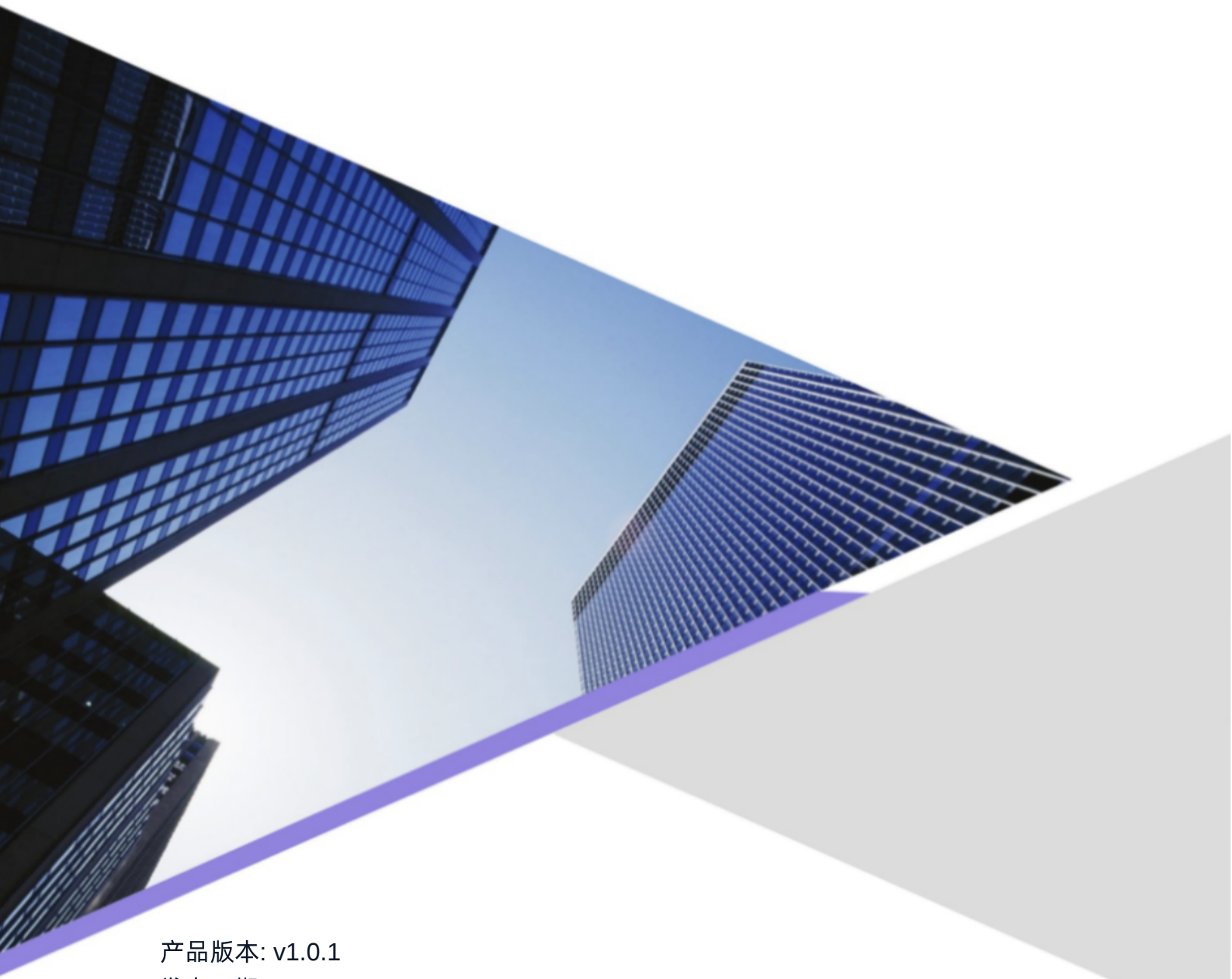
易捷行云
EasyStack

服务网格

使用手册



产品版本: v1.0.1
发布日期: 2024-02-05

目录

1 版本说明

1.1 版本说明书

版本信息

产品名称	产品版本	发布日期
服务网格	V1.0.1	2023-09-06

更新说明

新增功能

- 对数字原生引擎 EOS 6.1.1 版本进行适配，新增服务网格云产品部署
- 新增流量治理功能，包括有以下方面：
  - 灰度发布:可以逐步将流量切换到新版本,逐步验证新版本的稳定性
  - A/B 测试:可以按照指定百分比把流量分配到不同版本,进行 A/B 对比测试
  - 限流 & 熔断:可以根据阈值设置限流或者失败熔断,保护服务的可用性
  - 负载均衡:可以按照预设规则进行流量负载均衡,提高服务的可用性
  - 健康检查：可以按照规则配置主动健康检查，也可以配置异常值检测，当前遵循 http/grpc 协议描述，使用 http code和gcrp-status 鉴别服务端异常
  - 连接池：支持配置 L4 / L7 层流量最大的连接数
  - 会话保持：可以按照规则配置会话，如header，query和cookie等
  - 按需加载: 按照预设的规则，不会将非本命名空间的集群和路由信息下发给sidecar
  - 零信任安全：按照预设的规则，默认使用双向认证鉴别用户，当不支持双向认证，会自动降级为 raw 模式
- 新增多协议支持，包括以下方面：
  - Dubbo 2.0: 可以通过 Sidecar 代理 和定义dubbo2注册中心的方式支持

  - Spring Cloud: 可以通过 Sidecar 代理的方式支持 Spring Cloud 的服务调用
- 新增可观测性能力，默认提供指标，链路和日志收集
  - 日志:Istio 可以收集 Sidecar 代理和服务容器的日志,用于 debugging 和监控
  - 指标:Istio 提供 Prometheus 格式的指标,可以设置自定义指标,用于性能监控
  - 链路追踪:Istio 兼容 Zipkin 和 Jaeger,可以进行分布式链路追踪,用于分析服务调用性能和追踪问题根因
- 新增证书管理能力，包括以下方面
  - 对sidecar 的提供证书颁发和续签能力

依赖说明

> - 平台版本至少为V6.1.1
> - 平台监控告警服务至少为 v6.2.1

2 产品介绍

2.1 什么是服务网格

服务网格产品基于成熟的 istio 和 cert-manager 实现，提供卓越的基于服务身份的零信任网络,提供微服务应用的流量治理，可观测性和安全基础建设。

让服务间的网络调用更可靠、更安全、更易于监控。像网络中的“交换机”一样,负责处理服务间大量的网络调用，让开发人员更加关注业务逻辑的实现，而基础设施层的网络问题由服务网格组件来处理，以提高微服务应用的性能和稳定性。

产品优势

- **流量控制**
- 支持细粒度的流量控制,如金丝雀发布、A/B 测试、故障注入等
- 提供限流、熔断、重试、负载均衡等能力,提高服务稳定性
- **安全保障**
- 支持服务间 mTLS 认证和加密通信
- 提供细粒度的访问控制和权限管理
- **可观测性**
- 提供聚合的指标、日志和追踪,可以全局观察服务间调用
- 方便定位系统瓶颈和异常,优化服务性能
- **语言无关**
- 作为独立基础层,与业务语言无关
- 降低了语言之间调用的复杂度
- **云原生友好**
- 与 Kubernetes、Istio 等云原生技术无缝集成

- 方便在动态和分布式的环境中管理服务
- **与业务解耦**
- 服务网格独立于业务代码之外,减少了侵入性
- 可以将诸如监控、安全等非业务逻辑提炼出来

2.2 使用场景

多语言应用微服务治理

无需修改代码，服务网格就能为客户提供金丝雀发布、无损上下线、服务鉴权、标签路由等业务应用微服务治理能力，支持与Nacos服务注册中心打通，并提供与异构服务框架如SpringCloud的互通能力。

解决问题：

- 业务代码与治理功能紧耦合，不利于各组件独立快速迭代，服务网格可将治理能力独立出来，实现无侵入的服务治理
- 编程语言及框架的多样化，引入了不同的服务注册中心及治理策略，可利用服务网格实现注册中心的互通，统一治理体系

多集群应用统一流量管理

业务应用部署在多地域或混合云下的Kubernetes集群中，存在一致的可见性和流量管理等需求。服务网格可以为跨类型的计算基础设施构建的服务提供一致的流量管理。

解决问题：

- 部署在异构基础设施上的业务负载涉及Kubernetes、虚拟机等不同的运行环境，可进行统一的流量治理
- 能够以最佳方式将流量路由至某个服务位于多个地域的应用实例，可助力用户实现Active-Active的双活方案，或者Active-Standby的灾备方案

应用容器化平滑上云

线下环境有存量应用需要迁移上云，通过部署和配置服务网格，可以将流量动态路由到线下旧版环境或线上新版环境，较好地处理无状态服务迁移。

解决问题：

- 用户在搬站或上云过程中，涉及到测试、灰度、投产等多个阶段，应用流量控制的策略繁杂，可利用简化整个过程
- 对于有多云或混合云战略的用户，为应用服务提供了全局负载均衡能力，同时也支持服务就近访问

