

**API Gateway**

# **Best Practices**

**Issue**            02  
**Date**             2024-04-03



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

---

# Contents

---

<b>1</b>	<b>Selectively Exposing CCE Workloads with a Dedicated Gateway.....</b>	<b>1</b>
1.1	Introduction.....	1
1.2	Resource Planning.....	2
1.3	General Procedure.....	3
1.4	Implementation Procedure.....	3
<b>2</b>	<b>Selectively Exposing Service Capabilities of a Data Center Using a Dedicated Gateway.....</b>	<b>14</b>
<b>3</b>	<b>Developing a Custom Authorizer with FunctionGraph.....</b>	<b>18</b>
<b>4</b>	<b>Exposing Backend Services Across VPCs Using a Dedicated Gateway.....</b>	<b>22</b>
4.1	Introduction.....	22
4.2	Resource Planning.....	23
4.3	General Procedure.....	24
4.4	Implementation Procedure.....	25
<b>5</b>	<b>Interconnecting a Dedicated Gateway with WAF.....</b>	<b>35</b>
<b>6</b>	<b>Request Throttling 2.0 with a Dedicated Gateway.....</b>	<b>40</b>
6.1	Introduction.....	40
6.2	General Procedure.....	41
6.3	Implementation Procedure.....	42
<b>7</b>	<b>Two-Factor Authentication with a Dedicated Gateway.....</b>	<b>45</b>
7.1	Introduction.....	45
7.2	General Procedure.....	46
7.3	Implementation Procedure.....	47
<b>8</b>	<b>HTTP-to-HTTPS Auto Redirection with a Dedicated Gateway.....</b>	<b>51</b>
8.1	Introduction.....	51
8.2	General Procedure.....	51
8.3	Implementation Procedure.....	52
<b>9</b>	<b>Routing gRPC Service Requests Using a Dedicated Gateway.....</b>	<b>54</b>
9.1	Introduction.....	54
9.2	General Procedure.....	55
9.3	Implementation Procedure.....	55

---

<b>10 Client Authentication with a Dedicated Gateway.....</b>	<b>59</b>
10.1 Solution.....	59
10.2 General Procedure.....	60
10.3 Implementation Procedure.....	60

# 1 Selectively Exposing CCE Workloads with a Dedicated Gateway

## 1.1 Introduction

### Scenario

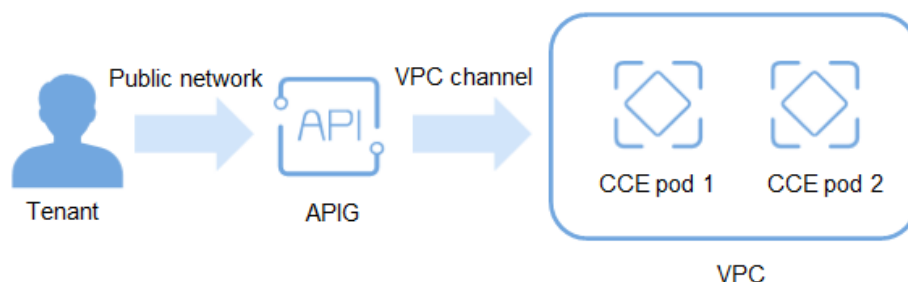
You can use APIG to selectively expose your workloads and microservices in Cloud Container Engine (CCE).

Expose CCE workloads using either of the following methods. **Method 1** is recommended.

- Method 1  
Create a load balance channel on APIG to access pod IP addresses in CCE workloads, dynamically monitoring the changes of these addresses. When opening APIs of a containerized application, specify a load balance channel to access the backend service.
- Method 2  
Import a CCE workload to APIG. APIs and a load balance channel are generated and associated with each other to dynamically monitor pod IP address changes. Expose workloads and microservices in CCE using these APIs.

### Solution Architecture

**Figure 1-1** Accessing CCE workloads (composed of pods) through APIG



## Advantages

- You do not need to set elastic IP addresses, reducing network bandwidth costs.  
Workload addresses in CCE can be accessed through a load balance channel that is manually created or generated by importing a workload.
- Workload pod addresses in CCE can be dynamically monitored and automatically updated by a load balance channel that is manually created or generated by importing a workload.
- CCE workloads can be released by tag for testing and version switching.
- Multiple authentication modes keep access secure.
- Request throttling policies ensure secure access to your backend service.  
Instead of direct access to containerized applications, APIG provides request throttling to ensure that your backend service runs stably.
- Pod load balancing improves resource utilization and system reliability.

## Restrictions

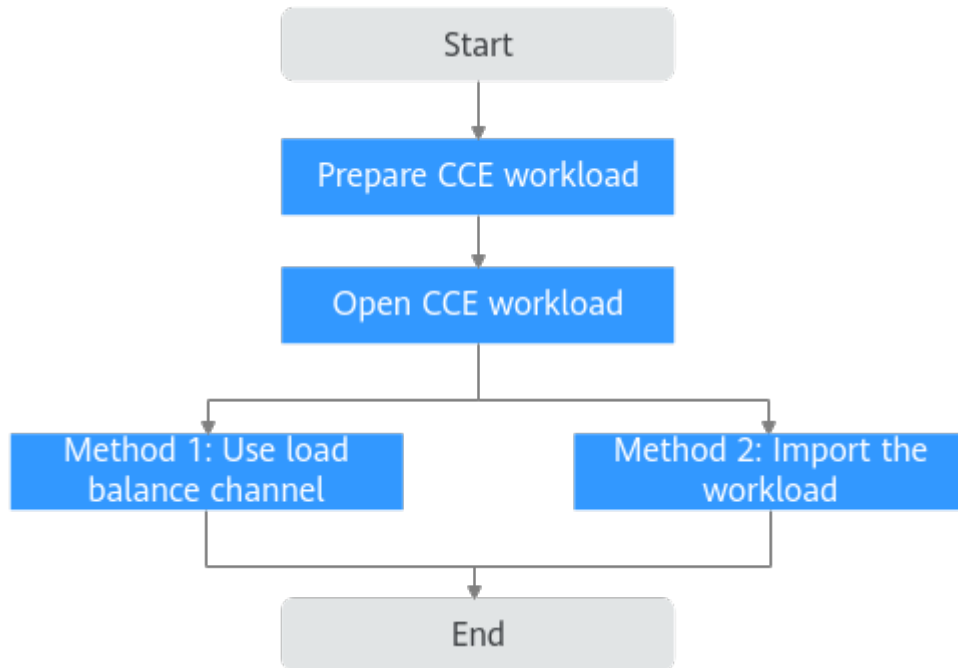
- Only CCE Turbo clusters and CCE clusters using the VPC network model are supported.
- The CCE cluster and your gateway must be in the same VPC or connected.
- If you select a CCE cluster that uses the VPC network model, add the container CIDR block of the cluster in the **Routes** area of the gateway details page.

## 1.2 Resource Planning

Table 1-1 Resource and cost planning

Resource	Quantity
CCE	1
Dedicated gateway	1

## 1.3 General Procedure



1. **Prepare CCE workload**

Before opening a container workload with APIG, buy a CCE cluster that uses the VPC network model or a Turbo cluster on the CCE console.

2. Open CCE workload

**Method 1:** Create a load balance channel on APIG to access pod addresses in the CCE workload.

**Method 2:** Import a CCE workload to APIG. APIs and a load balance channel are generated and associated with each other to access pod IP addresses in the workload.

3. **(Optional) Configure workload label for grayscale release**

Grayscale release is a service release policy that gradually switches traffic from an early version to a later version by specifying the traffic distribution weight.

## 1.4 Implementation Procedure

### Preparing a CCE Workload

**Step 1** Buy a cluster.

1. Log in to the CCE console, and buy a CCE cluster (VPC network model) or Turbo cluster on the **Clusters** page. Select **CCE Cluster** and set **Network Model** to **VPC network**. For details, see [Buying a CCE Cluster](#).
2. After the cluster is created, record the container CIDR block.
3. Add this CIDR block in the **Routes** area of a dedicated gateway.

- a. Log in to the APIG console, and choose **Gateways** in the navigation pane.
- b. Click the gateway name to go to the details page.
- c. Add the container CIDR block in the **Routes** area.

**Step 2** Create a workload.

1. On the **Clusters** page of the CCE console, click the cluster name to go to the details page.
2. In the navigation pane, choose **Workloads**.
3. Click **Create Workload**. Set **Workload Type** to **Deployment**. For details, see the [CCE User Guide](#).

In the **Advanced Settings > Labels and Annotations** area, set pod labels for switching the workload and service version. In this example, set **app=deployment-demo** and **version=v1**. If you create a workload by importing a YAML file, add pod labels in this file. For details about pod labels, see [Pod Labels and Annotations](#)

Add pod labels in a YAML file:

```
spec:
  replicas: 2
  selector:
    matchLabels:
      app: deployment-demo
      version: v1
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: deployment-demo
        version: v1
```

----End

## Method 1: Opening a CCE Workload by Creating a Load Balance Channel

**Step 1** Create a load balance channel.

1. Go to the APIG console, and choose **Gateways** in the navigation pane.
2. Choose **API Management > API Policies**.
3. On the **Load Balance Channels** tab, click **Create Load Balance Channel**.
  - a. Set the basic information.

**Table 1-2** Basic information parameters

Parameter	Description
Name	Enter a name that conforms to specific rules to facilitate search. In this example, enter <b>VPC_demo</b> .
Port	Container port of a workload for opening services. Set this parameter to <b>80</b> , which is the default HTTP port.



Parameter	Description
Routing Algorithm	Select <b>WRR</b> . This algorithm will be used to forward requests to each of the cloud servers you select in the order of server weight.
Type	Select <b>Microservice</b> .

- b. Configure microservice information.

**Table 1-3** Microservice configuration

Parameter	Description
Microservice Type	<b>Cloud Container Engine (CCE)</b> is always selected.
Cluster	Select the <b>purchased cluster</b> .
Namespace	Select a namespace in the cluster. In this example, select <b>default</b> .
Workload Type	Select <b>Deployment</b> . This parameter must be the same as the type of the created workload.
Service Label Key	Select the pod label <b>app</b> and its value <b>deployment-demo</b> of the <b>created workload</b> .
Service Label Value	

- c. Configure a server group.

**Table 1-4** Server group configuration

Parameter	Description
Server Group Name	Enter <b>server_group_v1</b> .
Weight	Enter <b>1</b> .
Backend Service Port	Enter <b>80</b> . This must be the same as the container port in the workload.
Description	Enter "Server group with version v1".
Tag	Select the pod label <b>version=v1</b> of the <b>created workload</b> .

- d. Configure health check.

**Table 1-5** Health check configuration

Parameter	Description
Protocol	Default: <b>TCP</b> .

Parameter	Description
Check Port	Backend server port in the channel.
Healthy threshold	Default: <b>2</b> . This is the number of consecutive successful checks required for a cloud server to be considered healthy.
Unhealthy Threshold	Default: <b>5</b> . This is the number of consecutive failed checks required for a cloud server to be considered unhealthy.
Timeout (s)	Default: <b>5</b> . This is the timeout used to determine whether a health check has failed.
Interval (s)	Default: <b>10</b> . This is the interval between consecutive checks.

- e. Click **Finish**.

In the load balance channel list, click a channel name to view details.

**Step 2** Open an API.

1. Create an API group.
  - a. Choose **API Management > API Groups**.
  - b. Click **Create API Group**, and choose **Create Directly**.
  - c. Configure group information and click **OK**.
2. Create an API and bind the preceding load balance channel to it.
  - a. Click the group name to go to the details page. On the **APIs** tab, click **Create**.
  - b. Configure frontend information and click **Next**.

**Table 1-6** Frontend configuration

Parameter	Description
API Name	Enter a name that conforms to specific rules to facilitate search.
Group	Select the <b>preceding API group</b> .
URL	<p><b>Method:</b> Request method of the API. Set this parameter to <b>ANY</b>.</p> <p><b>Protocol:</b> Request protocol of the API. Set this parameter to <b>HTTPS</b>.</p> <p><b>Subdomain Name:</b> The system automatically allocates a subdomain name to each API group for internal testing. The subdomain name can be accessed 1000 times a day.</p> <p><b>Path:</b> Path for requesting the API.</p>

Parameter	Description
Gateway Response	Select a response to be displayed if the gateway fails to process an API request. Default: <b>default</b> .
Matching	Select <b>Prefix match</b> .
Authentication Mode	API authentication mode. Select <b>None</b> . ( <b>None</b> : Not recommended for actual services. All users will be granted access to the API.)

- c. Configure backend information and click **Next**.

**Table 1-7** Parameters for defining an HTTP/HTTPS backend service

Parameter	Description
Load Balance Channel	Determine whether the backend service will be accessed using a load balance channel. For this example, select <b>Configure</b> .
URL	<b>Method</b> : Request method of the API. Set this parameter to <b>ANY</b> . <b>Protocol</b> : Set this parameter to <b>HTTP</b> . <b>Load Balance Channel</b> : Select the <b>created channel</b> . <b>Path</b> : Path of the backend service.

- d. Define the response and click **Finish**.
3. Debug the API.  
On the **APIs** tab, click **Debug**. Click the **Debug** button in red background. If the status code **200** is returned in the response result, the debugging is successful. Then go to the next step. Otherwise, rectify the error indicated in the error message.
  4. Publish the API.  
On the **APIs** tab, click **Publish**, retain the default option **RELEASE**, and click **OK**. When the exclamation mark in the upper left of the **Publish** button disappears, the publishing is successful. Then go to the next step. Otherwise, rectify the error indicated in the error message.

