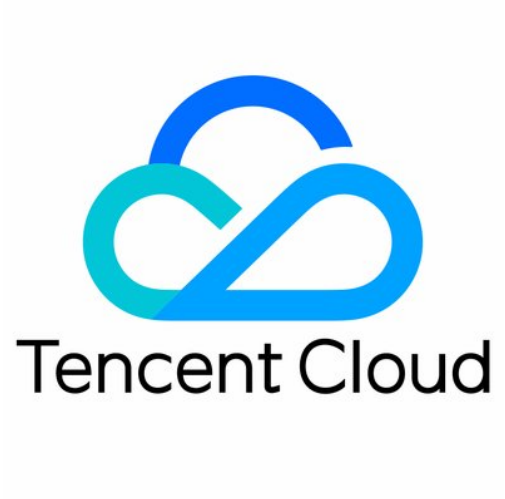
API Gateway

Plugin Usage

Product Documentation



Contents

Plugin Usage Overview

Last updated：2023-12-22 09:59:24

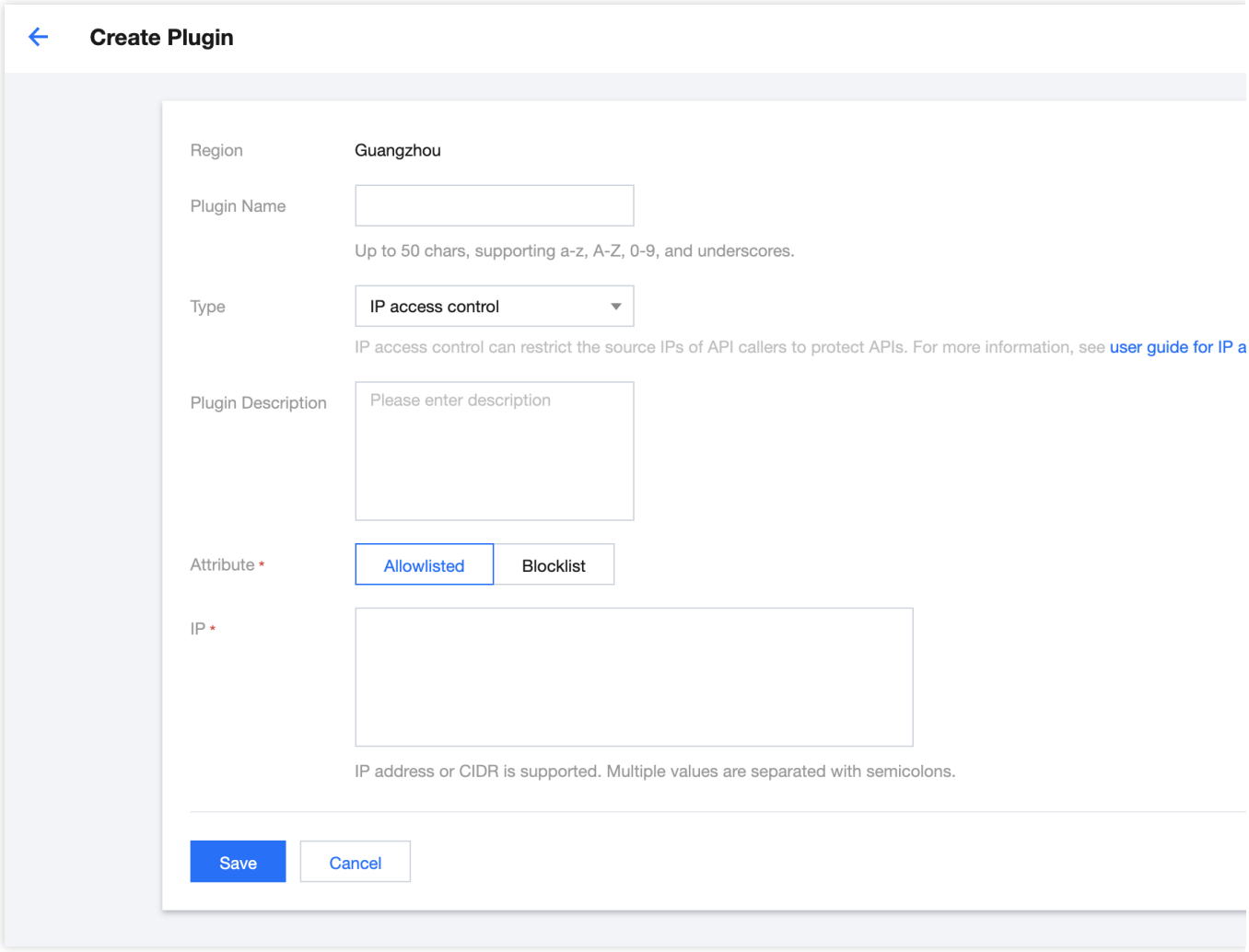
Plugin Overview

Plugins are advanced feature configurations provided by API Gateway. You can create configuration items such as IP access control through plugins and then bind the plugins to APIs for them to take effect.
Plugins have the following advantages over traditional configuration items:
Feature configuration is decoupled from API configuration. One plugin can be bound to multiple APIs under different services.
Hot update is supported for configurations. After a plugin is bound to an API, it can take effect without releasing the service.

Directions

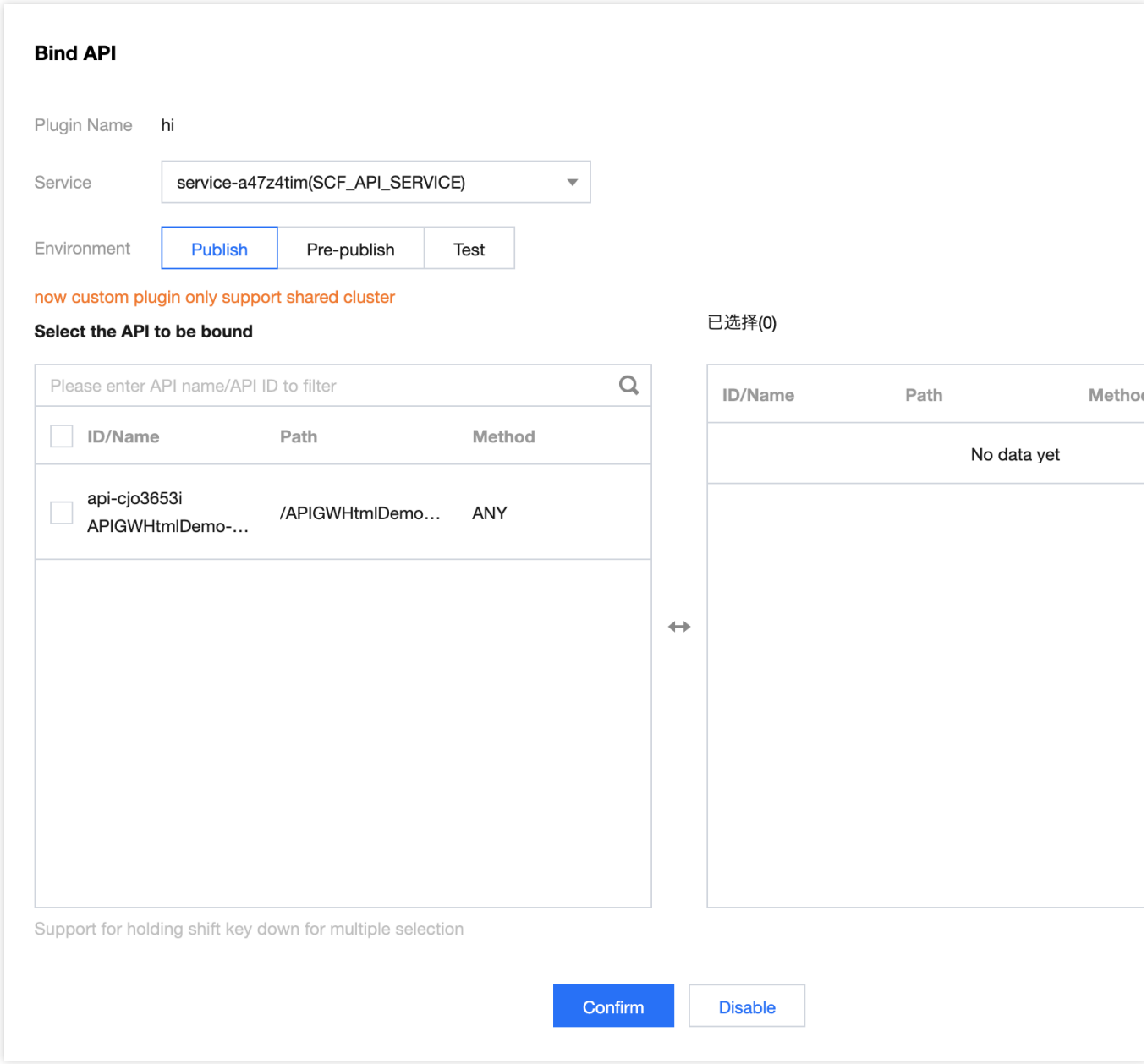
Step 1. Create a plugin

1. Log in to the API Gateway console.
2. Click **Plugin** in the left sidebar to open the plugin list page.
3. Click **Create** in the upper-left corner of the page to create a plugin.



Step 2. Bind an API and make the plugin effective

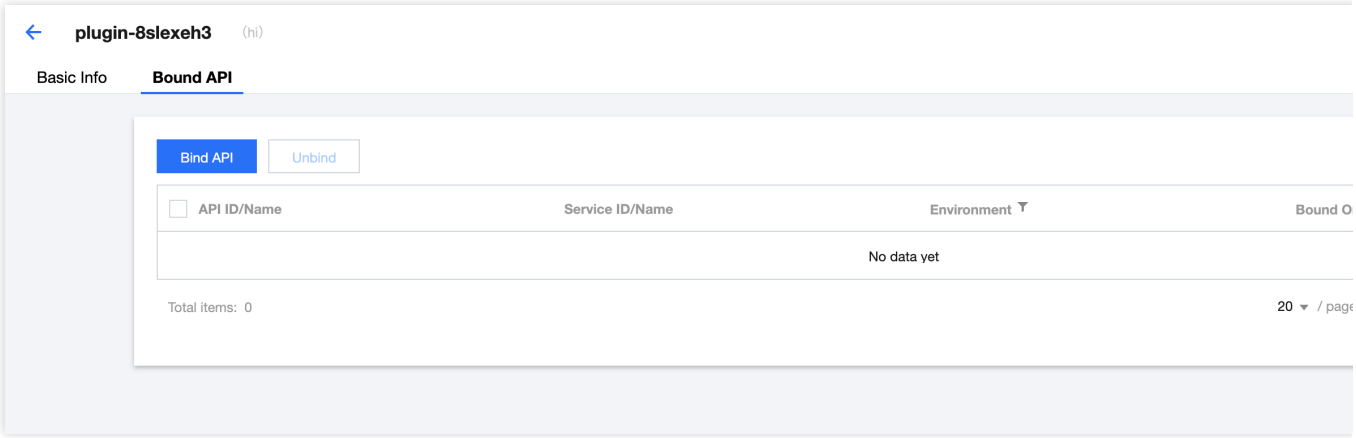
1. Select the just created plugin in the list and click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the API to which the plugin needs to be bound.



3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

Step 3. View the plugins bound to an API

1. Click **Service** in the left sidebar.
2. In the service list, click the name of the target service.
3. In the API list, click the name of the target API to enter the API details page.
4. On the API details page, select the tab of **Plugin management** to view the information of the plugins bound to the target API.



Supported Plugin Type

IP access control.

Note:

API Gateway currently only supports the IP access control plugin and will offer more plugins such as traffic throttling, circuit breaker, and parameter conversion in the future.

Plugin Rules

An API only allows one plugin of the same type to be bound to it.

Plugins are region-specific and can be bound to APIs only in the same region. Cross-region binding is not supported.

APIs can be bound to plugins only after they are released in the corresponding environment. Unreleased APIs cannot be bound.

The deactivation of APIs does not affect their binding relationships with plugins, and the plugins will still take effect after the APIs are re-released in the environment.

Plugins support hot updates, and all binding and unbinding operations can take effect without re-releasing the service.

After APIs are deleted, their binding relationships with plugins will also be deleted.

IP Access Control

Last updated：2023-12-22 09:59:35

Overview

IP access control is a security protection capability provided by API Gateway. It is mainly used to restrict the source IPs of API callers. You can allow/reject API requests from a certain source by configuring the IP allowlist/blocklist of an API.

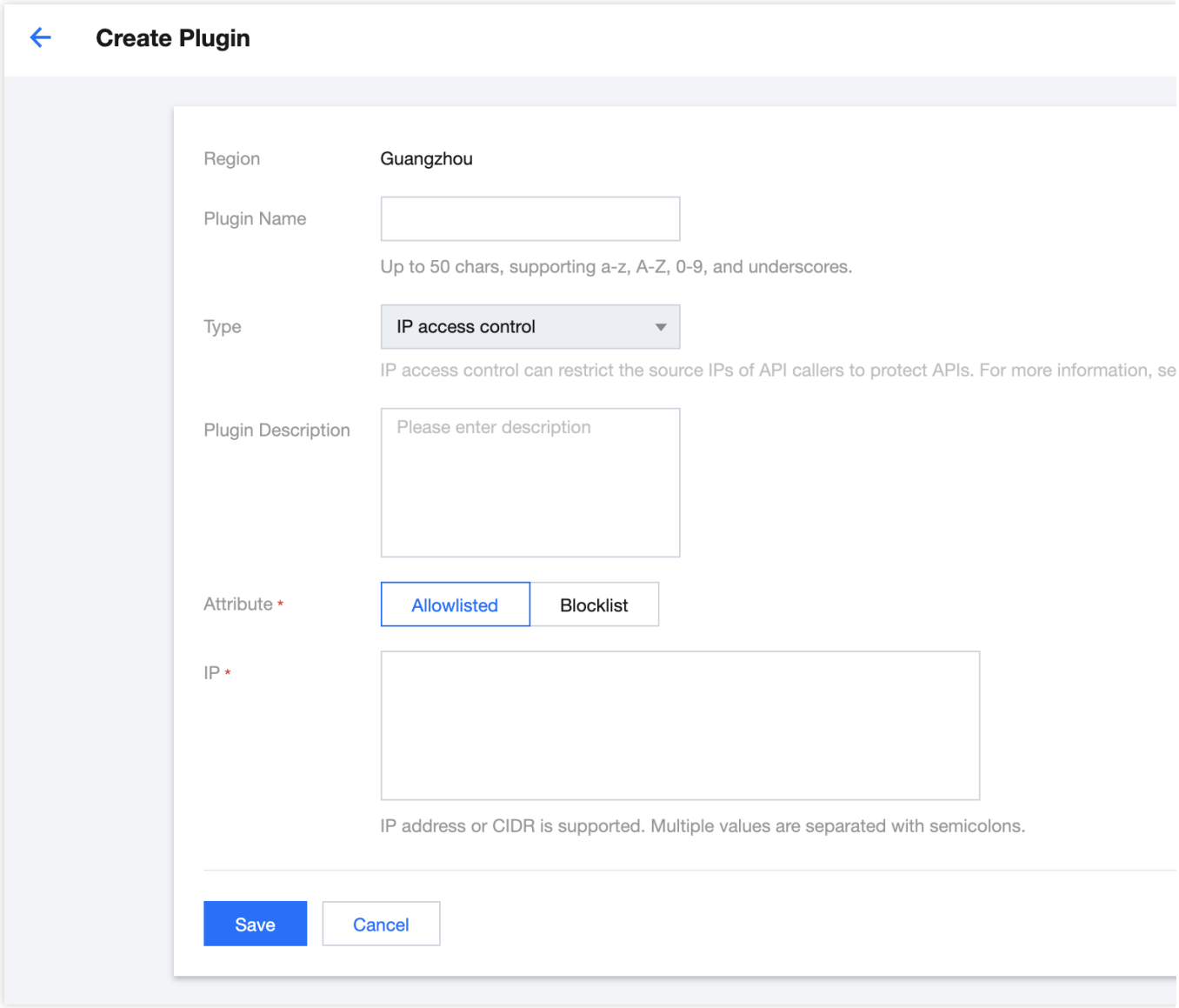
Note:

The original IP access control policy data has been migrated to the IP access control plugin, which can be managed on the Plugin page.

Directions

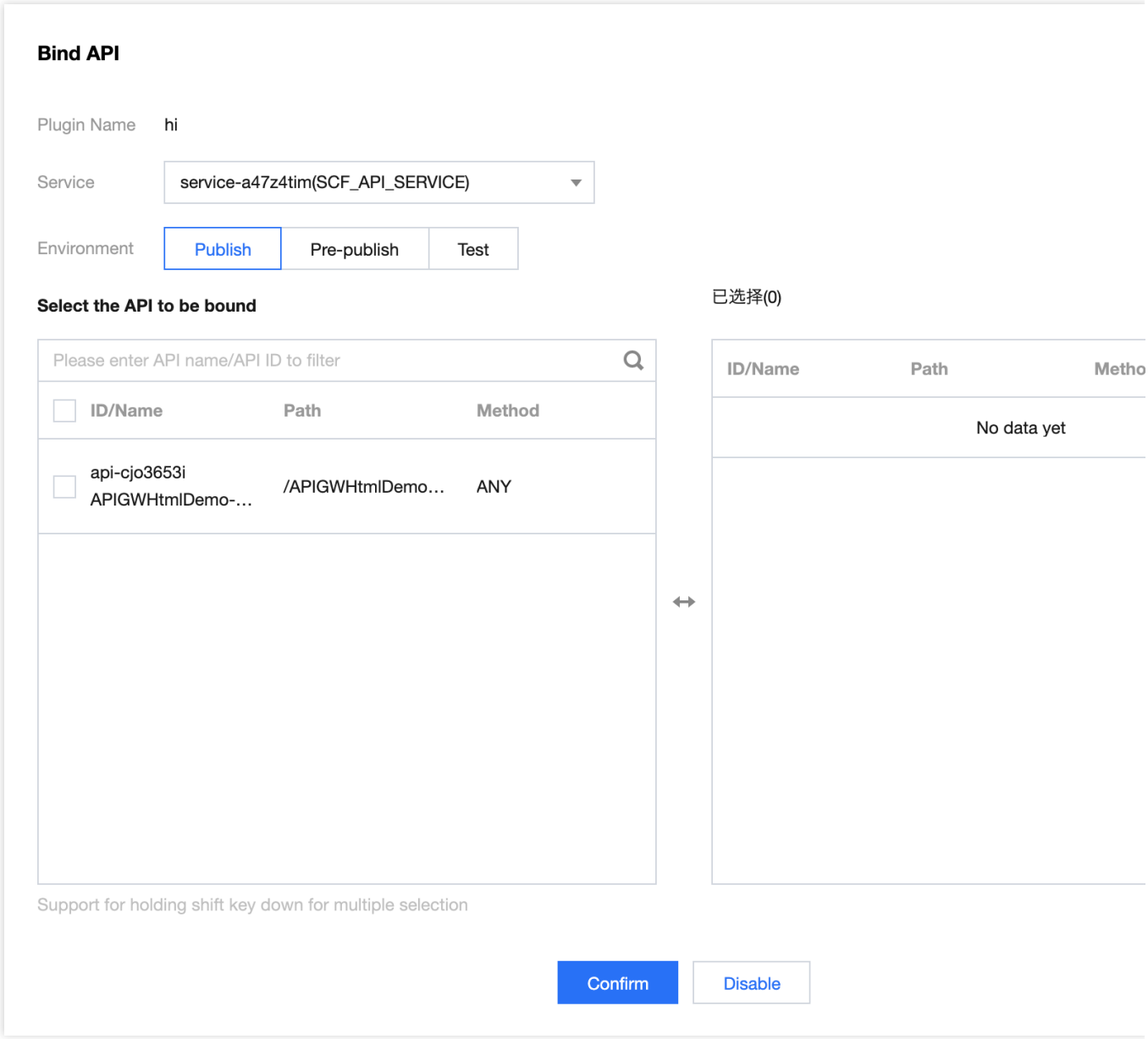
Step 1. Create the plugin

1. Log in to the API Gateway console.
2. Click **Plugin** on the left sidebar to open the plugin list page.
3. Click **Create** in the top-left corner to create an IP access control plugin.



