

# Alibaba Cloud

## Alibaba Cloud Service Mesh Traffic Management

Document Version: 20220627

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

# Table of Contents

1. Configure traffic routing for an ASM gateway .....	05
2. Use ASM to transfer TCP traffic .....	21
3. Use ASM to achieve gRPC load balancing .....	29
4. Use EnvoyFilter to add HTTP response headers in ASM .....	37
5. Use ASM to route traffic based on the locality .....	42
6. Use an ingress gateway to access a gRPC service in an ASM i... ..	55
7. Use an ingress gateway to access a WebSocket service in an A... ..	62
8. Access services over WebSocket connections in ASM .....	66
9. Use delegates to configure virtual services in an ASM instance .....	74
10. Use the local throttling feature of ASM .....	79
11. Enable graceful shutdown for the SLB instance of an ASM ga... ..	92
12. Traffic labeling and label-based routing .....	97
13. Use an ASMAdaptiveConcurrency to implement adaptive conc... ..	107
14. Manage Spring Cloud services .....	130

# 1. Configure traffic routing for an ASM gateway

You can create destination rules and virtual services for an Alibaba Cloud Service Mesh (ASM) gateway in a visualized manner without writing YAML files. This simplifies traffic management. This topic describes how to create traffic policies and routing rules for an ASM gateway in a visualized manner.

## Prerequisites

- 
- 
- 
- 
- Sidecar injection is enabled for the specified namespace. For more information, see [Enable automatic sidecar injection by using multiple methods](#).
- The IP address of the specified ASM gateway is obtained. For more information, see the "Step 3: Access an application by using the ingress gateway service" section of the [Define Istio resources](#) topic.

## Context


You can create a traffic policy that specifies the load balancing and connection pool settings for a service in a visualized manner. In this example, the reviews service is used. After the traffic policy is created, a YAML file for a destination rule is automatically generated. You can create routing rules for a service to match specific content in a URL, such as */productpage* and */login*. In this example, the Bookinfo service is used. This way, you can access the Bookinfo service by using a URL that contains information such as */productpage*.

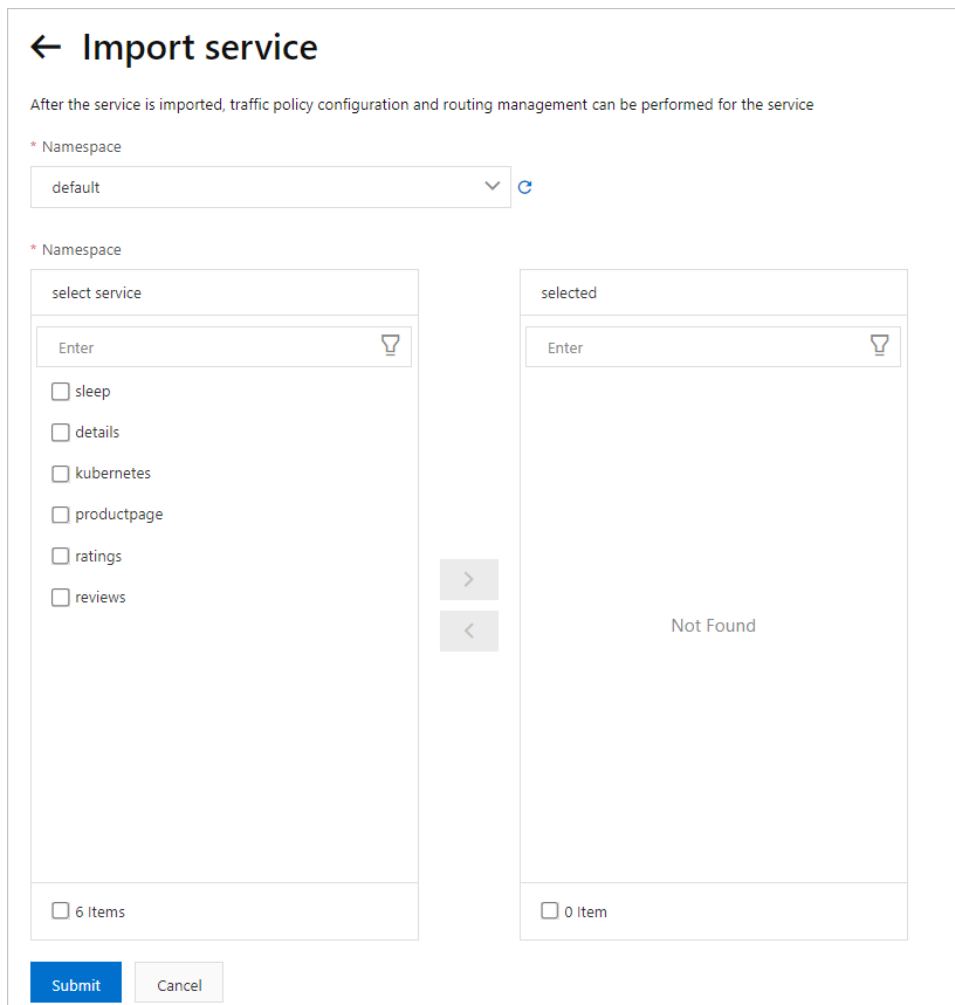
## Create a traffic policy

1. Import an upstream service.

Import the reviews service to the ASM gateway to associate the ASM gateway with the reviews service.

- i.
- ii.
- iii.
- iv. On the details page of the ASM instance, click **ASM Gateways** in the left-side navigation pane. On the page that appears, click the name of the ASM gateway to which you want to import the reviews service.
- v. On the details page of the ASM gateway, click **Upstream Service** in the left-side navigation pane.
- vi. On the page that appears, click **Import service**.

- vii. On the **Import service** page, select a namespace from the **Namespace** drop-down list. In the select service section, select the reviews service and click the  icon to move the reviews service to the selected section. Then, click **OK**.



## 2. Manage versions of the reviews service.

Group instances of the reviews service by version. In this example, the versions of the reviews service are v1, v2, and v3.

- i. On the **Upstream Service** page, find the reviews service and click **versions** in the Actions column.
- ii. On the **versions** page, click **Add Subset**. Set the **Name** parameter to *v1*. Then, click **Add Label** and set the **Name** parameter to *version* and the **Value** parameter to *v1*.
- iii. Click **Add Subset** for a second time. Set the **Name** parameter to *v2*. Then, click **Add Label** and set the **Name** parameter to *version* and the **Value** parameter to *v2*.

- iv. Click **Add Subset** for a third time. Set the **Name** parameter to `v3`. Then, click **Add Label** and set the **Name** parameter to `version` and the **Value** parameter to `v3`. Then, click **Submit**.

The screenshot shows the 'versions' configuration page. It has a blue header with a back arrow and the title 'versions'. Below the header is a blue bar with the text 'versions'. Underneath is a section titled 'Add Subset'. There are three subset configurations visible:

- Subset v1:** Name is 'v1'. It has one label with Name 'version' and Value 'v1'.
- Subset v2:** Name is 'v2'. It has one label with Name 'version' and Value 'v2'.
- Subset v3:** Name is 'v3'. It has one label with Name 'version' and Value 'v3'.

Each subset configuration includes an 'Add Label' button and a note: 'Each label is used to filter endpoints for the subset'. At the bottom of the page are 'Create' and 'Preview' buttons.

3. Create a traffic policy.
- On the **Upstream Service** page, find the reviews service and click **Traffic Policy** in the **Actions** column.
  - On the **Traffic Policy** page, click **Add Policy**. Select **Subset Level** and select `v3` from the Subset drop-down list. Turn on **Load Balancing** and select **Simple**. Then, set the **Mode** parameter to **Random** and click **Submit**.

← Traffic Policy

Traffic Policy

⊕ Add Policy

☐ Host Level
☐ Host Port Level
☒ Subset Level
☐ Subset Port Level

Host level policy applies to the host, subset level policy applies to the subset, and port level policy applies to the port

\* Subset

Select ▼

☒ Load Balancing

☒ Simple
☐ Consistent Hash

\* Mode

Random ▼

☐ Locality Load Balancing

☐ Connection Pool

☐ Outlier Ejection

☐ Client TLS

Submit

Preview

- (Optional) After the traffic policy is created, view the YAML file that is automatically generated for a destination rule.

On the **Upstream Service** page, find the reviews service and click **YAML** in the **Actions** column to view the YAML file in the **Preview** panel.

**Note** You can also choose **Traffic Management > DestinationRule** in the left-side navigation pane of the details page of the ASM instance. On the **DestinationRule** page, find the destination rule whose YAML file you want to view and click **YAML** in the **Actions** column. In the **Edit** panel, view the YAML file of the destination rule.



```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: reviews
  namespace: default
  labels:
    provider: asm
spec:
  host: reviews
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
    - name: v3
      labels:
        version: v3
  trafficPolicy:
    loadBalancer:
      simple: RANDOM
```

## Create routing rules

### 1. Import an upstream service.


Import the productpage service to the ASM gateway to associate the ASM gateway with the productpage service.

- i.
- ii.
- iii.

iv. On the details page of the ASM instance, click **ASM Gateways** in the left-side navigation pane. On the page that appears, click the name of the ASM gateway to which you want to import the productpage service.

v. On the details page of the ASM gateway, click **Upstream Service** in the left-side navigation pane.

vi. On the **Upstream Service** page, click **Import service**.

vii. On the **Import Service** page, select a namespace from the **Namespace** drop-down list. In the select service section, select the productpage service and click the  icon to move the productpage service to the selected section. Then, click **OK**.

### 2. Create routing rules.

i. Create a routing rule to match a URL that contains */productpage*.

a. On the details page of the ASM gateway, click **Route management** in the left-side navigation pane. On the page that appears, click **Create**.

b. In the **set route detail** step, set the parameters that are described in the following table, and then click **Next**.

← Create

1

2

3

4

set route detail

set route destination

advanced config

confirm route configuration

\* route type

http

Namespace

default

Name

productpage-route

description

productpage route

priority

3

☒ Matching URI

\* Method

Exact

\* Content

/productpage

⊕ Add Header Matching Rule

Next

Parameter	Description
route type	The type of the traffic to be routed. In this example, the value is http.
Namespace	The namespace in which you create the routing rule. In this example, the value is default.
Name	The name of the routing rule. In this example, the value is productpage-route.
description	The description of the routing rule. In this example, the value is The routing rule used to match a URL that contains /productpage.
priority	The priority of the routing rule. A smaller value indicates a higher priority. If a URL matches multiple routing rules, the routing rule that has the highest priority takes precedence. In this example, the value is 3.

Parameter	Description
Matching URI	<p>The content to be matched in a URL. Only requests with a URL that contains the specified content can be routed to the destination service.</p> <p>Turn on <b>Matching URI</b> and set the <b>Method</b> parameter to <b>Exact</b> and the <b>Content</b> parameter to <i>/productpage</i>.</p> <div> <p><b>Note</b> You can also click <b>Add Header Matching Rule</b> to configure settings for matching HTTP headers.</p> </div>

- c. In the **set route destination** step, click **Add Route Destination**, set the parameters that are described in the following table, and then click **Next**.

## ← Create

1 **set route detail**    2 **set route destination**    3 **advanced config**    4 **confirm route configuration**

⊕ Add Route Destination

\* select upstream service

Subset  Weight

Back Next

Parameter	Description
select upstream service	The service to which the routing rule applies. In this example, the value is <i>productpage</i> .
Subset	The version of the service to which the routing rule applies.
Weight	The traffic weight of the destination service.

- d. In the **advanced config** step, turn on **Fault Injection**. After that, turn on **Request Latency**, set the **Latency** parameter to *4s* and the **Injection Percentage** parameter to *100*, and then click **Next**.

The screenshot shows the 'Create' configuration page for a routing rule, specifically the 'advanced config' step (Step 3). The page has a progress bar at the top with four steps: 'set route detail' (Step 1, completed), 'set route destination' (Step 2, completed), 'advanced config' (Step 3, active), and 'confirm route configuration' (Step 4). Below the progress bar, there are several toggle switches: 'Redirect or Rewrite' (off), 'Timeout' (off), 'Retry' (off), 'Fault Injection' (on), 'Request Latency' (on), 'Request Abort' (off), 'VirtualService Delegate' (off), and 'Traffic Mirroring' (off). The 'Request Latency' section is expanded, showing two input fields: '\* Latency' with the value '4s' and '\* Injection Percentage' with the value '100'. At the bottom of the page, there are two blue buttons: 'Back' and 'Next'.

- e. In the **confirm route configuration** step, confirm the information about the routing rule and click **Create**.
- ii. Create a routing rule to match a URL that contains */static*.
- a. On the **Route management** page, click **Create**.
- b. In the **set route detail** step, set the parameters that are described in the following table, and then click **Next**.

## ← Create

1  
**set route detail**

2  
**set route destination**

3  
**advanced config**

4  
**confirm route configuration**

\* route type http ▼

Name static

description static resource

priority 0

Namespace default ▼ ↻

☒ Matching URI

\* Method Prefix ▼

\* Content /static

⊕ Add Header Matching Rule

Next

Parameter	Description
route type	The type of the traffic to be routed. In this example, the value is http.
Namespace	The namespace in which you create the routing rule. In this example, the value is default.
Name	The name of the routing rule. In this example, the value is static.
description	The description of the routing rule. In this example, the value is The routing rule used to match a URL that contains /static.
priority	The priority of the routing rule. A smaller value indicates a higher priority. If a URL matches multiple routing rules, the routing rule that has the highest priority takes precedence. In this example, the value is 0.

Parameter	Description
Matching URI	<p>The content to be matched in a URL. Only requests with a URL that contains the specified content can be routed to the destination service.</p> <p>Turn on <b>Matching URI</b> and set the <b>Method</b> parameter to <b>Prefix</b> and the <b>Content</b> parameter to <code>/static</code>.</p>

- c. The subsequent steps for creating the *static* routing rule are the same as those for creating the *productpage-route* routing rule. For more information, see the [Create a routing rule to match a URL that contains /productpage](#) section in this topic.
- iii. Create a routing rule to match a URL that contains `/login`.
  - a. On the **Route management** page, click **Create**.
  - b. In the **set route detail** step, set the parameters that are described in the following table, and then click **Next**.

← Create

1

2

3

4

set route detail

set route destination

advanced config

confirm route configuration

\* route type

http

Namespace

default

⌵

↺

Name

login

description

login request routing

priority

0

☒ Matching URI

\* Method

Prefix

⌵

\* Content

/login

⊕ Add Header Matching Rule

Next

Parameter	Description
route type	The type of the traffic to be routed. In this example, the value is http.

Parameter	Description
Namespace	The namespace in which you create the routing rule. In this example, the value is default.
Name	The name of the routing rule. In this example, the value is login.
description	The description of the routing rule. In this example, the value is The routing rule used to match a URL that contains /login.
priority	The priority of the routing rule. A smaller value indicates a higher priority. If a URL matches multiple routing rules, the routing rule that has the highest priority takes precedence. In this example, the value is 0.
Matching URI	<p>The content to be matched in a URL. Only requests with a URL that contains the specified content can be routed to the destination service.</p> <p>Turn on <b>Matching URI</b> and set the <b>Method</b> parameter to <b>Prefix</b> and the <b>Content</b> parameter to <i>/login</i>.</p>

- c. The subsequent steps for creating the *login* routing rule are the same as those for creating the *productpage-route* routing rule. For more information, see the [Create a routing rule to match a URL that contains /productpage](#) section in this topic.
- iv. Create a routing rule to match a URL that contains */logout*.
  - a. On the **Route management** page, click **Create**.
  - b. In the **set route detail** step, set the parameters that are described in the following table, and then click **Next**.

← Create

1

2

3

4

set route detail

set route destination

advanced config

confirm route configuration

\* route type

http

Namespace

default

Name

logout

description

logout route

priority

0

☒ Matching URI

\* Method

Prefix

\* Content

/logout

[Add Header Matching Rule](#)

Next

Parameter	Description
route type	The type of the traffic to be routed. In this example, the value is http.
Namespace	The namespace in which you create the routing rule. In this example, the value is default.
Name	The name of the routing rule. In this example, the value is logout.
description	The description of the routing rule. In this example, the value is The routing rule used to match a URL that contains /logout.
priority	The priority of the routing rule. A smaller value indicates a higher priority. If a URL matches multiple routing rules, the routing rule that has the highest priority takes precedence. In this example, the value is 0.



Parameter	Description
Matching URI	<p>The content to be matched in a URL. Only requests with a URL that contains the specified content can be routed to the destination service.</p> <p>Turn on <b>Matching URI</b> and set the <b>Method</b> parameter to <b>Prefix</b> and the <b>Content</b> parameter to <code>/logout</code>.</p>

- c. The subsequent steps for creating the `logout` routing rule are the same as those for creating the `productpage-route` routing rule. For more information, see the [Create a routing rule to match a URL that contains /productpage](#) section in this topic.
- v. Create a routing rule to match a URL that contains `/api/v1/products`.
  - a. On the **Route management** page, click **Create**.
  - b. In the **set route detail** step, set the parameters that are described in the following table, and then click **Next**.

**← Create**

1 **set route detail**      2 **set route destination**      3 **advanced config**      4 **confirm route configuration**

\* route type:       Namespace:

Name:

description:

priority:

☒ Matching URI

\* Method:       \* Content:

[⊕ Add Header Matching Rule](#)

**Next**

Parameter	Description
route type	The type of the traffic to be routed. In this example, the value is <code>http</code> .
Namespace	The namespace in which you create the routing rule. In this example, the value is <code>default</code> .

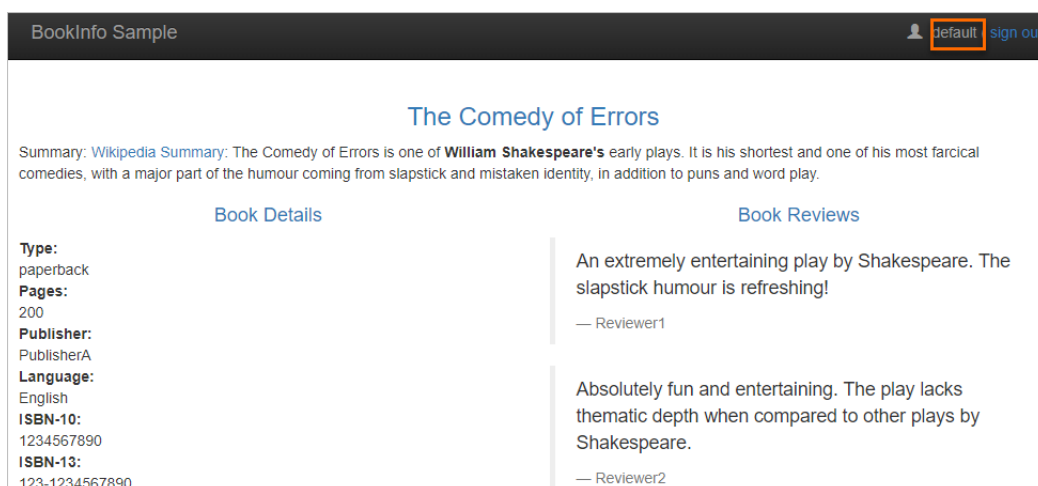
Parameter	Description
Name	The name of the routing rule. In this example, the value is <code>products-route</code> .
description	The description of the routing rule. In this example, the value is The routing rule used to match a URL that contains <code>/api/v1/products</code> .
priority	The priority of the routing rule. A smaller value indicates a higher priority. If a URL matches multiple routing rules, the routing rule that has the highest priority takes precedence. In this example, the value is 0.
Matching URI	<p>The content to be matched in a URL. Only requests with a URL that contains the specified content can be routed to the destination service.</p> <p>Turn on <b>Matching URI</b> and set the <b>Method</b> parameter to <b>Prefix</b> and the <b>Content</b> parameter to <code>/api/v1/products</code>.</p>

- c. The subsequent steps for creating the *products-route* routing rule are the same as those for creating the *productpage-route* routing rule. For more information, see the [Create a routing rule to match a URL that contains /productpage](#) section in this topic.

