应用服务网格(ASM)

最佳实践

文档版本 05

发布日期 2023-05-12

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 数据面 sidecar 升级不中断业务

应用服务网格作为集群网络管理的重要工具，为网格内的服务提供流量治理与流量监控的能力。sidecar作为应用服务网格数据面的重要组件，需要依赖服务业务pod的更新来实现sidecar的升级和重新注入。

本章节主要介绍如何实现数据面sidecar升级过程中，不中断服务的业务流量。

配置服务实例数

为保证您的服务在sidecar升级的过程中不中断业务流量，首先确保您的服务实例数大于等于2，升级策略为滚动升级（rollingUpdate）。

相关滚动升级策略如下，供参考：

kubectl get deploy nginx -n *namespace_name* **-oyaml | grep strategy -a10**

```
spec:
  minReadySeconds: 2
  progressDeadlineSeconds: 600
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
      version: v1
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
```

配置项说明：

- 服务实例数：deployment.spec.replicas >= 2
- 升级策略：deployment.spec.strategy.type == RollingUpdate
- 滚动升级最小存活实例数：deployment.spec.replicas - deployment.spec.strategy.maxUnavailable > 0

添加 readiness 探针

添加readiness探针，可以保证您的新实例pod在真正准备就绪时，才开始接管业务流量。这就避免了在新的实例pod未启动时，接管业务流量造成的访问不通问题。

相关配置如下：

kubectl get deploy nginx -n *namespace_name* **-oyaml | grep readinessProbe -a10**

```
      readinessProbe:
        failureThreshold: 3
        httpGet:
          path: /
          port: 80
          scheme: HTTP
        initialDelaySeconds: 1
        periodSeconds: 10
        successThreshold: 1
        timeoutSeconds: 1
```

配置项说明：

readiness探针配置项：deployment.spec.template.spec.containers[i].readinessProbe

其中，包括探针检查初始时间，检查间隔，超时时间等配置。

设置服务就绪时间

服务就绪时间，minReadySeconds：用于标识pod的ready时间至少保持多长时间，才会认为服务是运行中。

相关配置如下：

kubectl get deploy nginx -n *namespace_name* **-oyaml | grep minReadySeconds -a1**

```
spec:
  minReadySeconds: 2
  progressDeadlineSeconds: 600
```

配置项说明：

服务就绪时间：deployment.spec.minReadySeconds，可根据您业务的实际情况确定。

配置优雅关闭时间

terminationGracePeriodSeconds，优雅关闭时间。在滚动升级过程中，首先会移除旧的服务实例pod的endpoint，并将实例pod的状态置为Terminating，这时K8S会发送SIGTERM信号给pod实例，并等待优雅关闭时间后，将pod强制终止。您可以利用这段时间，处理未完成的请求：

kubectl get deploy nginx -n *namespace_name* **-oyaml | grep terminationGracePeriodSeconds -a1**

```
      securityContext: {}
      terminationGracePeriodSeconds: 30
      tolerations:
```

配置项说明：

优雅关闭时间：deployment.spec.template.spec.terminationGracePeriodSeconds，默认值为30s，可根据业务诉求适当调整。

配置 preStop

preStop会在实例pod中止前调用，可以用来实现业务pod的优雅关闭。此处需要根据业务诉求来进行相应的配置，以Nginx为例。

kubectl get deploy nginx -n *namespace_name* **-oyaml | grep lifec -a10**

```
    lifecycle:
      preStop:
        exec:
          command: ["/bin/sh", "-c", "nginx -s quit; sleep 10"]
```

在“lifecycle”下的“preStop”中，定义了一个命令**nginx -s quit; sleep 10**，这个命令首先会发送一个优雅关闭信号给Nginx进程，然后暂停10秒。这样，在Pod终止之前，Nginx会有足够的时间完成正在处理的请求并优雅地关闭。

需要注意的是，sleep 10中的10秒是一个示例值，您可以根据实际需要和应用程序的性能来调整这个值，关键是为Nginx提供足够的时间来优雅地关闭。

类似地，您可以选择运行自定义命令或自定义脚本，来优雅关闭您的服务进程。

2 面向 Dubbo 协议的服务治理

2.1 简介

Dubbo作为一种特有协议，需要有如下支持：

- 网格服务数据面Envoy支持对Dubbo协议的解析和流量管理。
- 网格控制面支持对Dubbo治理规则的配置。支持灰度发布、负载均衡、访问授权等服务管理。

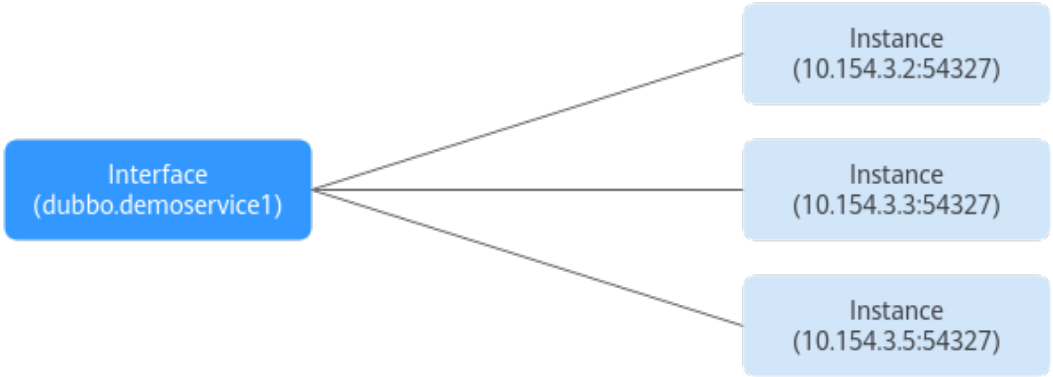
另外，Dubbo的服务发现模型和Kubernetes、Spring Cloud等服务发现模型不一致，需要额外的处理。

2.2 服务发现模型

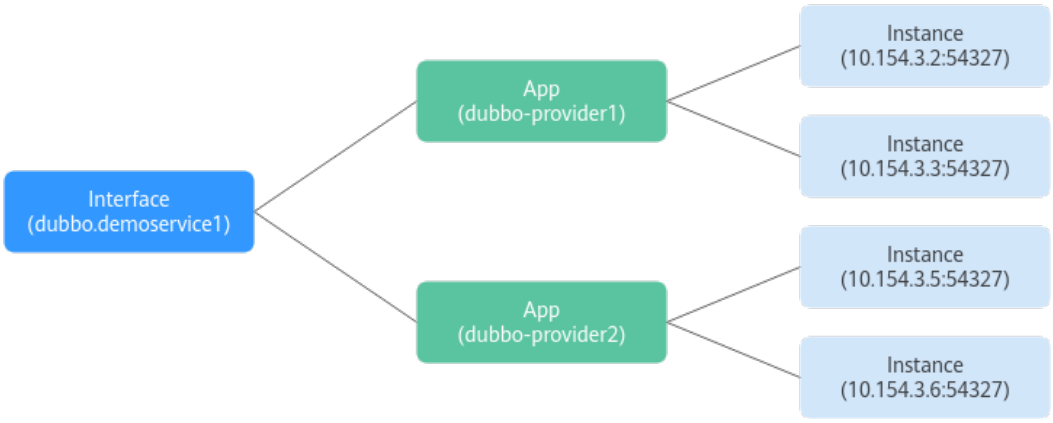
Dubbo现有模型存在的问题（来自Dubbo社区2.7.4总结）：

- 在微服务架构中，注册中心管理的对象是应用（服务），而非对外的服务接口，不过目前Dubbo的注册中心管理的对象是Dubbo服务接口，与Spring Cloud或Cloud Native注册方式背道而驰。
- 一个Dubbo应用（服务）允许注册N个Dubbo服务接口，当N越大时，注册中心的负载越重。

Dubbo现有服务模型：根据Dubbo接口查找服务实例。



Dubbo Cloud Native服务发现模型，将原来Interface一级的服务发现拆分成两级，基于App找实例地址。

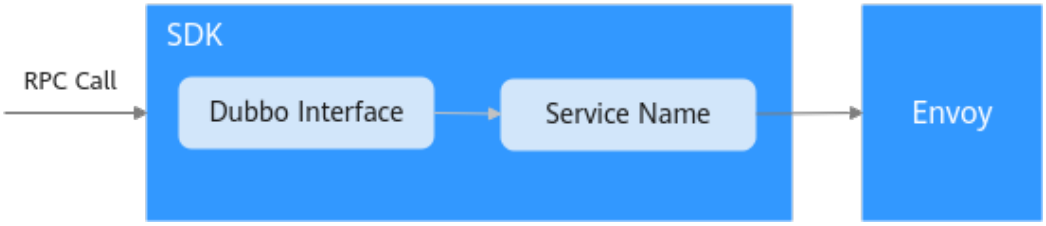


2.3 SDK 适配方式

2.3.1 PASSTHROUGH 方案

方案介绍

SDK中客户端使用Interface调用目标服务时，修改原有服务发现逻辑。将原有通过Interface查找服务实例，修改为通过接口查找服务名，直接对服务名发起访问。



详细说明

对于Dubbo协议的不同版本会有不同：

- 2.7.4+版本：2.7.4以上的Cloud Native版本中重构过与Kubernetes一致的服务发现模型，直接可以从Interface关联到服务信息。
- 2.7.3及之前版本：Dubbo社区版本未提供Interface到Service的二级关系，需要SDK根据自己的实际使用方式来维护Interface到服务的映射关系。如可以使用在服务注册时通过扩展信息的方式提供服务名等信息。

客户实际使用中可以根据自己的SDK使用情况选择灵活的处理方式。对于老版本的SDK可以基于现有的服务注册和服务发现流程，做如下处理：

1. 在注册信息中扩展Service的定义。在服务部署时会通过环境变量将服务的元数据信息注入到SDK中，包括appname、namespace，分别表示部署的服务名、部署的命名空间。
2. 在服务启动时向注册中心注册Dubbo接口和Kubernetes服务名和命名空间的关系。

3. 在客户端发起访问时，根据原有服务发现的流程根据Interface查询到服务的元数据，并用对应的服务信息组装RPC请求。建议在Dubbo请求中使用Attachment扩展字段存储appname、namespace信息。

2.3.2 静态目标服务

方案介绍

通过**dubbo:reference**在Dubbo服务的服务消费者中对引用的服务提供者进行配置。使用选项url定义点对点直连服务提供者地址，绕过注册中心，直接调用目标服务。

详细说明

如果原Dubbo服务中使用的是xml配置文件，则只需要修改配置文件即可。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <!-- 提供哪些接口给消费者调用 -->
    <dubbo:reference id="helloService " interface="com.dubbo.service.HelloService " url = "dubbo://
helloService:20880" />
</beans>
```

如果服务中使用注解的方式来定义引用的目标服务，则需要修改代码中对目标服务的注解。

```
@Reference(url = "dubbo://helloService:20880")
HelloService helloService;
```

3 网关访问保留源 IP

操作场景

服务通过网关访问时，默认情况下，目标容器中看到的不是客户端的源IP，如果需要保留源IP，请参考本节指导操作。

配置方法

请在CCE控制台“服务发现”页面，istio-system命名空间下，更新服务所关联的网关服务，将服务亲和改成“节点级别”。前提是已开启ELB的获取客户端IP功能（当前为默认开启）。

externalTrafficPolicy：表示此Service是否希望将外部流量路由到节点本地或集群范围的端点。有两个可用选项：Cluster（默认）和Local。Cluster隐藏了客户端IP，可能导致第二跳到另一个节点，但具有良好的整体负载分布。Local保留客户端源IP并避免LoadBalancer和NodePort类型服务的第二跳，但存在潜在的不均衡流量传播风险。

验证方式

结合httpbin镜像在“x-forward-for”字段中可以看到源IP，httpbin是一个HTTP Request & Response Service，可以向他发送请求，他将会按照指定的规则将请求返回。httpbin镜像可在SWR中搜索。使用httpbin镜像进行验证时请确保集群已开通网格。

1. 登录ASM应用服务网格控制台，选择一个可用的测试网格并单击进入。
2. 选择左侧“网格配置”查看其关联的集群。

3. 单击集群名称进入集群详情页，单击对应集群右上角第三个图标“工作负载”进入“工作负载”页签。

单击右上角“创建负载”按钮。

4. 配置工作负载的信息。

基本信息

- 负载类型：选择无状态工作负载Deployment。
- 负载名称：命名工作负载为httpbin。
- 命名空间：选择工作负载的命名空间，默认为default。
- 其余参数使用默认值。

容器配置

- 基本信息：
  - 容器名称：为容器自定义一个名称
  - 镜像名称：单击后方“选择镜像”，在右上角搜索框搜索“httpbin”镜像并选择，单击“确认”。
  - 镜像版本：选择一个镜像版本
  - 其余参数使用默认值。

服务配置

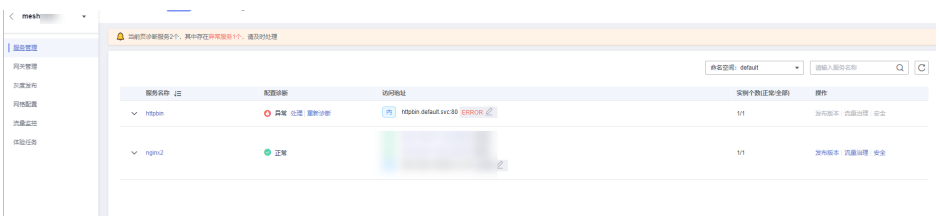
服务（Service）是用来解决Pod访问问题的。每个Service有一个固定IP地址，Service将访问流量转发给Pod，而且Service可以给这些Pod做负载均衡。

单击服务配置参数下面的“+”进入创建服务页面。

- Service名称：Service名称为工作负载名称。
- 访问类型：选择集群内访问。
- 端口设置：
  - 协议：TCP
  - 容器端口：80（以实际访问端口为准）
  - 服务端口：80（以实际访问端口为准）

5. 单击右下角“确定”完成服务创建。
6. 单击右下角“创建工作负载”完成工作负载创建。
7. 在集群详情页选择左侧“服务发现”页签，可在服务列表中查看到所创建的httpbin服务。

8. 返回ASM应用服务网格，选择左侧“服务管理”页签，在服务管理中可查看到httpbin的配置诊断显示为异常。



9. 单击此服务配置诊断中的“处理”按钮，按照弹出“配置诊断”页面对应的修复指导进行修复。

- 以“Service的端口名称是否符合Istio规范”为例，选择协议为“http”，单击“一键修复”。

10. 选择左侧“网关管理”页签，单击右上角“添加网关”，在弹出“添加网关”页面输入配置信息。

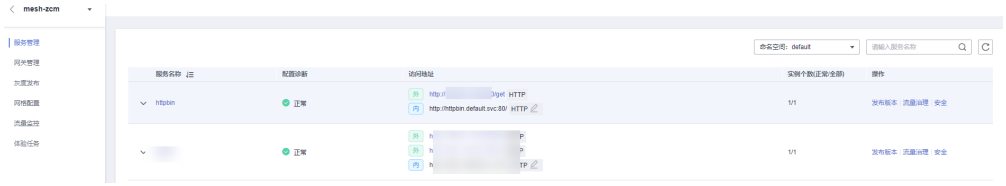
配置信息

- 网关名称：httpbin。
- 集群选择：网格所关联使用的集群。
- 负载均衡：选择公网、选择一个elb公网。
- 对外协议：选择HTTP。
- 对外端口：自定义端口。
- 外部访问地址：负载均衡中所选elb公网地址。

- 单击“路由配置”下方的“+”在弹出“添加路由”页面添加路由。
  - URL：选择完全匹配并输入映射。
  - 命名空间：服务所在命名空间。
  - 目标服务：默认设置。

配置完成后单击“确定”。

11. 单击“确定”完成网关添加。
12. 选择左侧“服务管理”页签，可以在“访问地址”查看到所创建路由的外部访问地址。



13. 在映射的外部访问地址后加上“?show_env=1”，访问添加字段后的外部访问地址，例如：http://xxx.xxx.xxx:80/get?show_env=1。可以在“x-forward-for”字段中查看网关获取的IP为容器段IP。

