ALIBABA CLOUD

阿里云

服务网格

流量管理

文档版本：20210204

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.通过ASM实现TCP应用流量迁移 ---- 05
2.使用ASM为网格内gRPC服务实现负载均衡 ---- 14
3.使用ASM实现基于位置的路由请求 ---- 23
4.在ASM上访问外部服务 ---- 39
5.通过入口网关访问网格内gRPC服务 ---- 44
6.通过AHAS对网格实例进行流量控制 ---- 53

1.通过ASM实现TCP应用流量迁移

服务网格ASM的流量管理功能可以实现应用的流量迁移。本文将通过示例介绍如何通过ASM实现TCP应用流量的迁移。

前提条件

- 已开通以下服务：
  - 容器服务
  - 负载均衡
- 已创建至少一个ACK集群。如果没有创建，请参见创建Kubernetes专有版集群和创建Kubernetes托管版集群。
- 已创建一个ASM实例，并已将ACK集群添加到ASM实例中，请参见创建ASM实例和添加集群到ASM实例。

背景信息

本文以Istio官方Task TCP-Traffic-Shifting为例来讲述如何实现在一个TCP服务的两个版本之间进行流量灰度切换。该Task中的服务是一个简单地Echo服务，在v1版本中，该服务在收到的数据在前面加上“one”并返回；在v2版本中，该服务在收到的数据前面加上“two”并返回。

步骤一：部署示例应用

我们需要部署两个Deployment，如下图所示，打开容器服务控制台，选择应用 - 无状态，然后选择您希望部署的目标集群和namespace，然后点击右上方的使用模版创建。

1. 部署TCP- Echo应用的2个版本。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。
   iv. 在**无状态**页面**命名空间**下拉列表中选择命名空间。
   v. 单击右上方的**使用模版创建**。
   vi. 在**使用模板创建**页面，将下面的Yaml模版粘贴到**模版**文本框中，单击**创建**。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: tcp-echo-v1
spec:
 replicas: 1
 selector:
  matchLabels:
   app: tcp-echo
   version: v1
 template:
  metadata:
   labels:
```

```
      app: tcp-echo
      version: v1
    spec:
     containers:
     - name: tcp-echo
       image: docker.io/istio/tcp-echo-server:1.1
       imagePullPolicy: IfNotPresent
       args: [ "9000", "one" ]
       ports:
       - containerPort: 9000
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: tcp-echo-v2
spec:
 replicas: 1
 selector:
  matchLabels:
   app: tcp-echo
   version: v2
 template:
  metadata:
   labels:
    app: tcp-echo
    version: v2
  spec:
   containers:
   - name: tcp-echo
     image: docker.io/istio/tcp-echo-server:1.1
     imagePullPolicy: IfNotPresent
     args: [ "9000", "two" ]
     ports:
     - containerPort: 9000
```

在**无状态**页面可以看到新创建的两个版本的TCP-Echo应用。

2. 创建一个服务，并将其对外暴露。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。

iv. 在集群管理页左侧导航栏中，单击**服务**。

v. 在**服务**页面**命名空间**下拉列表中选择命名空间。

vi. 在**服务**页面，单击右上角的**创建**。

vii. 在**创建服务**对话框中，设置服务的相关信息，然后单击**创建**。

需要设置的参数如下所示：

- **名称**：设为tcp-echo。
- **类型**：选择服务类型，即服务访问的方式。支持**虚拟集群IP**、**节点端口**和**负载均衡**。

> **说明** 您的服务类型为**虚拟集群IP**时，才能设置**实例间发现服务（Headless Service）**。您可以使用无头Service与其他服务发现机制进行接口，而不必与Kubernetes的实现捆绑在一起。

- **关联**：选择tcp-echo-v1。

> **说明** 将从关联的Deployment中提取app这个标签作为Service的Selector，这决定了Kubernetes service将流量转发到哪个Deployment。由于tcp-echo-v1和tcp-echo-v2两个Deployment拥有相同的标签，即app:tcp-echo，因此可以关联任一Deployment。

- **外部流量策略**：可选值为Local或Cluster。

> **说明** 您的服务类型为**节点端口**或**负载均衡**时，才能设置**外部流量策略**。

- **端口映射**：**名称**设为tcp；**服务端口**和**容器端口**设为9000；**协议**设为TCP。
- **注解**：为该服务添加一个注解（annotation），配置负载均衡的参数，例如设置 `service.beta.kubernetes.io/alicloud-loadbalancer-bandwidth:20` 表示将该服务的带宽峰值设置为20Mbit/s，从而控制服务的流量。更多参数请参见通过Annotation配置负载均衡。
- **标签**：您可为该服务添加一个标签，标识该服务。

在**服务（Service）**页面可以看到新创建的服务，type-echo。

步骤二：设置服务网格ASM的路由规则

通过设置服务网格的Istio网关、虚拟服务和目标规则，将流量全部指向tcp-echo服务的v1版本。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 创建服务网关。
   i. 在**控制平面**区域，选择**Gateway**页签，然后单击**新建**。

ii. 在**新建**页面，从**命名空间**下拉列表中选择default，并在文本框中输入以下Yaml文件内容，单击**确定**。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
 name: tcp-echo-gateway
spec:
 selector:
  istio: ingressgateway
 servers:
 - port:
    number: 31400
    name: tcp
    protocol: TCP
  hosts:
  - "*"
```

在**Gateway**页签下可以看到新创建的名为tcp-echo-gateway的Istio网关。

5. 创建虚拟服务。

i. 在**控制平面**区域，选择**VirtualService**页签，然后单击**新建**。

ii. 在**新建**页面，从**命名空间**下拉列表中选择default，并在文本框中输入以下Yaml文件内容，单击**确定**。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: tcp-echo
spec:
 hosts:
 - "*"
 gateways:
 - tcp-echo-gateway
 tcp:
 - match:
  - port: 31400
  route:
  - destination:
     host: tcp-echo
     port:
      number: 9000
     subset: v1
```

在VirtualService页签下可以看到新创建的名为tcp-echo的虚拟服务。

6. 创建目标规则。

i. 在**控制平面**区域，选择DestinationRule页签，然后单击**新建**。

ii. 在**新建**页面，从**命名空间**下拉列表中选择default，并在文本框中输入以下Yaml文件内容，单击**确定**。

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: tcp-echo-destination
spec:
 host: tcp-echo
 subsets:
 - name: v1
  labels:
   version: v1
 - name: v2
  labels:
   version: v2
```

在**DestinationRule**页签下可以看到新创建的名为tcp-echo-destination的目标规则。

步骤三：部署入口网关

在入口网关中，添加端口31400，并指向Istio网关的31400端口。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在**数据平面**区域，单击**部署入口网关**。
5. 在**部署入口网关**页面，为集群添加入口网关。
   i. 从**部署集群**列表中选择要部署入口网关的集群。
   ii. 指定负载均衡的类型为**公网访问**。
   iii. 选择负载均衡。
   - 使用已有负载均衡：从已有负载均衡列表中选择。
   - 新建负载均衡：单击**新建负载均衡**，从下拉列表中选择所需的负载均衡规格。

