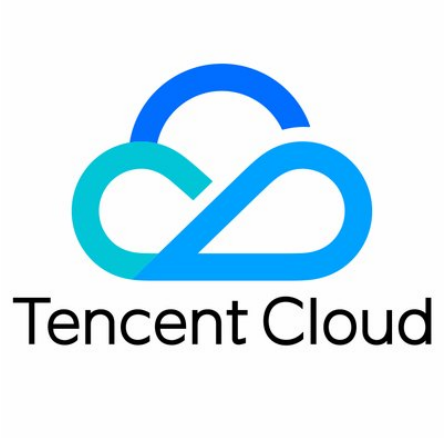
Serverless Cloud Function

Triggers



Service Notice

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

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

Contact Us

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

Contents

Triggers
Trigger Overview

Last updated: 2023-09-27 19:10:27

Tencent Cloud Function currently supports two triggering modes: **Event Trigger** and **HTTP Request Trigger**.

Event-Triggered

Event-Triggered is a typical serverless execution mode. Its core components are SCF functions and event sources, where an event source is a Tencent Cloud service or user-defined code that publishes an event, an SCF function is the handler of the event, and a function trigger is a set of correspondences between functions and event sources. For example, in the following scenarios:

- **Image/Video processing**: the application crops the images uploaded by users into an appropriate size, stores the images in COS, creates thumbnails of each image, and displays them on the user page. In this scenario, you need to select COS as the event source and publish the event to the SCF function when the file is created. The event data provides all the information about the bucket and the file.
- **Data processing**: the data collected during the day (such as clickstreams) is analyzed with a report generated at 00:00. In this scenario, you need to select a timer as the event source to publish an event to the SCF function at a specific time.
- **Custom Application**: In your application, the first image processing SCF function is invoked as a module. In this scenario, you need to manually call the Invoke API in the application to publish the event.

These event sources can be any of the following:

- **Internal event sources**: these are preconfigured Tencent Cloud services that can be used with SCF. If you configure one of these event sources as a function trigger, the function will be invoked automatically when an event occurs. The relationship between the event source and the function (i.e., event source mapping) will be maintained on the event source side.
- **Custom applications**: you can let custom applications publish events and invoke SCF functions.

Sample 1. COS publishes an event and invokes a function

You can configure the event source mapping for COS to determine what behaviors of COS will trigger an SCF function (such as object PUT or DELETE). The COS event source mapping is stored in COS and uses the bucket notification feature to direct COS to invoke the function when an event of a particular type occurs:

- A COS trigger is created.
- The user creates/deletes an object in the bucket.
- COS detects an object creation/deletion event.
- COS automatically invokes a function. It will be determined which function should be invoked based on the event source mapping stored in the COS configuration. The bucket and object information will be passed to the function as event data.

Sample 2. A timer publishes a time and invokes a function

The event source mapping of the timer is saved in the SCF function configuration to determine when the function should be automatically triggered:

- A timer trigger is created.
- The timer automatically invokes the function at the configured time.

Sample 3. A custom application invokes a function

If you need to invoke an SCF function in a custom application, you do not need to configure a function trigger or set up an event source mapping in this case; instead, you can use the `Invoke` API as the event source.

- The custom application uses the `Invoke` API to invoke the function and pass in the event data.
- The function receives the triggering request and is executed.
- If sync invocation is used, the function will return the result to the application.

> **Note**
> In this example, since the custom application and the function are produced by the same user, the user credentials (`APPID` , `SecretId` , and `SecretKey`) can be specified.

Precautions

1. The current trigger-related restrictions for a single function can be viewed in Quota Limits.
2. There are specific restrictions on event source mappings due to the limitations of different Tencent Cloud services. For example, for a COS trigger, the same event (such as file upload) in the same COS bucket cannot trigger multiple different functions.

HTTP Request-Triggered

HTTP request-triggered is a special trigger method supported by SCF HTTP-triggered Functions. A native HTTP request can be directly passed through to the function environment through API Gateway to trigger the execution and processing of the function. It is suitable for the development of web service scenarios. For detailed usage, please see Function Overview.

Trigger Event Message Structure Summary

Last updated: 2023-09-27 19:20:22

This document summarizes the message structures of all trigger events that are connected to SCF. For more information on trigger configuration and restrictions, please see Trigger Management.

> **Note**
> The event structure of the input parameter passed in by a trigger has been partially defined and can be used directly. You can get and use the Java library through the Java Cloud Event Definition or the Go library through the Go Cloud Event Definition.

Event Message Structures of Integration Request for API Gateway Trigger

Upon receiving a request, the API Gateway trigger sends event data in a JSON format similar to the one below to the bound SCF function. For more details, see API Gateway Trigger.

```
{
 "requestContext": {
  "serviceId": "service-f94sy04v",
  "path": "/test/{path}",
  "httpMethod": "POST",
  "requestId": "c6af9ac6-**-**-9a41-93e8deadbeef",
  "identity": {
   "secretId": "SECRET_ID" # Hardcoding keys into the code may expose them if the code is leaked. To protect the security of the keys, it is recommended to set them in the environment variables.
                           # Refer to https://cloud.tencent.com/document/product/1278/85305 for more information.
  },
  "sourceIp": "10.0.2.14",
  "stage": "release"
 },
 "headers": {
  "Accept-Language": "en-US,en,cn",
  "Accept": "text/html,application/xml,application/json",
  "Host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqloud.com",
  "User-Agent": "User Agent String"
 },
 "body": "{\"test\":\"body\"}",
 "pathParameters": {
  "path": "value"
 },
 "queryStringParameters": {
  "foo": "bar"
 },
 "headerParameters":{
```

```
    "Refer": "10.0.2.14"
  },
  "stageVariables": {
    "stage": "release"
  },
  "path": "/test/value",
  "queryString": {
    "foo" : "bar",
    "bob" : "alice"
  },
  "httpMethod": "POST"
}
```

Event Message Structure for Timer Trigger

When a function is invoked at a scheduled time, event data will be sent to the bound function in JSON format as shown below.

```
{
    "Type":"Timer",
    "TriggerName":"EveryDay",
    "Time":"2019-02-21T11:49:00Z",
    "Message":"user define msg body"
}
```

Event Message Structure for COS Trigger

When an object creation or deletion event occurs in the specified COS Bucket, event data will be sent to the bound SCF function in JSON format as shown below. For more details, see COS Trigger.

```
    "Records": [{
        "cos": {
            "cosSchemaVersion": "1.0",
            "cosObject": {
                "url": "http://testpic-1253970026.cos.ap-chengdu.myqcloud.com/testfile",
                "meta": {
                    "x-cos-request-id": "NWMxOWY4MGFfMjViMjU4NjRfMTUy****ZjM=",
                    "Content-Type": ""
                },
                "vid": "",
                "key": "/1253970026/testpic/testfile",
                "size": 1029
            },
            "cosBucket": {
                "region": "cd",
                "name": "testpic",
                "appid": "1253970026"
            },
            "cosNotificationId": "unkown"
```

```
      },
      "event": {
        "eventName": "cos:ObjectCreated:*",
        "eventVersion": "1.0",
        "eventTime": 1545205770,
        "eventSource": "qcs::cos",
        "requestParameters": {
          "requestSourceIP": "192.168.15.101",
          "requestHeaders": {
            "Authorization": "q-sign-algorithm=sha1&q-
ak=AKIDQm6iUh2NJ6jL41tVUis9KpY5Rgv49zyC&q-sign-time=1545205709;1545215769&q-key-
time=1545205709;1545215769&q-header-list=host;x-cos-storage-class&q-url-param-list=&q-
signature=098ac7dfe9cf21116f946c4b4c29001c2b449b14"
          }
        },
        "eventQueue": "qcs:0:lambda:cd:appid/1253970026:default.printevent.$LATEST",
        "reservedInfo": "",
        "reqid": 179398952
      }
    }]
}
```


Event Message Structure for CKafka Trigger

When a message is received in the specified CKafka Topic, the consumer module in the SCF backend consumes the message, assembles it into an event in a JSON format similar to the one shown below, triggers the bound function, and passes the data content as an argument to the function. For more details, see CKafka Trigger.


```
{
 "Records": [
  {
   "Ckafka": {
    "topic": "test-topic",
    "partition":1,
    "offset":36,
    "msgKey": "None",
    "msgBody": "Hello from Ckafka!"
   }
  },
  {
   "Ckafka": {
    "topic": "test-topic",
    "partition":1,
    "offset":37,
    "msgKey": "None",
    "msgBody": "Hello from Ckafka again!"
   }
  }
 ]
```



```
}
```

Event Structure for CMQ Topic Trigger

When a message is received in the specified CMQ Topic, event data will be sent to the bound SCF function in a JSON format similar to the one shown below. For more details, see CMQ Topic Trigger.

```
{
 "Records": [
  {
   "CMQ": {
    "type": "topic",
    "topicOwner":"120xxxxx",
    "topicName": "testtopic",
    "subscriptionName": "xxxxxx",
    "publishTime": "1970-01-01T00:00:00.000Z",
    "msgId": "123345346",
    "requestId":"123345346",
    "msgBody": "Hello from CMQ!",
    "msgTag": "tag1,tag2"
   }
  }
 ]
}
```

Event Message Structure for CLS Triggers

When a message is received by the specified CLS trigger, the consumer module in the CLS backend consumes the message and assembles it to asynchronously invoke your function. To ensure the efficiency of data transmission in a single trigger, the value of the data field is a Base64-encoded ZIP document. For more details, see CLS Trigger.

```
{
 "clslogs": {
  "data":
"ewogICAgIm1lc3NhZ2VUeXBlIjogIkRBVEFfTUVTU0FHRSIsCiAgICAib3duZXIiOiAiMTIzNDU2Nzg5MDE
yIiwKICAgICJsb2dHcm91cCI6I..."
 }
}
```

After being decoded and decompressed, the log data will look like the following JSON body (using decoded CLS Logs message data as an example):

```
{
  "topic_id": "xxxx-xx-xx-xx-yyyyyyyy",
  "topic_name": "testname",
  "records": [{
    "timestamp": "1605578090000000",
```

```
      "content": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    }, {
      "timestamp": "1605578090000003",
      "content": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    }]
}
```

