ALIBABA CLOUD

阿里云

服务网格

安全

文档版本：20220114

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.使用ASM网格审计

ASM的审计功能可以帮助网格管理人员记录或追溯不同用户的日常操作，是集群安全运维中的重要环节。本文介绍如何启用网格审计功能、查看审计报表、查看日志记录以及设置告警。

前提条件

开通日志服务。

背景信息

- 本文中所提及的资源指的是Istio资源，包括VirtualService、Gateway、DestinationRule、EnvoyFilter、Sidecar、ServiceEntry等。
- 审计功能开启后，审计日志会产生费用，计费方式请参见按量付费。

启用网格审计

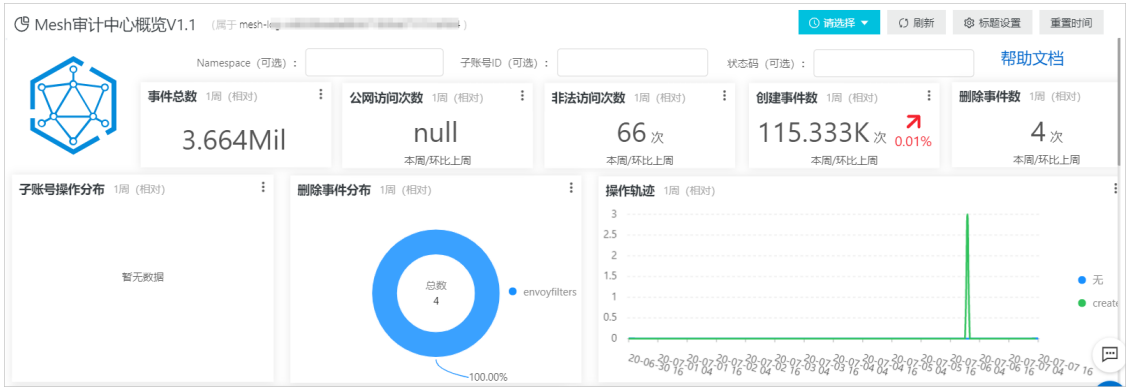
在创建ASM实例时，您可以在**创建新网格**对话框中启用网格审计功能，请参见创建ASM实例。

> **说明** 默认创建的审计LogProject名称为mesh-log-${mesh-id}，同时会在该LogProject中创建名为audit-${mesh-id}的LogStore用于存储审计日志。

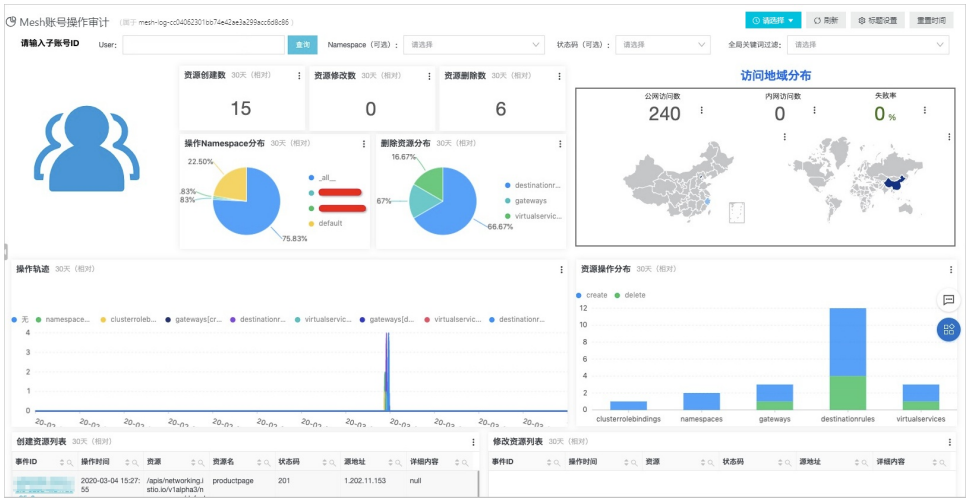
查看审计报表

ASM内置了四个审计报表，分别是审计中心概览、账号操作审计、资源操作概览、资源操作详细列表。

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. 在网格详情页面左侧导航栏，单击**网格审计**。
5. 在**网格审计**页面选中**启用网格审计**，然后单击**确定**。
6. 单击**审计中心概览**页签，查看网格实例的事件概览以及重要事件（如公网访问、命令执行、删除资源等）的详细信息。

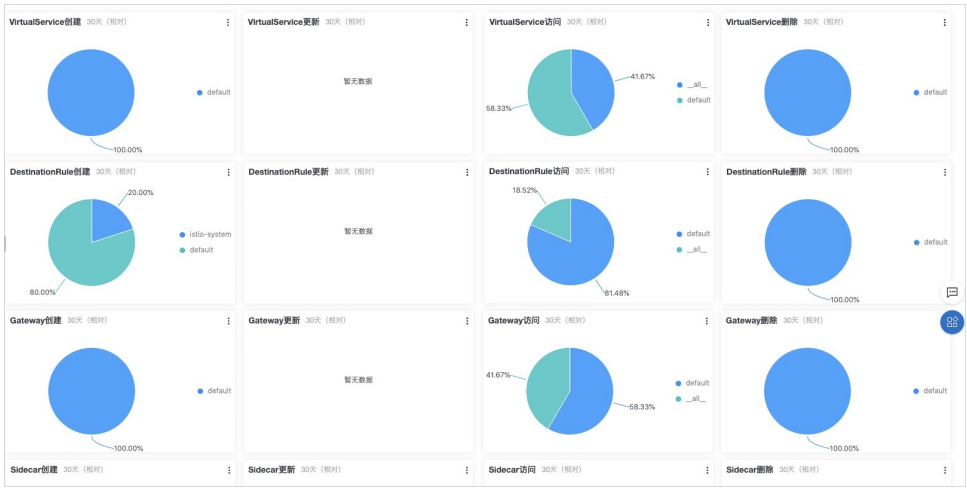


7. 单击**账号操作审计**页签，查看指定账号对网格实例操作的详细信息，包括对创建、修改和删除资源等操作的信息以及资源所属的命名空间分布、访问地理位置分布等信息。



8. 单击**资源操作概览**页签，查看网格实例中主要资源的操作统计信息。

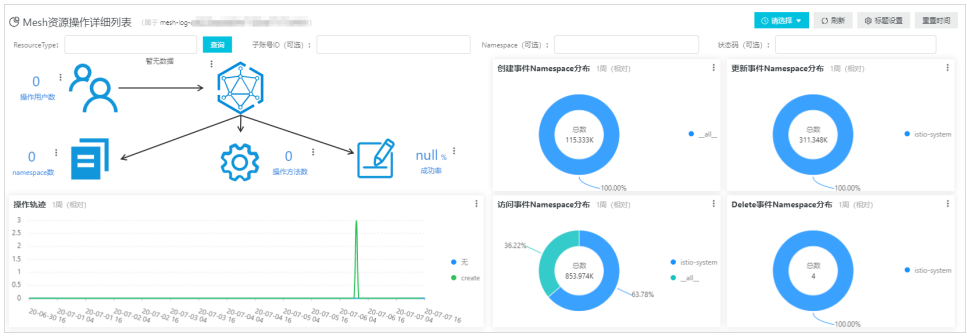
> **说明** 您可以执行以下操作来过滤筛选统计信息：
>
> - 自定义统计时间的范围，默认显示最近一周的统计信息。
> - 指定Namespace和子账号ID。
> - 选择一项或多项组合来筛选指定范围的事件。



9. 单击**资源操作详细列表**页签，查看网格实例中指定资源的详细操作列表。

   您需要选择或输入指定的资源类型进行实时查询，包括资源操作各类事件的总数、Namespace分布、成功率、时序趋势以及详细操作列表等。

> **说明** 您可以执行以下操作来过滤筛选统计信息：
>
> - 自定义统计时间的范围，默认显示最近一周的统计信息。
> - 指定Namespace和子账号ID。
> - 选择一项或多项组合来筛选指定范围的事件。



查看详细日志记录

如果您有自定义查询和分析审计日志的需求，可以进入日志服务管理控制台查看详细的日志记录。

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击名为mesh-log-${mesh-id}的日志Project。
3. 选择名称为audit-${mesh-id}的日志库，单击**查询分析**图标，查看对应的审计日志。

> **说明**
> - 在启用网格审计时，生成的日志project中会自动添加一个名为audit-${mesh-id}的日志库。
> - 审计日志的日志库默认已经配置索引。请不要修改索引，以免报表失效。

常见的审计日志搜索方式如下所示。

- 查询某一子账号的操作记录，直接在搜索框中输入子账号ID，单击**查询/分析**。
- 查询某一资源的操作，直接在搜索框中输入资源名，单击**查询/分析**。
- 过滤系统组件的操作，在搜索框中输入 `NOT user.username: node NOT user.username: serviceaccount NOT user.username: apiserver NOT user.username: kube-scheduler NOT user.username: kube-controller-manager` ，单击**查询/分析**。

更多查询和统计方式，请参见查询概述。

设置告警

若您需要对某些资源的操作进行实时告警，可以通过日志服务的告警功能实现。告警方式支持短信、钉钉机器人、邮件、自定义WebHook和通知中心。详情请参见简介。

关于审计日志的更多查询方式，您还可以通过审计报表中的查询语句来查询审计日志：

- 示例1：对容器执行命令时触发告警。

  某公司对于网格实例的使用有严格限制，不允许用户登录容器或对容器执行命令。如果有用户执行命令时需要立即给出告警，并希望告警时能够显示用户登录的具体容器、执行的命令、操作人、事件ID、时间、操作源IP等信息。

