Tencent Cloud Mesh
Operation Guide

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

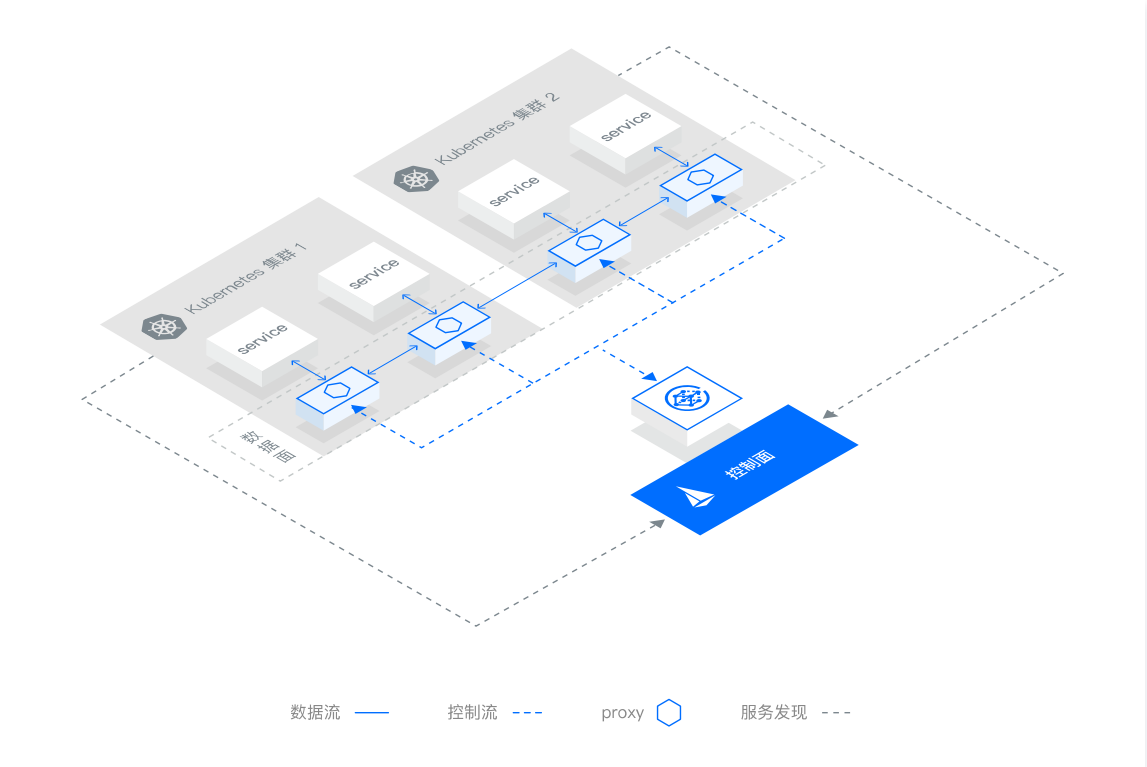
We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Operation Guide
Mesh Instance Management
Overview

Last updated: 2024-08-08 17:19:51

A mesh instance is a logical isolation space for managing services. Services within the same mesh can communicate with each other.



Below are the lifecycle status explanations for the mesh:

Status	Description
Creating	During mesh creation, mesh details cannot be viewed.
Running	The mesh is in normal operating status.
Upgrading	The mesh is undergoing a version upgrade, and some features are unavailable.
Idle	When the Service Discovery Cluster managed by the mesh is deleted or disassociated, it will enter the Idle State. In the idle state, the mesh can be viewed normally, but due to the lack of business entities, some features will be unavailable. You can re-add the Service Discovery Cluster to the mesh to restore its normal state.
Invalid	When the mesh remains in Idle State for more than 30 days, or the standalone mesh's main cluster is deleted, the mesh will enter an Invalid Status. You will no longer be able to perform operations on the mesh other than deletion.
Abnormal	Some components in the mesh are abnormal, causing the mesh features to be affected.

The following configurations are required during mesh creation:

- **Adding a service discovery cluster**
  By adding Kubernetes clusters for service discovery, the services within the cluster can be automatically discovered. It can also be achieved through manual service registration. The discovered services in the mesh will appear in the **TCM Console > Mesh Details > Services** page. Once a service is discovered, it can be accessed by other services within the mesh. For detailed guidelines, please refer to Service Discovery Management.

- **Creating an Edge Proxy Gateway**
  Edge Proxy Gateways are divided into Ingress and Egress types, serving as the entry and exit points for mesh traffic. The Ingress type of Edge Proxy Gateway must be created for traffic to enter the mesh, while the creation of Egress type Edge Proxy Gateway is optional. For detailed guidelines, please refer to Edge Proxy Gateway Management.
- **Sidecar Injection for Services**
  The Sidecar container is responsible for data plane traffic management, rule enforcement, and monitoring reporting, serving as the foundation for mesh traffic governance and observability. Therefore, for services requiring traffic management and observability, a Sidecar must be injected. For detailed guidelines, please refer to Mesh Configuration.
- **Observability Backend Service Configuration**
  Observability is divided into three parts: monitor metric viewing, link tracking, and log management. TCM supports integration with Tencent Cloud Prometheus Monitoring TMP, APM, and Log Service CLS, providing richer and integrated observability capabilities. TCM also supports integration with third-party Prometheus and Jaeger/Zipkin services, offering greater component extensibility for users. For detailed guidelines, please refer to Observability.

After the mesh is created, traffic rule scheduling can be performed for the mesh. Traffic governance rules can be created for the mesh through the console or by submitting YAML file configurations. Currently, TCM is fully compatible with Istio native syntax. For detailed guidelines, please refer to Traffic Management.

Creating a Mesh

Last updated: 2024-08-08 17:20:14

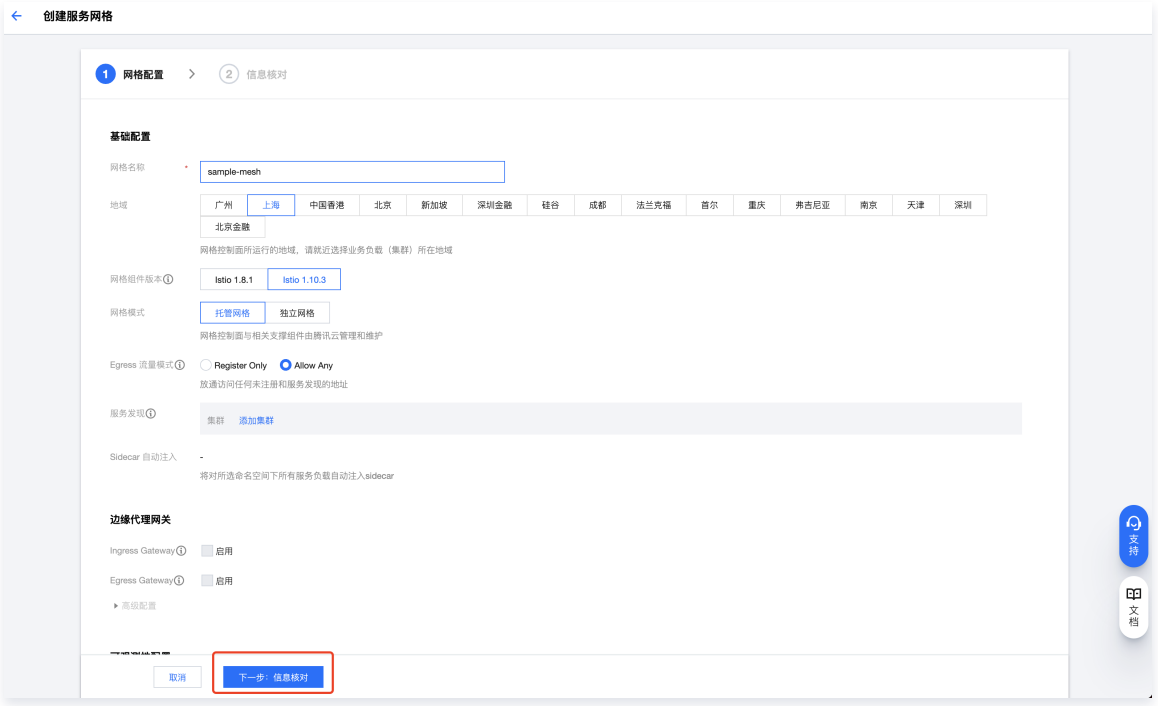
To use TCM, you first need to create a TCM instance. Mesh instances have regional attributes but can manage services across multiple regions. This article explains how to create a TCM instance through the console.

> **Note:**
> Each account is allowed to create up to 20 meshes by default. If you need to create more, you can apply by submitting a ticket.

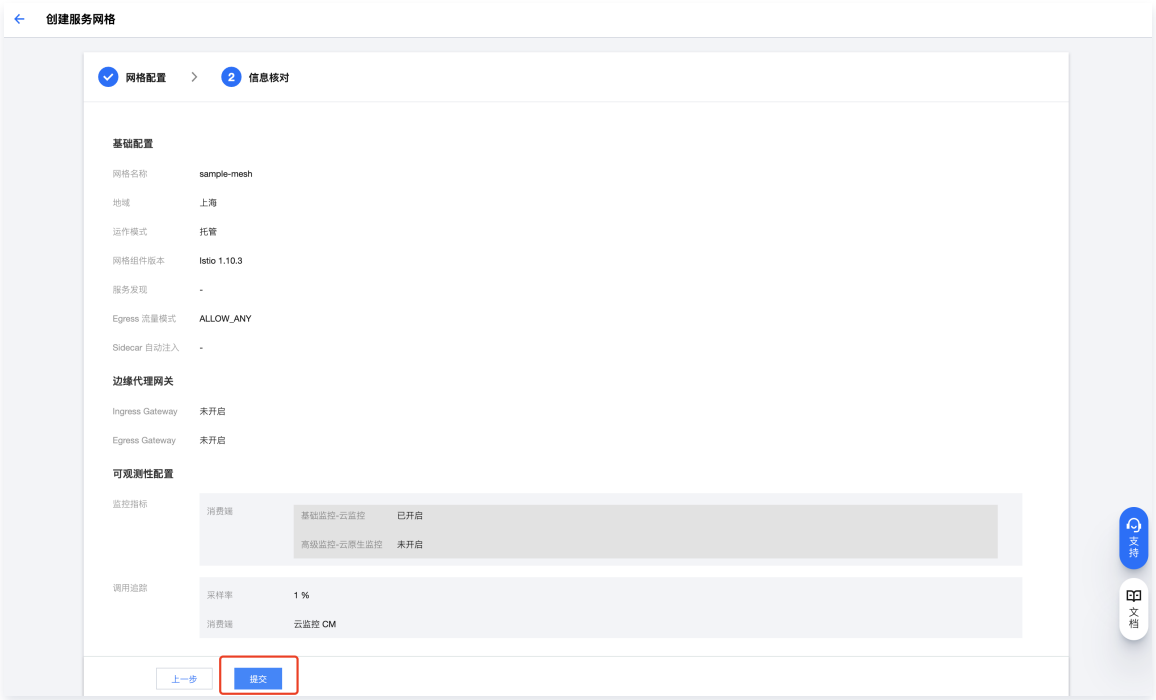
Operation step

The following are the console steps to create a new TCM instance:

1. log in to TCM console.
2. Select a region, click **Create** at the top-left corner of the page.
3. On the Create TCM page, fill in the mesh creation related configurations as needed. For the description of configuration items, please refer to Description of Mesh Creation Configuration Items. After completing, click **Next: Information Verification**. As shown below:



4. On the information verification page, confirm that the creation configurations are correct and click **Submit** to start the mesh creation process.



5. After the mesh creation process is completed, you can view the TCM instance on the TCM list page.

Description of Mesh Creation Configuration Items

Configuration Item	Description	Required
Mesh Name	The name of the created TCM.	Yes
Region	The region where the TCM control plane runs. The control plane running region can be different from the business load (e.g., cluster) region. It is recommended to choose the region nearest to the business load (cluster).	Yes
Mesh Component Version	Select the version of the control plane and data plane. TCM provides support for the latest two major versions compatible with the Istio community.	Yes
Mesh Mode	Choose the deployment mode of mesh control plane related components. The managed mesh control plane components are managed and maintained by Tencent Cloud. The standalone mesh control plane components will be deployed within the cluster you specify, and you need to manage and maintain the control plane components within the cluster. By default, the managed mesh is optional. The standalone mesh requires an allowlist to be used. You can apply for use through online consultation.	Yes
Egress Traffic Mode	Configure the pass-through policy for external access of services within the mesh. You can choose Registry Only (only supports accessing services automatically discovered by the mesh and manually registered services) or Allow Any (can access any address).	Yes

Service discovery	Specify the cluster for TCM automatic service discovery. The cluster must meet constraints such as versions, permissions, and segment conflict.	No
Sidecar Automatic Injection	Configure the Namespace for automatic sidecar injection. Once enabled, it will automatically inject sidecars into all service loads under the selected namespace. Automatic injection only takes effect for newly created service loads; existing service loads need to be restarted to inject the sidecar. For further details on exceptions to the custom sidecar injection, see Custom Sidecar Injection.	No
External Request Bypass Sidecar	Corresponds to excludeIPRanges. By default, the sidecar takes over all traffic within the current pod. If you want the access to specific IPs to bypass the sidecar proxy, you can configure this option. After configuration, request traffic for that IP range will not use Istio traffic management and observability features. Configuration changes only take effect on new pods, and existing pods need to be restarted.	No
Sidecar Readiness Assurance	Use the HoldApplicationUntilProxyStarts feature to configure business containers to wait until the sidecar has completed startup before starting, ensuring that pods dependent on the sidecar for business container runtime operate normally.	No
Sidecar Stop Assurance	After enabling, the Sidecar stops and waits for the processes in the business container to terminate completely, which will increase the Pod termination duration to some extent. It is recommended to enable it for services where business processes cannot be shut down at any time. For Istio versions before 1.12, TCM uses a preset container prestop script to check for no remaining business processes before allowing the business container to exit. This means that other prestop scripts configured by the user will interfere with this feature. For versions after 1.12, the new feature EXIT_ON_ZERO_ACTIVE_CONNECTIONS is used to achieve this.	No
Custom Definition Sidecar resources	By default, TCM configures a maximum resource limit of 2 cores and 1GB for the Sidecar container, which is sufficient in most cases. When your mesh scale grows or the logic in the Sidecar increases, the default resource limit may be insufficient. You can modify the resource limit based on your business needs.	No
Ingress Gateway	Create an Ingress Gateway for the mesh. For TKE/EKS clusters, a CLB type Ingress Gateway is created by default, and you need to configure related CLB creation options. For registered clusters, because it is uncertain whether the cluster can use Tencent Cloud CLB, only a LoadBalancer type Gateway Service is created.	No
Egress Gateway	If you need to centrally manage the outbound traffic of the mesh, such as unified exit, unified authentication, rule configuration, etc., you need to create an Egress Gateway. The backend will create a ClusterIP type Egress Gateway service for you.	No
Gateway Deployment Mode	You can choose between a normal deployment or a dedicated deployment mode. For details, see Edge Proxy Gateway Deployment Mode.	No
Gateway Scaling Policy	Configure the HPA strategy for the edge proxy gateway deployed in the specified cluster.	No
Gateway Resource Definition	Custom Definition of pod resource limits for Ingress/Egress Gateway.	No
Monitoring Metrics Consumer End	Configure the backend service for the mesh's monitoring metrics. Currently, Prometheus monitoring TMP is supported. After configuration, the monitoring metrics will be reported to TMP. TCM console metrics are displayed based on the TMP data source, and users can also use TMP independently in the TMP console. Without configuring the monitoring metrics consumer end, the mesh will not be able to use the monitoring metrics display, topology, and other related features.	No
Call Tracing Consumer End	Configure the backend service for the mesh's call tracing. Currently, APM is supported. After configuration, tracing data will be reported to APM from the Sidecar. TCM console metrics are displayed based on the APM data source, and users can also use APM independently in the APM console. Without configuring the call tracing consumer end, the mesh will not be able to use the call tracing view and other related features.	No
Call Tracing Sampling Rate	Grid collection and persistent invocation of Tracing's sampling ratio. Sidecar collection and reporting data consume resources and bandwidth in direct relation to data volume.	No

	Please configure as needed: 100% is recommended for development and testing environments, and 1% for production environments.	
Scope of Access Log Enabling	To avoid unnecessary overhead, TCM supports enabling Sidecar logs for specific gateways or namespaces.	No
Access Log Format	TCM supports two log formats: JSON and TXT.	No
Access Log Output Template	Sidecar log field settings offer two predefined templates: default and enhanced. The enhanced template adds a TraceID field to the default format. If users need further field customization, they can follow the Envoy Standard Specification for custom log field definition.	No
Access Log Consumer	Configure the backend service for Sidecar logs. Currently, CLS is supported. Once enabled, a log collection component will be deployed on cluster nodes to ensure the feature operates correctly.	No

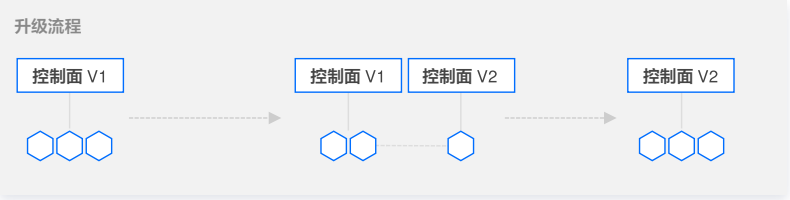
Upgrading a Mesh

Last updated: 2024-08-08 17:27:54

TCM provides a mesh upgrade service that allows users to upgrade their mesh from a lower version to a higher version. The upgrade process follows the gray upgrade principle and consists of the following steps:

1. Deploy the control plane of a new version to upgrade the control plane of Tencent Cloud Mesh.
2. Conduct a canary upgrade of the data plane, and restart services to update sidecars of existing service pods.
3. Verify the upgrade to check that the services are normal.
4. Old version control plane decommissioning, upgrade completed.

Before the control plane of the old version goes offline, you can roll back to the state before the upgrade. The upgrade process is shown as follows:



Operation step

1. log in to TCM console.
2. When the mesh version can be upgraded, the mesh interface will prompt that a new version is available, as shown below:

3. After selecting **More > Upgrade**, follow the guide to perform the upgrade.
   The upgrade will proceed in three phases: **Control Plane Upgrade > Data Plane Upgrade > Old Control Plane Decommissioning**.
   Before the old control plane goes offline, you can roll back to the state before the upgrade.

Control Plane Upgrade

Control Plane Upgrade, TCM will deploy new version control plane components:

Data Plane Upgrade

Data Plane Upgrade, including service data plane upgrade and Gateway upgrade.

The business data plane upgrade allows users to change the automatic injection of Sidecar in a specified Namespace to a new version. Once selected, newly created business Pods in that Namespace will inject the new version of Sidecar. **Existing business Pods will be updated to the new version only after being rebuilt**. Since restarting may impact the availability of services, the platform will not automatically rebuild business Pods. **Users need to manually rebuild them**.

Note:

- You can republish a service through a pipeline or manually rebuild a workload by directly using command lines such as kubectl patch and kubectl rollout restart.
- In some scenarios, sidecars will be uninstalled instead of being upgraded. For example, assume that a namespace has enabled sidecar injection, sidecars have been successfully injected into some service pods, and then namepsace-level sidecar injection is disabled. After a service pod is restarted, its sidecars will be uninstalled unless a sidecar injection label has been independently set for the pod.

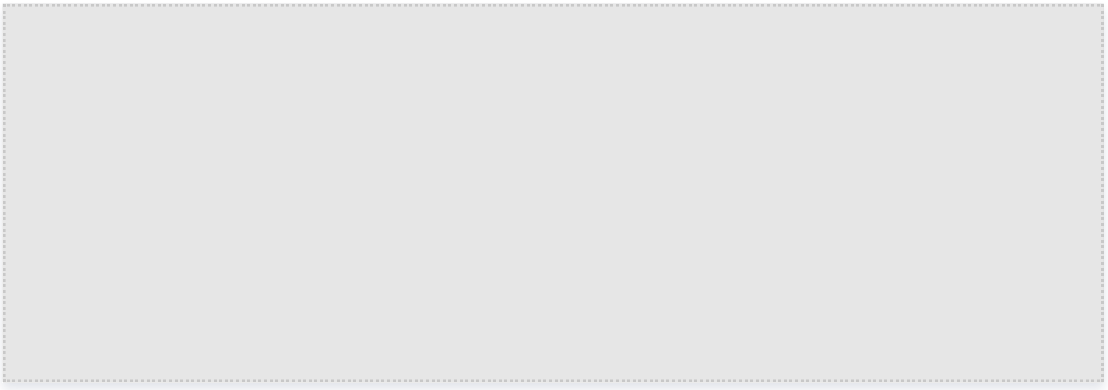
Gateway upgrade involves selecting all Gateways that need to be upgraded to the new version and clicking **Upgrade** on the right side as shown below:

After all data plane upgrades are completed, click **Upgrade** to proceed to the next step.

Upgrade Verification

Due to feature changes in the version upgrade, compatibility issues may arise. After rebuilding the business pods, you need to check if the business operates normally. If you find that the upgrade causes business abnormalities, you can select rollback in the upgrade interface to revert the data plane sidecar to the original version.

4. Choose **Complete** or **Cancel upgrade**. In "Data plane upgrade", clicking **Upgrade** or **Rollback** will check if the current stock of Pods meet the conditions to proceed to Next. When every Namespace has switched to the new version of the control plane, and all existing business Pods Sidecars have been updated to the new version, you can choose **Upgrade** to proceed to the Next step **of decommissioning the old version of the control plane** to complete the upgrade. Or, if all Namespaces have switched back to the old version of the control plane and all existing business Pods Sidecars are using the old version, you can click **Rollback** to proceed to Next and decommission the new version of the control plane, cancelling the upgrade.



Updating Mesh Configurations

Last updated: 2024-08-08 17:28:13

This topic describes how to update the configuration of a running service mesh.

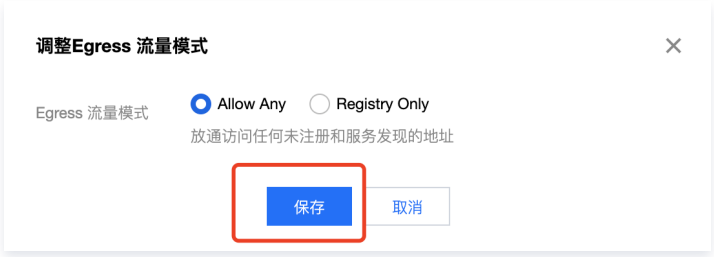
Modifying the Egress Traffic Mode

The egress traffic mode is a policy that configures the external access of services within the mesh. You can choose Registry Only (allows access to automatically discovered or manually registered services) or Allow Any (allows access to any address).
Steps for configuring the egress traffic mode for the mesh are as follows:

1. log in to TCM console, click the Grid ID you wish to modify to enter the grid's management page.

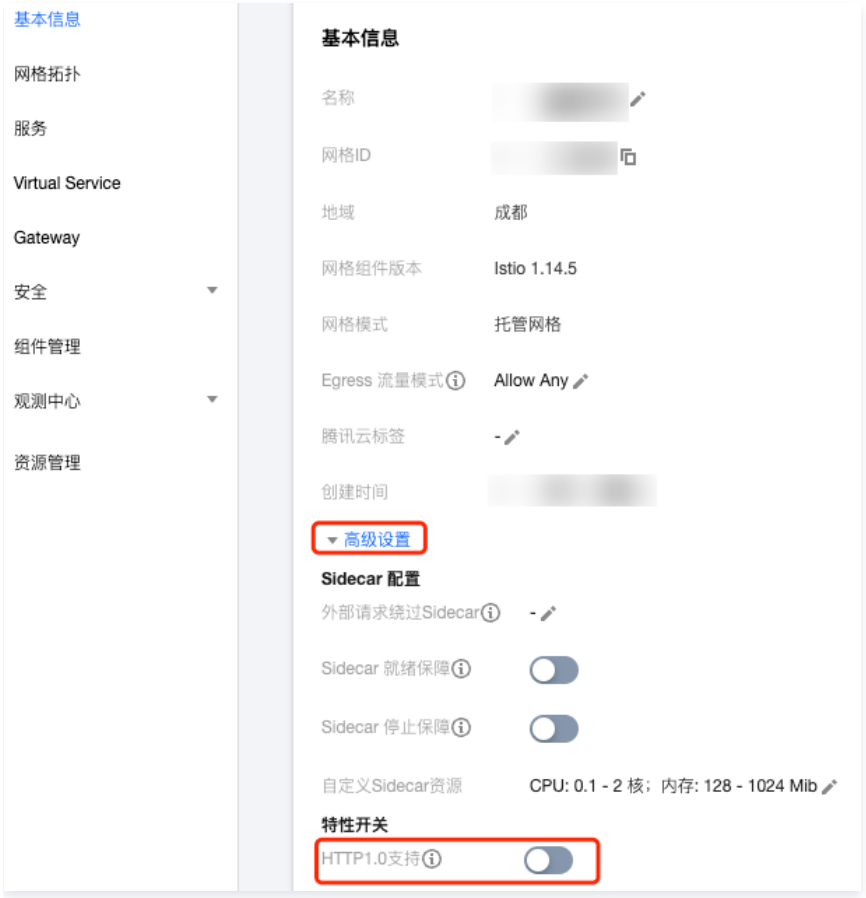
2. On the grid's basic information page, click the edit button in the Egress traffic mode section to enter the **Adjust Egress Traffic Mode** pop-up.

3. Choose either **Allow Any** or **Registry Only** as needed, then click **Save** to update the Egress traffic mode.



Enabling HTTP 1.0 Support

Istio does not support HTTP 1.0 by default. When necessary, you need to enable HTTP 1.0 support on the mesh basic information page:



Disabling HTTP Auto Retries

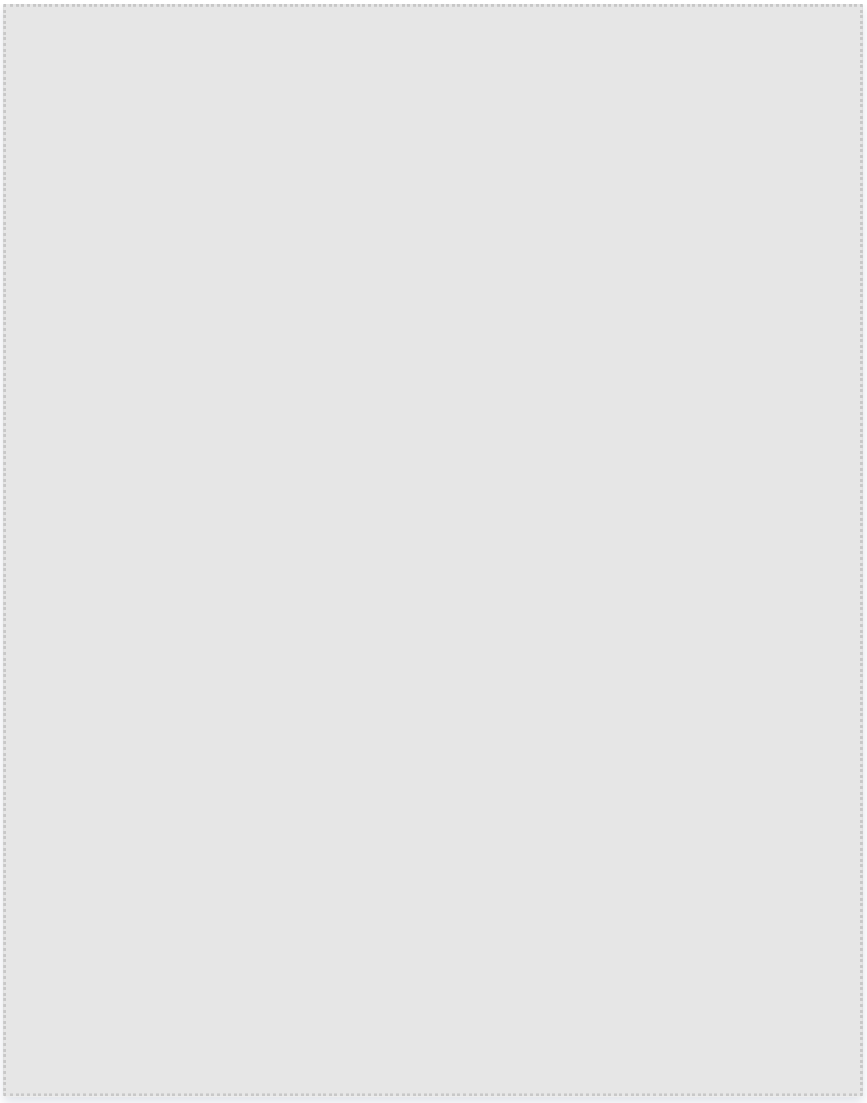
Istio automatically retries failed HTTP requests twice by default. If this does not meet your requirements, you can disable auto retries on the mesh basic information page:

Disabling auto retries applies to the entire mesh. However, you can still set explicit retry policies for specific virtual services.

Enabling DNS Proxy

Istio's sidecar supports DNS proxying. When enabled, DNS traffic will be intercepted and responded to directly by the sidecar. This allows for DNS caching to accelerate DNS responses, and in multi-cluster mesh scenarios, services can be resolved without creating same-named services in the client's cluster. Follow these steps to enable DNS forwarding.

1. log in to TCM console, click the Grid ID you wish to modify to enter the grid's management page.
2. On the basic information page, click **DNS Proxying > DNS Forwarding** on the right side of to enable DNS forwarding as shown below:



If you need to automatically assign IPs to ServiceEntries without Definition addresses, you can enable **Automatic IP Allocation**. See more in Address Auto Allocation.

Sidecar Injection and Configuration

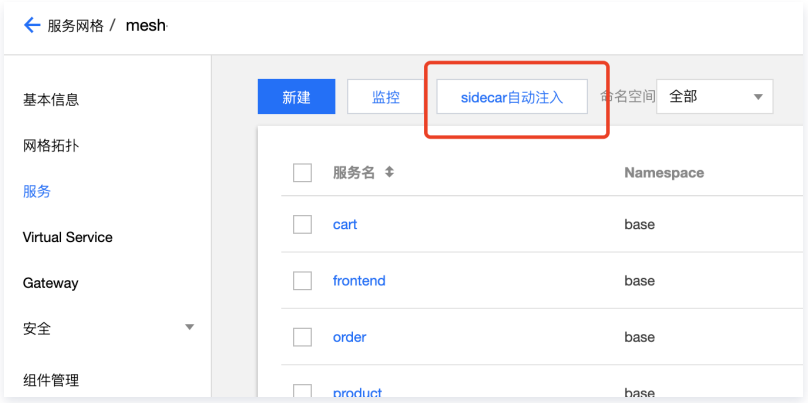
Last updated: 2024-08-08 17:28:33

Configure Sidecar Automatic Injection

TCM currently supports enabling Sidecar automatic injection for a specified Namespace via the console. Once enabled, any new workload created under that Namespace will automatically have the mesh Sidecar installed. Since injection is completed during workload creation, enabling injection will not automatically install the Sidecar for existing workloads. You can complete Sidecar automatic injection by rebuilding the workload.
Here are the steps to configure Namespace-level Sidecar automatic injection:

1. log in to TCM console, click the Grid ID you wish to modify to enter the grid's management page.

2. On the service list page, click **Sidecar Auto Injection** to open the **Sidecar Auto Injection Configuration** popup.



3. Select the **Sidecar** automatic injection for the required **namespace** and click **Confirm** to complete the Sidecar automatic injection configuration.

If you do not want to configure Sidecar automatic injection through the console, you can modify the Namespace's yaml by adding the label `istio.io/rev:{istio version}`:

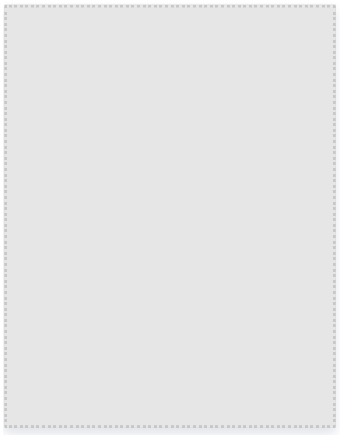
```
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: "2024-03-05T09:52:56Z"
  labels:
```

```
    istio.io/rev: 1-18-5 # Remember to modify the version number
    kubernetes.io/metadata.name: test
  name: test
  resourceVersion: "29528054435"
  uid: 9a7491bb-2f3d-4017-bf95-2a21ebb31ba4
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

Note:
It cannot be used together with the community default label `istio-injection: enabled`, otherwise it will not be effective!

Self Definition Sidecar Injection

TCM also supports enabling Sidecar automatic injection for specific workloads by editing the yaml. If needed, you can add the label on the Pod: `istio.io/rev:{istio version}` (Note the settings related to Sidecar injection tags, there's slight difference between TCM and istio default syntax). Example:



If you need to add an exception for a specific Pod under a Namespace that has automatic injection enabled to prevent Sidecar automatic injection, you can add the following to the Pod label: `sidecar.istio.io/inject="false"`. Pod-level injection override the Namespace-level injection. For more details about Sidecar automatic injection, refer to the Istio documentation Installing Sidecar.

Bypass Traffic for Specified Subnets

If you want to bypass certain traffic, such as traffic uploading files to COS COS storage (internal target IP starting with 169.254), traffic interception may cause high Sidecar memory usage if the downloaded file is large, as Sidecar caches the request content and reuses it during automatic retries. In this case, you can configure **External requests bypassing Sidecar** in the advanced settings on the grid's basic information page to add the subnets you want to bypass. Here are the steps:

1. log in to TCM console, click the Grid ID you wish to modify to enter the grid's management page.

2. On the Basic Information page, click **External Request Bypass Sidecar** on the right side of . As shown below:

3. In the **Edit External Requests Bypassing Sidecar** pop-up window, add the subnets you don't want to intercept. See the figure below:

4. click **Save**.

The above is a global configuration. If you want to target specific workloads, you can add the annotation `traffic.sidecar.istio.io/excludeOutboundIPRanges` to the Pod. For details, see Annotation List.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        'traffic.sidecar.istio.io/excludeOutboundIPRanges': '169.254.0.0/16'
      labels:
        app: nginx
```

Control Sidecar startup order

When Kubernetes starts a Pod, containers are started simultaneously. In the case of using Istio, since the traffic will be intercepted by the Sidecar, if the Sidecar starts slower than the business container, network requests at the start of the business container will fail because the traffic is intercepted by the Sidecar, but the Sidecar is not yet ready. Common scenarios include: large cluster scale, slow xds fetch resulting in slow Sidecar startup, and the business process needs to fetch configuration from the registry upon startup. Since traffic is intercepted by the Sidecar and it is not yet ready to handle the traffic, configuration fetching fails, the business process reports an error and exits, resulting in container exit.

There are mainly two solutions. The first is to make the business code more resilient, retrying continuously until the request succeeds at startup; the second is to start the Sidecar first, wait until it is ready, and then start the business container. You may refer to the following steps to enable Sidecar Readiness Guarantee.

1. log in to TCM console, click the Grid ID you wish to modify to enter the grid's management page.
2. On the Basic Information page, click **Sidecar Readiness Guarantee** on the right side of . As shown in the figure below:

The above is a global configuration. If you only wish to target specific workloads, you can add the following annotation to the Pod:

```
proxy.istio.io/config: '{ "holdApplicationUntilProxyStarts" : true }'
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        proxy.istio.io/config: '{ "holdApplicationUntilProxyStarts" : true }'
      labels:
        app: nginx
```

Sidecar graceful termination

During business release, the workload updates incrementally. During the termination process of a Pod, the Sidecar by default waits only a few seconds before it forcibly stops. If the business request itself takes a long time, or if it uses long connections, it might lead to the interruption of some normal business requests and connections. We hope that the Sidecar will wait for existing business requests and connections to end before exiting to achieve a graceful termination. Starting from istio 1.12, Sidecar has supported the `EXIT_ON_ZERO_ACTIVE_CONNECTIONS` environment variable, which functions to wait for the Sidecar to "drain", and during response, it also notifies the client to close long connections (for HTTP1, it responds with the header "Connection: close", and for HTTP2, it responds with the GOAWAY frame). You can refer to the following steps to enable Sidecar termination protection.