MPS Trigger Event Message Structure

When a message is received by the specified MPS trigger, the event structure and fields are illustrated using the WorkflowTask task as an example. For more details, see MPS Trigger. The example is as follows:

```
{
  "EventType":"WorkflowTask",
  "WorkflowTaskEvent":{
    "TaskId":"245**654-WorkflowTask-f46dac7fe2436c47****d71946986t0",
    "Status":"FINISH",
    "ErrCode":0,
    "Message":"",
    "InputInfo":{
      "Type":"COS",
      "CosInputInfo":{
        "Bucket":"macgzptest-125****654",
        "Region":"ap-guangzhou",
        "Object":"/dianping2.mp4"
      }
    },
    "MetaData":{
      "AudioDuration":11.261677742004395,
      "AudioStreamSet":[
        {
          "Bitrate":127771,
          "Codec":"aac",
          "SamplingRate":44100
        }
      ],
      "Bitrate":2681468,
      "Container":"mov,mp4,m4a,3gp,3g2,mj2",
      "Duration":11.261677742004395,
      "Height":720,
      "Rotate":90,
      "Size":3539987,
      "VideoDuration":10.510889053344727,
      "VideoStreamSet":[
        {
          "Bitrate":2553697,
          "Codec":"h264",
          "Fps":29,
          "Height":720,
```

```
            "Width":1280
         }
      ],
      "Width":1280
   },
   "MediaProcessResultSet":[
      {
         "Type":"Transcode",
         "TranscodeTask":{
            "Status":"SUCCESS",
            "ErrCode":0,
            "Message":"SUCCESS",
            "Input":{
               "Definition":10,
               "WatermarkSet":[
                  {
                     "Definition":515247,
                     "TextContent":"",
                     "SvgContent":""
                  }
               ],
               "OutputStorage":{
                  "Type":"COS",
                  "CosOutputStorage":{
                     "Bucket":"gztest-125****654",
                     "Region":"ap-guangzhou"
                  }
               },
               "OutputObjectPath":"/dasda/dianping2_transcode_10",
               "SegmentObjectName":"/dasda/dianping2_transcode_10_{number}",
               "ObjectNumberFormat":{
                  "InitialValue":0,
                  "Increment":1,
                  "MinLength":1,
                  "PlaceHolder":"0"
               }
            },
            "Output":{
               "OutputStorage":{
                  "Type":"COS",
                  "CosOutputStorage":{
                     "Bucket":"gztest-125****654",
                     "Region":"ap-guangzhou"
                  }
               },
               "Path":"/dasda/dianping2_transcode_10.mp4",
               "Definition":10,
               "Bitrate":293022,
               "Height":320,
               "Width":180,
               "Size":401637,
```



```
            "Duration":11.26200008392334,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Md5":"31dcf904c03d0cd78346a12c25c0acc9",
            "VideoStreamSet":[
               {
                  "Bitrate":244608,
                  "Codec":"h264",
                  "Fps":24,
                  "Height":320,
                  "Width":180
               }
            ],
            "AudioStreamSet":[
               {
                  "Bitrate":48414,
                  "Codec":"aac",
                  "SamplingRate":44100
               }
            ]
         }
      },
      "AnimatedGraphicTask":null,
      "SnapshotByTimeOffsetTask":null,
      "SampleSnapshotTask":null,
      "ImageSpriteTask":null
   },
   {
      "Type":"AnimatedGraphics",
      "TranscodeTask":null,
      "AnimatedGraphicTask":{
         "Status":"FAIL",
         "ErrCode":30010,
         "Message":"TencentVodPlatErr Or Unkown",
         "Input":{
            "Definition":20000,
            "StartTimeOffset":0,
            "EndTimeOffset":600,
            "OutputStorage":{
               "Type":"COS",
               "CosOutputStorage":{
                  "Bucket":"gztest-125****654",
                  "Region":"ap-guangzhou"
               }
            },
            "OutputObjectPath":"/dasda/dianping2_animatedGraphic_20000"
         },
         "Output":null
      },
      "SnapshotByTimeOffsetTask":null,
      "SampleSnapshotTask":null,
      "ImageSpriteTask":null
```



```
        },
        {
            "Type":"SnapshotByTimeOffset",
            "TranscodeTask":null,
            "AnimatedGraphicTask":null,
            "SnapshotByTimeOffsetTask":{
                "Status":"SUCCESS",
                "ErrCode":0,
                "Message":"SUCCESS",
                "Input":{
                    "Definition":10,
                    "TimeOffsetSet":[

                    ],
                    "WatermarkSet":[
                        {
                            "Definition":515247,
                            "TextContent":"",
                            "SvgContent":""
                        }
                    ],
                    "OutputStorage":{
                        "Type":"COS",
                        "CosOutputStorage":{
                            "Bucket":"gztest-125****654",
                            "Region":"ap-guangzhou"
                        }
                    },
                    "OutputObjectPath":"/dasda/dianping2_snapshotByOffset_10_{number}",
                    "ObjectNumberFormat":{
                        "InitialValue":0,
                        "Increment":1,
                        "MinLength":1,
                        "PlaceHolder":"0"
                    }
                },
                "Output":{
                    "Storage":{
                        "Type":"COS",
                        "CosOutputStorage":{
                            "Bucket":"gztest-125****654",
                            "Region":"ap-guangzhou"
                        }
                    },
                    "Definition":0,
                    "PicInfoSet":[
                        {
                            "TimeOffset":0,
                            "Path":"/dasda/dianping2_snapshotByOffset_10_0.jpg",
                            "WaterMarkDefinition":[
                                515247
```



```
                    ]
                }
            ]
        }
    },
    "SampleSnapshotTask":null,
    "ImageSpriteTask":null
},
{
    "Type":"ImageSprites",
    "TranscodeTask":null,
    "AnimatedGraphicTask":null,
    "SnapshotByTimeOffsetTask":null,
    "SampleSnapshotTask":null,
    "ImageSpriteTask":{
        "Status":"SUCCESS",
        "ErrCode":0,
        "Message":"SUCCESS",
        "Input":{
            "Definition":10,
            "OutputStorage":{
                "Type":"COS",
                "CosOutputStorage":{
                    "Bucket":"gztest-125****654",
                    "Region":"ap-guangzhou"
                }
            },
            "OutputObjectPath":"/dasda/dianping2_imageSprite_10_{number}",
            "WebVttObjectName":"/dasda/dianping2_imageSprite_10",
            "ObjectNumberFormat":{
                "InitialValue":0,
                "Increment":1,
                "MinLength":1,
                "PlaceHolder":"0"
            }
        },
        "Output":{
            "Storage":{
                "Type":"COS",
                "CosOutputStorage":{
                    "Bucket":"gztest-125****654",
                    "Region":"ap-guangzhou"
                }
            },
            "Definition":10,
            "Height":80,
            "Width":142,
            "TotalCount":2,
            "ImagePathSet":[
                "/dasda/imageSprite/dianping2_imageSprite_10_0.jpg"
            ],
```



```
                "WebVttPath":"/dasda/imageSprite/dianping2_imageSprite_10.vtt"
            }
          }
        }
      ]
    }
}
```

Event Message Structure for CLB Triggers

When a request is received by the CLB trigger, it sends event data in a format similar to the following JSON to the bound SCF function. For more details, see CLB Trigger Description.

```
{
 "headers": {
  "Content-type": "application/json",
  "Host": "test.clb-scf.com",
  "User-Agent": "Chrome",

  "X-Stgw-Time": "1591692977.774",
  "X-Client-Proto": "http",
  "X-Forwarded-Proto": "http",
  "X-Client-Proto-Ver": "HTTP/1.1",
  "X-Real-IP": "9.43.175.219",
  "X-Forwarded-For": "9.43.175.xx"

  "X-Vip": "121.23.21.xx",
  "X-Vport": "xx",
  "X-Uri": "/scf_location",
  "X-Method": "POST"
  "X-Real-Port": "44347",
 },
 "payload": {
  "key1": "123",
  "key2": "abc"
 },
 “isBase64Encoded”: “false”
}
```

Event Structure Passed in by EventBridge Trigger

Through the Tencent Cloud EventBridge, the function event trigger source can be further expanded. Events generated by the EventBridge will be sent to the cloud function in the following format. The content in the `"data"` field is determined by the event source, here taking TDMQ as an example:

```
 {
 "specversion": "0",
 "id": "13a3f42d-7258-4ada-da6d-023a33**",
 "type": "connector:tdmq",
```

```
  "source": "tdmq.cloud.tencent",
  "subjuect":
"qcs::tdmq:$region:$account:topicName/$topicSets.clusterId/$topicSets.environmentId/$topicSets.t
opicName/$topicSets.subscriptionName",
  "time": "1615430559146",
  "region": "ap-guangzhou",
  "datacontenttype": "application/json;charset=utf-8",
  "data": {
          "topic":  "persistent://appid/namespace/topic-1",
        "tags": "testtopic",
          "TopicType": "0",
          "subscriptionName": "xxxxxx",
          "toTimestamp": "1603352765001",
        "partitions": "0",
          "msgId": "123345346",
          "msgBody": "Hello from TDMQ!"
  }
```

API Gateway Trigger Overview

Last updated: 2023-09-27 19:21:50

You can implement backend web services by writing SCF functions and providing services through API Gateway which will pass the request content as parameters to the function and return the result from the function back to the requester as the response.

> **Note**
> API Gateway triggers can trigger both event-triggered functions and HTTP-triggered functions. This document only describes the request method of **event-triggered function** triggering. For more information on HTTP-triggered function triggering, see Trigger Management.

Characteristics of API Gateway triggers:

- **Push Model**
  Upon receiving an API request, if the API's backend configuration on the gateway is set to connect to a cloud function, that function will be triggered to run. Concurrently, the API Gateway will send the relevant information of the API request to the triggered function in the form of an event parameter. The relevant information of the API request includes details such as the specific service and API rules that received the request, the actual path of the request, the method, and the request's path, header, query, etc.
- **Synchronous Invocation**
  API Gateway invokes the function synchronously, waiting for the function to return before the timeout period configured in the API Gateway expires. For more information about invocation types, please refer to Invocation Types.

API Gateway Trigger Configuration

API Gateway triggers can be configured either in the SCF console or in the API Gateway console.

SCF console

In the **SCF console**, you can add an API Gateway trigger in the trigger method section. You can choose an existing API service or create a new one. It supports the definition of request methods (currently supporting **ANY, GET, HEAD, POST, PUT, DELETE** six types of requests), release environments (testing, pre-release, and release environments), and authentication methods (API Gateway key pair). For more details, see Trigger Management.

API Gateway console

When configuring API rules in the **API Gateway console**, you can select SCF as the backend and select functions in the same region as the API service. In the API Gateway console, you can configure

and manage advanced API services such as traffic throttling plan, blocklist, and allowlist.
When you configure the connection with SCF in API Gateway, you also need to configure the timeout period. The request timeout period in API Gateway and the execution timeout period in SCF take effect respectively. The timeout rules are as follows:

- **API Gateway Timeout Period > SCF Timeout Period**
  The SCF timeout takes effect first. The API request responds with a `200 HTTP code`, but the returned content is the SCF timeout error message.
- **API Gateway Timeout Period < SCF Timeout Period**
  The API Gateway timeout takes effect first. The API request responds with a `5xx HTTP code`, indicating a request timeout.

Limitation on API Gateway Trigger Binding

In API Gateway, one API rule can be bound to only one function, but one function can be bound to multiple API rules as the backend. You can create an API with different paths in the API Gateway console and point the backend to the same function. APIs with the same path, same request method, and different release environments are regarded as the same API and cannot be bound repeatedly.
API Gateway triggers currently can only be bound to functions in the same region; for example, a function created in the Guangzhou region can only be bound to and triggered by API rules created in the Guangzhou region. If you want to trigger a function through API Gateway configuration in a specific region, please create a function in that region.

Request and Response

Request method is the method to process request sent from API Gateway to SCF, and response method is the method to process the returned value sent from SCF to API Gateway. Both request and response methods can be planned and implemented by means of passthrough and integration.

Integration request and passthrough request

Integration request means that API Gateway converts the content of the HTTP request into request data structures which are passed to the function for handling as `event` input parameters of the function. The following details the request data structures.
For more information on passthrough requests, see Trigger Management.

> **Note**
> When transferring images or files to SCF through API Gateway, you need to Base64-encode them. If the size of a Base64-encoded file is above 6 MB, we recommend you upload the file to COS through the client and pass the object address to SCF first. Then, SCF will pull the file from COS to complete the upload.

Event Message Structures of Integration Request for API Gateway Trigger

Upon receiving a request, the API Gateway trigger sends event data in a format similar to the following JSON to the bound SCF.

```
{
 "requestContext": {
  "serviceId": "service-f94sy04v",
  "path": "/test/{path}",
  "httpMethod": "POST",
  "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
  "identity": {
   "secretId": "abdcdxxxxxxxsdfs"
 },
  "sourceIp": "10.0.2.14",
  "stage": "release"
 },
 "headers": {
  "accept-Language": "en-US,en,cn",
  "accept": "text/html,application/xml,application/json",
  "host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqloud.com",
  "user-Agent": "User Agent String"
 },
 "body": "{\"test\":\"body\"}",
 "pathParameters": {
  "path": "value"
 },
 "queryStringParameters": {
  "foo": "bar"
 },
 "headerParameters":{
  "Refer": "10.0.2.14"
 },
 "stageVariables": {
  "stage": "release"
 },
 "path": "/test/value",
 "queryString": {
  "foo" : "bar",
  "bob" : "alice"
 },
 "httpMethod": "POST"
}
```

The data structures are detailed as below:

Structure	Content
requestContext	Configuration information of the originating API Gateway, request identifier, authentication information, and source information are included. Specifically: • The serviceId, path, and httpMethod point to the service ID of the API Gateway, the path of the API, and the method respectively. • The stage points to the environment where the originating API is located. • The requestId signifies the unique ID for the current request. • The identity signifies the method of user authentication and the

	authenticated information. • The sourceIp signifies the originating IP of the request.
path	Records the complete `Path` information of the actual request.
httpMethod	Records the `HTTP` method of the actual request.
queryString	Records the complete `Query` content of the actual request.
body	Records the content of the actual request after being converted into a `String`.
headers	Records the complete `Header` content of the actual request.
pathParameters	Records the `Path` parameters configured in API Gateway and their actual values.
queryStringParameters	Records the `Query` parameters configured in API Gateway and their actual values.
headerParameters	Records the `Header` parameters configured in API Gateway and their actual values.

Note

- The content of `requestContext` may be increased during API Gateway iteration. At present, it is guaranteed that the content of the data structure will only be increased but not reduced, so that the existing structure will not be compromised.
- Parameters in real requests may appear in multiple locations and can be selected based on your business needs.

Integration response and passthrough response

Integration response refers to the process where API Gateway parses the return content from the cloud function and constructs an HTTP response based on the parsed content. By using integration response, you can control the status code, headers, and body content of the response through code, enabling custom format content responses, such as XML, HTML, JSON, or even JS content. When using integration response, it is necessary to follow the API Gateway trigger's integration response return data structure to be successfully parsed by API Gateway, otherwise, an error message `{"errno":403,"error":"Invalid scf response format. please check your scf response format."}` will appear. Passthrough response means that API Gateway directly passes the returned content of the function to the API requester. Generally, the data format of this type of responses is fixed at JSON format, the status code is defined according to the status of function execution, and status code 200 is returned if the function is successfully executed. With passthrough response, you can get the JSON format and parse the structures at the call location to get the content in the structures.

Note

- If the API Gateway trigger is configured in the API Gateway console, the way to handle the response is passthrough response by default. To enable integration response, select **Enable**

integration response at the backend configuration location in the API configuration and return the content in the data structures detailed below in the code.

- If the API Gateway trigger is configured in the SCF console, the integration response feature is enabled by default. Please pay attention to the format of the returned data.
- Upon enabling integration response, if asynchronous execution is also enabled, the function will return a 404 status code. It is recommended to avoid enabling these two configurations simultaneously.

