

Tencent Cloud Mesh

Getting Started

Product Documentation



Copyright Notice

©2013–2025 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by the Tencent corporate group, including its parent, subsidiaries and affiliated companies, as the case may be. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Getting Started

- Overview

- Demo Application Deployment

- Application Configuration

 - Configuring Public Network Access

- Traffic Control

 - Multi-version Routing

 - Canary Release

- Service HA

 - Fault Injection Testing

 - Service Timeout Configuration

 - Session Persistence

 - Connection Pool-based Concurrency Limiting

- Multi-cluster Disaster Recovery and Multi-active

 - Cross-Cluster Disaster Recovery

 - Locality Load Balancing

 - Nearby Access

- Security Reinforcement

 - Authentication

 - Authorization

Getting Started

Overview

Last updated: 2023-12-26 10:42:13

This document describes how to deploy and manage the demo application to help you quickly get started with Tencent Cloud Mesh, including common operations such as high service availability configuration, traffic management, multi-cluster disaster recovery, active-active deployment, and security enhancement.

Prerequisites

- You have created a service mesh instance as instructed in [Creating Mesh](#).
- You have deployed the Tencent Cloud Mesh demo application as instructed in [Deploying Demo Application](#).

Directions

After deploying the demo application, quickly get started with common features of a mesh as instructed below.

- [Configuring Public Network Access](#)
- [Multi-Version Routing](#)
- [Canary Release](#)
- [Fault Injection Test](#)
- [Service Timeout Configuration](#)
- [Session Persistence](#)
- [Limiting Concurrency Through Connection Pool](#)
- [Cross-Cluster Service Disaster Recovery](#)
- [Locality Load Balancing](#)
- [Nearby Access](#)
- [Authentication](#)
- [Authorization](#)

If you encounter any problems during deployment, [submit a ticket](#) for assistance.

Demo Application Deployment

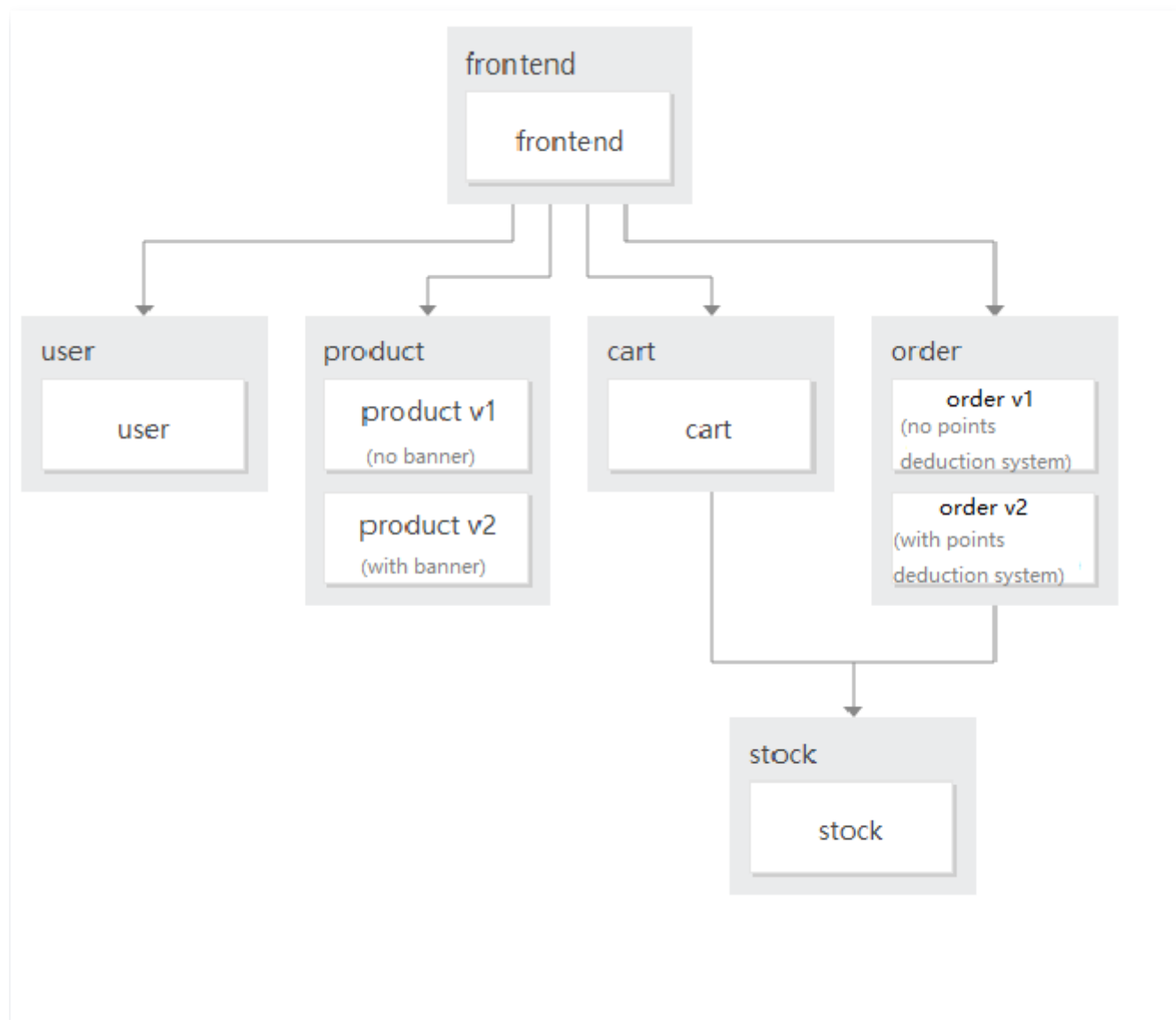
Last updated: 2023-12-26 10:42:45

Demo Application Overview

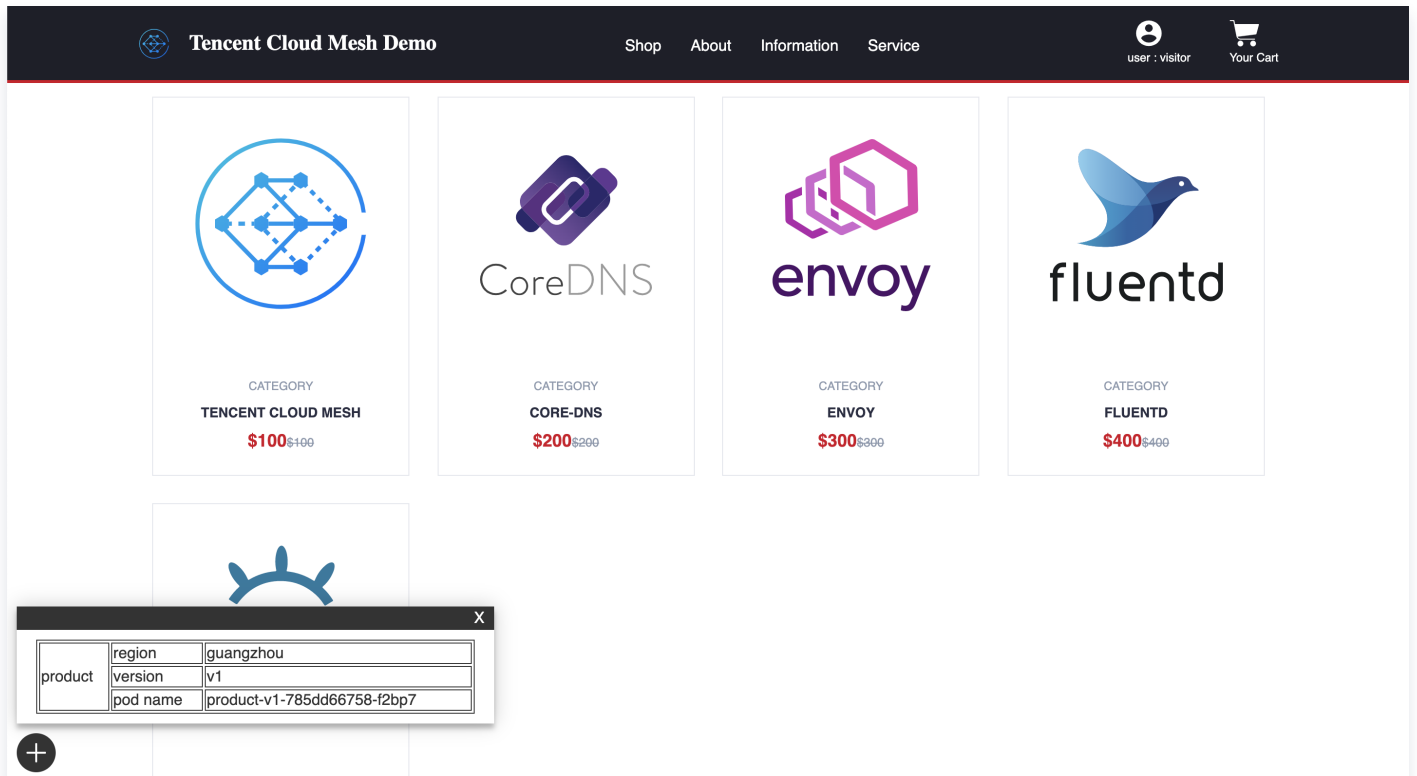
The demo application is an ecommerce website based on the Istio sample [Bookinfo](#). It consists of six services:

- **frontend**: It is the website frontend that calls the user, product, cart, and order services.
- **product**: It is the product service that provides product information and has two editions with and without an advertising banner at the top.
- **user**: It is the user login service.
- **cart**: It is the cart service that allows adding items to the cart and viewing the cart and reports stock alarms after calling the stock service. An order can be placed after login.
- **order**: It is the order checkout service that has two editions with and without freight deduction based on points. After logging in, click **CHECKOUT** to start the checkout process, which needs to call the stock service to query the stock. An order cannot be placed if the stock is insufficient.
- **stock**. It is the stock service that provides stock information for stock alarming of the cart service and stock querying of the order checkout service.

Demo application architecture



Demo application homepage



Demo Application Installation

You can go to the Tencent Cloud Mesh demo repository at [GitHub](#) to get the demo application. As Tencent Cloud Mesh's automatic sidecar injection requires labeling the Istio version, you need to select a branch on the same version as Istio or modify the master branch, specifically, the version label of the base namespace in the `mesh-demo/yamls/step01-apps-zone-a.yaml` path:

```
apiVersion: v1
kind: Namespace
metadata:
  name: base
  labels:
    istio.io/rev: 1-10-3
spec:
```

```
finalizers:
  - kubernetes
```

For example, if your Istio version is 1.8.1, you need to change `istio.io/rev: 1-10-3` to `istio.io/rev: 1-8-1`; otherwise, sidecar injection will fail.

You can use the following command to quickly deploy the demo application:

```
kubect1 apply -f yamls/step01-apps-zone-a.yaml
```

You can also go to the [TKE console](#), select **Cluster Details > Workload > Deployment**, select **Create Via YAML**, and copy the above YAML content to quickly create demo application resources.

Application Configuration

Configuring Public Network Access

Last updated: 2023-12-26 10:43:18

After the sample environment is created, all the website services are deployed in the Guangzhou cluster (the product and order services only have v1 deployed). The envoy sidecar is automatically injected to take over service traffic, `istio-ingressgateway` is created, but no listener or routing rules are configured to connect the frontend service to the public network.

1. Creating a gateway and configuring the listener rule

Create a gateway resource, configure the `istio-ingressgateway` listener rule, and set the port to 80 and protocol to HTTP. You only need to configure the gateway rule, and the Tencent Cloud Mesh backend will automatically sync the configurations of Pod, service, and associated CLB of

`istio-ingressgateway` .

Via the console

1. Log in to the [Tencent Cloud Mesh console](#) .
2. Click the target service mesh ID to enter the management page of created service meshes.
3. In **Gateway**, click **Create**.
4. In **Create Gateway**, set the gateway parameters as shown below:

← Create gateway

Gateway Name *

demo

Namespace *

default

Specify Ingress gateway *

Type *

☒ ingress

☐ egress

Access type *

☒ Public network

☐ Private network

Ingress (Egress) gateway list *

Singapore istio-ingressgateway

Selector

app: istio-ingressgateway
istio: ingressgateway

Port configuration

Protocol port *

HTTP

:

−

80

+

Please ensure that the port-level configuration for the same port of the same gateway (such as SSL termination configuration) does not conflict.

Hosts *

Enter domain names or IPs (one per line). Wildcard *** is supported;
separate multiple lines with carriage returns.

Add Port

Save

Cancel

5. Click **Save**.

Via kubectl

Submit the following YAML file to the **primary cluster** via kubectl:

```
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
```

2. Configuring the routing rule

After configuring the listener rule, configure the routing rule through the `VirtualService` resource to route the traffic from `istio-ingressgateway` to the frontend service.

Via the console

1. Log in to the [Tencent Cloud Mesh console](#).
2. Click the target service mesh ID to enter the management page of created service meshes.
3. In **Virtual Service**, click **Create**.
4. In **Create Virtual Service**, set the parameters as shown below:

The screenshot shows the 'Create Virtual Service' form in the Tencent Cloud Mesh console. The form is titled 'Create Virtual Service' and has a back arrow. It contains the following fields and options:

- Name:** frontend-vs
- Namespace:** base
- Associate hosts:** A text input field with a placeholder 'Please enter the host, and press Enter to complete'.
- Mount gateway:** base/frontend-gw
- Routing configuration:**
 - Type:** HTTP (selected), TCP, TLS
 - Condition:** uri (selected), exact (selected), [empty text box]
 - Destination:** frontend.base.svc.cluster.local, [Please select a version], Port, Weight

At the bottom of the form, there are 'Save' and 'Cancel' buttons.

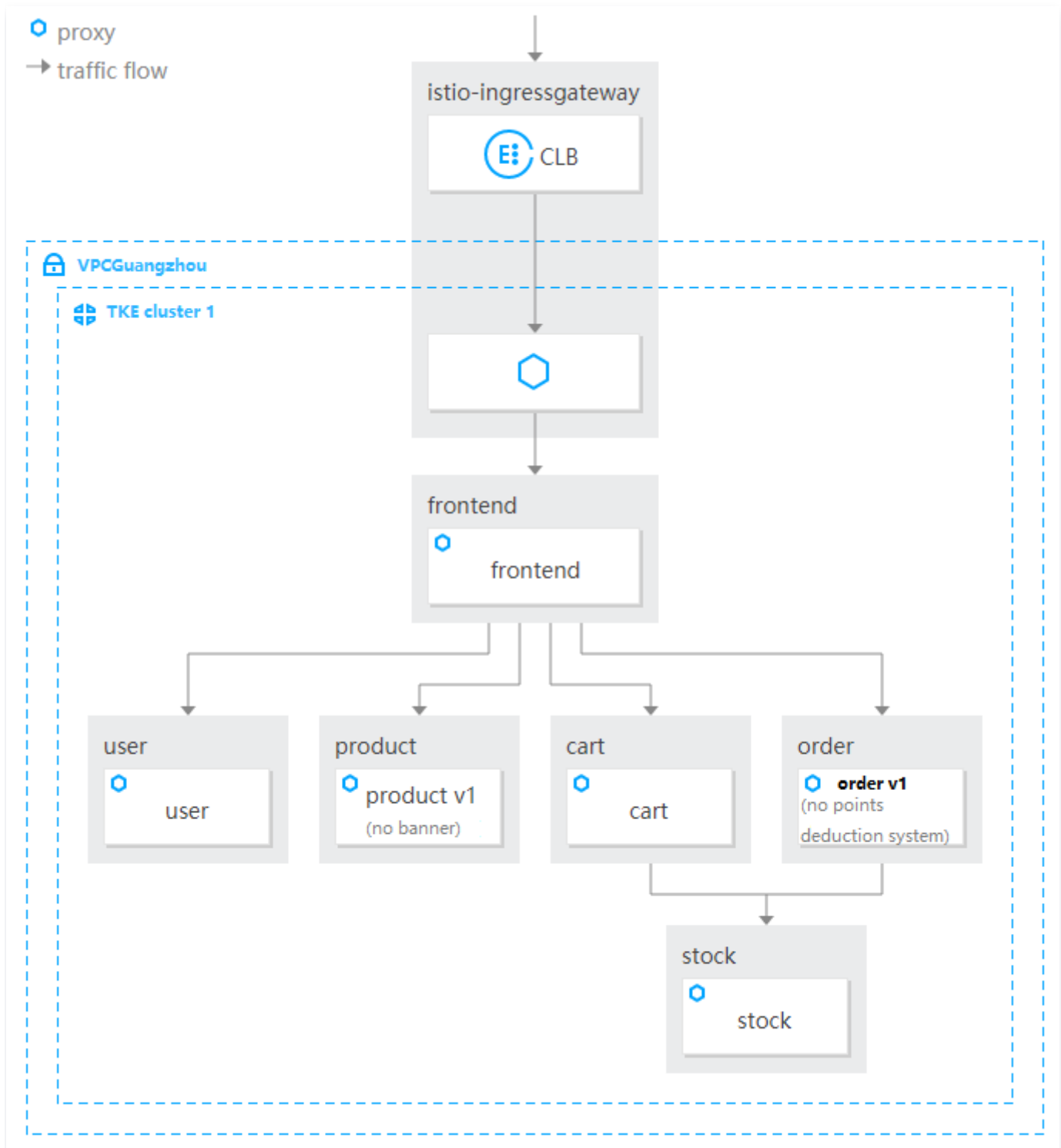
5. Click **Save**.