服务监控

增强容器、评估API端点的性能，为网格内的服务通信生成详细的遥测，这种遥测技术提供了服务行为的可观察性，允许运营商对其应用程序进行故障排除、维护和优化，而不会给服务开发人员带来任何额外负担。通过应用服务网格，运营商可以全面了解被监控的服务如何与其他服务以及组件本身进行交互。

解决问题：

- 非侵入监控数据采集：在复杂应用的场景下，服务间的访问拓扑，调用链，监控等都是对服务整体运行状况进行管理，服务访问异常时进行定位定界的必要手段。服务网格技术的一项重要能力就是以应用非侵入的方式提供这些监控数据的采集，用户只需关注自己的业务开发，无需额外关注监控数据的生成。
- 灵活的服务运行管理：在拓扑图上通过服务的访问数据，可以直观的观察到服务的健康状况，服务间的依赖情况。并且可以对关心的服务进行下钻，从服务级别下钻到服务版本级别，还可以进一步下钻到服务实例级别。通过实例级别的拓扑可以观察到配置了熔断规则后，网格如何隔离故障实例，使其逐渐接收不到流量。并且可以在故障实例正常时，如何进行实例的故障恢复，自动给恢复的实例重新分配流量

2.3 基本概念

工作负载

工作负载即Kubernetes对一组Pod的抽象模型，用于描述业务的运行载体，包括Deployment、Statefulset、Job、Deamonset等。

- 无状态工作负载（即Kubernetes中的“Deployments”）：Pod之间完全独立、功能相同，具有弹性伸缩、滚动升级等特性。如：Nginx、WordPress。
- 有状态工作负载（即Kubernetes中的“StatefulSets”）：Pod之间不完全独立，具有稳定的持久化存储和网络标示，以及有序的部署、收缩和删除等特性。如：mysql-HA、etcd。

实例（Pod）

Pod是Kubernetes部署应用或服务的最小的基本单位。一个Pod 封装多个应用容器（也可以只有一个容器）、存储资源、一个独立的网络 IP 以及管理控制容器运行方式的策略选项。

健康检查

主要分为主动健康检查和被动健康检查，主动健康检查配置 端点信息，如协议，主机和端口等。被动健康检查又被称为异常值检测，当前众多的协议中，如http ，grpc ，都有专门的配置字段code，表明错误的类型，被动健康检查根据异常值标记节点

金丝雀发布

又称灰度发布，是迭代的软件产品在生产环境安全上线的一种重要手段。在生产环境上引一部分实际流量对一个新版本进行测试，测试新版本的性能和表现，在保证系统整体稳定运行的前提下，尽早发现新版本在实际环境上的问题。

蓝绿发布

蓝绿发布提供了一种零宕机的部署方式。不停老版本，部署新版本进行测试，确认运行正常后，将流量切到新版本，然后老版本同时也升级到新版本。始终有两个版本同时在线，有问题可以快速切换。

流量治理

应用流量治理提供可视化云原生应用的网络状态监控，并实现在线的网络连接和安全策略的管理和配置，当前支持连接池、熔断、负载均衡、HTTP头域、故障注入等能力。

连接池管理

配置TCP和HTTP的连接和请求池相关阈值，保护目标服务，避免对服务的过载访问。

熔断

配置快速响应和隔离服务访问故障，防止网络和服务调用故障级联发生，限制故障影响范围，防止故障蔓延导致系统整体性能下降或者雪崩。

调用链分析

跟踪大规模复杂的分布式系统运行服务调用关系，解决分布式服务故障定位定界问题。

控制平面（Control Plane）

从架构设计上来看，Istio 服务网格逻辑上分为控制平面和数据平面两部分。控制平面负责管理和配置代理，从而实现路由流量。

数据平面（Data Plane）

数据平面由一组以 Sidecar 方式部署的智能代理（Envoy）组成，负责调节和控制微服务以及 Mixer 之间所有的网络通信。

虚拟服务（Virtual Service）

作为 Istio 自定义资源之一，虚拟服务（VirtualService）定义了一系列针对指定服务的流量路由规则。每个路由规则都针对特定协议定义流量匹配规则。如果流量符合这些特征，就会根据规则发送到服务注册表中的目标服务（或者目标服务的子集或版本）。

目标规则（Destination Rule）

作为 Istio 自定义资源之一，目标规则（DestinationRule）定义了在路由发生后应用于服务的流量策略。这些规则指定负载均衡的配置、来自 Sidecar 代理的连接池大小以及异常检测设置，从而实现从负载均衡池中检测和驱逐不健康的主机。

Istio 网关（Gateway）

作为 Istio 自定义资源之一，Istio 网关（Gateway）定义了在网格出入口操作的负载均衡器，用于接收传入或传出的 HTTP/TCP 连接。它描述了需要公开的一组端口、要使用的协议类型、负载均衡器的 SNI 配置等信息。

服务条目（Service Entry）

作为 Istio 自定义资源之一，服务条目（ServiceEntry）是用于将一个服务添加到 Istio 抽象模型或服务注册表中，这些注册的服务是由 Istio 内部维护的。添加服务条目后，Envoy 代理可以将流量发送到该服务，如同这个添加的服务条目是网格中的其他服务一样。

入口网关服务（IngressGateway Service）

与 Istio 网关（Gateway）概念容易混淆的入口网关服务并不是指 Istio 自定义资源，而是指 Kubernetes 服务。它是真实的入口网关服务的抽象，后面由对应的容器来提供支持。创建一个入口网关服务时，会部署一个 Kubernetes 服务和 Deployment 资源到用户集群中。

2.4 产品获取

前提条件

在执行下述产品获取操作步骤前，请确保以下条件均已满足：

- 已成功获取并安装云环境，并且要求云环境版本 >= 6.1.1 。

操作步骤

1. 获取并安装“服务网格”云产品。

在顶部导航栏中，依次选择[产品与服务]-[产品与服务管理]-[云产品]，进入“云产品”页面获取并安装“服务网格”云产品。具体的操作说明，请参考“产品与服务管理”帮助中“云产品”的相关内容。

2.5 权限说明

本章节主要用于说明服务网格各功能的用户权限范围。其中，√ 代表该类用户可对云平台内所有项目的操作对象执行此功能，**XX项目** 代表该类用户仅支持对XX项目内的操作对象执行此功能，未标注代表该类用户无权限执行此功能。

功能		云管理员	部门管理员/项目管理员	普通用户
目标规则（Destination Rule）	信息展示	√	已加入项目	已加入项目
	修改删除	√	已加入项目	已加入项目
虚拟服务（Virtual Service）	信息展示	√	已加入项目	已加入项目
	修改删除	√	已加入项目	已加入项目
Istio 网关（Gateway）	信息展示	√	已加入项目	已加入项目
	修改删除	√	已加入项目	已加入项目
服务条目（Service Entry）	信息展示	√	已加入项目	已加入项目
	修改删除	√	已加入项目	已加入项目

2.6 使用限制

集群限制

- 启用应用服务网格前，您需要创建或已有一个可用集群，并确保集群版本 >= v1.20
- 默认不使用 istio ingress，因此不能管理南北流量。启用 istio ingress 需要开启环境中 loadbalance 控制器
- 重新部署云产品会导致之前正运行 sidecar 因证书问题无法连接，需要重建
- 不支持安全容器类型的负载添加至网格

功能约束

- 服务和工作负载（Deployment）必须是一一对应关系，不允许多个服务对应一个工作负载，因为可能出现灰度发布、网关访问等功能异常。
- 网格实例一旦创建后，不支持变更容器网络。
- 每个网格实例消耗 cpu 0.2 和 memory 256MB 配额。
- 出向网关：使用egressgateway时，因为eks节点默认没有配置外部dns解析，可能无法解析外网域名导致失败，设置步骤：修改coredns配置，添加upstream指向外部DNS，如：”forward . 114.114.114.114”，然后重启coredns(删除pod)，即可正常访问外部DNS进行域名解析。
- 当客户端位于集群内，且配置客户端不连接注册中心时，服务网格支持 dubbo2 协议主要在可观测性上，包括请求, 编解码成功失败指标。 并且提供基础路由方式(当前是基于 interface名称和method名称 路由)，缺乏 dubbo2 的原生流量治理，详见用户指南-多协议和第三方注册中心支持

