百度智能云

CSM 文档

目录

功能发布记录

功能发布记录

发布时间	功能概述
2025-05	• 配置禁用 Sidecar 注入，详情参见 常见问题-如何禁用 sidecar 注入。
2025-04	• CSM 新增注解支持所有端口的入口流量都不经过 Sidecar 代理，详情参见 灵活控制流量免于经过 Sidecar 代理。
2024-10	• CSM 支持使用 Wasm Filter 扩展服务网格数据面的能力，详情参见 使用 Wasm Filter 扩展数据面。
2024-09	• CSM 支持链路追踪功能，可将请求经过的所有服务链路投递至第三方服务地址，以便您进行链路分析、快速排障，详情参见 链路追踪。 • CSM 支持灵活配置 Pod 端口流量是否经过 Sidecar 代理，以满足特定场景需求，详情参见 灵活控制流量免于经过 Sidecar 代理。
2024-08	• 托管网格的 Istio 资源支持存放至数据面集群，您可在创建托管网格时修改 Istio 资源配置，详情参见 托管网格实例配置。
2024-03	• 发布流量泳道功能，支持流量的全链路灰度，详情参见 流量泳道。
2023-12	• 新增诊断工具，支持服务网格异常分析及查看代理状态和数据面配置，详情参见 诊断工具。
2023-09	• 新增组件管理，支持第三方注册中心（consul），详情参见 组件管理。
2023-08	• 独立网格新增 1.16.x 版本 • 托管网关支持 ingress
2023-06	• 独立网格支持 Dubbo、bRPC 等私有协议 • 独立网格新增 1.14.x 版本
2023-05	• 服务网格 CSM 取消公测
2023-03	• 托管网关支持 Prometheus 监控，详情参见 托管网关-开启 Prometheus 监控。 • 托管网关支持 TLS 硬件加速、实例规格变更，详情参见 网关管理。
2023-02	• 托管网关支持 BLS 日志服务，支持数据面日志持久化，详情参见 服务网格对接 BLS。
2023-01	• 托管网关新增域名管理功能，支持 HTTPS，详情参见 网关管理-域名管理。 • 服务网格 CSM 支持广州地域

2022-12	• 发布托管网关，融合流量网关和微服务网关能力，详情参见 网关管理。
2022-11	• 新增托管服务网格类型，由 CSM 托管 Istio 控制平面等核心组件，兼容社区开源 Istio 服务网格。适用于稳定性要求较高的生产环境，具备简单、低成本、高可用、无需运维管理 Istio 控制平面的特点，详情参见 实例管理。
2022-06	• 可观测能力增强，对接 Prometheus 监控服务，一键完成监控指标采集和大盘展示。 • 选择性服发现：支持命名空间粒度的 CCE 集群管理，提升服务发现和配置下发效率。 • 支持多用户访问控制 IAM，提供系统策略和用户自定义策略，实现 CSM 产品级别和实例级别权限控制。 • 支持自定义 Sidecar 资源配置，用户可灵活配置 Sidecar 资源用量。
2022-04	• 发布服务网格 CSM 服务，提供免费公测，支持华北-北京、华北-保定、华南-广州、金融华中-武汉、中国香港地域。 • 服务网格 CSM 产品，全面兼容原生 Istio 服务网格，提供丰富的负载均衡、路由转发、超时重试、熔断限流、加密鉴权策略，支持跨集群、多语言、多协议、精细化服务治理，使服务间调用更加便捷、稳定、安全、易于追踪、便于管理。

产品公告

CSM结束公测公告

百度智能云服务网格产品CSM（Cloud Service Mesh）为了帮助用户更便捷的体验产品，决定于2023年05月31日结束公测，并对所有用户直接开放。

新用户将无需提交申请信息，可以直接使用CSM，老用户不受结束公测的影响。结束公测后，CSM将免费为用户提供服务，用户只需要为相关产品的资源使用付费（如BLB、ENI）。

版本支持说明

Istio版本	支持的集群版本
1.13.2	1.20.X 、 1.22.X
1.14.6	1.22.X 、 1.24.X
1.16.5	1.22.X 、 1.24.X

产品描述

产品介绍

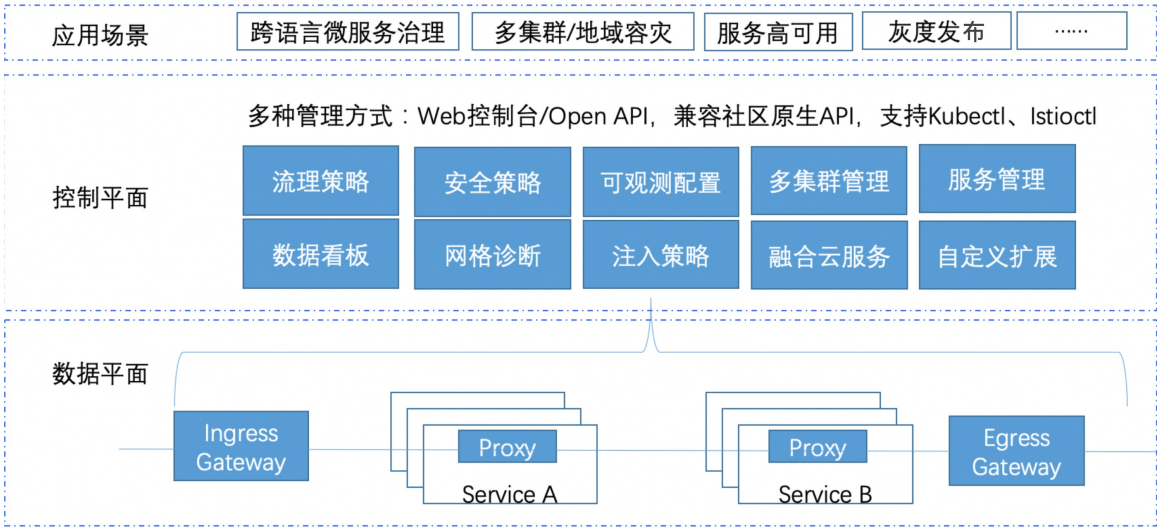
概述

百度智能云服务网格（Cloud Service Mesh，简称CSM）全面兼容原生Istio服务网格，提供丰富的负载均衡、路由转发、超时重试、熔断限流、加密鉴权等策略、支持跨集群、多语言、多协议、精细化服务治理，使服务间调用更加便捷、稳定、安全、易于追踪、便于管理。通过服务网格，将微服务治理能力下沉到基础设施层，让业务开发与微服务治理解耦，极大地减轻开发与运维的工作负担。

核心概念

百度智能云服务网格由Istio原生组件、百度增强能力的控制平面，由网关、Sidecar代理组件的数据转发平面两个平面构成，针对微服务治理的规则，总体上控制平面下发策略、数据平面执行策略。
产品完全兼容社区的Istio、Envoy，用户可通过百度智能云控制台、原生API等多种方式进行服务网格的产品管理。

- **产品架构**



- **托管网格**

  Istio相关组件部署在托管集群中，简化控制平面运维，用户无需运维控制平面。

- **控制平面**

  控制平面是告诉数据平面如何路由流量的组件，它存储、管理配置，帮助管理员与Sidecar代理进行交互并控制Istio服务网格。在转换分离的架构体系下，微服务间流量不会发送至控制面，在控制面短暂异常时，数据平面也会使用已下发的规则进行流量转发。

- **数据平面**

  Istio默认采用Envoy作为Sidecar和Gateway构成数据平面，Envoy代理业务流量，并按控制面下发的策略进行转发。服务网格CSM默认采用Envoy，保持与社区全面的兼容，同时可支持Proxyless等其它的数据面选型。

- **Istio 资源**

  服务网格的流量管理、可观测、安全等能力，通过Istio的众多CRD实现相关的配置，这些CRD在服务网格CSM中称之为Istio资源。包括Virtual Service、Destination Rule等，服务网格CSM完全兼容社区Istio的CRD。详细CRD及使用方式，参考Istio官方文档。

- **集群管理**

  同一个服务网格实例，可以同时治理多个Kubernetes集群中的服务，集群管理帮助用户快速将Kubernetes集群添加至CSM或从CSM移出。请确保添加的Kubernetes集群中运行的代理容器能访问服务网格实例暴露的控制面地址。

- **Sidecar 注入**

  服务网格的代理容器，以Sidecar容器的形式和业务容器部署在同一个Pod中，Sidecar代理容器不需要用户创建工作负载时手动添加。通过在命名空间或工作负载的Annotation中标识，服务网格就会在工作负载创建时，同时创建好Sidecar代理容器，这一过程称为Sidecar注入。通过服务网格CSM产品，可以实现Sidecar容器资源配额和是否开启注入的快速配置。

- **服务**

  服务网格推荐使用Kubernetes服务发现机制，同时服务网格支持用户通过Service Entry CRD手动将服务注册至服务网格。服务网格CSM中的服务，代表着一个提供服务的入口和对应的工作负载。当多个Kubernetes Service或Service Entry注册的是同一个服务的入口时，对于服务网格CSM认为是同一个服务，Kubernetes Service或Service Entry对应的多个工作负载，均视为这一个服务的实例。

优势

托管服务网格

- Istio相关组件部署在托管集群中，简化控制平面运维，用户无需运维控制平面。

完全兼容 Istio

- 快速跟进社区更新。完全兼容原生Istio和Envoy，独立网格支持Istio 1.13，托管服务网格支持 Istio 1.14。

高性能高可用

- 控制面和数据面在Istio社区版本基础上进行优化、加固，提供可视化视图辅助运维。

多集群统一流量管理

- 单实例可管理多Kubernetes集群，实现多集群统一服务发现，统一流量调度。

丰富的治理策略

- 提供丰富的负载均衡、路由转发、超时重试、熔断限流、加密鉴权策略。

功能

流量管理

- 服务网格丰富的流量管理规则可以让用户轻松地控制服务之间的流量，提供开箱即用的熔断、限流、重试等能力、支持动态进行多种负载均衡、百分比路由、基于成份的路由等转发规则。

可观测

- 服务网格内的所有通信无侵入的生成详细的遥测数据，可以提供详细的指标、分布式跟踪和完整的访问日志。

安全

- 提供了强大的身份、强大的策略、透明的TLS加密，以及验证、授权和审计工具来保护您的服务和数据。

应用场景

应用场景

多语言微服务治理

挑战：

- 业务代码与服务调用耦合
  基于Lib/SDK的微服务治理，SDK的维护、升级等变更会造成整个服务的维护升级、业务开发与微服务升级相互制约。
- 微服务框架治理能力差异大
  同一种框架无法支持多语言的微服务治理。使用不同的技术栈来开发微服务带来开发效率的提升，微服务治理手段参差不齐。

我们能提供：

- 开发运维关注点分离
  通过服务网格代理，将服务治理能力与业务代码相互独立，实现分离关注点。
- 统一的微服务治理方案
  服务网格能够支持多种开发语言的服务治理。无需针对多种开发语言维护多套微服务技术体系，明显节省人力成本。

跨集群/地域灾备

挑战：

- 服务注册发现难
  微服务容器化可通过Kubernetes的服务（Service）进行集群内服务发现。跨集群场景，如额外部署注册中心，即引入额外维

护成本也不具备Kubernetes原生方案在时效性等方成的诸多优势。

- 流量调度粒度粗
  集群入口级别的故障容灾策略，单一个微服务发生故障就需要整集群切流，部分可用集群承担了全部的流量压力，带来扩容压力大、资源浪费等问题。

我们能提供：

- 零成本跨集群服务发现
  无需额外部署注册中心，基于Kubernetes原生服务发现全自动实现跨地域的服务发现机制，实例迁移1秒同步。
- 服务级流量调度
  通过服务网格，可以实现服务级别的自动切流和自动恢复。影响范围小，资源变更少。

服务高可用

挑战：

- 故障场景难以避免
  微服务的数量众多，每个微服务的实例众多，不可避免的出现实例级别和微服务级别的故障，对重大隐患预防和故障止损提出了极高的要求。
- 可观测难
  微服务间流量调用复杂，微服务数量和实例数量众多，监控配置复杂，故障定位难。

我们能提供：

- 故障隔离
  通过主动和被动的健康检查机制，在流量转发时自动规避故障实例，提供限流、熔断能力，在服务级别故障发生时，防止雪崩，保障全局可用性。
- 故障注入
  用户可指定服务、指定比例，注入超时、中止等不同类型故障，提前发现微服务体系可用性的不足。
- 多种重试机制
  支持超时或指定的返回信息进行请求重试，提升用户请求成功率；支持重试比例设置，防止重试造成过载。
- 快速获得可观测能力
  Sidecar内置Exporter，支持OpenTelemetry接口，可无侵入采集流量指标、流量日志和调用链信息。

灰度发布与测试环境复用

挑战：

- 灰度发布场景多样，随着业务的运营精细化，灰度版本面向的目标用户群体有多种不同的分类方式。
- 测试环境搭建费时费力，单个微服务的测试时，也需要同时部署上下游的多个微服务，造成资源和人力的浪费。

我们能提供：

- 灵活的流量灰度策略
  服务网格可基于百分比、流量特征等多种方式进行流量分发，支撑用户基于不同场景的灰度发布需求。
- 测试环境复用能力
  借助服务网格的流量规则动态下发、流量镜像等能力，可复用基准环境或线上环境现有资源，仅额外部署要测试的单个微服务，即可达到端到端的测试效果。

快速入门

快速入门

快速入门流程

本文介绍如何使用控制台快速创建服务网格CSM实例(独立网格与托管网格)，将微服务应用纳入服务网格管理，用时大约五分钟。

基本操作流程如下图所示：

使用准备

注册及实名认证

使用百度智能云服务网格CSM前，用户需要拥有一个百度智能云账号并完成实名认证，具体操作如下：

1. 注册并登录百度智能云平台，请参考注册和登录。
2. 完成实名认证，操作细节请参考实名认证，实名认证后才可开通服务网格CSM。
3. 申请受理后，在百度智能云控制台 导航栏中选择“产品服务 > 云原生 > 服务网格 CSM”，即可开通使用服务网格CSM。

费用说明

1. 创建服务网格CSM实例，登录服务网格CSM控制台，点击"创建实例"开启服务，服务网格CSM独立网格、托管网格产品免费为用户提供服务，用户只需为使用的关联产品进行付费。
2. 服务网格创建同时，会创建按需计费负载均衡BLB实例，需要您的账户余额大于或等于100元。关于充值，请参见如何充值。

开始使用

创建实例

1. 首次使用服务网格CSM，登录百度智能云控制台，选择“产品服务 >云原生> 服务网格 CSM”，会进入“全局概览>指引”页面，可点击“创建实例”创建服务网格实例。
2. 在创建服务网格页面，完成基础配置，详见操作指南>实例管理。
3. 点击“确认”服务网格开始进行实例创建，在网格列表页面，可以看到“部署中”的服务网格实例，大约几分钟后会转为“运行中”。

接入服务

将微服务接入服务网格进行治理包括两个核心操作，服务发现和服务网格代理Sidecar的注入：

1. 点击具体的服务网格实例，通过服务网格“集群管理”可将CCE集群添加至服务网格，添加后，服务网格可自动发现CCE集群上运行的Kubernetes Service和对应的工作负载，详见操作指南>集群管理。
2. 对于Kubernetes工作负载，服务网格的具体CRD治理策略的执行与生效，可通过自动注入的方式，完成服务网格Sidecar代理容器的注入，详见操作指南>注入配置。

配置治理策略

服务网格CSM通过Istio资源（如 Virtual Service、Destination Rule）描述微服务治理策略，“Istio资源”页面通过控制台、原生API等不同方式提交的资源进行展示，用户可在“Istio资源”页面按需进行增删改查操作。详见操作指南>Istio 资源管理。

托管网格使用Demo

1.创建托管服务网格

进入百度智能云控制台 -> 选择服务网格 CSM -> 创建网格（选择实例类型为托管网格、填写必要的信息、选择托管实例所属的网络信息）-> 确认创建即可。

2.添加用户集群

添加需要使用服务网格的业务集群，以便 CSM 托管网格实例可以管理用户集群上的微服务应用。

因此需要提前准备一个百度智能云容器引擎 CCE 集群，用于部署示例应用程序。具体创建集群步骤可参考文档：创建集群。

注意：

- 托管网格纳管的CCE集群，如果选择vpc-eni网络模式，需要确保选择的安全组出站规则配置了能访问托管网格实例vpc的网段
- CCE集群创建worker节点时，镜像推荐选择unbuntu
- Baidulinux 操作系统请开启相关内核模块

CCE 集群准备完成后，可在 CSM 中添加用户集群，路径：进入百度智能云控制台 -> 选择服务网格 CSM -> 点击对应的网格实例 -> 集群管理 -> 添加集群 -> 确认。

3.开启Sidecar注入

开启自动注入功能，可实现业务无感注入 Sidecar 代理，该 Sidecar 代理用于实现服务网格中各项能力。开启对应命名空间自动注入功能后，重启或重新部署对应命名空间下的工作负载，可实现该命名空间下的 Pod自动注入了 Sidecar 容器。

路径：进入百度智能云控制台 -> 选择服务网格 CSM -> 点击对应的网格实例 ->添加集群 ->注入配置 -> 开启对应命名空间自动注入开关。

4.部署应用程序

在 CCE 容器引擎上可通过页面配置 或 YAML 的方式部署工作负载和服务。

路径：进入百度智能云控制台 -> 选择容器引擎 CCE -> 点击对应的集群 -> 工作负载 -> 无状态服务部署 -> 新建无状态部署/使用 YAML 创建。

以本文以 YAML 创建为例，consumer-demo的K8S deployment YAML 定义如下：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: consumer-demo-deployment
  labels:
    app: consumer-demo
    name: consumer-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer-demo
  template:
    metadata:
      labels:
        app: consumer-demo
    spec:
      restartPolicy: Always
      containers:
        - name: consumer-demo
          image: registry.baidubce.com/csm-offline/bms-orignal-consumer:dev
          imagePullPolicy: Always
          ports:
            - containerPort: 9999
```

为演示的演示服务网格的功能，需要为provider-demo定义不同的版本，即v1和v2。

```
  provider-demo v1版本K8S deployment YAML 定义如下：
apiVersion: apps/v1
kind: Deployment
metadata:
  name: provider-demo-deployment-v1
  labels:
    app: provider-demo
    name: provider-demo
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: provider-demo
      version: v1
  template:
    metadata:
      labels:
        app: provider-demo
        version: v1
    spec:
      restartPolicy: Always
      containers:
        - name: provider-demo
          image: registry.baidubce.com/csm-offline/bms-orignal-provider:dev
          imagePullPolicy: Always
          ports:
            - containerPort: 10001
```

provider-demo v2版本K8S deployment YAML 定义如下：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: provider-demo-deployment-v2
  labels:
    app: provider-demo
    name: provider-demo
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: provider-demo
      version: v2
  template:
    metadata:
      labels:
        app: provider-demo
        version: v2
    spec:
      restartPolicy: Always
      containers:
        - name: provider-demo
          image: registry.baidubce.com/csm-offline/bms-orignal-provider:dev
          imagePullPolicy: Always
          ports:
            - containerPort: 10001
```

接下来进入百度智能云控制台 -> 选择容器引擎 CCE -> 点击对应的集群 -> 网络 -> 服务 -> 新建服务。

provider-demo Service YAML 定义如下：

```
  apiVersion: v1
kind: Service
metadata:
  name: provider-demo
  annotations:
    prometheus.io/scrape: "true"
spec:
  selector:
    app: provider-demo
  type: ClusterIP
  sessionAffinity: None
  ports:
  - protocol: TCP
    port: 80
    targetPort: 10001
    name: http-port
```

验证流量

在本文的示例中 Consumer Demo 访问 Provider Demo 时通过 K8S Service 轮询访问后端服务，即轮询访问 V1 和 V2 版本 Provider Demo Pod，而无法精准控制流量。

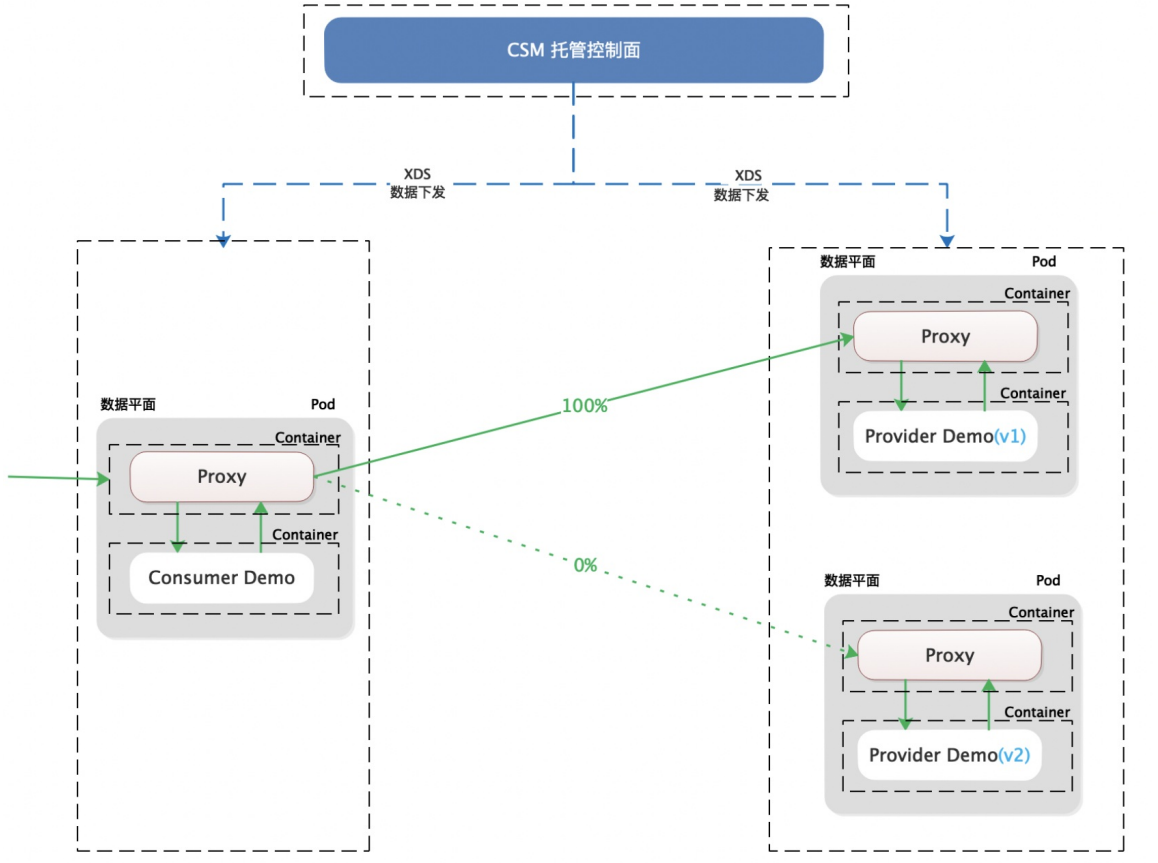
进入consumer-demo的pod容器中，访问接口：

```
 while true; do curl http://127.0.0.1:9999/echo-rest/hello && echo && sleep 1; done;
while true; do curl -H 'x-b3-user: admin' http://127.0.0.1:9999/echo-rest/hello && echo && sleep 1; done;
```

5.服务治理策略下发

现所有准备工作都已完成，即可快速使用 CSM 提供的服务网格微服务治理能力。

一个例子，通过服务网格服务治理能力可实现 Consumer Demo 访问 Provider Demo 的流量 100% 发往Provider Demo V1版本，而不再将流量发往 Provider Demo V2 版本，如下图所示：



进入百度智能云控制台 -> 选择服务网格 CSM -> 点击对应网格实例 -> Istio 资源管理 -> 创建 Istio 资源。

Istio 资源 YAML 定义如下：

```
  apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: dr-provider-demo
spec:
  host: provider-demo.default.svc.cluster.local
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

```
  apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: vs-provider-demo
spec:
  hosts:
  - provider-demo.default.svc.cluster.local
  http:
  - route:
    - destination:
        host: provider-demo.default.svc.cluster.local
        subset: v1
      weight: 100
    - destination:
        host: provider-demo.default.svc.cluster.local
        subset: v2
      weight: 0
```

验证流量

在本文的示例中 Consumer Demo 访问 Provider Demo 时 流量只会访问 V1 版本，而不再访问V2 版本，通过 CSM 可实现精准控制流量走向的目的。

同样可以进入consumer-demo的pod容器中，访问接口：