> **说明** 建议您为每个Kubernetes服务分配一个SLB。如果多个Kubernetes服务复用同一个SLB，存在以下风险和限制：
>
> - 使用已有的SLB会强制覆盖已有监听，可能会导致您的应用不可访问。
> - Kubernetes通过Service创建的SLB不能复用，只能复用您手动在控制台（或调用OpenAPI）创建的SLB。
> - 复用同一个SLB的多个Service不能有相同的前端监听端口，否则会造成端口冲突。
> - 复用SLB时，监听的名字以及虚拟服务器组的名字被Kubernetes作为唯一标识符。请勿修改监听和虚拟服务器组的名字。
> - 不支持跨集群复用SLB。

iv. 单击**添加端口**，将**名称**设为tcp，**服务端口**和**容器端口**设为31400。

> **说明** 服务端口指的是整个网格对外暴露的端口，是外部访问使用的端口；而容器端口指的是从服务端口进来的流量所指向的Istio网关端口，所以容器端口需要与Istio网关一致。

v. 单击**确定**。

步骤四：检查部署结果

通过Kubectl确认tcp-echo服务的流量指向是否符合预期。

1. 通过Kubectl客户端连接至Kubernetes集群。详情请参见通过kubectl连接Kubernetes集群。
2. 执行以下命令，获得服务的地址与端口。

```
export INGRESS_HOST=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="tcp")].port}')
```

3. 使用telnet指令向tcp-echo服务发起连接请求。

```
telnet $INGRESS_HOST $INGRESS_PORT
Trying xxx.xxx.xxx.xxx...
Connected to xxx.xxx.xxx.xxx.
Escape character is '^]'
```

4. 输入任意字符串，按Enter发送，返回的字符串前面带了“one”。这说明tcp-echo服务已经成功部署，且流量全部指向了tcp-echo-v1版本。

步骤五：按比例将流量路由到tcp-echo-v2

此处将20%的流量指向tcp-echo-v2版本，其余80%仍然打到tcp-echo-v1。

1. 修改ASM实例的虚拟服务配置。
   i. 在**控制平面**区域，选择**VirtualService**页签
   ii. 在**VirtualService**页面，单击tcp-echo所在**操作**列中的**yaml**。
   在**VirtualService**页签下可以看到新创建的名为tcp-echo的虚拟服务。

iii. 在**编辑实例**页面的文本框中输入以下Yaml文件内容，单击**确定**。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: tcp-echo
spec:
 hosts:
 - "*"
 gateways:
 - tcp-echo-gateway
 tcp:
 - match:
  - port: 31400
  route:
  - destination:
     host: tcp-echo
     port:
       number: 9000
     subset: v1
    weight: 80
  - destination:
     host: tcp-echo
     port:
       number: 9000
     subset: v2
    weight: 20
```

2. 执行以下命令，向tcp-echo服务发起10次请求。

```
$ for i in {1..10}; do \
docker run -e INGRESS_HOST=$INGRESS_HOST -e INGRESS_PORT=$INGRESS_PORT -it --rm busybox
sh -c "(date; sleep 1) | nc $INGRESS_HOST $INGRESS_PORT"; \
done
one Mon Nov 12 23:38:45 UTC 2018
two Mon Nov 12 23:38:47 UTC 2018
one Mon Nov 12 23:38:50 UTC 2018
one Mon Nov 12 23:38:52 UTC 2018
one Mon Nov 12 23:38:55 UTC 2018
two Mon Nov 12 23:38:57 UTC 2018
one Mon Nov 12 23:39:00 UTC 2018
one Mon Nov 12 23:39:02 UTC 2018
one Mon Nov 12 23:39:05 UTC 2018
one Mon Nov 12 23:39:07 UTC 2018
```

根据以上请求的分发情况，可以看到20%的流量指向了tcp-echo-v2。

> **说明** 您的测试结果可能不一定总是10次中有2次打到tcp-echo-v2，但从较长时间范围的总体比例来看，一定是接近20% 的。

2.使用ASM为网格内gRPC服务实现负载均衡

在使用gRPC（基于HTTP/2）的Kubernetes服务时，到目标的单个连接将在一个Pod处终止。如果从客户端发送了多条消息，则所有消息将由该Pod处理，从而导致负载不均衡。本文通过示例介绍gRPC服务间负载不均衡的问题以及如何实现负载均衡。

背景信息

gRPC是一种基于HTTP/2的服务通信协议，使用基于Protocol Buffers（简称为PB）格式的服务定义。服务之间调用的数据可以被序列化为较小的二进制格式进行传输。使用gRPC，可以从*.proto*文件生成多种语言的代码，这也就使得gRPC成为了多语言微服务开发的最佳选择之一。

使用基于HTTP/1.1的RPC时，一个简单的TCP负载平衡器足以胜任，因为这些连接都是短暂的，客户端将尝试重新连接，不会保持与运行中的旧Pod之间的连接。但是使用基于HTTP/2的gRPC时，TCP连接保持打开状态，这样将保持连接到即将失效的Pod，亦或使集群失去平衡。

gRPC服务间调用的负载不均衡

通过一个以下示例可以看出在Kubernetes下gRPC服务间调用的负载不均衡问题。

前提条件：

- 已创建至少一个Kubernetes集群。
- 已设置通过kubectl连接该集群，详情参见通过kubectl连接Kubernetes集群。

操作步骤：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理命名空间与配额**。
5. 在**命名空间**页面单击右上方的**创建**，在**创建命名空间**对话框，输入命名空间的名称，例如grpc-nosidecar，单击**确定**。
6. 在命名空间grpc-nosidecar下部署gRPC服务的服务端istio-grpc-server。

   假设，待创建的描述文件为*istio-grpc-server.yaml*，请执行如下命令：

   ```
   kubectl apply -n grpc-nosidecar -f istio-grpc-server.yaml
   ```

   其中，*istio-grpc-server.yaml*文件的内容如下：

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
    name: istio-grpc-server-v1
    labels:
      app: istio-grpc-server
      version: v1
   spec:
   ```

```
  replicas: 1
  selector:
   matchLabels:
    app: istio-grpc-server
    version: v1
  template:
   metadata:
    labels:
     app: istio-grpc-server
     version: v1
   spec:
    containers:
    - args:
     - --address=0.0.0.0:8080
     image: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-server
     imagePullPolicy: Always
     livenessProbe:
      exec:
       command:
       - /bin/grpc_health_probe
       - -addr=:8080
      initialDelaySeconds: 2
     name: istio-grpc-server
     ports:
     - containerPort: 8080
     readinessProbe:
      exec:
       command:
       - /bin/grpc_health_probe
       - -addr=:8080
      initialDelaySeconds: 2
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: istio-grpc-server-v2
 labels:
  app: istio-grpc-server
  version: v2
spec:
 replicas: 1
```

```
  selector:
    matchLabels:
      app: istio-grpc-server
      version: v2
  template:
    metadata:
      labels:
        app: istio-grpc-server
        version: v2
    spec:
      containers:
        - args:
            - --address=0.0.0.0:8080
          image: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-server
          imagePullPolicy: Always
          livenessProbe:
            exec:
              command:
                - /bin/grpc_health_probe
                - -addr=:8080
            initialDelaySeconds: 2
          name: istio-grpc-server
          ports:
            - containerPort: 8080
          readinessProbe:
            exec:
              command:
                - /bin/grpc_health_probe
                - -addr=:8080
            initialDelaySeconds: 2
---
apiVersion: v1
kind: Service
metadata:
  name: istio-grpc-server
  labels:
    app: istio-grpc-server
spec:
  ports:
  - name: grpc-backend
```

```
    port: 8080
    protocol: TCP
  selector:
    app: istio-grpc-server
  type: ClusterIP
---
```