1. log in to TCM console, click the Grid ID you wish to modify to enter the grid's management page.

2. On the Basic Information page, click **Sidecar Stopping Guarantee** on the right side of . As shown in the figure below:

The above method is for global enablement, which is generally recommended. If you only wish to target specific workloads, you can add the following annotation to the Pod:

```
proxy.istio.io/config: '{ "proxyMetadata": { "EXIT_ON_ZERO_ACTIVE_CONNECTIONS": "true" } }'
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        proxy.istio.io/config: '{ "proxyMetadata": { "EXIT_ON_ZERO_ACTIVE_CONNECTIONS": "true" } }'
      labels:
        app: nginx
```

Custom Definition for sidecar resources

Sidecar, as a container in the Pod, also needs to set request and limit. If necessary, custom definitions can be created. You can perform global custom definitions in the advanced settings on the grid basic information page. The steps are as follows:

1. log in to TCM console, click the Grid ID you wish to modify to enter the grid's management page.
2. On the Basic Information page, click **Custom Definition for Sidecar Resources** on the right side of . As shown in the figure below:

3. In the **Edit Custom Definition for Sidecar Resources** popup window, edit the custom-defined resources. As shown in the figure below:

4. click **Save**.

If you want different custom definitions for different workloads, you can add annotations similar to the following to the Pod:

```
sidecar.istio.io/proxyCPU: "0.5"
sidecar.istio.io/proxyCPULimit: '2'
sidecar.istio.io/proxyMemory: "256Mi"
sidecar.istio.io/proxyMemoryLimit: '2Gi'
```

If you are using TKE Serverless and do not want the request and limit of the Sidecar to significantly increase the Pod specification, you can use annotations to set only the request without setting the limit. Fill in the request according to actual needs; entering "0" indicates that the Pod specification will not be upgraded due to the Sidecar:

```
sidecar.istio.io/proxyCPU: "0"
sidecar.istio.io/proxyMemory: "0"
```

Deleting a Mesh

Last updated: 2024-08-08 17:28:47

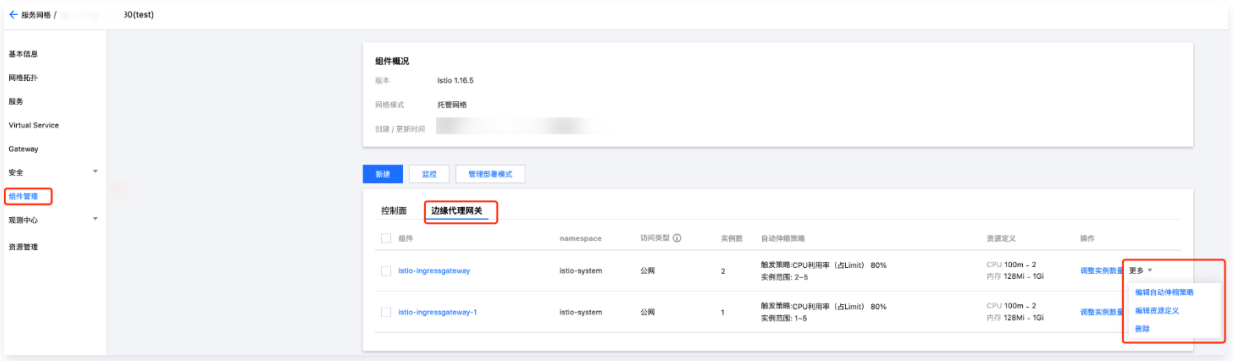
This section describes how to delete a service mesh instance.

Operation step

> **Note:**
> To prevent accidental operations, you need to ensure that **all edge proxy gateways created within the mesh are deleted** and **all clusters associated with the mesh are disassociated** before you can proceed with the mesh deletion.

Deleting a Gateway

1. log in to TCM Console to enter the mesh list page.
2. Select the region where TCM belongs at the top of the page.
3. click the mesh name to enter the details page, and select **Component Management** as shown below:



4. In the **Edge Proxy Gateway**, select **More > Delete** on the right side of the component.

Disassociate cluster

1. In the mesh list page, click the mesh name to enter the basic information page.
2. In **Service Discovery**, click **Disassociate** on the right side of the cluster as shown below:

Delete mesh instance

In the mesh list, click **Delete** in the operation column on the right side of the mesh and confirm the operation.


服务网格
地域
上海
扫码关注公众号
新建
一键体验
多个关键字用竖线 "|" 分隔，多个过滤标签用回车键
ID/名称
监控
状态
网格组件版本
网格模式
服务数量
集群
操作
mesh
运行中
Istio 1.10.3
托管网格
0
-
删除

Manage Mesh with TCCLI

Last updated: 2024-08-08 17:29:06

Tool Introduction

The Istio community provides TCCLI istioctl to help users manage meshes. However, for mesh instances hosted on TCM, due to the limitations of the hosted architecture and the managed service itself, direct use of Istioctl is not supported. Therefore, we offer TCCLI tcmctl for use with managed meshes.

Preconditions

1. Managed Mesh has been created and added to the TKE Standard Cluster.
2. The environment can connect to any cluster managed by the mesh using kubectl (with the correct kubeconfig configuration)

Compatibility

The following are the differences between tcmctl and istioctl:

Command type	istioctl(1.12) Commands	Feature	Compatibility
Basic Commands	admin	Manage control plane (istiod) configuration	Not supported, the managed control plane does not support users modifying configurations via Definition
	analyze	Analyze Istio configuration and print validation messages	Supported
	authz	(authz is experimental. Use istioctl experimental authz)	Supported
	bug-report	Cluster information and log capture support tool.	Supported
	completion	generate the autocompletion script for the specified shell	Supported
	create-remote-secret	Create a secret with credentials to allow Istio to access remote Kubernetes apiservers	Not supported, the managed control plane does not allow creating Secrets for the control plane
	dashboard	Access to Istio web UIs	Partially supported, limited by the managed control plane, only dashboard envoy commands are supported
	experimental	Experimental commands that may be modified or deprecated	Partially supported, for details see "Experimental Command Support"
	help	Help about any command	Supported
	install	Applies an Istio manifest, installing or reconfiguring Istio on a cluster.	Not supported
	upgrade	Upgrade Istio control plane in-place	Not supported, managed meshes do not support managing instance lifecycles via commands
	verify-install	Verifies Istio Installation Status	Supported
	manifest	Commands related to Istio manifests	Supported
	operator	Commands related to Istio operator controller.	Supported
	profile	Commands related to Istio configuration profiles	Supported

	kube-inject	Inject Istio sidecar into Kubernetes pod resources	Not supported, the TCM injection mechanism is uniformly managed by the control plane, command injection is not supported
	proxy-config	Retrieve information about proxy configuration from Envoy [kube only]	Supported
	proxy-status	Retrieves the synchronization status of each Envoy in the mesh [kube only]	Supported
	remote-clusters	Lists the remote clusters each istiod instance is connected to.	Supported
	tag	Command group used to interact with revision tags	Not supported. The tag mechanism is not used. Tag operations are not supported
	validate	Validate Istio policy and rules files	Supported
	version	Prints out build version information	Supported
Experimental command	add-to-mesh	Add workloads into Istio service mesh	Not supported. Managed Mesh does not allow self-control
	remove-from-mesh	Remove workloads from Istio service mesh	Supported
	authz	Inspect Istio AuthorizationPolicy	Supported
	create-remote-secret	Create a secret with credentials to allow Istio to access remote Kubernetes apiservers	Supported
	describe	Describe resource and related Istio configuration	Supported
	injector	List sidecar injector and sidecar versions	Not supported. Sidecar injector is managed. Display information cannot be changed
	internal-debug	Retrieves the debug information of istio	Not supported. Debug operations on the managed control plane are not allowed
	kube-uninject	Uninject Envoy sidecar from Kubernetes pod resources	Not supported. The injection mechanism is managed by the control plane. Sidecar cannot be removed via command
	metrics	Prints the metrics for the specified workload(s) when running in Kubernetes.	Not supported. Custom observability for core control plane components is not allowed.
	precheck	check whether Istio can safely be installed or upgrade	Not supported. Managed control plane does not support community inspection operations
	version	Prints out build version information	Not supported. Version information cannot be obtained through xds-address. Please use the "tcmctl version" command
	proxy-status	Retrieves the synchronization status of each Envoy in the mesh	Not supported. Proxy status information cannot be obtained through xds-address. Please use the "tcmctl proxy-status" command
	config	Configure istioctl defaults	Supported
	remote-clusters	Lists the remote clusters each istiod instance is connected to.	Supported

revision	Provide insight into various revisions (istiod, gateways) installed in the cluster	Not supported. TCM does not use the tag mechanism. Tag subcommands are not supported
uninstall	Uninstall Istio from a cluster	Not supported. Managed Mesh does not allow instance lifecycle management through commands
wait	Wait for an Istio resource	Supported
workload	Commands to assist in configuring and deploying workloads running on VMs and other non-Kubernetes environments	Not supported. Sidecar injection configuration is managed by the control plane, therefore modification operations are not supported

For TCCLI tcmctl supported commands, the syntax is completely consistent with the community istioctl. You can use `-help` to query related syntax. For more usage information on istioctl, you can refer to istioctl documentation.

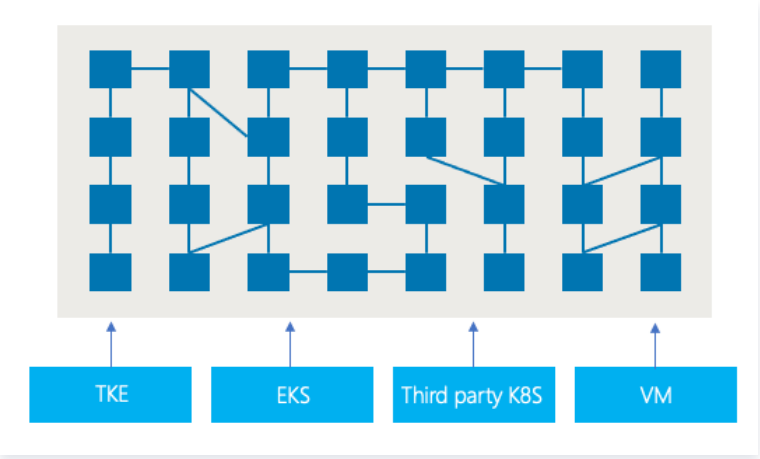
Use Method

1. Download the appropriate version of tcmctl. Choose and download the appropriate version based on your environment: tcmctl-linux, tcmctl-windows, tcmctl-mac.
2. Grant execution permission to the tool and add it to the PATH environment variable.
3. Connect to the mesh, similar to kubectl. By default, tcmctl uses `$HOME/.kube/config` to connect to clusters in the mesh. You can switch kubeconfig by configuring `--context`.

Service Discovery Management Overview

Last updated: 2024-08-09 15:27:52

Service discovery is to add specific business services to a mesh. It is a prerequisite for service governance and observation. TCM supports automatic discovery of services in K8s clusters and manual registration of other types of services, such as VMs and cloud databases. In terms of K8s clusters, it not only supports TKE Standard Clusters but also TKE Super Nodes and TKE Serverless Clusters. You can also register third-party K8s clusters to the TKE console, and TCM supports adding these registered third-party clusters to the mesh.



For details about how to add K8s clusters and heterogeneous services to TCM, please refer to the following documents:

- Automatic Service Discovery
- Manual Service Registration

Automatic Service Discovery

Last updated: 2024-08-09 15:28:29

Operation scenarios

A Tencent Cloud Mesh can associate with multiple TKE clusters and automatically discover K8s services in the clusters. You can associate multiple TKE clusters when creating the mesh or on the mesh basic information page, and Tencent Cloud Mesh will automatically display the services in the clusters on the **Service** page.

Use Limits

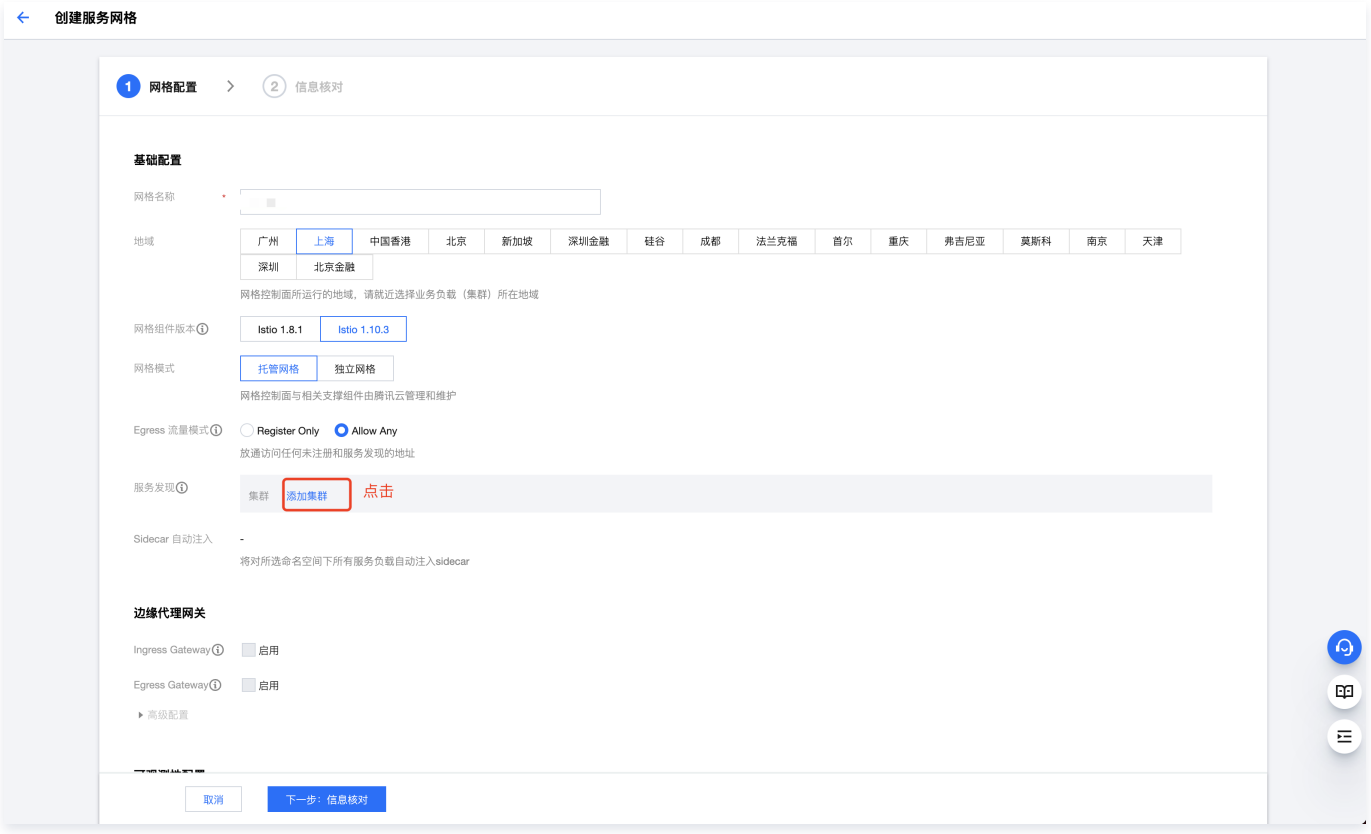
- **Cluster version**: Tencent Cloud Mesh does not enforce that the cluster versions are exactly the same, but the cluster versions should meet requirements of Istio for the corresponding K8s versions. For details, see Supported Releases.
- **Cluster permission**: You need to have admin permissions for the cluster to be added to the mesh. For details, see Adding Mesh Permissions for a Cluster.
- **VPC Network**: For multiple clusters not in the same VPC, to ensure normal cross-cluster Pod access, it is necessary to use CCN to connect the clusters. Please add the clusters to the same CCN instance.**Ensure there are no conflicts between the host CIDR and container CIDR in the VPCs of each end within the CCN instance.**
- **Container network**: If a TKE cluster uses the Global Router mode, you need to register the container network with CCN, so that newly added container CIDRs can be accessed.

Operation step

Mesh creation page

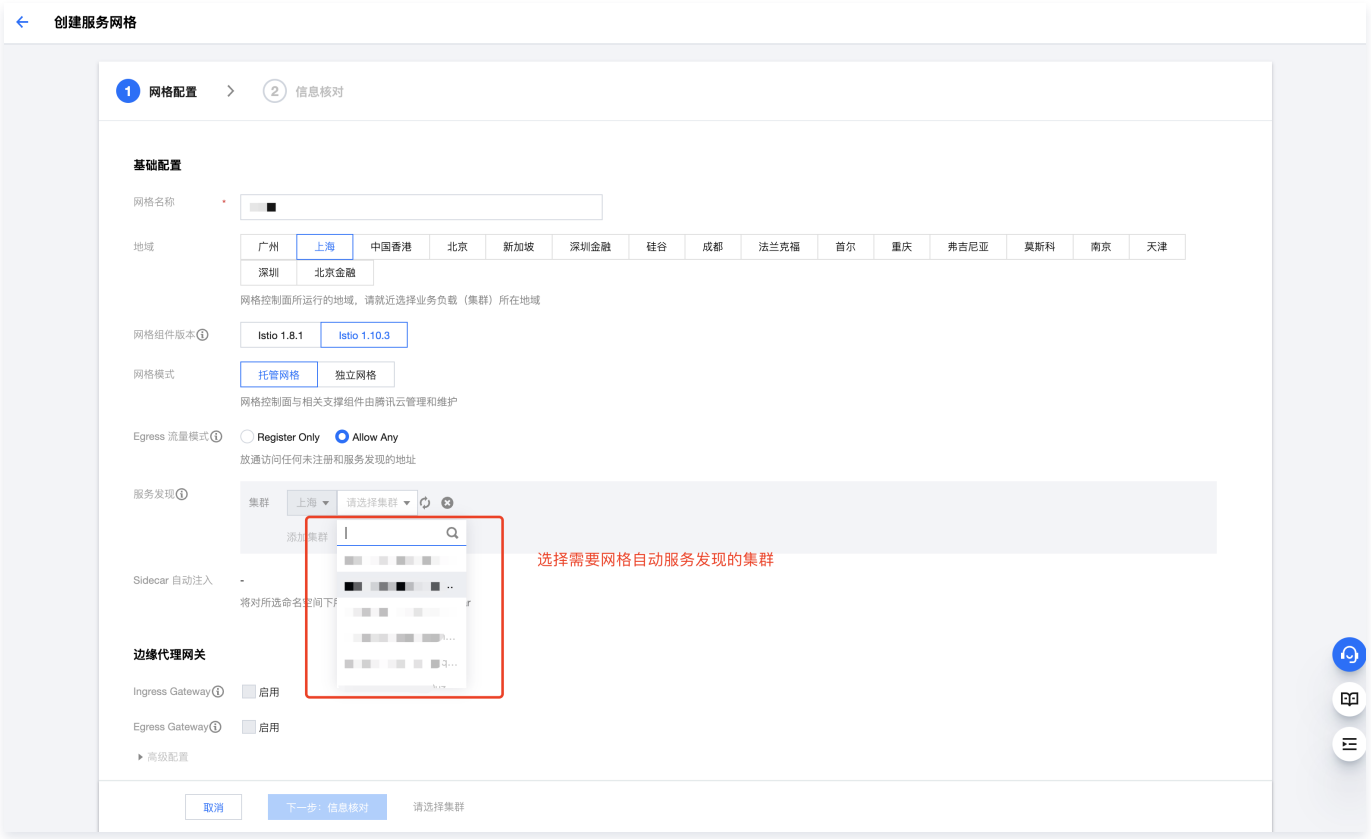
You can configure the automatic service discovery cluster when creating the mesh, on the mesh creation configuration page:

1. log in to TCM console.
2. Click **Create** to create Tencent Cloud Mesh.
3. Under **Basic Information** > **Service Discovery** parameters, click **Add Cluster**. As shown below:



4. Select the Kubernetes clusters for automatic service discovery that you want to add. You can add multiple clusters at the same time. After submitting the mesh creation request, the created mesh instance can automatically discover K8s services in the

clusters. As shown below:

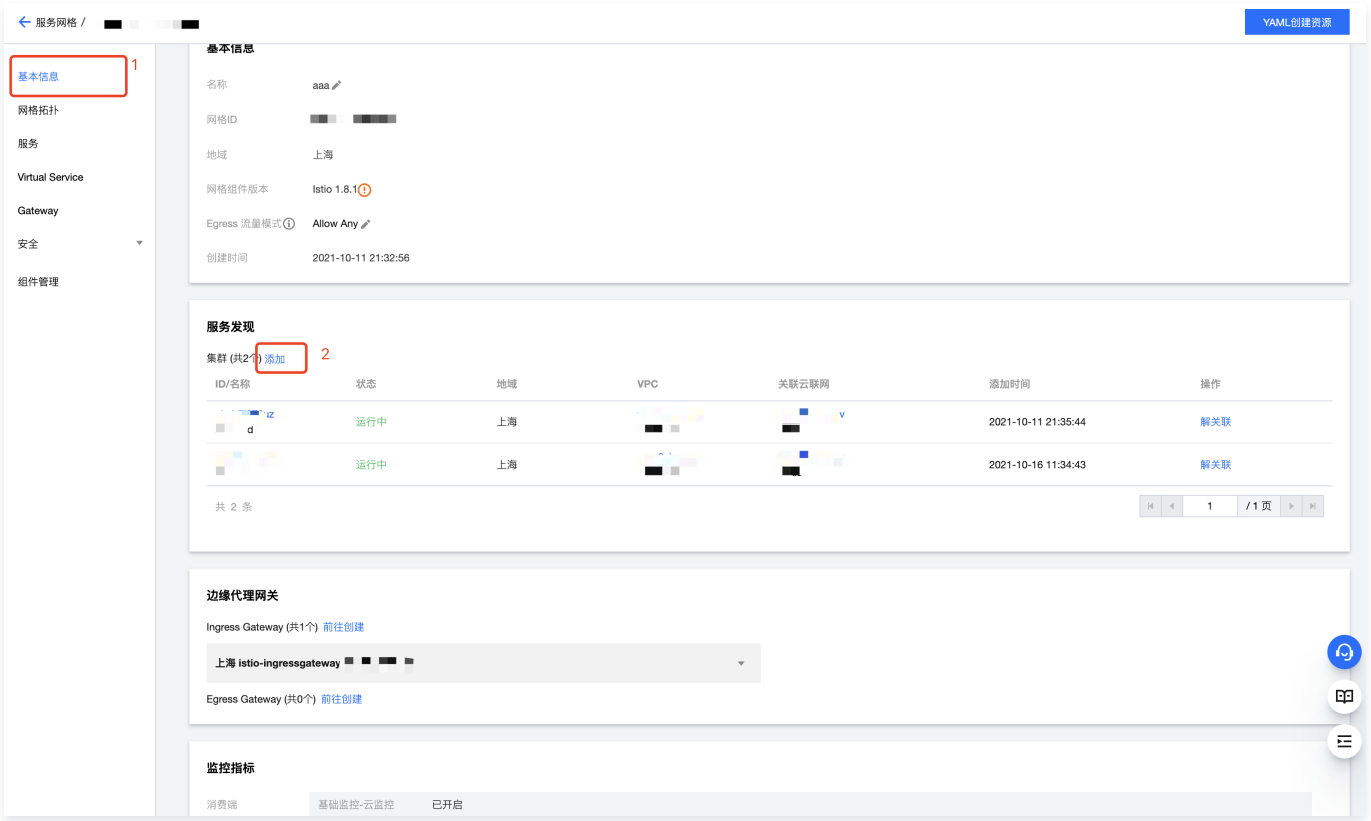


Mesh Details Page

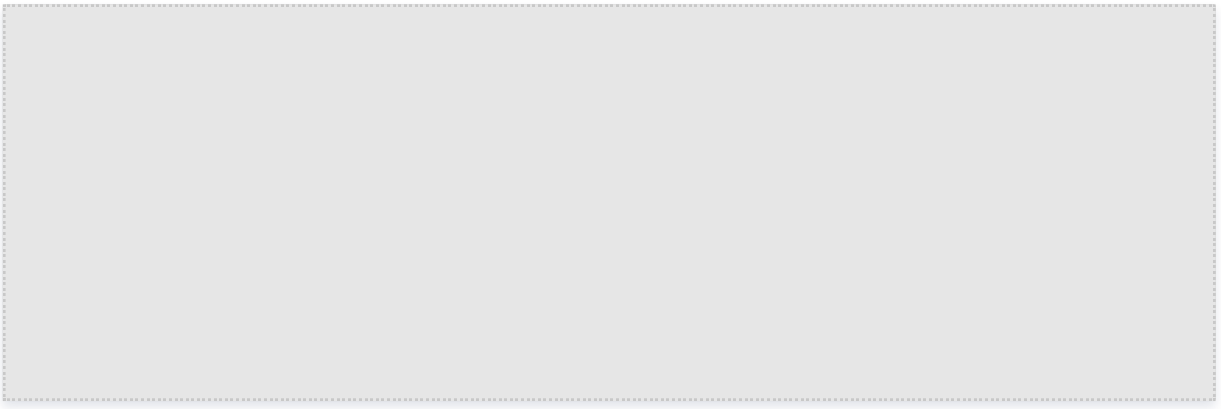
On the mesh details page, you can view the service discovery clusters associated with the current mesh instance, and add or disassociate an automatic service discovery cluster.

Adding a service discovery cluster

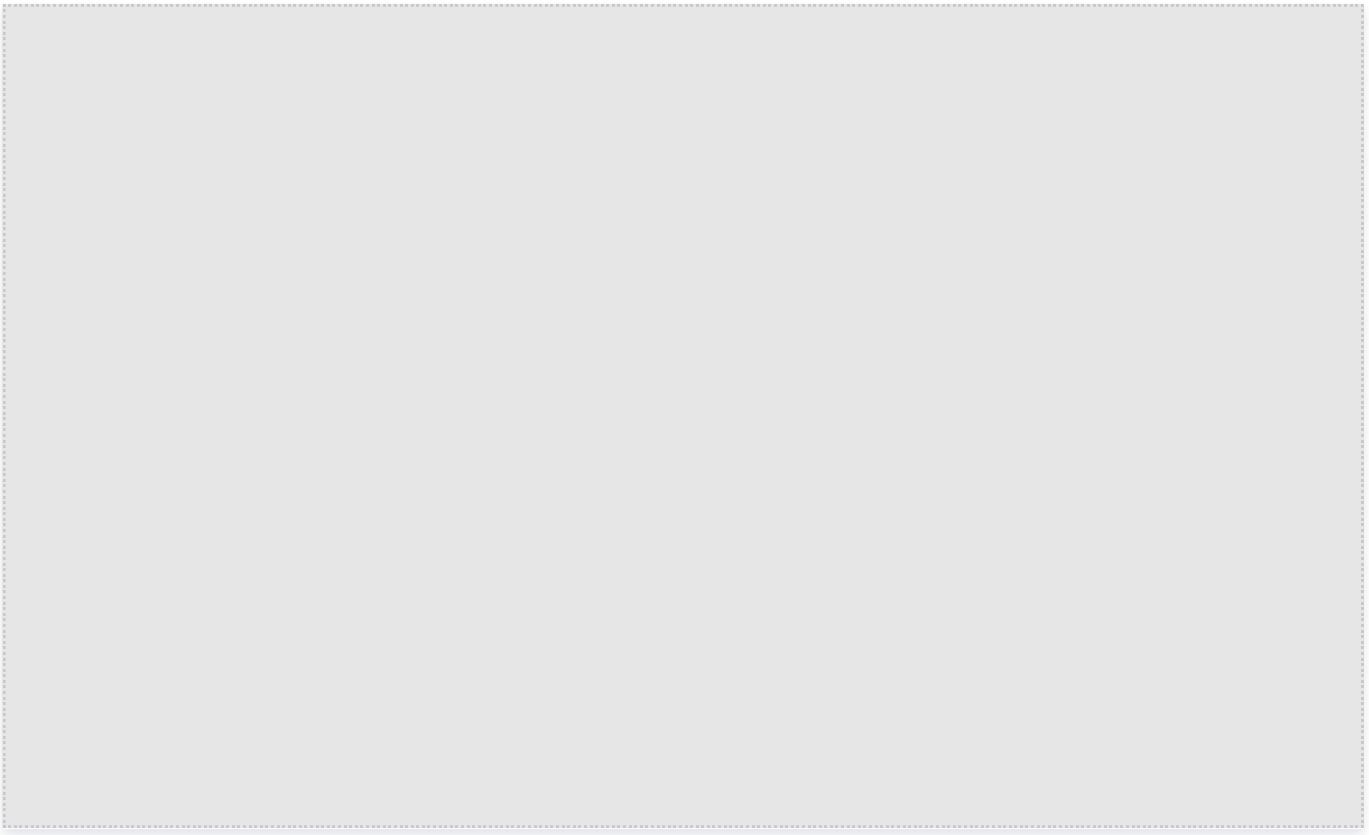
1. Go to the Mesh Details Page, click the sidebar **Basic Information**, and in the **service discovery** section, you can see the list of clusters with service discovery. Click **Add** to enter the **Add Service Discovery Cluster** popup. As shown below:



2. In the **Add Service Discovery Cluster** popup, select the Kubernetes clusters for automatic service discovery that you want to add. You can add multiple clusters at the same time. Click **Confirm**. As shown below:



3. After submitting the service discovery cluster adding request, wait for the cluster connection to complete to finish adding the service discovery cluster. As shown below:

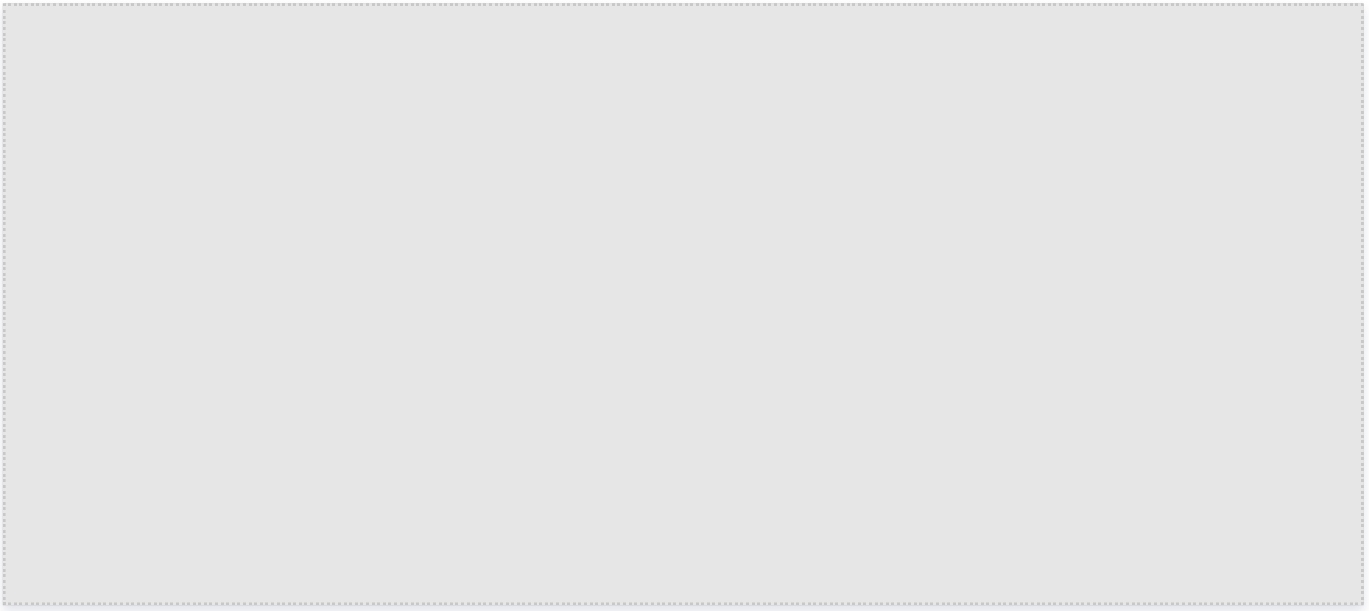


> **Note:**
> After the services are added to the mesh, you'll need to configure Sidecar injection in order to manage traffic, visualize observations, etc. Please refer to Mesh Configuration for related guidance.

Disassociating a service discovery cluster

If you need to disassociate a service discovery cluster that has been added to the mesh, you can follow these steps:

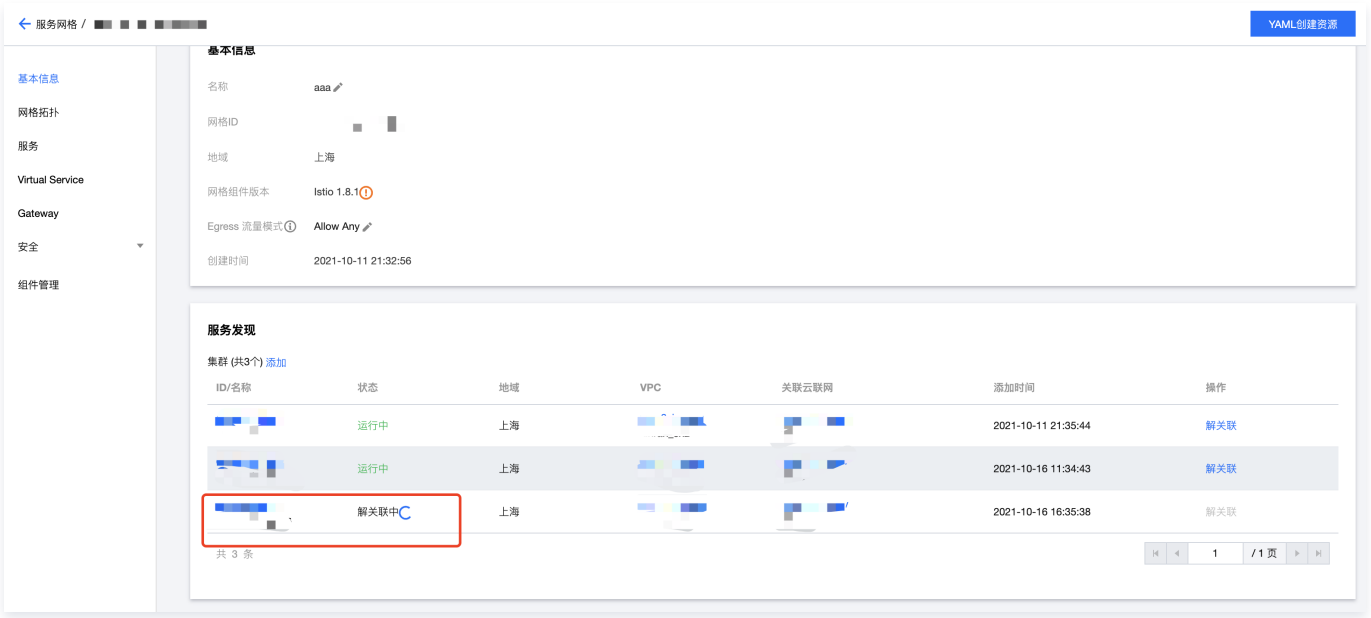
1. Enter the Grid Details page and click the sidebar **Basic Information**. In the **Service Discovery** module, you can view the current mesh service discovery cluster list. Select the cluster that needs to be disassociated and click **Disassociate** in the action column to enter the Confirm Disassociation popup. As shown below:



2. In the **Remove Cluster Association** popup, confirm the information about the service discovery cluster to be disassociated and click **Confirm** to submit the cluster disassociation request. Note that after the cluster is disassociated, the mesh will no longer be aware

of service instance changes in the cluster, and related service requests may encounter abnormalities. As shown below:

3. Wait for the disassociation operation to complete. As shown below:



Manual Service Discovery

Last updated: 2024-08-09 15:28:47

Overview

With Istio's Service Entry and Workload Entry mechanisms, you can add services in clusters that are not provided by TKE, such as traditional VM services and database services, on TCM. The Service Entry corresponds to the Service concept in K8S. Once a service is added to the mesh through Service Entry, it can be accessed by other automatically discovered services within the mesh according to routing rules. The Workload Entry corresponds to the K8S Workload concept, used to designate a set of service entity programs corresponding to the Service Entry. A Workload Entry with an installed sidecar can achieve capabilities such as traffic control, enhanced security, and service observation, just like other automatically discovered K8S workloads.

