



API Gateway

Best Practices

Date 2024-10-09

Contents

1	Selectively Exposing CCE Workloads with a Dedicated Gateway.....	1
1.1	Introduction.....	1
1.2	Resource Planning.....	2
1.3	General Procedure.....	3
1.4	Implementation Procedure.....	3
2	Selectively Exposing Service Capabilities of a Data Center Using a Dedicated Gateway.....	14
3	Developing a Custom Authorizer with FunctionGraph.....	16
4	Exposing Backend Services Across VPCs Using a Dedicated Gateway.....	20
4.1	Introduction.....	20
4.2	Resource Planning.....	21
4.3	General Procedure.....	22
4.4	Implementation Procedure.....	23
5	Interconnecting a Dedicated Gateway with WAF.....	31
6	Request Throttling 2.0 with a Dedicated Gateway.....	35
6.1	Introduction.....	35
6.2	General Procedure.....	36
6.3	Implementation Procedure.....	37
7	Two-Factor Authentication with a Dedicated Gateway.....	40
7.1	Introduction.....	40
7.2	General Procedure.....	41
7.3	Implementation Procedure.....	42
8	HTTP-to-HTTPS Auto Redirection with a Dedicated Gateway.....	46
8.1	Introduction.....	46
8.2	General Procedure.....	46
8.3	Implementation Procedure.....	47
9	Client Authentication with a Dedicated Gateway.....	49
9.1	Solution.....	49
9.2	General Procedure.....	50
9.3	Implementation Procedure.....	50

10 Invoking an API Policy Backend via Frontend Authentication Parameters (System Parameter)	55
10.1 Introduction.....	55
10.2 General Procedure.....	56
10.3 Implementation Procedure.....	56
11 Forwarding WebSocket Services Using Dedicated Gateway	60
12 Change History	63

1 Selectively Exposing CCE Workloads with a Dedicated Gateway

1.1 Introduction

Scenario

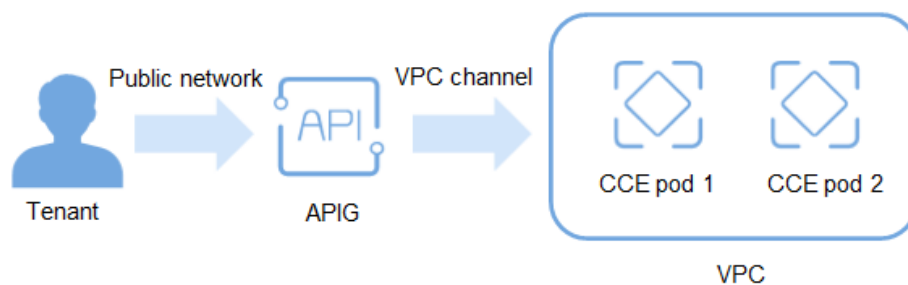
You can use APIG to selectively expose your workloads and microservices in Cloud Container Engine (CCE).

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

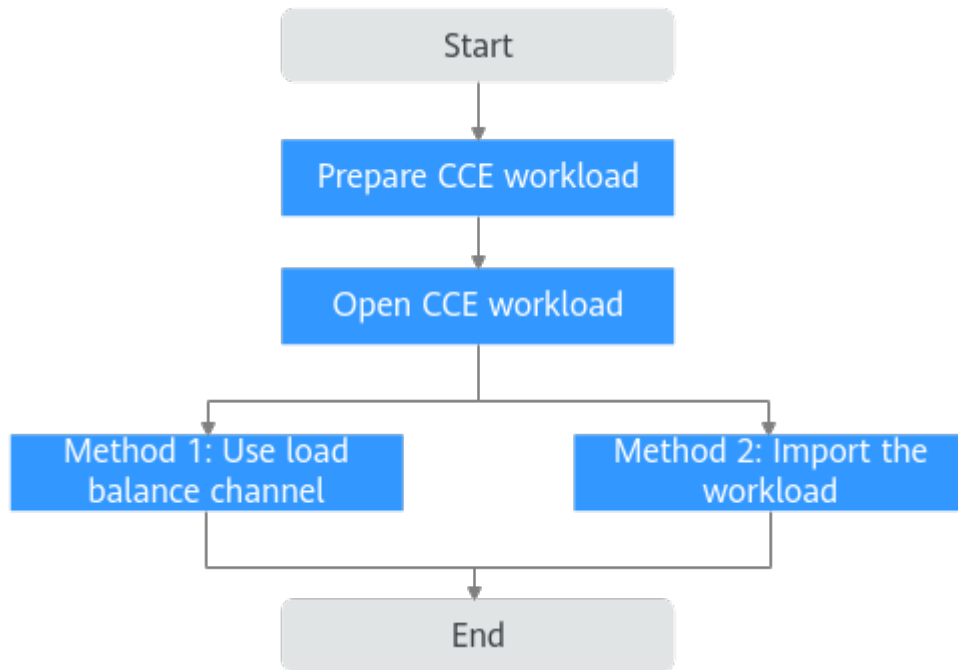
- Only CCE Turbo clusters and CCE clusters using the VPC network model are supported.
- The CCE cluster and your gateway must be in the same VPC or connected.
- If you select a CCE cluster that uses the VPC network model, add the container CIDR block of the cluster in the **Routes** area of the gateway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

1. Log in to the CCE console and buy a CCE standard or CCE Turbo cluster on the **Clusters** page. Select **CCE Standard Cluster** and set **Network Model** to **VPC network**. For details, see section "Buying a CCE Standard/Turbo Cluster" in the *CCE User Guide*.
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**. Select **Deployment**. For details, see the section "Creating a Deployment" in 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 section "Configuring Labels and Annotations" in the *CCE User Guide*.

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 VPC_demo .
Port	Container port of a workload for opening services. Set this parameter to 80 , which is the default HTTP port.

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

- b. Configure microservice information.

Table 1-3 Microservice configuration

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

- c. Configure a server group.

Table 1-4 Server group configuration

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

- d. Configure health check.

Table 1-5 Health check configuration

Parameter	Description
Protocol	Default: TCP .

Parameter	Description
Check Port	Backend server port in the channel.
Healthy threshold	Default: 2 . This is the number of consecutive successful checks required for a cloud server to be considered healthy.
Unhealthy Threshold	Default: 5 . This is the number of consecutive failed checks required for a cloud server to be considered unhealthy.
Timeout (s)	Default: 5 . This is the timeout used to determine whether a health check has failed.
Interval (s)	Default: 10 . 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, choose **Create API > Create API**.
 - 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 preceding API group .
URL	<p>Method: Request method of the API. Set this parameter to ANY.</p> <p>Protocol: Request protocol of the API. Set this parameter to HTTPS.</p> <p>Subdomain Name: 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>Path: 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: default .
Matching	Select Prefix match .
Authentication Mode	API authentication mode. Select None . (None : 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 Configure .
URL	<p>Method: Request method of the API. Set this parameter to ANY.</p> <p>Protocol: Set this parameter to HTTP.</p> <p>Load Balance Channel: Select the created channel.</p> <p>Path: Path of the backend service.</p>

- 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 fault by following the instructions provided in "Published API Calling" > "Error Codes" in the *API Gateway User Guide*.
 4. Publish the API.
On the **APIs** tab, click **Publish Latest Version**, 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 about how to bind a domain name, see "Binding a Domain Name" in the *API Gateway User Guide*.

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

- b. Configure API information.

Table 1-9 API information

Parameter	Description
Protocol	API request protocol. HTTPS is selected by default.
Request Path	API request path for prefix match. Default: /. In this example, retain the default value.
Port	Enter 80 . This must be the same as the container port in the workload.
Authentication Mode	Default: None . (None : Not recommended for actual services. All users will be granted access to the API.)
CORS	Disabled by default.
Timeout (ms)	Backend timeout. Default: 5000 .