Returned data structures of integration response for API Gateway trigger

If integration response is set for API Gateway, the data structure in the following JSON format should be returned to API Gateway.

```
{
  "isBase64Encoded": false,
  "statusCode": 200,
    "headers": {"Content-Type":"text/html"},
  "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

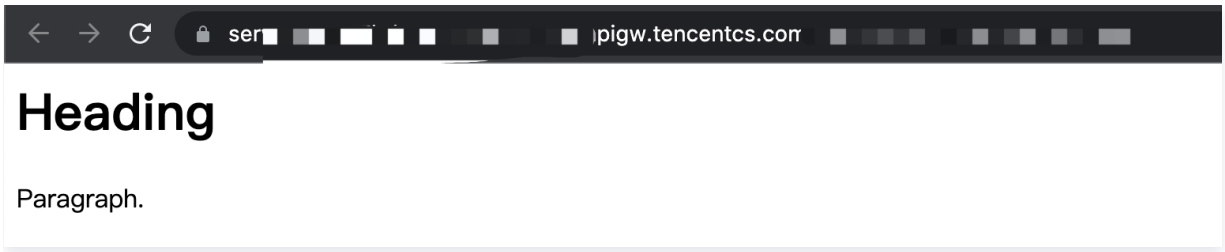
The data structures are detailed as below:

Structure	Content
isBase64Encoded	This indicates whether the content in the `body` is Base64-encoded binary. It should be `true` or `false` in JSON format. The specification of `true` and `false` varies by language, so you should adjust based on your actual language.
statusCode	HTTP return code, which should be an integer value.
headers	The content of the HTTP response headers should be multiple key-value pairs or `key:[value,value]` objects, where both key and value are strings. The headers request does not currently support the Location key.
body	HTTP return body.

Taking Python 3.6 as an example, the sample code is as follows:

```
# -*- coding: utf8 -*-
import json
def main_handler(event, context):
  return {
    "isBase64Encoded": False,
  "statusCode": 200,
    "headers": {"Content-Type":"text/html"},
  "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

The returned result of a function triggered by API Gateway is as follows:



If you need to return multiple headers with the same key, you can use a string array to describe different values; for example:

```
{
  "isBase64Encoded": false,
  "statusCode": 200,
  "headers": {"Content-Type":"text/html","Key":["value1","value2","value3"]},
  "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

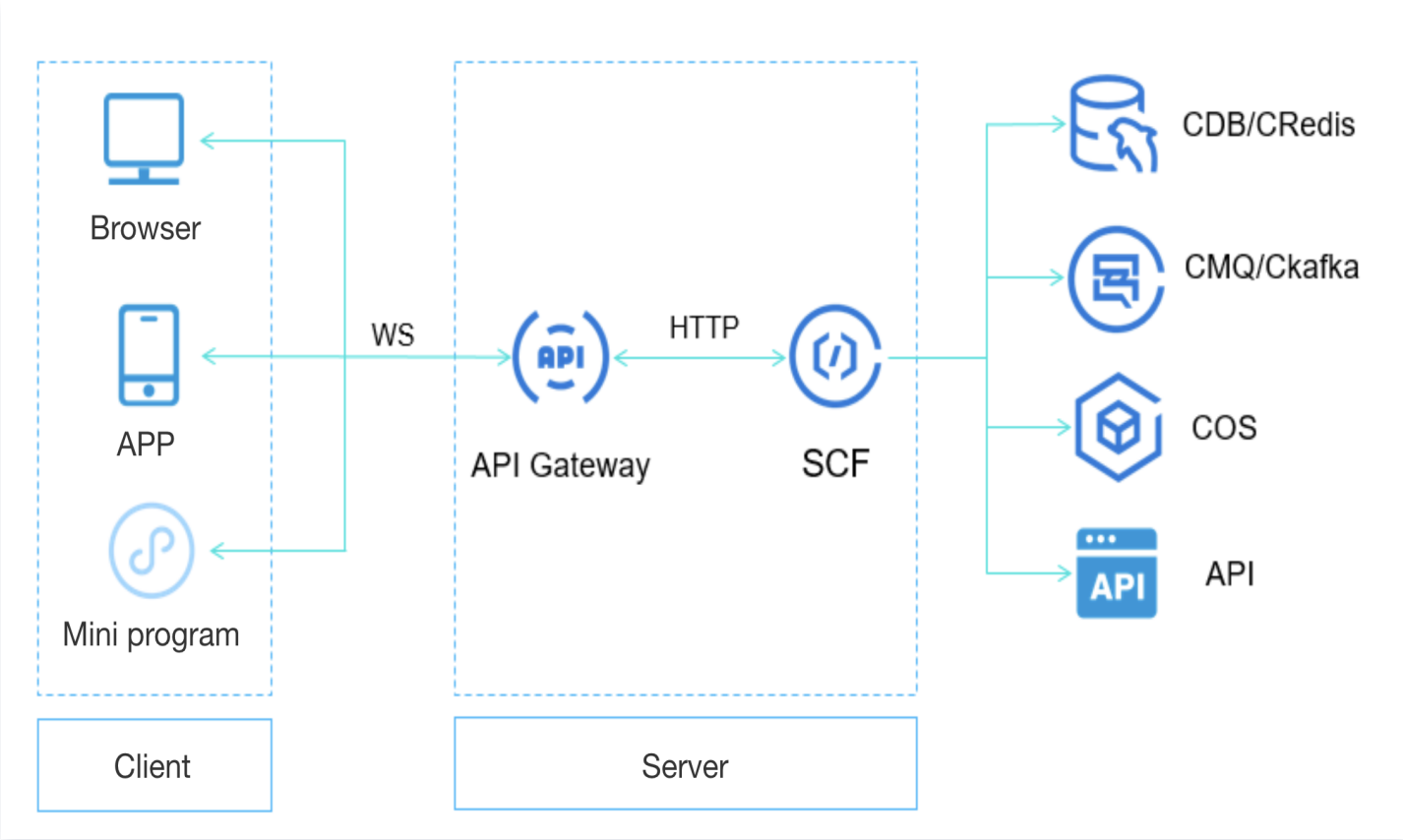
Websocket
How It Works

Last updated: 2023-09-27 19:24:32

> **Note**
> This document introduces the **Event Function** solution that supports WebSocket. Currently, the **Web Function** supports the native WebSocket protocol. For more details, please refer to WebSocket Protocol Support.

How to Implement

WebSocket is a new TCP-based network protocol. It implements full-duplex communication between browser and server, i.e., allowing server to actively send information to client. In contrast, a server using the traditional HTTP protocol only allows a client to get the data that needs to be pushed through polling or long polling.
Given that cloud functions are stateless and operate on a trigger basis, meaning they are only triggered when an event arrives, the implementation of WebSocket involves a combination of cloud functions and API Gateway. The API Gateway undertakes and maintains the connection with the client. You can consider that the API Gateway and SCF together constitute the server side. When the client sends a message, it is first passed to the API Gateway, which then triggers the cloud function. When the server-side cloud function needs to send a message to the client, the cloud function first POSTs the message to the reverse push link of the API Gateway, which then pushes the message to the client. The specific implementation architecture is as follows:



The entire lifecycle of WebSocket mainly consists of the following events:

- Connection establishment: the client requests to connect with the server and establishes a connection.
- Data upstream: the client sends data to the server through the established connection.
- Data downstream: the server sends data to the client through the established connection.
- Client disconnection: the client requests to close the established connection.
- Server disconnection: the server requests to close the established connection.

SCF and API Gateway handle the events throughout the lifecycle of WebSocket as follows:

- Connection establishment: the client establishes a WebSocket connection with API Gateway which sends a connection establishment event to SCF.
- Data upstream: the client sends data through WebSocket, and API Gateway forwards the data to SCF.
- Data downstream: SCF sends a request to the push address specified by API Gateway, which then sends the data to the client through WebSocket.
- Client disconnection: the client requests to disconnect, and API Gateway sends a disconnection event to SCF.
- Server disconnection: SCF sends a disconnection request to the push address specified by API Gateway, which will close the WebSocket connection after receiving the request.

Therefore, the interaction between API Gateway and SCF needs to be sustained by three types of functions:

- Registration function: this function is triggered when a WebSocket connection is requested and established between the client and API Gateway, notifying SCF of the `secConnectionID` of the

WebSocket connection. The `secConnectionID` is usually recorded in the persistent storage in this function for reverse push of subsequent data.

- Cleanup function: this function is triggered when the client initiates a WebSocket disconnection request, notifying SCF to prepare to disconnect the `secConnectionID`. The `secConnectionID` is usually cleaned from the persistent storage in this function.
- Transfer function: this function is triggered when the client sends data through the WebSocket connection, notifying SCF of the `secConnectionID` of the connection and the data sent. Business data is usually processed in this function. For example, it determines whether to push data to other `secConnectionID` values in the persistent storage.

> **Note**
> When you need to actively push data to a `secConnectionID` or disconnect a `secConnectionID`, the reverse push address of API Gateway has to be used.

Data Structure

Connection establishment

1. When the client initiates a WebSocket connection establishment request, API Gateway encapsulates the agreed upon JSON data structures in the request body and sends it to the registration function in the HTTP POST method. You can get the request body from the function's event. Below is a sample:

```
{
 "requestContext": {
  "serviceName": "testsvc",
  "path": "/test/{testvar}",
  "httpMethod": "GET",
  "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
  "identity": {
   "secretId": "abdcdxxxxxxxsdfs"
  },
  "sourceIp": "10.0.2.14",
  "stage": "prod",
  "websocketEnable":true
 },
 "websocket":{
  "action":"connecting",
  "secConnectionID":"xawexasdfewezdfsdfeasdfffa==",
  "secWebSocketProtocol":"chat,binary",
  "secWebSocketExtensions":"extension1,extension2"
 }
}
```

The data structures are detailed as below:

Structure	Content

Name	
requestContext	Configuration information, request ID, authentication information, and source information of API Gateway where the request comes from, including: serviceName, path, httpMethod: they point to API Gateway's service, API path, and method. stage: it points to the environment where the request source API is located. requestId: it identifies the unique ID of the current request. identity: it identifies the user's authentication method and authentication information. sourceIp: it identifies the request source IP.
websocket	Details of connection establishment, including: action: action of this request. secConnectionID: a string that identifies the ID of the WebSocket connection. The original length is 128 bits, which is a Base64-encoded string with a total of 32 characters. secWebSocketProtocol: string; optional.It represents the list of sub-protocols. If this field is in the original request, it will be passed to the function; otherwise, it will not appear. secWebSocketExtensions: string; optional.It represents the list of extensions. If this field is in the original request, it will be passed to the function; otherwise, it will not appear.

Note:
The content of `requestContext` may be increased significantly during API Gateway iteration. At present, it is guaranteed that the content of the data structure will only be increased but not reduced, so that the existing structure will not be compromised.

2. When the registration function receives the connection establishment request, it needs to return the response message of whether to agree to establish the connection to API Gateway at the end of function handling. The response body must be in JSON format as shown in the sample below:

```
{
  "errNo":0,
  "errMsg":"ok",
  "websocket":{
    "action":"connecting",
    "secConnectionID":"xawexasdfewezdfsdfeasdfffa==",
    "secWebSocketProtocol":"chat,binary",
    "secWebSocketExtensions":"extension1,extension2"
  }
}
```

The data structures are detailed as below:

Structure	Content

errNo	Integer, required. Response error code. When errNo is 0, it indicates that the handshake was successful and the connection establishment is agreed upon.
errMsg	String, required. Error reason. When errNo is non-zero, it indicates that it is effective.
websocket	Details of connection establishment, including: • action: action of this request. • secConnectionID: a string that identifies the ID of the WebSocket connection. The original length is 128 bits, which is a Base64-encoded string with a total of 32 characters. • secWebSocketProtocol: string; optional.It is the value of a single sub-protocol. If this field is in the original request, it will be passed through to the client by API Gateway. • secWebSocketExtensions: string; optional.It is the value of a single extension. If this field is in the original request, it will be passed through to the client by API Gateway.

Note:
- If the SCF request times out, it will be deemed by default that the connection establishment failed.
- When API Gateway receives the response message from SCF, it will check the HTTP response code first. If the response code is 200, it will parse the response body; otherwise, it will deem that SCF failed and connection establishment was refused.

Data transfer

Upstream data transfer

Transfer request

When the client sends data through WebSocket, API Gateway will encapsulate the agreed JSON data structures in the request body and send it to the transfer function in the HTTP POST method. You can get the request body from the function's event. Below is a sample:

```
{
 "websocket":{
  "action":"data send",
  "secConnectionID":"xawexasdfewezdfsdfeasdfffa==",
  "dataType":"text",
  "data":"xxx"
 }
}
```

The data structures are detailed as below:

Category	Content

websocket	Details of data transfer.
action	The action of this request, exemplified here by "data send".
secConnectionID	A string that identifies the WebSocket connection ID. The original length is 128Bit, which is a string encoded in base64, consisting of 32 characters.
dataType	Type of data transfer. "binary": denotes binary. "text": denotes text.
data	The data being transferred. If "dataType" is "binary", it is a binary stream encoded in base64; if "dataType" is "text", it is a string.

Transfer response

After the transfer function finishes executing, it will return an HTTP response to API Gateway which will act according to the response code:

- If the response code is 200, the function was executed successfully.
- If the response code is not 200, a system failure occurred, and API Gateway will actively send a FIN packet to the client.

> **Note**
> API Gateway does not handle the content in the response body.

Downstream data callback

Callback request

When SCF needs to push data to the client or actively disconnect, it can initiate a request, encapsulate the data in the request body, and send it to the reverse push address of API Gateway in the POST method. The request body must be in JSON format, as shown in the sample below:

```
{
 "websocket":{
  "action": "data transmission", // Sending data to the client
  "secConnectionID":"xawexasdfewezdfsdfeasdfffa==",
  "dataType":"text",
  "data":"xxx"
 }
}
```

```
{
 "websocket":{
  "action": "termination", // Sending a disconnection request
  "secConnectionID":"xawexasdfewezdfsdfeasdfffa=="
 }
}
```

The data structures are detailed as below:

Parameter	Content
websocket	Details of data transfer.
action	The action of this request, supporting two types of content: "data transmission" and "termination": "data transmission": This is to send data to the client. "termination": This initiates a disconnection request to the client and may not include the "dataType" and "data" content.
secConnectionID	A string that identifies the ID of the WebSocket connection. The original length is 128 bits, which is a Base64-encoded string with a total of 32 characters.
dataType	The type of data transmission, there are two kinds: "binary": denotes binary. "text": denotes text.
data	Data to be transmitted: If "dataType" is "binary", it is a Base64-encoded binary stream. If "dataType" is "text", it is a string.

Callback response

After the callback is over, the result of the callback can be determined based on the response code of API Gateway:

- If the response code is 200, the call succeeded.
- If the response code is not 200, a system failure occurred, and API Gateway will actively send a FIN packet to the client.

In addition, the response body in JSON format can be obtained in the response result as shown in the sample below:

```
{
  "errNo":0,
  "errMsg":"ok"
}
```

The data structures are detailed as below:

Parameter	Content
errNo	Integer; response error code. 0 means success.
errMsg	String; cause of error.

Connection cleanup

Active disconnection by client

Logout request

When the client actively initiates a WebSocket disconnection request, API Gateway will encapsulate the agreed upon JSON data structures in the request body and send it to the cleanup function in the HTTP POST method. You can get the request body from the function's event. Below is a sample:

```
{
 "websocket":{
  "action":"closing",
  "secConnectionID":"xawexasdfewezdfsdfeasdfffa=="
 }
}
```

The data structures are detailed as below:

Parameter	Content
websocket	Details of disconnection.
action	Action of this request, which is "closing" here.
secConnectionID	String. This is the ID that identifies the WebSocket connection. The original length is 128 bits, which is a string encoded in base64, consisting of 32 characters.

> **Note**
> In the cleanup function, you can get the `secConnectionID` from the event and delete the ID from the persistent storage (such as a database).

Logout response

After the cleanup function finishes executing, it will return an HTTP response to API Gateway, which will act according to the response code:

- If the response code is 200, the function was executed successfully.
- If the response code is not 200, a system failure occurred.

> **Note**
> API Gateway does not handle the content in the response body.

Active disconnection by server

Refer to Downlink Data Callback, initiate a Request request in the function, encapsulate the following data structure in the Body, and send it to the reverse push address of the API Gateway using the POST method.

```
{
 "websocket":{
  "action": "termination", // Sending a disconnection request
  "secConnectionID":"xawexasdfewezdfsdfeasdfffa=="
 }
}
```

> **Note**
> When actively disconnecting the link with the client, you need to get the `secConnectionID` of the client's WebSocket, enter it in the data structure, and then delete the ID from the persistent storage (such as a database).

COS Trigger
COS Trigger Description

Last updated: 2023-09-27 19:27:16

You can write an SCF function to handle object creation and deletion events in a COS bucket. COS can publish events to the function and invoke it by using the event data as parameters. You can add a bucket notification configuration in the COS bucket, which can identify information such as the trigger event type and name of the function to be invoked.
The COS trigger has the following features:

- **Push Model**
  COS monitors specified bucket actions (event types) and invokes related functions, pushing event data to the SCF function. The push model uses bucket notifications to maintain the event source mapping of COS.
- **Asynchronous Invocation**
  COS always uses asynchronous invocation to call functions, and the results are not returned to the caller. For more information about invocation types, please refer to Invocation Types.

COS Trigger Attributes

- COS bucket (required): The configured COS bucket, which can only be a COS bucket in the same region.
- Event type (required): it supports "file upload" and "file deletion" as well as finer-grained upload and deletion events. For specific event types, please see the table below. The event type determines when the trigger triggers the function. For example, if "File upload" is selected, the function will be triggered when there is a file uploaded to the COS bucket.

Event type	Description
cos:ObjectCreated:*	All upload events mentioned below can trigger the function.
cos:ObjectCreated:Put	The function will be triggered when a file is created through the `Put Object` API.
cos:ObjectCreated:Post	The function will be triggered when a file is created through the `Post Object` API.
cos:ObjectCreated:Copy	The function will be triggered when a file is created through the `Put Object - Copy` API.
cos:ObjectCreated:CompleteMultipartUpload	The function will be triggered when a file is created through the `CompleteMultipartUpload` API.
cos:ObjectCreated:Origin	The function will be triggered when an object is created through COS origin-pull.
cos:ObjectCreated:Replication	The function will be triggered when an object is created through cross-region replication.

cos:ObjectRemove:*	All deletion events mentioned below can trigger the function.
cos:ObjectRemove:Delete	The function will be triggered when an object in a bucket for which versioning is not enabled is deleted through the `Delete Object` API, or an object on a specified version is deleted with `versionid`.
cos:ObjectRemove:DeleteMarkerCreated	The function will be triggered when an object in a bucket for which versioning is enabled or suspended is deleted through the `Delete Object` API.
cos:ObjectRestore:Post	The function will be triggered when an archive restoration job is created.
cos:ObjectRestore:Completed	The function will be triggered when an archive restoration job is completed.

- Prefix filtering (optional): Prefix filtering is usually used to filter file events in a specified directory. For example, if the prefix to be filtered is `test/`, only file events in the `test/` directory can trigger the function, while those in the `hello/` directory cannot.
- Suffix filtering (optional): Suffix filtering is usually used to filter file events in a specified type or with a specified suffix. For example, if the suffix to be filtered is `.jpg`, only file events of the `.jpg` type can trigger the function, while those of the `.png/` type cannot.

COS Trigger Use Limits

To avoid errors in the production and delivery of COS events, COS limits each bucket to one triggerable function per event (such as file upload/file deletion) and prefix and suffix filtering combination. Therefore, when creating a COS trigger, do not configure the same rules for the same COS Bucket. For example, you can configure the "Created: *" event trigger for function A for the test Bucket (without configuring filtering rules), then the upload event for this test Bucket cannot be bound to other functions, these events include (Created:Put, Created:Post, etc.), but you can configure the "ObjectRemove" event trigger for function B for the test Bucket.
When using prefix and suffix filtering rules, to ensure the uniqueness of the trigger event for the same Bucket, it is important to note that the same Bucket cannot use overlapping prefixes, overlapping suffixes, or overlapping combinations of prefixes and suffixes to define filtering rules for the same event type. For example, if you have configured the "Created: *" event trigger for function A for the test Bucket with a prefix filter of "Log", then you cannot create another "Created: *" event trigger with a prefix filter of "Log" for this test Bucket.
In addition, COS triggers can only trigger functions in the same region; for example, for an SCF function created in the Guangzhou region, you can only select a COS bucket in the Guangzhou region (South China) when configuring a COS trigger. If you want to trigger a function through COS bucket events in a specific region, create a function in that region.
A COS trigger has limits in two dimensions: SCF and COS, as detailed below:

- SCF dimension: one function can be bound to 10 COS triggers at most.
- COS dimension: Only one function can be bound to the same event and prefix/suffix rules in a single COS bucket.

Event Message Structure for COS Trigger

When an object creation or deletion event occurs in the specified COS bucket, event data will be sent to the bound SCF function in JSON format as shown below.