Via kubectl

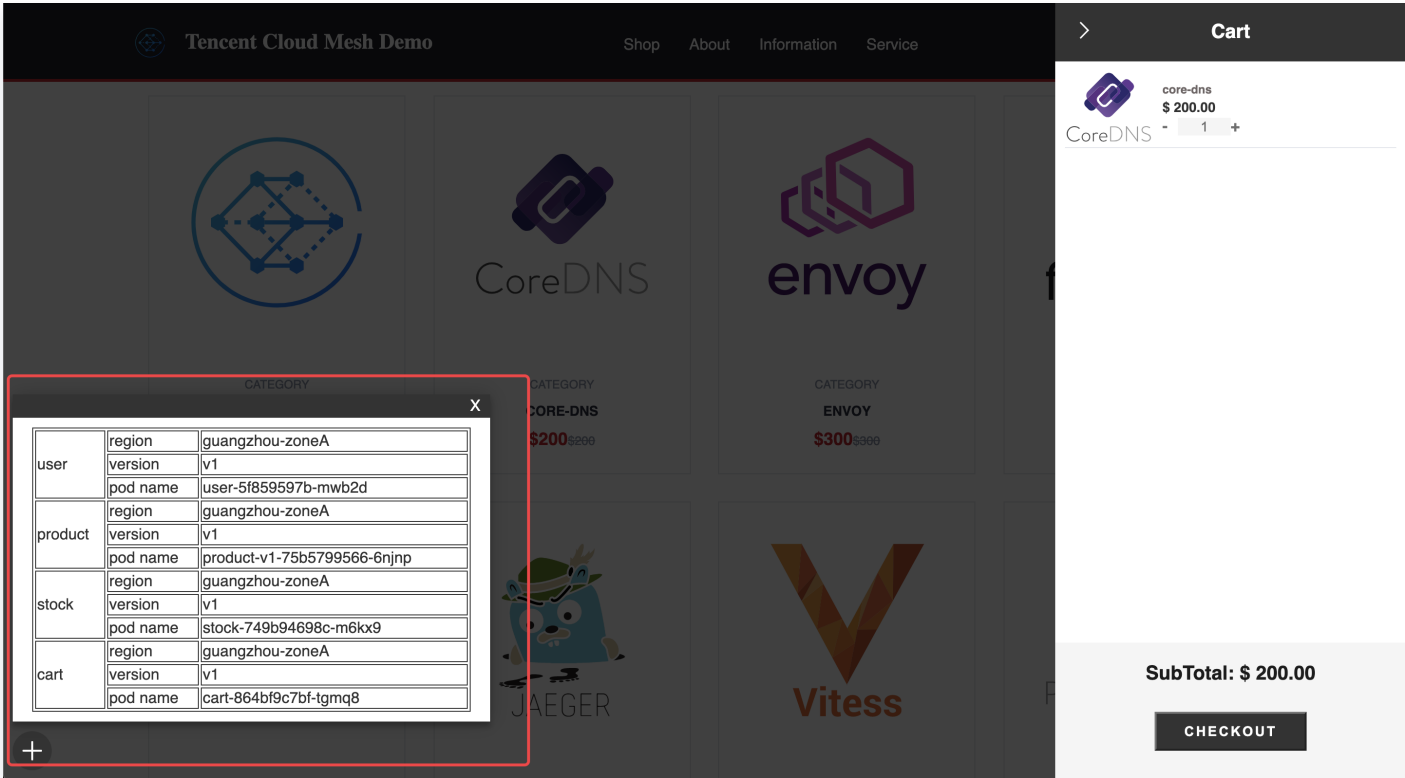
Submit the following YAML file to the **primary cluster** via kubectl:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: frontend-vs
  namespace: base
spec:
  hosts:
    - '*'
  gateways:
    - base/frontend-gw
  http:
    - route:
        - destination:
            host: frontend.base.svc.cluster.local
```

After the configuration, the demo website can be accessed through the public network IP address of `istio-ingressgateway`. The currently deployed website is as structured below:



Click the website link to log in (with accounts 1–5, including member accounts 1–3 and non-member accounts 4 and 5), add an item to the cart, and place an order to generate requests to call all the deployed services. The bottom-left floating window of the page displays the name, region, version, and Pod name of the service called by the frontend service, as shown below:

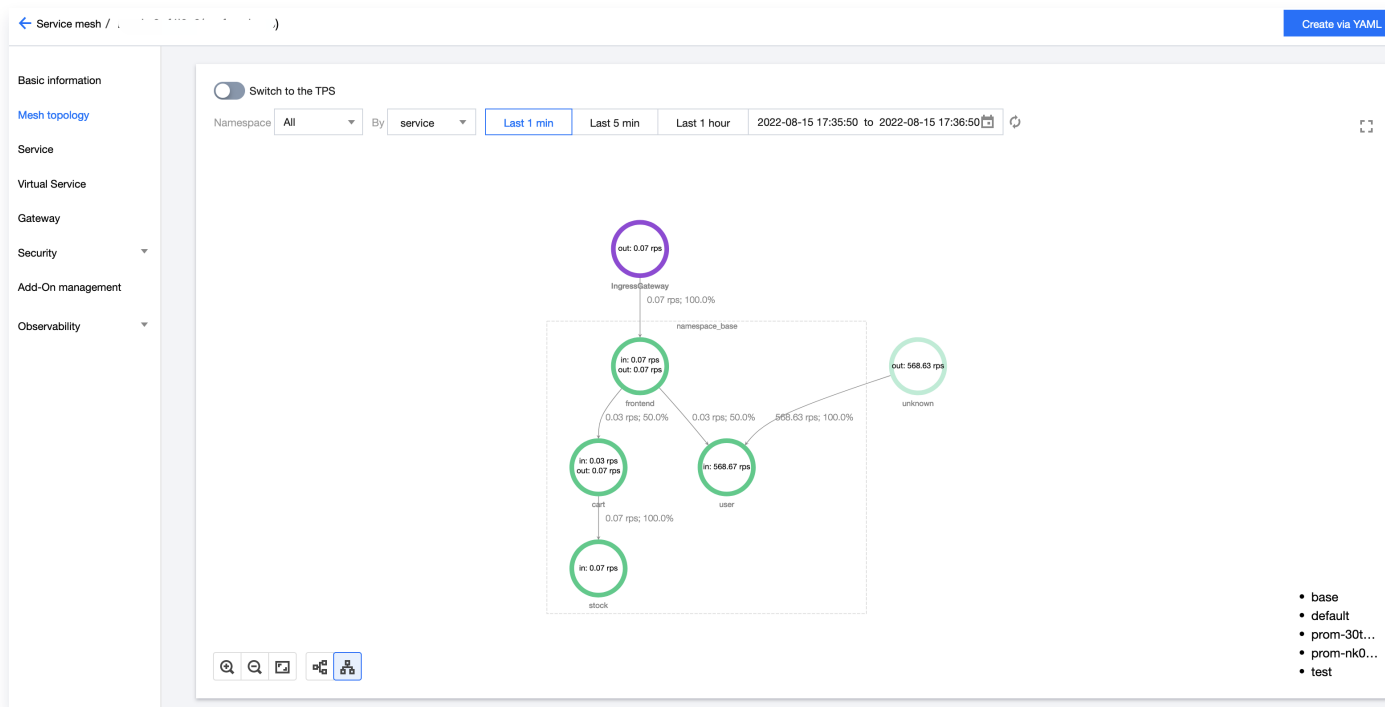


After traffic data is generated, click the **Network Topology** tab to view the network traffic topology in the mesh. Click the **Service** tab to enter the service details page, where you can view the call linkage of the request as well as the full linkage and details at each layer when the stock service is called.

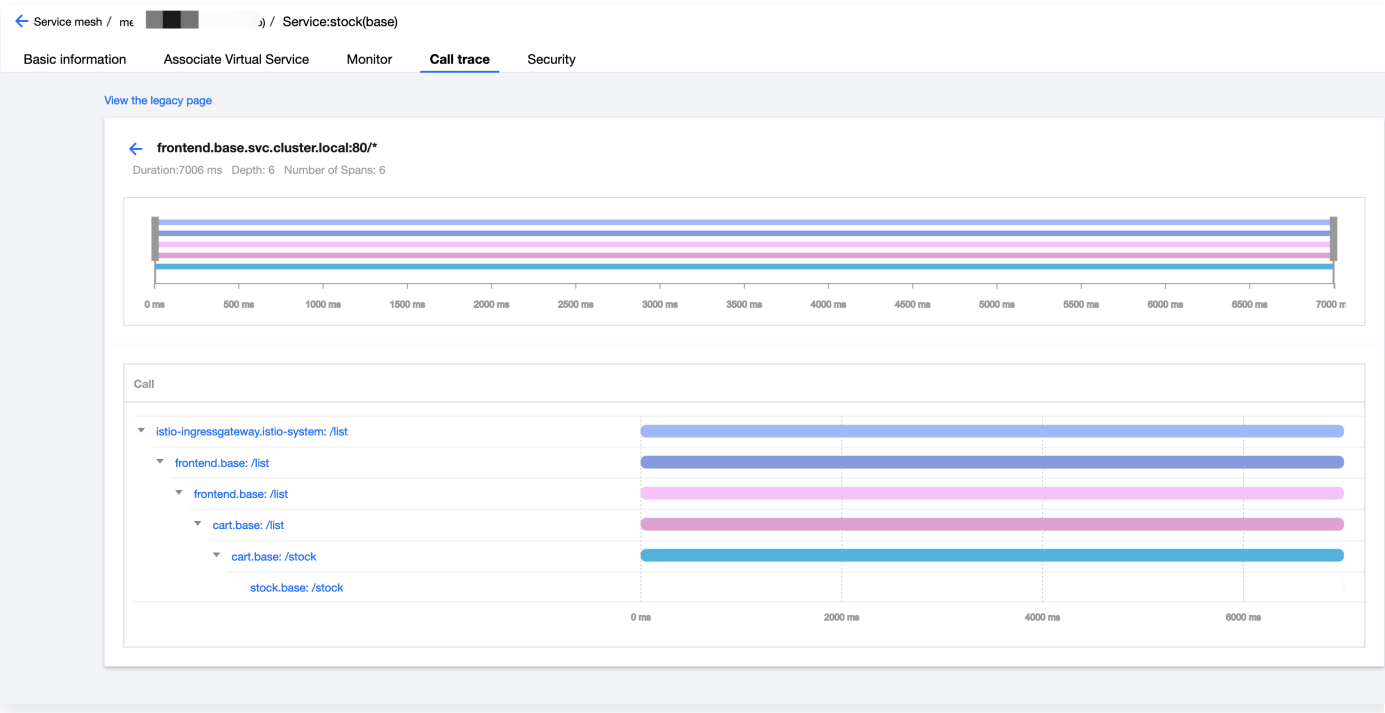
As the business accommodates more services, nearly each frontend request will have a complex call linkage. This calls for fast fault locating and analysis to determine the impact, organizing service call dependencies to determine their reasonableness, or analyzing performance parameters of the linkage, such as request duration, to optimize the call logic with serial/parallel analysis.

The full-linkage tracking system describes the traffic characteristics of the entire network to help you analyze the linkage.

- The network topology is as shown below:



- Linkage tracking is as shown below:



Traffic Control

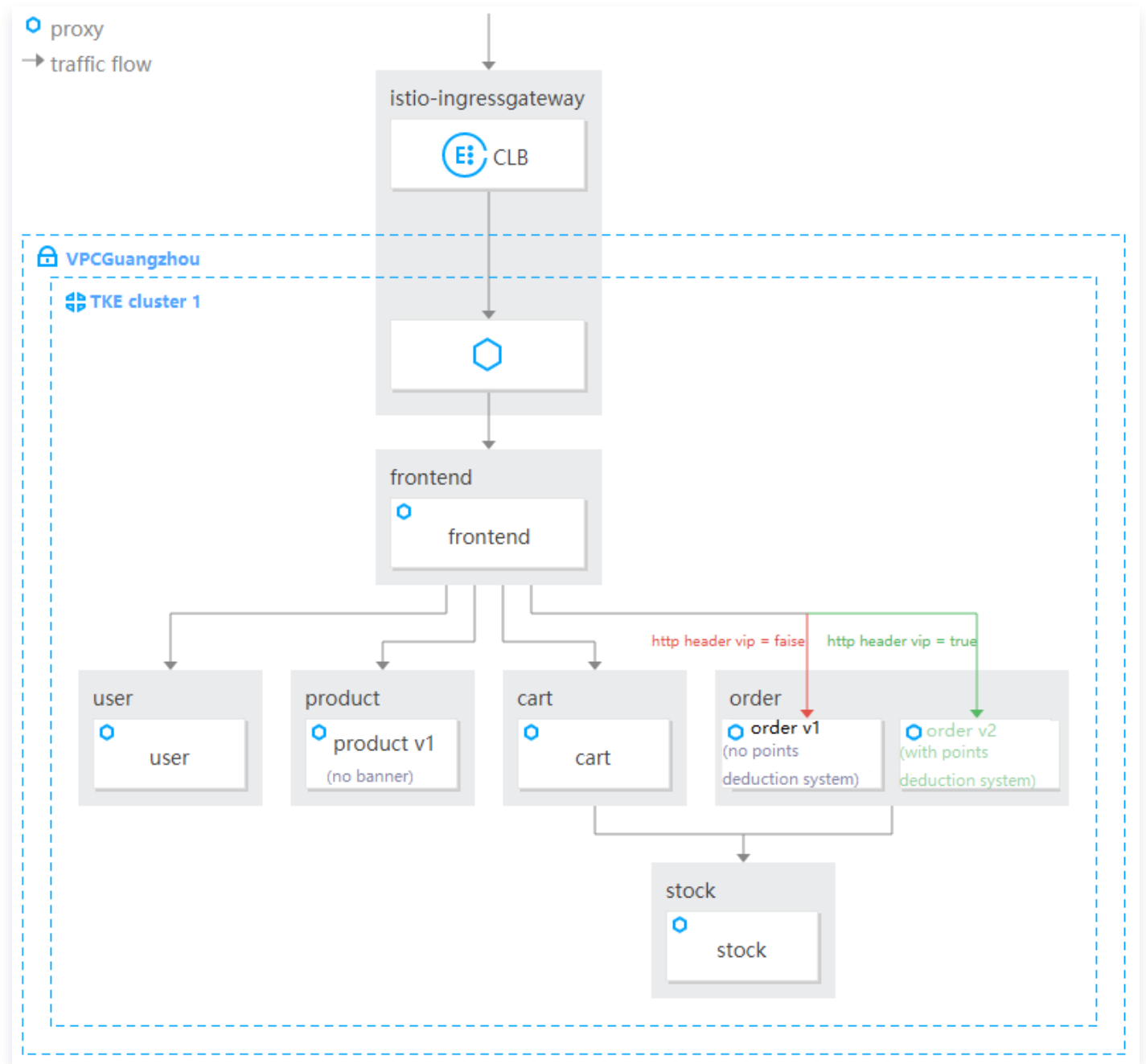
Multi-version Routing

Last updated: 2023-12-26 11:29:32

Overview

This document describes the feature of deducting the freight with member points designed to win over more members for an ecommerce website. The currently deployed order service provided by Deployment v1 does not support this feature, while the newly developed v2 of the order service does. Ideally, traffic is routed based on the cookies in the header; if a user is a member of the website as indicated in the cookies, traffic will be routed to order v2 (with freight deduction); otherwise, traffic will be routed to order v1 (without freight deduction).

An overview of multi-version service routing is as shown below:



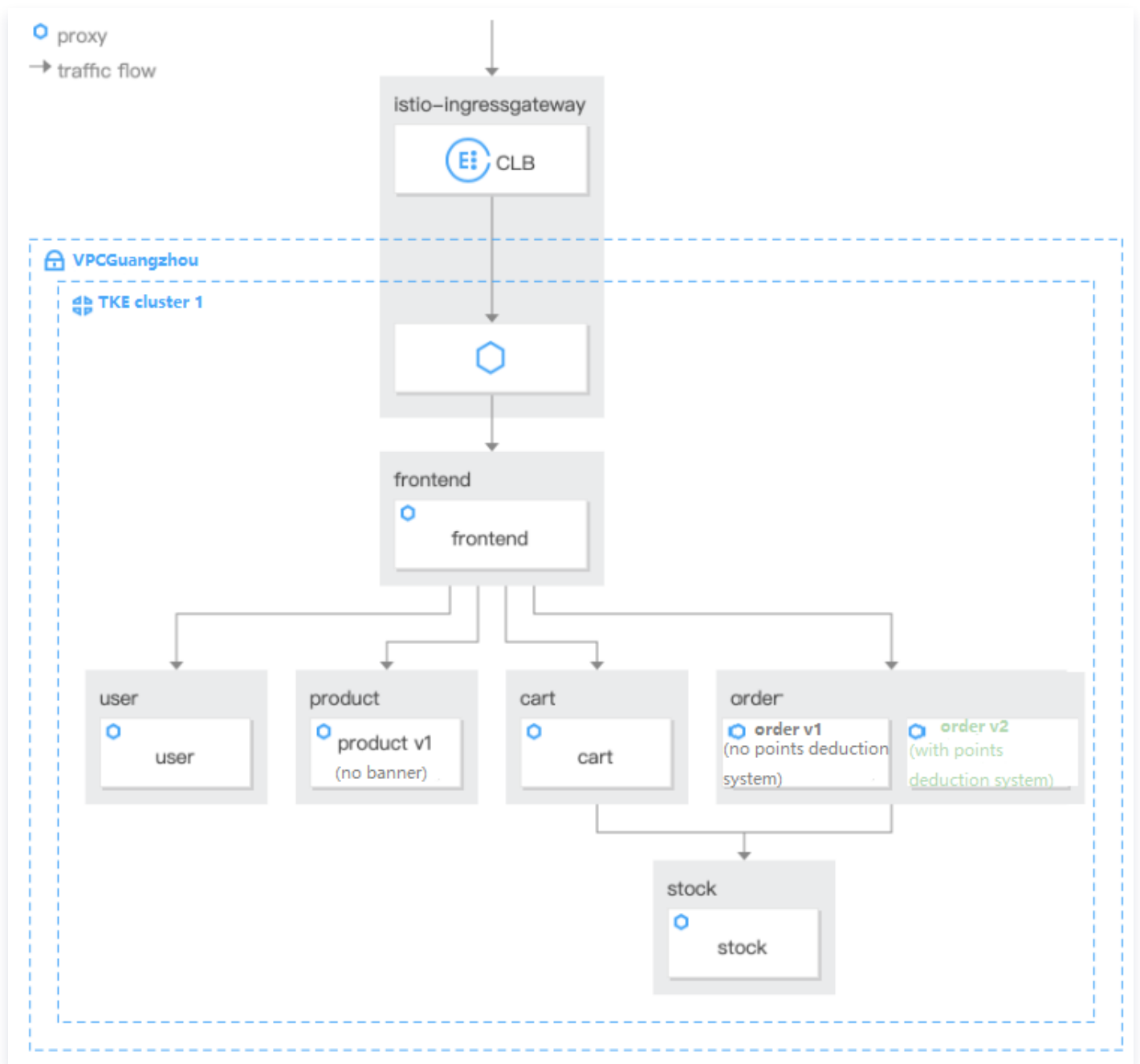
Directions

Submit the following YAML file to the primary cluster to deploy order v2 to the cluster.

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: order-v2
  namespace: base
  labels:
    app: order
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order
      version: v2
  template:
    metadata:
      labels:
        app: order
        version: v2
    spec:
      containers:
        - name: order
          image: ccr.ccs.tencentyun.com/zhulei/testorder2:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: REGION
              value: "guangzhou-zoneA"
          ports:
            - containerPort: 7000
              protocol: TCP
```