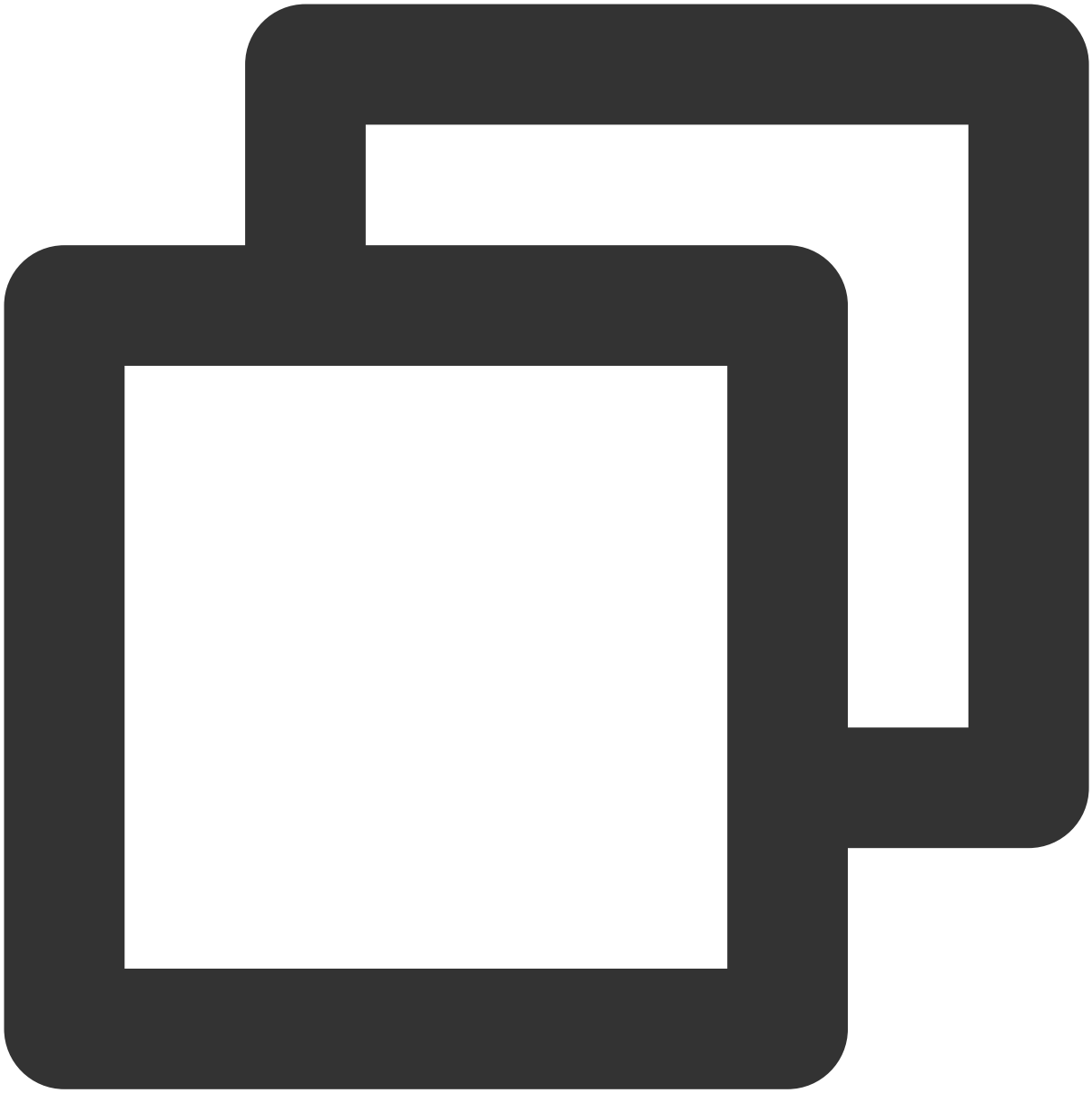
Step 2. Bind an API and make the plugin effective

1. Select the just created plugin in the list and click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the API to which the plugin needs to be bound.



3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

PluginData



```
{
    "type":"white_list",    // IP access control type. Valid values: white_list: al
    "blocks":"1.1.1.1\\n1.1.1.0/24"    // IP ranges separated with `\\n`
    "descriptions":{"1.1.1.1":"desc", "1.1.1.0/24":"desc"} // IP description, which
}
```

Notes

The IP access control plugin supports blocklist and allowlist modes. When the allowlist is used, requests from IPs not in the allowlist will be rejected by API Gateway; when the blocklist is used, requests from IPs in the blocklist will be rejected by API Gateway.
Multiple IPs or CIDR blocks can be entered in the IP access control plugin, which should be separated with semicolons.
You can add descriptions to IPs in the IP access control plugin in the `descriptions` field, which is optional.

Limits

Currently, a shared instance does not support access control of client IPs on the **private network**.

Basic Traffic Throttling

Last updated：2023-12-22 09:59:46

Overview

Basic traffic throttling plugin is a powerful traffic throttling component provided by API Gateway. It can throttle traffic in three dimensions of API, application, and client IP by second, minute, hour, or day. You can create a basic traffic throttling plugin and bind it to your API to take effect and protect your backend services.

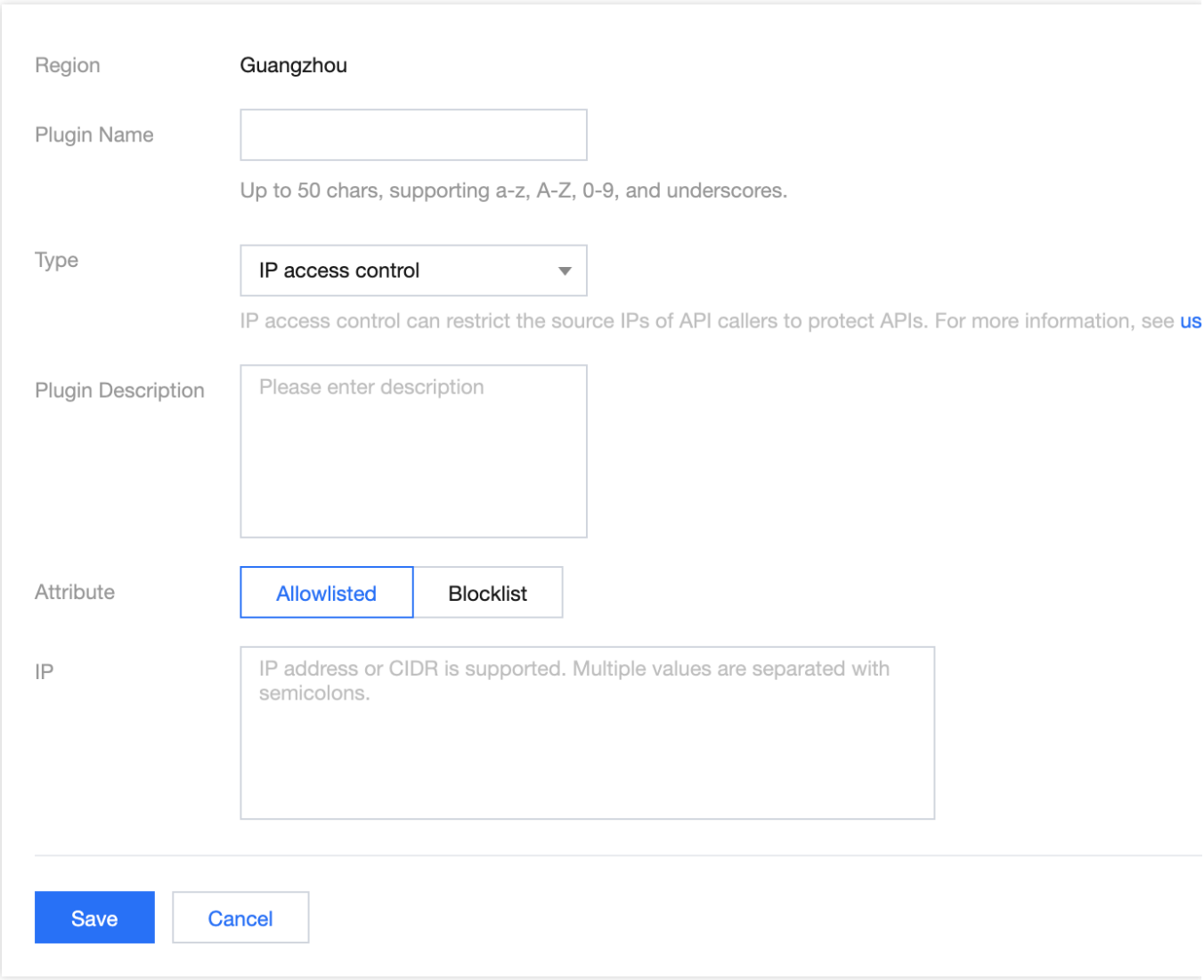
Directions

Step 1. Create the plugin

1. Log in to the API Gateway console.
2. Click **Plugin** on the left sidebar to open the plugin list page.
3. Click **Create** in the top-left corner of the page and select **Basic Throttling** as the plugin type to create a basic traffic throttling plugin.

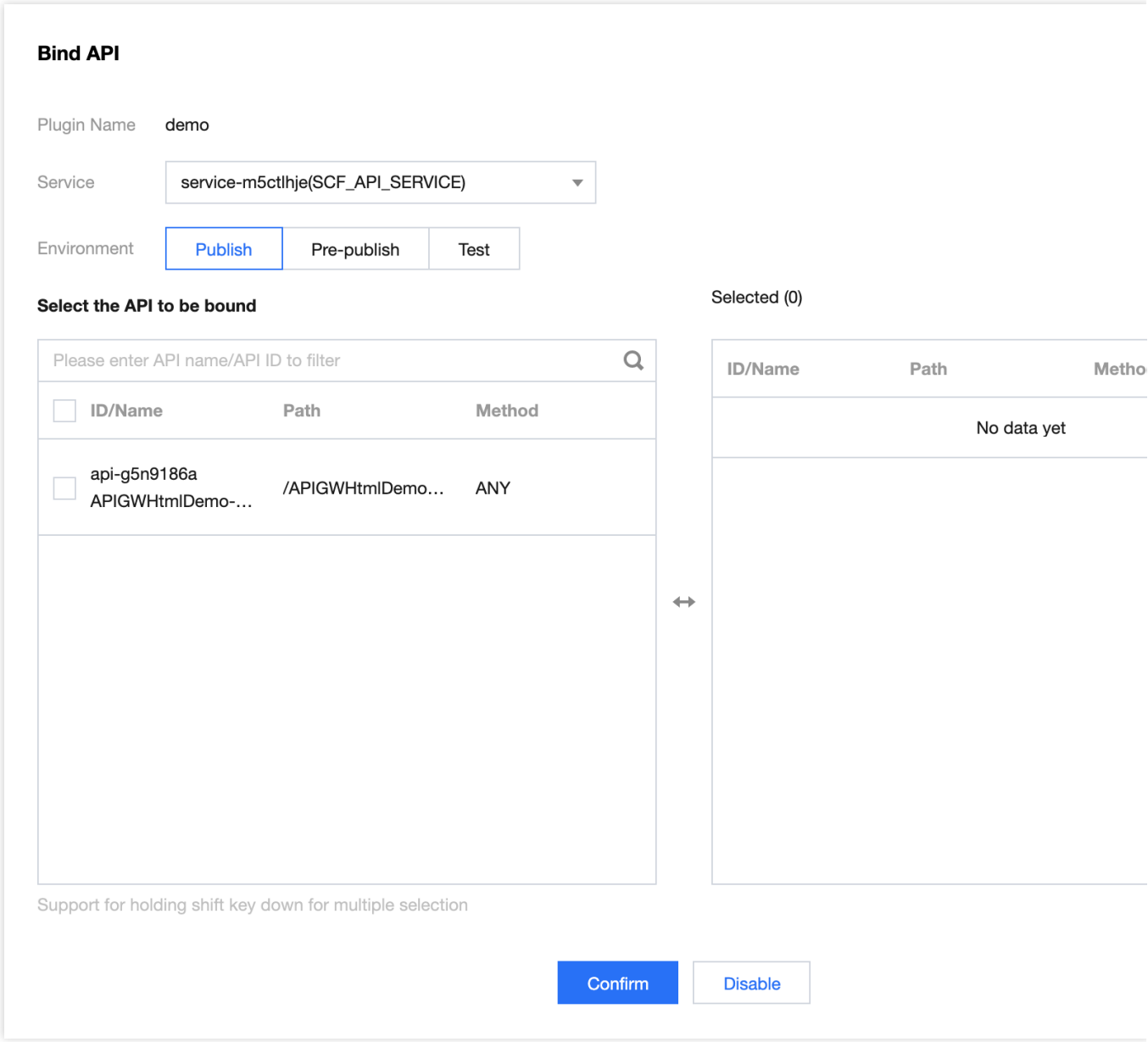
Parameter	Required	Description
Duration	Yes	The duration of traffic throttling, which supports four dimensions: second, minute, hour, and day. It is used in conjunction with "throttling limit" to indicate the upper limit of the number of requests per unit time.
API Throttling Limit	Yes	The upper limit of the number of times an API can be accessed within a certain period of time.
Application Throttling Limit	No	The upper limit of the number of times an application can be accessed within a certain period of time, which takes effect for all applications bound to this API.
Client IP Throttling Limit	No	The upper limit of the number of times a client IP can be accessed within a certain period of time, which takes effect for all client IPs bound to this API.
Special Application	No	Up to 30 items can be entered. For such applications, the basic API traffic throttling of the traffic throttling policy still takes effect, but you need to set an additional traffic throttling threshold for them. Meanwhile, the basic application traffic throttling and user traffic throttling of the traffic throttling policy will stop working for such applications.

Special Client IP	No	Up to 30 items can be entered. For such IPs, the basic API traffic throttling of the traffic throttling policy still takes effect, but you need to set an additional traffic throttling threshold for them. Meanwhile, the basic application traffic throttling and client IP traffic throttling of the traffic throttling policy will stop working for such IPs.



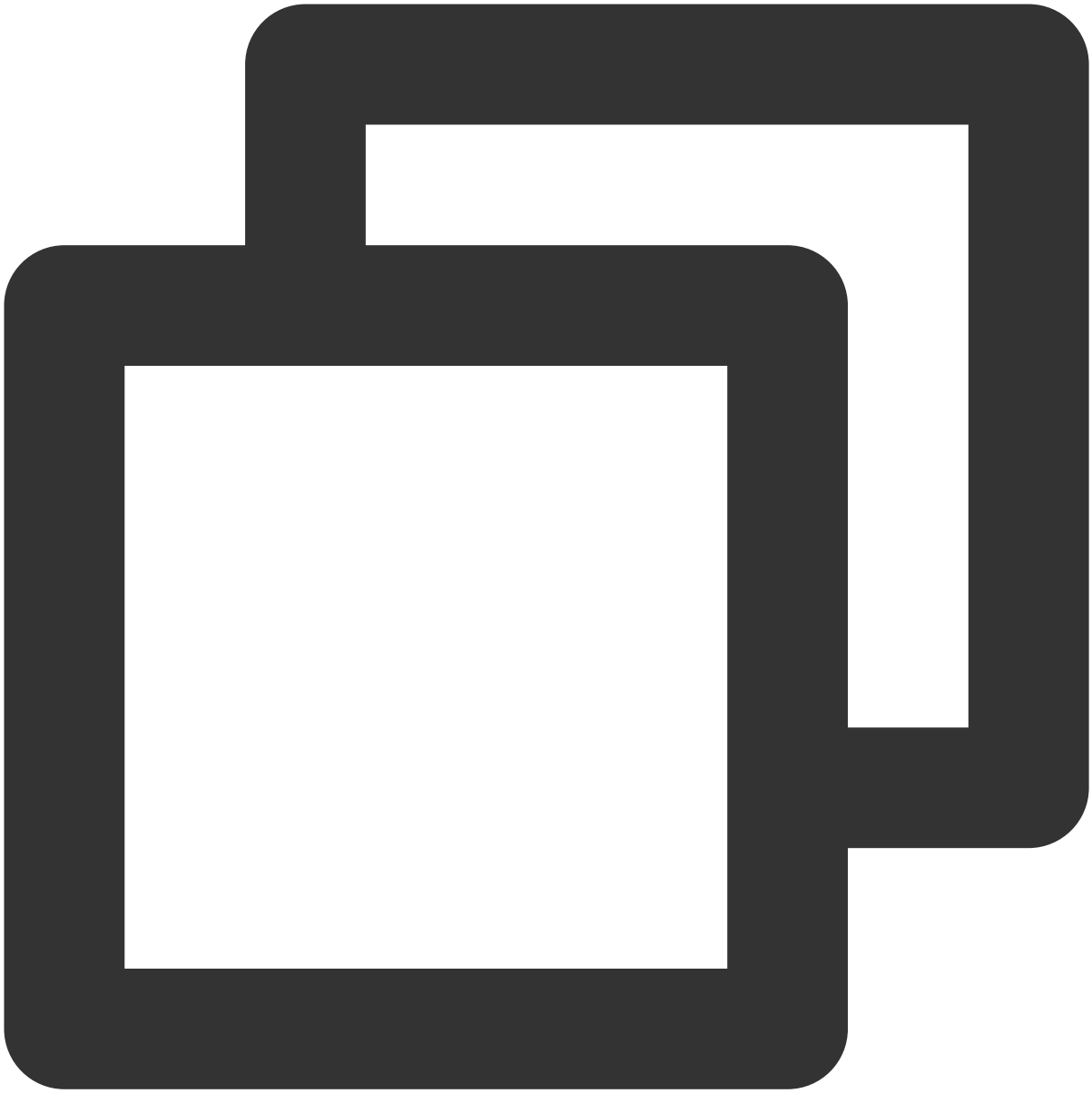
Step 2. Bind an API and make the plugin effective

1. Select the just created plugin in the list and click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the API to which the plugin needs to be bound.



3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

PluginData



```
{
    "expire_type":"hour", // Traffic throttling time window unit. Valid values: day
    "expire":1, // Traffic throttling time window
    "api_rate_limit":500, // API throttling limit, which must be a positive integer
    "app_rate_limit":1, // Application throttling limit, which must be a positive i
    "ip_rate_limit":2, // Client IP throttling limit, which must be a positive inte
    "spec_app_rate_limits":[ // List of special applications for traffic throttling
        {
            "app_id":"app-3q9l4909", // Application ID
            "rate_limit":10 // Throttling limit, which must be a positive integer
        }
```

```
    ],
    "spec_ip_rate_limits":[ // List of special client IPs for traffic throttling
        {
            "ip_key":"172.16.0.1", // Client IP
            "rate_limit":10 // Throttling limit, which must be a positive integer
        }
    ]
}
```

Notes

The basic traffic throttling plugin will be affected by service traffic throttling and API traffic throttling. If multiple traffic throttling rules take effect at the same time, the lowest throttling limit will prevail.
For example, if the traffic throttling threshold of an API is set to 500 QPS in the basic traffic throttling plugin, the throttling limit of the service to which the API belongs is 100 QPS, and the throttling limit of the API itself is 50 QPS, then the actually effective threshold is 50 QPS.

Parameter Traffic Throttling

Last updated：2023-12-22 10:00:00

Overview

The parameter traffic throttling plugin can control the traffic according to the configured request parameters and conditions. It can be configured as follows:
You can set traffic throttling by second, minute, hour, and day.
You can set conditions for client request parameters and built-in system parameters of API Gateway to use different traffic throttling dimensions.
You can use a single parameter or combine multiple parameters to configure traffic throttling.
Note:
As the parameter traffic throttling plugin and basic traffic throttling plugin are for the same purpose, an API can be bound to only one of them.

Directions

Step 1. Create the plugin

1. Log in to the API Gateway console.
2. Click **Plugin** on the left sidebar to open the plugin list page.
3. Click **Create** in the top-left corner of the page and select **Parameter Traffic Throttling** as the plugin type to create a parameter traffic throttling plugin.

Parameter	Required	Description
Enable default policy	Yes	Whether to enable the default policy. The default policy implements traffic throttling at the API level and is independent from all other traffic throttling policies. For each request, the default policy will be checked first before other policies.
Default throttling limit	Yes if the default policy is enabled	Upper limit of the number of times an API can be accessed within a certain period of time, which must be a positive integer.
Default traffic throttling duration	Yes if the default policy is enabled	It supports four units: second, minute, hour, and day, and is used together with the default throttling limit.

Traffic throttling policy	Yes	Parameter traffic throttling policy. You can create up to 10 policies in the same parameter traffic throttling plugin. The configuration items in a policy are as detailed below.