```
  while true; do curl http://127.0.0.1:9999/echo-rest/hello && echo && sleep 1; done;
while true; do curl -H 'x-b3-user: admin' http://127.0.0.1:9999/echo-rest/hello && echo && sleep 1; done;
```

另一个例子，服务网格也支持istio原生的envoyfilter资源

进入百度智能云控制台 -> 选择服务网格 CSM -> 点击对应网格实例 -> Istio 资源管理 -> 创建 Istio 资源。

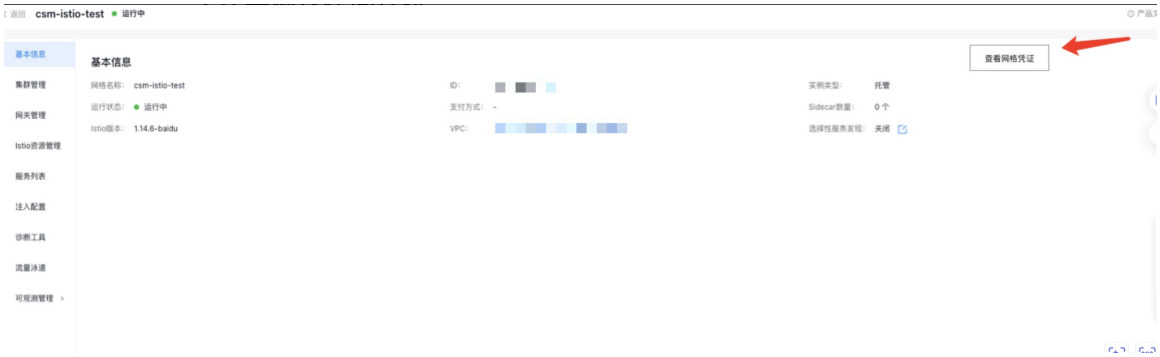
```
  apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: add-baibu-header
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      istio: ingressgateway  # 可以根据需要更改为特定的工作负载选择器
  configPatches:
  - applyTo: HTTP_FILTER
    match:
      context: SIDECAR_OUTBOUND
    patch:
      operation: INSERT_BEFORE
      value:
        name: envoy.lua
        typed_config:
          "@type": type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua
          inlineCode: |
            function envoy_on_request(request_handle)
              request_handle:headers():add("baidu", "some-value")
            end
```

6.托管网格控制面监控

具体可参考：对接Cprom实现监控告警

7.查看托管网格访问凭证

进入CSM控制台——>托管网格实例——>基本信息——>查看网格凭证，同时支持VPC访问凭证和公网访问凭证，您可以进行下载或者复制。您可利用访问凭证通过 kubectl 命令等方式提交 Istio 相关 CRD 资源(如 Envoyfilter、VirtualService、DestinationRule等)。



产品定价

产品定价

产品定价

百度智能云服务网格CSM为用户提供免费独立网格与托管网格，用户只需为使用的关联产品（如 BLB、CCE、CProm）等进行付费，具体计费标准可以参考具体产品的价格详情。

操作指南

实例管理

概述

百度智能云服务网格 CSM 提供托管网格与独立网格两种网格实例类型：

- 托管网格：CSM 托管 Istio 控制平面等核心组件，兼容社区开源 Istio 服务网格。适用于稳定性要求较高的生产环境，具备简单、低成本、高可用、无需运维管理 Istio 控制平面的特点。
- 独立网格：兼容社区开源 Istio 服务网格，Istio 相关组件部署在用户集群，用户完全拥有所有组件的权限。适用于对 Istio 有深入的理解与掌握，想要深度研究与定制化 Istio 的用户。

本文介绍如何通过控制台管理 CSM 实例，包括创建、查看、删除操作。

使用前提

- 已经开通 服务网格 CSM、负载均衡 BLB、容器引擎 CCE，且操作人员具有相关产品的权限。
- 独立网格实例的控制面运行在用户已经创建的 CCE 集群上，创建 CSM 独立网格实例时，需要选择部署控制面的 CCE 集群：
  - 部署 CSM 的 CCE 集群有 2个 2核4G 空闲资源的 worker 节点，用于运行控制面工作负载。
  - 部署 CSM 的 CCE 集群未安装其它 CSM 实例也未被其它 CSM 实例管理。
  - 部署 CSM 的 CCE 集群，如用户已安装过 Istio，服务网格 CSM 实例会删除原有 Istio 组件，并重新安装服务网格 CSM 相关组件。
- 开启监控指标采集，相关 CCE 集群需要使用 CProm 实例采集监控指标。
- 托管网格实例的控制面运行在百度智能云托管的 CCE 集群上。

创建 CSM 实例

按照以下步骤在控制台创建服务网格 CSM 实例：

1. 登录百度智能云控制台，选择**产品服务 > 云计算>容器 > 服务网格 CSM**。
2. 在左侧导航栏选择**全局概览**，在指引页中单击**创建实例**进入实例创建页面。也可在左侧导航栏选择**网格列表**，在网格列表页中单击**创建网格**进入实例创建页面。

3. 在创建网格页面，按需填写网格创建相关配置。
4. 点击**确认**创建 CSM 实例，服务网格开始进行实例创建，在网格列表页面，可以看到状态为“部署中”的服务网格实例，大约几分钟后会转为“运行中”。

托管网格实例配置说明

配置项	说明
当前地域	服务网格所在的地域，托管网格实例目前支持地域：华北-北京、华南-广州。
网格名称	服务网格的名称，创建后不可修改。
Istio 版本	使用的 Istio 版本。
控制面 BLB	会新建后付费的 BLB 实例，用于提供服务网格部署集群外的访问入口，便于跨集群服务治理和控制面管理，查看 BLB计费规则。
网络类型	托管网格实例所在的 VPC 网络和子网，若无可用 VPC 网络，可前往 私有网络VPC 创建。
安全组	托管网格实例需要使用的默认安全组。
服务发现范围配置	默认不开启，开启后，CSM 仅会监视和处理 CCE 集群匹配标签规则的命名空间中的服务和工作负载，提升服务发现和配置下发的效率。
Istio 资源配置	Istio 资源配置所在的集群。用户可将 Istio 资源提交至控制面托管集群或数据面集群： • 控制面托管集群：Istio 资源将提交至托管网格所在集群中。 • 数据面集群：Istio 资源将提交至首个被网格实例纳管的数据面集群中。
API Server 访问	若 Istio 资源配置至控制面托管集群，则可选择是否开启公网访问。
链路追踪	默认不开启，开启后，可将网格数据面内产生的 tracing 数据上报到第三方 Jaeger/Zipkin 服务，详情参见 链路追踪。

独立网格实例配置说明

配置项	说明
当前地域	服务网格所在的地域，独立网格实例目前支持地域：华北-北京、华北-保定、华南-广州、华东-苏州。
网格名称	服务网格的名称，创建后不可修改。
Istio 版本	使用的 Istio 版本。
主集群	独立网格控制面部署在用户自有的 CCE 集群上，用户需指定运行控制面的主集群。主集群需满足以下条件： • 至少有2个可用资源大于 2核4G 的 Worker 节点，若没有合适的集群，可前往 创建CCE集群。 • 仅能选择具有管理员权限的 CCE 集群，若您没有 CCE 管理员权限，可前往 权限管理 申请。 • 不可选择已被其它 CSM 实例管理的 CCE 集群，若 CCE 集群中安装过服务网格，会被覆盖安装，已经提交过的 Istio 资源不会被删除。
关联 BLB	会新建后付费的 BLB 实例，用于提供服务网格部署集群外的访问入口，便于跨集群服务治理和控制面管理，查看 BLB计费规则。
服务发现范围配置	默认不开启，开启后，CSM 仅会监视和处理 CCE 集群匹配标签规则的命名空间中的服务和工作负载，提升服务发现和配置下发的效率。
日志服务	默认不开启，开启后，将对接日志收集与投递服务 BLS，实现对数据面日志的持久化、查询和分析，详情参见 对接BLS实现日志持久化。
监控指标采集	默认不开启，开启后，会使用 CProm 采集服务间流量指标并进行大盘展示，详情参见 对接CProm实现监控告警。
链路追踪	默认不开启，开启后，可将网格数据面内产生的 tracing 数据上报到第三方 Jaeger/Zipkin 服务，详情参见 链路追踪。

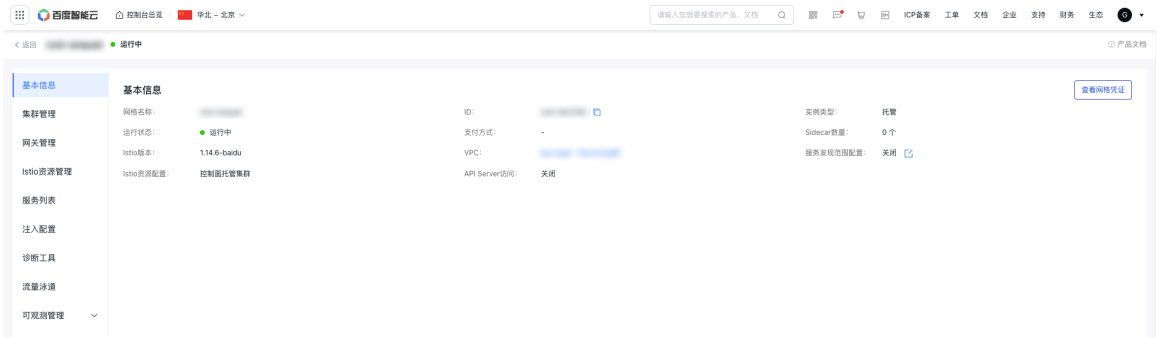
查看 CSM 实例

服务网格 CSM 实例的状态，可在**全局概览** > **资源仪表盘**和**网格列表**两个页面中查看。

1. **全局概览** > **资源仪表盘**页面，对所有地域的服务网格 CSM 实例进行了汇总展示，包括各地域的实例数量，实例整体的健康状况，各服务网格实例的接入规模和各服务网格实例运行状态、地域、纳管的集群数量等关键信息。

> **关于服务网格的健康度：**指正常“运行中”的服务网格数量在总量中的占比，部署中、删除中等过程状态的实例，会归类为“变更中”，各类未正常运行的实例归类为“异常”，具体异常状态可到具体实例侧查看。

2. **网格列表**页，展示了服务网格 CSM 实例更具体的信息，点击服务网格实例名称，可以查看服务网格实例的完整信息。
3. 以托管服务网格实例为例，您也可以在服务网格实例详情页面点击右上角按钮查看网格凭证。



删除 CSM 实例

- 删除服务网格 CSM 实例，将立即删除服务网格 CSM 实例的控制面组件、创建时开通的 BLB。
- 服务网格的 Sidecar 代理容器与业务容器在同一个工作负载中，不会立即删除，自动注入的 Sidecar，将会在业务工作负载重启后移除。
- 非自动注入的数据面代理，则需要额外的移除操作，如虚拟机工作负载接入服务网格时，在虚拟机中部署的代理容器。

在网格列表页，可通过选择操作列的**删除**按钮进行删除服务网格实例操作。

> **注意：**
>
> - 服务网格实例删除后，用户在实例下配置的 Istio 资源不会保存，通过服务网格实现的微服务治理能力将失效，为保证服务可用性不受损，删除前请确认服务已通过其它方式实现了微服务治理能力。
> - 在生产环境中，如需删除服务网格，建议逐个为服务移除 sidecar，确认服务可用后，再进行服务网格实例的删除操作。

集群管理

概述

服务网格所治理的微服务，可以运行在一个或者多个 Kubernetes 集群中，服务网格会自动发现管理用户加入服务网格的 Kubernetes 集群中的服务和对应的工作负载。百度智能云提供了托管和独立部署的 Kubernetes 产品 容器引擎 CCE。本节介绍如何在服务网格控制台管理 CCE 集群。

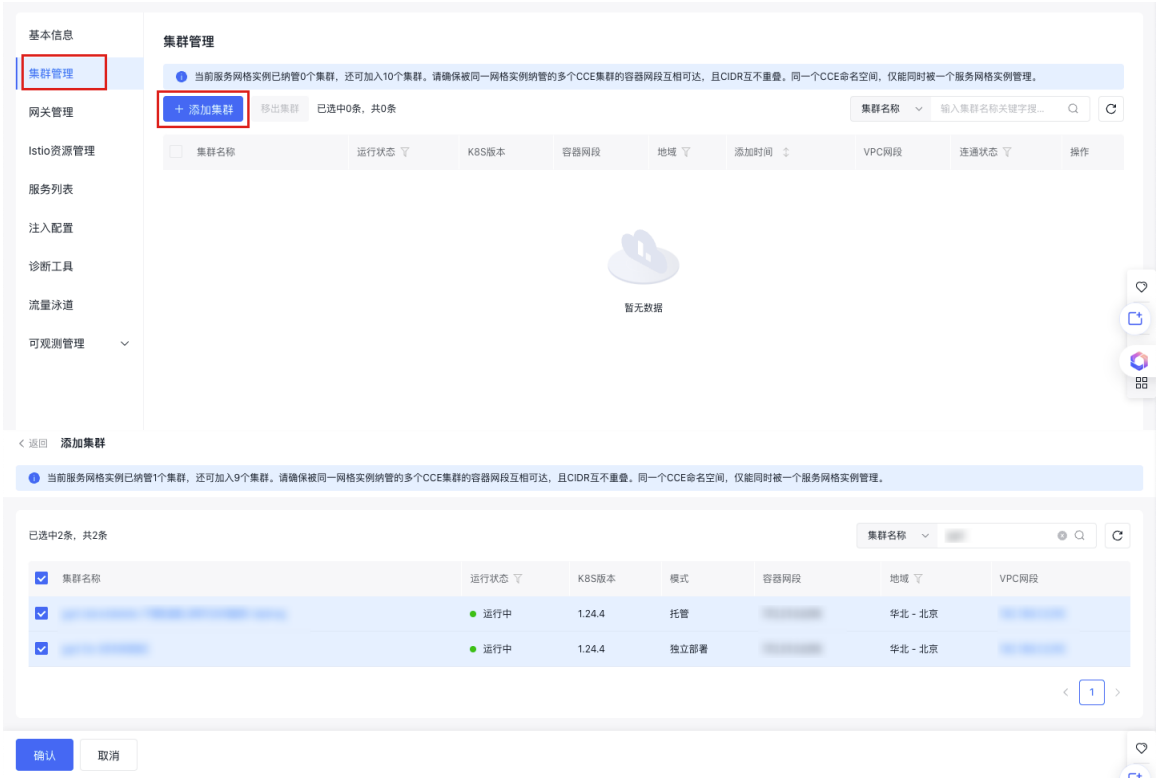
注意事项

- 纳管的多个 CCE(Kubernetes) 集群，所有集群中所有 Pod 和服务的 IP 地址都是可直接路由的，不会发生冲突，同时保证在一个集群中分配的 IP 地址不会在另一个集群中同时出现。实现方式，参见 CCE>操作指南>网络管理。
- 同一个 CCE 集群，不能同时被多个服务网格 CSM 实例管理。服务网格控制台操作时会避免这种情况的发现。如用户已在 CCE 集群上自主安装过 Istio，服务网格 CSM 会删除原有 Istio 组件并安装 CSM 控制面组件。用户提交过的 Istio 资源（Virtual Service、Destination Rule 等）不会删除。
- 托管服务网格实例控制面所在集群已被托管，集群管理处不可见。

- 托管服务网关实例不会在用户集群部署 Istio 控制平面，托管网格在集群管理处只能纳管跟用户集群处于同一 VPC 的 CCE 集群，否则需要自行打通托管网格实例 VPC 网络与用户集群 VPC 网络。
- 托管网格纳管的 CCE 集群，如果选择 vpc-eni 网络模式，需要确保选择的安全组出站规则配置了能访问托管网格实例 vpc 的网段。
- CCE 集群创建 worker 节点时，镜像推荐选择 unbuntu。

添加集群

1. 登录百度智能云控制台，选择**产品服务 > 云计算>容器 > 服务网格 CSM**。
2. 在服务网格控制台，对应地域的**网格列表**页中找到要操作的服务网格实例，点击**网格名称/ID**进入网格实例详情页，在左侧导航栏选择**集群管理**。



3. 独立网格实例，已默认添加了运行服务网格实例的 CCE 集群（主集群），且不可移出。在集群管理页面通过**添加集群**按钮进入添加集群页面，可添加更多集群。
4. 在添加集群页面中，勾选要添加的 CCE 集群，点击**确认**。

> **说明：**
>
> - 若需使用服务网格 CSM 纳管集群，需具有 CCE 集群的管理员权限，若您无相关权限，可前往 权限管理 申请。
> - 服务网格 CSM 的不同版本兼容不同 Kubernets 版本的 CCE 集群，详情参见 版本支持说明。在添加集群页面中有相应的提示，建议升级 CSM 实例的版本或 CCE 实例版本以保证可用性。

5. 单个 CSM 实例，最多可添加 10 个托管或独立 CCE 集群，独立网格暂不支持管理 CCE 的Serverless 集群。
6. CCE 集群添加后，会展示在集群列表中，并且展示“已联通”或“未联通”状态。“已联通”说明服务网格已可以正常发现被管理集群的服务和服务对应的实例。

> **说明：**
>
> - 添加 CCE 集群至服务网格 CSM 实例后，服务网格实例并不会立即在 CCE 集群中对业务注入 Sidecar 代理，也不会改变

> 集群中服务间的流量转发方式，**添加集群这一过程，对在 CCE 中的存量业务无任何感知。**
>
> - 用户对 CCE 集群中的业务进行 Sidecar 注入后，服务网格代理才会开始代理流量，关于 Sidecar 注入，详见 操作指南>注入配置。

7. 托管服务网格为例，展示集群管理。若在创建托管网格实例时 Istio 资源配置选择数据面集群，CSM 默认会将第一个纳管的集群作为 Istio 资源配置集群。

移出集群

用户可以通过服务网格 CSM 控制台随时将不需要服务网格 CSM 进行服务治理的 CCE 集群移出。

> **注意：**独立网格实例默认添加的 CCE 集群（运行服务网格 CSM 实例的主集群）无法被移出，托管网格实例设置的 Istio 资源配置集群无法被移出。

1. 登录百度智能云管理控制台，选择**产品服务>云计算>容器>服务网格 CSM**。
2. 在服务网格控制台，对应地域的**网格列表**页中找到要操作的服务网格实例，点击**网格名称/ID**进入网格实例详情页，在左侧导航栏选择**集群管理**。
3. 在集群列表中，找到实例，在操作列点击**移出**，即可完成移出操作。

集群移出后，Sidecar 的注入规则将同时被移除。自动注入的服务网格 Sidecar 在工作负载重新启动前仍然有效，继续使用移出前的规则并转发流量，在工作负载重启后，重新拉起的工作负载将不会再次注入 Sidecar，不再执行服务网格 CSM 配置的流量规则。

4. 托管服务网格删除示例如下所示：

网关管理

概述

本节介绍服务网格CSM下的托管网关实例的创建、查看、删除等相关操作。

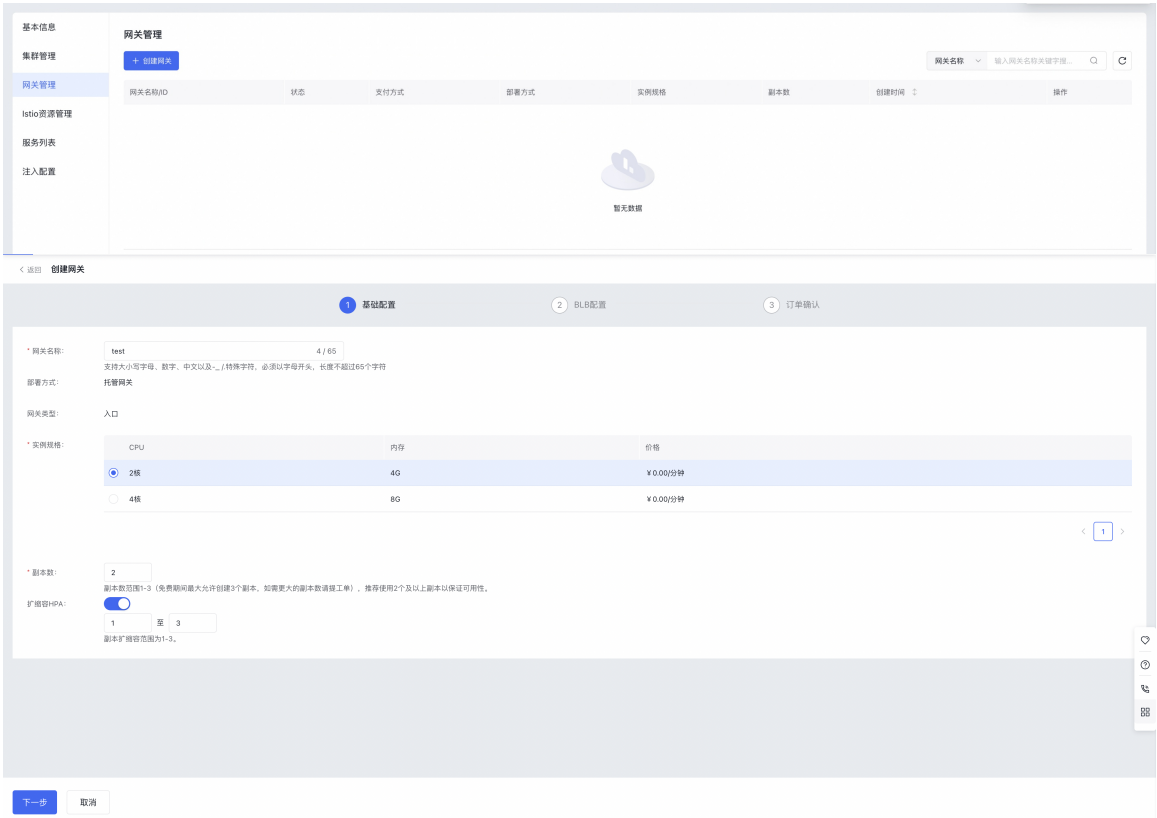
使用前提

目前的托管网关能力依托于托管网格作为控制面，在使用前需先创建一个服务网格CSM-托管网格实例，且一个托管网格下只允许创建一个托管网关。

创建托管网关

按以下步骤在控制台上创建托管网关：

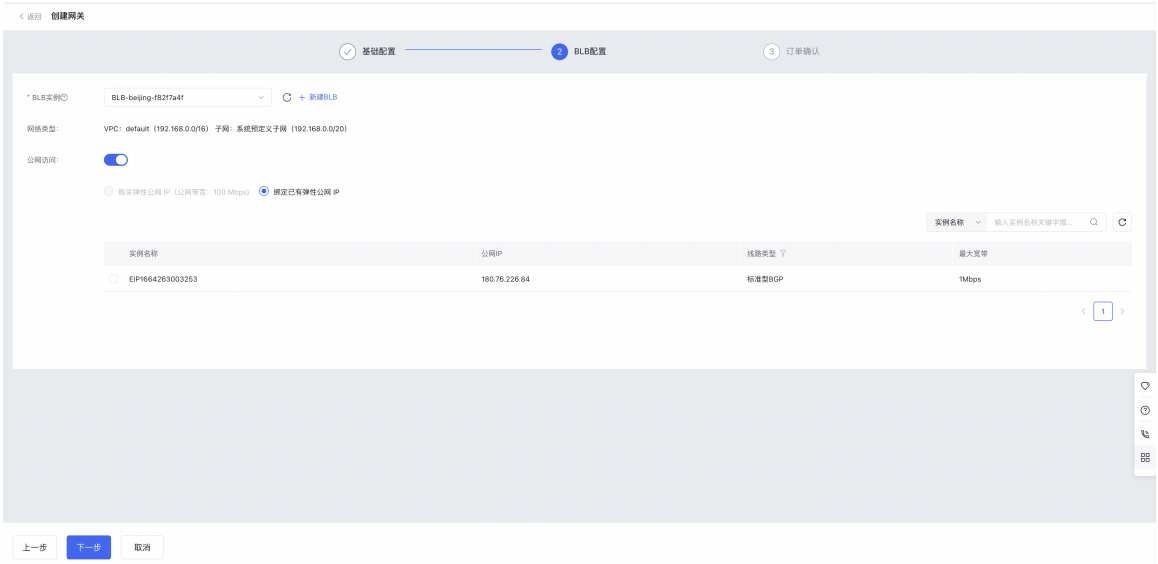
1. 登录百度智能云控制台，选择“产品服务 > 云原生 > 服务网格 CSM”。
2. 在服务网格控制台，对应地域的“网格列表”中找到要操作的托管类型的服务网格实例，点击实例名称可在左侧边栏看到“网关管理”。
3. 点击“网关管理”->“创建网关”：



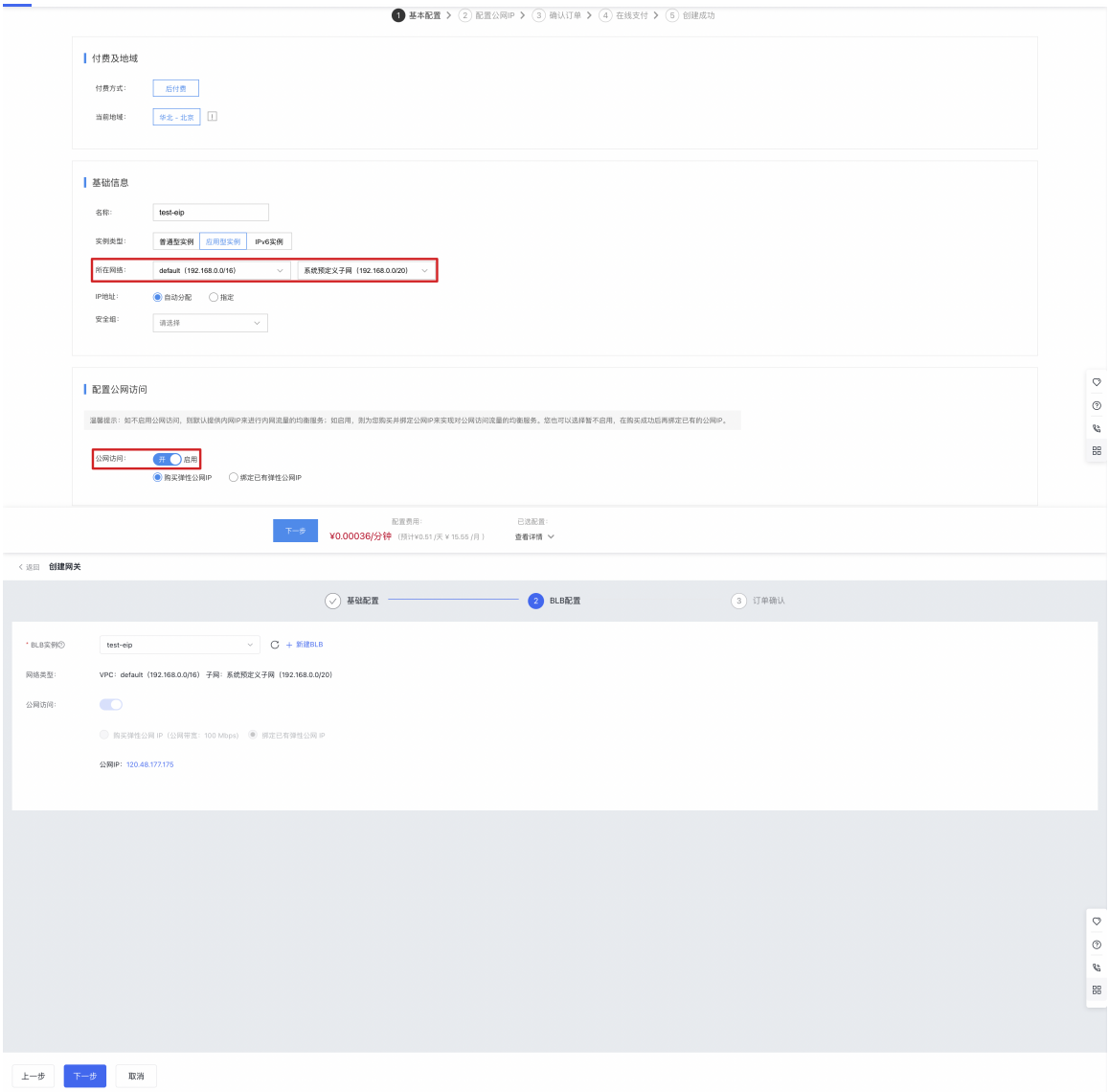
“基础配置”页配置项说明：

配置项	描述	备注
网关名称	托管网关名称（注意命名规则）	
部署方式	托管网关（默认）	
网关类型	入口网关（默认）	
实例规格	托管网关对应Pod的资源配额，目前有“2核4G”和“4核8G”两种规格可以选择	暂不支持用户自定义修改
副本数	托管网关工作负载对应的k8s副本数	
扩缩容HPA	HPA（Horizontal Pod Autoscaling），自动扩缩容工作负载	目前仅支持1-3的扩缩容范围

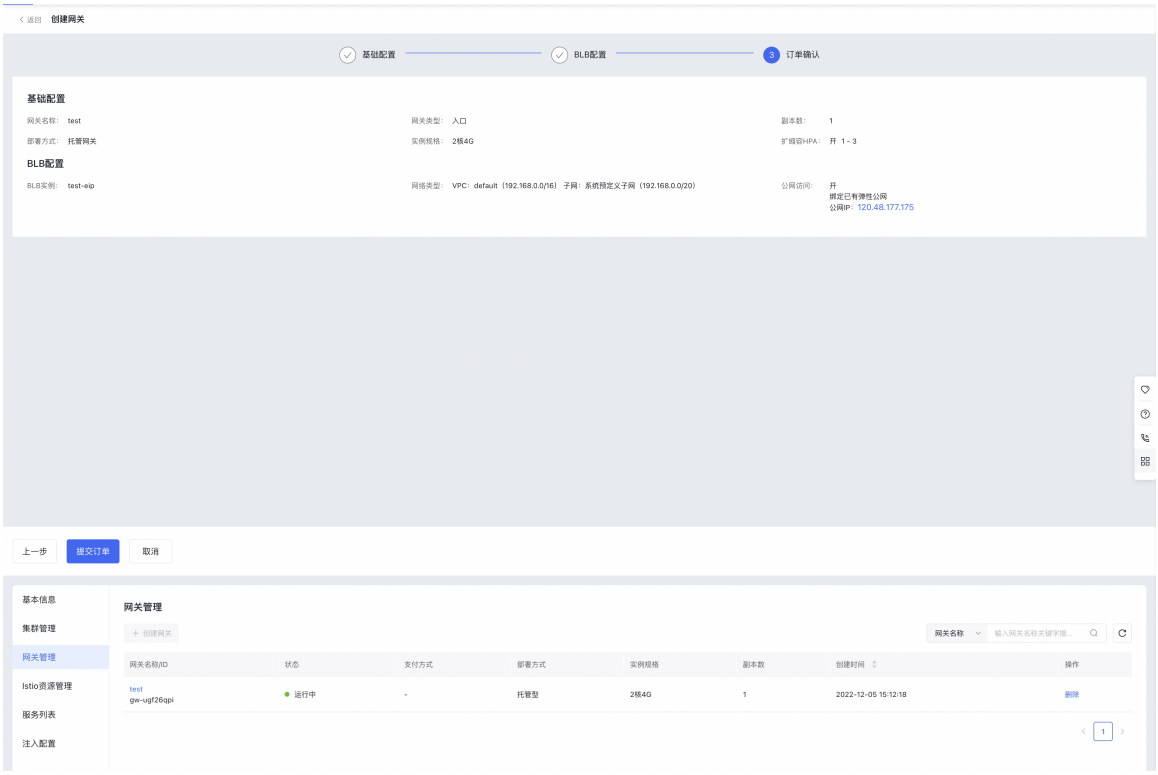
4. 点击下一步，进入BLB配置页面，托管网关要求绑定一个BLB实例作为流量入口进行访问：



列表中会展示当前“网络类型”VPC下的BLB实例，如果当前VPC下不存在BLB实例，则需点击“新建BLB”，跳转至BLB产品页面创建一个BLB实例，注意“所在网络”要与上述“网络类型”保持一致，并且如果在此处打开了“公网访问”开关，则回到托管网关产品“BLB配置”页面“公网访问”会自动打开：



5. 点击“下一步”，确认信息无误后点击“确认订单”开始进行实例创建，在网关管理页面，可以看到“部署中”的托管网关实例，大约几秒钟后会转为“运行中”。

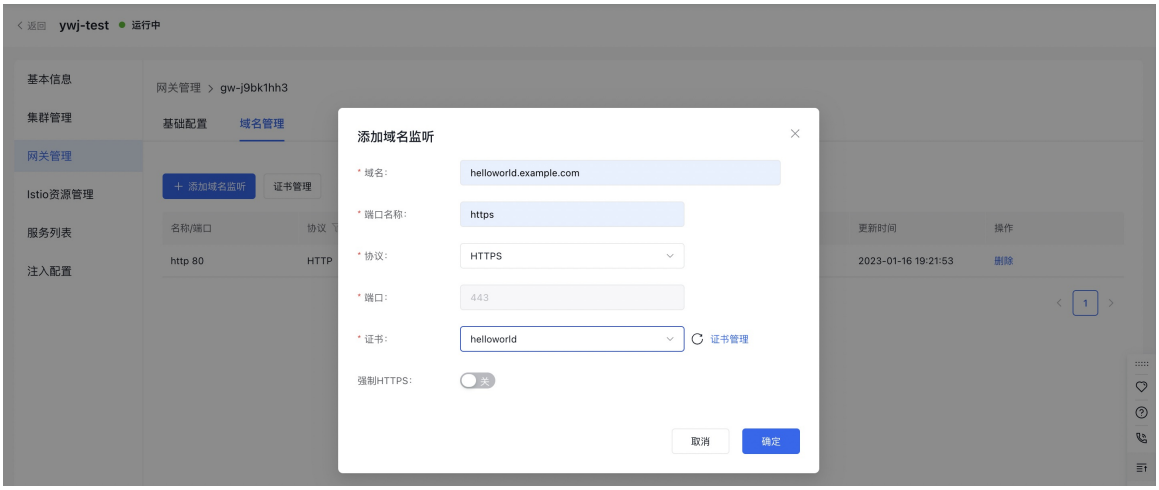


域名管理

添加80端口下的HTTP协议的域名监听

点击“网关管理”，点击网关名称进入“网关详情页”，点击“域名管理”，“添加域名监听”

添加443端口下的HTTPS协议的域名监听



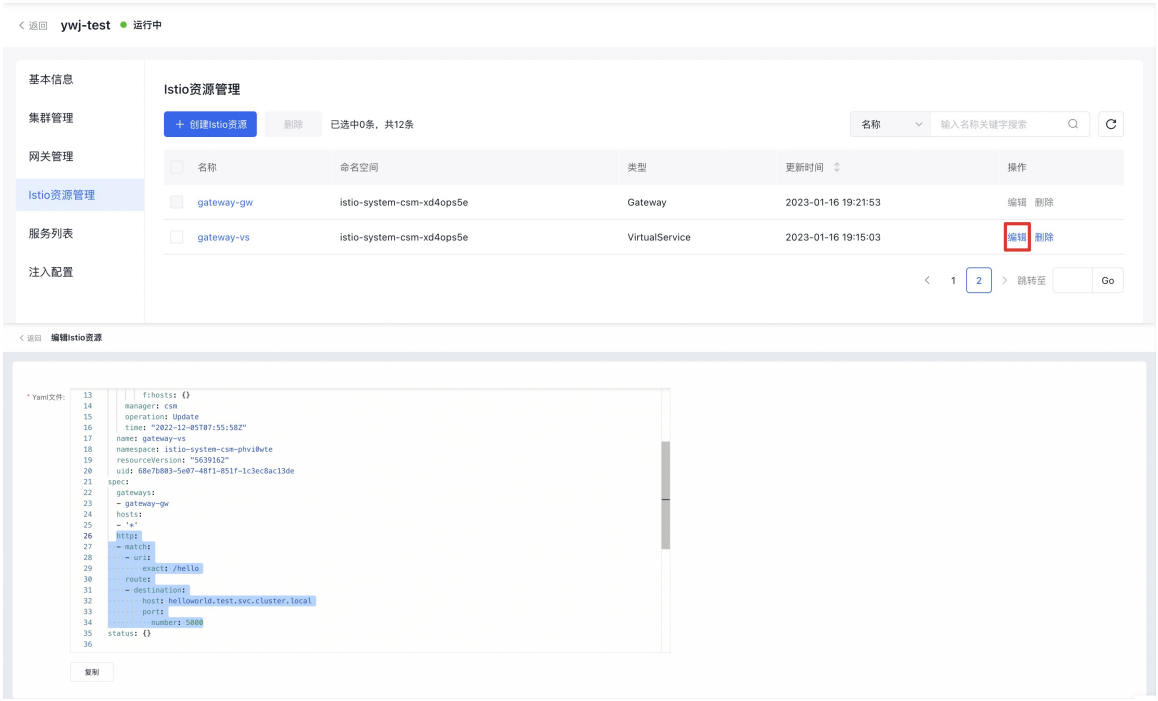
配置项说明：

配置项	描述	备注
域名	网关监听的访问域名，输入完整域名（例如：helloworld.example.com）或泛域名（例如：*.example.com）。在这个域名下，您可以独立管理该域名的协议及证书，路由配置可以通过域名实现相互隔离。	
端口名称	用户自定义的端口名称	
协议	选择HTTP协议或HTTPS协议（与端口强绑定），且HTTPS协议必须关联SSL证书。不同协议支持的端口号如下： • HTTP：支持80端口。 • HTTPS：支持443端口。	
证书	选择HTTPS协议所关联的百度智能云SSL证书服务上的证书。相关内容，请参见SSL证书服务文档。	
强制HTTPS	选中强制HTTPS，表示只生效HTTPS端口，正常HTTP端口将拒绝访问，请求将被重定向为HTTPS。	

流量验证

假设用户在被纳管的集群A中的test命名空间下部署了一个helloworld服务。

用户接下来需要在“istio资源管理”页面编辑“gateway-vs”文件，补充如下配置，保证当前域名的路径下已配置可访问的路由：



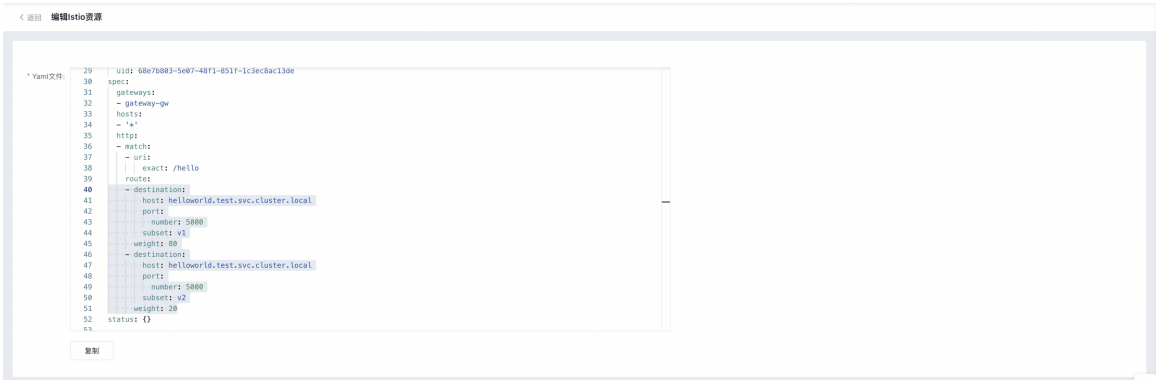
```
 apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
 name: helloworld-gateway
spec:
 gateways:
 - gateway-gw
 hosts:
 - '*'
 http:
 - match:
   - uri:
       exact: /hello
   route:
   - destination:
       host: helloworld.test.svc.cluster.local
       port:
         number: 5000
```

然后在终端通过以下方式访问该托管网关实例对应BLB上绑定的EIP来访问到remote集群的helloworld服务：

- 如果您的域名公网解析已生效，直接可以浏览器访问验证。
- 如果您的域名还未配置DNS解析，则通过以下方式进行验证。
  - 针对HTTP协议，使用`$ curl -v http://helloworld.example.com/hello --resolve "helloworld.example.com:80:{BLB绑定的公网IP}"`进行验证
  - 针对HTTPS协议，使用`$ curl -v -HHost:helloworld.example.com --resolve "helloworld.example.com:443:{BLB绑定的公网IP}" \ --cacert example_certs1/example.com.crt "https://helloworld.example.com:443/hello"`

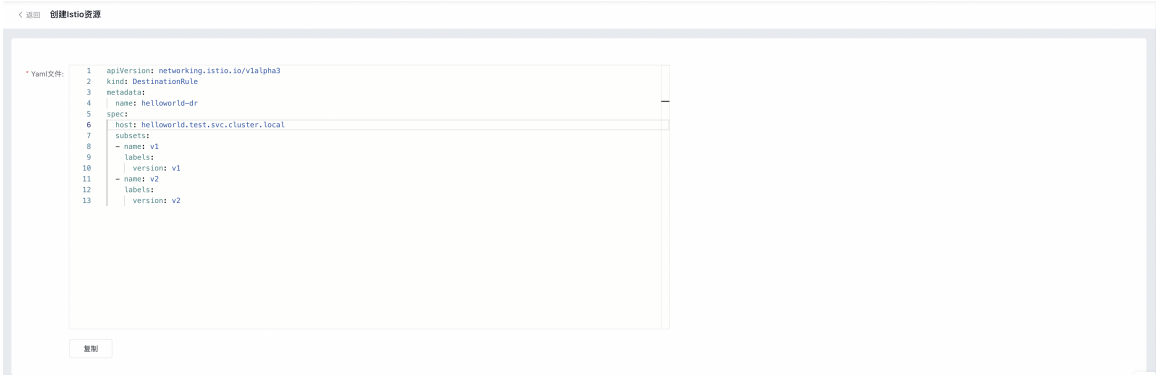
如果需要配置更详细的治理策略可根据用户需要在“istio资源管理”页面下发相应Crd。下面以配置helloworld的v1、v2版本的访问权重作为简单例子说明：

1. 修改gateway-vs文件，补充如下配置：



```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: helloworld-vs
  namespace: istio-system-csm-6gdec53e
spec:
  hosts:
  - helloworld.test-hu.svc.cluster.local
  http:
  - match:
    - uri:
        exact: /hello
    route:
    - destination:
        host: helloworld.test-hu.svc.cluster.local
        port:
          number: 5000
        subset: v1
      weight: 80
    - destination:
        host: helloworld.test-hu.svc.cluster.local
        port:
          number: 5000
        subset: v2
      weight: 20
```

2. 下发helloworld-dr文件：



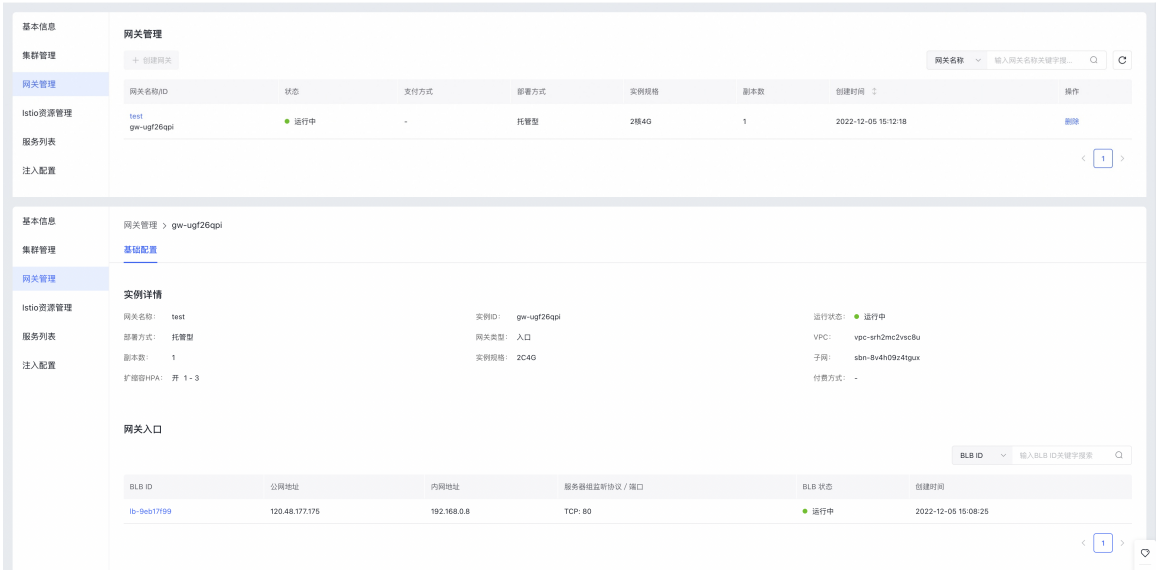

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: helloworld-dr
  namespace: istio-system-csm-6gdec53e
spec:
  host: helloworld.test-hu.svc.cluster.local
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

3. 按照上述方式进行流量比例验证。

查看托管网关详情

按以下步骤在控制台上查看托管网关详情：

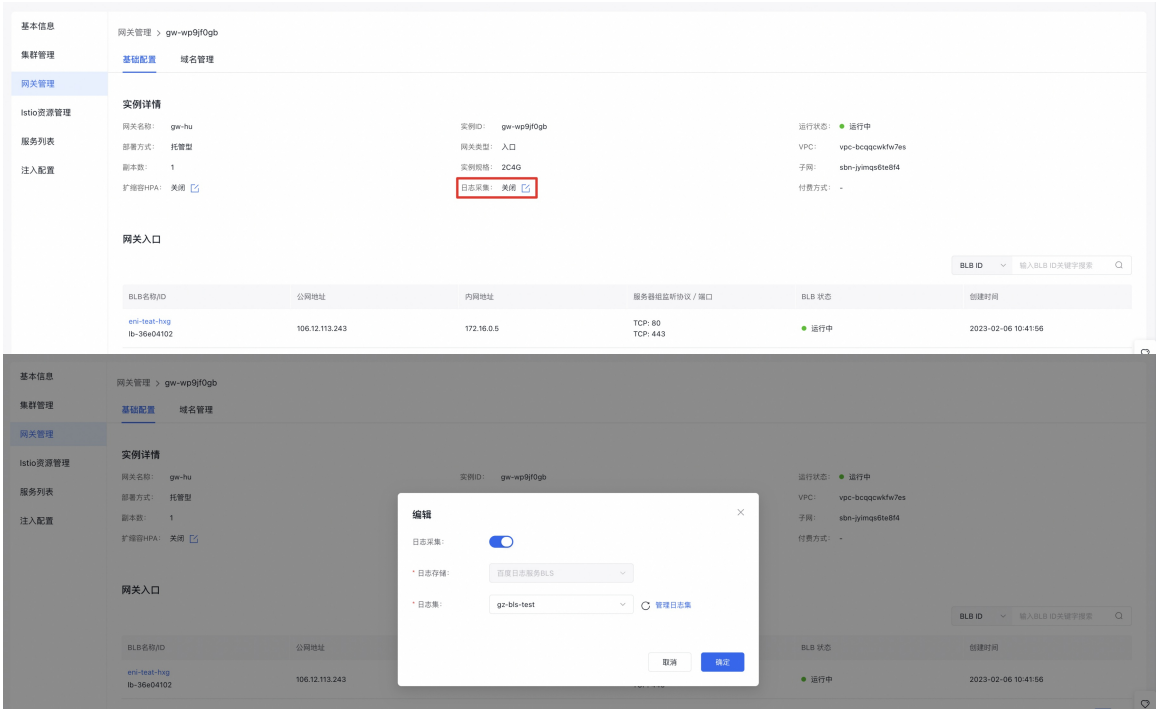
1. 登录百度智能云控制台，选择“产品服务 > 云原生 > 服务网格 CSM”。
2. 在服务网格控制台，对应地域的“网格列表”中找到要操作的托管类型的服务网格实例，点击实例名称可在左侧边栏看到“网关管理”。
3. 点击“网关管理”页面的托管网关实例名称，会跳转到网关详情页面，该页面展示内容如下：



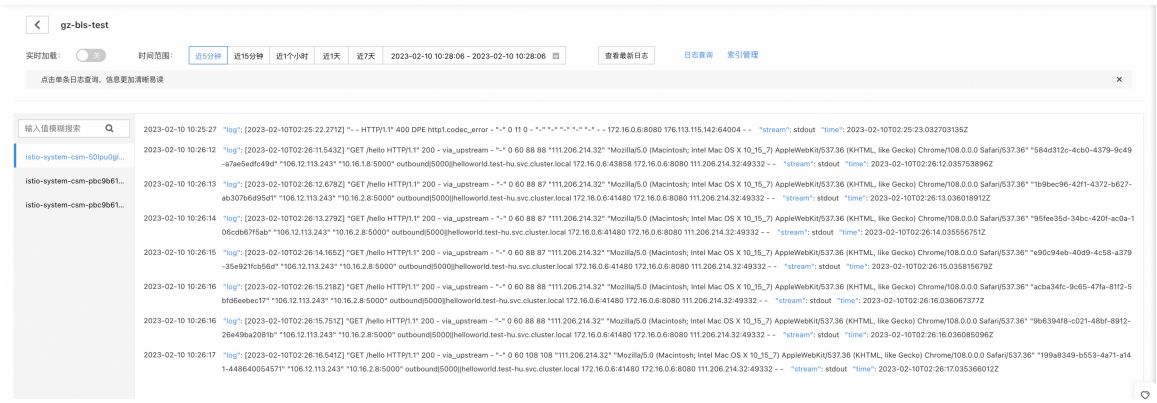
开启日志投递

按以下步骤在控制台上查看托管网关详情：

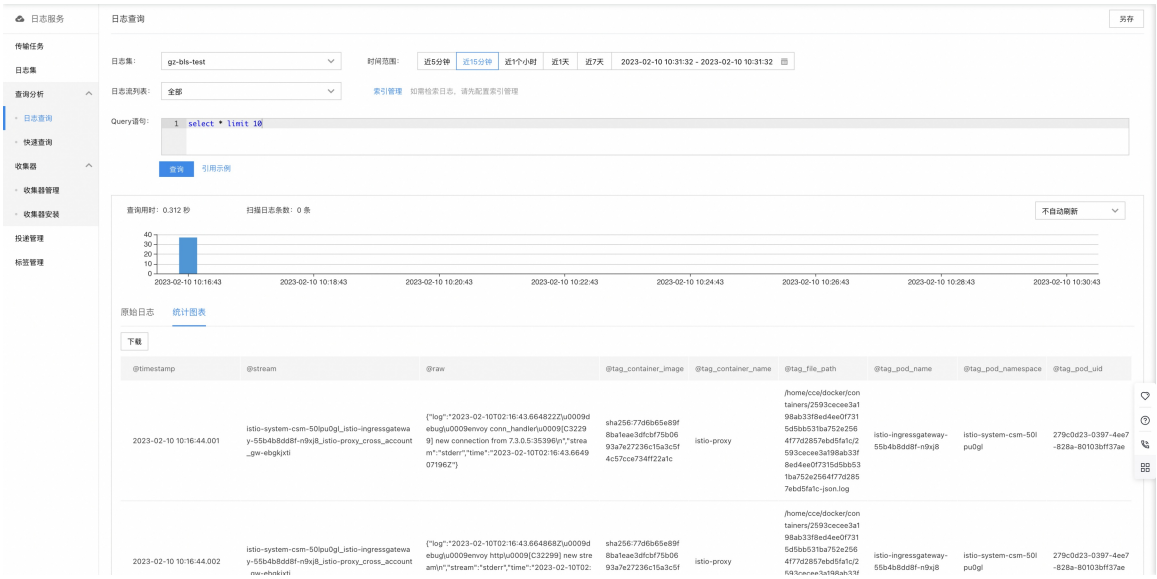
1. 登录百度智能云控制台，选择“产品服务 > 云原生 > 服务网格 CSM”。
2. 在服务网格控制台，对应地域的“网格列表”中找到要操作的托管类型的服务网格实例，点击实例名称可在左侧边栏看到“网关管理”。
3. 点击“网关管理”页面的托管网关实例名称，会跳转到网关详情页面，找到“日志服务”，点击编辑：



配置日志集，点击确认，开启网关日志投递服务，启用后，您可以在百度智能云日志服务BLS上的对应日志集中查看所有网关入口流量请求的日志信息：



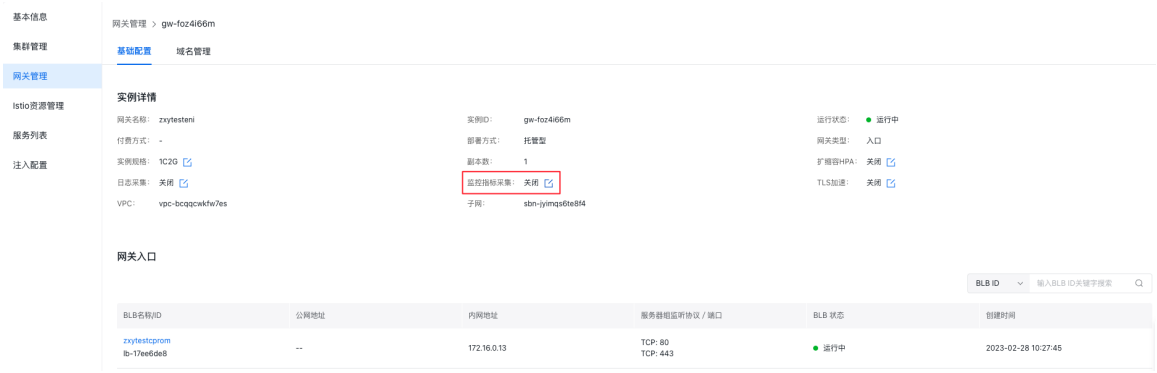
点击右上角的“日志查询“进行更详细的查询：

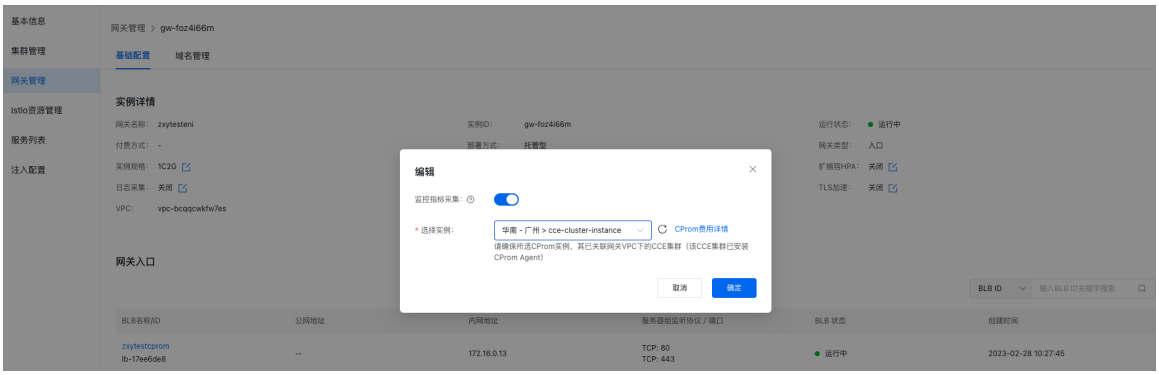


开启Prometheus监控

按以下步骤在控制台上查看托管网关详情：

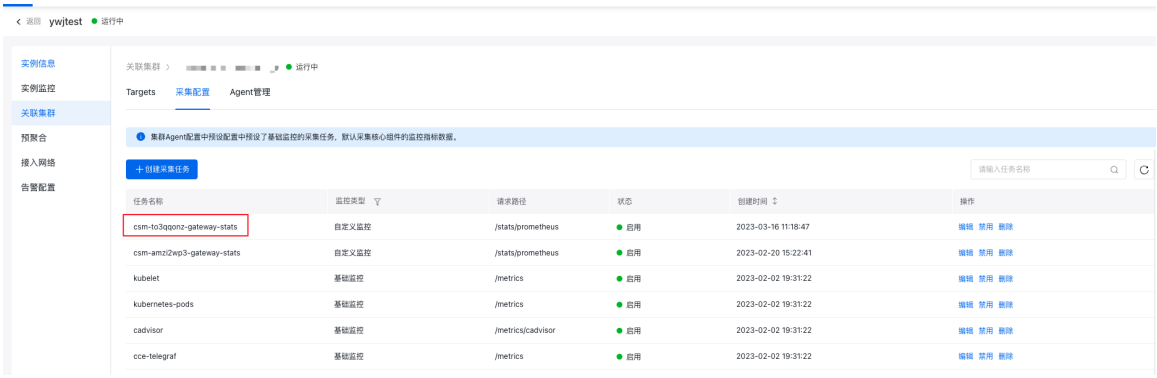
1. 登录百度智能云控制台，选择“产品服务 > 云原生 > 服务网格 CSM”。
2. 在服务网格控制台，对应地域的“网格列表”中找到要操作的托管类型的服务网格实例，点击实例名称可在左侧边栏看到“网关管理”。
3. 点击“网关管理”页面的托管网关实例名称，会跳转到网关详情页面，找到“监控指标采集”，点击编辑：





选择符合标准的Prometheus实例，点击确认，开启网关监控信息采集服务。(选择Prometheus实例的标准：已关联网关VPC下的CCE集群并且该CCE集群已安装Prometheus Agent)

启用后，您可以在百度智能云Prometheus监控服务上对应Prometheus实例的采集配置中，查看到网关的配置采集项：



点击"Grafana服务"查看网关的监控数据：

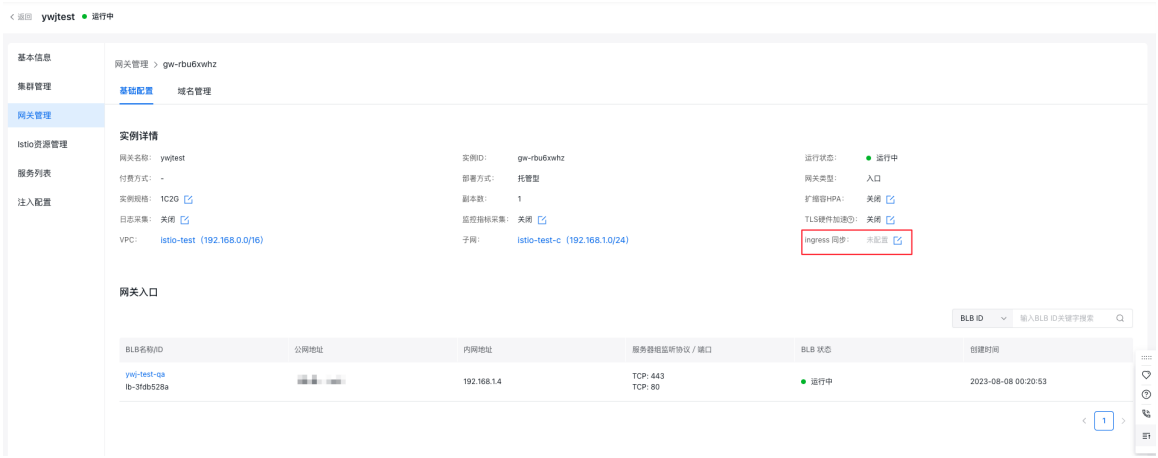
选择CSM目录下Istio Gateway Dashboard模板。数据源选择对应的Prometheus实例

Ingress同步

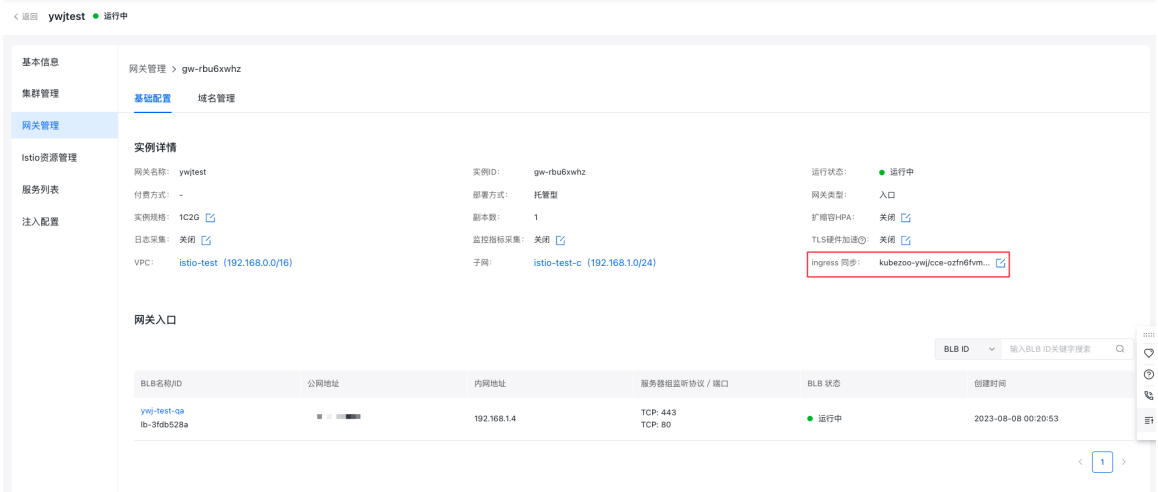
除了标准的使用方式，如通过Gateway、VirtualService等CRD配置CSM网关外，CSM产品还为CCE用户提供了“Ingress同步”功能，使他们能够便捷地将CCE集群中的现有ingress接入CSM网关，并最大化利用CSM网关的优势。相较于传统的K8S Ingress，CSM网关在稳定性和性能上都实现了显著的提升。

1、开启ingress同步

点击“网关管理”页面的托管网关实例名称，会跳转到网关详情页面，找到“ingress同步”，点击编辑：



点击确定以后，界面如下图所示



用户CCE集群部署测试服务