- 查询语句如下所示。

```
verb : create and objectRef.subresource:exec and stage:  ResponseStarted | SELECT audit
ID as "事件ID", date_format(from_unixtime(__time__), '%Y-%m-%d %T' ) as "操作时间",  reg
exp_extract("requestURI", '([^\?]*)/exec\?.*', 1)as "资源",  regexp_extract("requestURI
", '\?(.*)', 1)as "命令" ,"responseStatus.code" as "状态码",
 CASE
 WHEN "user.username" != 'kubernetes-admin' then "user.username"
 WHEN "user.username" = 'kubernetes-admin' and regexp_like("annotations.authorization.k
8s.io/reason", 'RoleBinding') then regexp_extract("annotations.authorization.k8s.io/rea
son", ' to User "(\w+)"', 1)
 ELSE 'kubernetes-admin' END
 as "操作账号",
CASE WHEN json_array_length(sourceIPs) = 1 then json_format(json_array_get(sourceIPs, 0
)) ELSE  sourceIPs END
as "源地址" limit 100
```

- 条件表达式如下所示。

```
操作事件 =~ ".*"
```

- 示例2：APIServer公网访问失败时触发告警。

  某网格实例开启了公网访问，为防止恶意攻击，需要监控公网访问的次数以及失败率。若访问次数到达一定阈值（例如10次）且失败率高于一定阈值（例如50%）则立即告警，并希望告警时能够显示用户的IP所属区域、操作源IP、是否高危IP等信息。

  - 查询语如下所示。

```
* | select ip as "源地址", total as "访问次数", round(rate * 100, 2) as "失败率%", failCo
unt as "非法访问次数", CASE when security_check_ip(ip) = 1 then 'yes' else 'no' end  as
"是否高危IP",  ip_to_country(ip) as "国家", ip_to_province(ip) as "省", ip_to_city(ip) as
"市", ip_to_provider(ip) as "运营商" from (select CASE WHEN json_array_length(sourceIPs)
= 1 then json_format(json_array_get(sourceIPs, 0)) ELSE  sourceIPs END
as ip, count(1) as total,
sum(CASE WHEN "responseStatus.code" < 400 then 0
ELSE 1 END) * 1.0 / count(1) as rate,
count_if("responseStatus.code" = 403) as failCount
from log  group by ip limit 10000) where ip_to_domain(ip) != 'intranet'  having "访问次
数" > 10 and "失败率%" > 50 ORDER by "访问次数" desc limit 100
```

  - 条件表达式如下所示。

```
源地址 =~ ".*"
```

重建网格审计

如果您误删了日志服务SLS中用于网格审计的Project，但是仍然想要使用网格审计功能，您需要重建用于网格审计的Project。

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. 在网格详情页面左侧导航栏，单击**网格审计**。

5. 在**网格审计**页面**重建网格审计**对话框中单击**重建**。

   重建后的Project名称为上一次Project名称基础上加上时间戳。

2.在ASM中实现自定义外部授权

ASM提供了自定义外部授权服务，在服务间互相通信时加入鉴权流程，以确保只有得到授权的情况下，才能访问关键服务。本文以httpbin应用为例，介绍如何实现自定义外部授权。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。

步骤一：创建外部授权应用

在集群中创建外部授权应用，该应用实现外部鉴权逻辑。本文使用的示例应用要求请求必须带有`x-ext-authz: allow`请求头，才能通过鉴权访问成功。

1. 通过kubectl工具连接集群。
2. 使用以下内容，创建*ext-authz.yaml*。

```
# Copyright Istio Authors
#
#   Licensed under the Apache License, Version 2.0 (the "License");
#   you may not use this file except in compliance with the License.
#   You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#   Unless required by applicable law or agreed to in writing, software
#   distributed under the License is distributed on an "AS IS" BASIS,
#   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#   See the License for the specific language governing permissions and
#   limitations under the License.
# Example configurations for deploying ext-authz server separately in the mesh.
apiVersion: v1
kind: Service
metadata:
  name: ext-authz
  labels:
    app: ext-authz
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 8000
  - name: grpc
    port: 9000
    targetPort: 9000
  selector:
    app: ext-authz
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ext-authz
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ext-authz
  template:
    metadata:
      labels:
        app: ext-authz
    spec:
      containers:
      - image: istio/ext-authz:0.6
        imagePullPolicy: IfNotPresent
        name: ext-authz
        ports:
        - containerPort: 8000
        - containerPort: 9000
---
```

3. 执行以下命令，在集群中部署外部授权服务。

```
kubectl apply -f ext-authz.yaml
```

预期输出：

```
service/ext-authz created
deployment.apps/ext-authz created
```

4. 执行以下命令， 验证应用是否正常工作。

```
kubectl logs "$(kubectl get pod -l app=ext-authz -n default -o jsonpath={.items..metada
ta.name})" -n default -c ext-authz
```

预期输出：

```
2021/01/07 22:55:47 Starting HTTP server at [::]:8000
2021/01/07 22:55:47 Starting gRPC server at [::]:9000
```

返回以上结果，说明外部授权服务部署成功。

5. 获取ext-authz应用的GRPC协议端口。
   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**详情**。
   iv. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
   v. 在**服务**页面单击ext-authz。
   在**端点**区域可以看到GRPC协议的端口为9000。

步骤二：创建示例应用

1. 使用以下内容，创建*httpbin.yaml*。

```
# Copyright Istio Authors
#
#   Licensed under the Apache License, Version 2.0 (the "License");
#   you may not use this file except in compliance with the License.
#   You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#   Unless required by applicable law or agreed to in writing, software
#   distributed under the License is distributed on an "AS IS" BASIS,
#   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#   See the License for the specific language governing permissions and
#   limitations under the License.
##################################################################################################
# httpbin service
##################################################################################################
apiVersion: v1
kind: ServiceAccount
metadata:
```

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

2. 执行以下命令，在集群中部署httpbin应用。

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

3. 使用以下内容，创建*sleep.yaml*。

```
# Copyright Istio Authors
#
#   Licensed under the Apache License, Version 2.0 (the "License");
#   you may not use this file except in compliance with the License.
#   You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
```

```
#
#   Unless required by applicable law or agreed to in writing, software
#   distributed under the License is distributed on an "AS IS" BASIS,
#   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#   See the License for the specific language governing permissions and
#   limitations under the License.
##################################################################################################
# Sleep service
##################################################################################################
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sleep
---
apiVersion: v1
kind: Service
metadata:
  name: sleep
  labels:
    app: sleep
    service: sleep
spec:
  ports:
  - port: 80
    name: http
  selector:
    app: sleep
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sleep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sleep
  template:
    metadata:
      labels:
        app: sleep
    spec:
      terminationGracePeriodSeconds: 0
      serviceAccountName: sleep
      containers:
      - name: sleep
        image: curlimages/curl
        command: ["/bin/sleep", "3650d"]
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - mountPath: /etc/sleep/tls
          name: secret-volume
```

```
      volumes:
      - name: secret-volume
        secret:
          secretName: sleep-secret
          optional: true
---
```

4. 执行以下命令，在集群中部署sleep应用。

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

步骤三：管理外部授权服务

您需要将步骤一创建的应用声明到服务网格中，使服务网格可以使用该应用进行鉴权。

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. 在网格详情页面左侧导航栏选择**零信任安全 > 授权策略**。
5. 在**授权策略**页面单击**管理外部授权服务**，单击**创建**。
6. 在**创建外部授权**页面设置参数，然后单击**创建**。

参数	描述
名称	自定义外部授权服务名称，本文设置为test。
协议	选择外部授权应用的协议，本文设置为GRPC。
服务地址	输入外部授权应用的服务地址 `<应用名称>.<命名空间名称>.svc.<集群域名>` ，本文设置为 `ext-authz.default.svc.cluster.local` 。
服务端口	输入外部授权应用的服务端口，本文设置为9000。
超时时间	如果鉴权应用未在该时间内返回，则认为鉴权服务不可用，本文设置为10秒。

步骤四：创建授权策略

您需要创建授权策略来配置需要鉴权的请求操作。

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. 在网格详情页面左侧导航栏选择**零信任安全 > 授权策略**。
5. 在**授权策略**页面单击**创建**。
6. 在创建页面配置参数，然后单击**确定**。

参数	描述
命名空间	选择示例应用部署的命名空间，本文设置为default。
名称	自定义授权策略名称，本文设置为test1。
策略	设置授权策略，本文设置为RULES。
动作	设置授权动作，本文设置为CUSTOM。
Provider Name	选择外部授权服务，本文设置为test。
工作负载标签选择	设置授权策略生效的工作负载。 打开**工作负载标签选择**开关，单击**新增匹配标签**，设置**名称**为app，**值**为httpbin。
请求操作	设置需要鉴权的请求操作。 打开**请求操作**开关，单击**增加请求操作到列表中**，单击**增加请求操作**，设置**请求操作域**为path，**值**为/headers。

步骤五：验证自定义外部授权是否成功

1. 执行以下命令，访问 `httpbin.default:8000/ip` 。

   ```
   kubectl exec "$(kubectl get pod -l app=sleep -n default -o jsonpath={.items..metadata.name})" -c sleep -n default -- curl "http://httpbin.default:8000/ip" -s -o /dev/null -w "%{http_code}\n"
   ```