```
...
...
...
x-forward-for: xxxx
```

14. 返回集群详情页，选择左侧导航栏“服务发现”，更改服务所关联的网关服务的配置。方法如下：

    下拉上方“命名空间”列表选择“istio-system”。

展开服务后方“更多”选项，单击“更新”，在弹出“更新服务”页面将“服务亲和”更改为“节点级别”，勾选“我已阅读《负载均衡使用须知》”，单击“确定”。

15. 返回**13**中访问的外部地址并刷新，若设置之后“x-forward-for”字段中显示的网关获取IP的结果为本机源IP，则完成验证。

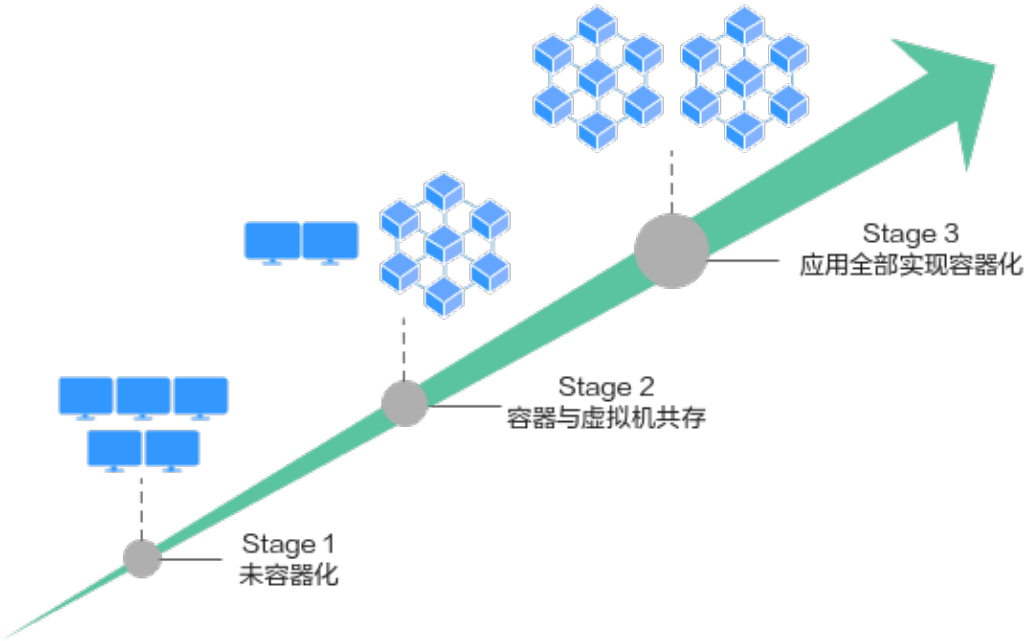
```
...
...
...
x-forward-for: xxxx
```


4 虚拟机治理

4.1 方案介绍

应用现状

将应用迁移到云原生架构，不断容器化的过程中，一般会经历三个阶段，如下图所示。



- 阶段一：所有应用都部署在虚拟机上。
- 阶段二：应用既部署在虚拟机上也部署在容器中，正在从虚拟机向容器迁移，并且使用Kubernetes管理容器。
- 阶段三：所有应用都部署在容器中，使用Kubernetes管理容器，并且使用Istio管理应用间的通信。

因为种种原因，容器与虚拟机共存将是一个长期的过程，但容器化的趋势不变。

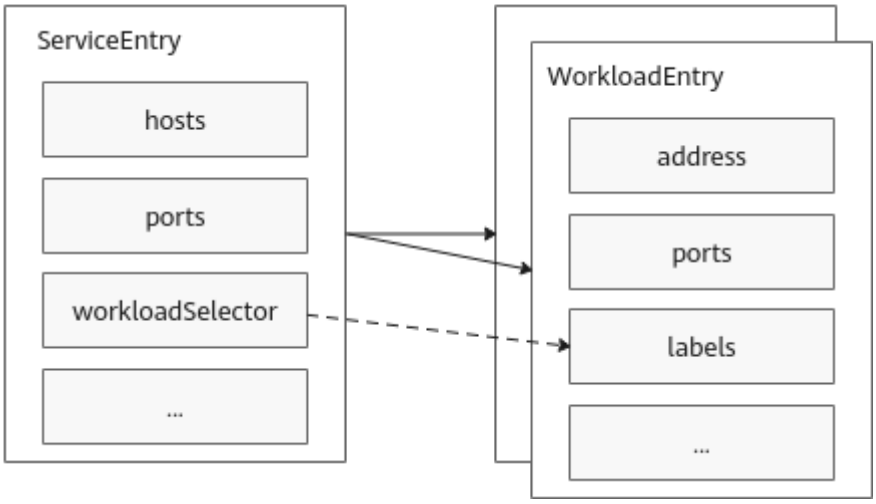
在阶段二中，大量的虚拟机应用并不能很快容器化，人们通常会将新业务和少量应用率先实现容器化，并部署到Kubernetes中。在应用尚未完全实现容器化，处于过渡阶段时会遇到诸多问题，比如虚拟机上的非容器化服务是否可以像容器化服务一样进行流量治理？虚拟机如何访问容器中的服务？是否可以将容器和虚拟机纳入一个统一的控制平面来管理？

本文介绍一种通过为虚拟机安装代理来实现虚拟机治理的方案，可以有效解决以上几个问题。

解决方案

原理上网格可以提供对容器和虚拟机同等的治理能力，不同于Kubernetes容器上网格的数据面Proxy可以自动注入，自动流量拦截，基于Kubernetes的服务和实例模型自动进行服务发现，在虚拟机场景下需要手动安装数据面Proxy（ASM-PROXY），在安装过程中初始化流量拦截。具体来说，ASM-PROXY是虚拟机部署工具包，用于连接虚拟机应用所在的数据平面与ASM提供的控制平面。ASM-PROXY部署在虚拟机节点中，负责与ASM通信获取xDS信息，拦截非容器应用流量并执行网格化操作，例如流量管理、请求安全认证、上报链路追踪数据等。

在服务发现上，虚拟机形态的服务通过WorkloadEntry来描述一个虚拟机上安装了Proxy的实例，该虚拟机实例可以和Pod一样通过标签选择器被ServiceEntry选中和管理。

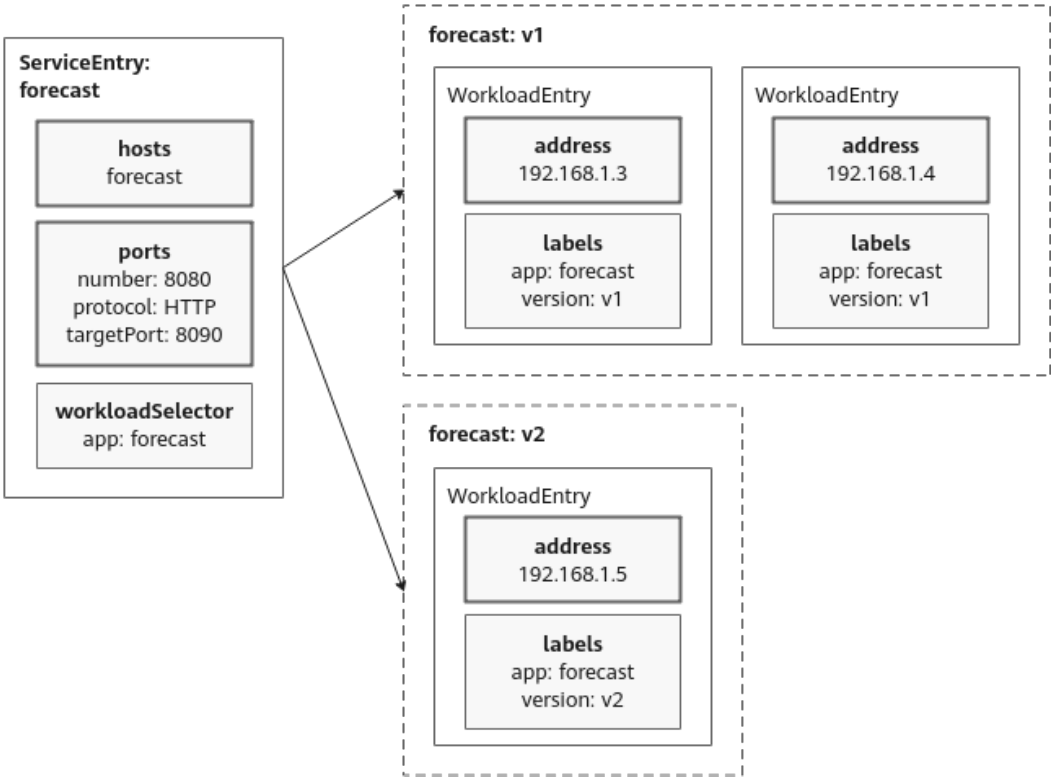


例如，ServiceEntry和三个WorkloadEntry的YAML文件如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: forecast
spec:
  hosts:
  - forecast
  location: MESH_INTERNAL
  ports:
  - number: 8080
    name: http
    protocol: HTTP
    targetPort: 8090
  resolution: STATIC
  workloadSelector:
    labels:
      app: forecast
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
```

```
metadata:
  name: forecast
spec:
  address: 192.168.1.3
  labels:
    app: forecast
    version: v1
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: forecast
spec:
  address: 192.168.1.4
  labels:
    app: forecast
    version: v1
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: forecast
spec:
  address: 192.168.1.5
  labels:
    app: forecast
    version: v2
```

那么ServiceEntry选择WorkloadEntry的机制可概括如下：



4.2 约束与限制

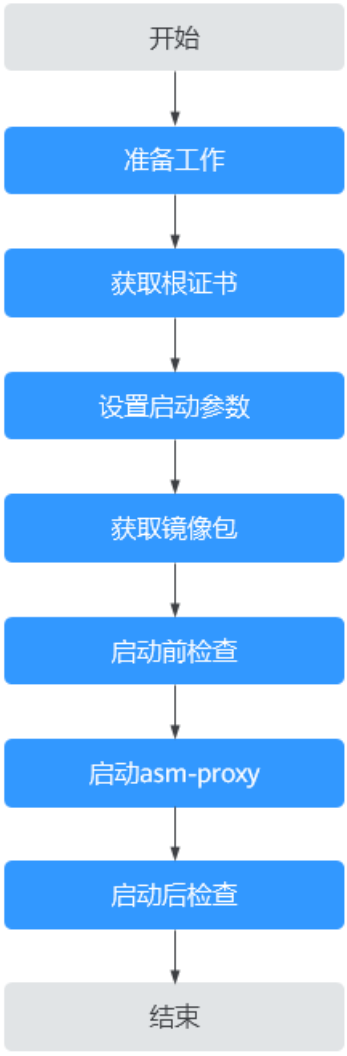
- 仅1.8 企业版 网格支持虚拟机治理，其他类型不支持，本文档后续章节均以1.8.4 网格版本为例。
- 支持v1.19及以上版本集群。

- 支持扁平和非扁平网络架构。
- 如果是多集群，需要集群在同一个VPC，否则虚拟机与其他VPC集群网络不通。
- 不支持容器隧道网络集群。
- 不支持虚拟机和容器实例属于同一个服务。

4.3 部署 ASM-PROXY

为了兼容不同虚拟机操作系统的版本差异，ASM-PROXY以Docker镜像的方式进行部署。部署流程如图4-1所示。

图 4-1 ASM-PROXY 部署流程



准备工作

- 已购买CCE集群，已在集群中创建虚拟机服务使用的命名空间（假设为vmns），已在该命名空间创建容器服务。具体操作请参见**购买CCE集群**。
- 已购买企业版网格，并添加集群到网格，虚拟机服务使用的命名空间已开启sidecar注入。具体操作请参购买网格、**添加集群**和**sidecar管理**。

- 准备两台已安装Docker的ECS虚拟机，并满足以下要求：

要求	操作方法
创建相应目录及文件	在ECS虚拟机上执行以下命令： **mkdir -p /opt/asm_proxy/certs/** **touch /opt/asm_proxy/asm_proxy.env**
备份虚拟机iptables规则	在ECS虚拟机上执行以下命令： **iptables-save > ~/iptables.backup**

⚠ 注意

请确保网络三层连通性：

- 为了保证虚拟机与CCE集群的Pod互通，准备的虚拟机需要与CCE集群处于同一VPC内。
- 虚拟机需放通安全组，入方向需添加到<cluster-name>-cce-control安全组的规则，否则ping不通Pod IP。

获取根证书

步骤1 通过kubectl访问CCE集群。

1. 登录云容器引擎控制台，在“集群管理”页面单击集群名称，进入集群详情页。
2. 在“连接信息”区域找到kubectl，单击查看后，按照教程为您的客户端机器配置kubectl，并访问CCE集群。

说明

假设配置了kubectl的客户端机器称为kubectl节点，下文将一律使用kubectl节点进行描述。

步骤2 在kubectl节点执行以下命令，创建临时工作目录。

export VM1_DIR="<a working directory for vm1>"

mkdir -p ${VM1_DIR}

步骤3 在kubectl节点执行以下命令，创建服务账号。

export NS=<vmns>

export SERVICE_ACCOUNT=<vmsa>

kubectl create serviceaccount "${SERVICE_ACCOUNT}" -n "${NS}"

其中，NS为虚拟机要加入的命名空间，SERVICE_ACCOUNT为虚拟机使用的服务账号。

步骤4 获取证书密钥文件。

1. 在kubectl节点执行以下命令，确认集群是否已开启TokenRequest API。

   kubectl get --raw /api/v1 | grep "serviceaccounts/token"

如果没有返回，则需要参考**Service Account Token Volume Projection**开启该API。

2. 在kubectl节点执行以下命令，获取证书密钥文件。

   export tokenexpiretime=$((3600*24*365))

   1、echo '{"kind":"TokenRequest","apiVersion":"authentication.k8s.io/v1","spec":{"audiences":["istio-ca"],"expirationSeconds":'$tokenexpiretime'}}' | kubectl create --raw /api/v1/namespaces/vmns/serviceaccounts/vmsa/token -f -

   其中，3600*24*365表示1年有效期，31536000s。

   3.复制token字段的值，在kubectl节点执行以下命令，将密钥保存在istio-token文件中。

   echo -n "{token字段的值}" > ${VM1_DIR}/istio-token

步骤5 在kubectl节点执行以下命令，获取根证书文件。

kubectl get secret -n istio-system cacerts -o yaml |grep "root-cert.pem" | head -1 | cut -d ":" -f 2 | sed s/[[:space:]]//g | base64 -d > ${VM1_DIR}/root-cert.pem

----结束

设置启动参数

步骤1 在kubectl节点执行以下命令，获取相关参数并写入${VM1_DIR}目录下的asm_proxy.env文件中。

1. 获取ASM控制面通信地址
   ```
   ISTIO_PILOT_ADDR=$(kubectl get pod -n istio-system -o jsonpath='{.items[2].status.podIP}')
   echo "ISTIO_PILOT_ADDR=${ISTIO_PILOT_ADDR}" > ${VM1_DIR}/asm_proxy.env
   ```
2. 获取集群ID
   ```
   ASM_CLUSTER_ID=$(kubectl get deploy -n istio-system -o yaml | grep -A1 "ISTIO_META_CLUSTER_ID" | tail -1 | awk -F':' '{print $NF}' | awk '$1=$1')

   echo "ISTIO_META_CLUSTER_ID=${ASM_CLUSTER_ID}" >> ${VM1_DIR}/asm_proxy.env
   ```
3. 获取网格ID
   ```
   ASM_MESH_ID=$(kubectl get deploy -n istio-system -o yaml | grep -A1 "ISTIO_META_ASM_MESH_ID" | tail -1 | awk -F':' '{print $NF}' | awk '$1=$1')

   echo "ISTIO_META_MESH_ID=${ASM_MESH_ID}" >> ${VM1_DIR}/asm_proxy.env
   ```
4. 获取监控服务端地址
   ```
   ZIPKIN_ADDRESS=$(kubectl get iop -n istio-system -o yaml | grep -A1 "zipkin" | sed -n '2p')
   TRACING_ZIPKIN_ADDRESS=$(echo ${ZIPKIN_ADDRESS#*:})

   echo "TRACING_ZIPKIN_ADDRESS=${TRACING_ZIPKIN_ADDRESS}" >> ${VM1_DIR}/asm_proxy.env
   ```
5. 获取虚拟机要加入的命名空间
   ```
   echo "NS=${NS}" >> ${VM1_DIR}/asm_proxy.env
   ```
6. 获取虚拟机使用的服务账号
   ```
   echo "SERVICE_ACCOUNT=${SERVICE_ACCOUNT}" >> ${VM1_DIR}/asm_proxy.env
   ```

步骤2 打开kubectl节点的${VM1_DIR}/asm_proxy.env文件，填写以下变量，保存文件。

```
# 集群服务网段，从ASM控制台“网格配置 > 基本信息”页面集群列表中获取
ISTIO_SERVICE_CIDR=x.x.x.x/x

# 虚拟机使用的Pod名称，自定义
POD_NAME=<vmapp1>
# 不指定代理云平台
CLOUD_PLATFORM=none
```

步骤3 执行以下命令检查文件内容。

cat ${VM1_DIR}/asm_proxy.env

正常回显如下：

```
ISTIO_PILOT_ADDR=10.252.2.36
ISTIO_META_CLUSTER_ID=cluster-vm
ISTIO_META_MESH_ID=e256c79c-8b13-11ec-94e9-0255ac10069a
TRACING_ZIPKIN_ADDRESS=otel-collector.monitoring.svc.cluster.local:9411
NS=vmns
SERVICE_ACCOUNT=vmsa
ISTIO_SERVICE_CIDR=10.233.0.0/16
POD_NAME=vmapp1
```

步骤4 将kubectl节点${VM1_DIR}目录下的asm_proxy.env文件上传到虚拟机/opt/asm_proxy/目录下。

在root下执行：

scp ${VM1_DIR}/asm_proxy.env root@${*虚拟机ip*}:/opt/asm_proxy/

步骤5 将kubectl节点${VM1_DIR}目录下的istio-token、root-cert.pem文件上传到虚拟机/opt/asm_proxy/certs目录下。

在root下执行：

scp ${VM1_DIR}/istio-token root@${*虚拟机ip*}:/opt/asm_proxy/certs

scp ${VM1_DIR}/root-cert.pem root@${*虚拟机ip*}:/opt/asm_proxy/certs

----结束

获取镜像包

步骤1 以root用户登录虚拟机。

步骤2 获取asm-proxy-1.8.4镜像包。

asm-proxy-1.8.4镜像包存放在SWR镜像仓库中，使用如下命令拉取镜像（x86和arm架构的命令相同）。

docker pull SWR镜像仓库地址**/istio/asm-proxy:1.8.4**

说明

SWR镜像仓库地址的获取方式如下：

在容器镜像服务控制台“总览”页面单击右上角的“登录指令”，弹出“登录指令”对话框，指令末尾的域名即为SWR镜像仓库地址，形式为swr.***.com。

----结束

启动前检查

步骤1 以root用户登录虚拟机。

步骤2 检查证书文件信息。

ls -l /opt/asm_proxy/certs/

以下为预期结果：

```
-rw------- 1 root root  757 Jan 24 10:41 istio-token
-rw------- 1 root root 1789 Jan 24 10:41 root-cert.pem
```

步骤3 检查启动参数文件信息。

cat /opt/asm_proxy/asm_proxy.env

以下为预期结果（具体值会不同，确保不为空）：

```
ISTIO_PILOT_ADDR=10.252.2.36
ISTIO_META_CLUSTER_ID=cluster-vm
ISTIO_META_MESH_ID=e256c79c-8b13-11ec-94e9-0255ac10069a
TRACING_ZIPKIN_ADDRESS=otel-collector.monitoring.svc.cluster.local:9411
NS=vmns
SERVICE_ACCOUNT=vmsa
ISTIO_SERVICE_CIDR=10.233.0.0/16
POD_NAME=vmapp1
```

----结束

启动 asm-proxy

步骤1 执行以下命令，启动asm-proxy容器。

```
docker run -d \
  --name=asm-proxy \
  --network=host \
  --restart=always \
  --env-file /opt/asm_proxy/asm_proxy.env \
  --cap-add=NET_ADMIN \
  --volume=/opt/asm_proxy/certs:/opt/istio/certs \
  SWR镜像仓库地址/istio/asm-proxy:1.8.4
```

其中，“SWR镜像仓库地址/istio/asm-proxy:1.8.4”为asm-proxy镜像包地址，请参考**获取镜像包**获取。

步骤2 若虚拟机为arm架构，还需要执行以下步骤：

1. 执行**docker ps | grep asm-proxy**，查看asm-proxy容器的ID。
   返回示例如下：
   ```
   fad1ab9530fc        asm-proxy:1.8.4     "/usr/bin/bash /usr/…"   24 h
   ```
2. 执行如下命令从容器中拷贝二进制文件。
   docker cp *fad1ab9530fc*:**/usr/local/bin/pilot-agent /usr/local/bin**
   其中，fad1ab9530fc为上一步获取的容器ID。
3. 为二进制文件赋予可执行权限。
   chmod 755 /usr/local/bin/pilot-agent
4. 定义环境变量并赋值。
   export ISTIO_SERVICE_CIDR=*x.x.x.x/x*
   其中，x.x.x.x/x为集群服务网段，从ASM控制台“网格配置 > 基本信息”页面集群列表中获取。

5. 设置虚拟机的iptables。

   pilot-agent istio-clean-iptables -m REDIRECT

   pilot-agent istio-iptables -p 15001 -z 15006 -u 1337 -m REDIRECT -i ${ISTIO_SERVICE_CIDR} -x "" -b "*" -d 15020

----结束

启动后检查

步骤1 在虚拟机上执行以下命令，检查istio的envoy进程是否正常。

ps -ef | grep istio

正常回显信息类似如下，envoy进程运行正常。

```
root     27254 27139  0 14:37 ?        00:00:00 su -s /bin/bash -c INSTANCE_IP=10.66.6.88 POD_NAME=vmapp1 POD_NAMESPACE=vmns exec /usr/local/bin/pilot-agent proxy --serviceCluster vmapp1 2> /var/log/istio/istio.err.log > /var/log/istio/istio.log istio-proxy
systemd+ 27714 27707  0 14:40 ?        00:00:00 /usr/local/bin/envoy -c etc/istio/proxy/envoy-rev0.json --restart-epoch 0 --drain-time-s 45 --parent-shutdown-time-s 60 --service-cluster vmapp1 --service-node sidecar~10.66.6.88~vmapp1.vmns~vmns.svc.cluster.local --local-address-ip-version v4 --bootstrap-version 3 --log-format-prefix-with-location 0 --log-format %Y-%m-%dT%T.%fZ?%l?envoy %n?%v -l warning --component-log-level misc:error
root     27745  3521  0 14:50 pts/0    00:00:00 grep --color=auto istio
```

步骤2 检查asm-proxy容器状态正常。

docker ps | grep asm-proxy

如果第5列没有提示Restarting，说明asm-proxy容器启动正常。

```
2bc3ee448c6d    asm-proxy:1.8.4    "/usr/bin/bash /usr/..."    52 seconds ago    Up 51 seconds    asm-proxy
```

启动异常显示如下：

```
root@cluster-        -vm:~# docker ps | grep asm-proxy
3b45cced1236   asm-proxy:1.8.4   "/usr/bin/bash /usr/…"   44 seconds ago   Restarting (1) 13 seconds ago            asm-proxy
root@cluster-        -vm:~#
```

步骤3 检查istio启动正常。

1. 执行以下命令进入asm-proxy容器。

   docker exec -it asm-proxy bash

2. 执行以下命令查看istio启动日志。

   vim /var/log/istio/istio.log

   正常启动日志如下：

```
2022-02-07T09:23:31.651059Z    info    xdsproxy        Envoy ADS stream established
2022-02-07T09:23:31.651156Z    info    xdsproxy        connecting to upstream XDS server: istiod.istio-system.svc:15012
2022-02-07T09:23:31.962409Z    info    sds     resource:ROOTCA new connection
2022-02-07T09:23:31.962484Z    info    sds     Skipping waiting for gateway secret
2022-02-07T09:23:31.962547Z    info    sds     resource:default new connection
2022-02-07T09:23:31.962592Z    info    sds     Skipping waiting for gateway secret
2022-02-07T09:23:31.962706Z    info    cache   adding watcher for file ./etc/certs/root-cert.pem
2022-02-07T09:23:31.962785Z    info    cache   GenerateSecret from file ROOTCA
2022-02-07T09:23:31.963252Z    info    sds     resource:ROOTCA pushed root cert to proxy
2022-02-07T09:23:32.041052Z    error   cache   cannot load key pair from /etc/certs: open /etc/certs/cert-chain.pem: no such file or directory
2022-02-07T09:23:32.055631Z    info    cache   GenerateSecret default
2022-02-07T09:23:32.056570Z    info    sds     resource:default pushed key/cert pair to proxy
```

