ALIBABA CLOUD

Alibaba Cloud

Alibaba Cloud Service Mesh

Security

Document Version: 20220106

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Use the mesh audit feature in ASM

The mesh audit feature of Alibaba Cloud Service Mesh (ASM) allows you to record and trace routine operations of users. This is an important feature that ensures secure cluster O&M. This topic describes how to enable the mesh audit feature, view audit reports and logs, and set alerts.

Prerequisites

Log Service is activated.

Context

- Resources in this topic refer to Istio resources, including virtual services, Istio gateways, destination rules, envoy filters, sidecar proxies, and service entries.
- After you enable the mesh audit feature, you are charged for the audit logs that are generated. For more information about the billing method, see Pay-as-you-go.

Enable the mesh audit feature

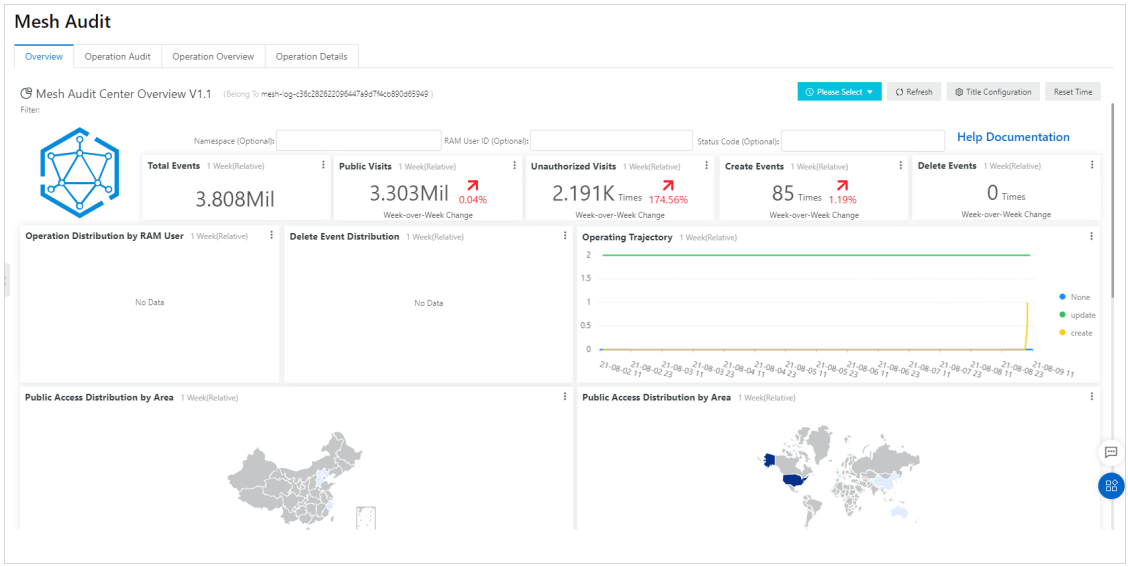
When you create an ASM instance, you can enable the mesh audit feature in the **Create ASM Instance** panel. For more information, see Create an ASM instance.

> **Note** By default, a project named **mesh-log-${mesh-id}** is created in Log Service, and a Logstore named **audit-${mesh-id}** is created in the project to store audit logs.

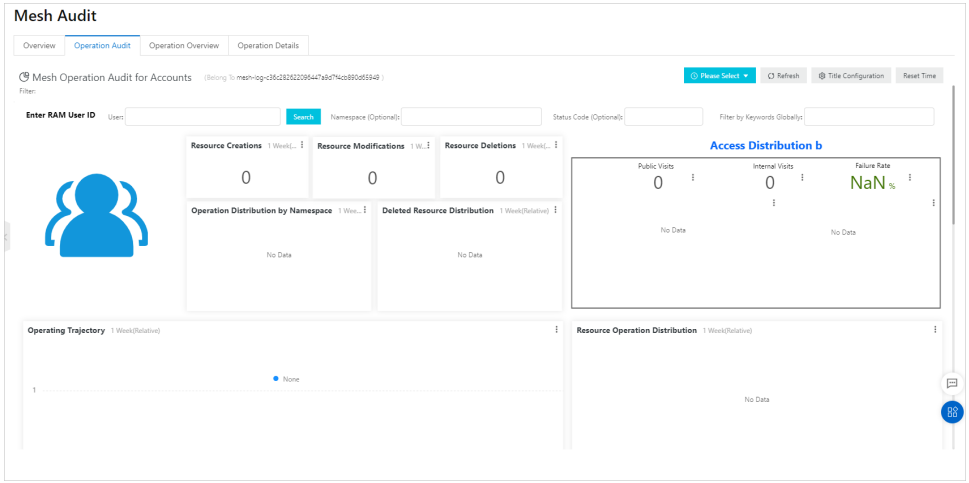
View audit reports

In the ASM console, you can view audit reports from different dimensions on the following tabs: Overview, Operation Audit, Operation Overview, and Operation Details.

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. In the left-side navigation pane, click **Mesh Audit**.
5. On the **Mesh Audit** page, select **Enable Mesh Audit** and click **OK**.
6. Click the **Overview** tab. This tab provides an overview of events in the ASM instance and detailed information about key events such as access from the Internet, command execution, and resource deletion.

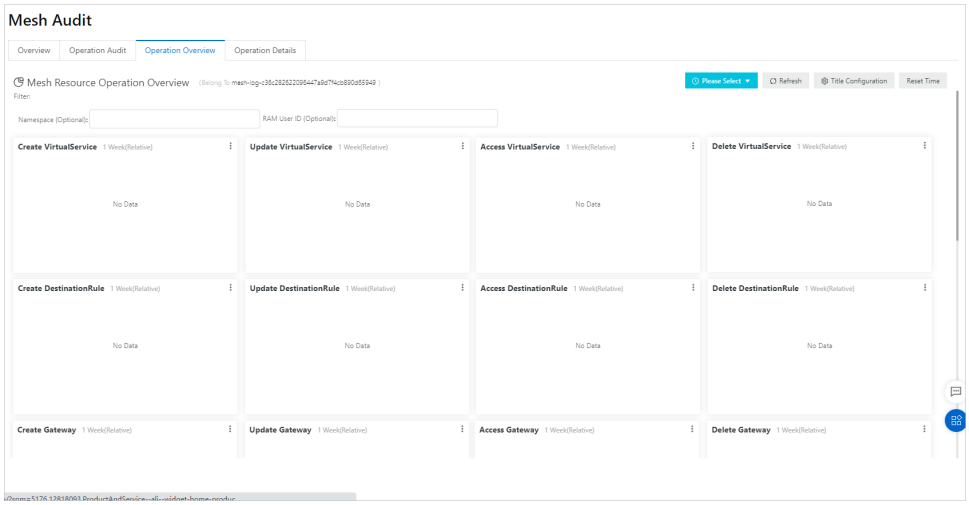


7. Click the **Operation Audit** tab. On this tab, you can view detailed information about the operations of a specified account on the ASM instance, such as resource creation, modification, and deletion. You can also view the distribution of operations by namespace and the distribution of access by geographical location.



8. Click the **Operation Overview** tab. On this tab, you can view the statistics of operations on main resources in the ASM instance.