**Step 3** Call the API.

1. Bind independent domain names to the group of this API.  
On the group details page, click the **Group Information** tab. The debugging domain name is only used for development and testing and can be accessed 1000 times a day. Bind independent domain names to expose APIs in the group.  
Click **Bind Independent Domain Name** to bind registered public domain names. For details, see [Binding a Domain Name](#).
2. Copy the URL of the API.

On the **APIs** tab, copy the API URL. Open a browser and enter the URL. When the defined success response is displayed, the invocation is successful.

**Figure 1-2** Copying the URL



Now, the CCE workload is opened by creating a load balance channel.

----End

## Method 2: Opening a CCE Workload by Importing It

**Step 1** Import a CCE workload.

1. Go to the APIG console, and choose **Gateways** in the navigation pane.
2. Choose **API Management > API Groups**.
3. Choose **Create API Group > Import CCE Workload**.
  - a. Enter information about the CCE workload to import.

**Table 1-8** Workload information

Parameter	Description
Group	Default: <b>New group</b> .
Cluster	Select the <b>purchased cluster</b> .
Namespace	Select a namespace in the cluster. In this example, select <b>default</b> .
Workload Type	Select <b>Deployment</b> . This parameter must be the same as the type of the created workload.
Service Label Key	Select the pod label <b>app</b> and its value <b>deployment-demo</b> of the <b>created workload</b> .
Service Label Value	
Tag	Another pod label <b>version=v1</b> of the workload is automatically selected.

- b. Configure API information.

**Table 1-9** API information

Parameter	Description
Protocol	API request protocol. <b>HTTPS</b> is selected by default.
Request Path	API request path for prefix match. Default: /. In this example, retain the default value.
Port	Enter <b>80</b> . This must be the same as the container port in the workload.
Authentication Mode	Default: <b>None</b> .
CORS	Disabled by default.
Timeout (ms)	Backend timeout. Default: <b>5000</b> .

4. Click **OK**. The CCE workload is imported, with an API group, API, and load balance channel generated.

**Step 2** View the generated API and load balance channel.

1. View the generated API.
  - a. Click the **API group name**, and then view the API name, request method, and publishing status on the **APIs** tab.
  - b. Click the **Backend Configuration** tab and view the bound load balance channel.
2. View the generated load balance channel.
  - a. Choose **API Management > API Policies**.
  - b. On the **Load Balance Channels** tab, click the channel name to view details.
3. Check that this load balance channel is the one bound to the API, and then go to the next step. If it is not, repeat **Step 1**.

**Step 3** Open the API.

Since importing a CCE workload already creates an API group and API, you only need to publish the API in an environment.

1. Debug the API.

On the **APIs** tab, click **Debug**. Click the **Debug** button in red background. If the status code **200** is returned in the response result, the debugging is successful. Then go to the next step.
2. Publish the API.

On the **APIs** tab, click **Publish**, retain the default option **RELEASE**, and click **OK**. When the exclamation mark in the upper left of the **Publish** button disappears, the publishing is successful. Then go to the next step.

**Step 4** Call the API.

1. Bind independent domain names to the group of this API.

On the group details page, click the **Group Information** tab. The debugging domain name is only used for development and testing and can be accessed 1000 times a day. Bind independent domain names to expose APIs in the group.

Click **Bind Independent Domain Name** to bind registered public domain names. For details, see [Binding a Domain Name](#).

2. Copy the URL of the API.

On the **APIs** tab, copy the API URL. Open a browser and enter the URL. When the defined success response is displayed, the invocation is successful.

**Figure 1-3** Copying the URL



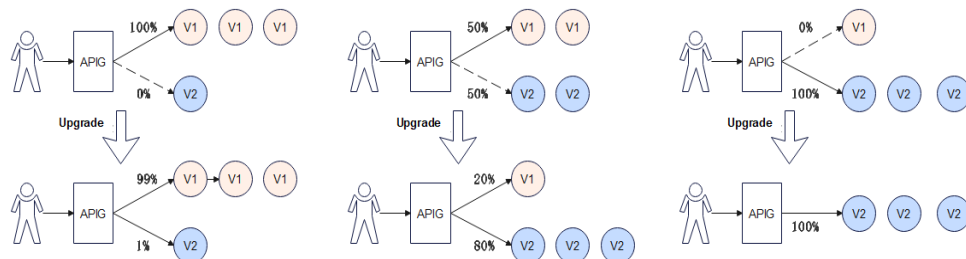
Now, the CCE workload has been opened by importing it.

----End

## (Optional) Configuring Workload Labels for Grayscale Release

Grayscale release is a service release policy that gradually switches traffic from an early version to a later version by specifying the traffic distribution weight. Services are verified during release and upgrade. If a later version meets the expectation, you can increase the traffic percentage of this version and decrease that of the early version. Repeat this process until a later version accounts for 100% and an early version is down to 0. Then the traffic is successfully switched to the later version.

**Figure 1-4** Grayscale release principle



CCE workloads are configured using the pod label selector for grayscale release. You can quickly roll out and verify new features, and switch servers for traffic processing. For details, see [Using Services to Implement Simple Grayscale Release and Blue-Green Deployment](#).

The following describes how to smoothly switch traffic from V1 to V2 through grayscale release.

**Step 1** Create a workload, set a pod label with the same value as the **app** label of the preceding workload. For details, see the [preceding workload](#).

On the workload creation page, go to the **Advanced Settings > Labels and Annotations** area, and set **app=deployment-demo** and **version=v2**. If you create a workload by importing a YAML file, add pod labels in this file.

**Step 2** For the server group with pod label **version=v1**, adjust the traffic weight.

1. On the APIG console, choose **Gateways** in the navigation pane.
2. Choose **API Management > API Policies**.
3. On the **Load Balance Channels** tab, click the name of the [created channel](#).
4. In the **Backend Server Address** area, click **Modify**.
5. Change the weight to **100**, and click **OK**.

Weight is the percentage of traffic to be forwarded. All traffic will be forwarded to the pod IP addresses in server group **server\_group\_v1**.

**Step 3** Create a server group with pod label **version=v2**, then set the traffic weight.

1. In the **Backend Server Address** area, click **Create Server Group**.

**Table 1-10** Server group configuration

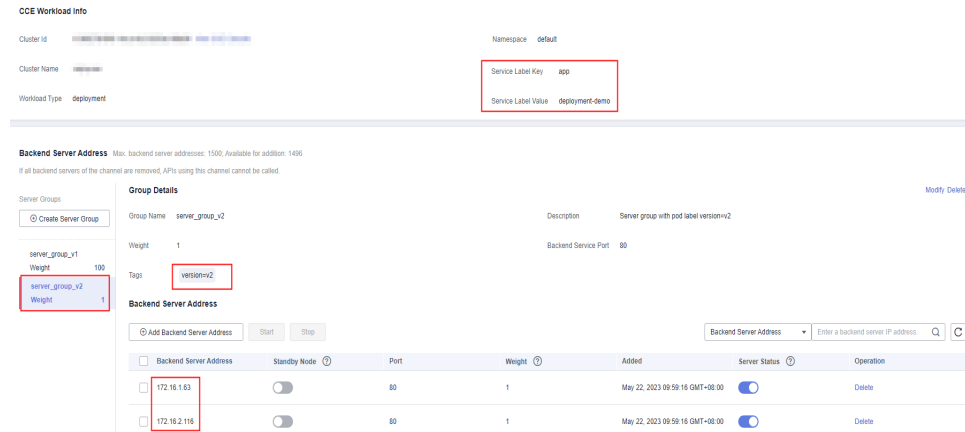
Parameter	Description
Server Group Name	Enter <b>server_group_v2</b> .
Weight	Enter <b>1</b> .
Backend Service Port	Enter <b>80</b> .
Tag	Select pod label <b>version=v2</b> .

2. Click **OK**.

**Step 4** Refresh the backend server addresses.

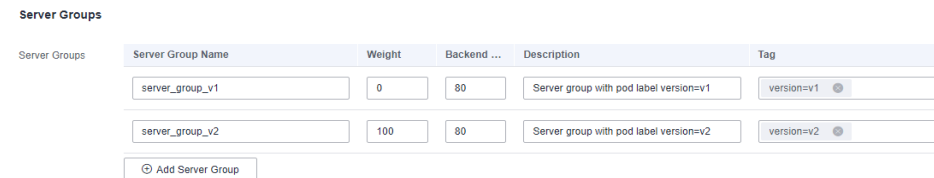
Refresh the page for the backend server addresses. The load balance channel automatically monitors the pod IP addresses of the workload and dynamically adds the addresses as backend server addresses. As shown in the following figure, tags **app=deployment-demo** and **version=v2** automatically match the pod IP addresses (backend server addresses) of the [workload](#).

**Figure 1-5** Pod IP addresses automatically matched



100 of 101 (server group weight of total weight) traffic is distributed to **server\_group\_v1**, and the remaining to the later version of **server\_group\_v2**.

**Figure 1-6** Click **Modify** in the upper right of the page.

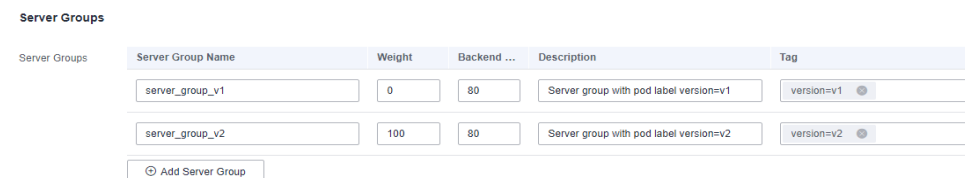


**Step 5** Check that the new features released to V2 through grayscale release are running stably.

If the new version meets the expectation, go to **Step 6**. Otherwise, the new feature release fails.

**Step 6** Adjust the weights of server groups for different versions.

Gradually decrease the weight of **server\_group\_v1** and increase that of **server\_group\_v2**. Repeat **Step 5** to **Step 6** until the weight of **server\_group\_v1** becomes **0** and that of **server\_group\_v2** reaches **100**.



As shown in the preceding figure, all requests are forwarded to **server\_group\_v2**. New features are switched from workload **deployment-demo** of **version=v1** to **deployment-demo2** of **version=v2** through grayscale release. (You can adjust the traffic weight to meet service requirements.)

**Step 7** Delete the backend server group **server\_group\_v1** of **version=v1**.

Now all traffic has been switched to the backend server group of **version=v2**. You can delete the server group of **version=v1**.



1. Go to the load balance channel details page on the APIG console, delete all IP addresses of the server group of **version=v1** in the **Backend Server Address** area.
2. Click **Delete** on the right of this area to delete the server group of **version=v1**.  
The backend server group **server\_group\_v2** of **version=v2** is kept.

----End

# 2 Selectively Exposing Service Capabilities of a Data Center Using a Dedicated Gateway

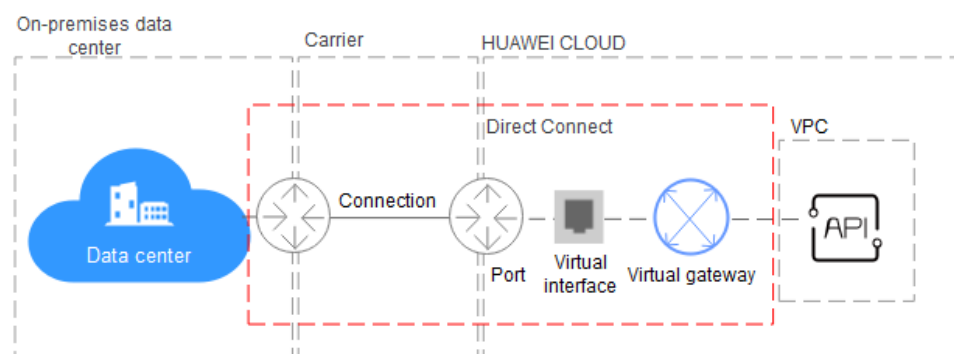
The backend services of APIG can be deployed in the following modes:

- Deployed in a VPC and accessible only using private IP addresses.  
You can create a VPC channel on APIG to enable network routing between APIG and the VPC.
- Deployed on the public network and accessible using a public IP address.
- Deployed in an on-premises data center and not accessible using a public IP address.

If you use a dedicated API gateway, you can set up a connection between your on-premises data center and the gateway (or the VPC bound to the gateway).

This section describes the precautions for using APIG to selectively expose APIs of backend services deployed in a local data center.

**Figure 2-1** Connecting a data center to a dedicated API gateway with Direct Connect



## Connecting a Data Center to APIG

**Step 1** Create a VPC.

For details, see the [Virtual Private Cloud User Guide](#).

To allow APIG to access services in your on-premises data center, bind a VPC to your dedicated gateway, and establish a connection between the data center and VPC.

**Figure 2-2** Creating a VPC

**Basic Information**

Region:

Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.

Name:

IPv4 CIDR Block:

Recommended: 10.0.0.0/8-24 (Select) 172.16.0.0/12-24 (Select) 192.168.0.0/16-24 (Select)

**⚠** The CIDR block 192.168.0.0/16 overlaps with a CIDR block of another VPC in the current region. If you intend to enable communication between VPCs or between a VPC and an on-premises data center, change the CIDR block. [View VPC CIDR blocks in current region](#)

Enterprise Project:  [Create Enterprise Project](#)

---

Advanced Settings ▼ Tag | Description

---

**Default Subnet**

AZ:

Name:

IPv4 CIDR Block:  Available IP Addresses: 251

The CIDR block cannot be modified after the subnet has been created.