2.7 与其他服务的关系

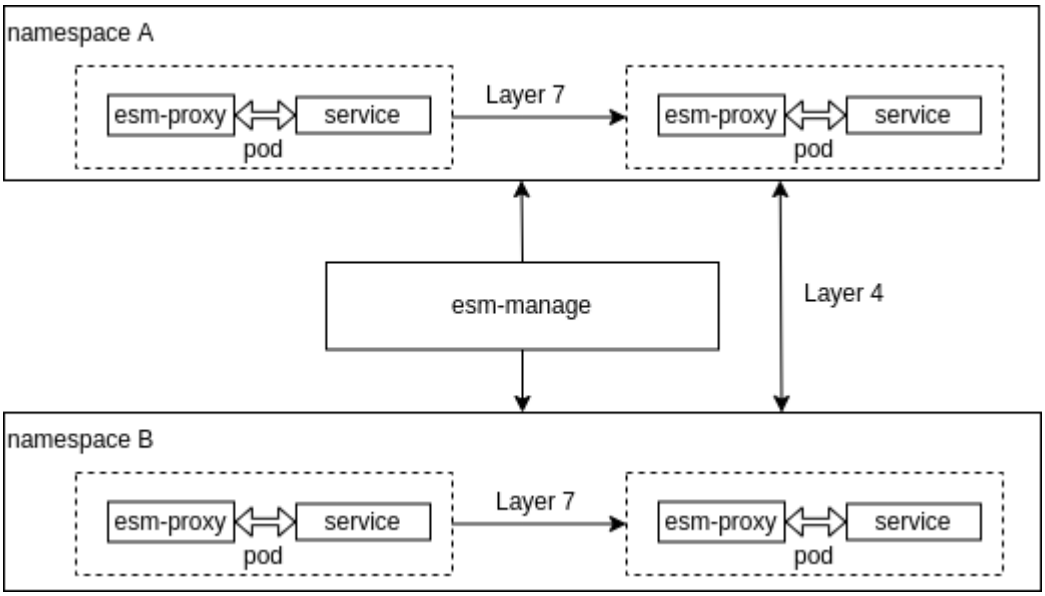
服务	关系说明
监控告警服务	平台服务，服务网格会将遥测数据存储在后端的监控告警服务中。

3 用户指南

3.1 产品组件说明

背景

当前服务网格是单控制面多数据面的架构，如下所示：



相同命名空间下的流量，可以识别到7层的流量信息，不同命名空间下的流量，当前需要手动配置 `sidecars.networking.istio.io` 资源才能识别7层流量信息，默认是4层流量信息。

问题

服务网格产品的控制器运行在kubernetes中，那么有哪些组件以及如何确定运行状态？

方法

进入kubernetes集群后台，或者使用kubeconfig文件，使用kubectl命令获取信息

```
# kubectl get po -n servicemesh
NAME                                          READY   STATUS
RESTARTS        AGE
cert-manager-cainjector-5fb6f8944-t9xjp       1/1     Running           0
41h9m
cert-manager-67d4c88cf9-n298m                 1/1     Running           0
41h9m
cert-manager-webhook-6cb88dd6b7-vrqfc         1/1     Running           0
41h9m
install-istio-all-crd-vw4m2                   0/1     Completed         0
41h9m
install-cert-g224n                            0/1     Completed         0
41h9m
gateway-api-admission-server-55bc5d7d76-sxp2j 1/1     Running           0
41h9m
kiali-6bd999cd9b-hjddm                        1/1     Running           0
41h9m
istiod-65dcf97b99-rkgfd                       1/1     Running           0
41h9m
grafana-7d5b57f4d4-gj6x7                      1/1     Running           0
41h9m
cert-manager-istio-csr-79f468f957-bsdw9       1/1     Running           0
41h9m
registryhub-64569c6f5d-c7jkl                  1/1     Running           0
41h9m
skywalking-ui-5c445d588c-t5trd                1/1     Running           0
41h9m
dubbo-controller-9846fbbc7-6zr7v              1/1     Running           0
41h9m
skywalking-oap-584b768f48-2mgkf               1/1     Running           1
41h9m
```

主要pod的作用描述如下:

- `cert-manager-*` :包括多个控制器，主要作用提供istio tls证书，包括客户端和CA等
- `install-*` : 主要是job有关pod，主要作用安装CRD，以及配置预装一些CR，如 envoyfilter，telemetry等用于istio监控相关配置
- `gateway-api-admission-server` :校验gw-api资源合法性的webhook
- `kiali` :istio管理的web ui开源工具，方便可视化

- `istiod` :核心istio控制器，包括pilot-discovery
- `registryhub, dubbo-controller` ： 多注册中心控制器
- `skywalking-*` :skywalking控制器，依赖elasticsearch服务，当配置错误 elasticsearch服务信息时，会crash

3.2 注入指导

本章节主要介绍使用服务网格

操作场景

业务容器如何注入sidecar ， 并开始使用服务网格

要求

1. pod 必须使用容器网络，不能是主机网络

部署

1. 命名空间下所有pod 使用服务网格，注意修改命名空间注解后，命名空间下的pod需要手动删除后才会触发注入 sidecar，才能使用服务网格

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    istio-injection: enabled    #添加该注解
  name: default
```

2. 某个pod使用服务网格，当不希望所有pod都注入，只在某些pod中注入时，以 deployment 控制器为例，修改如下

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld
  labels:
    app: helloworld
    version: v1
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: helloworld
      version: v1
  template:
    metadata:
      labels:
        app: helloworld
        version: v1
        sidecar.istio.io/inject: "true" # 添加该注解
    spec:
      containers:
      - name: helloworld
        image: nginx:1.25-alpine
        resources:
          requests:
            cpu: "100m"
```

3.3 灰度发布

本章节主要介绍在服务网格，如何实现灰度发布能力

操作场景

随着网站流量的增加，网站开始有了广告投放的需求，广告投放需要在商品页面增加广告位。网站的开发人员新开发了 product 服务的 v2版本，以 product v2的 deployment 的形式提供，并希望对 product-v2版本做灰度发布

部署

部署v1和v2版本的deployment以及对应的service至集群

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v2
  namespace: base
  labels:
    app: product
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v2
  template:
    metadata:
      labels:
        app: product
        version: v2
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
```

```
            - containerPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v1
  namespace: base
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v1
  template:
    metadata:
      labels:
        app: product
        version: v1
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: product
  name: product
  namespace: base
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: product
  type: ClusterIP
```

部署服务描述

通过 DR 定义服务版本 + 通过 VS 定义权重路由来完成灰度发布的第一步，将部分流量（50%）路由至 product v2 subset 以验证新版本，剩余部分（50%）的流量仍然路由至 product v1版本。将以下 YAML 文件提交至主集群即可完成以上设定

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: product-vs
  namespace: base
spec:
  hosts:
    - "product.base.svc.cluster.local"
  http:
    - match:
        - uri:
            exact: /
      route:
        - destination:
            host: product.base.svc.cluster.local
            subset: v1
            port:
              number: 80
          weight: 50
        - destination:
            host: product.base.svc.cluster.local
            subset: v2
            port:
              number: 80
          weight: 50
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: product
  namespace: base
spec:
  host: product
  subsets:
```

```
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
```

配置完成后，访问 product 服务的流量将有 50%被路由至 v1 版本，50%被路由至 v2 版本，刷新网站商品页面即可验证。

3.4 会话保持

本章节主要介绍在服务网格，如何实现会话保持能力

操作场景

购物车服务由多个 Pod 副本运行，需要会话保持功能，以保证同一用户请求被路由至同一个 Pod，保证同一用户的购物车信息不会丢失

部署

通过设置 card 服务 DestinationRule 的负载均衡策略实现，以请求中 header 中的 UserID 做一致性 hash 负载均衡，调用 card 服务验证会话保持功能，同一用户的多次请求会被路由至同一个 Pod

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: card
  namespace: base
spec:
  host: cart
  trafficPolicy:
    loadBalancer:
      consistentHash:
        httpHeaderName: UserID
  exportTo:
    - '*'
```

3.5 故障注入

本章节主要介绍在服务网格，如何实现故障注入能力

操作场景

业务团队需要模拟访问后端服务存在延迟或者错误故障时网站系统的行为，以测试服务弹性，网站用户的优化访问体验

部署

通过配置绑定stock服务的VritualService，设置访问stock服务的故障注入策略：100%的请求会有7秒的固定延迟，并且返回503错误。

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: stock
  namespace: base
spec:
  hosts:
    - stock.base.svc.cluster.local
  http:
    - route:
        - destination:
            host: stock.base.svc.cluster.local
      fault:
        delay:
          fixedDelay: 7s
          percentage:
            value: 100
        abort:
          httpStatus: 503
          percentage:
            value: 100
```

3.6 超时

本章节介绍服务网格如何通过VirtualService配置，实现http超时功能。

操作场景

当网站发生故障，导致网站用户的请求一直处于等待状态时，为优化网站用户的浏览体验，需要为服务配置timeout。

部署服务

部署 v1 和 v2 版本的 deployment以及对应的service 至集群

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v2
  namespace: base
  labels:
    app: product
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v2
  template:
    metadata:
      labels:
        app: product
        version: v2
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v1
  namespace: base
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v1
  template:
    metadata:
      labels:
        app: product
        version: v1
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: product
  name: product
  namespace: base
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: product
  type: ClusterIP
```

应用服务网格配置

通过DestinationRule定义服务的subset v1和v2，在VirtualService配置中关联service（product），并添加timeout字段，可以实现超时自动断开，并返回504错误（Gateway Timeout），从而优化网站用户体验。将以下VirtualService部署至集群：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: product-vs
  namespace: base
spec:
  hosts:
    - "product.base.svc.cluster.local"
  http:
    - match:
        - headers:
            cookie:
              exact: vip=false
      route:
        - destination:
            host: product.base.svc.cluster.local
            subset: v1
      timeout: 100ms
    - match:
        - headers:
            cookie:
              exact: vip=true
      route:
        - destination:
            host: product.base.svc.cluster.local
            subset: v2
      timeout: 300ms
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: product
  namespace: base
spec:
```

```
  host: product
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
```