7. 在命名空间grpc-nosidecar下部署gRPC服务的服务端istio-grpc-client。

   假设，待创建的描述文件为*istio-grpc-client.yaml*，请执行如下命令：

   ```
   kubectl apply -n grpc-nosidecar -f istio-grpc-client.yaml
   ```

   其中，*istio-grpc-client.yaml*文件的内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: istio-grpc-client
 labels:
  app: istio-grpc-client
spec:
 replicas: 1
 selector:
  matchLabels:
   app: istio-grpc-client
 template:
  metadata:
   labels:
    app: istio-grpc-client
  spec:
   containers:
   - image: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-client
     imagePullPolicy: Always
     command: ["/bin/sleep", "3650d"]
     name: istio-grpc-client
---
apiVersion: v1
kind: Service
metadata:
 name: istio-grpc-client
spec:
 ports:
 - name: grpc
  port: 8080
  protocol: TCP
 selector:
  app: istio-grpc-client
 type: ClusterIP
---
```

8. 执行以下命令，查看Pod运行状态：

```
kubectl get pod -n grpc-nosidecar
```

示例输出如下：

```
NAME                  READY  STATUS  RESTARTS  AGE
```

```
istio-grpc-client-dd56bcb45-hvmjt    1/1   Running  0     95m
istio-grpc-server-v1-546d9876c4-j2p9r  1/1   Running  0     95m
istio-grpc-server-v2-66d9b8847-276bd   1/1   Running  0     95m
```

9. 执行以下命令，登录到客户端Pod容器。

```
kubectl exec -it -n grpc-nosidecar istio-grpc-client-dd56bcb45-hvmjt sh
```

10. 进入容器后，执行以下命令：

```
/bin/greeter-client --insecure=true --address=istio-grpc-server:8080 --repeat=100
```

可以看到所有的请求都指向了一个Pod：

```
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
```

由此可见，从客户端发送了所有消息都由一个Pod处理，从而导致负载不均衡。

使用ASM实现负载均衡

下面将示例说明如何通过ASM实现负载均衡。

前提条件：

- 已创建至少一个服务网格ASM实例，并已经添加至少一个集群到该实例中。
- 已设置通过kubectl连接到该集群，详情参见通过kubectl连接Kubernetes集群。

操作步骤：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**节点管理命名空间与配额**。
5. 在**命名空间**页面单击右上方的**创建**，在**创建命名空间**对话框，输入命名空间的名称，例如grpc-nosidecar，并新增标签 `istio-injection:enabled` ，单击**确定**。
6. 命名空间grpc-nosidecar下部署gRPC服务的服务端istio-grpc-server。

   假设，待创建的描述文件为*istio-grpc-server.yaml*，请执行如下命令：

```
kubectl apply -n grpc-nosidecar -f istio-grpc-server.yaml
```

*istio-grpc-server.yaml*文件的内容，参见上文示例。

7. 命名空间grpc-nosidecar下部署gRPC服务的服务端istio-grpc-client。

   假设，待创建的描述文件为*istio-grpc-client.yaml*，请执行如下命令：

```
kubectl apply -n grpc-nosidecar -f istio-grpc-client.yaml
```

   *istio-grpc-client.yaml*文件的内容，参见上文示例。

8. 执行以下命令，查看Pod运行状态：

```
kubectl get pod -n grpc-nosidecar
```

   此时可以看到每个Pod中包含了2个容器，其中一个容器就是注入的Sidecar代理，示例输出如下：

```
NAME                              READY  STATUS   RESTARTS  AGE
istio-grpc-client-dd56bcb45-zhfsg     2/2    Running  0         1h15m
istio-grpc-server-v1-546d9876c4-tndsm  2/2    Running  0         1h15m
istio-grpc-server-v2-66d9b8847-99v62   2/2    Running  0         1h15m
```

9. 执行以下命令，登录到客户端Pod容器。

```
kubectl exec -it -n grpc-sidecar istio-grpc-client-dd56bcb45-zhfsg sh
```

10. 进入容器后，执行以下命令：

```
/bin/greeter-client --insecure=true --address=istio-grpc-server:8080 --repeat=100
```

   可以看到所有的请求分别指向了对应的2个Pod，比例接近于50：50，即负载均衡中的Round-Robin。

```
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
2020/01/14 14:37:14 Hello world from istio-grpc-server-v2-66d9b8847-276bd
```

11. 配置服务网格ASM实例的控制平面。

    i. 登录ASM控制台，进入待配置服务网格的详情页。

    ii. 在**控制平面**区域，选择**Namespace**页签，创建名为grpc-sidecar的命名空间。

iii. 选择DestinationRule页签，在grpc-sidecar命名空间下新建目标规则。YAML文件内容如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: dr-istio-grpc-server
spec:
 host: istio-grpc-server
 trafficPolicy:
  loadBalancer:
   simple: ROUND_ROBIN
 subsets:
  - name: v1
   labels:
    version: "v1"
  - name: v2
   labels:
    version: "v2"
```

iv. 选择VirtualService页签，在grpc-sidecar命名空间下新建虚拟服务。YAML文件内容如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: vs-istio-grpc-server
spec:
 hosts:
  - "istio-grpc-server"
 http:
  - match:
    - port: 8080
   route:
    - destination:
       host: istio-grpc-server
       subset: v1
      weight: 90
    - destination:
       host: istio-grpc-server
       subset: v2
      weight: 10
```

v. 执行以下命令，登录到客户端Pod容器。

```
kubectl exec -it -n grpc-sidecar istio-grpc-client-dd56bcb45-zhfsg sh
```

vi. 进入容器后，执行以下命令：

```
/bin/greeter-client --insecure=true --address=istio-grpc-server:8080 --repeat=100
```

可以看到所有的请求分别指向了对应的2个Pod，但比例接近于90：10，并非默认的Round-Robin。

vii. 选择**VirtualService**页签，找到grpc-sidecar命名空间下名为vs-istio-grpc-server的虚拟服务，单击**YAML**，调整虚拟服务的权重，以查看调用的不同结果。

修改内容如下：

```
route:
  - destination:
      host: istio-grpc-server
      subset: v1
    weight: 0
  - destination:
      host: istio-grpc-server
      subset: v2
    weight: 100
```

viii. 通过Kubectl登录容器，执行以下命令：

```
/bin/greeter-client --insecure=true --address=istio-grpc-server:8080 --repeat=100
```

可以看到所有的请求分别指向了对应的1个Pod，即版本v2对应的Pod。

3.使用ASM实现基于位置的路由请求

在大规模服务场景下，成千上万个服务运行在不同的地域，这些服务需要相互调用来完成完整的功能。为了确保获得最佳性能，应当将流量路由到最近的服务，使得流量尽可能在同一个区域内，而不是只依赖于Kubernetes默认提供的轮询方式进行负载均衡。基于Istio的阿里云服务网格ASM产品提供了基于位置的路由能力，可以将流量路由到最靠近的容器。这样可以确保服务调用的低延迟，并尽量保持服务调用在同一区域内，降低流量费用。本文介绍如何在ASM内实现基于位置的路由请求，以提高性能并节省成本。

前提条件

- 创建ASM实例。
- 创建一个多可用区的托管版ACK集群，集群使用的节点分别位于同一地域下的不同可用区（本文示例中分别为北京地域下的可用区G和H），详情请参见创建Kubernetes托管版集群。

部署后端示例服务

在ACK集群中分别部署Nginx的两个版本（V1和V2），并通过Nodelabel将版本V1部署到可用区G，将版本V2部署到可用区H，操作步骤如下所示。

1. 在ACK集群中部署Nginx版本V1，对应的YAML如下。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: mynginx-configmap-v1
 namespace: backend
data:
 default.conf: |-
  server {
    listen     80;
    server_name  localhost;
    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;
    location / {
      return 200 'v1\n';
    }
  }
```