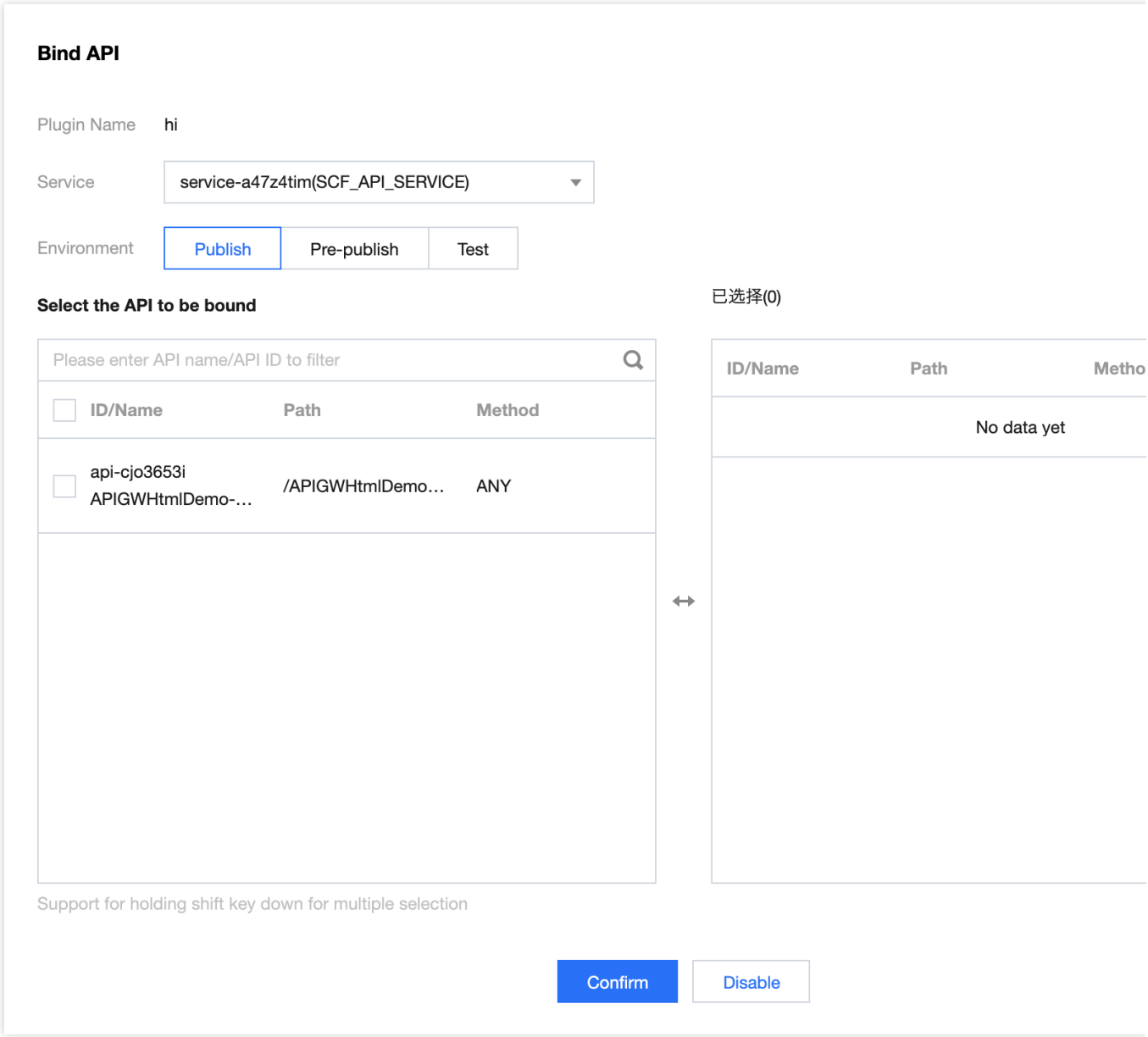
You can create up to 10 traffic throttling policies in the same parameter traffic throttling plugin. Each policy has the following configuration items:

Parameter	Required	Description
Policy name	Yes	Current policy name, which can contain up to 50 characters and must be unique in the same plugin.
Weight	Yes	You can enter a positive integer between 0 and 100, which must be unique in the same plugin. The greater the weight, the higher the priority.
Trigger condition	No	If you set a condition, only when it is met can this traffic throttling policy be executed. Enter a conditional expression. For more information on the rules of writing conditional expressions, see **Notes**.
Traffic throttling parameter location	Yes	You can enter only one traffic throttling parameter and need to select the location and set the parameter name. For example, `Header.ClientIP` indicates to perform traffic throttling for each value of the `ClientIP` parameter in `Header`. The following locations are supported: Header, Query, and Path. The `Path` parameter represents a complete API path, for which you don't need to enter the parameter name.
Traffic throttling parameter name	Yes	You can enter only one traffic throttling parameter and need to select the location and set the parameter name. For example, `Header.ClientIP` indicates to perform traffic throttling for each value of the `ClientIP` parameter in `Header`. The traffic throttling parameter name needs to be used together with the parameter location. The `Path` parameter represents a complete API path, for which you don't need to enter the parameter name.
Throttling limit	Yes	Traffic throttling threshold for the traffic throttling parameter in this policy, which must be a positive integer and used together with the traffic throttling duration.
Traffic throttling duration	Yes	It supports four units: second, minute, hour, and day, and is used together with the throttling limit.

Step 2. Bind an API and make the plugin effective

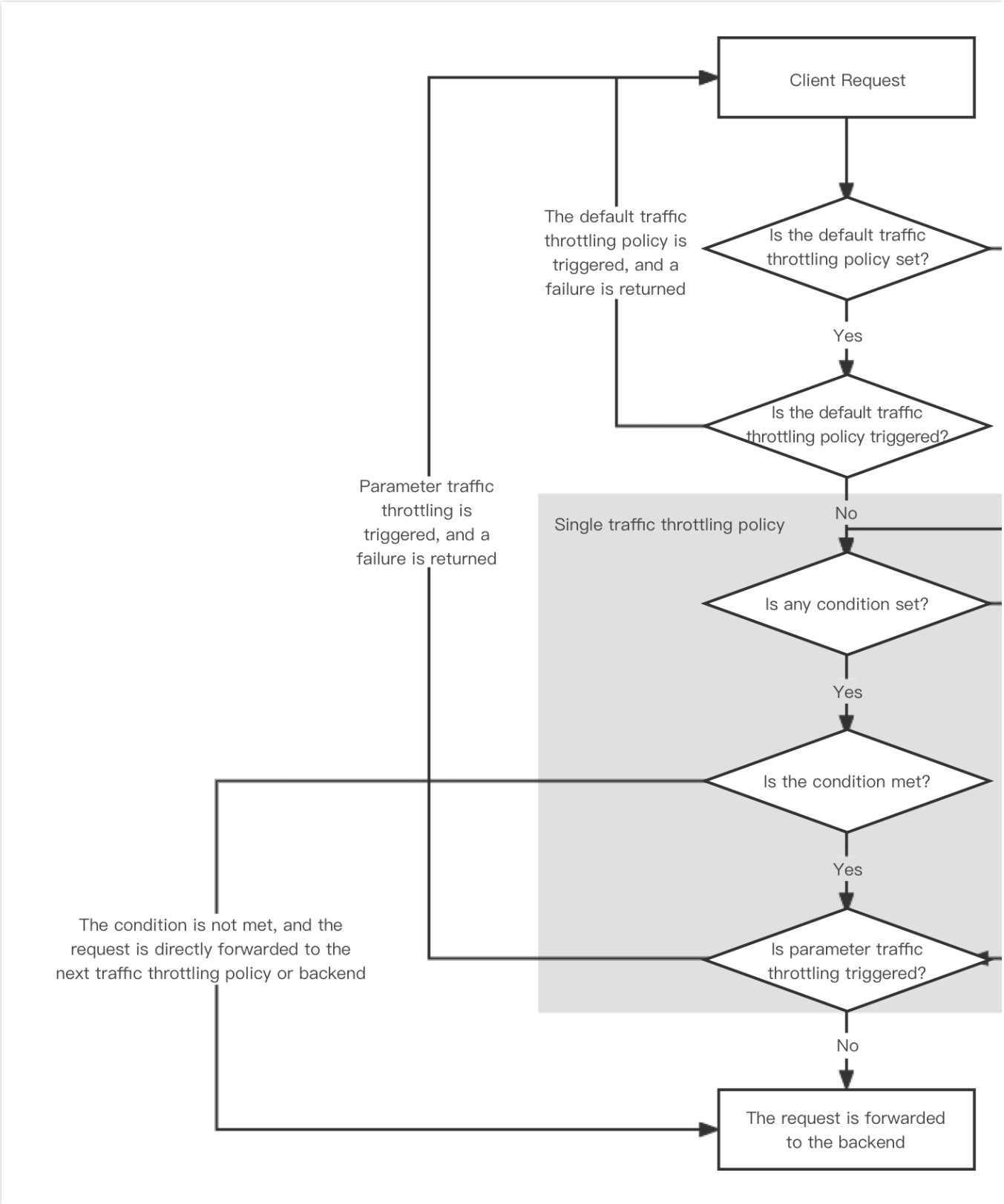
1. Select the just created plugin in the plugin list and click **Bind API** in the **Operation** column.

2. In the **Bind API** pop-up window, select the service, environment, and the API to which the plugin needs to be bound.



3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

How It Works

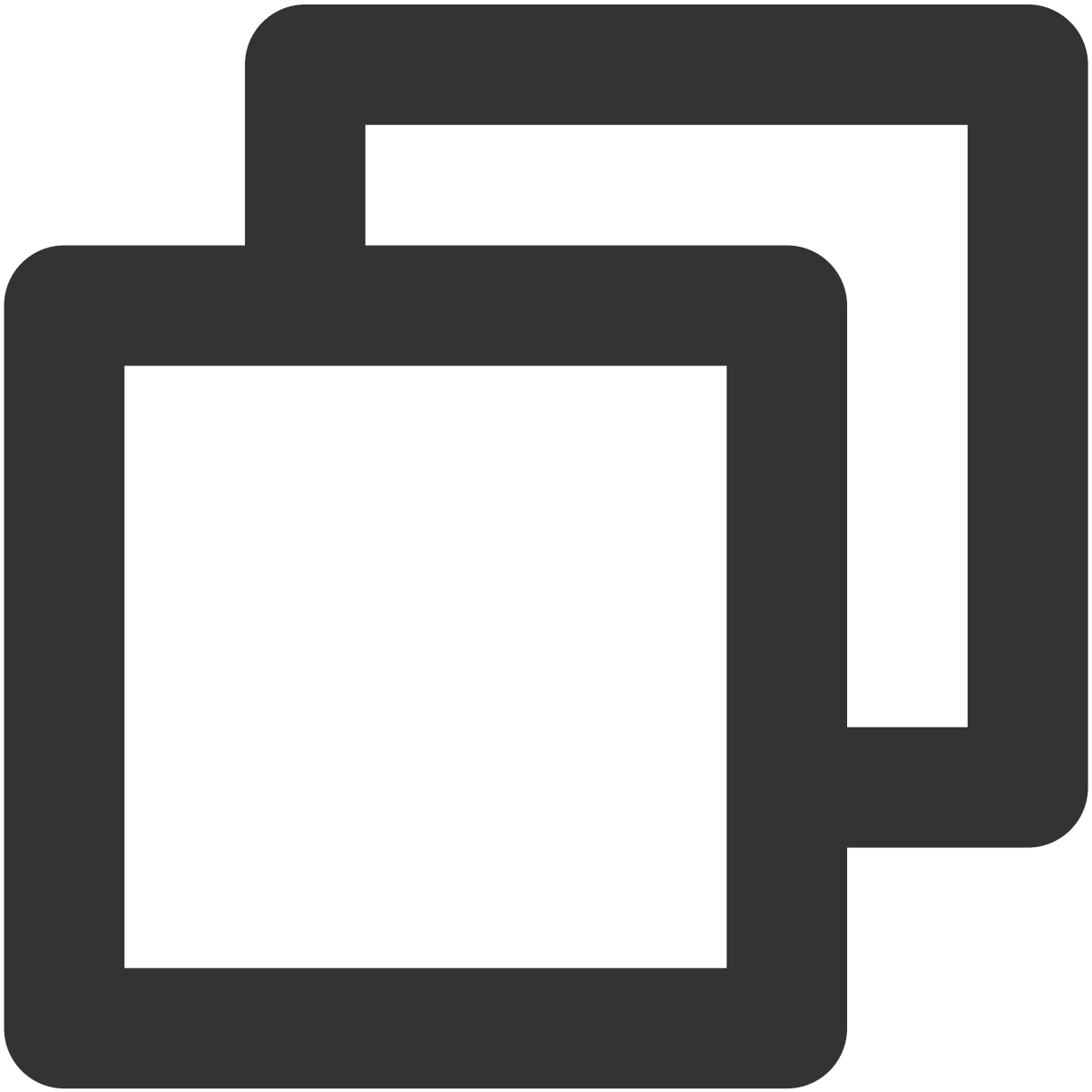


The default policy implements traffic throttling at the API level and is independent from all other traffic throttling policies. For each request, the default policy will be checked first before other policies.

If you configure multiple traffic throttling policies in a plugin, API Gateway will sort them in a descending order by the policy weight and check whether the request meets the condition of each policy one by one. If any policy is not met,

traffic throttling will be triggered, and the request will be rejected.
When a traffic throttling policy is being checked, if a conditional expression is configured, the condition will be checked first before the parameter; otherwise, the parameter will be checked directly.
If you set the threshold to 10 per minute for the `header.userid` parameter in a traffic throttling policy, the threshold will take effect for each different value of this parameter in the request.

PluginData



```
{
    "default_window":60, // Traffic throttling time window in seconds. If the value
    "default_rate_limit":5, // Traffic throttling threshold, which must be a positi
    "strategies":[ // List of parameter traffic throttling policies. You can add 1-
        {
            "name":"a", // Policy name.
            "strategy_weight":0, // Policy execution priority, which must be unique
            "parameters":[ // List of traffic throttling parameters. Currently, you
                {
                    "type":"query", // Type of traffic throttling parameter. Valid
                    "name":"a" // Name of traffic throttling parameter. If the para
                }
            ],
            "rate_limit":1, // Traffic throttling threshold, which must be a positi
            "window":1, // Traffic throttling time window in seconds
            "condition":"" // Policy trigger condition. If the value is an empty st
        }
    ]
}
```

Notes

Conditional expressions of the parameter traffic throttling plugin are the same as those of the conditional routing plugin. For more information on how to write expressions, see Conditional Expression Writing Guide.

CORS

Last updated：2023-12-22 10:00:10

Overview

Cross-Origin Resource Sharing (CORS) is a W3C standard. It allows web application servers to perform cross-origin access control, so that cross-origin data transfer can be conducted securely. Currently, API Gateway supports configuring CORS rules to allow or deny corresponding cross-origin requests as needed.
If the default CORS configuration of API Gateway cannot meet your needs, you can configure custom complex CORS rules through the CORS plugin and bind them to APIs for taking effect.

Directions

Step 1. Create the plugin

1. Log in to the API Gateway console.
2. Click **Plugin** on the left sidebar to open the plugin list page.
3. Click **Create** in the top-left corner of the page and select **Cross-Origin Resource Sharing (CORS)** as the plugin type to create a CORS plugin.

Parameter	Required	Description
Origin	Yes	Specify the origins of allowed cross-origin requests. You can specify multiple origins and separate them by commas. You can configure `*`, which means that all domain names are allowed. Be careful not to omit the protocol name `http` or `https`. If the port is not the default port 80, you also need to include the port.
Method	Yes	GET, PUT, POST, DELETE, and HEAD methods are supported. You can enumerate one or more allowed CORS request methods.
Allow-Headers	No	Specify the custom HTTP request headers that can be used for subsequent `OPTIONS` requests. You can specify multiple headers and separate them by commas. You can configure `*`, which means that all header are allowed. If you leave this parameter empty, all headers will be denied.
Expose-Headers	No	Specify the headers that can be exposed to the `XMLHttpRequest` object. You can specify multiple headers and separate them by commas. You can configure `*`, which means that all header are allowed.

		If you leave this parameter empty, all headers will be denied.
Allow Cookies	No	Specify whether to allow cookies.
Max-Age	Yes	Set the validity period of the result obtained by `OPTIONS` in seconds. The value must be a positive integer, such as 600.

Region Beijing

Plugin Name

Up to 50 chars, supporting a-z, A-Z, 0-9, and underscores.

Type Cross-Origin Resource Sharing

Configure custom W3C-compliant complicate cross-origin rules for API. See CORS Plugin Usage Guide

Plugin Description Please enter description

Origin *

Enter the allowed origins, which should start with "http://" or "https:/". Separate multiple origins with comr

Method * PUT GET POST DELETE HEAD

Allow-Headers

Enter the allowed headers. Separate multiple origins with commas (,). Enter * if all headers are allowed.

Expose-Headers

Enter the headers that can be exposed to XMLHttpRequest. Separate multiple headers with commas (,). E

