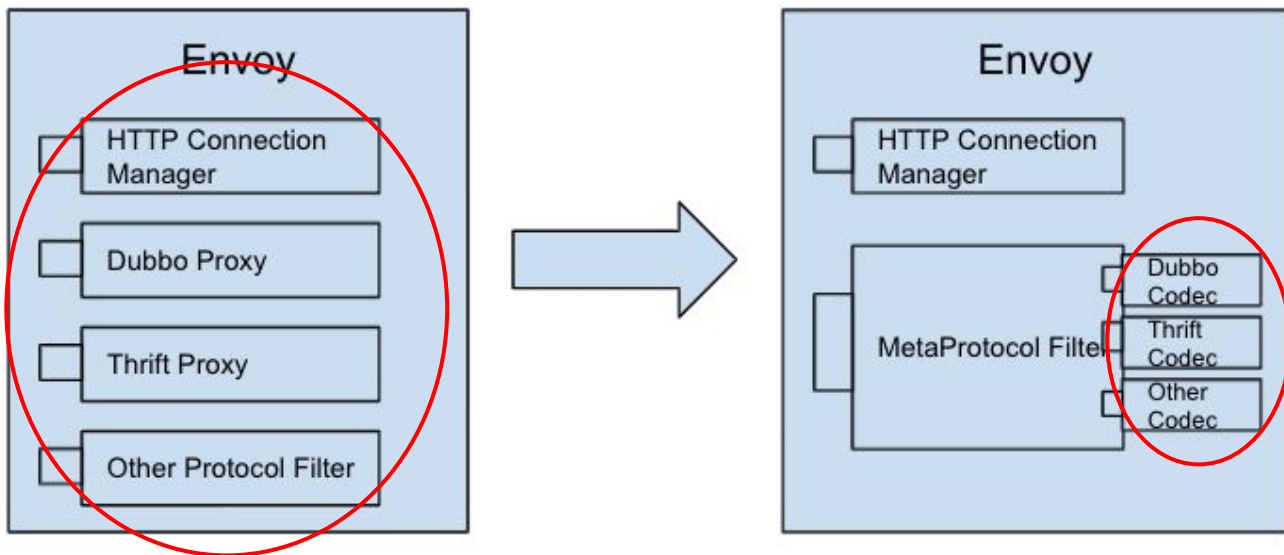
在 Service Mesh 中管理一个私有协议的所需工作量对比(采用/不采用 Aeraki Mesh)

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



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
```


Aeraki Mesh 项目当前进展

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 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.

Orchestration & Management - Service Mesh (15)

Project	Stars	Market Cap	Funding
Aeraki Mesh	371	\$11	
AWS App Mesh	157		
Consul	24,350	\$8.8B	
EaseMesh	310		
GLASNOSTIC			\$2.1M
Gloo Mesh	144		\$171.5M
Grey Matter			
Istio	25,633	\$1.8T	
Kuma	2,619		\$3M
LINKERD	6,136		\$3M
MESHERY	1,245		\$3M
Open Service Mesh	2,209		\$3M
Service Mesh Interface	307		\$3M
Service Mesh Performance	211		\$3M
traefik mesh	1,615		\$11.1M

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 zhao_huabing 进群

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

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

Istio + Aeraki Mesh在腾讯音乐的 服务网格落地

腾讯音乐 · 王诚强



个人简介

王诚强 whitefirer@gmail.com

目前为腾讯音乐高级工程师
主要负责基础架构的开发建设
推动服务上云并云原生化
主导服务网格在业务的应用
建设完善的微服务管理体系

目录

- ❑ 项目背景
- ❑ 私有协议支持
- ❑ 外部服务发现
- ❑ 多控制面服务网格合并
- ❑ Aeraki服务治理应用
- ❑ 总结与规划



01. 音乐社交 黏度高受众广

腾讯音乐的业务具有用户黏度高且受众广的特点，对于服务的稳定性较为敏感，同时发展较久较快，也遗留了不少历史问题。



02.业务复杂

系统随着业务的快速发展，日趋复杂。

- 历史问题多
- 服务治理难
- 多协议
- 多Mesh/非Mesh
- 多服务发现
- 多语言、框架
- HTTP/gRPC/其它私有协议
- Istio1.3.6/Istio1.10/其它异构Mesh
- L5/Polaris/Consul/DNS
- C++/Golang/Node.js/...