步骤4 在虚拟机上执行以下命令，确认iptables规则正确配置。

iptables -t nat -L -v

如果回显信息类似如下，说明iptables规则配置成功。

```
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
  106  4240 ISTIO_INBOUND  tcp  --  any    any     anywhere             anywhere

Chain INPUT (policy ACCEPT 106 packets, 4240 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 207 packets, 16621 bytes)
 pkts bytes target     prot opt in     out     source               destination
   50  3000 ISTIO_OUTPUT  tcp  --  any    any     anywhere             anywhere

Chain POSTROUTING (policy ACCEPT 207 packets, 16621 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain ISTIO_INBOUND (1 references)
 pkts bytes target     prot opt in     out     source               destination
0     0 RETURN     tcp  --  any    any     anywhere             anywhere
tcp dpt:15008
    0     0 RETURN     tcp  --  any    any     anywhere             anywhere             tcp dpt:ssh
    0     0 RETURN     tcp  --  any    any     anywhere             anywhere             tcp dpt:15020
  106  4240 ISTIO_IN_REDIRECT  tcp  --  any    any     anywhere             anywhere

Chain ISTIO_IN_REDIRECT (3 references)
 pkts bytes target     prot opt in     out     source               destination
  106  4240 REDIRECT   tcp  --  any    any     anywhere             anywhere             redir ports 15006

Chain ISTIO_OUTPUT (1 references)
 pkts bytes target     prot opt in     out     source               destination
0 ++++++++++++++++++++++++++++++++++++++++++++
    0 RETURN     all  --  any    lo      127.0.0.6            anywhere
    0     0 ISTIO_IN_REDIRECT  all  --  any    lo      anywhere             !localhost            owner UID match 1337
    0     0 RETURN     all  --  any    lo      anywhere             anywhere             ! owner UID match 1337
    0     0 RETURN     all  --  any    any     anywhere             anywhere             owner UID match 1337
    0     0 ISTIO_IN_REDIRECT  all  --  any    lo      anywhere             !localhost            owner GID match 1337
    0     0 RETURN     all  --  any    lo      anywhere             anywhere             ! owner GID match 1337
    0     0 RETURN     all  --  any    any     anywhere             anywhere             owner GID match 1337
    0     0 RETURN     all  --  any    any     anywhere             localhost
    0     0 ISTIO_REDIRECT  all  --  any    any     anywhere             10.228.0.0/16
   50  3000 RETURN     all  --  any    any     anywhere             anywhere

Chain ISTIO_REDIRECT (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 REDIRECT   tcp  --  any    any     anywhere             anywhere             redir ports 15001
```

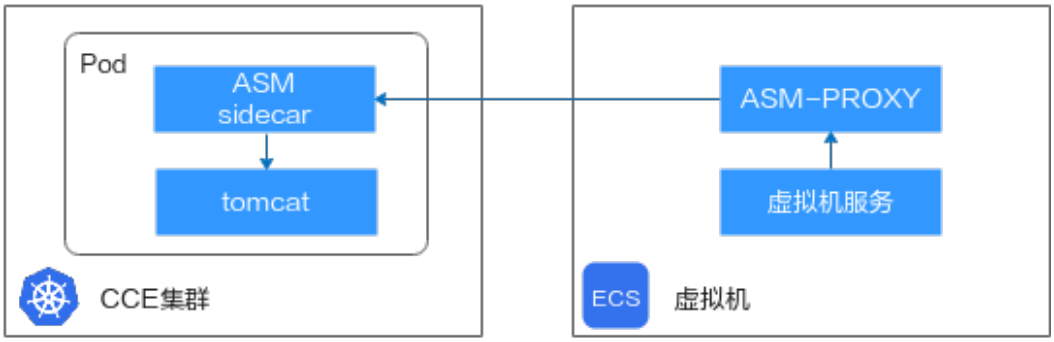
其中，10.228.0.0/16为集群服务网段。

----结束

4.4 验证服务访问

4.4.1 虚拟机服务访问容器服务

启动ASM-PROXY后，虚拟机服务可以访问容器内的服务，如下图所示。



验证流程如下：

1. **部署容器服务**：在CCE集群中部署容器服务tomcat。
2. **容器服务加入网格**：将tomcat服务加入网格，确保服务诊断状态为正常。
3. **访问容器服务**：编辑虚拟机的/etc/hosts文件，添加域名解析，然后通过curl命令访问容器服务。

部署容器服务

步骤1 登录云容器引擎控制台，在“集群管理”页面单击集群名称，进入集群详情页。

步骤2 在左侧导航栏选择“资源 > 工作负载”，单击右上角“镜像创建”。

步骤3 设置工作负载参数。

- 负载类型：选择“无状态负载”。
- 命名空间：选择在**步骤3**中填写的命名空间“vmns”。
- 镜像：选择“tomcat”。
- 服务配置：添加服务，服务端口和容器端口设置为8080。

其余参数按需填写。

说明

如果是TCP协议服务，需要服务监听在127.0.0.1或0.0.0.0。

步骤4 单击“创建工作负载”。

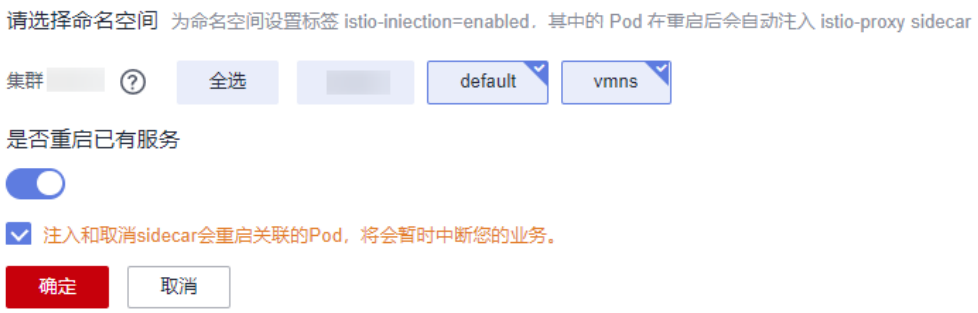
----结束

容器服务加入网格

步骤1 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。

步骤2 在左侧导航栏选择“网格配置”，单击“sidecar管理”页签，为命名空间“vmns”注入sidecar。

图 4-2 注入 sidecar



步骤3 在左侧导航栏选择“服务管理”，选择命名空间“vmns”。

步骤4 按下述指导修复异常状态的tomcat服务。

单击“处理”进入自动修复项页面，单击“一键修复”，自动处理异常状态的检查项，直到配置诊断为正常状态。

图 4-3 服务状态

----结束

访问容器服务

步骤1 获取tomcat服务在集群中的IP地址。

1. 登录云容器引擎控制台，在“集群管理”页面单击集群名称，进入集群详情页。
2. 在左侧导航栏选择“资源 > 工作负载”，选择命名空间“vmns”。
3. 单击工作负载名称“tomcat”，进入详情页面。
4. 单击“访问方式”页签，获取服务在集群中的IP地址。

图 4-4 IP 地址

步骤2 登录虚拟机，执行**vim /etc/hosts**打开本地hosts文件，添加域名解析。

```
<cluster_ip> <tomcat>.<vmns>.svc
```

其中，cluster_ip为**步骤1**中获取的IP地址，tomcat为容器服务名称，vmns为命名空间。

步骤3 执行以下命令，验证访问容器服务是否正常。

curl <tomcat>.<vmns>.svc:8080

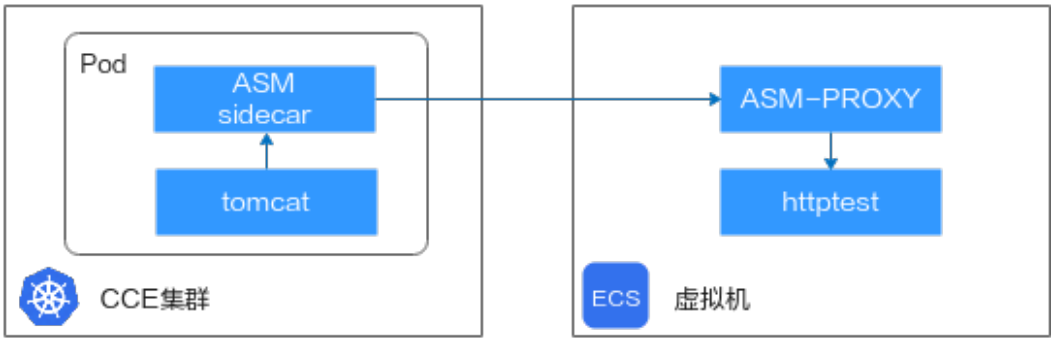
如果回显信息类似如下，说明虚拟机服务访问容器服务正常。

```
root                    :~# curl tomcat.vmns.svc:8080


<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <title>Apache Tomcat/8.5.32</title>
        <link href="favicon.ico" rel="icon" type="image/x-icon" />
        <link href="favicon.ico" rel="shortcut icon" type="image/x-icon" />
        <link href="tomcat.css" rel="stylesheet" type="text/css" />
    </head>

    <body>
        <div id="wrapper">
            <div id="navigation" class="curved container">
                <span id="nav-home"><a href="http://tomcat.apache.org/">Home</a></span>
```

----结束

4.4.2 容器服务访问虚拟机服务

启动ASM-PROXY后，容器内的服务可以访问虚拟机上的服务，如下图所示。



验证流程如下：

1. **部署虚拟机服务**：在虚拟机中部署httptest应用。
2. **部署容器服务**：在CCE集群中部署容器服务tomcat。
3. **添加虚拟机服务到网格**：不同于容器服务有自动注册能力，当把一个虚拟机服务加入网格时，需要用户手动注册服务的基础信息。因此需要手动创建如下资源：Service、WorkloadEntry、ServiceEntry、DestinationRule和VirtualService。
4. **访问虚拟机服务**：容器服务通过服务名访问虚拟机服务。

部署虚拟机服务

此处以httptest测试程序作为示例。httptest为go http应用，监听在8080端口，访问正常返回“hello, http protocol server”消息。

步骤1 上传httptest应用到虚拟机。

步骤2 修改权限。

chmod +x httptest

步骤3 启动httptest应用。

./httptest

----结束

部署容器服务

步骤1 登录云容器引擎控制台，在“集群管理”页面单击集群名称，进入集群详情页。

步骤2 在左侧导航栏选择“资源 > 工作负载”，单击右上角“镜像创建”。

步骤3 设置工作负载参数。

- 负载类型：选择“无状态负载”。
- 命名空间：选择在**步骤3**中填写的命名空间“vmns”。
- 镜像：选择“tomcat”。
- 服务配置：添加服务，服务端口和容器端口设置为8080。

其余参数按需填写。

说明

如果是TCP协议服务，需要服务监听在127.0.0.1或0.0.0.0。

步骤4 单击“创建工作负载”。

----结束

添加虚拟机服务到网格

步骤1 创建Service。

1. 登录云容器引擎控制台，在“集群管理”页面单击集群名称，进入集群详情页。
2. 在左侧导航栏选择“资源 > 服务发现”，选择命名空间“vmns”，单击右上角“YAML创建”。
3. 输入以下内容：

```
apiVersion: v1
kind: Service
metadata:
  name: <vm-server1>
  namespace: "<NS>"
  labels:
    app: <vm-server1>
spec:
  ports:
  - port: 8080
    name: http-<vm-server1>
    targetPort: 8080
  selector:
    app: <vm-server1>
```

其中，<vm-server1>为虚拟机服务名称，<NS>为**步骤3**中填写的命名空间。

示例：


```
apiVersion: v1
kind: Service
metadata:
  name: vm-server1
  namespace: "vmns"
  labels:
    app: vm-server1
spec:
  ports:
  - port: 8080
    name: http-vm-server1
    targetPort: 8080
  selector:
    app: vm-server1
```

4. 单击“确定”完成创建。

步骤2 创建WorkloadEntry。

使用WorkloadEntry定义虚拟机服务的工作负载属性。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：workloadentries”，命名空间选择**步骤3**中填写的命名空间“vmns”，单击“创建”。

   说明

   这里的“vmns”命名空间中必须存在服务在网格中正常使用，否则选择不到该命名空间。你可以将**部署容器服务**中的tomcat服务加入网格，并确保配置诊断通过。

4. 输入以下内容：

```
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: "<vm-server1>"
  namespace: "<NS>"
spec:
  address: "<VM_IP>"
  labels:
    app: <vm-server1>
    version: v1
  serviceAccount: "<SERVICE_ACCOUNT>"
```

其中，<vm-server1>为虚拟机服务名称，<NS>为**步骤3**中填写的命名空间，<VM_IP>为虚拟机的私有IP，<SERVICE_ACCOUNT>为**步骤3**中填写的服务账号名称。

示例：

```
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: "vm-server1"
  namespace: "vmns"
spec:
  address: "10.66
  labels:
    app: vm-server1
    version: v1
  serviceAccount: "vmsa"
```

5. 单击“确定”完成创建。

步骤3 创建ServiceEntry。

使用ServiceEntry将虚拟机服务添加到网格内部维护的服务注册表，以便网格中自动发现的服务可以访问或路由到这些手动指定的服务。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：serviceentries”，命名空间选择**步骤3**中填写的命名空间“vmns”，单击“创建”。
4. 输入以下内容：

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: "<vm-server1>"
  namespace: "<NS>"
spec:
  hosts:
  - <vm-server1>.<NS>.svc.cluster.local
  location: MESH_INTERNAL
  ports:
  - number: 8080
    name: http
    protocol: HTTP
  resolution: STATIC
  workloadSelector:
    labels:
      app: <vm-server1>
```

其中，<vm-server1>为虚拟机服务名称，<NS>为**步骤3**中填写的命名空间。

示例：

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: "vm-server1"
  namespace: "vmns"
spec:
  hosts:
  - vm-server1.vmns.svc.cluster.local
  location: MESH_INTERNAL
  ports:
  - number: 8080
    name: http
    protocol: HTTP
  resolution: STATIC
  workloadSelector:
    labels:
      app: vm-server1
```

5. 单击“确定”完成创建。

步骤4 创建DestinationRule。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：destinationrules”，命名空间选择**步骤3**中填写的命名空间“vmns”，单击“创建”。
4. 输入以下内容：

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: <vm-server1>
  namespace: "<NS>"
spec:
  host: <vm-server1>
  subsets:
  - labels:
```

```
    version: v1
  name: v1
```

其中，<vm-server1>为虚拟机服务名称，<NS>为**步骤3**中填写的命名空间。

示例：

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: vm-server1
  namespace: "vmns"
spec:
  host: vm-server1
  subsets:
  - labels:
      version: v1
    name: v1

```

5. 单击“确定”完成创建。

步骤5 创建VirtualService。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：virtualservices”，命名空间选择**步骤3**中填写的命名空间“vmns”，单击“创建”。
4. 输入以下内容：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: <vm-server1>-8080
  namespace: "<NS>"
spec:
  hosts:
  - <vm-server1>
  http:
  - match:
    - gateways:
      - mesh
      port: 8080
    route:
    - destination:
        host: <vm-server1>.<NS>.svc.cluster.local
        port:
          number: 8080
        subset: v1
```

其中，<vm-server1>为虚拟机服务名称，8080为服务端口，<NS>为**步骤3**中填写的命名空间。

示例：


```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: vm-server1-8080
  namespace: "vmns"
spec:
  hosts:
  - vm-server1
  http:
  - match:
    - gateways:
      - mesh
      port: 8080
    route:
    - destination:
        host: vm-server1.vmns.svc.cluster.local
        port:
          number: 8080
        subset: v1
```

5. 单击“确定”完成创建。

----结束

访问虚拟机服务

步骤1 在kubectl节点执行以下命令，获取tomcat工作负载的Pod名称。

kubectl get pods -n <vmns>|grep tomcat

回显示例：

```
tomcat-75cbb4b948-nzrfs      2/2     Running   0      19h
```

步骤2 执行以下命令，测试访问虚拟机服务。

kubectl exec -it <tomcat-xxx> -n <vmns> -- curl <vm-server1>.<vmns>.svc.cluster.local:8080

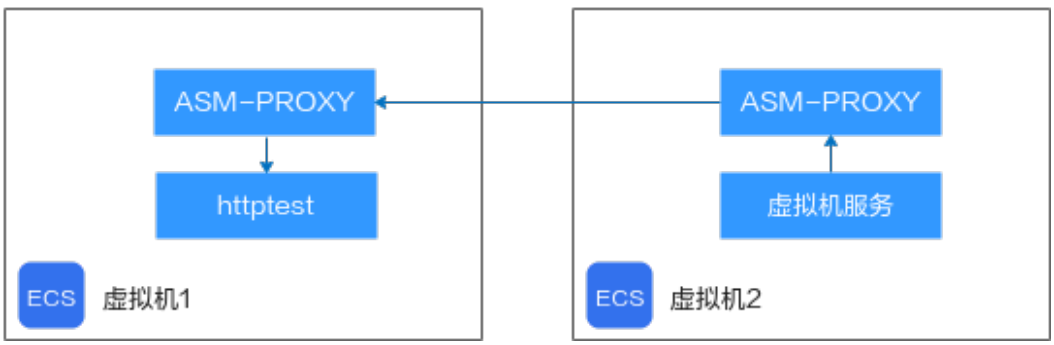
其中，<vm-server1>为虚拟机服务名称，<vmns>为**步骤3**中填写的命名空间。

若回显“hello, http protocol server”，则容器服务访问虚拟机服务成功。

----结束

4.4.3 虚拟机服务访问虚拟机服务

启动ASM-PROXY后，虚拟机服务之间可以互相访问，如下图所示。



验证流程

步骤1 准备两台ECS虚拟机。

虚拟机1请参考**部署ASM-PROXY**部署PROXY，虚拟机2可以不用部署。**但请确保虚拟机间网络正常，且安全组、防火墙已放通。**

步骤2 在虚拟机1上部署httptest应用。

具体操作请参见**部署虚拟机服务**。

步骤3 添加虚拟机1上的服务到网格。

具体操作请参见**添加虚拟机服务到网格**。

步骤4 虚拟机2访问虚拟机1上的服务。

1. 登录虚拟机2，执行**vim /etc/hosts**添加本地域名解析。
   ```
   10.66.xx.xx <vm-server1>.<vmns>.svc.cluster.local
   ```
   其中，10.66.xx.xx为虚拟机1的私有IP，<vm-server1>为虚拟机1上的服务名称，<vmns>为**步骤3**中填写的命名空间。
2. 执行以下命令，访问虚拟机1上的服务。

   curl <vm-server1>.<vmns>.svc.cluster.local:8080

   若回显“hello, http protocol server”，则虚拟机2访问虚拟机1上的服务成功。

----结束

4.5 流量治理

4.5.1 虚拟机服务添加网关和路由

网关（Gateway）定义了在网格出入口操作的负载均衡器，用于接收传入或传出的HTTP/TCP连接。

前提条件

- 已执行**添加虚拟机服务到网格**，即已创建v1版本的WorkloadEntry、ServiceEntry，已创建VirtualService、DestinationRule。
- 已创建负载均衡实例，要求所属VPC为集群所在的VPC。

创建 Loadbalancer 服务

步骤1 登录云容器引擎控制台，在“集群管理”页面单击集群名称，进入集群详情页。

步骤2 在左侧导航栏选择“资源 > 服务发现”，选择命名空间“istio-system”，单击右上角“YAML创建”。

步骤3 输入以下内容：

```
apiVersion: v1
kind: Service
metadata:
  name: <vm-server1>-gw-svc
  namespace: istio-system
  annotations:
    kubernetes.io/elb.class: <ELB_TYPE>
```