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. Otherwise, rectify the fault by following the instructions provided in "Published API Calling" > "Error Codes" in the *API Gateway User Guide*.
2. Publish the API.

On the **APIs** tab, click **Publish Latest Version**, 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.

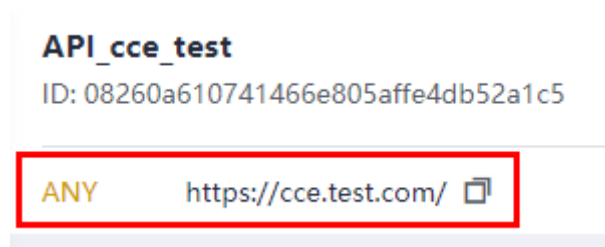
On the group details page, click the **Group Information** tab. The debugging domain name is only used for development and testing and can be accessed 1000 times a day. Bind independent domain names to expose APIs in the group.

Click **Bind Independent Domain Name** to bind registered public domain names. For details about how to bind a domain name, see "Binding a Domain Name" in the *API Gateway User Guide*.

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



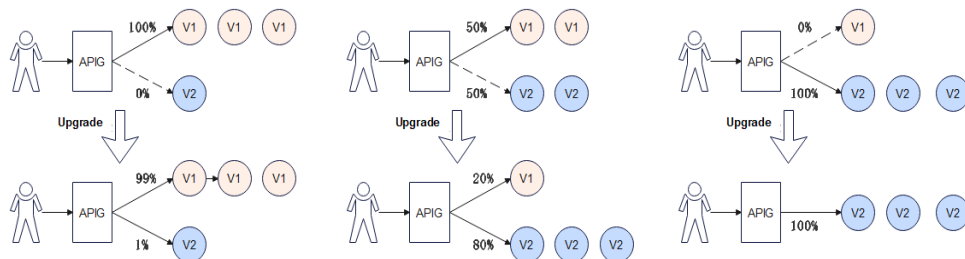
Now, the CCE workload has been opened by importing it.

----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.

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

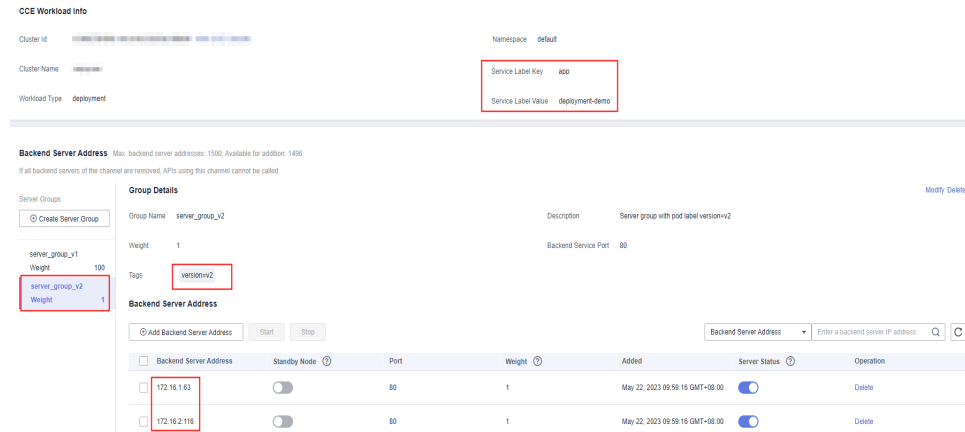
Parameter	Description
Server Group Name	Enter server_group_v2 .
Weight	Enter 1 .
Backend Service Port	Enter 80 .
Tag	Select pod label version=v2 .

2. Click **OK**.

Step 4 Refresh the backend server addresses.

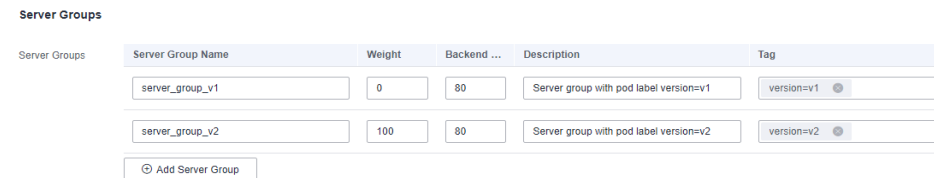
Refresh the page for the backend server addresses. The load balance channel automatically monitors the pod IP addresses of the workload and dynamically adds the addresses as backend server addresses. As shown in the following figure, tags **app=deployment-demo** and **version=v2** automatically match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

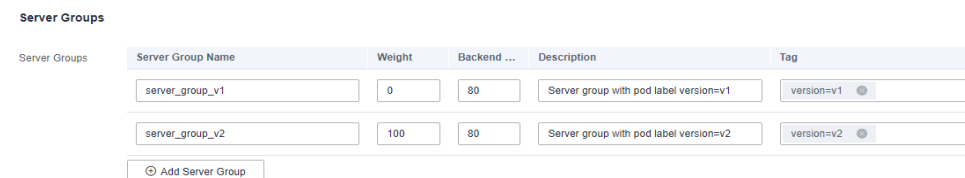


Step 5 Check that the new features released to V2 through grayscale release are running stably.

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.

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

Connecting a Data Center to APIG

Step 1 Create a VPC.

For details, see the section "Creating a VPC" in 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.

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 Create a dedicated API gateway.

For details, see section "Creating a Gateway" in the *API Gateway User Guide*.

Step 3 Enable Direct Connect by referring to the *Direct Connect User Guide*.

1. Create a connection.

Apply for a connection from your account manager. If you do not have an account manager, contact technical support.

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**.

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 section "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

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 section "Creating a Function for Frontend Custom Authentication" in the *API Gateway Developer Guide*.

Specify the runtime as Python 3.6.

Table 3-1 Function configuration

Parameter	Description
Region	Select the same region as that of APIG.
Function Type	Default: Event Function
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 Use no agency .
Enterprise Project	Enterprise projects group and manage resources across regions. Resources in enterprise projects are logically isolated. Select default .

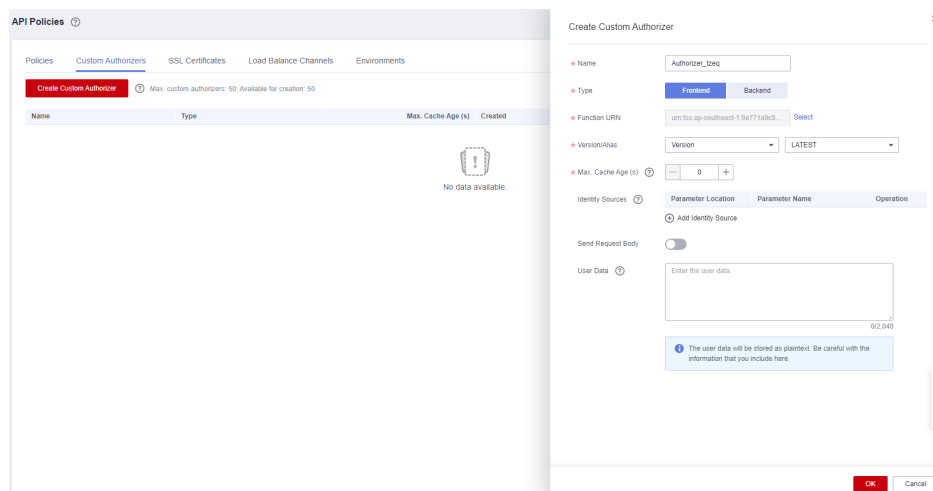
Parameter	Description
Runtime	Select Python 3.6 .

After the function is created, go to the function details page. 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 section "Creating an API" in the *API Gateway User Guide*. Set the authentication mode to **Custom**, and select the custom authorizer created in the preceding section. After modifying the API, publish it.