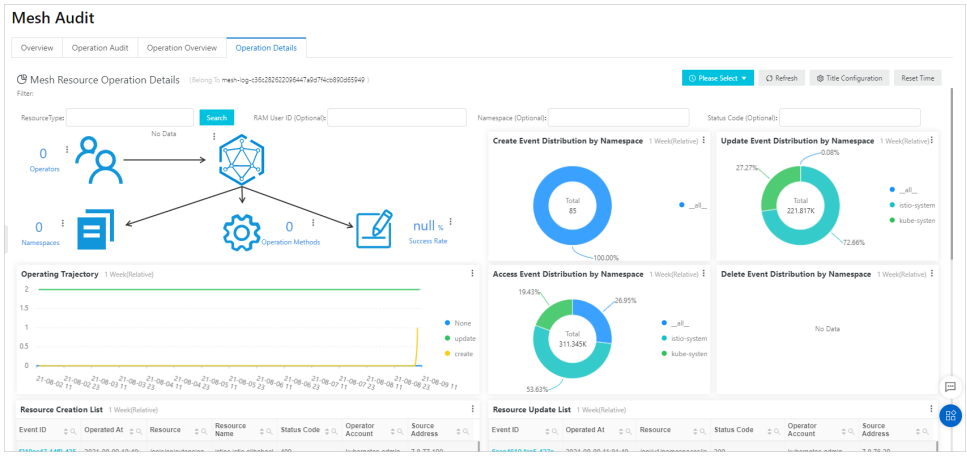
> **Note** You can use the following methods to filter the operation statistics:
>
> - Specify the time range. By default, the statistics in the last seven days are displayed.
> - Specify the namespace and RAM user ID.
> - Specify one or more filters.



9. Click the **Operation Details** tab. On this tab, you can view detailed information about the operations on specified resources in the ASM instance.

   You can specify the resource type to query detailed information about operations on such resources in real time. You can view the total number of operations, the distribution of operations by namespace, operation success rate, temporal order of operations, and detailed operation lists.

   > **Note** You can use the following methods to filter the operation statistics:
   >
   > - Specify the time range. By default, the statistics in the last seven days are displayed.
   > - Specify the namespace and RAM user ID.
   > - Specify one or more filters.



View audit logs

If you want to query and analyze audit logs, go to the Log Service console to view detailed logs.

1. Log on to the Log Service console.
2. On the **Projects** tab, click the project that is named mesh-log-${mesh-id}.
3. Click the audit-${mesh-id} Logstore that is created for the ASM instance. Then, click the **Search & Analysis** icon to view the audit logs.

> **Note**
>
> - When you enable the mesh audit feature, a Logstore that is named audit-${mesh-id} is created in the specified project.
> - By default, indexes are already set up in the Logstore. Do not modify the indexes. Otherwise, reports may fail to be generated from the audit logs.

You can use the following methods to query audit logs:

- To query the operations that are performed by a RAM user, enter the RAM user ID in the search box and click **Search & Analyze**.
- To query the operations that are performed on a resource, enter the resource name in the search box and click **Search & Analyze**.
- To query the operations that are performed by system components, enter `NOT user.username: node NOT user.username: serviceaccount NOT user.username: apiserver NOT user.username: kube-scheduler NOT user.username: kube-controller-manager` in the search box and click **Search & Analyze**.

For more information about the query and statistical methods, see Log search overview.

Set alerts

Log Service allows you to set alerts to monitor the operations that are performed on specific resources in real time. Alerts can be sent by using SMS messages, DingTalk chatbots, emails, custom webhooks, and Message Center of the Alibaba Cloud Management Console. For more information, see Overview.

You can also execute query statements in audit reports to query audit logs.

- Example 1: Set an alert on command execution on containers

  A company requires strict access control on its ASM instances and does not allow users to log on to or run commands on containers in the ASM instances. The company wants to be notified of an alert immediately if a user attempts to log on to or run commands on a container. The alert notification is required to include the following information: the container that was logged on to, executed commands, operator, event ID, operation time, and source IP address.

- The following code shows you a sample query statement:

```
verb : create and objectRef.subresource:exec and stage:  ResponseStarted | SELECT audit
ID as "Event ID", date_format(from_unixtime(__time__), '%Y-%m-%d %T' ) as "Operation ti
me",  regexp_extract("requestURI", '([^\?]*)/exec\?.*', 1)as "Resource",  regexp_extrac
t("requestURI", '\?(.*)', 1)as "Command" ,"responseStatus.code" as "Status code",
 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 "Username",
CASE WHEN json_array_length(sourceIPs) = 1 then json_format(json_array_get(sourceIPs, 0
)) ELSE  sourceIPs END
as "Source IP address" limit 100
```

- The following code shows you a sample conditional expression:

```
event =~ "*"
```

- Example 2: Set an alert on failed Internet access to the API server

  To prevent attacks on an ASM instance that allows Internet access, a company monitors the number of connections from a source IP address over the Internet and the connection failure rate. The company requires alerts to be sent immediately when the number of connections from the source IP address and the connection failure rate exceed the specified thresholds. For example, the company requires an alert to be sent when the number of connections from a source IP address reaches 10 and more than five of the connections failed. The alert notification is required to include the following information: the source IP address, the region to which the source IP address belongs, and whether the source IP address is risky.

  - The following code shows you a sample query statement:

```
* | select ip as "Source IP address", total as "Number of Internet connection requests"
, round(rate * 100, 2) as "Connection failure rate", failCount as "Number of unauthoriz
ed Internet connection requests", CASE when security_check_ip(ip) = 1 then 'yes' else '
no' end  as "Whether the IP address is risky",  ip_to_country(ip) as "Country", ip_to_p
rovince(ip) as "Province", ip_to_city(ip) as "City", ip_to_provider(ip) as "ISP" from (
select CASE WHEN json_array_length(sourceIPs) = 1 then json_format(json_array_get(sourc
eIPs, 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 "Number
of connections" > 10 and "Connection failure rate × 100" > 50 ORDER by "Number of conne
ctions" desc limit 100
```

  - The following code shows you a sample conditional expression:

```
source IP address =~ "*"
```

Recreate a deleted project

If you accidentally delete a project that is used for mesh audit from Log Service but still want to use the mesh audit feature, you must recreate the deleted project.

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. In the left-side navigation pane, click **Mesh Audit**.
5. On the **Mesh Audit** page, click **Recreate** in the **Rebuild Mesh Audit** message.

   The recreated project is named after the project name before deletion and suffixed with the timestamp when the project is recreated.

2.Enable external authorization in ASM

Services in a mesh can respond to call requests from each other only after the requests obtain the external authorization and pass the check of EnvoyFilter. This topic describes how to enable external authorization in Alibaba Cloud Service Mesh (ASM).

Prerequisites

- An ASM instance is created. The ACK cluster is added to the ASM instance. For more information, see Create an ASM instance and Add a cluster to an ASM instance.
- Connect to ACK clusters by using kubectl.
- Use kubectl to connect to an ASM instance.
- Applications are deployed in the ASM instance. For more information, see Deploy applications in an ASM instance.
- Istio resources are defined. For more information, see Define Istio resources.

Context

Services in a mesh may send call requests to each other. When a service sends a request to call another service, the gRPC framework that runs outside the mesh decides whether to authorize the request based on the specified rules. Then, EnvoyFilter calls an external authorization service to check whether the inbound request is authorized. If the request is unauthorized, the request is denied.

> **Note** We recommend that you put the EnvoyFilter resources in the first place in filter chains. This operation ensures that other filters only need to check the authorized requests.

For more information about external authorization of Envoy, visit External Authorization.

gRPC must define an operation to call the `Check()` method. The following code shows you the context for the request and response. For more information, visit 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);
}
```

Step 1: Configure an external authorization service

Based on the preceding defined gRPC operation, the following code provides an example on how to configure external authorization by calling the `Check()` method to check whether the HTTP header contains a bearer token that starts with `asm-`.

> **Note** The defined gRPC operation supports more complex rules for external authorization.

```
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.Che
ckResponse, 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
```

```
                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)
    }
}
```

You can directly use the image `registry.cn-beijing.aliyuncs.com/istio-samples/ext-authz-grpc-service:latest` to configure external authorization. Alternatively, you can build an image based on the Dockerfile in the following example: istio_ext_authz_filter_sample.

Step 2: Deploy an external authorization server

1. Download the YAML file for deploying an external authorization server from GitHub.
2. Connect the kubectl client to the ACK cluster that is added to the ASM instance and run the following command:

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

   If the following command output appears, an external authorization server is deployed:

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

Step 3: Deploy sample services

1. Download the YAML file for deploying the sample service httpbin from GitHub.
2. Connect the kubectl client to the ACK cluster that is added to the ASM instance and run the following command:

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

3. Deploy the sample client service sleep for testing.

   Download the YAML file for deploying the sample service sleep from GitHub.
4. Connect the kubectl client to the ACK cluster that is added to the ASM instance and run the following command:

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

Step 4: Define an EnvoyFilter resource

Run the following command to define an EnvoyFilter resource:

```
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"
      proxy:
        proxyVersion:  "^1\.(7|8|9).*"
    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 c
onnection pooling
            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
  - applyTo: HTTP_FILTER
    match:
      context: SIDECAR_INBOUND
      listener:
        name: virtualInbound
        filterChain:
          filter:
            name: "envoy.http_connection_manager"
      proxy:
        proxyVersion:  "^1\.(7|8|9).*"
```

```
        proxyVersion: "1\.(7|8|9).*"
    patch:
      operation: INSERT_BEFORE
      value:
        # Configure the envoy.ext_authz here:
        name: envoy.ext_authz
        typed_config:
          "@type": "type.googleapis.com/envoy.extensions.filters.http.ext_authz.v3.ExtAuthz
"
          grpc_service:
            # NOTE: *SHOULD* use envoy_grpc as ext_authz can use dynamic clusters and has c
onnection pooling
            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
```

Note The `proxyVersion` parameter must be set to the version of the ASM instance.

proxyVersion: Set the parameter to the version of the ASM instance.

If the following command output appears, an EnvoyFilter resource is defined:

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

Step 5: Verify external authorization

1. Log on to the Sleep pod and run the following command:

   ```
   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'
   ```

   The following command output appears:

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

   The command output indicates that the request from the sleep service failed to obtain the external authorization because the request header does not contain a bearer token that starts with `asm-`.

2. Run the following command to add a header that contains a bearer token that starts with `asm-` to the request:

   ```
   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-token
   1" http://httpbin:8000/headers'
   ```

   The following command output appears:

```
{
  "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"
  }
}
```


The command output indicates that the sleep service obtains the external authorization.

3.Use OPA to implement fine-grained access control in ASM

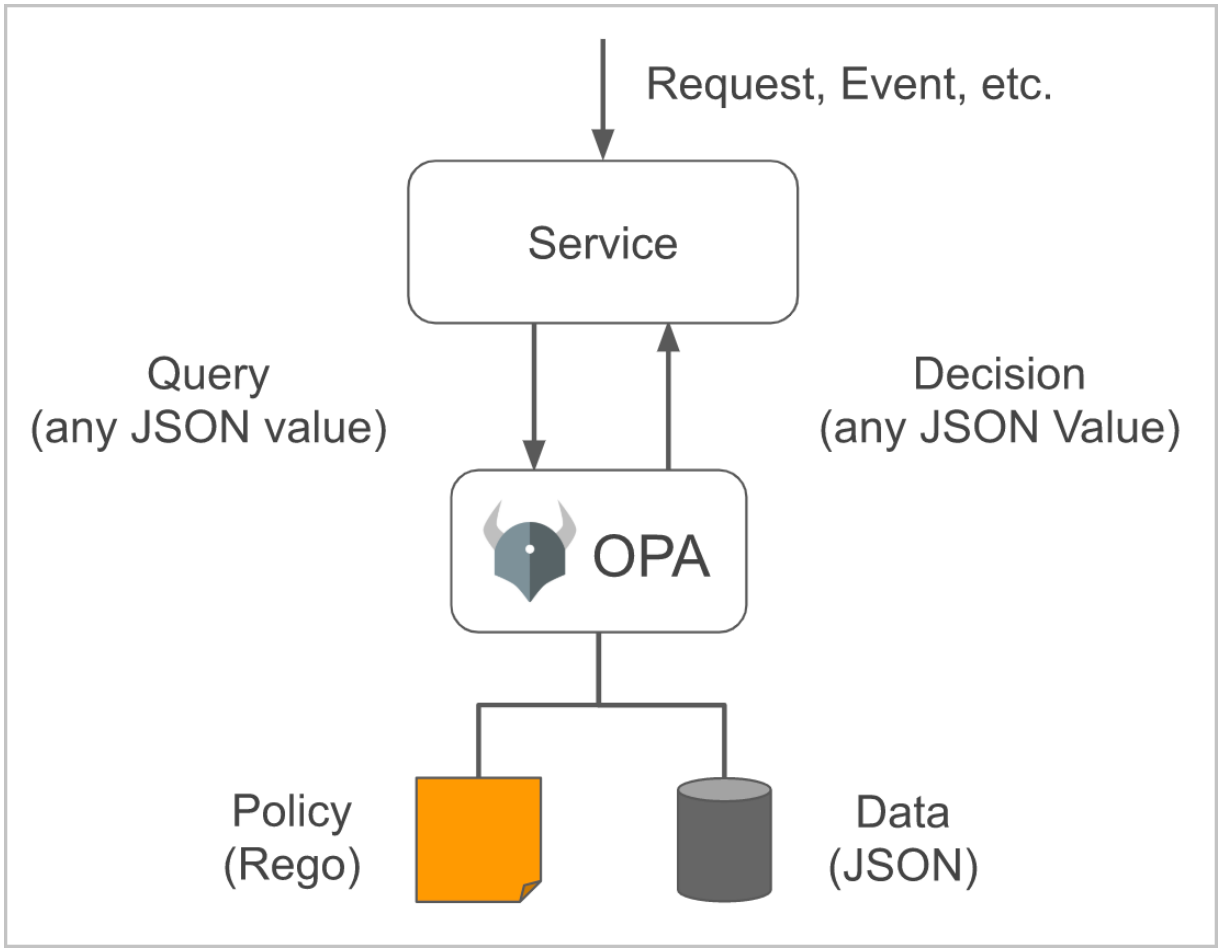
Alibaba Cloud Service Mesh (ASM) integrates with Open Policy Agent (OPA). You can use OPA to define access control policies to implement fine-grained access control on your applications. ASM allows you to define OPA policies on the control plane and push the policies to clusters on the data plane. This topic describes how to define OPA policies in ASM to implement fine-grained access control on your applications. For example, you can define OPA policies to allow or block requests based on request URLs or tokens in request headers. This topic also provides sample scenarios in which OPA policies are used to implement access control.

Prerequisites

- An ASM instance whose version is v1.9.7 or later is created. For more information, see Create an ASM instance.
- A Container Service for Kubernetes (ACK) cluster is created. For more information, see Create an ACK managed cluster.
- The ACK cluster is added to the ASM instance. For more information, see Add a cluster to an ASM instance.

Context

OPA is a Graduated project of Cloud Native Computing Foundation (CNCF). As a policy engine, OPA can be used to implement fine-grained access control on your applications. You can deploy OPA as a standalone service along with microservices. To protect an application, make sure that each request to a microservice of the application is authorized before the request is processed. To check the authorization, the microservice makes an API call to OPA to decide whether the request is authorized.



Procedure

Step 1: Enable OPA

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. In the **Basic Information** section, click **Settings** in the upper-right corner.
5. In the **Settings Update** panel, select **Enable OPA Plug-in**.
6. Click **OK**.
   In the **Basic Information** section, you can find that the status in the **OPA Plug-in** field changes to **Enabled**.

Step 2: Create an OPA policy

After you create an OPA policy on the control plane of the ASM instance, the OPA policy is pushed to clusters on the data plane. Then, OPA in the pods of the clusters uses the OPA policy to implement fine-grained access control.

Create an OPA policy in the ASM console

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.

3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4.
5. In the **Create** panel, select default from the **Namespace** drop-down list, specify a policy name, click **Add Matching Label**, add a label with the name of version and the value of v1, copy the following content to the code editor, and then click **OK**.

```
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"},
    ],
}
```

Create an OPA policy by using kubectl

1. Use kubectl to connect to the ASM instance. For more information, see Use kubectl to connect to an ASM instance.
2. Create an *opa.yaml* file and add the following content to the file:

```
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"},
        ],
    }
```

> **Note**
>
> - When you define OPA policies for a pod, make sure that only one OPA policy contains the `default allow` field. If multiple OPA policies apply to a pod and the `default allow` field is defined in each OPA policy, the OPA policies fail to be dynamically updated due to the multiple `default allow` fields.
> - We recommend that you use labels to specify the effective scope of OPA policies when you define the OPA policies. Invalid Rego policies make services inaccessible.
> - OPA resides in the same pod as business containers and occupies ports 15081 and 9191.
> - By default, the `default allow` field in an OPA policy is set to false. Do not set the `default allow` field again. Otherwise, a conflict occurs.

- spec: the policy content that is written in Rego. For more information about Rego syntax, see Policy Language.
- workloadSelector: the effective scope of the OPA policy in the specified namespace. By default, the OPA policy applies to all pods in the namespace. After you set this parameter, the OPA policy applies only to pods with specified labels.
- user_roles: assigns roles to users. In this example, the guest role is assigned to the guest1 user and the admin role to the admin1 user.
- role_perms: the permissions of each role. In this example, the guest role is granted the permissions to access an application by using a URL that contains */productpage*, and the admin role is granted the permissions to access an application by using a URL that contains */productpage* or */api/v1/products*.

3. Run the following command to create the OPA policy:

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

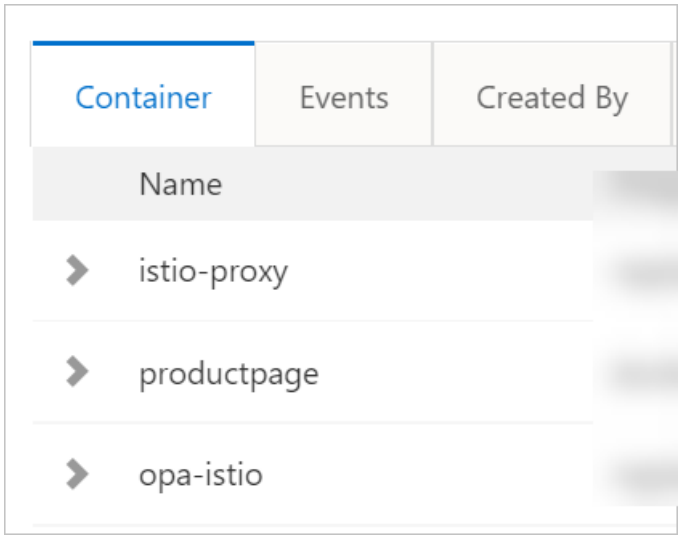
Step 3: Inject OPA

Deploy the sample application Bookinfo in the ASM instance and check whether OPA is injected into each pod of the Bookinfo application.

1. Deploy the Bookinfo application in the ASM instance. For more information, see Deploy an application in an ASM instance.
2. Define an Istio virtual service and an ingress gateway service for the application as required. For more information, see Define Istio resources.
3. Check whether OPA is injected into the pod of each application in the Bookinfo application.
   i. Log on to the ACK console.
   ii. In the left-side navigation pane of the ACK console, click **Clusters**.
   iii. On the **Clusters** page, find the cluster that you want to manage and click the name of the cluster or click **Details** in the **Actions** column. The details page of the cluster appears.
   iv. In the left-side navigation pane of the details page, choose **Workloads > Pods**.

v. At the top of the **Pods** page, select **default** from the **Namespace** drop-down list. Click the pod name of an application.

On the **Container** tab, you can find that a sidecar proxy that is named istio-proxy and OPA that is named opa-istio are injected into each container. Check the containers of each application in turn to ensure that the sidecar proxy and OPA are injected into each container.



Step 4: Verify that the OPA policy implements access control as expected

- Run the following cURL commands. The expected results indicate that the guest role is assigned to the guest1 user. Therefore, the guest1 user has the permissions to access the application by using a URL that contains */productpage*, but not */api/v1/products*.
  - Use the URL that contains */productpage* to access the application.

    ```
    curl -X GET http://<The IP address of the ingress gateway service>/productpage --user guest1:password -I
    ```

    Expected output:

    ```
    HTTP/1.1 200 OK
    ```

  - Use the URL that contains */api/v1/products* to access the application.

    ```
    curl -X GET http://<The IP address of the ingress gateway service>/api/v1/products --user guest1:password -I
    ```

    Expected output:

    ```
    HTTP/1.1 403 Forbidden
    ```

- Run the following cURL commands. The expected results indicate that the admin role is assigned to the admin1 user. Therefore, the admin1 user has the permissions to access the application by using a URL that contains /productpage or /api/v1/products.

- Use the URL that contains */productpage* to access the application.

  ```
  curl -X GET http://<The IP address of the ingress gateway service>/productpage --user admin1:password -I
  ```

  Expected output:

  ```
  HTTP/1.1 200 OK
  ```

- Use the URL that contains */api/v1/products* to access the application.

  ```
  curl -X GET http://<The IP address of the ingress gateway service>/api/v1/products --user admin1:password -I
  ```

  Expected output:

  ```
  HTTP/1.1 200 OK
  ```

The preceding results indicate that the defined OPA policy implements access control as expected.

Step 5: Dynamically update the OPA policy

Run the following command to go to the editing screen of the OPA policy:

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

In the command output, edit the OPA policy to assign both the guest and admin roles to the guest1 user.

```
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: assigns roles to users. In this example, the guest and admin roles are assigned to the guest1 user and the admin role to the admin1 user.
- role_perms: the permissions of each role. In this example, the guest role is granted the permissions to access an application by using a URL that contains */productpage*, and the admin role is granted the permissions to access an application by using a URL that contains */productpage* or */api/v1/products*.

Step 6: Verify that the OPA policy is dynamically updated

Run the following cURL commands. The expected results indicate that the guest and admin roles are assigned to the guest1 user. Therefore, the guest1 user has the permissions to access the application by using a URL that contains /productpage or /api/v1/products.

- Use the URL that contains */productpage* to access the application.

```
curl -X GET http://<The IP address of the ingress gateway service>/productpage --user guest1:password -I
```

Expected output:

```
HTTP/1.1 200 OK
```

- Use the URL that contains */api/v1/products* to access the application.

```
curl -X GET http://<The IP address of the ingress gateway service>/api/v1/products --user guest1:password -I
```

Expected output:

```
HTTP/1.1 200 OK
```

Before the OPA policy is updated, the guest1 user can access the application by using a URL that contains /productpage, but not /api/v1/products. After the OPA policy is updated, the guest1 user can access the application by using a URL that contains /productpage or /api/v1/products. The results indicate that the OPA policy is dynamically updated.

Sample scenarios

Scenario 1: Authenticate a request by checking the JWT

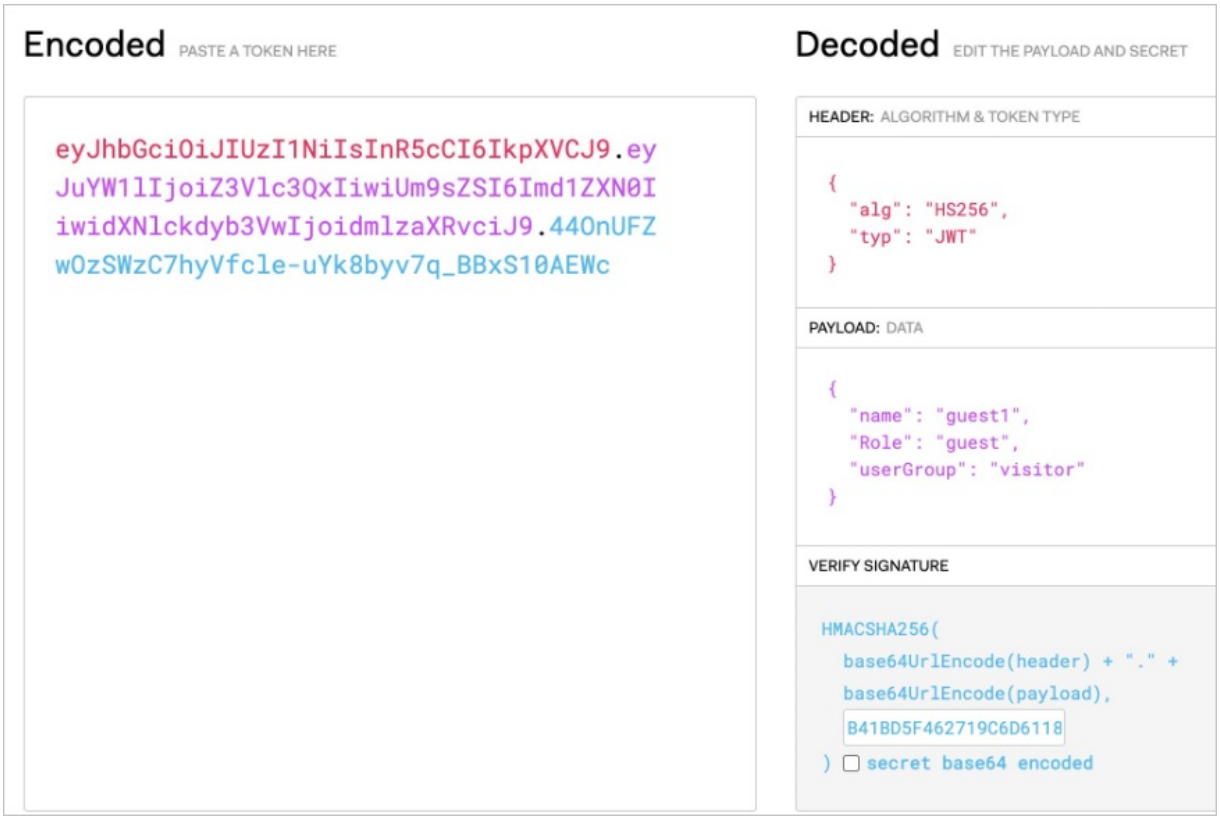
When an application receives a request, OPA checks the JSON Web Token (JWT) in the headers of the request. The request to access the application is allowed only if the JWT meets specified requirements.

The following OPA policy defines that a request to access the Productpage application is allowed only if the request uses the GET method and the request contains a JWT in which the `Role` field is set to `guest` and the `userGroup` field is set to `visitor`:

```
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: the request method. In this example, the value is set to `GET`.
- input.parsed_path[0]: the application that is to be accessed.
- claims.Role: the expected value of the `Role` field in the JWT. In this example, the value is set to `guest`.
- claims.userGroup: the expected value of the `userGroup` field in the JWT. In this example, the value is set to `visitor`.

You can use a JWT tool to encode request information such as the `Role` and `userGroup` fields into a JWT string.



Run the following command to access the Productpage application:

```
curl --location --request GET 'http://{The IP address of the ingress gateway service}/productpage' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiZ3Vlc3QxIiwiUm9sZSI6Imd1ZXN0IiwidXNlckdyb3VwIjoidmlzaXRvciJ9.44OnUFZwOzSWzC7hyVfcle-uYk8byv7q_BBxS10AEWc'
```

Expected output:

```
200
```

The status code 200 is returned, which indicates that the Productpage application can be accessed by using the GET request that contains a JWT in which the `Role` field is set to `guest` and the `userGroup` field is set to `visitor`. If you use an invalid JWT or the request contains no JWT, the error code 403 is returned. This indicates that the request to access the Productpage application is rejected.

Scenario 2: Authenticate a request by checking the HTTP request body and JWT

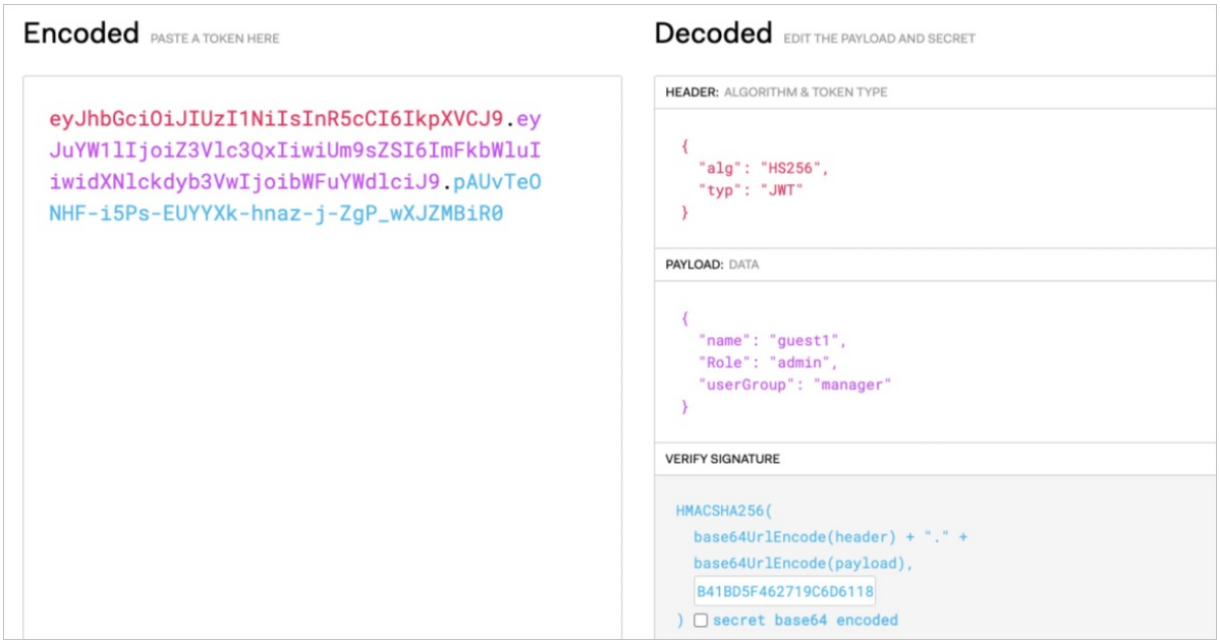
You can use an OPA policy to allow an HTTP request only if the value of the username field in the request body is the same as the value of the Role field in the JWT of the request.

The following OPA policy defines that a request to access the Productpage application is allowed only if the request uses the GET method, the value of the `username` field in the request body is the same as the value of the `Role` field in the JWT of the request, and the `userGroup` field in the JWT is set to `manager`:

```
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: the request method. In this example, the value is set to `GET`.
- input.parsed_path[0]: the application that is to be accessed.
- claims.Role: the expected value of the Role field in the JWT. In this example, the value is set to `input.parsed_body.username`. This indicates that the value of the `username` field in the request body must be the same as the value of the Role field in the JWT of the request.
- claims.userGroup: the expected value of the `userGroup` field in the JWT. In this example, the value is set to `manager`.

You can use a JWT tool to encode request information such as the `Role` and `userGroup` fields into a JWT string.



Run the following command to access the Productpage application:

```
curl --location --request GET 'http://{The IP address of the ingress gateway service}/produ
ctpage' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiZ3Vlc3QxIi
wiUm9sZSI6ImFkbWluIiwidXNlckdyb3VwIjoibWFuYWdlciJ9.pAUvTeONHF-i5Ps-EUYYXk-hnaz-j-ZgP_wXJZMB
iR0' \
--header 'Content-Type: application/json' \
--header 'Cookie: session=eyJ1c2VyIjoiYWRtaW4ifQ.YRz90g.GT34_5BqlFTwGqabZk_qGZzxYQ0' \
--data-raw '{
    "username":"admin",
    "password":"12****"
```

Expected output:

```
200
```

The status code 200 is returned, which indicates that the Productpage application can be accessed by using the GET request that contains a JWT in which the value of the `Role` field is the same as the value of the `username` field in the request body and the `userGroup` field is set to `manager`. If you use an invalid JWT or the request contains no JWT, the error code 403 is returned. This indicates that the request to access the productpage application is rejected.

Scenario 3: Authenticate a request by checking more context information

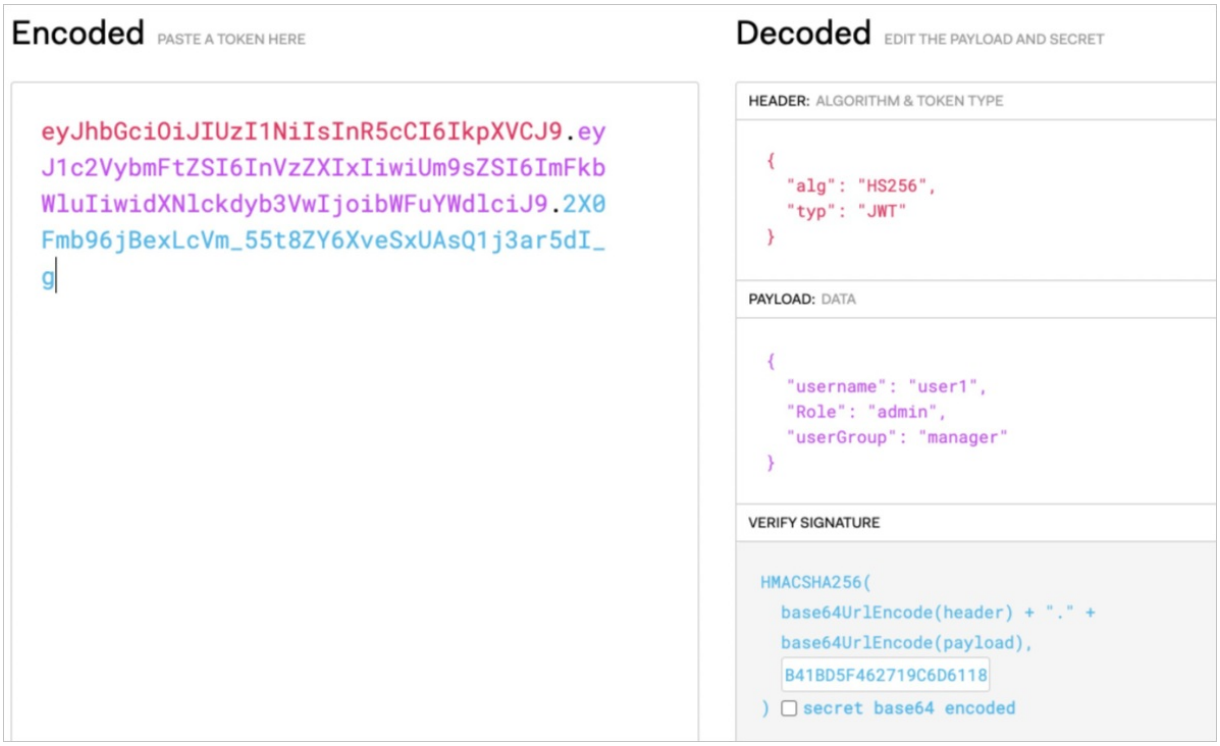
You can use an OPA policy to check more context information in addition to the information that is checked in Scenario 2. In this example, the value of the `username` field in the JWT must falls into the value range specified by the `bookinfo_managers` field.

The following OPA policy defines that a request to access the Productpage application is allowed only if the request uses the GET method, the value of the `username` field in the request body is the same as the value of the `Role` field in the JWT of the request, the value of the `username` field in the JWT falls into the value range specified by the `bookinfo_managers` field, and the `userGroup` field in the JWT is set to `manager`:

```
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: the request method. In this example, the value is set to `GET`.
- input.parsed_path[0]: the application that is to be accessed.
- claims.Role: the expected value of the Role field in the JWT. In this example, the value is set to `input.parsed_body.username`. This indicates that the value of the username field in the request body must be the same as the value of the Role field in the JWT of the request.
- claims.userGroup: the expected value of the `userGroup` field in the JWT. In this example, the value is set to `manager`.
- claims.username: the expected value of the username field in the JWT. In this example, the value is set to `bookinfo_managers[_].name`. This indicates that the value of the `username` field in the JWT must fall into the value range specified by the `bookinfo_managers` field.

You can use a JWT tool to encode request information into a JWT string.



Run the following command to access the Productpage application:

```
curl --location --request GET 'http://{The IP address of the ingress gateway service}/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****"
}'
```

Expected output:

```
200
```

The status code 200 is returned, which indicates that the Productpage application can be accessed by using the GET request that contains a JWT in which the value of the `Role` field is the same as the value of the `username` field in the request body, the value of the `username` field in the JWT falls into the value range specified by the `bookinfo_managers` field, and the `userGroup` field is set to `manager`. If you use an invalid JWT or the request contains no JWT, the error code 403 is returned. This indicates that the request to access the Productpage application is rejected.

FAQ

How do I check whether a pod uses OPA policies?

OPA is deployed in sidecar mode in the same pod as business containers. To check whether a pod uses OPA policies, connect to the pod and run the following command:

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

How do I check the policies that are written in Rego?

The official OPA website provides an online tool that you can use to check the policies that are written in Rego.

References

If you want to use ConfigMaps to define OPA policies, read the following topics:

- Use OPA to implement fine-grained access control in ASM
- Dynamically update OPA policies in ASM

4.Configure JWT authorization in ASM

In Alibaba Cloud Service Mesh (ASM), you can configure JSON Web Token (JWT) authorization to authenticate the source of requests. This method is also called end-user authentication. When an application on an ASM instance configured with a JWT authorization policy receives a request, the system checks whether the request header contains a valid JWT. Only requests with valid JWTs are allowed. This topic describes how to configure a JWT authorization policy for an ASM instance.

Prerequisites

- Make sure that the Istio version is 1.6 or later. Otherwise, you cannot use the RequestAuthentication feature.
- An ASM instance is created and a Container Service for Kubernetes (ACK) cluster is added to the instance. For more information, see Add a cluster to an ASM instance.

Context

ASM supports the following authentication methods:

- Transmission authentication: This method is based on Mutual Transport Layer Security (mTLS) and used to authenticate communications among services.
- Source authentication: This method is based on JWT and used to authenticate requests from the client to the server.

JWT is a standard that uses representative claims to secure information transmission between parties. It is used to authenticate the source of requests. For more information about JWT, see JWT official documentation.

Step 1: Deploy a sample service

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. On the details page of the ASM instance, click **Namespace** in the left-side navigation pane. On the Namespace page, click **Create**.
5. In the **Create Namespace** panel, set the **Name** and **Tag** parameters. In this example, set the Name parameter to `foo` and the Tag parameter to `istio-injection:enabled`. The settings are used to create a namespace named `foo` with `automatic sidecar injection` enabled.
6. Click **OK** to create the `foo` namespace.
7. Run the following commands to deploy the official sample services `httpbin` and `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. Run the following commands to keep the system waiting until pods of the httpbin and sleep services get ready:

```
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
```

Execution result:

In the pod of the `sleep` service, edit the YAML file of the ACK cluster to check whether requests can be sent to `httpbin`. If `HTTP code 200` is returned, requests can be sent to httpbin.

```
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
```

Step 2: Create a request authentication policy

1. On the details page of the ASM instance, choose **Zero Trust Security** > **RequestAuthentication** in the left-side navigation pane. On the RequestAuthentication page, click **Create RequestAuthentication**.
2. In the **Create** panel, select **foo** from the **Namespaces** drop-down list. In the code editor, edit a YAML file in the `foo` namespace to create a request authentication policy.

   The following code provides an example of the content for the jwt-example.yaml file:

```
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"}]}'
```

> **Note**
>
> The `jwtRules` parameter in the preceding YAML code specifies a rule for requests of the `httpbin` service. If the request header contains an access token, the decoded value of the `iss` parameter must be `testing@secure.istio.io`. The `jwks` parameter specifies how to generate the access token. For more information, see JSON Web Key (JWK).

3. Click **OK** to create the request authentication policy.
   Execution result:

   If the request header contains a valid access token, status code 200 is returned. Otherwise, status code 401 is returned.

   - Use the following code to determine when to return status code 200:

     ```
     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
     ```

   - Use the following code to determine when to return status code 401:

```
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
```

Step 3: Create a JWT authorization policy

1. On the details page of the ASM instance, choose **Zero Trust Security > AuthorizationPolicy** in the left-side navigation pane. On the AuthorizationPolicy page, click **Create AuthorizationPolicy**.
2. In the **Create** panel, select **foo** from the **Namespaces** drop-down list. In the code editor, edit a YAML file in the `foo` namespace to create an authorization policy.

   The following code provides an example of the content for the require-jwt.yaml file:

```
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"]
```

> **Note**
>
> The preceding authorization policy implements the following logic: When a request is sent to `httpbin`, the system checks the decoded access token in the request header. The request is allowed only when the values of the `iss` and `sub` parameters in the decoded access token are the same as those indicated by the `source.requestPrincipals` parameter. The source.requestPrincipals parameter is in the format of `iss/sub`. In this example, the value of the `source.requestPrincipals` parameter is `testing@secure.istio.io/testing@secure.istio.io`.

- The following code provides an example of the access token:

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

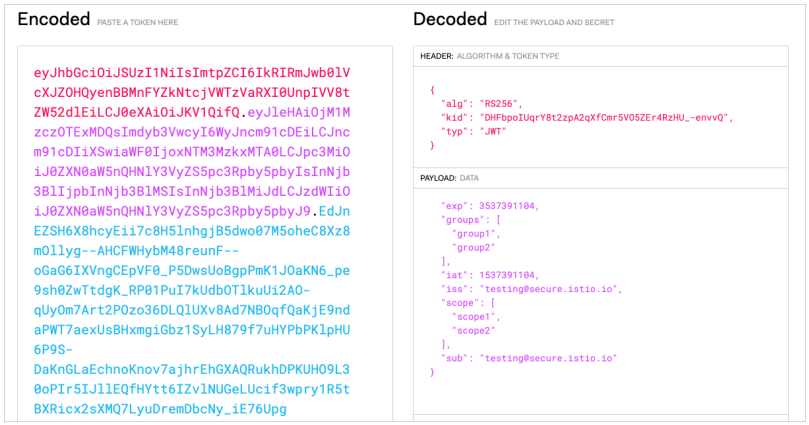
- Run the following command to decode the access token:

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

- The following response shows the decoded value of the sample access token:

```
null
```

The JWT official website also provides a GUI for you to decode access tokens, as shown in the following figure.



3. Click **OK** to create the JWT authorization policy.
   Execution result:

   If the request header contains a valid access token, status code 200 is returned. Otherwise, status code 403 is returned.

   - Use the following code to determine when to return status code 200:

```
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
```

- Use the following code to determine when to return status code 403:

```
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
```

Step 4: Update the JWT authorization policy

1. On the details page of the ASM instance, choose **Zero Trust Security > AuthorizationPolicy** in the left-side navigation pane.
2. On the **AuthorizationPolicy** page, find the authorization policy that you want to modify and click **YAML** in the **Actions** column.
3. In the **Edit** dialog box, add specified content in the code editor.

   The following code provides an example of the content to be added to the require-jwt-group.yaml file:

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

   The following code provides an example of the complete content for the require-jwt-group.yaml file:

```
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"]
```

> **Note**
>
> When a request is sent to `httpbin`, the system checks the decoded access token in the request header. Then, the system applies the updated authorization policy to the request. The request is allowed only when the decoded access token meets the following conditions: The value of the iss parameter is testing@secure.istio.io, the value of the sub parameter is testing@secure.istio.io, and the `groups` parameter contains `group1`.

- The following code provides an example of the access token:

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

- Run the following command to decode the access token:

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

- The following response shows the decoded value of the sample access token:

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

4. Click **OK** to update the JWT authorization policy.
   Execution result:

   If the request header contains a valid access token, status code 200 is returned. Otherwise, status code 403 is returned.

   - Use the following code to determine when to return status code 200:

```
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
```

   - Use the following code to determine when to return status code 403:

```
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.Use authorization policies to enable access control

Authorization policies enable access control on workloads in Alibaba Cloud Service Mesh (ASM) instances. This topic shows you how to use authorization policies to enable access control.

Prerequisites

- An ASM instance is created. For more information, see Create an ASM instance.
- A Container Service for Kubernetes (ACK) cluster is created. For more information, see Create an ACK managed cluster.
- The ACK cluster is added to the ASM instance. For more information, see Add a cluster to an ASM instance.
- An ingress gateway service is deployed. For more information, see Deploy an ingress gateway service.