After the deployment, as no routing rule is configured, traffic to the order service will be randomly routed to v1 or v2 as shown below:



To configure a routing rule based on traffic characteristics, define the two versions of the order service through `DestinationRule` as shown below:

Define the versions of the order service as shown below:

Create Destination Rule

Service version

Add Version

Service Version 1

Collapse

Delete

Name

v1

Labels

version

:

v1

×

Add label

Labels apply a filter over the endpoints of a service in the service registry.

Corresponding workload

product-v

Service Version 2

Collapse

Delete

Name

v2

Labels

version

:

v2

×

Add label

Labels apply a filter over the endpoints of a service in the service registry.

Corresponding workload

-

Traffic policy

Add policy

Save

Cancel

©2013-2025 Tencent Cloud. All rights reserved.

Page 21 of 75

The versions of the order service are as defined below:

Destination Rule: product

Edit via YAML ☐

Service version

Create

Name	Tag	Corresponding workload	Operation
v1	version:v1	product-v1	Edit Delete
v2	version:v2	-	Edit Delete

Traffic policy

Create

Version range	Load Balancing policy	Connection pool	Locality load balancing	Health Check	Operation
No traffic policy					

Or you can submit the following YAML file to the primary cluster to create a destination rule:

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: order
  namespace: base
spec:
  host: order
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
  exportTo:
    - '*'
```

After the two versions are defined, routing will be performed based on traffic characteristics through the `VirtualService` definition. If the `header-cookie` of a request contains `vip=false`, traffic will be routed to v1 subset of the order service; otherwise, traffic will be routed to v2 subset. That is, member

requests and non-member requests will be routed to order v2 and order v1, respectively. This configuration can be performed by submitting the following YAML file to the primary cluster.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: order-vs
  namespace: base
spec:
  hosts:
    - order.base.svc.cluster.local
  http:
    - match:
        - headers:
            cookie:
              exact: vip=false
      route:
        - destination:
            host: order.base.svc.cluster.local
            subset: v1
    - match:
        - headers:
            cookie:
              exact: vip=true
      route:
        - destination:
            host: order.base.svc.cluster.local
            subset: v2
```

After the configuration, if you log in to a member account (ID: 1–3), add an item to the cart, and make the payment, you will find that the freight is deducted, and the traffic is routed to order v2; if you log in to a non-member account (ID: 4–5), add an item to the cart, and make the payment, you will find that the freight is not deducted, and according to the `VIP` field in the header, the traffic is routed to the originally deployed order v1. Version information can be viewed in the floating window in the bottom-left corner. The request from a member is routed to v2 as shown below:

Tencent Cloud Mesh Demo

ShopAboutInformationService

user : James

Your Cart

Your Order

Products	Total
core-dns	\$200.00

Subtotal

\$ 200.00

Shipping

\$10.00

Total

\$200.00

user	region	guangzhou-zoneA
	version	v1
	pod name	user-5f859597b-mwb2d
stock	region	guangzhou-zoneA
	version	v1
	pod name	stock-749b94698c-m6kx9
order	region	guangzhou-zoneA
	version	v2
	pod name	order-v2-675bb7bd8c-9gtn4

+

connect

INFORMATION

SERVICE

Copyright © 2020 Tencent TCM Team

MyAccount

The request from a non-member user is routed to v1 as shown below:

Tencent Cloud Mesh Demo

ShopAboutInformationService

user : John

Your Cart

Your Order

Products	Total
Tencent Cloud Mesh	\$100.00

Subtotal

\$ 100.00

Shipping

\$10.00

Total

\$110.00

user	region	guangzhou-zoneA
	version	v1
	pod name	user-5f859597b-mwb2d
stock	region	guangzhou-zoneA
	version	v1
	pod name	stock-749b94698c-m6kx9
order	region	guangzhou-zoneA
	version	v1
	pod name	order-v1-797bd5df47-hh28q

+

connect

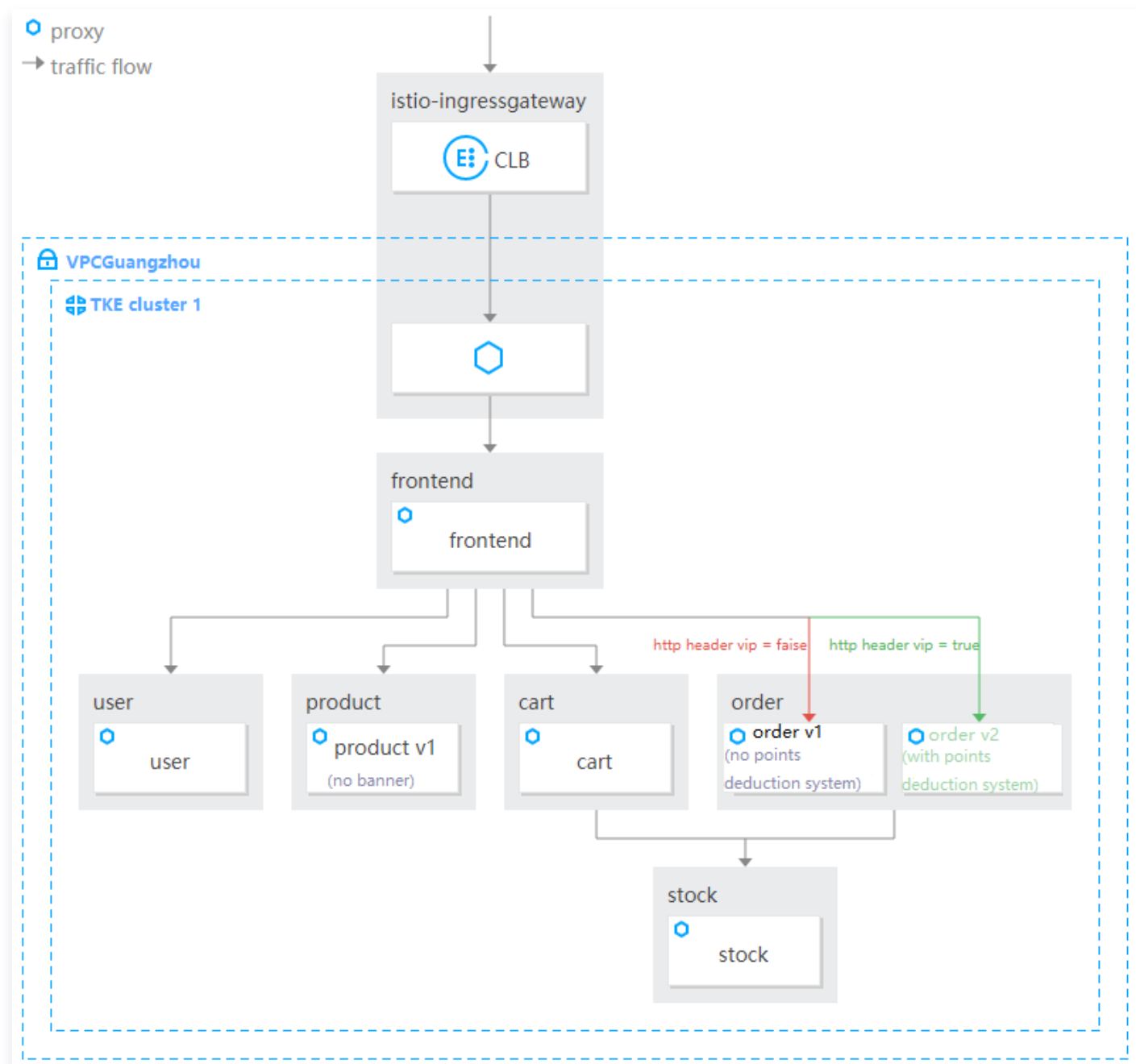
INFORMATION

SERVICE

Copyright © 2020 Tencent TCM Team

MyAccount

Routing based on traffic rules is as shown below:



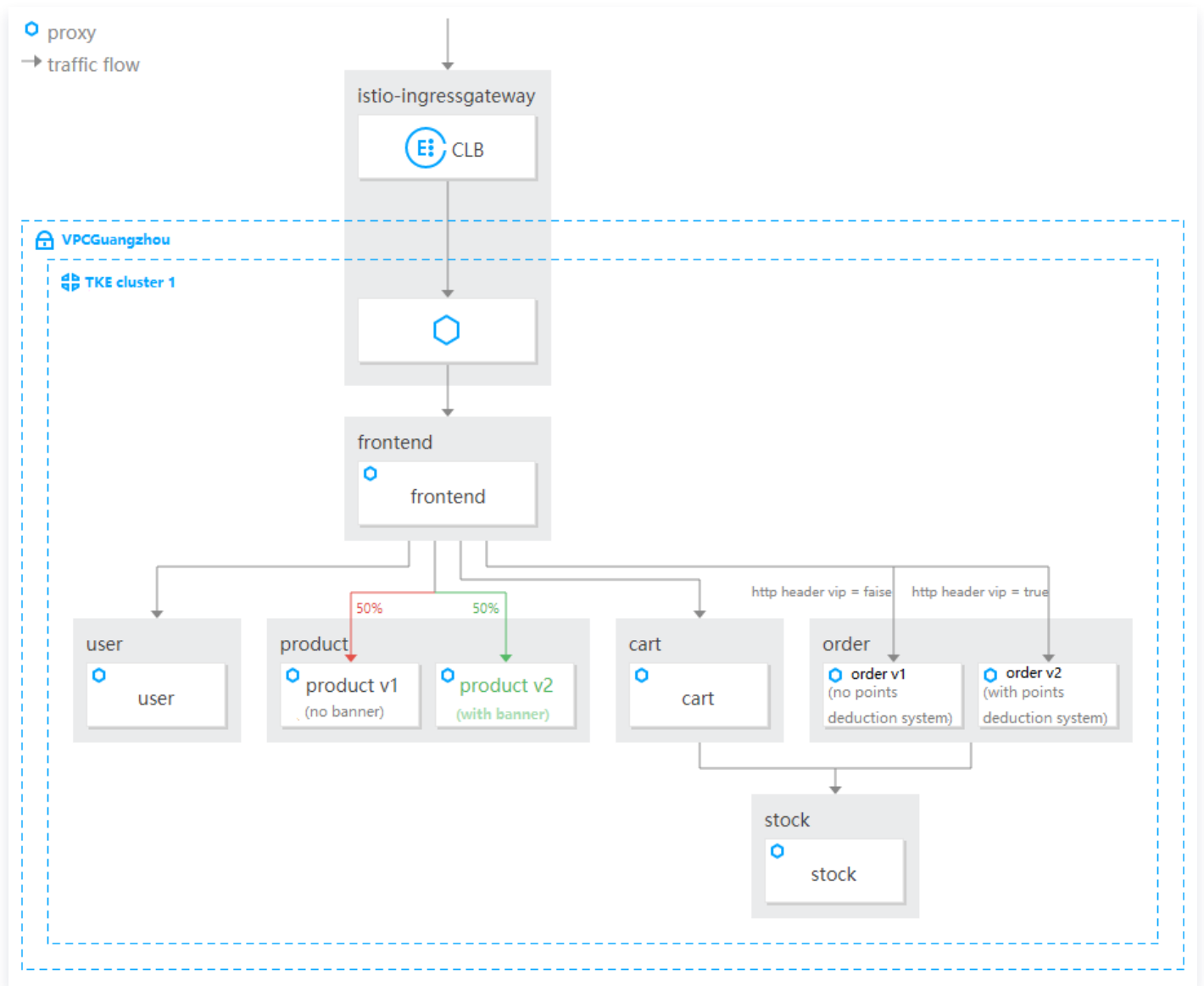
Canary Release

Last updated: 2023-12-26 10:44:22

Overview

Website traffic increase brings along the need to place ads on the product page. Therefore, website developers developed the product v2 as a Deployment and wish to perform a canary release.