- Kubernetes/VM

03.业务期望

- 平滑迁移, 业务无感知
- 业务代码改动少、低侵入
- 流量透明、可控
- 支持私有协议、多种服务发现接入
- 兼容、适用性强

04. 技术选型



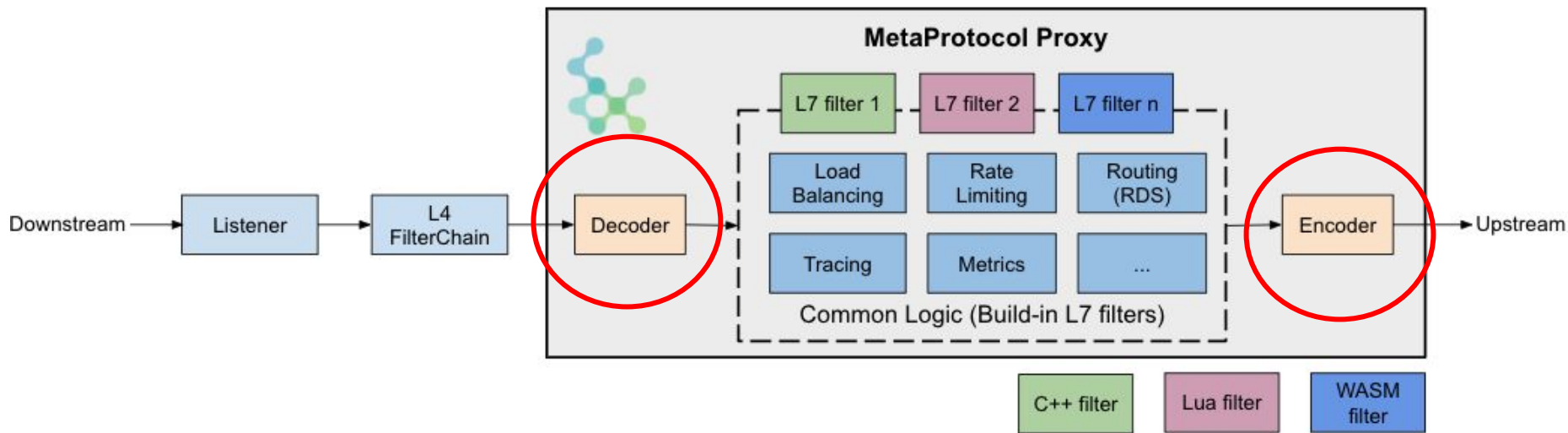
- **通用性** 基于边车的设计，适用于不同的业务框架
- **兼容性** Aeraki作为第二控制面，无需修改stio控制面
- **易用性** 借助Aeraki MetaProtocol，轻松实现私有协议支持
- **完备性** 功能完备，并对原有功能增强
- **可持续性** 基于社区能力，不断迭代更新

私有协议支持

Istio虽然可以通过实现EnvoyFilter来进行私有协议的解析，但同时也存在几个问题：

1. 实现协议过于复杂；
2. 需要专门实现控制面；

使用Aearki则可避免以上问题，仅需完成几百行的编解码接口，即可拥有完整的网格治理能力。



私有协议支持

只需实现:

- 1、解码(decode)
- 2、编码(encode)
- 3、请求错误处理(onError)

```
namespace Envoy {
namespace Extensions {
namespace NetworkFilters {
namespace MetaProtocolProxy {
namespace Qza {

enum class QzaDecodeStatus { ...
};

/**
 * Codec for Qza protocol.
 */
class QzaCodec : public MetaProtocolProxy::Codec,
                 public Logger::Loggable<Logger::Id::misc> {
public:
    QzaCodec() {};
    ~QzaCodec() override = default;

    MetaProtocolProxy::DecodeStatus decode(Buffer::Instance& buffer,
                                         MetaProtocolProxy::Metadata& metadata) override;
    void encode(const MetaProtocolProxy::Metadata& metadata,
               const MetaProtocolProxy::Mutation& mutation, Buffer::Instance& buffer) override;
    void onError(const MetaProtocolProxy::Metadata& metadata, const MetaProtocolProxy::Error& error,
                 Buffer::Instance& buffer) override;

protected: ...
private: ...