```
{
  "Records": [{
    "cos": {
      "cosSchemaVersion": "1.0",
      "cosObject": {
        "url": "http://testpic-1253970026.cos.ap-chengdu.myqcloud.com/testfile",
        "meta": {
          "x-cos-request-id": "NWMxOWY4MGFfMjViMjU4NjRfMTUyMVxxxxxxxxx=",
          "Content-Type": "",
          "x-cos-meta-mykey": "myvalue"
        },
        "vid": "",
        "key": "/1253970026/testpic/testfile",
        "size": 1029
      },
      "cosBucket": {
        "region": "cd",
        "name": "testpic",
        "appid": "1253970026"
      },
      "cosNotificationId": "unkown"
    },
    "event": {
      "eventName": "cos:ObjectCreated:*",
      "eventVersion": "1.0",
      "eventTime": 1545205770,
      "eventSource": "qcs::cos",
      "requestParameters": {
        "requestSourceIP": "192.168.15.101",
        "requestHeaders": {
          "Authorization": "q-sign-algorithm=sha1&q-ak=xxxxxxxxxxxxxx&q-sign-
time=1545205709;1545215769&q-key-time=1545205709;1545215769&q-header-list=host;x-cos-
storage-class&q-url-param-list=&q-signature=xxxxxxxxxxxxxx"
        }
      },
      "eventQueue": "qcs:0:scf:cd:appid/1253970026:default.printevent.$LATEST",
      "reservedInfo": "",
      "reqid": 179398952
    }
  }]
}
```

The data structures are detailed as below:

Structure	Content

Records	List structure. There may be multiple messages merged in the list.
event	This records the event information, including event version, event source, event name, time, queue information, request parameters, and request ID.
cos	This records the COS information corresponding to the event.
cosBucket	This records the specific bucket where the event occurred, including the bucket name, region, and the user's APPID. The APPID can be obtained from the Account Information page.
cosObject	This records the object of the specific event, including object file path, size, custom metadata, and access URL.

Relevant Examples

The following is a sample COS trigger in Java for your reference:

https://github.com/tencentyun/scf-demo-java/blob/master/src/main/java/example/Cos.java

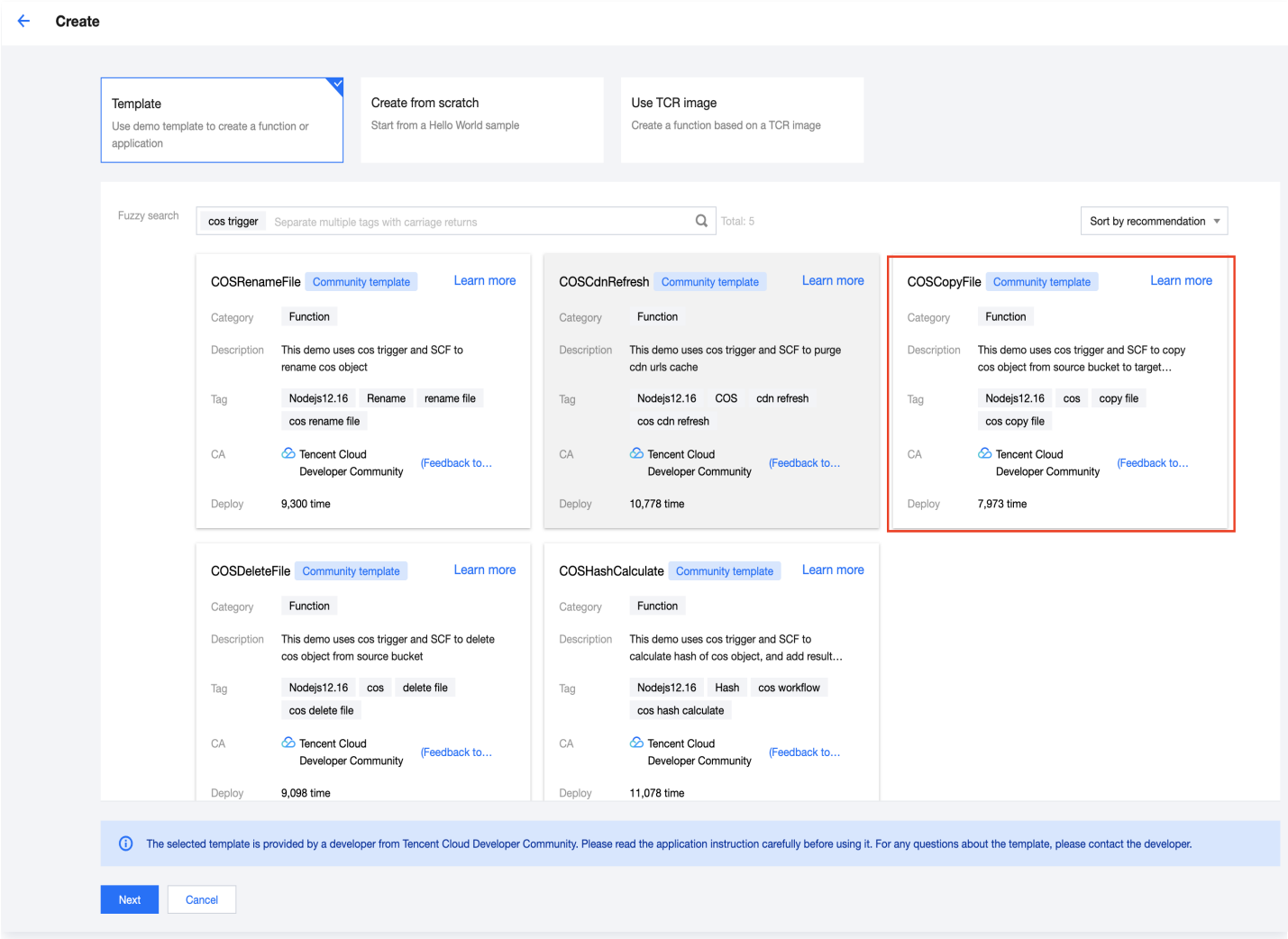
Usage

Last updated: 2023-09-27 19:27:35

This document will guide you on how to create a COS trigger and successfully invoke a function.

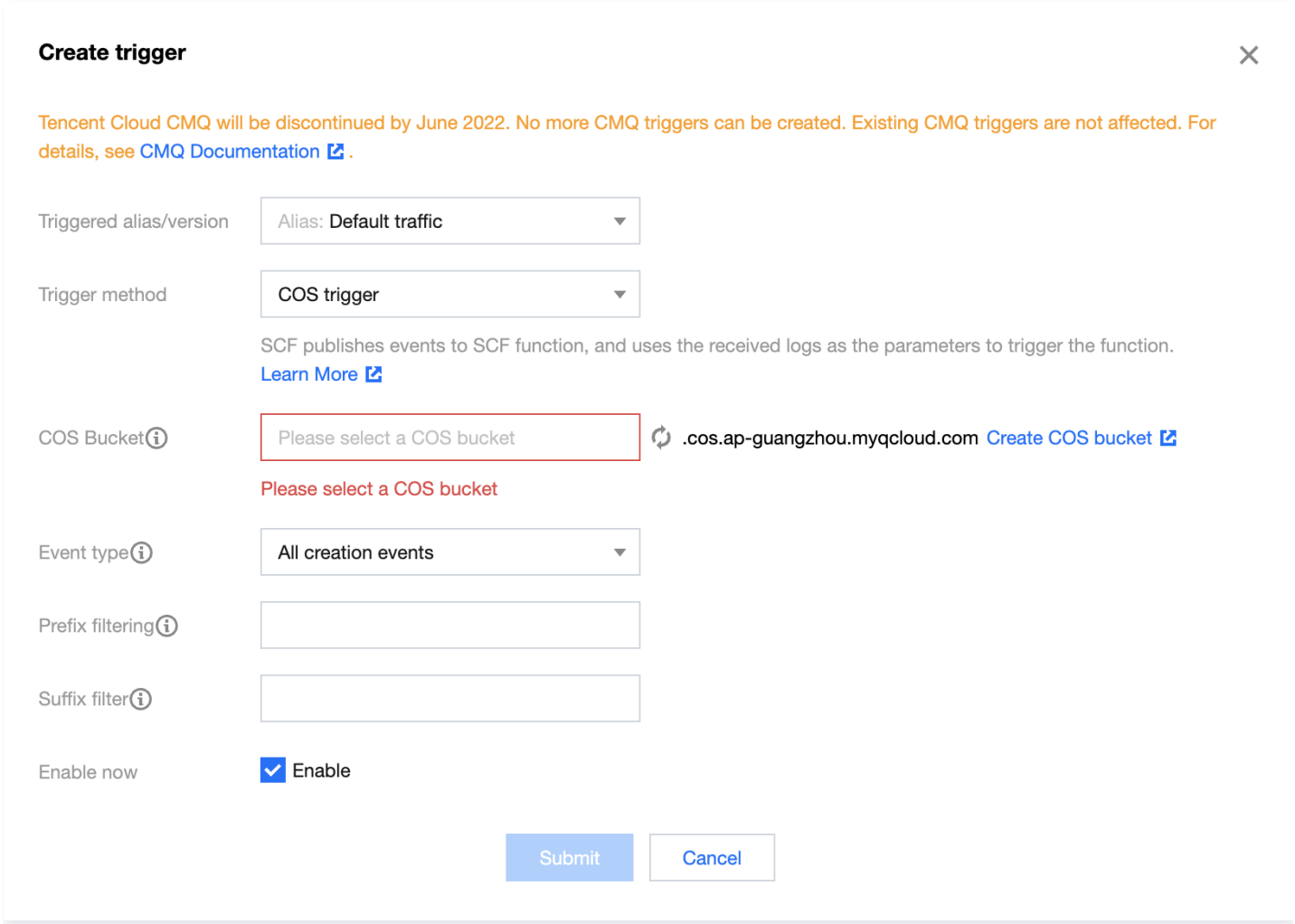
Step 1. Create a function

Log in to the Serverless console, complete the upload and deployment of your function code on the new function page. For more details, refer to Creating an Event Function Using the Console.
Using the COS sample template as an example, create a function project. The template defaults to configuring the trigger during the creation process. However, you can also configure it after the creation is complete. This guide will illustrate the latter scenario:



Step 2. Configure a trigger

After selecting the **COS trigger**, follow the guide to configure the specified Bucket, trigger event type, and other information to complete the creation of the trigger:



Step 3. Manage the trigger

Upon completion, the information of the created trigger can be viewed on the "Trigger Management" page.

Timer Trigger
Timer Trigger Description

Last updated: 2023-09-27 19:29:27

You can write an SCF function to handle a scheduled task (which can be triggered in seconds). The timer will automatically trigger the function at the specified time. Timer triggers have the following characteristics:

- **Push Model**: When the timer reaches the specified time, it directly calls the Invoke interface of the related function to trigger it. This event source mapping is stored in the SCF function.
- **Async invocation**: a timer trigger always invokes a function asynchronously, and the result is not returned to the invoker. For more information on invocation types, please see Invocation Types.

Timer Trigger Attributes

- Timer name (required): it can contain up to 60 characters out of `a-z`, `A-Z`, `0-9`, `-`, and `_` and must begin with a letter and be unique under the same function.
- Triggering cycle (required): this is the specified function triggering time. You can use the default value in the console or customize a standard cron expression to decide when to trigger the function. For more information on cron expressions, please see below.
- Input parameter (optional): it supports a string up to 4KB, which can be obtained in the "event" parameter of the entry function.

Cron Expression

The Timer Trigger now supports second-level triggering. When creating a Timer Trigger, users can customize the trigger time using standard Cron expressions.

Cron Expression Syntax

A cron expression has seven required fields, separated by spaces.

First	Second	Third	Fourth	Fifth	Sixth	Seventh
Second	Minute	Hour	Day	Month	Week	Year

Each field has a corresponding value range:

Field	Value	Wildcards	Symbols
Second	An integer between 0 and 59	, - * /	None
Minute	An integer between 0 and 59	, - * /	None
Hour	An integer between 0 and 23	, - * /	None
Day	An integer between 1 and 31 (the number of days in the month needs to be considered)	, - * /	? L W

Month	An integer between 1 and 12 or JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC	, – * /	None
Week	An integer between 0 and 6 or SUN, MON, TUE, WED, THU, FRI, SAT. Where 0 denotes Sunday, 1 denotes Monday, and so forth.	, – * /	? L #
Year	An integer between 1970 and 2099	, – * /	None

Wildcards

Wildcards	Description
Comma	Represents the union of characters separated by commas. For instance, in the "hour" field, 1,2,3 denotes 1 o'clock, 2 o'clock, and 3 o'clock.
Hyphen	Includes all values within the specified range. For example, in the "day" field, 1 – 15 encompasses the 1st to the 15th of the specified month.
Asterisk (*)	Represents all values. In the "hour" field, * signifies every hour.
Slash (/)	Specifies increments. In the "minute" field, input 1/10 to specify a repetition every ten minutes starting from the first minute. For instance, the 11th minute, the 21st minute, the 31st minute, and so forth.

Symbols

Symbols	Description
?	Can only be used in the "day" and "week" fields, indicating that no specific day or day of the week is specified. For example, when the "day" field is set to 2 and the "week" field is set to "?", it means that only the date is judged to be the 2nd, without specifying what day of the week the 2nd is.
L	Can only be used in the "day" and "week" fields, indicating "last". In the "day" field, it represents the last day of the month; in the "week" field, it represents the last day of the week, with a range limit of integers from 0 to 6. For example, 5L signifies the last Friday.
W	Can only be used in the "day" field, indicating the workday (Monday to Friday) closest to a certain day. For example, 10W signifies the workday closest to the 10th. Combined with the "month" field, it can represent the workday closest to a certain day in a certain month; in the "day" field, the combination of "L" and "W" as "LW" represents the last workday of the month.
#	Can only be used in the "week" field, and numbers must be given before and after "#", indicating which week of the week. The number before "#" represents the day of the week, with a range of integers from 0 to 6; the number after "#" represents which one, with a range of integers from 1 to 5. For example, 2#3 signifies the third Tuesday.

Precautions

1. When specific values are simultaneously specified in the "day" and "week" fields of the Cron expression, they are in an "or" relationship, that is, the conditions of both are separately effective.
2. When using the "W" symbol, the specified day and its nearest workday are in the same month and will not cross the month. For example, if "1W" is specified, it means the workday closest to the 1st. If the 1st is a Saturday, the workday closest to the 1st is the 3rd of this month (Monday), not the last day of the previous month (Friday).

Samples

Below are some examples of cron expressions and their meanings:

Expression	Description
`*/5 * * * * * *`	Triggers once every 5 seconds
`0 15 10 1 * ? *`	Indicates triggering at 10:15 AM on the first day of every month
`0 15 10 ? * MON-FRI *`	Indicates triggering at 10:15 AM every weekday from Monday to Friday
`0 0 10,14,16 * * * *`	Triggers every day at 10 am, 2 pm, and 4 pm
`0 */30 9-17 * * * *`	Triggers every half hour from 9 am to 5 pm every day
`0 0 12 ? * WED *`	Triggers at 12:00 noon every Wednesday
`0 0 0 L * * *`	Indicates triggering at midnight on the last day of every month
`0 0 0 * 5 4L *`	Indicates triggering at midnight on the last Thursday of May
`0 0 0 12W 6 * *`	Indicates triggering at midnight on the business day closest to the 12th of June
`0 0 0 ? * 2#3 *`	Indicates triggering at midnight on the third Tuesday of every month
`0 0 0 LW * ? *`	Indicates triggering at midnight on the last business day of every month

Input Parameters of Timer Triggers

When a timer trigger triggers a function, the following data structures will be encapsulated in `event` and passed to the function. In addition, you can specify to pass the `message` for a timer trigger, which is empty by default.