An overview of the canary release is as shown below:



Directions

Deploy the product v2 Deployment to the primary cluster first:

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: product-v2
  namespace: base
  labels:
    app: product
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v2
  template:
    metadata:
      labels:
        app: product
        version: v2
    spec:
      containers:
        - name: product
          image: ccr.ccs.tencentyun.com/zhulei/testproduct2:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: REGION
              value: "guangzhou-zoneA"
          ports:
            - containerPort: 7000
```

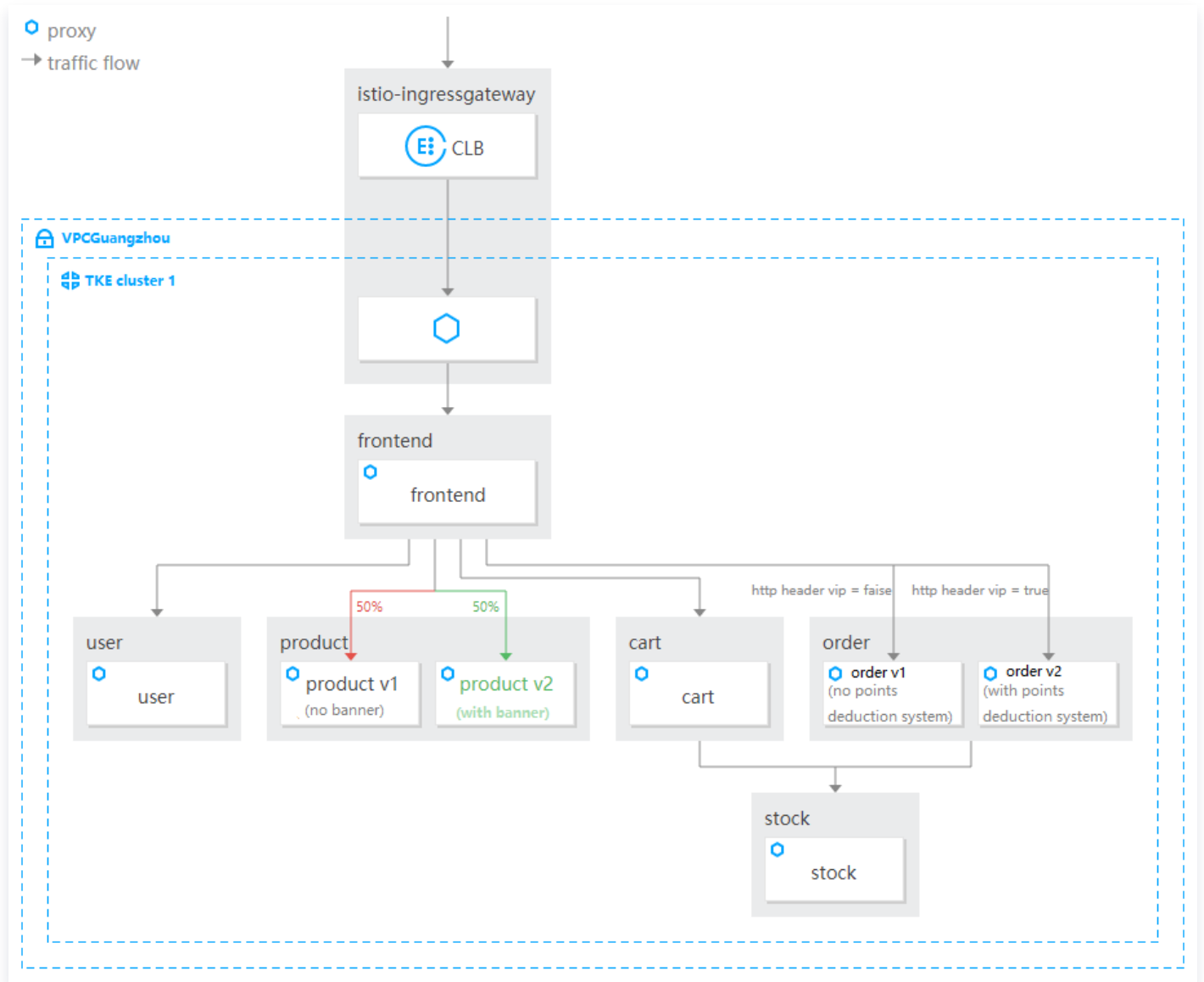
In the first step, define the service version through disaster recovery and the weighted routing through `VirtualService` . Route 50% of the traffic to the product v2 subset for verification and the other 50% to product v1. This can be configured by submitting the following YAML file to the primary cluster.

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

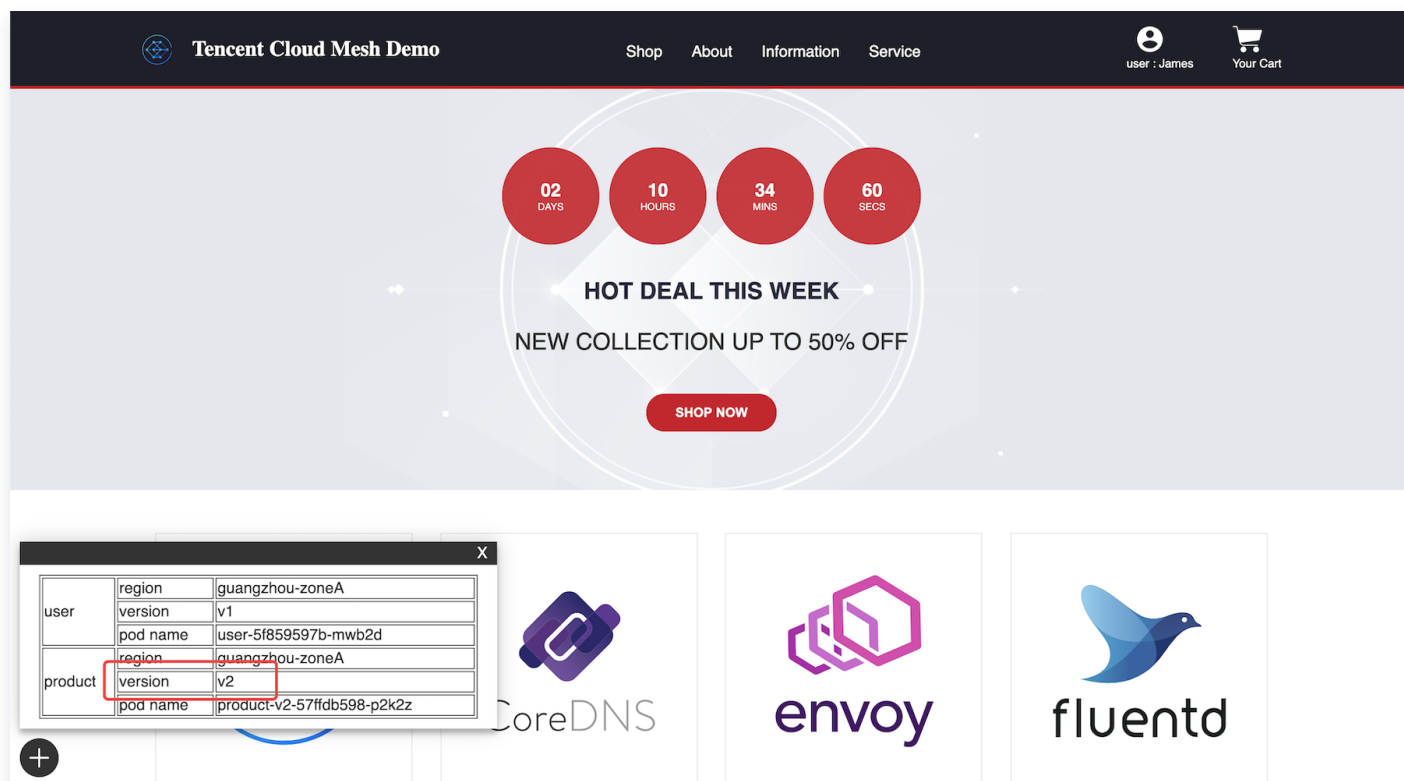
```
name: product-vs
namespace: base
spec:
  hosts:
    - "product.base.svc.cluster.local"
  http:
    - match:
        - uri:
            exact: /product
      route:
        - destination:
            host: product.base.svc.cluster.local
            subset: v1
            port:
              number: 7000
            weight: 50
        - destination:
            host: product.base.svc.cluster.local
            subset: v2
            port:
              number: 7000
            weight: 50
    ---
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
```

After the configuration, 50% of the traffic to the product service will be routed to product v1 and the other 50% to product v2. Refresh the product page for verification.

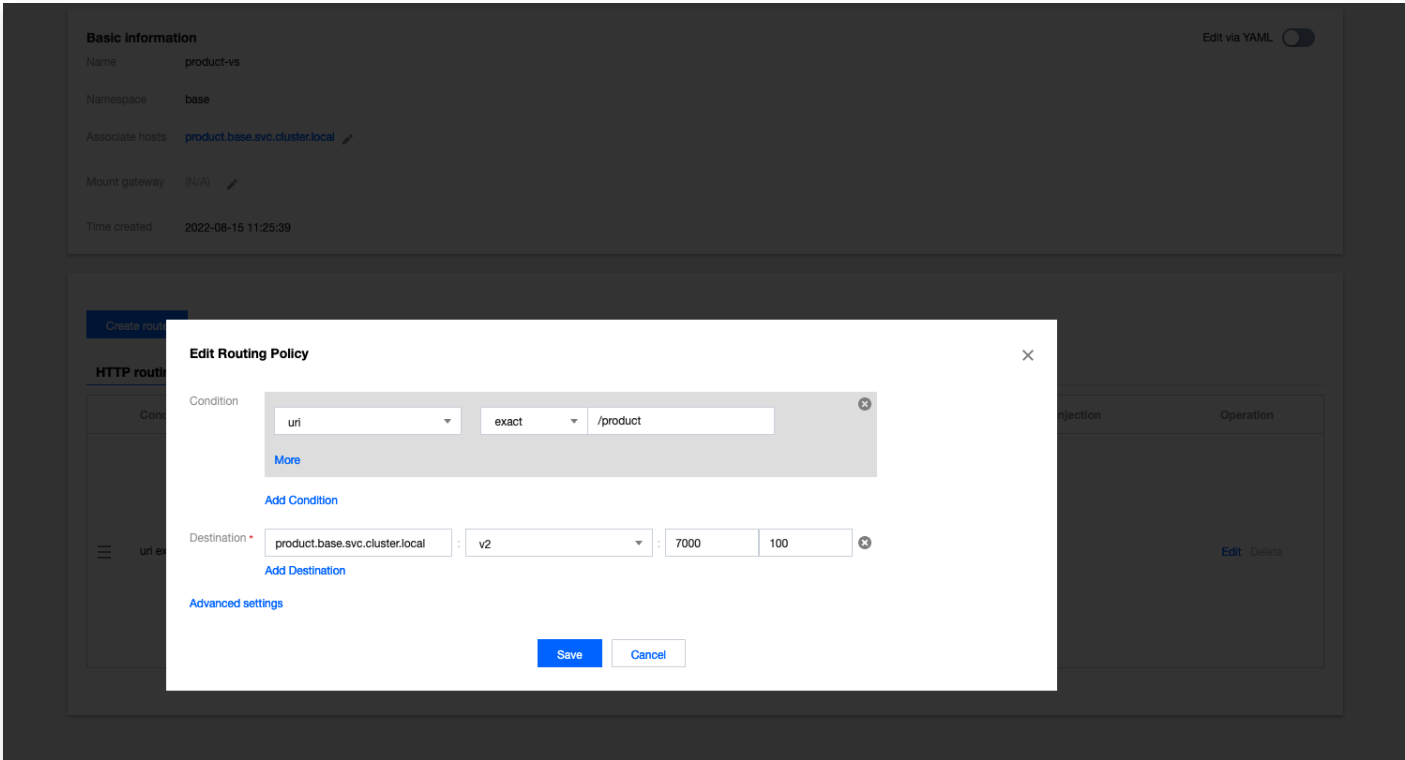
The weighted routing is as shown below:



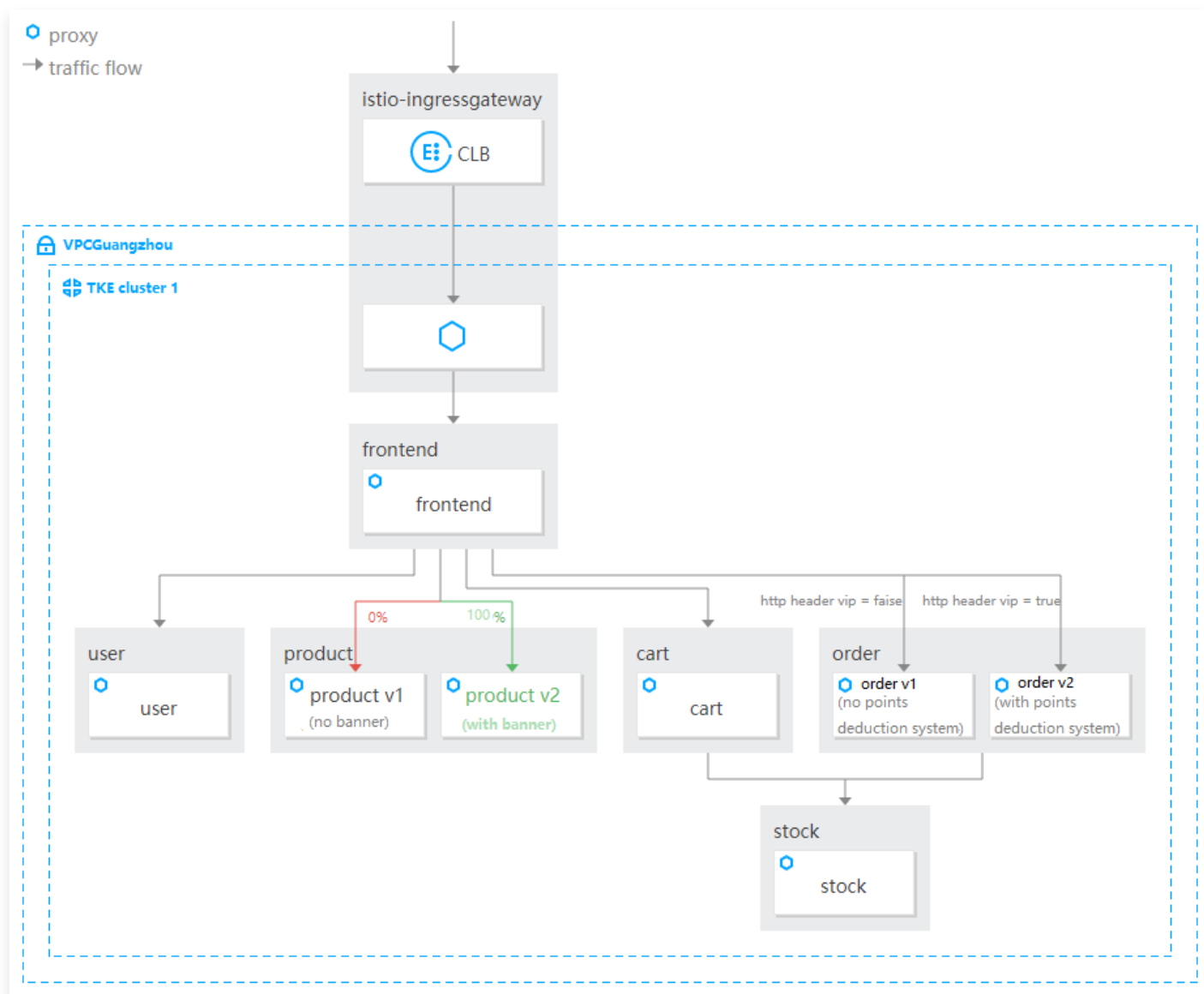
50% of the request traffic is routed to product v2 as shown below:



After product v2 passes the verification, modify the destination weight of the routing rule of the associated `VirtualService`. Set to route all the traffic (100%) to the product service to product v2 and refresh the product list page for verification. The weight is adjusted based on `VirtualService` as shown below:



Canary release is completed as shown below:



Service HA

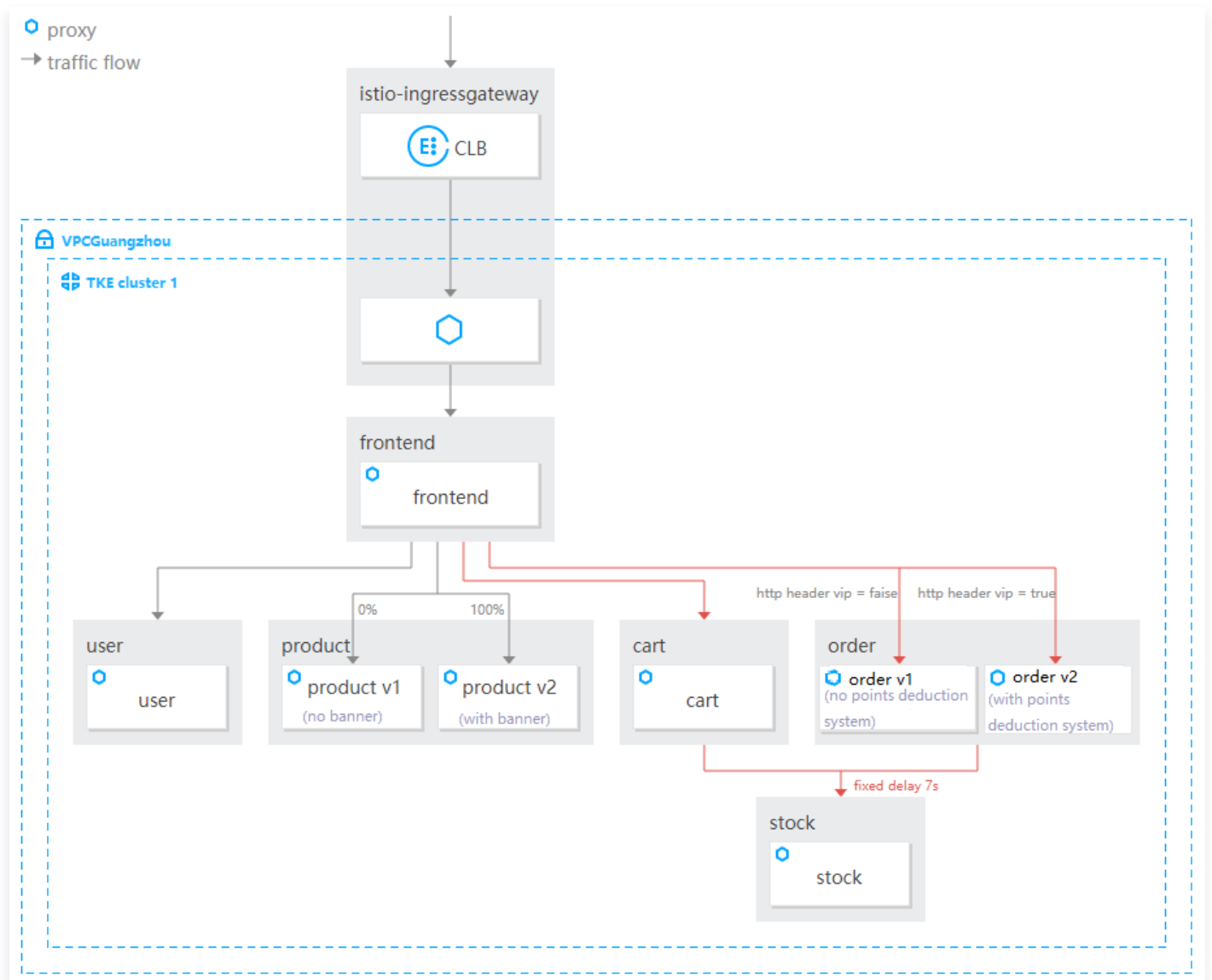
Fault Injection Testing

Last updated: 2023-12-26 11:31:04

Overview

This document simulates website system behaviors upon latency faults when the stock service is accessed, aiming to test the service elasticity and optimize the access experience of website users.

The stock service has a fixed latency of seven seconds as shown below:



Directions

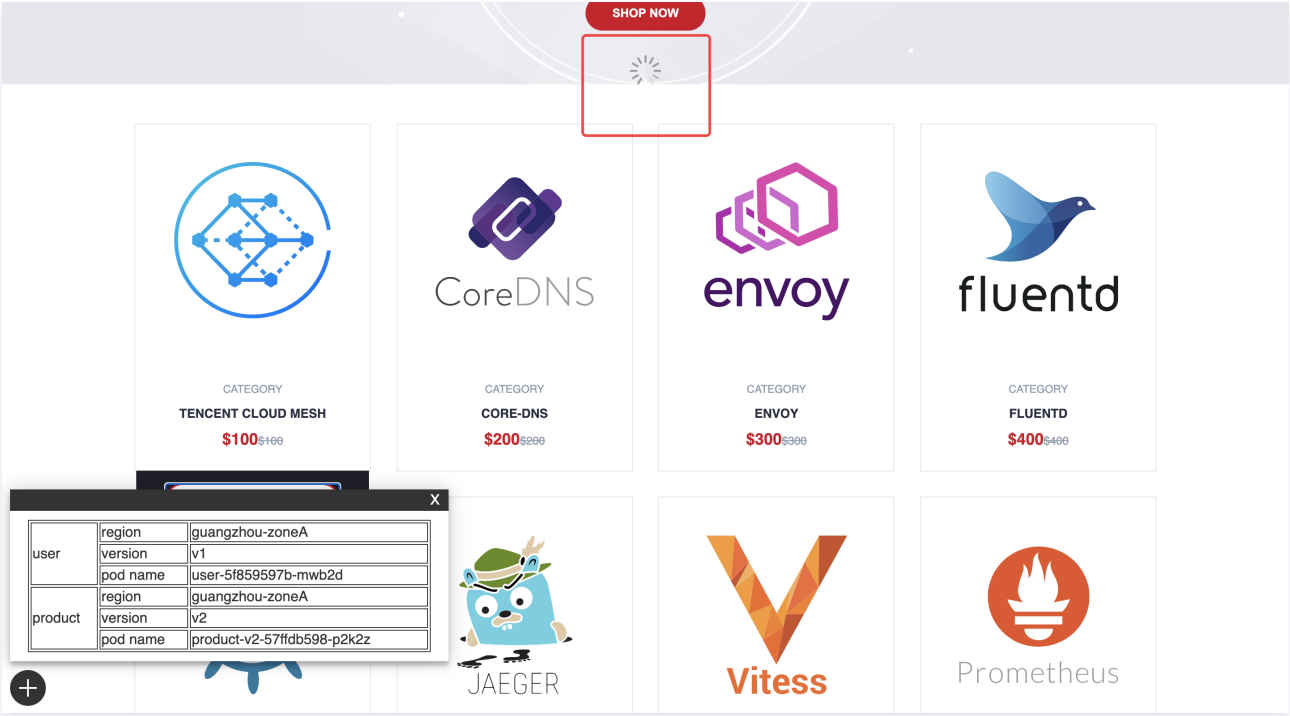
Configure the `VirtualService` bound to the stock service and set the fault injection policy for accessing the stock service: 100% of the requests will have a fixed latency of seven seconds.

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