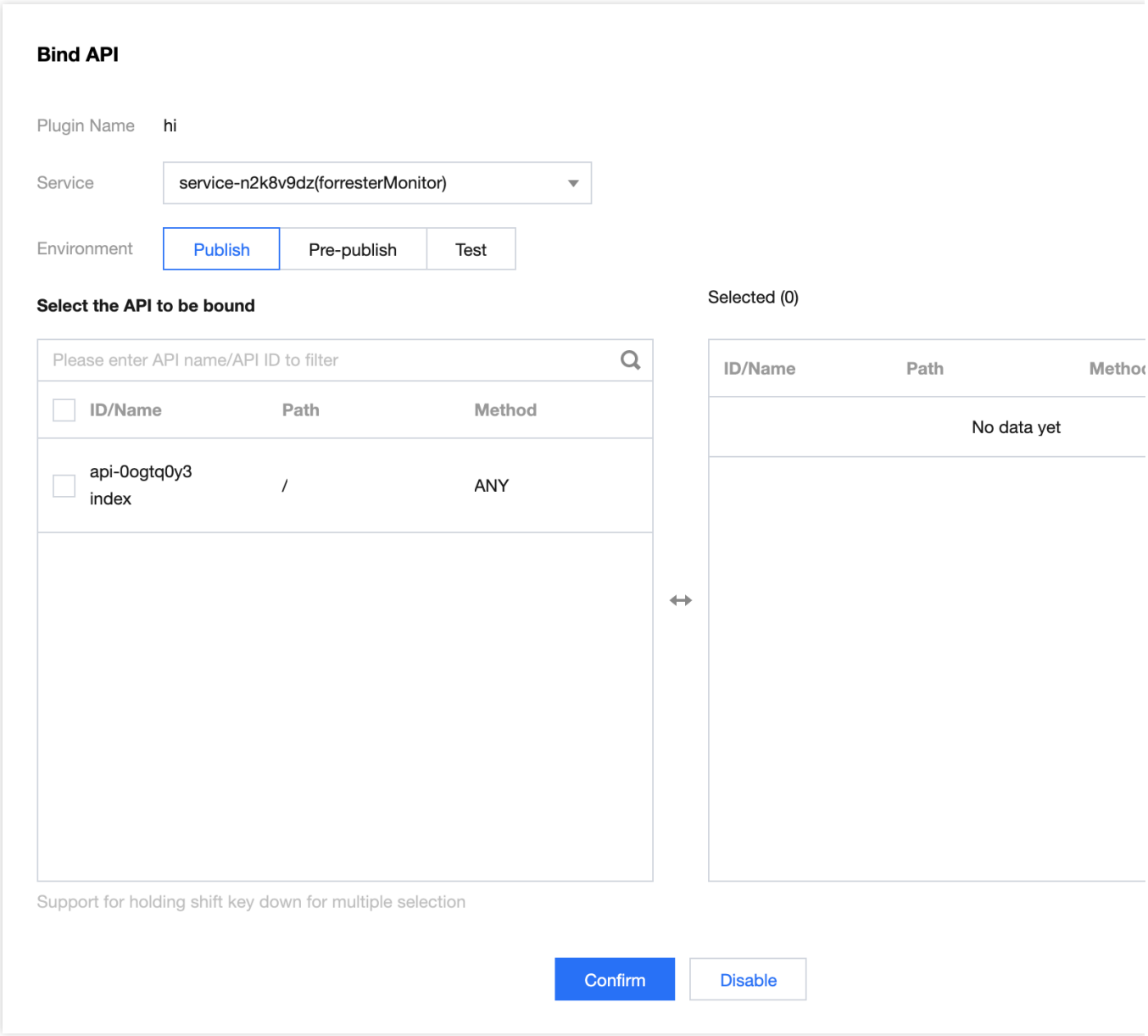
Allow cookies

Max-Age * Enter a positiv seconds

Save Cancel

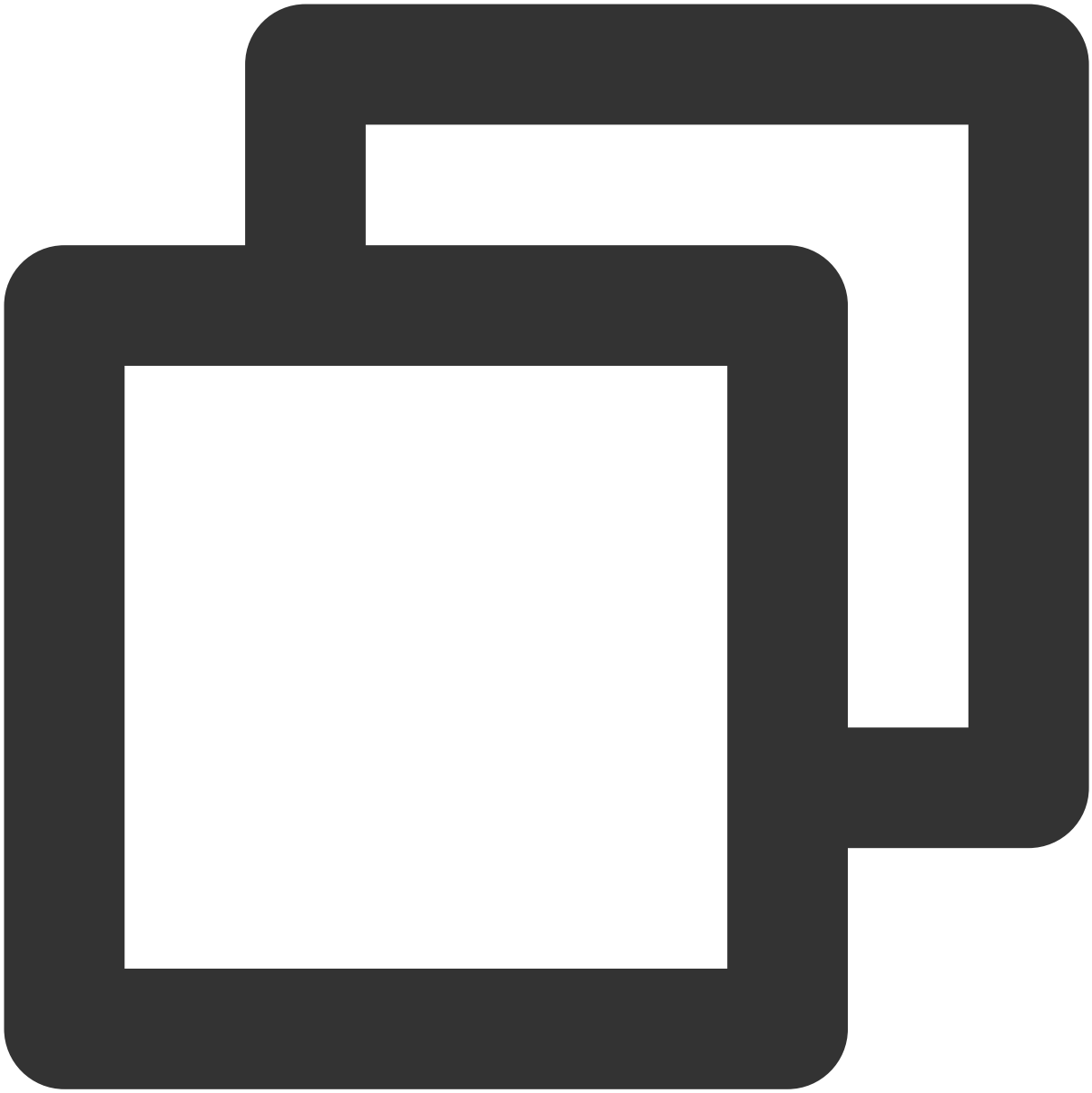
Step 2. Bind an API and make the plugin effective

1. Select the just created plugin in the list and click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the API to which the plugin needs to be bound.



3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

PluginData



```
{
    "allow_origin":[ // Allowed origins. * is supported, indicating that all domain
        "*"
    ],
    "allow_methods":[ // Allowed method. Valid values: GET, PUT, POST, DELETE, HEAD
        "PUT",
        "GET",
        "POST",
        "DELETE",
        "HEAD"
    ],
```

```
    "allow_headers":[ // Allowed request headers. * is supported, indicating that a
        "X-Api-ID"
    ],
    "expose_headers":[ // Headers that can be exposed to the `XMLHttpRequest` objec
        "X-Api-ID"
    ],
    "allow_credentials":true, // Whether to allow cookies
    "max_age":600 // Validity period of the result obtained by `OPTIONS` in seconds
}
```

Notes

Currently, there are two places in API Gateway where you can set CORS rules:

Create API > frontend configuration > CORS is supported: Enable the **CORS is supported** configuration item when creating an API, and API Gateway will add `Access-Control-Allow-Origin : *` in the response header by default.

For more information on the CORS plugin described in this document, see Directions.

The CORS plugin has a higher priority than the **CORS is supported** configuration item. When the former is bound to an API, the latter of the API will not take effect.

Conditional Routing

Last updated：2023-12-22 10:00:22

Overview

The conditional routing plugin can forward different client requests to different backend addresses based on the request and system parameter values. It is widely used in scenarios such as grayscale release, blue-green deployment, and tenant routing.

Directions

Step 1. Create the plugin

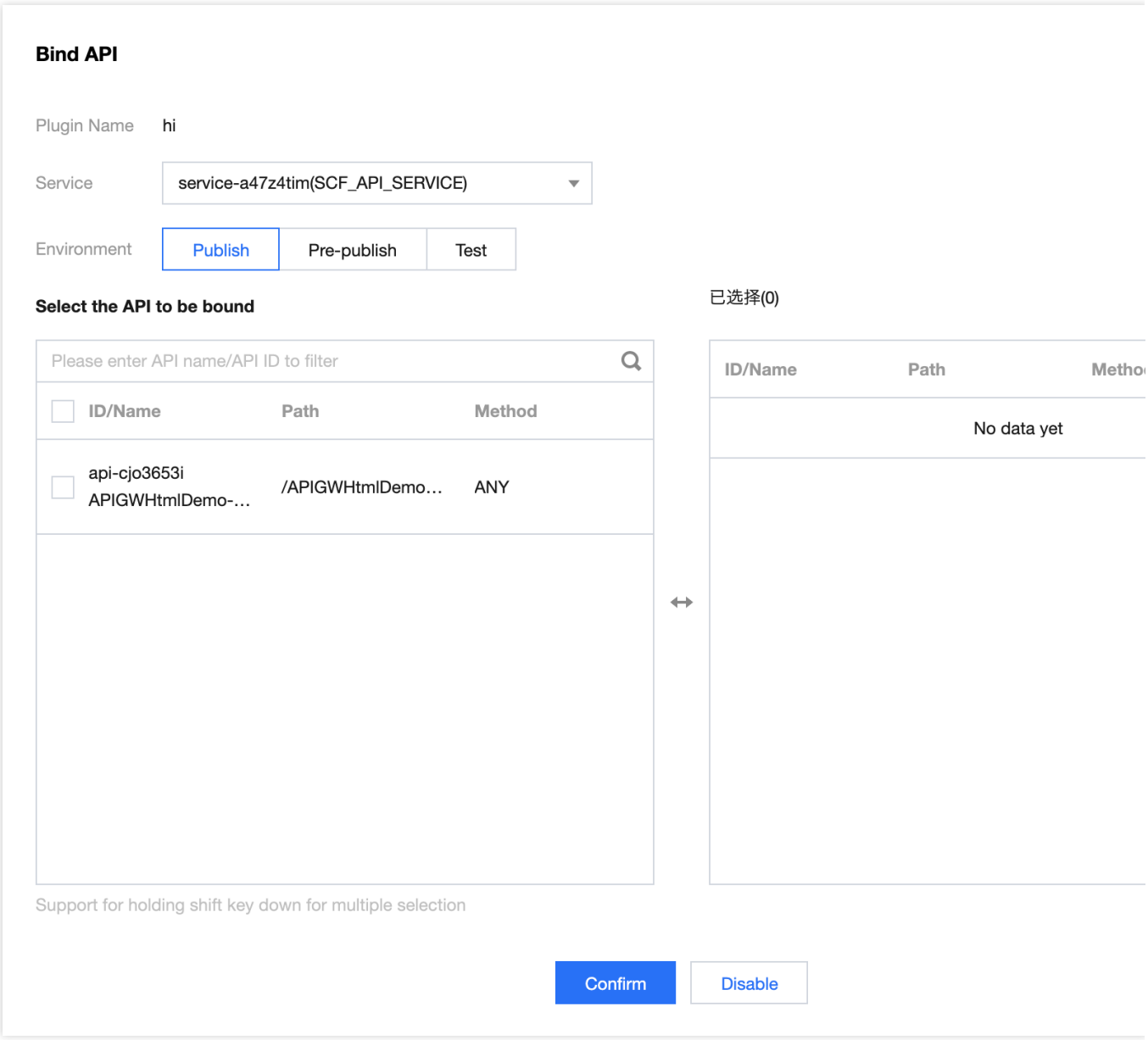
1. Log in to the API Gateway console.
2. Click **Plugin** on the left sidebar to open the plugin list page.
3. Click **Create** in the top-left corner of the page and select **Conditional Routing** as the plugin type to create a conditional routing plugin.

You can create up to 10 routing policies in a conditional routing plugin and need to enter the following content in each policy:

Parameter	Required	Description
Policy name	Yes	Current policy name, which can contain up to 50 characters and must be unique in the same plugin.
Weight	Yes	Priority for policy match. You can enter a positive integer between 0 and 100. If it is not set, it will be 0 by default. The greater the weight, the higher the match priority. For policies with the same weight, the later the creation time, the higher the priority.
Trigger condition	Yes	Conditional expression that judges whether the client request meets the condition. For more information, see the relevant description below.
Backend type	Yes	Public network URL/IP, VPC resources, SCF, Mock, and TSF are supported.
Backend configuration	Yes	Backend to which the client request will be forwarded if the request meets the condition. Enter the backend configuration in YAML format.

Step 2. Bind an API and make the plugin effective

1. Select the just created plugin in the list and click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the API to which the plugin needs to be bound.



3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

Conditional Expression Writing Guide

Parameter description

A conditional expression supports the following two types of parameters:

Request parameter in a certain location: Currently, only `Header`, `Path`, and `Query` parameters are supported.

If the `a` parameter in the `Header` location is configured in the frontend configuration, you can use `header.a` to represent this parameter in the plugin.

The `Path` parameter has no parameter name and is therefore represented by a path; for example, `path='/test'` indicates that the condition is met if the request path is `/test`.

The value definition of the `Path` parameter must start with `/`, such as `'/test'`.

You can **use `sysparam` to reference system parameters of the current request** without defining them in the API; however, if you define a parameter with the same name in the API, the value in the API will be overwritten by the value of the custom parameter. We recommend you enter the parameter value in lower camel case and make it case-insensitive. The following system parameters can be used in the routing plugin:

sysparam.clientIp: Client IP.

sysparam.httpScheme: Request protocol, which is HTTP or HTTPS.

sysparam.clientUa: `UserAgent` field uploaded by the client.

Note:

The parameter location is case-insensitive, while the parameter name is case-sensitive; for example, in the above `header.a` parameter, the parameter location `header` is case-insensitive, while the parameter name `a` is case-sensitive.

A conditional expression supports the following constant types:

Constant Type	Description	Example
STRING	String	Double and single quotation marks are supported, such as "Hello" and 'hello'
INTEGER	Integer	1001, -1
NUMBER	Floating point	0.1, 100.0
BOOLEAN	Boolean	true, false

Writing rules

You can use `and` and `or` to concatenate different expressions.

You can use parentheses `()` to specify the priority for condition match.

As a built-in function, `Random()` can generate a floating point parameter where the value is in `NUMBER` data type and ranges from 0 to 1 for random judgment.

If a nonexistent parameter such as `param.unknown = 1` is used in the expression, `false` will be returned.

You can use the regular expression function `regex()` to match the parameter value, for example, `regex(query.name,"colou?r")`. You need to enclose the string in the regular expression with single or

double quotation marks.

You can use the `exists()` function to specify whether a parameter exists, for example, `exists(header.Accept)`.

Both `==` and `=` can be used to judge the "equal to" relationship.

Sample conditional expressions

The result is true with a probability of 5%: **Random() < 0.05**

`UserName` in the custom `Header` parameter is `Admin`, and the source IP is 47.47.74.77: **header.UserName = 'Admin' and sysparam.clientIp = '47.47.74.77'**

The user ID (a parameter in `Header`) of the current request is `1001`, `1098`, or `2011`, and the request uses the HTTPS protocol: **sysparam.httpScheme = 'https' and (header.id = 1001 or header.id = 1098 or header.id = 2011)**

Backend Configuration Writing Guide

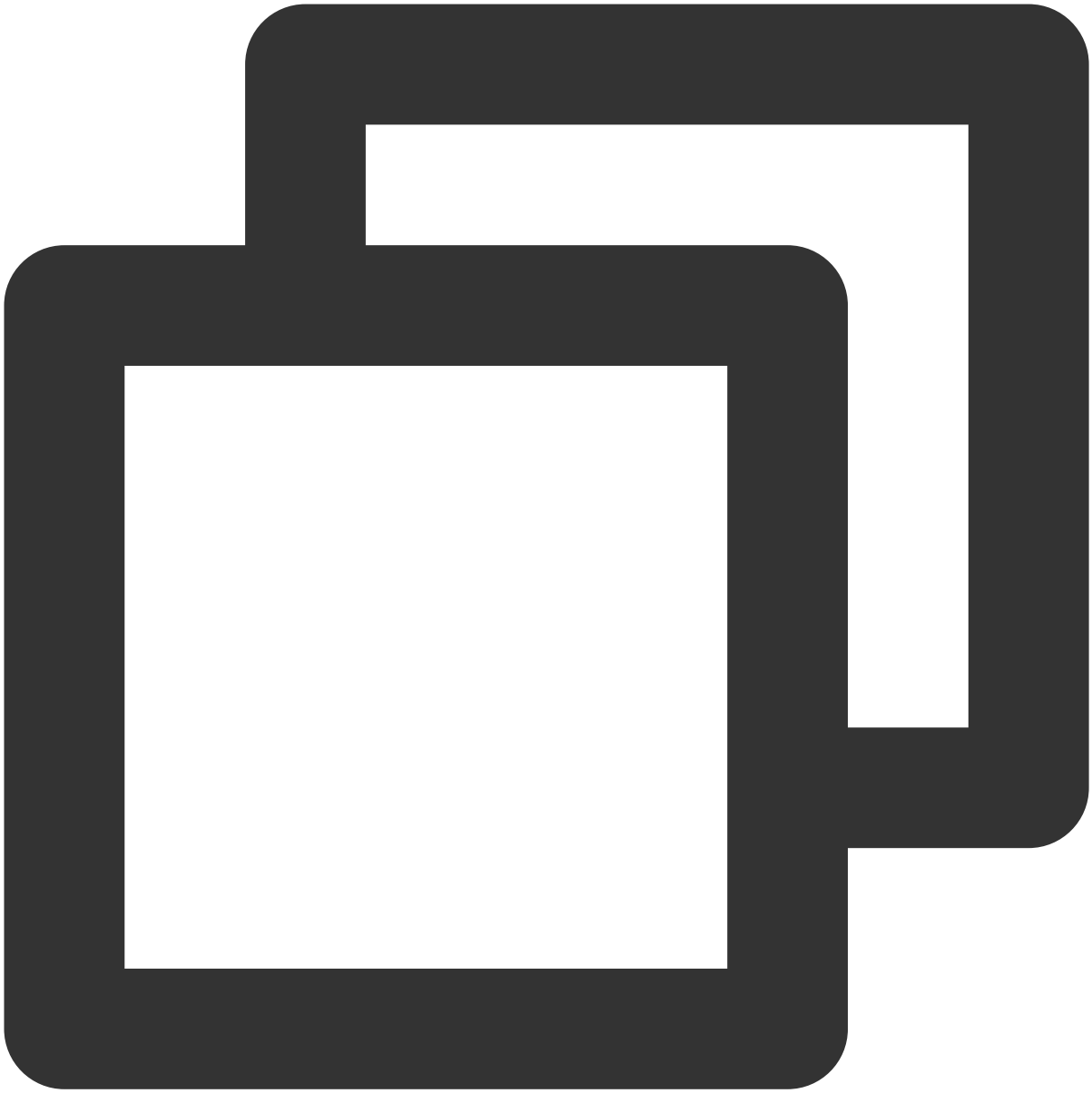
You can enter the configuration of the backend connected to public network URL/IP, VPC resources, SCF, Mock, and TSF:

The backend configuration must be in the YAML format.

The fields in the backend configuration are in one to one correspondence to the fields in the CreateApi API.

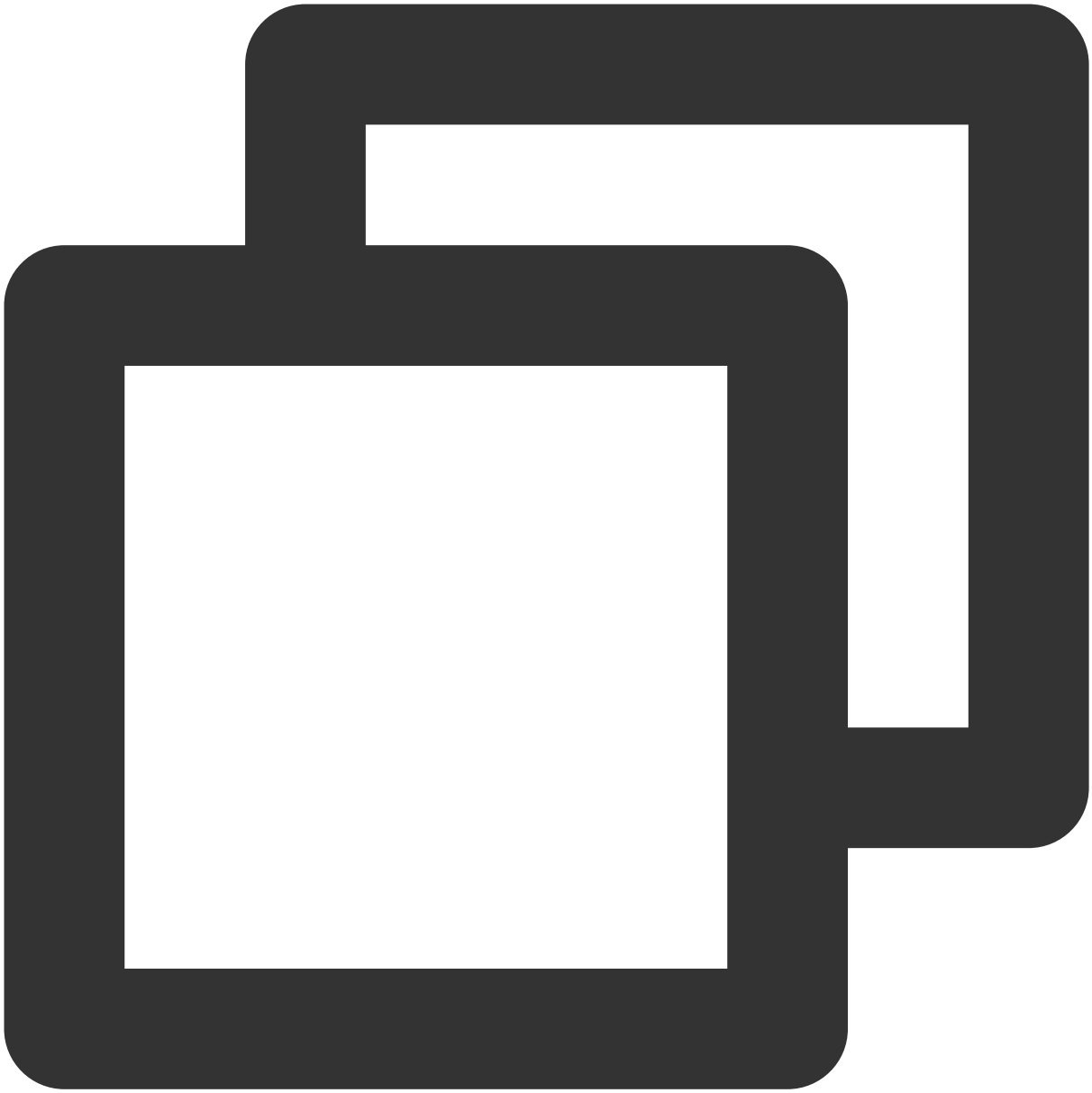
The conditional routing plugin creation page lists the demo configuration code for each type of backend. You can configure the backend simply by modifying the parameter values.