   预期输出：

   ```
   200
   ```

   返回以上结果，说明没有触发鉴权。

2. 执行以下命令，带有 `x-ext-authz: deny` 请求头访问 `httpbin.default:8000/headers` 。

   ```
   kubectl exec " $(kubectl get pod -l app=sleep -n default -o jsonpath={.items..metadata.name})" -c sleep -ndefault -- curl "http://httpbin.default:8000/headers" -H "x-ext-authz: deny" -s
   ```

   预期输出：

   ```
   denied by ext_authz for not found header `x-ext-authz: allow` in the request
   ```

   返回以上结果，说明触发鉴权，但是鉴权未通过。

3. 执行以下命令，带有 `x-ext-authz: allow` 请求头访问 `httpbin.default:8000/headers` 。

```
kubectl exec "$(kubectl get pod -l app=sleep -n default -o jsonpath={.items..metadata.name})" -c sleep -n default -- curl "http://httpbin.default:8000/headers" -H "x-ext-authz: allow" -s
```

预期输出：

```
{
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin:8000",
    "User-Agent": "curl/7.76.0-DEV",
    "X-B3-Parentspanid": "430f770aeb7ef215",
    "X-B3-Sampled": "0",
    "X-B3-Spanid": "60ff95c5acdf5288",
    "X-B3-Traceid": "fba72bb5765daf5a430f770aeb7e****",
    "X-Envoy-Attempt-Count": "1",
    "X-Ext-Authz": "allow",
    "X-Ext-Authz-Check-Result": "allowed",
    "X-Forwarded-Client-Cert": "By=spiffe://cluster.local/ns/default/sa/httpbin;Hash=e5178ee79066bfbafb1d98044fcd0cf80db76be8714c7a4b630c7922df520bf2;Subject=\"\";URI=spiffe://cluster.local/ns/default/sa/sleep"
  }
}
```

返回以上结果，说明触发鉴权，并且鉴权通过。

根据以上结果，可以看到访问 `httpbin.default:8000/headers` 时，请求中必须带有 `x-ext-authz: allow` 才能访问成功，说明自定义外部授权成功。

3.在ASM中使用OPA实现细粒度访问控制

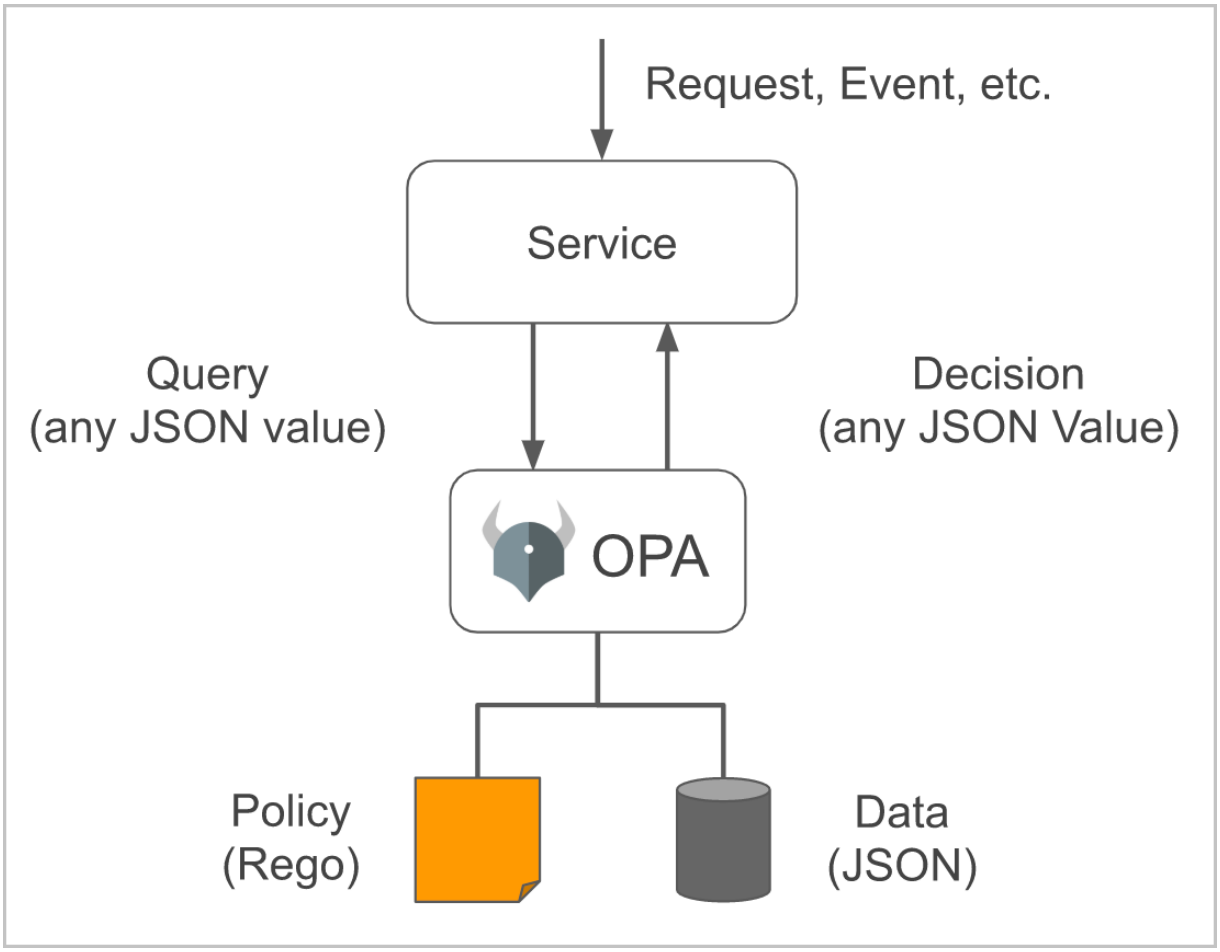
服务网格ASM集成了开放策略代理（OPA）插件，通过OPA定义访问控制策略，可以使您的应用实现细粒度的访问控制。ASM支持在控制平面定义OPA策略，然后推送到数据平面集群中。本文介绍如何在ASM定义OPA策略实现访问应用时的权限细粒度控制，包括对请求URL、请求头信息中的Token等访问限制。

前提条件

- 已创建ASM实例，且ASM实例为v1.9.7及以上版本。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。

背景信息

作为由CNCF托管的一个孵化项目，开放策略代理（OPA）是一个策略引擎，可用于为您的应用程序实现细粒度的访问控制。OPA作为通用策略引擎，可以与微服务一起部署为独立服务。为了保护应用程序，必须先授权对微服务的每个请求，然后才能对其进行处理。为了检查授权，微服务对OPA进行API调用，以确定请求是否被授权。



操作步骤

步骤一：启用OPA插件

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**基本信息**页面单击右上角的**功能设置**。
5. 在**功能设置更新**页面中选中**启用OPA插件**。
6. 单击**确定**。
   在**基本信息**页面可以看到**OPA插件**的状态变为**开启**。

步骤二：创建OPA策略

在ASM控制平面创建ASMOPAPolicy资源，该资源会被推送到数据平面集群，然后Pod中OPA引擎使用该资源实现细粒度访问控制。

使用ASM控制台创建OPA策略

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.
5. 在**新建**面板中设置**命名空间**为default，设置OPA策略的名称，单击**新增匹配标签**，设置**名称**为version，值为v1，将以下内容复制到文本框中，然后单击**确定**。

```
package istio.authz
import input.attributes.request.http as http_request
allow {
    roles_for_user[r]
    required_roles[r]
}
roles_for_user[r] {
    r := user_roles[user_name][_]
}
required_roles[r] {
    perm := role_perms[r][_]
    perm.method = http_request.method
    perm.path = http_request.path
}
user_name = parsed {
    [_, encoded] := split(http_request.headers.authorization, " ")
    [parsed, _] := split(base64url.decode(encoded), ":")
}
user_roles = {
    "guest1": ["guest"],
    "admin1": ["admin"]
}
role_perms = {
    "guest": [
        {"method": "GET",  "path": "/productpage"},
    ],
    "admin": [
        {"method": "GET",  "path": "/productpage"},
        {"method": "GET",  "path": "/api/v1/products"},
    ],
}
```

使用kubectl命令行创建OPA策略

1. 通过kubectl连接ASM实例。
2. 使用以下内容创建*opa.yaml*。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMOPAPolicy
metadata:
  name: bookinfo-opa
  namespace: default
spec:
  workloadSelector:
     labels:
       version: v1
  policy: |
    package istio.authz
    import input.attributes.request.http as http_request
    allow {
        roles_for_user[r]
        required_roles[r]
    }
    roles_for_user[r] {
        r := user_roles[user_name][_]
    }
    required_roles[r] {
        perm := role_perms[r][_]
        perm.method = http_request.method
        perm.path = http_request.path
    }
    user_name = parsed {
        [_, encoded] := split(http_request.headers.authorization, " ")
        [parsed, _] := split(base64url.decode(encoded), ":")
    }
    user_roles = {
        "guest1": ["guest"],
        "admin1": ["admin"]
    }
    role_perms = {
        "guest": [
            {"method": "GET",  "path": "/productpage"},
        ],
        "admin": [
            {"method": "GET",  "path": "/productpage"},
            {"method": "GET",  "path": "/api/v1/products"},
        ],
    }
```

> **说明**
>
> - OPA定义一个Pod的OPA策略时，只允许有一条 `default allow` 字段。如果多个OPA策略生效于一个Pod，且每个策略中都定义了 `default allow` 字段，则多条 `default allow` 字段会导致动态更新失败。
> - 设定OPA策略时，建议通过标签控制策略生效的范围，不恰当的Rego规则会导致无法访问服务。
> - OPA执行引擎和业务容器在一个Pod中，需要占用15081和9191端口。
> - OPA Policy中已经默认设置 `default allow = false` ，不可以再重复设置 `default allow` ，否则会导致冲突。

- spec：使用Rego语法表述的具体规则。关于Rego语法的更多信息，请参见Rego。
- workloadSelector：设定该OPA策略在此命名空间生效范围，如果不设定，默认在全部Pod中生效，设定后只生效于符合标签条件的Pod。
- user_roles：为用户授予角色权限。本例设置guest1拥有guest角色权限，admin1拥有admin角色权限。
- role_perms：设置角色拥有的权限。本文设置guest角色可以通过*/productpage*访问应用，admin角色可以通过*/productpage*和*/api/v1/products*访问应用。

3. 执行以下命令，创建OPA策略。