完成配置之后，当请求来临时，cookie中如包含vip=false，其超时时间为100ms；cookie中如包含vip=true，其超时时间为300ms。

3.7 重试

本章介绍服务网格中，如何配置错误重试。

操作场景

当网站报错时，某些场景下重试能够避免这类错误，此时通过配置重试策略，能够有效降低这类错误，提升用户访问网站的体验。

部署服务

部署 v1 和 v2 版本的 deployment以及对应的service 至集群

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v2
  namespace: base
  labels:
    app: product
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v2
  template:
    metadata:
      labels:
        app: product
        version: v2
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v1
  namespace: base
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v1
  template:
    metadata:
      labels:
        app: product
        version: v1
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: product
  name: product
  namespace: base
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: product
  type: ClusterIP
```

应用服务网格配置

通过VirtualService对服务的重试策略进行配置，控制服务的重试行为。将以下VirtualService的yaml文件部署到集群：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: product-vs
  namespace: base
spec:
  gateways:
  - mesh
  hosts:
  - product
  http:
  - route:
    - destination:
        host: product
        port:
          number: 80
    retries:
      attempts: 3
      retryOn: 5xx
```

部署完成后，当服务返回5xx（服务端错误）错误时，网格会自动为请求进行重试，重试次数上限的3次。

3.8 流量镜像

本章介绍如何在服务网格中，实现流量镜像功能。

操作场景

在预发布过程中，为了尽可能使新版本服务的流量与线上保持一致，且不对线上环境产生影响，可以将线上流量镜像到预发布版本。同时通过流量镜像功能也可以对线上流量进行采集分析。

部署服务

部署 v1 和 v2 版本的 deployment以及对应的service 至集群

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v2
  namespace: base
  labels:
    app: product
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v2
  template:
    metadata:
      labels:
        app: product
        version: v2
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v1
  namespace: base
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v1
  template:
    metadata:
      labels:
        app: product
        version: v1
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: product
  name: product
  namespace: base
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: product
  type: ClusterIP
```

应用服务网格配置

假设v1为线上版本，v2为预发布版本，通过DestinationRule定义subset v1和v2，同时在VirtualService中对流量镜像的策略进行定义，将线上流量复制到预发布版本。将以下yaml配置部署至集群：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: product-vs
  namespace: base
spec:
  gateways:
  - mesh
  hosts:
  - product
  http:
  - route:
    - destination:
        host: product
        port:
          number: 80
        subset: v1
    mirror:
      host: product
      subset: v2
      port:
        number: 80
    mirrorPercentage:    # 可选，默认100.0
      value: 50.0
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: product
  namespace: base
spec:
  host: product
  subsets:
    - name: v1
      labels:
```

```
      version: v1
  - name: v2
    labels:
      version: v2
```

上述配置实现了将v1流量的50%镜像到v2。

3.9 边缘网关

本章介绍在服务网格中，如何配置入向和出向的边缘网关功能。

操作场景

在实际使用中，服务经常会有南北向流量的需求，如服务需要对外暴露，接收外部请求（入向流量），并且需要向外部服务发起请求（出向流量）。

部署服务

部署v1和v2版本的deployment以及对应的service至集群

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v2
  namespace: base
  labels:
    app: product
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v2
  template:
    metadata:
      labels:
        app: product
        version: v2
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v1
  namespace: base
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v1
  template:
    metadata:
      labels:
        app: product
        version: v1
    spec:
      containers:
        - name: product
          image: nginx:1.25-alpine
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: product
  name: product
  namespace: base
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: product
  type: ClusterIP
```

绑定ingress（入向）

前提：istio-ingressgateway服务具有外部IP，能够被集群外访问。 通过Gateway定义服务对外暴露的域名：

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: product-gateway
  namespace: base
spec:
  selector:
    istio: istio-ingressgateway
  servers:
  - hosts:
    - www.product.com
    port:
      name: http
      number: 80
      protocol: HTTP
```

如上所示，对外暴露的域名为 `www.product.com` 。 使用VirtualService绑定Gateway，将流量导到product服务：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: product-vs
  namespace: base
spec:
  gateways:
  - product-gateway
  - mesh
  hosts:
  - www.product.com
  - product
  http:
  - route:
    - destination:
        host: product
```

```
        port:
          number: 80
```

上述配置中通过 `gateways` 字段指定 `product` 服务同时接收来自网关（ `product-gateway` ）和网格内部（ `mesh` ）的http请求，外部域名为 `www.product.com` ，内部域名为 `product` （k8s服务名）。

将配置部署到集群，即可实现product服务对集群外暴露。

绑定egress（出向）

前提：EKS的coredns中配置了外部dns，能够解析外部域名。

product服务需要访问集群外网站，假设为 `www.baidu.com` ，为了统一出口，外部访问均通过egress统一处理，可以通过Gateway将 `www.baidu.com` 绑定到istio-egressgateway：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: product-egressgateway
  namespace: base
spec:
  selector:
    istio: istio-egressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - www.baidu.com
```

使用VirtualService定义访问路由：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: product-vs-out
  namespace: base
spec:
```

```
  hosts:
  - www.baidu.com
  gateways:
  - product-egressgateway
  - mesh
  http:
  - match:
    - gateways:
      - mesh
      port: 80
    route:
    - destination:
        host: istio-egressgateway.servicemesh.svc.cluster.local
        port:
          number: 80
  - match:
    - gateways:
      - product-egressgateway
      port: 80
    route:
    - destination:
        host: www.baidu.com
        port:
          number: 80
```

配置中将网格内（ `mesh` ）访问 `www.baidu.com` 的流量路由到了istio-egressgateway，流量到达istio-egressgateway时匹配 `product-egressgateway` 规则，通过coredns对域名进行解析，将流量路由到真实的外部后端。

注意：当前版本的meshConfig中没有开启REGISTRY_ONLY选项，故不需要额外配置ServiceEntry，后续版本如有变化，须同步新增ServiceEntry定义。

3.10 连接池限制

本章节主要介绍在服务网格，如何实现连接池限制能力

操作场景

随着网站业务规模的增大，对网站的访问请求并发量开始增加，网站业务人员计划限制服务最大并发数，保证服务运行健壮性

部署

为模拟"高并发"请求场景，首先通过提交以下 YAML 部署 client 服务，模拟对 user 服务的高并发请求

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: client
  namespace: base
spec:
  restartPolicy: Never
  containers:
  - image: docker.yylt.ml/yylt/tool:202307061000
    imagePullPolicy: IfNotPresent
    name: wrk
    command:
    - wrk
    - -c
    - "5"
    - -t
    - "5"
    - http://server.base.svc
```

此时对于访问 user 服务没有最大并发数限制，所有请求均可访问成功。通过控制台 client deployment 查看 client pod 日志可以发现均成功