Backend connection to public network URL/IP



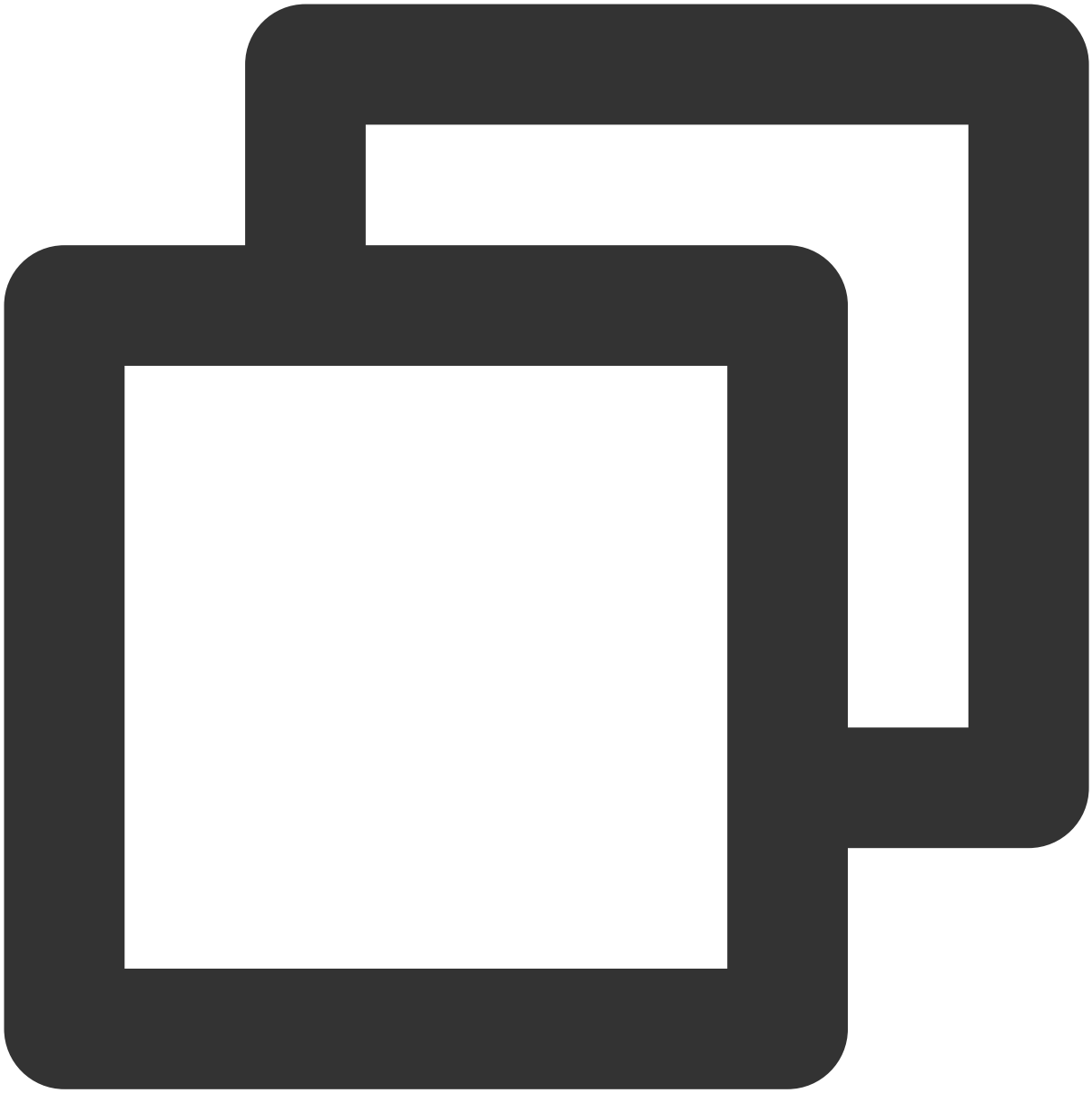
```
ServiceConfig:
  Method: GET
  Path: /test
  Url: 'http://test.com'
ServiceType: HTTP
```

Backend connection to VPC resources



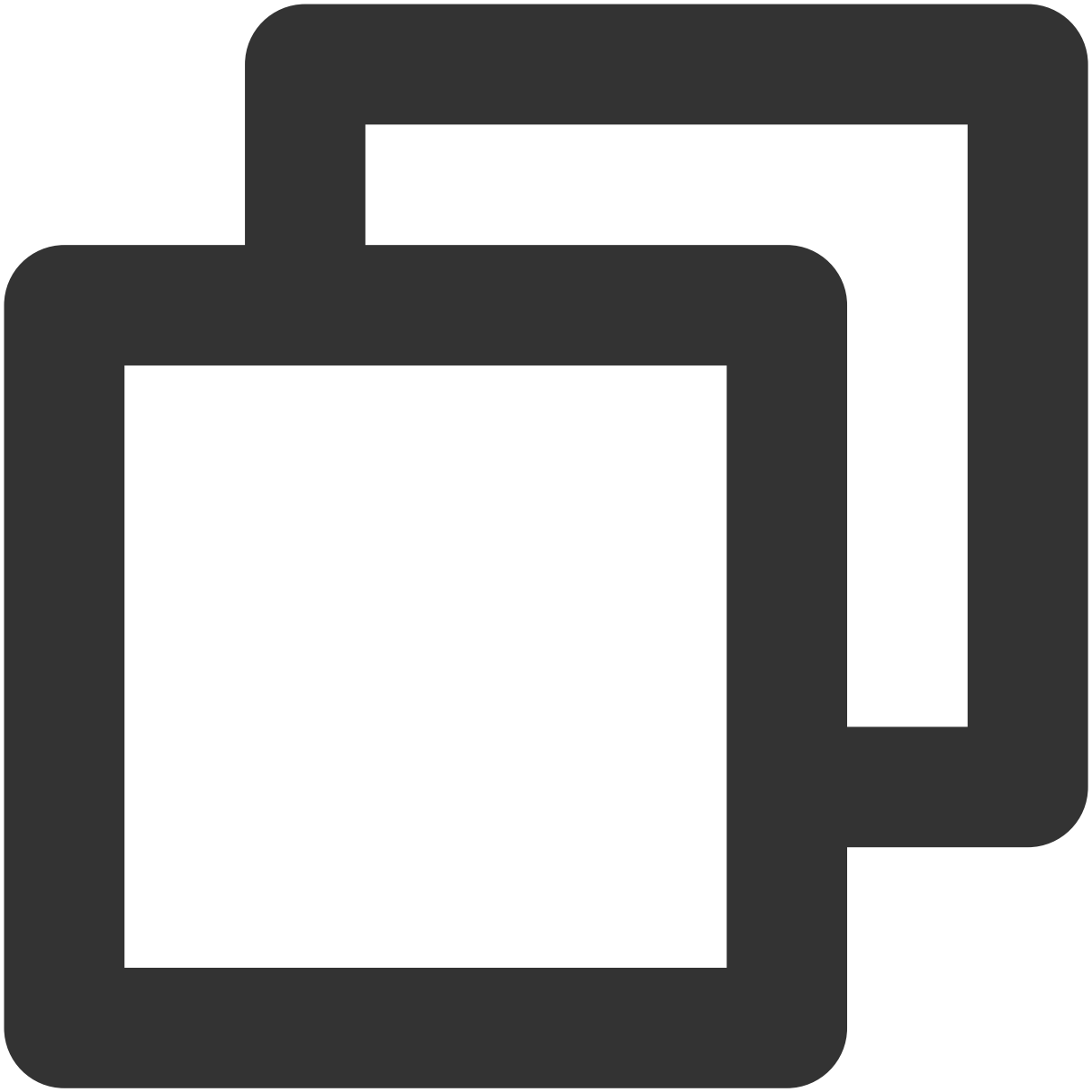
```
ServiceConfig:
  Method: GET
  Path: /test
  Url: 'http://test.com'
  UniqVpcId: vpc-xxxxx
  Product: clb
ServiceType: HTTP
```

Backend connection to SCF



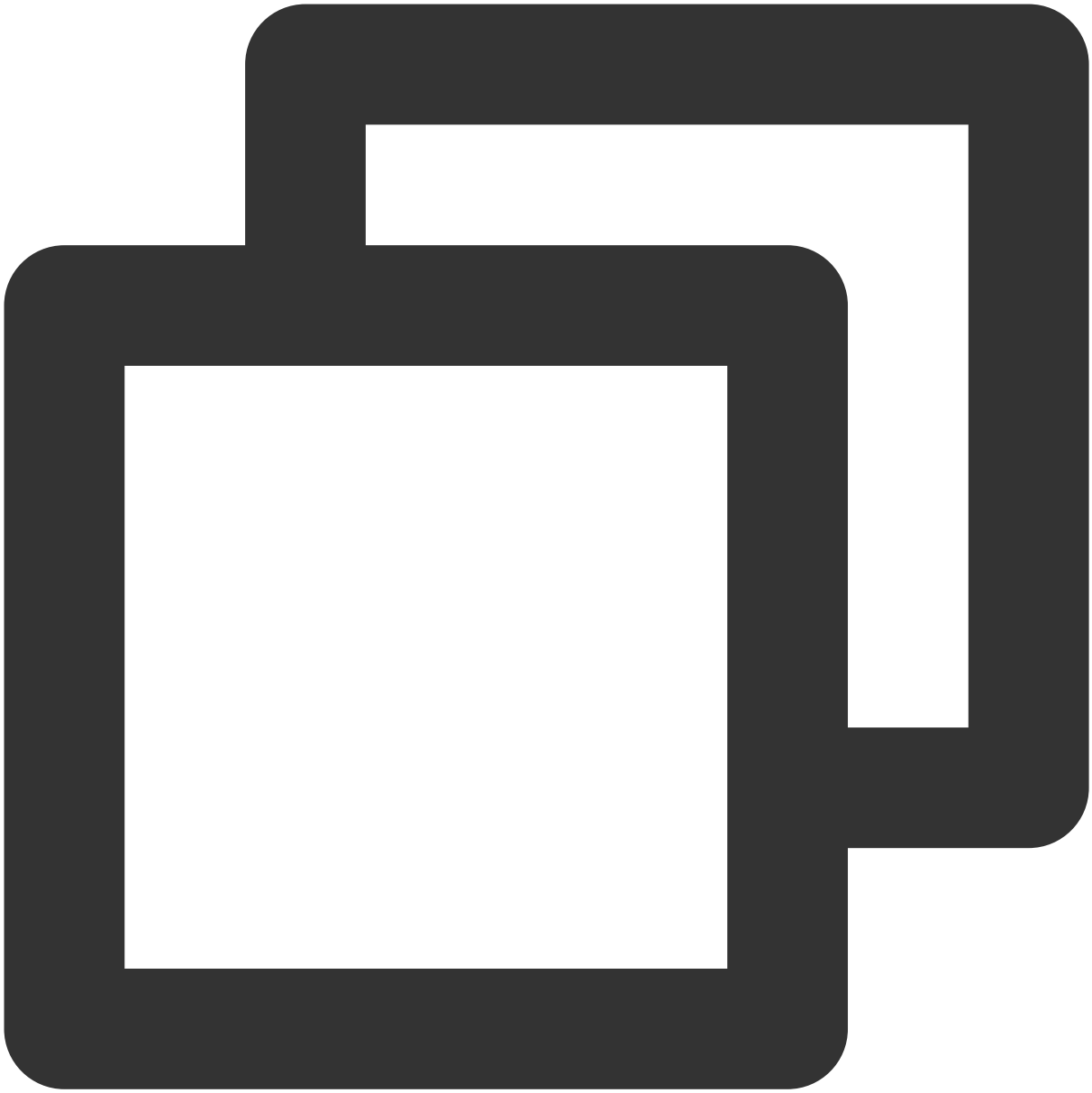
```
ServiceScfFunctionName: scftest
ServiceScfFunctionNamespace: mynamespace
ServiceScfFunctionQualifier: $DEFAULT
ServiceScfFunctionType: EVENT
ServiceScfIsIntegratedResponse: false
ServiceType: SCF
```

Backend connection to Mock



```
ServiceMockReturnMessage: hello mock from strategy
ServiceType: MOCK
```

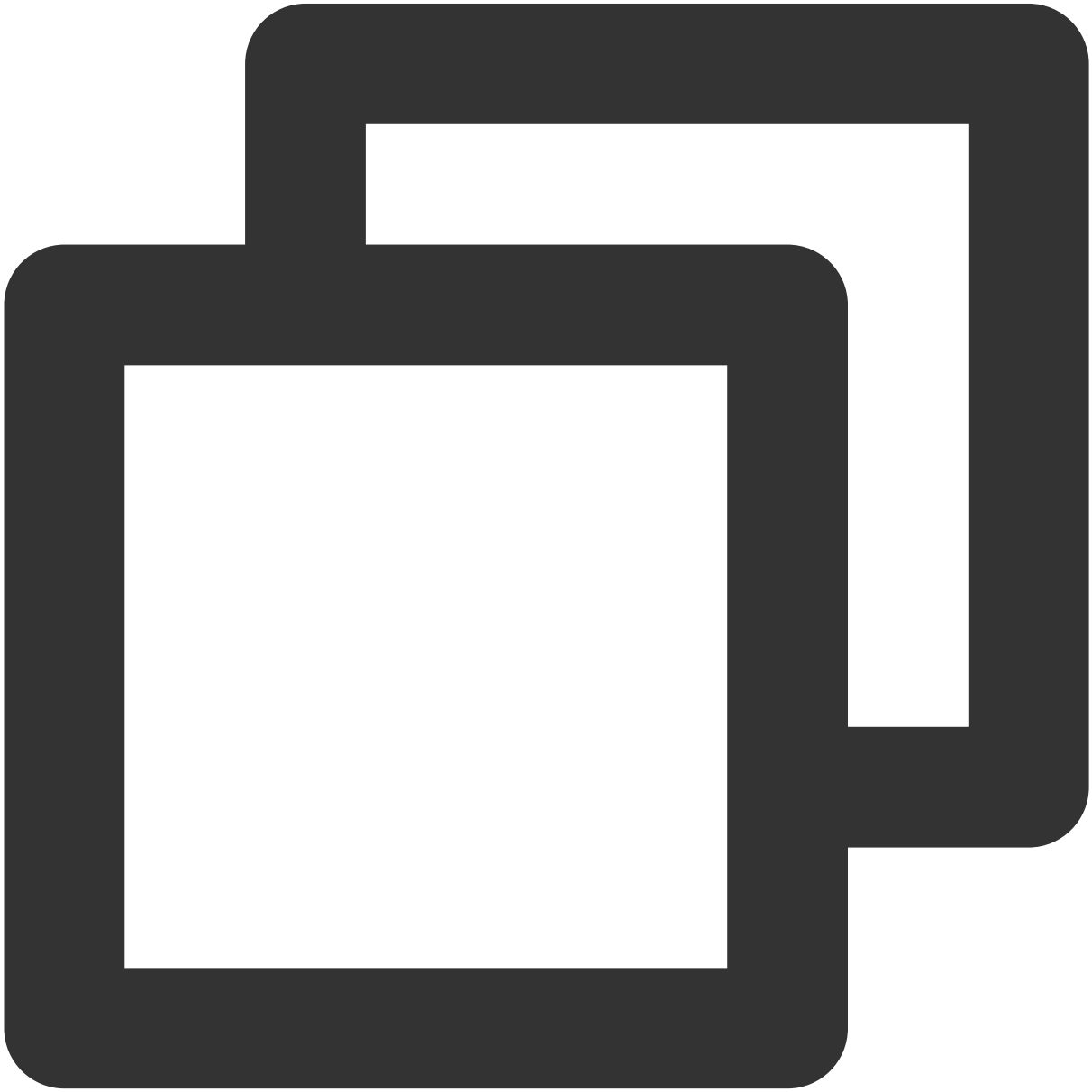
Backend connection to TSF



```
X-MicroService-Name: consumer-demo
X-NameSpace-Code: mytsf
MicroServices:
  - ClusterId: cls-xxxxxx
    MicroServiceName: tsf-demo
    NamespaceId: namespace-xxxxxx
ServiceConfig:
  Method: ANY
  Path: /xxxx
  Url: ''
ServiceTsfHealthCheckConf:
```

```
  IsHealthCheck: true
ServiceTsfLoadBalanceConf:
  IsLoadBalance: true
  Method: RoundRobinRule
  SessionStickRequired: false
ServiceType: TSF
```

pluginData



```
[
    {
        "strategy_name":"route-to-http", // Policy name, which can contain up to 50
        "strategy_weight":2, // Priority for policy match. You can enter a positive
        "condition":"query.age<30 and query.need_verify=false or query.level>3", //
        "backend_type":"HTTP", // Type of the backend for routing and forwarding. V
        "backend_config":{  // Backend configuration
            "ServiceConfig":{
                "Method":"GET",
                "Path":"/v1/bpi/currentprice.json",
                "Url":"https://api.coindesk.com"
            },
            "ServiceType":"HTTP"
        }
    }
]
```

Notes

If a request cannot match a routing policy configured in the conditional routing plugin bound to an API, it will be forwarded to the default API backend.

Cache

Last updated：2023-12-22 10:01:23

Overview

You can configure the cache plugin to enable API Gateway to store backend responses. When receiving requests with the same parameters, API Gateway can directly return the cached responses with no need to forward the requests to the backend service. This reduces the load on the backend, shortens the latency, and makes your business smoother.

Directions

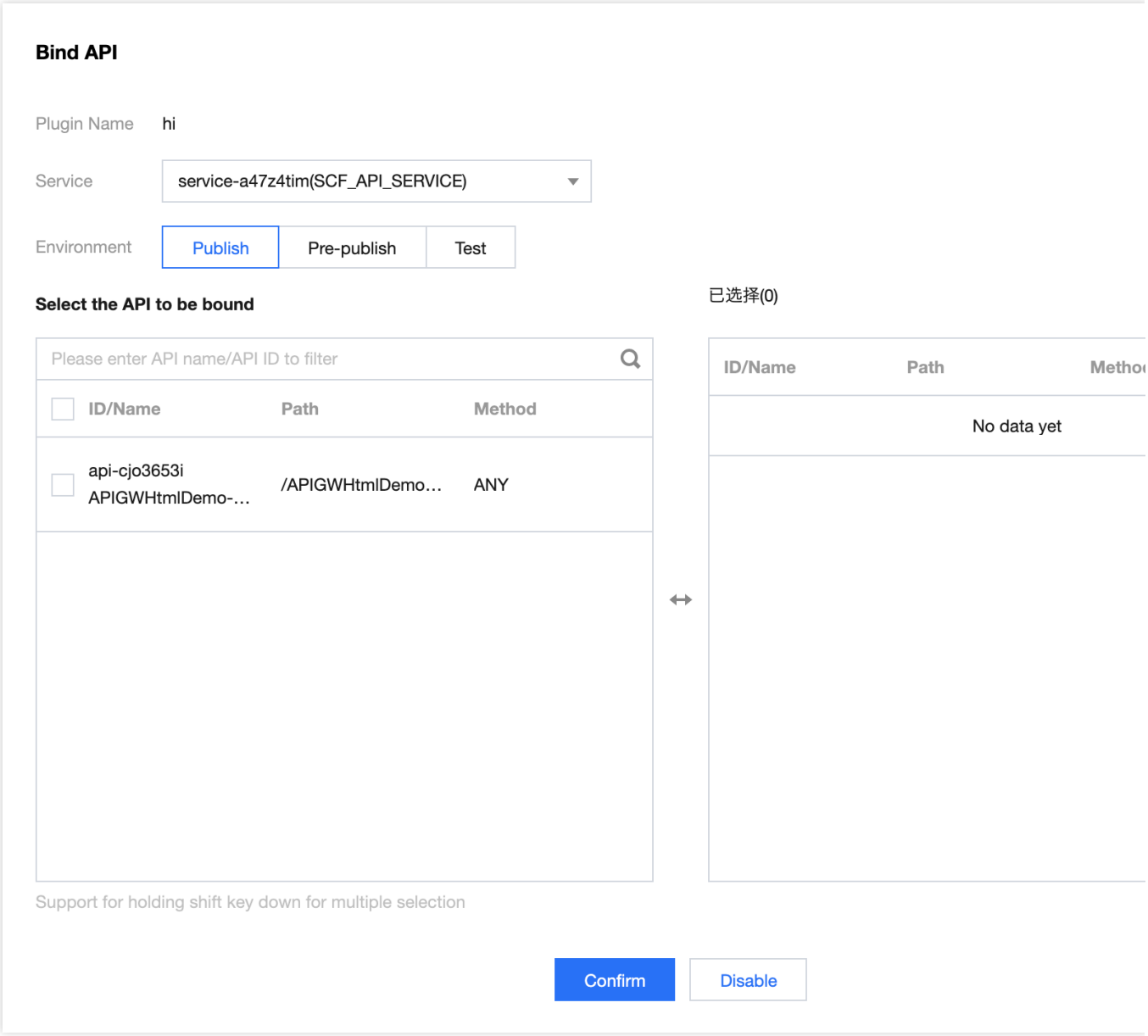
Step 1. Create a plugin

1. Log in to the API Gateway console
2. On the left sidebar, click **Plugin** > **System Plugin** to enter the system plugin list page.
3. Click **Create** in the top-left corner of the page and select **Cache** as the plugin type to create a cache plugin. The plugin configuration items are as detailed below:

Parameter	Required	Description
Cache parameter	Yes	The cached content can be distinguished by the parameter or the combination of multiple parameters. Supported parameter locations include Header, Path, and Query.
Request method	Yes	You can select multiple methods. Valid values: GET, POST, PUT, DELETE, HEAD.
Cache status code	Yes	Only responses with a status code set in the cache plugin will be cached. You can separate multiple status codes with commas.
Cache-Control	Yes	It specifies whether to use the `Cache-Control` request header to affect the cache policy. It is disabled by default.
Cache duration	Yes	It is the cache validity period, which is a positive integer between 0 and 3600.