```
    kubernetes.io/elb.id: <ELB_ID>
    service.protal.kubernetes.io/type: LoadBalancer
spec:
  ports:
    - name: http
      protocol: TCP
      port: <345>
      targetPort: <2050>
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  type: LoadBalancer
  sessionAffinity: None
  loadBalancerIP: <ELB_IP>
  externalTrafficPolicy: Cluster
```

其中，

- <vm-server1>为虚拟机服务名称
- <ELB_TYPE>为负载均衡类型，可设置为union（共享型）或performance（独享型）
- <ELB_ID>为ELB实例ID
- <345>为服务port
- <2050>为服务targetPort

  如果port小于1025，targetPort需要设置为**未被占用**且范围为1025~65535的值；如果port大于等于1025，则targetPort可直接设置为port值。

  说明

  判断targetPort是否被占用的方法：

  1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
  2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
  3. 选择“istio资源：gateways”，命名空间选择“istio-system”，查看Gateway资源的YAML文件中server的port.number值。

- <ELB_IP>为ELB实例的弹性IP

步骤4 单击“确定”完成创建。

----结束

创建 Gateway 资源

步骤1 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。

步骤2 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。

步骤3 选择“istio资源：gateways”，命名空间选择“istio-system”，单击“创建”。

步骤4 输入以下内容：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  annotations:
    asm/elbPort: '<345>'
    asm/gatewaySvc: <vm-server1>-gw-svc.istio-system
  name: <vm-server1>-gw
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
```



```
servers:
  - hosts:
      - <ELB_IP>
    port:
      name: http
      number: <2050>
      protocol: http
```

其中，<vm-server1>为虚拟机服务名称，<345>为对外端口（同**创建Loadbalancer服务**中的服务port），<ELB_IP>为ELB实例的公网IP或域名，<2050>同**创建Loadbalancer服务**中的服务targetPort。

步骤5 单击“确定”完成创建。

----结束

创建 VirtualService 资源

步骤1 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。

步骤2 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。

步骤3 选择“istio资源：virtualservices”，命名空间选择“istio-system”，单击“创建”，输入以下内容：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: <vm-server1>-root-vs
  namespace: istio-system
spec:
  gateways:
    - istio-system/<vm-server1>-gw
  hosts:
    - <ELB_IP>
  http:
    - delegate:
        name: <vm-server1-8080>
        namespace: <vmns>
      match:
        - uri:
            prefix: /test
    - delegate:
        name: <vm-server1-8080>
        namespace: <vmns>
      match:
        - uri:
            prefix: /
```

其中，<vm-server1>为虚拟机服务名称，<ELB_IP>为ELB实例的公网IP或域名，<vm-server1-8080>同**创建VirtualService**中的VirtualService name，<vmns>同**创建VirtualService**中的VirtualService namespace。

说明

上述http.delegate列表配置了“/”和“/test”两个路由，请根据需要添加或者删除。

步骤4 选择“istio资源：virtualservices”，命名空间选择“istio-system”，单击“创建”再创建一个VirtualService资源，输入以下内容：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: <vm-server1>-mesh-root-vs
  namespace: istio-system
spec:
```



```
hosts:
  - <vm-server1>.<vmns>.svc.cluster.local
http:
  - delegate:
      name: <vm-server1-8080>
      namespace: <vmns>
    match:
      - uri:
          prefix: /test
  - delegate:
      name: <vm-server1-8080>
      namespace: <vmns>
    match:
      - uri:
          prefix: /
```

其中，<vm-server1>为虚拟机服务名称，<vm-server1-8080>同**创建VirtualService**中的VirtualService name，<vmns>同**创建VirtualService**中的VirtualService namespace。

说明

上述http.delegate列表配置了“/”和“/test”两个路由，请根据需要添加或者删除。

----结束

更新 VirtualService 资源

步骤1 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。

步骤2 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。

步骤3 选择“istio资源：virtualservices”，命名空间选择“vmns”，选择**创建VirtualService**中创建的VirtualService，单击操作列的“编辑”，进行如下修改。

1. 移除hosts字段配置
2. 复制spec.http[0]后，在下方并列位置添加如下内容：

   ```
   - match:
       - gateways:
           - istio-system/<vm-server1>-gw
         port: <2050>
     route:
       - destination:
           host: <vm-server1>.<vmns>.svc.cluster.local
           port:
             number: 8080
           subset: v1
   ```

   其中，<vm-server1>为虚拟机服务名称，<2050>同**创建Loadbalancer服务**中的服务targetPort，<vmns>为**步骤3**中填写的命名空间。

   示例：下图红框中为所添加的内容，在结构上需要与spec.http[0]对齐。

```
  name: vm-server1-8080
  namespace: vmns
  resourceVersion: '1488103'
  selfLink: >-
    /apis/networking.istio.io/v1alpha3/namespaces/vmns/virtualservices/vm-server1-8080
  uid: 99c05147-7fec-4876-9401-ab88a2580226
spec:
  http:
    - match:
        - gateways:
            - mesh
          port: 8080
      route:
        - destination:
            host: vm-server1.vmns.svc.cluster.local
            port:
              number: 8080
            subset: v1
    - match:
        - gateways:
            - istio-system/vm-server1-gw
          port: 2050
      route:
        - destination:
            host: vm-server1.vmns.svc.cluster.local
            port:
              number: 8080
            subset: v1
```

步骤4 单击“确定”完成修改。

----结束

说明

如果虚拟机服务需要移除网关，请删除以上创建的资源对象，移除添加的字段即可。

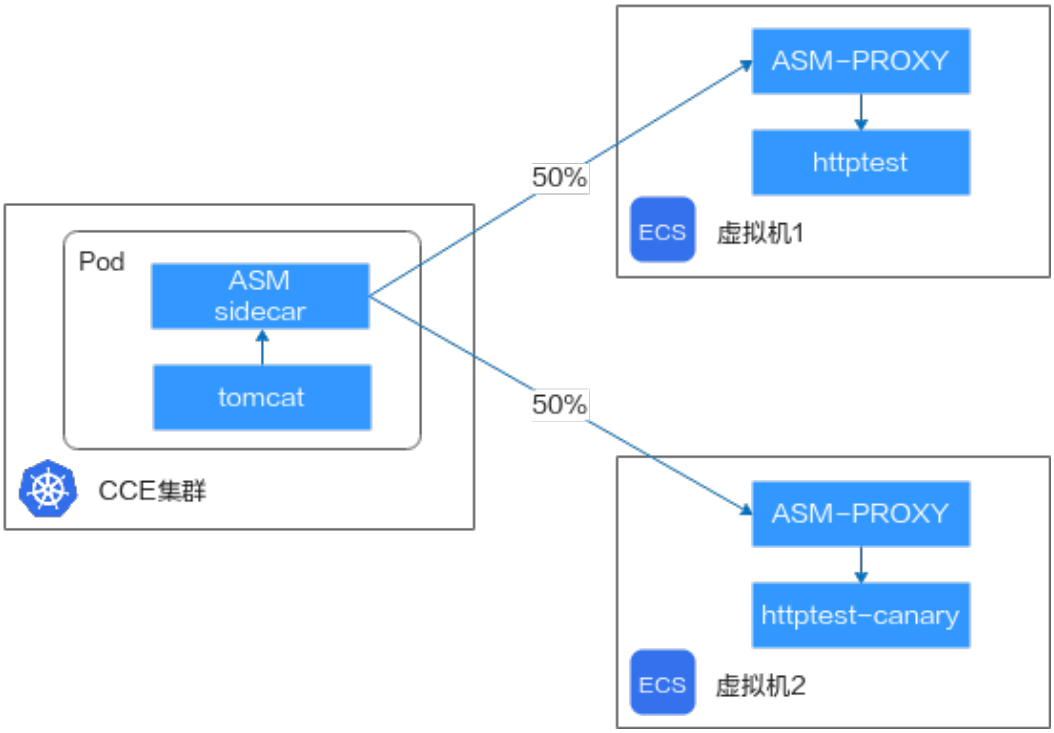
结果验证

登录ASM管理控制台，在“网关管理”页面查看网关状态为“配置正常”，在浏览器打开访问地址，返回“hello, http protocol server”。

图 4-5 网关管理

4.5.2 虚拟机服务灰度发布

虚拟机上部署ASM-PROXY，且网格化后，可以设置灰度策略实现简单的灰度发布。如下图所示，虚拟机1上部署httptest应用（原版本v1），虚拟机2上部署httptest-canary应用（灰度版本v2），配置v1版本、v2版本分别50%的流量比例。当容器服务访问虚拟机服务时，50%的请求将由v1版本响应，50%的请求由v2版本响应。



实现灰度发布的流程如下：

1. **部署虚拟机服务灰度版本**：在虚拟机2上部署httptest-canary应用，作为虚拟机服务灰度版本v2。
2. **添加虚拟机服务灰度版本到网格**：创建v2版本的WorkloadEntry、ServiceEntry，更新VirtualService和DestinationRule，并在其中配置基于流量比例的灰度策略。
3. **访问虚拟机服务**：容器服务通过服务名访问虚拟机服务，验证灰度策略是否生效。
4. **移除虚拟机服务灰度版本**：完成灰度发布后，更新VirtualService和DestinationRule，移除虚拟机服务灰度版本。

前提条件

- 已执行**添加虚拟机服务到网格**，即已创建v1版本的WorkloadEntry、ServiceEntry，已创建VirtualService、DestinationRule。
- 已准备虚拟机2，用来部署虚拟机服务灰度版本。

部署虚拟机服务灰度版本

此处以httptest-canary测试程序作为示例。httptest-canary为go http应用，监听在8080端口，访问正常返回“hello, canary http server”消息。

步骤1 上传httptest-canary应用到虚拟机2。

步骤2 修改权限。

chmod +x httptest-canary

步骤3 启动httptest-canary应用。

./httptest-canary

----结束

添加虚拟机服务灰度版本到网格

步骤1 创建v2版本的WorkloadEntry。

使用WorkloadEntry定义虚拟机服务的工作负载属性。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：workloadentries”，命名空间选择**步骤3**中填写的命名空间“vmns”，单击“创建”。
4. 输入以下内容：

```
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: "<vm-server1>-v2"
  namespace: "<NS>"
spec:
  address: "<VM2_IP>"
  labels:
    app: <vm-server1>
    version: v2
  serviceAccount: "<SERVICE_ACCOUNT>"
```

其中，<vm-server1>为虚拟机服务名称，<NS>为**步骤3**中填写的命名空间，<VM2_IP>为虚拟机2的私有IP，<SERVICE_ACCOUNT>为**步骤3**中填写的服务账号名称。

示例：

```
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: "vm-server1-v2"
  namespace: "vmns"
spec:
  address: "10.66.[redacted]"
  labels:
    app: vm-server1
    version: v2
  serviceAccount: "vmsa"
```

5. 单击“确定”完成创建。

步骤2 更新DestinationRule。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：destinationrules”，命名空间选择“vmns”，选择**创建DestinationRule**中创建的DestinationRule，单击操作列的“编辑”，在spec.subsets下将原有列表复制后并列添加以下内容：

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

示例：

```
  selfLink: >-
    /apis/networking.istio.io/v1alpha3/namespaces/vmns/destinationrules/vm-server1
  uid: 64ae123b-3268-43de-9f8f-59a53798dc94
spec:
  host: vm-server1
  subsets:
    - labels:
        version: v1
      name: v1
    - labels:
        version: v2
      name: v2

```

4. 单击“确定”完成修改。

步骤3 更新VirtualService。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：virtualservices”，命名空间选择“vmns”，选择**创建VirtualService**中创建的VirtualService，单击操作列的“编辑”，区分两种情况：
   - 情况一：虚拟机服务未配置网关

     复制spec.http[0].route下的destination后，在下方并列位置添加以下内容：

     ```
         weight: 50
       - destination:
           host: <vm-server1>.<vmns>.svc.cluster.local
           port:
             number: 8080
           subset: v2
         weight: 50
     ```

     其中，<vm-server1>为虚拟机服务名称，<vmns>为**步骤3**中填写的命名空间。配置v1、v2版本分别50%的流量比例。

     示例：

     ```
       selfLink: >-
         /apis/networking.istio.io/v1alpha3/namespaces/vmns/virtualservices/vm-server1-8080
       uid: 4d653fb8-62bc-4462-a159-07bd8445138d
     spec:
       hosts:
         - vm-server1
       http:
         - match:
             - gateways:
                 - mesh
               port: 8080
           route:
             - destination:
                 host: vm-server1.vmns.svc.cluster.local
                 port:
                   number: 8080
                 subset: v1
               weight: 50
             - destination:
                 host: vm-server1.vmns.svc.cluster.local
                 port:
                   number: 8080
                 subset: v2
               weight: 50
     ```

   - 情况二：虚拟机服务已配置网关

     复制spec.http[0].route下的destination后，在下方并列位置添加以下内容：

     ```
         weight: 50
       - destination:
           host: <vm-server1>.<vmns>.svc.cluster.local
           port:
             number: 8080
     ```

```
      subset: v2
    weight: 50
```

复制spec.http[1].route下的destination后，在下方并列位置添加以下内容：

```
    weight: 50
  - destination:
      host: <vm-server1>.<vmns>.svc.cluster.local
      port:
        number: 8080
      subset: v2
    weight: 50
```

其中，<vm-server1>为虚拟机服务名称，<vmns>为**步骤3**中填写的命名空间。配置v1、v2版本分别50%的流量比例。

示例：

```
16  name: vm-server1-8080
17  namespace: vmns
18  resourceVersion: '1511871'
19  selfLink: >-
20    /apis/networking.istio.io/v1alpha3/namespaces/vmns/virtualservices/vm-server1-8080
21  uid: 99c05147-7fec-4876-9401-ab88a2580226
22 spec:
23  http:
24    - match:
25        - gateways:
26            - mesh
27          port: 8080
28      route:
29        - destination:
30            host: vm-server1.vmns.svc.cluster.local
31            port:
32              number: 8080
33            subset: v1
34          weight: 50
35        - destination:
36            host: vm-server1.vmns.svc.cluster.local
37            port:
38              number: 8080
39            subset: v2
40          weight: 50
41    - match:
42        - gateways:
43            - istio-system/vm-server1-gw
44          port: 2022
45      route:
46        - destination:
47            host: vm-server1.vmns.svc.cluster.local
48            port:
49              number: 8080
50            subset: v1
51          weight: 50
52        - destination:
53            host: vm-server1.vmns.svc.cluster.local
54            port:
55              number: 8080
56            subset: v2
57          weight: 50
```

4. 单击“确定”完成修改。

----结束

访问虚拟机服务

说明

如果虚拟机2未部署ASM-PROXY，未加入网格，要想正常访问到v2版本，需要配置关闭mTLS认证，即在vmns命名空间下的DestinationRule资源对象的spec下添加如下内容：

```
trafficPolicy:
  tls:
    mode: DISABLE
```

```
spec:
  host: vm-server1
  subsets:
    - labels:
        version: v1
      name: v1
    - labels:
        version: v2
      name: v2
  trafficPolicy:
    tls:
      mode: DISABLE
```

步骤1 在CCE集群中部署容器服务tomcat，具体操作请参见**部署容器服务**。

步骤2 在kubectl节点执行以下命令，获取tomcat工作负载的Pod名称。

kubectl get pods -n <vmns>|grep tomcat

回显示例：

```
tomcat-75cbb4b948-nzrfs      2/2     Running   0      19h
```

步骤3 多次执行以下命令，对虚拟机服务发起访问。

kubectl exec -it <tomcat-xxx> -n <vmns> -- curl <vm-server1>.<vmns>.svc.cluster.local:8080

其中，<vm-server1>为虚拟机服务名称，<vmns>为**步骤3**中填写的命名空间。

若交替回显“hello, http protocol server”和“hello, canary http server”，则访问虚拟机服务成功，并且配置的灰度策略生效。

```
[root@cluster                ]# kubectl exec -it tomcat-75cbb4b948-nzrfs -n vmns -- curl vm-server1.vmns.svc.cluster.local:8080
Defaulting container name to container-1.
Use 'kubectl describe pod/tomcat-75cbb4b948-nzrfs -n vmns' to see all of the containers in this pod.
hello, canary http server
[root@cluster                ]#
[root@cluster                ]# kubectl exec -it tomcat-75cbb4b948-nzrfs -n vmns -- curl vm-server1.vmns.svc.cluster.local:8080
Defaulting container name to container-1.
Use 'kubectl describe pod/tomcat-75cbb4b948-nzrfs -n vmns' to see all of the containers in this pod.
hello, http protocol server
```

----结束

移除虚拟机服务灰度版本

步骤1 更新VirtualService。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：virtualservices”，命名空间选择“vmns”，选择**创建VirtualService**中创建的VirtualService，单击操作列的“编辑”，区分两种情况：
   - 情况一：虚拟机服务未配置网关

删除spec.http[0].route下v2版本的destination配置，以及v1版本的weight参数：

```
      weight: 50
    - destination:
        host: <vm-server1>.<vmns>.svc.cluster.local
        port:
          number: 8080
        subset: v2
      weight: 50
```

删除后配置如下：

```
spec:
  hosts:
    - vm-server1
  http:
    - match:
        - gateways:
            - mesh
          port: 8080
      route:
        - destination:
            host: vm-server1.vmns.svc.cluster.local
            port:
              number: 8080
            subset: v1
```

- 情况二：虚拟机服务已配置网关

  分别删除spec.http[0].route和spec.http[1].route下v2版本的destination配置，以及v1版本的weight参数：

```
      weight: 50
    - destination:
        host: <vm-server1>.<vmns>.svc.cluster.local
        port:
          number: 8080
        subset: v2
      weight: 50
```

  删除后配置如下：

```
spec:
  hosts:
    - vm-server1
  http:
    - match:
        - gateways:
            - mesh
          port: 8080
      route:
        - destination:
            host: vm-server1.vmns.svc.cluster.local
            port:
              number: 8080
            subset: v1
    - match:
        - gateways:
            - istio-system/whgw
          port: 1026
      route:
        - destination:
            host: vm-server1.vmns.svc.cluster.local
            port:
              number: 8080
            subset: v1
```

4. 单击“确定”完成修改。

步骤2 更新DestinationRule。

1. 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。
2. 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。
3. 选择“istio资源：destinationrules”，命名空间选择“vmns”，选择**创建DestinationRule**中创建的DestinationRule，单击操作列的“编辑”，删除spec.subsets下v2版本配置：

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

删除后配置如下：

```
spec:
  host: vm-server1
  subsets:
    - labels:
        version: v1
      name: v1
```

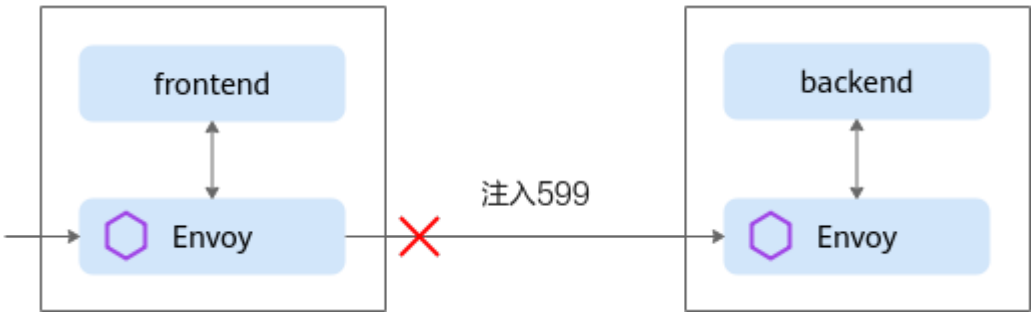
4. 单击“确定”完成修改。

----结束

4.5.3 虚拟机服务配置故障注入

故障注入是一种评估系统可靠性的有效方法，故意在待测试的系统中引入故障，从而测试其健壮性和应对故障的能力。有两种类型的故障注入，可以在转发前延迟（delay）请求，模拟缓慢的网络或过载的服务，也可以中断（abort）HTTP请求，并返回一个特定的HTTP状态码给调用者。通过中断，可以模拟一个有故障的上游服务。

如下图所示，注入一个指定的HTTP状态码（如599）后，对于访问的客户端来说，就跟服务端发生异常一样。



本文以配置中断故障为例，其他治理策略的配置方法可参考**Istio官网**。

前提条件

已执行**添加虚拟机服务到网格**，即已创建v1版本的WorkloadEntry、ServiceEntry，已创建VirtualService、DestinationRule。