```
{
  "Type":"Timer",
  "TriggerName":"EveryDay",
  "Time":"2019-02-21T11:49:00Z",
  "Message":"user define msg body"
}
```

Field	Description
Type	The type of the trigger, with the value being `Timer`.
TriggerName	The name of the timer trigger. It can contain up to 60 characters, including `[a-z]`, `[A-Z]`, `[0-9]`, `[-]`, and `[_]`. It must start with a letter, and multiple timer triggers with the same name under one function are not supported.
Time	Creation time of the trigger, in timezone 0.
Message	String Type.

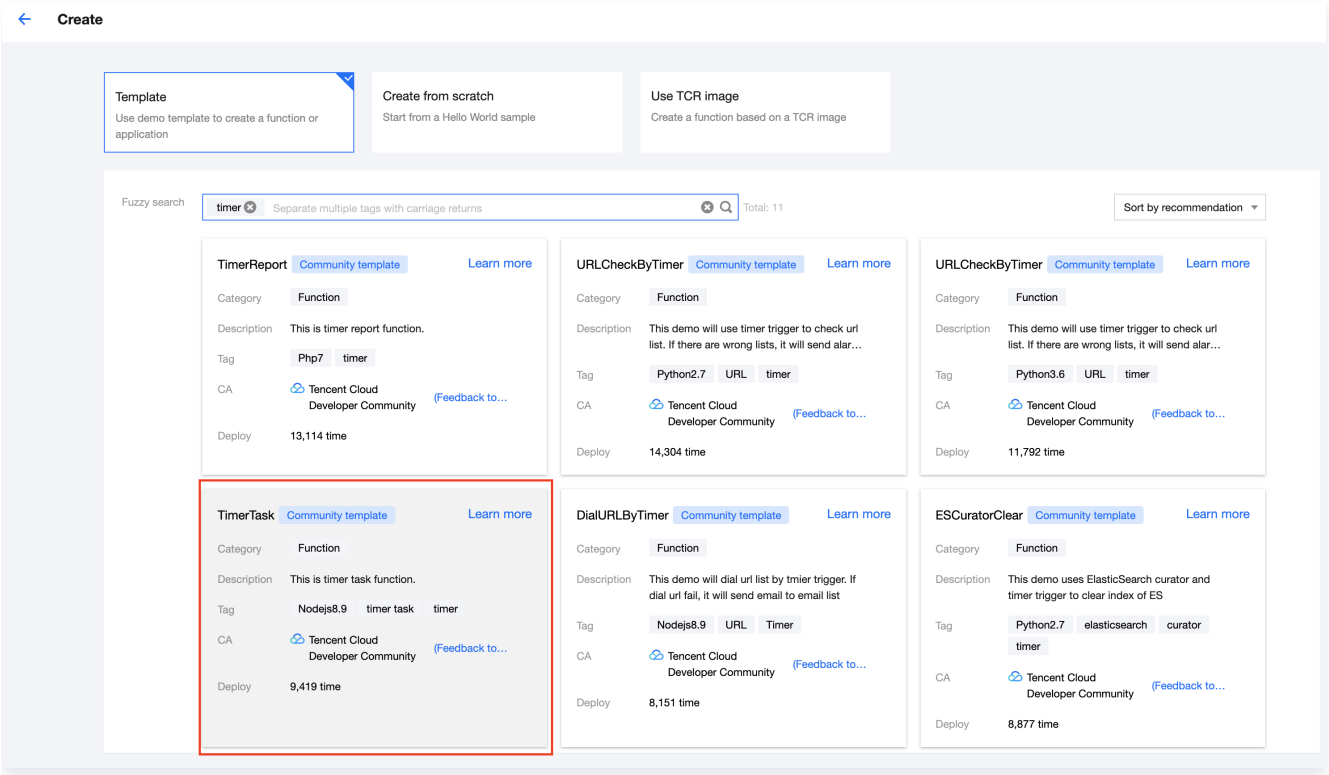
Usage

Last updated: 2023-09-28 15:47:29

This document describes how to create a timer trigger and invoke a function.

Step 1. Create a function

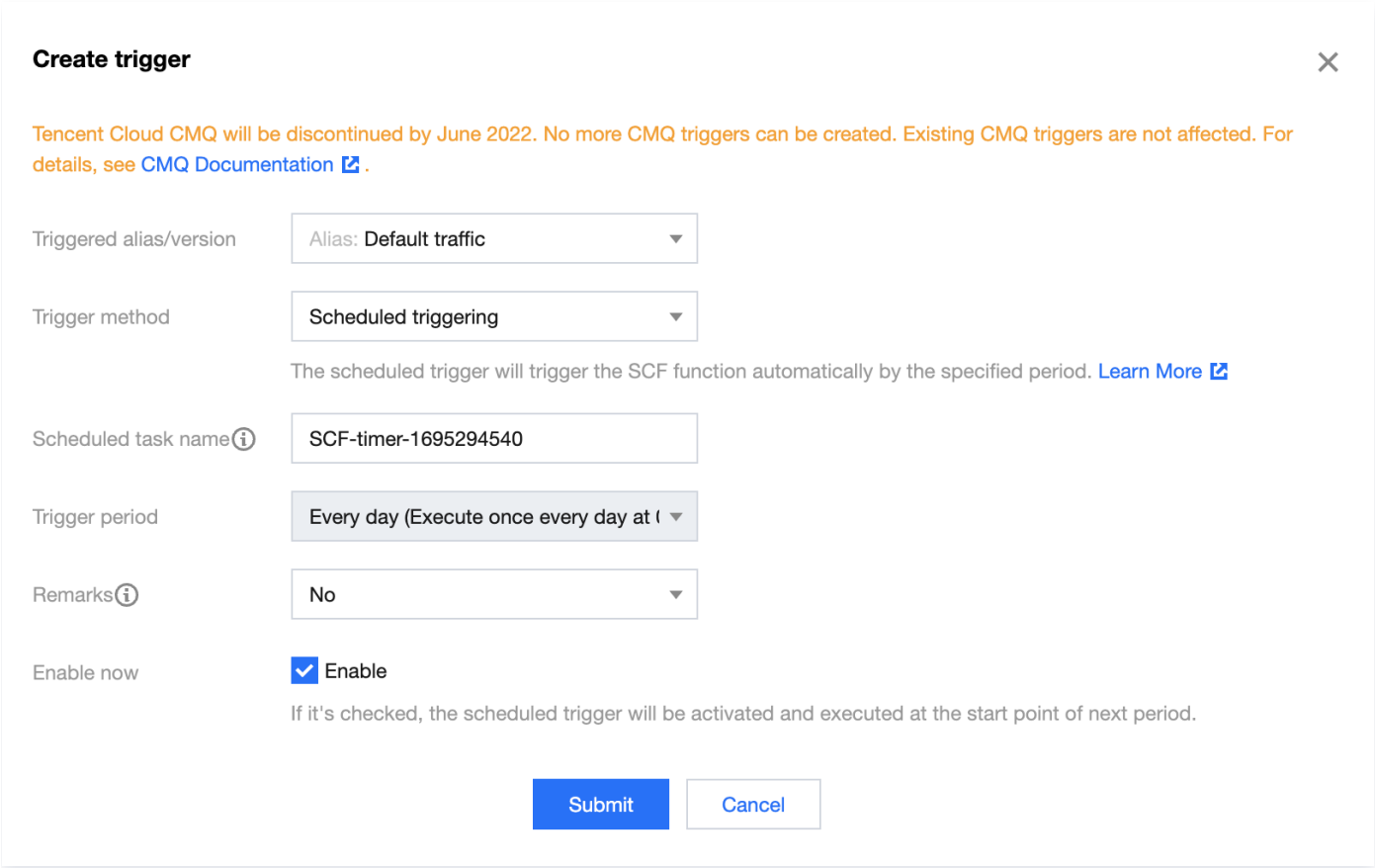
Log in to the Serverless Console, and on the new function page, complete the upload and deployment of your function code. For more details, refer to Creating an Event Function Using the Console.
This document uses the timer event sample template as an example to create a function project. The template defaults to configuring the trigger directly in the creation process, as shown in the following figure:



In practice, you can also configure the trigger after the project is created. For more details, refer to Creating a Trigger.

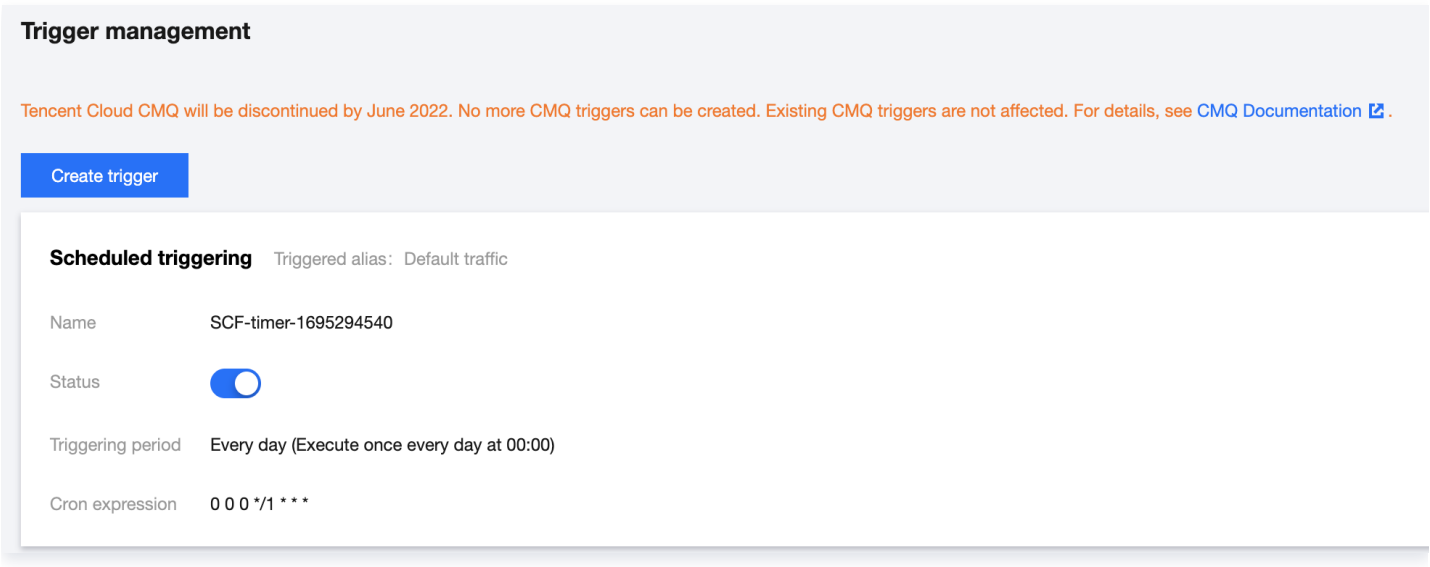
Step 2. Configure a trigger

During the trigger configuration step, after selecting **Scheduled triggering** as the trigger method, configure the timer task name, trigger cycle, and other information to complete the creation of the trigger:



Step 3. Manage the trigger

Upon completion, you can view the created trigger information on the "Trigger management" page and toggle the trigger on and off.



CKafka Trigger
CKafka Trigger Description

Last updated: 2023-09-27 19:30:19

You can write an SCF function to process messages received in the specific CKafka instance. The SCF backend can consume the messages in CKafka as a consumer and pass them to the function.
Characteristics of CKafka triggers:

- **Pull model**: the backend module of SCF acts as a consumer, connects to the CKafka instance, and consumes messages. When the backend module gets the message, it will encapsulate the message into data structures and invoke the specified function to pass the message data to the function.
- **Sync invocation**: a CKafka trigger always invokes a function synchronously. For more information on invocation types, please see Invocation Types.

> **Note**
> - For execution errors (including user code errors and runtime errors), the CKafka trigger will retry according to the configured retry times, which is 10,000 by default.
> - For system errors, the CKafka trigger will continue to retry in an exponential backoff manner until it succeeds.

CKafka Trigger Attributes

- **CKafka instance**: configure the CKafka instance you want to connect to. It can only be an instance in the same region as the function.
- **Topic**: it can be an existing topic in the CKafka instance (only topics without ACL are supported).
- **Maximum messages**: the maximum number of messages that can be pulled and batch delivered to the current function at a time, which can be up to 10,000 currently. According to the message size and writing speed, the number of messages delivered when the function is triggered each time may not always reach the maximum number; instead, it is a variable value between 1 and the maximum number.
- **Start Point**: the start position from which the trigger consumes messages. Valid values: latest (default), oldest, specified time point.
- **Retry Attempts**: the maximum number of retries when an error occurs during function execution (including user code errors and runtime errors).

CKafka Consumption and Message Delivery

CKafka does not push messages actively. The consumer needs to pull messages and consume them. Therefore, if a CKafka trigger is configured, the SCF backend will launch a CKafka consumption module as the consumer to create an independent consumer group in CKafka for message consumption.
After consuming messages, the SCF backend consumption module will encapsulate them into event structures according to the **timeout period**, **accumulated messages**, and **maximum messages** and then initiate function invocation (sync invocation). Applicable limits are as follows:

- **Timeout period**: the current timeout period of the consumption module on the backend of SCF is 60 seconds, which avoids waiting for too long before consuming. For example, if the CKafka topic has

very few messages written in, and the consumption module fails to collect the configured maximum number of messages in 60 seconds, then the function invocation will still be initiated.

- **Event size limit for sync invocation**: 6 MB. For more information, please see Limits. If the messages in the CKafka topic are large (for example, one single message is over 6 MB in size), then due to the 6 MB limit for sync invocation, there will be only one message in the event structure passed to the function instead of the user-configured maximum number of messages.
- **Maximum messages**: this is the same as the user-defined CKafka trigger attribute, which can be up to 10,000 currently.

The consumption module on the backend of SCF will loop this process and ensure the order of message consumption, that is, the next batch of messages will be consumed only after the previous batch is completely consumed (sync invocation).

> **Note**
> - In this process, the number of encapsulated messages is different in each event structure, which ranges from **1 to the maximum number**. If the maximum number of messages is too high, there may be cases where the number of messages in an event structure will never reach the maximum number.
> - After the event content is obtained by the function, each message can be guaranteed for processing by loop handling, and it should not be assumed that the number of messages passed each time is constant.

Event Message Structure for CKafka Trigger

When the specified CKafka topic receives a message, the backend consumption module of SCF will consume the message and encapsulate it into an event in JSON format like the one below, which will trigger the bound function and pass the data content as input parameters to the function.