```
kind: VirtualService
metadata:
  name: stock-vs
  namespace: base
spec:
  hosts:
    - stock.base.svc.cluster.local
  http:
    - route:
        - destination:
            host: stock.base.svc.cluster.local
  fault:
    delay:
      fixedDelay: 7000ms
      percentage:
        value: 100
```

After the configuration, click **ADD TO CART** on the demo website page to add an item to the cart or click **YOUR CART** to call the cart service. The cart service will then call the stock service to query the stock, and the stock service will experience a fixed latency of seven seconds. Click **CHECKOUT** on the cart page to start the checkout process to call the order service. The order service will then call the stock service to query the stock, and the stock service will experience a fixed latency of seven seconds. The latency means that the page will be in the loading status for seven seconds until the fault disappears, which adversely affects users' browsing experience.

The waiting status after the cart service calls the stock service is as shown below:



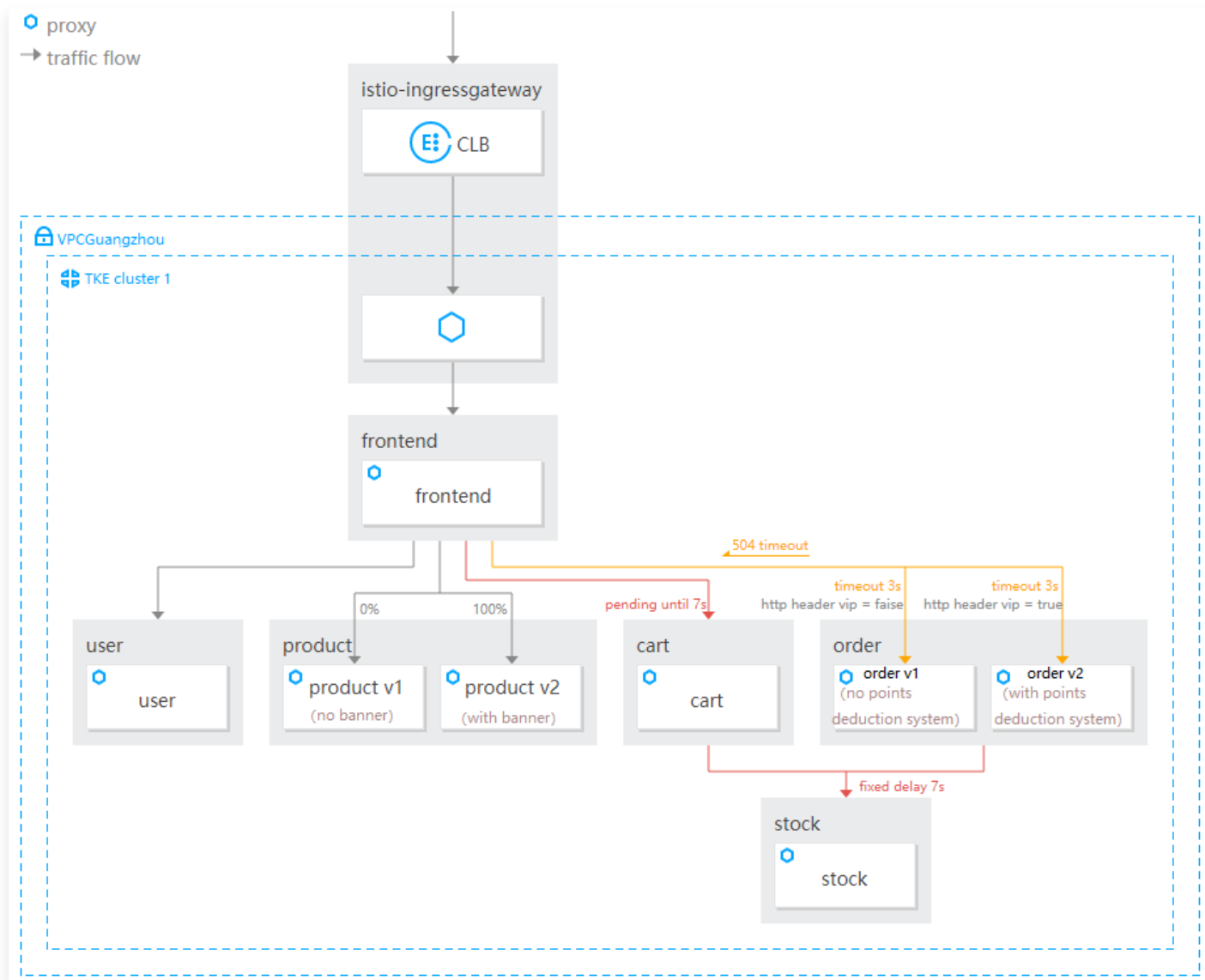
A timeout period can be configured to optimize the browsing experience of website users.

Service Timeout Configuration

Last updated: 2023-12-26 10:45:52

Overview

The order service has a timeout period of three seconds as shown below:



After fault injection is configured for the stock service, it is found that user requests to the website will be in the waiting status due to the fault. To optimize the browsing experience, a timeout period needs to be configured.

Directions

Configure a timeout period of three seconds for the order service by applying the following

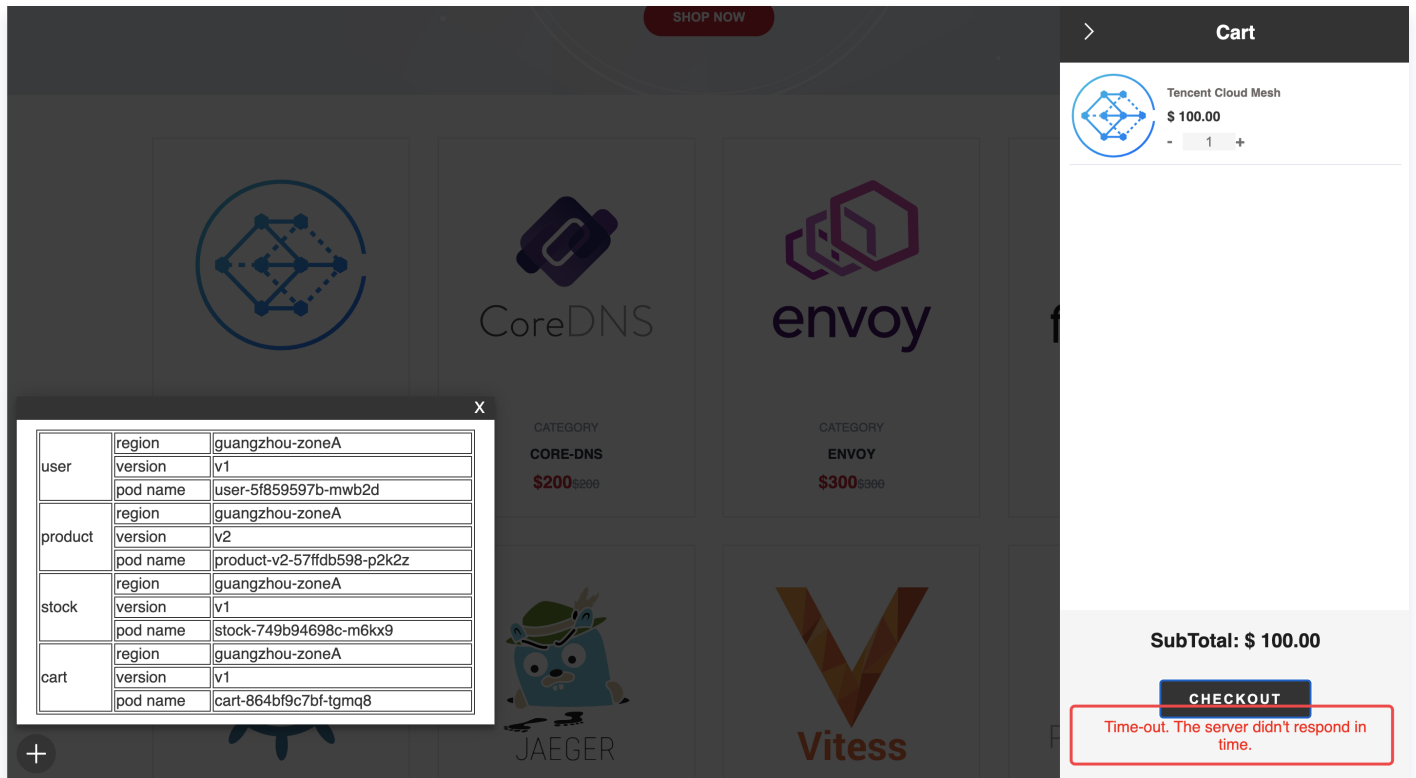
`VirtualService` and no timeout period for the cart service.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: order-vs
  namespace: base
spec:
  hosts:
    - order.base.svc.cluster.local
  http:
    - match:
        - headers:
            cookie:
              exact: vip=false
      route:
        - destination:
            host: order.base.svc.cluster.local
            subset: v1
        timeout: 3000ms
    - match:
        - headers:
            cookie:
              exact: vip=true
      route:
        - destination:
            host: order.base.svc.cluster.local
            subset: v2
        timeout: 3000ms
```

