ALIBABA CLOUD

阿里云

服务网格

安全

文档版本：20201021

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. 在左侧导航栏中，选择**服务网格 > 网格审计**。
3. 在**网格审计**页面，从**网格**下拉列表中选择需要查看的网格实例。
4. 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。

   审计中心概览
5. 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。

   账号操作审计
6. 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。

   资源操作概览
7. 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。

资源操作详细列表

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，请参见查询简介。

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 auditID as "事件ID", date_format(from_unixtime(__time__), '%Y-%m-%d %T' ) as "操作时间",  regexp_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.k8s.io/reason", 'RoleBinding') then regexp_extract("annotations.authorization.k8s.io/reason", ' 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%", failCount 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 =~ ".*"
```

2.在ASM中实现自定义外部授权

服务网格各服务间的调用请求需要经过外部授权并通过授权过滤器的检查，才能给予响应。本文介绍了如何在ASM中实现自定义外部授权服务。

前提条件

- 创建至少一个ASM实例，并添加至少一个ACK集群到该实例中。
- 通过kubectl连接Kubernetes集群。
- 通过kubectl连接ASM实例。
- 部署应用到ASM实例
- 定义Istio资源。

背景信息

在服务网格中，服务间存在着调用请求。运行在网格外部的gRPC服务会根据制定的判断规则来决定是否对这些请求进行授权。然后，外部授权过滤器将调用授权服务来检查传入的请求是否被授权。如果外部授权过滤器发现某请求未被授权服务器授权，则该请求将被拒绝响应。

> 说明 建议将这些授权过滤器配置为过滤器链中的第一个过滤器，以便在其余过滤器处理请求之前确保该请求已被授权。

更多Envoy外部授权的内容，请参见External Authorization。

gRPC外部服务需要相应的接口，实现该 `Check()` 方法。请求响应上下文定义如下，详情请参见external_auth.proto。

```
// A generic interface for performing authorization check on incoming
// requests to a networked service.
service Authorization {
 // Performs authorization check based on the attributes associated with the
 // incoming request, and returns status `OK` or not `OK`.
 rpc Check(v2.CheckRequest) returns (v2.CheckResponse);
}
```

步骤一：实现外部授权服务

基于上述gRPC服务接口定义，本文以在 `Check()` 方法中判断Bearer Token值是否以 `asm-` 开头为例，实现外部授权服务如下。

> 说明 只要符合该接口定义，您还可以添加更为复杂的处理逻辑进行检查。

```
package main

import (
  "context"
  "log"
```

```
    "net"
    "strings"

    "github.com/envoyproxy/go-control-plane/envoy/api/v2/core"
    auth "github.com/envoyproxy/go-control-plane/envoy/service/auth/v2"
    envoy_type "github.com/envoyproxy/go-control-plane/envoy/type"
    "github.com/gogo/googleapis/google/rpc"
    "google.golang.org/grpc"
)

// empty struct because this isn't a fancy example
type AuthorizationServer struct{}

// inject a header that can be used for future rate limiting
func (a *AuthorizationServer) Check(ctx context.Context, req *auth.CheckRequest) (*auth.CheckRespo
nse, error) {
    authHeader, ok := req.Attributes.Request.Http.Headers["authorization"]
    var splitToken []string
    if ok {
        splitToken = strings.Split(authHeader, "Bearer ")
    }
    if len(splitToken) == 2 {
        token := splitToken[1]
        // Normally this is where you'd go check with the system that knows if it's a valid token.

        if strings.HasPrefix(token, "asm-") {
            return &auth.CheckResponse{
                Status: &rpc.Status{
                    Code: int32(rpc.OK),
                },
                HttpResponse: &auth.CheckResponse_OkResponse{
                    OkResponse: &auth.OkHttpResponse{
                        Headers: []*core.HeaderValueOption{
                            {
                                Header: &core.HeaderValue{
                                    Key:   "x-custom-header-from-authz",
                                    Value: "some value",
                                },
                            },
                        },
```