2. 通过Nodelabel将版本V1部署到可用区G，对应的YAML如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-v1
 namespace: backend
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
   version: v1
 template:
  metadata:
   labels:
    app: nginx
    version: v1
  spec:
   containers:
   - image: docker.io/nginx:1.15.9
    imagePullPolicy: IfNotPresent
    name: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: nginx-config
     mountPath: /etc/nginx/conf.d
     readOnly: true
   volumes:
   - name: nginx-config
    configMap:
     name: mynginx-configmap-v1
   nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "cn-beijing-g"
```

3. 在ACK集群中部署Nginx版本V2，对应的YAML如下。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: mynginx-configmap-v2
 namespace: backend
data:
 default.conf: |-
  server {
    listen     80;
    server_name  localhost;
    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;
    location / {
      return 200 'v2\n';
    }
  }
```

4. 通过Nodelabel将版本V2部署到可用区H，对应的YAML如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-v2
 namespace: backend
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
   version: v2
 template:
  metadata:
   labels:
    app: nginx
    version: v2
  spec:
   containers:
   - image: docker.io/nginx:1.15.9
    imagePullPolicy: IfNotPresent
    name: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: nginx-config
     mountPath: /etc/nginx/conf.d
     readOnly: true
   volumes:
   - name: nginx-config
    configMap:
     name: mynginx-configmap-v2
   nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "cn-beijing-h"
```

5. 部署后端示例应用服务，对应的YAML如下。

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 namespace: backend
 labels:
  app: nginx
spec:
 ports:
 - name: http
  port: 8000
  targetPort: 80
 selector:
  app: nginx
```

部署客户端示例服务

在ACK集群中分别部署2个客户端示例服务，并通过Nodelabel将版本V1部署到可用区G，将版本V2部署到可用区H，对应的操作步骤如下。

1. 在ACK集群中部署客户端示例服务，通过Nodelabel将版本V1部署到可用区G，对应的YAML如下所示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: sleep-cn-beijing-g
 namespace: backend
spec:
 replicas: 1
 selector:
  matchLabels:
   app: sleep
   version: v1
 template:
  metadata:
   labels:
    app: sleep
    version: v1
  spec:
   containers:
   - name: sleep
    image: tutum/curl
    command: ["/bin/sleep","infinity"]
    imagePullPolicy: IfNotPresent
   nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "cn-beijing-g"
```

2. 在ACK集群中部署客户端示例服务，通过Nodelabel将版本V2部署到可用区H，对应的YAML如下所示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: sleep-cn-beijing-h
 namespace: backend
spec:
 replicas: 1
 selector:
  matchLabels:
   app: sleep
   version: v2
 template:
  metadata:
   labels:
    app: sleep
    version: v2
  spec:
   containers:
   - name: sleep
    image: tutum/curl
    command: ["/bin/sleep","infinity"]
    imagePullPolicy: IfNotPresent
   nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "cn-beijing-h"
```

3. 部署客户端示例服务，对应的YAML如下。

```
apiVersion: v1
kind: Service
metadata:
 name: sleep
 namespace: backend
 labels:
  app: sleep
spec:
 ports:
 - name: http
  port: 80
 selector:
  app: sleep
```

4. 执行如下脚本，在2个Sleep Pod中访问后端示例服务。

```
echo "entering into the 1st container"
export SLEEP_ZONE_1=$(kubectl get pods -lapp=sleep,version=v1 -n backend  -o 'jsonpath={.items[0].metadata.name}')
for i in {1..20}
do
 kubectl exec -it $SLEEP_ZONE_1 -c sleep -n backend -- sh -c 'curl  http://nginx.backend:8000'
done
echo "entering into the 2nd container"
export SLEEP_ZONE_2=$(kubectl get pods -lapp=sleep,version=v2 -n backend  -o 'jsonpath={.items[0].metadata.name}')
for i in {1..20}
do
 kubectl exec -it $SLEEP_ZONE_2 -c sleep -n backend -- sh -c 'curl  http://nginx.backend:8000'
done
```

执行结果

通过执行结果可以看到后端示例服务以循环方式进行负载均衡。

```
entering into the 1st container
v2
v1
v1
v2
v2
v1
v1
v2
v2
v1
v1
v2
v2
v1
v2
v1
v1
v2
v2
v1
entering into the 2nd container
v2
```

```
v2
v1
v1
v2
v1
v2
v1
v2
v2
v1
v1
v2
v2
v1
v2
v1
v2
v1
v1
```

本地优先负载均衡

本地优先的负载均衡算法会优先访问同一可用区的Pod，并在本可用区Pod不可用时，调用其他可用区的Pod。

为了实现本地优先的负载均衡，您需要具有虚拟服务和带有异常策略的目标规则。离群策略检查Pod是否健康，并做出路由决策。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在**控制平面**区域，单击VirtualService页签，然后单击**新建**。
5. 在**新建**页面，按照以下步骤定义VirtualService，然后单击**确定**。
    i. 选择相应的**命名空间**。本文以选择backend命名空间为例。

ii. 在文本框中，定义Istio虚拟服务，YAML定义如下所示。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: nginx
 namespace: backend
spec:
 hosts:
  - nginx
 http:
 - route:
  - destination:
     host: nginx
```

6. 在**新建**页面，按照以下步骤定义**DestinationRule**，然后单击**确定**。

i. 选择相应的**命名空间**。本文以选择backend命名空间为例。

ii. 在文本框中，定义Istio目标规则，YAML定义如下所示。

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: nginx
 namespace: backend
spec:
 host: nginx
 trafficPolicy:
  outlierDetection:
   consecutiveErrors: 7
   interval: 30s
   baseEjectionTime: 30s
```

7. 执行如下脚本，在2个Sleep Pod中再次访问后端示例服务。

```
echo "entering into the 1st container"
export SLEEP_ZONE_1=$(kubectl get pods -lapp=sleep,version=v1 -n backend  -o 'jsonpath={.items[0].metadata.name}')
for i in {1..20}
do
 kubectl exec -it $SLEEP_ZONE_1 -c sleep -n backend -- sh -c 'curl  http://nginx.backend:8000'
done
echo "entering into the 2nd container"
export SLEEP_ZONE_2=$(kubectl get pods -lapp=sleep,version=v2 -n backend  -o 'jsonpath={.items[0].metadata.name}')
for i in {1..20}
do
 kubectl exec -it $SLEEP_ZONE_2 -c sleep -n backend -- sh -c 'curl  http://nginx.backend:8000'
done
```

执行结果

通过执行结果可以看到以本地优先方式进行负载均衡，执行结果如下。

```
entering into the 1st container
v1
v1
v1
v1
v1
v1
v1
v2
v1
v1
v2
v1
v1
v1
v1
v1
v1
v1
v1
v1
v1
entering into the 2nd container
v2
```

```
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
```

局部加权的负载均衡

大多数情况下使用本地优先的负载均衡可以满足平衡的要求，但是有时候您可能需要将流量分成多个区域，如果所有请求都来自单个区域，则容易使一个区域超载。此时可以使用局部加权的负载均衡。