```
kubectl apply -f opa.yaml
```

步骤三：注入OPA代理

部署示例应用Bookinfo到ASM实例，确认每个Pod都注入了OPA代理。

1. 部署示例应用Bookinfo到ASM实例，请参见部署应用到ASM实例。
2. 定义相应的Istio虚拟服务和入口网关，请参见定义Istio资源。
3. 检查每个应用是否都注入OPA代理。
   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**详情**。
   iv. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**。

v. 在**容器组**页面，从**命名空间**下拉列表中选择**default**，单击目标应用容器组的名称。

在**容器**页签下可以看到容器被注入了Sidecar代理（istio-proxy）和OPA代理（opa-istio）。依次检查每个应用的容器，确保每个容器都被注入了Sidecar代理和OPA代理。

步骤四：验证使用OPA定义访问控制策略是否成功

- 执行以下命令，可以看到guest1被授予guest角色，并且可以使用带有*/productpage*的URL访问应用，但不能使用带有*/api/v1/products*的URL访问应用。
  - 使用带有*/productpage*的URL访问应用。

    ```
    curl -X GET http://<入口网关服务的IP地址>/productpage --user guest1:password -I
    ```

    预期输出：

    ```
    HTTP/1.1 200 OK
    ```

  - 使用带有*/api/v1/products*的URL访问应用。

    ```
    curl -X GET http://<入口网关服务的IP地址>/api/v1/products --user guest1:password -I
    ```

    预期输出：

    ```
    HTTP/1.1 403 Forbidden
    ```

- 执行以下命令，可以看到admin1被授予admin角色，并且可以使用带有/productpage和/api/v1/products的URL访问应用。
  - 使用带有*/productpage*的URL访问应用。

    ```
    curl -X GET http://{{入口网关服务的IP地址}}/productpage --user admin1:password -I
    ```

    预期输出：

    ```
    HTTP/1.1 200 OK
    ```

- 使用带有*/api/v1/products*的URL访问应用。

  ```
  curl -X GET http://<入口网关服务的IP地址>/api/v1/products --user admin1:password -I
  ```

  预期输出：

  ```
  HTTP/1.1 200 OK
  ```

根据以上结果，可以看到OPA定义访问控制策略是成功的。

步骤五：动态更新OPA策略

执行以下命令，修改OPA策略。

```
kubectl edit asmopapolicy bookinfo-opa -n default
```

在返回结果中编辑OPA策略，使guest1同时用于guest和admin权限。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMOPAPolicy
metadata:
  name: bookinfo-opa
  namespace: default
spec:
  policy: |
    package istio.authz
    import input.attributes.request.http as http_request
    allow {
        roles_for_user[r]
        required_roles[r]
    }
    roles_for_user[r] {
        r := user_roles[user_name][_]
    }
    required_roles[r] {
        perm := role_perms[r][_]
        perm.method = http_request.method
        perm.path = http_request.path
    }
    user_name = parsed {
        [_, encoded] := split(http_request.headers.authorization, " ")
        [parsed, _] := split(base64url.decode(encoded), ":")
    }
    user_roles = {
        "guest1": ["guest", "admin"],
        "admin1": ["admin"]
    }
    role_perms = {
        "guest": [
            {"method": "GET",  "path": "/productpage"},
        ],
        "admin": [
            {"method": "GET",  "path": "/productpage"},
            {"method": "GET",  "path": "/api/v1/products"},
        ],
    }
```

- user_roles：为用户授予角色权限。本例设置guest1同时拥有guest和admin角色权限，admin1拥有admin角色权限。
- role_perms：设置角色拥有的权限。本例设置guest角色可以通过*/productpage*访问应用，admin角色可以通过*/productpage*和*/api/v1/products*访问应用。

步骤六：验证动态更新OPA策略是否成功

执行以下命令，可以看到guest1被授予了admin角色，此时可以使用/productpage和/api/v1/products访问应用。

- 使用带有*/productpage*的URL访问应用。

```
curl -X GET http://<入口网关服务的IP地址>/productpage --user guest1:password -I
```

预期输出：

```
HTTP/1.1 200 OK
```

- 使用带有*/api/v1/products*的URL访问应用。

```
curl -X GET http://<入口网关服务的IP地址>/api/v1/products --user guest1:password -I
```

预期输出：

```
HTTP/1.1 200 OK
```

在没有更新OPA策略之前，guest1只能使用带有/productpage的URL访问应用，但不能使用带有/api/v1/products的URL访问应用。更新OPA策略之后，guest1可以使用带有/productpage和/api/v1/products的URL访问应用。说明动态更新OPA策略成功。

场景示例

场景一：JWT请求授权

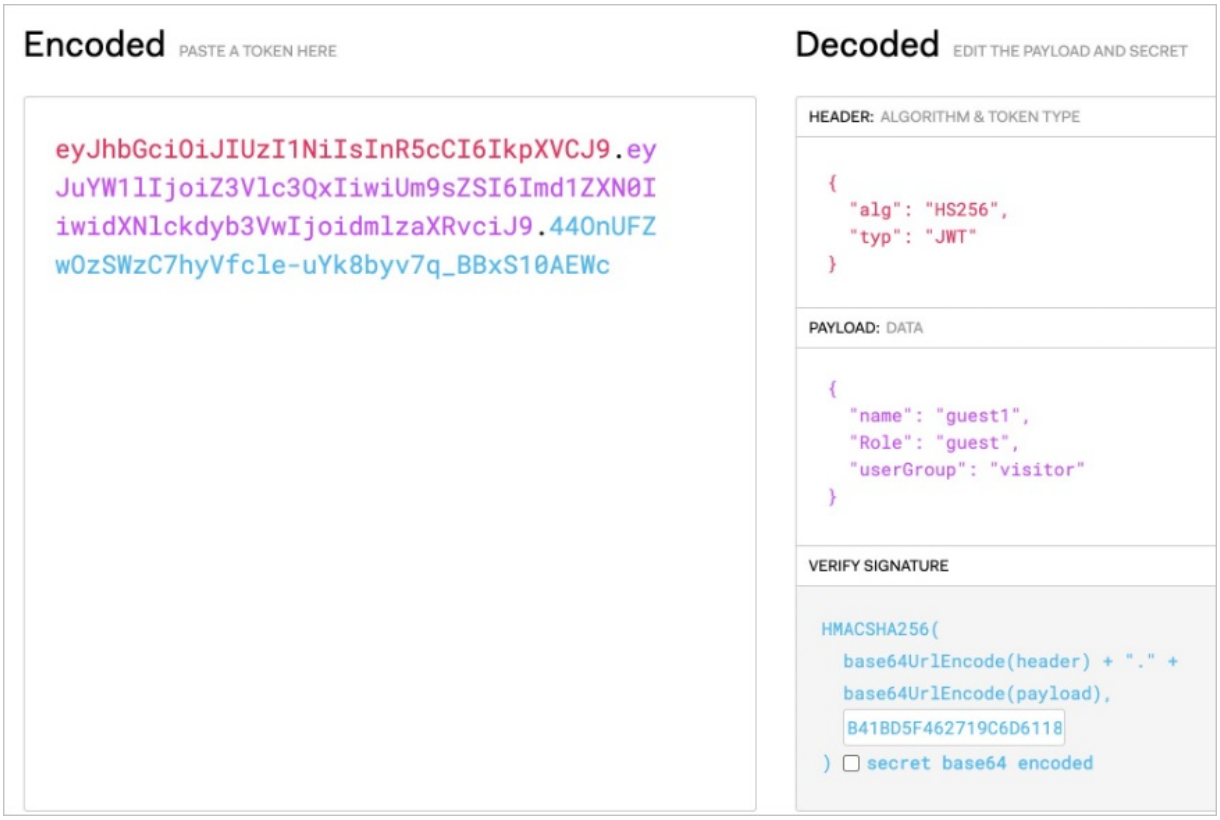
在接收用户请求时，认证请求头信息中的JWT Token是否可信，只有符合要求的请求才能访问到应用。