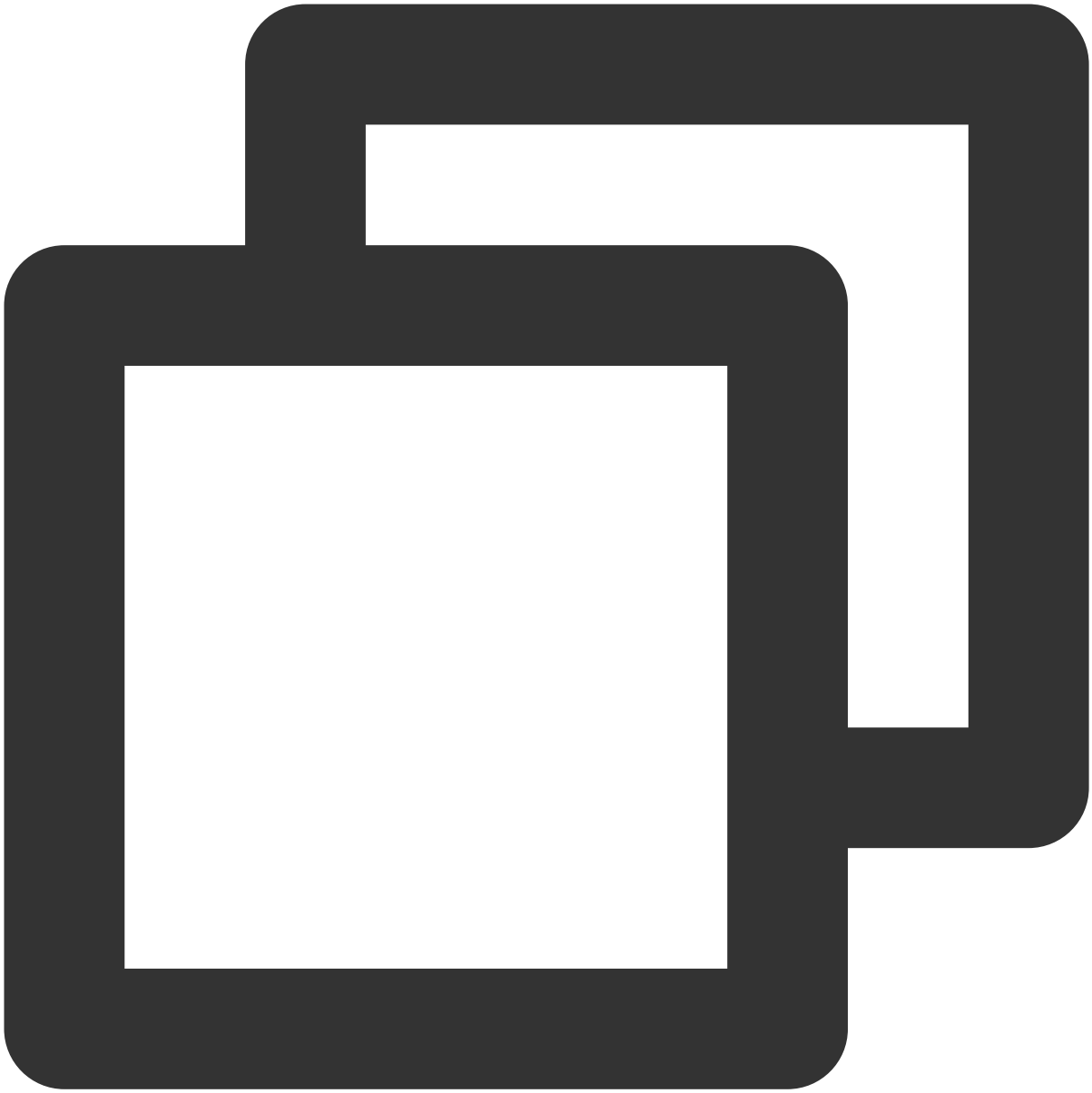
Step 2. Bind an API and make the plugin effective

1. Select the just created plugin in the list and click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the API to which the plugin needs to be bound.



3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

PluginData



```
{
    "cache_key_params": [{ // Parameter for distinguishing between cached responses
        "parameter": "param1",
        "position": "header"
    }, {
        "parameter": "param2",
        "position": "query"
    }, {
        "parameter": "param3",
        "position": "path"
    }],
```

```
    "cacheable_methods": ["GET", "POST"], // HTTP methods of requests whose respons
    "cacheable_response_codes": [200, 301, 404], // HTTP return codes of responses
    "cache_control": false, // Whether to enable the `Cache-Control` syntax in the
    "ttl": 300 // Custom cache validity period, which will take effect if `cache_co
}
```

Notes

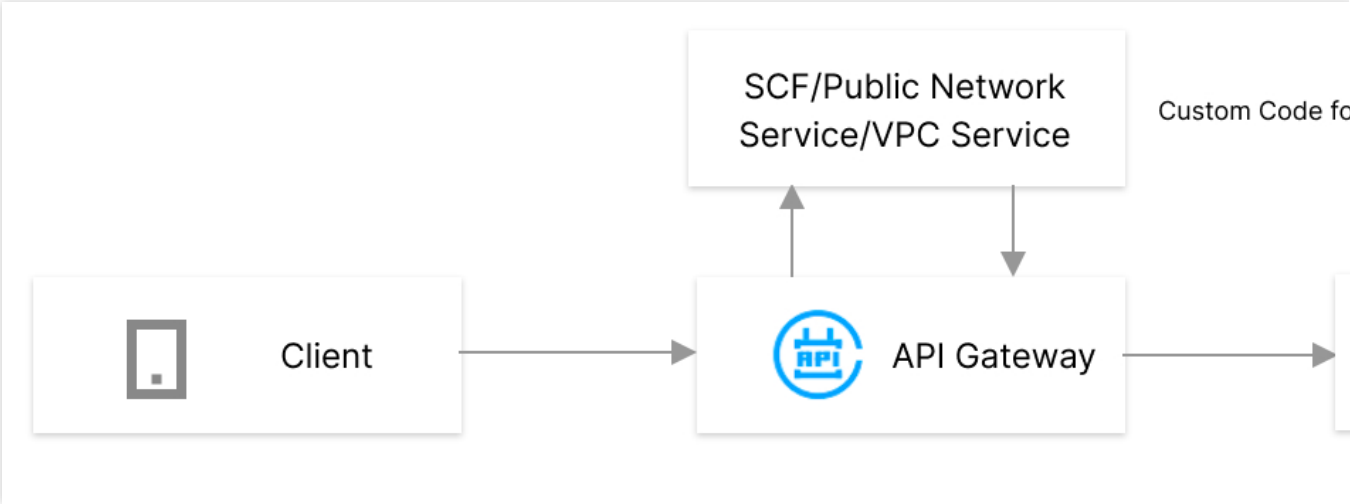
The values are case-insensitive for parameter verification and cache hitting conducted by API Gateway.
For a shared instance, the maximum cache capacity in each region for each user is 5 MB. The maximum of total cache capacity of each shared instance is 1 GB.
After `Cache-Control` is enabled, the gateway will process the cache according to the convention in the `Cache-Control` request/response header. In this case, if the gateway cannot get the `Cache-Control` header, the response will be cached by default, and the `cache duration` field configured in the plugin will be used as the cache validity period.

Custom Verification

Last updated：2023-12-22 10:01:35

Overview

If the verification and authentication method provided by API Gateway cannot meet your requirements, you can use the custom authentication plugin with custom code.
The custom verification plugin applies during the request process. API Gateway will forward the request to the verification function after receiving it from the client. You can deploy the verification function in SCF, on the public network, or in a VPC. Then, the request will be forwarded to the service backend only if it passes the verification; otherwise, the request will be denied.



Prerequisites

For verification services deployed in SCF, you need to enable the SCF service.

Directions

Step 1. Create a verification function

For verification functions deployed on the public network or in a VPC, you can skip this step.
1. Log in to the SCF console.

2. Click **Functions** on the left sidebar to enter the function list page.
3. Click **Create** at the upper-left corner of the page to create a verification SCF.

Step 2. Create a custom verification plugin

1. Log in to the API Gateway console.
2. On the left sidebar, click **Plugin** > **Custom Plugin** to enter the custom plugin list page.
3. Click **Create** in the top-left corner of the page to create a custom verification plugin.

For verification services deployed in SCF, you need to enter the following data when creating the custom verification plugin:

Parameter	Required	Description
Function	Yes	Select the namespace, name, and version of the verification function.
Backend timeout	Yes	Set the backend timeout that API Gateway forwards the request to the verification function. The maximum time limit is 30 minutes. When no response is returned before the timeout after API Gateway calls the function, API Gateway will end the call and return an error message.
Whether to send the Body	Yes	When the value is `Yes`, the Header, Body, and Query requested by the client will be sent to the function. When the value is `No`, the requested Body will not be sent.
Verification Parameters	No	Set the request parameters for verification. When **Cache Period** is not `0`, this parameter must be set. When caching is enabled, the verification result will be queried with this parameter as the search criterion.
Cache Period	Yes	Set the cache validity period for the verification result. `0` indicates that caching is not enabled. The cache validity period can be up to 3,600 seconds.

For verification services deployed on the public network, you need to enter the following data when creating the custom verification plugin:

Parameter	Required	Description
Request method	Yes	Request method of the custom verification function, which can be GET, POST, PUT, DELETE, HEAD, and ANY.
Public network service	Yes	Access address of the custom verification service, which can be an HTTP or HTTPS address.
Path match mode	Yes	It can be backend path or full path match.

		Backend path match: The configured path is used to request the service. Full path match: The overlapping part is used to request the service. For example, if the configured API path is `/a/` and the request path is `/a/b`, then the path transferred to the service will be `/b` after full path match is enabled.

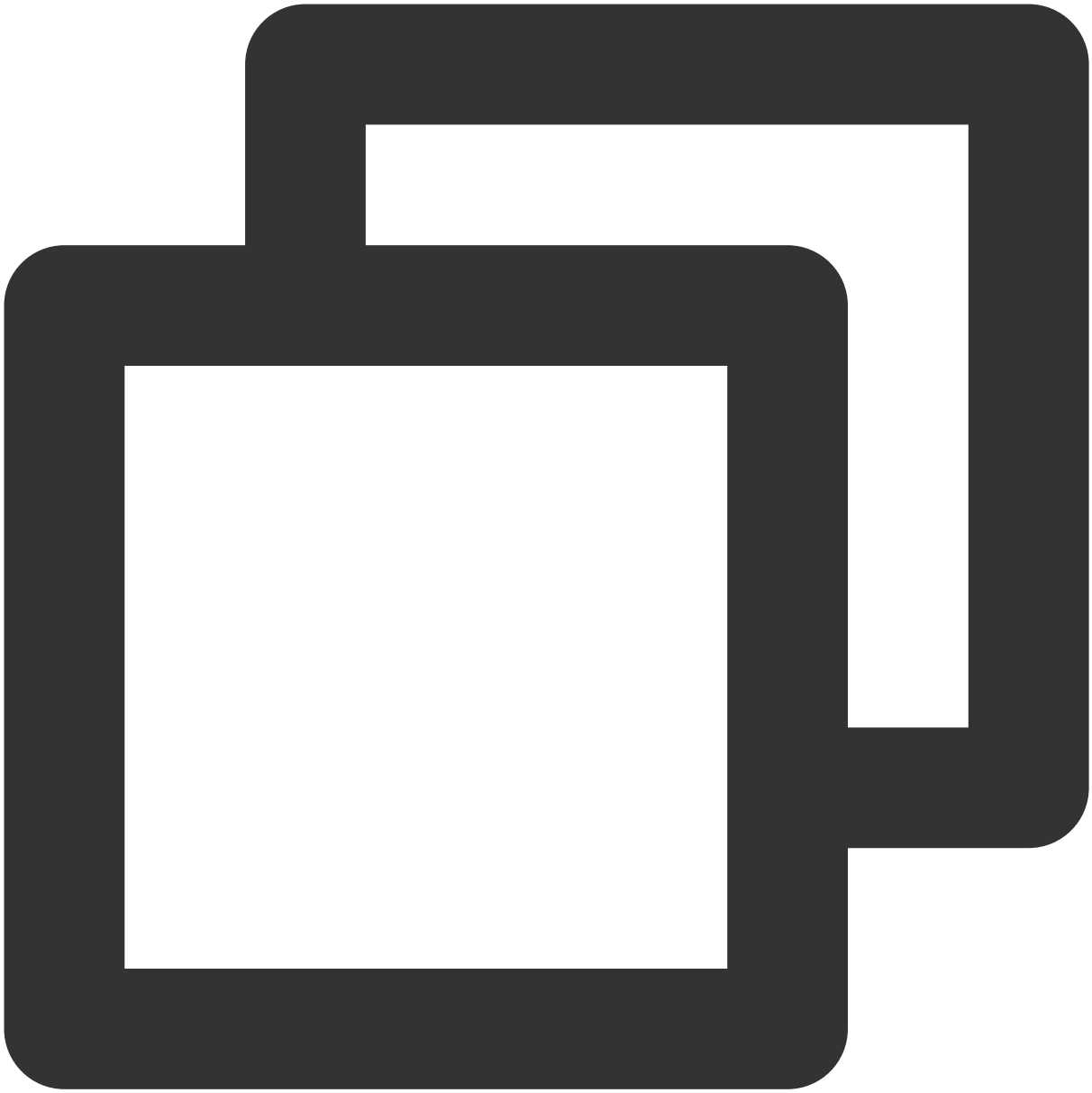
For verification services deployed in a VPC, you need to enter the following data when creating the custom verification plugin:

Parameter	Required	Description
VPC	Yes	Select the VPC of the verification service.
Request method	Yes	Request method of the custom verification function, which can be GET, POST, PUT, DELETE, HEAD, and ANY.
Backend address	Yes	Access address of the custom verification service, which can be an HTTP or HTTPS address.

Step 3. Bind the API

1. Select the just created plugin in the list and click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the target API.
3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

pluginData



```
{
    "cache_time":10,   // Verification result caching duration in seconds. Value ra
    "endpoint_timeout":15, // Backend timeout period in seconds. Value range: 0-60
    "func_name":"test_name", // Custom SCF name
    "func_namespace":"test_namespace", // Custom SCF namespace
    "func_qualifier":"$LATEST", // Custom SCF version
    "is_send_body":true, // Whether to send the request Body to the SCF
    "header_auth_parameters":[ // Verification parameter in Header location. The pl
        "Header1"
    ],
    "query_auth_parameters":[ // Verification parameter in Query location. The plug
```

```
        "Query1"
    ],
    "user_id":1253970226, // appid
    "version":"2021-12-26 17:17:49" // Plugin version in the format of `yyyy-MM-dd
}
```

Notes

When you enable caching and configure the verification parameter, the API Gateway will conduct parameter verification. If the request does not transfer the verification parameter, the API Gateway will report an error message "xxx parameter is missing". The values are case insensitive for parameter verification and cache hitting conducted by the API Gateway.

Binding a custom plugin to the API means creating a trigger for the SCF to trigger the API. Deleting the trigger on the SCF side means unbinding the plugin from the API.

For now, the custom verification plugin only supports event-triggered function and does not support HTTP-triggered function.

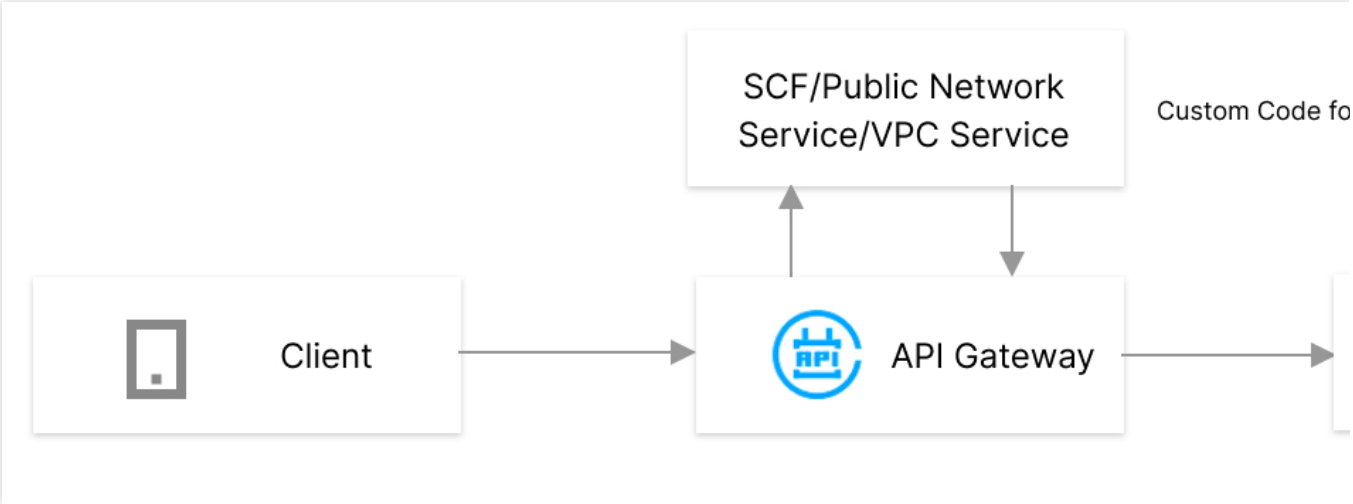
The custom verification plugin can coexist with the verification method provided by the API Gateway. The latter takes priority. We recommend that you set the API where the custom verification plugin bound to "free from verification".

Custom Request Body

Last updated：2023-12-22 10:01:48

Overview

The request body sent to service backend by the Client contains many fields. If you want to modify the content of the request body, you can do so via custom request body plugin.
The custom request body plugin applies during the request process. The request body rewriting service can be deployed in SCF, on the public network, or in a VPC. API Gateway will forward the request content to the request body rewriting service after receiving it from the client. The rewriting service will send the content of the request body to API Gateway after modifying it. Then, API Gateway will forward the modified request body to the service backend.



Prerequisites

For verification services deployed in SCF, you need to enable the SCF service.

Directions

Step 1. Create a function to modify the request body

For verification functions deployed on the public network or in a VPC, you can skip this step.
1. Log in to the SCF console.

2. Click **Functions** on the left sidebar to enter the function list page.
3. Click **Create** at the upper-left corner of the page to create a SCF that is used to modify the request body.

Step 2. Create a custom request body plugin

1. Log in to the API Gateway console.
2. On the left sidebar, click **Plugin** > **Custom Plugin** to enter the custom plugin list page.
3. Click **Create** in the top-left corner of the page to create a custom request body plugin.

For verification services deployed in SCF, you need to enter the following data when creating the custom request body plugin:

Parameter	Required	Description
Function	Yes	Select the namespace, name, and version of the function that is used to modify the request body.
Backend timeout	Yes	Set the backend timeout that API Gateway forwards the request to the function that is used to modify the request body. The maximum time limit is 30 minutes. When no response is returned before the timeout after API Gateway calls the function, API Gateway will end the call and return an error message.
Custom content	Yes	Set the request content sent by API Gateway to the function used to modify the request body. You can select Header, Body, and Query. The request content not selected will not be modified and will be forwarded to the service backend as is.
Base64 Encoding	Yes	Specify whether to forward the request content to the function after applying Base64 encoding. Generally, it is applicable to binary content.

For verification services deployed on the public network, you need to enter the following data when creating the custom verification plugin:

Parameter	Required	Description
Request method	Yes	Request method of the custom request body function, which can be GET, POST, PUT, DELETE, HEAD, and ANY.
Public network service	Yes	Access address of the request body service, which can be an HTTP or HTTPS address.
Path match mode	Yes	It can be backend path or full path match. Backend path match: The configured path is used to request the service. Full path match: The overlapping part is used to request the service. For example, if the configured API path is `/a/` and the request path is `/a/b`, then the path transferred to the service will be `/b` after full path match is enabled.