```
  apiVersion: v1
kind: Service
metadata:
  name: helloworld
  labels:
    app: helloworld
    service: helloworld
spec:
```

```
  ports:
  - port: 5000
    name: http
  selector:
    app: helloworld
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld-v1
  labels:
    app: helloworld
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: helloworld
      version: v1
  template:
    metadata:
      labels:
        app: helloworld
        version: v1
    spec:
      containers:
      - name: helloworld
        image: docker.io/istio/examples-helloworld-v1
        resources:
          requests:
            cpu: "100m"
        imagePullPolicy: IfNotPresent #Always
        ports:
        - containerPort: 5000
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld-v2
  labels:
    app: helloworld
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: helloworld
      version: v2
  template:
    metadata:
      labels:
        app: helloworld
        version: v2
    spec:
      containers:
      - name: helloworld
        image: docker.io/istio/examples-helloworld-v2
        resources:
          requests:
            cpu: "100m"
        imagePullPolicy: IfNotPresent #Always
        ports:
```

```
      - containerPort: 5000
```

用户CCE集群部署ingress

```
 apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: helloworld
spec:
 rules:
 - host: helloworld.com
   http:
     paths:
     - path: /hello
       pathType: Exact
       backend:
         service:
           name: helloworld
           port:
             number: 5000
```

apply ingress.yaml之后稍等1分钟左右，执行下面命令，其中"公网IP"来源于网关入口中的公网地址。

```
curl -v http://helloworld.com/hello --resolve "helloworld.com:80:{公网IP}"
```

预期结果如下

```
➜  csm-gateway curl -v http://helloworld.com/hello --resolve "helloworld.com:80:106.[illegible].84"
* Added helloworld.com:80:106.[illegible].84 to DNS cache
* Hostname helloworld.com was found in DNS cache
*   Trying 106.[illegible].84:80...
* Connected to helloworld.com (106.[illegible].84) port 80 (#0)
> GET /hello HTTP/1.1
> Host: helloworld.com
> User-Agent: curl/7.77.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< server: istio-envoy
< date: Mon, 07 Aug 2023 10:20:35 GMT
< content-type: text/html; charset=utf-8
< content-length: 59
< x-envoy-upstream-service-time: 133
<
Hello version: v2, instance: helloworld-v2-54df5f84b-jsb8t
* Connection #0 to host helloworld.com left intact
```

2、关闭ingress同步

稍等1分钟左右，再次执行命令

```
curl -v http://helloworld.com/hello --resolve "helloworld.com:80:{公网IP}"
```

预期访问失败如下

```
→  csm-gateway curl -v http://helloworld.com/hello --resolve "helloworld.com:80:106.[illegible].84"
* Added helloworld.com:80:106.[illegible].84 to DNS cache
* Hostname helloworld.com was found in DNS cache
*   Trying 106.[illegible]34:80...
* Connected to helloworld.com (106.[illegible].84) port 80 (#0)
> GET /hello HTTP/1.1
> Host: helloworld.com
> User-Agent: curl/7.77.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 404 Not Found
< date: Mon, 07 Aug 2023 10:32:49 GMT
< server: istio-envoy
< content-length: 0
<
* Connection #0 to host helloworld.com left intact
```

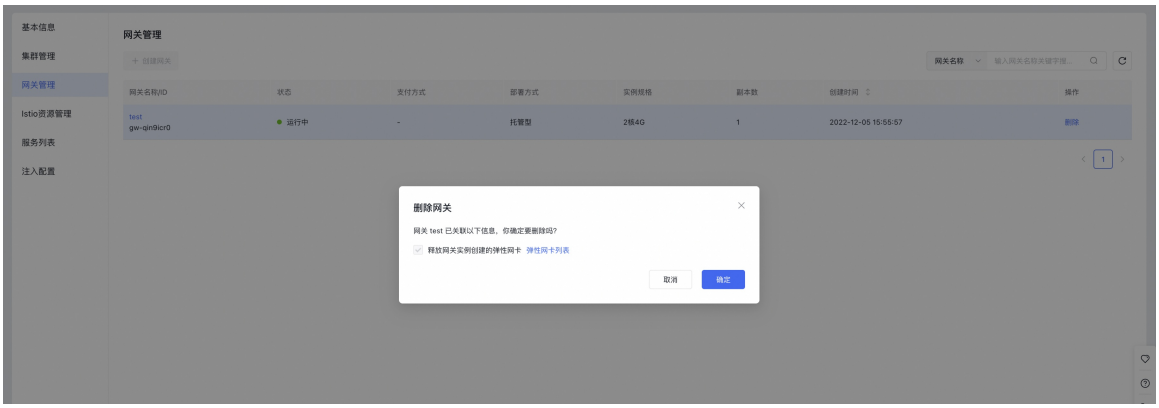
3、适配大部分nginx标签，示例： 配置重试标签注解

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: helloworld
  namespace: test-ywj
  annotations:
    nginx.ingress.kubernetes.io/proxy-next-upstream: http_502
    nginx.ingress.kubernetes.io/proxy-next-upstream-timeout: "5"
    nginx.ingress.kubernetes.io/proxy-next-upstream-tries: "10"
spec:
  rules:
  - host: helloworld.exa12.com
    http:
      paths:
      - path: /hello1
        pathType: Exact
        backend:
          service:
            name: helloworld
            port:
              number: 5000
```

删除托管网关

按以下步骤在控制台上删除托管网关：

1. 登录百度智能云控制台，选择“产品服务 > 云原生 > 服务网格 CSM”。
2. 在服务网格控制台，对应地域的“网格列表”中找到要操作的托管类型的服务网格实例，点击实例名称可在左侧边栏看到“网关管理”。
3. 点击“网关管理”页面的托管网关实例后面的“删除”按钮，即可删除托管网关，同时会默认解绑BLB和与之绑定的EIP：

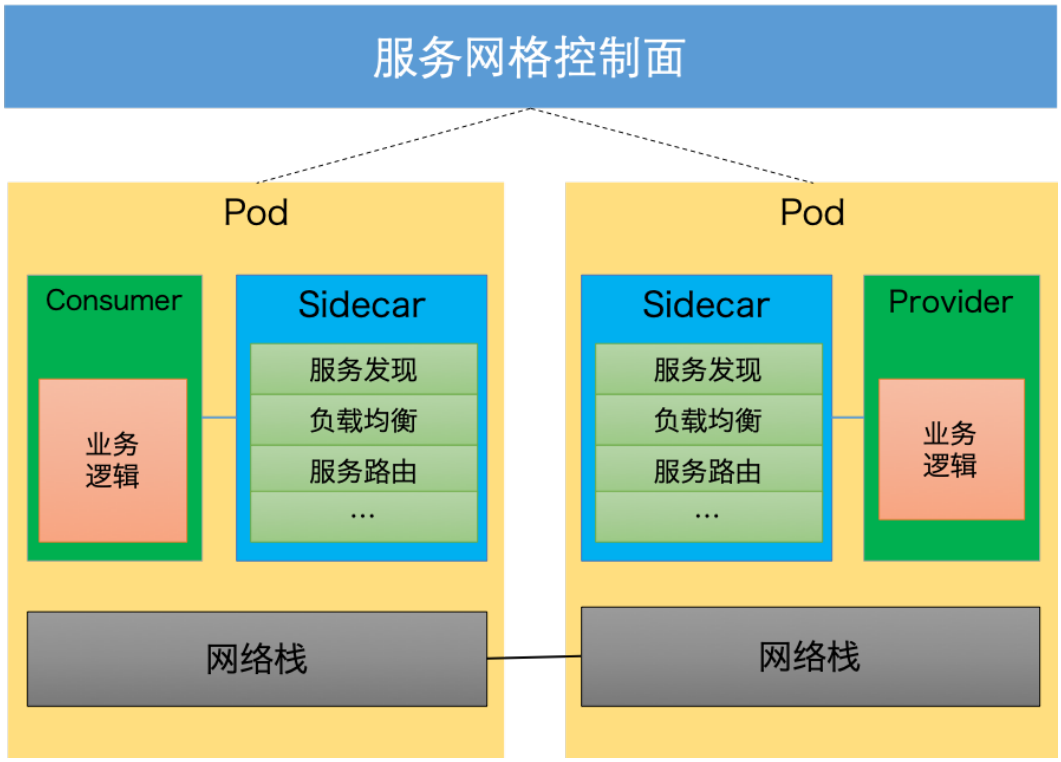


4. 在删除托管网关实例之后，建议用户将之前创建下发的托管网关相关istio crd一并删除。

注入配置

概述

服务网格通过容器中的 Sidecar 劫持服务间的流量并执行微服务的治理策略，实现对底层网络功能的抽象化管理，例如负载均衡、服务发现、流量控制等。本文介绍如何通过服务网格 CSM 控制台，对 CCE 集群中的工作负载进行 Sidecar 的自动注入配置。



前提条件

已添加 CCE 集群到 CSM 实例中，如何添加集群可参见 集群管理。

Sidecar 资源配置

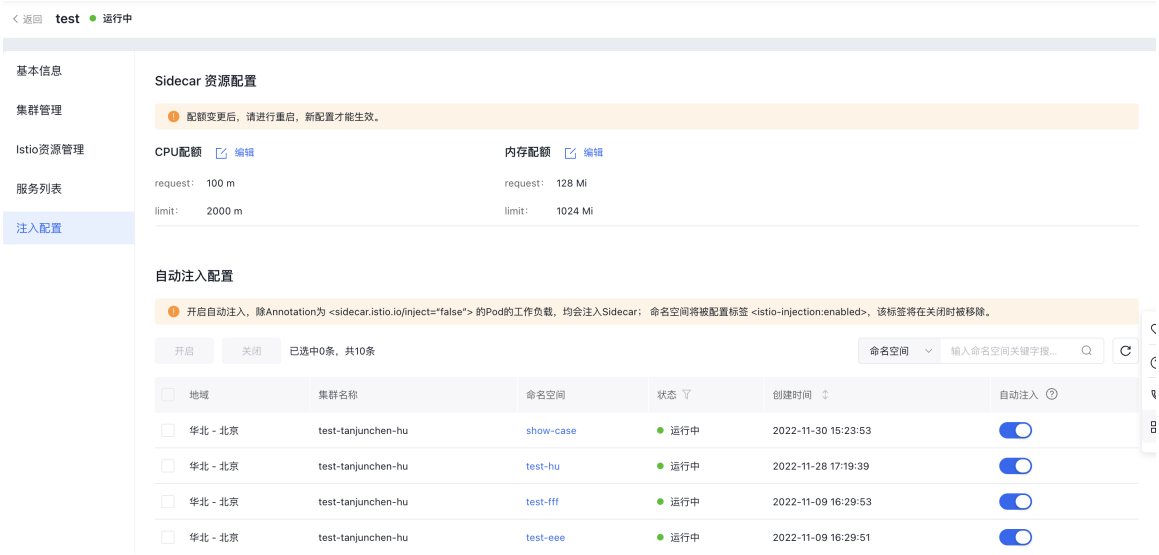
1. 登录 百度智能云控制台，选择**产品服务 > 云计算>容器 > 服务网格 CSM**。
2. 在服务网格控制台，左侧导航栏选择**网格列表**，在网格列表页面点击期望操作的**网格名称/ID**，在左侧导航栏选择**注入配置**。
3. 注入配置页面包含两个模块：Sidecar 资源配置和自动注入配置。Sidecar 资源配置展示了当前的 Sidecar CPU、内存资源配置信息。通常不需要变更这一配置，可基于监控指标和测试进行调整，Istio 官方社区性能测试结果 供您参考。

Sidecar 自动注入配置

用户可以通过 Sidecar 的自动注入功能，在业务部署包完全不变更的情况下，实现部署时注入 Sidecar 代理。服务网格 CSM 控制台提供了命名空间级别的注入配置入口，用户也可以在 Pod 级别进行标记，实现工作负载级别的自动注入。

为命名空间开启自动注入

1. 在注入配置页面中的**自动注入配置**模块，可以查看服务网格 CSM 实例管理的所有集群中的命名空间。可以在右上角搜索框输入命名空间/集群名称查找指定集群下的目标命名空间。
2. 可以在自动注入列开启**自动注入开关**，或者勾选想要开启自动注入的命名空间，点击上方**开启**按钮，即可为指定命名空间开启命名空间级别的 Sidecar 自动注入。



3. 开启 Sidecar 自动注入后，会为命名空间配置标签 `istio-injection:enabled`，关闭自动注入则会移除 istio-injection 标签，具体的效果如下：

操作	效果
开启自动注入后的命名空间	**除** Annotation 为 **sidecar.istio.io/inject="false"** 的 Pod，命名空间内的其余 Pod 均会被自动注入 Sidecar。
未开启自动注入的命名空间	**仅** Annotation 为 **sidecar.istio.io/inject="true"** 的 Pod 会被自动注入 Sidecar。

> **说明：**改变自动注入策略时，已在运行中的工作负载，不会立即注入或移出 Sidecar 代理，在重启后将执行变更后的自动注入策略，**服务网格 CSM 不会触发工作负载的重启**。

4. 在开启自动注入后的命名空间中创建应用，验证应用的 Pod 是否注入 Sidecar 代理。
   - 登录 CCE 容器引擎，在集群列表页面单击目标集群名称进入集群详情页，左侧导航栏选择**工作负载 > 容器组**。
   - 在容器组列表页中选择开启了自动注入的命名空间，点击 **Pod 名称**进入容器组详情页。
   - 在容器详情页的**容器列表**中可以观察到除自身 Pod 外，新增了一个名为 istio-proxy 的容器，表明自动注入 Sidecar 代理成功。

以 Pod 为单位，单独为 Pod 设置自动注入配置

若您不希望以命名空间为维度批量设置自动注入配置，也可以单独为 Pod 设置自动注入配置。通过为 Pod 设置注解 `sidecar.istio.io/inject="true"` 即可以 Pod 为单位开启自动注入。

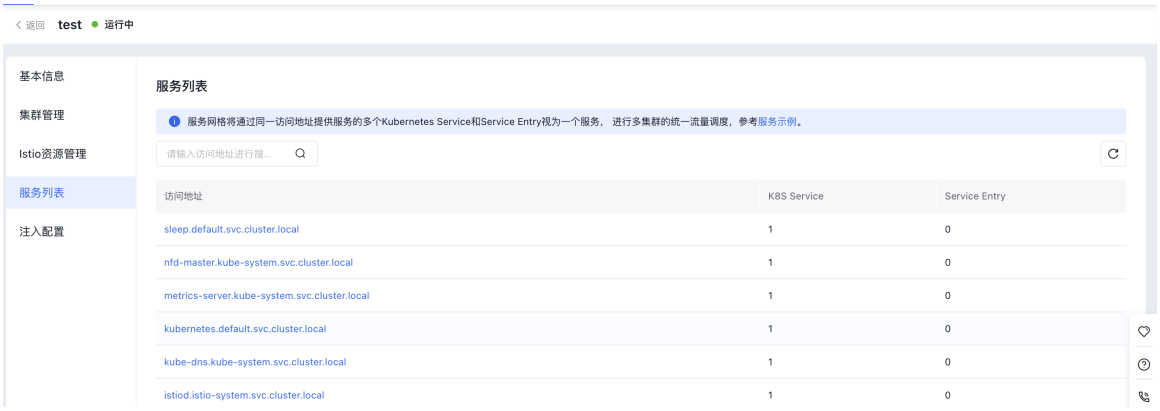
服务列表

概述

服务网格CSM中的服务可以跨Kubernetes集群，服务发现机制支持自动发现Kubernetes Service和通过提交Service Entry手动注册到服务网格。为便于用户了解一个服务的构成，并进行服务级别的维护操作，服务列表对服务网格实例中的服务进行了展示。

服务示例

- 当服务网格同时管理多个Kubernetes集群，且用户在不同集群的相同命名空间下，创建了相同名称的服务，基于Kubernetes生成服务访问地址，会生成相同的“访问地址”，比如：service-name.namespace-name.svc.cluster.local。当有流量通过服务网格的代理发向这个“访问地址”，服务网格会同时解析到多个不同集群中的Pod地址，并将流量在多个Kubernetes集群的Pod中负载均衡。
- 服务网格同时支持用户通过Service Entry方式，将非Kubernetes Service注册至服务网格，如果注册的hosts也为service-name.namespace-name.svc.cluster.local，当有流量通过服务网格数据面发往service-name.namespace-name.svc.cluster.local时，服务网格的数据面通过这个“访问地址”，会同时解析到Kubernetes Service对应的Pod地址和Service Entry中hosts对应的工作负载地址，并在这些工作负载中负载均衡。
- 这个“访问地址”和对应的工作负载，就构成了一个服务网格中的服务。
- 服务网格列表展示:



查看服务列表

1. 登录百度智能云控制台，选择“产品服务>云原生>服务网格 CSM”。
2. 在服务网格控制台中，找到对应的服务网格实例，点击服务网格实例名称可在左侧边栏看到“服务列表”。
3. 当多个Kubernetes Service或Service Entry注册的是同一个提供服务的“访问地址”时，对于服务网格CSM认为是同一个服务，展示为一行。服务网格列表展示了注册至服务网格的服务来源，可能通过多个Kubernetes Service或通过Service Entry注册至服务网格。
4. 点击服务的"访问地址"，即可查看服务所对应的所有Kubernetes Service和Service Entry。

Istio资源管理

概述

服务网格CSM通过Istio CRD资源（如Virtual Service、Destination Rule）描述微服务治理策略，“Istio 资源”页面对通过控制台、原生API等不同方式提交的资源集中展示，用户可在“Istio 资源”页面按需进行增删改查等操作。

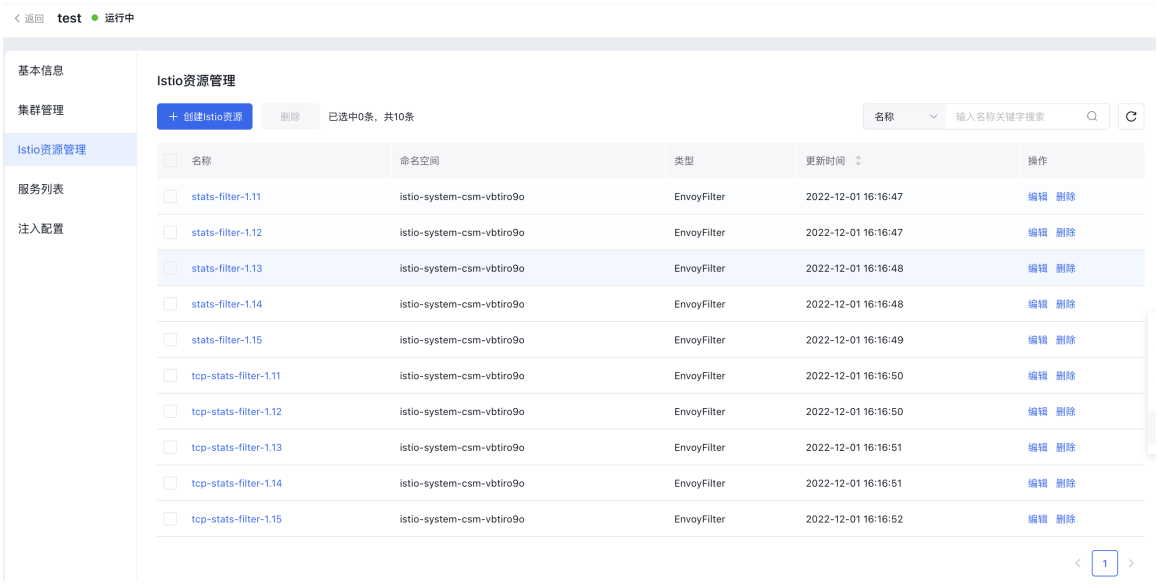
注意事项

- 对于注入了服务网格代理的工作负载，如没有提交任何Istio CRD，流量仍然会被代理劫持，以轮询的负载均衡方式发往目的服务的多个实例。

创建Istio资源

按以下步骤在控制台上创建与更改Istio CRD资源：

1. 登录百度智能云控制台，选择“产品服务>云计算>容器 > 服务网格 CSM”。
2. 在服务网格控制台，“网格列表”页面选择地域后，找到相要操作的服务网格实例，点击服务网格实例名称可在左侧边栏看到Istio资源管理。
3. 通过“创建 Istio 资源”按钮，即可进入Istio资源提交页面，目前暂只支持YAML提交。



4. Istio资源创建后，会立即下发至服务网格的数据面生效相关规则。

> Istio 资源的命名空间，是指Istio资源的存储位置，生效的范围是在Spec字段中描述。

变更Istio资源

1. 在“Istio 资源管理”页，通过点击Istio资源列表中具体的Istio资源名称或操作列的“编辑”按钮，即可进行编辑操作。
2. 在“Istio 资源管理”页，通过Istio资源列表操作列的“删除”按钮，可以在控制台上完成Istio资源的删除；如需多条资源进行批量删除，可以选中后通过Istio资源列表上方的“删除”进行批量操作。

参考示例

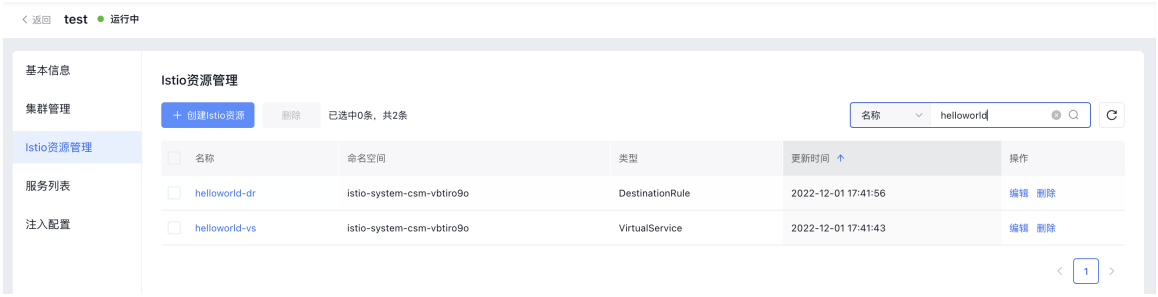
我们根据上述Sidecar自动注入、集群管理等手册操作完成后，我们给sleep/helloworld应用配置Istio CRD策略，具体如下所示：
VirtualService:

```
 apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: helloworld-vs
spec:
 hosts:
  - helloworld.test.svc.cluster.local
 exportTo:
  - test
 http:
 - route:
  - destination:
      host: helloworld.test.svc.cluster.local
      subset: v1
    weight: 100
  - destination:
      host: helloworld.test.svc.cluster.local
      subset: v2
    weight: 0
```

DestinationRule:

```
 apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: helloworld-dr
spec:
 exportTo:
  - test
 host: helloworld.test.svc.cluster.local
 subsets:
 - name: v1
   labels:
     version: v1
 - name: v2
   labels:
     version: v2
```

通过Istio资源管理提交上述 Yaml 文件后，如下所示：



其他参考案例

Istio 社区示例

多用户访问控制

概述

多用户访问控制(Identity and Access Management, IAM)，主要用于百度智能云身份管理和访问控制，解决云账户的集中授权与管理、资源分享与多用户协同工作等问题。多用户访问控制适用于企业内的不同职能角色，你可以对不同员工赋予产品的不同权限，以共享你账户内的资源，完成他们的工作。当你的企业存在需要多用户协同工作、分享资源时，推荐你使用多用户访问

控制。

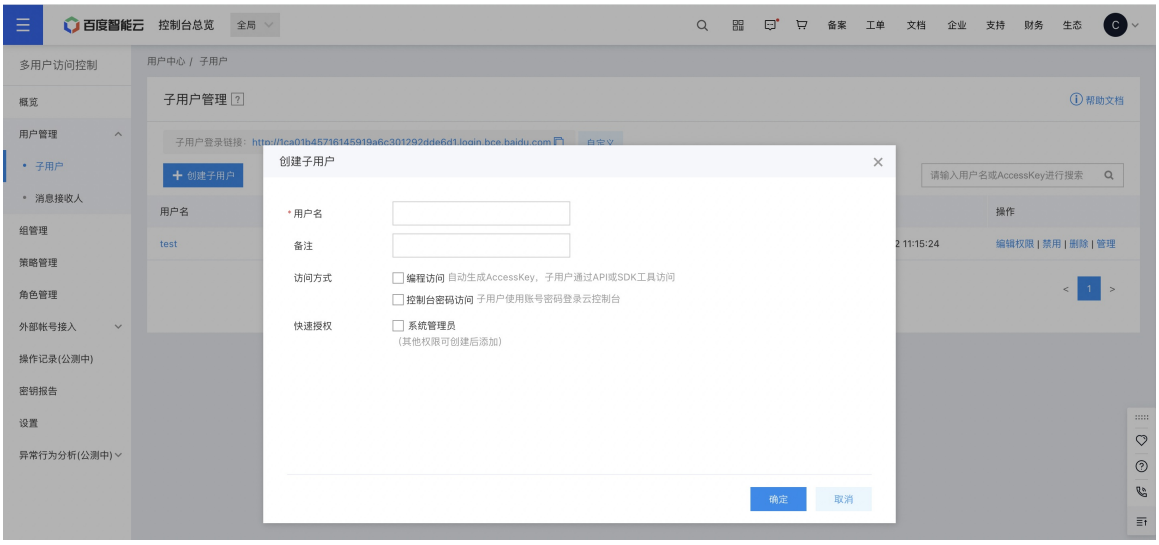
以下是多用户访问控制适用的典型场景：

- 中大型企业客户：对公司内多个部门的不同员工进行集中资源和权限管理；
- 独立软件服务商(ISV)或SaaS平台商：对代理客户进行集中的资源和权限管理；
- 中小开发者或小企业：添加项目成员或协作者，进行资源和权限管理。

创建用户

1. 主账号用户登录后在控制台右上角，选择“多用户访问控制”，进入“多用户访问控制”页面。

2. 在左侧导航栏点击“用户管理”，在“子用户”页，点击“创建子用户”。



3. 在弹出的“创建子用户”对话框中，完成填写“用户名”等信息，“确定”后返回“子用户管理”列表区可以查看到刚刚创建的子用户。

配置策略

服务网格CSM支持系统策略和用户自定义策略两种，分别实现CSM的产品级权限和实例级权限控制。

系统策略：百度智能云系统为管理资源而预定义的权限集，这类策略可直接为子用户授权，用户只能使用而不能修改。

自定义策略：由用户自己创建，更细化的管理资源的权限集，可以针对单个实例配置权限，相比系统策略更加灵活的满足账户对不同用户的差异化权限管理。

系统策略

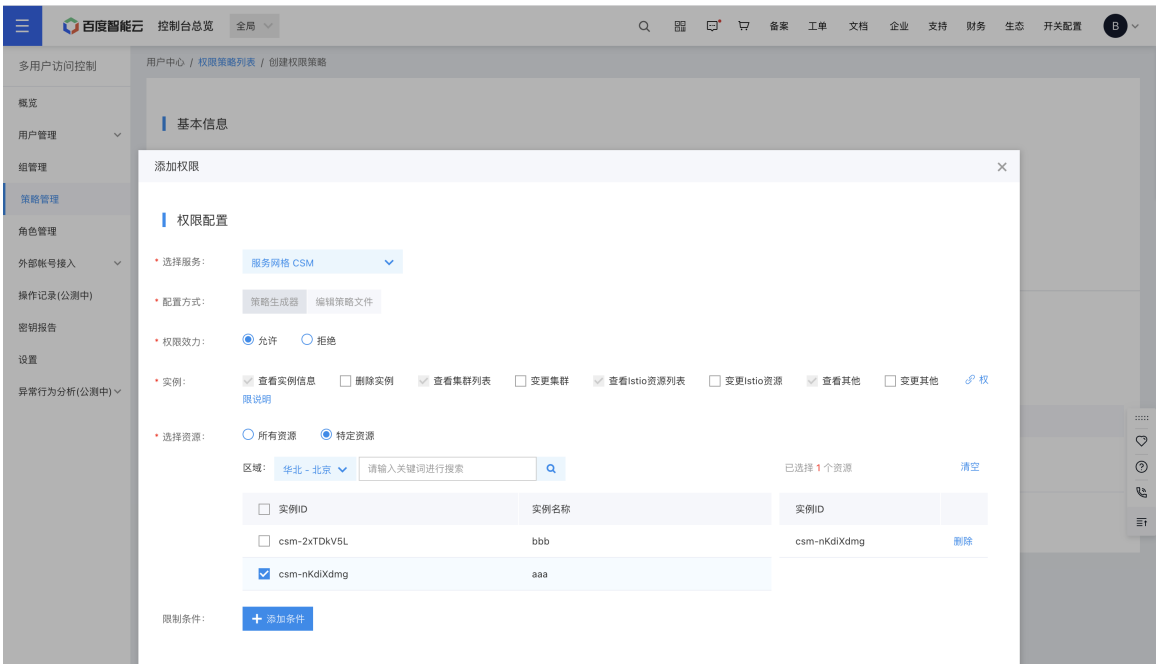
本节说明接入IAM的产品系统策略，当前服务网格CSM系统策略包含只读权限、运维权限和管理权限。

策略名称	权限说明	权限范围
CSMFullControlAccessPolicy	完全控制管理服务网格(CSM)的权限	• 查看概览 • 查看网格列表 • 查看、创建、删除、升级实例 • 查看集群列表 • 添加、移出集群 • 查看、创建、编辑、删除Istio资源 • 查看服务列表、服务详情 • 查看、编辑自动注入配置 • 修改Sidecar资源配额等全部操作
CSMOperateAccessPolicy	运维操作服务网格(CSM)的权限	• 查看概览 • 查看网格列表 • 查看、升级实例 • 查看集群列表 • 查看、创建、编辑、删除Istio资源 • 查看服务列表、服务详情 • 查看、编辑自动注入配置 • 修改Sidecar资源配额等变更操作
CSMReadAccessPolicy	只读访问服务网格(CSM)的权限	• 查看概览 • 查看网格列表 • 查看实例 • 查看集群列表 • 查看Istio资源 • 查看服务列表、服务详情 • 查看自动注入配置 • 查看Sidecar资源配额等只读权限

自定义策略

本节用于说明接入IAM产品的自定义策略内的权限操作类型以及权限的范围。自定义策略是从实例维度进行授权，与系统策略不同，只对选定的实例生效，无法创建新的实例。

在“多用户访问控制”页面通过左侧导航栏进入“策略管理”，然后点击“创建策略”，策略生成方式默认为策略生成器，不需要修改，选择服务“服务网格CSM”进入如下页面，进行具体实例的权限设置。



权限说明	权限范围
只读权限	• 查看概览 • 查看网格列表 • 查看实例信息 • 查看集群列表 • 查看Istio资源 • 指定实例的其它查看权限
变更权限	可按需按需选择以下权限： • 删除实例 • 添加、移出集群 • 变更Istio资源 • 实例的其它变更操作