};

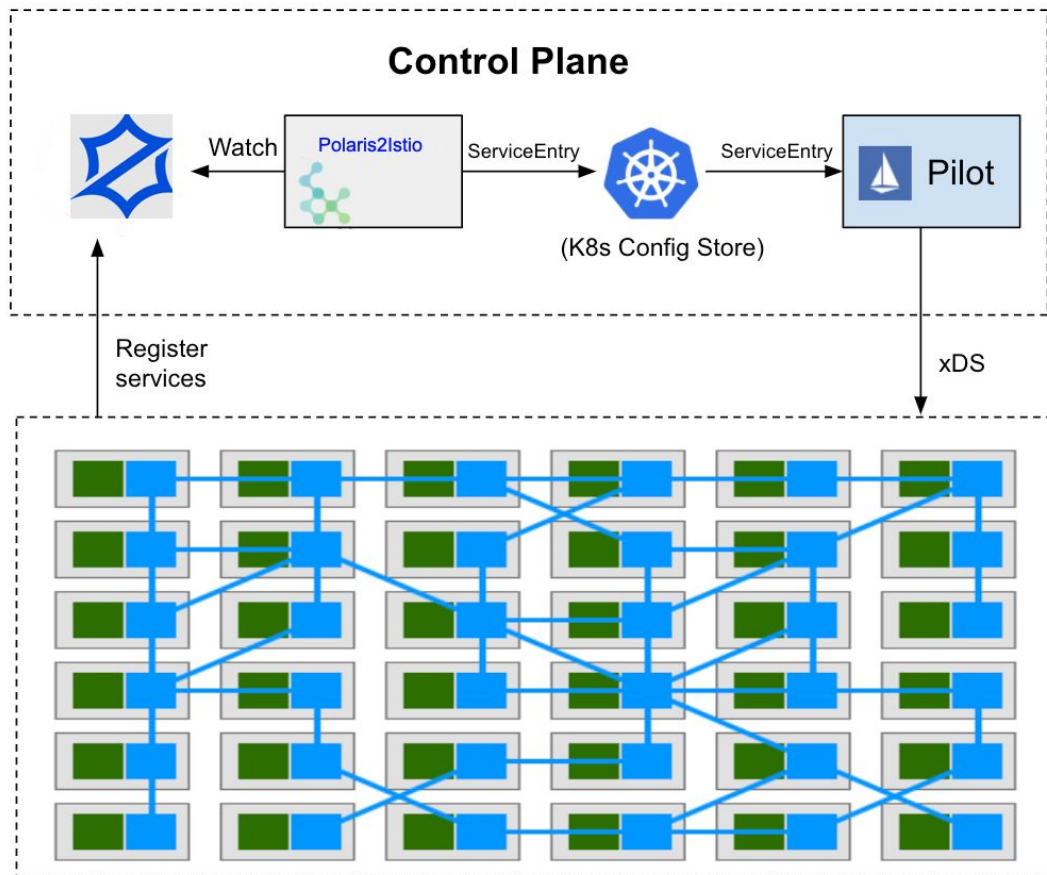
} // namespace Qza
} // namespace MetaProtocolProxy
} // namespace NetworkFilters
} // namespace Extensions
} // namespace Envoy
```

外部服务发现

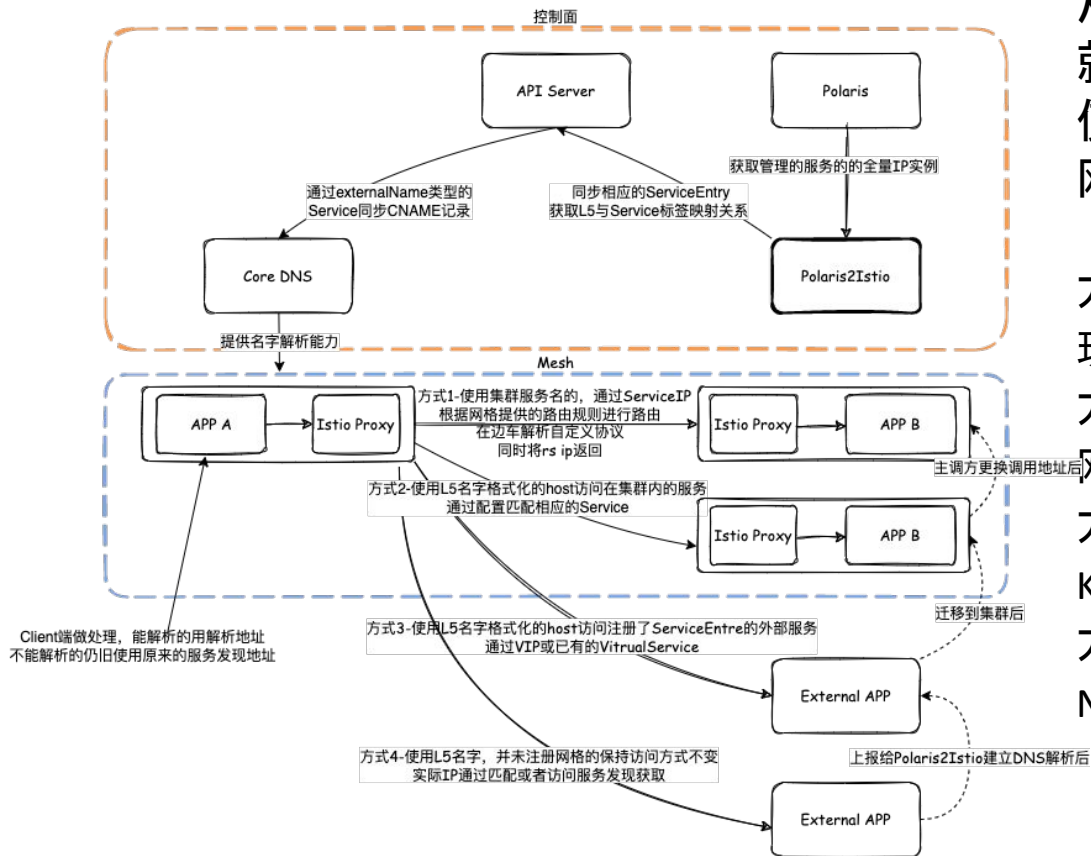
服务发现以北极星 (Polaris) 为例：

- 首先通过Polaris-Controller组件自动注册到北极星。
- 然后编写了Polaris2Istio组件来watch北极星的变更，并同步到Istio Mesh里的ServiceEntry当中
- 最后再由Pilot通过xDS下发到数据面。

Aeraki当中还提供了其它类型的服务发现接入方式，如Consul2Istio、Dubbo2Istio、Eurake2Istio等。



外部服务发现



从方式4一步步自动迁移到方式1, 我们就实现了服务发现的网格内外兼容, 以使我们的服务平滑地从网格外迁移到了网格内。

方式4: 服务在网格外, 仍然用的旧的服务发现

方式3: 服务在网格外, 用旧的服务名并使用网格内的ServiceEntry接入

方式2: 服务在网格内, 用旧的服务名并指向K8S Service

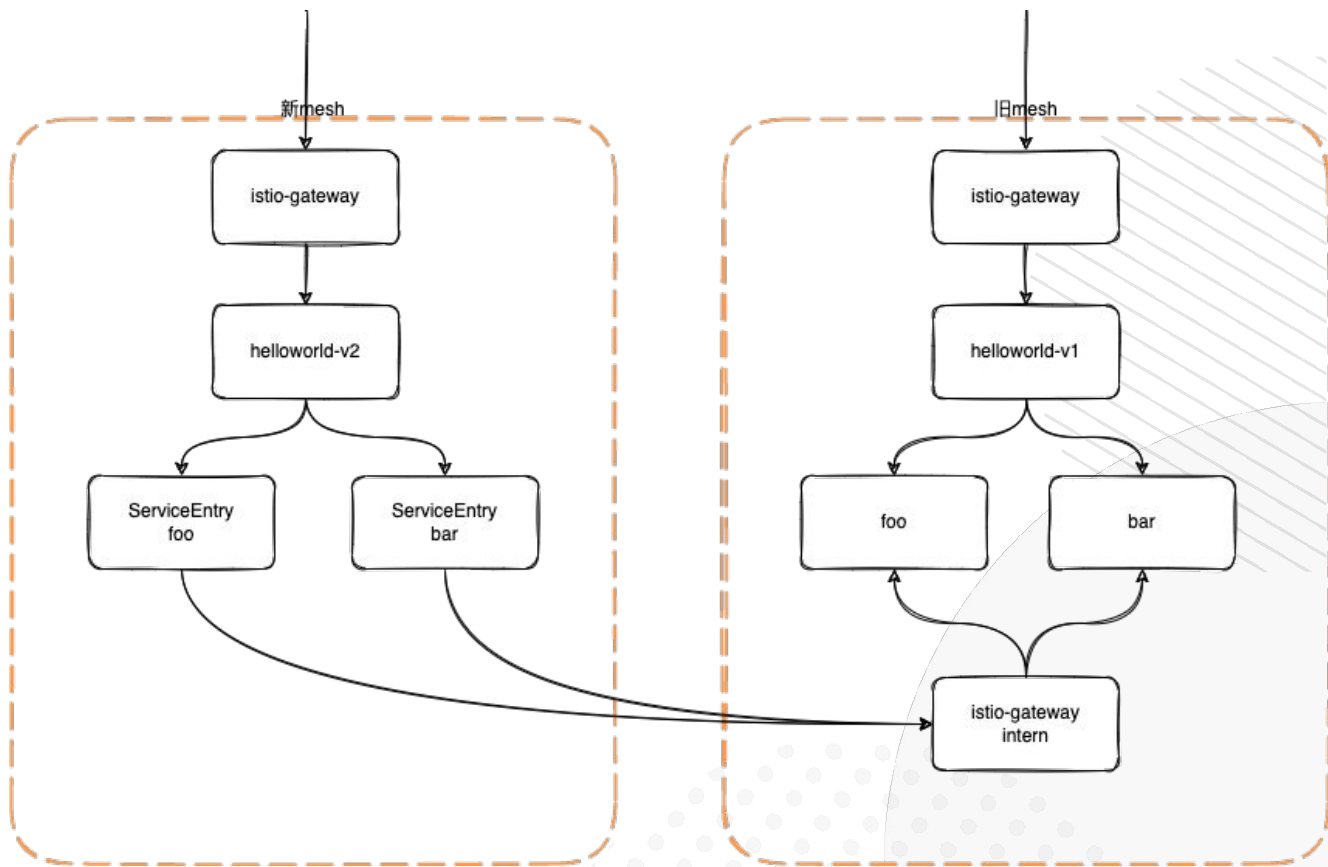
方式1: 服务在网格内直接使用K8S Service Name, 旧服务名作为别名

多控制面服务网格合并

一阶段：

1. 创建回旧mesh的 gateway；
2. 创建相应的 VirtualService；
3. 创建指回旧mesh gateway的 ServiceEntry；
4. 验证网络；
5. 迁移服务到新mesh；

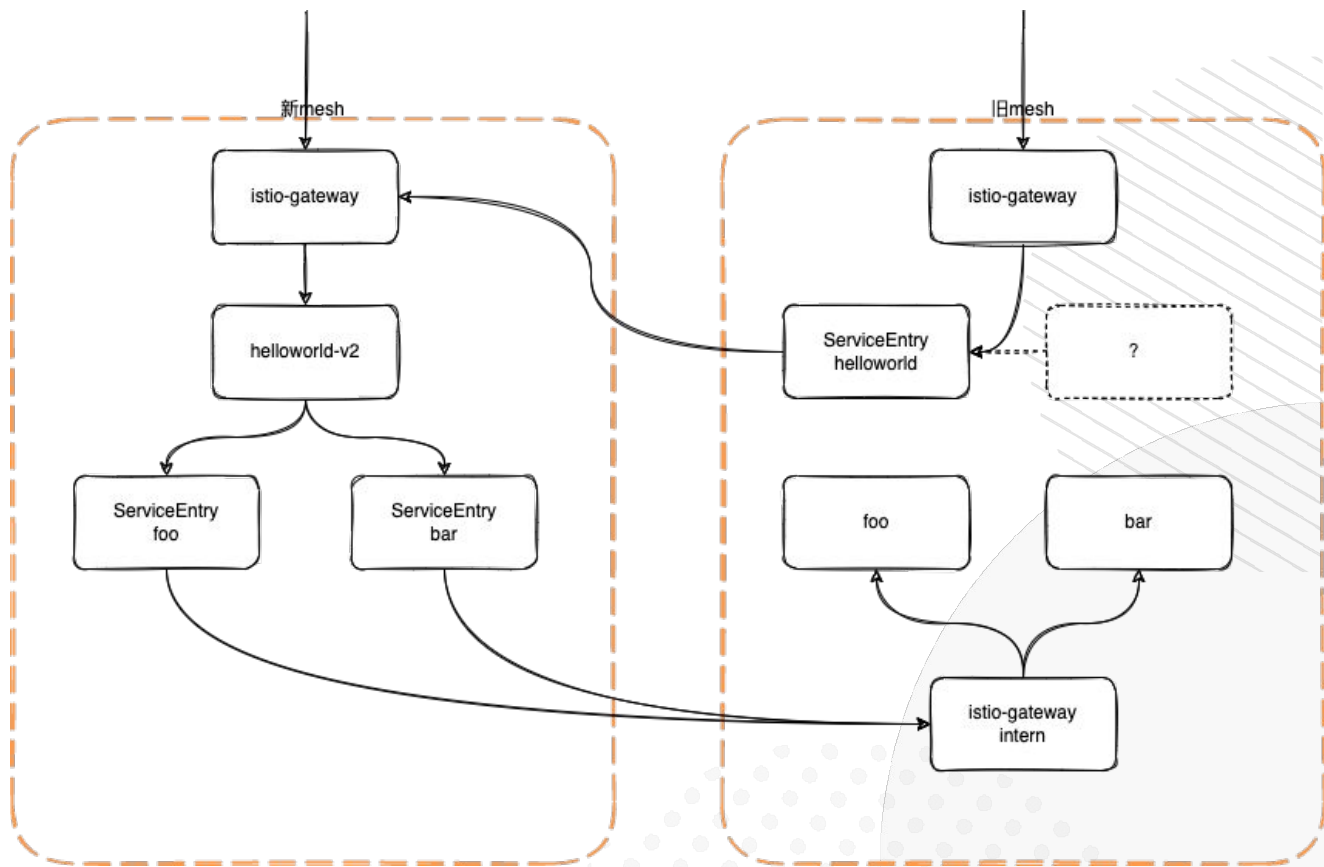
此处ServiceEntry可编程同步
自动创建



多控制面服务网格合并

二阶段：

1. 流量权重改到新mesh；
2. 在旧mesh创建指向新mesh gateway的 ServiceEntry用于占位；
3. 卸载在旧Mesh的服务；



多控制面服务网格合并

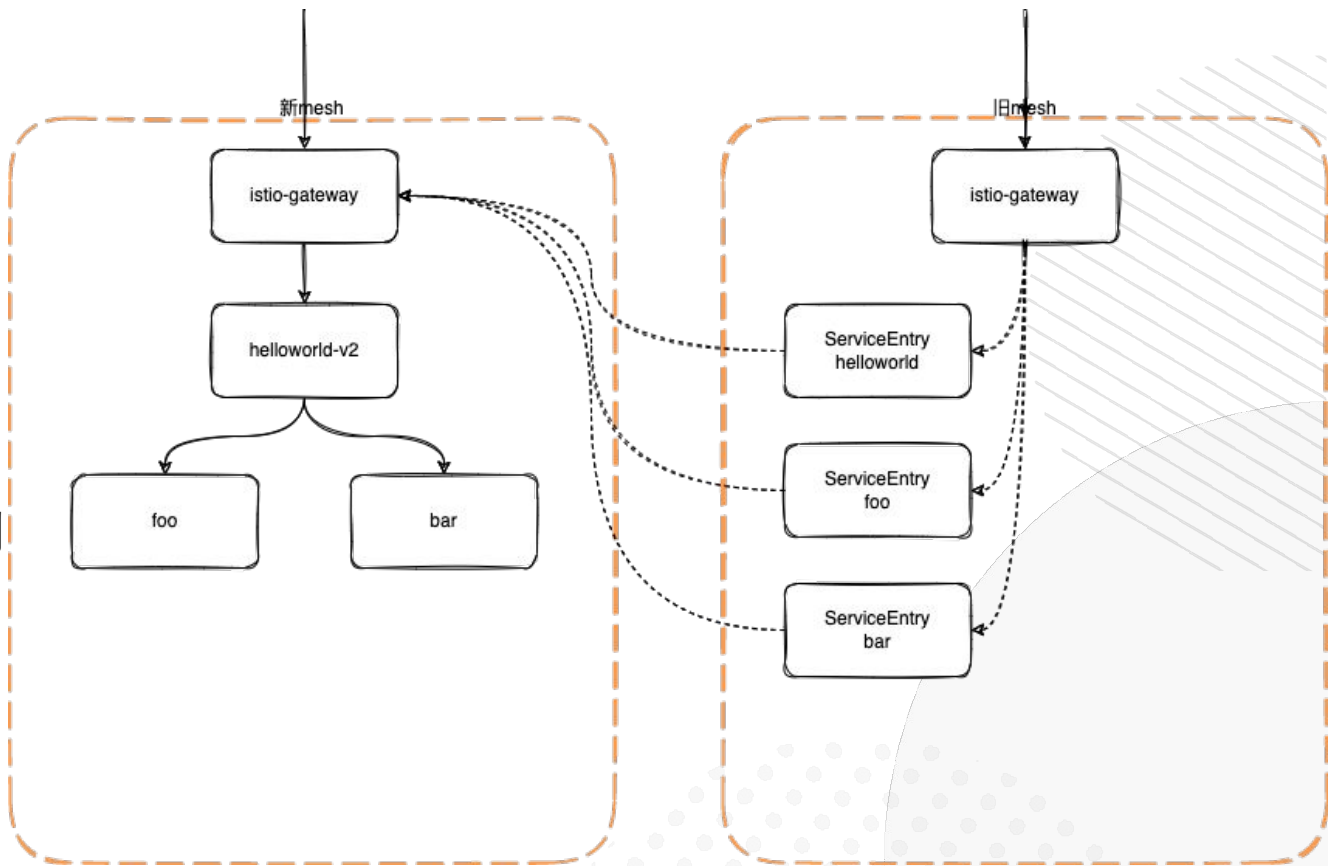
三阶段：

1. 移除指向旧Mesh的 ServiceEntry；

最终结果，所有服务均迁到了新mesh。

实际操作中还要注意其它问题，如：

- 资源情况；
- 网络互通；



Aeraki服务治理应用--流量分发

```
1 apiVersion: metaprotocol.aeraki.io/v1alpha1
2 kind: MetaRouter
3 metadata:
4   name: test-metaprotocol-qza-route
5 spec:
6   hosts:
7     - qza-sample-server.meta-qza.svc.cluster.local
8   routes:
9     - name: v1
10       match:
11         attributes:
12           cmd:
13             exact: "153"
14           sub_cmd:
15             regex: "1|2"
16       route:
17         - destination:
18             host: qza-sample-server.meta-qza.svc.cluster.local
19             subset: v1
```

```
1 apiVersion: metaprotocol.aeraki.io/v1alpha1
2 kind: MetaRouter
3 metadata:
4   name: test-metaprotocol-qza-route
5 spec:
6   hosts:
7     - qza-sample-server.meta-qza.svc.cluster.local
8   routes:
9     - name: v1
10       match:
11         attributes:
12           cmd:
13             exact: "153"
14           sub_cmd:
15             regex: "1"
16       route:
17         - destination:
18             host: qza-sample-server.meta-qza.svc.cluster.local
19             subset: v1
20+
21+     - name: v2
22+       match:
23+         attributes:
24+           cmd:
25+             exact: "153"
26+           sub_cmd:
27+             regex: "2"
28+       route:
29+         - destination:
30+             host: qza-sample-server.meta-qza.svc.cluster.local
31+             subset: v2
```

Aeraki服务治理应用--流量分发

变更路由前：

流量不区分命令字全都往v1

变更路由后：