Step 1: Deploy a sample application in the ACK cluster

1. Enable automatic sidecar injection for the default namespace. For more information, see Enable automatic sidecar injection.
2. Use kubectl to connect to the ACK cluster. For more information, see Connect to ACK clusters by using kubectl.
3. Use the following content to create a YAML file that is named *httpbin*:

```
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. Run the following command to create the httpbin application:

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

5. Run the following command to check whether the pod of the httpbin application runs as expected:

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

The following output is expected:

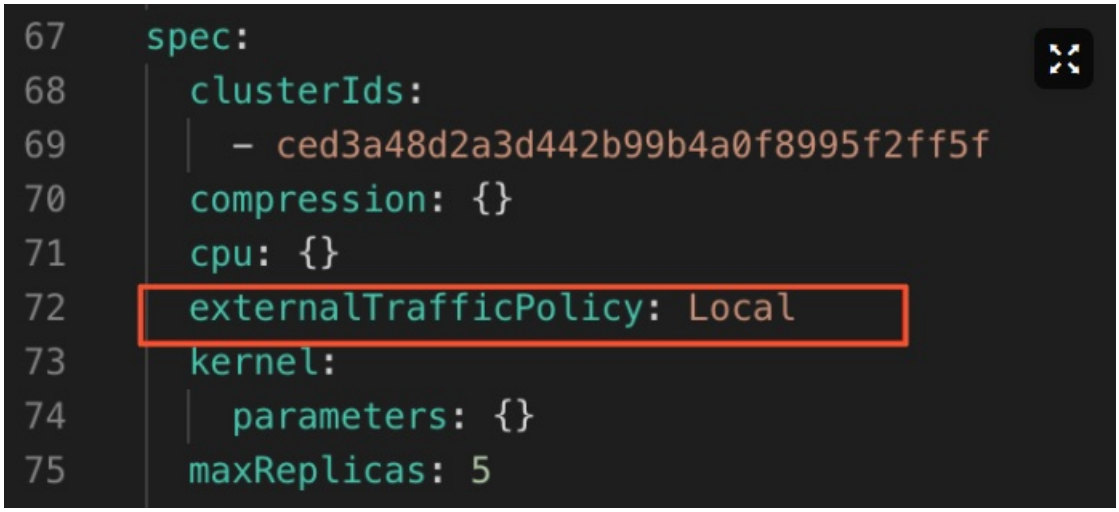
```
httpbin-66cdbdb6c5-vhsh6          2/2     Running   0          11s
```

Step 2: Preserve the source IP address of the client that sends requests

Note You can use the following methods to preserve the source IP address of the client that sends requests:

- Set the `externalTrafficPolicy` parameter to Local to preserve the source IP address and use the `ipBlocks` parameter to create an authorization policy. We recommend that you use this method.
- Use the X-Forwarded-For HTTP header or the proxy protocol to preserve the source IP address and use the `remoteIpBlocks` parameter to create an authorization policy.

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. On the details page of the ASM instance, click **ASM Gateways** in the left-side navigation pane.
5. Find the ingress gateway that is named ingressgateway and click **YAML** in the **Actions** column.
6. In the **Edit** panel, enter `externalTrafficPolicy: Local` and click **OK**.



7. Run the following command to enable role-based access control (RBAC) debugging for the ingress gateway:

```
kubectl exec -it -n istio-system <istio-ingressgateway pod name> -- curl-x post localhost:15000/logging?rbac=debug
```

8. Run the following command in the ACK cluster to query the IP address of the client:

```
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"
```

Step 3: Configure a routing rule in the ASM instance

Configure an ingress gateway and a virtual service to allow all requests to access the httpbin application.

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. Create an ingress gateway.
   i. On the details page of the ASM instance, choose **Traffic Management > Gateway** in the left-side navigation pane. On the Gateway rules page, click **Create from YAML**.
   ii. In the **Create** panel, select default from the **Namespace** drop-down list and copy the following content to the code editor. Then, click **OK**.

```
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. Create a virtual service.
   i. On the details page of the ASM instance, choose **Traffic Management > VirtualService** in the left-side navigation pane. On the Virtual service page, click **Create from YAML**.
   ii. In the **Create** panel, select default from the **Namespace** drop-down list and copy the following content to the code editor. Then, click **OK**.

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

6. Run the following command to check whether the httpbin application is accessible:

```
curl "<IP address of the ingress gateway>:<Port number of the ingress gateway>"/headers
-s -o /dev/null -w "%{http_code}\n"
```

> **Note** For more information about how to query the IP address of the ingress gateway, see Define Istio resources.

The following output is expected:

```
200
```

A return value of 200 indicates that the httpbin application is accessible.

Step 4: Configure an authorization policy

Configure an authorization policy to allow specified IP addresses to access the httpbin application

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. On the details page of the ASM instance, choose **Zero Trust Security > AuthorizationPolicy** in the left-side navigation pane. On the AuthorizationPolicy page, click **Create AuthorizationPolicy**.
5. In the **Create** panel, select istio-system from the **Namespace** drop-down list and copy the following content to the code editor. Then, click **OK**.

```
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"]
```

Set the `action` parameter to `ALLOW` and the `ipBlocks` parameter to `["1.2.3.4", "5.6.7.0/24"]`. The settings specify that only the 1.2.3.4 IP address and IP addresses in the 5.6.7.0/24 block can access the httpbin application.

6. Run the following command to check whether the IP address of the client can access the httpbin application:

```
curl "<IP address of the ingress gateway>:<Port number of the ingress gateway>"/headers
-s -o /dev/null -w "%{http_code}\n"
```

The following output is expected:

```
403
```

A return value of 403 indicates that the client fails to access the httpbin application. This means that IP addresses other than 1.2.3.4 and 5.6.7.0/24 cannot access the httpbin application.

Configure an authorization policy to allow the client to access the httpbin application

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. On the details page of the ASM instance, choose **Zero Trust Security > AuthorizationPolicy** in the left-side navigation pane. On the AuthorizationPolicy page, click **Create AuthorizationPolicy**.
5. In the **Create** panel, select istio-system from the **Namespace** drop-down list and copy the following content to the code editor. Then, click **OK**.

```
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"]
```

Set the `action` parameter to `ALLOW` and the `ipBlocks` parameter to `["1.2.3.4", "5.6.7.0/24", "$CLIENT_IP"]`. The settings specify that only 1.2.3.4, 5.6.7.0/24, and the IP address of the client can access the httpbin application.

6. Run the following command to check whether the IP address of the client can access the httpbin application:

```
curl "<IP address of the ingress gateway>:<Port number of the ingress gateway>"/headers
-s -o /dev/null -w "%{http_code}\n"
```

The following output is expected:

```
200
```

A return value of 200 indicates that the client can access the httpbin application.

Configure an authorization policy to deny access to the httpbin application from specified IP addresses

1. Log on to the ASM console.
2. In the left-side navigation pane, choose **Service Mesh > Mesh Management**.
3. On the **Mesh Management** page, find the ASM instance that you want to configure. Click the name of the ASM instance or click **Manage** in the **Actions** column of the ASM instance.
4. On the details page of the ASM instance, choose **Zero Trust Security > AuthorizationPolicy** in the left-side navigation pane. On the AuthorizationPolicy page, click **Create AuthorizationPolicy**.
5. In the **Create** panel, select istio-system from the **Namespace** drop-down list and copy the following content to the code editor. Then, click **OK**.

   ```
   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"]
   ```

   Set the `action` parameter to `DENY` and the `ipBlocks` parameter to `["1.2.3.4", "5.6.7.0/24", "$CLIENT_IP"]`. The settings specify that 1.2.3.4, 5.6.7.0/24, and the IP address of the client cannot access the httpbin application.
6. Run the following command to check whether the IP address of the client can access the httpbin application:

   ```
   curl "<IP address of the ingress gateway>:<Port number of the ingress gateway>"/headers -s -o /dev/null -w "%{http_code}\n"
   ```

   The following output is expected:

   ```
   403
   ```

   A return value of 403 indicates that the client fails to access the httpbin application.