```
                },
            },
        }, nil
    }
  }
  return &auth.CheckResponse{
    Status: &rpc.Status{
      Code: int32(rpc.UNAUTHENTICATED),
    },
    HttpResponse: &auth.CheckResponse_DeniedResponse{
      DeniedResponse: &auth.DeniedHttpResponse{
        Status: &envoy_type.HttpStatus{
          Code: envoy_type.StatusCode_Unauthorized,
        },
        Body: "Need an Authorization Header with a character bearer token using asm- as prefix!",
      },
    },
  }, nil
}

func main() {
  // create a TCP listener on port 4000
  lis, err := net.Listen("tcp", ":4000")
  if err != nil {
    log.Fatalf("failed to listen: %v", err)
  }
  log.Printf("listening on %s", lis.Addr())

  grpcServer := grpc.NewServer()
  authServer := &AuthorizationServer{}
  auth.RegisterAuthorizationServer(grpcServer, authServer)

  if err := grpcServer.Serve(lis); err != nil {
    log.Fatalf("Failed to start server: %v", err)
  }

}
```

可以直接使用镜像： `registry.cn-beijing.aliyuncs.com/istio-samples/ext-authz-grpc-service:latest` 。或者可以基于以下Dockerfile构建镜像，具体代码参见istio_ext_authz_filter_sample。

步骤二：启动外部授权服务器

1. 进入Github项目库，下载部署YAML文件。
2. 通过kubectl连接到ASM实例中新添加的ACK集群，执行以下命令。

```
kubectl apply -n istio-system -f extauth-sample-grpc-service.yaml
```

看到以下输出显示部署成功。

```
service/extauth-grpc-service created
deployment.extensions/extauth-grpc-service created
```

步骤三：部署示例应用

1. 进入Github项目库，下载部署示例httpbin服务的YMAL文件。
2. 通过kubectl连接到ASM实例中新添加的ACK集群，执行以下命令。

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

3. 部署用于测试的客户端示例应用sleep。 进入Github项目库，下载部署示例sleep服务的YAML文件。
4. 通过kubectl连接到ASM实例中新添加的ACK集群，执行以下命令。

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

步骤四：定义EnvoyFilter

执行以下命令，定义Istio EnvoyFilter。

```
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
 # This needs adjusted to be the app name
 name: extauth-sample
spec:
 workloadSelector:
  labels:
   # This needs adjusted to be the app name
   app: httpbin

 # Patch the envoy configuration
 configPatches:

 # Adds the ext_authz HTTP filter for the ext_authz API
 - applyTo: HTTP_FILTER
  match:
   context: SIDECAR_INBOUND
```

```
    listener:
      name: virtualInbound
      filterChain:
        filter:
          name: "envoy.http_connection_manager"
  patch:
    operation: INSERT_BEFORE
    value:
      # Configure the envoy.ext_authz here:
      name: envoy.ext_authz
      config:
        grpc_service:
          # NOTE: *SHOULD* use envoy_grpc as ext_authz can use dynamic clusters and has connection p
ooling
          google_grpc:
            target_uri: extauth-grpc-service.istio-system:4000
            stat_prefix: ext_authz
          timeout: 0.2s
        failure_mode_allow: false
        with_request_body:
          max_request_bytes: 8192
          allow_partial_message: true
EOF
```

看到以下输出，表示过滤器已成功定义。

```
envoyfilter.networking.istio.io/extauth-sample created
```

步骤五：验证外部授权

1. 登录到Sleep Pod容器中，执行以下命令。

   ```
   export SLEEP_POD=$(kubectl get pod -l app=sleep -o jsonpath={.items..metadata.name})
   kubectl exec -it $SLEEP_POD -c sleep -- sh -c 'curl  http://httpbin:8000/headers'
   ```

   返回以下结果：

   ```
   Need an Authorization Header with a character bearer token using asm- as prefix!
   ```

   可以看到，示例应用程序的请求没有通过外部授权的许可，原因是请求头中并没有满足Bearer Token值以 `asm-` 开头。

2. 执行以下命令，在请求中添加以 `asm-` 开头的Bearer Token请求头。