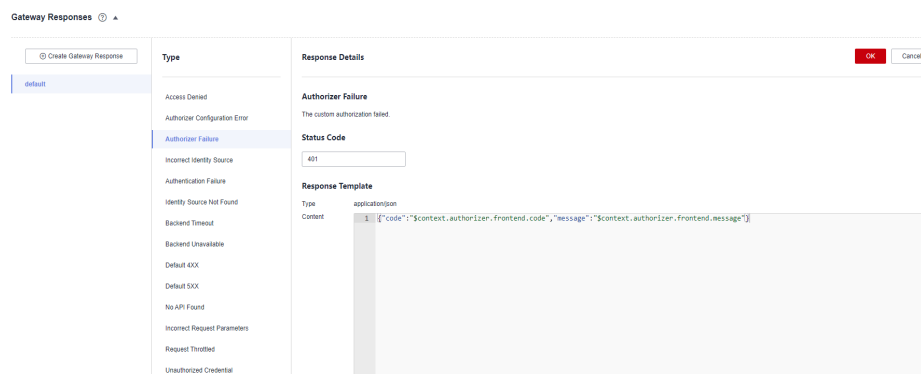
Setting the Error Response

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}"}
```



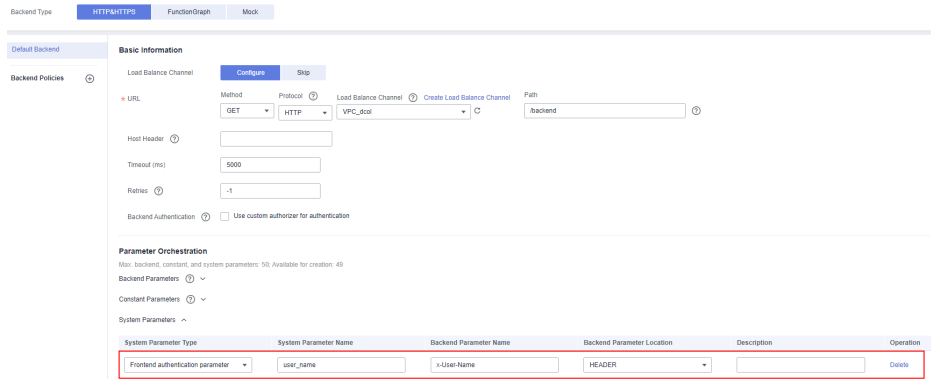
After the modification, if incorrect authentication is transferred when calling the API, the status code **401** is returned and the response result is as follows:

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

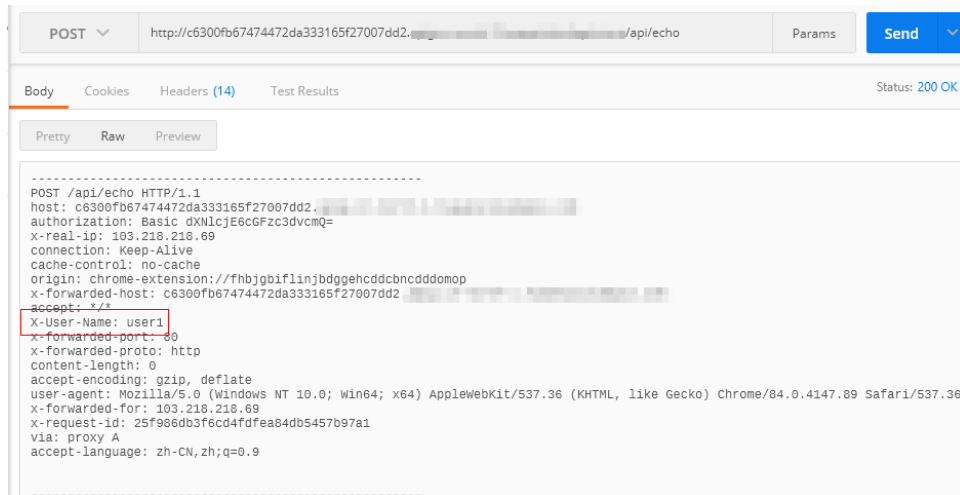
Mapping Frontend Authentication Parameters to Backend Parameters

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

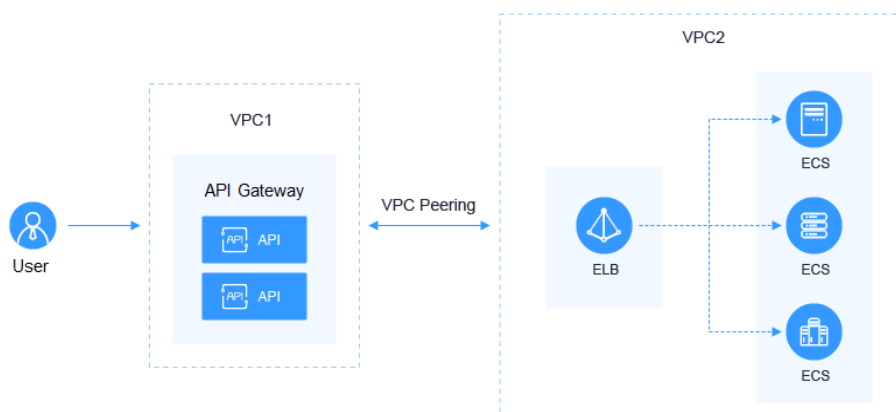
4.1 Introduction

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

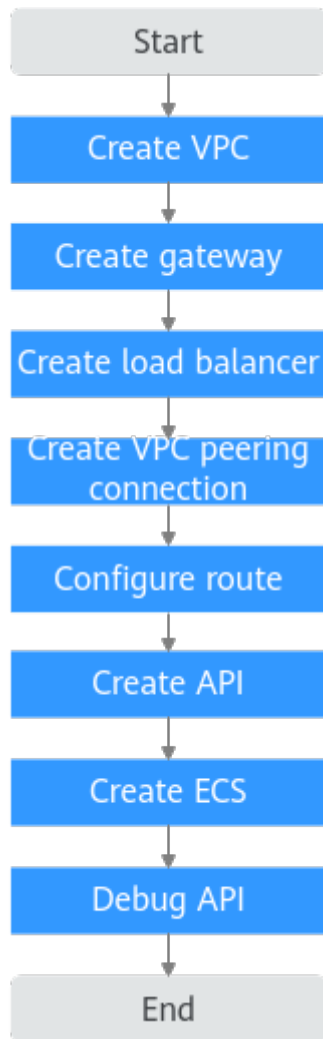
VPC 1, VPC 2, and the VPC CIDR block of your gateway cannot overlap. For details about the VPC CIDR block planning of the gateway, see [Table 4-3](#).

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 created 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: eu-de

Name: vpc1

IPv4 CIDR Block: 192.168.0.0/16

Enterprise Project: default

Default Subnet

Name: subnet-242b

IPv4 CIDR Block: 192.168.0.0/24

IPv6 CIDR Block: Enable

Associated Route Table: Default

[Add Subnet](#)

Table 4-2 Configuration information

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

Parameter	Description
Enterprise Project	Select default .
AZ	The AZ to which the subnet belongs. Select AZ1 .
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
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 AZ1 .
Gateway Name	Enter a name that conforms to specific rules to facilitate search.
Edition	Select Professional . 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 22:00:00---02:00:00 .

Parameter	Description
Enterprise Project	Select the enterprise project to which the gateway belongs. For this example, retain the default value default .
Network	Select VPC 1 and a subnet.
Security Group	Click Manage Security Groups and create a security group. Ensure that you have selected default for Enterprise Project .
Description	Description of the gateway.

Step 4 Click **Next**.

Step 5 If the gateway configurations are correct, create the instance.

----End

Creating 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 **Create Elastic Load Balancer**.

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

Table 4-5 Load balancer parameters

Parameter	Description
Type	Type of the load balancer.
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 AZ1 .
Name	Enter a load balancer name that conforms to specific rules to facilitate search.
Enterprise Project	Select default .
Specification	Select Fixed – Application load balancing (HTTP/HTTPS) and Network load balancing (TCP/UDP) by default.
Network Type	Select Private IPv4 network by default.
VPC	Select VPC 2 .
Frontend Subnet	Select a subnet.

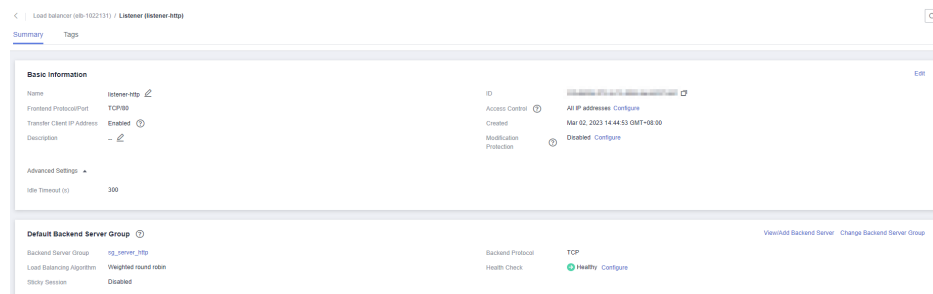
Step 5 Click **Create Now**.

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 Viewing the basic information and backend server group of the listener



----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
VPC Peering Connection Name	Enter a VPC peering connection name that conforms to specific rules to facilitate search.
Local VPC	Select VPC 1 .
Account	By default, My account is selected.
Peer Project	Select a project
Peer VPC	Select VPC 2 .

Step 3 Click **OK**.

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

Step 5 On the **Associated Routes** area, click **Add Route**.

1. In the displayed dialog box, enter the route information.

Table 4-7 Local and peer routing information

Parameter	Description
Local routes	
VPC	Select VPC 1 .
Route Table	VPC 1 route table
Destination	Enter the service address displayed on the details page of the load balancer .
Peer routes	
VPC	Select VPC 2 .
Route Table	VPC 2 route table
Destination	Enter the private outbound address displayed on the details page of the dedicated gateway .

2. Click **OK**.

----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 created is located.

Step 5 Click **Save**.

----End

Creating an API

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

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

Table 4-8 Frontend configuration

Parameter	Description
API Name	Enter a name that conforms to specific rules to facilitate search.
Group	The default option is DEFAULT .
URL	<p>Method: Request method of the API. Set this parameter to GET.</p> <p>Protocol: Request protocol of the API. Set this parameter to HTTPS.</p> <p>Subdomain Name: 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>Path: Path for requesting the API.</p>
Gateway Response	<p>Select a response to be displayed if the gateway fails to process an API request.</p> <p>The default gateway response is default.</p>
Authentication Mode	API authentication mode. Select None . (None: Not recommended for actual services. All users will be granted access to the API.)

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

Table 4-9 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 Skip .
URL	<p>Method: Request method of the API. Set this parameter to GET.</p> <p>Protocol: Set this parameter to HTTP.</p> <p>Backend Address: Enter the service address of the load balancer you created.</p> <p>Path: Path of the backend service.</p>

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

----End

Creating an ECS

Step 1 Log in to the cloud server console.

Step 2 Click **Create ECS**.

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

Table 4-10 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 Pay-per-use .
AZ	Select the AZ where the ECS is located.
CPU Architecture	The default option is x86 .
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 4-11 Network settings

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

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

Table 4-12 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 Password .
Username	The default user is root .
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 **Apply Now**.

----End

Debugging the API

Step 1 In the **load balancer** listener details, click **View/Add Backend Server**.

Step 2 On the **Backend Servers** page, add the **created ECS**.

Step 3 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 4 Enter the request parameters and click **Debug**.

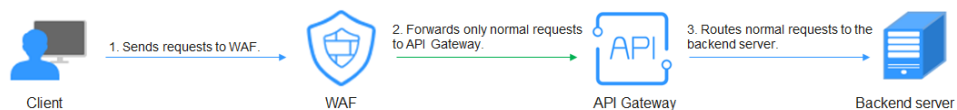
If the status code is **200**, the debugging is successful. Otherwise, rectify the fault by following the instructions provided in "Published API Calling" > "Error Codes" in the *API Gateway User Guide*.

----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



(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.

1. Log in to the APIG console, and choose **API Management > API Groups**.
2. Click **Create API Group > Create Directly** and enter the group name.
3. Click the name of the created group. The group details page is displayed.
4. On the **Group Information** tab, view and record the debugging domain name. It is unique and cannot be changed. It can be accessed up to 1,000 times a day.