限流

配置 user 服务的 Destination Rule 限制最大并发数为1

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: user
  namespace: base
spec:
  host: user
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 1
        http2MaxRequests: 1
        maxRequestsPerConnection: 1
  exportTo:
    - '*'
```

此时通过控制台client deployment查看client pod日志可以发现只有一个成功。

3.11 多协议和第三方注册中心支持

本章节主要介绍在服务网格，如何实现多协议和第三方注册中心的支持。

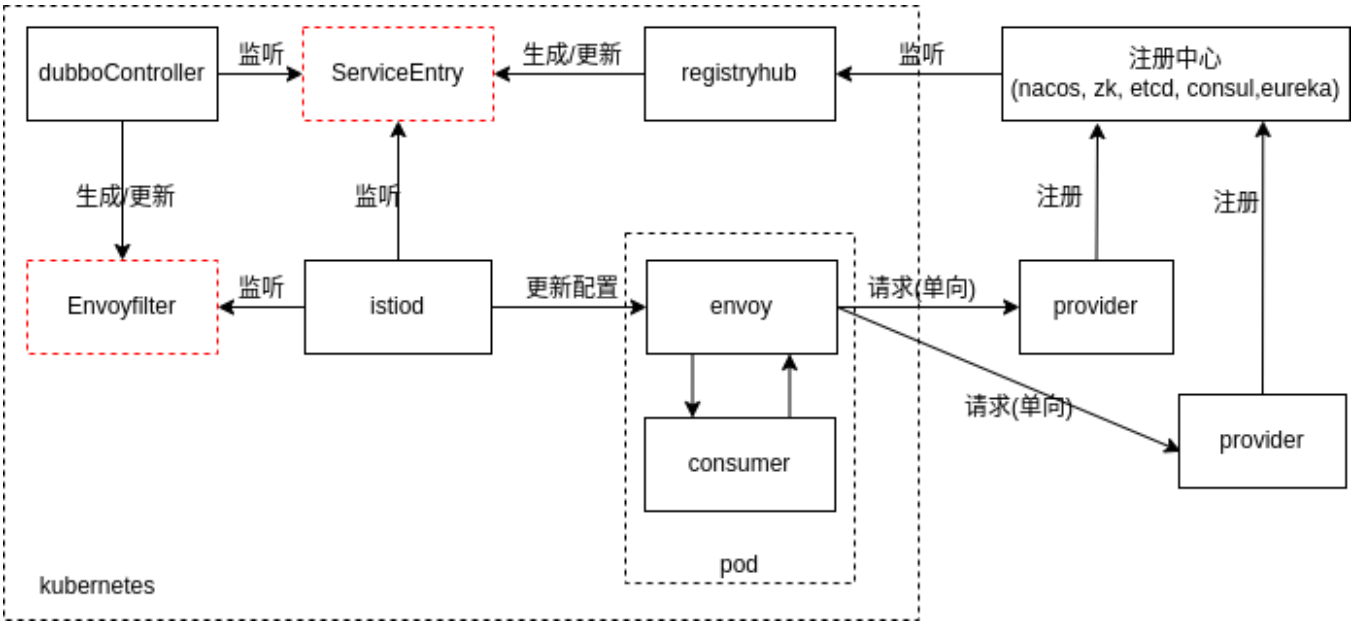
操作场景

多协议支持是由envoy自身特点支持，当前支持的协议包括不限于dubbo，mysql，redis，kafka，postgres和mongo。通过协议识别，envoy会记录请求关键信息和请求返回码，并作为可观测性指标等信息暴露服务状态，这是多协议数据面支持的基础。

这里通过基于aeraki-mesh改造的dubbo-controller和registry-hub服务，来支持以下两种场景：

- dubbo 2协议应用场景：支持zookeeper和nacos注册中心
- springboot应用场景：支持eureka和consul注册中心

具体架构场景如下：



1. consumer客户端位于集群内，且不会有集群外主动访问集群内流量
2. provider服务端位于集群外

dubbo 2协议应用场景

dubbo协议场景使用限制

- 当前对dubbo的治理主要 简单路由(interface) 和指标层面，不是很完善，包括重试，负载均衡，缓存等，如需要这些能力，建议用dubbo的流量治理能力或者配置consumer客户端
- 当使用服务网格的服务发现时，要求 consumer 客户端不连接注册中心，交给服务网格的服务发现，因为服务网格只提供服务发现能力，不能向 consumer客户端动态下发治理相关配置(如重试，负载均衡，缓存等)，因此需在启动配置中填写相关配置，比如在dubbo-consumer.properties配置以下项(参考dubbo服务消费者配置)，修改配置需要重启consumer客户端。而 provider 服务端则不受影响，仍然支持动态配置（参考dubbo流量管控动态配置）。

```
dubbo.consumer.timeout
dubbo.consumer.retries
dubbo.consumer.loadbalance
dubbo.consumer.async
dubbo.consumer.connections
dubbo.consumer.cache
等...
```

- 如果需要完整的服务网格的治理功能，可以使用基于grpc的dubbo 3协议，不用注册中心，直接用kubernetes service访问provider端。

zookeeper注册中心

```yaml
apiVersion: ecns.easystack.cn/v1alpha1
kind: RegistryHub
metadata:
  name: registryhub-sample
  namespace: xxx
spec:
  # 生成的 serviceentry 导出到哪个空间
  generate_to: dubbozk
  # 注册中心类型
  type: zookeeper
  # 注册中心地址，需加命名空间
  address: "zookeeper.dubbozk:2181"
```

如果 `dubbozk` 命名空间中的 `zookeeper` 有服务注册信息，则会在 `generate_to` 指定的空间 `dubbozk` 下生成如下的serviceentry：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  annotations:
    interface: org.apache.dubbo.samples.api.GreetingService
    workloadSelector: ""
  creationTimestamp: "2023-08-22T03:02:18Z"
  generation: 3
  labels:
    manager: aeraki
    registry: dubbo2istio
  name: aeraki-org-apache-dubbo-samples-api-greetingservice-1-0-0-dubbozk
  namespace: dubbozk
  resourceVersion: "4902168"
  uid: cb1b9086-fccf-47aa-a593-7ed3c5aacdfe
spec:
  addresses:
  - 240.240.0.1
  endpoints:
  - address: 10.42.0.57
    labels:
      anyhost: "true"
      application: zookeeper-demo-provider
      background: "false"
      deprecated: "false"
      dubbo: 2.0.2
      dynamic: "true"
      generic: "false"
      interface: org.apache.dubbo.samples.api.GreetingService
      methods: sayHello
      pid: "7"
      registryName: registryhub-sample
      release: 1.0-SNAPSHOT
      revision: 1.0-SNAPSHOT
      service-name-mapping: "true"
      side: provider
      timestamp: "1692846277949"
      version: 1.0.0
    ports:
      tcp-metaprotocol-dubbo: 20880
    serviceAccount: default
```

```
  hosts:
  - org.apache.dubbo.samples.api.greetingservice-1.0.0.dubbozk
  location: MESH_INTERNAL
  ports:
  - name: tcp-metaprotocol-dubbo
    number: 20880
    protocol: tcp
    targetPort: 20880
  resolution: STATIC
```

同时会在servicemesh空间下生成envoyfilter：

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: inbound-org.apache.dubbo.samples.api.greetingservice-240.240.0.1-
20880
spec:
  configPatches:
  - applyTo: NETWORK_FILTER
    match:
      listener:
        name: 240.240.0.1_20880
        filterChain:
          filter:
            name: "envoy.filters.network.tcp_proxy"
    patch:
      operation: REPLACE
      value:
        name: envoy.filters.network.dubbo_proxy
        typed_config:
          "@type":
type.googleapis.com/envoy.extensions.filters.network.dubbo_proxy.v3.DubboPro
xy
          protocol_type: Dubbo
          serialization_type: Hessian2
          statPrefix:
outbound|20880||org.apache.dubbo.samples.api.greetingservice
          route_config:
          - name:
outbound|20880||org.apache.dubbo.samples.api.greetingservice
```

```
            interface: org.apache.dubbo.samples.api.greetingservice
            routes:
            - match:
                method:
                  name:
                    exact: sayHello
              route:
                cluster:
outbound|20880||org.apache.dubbo.samples.api.greetingservice
```

这样dubbo协议的应用consumer就可以通过生成的 `ServiceEntry` 的hosts `org.apache.dubbo.samples.api.greetingservice-1.0.0.dubbozk` 来访问provider服务了。host的生成规则为： `${interface}-${version}.${generate_to}` ，其中：

- interface： provider服务的接口名
- version：provider服务的版本号
- generate_to： `RegistryHub cr` 中的generate_to字段值

nacos注册中心

跟zookeeper类似：

```
kind: RegistryHub
metadata:
  name: registryhub-sample
spec:
  # serviceentry 导出到哪个空间
  generate_to: default
  # 注册中心类型
  type: nacos
  # 注册中心地址，需加命名空间
  address: "nacos.dubbonacos:8848"
```

springboot场景应用

springboot直接使用的http协议，因此只需要将注册中心数据生成serviceentry即可。

eureka注册中心

```yaml
apiVersion: ecns.easystack.cn/v1alpha1
kind: RegistryHub
metadata:
  name: registryhub-sample
spec:
  generate_to: springnacos
  type: eureka
  address: "http://eureka.springnacos:8761/eureka"
```

生成的serviceentry为：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  annotations:
    manager: encs
    registry: eureka
  creationTimestamp: "2023-08-24T04:35:56Z"
  generation: 1
  labels:
    registry-name: registryhub-sample
  name: eureka-provider.springnacos
  namespace: springnacos
  resourceVersion: "4907683"
  uid: 544a90ac-9f60-493f-9d84-d3658a3aa37c
spec:
  endpoints:
  - address: 10.42.0.59
    labels:
      eurekaname: EUREKA-PROVIDER
  hosts:
  - eureka-provider.springnacos
  location: MESH_INTERNAL
  ports:
  - name: "9000"
    number: 9000
    protocol: tcp
    targetPort: 9000
  resolution: STATIC
```

这样springboot应用consumer就可以通过生成的 `ServiceEntry` 的hosts `eureka-provider.springnacos` 来访问provider服务了。host的生成规则为： `${app_name}.${generate_to}` ，其中：

- app_name： provider服务名
- generate_to： `RegistryHub cr` 中的generate_to字段值

consul注册中心

跟eureka类似：

```
apiVersion: ecns.easystack.cn/v1alpha1
kind: RegistryHub
metadata:
  name: registryhub-sample
spec:
  generate_to: springconsul
  type: consul
  address: "http://consul.springconsul:8500"
```

3.12 授权

本章节主要介绍在服务网格，如何实现访问授权控制能力

操作场景

生产环境（product namespace）的服务已经稳定运行，电商网站业务团队希望对网格中的服务做权限控制，限制生产环境（product namespace）下的服务不能被测试环境（test namesapce）下的服务访问。

部署

对网格中的服务进行权限控制可通过配置 AuthorizationPolicy 实现，配置以下 AuthorizationPolicy 策略限制 product namespace 下所有服务不能被 test namespace 下的服务访问

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: base-authz
  namespace: product
spec:
  action: DENY
  rules:
  - from:
    - source:
        namespaces:
        - test
```

3.13 认证

本章节主要介绍在服务网格，如何实现连接认证能力

操作场景

希望限制访问生产环境（base namespace）下所有服务间的访问必须开启双向认证 mTLS，以防御中间人攻击。

服务网格中服务间通信模式默认为 PERMISSIVE 宽容模式，即服务间的通信既可以使用 mTLS 加密，也可以使用 plaintext 明文连接。

此时在容器服务控制台登录 client 容器，使用明文连接对生产环境（base namespace）product 服务发起请求：curl `http://product.base.svc.cluster.local/` ，此时明文连接也可正常访问 product 服务

部署

限制 base namespace 下的服务间通信必须采用 mTLS 模式可以通过 PeerAuthentication 策略设置 mTLS 模式为 STRICT 完成

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: base-strict
  namespace: base
spec:
  mtls:
    mode: STRICT
```

4 常见问题

4.1 为什么没注入

描述

为什么有些 pod 没有注入 sidecar?