流量严格匹配命令字前往对应版本

意味着私有协议它也有和http/gRPC等通用协议一样的流量分发能力，我们可以据此实现全链路染色，实现

1. 开发环境按分支版本隔离；
2. 生产环境按业务需求区分版本；

```
→ aeraki-ops k apply -f test-metaprotocol-qza-route.yaml
metarouter.metaprotocol.aeraki.io/test-metaprotocol-qza-route configured
→ aeraki-ops k logs -f qza-sample-client-65c956fd4b-gt6kv --tail=10
Request cmd: 0x99/2, seq: 67; Response ask: 66, label: v1
Request cmd: 0x99/2, seq: 66; Response ask: 65, label: v1
Request cmd: 0x99/2, seq: 65; Response ask: 64, label: v1
Request cmd: 0x99/2, seq: 64; Response ask: 63, label: v1
Request cmd: 0x99/1, seq: 63; Response ask: 64, label: v1
Request cmd: 0x99/2, seq: 64; Response ask: 63, label: v1
Request cmd: 0x99/2, seq: 63; Response ask: 62, label: v1
Request cmd: 0x99/2, seq: 62; Response ask: 61, label: v1
Request cmd: 0x99/2, seq: 61; Response ask: 60, label: v1
Request cmd: 0x99/1, seq: 60; Response ask: 61, label: v1
Request cmd: 0x99/2, seq: 61; Response ask: 60, label: v1
Request cmd: 0x99/1, seq: 60; Response ask: 61, label: v1
Request cmd: 0x99/2, seq: 61; Response ask: 60, label: v1
Request cmd: 0x99/2, seq: 60; Response ask: 59, label: v1
Request cmd: 0x99/1, seq: 59; Response ask: 60, label: v1
Request cmd: 0x99/2, seq: 60; Response ask: 59, label: v1
^C