```
{
 "Records": [
  {
   "Ckafka": {
    "topic": "test-topic",
    "Partition":1,
    "offset":36,
    "msgKey": "None",
    "msgBody": "Hello from Ckafka!"
   }
  },
  {
   "Ckafka": {
    "topic": "test-topic",
    "Partition":1,
    "offset":37,
    "msgKey": "None",
    "msgBody": "Hello from Ckafka again!"
   }
```

```
    }
  ]
}
```

The data structures are detailed as below:

Structure	Content
Records	List structure. There may be multiple messages merged in the list
Ckafka	Identifies the event source as CKafka
topic	Message source topic
partition	Partition ID of message source
offset	Consumption offset number
msgKey	Message key
msgBody	Message content

FAQs

What should I do if a lot of CKafka messages heap up?

If a CKafka trigger is configured, the SCF backend will launch a CKafka consumption module as the consumer to create an independent consumer group in CKafka for message consumption. In addition, the number of consumption modules is equal to the number of partitions in the CKafka topic.
If a lot of CKafka messages heap up, you need to increase the consumption capability in the following ways:

- Increase the number of partitions of the CKafka topic. The consumption capability of the function is proportional to the number of partitions. The CKafka consumption modules on the backend of the function will automatically match the number of CKafka topic partitions, that is, the consumption capability can be improved by adding partitions.
- Optimize the execution duration of the function. The shorter the duration, the higher the consumption capability. If the duration becomes longer (for example, the database in the function needs to be written but the response of the database becomes slower), the consumption speed will decrease.

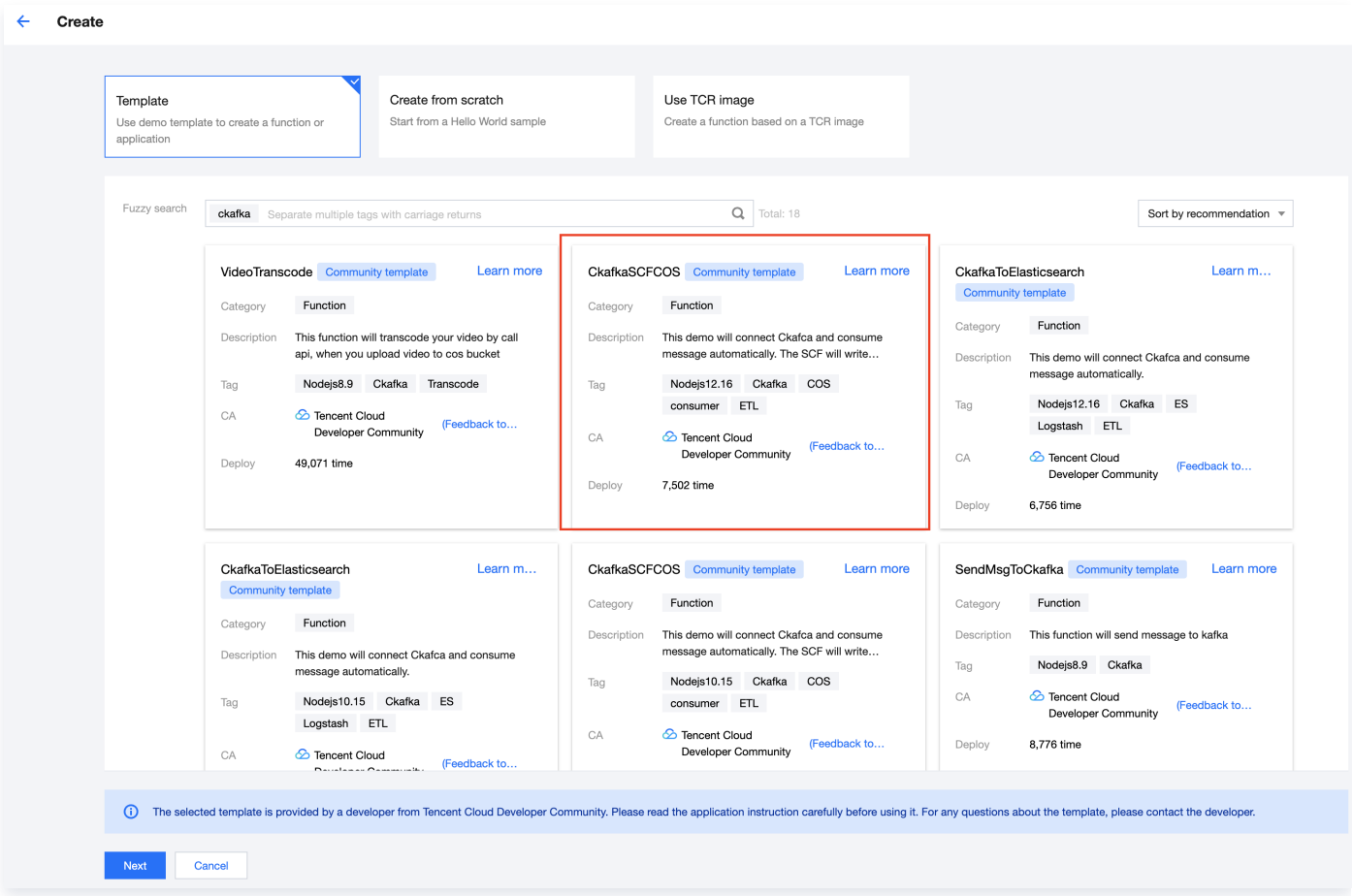
Usage

Last updated: 2023-09-28 15:51:51

This document describes how to create a CKafka trigger and invoke a function.

Step 1. Create a function

1. Log in to the SLS console.
2. On the Create Function page, select **Template** to create a new function. In the search box, filter by `CKafka` and select "CKafka message dump to COS". As shown in the following figure:

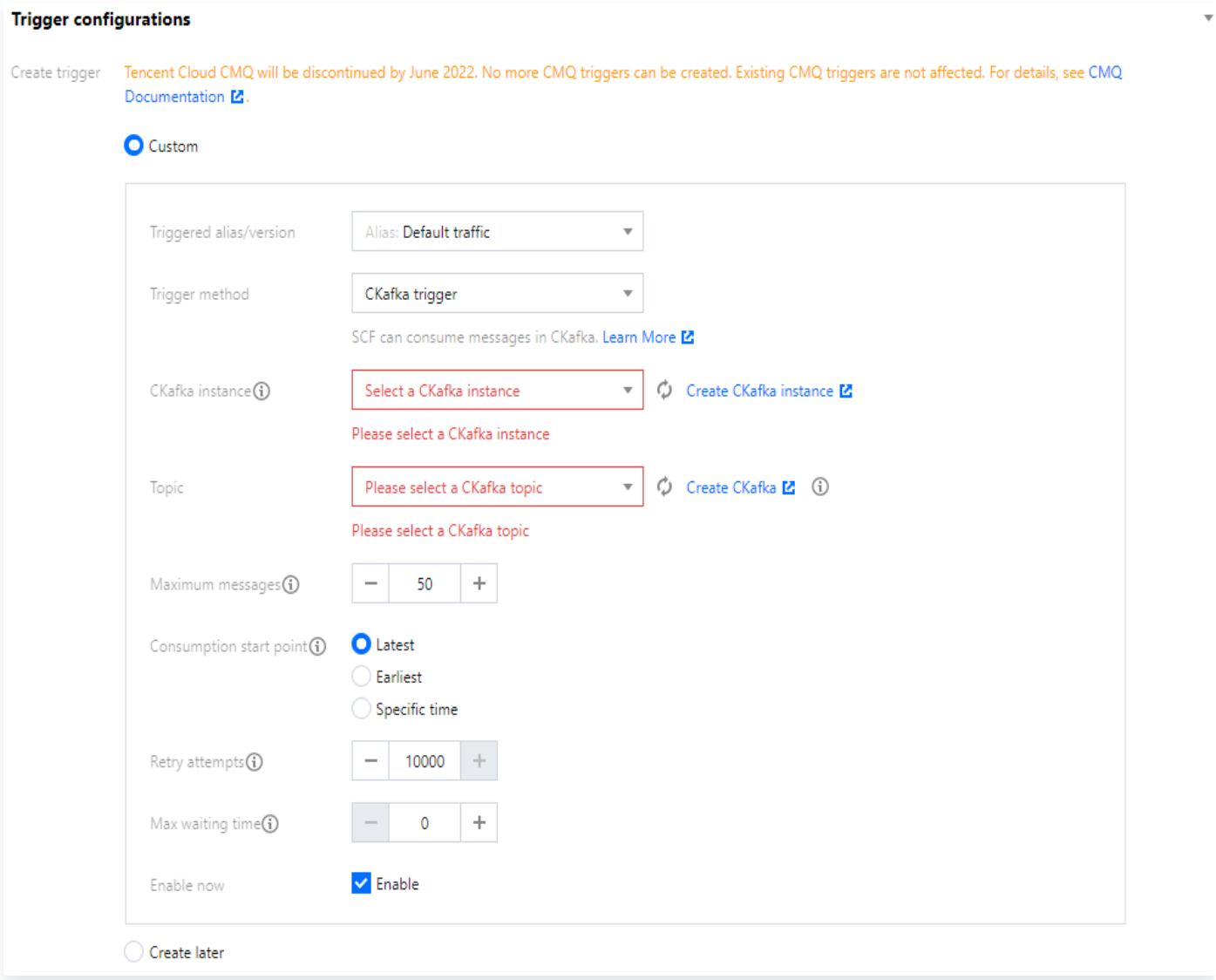


3. Click **Next**.

Step 2. Configure a trigger

On the Function Configuration page, fill in the basic function settings.

1. In "Trigger configurations", select **Custom** to create a new trigger. As shown in the following figure:



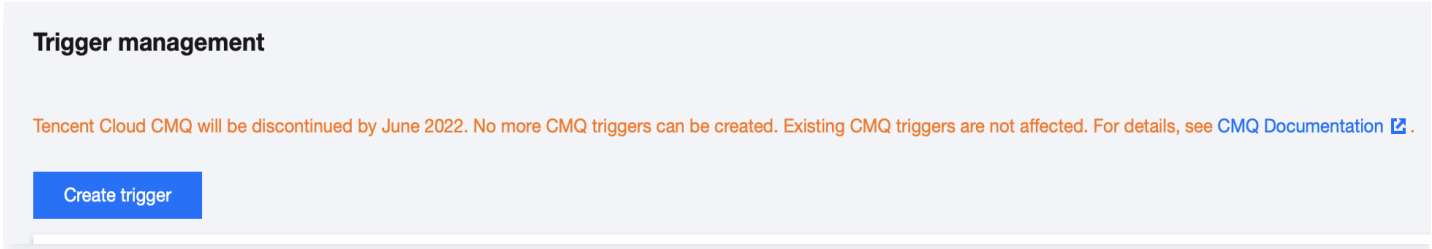
2. After selecting **CKafka trigger**, follow the instructions to configure the name, topic, and other information of the CKafka instance from which the messages originate. You can also choose to Create CKafka.

> **Note**
> Make sure that your function and CKafka instance are in the same VPC.

3. Click **Complete**.

Step 3. Manage the trigger

After the function is created, navigate to the function details page. You can view the information of the created trigger in "Trigger management". You also have the option to enable or disable the trigger.



MPS Trigger

Last updated: 2023-09-27 19:43:00

Video Processing (Media Processing Service, MPS) is a cloud-based transcoding and audio-visual processing service designed for massive multimedia data. You can write cloud functions to handle callback information in MPS, assisting in the dumping, delivery, and processing of relevant events and subsequent content in video tasks through the reception of related callbacks.
The MPS trigger possesses the following characteristics:

- **Push Model**: The MPS trigger monitors callback information from video processing and pushes event data to the SCF function through a single trigger mechanism.
- **Asynchronous Invocation**: The MSP trigger always uses the asynchronous invocation type to call functions, and the results are not returned to the caller. For more information about invocation types, please refer to Invocation Types.

MPS Trigger Attributes

- **Event Types**: The MPS trigger pushes Event events based on account dimension event types. Currently, it supports triggering by two types of events: Workflow Task (WorkflowTask) and Video Editing Task (EditMediaTask).
- **Event Handling**: The MPS trigger uses events generated at the service level as the event source, without distinguishing attributes such as region or resources. Each account can only create one MPS trigger across all regions. If multiple functions need to process tasks in parallel, please refer to Inter-Function SDK Calls.

MPS Trigger Event Message Structure

When the specified MPS trigger receives a message, the event structure and fields are exemplified by WorkflowTask, as shown below:


```
{
  "EventType":"WorkflowTask",
  "WorkflowTaskEvent":{
    "TaskId":"245**654-WorkflowTask-f46dac7fe2436c47****d71946986t0",
    "Status":"FINISH",
    "ErrCode":0,
    "Message":"",
    "InputInfo":{
      "Type":"COS",
      "CosInputInfo":{
        "Bucket":"macgzptest-125****654",
        "Region":"ap-guangzhou",
        "Object":"/dianping2.mp4"
      }
    },
    "MetaData":{
      "AudioDuration":11.261677742004395,
      "AudioStreamSet":[
```



```
            {
               "Bitrate":127771,
               "Codec":"aac",
               "SamplingRate":44100
            }
         ],
         "Bitrate":2681468,
         "Container":"mov,mp4,m4a,3gp,3g2,mj2",
         "Duration":11.261677742004395,
         "Height":720,
         "Rotate":90,
         "Size":3539987,
         "VideoDuration":10.510889053344727,
         "VideoStreamSet":[
            {
               "Bitrate":2553697,
               "Codec":"h264",
               "Fps":29,
               "Height":720,
               "Width":1280
            }
         ],
         "Width":1280
      },
      "MediaProcessResultSet":[
         {
            "Type":"Transcode",
            "TranscodeTask":{
               "Status":"SUCCESS",
               "ErrCode":0,
               "Message":"SUCCESS",
               "Input":{
                  "Definition":10,
                  "WatermarkSet":[
                     {
                        "Definition":515247,
                        "TextContent":"",
                        "SvgContent":""
                     }
                  ],
                  "OutputStorage":{
                     "Type":"COS",
                     "CosOutputStorage":{
                        "Bucket":"gztest-125****654",
                        "Region":"ap-guangzhou"
                     }
                  },
                  "OutputObjectPath":"/dasda/dianping2_transcode_10",
                  "SegmentObjectName":"/dasda/dianping2_transcode_10_{number}",
                  "ObjectNumberFormat":{
                     "InitialValue":0,
```