After the configuration, add an item to the cart, and the cart service will experience an access latency of seven seconds due to the injected fault, without a timeout. Then, click **CHECKOUT** to start the checkout process and call the order service, and, in addition to the access latency of seven seconds, the order service

will experience a timeout period of three seconds, which means a timeout will occur if no operation is performed within three seconds after the order service is called.

The timeout occurring when the cart service calls the order service is as shown below:



After completing the timeout configuration and the fault injection test on the service, delete the `VirtualService` resource associated with the stock service to disassociate the configured fault injection policy from the stock service.

Delete the `VirtualService` associated with the stock service as shown below:

Create

Namespace All

Search by name

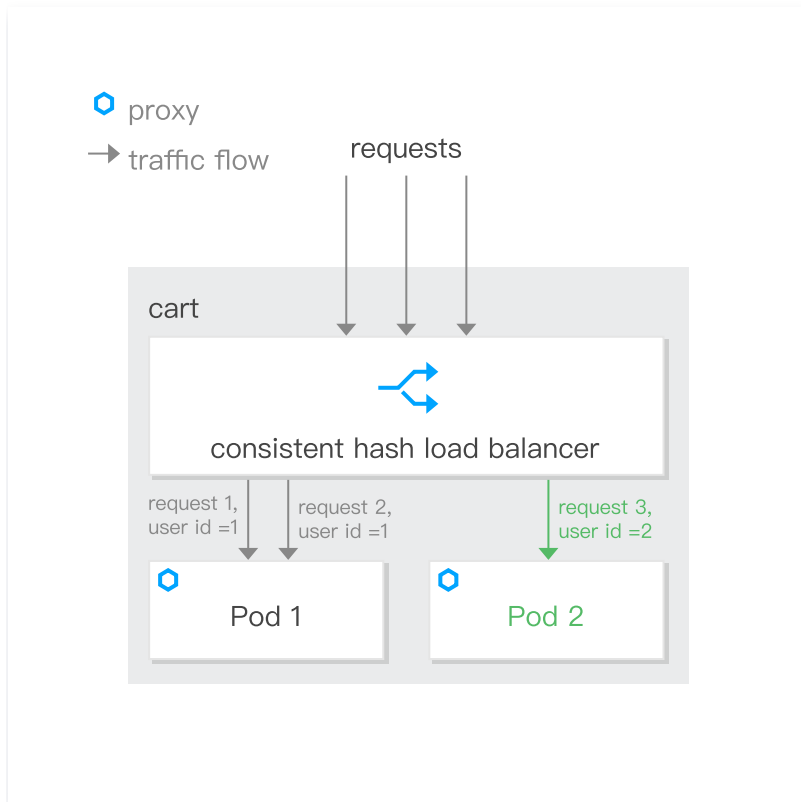
Name	Hosts	Gateway	HTTP routing	TCP routing	TLS routing	Operation
frontend-vs	*	base/frontend-gw	1	0	0	Edit Delete
order-vs	order.base.svc.cluster.local	-	2	0	0	Edit Delete
product-vs	product.base.svc.cluster.local	-	1	0	0	Edit Delete
stock-vs	stock.base.svc.cluster.local	-	1	0	0	Edit Delete

Session Persistence

Last updated: 2023-12-26 10:46:12

Overview

The cart service has multiple running Pods, requiring the session persistence feature to ensure that requests from the same user are routed to the same Pod and the user's cart information will not be lost. Session persistence is as shown below:



Directions

Implement the session persistence feature by setting the `DestinationRule` loading balancing policy of the cart service, with `UserID` in the request header for consistent hashing.

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

```
loadBalancer:
  consistentHash:
    httpHeaderName: UserID
exportTo:
  - '*'
```

After the configuration, click **YOUR CART** multiple times or **ADD TO CART** after login to call the cart service to verify whether requests from the same user are routed to the same Pod. The name of the Pod providing the cart service can be viewed in the bottom-left floating window, which is the same for requests from the same user. Requests from the same user are load balanced to the same Pod as shown below:

Tencent Cloud Mesh Demo

ShopAboutInformationService

02 DAYS

10 HOURS

34 MINS

60 SECS

HOT DEAL THIS WEEK

NEW COLLECTION UP TO 50% OFF

SHOP NOW

user	region	guangzhou-zoneA
	version	v1
	pod name	user-5f859597b-mwb2d
product	region	guangzhou-zoneA
	version	v2
	pod name	product-v2-57fdb598-p2k2z
stock	region	guangzhou-zoneA
	version	v1
	pod name	stock-749b94698c-m6kx9
cart	region	guangzhou-zoneA
	version	v1
	pod name	cart-864bf9c7bf-rjbzf

CoreDNS

envoy

Cart

core-dns \$ 200.00

CoreDNS 1

SubTotal: \$ 200.00

CHECKOUT

Connection Pool–based Concurrency Limiting

Last updated: 2023-12-26 11:35:08

Overview

This document describes how to limit the maximum number of concurrent access requests to an ecommerce website to ensure robust service running.

Directions

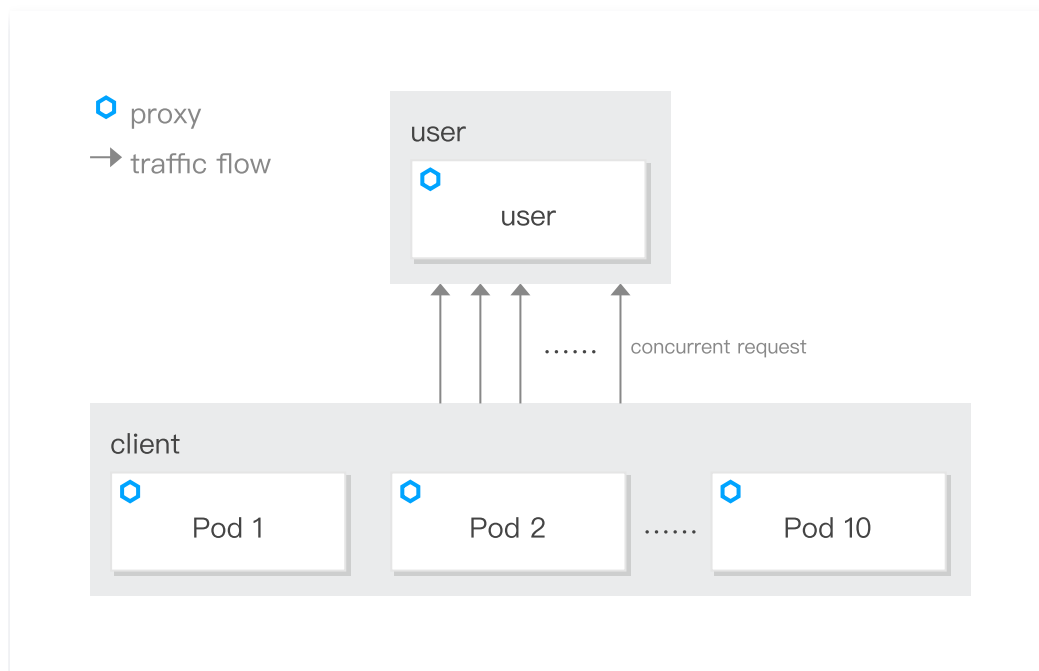
To simulate a high number of concurrent requests to the user service, submit the following YAML file to deploy the client service (with ten Pods).

```
apiVersion: v1
kind: Namespace
metadata:
  name: test
  labels:
    istio-injection: enabled
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:
```

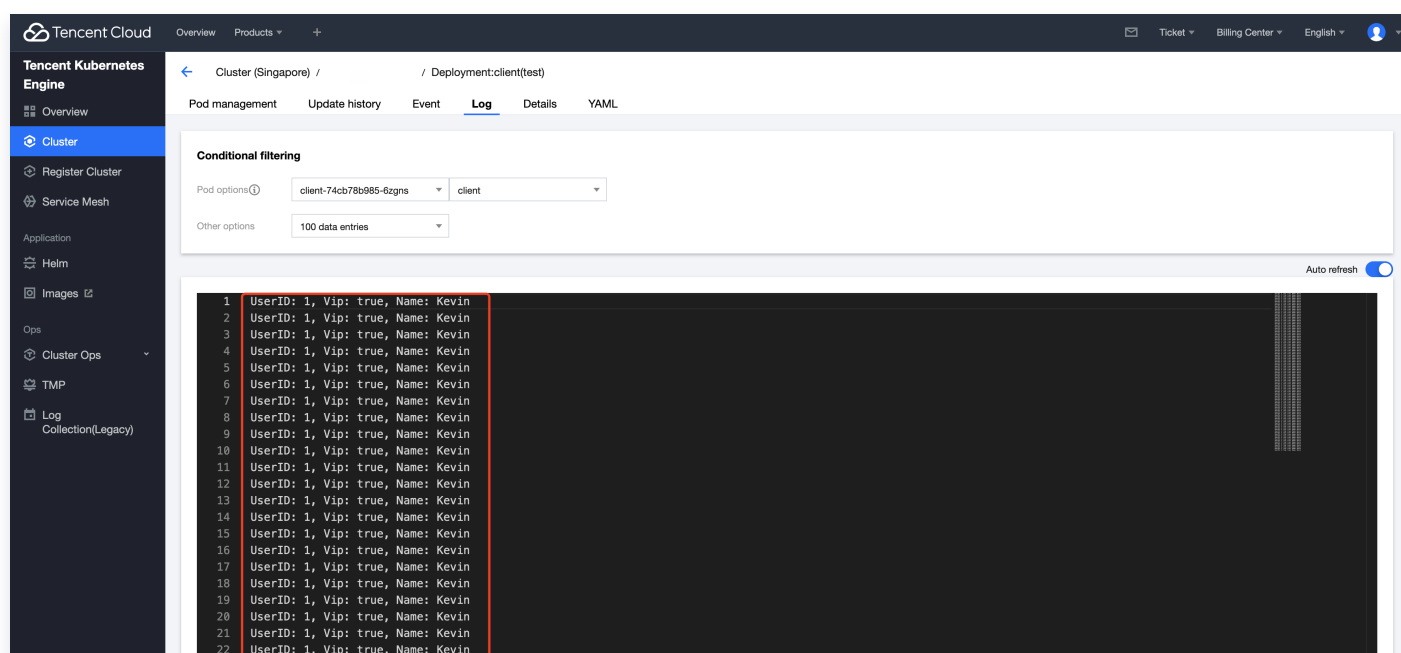
```
    app: client
  spec:
    containers:
      - name: client
        image: ccr.ccs.tencentyun.com/zhulei/testclient:v1
        imagePullPolicy: Always
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: 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
```

At this point, all the requests to the user service can pass, as there is no limit on the maximum concurrency. View the client Pod log through the client Deployment in the TKE console, which shows that all the requests returned the username `Kevin`, indicating that requests succeeded.

A high number of concurrent requests is as shown below:



All the requests succeeded as shown below:



Configure the `Destination Rule` of the user service to limit the maximum concurrency to `1` :