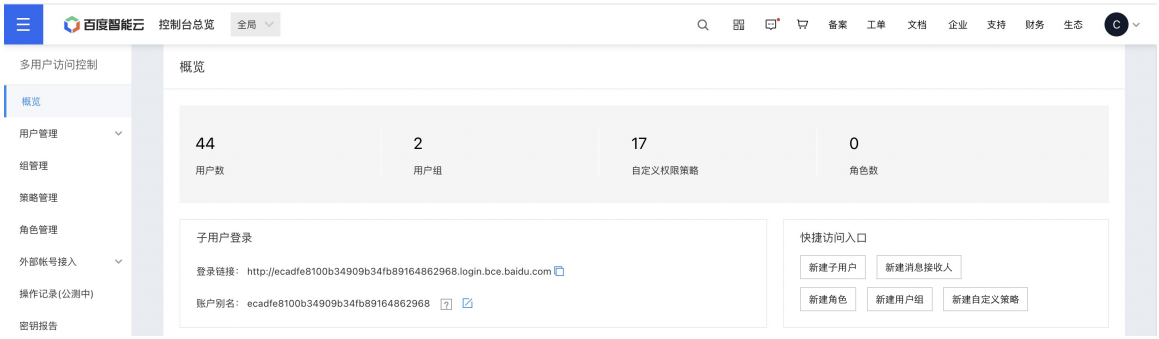
用户授权

在“用户管理”下的“子用户”页面中，对应子用户的“操作”列选择“添加权限”，为用户选择系统权限或自定义策略并进行授权。

> 说明：如果在不修改已有策略规则的情况下修改某子用户的权限，只能通过删除已有的策略并添加新的策略来实现，不能取消勾选已经添加过的策略权限。

子用户登录

主账号完成对子用户的授权后，可以将链接发送给子用户；子用户可以通过IAM用户登录链接登录主账号的管理控制台，根据被授权的策略对主账户资源进行操作和查看。



其他详细操作参考：多用户访问控制。

多协议管理

概述

本节介绍服务网格CSM支持Dubbo、bRPC等私有协议的相关操作。

使用前提

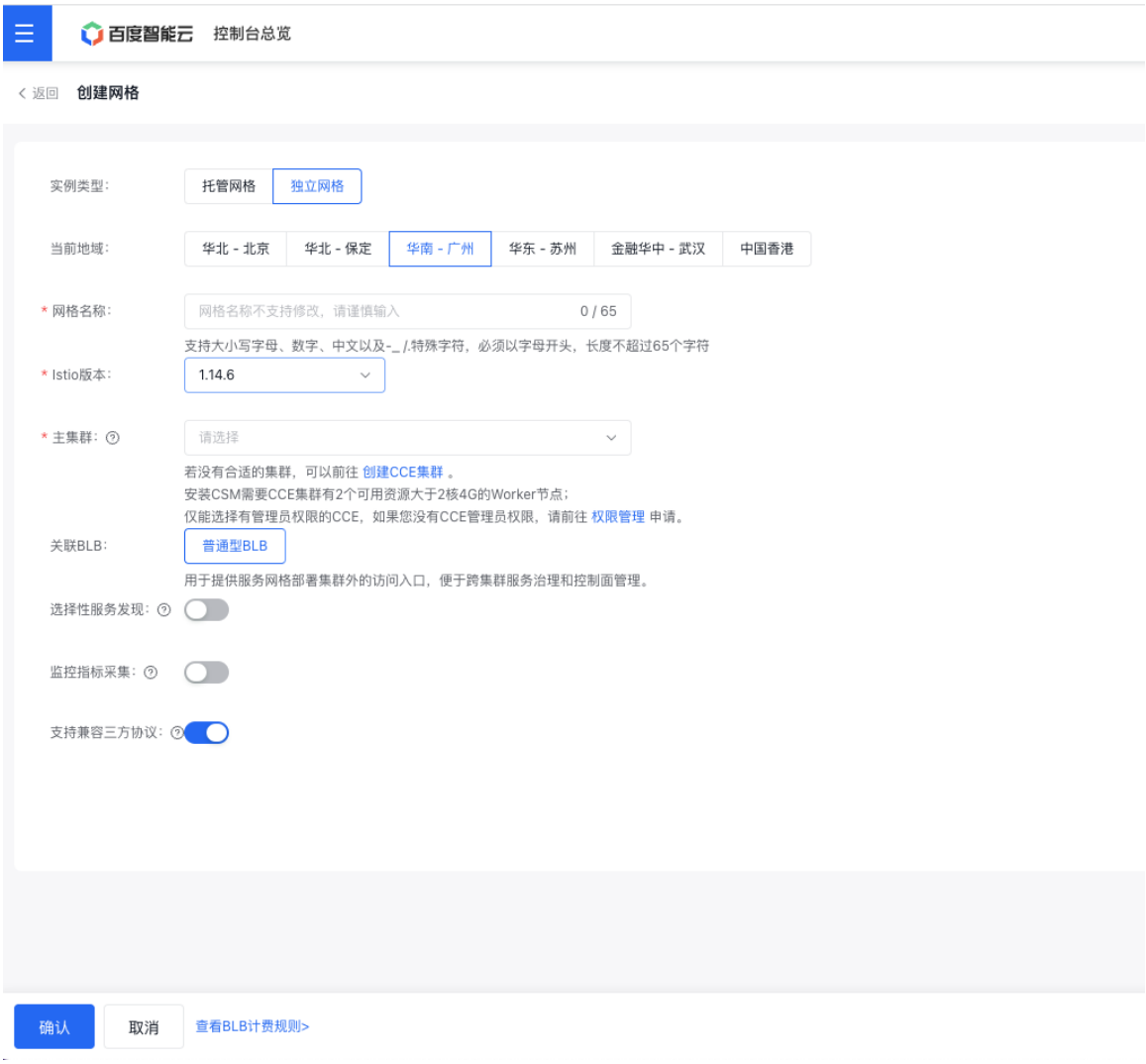
创建独立网格实例，并且开启支持兼容三方协议按钮。

创建实例

按以下步骤在控制台上创建独立网格实例并且开启多协议支持：

1. 登录百度智能云控制台，选择“产品服务 > 云计算 > 容器 > 服务网格 CSM”；
2. 在全局概览页面通过“创建实例”按钮进入实例创建页面；

3. 在创建服务网格页面，完成基础配置；

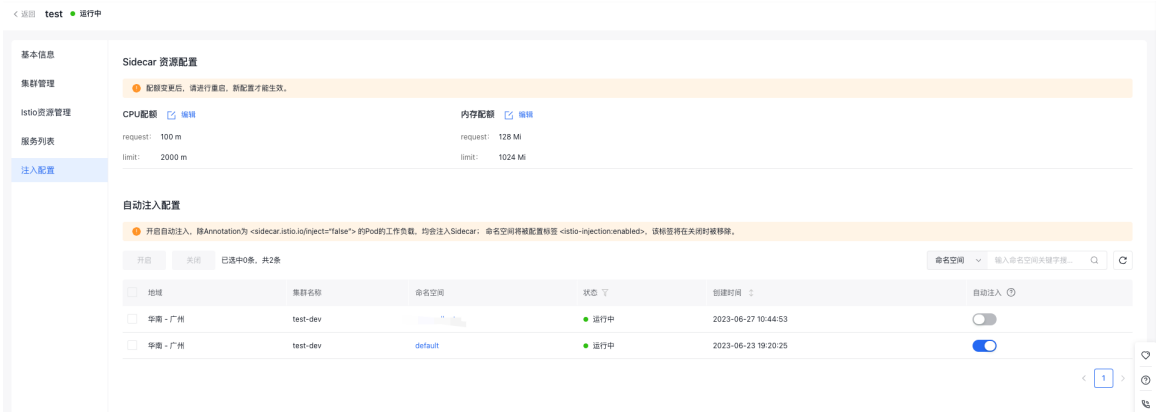


4. 查看网络列表，如下所示：

部署应用

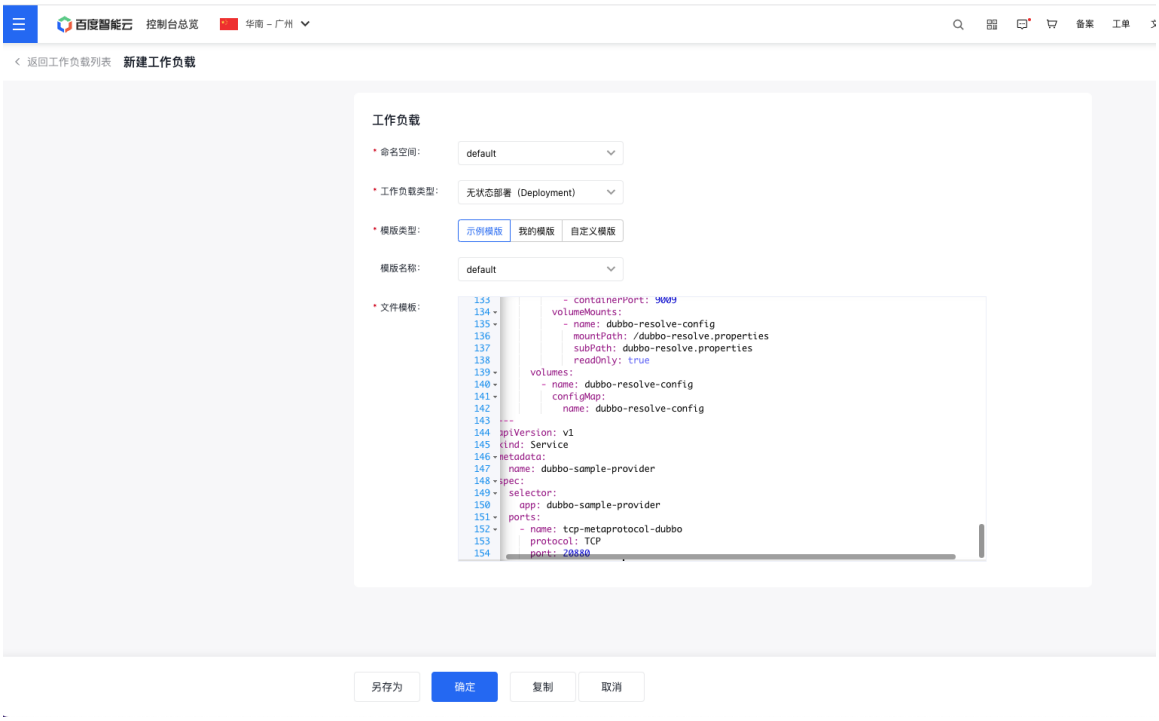
按以下步骤在CCE集群部署应用：

1. 在CSM/注入配置处，给default命名空间开启自动注入；



2. 通过CSM页面跳转到CCE集群，部署应用；

3. 在工作负载/无状态部署新建测试应用。



测试应用的Yaml文件如下所示：

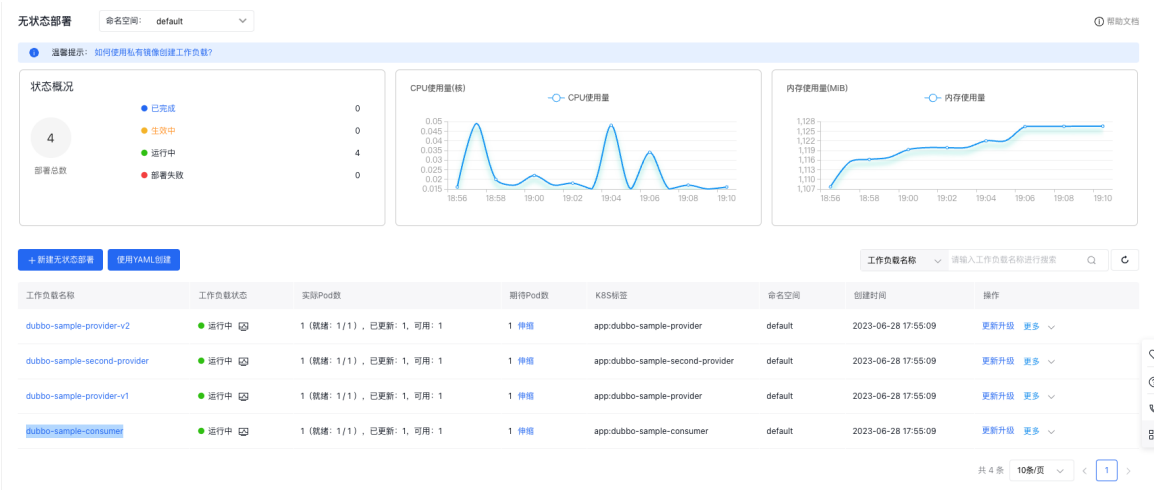
```
 apiVersion: v1
kind: ConfigMap
metadata:
  name: dubbo-resolve-config
data:
  dubbo-resolve.properties: |-
    org.apache.dubbo.samples.basic.api.DemoService=dubbo://dubbo-sample-provider:20880
    org.apache.dubbo.samples.basic.api.TestService=dubbo://dubbo-sample-provider:20880
    org.apache.dubbo.samples.basic.api.ComplexService=dubbo://dubbo-sample-provider:20880
    org.apache.dubbo.samples.basic.api.SecondService=dubbo://dubbo-sample-second-provider:20880
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dubbo-sample-provider-v1
  labels:
    app: dubbo-sample-provider
spec:
  selector:
    matchLabels:
      app: dubbo-sample-provider
  replicas: 1
  template:
    metadata:
      annotations:
        sidecar.istio.io/bootstrapOverride: aeraki-bootstrap-config
      labels:
        app: dubbo-sample-provider
        version: v1
```

```
        service_group: user
    spec:
      containers:
        - name: dubbo-sample-provider
          image: registry.baidubce.com/csm/dubbo-sample-provider:latest
          ports:
            - containerPort: 20880
          volumeMounts:
            - name: dubbo-resolve-config
              mountPath: /dubbo-resolve.properties
              subPath: dubbo-resolve.properties
              readOnly: true
      volumes:
        - name: dubbo-resolve-config
          configMap:
            name: dubbo-resolve-config
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dubbo-sample-provider-v2
  labels:
    app: dubbo-sample-provider
spec:
  selector:
    matchLabels:
      app: dubbo-sample-provider
  replicas: 1
  template:
    metadata:
      annotations:
        sidecar.istio.io/bootstrapOverride: aeraki-bootstrap-config
      labels:
        app: dubbo-sample-provider
        version: v2
        service_group: batchjob
    spec:
      containers:
        - name: dubbo-sample-provider
          image: registry.baidubce.com/csm/dubbo-sample-provider:latest
          ports:
            - containerPort: 20880
          volumeMounts:
            - name: dubbo-resolve-config
              mountPath: /dubbo-resolve.properties
              subPath: dubbo-resolve.properties
              readOnly: true
      volumes:
        - name: dubbo-resolve-config
          configMap:
            name: dubbo-resolve-config
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dubbo-sample-second-provider
  labels:
    app: dubbo-sample-second-provider
spec:
  selector:
    matchLabels:
      app: dubbo-sample-second-provider
```

```
  replicas: 1
  template:
    metadata:
      annotations:
        sidecar.istio.io/bootstrapOverride: aeraki-bootstrap-config
      labels:
        app: dubbo-sample-second-provider
        version: v2
        service_group: batchjob
    spec:
      containers:
        - name: dubbo-sample-second-provider
          image: registry.baidubce.com/csm/dubbo-sample-second-provider
          ports:
            - containerPort: 20880
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dubbo-sample-consumer
  labels:
    app: dubbo-sample-consumer
spec:
  selector:
    matchLabels:
      app: dubbo-sample-consumer
  replicas: 1
  template:
    metadata:
      annotations:
        sidecar.istio.io/bootstrapOverride: aeraki-bootstrap-config
      labels:
        app: dubbo-sample-consumer
    spec:
      containers:
        - name: dubbo-sample-consumer
          image: registry.baidubce.com/csm/dubbo-sample-consumer:latest
          env:
            - name: mode
              value: demo
          ports:
            - containerPort: 9009
          volumeMounts:
            - name: dubbo-resolve-config
              mountPath: /dubbo-resolve.properties
              subPath: dubbo-resolve.properties
              readOnly: true
      volumes:
        - name: dubbo-resolve-config
          configMap:
            name: dubbo-resolve-config
---
apiVersion: v1
kind: Service
metadata:
  name: dubbo-sample-provider
spec:
  selector:
    app: dubbo-sample-provider
  ports:
    - name: tcp-metaprotocol-dubbo
      protocol: TCP
      port: 20880
```

```
    port: 20880
    targetPort: 20880
```

4. 工作负载如下所示：



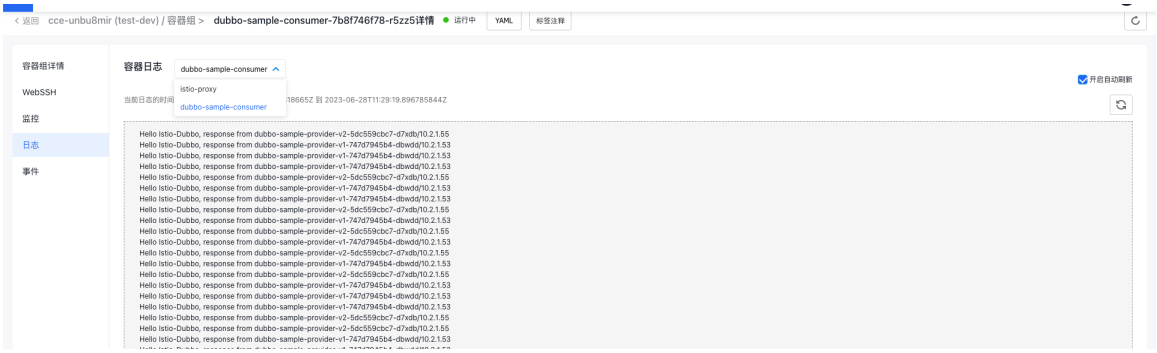
5. 在CSM/Istio资源管理处下发destinationrule规则。

   destinationrule的Yaml文件如下所示：

```
 apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: dubbo-sample-provider
spec:
 host: dubbo-sample-provider.default.svc.cluster.local
 subsets:
 - name: v1
   labels:
     version: v1
 - name: v2
   labels:
     version: v2
```

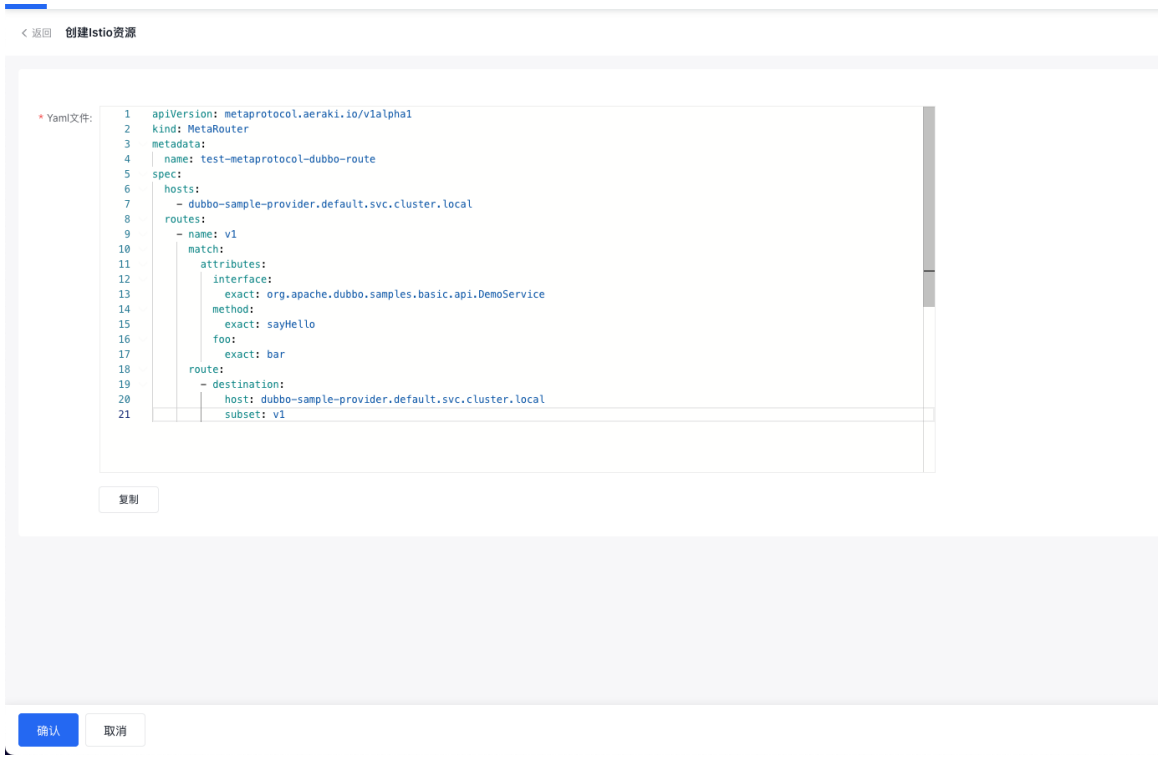
流量验证

1. 通过CCE集群中的工作负载查看业务容器中的日志：



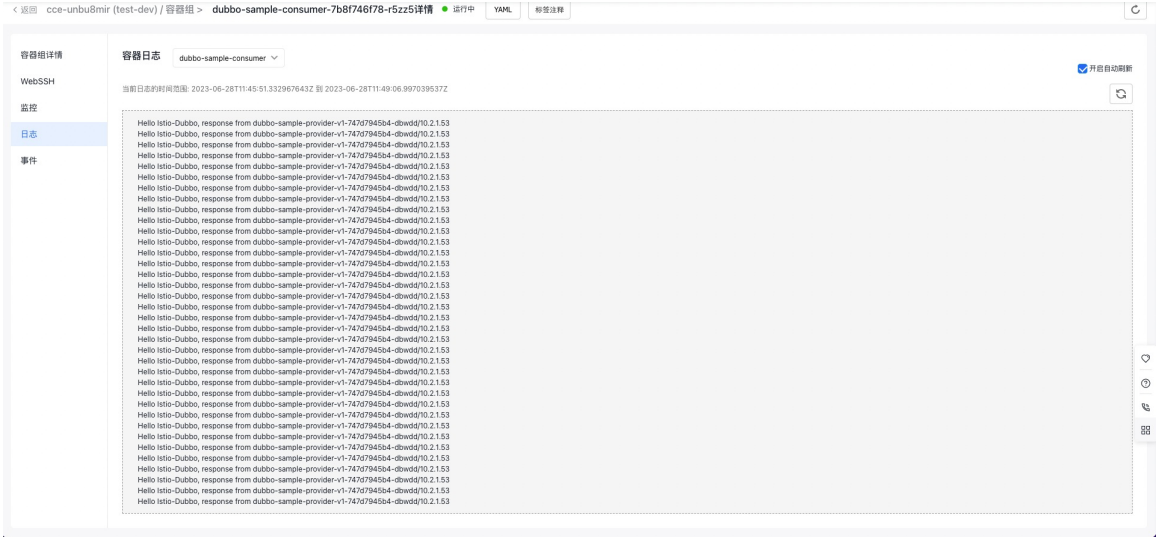
   从日志中可以得知，consumer请求provider-demo,轮询调用v1与v2版本，符合预期。

2. 从CSM/Istio资源管理处下发流量治理策略，全部访问v1版本。




```
 apiVersion: metaprotocol.aeraki.io/v1alpha1
kind: MetaRouter
metadata:
 name: test-metaprotocol-dubbo-route
spec:
 hosts:
  - dubbo-sample-provider.default.svc.cluster.local
 routes:
  - name: v1
    match:
      attributes:
        interface:
          exact: org.apache.dubbo.samples.basic.api.DemoService
        method:
          exact: sayHello
        foo:
          exact: bar
    route:
      - destination:
          host: dubbo-sample-provider.default.svc.cluster.local
          subset: v1
```

3. 通过CCE集群中的工作负载在此查看业务容器中的日志：



从日志中可以得知，consumer请求provider-demo,全部调用v1版本，符合预期。

说明

更多详细用法参考aeraki,在此感谢aeraki。

组件管理

概述

本节介绍服务网格CSM提供支持注册中心（consul）等组件管理功能（只有独立网格有此功能，托管网格没有）。

创建实例

按以下步骤在控制台上创建独立网格实例。

1. 登录百度智能云控制台，选择“产品服务 > 云计算 > 容器 > 服务网格 CSM”；
2. 在全局概览页面通过“创建实例”按钮进入实例创建页面；

3. 在创建服务网格页面，完成基础配置；

4. 查看网络实例组件详情，如下所示：

部署consul

在CCE集群部署consul服务。

测试的consul的Yaml文件如下所示：

```
 apiVersion: v1
kind: ConfigMap
metadata:
 name: consul
data:
 k8s-consul-config.json: |
  {
      "skip_leave_on_interrupt": true,
      "acl": {
         "enabled": true,
```

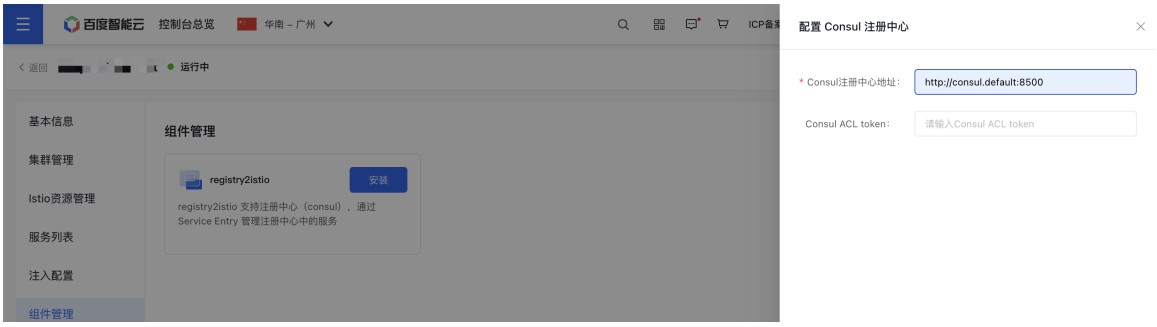
```
            "default_policy": "deny",
            "down_policy": "extend-cache",
            "tokens": {
                "master": "14d54c5e-24ca-41cc-8c9e-987ba7a96ffv"
            }
        }
    }

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: consul
  labels:
    app: consul
spec:
  selector:
    matchLabels:
      app: consul
  replicas: 1
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "false"
      labels:
        app: consul
    spec:
      containers:
        - name: consul
          image: registry.baidubce.com/csm-offline/consul:1.9.17
          args:
            - "agent"
            - "-server"
            - "-bootstrap-expect=1"
            - "-ui"
            - "-node=server1"
            - "-data-dir=/consul/data"
            - "-config-file=/etc/consul/config/k8s-consul-config.json"
            - "-bind=0.0.0.0"
            - "-client=0.0.0.0"
            - "-datacenter=dc1"
            - "-enable-script-checks=true"
          ports:
            - containerPort: 8500
          volumeMounts:
            - name: data
              mountPath: /consul/data
            - name: config
              mountPath: /etc/consul/config
      volumes:
        - name: data
          emptyDir: {}
        - name: config
          configMap:
            name: consul
---
apiVersion: v1
kind: Service
metadata:
  name: consul
spec:
  type: LoadBalancer
  selector:
```

```
selector:
  app: consul
ports:
  - name: tcp
    port: 8500
    protocol: TCP
    targetPort: 8500
```

安装registry2istio组件

1. 点击服务网格实例，在组件管理处点击安装registry2istio，填写consul地址与acl token（看consul集群是否开启acl token认证）。

2. consul信息填写如下所示：

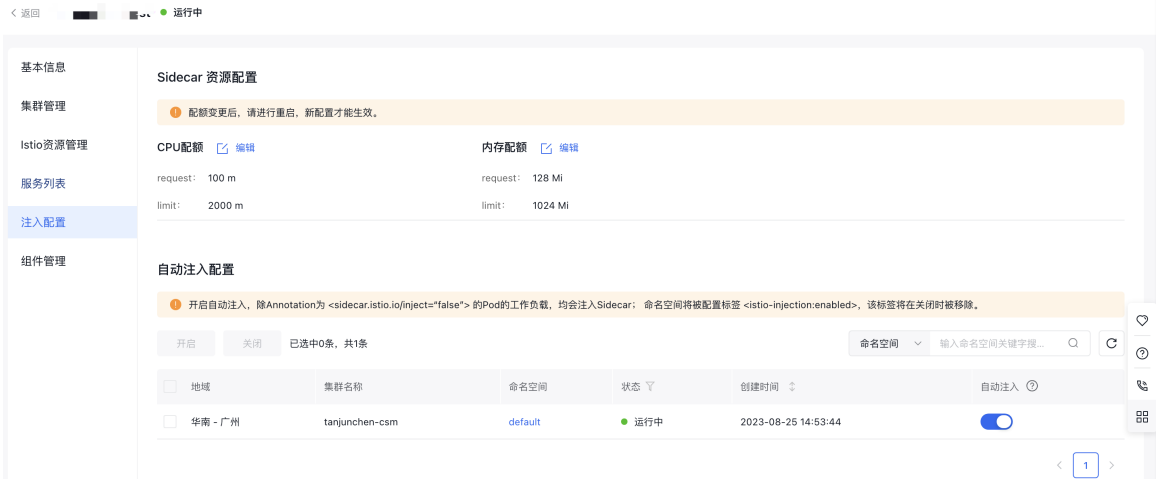


3. 安装成功，如下所示：

部署应用

在CCE集群部署应用：

1. 在CSM注入配置处，给default命名空间开启自动注入；



2. 通过CSM页面跳转到CCE集群，部署应用；

3. 在工作负载/无状态部署新建测试应用；

测试Yaml文件如下所示：

```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: provider-demo-v1
  labels:
    app: provider-demo
spec:
  selector:
    matchLabels:
      app: provider-demo
  replicas: 1
  template:
    metadata:
      labels:
        app: provider-demo
        version: v1
    spec:
      containers:
        - name: provider-demo
          image: registry.baidubce.com/csm-offline/csm-consul-mesh-provider:dev-consul-acl-token
          imagePullPolicy: Always
          ports:
            - containerPort: 10001
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: consumer-demo
  labels:
    app: consumer-demo
spec:
  selector:
    matchLabels:
      app: consumer-demo
  replicas: 1
  template:
    metadata:
      labels:
        app: consumer-demo
    spec:
      containers:
        - name: consumer-demo
          image: registry.baidubce.com/csm-offline/csm-consul-mesh-consumer:dev-consul-acl-token
          imagePullPolicy: Always
          ports:
            - containerPort: 9999
```

4. 工作负载如下所示：

```
→ kubectl get pod -owide
NAME                                READY   STATUS        RESTARTS   AGE   IP           NODE           NOMINATED NODE   READINESS
GATES
consul-668c95fb7c-d9p68             1/1     Running       0          39m   10.0.1.172   192.168.1.13   <none>           <none>
consumer-demo-8696cd6b8c-dn7pr      2/2     Terminating   0          39m   10.0.1.27    192.168.1.13   <none>           <none>
consumer-demo-8696cd6b8c-l8dr4      2/2     Running       0          4s    10.0.1.167   192.168.1.13   <none>           <none>
provider-demo-v1-64cb4657cf-6z5m2   2/2     Running       0          4s    10.0.0.84    192.168.1.15   <none>           <none>
provider-demo-v1-64cb4657cf-nlv65   2/2     Terminating   0          39m   10.0.1.195   192.168.1.13   <none>           <none>
→
```

流量验证

1. 通过CCE集群中的工作负载consumer-demo访问provider-demo，测试流量是否能够正常通信；

```
➜ registry2istio git:(master) kubectl exec -it consumer-demo-8696cd6b8c-l8dr4 -- curl 10.0.1.167:9999/echo-rest/aaaaaaaaaaa
echo() -> ip [ 10.0.0.84 ] param [ aaaaaaaaaaa ] %
➜ registry2istio git:(master) kubectl exec -it consumer-demo-8696cd6b8c-l8dr4 -- curl 10.0.1.167:9999/echo-rest/aaaaaaaaaaa
echo() -> ip [ 10.0.0.84 ] param [ aaaaaaaaaaa ] %
```

2. 测试结果如下所示：

3. consul中的consumer-demo与provider-demo服务成功被istiod管理，流量请求成功。

流量泳道

流量泳道概述

流量泳道是一种流量管理技术，用于将流量按照不同的需求或优先级分配到不同的服务或应用程序中。具体来说，流量泳道将流量划分为不同的路径或“泳道”，每个泳道对应一个特定的服务或应用程序的版本。

服务网格CSM支持根据预设的规则和策略，将流量动态地分配到不同的泳道中，以满足不同的业务需求和性能目标。

在CSM中，您仅需确保创建一条基线泳道，该泳道需涵盖调用链路中的所有服务。其他泳道则无需包含完整的调用链路服务。当服务在某一泳道内相互调用时，若目标服务不在当前泳道内，请求将被重定向至基线泳道中的对应服务。一旦目标服务在当前泳道内被发现，请求将再次被转发回原泳道。若您选择使用灵活的流量泳道模式，您的应用程序必须包含一个能够在整个调用链路中持续传递的请求头，即链路透传请求头，并确保每个请求都具有独特的请求头值。

前提条件

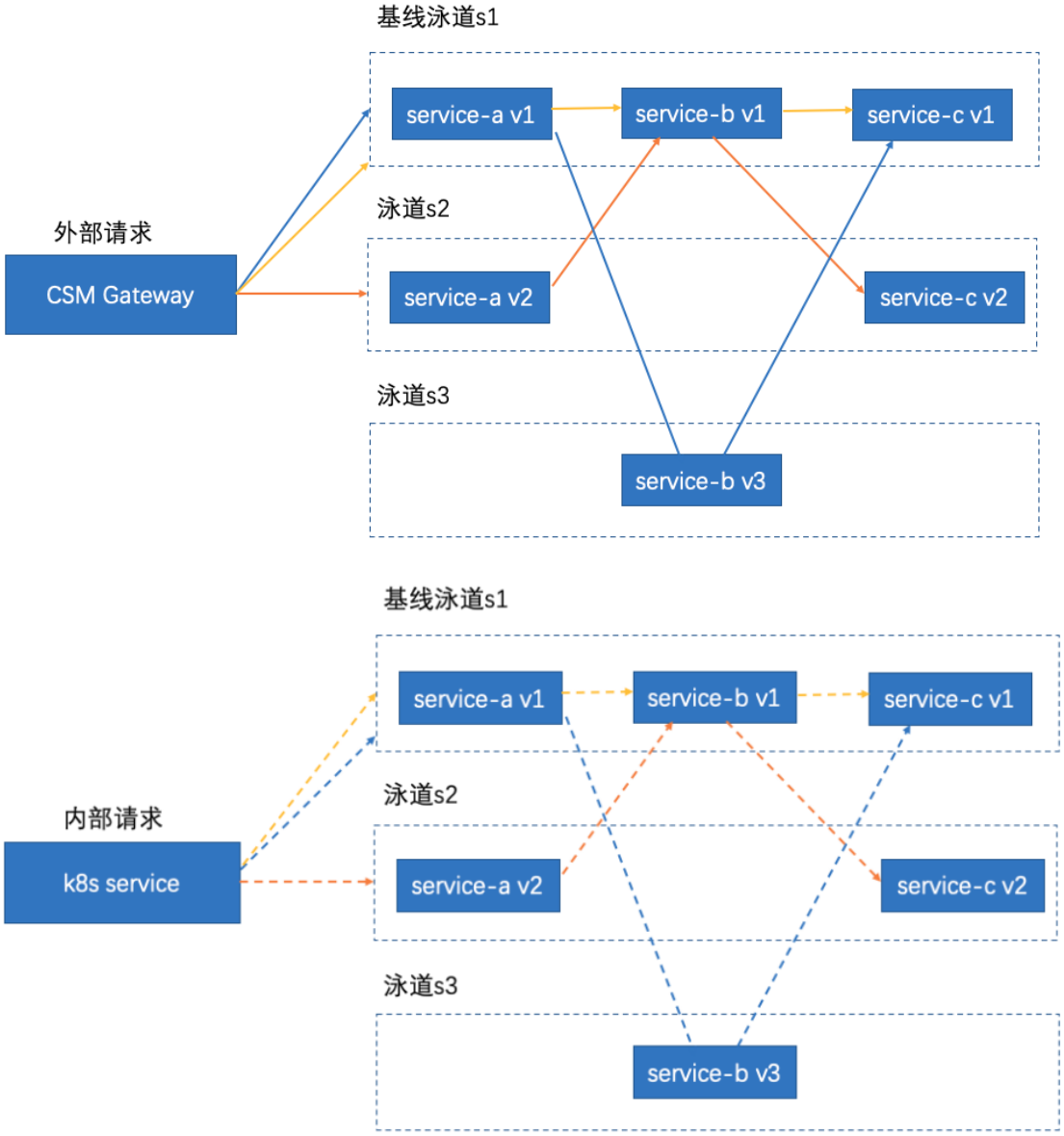
- 已创建CSM网格实例，且版本为1.14.6及以上。具体操作，请参见CSM实例管理
- 已添加CCE集群到CSM实例。

功能介绍

您的应用程序必须包含一个能够在整条调用链路中透传的请求头（链路透传请求头），且链路透传请求头的值对于每条请求都各不相同。同时，您需要指定一个引流的请求头，CSM网关将会根据引流请求头的内容将流量发往不同的流量泳道，保障链路完整性，简化流量管理。

- 链路透传请求头：在服务链路中沿着请求路径传递的HTTP头部信息，这些header信息通常包括用于跟踪、日志记录或安全目的的数据，如跟踪ID（trace ID）。透传header有助于在跨多个服务的调用中维持请求上下文的连续性。
- 引流请求头：专门用于控制和管理流量分配的HTTP头部信息，特别是在灰度发布或A/B测试中。通过设置引流header，可以根据特定的标识（如版本号或用户群组）将流量引导至不同的服务版本或实例。可以在不影响所有用户的情况下测试新功能，或将特定用户群体的流量引入特定的处理路径。引流header是实现精细流量控制的关键工具。

以service-a、service-b、service-c三个服务为例，您可以创建三个泳道：s1、s2和s3，它们分别代表了服务调用链的三个不同版本。其中，s1作为基线泳道，完整地包含了这三个服务；s2则仅包含service-a和service-c两个服务；而s3仅包含service-b一个服务。通过这样的配置，CSM网关可以根据引流请求头的内容，将流量导向相应的泳道，从而实现对服务调用的灵活控制和流量的有效管理。



注：流量泳道功能既支持CSM Gateway的外部请求，也支持K8s集群内服务的内部请求

步骤一：部署示例服务

1. 为default命名空间启用Sidecar网格代理自动注入。具体操作，请参见注入配置。

2. 在纳管CCE集群中执行以下命令，部署示例服务。

```
 kubectl apply -f service-a.yaml
kubectl apply -f service-b.yaml
nkubectl apply -f service-c.yaml
```

service-a/b/c.yaml 文件如下：

```
 apiVersion: v1
kind: Service
metadata:
  name: service-a
spec:
  selector:
    app: service-a
  ports:
    - protocol: TCP
      port: 80
```

```
    targetPort: 8080
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-a-v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-a
      version: v1
  template:
    metadata:
      labels:
        app: service-a
        version: v1
    spec:
      containers:
      - name: service-a
        image: registry.baidubce.com/csm-offline/service-a:dev
        imagePullPolicy: Always
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        ports:
        - containerPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-a-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-a
      version: v2
  template:
    metadata:
      labels:
        app: service-a
        version: v2
    spec:
      containers:
      - name: service-a
        image: registry.baidubce.com/csm-offline/service-a:dev
        imagePullPolicy: Always
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        ports:
        - containerPort: 8080
```

```
  apiVersion: v1
kind: Service
metadata:
```

```
  name: service-b
spec:
  selector:
    app: service-b
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-b-v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-b
      version: v1
  template:
    metadata:
      labels:
        app: service-b
        version: v1
    spec:
      containers:
      - name: service-b
        image: registry.baidubce.com/csm-offline/service-b:dev
        imagePullPolicy: Always
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        ports:
        - containerPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-b-v3
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-b
      version: v3
  template:
    metadata:
      labels:
        app: service-b
        version: v3
    spec:
      containers:
      - name: service-b
        image: registry.baidubce.com/csm-offline/service-b:dev
        imagePullPolicy: Always
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
```

```
            fieldPath: metadata.name
        ports:
        - containerPort: 8080
```

```
 apiVersion: v1
kind: Service
metadata:
  name: service-c
spec:
  selector:
    app: service-c
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-c-v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-c
      version: v1
  template:
    metadata:
      labels:
        app: service-c
        version: v1
    spec:
      containers:
      - name: service-c
        image: registry.baidubce.com/csm-offline/service-c:dev
        imagePullPolicy: Always
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        ports:
        - containerPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-c-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-c
      version: v2
  template:
    metadata:
      labels:
        app: service-c
        version: v2
    spec:
      containers:
```

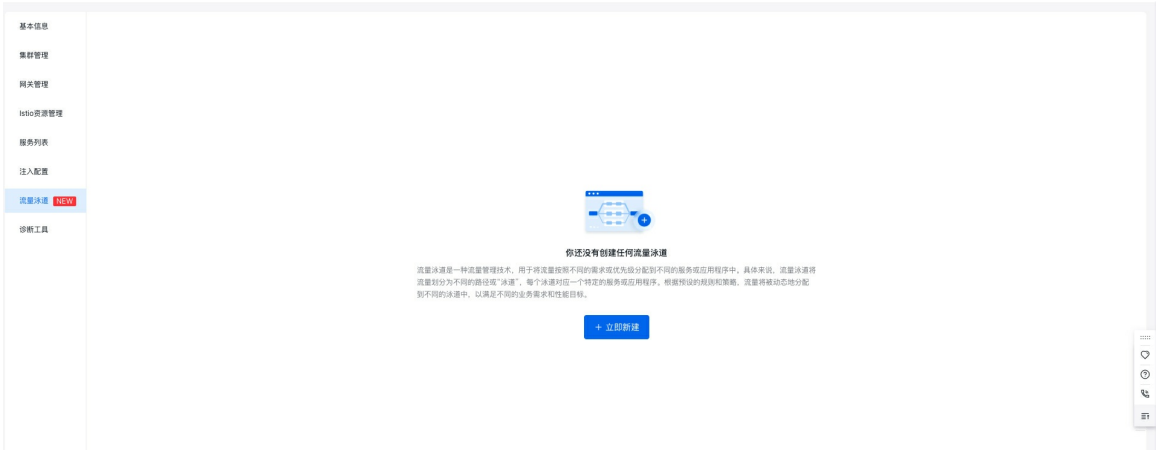
```
  - name: service-c
    image: registry.baidubce.com/csm-offline/service-c:dev
    imagePullPolicy: Always
    env:
    - name: POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    ports:
    - containerPort: 8080
```

步骤二：创建泳道组和对应泳道

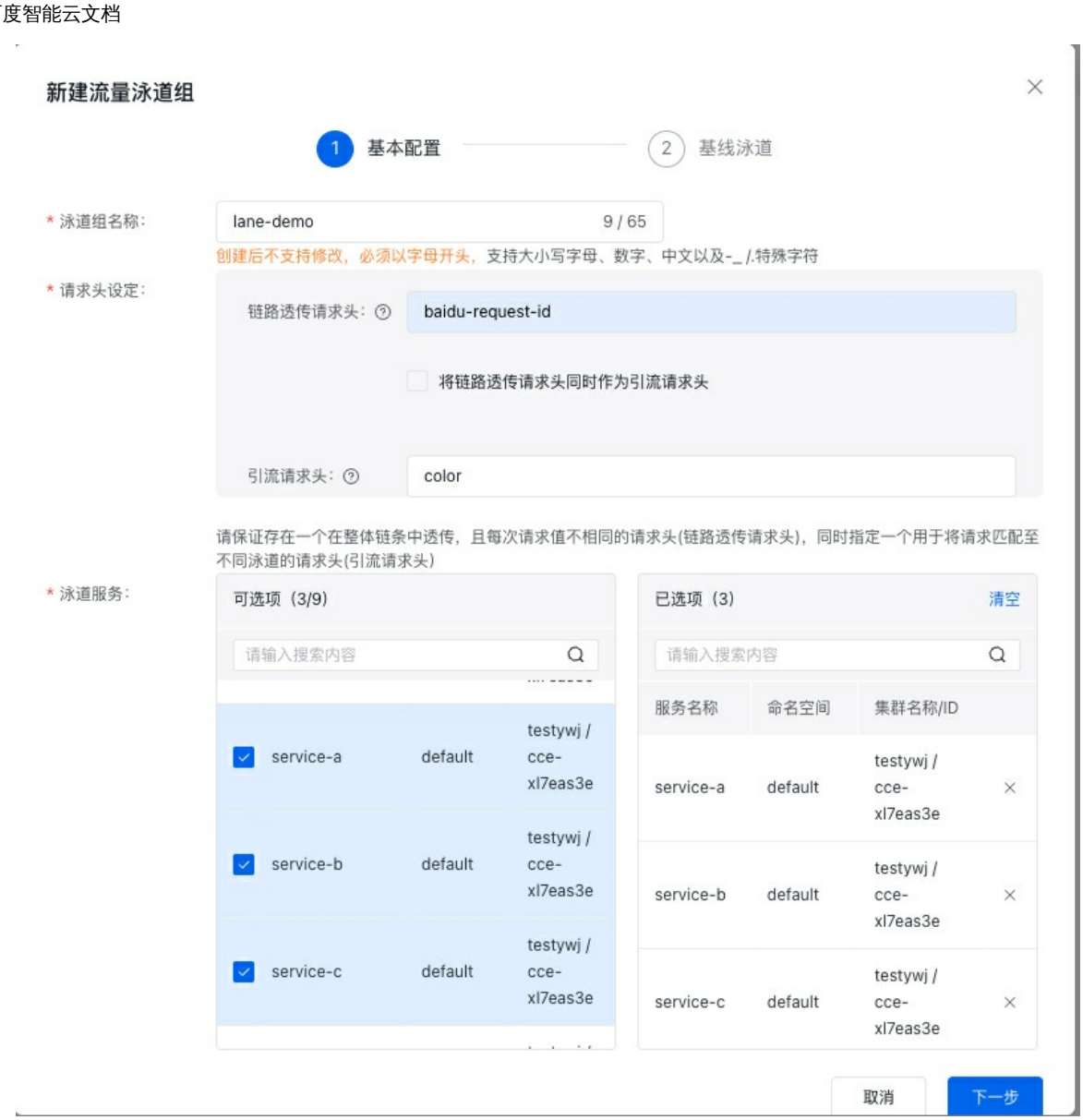
1. 创建泳道组

a. 登录百度智能云控制台，选择“产品服务 > 云计算 > 容器 > 服务网格 CSM”。

b. 在服务网格控制台，点击期望操作的服务网格实例名称，点击"流量泳道"。



c. 在流量泳道页面，单击立即新建，在创建泳道组面板，配置相关信息，然后单击下一步。



配置项	说明
泳道组名称	本示例配置为**lane-demo**。
请求头设定	**链路透传请求头：**由于示例应用在调用链路中透传了请求头baidu-request-id，因此本示例配置为**baidu-request-id**。 **引流请求头：**用于网关根据请求头内容向不同泳道引流及泳道上下文保持，可任意指定。本示例配置为**color**。
泳道服务	列表包含网格实例下所有纳管cce集群的k8s服务和ServiceEntry。在下方列表中选中**servce-a**、**service-b**和**service-c**服务，点击勾选☑，添加目标服务到**已选择**区域。