以下ASMOPAPolicy定义了只有通过get请求的方式，且JWT Token的`Role`为`guest`，`userGroup`是`visitor`的请求才能访问Productpage应用。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMOPAPolicy
metadata:
  name: policy-jwt
  namespace: default
spec:
  policy: |
    package istio.authz
    allow {
      input.attributes.request.http.method == "GET"
      input.parsed_path[0] == "productpage"
      # set certificate 'B41BD5F462719C6D6118E673A2389'
      io.jwt.verify_hs256(bearer_token, "B41BD5F462719C6D6118E673A2389")
      claims.Role == "guest"
      claims.userGroup == "visitor"
    }
    claims := payload {
       [_, payload, _] := io.jwt.decode(bearer_token)
    }
    bearer_token := t {
      v := input.attributes.request.http.headers.authorization
      startswith(v, "Bearer ")
      t := substring(v, count("Bearer "), -1)
    }
```

- input.attributes.request.http.method：请求方法，本文设置为`GET`。
- input.parsed_path[0]：访问的应用。
- claims.Role：对JWT Token进行限制，本文设置`Role`为`guest`。
- claims.userGroup：对JWT Token进行限制，本文设置`userGroup`为`visitor`。

使用JWT工具将 `Role` 、 `userGroup` 等请求信息编码成JWT字符串。



执行以下命令，访问Productpage应用。

```
curl --location --request GET 'http://{入口网关服务的IP地址服务IP}/productpage' \
--header 'Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiZ3Vlc3QxIiwiUm9sZSI6Imd1ZXN0IiwidXNlckdyb3VwIjoidmlzaXRvciJ9.44OnUFZwOzSWzC7hyVfcle-uYk8byv7q_BBxS10AEWc'
```

预期输出：

```
200
```

返回200，说明通过get请求的方式，且JWT Token的 `Role` 为 `guest` ， `userGroup` 是 `visitor` 的请求访问Productpage应用成功。如果您使用错误的JWT Token，或者请求中不包含JWT Token，将会返回403，说明访问Productpage应用失败。

场景二：对HTTP请求的请求体body进行限制

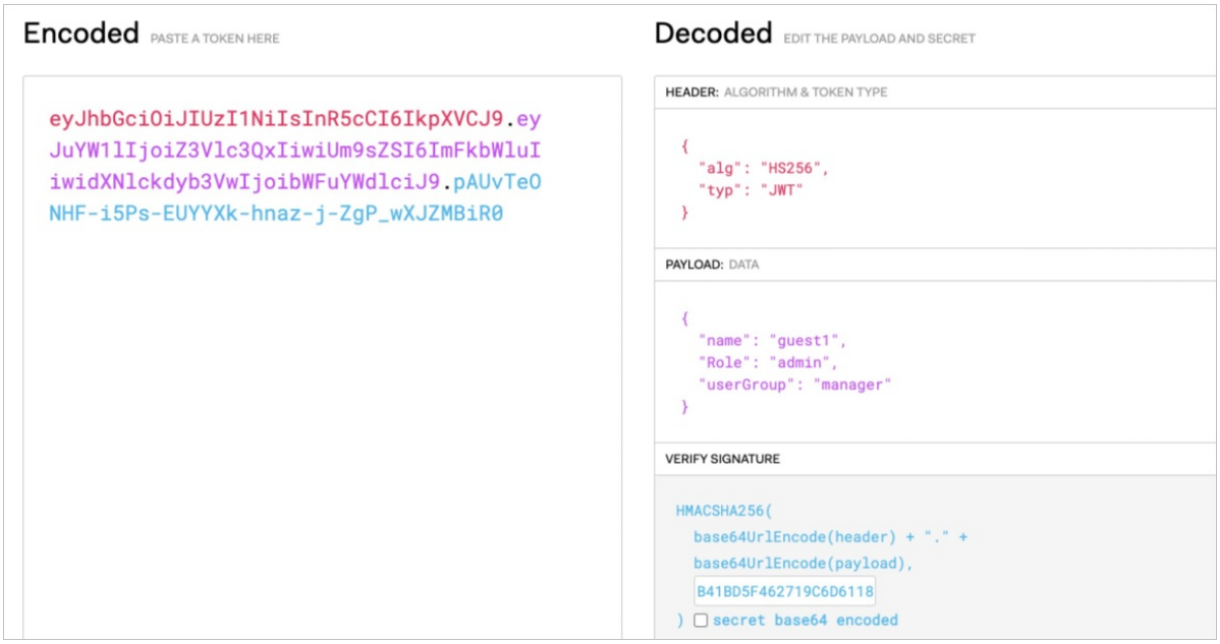
对HTTP请求的请求体body进行限制，只有请求体body与JWT中的Role相同的请求才能访问应用。

以下ASMOPAPolicy定义了只有通过get请求的方式，且请求体body的 `username` 与JWT中的 `Role` 相同，并且 `userGroup` 为 `manager` 的请求才能访问Productpage应用。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMOPAPolicy
metadata:
  name: policy-body
  namespace: default
spec:
  policy: |
    package istio.authz
    allow {
      input.attributes.request.http.method == "GET"
      input.parsed_path[0] == "productpage"
      io.jwt.verify_hs256(bearer_token, "B41BD5F462719C6D6118E673A2389")
      claims.Role == input.parsed_body.username
      claims.userGroup == "manager"
    }
    claims := payload {
       [_, payload, _] := io.jwt.decode(bearer_token)
    }
    bearer_token := t {
      v := input.attributes.request.http.headers.authorization
      startswith(v, "Bearer ")
      t := substring(v, count("Bearer "), -1)
    }
```

- input.attributes.request.http.method：请求方法，本文设置为 `GET` 。
- input.parsed_path[0]：访问的应用。
- claims.Role：对JWT Token进行限制，本文设置 `input.parsed_body.username` ，限制请求体body的 `username` 与JWT中的Role必须相同。
- claims.userGroup：对JWT Token进行限制，本文设置 `userGroup` 为 `manager` 。

使用JWT工具将 `Role` 、 `userGroup` 等请求信息编码成JWT字符串。



执行以下命令，访问Productpage应用。

```
curl --location --request GET 'http://{入口网关服务的IP地址}/productpage' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiZ3Vlc3QxIiwiUm9sZSI6ImFkbWluIiwidXNlckdyb3VwIjoibWFuYWdlciJ9.pAUvTeONHF-i5Ps-EUYYXk-hnaz-j-ZgP_wXJZMBiR0' \
--header 'Content-Type: application/json' \
--header 'Cookie: session=eyJ1c2VyIjoiYWRtaW4ifQ.YRz90g.GT34_5BqlFTwGqabZk_qGZzxYQ0' \
--data-raw '{
    "username":"admin",
    "password":"12****"
```

预期输出：

```
200
```

返回200，说明通过get请求的方式，请求体body的 `username` 与JWT中的 `Role` 相同，并且JWT Token中的 `userGroup` 为 `manager` 的请求访问Productpage应用成功。如果您使用错误的JWT Token，或者请求中不包含JWT Token，将会返回403，说明访问Productpage应用失败。

场景三：限制更多上下文信息

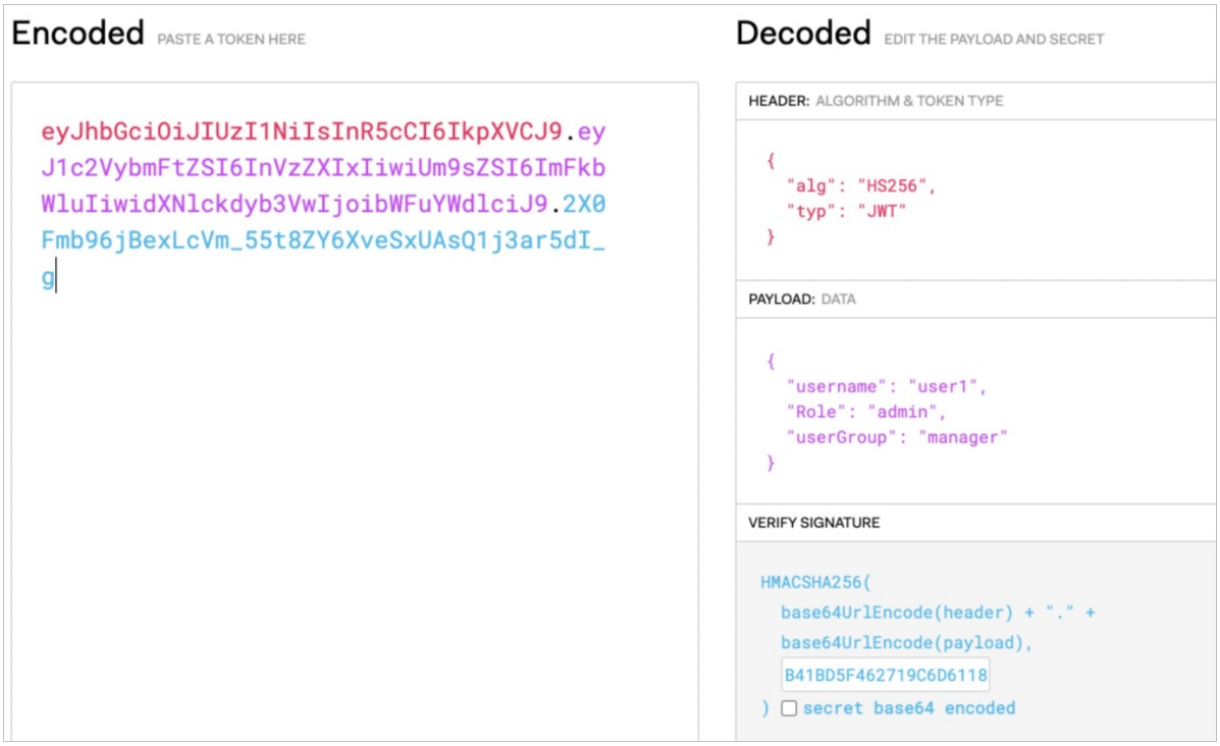
在场景二的基础上限制更多上下文信息，限制JWT中的 `username` 信息必须包含在 `bookinfo_managers` 范围内。