检查方法

1. 首先检查pod所在命名空间，是否有注解： `istio-injection: enabled` ，如没有注解，则不会主动注入pod
2. 检查pod是否有注解： `sidecar.istio.io/inject: "true"` ，如有该注解，即使命名空间不设置注解也会主动注入
3. 检查pod类型，当前job，cronjob类型控制器产生的pod不会注入，即使配置1/2 ，也不会注入 sidecar
4. 检查pod网络类型，主机网络类型的不会注入，即使配置1/2中的注解也不会注入sidecar
5. 检查pod的注解，当有注解： `sidecar.istio.io/inject: "false"` ， 即使配置1 ，也不会主动注入sidecar

4.2 Skywalking工作负载Crash

描述

使用Skywalking实现链路追踪需要自建ElasticSearch，在Skywaking工作负载中填入正确的ElasticSearch信息。如果ElasticSearch连接异常会导致Skywalking工作负载Crash。

解决方案

步骤一

修改工作负载的yaml文件 `kubectl edit deploy skywalking-oap -n servicemesh` 修改 `skywalking-oap` 容器的环境变量:

```
        containers:
        - name: skywalking-oap
          image: {{ tuple .Values.images.tags "skywalking_oap_server" . |
include "helm-toolkit.utils.update_image" }}
          env:
            - name: SW_STORAGE_ES_CLUSTER_NODES
              value: "elasticsearch.servicemesh:9200"
            - name: SW_STORAGE_ES_HTTP_PROTOCOL
              value: "HTTP"
            - name: SW_ES_USER
              value: ""
            - name: SW_ES_PASSWORD
              value: ""
```

变量含义:

- `SW_STORAGE_ES_CLUSTER_NODES` :ElasticSearch服务地址，默认为 `elasticsearch.servicemesh:9200`
- `SW_STORAGE_ES_HTTP_PROTOCOL` :ElasticSearch连接协议，默认为HTTP
- `SW_ES_USER` :ElasticSearch用户，默认为空
- `SW_ES_PASSWORD` :ElasticSearch密码，默认为空 确保ElasticSearch相关信息填写正确且网络可达。

步骤二

保存退出后工作负载滚动升级，等待并观察Skywalking工作负载是否处于Running状态且Ready字段为1/1

kubectl get po -n servicemesh|grep skywalking-oap

```
skywalking-oap-xxxxxxxxxx-xxxxx                1/1     Running   0
1h
```

4.3 访问跨命名空间的服务没有度量和链路信息

描述

在某个命名空间a的客户端应用，访问另一个命名空间b的服务端应用后，通过kiali等页面查看时，发现没有度量信息和链路信息。

解决办法

为了减少服务网格下发到envoy sidecar的服务配置数量，在istio层面默认做了命名空间隔离，只能看到当前命名空间和指定的服务kubernetes和tracing-oap，可通过以下命令查看：

```
kubectl -nservicemesh get sidecar default -oyaml
apiVersion: networking.istio.io/v1beta1
kind: Sidecar
metadata:
  name: default
  namespace: servicemesh
spec:
  egress:
  - hosts:
    - ./*
    - default/kubernetes.default.svc.cluster.local
    - servicemesh/tracing-oap.servicemesh.svc.cluster.local
```

此时如果不在a命名空间设置sidecar资源，a空间下的客户端服务是没有经过sidecar的，因此没有度量和链路信息。需要在a命名空间下设置sidecar资源，让a空间下的sidecar能够访问b空间的服务端应用，示例设置如下，a空间默认可以访问b空间的服务b：

```
apiVersion: networking.istio.io/v1beta1
kind: Sidecar
metadata:
  name: default
  namespace: a
spec:
  egress:
```

```
  - hosts:
    - "b/svcb"
```

4.4 添加第三方注册中心后没生成服务对应的ServiceEntry资源

描述

部署RegistryHub资源后，返回 `registryhub.ecns.easystack.cn/xxx created` ，但是在导出的命名空间一直没有服务端对应的ServiceEntry资源。

解决方案

1. 通过查看RegistryHub资源描述，如果出现错误，在 `status.phase` 状态会置为 `failed` ，同时在 `status.conditions` 中会标注出错误原因，比如下面的错误是zookeeper地址设置错误，通过修改为正确地址后，再重新部署一下资源即可：

```
kubectl get registryhub registryhub-sample -oyaml
apiVersion: ecns.easystack.cn/v1alpha1
kind: RegistryHub
metadata:
  name: registryhub-sample
  namespace: default
spec:
  address: zookeeper1.default:2181
  generate_to: default
  type: zookeeper
status:
  address: zookeeper1.default:2181
  conditions:
  - failed_reason: 'lookup zookeeper1.default on 10.43.0.10:53: no such
host'
    record_time: "2024-02-04T02:53:25Z"
  generate_to: default
  phase: failed
  serviceentries: {}
  type: zookeeper
```

2. 查看RegistryHub资源描述，`status.phase` 显示为 `success` ，但仍然没有生成服务端对应的ServiceEntry资源，`status.serviceentryies` 只有几条 `ServiceEntry` 记录：

```
status:
  address: zookeeper.default:2181
  generate_to: default
  phase: success
  serviceentries:
    aeraki-org-apache-dubbo-samples-api-greetingservice-1-0-0-default:
"2024-02-04T03:04:20Z"
  type: zookeeper
```

可从以下两个方面排查：

- 是不是刚部署registryhub资源：如果是，可以继续等待几分钟，同时关注 `status.serviceentryies` 列表是否在持续增加，如果在增加，说明还在同步，稍后客户端再重试即可。
- 注册中心是否有服务端信息：如果registryhub的 `status.serviceentryies` 没有增加了，也没有看到服务端对于的ServiceEntry资源，可以排查注册中心是否存在服务端信息。

4.5 Envoy默认会将Header转换为小写

问题描述

Envoy缺省会把http header的key转换为小写，例如有一个http `header Test-Upper-Case-Header: some-value` ，经过envoy代理后会变成 `test-upper-case-header: some-value` 。这个在正常情况下没问题，RFC 2616 规范也说明了处理HTTP Header是大小写不敏感的

通常 header转换为小写不存在规范问题，但有些情况下对header大小写敏感会存在以下问题，例如：

- 业务解析header依赖大小写。
- 使用的SDK对Header大小写敏感，如读取 `Content-Length` 来判断response长度时依赖首字母大写。

解决方案

这种情况强制指定为TCP协议。将服务声明为TCP协议，不让istio进行七层处理，也就不会更改http header大小写了，但需要注意的是同时也会丧失istio的七层能力

当是集群内服务时，可以配置service使用TCP协议

```
kind: Service
metadata:
  name: myservice
spec:
  ports:
  - number: 80
    name: tcp-web # 指定该端口协议为 tcp
```

当是集群外服务时，可以配置serviceEntry数据结构

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: cos
spec:
  hosts:
  - "private.cos.guangzhou.mycloud.com"
```

```
  location: MESH_INTERNAL
  addresses:
  - 169.254.0.47
  ports:
  - number: 80
    name: tcp
    protocol: TCP
  resolution: DNS
```

4.6 第三方注册中心Zookeeper、Nacos生成ServiceEntry资源后，仍然不能访问

描述

使用 `RegistryHub` 资源，注册了基于 `dubbo` 协议的第三方注册中心Zookeeper、Nacos，也生成 `ServiceEntry` 资源，但是客户端仍然无法访问，比如生成的 `ServiceEntry` 如下：

```
kubectl -ndefault get serviceentry
NAME                                                                    HOSTS
LOCATION        RESOLUTION   AGE
aeraki-org-apache-dubbo-samples-api-greetingservice-1-0-0-default
["org.apache.dubbo.samples.api.greetingservice-1.0.0.default"]
MESH_INTERNAL   STATIC       32s
```

解决办法

使用基于 `dubbo` 协议的第三方注册中心Zookeeper、Nacos，因istio官方不知道 `dubbo` 协议，我们服务网格产品还会在 `servicemesh` 命名空间生成 `envoyfilter` ，来将下方的协议替换为 `envoy` 支持的 `dubbo` 协议，可通过以下命令查看是否生成对于的 `envoyfilter` ：