5. On the APIs tab, click **Create API > Create API** to add an API.

Step 2 Add a domain name to WAF. Log in to the WAF console and choose **Website Settings > Add Website**. When configuring the **Server Address**, you need to enter the domain name of the API group and add a certificate. After adding a domain name, you need to whitelist the back-to-origin IP addresses, test WAF locally, and modify the DNS resolution settings of the domain name. For details, see section "Connection Process (Cloud Mode)" in the *Web Application Firewall User Guide*.

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.

Basic Settings

Protected Domain Name ?
 [Quick Add Domain Names Hosted on Cloud](#)
Only domain names that have been registered with ICP licenses can be added to WAF. View details at <https://beian.xinnet.com/>

Website Name (Optional)

Website Remarks (Optional)

Protected Port
 [View Ports You Can Use](#)
Standard ports 80 and 443 are the default ports reserved for HTTP and HTTPS protocols, respectively.

Server Configuration ?

Client Protocol	Server Protocol	Server Address	Server Port	Operation
HTTPS	HTTPS	IPv4 <input type="text" value=""/>	443	Delete

[Add Address](#) Origin server addresses you can add: 49

International

Certificate [Import New Certificate](#)

Proxy Your Website Uses ?

Layer-7 proxy
 Layer-4 proxy
 No proxy

- Step 3** On the gateway details page, bind the domain name to the API group.
- Go to the APIG console, and choose **API Management > API Groups**.
 - Click the name of a created group.
 - On the **Group Information** tab, click **Bind Independent Domain Name**.
 - In the dialog box that is displayed, add the domain name.

- Step 4** Enable **real_ip_from_xff** and set the parameter value to 1.
- In the navigation pane of the APIG console, choose **Gateways**.
 - On the **Parameters** tab, configure **real_ip_from_xff**.

Parameter	Default Value	Value Range	Current Value	Updated	Operation
real_ip_from_xff	Off	On/Off	On	Nov 17, 2022 14:57:29 GMT+08:00	Modify
real_index	-1	Valid int32 value	1	Nov 17, 2022 14:57:29 GMT+08:00	Modify

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.

----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.

1. Log in to the APIG console, and choose **Gateways** in the navigation pane.
2. Click the gateway name or **Access Console**.
3. On the **Basic Information** tab, view the inbound access.
 - **VPC Access Address:** VPC private access address
 - **EIP:** public network access address

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

1. Go to the APIG console, and choose **API Management > API Groups**.
2. Click the name of the **DEFAULT** group.
3. Click **Create API > Create API** to add an API.

Step 3 Add a domain name to WAF. Log in to the WAF console and choose **Website Settings > Add Website**. Set the **Server Address** to the **inbound access** of the gateway and add a certificate. After adding a domain name, you need to **whitelist the back-to-origin IP addresses, test WAF locally, and modify the DNS resolution settings of the domain name**. For details, see section "Connection Process (Cloud Mode)" in the *Web Application Firewall User Guide*.

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.

Basic Settings

Protected Domain Name ?
 [Quick Add Domain Names Hosted on Cloud](#)
Only domain names that have been registered with ICP licenses can be added to WAF. View details at <https://beian.xinnet.com/>

Website Name (Optional)

Website Remarks (Optional)

Protected Port
 [View Ports You Can Use](#)
Standard ports 80 and 443 are the default ports reserved for HTTP and HTTPS protocols, respectively.

Server Configuration ?

Client Protocol	Server Protocol	Server Address	Server Port	Operation
HTTPS	HTTPS	IPv4 <input type="text" value=":::"/>	443	Delete

[Add Address](#) Origin server addresses you can add: 49

International

Certificate [Import New Certificate](#)

Proxy Your Website Uses ?

Layer-7 proxy
 Layer-4 proxy
 No proxy

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

1. Go to the APIG console, and choose **API Management > API Groups**.
2. Click the name of the **DEFAULT** group.
3. On the **Group Information** tab, click **Bind Independent Domain Name**.
4. In the dialog box that is displayed, add the domain name.

Step 5 Enable **real_ip_from_xff** and set the parameter value to **1**.

1. In the navigation pane of the APIG console, choose **Gateways**.
2. On the **Parameters** tab, configure **real_ip_from_xff**.

Parameter	Default Value	Value Range	Current Value	Updated	Operation
real_ip_from_xff	off	on/off	on	Nov 17, 2022 14:57:29 GMT+08:00	Modify
xff_index	-1	Valid int32 value	1	Nov 17, 2022 14:57:29 GMT+08:00	Modify

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**. APIG resolves the actual IP address of the user based on the header.

----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

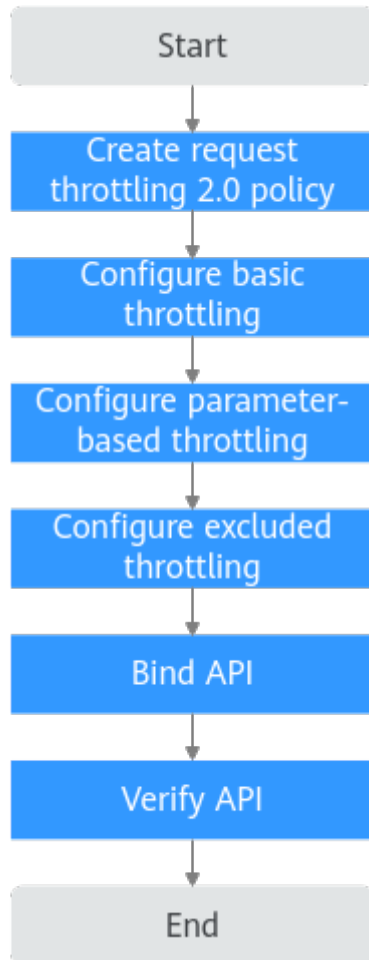
- Adding a request throttling 2.0 policy to an API means binding them together. An API can be bound with only one such policy in an environment, but each policy can be bound to multiple APIs. The APIs bound with request throttling 2.0 policies must have been published.
- For APIs not bound with a request 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 section "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 High-performance .
Policy Type	Select API-specific , 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).

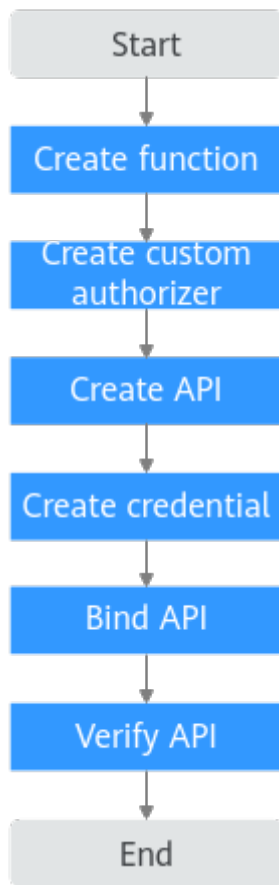
Advantages

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
Project	Projects group and isolate resources (including compute, storage, and network resources) across physical regions. A default project is provided for each region, and subprojects can be created under each default project. Users can be granted permissions to access all resources in a specific project. The selected region is used by default.
Region	Select the same region as that of APIG.
Function Type	Default: Event Function
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 Use no agency .
Enterprise Project	Enterprise projects group and manage resources across regions. Resources in enterprise projects are logically isolated. Select default .
Runtime	Select Python 3.9 .

2. After the function is created, go to the function details page. On the **Configuration** tab, choose **Environment Variables** in the left pane, and click **Edit Environment Variable > Add**. **test** is a header for identity authentication, and **query** is for parameter query. If **token** involves sensitive data, enable the **Encrypted** option.

Edit Environment Variable

Key	Value	Encrypted	
<input type="text" value="test"/>	<input type="text" value="user@123"/>	<input type="checkbox"/>	Delete
<input type="text" value="token"/>	<input type="text" value="*****"/>	<input checked="" type="checkbox"/>	Delete

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