更新 VirtualService

步骤1 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。

步骤2 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。

步骤3 选择“istio资源：virtualservices”，命名空间选择“vmns”，选择**创建VirtualService**中创建的VirtualService，单击操作列的“编辑”，在spec.http[0]下添加以下内容：

```
fault:
  abort:
    httpStatus: 599
    percentage:
      value: 100
```

```
spec:
  hosts:
    - vm-server1
  http:
    - match:
        - gateways:
            - mesh
          port: 8080
      route:
        - destination:
            host: vm-server1.vmns.svc.cluster.local
            port:
              number: 8080
            subset: v1
          weight: 50
        - destination:
            host: vm-server1.vmns.svc.cluster.local
            port:
              number: 8080
            subset: v2
          weight: 50
      fault:
        abort:
          httpStatus: 599
          percentage:
            value: 100
```

其中，599为自定义的HTTP状态码，100为故障百分比。

步骤4 单击“确定”完成修改。

----结束

验证故障注入

步骤1 在CCE集群中部署容器服务tomcat，具体操作请参见**部署容器服务**。

步骤2 在kubectl节点执行以下命令，获取tomcat工作负载的Pod名称。

kubectl get pod -n <vmns>|grep tomcat

回显示例：

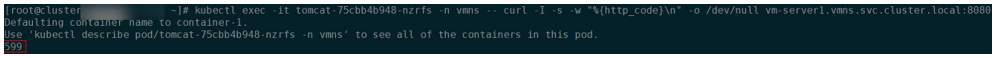
```
tomcat-75cbb4b948-nzrfs      2/2     Running   0      19h
```

步骤3 执行以下命令，对虚拟机服务发起访问。

kubectl exec -it <tomcat-xxx> -n <vmns> -- curl -I -s -w "%{http_code}\n" -o /dev/null <vm-server1>.<vmns>.svc.cluster.local:8080

其中，<vm-server1>为虚拟机服务名称，<vmns>为**步骤3**中填写的命名空间。

若回显“599”，则表示配置的中断故障生效。



----结束

移除故障注入

步骤1 登录应用服务网格控制台，单击服务网格的名称，进入网格详情页面。

步骤2 在左侧导航栏选择“网格配置”，单击“istio资源管理”页签。

步骤3 选择“istio资源：virtualservices”，命名空间选择“vmns”，选择**创建VirtualService**中创建的VirtualService，单击操作列的“编辑”，删除**更新VirtualService**中添加的字段。

步骤4 单击“确定”完成修改。

----结束

4.6 卸载 ASM-PROXY

如果已完成虚拟机治理任务，可以将其中的ASM-PROXY卸载，释放资源。

删除已添加的 iptables 规则

步骤1 以root用户登录虚拟机。

步骤2 执行以下命令删除添加的istio iptables规则。

iptables -t nat -D PREROUTING -p tcp -j ISTIO_INBOUND

iptables -t nat -D OUTPUT -p tcp -j ISTIO_OUTPUT

iptables -t nat -F ISTIO_OUTPUT

iptables -t nat -X ISTIO_OUTPUT

iptables -t nat -F ISTIO_INBOUND

iptables -t nat -X ISTIO_INBOUND

iptables -t nat -F ISTIO_REDIRECT

iptables -t nat -X ISTIO_REDIRECT

iptables -t nat -F ISTIO_IN_REDIRECT

iptables -t nat -X ISTIO_IN_REDIRECT

iptables-save

步骤3 执行以下命令查看虚拟机当前的iptables规则。

iptables -t nat -L -v

回显如下，可以看到istio相关规则已经清除。

```
Chain PREROUTING (policy ACCEPT 2 packets, 104 bytes)
 pkts bytes target     prot opt in     out     source               destination
```



```
  47  2498 DOCKER     all  --  any    any     anywhere             anywhere             ADDRTYPE match dst-type
LOCAL

Chain INPUT (policy ACCEPT 2 packets, 104 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 5 packets, 350 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DOCKER     all  --  any    any     anywhere            !localhost/8          ADDRTYPE match dst-type
LOCAL

Chain POSTROUTING (policy ACCEPT 5 packets, 350 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  any    !docker0  172.17.0.0/16        anywhere

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
0     0 RETURN     all  --  docker0 any     anywhere             anywhere
```

----结束

卸载容器

步骤1 在虚拟机上执行以下命令，获取asm-proxy容器ID。

docker ps | grep asm-proxy

回显第一列为asm-proxy容器的ID。

```
46afed024f46    asm-proxy:1.8.4    "/usr/bin/bash /usr/..."    2 days ago    Up 2 days       asm-proxy
```

步骤2 执行以下命令，停止asm-proxy容器。

docker rm -f *xxx*

其中，*xxx*为**步骤1**获取的asm-proxy容器ID。

----结束

5 如何搭建 IPv4/IPv6 双栈网格

当前CCE支持创建IPv4/IPv6双栈集群，在双栈集群基础上支持开启IPv4/IPv6网格。启用双栈网格后，服务拥有 IPv4地址和IPv6地址，通过这两个地址都可以进行服务间的访问。本教程将指引您搭建一个IPv4/IPv6双栈的网格，使网格内的服务可以通过IPv6地址互访。添加双栈网关后，可以为IPv6终端的客户提供对外访问。

使用场景

- 如果您的服务需要使用IPv6地址进行服务间的互访和流量治理，您需要使用IPv6双栈。
- 如果您要为使用IPv6终端的客户提供对外访问，您需要使用IPv6双栈网格来创建网关。

约束与限制

- 支持启用双栈网格的约束。

网格类型	Istio版本	支持启用的集群类型	集群网络类型	其他说明
基础版	1.18及以上	CCE Turbo集群	云原生网络2.0	集群需要已启用IPv6，请参考**通过CCE搭建IPv4/IPv6双栈集群**

- 支持创建IPv4/IPv6网关的约束。

网格类型	Istio版本	支持启用的ELB类型	ELB规格	其他说明
基础版	1.18及以上	独享型ELB	4层网络型	ELB需要配置IPv6地址

- 您的网格开启IPv4/IPv6双栈后不支持关闭， 未开启双栈的网格也不支持创建完成之后开启双栈。

- 您低版本的网格升级到1.18及以上时，不支持升级后开启IPv4/IPv6双栈。

创建 IPv6 网格

步骤1 登录ASM控制台，购买一个ASM网格，网格参数填写如下：

- 网格类型： 选择“基础版”。
- 网格名称： 输入网格名称。
- Istio版本： 选择"1.18"或以上版本。
- 启用IPv6：打开启用开关，开启后将过滤满足条件的CCE集群。

其他配置参数根据实际情况填写。

步骤2 创建完成后，单击网格名称进入，在“网格配置-基本信息”中可以看到“IPv6双栈 已开启”字样。

----结束

添加 IPv4/IPv6 双栈网关

步骤1 登录ASM控制台，在网格列表页面单击已经开启IPv6双栈的网格名称，单击“网关管理-添加网关”。网关参数如下填写

- 访问方式： 选择“DualStack"。
- 负载均衡： 选择独享型，选中的独享型ELB需要有IPv6地址。

其他配置参数根据实际情况填写。

说明

当前IPv4/IPv6双栈网关添加IPv6外部访问地址时，只支持添加域名，通过域名访问，不支持添加IPv6地址访问。

----结束

验证方式

1. 客户端通过配置域名解析，解析域名到网关的IPv6地址，可以通过域名访问实现IPv6客户端访问。

```
[                                   ~]# curl -v http://test.com:8678
*   Trying 2407:c080:11f0:5fa:4ede:f73d:a9e9:c3c2:8678...
* Connected to test.com (2407:c080:11f0:5fa:4ede:f73d:a9e9:c3c2) port 8678 (#0)
```

2. 查看ingressgateway日志有对应IPv6请求日志信息。

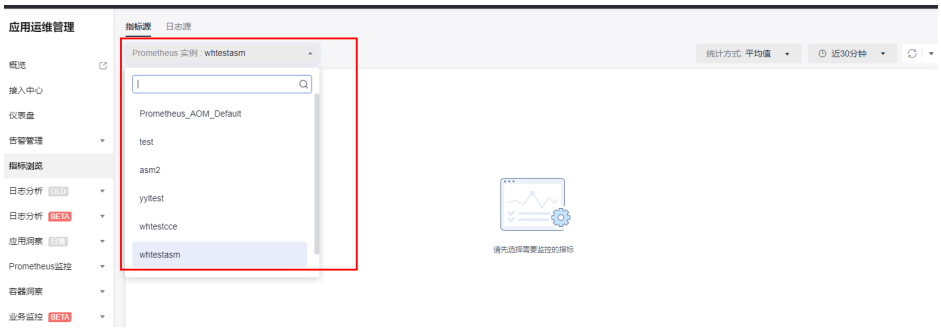
{"user_agent":"curl/7.79.1","upstream_transport_failure_reason":null,"cluster_id":"072f35ec-8388-11ee-9bad-0255ac100b06","mesh_id":"59bac675-94d9-11ee-877b-0255ac10069a","downstream_local_address":"[2407:c080:11f0:601:7d2f:d21b:504d:a96c]:1027","upstream_local_address":"[2407:c080:11f0:601:7d2f:d21b:504d:a96c]:51094","requested_server_name":null,"upstream_service_time":"3","request_id":"805241c0-f1b6-49f3-8ebc-d040d1b7ae5c","connection_termination_details":null,"upstream_host":"[2407:c080:11f0:601:42e0:1a9f:a9ef:8c64]:8080","path":"/","pod_namespace":"istio-system","response_flags":"-","duration":3,"x_forwarded_for":"2407:c080:11f0:5fa:4b12:f0c9:6655:dea1","authority":"test.com:8678","bytes_received":0,"route_name":"tomcat-http-v6.default.svc.cluster.local:8082","response_code_details":"via_upstream","response_code":200,"protocol":"HTTP/1.1","upstream_cluster":"outbound|8082|v1|tomcat-http.v6.default.svc.cluster.local","downstream_remote_address":"[2407:c080:11f0:5fa:4b12:f0c9:6655:dea1]:53300","method":"GET","start_time":"2023-12-09T08:24:23.704Z","bytes_sent":11230,"pod_name":"istio-ingressgateway-1-18-5-rl-5dbfd5d49b-sprl2"}

6 AOM 页面查询应用服务网格详细指标

服务网格开启应用指标并接入华为云AOM后，可以在AOM页面查询网格的详细指标，具体步骤如下：

步骤1 登录AOM页面，进入“指标浏览”页面。

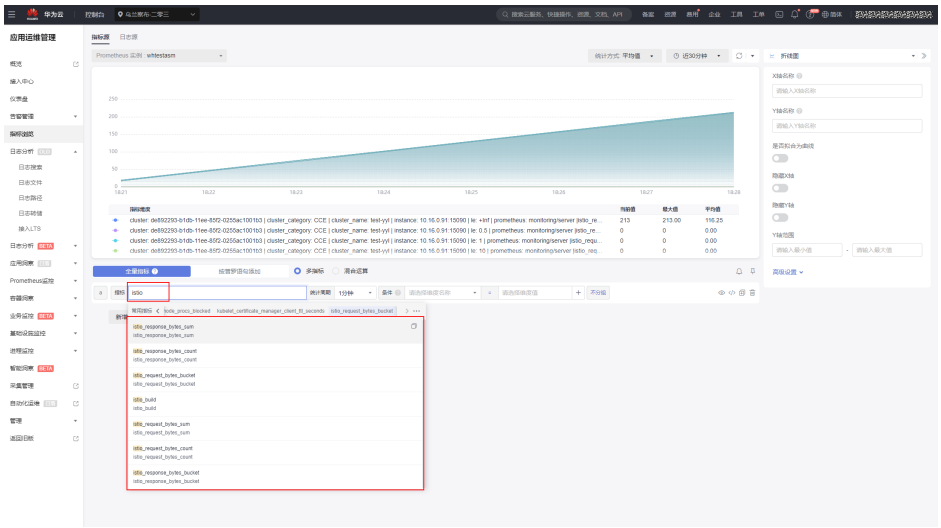
步骤2 选择“指标源”，选择想要查询的网格上报指标的Prometheus实例。



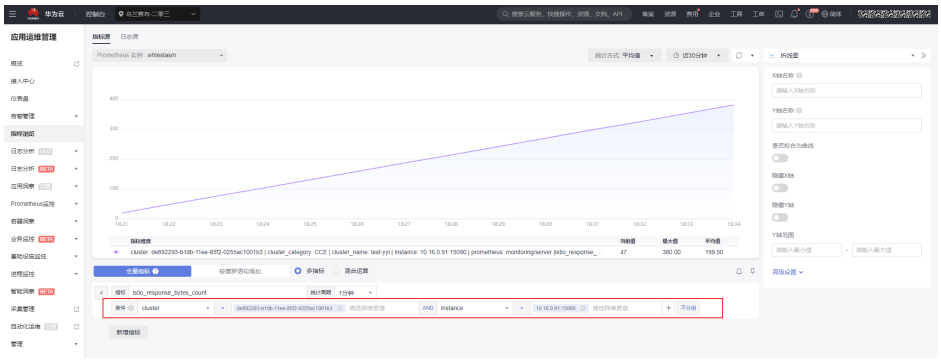
步骤3 在页面上查询上报至AOM的指标。

- 全量指标查询

通过输入具体指标名称，即可查阅istio网格相关指标。

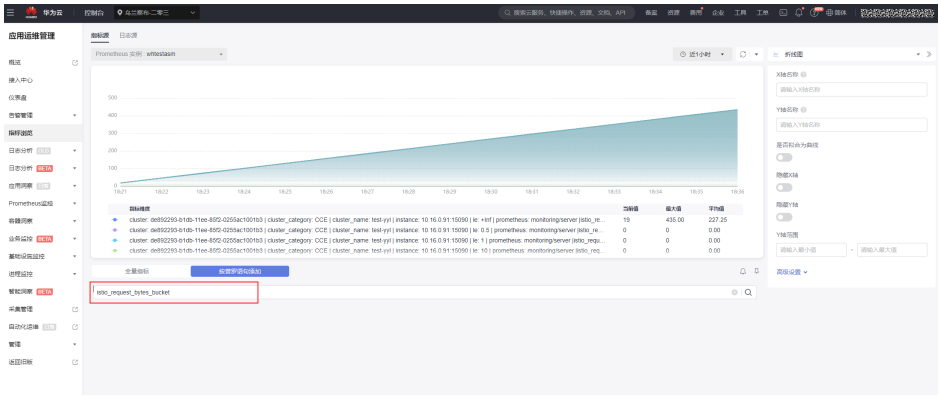


同时支持通过增加条件过滤相关指标信息。



- 按普罗语句查询

通过输入promql语句查询网格指标。



----结束

7 1.0 企业版网格迁移到基础版

前言

本教程介绍了如何将企业版网格迁移到基础版本网格。您可以根据企业版网格的场景选择不同的**迁移方案**。版本差异详见**基础版、企业版及社区开源版本对比**。

费用

企业版网格迁移到基础版网格后，网格将会免费。当前基础版网格不收费。

迁移方案选择

表 7-1 迁移方案

匹配条件	迁移方案
• 仅存在一个集群 • 业务允许小时级别中断	原集群卸载重装方案
• 网格使用了流量治理规则，envoyfilter等配置 • 对外ip端口可替换 • 客户业务可重新部署	新建集群和网格方案
• 仅使用了网格网关能力 • 业务允许闪断	使用ingress中转方案
• 业务多集群 • 业务允许小时级别中断	1.0企业版多集群场景（使用原集群创建网格）
• 业务多集群 • 业务允许闪断	1.0企业版多集群场景（新建集群和网格方案）

准备工作

步骤1 结束现有的灰度发布任务。

步骤2 联系ASM oncall, 配置控制面集群kubectl,使用如下命令进行资源备份。

```
kubectl get Service --all-namespaces -o yaml > all-svc.yaml
kubectl get Secret --all-namespaces -o yaml > all-secret.yaml
kubectl get VirtualService --all-namespaces -o yaml > all-vs.yaml
kubectl get DestinationRule --all-namespaces -o yaml > all-dr.yaml
kubectl get Gateway --all-namespaces -o yaml > all-gw.yaml
kubectl get ServiceEntry --all-namespaces -o yaml > all-se.yaml
kubectl get EnvoyFilter --all-namespaces -o yaml > all-ef.yaml
kubectl get Sidecar --all-namespaces -o yaml > all-sidecar.yaml
kubectl get WorkloadEntry --all-namespaces -o yaml > all-we.yaml
kubectl get WorkloadGroup --all-namespaces -o yaml > all-wg.yaml
kubectl get PeerAuthentication --all-namespaces -o yaml > all-pa.yaml
kubectl get RequestAuthentication --all-namespaces -o yaml > all-ra.yaml
kubectl get AuthorizationPolicy --all-namespaces -o yaml > all-ap.yaml
```

步骤3 配置数据面集群kubectl，使用如下命令进行备份。

```
kubectl get Service --all-namespaces -o yaml > user-all-svc.yaml
kubectl get Secret --all-namespaces -o yaml > user-all-secret.yaml
```

步骤4 若网格内存在多集群，可按照规划，通过过滤namespace /deployment等对特定集群上的资源进行备份。

```
kubectl get vs -A |grep {namespaces}
kubectl get vs -A |grep {deployment}
```

步骤5 针对Envoyfilter和Sidecar等资源，如果仅作用在特定集群的服务上，则可按照作用范围进行分类，如workloadSelector等

```yaml
kind: Sidecar
metadata:
  name: partial-ip-tables
  namespace: prod-us1
spec:
  workloadSelector:
    labels:
      app: productpage
  ingress:
  - bind: 172.16.1.32
    port:
      number: 80 # binds to 172.16.1.32:80
      protocol: HTTP
      name: somename
    defaultEndpoint: 127.0.0.1:8080
    captureMode: NONE
  egress:
    # use the system detected defaults
    # sets up configuration to handle outbound traffic to services
    # in 192.168.0.0/16 subnet, based on information provided by the
    # service registry
  - captureMode: IPTABLES
    hosts:
    - "*/*"
```

步骤6 （1.6企业版涉及）Istio 1.8及以上版本网格网关默认端口需要大于等于1024，整改方案如下：

- 针对gateway资源，将servers.port下的name和number改为大于1024的端口，如80改为1080。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  annotations:
    asm/elbPort: '8001'
  name: test-gw
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
    - hosts:
        - *.*.*.*
      port:
        name: http-80     --> http-1080
        number: 80        --> 1080
        protocol: http
```

- 针对该网关下的svc资源，需要同步修改service targetPort端口。

```
apiVersion: v1
kind: Service
metadata:
  name: test-gw-svc
  namespace: istio-system
spec:
  ports:
    - name: http-f406803d
      protocol: TCP
      port: 8001
      targetPort: 80  --> 1080
      nodePort: 32608
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  clusterIP: *.*.*.*
  clusterIPs:
    - *.*.*.*
  type: LoadBalancer
  sessionAffinity: None
  loadBalancerIP: *.*.*.*
  externalTrafficPolicy: Cluster
status:
  loadBalancer:
    ingress:
      - ip: *.*.*.*
      - ip: *.*.*.*
```

步骤7 （可选）Deletegate VS整改，详见**1.3升级1.8 VirtualService支持Delegate切换**。

说明

仅1.6企业版涉及，1.8版本以上console仅支持delegate vs展示，如果不使用console可不整改

步骤8 检查是否使用了如下废弃资源clusterrbacconfigs、serviceroles、servicerolebindings。若存在这些资源，需要删除对应的资源，因为在1.8以上版本中这些资源已经废弃。