IPv6 CIDR Block:  Enable

Associated Route Table:

**NOTE**

- Specify a subnet for your dedicated gateway.
- A connection can be used to connect a local data center to only one VPC. You are advised to bind the same VPC to all your cloud resources to reduce costs.
- If a VPC already exists, you do not need to create a new one.

**Step 2** Buy a dedicated API gateway.

For details, see [Buying a Dedicated Gateway](#).

Figure 2-3 Buying a dedicated gateway

The screenshot displays the AWS API Gateway console configuration for a dedicated gateway. The 'Edition' section is expanded, showing four options:

Edition	Max. Requests per Second	SLA	Price
Basic	2,000	99.95%	\$0.76/hour
Professional	4,000	99.95%	\$3.46/hour
Enterprise	6,000	99.95%	\$5.19/hour
Platinum 1	10,000	99.99%	\$8.65/hour

Other configuration options include:

- Billing Mode: Pay per use
- Region: [Region]
- AZ: AZ1, AZ2, AZ3, AZ7
- Gateway Name: apig-0wme
- Scheduled Maintenance: 22:00:00-02:00:00 GMT+08:00
- Enterprise Project: default
- Public Inbound Access: Enabled
- Public Outbound Access: Enabled
- VPC: vpc-kafka, subnet-kafka
- Security Group: sg-kafka
- Advanced Settings: Configure Now
- Description: [Empty text box]

### Step 3 Buy a connection.

To buy a connection for connecting the data center to APIG (bound VPC), do as follows:

#### 1. Create a Connection

Buy a connection to establish connectivity between your on-premises data center and Huawei Cloud. You are advised to choose **Full Service Installation**, which means that Huawei Cloud will complete the construction.

If you already have a connection between your data center and Huawei Cloud, use the connection instead.

#### 2. Create a Virtual Gateway

The virtual gateway is a logical gateway for accessing the VPC bound to the dedicated gateway.

#### NOTE

Select the subnet that the dedicated gateway uses, to connect to the VPC. For details about the subnet, go to the gateway details page.

#### 3. Create a Virtual Interface

The virtual interface links the connection with the virtual gateway, enabling connectivity between the connection and the VPC of the dedicated gateway.

Configure the remote gateway and remote subnet as the gateway and subnet for accessing the open API of your on-premises data center. For example, if the API calling address of your data center is **http://192.168.0.25:80/{URI}**, configure the remote gateway and remote subnet as those of **192.168.0.25**.

#### 4. Configure Routes

Configure routes at your premises if the subnet of your data center is within the following three segments: 10.0.0.0/8-24, 172.16.0.0/12-24, and 192.168.0.0/16-24.

**Step 4** Verify the network connectivity.

Create another pay-per-use ECS and select the same VPC, subnet, and security group as the dedicated gateway. If the data center can connect to the ECS, the data center can also connect to the dedicated gateway.

----End

## Exposing APIs with the Dedicated Gateway

After you connect the data center to the dedicated gateway, you can expose APIs using the gateway. For details, see [Getting Started](#) in the *API Gateway User Guide*.

When creating an API, specify the backend address as the API calling address of your data center.

# 3 Developing a Custom Authorizer with FunctionGraph

## Overview

The best practice for Huawei Cloud APIG guides you through custom authorizer development.

In addition to IAM and app authentication, APIG also supports custom authentication with your own authentication system, which can better adapt to your business capabilities.

Custom authentication is implemented using the FunctionGraph service. You can create a FunctionGraph function so that APIG can invoke it to authenticate requests for your API. This section uses basic authentication as an example to describe how to implement custom authentication with FunctionGraph.

## Developing a Custom Authentication Function

Create a function on the FunctionGraph console by referring to [Creating a Function for Frontend Custom Authentication](#).

Specify the runtime as Python 3.6.

**Table 3-1** Function configuration

Parameter	Description
Function Type	Default: <b>Event Function</b>
Region	Select the same region as that of APIG.
Function Name	Enter a name that conforms to specific rules to facilitate search.
Agency	An agency that delegates FunctionGraph to access other cloud services. For this example, select <b>Use no agency</b> .
Enterprise Project	The default option is <b>default</b> .

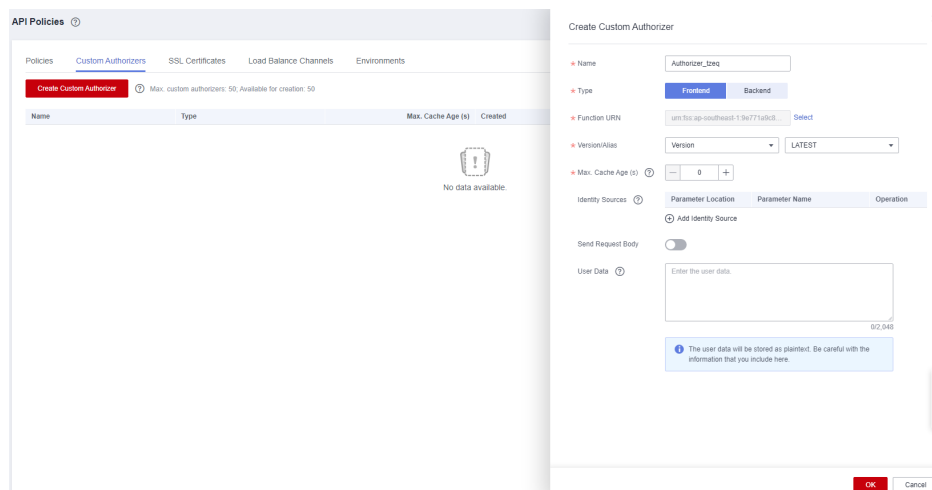
Parameter	Description
Runtime	Select <b>Python 3.6</b> .

On the **Code** tab page, copy the following code to **index.py** (if you are using a dedicated gateway, for which the **authorizer\_context\_support\_num\_bool** parameter has been enabled, the type of **value** in **context** can be boolean or number).

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    # If the authentication information is correct, the username is returned.
    if event["headers"]["authorization"]=="Basic dXN****cmQ=:":
        return {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user_name":"user1"
                }
            })
        }
    else:
        return {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
                "context":{
                    "code": "1001",
                    "message":"incorrect username or password",
                    "authorizer_success": "false"
                }
            })
        }
}
```

## Creating a Custom Authorizer

On the APIG console, go to the **Create Custom Authorizer** page, set **Type** to **Frontend**, select the function created in the preceding section, and click **OK**.



## Creating a Custom Authentication API

Create an API by referring to [Creating an API](#). Set the authentication mode to **Custom**, and select the custom authorizer created in the preceding section. After modifying the API, publish it.

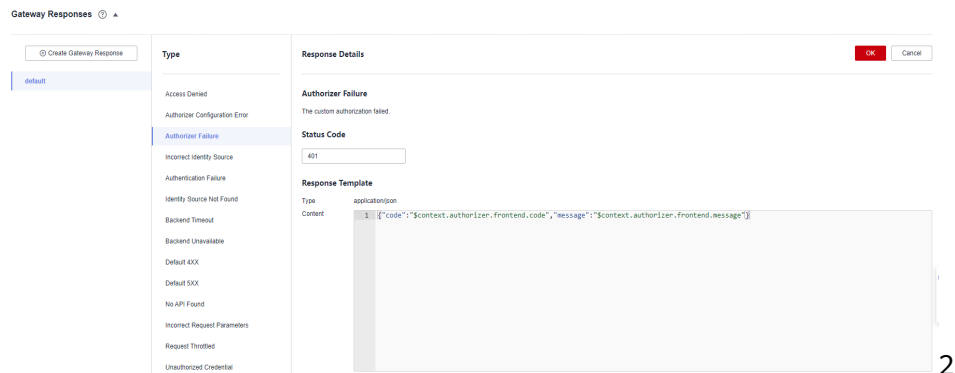
## Setting the Error Response

If incorrect authentication information is carried in a request for the API, the response is displayed as follows:

```
{"error_msg":"Incorrect authentication information: frontend authorizer","error_code":"APIG.0305","request_id":"36e42b3019077c2b720b6fc847733ce9"}
```

To return the field in the function's **context** as the API response (if you are using a dedicated gateway, for which the **authorizer\_context\_support\_num\_bool** parameter has been enabled, the type of **value** in **context** can be boolean or number), modify the gateway response template. On the details page of the group to which the API belongs, navigate to the **Gateway Responses** area on the **Gateway Information** tab, and click **Edit**. Change the status code to **401**, modify the response template with the following code, and click **OK** (no need to add double quotes for variables of the boolean or number type):