```
kubectl -nservicemesh get envoyfilter
```

对于上面的 `serviceentry` `aeraki-org-apache-dubbo-samples-api-greetingservice-1-0-0-default` 会生成以下两条记录：

```
aeraki-outbound-org.apache.dubbo.samples.api.greetingservice-1.0.0.default-
240.240.0.1-20880   3m53s
aeraki-inbound-org.apache.dubbo.samples.api.greetingservice-1.0.0.default-
20880               3m53s
```

如果查看没有生成对于的 `envoyfilter` ，可重启服务网格中的 `dubbo-controller` 服务，稍后查看是否解决。

```
kubectl -nservicemesh rollout restart deployment/dubbo-controller
```

4.7 Headless service相关问题

问题描述

服务间通过注册中心调用响应404，在Kubernetes的服务发现中，会使用service域名方式注册，如服务间调用不经过域名解析，直接向获取到的目的IP发起调用会导致404问题

解决方案

注册中心不直接注册Pod IP地址，注册service域名。 客户端请求时带上hosts（需要更新代码）。

4.8 Virtual Service 路由匹配顺序问题

问题描述

在写 VirtualService 路由规则时，通常会 match 各种不同路径转发到不同的后端服务，有时候不小心命名冲突了，导致始终只匹配到前面的服务，如下例子:

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: test
spec:
  gateways:
  - default/example-gw
  hosts:
  - 'test.example.com'
  http:
  - match:
    - uri:
        prefix: /usrv
    rewrite:
      uri: /
    route:
    - destination:
        host: usrv.default.svc.cluster.local
        port:
          number: 80
  - match:
    - uri:
        prefix: /usrv-expand
    rewrite:
      uri: /
    route:
    - destination:
        host: usrv-expand.default.svc.cluster.local
        port:
          number: 80
```

istio匹配是按顺序匹配，不像nginx那样使用最长前缀匹配。这里使用prefix进行匹配，第一个是 `/usrv` ，表示只要访问路径前缀含 `/usrv` 就会转发到第一个服务，由于第二个匹配路径 `/usrv-expand` 本身也属于带 `/usrv` 的前缀，所以永远不会转发到第二个匹配路径的服务

解决方案

这种情况可以调整下匹配顺序，如果前缀有包含的冲突关系，越长的放在越前面

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: test
spec:
  gateways:
  - default/example-gw
  hosts:
  - 'test.example.com'
  http:
  - match:
    - uri:
        prefix: /usrv-expand
    rewrite:
      uri: /
    route:
    - destination:
        host: usrv-expand.default.svc.cluster.local
        port:
          number: 80
  - match:
    - uri:
        prefix: /usrv
    rewrite:
      uri: /
    route:
    - destination:
        host: usrv.default.svc.cluster.local
        port:
          number: 80
```

4.9 重试策略导致服务异常

问题描述

Istio为Envoy设置了缺省的重试策略，会在connect-failure,refused-stream, unavailable, cancelled, retriable-status-codes等情况下缺省重试两次。出现错误时，可能已经触发了服务器逻辑，在操作不是幂等（任意多次执行所产生的影响均与一次执行的影响相同）的情况下，可能会导致错误

解决方案

可以通过配置 VS 关闭重试

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - retries:
      attempts: 0
```

4.10 如何调整istio-proxy容器resources requests取值

描述

istio-proxy容器资源占用大小的默认配置如下。如果不符合要求，可按照实际需求进行修改。

```
resources:
  requests:
    cpu: 500m
    memory: 128Mi
  limits:
    cpu: 2000m
    memory: 1024Mi
```

解决方案

调整网格中的某个服务

步骤一：

修改服务的yaml文件。 `kubectl edit deploy <nginx> -n <namespace>`

步骤二：

在 `spec.template.metadata.annotations` 下添加如下配置（大小仅供参考，请自行替换）。

```
sidecar.istio.io/proxyCPU: 500m
sidecar.istio.io/proxyCPULimit: 500m
sidecar.istio.io/proxyMemoryLimit: 1024Mi
sidecar.istio.io/proxyMemory: 1024Mi
```

步骤三

修改后服务滚动升级，确保不会断服。

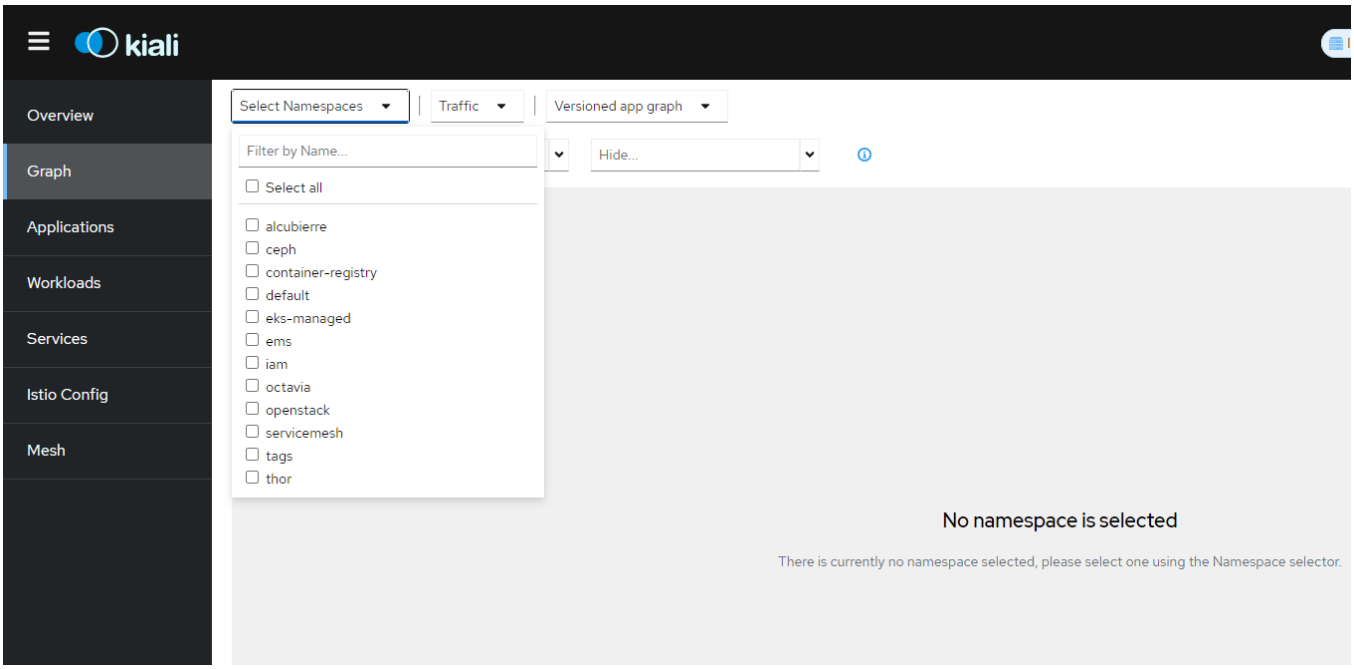
4.11 Kiali流量监控拓扑图中找不到服务

描述

Kiali页面流量拓扑图未展示用户部署的服务。

解决方案

步骤一：确认命名空间是否选择正确



确保选择服务部署对应的命名空间。

步骤二：确认服务是否有网络流量

确认服务有发起或者接收到请求，可结合pod日志进行查看：`kubectl logs <服务对应pod名称> -n <服务所在命名空间>`

步骤三：确认命名空间sidecar注入标签

```
kubectl get namespace <服务所在命名空间> -oyaml
```

```
apiVersion: v1
kind: Namespace
metadata:
  ...
  labels:
    kubernetes.io/metadata.name: <服务所在命名空间>
    istio-injection: enabled
  name: <服务所在命名空间>
...
```

如果命名空间不存在标签 `istio-injection=enabled` ，执行以下命令给命名空间打上标签。 `kubectl label namespace <服务所在命名空间> istio-injection=enabled` 或者如果只希望单独为服务添加注入标签开启sidecar注入，请参考步骤四中的场景2。

步骤四：确认sidecar注入相关的工作负载标签和注解

场景一：所在命名空间已有sidecar注入标签

确认pod模板没有禁止sidecar注入的标签或者注解（注解方式不推荐使用） `kubectl get deploy <服务对应的deployment名称> -n <服务所在命名空间>`

```
...
spec:
  ...
  template:
    metadata:
      ...
      labels:
        sidecar.istio.io/inject: false
        ...
      annotations:
        sidecar.istio.io/inject: false #注解方式不推荐使用，与标签二选一配置即可
    spec:
      ...
...
```

如标签或者注解为 `sidecar.istio.io/inject: false` ，编辑 `deployment` ，删除 `pod` 模板中的 `sidecar.istio.io/inject` 标签或者注解即可。 `kubectl edit deploy <服务对应的deployment名称> -n <`