```
kubectl get clusterrbacconfigs -A
kubectl get serviceroles-A
kubectl get servicerolebindings-A
```

步骤9 检查业务pod是否仅监听在了loopback interface（lo），从Istio 1.10版本开始，Sidecar不再重定向流量到 loopback interface（lo），而是将流量重定向到应用的eth0。若您的业务pod监听到了lo，需要改成监听到eth0或者全零监听，否则升级后将导致服务无法访问，详见**升级操作指导**。

步骤10 envoyfilter修改

请参考https://www.envoyproxy.io/docs/envoy/latest/version_history/version_history。

步骤11 查看已开启自动注入的命名空间并记录，如下图 “default”命名空间开启了自动注入。

----结束

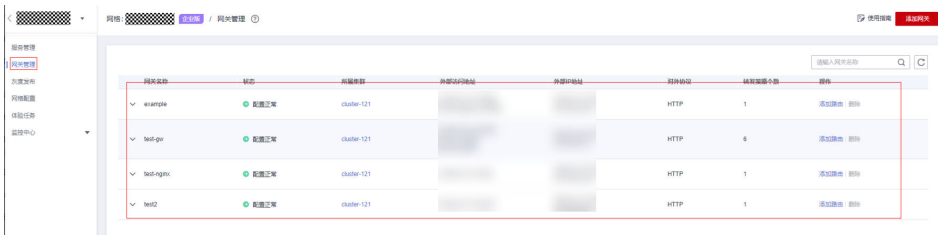
7.1 原集群卸载重装方案

前提

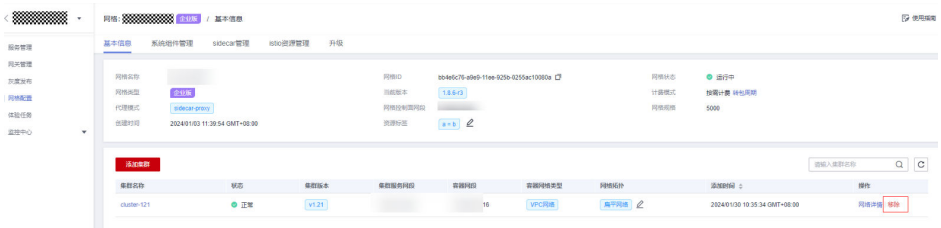
确保准备工作已完成。

移除集群

步骤1 删除网格内的网关资源。



步骤2 单击企业版网格名称进入网格详情页，依次单击“网格配置 -> 基本信息 -> 移除集群” 。



步骤3 选择“重启已有服务”。

----结束

（可选）升级集群版本

步骤1 参考**CCE集群升级指南**进行集群版本升级，推荐升级到1.28版本。

----结束

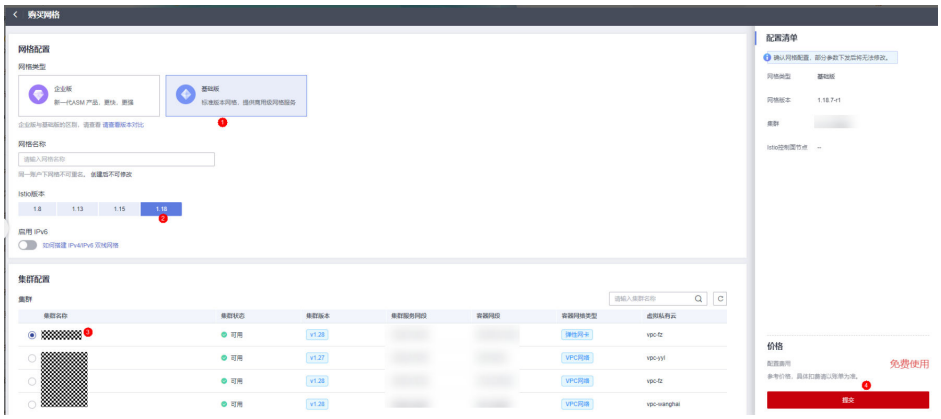
创建新网格

步骤1 根据如下支持列表选择创建基础网格。

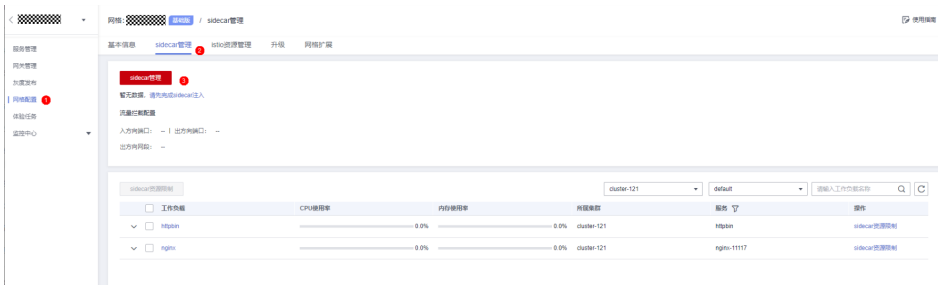
表 7-2 网格版本支持的集群版本

网格版本	CCE集群版本
1.8	1.15-1.21
1.13	1.21-1.23
1.15	1.21-1.27
1.18	1.25-1.28

步骤2 选择从企业版网格移除出来的集群创建基础版网格。



步骤3 选择需要开始自动注入的命名空间（准备工作中记录的命名空间） -> 选择“重启已有服务”。



----结束

配置同步

步骤1 将准备工作中备份的配置文件在新网格中恢复，若未配置kubectl命令，可参考**CCE配置kubectl** 。

```
kubectl create -f all-svc.yaml

kubectl create -f all-vs.yaml
kubectl create -f all-dr.yaml
kubectl create -f all-gw.yaml
kubectl create -f all-se.yaml
```

```
kubectl create -f all-ef.yaml
kubectl create -f all-sidecar.yaml
kubectl create -f all-we.yaml
kubectl create -f all-wg.yaml
kubectl create -f all-pa.yaml
kubectl create -f all-ra.yaml
kubectl create -f all-ap.yaml

kubectl create -f user-all-svc.yaml
kubectl create -f user-all-secret.yaml
```

⚠ 注意

若出现“Error from server (AlreadyExists): xxxxx already exists” 已存在的报错则忽略。

步骤2 删除新版本无用配置。

- 若源版本是1.6企业版则执行如下命令：

```
kubectl -nistio-system delete svc istiod-remote
kubectl -nistio-system delete svc istiod-elb
kubectl -nistio-system delete vs istiod
```

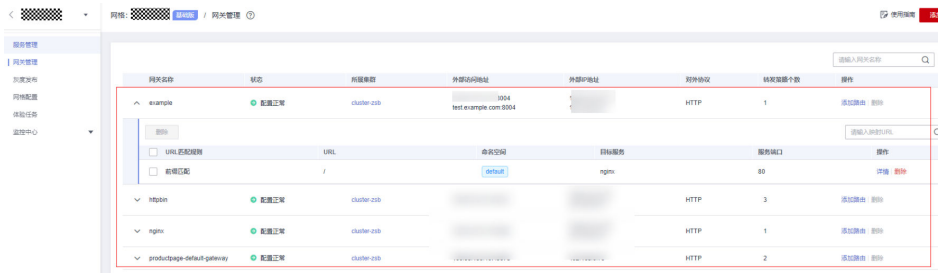
- 若源版本是1.8企业版则执行如下命令：

```
kubectl -nistio-system delete envoyfilter metadata-exchange-1.6
kubectl -nistio-system delete envoyfilter metadata-exchange-1.7
kubectl -nistio-system delete envoyfilter metadata-exchange-1.8
kubectl -nistio-system delete envoyfilter stats-filter-1.6
kubectl -nistio-system delete envoyfilter stats-filter-1.7
kubectl -nistio-system delete envoyfilter stats-filter-1.8
kubectl -nistio-system delete envoyfilter tcp-metadata-exchange-1.6
kubectl -nistio-system delete envoyfilter tcp-metadata-exchange-1.7
kubectl -nistio-system delete envoyfilter tcp-metadata-exchange-1.8
kubectl -nistio-system delete envoyfilter tcp-stats-filter-1.6
kubectl -nistio-system delete envoyfilter tcp-stats-filter-1.7
kubectl -nistio-system delete envoyfilter tcp-stats-filter-1.8
kubectl -nistio-system delete svc istiod-remote
kubectl -nistio-system delete svc istiod-elb
kubectl -nistio-system delete vs istiod
```

----结束

功能验证

步骤1 查看console功能是否正常，如网关路由显示 ，网关访问等。



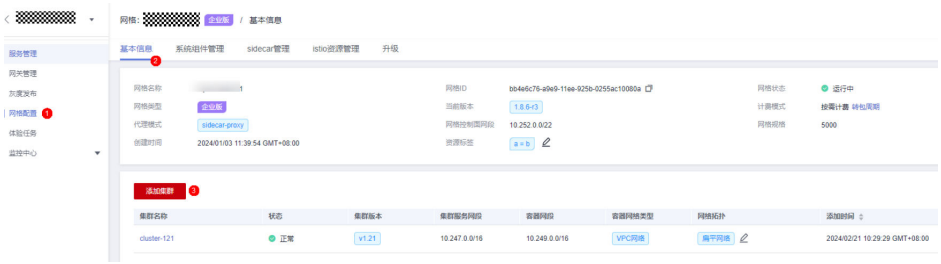
步骤2 验证业务功能是否正常。

----结束

异常回退

步骤1 单击删除按钮卸载基础版网格，选择“重启已有服务”。

步骤2 卸载后将集群添加回1.0企业版。



步骤3 选择集群 -> 选择“扁平网格” -> 选择需要开始自动注入的命名空间（准备工作中记录的命名空间） -> 选择“重启已有服务”。

步骤4 使用控制面kubectl 执行如下命令。

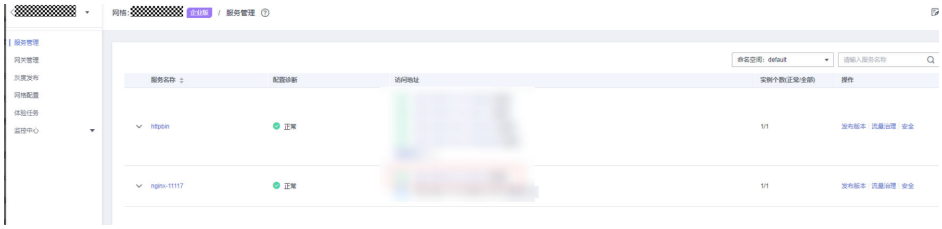
```
kubectl create -f all-svc.yaml

kubectl create -f all-vs.yaml
kubectl create -f all-dr.yaml
kubectl create -f all-gw.yaml
kubectl create -f all-se.yaml
kubectl create -f all-ef.yaml
kubectl create -f all-sidecar.yaml
kubectl create -f all-we.yaml
kubectl create -f all-wg.yaml
kubectl create -f all-pa.yaml
kubectl create -f all-ra.yaml
kubectl create -f all-ap.yaml
```

步骤5 使用数据面集群kubectl执行如下命令。

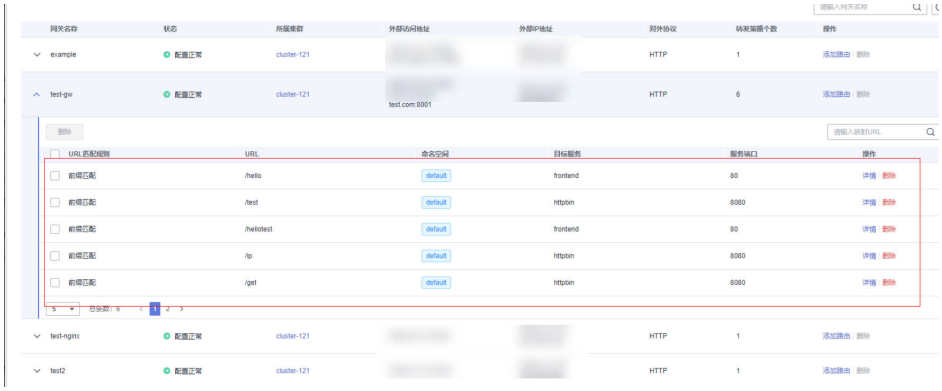
```
kubectl create -f user-all-svc.yaml
kubectl create -f user-all-secret.yaml
```

步骤6 功能验证。

- 网关访问正常



- 路由显示正常



----结束

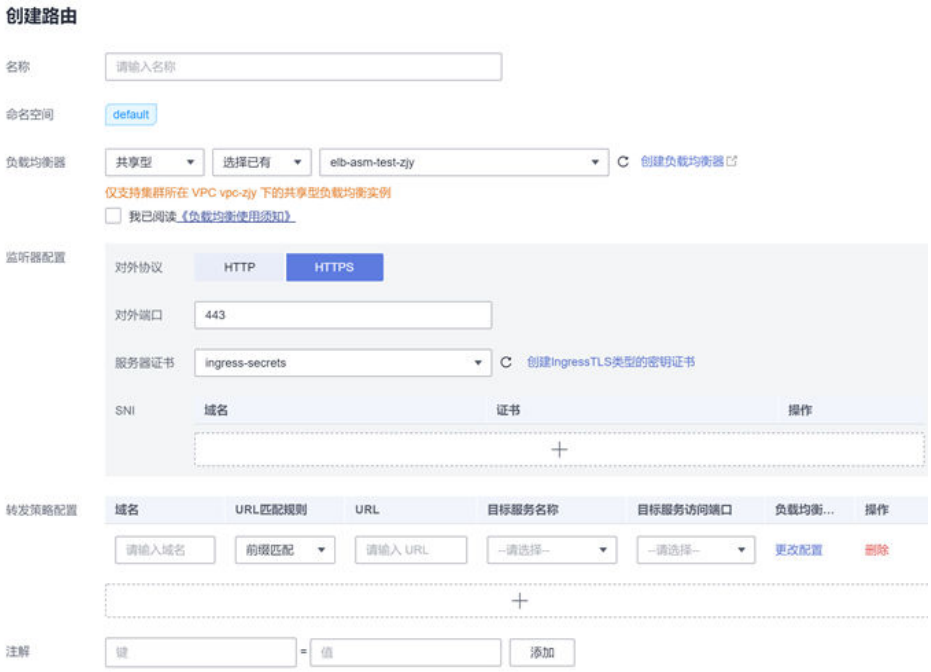
7.2 使用 ingress 中转方案

前提

- 确保准备工作已完成。
- 网格仅使用了网关能力。
- 已经创建和原网格网关同规格的ELB实例。

创建 Ingress 资源

步骤1 （推荐）根据现有的Ingress-gateway配置，在CCE环境下创建Ingress配置，并配置https证书。



步骤2 验证流量是否正常，确保Ingress流量正常后方可进行下一步。

```
[root@cluster-zjy-cluster-91573-lt5ww ~]# curl -v -HHost:httpbin.example.com --cacert httpbin.example
com/2_intermediate/certs/ca-chain.cert.pem https://httpbin.example.com:443/
* About to connect() to httpbin.example.com port 443 (#0)
*   Trying [redacted]...
* Connected to httpbin.example.com ([redacted]) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
*   CAfile: httpbin.example.com/2_intermediate/certs/ca-chain.cert.pem
  CApath: none
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*       subject: CN=httpbin.example.com,O=Dis,L=Springfield,ST=Denial,C=US
*       start date: May 12 02:48:06 2022 GMT
*       expire date: May 22 02:48:06 2023 GMT
*       common name: httpbin.example.com
*       issuer: CN=httpbin.example.com,O=Dis,ST=Denial,C=US
> GET / HTTP/1.1
```

ingress和istio-gateway同时支持httpbin.example.com域名，通过更改host可以切换不同的流量。

```
[root@cluster-update-validate-94611-vzptt test0511]# cat /etc/hosts
::1     localhost       localhost.localdomain   localhost6      localhost6.localdomain6

127.0.0.1       localhost       localhost.localdomain   localhost4      localhost4.localdomain4
127.0.0.1       x86-euleros-40g-notdelete-fuxi-pipeline-02      x86-euleros-40g-notdelete-fuxi-pipeline-02
127.0.0.1       cluster-update-validate-94611-vzptt     cluster-update-validate-94611-vzptt

#100.93.3.134  httpbin.example.com
100.95.144.109 httpbin.example.com
127.0.0.1 cluster-update-validate-94611-vzptt
[root@cluster-update-validate-94611-vzptt test0511]#
```

```
[root@cluster-update-validate-94611-vzptt test0511]# curl -v -HHost:httpbin.example.com --cacert httpbin.example.com/2_intermediate/certs/ca-chain.cert.pem https://httpbin.example.com:443/productpage
* About to connect() to httpbin.example.com port 443 (#0)
*   Trying [redacted]
* Connected to httpbin.example.com ([redacted]) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
*   CAfile: httpbin.example.com/2_intermediate/certs/ca-chain.cert.pem
  CApath: none
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*       subject: CN=httpbin.example.com,O=Dis,L=Springfield,ST=Denial,C=US
*       start date: May 12 02:48:06 2022 GMT
*       expire date: May 22 02:48:06 2023 GMT
*       common name: httpbin.example.com
*       issuer: CN=httpbin.example.com,O=Dis,ST=Denial,C=US
> GET /productpage HTTP/1.1
> User-Agent: curl/7.29.0
> Accept: */*
> Host:httpbin.example.com
>
```

----结束

业务切流

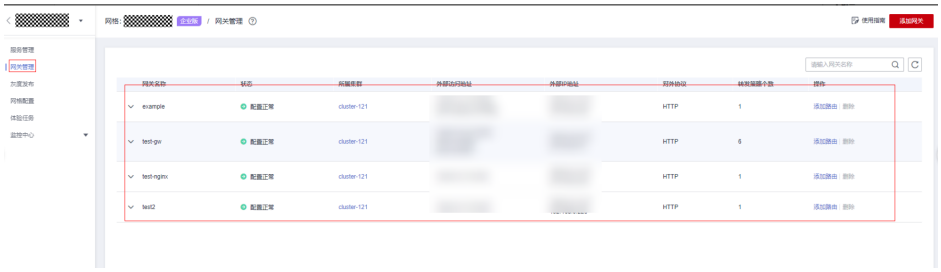
步骤1 在本地host将域名对应的ELB IP地址改为新的ELB IP。

步骤2 本地验证功能，验证成功后修改DNS 配置，将域名解析IP地址改为新ELB的IP。

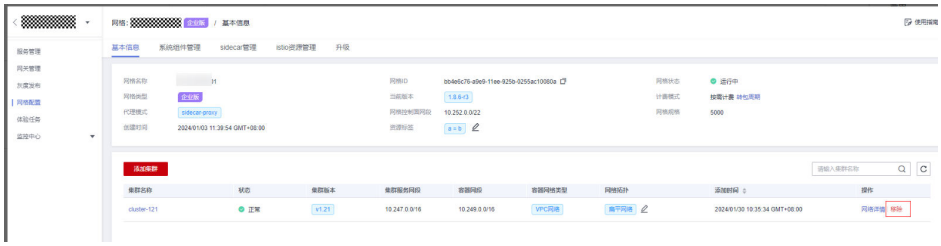
----结束

迁移

步骤1 删除网格内的网关资源。

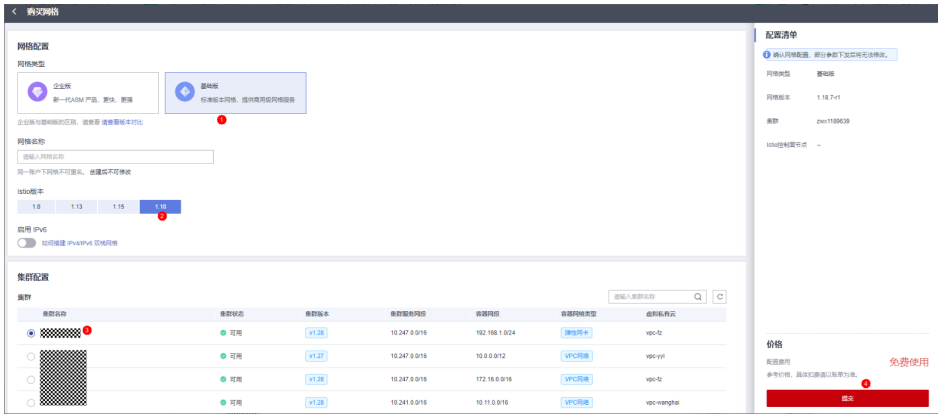


步骤2 单击企业版网格 -> 网格配置 -> 基本信息 -> 移除集群。

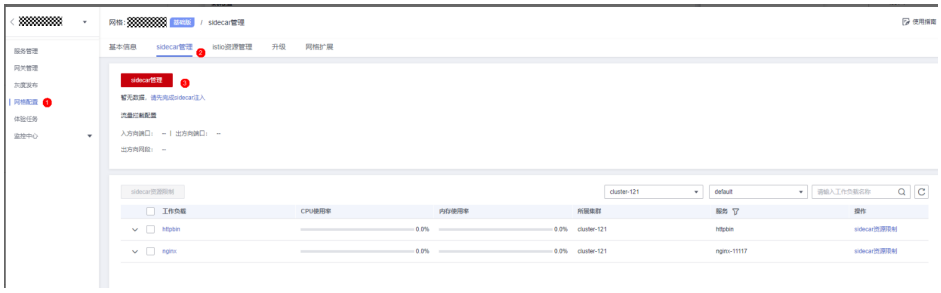


步骤3 选择重启已有服务。

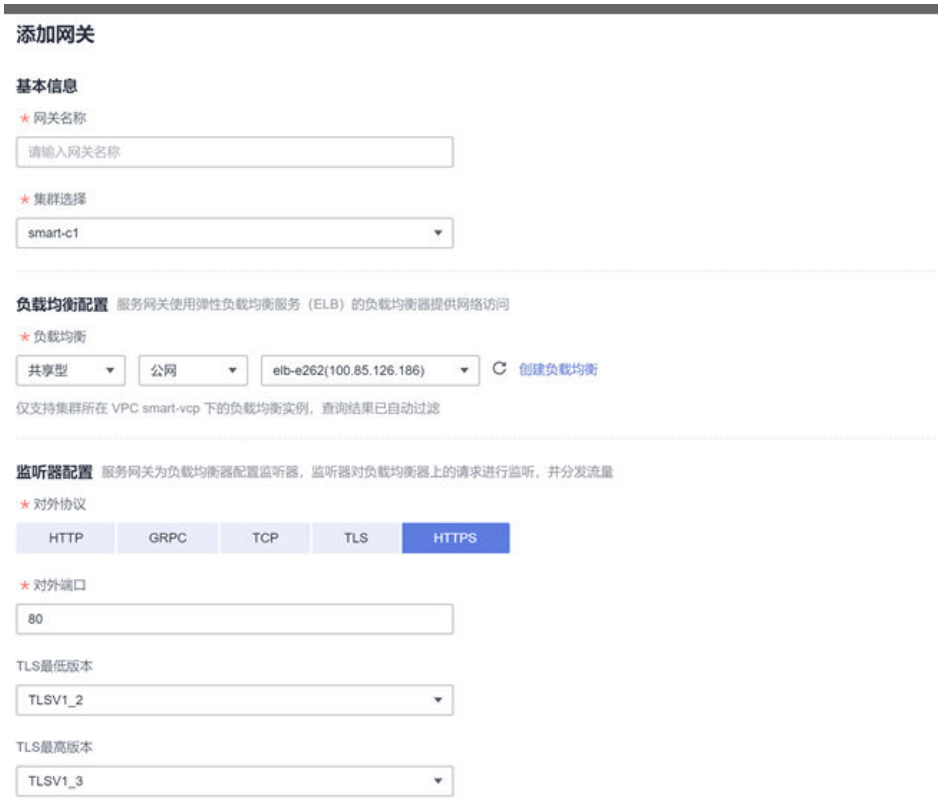
步骤4 选择从企业版网格移除出来的集群创建基础版网格。



步骤5 选择需要开始自动注入的命名空间（准备工作中记录的命名空间） -> 选择“重启已有服务”。



步骤6 手动创建网关，选择原ELB实例。



步骤7 将准备工作中备份的配置文件在新网格中恢复，若未配置kubectl命令，可参考**CCE配置kubectl** 。