For verification services deployed in a VPC, you need to enter the following data when creating the custom verification plugin:

Parameter	Required	Description
VPC	Yes	Select the VPC of the custom request rewriting service.
Request method	Yes	Request method of the custom request rewriting service, which can be GET, POST, PUT, DELETE, HEAD, and ANY.
Backend address	Yes	Access address of the custom request body service, which can be an HTTP or HTTPS address.

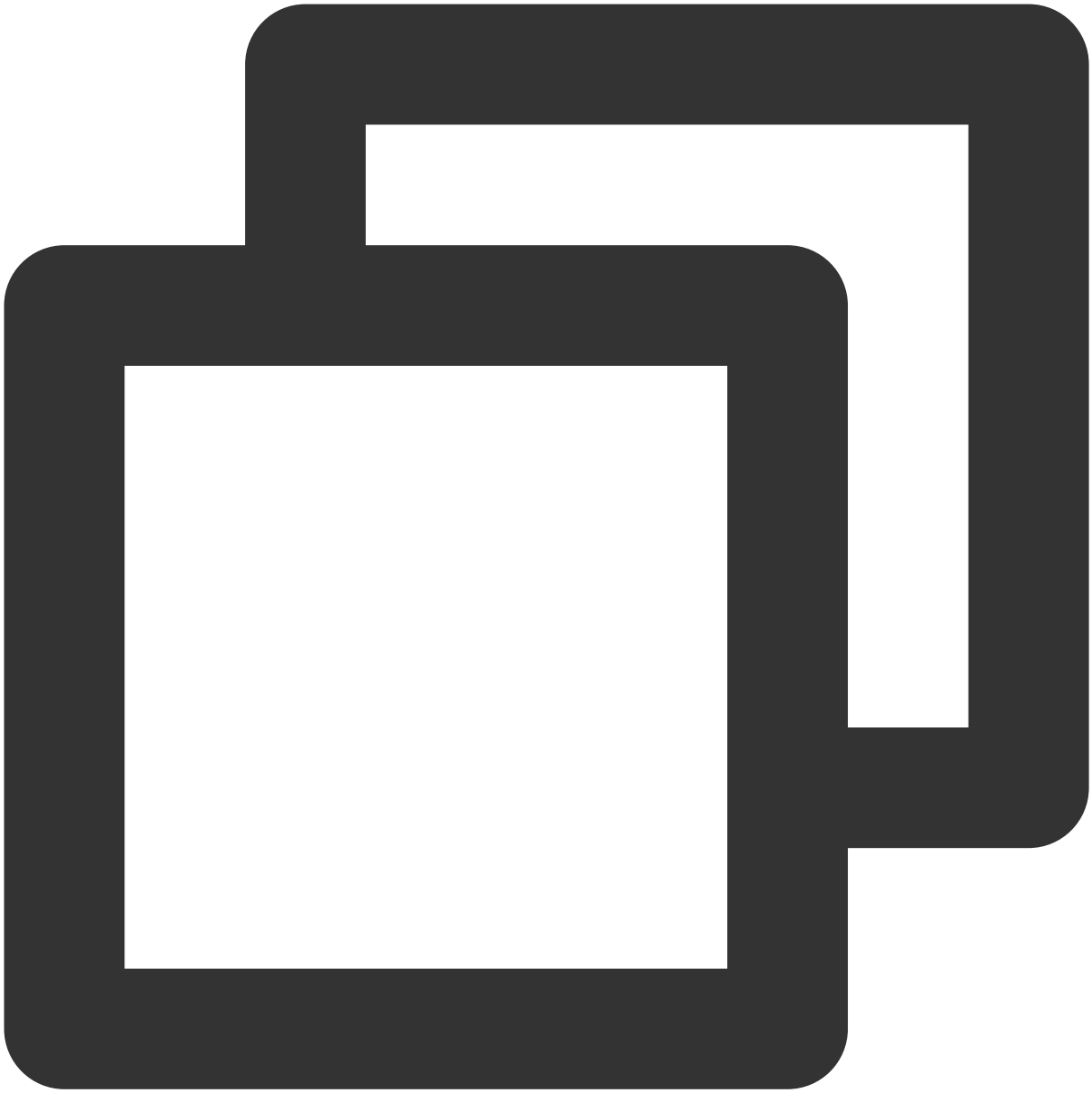
Step 3. Bind the API

1. Select the plugin created in step 2 from the list. Click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the target API.
3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

How to Write Custom Request Body Function

Returned value definition

The custom request body plugin of API Gateway needs to receive the response in a certain format returned by the custom request body SCF. The specific format is as follows:



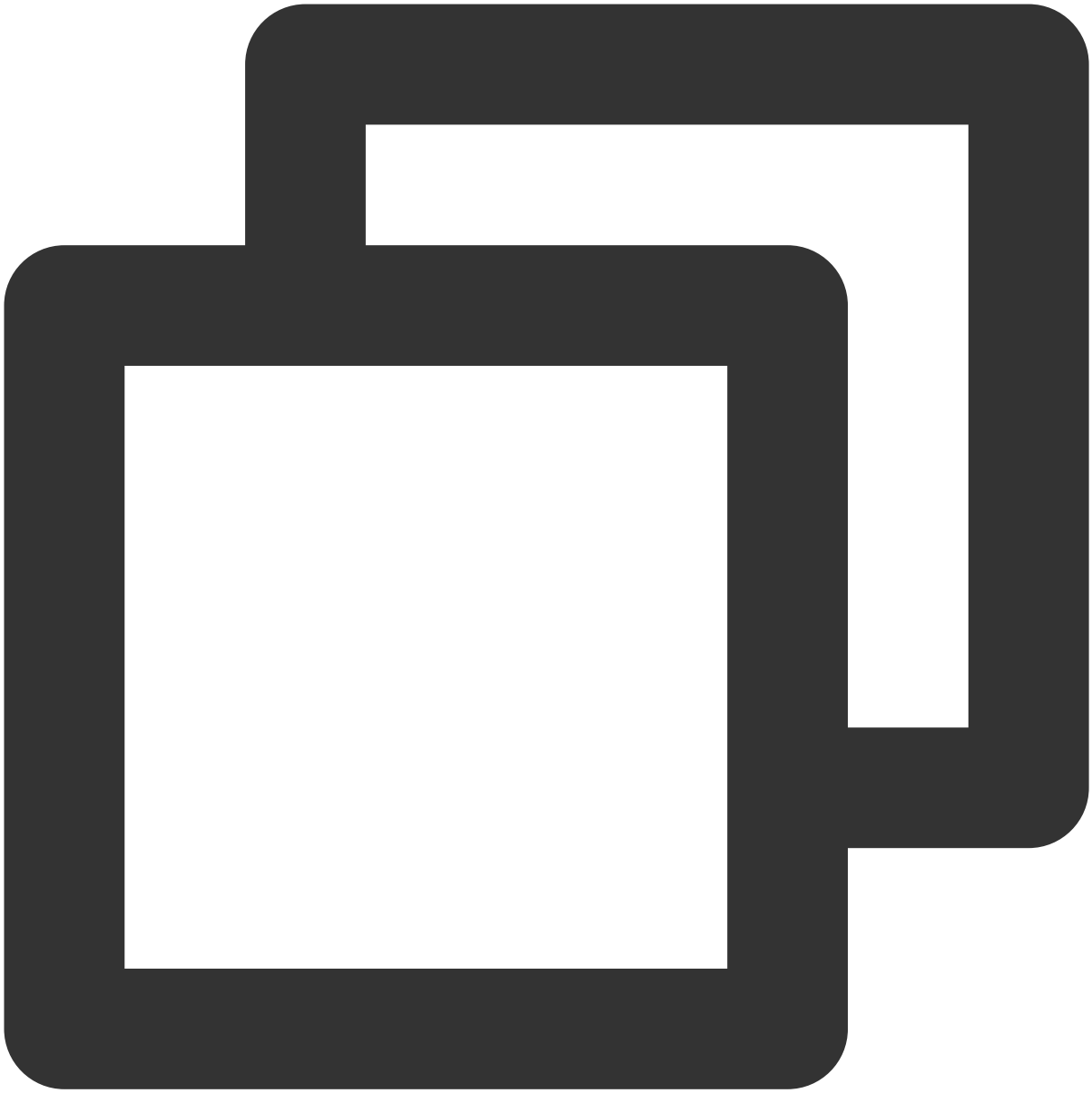
```
{
  "replace_headers":{
    "header1":"header1-value",
    "header2":"header2-value"
  },
  "remove_headers":[
    "header3",
    "header4"
  ],
  "replace_body":"hello",
  "replace_querys":{
```

```
    "query1":"query1-value",
    "query2":"query2-value"
  },
  "remove_querys":[
    "query3",
    "query4"
  ]
}
```

Demo for Python

For more information on how to write the function that is used to modify the request body in Python, see Custom Request Body Demo for Python.

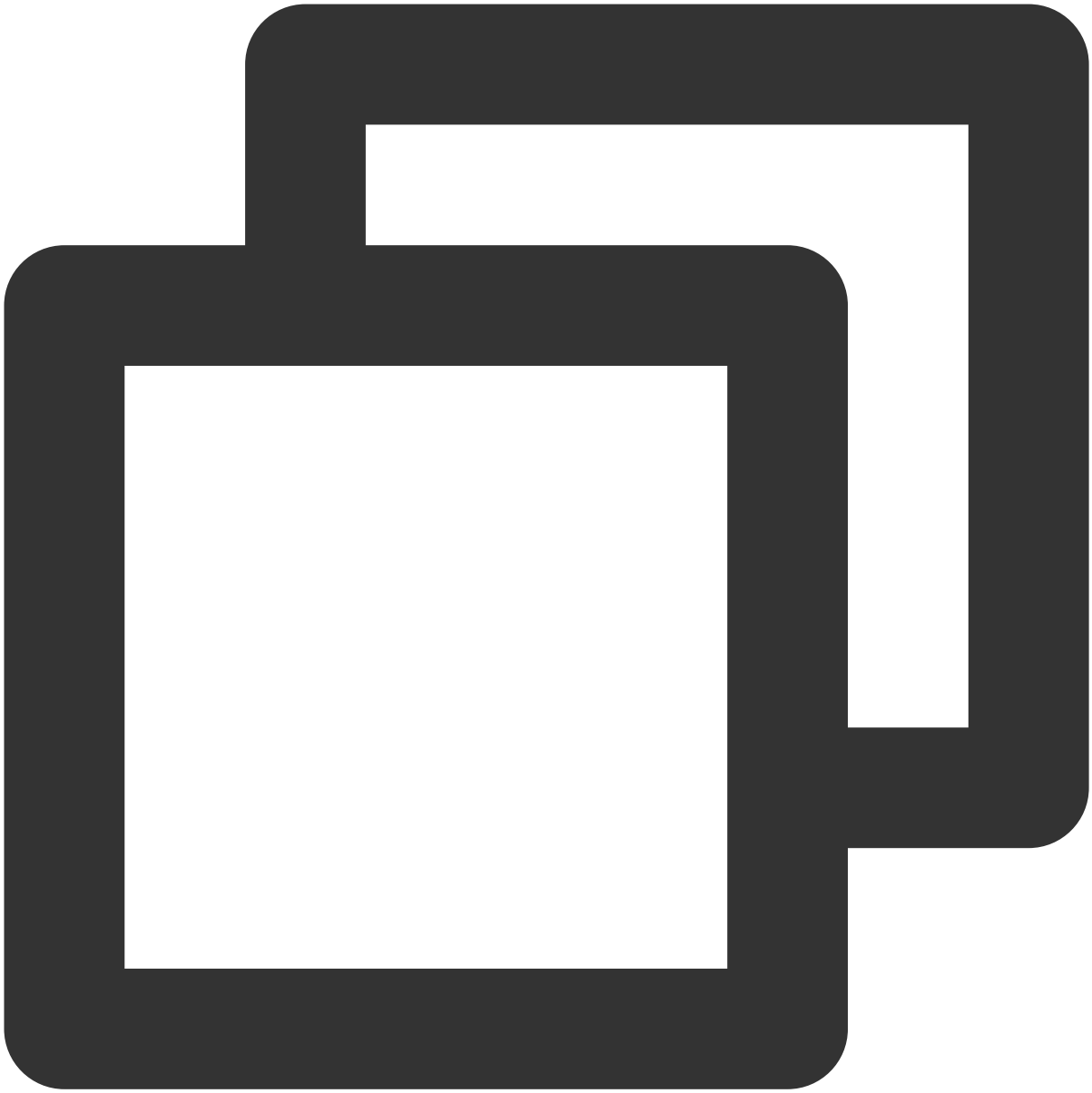
Demo for Java



```
package com.example.demo;

import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.qcloud.services.scf.runtime.events.APIGatewayProxyRequestEvent;

public class Demo {

    public String mainHandler(APIGatewayProxyRequestEvent request) {
        System.out.println("helloworld");
        System.out.println(request.getHttpMethod());
```

```
        JsonObject resp = new JsonObject();
        headerHandler(request, resp);
        headerQuery(request, resp);
        headerBody(request, resp);
        return resp.toString();
    }

    private void headerHandler(APIGatewayProxyRequestEvent request, JsonObject resp
        JsonObject replace_headers = new JsonObject();
        JsonArray remove_headers = new JsonArray();
        // Sample: Replace or add `header1` and `header2`
        replace_headers.addProperty("header1", "header1-value");
        replace_headers.addProperty("header2", "header2-value");

        // Sample: Delete `query3`
        remove_headers.add("header3");
        resp.add("replace_headers", replace_headers);
        resp.add("remove_headers", remove_headers);
    }

    private void headerQuery(APIGatewayProxyRequestEvent request, JsonObject resp)
        JsonObject replace_querys = new JsonObject();
        JsonArray remove_querys = new JsonArray();

        // Sample: Replace or add `query1` and `query2`
        replace_querys.addProperty("query1", "query1-value");
        replace_querys.addProperty("query2", "query2-value");

        // Sample: Delete `query3`
        remove_querys.add("query3");
        resp.add("replace_querys", replace_querys);
        resp.add("remove_querys", remove_querys);
    }

    private void headerBody(APIGatewayProxyRequestEvent request, JsonObject resp) {
        resp.addProperty("replace_body", "{'name':'Yagr'}");
    }
}
```

PluginData



```
{
    "endpoint_timeout":15, // Backend timeout period in seconds. Value range: 0-60
    "func_name":"test_name", // Custom SCF name
    "func_namespace":"test_namespace", // Custom SCF namespace
    "func_qualifier":"$LATEST", // Custom SCF version
    "is_base64_encoded":true, // Whether to forward the request content to the SCF
    "is_send_req_body":true, // Whether to send the request Body content to the SCF
    "is_send_req_headers":true, // Whether to send the request Header content to th
    "is_send_req_querys":true, // Whether to send the request Query content to the
    "user_id":1253970226 // appid
}
```

Notes

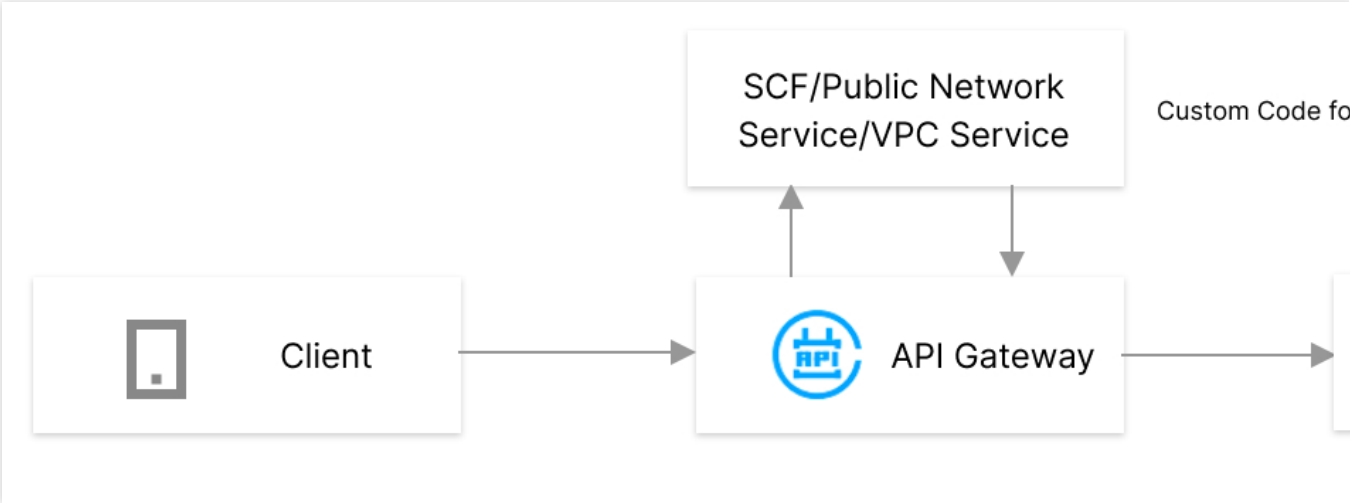
Binding a custom plugin to the API means creating a trigger for the SCF to trigger the API. Deleting the trigger on the SCF side means unbinding the plugin from the API.
For now, the custom request body plugin only supports event-triggered function and does not support HTTP-triggered function.

Custom Response Body

Last updated：2023-12-22 10:01:59

Scenarios

The response body sent to the Client by API Gateway contains many fields. If you want to modify the content of the response body, you can do so via custom response body plugin.
The custom response body plugin applies during the response process, and the response content rewriting service can be deployed in SCF, on the public network, or in a VPC. The service backend will send the response body to API Gateway after processing the request message. API Gateway will forward the response content to the response body modification service after receiving it. The modification service will send the modified content of the response body to API Gateway after modifying it. Then, API Gateway will forward the modified response body to the service backend.



Prerequisites

Any one of the following:
1. Activate SCF.
2. Connect your service to the public network.
3. Activate VPC.

Directions

We take SCF as an example in this document.

Step 1. Create a function to modify the response body

1. Log in to the SCF console.
2. Click **Function Service** in the left sidebar to open the function list page.
3. Click **Create** in the top-left corner of the page to create a function that is used to modify the response body.

Step 2. Create a custom response body plugin

1. Log in to the API Gateway console.
2. On the left sidebar, click **Plugin** > **Custom Plugin** to enter the custom plugin list page.
3. Click **Create** in the top-left corner of the page to create a custom response body plugin.

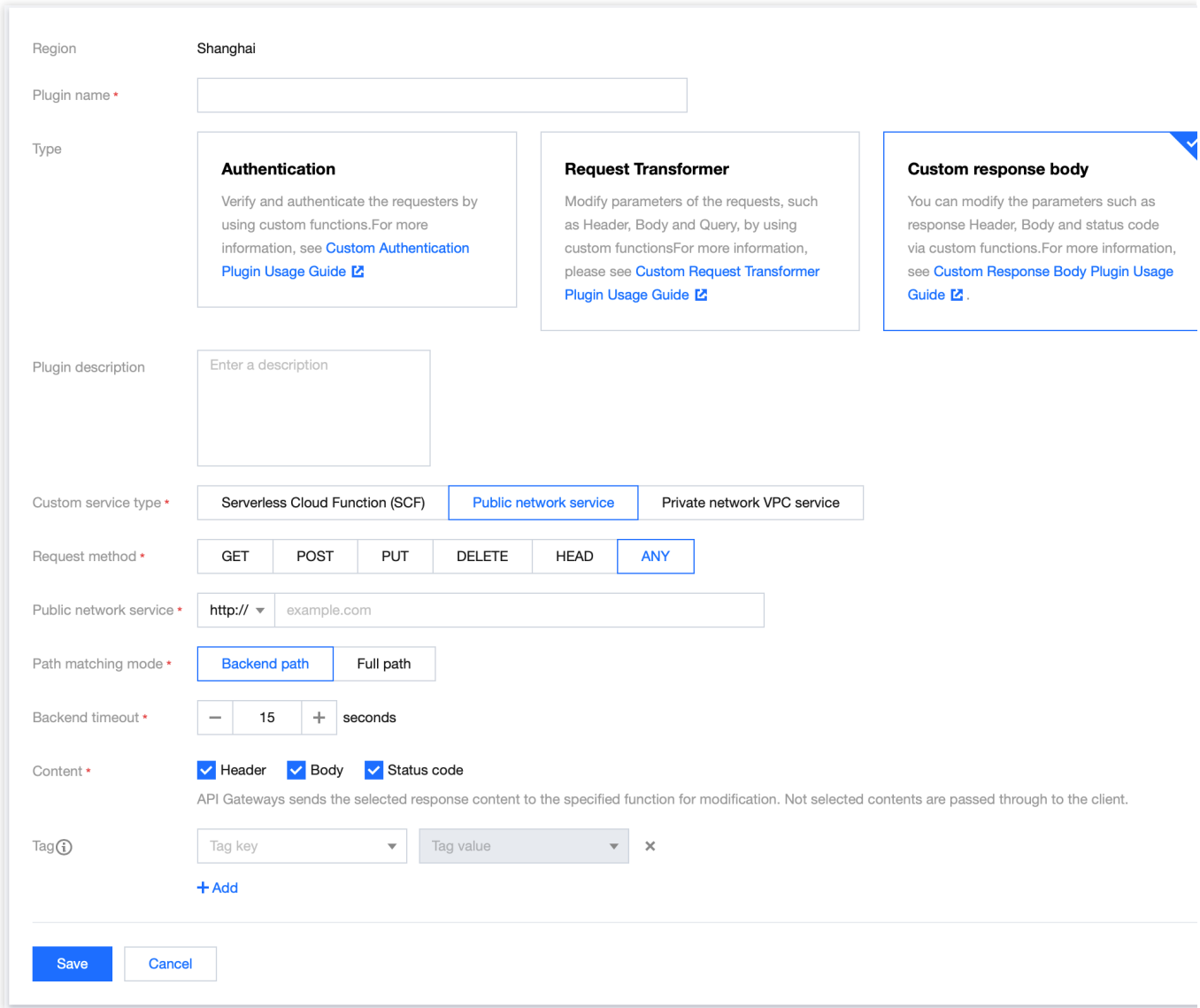
For verification services deployed in SCF, you need to enter the following data when creating the custom response plugin:

Parameter	Required	Description
Function	Yes	Select the namespace, name, and version of the response body modification function.
Backend timeout	Yes	Set the backend timeout that API Gateway forwards the request to the function that is used to modify the response body. The maximum time limit is 30 minutes. When no response is returned before the timeout after API Gateway calls the function, API Gateway will end the call and return an error message.
Custom content	Yes	Set the response content sent by API Gateway to the function used to modify the response body. You can select Header, Body, and Query. The response content not selected will not be modified and will be forwarded to the client as is.
Base64 Encoding	Yes	Specify whether to Base64-encode the response content to be forwarded by the service backend to the function. Generally, it is applicable to binary content.