以下ASMOPAPolicy定义了只有通过get请求的方式，且请求体body的 `username` 与JWT中的 `Role` 相同，JWT中的 `username` 信息必须包含在 `bookinfo_managers` 范围内， `userGroup` 为 `manager` 的请求才能访问Productpage。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMOPAPolicy
metadata:
  name: policy-range
  namespace: default
spec:
  policy: |
    package istio.authz
    bookinfo_managers = [{"name": "user1"}, {"name": "user2"}, {"name": "user3"}]
    allow {
      input.attributes.request.http.method == "GET"
      input.parsed_path[0] == "productpage"
      io.jwt.verify_hs256(bearer_token, "B41BD5F462719C6D6118E673A2389")
      claims.Role == input.parsed_body.username
      claims.userGroup == "manager"
      claims.username == bookinfo_managers[_].name
    }
    claims := payload {
       [_, payload, _] := io.jwt.decode(bearer_token)
    }
    bearer_token := t {
      v := input.attributes.request.http.headers.authorization
      startswith(v, "Bearer ")
      t := substring(v, count("Bearer "), -1)
    }
```

- input.attributes.request.http.method：请求方法，本文设置为 `GET` 。
- input.parsed_path[0]：访问的应用。
- claims.Role：对JWT Token进行限制，本文设置 `input.parsed_body.username` ，限制请求体body的username与JWT中的Role必须相同。
- claims.userGroup：对JWT Token进行限制，本文设置 `userGroup` 为 `manager` 。
- claims.username：对JWT Token进行限制，本文设置 `bookinfo_managers[_].name` ，限制JWT中的 `username` 信息必须包含在 `bookinfo_managers` 范围内。

使用JWT工具将请求信息编码成JWT字符串。



执行以下命令，访问productpage应用。

```
curl --location --request GET 'http://{入口网关服务的IP地址}/productpage' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InVzZXIxIiwiUm9sZSI6ImFkbWluIiwidXNlckdyb3VwIjoibWFuYWdlciJ9.2X0Fmb96jBexLcVm_55t8ZY6XveSxUAsQ1j3ar5dI_g' \
--header 'Content-Type: application/json' \
--header 'Cookie: session=eyJ1c2VyIjoiYWRtaW4ifQ.YRz90g.GT34_5BqlFTwGqabZk_qGZzxYQ0' \
--data-raw '{
    "username":"admin",
    "password":"12****"
}'
```

预期输出：

```
200
```

返回200，说明通过get请求的方式，请求体body的 `username` 与JWT中的 `Role` 相同，JWT中的 `username` 信息必须包含在 `bookinfo_managers` 范围内。并且JWT Token中的 `userGroup` 为 `manager` 的请求访问Productpage应用成功。如果您使用错误的JWT Token，或者请求中不包含JWT Token，将会返回403，说明访问Productpage应用失败。

FAQ

如何查看哪些Pod使用OPA策略？

OPA的执行引擎以Sidecar的方式和业务容器部署在同一个Pod中。您需要进入到Pod中，执行以下命令，可以看到应用在Pod上的全部OPA策略。

```
curl 127.0.0.1:15081/v1/policies
```

如何对Rego语法进行测试？

OPA Policy Agent官方提供了在线测试工具，您可以使用该工具对Rego编写的策略进行测试。

相关文档

如果您之前使用的是Configmap配置OPA策略，您可以参考以下文档：

- 在ASM中使用OPA定义细粒度访问控制
- 在ASM中实现动态更新OPA策略

4.在ASM中实现JWT请求授权

在服务网格中配置JWT（JSON Web Token）请求授权，可以实现来源认证，又称为最终用户认证。在接收用户请求时，该配置用于认证请求头信息中的Access Token是否可信，并授权给来源合法的请求。本文介绍如何在ASM中实现JWT请求授权。

前提条件

- 请确保Istio版本≥1.6，否则将不支持RequestAuthentication功能。
- 已创建至少一个ASM实例，并添加至少一个ACK集群到该实例中。详情请参见添加集群到ASM实例。

背景信息

服务网格包含两种认证方式：

- 传输认证：基于双向TLS技术，常用于服务间通信认证。
- 来源认证：基于JWT技术，常用于客户端和服务之间的请求认证。

本文介绍的JWT请求授权正是为了实现来源认证，JWT是一种用于双方之间传递安全信息的表述性声明规范。JWT的相关资料请参见JWT官方文档。

步骤一：部署示例服务

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. 在网格详情页面左侧导航栏单击**命名空间**，然后在右侧页面单击**新建**。
5. 在**新建命名空间**面板中，填写**名称**和**标签**。名称为 `foo` ，标签为 `istio-injection:enabled` 。表示创建命名空间为 `foo` ，并启用 `istio-injection` 。
6. 单击**确定**，完成 `foo` 的命名空间创建。
7. 在本地执行以下命令，部署官方示例服务 `httpbin` 和 `sleep` 。

```
kubectl \
  --kubeconfig "$USER_CONFIG" \
  -n foo \
  apply -f "$ISTIO_HOME"/samples/httpbin/httpbin.yaml
kubectl \
  --kubeconfig "$USER_CONFIG" \
  -n foo \
  apply -f "$ISTIO_HOME"/samples/sleep/sleep.yaml
```

8. 执行如下命令，从而实现在Pod就绪前系统会一直等待。

```
kubectl --kubeconfig "$USER_CONFIG" -n foo get po
kubectl --kubeconfig "$USER_CONFIG" -n foo wait --for=condition=ready pod -l app=httpbi
n
kubectl --kubeconfig "$USER_CONFIG" -n foo wait --for=condition=ready pod -l app=sleep
```

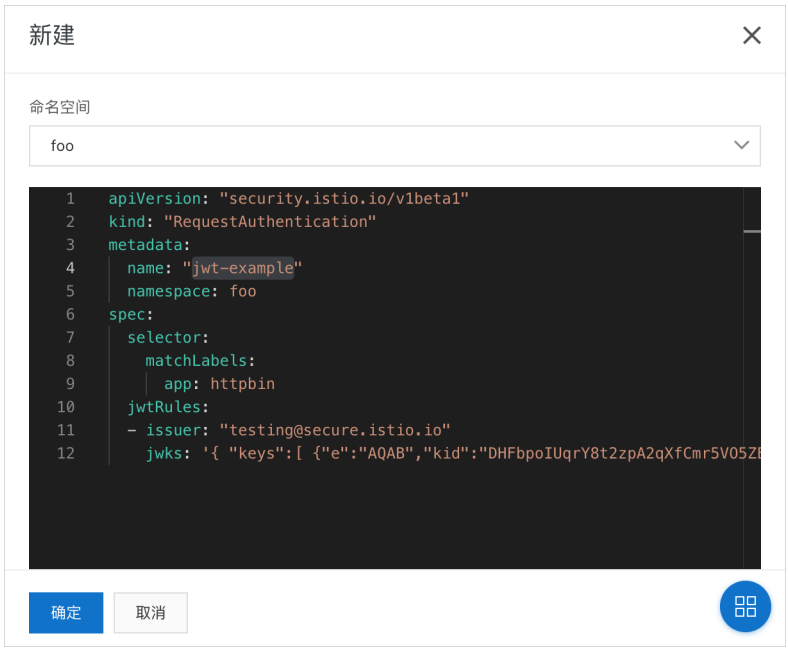
执行结果：

在 `sleep` 容器内，编辑YAML，验证是否可以请求 `httpbin` 。验证结果 `http_code` 值为200即请求成功。

```
sleep_pod=$(kubectl --kubeconfig "$USER_CONFIG" get pod -l app=sleep -n foo -o jsonpath
={.items..metadata.name})
RESULT=$(kubectl \
  --kubeconfig "$USER_CONFIG" \
  exec "$sleep_pod" -c sleep -n foo -- curl http://httpbin.foo:8000/ip -s -o /dev/null
-w "%{http_code}")
if [[ $RESULT != "200" ]]; then
  echo "http_code($RESULT) should be 200"
  exit
fi
```

步骤二：创建请求认证

1. 在网格详情页面左侧导航栏选择**零信任安全 > 请求身份认证**，然后在右侧页面单击**新建**。
2. 在**新建**面板，**命名空间**选择**foo**，然后在文本框中输入请求身份认证的YAML内容。



jwt-example.yaml示例如下：