```
# -*- coding:utf-8 -*-
import json
```

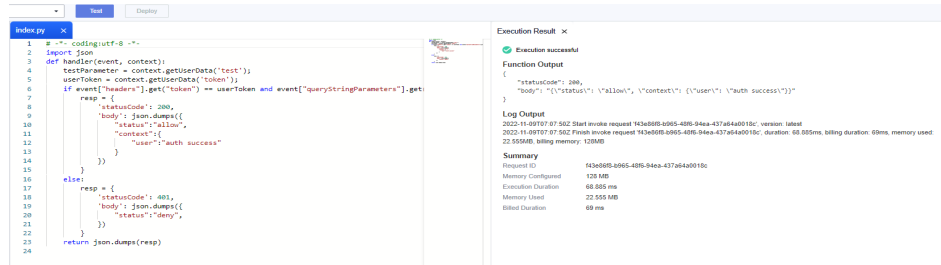
```
def handler(event, context):
    testParameter = context.getUserData('test');
    userToken = context.getUserData('token');
    if event["headers"].get("token") == userToken and event["queryStringParameters"].get("test") == testParameter:
        resp = {
            'statusCode': 200,
            '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 Frontend .
Function URN	Click Select and select the created function .
Version/Alias	Version is selected by default.

Parameter	Description
Max. Cache Age (s)	30
Identity Sources	Set two identity sources: header token and query string test .

Step 3 Choose **API Management > APIs**, and click **Create API > 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 DEFAULT .
URL	Method: Request method of the API. Set this parameter to GET . Protocol: Request protocol of the API. Set this parameter to HTTPS . Subdomain Name: The system automatically allocates a subdomain name to each API group for internal testing. The subdomain name can be accessed 1000 times a day. Path: Path for requesting the API. Enter /api/two_factor_authorization .
Gateway Response	Select a response to be displayed if the gateway fails to process an API request. The default gateway response is default .
Authentication Mode	API authentication mode. Set this parameter to App .
Two-Factor Authentication	Enable this option and select a custom authorizer .

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**.

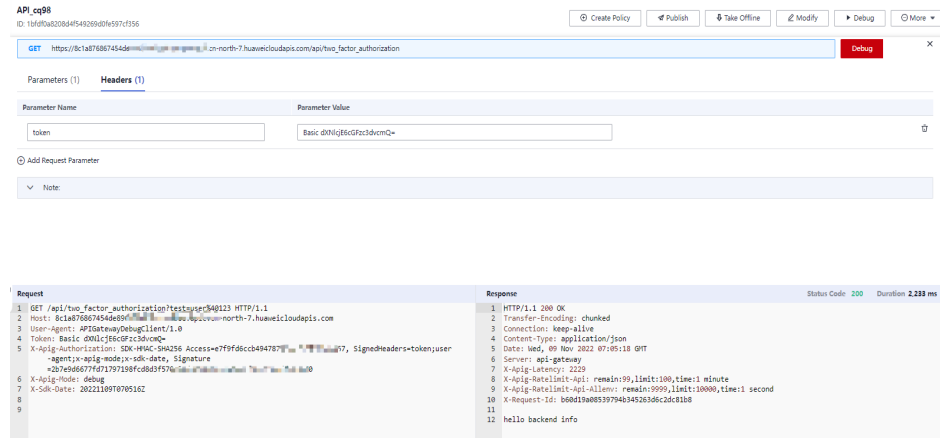
Step 5 Bind this credential to the API.