```
kubectl create -f all-vs.yaml
kubectl create -f all-dr.yaml
```

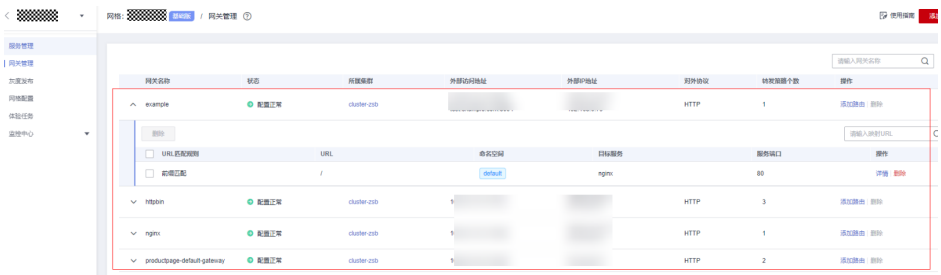
⚠ 注意

若出现“Error from server (AlreadyExists): xxxxx already exists” 已存在的报错则忽略。

----结束

功能验证

步骤1 查看console功能是否正常，如网关路由显示 ，网关访问等。



步骤2 修改host地址为老ELB IP，验证业务功能是否正常。

步骤3 若功能正常则将DNS解析地址改为老ELB IP。

步骤4 验证正常后删除CCE ingress。

----结束

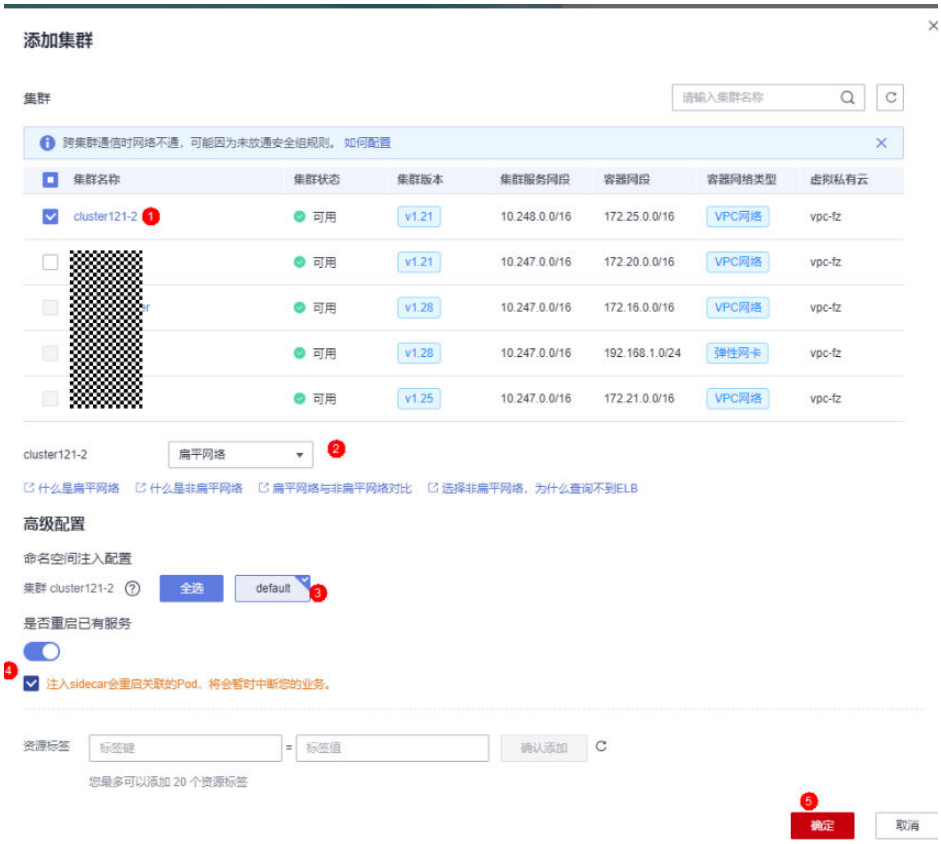
异常回退

若在迁移过程中出现异常，可按照如下步骤回退。

步骤1 单击删除按钮卸载基础版网格，选择“重启已有服务”。

步骤2 卸载后将集群添加回1.0企业版。

步骤3 选择集群 -> 选择“扁平网络” -> 选择需要开始自动注入的命名空间（准备工作中记录的命名空间） -> 选择“重启已有服务”。

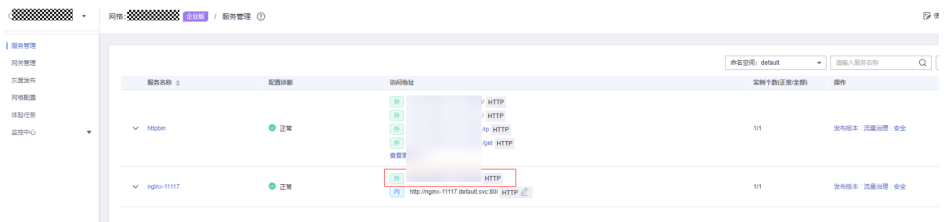


步骤4 使用控制面kubectl 执行如下命令。

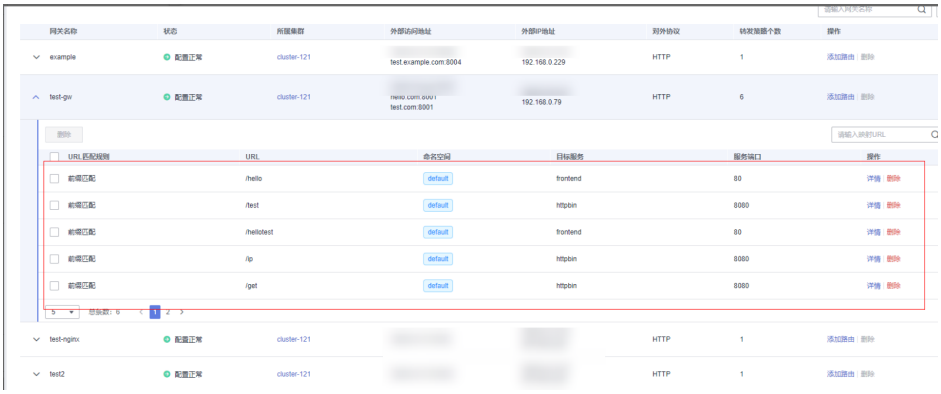
```
kubectl create -f all-svc.yaml
kubectl create -f all-vs.yaml
kubectl create -f all-dr.yaml
kubectl create -f all-gw.yaml
```

步骤5 功能验证。

- 网关访问正常



- 路由显示正常



步骤6 修改DNS解析地址为网关ELB IP。

步骤7 删除CCE ingress。

----结束

7.3 新建集群和网格迁移方案

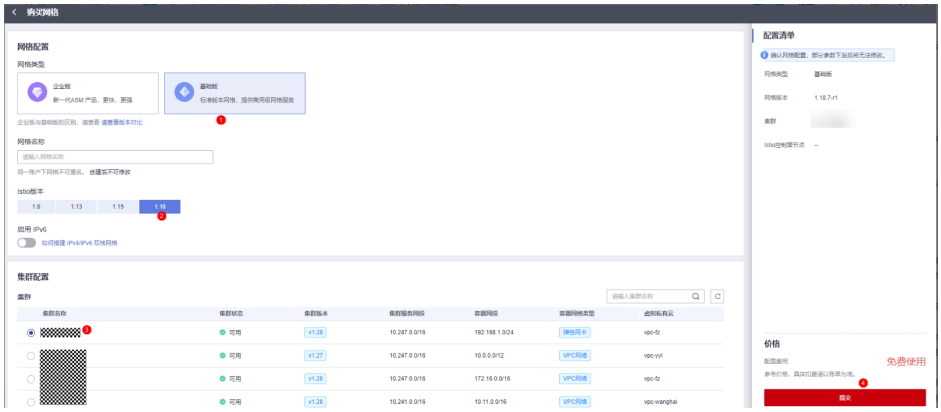
注意

- 请务必确保**准备工作**完成。
- 请根据自身需求选择集群版本和网格版本，推荐使用最新集群和网格版本。

新建集群和网格

步骤1 登录CCE console 选择在当前企业版集群所在的同VPC下**创建集群**。

步骤2 在ASM console ，创建基础版本网格，选择新建的集群。



步骤3 选择需要开始自动注入的命名空间（准备工作中记录的命名空间） -> 选择“重启已有服务”。

步骤4 查看网格使用了几个ELB，新建同等数量和规格的ELB。

----结束

配置同步

方案一 手动同步网格配置（推荐）

手动在新建的网格上添加网关路由，并同步1.0企业版网格配置。

方案二 使用备份网格资源还原配置

步骤1 查看企业版网格网关使用了几个ELB，使用新建的几个ELB进行替换，如下图所示使用了两个ELB：

ELB1：（3dbb4787-75c1-42f0-897a-1f683f7e89a0）*.*.*.*

ELB2：（e60fdaa7-3398-4a19-8bd1-d53598c6917e）*.*.*.*

新建两个ELB。

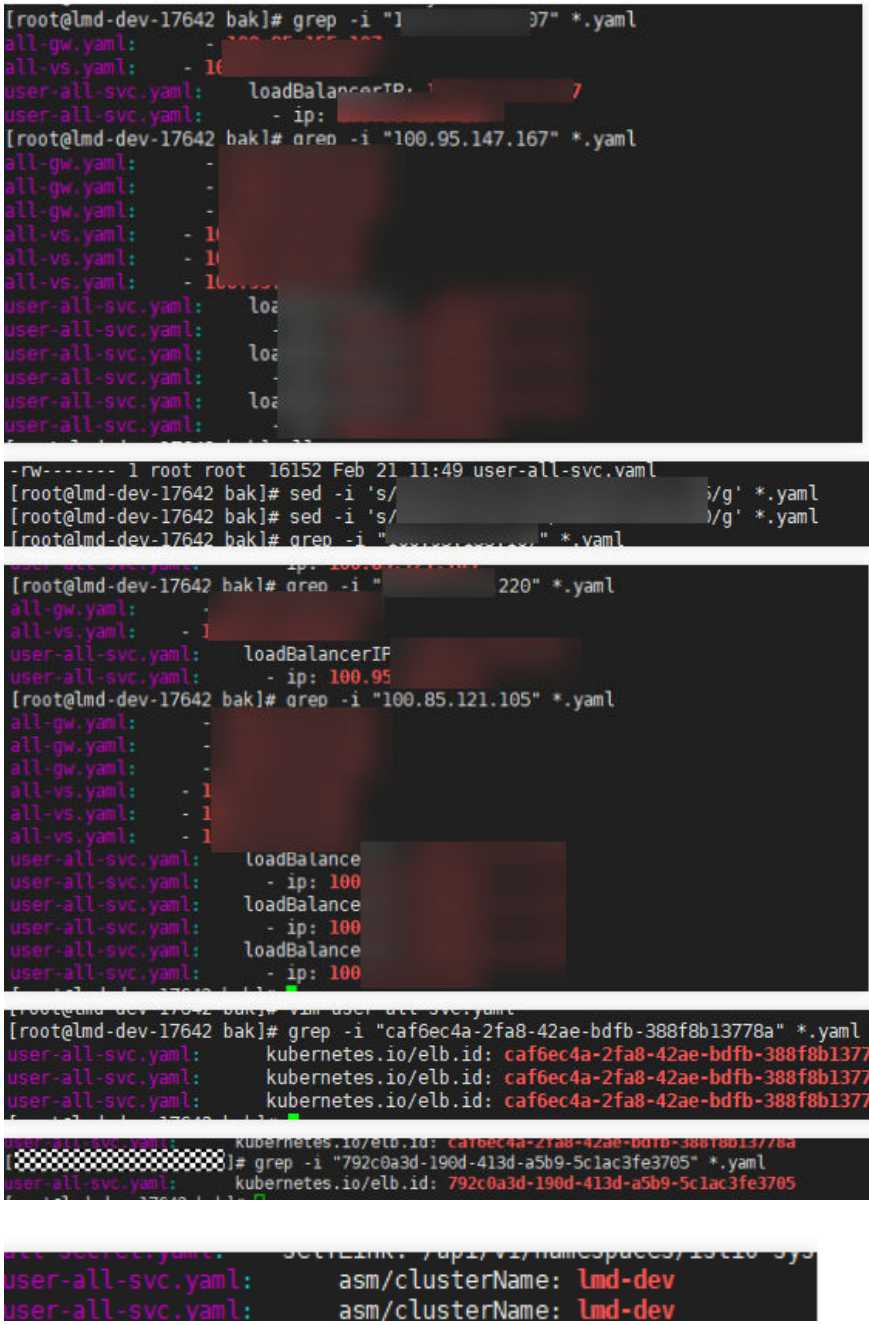
newELB1：（caf6ec4a-2fa8-42ae-bdfb-388f8b13778a）*.*.*.*

newELB2：（792c0a3d-190d-413d-a5b9-5c1ac3fe3705）* .*.*.*

将准备工作中备份的istio 配置文件拷贝一份到新集群节点上, 执行如下命令：

```
#查看老ELB IP
grep -i "老ELB IP" *.yaml

#ELB IP 替换
sed -i 's/老ELB IP/新ELB IP/g' *.yaml

#ELB ID 替换
sed -i 's/老ELB ID/新ELB ID/g' *.yaml

#替换clusterID
sed -i 's/老clusterID/新clusterID/g' *.yaml

#替换CluseterName
sed -i 's/老CluseterName/新CluseterName/g' *.yaml

#替换完成后查看
grep -i "新ELB IP" *.yaml
grep -i "新ELB ID" *.yaml
grep -i "新clusterID" *.yaml
grep -i "新CluseterName" *.yaml
```

替换完成后如下所示：



步骤2 将准备工作中备份的配置文件在新网格中恢复，若未配置kubectl命令，可参考**CCE配置kubectl** 。

```
kubectl create -f all-svc.yaml
//kubectl create -f all-secret.yaml
kubectl create -f all-vs.yaml
kubectl create -f all-dr.yaml
kubectl create -f all-gw.yaml
kubectl create -f all-se.yaml
kubectl create -f all-ef.yaml
kubectl create -f all-sidecar.yaml
kubectl create -f all-we.yaml
kubectl create -f all-wg.yaml
kubectl create -f all-pa.yaml
kubectl create -f all-ra.yaml
kubectl create -f all-ap.yaml
```



```
kubectl create -f user-all-svc.yaml
kubectl create -f user-all-secret.yaml
```

⚠ 注意

若出现“Error from server (AlreadyExists): xxxxx already exists” 已存在的报错则忽略。

步骤3 删除新版本无用配置。

- 若源版本是1.6企业版则执行如下命令：

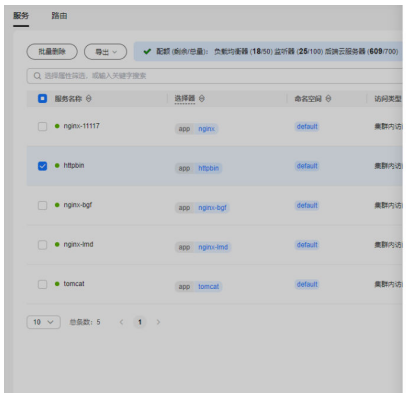
```
kubectl -nistio-system delete svc istiod-remote
kubectl -nistio-system delete svc istiod-elb

kubectl -nistio-system delete vs istiod
```

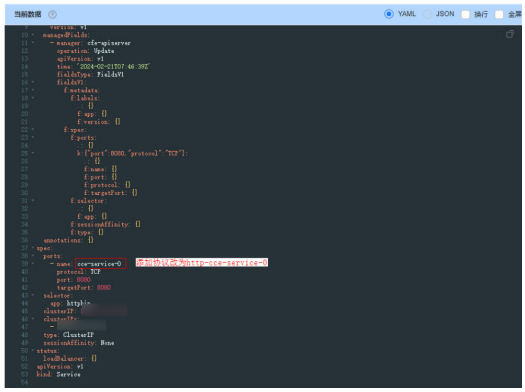
- 若源版本是1.8企业版则执行如下命令：