```
export SLEEP_POD=$(kubectl get pod -l app=sleep -o jsonpath={.items..metadata.name})
kubectl exec -it $SLEEP_POD -c sleep -- sh -c 'curl -H "Authorization: Bearer asm-token1" http://ht
tpbin:8000/headers'
```

返回以下结果：

```
{
 "headers": {
  "Accept": "*/*",
  "Authorization": "Bearer asm-token1",
  "Content-Length": "0",
  "Host": "httpbin:8000",
  "User-Agent": "curl/7.64.0",
  "X-B3-Parentspanid": "dab85d9201369071",
  "X-B3-Sampled": "1",
  "X-B3-Spanid": "c29b18886e98a95f",
  "X-B3-Traceid": "66875d955ac13dfcdab85d9201369071",
  "X-Custom-Header-From-Authz": "some value"
 }
}
```

可以看到，示例应用程序的请求已通过外部授权的许可。

3.在ASM中使用OPA定义细粒度访问控制

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。

前提条件

- 创建至少一个ASM实例，并添加至少一个ACK集群到该实例中。
- 通过kubectl连接Kubernetes集群。
- 通过kubectl连接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。

步骤一：启用OPA插件

在创建ASM实例的时候可以启用OPA插件。如果创建时没有启用，按以下步骤启用该插件。

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**页面中勾选**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：部署OPA

部署业务Pod之前，需要部署OPA配置文件和OPA策略的配置项Configmap。

1. 通过kubectl连接到ASM实例中新添加的ACK集群，执行以下命令，部署OPA配置文件。

```
kubectl apply -n {替换成实际的namespace} -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
 name: opa-istio-config
data:
 config.yaml: |
  plugins:
   envoy_ext_authz_grpc:
    addr: :9191
    path: istio/authz/allow
EOF
```

2. 通过kubectl连接到ASM实例中新添加的ACK集群，执行以下命令，部署OPA策略。

```
kubectl apply -n {替换成实际的namespace} -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
 name: opa-policy
data:
 policy.rego: |  ###以下为示例策略定义，需要替换成实际的策略定义
   package istio.authz
   import input.attributes.request.http as http_request
   default allow = false
   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"},
      ],
   }
EOF
```

步骤三：注入OPA代理

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

1. 重新部署示例应用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. 登录容器服务控制台。
4. 在控制台左侧导航栏中，单击**集群**。
5. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**应用管理**。
6. 在工作负载页面单击**容器组**页签。
7. 在**容器组**页面，从**命名空间**下拉列表中选择**default**。
   此时Bookinfo应用的Pod状态应为**运行中**，并且每一个Pod内都被注入了Sidecar代理（istio-proxy）和OPA代理（opa-istio）。

执行结果

以上配置的OPA策略限制了对BookInfo的访问，定义如下所示。

- guest1被授予**guest**角色，并且可以访问*/productpage*但不能访问*/v1/api/products*。

```
curl -X GET http://{{入口网关服务的IP地址}}/productpage --user guest1:password -I
HTTP/1.1 200 OK
......
```

```
curl -X GET http://{{入口网关服务的IP地址}}/api/v1/products --user guest1:password -I
HTTP/1.1 403 Forbidden
......
```

- admin1被授予**admin**角色，并且可以访问*/productpage*和*/v1/api/products*。

```
curl -X GET http://{{入口网关服务的IP地址}}/productpage --user admin1:password -I
HTTP/1.1 200 OK
......
```