→ aeraki-ops k apply -f test-metaprotocol-qza-traffic.yaml
metarouter.metaprotocol.aeraki.io/test-metaprotocol-qza-route configured
→ aeraki-ops k logs -f qza-sample-client-65c956fd4b-gt6kv --tail=10
Request cmd: 0x99/2, seq: 53; Response ask: 52, label: v2
Request cmd: 0x99/2, seq: 52; Response ask: 51, label: v2
Request cmd: 0x99/2, seq: 51; Response ask: 50, label: v2
Request cmd: 0x99/1, seq: 50; Response ask: 51, label: v1
Request cmd: 0x99/1, seq: 51; Response ask: 52, label: v1
Request cmd: 0x99/2, seq: 52; Response ask: 51, label: v2
Request cmd: 0x99/1, seq: 51; Response ask: 52, label: v1
Request cmd: 0x99/1, seq: 52; Response ask: 53, label: v1
Request cmd: 0x99/2, seq: 53; Response ask: 52, label: v2
Request cmd: 0x99/1, seq: 52; Response ask: 53, label: v1
Request cmd: 0x99/1, seq: 53; Response ask: 54, label: v1
Request cmd: 0x99/1, seq: 54; Response ask: 55, label: v1
```

Aeraki服务治理应用--限流

限流分为本地限流和全局限流

本地限流是针对单个pod的，会随pod数增长而总体规模增长，如图中限流为，pod数为2的话，那一分钟内前4次请求成功，其余失败。

```
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v1
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v1
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v2
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v2
Request cm: 0x99/2, Response err: (code:51, msg:)