譬如设置如下的加权规则：

- 将从cn-beijing/cn-beijing-g的80%流量路由到cn-beijing/cn-beijing-g，20%的流量路由到cn-beijing/cn-beijing-h。
- 将从cn-beijing/cn-beijing-h的20%流量路由到cn-beijing/cn-beijing-g，80%的流量路由到cn-beijing/cn-beijing-h。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在**控制平面**区域，单击VirtualService页签，然后单击**新建**。
5. 在**新建**页面，按照以下步骤定义VirtualService，然后单击**确定**。
    i. 选择相应的**命名空间**。本文以选择backend命名空间为例。

ii. 在文本框中，定义Istio虚拟服务，YAML定义如下所示。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: nginx
 namespace: backend
spec:
 hosts:
  - nginx
 http:
 - route:
  - destination:
     host: nginx
```

6. 在**新建**页面，按照以下步骤定义**DestinationRule**，然后单击**确定**。
   i. 选择相应的**命名空间**。本文以选择backend命名空间为例。
   ii. 在文本框中，定义Istio目标规则，YAML定义如下所示。

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: nginx
 namespace: backend
spec:
 host: nginx
 trafficPolicy:
  outlierDetection:
   consecutiveErrors: 7
   interval: 30s
   baseEjectionTime: 30s
  loadBalancer:
   localityLbSetting:
    enabled: true
    distribute:
    - from: cn-beijing/cn-beijing-g/*
      to:
       "cn-beijing/cn-beijing-g/*": 80
       "cn-beijing/cn-beijing-h/*": 20
    - from: cn-beijing/cn-beijing-h/*
      to:
       "cn-beijing/cn-beijing-g/*": 20
       "cn-beijing/cn-beijing-h/*": 80
```

7. 执行如下脚本，在2个Sleep Pod中再次访问后端示例服务。

```
echo "entering into the 1st container"
export SLEEP_ZONE_1=$(kubectl get pods -lapp=sleep,version=v1 -n backend  -o 'jsonpath={.items[0].
metadata.name}')
for i in {1..20}
do
 kubectl exec -it $SLEEP_ZONE_1 -c sleep -n backend -- sh -c 'curl  http://nginx.backend:8000'
done
echo "entering into the 2nd container"
export SLEEP_ZONE_2=$(kubectl get pods -lapp=sleep,version=v2 -n backend  -o 'jsonpath={.items[0].
metadata.name}')
for i in {1..20}
do
 kubectl exec -it $SLEEP_ZONE_2 -c sleep -n backend -- sh -c 'curl  http://nginx.backend:8000'
done
```

执行结果

通过执行结果可以看到以局部加权比例进行负载均衡，执行结果如下。

```
entering into the 1st container
v1
v1
v1
v1
v2
v1
v1
v2
v1
v1
v1
v2
v1
v1
v1
v1
v1
v2
v1
v1
entering into the 2nd container
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v2
v1
```

```
v2
v1
v2
v2
```

4.在ASM上访问外部服务

服务网格ASM提供了访问外部服务的三种方式，包含设置外部服务访问策略、配置ServiceEntry和设置拦截对外访问的网段。本文介绍如何在服务网格ASM上访问外部服务。

设置外部服务访问策略

您可以在服务网格ASM中设置**对外部服务的访问策略OutboundTrafficPolicy**，用于配置对外部服务（即未在Istio的内部服务注册表中定义的服务）的访问策略。

> **说明** 登录ASM控制台，在左侧导航栏中单击**概览**，选择目标网格实例，您可以查看Istio的内部服务注册表中定义的服务。

- 如果该选项设置为*ALLOW_ANY*，则Sidecar代理允许对未知服务的透传通过。优点是直接透传对外部服务的访问，缺点是失去了对外部服务流量的网格层面的监视和控制。
- 如果该选项设置为*REGISTRY_ONLY*，则Sidecar代理将阻止任何没有在网格中定义了HTTP服务或服务条目的主机。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面右上角单击**功能设置**。
5. 在**功能设置更新**对话框中单击**展开高级选项**，设置**对外部服务的访问策略OutboundTrafficPolicy**为**ALLOW_ANY**，单击**确定**。
   在注入了Sidecar代理的应用容器中，运行**curl**命令请求访问外部HTTP或HTTPS服务，可以看到正常返回结果。

   > 说明

   - 访问外部HTTP服务

     ```
     curl -I http://www.aliyun.com/
     ```

     ```
     HTTP/1.1 301 Moved Permanently
     server: envoy
     date: Mon, 07 Sep 2020 09:28:54 GMT
     content-type: text/html
     content-length: 239
     location: https://www.aliyun.com/
     eagleeye-traceid: 0be3e0a615994709353116335ea5ea
     timing-allow-origin: *
     x-envoy-upstream-service-time: 67
     ```

   - 访问外部HTTPS服务

     ```
     curl -I https://www.aliyun.com/
     ```

```
HTTP/2 200
server: Tengine
date: Mon, 07 Sep 2020 09:16:31 GMT
content-type: text/html; charset=utf-8
vary: Accept-Encoding
vary: Accept-Encoding
strict-transport-security: max-age=31536000
x-download-options: noopen
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
x-readtime: 0
eagleeye-traceid: 0b57ff8715994701916963132ec7ad
strict-transport-security: max-age=0
timing-allow-origin: *
```

配置ServiceEntry

如果设置**对外部服务的访问策略OutboundTrafficPolicy为REGISTRY_ONLY**，在注入了Sidecar代理的应用容器中，运行**curl**命令请求访问外部HTTP或HTTPS服务，可以看到不能正常返回结果。

- 访问外部HTTP服务。

  ```
  curl -I  http://www.aliyun.com/
  ```

  ```
  HTTP/1.1 502 Bad Gateway
  date: Mon, 07 Sep 2020 09:25:58 GMT
  server: envoy
  transfer-encoding: chunked
  ```

- 访问外部HTTPS服务。

  ```
  curl -I https://www.aliyun.com/
  ```

  ```
  curl: (35) LibreSSL SSL_connect: SSL_ERROR_SYSCALL in connection to www.aliyun.com:443
  ```

您可以使用ServiceEntry配置，从网格内访问网格外部的可公开访问的服务，同时保留Istio的流量监视和控制功能，实现对外部服务的受控访问。

1. 按以下示例创建ServiceEntry，详细介绍请参见管理ServiceEntry。

   根据实际需求设置hosts，在本示例中hosts为*www.aliyun.com*。

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
 name: aliyun-com-ext
spec:
 hosts:
 - 'www.aliyun.com'
 ports:
 - number: 80
  name: http
  protocol: HTTP
 - number: 443
  name: https
  protocol: HTTPS
 resolution: DNS
 location: MESH_EXTERNAL
```

2. 访问外部HTTP或HTTPS服务，可以看到正常返回结果。
   - 访问外部HTTP服务。

     ```
     curl -I  http://www.aliyun.com/
     ```

     ```
     HTTP/1.1 301 Moved Permanently
     server: envoy
     date: Mon, 07 Sep 2020 09:49:17 GMT
     content-type: text/html
     content-length: 239
     location: https://www.aliyun.com/
     eagleeye-traceid: 0be3e0a915994721583014504e7b31
     timing-allow-origin: *
     x-envoy-upstream-service-time: 66
     ```

   - 访问外部HTTPS服务。

     ```
     curl -I https://www.aliyun.com/
     ```