Description of Major ServiceEntry Fields

Field Name	Field Format	Field description
spec.hosts	string	Host name in the URL of a service. Multiple host names are allowed.
spec.ports	Port[]	Port number of the service. Multiple port numbers are allowed.
spec.resolution	string	• Static: Uses a static endpoint IP address as the service instance. • DNS: Resolves the service's endpoint IP address through DNS, which is often used for external services. The declared endpoint must use a DNS domain name, and in the absence of an endpoint, the service will resolve to the host's domain name. • NONE: Selected when the service does not require IP resolution.
spec.location	string	Used to indicate whether this service is within the mesh. Some Istio capabilities are not available for external services, such as mTLS for external services. MESH_EXTERNAL represents external services, MESH_INTERNAL represents services within the mesh.
spec.endpoints	String	The service's access point, multiple entries can be filled, but eventually, only one will be used at a time.

Description of Major WorkloadEntry Fields

Field Name	Field Format	Field description
spec.address	string	Address of the current endpoint. It is similar to a pod IP address.
spec.labels	string	Labels of the current endpoint. They are used to associate with the ServiceEntry.
sepc.serviceAccount	string	Permission information about a sidecar. This field must be specified when you need to add a sidecar for the endpoint.

For detailed information about ServiceEntry and WorkloadEntry, see Service Entry and Workload Entry.

Manually Registering a Service

YAML Configuration Example

Service Entry

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: external-svc-https
spec:
  hosts:
  - api.dropboxapi.com
  - www.googleapis.com
```

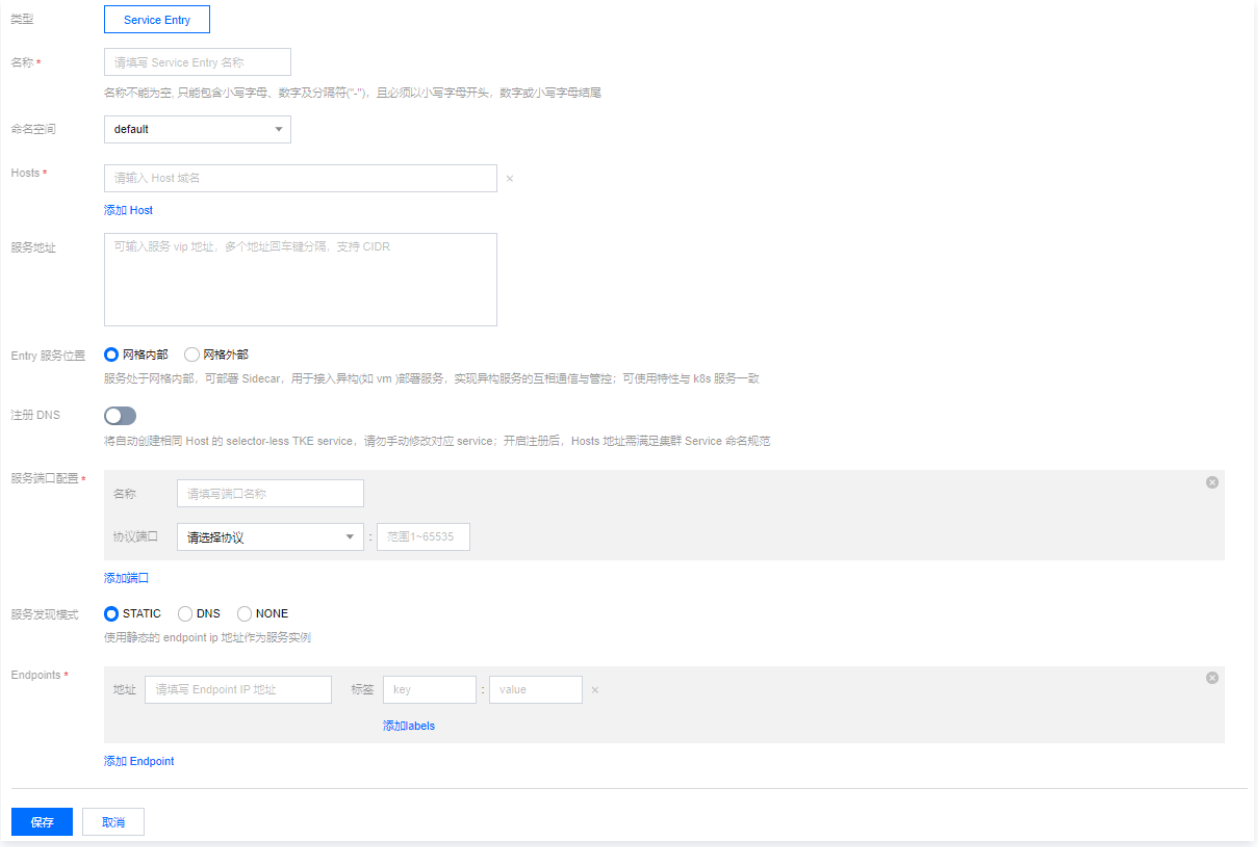
```
  - api.facebook.com
  location: MESH_EXTERNAL
  ports:
  - number: 443
    name: https
    protocol: TLS
  resolution: DNS
```

Workload Entry

```
apiVersion: networking.istio.io/v1alpha3
kind: WorkloadEntry
metadata:
  name: details-svc
spec:
  serviceAccount: details-legacy
  address: 2.2.2.2
  labels:
    app: details-legacy
    instance-id: vm1
```

Console Configuration Example

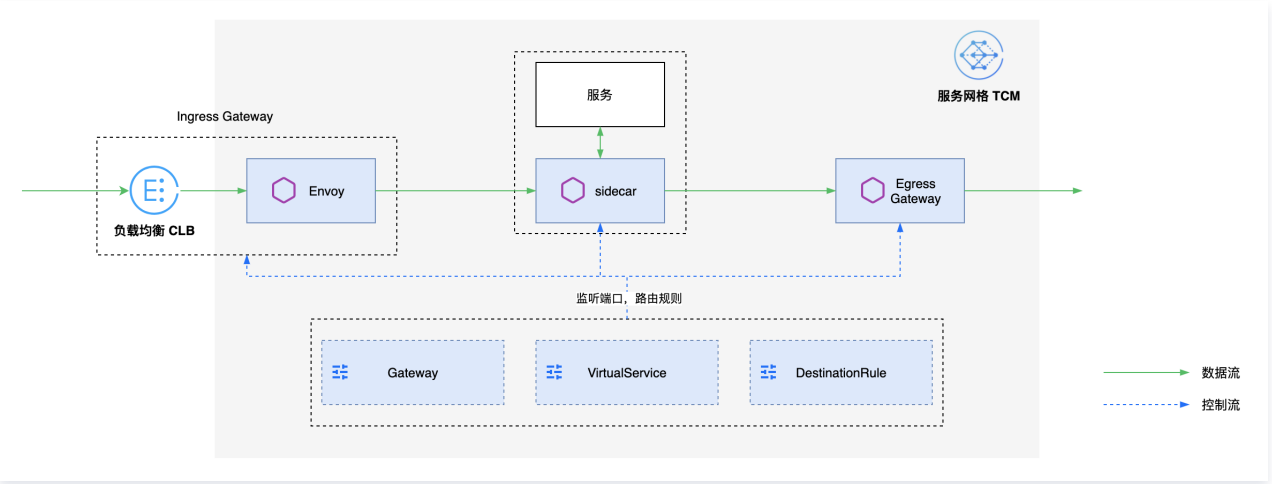
1. log in to TCM console.
2. Click **ID/Name** to enter the mesh details page.
3. Click **Service** > **Create**, and provide basic service information to register a non-containerized third-party service into the mesh service, as shown below:



Gateway
Gateway Management

Last updated: 2024-08-09 15:29:19

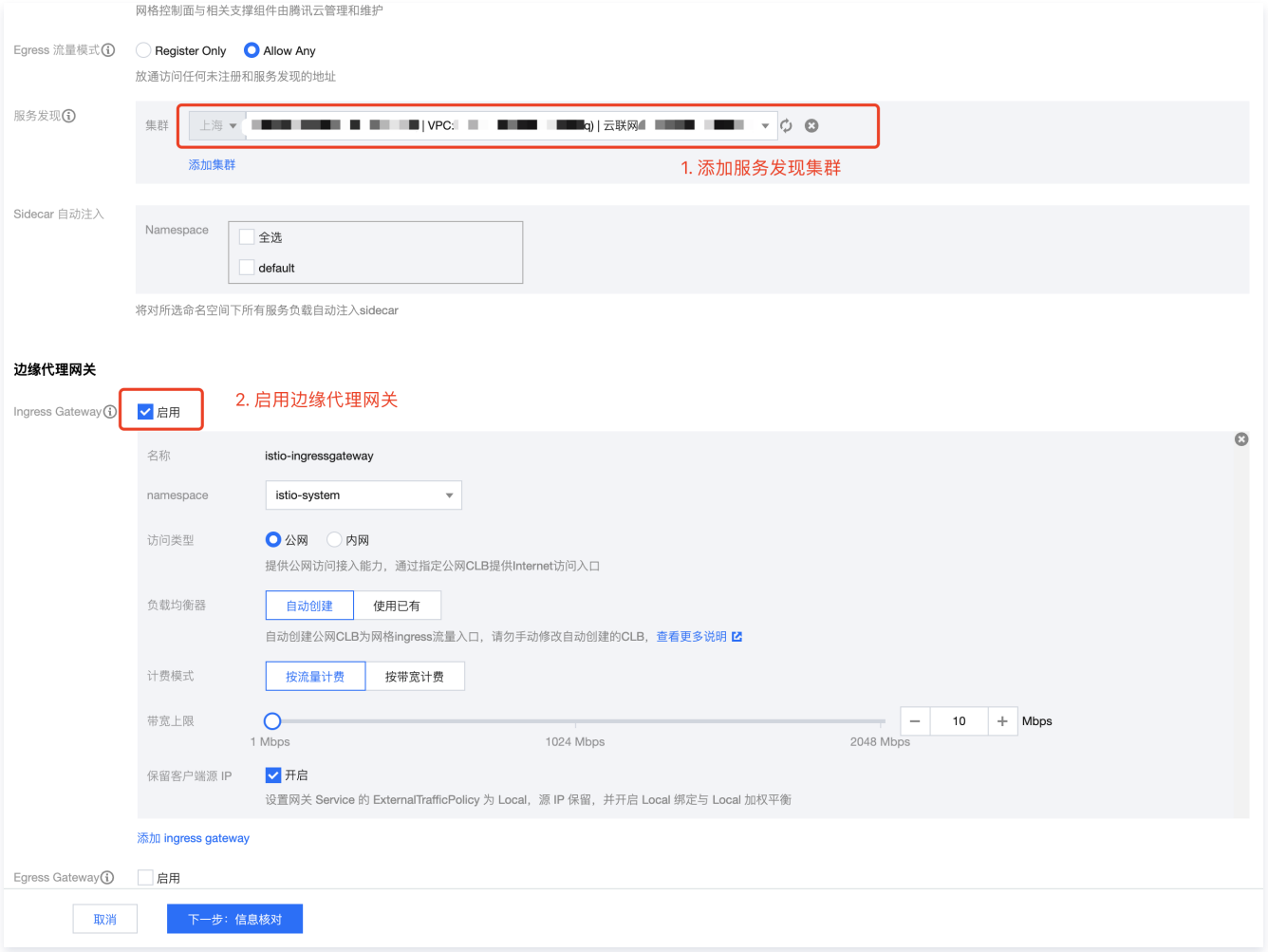
The Edge Proxy Gateway is responsible for the special data plane of mesh ingress and egress traffic CLB. It is deployed in your cluster as an independent Pod rather than in the form of a Sidecar, divided into Ingress Gateway and Egress Gateway types. Among them, an Ingress Gateway instance includes the data plane's Envoy Pod and its associated CLB instance (public or private network). TCM provides a managed Edge Proxy Gateway Controller, which has implemented automatic integration of Ingress Gateway configuration with CLB. You can configure the Ingress Gateway through Istio CRD, and TCM will automatically sync the related settings to the associated CLB instance. The synced configuration includes port configuration and enhanced feature's port listening rule configuration. Thus, the Envoy container and associated CLB act as a whole, providing you with ingress edge proxy gateway capabilities.
If you need the capability to balance the ingress and egress traffic of the mesh, you need to create an ingress gateway or egress gateway instance, and then configure listening rules and traffic management (routing) rules of the gateway by using Istio CRDs such as Gateway, VirtualService, and DestinationRule. The listening rules are configured by using the Gateway CRD, and the traffic management rules are configured by using the VirtualService and DestinationRule CRDs (consistent with the east-west traffic management syntax). The following figure is a schematic diagram of the relationship between gateway instances and Istio CRD configurations.



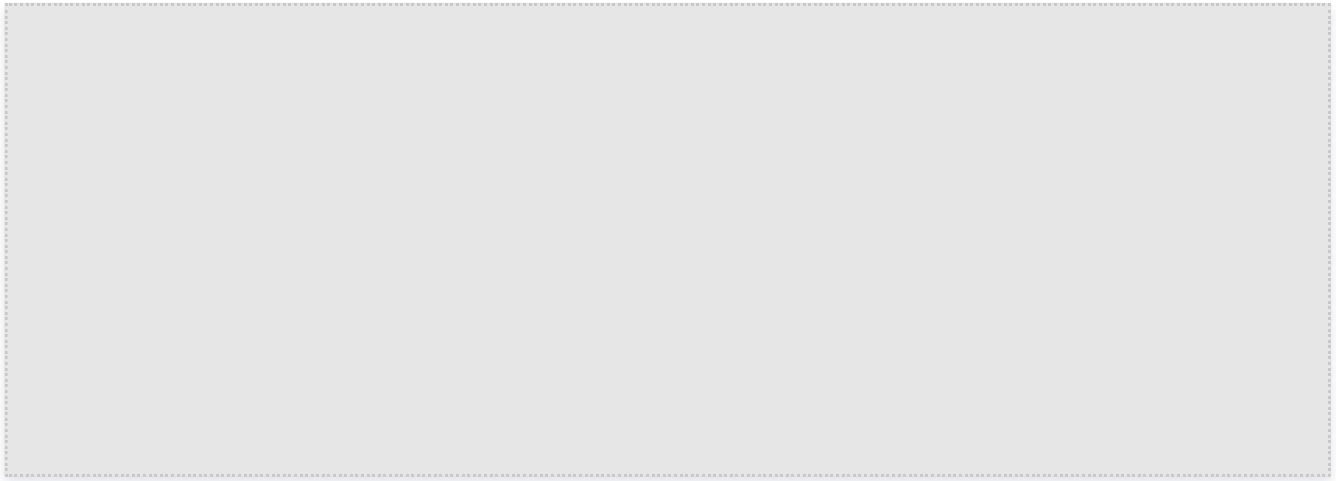
Creating a Gateway

1. log in to TCM console.

2. You can create an Edge Proxy Gateway after adding a service discovery cluster on the mesh creation page, as shown below:



or click **New** to create an Edge Proxy Gateway on the Edge Proxy Gateway tab in the mesh details page, as shown below:



Explanation of important configuration items for creating a gateway:

Configuration Item	Description
Type	Whether an ingress gateway or an egress gateway is to be created.
Access Cluster	Kubernetes cluster in which the gateway is to be created.
namespace	Namespace in which the gateway is to be created.
Access type	Ingress gateway parameter. Select the CLB access type, supporting public and private networks.

Load Balancer	Ingress Gateway Parameters. Choose to automatically create a CLB, or reuse an existing CLB. For more information on reusing an existing CLB, see Service Using Existing CLB.
Billing mode	Ingress Gateway Parameters. Choose the CLB's billing mode, which supports billing by traffic or by bandwidth. For more information on CLB billing, see CLB Billing Overview.
Bandwidth cap	Ingress gateway parameter. Select a CLB bandwidth cap, which ranges from 0 to 2048 Mbps.
CLB-to-Pod direct access	Ingress Gateway Parameters. For example, if the gateway's access cluster network mode is VPC-CNI, you can enable CLB to directly connect to Pods, eliminating the need for NodePort forwarding and achieving higher performance. It supports client source IP preservation, pod-level session maintenance, and health checks. For more details, see Using LoadBalancer to Directly Connect to Pod Mode Service.
Preserve client source IP	Ingress Gateway Parameters. Set the gateway Service's ExternalTrafficPolicy to Local, to preserve the source IP and enable local binding and local load balancing. When CLB direct connection to Pod is enabled, this parameter becomes invalid. For more details, see Service Backend Selection.
Component Configurations	Configurations about CPU and memory resources and HPA policies of the gateway.

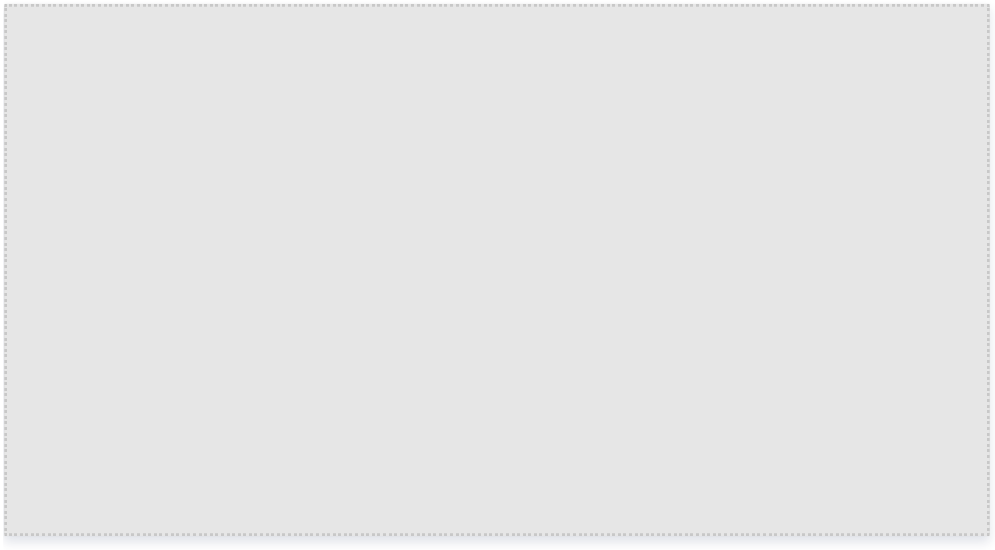
Gateway Deployment Modes

The Edge Proxy Gateway has two deployment forms: **Normal Mode** and **Dedicated Mode**. Among them:

- **Normal Mode**: Gateway services are deployed in the selected business cluster, indistinguishably from other business Pods.
- **Dedicated Mode**: In some scenarios, to enhance the stability of services, you need to deploy the Edge Proxy Gateway on dedicated nodes. Dedicated mode requires selecting some cluster nodes to be added to the dedicated resource pool first. The mesh will set taints on the selected nodes to ensure exclusivity. Once set, all Ingress/Egress Gateways will only be deployed on the selected nodes. You can also specify nodes for a specific Gateway in advanced settings.

You can readjust the deployment mode of the gateway on the mesh creation page or the component management page:

> **Note:**
> Adjusting the deployment mode will trigger gateway service scheduling, which may adversely affect service traffic.



Updating Gateway Configurations

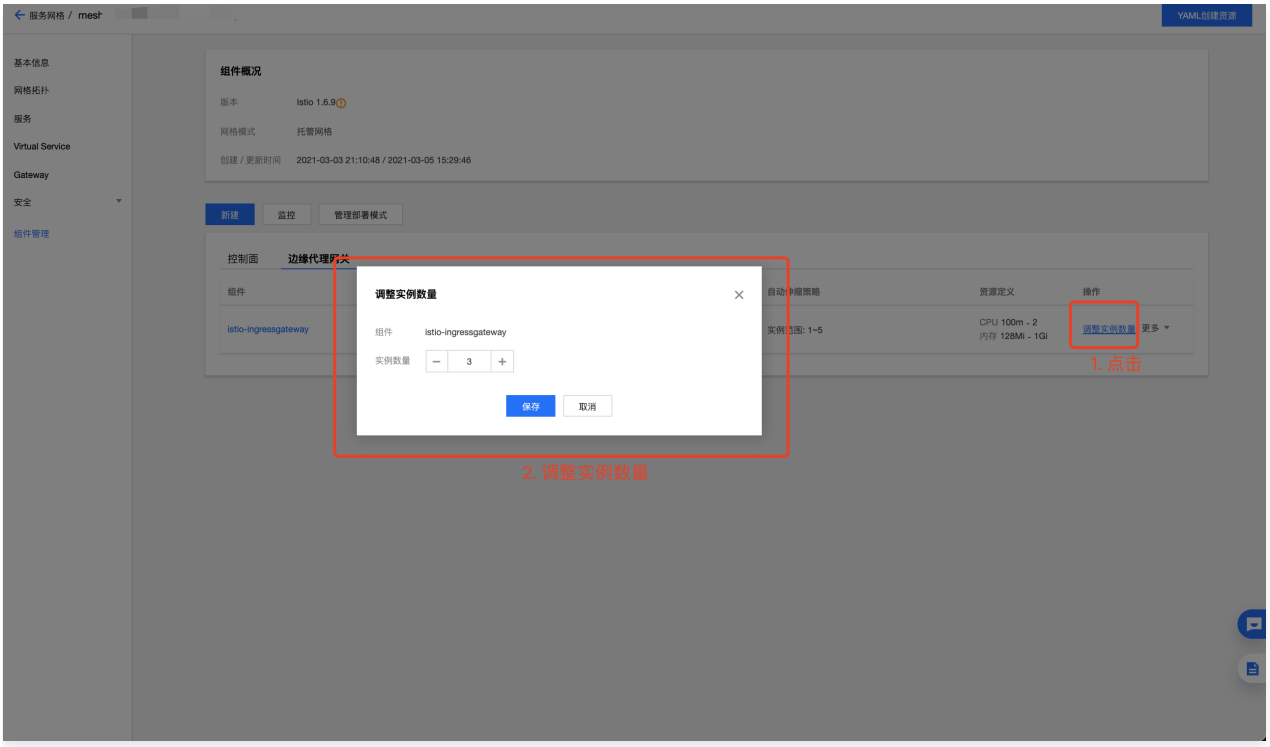
After a gateway is created, you can modify the associated CLB bandwidth (supported only for an ingress gateway), the number of instances, HPA policies, and resource definitions of the gateway.

Modifying the CLB Bandwidth

You can modify the bandwidth of the CLB instance associated with the Ingress Gateway. The binding CLB configuration for the Ingress Gateway can be edited in the **Basic Information** Tab Edge Proxy Gateway section of the corresponding mesh detail page.

Modifying the component instance quantity

In **Grid Details > Component Management**, you can adjust the number of instances for a component.

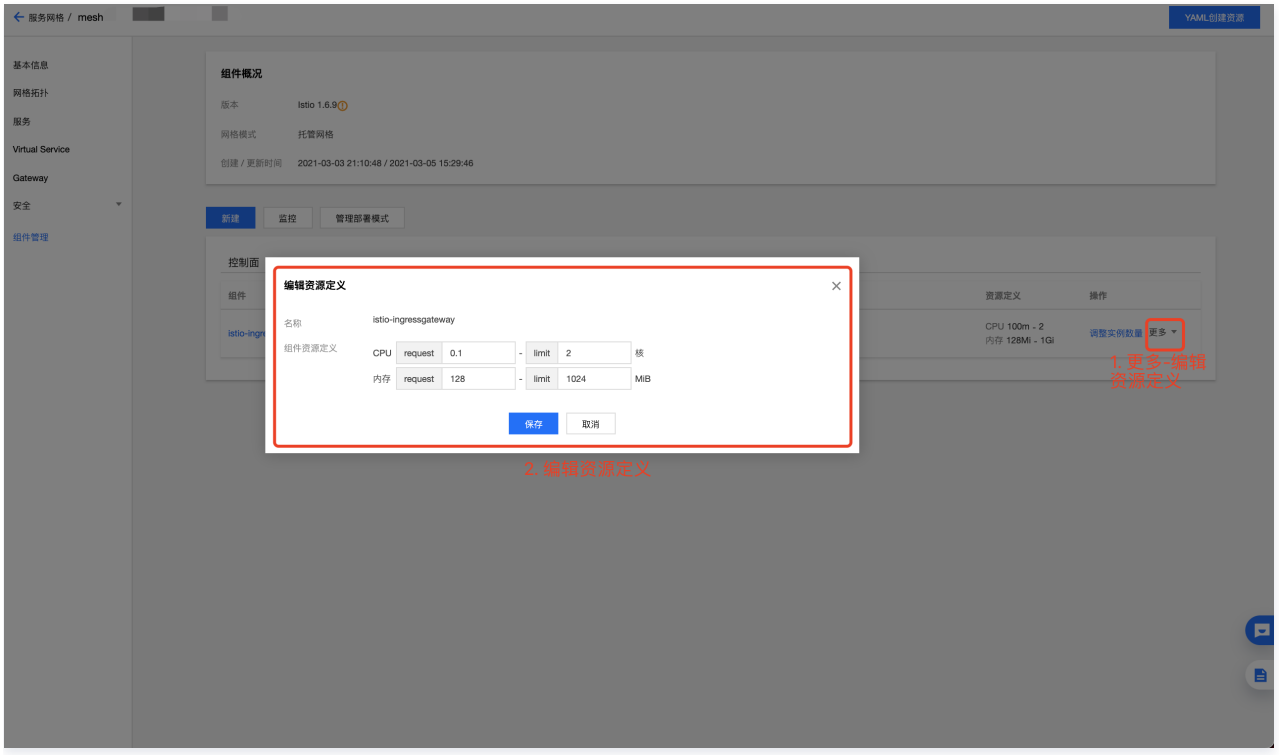


Modifying HPA Policies of Components

In **Grid Details > Component Management**, you can edit the HPA policies for a component. You can configure scaling policies based on CPU, memory, network, and hard disk indicators. As shown below:

Modifying Component Resource Definitions

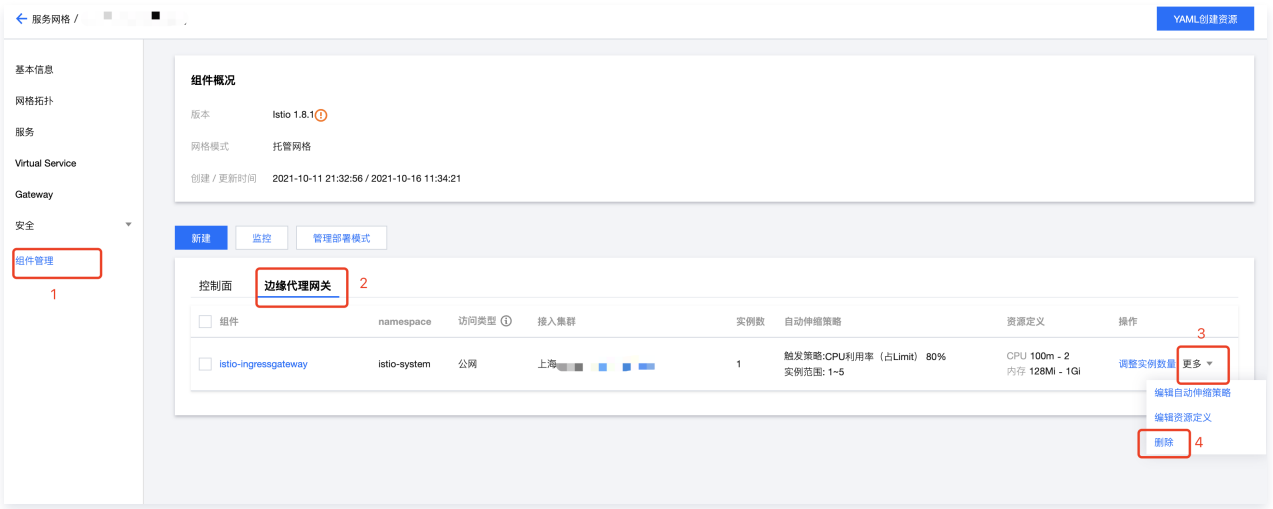
In **Grid Details > Component Management**, you can edit the resource definitions of a component, including the request and limit values for CPU and memory. As shown below:



Deleting a Gateway

You can delete a specified edge proxy gateway in Grid Details – Component Management – Edge Proxy Gateway. The steps are as follows:

1. Access the Grid Details page, click **Component Management**, choose **Edge Proxy Gateway**, and click **Operations** > **More** > **Delete** in the Operation column where the gateway to be deleted resides. As shown below:

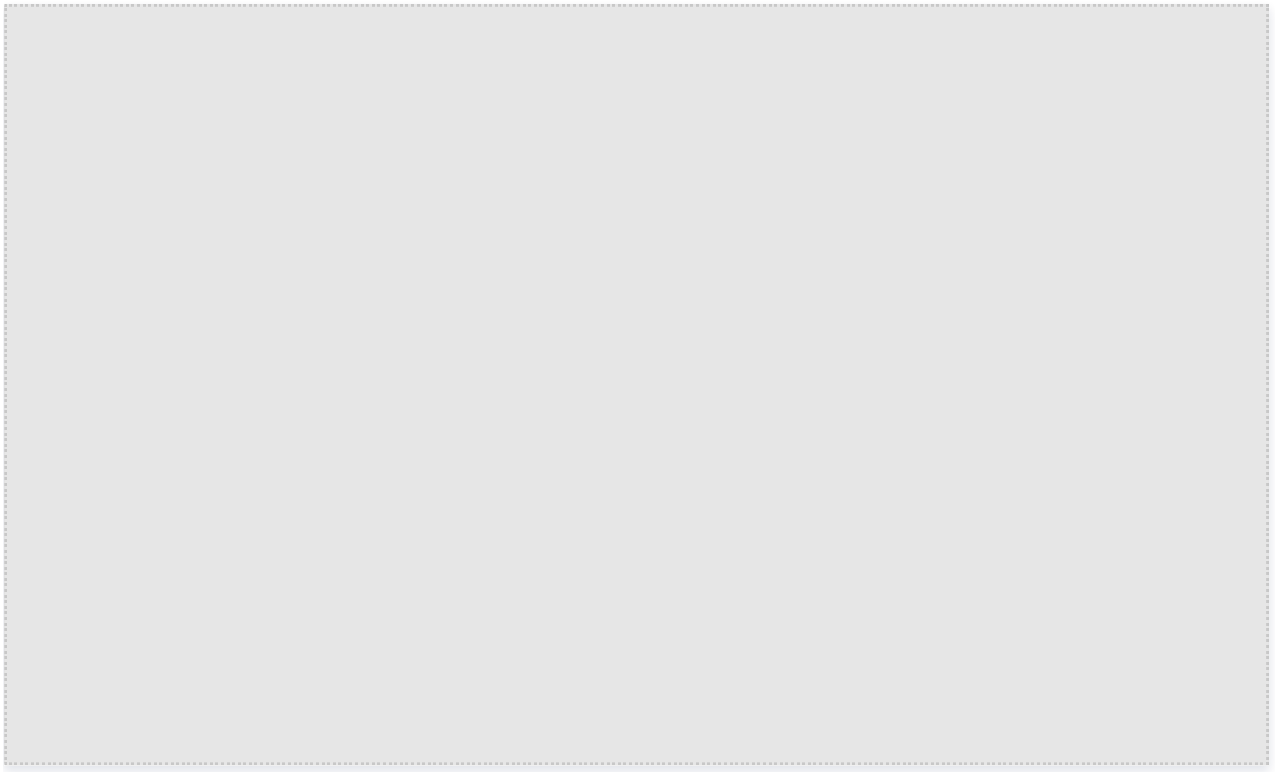


2. In the **Delete Edge Proxy Gateway** dialog box, confirm the name of the gateway to be deleted, and click **OK** to proceed. As shown below:

The TCM gateway controller enables automatic integration with CLB

Tencent Cloud Mesh implements the managed gateway controller. The controller monitors the gateway configurations delivered to an ingress gateway in real time, parses the current port configurations, and synchronizes the current port configurations to CLB, so that you no longer need to manually configure CLB ports. CLB ports, ingress gateway service ports, and ingress gateway container ports are in one-to-one mapping. To be specific, if the `80` port is defined in the Gateway CRD, the gateway controller of Tencent Cloud Mesh will configure the container port as `80` and the service port as `80` for the ingress gateway instance and enables the `80` port of the associated CLB synchronously.

The gateway controller of Tencent Cloud Mesh also implements the feature of enabling SSL certificate offloading to take place at CLB. In this way, after certificate offloading takes place at CLB, the ingress gateway provides traffic management capabilities. After this feature is configured on the gateway, the gateway controller will resolve the port, domain name, and certificate that are involved in feature configurations, and synchronize the configurations to the CLB instance bound to the ingress gateway.



Gateway Configuration

Last updated: 2024-08-09 15:29:44

Port of the Edge Proxy Gateway. Listening rules are configured through the Gateway CRD. Below is an example of a Gateway configuration with explanations of important fields provided through comments:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: gateway-sample
  namespace: defualt
spec:
  selector: # Match the Gateway configuration based on the provided Tag for the Pod
    istio: ingressgateway
    app: istio-ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - uk.bookinfo.com
    - eu.bookinfo.com
    tls:
      httpsRedirect: true # Send 301 HTTPS redirect
  - port:
      number: 443
      name: https-443
      protocol: HTTPS # Enable port HTTPS
    hosts:
    - uk.bookinfo.com
    - eu.bookinfo.com
    tls:
      mode: SIMPLE # TLS simple authentication
      serverCertificate: /etc/certs/servercert.pem # Load certificate using file mount method
      privateKey: /etc/certs/privatekey.pem
  - port:
      number: 9443
      name: https-9443
      protocol: HTTPS # Enable port HTTPS
    hosts:
    - "bookinfo-namespace/*.bookinfo.com"
    tls:
      mode: SIMPLE # TLS simple authentication
      credentialName: bookinfo-secret # Load certificate from Kubernetes secret using SDS method
  - port:
      number: 5443
      name: https-ssl
      protocol: HTTPS # Enable port HTTPS
    hosts:
    - "*"
    tls:
      mode: SIMPLE # TLS simple authentication
      credentialName: qcloud-abcdABCD # Load certificate from Tencent Cloud SSL Platform with certificate ID
abcdABCD using SDS method
  - port:
      number: 6443
      name: clb-https-6443-ABCDabcd # Enable certificate unpacking at port 6443 in CLB using SSL platform
certificate with ID ABCDabcd
      protocol: HTTP
```

```
    hosts:
    - "tcm.tencent.com"
```

Explanation of Gateway Configuration Fields

Below is an explanation of important fields in the Gateway CRD:

Field Name	Field Type	Field description
`metadata.name`	`string`	Gateway Name.
`metadata.namespace`	`string`	Gateway Namespace.
`spec.selector`	`map<string, string>`	Gateway uses the filled Tag key-value pair to match and configure the edge proxy gateway instance.
`spec.servers.port.number`	`uint32`	Port.
`spec.servers.port.protocol`	`string`	Communication protocol, supported: `HTTP, HTTPS, GRPC, HTTP2, MONGO, TCP, TLS`. Note that protocol configuration for the same port on the same gateway must be consistent.
`spec.servers.port.name`	`string`	Port Name. TCM currently implements the feature of specifying SSL Certificates unpacking to CLB by port name. If you need to configure certificate unpacking, name it as `clb-https-{port}-{ssl platform certificate ID}`. The certificate unpacking feature only takes effect when the communication protocol of the current port is specified as HTTP. The edge proxy gateway controller will automatically create a CLB Layer 7 listener for certificate unpacking. After CLB SSL unpacking is completed, the CLB instance and the Ingress Gateway Pod will communicate in plaintext. Note that the certificate unpacking configuration for the same port on the same gateway must be consistent to avoid configuration conflicts.
`spec.severs.hosts`	`string[]`	Domain name, supporting wildcard `*`.
`spec.servers.tls.httpsRedirect`	`bool`	When the value is `true`, the edge proxy gateway will return a 301 redirect for all HTTP requests, requiring the client to initiate an HTTPS request.
`spec.servers.tls.mode`	-	Configure the TLS security authentication mode for the current port. If you need to enable security authentication for the current port, you need to fill it in. Supported: `PASSTHROUGH, SIMPLE, MUTUAL, AUTO_PASSTHROUGH, ISTIO_MUTUAL`
`spec.servers.tls.credentialName`	`string`	Configure the name of the secret for discovering the TLS certificate key. Supports loading certificates and keys from the Kubernetes secret in the same namespace as the Ingress Gateway instance. You need to ensure that the secret you fill in contains the appropriate certificates and keys. TCM also implements a feature of loading Tencent Cloud SSL platform certificates. Fill in this field in the format of `qcloud-{ssl platform certificate ID}`, and the TCM edge proxy gateway controller will load the SSL platform certificates for the edge proxy gateway. Currently, only server certificates and private keys for one-way authentication SIMPLE mode from the SSL platform are supported.
`spec.servers.tls.serverCertificate`	`string`	Set the certificate path field to be filled in when mounting the TLS certificate key for the port via file mount (not recommended, it is recommended to use the `credentialName` field to load the certificate private key). Istio by default uses the istio-ingressgateway-certs secret under the namespace where the gateway is located to load certificates to the path `/etc/istio/ingressgateway-certs`.