```
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  jwtRules:
  - issuer: "testing@secure.istio.io"
    jwks: '{ "keys":[ {"e":"AQAB","kid":"DHFbpoIUqrY8t2zpA2qXfCmr5VO5ZEr4RzHU_-envvQ","
kty":"RSA","n":"xAE7eB6qugXyCAG3yhh7pkDkT65pHymX-P7KfIupjf59vsdo91bSP9C8H07pSAGQO1MV_xF
j9VswgsCg4R6otmg5PV2He95lZdHtOcU5DXIg_pbhLdKXbi66GlVeK6ABZOUW3WYtnNHD-91gVuoeJT_DwtGGcp
4ignkgXfkiEm4sw-4sfb4qdt5oLbyVpmW6x9cfa7vs2WTfURiCrBoUqgBo_-4WTiULmmHSGZHOjzwa8WtrtOQGs
AFjIbno85jp6MnGGGZPYZbDAa_b3y5u-YpW7ypZrvD8BgtKVjgtQgZhLAGezMt0ua3DRrWnKqTZ0BJ_EyxOGuHJ
rLsn00fnMQ"}]}'
```

> **说明**
>
> 这段YAML实现的策略为：当请求 `httpbin` 服务时，需匹配 `jwtRules` 中定义的规则，即请求头中如果包含Access Token信息，解码后的 `iss` 的值必须为 `testing@secure.istio.io` 。 `jwks` 中定义了Token生成的相关信息，详情请参见JWT官方文档。

3. 单击**确定**。
   执行结果：

   请求头中包含合法的Access Token时返回状态码为200，否则返回状态码为401。

   - 返回状态码为200如下所示。

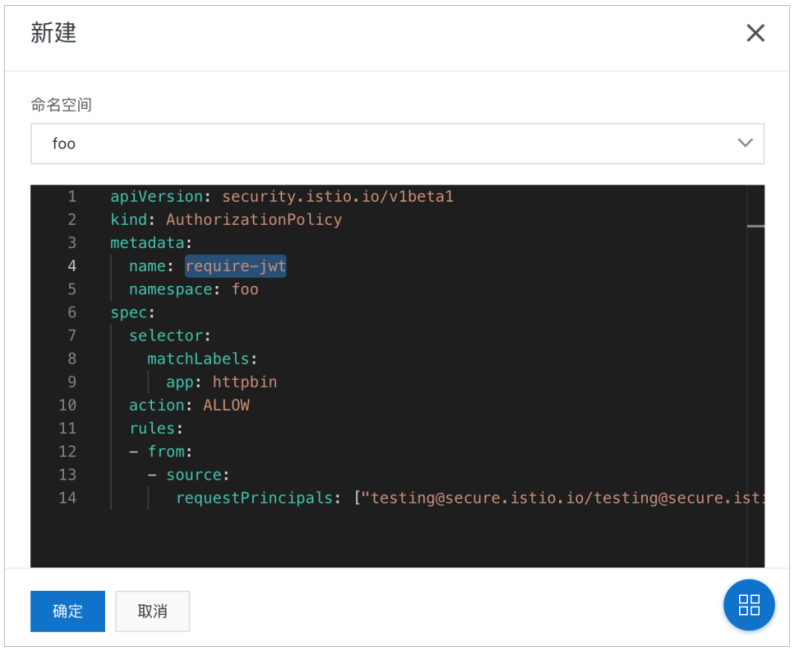
```
for ((i = 1; i <= 10; i++)); do
  RESULT=$(kubectl \
    --kubeconfig "$USER_CONFIG" \
    exec "$sleep_pod" \
    -c sleep \
    -n foo \
    -- curl "http://httpbin.foo:8000/headers" \
    -s \
    -o /dev/null \
    -w "%{http_code}")
  if [[ $RESULT != "200" ]]; then
    echo "http_code($RESULT) should be 200"
    exit
  fi
done
```

   - 返回状态码为401如下所示。

```
for ((i = 1; i <= 5; i++)); do
    RESULT=$(kubectl \
      --kubeconfig "$USER_CONFIG" \
      exec "$sleep_pod" \
      -c sleep \
      -n foo \
      -- curl "http://httpbin.foo:8000/headers" \
      -s \
      -o /dev/null \
      -H "Authorization: Bearer invalidToken" \
      -w "%{http_code}")
    if [[ $RESULT != "401" ]]; then
      echo "http_code($RESULT) should be 401"
      exit
    fi
done
```

步骤三：创建JWT授权策略

1. 在网格详情页面左侧导航栏选择**零信任安全 > 授权策略**，然后在右侧页面单击**新建**。
2. 在**新建**面板，**命名空间**选择**foo**，然后在文本框中输入授权策略的YAML内容。



require-jwt.yaml示例如下：

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  action: ALLOW
  rules:
  - from:
    - source:
       requestPrincipals: ["testing@secure.istio.io/testing@secure.istio.io"]
```

> **说明**
>
> 这段YAML实现的策略为：请求 `httpbin` 服务时，只有请求头Token解码后，符合 `iss` 的值+ `/` + `sub` 的值（即 `source.requestPrincipals` ）为 `testing@secure.istio.io/testing@secure.istio.io` ，请求权限才为 `ALLOW` 。

- Token信息如下所示。

```
TOKEN='eyJhbGciOiJSUzI1NiIsImtpZCI6IkRIRmJwb0lVcXJZOHQyenBBMnFYZkNtcjVWTzVaRXI0UnpIVV
8tZW52dlEiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjQ2ODU5ODk3MDAsImZvbyI6ImJhciIsImlhdCI6MTUzMjM4
OTcwMCwiaXNzIjoidGVzdGluZ0BzZWN1cmUuaXN0aW8uaW8iLCJzdWIiOiJ0ZXN0aW5nQHNlY3VyZS5pc3Rpb
y5pbyJ9.CfNnxWP2tcnR9q0vxyxweaF3ovQYHYZl82hAUsn21bwQd9zP7c-LS9qd_vpdLG4Tn1A15NxfCjp5f
7QNBUo-KC9PJqYpgGbaXhaGx7bEdFWjcwv3nZzvc7M__ZpaCERdwU7igUmJqYGBYQ51vr2njU9ZimyKkfDe3a
xcyiBZde7G6dabliUosJvvKOPcKIWPccCgefSj_GNfwIip3-SsFdlR7BtbVUcqR-yv-XOxJ3Uc1MI0tz3uMii
ZcyPV7sNCU4KRnemRIMHVOfuvHsU60_GhGbiSFzgPTAa9WTltbnarTbxudb_YEOx12JiwYToeX0DCPb43W1tz
IBxgm8NxUg'
```

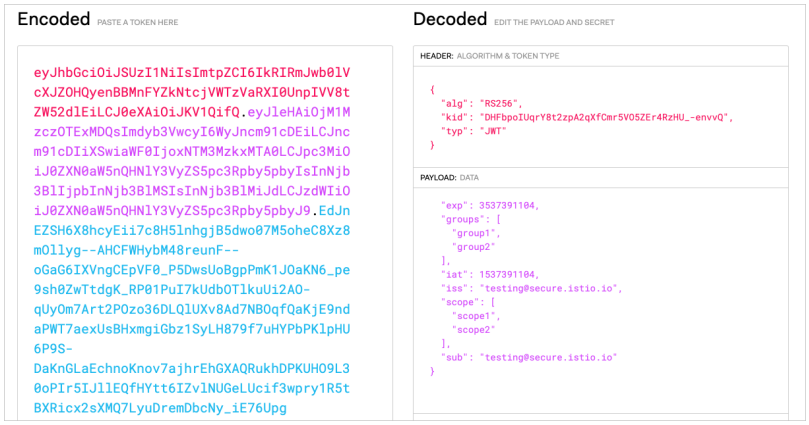
- 解析Token如下所示。

```
echo $TOKEN | cut -d '.' -f2 - | base64 --decode -
```

- 解析Token的输出信息如下所示。

```
{"exp":4685989700,"foo":"bar","iat":1532389700,"iss":"testing@secure.istio.io","sub":
"testing@secure.istio.io"}
```

JWT官网也提供了同样的解析Token能力，图形化输出如下图所示。



3. 单击**确定**。
   执行结果：

   请求头中包含合法的Access Token时返回状态码为200，否则返回状态码为403。

   - 返回状态码为200如下所示。

```
for ((i = 1; i <= 10; i++)); do
    RESULT=$(kubectl \
      --kubeconfig "$USER_CONFIG" \
      exec "$sleep_pod" \
      -c sleep \
      -n foo \
      -- curl "http://httpbin.foo:8000/headers" \
      -s \
      -o /dev/null \
      -H "Authorization: Bearer $TOKEN" \
      -w "%{http_code}")
    if [[ $RESULT != "200" ]]; then
      echo "http_code($RESULT) should be 200"
      exit
    fi
done
```

   - 返回状态码为403如下所示。

```
for ((i = 1; i <= 10; i++)); do
    RESULT=$(kubectl \
      --kubeconfig "$USER_CONFIG" \
      exec "$sleep_pod" \
      -c sleep \
      -n foo \
      -- curl "http://httpbin.foo:8000/headers" \
      -s \
      -o /dev/null \
      -w "%{http_code}")
    if [[ $RESULT != "403" ]]; then
      echo "http_code($RESULT) should be 403"
      exit
    fi
done
```

步骤四：追加JWT授权策略

1. 在网格详情页面左侧导航栏选择**零信任安全 > 授权策略**。
2. 在**授权策略**页面单击require-jwt**操作**列的**YAML**。
3. 在**编辑**对话框中追加补充以下内容。

   require-jwt-group.yaml补充的内容。

```
    when:
    - key: request.auth.claims[groups]
      values: ["group1"]
```

   以下为完整的require-jwt-group.yaml示例。

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  action: ALLOW
  rules:
  - from:
    - source:
       requestPrincipals: ["testing@secure.istio.io/testing@secure.istio.io"]
    when:
    - key: request.auth.claims[groups]
      values: ["group1"]
```

> **说明**
>
> 这段YAML实现的策略为：请求 `httpbin` 服务时，只有请求头Token解码后，符合 `groups` 的值包含group1，请求权限才为 `ALLOW` 。

   - Token信息如下所示。