```
curl -X GET http://{{入口网关服务的IP地址}}/api/v1/products --user admin1:password -I
HTTP/1.1 200 OK
......
```

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**控制平面的Namespace**页签中，单击**新建**。
5. 在**新建命名空间**页面中，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=httpbin
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. 单击控制平面区域的RequestAuthentication页签，单击新建。
2. 在新建页面，命名空间选择foo。在命名空间 `foo` 中，编辑命名空间的YAML，从而新建RequestAuthentication CRD，创建请求认证。

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":"R
SA","n":"xAE7eB6qugXyCAG3yhh7pkDkT65pHymX-P7KfIupjf59vsdo91bSP9C8H07pSAGQO1MV_xFj9Vs
wgsCg4R6otmg5PV2He95lZdHtOcU5DXIg_pbhLdKXbi66GlVeK6ABZOUW3WYtnNHD-91gVuoeJT_DwtG
Gcp4ignkgXfkiEm4sw-4sfb4qdt5oLbyVpmW6x9cfa7vs2WTfURiCrBoUqgBo_-4WTiULmmHSGZHOjzwa8
WtrtOQGsAFjIbno85jp6MnGGGZPYZbDAa_b3y5u-YpW7ypZrvD8BgtKVjgtQgZhLAGezMt0ua3DRrWnK
qTZ0BJ_EyxOGuHJrLsn00fnMQ"}]}'
```

> **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**，完成请求认证的创建。
   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. 单击控制平面区域的AuthorizationPolicy页签，单击新建。
2. 在新建页面，命名空间选择foo。在命名空间 `foo` 中，编辑命名空间的YAML，从而新建AuthorizationPolicy CDR，创建JWT认证策略。

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
OTcwMCwiaXNzIjoidGVzdGluZ0BzZWN1cmUuaXN0aW8uaW8iLCJzdWIiOiJ0ZXN0aW5nQHNlY3VyZS5
pc3Rpby5pbyJ9.CfNnxWP2tcnR9q0vxyxweaF3ovQYHYZl82hAUsn21bwQd9zP7c-LS9qd_vpdLG4Tn1
A15NxfCjp5f7QNBUo-KC9PJqYpgGbaXhaGx7bEdFWjcwv3nZzvc7M__ZpaCERdwU7igUmJqYGBYQ51v
r2njU9ZimyKkfDe3axcyiBZde7G6dabliUosJvvKOPcKIWPccCgefSj_GNfwIip3-SsFdlR7BtbVUcqR-yv-X
OxJ3Uc1MI0tz3uMiiZcyPV7sNCU4KRnemRIMHVOfuvHsU60_GhGbiSFzgPTAa9WTltbnarTbxudb_YEO
x12JiwYToeX0DCPb43W1tzIBxgm8NxUg'
```

- 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@s
ecure.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。

JWT

3. 单击**确定**，完成JWT认证策略的创建。
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. 单击控制平面区域的AuthorizationPolicy页签。
2. 在AuthorizationPolicy页签单击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='eyJhbGciOiJSUzI1NiIsImtpZCI6IkRIRmJwb0lVcXJZOHQyenBBMnFYZkNtcjVWTzVaRXI
0UnpIVV8tZW52dlEiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjM1MzczOTExMDQsImdyb3VwcyI6WyJncm91cDEi
LCJncm91cDIiXSwiaWF0IjoxNTM3MzkxMTA0LCJpc3MiOiJ0ZXN0aW5nQHNlY3VyZS5pc3Rpby5pbyIsIn
Njb3BlIjpbInNjb3BlMSIsInNjb3BlMiJdLCJzdWIiOiJ0ZXN0aW5nQHNlY3VyZS5pc3Rpby5pbyJ9.EdJnEZSH
6X8hcyEii7c8H5lnhgjB5dwo07M5oheC8Xz8mOllyg--AHCFWHybM48reunF--oGaG6IXVngCEpVF0_P5D
wsUoBgpPmK1JOaKN6_pe9sh0ZwTtdgK_RP01PuI7kUdbOTlkuUi2AO-qUyOm7Art2POzo36DLQlUXv8
Ad7NBOqfQaKjE9ndaPWT7aexUsBHxmgiGbz1SyLH879f7uHYPbPKlpHU6P9S-DaKnGLaEchnoKnov7
ajhrEhGXAQRukhDPKUHO9L30oPIr5IJllEQfHYtt6IZvlNUGeLUcif3wpry1R5tBXRicx2sXMQ7LyuDremD
bcNy_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**，完成JWT授权策略的更新。
   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
```