```
{"code":"${context.authorizer.frontend.code}","message":"${context.authorizer.frontend.message}","authorizer_success": "${context.authorizer.frontend.authorizer_success}"}
```



After the modification, if incorrect authentication is transferred when calling the API, the status code **401** is returned and the response result is as follows:

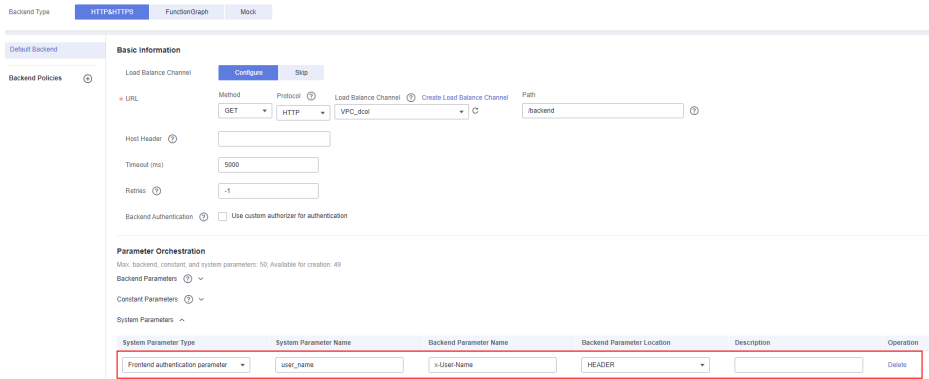
```
{"code":"1001","message":"incorrect username or password","authorizer_success": "false"}
```

## Mapping Frontend Authentication Parameters to Backend Parameters

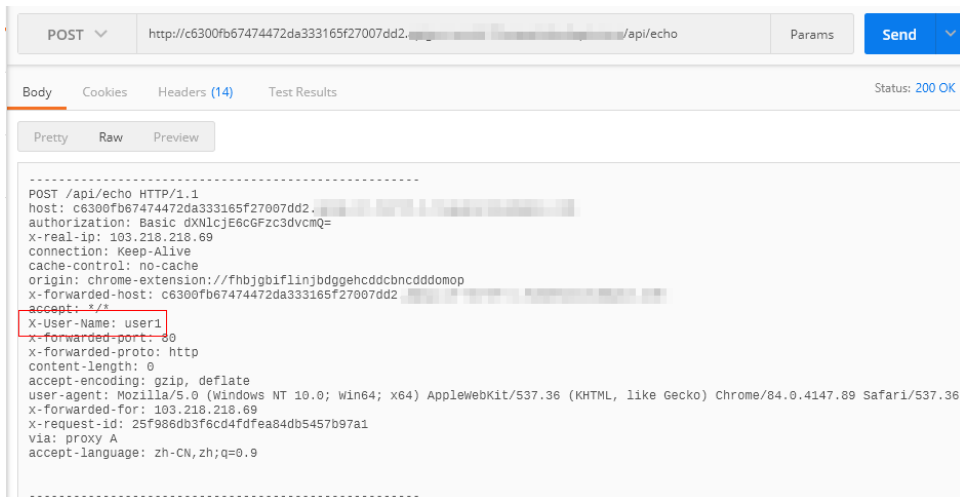
If the authentication is successful, the context information returned by the function can be transferred to the backend of the API. To do this, perform the following configurations:

On the **APIs** page, choose **More > Edit** in the row that contains the API, and go to the **Define Backend Request** page. Add a system parameter, specify the parameter type as **Frontend authentication parameter**, set the parameter name to the content of the **context** field in the function response, and set the name and location of the backend parameter to which you want to map the frontend authentication parameter.





After modifying the API, publish it again. If the authentication information carried in a request for the API is correct, the response result contains the **X-User-Name** header field whose value is the same as that of **user\_name** in the **context** field of the authentication function.



# 4 Exposing Backend Services Across VPCs Using a Dedicated Gateway

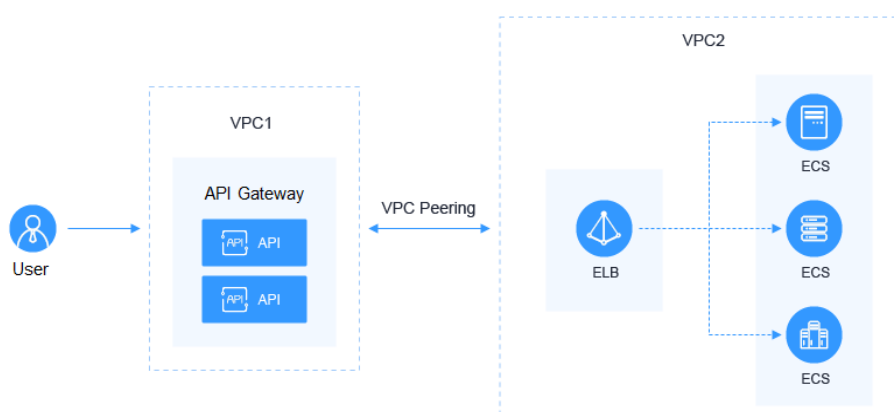
## 4.1 Introduction

### Scenario

If the VPC of your backend server is different from that of your gateway, how do you configure cross-VPC interconnection? This section uses Elastic Load Balance (ELB) as an example to describe how to expose services in a private network load balancer using APIG.

### Solution Architecture

Figure 4-1 Exposing backend services across VPCs



### Advantages

Without modifying the existing network architecture, you can have all requests directly forwarded to your backend server through flexible configuration.

## Restrictions

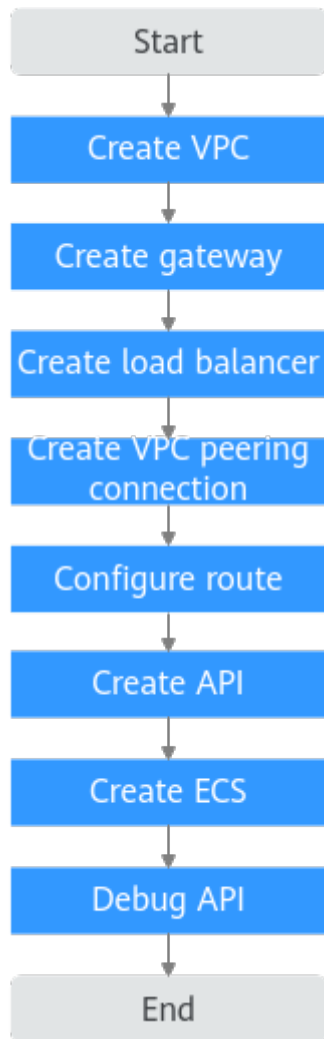
VPC 1, VPC 2, and the VPC CIDR block of your gateway cannot overlap. For details about the VPC CIDR block planning of the gateway, see [Table 4-3](#).

## 4.2 Resource Planning

**Table 4-1** Resource planning

Resource	Quantity
VPC	2
Dedicated gateway	1
Load balancer	1
ECS	1

## 4.3 General Procedure



1. **Create a VPC.**  
Create two VPCs, one for your gateway and the other for your backend service.
2. **Create a gateway.**  
Create a dedicated gateway in VPC 1.
3. **Create a load balancer.**  
Create a load balancer in VPC 2.
4. **Create a VPC peering connection.**  
Create a VPC peering connection to connect VPC 1 and VPC 2.
5. **Configure a route.**  
Configure a route for the dedicated gateway by setting the IP address to the IPv4 CIDR block of VPC 2 where the purchased load balancer is located.
6. **Create an API.**

Create an API and set the backend service address to the IP address of the load balancer.

7. **Create an ECS.**

Create an ECS in VPC 2, and deploy the backend service on the ECS.

8. **Debug the API.**

Verify that the connection to the private network load balancer is successful.

## 4.4 Implementation Procedure

### Creating a VPC

**Step 1** Log in to the network console.

**Step 2** In the navigation pane, choose **Virtual Private Cloud > My VPCs**.

**Step 3** On the **Virtual Private Cloud** page, click **Create VPC**, and configure the parameters by referring to [Table 4-2](#) and [Table 4-3](#). For details, see sections "Creating a VPC" and "Creating a Subnet for the VPC" in the *Virtual Private Cloud User Guide*.

**Basic Information**

Region: [Region selected]

Name: VPC1

IPv4 CIDR Block: 192.168.0.0 / 16

Recommended: 10.0.0.0/8-24 ( Select ) 172.16.0.0/12-24 ( Select ) 192.168.0.0/16-24 ( Select )

**Warning:** The CIDR block 192.168.0.0/16 overlaps with a CIDR block of another VPC in the current region. If you intend to enable communication between VPCs or between a VPC and an on-premises data center, change the CIDR block. View VPC CIDR blocks in current region.

Enterprise Project: default

**Default Subnet**

Name: subnet-bf15

IPv4 CIDR Block: 192.168.0.0 / 24

IPv6 CIDR Block:  Enable

Associated Route Table: Default

**Table 4-2** Configuration information

Parameter	Description
Region	Select a region.
Name	Enter <b>VPC1</b> . This VPC will be used to run a gateway.

Parameter	Description
Enterprise Project	Select <b>default</b> .
Name	A subnet is automatically created when you create a VPC.

**Table 4-3** VPC CIDR block planning

VPC 1	VPC of APIG	VPC 2
10.X	172.31.0.0/16	Must be different from VPC 1 and the VPC of the gateway.
172.X	192.168.0.0/16	
192.X	172.31.0.0/16	

**Step 4** Click **Create Now**.

**Step 5** Repeat **Step 3** to **Step 4** to create **VPC2** for running your backend service.

----End

## Creating a Gateway

**Step 1** Go to the APIG console.

**Step 2** In the navigation pane, choose **Gateways**.

**Step 3** Click **Buy Gateway**.

**Table 4-4** Gateway information

Parameter	Description
Billing Mode	Billing mode of the gateway. Select <b>Pay-per-use</b> .
Region	Select the region where the gateway is located. It must be the same as the region of VPC 1.
AZ	The AZ where the gateway is located. Select <b>AZ1</b> .
Gateway Name	Enter a name that conforms to specific rules to facilitate search.
Edition	Select <b>Professional</b> . The edition cannot be changed after the gateway is created.
Scheduled Maintenance	Select a time period when the gateway can be maintained by technical support engineers. A period with low service traffic is recommended. For this example, retain the default value <b>22:00:00---02:00:00</b> .
Enterprise Project	Select the enterprise project to which the gateway belongs. For this example, retain the default value <b>default</b> .
Network	Select <b>VPC 1</b> and a subnet.
Security Group	Click <b>Manage Security Groups</b> and create a security group. Ensure that you have selected <b>default</b> for <b>Enterprise Project</b> .
Description	Description of the gateway.

**Step 4** Click **Next**.

**Step 5** If the gateway configurations are correct, read and confirm your acceptance of the customer agreement and privacy statement, and click **Pay Now**.

----End

## Buying a Load Balancer

**Step 1** Return to the network console.

**Step 2** In the navigation pane, choose **Elastic Load Balance > Load Balancers**.

**Step 3** Click **Buy Elastic Load Balancer**.

**Step 4** Configure the load balancer information. For details, see section **Load Balancer** in the *Elastic Load Balance User Guide*.

**Basic Information**

Type: **Dedicated** | Shared | Learn more

Billing Mode: **Yearly/Monthly** | **Pay-per-use**

Region: [Region]

AZ: **AZ1**  
You can choose to deploy the load balancer in multiple AZs for higher availability.

---

**Network Configuration**

IP as a Backend:

Network Type:  Public IPv4 network (Public network traffic) |  **Private IPv4 network** (Private network traffic) |  IPv6 network (Public and private network traffic)

VPC: **vpc2** | View VPCs

Subnet: **subnet-02192.168.0.0/24** | View Subnet  
Available private IP addresses: 251

IPv4 Address: **Automatically assign IP a...**

---

Specifications

The specification determines the protocol of the listener you can add to your load balancer.

Application load balancing (HTTP/HTTPS) | **Network load balancing (TCP/UDP)**

Specifications	CPS	Maximum Connections	Bandwidth (Mbit/s)	LCU
<input checked="" type="radio"/> Small I	10,000	500,000	50	10
<input type="radio"/> Small II	20,000	1,000,000	100	20
<input type="radio"/> Medium I	40,000	2,000,000	200	40
<input type="radio"/> Medium II	80,000	4,000,000	400	80
<input type="radio"/> Large I	200,000	10,000,000	1,000	200
<input type="radio"/> Large II	400,000	20,000,000	2,000	400

Selected specifications: **Network load balancing (TCP/UDP) | Small I**  
ebv3.basic.1az | 10 LCU/s

---

Name: **eb-2jy**

Enterprise Project: **default** | Create Enterprise Project

Advanced Settings: Backend Subnet | Description | Tag

**Table 4-5** Load balancer parameters

Parameter	Description
Type	Type of the load balancer.
Billing Mode	By default, <b>Pay-per-use</b> is selected.
Region	Select the region where the load balancer is located. It must be the same as the region of VPC 2.
AZ	The AZ where the load balancer is located. Select <b>AZ1</b> .
Network Type	Select <b>Private Network</b> .
VPC	Select <b>VPC 2</b> .
Subnet	Select a subnet.
Specification	Select <b>Network load balancing</b> .
Name	Enter a load balancer name that conforms to specific rules to facilitate search.
Enterprise Project	Select <b>default</b> .



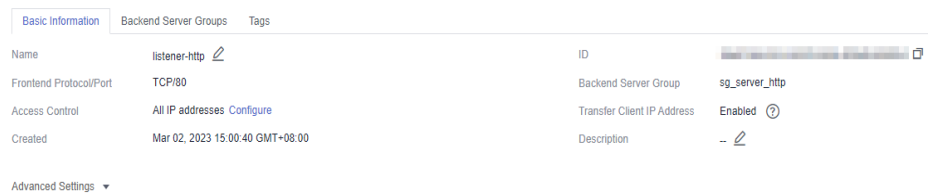
**Step 5** Click **Next**.

**Step 6** Confirm the configuration and click **Submit**.

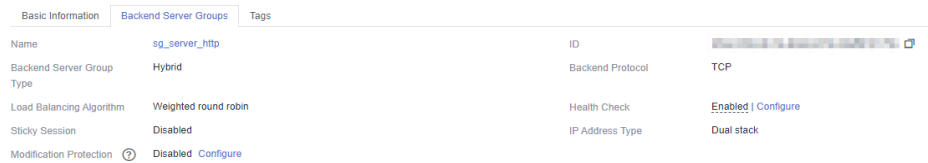
**Step 7** Add a listener.

1. Click the name of the load balancer. On the **Listeners** tab page, click **Add Listener**.
2. Configure the listener name, frontend protocol, and port, and click **Next**.
3. Configure the backend server group name, backend protocol, and load balancing algorithm. Then click **Next**.
4. Add backend servers and click **Next**.
5. Click **Submit**. The following figure shows the configuration.

**Figure 4-2** Listener information



**Figure 4-3** Backend server group information



----End

## Creating a VPC Peering Connection

**Step 1** In the navigation pane, choose **Virtual Private Cloud > VPC Peering Connections**.

**Step 2** Click **Create VPC Peering Connection** and configure the parameters.

**Table 4-6** Configuring a VPC peering connection

Parameter	Description
Name	Enter a VPC peering connection name that conforms to specific rules to facilitate search.
Local VPC	Select <b>VPC 1</b> .
Account	By default, <b>My account</b> is selected.
Peer Project	Select a project
Peer VPC	Select <b>VPC 2</b> .

**Step 3** Click **OK**.

**Step 4** In the displayed dialog box, click **Add Route** to go to the VPC peering connection details page.

**Step 5** On the **Local Routes** tab page, click **Route Tables**.

1. Under **Routes**, click **Add Route**.
2. In the displayed dialog box, enter the route information.
  - **Destination:** Enter the service address displayed on the details page of the **load balancer**.
  - **Next Hop Type:** Select **VPC peering connection**.
3. Click **OK**.

**Figure 4-4** Local routes

Destination	Next Hop Type	Next Hop	Route Table	Description
10.101.0.191/32	VPC peering connection	pc-01(4655d70e-275a-403-88e6-62016c2c3887)	rtb-vpc-001	--

**Step 6** Go to the **Peer Routes** tab page, and click **Route Tables**.

1. Under **Routes**, click **Add Route**.
2. In the displayed dialog box, enter the route information.
  - **Destination:** Enter the private outbound address displayed on the details page of the **dedicated gateway**.
  - **Next Hop Type:** Select **VPC peering connection**.
3. Click **OK**.

**Figure 4-5** Peer routes

Destination	Next Hop Type	Next Hop	Route Table	Description
192.168.0.180/32	VPC peering connection	peering-v1v2(2a1733a3-5315-4e90-89ce-bee5ee9b263)	rtb-vpc-002	--
192.168.0.239/32	VPC peering connection	peering-v1v2(2a1733a3-5315-4e90-89ce-bee5ee9b263)	rtb-vpc-002	--

----End

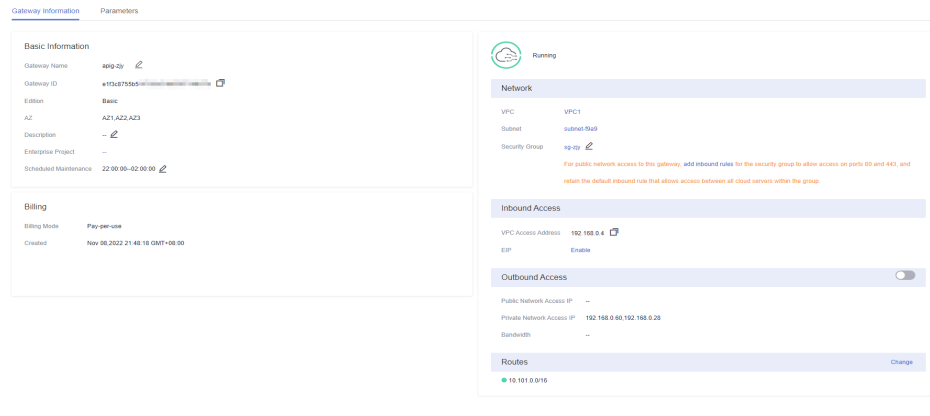
## Configuring a Route

**Step 1** Return to the APIG console.

**Step 2** In the navigation pane, choose **Gateways**.

**Step 3** Click the name of the created **dedicated gateway** or click **Access Console**.

**Step 4** Click **Change** in the **Routes** area, enter the IPv4 CIDR block of VPC 2 where the load balancer you purchased is located.



**Step 5** Click **Save**.

----End

## Creating an API

**Step 1** On the APIG console, choose **API Management > APIs**, and click **Create API**.

**Step 2** Configure the frontend information and click **Next**.

**Table 4-7** Frontend configuration

Parameter	Description
API Name	Enter a name that conforms to specific rules to facilitate search.
Group	The default option is <b>DEFAULT</b> .
URL	<b>Method:</b> Request method of the API. Set this parameter to <b>GET</b> . <b>Protocol:</b> Request protocol of the API. Set this parameter to <b>HTTPS</b> . <b>Subdomain Name:</b> The system automatically allocates a subdomain name to each API group for internal testing. The subdomain name can be accessed 1000 times a day. <b>Path:</b> Path for requesting the API.
Gateway Response	Select a response to be displayed if the gateway fails to process an API request. The default gateway response is <b>default</b> .
Authentication Mode	API authentication mode. Select <b>None</b> .

**Step 3** Configure the backend information and click **Next**.

**Table 4-8** Parameters for defining an HTTP/HTTPS backend service

Parameter	Description
Load Balance Channel	Determine whether the backend service will be accessed using a load balance channel. For this example, select <b>Skip</b> .
URL	<p><b>Method:</b> Request method of the API. Set this parameter to <b>GET</b>.</p> <p><b>Protocol:</b> Set this parameter to <b>HTTP</b>.</p> <p><b>Backend Address:</b> Enter the service address of the load balancer you purchased.</p> <p><b>Path:</b> Path of the backend service.</p>

**Step 4** Define the response and click **Finish**.

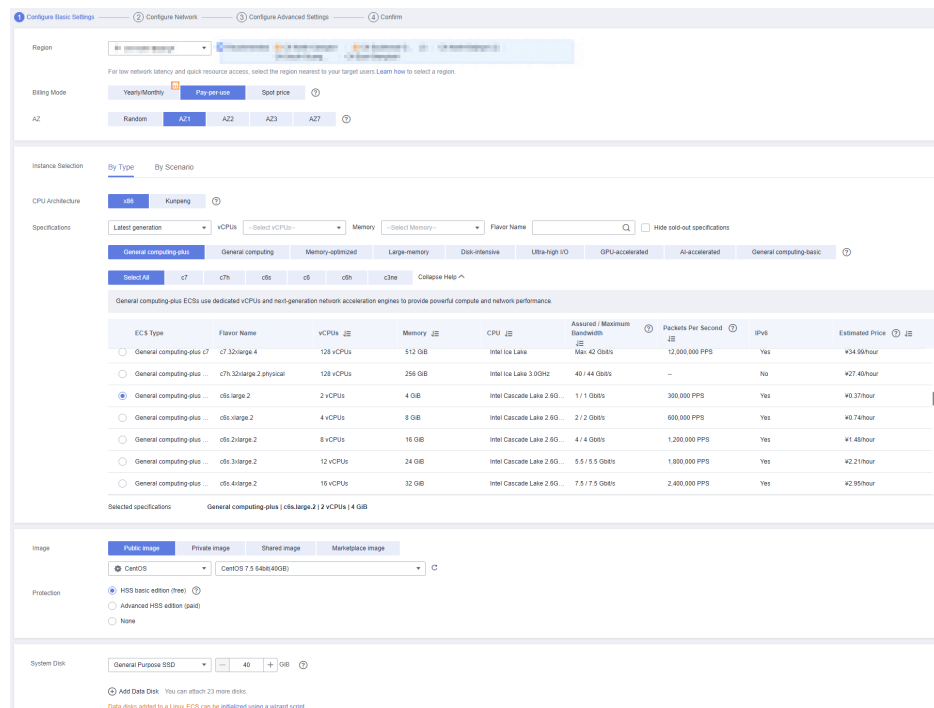
----End

## Buying an ECS

**Step 1** Log in to the cloud server console.

**Step 2** Click **Buy ECS**.

**Step 3** Configure the basic settings and click **Next: Configure Network**.



**Table 4-9** Basic settings

Parameter	Description
Region	Select the region where the ECS is located. It must be the same as the region of VPC 2.
Billing Mode	Select <b>Pay-per-use</b> .
AZ	Select the AZ where the ECS is located.
CPU Architecture	The default option is <b>x86</b> .
Specifications	Select specifications that match your service planning.
Image	Select an image that matches your service planning.

**Step 4** Configure the network settings and click **Next: Configure Advanced Settings**.

**Table 4-10** Network settings

Parameter	Description
Network	Select <b>VPC 2</b> and a subnet.
Security Group	Select the security group created for the <b>dedicated gateway</b> .
EIP	Select <b>Not required</b> .

**Step 5** Configure advanced settings and click **Next: Confirm**.

**Table 4-11** Advanced settings

Parameter	Description
ECS Name	Enter a name that conforms to specific rules to facilitate search.
Login Mode	Credential for logging in to the ECS. The default option is <b>Password</b> .
Username	The default user is <b>root</b> .
Password	Set a password for logging in to the ECS.
Confirm Password	Enter the password again.

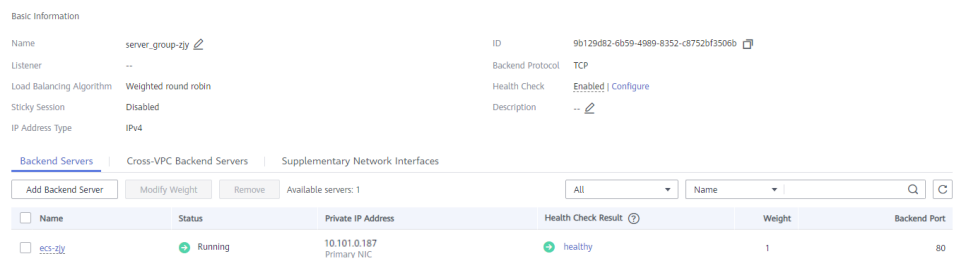
**Step 6** Confirm the configuration and select enterprise project **default**.

**Step 7** Read and confirm your acceptance of the agreement. Then click **Submit**.

----End

## Debugging the API

**Step 1** On the **Backend Server Groups** tab page of **the load balancer**, add **the ECS**.



**Step 2** Go to the **API Management > APIs** page of **the dedicated gateway**, and choose **More > Debug** in the row that contains **the API you created**.

**Step 3** Enter the request parameters and click **Debug**.

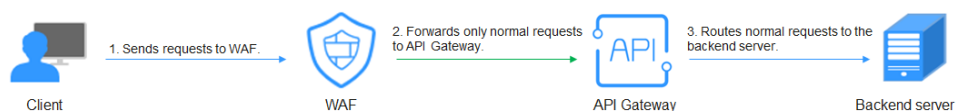
If the status code is **200**, the debugging is successful.

----End

# 5 Interconnecting a Dedicated Gateway with WAF

To protect API Gateway and your backend servers from malicious attacks, deploy Web Application Firewall (WAF) between API Gateway and the external network.

Figure 5-1 Access to a backend server



## NOTE

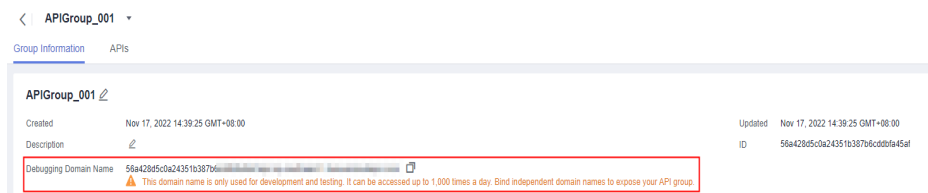
The following instructions are based on the new console. If you are using the old console, see the [API Gateway User Guide](#).

## (Recommended) Solution 1: Register API Group Debugging Domain Name on WAF and Use the Domain Name to Access the Backend Service

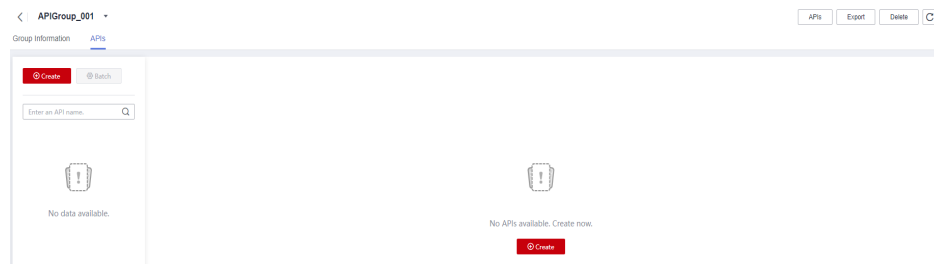