`spec.servers.tls.privateKey`	`string`	Set the private key path field to be filled in when mounting the TLS certificate key for the port via file mount (not recommended, it is recommended to use the `credentialName` field to load the certificate private key). Istio by default uses the istio-ingressgateway-certs secret under the namespace where the gateway is located to load private keys to the path `/etc/istio/ingressgateway-certs`.
`spec.servers.tls.caCertificates`	`string`	Set the root certificate path field to be filled in when mounting the TLS certificate key for the port via file mount (not recommended, it is recommended to use the `credentialName` field to load the certificate private key). Istio by default uses the istio-ingressgateway-ca-certs secret under the namespace where the gateway is located to load root certificates to the path `/etc/istio/ingressgateway-ca-certs`. Root certificates need to be configured for two-way authentication.

Sample code

Example of loading certificates from Kubernetes Secret to edge proxy gateway configuration

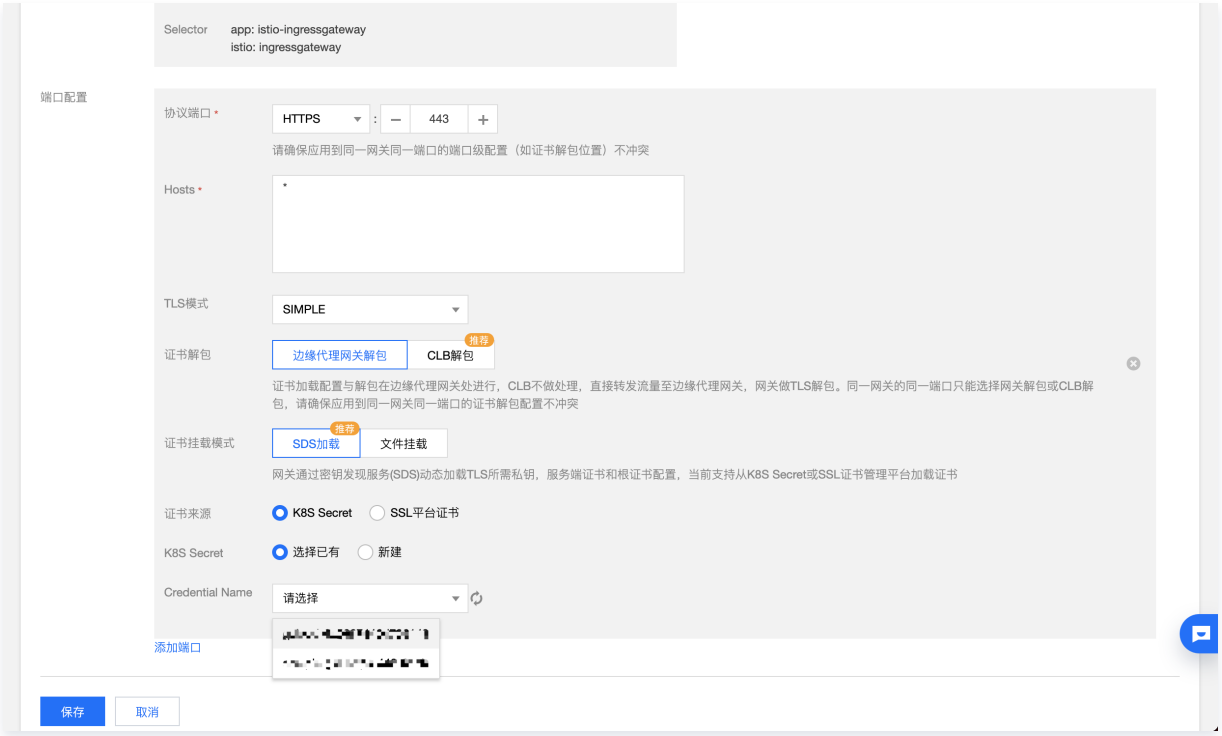
YAML Configuration Example

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: sample-gw
  namespace: default
spec:
  servers:
    - port:
        number: 443
        name: HTTPS-443-6cph
        protocol: HTTPS
      hosts:
        - '*'
      tls:
        mode: SIMPLE
        credentialName: {kubernetes secret name}
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
```

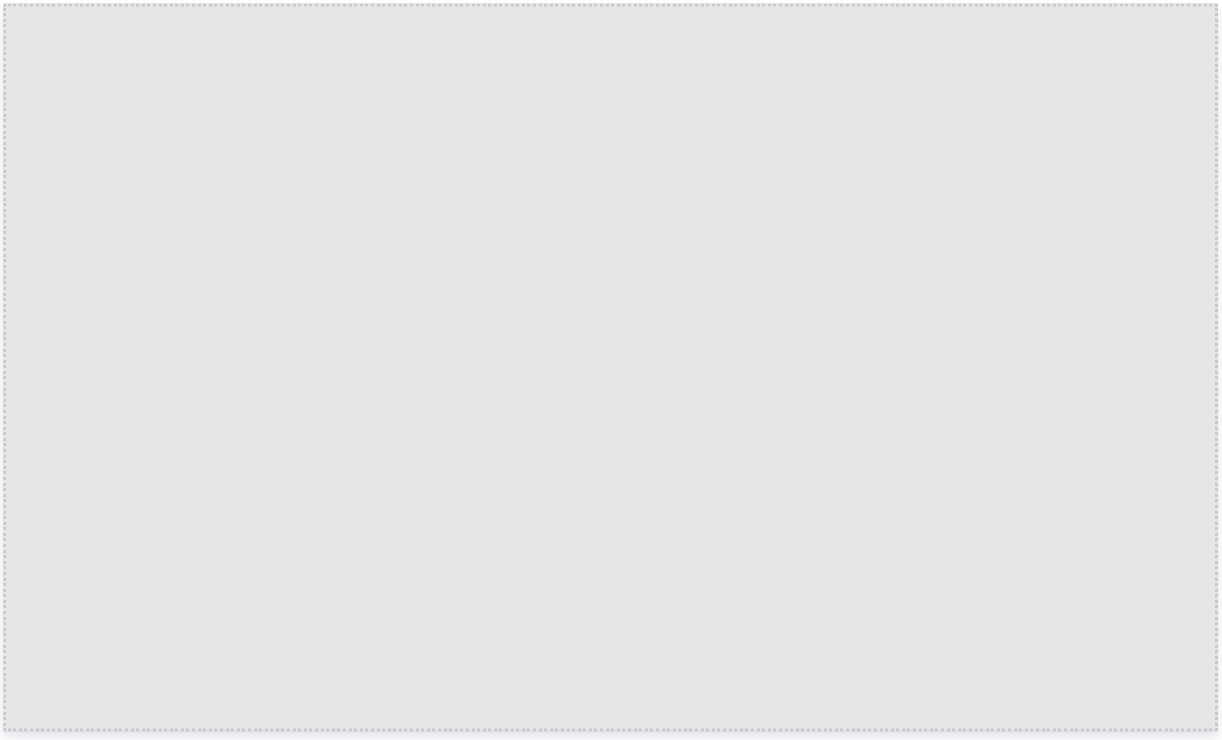
Console Configuration Example

The process for configuring an Ingress Gateway with HTTPS protocol and SSL Certificates loaded from Kubernetes secret (one-way authentication) in the Console is as follows:

1. Select **HTTPS** as the protocol, and **SIMPLE** as the TLS mode.
2. Select **Edge Proxy Gateway Unpacking** for certificate unpacking.
3. Select **SDS Loading** for the certificate mount mode.
4. Select **K8S Secret** as the certificate source.
5. For K8S Secret, select **Select Existing**, and choose the Secret under the namespace where the current Edge Proxy Gateway is located. Please ensure that the selected Secret contains the appropriate Certificate/Private Key/Root Certificate.



6. If the current Secret does not have an appropriate certificate, you can choose to **Create New** K8S Secret and copy the appropriate Certificate/Private Key/Root Certificate content to the corresponding input fields.



Example of loading certificates from the SSL Platform to Edge Proxy Gateway configuration

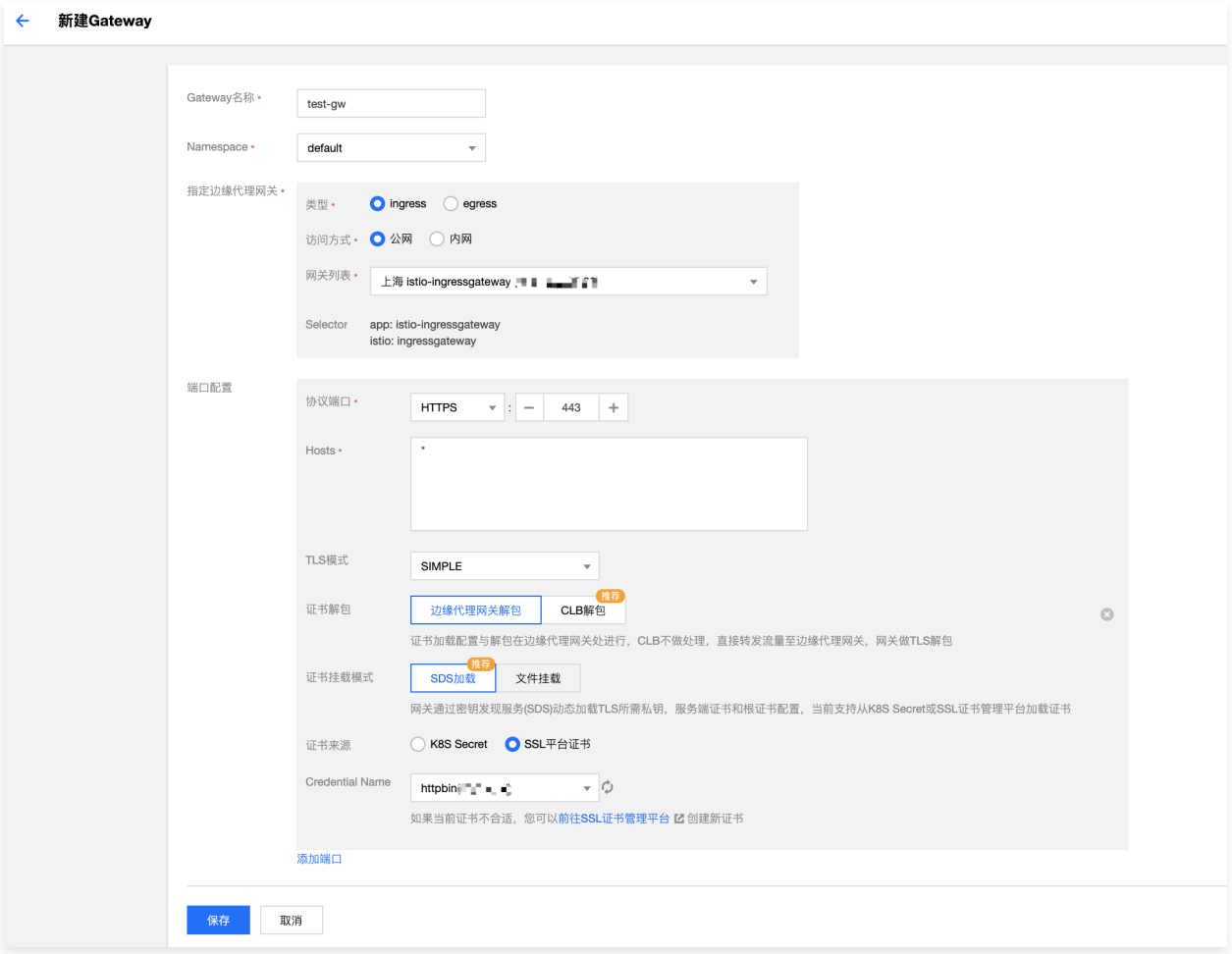
YAML Configuration Example

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
```

```
  name: test-gw
spec:
  servers:
    - port:
        number: 443
        name: HTTPS-443-9ufr
        protocol: HTTPS
      hosts:
        - '*'
      tls:
        mode: SIMPLE
        credentialName: qcloud-{Certificate ID}
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
```

Console Configuration Example

Besides configuring through YAML files, you can also create a Gateway configuration on the Console via the UI. The following is an example of configuring to load certificates from the SSL Platform to the Edge Proxy Gateway. Select **SSL Platform Certificate** as the certificate source to choose the SSL platform certificate you need to load.



Example of SSL Certificates Unpacking Upward to CLB configuration

YAML Configuration Example

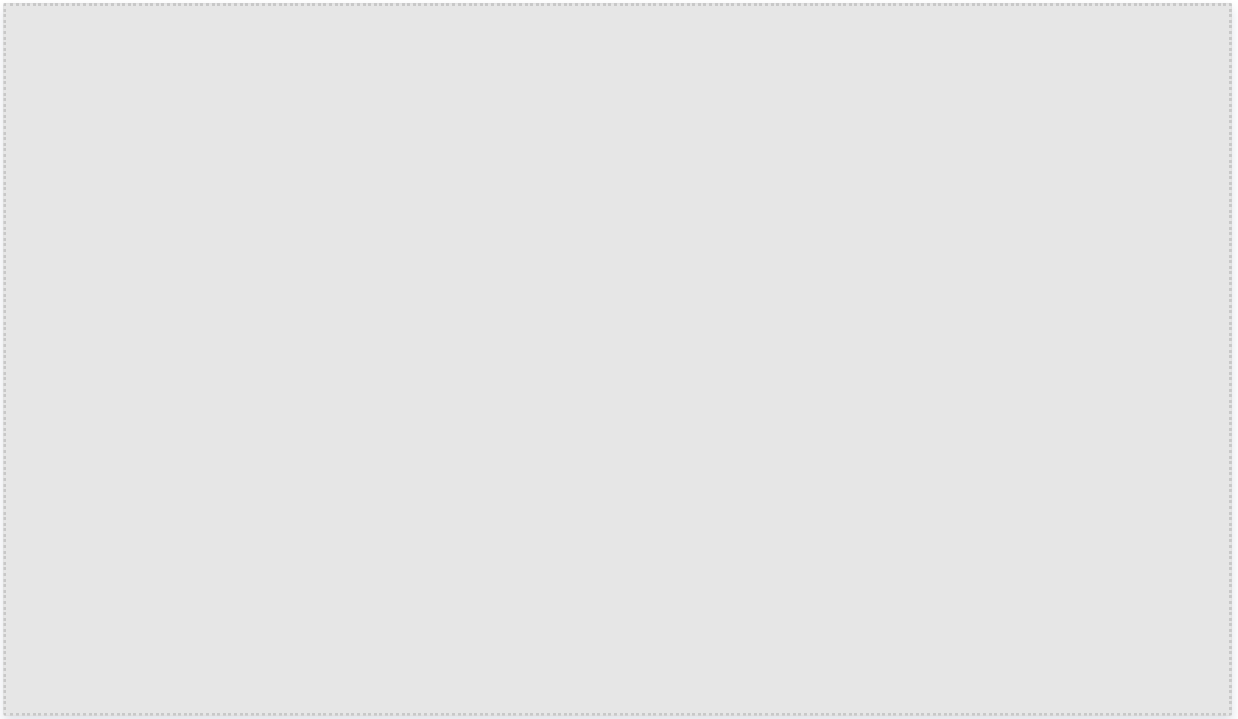
The following is the configuration to unpack the certification upward to CLB for port 443 and enable SNI for that port. Domain name `sample.hosta.org` uses Certificate 1, whereas domain name `sample.hostb.org` uses Certificate 2.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: test-gw
spec:
  servers:
    - port:
        number: 443
        name: clb-https-443-{Certificate ID 1}
        protocol: HTTP
      hosts:
        - sample.hosta.org
    - port:
        number: 443
        name: clb-https-443-{Certificate ID 2}
        protocol: HTTP
      hosts:
        - sample.hostb.org
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
```

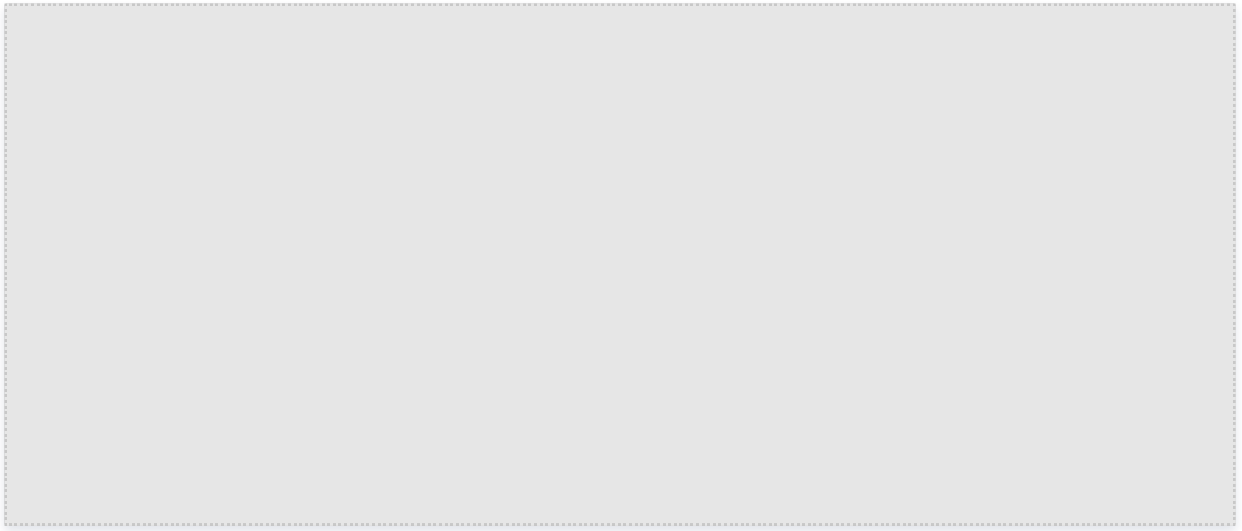
Console Configuration Example

The process for creating a Gateway configuration using the certificate unpacking feature in the Console UI is as follows:

1. Select the protocol as HTTPS. The option for **TLS Mode** will appear.
2. Choose **TLS Mode** as **SIMPLE**.
3. Select **Certificate Unpacking** as **CLB Unpacking**. At this point, the port protocol will automatically change to HTTP (after choosing certificate unpacking, the gateway processes the traffic as plaintext HTTP).
4. Choose the appropriate **Server Certificate**.



5. Upon successful creation, it will redirect to the details page of the created Gateway CRD:

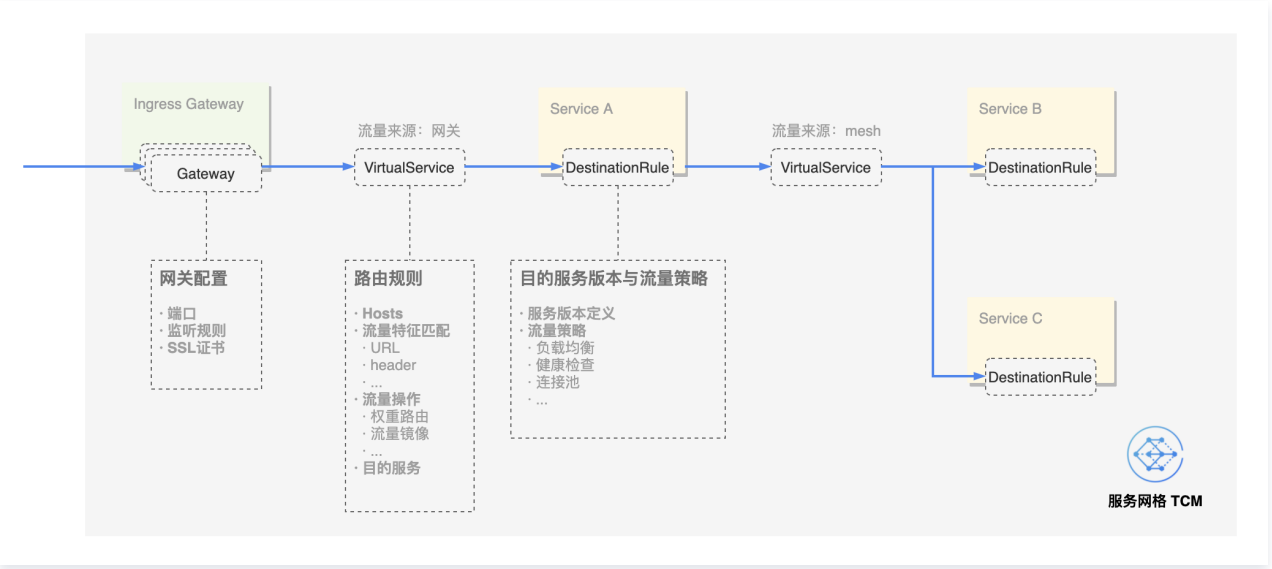


Traffic Management Overview

Last updated: 2024-08-09 15:30:11

Traffic Management Model of Tencent Cloud Mesh

Tencent Cloud Mesh is fully compatible with Istio's native traffic management CRDs Gateway, VirtualService, and DestinationRule, and presents the native traffic management syntax as a product. The following figure shows the traffic management model of Tencent Cloud Mesh:



Tencent Cloud Mesh uses Gateway, VirtualService, and DestinationRule to manage traffic.

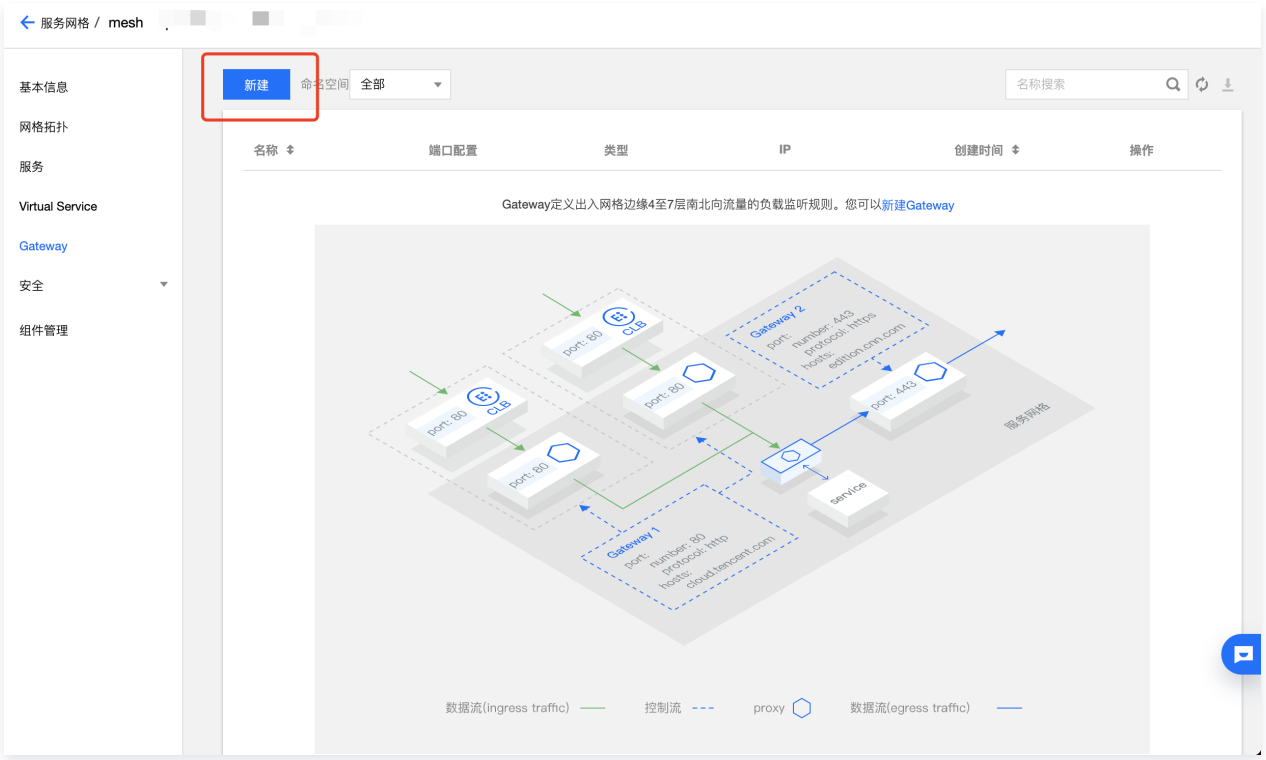
- Gateway: defines the port, listening rule, and certificate configurations of a gateway. Gateways and gateway configurations are in a one-to-many relationship. The Gateway specifies a gateway to which the configurations are to be delivered through the selector field.
- VirtualService: defines routing rules and traffic operation rules for a specified host. The VirtualService specifies a bound domain name through the hosts field. It can specify that traffic comes from a gateway or an internal component of a mesh.
- DestinationRule: defines versions and traffic policies of a service. The traffic policies include load balancing, health check, and connection pools. Services and DestinationRules are in a one-to-one binding relationship.

Traffic Management Configuration Methods

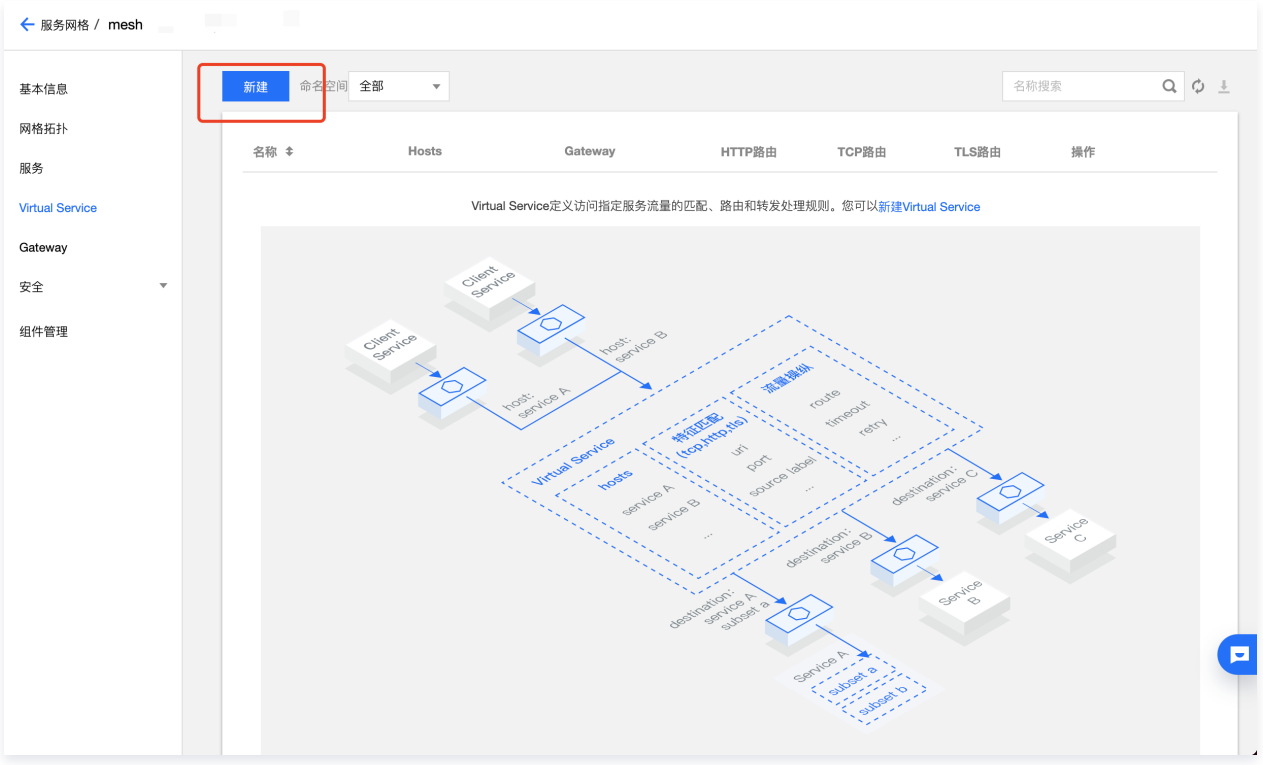
Currently, TCM provides the following two methods to configure Gateway, VirtualService, and DestinationRule:

Console UI Configuration
You can use the Console UI to create, delete, update, and view Gateways, VirtualServices, and DestinationRules.

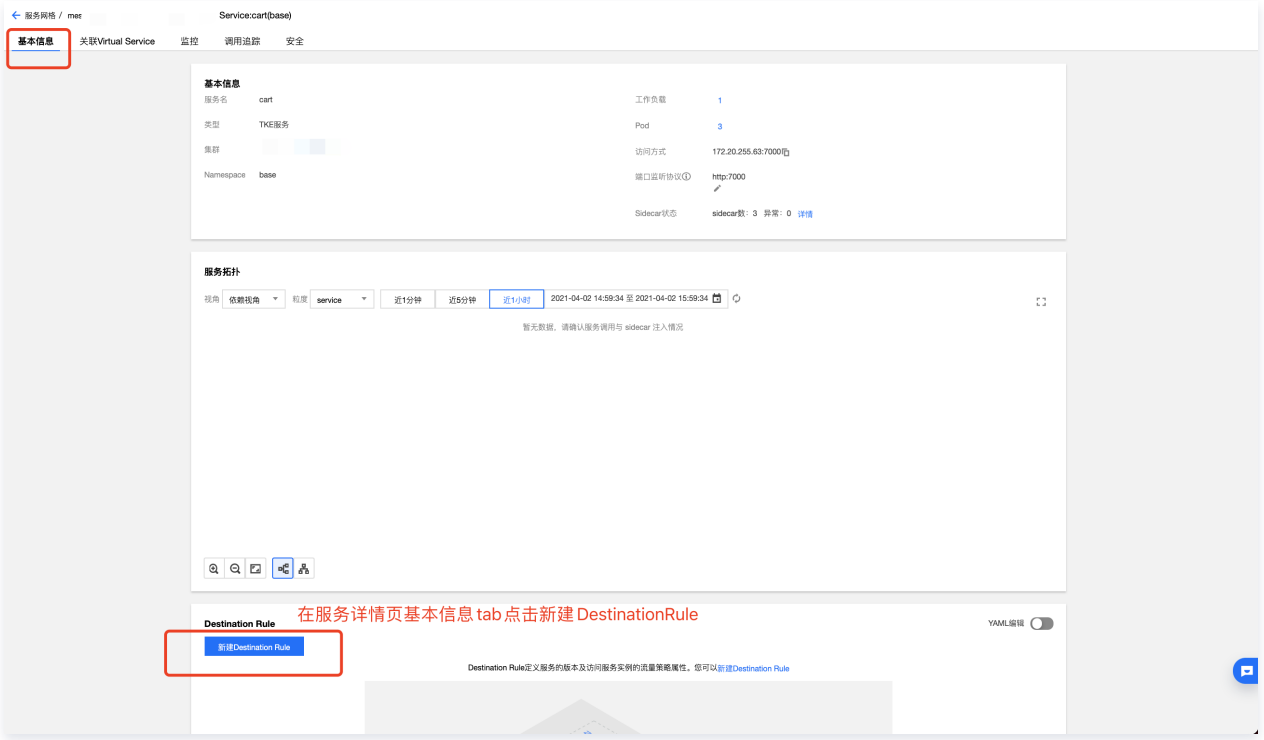
- Creating a Gateway:



- Creating a VirtualService:

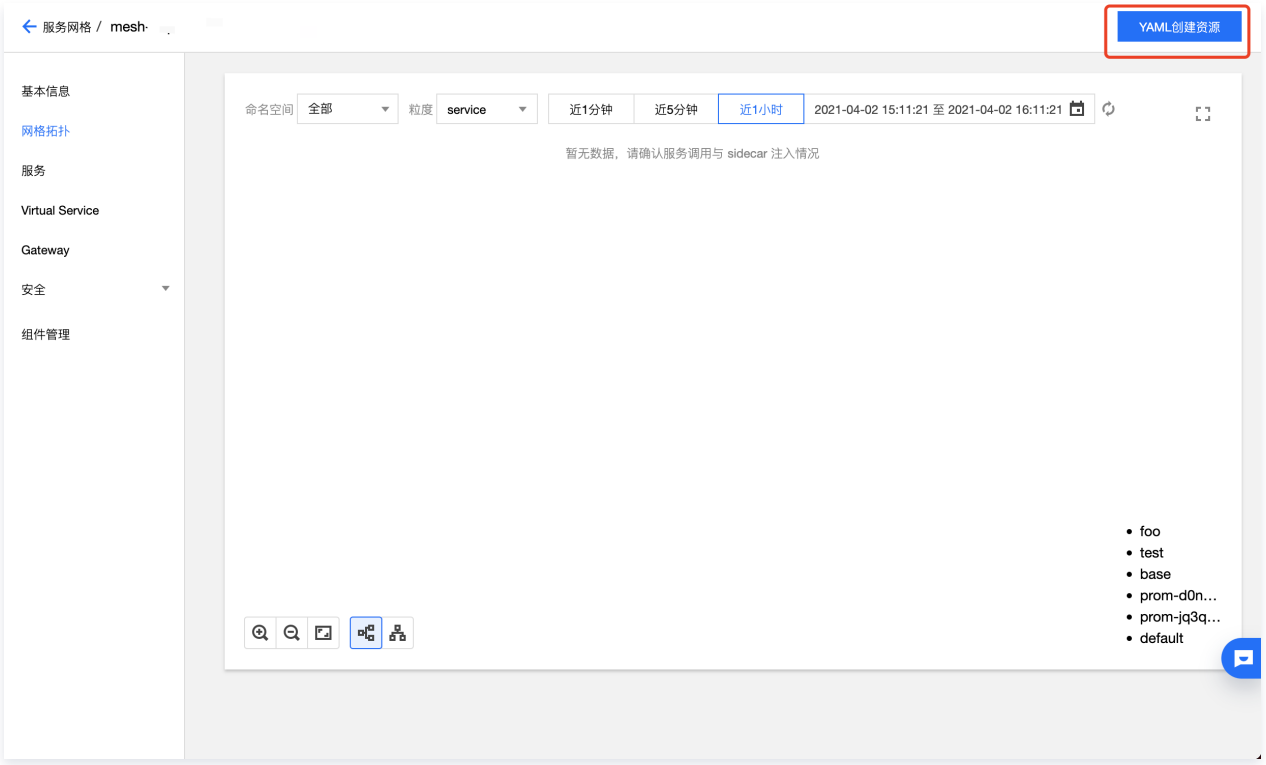


- Creating a DestinationRule: A DestinationRule is bound to a service one-to-one. It is created and managed on the service details page:



Resource Creation via YAML

You can create Istio or Kubernetes resources by clicking **Create via YAML** in the upper right corner of the grid management interface. If the YAML you submit contains a Kubernetes resource and the current mesh manages multiple clusters, you need to select a destination cluster to which the YAML-created resource is submitted.



Using VirtualService to Configure Routing Rules

Last updated: 2024-08-09 15:30:30

VirtualService defines a set of routing rules and traffic operations (such as weighted routing and fault injection) for a specified host. Each routing rule defines a matching rule for traffic of a specified protocol. If the traffic is matched, it is routed to a specified service or a version of the service. VirtualService configurations mainly include the following parts:

- **hosts**: defines hosts associated with routing rules, which can be a DNS name with a wildcard or an IP address.
- **gateways**: defines the source of traffic to which routing rules are applied. The source can be:
  - One or more gateways.
  - Sidecar within the mesh.
- **Routing Rules**: defines detailed routing rules, including routing rules for HTTP, TLS/HTTPS, and TCP.
  - http: defines an ordered list of routing rules for HTTP traffic.
  - tcp: defines an ordered list of routing rules for TCP traffic.
  - tls: defines an ordered list of routing rules for non-terminated TLS or HTTPS traffic.

Description of Major VirtualService Fields

The following are descriptions of major VirtualService fields:

Field Name	Field Type	Field description
`spec.hosts`	`string[]`	Defines a set of hosts associated with routing rules, which can be a DNS name with a wildcard or an IP address (an IP address can only be applied when the source traffic is from an edge proxy gateway). This field can be applied to both HTTP and TCP traffic. In a Kubernetes environment, you can use the name of the service as an abbreviation. Istio will complete the abbreviation with the namespace of the VirtualService. For example, a VirtualService in the default namespace containing the host abbreviation `reviews` will be completed as `reviews.default.svc.cluster.local`. To avoid misconfiguration, it is recommended to fill in the full name of the host.
`spec.gateways`	`string[]`	Defines the source of traffic to which routing rules are applied. The source can be one or more gateways or sidecars within the mesh. The specified format should be `<gateway namespace>/<gateway name>`. The reserved field `mesh` represents all sidecars within the mesh. If this parameter is omitted, `mesh` is used by default, which means the source of traffic for the routing rule is all sidecars within the mesh.
`spec.http`	`HTTPRoute[]`	Defines an ordered list of routing rules (with higher priority matching rules listed first) applied to HTTP traffic. HTTP routing rules are applied to service ports named with `http-`, `http2-`, `grpc-` within the mesh, and traffic protocols from the gateway such as `HTTP, HTTP2, GRPC, TLS-Terminated-HTTPS`.
`spec.http.match`	`HTTPMatchRequest[]`	Defines the list of routing match rules. All conditions within a single match rule item have an AND relationship, while multiple match rules in the list have an OR relationship.
`spec.http.route`	`HTTPRouteDestination[]`	Defines the list of forwarding destinations. An HTTP route can either be a redirect or a forward (default). The forwarding destination can be one or more services (service versions). You can also configure behavior like weights, header operations, etc.
`spec.http.redirect`	`HTTPRedirect`	Defines HTTP route redirection. An HTTP route can either be a redirect or a forward (default). If the `passthrough` option is specified in the rules, both route and redirect will be ignored. HTTP 301 can redirect to another URL or Authority.
`spec.http.rewrite`	`HTTPRewrite`	Defines the rewriting of HTTP URL or Authority headers, which cannot be configured simultaneously with redirection. Rewriting occurs before forwarding.
`spec.http.timeout`	Duration	Defines the timeout for an HTTP request.

`spec.http.retries`	`HTTPRetry`	Defines the retry strategy for an HTTP request.
`spec.http.fault`	`HTTPFaultInjection`	Definition of the Fault Injection Strategy for HTTP Traffic. When enabled, timeout and retry strategies will not be activated.
`spec.http.mirror`	`Destination`	Mirror HTTP traffic to a another specified destination. Mirrored traffic is on a "best effort" basis where the sidecar or gateway will not wait for a response to traffic mirroring before returning the response from the original destination. Statistics will be generated for the mirrored destination.
`spec.http.mirrorPercent`	`uint32`	Definition of the Replication Percentage for Traffic Mirroring. If not specified, 100% of the traffic will be mirrored. The maximum value is 100.
`spec.http.corsPolicy`	`CorsPolicy`	Definition of the CORS Policy (Cross-Origin Resource Sharing, CORS). For more information about CORS, see CORS. For Istio CORS Policy configuration syntax, see CorsPolicy
`spec.http.headers`	`Headers`	Definition of Header Operation Rules, including adding, updating, and removing request and response headers.
`spec.tcp`	`TCPRoute[]`	Definition of an ordered set of routing rules for TCP Traffic (prioritizing the front rules). These routing rules apply to any non-HTTP and non-TLS ports.
`spec.tcp.match`	`L4MatchAttributes[]`	Definition of the Match Rule List for TCP Traffic routing. All conditions within a single match rule must be true, while multiple match rules in the list are evaluated using OR logic.
`spec.tcp.route`	`RouteDestination[]`	Definition of the destination for TCP connection forwarding.
`spec.tls`	`TLSRoute[]`	Definition of an ordered set of routing rules for non-terminated TLS or HTTPS Traffic (prioritizing the front rules). These rules apply to mesh-internal service ports prefixed with `https-` or `tls-` and to non-terminated encrypted traffic from gateway ports using the `HTTPS, TLS` protocols. The Service Entry uses `HTTPS, TLS` protocol ports. Traffic on ports prefixed with `https-` or `tls-` without associated VirtualService rules will be treated as TCP Traffic.
`spec.tls.match`	`TLSMatchAttributes[]`	Definition of the Match Rule List for TLS Traffic routing. All conditions within a single match rule must be true, while multiple match rules in the list are evaluated using OR logic.
`spec.tls.route`	`RouteDestination[]`	Definition of the destination for connection forwarding.

Configuring Routing Rules for Traffic (South-North) from a Gateway

VirtualServices can be configured by using the console UI or YAML editing. The following shows VirtualService configurations for routing traffic from a gateway to the service frontend. The relevant gateway configurations are as follows:

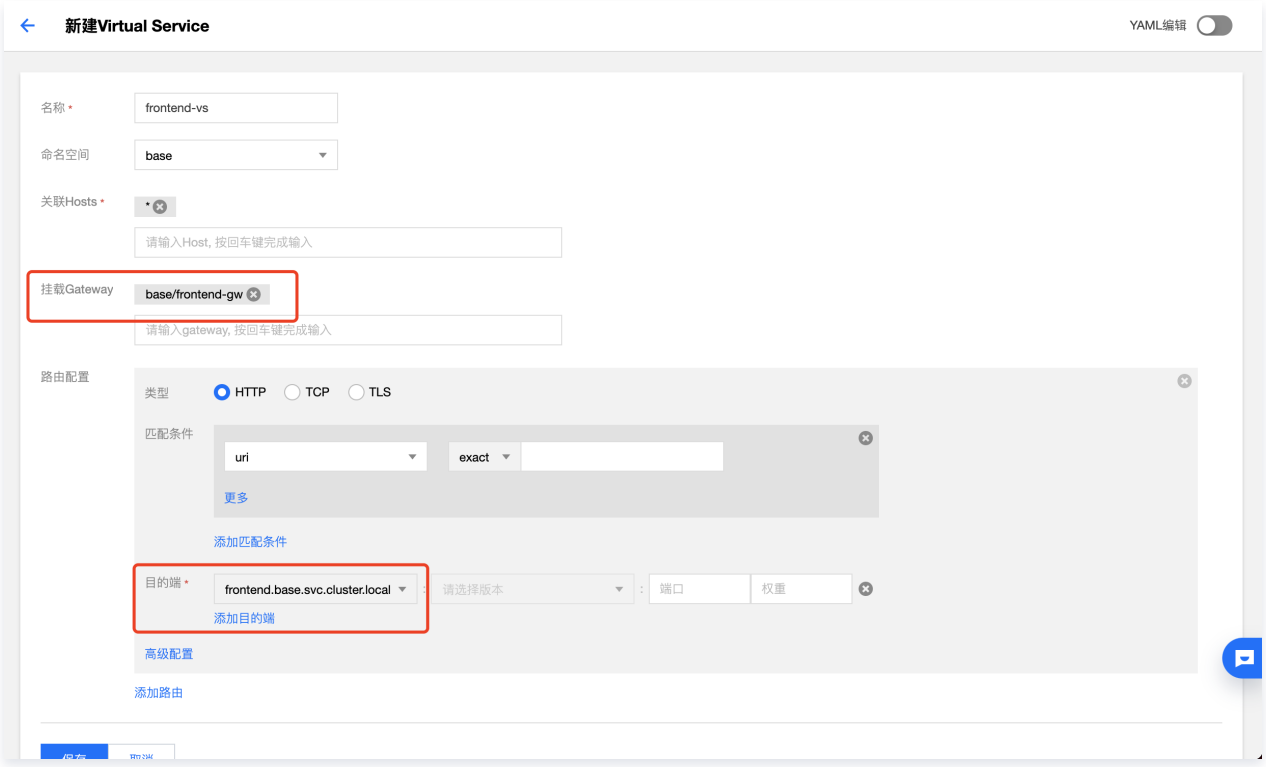
```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: frontend-gw
  namespace: base
spec:
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - '*'
  selector:
```