```
TOKEN_GROUP='eyJhbGciOiJSUzI1NiIsImtpZCI6IkRIRmJwb0lVcXJZOHQyenBBMnFYZkNtcjVWTzVaRXI0
UnpIVV8tZW52dlEiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjM1MzczOTExMDQsImdyb3VwcyI6WyJncm91cDEiLC
Jncm91cDIiXSwiaWF0IjoxNTM3MzkxMTA0LCJpc3MiOiJ0ZXN0aW5nQHNlY3VyZS5pc3Rpby5pbyIsInNjb3B
lIjpbInNjb3BlMSIsInNjb3BlMiJdLCJzdWIiOiJ0ZXN0aW5nQHNlY3VyZS5pc3Rpby5pbyJ9.EdJnEZSH6X8
hcyEii7c8H5lnhgjB5dwo07M5oheC8Xz8mOllyg--AHCFWHybM48reunF--oGaG6IXVngCEpVF0_P5DwsUoBg
pPmK1JOaKN6_pe9sh0ZwTtdgK_RP01PuI7kUdbOTlkuUi2AO-qUyOm7Art2POzo36DLQlUXv8Ad7NBOqfQaKj
E9ndaPWT7aexUsBHxmgiGbz1SyLH879f7uHYPbPKlpHU6P9S-DaKnGLaEchnoKnov7ajhrEhGXAQRukhDPKUH
O9L30oPIr5IJllEQfHYtt6IZvlNUGeLUcif3wpry1R5tBXRicx2sXMQ7LyuDremDbcNy_iE76Upg'
```

   - 解析Token如下所示。

```
echo "$TOKEN_GROUP" | cut -d '.' -f2 - | base64 --decode - | jq
```

   - 解析Token的输出信息如下所示。

```
{
  "exp": 3537391104,
  "groups": [
    "group1",
    "group2"
  ],
  "iat": 1537391104,
  "iss": "testing@secure.istio.io",
  "scope": [
    "scope1",
    "scope2"
  ],
  "sub": "testing@secure.istio.io"
}
```

4. 单击**确定**。
   执行结果：

   请求头中包含合法的Access Token时返回状态码为200，否则返回状态码为403。

   - 返回状态码为200如下所示。

```
for ((i = 1; i <= 10; i++)); do
    RESULT=$(kubectl \
      --kubeconfig "$USER_CONFIG" \
      exec "$sleep_pod" \
      -c sleep \
      -n foo \
      -- curl "http://httpbin.foo:8000/headers" \
      -s \
      -o /dev/null \
      -H "Authorization: Bearer $TOKEN_GROUP" \
      -w "%{http_code}")
    if [[ $RESULT != "200" ]]; then
      echo "http_code($RESULT) should be 200"
      exit
    fi
done
```

   - 返回状态码为403如下所示。

```
for ((i = 1; i <= 10; i++)); do
    RESULT=$(kubectl \
      --kubeconfig "$USER_CONFIG" \
      exec "$sleep_pod" \
      -c sleep \
      -n foo \
      -- curl "http://httpbin.foo:8000/headers" \
      -s \
      -o /dev/null \
      -H "Authorization: Bearer $TOKEN" \
      -w "%{http_code}")
    if [[ $RESULT != "403" ]]; then
      echo "http_code($RESULT) should be 403"
      exit
    fi
done
```

5.使用实现访问控制

您可以使用授权策略对服务网格中的工作负载进行访问控制。本文介绍如何使用授权策略实现访问控制。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。

步骤一：在ACK集群中部署示例应用

1. 为default命名空间开启自动注入。具体操作，请参见启用自动注入。
2. 通过kubectl连接集群。具体操作，请参见通过kubectl工具连接集群。
3. 使用以下内容，创建名为*httpbin*的YAML文件。

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

4. 执行以下命令，创建httpbin应用。

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

5. 执行以下命令，验证Pod是否正常运行。

```
kubectl get pod |grep httpbin
```

预期输出：

```
httpbin-66cdbdb6c5-vhsh6          2/2     Running   0          11s
```

步骤二：保留请求方客户端源IP

> **说明** 可以使用两种方式保留请求客户端源IP：
>
> - 使用 `externalTrafficPolicy: Local` 保留请求方客户端源IP，在AuthorizationPolicy中使用 `ipBlocks` 设置授权策略。推荐使用此方式。
> - 使用X-Forwarded-For HTTP标头或代理协议保留请求方客户端源IP，在AuthorizationPolicy中使用 `remoteIpBlocks` 设置授权策略。

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. 在网格详情页面左侧导航栏单击**ASM网关**。
5. 单击ingressgateway**操作**列下的**YAML**。
6. 在**编辑**面板的文本框中输入 `externalTrafficPolicy: Local` ，然后单击**确定**。

```
spec:
  clusterIds:
    - ced3a48d2a3d442b99b4a0f8995f2ff5f
  compression: {}
  cpu: {}
  externalTrafficPolicy: Local
  kernel:
    parameters: {}
  maxReplicas: 5
```

7. 在ACK集群中执行以下命令，为入口网关打开RBAC调试日志。

```
kubectl exec -it -n istio-system <istio-ingressgateway Pod名称> -- curl -X POST localhost:15000/logging?rbac=debug
```

8. 在ACK集群中执行以下命令，获得请求客户端IP地址。

```
CLIENT_IP=$(kubectl get pods -n istio-system -o name -l istio=ingressgateway | sed 's|pod/||' | while read -r pod; do kubectl logs "$pod" -n istio-system | grep remoteIP; done | tail -1 | awk -F, '{print $3}' | awk -F: '{print $2}' | sed 's/ //') && echo "$CLIENT_IP"
```

步骤三：在ASM中设置路由规则

通过配置网关规则和虚拟服务实现任意请求都可访问httpbin应用。

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. 创建网关规则。
   i. 在网格详情页面左侧导航栏选择**流量管理 > 网关规则**，然后在右侧页面单击**新建**。
   ii. 在**新建**面板设置**命名空间**为default，将以下内容复制到文本框中，然后单击**确定**。

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

5. 创建虚拟服务。
   i. 在网格详情页面左侧导航栏选择**流量管理 > 虚拟服务**，然后在右侧页面单击**新建**。
   ii. 在**新建**面板设置**命名空间**为default，将以下内容复制到文本框中，然后单击**确定**。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
  - "*"
  gateways:
  - httpbin-gateway
  http:
  - route:
    - destination:
        host: httpbin
        port:
          number: 8000
```

6. 执行以下命令，验证是否可以正常访问httpbin应用。

```
curl "<入口网关的IP地址>:<入口网关的端口>"/headers -s -o /dev/null -w "%{http_code}\n"
```

> **说明** 关于入口网关IP地址获取方式，请参见定义Istio资源。

预期输出：

```
200
```

返回结果为200，说明访问成功。

步骤四：设置授权策略

设置授权策略实现指定的IP访问应用

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. 在网格详情页面左侧导航栏选择**零信任安全 > 授权策略**，然后在右侧页面单击**新建**。
5. 在**新建**面板设置**命名空间**为istio-system，将以下内容复制到文本框中，然后单击**确定**。

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
  - from:
    - source:
        ipBlocks: ["1.2.3.4", "5.6.7.0/24"]
```

设置 `action` 为 `ALLOW`，设置 `ipBlocks` 为 `["1.2.3.4", "5.6.7.0/24"]`，表示只有1.2.3.4和5.6.7.0/24可以访问httpbin应用。

6. 执行以下命令，使用请求客户端访问httpbin应用，验证1.2.3.4和5.6.7.0/24之外的IP地址是否可以访问httpbin应用。

```
curl "<入口网关的IP地址>:<入口网关的端口>"/headers -s -o /dev/null -w "%{http_code}\n"
```

预期输出：

```
403
```

返回结果为403，请求客户端访问应用失败。说明1.2.3.4和5.6.7.0/24之外的IP地址无法访问httpbin应用。

设置授权策略实现请求客户端访问应用

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. 在网格详情页面左侧导航栏选择**零信任安全 > 授权策略**，然后在右侧页面单击**新建**。
5. 在**新建**面板设置**命名空间**为istio-system，将以下内容复制到文本框中，然后单击**确定**。

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
  - from:
    - source:
        ipBlocks: ["1.2.3.4", "5.6.7.0/24", "$CLIENT_IP"]
```

设置 `action` 为 `ALLOW`，设置 `ipBlocks` 为 `["1.2.3.4", "5.6.7.0/24", "$CLIENT_IP"]`，表示只有1.2.3.4、5.6.7.0/24和CLIENT_IP可以访问httpbin应用。

6. 执行以下命令，验证请求客户端IP地址是否可以访问httpbin应用。

```
curl "<入口网关的IP地址>:<入口网关的端口>"/headers -s -o /dev/null -w "%{http_code}\n"
```

预期输出：

```
200
```

返回结果为200，请求客户端可以访问应用。

设置授权策略实现指定IP不可以访问应用

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. 在网格详情页面左侧导航栏选择**零信任安全 > 授权策略**，然后在右侧页面单击**新建**。
5. 在**新建**面板设置**命名空间**为istio-system，将以下内容复制到文本框中，然后单击**确定**。

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: DENY
  rules:
  - from:
    - source:
        ipBlocks: ["1.2.3.4", "5.6.7.0/24", "$CLIENT_IP"]
```

设置 `action` 为 `DENY`，设置 `ipBlocks` 为 `["1.2.3.4", "5.6.7.0/24", "$CLIENT_IP"]`，表示1.2.3.4、5.6.7.0/24和CLIENT_IP不可以访问httpbin应用。

6. 执行以下命令，验证请求客户端IP地址是否可以访问httpbin应用。

```
curl "<入口网关的IP地址>:<入口网关的端口>"/headers -s -o /dev/null -w "%{http_code}\n"
```

预期输出：

```
403
```

返回结果为403，请求客户端访问应用失败。