API groups provide services using domain names for high scalability.

- Step 1** Create an API group in a gateway, record the domain name, and create an API in the group.

Figure 5-2 Creating an API group and recording the debugging domain name



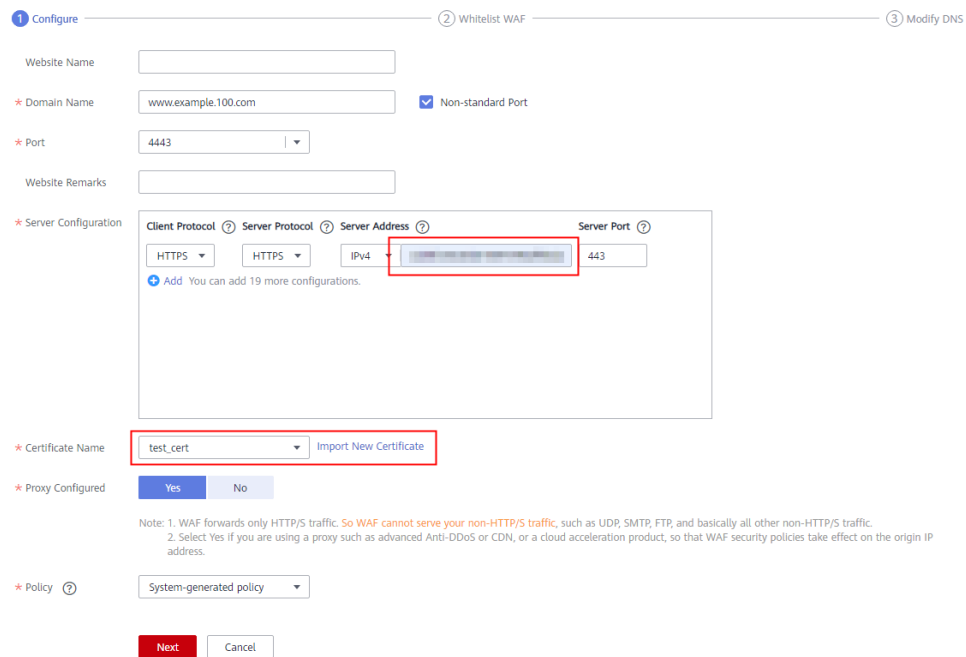
**Figure 5-3** Creating an API



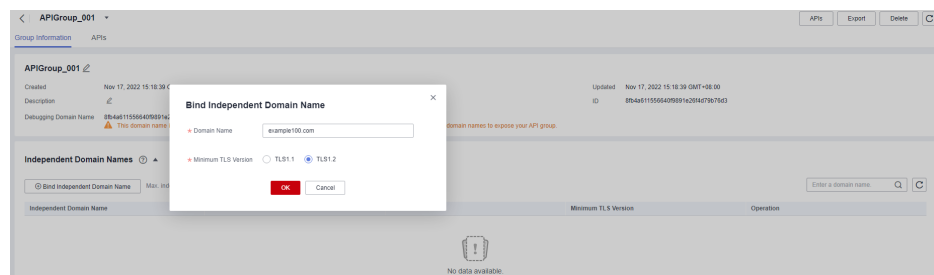
**Step 2** Go to the WAF console, and add a domain name by configuring **Server Address** as the API group domain name and adding a certificate. For details, see [Connection Process \(Cloud Mode\)](#).

**NOTE**

You can use a public network client to access WAF with its domain name. WAF then uses the same domain name to forward your requests to API Gateway. There is no limit on the number of requests that API Gateway can receive for the domain name.



**Step 3** On the gateway details page, bind the domain name to the API group.





**Step 4** Enable `real_ip_from_xff` and set the parameter value to `1`.

**NOTE**

When a user accesses WAF using a public network client, WAF records the actual IP address of the user in the HTTP header **X-Forwarded-For**. API Gateway resolves the actual IP address of the user based on the header.

Parameter	Default Value	Value Range	Current Value	Updated	Operation
request_rate_limit	200 per second	1-1,000,000 per second	200 per second	--	Modify
request_body_size	12 MB	1-8,536 MB	12 MB	--	Modify
backend_timeout	60,000 ms	1-600,000 ms	60,000 ms	--	Modify
ssl_verify	Off	On/Off	Off	--	Modify
ssl_verify	Off	On/Off	Off	--	Modify
ssl_verify	Off	On/Off	Off	--	Modify
ssl_verify	Off	On/Off	Off	--	Modify
ssl_verify	Off	On/Off	Off	--	Modify
ssl_verify	Off	On/Off	Off	--	Modify
ssl_certificate			Off	--	Modify
ssl_certificate	ECDSA-ECDHE-AES256-GCM-SHA384	ECDSA-ECDHE-AES256-GCM-SHA384	ECDSA-ECDHE-AES256-GCM-SHA384	--	Modify
real_ip_from_xff	Off	On/Off	On	Nov 17, 2022 14:57:29 GMT+08:00	Modify
Parameter	Default Value	Value Range	Current Value	Updated	Operation
ssl_index	-1	Valid int32 value	1	Nov 17, 2022 14:57:29 GMT+08:00	Modify
vpc_name_modifiable	On	On/Off	On	Nov 2, 2022 19:57:59 GMT+08:00	Modify

----End

**Solution 2: Forward Requests Through the DEFAULT Group and Use Gateway Inbound Access Address to Access the Backend Service from WAF**

**Step 1** View the inbound access addresses of your gateway. There is no limit on the number of times the API gateway can be accessed using an IP address.

- **VPC Ingress Address:** VPC access address
- **EIP:** public network access address

The screenshot shows the 'Basic Information' and 'Network' sections of the API Gateway console. The 'Basic Information' section includes details like Gateway Name (api-199c), Gateway ID (483419e0-...), Edition (Professional), AZ (AZ1, AZ2, AZ3), and Description. The 'Network' section shows the gateway is 'Running' and lists the VPC (vpc-kafka), Subnet (subnet-kafka), and Security Group (sg-kafka). Under 'Inbound Access', it displays the VPC Access Address (192.168.0.147) and EIP (EIP) (Unbound EIP, Bandwidth: 5 Mbps).

**Step 2** Create an API in the **DEFAULT** group.

The screenshot shows the 'DEFAULT' group page in the API Gateway console. It features a 'Create' button and a search bar for API names. Below the search bar, there are two placeholder cards indicating 'No data available.' and 'No APIs available. Create now.' with a 'Create' button.

**Step 3** Go to the WAF console, add a domain name by configuring **Server Address** as an **inbound access address** of your API gateway and adding a certificate, and then

copy the WAF back-to-source IP addresses. For details, see [Connection Process \(Cloud Mode\)](#).

**NOTE**

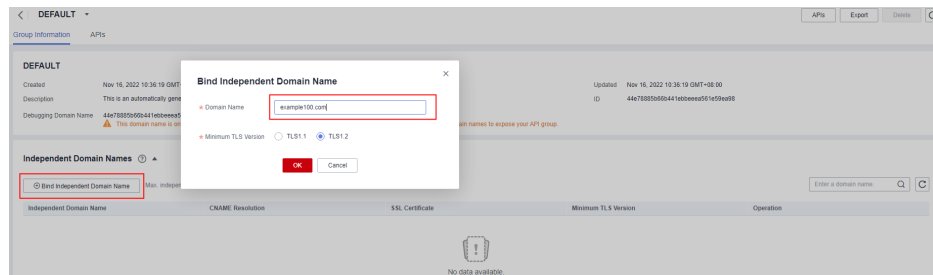
- If WAF and your gateway are in the same VPC, set **Server Address** to the VPC access address.
- If your gateway is bound with an EIP, set **Server Address** to the EIP.

The screenshot shows the configuration page for a WAF instance. It includes the following fields and options:

- Website Name:** A text input field.
- Domain Name:** A text input field containing "www.example.100.com".
- Non-standard Port:** A checked checkbox.
- Port:** A dropdown menu set to "4443".
- Website Remarks:** A text input field.
- Server Configuration:** A table with columns for Client Protocol (HTTPS), Server Protocol (HTTPS), Server Address (IPV4), and Server Port (8005). A red box highlights the Server Address field.
- Certificate Name:** A dropdown menu set to "test\_cert". A red box highlights this field.
- Proxy Configured:** Radio buttons for "Yes" and "No".
- Policy:** A dropdown menu set to "System-generated policy".

Buttons for "Next" and "Cancel" are located at the bottom of the form.

**Step 4** On the gateway details page, bind the domain name to the **DEFAULT** group.



**Step 5** Enable **real\_ip\_from\_xff** and set the parameter value to **1**.

**NOTE**

When a user accesses WAF using a public network client, WAF records the actual IP address of the user in the HTTP header **X-Forwarded-For**. API Gateway resolves the actual IP address of the user based on the header.

Gateway Information Parameters VPC Endpoints

Parameter	Default Value	Value Range	Current Value	Updated	Operation
request_rate_limits	200 per second	1-1,000,000 per second	200 per second	--	Modify
request_body_size	12 MB	1-8,536 MB	12 MB	--	Modify
backend_timeout	60,000 ms	1-600,000 ms	60,000 ms	--	Modify
esp_token	Off	On/Off	Off	--	Modify
esp_basic	Off	On/Off	Off	--	Modify
esp_secret	Off	On/Off	Off	--	Modify
esp_route	Off	On/Off	Off	--	Modify
backend_client_certificate			Off	--	Modify
ssl_ciphers	ECDSA-ECDHE-AES256-GCM-SHA384:ECDSA-ECDHE-RSA-AE		ECDSA-ECDHE-AES256-GCM-SHA384:ECDSA-ECDHE-RSA-AE	--	Modify
real_ip_header	Off	On/Off	On	Nov 17, 2022 14:57:29 GMT+08:00	Modify
Parameter	Default Value	Value Range	Current Value	Updated	Operation
ssl_protocols	-1	Valid int32 value	1	Nov 17, 2022 14:57:29 GMT+08:00	Modify
vpc_name_modifiable	On	On/Off	On	Nov 2, 2022 19:57:50 GMT+08:00	Modify

----End

# 6 Request Throttling 2.0 with a Dedicated Gateway

---

## 6.1 Introduction

### Scenario

If the number of requests initiated from public networks for open APIs on APIG is not limited, the continuous increase in users will deteriorate the backend performance. And what's worse, the website or program will break down due to a large number of requests sent by malicious users. The traditional request throttling policies of APIG throttle requests by API, user, credential, and source IP address.

However, as users and their demands become more diversified, these traditional policies cannot meet the requirements for more refined rate limiting. To resolve this issue, APIG has launched request throttling 2.0, which is a type of plug-in policy. The 2.0 policies enable you to configure more refined throttling, for example, to throttle requests based on a certain request parameter or tenant.

This section describes how to create a request throttling 2.0 policy for rate limiting in different scenarios.

### Advantages

- A request throttling 2.0 policy limits the number of times that an API can be called within a specific time period. Basic, parameter-based, and excluded throttling is supported.
  - Basic throttling: Throttle requests by API, user, credential, or source IP address. This function is similar to a traditional request throttling policy but is incompatible with it.
  - Parameter-based throttling: Throttle requests based on headers, path parameter, method, query strings, or system parameters.
  - Excluded throttling: Throttle requests for specific tenants or credentials.
- API requests allowed in a time period can be limited by user or credential.

- Request throttling can be precise to the day, hour, minute, or second.

## Restrictions

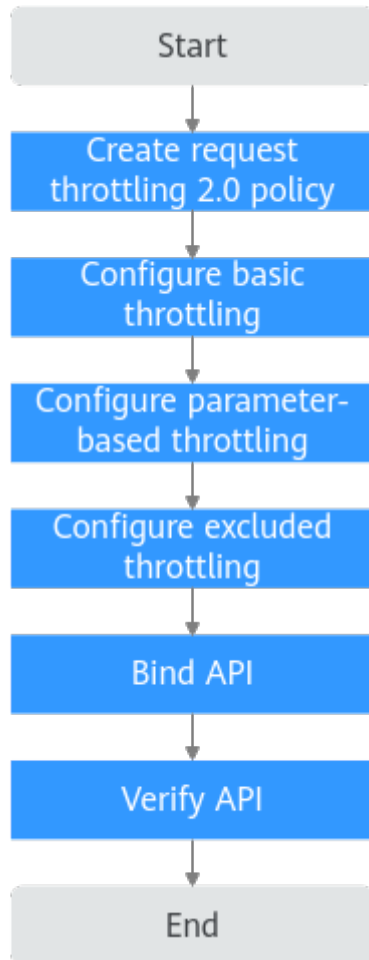
- Adding a request throttling 2.0 policy to an API means binding them together. An API can be bound with only one such policy in an environment, but each policy can be bound to multiple APIs. The APIs bound with request throttling 2.0 policies must have been published.
- For APIs not bound with a request throttling 2.0 policy, the throttling limit is the value of **ratelimit\_api\_limits** set on the **Parameters** page of the gateway.
- A traditional request throttling policy becomes invalid if a request throttling 2.0 policy is bound to the same API as the traditional one.
- You can define a maximum of 100 parameter-based throttling rules.
- The policy content cannot exceed 65,535 characters.
- If your gateway does not support request throttling 2.0, contact technical support.

## 6.2 General Procedure

Assume that you have the following request throttling requirements for an API:

1. The API can be called up to 10 times per 60s but can be called by a user only 5 times per 60s.
2. Only 10 requests containing header field **Host=www.abc.com** are allowed in 60s.
3. Only 10 requests with method **GET** and path **reqPath = /list** are allowed in 60s.
4. Only 10 requests with path **reqPath = /fc** are allowed in 60s.
5. Each excluded tenant can only call the API 5 times per 60s.

Following this procedure to create a request throttling 2.0 policy and bind it to an API.



1. **Create a policy.**  
Enter the basic information of the request throttling 2.0 policy.
2. **Configure basic throttling.**  
Configure the basic throttling settings.
3. **Configure parameter-based throttling.**  
Enable parameter-based throttling, and define parameters and rules.
4. **Configure excluded throttling.**  
Enable excluded throttling, and configure excluded tenants and credentials.
5. **Bind the policy to an API.**  
Bind the request throttling 2.0 policy to the API.
6. **Verify the API.**  
Call the API and verify whether the request throttling 2.0 policy has taken effect.

## 6.3 Implementation Procedure

### Step 1 Create a policy.

Log in to the APiG console and create a request throttling 2.0 policy. For details, see [Request Throttling 2.0](#) in the *API Gateway User Guide*.

In the navigation pane, choose **API Management > API Policies**. Click **Create Policy**, and select **Request Throttling 2.0**.

Configure basic policy information to meet your demands.

**Table 6-1** Policy basic Information

Parameter	Description
Name	Enter a name that conforms to specific rules to facilitate search.
Throttling	Select <b>High-performance</b> .
Policy Type	Select <b>API-specific</b> , which means measuring and throttling requests of a single API.
Period	Throttling period. Set this parameter to 60s.



**Step 2** Configure basic throttling.

As required in **1**, set **Max. API Requests** to 10 times per 60s and **Max. User Requests** to 5 times per 60s.

**Table 6-2** Basic throttling

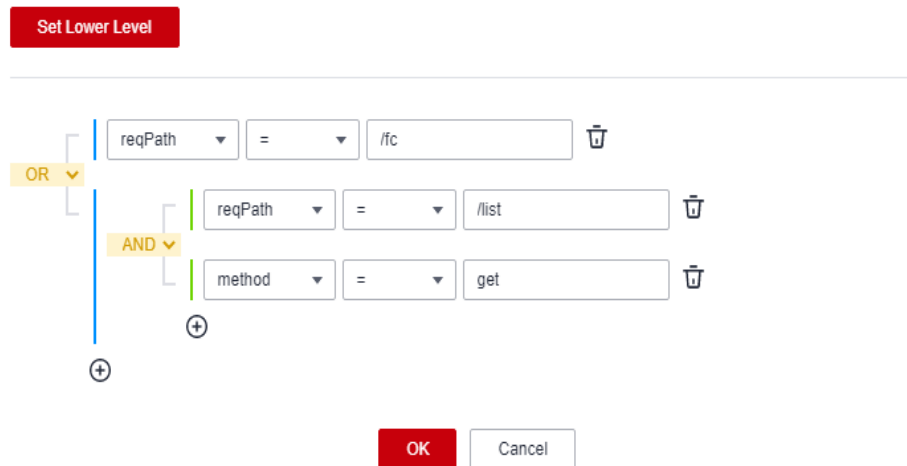
Parameter	Description
Max. API Requests	10
Max. User Requests	5

**Step 3** Configure parameter-based throttling.

1. As required in **2**, enable parameter-based throttling, and define the header and rule.
  - a. Click **Add Parameter**, select **header** for **Parameter Location**, and enter **Host** for **Parameter**.
  - b. In the **Rules** area, click **Add Rule**, and set **Max. API Requests** to **10** and **Period** to **60 seconds**. Then click , and set the matching condition **Host = www.abc.com**.
  - c. Click **OK**. The header matching rule **Host = www.abc.com** is generated, indicating that an API bound with this policy can only be called 10 times per 60s by requests whose **Host** header is **www.abc.com**.
2. As required in **3** and **4**, define multiple rules with parameter **Path**.
  - a. In the **Rules** area, click **Add Rule**, and set **Max. API Requests** to **10** and **Period** to **60 seconds**. Then click  to open the **Condition Expressions** dialog box.
  - b. Add these three condition expressions: **reqPath = /fc**, **reqPath = /list**, and **method = get**.
  - c. Click **Set Lower Level**.

- d. Put the two **reqPath** expressions in an **OR** relationship. This means the condition is met when either of the two paths is matched.
- e. Select **reqPath = /list** and **method = get**, click **Set Lower Level**, and select **AND**.

**Condition Expressions**



- f. Click **OK**. It indicates that APIs with path **/list** and method **GET** or APIs with path **/fc** bound with this policy can only be called 10 times per 60s.

**Step 4** Configure excluded throttling.

As required in 5, enable excluded throttling. Add an excluded tenant with a threshold of 5 requests per 60s.

**Table 6-3** Excluded throttling

Parameter	Description
Account ID	Tenant ID
Threshold	5

**Step 5** Click **OK**. The request throttling 2.0 policy is configured.

**Step 6** Bind this policy to an API.

1. Click the policy name to go to the policy details page.
2. In the **APIs** area, select environment **RELEASE** and click **Bind to APIs**. Select an API and click **OK**.

**Step 7** Verify the API.

Call the API and verify whether the request throttling 2.0 policy has taken effect.

----End



# 7 Two-Factor Authentication with a Dedicated Gateway

---

## 7.1 Introduction

### Scenario

APIG provides flexible authentication modes and allows you to configure a custom authorizer for two-factor authentication. This section describes how to create an API that uses two-factor authentication (app + custom).

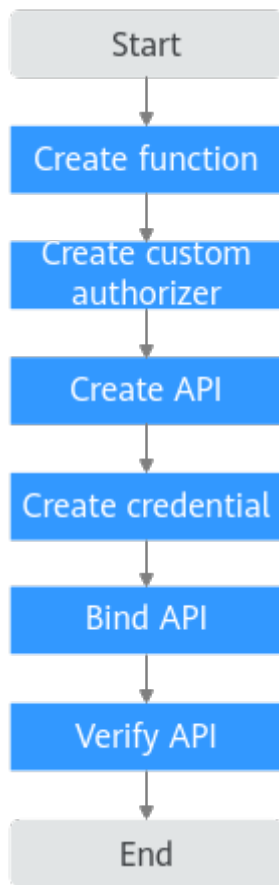
### Advantages

In addition to secure app authentication, you can use a custom authorizer to ensure API security.

### Restrictions

Custom authentication relies on FunctionGraph.

## 7.2 General Procedure



1. **Create a function.**  
The function will be used for custom authentication.
2. **Create a custom authorizer.**  
Set the authorizer type to **Frontend**, and select the function created in the previous step.
3. **Create an API.**  
Set authentication mode to **App**, enable **Two-Factor Authentication**, and select the custom authorizer created in the previous step.
4. **Create a credential.**  
APIs that use app authentication require a credential to call. Create a credential to generate an ID and key/secret pair.
5. **Bind the credential to the created API.**  
APIs that use app authentication can be called only with bound credentials.
6. **Verify the API.**  
Call the API to check whether two-factor authentication is configured successfully.

## 7.3 Implementation Procedure

**Step 1** Log in to the FunctionGraph console. On the **Dashboard** page, click **Create Function**. For details, see [Developing a Custom Authorizer with FunctionGraph](#).

1. Set the parameters according to the following table, and click **Create Function**.

**Table 7-1** Function configuration

Parameter	Description
Function Type	Default: <b>Event Function</b>
Region	Select the same region as that of APIG.
Function Name	Enter a name that conforms to specific rules to facilitate search.
Agency	An agency that delegates FunctionGraph to access other cloud services. For this example, select <b>Use no agency</b> .
Enterprise Project	The default option is <b>default</b> .
Runtime	Select <b>Python 3.9</b> .

2. On the **Configuration** tab, choose **Environment Variables** in the left pane, and click **Add**. **test** is a header for identity authentication, and **query** is for parameter query. If **token** involves sensitive data, enable the **Encrypted** option.



3. On the **Code** tab, copy the following code to **index.py**, and click **Deploy**. For details about coding, see [Creating a Function for Frontend Custom Authentication](#) in the *API Gateway Developer Guide*.

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    testParameter = context.getUserData('test');
    userToken = context.getUserData('token');
    if event["headers"].get("token") == userToken and event["queryStringParameters"].get("test") == testParameter:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status": "allow",
                "context": {
                    "user": "auth success"
                }
            })
        }
    else:
```