```
HTTP/2 200
server: Tengine
date: Mon, 07 Sep 2020 09:49:31 GMT
content-type: text/html; charset=utf-8
vary: Accept-Encoding
vary: Accept-Encoding
strict-transport-security: max-age=31536000
x-download-options: noopen
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
x-readtime: 1
eagleeye-traceid: 0be3e0b115994721709577294ed9e8
strict-transport-security: max-age=0
timing-allow-origin: *
```

3. 按以下示例创建虚拟服务，详细介绍请参见管理VirtualService。为使用ServiceEntry配置访问的外部服务设置路由规则。在本示例中，对www.aliyun.com服务的调用注入了延迟时间fixedDelay的规则。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: aliyun-com-ext
spec:
 hosts:
  - 'www.aliyun.com'
 http:
 - fault:
    delay:
     percent: 100
     fixedDelay: 5s
  route:
   - destination:
      host: www.aliyun.com
     weight: 100
```

4. 查看定义的路由规则是否生效。

   返回结果中real的时间为5.07s，说明路由规则已生效。

```
time curl -o /dev/null -s -w "%{http_code}\n" http://www.aliyun.com/
```

```
301
real 0m 5.07s
user 0m 0.00s
sys 0m 0.00s
```

设置网格拦截指定地址范围

设置指定的网格拦截IP范围，使得未被指定的其他IP范围不被网格内的Sidecar代理流量拦截，从而可以绕过Sidecar代理直接访问目标服务。

您可以在服务网格ASM中配置**拦截对外访问的地址范围**，设置流量被拦截的IP范围，通常设置为所管理的Kubernetes集群的Service CIDR。即访问集群内目标服务需要经过服务网格内的Sidecar代理进行流量拦截，非集群内目标则绕过服务网格内的Sidecar代理。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面右上角单击**功能设置**。
5. 在**功能设置更新**对话框中单击**展开高级选项**，在**拦截对外访问的地址范围**文本框中设置拦截的网段，单击**确定**。

> **说明** 拦截对外访问的地址范围默认为*，表示对所有的IP网段进行拦截。可以根据实际需要设置拦截的IP网段，一般情况下可以设置为Kubernetes集群的Service CIDR。

5.通过入口网关访问网格内gRPC服务

服务网格ASM的流量管理功能支持通过入口网关访问内部的gRPC服务。本文通过示例介绍如何通过ASM入口网关访问内部gRPC服务，并在gRPC的两个版本之间进行流量切换。

前提条件

- 已开通以下服务：
  - 容器服务Kubernetes版
  - 负载均衡
- 已创建至少一个ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 已创建一个ASM实例，并已将ACK集群添加到ASM实例中。具体操作，请参见创建ASM实例和添加集群到ASM实例。

步骤一：部署示例应用

部署名为istio-grpc-server-v1和istio-grpc-server-v2的示例应用。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。
4. 在**无状态**页面**命名空间**下拉列表中选择命名空间。

   > **说明** 当前选中的命名空间应当已标注自动注入Sidecar，即包含istio-system=enabled标签，详细介绍请参见设置Sidecar自动注入。

5. 在**无状态**页面单击**使用模板创建**。
6. 在模板创建页面将下面的YAML模版粘贴到模版文本框中，单击**创建**。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: istio-grpc-server-v1
 labels:
  app: istio-grpc-server
  version: v1
spec:
 replicas: 1
 selector:
  matchLabels:
   app: istio-grpc-server
   version: v1
 template:
  metadata:
   labels:
```

```
      app: istio-grpc-server
      version: v1
    spec:
     containers:
     - args:
      - --address=0.0.0.0:8080
      image: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-server
      imagePullPolicy: Always
      livenessProbe:
       exec:
        command:
        - /bin/grpc_health_probe
        - -addr=:8080
       initialDelaySeconds: 2
      name: istio-grpc-server
      ports:
      - containerPort: 8080
      readinessProbe:
       exec:
        command:
        - /bin/grpc_health_probe
        - -addr=:8080
       initialDelaySeconds: 2
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: istio-grpc-server-v2
 labels:
  app: istio-grpc-server
  version: v2
spec:
 replicas: 1
 selector:
  matchLabels:
   app: istio-grpc-server
   version: v2
 template:
  metadata:
   labels:
    app: istio-grpc-server
```

```
    version: v2
  spec:
   containers:
    - args:
      - --address=0.0.0.0:8080
      image: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-server
      imagePullPolicy: Always
      livenessProbe:
       exec:
        command:
         - /bin/grpc_health_probe
         - -addr=:8080
       initialDelaySeconds: 2
      name: istio-grpc-server
      ports:
       - containerPort: 8080
      readinessProbe:
       exec:
        command:
         - /bin/grpc_health_probe
         - -addr=:8080
       initialDelaySeconds: 2
---
apiVersion: v1
kind: Service
metadata:
 name: istio-grpc-server
 labels:
  app: istio-grpc-server
spec:
 ports:
 - name: grpc-backend
  port: 8080
  protocol: TCP
 selector:
  app: istio-grpc-server
 type: ClusterIP
---
```

步骤二：设置服务网格ASM的路由规则

设置服务网格的服务网关、虚拟服务和目标规则，将流量全部指向istio-grpc-server-v1。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 创建服务网关。
   i. 在**控制平面**区域单击**Gateway**页签，然后单击**新建**。
   ii. 在**新建**页面设置**命名空间**为**default**，将下面的YAML模板粘贴到文本框中，单击**确定**。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
 name: grpc-gateway
spec:
 selector:
  istio: ingressgateway # use Istio default gateway implementation
 servers:
 - port:
    number: 8080
    name: grpc
    protocol: GRPC
  hosts:
  - "*"
```

   在**Gateway**页签下可以看到名为grpc-gateway的服务网关。
5. 创建目标规则。
   i. 在**控制平面**区域单击**DestinationRule**页签，然后单击**新建**。

ii. 在**新建**页面设置**命名空间**为**default**，将下面的YAML模板粘贴到文本框中，单击**确定**。

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: dr-istio-grpc-server
spec:
 host: istio-grpc-server
 trafficPolicy:
  loadBalancer:
   simple: ROUND_ROBIN
 subsets:
  - name: v1
   labels:
    version: "v1"
  - name: v2
   labels:
    version: "v2"
```

在**DestinationRule**页签下可以看到名为dr-istio-grpc-server的目标规则。

6. 创建虚拟服务。
   i. 在**控制平面**区域单击**VirtualService**页签，然后单击**新建**。
   ii. 在**新建**页面设置**命名空间**为**default**，将下面的YAML模板粘贴到文本框中，单击**确定**。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: grpc-vs
spec:
 hosts:
 - "*"
 gateways:
 - grpc-gateway
 http:
  - match:
    - port: 8080
   route:
    - destination:
      host: istio-grpc-server
      subset: v1
     weight: 100
    - destination:
      host: istio-grpc-server
      subset: v2
     weight: 0
```

在VirtualService页签下可以看到名为grpc-vs的虚拟服务。

步骤三：部署入口网关

在入口网关中，添加端口8080，并指向服务网关的8080端口。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在**数据平面**区域单击**入口网关服务**页签，单击**部署默认入口网关**。
5. 在**部署入口网关**页面配置参数。