Click the credential name to go to the details page. In the **APIs** area, click **Bind to APIs**, select an API, and click **OK**.

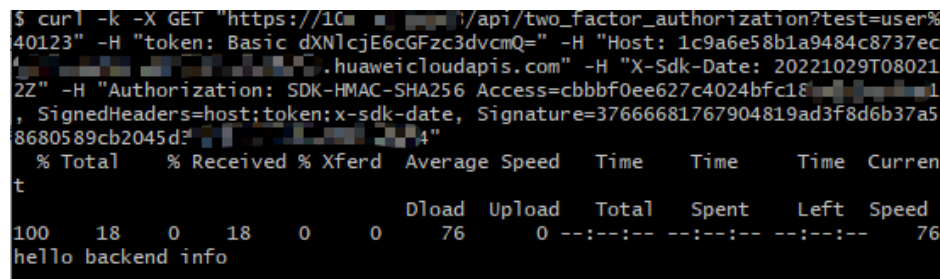
Step 6 Verify the API.

- Call the API on the debugging page of APIG to verify if two-factor authentication is configured successfully.

Add **test** on the **Parameters** tab and add **token** on the **Headers** tab. Use the same parameter values set for the custom authentication function. If the parameter values are different, the server will return a 401 message indicating that the authentication fails.



- Alternatively, call the API with a **curl** command. Download the JavaScript SDK first. To call the API, input a key and secret as well as the header and query string to generate a **curl** command, and then copy this command to your CLI for execution. For details, see section "curl" in the *API Gateway Developer Guide*.



----End

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

8.1 Introduction

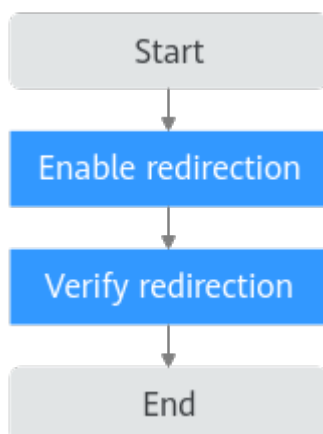
Scenario

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.

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

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.

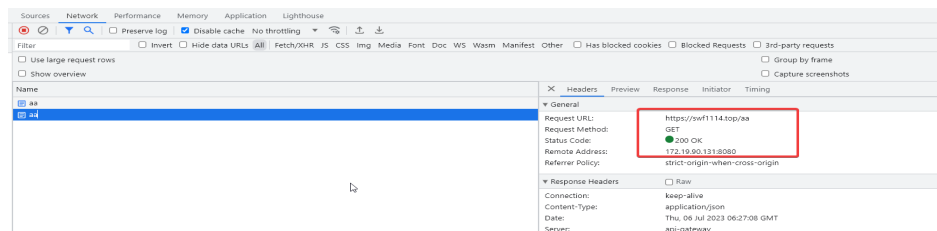
Step 3 On the **Group Information** tab, locate the bound independent domain name, and enable **HTTP-to-HTTPS Auto Redirection**.

----End

Verifying Redirection

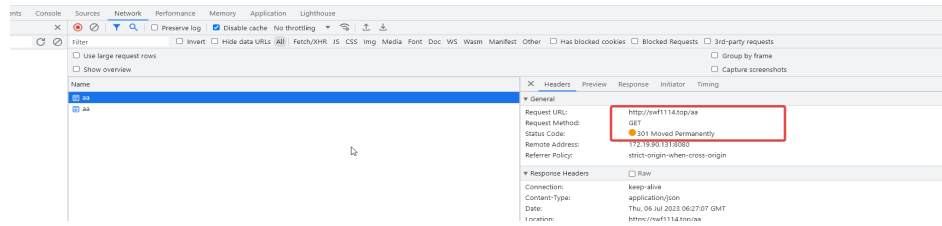
Step 1 Use a browser to call the API through HTTPS.

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

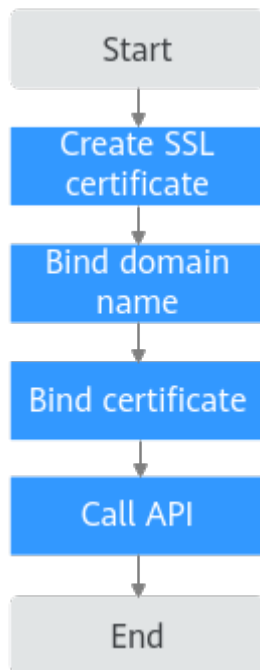
9 Client Authentication with a Dedicated Gateway

9.1 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.

9.2 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.

9.3 Implementation Procedure

One-Way Authentication

Step 1 Log in 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 9-1 Certificate configuration for one-way authentication

Parameter	Description
Name	Enter a certificate name that conforms to specific rules to facilitate search.
Instances Covered	Select Current .
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 9-2 Independent domain name configuration

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

4. Click **OK**.

Step 5 Bind a certificate.

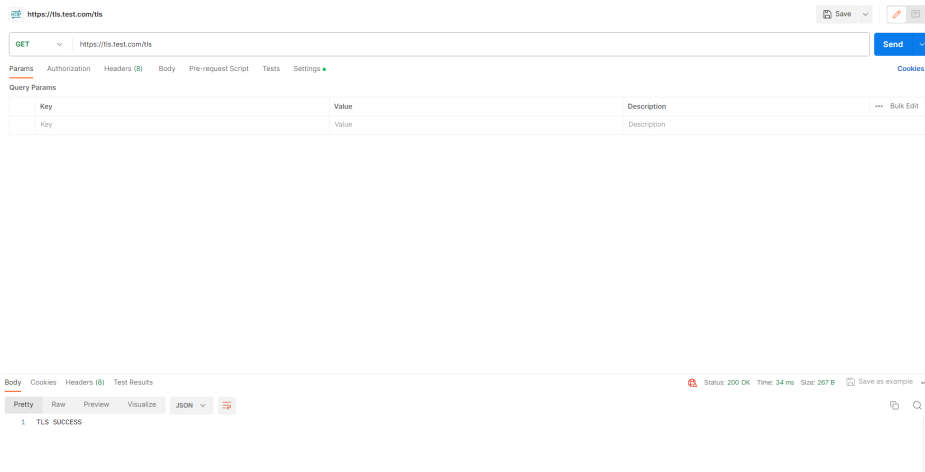
1. In the row that contains the domain name, click **Select SSL Certificate**.
2. Select the created certificate and click **OK**.

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. Otherwise, rectify the fault by following the instructions provided in "Published API Calling" > "Error Codes" in the *API Gateway User Guide*.



----End

Two-Way Authentication

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

Table 9-3 Certificate configuration for two-way authentication

Parameter	Description
Name	Enter a certificate name that conforms to specific rules to facilitate search.
Instances Covered	Select Current .
Content	Enter the certificate content. -----Start certificate----- MIICXgIBAAKBgQC6ndRHy5Dv5TcZiVzT6qF iaMGy61ZibUrmBhUn61vMdvOHmtblST+fSl ZheNACv2hQR4aqJLi4wrcerTaRyG9op3OSh... -----End certificate-----

Parameter	Description
Key	Enter the key. -----Start RSA private key----- MIICXgIBAAKBgQC6ndRHy5Dv5TcZiVzT6qF iaMGy61ZlbUrmBhUn61vMdvOHmtblST+fSl ZheNAcv2hQR4aqJLi4wrcerTaRyG9op3OSh... -----End RSA private key-----
CA	Enter the CA certificate content. After the CA certificate is configured, bind the SSL certificate to the independent domain name and enable Client Authentication. -----Start certificate----- MIICXgIBAAKBgQC6ndRHy5Dv5TcZiVzT6qF iaMGy61ZlbUrmBhUn61vMdvOHmtblST+fSl ZheNAcv2hQR4aqJLi4wrcerTaRyG9op3OSh... -----End certificate-----

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 9-4 Independent domain name configuration

Parameter	Description
Domain Name	Enter a licensed domain name.
Minimum TLS Version	Select TLS1.2 .
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, select **Enable Client Authentication**, and click **OK**.