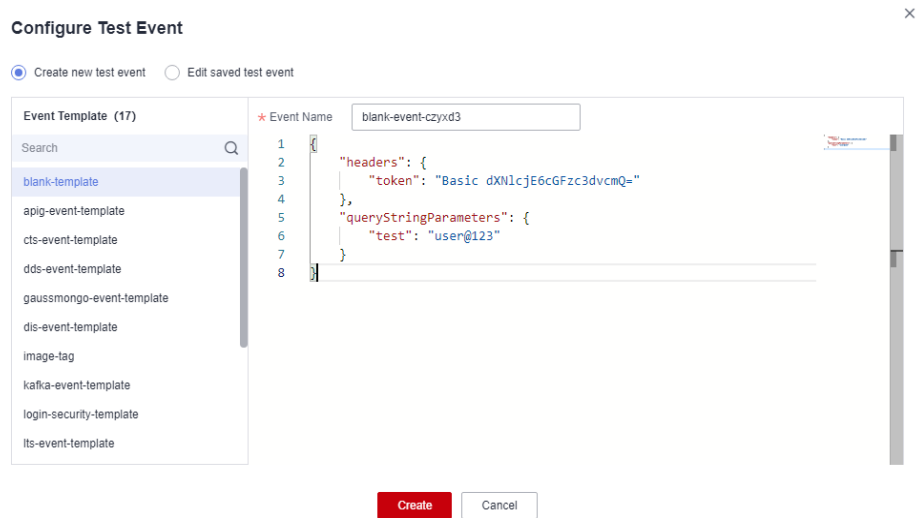
```

resp = {
    'statusCode': 401,
    'body': json.dumps({
        "status":"deny",
    })
}
return json.dumps(resp)
    
```