Request cm: 0x99/2, Response err: (code:51, msg:)
Request cm: 0x99/2, Response err: (code:51, msg:)
```

```
1  apiVersion: metaprotocol.aeraki.io/v1alpha1
2  kind: MetaRouter
3  metadata:
4    name: test-metaprotocol-qza-route
5  spec:
6    hosts:
7      - qza-sample-server.meta-qza.svc.cluster.local
8+   localRateLimit:
9+     tokenBucket:
10+      fillInterval: 60s
11+      maxTokens: 2
12+      tokensPerFill: 2
13  routes:
14    - name: v1
15      match:
16        attributes:
17          cmd:
18            exact: "153"
19            sub_cmd:
20              regex: "1|2"
21        route:
22          - destination:
23              host: qza-sample-server.meta-qza.svc.cluster.local
24              subset: v1
```

Aeraki服务治理应用--限流

当然也可以根据条件来限流，比如图中就是只限于命令为2的请求，而子命令的则正常通过。

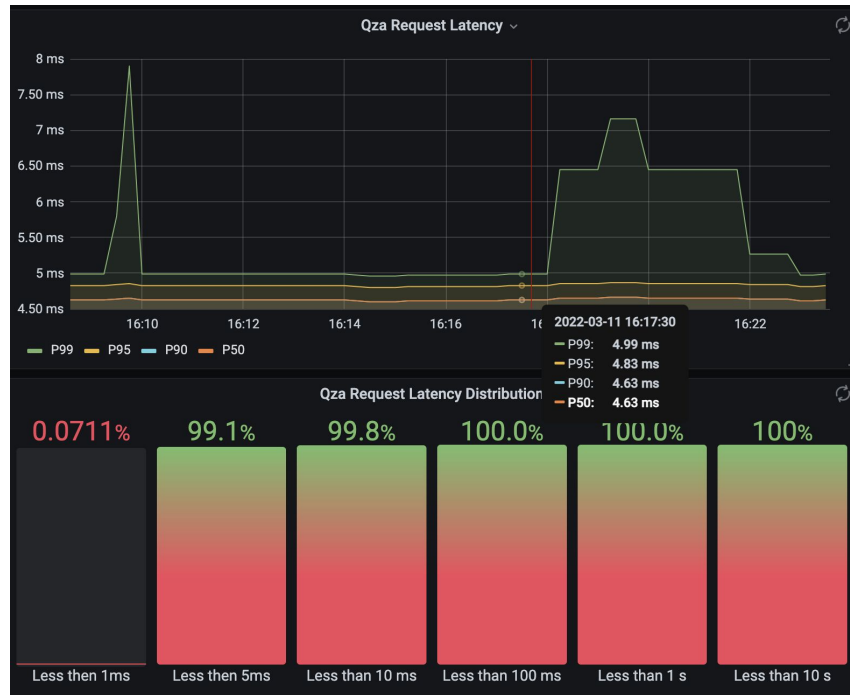
```
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v1
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v1
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v2
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v2
Request cm: 0x99/2, Response err: (code:51, msg:)
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v1
Request cm: 0x99/1, seq: 10; Response ask: 11, label: v1
Request cm: 0x99/2, Response err: (code:51, msg:)