> 在链路中未透传引流请求头，引流请求头与链路透传请求头并不相同（分别为version和baidu-request-id）。在这种情况下，需要链路透传请求头中的内容针对每次请求都不相同（即每次调用链路都有唯一的链路ID）。如果同时将链路透传请求头指定为引流请求头，则针对链路透传请求头不再需要上述的限制，只需用链路透传请求头的内容向不同泳道引流即可。

d. 创建基线泳道，在"基线泳道"页面，配置泳道名称和标签信息，然后单击确定。

注：基线泳道必须包含所有泳道组服务，因此创建基线泳道时，默认选中所有服务且不能修改。

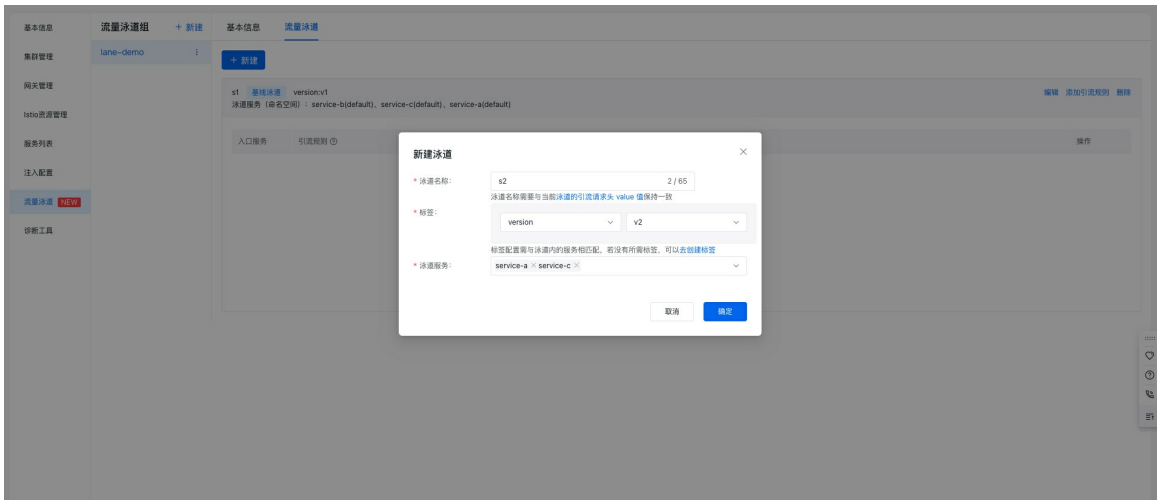
配置项	说明
泳道名称	基线泳道配置为s1。**泳道名称需要与当前泳道的引流请求头 value 值保持一致**。 例如：在本示例中只要请求中包含"color: s1"信息的请求，都会匹配到基线泳道s1（泳道名称）中来。
标签	标签名称：本示例选择verison。 标签值：本示例选择选择v1。
泳道服务	基线泳道（默认全选）：选择service-a(default)、service-b(default)、service-c(default)。

成功创建完泳道组和基线泳道后，可以在"istio资源管理"中查看泳道相关的virtualService 和 DestinationRule。示例如下：

```
  apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-delegate-service-a
  namespace: {namespace}
spec:
  hosts:
  - service-a.default.svc.cluster.local
  http:
  - delegate:
      name: vs-l-i4ew8u31-service-a
      namespace: {namespace}
    match:
    - headers:
        color:
          exact: s1
  - route:
    - destination:
        host: service-a.default.svc.cluster.local
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-l-i4ew8u31-service-a
  namespace: {namespace}
spec:
  http:
  - match:
    - headers:
        color:
          exact: s1
    route:
    - destination:
        host: service-a.default.svc.cluster.local
        subset: s1
```

```
  apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: dr-l-i4ew8u31-service-a
  namespace: {namespace}
spec:
  host: service-a.default.svc.cluster.local
  subsets:
  - labels:
      version: v1
    name: s1
```

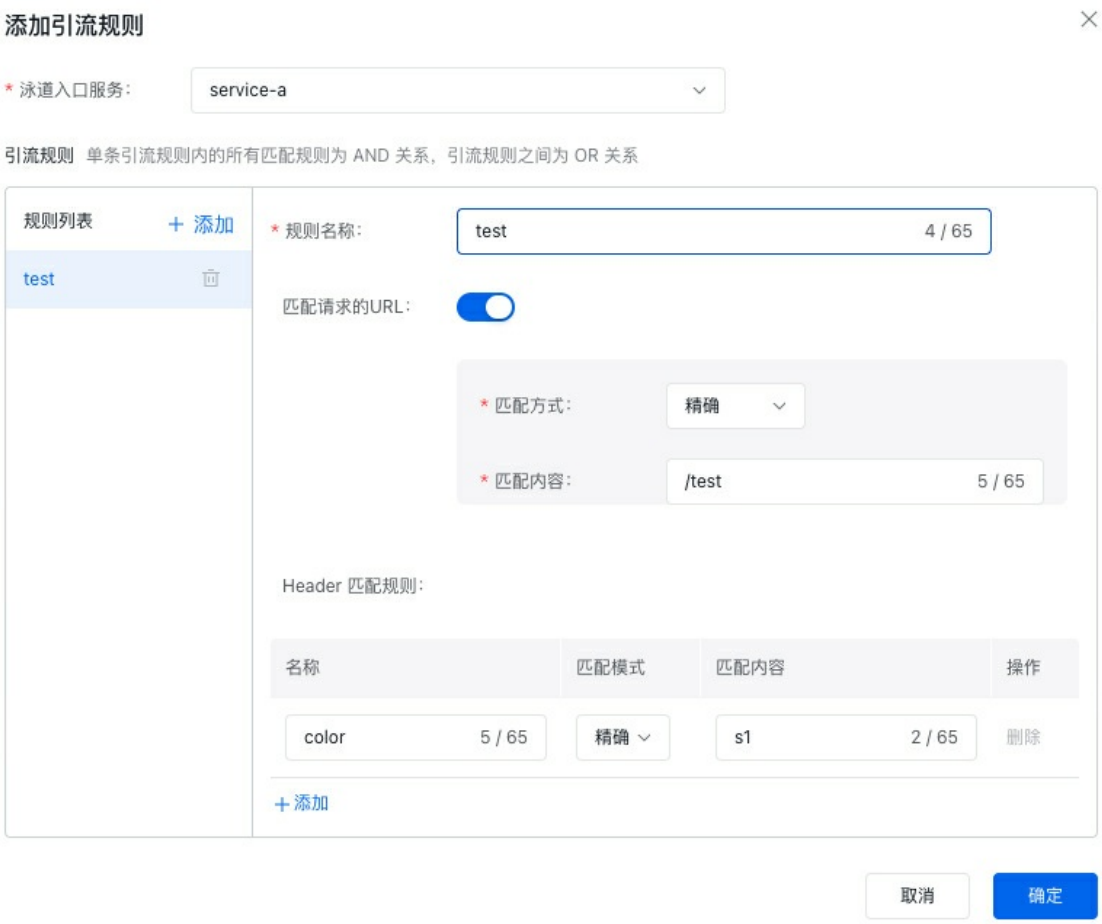
2.新建泳道



配置项	说明
泳道名称	两条泳道分别配置为**s2**、**s3**。
标签	**标签名称**：选择**verison** **标签值**：两条泳道分别选择**v2**、**v3**。
泳道服务	s2泳道：选择**service-a(default)**、**service-c(default)**。 s3泳道：选择**service-b(default)**。

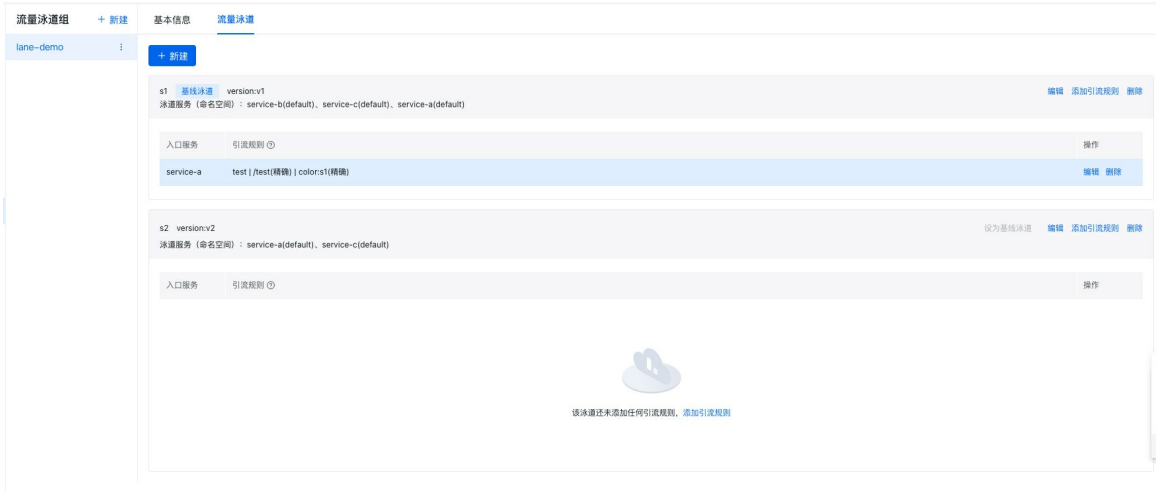
3. 创建引流规则 a. 在**流量泳道**页面，单击目标泳道右侧操作列下的添加**引流规则**。

b. 在**添加引流规则**弹出框，配置相关信息，然后单击**确定**。本文以泳道服务对应匹配请求的URL均为/test为例，为service-a服务配置引流规则。



配置项	说明
入口服务	选择**service-a**。
规则名称	本示例为test。
匹配请求的URI	配置**匹配方式**为**精确**，**匹配内容**为**/test**。
Header匹配规则	每一条引流规则都默认添加一个Header匹配规则—— "{引流请求头}：{泳道名称}" 本示例中默认添加"color：s1"

创建成功后，示例效果如下:



c. 创建成功后，会自动生成对应的引流规则，即虚拟服务VirtualService。例如，针对泳道s1的service-a服务会生成如下的虚拟服务VirtualService。

```
  apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-l-i4ew8u31-service-a
  namespace: {namespace}
spec:
  http:
  - match:
    - headers:
        color:
          exact: s1
      name: test
      uri:
        exact: /test
    - headers:
        color:
          exact: s1
    route:
    - destination:
        host: service-a.default.svc.cluster.local
        subset: s1
```

步骤三：验证全链路灰度功能是否生效

集群内服务访问

1. 在cce集群内default命名空间下导入sleep服务，sleep服务见istio官方文档 sleep服务。

```
kubectl apply -f samples/sleep/sleep.yaml
```

2. 等待sleep服务pod创建成功，验证全链路灰度功能是否生效。
   a. 执行以下命令，查看s1泳道的访问效果。

```
for i in {1..20};  do kubectl exec -it {sleep-pod-name} -c sleep -- curl  -H 'color: s1' -H'baidu-request-id: x0000'$i http://service-a/test ;  echo ''; sleep 0.2; done;
```

预期结果如下

```
Handled by: service-a-v1-548cb45968-fj2rc -> service-b-v1-64c976c64c-n2m6w -> service-c-v1-8f9c58dcd-ctjhl
```

由预期输出得到，通过设置HTTP标头color: s1声明的流量流向s1泳道下的相关服务，符合预期。
b. 执行以下命令，查看s2泳道的访问效果。

```
for i in {1..20};  do kubectl exec -it {sleep-pod-name} -c sleep -- curl  -H 'color: s2' -H'baidu-request-id: x0001'$i http://service-a/test ;  echo ''; sleep 0.2; done;
```

预期结果如下

```
Handled by: service-a-v2-585865d66f-z76zl -> service-b-v1-64c976c64c-n2m6w -> service-c-v2-579b589cf9-v4wcp
```

由预期输出得到，通过设置HTTP标头color: s2声明的流量流向s2泳道下的相关服务。当流量发往泳道s2中不存在的服务service-b时，流量发往基线泳道s1中的service-b服务，后续流量发往service-c服务时，目标重新设定为s2泳道中的service-c服务，符合预期。
c. 执行以下命令，查看s3泳道的访问效果。

```
for i in {1..20};  do kubectl exec -it {sleep-pod-name} -c sleep -- curl  -H 'color: s3' -H'baidu-request-id: x0000'$i http://service-a/test ;  echo ''; sleep 0.2; done;
```

预期结果如下

```
Handled by: service-a-v1-548cb45968-fj2rc -> service-b-v3-6bdc685945-r97zt -> service-c-v1-8f9c58dcd-ctjhl
```

由预期输出得到，通过设置HTTP标头color: s3声明的流量流向s3泳道下的相关服务。当流量发往泳道s3中不存在的服务service-a、service-c时，流量发往基线泳道s1中的service-a、service-c服务，符合预期。

托管网关验证 前提：必须为托管网格实例，并且创建了托管网关。创建网关详情见网关管理。

1. 获取托管网关的公网IP（来源于网关入口中的公网地址）。

2. 设置环境变量，xxx.xxx.xxx.xxx为上一步获取的IP。

```
export GATEWAY_IP=xxx.xxx.xxx.xxx
```

3. 在"istio资源管理"页面中，中，修改gateway-vs，添加域名和相应的网关引流规则。例如：配置一个s1泳道到service-a服务的网关引流规则如下：

```
  apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: gateway-vs
  namespace: {namespace}
spec:
  gateways:
  - gateway-gw
  hosts:
  - service-a.com
  http:
  - match:
    - headers:
        color:
          exact: s1
      uri:
        exact: /test
    route:
    - destination:
        host: service-a.default.svc.cluster.local
        port:
          number: 80
        subset: s1
```

4. 验证全链路灰度功能是否生效。本示例只演示托管网关请求泳道s1，执行以下命令，查看s1泳道的访问效果。

```
  for i in {1..20}; do curl -H "baidu-request-id: 11" -H "color:s1" http://{service-a服务对应域名}/test --resolve "{service-a服务对应域名}:80:${GATEWAY_IP}"； echo ''; sleep 0.2; done;
```

预期结果

```
  Handled by: service-a-v1-548cb45968-fj2rc -> service-b-v1-64c976c64c-n2m6w -> service-c-v1-8f9c58dcd-ctjhl
```

由预期输出得到，通过设置HTTP标头color: s1声明的流量流向s1泳道下的相关服务，符合预期。

可观测管理

对接Cprom实现监控告警

概述

服务网格可实现微服务无侵入地获得服务间请求的监控指标数据，本文档帮助用户实现服务网格CSM产品对接Prometheus监控服务（CProm产品），实现对服务网格中指标的监控告警配置和大盘展示。

前提条件

1. 已创建**与Kubernetes集群同地域的CProm实例**，详情请参考：创建CProm实例。
2. 对于已运行工作负载的Kubernetes集群，需要安装CProm采集Agent，用于采集指标，详情请参考：Agent管理。
3. 注意：托管服务网格暂不支持数据面监控告警。

操作步骤

托管网格 开启控制面指标监控 注意：被服务网格实例纳管的CCE集群，需要安装CProm采集Agent，否则无法选择相关的Cprom实例

1.登录百度智能云控制台，选择"产品服务>云计算>容器>服务网格 CSM>网格列表"

2.单击目标网格实例名称，然后左侧导航栏选择**可观测管理 > Prometheus监控**，**在Prometheus监控页面选择立即开启**，并选

择对应的Cprom实例

配置控制面指标监控 开启控制面指标监控后，您通过可观测管理 > Prometheus监控页面

- 选择**Grafana服务**，跳转至Grafana信息页，您可通过Grafana公网域名以及对应的用户名密码访问Grafana大盘；
- 选择**查看详情**，您可查看当前Cprom实例信息；
- 选择**配置告警**，跳转至Cprom对应页面进行配置，具体指标选择及告警规则配置可参考下文：

控制面监控指标

指标名称	类型	描述
endpoint_no_pod	LastValue	没有关联Pod的端点
pilot_endpoint_not_ready	LastValue	发现处于未就绪状态的端点
pilot_destrule_subsets	LastValue	相同主机目的地规则的重复子集
pilot_duplicate_envoy_clusters	LastValue	由具有相同主机名的服务条目引起的重复envoy集群
pilot_no_ip	LastValue	在端点表中找不到Pod，可能无效
pilot_eds_no_instances	LastValue	没有实例的集群数量
pilot_vservice_dup_domain	LastValue	具有重复域的虚拟服务数量
citadel_server_root_cert_expiry_timestamp	LastValue	Citadel根证书过期的unix时间戳（秒）。负时间表示证书已过期
galley_validation_failed	Sum	资源验证失败总数

控制面告警配置参考

```
  ##### Pilot
- alert: IstioPilotPodNotInEndpointTable
  annotations:
    summary: "Pilot pods not found in the endpoint table"
    description: "Pods not found in the endpoint table, possibly invalid"
  expr: > pilot_no_ip > 0

- alert: IstioPilotEndpointNotReady
  annotations:
    summary: "Pilot endpoint found in unready state"
    description: "Pilot endpoint found in unready state for 30 second"
  expr: > pilot_endpoint_not_ready > 0
  for: 30s

- alert: IstioPilotDestruleSubsetsException
  annotations:
    summary: "Pilot pilot_destrule_subsets is greater than 0"
    description: "pilot_destrule_subsets Duplicate subsets across destination rules for same host"
  expr: > pilot_destrule_subsets > 0

- alert: IstioPilotDuplicateEnvoyClustersException
  annotations:
    summary: "Pilot pilot_duplicate_envoy_clusters is greater than 0"
    description: "pilot_duplicate_envoy_clusters Duplicate envoy clusters caused by service entries with same hostname"
  expr: > pilot_duplicate_envoy_clusters > 0

- alert: IstioPilotDuplicateEntry
  expr: sum(rate(pilot_duplicate_envoy_clusters{}[1m])) > 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: Istio Pilot Duplicate Entry on clusterID {{ $labels.clusterID }} and region {{ $labels.region }})
    description: "Istio pilot duplicate entry error.\n  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"

- alert: IstioPilotEndpointNoPodException
  annotations:
    summary: "Pilot endpoint_no_pod is greater than 0"
    description: "endpoint_no_pod Endpoints without an associated pod"
  expr: > endpoint_no_pod > 0

  alert: IstioPilotEdsNoInstancesException
```

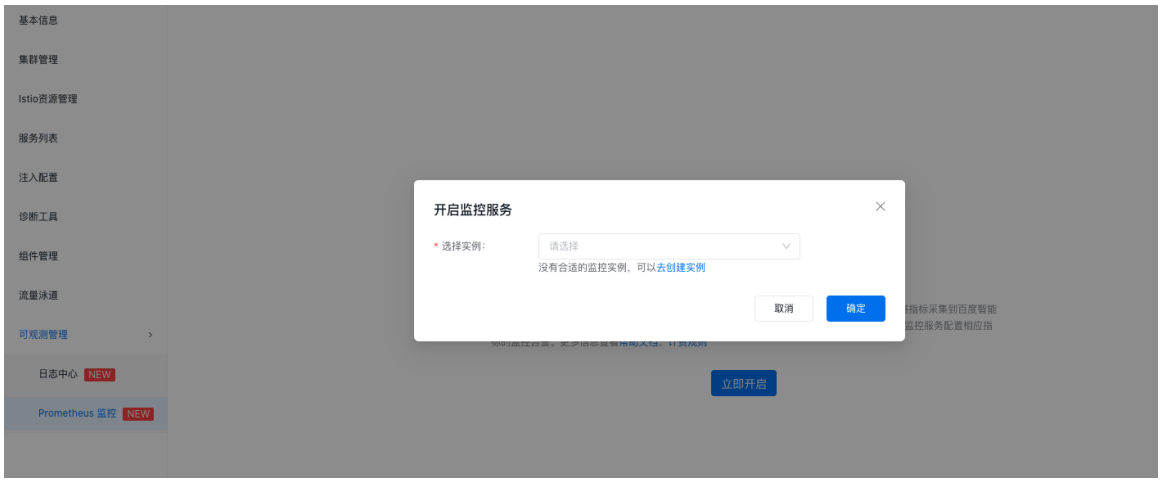
```
- alert: IstioPilotEdsNoInstancesException
  annotations:
    summary: "Pilot pilot_eds_no_instances is greater than 0"
    description: "pilot_eds_no_instances Number of clusters without instances"
  expr: > pilot_eds_no_instances > 0

- alert: IstioPilotVserviceDupDomainException
  annotations:
    summary: "Pilot pilot_vservice_dup_domain is greater than 0"
    description: "pilot_vservice_dup_domain Virtual services with dup domains"
  expr: > pilot_vservice_dup_domain > 0

##### CITADEL
- alert: IstioCitadelRootCertError
  annotations:
    summary: "Citadel root certificate internal error occured"
    description: "Citadel root certificate internal error occured on clusterID {{ $labels.clusterID }} and region {{
$labels.region }}"
  expr: >
  citadel_server_root_cert_expiry_timestamp < 0

- alert: IstioCitadelCertIssuanceFailure
  annotations:
    summary: "Citadel certificate issuance failed"
    description: "Citadel certificate issuance failed in last 1 minutes"
  expr: >
  (citadel_server_csr_count - citadel_server_success_cert_issuance_count) > (citadel_server_csr_count offset 1m -
citadel_server_success_cert_issuance_count offset 1m)

- alert: IstioCitadelCsrSignError
  annotations:
    summary: "Citadel CSR signing error"
    description: "Citadel CSR signing error occured in last 1 minutes on clusterID {{ $labels.clusterID }} and region {{
$labels.region }}"
  expr: >
  (absent(citadel_server_csr_sign_err_count offset 1m) == 1 and citadel_server_csr_sign_err_count > 0) or
(citadel_server_csr_sign_err_count - citadel_server_csr_sign_err_count offset 1m > 0)

##### GALLEY
- alert: IstioGalleyValidationFailed
  annotations:
    summary: "Galley validation failed"
    description: "Galley validation failed in last 1 minutes"
  expr: >
  (absent(galley_validation_failed offset 1m) == 1 and galley_validation_failed > 0) or (galley_validation_failed -
galley_validation_failed offset 1m > 0)
```

独立网格 开启数据面指标监控

注意：被服务网格实例纳管的cce集群，需要安装CProm采集Agent，否则无法选择相关的Cprom实例

- 方式一：在服务网格实例创建时，开启“监控指标采集”并选择对应的Cprom实例

- 方式二：针对已有的服务网格，选择服务网格 > 网格管理，在网格管理页面，单击目标实例名称，然后左侧导航栏选择可观测管理 > Prometheus监控，在监控页面选择开启，并选择对应的Cprom实例



配置数据面指标监控

开启数据面指标监控后，您通过可观测管理 > Prometheus监控页面

- 选择Grafana服务，跳转至Grafana信息页，您可通过Grafana公网域名访问Grafana大盘；
- 选择查看详情，您可查看当前Cprom实例信息；
- 选择配置告警，跳转至Cprom对应页面进行配置，具体指标选择及告警规则配置可参考下文：

数据面监控指标：

范围	名称	功能
Envoy	IstioEnvoyInternalUpstreamReq503TooHigh	503内部上游响应的数量高于1%，比例过高。
Envoy	IstioEnvoyInternalUpstreamReq200TooLow	200内部上游响应的数量低于99.9%，比例过低。
Envoy	IstioEnvoyUpstreamReq503TooHigh	Envoy 的 HTTP 503 上游响应的百分比过高
Envoy	IstioEnvoyUpstreamReq200TooLow	Envoy 的 HTTP 200 上游响应的百分比过低
Envoy	IstioEnvoyClusterBindErrors	Envoy Cluster 集群绑定错误
Envoy	IstioEnvoyClusterDstHostInvalid	Envoy Cluster 集群目标主机无效

数据面告警配置参考：

```
  - alert: IstioEnvoyInternalUpstreamReq503TooHigh
    annotations:
      summary: 'Envoy Percentage of HTTP 503 internal upstream responses is too high'
      description: "The amount of 503 internal upstream responses is higher than 1%. It is too high"
    expr: >
    rate(envoy_cluster_internal_upstream_rq_503[1m])/rate(envoy_cluster_internal_upstream_rq_completed[1m]) > 0.01

  - alert: IstioEnvoyInternalUpstreamReq200TooLow
    annotations:
      summary: 'Envoy Percentage of HTTP 200 internal upstream responses is too low'
      description: "The amount of 200 internal upstream responses is lower than 99.9%. It is too low"
    expr: >
    rate(envoy_cluster_internal_upstream_rq_200[1m])/rate(envoy_cluster_internal_upstream_rq_completed[1m]) < 0.999

  - alert: IstioEnvoyUpstreamReq503TooHigh
    annotations:
      summary: 'Envoy Percentage of HTTP 503 upstream responses is too high'
      description: "The amount of 503 upstream responses is higher than 1%. It is too high"
    expr: >
    rate(envoy_cluster_upstream_rq_503[1m])/rate(envoy_cluster_upstream_rq_completed[1m]) > 0.01

  - alert: IstioEnvoyUpstreamReq200TooLow
    annotations:
      summary: 'Envoy Percentage of HTTP 200 upstream responses is too low'
      description: "The amount of 200 upstream responses is lower than 99.9%. It is too low"
    expr: >
    rate(envoy_cluster_upstream_rq_200[1m])/rate(envoy_cluster_upstream_rq_completed[1m]) < 0.999

  - alert: IstioEnvoyClusterBindErrors
    annotations:
      summary: "Envoy cluster binding errors"
      description: "Error in binding cluster with {{ $labels.pod_name }}  pod in {{ $labels.namespace }} namespace."
    expr: >
    envoy_cluster_bind_errors > 0

  - alert: IstioEnvoyClusterDstHostInvalid
    annotations:
      summary: "Envoy cluster destination host invalid"
      description: "Envoy cluster destination host {{ $labels.pod_name }} in {{ $labels.namespace }} namespace invalid for 1
minutes"
    expr: > envoy_cluster_original_dst_host_invalid > 0
    for: 1m
```

对接 BLS 实现数据平面日志持久化

概述

本文档帮助用户实现服务网格 CSM 产品对接日志收集与投递服务 BLS，实现对数据面日志的持久化、查询和分析。

前提条件

1. 已开通日志服务产品，您可以登录日志服务控制台确认是否开通。
2. 已创建与 Kubernetes 集群**同地域**的 BLS 日志集，详情请参考：日志集。
3. Kubernetes 集群已安装日志采集组件（CCE Log Operator），详情参考安装日志采集组件。
4. 注意：托管服务网格暂不支持日志持久化查询分析。

操作步骤：

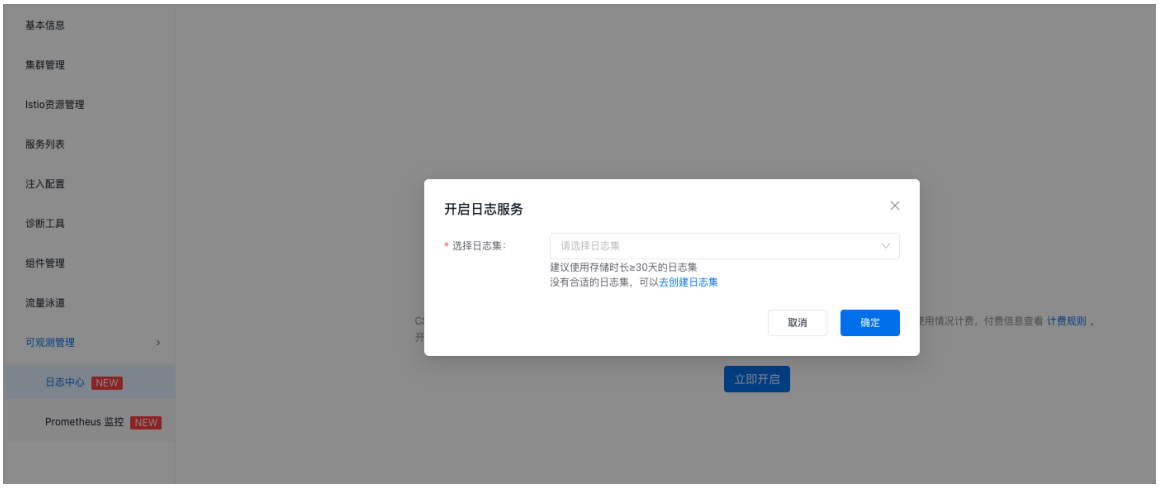
安装日志采集组件 开启网格日志接入 BLS 服务时，如果对应 Kubernetes 集群未安装日志采集组件（CCE Log Operator），可

以按照提示跳转到对应集群的**组件管理**页面。在“组件管理”页面的“日志”页签下，找到 CCE Log Operator 组件，单击**安装**。

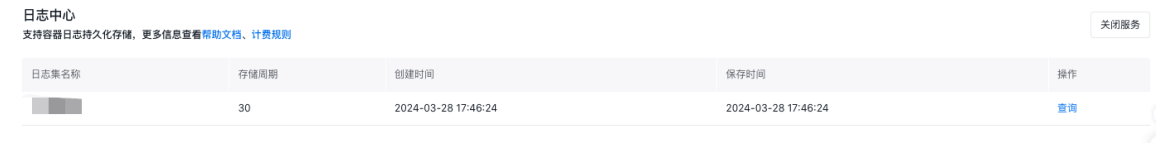
开启数据面日志

- 方式一：在服务网格实例创建时，开启“日志服务”并选择对应的日志集

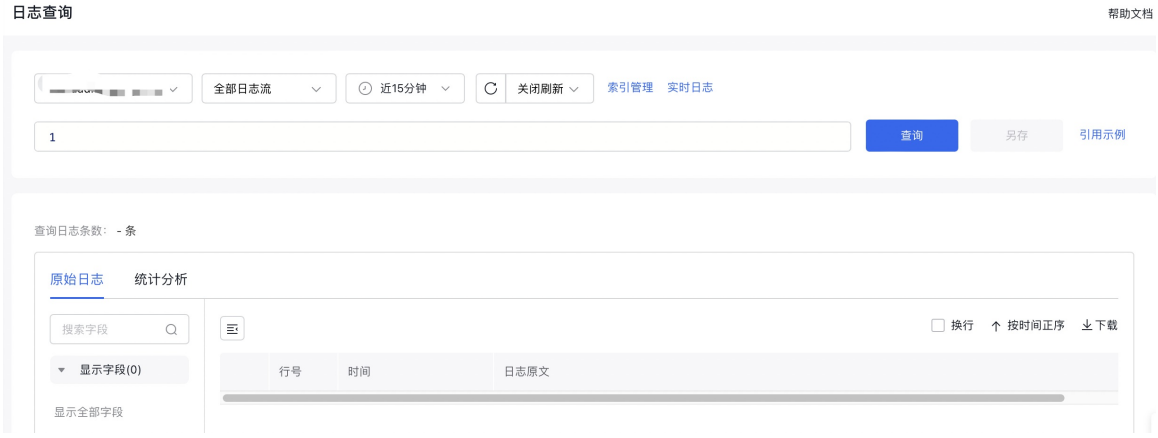
- 方式二：针对已有服务网格，选择**服务网格 > 网格管理**，在**网格管理**页面，单击目标实例名称，然后左侧导航栏选择**可观测管理 > 日志中心**，需要在页面选择开启，并选择对应的 BLS 日志集



查询数据面日志 开启数据面日志后，您通过**可观测管理 > 日志中心页面**，选择查询，跳转至 BLS 日志查询页面。



日志查询用于查询和统计已有日志集的数据内容。日志服务 BLS 提供一套完备的数据查询语法，详情请参见日志查询。



关闭数据面日志 选择**服务网格 > 网格管理**，在**网格管理**页面，单击目标实例名称，然后左侧导航栏选择**可观测管理 > 日志中心**，在页面选择**关闭服务**。 BLS 日志集会停止采集新的网格日志，但之前已经同步的日志信息会一直保留到日志集的存储周期结束。

链路追踪

概述

百度智能云 CSM 数据面支持链路追踪，可以将网格数据面内产生的 tracing 数据上报到 Jaeger/Zipkin 服务，上报后可在 Jaeger/Zipkin 界面查看调用链信息，帮助您理解服务间的依赖关系、分析链路情况或快速排障。本文以 Jaeger 为例介绍链路追踪使用流程。

前提条件

- 已添加 Kubernetes 集群到 CSM 实例。
- 部署 Jaeger 并获取服务地址。
- 已配置采样率与服务地址并开启链路追踪功能。

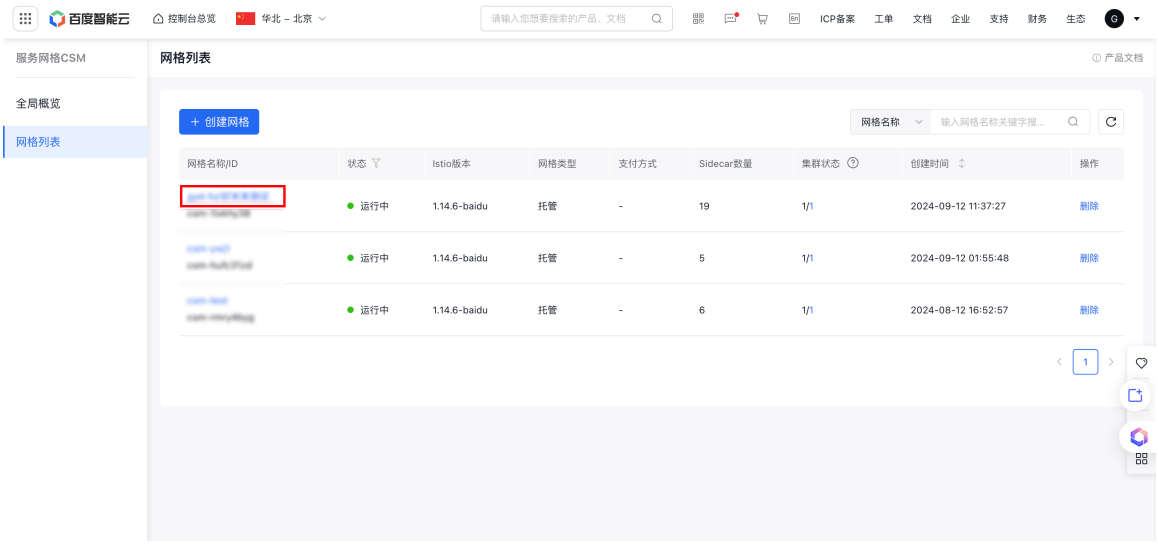
操作步骤

步骤一：为网格实例启用链路追踪

为已有网格实例开启链路追踪

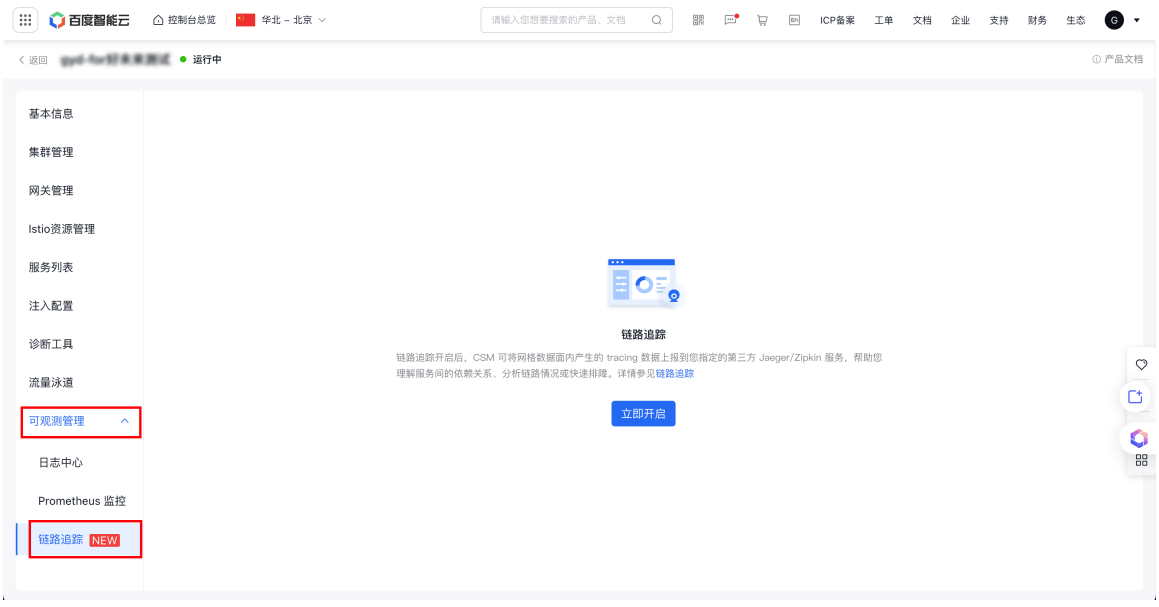
1. 登录 CSM 控制台，左侧选择网格列表进入网格列表页面，点击需要开启链路追踪的**网格实例名称/ID**，进入网格实例详情

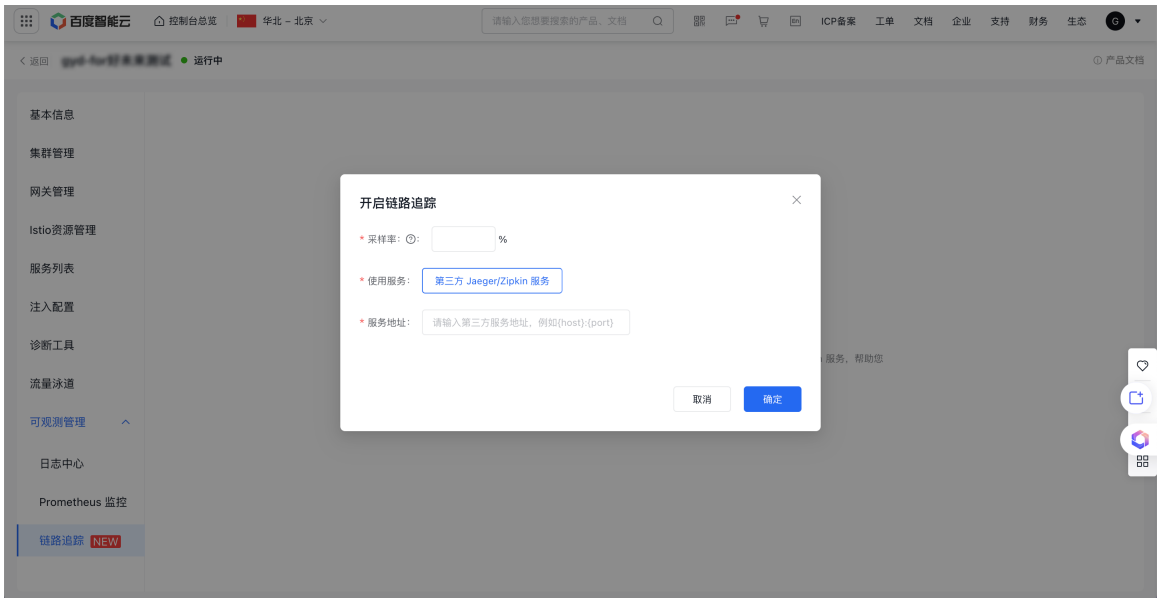
页。



2. 左侧 Tab 选择可观测管理，点击**链路追踪**进入链路追踪详情页。点击**立即开启**，在链路追踪配置弹窗中配置相关参数，点击**确认**即可启用链路追踪，参数及说明如下：

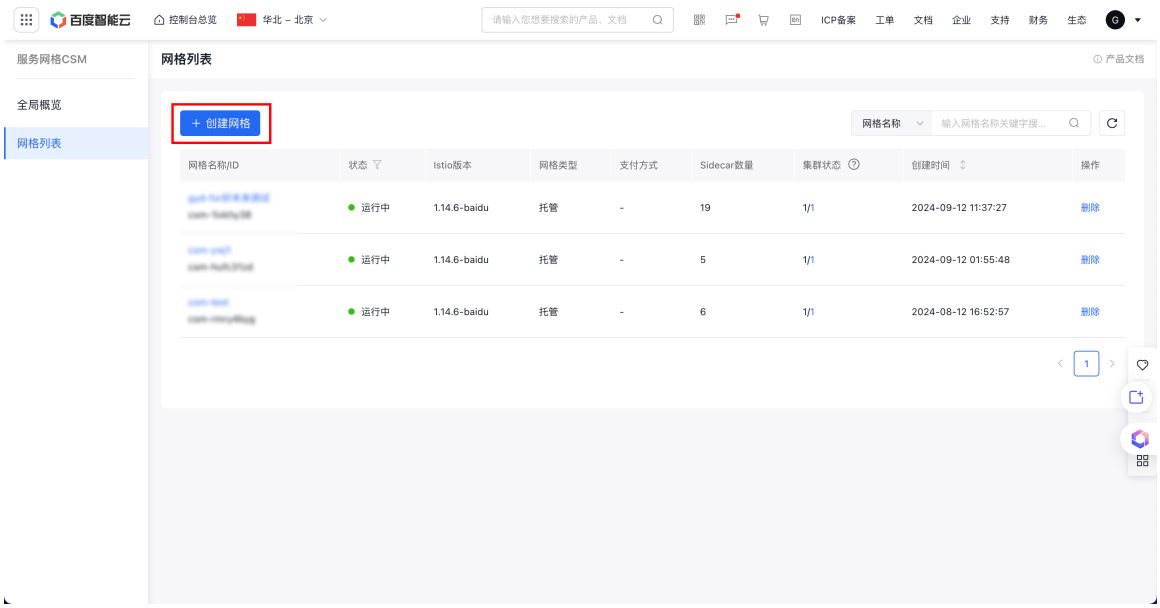
参数	说明
采样率	（必填）网格采集并持久化 tracing 的采样比例，范围 0-100。Sidecar 采集、上报数据消耗的资源与带宽和采样率成正比，请按需配置。
使用服务	（默认）第三方 Jaeger/Zipkin 服务
服务地址	（必填）第三方服务地址，仅支持输入英文字母、数字以及特殊字符（.）、（:）和（-）。**注意：请确保第三方服务地址可达。**



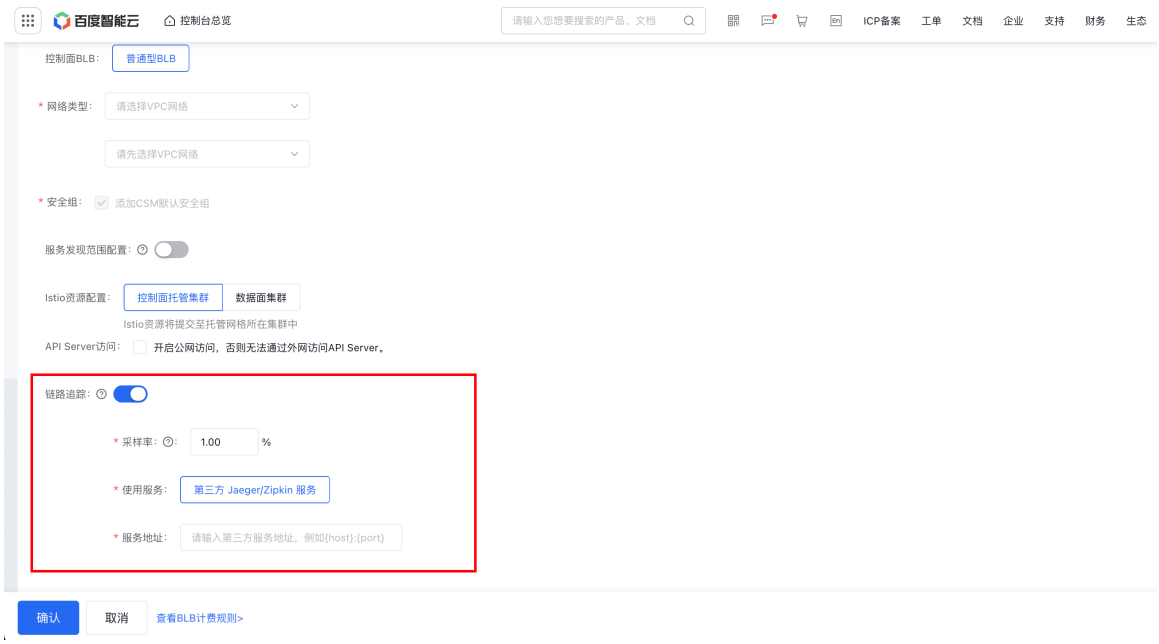


创建网格实例时开启链路追踪

1. 登录 CSM 控制台，左侧选择网格列表进入网格列表页面，点击**创建网格**，进入创建网格页。



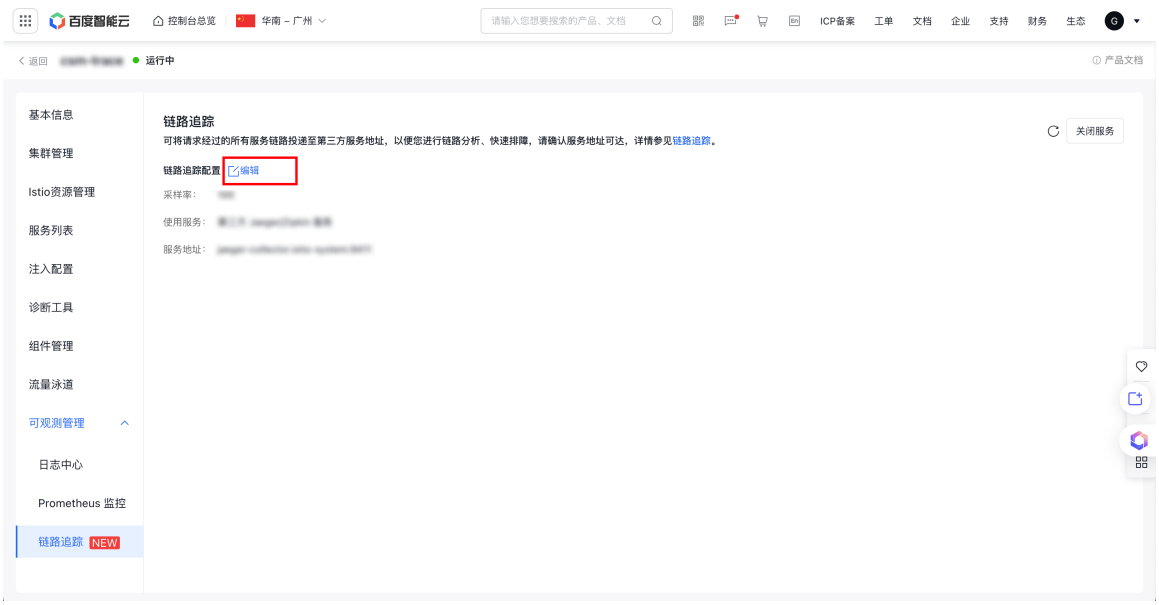
2. 开启**链路追踪开关**，并配置采样率和服务地址，配置完毕后点击**确认**。

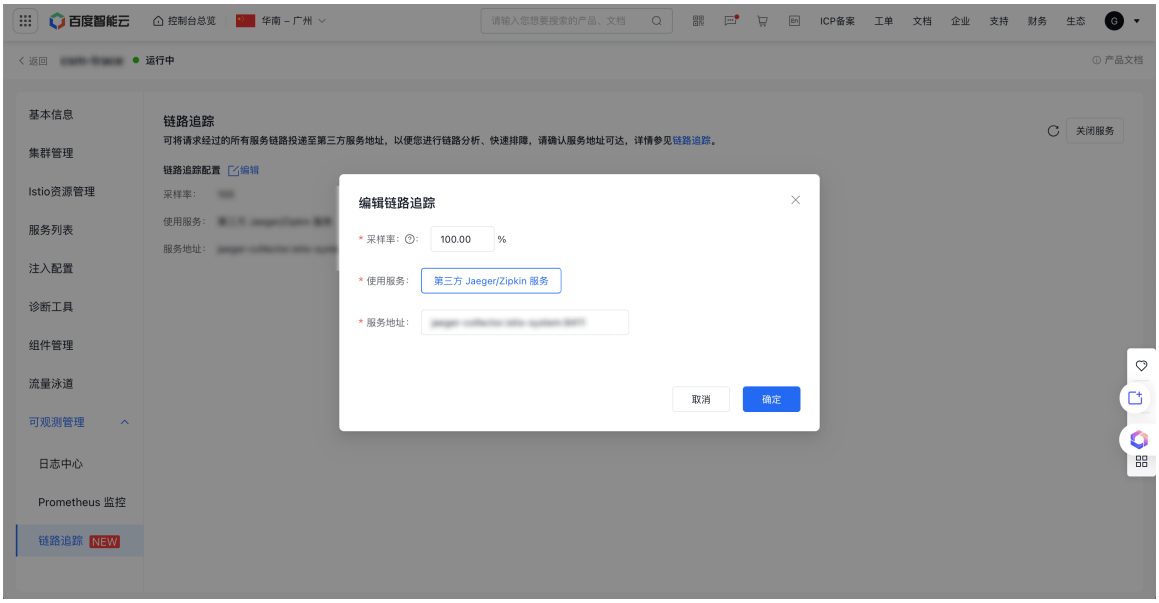


修改链路追踪配置

1. 登录 CSM 控制台，左侧选择网格列表进入网格列表页面，点击需要修改链路追踪配置的**网格实例名称/ID**，进入网格实例详情页。
2. 左侧 Tab 选择可观测管理，点击**链路追踪**进入链路追踪详情页。点击**编辑**按钮，在链路追踪编辑弹窗中可以修改链路追踪相关配置，点击确认后生效。

> **注意：**修改链路追踪配置会导致控制面重启，建议您在系统负载较低时修改。





步骤二：在集群部署 Jaeger 和服务

> **说明：**如果您使用本文档中 YAML 文件部署 Deployment 和 Service，请确保 Deployment 和 Service 均部署在 istio-system 命名空间下。

1. 使用以下示例内容，创建 jaeger.yaml 文件

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jaeger
```

```
  namespace: istio-system
  labels:
    app: jaeger
spec:
  selector:
    matchLabels:
      app: jaeger
  template:
    metadata:
      labels:
        app: jaeger
      annotations:
        sidecar.istio.io/inject: "false"
        prometheus.io/scrape: "true"
        prometheus.io/port: "14269"
    spec:
      containers:
        - name: jaeger
          image: "registry.baidubce.com/csm-offline/all-in-one:1.35"
          env:
            - name: BADGER_EPHEMERAL
              value: "false"
            - name: SPAN_STORAGE_TYPE
              value: "badger"
            - name: BADGER_DIRECTORY_VALUE
              value: "/badger/data"
            - name: BADGER_DIRECTORY_KEY
              value: "/badger/key"
            - name: COLLECTOR_ZIPKIN_HOST_PORT
              value: ":9411"
            - name: MEMORY_MAX_TRACES
              value: "50000"
            - name: QUERY_BASE_PATH
              value: /jaeger
          livenessProbe:
            httpGet:
              path: /
              port: 14269
          readinessProbe:
            httpGet:
              path: /
              port: 14269
          volumeMounts:
            - name: data
              mountPath: /badger
          resources:
            requests:
              cpu: 10m
      volumes:
        - name: data
          emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: tracing
  namespace: istio-system
  labels:
    app: jaeger
spec:
  type: ClusterIP
  ports:
```

```
    - name: http-query
      port: 80
      protocol: TCP
      targetPort: 16686
    # Note: Change port name if you add '--query.grpc.tls.enabled=true'
    - name: grpc-query
      port: 16685
      protocol: TCP
      targetPort: 16685
  selector:
    app: jaeger
---
##### Jaeger implements the Zipkin API. To support swapping out the tracing backend, we use a Service named Zipkin.
apiVersion: v1
kind: Service
metadata:
  labels:
    name: zipkin
  name: zipkin
  namespace: istio-system
spec:
  ports:
    - port: 9411
      targetPort: 9411
      name: http-query
  selector:
    app: jaeger
---
apiVersion: v1
kind: Service
metadata:
  name: jaeger-collector
  namespace: istio-system
  labels:
    app: jaeger
spec:
  type: ClusterIP
  ports:
  - name: jaeger-collector-http
    port: 14268
    targetPort: 14268
    protocol: TCP
  - name: jaeger-collector-grpc
    port: 14250
    targetPort: 14250
    protocol: TCP
  - port: 9411
    targetPort: 9411
    name: http-zipkin
  selector:
    app: jaeger
```

2. 执行以下命令，将该配置应用到数据面集群：

```
$ kubectl apply -f jaeger.yaml
```

3. 部署完毕后，查看 jaeger 服务，确认 pod 和 service 均正常启动：

```
$ kubectl get pods -n istio-system
$ kubectl get svc -n istio-system
```

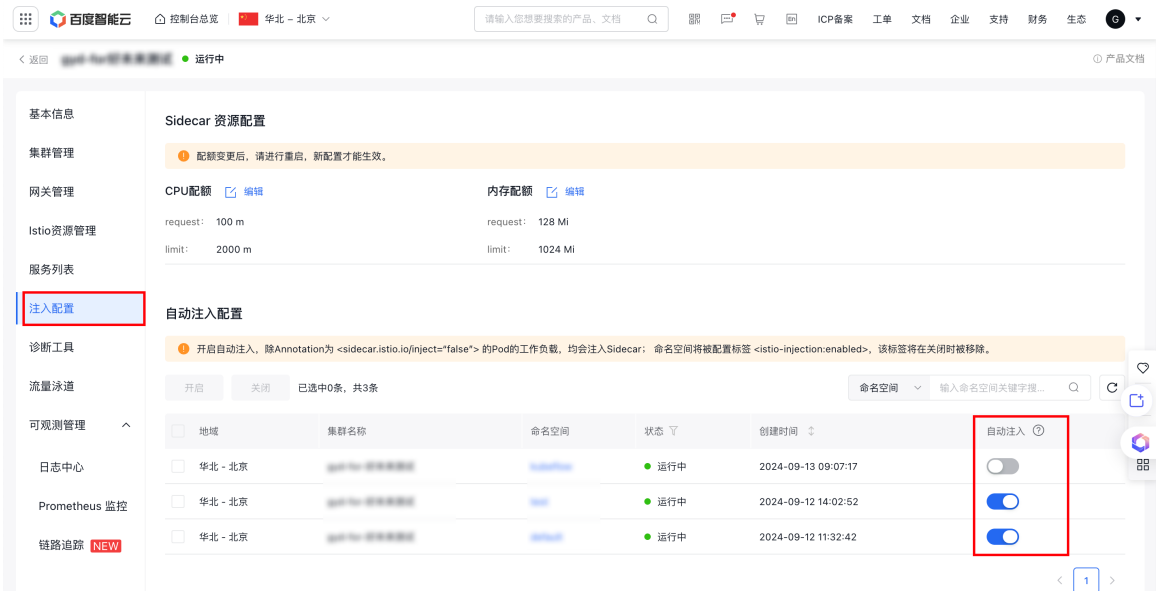
4. 打开 jaeger 页面：

```
$ istioctl dashboard jaeger
```

步骤三：部署微服务应用程序

1. 开启 Sidecar 自动注入

   方式一：登陆 CSM 控制台，选择目标网格实例，选择【注入配置】，在自动注入配置模块选择需要部署的命名空间，点击【开启】即可开启 Sidecar 自动注入，也可通过切换开关状态开启自动注入。



   方式二：使用 kubectl 开启自动注入 Sidecar 功能：

```
# 开启自动注入 Sidecar 功能
$ kubectl label namespace ${namespace} istio-injection=enabled --overwrite
```

2. 执行以下命令，将测试应用 consumer-demo 和 provider-demo 部署到集群中：

```
$ kubectl apply -f consumer-demo-deployment.yaml
$ kubectl apply -f provider-demo-deployment.yaml
```

consumer-demo-deployment.yaml 文件如下：

```yaml
 apiVersion: apps/v1
kind: Deployment
metadata:
 name: consumer-demo-deployment
labels:
  app: consumer-demo
  name: consumer-demo
spec:
 replicas: 1
 selector:
  matchLabels:
     app: consumer-demo
template:
   metadata:
   labels:
      app: consumer-demo
   spec:
   restartPolicy: Always
   containers:
      - name: consumer-demo
      image: registry.baidubce.com/csm-offline/bms-orignal-consumer:dev
      imagePullPolicy: Always
      env:
      - name: OTEL_TRACES_EXPORTER
         value: otlp
      - name: OTEL_EXPORTER_OTLP_ENDPOINT
         value: http://otel-collector.istio-system.svc.cluster.local:4317
      - name: OTEL_JAVAAGENT_ENABLED
         value: 'true'
      ports:
         - containerPort: 9999
      resources:
         limits:
         cpu: 350m
         memory: 612Mi
         requests:
         cpu: 350m
         memory: 612Mi
```

provider-demo-deployment-v1.yaml 文件如下：

```
 apiVersion: apps/v1
kind: Deployment
metadata:
name: provider-demo-deployment
labels:
    app: provider-demo
    name: provider-demo
spec:
replicas: 1
selector:
    matchLabels:
    app: provider-demo
template:
    metadata:
    labels:
        app: provider-demo
    spec:
    restartPolicy: Always
    containers:
        - name: provider-demo
        image: registry.baidubce.com/csm-offline/bms-orignal-provider:dev
        imagePullPolicy: Always
        env:
        - name: OTEL_TRACES_EXPORTER
            value: otlp
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
            value: http://otel-collector.istio-system.svc.cluster.local:4317
        - name: OTEL_JAVAAGENT_ENABLED
            value: 'true'
        ports:
            - containerPort: 10001
        resources:
            limits:
            cpu: 350m
            memory: 612Mi
            requests:
            cpu: 350m
            memory: 612Mi
```

3. 执行以下命令，部署应用的 Service：

```
 $ kubectl apply -f consumer-demo-service.yaml
$ kubectl apply -f provider-demo-service.yaml
```

consumer-demo-service.yaml 文件如下：

```
  apiVersion: v1
kind: Service
metadata:
  name: consumer-demo
  annotations:
    prometheus.io/scrape: "true"
spec:
  selector:
    app: consumer-demo
  type: ClusterIP
  sessionAffinity: None
  ports:
  - protocol: TCP
    port: 9999
    targetPort: 9999
    name: http-port
```

provider-demo-service.yaml 文件如下：

```
  apiVersion: v1
kind: Service
metadata:
  name: provider-demo
  annotations:
    prometheus.io/scrape: "true"
spec:
  selector:
    app: provider-demo
  type: ClusterIP
  sessionAffinity: None
  ports:
  - protocol: TCP
    port: 80
    targetPort: 10001
    name: http-port
```

步骤四：访问微服务应用产生追踪数据 进入 consumer-demo 的 pod 容器中，访问接口 ：

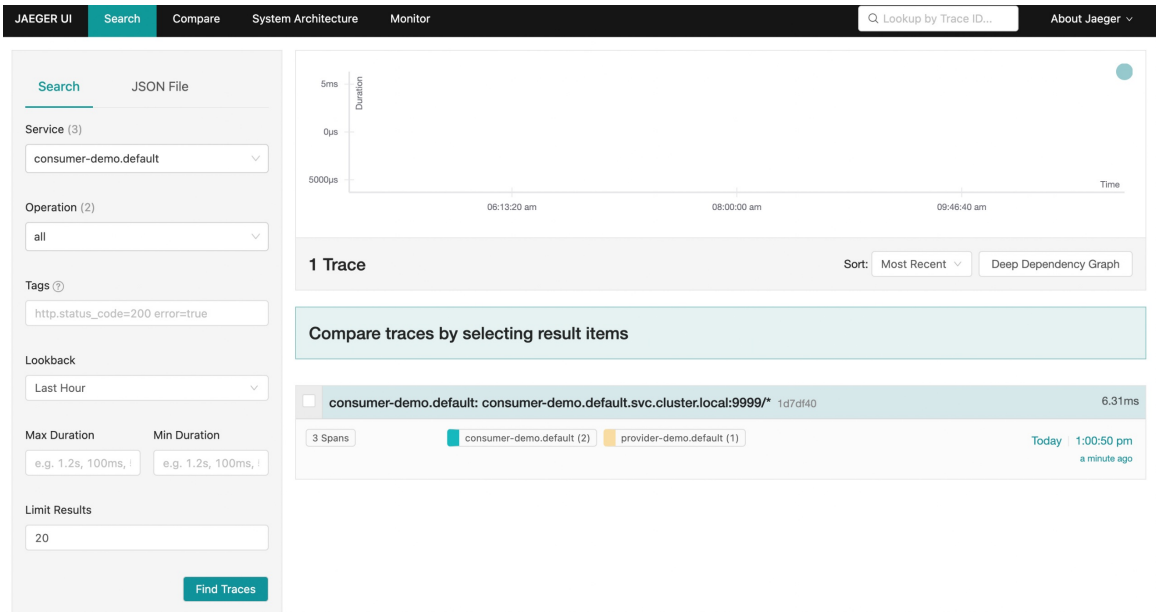
```
$ while true; do curl http://{consumer-demo-pod-ip}:9999/echo-rest/test && echo && sleep 1; done;
```

步骤五：查看链路追踪数据

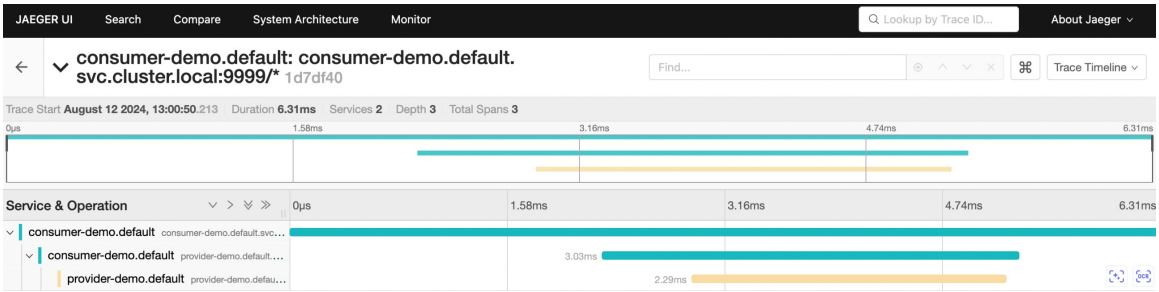
1. 执行以下命令，打开 jaeger 页面：

```
$ istioctl dashboard jaeger
```

2. 查看 tracing 采集数据，预期输出如下：



3. 点击具体某条访问链路页面查询效果如下：



排查步骤 若观察到数据不符合预期，可按以下步骤进行排查：

1. 确认是否成功安装 jaeger。

   通过以下命令查看 jaeger 的 pod 和 service 是否均正常启动：

   ```
   $ kubectl get pods -n istio-system
   $ kubectl get svc -n istio-system
   ```

2. 确认 jaeger 服务地址及端口是否正确。

   通过以下命令查看您在链路追踪所配置的服务地址与 jaeger-collector 服务地址及端口是否一致：

   ```
   $ kubectl get svc -n istio-system | grep jaeger-collector
   ```

3. 确认您部署的微服务应用程序 pod 是否成功注入 sidecar。

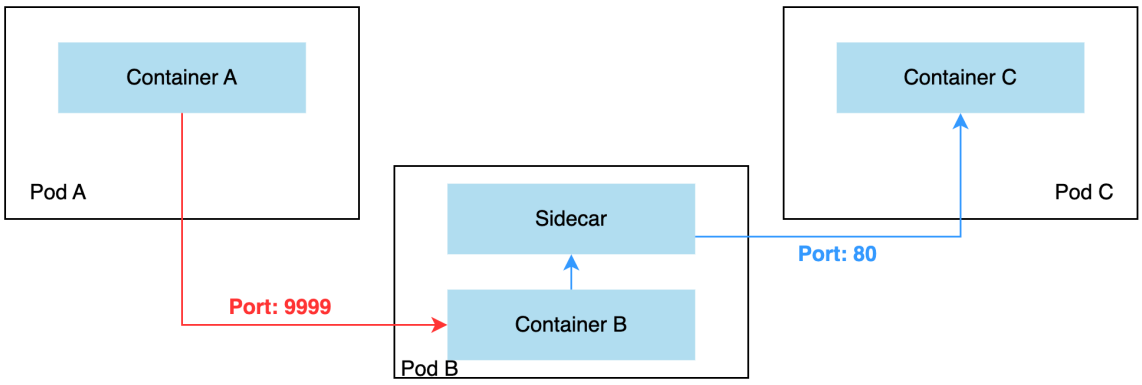
   ```
   $ kubectl get pod -n  ${namespace}
   ```

灵活控制流量免于经过Sidecar代理

功能介绍

Sidecar 代理主要用于管理服务间流量、网络安全性、可观测性。默认情况下，Sidecar 代理会全部拦截进出 Pod 的流量。然而，在一些特定场景下，有需求灵活设置 Pod 中某些端口进出流量是否经过 Sidecar 代理。

本文将以配置所有端口的入站流量不经过 Sidecar 代理（sidecar-include-inbound-ports=""），而 80 端口的出站流量经过 Sidecar 代理（sidecar-include-outbound-ports="80"）为例，展示如何实现特定的入站流量免于经过 Sidecar 代理，特定的出站流量经过 Sidecar 代理。



注解说明

下述注解作用在命名空间级别，注解说明如下：

- sidecar-exclude-inbound-ports：在命名空间级别上配置端口 使配置的端口入口流量不经过 Sidecar 代理，其他端口入口流量将继续通过 Sidecar 代理处理。若为多个端口，可以设置为一个以逗号分隔的端口列表，如：sidecar-exclude-inbound-ports: "9999"
- sidecar-include-outbound-ports：在命名空间级别上配置端口使 配置的特定出口流量经过 Sidecar 代理，其他出口流量不经过 Sidecar 代理。若为多个端口，可以设置为一个以逗号分隔的端口列表，如：sidecar-include-outbound-ports: "80,8099"
- sidecar-include-inbound-ports：在命名空间级别上配置端口使 配置的特定入口流量经过 Sidecar 代理，其他出口流量不经过 Sidecar 代理。若为多个端口，可以设置为一个以逗号分隔的端口列表，如：sidecar-include-inbound-ports: "80,8099"
  - 如果想让所有的端口入口流量都不经过 Sidecar，可以设置为 sidecar-include-inbound-ports=""，这样所有的流量都会绕过 Sidecar 代理。

> 注意：因为上述配置为命名空间级别，请谨慎配置。
>
> 如果流量链路上涉及的上下游应用开启了 Sidecar 注入，不能配置 Client 某端口出口流量经过 Sidecar 代理，而 Server 端入口流量不经过 Sidecar 代理，该场景会出现 TLS 报错。 举个例子：如设置 Client 端 sidecar-include-outbound-ports="80" ，设置了 Server 端 sidecar-include-inbound-ports=""，请求会报错查看日志显示 TLS error: 268435703:SSL routines:OPENSSL_internal:WRONG_VERSION_NUMBER

前提条件

- 已创建CSM网格实例。具体操作，请参见CSM实例管理。
- 已添加CCE集群到CSM实例。
- 已开启目标命名空间 sidecar 自动注入。

操作步骤

添加命名空间级别注解

为目标命名空间添加以下注解，以配置所有端口的入站流量不经过 Sidecar 代理，同时确保 80 端口的出站流量经过 Sidecar 代理：

```
 kubectl annotate ns default sidecar-include-inbound-ports=""
kubectl annotate ns default sidecar-include-outbound-ports="80"
```

> 注意⚠ 如果命名空间下已有 Pod，需要重启这些 Pod 以使配置生效。此配置在命名空间级别全局生效，配置错误将导致该命名空间下所有 Pod 在新建或重启时出现异常。

部署应用

1. 通过 CSM > 集群管理 跳转到 CCE 集群，进行应用部署。

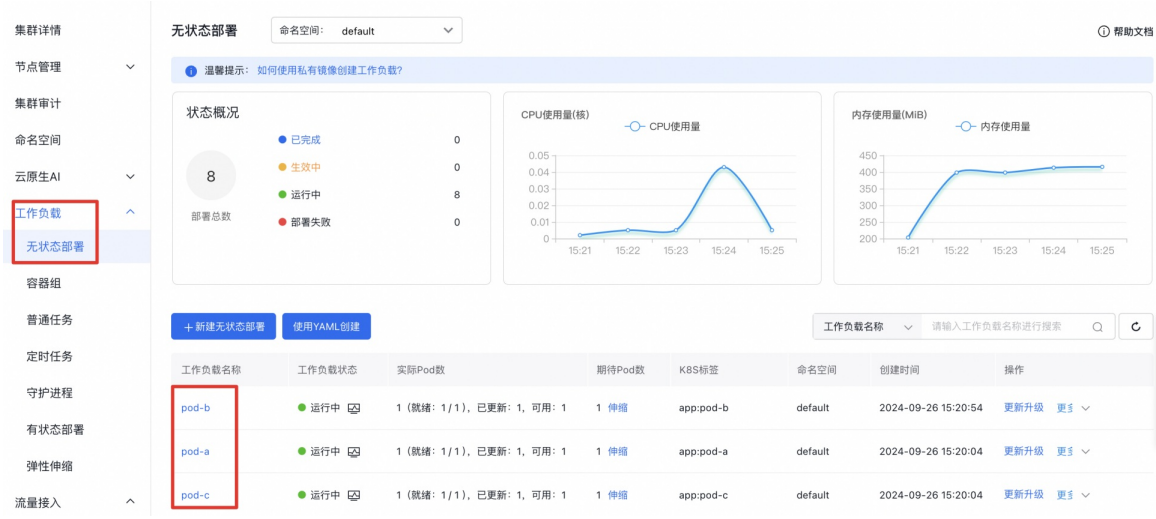
2. 点击 “工作负载 > 无状态部署 > 使用YAML部署”，部署测试应用。

测试Yaml文件如下所示：

```
 apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-a
  labels:
    app: pod-a
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pod-a
  template:
    metadata:
      labels:
        app: pod-a
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
      containers:
      - name: pod-a
        image: registry.baidubce.com/csm-offline/helloworld:v1
        resources:
          requests:
            cpu: "100m"
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 5000
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-b
  labels:
    app: pod-b
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pod-b
  template:
    metadata:
      labels:
```

```
    labels:
      app: pod-b
  spec:
    restartPolicy: Always
    containers:
      - name: pod-b
        image: registry.baidubce.com/csm-offline/bms-orignal-consumer:dev
        imagePullPolicy: Always
        ports:
          - containerPort: 9999
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-c
  labels:
    app: pod-c
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pod-c
  template:
    metadata:
      labels:
        app: pod-c
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
      restartPolicy: Always
      containers:
        - name: pod-c
          image: registry.baidubce.com/csm-offline/bms-orignal-provider:dev
          imagePullPolicy: Always
          ports:
            - containerPort: 10001
```

工作负载如下所示：

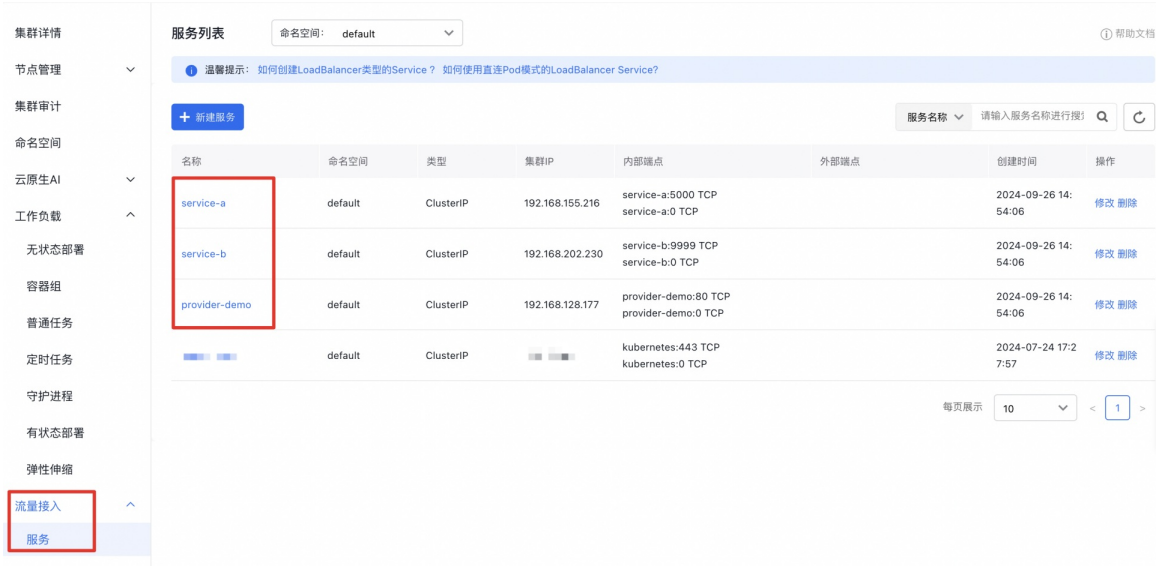


3. 点击 “流量接入 > 服务 > 使用YAML部署”，部署测试应用的服务。

测试Yaml文件如下所示：

```
  apiVersion: v1
kind: Service
metadata:
  name: service-a
  labels:
    app: pod-a
spec:
  ports:
  - port: 5000
    name: http
  selector:
    app: pod-a
---
apiVersion: v1
kind: Service
metadata:
  name: service-b
  annotations:
    prometheus.io/scrape: "true"
spec:
  selector:
    app: pod-b
  type: ClusterIP
  sessionAffinity: None
  ports:
  - protocol: TCP
    port: 9999
    targetPort: 9999
    name: http-port
---
apiVersion: v1
kind: Service
metadata:
  name: provider-demo
  annotations:
    prometheus.io/scrape: "true"
spec:
  selector:
    app: pod-c
  type: ClusterIP
  sessionAffinity: None
  ports:
  - protocol: TCP
    port: 80
    targetPort: 10001
    name: http-port
```

服务 service 如下所示：



流量验证

进入 pod-b pod 中 istio-proxy 容器，执行下述命令开启 pod-b 容器的 Sidecar debug 日志：

```
curl -s -X POST 127.1:15000/logging\?level\=debug
```

进入 pod-a 容器中执行以下命令，并查看 pod-b 的 Sidecar debug 日志。

```
curl http://<pod-b-podIP>:9999/echo-rest/hello && echo
```

预期输出

流量正常通过，且 Sidecar 日志中仅记录 80 端口的出站流量，而没有入站端口的流量。

> 请注意，当 Pod 单独设置了注解 traffic.sidecar.istio.io/excludeInboundPorts、traffic.sidecar.istio.io/includeInboundPorts 或 traffic.sidecar.istio.io/includeOutboundPorts，且这些注解值与命名空间的注解值不一致时，Pod 的注解将优先于命名空间的注解。

使用 Wasm Filter 扩展数据面

概述

WebAssembly（简称 Wasm）是一种高效的二进制代码格式，它允许开发者将编写的指令集加载至 Envoy 过滤器链中，从而扩展服务网格的数据面功能。这种方法实现了 Envoy 核心与扩展组件之间解耦，避免用户为了扩展功能而需要修改 Envoy 源代码或编译定制版本的 Envoy。此外，Wasm 还带来了动态加载和运行时安全隔离等显著优势。本文介绍如何使用 Wasm Filter 扩展服务网格数据面的能力。

步骤一：构建 wasm 文件并挂载至 configmap 中

1. 以下采用名为 wasm-example-filter.wasm 的 wasm 文件举例。
2. 创建 configmap，将 wasm filter 存储到 configmap 中。

```
#### 创建名为 wasm-example-filter 的 configmap，将 wasm-example-filter.wasm 注入到 configmap 中
kubectl create cm wasm-example-filter --from-file=wasm-example-filter.wasm
```

步骤二：命名空间级别 wasm 注入

1. 通过 configmap 将 wasm filter 挂载到工作负载上。
2. 服务网格 CSM 提供命名空间级别的 wasm 注入，可以通过为命名空间添加标签 `sidecar-injection-wasm-configmap: {configmap 文件名}` 的方式开启命名空间注入 wasm 文件，该功能会将指定 configmap 中的内容挂载到指定命名空间的所有工作负载中。即挂载到 **istio-proxy** 容器的 **/var/local/lib/wasm-filters** 目录下。
3. Pod 重新创建后，在 istio-proxy 的 /var/local/lib/wasm-filters 目录下可以查看到 wasm-example-filter.wasm 文件。

```
#### 为 test 命名空间开启 wasm 注入，将名为 wasm-example-filter 的 configmap 挂载到 sidecar 容器中
kubectl label namespace test sidecar-injection-wasm-configmap=wasm-example-filter
```

> **说明：** 正在运行中的 Pod 重启后才会生效。

步骤三：创建 EnvoyFilter

1. 目前已经成功将编译后的 wasm 文件挂载到了 Envoy 中，但此时 Envoy 并不知道这个文件是用来做什么的。因此，需要创建一个名为 EnvoyFilter 的 CRD 资源对象，将 wasm filter 添加到对应工作负载的 envoy filter chain 中，使其生效。
2. 比如可以使用下面的配置来将 wasm 插件挂载到 Envoy 中：

```
#### my-envoyfilter.yaml
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: my-envoyfilter
spec:
  workloadSelector:
    labels:
      app: app1 # 表示只对 app1 这个 Pod 进行过滤
  configPatches: # 用于配置 Envoy 的过滤器
    - applyTo: HTTP_FILTER
      match: # 用于匹配 Envoy 的过滤器
        context: SIDECAR_INBOUND # 仅对入站流量进行过滤
        listener:
          filterChain:
            filter:
              name: envoy.filters.network.http_connection_manager
              subFilter:
                name: envoy.filters.http.router
      patch: # 用于指定要挂载的 wasm 插件
        operation: INSERT_BEFORE # 在 router 之前插入
        value:
          name: mydummy
          typed_config:
            "@type": type.googleapis.com/udpa.type.v1.TypedStruct
            type_url: type.googleapis.com/envoy.extensions.filters.http.wasm.v3.Wasm
            value:
              config: # 指定了插件的配置信息以及 wasm 插件的路径
                configuration:
                  "@type": type.googleapis.com/google.protobuf.StringValue
                  value: dummy
                root_id: "regex_replace"
                vm_config:
                  code:
                    local:
                      filename: var/local/lib/wasm-filters/wasm-example-filter.wasm
                  runtime: envoy.wasm.runtime.v8
                  vm_id: myvmdummy
```

3. 应用上面的资源对象：

```
kubectl apply -f httpbin-wasm-filter.yaml
```

4. 至此，wasm filter 部署完成。

运维指南

常见问题

常见调试工具

1. 查看控制平面debug级别的日志，编辑 deploy istiod --log_output_level=default:debug
2. 更改业务 Pod 中的 istio-proxy 日志等级，查看 envoy 的日志

```
#### 1. 修改日志级别（包含none、error、warn、info、debug）
istioctl -n namespace proxy-config log <pod> --level 日志级别

#### 2. 变更 envoy 中日志级别
kubectl -n default exec -it  pod名称 -c istio-proxy bash
#### 变更 envoy 中的所有组件的日志级别
curl -s -X POST 127.1:15000/logging?level=trace
#### 变更 envoy 中的 lua 日志级别
curl -s -X POST 127.1:15000/logging?lua=debug
```

3. 查看 Envoy 的 xds 配置

```
#### 在 istio-proxy 容器中
curl localhost:15000/config_dump > config_dump
```

4. 获取 istiod 控制平面中的所有的 CRD 信息

```
curl -s 127.1:15014/debug/configz
```

5. 获取控制平面的内部状态和指标信息

```
curl -s 127.1:15014/metrics
```

6. 获取数据平面 Envoy 收取到服务发现的实例信息及状态指标

```
curl -s 127.1:15000/clusters
```

配置完 DestinationRule 后出现503

排查：

可以查看是否由于 DestinationRule 与全局的 TLS 策略有冲突导致，此错误只有在安装期间禁用了自动双向 TLS 时才会看到。

使用注意事项：

任何时候应用 DestinationRule，都要确保 trafficPolicy TLS mode 和全局的配置一致。

控制平面出现 pilot_eds_no_instances 指标

原因：

可能是 service selector 标签没有匹配到相关联的 pod，或者 service 与之关联的 pod 没有启动。

排查：

可以排查 pod 的标签与 Service selector 字段是否匹配，DestinationRule 子集标签是否配置正确，或者查看 ep 是否关联了这个 pod。

案例：

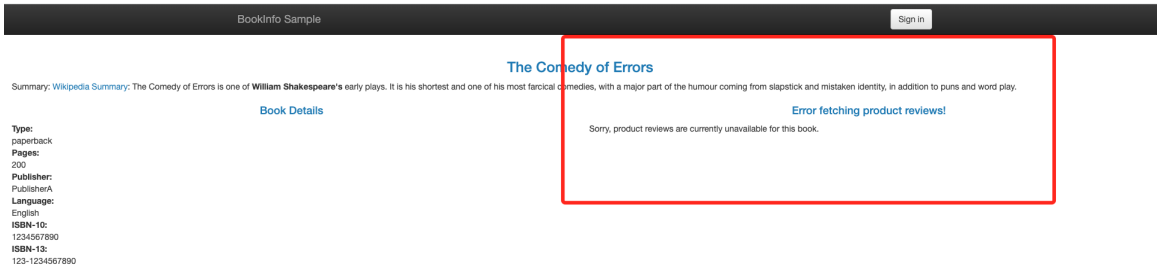
展示 istioctl ps 如下所示：

手动编辑 DestinationRule review name v1 的 version 为 v1-bug。

```
→  istio-1.15.0 istioctl   --context=tanjunchen-istio -n default ps
NAME                                                   CLUSTER        CDS        LDS        EDS        RDS          ECDS         ISTIOD
          VERSION
details-v1-7d88846999-wtjct.default                    Kubernetes     SYNCED     SYNCED     SYNCED     SYNCED       NOT SENT     istiod-cdb755b6
9-7x4tl     1.15.0
istio-egressgateway-79c57f9f64-b49zs.istio-system      Kubernetes     SYNCED     SYNCED     SYNCED     NOT SENT     NOT SENT     istiod-cdb755b6
9-7x4tl     1.15.0
istio-ingressgateway-fd7c86d97-4bbsp.istio-system      Kubernetes     SYNCED     SYNCED     SYNCED     SYNCED       NOT SENT     istiod-cdb755b6
9-7x4tl     1.15.0
productpage-v1-7795568889-npt8f.default                Kubernetes     SYNCED     SYNCED     SYNCED     SYNCED       NOT SENT     istiod-cdb755b6
9-7x4tl     1.15.0
ratings-v1-754f9c4975-cvp2h.default                    Kubernetes     SYNCED     SYNCED     SYNCED     SYNCED       NOT SENT     istiod-cdb755b6
9-7x4tl     1.15.0
reviews-v1-55b668fc65-vhdsf.default                    Kubernetes     SYNCED     SYNCED     SYNCED     SYNCED       NOT SENT     istiod-cdb755b6
9-7x4tl     1.15.0
reviews-v2-858f99c99-kdlcj.default                     Kubernetes     SYNCED     SYNCED     SYNCED     SYNCED       NOT SENT     istiod-cdb755b6
9-7x4tl     1.15.0
reviews-v3-7886dd86b9-zs89p.default                    Kubernetes     SYNCED     SYNCED     SYNCED     SYNCED       NOT SENT     istiod-cdb755b6
9-7x4tl     1.15.0
→  istio-1.15.0
```

```
→  istio-1.15.0 kubectl --context=tanjunchen-istio -n default get dr  reviews -oyaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"networking.istio.io/v1alpha3","kind":"DestinationRule","metadata":{"ann
},"spec":{"host":"reviews","subsets":[{"labels":{"version":"v1"},"name":"v1"},{"labels":{"ve
},"name":"v3"}]}}
  creationTimestamp: "2022-11-01T11:29:00Z"
  generation: 2
  name: reviews
  namespace: default
  resourceVersion: "1693814"
  uid: 8344c49d-5df1-4209-8e1f-c089f53dd426
spec:
  host: reviews
  subsets:
  - labels:
      version: v1-bug
    name: v1
  - labels:
      version: v2
    name: v2
  - labels:
      version: v3
    name: v3
→  istio-1.15.0
```

重新从浏览器打开页面，review 页面出现报错。



快速定位：

1、查看 istiod 控制平面的日志，出现 pilot_eds_no_instances 异常指标。

```
2022-11-01T11:38:23.559860Z     info    ads     LDS: PUSH for node:istio-ingressgateway-fd7c86d97-4bbsp.istio-system resources:1 size:3.7kB
2022-11-01T11:38:23.559998Z     info    ads     RDS: PUSH for node:istio-ingressgateway-fd7c86d97-4bbsp.istio-system resources:1 size:2.4kB cac
hed:0/0
2022-11-01T11:38:23.562386Z     info    ads     RDS: PUSH for node:reviews-v3-7886dd86b9-zs89p.default resources:9 size:11.2kB cached:9/9
2022-11-01T11:38:27.233981Z     info    ads     Push Status: {
    "pilot_eds_no_instances": {
        "outbound|9080|v1|reviews.default.svc.cluster.local": {}
    }
}
2022-11-01T11:40:41.508818Z     info    ads     Push debounce stable[38] 1 for config Secret/kube-system/cce-plugin-token: 100.211634ms since l
ast change, 100.211409ms since last push, full=false
2022-11-01T11:40:41.508917Z     info    ads     XDS: Incremental Pushing:2022-11-01T11:38:23Z/20 ConnectedEndpoints:8 Version:2022-11-01T11:38:
23Z/20
```

2、查看 productpage Pod 中的业务 istio-proxy 日志。

```
.2.1.35:9080" inbound|9080|| 127.0.0.6:50724 10.2.1.35:9080 192.168.0.16:0 outbound_.9080_._.productpage.default.svc.cluster.local default
[2022-11-01T11:42:50.855Z] "GET /details/0 HTTP/1.1" 200 - via_upstream - "-" 0 178 2 2 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Ap
pleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36" "ef4458a8-665d-9cac-b155-e301a9282498" "details:9080" "10.2.0.29:9080" out
bound|9080|v1|details.default.svc.cluster.local 10.2.1.35:34480 172.16.201.57:9080 10.2.1.35:33104 - -
[2022-11-01T11:42:50.860Z] "GET /reviews/0 HTTP/1.1" 503 UH no_healthy_upstream - "-" 0 19 0 - "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_1
5_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36" "ef4458a8-665d-9cac-b155-e301a9282498" "reviews:9080" "-" outbound|
9080|v1|reviews.default.svc.cluster.local - 172.16.83.239:9080 10.2.1.35:52642 - -
[2022-11-01T11:42:50.863Z] "GET /reviews/0 HTTP/1.1" 503 UH no_healthy_upstream - "-" 0 19 0 - "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_1
5_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36" "ef4458a8-665d-9cac-b155-e301a9282498" "reviews:9080" "-" outbound|
9080|v1|reviews.default.svc.cluster.local - 172.16.83.239:9080 10.2.1.35:52644 - -
[2022-11-01T11:42:50.851Z] "GET /productpage HTTP/1.1" 200 - via_upstream - "-" 0 3889 13 13 "192.168.0.16" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36" "ef4458a8-665d-9cac-b155-e301a9282498" "106.13.58.151" "10
.2.1.35:9080" inbound|9080|| 127.0.0.6:52888 10.2.1.35:9080 192.168.0.16:0 outbound_.9080_._.productpage.default.svc.cluster.local default
2022-11-01T11:50:19.170860Z     info    xdsproxy        connected to upstream XDS server: istiod.istio-system.svc:15012
```

3、进入到 productpage Pod 中的 istio-proxy 中，查看 clusters。可以查看到 envoy 没有 reviews v1 相关的 cluster，v2/v3 而是有相关的 cluster。

```
curl localhost:15000/clusters | grep reviews
```

```
istio-proxy@productpage-v1-7795568889-npt8f:/$ curl localhost:15000/clusters | grep 'outbound|9080|v1|reviews.default.svc.cluster'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 91898    0 91898    0     0  64.7M      0 --:--:-- --:--:-- --:--:-- 87.6M
outbound|9080|v1|reviews.default.svc.cluster.local::observability_name::outbound|9080|v1|reviews.default.svc.cluster.local
outbound|9080|v1|reviews.default.svc.cluster.local::default_priority::max_connections::4294967295
outbound|9080|v1|reviews.default.svc.cluster.local::default_priority::max_pending_requests::4294967295
outbound|9080|v1|reviews.default.svc.cluster.local::default_priority::max_requests::4294967295
outbound|9080|v1|reviews.default.svc.cluster.local::default_priority::max_retries::4294967295
outbound|9080|v1|reviews.default.svc.cluster.local::high_priority::max_connections::1024
outbound|9080|v1|reviews.default.svc.cluster.local::high_priority::max_pending_requests::1024
outbound|9080|v1|reviews.default.svc.cluster.local::high_priority::max_requests::1024
outbound|9080|v1|reviews.default.svc.cluster.local::high_priority::max_retries::3
outbound|9080|v1|reviews.default.svc.cluster.local::added_via_api::true
```

4、查看 productpage Pod 中的 istio-proxy 的 config_dump 文件。

```
kubectl -n default exec -it productpage-v1-7795568889-npt8f -c istio-proxy --  curl localhost:15000/config_dump > config_dump
```

5、使用 istioctl 工具，pc、ps 查看 all、bootstrap、cluster、endpoint、listener、log、route、secret 等 envoy 运行时的参数。

```
➜  istio-1.15.0 istioctl --context=tanjunchen-istio pc endpoint productpage-v1-7795568889-npt8f.default  | grep reviews
10.2.0.32:9080                                          HEALTHY     OK                outbound|9080||reviews.default.svc.cluster.local
10.2.0.33:9080                                          HEALTHY     OK                outbound|9080|v2|reviews.default.svc.cluster.local
10.2.0.33:9080                                          HEALTHY     OK                outbound|9080||reviews.default.svc.cluster.local
10.2.0.34:9080                                          HEALTHY     OK                outbound|9080|v3|reviews.default.svc.cluster.local
10.2.0.34:9080                                          HEALTHY     OK                outbound|9080||reviews.default.svc.cluster.local
➜  istio-1.15.0 istioctl --context=tanjunchen-istio pc cluster productpage-v1-7795568889-npt8f.default  | grep reviews
reviews.default.svc.cluster.local                       9080      -               outbound      EDS             reviews.default
reviews.default.svc.cluster.local                       9080      v1              outbound      EDS             reviews.default
reviews.default.svc.cluster.local                       9080      v2              outbound      EDS             reviews.default
reviews.default.svc.cluster.local                       9080      v3              outbound      EDS             reviews.default
➜  istio-1.15.0 istioctl --context=tanjunchen-istio pc route productpage-v1-7795568889-npt8f.default  | grep reviews
80                                                                      reviews.default.svc.cluster.local                 /*                    revi
ews.default
9080                                                                    reviews, reviews.default + 1 more...              /*                    revi
ews.default
➜  istio-1.15.0 istioctl --context=tanjunchen-istio pc  bootstrap productpage-v1-7795568889-npt8f.default  | grep reviews
➜  istio-1.15.0
```

v1?

编辑 Destinationrule reviews 恢复正常。

```
➜  istio-1.15.0 istioctl --context=tanjunchen-istio pc endpoint productpage-v1-7795568889-npt8f.default  | grep reviews
10.2.0.32:9080                                          HEALTHY     OK                outbound|9080|v1|reviews.default.svc.cluster.local
10.2.0.32:9080                                          HEALTHY     OK                outbound|9080||reviews.default.svc.cluster.local
10.2.0.33:9080                                          HEALTHY     OK                outbound|9080|v2|reviews.default.svc.cluster.local
10.2.0.33:9080                                          HEALTHY     OK                outbound|9080||reviews.default.svc.cluster.local
10.2.0.34:9080                                          HEALTHY     OK                outbound|9080|v3|reviews.default.svc.cluster.local
10.2.0.34:9080                                          HEALTHY     OK                outbound|9080||reviews.default.svc.cluster.local
➜  istio-1.15.0 istioctl --context=tanjunchen-istio pc route productpage-v1-7795568889-npt8f.default  | grep reviews
80                                                                      reviews.default.svc.cluster.local                 /*                    revi
ews.default
9080                                                                    reviews, reviews.default + 1 more...              /*                    revi
ews.default
➜  istio-1.15.0 istioctl --context=tanjunchen-istio pc cluster productpage-v1-7795568889-npt8f.default  | grep reviews
reviews.default.svc.cluster.local                       9080      -               outbound      EDS             reviews.default
reviews.default.svc.cluster.local                       9080      v1              outbound      EDS             reviews.default
reviews.default.svc.cluster.local                       9080      v2              outbound      EDS             reviews.default
reviews.default.svc.cluster.local                       9080      v3              outbound      EDS             reviews.default
➜  istio-1.15.0 istioctl --context=tanjunchen-istio pc  bootstrap productpage-v1-7795568889-npt8f.default  | grep reviews
➜  istio-1.15.0
```

控制平面出现 pilot_conflict_inbound_listener 指标

原因：

该指标是由于协议冲突导致的 。

排查：

重点排查 k8s Service 资源，是否存在相同端口上配置了不同协议的情况。

灰度发布场景中，未按照预期配置进行流量治理

排查：

1. 可以排查 VirtualService 的 host 字段是否配置错误，match 字段的匹配规则是否错误，和关于子集的配置是否错误，还有 DestinationRule 下标签和子集是否匹配等方面。
2. 添加或删除配置时候，可能出现503错误，这是由于配置传播未到达导致。为确保不出现此情况，在添加配置时，应当先更新 DestinationRule，再更新 VirtualService 的配置对新添加的子集进行引用。当删除子集时，应当先删除 VirtualService 对任何子集的引用，再更新 DestinationRule ，删除不用的子集。
3. 还可能出现配有故障注入和重试/超时策略的 VirtualService 未按预期工作。 Istio 不支持在同一个 VirtualService 上配置故障注入和重试或超时策略，如果有此需求，可以通过 EnvoyFilter 注入故障。

配置 ServiceEnvoy 将外部服务注册到网格内部时未生效

排查：

可能是网格内部未开启对 DNS 的相关配置，可以检查下 istio-system 命名空间下的 configmap istio proxyMetadata字段。

```
v_yangjialiang@MacBook-Pro java-test % kubectl -n istio-system get cm istio -o json
{
    "apiVersion": "v1",
    "data": {
        "mesh": "accessLogFile: /dev/stdout\ndefaultConfig:\n  discoveryAddress: istiod.istio-system.svc:15012\n  proxyMetadata: \n    ISTIO_META_DNS_AUTO_ALLOCATE: \"true\"\n    ISTIO_META_DNS_CAPTURE: \"true\"\n  tracing:\n    zipkin:\n      address: zipkin.istio-system:9411\nenablePrometheusMerge: true\nextensionProviders:\n- envoyOtelAls:\n    port: 4317\n    service: otel-collector.istio-system.svc.cluster.local\n  name: otel\nrootNamespace: istio-system\ntrustDomain: cluster.local",
        "meshNetworks": "networks: {}"
```

sidecar 自动注入失败

排查：

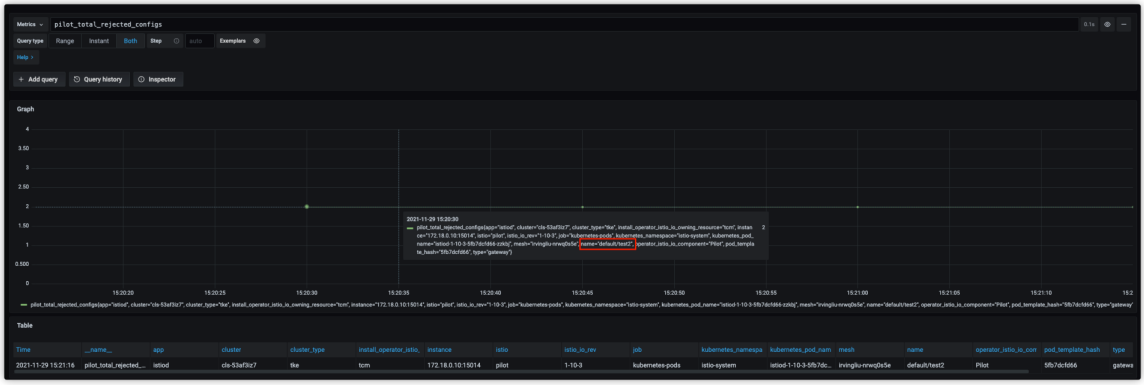
1. 首先确保 pod 不在 kube-system 或 kube-public命名空间中。 这些命名空间中的 Pod 将忽略 Sidecar 自动注入 。
2. 确保 Pod 定义中没有 hostNetwork：true。 hostNetwork：true 的 Pod 将忽略 Sidecar 自动注入。
3. 检查 webhook 关于自动注入的标签是否出现问题 或者 istio-system 命名空间下的 configmap istio-sidecar-injector 的默认注入策略是否有误。
4. 排查 pod 中的标签 sidecar.istio.io/inject: "true" 是否设置有误，true 为强制注入，false 则不会强制注入 sidecar。
5. 在注入失败的 Pod 的 Deployment 上运行 kubectl -n namespace describe deployment name。通常能在事件中看到调用注入 webhook 失败的原因。

控制平面出现 pilot_total_rejected_configs 指标

现象：

网格中同时存在以下两个 Gateway

```yaml
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: test1
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - test1.example.com
    port:
      name: https
      number: 443
      protocol: HTTPS
    tls:
      credentialName: example-credential
      mode: SIMPLE
---
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: test2
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - test1.example.com
    - test2.example.com
    port:
      name: https
      number: 443
      protocol: HTTPS
    tls:
      credentialName: example-credential
      mode: SIMPLE
```



请求 https://test1.example.com 正常返回 404，说明访问到了，请求 https://test2.example.com 出现异常。

原因：

通过 istiod 监控发现 pilot_total_rejected_configs 指标异常，显示 default/test2 配置被拒绝，导致被拒绝的原因是 每个域名在同一端口上只能配置一次 TLS，我们这里 test1.example.com 在 2 个 Gateway 的 443 端口都配置了 TLS， 导致其中一个被拒绝，通过监控确认被拒绝的是 test2，test2.example.com 和 test1.example.com 配置在 test2 的同一个 Server，Server 配置被拒绝导致请求异常，解决此问题可以把 test 1删除。删除后，请求恢复正常。

Envoy 日志中出现 headers size exceeds limit 错误

背景：

某个客户接入mesh后有个API接口调用失败，envoy 包含inbound、outbound两个方向，具体报错详情参考如下所示。

报错日志：

```
日志报错  2022-12-31T14:24:27.501043Z    debug   envoy client    [C4689] protocol error: headers size exceeds limit
```

相关问题：

https://github.com/envoyproxy/envoy/issues/13827

解决方案：

通过 envoyfilter 调大默认限制值。

```
 apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
 name: max-header
 namespace: istio-system
spec:
 configPatches:
 - applyTo: NETWORK_FILTER
   match:
     context: ANY
     listener:
       filterChain:
         filter:
           name: "envoy.http_connection_manager"
   patch:
     operation: MERGE
     value:
       typed_config:
         "@type":
"type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager"
         max_request_headers_kb: 80
         common_http_protocol_options:
           max_headers_count: 500
 - applyTo: CLUSTER
   match:
     context: ANY
     cluster:
       portNumber: 5000
   patch:
     operation: MERGE
     value: # cluster specification
       common_http_protocol_options:
         max_headers_count: 500
```

总结：

原因1：pod 中监听的端口与程序启动监听的端口不匹配，会导致出现 503 问题

原因2：envoy 自身对于 header 数量的限制，会出现 502 协议问题。可参见：https://github.com/envoyproxy/envoy/issues/13827

排错思路积累：

503 错误可以考虑 程序监听的端口与 Pod 配置的端口不一致。

502 错误可以考虑 header 数量过大导致，通过 istio-proxy debug 日志可以观察。

参考：

1. https://github.com/envoyproxy/envoy/issues/13827
2. https://www.gl.sh.cn/2022/04/11/network_filter-httpconnectionmanager.html
3. https://github.com/envoyproxy/envoy/issues/11774
4. https://github.com/envoyproxy/envoy/pull/12216

如何给Envoy开启AccessLog访问日志

我们可以通过 Telemetry API、Mesh Config、EnvoyFilter 三种方式开启 Envoy 的 access log 日志。

Telemetry API

```
 cat <<EOF > ./telemtry-example.yaml
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: mesh-default
spec:
  selector:
    matchLabels:
      app: helloworld
  accessLogging:
    - providers:
      - name: envoy
EOF

kubectl apply -f ./telemtry-example.yaml
```

我们查看 helloworld 中的 istio-proxy 中的日志，可以看到下发的 Telemetry 配置生效了。

```
 $ kubectl -n test logs helloworld-v2-7d55d87964-6tncj -c istio-proxy --tail=10
2023-04-17T09:27:13.557757Z info xdsproxy connected to upstream XDS server: 192.168.0.6:15012
2023-04-17T09:57:43.387553Z info xdsproxy connected to upstream XDS server: 192.168.0.6:15012
2023-04-17T10:30:52.106631Z info xdsproxy connected to upstream XDS server: 192.168.0.6:15012
2023-04-17T10:59:38.532144Z info xdsproxy connected to upstream XDS server: 192.168.0.6:15012
[2023-04-17T11:02:59.500Z] "GET /hello HTTP/1.1" 200 - via_upstream - "-" 0 60 119 119 "-" "curl/7.87.0-DEV"
"4d166eba-cb47-4147-a182-7a3896643d06" "helloworld:5000" "x.x.x.x:5000" inbound|5000|| x.x.x.x:37311
x.x.x.x:5000 x.x.x.x:54644 outbound_.5000_._.helloworld.test.svc.cluster.local default
[2023-04-17T11:03:00.923Z] "GET /hello HTTP/1.1" 200 - via_upstream - "-" 0 60 123 122 "-" "curl/7.87.0-DEV"
"fbd2b87a-913c-4d6f-8f8a-c625f1aaf80e" "helloworld:5000" "x.x.x.x:5000" inbound|5000|| x.x.x.x:38235 x.x.x.x:5000
x.x.x.x:54656 outbound_.5000_._.helloworld.test.svc.cluster.local default
[2023-04-17T11:03:39.534Z] "GET /hello HTTP/1.1" 200 - via_upstream - "-" 0 60 119 119 "-" "curl/7.87.0-DEV"
"bc364db9-dc03-4331-95db-ffea73fe5c0a" "helloworld:5000" "x.x.x.x:5000" inbound|5000|| x.x.x.x:52206 x.x.x.x:5000
x.x.x.x:54644 outbound_.5000_._.helloworld.test.svc.cluster.local default
[2023-04-17T11:03:42.281Z] "GET /hello HTTP/1.1" 200 - via_upstream - "-" 0 60 116 116 "-" "curl/7.87.0-DEV"
"b88fa88a-5ec1-4747-945b-8ace802db94f" "helloworld:5000" "x.x.x.x:5000" inbound|5000|| x.x.x.x:33526
x.x.x.x:5000 x.x.x.x:54644 outbound_.5000_._.helloworld.test.svc.cluster.local default
[2023-04-17T11:03:43.371Z] "GET /hello HTTP/1.1" 200 - via_upstream - "-" 0 60 120 119 "-" "curl/7.87.0-DEV"
"dd20b26a-af35-411b-a2e5-bc7f7f847c87" "helloworld:5000" "x.x.x.x:5000" inbound|5000|| x.x.x.x:55067 x.x.x.x:5000
x.x.x.x:54656 outbound_.5000_._.helloworld.test.svc.cluster.local default
[2023-04-17T11:03:44.500Z] "GET /hello HTTP/1.1" 200 - via_upstream - "-" 0 60 116 115 "-" "curl/7.87.0-DEV"
"be3a1993-0502-4f50-ad2d-c1672594962c" "helloworld:5000" "x.x.x.x:5000" inbound|5000|| x.x.x.x:34296
x.x.x.x:5000 x.x.x.x:54644 outbound_.5000_._.helloworld.test.svc.cluster.local default
```

使用 Mesh Config

修改 istio 配置:

```
 kubectl -n istio-system edit configmap istio
```

添加以下内容：

```
 apiVersion: v1
data:
  mesh: |-
    # 全局修改 Envoy 输出 accesslog
    accessLogEncoding: JSON
    accessLogFile: /dev/stdout
    accessLogFormat: ""
    defaultConfig:
      holdApplicationUntilProxyStarts: true
    rootNamespace: istio-system
kind: ConfigMap
metadata:
  name: istio
```

```
 accessLogEncoding: 表示 accesslog 输出格式，istio 预定义了 TEXT 和 JSON 两种日志输出格式。默认使用 TEXT，通常我们习惯改成 JSON 以提升可读性，同时也利于日志采集。
accessLogFile: 表示 accesslog 输出到哪里，通常我们指定到 /dev/stdout (标准输出)，以便使用 kubectl logs 来查看日志，同时也利于日志采集。
accessLogFormat: 如果不想使用 istio 预定义的 accessLogEncoding，我们也可以使用这个配置来自定义日志输出格式。
```

使用 EnvoyFilter

```
 cat <<EOF > ./envoyfilter-example.yaml
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
 name: enable-accesslog
spec:
 configPatches:
 - applyTo: NETWORK_FILTER
   match:
     context: ANY
     listener:
       filterChain:
         filter:
           name: envoy.http_connection_manager
   patch:
     operation: MERGE
     value:
       typed_config:
         "@type":
"type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager"
         access_log:
         - name: envoy.access_loggers.file
           typed_config:
             "@type": type.googleapis.com/envoy.extensions.access_loggers.file.v3.FileAccessLog
             path: "/dev/stdout"
             log_format:
               json_format:
                 authority: "%REQ(:AUTHORITY)%"
                 bytes_received: "%BYTES_RECEIVED%"
                 bytes_sent: "%BYTES_SENT%"
                 downstream_local_address: "%DOWNSTREAM_LOCAL_ADDRESS%"
                 downstream_remote_address: "%DOWNSTREAM_REMOTE_ADDRESS%"
                 duration: "%DURATION%"
                 method: "%REQ(:METHOD)%"
                 path: "%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%"
                 protocol: "%PROTOCOL%"
                 request_id: "%REQ(X-REQUEST-ID)%"
                 requested_server_name: "%REQUESTED_SERVER_NAME%"
                 response_code: "%RESPONSE_CODE%"
                 response_flags: "%RESPONSE_FLAGS%"
                 route_name: "%ROUTE_NAME%"
                 start_time: "%START_TIME%"
                 upstream_cluster: "%UPSTREAM_CLUSTER%"
                 upstream_host: "%UPSTREAM_HOST%"
                 upstream_local_address: "%UPSTREAM_LOCAL_ADDRESS%"
                 upstream_service_time: "%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)%"
                 upstream_transport_failure_reason: "%UPSTREAM_TRANSPORT_FAILURE_REASON%"
                 user_agent: "%REQ(USER-AGENT)%"
                 x_forwarded_for: "%REQ(X-FORWARDED-FOR)%"
EOF

kubectl apply -f ./envoyfilter-example.yaml
```

部分 workload 启用 accesslog

如果想要精确到只为指定的 workload 启用 accesslog，可以在 EnvoyFilter 上加一下 workloadSelector:

```
  spec:
  workloadSelector:
    labels:
      app: helloworld
```

我们查看 helloworld 中的 istio-proxy 中的日志，可以看到下发 EnvoyFilter 配置后生效了。

```
  {"authority":"helloworld:5000","method":"GET","user_agent":"curl/7.87.0-
DEV","bytes_received":0,"route_name":"default","upstream_host":"x.x.x.x:5000","upstream_service_time":"117","request_
-28a0-4dc3-a490-
6547474ff858","upstream_local_address":"x.x.x.x:39069","bytes_sent":60,"duration":117,"x_forwarded_for":null,"path":"/
-04-17T11:22:56.569Z","response_flags":"-","protocol":"HTTP/1.1","response_code":200}
```

使用SidecarCRD降低数据平面Envoy资源消耗

目的

给 istio 增加 Sidecar 配置，降低数据平面 Envoy 的资源消耗。

单namespace示例

单namespace集群的sidecar配置如下：

```
  apiVersion: networking.istio.io/v1beta1
kind: Sidecar
metadata:
  name: test-sidecar
  namespace: test
spec:
  egress:
  - hosts:
    - test/*  # 业务所在的命名空间
    - istio-system-csm-testtest/*   # istiod 所在的命名空间，csm 实例
```

多namespace示例

- 涉及的namespace：test、test-1

sidecar配置如下：

```
 #### ns为test的sidecar示例
apiVersion: networking.istio.io/v1beta1
kind: Sidecar
metadata:
  name: test-sidecar
  namespace: test
spec:
  egress:
  - hosts:
    - test/*  # 业务所在的命名空间
    - test-1/*  # 业务所在的命名空间
    - istio-system-csm-testtest/*   # istiod 所在的命名空间，csm 实例

#### ns为test-1的sidecar
apiVersion: networking.istio.io/v1beta1
kind: Sidecar
metadata:
  name: test-sidecar
  namespace: test-1
spec:
  egress:
  - hosts:
    - test/*  # 业务所在的命名空间
    - test-1/*  # 业务所在的命名空间
    - istio-system-csm-testtest/*   # istiod 所在的命名空间，csm 实例
```

Envoy删除部分Header

现象

业务注入的 Envoy 后，返回的 Header 中会有 x-envoy 信息，业务需要删除 x-envoy-decorator-operation header。

原因

envoy 默认会生成这些特定的 header。

解决方案

使用 envoyfilter 删除这些 header。参考的 header 如下所示：

```
  apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: response-headers-filter
spec:
  configPatches:
  - applyTo: NETWORK_FILTER
    match:
      context: ANY
      listener:
        filterChain:
          filter:
            name: "envoy.filters.network.http_connection_manager"
    patch:
      operation: MERGE
      value:
        typed_config:
          "@type":
"type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager"
          server_header_transformation: PASS_THROUGH
  - applyTo: HTTP_ROUTE
    match:
      context: SIDECAR_INBOUND
    patch:
      operation: MERGE
      value:
        decorator:
          propagate: false # removes the decorator header
        response_headers_to_remove:
        - x-envoy-upstream-service-time
        - x-powered-by
        - server
```

参考

https://janine.pcfe.net/posts/2022-10-10-istio-envoyfilter-strip-headers/

主集群primary Istiod 总是重启

现象

主集群 Istiod 总是发生 OOM 导致 Istiod Pod 重启，进而数据平面不能正常连接到 istiod 控制平面。

```
  Exec lifecycle hook ([pilot-agent wait]) for Container "istio-proxy" in Pod "podName" failed - error: command 'pilot-agent wait' exited with 255: Error: timeout waiting for Envoy proxy to become ready. Last error: HTTP status code 503 , message: "2023-03-31T01:30:00.231989Z\tinfo\tWaiting for Envoy proxy to be ready (timeout: 60 seconds)...\n2023-03-31T01:31:00.457233Z\terror\ttimeout waiting for Envoy proxy to become ready. Last error: HTTP status code 503\nError: timeout waiting for Envoy proxy to become ready. Last error: HTTP status code 503\n"
```

原因

istiod 所在的 k8s 集群 node、pod 等 Endpoint 元数据太多，导致 xds 推送频繁，耗费较多的资源，进而 istiod 容易 OOM。

解决方案

方案一：主集群 primary 添加 node 节点，保证 istiod 能够正常调度。

方案二：使用 sidecar 限制 istiod 计算范围。案例如下所示：

```
  apiVersion: networking.istio.io/v1beta1
kind: Sidecar
metadata:
  name: healthsvc-sidecar
  namespace: healthsvc-xxx
spec:
  egress:
  - hosts:
    - healthsvc-xxx/*  # 业务所在的命名空间
    - istio-system-xxx/*   # istiod 所在的命名空间
```

参考

1. https://istio.io/latest/docs/tasks/observability/distributed-tracing/jaeger/
2. https://istio.io/latest/docs/tasks/observability/metrics/customize-metrics/
3. https://github.com/istio/istio/issues/44266

某业务线上接口接入Mesh后不通

现象

某个业务线上服务接入 Mesh 后，某个接口出现 502。

```
  正常：curl -I 'http://x.x.x.x:8080/test/hospitalImjjh/card3/s?word=西安冠心病' -H 'host: xxxx.baidu.com'

异常：curl -I 'http://x.x.x.x:8080/test/hospitalImjjh/card3/s?
word=%E8%A5%BF%E5%AE%89%E5%86%A0%E5%BF%83%E7%97%85' -H 'host: xxxx.baidu.com'
```

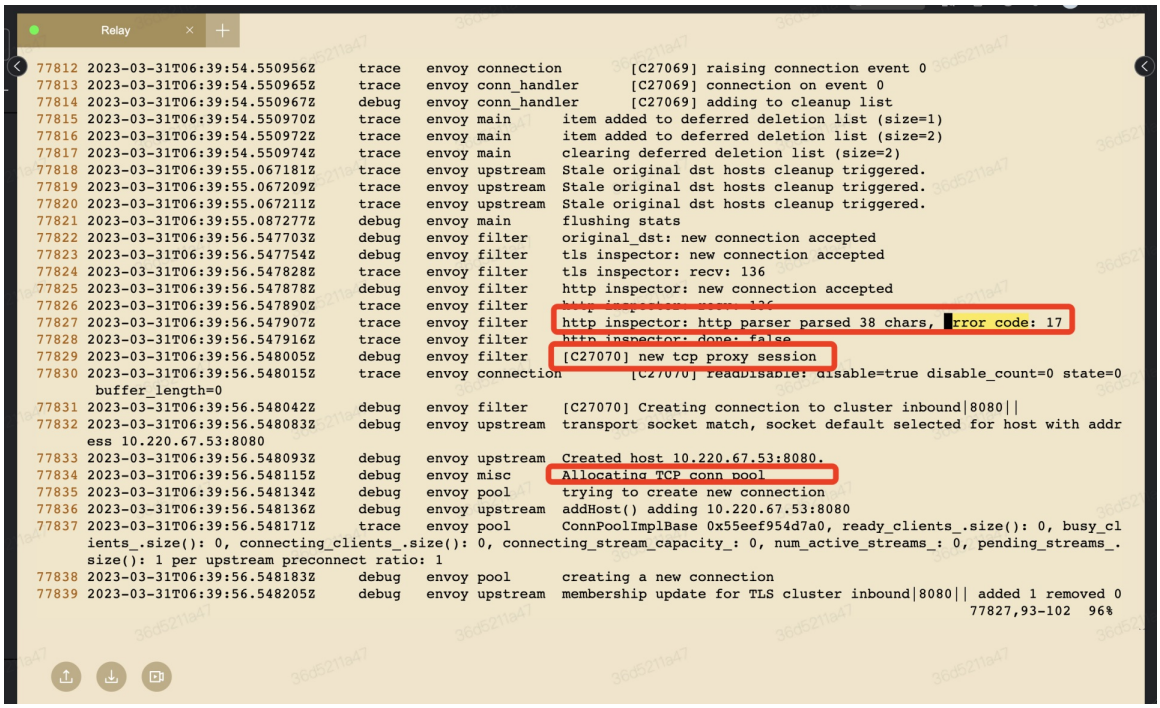
Envoy 的报错日志如下所示：

```
2023-03-31T02:56:51.929765Z trace envoy filter [C5806] upstream connection received 14240 bytes, end_stream=false
2023-03-31T02:56:51.929785Z trace envoy connection [C5806] writing 14240 bytes, end_stream false
2023-03-31T02:56:51.929795Z trace envoy connection [C5807] socket event: 2
2023-03-31T02:56:51.929798Z trace envoy connection [C5807] write ready
2023-03-31T02:56:51.929801Z trace envoy connection [C5806] socket event: 2
2023-03-31T02:56:51.929802Z trace envoy connection [C5806] write ready
2023-03-31T02:56:51.929844Z trace envoy connection [C5806] write returns: 14240
2023-03-31T02:56:51.930000Z trace envoy connection [C5805] socket event: 3
2023-03-31T02:56:51.930015Z trace envoy connection [C5805] write ready
2023-03-31T02:56:51.930020Z trace envoy connection [C5805] read ready. dispatch_buffered_data=false
2023-03-31T02:56:51.930039Z trace envoy connection [C5805] read returns: 14333
2023-03-31T02:56:51.930049Z trace envoy connection [C5805] read error: Resource temporarily unavailable
2023-03-31T02:56:51.930055Z trace envoy http [C5805] parsing 14333 bytes
2023-03-31T02:56:51.930063Z trace envoy http [C5805] message begin
2023-03-31T02:56:51.930076Z trace envoy http [C5805] completed header: key=Content-Type value=text/html;charset=utf-8
2023-03-31T02:56:51.930089Z trace envoy http [C5805] completed header: key=Transfer-Encoding value=chunked
2023-03-31T02:56:51.930107Z trace envoy http [C5805] completed header: key=Connection value=close
2023-03-31T02:56:51.930118Z trace envoy http [C5805] completed header: key=Vary value=Accept-Encoding
2023-03-31T02:56:51.930129Z trace envoy http [C5805] completed header: key=Server value=nginx/1.8.0
2023-03-31T02:56:51.930139Z trace envoy http [C5805] completed header: key=Date value=Fri, 31 Mar 2023 02:56:51 GMT
2023-03-31T02:56:51.930143Z trace envoy http [C5805] completed header: key=Vary value=Accept-Encoding
2023-03-31T02:56:51.930146Z trace envoy http [C5805] completed header: key=Set-Cookie value=58D4878A25DC3F9C0E2FB62870D096D3:FG=1; max-age=31536000; expires=Sat, 30-Mar-24 02:56:50 GMT; path=/; version=1; comment=bd
2023-03-31T02:56:51.930156Z debug envoy client [C5805] Error dispatching received data: http/1.1 protocol error: HPE_INVALID_HEADER_TOKEN
2023-03-31T02:56:51.930160Z debug envoy connection [C5805] closing data_to_write=0 type=1
2023-03-31T02:56:51.930162Z debug envoy connection [C5805] closing socket: 1
2023-03-31T02:56:51.930199Z trace envoy connection [C5805] raising connection event 1
2023-03-31T02:56:51.930206Z debug envoy client [C5805] disconnect. resetting 1 pending requests
2023-03-31T02:56:51.930210Z debug envoy client [C5805] request reset
2023-03-31T02:56:51.930213Z trace envoy main item added to deferred deletion list (size=1)
2023-03-31T02:56:51.930222Z debug envoy router [C5804][S6039679106682162641] upstream reset: reset reason: protocol error, transport failure reason:
2023-03-31T02:56:51.930231Z trace envoy connection [C5806] socket event: 3
2023-03-31T02:56:51.930244Z trace envoy connection [C5806] write ready
2023-03-31T02:56:51.930247Z trace envoy connection [C5806] read ready. dispatch_buffered_data=false
```

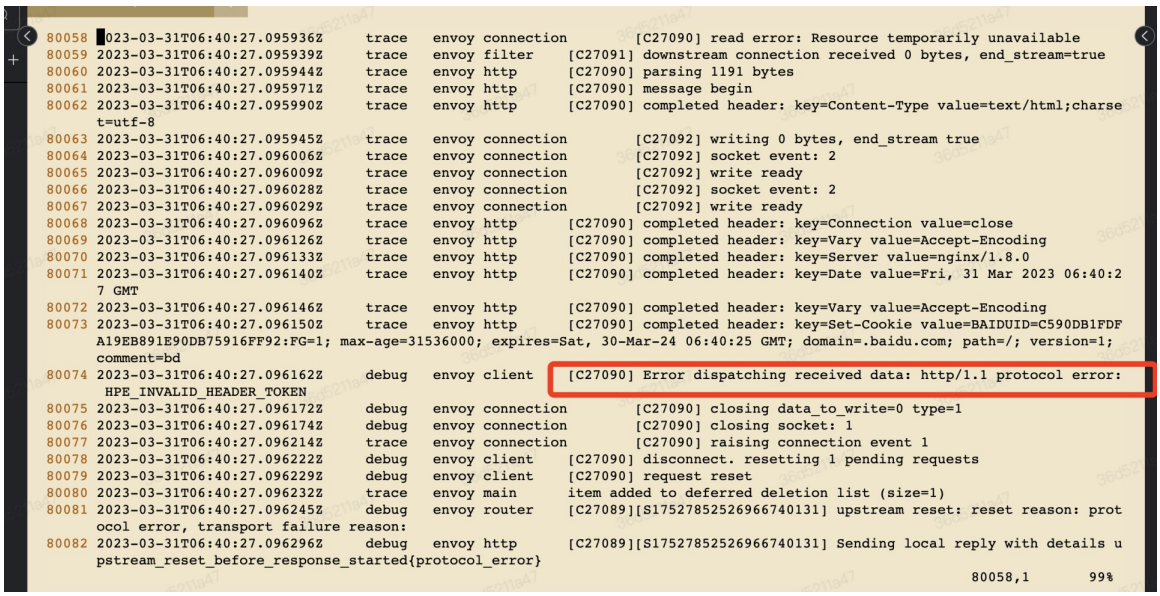
原因

1. 为什么两个请求，第一个请求正常 第二个会报错？

第一个能够通信的原因是：流量当成 TCP 处理。



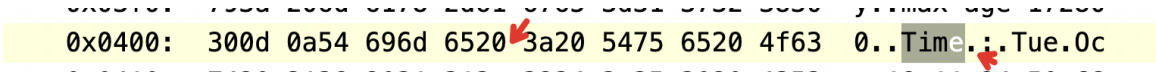
2. 导致第二个请求报错的原因是：



```
 80058 2023-03-31T06:40:27.095936Z    trace   envoy connection       [C27090] read error: Resource temporarily unavailable
80059 2023-03-31T06:40:27.095939Z    trace   envoy filter    [C27091] downstream connection received 0 bytes, end_stream=true
80060 2023-03-31T06:40:27.095944Z    trace   envoy http      [C27090] parsing 1191 bytes
80061 2023-03-31T06:40:27.095971Z    trace   envoy http      [C27090] message begin
80062 2023-03-31T06:40:27.095990Z    trace   envoy http      [C27090] completed header: key=Content-Type value=text/html;charse      t=utf-8
80063 2023-03-31T06:40:27.095945Z    trace   envoy connection       [C27092] writing 0 bytes, end_stream true
80064 2023-03-31T06:40:27.096006Z    trace   envoy connection       [C27092] socket event: 2
80065 2023-03-31T06:40:27.096009Z    trace   envoy connection       [C27092] write ready
80066 2023-03-31T06:40:27.096028Z    trace   envoy connection       [C27092] socket event: 2
80067 2023-03-31T06:40:27.096029Z    trace   envoy connection       [C27092] write ready
80068 2023-03-31T06:40:27.096096Z    trace   envoy http      [C27090] completed header: key=Connection value=close
80069 2023-03-31T06:40:27.096126Z    trace   envoy http      [C27090] completed header: key=Vary value=Accept-Encoding
80070 2023-03-31T06:40:27.096133Z    trace   envoy http      [C27090] completed header: key=Server value=nginx/1.8.0
80071 2023-03-31T06:40:27.096140Z    trace   envoy http      [C27090] completed header: key=Date value=Fri, 31 Mar 2023 06:40:2      7 GMT
80072 2023-03-31T06:40:27.096146Z    trace   envoy http      [C27090] completed header: key=Vary value=Accept-Encoding
80073 2023-03-31T06:40:27.096150Z    trace   envoy http      [C27090] completed header: key=Set-Cookie value=C590DB1FDF      A19EB891E90DB75916FF92:FG=1; max-age=31536000; expires=Sat, 30-Mar-24 06:40:25 GMT; path=/; version=1;       comment=bd
80074 2023-03-31T06:40:27.096162Z    debug   envoy client    [C27090] Error dispatching received data: http/1.1 protocol error:       HPE_INVALID_HEADER_TOKEN
80075 2023-03-31T06:40:27.096172Z    debug   envoy connection       [C27090] closing data_to_write=0 type=1
80076 2023-03-31T06:40:27.096174Z    debug   envoy connection       [C27090] closing socket: 1
80077 2023-03-31T06:40:27.096214Z    trace   envoy connection       [C27090] raising connection event 1
80078 2023-03-31T06:40:27.096222Z    debug   envoy client    [C27090] disconnect. resetting 1 pending requests
80079 2023-03-31T06:40:27.096229Z    debug   envoy client    [C27090] request reset
80080 2023-03-31T06:40:27.096232Z    trace   envoy main      item added to deferred deletion list (size=1)
80081 2023-03-31T06:40:27.096245Z    debug   envoy router    [C27089][S17527852526966740131] upstream reset: reset reason: prot      ocol error, transport failure reason:
80082 2023-03-31T06:40:27.096296Z    debug   envoy http      [C27089][S17527852526966740131] Sending local reply with details u      pstream_reset_before_response_started{protocol_error}
```

业务流量抓包

```
 0x03d0:  7061 7468 3d2f 0d0a 5374 7269 6374 2d54  path=/..Strict-T
0x03e0:  7261 6e73 706f 7274 2d53 6563 7572 6974  ransport-Securit
0x03f0:  793a 206d 6178 2d61 6765 3d31 3732 3830  y:.max-age=17280
0x0400:  300d 0a54 696d 6520 3a20 5475 6520 4f63  0..Time.:.Tue.Oc
0x0410:  7420 3138 2031 313a 3234 3a35 3020 4353  t.18.11:24:50.CS
0x0420:  5420 3230 3232 0d0a 5472 6163 6569 643a  T.2022..Traceid:
0x0430:  2031 3638 3033 3535 3131 3530 3537 3335  .168035511505735
0x0440:  3937 3435 3038 3739 3139 3332 3734 3131  9745087919327411
0x0450:  3330 3639 3832 3438 0d0a 5661 7279 3a20  30698248..Vary:.
0x0460:  4163 6365 7074 2d45 6e63 6f64 696e 670d  Accept-Encoding.
0x0470:  0a58 2d47 732d 466c 6167 3a20 3078 300d  .X-Gs-Flag:.0x0.
```



在线工具网

在线16进制字符串转换工具

在线16进制字符串转换的操作指引： ☆加入收藏

在下方文本框输入原始字符串，点击"字符串转16进制"按钮，即可在最下方的文本框显示编码后的16进制字符串；相反，点击"16进制转字符串"按钮，即可把16进制字符串解码成原始字符串。

如果要根据16进制字符串转化保存为对应的二进制文件，则需使用：在线16进制字符串转文件工具

请输入要进行 16进制字符串转换 的字符。

54696d65203a20547565204f63

字符串转16进制 (Encode) 16进制转字符串 (Decode) ↕ 交换 (编码快捷键: Ctrl + Enter)

16进制字符串转换 的结果： 编/解码后自动全选

Time：Tue Oc

有问题的 header key

```
0x0400:  300d 0a54 696d 6520 3a20 5475 6520 4f63  0..Time.:.Tue.Oc
```

"Time :" 这里有个空格，不符合 header 规范，触发envoy检测header报错。

> Error dispatching received data: http/1.1 protocol error: HPE_INVALID_HEADER_TOKEN

线下环境复现和调试

下载envoy (使用1.18.3版本)，使用如下最简配置文件启动envoy，转发流量到业务服务。

```
admin:
 address:
  socket_address: { address: 127.0.0.1, port_value: 9901 }

static_resources:
 listeners:
 - name: listener_0
  address:
   socket_address: { address: 127.0.0.1, port_value: 10000 }
  filter_chains:
  - filters:
   - name: envoy.filters.network.http_connection_manager
     typed_config:
       "@type":
type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager
       stat_prefix: ingress_http
       codec_type: AUTO
       route_config:
         name: local_route
         virtual_hosts:
         - name: local_service
           domains: ["*"]
           routes:
           - match: { prefix: "/" }
             route: { cluster: some_service }
       http_filters:
       - name: envoy.filters.http.router
         typed_config:
           "@type": type.googleapis.com/envoy.extensions.filters.http.router.v3.Router
 clusters:
 - name: some_service
  connect_timeout: 0.25s
  type: STATIC
  lb_policy: ROUND_ROBIN
  load_assignment:
   cluster_name: some_service
   endpoints:
   - lb_endpoints:
     - endpoint:
         address:
           socket_address:
             address: 业务IP
             port_value: 业务端口
```

请求envoy 发送业务请求，即可看到如下报错（开启envoy日志等级为debug）。

```
 [2023-04-01 18:14:44.358][1535551][trace][connection] [source/common/network/raw_buffer_socket.cc:38] [C17] read error: Resource temporarily unavailable
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:555] [C17] parsing 1079 bytes
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:834] [C17] message begin
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=idcinfo value=hba
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Bdqid value=43e2823cd8d4f5df
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Bdqid value=43e2823cd8d4f5df
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Bdsvrtm value=2374
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Cache-Control value=no-cache
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Content-Type value=text/html;charset=utf-8
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Date value=Sat, 01 Apr 2023 10:14:44 GMT
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=P3p value=CP=" OTI DSP COR IVA OUR IND COM "
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Rpql value=1
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Server value=apache
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Set-Cookie value=7F014F6421F98673E55D5BA0F02EB03D:FG=1; max-age=31536000; expires=Sun, 31-Mar-24 10:14:42 GMT; path=/; version=1; comment=bd
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Set-Cookie value=delPer=0; path=/
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Set-Cookie value=search_mapping=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; path=/
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Set-Cookie value=bzt_device_sn=deleted; expires=Thu, 01-Jan-1970 17:00:00 GMT; path=/
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Set-Cookie value=bzt_username=deleted; expires=Thu, 01-Jan-1970 17:00:00 GMT; path=/
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Set-Cookie value=BDSVRTM=2374; path=/
[2023-04-01 18:14:44.358][1535551][trace][http] [source/common/http/http1/codec_impl.cc:475] [C17] completed header: key=Set-Cookie value=PSINO=1; path=/
[2023-04-01 18:14:44.358][1535551][debug][client] [source/common/http/codec_client.cc:149] [C17] Error dispatching received data: http/1.1 protocol error: HPE_INVALID_HEADER_TOKEN
[2023-04-01 18:14:44.358][1535551][debug][connection] [source/common/network/connection_impl.cc:133] [C17] closing data_to_write=0 type=1
[2023-04-01 18:14:44.358][1535551][debug][connection] [source/common/network/connection_impl.cc:243] [C17] closing socket: 1
```

验证Time\s: 确实会触发报错

随便找个不报错的服务，本地启动一个nginx，转发流量到该服务，且添加header Time：xxx，将enovy本地配置转发到改 nginx 服务，即可复现错误，且可以通过调整是否添加此header来验证。

```
 location / {
    add_header "Time " "xxx";
    prox_pass http://服务;
}
```

Envoy 源码报错定位

日志debug信息

```
source/common/http/http1/codec_impl.cc:475 [C17] completed header: key=Set-Cookie value=PSINO=1; d
source/common/http/codec_client.cc:149 [C17] Error dispatching received data: http/1.1 protocol error:
HPE_INVALID_HEADER_TOKEN
```

代码校验header位置：https://github.com/nodejs/http-parser/blob/5c5b3ac62662736de9e71640a8dc16da45b32503/http_parser.c

http_parser.c

```
1255
1256    case s_header_field:
1257    {
1258      const char* start = p;
1259      for (; p != data + len; p++) {
1260        ch = *p;
1261        c = TOKEN(ch);
1262
1263        if (!c)
1264          break;
1265
1266        switch (parser->header_state) {
```

```
430
431 #define STRICT_TOKEN(c) ((c == ' ') ? 0 : tokens[(unsigned char)c])
432
433 #if HTTP_PARSER_STRICT
434 #define TOKEN(c)        STRICT_TOKEN(c)
435 #define IS_URL_CHAR(c)      (BIT_AT(normal_url_char, (unsigned char)c))
436 #define IS_HOST_CHAR(c)     (IS_ALPHANUM(c) || (c) == '.' || (c) == '-')
437 #else
438 #define TOKEN(c)            tokens[(unsigned char)c]
439 #define IS_URL_CHAR(c)                                              \
440   (BIT_AT(normal_url_char, (unsigned char)c) || ((c) & 0x80))
441 #define IS_HOST_CHAR(c)                                             \
442   (IS_ALPHANUM(c) || (c) == '.' || (c) == '-' || (c) == ' ')
```

```
1255
1256    case s_header_field:
1257    {
1258      const char* start = p;
1259      for (; p != data + len; p++) {…
1374
1375      if (p == data + len) {
1376        --p;
1377        COUNT_HEADER_SIZE(p - start);
1378        break;
1379      }
1380
1381      COUNT_HEADER_SIZE(p - start);
1382
1383      if (ch == ':') {
1384        UPDATE_STATE(s_header_value_discard_ws);
1385        CALLBACK_DATA(header_field);
1386        break;
1387      }
1388
1389      SET_ERRNO(HPE_INVALID_HEADER_TOKEN);
1390      goto error;
1391    }
```

解决方案

解决方式：修改业务中相关的接口代码，将 header 中的 key 设置正确。

参考

1. https://github.com/istio/istio/issues/36711
2. https://github.com/istio/istio/issues/31458
3. https://stackoverflow.com/questions/72808664/istio-error-dispatching-received-data-http-1-1-protocol-error-unsupported-tra
4. https://github.com/istio/istio/issues/39706

某业务接入Mesh某个接口不通

现象

某个业务接入 Mesh 后，部分接口出现 502。

访问以下接口，直接返回 502。

```
curl -v 'http://x.x.x.x:80/api/v1/hub/meta/attachment?attachmentId=20230303035842-xhchy&fileName=%E7%A7%81%E5%AF%86%E9%A1%B9%E7%9B%AE%E5%8F%82%E4%B8%8E%E4%BA%BA%E5%91%98%E4%
' -H'cookie: xxx' -H'host:xxx.baidu.com'
```

Envoy 的 trace 日志是：

2023-03-06T13:23:40.384836Z debug envoy client [C14158] Error dispatching received data: http/1.1 protocol error: HPE_INVALID_HEADER_TOKEN。

```
2023-03-06T13:23:40.381998Z debug envoy router [C14938][S1043862392353535809] upstream headers complete: end_stream=false
'date', 'Mon, 06 Mar 2023 13:23:40 GMT'
'content-type', 'application/pdf'
'content-length', '108860'
'accept-ranges', 'bytes'
'content-md5', '+e4pDfRHe+JDTsww5MGeOw=='
'etag', '"f9ee290df4477be2434ecc30e4c19e3b"'
'expires', 'Thu, 09 Mar 2023 13:23:40 GMT'
'last-modified', 'Thu, 02 Mar 2023 19:58:42 GMT'
'server', 'envoy'
'x-bce-expiration-date', '2022-11-25T00:00:00Z'
'x-bce-request-id', '358f4bdc-5d3e-4209-92c5-8156b65a1a06'
'x-bce-storage-class', 'STANDARD'
'x-envoy-upstream-service-time', '52'

2023-03-06T13:23:40.384836Z debug envoy client [C14158] Error dispatching received data: http/1.1 protocol error: HPE_INVALID_HEADER_TOKEN
2023-03-06T13:23:40.384855Z debug envoy connection [C14158] closing data_to_write=0 type=1
2023-03-06T13:23:40.384904Z debug envoy client [C14158] request reset
2023-03-06T13:23:40.384913Z debug envoy router [C14935][S15319585049759823265] upstream reset: reset reason: protocol error, transport failure reason:
2023-03-06T13:23:40.384978Z debug envoy http [C14935][S15319585049759823265] Sending local reply with details upstream_reset_before_response_started{protocol_error}
2023-03-06T13:23:40.385016Z debug envoy http [C14935][S15319585049759823265] encoding headers via codec (end_stream=false):
':status', '502'
'content-length', '87'
'content-type', 'text/plain'
'date', 'Mon, 06 Mar 2023 13:23:39 GMT'
'server', 'istio-envoy'
'x-envoy-decorator-operation', ':0/*'
```

原因

问题根因：业务代码在返回response时新增header，header中为encode文件名，导致502错误。

```
response.addHeader( name: "Content-Disposition",
        String.format("attachment;filename=%s", fileName));
```

解决方案

解决方式：修改业务中相关的接口代码，将 fileName 正确地 Encoding。

某业务访问某服务 mesh配置失效问题

现象

某业务配置了envoyfilter去给请求增加header，请求集群内的服务有header，请求集群外的服务没有header。

原因

上游未找到被访问服务对应的信息，导致outbound透传了，没有执行用户配置的envoyfilter逻辑。

```
2023-09-12T12:16:12.597153Z     info    xdsproxy        connected to upstream XDS server: istiod-remote.istio-system-csm-z04gaeve.svc:15012
2023-09-12T12:20:05.770165Z     info    xdsproxy        connected to upstream XDS server: istiod-remote.istio-system-csm-z04gaeve.svc:15012
[2023-09-12T12:23:47.861Z] "- - -" 0 - - - "-" 104 328 1025 - "-" "-" "-" "-" "10.11.1.159:80" PassthroughCluster 10.220.105.130:16932 10.11.1.159:80 10.220.105.130:16930 - -
```

解决方式

将被访问服务信息通过serviceEntry信息导入到集群中。参考https://istio.io/latest/docs/reference/config/networking/service-entry/

Istiod出现报错

现象

istiod 启动过程中出现以下报错。

```
Warning  FailedMount  8s (x6 over 24s)  kubelet            MountVolume.SetUp failed for volume "istio-token" : failed to fetch token: the API server does not have TokenRequest endpoints enabled
```

原因

具体问题，可见 https://istio.io/latest/docs/ops/best-practices/security/#configure-third-party-service-account-tokens

解决方案

根据 k8s 平台是否支持 jwtPolicy third-party 自动调整 values.global.jwtPolicy 策略。

```
istioctl manifest generate --set values.global.jwtPolicy=first-party-jwt  > first-party-jwt.yaml

istioctl manifest generate --set values.global.jwtPolicy=third-party-jwt  > third-party-jwt.yaml
```

参考

https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/#service-account-token-volume-projection

pod处于terminating状态，流量丢失问题

问题

pod处于terminating状态，流量丢失问题。

原因

用户部署过程中，某服务侧同步新版本 pod 需要耗时几分钟，这个时候旧版本 pod 处于 terminating 状态，业务使用 preStop 让当前状态的 pod 还可以接受流量，直到服务侧更新完成。但是注入sidecar容器后，sidecar容器会在 pod 状态变为 terminating 后直接拒绝流量，导致在变更的几分钟 导向旧版本 pod 的流量全部丢失。

解决方案

临时方案

直接在sidecar的cm中添加prestop。

```
{{- else if $holdProxy }}
lifecycle:
  postStart:
    exec:
      command:
      - pilot-agent
      - wait
  preStop:
    exec:
      command:
      - sh
      - -c
      - |
        while true; do
          if curl -s http://127.0.0.1:8080/health; then
            sleep 5
          else
            break
          fi
        done
{{- end }}
```

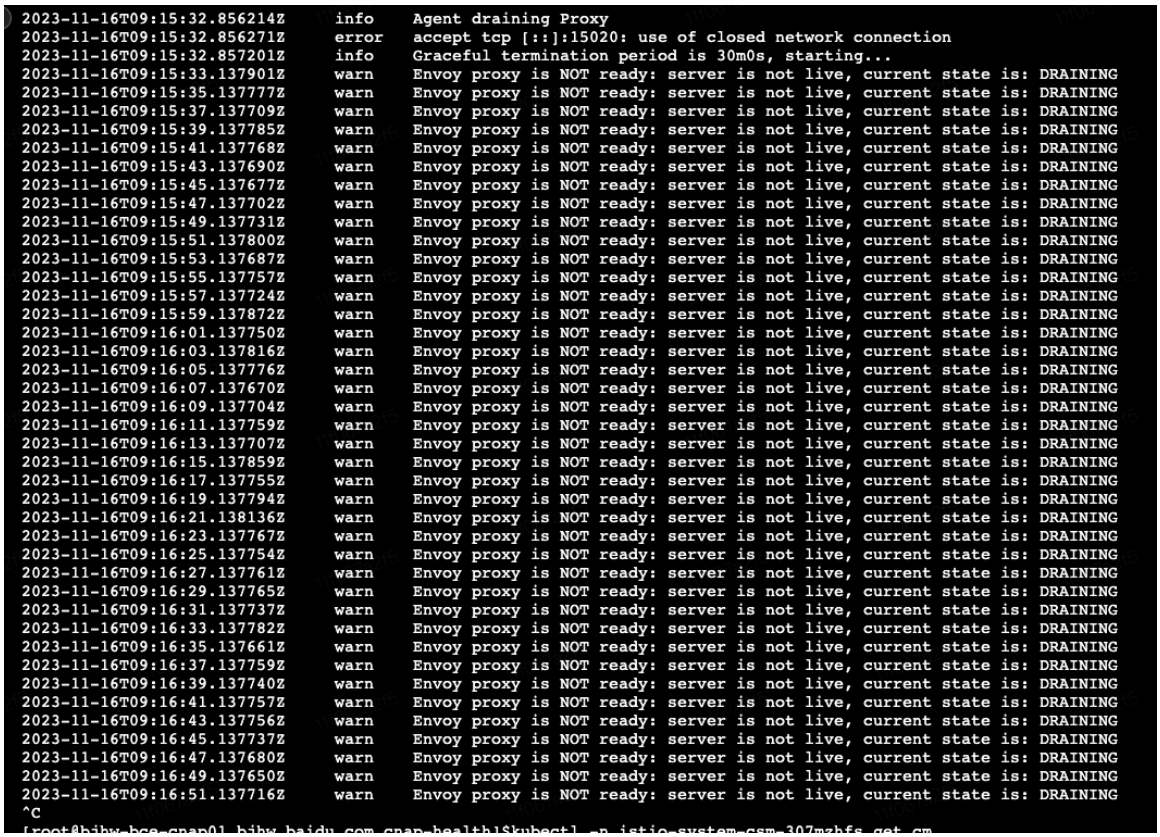
期望方案

terminationDrainDuration

https://istio.io/latest/docs/reference/config/istio.mesh.v1alpha1/

	gateway workloads.	
uration	The amount of time allowed for connections to complete on proxy shutdown. On receiving `SIGTERM` or `SIGINT`, `istio-agent` tells the active Envoy to start draining, preventing any new connections and allowing existing connections to complete. It then sleeps for the `termination_drain_duration` and then kills any remaining active Envoy processes. If not set, a default of `5s` will be applied.	No

cm istio中 设置这个参数。设置为30min后，退出时envoy日志如下：



实测，主容器存活的情况下，curl主容器的health端口会被拒绝，看起来与描述一致，拒绝了新请求。

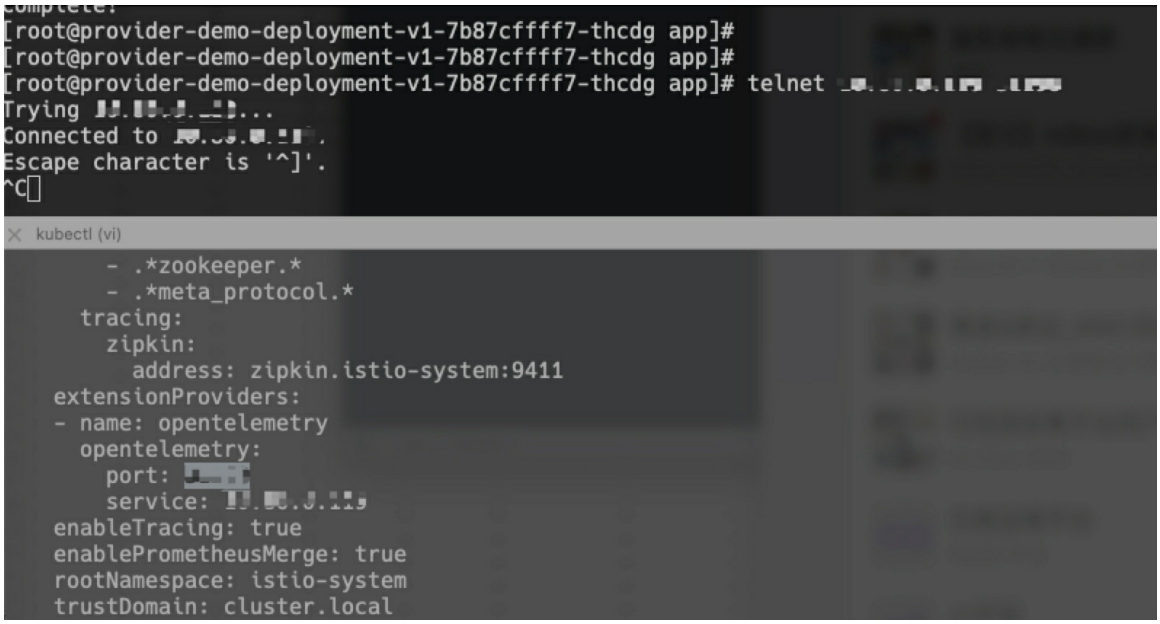
参考

https://github.com/istio/istio/issues/36517 https://github.com/istio/istio/pull/35059

istio 支持的 openTelemetry 没有收集到 trace 数据

问题

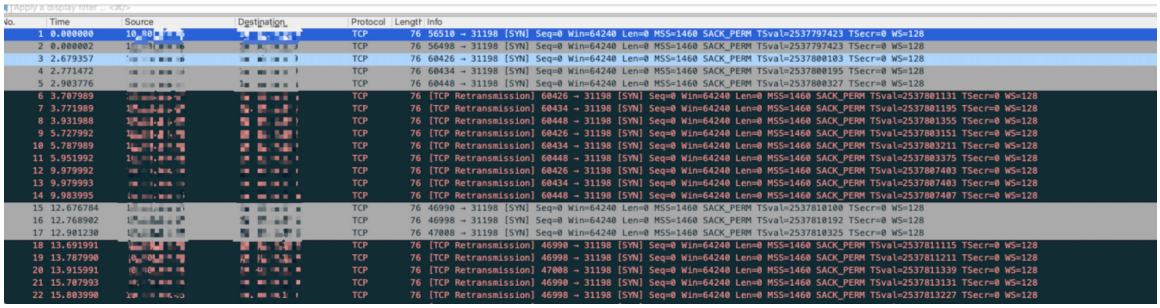
OpenTelemetry 不能收集到 trace 数据。



原因

istio configmap 配置的地址网络不通。

```
reason:
2023-08-16T13:16:38.534429Z debug   envoy http  async http request response headers (end_stream=true):
':status', '200'
'content-type', 'application/grpc'
'grpc-status', '14'
'grpc-message', 'upstream connect error or disconnect/reset before headers. reset reason: connection failure'

2023-08-16T13:16:38.534429Z debug   envoy http  async http request response headers (end_stream=true):
':status', '200'
'content-type', 'application/grpc'
'grpc-status', '14'
'grpc-message', 'upstream connect error or disconnect/reset before headers. reset reason: connection failure'

2023-08-16T13:16:38.534449Z debug   envoy pool  invoking idle callbacks - is_draining_for_deletion_=false
2023-08-16T13:16:38.534449Z debug   envoy pool  invoking idle callbacks - is_draining_for_deletion_=false
2023-08-16T13:16:38.534470Z debug   envoy router    [C0][S13727444527920063367] cluster 'outbound|31198||otel-trace.dev.com' match for URL
'/opentelemetry.proto.collector.trace.v1.TraceService/Export'
2023-08-16T13:16:38.534497Z debug   envoy router    [C0][S983716602175759168] cluster 'outbound|31198||otel-trace.dev.com' match for URL
'/opentelemetry.proto.collector.trace.v1.TraceService/Export'
2023-08-16T13:16:38.534509Z debug   envoy router    [C0][S13727444527920063367] router decoding headers:
':method', 'POST'
':path', '/opentelemetry.proto.collector.trace.v1.TraceService/Export'
':authority', 'otel-trace.dev.com'
':scheme', 'http'
'te', 'trailers'
'content-type', 'application/grpc'
'x-envoy-internal', 'true'
'x-forwarded-for', '[redacted]'

2023-08-16T13:16:38.534522Z debug   envoy router    [C0][S983716602175759168] router decoding headers:
':method', 'POST'
':path', '/opentelemetry.proto.collector.trace.v1.TraceService/Export'
':authority', 'otel-trace.dev.com'
':scheme', 'http'
```



解决方式

配置正确的 trace 地址。

参考

https://istio.io/latest/docs/tasks/observability/logs/otel-provider/

istio-proxy 没有 access_log 日志

问题

istio-system 命名空间配置了accesslog，但是 istio-proxy 没有日志。

配置如下：

```
  apiVersion: v1
data:
  mesh: |-
    accessLogEncoding: JSON
    accessLogFile: /dev/stdout
    defaultConfig:
      discoveryAddress: istiod/istio-system.svc:15012
      holdApplicationUntilProxyStarts: true
      meshId: csm-u7r86kt8
      proxyMetadata:
        ISTIO_META_DNS_AUTO_ALLOCATE: "true"
        ISTIO_META_DNS_CAPTURE: "true"
      proxyStatsMatcher:
        inclusionPrefixes:
        - thrift
        - dubbo
        - kafka
        - meta_protocol
        inclusionRegexps:
        - .*dubbo.*
        - .*thrift.*
        - .*kafka.*
        - .*zookeeper.*
        - .*meta_protocol.*
      tracing:
        zipkin:
          address: zipkin.istio-system:9411
    enableTracing: true
    extensionProviders:
    - name: otel
      opentelemetry:
        port: 31504
        service: x.x.x.x
    enablePrometheusMerge: true
    rootNamespace: istio-system
    trustDomain: cluster.local
  meshNetworks: 'networks: {}'
kind: ConfigMap
metadata:
  creationTimestamp: "2023-08-11T07:31:14Z"
  labels:
    install.operator.istio.io/owning-resource: unknown
    istio.io/rev: default
    operator.istio.io/component: Pilot
    release: istio
  name: istio
  namespace: istio-system
  resourceVersion: "1536380"
  uid: d89e1f61-f9af-4f5e-9eaa-b26616fb267e
```

原因

discoveryAddress: istiod/istio-system.svc:15012 配置错误，可以通过 istiod 控制平面报错日志看出来。

解决方式

将 discoveryAddress: istiod/istio-system.svc:15012 配置为正确的地址 discoveryAddress: istiod.istio-system.svc:15012。

参考

https://cloud.tencent.com/developer/article/2172765

注入 sidecar pod 启动失败

问题：按照CSM操作手册，某个Pod开启sidecar注入后，启动失败，查看pod中的istio-init容器日志，如下报错：

```
2023-11-02T02:25:27.723319Z info Running command: iptables-restore --noflush /tmp/iptables-rules-1698891927723191413.txt835755000
2023-11-02T02:25:27.724588Z error Command error output: xtables parameter problem: iptables-restore: unable to initialize table 'raw'

Error occurred at line: 1
Try `iptables-restore -h' or 'iptables-restore --help' for more information.
2023-11-02T02:25:27.724604Z error Failed to execute: iptables-restore --noflush /tmp/iptables-rules-1698891927723191413.txt835755000, exit status 2
```

原因：CCE集群中可能存在Istio不支持的Node节点，如BaiduLinux、Centos8等内核。具体原因可参见Istio前置条件。Centos8及一些红帽系Linux使用iptables-nftables，不使用iptables模块。Istio通过使用iptables添加nat规则来拦截流量，Linux应该启用netfix linux内核模块。

解决办法：
永久生效(需要重启CCE集群上的Node节点机器)，在节点Node上执行以下操作：

```
cat >/etc/modules-load.d/99-istio-modules.conf <<EOF
br_netfilter
nf_nat
nf_nat_redirect
xt_REDIRECT
xt_owner
iptable_nat
iptable_mangle
iptable_filter
EOF
#### 重启下机器
reboot
```

临时生效(不需要重启CCE集群上的Node节点机器，机器重启后失效)，在节点Node上执行以下操作：

```
modprobe br_netfilter
modprobe nf_nat
modprobe nf_nat_redirect
modprobe xt_REDIRECT
modprobe xt_owner
modprobe iptable_nat
modprobe iptable_mangle
modprobe iptable_filter
```

如何禁用 Sidecar 注入

如果命名空间启用了 Sidecar 自动注入，但希望该命名空间下的某个 pod 不注入 Sidecar，可以通过在 Pod 模板中添加 label 实现：

```
template:
 metadata:
   labels:
     sidecar.istio.io/inject: "false"
```

以 cronjob 为例

```
 apiVersion: batch/v1
kind: CronJob
metadata:
 name: hello-cronjob
spec:
 schedule: "* * * * *"
 jobTemplate:
   spec:
     template:
       metadata:
         labels:
           sidecar.istio.io/inject: "false"
       spec:
         containers:
         - name: hello
           image: registry.baidubce.com/csm-offline/busybox
           command: ["/bin/sh", "-c", "date; echo Hello from Kubernetes CronJob"]
         restartPolicy: OnFailure
```

诊断工具

概述

本节介绍服务网格 CSM 提供异常诊断功能。

异常分析

功能简介

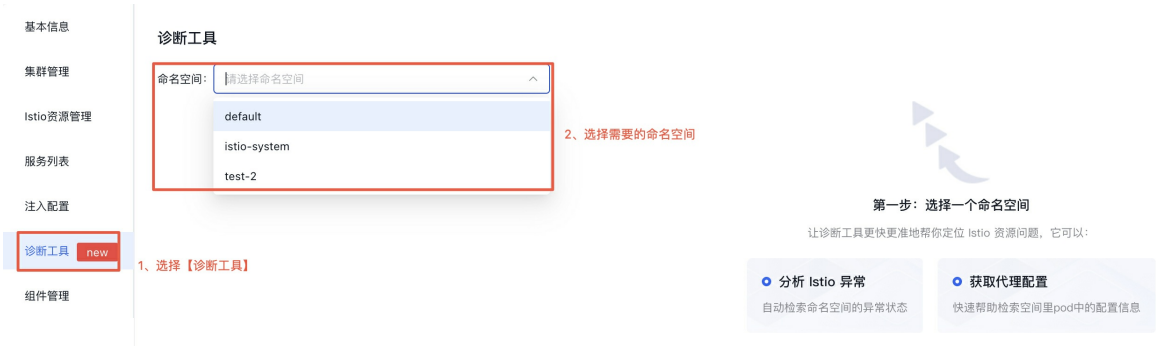
该功能用于对网格数据面配置进行静态分析，以便在部署后识别潜在的配置错误和问题。它能够扫描和验证网格的自定义资源配置，输出警告、错误和建议，帮助用户确保服务网格配置的健康情况。分析结果可以以多种格式输出，方便用户提高配置的准确性，并预防潜在的运行时问题。

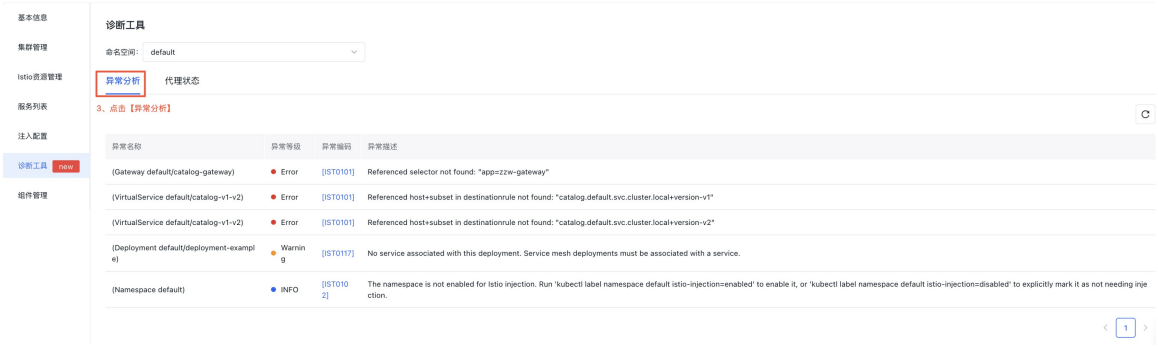
查看异常分析

按以下步骤在控制台上查看异常分析：

1. 登录 百度智能云控制台，选择“产品服务>云计算>容器>服务网格 CSM”。
2. 在服务网格控制台，“网格列表”页面选择地域后，找到想要操作的服务网格实例，点击服务网格实例名称 可在左侧边栏看到【诊断工具】。
3. 进入【诊断工具】页面后点击“异常分析”，即可进入页面。

该页面展示了相应命名空间下的异常服务的相关信息。





代理状态

功能简介

该功能可以查看网格的概况，检查当前 XDS 通讯状态，来确认数据面与控制面之间的配置同步情况，是否有 sidecar 无法接收配置或无法保持同步。如果某个代理没有出现在输出列表中，则说明该代理没有连接到 istiod 实例，因此也无法接收任何配置信息。

展示字段包括：

- CDS：Cluster Discovery Service（CDS）状态，表明集群发现服务是否与所有配置同步。
- LDS：Listener Discovery Service（LDS）状态，表示监听器发现服务是否与所有配置同步。
- EDS：Endpoint Discovery Service（EDS）状态，表示端点发现服务是否与所有配置同步。
- RDS：Route Discovery Service（RDS）状态，表示路由发现服务是否与所有配置同步。
- ECDS：Endpoint Configuration Discovery Service（ECDS）状态，表示端点配置发现服务是否与所有配置同步。

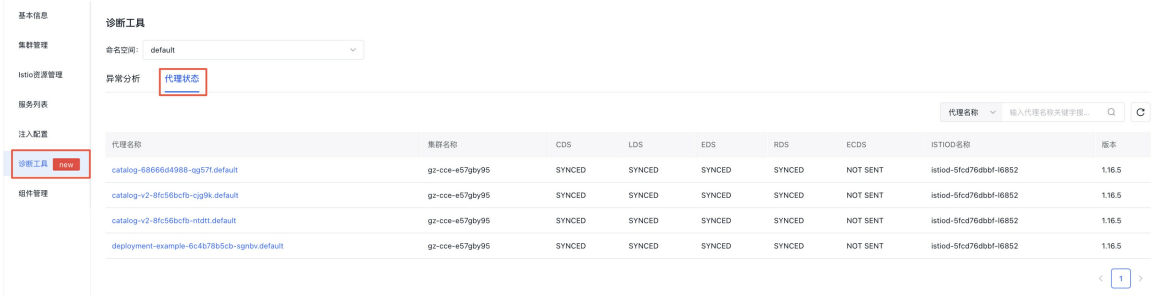
状态信息如下：

- SYNCED：表示 Envoy 确认了 istiod 发过来的配置。
- NOT SENT：表示 istiod 还没有发送配置到 Envoy。通常是因为 istiod 当前没有需要发送的配置信息。
- STALE：表示 istiod 发送了一个更新到 Envoy，但没有接收到确认。通常表示 Envoy 和 istiod 之间的网络出现了问题，或 istio 本身出现了bug。

查看代理状态

1. 在【诊断工具】页点击“代理状态”，即可进入页面。

该页面展示了相应命名空间下代理的 XDS 通讯状态。通过该页面可以获取网格的概述并确定导致问题的代理。



Envoy 配置分析

功能简介

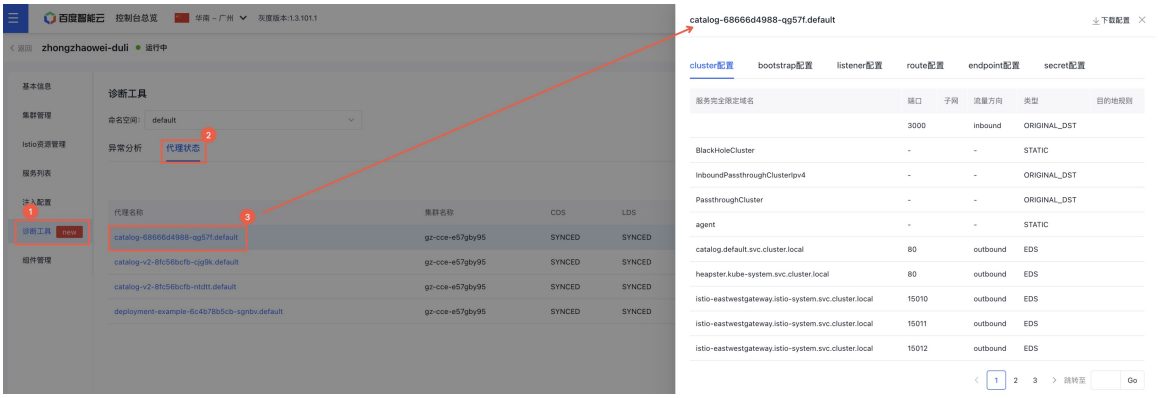
该功能可以查看一个 Envoy 实例的具体配置信息，用于定位无法通过查看 istio 配置和用户资源发现的问题。

包括

- cluster：Pod 中 Envoy 代理的集群配置，包括 Envoy 如何连接到其他集群中的服务等信息。
- bootstrap：Pod 中 Envoy 代理的引导（bootstrap）配置，包含 Envoy 启动所需的基本配置信息。
- listener：Pod 中 Envoy 代理的监听器（Listener）配置，包括 Envoy 如何监听传入连接的配置。
- route：Pod 中 Envoy 代理的路由配置，包括请求应该如何路由到不同服务的信息。
- endpoint：Pod 中 Envoy 代理的端点（Endpoint）配置，包含可访问的后端服务的地址和端口。
- secret：Pod 中 Envoy 代理的密钥配置，包括 Envoy 使用的加密密钥等信息。

查看Envoy配置

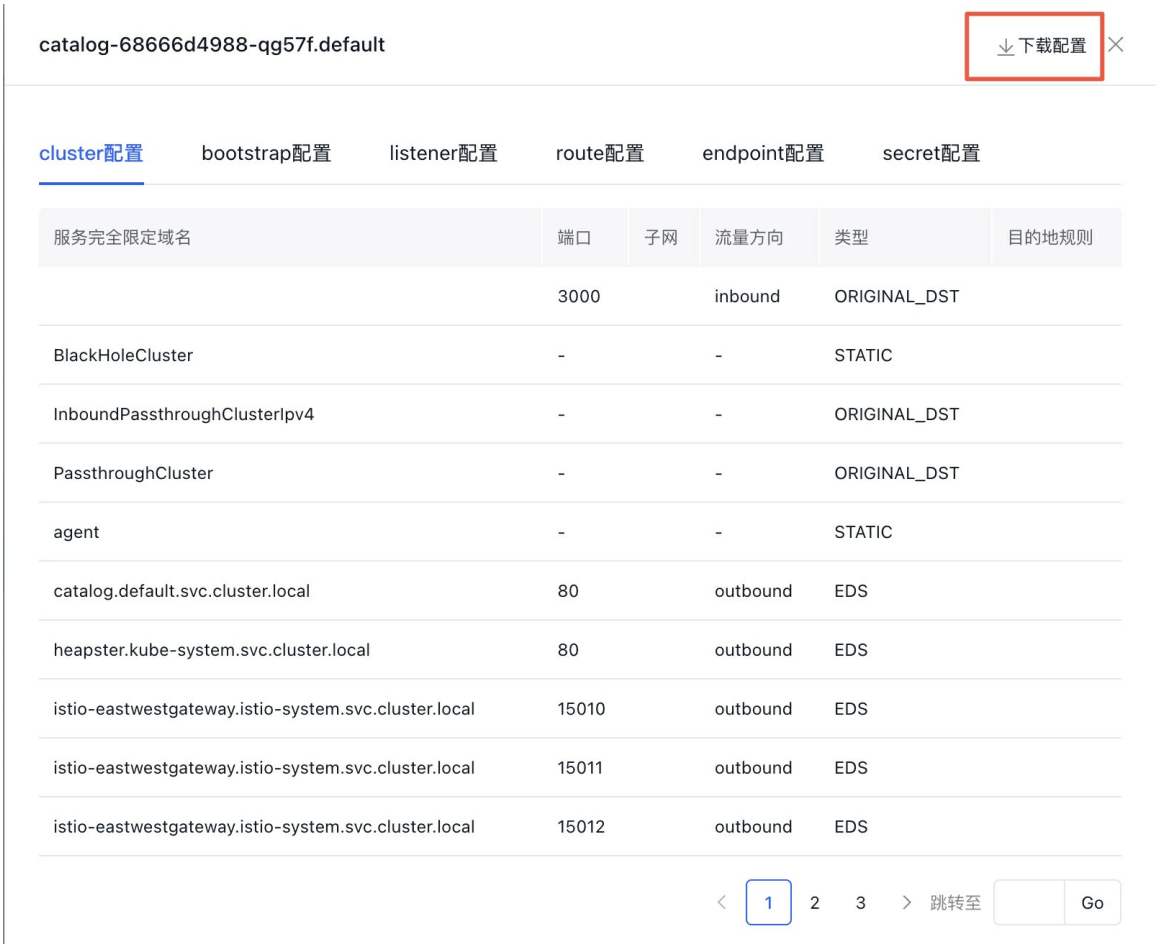
1. 在【诊断工具】页点击“代理状态”
2. 选择具体某一代理 点击进入，即可查看 envoy 实例具体配置信息。



下载配置

- 该功能可实现 istio-proxy 中 config_dump 一键下载。

在 查看 Envoy 配置页 ，点击右上角下载配置即可完成。


catalog-68666d4988-qg57f.default
下载配置
cluster配置
bootstrap配置
listener配置
route配置
endpoint配置
secret配置
服务完全限定域名 端口 子网 流量方向 类型 目的地规则
3000 inbound ORIGINAL_DST
BlackHoleCluster - - STATIC
InboundPassthroughClusterIpv4 - - ORIGINAL_DST
PassthroughCluster - - ORIGINAL_DST
agent - - STATIC
catalog.default.svc.cluster.local 80 outbound EDS
heapster.kube-system.svc.cluster.local 80 outbound EDS
istio-eastwestgateway.istio-system.svc.cluster.local 15010 outbound EDS
istio-eastwestgateway.istio-system.svc.cluster.local 15011 outbound EDS
istio-eastwestgateway.istio-system.svc.cluster.local 15012 outbound EDS
1 2 3 跳转至 Go