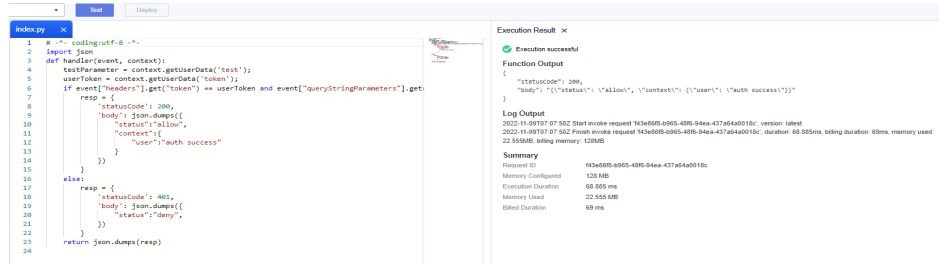
4. Configure a test event to debug the code.
  - a. Select **Configure Test Event** from the drop-down list and configure a test event.

**NOTE**

The parameter values in the test event must be the same as those of the environment variables.



- b. Click **Test**.



- c. Click **Deploy**.

**Step 2** Go to the APIG console, and choose **API Management > API Policies**.

On the **Custom Authorizers** tab, create a custom authorizer.

**Table 7-2** Custom authorizer configuration

Parameter	Description
Name	Enter a name that conforms to specific rules to facilitate search.
Type	Select <b>Frontend</b> .

Parameter	Description
Function URN	Click <b>Select</b> and select the <b>created function</b> .
Version/Alias	<b>Version</b> is selected by default.
Max. Cache Age (s)	30
Identity Sources	Set two identity sources: header <b>token</b> and query string <b>test</b> .

**Step 3** Choose **API Management > APIs**, and click **Create API**.

1. Configure the frontend information according to the following table.

**Table 7-3** Frontend configuration

Parameter	Description
API Name	Enter a name that conforms to specific rules to facilitate search.
Group	The default option is <b>DEFAULT</b> .
URL	<b>Method:</b> Request method of the API. Set this parameter to <b>GET</b> . <b>Protocol:</b> Request protocol of the API. Set this parameter to <b>HTTPS</b> . <b>Subdomain Name:</b> The system automatically allocates a subdomain name to each API group for internal testing. The subdomain name can be accessed 1000 times a day. <b>Path:</b> Path for requesting the API. Enter <b>/api/two_factor_authorization</b> .
Gateway Response	Select a response to be displayed if the gateway fails to process an API request. The default gateway response is <b>default</b> .
Authentication Mode	API authentication mode. Set this parameter to <b>App</b> .
Two-Factor Authentication	Enable this option and select a <b>custom authorizer</b> .

2. Click **Next** and set the backend type to **Mock**.  
Select a status code, set the response, and click **Finish**.
3. Publish the API.

**Step 4** In the navigation pane, choose **API Management > Credentials**.

Click **Create Credential**, enter a credential name, and click **OK**.

**Step 5** Bind this credential to the API.



# 8 HTTP-to-HTTPS Auto Redirection with a Dedicated Gateway

## 8.1 Introduction

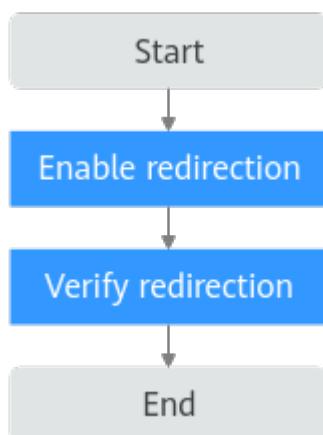
### Scenario

APIG supports HTTP-to-HTTPS redirection. HTTP APIs are insecure in transmission and authentication. You can upgrade them for access over HTTPS while ensuring HTTP compatibility. This function is supported by gateways created after November 30, 2022.

### Restrictions

Redirection is only suitable for GET and HEAD requests. Redirecting other requests may cause data loss due to browser restrictions.

## 8.2 General Procedure



1. **Enable redirection.**  
Ensure that the API for which you enable redirection uses HTTPS or HTTP&HTTPS for frontend requests.

## 2. [Verify redirection.](#)

Verify that the redirection function is working.

# 8.3 Implementation Procedure

## NOTE

### Prerequisites

- You have created an API whose frontend request protocol is set to **HTTPS** or **HTTP&HTTPS**.
- The API has been published.
- An independent domain name and SSL certificate have been bound to the API group to which the API belongs.

For details about these operations, see the [APIG User Guide](#)

### Enabling Redirection

**Step 1** Log in to the APIG console, and choose **API Management > API Groups**.

**Step 2** Click a group name to go to the details page.

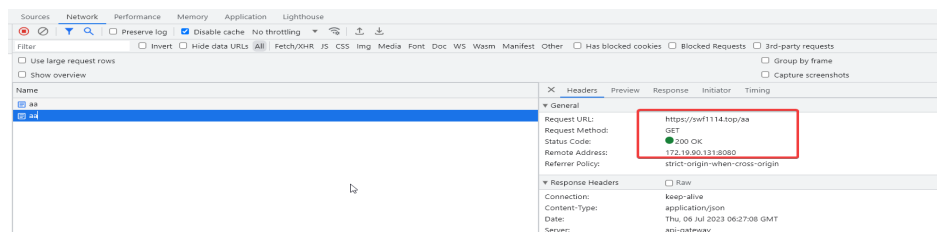
**Step 3** On the **Group Information** tab, locate the bound independent domain name, and enable **HTTP-to-HTTPS Auto Redirection**.

----End

### Verifying Redirection

**Step 1** Use a browser to call the API through HTTPS.

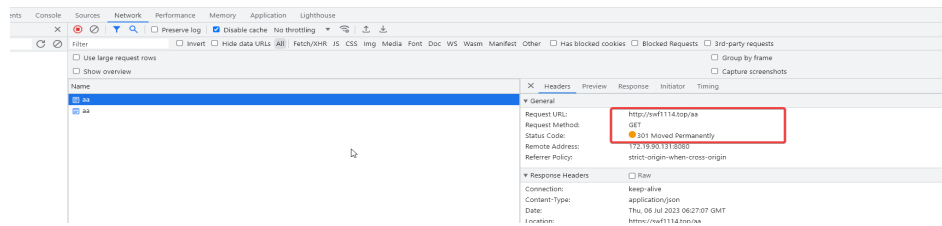
1. In the address bar of the browser, enter **https://API request path**, and press **Enter**.
2. Press **F12**.
3. Check the status code displayed on the **Network** tab. **200** means the calling is successful.



**Step 2** Use a browser to call the API through HTTP.

1. In the address bar of the browser, enter **http://API request path**, and press **Enter**.
2. Press **F12**.
3. Check the status code displayed on the **Network** tab. **301** means the redirection is successful.





----End

# 9 Routing gRPC Service Requests Using a Dedicated Gateway

---

## 9.1 Introduction

### Scenario

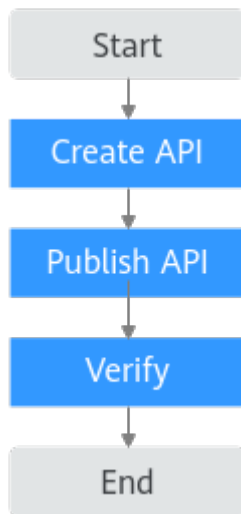
gRPC is a modern, open-source, high-performance Remote Procedure Call (RPC) framework that can run in any environment. You only need to define the request and response of each API, and let the gRPC framework take care of the rest. gRPC uses protocol buffers (protobuf) both as its Interface Definition Language (IDL) and for bottom-layer message exchange.

When you use a gRPC service, you can create an API in APIG to route requests for the service.

### Restrictions

Due to the constraints of the gRPCs protocol, gRPC APIs cannot be imported, exported, or debugged, and do not support third-party or custom authentication policies.

## 9.2 General Procedure



1. **Create an API.**  
Create a gRPC API with gRPCs as both the frontend and backend protocols.
2. **Publish the API.**  
Publish the gRPC API in an environment.
3. **Verify the API.**  
Use a gRPC client to test the gRPC service. If the server returns a response normally, the gRPC service is available.

## 9.3 Implementation Procedure

### Prerequisites

- Both the client and server are of the gRPC type.
- The server has a proto file that defines API request and response parameters. The proto file is used in gRPC to define data structures and service APIs. It describes data structures and interactions using protobuf and serves as a contract for communication between the client and server.

### Creating an API

- Step 1** Go to the APIG console.
- Step 2** Select a gateway at the top of the navigation pane.
- Step 3** In the navigation pane, choose **API Management > APIs**.
- Step 4** Click **Create API > Create gRPC API**. For details, see [Creating a gRPC API](#).
- Step 5** Configure frontend information and click **Next**.