参数	配置项
部署集群	选择要部署入口网关的集群。
负载均衡类型	此处指定负载均衡的类型为**公网访问**。

参数	配置项
负载均衡	选择负载均衡。 ○ 使用已有负载均衡：从已有负载均衡列表中选择。 ○ 新建负载均衡：单击**新建负载均衡**，从下拉列表中选择所需的负载均衡规格。 **说明** 建议您为每个Kubernetes服务分配一个SLB。如果多个Kubernetes服务复用同一个SLB，存在以下风险和限制： ○ 使用已有的SLB会强制覆盖已有监听，可能会导致您的应用不可访问。 ○ Kubernetes通过Service创建的SLB不能复用，只能复用您手动在控制台（或调用OpenAPI）创建的SLB。 ○ 复用同一个SLB的多个Service不能有相同的前端监听端口，否则会造成端口冲突。 ○ 复用SLB时，监听的名字以及虚拟服务器组的名字被Kubernetes作为唯一标识符。请勿修改监听和虚拟服务器组的名字。 ○ 不支持跨集群复用SLB。
端口映射	单击**添加端口**，设置**名称**为tcp，**服务端口**和**容器端口**为8080。 **说明** 服务端口指的是整个网格对外暴露的端口，是外部访问使用的端口。而容器端口指的是从服务端口进来的流量所指向的服务网关端口。所以容器端口需要与服务网关端口一致。

6. 单击**确定**。

步骤四：运行gRPC客户端

1. 启动gRPC客户端。

```
docker run -d --name grpc-client registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-client 365d
```

2. 登录到容器。

```
docker exec -it grpc-client sh
```

3. 访问网格内的gRPC服务。

```
/bin/greeter-client --insecure=true --address=<入口网关的IP地址>:8080 --repeat=100
```

返回以下结果，可以看到所有的请求都指向了istio-grpc-server-v1。

```
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
```

步骤五：按比例将流量路由到v2

将40%的流量指向istio-grpc-server-v2，其余60%的流量仍然指向istio-grpc-server-v1。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在**控制平面**区域单击VirtualService页签。
5. 在VirtualService页签单击grpc-vs**操作**列的**YAML**。
6. 在**编辑**对话框中更新以下YAML内容，单击**确定**。

```
....
    route:
     - destination:
        host: istio-grpc-server
        subset: v1
      weight: 60
     - destination:
        host: istio-grpc-server
        subset: v2
      weight: 40
```

7. 登录到容器，执行以下命令，访问网格内的gRPC服务。

```
/bin/greeter-client --insecure=true --address=<入口网关的IP地址>:8080 --repeat=100
```

返回以下结果，可以看到40%的流量指向了istio-grpc-server-v2。

> **说明** 您的测试结果不一定总是100次中有40次指向istio-grpc-server-v2，但从总体比例来看，一定是接近40% 的。

```
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h74lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h74lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h74lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h74lw
```

6.通过AHAS对网格实例进行流量控制

服务网格ASM配合阿里云应用高可用服务AHAS可以对部署在服务网格内的应用进行流量控制。本文介绍如何创建一个Mesh流控集群，并对网格实例进行流量控制。

前提条件

- 已创建至少一个ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 已创建一个ASM实例，并已将ACK集群添加到ASM实例中。具体操作，请参见创建ASM实例和添加集群到ASM实例。
- 已为ACK集群部署入口网关，请参见添加入口网关服务。
- 已开通AHAS专业版，若没有开通，请进入开通页面。

步骤一：部署演示应用Bookinfo

1. 创建并拷贝以下内容到*bookinfo.yaml*中。

```
apiVersion: v1
kind: Service
metadata:
 name: details
 labels:
  app: details
  service: details
spec:
 ports:
 - port: 9080
  name: http
 selector:
  app: details
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: bookinfo-details
 labels:
  account: details
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: details-v1
 labels:
  app: details
```

```
  version: v1
spec:
 replicas: 1
 selector:
  matchLabels:
   app: details
   version: v1
 template:
  metadata:
   labels:
    app: details
    version: v1
  spec:
   serviceAccountName: bookinfo-details
   containers:
   - name: details
    image: docker.io/istio/examples-bookinfo-details-v1:1.16.2
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 9080
---
##################################################################################################
############
# Ratings service
##################################################################################################
############
apiVersion: v1
kind: Service
metadata:
 name: ratings
 labels:
  app: ratings
  service: ratings
spec:
 ports:
 - port: 9080
  name: http
 selector:
  app: ratings
---
apiVersion: v1
```

```
kind: ServiceAccount
metadata:
 name: bookinfo-ratings
 labels:
  account: ratings
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: ratings-v1
 labels:
  app: ratings
  version: v1
spec:
 replicas: 1
 selector:
  matchLabels:
   app: ratings
   version: v1
 template:
  metadata:
   labels:
    app: ratings
    version: v1
  spec:
   serviceAccountName: bookinfo-ratings
   containers:
   - name: ratings
    image: docker.io/istio/examples-bookinfo-ratings-v1:1.16.2
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 9080
---
##################################################################################################
############
# Reviews service
##################################################################################################
############
apiVersion: v1
kind: Service
```

```
metadata:
 name: reviews
 labels:
  app: reviews
  service: reviews
spec:
 ports:
 - port: 9080
  name: http
 selector:
  app: reviews
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: bookinfo-reviews
 labels:
  account: reviews
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: reviews-v1
 labels:
  app: reviews
  version: v1
spec:
 replicas: 1
 selector:
  matchLabels:
   app: reviews
   version: v1
 template:
  metadata:
   labels:
    app: reviews
    version: v1
  spec:
   serviceAccountName: bookinfo-reviews
   containers:
   - name: reviews
```

```
      image: docker.io/istio/examples-bookinfo-reviews-v1:1.16.2
      imagePullPolicy: IfNotPresent
      env:
      - name: LOG_DIR
        value: "/tmp/logs"
      ports:
      - containerPort: 9080
      volumeMounts:
      - name: tmp
        mountPath: /tmp
      - name: wlp-output
        mountPath: /opt/ibm/wlp/output
    volumes:
    - name: wlp-output
      emptyDir: {}
    - name: tmp
      emptyDir: {}
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: reviews-v2
 labels:
   app: reviews
   version: v2
spec:
 replicas: 1
 selector:
   matchLabels:
     app: reviews
     version: v2
 template:
   metadata:
     labels:
       app: reviews
       version: v2
   spec:
     serviceAccountName: bookinfo-reviews
     containers:
     - name: reviews
       image: docker.io/istio/examples-bookinfo-reviews-v2:1.16.2
```

```
      image: docker.io/istio/examples-bookinfo-reviews-v2:1.16.2
      imagePullPolicy: IfNotPresent
      env:
      - name: LOG_DIR
        value: "/tmp/logs"
      ports:
      - containerPort: 9080
      volumeMounts:
      - name: tmp
        mountPath: /tmp
      - name: wlp-output
        mountPath: /opt/ibm/wlp/output
    volumes:
    - name: wlp-output
      emptyDir: {}
    - name: tmp
      emptyDir: {}
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: reviews-v3
 labels:
   app: reviews
   version: v3
spec:
 replicas: 1
 selector:
   matchLabels:
     app: reviews
     version: v3
 template:
   metadata:
     labels:
       app: reviews
       version: v3
   spec:
     serviceAccountName: bookinfo-reviews
     containers:
     - name: reviews
       image: docker.io/istio/examples-bookinfo-reviews-v3:1.16.2
```

```
      imagePullPolicy: IfNotPresent
      env:
      - name: LOG_DIR
        value: "/tmp/logs"
      ports:
      - containerPort: 9080
      volumeMounts:
      - name: tmp
        mountPath: /tmp
      - name: wlp-output
        mountPath: /opt/ibm/wlp/output
    volumes:
    - name: wlp-output
      emptyDir: {}
    - name: tmp
      emptyDir: {}
---
##################################################################################################
############
# Productpage services
##################################################################################################
############
apiVersion: v1
kind: Service
metadata:
 name: productpage
 labels:
   app: productpage
   service: productpage
spec:
 ports:
 - port: 9080
   name: http
 selector:
   app: productpage
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: bookinfo-productpage
 labels:
```

```
  account: productpage
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: productpage-v1
 labels:
  app: productpage
  version: v1
spec:
 replicas: 1
 selector:
  matchLabels:
   app: productpage
   version: v1
 template:
  metadata:
   labels:
    app: productpage
    version: v1
  spec:
   serviceAccountName: bookinfo-productpage
   containers:
   - name: productpage
    image: docker.io/istio/examples-bookinfo-productpage-v1:1.16.2
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 9080
    volumeMounts:
    - name: tmp
      mountPath: /tmp
   volumes:
   - name: tmp
    emptyDir: {}
---
```

2. 执行以下命令，将Bookinfo应用部署到ASM实例的集群中。

> 说明 `DATA_PLANE_KUBECONFIG` 表示ASM实例集群的Kubeconfig保存在本地的路径。

```
kubectl --kubeconfig=${DATA_PLANE_KUBECONFIG} apply -f bookinfo.yaml
```

3. 创建并拷贝以下内容到*virtual-service-all-v1.yaml*中。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: productpage
spec:
 hosts:
 - productpage
 http:
 - route:
  - destination:
     host: productpage
     subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: reviews
spec:
 hosts:
 - reviews
 http:
 - route:
  - destination:
     host: reviews
     subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: ratings
spec:
 hosts:
 - ratings
 http:
 - route:
  - destination:
     host: ratings
     subset: v1
```

```
    subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: details
spec:
 hosts:
 - details
 http:
 - route:
  - destination:
    host: details
    subset: v1
---
```

4. 执行以下命令，部署Bookinfo应用的VirtualServices。

> 说明 ASM_KUBECONFIG 表示网格实例的Kubeconfig保存在本地的路径。

```
kubectl --kubeconfig=${ASM_KUBECONFIG} apply -f virtual-service-all-v1.yaml
```

5. 创建并拷贝以下内容到*destination-rule-all.yaml*中。

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: productpage
spec:
 host: productpage
 subsets:
 - name: v1
  labels:
   version: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: reviews
spec:
 host: reviews
 subsets:
 - name: v1
```

```
  labels:
    version: v1
 - name: v2
  labels:
    version: v2
 - name: v3
  labels:
    version: v3
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: ratings
spec:
 host: ratings
 subsets:
 - name: v1
  labels:
    version: v1
 - name: v2
  labels:
    version: v2
 - name: v2-mysql
  labels:
    version: v2-mysql
 - name: v2-mysql-vm
  labels:
    version: v2-mysql-vm
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
 name: details
spec:
 host: details
 subsets:
 - name: v1
  labels:
    version: v1
 - name: v2
```

```
  labels:
    version: v2
---
```

6. 执行以下命令，部署Bookinfo应用的Destination。

> 说明 ASM_KUBECONFIG 表示网格实例的Kubeconfig保存在本地的路径。

```
kubectl --kubeconfig=${ASM_KUBECONFIG} apply -f destination-rule-all.yaml
```

7. 创建并拷贝以下内容到*bookinfo-gateway.yaml*中。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
 name: bookinfo-gateway
spec:
 selector:
  istio: ingressgateway # use istio default controller
 servers:
 - port:
    number: 80
    name: http
    protocol: HTTP
  hosts:
  - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: bookinfo
spec:
 hosts:
 - "*"
 gateways:
 - bookinfo-gateway
 http:
 - match:
  - uri:
     exact: /productpage
  - uri:
     prefix: /static
  - uri:
```

```
    exact: /login
  - uri:
    exact: /logout
  - uri:
    prefix: /api/v1/products
  route:
  - destination:
    host: productpage
    port:
     number: 9080
```

8. 执行以下命令，部署Bookinfo应用的Gateway。

> **说明** ASM_KUBECONFIG 表示网格实例的Kubeconfig保存在本地的路径。

```
kubectl --kubeconfig=${ASM_KUBECONFIG} apply -f bookinfo-gateway.yaml
```

步骤二：创建Mesh流控集群

1. 登录AHAS控制台。
2. 在左侧导航栏选择**流量防护 > Mesh防护**。
3. 在**Mesh防护**页面左上角选择地域，单击右上角的**申请Mesh流控集群**。

> **说明** 您需要选择与ACK集群相同的地域，保证Mesh流控集群与ACK集群位于同一地域。

4. 在**集群基本信息**页，填写**集群名称**等信息。

参数	描述
集群名称	填写集群的名称。集群名称应包含1~63个字符，可包含数字、汉字、英文字符或连字符（-）。
Envoy流控domain	流控服务的域。您可以随意设置流控服务的域，与网格内服务域名没有关联。
集群预计QPS	该集群内需要流控的接口预估的最大QPS（每秒查询率），代表可能到来的最大流量，而非限流阈值，用于为Token Server自动分配提供参考。 **说明** 设置的集群预计QPS是用于AHAS进行Server分配，不影响任何限流的实质逻辑。

5. 单击**下一步**。

步骤三：接入AHAS限流服务

1. 在Mesh集群配置页面单击**Mesh流控接入**。
2. 在**Istio接入**页签复制页面中的代码，保存页面中的YAML配置文件。

> **说明** 如果 generic_key 中 descriptor_value 的值为 AHASundefined ，则说明AHAS限流服务尚未准备就绪，请等待并刷新页面，直到该值变化后再进行这一步操作。

3. 设置EnvoyFilter。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
   iv. 在**控制平面**区域，选择**EnvoyFilter**页签，然后单击**新建**。
   v. 在**新建**页面，设置**命名空间**为**istio-system**，在文本框中拷贝步骤中保存的YAML配置文件，单击**确定**。

在网格管理详情页面**基本信息**区域的**流量控制**显示**已接入**，表示已接入AHAS限流服务。若**流量控制**显示**点击接入**，表示未接入AHAS限流服务。

步骤四：添加流控规则

为了便于触发流控，本示例中添加一个0.1QPS的流控规则，即10 s内只允许一次访问。

1. 在Mesh集群配置页面**集群规则配置**区域单击**添加**。
2. 设置流控规则。

参数	描述
规则名称	用于标识该规则，不超过64个字符，规则名称不能重复。
阈值	集群阈值，满足流控条件的请求量到达该阈值后会被拒绝。本例中设置为1。
统计窗口时长	集群流控统计的时间窗口长度，取值范围为1秒~24小时。本例中设置为10秒
匹配规则	流控针对的请求属性的匹配规则，如针对不同来源的请求分别限制不同的规则。单击**添加匹配规则**，设置**匹配规则类型**为**来源集群**，设置**针对方式**为**针对具体值**，设置**针对值**为productpage.default。单击**新增**。 **说明** 针对值的命名规则为{Pod中labels app的值}.{命名空间名称}，例如productpage.default。
是否开启	打开流控开关。

3. 单击**保存**。

步骤五：触发限流规则

根据代码逻辑Productpage应用会在收到访问请求时调用Reviews和Details应用，这两个调用几乎同时发出。按照0.1QPS的限制，这两个调用会有一个失败。通过服务网格ASM控制台获得入口网关地址，并在浏览器中输入*http://{入口网关地址}/productpage*，访问测试应用。可以看到productpage页面Reviews区域显示调用失败，限流规则成功生效。