服务所在命名空间>

场景二：所在命名空间没有sidecar注入标签，希望单独为服务添加注入标签

`kubectl edit deploy <服务对应的deployment名称> -n <服务所在命名空间>` 编辑 `deployment` 中的 `pod` 模板，为 `pod` 添加注入标签

```
...
spec:
  ...
  template:
    metadata:
      ...
      labels:
        sidecar.istio.io/inject: true #推荐用法
        ...
      annotations:
        sidecar.istio.io/inject: true #注解方式不推荐使用，与标签二选一配置即可
    spec:
      ...
...
```

保存退出，等待服务滚动升级完成。

步骤五：标签配置正确的情况，检查pod中是否有sidecar

如果前面步骤检查都没问题，但Kiali页面仍未展示相关的流量拓扑，需要检查sidecar实际是否注入。

`kubectl get pod <服务对应的pod> -o jsonpath={.spec.containers[*].name} -n <服务所在命名空间>`

```
<服务对应的容器> ... istio-proxy
```

输出的容器名称有 `istio-proxy` 即已经注入了 `sidecar` 。 如果 `istio-proxy` 容器不存在，可以选择对相应的 `deployment` 进行滚动升级， `sidecar` 会自动注入。 `kubectl edit deploy <服务对应的deployment名称> -n <服务所在命名空间>` 编辑服务对应的 `deployment` ，推荐通过添加自定义 `pod` 模板标签的方式进行滚动升级。

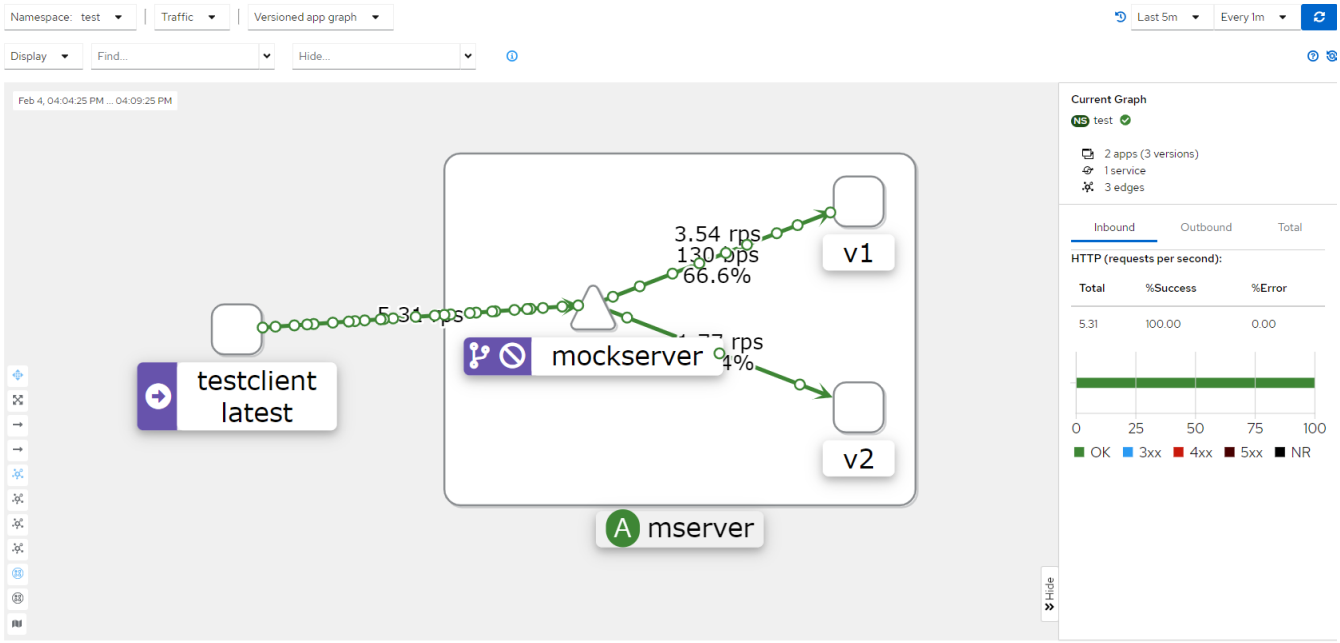
```
...
spec:
```

```
  ...
  template:
    metadata:
      ...
      labels:
        try-inject: '12345678'    #无特殊要求，可自定义
        ...
    spec:
      ...
...
```

等待滚动升级完成。

步骤六：完成所有检查后，打开Kiali页面确认流量拓扑正确展示

需要注意选择查询的时间长度以及刷新周期。也可点击刷新按钮手动刷新流量拓扑。



5 部署指南

5.1 安装部署手册

部署架构图

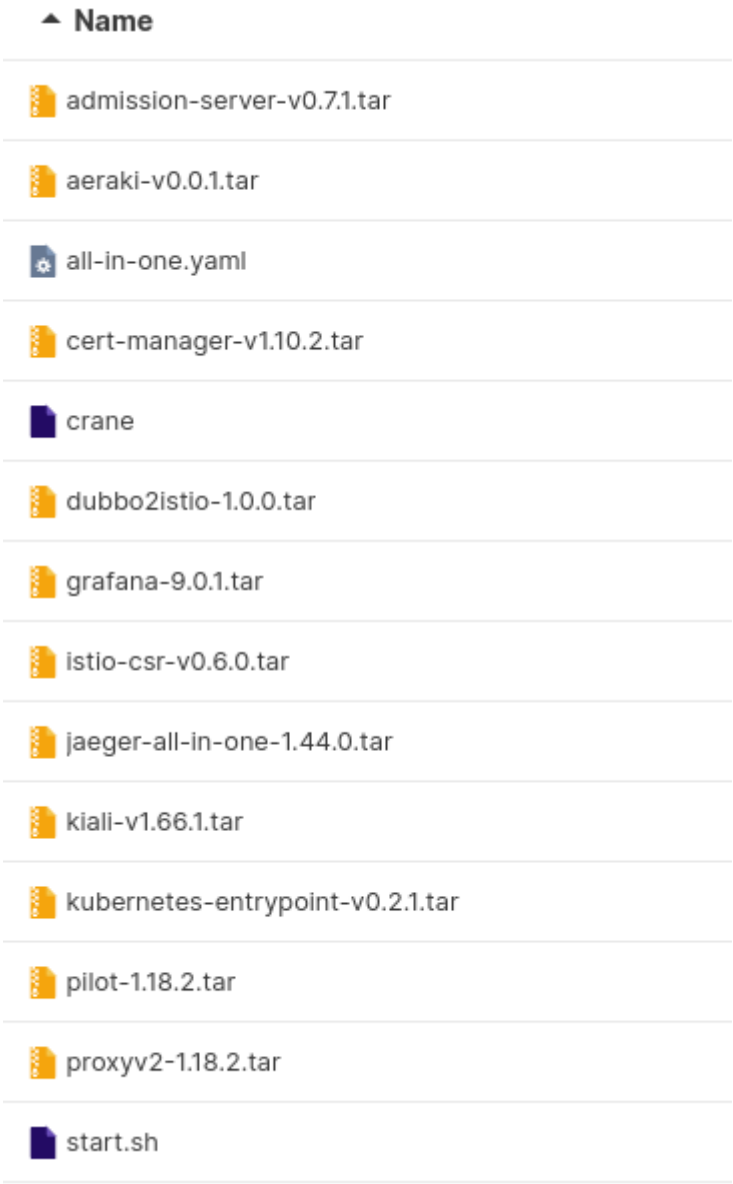
服务网格作为产品，当前因为部署环境不同，需要执行不同方式，主要有以下两种部署环境

- EOS： 在该环境中部署，使用云产品部署模式，具体不做介绍
- EKS：在该环境中部署，因无法使用云产品方式，所以针对不同版本和架构下载对应的物料包，具体过程将在下文中介绍

物料和部署

以下物料仅针对 EKS 环境中部署服务网格产品

编号	名称	作用	下载链接
1	servicemesh_1.0.1_linux_amd64.zip	包含服务网格产品镜像，安装脚本和必要的工具，安装在amd64 架构下的 eks 环境	http://minio-console.easystack.io/
2	servicemesh_1.0.1_linux_arm64.zip	包含服务网格产品镜像，安装脚本和必要的工具，安装在arm64 架构下的 eks 环境	http://minio-console.easystack.io/



压缩包内包含以下内容：

步骤

1. 进入 eks 主机内，注意需要是 master 主机，不能是 跳板机
2. 拷贝物料包(zip压缩文件) 到 eks master 的主机内
3. 解压物料包，并进入目录内
4. 执行 bash start.sh -h，该脚本有以下介绍

脚本配置说明:

```
start.sh usage:
     -k [filepath]: 指定 kubeconfig 文件, 如 /root/.kube/config
     -l: 设置 istio-ingress 服务类型为 loadbalance, 默认不是 loadbalance
     -h: 打印帮助信息
```

咨询热线：400-100-3070

北京易捷思达科技发展有限公司：
北京市海淀区西北旺东路10号院东区1号楼1层107-2号
南京易捷思达软件科技有限公司：
江苏省南京市雨花台区软件大道168号润和创智中心4栋109-110

邮箱：
contact@easystack.cn (业务咨询)
partners@easystack.cn(合作伙伴咨询)
marketing@easystack.cn (市场合作)