**Table 9-1** Frontend configuration

Parameter	Description
API Name	Enter a name that conforms to specific rules to facilitate search.
Group	The default option is <b>DEFAULT</b> .
URL	<b>Method:</b> Request method of the API. Default: <b>POST</b> . <b>Protocol:</b> Request protocol of the API. Default: <b>GRPCS</b> . <b>Subdomain Name:</b> The system automatically allocates a subdomain name to each API group for internal testing. The subdomain name can be accessed 1000 times a day. <b>Path:</b> Path for requesting the API. In this example, enter <b>/helloworld.Greeter</b> . For details about the request path, see the <a href="#">proto file</a> . <b>helloworld</b> indicates the package name, and <b>Greeter</b> indicates the service name.
Gateway Response	Select a response to be displayed if the gateway fails to process an API request. The default gateway response is <b>default</b> .
Matching	Select <b>Prefix match</b> .
Authentication Mode	API authentication mode. Select <b>None</b> .

**Step 6** Configure the backend information and click **Next**.

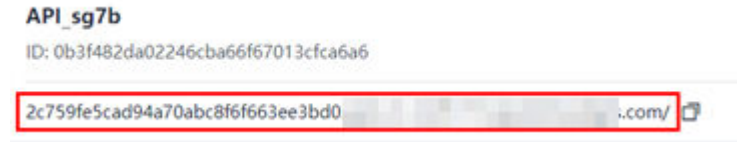
**Table 9-2** Backend configuration

Parameter	Description
Load Balance Channel	Determine whether the backend service will be accessed using a load balance channel. For this example, select <b>Skip</b> .
URL	<b>Method:</b> Request method of the API. Default: <b>POST</b> . <b>Protocol:</b> Request protocol of the backend service. Default: <b>GRPCS</b> . <b>Backend Address:</b> Address and port of the backend service. <b>Path:</b> Path of the backend service. In this example, enter <b>/</b> .

**Step 7** Click **Finish**.

**NOTE**

On the **APIs** page, the API URL only shows the domain name and path. It does not show the request method or protocol. When sending a gRPC request, enter the domain name.



----End

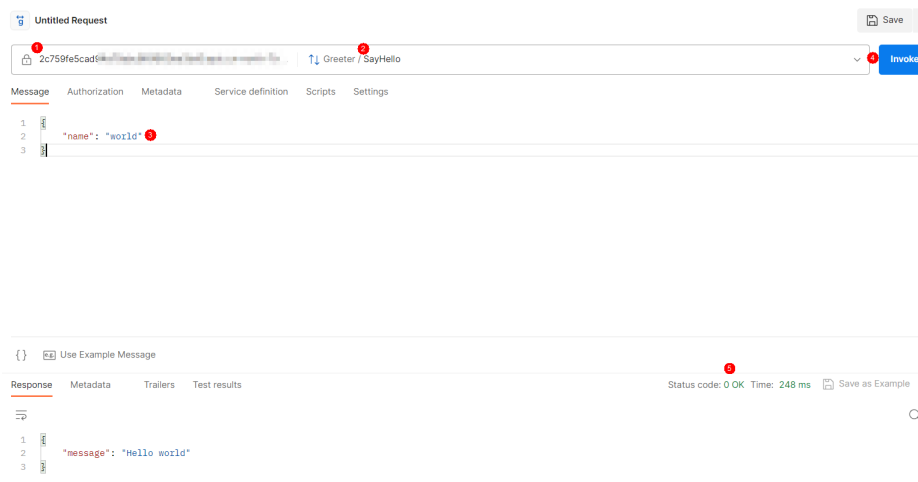
## Publishing the API

- Step 1** On the **APIs** tab, select the created API and click **Publish**.
- Step 2** Select the environment where the API will be published, and enter a description.
- Step 3** Click **OK**. After the API is published, the red exclamation mark (!) in the upper left corner of the **Publish** button disappears.

----End

## Verifying the API

Use the **API test** tool to invoke the created API, or invoke this API on a client.



**Step 1** Enter the debugging domain name of the API's group.

**Step 2** Import the proto file of the server.

The proto file contains the following content:

```
syntax = "proto3";
package helloworld;
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}
```

```
// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

- **helloworld**: package name
- **Greeter**: service name
- **SayHello**: method name
- **HelloRequest**: request body
- **HelloReply**: response body

**Step 3** Enter an API request in the **message** area according to the proto file.

```
{
  "name": "world"
}
```

**Step 4** Click **Invoke** to send the request.

**Step 5** View the response in the **Response** area. If status code **0 OK** is displayed, the invocation is successful.

----End

# 10 Client Authentication with a Dedicated Gateway

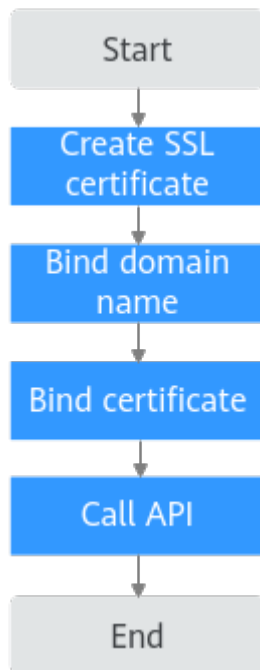
---

## 10.1 Solution

If the API frontend supports HTTPS, you need to add an SSL certificate for the independent domain name bound to the API group. An SSL certificate is used for data encryption and identity authentication. If an SSL certificate contains a **CA certificate**, client authentication (two-way authentication) is enabled by default. Or one-way authentication will be used.

- One-way authentication: When a client connects to a server, the client verifies the validity of the SSL certificate of the server.
- Two-way authentication: When a client connects to a server, both the client and server verify the validity of the SSL certificate.

## 10.2 General Procedure



Dedicated gateways support both one-way and two-way authentication. These two modes have the same procedure. The following will take one-way authentication as an example. For details about two-way authentication, see [Two-Way Authentication](#).

1. **Create an SSL certificate.**  
An SSL certificate is used for data encryption and identity authentication.
2. **Bind a domain name.**  
Bind the group to which the API belongs with a licensed and resolved independent domain name.
3. **Bind a certificate.**  
Bind the independent domain name to the created SSL certificate.
4. **Call the API.**  
Check whether the API call is successful.

## 10.3 Implementation Procedure

### One-Way Authentication

**Step 1** Go to the APIG console.

**Step 2** Select a gateway at the top of the navigation pane.

**Step 3** Create an SSL certificate.

1. In the navigation pane, choose **API Management > API Policies**.



2. On the **SSL Certificates** tab, click **Create SSL Certificate**.

**Table 10-1** Certificate configuration for one-way authentication

Parameter	Description
Name	Enter a certificate name that conforms to specific rules to facilitate search.
Instances Covered	Select <b>Current</b> .
Content	-----Start certificate----- MIICXglBAAKBgQC6ndRHy5Dv5TcZiVzT6qF iaMGy61ZlBurmBhUn61vMdvOHmtblST+fSl ZheNAcv2hQR4aqJLi4wrCerTaRyG9op3OSh... -----End certificate-----
Key	-----Start RSA private key----- MIICXglBAAKBgQC6ndRHy5Dv5TcZiVzT6qF iaMGy61ZlBurmBhUn61vMdvOHmtblST+fSl ZheNAcv2hQR4aqJLi4wrCerTaRyG9op3OSh... -----End RSA private key-----
CA	No CA certificate is required for one-way authentication.

3. Click **OK**.

**Step 4** Bind a domain name.

1. In the navigation pane, choose **API Management > API Groups**.
2. Click the name of the group to which the API belongs. The group details page is displayed.
3. On the **Group Information** tab page, click **Bind Independent Domain Name**.

**Table 10-2** Independent domain name configuration

Parameter	Description
Domain Name	Enter a licensed domain name.
Minimum TLS Version	Select <b>TLS1.2</b> .
HTTP-to-HTTPS Auto Redirection	Disabled by default.

4. Click **OK**.

**Step 5** Bind a certificate.

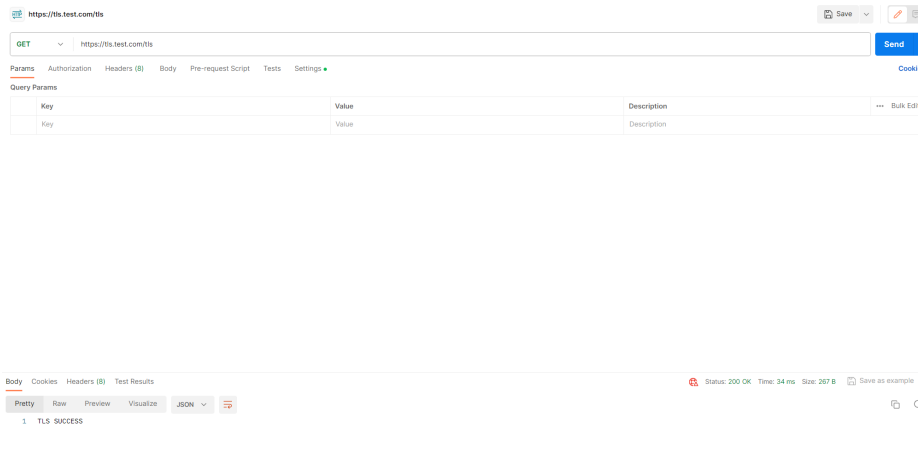
1. In the row that contains the domain name, click **Select SSL Certificate**.
2. Select the created certificate and click **OK**.

**NOTICE**

Client authentication should be disabled for one-way authentication.

**Step 6** Call the API.

Use the API test tool to call the API. If the status code is **200**, the API is successfully called.



----End

## Two-Way Authentication

**Step 1** On the **SSL Certificates** tab, click **Create SSL Certificate**.

**Table 10-3** Certificate configuration for two-way authentication

Parameter	Description
Name	Enter a certificate name that conforms to specific rules to facilitate search.
Instances Covered	Select <b>Current</b> .
Content	Enter the certificate content. -----Start certificate----- MIICXgIBAAKBgQC6ndRHy5Dv5TcZiVzT6qF iaMGy61ZibUrmBhUn61vMdvOHmtbIST+fSl ZheNAcv2hQR4aqJLi4wrcerTaRyG9op3OSh... -----End certificate-----
Key	Enter the key. -----Start RSA private key----- MIICXgIBAAKBgQC6ndRHy5Dv5TcZiVzT6qF iaMGy61ZibUrmBhUn61vMdvOHmtbIST+fSl ZheNAcv2hQR4aqJLi4wrcerTaRyG9op3OSh... -----End RSA private key-----

Parameter	Description
CA	<p>Enter the CA certificate content. <b>After the CA certificate is configured, client authentication is enabled by default as long as the independent domain name is bound to the SSL certificate.</b></p> <p>-----Start certificate-----                      MIICXglBAAKBgQC6ndRHy5Dv5TcZiVzT6qF                      iaMGy61ZlbUrmBhUn61vMdvOHmtblST+fSl                      ZheNAcv2hQR4aqJLi4wrcerTaRyG9op3OSh...                      -----End certificate-----</p>

**Step 2** Click **OK**.

**Step 3** Bind a domain name.

1. In the navigation pane, choose **API Management > API Groups**.
2. Click the name of the group to which the API belongs. The group details page is displayed.
3. On the **Group Information** tab page, click **Bind Independent Domain Name**.

**Table 10-4** Independent domain name configuration

Parameter	Description
Domain Name	Enter a licensed domain name.
Minimum TLS Version	Select <b>TLS1.2</b> .
HTTP-to-HTTPS Auto Redirection	Disabled by default.

4. Click **OK**.

**Step 4** Bind a certificate.

1. In the row that contains the domain name, click **Select SSL Certificate**.
2. Select the created certificate and click **OK**.

**NOTICE**

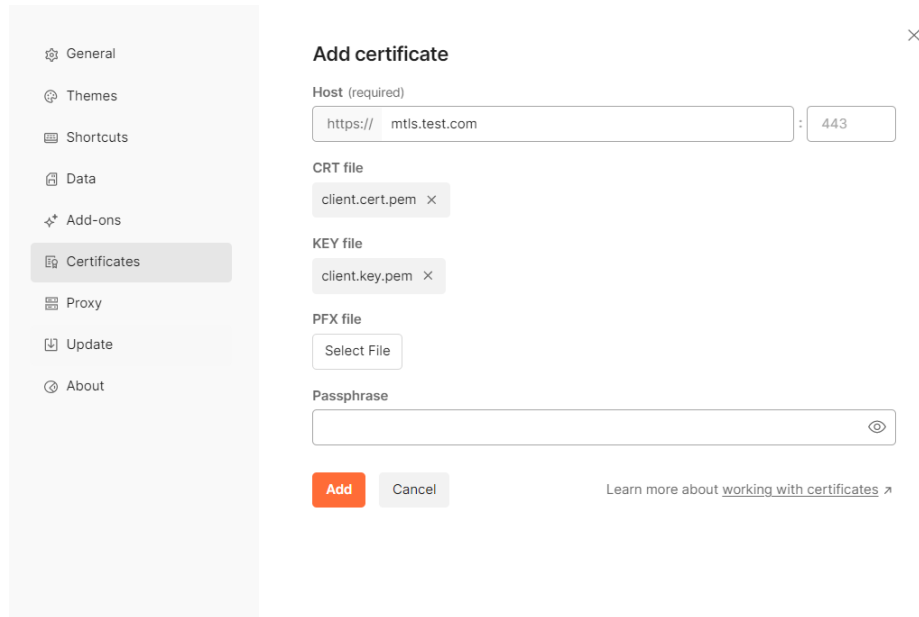
After binding an SSL certificate, two-way authentication is automatically enabled.

**Step 5** Call the API.

Use the API test tool to call the API. If the status code is **200**, the API is successfully called.

You need to configure the client certificate when accessing APIs.

If Postman is used to call APIs, you need to add client certificates to **Certificates** in **Setting** and upload the client certificates and key.



----End