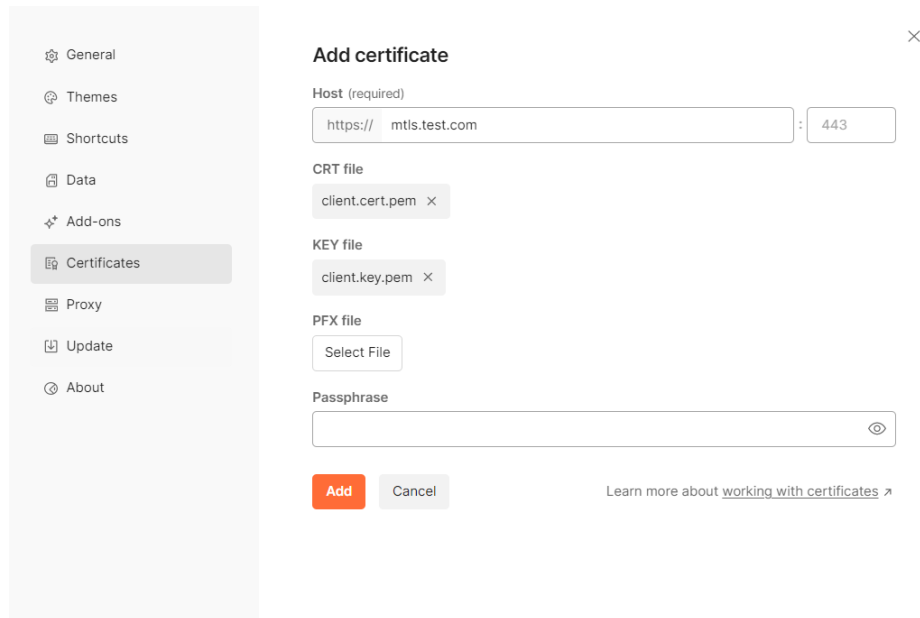
Step 5 Call the API.

Use the API test tool to call the API. If the status code is **200**, the API is successfully called. Otherwise, rectify the fault by following the instructions

provided in "Published API Calling" > "Error Codes" in the *API Gateway User Guide*.

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

10 Invoking an API Policy Backend via Frontend Authentication Parameters (System Parameter)

10.1 Introduction

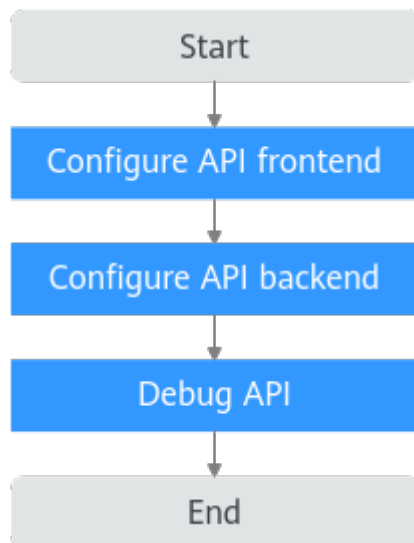
Scenario

APIG allows you to define multiple backend policies and forward API requests to different backends based on these different policies. For example, to distinguish special calls from regular calls, you can define a policy backend that uses frontend custom authentication parameters. This section uses frontend authentication parameters (system parameter) as an example to describe how to forward API requests to a specified backend.

Restrictions

- Before adding a backend policy, set the security authentication mode of the frontend to **Custom** or enable **Two-Factor Authentication** (App or IAM authentication).
- API requests that do not meet the conditions of any backend will be forwarded to the default backend.
- The **Condition Value** of the **System parameter - Frontend authentication parameter** can only be character string, integer, or Boolean.
- A maximum of five policy backends can be defined for an API.

10.2 General Procedure



1. **Configuring the API Frontend**
Set the security authentication mode of the API frontend to **Custom** or enable **Two-Factor Authentication** (app or IAM authentication), and select a custom authorizer. If no custom authorizer is available, click **Create Custom Authorizer**.
2. **Configuring the API Backend**
Add a policy for the backend. Set the **Source** of the **Policy Conditions** to **System parameter - Frontend authentication parameter**, and configure the **Parameter Name**, **Condition Type**, and **Condition Value**. The **Parameter Name** and **Condition Value** must be the same as the key-value pair in the **context** field in the return value of the frontend custom authentication function.
3. **Debugging the API**
Debug the API and check whether the added policy backend is called.

10.3 Implementation Procedure

Prerequisites

1. A custom authentication function has been created. For details, see section "Creating a Function from Scratch" in the *FunctionGraph User Guide*. The function's return value must have a **context** field with key-value pairs, where the values are limited to string, boolean, or integer. The key-value pair corresponds to the **Parameter Name** and **Condition Value** of the **System parameter - Frontend authentication parameter** configured in **Policy Conditions**.

Figure 10-1 Custom authentication function

```

1  # -*- coding:utf-8 -*-
2  import json
3  def handler (event, context):
4      resp = {
5          "statusCode": 200,
6          "body": json.dumps({
7              "status": "allow",
8              "context": {
9                  "test": "123",
10                 "authstatus1": False,
11                 "authstatus2": True,
12                 "num": 1001
13             }
14         }
15         ),
16         "headers": {
17             "Content-Type": "application/json"
18         }
19     }
20     print(resp)
21     return json.dumps(resp)
    
```

2. A custom frontend authorizer has been created. If no custom authorizer is available, create one by following the instructions in section "Custom Authorizer" in the *API Gateway User Guide*.

Configuring the API Frontend

- Step 1** Log in 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 API** and configure the frontend.

Table 10-1 Frontend configuration

Parameter	Description
API Name	Enter a name that conforms to specific rules to facilitate search.
Group	The default option is DEFAULT .
URL	<p>Method: Request method of the API. Default: GET.</p> <p>Protocol: Request protocol of the API. Default: HTTPS.</p> <p>Subdomain Name: The system automatically allocates a subdomain name to each API group for internal testing. The subdomain name can be accessed 1,000 times a day.</p> <p>Path: Path for requesting the API. In this example, enter /1234.</p>

Parameter	Description
Gateway Response	Select a response to be displayed if the gateway fails to process an API request. The default gateway response is default .
Authentication Mode	API authentication mode. Select Custom .
Custom Authorizer	Select the custom authorizer created in Prerequisites .

----End

Configuring the API Backend

Step 1 After configuring the frontend, click **Next**.

Set **Backend Type** to **Mock** and enter **Default backend** in the **Response**.


Step 2 Click  to add a backend policy based on the following table.

Table 10-2 Policy backend configuration

Parameter	Description
Name	Enter a policy name that conforms to specific rules to facilitate search.
Backend Type	Select Mock .
Response	Enter Policy backend .
Policy Conditions	<ul style="list-style-type: none"> • Source: Select System parameter - Frontend authentication parameter. • Parameter Name: Enter authstatus1 under the context field in the response body of the custom authentication function created in Prerequisites. • Condition Type: Select Equal. • Condition Value: Enter False under the context field in the response body of the custom authentication function created in Prerequisites.

Step 3 Click **Finish**.

----End

Debugging the API

The **Mock** backend type helps you view the result with a response. For other backend types, you can check whether the backend address is successfully accessed.

Step 1 On the **APIs** page, click **Debug** to debug the created API.

If **200 OK** is displayed in **Response**, the API is successfully invoked. If the **Policy backend** is also returned, the configured policy backend condition matches the key-value pair in the **context** field in the response body of the custom authentication function, and the API invokes the policy backend. Otherwise, rectify the fault by following the instructions provided in "Published API Calling" > "Error Codes" in the *API Gateway User Guide*.