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: user
  namespace: base
spec:
  host: user
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 1
        http2MaxRequests: 1
        maxRequestsPerConnection: 1
  exportTo:
    - '*'
```

View the client Pod log, which shows that some requests are abnormal and failed, with no username returned. In this case, the connection pool succeeded in limiting the maximum number of concurrent requests to the service.

Some access requests failed as shown below:

The screenshot shows the Tencent Cloud Kubernetes Engine console. The left sidebar contains navigation options: Overview, Cluster, Register Cluster, Service Mesh, Application, Helm, Images, Ops, Cluster Ops, TMP, and Log Collection(Legacy). The main panel is titled 'Cluster (Singapore)' and 'Deployment: client(test)'. The 'Log' tab is selected, displaying a list of log entries. The 'Conditional filtering' section shows 'Pod options' set to 'client-5f968b69b4-29448' and 'client', and 'Other options' set to '100 data entries'. The log entries are as follows:

Line	Log Entry
1	UserID: 0, Vip: false, Name:
2	UserID: 0, Vip: false, Name:
3	UserID: 0, Vip: false, Name:
4	UserID: 0, Vip: false, Name:
5	UserID: 0, Vip: false, Name:
6	UserID: 0, Vip: false, Name:
7	UserID: 0, Vip: false, Name:
8	UserID: 0, Vip: false, Name:
9	UserID: 0, Vip: false, Name:
10	UserID: 0, Vip: false, Name:
11	UserID: 0, Vip: false, Name:
12	UserID: 1, Vip: true, Name: Kevin
13	UserID: 0, Vip: false, Name:
14	UserID: 0, Vip: false, Name:
15	UserID: 1, Vip: true, Name: Kevin
16	UserID: 0, Vip: false, Name:
17	UserID: 0, Vip: false, Name:
18	UserID: 0, Vip: false, Name:
19	UserID: 0, Vip: false, Name:
20	UserID: 0, Vip: false, Name:
21	UserID: 0, Vip: false, Name:
22	UserID: 0, Vip: false, Name:
23	UserID: 0, Vip: false, Name:

After the connection pool testing, delete the traffic policy configuration of the connection pool on the details page of the user service.

Multi-cluster Disaster Recovery and Multi-active

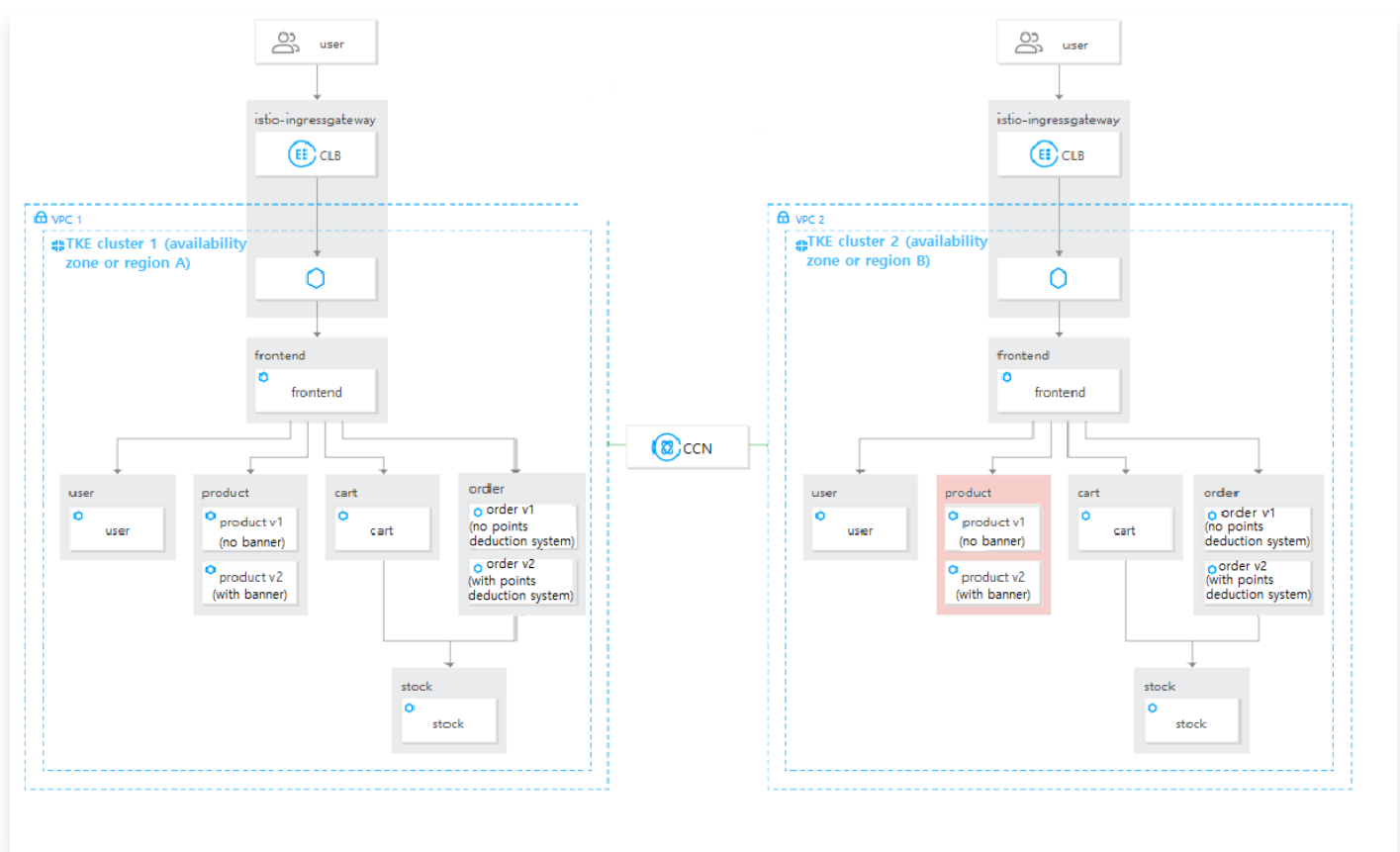
Cross-Cluster Disaster Recovery

Last updated: 2023-12-26 11:35:33

Overview

This document describes how to implement disaster recovery capabilities across regions (AZs) for all services. In this way, when a service in a region (AZ) fails, traffic to the service will be automatically switched to another region (AZ).

Cross-region service disaster recovery is as shown below:

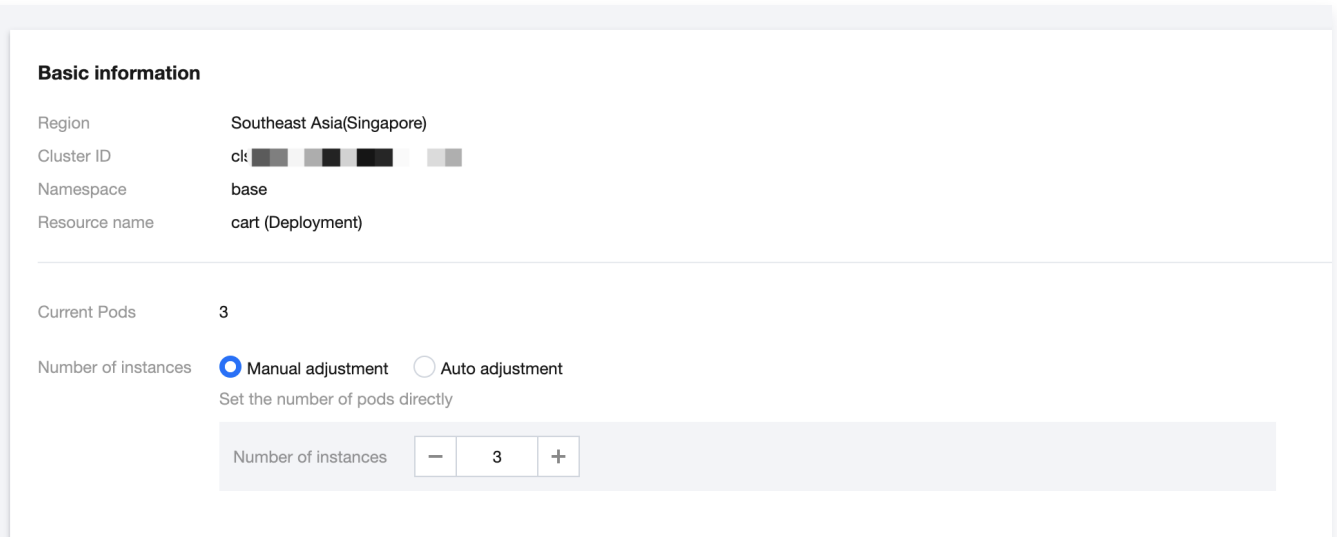


Verification

By default, Tencent Cloud Mesh provides cross-region (AZ) service disaster recovery capabilities, as verified below.

To begin with, set the number of Pods in the `product-v2` Deployment of the primary cluster to `0` in the TKE console. This is to simulate product service failures in the AZ of the primary cluster.

Adjust the number of Pods as shown below:



Basic information

Region	Southeast Asia(Singapore)
Cluster ID	ck
Namespace	base
Resource name	cart (Deployment)

Current Pods 3

Number of instances ☒ Manual adjustment ☐ Auto adjustment

Set the number of pods directly

Number of instances 3

After the configuration, access the demo ecommerce website through the IP address of the edge gateway of the primary cluster. Even if the product service in AZ A of the primary cluster fails, you can still access the product page and view service calls in the floating window in the bottom-left corner. The current page calls the product service in another AZ, indicating that access requests to the product service in AZ A are routed to the product service in AZ B, implementing cross-AZ disaster recovery at the service level.

Deploy two clusters in different regions to implement cross-region disaster recovery at the service level.

Cross-AZ disaster recovery of the product service is as shown below:

Tencent Cloud Mesh Demo

Shop

About

Information

Service

user : James

Your Cart

02
DAYS

10
HOURS

34
MINS

60
SECS

HOT DEAL THIS WEEK

NEW COLLECTION UP TO 50% OFF

SHOP NOW

+

user	region	guangzhou-zoneA
	version	v1
	pod name	user-5f859597b-mwb2d
product	region	guangzhou-zoneB
	version	v2
	pod name	product-v2-8686cb99f5-mvh5k

CoreDNS

envoy

fluentd

©2013-2025 Tencent Cloud. All rights reserved.

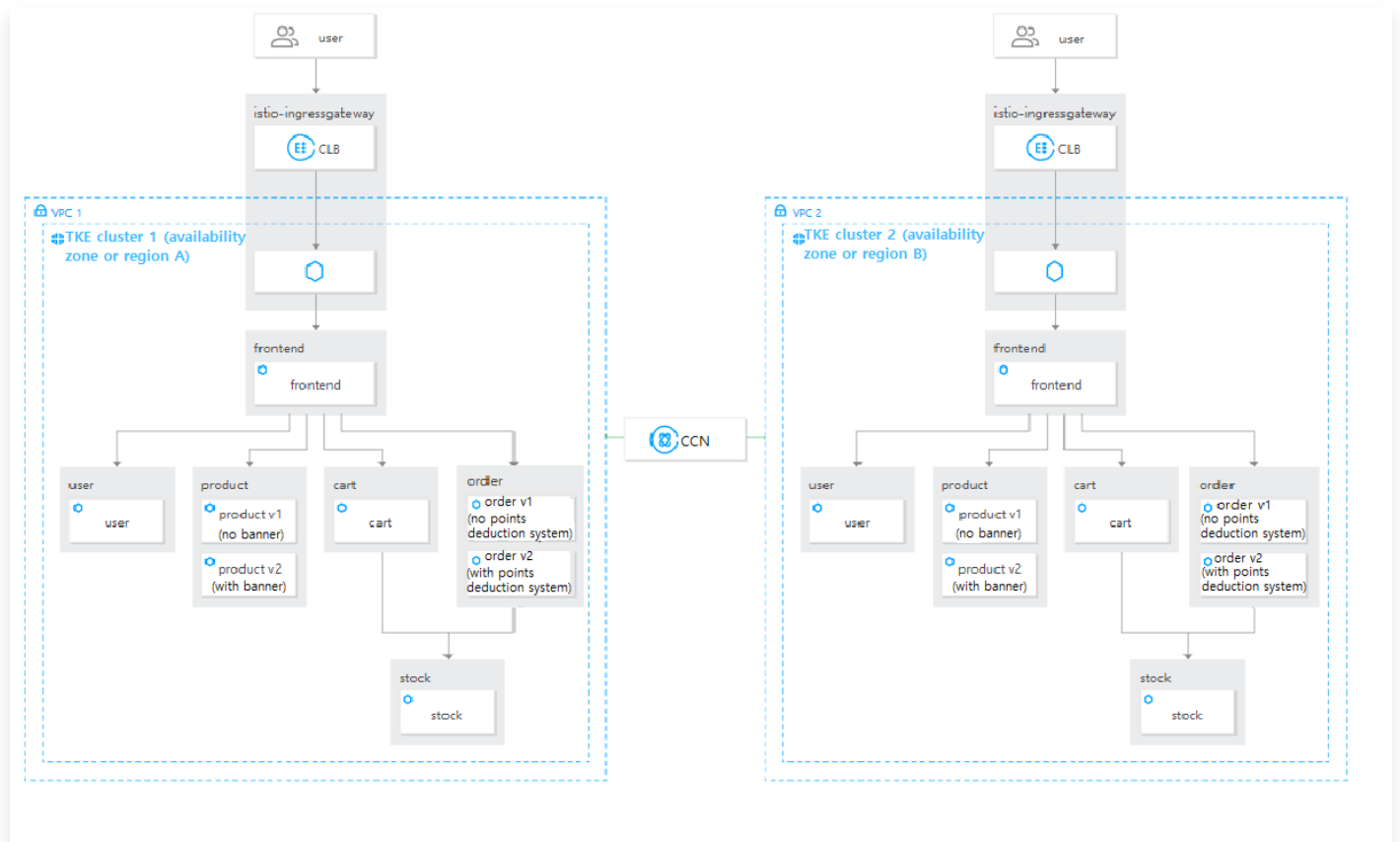
Page 51 of 75

Locality Load Balancing

Last updated: 2023-12-26 11:36:12

Overview

This document describes how to deploy another set of website businesses in a cluster in another AZ to increase availability when the business expands. In particular, two sets of the same website businesses are available in two clusters at the same time. Locality load balancing is as shown below:



Directions

When the two sets of website businesses run normally, the ingress gateway will preferentially route traffic to the frontend service in the local region or AZ, even though the other cluster also has the frontend service. In

addition, the frontend service will access the user, product, order, and cart services in the same AZ, and the order and cart services will access the stock service in the same AZ.

In Kubernetes, the Pod region is determined by the region and zone labels of the deployed node. The zone label is set for a workload in the YAML content of the demo application. First, deploy all the website services to the cluster (sub-cluster) in another AZ:

```
apiVersion: v1
kind: Namespace
metadata:
  name: base
  labels:
    istio.io/rev: 1-6-9
spec:
  finalizers:
    - kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: base
  labels:
    app: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: ccr.ccs.tencentyun.com/chloeyhuang/demo:v202007101540
          imagePullPolicy: Always
          env:
            - name: POD_NAME
```

```
        valueFrom:
          fieldRef:
            fieldPath: metadata.name
      - name: REGION
        value: "guangzhou-zoneB"
    ports:
      - containerPort: 80
---

apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: base
  labels:
    app: frontend
spec:
  ports:
    - port: 80
      name: http
  selector:
    app: frontend
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v1
  namespace: base
  labels:
    app: product
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v1
  template:
    metadata:
```

```
    labels:
      app: product
      version: v1
  spec:
    containers:
      - name: product
        image: ccr.ccs.tencentyun.com/zhulei/testproduct1:v1
        imagePullPolicy: Always
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: REGION
            value: "guangzhou-zoneB"
        ports:
          - containerPort: 7000
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-v2
  namespace: base
  labels:
    app: product
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product
      version: v2
  template:
    metadata:
      labels:
        app: product
        version: v2
    spec:
      containers:
```

```
- name: product
  image: ccr.ccs.tencentyun.com/zhulei/testproduct2: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: product
  namespace: base
  labels:
    app: product
spec:
  ports:
    - port: 7000
      name: http
  selector:
    app: product
---

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
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: stock
```

```
namespace: base
labels:
  app: stock
spec:
  replicas: 1
  selector:
    matchLabels:
      app: stock
  template:
    metadata:
      labels:
        app: stock
    spec:
      containers:
        - name: stock
          image: ccr.ccs.tencentyun.com/zhulei/teststock: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: stock
  namespace: base
  labels:
    app: stock
spec:
  ports:
    - port: 7000
      name: http
  selector:
```

```
    app: stock
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: cart
  namespace: base
  labels:
    app: cart
spec:
  replicas: 3
  selector:
    matchLabels:
      app: cart
  template:
    metadata:
      labels:
        app: cart
    spec:
      containers:
        - name: cart
          image: ccr.ccs.tencentyun.com/zhulei/testcart:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: REGION
              value: "guangzhou-zoneB"
          ports:
            - containerPort: 7000
              protocol: TCP
---

apiVersion: v1
kind: Service
metadata:
  name: cart
```

```
namespace: base
labels:
  app: cart
spec:
  ports:
    - name: http
      port: 7000
      protocol: TCP
  selector:
    app: cart
  type: ClusterIP
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-v1
  namespace: base
  labels:
    app: order
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order
      version: v1
  template:
    metadata:
      labels:
        app: order
        version: v1
    spec:
      containers:
        - name: order
          image: ccr.ccs.tencentyun.com/zhulei/testorder1:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
```

```
        fieldRef:
          fieldPath: metadata.name
      - name: REGION
        value: "guangzhou-zoneB"
    ports:
      - containerPort: 7000
        protocol: TCP
---