```
    app: istio-ingressgateway
    istio: ingressgateway
```

YAML Configuration Example

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: frontend-vs
  namespace: base
spec:
  hosts:
    - '*'
  gateways:
    - base/frontend-gw # The Gateway attached to VS is written as {namespace}/{Gateway name}
  http:
    - route:
        - destination:
            host: frontend.base.svc.cluster.local # The routing destination includes the full host name
of the frontend service
```

Console Configuration Example



Configuring Routing Rules for Traffic (East-West) from a Mesh

Below is a configuration example of the VirtualService for routing 50% of the traffic to Version v1 and 50% to Version v2 for the product service host `product.base.svc.cluster.local` accessed internally from the mesh. The service version of the product is defined through the following DestinationRule: (Gray release)

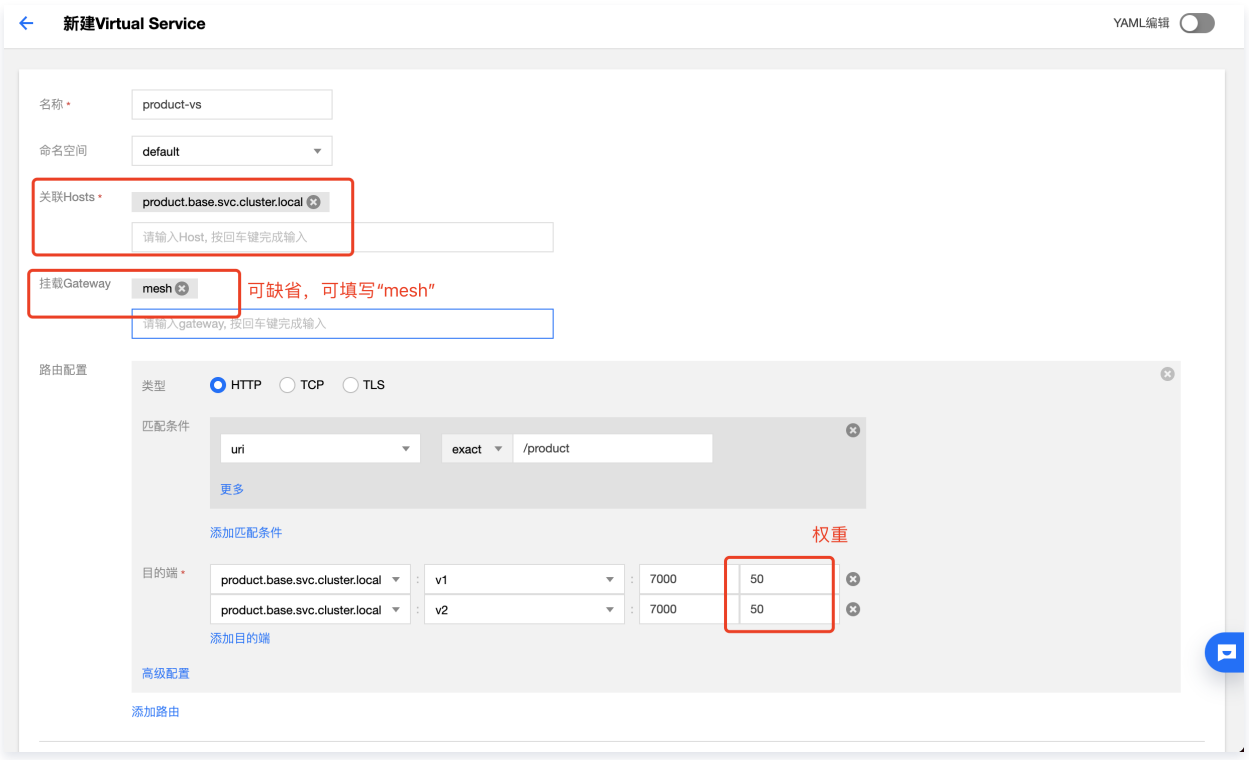
```
apiVersion: networking.istio.io/v1alpha3
```

```
kind: DestinationRule
metadata:
  name: product
  namespace: base
spec:
  host: product
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
```

YAML Configuration Example

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: product-vs
  namespace: base
spec: # Default gateway parameters, indicating that this routing configuration applies to sidecar traffic
within the mesh
  hosts:
    - "product.base.svc.cluster.local" # Matches traffic accessing this host
  http:
    - match:
        - uri:
            exact: /product
      route:
        - destination: # Configure the destination and weight
            host: product.base.svc.cluster.local
            subset: v1
            port:
              number: 7000
          weight: 50
        - destination:
            host: product.base.svc.cluster.local
            subset: v2
            port:
              number: 7000
          weight: 50
```

Console Configuration Example



Using DestinationRule to Configure Service Versions and Traffic Policies

Last updated: 2024-08-09 15:30:50

DestinationRule defines versions of a service and traffic policies for the service after routing has occurred. These rules include load balancing, connection pool size, and health check (to detect and evict unhealthy hosts from the load balancing backend).

Description of Major DestinationRule Fields

Below is a description of the important fields in DestinationRule:

Field Name	Field Type	Field description
`spec.host`	`string`	The name of the service associated with the DestinationRule configuration. This can be an automatically discovered service (such as a Kubernetes service name) or hosts declared through ServiceEntry. If the specified service name cannot be found in the aforementioned sources, then the rules defined in this DestinationRule are invalid.
`spec.subsets`	`Subset[]`	Define versions (subsets) of the service. Versions can be matched to endpoints within the service through tag key-value pairs. Traffic policies can override at the subset level.
`spec.trafficPolicy`	`trafficPolicy`	Define traffic policies, including CLB, connection pools, health checks, and TLS policies.
`spec.trafficPolicy.loadBalancer`	-	Configure CLB algorithms. You can configure simple CLB algorithms (round robin, least conn, random...), consistent hashing (session persistence, supporting hashing based on header name, cookie, IP, query parameters), and region-aware CLB algorithms.
`spec.trafficPolicy.connectionPool`	-	Configure the number of connections with upstream services. You can set up TCP/HTTP connection pools.
`spec.trafficPolicy.outlierDetection`	-	Configure the eviction of unhealthy hosts from the CLB pool.
`spec.trafficPolicy.tls`	-	Configure client-side TLS related to connecting to upstream services, to be used in conjunction with PeerAuthentication policies (server-side TLS mode configuration).
`spec.trafficPolicy.portLevelSettings`	-	Configure port-level traffic policies. Port-level traffic policies will override service/subset-level traffic policies.

Defining Service Versions (Subsets)

DestinationRule can define versions (subsets) of a service, and a subset is the smallest traffic management unit of Tencent Cloud Mesh. For example, you can configure traffic to be routed to a specified subset of a specified service. The following is a configuration example of using DestinationRule to define two subsets of the product service.

YAML Configuration Example

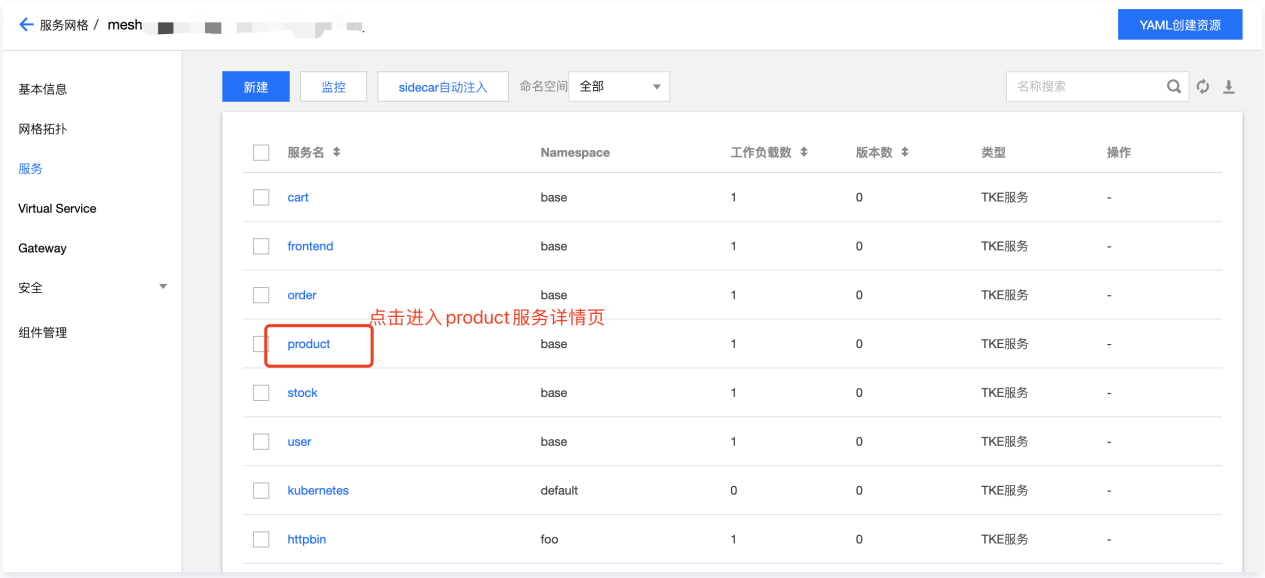
```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: product
  namespace: base
spec:
  host: product
  subsets:
```

```
  - name: v1
    labels:
      version: v1 # subset v1 matches the service endpoint through the tag version:v1
  - name: v2
    labels:
      version: v2 # subset v2 matches the service endpoint through the tag version:v2
```

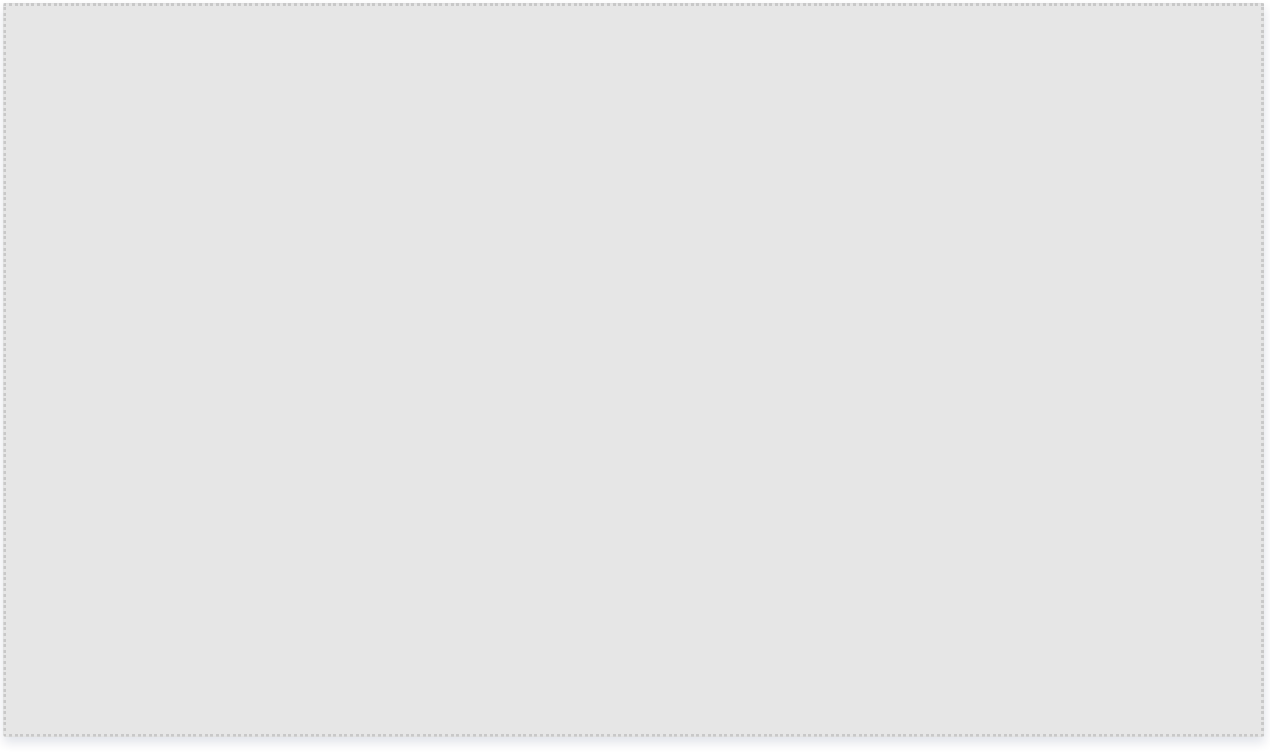
Console Configuration Example

DestinationRule and service are in a one-to-one binding relationship. To configure the DestinationRule for the product service, enter the product service details page from the service list page and configure it in the Basic Information tab. The steps to configure two versions of the product service in the console are as follows:

1. On the service list page, click to enter the product service detail page.



2. In the service detail page **Basic Information**, in the third DestinationRule card area, click **Create Destination Rule** to enter the creation popup.



3. In the popup page, add two versions for the product service, then click save.

4. Version configuration of the product service is complete.

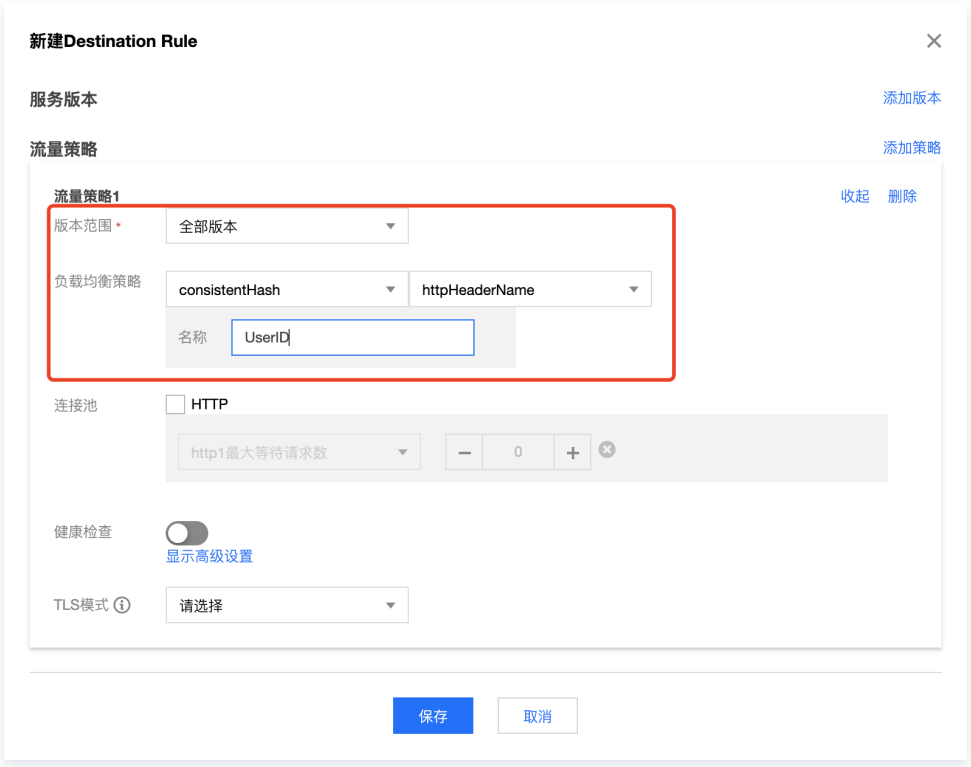
Configuring Consistent Hash-based Load Balancing

The following is a configuration example of using DestinationRule to configure the cart service to perform consistent hash-based load balancing based on the HTTP header name.

YAML Configuration Example

```
kind: DestinationRule
metadata:
  name: cart
  namespace: base
spec:
  host: cart
  trafficPolicy:
    loadBalancer:
      consistentHash:
        httpHeaderName: UserID # Configure the traffic to the cart service to perform consistent hash-
based load balancing based on the header UserID
```

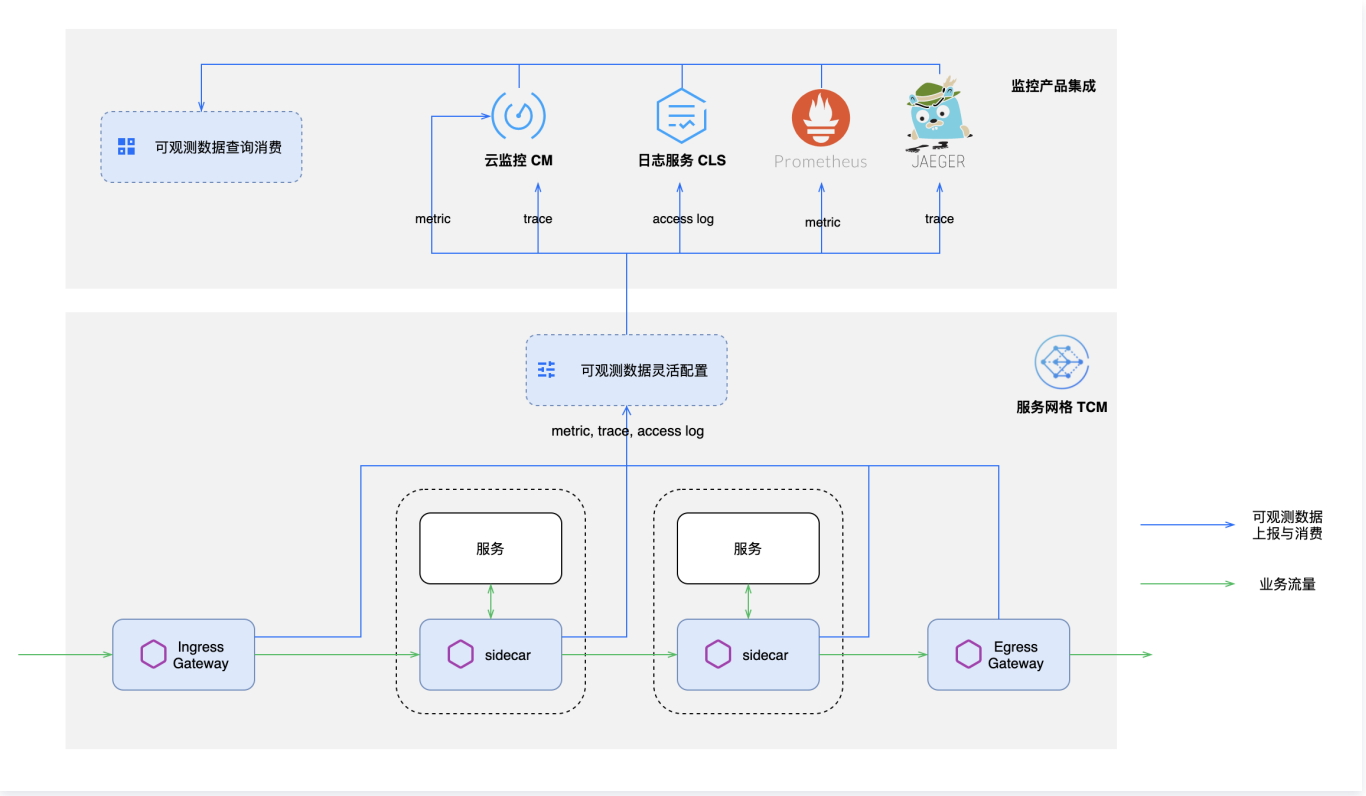
Console Configuration Example



Observability Overview

Last updated: 2024-08-09 15:31:15

Tencent Cloud Mesh provides end-to-end observability between north-south and east-west services.
The collection of observation data depends on reporting of the envoy sidecar proxy (data plane) of a service that has been injected with sidecars. You can flexibly control the production and calculation of observable data on the data plane through Tencent Cloud Mesh. Tencent Cloud Mesh integrates the observation data into suitable monitoring products to provide you with the observability of traffic between services at the edge of a mesh and services inside the mesh.



Tencent Cloud Mesh provides three types of observable data:

Observable Data Types	Description
Monitoring Metrics	Monitoring metrics can provide traffic observation data for services or gateways, suitable for single service developers.
Call Traces	Call tracing can link multiple layers of calls in a single request into a call chain, making it easier to observe the call structure, perform performance analysis, and locate anomalies.
Access Logs	Access logging is the complete record generated by the envoy proxy for each request, including information from both the request layer and the sidecar proxy layer, aiding operations personnel in access auditing and fault troubleshooting.

The three types of observable data are described as follows:

Observable Data	Recorded Information	Applicable Scenario or Role
Metric	Traffic observation data for a single service or gateway, including but not limited to metrics such as latency, number of requests, and request size, is unsampled. For more metric information, see Istio Standard Metrics.	Developers of a single service monitor the operating status of the service.
Trace	Records the dependencies between service calls, adding URL dimension information compared to metrics. However, the recorded	Overall service developers perform call dependencies

	data is generally sampled	and performance analysis of all services.
Access Log	Complete information about each request, including rich information output at the sidecar proxy layer. For more information, see Envoy Access Logging.	Mesh operations personnel conduct access auditing and fault troubleshooting.

Monitoring Metrics

Last updated: 2024-08-09 15:31:29

Currently, Tencent Cloud TCM uses Prometheus for monitoring metrics storage. You can choose to use Prometheus Monitoring TMP to provide the collection, storage, and display of service traffic metric data, or you can use a third-party Prometheus service as a monitoring metric storage service.
TCM data plane Sidecar will report metric data to TMP or a third-party Prometheus service. The console queries metric data from the corresponding service, providing display and analysis of mesh topology, service topology, and service monitoring (number of requests, request status code distribution, request duration, request size) charts. Additionally, if you have custom monitoring requirements, you can set up custom monitoring dashboards using Grafana in TMP.

Operation step

Enabling TMP Monitoring

In the Create a mesh or **Mesh Basic Information page**, under Observability Configuration > Monitoring Metrics, select **Tencent Cloud Prometheus Monitoring TMP**. You can choose to automatically create or associate an existing TMP instance as needed. Once enabled, Sidecar will report metric data to the corresponding instance, and you can also check this instance in the TMP console.

Enable Third-party Prometheus Service

In the Create Grid or **Mesh Basic Information Page** under Observability Configuration > Monitoring Indicators, check Third-party Prometheus Service and fill in the VPC information, address, and authentication information corresponding to that service.

> **Note:**
> Since the TCM console queries and displays monitoring metric data from Prometheus services, high network reachability and stability of the data source service are required. Therefore, currently, only intranet access is supported.

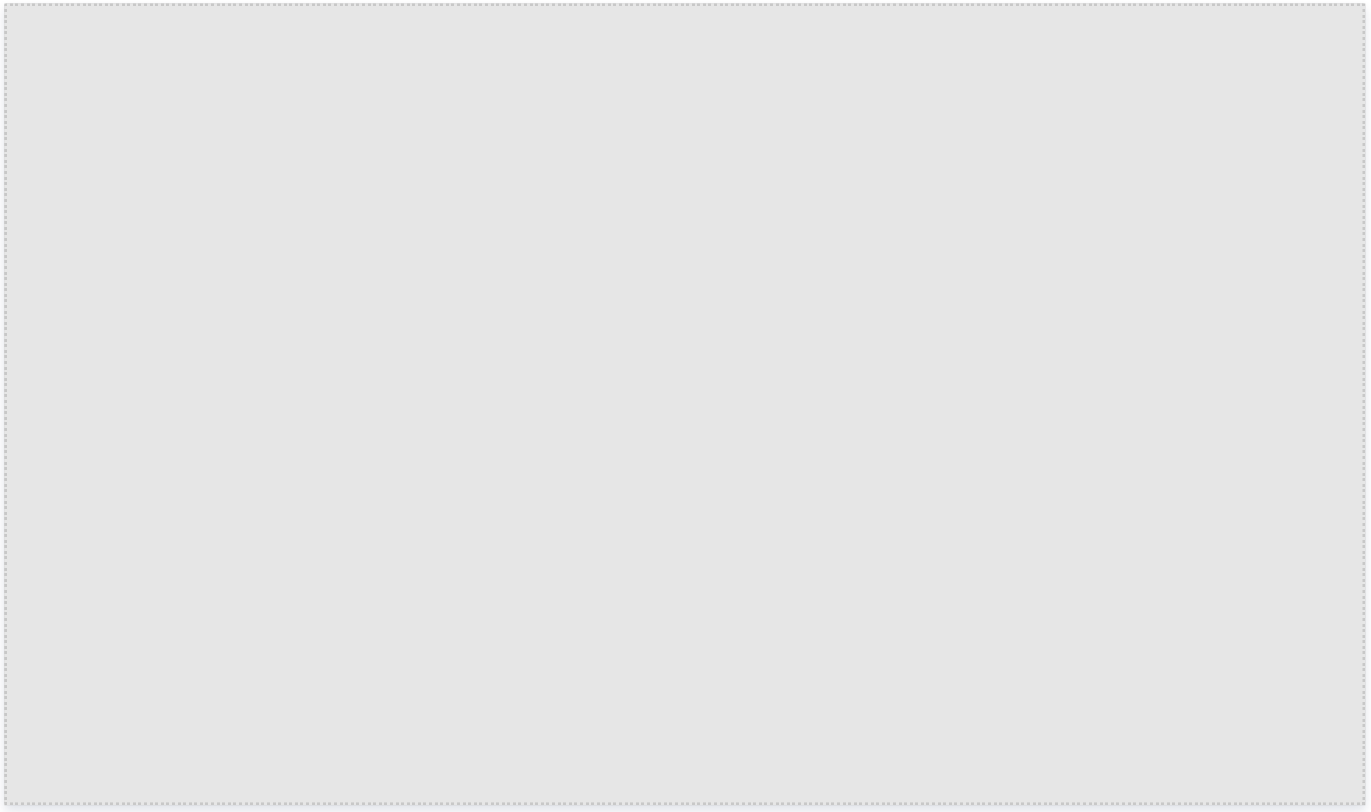
Viewing Monitoring Charts

Mesh topology

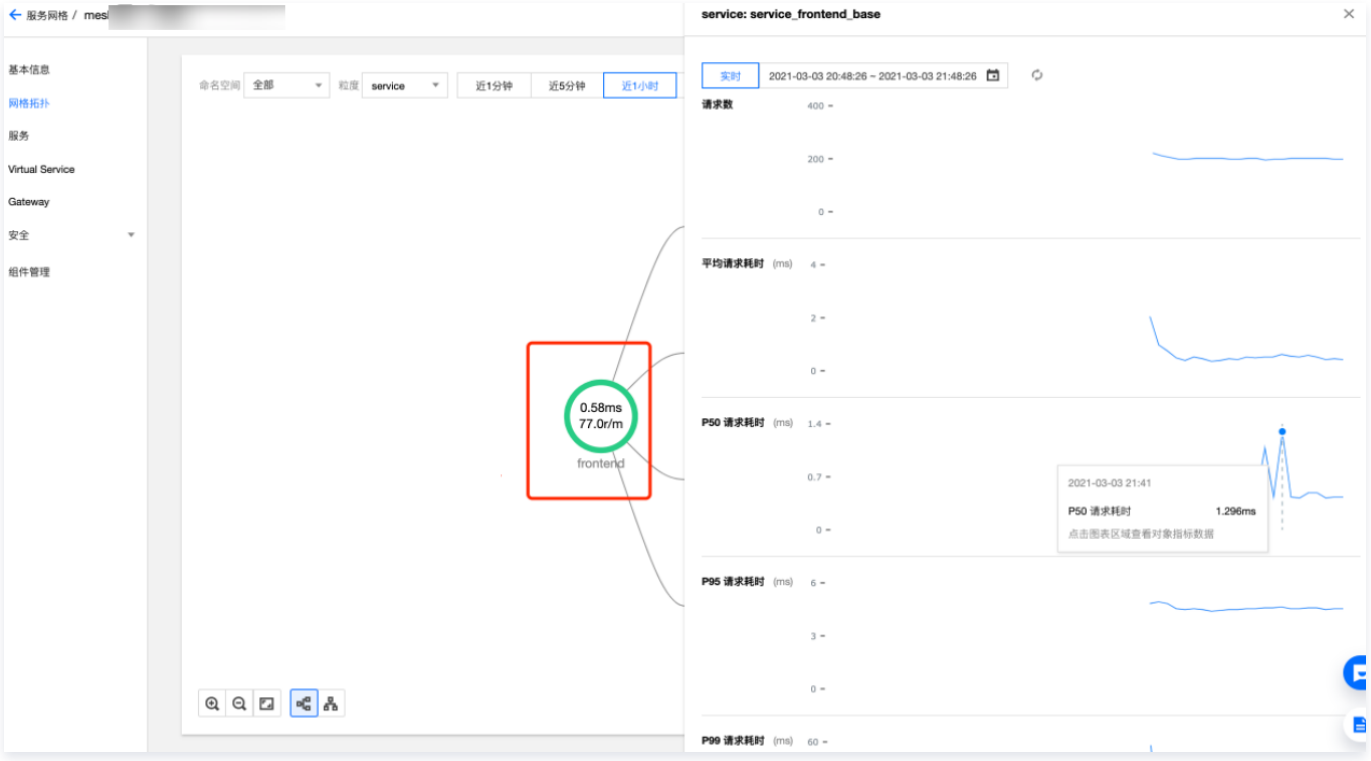
A mesh topology records call structures of all services in a service mesh. Before viewing the mesh topology, ensure that sidecars have been injected into related services and that there is request traffic. The procedure of viewing the mesh topology of a specified mesh is as follows:

1. Log in to TCM Console, and click the specified mesh ID in the list to enter the mesh details page.

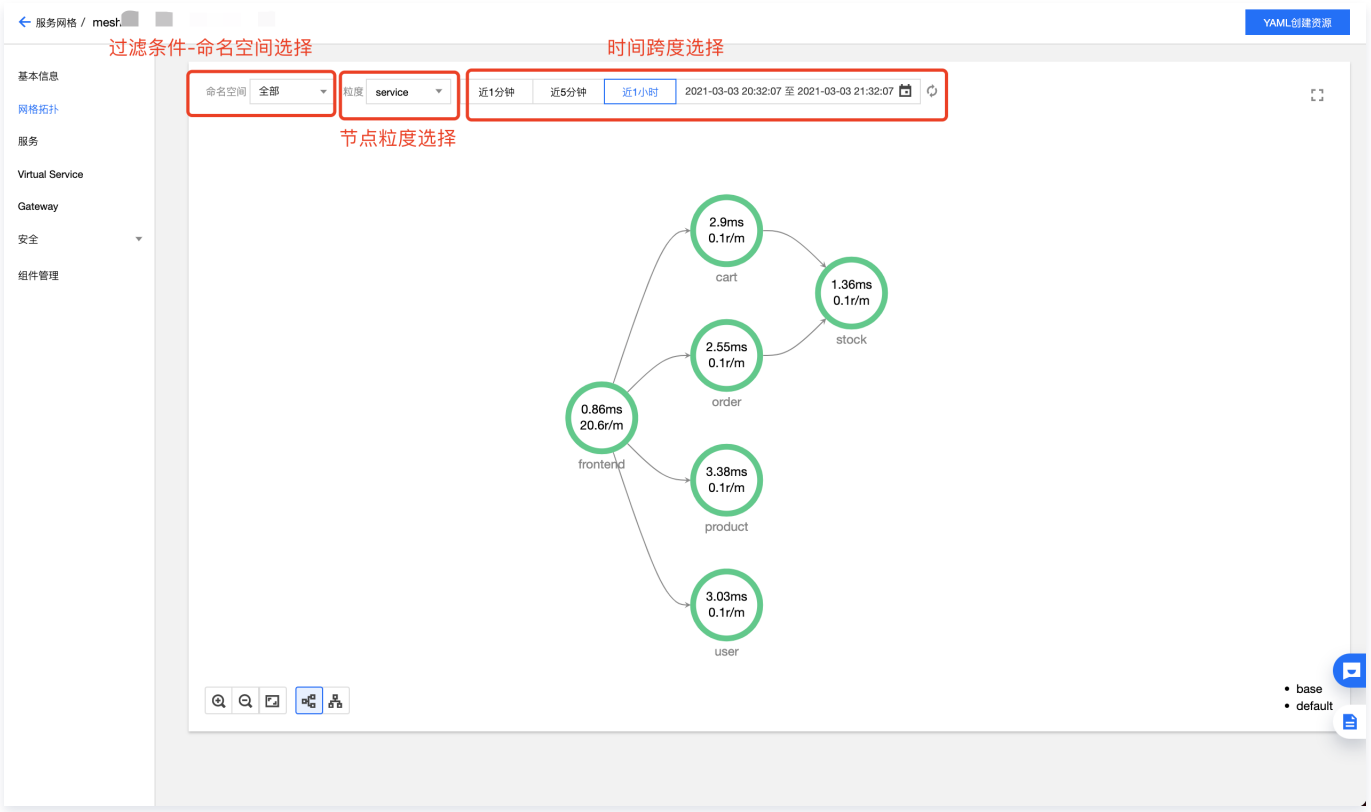
2. Click the left tab **Mesh Topology** to view the current TCM topology diagram. As shown below:



3. Click a node to display monitoring details related to the node. As shown below:



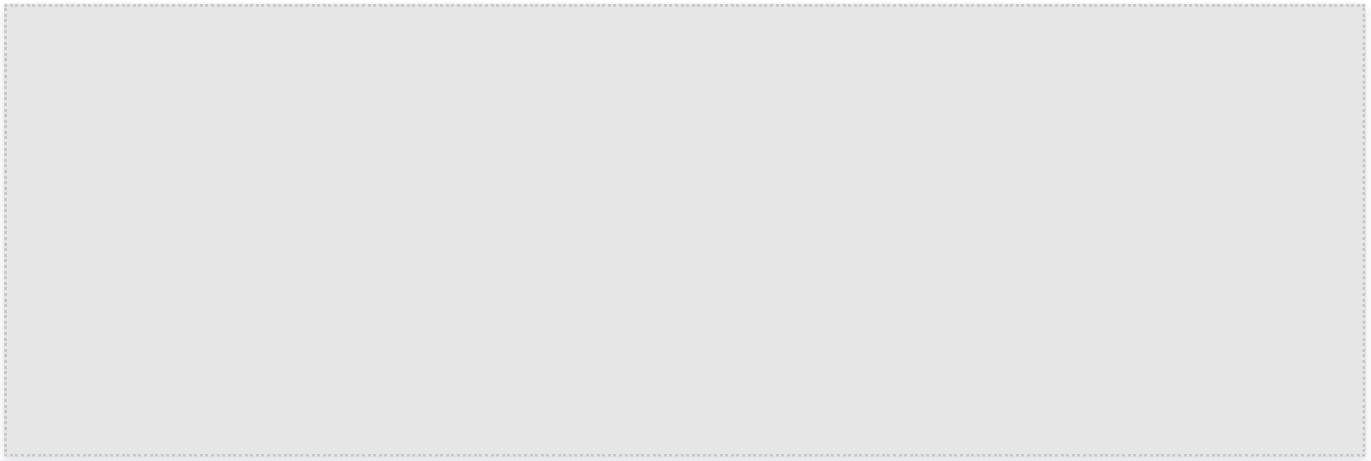
4. At the top of the page, select data filtering conditions (including namespace and time span) and granularity of nodes. Currently, service granularity and workload granularity are supported. As shown below:



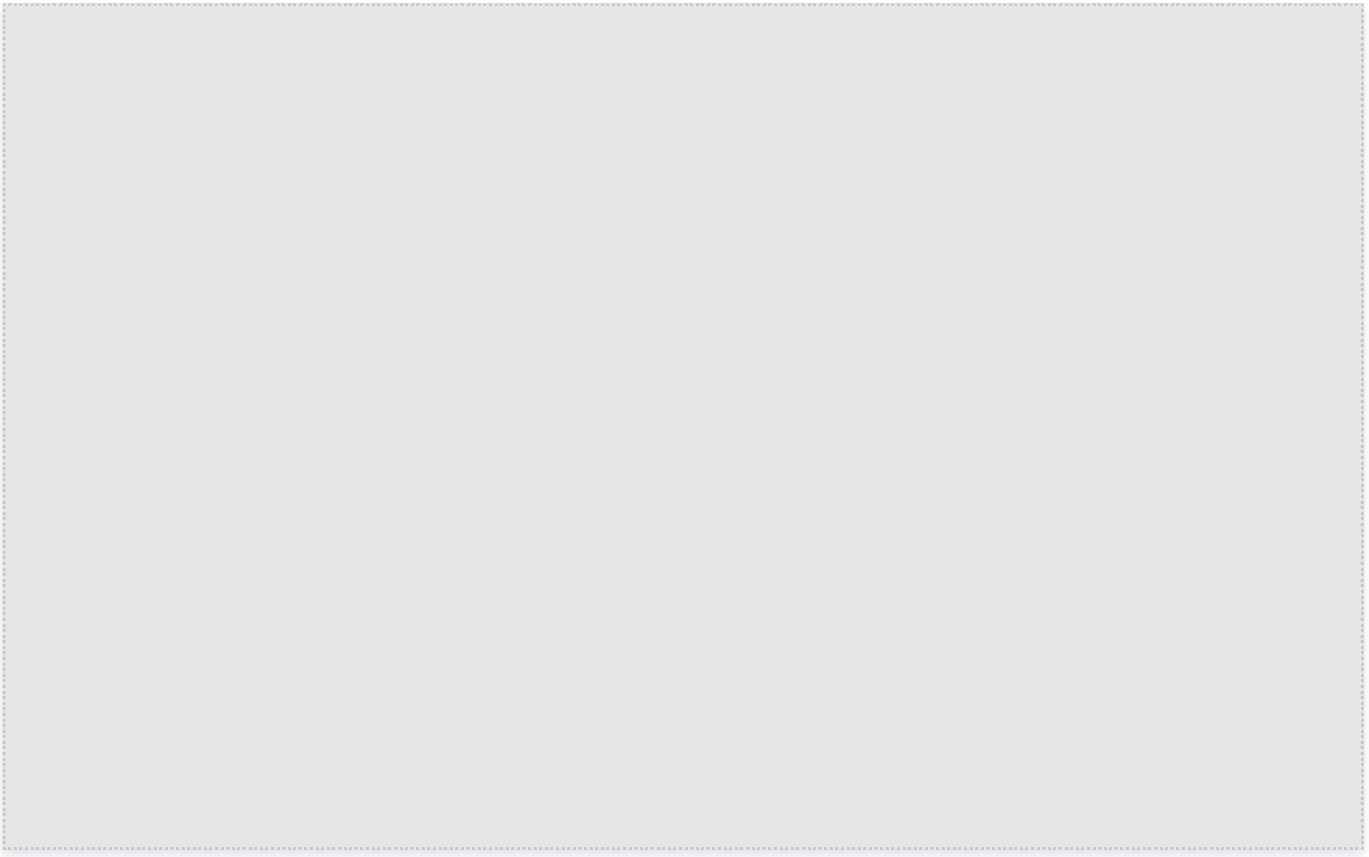
Service topology

A service topology records dependencies between previous and next calls of a service. The procedure of viewing the service topology of a specified service is as follows:

1. On the details page of the specified mesh, click **Service** in the left sidebar to enter the service list page.
2. Click the service to be viewed to enter the service details page.



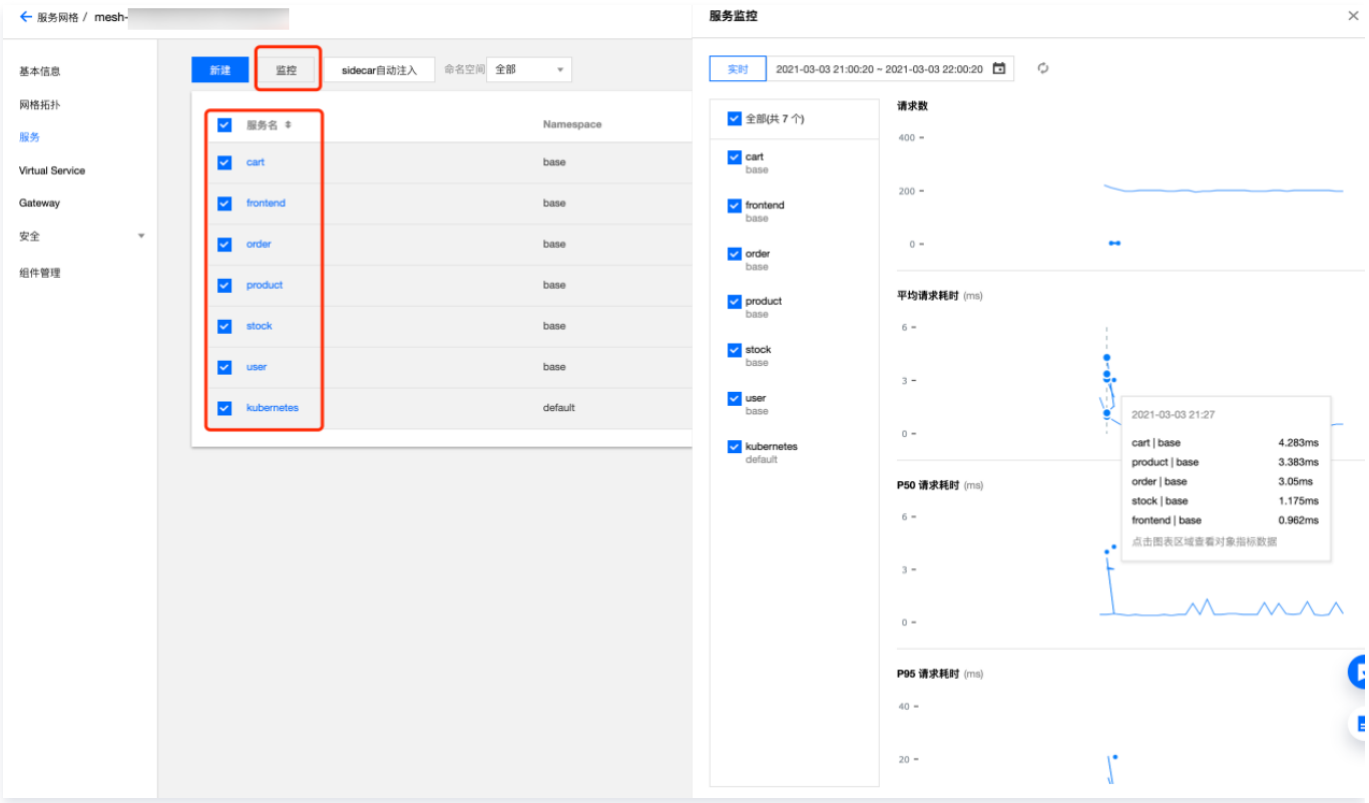
3. On the service details page, under "Basic Information," you can view the service topology of the service. As shown below:



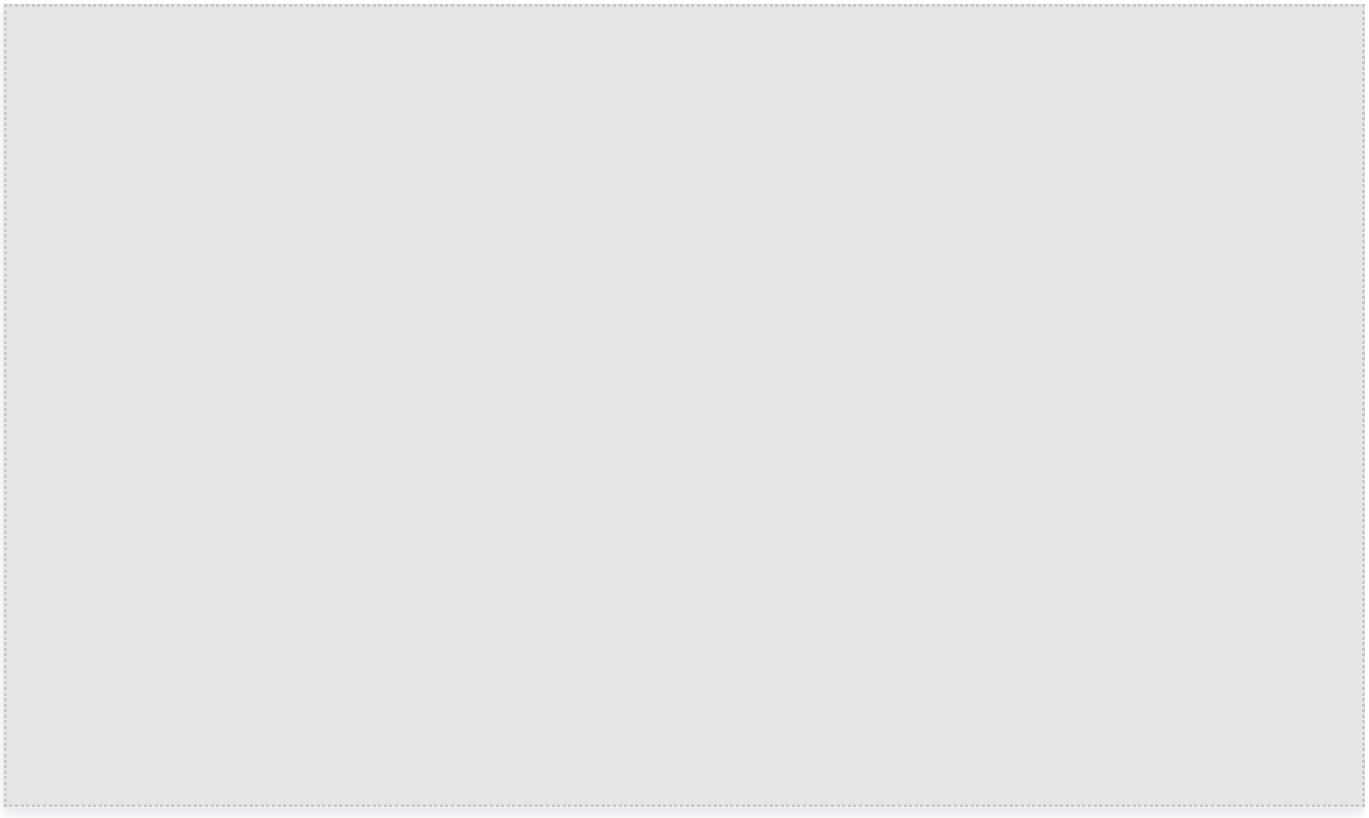
Service monitoring

You can compare the monitoring data (such as the number of requests, request duration, request size) of multiple services on the service list page, or view the monitoring details of a specified service on the service details page.

- **View monitoring data of multiple services on the service list page:**
  1.1 Log in to TCM Console, and click the specified mesh ID in the list to enter the mesh details page.
  1.2 Select **Service > Monitoring**, click the service you want to view the monitoring data for and check the service monitoring data on the right side. See the figure below:



- **Viewing the detailed monitoring data charts of a specified service on the service details page:**
  1.1 On the details page of the specified mesh, click **Service** in the left sidebar to enter the service list page.
  1.2 Click the service to be viewed to enter the service details page.
  1.3 You can view it in the "Monitoring" section on the service details page.



Disabling monitoring

You can choose to edit the observability configuration on the **Mesh Basic Information Page**, and deselect **Tencent Cloud Prometheus Monitoring TMP**. After deselection, the TMP instance will not be deleted on the TCM side. If necessary, go to the TMP Console to further delete the TMP instance.

Call Traces

Last updated: 2024-08-09 15:31:44

By default, TCM integrates APM as the consumer end for call tracing. After enabling, the mesh will create an APM instance for you and report tracing data to the corresponding APM instance. You can view a complete call waterfall chart of a request in the mesh and tracing log information about calls at each layer on the TCM console. This helps you understand service call dependencies and conduct latency analysis in the mesh. You can also view call data directly on the APM console.
In addition to APM, the mesh supports reporting call data to third-party Jaeger/Zpkin services. If third-party tracing services are enabled, the TCM console will not be able to display call tracing information, and you will need to view it in the third-party service.

Configuring a Call Tracing Sampling Rate

A call tracing sampling rate is a sampling ratio of tracing data, and the resources consumed by sidecars during data collection and reporting are positively related to the bandwidth and sampling rate. Usually, in a production environment, it is not necessary to generate, collect, or report tracing data for all calls, so as to avoid excessive consumption of computing and bandwidth resources. Instead, only a certain proportion needs to be configured. It is recommended that a 100% sampling rate is configured for a development and test environment and a 1% sampling rate is configured for a production environment.
You can configure the call tracing sampling rate in the following two ways:

Method 1: Configure the sampling rate when creating the mesh

1. log in to TCM console.
2. Select a region, click **Create** at the top-left corner of the page.
3. In the Create TCM page, fill in the necessary configurations for creating the grid. In the "Observability Configuration", configure the sampling rate as shown in the following image:

Method 2: Modify the sampling rate configuration on the mesh basic information page

1. log in to TCM console.
2. Select the region, click the mesh ID to be reconfigured, and enter the mesh management page.

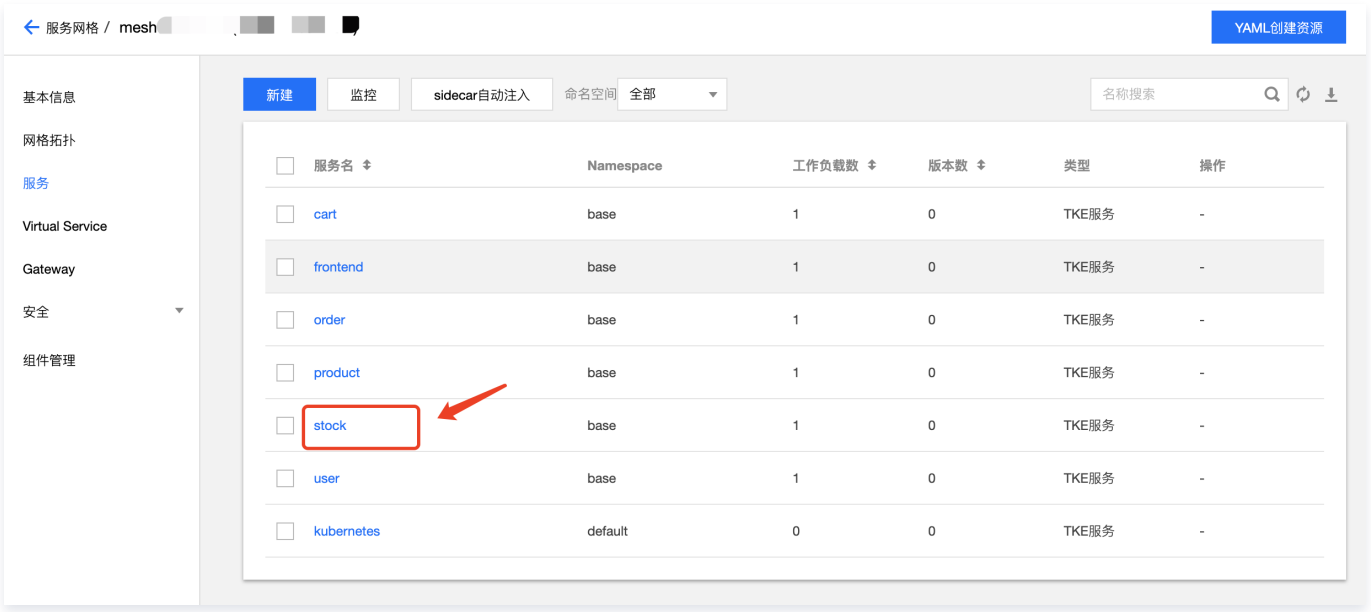
3. On the Grid Basic Information page, click **Trace Sampling > Sampling Rate** link to adjust the sampling rate as shown in the following image:

Viewing Call Tracing

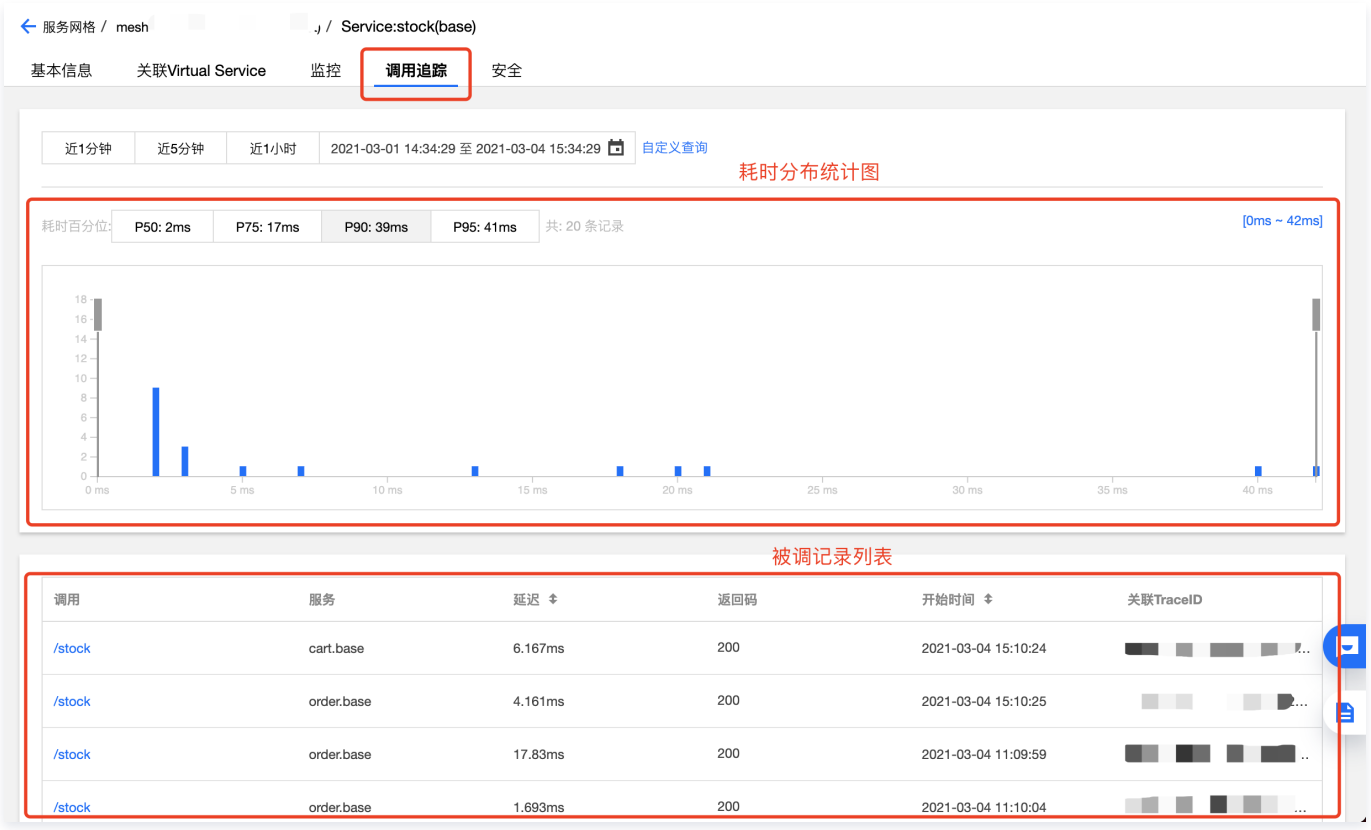
> **Note:**
> Since the service in TCM does not always correspond one-to-one with the k8s service, the actual service name used when reporting in the tracing span is the app label value in the pod (k8s service is also associated with the pod through this label, but the service name can be different from the label). When querying service call chain information on the console, the service name defaults to the k8s service name. Therefore, if the k8s service name doesn't match the pod's app label, the console will not be able to retrieve the corresponding data.

The procedure for viewing call tracing is as follows:

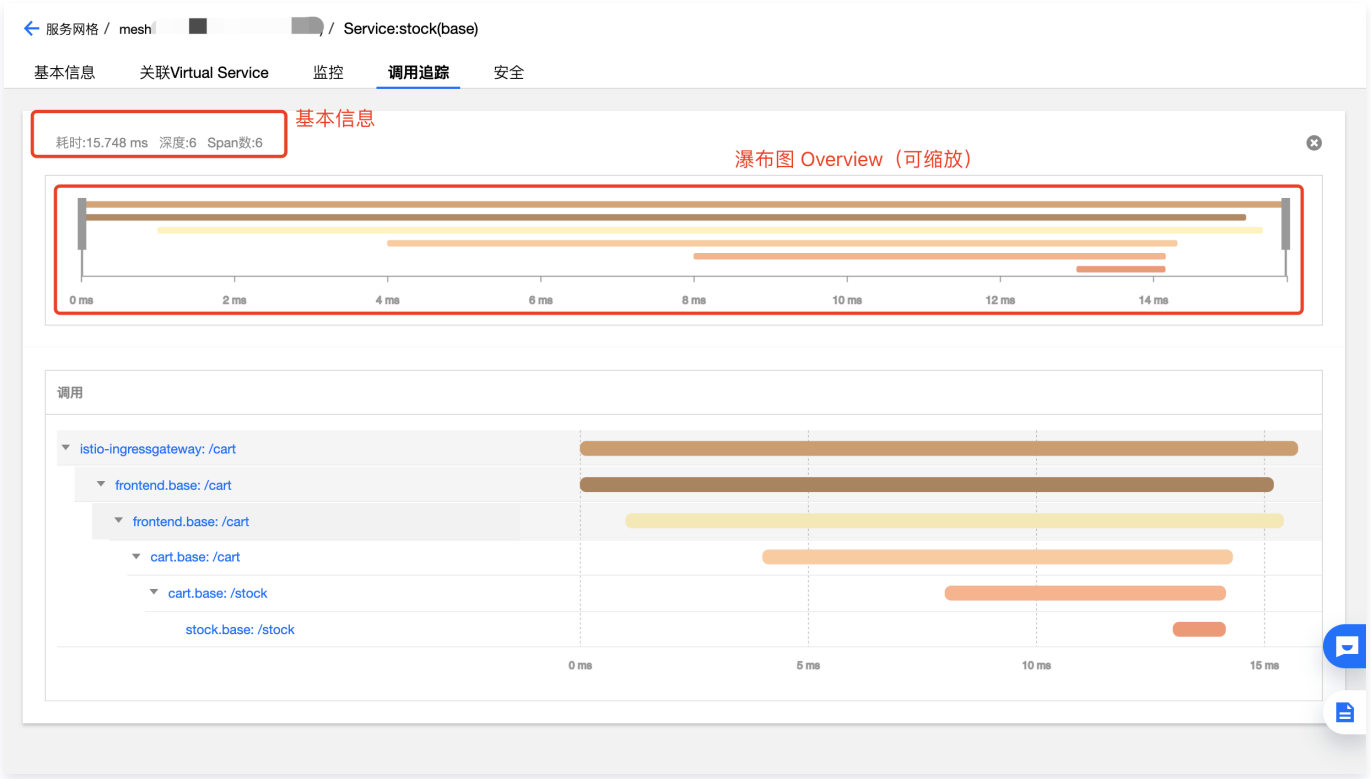
1. To focus on the calling information of a specific service in the mesh, click the service on the mesh service list page to enter the service details page.

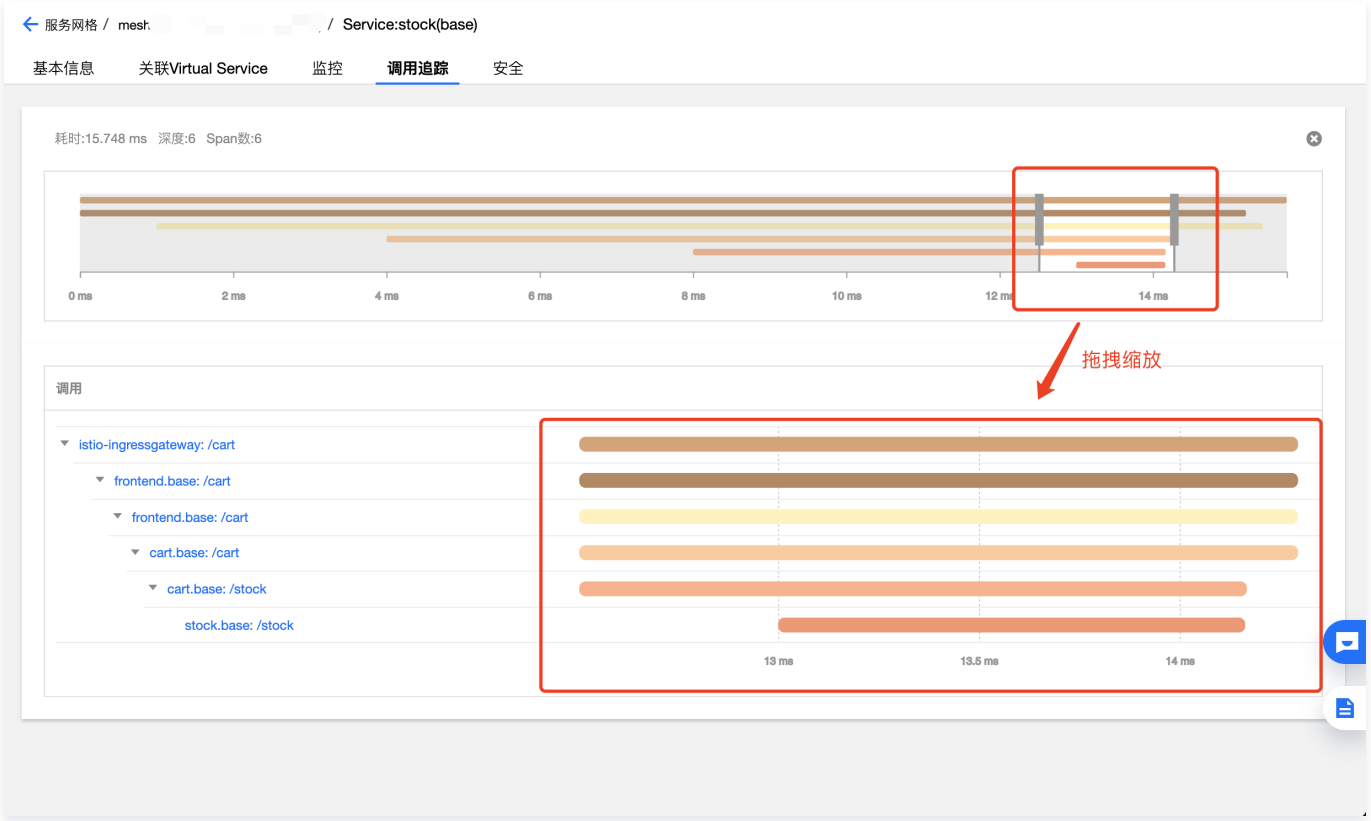


2. On the service details page, click **Call Tracing** to view the list of call records where the service acts as the callee, and the time-consuming distribution histogram of these call records.

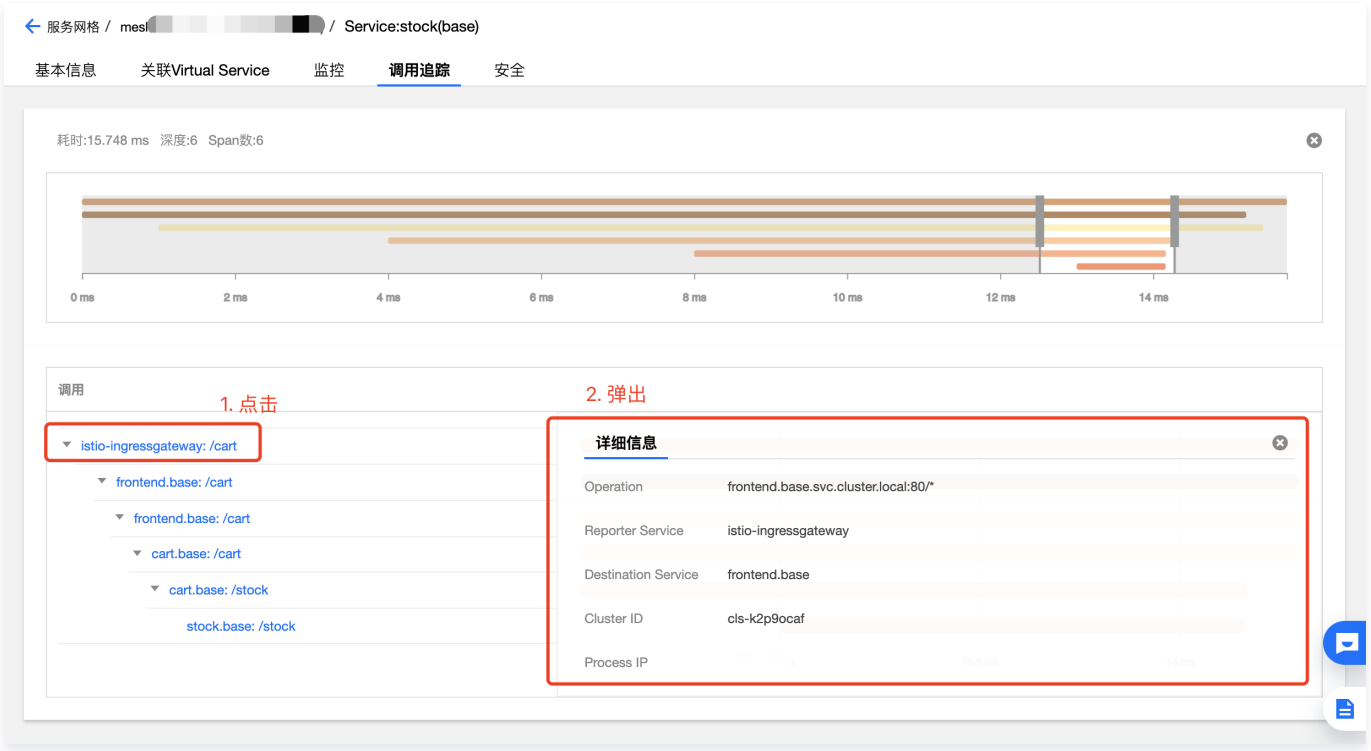


3. The first column in the called record list records the URL of the call. Click to view the complete call trace waterfall chart related to the call. The overview of the waterfall chart above can be zoomed through dragging.

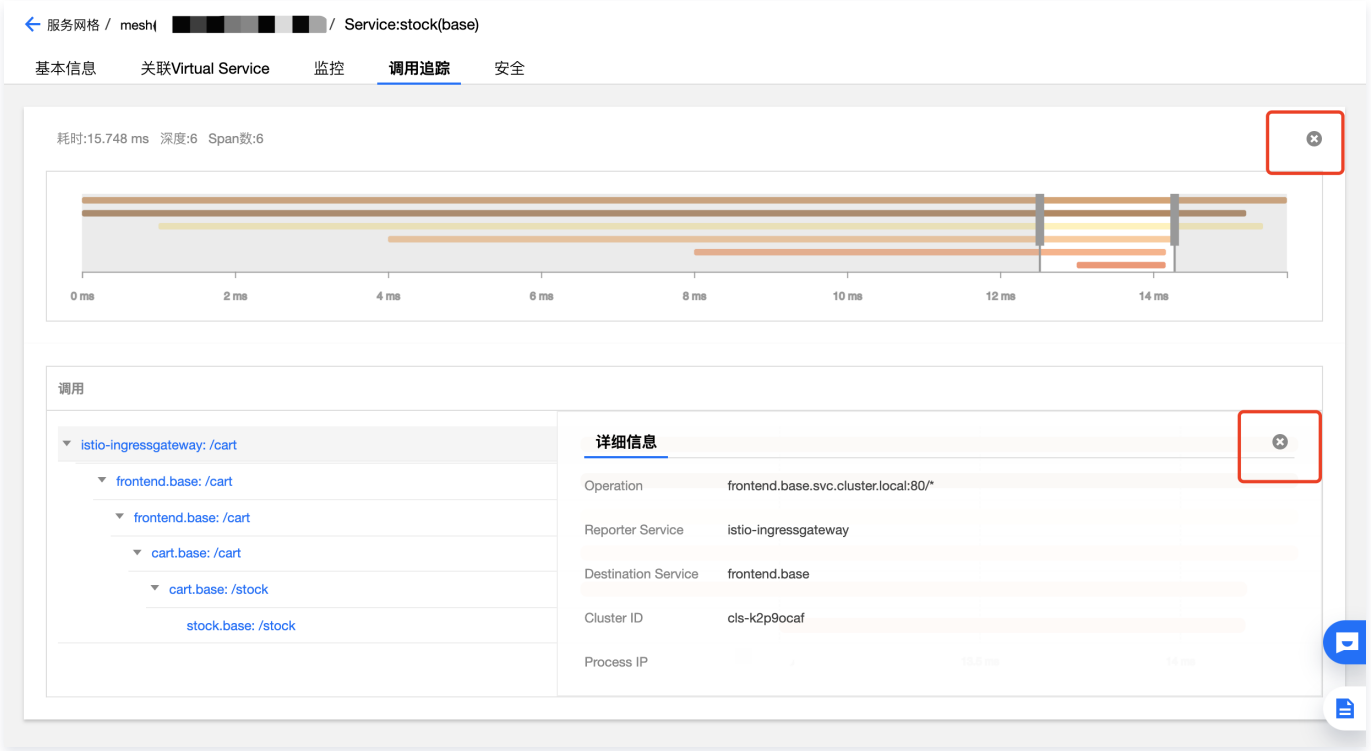




4. Click the call whose details you want to view to see the detailed tracing logs of the call.

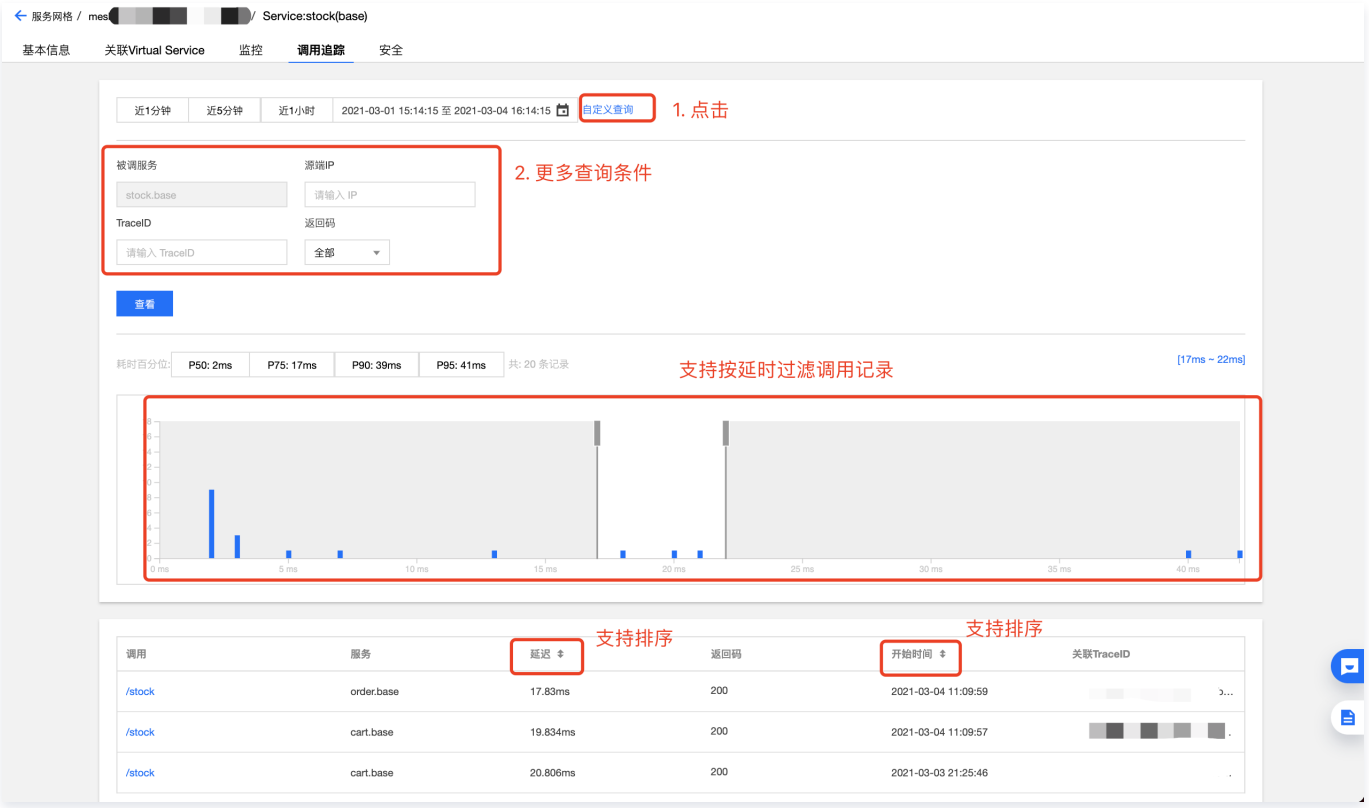


5. Click the close button to close the span details and return to the called record list page.



6. Tips for querying service's called records: You can filter the called records by duration, time span, source IP, Trace ID, and return code. After filtering, you can sort the call records by **Latency** and **Start Time**, making it easier for you to select the calls you need to

focus on.



View full call chain

The basic condition for forming a complete call chain is that the call chain information of each segment of the request must be passed down until the request ends. Although the Sidecar adds link tracing-related information when forwarding requests, because the Sidecar handles inbound and outbound traffic in two separate steps with business logic steps in between, which the Sidecar cannot intervene (the business logic steps break the transmission chain, and the Sidecar loses the inbound and outbound traffic mapping relationship), the link formation will be interrupted if the business code does not forward these additional call chain information. Therefore, TCM cannot achieve fully non-intrusive full-link tracing and requires forwarding logic for link tracing-related information in the business code

TCM adds headers conforming to the B3 Trace Specification to the layer 7 traffic to record call chain information. Thus, you need to make minor modifications to the business code to pass these headers so that the mesh can correctly associate inbound and outbound spans, forming a complete call chain. These headers include:

- x-request-id
- x-b3-traceid
- x-b3-spanid
- x-b3-parentspanid
- x-b3-sampled
- x-b3-flags
- x-ot-span-context

For more information on Envoy-based tracing, please refer to Istio Distributed Tracing FAQ.

Access Logs

Last updated: 2024-08-09 15:31:58

You can configure the scope, output format, and automatic collection of Access Log for TCM data plane (container standard output) to integrate with the CLS product on the cloud. You can configure access logs when creating a mesh, and after mesh creation, you can also modify access log configurations on the basic information page.

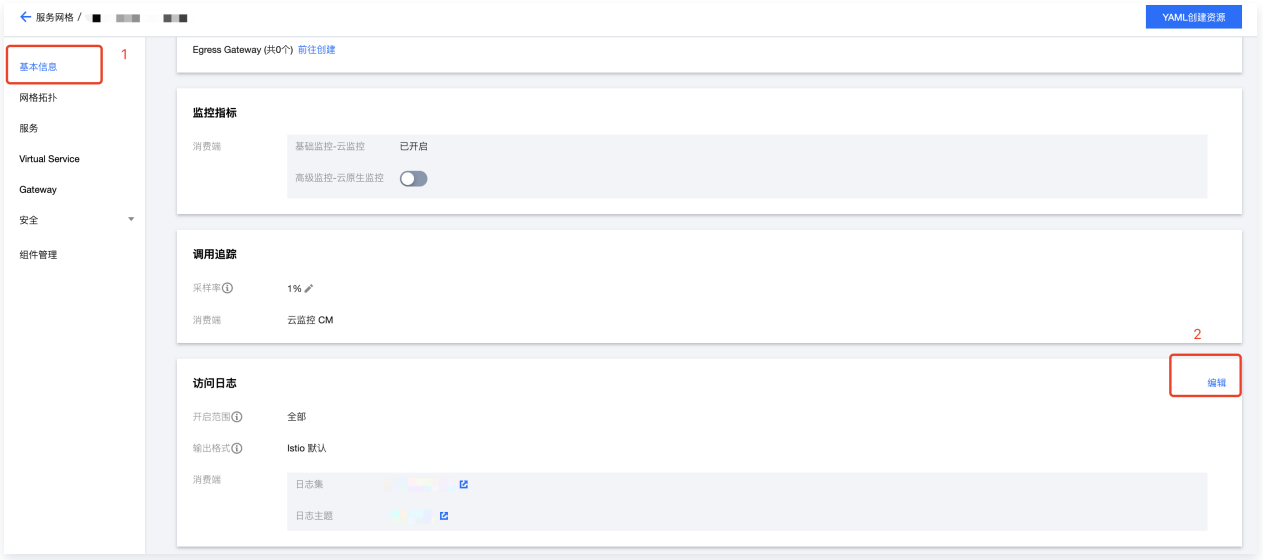
Access Log Configuration

The currently supported access log configurations are as follows:

Configuration Item	Description
Scope	Configure the data plane (Edge Proxy Gateway and istio-proxy sidecar) for access log output. You can enable access logs for a specified Edge Proxy Gateway, all data planes under a specified namespace, or all data planes of the mesh to the container standard output.
Output format	Configure the fields and format template for access log output. The default format outputs the fields as per Istio's default settings. The enhanced format adds Trace ID output on top of the default format. The custom format allows defining the log output format (see Envoy Log Format).
Consumption end	Configure the collection of access logs from the data plane container standard output to CLS. You need to choose the CLS log set and log topic for storing access logs. You can select to automatically create log sets/topics or associate existing log sets/topics. The naming convention for automatically created log sets is `{mesh ID}`, and automatically created log topics will have TCM identifiers, naming convention being `{mesh ID}-accesslog`. Upon enabling the collection of access logs to CLS, the log collection feature for the mesh management cluster will be activated, deploying the tke-log-agent (DaemonSet) in the cluster, and configuring the log collection rules and index for TCM access logs. This feature is based on TKE's Log Collection feature. Ensure that CLS is activated, and the TKE service role `TKE_QCSRole` is associated with the CLS maintenance management preset policy `QcloudAccessForTKERoleInOpsManagement`. For more details, see TKE Role Permission Description.

- Configuring Access Log when creating a mesh:

- Configuring Access Log after mesh creation:



Viewing Access Logs

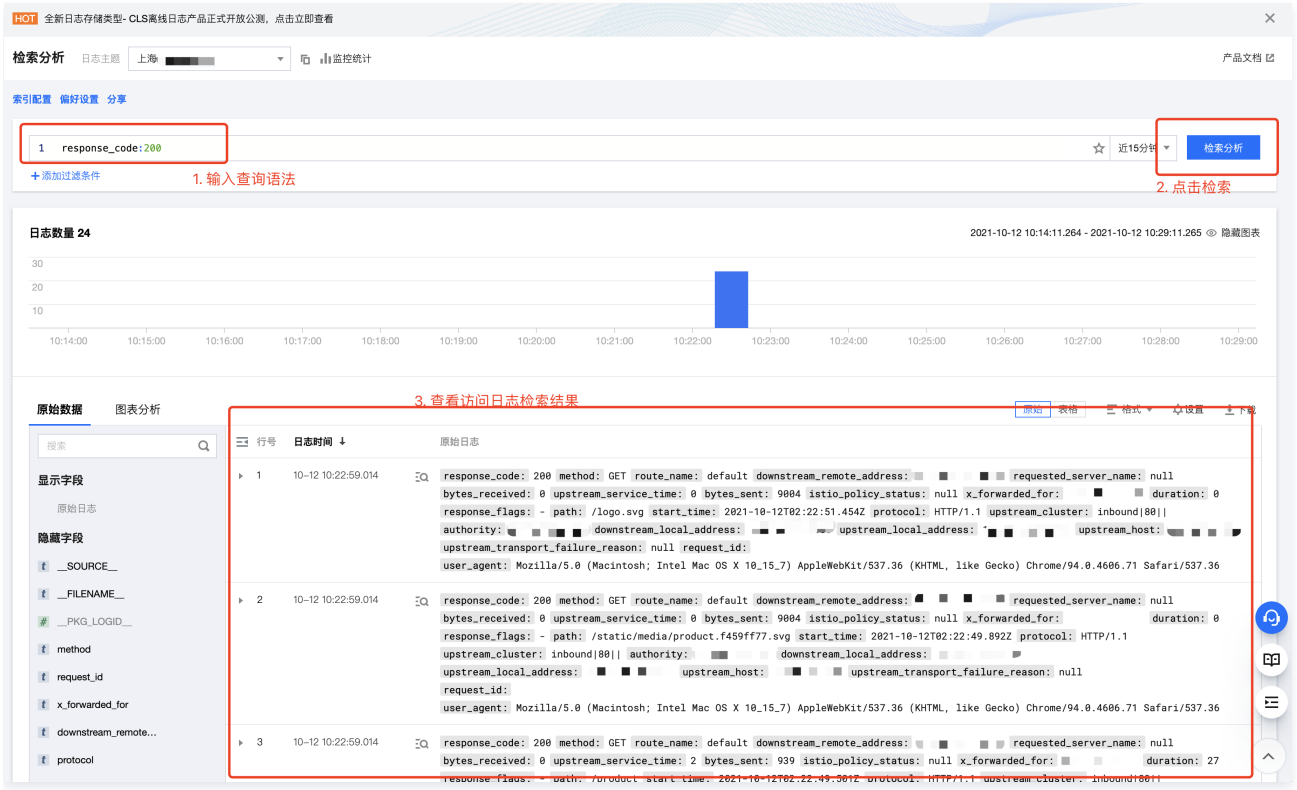
Viewing Access Logs via Container Standard Output

TCM data plane access logs (Access Log) are outputted to container standard output. You can view them through your Kubernetes Cluster API Server by checking the standard output of the istio-proxy container:

```
kubectl -n {namespace} logs {Pod Name} -c istio-proxy --tail 5
```

Viewing Access Logs via CLS Log Retrieval

If you have enabled the consumption end configuration for access logs and collected TCM data plane access logs to CLS, you can view the access logs in the CLS console under log retrieval by selecting the corresponding log topic. For CLS log retrieval syntax, see CLS Log Retrieval Syntax.



Security
Authentication Policy Configuration

Last updated: 2024-08-09 15:40:16

Authentication policies include PeerAuthentication and RequestAuthentication. The PeerAuthentication policy is used to configure the mTLS mode of service communication, and the RequestAuthentication policy is used to configure a request authentication method of a service.

PeerAuthentication Configuration Field Description

The following are important field descriptions for PeerAuthentication:

Field Name	Field Type	Field description
`metadata.name`	`string`	PeerAuthentication name.
`metadata.namespace`	`string`	PeerAuthentication namespace.
`spec.selector`	`map<string, string>`	PeerAuthentication uses the provided tag key-value pairs and the namespace to match the range of workloads for configuration application: • If namespace is set to istio-system and the selector field is not filled, the policy applies to the entire mesh. • If namespace is set to a non-istio-system namespace and the selector field is not filled, the policy applies to the specified namespace. • If namespace is set to a non-istio-system namespace and the selector field is filled with valid key-value pairs, the policy applies to workloads in the specified namespace that match the selector.
`spec.mtls.mode`	-	Configure the mTLS mode, supports: `UNSET`
`spec.portLevelMtls`	`map<uint32, mTLS mode>`	Set the port-level mTLS mode.

Note:
The effective priorities of mTLS mode configurations are as follows: port > service/workload > namespace > mesh.

Using PeerAuthentication to Configure the mTLS Mode for Service Communication in a Mesh

The mTLS mode in Tencent Cloud Mesh is PERMISSIVE by default, that is, the communication between services can be encrypted using mTLS or implemented through plaintext connections.
To test the effect of the mTLS mode configurations, you can first initiate a plaintext request to a service in your mesh and test the connectivity of the plaintext request. The following is an example of logging in to the istio-proxy container in the mesh and initiating a plaintext request to another service:

1. In the console of a TKE cluster managed by the mesh, log in to the istio-proxy container.

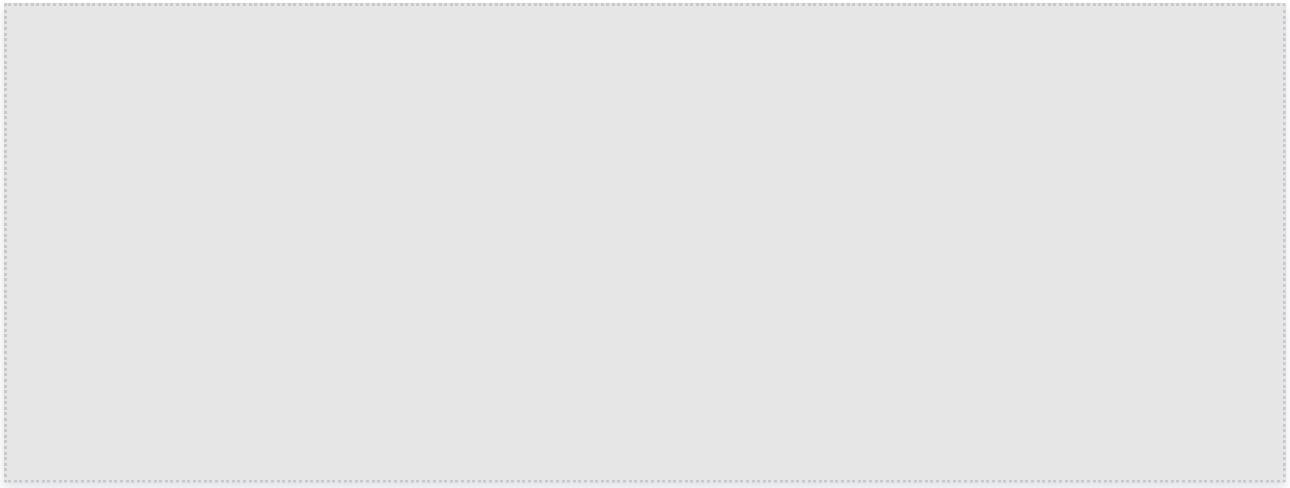
2. Enter the command `curl http://product.base.svc.cluster.local:7000/product` to access the product service in the base namespace in plaintext mode.
3. View the plaintext access result. If the product information is correctly returned, the plaintext access is successful.

Next, we will configure the mTLS mode for the base namespace as STRICT and validate the configuration.

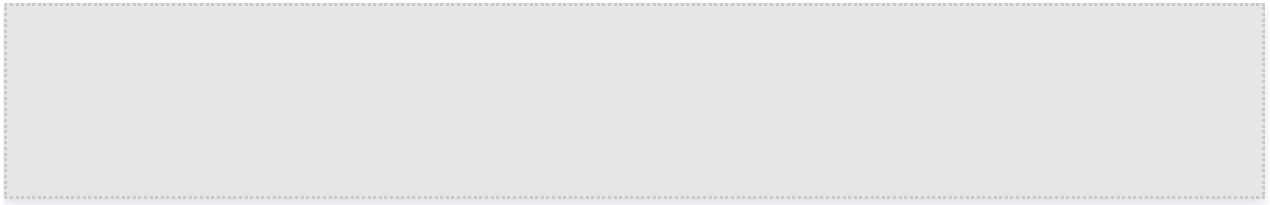
YAML Configuration Example

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: base-strict
  namespace: base
spec:
  mtls:
    mode: STRICT
```

Console Configuration Example



After the configuration is complete, you are prompted that the access fails when you access the product service in the base namespace in the plaintext mode again. This indicates that the mTLS STRICT mode has taken effect.



RequestAuthentication Configuration Field Description

The following are important configuration field descriptions for RequestAuthentication:

Field Name	Field Type	Field description
`metadata.name`	`string`	RequestAuthentication Name.
`metadata.namespace`	`string`	RequestAuthentication Namespace.
`spec.selector`	`map<string, string>`	RequestAuthentication uses the specified Tag key-value pair, in conjunction with the specified namespace, to match the scope of the configured Workload. If the namespace is set to 'istio-system' and the selector field is not filled, the policy applies to the entire mesh. If a non-istio-system namespace is provided and the selector field is not filled, the policy applies to the specified namespace. If a non-istio-system namespace is provided and the selector field contains valid key-value pairs, the policy applies to the Workload matched by the selector within the specified namespace.
`spec.jwtRules.issuer`	`string`	Configure the issuer of the JWT token. For details, see iss claim
`spec.jwtRules.audiences`	`string[]`	Configure the list of allowed JWT audiences. When the audience list is empty, the service name is allowed to access.
`spec.jwtRules.jwksUri`	`string`	Configure the public key URL for verifying the JWT signature. For details, see OpenID Discovery. When both jwksUri and jwks fields are configured, jwksUri will be ignored.
`spec.jwtRules.jwks`	`string`	Verify the JWT signature's JSON Web Key Set public key. When both jwksUri and jwks fields are configured, jwksUri will be ignored.
`spec.jwtRules.fromHeaders`	`map<string,string>[]`	Configure the list of locations from which to extract the JWT from the header.

`spec.jwtRules.fromParams`	`string[]`	Configure the parameters for extracting the JWT from the header, such as extracting from the parameter 'mytoken' (`/path?my_token=`).
`spec.jwtRules.outputPayloadToHeader`	`string`	Configure the header name for the successfully validated JWT payload output. The forwarded data is `base64_encoded(jwt_payload_in_JSON)` . If not filled, the JWT payload will not be output by default.
`spec.jwtRules.forwardOriginalToken`	`bool`	Configure whether to forward the original JWT to the upstream. The default value is `false` .

Using RequestAuthentication to Configure JWT Request Authentication

To verify the effect of the configured JWT authentication for requests, you first need to deploy a test program `httpbin.foo` , and configure it to expose this service to the public through the Ingress Gateway:

- Create the foo namespace, enable Sidecar automatic injection, and deploy the httpbin service to the foo namespace:

```
apiVersion: v1
kind: Namespace
metadata:
  name: foo
  labels:
    istio.io/rev: 1-6-9 # Enable Sidecar automatic injection for the namespace (Istio version 1.6.9)
spec:
  finalizers:
    - kubernetes
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: httpbin
  namespace: foo
---
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  namespace: foo
  labels:
    app: httpbin
    service: httpbin
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
  namespace: foo
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
```

- Expose the httpbin service to public access through Ingress Gateway:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
  namespace: foo
spec:
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
  namespace: foo
spec:
  hosts:
  - "*"
  gateways:
  - httpbin-gateway
  http:
  - route:
    - destination:
        port:
          number: 8000
        host: httpbin.foo.svc.cluster.local
```

- Test the service connectivity with the curl command `curl "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"`. Note that you need to replace `$INGRESS_IP` in the code with your edge proxy gateway IP address. Normally, it would return the `200` status code.

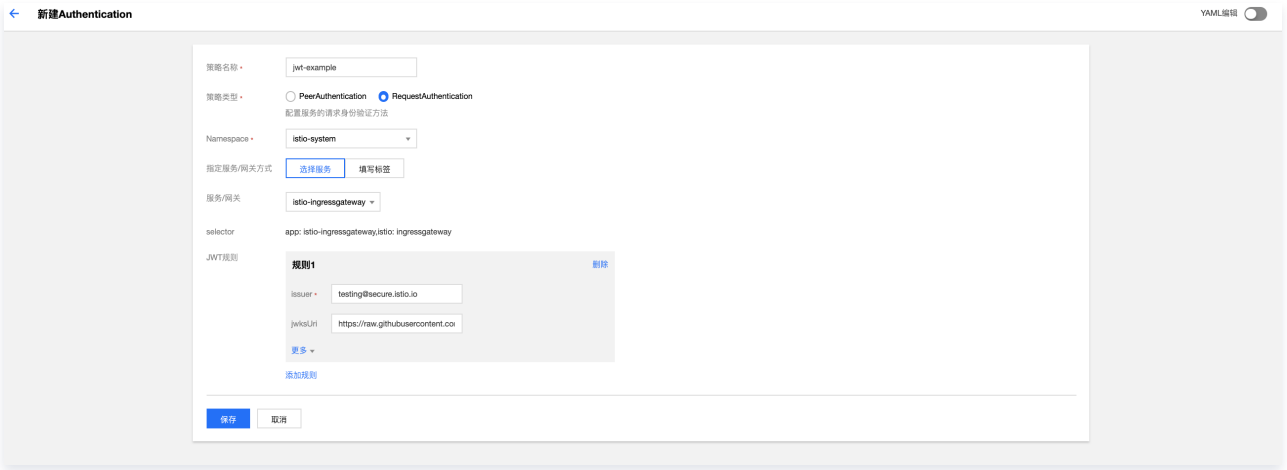
The following configures JWT authentication rules for the ingress gateway to allow requests carrying eligible JWT tokens.

YAML Configuration Example

```
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
```

```
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
      app: istio-ingressgateway
  jwtRules:
  - issuer: "testing@secure.istio.io"
    jwksUri: "https://raw.githubusercontent.com/istio/istio/release-
1.9/security/tools/jwt/samples/jwks.json"
```

Console Configuration Example



After the configuration is complete, verify whether the configured JWT authentication rule takes effect.

- Access the service using the following code with an invalid JWT token. Note that you need to replace `$INGRESS_IP` in the code with your edge proxy gateway IP address. The edge proxy gateway will not allow requests with invalid JWT tokens, thus it will return a `401` status code.

```
curl --header "Authorization: Bearer deadbeef" "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"
```

- Access the service using the following code with a valid JWT token. Note that you need to replace `$INGRESS_IP` in the code with your edge proxy gateway IP address. The edge proxy gateway will allow requests with valid JWT tokens, thus it will return a `200` status code.

```
TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-1.9/security/tools/jwt/samples/demo.jwt
-s)
curl --header "Authorization: Bearer $TOKEN" "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"
```

Through verification, you can find that the JWT authentication rule configured for your Edge Proxy Gateway has taken effect. However, at this stage, only the JWT authentication rule is configured, and Ingress Gateway will still allow requests that do not carry a JWT token. To restrict requests that do not carry a JWT token, you need to configure AuthorizationPolicy. Apply the following YAML file to TCM to restrict the Ingress Gateway from accepting requests that do not carry a JWT token:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: frontend-ingress
  namespace: istio-system
spec:
  selector:
```

```
    matchLabels:
      app: istio-ingressgateway
      istio: ingressgateway
  rules:
    - from:
        - source:
            notRequestPrincipals:
              - '*'
  action: DENY
```

Attempt access again without carrying a JWT token with the command `curl "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"`, and find that the access is denied, returning a `403` status code, indicating the AuthorizationPolicy policy is in effect.

Authorization Policy Configuration

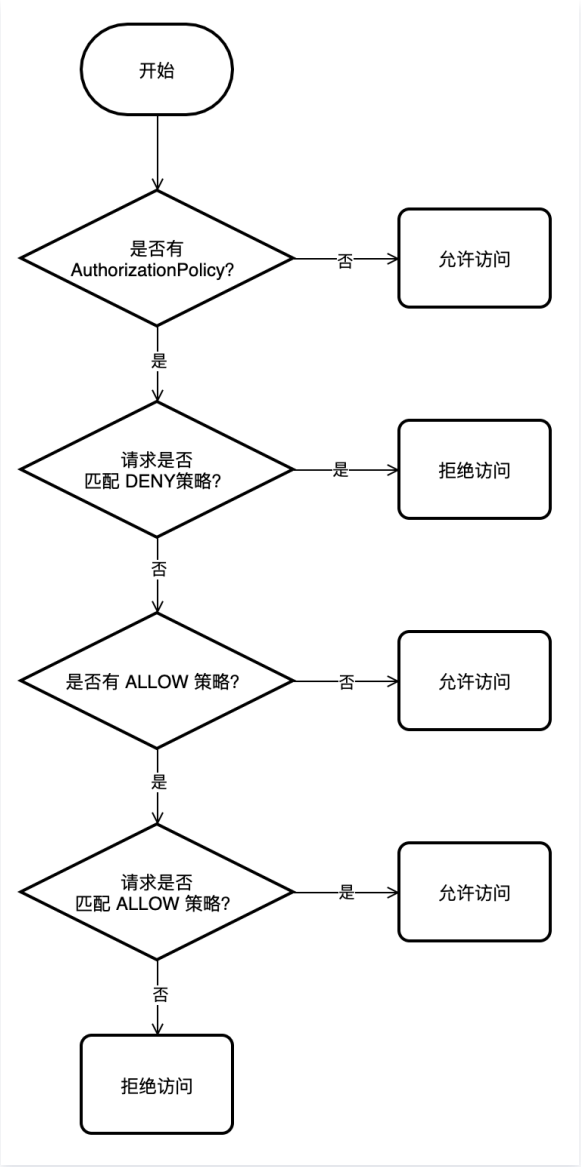
Last updated: 2024-08-09 15:40:32

An authorization policy is used to configure access management rules in scopes such as a mesh, namespace, and service/workload. You can configure authorization rules by using an AuthorizationPolicy CRD. AuthorizationPolicy includes the following parts:

- selector: specifies the effective range of the policy.
- action: specifies whether the policy is an `ALLOW` policy or a `DENY` policy.
- rules: specifies an authorization rule body, consisting of from, to, and where.
  - from: specifies the source of a request.
  - to: specifies the operation of a request.
  - when: specifies a condition for an authorization rule to take effect.

When `ALLOW` and `DENY` policies of AuthorizationPolicy are applied to the same range, the `DENY` policy takes precedence over the `ALLOW` policy. The effective rules are as follows:

1. If any `DENY` policy matches the request, deny the request.
2. If there are no `ALLOW` policies for the scope, allow the request.
3. If there are any `ALLOW` policies for the scope and any of the `ALLOW` policies match the request, allow the request.
4. Access to this request is denied.



The following are two special AuthorizationPolicy examples:

- Services in the default namespace allow all requests:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
name: allow-all
namespace: default
spec:
action: ALLOW
rules:
- {} # Rules can match any request
```

- Services in the default namespace deny all requests:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
name: deny-all
namespace: default
spec:
{} # The default action is ALLOW if the action field is not filled, thus requests cannot match any rules
```

Description of Major AuthorizationPolicy Fields

The following are descriptions of important fields in AuthorizationPolicy:

Field Name	Field Type	Field description
`metadata.name`	`string`	AuthorizationPolicy name.
`metadata.namespace`	`string`	AuthorizationPolicy namespace.
`spec.selector`	`map<string, string>`	AuthorizationPolicy uses the provided Tag key-value pairs and the specified namespace to match the configured workload range • If namespace is set to istio-system and the selector field is not filled, the policy applies to the entire mesh. • If namespace is set to a non-istio-system namespace and the selector field is not filled, the policy applies to the specified namespace. • If namespace is set to a non-istio-system namespace and the selector field is filled with valid key-value pairs, the policy applies to workloads in the specified namespace that match the selector.
`spec.action`	-	Specifies whether the policy is an `ALLOW` policy or a `DENY` policy.
`spec.rules.from.source.principals`	`string[]`	Source peer identity list (i.e., service account), matches the `source.principal` field. Requires mTLS. If not specified, any principal is allowed.
`spec.rules.from.source.requestPrincipals`	`string[]`	Request identity list (i.e., iss/sub claim), matches the `request.auth.principal` field. If not specified, any requestPrincipals are allowed.
`spec.rules.from.source.namespaces`	`string[]`	Request source namespace list, matches the `source.namespace` field. Requires mTLS. If not specified, requests from any namespace are allowed.
`spec.rules.from.source.ipBlocks`	`string[]`	IP block list, matches the `source.ip` field. Supports single IP (e.g., `1.2.3.4`) or CIDR notation (e.g., `1.2.3.4/24`). If not specified, any source IP is allowed.
`spec.rules.to.operation.hosts`	`string[]`	Request domain name list, matches the `request.host` field. If not specified, any domain is allowed. Only supported for HTTP protocol requests.
`spec.rules.to.operation.ports`	`string[]`	Request port list, matches the `destination.port` field. If not specified, any port is allowed.

`spec.rules.to.operation.methods`	`string[]`	Request method list, matches the `request.method` field. For gRPC protocol, this value should always be `POST`. If not specified, any method is allowed. Only supported for HTTP protocol requests.
`spec.rules.to.operation.paths`	`string[]`	Request path, matches the `request.url_path` field. If not specified, any path is allowed. Only supported for HTTP protocol requests.
`spec.rules.when.condition.key`	`string`	Condition field names supported by Istio, see Authorization Policy Conditions for details
`spec.rules.when.condition.values`	`string[]`	Fill in the list of values corresponding to the conditions.

Using AuthorizationPolicy to Configure Namespace Access Permissions

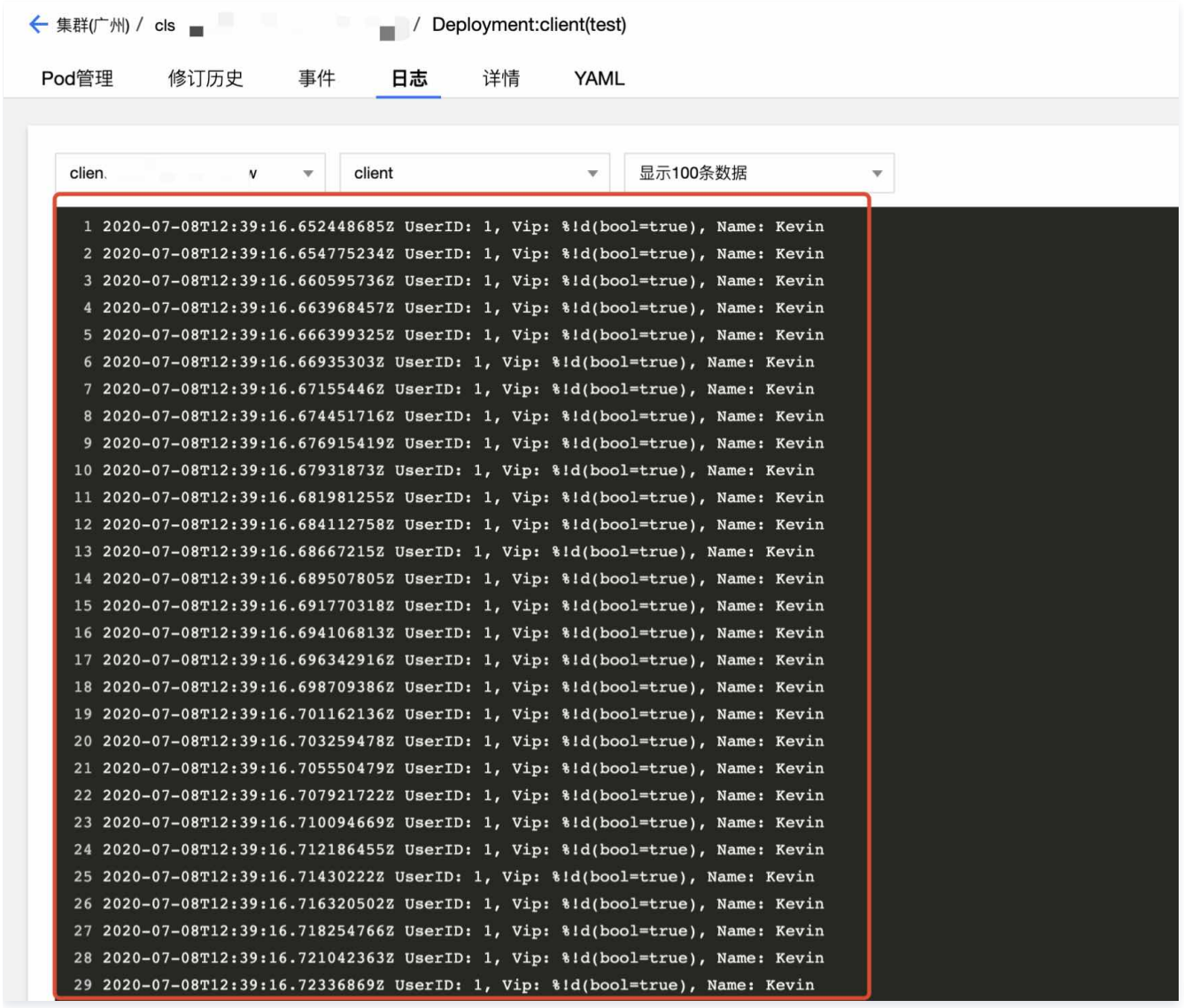
To view the effect of the configured AuthorizationPolicy, we first deploy a test program to a mesh-managed cluster. After deployment, the client service in the test namespace will automatically initiate access to the user service in the base namespace:

```
apiVersion: v1
kind: Namespace
metadata:
  name: test
  labels:
    istio.io/rev: 1-6-9 # Automatic sidecar injection (Istio 1.6.9)
spec:
  finalizers:
    - kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: client
  namespace: test
  labels:
    app: client
spec:
  replicas: 10
  selector:
    matchLabels:
      app: client
  template:
    metadata:
      labels:
        app: client
    spec:
      containers:
        - name: client
          image: ccr.ccs.tencentyun.com/zhulei/testclient:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: REGION
              value: "guangzhou-zoneA"
          ports:
            - containerPort: 7000
              protocol: TCP
---

apiVersion: v1
```

```
kind: Service
metadata:
  name: client
  namespace: test
  labels:
    app: client
spec:
  ports:
    - name: http
      port: 7000
      protocol: TCP
  selector:
    app: client
  type: ClusterIP
---
apiVersion: v1
kind: Namespace
metadata:
  name: base
  labels:
    istio.io/rev: 1-6-9
spec:
  finalizers:
    - kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user
  namespace: base
  labels:
    app: user
spec:
  replicas: 1
  selector:
    matchLabels:
      app: user
  template:
    metadata:
      labels:
        app: user
    spec:
      containers:
        - name: user
          image: ccr.ccs.tencentyun.com/zhulei/testuser:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: REGION
              value: "guangzhou-zoneB"
          ports:
            - containerPort: 7000
---

apiVersion: v1
kind: Service
metadata:
  name: user
  namespace: base
```

```
  labels:
    app: user
spec:
  ports:
    - port: 7000
      name: http
  selector:
    app: user
```

Check the logs of the client container and you will see successful access, correctly returning user information:



Next, configure the Authorization policy to disallow services in the test namespace from accessing services in the base namespace (mTLS must be enabled).

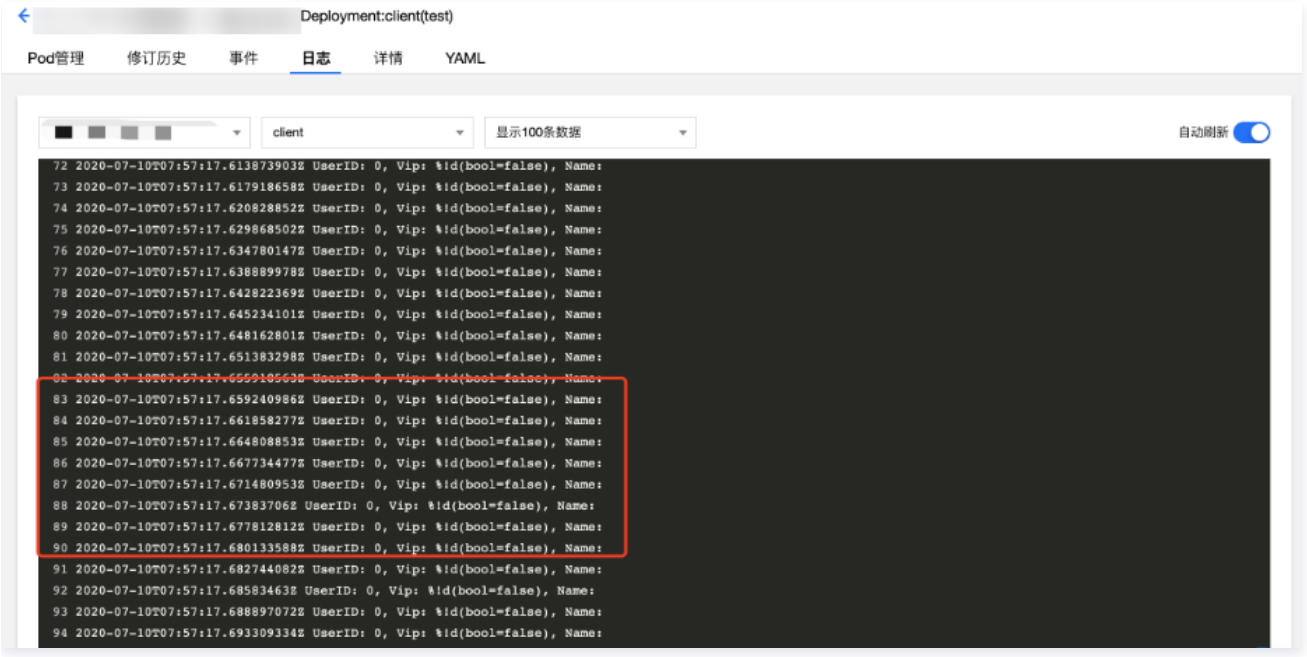
YAML Configuration Example

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: base-authz
  namespace: base
spec:
  action: DENY
```

```
rules:
  - from:
      - source:
          namespaces:
            - test
```

Console Configuration Example

After the configuration is complete, view logs of the client container again. It is found that all access requests fail and no user information is returned, indicating that AuthorizationPolicy has taken effect.



Using AuthorizationPolicy to Configure an IP Blocklist/Allowlist of the Ingress Gateway

You can use AuthorizationPolicy to configure an IP blocklist/allowlist for the ingress gateway.
To verify the effect of the black/allowlist configuration, you first need to deploy a test program `httpbin.foo`, and expose this service to the public through the Ingress Gateway:

- Create the foo namespace, enable Sidecar automatic injection, and deploy the httpbin service to the foo namespace:

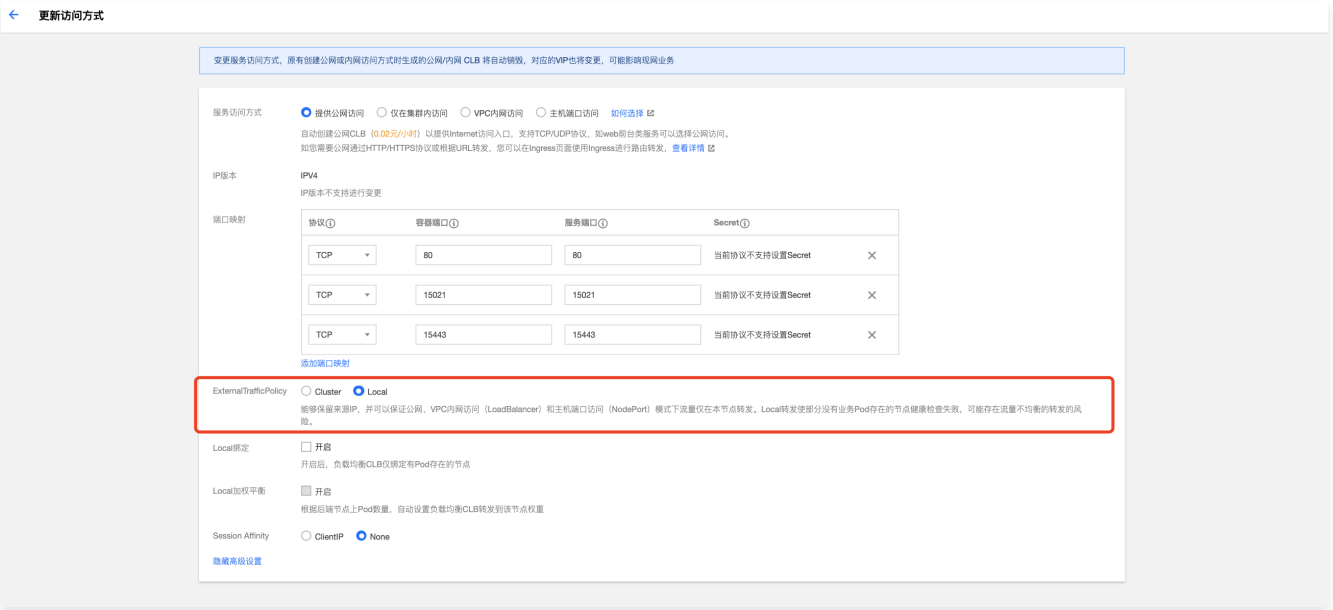
```
apiVersion: v1
kind: Namespace
metadata:
  name: foo
  labels:
    istio.io/rev: 1-6-9 # Enable Sidecar automatic injection for the namespace (Istio version 1.6.9)
spec:
  finalizers:
    - kubernetes
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: httpbin
  namespace: foo
---
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  namespace: foo
  labels:
    app: httpbin
    service: httpbin
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
  namespace: foo
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
```

- Expose the httpbin service to public access through Ingress Gateway:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
```

```
metadata:
  name: httpbin-gateway
  namespace: foo
spec:
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
  namespace: foo
spec:
  hosts:
  - "*"
  gateways:
  - httpbin-gateway
  http:
  - route:
    - destination:
        port:
          number: 8000
        host: httpbin.foo.svc.cluster.local
```

- Test the service connectivity with the curl command `curl "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"`. Note that you need to replace `$INGRESS_IP` in the code with your edge proxy gateway IP address. Normally, it would return the `200` status code.
- To enable the Ingress Gateway to correctly obtain the real client source IP, we need to set the ExternalTrafficPolicy of the Ingress Gateway Service to Local, ensuring that the traffic is forwarded on this node only without SNAT.

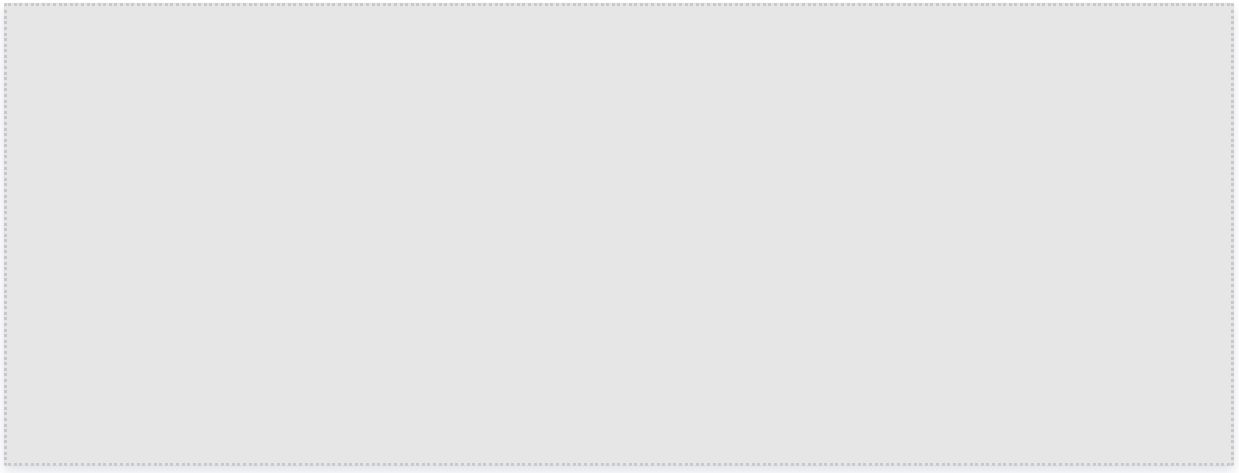


The following uses AuthorizationPolicy to add the IP address of the local host to the blocklist of the ingress gateway, and verify whether the blocklist takes effect.

YAML Configuration Example

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: black-list
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
      istio: ingressgateway
  rules:
    - from:
        - source:
            ipBlocks:
              - $ IP address of your local host
  action: DENY
```

Console Configuration Example



After the configuration is complete, test the connectivity of the service again using the curl statement `curl "$INGRESS_IP:80/headers" -s -o /dev/null -w "%{http_code}\n"`. Note that you need to replace `$INGRESS_IP` in the code with the IP address of your edge proxy gateway. In this case, the access fails and a `403` return code is returned, indicating that the blocklist policy has taken effect.

Access Management Overview

Last updated: 2024-08-09 15:40:57

TCM's permission management consists of 2 parts: CAM (CAM) permissions and TKE (TKE) RBAC permissions.
By default, a sub-account does not have CAM permissions, and a sub-account that is not a cluster creator does not have RBAC permissions for the related cluster. You need to create and associate CAM policies and TKE RBAC authorization policies to allow sub-accounts to access or normally use service mesh resources they need.
The editing and granting of CAM permission policies are completed by CAM administrators (usually the main account or a sub-account with CAM permissions). For more information on basic CAM policies, please see CAM policies. The editing and granting of TKE cluster's RBAC permission policies are usually done by the respective cluster administrators (usually the main account or the cluster creation account), see TKE RBAC Authorization for authorization methods.

> **Note:**
> Skip this chapter if you do not need to manage the access permission of sub-accounts for Tencent Cloud Mesh resources. This will not affect your understanding and use of the other sections of the document.

CAM-based Permission Control

Currently, TCM supports resource-level permission control based on CAM, which allows specific **sub-accounts** to perform specific **operations** on designated **resources**. By default, sub-accounts do not have TCM-related CAM permissions, and you need to associate the policy with a sub-account to complete the authorization.
In addition, Tencent Cloud Mesh supports CAM-based resource-level permission control at a granularity of mesh instance. In other words, you can control specified sub-account to perform specified operations on a specified mesh.

RBAC permission management of TKE, a product related to Tencent Cloud Mesh

The use of Tencent Cloud Mesh involves read and write operations on Kubernetes resources in the TKE clusters managed by Tencent Cloud Mesh. These operations require sufficient TKE RBAC permissions are available. By default, a sub-account that is not the cluster creator does not have the RBAC permissions for the cluster. The cluster administrator needs to grant the RBAC permissions for the corresponding cluster to the sub-account before the sub-account can use Tencent Cloud Mesh normally.
The following operations require administrator (tke:admin) permissions for the corresponding cluster: creating/deleting/updating a service mesh in the selected cluster, adding/dissociating a service discovery cluster, and creating/deleting an ingress gateway in the selected cluster. Operations on Istio resources (such as Gateway, VirtualService, DestinationRule, and ServiceEntry) in the mesh do not require RBAC permissions for the cluster.
For more information on TKE Kubernetes object-level permission control, please see TKE Kubernetes Object-Level Permission Control. For TKE RBAC authorization methods, please refer to Authorization Mode Comparison.

CAM Service Role Authorization

Last updated: 2024-08-09 15:44:17

In the process of using Tencent Cloud TCM (Tencent Cloud Mesh, TCM), it involves the usage of TCM-related cloud resources. To use TCM features normally, you need to authorize the TCM service role `TCM_QCSRole`. After authorization, the TCM service can use the related cloud resources.
Scenarios that require service authorization primarily include initial log in to TCM Console and initial use of TCM one-click experience feature. These scenarios correspond to two preset policies, `QcloudAccessForTCMRole` and `QcloudAccessForTCMRoleInSampleDeployment`, respectively.

Initial Login to the Tencent Cloud Mesh Console

Authorization scenario

Once you have registered and logged in to your Tencent Cloud account, the first time you log in to the TCM console, you will need to go to the **CAM** page to grant your account permissions to operate Tencent Cloud TCM on cloud resources like Tencent Kubernetes Engine (TKE), SSL Certificates (SSL), and Cloud Log Service (CLS). This permission is granted by associating the preset policy `QcloudAccessForTCMRole` with the TCM service role `TCM_QCSRole`. If you have not created a TCM service role before, this authorization process will also involve creating the TCM service role.

Authorization steps

1. The first time you log in to the TCM console, a **service authorization** popup will automatically appear.

2. Click **Go to CAM** to access the CAM console service authorization page.
3. Click **Agree to authorize**, and upon successful identity verification, the authorization will be completed.

Permission content

TKE related

Permissions	Description	Resources
DescribeClusterSecurity	Querying cluster keys	All resources *

SSL related

Permissions	Description	Resources
DescribeCertificateDetail	Obtaining certificate details	All resources *

CLS related

Permissions	Description	Resources
getLogset	Obtaining logset details	All resources *
getTopic	Obtaining log topic details	All resources *
createLogset	Creating a logset	All resources *
createTopic	Creating a log topic	All resources *
modifyIndex	Modifies indexes	All resources *
listLogset	Getting the list of logsets	All resources *
listTopic	Getting the list of log topics	All resources *

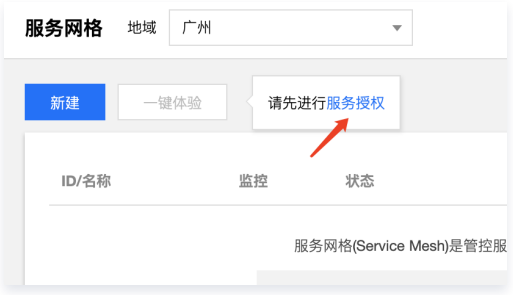
First use TCM one-click experience feature

Authorization scenario

When you use the TCM one-click experience feature for the first time, you need to grant your current Tencent Cloud TCM account permissions to operate cloud resources such as Virtual Private Cloud (VPC), Cloud Connect Network (CCN), Tencent Kubernetes Engine (TKE), Cloud Virtual Machine (CVM), and to purchase Cloud Virtual Machine (CVM) financial permissions via **CAM**. The permissions are granted by associating the preset policy `QcloudAccessForTCMRoleInSampleDeployment` with the TCM service role `TCM_QCSRole`. If you haven't created a TCM service role before, this authorization process also involves creating a TCM service role.

Authorization steps

1. Log in to TCM console, hover over the **one-click experience** button, and click **service authorization** to open the authorization prompt window.



2. Click **go to CAM** to enter the service authorization page of the CAM console.

3. Click **Agree to authorize**, and upon successful identity verification, the authorization will be completed.

Permission content

VPC related

Permissions	Description	Resources
CreateSubnet	Create a subnet	All resources *
CreateVpc	Create a VPC	All resources *
DeleteVpc	Delete a VPC	All resources *
DescribeVpcEx	Query the VPC list	All resources *
DescribeSubnetEx	View subnet list	All resources *

CCN related

Permissions	Description	Resources
AttachCcnInstances	Associates CCN instances	All resources *
CreateCcn	Create CCN	All resources *
DeleteCcn	Delete CCN	All resources *
DescribeCcnAttachedInstances	Query the list of CCN associated instances	All resources *
DescribeCcns	Query CCN list	All resources *

TKE related

Permissions	Description	Resources
CreateCluster	Creating cluster	All resources *
CreateClusterEndpointVip	Enable cluster access port (Managed Cluster External Network)	All resources *
DeleteCluster	Deleting cluster	All resources *
DeleteClusterEndpoint	Delete cluster access port	All resources *
DeleteClusterEndpointVip	Delete cluster access port (Managed	All resources *

	Cluster External Network)	
DescribeClusterEndpointVipStatus	Query cluster access port status (Managed Cluster External Network)	All resources `*`
DescribeClusters	Get cluster list	All resources `*`

CVM related

Permissions	Description	Resources
CreateSecurityGroup	Creating security group	All resources `*`
DeleteSecurityGroup	Deleting Security Groups	All resources `*`
DescribeImages	Query an image	All resources `*`
DescribeInstances	Query cloud host	All resources `*`
DescribeSecurityGroups	Querying Security Group	All resources `*`
ModifySecurityGroupPolicys	Modifying a Security Group Rule	All resources `*`
RunInstances	Create a cloud host	All resources `*`

Financial permissions

Permissions	Description	Resources
`finance:*`	Financial permissions for purchasing CVM	CVM `qcs::cvm:::*`

CAM Preset Policy Authorization

Last updated: 2024-08-09 15:44:36

You can associate the TCM-related predefined strategies in CAM to sub-accounts to quickly complete the CAM authorization for TCM.

TCM Related Predefined Strategies

You can grant relevant permissions to sub-accounts by using the following preset policies:

Policies	Description
`QcloudTCMFullAccess`	TCM Full Read-Write Access (create, delete, and all other operations)
`QcloudTCMReadOnlyAccess`	TCM Read-only Access (can view all resources under TCM but cannot create, update, or delete resources)

TCM Full Read-Write Preset Strategy

Policy name: QcloudTCMFullAccess, Policy content:

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "tcm:*"
            ],
            "resource": "*",
            "effect": "allow"
        }
    ]
}
```

TCM Read-only Preset Strategy

Policy name: QcloudTCMReadOnlyAccess, Policy content:

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "tcm:List*",
                "tcm:Describe*",
                "tcm:ForwardRequestRead"
            ],
            "resource": "*",
            "effect": "allow"
        }
    ]
}
```

CAM Permissions for TCM-related Products

Using TCM also involves corresponding CAM permissions for related products such as VPC, CCN, CLB, and TKE. You can refer to the CAM authorization documentation for the respective products to grant appropriate permissions to sub-accounts:

TCM Related Products	Authorization Guide Documentation
Virtual Private Cloud (VPC)	VPC CAM Overview

Cloud Load Balancer (CLB)	CLB CAM Overview
Tencent Kubernetes Engine (TKE)	TKE Permission Management Overview

Associating Sub-accounts with Preset Policies

You can associate a preset policy to the sub-account in the "Set User Permissions" step by either direct association or group association.

Direct association

You can directly associate a policy with the sub-account to obtain the permissions included in the policy.

1. Log in to the CAM console, select **Users** > **User List** from the left sidebar.
2. On the **User List** management page, select the sub-account for which you want to set permissions, and click **Authorize** on the right side of the row.
3. In the popped-up "Associate Policy" window, check the policy that needs authorization.
4. Click **OK**.

Association via group

You can add the sub-account to a user group, and it will automatically obtain the permissions associated with that user group. If you need to disassociate the policy associated with the group, simply remove the sub-account from the corresponding user group.

1. Log in to the CAM console, select **Users** > **User List** from the left sidebar.
2. On the **User List** management page, select the sub-account for which you want to set permissions, and click **More Operations** > **Add to Group** on the right side of the row.
3. In the popped-up "Add to Group" window, check the user group to join.
4. Click **OK**.

Logging in to the sub-account for verification

Log in to the Tencent Cloud TCM Console, and verify that the sub-account can use the features corresponding to the authorized policies, indicating that the authorization was successful.

CAM Custom Policy Authorization

Last updated: 2024-08-09 15:44:46

If you have custom permission management requirements, you can create a custom CAM policy and associate it with a sub-account to implement custom authorization. You can perform configuration based on actual service requirements by referring to the following description.

CAM Element Reference

Core elements of a CAM custom policy include: action, resource, condition, and effect.

1. Action

This required element describes allowed or denied actions. An action can be an API (described with a name prefix) or a feature set (a group of specific APIs, described with an actionName prefix). You can view CAM APIs accessed by TCM.

2. Resource

This element describes specific data that is to be authorized. A resource is described in six paragraphs. You can view Tencent Cloud Mesh resource description.

3. Condition

This element describes the condition for the policy to take effect. A condition consists of operator, action key, and action value. A condition value may contain information such as time and IP address.

4. Effect

This required element describes whether the statement result is an "allow" or "explicit deny".

5. Custom policy sample

This policy defines that it is allowed to obtain details about two mesh instances mesh-abcd1234 and mesh-1234abcd in Guangzhou.

```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
            "resource": [
                "qcs::tcm:gz:uin/1234567:mesh/mesh-abcd1234",
                "qcs::tcm:gz:uin/1234567:mesh/mesh-1234abcd"
            ],
            "action": [
                "name/tcm:DescribeMesh"
            ]
        }
    ]
}
```

For more information about syntax logic of CAM custom policies, see CAM Syntax Logic.

Tencent Cloud Mesh Resources That Can Be Authorized on CAM

Resources	Resource Description Method in Access Policies
Service mesh	`qcs::tcm:$region:$account:mesh/$meshid`

Where:

- `$region`: describes region information. It is an ID of a region. For example, `gz` is the ID of Guangzhou.
- `$account`: describes root account information about a resource owner. It is expressed in the `uin/${uin}` format, for example, `uin/12345678`. If this field is left blank, it indicates the root account to which the CAM user who creates the policy belongs.

- `$meshid`: describes mesh instance information. It is an ID of a mesh, or `*`.

For information on how to describe resources in authorization policies, see Resource Description Method.

Interfaces in CAM to authorize TCM

In CAM, you can authorize the following actions on TCM mesh resources.

Related to mesh instances

Operations through API	API Description	Resources
CreateMesh	Creating a service mesh	mesh resources `qcs::tcm:$region:$account:mesh/*`
DeleteMesh	Deleting a service mesh	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
DescribeMesh	Obtaining a specified service mesh	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
ListMeshes	Obtaining a service mesh list	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
ModifyMesh	Modifying service mesh configurations	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
UpgradeMesh	Upgrading a service mesh	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`

Istio Resource

Operations through API	API Description	Resources
ForwardRequestRead	Reading Istio CRD resources	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
ForwardRequestWrite	Writing Istio CRD resources	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`

Service Discovery

Operations through API	API Description	Resources
LinkClusterList	Associating a cluster with a service mesh instance	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
UnlinkCluster	Disassociating a cluster	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`

Edge Proxy Gateway related

Operations through API	API Description	Resources
CreateIngressGateway	Create IngressGateway	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
DeleteGatewayInstance	Delete IngressGateway	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
DescribeIngressGatewayList	Querying an ingress gateway list	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`
ModifyIngressGateway	Modify IngressGateway	mesh resources `qcs::tcm:$region:$account:mesh/$meshid`

Experience Environment related

Operations through API	API Description	Resources

CreateTrial	Create Tencent Cloud Mesh sample deployment	Authorizing only interfaces *
DeleteTrial	Delete Tencent Cloud Mesh sample deployment	Authorizing only interfaces *
RetryTrialTask	Retrying creating Tencent Cloud Mesh sample deployment	Authorizing only interfaces *

Extended Features
Using a Wasm Filter o Extend the Data Plane

Last updated: 2024-08-09 15:45:08

Wasm is short for WebAssembly, which can compile binary instructions and load them into the Envoy's filter chain to extend mesh data plane capabilities. In this way, Envoy and extension components are decoupled, and users no longer need to extend capabilities by modifying Envoy code and compiling special Envoy versions. In addition, wasm delivers advantages of dynamic loading and secure isolation.
From Istio version 1.6 onwards, the Proxy-Wasm Sandbox API replaced Mixer as Istio's primary extension implementation solution, used to enable interaction between Envoy and the Wasm virtual machine. Therefore, extending Envoy through a wasm filter requires using the Proxy-WASM SDK.

Usually, steps of compiling a wasm file to extend mesh data plane capabilities include the following:

1. Compile a wasm filter by following Examples.
2. Inject the wasm filter into a ConfigMap to mount the wasm filter to any workload through the ConfigMap, thereby preventing the wasm filter from being copied to multiple nodes.

```
kubectl create cm -n foo example-filter --from-file=example-filter.wasm
```

3. To mount the wasm filter to business workloads, you can utilize Istio's annotation mechanism to automatically mount the corresponding files when creating the workload:

```
sidecar.istio.io/userVolume: '[{"name":"wasmfilters-dir","configMap": {"name": "example-filter"}}]'
sidecar.istio.io/userVolumeMount: '[{"mountPath":"/var/local/lib/wasm-filters","name":"wasmfilters-dir"}]'
```

Apply the annotation to the corresponding workload:

```
kubectl patch deployment -n foo frontpage-v1 -p '{"spec":{"template":{"metadata":{"annotations":
{"sidecar.istio.io/userVolume":"[{\"name\":\"wasmfilters-dir\",\"configMap\": {\"name\": \"example-
filter\"}}]","sidecar.istio.io/userVolumeMount":"[{\"mountPath\":\"/var/local/lib/wasm-
filters\",\"name\":\"wasmfilters-dir\"}]"}}}}}'
```

4. Create an Envoy filter, and add the wasm filter to the Envoy filter chain of the corresponding workload to have it to take effect.

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: frontpage-v1-examplefilter
  namespace: foo
spec:
  configPatches:
  - applyTo: HTTP_FILTER
    match:
      listener:
        filterChain:
          filter:
            name: envoy.http_connection_manager
            subFilter:
              name: envoy.router
    patch:
      operation: INSERT_BEFORE
      value:
        name: envoy.filters.http.wasm
        typed_config:
          '@type': type.googleapis.com/envoy.extensions.filters.http.wasm.v3.Wasm
          config:
```

```
            name: example-filter
            root_id: my_root_id
            vm_config:
              code:
                local:
                  filename: /var/local/lib/wasm-filters/example-filter.wasm
              runtime: envoy.wasm.runtime.v8
              vm_id: example-filter
              allow_precompiled: true
    workloadSelector:
      labels:
        app: frontpage
        version: v1
```

Till now, the wasm filter has been deployed. The wasm filter can also be used as an image. For details, see Build a wasm filter image. For details about how to use the WASME tool to deploy the wasm filter, see Deploying Wasm Filters with Wasme.