Request	Response
1 GET /1234 HTTP/1.1	1 HTTP/1.1 200 OK
2 Host: 09c26e8adcf4433066e170f17c...	2 Transfer-Encoding: chunked
3 User-Agent: APIGatewayDebug/1.0	3 Connection: keep-alive
4 X-Api-Authentication: SDK-WMAC-SHA256 Access=b74626c01209401f01808446458e16fc, SignedHeaders=user-agent;x-api-mode;x-sdk-date, Signature=065463261d1f462caae20f7ad9c23eab07150e23f1c1de4e0f74a2dc7a3d3	4 Content-Type: application/json
5 X-Api-Mode: debug	5 Date: Wed, 10 Apr 2024 07:29:49 GMT
6 X-Sdk-Date: 20240410T072949Z	6 Server: api-gateway
7	7 X-Api-Latency: 1117
8	8 X-Api-RateLimit-Api: remain:99,limit:100,time:1 minute
	9 X-Api-RateLimit-Api-Allom: remain:5999,limit:6000,time:1 second
	10 X-Request-Id: bc2854c6fa28a92fb4f588a672147aad
	11
	12 Policy backend

Step 2 On the **APIs** page, click **Modify**. The API configuration page is displayed.

Step 3 Click **Next**. On the backend configuration page, change **Condition Value** of the added policy to **True**.

Step 4 Click **Finish**.

Step 5 Debug the API again.

The policy backend parameter and condition value do not match any key-value pair of the frontend custom authentication function. Therefore, the API policy backend cannot be matched based on the frontend authentication parameters. In this case, **200 OK** is displayed with the return result of **Default backend**, indicating that the default backend of the API is invoked. Otherwise, rectify the fault by following the instructions provided in "Published API Calling" > "Error Codes" in the *API Gateway User Guide*.

Request	Response
1 GET /1234 HTTP/1.1	1 HTTP/1.1 200 OK
2 Host: 09c26e8adcf4433066e170f17c...	2 Transfer-Encoding: chunked
3 User-Agent: APIGatewayDebug/1.0	3 Connection: keep-alive
4 X-Api-Authentication: SDK-WMAC-SHA256 Access=b74626c01209401f01808446458e16fc, SignedHeaders=user-agent;x-api-mode;x-sdk-date, Signature=f01c3080186f6a193f7b34405dc0c1d51a0a11d0518ee04788c7d51ee22f9a19	4 Content-Type: application/json
5 X-Api-Mode: debug	5 Date: Wed, 10 Apr 2024 07:35:03 GMT
6 X-Sdk-Date: 20240410T073503Z	6 Server: api-gateway
7	7 X-Api-Latency: 1225
8	8 X-Api-RateLimit-Api: remain:99,limit:100,time:1 minute
	9 X-Api-RateLimit-Api-Allom: remain:5999,limit:6000,time:1 second
	10 X-Request-Id: 58c1af3040fea42916408900f7b790e1
	11
	12 Default backend

----End

11 Forwarding WebSocket Services Using Dedicated Gateway

Scenario

You can create WebSocket APIs using APIG same as creating an HTTP API. WebSocket is a protocol for full-duplex communication over a single TCP connection. It outweighs HTTP in real-time, two-way communication between the client and server, proving invaluable for applications such as instant messaging, online gaming, and real-time data update in the financial sector.

Restrictions

- WebSocket APIs cannot be debugged on the APIG console.
- The WebSocket API has a timeout, and if the idle time of a WebSocket connection without ping/pong exceeds the configured timeout, the connection expires.

Prerequisites

Prepare a WebSocket service backend for which ping/pong keepalive has been performed.

Procedure

- Step 1** Log in 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 API** and configure the frontend.

Table 11-1 Frontend configuration

Parameter	Description
API Name	Enter a name that conforms to specific rules to facilitate search. For example, API01 .

Parameter	Description
Group	The default option is DEFAULT .
URL	<p>Method: Request method of the API. Set this parameter to GET.</p> <p>Protocol: Request protocol of the API. Default: HTTPS, which corresponds to the WebSocket Secure (WSS).</p> <p>Subdomain Name: The system automatically allocates a subdomain name to each API group for internal testing. The subdomain name can be accessed 1,000 times a day.</p> <p>Path: Path for requesting the API. In this example, enter /hello.</p>
Gateway Response	Select a response to be displayed if the gateway fails to process an API request. default is selected by default.
Authentication Mode	API authentication mode. Select None . (None: Not recommended for actual services. All users will be granted access to the API.)

Step 5 Click **Next** and configure the backend information.

Table 11-2 Backend configuration

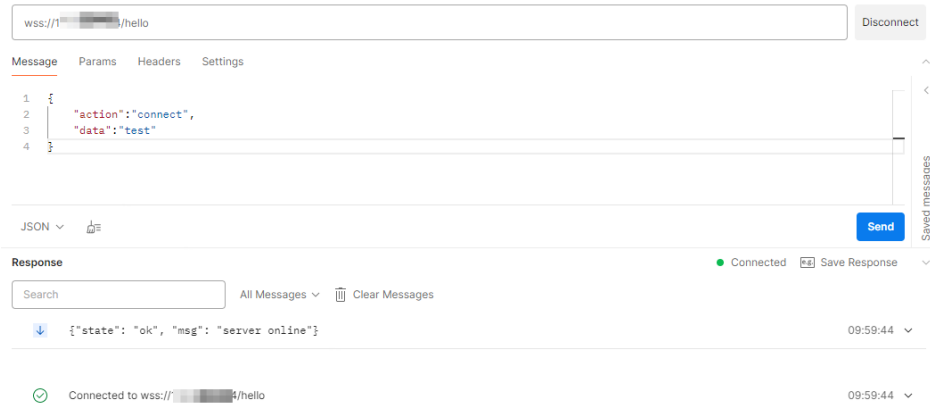
Parameter	Description
Backend Type	Select HTTP&HTTPS .
Load Balance Channel	Determine whether the backend service will be accessed using a load balance channel. For this example, select Skip .
URL	<ul style="list-style-type: none"> • Method: Request method of the API. Set this parameter to GET. • Protocol: Set this parameter to HTTP. • Backend Address: Address and port of the WebSocket backend service. • Path: Path of the backend service. In this example, enter /.
Timeout (ms)	Adjust the backend timeout to be longer than the ping/pong heartbeat interval. For example, if the ping/pong heartbeat interval is 20s, the timeout can be set to (20000 ms, 60000 ms].

Step 6 Click **Finish**.

Step 7 After the API is created, click **Publish Latest Version** on the **APIs** page to publish the API.

Step 8 Use the API test tool to call the API.

In this practice, enter **wss://IP address/hello** to call an API in the DEFAULT group. The IP address is the EIP in the **Gateway Information** on the APIG console.



----End

12 Change History

Table 12-1 Change history

Date	Description
2024-10-09	This issue incorporates the following changes: <ul style="list-style-type: none"> • Updated Selectively Exposing CCE Workloads with a Dedicated Gateway. • Added Client Authentication with a Dedicated Gateway. • Added Invoking an API Policy Backend via Frontend Authentication Parameters (System Parameter). • Added Forwarding WebSocket Services Using Dedicated Gateway.
2023-07-30	This issue incorporates the following change: Added HTTP-to-HTTPS Auto Redirection with a Dedicated Gateway .
2023-04-30	This issue incorporates the following changes: <ul style="list-style-type: none"> • Updated this document for the new console. • Added Request Throttling 2.0 with a Dedicated Gateway and Two-Factor Authentication with a Dedicated Gateway.
2023-04-12	This issue incorporates the following change: Added Exposing Backend Services Across VPCs Using a Dedicated Gateway and Interconnecting a Dedicated Gateway with WAF .
2021-09-30	This issue is the first official release.