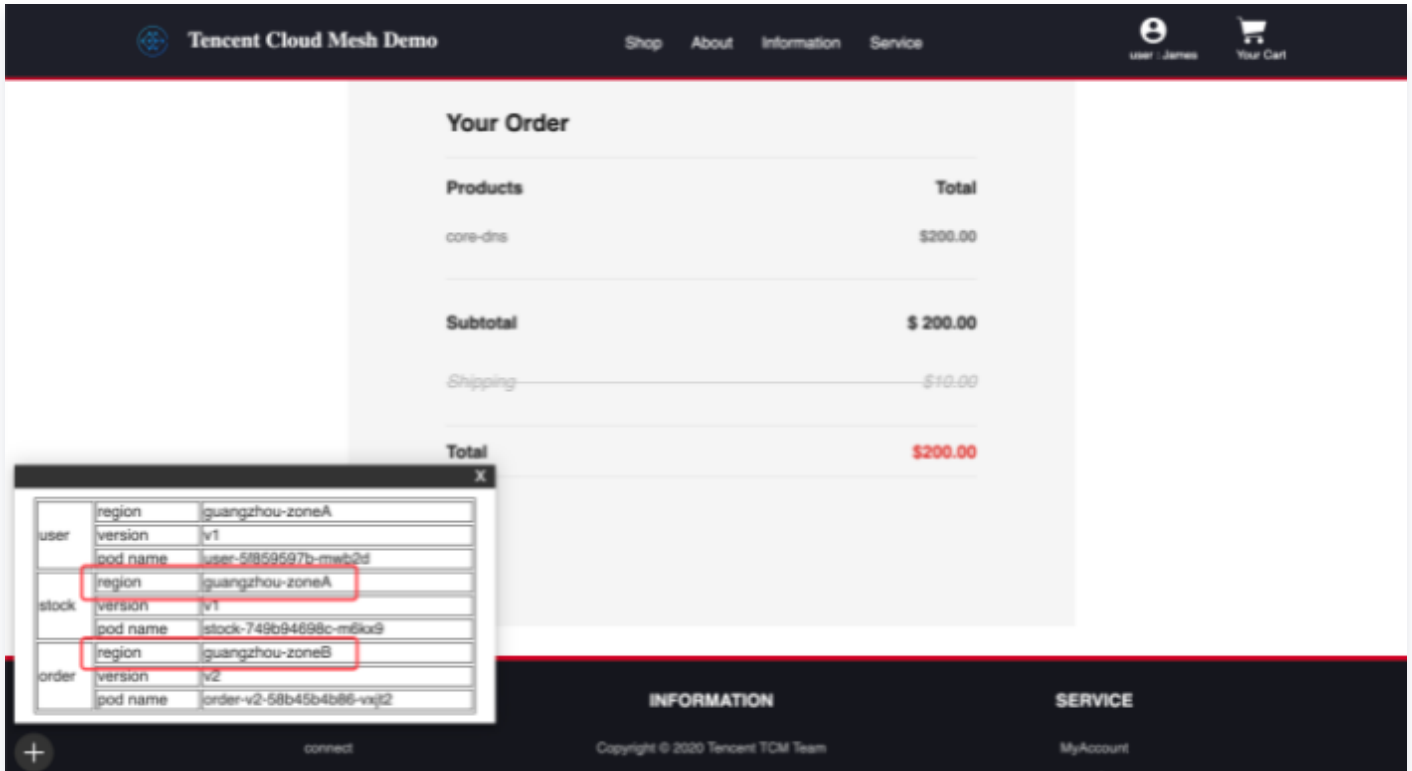
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-v2
  namespace: base
  labels:
    app: order
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order
      version: v2
  template:
    metadata:
      labels:
        app: order
        version: v2
    spec:
      containers:
        - name: order
          image: ccr.ccs.tencentyun.com/zhulei/testorder2:v1
          imagePullPolicy: Always
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: REGION
              value: "guangzhou-zoneB"
```

```
      ports:
        - containerPort: 7000
          protocol: TCP
---

apiVersion: v1
kind: Service
metadata:
  name: order
  namespace: base
  labels:
    app: order
spec:
  ports:
    - name: http
      port: 7000
      protocol: TCP
  selector:
    app: order
  type: ClusterIP
```

After the deployment, locality load balancing will not take effect if health check is not configured. In this case, services are called randomly in the two AZs, without adherence to the nearby access principle.

The order service calls the stock service in another AZ as shown below:



To enable locality load balancing for service access, you need to configure the health check feature for all the services by submitting the following YAML file to the primary cluster:

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: cart
  namespace: base
spec:
  host: cart
  trafficPolicy:
    loadBalancer:
      consistentHash:
        httpHeaderName: UserID
  outlierDetection:
    consecutiveErrors: 5
    interval: 10000ms
    baseEjectionTime: 30000ms
    maxEjectionPercent: 10
    minHealthPercent: 50
  exportTo:
```

```
- '*'

---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: frontend
  namespace: base
spec:
  host: frontend
  trafficPolicy:
    outlierDetection:
      consecutiveErrors: 5
      interval: 10000ms
      baseEjectionTime: 30000ms
      maxEjectionPercent: 10
      minHealthPercent: 50
  exportTo:
    - '*'

---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: order
  namespace: base
spec:
  host: order
  trafficPolicy:
    outlierDetection:
      consecutiveErrors: 5
      interval: 10000ms
      baseEjectionTime: 30000ms
      maxEjectionPercent: 10
      minHealthPercent: 50
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
```



```
    labels:
      version: v2
  exportTo:
    - '*'

---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: product
  namespace: base
spec:
  host: product
  trafficPolicy:
    outlierDetection:
      consecutiveErrors: 5
      interval: 10000ms
      baseEjectionTime: 30000ms
      maxEjectionPercent: 10
      minHealthPercent: 50
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2

---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: stock
  namespace: base
spec:
  host: stock
  trafficPolicy:
    outlierDetection:
      consecutiveErrors: 5
      interval: 10000ms
```

```
    baseEjectionTime: 30000ms
    maxEjectionPercent: 10
    minHealthPercent: 50
  exportTo:
    - '*'

---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: user
  namespace: base
spec:
  host: user
  trafficPolicy:
    outlierDetection:
      consecutiveErrors: 5
      interval: 10000ms
      baseEjectionTime: 30000ms
      maxEjectionPercent: 10
      minHealthPercent: 50
  exportTo:
    - '*'
```

After health check is configured, when a user accesses a website service, browses a product page, adds an item to the cart, or places an order through the edge gateway in the cluster of AZ A, the gateway will route the traffic to the frontend service in the same AZ, and the frontend service will also call the user, cart, order, and stock services in the same AZ. Similarly, when a user accesses a website service through the edge gateway of AZ B, the request will be routed to the frontend service in AZ B, and services in AZ B will be called. You can view the AZ information of the currently called service in the floating window in the bottom-left corner of the demo page. Locality load balancing is as shown below:

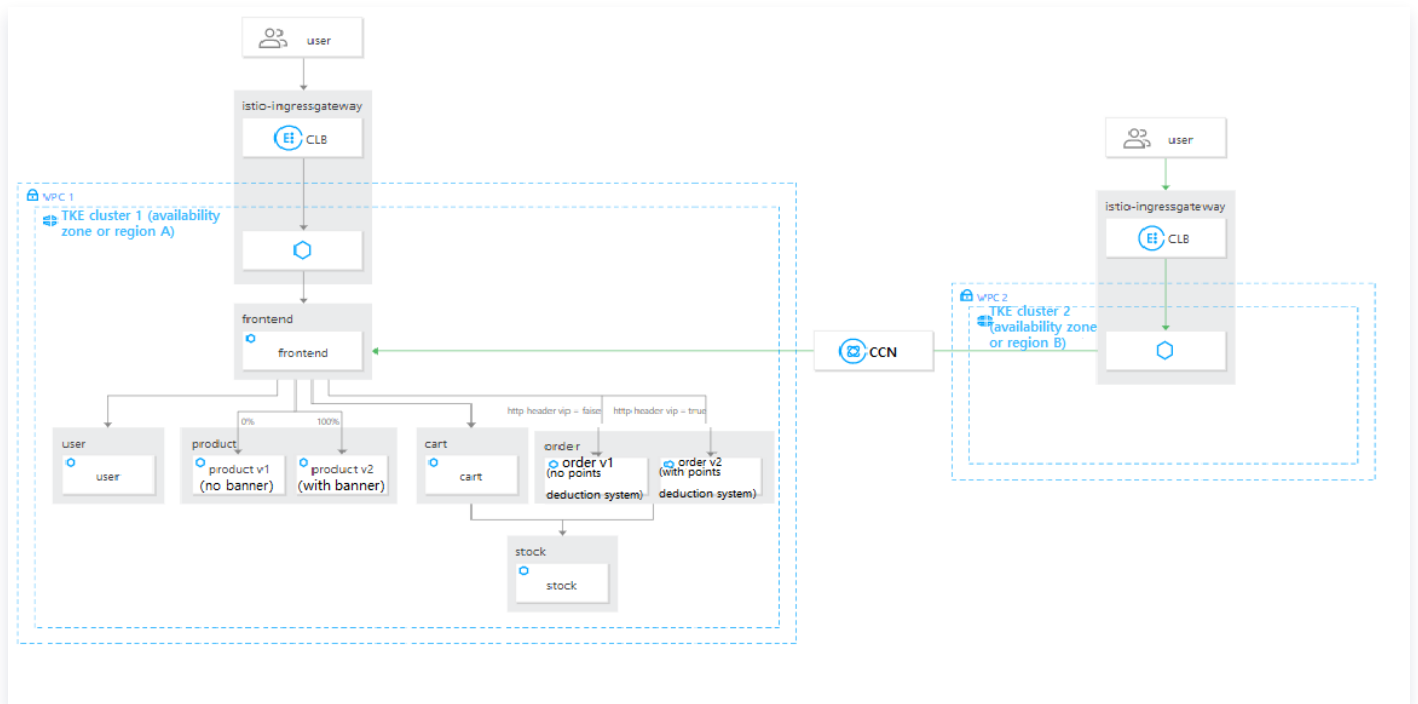


Nearby Access

Last updated: 2023-12-26 11:37:02

Overview

As the ecommerce website business expands, businesses need to be quickly deployed to and accessed from a cluster in another region/AZ. Instead of deploying a complete set of businesses, you only need to deploy an edge gateway in the meshed cluster and configure the listener rule. Nearby access is as shown below:



Directions

Apply the following configuration to the primary cluster. Configure the listener rule of the edge gateway in cluster 2 (sub-cluster), open port 80, and configure the HTTP protocol and `VirtualService` to route the traffic from the edge gateway to the frontend service.

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

```
metadata:
  name: frontend-gw-2
  namespace: base
spec:
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - '*'
  selector:
    app: istio-ingressgateway-1
    istio: ingressgateway

---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: frontend-vs
  namespace: base
spec:
  hosts:
    - '*'
  gateways:
    - base/frontend-gw
    - base/frontend-gw-2
  http:
    - route:
        - destination:
            host: frontend.base.svc.cluster.local
```

After the configuration, the ecommerce website can be accessed through the IP address of the edge gateway in cluster 2 (sub-cluster), as traffic is routed to the service in the primary cluster, even when no ecommerce website services are configured in the Shanghai cluster.

Requests from the secondary cluster are routed to the nearby service in the primary cluster as shown below:

Tencent Cloud Mesh Demo

Shop

About

Information

Service

user : visitor

Your Cart

CATEGORY
TENCENT CLOUD MESH
\$100\$100

CATEGORY
CORE-DNS
\$200\$200

CATEGORY
ENVOY
\$300\$300

CATEGORY
FLUENTD
\$400\$400

CATEGORY
JAEGER
\$500\$500

CATEGORY
VITEESS
\$600\$600

CATEGORY
PROMETHEUS
\$700\$700

+

product	region	guangzhou-zoneA
	version	v1
	pod name	product-v1-5b6f956789-79869

Security Reinforcement Authentication

Last updated: 2023-12-26 11:37:49

Overview

This document describes how to implement mutual authentication of mTLS for all service accesses in the production environment (base namespace) to prevent man-in-the-middle attacks.

Directions

The mTLS mode defaults to `PERMISSIVE`, that is, both mTLS encryption and plaintext connection can be used for service communications.

Log in to the `istio-proxy` container in the TKE console and use plaintext connection to send the `curl http://product.base.svc.cluster.local:7000/product` request to the product service in the production environment (base namespace). In this case, the product service can be accessed via plaintext connection, as shown below:

Login to container

This pod has 2 containers

Container Name	Status	Operation
cart	Running	Log In
istio-proxy	Running	Log In

Shell environment (default environment. You can change it after logging in)

☒ /bin/bash ☐ /bin/sh

The access via plaintext connection is successful as shown below:

```

$ curl http://product.base.svc.cluster.local:7000/product
{"product": [{"name": "Tencent Cloud Mesh", "pid": 1, "price": 100, "url": "https://landscape.cncf.io/logos/containerd.svg"}, {"name": "core-dns", "pid": 2, "price": 200, "url": "https://landscape.cncf.io/logos/core-dns.svg"}, {"name": "envoy", "pid": 3, "price": 300, "url": "https://landscape.cncf.io/logos/envoy.svg"}, {"name": "fluentd", "pid": 4, "price": 400, "url": "https://landscape.cncf.io/logos/fluentd.svg"}, {"name": "helm", "pid": 5, "price": 500, "url": "https://landscape.cncf.io/logos/helm.svg"}], "url": "https://landscape.cncf.io/logos/kubernetes.svg", "info": [{"Service": "product-v2", "Pod": "product-v2-584c7d7797-2d648", "Region": "shanghai"}]}

```

Implement the mTLS mode for service communications in the base namespace by setting the mTLS mode to **STRICT** in the **PeerAuthentication** policy:

Create Authentication YAML edit

Policy Name

Policy Type ☒ PeerAuthentication ☐ RequestAuthentication
Configure the mTLS mode of service communication

Namespace

Specify Service/Gateway Method

Service/Gateway

selector N/A

Policy Content

Mode ☐ DISABLE ☐ PERMISSIVE ☒ STRICT ☐ UNSET
Connection is encrypted with mTLS (TLS with client certificate is required)

Or submit the following YAML file to the primary cluster via kubectl:

```

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

```

After the configuration, log in to the **istio-proxy** container in the TKE console and use plaintext connection to send the `curl http://product.base.svc.cluster.local:7000/product` request to

the product service in the production environment (base namespace). In this case, the product service cannot be accessed via plaintext connection, as shown below:

A terminal window screenshot showing a failed curl command. The terminal has a dark background with light gray text. The command entered is `curl http://product.base.svc.cluster.local:7000/product`. The output shows an error: `curl: (56) Recv failure: Connection reset by peer`. The prompt `/$` is visible at the end of the line.

```
/$ curl http://product.base.svc.cluster.local:7000/product
curl: (56) Recv failure: Connection reset by peer
/$
```

Authorization

Last updated: 2023-12-26 11:38:20

Overview

This document describes how to implement permission controls to prevent running services in the production environment (base namespace) from being accessed by those in the test environment (test namespace).

Directions

Configure the following `AuthorizationPolicy` to prevent services in the test namespace from accessing those in the base namespace.

Configure the authorization rule in the console as shown below:

The screenshot shows the 'Create Authorization Policy' interface in the Tencent Cloud Mesh console. The form is titled 'Create Authorization Policy' and has a 'YAML edit' toggle in the top right corner. The form fields are as follows:

- Policy Name:** base-authz
- Namespace:** base
- Specify Service:** Select Service (button), By labels (button)
- Service/Gateway:** all (dropdown), all (dropdown)
- selector:** N/A
- Policy:** ALLOW (radio), DENY (radio, selected)
- Matching Rule:**
 - Rule 1:** (Delete button)
 - Source:** namespace: test (Add Source button)
 - Operation:** Add Operation (button)
 - Condition:** Add Condition (button)
 - Add Rule:** (button)

At the bottom of the form are 'Save' and 'Cancel' buttons.

Or submit the following YAML file to the primary cluster:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
```

```
name: base-authz
namespace: base
spec:
  action: DENY
  rules:
    - from:
        - source:
            namespaces:
              - test
```

After the configuration, view the Pod log of the client service in the test namespace in the TKE console, which shows that the client service failed to access the user service in the base namespace. In this case, the authorization policy is effective.

After the authorization rule is configured, a failed access is as shown below:

