服务网格

操作指南

腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

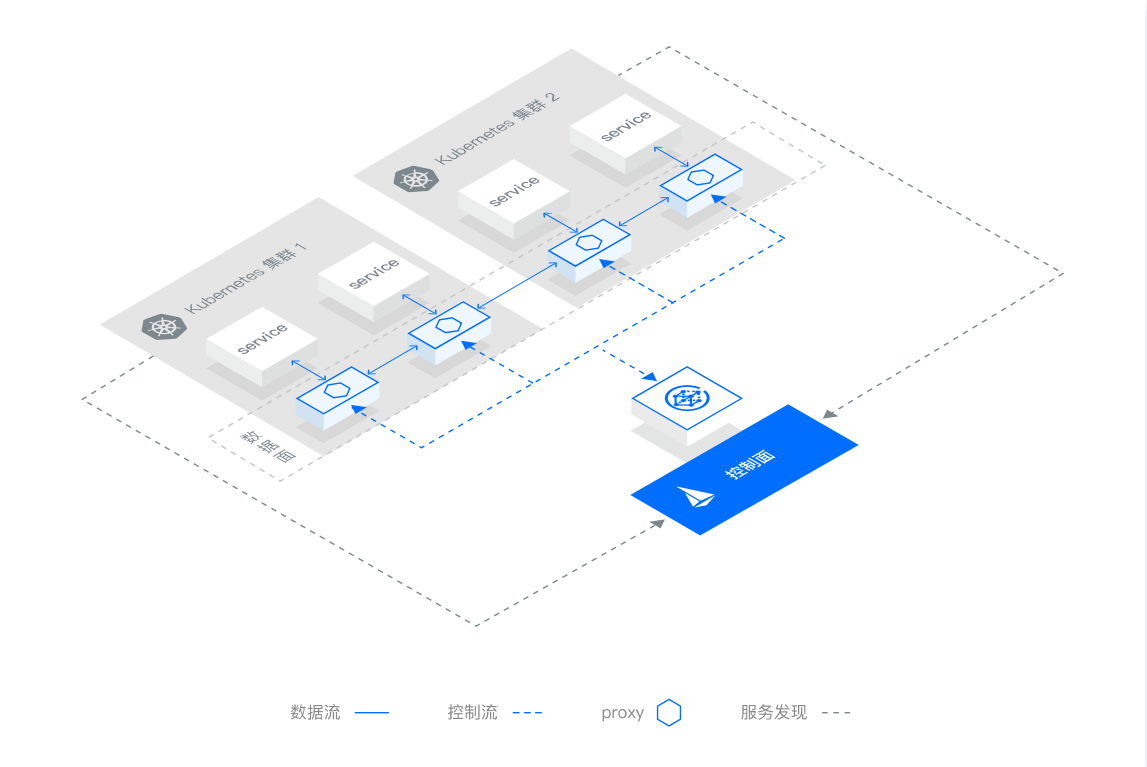
我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

操作指南
网格实例管理
概述

最近更新时间：2024-03-22 14:07:41

网格（mesh）实例，是管理服务的逻辑隔离空间，同一网格内的服务之间可通信。



以下是网格生命周期状态说明：

状态	说明
创建中	网格创建过程中，网格详情无法查看。
运行中	网格正常运行状态。
升级中	网格正在进行版本升级，部分功能无法使用。
闲置	当网格管理的服务发现集群被删除或解关联后，将进入闲置状态（idle），闲置状态的网格可正常查看，但由于没有业务实体，部分功能将不可用。您可以为网格重新添加服务发现集群使其恢复正常状态。
无效	当网格处于闲置状态超过30天后，或独立网格主集群被删除，网格将进入无效状态，您将不再能对网格进行删除之外的其他操作。
异常	网格中部分组件异常导致网格功能已经受到影响。

网格创建过程中需要进行以下几部分配置：

- **添加服务发现集群**
  通过添加服务发现的 Kubernetes 集群自动发现集群内的服务实现，也可通过手动注册服务实现。网格中已发现的服务将出现在**服务网格控制台 > 网格详情 > 服务**列表页中。服务被发现后，可以被网格中其他服务访问。详细指引请参见 服务发现管理 。
- **创建边缘代理网关**
  边缘代理网关分为 Ingress 和 Egress 两种类型，是网格流量的出入口，其中 Ingress 类型的边缘代理网关必须创建，流量才能进入网格，Egress 类型边缘代理网关可选创建，详细指引请参见 边缘代理网关管理 。
- **为服务注入 Sidecar**
  Sidecar 容器承担数据面流量管理、规则生效、监控上报等网格治理工作，是网格流量治理和观测的基础，因此对于需要进行流量管理和观测的服务，需要为其注

入 Sidecar，详细指引请参见 网格配置 。

- **可观测性后端服务配置**
  可观测性分为监控指标查看、链路追踪、日志管理三部分。服务网格支持与腾讯云 Prometheus 监控 TMP 、应用性能监控 APM、日志服务 CLS 集成，提供更丰富、一体化的可观测能力，同时服务网格也支持对接第三方 Prometheus、Jaeger/Zpkin 服务，为用户提供更大的组件扩展性。详细指引请参见 可观测性 。

网格创建后，对网格进行流量规则调度，可以通过控制台或提交 YAML 文件配置的形式为网格创建流量治理规则，当前 TCM 完全兼容 Istio 原生语法，详细指引请参见 流量管理 。

创建网格

最近更新时间：2025-06-10 15:11:52

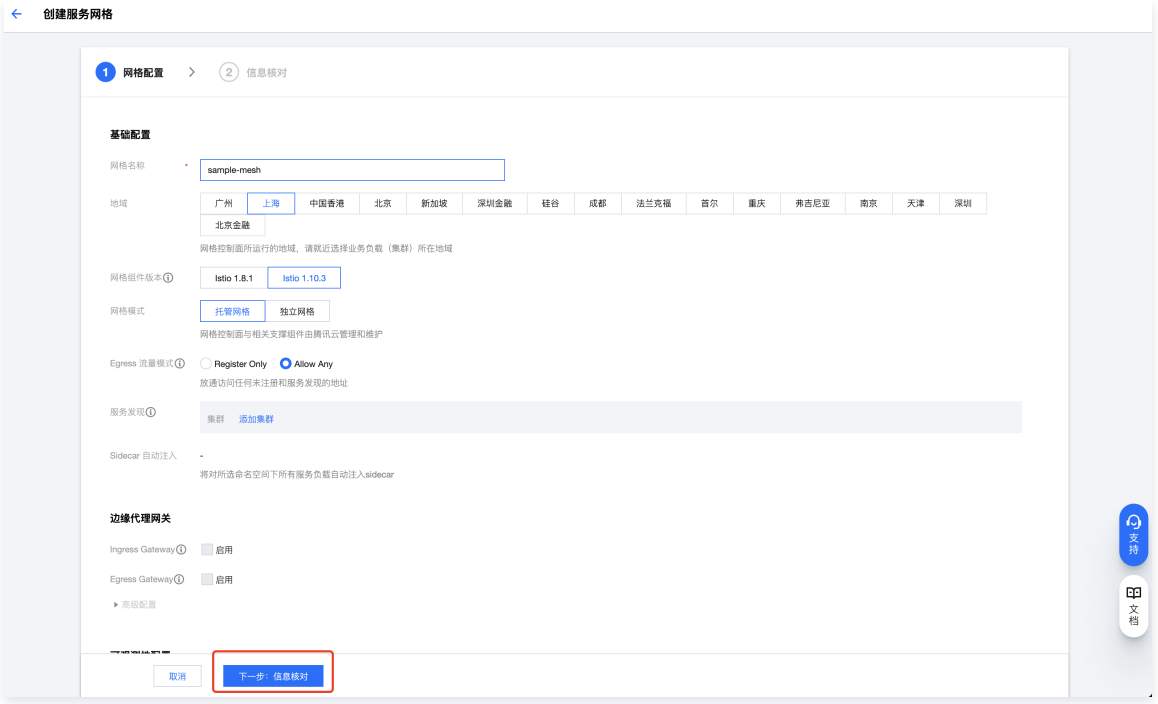
使用服务网格，首先需要创建一个服务网格实例。网格实例有地域属性，但可以管理多个地域的服务。本文介绍如何通过控制台创建服务网格实例。

> **说明：**
> 每个账号默认允许创建20个网格，如果需要创建更多个数，可以通过 提交工单 申请。

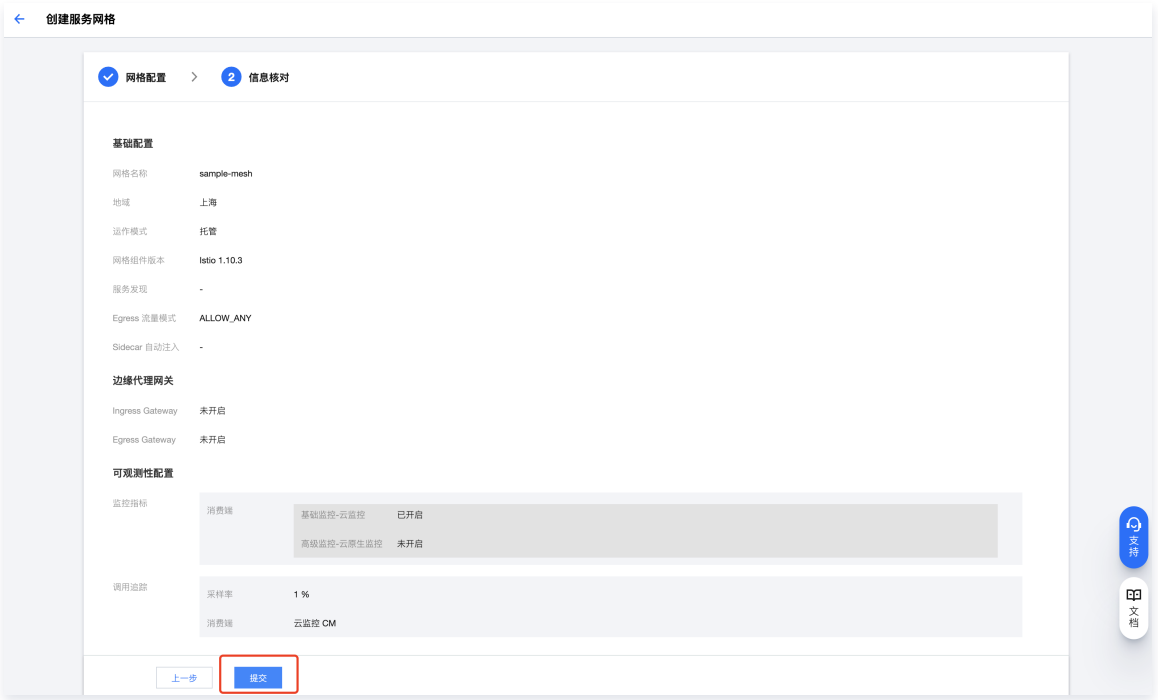
操作步骤

以下是新建服务网格实例的控制台操作流程：

1. 登录 服务网格控制台 。
2. 选择地域，单击页面左上角的**新建**。
3. 在创建服务网格页面，按需填写网格创建相关配置，配置项说明请参见 创建网格配置项说明 ，完成后单击**下一步：信息核对**。如下图所示：



4. 在信息核对页面确认创建配置无误后单击**提交**，即可开始网格创建流程。



5. 网格创建流程完成后，即可在服务网格列表页查看服务网格实例。

创建网格配置项说明

配置项	描述	是否必填
网格名称	创建的服务网格的名称。	是
地域	服务网格控制面运行的地域，控制面运行地域可与业务负载（例如集群）地域不同，建议就近选择业务负载（集群）所在地域。	是
网格组件版本	选择控制面和数据面的版本，服务网格 TCM 提供支持兼容 Istio 社区的最新两个大版本。	是
网格模式	选择网格控制面相关组件的部署模式，托管网格控制面相关组件由腾讯云管理和维护，独立网格控制面相关组件会部署在您指定的集群内，您需要管理和维护集群内的控制面组件，默认可选托管网格，独立网格需要开白名单使用，您可以通过 在线咨询 申请使用。	是
Egress 流量模式	配置网格内服务对外访问的放通策略，可选择 Registry Only（仅支持访问网格自动发现的服务与手动注册的服务）或 Allow Any（可访问任何地址）。	是
服务发现	指定服务网格自动服务发现的集群，集群需满足版本、权限、网段冲突等约束条件。	否
Sidecar 自动注入	配置自动注入 Sidecar 的 Namespace，开启后将对所选命名空间下的所有服务负载自动注入 Sidecar，自动注入仅对新创建的服务负载生效，存量服务负载需要重启后才会注入 Sidecar。如果需要进一步自定义 Sidecar 注入的例外情况，详情见 自定义 Sidecar 注入。	否

外部请求绕过 Sidecar	对应 excludeIPRanges ，默认情况下，Sidecar 会接管当前 Pod 内所有的流量，如果想让针对特定的 IP 的访问不经过 Sidecar 代理，则可以配置此选项，配置后对该 IP 范围的请求流量将无法使用 Istio 流量管理、可观测性等特性。配置变更后仅对新增 Pod 生效，存量 Pod 需重启后生效。	否
Sidecar 就绪保障	使用 HoldApplicationUntilProxyStarts 功能 ，配置业务容器等待 Sidecar 完成启动后再启动，确保业务容器运行依赖 Sidecar 的 Pod 正常运行。	否
Sidecar 停止保障	开启后 Sidecar 停止需要等待业务容器中进程完全终止，将一定程度增加 Pod 停止时长，建议对业务进程无法随时关闭的服务开启。对于 1.12 之前的 Istio 版本，TCM 使用预置的容器 prestop 脚本检查不再有业务进程后才允许业务容器退出，这也意味着用户如果配置其他的 prestop 脚本将对此功能产生干扰。对于 1.12 之后的版本，使用新特性 EXIT_ON_ZERO_ACTIVE_CONNECTIONS 来实现。	否
自定义 Sidecar 资源	默认情况下，TCM 为 Sidecar container 配置最高 2 核 1G 的资源限制，这在大部分情况下是足够的。当您的网格规模扩大或 Sidecar 中逻辑增多时，默认资源限制可能不足，您可以自行根据业务需要修改资源限制。	否
Ingress Gateway	为网格创建 Ingress Gateway，如果是 TKE/EKS 集群，则默认创建 CLB 类型的 Ingress Gateway，您需要配置 CLB 创建相关选项。如果是注册集群，由于不确定集群是否能使用腾讯云 CLB ，因此仅创建 LoadBalancer 类型的 Gateway Service。	否
Egress Gateway	如果您需要对网格的出流量进行集中管理，如，统一出口，统一鉴权，规则配置等，则需要创建 Egress Gateway 。 后台将为您创建一个 ClusterIP 类型的 Egress Gateway service 。	否
网关部署模式	可以选择普通部署或专有部署模式，详情见 边缘代理网关部署模式 。	否
网关伸缩策略	配置部署在指定集群内的边缘代理网关的 HPA 策略。	否
网关资源定义	自定义 Ingress/Egress Gateway 的 pod 资源限制。	否
监控指标消费端	配置网格的监控指标后端服务，目前支持对接 Prometheus 监控 TMP，配置后监控指标将上报到 TMP，TCM 控制台指标基于 TMP 数据源展示，用户也可以在 TMP 控制台独立使用。不配置监控指标消费端，网格将无法使用监控指标展示、拓扑图等相关监控功能。	否
调用追踪消费端	配置网格的调用追踪后端服务，目前支持对接应用性能监控 APM，配置后 Tracing 数据将从 Sidecar 将上报到 APM, TCM 控制台指标基于 APM 数据源展示，用户也可以在 APM 控制台独立使用。不配置调用追踪消费端，网格将无法使用调用链查看相关功能。	否
调用追踪采样率	网格采集并持久化调用 Tracing 的采样比例。Sidecar 采集与上报数据消耗资源与带宽和数据量成正相关，请按需配置，开发测试环境建议100%，生产环境建议1%。	否
访问日志开启范围	为避免不必要的开销，TCM 支持针对特定网关或者特定 Namespace 开启 Sidecar 日志。	否
访问日志日志格式	TCM 支持 Json 或 Txt 两种格式的日志格式化。	否
访问日志输出模板	Sidecar 日志字段设置，有默认、增强两种预定义模板可选，增强在默认格式基础上增加了 TraceID 字段输出，如果用户需要进一步更改字段设置，可按照 Envoy 标准规范 自定义日志字段。	否
访问日志消费端	配置 Sidecar 日志的后端服务，目前支持对接日志服务 CLS，开启后会在集群节点上部署日志采集组件以确保功能正常使用。	否

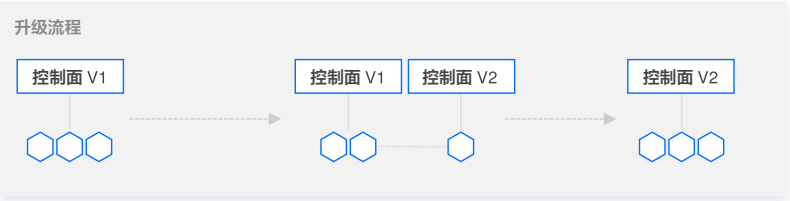
升级网格

最近更新时间：2024-03-22 14:07:41

TCM 提供网格升级服务， 用户可将低版本的网格升级到高版本。升级过程遵循灰度升级原则，分为以下几步：

1. TCM 部署新版本控制面，完成控制面升级。
2. 数据面灰度升级，用户重启业务，使存量业务 Pod 完成 Sidecar 更新。
3. 升级验证，验证升级后业务是否正常。
4. 老版本控制面下线，升级完成。

在老控制面下线前，可以回滚到升级前的状态。升级流程如下图所示：



操作步骤

1. 登录 服务网格控制台 。
2. 当网格版本可升级时，网格界面将会提示有可升级的新版本，如下图所示：

3. 选择**更多 > 升级**后，根据引导进行升级。
   升级将按照**控制面升级 > 数据面升级 > 旧控制面下线**三个阶段进行，在旧控制面下线前，可以回滚到升级前的状态。

控制面升级

控制面升级，TCM 将部署新版本控制面组件：

数据面升级

数据面升级，分为业务数据面升级和 Gateway 升级。

其中业务数据面升级是用户将指定 Namespace 的 Sidecar 自动注入改为使用新版本，勾选后，该 Namespace 下**新创建**的业务 Pod 将注入新版本的 Sidecar，**存量的业务 Pod 重建后才会更新为新版本**，由于重启涉及到业务可用性影响，平台不会自动重建业务 Pod，**需要用户手动重建**。

> **说明：**
> - 您可以通过流水线重新发布业务或直接使用 kubectl patch、kubectl rollout restart 等命令行工具手工重建工作负载。
> - 部分场景下 Sidecar 将被卸载而不是升级，例如：Namespace 曾开启过 Sidecar 注入，并且部分业务 Pod 已成功注入 Sidecar，之后又关闭了 Namepsace 级 Sidecar 注入，则重启业务 Pod 后，除非为 Pod 单独设置了 Sidecar 注入标记，否则 Sidecar 将被卸载。

Gateway 升级，将所有需要升级的 Gateway 组件勾选到新版本并单击右侧的**升级**。如下图所示：

所有数据面升级完成后，单击**升级**进入下一步。

升级验证

由于版本升级涉及到特性变更，可能存在兼容性问题，业务 Pod 重建后，您需要检查业务是否正常。如果发现升级导致业务异常，您可以在升级界面中选择回滚，将数据面 Sidecar 退回到原版本。

4. 选择**完成**或**取消升级**。在“数据面升级”中，单击**升级**或**回滚**将检查当前存量 Pod 是否满足进入下一步的条件。当所有 Namespace 均切换到新版本控制面，并且所有存量业务 Pod 中 Sidecar 均已更新到新版本后，可以选择**升级**，进入下一步**下线旧版本控制面**完成升级。或当所有的 Namespace 均切换回旧版本控制面，并且所有存量业务 Pod 中 Sidecar 均使用旧版本控制面时，可以单击**回滚**进入下一步将新版本控制面下线，取消升级。

更新网格配置

最近更新时间：2024-08-19 15:44:51

您可以更新运行中服务网格的配置，本文将介绍如何更新网格配置。

修改 Egress 流量模式

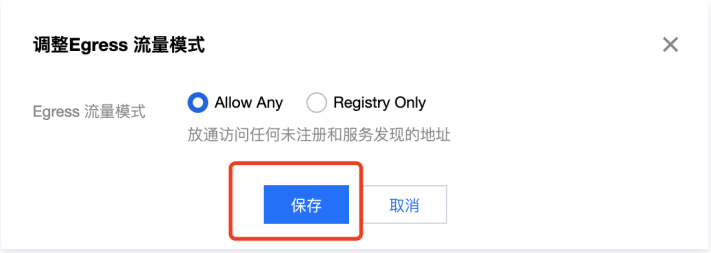
Egress 流量模式是配置网格内服务对外访问的放通策略，可选择 Registry Only（仅支持访问网格自动发现的服务与手动注册的服务）或 Allow Any（可访问任何地址）。

以下是配置网格 Egress 流量模式的步骤：

1. 登录 服务网格控制台，单击需要变更配置的网格 ID，进入网格的管理页面。

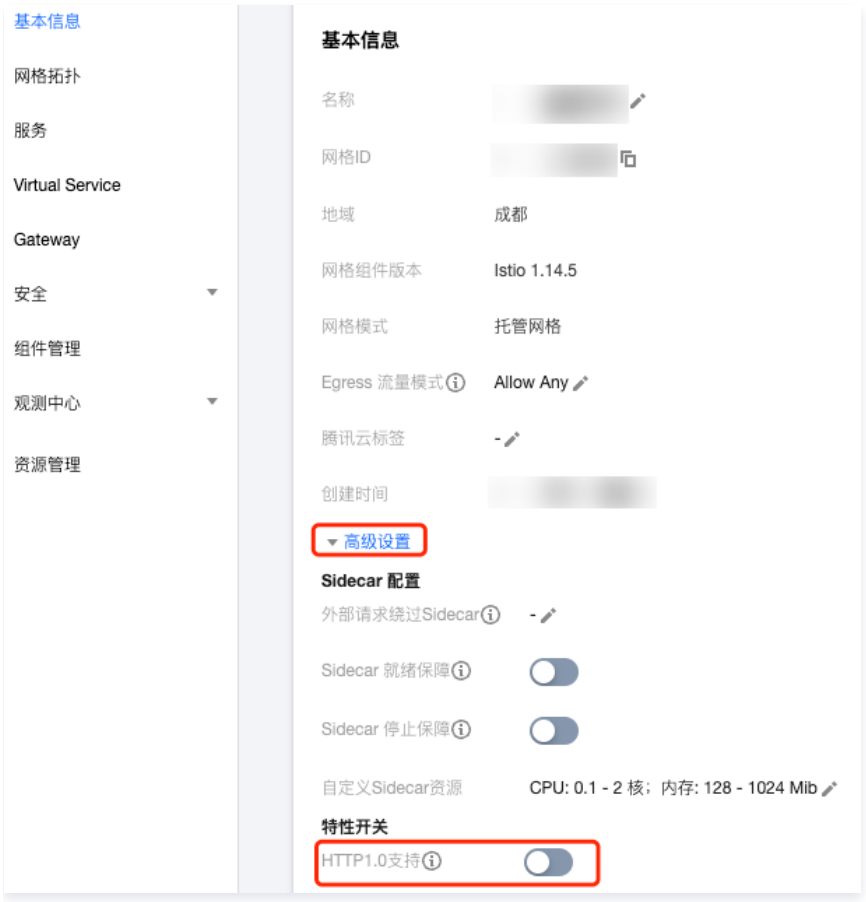
2. 在网格基本信息页面单击 Egress 流量模式栏的编辑按钮，进入**调整 Egress 流量模式**弹窗。

3. 按照需要选择 **Allow Any** 或 **Registry Only**，单击**保存**更新 Egress 流量模式。



启用 HTTP1.0 支持

Istio 默认不支持 HTTP 1.0，如有需要，可以在网格基本信息中开启：



关闭 HTTP 自动重试

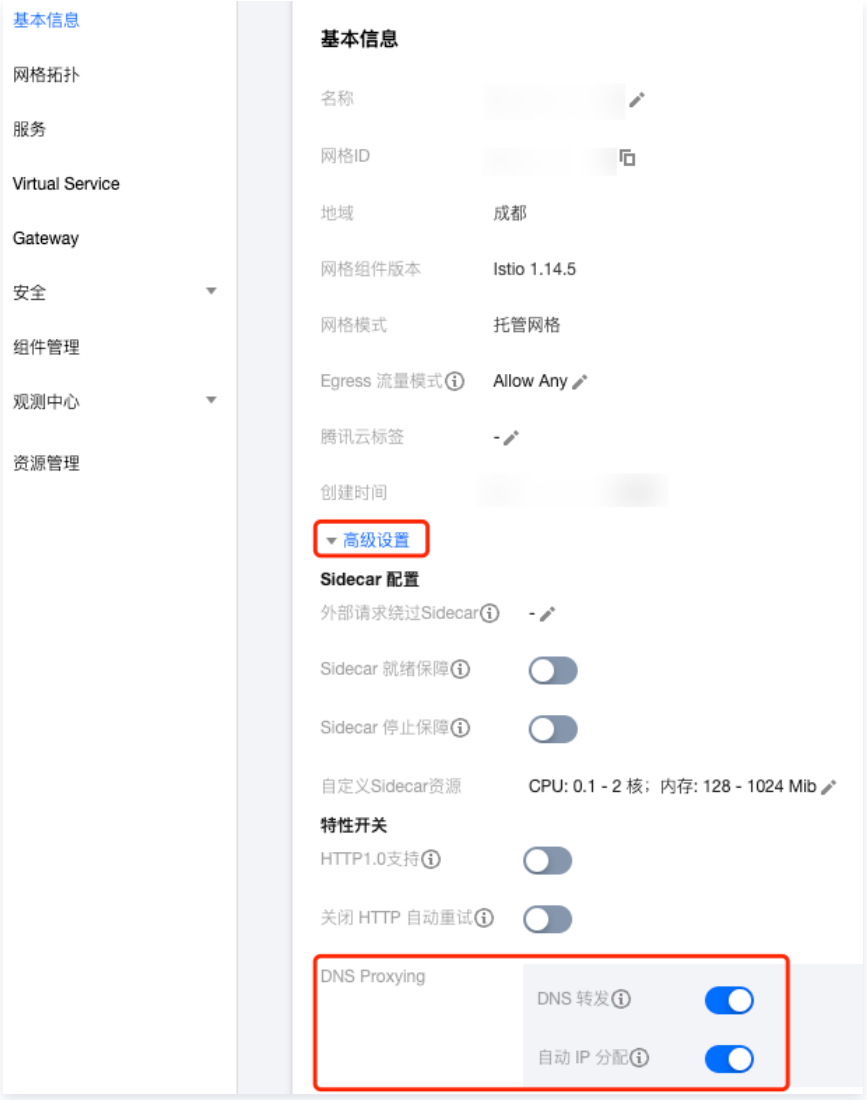
对于失败的 HTTP 请求，Istio 默认会重试2次，某些情况下这不符合您的预期，可以在网格基本信息页里关闭自动重试：

关闭对全网格生效，但您仍然可以对特定的 Virtual Service 显式设定重试策略。

启用 DNS 代理

istio 的 Sidecar 支持 DNS 代理，启用后，DNS 的流量也会被拦截，由 Sidecar 直接响应 DNS 请求，一方面 sidecar 会有 DNS 缓存，可以加速 DNS 响应，另一方面，多集群网格的场景，跨集群访问 service 时，不需要在 client 所在集群创建同名 service 也能正常解析 service。您可以参考以下步骤开启 DNS 转发。

1. 登录 服务网格控制台，单击需要变更配置的网格 ID，进入网格的管理页面。
2. 在基本信息页面，单击 **DNS Proxying > DNS 转发**右侧的，开启 DNS 转发。如下图所示：



如果您需要为没有定义 addresses 的 ServiceEntry 自动分配 IP，可以开启**自动 IP 分配**。更多内容可参见 Address auto allocation 。

Sidecar 注入与配置

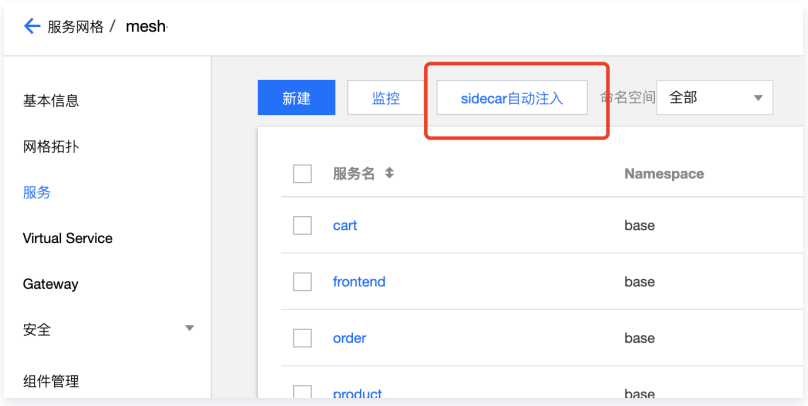
最近更新时间：2024-05-06 14:31:01

配置 Sidecar 自动注入

TCM 当前支持在控制台为指定的 Namespace 开启 Sidecar 自动注入，开启后该 Namespace 下新创建的 workload 将会自动安装网格 Sidecar，由于注入是在 workload 创建过程中完成的，因此开启注入无法为已存在的 workload 自动安装 Sidecar，您可以通过重建 workload 完成 Sidecar 自动注入。
以下是配置 Namespaces 级 Sidecar 自动注入的步骤：

1. 登录 服务网格控制台，单击需要变更配置的网格 ID，进入网格的管理页面。

2. 在服务列表页点击单击 **Sidecar 自动注入**，进入 **Sidecar 自动注入配置**弹窗。



3. 按需勾选需要 **Sidecar 自动注入的 namespace**，单击**确定**完成 Sidecar 自动注入配置。

如果不想通过控制台配置 Sidecar 自动注入，您也可以修改 namespace 的 yaml，添加 label `istio.io/rev: {istio 版本号}`：

```
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: "2024-03-05T09:52:56Z"
  labels:
    istio.io/rev: 1-18-5 # 注意修改版本号
    kubernetes.io/metadata.name: test
  name: test
  resourceVersion: "29528054435"
```

```
  uid: 9a7491bb-2f3d-4017-bf95-2a21ebb31ba4
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

> **注意：**
> 不能与社区默认的 `istio-injection: enabled` 这样的 label 同时使用，否则将不会生效！

自定义 Sidecar 注入

TCM 也支持您通过编辑 yaml 为特定的工作负载开启 Sidecar 自动注入，如有需要，您可以在 Pod 上添加 label：`istio.io/rev：{istio 版本号}`（注意 Sidecar 注入相关的标签设置，TCM 与 istio 默认语法略有区别），示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        istio.io/rev: 1-14-5
    spec:
      containers:
      - name: nginx
        image: nginx:latest
```

如果您需要为已经开启了自动注入的 Namespace 下的特定 Pod 添加特例，使其不自动注入 Sidecar，可在 Pod label 中添加：`sidecar.istio.io/inject="false"`。Pod 级别的注入开关优先级高于 Namespace 级别，关于 Sidecar 自动注入的更多细节，请参考 Istio 文档 安装 sidecar。

不拦截指定网段的流量

如果您不希望拦截某些流量，例如上传文件到 COS 对象存储存储的流量（169.254 开头的内网目标 IP），如果被拦截，且下载的文件较大，则可能导致 Sidecar 内存资源占用很高，因为 Sidecar 会缓存请求内容，出错自动重试时会复用请求内容。这时，您可以在网格基本信息页中的高级设置中配置**外部请求绕过Sidecar**，添加不希望拦截的网段。操作步骤如下：

1. 登录 服务网格控制台，单击需要变更配置的网格 ID，进入网格的管理页面。

2. 在基本信息页面，单击**外部请求绕过 Sidecar** 右侧的 。如下图所示：

3. 在**编辑外部请求绕过 Sidecar** 弹窗中，添加您不希望拦截的网段。如下图所示：

4. 单击**保存**。

以上是全局配置，如果希望只针对某些工作负载，可以给 Pod 添加注解 `traffic.sidecar.istio.io/excludeOutboundIPRanges` 。详情可参见 注解列表。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        'traffic.sidecar.istio.io/excludeOutboundIPRanges': '169.254.0.0/16'
      labels:
        app: nginx
```

控制 sidecar 启动顺序

Kubernetes 拉起 Pod 的时候是各个容器同时拉起，使用 istio 的场景，因为流量会被 Sidecar 拦截，当遇到 Sidecar 比业务容器启动更慢的时候，业务容器刚启动时的网络请求将会失败，因为流量被 Sidecar 拦截但 Sidecar 还没启动就绪。比较常见的场景是：集群规模较大，Sidecar 启动时拉取 xds 较慢导致 Sidecar 启动较慢，而业务进程启动时要从注册中心拉取配置，由于流量被 Sidecar 拦截而 Sidecar 此时还无法处理流量导致配置拉取失败，然后业务进程报错退出，导致容器退出。

主要有两种解决方案，第一种是让业务代码更健壮，启动时请求失败要不断重试，直至成功；第二种是让 Sidecar 先启动，等 Sidecar 就绪后再拉起业务容器。您可以参考以下步骤开启 Sidecar 就绪保障。

1. 登录 服务网格控制台，单击需要变更配置的网格 ID，进入网格的管理页面。

2. 在基本信息页面，单击 **Sidecar 就绪保障**右侧的 。如下图所示：

以上是全局配置，如果只想针对某些工作负载，可以为 Pod 添加如下注解：

```
proxy.istio.io/config: '{ "holdApplicationUntilProxyStarts" : true }'
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        proxy.istio.io/config: '{ "holdApplicationUntilProxyStarts" : true }'
      labels:
        app: nginx
```

Sidecar 优雅终止

业务发版时，工作负载滚动更新，Pod 在终止过程中，Sidecar 默认只等待几秒然后就会强制停止，如果业务请求本身耗时较长，或者使用长连接，可能会导致部分正常业务请求和连接中断，我们希望 Sidecar 会等待存量的业务请求和连接结束再退出以实现优雅终止。自 istio 1.12 开始，Sidecar 支持了 `EXIT_ON_ZERO_ACTIVE_CONNECTIONS` 这个环境变量，作用就是等待 Sidecar “排水” 完成，在响应时，也通知客户端去关闭长连接（对于 HTTP1 响应 “Connection: close” 这个头，对于 HTTP2 响应 GOAWAY 这个帧）。您可以参考以下步骤开启 Sidecar 停止保障。

1. 登录 服务网格控制台，单击需要变更配置的网格 ID，进入网格的管理页面。

2. 在基本信息页面，单击 **Sidecar 停止保障**右侧的 。如下图所示：

以上是全局启用的方法，通常也建议全局启用，如果只想针对某些工作负载开启，也可以为 Pod 添加如下注解：

```
proxy.istio.io/config: '{ "proxyMetadata": { "EXIT_ON_ZERO_ACTIVE_CONNECTIONS": "true" } }'
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        proxy.istio.io/config: '{ "proxyMetadata": { "EXIT_ON_ZERO_ACTIVE_CONNECTIONS": "true" } }'
      labels:
        app: nginx
```

自定义 sidecar 资源

Sidecar 作为 Pod 中的一个 container，也需要设置 request 和 limit，如有需要，也可以自定义，您可以在网格基本信息页的高级设置中可以进行全局自定义。操作步骤如下：

1. 登录 服务网格控制台，单击需要变更配置的网格 ID，进入网格的管理页面。

2. 在基本信息页面，单击**自定义 Sidecar 资源**右侧的 。如下图所示：

3. 在**编辑自定义 Sidecar 资源**弹窗中，编辑自定义资源。如下图所示：

4. 单击**保存**。

如果希望对不同工作负载进行不同的自定义，也可以为 Pod 添加类似如下的注解：

```
sidecar.istio.io/proxyCPU: "0.5"
sidecar.istio.io/proxyCPULimit: '2'
sidecar.istio.io/proxyMemory: "256Mi"
sidecar.istio.io/proxyMemoryLimit: '2Gi'
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        sidecar.istio.io/proxyCPU: "0.5"
        sidecar.istio.io/proxyCPULimit: '2'
        sidecar.istio.io/proxyMemory: "256Mi"
        sidecar.istio.io/proxyMemoryLimit: '2Gi'
      labels:
        app: nginx
```

如果使用了 TKE Serverless，不希望因 Sidecar 的 request 和 limit 导致 Pod 规格变高很多，可以用注解只设置 request 不设置 limit，request 根据实际需要填写，填 “0” 就表示 Pod 规格完全不会因 Sidecar 而升配：

```
sidecar.istio.io/proxyCPU: "0"
sidecar.istio.io/proxyMemory: "0"
```

删除网格

最近更新时间：2024-04-10 16:37:31

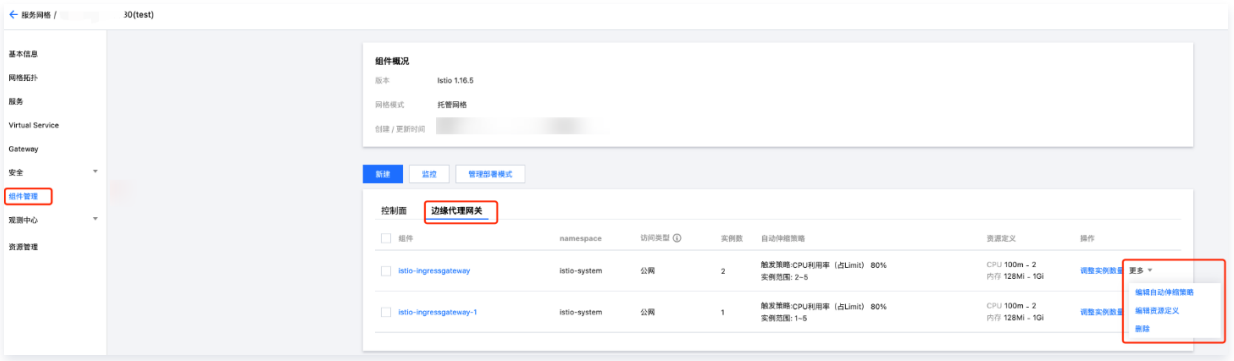
本文档介绍服务网格实例删除的操作方法。

操作步骤

> **注意：**
> 为了避免误操作，您需要在删除网格之前，确保**网格中创建的边缘代理网关全部被删除**，同时需要将**网格中关联的集群全部解关联**，才能执行网格的删除操作。

删除边缘代理网关

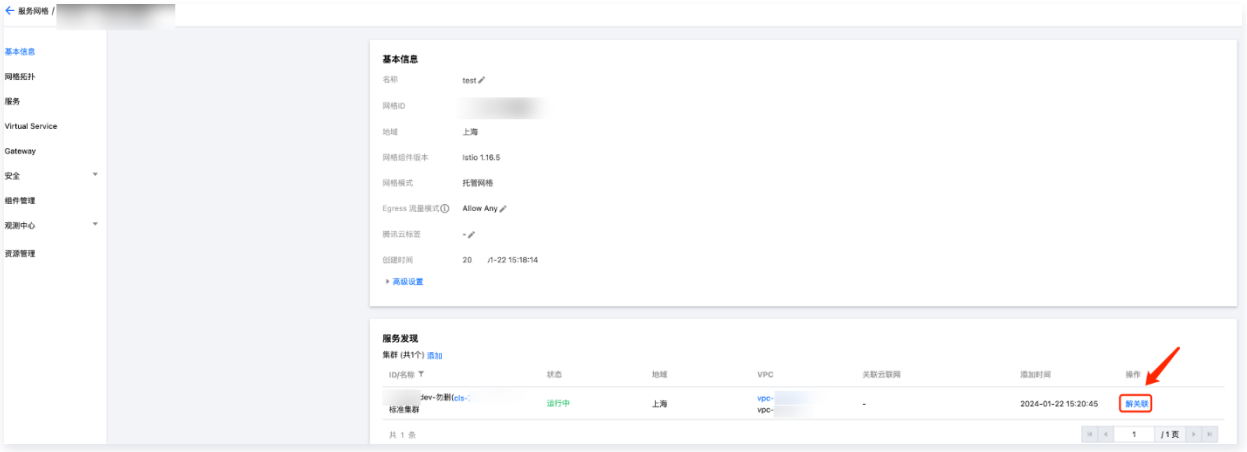
1. 登录 服务网格控制台，进入网格列表页。
2. 在页面顶部选择服务网格所属地域。
3. 单击 mesh 名称，进入详情页，选择**组件管理**。如下图所示：



4. 在**边缘代理网关**中，选择组件右侧的**更多 > 删除**。

解关联集群

1. 在网格列表页，单击 mesh 名称，进入基本信息页。
2. 在**服务发现**中，单击集群右侧的**解关联**。如下图所示：



删除网格实例

在网格列表中，单击 mesh 右侧操作中的**删除**，并确认操作。


服务网格
地域
上海
扫码关注公众号
新建
一键体验
多个关键字用竖线 "|" 分隔，多个过滤标签用回车键
ID/名称
监控
状态
网格组件版本
网格模式
服务数量
集群
操作
mesh
运行中
Istio 1.10.3
托管网格
0
-
删除

使用命令行工具管理网格

最近更新时间：2024-03-22 14:07:41

工具简介

Istio 社区提供了命令行工具 istioctl 帮助用户管理网格，但对于托管在 TCM 的网格实例，受限于托管架构以及托管服务本身的限制，并不支持直接使用 Istioctl，因此我们提供了供托管网格使用的命令行工具 tcmctl。

前置条件

1. 已创建托管网格，并添加 TKE 标准集群。
2. 使用环境可通过 kubectl 连接任一被网格管理的集群。（具备正确的 kubeconfig 配置）

兼容性

以下为 tcmctl 与 istioctl 差异对比：

命令类型	istioctl(1.12) 命令	功能	兼容情况
基础命令	admin	Manage control plane (istiod) configuration	不支持，托管控制面不支持用户自定义修改配置
	analyze	Analyze Istio configuration and print validation messages	支持
	authz	(authz is experimental. Use istioctl experimental authz)	支持
	bug-report	Cluster information and log capture support tool.	支持
	completion	generate the autocompletion script for the specified shell	支持
	create-remote-secret	Create a secret with credentials to allow Istio to access remote Kubernetes apiservers	不支持，托管控制面不允许自行为控制面创建 Secret
	dashboard	Access to Istio web UIs	部分支持，受限于控制面托管，仅支持 dashboard envoy 命令
	experimental	Experimental commands that may be modified or deprecated	部分支持，详情见“实验命令支持情况”
	help	Help about any command	支持
	install	Applies an Istio manifest, installing or reconfiguring Istio on a cluster.	不支持
	upgrade	Upgrade Istio control plane in-place	不支持，托管网格不支持通过命令管理实例生命周期
	verify-install	Verifies Istio Installation Status	支持
	manifest	Commands related to Istio manifests	支持
	operator	Commands related to Istio operator controller.	支持
	profile	Commands related to Istio configuration profiles	支持
	kube-inject	Inject Istio sidecar into Kubernetes pod resources	不支持，TCM 注入机制由控制面统一管理，不支持命令注入
	proxy-config	Retrieve information about proxy configuration from Envoy [kube only]	支持
	proxy-status	Retrieves the synchronization status of each Envoy in the mesh [kube only]	支持

	remote-clusters	Lists the remote clusters each istiod instance is connected to.	支持
	tag	Command group used to interact with revision tags	不支持，未使用 tag 机制，不支持 tag 操作
	validate	Validate Istio policy and rules files	支持
	version	Prints out build version information	支持
实验命令	add-to-mesh	Add workloads into Istio service mesh	不支持，托管网格不允许自行控制
	remove-from-mesh	Remove workloads from Istio service mesh	支持
	authz	Inspect Istio AuthorizationPolicy	支持
	create-remote-secret	Create a secret with credentials to allow Istio to access remote Kubernetes apiservers	支持
	describe	Describe resource and related Istio configuration	支持
	injector	List sidecar injector and sidecar versions	不支持，sidecar injector 托管，不支持改显示改信息
	internal-debug	Retrieves the debug information of istio	不支持，不允许对托管控制面进行 debug 操作
	kube-uninject	Uninject Envoy sidecar from Kubernetes pod resources	不支持，注入机制有控制面统一管理，不支持命令删除 sidecar
	metrics	Prints the metrics for the specified workload(s) when running in Kubernetes.	不支持，不允许对控制面核心组件自建可观测性。
	precheck	check whether Istio can safely be installed or upgrade	不支持，控制面托管，不支持社区的检查操作
	version	Prints out build version information	不支持，无法通过 xds-address 来获取 version 信息，请使用“tcmctl version”命令
	proxy-status	Retrieves the synchronization status of each Envoy in the mesh	不支持，无法通过 xds-address 来获取 proxy 状态信息，请使用“tcmctl proxy-status”命令
	config	Configure istioctl defaults	支持
	remote-clusters	Lists the remote clusters each istiod instance is connected to.	支持
	revision	Provide insight into various revisions (istiod, gateways) installed in the cluster	不支持，TCM 未使用 tag 机制，不支持 tag 子命令
	uninstall	Uninstall Istio from a cluster	不支持，托管网格不允许通过命令管理实例生命周期
	wait	Wait for an Istio resource	支持
	workload	Commands to assist in configuring and deploying workloads running on VMs and other non-Kubernetes environments	不支持，sidecar 的注入配置由控制面管理，因此不支持修改操作

对于以上命令行工具 tcmctl 支持的命令，使用语法与社区 istioctl 完全一致，您可以使用 `-help` 查询相关语法。更多关于 istioctl 的使用，可查看 istioctl 文档。

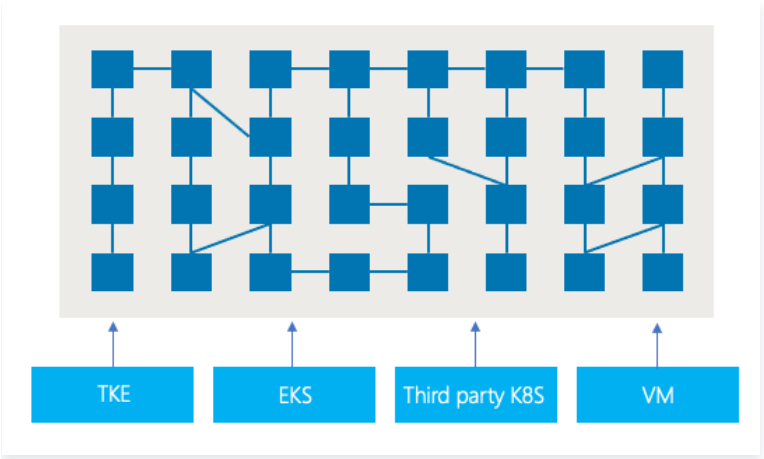
使用方式

1. 下载合适版本的 tcmctl。根据您的使用环境，选择并下载合适的版本：tcmctl-linux，tcmctl-windows，tcmctl-mac。
2. 为工具增加执行权限，并加入到 PATH 环境变量。
3. 连接网格，与 kubectl 类似。默认情况下 tcmctl 使用 `$HOME/.kube/config` 连接网格中的集群，您可以通过配置 `--context` 切换 kubeconfig。

服务发现管理
概述

最近更新时间：2024-03-22 14:07:41

服务发现是将特定的业务服务加入到网格中，是可对服务进行治理和观测的先决条件。TCM 支持自动发现 K8S 集群中的服务和手工注册其他类型的服务，如 VM、云数据库等。在 K8S 集群方面，不仅支持 TKE 标准集群，也支持 TKE 超级节点和 TKE Serverless 集群，同时您也可以通过注册集群的形式将第三方 K8S 集群注册到容器服务控制台，TCM 支持将注册的第三方集群加入网格。



关于如何将 K8S 集群、异构服务添加到 TCM 中，请参考以下文档：

- 自动服务发现
- 手动服务注册

自动服务发现

最近更新时间：2024-03-22 14:07:41

操作场景

一个 TCM 网格支持关联多个 TKE 集群，并自动发现其中的 K8S 服务，您可以在创建网格时或在网格基本信息页中关联多个 TKE 集群，TCM 将自动将集群中的服务展示在**服务**页面中。

使用限制

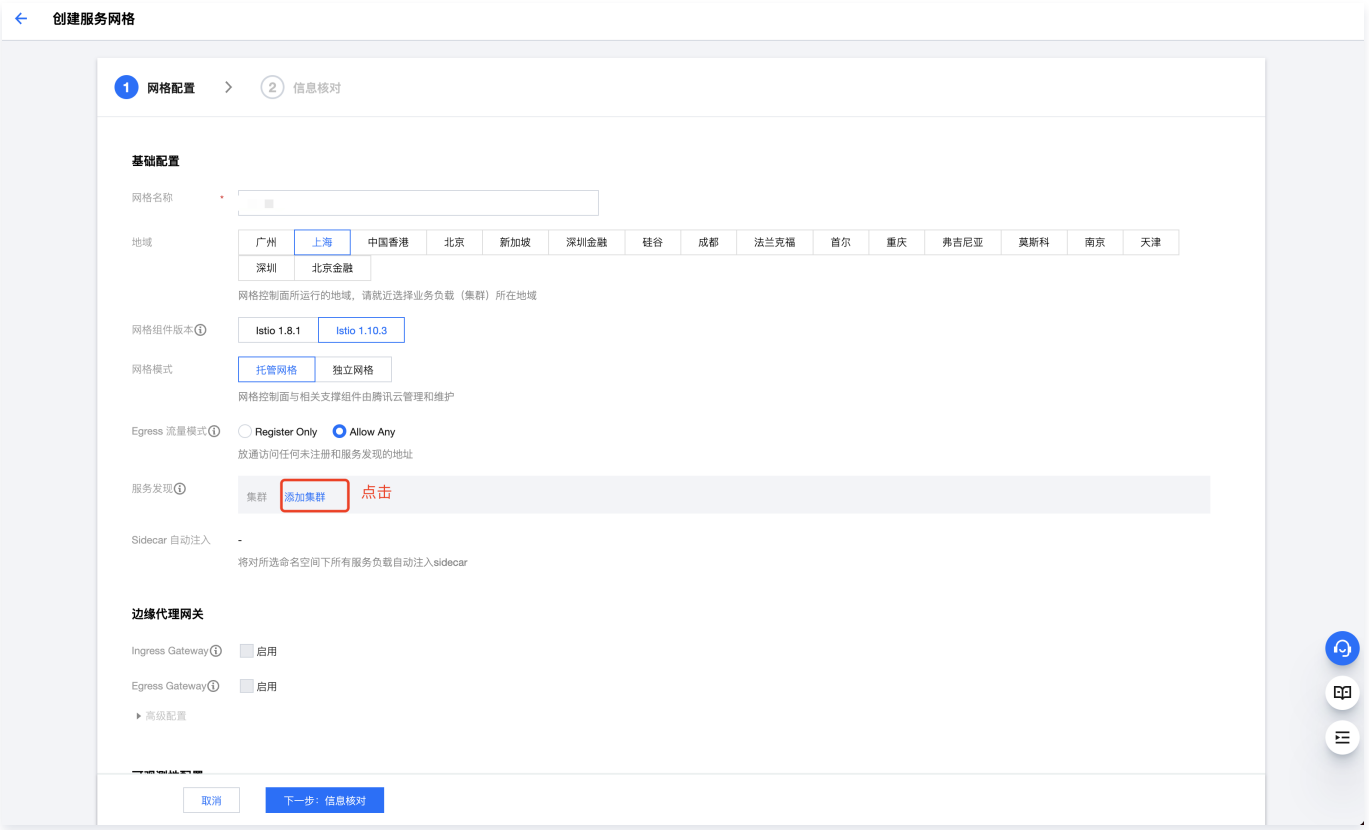
- **集群版本**：TCM 不强制要求集群版本完全相同，但集群版本应符合对应的 Istio 对 K8S 版本的要求，详情请参见 Istio K8S 支持情况 。
- **集群权限**：您需要对加入网格的集群有 admin 权限，详见 为集群添加网格权限 。
- **VPC 网络**：对于不在同一 VPC 的多个集群，为确保跨集群 Pod 正常访问，需使用 云联网 联通的多个集群，请将集群添加到同一个云联网实例中。**云联网实例中各端的 VPC 中主机 CIDR 、容器 CIDR 不冲突。**
- **容器网络**：如果 TKE 集群使用 Global Route 网络模式，需要 将容器网络注册至云联网 ，以便于新添加的容器 CIDR 可被访问。

操作步骤

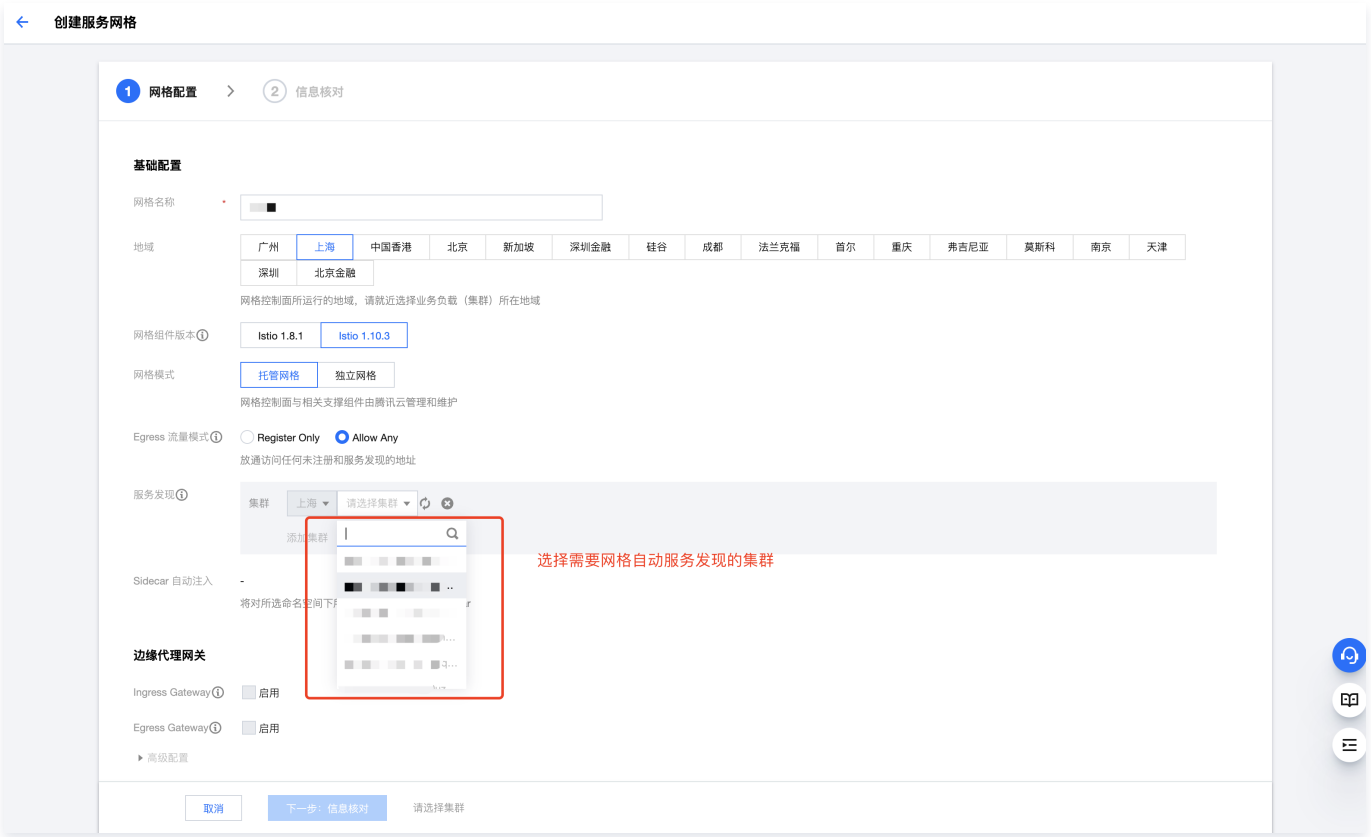
创建网格页面

您可以在创建网格时配置添加自动服务发现的集群，在网格创建配置页面：

1. 登录 服务网格控制台 。
2. 单击**新建**，创建服务网格。
3. 在**基础配置** > **服务发现**参数，单击**添加集群**。如下图所示：



4. 选择需要添加的自动服务发现 Kubernetes 集群，可同时添加多个集群。提交网格创建请求后，创建的网格实例即可自动发现集群内的 K8S 服务。如下图所示：

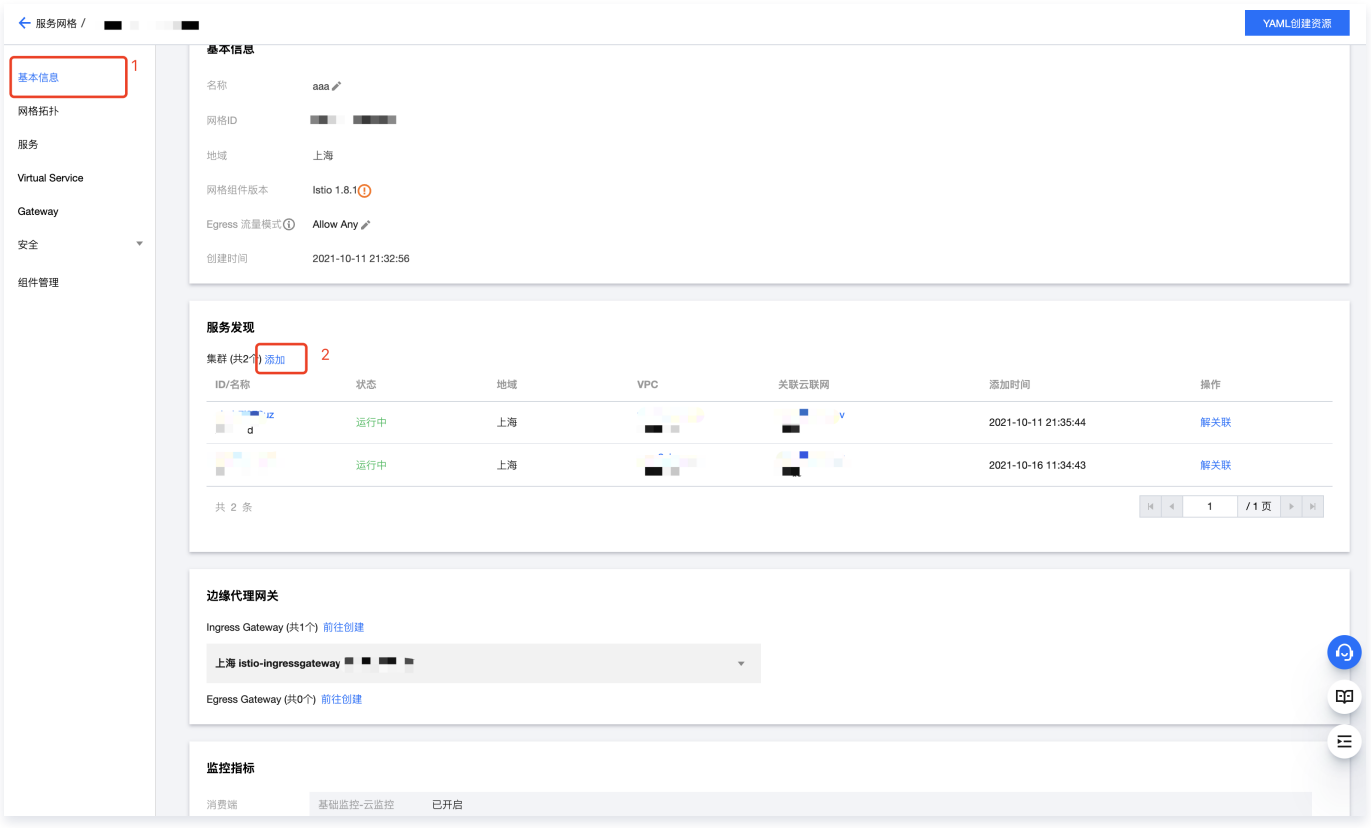


网格详情页面

您可以在网格详情页面，查看当前网格实例的服务发现集群，并添加，或解关联自动服务发现集群。

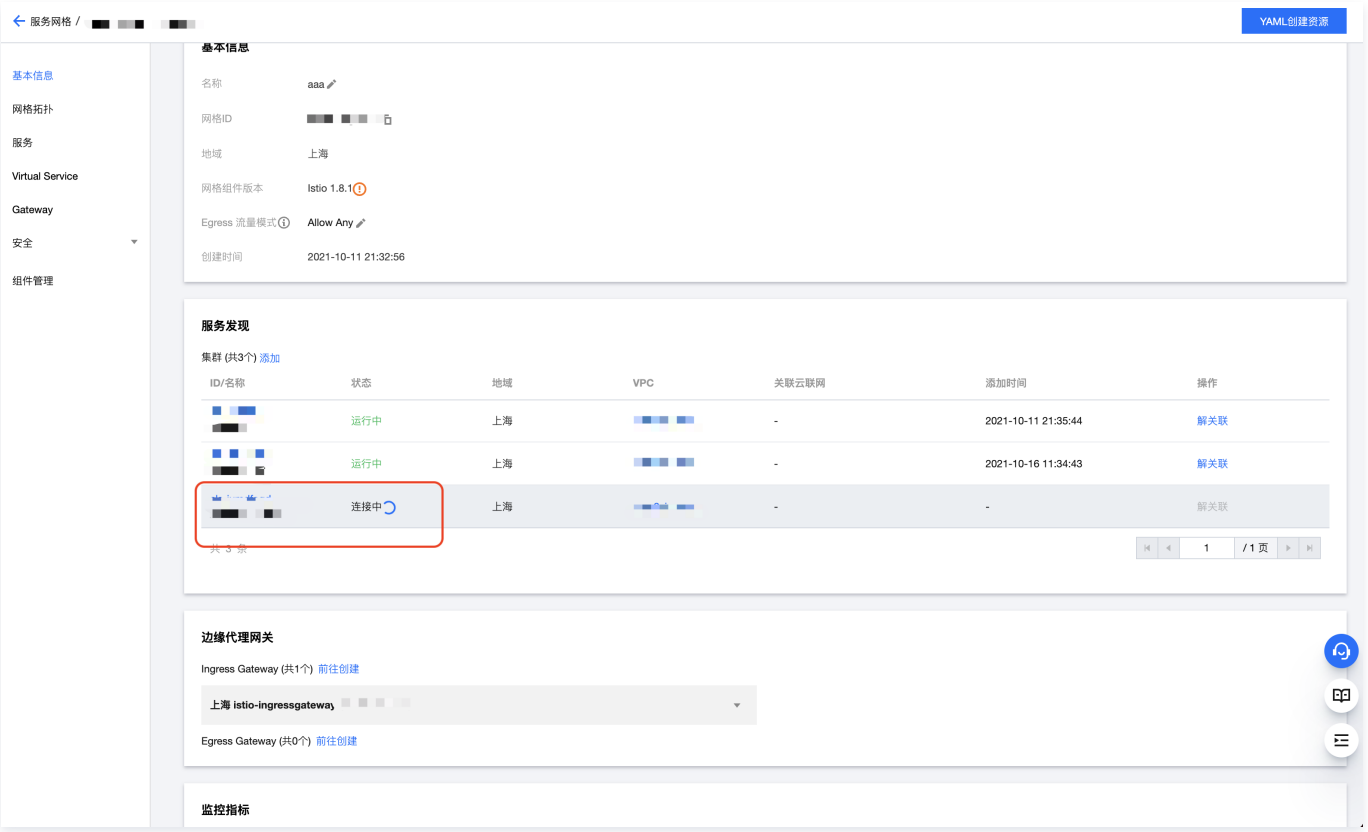
添加服务发现集群

1. 进入网格详情页面，单击侧边栏**基本信息**，在**服务发现**模块，可查看当前网格服务发现的集群列表，单击**添加**进入**添加服务发现集群**弹窗。如下图所示：



2. 在**添加服务发现集群**弹窗中，选择需要添加的自动服务发现 Kubernetes 集群，可同时添加多个集群，单击**确定**。如下图所示：

3. 提交添加服务发现集群后，等待集群连接完成后，即可完成服务发现集群添加。如下图所示：

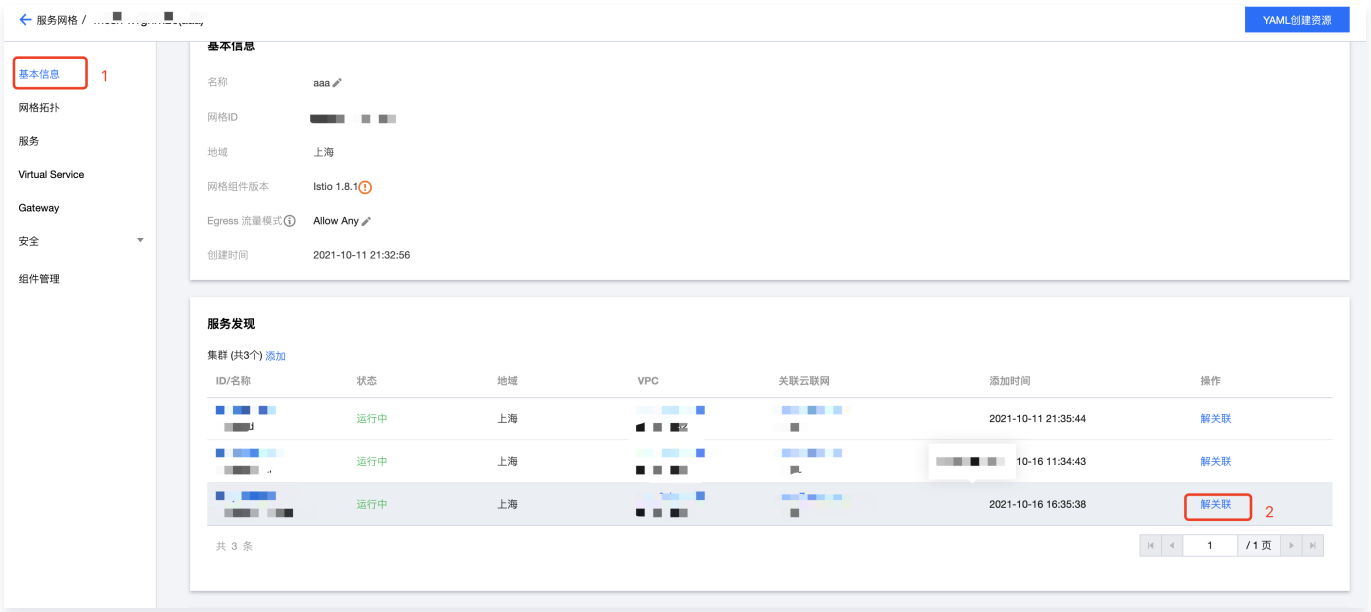


> **注意：**
> 服务添加到网格内后，如果需要进程流量管理、可视化观测等管理操作，还需要为网格配置 Sidecar 注入，相关指引请参考 网格配置 。

解关联服务发现集群

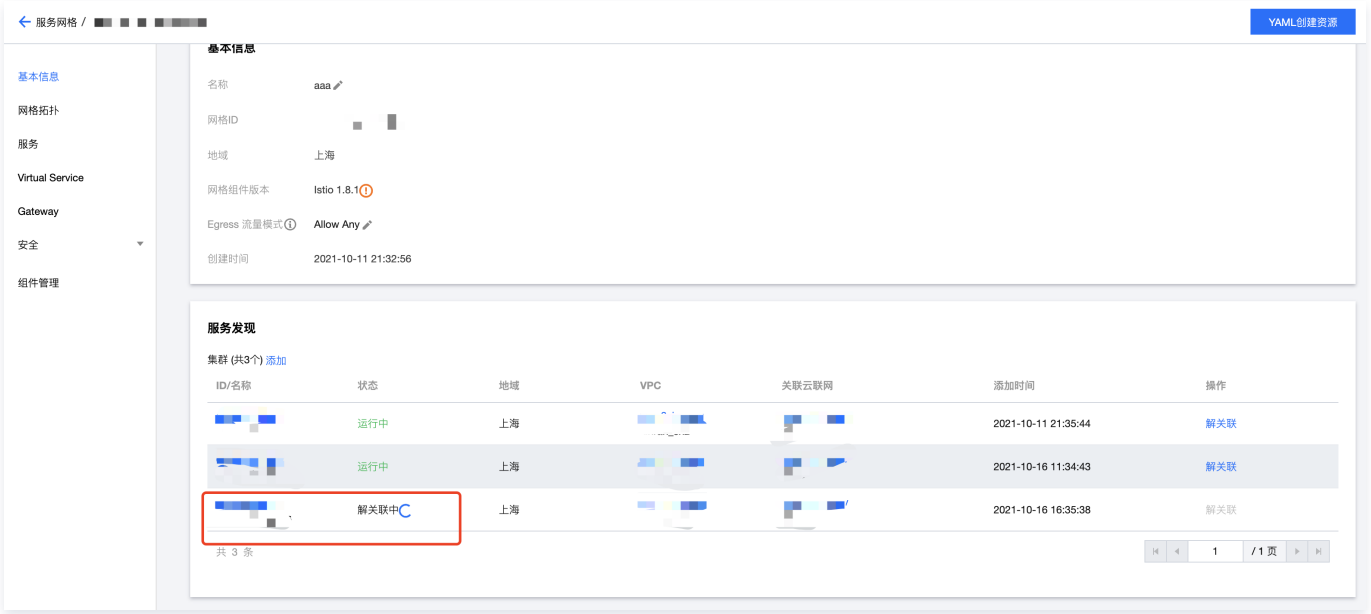
如您需要解关联网格已添加的服务发现集群，您可以按一下步骤操作：

1. 进入网格详情页面，单击侧边栏**基本信息**，在**服务发现**模块，可查看当前网格服务发现的集群列表，选择需要解关联的集群，单击操作栏的**解关联**进入确认解关联弹窗。如下图所示：



2. 在**解除集群关联**弹窗中，确认需要解关联的服务发现集群信息，单击**确定**提交解关联集群请求。注意解除集群关联后，网格不再感知该集群下服务实例变更，相关服务请求可能出现异常。如下图所示：

3. 等待解关联操作完成即可。如下图所示：



手动服务注册

最近更新时间：2024-03-22 14:07:41

简介

借助 Istio 的 Service Entry、Workload Entry 机制，可以在 TCM 中添加非 TKE 提供的集群中的服务，例如传统的 VM 服务、数据库服务等。Service Entry 对应 K8S 中的 Service 概念，某个服务通过 Service Entry 加入网格后，将可被网格内其他自动发现的服务按照路由规则访问。Workload Entry 对应 K8S 中 Workload 概念，用于标记一组与 Service Entry 对应的服务实体程序，安装 Sidecar 后的 workload Entry 可与其他自动发现的 K8S workload 一样，实现流量控制、安全增强、服务观测等能力。

Service Entry 重要字段说明

字段名称	字段格式	字段说明
spec.hosts	string	服务的 URL 中的 hostname，可以有多个。
spec.ports	Port[]	服务端口号，可以有多个。
spec.resolution	string	• Static：使用静态的 endpoint ip 地址作为服务实例。 • DNS：通过 DNS 解析服务 endpoint ip 地址，多用于外部服务；申明的 endpoint 需使用 DNS 域名，在无 endpoint 情况下将解析服务为 hosts 域名。 • NONE：当服务无需 IP 解析时选择。
spec.location	string	用于标记此服务是否在网格内，部分 Istio 能力特性不能在网格外服务使用，例如网格外的服务不支持mTLS。MESH_EXTERNAL 代表网格外的服务，MESH_INTERNAL 代表网格内服务。
spec.endpoints	String	服务的接入点，可填写多个，但最终只会同时使用一个。

Workload Entry 重要字段说明

字段名称	字段格式	字段说明
spec.address	string	当前 endpoint 的地址，类似于 pod IP。
spec.labels	string	当前 endpoint 的标签，用于与 Service Entry 关联。
sepc.serviceAccount	string	sidecar 的权限信息，当需要为 endpoint 添加 sidecar 的时候需要指定。

关于 Service Entry、Workload Entry 的详细介绍，请参见 Service Entry 详细介绍，Workload Entry详细介绍 。

手动注册服务

YAML 配置示例

Service Entry

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: external-svc-https
spec:
  hosts:
  - api.dropboxapi.com
  - www.googleapis.com
  - api.facebook.com
  location: MESH_EXTERNAL
  ports:
  - number: 443
    name: https
    protocol: TLS
```

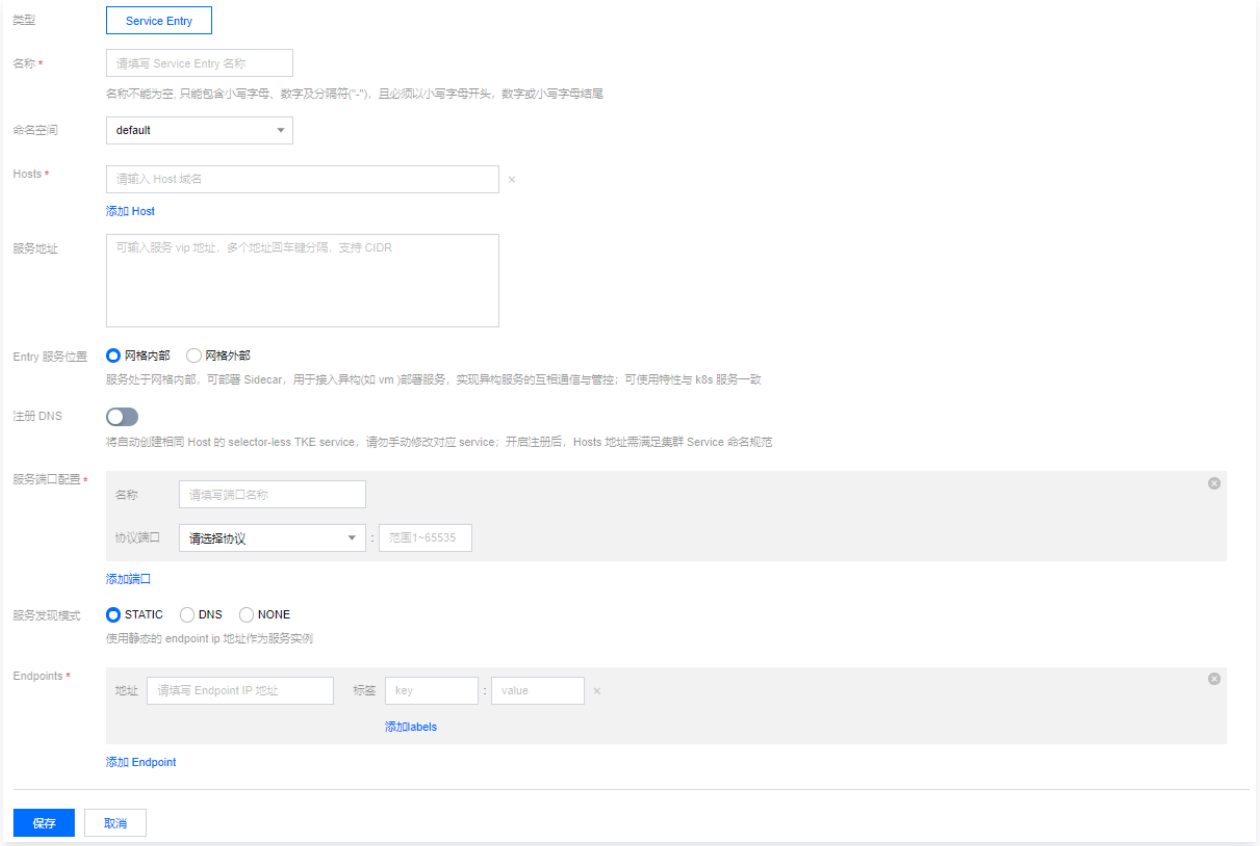
```
  resolution: DNS
```

Workload Entry

```
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: details-svc
spec:
  serviceAccount: details-legacy
  address: 2.2.2.2
  labels:
    app: details-legacy
    instance-id: vm1
```

控制台配置示例

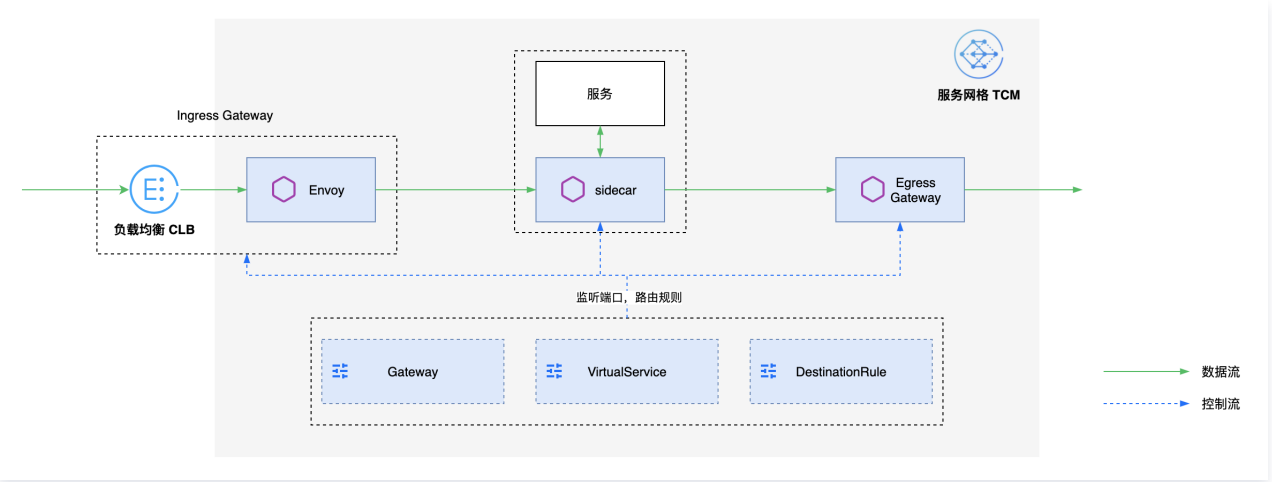
1. 登录 服务网格控制台 。
2. 单击 ID/名称，进入网格详情页面。
3. 单击**服务 > 新建**，填写服务基本信息，即可将非容器化的第三方服务注册到网格服务中，如下图所示：



边缘代理网关
边缘代理网关管理

最近更新时间：2024-03-22 14:07:41

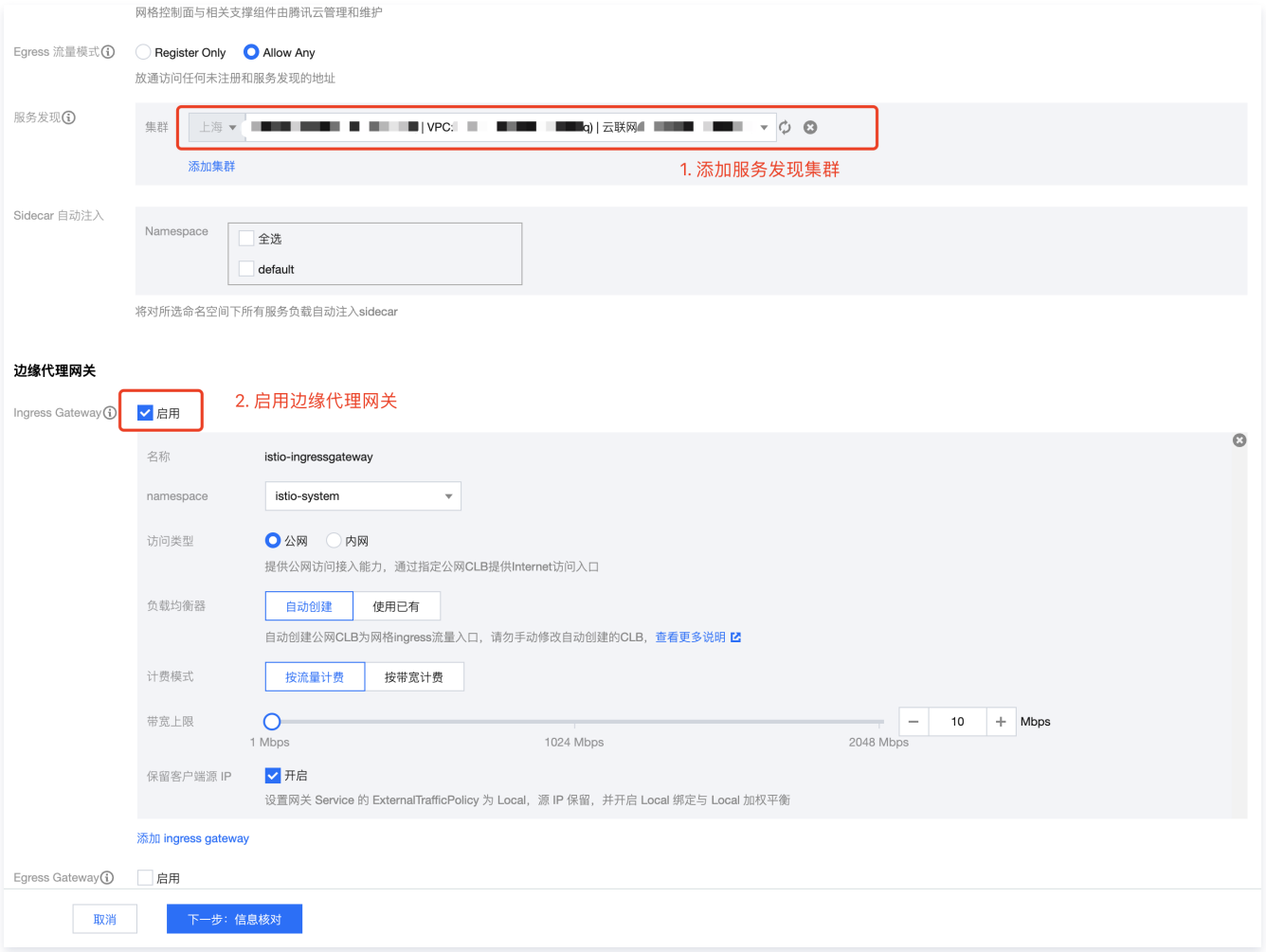
边缘代理网关是负责网格出口与入口流量负载均衡的特殊数据面，它不以 Sidecar 的形式，而是以独立 Pod 的形式部署在您的集群内，分为 Ingress Gateway 和 Egress Gateway 两种类型。其中，一个 Ingress Gateway 实例包含数据面的 Envoy Pod 和它关联的负载均衡 CLB 实例（公网或内网），TCM 提供托管的 边缘代理网关控制器，已经实现了 Ingress Gateway 配置与 CLB 的自动化集成，您可以通过 Istio CRD 配置 Ingress Gateway，相关设置 TCM 会自动同步至关联的 CLB 实例，同步的配置包括端口配置和增强功能的端口监听规则配置两部分。即 Envoy 容器和关联的 CLB 作为一个整体，为您提供入口边缘代理网关的能力。
如您需要网格出入口流量负载均衡的能力，您需先要创建 Ingress Gateway 或 Egress Gateway 实例，再通过 Gateway，VirtualService，DestinationRule 等 Istio CRD 配置边缘代理网关的监听规则和流量管理（路由）规则。监听规则通过 Gateway CRD 配置，流量管理规则通过 VirtualService、Destination Rule 配置（与东西向流量管理语法一致）。下图是边缘代理网关实例与 Istio CRD 配置的关系示意图。



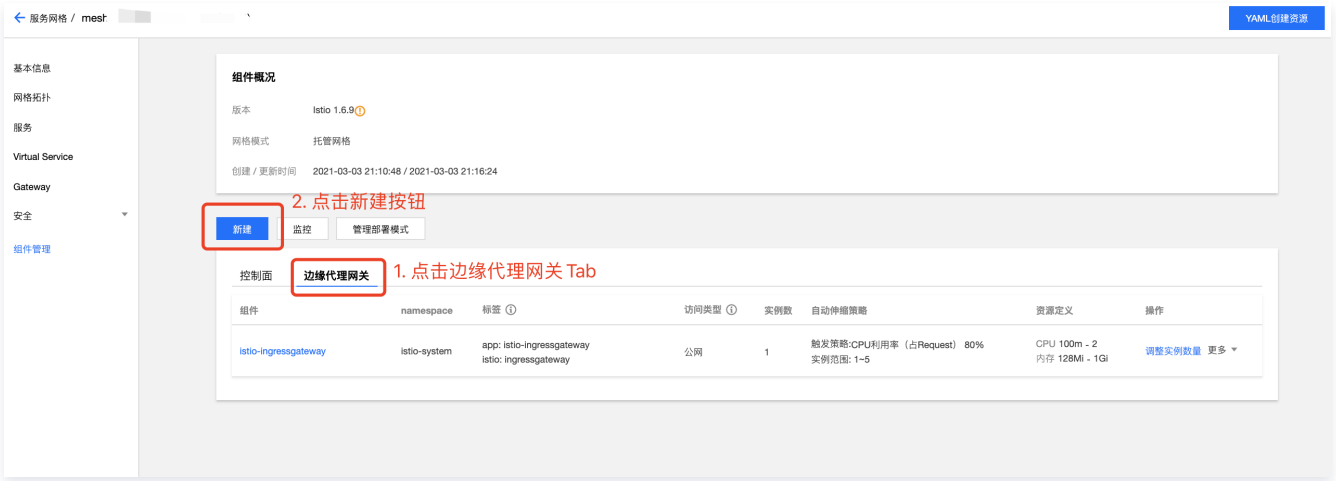
创建边缘代理网关

1. 登录 服务网格控制台 。

2. 您可以在网格创建页面，添加服务发现集群后，创建边缘代理网关。如下图所示：



或在网格详情页面边缘代理网关页签单击**新建**创建边缘代理网关。如下图所示：



创建边缘代理网关重要配置项说明：

配置项	描述
类型	选择创建 Ingress 网关或 Egress 网关。
接入集群	选择网关创建的 Kubernetes 集群。
namespace	选择网关创建的命名空间。
访问类型	Ingress Gateway 参数。选择 CLB 的访问类型，支持公网和内网。

负载均衡器	Ingress Gateway 参数。选择自动创建负载均衡器，或选择复用现有负载均衡器，复用已有负载均衡器更多介绍参见 Service 使用已有 CLB 。
计费模式	Ingress Gateway 参数。选择 CLB 的计费模式，支持按流量计费或按带宽计费，更多关于 CLB 计费的信息，参见 CLB 计费概述 。
带宽上限	Ingress Gateway 参数。选择 CLB 的带宽上限，0 – 2048 Mbps。
CLB 直连 Pod	Ingress Gateway 参数。例如，网关接入集群网络模式为 VPC-CNI，即可开启 CLB 直连 Pod，将不再进行 NodePort 转发，具有更高性能，支持保留客户端源 IP、Pod 层级会话保持与健康检查，更多详情请参见 使用 LoadBalancer 直连 Pod 模式 Service 。
保留客户端源 IP	Ingress Gateway 参数。设置网关 Service 的 ExternalTrafficPolicy 为 Local，源 IP 保留，并开启 Local 绑定与 Local 加权平衡。开启 CLB 直连 Pod 时，该参数失效。更多详情请参见 Service 后端选择 。
组件配置	配置网关的 CPU 和 内存资源，以及 HPA 伸缩策略。

边缘代理部署模式

边缘代理网关有**普通模式**，**专有模式**两种部署形态，其中：

- **普通模式**：网关服务部署在所选的业务集群中，与其他业务 Pod无差别部署。
- **专有模式**：某些场景下为了提升服务的稳定性，您需要将边缘代理网关部署在专有节点上。专有模式需要先选择将部分集群节点添加到专有资源池中，网格将会为选中的节点设置污点以保证独享。设置后所有 Ingress/Egress Gateway 都将仅部署在选中的节点中。您还可以在高级设置中为特定的 Gateway 进一步指定节点。

您可以在网格创建页、或组件管理页重新调整网关部署模式：

> **说明：**
> 调整部署模式会触发网关服务调度，可能会对业务流量产生影响。

更新边缘代理网关配置

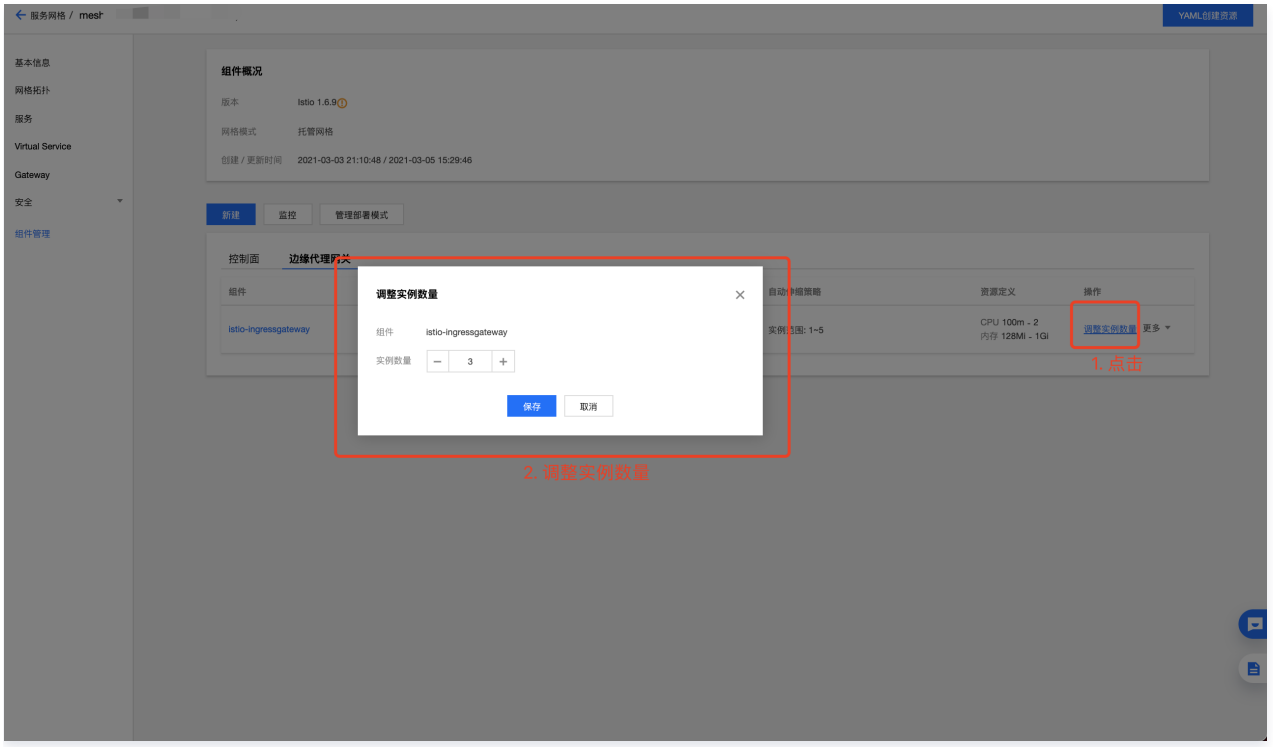
边缘代理网关创建后，您可以修改边缘代理网关的关联 CLB 带宽（仅 Ingress Gateway）、实例数量、HPA 伸缩策略及资源定义配置。

修改 CLB 带宽

您可以修改 Ingress Gateway 关联的 CLB 实例的带宽。在对应网格详情页面的**基本信息** Tab 边缘代理网关部分可以编辑 Ingress Gateway 绑定的 CLB 配置。

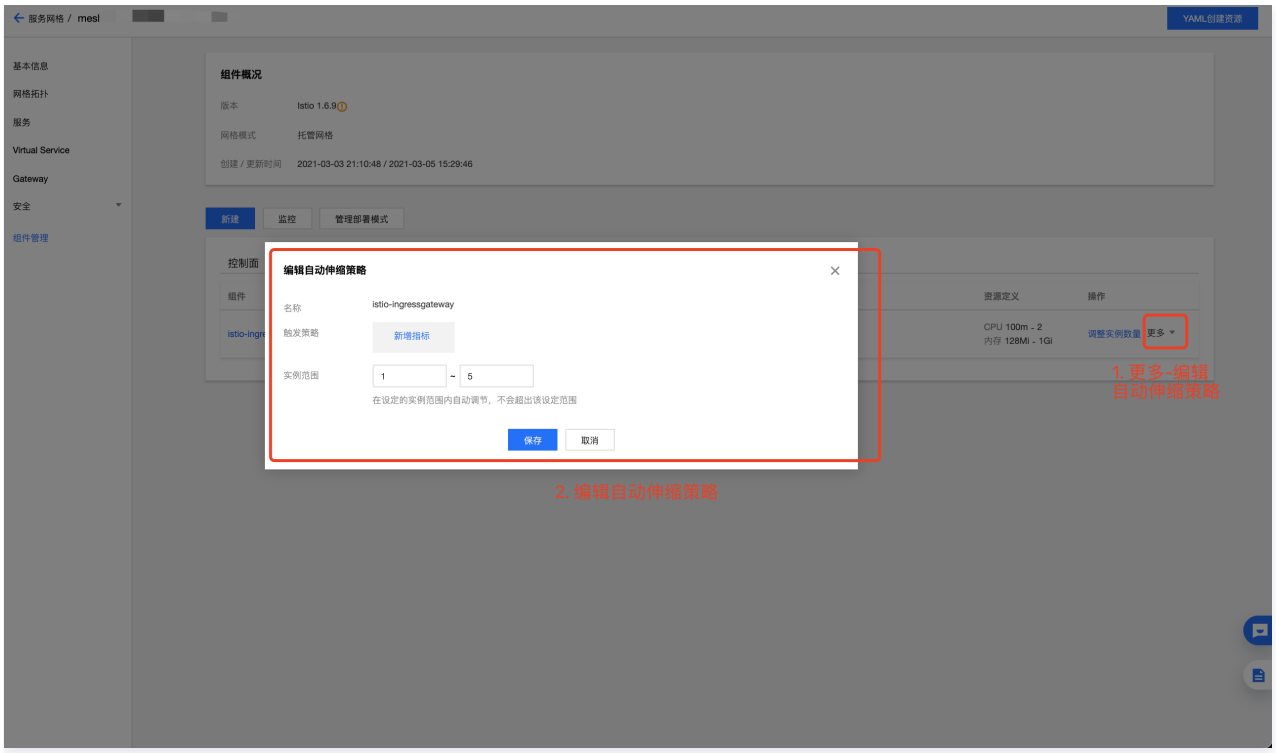
修改组件实例数量

在**网格详情 > 组件管理**可以调整组件的实例数量。



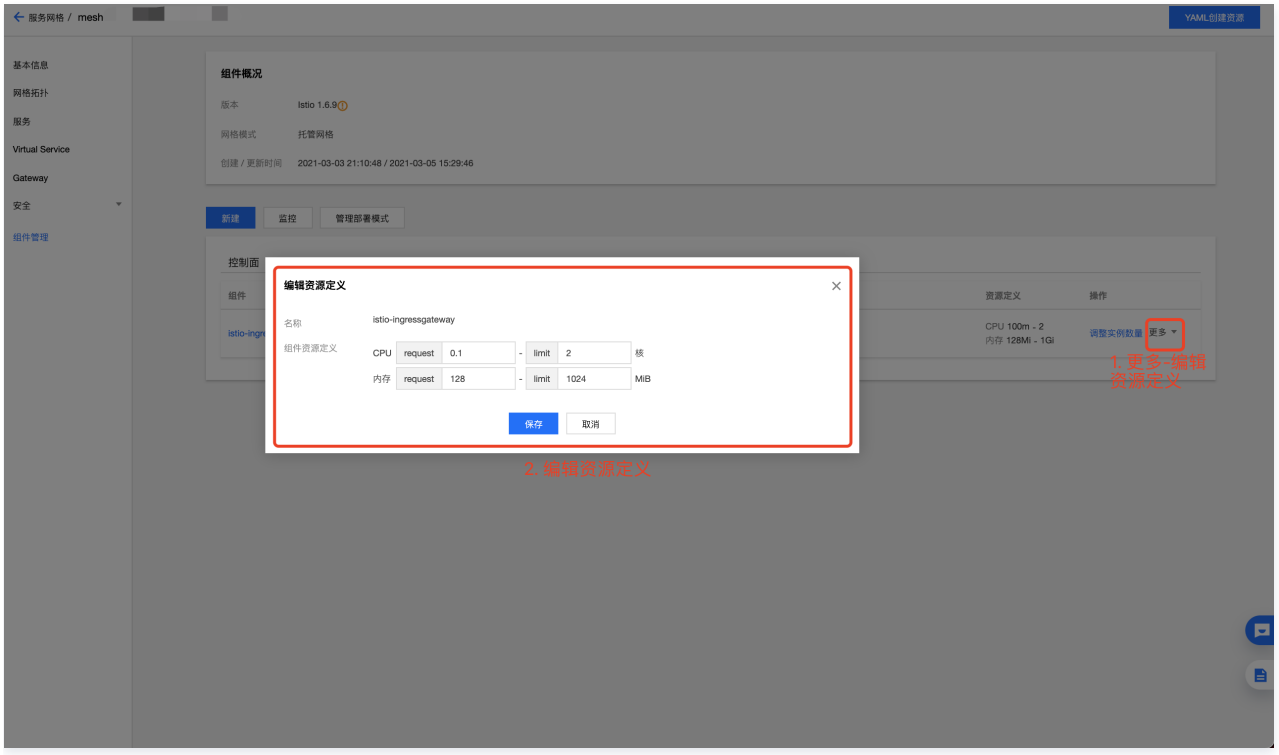
修改组件 HPA 伸缩策略

在**网格详情 > 组件管理**可以编辑组件的 HPA 策略。支持按照 CPU、内存、网络、硬盘指标配置伸缩策略。如下图所示：



修改组件资源定义

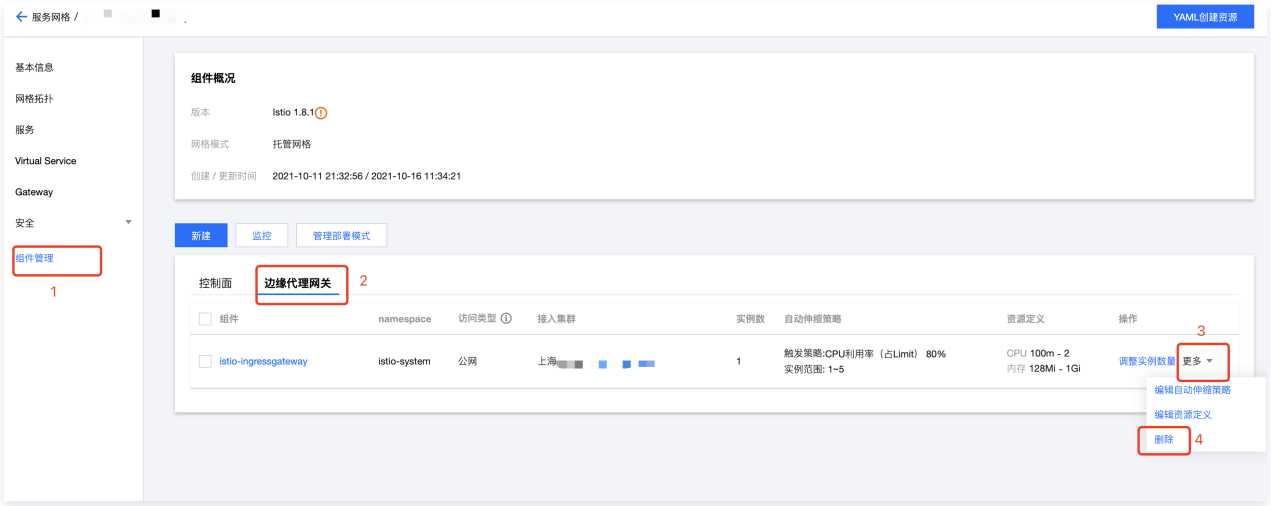
在**网格详情 > 组件管理**可以编辑组件的资源定义，包括 CPU、内存的 request 和 limit 值。如下图所示：



删除边缘代理网关

您可以在网格详情–组件管理–边缘代理网关删除指定边缘代理网关。步骤如下：

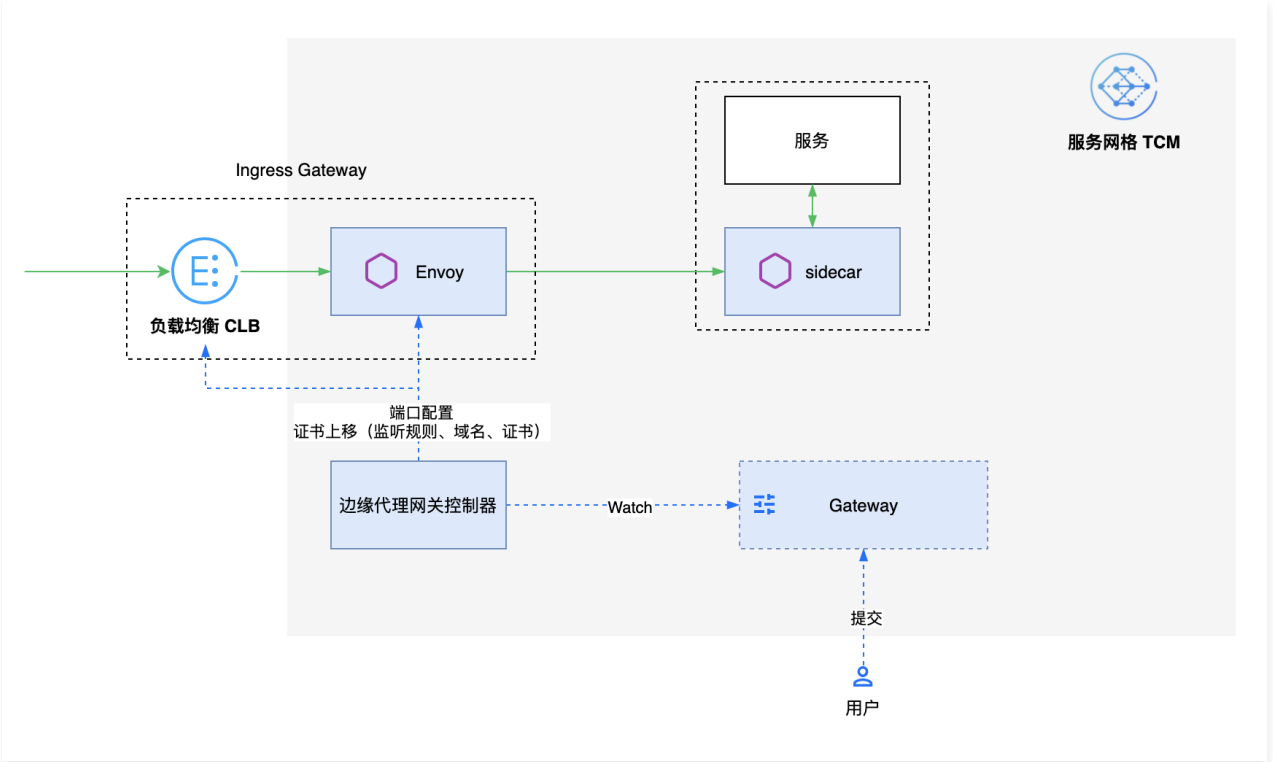
1. 进入网格详情页，单击**组件管理**，选择**边缘代理网关**，在需要删除的网关对应行**操作**列单击**更多 > 删除**。如下图所示：



2. 在**删除边缘代理网关**弹窗，确认删除的网关名称，单击**确定**完成删除。如下图所示：

TCM 边缘代理网关控制器实现自动对接 CLB

TCM 实现了托管的边缘代理网关控制器，该控制器会实时监测下发到 Ingress Gateway 的 Gateway 配置，解析当前的端口配置，同步当前端口配置到 CLB，您无需再手动配置 CLB 端口。CLB 端口与 Ingress Gateway Service 端口、Ingress Gateway 容器端口的映射关系是一一映射，即如果在 Gateway CRD 中定义 `80` 端口，TCM 边缘代理网关控制器会配置 Ingress Gateway 实例的容器端口为 `80`，服务端口为 `80`，以及同步开启关联 CLB 的 `80` 端口。
TCM 边缘代理网关控制器还实现了 SSL 证书解包上移至 CLB 的功能，实现在 CLB 做证书解包后，Ingress Gateway 再提供流量管理的能力。在 Gateway 中配置证书上移后，边缘代理网关控制器会解析配置上移的端口、域名、证书，并将配置同步至 Ingress Gateway 绑定的 CLB 实例。



Gateway 配置

最近更新时间：2024-03-22 14:07:41

边缘代理网关的端口、监听规则通过 Gateway CRD 配置。以下是一个 Gateway 配置的示例，重要字段的解释通过注释说明：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: gateway-sample
  namespace: defualt
spec:
  selector: # 根据填写的标签匹配 Gateway 配置下发的 Pod
    istio: ingressgateway
    app: istio-ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - uk.bookinfo.com
    - eu.bookinfo.com
    tls:
      httpsRedirect: true # 发送 301 https 重定向
  - port:
      number: 443
      name: https-443
      protocol: HTTPS # 开启端口 HTTPS
    hosts:
    - uk.bookinfo.com
    - eu.bookinfo.com
    tls:
      mode: SIMPLE # TLS 单向认证
      serverCertificate: /etc/certs/servercert.pem # 文件挂载方式加载证书
      privateKey: /etc/certs/privatekey.pem
  - port:
      number: 9443
      name: https-9443
      protocol: HTTPS # 开启端口 HTTPS
    hosts:
    - "bookinfo-namespace/*.bookinfo.com"
    tls:
      mode: SIMPLE # TLS 单向认证
      credentialName: bookinfo-secret # 通过 SDS 方式从 Kubernetes secret 加载证书
  - port:
      number: 5443
      name: https-ssl
      protocol: HTTPS # 开启端口 HTTPS
    hosts:
    - "*"
    tls:
      mode: SIMPLE # TLS 单向认证
      credentialName: qcloud-abcdABCD # 通过 SDS 方式从腾讯云 SSL 平台加载证书 ID 为 abcdABCD 的证书
  - port:
      number: 6443
      name: clb-https-6443-ABCDabcd # 6443启用证书解包上移至 CLB，使用 ID 为 ABCDabcd 的 SSL 平台证书
      protocol: HTTP
    hosts:
    - "tcm.tencent.com"
```

Gateway 配置字段说明

以下是 Gateway CRD 重要字段的说明：

字段名称	字段类型	字段说明
`metadata.name`	`string`	Gateway 名称。
`metadata.namespace`	`string`	Gateway 命名空间。
`spec.selector`	`map<string, string>`	Gateway 使用填写的标签键值对匹配配置下发的边缘代理网关实例。
`spec.servers.port.number`	`uint32`	端口。
`spec.servers.port.protocol`	`string`	通信协议，支持：`HTTP, HTTPS, GRPC, HTTP2, MONGO, TCP, TLS`，请注意同一网关同一端口的协议配置需要保持一致。
`spec.servers.port.name`	`string`	端口名称，当前 TCM 实现了通过端口名称指定 SSL 证书解包上移至 CLB 的功能，如您需要配置证书上移，您可以按照 `clb-https-{端口号}-{ssl 平台证书 ID}` 的方式命名，证书上移功能仅在当前端口通信协议指定为 HTTP 时生效，边缘代理网关控制器会自动创建 CLB 7层监听器实现证书上移，CLB SSL 解包完成后，CLB 实例与 Ingress Gateway Pod 采用明文通信。请注意同一网关同一端口的证书上移配置需要保持一致，否则会引起配置冲突。
`spec.severs.hosts`	`string[]`	域名，支持通配符 `*` 。
`spec.servers.tls.httpsRedirect`	`bool`	值为 `true` 时，边缘代理网关会对所有 http 请求返回 301 重定向，要求客户端发起 https 请求。
`spec.servers.tls.mode`	-	配置当前端口的 TLS 安全认证模式，如需要开启当前端口的安全认证则需要填写。支持：`PASSTHROUGH, SIMPLE, MUTUAL, AUTO_PASSTHROUGH, ISTIO_MUTUAL`
`spec.servers.tls.credentialName`	`string`	配置发现 TLS 证书密钥的 secret 的名称，支持从 Ingress Gateway 实例在同一 namespace 下的 Kubernetes secret 中加载证书与密钥，您需要确保填写的 secret 中包含合适的证书与密钥。TCM 还实现了加载腾讯云 SSL 平台证书的功能，按照 `qcloud-{ssl 平台证书 ID}` 格式填写本字段，TCM 边缘代理网关控制器即会为边缘代理网关加载 SSL 平台的证书。当前仅支持从 SSL 平台加载单向认证 SIMPLE 模式的服务器证书和私钥。
`spec.servers.tls.serverCertificate`	`string`	设置端口的 TLS 证书密钥通过 file mount 形式（不推荐，推荐采用填写 `credentialName` 字段加载证书私钥）挂载时需要填写的证书路径字段，Istio 默认使用网关所在命名空间下 istio-ingressgateway-certs secret 加载证书至路径 `/etc/istio/ingressgateway-certs` 。
`spec.servers.tls.privateKey`	`string`	设置端口的 TLS 证书密钥通过 file mount 形式（不推荐，推荐采用填写 `credentialName` 字段加载证书私钥）挂载时需要填写的私钥路径字段，Istio 默认使用网关所在命名空间下 istio-ingressgateway-certs secret 加载私钥至路径 `/etc/istio/ingressgateway-certs` 。
`spec.servers.tls.caCertificates`	`string`	设置端口的 TLS 证书密钥通过 file mount 形式（不推荐，推荐采用填写 `credentialName` 字段加载证书私钥）挂载时需要填写的跟证书路径字段，Istio 默认使用网关所在命名空间下 istio-ingressgateway-ca-certs 加载根证书至路径 `/etc/istio/ingressgateway-ca-certs` ，双向认证时需要配置根证书。

示例

从 Kubernetes Secret 加载证书至边缘代理网关配置示例

YAML 配置示例

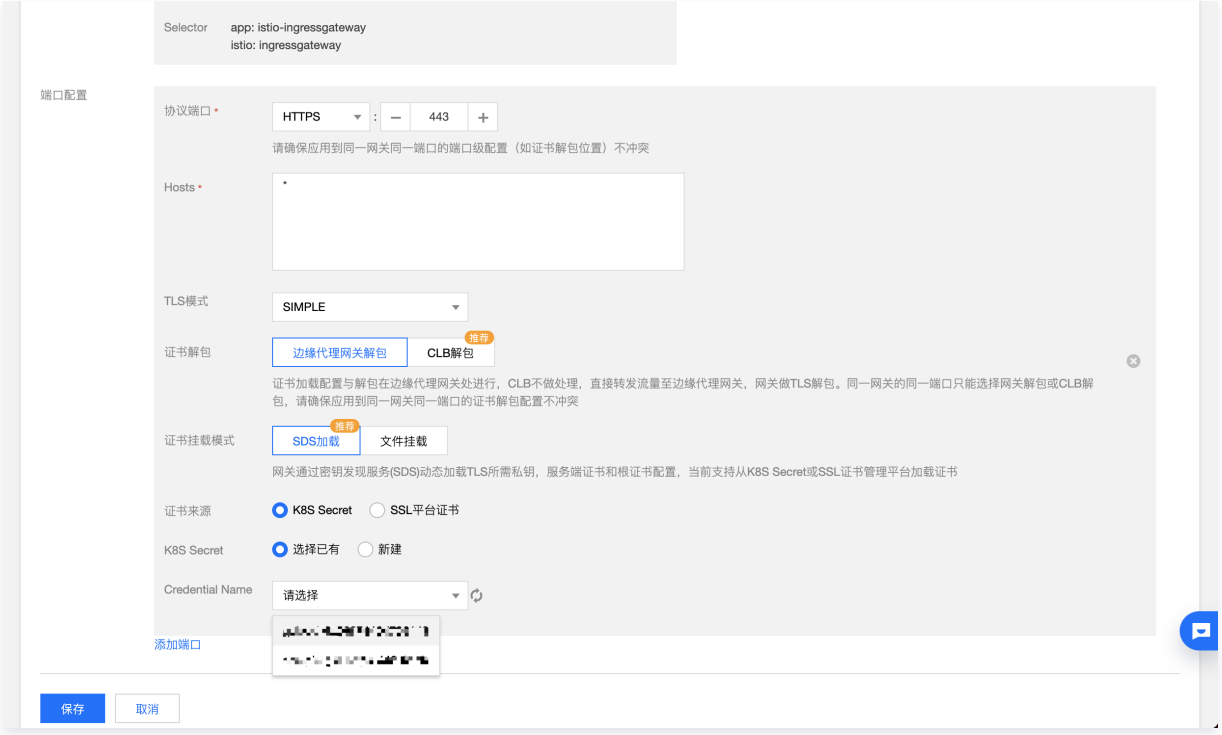
```
apiVersion: networking.istio.io/v1alpha3
```

```
kind: Gateway
metadata:
  name: sample-gw
  namespace: default
spec:
  servers:
    - port:
        number: 443
        name: HTTPS-443-6cph
        protocol: HTTPS
      hosts:
        - '*'
      tls:
        mode: SIMPLE
        credentialName: {kubernetes secret 名称}
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
```

控制台配置示例

控制台创建 Gateway 配置 Ingress Gateway HTTPS 协议 SSL 证书从 Kubernetes secret 加载（单向认证）的过程如下：

1. 选择协议为 **HTTPS**，TLS 模式为 **SIMPLE**。
2. 证书解包选择**边缘代理网关解包**。
3. 证书挂载模式选择 **SDS 加载**。
4. 证书来源选择 **K8S Secret**。
5. K8S Secret 选择**选择已有**，选择当前所选边缘代理网关所在 namespace 下的 Secret，请您确保所选 Secret 中包含合适的证书/私钥/根证书。



6. 如当前 Secret 中未有合适证书，您可以选择**新建** K8S Secret，复制合适的证书/私钥/跟证书内容至对应输入框。

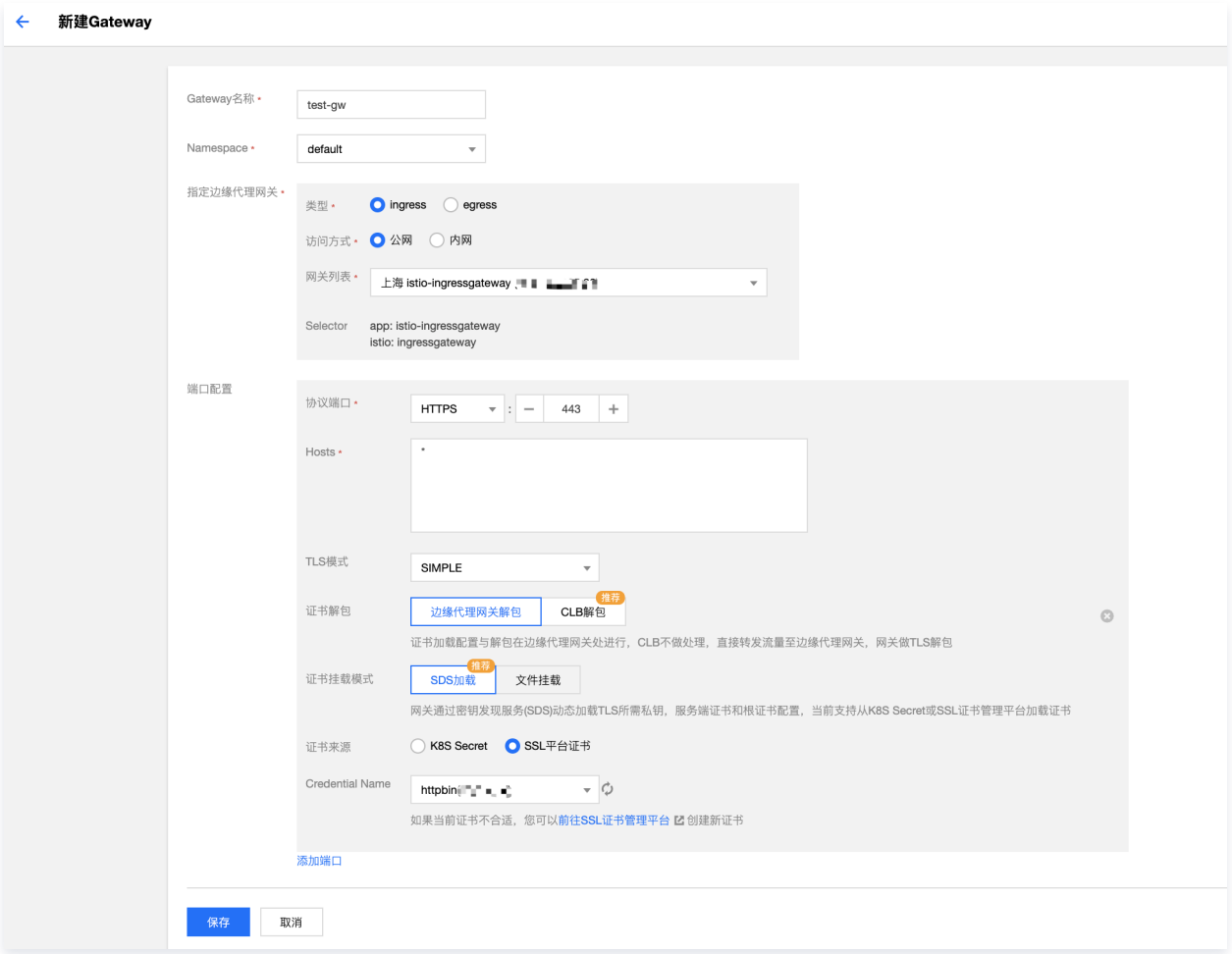
从 SSL 平台加载证书至边缘代理网关配置示例

YAML 配置示例

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: test-gw
spec:
  servers:
    - port:
        number: 443
        name: HTTPS-443-9ufr
        protocol: HTTPS
      hosts:
        - '*'
      tls:
        mode: SIMPLE
        credentialName: qcloud-{证书ID}
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
```

控制台配置示例

除了通过 YAML 文件配置，您也可以在控制台上通过 UI 创建 Gateway 配置。以下是配置从 SSL 平台加载证书至边缘代理网关的配置示例，您选择证书来源为 **SSL 平台证书**即可选择需要加载的 SSL 平台证书。



SSL 证书解包上移至 CLB 配置示例

YAML 配置示例

以下是配置 443 端口证书解包上移至 CLB，且为该端口启用 SNI，域名 `sample.hosta.org` 使用证书1，域名 `sample.hostb.org` 使用证书2。

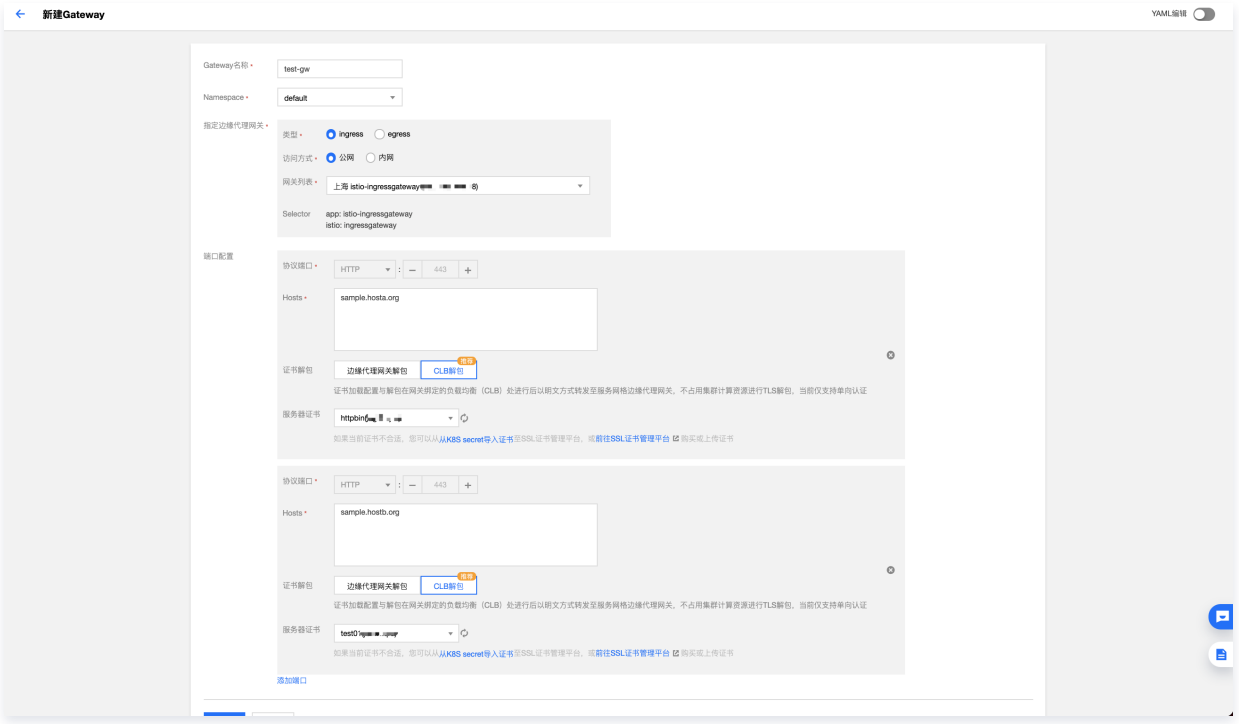
```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: test-gw
spec:
  servers:
    - port:
        number: 443
        name: clb-https-443-{证书ID 1}
        protocol: HTTP
      hosts:
        - sample.hosta.org
    - port:
        number: 443
        name: clb-https-443-{证书ID 2}
        protocol: HTTP
      hosts:
        - sample.hostb.org
  selector:
    app: istio-ingressgateway
```

```
    istio: ingressgateway
```

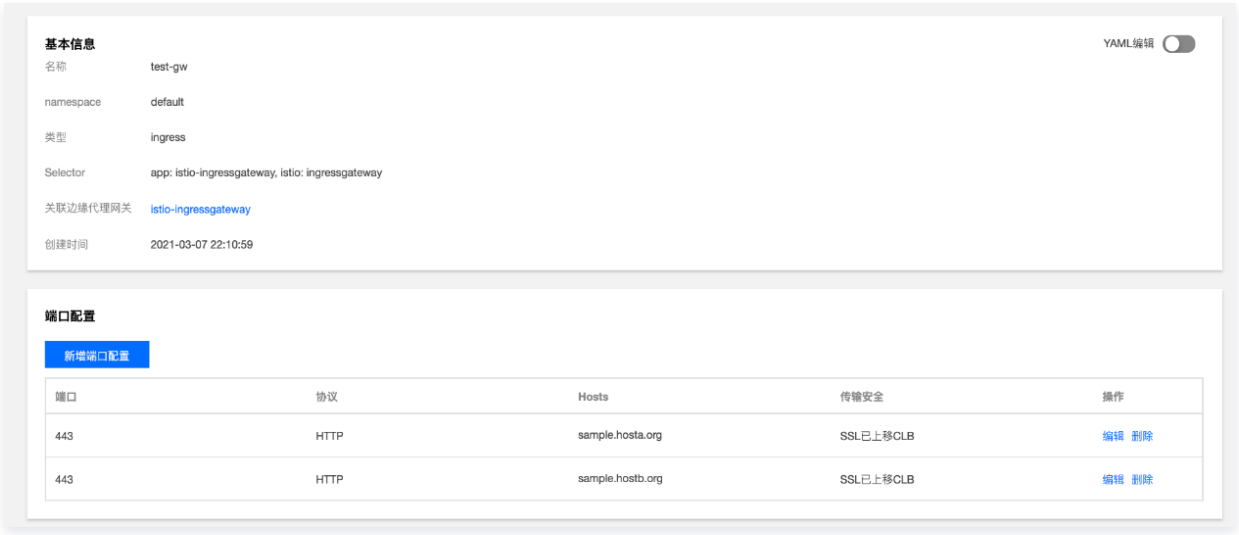
控制台配置示例

在控制台 UI 创建 Gateway 配置使用证书上移功能流程如下：

1. 选择协议为 HTTPS，出现 **TLS 模式选项**。
2. 选择 **TLS 模式为 SIMPLE**。
3. 选择**证书解包为 CLB 解包**，此时端口协议将自动变化为 HTTP（选择证书上移后网关处按照明文 HTTP 处理流量）。
4. 选择合适的**服务器证书**。



5. 创建成功将跳转至创建完成的 Gateway CRD 详情页面：

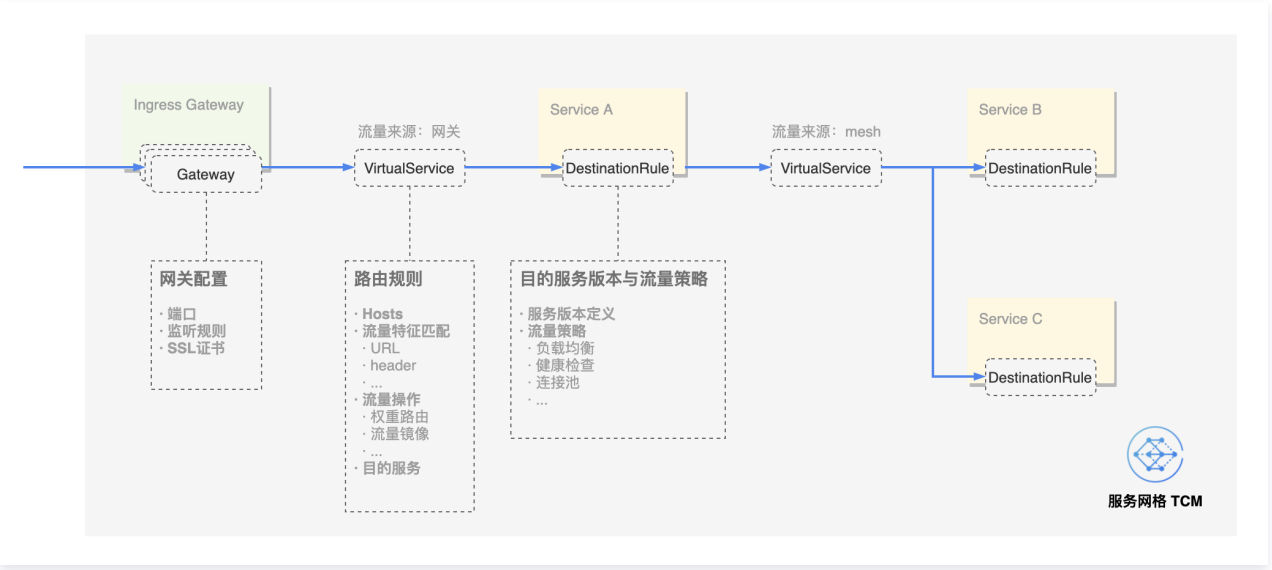


流量管理
概述

最近更新时间：2024-03-22 14:07:41

TCM 流量管理模型

TCM 完全兼容 Istio 管理流量的原生 CRD：Gateway，VirtualService 和 DestinationRule，并对原生流量管理语法做了产品化呈现，下图是 TCM 的流量管理模型：



TCM 使用 Gateway，VirtualService 和 DestinationRule 管理流量。

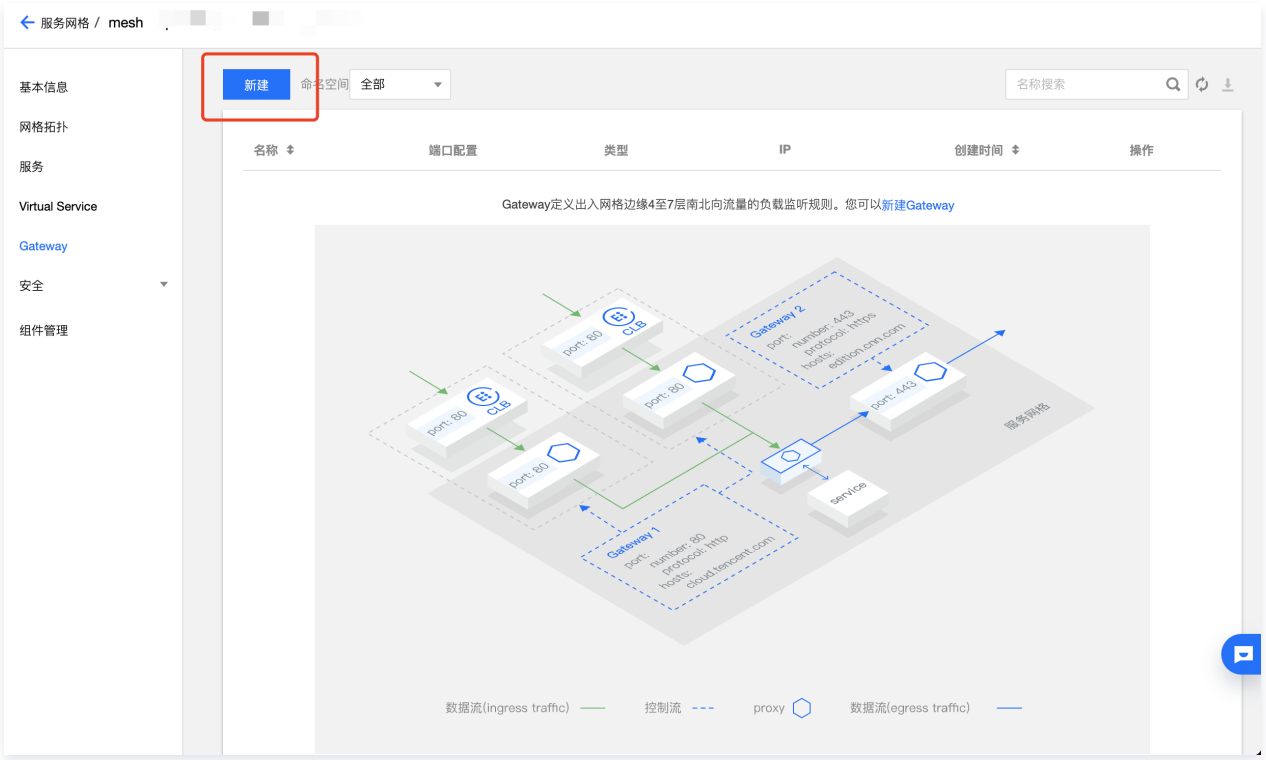
- Gateway：定义网关的端口、监听规则、证书配置，网关与 Gateway 配置是一对多的关系，Gateway 通过 selector 字段指定配置下发的边缘代理网关。
- VirtualService：定义指定 Host 的路由规则和流量操作规则，VirtualService 通过 hosts 字段指定绑定的域名。可规定流量来源是边缘代理网关或 mesh 内部。
- DestinationRule：定义服务的版本和流量策略，包括负载均衡、健康检查、连接池等流量策略。服务与 DestinationRule 是一对一的绑定关系。

流量管理配置方式

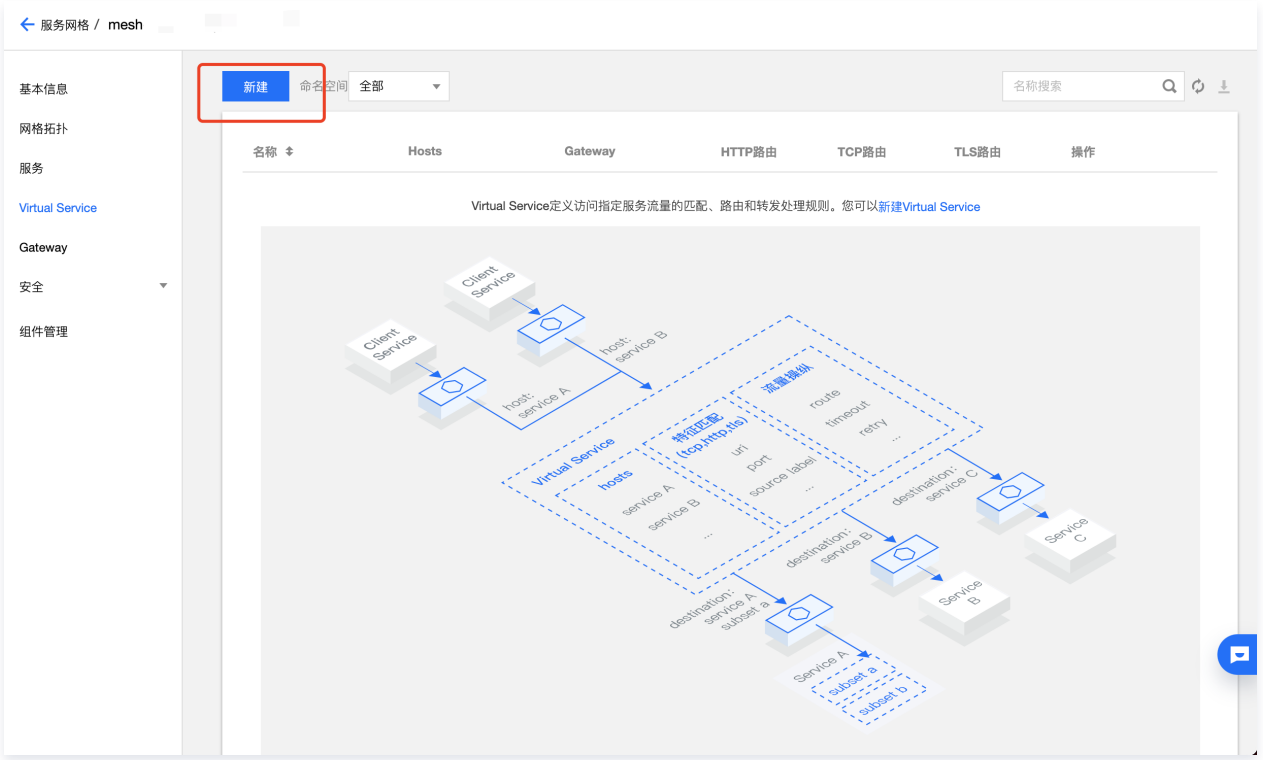
当前 TCM 提供以下两种方式配置 Gateway，VirtualService 和 DestinationRule：

控制台 UI 配置
您可以通过控制台 UI 创建、删除、更新、查看 Gateway，VirtualService，DestinationRule。

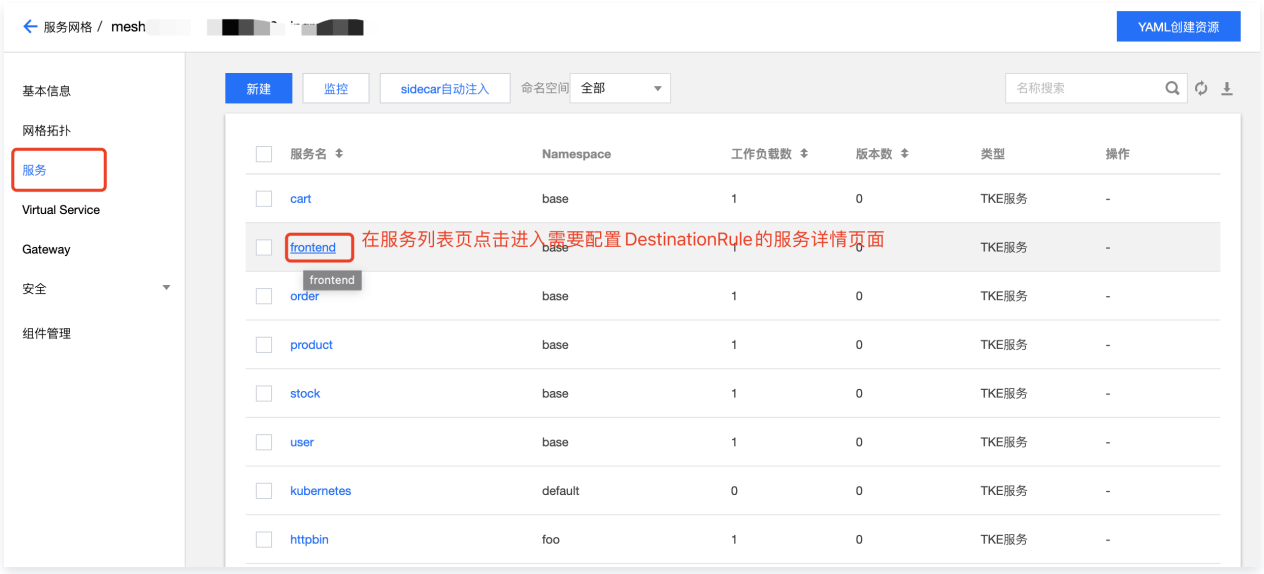
- 创建 Gateway：

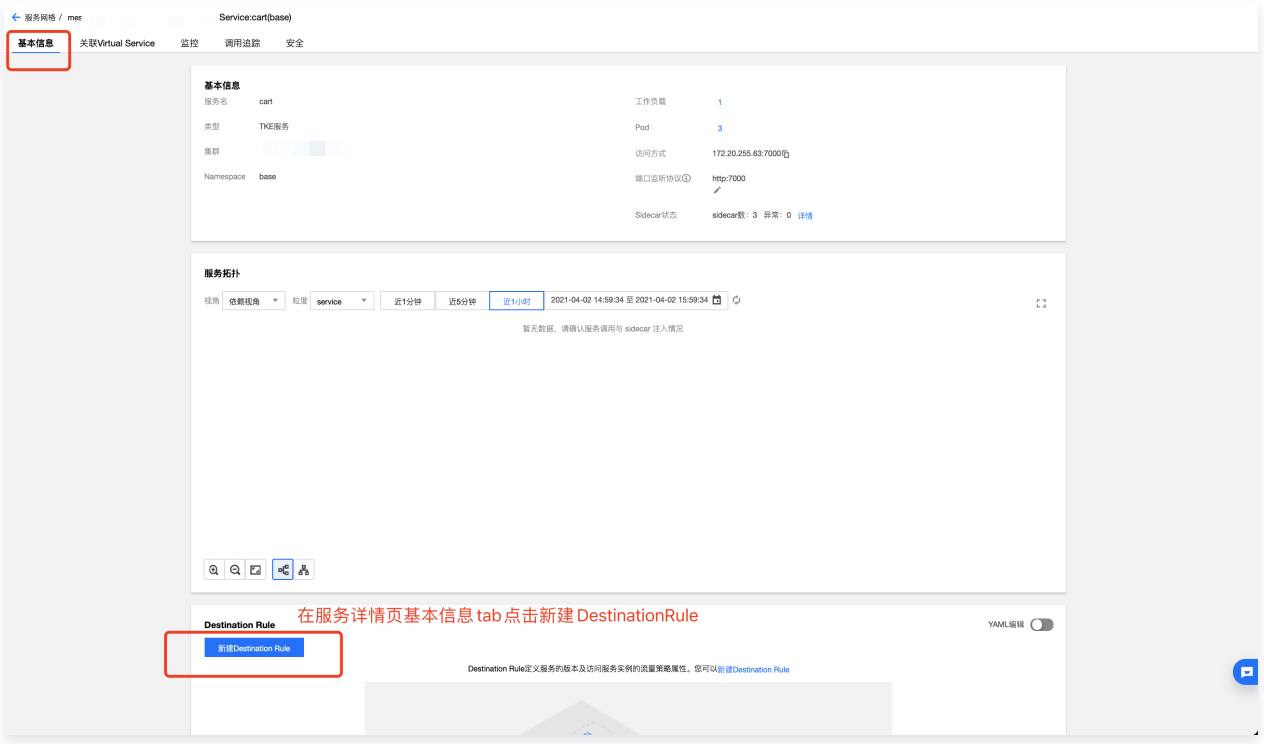


- 创建 VirtualService：



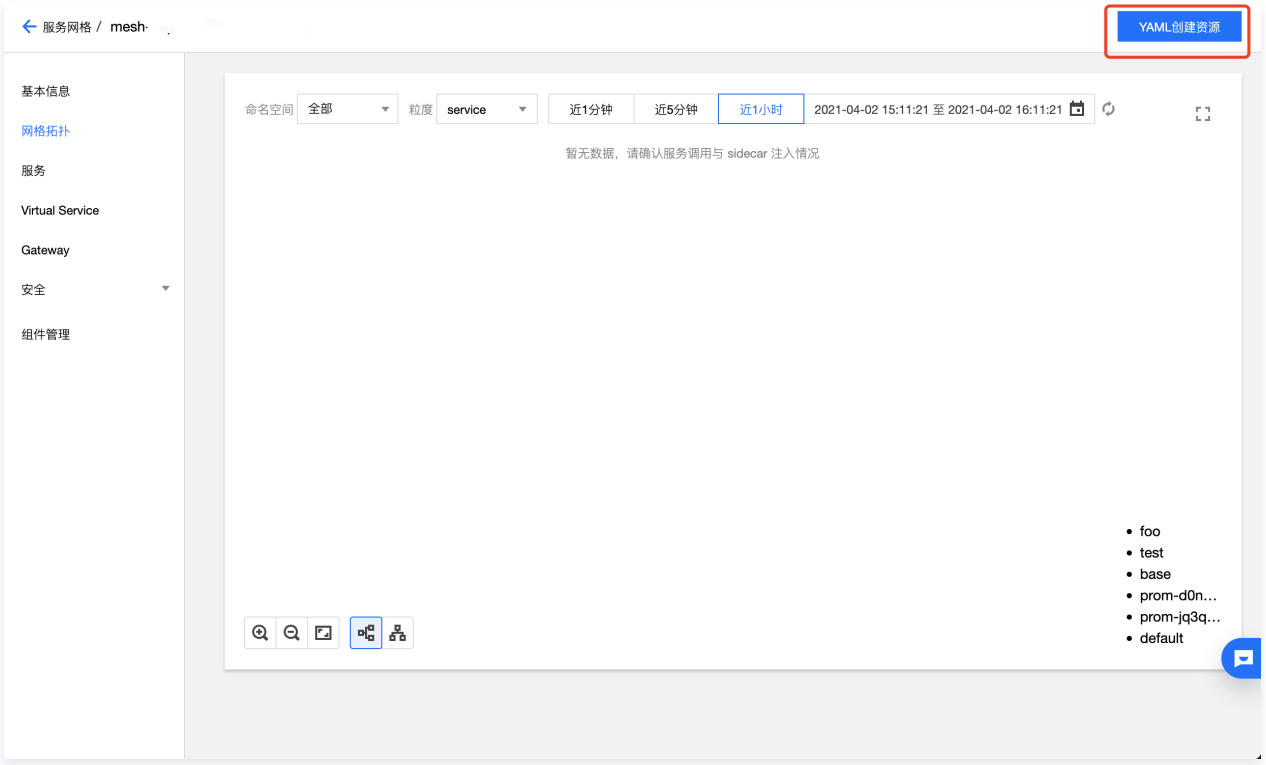
- 创建 DestinationRule：DestinationRule 与服务是一对一绑定关系，创建与管理在服务详情页面：





YAML 创建资源

您可以通过网格管理界面右上角的 **YAML 创建资源**创建 Istio 资源或 Kubernetes 资源。如您提交的 YAML 中含有 Kubernetes 资源，且当前网格管理了多个集群时，您需要选择 YAML 资源提交的目的集群。


服务网格 / mesh-
YAML创建资源
基本信息
网格拓扑
服务
Virtual Service
Gateway
安全
组件管理
命名空间
全部
粒度
service
近1分钟
近5分钟
近1小时
2021-04-02 15:11:21 至 2021-04-02 16:11:21
暂无数据，请确认服务调用与 sidecar 注入情况
foo
test
base
prom-d0n...
prom-jq3q...
default

VirtualService 配置路由规则

最近更新时间：2024-03-22 14:07:41

VirtualService 定义了指定 hosts 的一系列路由规则和流量操作（权重路由，故障注入等），其中每一条路由规则都定义了指定流量协议的匹配规则，如流量匹配，则被路由至指定的服务或服务的版本。VirtualService 配置主要包含以下部分：

- **hosts**：定义路由规则关联的 hosts，可以是带有通配符的 DNS 名称或者 IP 地址。
- **gateways**：定义应用路由规则的来源流量，可以是：
  - 一个或多个网关。
  - 网格内部的 sidecar。
- **路由规则**：定义详细的路由规则，包括 HTTP，TLS/HTTPS，TCP 三种协议类型的路由规则。
  - http：定义一组有序的应用于 HTTP 流量的路由规则。
  - tcp：定义一组有序的应用于 TCP 流量的路由规则。
  - tls：定义一组有序的应用于未终止的 TLS 或 HTTPS 流量的路由规则。

VirtualService 重要字段说明

以下是 VirtualService 的重要字段说明：

字段名称	字段类型	字段说明
`spec.hosts`	`string[]`	定义路由规则关联一组的 hosts，可以是带有通配符的 DNS 名称或者 IP 地址（IP 地址仅能应用于来源流量为边缘代理网关）。该字段能应用于 HTTP 和 TCP 流量。在 Kubernetes 环境中，可以使用 service 的名称作为缩写，Istio 会按照 VirtualService所在 namespace 补齐缩写，例如在 default namespace 的 VirtualService 包含 host 缩写 `reviews` 会被补齐为 `reviews.default.svc.cluster.local`。为避免误配置，推荐填写 host 全称。
`spec.gateways`	`string[]`	定义应用路由规则的来源流量，可以是一个或多个网关，或网格内部的 sidecar，指定方式为 `<gateway namespace>/<gateway name>`，保留字段 `mesh` 表示网格内部所有的 sidecar，当该参数缺省时，会默认填写 `mesh`，即该路由规则的来源流量为网格内部所有 sidecar。
`spec.http`	`HTTPRoute[]`	定义一组有序的（优先匹配靠前的路由规则）应用于 HTTP 流量的路由规则，HTTP 路由规则会应用于网格内部的 service 端口命名为 `http-`，`http2-`，`grpc-` 开头的流量以及来自 gateway 的协议为 `HTTP, HTTP2, GRPC, TLS-Terminated-HTTPS` 的流量。
`spec.http.match`	`HTTPMatchRequest[]`	定义路由的匹配规则列表，单个匹配规则项内所有条件是且关系，列表中多个匹配规则之间为或关系。
`spec.http.route`	`HTTPRouteDestination[]`	定义路由转发目的地列表，一条 HTTP 路由可以是重定向或转发（默认），转发的目的地可以是一个或多个服务（服务版本）。同时也可以配置权重、header 操作等行为。
`spec.http.redirect`	`HTTPRedirect`	定义路由重定向，一条 HTTP 路由可以是重定向或转发（默认），如规则中指定了 `passthrough` 选项，route、redirect 均会被忽略。可将 HTTP 301 重定向到另外的 URL 或 Authority。
`spec.http.rewrite`	`HTTPRewrite`	定义重写 HTTP URL 或 Authority headers，不能与重定向同时配置，重写操作会在转发前执行。
`spec.http.timeout`	Duration	定义 HTTP 请求的超时时间。
`spec.http.retries`	`HTTPRetry`	定义 HTTP 请求的重试策略。
`spec.http.fault`	`HTTPFaultInjection`	定义 HTTP 流量的故障注入策略，开启时超时和重试策略不会开启。
`spec.http.mirror`	`Destination`	定义将 HTTP 流量复制到另一个指定的目的端，被复制的流量按照“best effort”原则，sidecar/网关不会等待复制流量的响应结果就会从源目的端返回响应。镜像流量的目的服务端会产生监控指标。

spec.http.mirrorPercent	uint32	定义流量镜像的复制百分比，缺省时复制100%的流量。最大值为100。
spec.http.corsPolicy	CorsPolicy	定义 CORS 策略（跨域资源共享，Cross-Origin Resource Sharing，CORS），更多关于 CORS 的介绍请参见 CORS，关于 Istio CORS 策略配置语法请参见 CorsPolicy
spec.http.headers	Headers	定义 header 操作规则，包括 request 和 response header 的更新，增加，移除操作。
spec.tcp	TCPRoute[]	定义一组有序的（优先匹配靠前的路由规则）应用于 TCP 流量的路由规则，该路由规则会应用于任何非 HTTP 和 TLS 的端口。
spec.tcp.match	L4MatchAttributes[]	定义 TCP 流量路由的匹配规则列表，单个匹配规则项内所有条件是且关系，列表中多个匹配规则之间为或关系。
spec.tcp.route	RouteDestination[]	定义 TCP 连接转发的目的端。
spec.tls	TLSRoute[]	定义一组有序的（优先匹配靠前的路由规则）应用于未终止的 TLS 或 HTTPS 流量的路由规则，该路由规则会应用于网格内部的 service 端口命名为 `https-`，`tls-` 开头的流量，来自 gateway 的端口协议为 `HTTPS, TLS` 的未终止加密流量，Service Entry 使用 `HTTPS, TLS` 协议的端口。当 `https-`，`tls-` 端口未关联 VirtualService 规则时将会被视为 TCP 流量。
spec.tls.match	TLSMatchAttributes[]	定义 TLS 流量路由的匹配规则列表，单个匹配规则项内所有条件是且关系，列表中多个匹配规则之间为或关系。
spec.tls.route	RouteDestination[]	定义连接转发的目的端。

配置来自 Gateway 流量（南北向）的路由规则

VirtualService 可通过控制台 UI 和 YAML 编辑两种方式配置。下面将展示将来自 Gateway 流量路由至服务 frontend 的 VirtualService 配置。相关的 Gateway 配置如下：

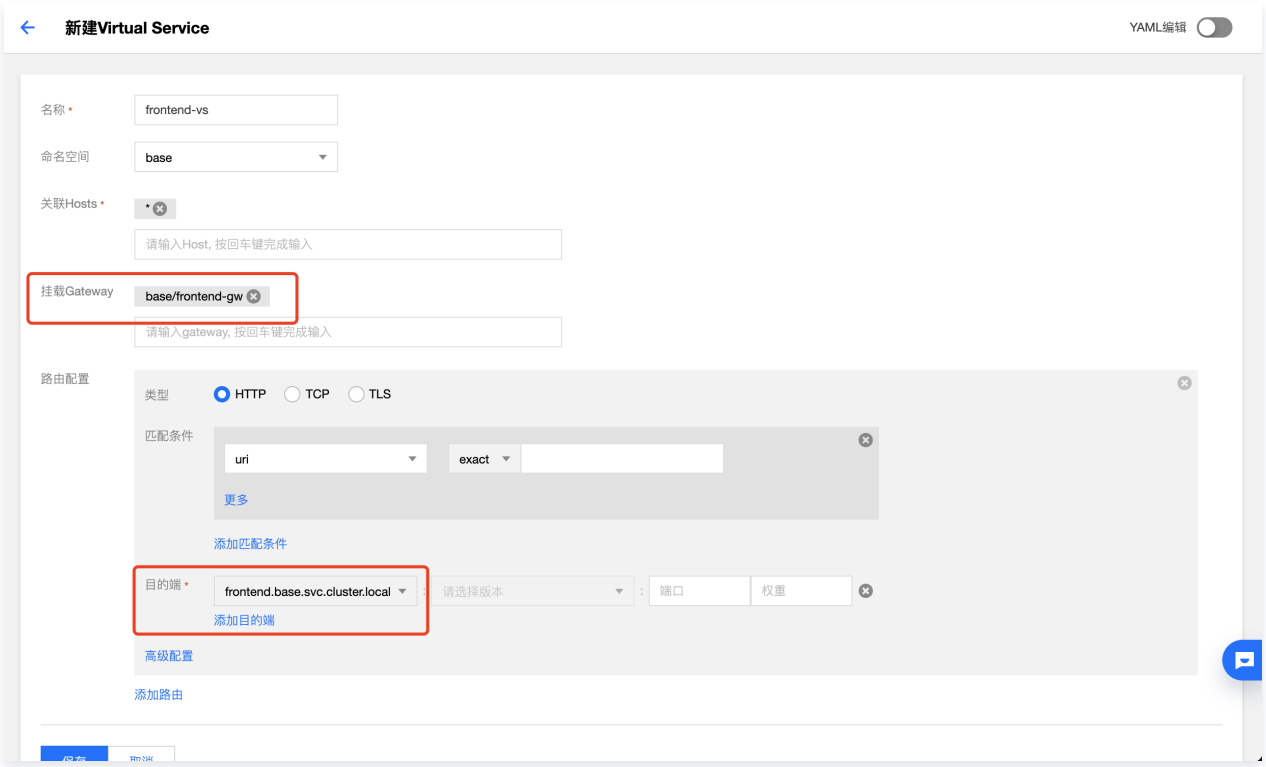
```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: frontend-gw
  namespace: base
spec:
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - '*'
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
```

YAML 配置示例

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

```
  hosts:
    - '*'
  gateways:
    - base/frontend-gw # VS 挂载的 Gateway 填写按照 {namespace}/{Gateway 名称}
  http:
    - route:
        - destination:
            host: frontend.base.svc.cluster.local # 路由目的地填写 frontend 服务的 host 全称
```

控制台配置示例



配置来自 Mesh 内部流量（东西向）的路由规则

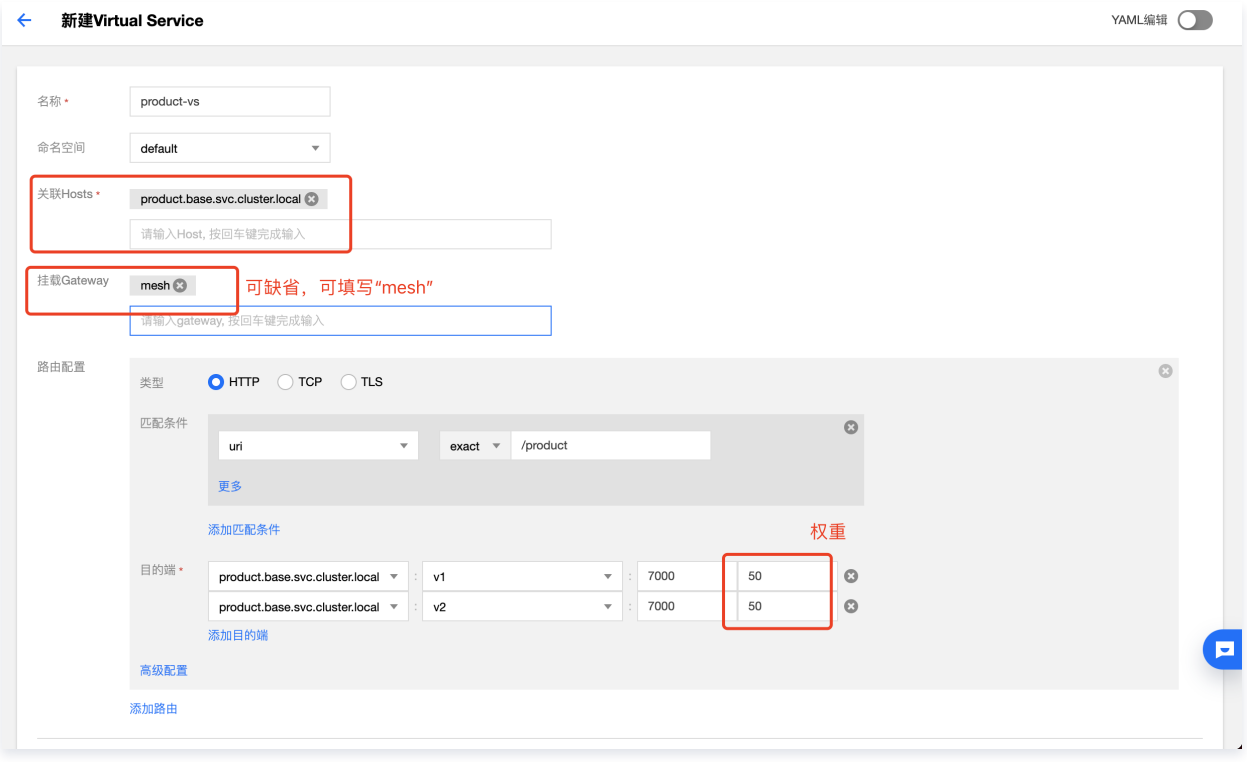
以下是设置将来自网格内部访问 product 服务 host: `product.base.svc.cluster.local` 的流量的路由规则：50%的流量路由至 v1 版本，50%的流量路由至 v2 版本的 VirtualService 配置示例（灰度发布）。其中 product 的服务版本是通过以下 DestinationRule 定义：

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: product
  namespace: base
spec:
  host: product
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
```

YAML 配置示例

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: product-vs
  namespace: base
spec: # 缺省 gateway 参数，表示该路由配置应用于 mesh 内部 sidecar 的流量
  hosts:
    - "product.base.svc.cluster.local" # 匹配访问该 host 的流量
  http:
    - match:
        - uri:
            exact: /product
      route:
        - destination: # 配置目的端及权重
            host: product.base.svc.cluster.local
            subset: v1
            port:
              number: 7000
          weight: 50
        - destination:
            host: product.base.svc.cluster.local
            subset: v2
            port:
              number: 7000
          weight: 50
```

控制台配置示例



DestinationRule 配置服务版本和流量策略

最近更新时间：2024-03-22 14:07:41

DestinationRule 定义服务的版本和路由发生后的流量策略，包括负载均衡、连接池大小、健康检查（从负载均衡后端中剔除不健康的 hosts）等流量策略。服务与 DestinationRule 是一对一的绑定关系。

DestinationRule 重要字段说明

以下是 DestinationRule 的重要字段说明：

字段名称	字段类型	字段说明
`spec.host`	`string`	关联 DestinationRule 配置的服务名称，可以是自动发现的服务（例如 Kubernetes service name），或通过 ServiceEntry 声明的 hosts。如填写的服务名无法在上述源中找到，则该 DestinationRule 中定义的规则无效。
`spec.subsets`	`Subset[]`	定义服务的版本（subsets），版本可通过标签键值对匹配服务中的 endpoints。可以在 subsets 级覆盖流量策略配置。
`spec.trafficPolicy`	`trafficPolicy`	定义流量策略，包括负载均衡、连接池、健康检查、TLS 策略。
`spec.trafficPolicy.loadBalancer`	–	配置负载均衡算法，可配置：简单负载均衡算法（round robin, least conn, random...），一致性哈希（会话保持，支持按 header name，cookie，IP，query parameter 哈希），地域感知负载均衡算法。
`spec.trafficPolicy.connectionPool`	–	配置与上游服务的连接量，可设置 TCP/HTTP 的连接池。
`spec.trafficPolicy.outlierDetection`	–	配置从负载均衡池中驱逐不健康的 hosts。
`spec.trafficPolicy.tls`	–	连接上游服务的 client 端 TLS 相关配置，与 PeerAuthentication 策略（server 端 TLS 模式配置）配合使用。
`spec.trafficPolicy.portLevelSettings`	–	配置端口级别的流量策略，端口级别的流量策略会覆盖服务 / subsets 级别的流量策略配置。

定义服务的版本（subsets）

DestinationRule 可定义服务的版本（subsets），而 subset 则是腾讯云服务网格的最小流量管理单元。例如，您可以配置将流量路由至某个指定服务的指定 subset。以下是 DestinationRule 定义 product 服务两个 subset 的配置示例。

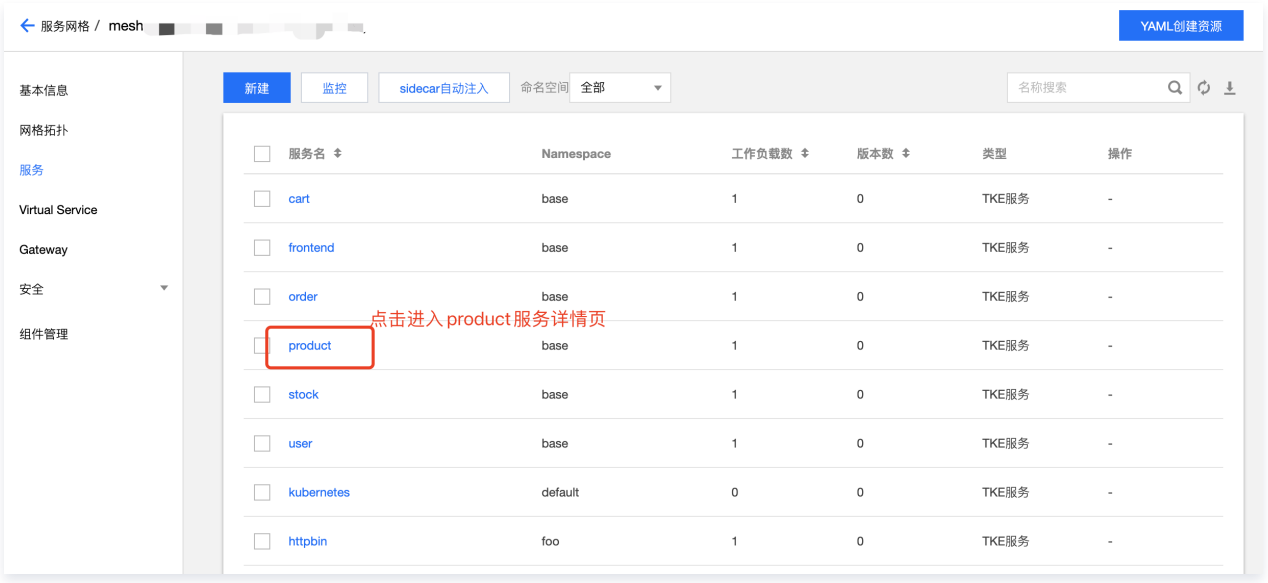
YAML 配置示例

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: product
  namespace: base
spec:
  host: product
  subsets:
    - name: v1
      labels:
        version: v1 # subset v1 通过标签 version:v1 来匹配该服务的endpoint
    - name: v2
      labels:
        version: v2 # subset v2 通过标签 version:v2 来匹配该服务的endpoint
```

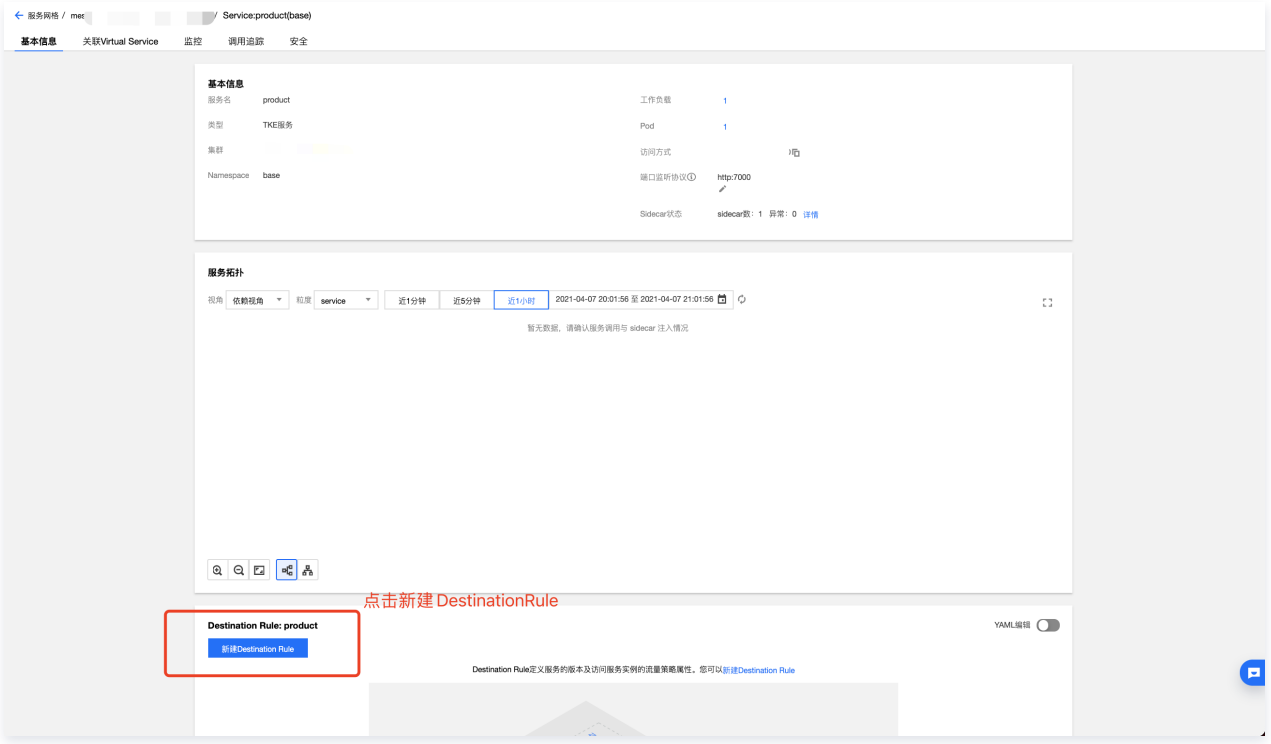
控制台配置示例

DestinationRule 和服务是一一对应的绑定关系，配置 product 服务的 DestinationRule，需要从服务列表页进入 product 服务的详情页，在基本信息 Tab 配置。控制台配置 product 服务两个版本的步骤如下：

1. 在服务列表页面，单击进入 product 服务的详情页面。



2. 在服务详情页**基本信息**，第三个 DestinationRule card 区域，单击**新建 DestinationRule** 进入新建弹窗。



3. 在弹窗页面为 product 服务添加两个版本，点击保存。

4. product 服务版本配置完成。

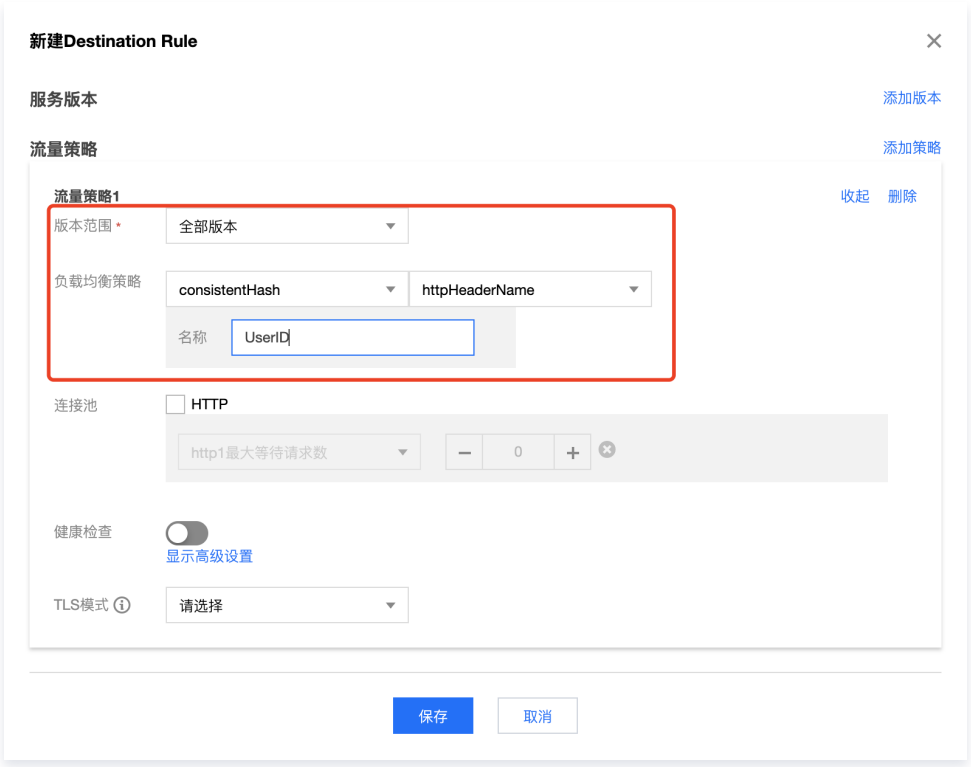
配置一致性哈希负载均衡

以下是用 DestinationRule 配置 cart 服务按照 http header name 做一致性哈希负载均衡的配置示例。

YAML 配置示例

```
kind: DestinationRule
metadata:
  name: cart
  namespace: base
spec:
  host: cart
  trafficPolicy:
    loadBalancer:
      consistentHash:
        httpHeaderName: UserID # 配置访问 cart 服务的流量按照 header UserID 做一致性哈希负载均衡
```

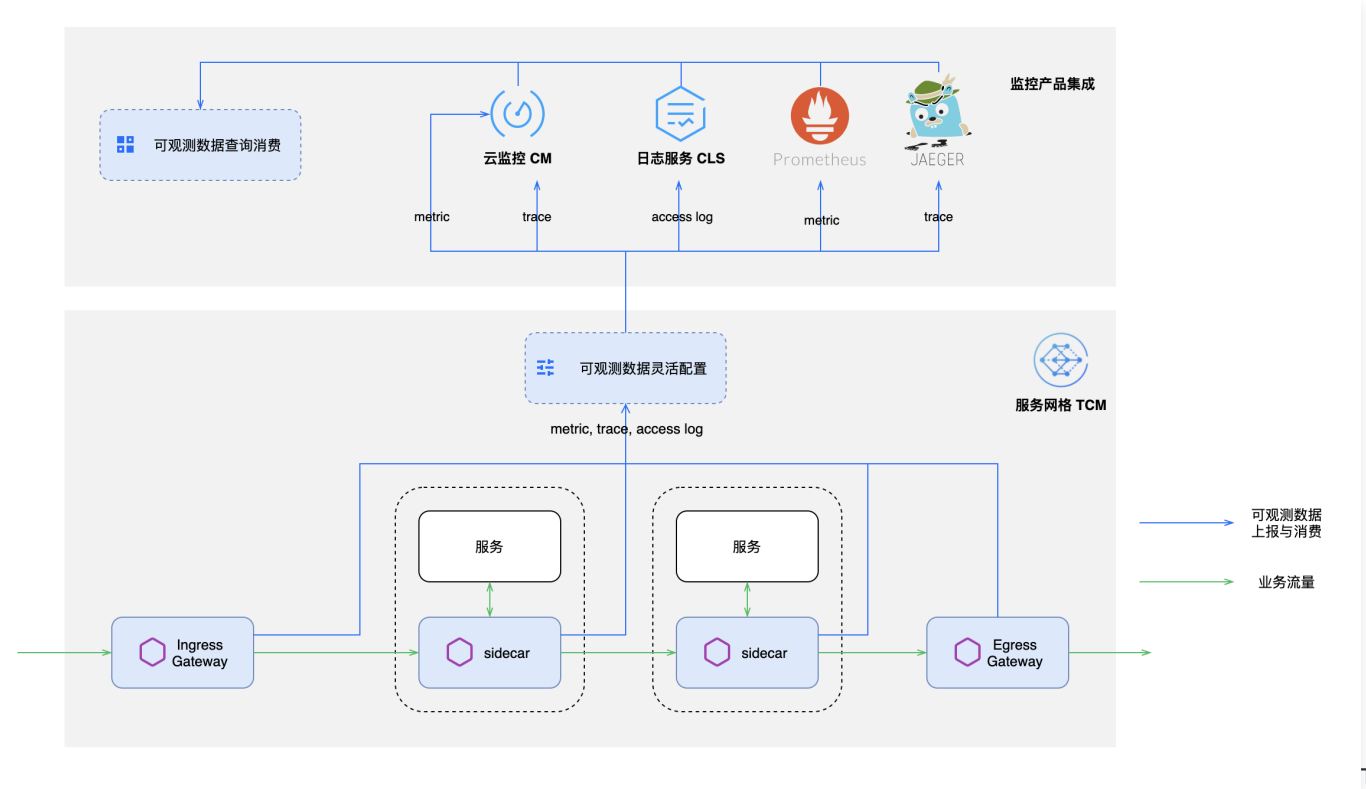
控制台配置示例



可观测性
概述

最近更新时间：2024-03-22 14:07:41

腾讯云服务网格提供南北向和东西向的服务间端到端的可观测能力。
观测数据的收集，依赖于注入服务的 envoy sidecar proxy （数据面）上报来完成。您可以通过 TCM 灵活控制数据面可观测数据的生产计算，TCM 会将观测数据集成到合适的监控周边产品，为您提供网格边缘与内部的服务间流量的可观测能力。



TCM

提供三种类型的可观测数据：

可观测数据类型	说明
监控指标 Metric	监控指标可为您提供服务或网关的流量观测数据，适合单个服务的开发者关注。
调用追踪 Trace	调用追踪可将一次业务请求的多层调用联系成为一次调用链路，便于您观测调用结构，做性能分析与异常定位。
访问日志 Access Log	访问日志是 envoy proxy 生成每次请求的完整记录，包括请求层和 sidecar 代理层的信息，便于运维人员进行访问审计与错误排查。

三种类型的可观测数据信息如下表所示：

观测数据	记录信息	适用场景或角色
Metric	单个服务或网关的流量观测数据，包括但不限于延时、请求数、请求大小等指标，未采样。更多指标信息请参见 Istio Standard Metrics 。	单个服务业务开发监测服务运行状况。
Trace	记录服务间的调用依赖，相比 metric 增加 url 维度的信息，但记录的数据一般经过了采样	业务整体开发者，做所有服务的调用依赖与性能分析。
Access Log	完整记录每次请求的信息，包括 sidecar 代理层丰富的信息输出，更多信息请参见 Envoy Access Logging 。	服务网格运维人员做访问审计与错误排查。

监控指标 Metric

最近更新时间：2024-09-02 17:00:21

当前腾讯云服务网格使用 Prometheus 作为监控指标存储方案，您可选择使用 Prometheus 监控 TMP 为您提供服务流量 metric 数据的收集、存储与展示，也可以使用第三方 Prometheus 服务，作为监控指标存储服务。
服务网格数据面 Sidecar 将 Metric 数据上报到 TMP 或第三方 Prometheus 服务，控制台从对应的服务查询指标数据，并提供网格拓扑、服务拓扑、服务监控（请求数、请求状态码分布、请求耗时、请求大小）图表的展示分析。此外，如果您有自定义监控的诉求，可以通过 TMP 中的 Grafana 面板设置自定义的监控面板。

操作步骤

开启 TMP 监控

在 创建网格 或网格基本信息页中的可观测性配置 > 监控指标中，勾选**腾讯云 Prometheus 监控 TMP**，按需选择自动创建或者关联已有 TMP 实例即可。开启后，Sidecar 将会将 metric 数据上报到对应的实例，您也可以在 TMP 控制台查看该实例。

开启第三方 Prometheus 服务

在 创建网格 或网格基本信息页中的可观测性配置 > 监控指标中，勾选第三方 Prometheus 服务，填写该服务对应的 VPC 信息、地址、认证信息。

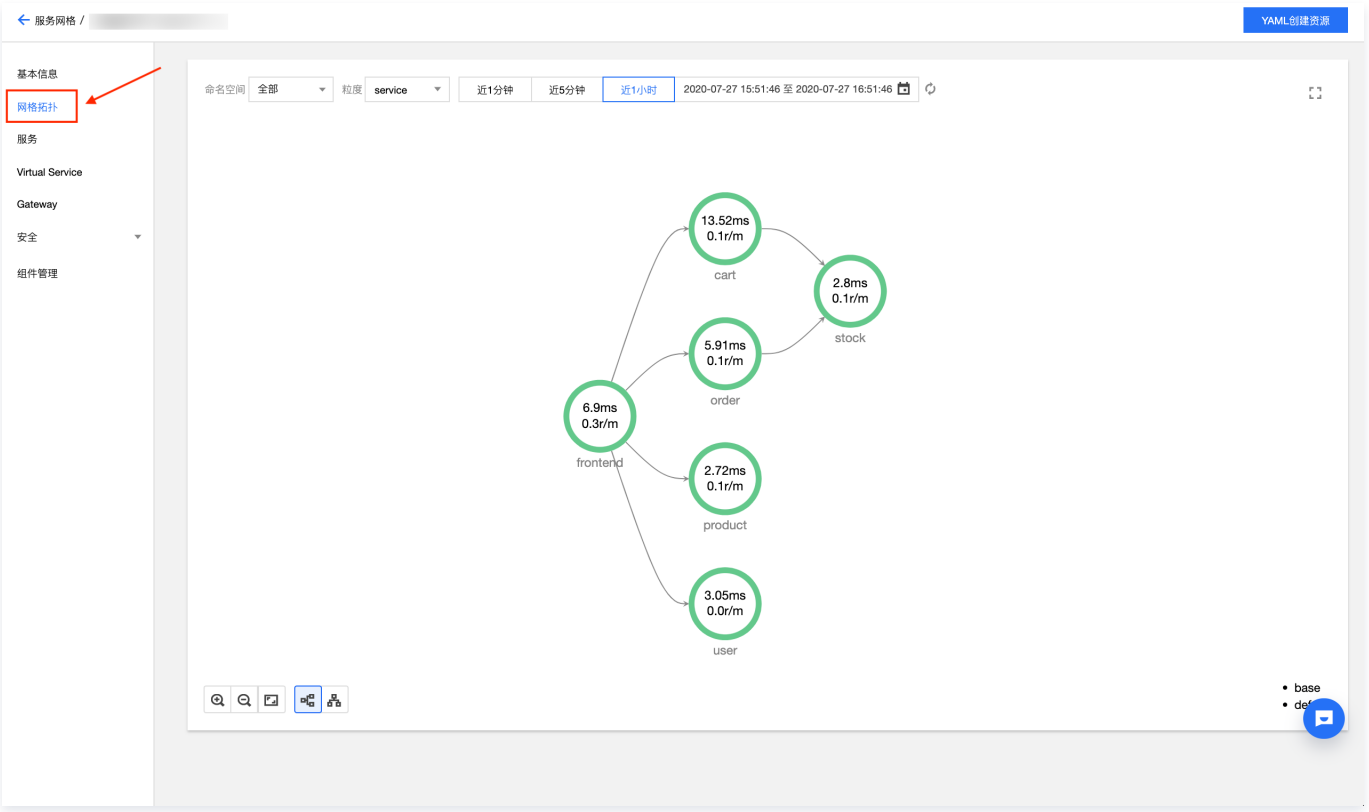
> **说明：**
> 由于 TCM 控制台会从 Prometheus 服务中查询和展示监控指标数据，对数据源服务的网络可达性、稳定性要求较高，因此当前仅支持通过内网的形式访问。

查看监控相关图表

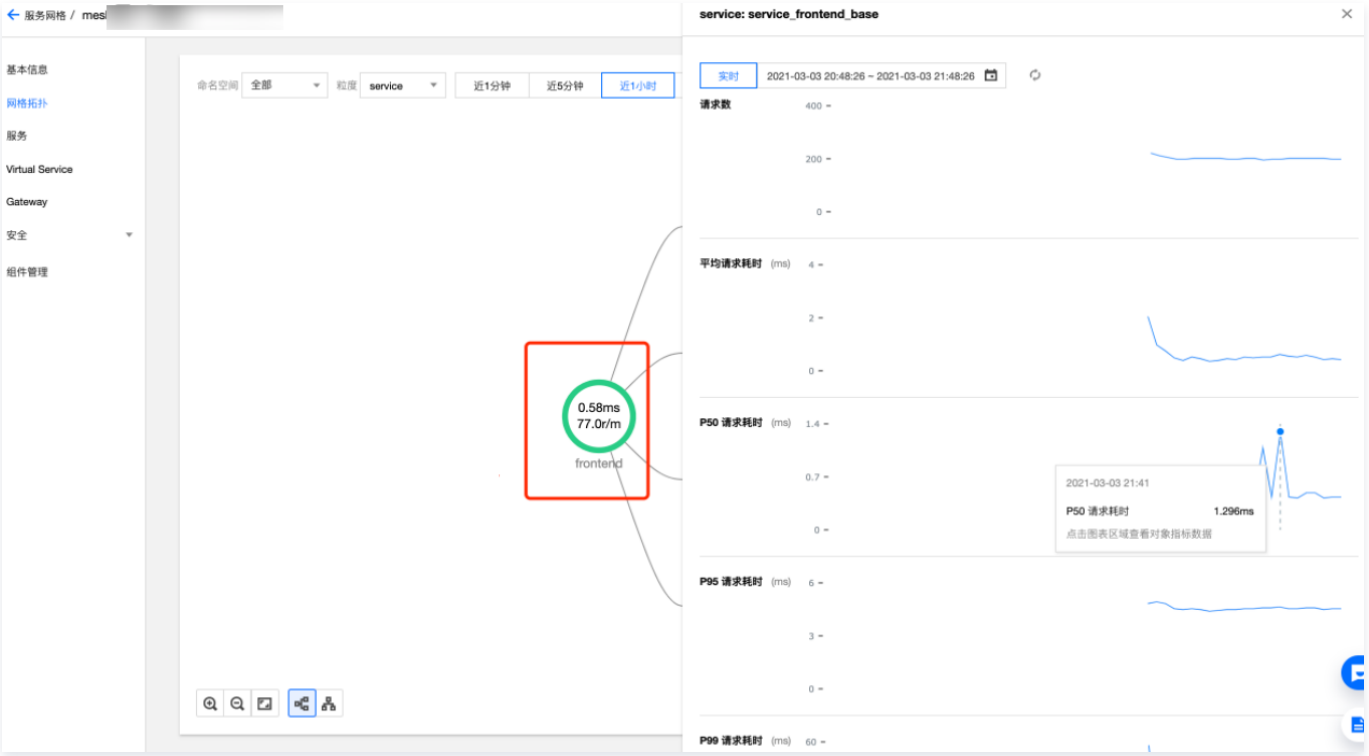
网格拓扑

记录服务网格所有服务的调用结构，查看网络拓扑需确保相关服务已注入 sidecar，且存在请求流量。查看指定网格的网络拓扑流程如下：
1. 登录 服务网格控制台，在列表页面单击指定网格 ID，进入网格详情页面。

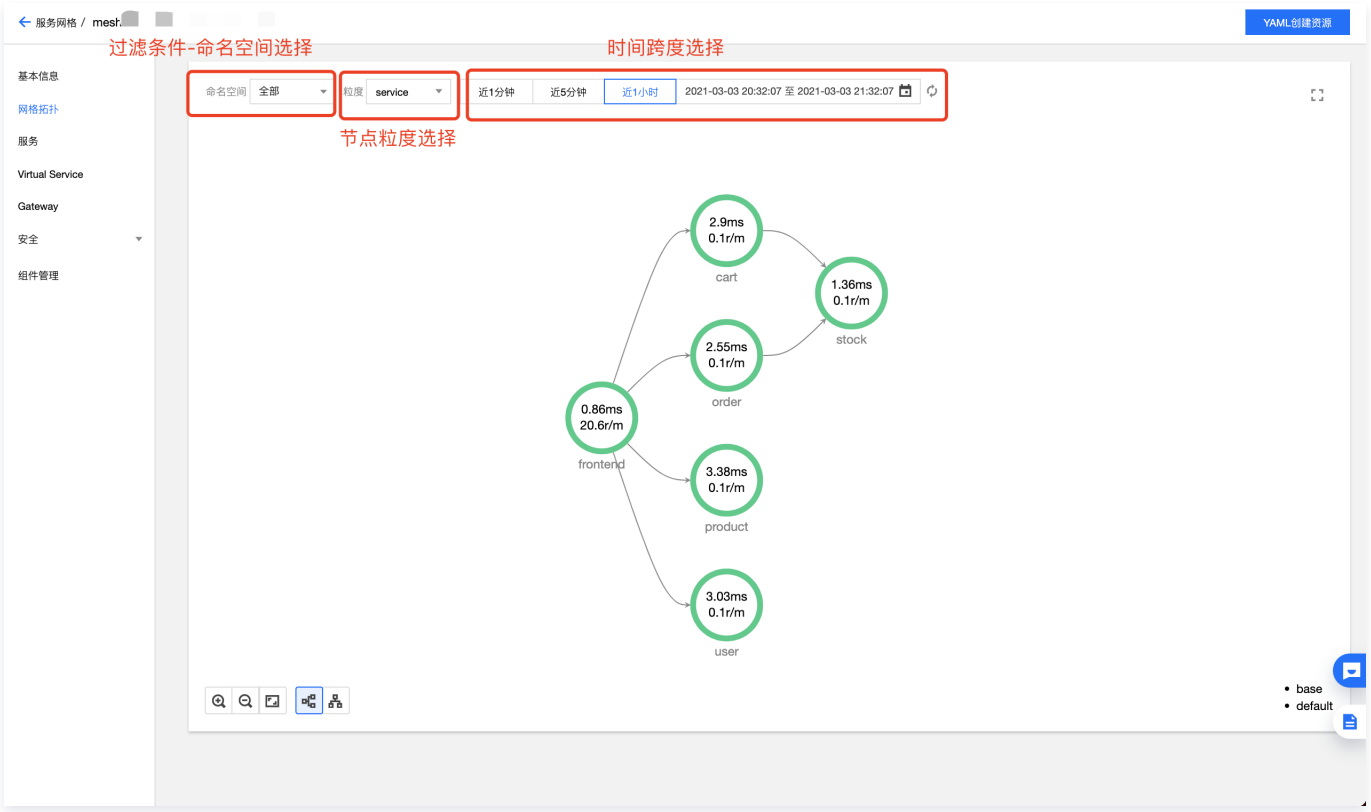
2. 单击左侧**网格拓扑**页签，即可查看当前服务网格拓扑图。如下图所示：



3. 单击节点可展示该节点相关的监控详情。如下图所示：



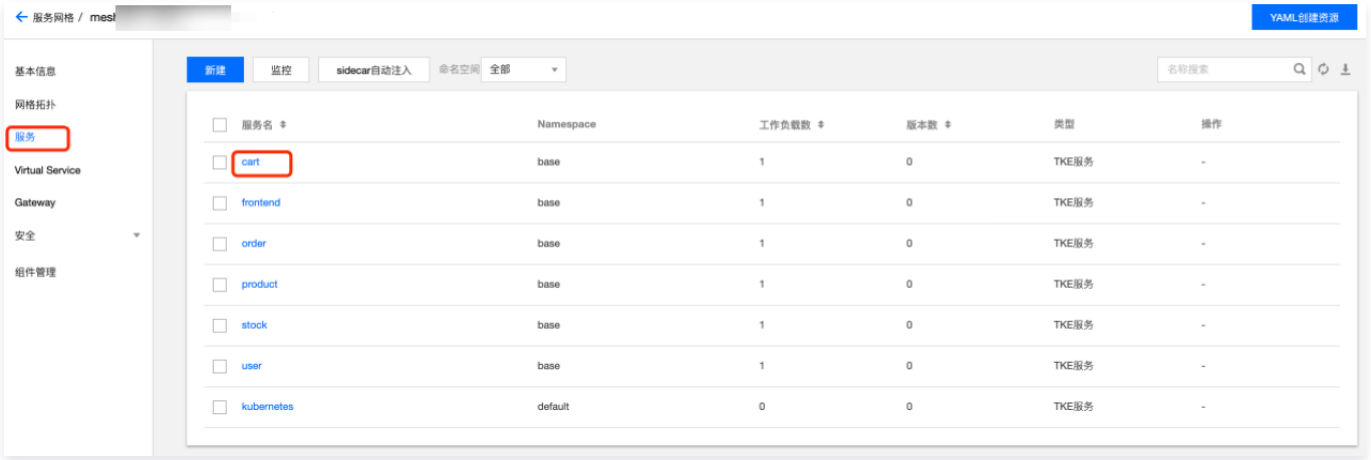
4. 界面上方可以选择数据过滤条件，包括 namespace 与时间跨度；支持切换节点的粒度，当前支持 service 粒度和 workload 粒度。如下图所示：



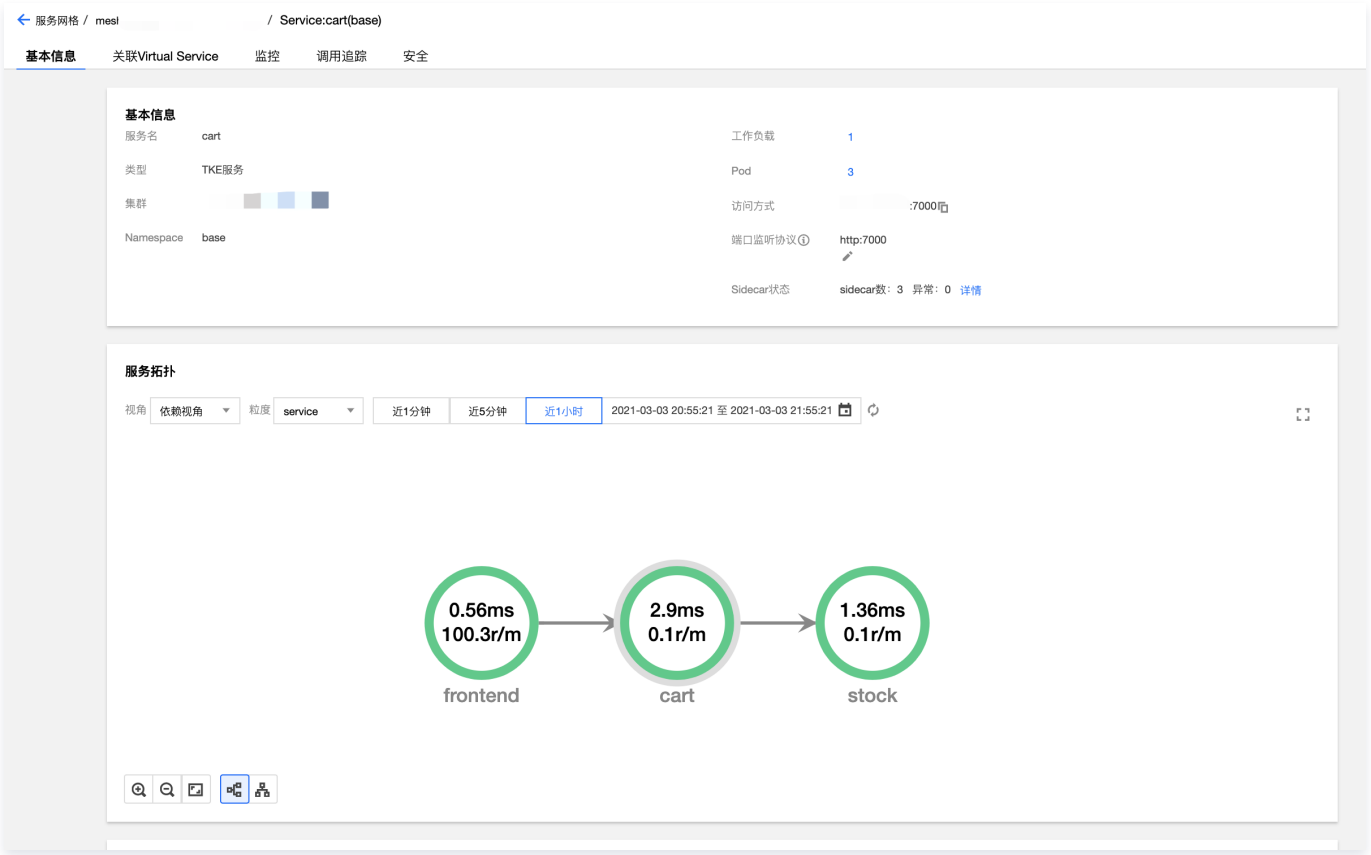
服务拓扑

记录某个服务的前后调用依赖关系，查看指定服务的服务拓扑流程如下：

1. 在指定网格的详情页面，单击左侧**服务**进入服务列表页。
2. 单击想要查看的服务，进入服务详情页。



3. 在服务详情页面的“基本信息”中，即可查看该服务的服务拓扑。如下图所示：



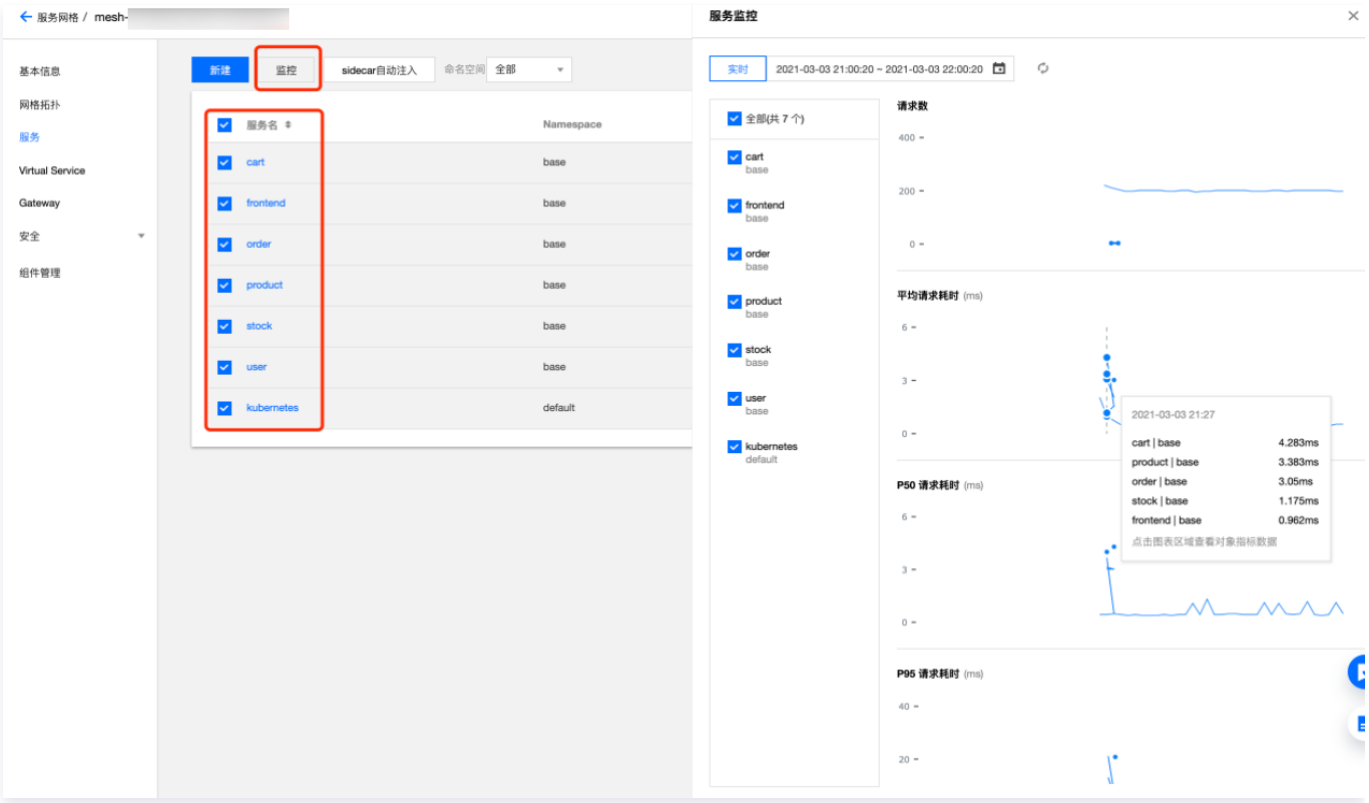
服务监控

您可以在服务列表页面对比多个服务的监控数据（请求数、请求耗时、请求大小等），或在服务详情页面查看指定服务的监控详情。

- **在服务列表页查看多个服务的监控数据：**

  1.1 登录 服务网格控制台，在列表页面单击指定网格 ID，进入网格详情页面。

1.2 选择**服务 > 监控**，单击需查看监控数据的服务并在右侧查看服务监控数据。如下图所示：



- **在服务详情页面查看指定服务的监控数据详情图表：**

  1.1 在指定网格的详情页面，单击左侧**服务**进入服务列表页。

  1.2 单击想要查看的服务，进入服务详情页。

  1.3 在服务详情页面的“监控”中即可查看。

关闭监控

您可以在**网格基本信息页**选择编辑可观测性配置，取消勾选**腾讯云 Prometheus 监控 TMP**，取消勾选后，服务网格侧并不会删除 TMP 实例，如有需要，请在 TMP 控制台 进一步删除该 TMP 实例。

调用追踪 Trace

最近更新时间：2024-03-22 14:07:42

服务网格默认集成应用性能监控 APM 作为调用追踪消费端，开启后网格将会为您创建一个 APM 实例并将 tracing 数据上报到对应的 APM 实例，您可以在服务网格控制台查看到请求在网格内的完整调用瀑布图和每层调用的 trace 日志信息，帮助您理解服务的调用依赖，以及做网格内的延迟分析。您也可以直接在 APM 控制台查看调用相关数据。
除了 APM 之外，网格支持将调用数据上报到第三方 Jaeger/Zpkin 服务，如果开启使用第三方 tracing 服务，则服务网格控制台将无法展示调用追踪相关信息，需要在第三方服务中查看。

配置调用追踪采样率

调用追踪采样率是 trace 数据的采样比例，sidecar 采集与上报数据消耗资源与带宽和采样率成正相关。通常生产环境下，不需要为所有调用都生成 trace 数据并采集上报，避免过多消耗计算资源与带宽资源，只需要配置一定比例即可。建议开发测试环境可以配置100%采样率，生产环境配置1%采样率。
您可以通过以下两种方式配置调用追踪采样率：

方式1：在网格创建时配置采样率

1. 登录 服务网格控制台 。
2. 选择地域，单击页面左上角的**新建**。
3. 在创建服务网格页面，按需填写网格创建相关配置，在“可观测性配置”中配置采样率。如下图所示：

方式2：在网格基本信息页面修改采样率配置

1. 登录 服务网格控制台 。
2. 选择地域，单击需要变更配置的网格 ID，进入网格的管理页面。

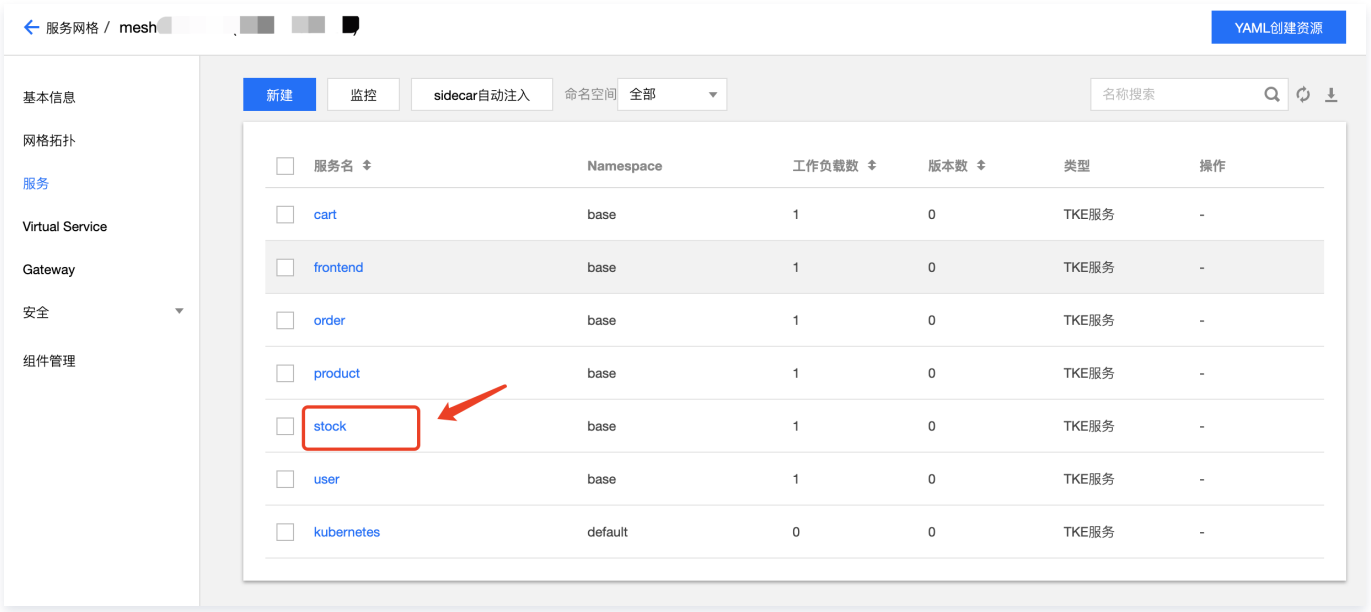
3. 在网格基本信息页面，单击**调用追踪 > 采样率** 的 ，调整采样率。如下图所示：

查看调用追踪

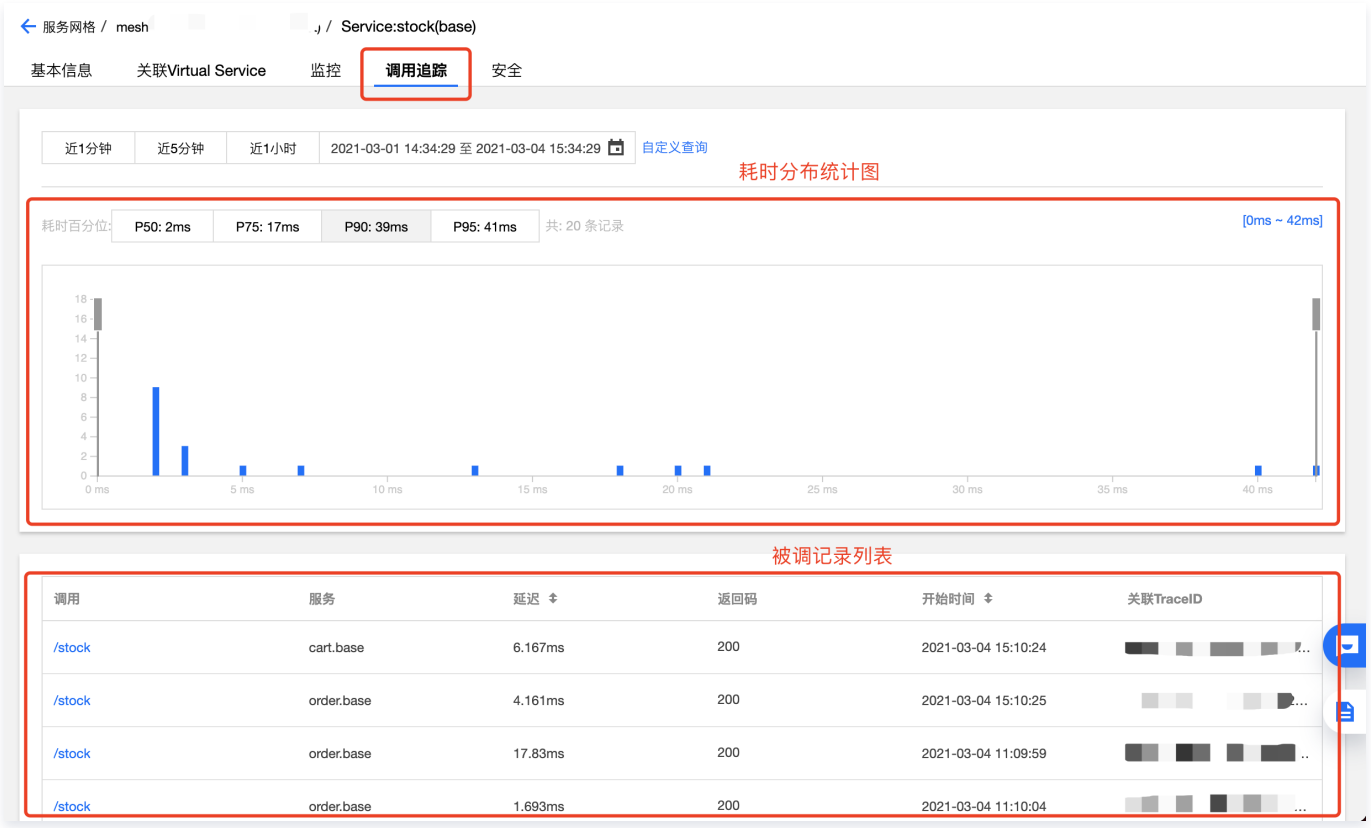
> **说明：**
> 由于服务网格中的 service 并非任何情况下都与 k8s service 完全一一对应，tracing span 中上报服务名时实际使用 pod 中 app label值作为服务名（k8s service 亦是通过此 label 关联 pod，但 service name 可以和 label 不同）。控制台查询服务调用链信息时，服务名默认使用 k8s service 名查询，因此如果 k8s service 名 和 pod 的 app label 不匹配时，控制台无法查询到相应数据。

查看调用追踪的流程如下：

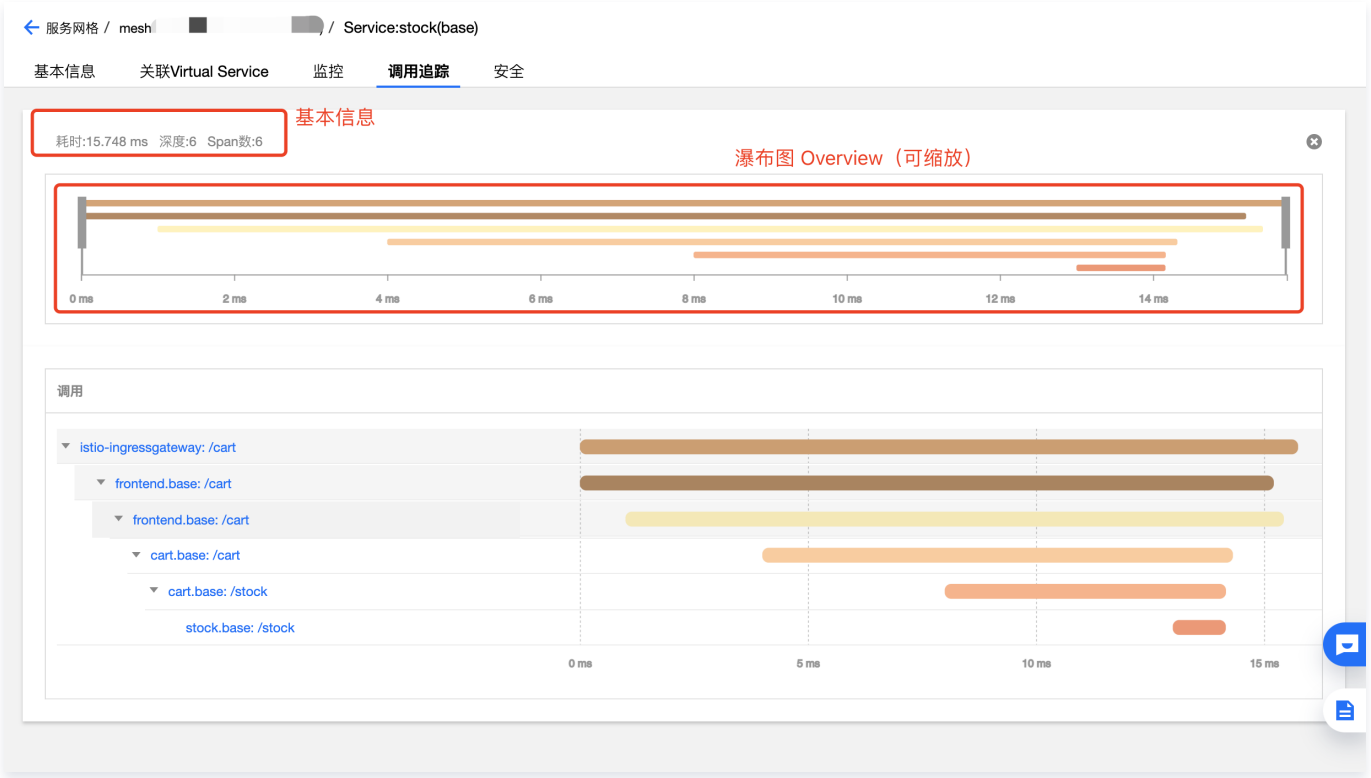
1. 需要关注网格某个服务的调用情况，在网格的服务列表页面单击该服务，进入服务详情页面。

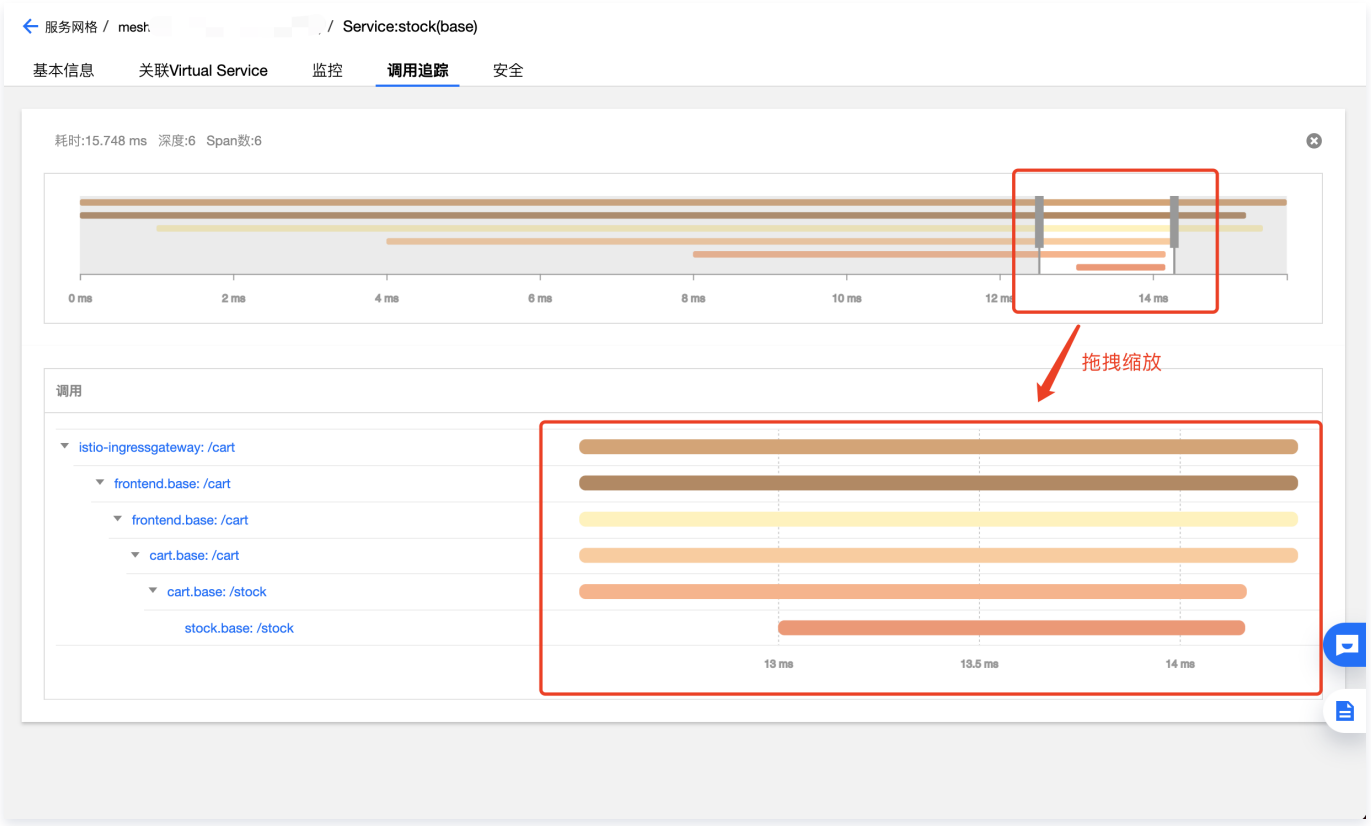


2. 在服务详情页面单击**调用追踪**，您可以查看到该服务作为被调方，被调用的记录列表，以及这些调用记录的耗时分布统计直方图。

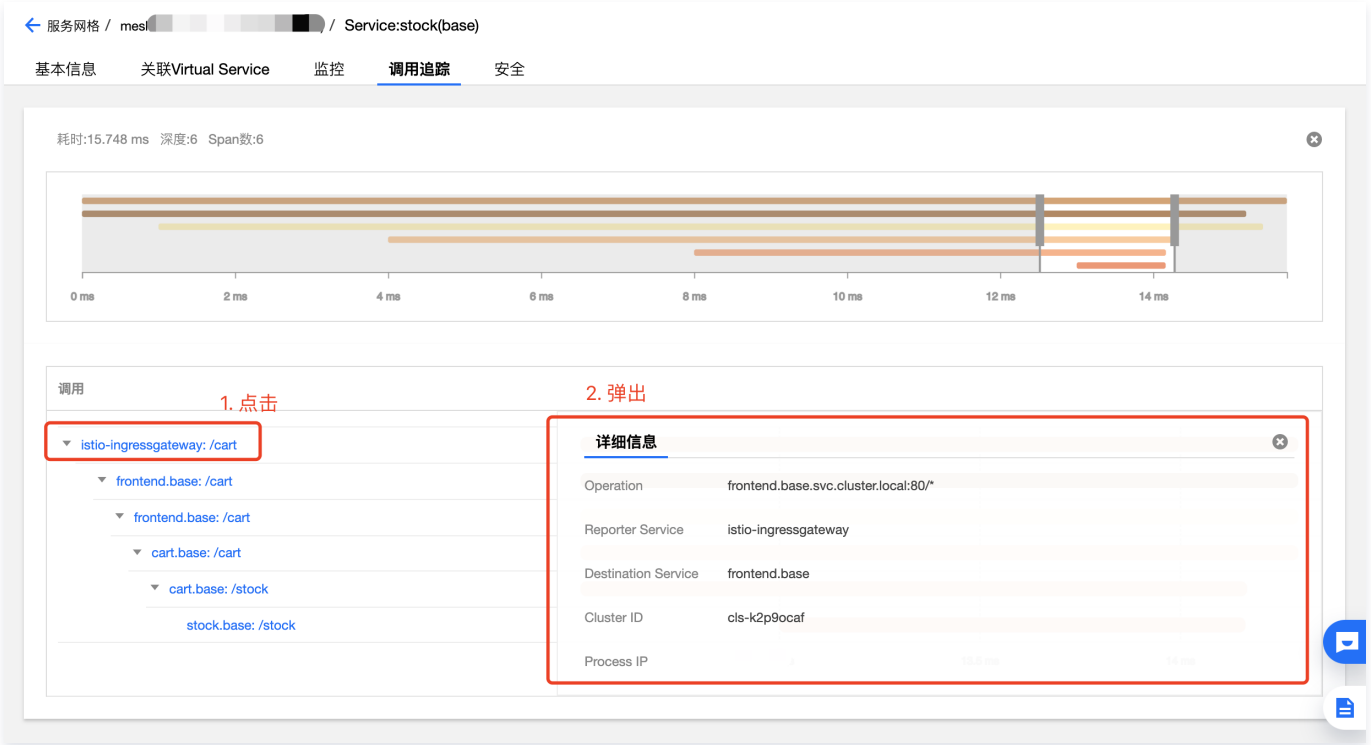


3. 被调记录列表第一列记录了调用的 URL，单击即可查看该调用相关的完整调用链路瀑布图，上方瀑布图概览可以拖拽实现缩放。

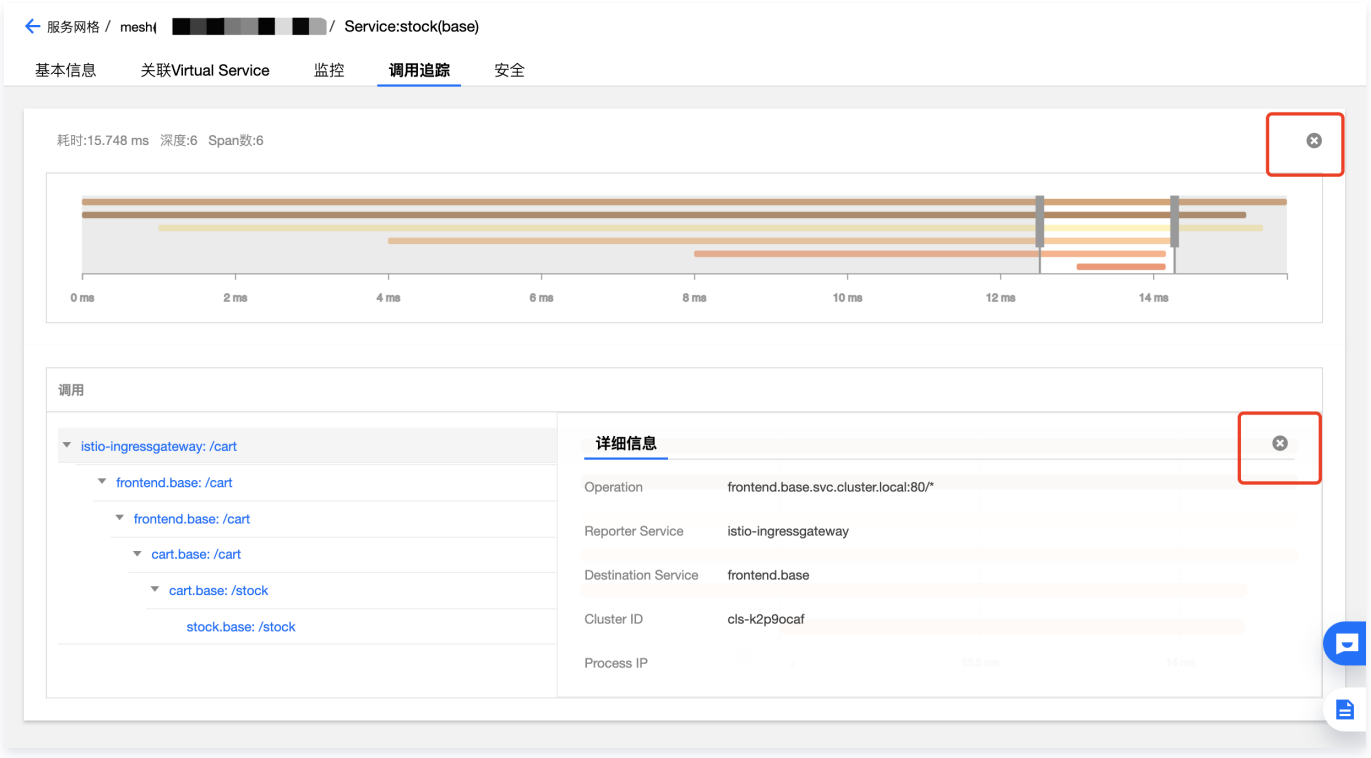




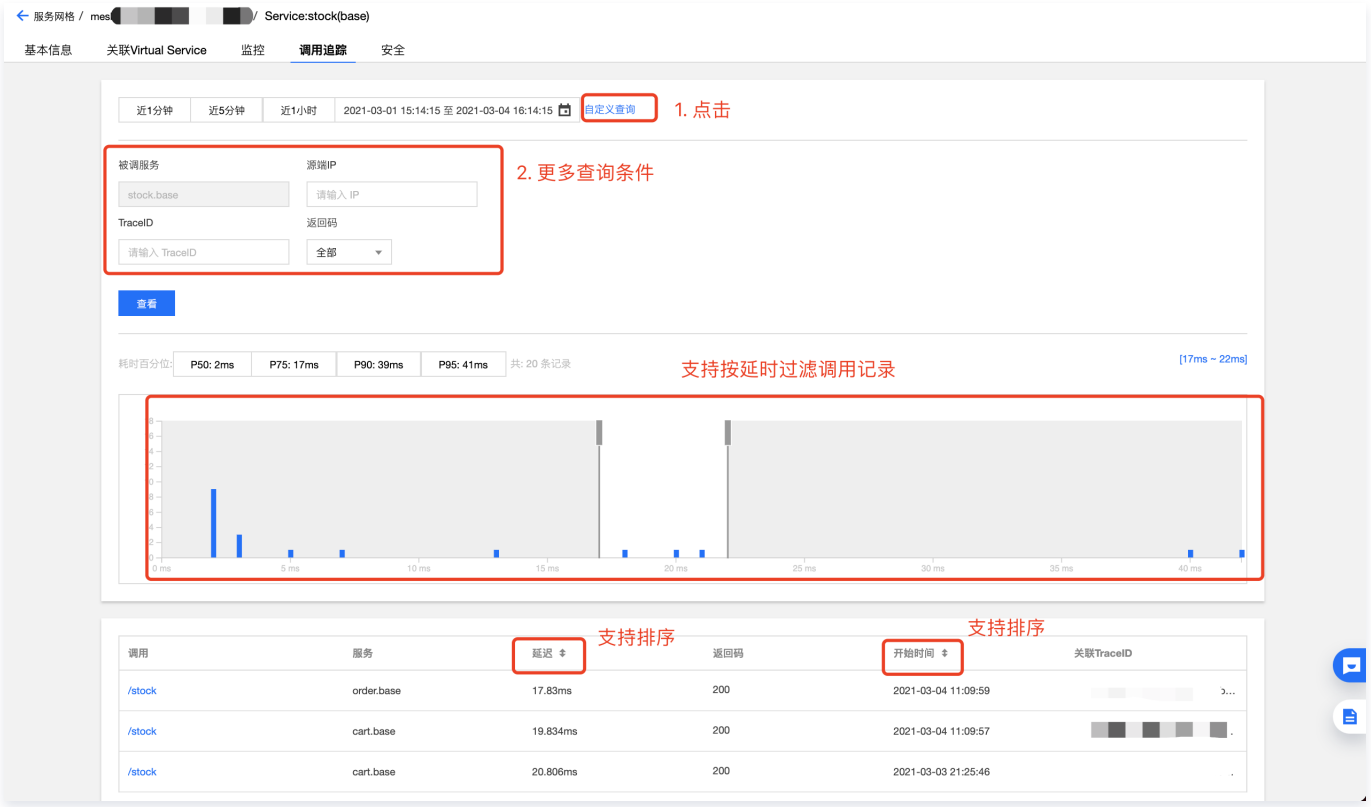
4. 单击想要查看详情的调用，可以查看对应调用环节的详细 trace 日志。



5. 单击关闭按钮可关闭 span 详情，以及返回服务被调记录列表页。



6. 查询服务被调记录 Tips：您可以按照耗时、时间跨度、源端IP、Trace ID、返回码过滤被调记录，过滤完成后您可以按照**延时**和**开始时间**对调用记录排序，方便您选择查看需要关注的调用。



查看完整调用链

完整调用链形成的基本条件是需要请求经过每个环节的调用链路信息都能往下传递直到请求结束，虽然 Sidecar 在转发请求时会增加链路追踪相关的信息。但由于 Sidecar 处理 in、out 流量是在两个独立的步骤中进行的，且这两个步骤之间是 Sidecar 无法干预的业务逻辑环节（由于传递链路被业务逻辑步骤打断，Sidecar 也失去了 in、out 流量映射关系），所以如果业务代码处理业务请求时不转发这些额外的调用链路信息，则链路形成将会中断。因此服务网格无法实现完全无侵入的全链路追踪，需要在业务代码中添加链路追踪相关信息的转发逻辑：

服务网格通过在7层流量中增加符合 B3 trace 规范的 header 来记录调用链信息，因此您需要对业务代码做少量改造，传递这些 header，以便网格可以正确关联入站和出站的 span，形成完整调用链路。这些 header 包括：

- x-request-id
- x-b3-traceid
- x-b3-spanid
- x-b3-parentspanid
- x-b3-sampled
- x-b3-flags
- x-ot-span-context

更多关于 Envoy-based tracing 的信息，请参见 Istio Distributed Tracing FAQ 。

访问日志 Access Log

最近更新时间：2024-03-22 14:07:42

您可以配置服务网格数据面访问日志 Access Log 输出（容器标准输出）的开启范围、输出格式、以及开启 Access Log 日志自动采集对接到云日志服务产品 CLS 的日志集-日志主题。您可以在创建网格时配置访问日志，网格创建后您也可以在基本信息页面修改访问日志配置。

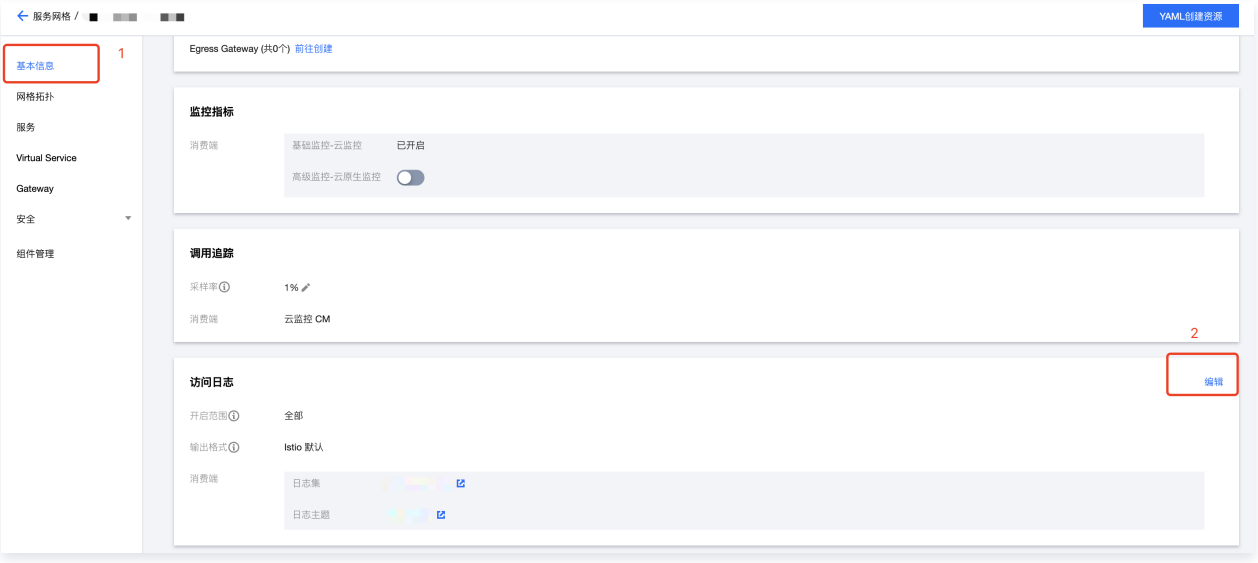
访问日志配置

当前支持的访问日志配置如下表：

配置项	描述
开启范围	配置开启访问日志输出的数据面（边缘代理网关 和 istio-proxy sidecar），可以开启指定边缘代理网关、指定 namespace 下所有数据面、或网格所有数据面的访问日志到容器标准输出。
输出格式	配置访问日志输出的字段和格式模板。默认格式输出的字段为 Istio 默认输出的字段，增强格式在默认格式基础上增加了 Trace ID 输出，自定义则是可以自行定义日志输出格式（请参见 envoy日志格式 ）。
消费端	配置将数据面容器标准输出的访问日志采集到日志服务 CLS。需要选择存储访问日志的 CLS 日志集与日志主题，可以选择自动创建日志集/主题，或关联已有的日志集/主题。自动创建的日志集命名规则为 `{mesh ID}`，自动创建的日志主题带有 TCM 标识，命名规则为 `{mesh ID}-accesslog`。开启访问日志采集到 CLS 提交后，会开启网格管理集群的日志采集功能，在网格管理的集群中部署日志采集组件 tke-log-agent (DaemonSet)，并配置 TCM 访问日志的采集规则与索引。该功能是基于容器服务的 日志采集功能，请确保已开通 日志服务 CLS，并且容器服务的服务角色 `TKE_QCSRole` 已关联日志服务运维管理的预设策略 `QcloudAccessForTKERoleInOpsManagement`，更多说明请参见 容器服务角色权限说明。

- 创建网格时配置 Access Log：

- 网格创建完成后配置 Access Log：



访问日志查看

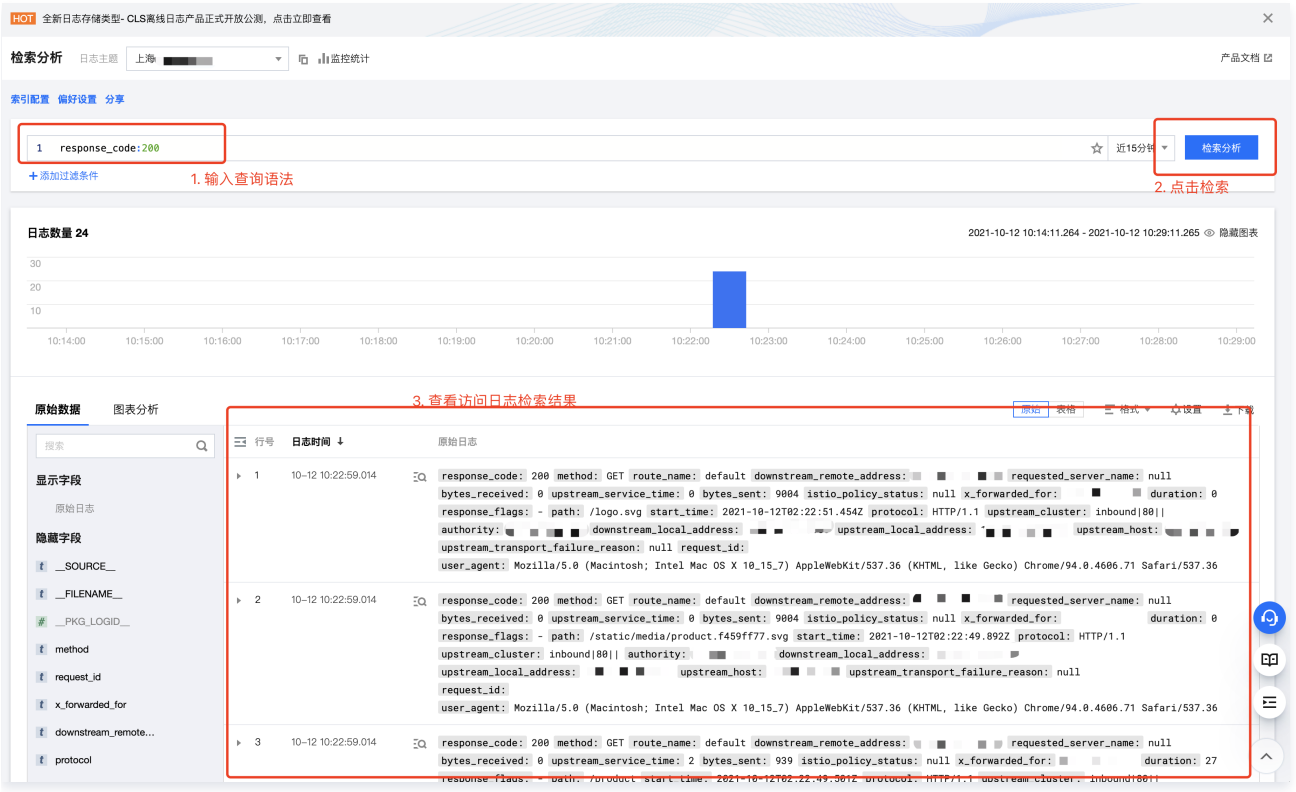
通过容器标准输出查看

TCM 的数据面访问日志 Access Log 是输出到容器标准输出，您可以通过您的 Kubernetes 集群 API Server 查看 istio-proxy 容器标准输出的访问日志：

```
kubectl -n {命名空间} logs {Pod 名称} -c istio-proxy  --tail 5
```

通过日志服务 CLS 日志检索查看

如您开启了访问日志的消费端配置，将 TCM 数据面访问日志 Access Log 采集到了日志服务 CLS，则您可以在 CLS 控制台检索分析处选择对应日志主题查看 TCM 数据面访问日志。CLS 日志检索语法，请参见 CLS 日志检索语法与规则 。



安全
Authentication 认证策略配置

最近更新时间：2024-03-22 14:07:42

认证策略包含 PeerAuthentication 和 RequestAuthentication。其中，PeerAuthentication 策略用于配置服务通信的 mTLS 模式，RequestAuthentication 策略用于配置服务的请求身份验证方法。

PeerAuthentication 配置字段说明

以下是 PeerAuthentication 重要字段说明：

字段名称	字段类型	字段说明
`metadata.name`	`string`	PeerAuthentication 名称。
`metadata.namespace`	`string`	PeerAuthentication 命名空间。
`spec.selector`	`map<string, string>`	PeerAuthentication 使用填写的标签键值对，配合填写的 namespace，匹配配置下发的 Workload 范围： • namespace 填写 istio-system，且 selector 字段不填写时，该策略生效范围为整个网格。 • namespace 填写非 istio-system 的 namespace，且 selector 字段不填写时，策略生效范围为填写的 namespace。 • namespace 填写非 istio-system 的 namespace，且 selector 字段填写了有效键值对时，策略的生效范围为在所填 namespace 下根据 selector 匹配到的 Workload。
`spec.mtls.mode`	-	配置 mTLS 的模式，支持：`UNSET`
`spec.portLevelMtls`	`map<uint32, mTLS mode>`	设置端口级别的 mTLS 模式。

说明：
mTLS 模式配置，不同选择范围的生效效力为：端口 > 服务/Workload > namespace > 网格。

使用 PeerAuthentication 配置网格内服务通信 mTLS 模式

服务网格默认网格内 mTLS 模式为 PERMISSIVE，即服务间的通信既可以使用 mTLS 加密，也可以使用 plaintext 明文连接。
为测试 mTLS 模式配置的效果，您可以首先对您网格内的服务发起明文请求，测试明文请求的连通性。以下是登录网格内 istio-proxy 容器对另外的服务发起明文请求的示例：

1. 在网格管理的 TKE 集群控制台，登录 istio-proxy 容器。

2. 输入命令 `curl http://product.base.svc.cluster.local:7000/product` 明文访问命名空间 base 下的 product 服务。

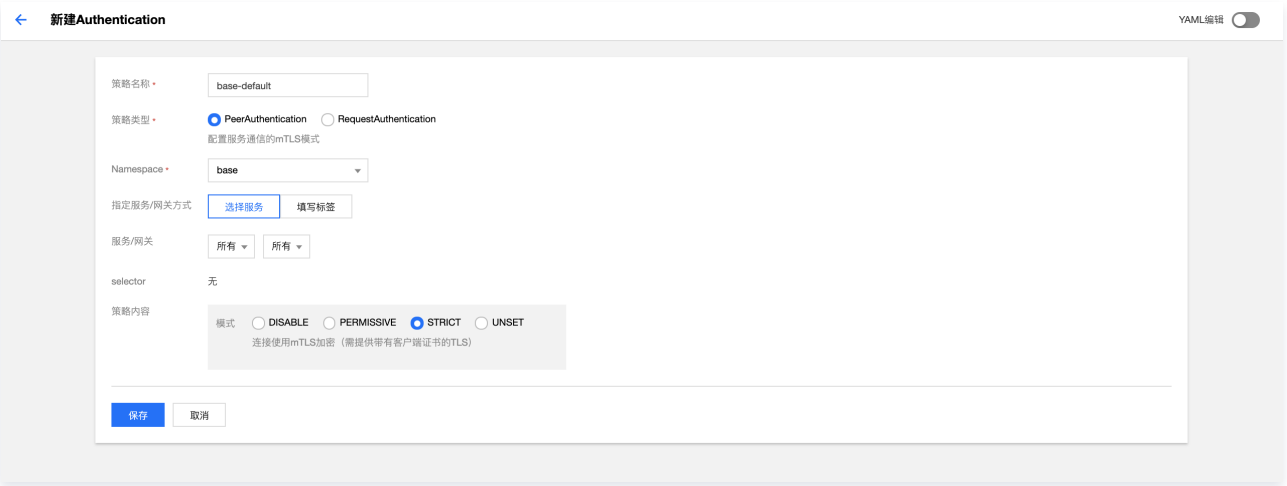
3. 查看明文访问结果，正确返回了 Product 信息，明文访问成功。

下面我们将会配置 base namespace 的 mTLS 模式为 STRICT，并验证配置是否生效。

YAML 配置示例

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

控制台配置示例



配置完成后，重新以明文的方式访问 base 命名空间下的 product 服务，提示访问失败，mTLS STRICT 模式配置生效。

RequestAuthentication 配置字段说明

以下是 RequestAuthentication 重要配置字段说明：

字段名称	字段类型	字段说明
`metadata.name`	`string`	RequestAuthentication 名称。

metadata.namespace	string	RequestAuthentication 命名空间。
spec.selector	map<string, string>	RequestAuthentication 使用填写的标签键值对，配合填写的namespace，匹配配置下发的 Workload 范围，namespace 填写 istio-system，且 selector 字段不填写时，该策略生效范围为整个网格；namespace 填写非 istio-system 的 namespace，且 selector 字段不填写时，策略生效范围为填写的 namespace；namespace 填写非 istio-system 的 namespace，且 selector 字段填写了有效键值对时，策略的生效范围为在所填 namespace 下根据 selector 匹配到的Workload。
spec.jwtRules.issuer	string	配置 JWT token 的 issuer，详情参见 iss claim
spec.jwtRules.audiences	string[]	配置允许访问的 JWT audience 列表。当 audience 列表为空时，使用 service name 则被允许访问。
spec.jwtRules.jwksUri	string	配置验证 JWT 签名的公钥 URL，详情参见 OpenID Discovery。同时配置 jwksUri 和 jwks 字段时，jwksUri 将被忽略。
spec.jwtRules.jwks	string	验证 JWT 签名的 JSON Web Key Set 公钥。同时配置 jwksUri 和 jwks 字段时，jwksUri 将被忽略。
spec.jwtRules.fromHeaders	map<string,string>[]	配置 JWT 从 header 中的提取位置列表。
spec.jwtRules.fromParams	string[]	配置 JWT 从 header 中提取的 parameters，例如从 parameter mytoken（/path?my_token=）中提取。
spec.jwtRules.outputPayloadToHeader	string	配置成功验证的 JWT payload 输出的 header 名称，转发的数据为 base64_encoded(jwt_payload_in_JSON)。未填写时默认不会输出 JWT payload。
spec.jwtRules.forwardOriginalToken	bool	配置是否将原始 JWT 转发至 upstream。默认值为 false。

使用 RequestAuthentication 配置请求 JWT 认证

为验证请求 JWT 认证配置的效果，您首先需要部署一个测试程序 httpbin.foo，并配置通过 Ingress Gateway 暴露此服务到公网：

- 创建 foo namespace，开启 sidecar 自动注入，部署httpbin服务到foo namespace：

```
apiVersion: v1
kind: Namespace
metadata:
  name: foo
  labels:
    istio.io/rev: 1-6-9 # 开启 namespace 的 sidecar 自动注入（istio 版本 1.6.9）
spec:
  finalizers:
    - kubernetes
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: httpbin
  namespace: foo
---
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  namespace: foo
  labels:
    app: httpbin
    service: httpbin
spec:
  ports:
```

```
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
  namespace: foo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
      version: v1
  template:
    metadata:
      labels:
        app: httpbin
        version: v1
    spec:
      serviceAccountName: httpbin
      containers:
      - image: docker.io/kennethreitz/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        ports:
        - containerPort: 80
```

- 配置通过 Ingress Gateway 暴露 httpbin 服务至公网访问：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
  namespace: foo
spec:
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
  namespace: foo
spec:
  hosts:
  - "*"
  gateways:
  - httpbin-gateway
  http:
  - route:
```

```
    - destination:
        port:
          number: 8000
        host: httpbin.foo.svc.cluster.local
```

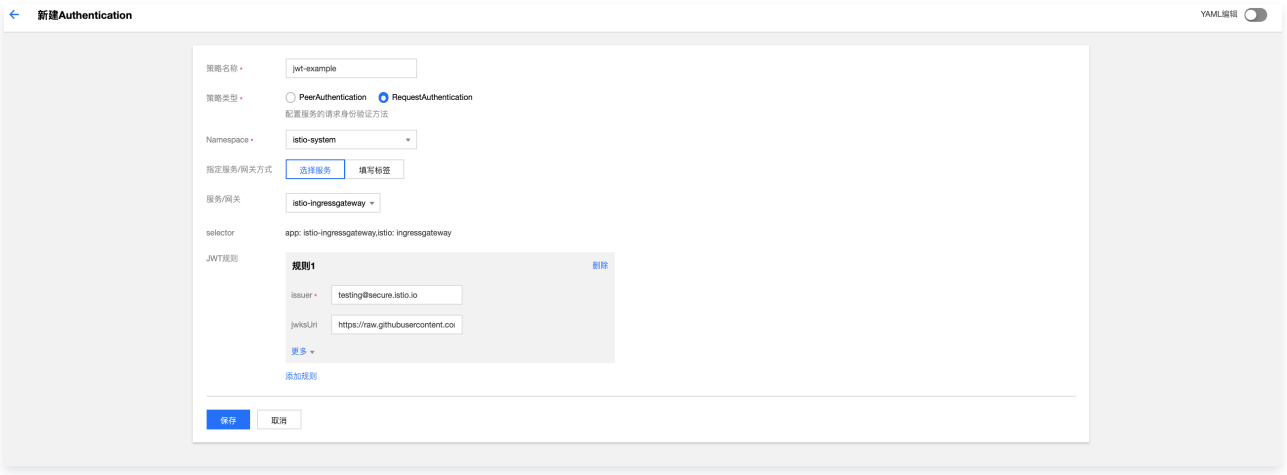
- 通过 curl 语句 `curl "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"` 测试服务的连通性，注意您需要将代码中的 `$INGRESS_IP` 替换为您的边缘代理网关 IP 地址，正常情况下会返回 `200` 返回码。

下面将会为边缘代理网关配置 JWT 认证规则，放通带有符合条件的 JWT 令牌的请求。

YAML 配置示例

```
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
      app: istio-ingressgateway
  jwtRules:
  - issuer: "testing@secure.istio.io"
    jwksUri: "https://raw.githubusercontent.com/istio/istio/release-
1.9/security/tools/jwt/samples/jwks.json"
```

控制台配置示例



配置完成后，我们来验证配置的 JWT 验证规则是否生效。

- 通过以下代码，携带一个非法的 JWT 令牌发起访问，注意您需要将代码中的 `$INGRESS_IP` 替换为您的边缘代理网关 IP 地址。边缘代理网关不会放通携带非法 JWT 令牌的请求，因此会返回 `401` 返回码。

```
curl --header "Authorization: Bearer deadbeef" "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"
```

- 通过以下代码，携带一个合法的 JWT 令牌发起访问，注意您需要将代码中的 `$INGRESS_IP` 替换为您的边缘代理网关 IP 地址。边缘代理网关会放通携带合法 JWT 令牌的请求，因此会返回 `200` 返回码。

```
TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-1.9/security/tools/jwt/samples/demo.jwt
-s)
curl --header "Authorization: Bearer $TOKEN" "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"
```

通过验证，您可以发现您为边缘代理网关配置的请求 JWT 认证规则已经生效。但此时仅仅配置了 JWT 认证规则，Ingress Gateway 仍会放通未携带 JWT 令牌的请求。限制未携带 JWT 令牌的请求需要配置 AuthorizationPolicy。应用以下 YAML 文件至服务网格即可限制 Ingress Gateway 拒绝未携带 JWT 令牌的请求：

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: frontend-ingress
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
      istio: ingressgateway
  rules:
    - from:
        - source:
            notRequestPrincipals:
              - '*'
  action: DENY
```

再次用未携带 JWT 令牌的方式发起访问 `curl "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"`，发现访问失败，返回 `403` 返回码，AuthorizationPolicy 策略生效。

Authorization 授权策略配置

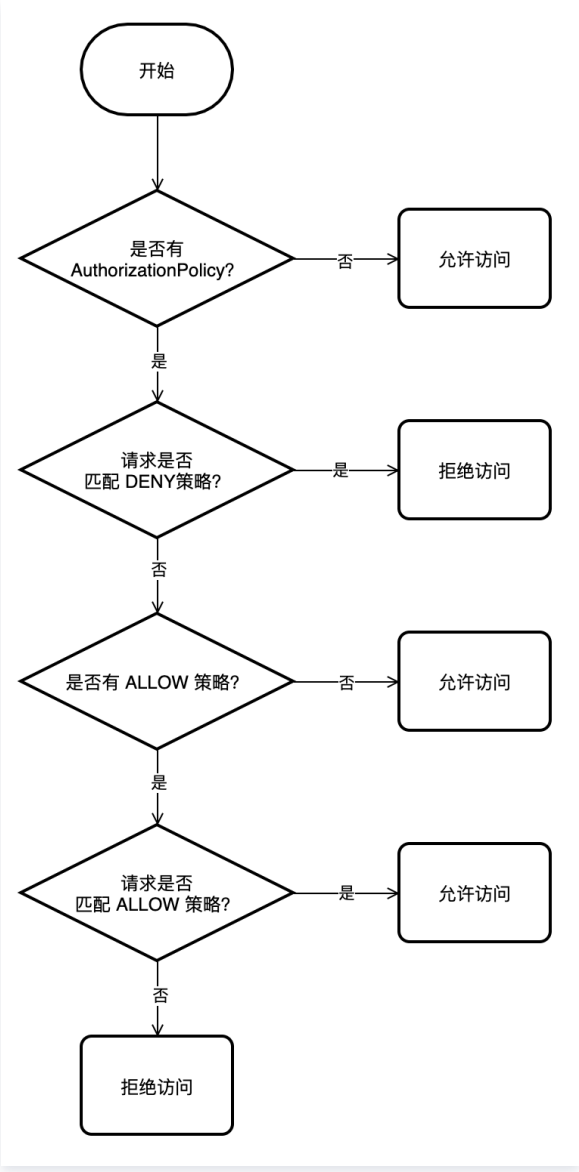
最近更新时间：2024-03-22 14:07:42

授权策略用于配置网格、namespace、服务/Workload 范围的访问管理规则。您可以通过 AuthorizationPolicy CRD 配置授权规则。AuthorizationPolicy 主要包含以下部分：

- selector：指定策略的生效范围。
- action：指定该策略是 `ALLOW` 策略还是 `DENY` 策略。
- rules：授权规则主体，由 from，to，where 3 部分构成。
  - from：指定请求的来源特征。
  - to：指定请求的操作特征。
  - when：指定授权规则的生效条件。

当有 AuthorizationPolicy 的 `ALLOW` 和 `DENY` 策略应用于同一范围时， `DNEY` 策略的优先级高于 `ALLOW` 策略，生效的规则如下：

1. 如请求匹配任何一条 `DENY` 策略，则拒绝该请求的访问。
2. 如该范围没有任何 `ALLOW` 策略，则允许该请求的访问。
3. 如当前该范围存在 `ALLOW` 策略，且请求匹配到了任何一条 `ALLOW` 策略，则允许该请求的访问。
4. 拒绝该请求的访问。



以下是两种特殊 AuthorizationPolicy 示例：

- default namespace 的服务允许所有请求访问：

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
name: allow-all
namespace: default
spec:
action: ALLOW
rules:
- {} # 规则可以匹配任何请求
```

- default namespace 的服务拒绝所有请求访问：

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
name: deny-all
namespace: default
spec:
{} # action 字段没有填写时默认是 ALLOW，此时请求无法匹配任何规则
```

AuthorizationPolicy 重要字段说明

以下是 AuthorizationPolicy 重要字段说明：

字段名称	字段类型	字段说明
`metadata.name`	`string`	AuthorizationPolicy 名称。
`metadata.namespace`	`string`	AuthorizationPolicy 命名空间。
`spec.selector`	`map<string, string>`	AuthorizationPolicy 使用填写的标签键值对，配合填写的 namespace，匹配配置下发的 Workload 范围： • namespace 填写 istio-system，且 selector 字段不填写时，该策略生效范围为整个网格。 • namespace 填写非 istio-system 的 namespace，且 selector 字段不填写时，策略生效范围为填写的 namespace。 • namespace 填写非 istio-system 的 namespace，且 selector 字段填写了有效键值对时，策略的生效范围为在所填 namespace 下根据 selector 匹配到的 Workload。
`spec.action`	-	指定该策略是 `ALLOW` 策略还是 `DENY` 策略。
`spec.rules.from.source.principals`	`string[]`	源对等身份列表（即 service account），匹配 `source.principal` 字段，要求启用 mTLS，未填写时则允许任何 principal。
`spec.rules.from.source.requestPrincipals`	`string[]`	请求身份列表（即 iss/sub claim），匹配 `request.auth.principal` 字段，未填写时则允许任何 requestPrincipals。
`spec.rules.from.source.namespaces`	`string[]`	请求源的 namespace 列表，匹配 `source.namespace` 字段，要求启用 mTLS，未填写时允许来自任何 namespace 的请求。
`spec.rules.from.source.ipBlocks`	`string[]`	IP block 列表，匹配 `source.ip` 字段，支持单 IP 写法（如 `1.2.3.4`）或 CIDR 写法（如 `1.2.3.4/24`），未填写时允许任何源 IP 的访问。
`spec.rules.to.operation.hosts`	`string[]`	请求的域名列表，匹配 `request.host` 字段，未填写时允许任何域名，仅支持在 HTTP 协议请求中使用。
`spec.rules.to.operation.ports`	`string[]`	请求的端口列表，匹配 `destination.port` 字段，未填写时允许任何端口。
`spec.rules.to.operation.methods`	`string[]`	请求的方法列表，匹配 `request.method` 字段，使用 gRPC 协议时该值始终应为 `POST`。未填写时允许任何方法，仅支持在 HTTP 协议请求中使用。

spec.rules.to.operation.paths	string[]	请求的路径，匹配 request.url_path 字段，未填写时允许任何路径，仅支持在 HTTP 协议请求中使用。
spec.rules.when.condition.key	string	Istio 支持的条件字段名称，详见 Authorization Policy Conditions
spec.rules.when.condition.values	string[]	填写对应条件的值列表。

使用 AuthorizationPolicy 配置 namespace 的访问权限

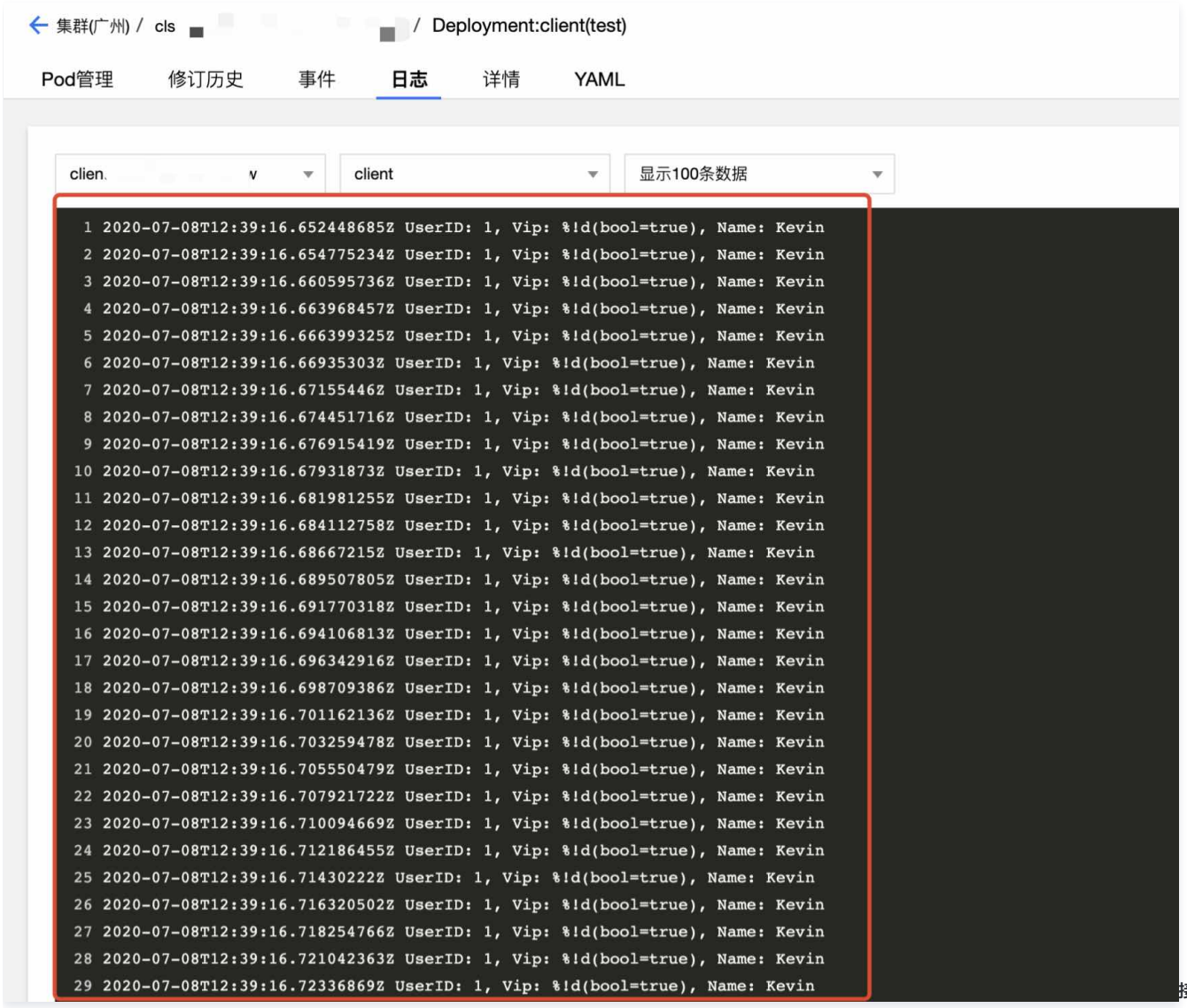
为查看配置的 AuthorizationPolicy 策略效果，我们首先部署一套测试程序到网格管理的集群，部署完成后位于 test namespace 的 client 服务会自动发起对 base namespace user 服务的访问：

```
apiVersion: v1
kind: Namespace
metadata:
  name: test
  labels:
    istio.io/rev: 1-6-9 # sidecar 自动注入（istio 1.6.9）
spec:
  finalizers:
    - kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: client
  namespace: test
  labels:
    app: client
spec:
  replicas: 10
  selector:
    matchLabels:
      app: client
  template:
    metadata:
      labels:
        app: client
    spec:
      containers:
        - name: client
          image: ccr.ccs.tencentyun.com/zhulei/testclient:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: REGION
              value: "guangzhou-zoneA"
          ports:
            - containerPort: 7000
              protocol: TCP
---

apiVersion: v1
kind: Service
metadata:
  name: client
```

```
  namespace: test
  labels:
    app: client
spec:
  ports:
    - name: http
      port: 7000
      protocol: TCP
  selector:
    app: client
  type: ClusterIP
---
apiVersion: v1
kind: Namespace
metadata:
  name: base
  labels:
    istio.io/rev: 1-6-9
spec:
  finalizers:
    - kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user
  namespace: base
  labels:
    app: user
spec:
  replicas: 1
  selector:
    matchLabels:
      app: user
  template:
    metadata:
      labels:
        app: user
    spec:
      containers:
        - name: user
          image: ccr.ccs.tencentyun.com/zhulei/testuser:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: REGION
              value: "guangzhou-zoneB"
          ports:
            - containerPort: 7000
---

apiVersion: v1
kind: Service
metadata:
  name: user
  namespace: base
  labels:
    app: user
spec:
```

```
  ports:
    - port: 7000
      name: http
  selector:
    app: user
```

查看 client 容器的日志，会发现访问成功，正确返回了 user 信息：



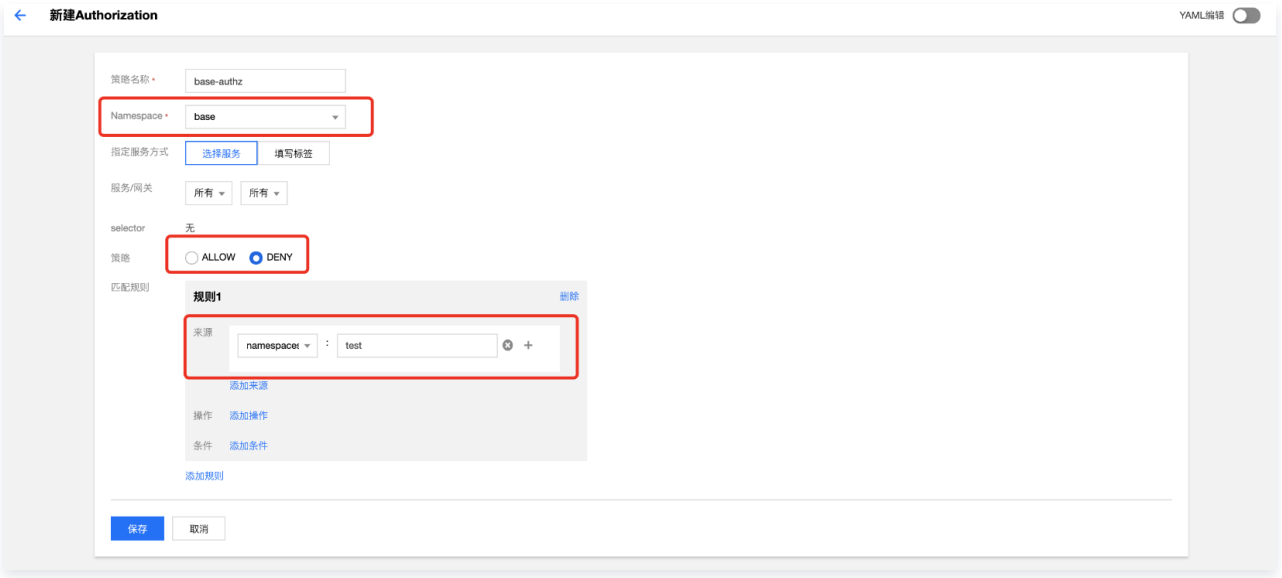
接下来将配置 Authorization 策略，不允许 base namespace 的服务被 test namespace 的服务访问（需要开启 mTLS）。

YAML 配置示例

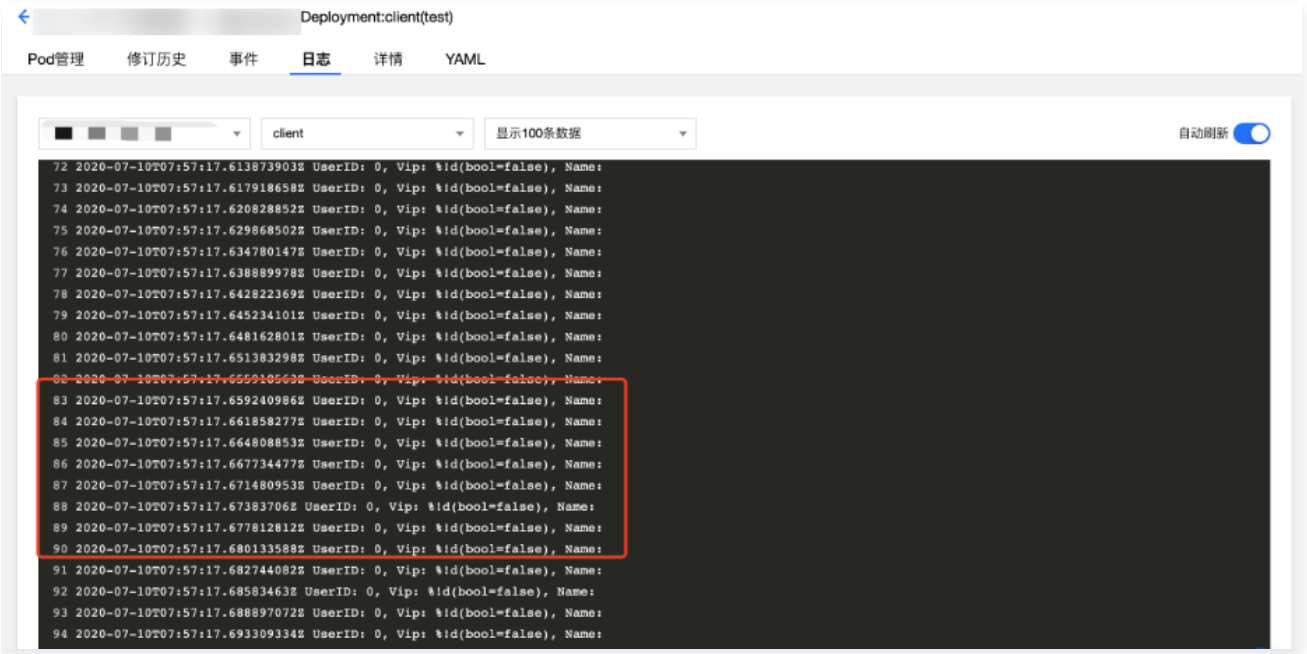
```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: base-authz
  namespace: base
spec:
  action: DENY
  rules:
    - from:
        - source:
            namespaces:
```

```
        - test
```

控制台配置示例



配置完成后再次查看 client 的容器日志，发现所有访问均失败，没有返回 user 信息，AuthorizationPolicy 生效。



使用 AuthorizationPolicy 配置 Ingress Gateway 的 IP 黑白名单

您可以使用 AuthorizationPolicy 为边缘代理网关 Ingress Gateway 配置 IP 黑/白名单。

为验证黑白名单的配置效果，您首先需要部署一个测试程序 `httpbin.foo` ，并配置通过 Ingress Gateway 暴露此服务到公网：

- 创建 foo namespace，开启 sidecar 自动注入，部署 httpbin 服务到 foo namespace：

```
apiVersion: v1
kind: Namespace
metadata:
  name: foo
  labels:
```

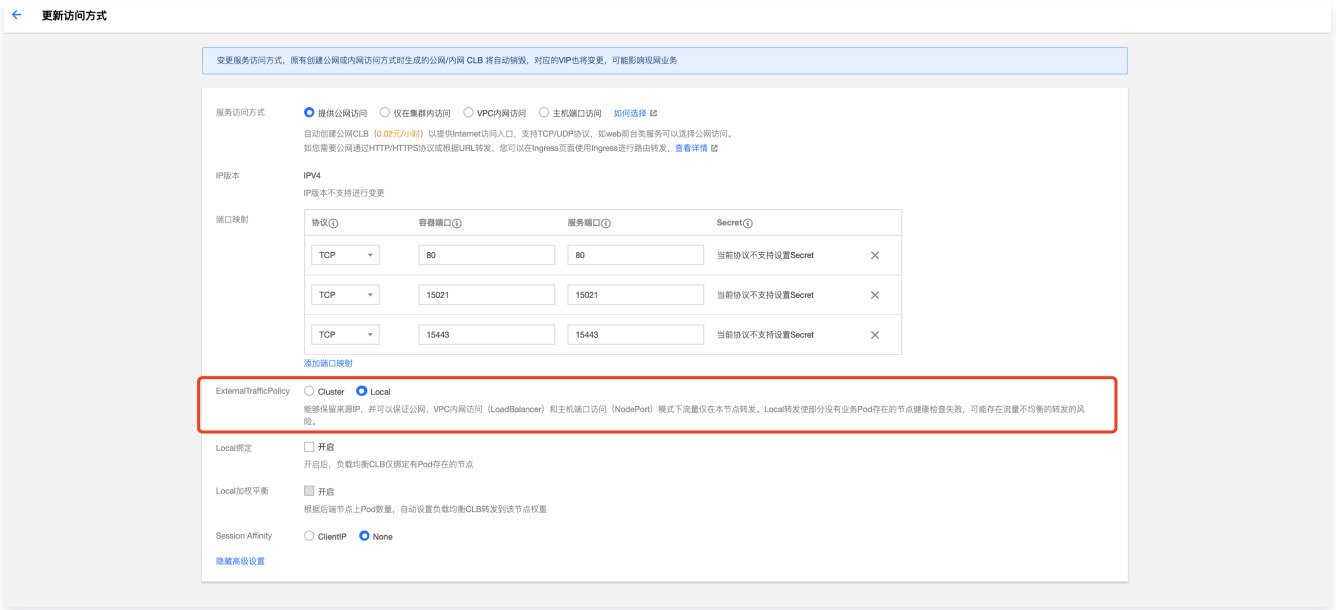
```
    istio.io/rev: 1-6-9 # 开启 namespace 的 sidecar 自动注入（istio 版本 1.6.9）
spec:
  finalizers:
    - kubernetes
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: httpbin
  namespace: foo
---
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  namespace: foo
  labels:
    app: httpbin
    service: httpbin
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
  namespace: foo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
      version: v1
  template:
    metadata:
      labels:
        app: httpbin
        version: v1
    spec:
      serviceAccountName: httpbin
      containers:
      - image: docker.io/kennethreitz/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        ports:
        - containerPort: 80
```

- 配置通过 Ingress Gateway 暴露 httpbin 服务至公网访问：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
  namespace: foo
spec:
  selector:
    app: istio-ingressgateway
```

```
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
  namespace: foo
spec:
  hosts:
  - "*"
  gateways:
  - httpbin-gateway
  http:
  - route:
    - destination:
        port:
          number: 8000
        host: httpbin.foo.svc.cluster.local
```

- 通过 curl 语句 `curl "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"` 测试服务的连通性，注意您需要将代码中的 `$INGRESS_IP` 替换为您的边缘代理网关 IP 地址，正常情况下会返回 `200` 返回码。
- 为使 Ingress Gateway 能正确获取真实客户端源 IP，我们需要修改 Ingress Gateway Service 的 ExternalTrafficPolicy 为 Local，保证流量仅在本节点转发不做 SNAT。



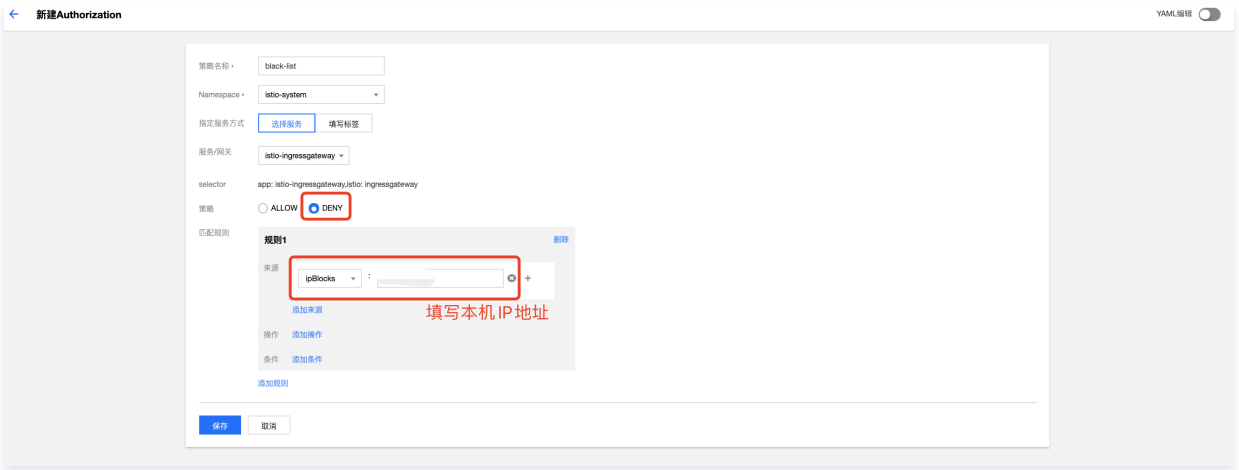
下面将会使用 AuthorizationPolicy 把本机的 IP 地址列入 Ingress Gateway 的黑名单，并验证黑名单是否生效。

YAML 配置示例

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: black-list
  namespace: istio-system
```

```
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
      istio: ingressgateway
  rules:
    - from:
        - source:
            ipBlocks:
              - $您的本机 IP 地址
  action: DENY
```

控制台配置示例



配置完成后再次通过 curl 语句 `curl "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"` 测试服务的连通性，注意您需要将代码中的 `$INGRESS_IP` 替换为您的边缘代理网关 IP 地址，此时访问失败，返回 `403` 返回码，黑名单策略生效。

访问管理

概述

最近更新时间：2024-03-22 14:07:42

服务网格的权限管理包含2个部分：访问管理（CAM） 权限与 容器服务（TKE）RBAC 权限。
默认情况下，子账号不具备 CAM 的权限，非集群创建者的子账号不具备相关集群 RBAC 的权限。您需要创建关联 CAM 策略和 TKE RBAC 授权策略来允许子账号访问或正常使用他们所需要的服务网格资源。
CAM 权限策略的编辑和授予是由 CAM 管理员（通常是主账号或拥有 CAM 权限的子账号）完成，更多关于 CAM 策略的基本信息，请参见 CAM 策略 。TKE 集群的 RBAC 权限策略的编辑和授予通常是由相应集群管理员（通常是主账号或集群创建账号）完成，授权方式参见 TKE RBAC 授权 。

> **说明：**
> 若您不需要对子账户进行 TCM 相关资源的访问管理，您可以跳过此章节，跳过这些部分不会影响您对文档中其余部分的理解和使用。

TCM 基于 CAM 的权限控制

当前 TCM 支持基于 CAM 的资源级的权限控制，即能够允许指定**子账号**对指定**资源**的指定**操作**。子账号默认没有 TCM 相关 CAM 权限，您需要将策略关联至子账号完成授权。
当前 TCM 基于 CAM 的资源级权限控制颗粒度可达到网格实例级别，即您可以控制指定子账号对指定网格能够执行指定的操作。

TCM 相关产品 TKE 的 RBAC 权限管理

使用 TCM 过程中，会涉及到对 TCM 管理的 TKE 集群内 Kubernetes 资源的读写操作，这些操作需要有足够的 TKE RBAC 权限。默认非集群创建者的子账号没有该集群的 RBAC 权限，需要集群管理员授予该子账号对应集群的 RBAC 权限，子账号才能正常使用 TCM。
在所选集群创建/删除/更新服务网格、添加/解关联服务发现集群、在所选集群创建/删除 Ingress Gateway 网关均需要相应集群的管理员（tke:admin）权限。对网格内 Istio 资源（Gateway，VirtualService，DestinationRule，ServiceEntry 等）的操作不需要集群的 RBAC 权限。
更多 TKE Kubernetes 对象级权限控制信息，请参见 TKE Kubernetes 对象级权限控制 。TKE RBAC 授权方式，请参见 授权模式对比 。

CAM 服务角色授权

最近更新时间：2024-03-22 14:07:42

在使用腾讯云服务网格（Tencent Cloud Mesh，TCM）的过程中，涉及到服务网格相关云资源的使用，为了您能正常使用 TCM 的功能，您需要对 TCM 的服务角色 `TCM_QCSRole` 进行授权，授权后，TCM 服务才能使用相关云资源。
需要服务授权的场景主要包含 首次登录服务网格控制台 以及 首次使用 TCM 一键体验功能 两个场景，分别对应 `QcloudAccessForTCMRole` 和 `QcloudAccessForTCMRoleInSampleDeployment` 两个预设策略。

首次登录服务网格控制台

授权场景

当您已注册并登录腾讯云账号后，首次登录 服务网格控制台 时，需前往**访问管理**页面对当前账号授予腾讯云服务网格操作容器服务（TKE）、SSL证书（SSL）、日志服务（CLS）等云资源的权限。该权限授予通过关联预设策略 `QcloudAccessForTCMRole` 至 TCM 服务角色 `TCM_QCSRole` 完成。如您之前未创建过 TCM 服务角色，该授权流程还会涉及 TCM 服务角色的创建。

授权步骤

1. 首次登录 服务网格控制台，自动弹出**服务授权**窗口。

2. 单击**前往访问管理**，进入 CAM 控制台服务授权页面。
3. 单击**同意授权**，完成身份验证后即可成功授权。

权限内容

容器服务（TKE）相关

权限	描述	资源
DescribeClusterSecurity	查询集群密钥	所有资源 *

SSL 证书（SSL）相关

权限	描述	资源
DescribeCertificateDetail	获取证书详情	所有资源 *

日志服务（CLS）相关

权限	描述	资源
getLogset	获取日志集详情	所有资源 *
getTopic	获取日志主题详情	所有资源 *
createLogset	创建日志集	所有资源 *
createTopic	创建日志主题	所有资源 *
modifyIndex	修改索引	所有资源 *
listLogset	获取日志集列表	所有资源 *
listTopic	获取日志主题列表	所有资源 *

首次使用 TCM 一键体验功能

授权场景

首次使用 TCM 一键体验功能时，您需要前往**访问管理**对当前账号授予腾讯云服务网格操作私有网络（VPC）、云联网（CCN）、容器服务（TKE）、云服务器（CVM）等云资源以及购买云服务器（CVM）的财务权限。该权限授予通过关联预设策略 `QcloudAccessForTCMRoleInSampleDeployment` 至 TCM 服务角色 `TCM_QCSRole` 完成。如您之前未创建过 TCM 服务角色，该授权流程还会涉及 TCM 服务角色的创建。

授权步骤

1. 登录 服务网格控制台，鼠标移动至**一键体验**按钮，单击**服务授权**弹出授权提示窗口。

2. 单击**前往访问管理**，进入 CAM 控制台服务授权页面。

3. 单击**同意授权**，完成身份验证后即可成功授权。

权限内容

私有网络（VPC）相关

权限	描述	资源
CreateSubnet	创建子网	所有资源 *
CreateVpc	创建私有网络	所有资源 *
DeleteVpc	删除私有网络	所有资源 *
DescribeVpcEx	查询私有网络列表	所有资源 *
DescribeSubnetEx	查看子网列表	所有资源 *

云联网（CCN）相关

权限	描述	资源
AttachCcnInstances	关联云联网实例	所有资源 *
CreateCcn	创建云联网	所有资源 *
DeleteCcn	删除云联网	所有资源 *
DescribeCcnAttachedInstances	查询云联网关联实例列表	所有资源 *
DescribeCcns	查询云联网列表	所有资源 *

容器服务（TKE）相关

权限	描述	资源
CreateCluster	创建集群	所有资源 *
CreateClusterEndpointVip	开启集群访问端口（托管集群外网）	所有资源 *
DeleteCluster	删除集群	所有资源 *
DeleteClusterEndpoint	删除集群访问端口	所有资源 *
DeleteClusterEndpointVip	删除集群访问端口（托管集群外网）	所有资源 *
DescribeClusterEndpointVipStatus	查询集群开启访问端口状态（托管集群外网）	所有资源 *

DescribeClusters	获取集群列表	所有资源 *

云服务器（CVM）相关

权限	描述	资源
CreateSecurityGroup	创建安全组	所有资源 *
DeleteSecurityGroup	删除安全组	所有资源 *
DescribeImages	查询镜像	所有资源 *
DescribeInstances	查询云主机	所有资源 *
DescribeSecurityGroups	查询安全组	所有资源 *
ModifySecurityGroupPolicys	修改安全组规则	所有资源 *
RunInstances	创建云主机	所有资源 *

财务权限

权限	描述	资源
finance:*	购买云服务器的财务权限	云服务器 qcs::cvm:::*

CAM 预设策略授权

最近更新时间：2024-03-22 14:07:42

您可以把 CAM 中的 TCM 相关预设策略关联至子账号，快速完成 TCM 的 CAM 授权。

TCM 相关预设策略

您可以使用以下预设策略为您的子账号授予相关权限：

策略	描述
QcloudTCMFullAccess	服务网格（TCM）全读写访问权限（创建、删除等全部操作）
QcloudTCMReadOnlyAccess	服务网格（TCM）只读访问权限（可以查看TCM下所有资源，但无法创建、更新或删除资源）

服务网格全读写预设策略

策略名：QcloudTCMFullAccess，策略内容：

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "tcm:*"
            ],
            "resource": "*",
            "effect": "allow"
        }
    ]
}
```

服务网格只读预设策略

策略名：QcloudTCMReadOnlyAccess，策略内容：

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "tcm:List*",
                "tcm:Describe*",
                "tcm:ForwardRequestRead"
            ],
            "resource": "*",
            "effect": "allow"
        }
    ]
}
```

TCM 相关产品的 CAM 权限

使用 TCM 还涉及到关联的 VPC、CCN、CLB、TKE 等产品的 CAM 权限，您可以参考相应产品的 CAM 授权文档，为子账号授予合适的权限：

TCM 相关产品	授权指南文档
私有网络（VPC）	VPC 访问管理概述
负载均衡（CLB）	CLB 访问管理概述

容器服务（TKE）	TKE 权限管理概述

子账号关联预设策略

您可在创建子账号的“设置用户权限”步骤中，通过 直接关联 或 随组关联 方式，为该子账户关联预设策略。

直接关联

您可以直接为子账号关联策略以获取策略包含的权限。

1. 登录访问管理控制台，选择左侧导航栏中的**用户** > **用户列表**。
2. 在**用户列表**管理页面，选择需要设置权限的子账号所在行右侧的**授权**。
3. 在弹出的“关联策略”窗口中，勾选需授权的策略。
4. 单击**确定**即可。

随组关联

您可以将子账号添加至用户组，该子账号将自动获取该用户组所关联策略的权限。如需解除随组关联策略，仅需将子账号移出相应用户组即可。

1. 登录访问管理控制台，选择左侧导航栏中的**用户** > **用户列表**。
2. 在**用户列表**管理页面，选择需要设置权限的子账号所在行右侧的**更多操作** > **添加到组**。
3. 在弹出的“添加到组”窗口中，勾选需加入的用户组。
4. 单击**确定**即可。

登录子账号验证

登录 腾讯云服务网格控制台，验证可使用所授权策略对应功能，则表示子账号授权成功。

CAM 自定义策略授权

最近更新时间：2025-06-10 15:11:52

如您有自定义的权限管理诉求，您可以通过创建 CAM 自定义策略，并关联至子账号实现自定义授权。您可参考文本并根据实际业务诉求进行配置。

CAM 元素参考

CAM 自定义策略核心元素包括：操作（action）、资源（resource）、生效条件（condition）以及效力（effect）。

1. 操作（action）

描述允许或拒绝的操作。操作可以是 API（以 name 前缀描述）或者功能集（一组特定的 API，以 actionName 前缀描述）。该元素是必填项。您可以查看 TCM 接入 CAM 的 API。

2. 资源（resource）

描述授权的具体数据。资源是用六段式描述。您可以查看 TCM 资源描述。

3. 生效条件（condition）

描述策略生效的约束条件。条件包括操作符、操作键和操作值组成。条件值可包括时间、IP 地址等信息。

4. 效力（effect）

描述声明产生的结果是“允许”还是“显式拒绝”。包括 allow（允许）和 deny（显式拒绝）两种情况。该元素是必填项。

5. 自定义策略样例

该策略为：允许对广州的两个 mesh 实例：mesh-abcd1234 和 mesh-1234abcd 做获取详情操作。

```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
            "resource": [
                "qcs::tcm:gz:uin/1234567:mesh/mesh-abcd1234",
                "qcs::tcm:gz:uin/1234567:mesh/mesh-1234abcd"
            ],
            "action": [
                "name/tcm:DescribeMesh"
            ]
        }
    ]
}
```

更多关于 CAM 自定义策略语法逻辑，请参见 CAM 语法逻辑。

CAM 中可授权的 TCM 资源

资源	授权策略中的资源描述方法
服务网格	`qcs::tcm:$region:$account:mesh/$meshid`

其中：

- `$region`：描述地域信息，应为某个 region 的 ID，例如 `gz` 为广州。
- `$account`：描述资源拥有者的主账号信息，表示为 `uin/${uin}`，例如 `uin/12345678`，若值为空则表示创建策略的 CAM 用户所属的主账号。
- `$meshid`：描述mesh实例信息，应为某个网格的 ID，或者 `*`。

关于授权策略中的资源描述方式，请参见 资源描述方式。

CAM 中可对 TCM 进行授权的接口

在 CAM 中，可以对 TCM mesh 资源进行以下操作（action）的授权。

Mesh 实例相关

API 操作	API 描述	资源
CreateMesh	创建服务网格	mesh 资源 `qcs::tcm:$region:$account:mesh/*`
DeleteMesh	删除服务网格	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
DescribeMesh	获取指定服务网格	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
ListMeshes	获取服务网格列表	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
ModifyMesh	修改服务网格配置	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
UpgradeMesh	升级服务网格	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`

Istio 资源相关

API 操作	API 描述	资源
ForwardRequestRead	读 Istio 的 CRD 资源	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
ForwardRequestWrite	写 Istio 的 CRD 资源	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`

服务发现相关

API 操作	API 描述	资源
LinkClusterList	关联集群到服务网格实例	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
UnlinkCluster	解除关联集群	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`

边缘代理网关相关

API 操作	API 描述	资源
CreateIngressGateway	创建 IngressGateway	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
DeleteGatewayInstance	删除 IngressGateway	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
DescribeIngressGatewayList	查询 IngressGateway 列表	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`
ModifyIngressGateway	修改 IngressGateway	mesh 资源 `qcs::tcm:$region:$account:mesh/$meshid`

体验环境相关

API 操作	API 描述	资源
CreateTrial	创建服务网格一键体验环境	只对接口进行鉴权 `*`
DeleteTrial	删除服务网格一键体验环境	只对接口进行鉴权 `*`
RetryTrialTask	重试创建服务网格一键体验环境	只对接口进行鉴权 `*`

功能扩展
使用 wasm filter 扩展数据面

最近更新时间：2024-03-22 14:07:42

Wasm 是 WebAssembly 的缩写，可以编写二进制形式的指令加载到 envoy filter chain 中，实现网格数据面能力扩展。这种形式使得 envoy 和扩展组件的解耦，用户不再需要通过修改 envoy 代码、编译特殊的 envoy 版本来实现能力扩展，并且还具备动态加载和安全隔离等优势。
从 Istio 1.6 版本开始，Proxy-Wasm 沙盒 API 取代了 Mixer 作为 Istio 主要的扩展实现方案，用于实现 envoy 和 wasm 虚拟机之间的交互，因此通过 wasm filter 来扩展 envoy 需要使用 Proxy-WASM SDK 。

通常编写 wasm 文件扩展网格数据面能力主要分为以下几步：

1. 编写 wasm filter，可请参见 示例 。
2. 将 wasm filter 注入到 configmap 中，通过 configmap 将 wasm filter 挂载到任意工作负载，避免将 wasm filter 拷贝到多个 node 上。

```
kubectl create cm -n foo example-filter --from-file=example-filter.wasm
```

3. 将 wasm filter 挂载到业务工作负载，可利用 Istio 的 annotation 机制，在创建工作负载的时候自动挂载相应的文件：

```
sidecar.istio.io/userVolume: '[{"name":"wasmfilters-dir","configMap": {"name": "example-filter"}}]'
sidecar.istio.io/userVolumeMount: '[{"mountPath":"/var/local/lib/wasm-filters","name":"wasmfilters-dir"}]'
```

将 annotation 应用到对应的工作负载之上：

```
kubectl patch deployment -n foo frontpage-v1 -p '{"spec":{"template":{"metadata":{"annotations":
{"sidecar.istio.io/userVolume":"[{\"name\":\"wasmfilters-dir\",\"configMap\": {\"name\": \"example-
filter\"}}]","sidecar.istio.io/userVolumeMount":"[{\"mountPath\":\"/var/local/lib/wasm-
filters\",\"name\":\"wasmfilters-dir\"}]"}}}}}'
```

4. 创建 envoyfilter，将 wasm filter 添加到对应工作负载的 envoy filter chain 中，使其生效。

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: frontpage-v1-examplefilter
  namespace: foo
spec:
  configPatches:
  - applyTo: HTTP_FILTER
    match:
      listener:
        filterChain:
          filter:
            name: envoy.http_connection_manager
            subFilter:
              name: envoy.router
    patch:
      operation: INSERT_BEFORE
      value:
        name: envoy.filters.http.wasm
        typed_config:
          '@type': type.googleapis.com/envoy.extensions.filters.http.wasm.v3.Wasm
          config:
            name: example-filter
            root_id: my_root_id
            vm_config:
              code:
                local:
                  filename: /var/local/lib/wasm-filters/example-filter.wasm
```

```
            runtime: envoy.wasm.runtime.v8
            vm_id: example-filter
            allow_precompiled: true
  workloadSelector:
    labels:
      app: frontpage
      version: v1
```

至此，wasm filter 部署完成，另一种 wasm filter 的使用形式是镜像，请参见 制作 wasm filter 镜像，利用 WASME 工具部署，请参见 使用 wasme 部署 wasm filter 。