For verification services deployed on the public network, you need to enter the following data when creating the custom response plugin:

Parameter	Required	Description
Request method	Yes	Request method of the custom response body function, which can be GET, POST, PUT, DELETE, HEAD, and ANY.
Public network service	Yes	Access address of the custom response body rewriting service, which can be an HTTP or HTTPS address.

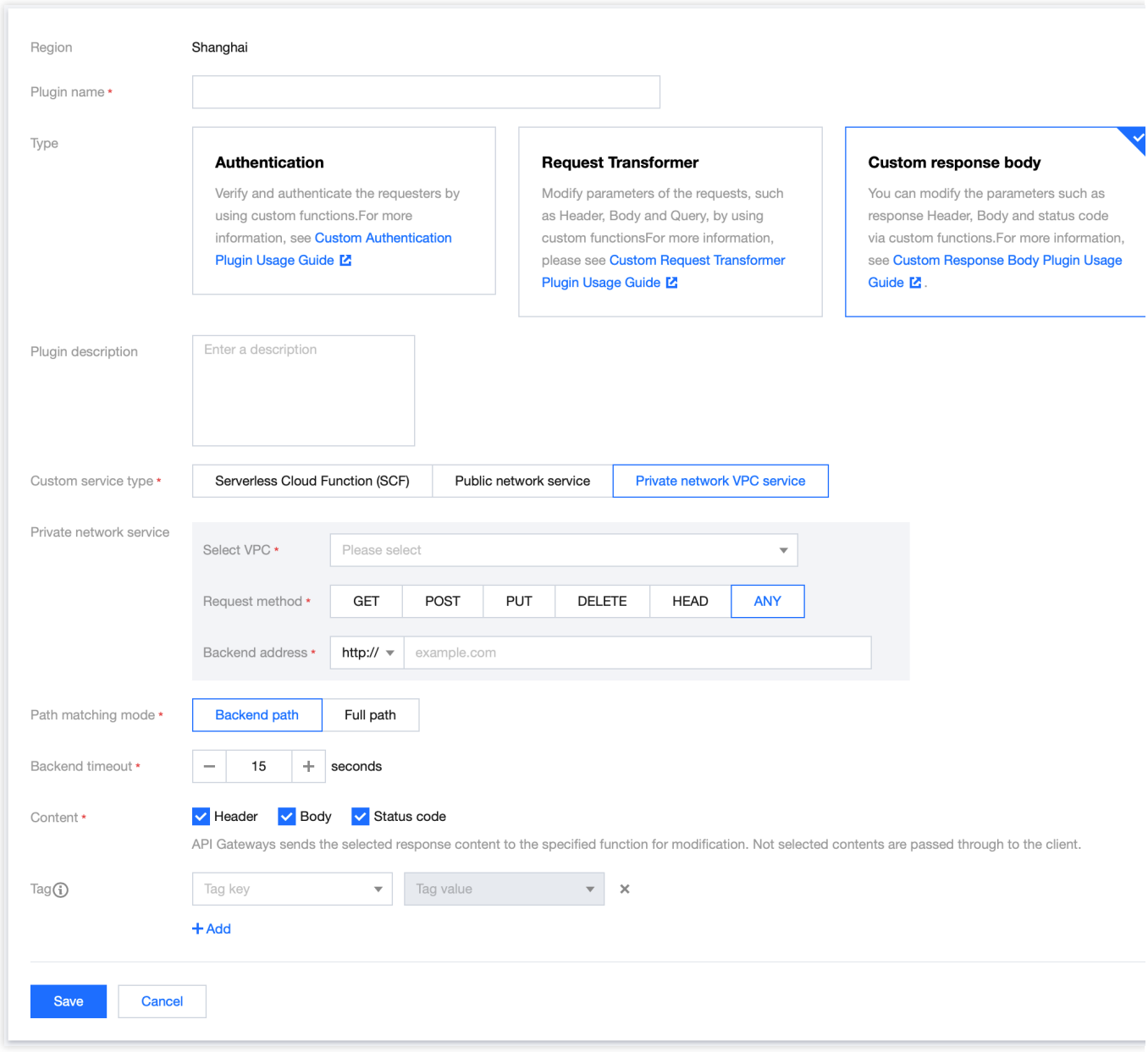
Path match mode	Yes	It can be backend path or full path match. Backend path match: The configured path is used to request the service. Full path match: The overlapping part is used to request the service. For example, if the configured API path is `/a/` and the request path is `/a/b`, then the path transferred to the service will be `/b` after full path match is enabled.



For verification services deployed in a VPC, you need to enter the following data when creating the custom response plugin:

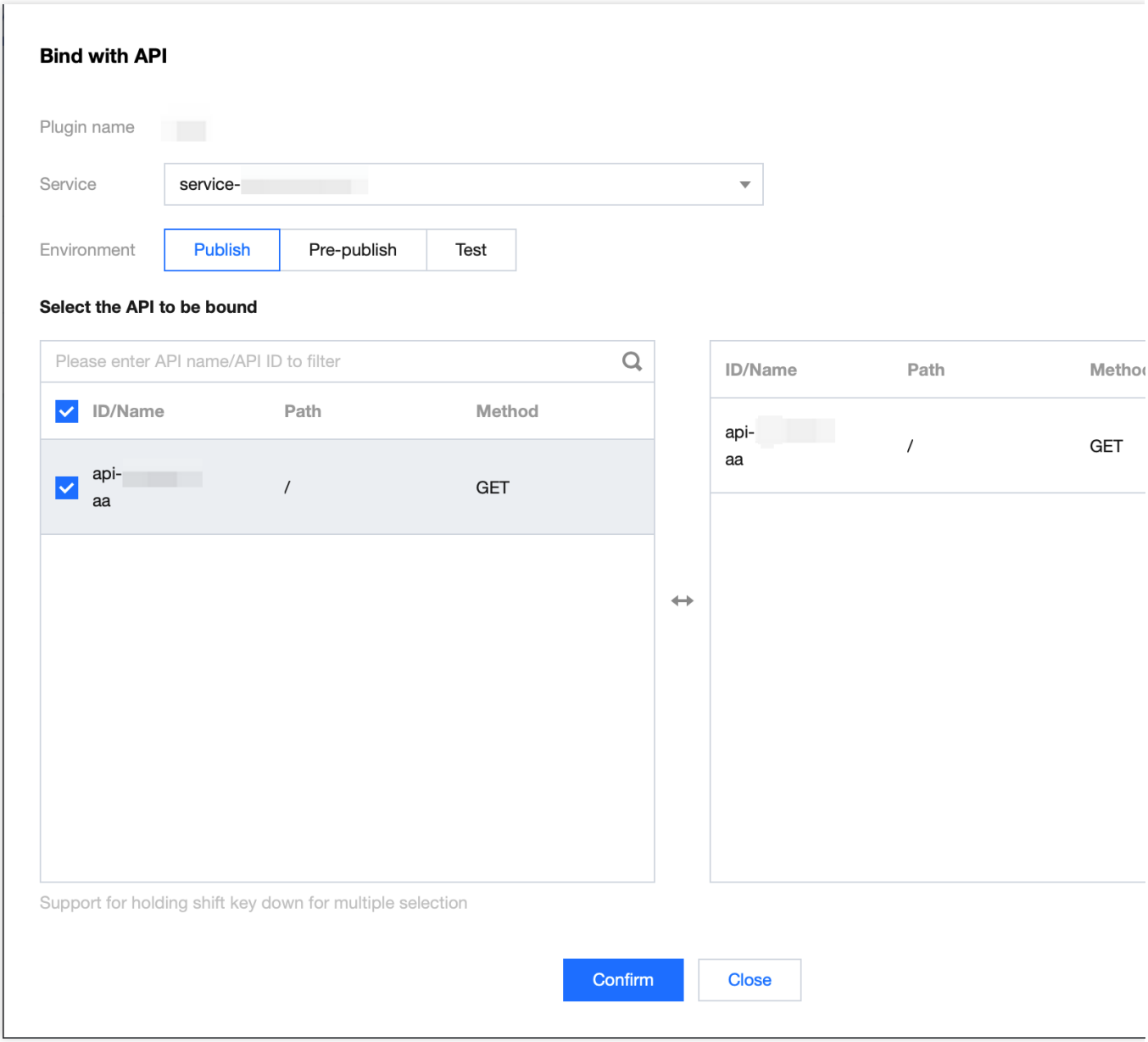
Parameter	Required	Description
VPC	Yes	Select the VPC of the response body rewriting service.
Request method	Yes	Request method of the response body rewriting function, which can be GET,

		POST, PUT, DELETE, HEAD, and ANY.
Backend address	Yes	Access address of the response body rewriting service, which can be an HTTP or HTTPS address.



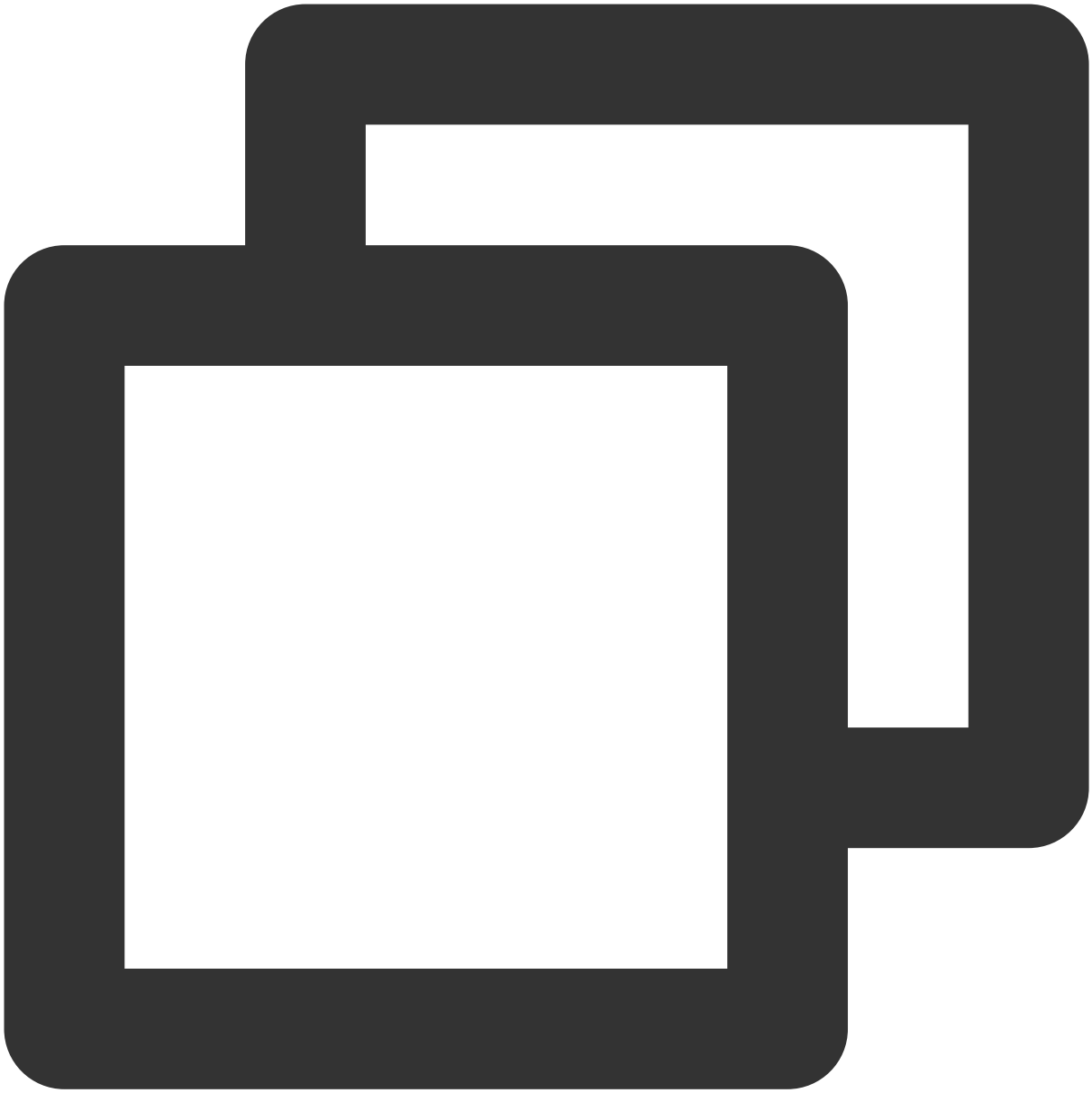
Step 3: binding the API

1. Select the plugin created in step 2 from the plugin list. Click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the API that needs to be bound to the plugin.



3. Click **OK** to bind the plugin to the API. At this time, the configuration of the plugin has taken effect for the API.

pluginData



```
{
    "endpoint_timeout":15, // Backend timeout period in seconds. Value range: 0-60
    "func_name":"test_name", // Custom SCF name
    "func_namespace":"test_namespace", // Custom SCF namespace
    "func_qualifier":"$LATEST", // Custom SCF version
    "is_base64_encoded":true, // Whether to Base64-encode the response content to b
    "is_custom_status":true, // Whether to send the response status code content to
    "is_custom_headers":true, // Whether to send the response Header content to the
    "is_custom_body":true, // Whether to send the response Body content to the SCF
    "user_id":1253970226 // appid
}
```

Notes

Binding a custom plugin to the API means creating a trigger for the function to trigger the API. Deleting the trigger on the SCF side means unbinding the plugin from the API.
Currently, the custom response body plugin supports only event-triggered functions but not HTTP-triggered functions.
The priority of the custom response body plugin is lower than that of all plugins applied during the request process.
If the custom response body plugin is bound to an API with a Mock or TSF backend, it will not take effect.
The custom response body plugin does not support the HTTP2 protocol.
The custom response body plugin does not support the response body compressed in the gzip format returned by the backend.

Anti-Replay Plugin

Last updated：2023-12-22 10:02:09

Overview

The anti-replay plugin is provided by API Gateway to protect APIs from replay attacks. You can create an anti-replay plugin and bind it to an API to protect your backend services.

Directions

Step 1. Create a plugin

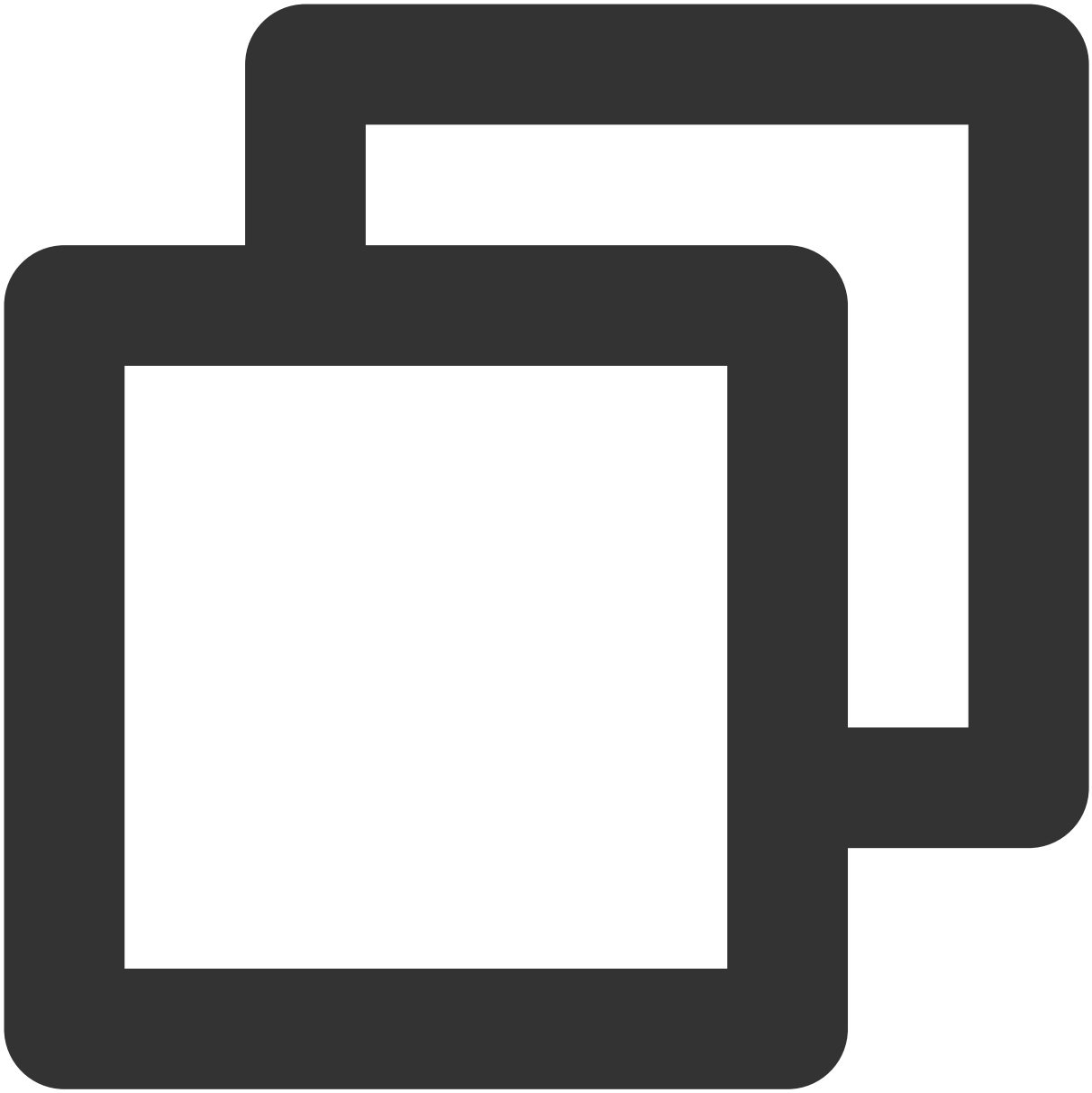
1. Log in to the API Gateway console.
2. On the left sidebar, click **Plugin** to enter the plugin list page.
3. Click **Create** in the top-left corner of the page and select **Anti-Replay** as the **Plugin Type** to create an anti-replay plugin.

Parameter	Required	Description
Forced anti-replay	No	Whether to enable forced anti-replay. It is disabled by default. If you need to enable it, you must add the `x-apigw-nonce` field to the request header.
Anti-replay period	No	Valid anti-replay period, which is 900 seconds by default. Value range: 1–1800.

Step 2. Bind an API and make the plugin effective

1. Select the just created plugin in the list and click **Bind API** in the **Operation** column.
2. In the **Bind API** pop-up window, select the service, environment, and the target API.

PluginData



```
{
    "force_nonce":true, // Whether to enable forced anti-replay. It is disabled by
    "nonce_ttl":1       //  Valid period of `nonce`, which is 900 seconds by defaul
}
```