```
kubectl -nistio-system delete envoyfilter metadata-exchange-1.6
kubectl -nistio-system delete envoyfilter metadata-exchange-1.7
kubectl -nistio-system delete envoyfilter metadata-exchange-1.8
kubectl -nistio-system delete envoyfilter stats-filter-1.6
kubectl -nistio-system delete envoyfilter stats-filter-1.7
kubectl -nistio-system delete envoyfilter stats-filter-1.8
kubectl -nistio-system delete envoyfilter tcp-metadata-exchange-1.6
kubectl -nistio-system delete envoyfilter tcp-metadata-exchange-1.7
kubectl -nistio-system delete envoyfilter tcp-metadata-exchange-1.8
kubectl -nistio-system delete envoyfilter tcp-stats-filter-1.6
kubectl -nistio-system delete envoyfilter tcp-stats-filter-1.7
kubectl -nistio-system delete envoyfilter tcp-stats-filter-1.8

kubectl -nistio-system delete svc istiod-remote
kubectl -nistio-system delete svc istiod-elb

kubectl -nistio-system delete vs istiod
```

步骤4 验证业务功能若出现服务异常场景，单击处理，查看异常错误。

在CCE service页面修改





----结束

业务切流

业务切流有两种方案，可根据需要进行选择。

方案一 使用DNS切换后端ELB IP

步骤1 在本地host将域名对应的ELB IP地址改为新的ELB IP。

步骤2 本地验证功能，验证成功后修改DNS 配置，将域名解析IP地址改为新ELB的IP。

----结束

方案二 使用ServiceEntry和WorkloadEntry进行灰度切流（待定，感觉没使用场景）

功能验证

步骤1 业务切流进行全面功能验证。

----结束

异常回退

若在迁移过程中出现异常，可按照如下步骤回退

步骤1 将DNS 配置复原。

步骤2 将新建网格集群删除。

----结束

7.4 1.0 企业版多集群场景（新建集群和网格方案）

前提

1. 确保准备工作已完成。
2. 梳理网格下各个集群的网关和跨集群svc。

集群	网关	跨集群访问的服务
A	a-gw	集群A 、集群B

B	b-gw	集群B
...	...	...

3. 关闭mtls以及访问授权，是用如下命名查看是否存在对应资源。

```
kubectl get PeerAuthentication -A
kubectl get AuthorizationPolicy -A
kubectl get RequestAuthentication -A
```

说明

- 查看查询到的PeerAuthentication ，若spec.mtls.mode为STRICT，则需要将STRICT改为PERMISSIVE。
- 查看查询到的AuthorizationPolicy， 若使用了该功能，则需要在迁移过程中暂时删除访问授权。
- 查看查询到的RequestAuthentication， 若使用了该功能，则需要在迁移过程中暂时删除请求认证。

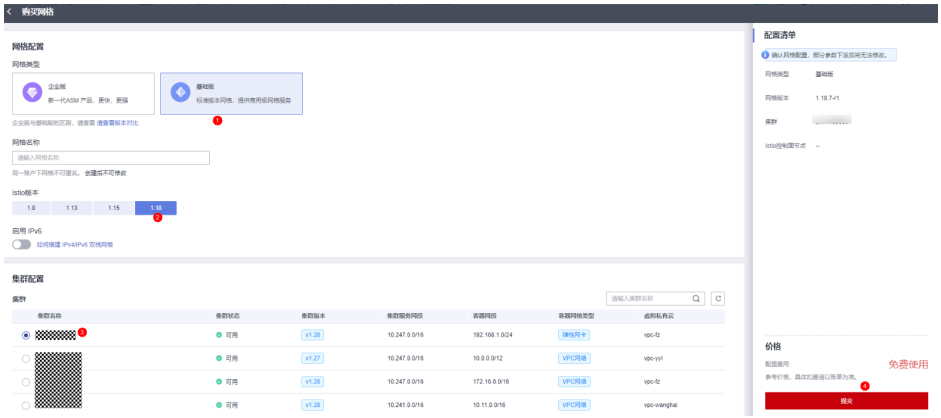
4. 解除业务上使用k8s域名进行跨集群访问。

新建集群和网格

步骤1 登录CCE console 选择在当前企业版集群所在的同VPC下创建集群。

步骤2 重新部署业务。

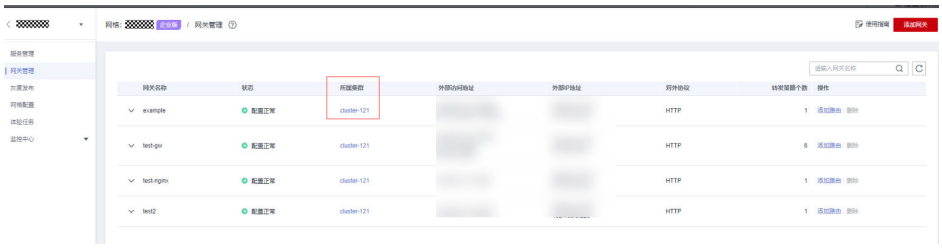
步骤3 在ASM console ，创建基础版本网格，选择新建的集群。



步骤4 择需要开始自动注入的命名空间（准备工作中记录的命名空间） -> 选择“重启已有服务”。

步骤5 查看网格使用了几个ELB，新建同等数量和规格的ELB，重新创建网关和路由。

- 方案一：
  a. 查看待迁移集群上有几个网关路由。



b. 在新基础版网格上选用新建的ELB，重新创建网关和路由。

- 方案二：

a. 将准备工作中备份的该集群资源上传到集群节点，执行如下命令恢复，其中xx为备份的文件名。

```
#查看老资源
grep -i "老ELB IP" *.yaml
grep -i "老ELB ID" *.yaml
grep -i "老clusterID" *.yaml
grep -i "老CluseterName" *.yaml

#ELB IP 替换
sed -i 's/老ELB IP/新ELB IP/g' *.yaml

#ELB ID 替换
sed -i 's/老ELB ID/新ELB ID/g' *.yaml

#替换clusterID
sed -i 's/老clusterID/新clusterID/g' *.yaml

#替换CluseterName
sed -i 's/老CluseterName/新CluseterName/g' *.yaml

#替换完成后查看
grep -i "新ELB IP" *.yaml
grep -i "新ELB ID" *.yaml
grep -i "新clusterID" *.yaml
grep -i "新CluseterName" *.yaml

kubectl create -f xx.yaml
```

----结束

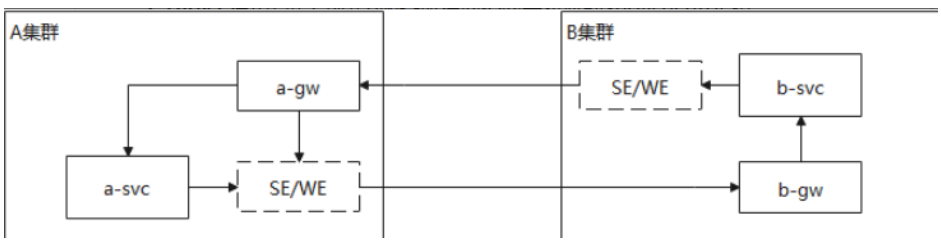
【可选】解除业务跨集群 k8s 服务访问

- 方案一：将有访问关系的服务迁移到同一个集群中。
- 方案二：通过创建网关进行服务间访问。

业务解除跨集群流量访问，如A集群a服务访问B集群b服务，以此为例需进行如下操作

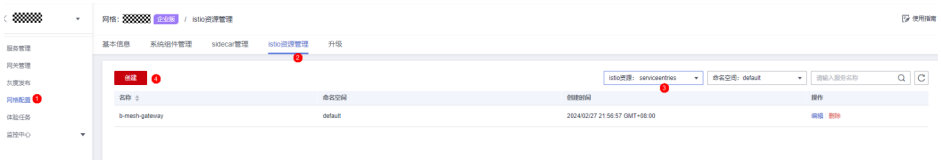
步骤1 在新基础版网格上创建网关访问b服务，路由选择前缀匹配，URL选择 "/"。

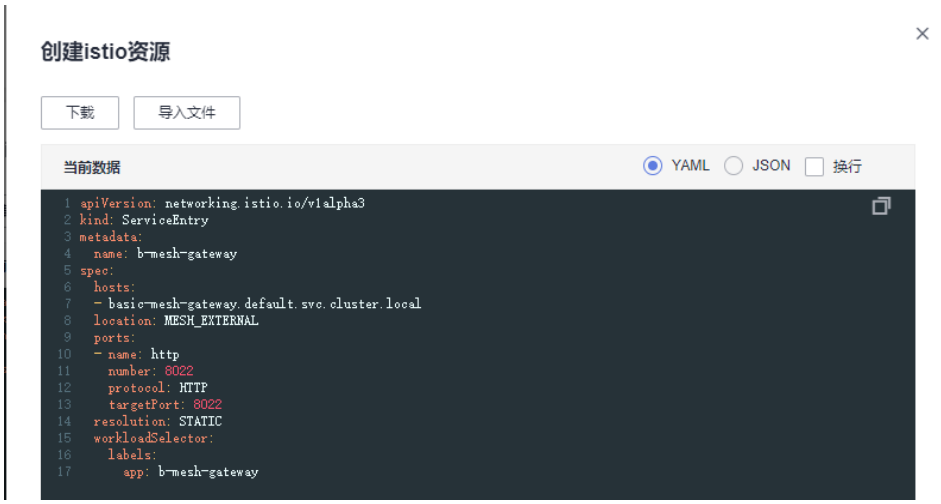
步骤2 (可选) 在A集群所在的网格上创建b服务所在网格网关对应的ServiceEntry 和WorkloadEntry代替b服务，在B网格新建的Gateway中添加'*', VirtualService中的hosts 改为 '*' 。



```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: b-mesh-gateway
spec:
  hosts:
  - b-mesh-gateway.default.svc.cluster.local
  location: MESH_EXTERNAL
  ports:
  - name: http
    number: 80      //创建的网关访问端口
    protocol: HTTP
    targetPort: 80  //创建的网关访问端口
  resolution: STATIC
  workloadSelector:
    labels:
      app: b-mesh-gateway
---
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: b-mesh-gateway
spec:
  address: x.x.x.x  //新ELB的IP地址
  labels:
    app: b-mesh-gateway
```

如下图所示：





gateway和virtualSevice host添加‘*’

```
  uid: 7aa3e9ad-509f-4cba-a070-4dc09b80b21f
spec:
  selector:
    istio: ingressgateway
  servers:
    - hosts:
        -
        - boutique.io
        - '*'
      port:
        name: http-8a3775b4
        number: 1025
        protocol: http
```

```
  uid: e1b71c9a-4db6-4d21-66e0-578d280c3f21
spec:
  gateways:
    - istio-system/test-gw
  hosts:
    - '*'
  http:
    - delegate:
        name: frontend-80
        namespace: default
      match:
        - uri:
            prefix: /
```

步骤3 开启istio DNS访问

在A业务集群上使用如下命令

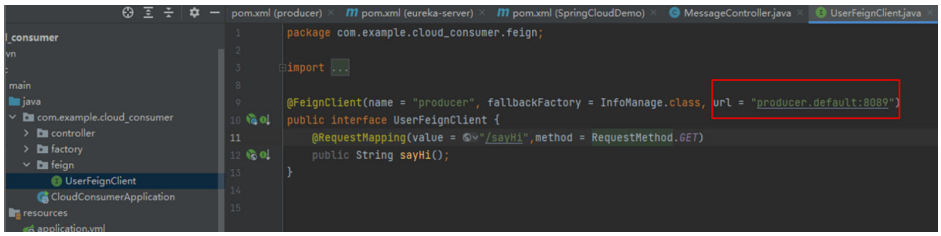
```
kubectl edit iop {iop名} -nistio-system
```

在spec.meshConfig.defaultConfig.proxyMetadata下如下添加配置

```
ISTIO_META_DNS_CAPTURE: "true"
ISTIO_META_DNS_AUTO_ALLOCATE: "true"
```

步骤4 修改访问，确保流量已经访问到新集群。

1. 如使用springcloud，需要修改FeignClient访问地址，在注解中指定新url地址（producer.default:8089 -> b-mesh-gateway.default:8089，若第二步未操作，则直接替换为网关地址）



2. 创建灰度任务，切部分流量到修改了配置的新版本。
3. 验证功能正常后将在CCE界面将镜像替换成新版本，然后取消灰度任务。

----结束

移除企业版集群

步骤1 从DNS上将待移除的网关的地址改为新基础版集群的网关。

步骤2 删除该集群上部署的网关，可以在网关管理页面查看网关所属集群。

步骤3 将对应的集群移除出企业版网格。

步骤4 重复1-3步骤直到所有集群都移除出网格

----结束

功能验证

验证业务功能正常可用。

异常回退

步骤1 将集群添加回企业版网格。

步骤2 使用控制面kubectl 执行如下命令：

```
kubectl create -f all-gw.yaml
```

使用数据面kubectl 执行如下命令：

```
kubectl create -f user-all-svc.yaml
```

步骤3 功能验证。

步骤4 将新建网格集群删除。

----结束

7.5 1.0 企业版多集群场景（使用原集群创建网格）

前提

1. 确保准备工作已完成。
2. 梳理网格下各个集群的网关和跨集群svc。

集群	网关	跨集群访问的服务

A	a-gw	集群A 、集群B
B	b-gw	集群B
...	...	...

3. 关闭mtls以及访问授权，是用如下命名查看是否存在对应资源。

```
kubectl get PeerAuthentication -A
kubectl get AuthorizationPolicy -A
kubectl get RequestAuthentication -A
```

说明

- 查看查询到的PeerAuthentication ，若spec.mtls.mode为STRICT，则需要将STRICT改为PERMISSIVE。
- 查看查询到的AuthorizationPolicy， 若使用了该功能，则需要在迁移过程中暂时删除访问授权。
- 查看查询到的RequestAuthentication， 若使用了该功能，则需要在迁移过程中暂时删除请求认证。

4. 允许迁移期间业务访问中断。
5. 解除业务上使用k8s域名进行跨集群访问。

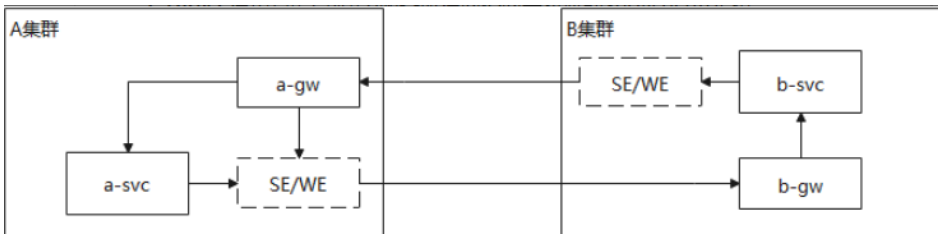
解除业务跨集群 k8s 服务访问（可选）

- 方案一：将有访问关系的服务迁移到同一个集群中。
- 方案二：通过创建网关进行服务间访问。

业务解除跨集群流量访问，如A集群a服务访问B集群b服务，此步骤需要在迁移集群后操作。

步骤1 在B集群移除出企业版网格之后新建的基础版网格上新建网关访问b服务，路由选择前缀匹配，URL选择 "/"。

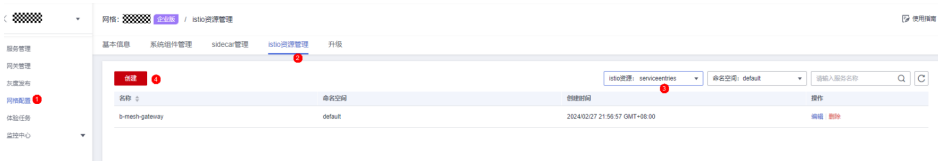
步骤2 (可选) 在A集群上创建b服务所在网格网关对应的ServiceEntry 和 WorkloadEntry代替b服务，在B网格新建的Gateway中添加'*', VirtualService中的 hosts 改为 '*' 。

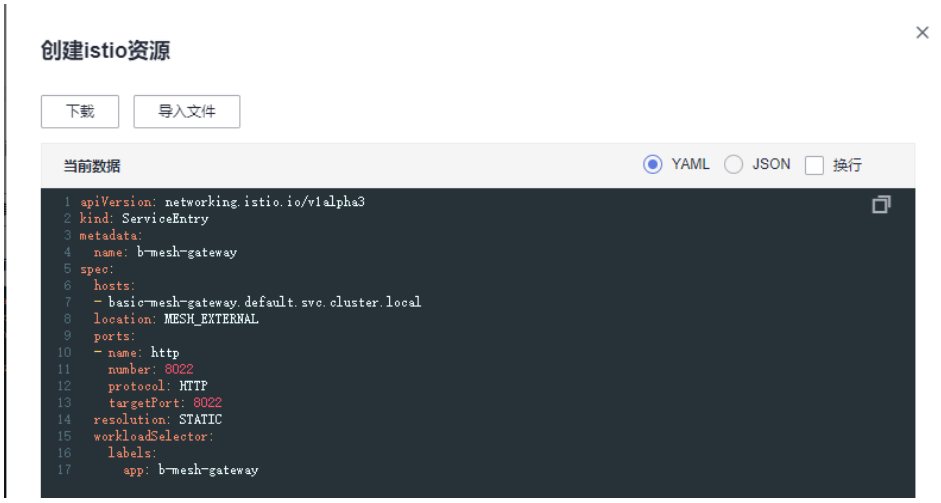



```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: b-mesh-gateway
spec:
  hosts:
  - b-mesh-gateway.default.svc.cluster.local
  location: MESH_EXTERNAL
  ports:
  - name: http
    number: 80      //创建的网关访问端口
    protocol: HTTP
```

```
    targetPort: 80  //创建的网关访问端口
  resolution: STATIC
  workloadSelector:
    labels:
      app: b-mesh-gateway
---
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: b-mesh-gateway
spec:
  address: x.x.x.x  //新ELB的IP地址
  labels:
    app: b-mesh-gateway
```

如下图所示：





gateway和virtualSevice host添加‘*’

```
  uid: 7aa3e9ad-509f-4cba-a070-4dc09b80b21f
spec:
  selector:
    istio: ingressgateway
  servers:
    - hosts:
        -
        - boutique.io
        - '*'
      port:
        name: http-8a3775b4
        number: 1025
        protocol: http
```

```
  uid: e1b71c9a-4db6-4d21-86e0-b78d280c3f21
spec:
  gateways:
    - istio-system/test-gw
  hosts:
    - '*'
  http:
    - delegate:
        name: frontend-80
        namespace: default
      match:
        - uri:
            prefix: /
```

步骤3 开启istio DNS访问

在A业务集群上使用如下命令

```
kubectl edit iop {iop名} -nistio-system
```

在spec.meshConfig.defaultConfig.proxyMetadata下如下添加配置

```
ISTIO_META_DNS_CAPTURE: "true"
ISTIO_META_DNS_AUTO_ALLOCATE: "true"
```

步骤4 修改访问地址，确保流量已经访问到新集群

1. 如使用springcloud，需要修改FeignClient访问地址，在注解中指定新url地址（producer.default:8089 -> b-mesh-gateway.default:8089，若第二步未操作，则直接替换为网关地址）

2. 创建灰度任务，切部分流量到修改了配置的新版本
3. 验证功能正常后将流量都切到新版本

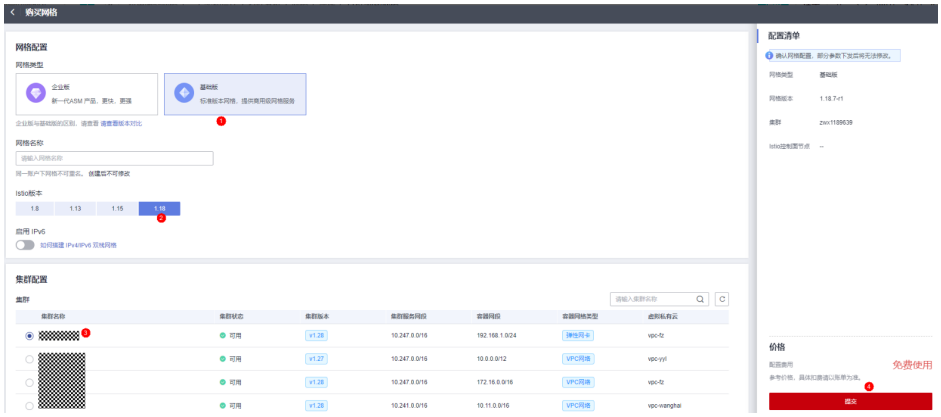
----结束

迁移集群

步骤1 删除该集群上部署的网关，可以在网关管理页面查看网关所属集群。

步骤2 将对应的集群移除出企业版网格。

步骤3 在ASM console ，创建基础版本网格，选择移除出企业版网格的集群。



步骤4 选择需要开始自动注入的命名空间（准备工作中记录的命名空间） -> 选择“重启已有服务”。

步骤5 进行服务诊断，并处理异常。

步骤6 恢复istio资源，选择如下任意一个方案执行。

方案一：重新手动创建网关，并添加路由。

方案二：将准备工作中备份的该集群资源上传到集群节点，执行如下命令恢复，其中xx为备份的文件名。

```
kubectl create -f xx.yaml
```

步骤7 （可选）执行解除业务跨集群访问方案二

步骤8 重复执行上述步骤1-7，直至所有集群迁移完毕。

----结束

功能验证

验证业务功能。