3. Check whether the routing rules take effect.

- i. In the address bar of your browser, enter `http://{IP address of the ASM gateway}/productpage`. In this example, Google Chrome is used.

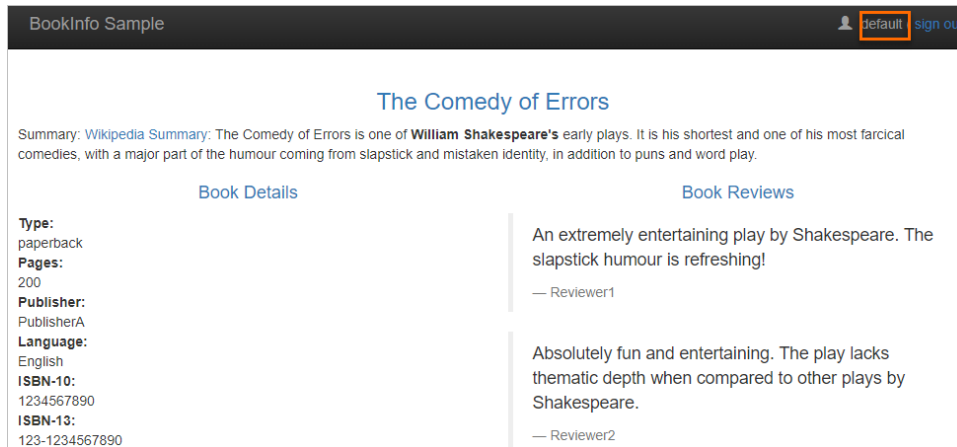
The following figure shows the sample page that appears.



- ii. In the upper-right corner of the Bookinfo page, click **Sign in**.

- iii. On the page that appears, enter your account and password and click **Sign in**.

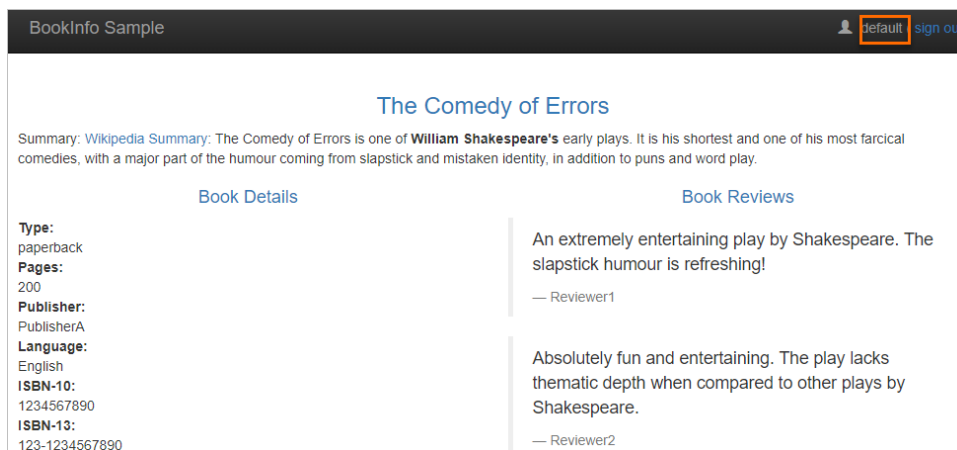
The following figure shows the page of the Bookinfo service after logon.



You can also use a URL that contains `/logout`, `/static`, or `/api/v1/products` to access the Bookinfo service. The corresponding procedures are omitted in this section.

- iv. In the upper-right corner of Google Chrome, click the More icon and choose **More Tools > Developer Tools**.
- v. Refresh the page whose address is `http://{IP address of the ASM gateway}/productpage`.

On the **Network** tab, a delay of about 4 seconds is reported.



4. (Optional) After the routing rules are created, view the YAML files that are automatically generated for the corresponding virtual services.

On the details page of the ASM instance, choose **Traffic Management > VirtualService** in the left-side navigation pane. On the **VirtualService** page, find the virtual service whose YAML file you want to view and click **YAML** in the **Actions** column. In the **Edit** panel, view the automatically generated YAML file.

## Related operations

### View the details of an imported service


View the details of an imported service, such as the settings of the sidecar proxy injection and the region.

- 1.
- 2.

- 3.
4. On the details page of the ASM instance, click **ASM Gateways** in the left-side navigation pane. On the page that appears, click the name of the ASM gateway for which you want to view service details.
5. On the details page of the ASM gateway, click **Upstream Service** in the left-side navigation pane.
6. On the **Upstream Service** page, find the service whose details you want to view and click **service details** in the **Actions** column.  
On the details page of the service, view the details of the service, such as the settings of the sidecar proxy injection and the region.

## Disable a routing rule

You can disable a routing rule so that this routing rule no longer takes effect.

 **Note** You can enable a disabled routing rule by performing the following steps:  
On the **Route management** page, find the routing rule that you want to enable and click **online** in the **Actions** column. In the message that appears, click **OK**.

- 1.
- 2.
- 3.
4. On the details page of the ASM instance, click **ASM Gateways** in the left-side navigation pane. On the page that appears, click the name of the ASM gateway for which you want to disable a routing rule.
5. On the details page of the ASM gateway, click **Route management** in the left-side navigation pane.
6. On the page that appears, find the routing rule named login and click **offline** in the **Actions** column.
7. In the message that appears, click **OK**.
8. Verify that the disabled routing rule no longer takes effect.
  - i. In the address bar of Google Chrome, enter *http://{IP address of the ASM gateway}/productpage*.
  - ii. In the upper-right corner of the Bookinfo page, click **Sign in**.
  - iii. On the page that appears, enter your account and password and click **Sign in**.  
A 404 error occurs, which indicates that the login routing rule is disabled and no longer takes effect.

## 2. Use ASM to transfer TCP traffic

You can use Alibaba Cloud Service Mesh (ASM) to transfer traffic from one version of an application to another. This topic describes how to use ASM to transfer TCP traffic.

### Prerequisites

- The following services are activated:
  - [Container Service for Kubernetes \(ACK\)](#)
  - [Server Load Balancer \(SLB\)](#)
- An ACK cluster is created. For more information, see [Create an ACK dedicated cluster](#) and [Create an ACK managed cluster](#).
- An ASM instance is created and 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](#).

### Context

This topic describes how to transfer TCP traffic between two versions of an application based on the TCP Traffic Shifting example given by Istio. In this example, an application named tcp-echo has two versions: v1 and v2. In version v1, the application adds the prefix "one" to the timestamps in the response before it returns the response. In version v2, the application adds the prefix "two" to the timestamps in the response before it returns the response.

### Step 1: Deploy the two versions of the sample application

1. Deploy the two versions of the tcp-echo application.
  - i.
  - ii.
  - iii.
  - iv. At the top of the **Deployments** page, select a namespace from the **Namespace** drop-down list.
  - v. In the upper-right corner of the Deployments page, click **Create from YAML**.
  - vi. Select **Custom** from the **Sample Template** drop-down list, copy the following YAML template to the code editor in the **Template** section, and then click **Create**.

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

Return to the **Deployments** page. Then, you can find the two versions of the tcp-echo application.


2. Create a service named tcp-echo and expose the service.

i.


- ii.
- iii.
- iv.
- v. At the top of the **Services** page, select a namespace from the **Namespace** drop-down list and click **Create** in the upper-right corner.
- vi. In the **Create Service** dialog box, set the following parameters and click **Create**.

The following section describes the parameters:


- **Name:** the name of the service. In this example, tcp-echo is used.
- **Type:** the type of the service, which specifies how the service is exposed. Valid values: **Cluster IP**, **Node Port**, and **Server Load Balancer**.

 **Note** The **Headless Service** check box is displayed only when you set the Type parameter to **Cluster IP**. If you select this check box, you can use a headless service to interface with other service discovery mechanisms, without being tied to the implementation of service discovery in Kubernetes.

- **Backend:** the Deployment to be associated with the service. In this example, tcp-echo-v1 is selected.

 **Note** The service uses the `app` label of the associated Deployment as the selector to determine to which Deployment the traffic is routed. The tcp-echo-v1 and tcp-echo-v2 Deployments share the same app label, which is `app:tcp-echo`. Therefore, the service can be associated with either one of the two Deployments.

- **External Traffic Policy:** You can select Local or Cluster.

 **Note** The **External Traffic Policy** parameter is displayed only when you set the Type parameter to **Node Port** or **Server Load Balancer**.

- **Port Mapping:** Set the **Name** parameter to tcp, the **Service Port** and **Container Port** parameters to 9000, and the **Protocol** parameter to TCP.
- **Annotations:** Add annotations to the service. For example, you can add the LoadBalancer annotation `service.beta.kubernetes.io/alibaba-cloud-loadbalancer-bandwidth:20`, which sets the bandwidth limit of the service to 20 Mbit/s. This annotation limits the amount of traffic flows to the service. For more information about annotations, see [Use annotations to configure load balancing](#).
- **Label:** Add one or more labels to the service.

After you create the service, the service appears on the **Services** page.

## Step 2: Set routing rules for the ASM instance

You can create an Istio gateway, a virtual service, and a destination rule for the ASM instance to route all traffic to version v1 of the tcp-echo application.

- 1.
- 2.

3.

4. Create a service gateway.

- i.
- ii. On the Create page, select default from the **Namespace** drop-down list and copy the following YAML template to the code editor. Then, click .

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

```

5. Create a virtual service.

- i.
- ii. On the Create page, select default from the **Namespace** drop-down list and copy the following YAML template to the code editor. Then, click .

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

```

6. Create a destination rule.

- i.



- ii. On the Create page, select default from the **Namespace** drop-down list and copy the following YAML template to the code editor. Then, click .

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

### Step 3: Deploy an ingress gateway service

Add port 31400 to the Ingress gateway and map the port to port 31400 of the Istio gateway.

- 1.
- 2.
- 3.
- 4.
5. Set the parameters about the ingress gateway. Use the default values for other parameters.
  - i. Select the cluster in which you want to deploy the Ingress gateway from the **Cluster** drop-down list.
  - ii. Set the SLB Instance Type parameter to **Internet Access**.

## iii. Configure an SLB instance.

- Use Existing SLB Instance: Select an SLB instance from the drop-down list.
- Create SLB Instance: Click **Create SLB Instance** and select an SLB instance type from the drop-down list.

**Note** We recommend that you assign a dedicated SLB instance to each Kubernetes service in the cluster. If multiple Kubernetes services share the same SLB instance, the following risks and limits exist:

- If you assign a Kubernetes service with an SLB instance that is used by another Kubernetes service, the existing listeners of the SLB instance are forcibly overwritten. This may interrupt the original Kubernetes service and make your application unavailable.
- If you create an SLB instance when you create a Kubernetes service, the SLB instance cannot be shared among Kubernetes services. Only SLB instances that you create in the SLB console or by calling API operations can be shared.
- Kubernetes services that share the same SLB instance must use different frontend listening ports. Otherwise, port conflicts may occur.
- If multiple Kubernetes services share the same SLB instance, you must use the listener names and the vServer group names as unique identifiers in Kubernetes. Do not modify the names of listeners or vServer groups.
- You cannot share SLB instances across clusters.

iv. Click **Add Port** and set the **Name** parameter to tcp and the **Service Port** and **Container Port** parameters to 31400.

**Note** The service port exposes the ASM instance to external access. The ASM instance distributes inbound traffic received on the service port to the container port, which refers to the specified port on the Istio gateway. Make sure that the container port is the same as the specified port on the Istio gateway.

v. Click **OK**.

## Step 4: Verify the result

Use a kubectl client to check whether the tcp-echo service distributes traffic to the application.

1. Use the kubectl client to connect to the ACK cluster. For more information, see [Step 2: Select a type of cluster credentials](#).
2. Run the following commands to query the IP address and port number of the tcp-echo service:

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

3. Run the telnet command to connect to the tcp-echo service.

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

4. Enter a random string and press Enter. If the return string is prefixed with "one", the tcp-echo service is properly deployed and all of the traffic is routed to the tcp-echo-v1 application.

## Step 5: Transfer a specific proportion of traffic to the tcp-echo-v2 version

In this example, 20% of the traffic is routed to the tcp-echo-v2 version and the remaining 80% is routed to the tcp-echo-v1 version.


1. Modify the configuration of the virtual service of the ASM instance.
  - i.
  - ii. On the page, find the tcp-echo virtual service and click **YAML** in the **Actions** column.
  - iii. In the **Edit** panel, copy the following YAML template to the code editor and click **OK**:

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

2. Run the following command to send 10 requests to the tcp-echo service:

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

The preceding output indicates that 20% of the traffic is routed to the tcp-echo-v2 version.

 **Note** If you send 10 requests in a test, the traffic may not always be routed to the tcp-echo-1 and tcp-echo-v2 versions at the specified ratio. However, the actual ratio is close to 80:20 when the sample size increases.

## 3. Use ASM to achieve gRPC load balancing

When you use HTTP 2-based gRPC to call services in Kubernetes, a connection is always pointed to a specific pod. If the gRPC client sends multiple requests, all of the requests are routed to the pod for processing, which causes unbalanced load. This topic describes the unbalanced load issue between gRPC service calls. This topic also shows you how to use Alibaba Cloud Service Mesh (ASM) to achieve gRPC load balancing.

### Context

gRPC is an HTTP 2-based communication protocol for services. You can specify service definitions in a format that is called protocol buffers. The data of service calls is serialized into a small binary format for transmission. gRPC allows you to generate boilerplate code from *.proto* files into multiple programming languages. This makes gRPC an ideal choice for polyglot microservices.

HTTP 1.1-based remote procedure calls (RPCs) build temporary connections between the client and pods. In this case, a TCP load balancer is enough to balance the load between RPCs. However, for an HTTP 2-based gRPC service call, the TCP connection between the client and a pod is persistent. If the pod expires, the load in the cluster becomes unbalanced.

### Unbalanced load between gRPC service calls

The following example shows unbalanced load between gRPC service calls.

Prerequisites:

- A Container Service for Kubernetes (ACK) cluster is created.
- The `kubectl` client is connected to the ACK cluster. For more information, see [Connect to ACK clusters by using kubectl](#).

Procedure

- 1.
- 2.
- 3.
- 4.
5. On the **Namespace** page, click **Create** in the upper-right corner. In the **Create Namespace** dialog box, enter a name for the namespace, such as `grpc-nosidecar`, and click **OK**.
6. Deploy a gRPC server that is named `istio-grpc-server` in the created `grpc-nosidecar` namespace.

Run the following command to deploy the gRPC server by using the description YAML file *istio-grpc-server.yaml*:

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

The *istio-grpc-server.yaml* file contains the following content:

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

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

```

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

```

7. Deploy a gRPC client that is named `istio-grpc-client` in the created `grpc-nosidecar` namespace.

Run the following command to deploy the gRPC client by using the description YAML file *istio-grpc-client.yaml*:

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

The *istio-grpc-client.yaml* file contains the following content:

```

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

8. Run the following command to query the pod status:

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

The command output is similar to the following example:

NAME	READY	STATUS	RESTARTS	AGE
istio-grpc-client-dd56bcb45-hvmjt	1/1	Running	0	95m
istio-grpc-server-v1-546d9876c4-j2p9r	1/1	Running	0	95m
istio-grpc-server-v2-66d9b8847-276bd	1/1	Running	0	95m

9. Run the following command to log on to the default container of the pod of the gRPC client:

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

10. After you log on to the container, run the following command:

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



The command output indicates that all requests are routed to the same pod:

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

All requests from the gRPC client are processed by the same pod, which causes unbalanced load.

## Use ASM to balance the load between gRPC service calls

The following example describes how to use ASM to achieve gRPC load balancing.

Prerequisites:

- An ASM instance is created and an ACK cluster is added to the instance.
- The kubectl client is connected to the ACK cluster. For more information, see [Connect to ACK clusters by using kubectl](#).

Procedure

- 1.
- 2.
- 3.
- 4.
5. On the **Namespace** page, click **Create** in the upper-right corner. In the **Create Namespace** dialog box, enter a namespace name, such as `grpc-sidecar`, create a tag `istio-injection:enabled`, and then click **OK**.
6. Deploy a gRPC server that is named `istio-grpc-server` in the created `grpc-sidecar` namespace.

Run the following command to deploy the gRPC server by using the description YAML file *istio-grpc-server.yaml*:

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

For more information about the *istio-grpc-server.yaml* file, see the sample file in the preceding section.

7. Deploy a gRPC client that is named `istio-grpc-client` in the created `grpc-sidecar` namespace.

Run the following command to deploy the gRPC client by using the description YAML file *istio-grpc-client.yaml*:

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

For more information about the *istio-grpc-client.yaml* file, see the sample file in the preceding section.

8. Run the following command to query the pod status:

```
kubectl get pod -n grpc-sidecar
```

The command output indicates that each pod contains two containers, one of which is the container for the injected sidecar proxy. The command output is similar to the following

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

example:

9. Run the following command to log on to the default container of the pod of the gRPC client:

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

10. After you log on to the container, run the following command:

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

The command output indicates that requests are routed to the two pods at a ratio of almost 50 to 50 in round-robin scheduling mode:

```
2020/01/14 14:53:16 Hello world from istio-grpc-server-v1-546d9876c4-tndsm
2020/01/14 14:53:16 Hello world from istio-grpc-server-v2-66d9b8847-99v62
2020/01/14 14:53:16 Hello world from istio-grpc-server-v1-546d9876c4-tndsm
2020/01/14 14:53:16 Hello world from istio-grpc-server-v2-66d9b8847-99v62
2020/01/14 14:53:16 Hello world from istio-grpc-server-v1-546d9876c4-tndsm
2020/01/14 14:53:16 Hello world from istio-grpc-server-v2-66d9b8847-99v62
2020/01/14 14:53:16 Hello world from istio-grpc-server-v1-546d9876c4-tndsm
2020/01/14 14:53:16 Hello world from istio-grpc-server-v2-66d9b8847-99v62
2020/01/14 14:53:16 Hello world from istio-grpc-server-v1-546d9876c4-tndsm
2020/01/14 14:53:16 Hello world from istio-grpc-server-v2-66d9b8847-99v62
2020/01/14 14:53:16 Hello world from istio-grpc-server-v1-546d9876c4-tndsm
2020/01/14 14:53:16 Hello world from istio-grpc-server-v2-66d9b8847-99v62
```

11. Configure the control plane for the ASM instance.

- i.
- ii.
- iii.
- iv.
- v. In the **Create Namespace** dialog box, enter a name for the namespace, such as `grpc-sidecar`, and click **OK**.

- vi. Create a destination rule by using the following content. For more information, see [Manage destination rules](#).

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

- vii. Create a virtual service by using the following content. For more information, see [Manage virtual services](#).

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

- viii. Run the following command to log on to the default container of the pod of the gRPC client:

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

- ix. After you log on to the container, run the following command:

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

The command output indicates that requests are routed to the two pods at a ratio of 90 to 10, instead of 50 to 50 in round-robin scheduling mode.

- x. On the details page of the ASM instance, choose Traffic Management > in the left-side navigation pane. Click **YAML** in the Actions column of the `vs-istio-grpc-server` virtual service that is deployed in the `grpc-sidecar` namespace to modify the weight assigned to each service version.

Use the following content to modify the weight:

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

- xi. Use the `kubectl` client to log on to the default container of the pod of the gRPC client and run the following command:

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

The command output indicates that all requests are routed to the pod of the version v2.

## 4. Use EnvoyFilter to add HTTP response headers in ASM

You can add HTTP response headers for web applications to improve the security of the web applications. This topic describes how to define an EnvoyFilter resource to add HTTP response headers in Alibaba Cloud Service Mesh (ASM).

### Prerequisites

- 
- The kubectl client is connected to the ACK cluster. For more information, see [Use kubectl to connect to an ASM instance](#).
- Applications are deployed in the ASM instance. For more information, see [Deploy an application in an ASM instance](#).
- Istio resources are defined. For more information, see [Define Istio resources](#).

### Context

Open Web Application Security Project (OWASP) provides best practices and a coding framework to describe how to use HTTP response headers to improve the security of applications. The following table describes basic HTTP response headers.

Basic HTTP response headers

HTTP response header	Default value	Description
Content-Security-Policy	frame-ancestors none;	Prevents clickjacking attacks from other websites.
X-XSS-Protection	1;mode=block	Activates the cross-site scripting (XSS) filter (if it is available) of a browser so that the browser can stop rendering when XSS attacks are detected.
X-Content-Type-Options	Nosniff	Disables content sniffing of a browser.
Referrer-Policy	no-referrer	Specifies to send no referrer information along with requests.
X-Download-Options	noopen	Prevents old versions of Internet Explorer from allowing downloads to be automatically executed.
X-DNS-Prefetch-Control	off	Disables DNS prefetching for external hyperlinks on web pages.

HTTP response header	Default value	Description
Server	envoy	The server that generates the response. This HTTP response header is automatically set by the Istio ingress gateway.
X-Powered-by	N/A	Contains information about the hosting environments or other frameworks. Do not set this HTTP response header if you want to hide the name and version information of vulnerable application servers.
Feature-Policy	camera 'none'; microphone 'none'; geolocation 'none'; encrypted-media 'none'; payment 'none'; speaker 'none'; usb 'none';	Specifies the features and API operations that are available to browsers.

This topic takes the Bookinfo application as an example. For more information, see [Deploy an application in an ASM instance](#). You can run the following curl command to query the HTTP response headers of the application:

```
curl -I http://{IP address of the ingress gateway service}/productpage
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5183
server: istio-envoy
date: Tue, 28 Jan 2020 08:15:21 GMT
x-envoy-upstream-service-time: 28
```

The command output indicates that the response from the application homepage does not contain the security-related HTTP response headers that are described in [Basic HTTP response headers](#). You can define an EnvoyFilter resource to add security-related HTTP response headers for the application.

## Procedure

1. Run the following command to deploy an Istio service entry:

```
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: addheader-into-ingressgateway
  namespace: istio-system
spec:
```

```

workloadselector:
  # select by label in the same namespace
  labels:
    istio: ingressgateway
configPatches:
  # The Envoy config you want to modify
  - applyTo: HTTP_FILTER
    match:
      context: GATEWAY
      proxy:
        proxyVersion: '^1\.8.*'
    listener:
      filterChain:
        filter:
          name: "envoy.http_connection_manager"
          subFilter:
            name: "envoy.router"
    patch:
      operation: INSERT_BEFORE
      value: # lua filter specification
        name: envoy.lua
        typed_config:
          "@type": "type.googleapis.com/envoy.config.filter.http.lua.v2.Lua"
          inlineCode: |-
            function envoy_on_response(response_handle)
              function hasFrameAncestors(rh)
                s = rh:headers():get("Content-Security-Policy");
                delimiter = ";";
                defined = false;
                for match in (s..delimiter):gmatch("(.)"..delimiter) do
                  match = match:gsub("%s+", "");
                  if match:sub(1, 15)=="frame-ancestors" then
                    return true;
                  end
                end
                return false;
              end
              if not response_handle:headers():get("Content-Security-Policy") then
                csp = "frame-ancestors none;";
                response_handle:headers():add("Content-Security-Policy", csp);
              elseif response_handle:headers():get("Content-Security-Policy") then
                if not hasFrameAncestors(response_handle) then
                  csp = response_handle:headers():get("Content-Security-Policy");
                  csp = csp .. ";frame-ancestors none;";
                  response_handle:headers():replace("Content-Security-Policy", csp);
                end
              end
              if not response_handle:headers():get("X-Frame-Options") then
                response_handle:headers():add("X-Frame-Options", "deny");
              end
              if not response_handle:headers():get("X-XSS-Protection") then
                response_handle:headers():add("X-XSS-Protection", "1; mode=block");
              end
              if not response_handle:headers():get("X-Content-Type-Options") then
                response_handle:headers():add("X-Content-Type-Options", "nosniff");
              end
            end

```

```

response_handle:headers():add("X-Download-Options", "noopen");
end
if not response_handle:headers():get("Referrer-Policy") then
response_handle:headers():add("Referrer-Policy", "no-referrer");
end
if not response_handle:headers():get("X-DNS-Prefetch-Control") then
response_handle:headers():add("X-DNS-Prefetch-Control", "off");
end
if not response_handle:headers():get("Feature-Policy") then
response_handle:headers():add("Feature-Policy",
                                "camera 'none';"..
                                "microphone 'none';"..
                                "geolocation 'none';"..
                                "encrypted-media 'none';"..
                                "payment 'none';"..
                                "speaker 'none';"..
                                "usb 'none';");
end
if response_handle:headers():get("X-Powered-By") then
response_handle:headers():remove("X-Powered-By");
end
end
end

```

proxyVersion: Set this parameter to your Istio version. When you create an EnvoyFilter resource, you must set the proxyVersion parameter to specify the Istio version to which the EnvoyFilter resource applies. Some fields in the EnvoyFilter configuration may be incompatible across different Istio versions. The EnvoyFilter configuration varies with Istio versions.

- If you use Istio 1.8 or later, set the proxyVersion parameter to your Istio version.
- If you use Istio 1.9 or later, set the proxyVersion parameter to your Istio version and replace *envoy.http\_connection\_manager* with *envoy.filters.network.http\_connection\_manager*, *envoy.router* with *envoy.filters.http.router*, and *type.googleapis.com/envoy.config.filter.http.lua.v2.Lua* with *type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua* in the EnvoyFilter configuration.

## 2. Verify the HTTP response headers.

Run the following curl command to check whether the HTTP response headers are added for the application:



```
curl -I http://{IP address of the ingress gateway service}/productpage
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 4183
server: istio-envoy
date: Tue, 28 Jan 2020 09:07:01 GMT
x-envoy-upstream-service-time: 17
content-security-policy: frame-ancestors none;
x-frame-options: deny
x-xss-protection: 1; mode=block
x-content-type-options: nosniff
referrer-policy: no-referrer
x-download-options: noopen
x-dns-prefetch-control: off
feature-policy: camera 'none';microphone 'none';geolocation 'none';encrypted-media 'none';payment 'none';speaker 'none';usb 'none';
```

The command output indicates that the response from the application homepage contains the security-related HTTP response headers that are described in [Basic HTTP response headers](#).

## 5. Use ASM to route traffic based on the locality

In large-scale business scenarios, thousands of services may call each other across different regions to implement required features. By default, Kubernetes implements load balancing in a round-robin manner. However, to achieve the best service performance, we recommend that you route traffic to the closest services and limit the traffic to the same region as much as possible. Alibaba Cloud Service Mesh (ASM) is based on Istio and provides locality-based routing. You can use ASM to route traffic to pods that are closest to the originating pod. This reduces service latency and saves traffic fees because services call each other within the same region as much as possible. This topic shows you how to use ASM to route traffic based on the locality. You can use this feature to improve service performance and save costs.

### Prerequisites

- An ASM instance is created. For more information, see [Create an ASM instance](#).
- A managed multi-zone cluster is created in the Container Service for Kubernetes (ACK) console. Nodes in the ACK cluster reside in different zones of the same region. In this topic, Zone G and Zone H of the China (Beijing) region are used. For more information, see [Create an ACK managed cluster](#).

### Deploy a sample backend service

Deploy the version 1 and version 2 of the backend service that is named `nginx` in the ACK cluster. Configure node selectors to assign the pod of the version 1 to the node in Zone G and the pod of the version 2 to the node in Zone H, as described in the following steps:

1. Deploy the version 1 of the `nginx` backend service in the ACK cluster. The following YAML code provides an example:

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

2. Configure a node selector to assign the pod of the version 1 of the `nginx` backend service to the node in Zone G. The following YAML code provides an example:

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

3. Deploy the version 2 of the nginx backend service in the ACK cluster. The following YAML code provides an example:

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

4. Configure a node selector to assign the pod of the version 2 of the nginx backend service to the node in Zone H. The following YAML code provides an example:

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

5. Deploy the nginx backend service. The following YAML code provides an example:

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

## Deploy a sample client service

Deploy the version 1 and version 2 of the client service that is named sleep in the ACK cluster. Configure node selectors to assign the pod of the version 1 to the node in Zone G and the pod of the version 2 to the node in Zone H, as described in the following steps:

1. Deploy the version 1 of the sleep client service in the ACK cluster and configure a node selector to assign the pod of the version 1 to the node in Zone G. The following YAML code provides an example:

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

2. Deploy the version 2 of the sleep client service in the ACK cluster and configure a node selector to assign the pod of the version 2 to the node in Zone H. The following YAML code provides an example:

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

3. Deploy the sample client service. The following YAML code provides an example:

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

4. Execute the following script to access the nginx backend service from the two pods of the sleep client service:

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

### Verify the result

The following execution result indicates that traffic is sent to the two pods of the nginx backend service in round-robin mode:

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

## Locality-prioritized load balancing

By default, the algorithm of locality-prioritized load balancing guides traffic from a client service pod to a backend service pod in the same zone. If no backend service pod in the same zone is available, the traffic is guided to a pod in another zone.

To enable locality-prioritized load balancing, you must configure a virtual service and a destination rule with an outlier policy. The outlier policy checks the health status of pods and makes decisions on routing.

- 1.



- 2.
- 3.
- 4.
5. On the Create page, perform the following steps to define a virtual service. Then, click .
  - i. Select a namespace as required. In this example, the backend namespace is selected.
  - ii. Copy the following YAML code to the code editor to define a virtual service:

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

- 6.
7. On the Create page, perform the following steps to define a destination rule. Then, click .
  - i. Select a namespace as required. In this example, the backend namespace is selected.
  - ii. Copy the following YAML code to the code editor to define a destination rule:

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

8. Execute the following script to access the nginx backend service from the two pods of the sleep client service:

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

### Verify the result

The following execution result indicates that traffic is sent to the two pods of the nginx backend service in locality-prioritized mode:

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

## Locality-weighted load balancing

Most use cases work well with locality-prioritized load balancing. However, sometimes you may want to split traffic and send the traffic to multiple zones. For example, if all traffic comes from a zone, you can split traffic to prevent the zone from being overloaded. In this case, you can use locality-weighted load balancing.

In this example, the following rules are applied:

- Route 80% of traffic from China (Beijing) Zone G to China (Beijing) Zone G and 20% to China (Beijing) Zone H.

- Route 20% of traffic from China (Beijing) Zone H to China (Beijing) Zone G and 80% to China (Beijing) Zone H.

- 1.
- 2.
- 3.
- 4.

5. On the Create page, perform the following steps to define a virtual service. Then, click .
  - i. Select a namespace as required. In this example, the backend namespace is selected.
  - ii. Copy the following YAML code to the code editor to define a virtual service:

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

- 6.
7. On the Create page, perform the following steps to define a destination rule. Then, click .
  - i. Select a namespace as required. In this example, the backend namespace is selected.

- ii. Copy the following YAML code to the code editor to define a destination rule:

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

8. Execute the following script to access the nginx backend service from the two pods of the sleep client service:

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

### Verify the result

The following execution result indicates that traffic is sent to the two pods of the nginx backend service in locality-weighted mode:

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

## 6. Use an ingress gateway to access a gRPC service in an ASM instance


You can route traffic to gRPC services in an Alibaba Cloud Service Mesh (ASM) instance by using an ingress gateway. This topic describes how to use an ingress gateway to access a gRPC service in an ASM instance. This topic also describes how to switch traffic between two versions of a gRPC service.

### Prerequisites

#### Step 1: Deploy the two versions of a gRPC service

Deploy version 1 and version 2 of a gRPC service: `istio-grpc-server-v1` and `istio-grpc-server-v2`.

- 1.
- 2.
- 3.
- 4.
5. At the top of the **Deployments** page, select a namespace from the **Namespace** drop-down list. Click **Create from YAML**.

 **Note** Select a namespace that has the `istio-system=enabled` tag, which indicates that automatic sidecar injection is enabled. For more information, see [Upgrade sidecar proxies](#).

6. Copy the following YAML code to the Template editor. Then, click **Create**.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: istio-grpc-server-v1
  labels:
    app: istio-grpc-server
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: istio-grpc-server
      version: v1
  template:
    metadata:
      labels:
        app: istio-grpc-server
        version: v1
    spec:
      containers:
        - args:
            - --address=0.0.0.0:8080
          image: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-server
```

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



```

      - --addr=:8080
      initialDelaySeconds: 2
---
apiVersion: v1
kind: Service
metadata:
  name: istio-grpc-server
  labels:
    app: istio-grpc-server
spec:
  ports:
    - name: grpc-backend
      port: 8080
      protocol: TCP
  selector:
    app: istio-grpc-server
  type: ClusterIP
---

```

## Step 2: Set routing rules for the ASM instance

Create an Istio gateway, a virtual service, and a destination rule for the ASM instance to route all inbound traffic to istio-grpc-server-v1.

- 1.
- 2.
- 3.
4. Create an .
  - i.
  - ii. In the **Create** panel, select **default** from the **Namespace** drop-down list and copy the following YAML code to the code editor. Then, click **Create**.

```

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

```

5. Create a .
  - i.

- ii. In the **Create** panel, select **default** from the **Namespace** drop-down list and copy the following YAML code to the code editor. Then, click **Create**.

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

## 6. Create a .

- i.
- ii. In the **Create** panel, select **default** from the **Namespace** drop-down list and copy the following YAML code to the code editor. Then, click **Create**.


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


## Step 3: Deploy an ingress gateway service

Enable port 8080 in the ingress gateway service and point the port to port 8080 of the Istio gateway.

- 1.
- 2.

- 3.
- 4.
5. On the **ASM Gateways** page, click **Create**. In the **Create** panel, set the parameters as required.

Parameter	Description
Cluster	Select the cluster in which you want to deploy an ingress gateway service.
SLB Instance Type	Select <b>Internet Access</b> .
Use Existing SLB Instance or Create SLB Instance	<p>Configure a Server Load Balancer (SLB) instance.</p> <ul style="list-style-type: none"> <li>◦ Use Existing SLB Instance: Select an SLB instance from the drop-down list.</li> <li>◦ Create SLB Instance: Click <b>Create SLB Instance</b> and select an SLB instance type from the drop-down list.</li> </ul> <div style="background-color: #e6f2ff; padding: 10px; margin-top: 10px;"> <p> <b>Note</b> We recommend that you assign a dedicated SLB instance to each Kubernetes service in the cluster. If multiple Kubernetes services share the same SLB instance, the following risks and limits exist:</p> <ul style="list-style-type: none"> <li>◦ If you assign a Kubernetes service with an SLB instance that is used by another Kubernetes service, the existing listeners of the SLB instance are forcibly overwritten. This may interrupt the original Kubernetes service and make your application unavailable.</li> <li>◦ If you create an SLB instance when you create a Kubernetes service, the SLB instance cannot be shared among Kubernetes services. Only SLB instances that you create in the SLB console or by calling API operations can be shared.</li> <li>◦ Kubernetes services that share the same SLB instance must use different frontend listening ports. Otherwise, port conflicts may occur.</li> <li>◦ If multiple Kubernetes services share the same SLB instance, you must use the listener names and the VServer group names as unique identifiers in Kubernetes. Do not modify the names of listeners or VServer groups.</li> <li>◦ You cannot share SLB instances across clusters.</li> </ul> </div>

Parameter	Description
Port Mapping	<p>Click <b>Add Port</b> and set the <b>Protocol</b> parameter to TCP, the <b>Service Port</b> parameter to 8080, and the <b>Container Port</b> parameter to 8080.</p> <div style="background-color: #e6f2ff; padding: 10px; border: 1px solid #d9e1f2;"> <p> <b>Note</b> The service port exposes the ASM instance to external access. Inbound traffic accesses the ASM instance by using the service port and is routed to a port on the Istio gateway, which serves as the container port. Make sure that the container port that you enter is the same as the specified port on the Istio gateway.</p> </div>

6. Click **Create**.

## Step 4: Start the gRPC client

1. Run the following command to start the gRPC client:

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

2. Run the following command to log on to the default container of the pod where the gRPC client resides:

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

3. Run the following command to access the gRPC service that you deployed in the ASM instance:

```
/bin/greeter-client --insecure=true --address=<IP address of the ingress gateway service>:8080 --repeat=100
```

The command output indicates that all requests are routed to istio-grpc-server-v1.

```
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
```

## Step 5: Transfer a specific ratio of the traffic to istio-grpc-server-v2

Route 40% of the traffic to istio-grpc-server-v2 and 60% of the traffic to istio-grpc-server-v1.

1.


- 2.
- 3.
- 4.
5. On the page, click **YAML** in the **Actions** column of the `grpc-vs` virtual service.
6. In the **Edit** panel, copy the following YAML code to the code editor. Then, click **OK**.

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

7. Log on to the default container of the pod where the gRPC client resides. Run the following command to access the gRPC service that you deployed in the ASM instance:

```
/bin/greeter-client --insecure=true --address=<IP address of the ingress gateway service>:8080 --repeat=100
```

The command output indicates that 40% of the traffic is routed to `istio-grpc-server-v2`.

 **Note** If you send 100 requests in a test, the traffic may not always be routed to `istio-grpc-server-v1` and `istio-grpc-server-v2` at an exact ratio of 60 to 40, but will be close.

```
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h741w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h741w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n851w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h741w
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h741w
```

# 7. Use an ingress gateway to access a WebSocket service in an ASM instance

WebSocket is a computer communications protocol that provides full-duplex communication channels over a single Transmission Control Protocol (TCP) connection. WebSocket is located at the application layer in the Open Systems Interconnection (OSI) model. WebSocket allows a server to push data to clients. Services that comply with WebSocket are WebSocket services. This topic shows you how to use an ingress gateway to access a WebSocket service in an Alibaba Cloud Service Mesh (ASM) instance.

## Prerequisites

### Step 1: Deploy a sample application

1. Use `kubectl` to connect to the ACK cluster. For more information, see [Connect to ACK clusters by using `kubectl`](#).
2. Use the following content to create a YAML file that is named *tornado*:

```
apiVersion: v1
kind: Service
metadata:
  name: tornado
  labels:
    app: tornado
    service: tornado
spec:
  ports:
    - port: 8888
      name: http
  selector:
    app: tornado
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tornado
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tornado
      version: v1
  template:
    metadata:
      labels:
        app: tornado
        version: v1
    spec:
      containers:
        - name: tornado
          image: hiroakis/tornado-websocket-example
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8888
---
```

3. Run the following command to create the tornado application:

```
kubectl apply -f tornado.yaml
```

## Step 2: Configure a routing rule

- 1.
- 2.
- 3.
4. Create an Istio gateway.
  - i.

- ii. On the Create page, select **default** from the **Namespace** drop-down list and copy the following content to the code editor. Then, click .

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

```

Set the `number` parameter to `80` . This way, the WebSocket service can receive inbound or outbound HTTP and TCP traffic over port 80.

#### 5. Create a virtual service.

- i.
- ii. On the Create page, select **default** from the **Namespace** drop-down list and copy the following content to the code editor. Then, click .

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: tornado
spec:
  hosts:
  - "*"
  gateways:
  - tornado-gateway
  http:
  - match:
    - uri:
        prefix: /
      route:
      - destination:
          host: tornado
          weight: 100

```

Set the `hosts` parameter to `*` . This way, all requests can access the WebSocket service.

### Step 3: Query the IP address of the ingress gateway

- 1.
- 2.
- 3.
- 4.



5. At the top of the **Services** page, select **istio-system** from the **Namespace** drop-down list. Find the ingress gateway that is named **istio-ingressgateway** and view the IP address whose port is 80 in the **External Endpoint** column.

## Step 4: Use the ingress gateway to access the WebSocket service

1. Enter `http://<IP address of the ingress gateway>` in the address bars of four different browsers.

### tornado WebSocket example

WebSocket status : open

The following table shows values by using WebSocket

No.	id	value
1	id 1	0
2	id 2	0
3	id 3	0

No.	id	value
4	id 4	0
5	id 5	0
6	id 6	0

No.	id	value
7	id 7	0
8	id 8	0
9	id 9	0

2. Run the following commands to access the WebSocket service:

```
curl "http://<IP address of the ingress gateway>/api?id=8&value=300"
```

```
curl "http://<IP address of the ingress gateway>/api?id=5&value=600"
```

```
curl "http://<IP address of the ingress gateway>/api?id=1&value=200"
```

```
curl "http://<IP address of the ingress gateway>/api?id=3&value=290"
```

View the data of the WebSocket service in the four browsers. The data of the WebSocket service in the four browsers is updated at the same time, and the same data is displayed.

## 8. Access services over WebSocket connections in ASM

WebSocket is a communications protocol that enables interaction between a client and a server. The WebSocket protocol is standardized by RFC 6455. Istio sidecar proxies support the WebSocket protocol out of the box and allow you to access services by establishing WebSocket connections with ease. This topic describes how to access services over WebSocket connections initiated by using HTTP/1.1 or HTTP/2 in Alibaba Cloud Service Mesh (ASM).

### Prerequisites

- 
- Sidecar injection is enabled for the specified namespace. For more information, see [Enable automatic sidecar injection by using multiple methods](#). In this example, the default namespace is used.

### Context

A WebSocket connection is established by using the HTTP Upgrade header. This is different from the way to establish an HTTP connection. Istio cannot recognize the WebSocket protocol. However, Istio sidecar proxies provide out-of-the-box support for the WebSocket protocol. For more information about how to use the WebSocket protocol in Istio, see [HTTP upgrades](#), [HTTP connection manager](#), and [Protocol Selection](#).

WebSocket connections initiated by using HTTP/1.1 and those initiated by using HTTP/2 have the following differences:

- WebSocket connections initiated by using HTTP/1.1: A connection is established to process only one request. After a response is returned for the request, the connection is closed.
- WebSocket connections initiated by using HTTP/2: Multiple requests can be processed in parallel over a connection. If a single request is time-consuming, other requests over the same connection are not affected.

### Deploy a WebSocket client and a WebSocket server

1. [Connect to ACK clusters by using kubectl](#).
2. Deploy a WebSocket server.

In this example, the WebSocket server for Python provided by the WebSocket community is used. For more information about how to modify the configurations for deploying the WebSocket server, see [Deploy to Kubernetes](#).

- i. Create the *websockets-server.yaml* file that contains the following content :

```
apiVersion: v1
kind: Service
metadata:
  name: websockets-server
  labels:
    app: websockets-server
spec:
  type: ClusterIP
  ports:
    - port: 8080
      targetPort: 80
      name: http-websocket
  selector:
    app: websockets-server
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: websockets-server
  labels:
    app: websockets-server
spec:
  selector:
    matchLabels:
      app: websockets-server
  template:
    metadata:
      labels:
        app: websockets-server
    spec:
      containers:
        - name: websockets-test
          image: registry.cn-beijing.aliyuncs.com/aliacs-app-catalog/istio-websockets
            -test:1.0
          ports:
            - containerPort: 80
```

- ii. Deploy the WebSocket server in the default namespace.

```
kubectl apply -f websockets-server.yaml -n default
```

### 3. Deploy a WebSocket client.

In this example, a Dockerfile is used to build an image for a WebSocket client. The *websockets-client.yaml* file contains the configurations for using the image of the WebSocket client to deploy the WebSocket client to a cluster.

```
FROM python:3.9-alpine
RUN pip3 install websockets
```

- i. Create the *websockets-client.yaml* file that contains the following content:

```
apiVersion: v1
kind: Service
metadata:
  name: websockets-client
  labels:
    app: websockets-client
spec:
  type: ClusterIP
  ports:
    - port: 8080
      targetPort: 80
      name: http-websockets-client
  selector:
    app: websockets-client
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: websockets-client-sleep
  labels:
    app: websockets-client
spec:
  selector:
    matchLabels:
      app: websockets-client
  template:
    metadata:
      labels:
        app: websockets-client
    spec:
      containers:
        - name: websockets-client
          image: registry.cn-beijing.aliyuncs.com/aliacs-app-catalog/istio-websockets
            -client-test:1.0
          command: ["sleep", "14d"]
```

- ii. Deploy the WebSocket client in the default namespace.

```
kubectl apply -f websockets-client.yaml -n default
```

## Use WebSocket connections initiated by using HTTP/1.1

Even if the WebSocket client sends multiple requests to the WebSocket server, the logs of sidecar proxies contain only one access log entry for the WebSocket connection. Envoy treats WebSocket connections as TCP byte streams and can recognize only requests and responses that contain the HTTP Upgrade header. Therefore, Envoy generates an access log entry only after the connection is closed. In addition, the log entry does not contain details of each TCP request.

1. Open the CLI shell in the container of the WebSocket client.
  - i.
  - ii.

- iii.
- iv.
- v. On the **Pods** page, find `websockets-client` and click **Terminal** in the **Actions** column. After that, click **Container: websockets-client**.
- vi. Run the following command to open the CLI shell.

```
kubectl exec -it -n <namespace> websockets-client-sleep... -c websockets-client -- sh
```

2. Run the following command to access the WebSocket server:

```
python3 -m websockets ws://websockets-server.<namespace>.svc.cluster.local:8080
```

Expected output:

```
Connected to ws://websockets-server.default.svc.cluster.local:8080.
```

If you enter "hello" and "world", the two words are returned as expected.

```
> hello
< hello
> world
< world
Connection closed: 1000 (OK).
```

3. Query the logs of sidecar proxies.

- o Query the logs of the sidecar proxy for the WebSocket client.
  - a. On the **Pods** page, click the name of the container of the WebSocket client.
  - b. Click the **Logs** tab and select **istio-proxy** from the **Container** drop-down list.

You can see that the logs contain an entry about the WebSocket connection initiated by using HTTP/1.1.

```
{... "upstream_host": "10.208.0.105:80", "bytes_sent": 23, "protocol": "HTTP/1.1", ...}
```

- o Query the logs of the sidecar proxy for the WebSocket server.
  - a. On the **Pods** page, click the name of the container of the WebSocket server.
  - b. Click the **Logs** tab and select **istio-proxy** from the **Container** drop-down list.

You can see that the logs contain a record of the WebSocket connection initiated by using HTTP/1.1.

```
{... "downstream_local_address": "10.208.0.105:80", "upstream_local_address": "127.0.0.1:53983", "protocol": "HTTP/1.1", ...}
```

## Use WebSocket connections initiated by using HTTP/2

If you use Istio of a version earlier than 1.12, WebSocket connections cannot be established to the WebSocket server by upgrading HTTP/2 connections. In this case, the HTTP status code 503 is returned.

If the HTTP status code 503 is returned after you send a request to upgrade an HTTP/2 connection to a WebSocket connection, you can set the `h2UpgradePolicy` parameter to `DO_NOT_UPGRADE` in the destination rule. This way, you can establish WebSocket connections by upgrading HTTP/1.1 connections without the need to update the version of Istio.

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  labels:
    provider: asm
  name: websockets-server
spec:
  host: websockets-server
  trafficPolicy:
    connectionPool:
      http:
        h2UpgradePolicy: DO_NOT_UPGRADE
```

If you use Istio 1.12 or later, you can perform the following steps to establish WebSocket connections by using HTTP/2.

1. Create a destination rule.

- i.
- ii.
- iii.
- iv.
- v. On the Create page, select default from the **Namespace** drop-down list and copy the following content to the code editor. Then, click **Create**.

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  labels:
    provider: asm
  name: websockets-server
spec:
  host: websockets-server
  trafficPolicy:
    connectionPool:
      http:
        h2UpgradePolicy: UPGRADE
```

`h2UpgradePolicy`: A value of `UPGRADE` indicates that HTTP/2 is enabled for the WebSocket server.

2. Create an Envoy filter.

By default, WebSocket does not work with the HTTP/2 protocol. However, Envoy supports tunnel transmission for WebSocket data over HTTP/2. This way, all communication can be performed over HTTP/2 in the ASM instance. You can set the `allow_connect` parameter of an Envoy filter to true for the destination workload. After that, HTTP/2 connections are supported by the WebSocket server.

- i. Create an Envoy filter template.
  - a.
  - b.
  - c.
  - d. In the left-side navigation pane, choose **Plugin Center > Market Place**.
  - e. On the **Market Place** page, select the setting **allow\_connect true** for **HTTP protocol upgrade** template and click **Apply this template**.
  - f. In the **Fill template parameters** step, select **SIDECAR\_INBOUND** from the drop-down list and click **Next**.
  - g. Enter the name of the template in the **Name** field and click **OK**.
- ii. Bind the Envoy filter template to a workload.
  - a. In the left-side navigation pane, choose **Plugin Center > EnvoyFilter Template**.
  - b. On the **EnvoyFilter Template** page, find the Envoy filter template that you create in the previous step and click **Edit template** in the **Actions** column.
  - c. Click the **Bind template to workloads** tab and click **Bind EnvoyFilter to Workloads**.
  - d. In the **Bind EnvoyFilter to Workloads** dialog box, set the **Namespace** parameter to **default** and the **Workload Type** parameter to **Deployment**. In the **Not bound** section, click **Bind** in the **Actions** column for **websockets-server**. After that, click **OK**.

After you bind the Envoy filter template to a workload, ASM automatically creates an Envoy filter. The following content describes the YAML code of the Envoy filter:

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: h2-upgrade-wss
  labels:
    asm-system: 'true'
    provider: asm
spec:
  workloadSelector:
    labels:
      app: websockets-server
  configPatches:
    - applyTo: NETWORK_FILTER
      match:
        context: SIDECAR_INBOUND
        proxy:
          proxyVersion: '^1\.*.*'
      listener:
        filterChain:
          filter:
            name: "envoy.filters.network.http_connection_manager"
      patch:
        operation: MERGE
        value:
          typed_config:
            '@type': type.googleapis.com/envoy.extensions.filters.network.http_co
nnection_manager.v3.HttpConnectionManager
            http2_protocol_options:
              allow_connect: true
```

3. Run the following command to access the WebSocket server:

```
python3 -m websockets ws://websockets-server.<namespace>.svc.cluster.local:8080
```

Expected output:

```
Connected to ws://websockets-server.default.svc.cluster.local:8080.
```

If you enter "hello" and "world", the two words are returned as expected.

```
> hello
< hello
> world
< world
Connection closed: 1000 (OK).
```

4. Query the logs of sidecar proxies.

- Query the logs of the sidecar proxy for the WebSocket client.
  - a. On the **Pods** page, click the name of the container of the WebSocket client.



- b. Click the **Logs** tab and select **istio-proxy** from the **Container** drop-down list.

You can see that the logs contain an entry about the WebSocket connection initiated by using HTTP/1.1. This indicates that the WebSocket client sends requests by using HTTP/1.1.

```
{... "authority": "websockets-server.default.svc.cluster.local:8080", "upstream_service_time": null, "protocol": "HTTP/1.1", ...}
```

- o Query the logs of the sidecar proxy for the WebSocket server.
    - a. On the **Pods** page, click the name of the container of the WebSocket server.
    - b. Click the **Logs** tab and select **istio-proxy** from the **Container** drop-down list.

You can see that the logs contain an entry about the WebSocket connection initiated by using HTTP/2. This indicates that the WebSocket server receives requests whose protocols are upgraded to HTTP/2.

```
{... "method": "GET", "upstream_local_address": "127.0.**.**:34477", "protocol": "HTTP/2", ...}
```

## Related information


- [Use an ingress gateway to access a WebSocket service in an ASM instance](#)

# 9. Use delegates to configure virtual services in an ASM instance

When you use a virtual service to define routing rules for a service that contains multiple microservices or routing rules, the virtual service is complex, which brings much maintenance work. To reduce risks brought by changes in routing rules, Alibaba Cloud Service Mesh (ASM) introduces delegates to manage routing rules in a more fine-grained manner. This topic shows you how to use multiple virtual services to define routing rules for a service. The Bookinfo application is used as an example.

## Prerequisites

- An ASM instance is created. For more information, see [Create an ASM instance](#).

 **Note** The ASM instance must be v1.8.6.4-g814070fe-aliyun or later.

- 
- 
- 
- An application is deployed in the ACK cluster that is added to the ASM instance. For more information, see [Deploy an application in an ASM instance](#).

## Context

In an ASM instance, a virtual service is used to define routing rules for a service. The routing rules control the traffic routing of the service. In actual scenarios, you may need to maintain a complex virtual service. Each time you modify the routing rules that are defined by the virtual service, you must modify the virtual service. After the routing rules are modified, the routing rules may conflict with each other, specific routing rules may be redundant, or the routing rules may be coupled with each other. Configuration errors of routing rules may affect the services that are deployed in the cluster on the data plane or even make all services inaccessible.

ASM introduces delegates to extend the configuration of virtual services, which allows you to decouple routing rules that are defined for a service. You can split a virtual service into a primary virtual service and multiple secondary virtual services. The primary virtual service defines the overall routing rules of a service. Each secondary virtual service defines a detailed routing rule of the service. The primary virtual service is maintained by administrators, while the secondary virtual services are maintained by service maintainers. This greatly reduces risks that are brought by changes in routing rules and improves the efficiency of independent service deployment and updates.

## Usage notes

- You can set delegate parameters only in a primary virtual service. If you set delegate parameters both in a primary virtual service and secondary virtual services, the primary virtual service and secondary virtual services do not take effect.
- The HTTPMatchRequest configuration in secondary virtual services must be a subset of that in the primary virtual service. Otherwise, a conflict exists, and the HTTPRoute configuration does not take effect.

- Delegate parameters can be specified only when the route and redirect parameters in the primary virtual service are left empty. The hosts parameter in secondary virtual services must be left empty. The routing rules that are defined by secondary virtual services are combined with the routing rules that are defined by the primary virtual service.

## Step 1: Configure an Istio gateway

- 1.
- 2.
- 3.
- 4.
5. On the Create page, select a namespace from the **Namespace** drop-down list, copy the following content to the code editor, and then click . In this example, the default namespace is used.

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

```

Set the `number` parameter to `80`. This setting specifies that the Bookinfo application can receive inbound or outbound HTTP connections by using port 80.

## Step 2: Configure a primary virtual service

- 1.
- 2.
- 3.
- 4.
5. On the Create page, select a namespace from the **Namespace** drop-down list, copy the following content to the code editor, and then click . In this example, the default namespace is used.

The following content can be used to create a vs-1 delegate and a vs-2 delegate. The vs-1 delegate specifies that only request URLs that contain `/log` can be used to access the Bookinfo application. The vs-2 delegate specifies that only request URLs that contain `/` can be used to access the Bookinfo application.

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: bookinfo
  namespace: default
spec:
  gateways:
  - bookinfo-gateway
  hosts:
  - '*'
  http:
  - delegate:
      name: vs-1
      namespace: ns1
      match:
      - uri:
          prefix: /log
  - delegate:
      name: vs-2
      namespace: ns1
      match:
      - uri:
          prefix: /
```

Description of the delegate parameter:

- name: the name of delegate.
- namespace: the namespace of delegate.

### Step 3: Configure secondary virtual services

- 1.
- 2.
- 3.
- 4.
5. On the Create page, select a namespace from the **Namespace** drop-down list, copy the following content to the code editor, and then click . In this example, the ns1 namespace is used.

```

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-1
  namespace: ns1
spec:
  http:
    - match:
        - uri:
            exact: /login
        - uri:
            exact: /logout
      route:
        - destination:
            host: productpage.default.svc.cluster.local
            port:
              number: 9080

```

- o **metadata**: associates the secondary virtual service with a delegate in the primary virtual service. The settings of metadata must be the same as those of a delegate in the primary virtual service. In this substep, the settings of metadata are the same as those of the vs-1 delegate. This means that the secondary virtual service is associated with the vs-1 delegate.
  - o **match**: specifies filter conditions for HTTP requests. In this substep, the `uri` field is set to `exact: /login` and `exact: /logout`. This setting specifies that you can log on to or log off from the Bookinfo application.
6. Repeat Substep 5. On the Create page, select a namespace from the **Namespace** drop-down list, copy the following content to the code editor, and then click .

```

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-2
  namespace: ns1
spec:
  http:
    - match:
        - uri:
            exact: /productpage
        - uri:
            prefix: /static
        - uri:
            prefix: /api/v1/products
      route:
        - destination:
            host: productpage.default.svc.cluster.local
            port:
              number: 9080

```

- o **metadata**: associates the secondary virtual service with a delegate in the primary virtual service. The settings of metadata must be the same as those of a delegate in the primary virtual service. In this substep, the settings of metadata are the same as those of the vs-2 delegate. This means that the secondary virtual service is associated with the vs-2 delegate.

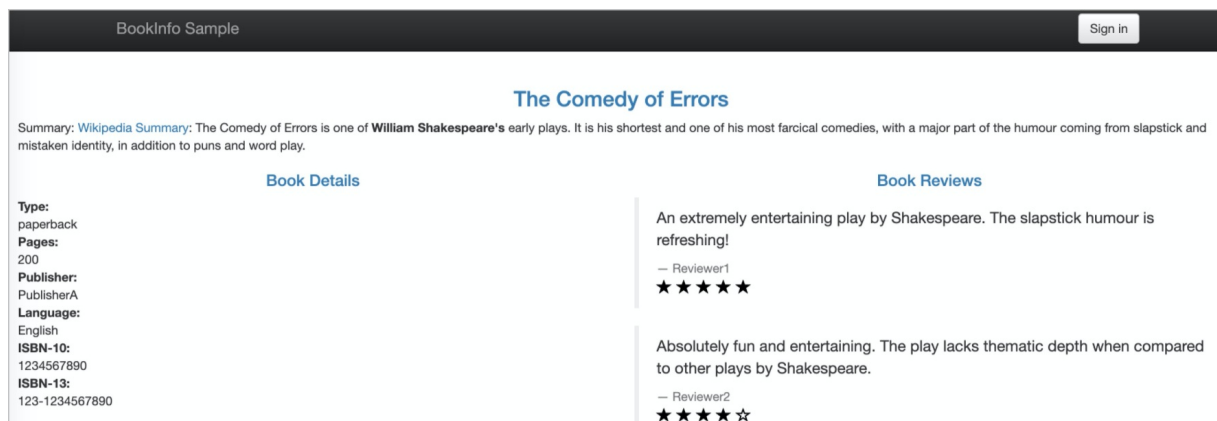
- **match**: specifies filter conditions for HTTP requests. In this substep, the setting specifies that only request URLs that contain `/productpage` or start with `/static` or `/api/v1/products` can be used to access the Bookinfo application.

## Step 4: Query the IP address of the ingress gateway

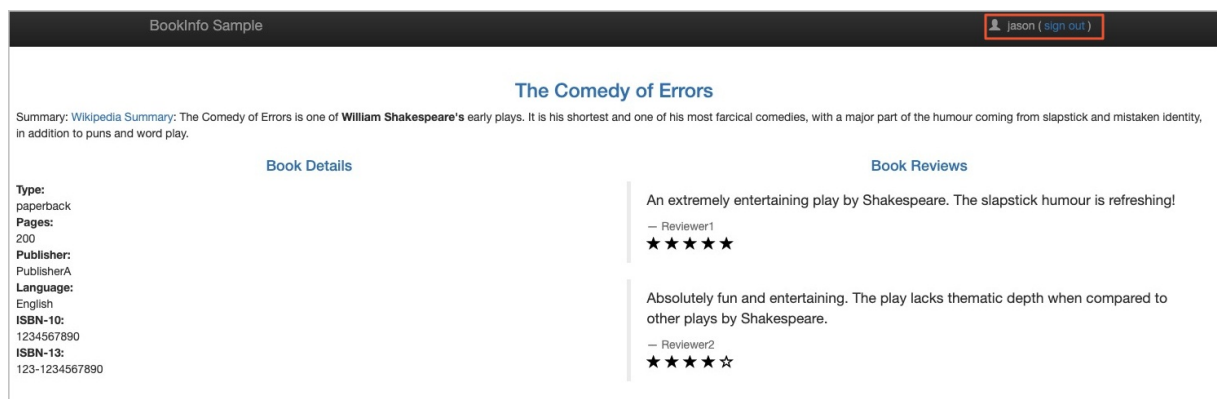
- 1.
- 2.
- 3.
- 4.
5. At the top of the **Services** page, select `istio-system` from the **Namespace** drop-down list. Find the ingress gateway that is named `istio-ingressgateway` and view the IP address whose port is 80 in the **External Endpoint** column.

## Verify the result

Enter `http://<IP address of the ingress gateway>/productpage` in the address bar of a browser. The following page appears. This means that the Bookinfo application is accessed by using the request URL that contains `/productpage`. The result indicates that the secondary virtual service that is associated with the `vs-2` delegate takes effect.



Click **Sign in** in the upper-right corner. In the dialog box that appears, enter the username and password to log on to the Bookinfo application. The following page appears. This means that you have logged on to the Bookinfo application. The result indicates that the secondary virtual service that is associated with the `vs-1` delegate takes effect.



# 10. Use the local throttling feature of ASM

In scenarios such as flash sales, the traffic may instantaneously reach a peak that exceeds the maximum load supported by your system. As a result, a large number of calls are waiting to be processed, and the system stops responding. Alibaba Cloud Service Mesh (ASM) provides the local throttling feature that you can use to throttle traffic for gateways and services. This way, you can protect your system from being overloaded. This topic describes how to use the local throttling feature of ASM.

## Prerequisites

- An ASM instance is created. The ASM instance meets the following requirements:
  - If the ASM instance is of a commercial edition, the version of the ASM instance must be V1.11.5.30 or later. For more information about how to update an ASM instance, see [升级ASM实例](#).
  - If the ASM instance is of Standard Edition, the version of the ASM instance must be V1.9 or later. In addition, you can use only the native rate limiting feature of Istio to implement local throttling for the ASM instance. The reference document varies with the Istio version. For more information about how to configure local throttling for the latest Istio version, see [Enabling Rate Limits using Envoy](#).
- 
- 
- The Bookinfo and NGINX services are created. For more information, see [Deploy an application in an ASM instance](#). In this topic, the Bookinfo service is deployed in the default namespace, and the NGINX service is deployed in the foo namespace.
- A gateway is created for the ASM instance by using the following configurations. For more information, see [Deploy an ingress gateway service](#). In this topic, the ASM gateway is deployed in the istio-system namespace.

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: bookinfo-gateway
  namespace: default
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - bf2.example.com
    port:
      name: http
      number: 80
      protocol: http
```

- A virtual service is created by using the following configurations. For more information, see [Manage virtual services](#).

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: bookinfo
  namespace: default
spec:
  gateways:
  - bookinfo-gateway
  hosts:
  - bf2.example.com
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        prefix: /static
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    name: productpage-route-name1
    route:
    - destination:
        host: productpage
        port:
          number: 9080
  - match:
    - uri:
        prefix: /nginx
    name: nginx-route-name1
    rewrite:
      uri: /
    route:
    - destination:
        host: nginx.foo.svc.cluster.local
        port:
          number: 80
```

- The traffic generation tool hey is installed. For more information, visit [hey](#) at [GitHub](#).

## Declarative configurations of an ASMLocalRateLimiter

ASM allows you to declaratively define the configurations of an ASMLocalRateLimiter by using a custom resource definition (CRD). The following table describes the fields in the Spec section.

Field	Type	Description	Required
-------	------	-------------	----------



Field	Type	Description	Required
workloadSelector	map<string, string>	The one or more labels that specify a set of pods or VMs on which the throttling rule takes effect. The scope of label search is restricted to the configuration namespace in which the resources reside. For more information, see <a href="#">Workload Selector</a> .	Yes
isGateway	bool	Specifies whether the throttling rule takes effect on a gateway. Default value: <code>false</code> .	No
configs	<a href="#">LocalRateLimiterConfig</a> []	The local throttling rule. You can configure multiple local throttling rules.	Yes

#### Attribute fields of LocalRateLimiterConfig

Field	Type	Description	Required
name	string	The name of the throttling rule.	No
match	<a href="#">RateLimitMatch</a>	The match conditions.	No
limit	LimitConfig	The threshold configurations.	No

#### Attribute fields of RateLimitMatch

Field	Type	Description	Required
vhost	<a href="#">VirtualHostMatch</a>	The match conditions for the vhost.	No

#### Attribute fields of LimitConfig

Field	Type	Description	Required
fill_interval	Duration	The interval of issuing tokens. Examples: <code>seconds: 1</code> and <code>nanos: 1000</code> . <code>nanos</code> indicates nanoseconds.	No

Field	Type	Description	Required
quota	int	The number of tokens. The value must be an integer. Example: 1000.	No

#### Attribute fields of VirtualHostMatch

Field	Type	Description	Required
name	string	The name of the matched vhost.	No
port	int	The matched request port.	No
route	RouteMatch	The name of the route corresponding to the matched request port.	No

#### Attribute fields RouteMatch

Field	Type	Description	Required
name_match	string	The name of the matched route. This parameter specifies a single route in a virtual service.	No
header_match	HeaderMatcher[]	The header for matching service requests. You can configure multiple headers.	No

#### Attribute fields of HeaderMatcher

Field	Type	Description	Required
name	string	The name of the header.	No
regex_match	string	The value that is used for regular expression match.	No
exact_match	string	The value that is used for exact match.	No
prefix_match	string	The value that is used for prefix match.	No

Field	Type	Description	Required
suffix_match	string	The value that is used for suffix match.	No
present_match	bool	Specifies whether to check only the availability of the specified header. If you set this parameter to <code>true</code> , the system only checks whether the header exists regardless of the value of the header.	No
invert_match	bool	Specifies whether to invert the result of regular expression match. Default value: <code>false</code> . If you set this parameter to <code>true</code> , the result of regular expression match is inverted.	No

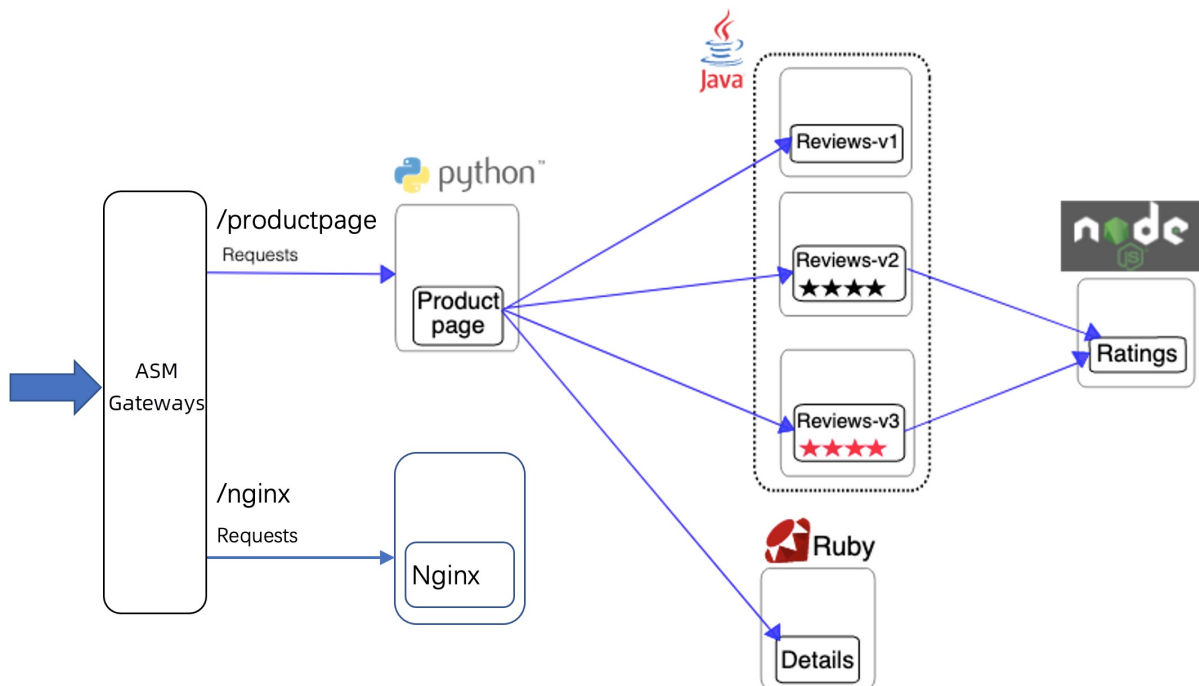
## Application scope

The local throttling feature of ASM is applicable to ASM gateways and services for which sidecar proxies are injected.

 **Note** You can download the [configuration files](#) that are used in the following sample scenarios.

## Sample scenario description

In the sample scenarios, the Bookinfo and NGINX services are used to describe how to throttle traffic for gateways and services. The NGINX service is separately deployed in the foo namespace to verify the scope in which throttling takes effect.



## Throttle traffic for a gateway

You can throttle traffic for a gateway that serves as a traffic ingress. This way, you can prevent downstream services from being overloaded.

Use `kubectl` to connect to the ASM instance. For more information, see [Use kubectl to connect to an ASM instance](#). Create an `ASMLocalRateLimiter` by using the following configurations:

**Notice** In the configurations, the setting of the `limit.quota` field applies only to a single gateway instance. If the gateway has  $n$  instances, the threshold for the backend service corresponding to the `test1` route is  $n \times \text{quota}$ . If the number of backend service instances is changed, you must change the threshold accordingly.

```

apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMLocalRateLimiter
metadata:
  name: for-api-test
  namespace: default
spec:
  workloadSelector:
    labels:
      app: istio-ingressgateway
  isGateway: true
  configs:
    - match:
        vhost:
          name: "www.example1.com"  ## If multiple vhosts are configured for the gateway,
          port: 80                  enter the name of the last vhost.
          route:
            name_match: "test1"    ## The name of the route that is configured for the virtu
            al service. If the virtual service does not have the specified route, the throttling does n
            ot take effect.
        limit:
          fill_interval:
            seconds: 1
          quota: 10
    - match:
        vhost:
          name: "www.example2.com"
          port: 80
          route:
            name_match: "test1"
        limit:
          fill_interval:
            seconds: 1
          quota: 100

```

- **workloadSelector:** the one or more labels that specify a set of pods or VMs on which the throttling rule takes effect. In this example, the *app: istio-ingressgateway* label is specified so that the ASMLocalRateLimiter takes effect on a gateway.
- **isGateway:** specifies whether the throttling rule takes effect on a gateway. In this example, the value is set to *true*.
- **seconds in fill\_interval:** the interval of issuing tokens.
- **quota:** the number of tokens that are issued.

In this topic, the seconds field is set to *1*, and the quota field is set to *100*. This indicates that 100 tokens are issued within 1 second. This way, the gateway can process up to 100 requests within 1 second.

## Sample scenarios of gateway traffic throttling

### Scenario 1: Configure a throttling rule for a single interface

Configure throttling for the `productpage-route-name1` route of the vhost `bf2.example.com:80`. This way, the access is throttled for the `/productpage`, `/static`, `/login`, and `/logout` interfaces that use the `productpage-route-name1` route.

1. Use `kubectl` to connect to the ASM instance. For more information, see [Use kubectl to connect to an ASM instance](#).
2. Create an `ASMLocalRateLimiter`.
  - i. Create the `asmlocalratelimiter-test-gw.yaml` file that contains the following content:

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMLocalRateLimiter
metadata:
  name: ingressgateway
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: istio-ingressgateway
  isGateway: true
  configs:
  - limit:
      fill_interval:
        seconds: 1
      quota: 10
  match:
    vhost:
      name: bf2.example.com
      port: 80
      route:
        name_match: productpage-route-name1 ## The name must be the same as that
        configured in the route configuration of the virtual service.
```

- ii. Run the following command to create an `ASMLocalRateLimiter`:

```
kubectl apply -f asmlocalratelimiter-test-gw.yaml
```

3. Run the following commands in `hey` to generate continuous pressure traffic:

```
hey -host bf2.example.com -c 10 -n 100000 http://<IP address of the ASM gateway>/productpage
```

```
hey -host bf2.example.com -c 10 -n 100000 http://<IP address of the ASM gateway>/nginx
```

4. Run the following command to access the `/productpage` interface of the Bookinfo service:

```
curl -H 'host: bf2.example.com' http://<IP address of the ASM gateway>/productpage -v
```

Expected output:

```
< HTTP/1.1 429 Too Many Requests
< Content-Length: 18
< Content-Type: text/plain
< Date: Thu, 13 Jan 2022 03:03:09 GMT
< Server: istio-envoy
<
local_rate_limited
```

The access to the Bookinfo service is throttled.

5. Run the following command to access the `/nginx` interface of the Bookinfo service:

```
curl -H 'host: bf2.example.com' http://${ASM_GATEWAY_IP}/nginx -v
```

The 429 error code is not found in the returned result. This indicates that the access is not throttled.

## Scenario 2: Configure a global throttling rule for a vhost and port on a gateway

Configure throttling for the vhost `bf2.example.com:80`. This way, the access is throttled for the `/productpage`, `/static`, `/login`, `/logout`, and `/nginx` interfaces of the Bookinfo service.

1. Use `kubectl` to connect to the ASM instance. For more information, see [Use kubectl to connect to an ASM instance](#).
2. Create an `ASMLocalRateLimiter`.
  - i. Create the `asmlocalratelimiter-test-gw-global.yaml` file that contains the following content:

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMLocalRateLimiter
metadata:
  name: ingressgateway
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: istio-ingressgateway
  isGateway: true
  configs:
    - match:
        vhost:
          name: "bf2.example.com"
          port: 80
      limit:
        fill_interval:
          seconds: 1
        quota: 10
```

- ii. Run the following command to create an `ASMLocalRateLimiter`:

```
kubectl apply -f asmlocalratelimiter-test-gw-global.yaml
```

3. Run the following command in `hey` to generate continuous pressure traffic:

```
hey -host bf2.example.com -c 10 -n 100000 http://${ASM_GATEWAY_IP}/nginx
```

4. Run the following command to access the `/nginx` interface of the Bookinfo service:

```
curl -H 'host: bf2.example.com' http://${ASM_GATEWAY_IP}/nginx -v
```

The message `HTTP/1.1 429 Too Many Requests` is returned, indicating that the access to the `/nginx` interface of the Bookinfo service is throttled.

### Scenario 3: Delete the throttling configurations to stop throttling the access

1. Use `kubectl` to connect to the ASM instance. For more information, see [Use kubectl to connect to an ASM instance](#).
2. Run the following commands to delete the configuration files for throttling:

```
kubectl delete -f asmlocalratelimiter-test-gw.yaml
```

```
kubectl delete -f asmlocalratelimiter-test-gw-global.yaml
```

3. Run the following command to access the `/nginx` interface of the Bookinfo service:


```
curl -H 'host: bf2.example.com' http://${ASM_GATEWAY_IP}/nginx -v
```

The 429 error code is not found in the returned result. This indicates that the access is not throttled.

## Throttle traffic for a service

You can throttle traffic for a service to limit the number of requests that can be processed by the service within a specific period of time.

Use `kubectl` to connect to the ASM instance. For more information, see [Use kubectl to connect to an ASM instance](#). Create an `ASMLocalRateLimiter` by using the following configurations:

 **Note** The threshold that you specify applies only to a single service instance. If a service has  $n$  instances, the overall threshold for the service is  $n \times \text{quota\_per\_instance}$ .

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMLocalRateLimiter
metadata:
  name: reviews-v3-local-ratelimiter
  namespace: default
spec:
  workloadSelector:
    labels:
      app: reviews
      version: v3
  ....
```

If a service has multiple versions, you can use a label in `workloadSelector` to specify the deployment on which the throttling takes effect. For example, the preceding configurations indicate that the throttling takes effect only on the v3 version of the reviews service.

## Sample scenarios of service traffic throttling



**Note** Sidecar proxies must be injected for the services for which you want to configure throttling. Before you configure throttling for services, delete the throttling configurations that you configure for gateways to prevent the throttling of services from being affected.

### Scenario 1: Configure a throttling rule for the reviews service

1. Use `kubectl` to connect to the ASM instance. For more information, see [Use kubectl to connect to an ASM instance](#).
2. Create an `ASMLocalRateLimiter`.
  - i. Create the `asmlocalratelimiter-test-reviews.yaml` file that contains the following content:

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMLocalRateLimiter
metadata:
  name: reviews
  namespace: default
spec:
  workloadSelector:
    labels:
      app: reviews
  configs:
    - match:
        vhost:
          name: "*"
          port: 9080
          route:
            header_match:
              - name: ":path"
                prefix_match: "/"
        limit:
          fill_interval:
            seconds: 1
          quota: 10
```

The preceding configurations take effect on port 9080 of the reviews service. The header match conditions specified by the `header_match` field are used to match service requests in which the path starts with a forward slash (/).

- ii. Run the following command to create an `ASMLocalRateLimiter`:

```
kubectl apply -f asmlocalratelimiter-test-reviews.yaml
```

3. Run the following command in `hey` to generate continuous pressure traffic:

```
hey -host bf2.example.com -c 10 -n 100000 http://<IP address of the ASM gateway>/productpage
```

4. Run the following command to view the sidecar logs for the productpage service:

```
kubectl logs -f productpage-v1-b84f8bfdd-wgxlc -c istio-proxy
```

Expected output:

```
[2022-01-14T07:56:13.086Z] "GET /reviews/0 HTTP/1.1" 429 - via_upstream - "-" 0 18 1 1
 "-" "hey/0.0.1" "9295da56-9a6b-9476-b662-1cbd61a82898" "reviews:9080" "10.180.0.190:9080" outbound9080v1reviews.default.svc.cluster.local 10.180.0.196:36702 192.168.195.113:9080 10.180.0.196:33522 - -
[2022-01-14T07:56:13.091Z] "GET /reviews/0 HTTP/1.1" 429 - via_upstream - "-" 0 18 0 0
 "-" "hey/0.0.1" "9295da56-9a6b-9476-b662-1cbd61a82898" "reviews:9080" "10.180.0.190:9080" outbound9080v1reviews.default.svc.cluster.local 10.180.0.196:36702 192.168.195.113:9080 10.180.0.196:33528 - -
[2022-01-14T07:56:13.051Z] "GET /details/0 HTTP/1.1" 200 - via_upstream - "-" 0 178 41 1
 "-" "hey/0.0.1" "061d3542-52a7-9511-b217-7fdf9ee9a1dd" "details:9080" "10.180.0.160:9080" outbound9080details.default.svc.cluster.local 10.180.0.196:58724 192.168.127.75:9080 10.180.0.196:57754 - default
[2022-01-14T07:56:13.095Z] "GET /reviews/0 HTTP/1.1" 429 - via_upstream - "-" 0 18 0 0
 "-" "hey/0.0.1" "061d3542-52a7-9511-b217-7fdf9ee9a1dd" "reviews:9080" "10.180.0.190:9080" outbound9080v1reviews.default.svc.cluster.local 10.180.0.196:36702 192.168.195.113:9080 10.180.0.196:33534 - -
```

The error code 429 is returned for requests to `reviews:9080`. This indicates that the access is throttled.

## Scenario 2: Configure a throttling rule for the reviews service of the v3 version

1. Use `kubectl` to connect to the ASM instance. For more information, see [Use kubectl to connect to an ASM instance](#).
2. Create an `ASMLocalRateLimiter`.

- i. Create the `asmlocalratelimiter-test-reviews-only-v3.yaml` file that contains the following content:

```

apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMLocalRateLimiter
metadata:
  name: reviews
  namespace: default
spec:
  workloadSelector:
    labels:
      app: reviews
      version: v3
  configs:
    - match:
        vhost:
          name: "*"
          port: 9080
          route:
            header_match:
              - name: ":path"
                prefix_match: "/"
        limit:
          fill_interval:
            seconds: 1
          quota: 10

```

The preceding configurations take effect on port 9080 of the reviews service of the v3 version. The header match conditions specified by the `header_match` field are used to match service requests in which the path starts with a forward slash (/).

- ii. Run the following command to create an `ASMLocalRateLimiter`:

```
kubectl apply -f asmlocalratelimiter-test-reviews-only-v3.yaml
```

3. Run the following command in `hey` to generate continuous pressure traffic:

```
hey -host bf2.example.com -c 10 -n 100000 http://<IP address of the ASM gateway>/productpage
```

4. Run the following command to view the sidecar logs for the pods of the reviews service of the v1, v2, and v3 versions:

```
kubectl logs -f ${your-reviews-pod-name} -c istio-proxy
```

The error code 429 can be found only in the access logs for requests to the reviews service of the v3 version. The status code 200 is returned for all requests to the reviews service of the v1 and v2 versions. This indicates that only the access to the reviews service of the v3 version is throttled. The access to the reviews service of the v1 and v2 versions is not throttled.

# 11.Enable graceful shutdown for the SLB instance of an ASM gateway to prevent traffic loss

If you perform a scale-in or rolling restart operation on an Alibaba Cloud Service Mesh (ASM) gateway, a small amount of traffic is lost because the number of gateway pods is reduced. To resolve this issue, you can enable graceful shutdown for the Server Load Balancer (SLB) instance of the ASM gateway. This way, traffic can continue to be transferred by using the SLB instance within the specified period of time even if the number of gateway pods is reduced. This ensures that no traffic is lost. This topic describes how to enable graceful shutdown for the SLB instance of an ASM gateway.

## Prerequisites

- An ASM instance of Enterprise Edition or Ultimate Edition is created. For more information, see [Create an ASM instance](#).
- 

## Step 1: Enable graceful shutdown for an SLB instance


When you create an ASM gateway, you can enable graceful shutdown for the SLB instance that you create for the ASM gateway. You can also enable graceful shutdown for the SLB instance of an existing ASM gateway.

### Enable graceful shutdown for an SLB instance when you create an ASM gateway

- 1.
- 2.
- 3.
- 4.
5. On the **Create** page, select a cluster, select **Internet Access** as **SLB Instance Type**, select an SLB instance type under **Create SLB Instance**, and then set **Gateway instances** to **10**. Use the default values for other parameters.
6. Click **Advanced Options**, select **SLB graceful offline**, specify a connection timeout for the SLB instance, and then click **Create**.

### Enable graceful shutdown for the SLB instance of an existing ASM gateway

- 1.
- 2.
- 3.
- 4.
5. On the **ASM Gateways** page, click the name of the ASM gateway of which you want to enable graceful shutdown for the SLB instance.

6. In the **Advanced Options** section of the **Gateway Details** page, click the  icon next to **SLB graceful offline**, select **SLB graceful offline**, specify a connection timeout for the SLB instance of the ASM gateway, and then click **OK**.

## Step 2: Deploy a sample application

- 1.
2. Create an *httpbin.yaml* file that contains the following content:

```
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
```

3. Run the following command to deploy the httpbin application:

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

## Step 3: Create a virtual service and an Istio gateway

1. Create a virtual service.

- i.
- ii.
- iii.
- iv.
- v. On the **Create** page, set the **Namespace** and **Template** parameters, replace the content in the YAML code editor with the following content, and then click **Create**.

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: httpbin
  namespace: default
spec:
  gateways:
    - httpbin-gateway
  hosts:
    - '*'
  http:
    - route:
        - destination:
            host: httpbin
            port:
                number: 8000
```

2. Create an Istio gateway.

- i.
- ii.
- iii.
- iv.

- v. On the **Create** page, set the **Namespace** and **Template** parameters, replace the content in the YAML code editor with the following content, and then click **Create**.

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: httpbin-gateway
  namespace: default
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - '*'
    port:
      name: http
      number: 80
      protocol: HTTP
```

3. Verify that the route configuration is successful.
  - i. Obtain the IP address of the ASM gateway. For more information, see [Deploy an ingress gateway service](#).
  - ii. In the address bar of your browser, enter *http://<IP address of the ASM gateway>*.

httpbin

If the page shown in the preceding figure appears, the route configuration is successful.

## Step 4: Verify traffic loss before and after you enable graceful shutdown for the SLB instance of the ASM gateway

1. Download and install the lightweight stress testing tool **hey** of a version that is suitable for your operating system. For more information, visit [hey at GitHub](#).

The following **hey** versions are supported for different operating systems:

- o Linux 64-bit: [hey-release.s3.us-east-2.amazonaws.com/hey\\_linux\\_amd64](#)
- o macOS 64-bit: [hey-release.s3.us-east-2.amazonaws.com/hey\\_darwin\\_amd64](#)
- o Windows 64-bit: [hey-release.s3.us-east-2.amazonaws.com/hey\\_windows\\_amd64](#)

2. Scale in the ASM gateway.
  - i.
  - ii.
  - iii.
  - iv.
  - v. On the **ASM Gateways** page, find the ASM gateway that you want to scale in and click **YAML** in the **Actions** column.
  - vi. In the **Edit** panel, set the **replicaCount** parameter to **1** and click **OK**.

3. Check whether traffic loss occurs after you enable graceful shutdown for the SLB instance of the ASM gateway.

Run the following command to send requests to access the httpbin application. Set the number of requests to run concurrently to 200 and the total number of requests to run to 50000.

```
hey -c 200 -n 50000 -disable-keepalive http://{IP address of the ASM gateway}/
```

Expected output:

```
.....  
Status code distribution:  
[200] 50000 responses
```

The preceding output shows that the status code 200 is returned for all 50,000 access requests. This indicates that all 50,000 access requests are successful and no traffic is lost.

Disable graceful shutdown for the SLB instance of the ASM gateway, scale in the ASM gateway, and then run the preceding hey command. Set the number of requests to run concurrently to 200 and the total number of requests to run to 50000.

```
Status code distribution:  
[200] 49747 responses  
Error distribution:  
[253] Get "http://47.55.2xx.xx": dial tcp 47.55.2xx.xx:80: connect: connection refused
```

The preceding output shows that the status code 200 is returned for 49,747 access requests out of the 50,000 access requests. This indicates that only 49,747 access requests are successful and a small amount of traffic is lost.



# 12. Traffic labeling and label-based routing

Alibaba Cloud Service Mesh (ASM) allows you to configure traffic labels by using TrafficLabel Custom Resource Definitions (CRDs) and route traffic to different workloads based on the traffic labels. This topic describes the traffic labeling and label-based routing features.

## Context

Traffic labeling is also called traffic coloring. Traffic labeling is to label the requests that have specific traffic characteristics. Istio transparently hijacks application traffic. You can use a sidecar to label application traffic based on Istio.

ASM Commercial Edition (Professional Edition) provides a new TrafficLabel CRD. You can use this CRD to define traffic labeling logic and label traffic for a specific namespace, workload, or API operation.

## Usage notes

Only ASM Commercial Edition (Professional Edition) V1.10.5.40 and later support the traffic labeling and label-based routing features.

## Label traffic


### Method 1: Label traffic by namespace

You can use this method to label traffic of all applications in a specific namespace.

1. Use `kubectl` to connect to an ASM instance.
2. Create an `example1.yaml` file that contains the following code:

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: TrafficLabel
metadata:
  name: example1
  namespace: default
spec:
  rules:
  - labels:
    - name: userdefinelabel1
      valueFrom:
      - $getContext(x-request-id)
      - $localLabel
    attachTo:
    - opentracing
    # The protocols that take effect. If you leave the protocols parameter empty, no protocol takes effect. If you set the protocols parameter to an asterisk (*), all protocols take effect.
    protocols: "*"
    hosts: # The services that take effect.
    - "*"

```

Parameter	Description
namespace	The namespace in which the TrafficLabel CRD takes effect.
name under the labels parameter	The name of the traffic label.
valueFrom under the labels parameter	<p>The value of the traffic label. Valid values:</p> <div><p> <b>Note</b> The values of the valueFrom parameter are sequenced by priority. For example, the sequence in the preceding code indicates that the <code>getContext(x-request-id)</code> variable is preferentially used to obtain traffic label values. The <code>localLabel</code> variable is used only if the <code>getContext(x-request-id)</code> variable fails to obtain traffic label values.</p><ul style="list-style-type: none"><li>◦ <code>getContext(x-request-id)</code>: obtains traffic label values based on the traffic context.</li><li>◦ <code>localLabel</code>: obtains traffic label values based on the <code>ASM_TRAFFIC_TAG</code> label that is added to the pod involved in the workload.</li></ul></div>
attachTo	Set the value to <code>opentracing</code> . In this case, if the traffic label applies to the HTTP or gRPC protocol, the traffic label is added to the request header.

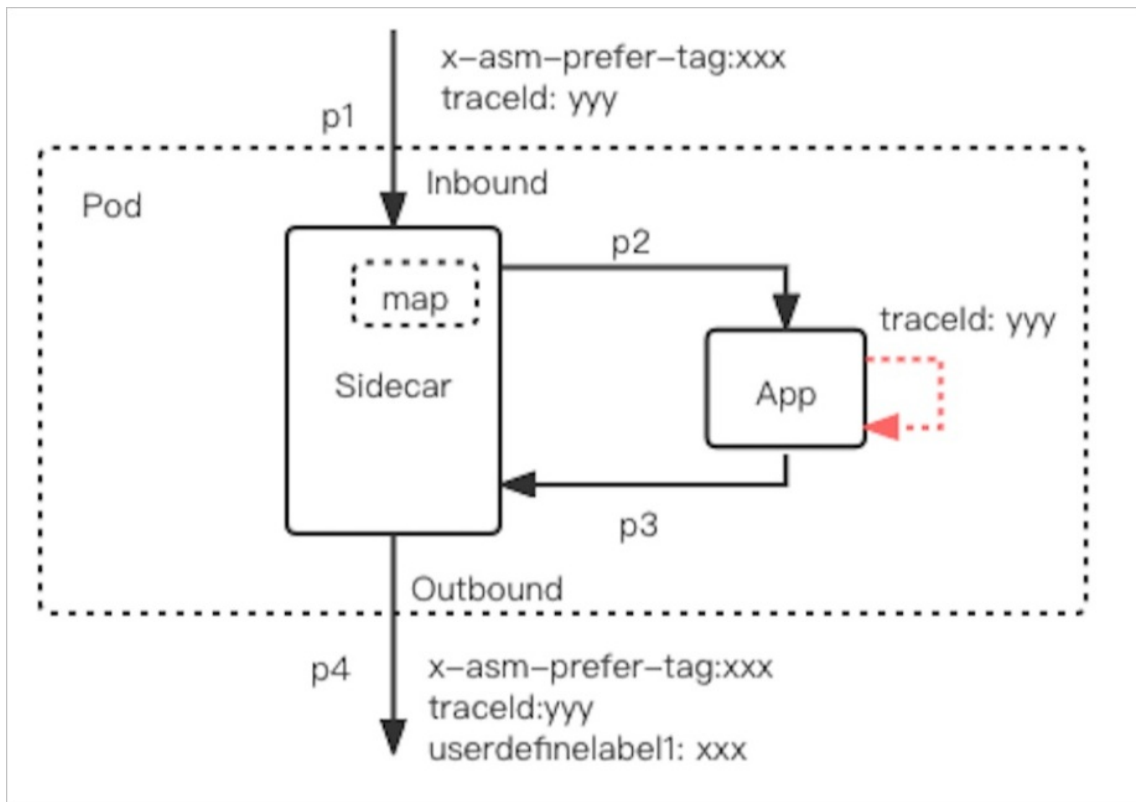
The following content describes the `getContext(xxx)` and `localLabel` variables:

- `getContext(x-request-id)`

`getContext(x-request-id)` is a function that serves as a variable. The variable is used to obtain the traffic label based on the traffic context. The `x-request-id` parameter specifies the trace ID. Different tracing systems use different trace IDs.

 **Note** The `getContext` variable takes effect only for a sidecar.

A sidecar involves inbound traffic and outbound traffic. Traffic labeling is to label the outbound traffic.



By default, a sidecar of ASM Commercial Edition (Professional Edition) attempts to obtain a traffic label from the inbound traffic by using the `x-asm-prefer-tag` header and records the traffic label to the context in the format of `map<traceId, tag>`. The trace ID is used as the key. If a container initiates an outbound request, the sidecar searches for the traffic label in the map based on the trace ID. If the traffic label is found, the sidecar transfers the traffic label together with the automatically transferred `x-asm-prefer-tag` header and the traffic label name `userdefinelabel1` that is defined by the TrafficLabel CRD.

**Note** Different tracing systems use different trace IDs. For more information, see [Tracing](#).

If you do not set the parameter of the `getContext` function, the sidecar uses the `x-request-id` as the trace ID by default. In this case, your application must propagate the trace ID by using a header. If no header propagation logic exists, the trace ID of an outbound request cannot be obtained. Then, the sidecar may fail to obtain the corresponding traffic label from `map<traceId, tag>`.

If your application is connected to a tracing system, you can use the SDKs provided by the tracing system to propagate the context in the code by using headers. For example, you can connect your application to [Application Real-Time Monitoring Service \(ARMS\) for tracing](#). If your application is connected to Prometheus Service (Prometheus) in ARMS, you need only to change the `getContext(x-request-id)` variable under `valueFrom` in the TrafficLabel CRD to `getContext(x-b3-traceid)`. The `x-b3-traceid` parameter specifies a trace header in the Zipkin system.

- `localLabel`

The `localLabel` variable is used to obtain traffic label values based on the `ASM_TRAFFIC_TAG` label that is added to the pod involved in the sidecar and label the outbound traffic. In the following YAML file of a Deployment, the value of the `ASM_TRAFFIC_TAG` label is `test`.

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

3. Run the following command to make the TrafficLabel CRD take effect:

```
kubectl apply -f example1.yaml
```

## Method 2: Label traffic by workload


You can use this method to label traffic for a specific application in a namespace.

1. Use `kubectl` to connect to an ASM instance.
2. Create an `example2.yaml` file that contains the following code:

```

apiVersion: istio.alibabacloud.com/v1beta1
kind: TrafficLabel
metadata:
  name: example2
  namespace: workshop
spec:
  workloadSelector:
    labels:
      app: test
  rules:
    - labels:
        - name: userdefinelabel1
          valueFrom:
            - $localLabel
            - $getContext(x-request-id)
      attachTo:
        - opentracing
      protocols: "*"
  hosts:
    - "*"

```

Parameter	Description
namespace	The namespace in which the TrafficLabel CRD takes effect.
name under the labels parameter	The name of the traffic label.
valueFrom under the labels parameter	<p>The value of the traffic label. Valid values:</p> <div data-bbox="842 1218 1385 1525" data-label="Text"> <p> <b>Note</b> The values of the valueFrom parameter are sequenced by priority. For example, the sequence in the preceding code indicates that the <i>getContext(x-request-id)</i> variable is preferentially used to obtain traffic label values. The <i>localLabel</i> variable is used only if the <i>getContext(x-request-id)</i> variable fails to obtain traffic label values.</p> </div> <ul style="list-style-type: none"> <li>◦ <i>getContext(x-request-id)</i>: obtains traffic label values based on the traffic context.</li> <li>◦ <i>localLabel</i>: obtains traffic label values based on the <code>ASM_TRAFFIC_TAG</code> label added to the pod involved in the workload.</li> </ul>

Parameter	Description
app under the workloadSelector parameter	The label used to match a workload. For example, the parameter value in the preceding code indicates that the TrafficLabel CRD applies only to a workload with the test label.
attachTo	Set the value to <i>opentracing</i> . In this case, if the traffic label applies to the HTTP or gRPC protocol, the traffic label is added to the request header.

3. Run the following command to make the TrafficLabel CRD take effect:

```
kubectl apply -f example2.yaml
```

### Method 3: Label traffic by API operation


You can use this method to label traffic for a specific API operation of an application in a namespace.

1. Use `kubectl` to connect to an ASM instance.
2. Create an `example3.yaml` file that contains the following code:

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: TrafficLabel
metadata:
  name: example3
  namespace: workshop
spec:
  workloadSelector:
    labels:
      app: test
  rules:
    - match:
        - headers:
            end-user: "jason"
          uri:
            prefix: "/test"
          labels:
            - name: userdefinelabel1
              valueFrom:
                - $getContext(x-b3-traceid)
      attachTo:
        - opentracing
      protocols: "*"
  hosts:
    - "*"

```

Parameter	Description
namespace	The namespace in which the TrafficLabel CRD takes effect.

Parameter	Description
app under the workloadSelector parameter	The label used to match a workload. For example, the parameter value in the preceding code indicates that the TrafficLabel CRD applies only to a workload with the <code>test</code> label.
headers under the match parameter	The request header. In this example, the end-user parameter under headers is set to <code>json</code> .
prefix under the match parameter	The prefix of the uniform resource identifier (URI) for the request. You can use the prefix to specify an API request. In this example, the prefix of the request URI is set to <code>/test</code> .
name under the labels parameter	The name of the traffic label.
valueFrom under the labels parameter	<p>The value of the traffic label. Valid values:</p> <div> <p> <b>Note</b> The values of the valueFrom parameter are sequenced by priority. For example, the sequence in the preceding code indicates that the <code>getContext(x-request-id)</code> variable is preferentially used to obtain traffic label values. The <code>localLabel</code> variable is used only if the <code>getContext(x-request-id)</code> variable fails to obtain traffic label values.</p> <ul style="list-style-type: none"> <li>◦ <code>getContext(x-request-id)</code>: obtains traffic label values based on the traffic context.</li> <li>◦ <code>localLabel</code>: obtains traffic label values based on the <code>ASM_TRAFFIC_TAG</code> label added to the pod involved in the workload.</li> </ul> </div>
attachTo	Set the value to <code>opentracing</code> . In this case, if the traffic label applies to the HTTP or gRPC protocol, the traffic label is added to the request header.

3. Run the following command to make the TrafficLabel CRD take effect:

```
kubectl apply -f example3.yaml
```

## Route traffic based on traffic labels

After you define the traffic labels whose names are `userdefinelabel1` and whose values are `test1`, `test2`, and `test3` by using the TrafficLabel CRD, you need to create a destination rule and a virtual service to route traffic to corresponding workloads based on the traffic labels.

1. Create an `example4.yaml` file that contains the following code.

The following code divides product page into multiple subsets such as `test1`, `test2`, and `test3`.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: productpage
spec:
  host: productpage
  subsets:
  - name: test1
    labels:
      version: test1
  - name: test2
    labels:
      version: test2
  - name: test3
    labels:
      version: test3
  ...
  - name: testn
    labels:
      version: testn
  - name: base
    labels:
      version: base
```

2. Run the following command to create a destination rule:

```
kubectl apply -f example4.yaml
```

3. Create an *example5.yaml* file that contains the following code:



```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: example-app
  namespace: default
spec:
  hosts:
    - example
  http:
    - match:
        - headers:
            userdefinlabel1:
              exact: test1
          route:
            - destination:
                host: example
                subset: test1
        - match:
            - headers:
                userdefinlabel1:
                  exact: test2
            route:
              - destination:
                  host: example
                  subset: test2
        - match:
            - headers:
                userdefinlabel1:
                  exact: test3
            route:
              - destination:
                  host: example
                  subset: test3
    - route:
        - destination:
            host: example
            subset: base
```

The headers and subset parameters determine the corresponding workloads to which traffic is routed based on the traffic labels. For example, in the preceding code, a request with the `userdefinlabel1:test1` label is routed to the workload that belongs to the subset `test1`. The following table describes the parameters.

Parameter	Description
<code>userdefinlabel1</code> under the <code>headers</code> parameter	The name of the traffic label.
<code>exact</code> under the <code>headers</code> parameter	The value of the traffic label.
<code>subset</code>	The subset to which the destination workload belongs.

4. Run the following command to create a virtual service:

```
kubectl apply -f example5.yaml
```

## Simplify the variable configuration of a virtual service

If a great many workload versions exist, the virtual service configuration is complicated. You can use the following code to simplify the variable configuration of the virtual service and degrade the traffic:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: example-app
  namespace: default
spec:
  hosts:
    - example
  http:
    - route:
        - destination:
            host: example
            subset: $userdefinelabel1
      fallback:
        case: noinstances|noavailable
        target:
          host: example
          subset: v1
```

The following table describes the parameters under `fallback`.

Parameter	Description
case	The scenario for traffic routing. You can specify multiple scenarios by using strings that are separated by vertical bars ( ). Valid values: <ul style="list-style-type: none"><li><i>noinstances</i>: No instance is specified.</li><li><i>noavailable</i>: The backend instance is unavailable.</li><li><i>noisolationunit</i>: The backend subset does not exist.</li></ul>
target	The service to which traffic is routed. The preceding code shows how to route traffic to the workload that belongs to the subset v1 if no instance is specified or the backend instance is unavailable.

## Related information

- [Use an ASM instance of a commercial edition to implement an end-to-end canary release](#)

# 13. Use an ASM AdaptiveConcurrency to implement adaptive concurrency control (Beta)

An ASM AdaptiveConcurrency can dynamically adjust the maximum number of concurrent requests that are allowed for a service based on the sampled request data. If the number of concurrent requests exceeds the maximum value supported by the service, excess requests are rejected to protect the service. This topic describes how to use an ASM AdaptiveConcurrency to implement adaptive concurrency control.

## Prerequisites

- An ASM instance of Enterprise Edition or Ultimate Edition is created. The version of the instance is V1.12.4.19 or later. For more information, see [Create an ASM instance](#).
- 

## Context

Services are expected to reject excess requests if the load capacity is exceeded. This prevents other chain reactions. You can use destination rules of ASM to implement a basic circuit breaking capability. You must specify a threshold for triggering circuit breaking, such as a specific number of pending requests. If the number of access requests on the data plane of an ASM instance exceeds the threshold, excess requests are rejected. However, it is difficult to accurately estimate the load capacity of a service in actual scenarios.

An ASM AdaptiveConcurrency uses an adaptive concurrency control algorithm to dynamically adjust the concurrency limit for a service by periodically comparing the sample latency with the calculated minimum latency and performing a series of computations. This way, the concurrency limit is near the load capacity of the service, and excess requests are rejected. If a request is rejected, the HTTP status code `503` and the error message `reached concurrency limit` are returned.

During the periodic calculation of the minimum round-trip time (MinRTT), the number of connections is limited to a small value that is specified by the `min_concurrency` parameter. After you create an ASM AdaptiveConcurrency for a service, we recommend that you create a destination rule to enable the retry feature for the service. This way, the requests that are rejected during MinRTT calculation can be served as much as possible based on the retries of the sidecar proxy.

## Step 1: Deploy sample applications

Deploy the testserver and gotest applications. Set the load capacity of the testserver application to 500 concurrent requests. Requests that exceed the concurrency limit are queued for processing. Set the time required to process each request to 1000 ms. Set each replica of the gotest application to initiate 200 requests at a time.

1. [Connect to ACK clusters by using kubectl](#).
2. Deploy the testserver application.

- i. Create a file named *testserver.yaml* and add the following content to the file:

Expand to view details

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: testserver
  name: testserver
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: testserver
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: testserver
    spec:
      containers:
      - args:
        - -m
        - "500"
        - -t
        - "1000"
        command:
        - /usr/local/bin/limited-concurrency-http-server
        image: registry.cn-hangzhou.aliyuncs.com/acs/asm-limited-concurrency-http-s
server:v0.1.1-gee0b08f-aliyun
        imagePullPolicy: IfNotPresent
        name: testserver
        ports:
        - containerPort: 8080
          protocol: TCP
```

The `-m` parameter specifies the maximum number of concurrent requests that the application supports. The `-t` parameter specifies the time required to process each request.

- ii. Run the following command to deploy the testserver application:

```
kubectl apply -f testserver.yaml
```

3. Deploy the testserver service.

- i. Create a file named *testservice.yaml* and add the following content to the file:

Expand to view details

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: testserver
  name: testserver
  namespace: default
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - name: http
      port: 8080
      protocol: TCP
      targetPort: 8080
    - name: metrics
      port: 15020
      protocol: TCP
      targetPort: 15020
  selector:
    app: testserver
  type: ClusterIP
```

- ii. Run the following command to deploy the testserver service:

```
kubectl apply -f testservice.yaml
```

#### 4. Deploy the gotest application.

- i. Create a file named *gotest.yaml* and add the following content to the file:

Expand to view details

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: gotest
    name: gotest
    namespace: default
spec:
  replicas: 0
  selector:
    matchLabels:
      app: gotest
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: gotest
    spec:
      containers:
      - args:
        - -c
        - "200"
        - -n
        - "10000"
        - -u
        - testserver:8080
        command:
        - /root/go-stress-testing-linux
        image: xocoder/go-stress-testing-linux:v0.1
        imagePullPolicy: Always
        name: gotest
        resources:
          limits:
            cpu: 500m
```

- ii. Run the following command to deploy the gotest application:

```
kubectl apply -f gotest.yaml
```

## Step 2: Create an ASMA adaptiveConcurrency

1. Use `kubectl` to connect to an ASM instance.
2. Create a file named `adaptiveconcurrency.yaml` and add the following content to the file:

Expand to view details

```

apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMAAdaptiveConcurrency
metadata:
  name: sample-adaptive-concurrency
  namespace: default
spec:
  workload_selector:
    labels:
      app: testserver
  sample_aggregate_percentile:
    value: 60
  concurrency_limit_params:
    max_concurrency_limit: 500
    concurrency_update_interval: 15s
  min_rtt_calc_params:
    interval: 60s
    request_count: 100
    jitter:
      value: 15
    min_concurrency: 50
    buffer:
      value: 25

```

Parameter	Type	Description	Required
workload_selector	WorkloadSelector	The workload selector that defines how to select the pod to use.	Yes
labels	map	The labels that are used to match the pod to select.	Yes
sample_aggregate_percentile	Percent	The sampling percentile. Requests are sampled based on this percentile to calculate the sample round-trip time (SampleRTT).	Yes
value	int	The percentile value. Valid values: 0 to 100.	Yes
concurrency_limit_params	ConcurrencyLimitParams	The configurations related to the concurrency limit.	Yes
max_concurrency_limit	int	The concurrency limit, which is the maximum number of concurrent requests that are allowed. Default value: 1000.	No
concurrency_update_interval	duration	The interval for updating the concurrency limit. Example: 60s.	Yes
min_rtt_calc_params	MinRTTCalcParams	The configurations related to MinRTT calculation.	Yes

Parameter	Type	Description	Required
interval	duration	The interval for calculating the MinRTT. Example: 120s.	No
request_count	int	The number of requests that are used to calculate the MinRTT. Default value: 50.	No
jitter	Percent	The percentage that is used to add a random jitter to the interval for calculating the MinRTT. For example, if the interval parameter is set to 120s and the jitter parameter is set to 50, the interval for calculating the MinRTT is random(120, 120 + (120 × 50%)). Default value: 15.	No
min_concurrency	int	The number of concurrent requests that are used to calculate the MinRTT. The value is also the initial concurrency limit of the ASMA adaptive concurrency. The value of this parameter must be much lower than the load capacity of the service. This ensures that responses are returned at the minimum latency during MinRTT calculation. Default value: 3.	No
buffer	Percent	The fluctuation range of the proper latency. The value is a percentage. For example, if a 10% fluctuation around a latency of 100 ms falls in a reasonable range, set this parameter to 10. Default value: 25.	No

3. Run the following command to create an ASMA adaptive concurrency:

```
kubectl apply -f adaptiveconcurrency.yaml
```

## Step 3: Enable ARMS Prometheus

To monitor the status of the ASMA adaptive concurrency and facilitate parameter tuning, you can use the Prometheus monitoring feature of Application Real-Time Monitoring Service (ARMS) to monitor the metrics of the ASMA adaptive concurrency.

1. For more information about how to enable ARMS Prometheus, see [Enable ARMS Prometheus](#).
2. Create a ServiceMonitor in the cluster in which the testserver service resides so that ARMS Prometheus can obtain the data of the testserver service.



- i. Create a file named *servicemonitor.yaml* and add the following content to the file:

Expand to view details

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: testserver-envoy-metrics
  namespace: default
spec:
  endpoints:
    - interval: 5s
      path: /stats/prometheus
      port: metrics
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      app: testserver
```

- ii. Run the following command to create a ServiceMonitor:

```
kubectl apply -f servicemonitor.yaml
```

## Step 4: Verify that the ASMAptiveConcurrency takes effect

1. Set the number of replicas to 5 for the gotest application.

A replica of the gotest application initiates 200 requests at a time. Five replicas of the application initiate 1,000 requests at a time in total.

- i.
- ii.
- iii.
- iv.
- v. On the **Deployments** page, set the **Namespace** parameter to default, find the gotest application, and then choose **More > View in YAML** in the **Actions** column.
- vi. In the **Edit YAML** dialog box, set the replicas parameter to 5 and click **Update**.
2. In the ARMS console, import a dashboard and view the status of the ASMAptiveConcurrency.
  - i.
  - ii.
  - iii.
  - iv. In the left-side navigation pane of the cluster details page, choose **Operations > Prometheus Monitoring**.
  - v. On the **Prometheus Monitoring** page, click **Go to ARMS Prometheus** in the upper-right corner.
  - vi. On the **Dashboards** page, click the name of a dashboard.
  - vii. In the left-side navigation pane, choose  > **Import**.
  - viii. Copy the following content in the **Import via panel json** field and click **Load**:

Expand to view details

```
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": "-- Grafana --",
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "description": "monitoring ASM Adaptive Concurrency",
  "editable": true,
  "gnetId": 6693,
  "graphTooltip": 0,
  "id": 3239002,
  "iteration": 1651922323976,
  "links": [],
  "panels": [
    {
      "aliasColors": {},
      "bars": false,
      "dashLength": 10,
      "dashes": false,
      "datasource": "$cluster",
      "fieldConfig": {
        "defaults": {
          "custom": {}
        },
        "overrides": []
      },
      "fill": 1,
      "fillGradient": 0,
      "gridPos": {
        "h": 8,
        "w": 12,
        "x": 0,
        "y": 0
      },
      "hiddenSeries": false,
      "id": 22,
      "legend": {
        "avg": false,
        "current": false,
        "max": false,
        "min": false,
        "show": true,
        "total": false,
        "values": false
      },
```

```

    },
    "lines": true,
    "linewidth": 1,
    "nullPointMode": "null",
    "options": {
      "alertThreshold": true
    },
    "percentage": false,
    "pluginVersion": "7.4.0-pre",
    "pointRadius": 2,
    "points": false,
    "renderer": "flot",
    "seriesOverrides": [],
    "spaceLength": 10,
    "stack": false,
    "steppedLine": false,
    "targets": [
      {
        "expr": "envoy_http_inbound_0_0_0_8080_adaptive_concurrency_gradient_controller_rq_blocked{service=\"$service\", pod=\"$pod\"}",
        "interval": "",
        "legendFormat": "{{service}}-{{pod}}",
        "queryType": "randomWalk",
        "refId": "A"
      }
    ],
    "thresholds": [],
    "timeFrom": null,
    "timeRegions": [],
    "timeShift": null,
    "title": "RqBlocked",
    "tooltip": {
      "shared": true,
      "sort": 0,
      "value_type": "individual"
    },
    "type": "graph",
    "xaxis": {
      "buckets": null,
      "mode": "time",
      "name": null,
      "show": true,
      "values": []
    },
    "yaxes": [
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      },
      {
        "format": "short"
      }
    ]
  }

```

```
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
    }
},
"yaxis": {
    "align": false,
    "alignLevel": null
}
},
{
    "aliasColors": {},
    "bars": false,
    "dashLength": 10,
    "dashes": false,
    "datasource": "$cluster",
    "fieldConfig": {
        "defaults": {
            "custom": {}
        },
        "overrides": []
    },
    "fill": 1,
    "fillGradient": 0,
    "gridPos": {
        "h": 8,
        "w": 12,
        "x": 12,
        "y": 0
    },
    "hiddenSeries": false,
    "id": 24,
    "legend": {
        "avg": false,
        "current": false,
        "max": false,
        "min": false,
        "show": true,
        "total": false,
        "values": false
    },
    "lines": true,
    "linewidth": 1,
    "nullPointMode": "null",
    "options": {
        "alertThreshold": true
    },
    "percentage": false,
    "pluginVersion": "7.4.0-pre",
    "pointRadius": 2,
    "points": false,
    "renderer": "flot",
```

```

    "seriesOverrides": [],
    "spaceLength": 10,
    "stack": false,
    "steppedLine": false,
    "targets": [
      {
        "expr": "envoy_http_inbound_0_0_0_0_8080_adaptive_concurrency_gradient_co
ntroller_burst_queue_size{service=\"$service\", pod=\"$pod\"}",
        "format": "time_series",
        "interval": "",
        "legendFormat": "{{service}}-{{pod}}",
        "queryType": "randomWalk",
        "refId": "A"
      }
    ],
    "thresholds": [],
    "timeFrom": null,
    "timeRegions": [],
    "timeShift": null,
    "title": "HeadRoom",
    "tooltip": {
      "shared": true,
      "sort": 0,
      "value_type": "individual"
    },
    "type": "graph",
    "xaxis": {
      "buckets": null,
      "mode": "time",
      "name": null,
      "show": true,
      "values": []
    },
    "yaxes": [
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      },
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      }
    ],
    "yaxis": {
      "align": false,
      "alignLevel": null
    }
  }

```

```
    }
  },
  {
    "aliasColors": {},
    "bars": false,
    "dashLength": 10,
    "dashes": false,
    "datasource": "$cluster",
    "fieldConfig": {
      "defaults": {
        "custom": {}
      },
      "overrides": []
    },
    "fill": 1,
    "fillGradient": 0,
    "gridPos": {
      "h": 8,
      "w": 12,
      "x": 0,
      "y": 8
    },
    "hiddenSeries": false,
    "id": 26,
    "legend": {
      "avg": false,
      "current": false,
      "max": false,
      "min": false,
      "show": true,
      "total": false,
      "values": false
    },
    "lines": true,
    "linewidth": 1,
    "nullPointMode": "null",
    "options": {
      "alertThreshold": true
    },
    "percentage": false,
    "pluginVersion": "7.4.0-pre",
    "pointradius": 2,
    "points": false,
    "renderer": "flot",
    "seriesOverrides": [],
    "spaceLength": 10,
    "stack": false,
    "steppedLine": false,
    "targets": [
      {
        "expr": "envoy_http_inbound_0_0_0_8080_adaptive_concurrency_gradient_co  
ntroller_concurrency_limit{service=\"$service\",pod=\"$pod\"}",
        "interval": "",
        "legendFormat": "{{service}}-{{pod}}",
```

```
        "queryType": "randomWalk",
        "refId": "A"
    }
],
"thresholds": [],
"timeFrom": null,
"timeRegions": [],
"timeShift": null,
"title": "ConcurrencyLimit",
"tooltip": {
    "shared": true,
    "sort": 0,
    "value_type": "individual"
},
"type": "graph",
"xaxis": {
    "buckets": null,
    "mode": "time",
    "name": null,
    "show": true,
    "values": []
},
"yaxes": [
    {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
    },
    {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
    }
],
"yaxis": {
    "align": false,
    "alignLevel": null
}
},
{
    "aliasColors": {},
    "bars": false,
    "dashLength": 10,
    "dashes": false,
    "datasource": "$cluster",
    "fieldConfig": {
        "defaults": {
            "custom": {}
        }
    }
}
```

```
    },
    "overrides": [],
  },
  "fill": 1,
  "fillGradient": 0,
  "gridPos": {
    "h": 8,
    "w": 12,
    "x": 12,
    "y": 8
  },
  "hiddenSeries": false,
  "id": 28,
  "legend": {
    "avg": false,
    "current": false,
    "max": false,
    "min": false,
    "show": true,
    "total": false,
    "values": false
  },
  "lines": true,
  "linewidth": 1,
  "nullPointMode": "null",
  "options": {
    "alertThreshold": true
  },
  "percentage": false,
  "pluginVersion": "7.4.0-pre",
  "pointradius": 2,
  "points": false,
  "renderer": "flot",
  "seriesOverrides": [],
  "spaceLength": 10,
  "stack": false,
  "steppedLine": false,
  "targets": [
    {
      "expr": "envoy_http_inbound_0_0_0_8080_adaptive_concurrency_gradient_controller_gradient{service=\"$service\",pod=\"$pod\"}",
      "interval": "",
      "legendFormat": "{{service}}-{{pod}}",
      "queryType": "randomWalk",
      "refId": "A"
    }
  ],
  "thresholds": [],
  "timeFrom": null,
  "timeRegions": [],
  "timeShift": null,
  "title": "Gradient",
  "tooltip": {
    "shared": true,
    "sort": 0
```



```

      "sort": 0,
      "value_type": "individual"
    },
    "type": "graph",
    "xaxis": {
      "buckets": null,
      "mode": "time",
      "name": null,
      "show": true,
      "values": []
    },
    "yaxes": [
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      },
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      }
    ],
    "yaxis": {
      "align": false,
      "alignLevel": null
    }
  },
  {
    "aliasColors": {},
    "bars": false,
    "dashLength": 10,
    "dashes": false,
    "datasource": "$cluster",
    "fieldConfig": {
      "defaults": {
        "custom": {}
      },
      "overrides": []
    },
    "fill": 1,
    "fillGradient": 0,
    "gridPos": {
      "h": 8,
      "w": 12,
      "x": 0,
      "y": 16
    },
    "hiddenSeries": false.
  }

```

```

    "id": 32,
    "legend": {
      "avg": false,
      "current": false,
      "max": false,
      "min": false,
      "show": true,
      "total": false,
      "values": false
    },
    "lines": true,
    "linewidth": 1,
    "nullPointMode": "null",
    "options": {
      "alertThreshold": true
    },
    "percentage": false,
    "pluginVersion": "7.4.0-pre",
    "pointRadius": 2,
    "points": false,
    "renderer": "flot",
    "seriesOverrides": [],
    "spaceLength": 10,
    "stack": false,
    "steppedLine": false,
    "targets": [
      {
        "expr": "envoy_http_inbound_0_0_0_8080_adaptive_concurrency_gradient_controller_min_rtt_msecs{service=\"$service\",pod=\"$pod\"}",
        "interval": "",
        "legendFormat": "{{service}}-{{pod}}",
        "queryType": "randomWalk",
        "refId": "A"
      }
    ],
    "thresholds": [],
    "timeFrom": null,
    "timeRegions": [],
    "timeShift": null,
    "title": "MinRTT(msec)",
    "tooltip": {
      "shared": true,
      "sort": 0,
      "value_type": "individual"
    },
    "type": "graph",
    "xaxis": {
      "buckets": null,
      "mode": "time",
      "name": null,
      "show": true,
      "values": []
    },
    "yaxes": [

```

```
{
  "format": "ms",
  "label": null,
  "logBase": 1,
  "max": null,
  "min": null,
  "show": true
},
{
  "format": "short",
  "label": null,
  "logBase": 1,
  "max": null,
  "min": null,
  "show": true
}
],
"yaxis": {
  "align": false,
  "alignLevel": null
}
},
{
  "aliasColors": {},
  "bars": false,
  "dashLength": 10,
  "dashes": false,
  "datasource": "$cluster",
  "fieldConfig": {
    "defaults": {
      "custom": {}
    },
    "overrides": []
  },
  "fill": 1,
  "fillGradient": 0,
  "gridPos": {
    "h": 8,
    "w": 12,
    "x": 12,
    "y": 16
  },
  "hiddenSeries": false,
  "id": 34,
  "legend": {
    "avg": false,
    "current": false,
    "max": false,
    "min": false,
    "show": true,
    "total": false,
    "values": false
  },
  "lines": true,
```

```
"linewidth": 1,
"nullPointMode": "null",
"options": {
  "alertThreshold": true
},
"percentage": false,
"pluginVersion": "7.4.0-pre",
"pointRadius": 2,
"points": false,
"renderer": "flot",
"seriesOverrides": [],
"spaceLength": 10,
"stack": false,
"steppedLine": false,
"targets": [
  {
    "expr": "envoy_http_inbound_0_0_0_8080_adaptive_concurrency_gradient_controller_sample_rtt_msecs{service=\"$service\",pod=\"$pod\"}",
    "interval": "",
    "legendFormat": "{{service}}-{{pod}}",
    "queryType": "randomWalk",
    "refId": "A"
  }
],
"thresholds": [],
"timeFrom": null,
"timeRegions": [],
"timeShift": null,
"title": "SampleRTT(msec)",
"tooltip": {
  "shared": true,
  "sort": 0,
  "value_type": "individual"
},
"type": "graph",
"xaxis": {
  "buckets": null,
  "mode": "time",
  "name": null,
  "show": true,
  "values": []
},
"yaxes": [
  {
    "format": "ms",
    "label": null,
    "logBase": 1,
    "max": null,
    "min": null,
    "show": true
  },
  {
    "format": "short",
    "label": null,
```

```

        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
    }
],
"yaxis": {
    "align": false,
    "alignLevel": null
}
},
{
    "aliasColors": {},
    "bars": false,
    "dashLength": 10,
    "dashes": false,
    "datasource": "test-adaptive-concurrency_1217520382582089",
    "fieldConfig": {
        "defaults": {
            "custom": {}
        },
        "overrides": []
    },
    "fill": 1,
    "fillGradient": 0,
    "gridPos": {
        "h": 8,
        "w": 12,
        "x": 0,
        "y": 24
    },
    "hiddenSeries": false,
    "id": 30,
    "legend": {
        "avg": false,
        "current": false,
        "max": false,
        "min": false,
        "show": true,
        "total": false,
        "values": false
    },
    "lines": true,
    "linewidth": 1,
    "nullPointMode": "null",
    "options": {
        "alertThreshold": true
    },
    "percentage": false,
    "pluginVersion": "7.4.0-pre",
    "pointradius": 2,
    "points": false,
    "renderer": "flot",
    "seriesOverrides": [],
    "seriesThreshold": 10
}

```

```
    "spaceLength": 10,
    "stack": false,
    "steppedLine": false,
    "targets": [
      {
        "expr": "envoy_http_inbound_0_0_0_8080_adaptive_concurrency_gradient_controller_min_rtt_calculation_active(service=\"${service}\",pod=\"${pod}\"),",
        "interval": "",
        "legendFormat": "{{service}}-{{pod}}",
        "queryType": "randomWalk",
        "refId": "A"
      }
    ],
    "thresholds": [],
    "timeFrom": null,
    "timeRegions": [],
    "timeShift": null,
    "title": "MinRTTCalc",
    "tooltip": {
      "shared": true,
      "sort": 0,
      "value_type": "individual"
    },
    "type": "graph",
    "xaxis": {
      "buckets": null,
      "mode": "time",
      "name": null,
      "show": true,
      "values": []
    },
    "yaxes": [
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      },
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      }
    ],
    "yaxis": {
      "align": false,
      "alignLevel": null
    }
  }
}
```

```

    "refresh": "5s",
    "schemaVersion": 26,
    "style": "dark",
    "tags": [],
    "templating": {
      "list": [
        {
          "current": {
            "selected": true,
            "text": "edas120_1217520382582089",
            "value": "edas120_1217520382582089"
          },
          "error": null,
          "hide": 0,
          "includeAll": false,
          "label": null,
          "multi": false,
          "name": "cluster",
          "options": [],
          "query": "prometheus",
          "queryValue": "",
          "refresh": 1,
          "regex": "",
          "skipUrlSync": false,
          "type": "datasource"
        },
        {
          "allValue": null,
          "current": {
            "isNone": true,
            "selected": false,
            "text": "None",
            "value": ""
          },
          "datasource": "$cluster",
          "definition": "label_values(envoy_http_inbound_0_0_0_8080_adaptive_concurrency_gradient_controller_burst_queue_size,service)",
          "error": null,
          "hide": 0,
          "includeAll": false,
          "label": null,
          "multi": false,
          "name": "service",
          "options": [],
          "query": "label_values(envoy_http_inbound_0_0_0_8080_adaptive_concurrency_gradient_controller_burst_queue_size,service)",
          "refresh": 2,
          "regex": "",
          "skipUrlSync": false,
          "sort": 1,
          "tagValuesQuery": "",
          "tags": [],
          "tagsQuery": "",
          "type": "query",

```

```
        "useTags": false
      },
      {
        "allValue": null,
        "current": {
          "selected": false,
          "text": "All",
          "value": "$__all"
        },
        "datasource": "$cluster",
        "definition": "label_values(envoy_http_inbound_0_0_0_8080_adaptive_concur  
rency_gradient_controller_concurrency_limit, pod)",
        "error": null,
        "hide": 0,
        "includeAll": true,
        "label": null,
        "multi": true,
        "name": "pod",
        "options": [],
        "query": "label_values(envoy_http_inbound_0_0_0_8080_adaptive_concurrency  
_gradient_controller_concurrency_limit, pod)",
        "refresh": 2,
        "regex": "",
        "skipUrlSync": false,
        "sort": 0,
        "tagValuesQuery": "",
        "tags": [],
        "tagsQuery": "",
        "type": "query",
        "useTags": false
      }
    ]
  },
  "time": {
    "from": "now-15m",
    "to": "now"
  },
  "timepicker": {
    "refresh_intervals": [
      "5s",
      "10s",
      "30s",
      "1m",
      "5m",
      "15m",
      "30m",
      "1h",
      "2h",
      "1d"
    ],
    "time_options": [
      "5m",
      "15m",
      "1h",

```



```

    "6h",
    "12h",
    "24h",
    "2d",
    "7d",
    "30d"
  ]
},
"timezone": "",
"title": "ASM Adaptive Concurrency",
"uid": "000000084",
"version": 3
}

```

ix. On the **Options** page, click **Import**.

**Note** If a message is displayed on the **Options** page, indicating that a dashboard UID conflict occurs, change the dashboard UID and click **Import**.

x. On the dashboard, select the cluster in which the ASM AdaptiveConcurrence resides. Set the **Service** parameter to **testserver** and the **Pod** parameter to **ALL**.



As shown in the preceding figure, the `gotest` application initiates 1,000 requests to the `testserver` service, but the number of concurrent requests that are processed by the `testserver` service stays in 500. This indicates that the ASM AdaptiveConcurrence takes effect.

# 14. Manage Spring Cloud services

You can connect Spring Cloud applications to Alibaba Cloud Service Mesh (ASM) so that you can use cloud-native service governance capabilities to manage Spring Cloud services without modifying business code. This topic describes how to use ASM to manage Spring Cloud services.

## Prerequisites

- An ASM instance of Enterprise Edition or Ultimate Edition is created. For more information, see [Create an ASM instance](#).
- 
- 

## Context

Spring Cloud is a standard with different implementations, such as Spring Cloud Netflix, Spring Cloud Alibaba, and Spring Cloud Consul. For ASM, the core difference among different Spring Cloud implementations is that they use different registries. The following table describes whether ASM supports migration between registries for different Spring Cloud versions.

Spring Cloud version	Registry	Whether migration is supported without code modification
Spring Cloud Alibaba	Microservice Engine (MSE) Nacos (an Alibaba Cloud service)	Supported
Spring Cloud Alibaba	Self-managed Nacos	Supported
Spring Cloud Netflix	Eureka	Not supported. We recommend that you use Nacos as the registry.
Spring Cloud Consul	Consul	Not supported. We recommend that you use Nacos as the registry.
Spring Cloud Zookeeper	Zookeeper	Not supported. We recommend that you use Nacos as the registry.

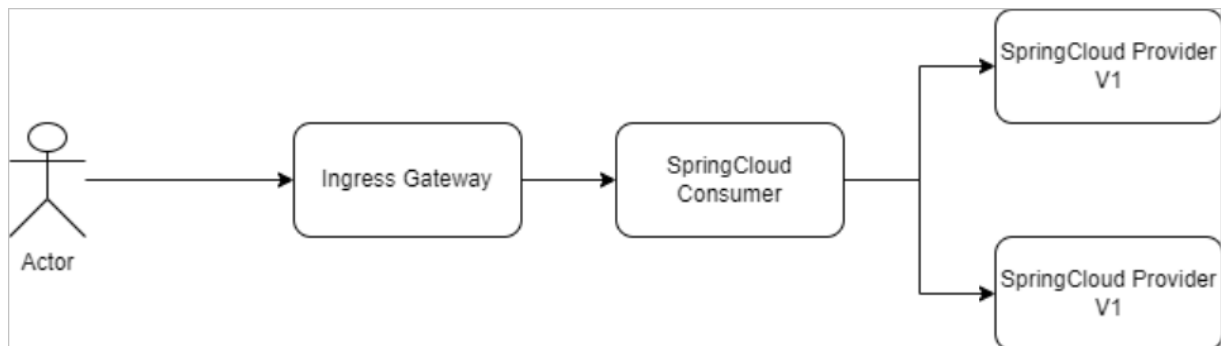
## Demo introduction

You can download the sample files for the Spring Cloud services that are deployed in this topic from the [nacos-examples](#) directory on GitHub.

The Spring Cloud services involved include a consumer service and a provider service. The provider service has two versions: V1 and V2. Both versions of the provider service are registered with the Nacos registry. The consumer service synchronizes the endpoints of the two versions of the provider service from the Nacos registry and sends requests to the endpoints in a load balancing manner. The consumer service exposes port 8080 and provides an echo interface. After the requests are forwarded to the provider service, the provider service returns corresponding responses, and then the consumer service delivers the responses. Different versions of the provider service return different responses.

- The provider service of V1 responds to an echo request with the following information: `Hello Nacos Discovery From v1xxx`.
- The provider service of V2 responds to an echo request with the following information: `Hello Nacos Discovery From v2xxx`.

The `xxx` string in a response indicates the specific parameter in the corresponding echo request. For example, if the `/echo/world` request is sent to the provider service of V1, the `Hello Nacos Discovery From v1world` response is returned.



## Step 1: Configure a service entry and an Envoy filter

You must configure a service entry and an Envoy filter in ASM so that ASM can manage Spring Cloud services.

1. Use `kubectl` to connect to an ASM instance.
2. Create a service entry.
  - i. Create an `external-nacos-svc.yaml` file that contains the following code:

```

apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: external-nacos-svc
spec:
  hosts:
    - "NACOS_SERVER_HOST" ## Set this parameter to the endpoint of your Nacos server
                           host. Example: mse-xxx-p.nacos-ans.mse.aliyuncs.com.
  location: MESH_EXTERNAL
  ports:
    - number: 8848
      name: http
  resolution: DNS
  
```

Port 8848 is the default port used by Nacos. If you use a self-managed Nacos server and a custom port number, you must set the number parameter to the port number that you use.

- ii. Run the following command to create a service entry:

```
kubectl apply -f external-nacos-svc.yaml
```

### 3. Create an Envoy filter.

- i. Create an *external-envoyfilter.yaml* file that contains the following code:

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  labels:
    provider: "asm"
    asm-system: "true"
  name: nacos-subscribe-lua
  namespace: istio-system
spec:
  configPatches:
    # The first patch adds the lua filter to the listener/http connection manager
    - applyTo: HTTP_FILTER
      match:
        proxy:
          proxyVersion: "^1.*"
        context: SIDECAR_OUTBOUND
        listener:
          portNumber: 8848
          filterChain:
            filter:
              name: "envoy.filters.network.http_connection_manager"
              subFilter:
                name: "envoy.filters.http.router"
      patch:
        operation: INSERT_BEFORE
        value: # lua filter specification
        name: envoy.lua
        typed_config:
          "@type": "type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua"
          inlineCode: |
            -- copyright: ASM (Alibaba Cloud ServiceMesh)
            function envoy_on_request(request_handle)
              local request_headers = request_handle:headers()
              -- /nacos/v1/ns/instance/list?healthyOnly=false&namespaceId=public&clientIP=11.122.XX.XX&serviceName=DEFAULT_GROUP%40%40service-provider&udpPort=53174&encoding=UTF-8
              local path = request_headers:get(":path")
              if string.match(path, "^/nacos/v1/ns/instance/list") then
                local servicename = string.gsub(path, ".*&serviceName=.%40([%w\\.\\_\\-\\-]+)&.*", "%1")
                request_handle:streamInfo():dynamicMetadata():set("context", "request.path", path)
                request_handle:streamInfo():dynamicMetadata():set("context", "request.servicename", servicename)
                request_handle:logInfo("subscribe for serviceName: " .. servicename)
              else
```

```

        request_handle:streamInfo():dynamicMetadata():set("context", "request.path", "")
    end
end
function envoy_on_response(response_handle)
    local request_path = response_handle:streamInfo():dynamicMetadata():get("context")["request.path"]
    if request_path == "" then
        return
    end
    local servicename = response_handle:streamInfo():dynamicMetadata():get("context")["request.servicename"]
    response_handle:logInfo("modified response ip to serviceName:" .. servicename)

    local bodyObject = response_handle:body(true)
    local body= bodyObject:getBytes(0,bodyObject:length())
    body = string.gsub(body, "%s+", "")
    body = string.gsub(body, "(ip\\":\\") (%d+.%d+.%d+.%d+)", "%1"..servicename)

    response_handle:body():setBytes(body)
end

```

- ii. Run the following command to create an Envoy filter:

```
kubectl apply -f external-envoyfilter.yaml
```

## Step 2: Create Spring Cloud services in an ACK cluster

### Note

- The registration process needs to be intercepted. Therefore, you must create an Envoy filter before you create deployments. If specific deployments are created before you create the Envoy filter, you must enable rolling update for the deployments.
- For Spring Cloud services, you must create Kubernetes service resources and provide a cluster IP address.

1. [Connect to ACK clusters by using kubectl.](#)
2. Run the following commands to create Spring Cloud services:

```

export NACOS_ADDRESS=xxxx # Replace xxxx with the endpoint of the MSE Nacos registry or
a self-managed Nacos registry. We recommend that you use a virtual private cloud (VPC)
endpoint.
wget https://alibabacloudservicemesh.oss-cn-beijing.aliyuncs.com/asm-labs/springcloud/demo.yaml -O demo.yaml
sed -e "s/NACOS_SERVER_CLUSTERIP/$NACOS_ADDRESS/g" demo.yaml | kubectl apply -f -

```

3. Run the following command to check the Spring Cloud services:

```
kubectl get pods
```

Expected output:

consumer-bdd464654-jn8q7	2/2	Running	0	25h
provider-v1-66bc67fb6d-46pgl	2/2	Running	0	25h
provider-v2-76568c45f6-85z87	2/2	Running	0	25h

## Step 3: Create a gateway rule and a virtual service

1. Use `kubectl` to connect to an ASM instance.
2. Create a gateway rule.
  - i. Create a `test-gateway.yaml` file that contains the following code:

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

```

- ii. Run the following command to create a gateway rule:

```
kubectl apply -f test-gateway.yaml
```

3. Create a virtual service.
  - i. Create a `consumer.yaml` file that contains the following code:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: consumer
spec:
  hosts:
  - "*"
  gateways:
  - test-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        host: consumer.default.svc.cluster.local
        port:
          number: 8080

```

- ii. Run the following command to create a virtual service:

```
kubectl apply -f consumer.yaml
```

## Step 4: Check whether ASM can manage the Spring Cloud services

1. Query the IP address of the ingress gateway.

- i.
- ii.
- iii.

- iv. On the details page of the ASM instance, click **ASM Gateways** in the left-side navigation pane.

On the ASM Gateways page, view the IP address of the ingress gateway in the **Kubernetes Service** column.

2. Run the following command to initiate requests from the ingress gateway to the Spring Cloud consumer service:

```
curl <IP address of the ingress gateway>/echo/world
```

Expected output:

```
Hello Nacos Discovery From v1world
Hello Nacos Discovery From v2world
Hello Nacos Discovery From v1world
Hello Nacos Discovery From v2world
```

The output shows that traffic is routed to V1 and V2 of the provider service in polling mode by default.

3. Create a destination rule and a virtual service.

- i. Create a *service-provider.yaml* file that contains the following code:

```
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: service-provider
spec:
  host: service-provider
  subsets:
  - name: v1
    labels:
      label: v1
  - name: v2
    labels:
      label: v2
```

- ii. Run the following command to create a destination rule:

```
kubectl apply -f service-provider.yaml
```

- iii. Create a *service-provider1.yaml* file that contains the following code.

The virtual service to be created defines that */echo/hello* requests are routed to the provider service of V1 and other requests are routed to the provider service of V2.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: service-provider
spec:
  hosts:
  - service-provider
  http:
  - name: "hello-v1"
    match:
    - uri:
        prefix: "/echo/hello"
    route:
    - destination:
        host: service-provider
        subset: v1
  - name: "default"
    route:
    - destination:
        host: service-provider
        subset: v2
```

- iv. Run the following command to create a virtual service:

```
kubectl apply -f service-provider1.yaml
```

4. Run the following command to initiate requests to the Spring Cloud consumer service:

```
curl <IP address of the ingress gateway>/echo/hello
```

Expected output:

```
Hello Nacos Discovery From v1hello
Hello Nacos Discovery From v1hello
```

The output shows that */echo/hello* requests are routed to the provider service of V1 and other requests are routed to the provider service of V2. This indicates that Spring Cloud traffic is taken over by Istio and Custom Resource Definitions (CRDs) provided by Istio can be used to configure routing rules. In this case, ASM can manage the Spring Cloud services.

## FAQ

**What do I do if the Spring Cloud services that I deployed by following the steps in this topic do not take effect?**

1. Check whether throttling is enabled for the port or IP address of Nacos.
2. Make sure that an Envoy filter is created before you create deployments so as to intercept the registration process. If specific deployments are created before you create the Envoy filter, you must enable rolling update for the deployments.
3. Check whether you have created Kubernetes service resources and provided a cluster IP address for



the Spring Cloud services.

4. Make sure that the client SDK of Nacos is of a version earlier than V2.0. The client SDK of V2.0 or later uses gRPC to connect to the server, which is beyond the application scope of the solution provided in this topic.