Request cm: 0x99/2, Response err: (code:51, msg:)
{Request cm: 0x99/1, seq: 10; Response ask: 11, label: v1
```

```
1  apiVersion: metaprotocol.aeraki.io/v1alpha1
2  kind: MetaRouter
3  metadata:
4    name: test-metaprotocol-qza-route
5  spec:
6    hosts:
7      - qza-sample-server.meta-qza.svc.cluster.local
8+   localRateLimit:
9+     conditions:
10+    - match:
11+      attributes:
12+        cmd:
13+          exact: "153"
14+          sub_cmd:
15+            regex: "2"
16+      tokenBucket:
17+        fillInterval: 60s
18+        maxTokens: 2
19+        tokensPerFill: 2
20  routes:
21    - name: v1
22      match:
23        attributes:
24          cmd:
25            exact: "153"
26            sub_cmd:
27              regex: "1|2"
28        route:
29          - destination:
30            host: qza-sample-server.meta-qza.svc.cluster.local
31            subset: v1
```

其它应用场景

除上面演示的几种应用场景外, aeraki还支持了自定协议的度量指标(如图)、全局限流、负载均衡、熔断和修改消息头等。因时间关系, 就不一一演示了。

可以通过[Aeraki官方教程](#)了解。



总结与规划

Aeraki很好地帮助我们扩展了Istio的能力，特别是以下几点：

1. 协议的扩展；
2. 多种服务发现的接入；
3. 无侵入性地兼容异构系统；

项目往往是复杂多变的，不太可能完全按理想情况下进行，借助Aeraki，使得各种异构系统、自定义协议、服务发现等能以较简单的形式统一以Istio服务网格进行管理，大大解放了业务的生产力。

未来规划

- 相关业务全面覆盖
- Tracing全面打通
- 应用管理、服务治理平台化

Thank you!

#IstioCon