```
                "Increment":1,
                "MinLength":1,
                "PlaceHolder":"0"
            }
        },
        "Output":{
            "OutputStorage":{
                "Type":"COS",
                "CosOutputStorage":{
                    "Bucket":"gztest-125****654",
                    "Region":"ap-guangzhou"
                }
            },
            "Path":"/dasda/dianping2_transcode_10.mp4",
            "Definition":10,
            "Bitrate":293022,
            "Height":320,
            "Width":180,
            "Size":401637,
            "Duration":11.26200008392334,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Md5":"31dcf904c03d0cd78346a12c25c0acc9",
            "VideoStreamSet":[
                {
                    "Bitrate":244608,
                    "Codec":"h264",
                    "Fps":24,
                    "Height":320,
                    "Width":180
                }
            ],
            "AudioStreamSet":[
                {
                    "Bitrate":48414,
                    "Codec":"aac",
                    "SamplingRate":44100
                }
            ]
        }
    },
    "AnimatedGraphicTask":null,
    "SnapshotByTimeOffsetTask":null,
    "SampleSnapshotTask":null,
    "ImageSpriteTask":null
},
{
    "Type":"AnimatedGraphics",
    "TranscodeTask":null,
    "AnimatedGraphicTask":{
        "Status":"FAIL",
        "ErrCode":30010,
```

```
            "Message":"TencentVodPlatErr Or Unkown",
            "Input":{
               "Definition":20000,
               "StartTimeOffset":0,
               "EndTimeOffset":600,
               "OutputStorage":{
                  "Type":"COS",
                  "CosOutputStorage":{
                     "Bucket":"gztest-125****654",
                     "Region":"ap-guangzhou"
                  }
               },
               "OutputObjectPath":"/dasda/dianping2_animatedGraphic_20000"
            },
            "Output":null
         },
         "SnapshotByTimeOffsetTask":null,
         "SampleSnapshotTask":null,
         "ImageSpriteTask":null
      },
      {
         "Type":"SnapshotByTimeOffset",
         "TranscodeTask":null,
         "AnimatedGraphicTask":null,
         "SnapshotByTimeOffsetTask":{
            "Status":"SUCCESS",
            "ErrCode":0,
            "Message":"SUCCESS",
            "Input":{
               "Definition":10,
               "TimeOffsetSet":[

               ],
               "WatermarkSet":[
                  {
                     "Definition":515247,
                     "TextContent":"",
                     "SvgContent":""
                  }
               ],
               "OutputStorage":{
                  "Type":"COS",
                  "CosOutputStorage":{
                     "Bucket":"gztest-125****654",
                     "Region":"ap-guangzhou"
                  }
               },
               "OutputObjectPath":"/dasda/dianping2_snapshotByOffset_10_{number}",
               "ObjectNumberFormat":{
                  "InitialValue":0,
                  "Increment":1,
```



```
                "MinLength":1,
                "PlaceHolder":"0"
            }
        },
        "Output":{
            "Storage":{
                "Type":"COS",
                "CosOutputStorage":{
                    "Bucket":"gztest-125****654",
                    "Region":"ap-guangzhou"
                }
            },
            "Definition":0,
            "PicInfoSet":[
                {
                    "TimeOffset":0,
                    "Path":"/dasda/dianping2_snapshotByOffset_10_0.jpg",
                    "WaterMarkDefinition":[
                        515247
                    ]
                }
            ]
        }
    },
    "SampleSnapshotTask":null,
    "ImageSpriteTask":null
},
{
    "Type":"ImageSprites",
    "TranscodeTask":null,
    "AnimatedGraphicTask":null,
    "SnapshotByTimeOffsetTask":null,
    "SampleSnapshotTask":null,
    "ImageSpriteTask":{
        "Status":"SUCCESS",
        "ErrCode":0,
        "Message":"SUCCESS",
        "Input":{
            "Definition":10,
            "OutputStorage":{
                "Type":"COS",
                "CosOutputStorage":{
                    "Bucket":"gztest-125****654",
                    "Region":"ap-guangzhou"
                }
            },
            "OutputObjectPath":"/dasda/dianping2_imageSprite_10_{number}",
            "WebVttObjectName":"/dasda/dianping2_imageSprite_10",
            "ObjectNumberFormat":{
                "InitialValue":0,
                "Increment":1,
```



```
                "MinLength":1,
                "PlaceHolder":"0"
            }
        },
        "Output":{
            "Storage":{
                "Type":"COS",
                "CosOutputStorage":{
                    "Bucket":"gztest-125****654",
                    "Region":"ap-guangzhou"
                }
            },
            "Definition":10,
            "Height":80,
            "Width":142,
            "TotalCount":2,
            "ImagePathSet":[
                "/dasda/imageSprite/dianping2_imageSprite_10_0.jpg"
            ],
            "WebVttPath":"/dasda/imageSprite/dianping2_imageSprite_10.vtt"
        }
    }
  }
 ]
}
}
```


WorkflowTask Event

The detailed fields of the WorkflowTask event message body are as follows:

```
{
  "EventType":"WorkflowTask",
  "WorkflowTaskEvent":{
    // WorkflowTaskEvent Fields
  }
}
```

Detailed explanation of the WorkflowTask data structure and field content:

Name	Local Disk Types	Description
TaskId	String	Video Processing Task ID.
Status	String	Task flow status, with the following possible values: • PROCESSING: Currently being processed. • FINISH: Completed.
ErrCode	Integer	Deprecated, please use the ErrCode of each

		specific task.
Message	String	Deprecated, please use the Message of each specific task.
InputInfo	MediaInputInfo	Target file information for video processing. Note: This field may return null, indicating that no valid values can be obtained.
MetaData	MediaMetaData	Metadata of the original video. Note: This field may return null, indicating that no valid values can be obtained.
MediaProcessResultSet	Array of MediaProcessTaskResult	The execution status and result of the video processing task.
AiContentReviewResultSet	Array of AiContentReviewResult	The execution status and result of the video content moderation task.
AiAnalysisResultSet	Array of AiAnalysisResult	The execution status and result of the video content analysis task.
AiRecognitionResultSet	Array of AiRecognitionResult	The execution status and result of the video content recognition task.

EditMediaTask event.

The detailed fields of the EditMediaTask event message body are as follows:

```
{
  "EventType":"EditMediaTask",
  "EditMediaTaskEvent":{
    // Fields of EditMediaTask
   }
}
```

Detailed explanation of the EditMediaTask data structure and field content:

Name	Local Disk Types	Description
TaskId	String	Job ID.
Status	String	Task status, with the following possible values: • PROCESSING: Currently being processed. • FINISH: Completed.
ErrCode	Integer	Error code `0`: Success; any other value: Failure.
Message	String	Error message.

Input	EditMediaTaskInput	Input for the video editing task.
Output	EditMediaTaskOutput	Output for the video editing task. Note: This field may return null, indicating that no valid values can be obtained.

CLB Trigger Description

Last updated: 2023-09-27 19:44:10

You can implement backend web services by writing SCF functions and providing services through CLB, which will pass the request content as parameters to the function and return the result from the function back to the requester as the response.
Characteristics of CLB triggers:

- **Push Model**
  Upon receiving a request from the CLB side, the CLB listener will trigger the corresponding cloud function configured at the backend. Concurrently, the CLB will send the relevant request information to the triggered function in the form of an event parameter. This information includes the specific request method received, the request's path, header, query, and other content.
- **Synchronous Invocation**
  CLB triggers invoke functions synchronously. For more information about invocation types, please refer to Invocation Types.

> **Note**
> CLB accounts are divided into standard and traditional types. Traditional account types do not support binding with SCF, hence it is recommended to upgrade to a standard account type. For more details, refer to Account Type Upgrade Instructions.

CLB Trigger Configuration

CLB triggers can be configured either in the Serverless Console or in the Load Balancer Console.

SCF console

In the **SCF console**, you can add CLB triggers in trigger method, select existing CLB instances, create routing rules, and configure URL request paths.

CLB console

In the **Load Balancer Console**, when configuring routing rules, Cloud Function can be selected as the backend configuration. After selecting Cloud Function, you can choose a cloud function in the same region as the CLB. On the Load Balancer Console, you can configure and manage advanced CLB services, such as WAF protection, SNI multi-domain certificates, and elastic network interfaces.

CLB Trigger Binding Limits

- One CLB path rule can be bound to only one function, but one function can be bound to multiple CLB rules as the backend. Rules with the same path, listener, and host are regarded as the same rule and cannot be bound repeatedly.

- Currently, CLB triggers can only be bound to functions in the same region; for example, a function created in the Guangzhou region can only be bound to and triggered by CLB rules created in the Guangzhou region.

Request and Response

Request method refers to the method to process request sent from CLB to SCF, and response method refers to the method to process the returned value sent from SCF to CLB. Both request and response methods are automatically processed by the CLB trigger. When it triggers the function, data structures must be returned in the request method.

> **Note**
> `X-Vip`, `X-Vport`, `X-Uri`, `X-Method`, and `X-Real-Port` fields must be customized in the CLB console before they can be transferred. For custom configurations, see Layer-7 Custom Configuration.

Event message structure of integration request for CLB trigger

When a CLB trigger receives a request, event data will be sent to the bound function in JSON format as shown below.

> **Note**
> In the CLB trigger scenario, all requests and responses need to be transferred in JSON. For images, files, and other data, as directly passing in JSON content will cause invisible characters to be lost, Base64 encoding is required as detailed below:
> - If the `Content-type` is `text/*`, `application/json`, `application/javascript`, or `application/xml`, CLB will not transcode the body content.
> - For all other types, CLB will Base64-encode them first and then forward them.

```
{
 "headers": {
  "Content-type": "application/json",
  "Host": "test.clb-scf.com",
  "User-Agent": "Chrome",

  "X-Stgw-Time": "1591692977.774",
  "X-Client-Proto": "http",
  "X-Forwarded-Proto": "http",
  "X-Client-Proto-Ver": "HTTP/1.1",
  "X-Real-IP": "9.43.175.219",
  "X-Forwarded-For": "9.43.175.xx",

  "X-Vip": "121.23.21.xx",
  "X-Vport": "xx",
  "X-Uri": "/scf_location",
  "X-Method": "POST",
  "X-Real-Port": "44347",
 },
```

```
 "payload": {
  "key1": "123",
  "key2": "abc"
 },
}
```

The data structures are detailed as below:

Structure	Content
X-Stgw-Time	Request start timestamp
X-Forwarded-Proto	`scheme` structure of the request
X-Client-Proto-Ver	Protocol type
X-Real-IP	Client IP address
X-Forward-For	Transited Proxy IP address
X-Real-Port	Records the configured Path parameters in CLB and their actual values. (Optional, CLB custom configuration)
X-Vip	CLB VIP address (optional custom configuration of CLB)
X-Vport	CLB Vport (optional custom configuration of CLB)
X-Url	CLB request path (optional custom configuration of CLB)
X-Method	CLB request method (optional custom configuration of CLB)

Note

- The content may be increased significantly during CLB iteration. At present, it is guaranteed that the content of the data structure will only be increased but not reduced, so that the existing structure will not be compromised.
- Parameters in real requests may appear in multiple locations and can be selected based on your business needs.

Integration response

Integration response refers to the process where CLB parses the return content from the cloud function and constructs an HTTP response based on the parsed content. By utilizing integration response, you can control the status code, headers, and body content of the response through code, enabling the delivery of custom formatted content responses such as XML, HTML, JSON, or even JS content. When using integration response, it is necessary to follow the CLB trigger's integration response return data structure to ensure successful parsing, otherwise, an error message `{"errno":403,"error":"Analyse scf response failed."}` will occur.

Returned data structures of integration response for CLB trigger

When setting up CLB for integration response, a data structure similar to the following needs to be returned.

```
{
  "isBase64Encoded": false,
  "statusCode": 200,
  "headers": {"Content-Type":"text/html"},
  "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

The data structures are detailed as below:

Structure	Content
isBase64Encoded	This indicates whether the content in the `body` is Base64-encoded binary. It should be `true` or `false` in JSON format.
statusCode	HTTP return code, which should be an integer value.
headers	HTTP return header. It should contain multiple key-value objects or a `key:[value,value]` object where both key and value are strings.
body	HTTP return body.

If you need to return multiple headers with the same key, you can use a string array to describe different values; for example:

```
{
  "isBase64Encoded": false,
  "statusCode": 200,
  "headers": {"Content-Type":"text/html","Key":["value1","value2","value3"]},
  "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

TencentCloud API Trigger

Last updated: 2023-09-27 19:44:22

Overview

Users can write SCF functions to handle their own business logic and trigger cloud functions through the management interface exposed by SCF. This management interface is uniformly referred to as Cloud API in Tencent Cloud. By using the Invoke interface in the SCF Cloud API, users can trigger cloud functions on demand.
For detailed methods of calling the Cloud API interface, please refer to the Run Function API document.
The Cloud API trigger has the following characteristics:

- **Invocation Mode is Selectable**: You can define synchronous or asynchronous triggering methods based on the InvocationType parameter.
- **Custom Events**: You can define the event or data content that triggers the cloud function based on the ClientContext parameter. The content needs to be encoded in JSON format.

TencentCloud API Invocation

To trigger a cloud function via TencentCloud API, you need to:

1. Authenticate via the API interface;
2. Fill in the Common Parameters;
3. Parse the return results based on the Returned Results.

Additionally, to avoid manually constructing and parsing request content, users can directly trigger cloud functions using the TencentCloud API SDK. The TencentCloud API SDK provides implementations in Python, PHP, Java, GO, .NET, and Node.js. For specific usage of each language's SDK, refer to the TencentCloud SDK Center.

EventBridge Trigger
EventBridge Trigger Description

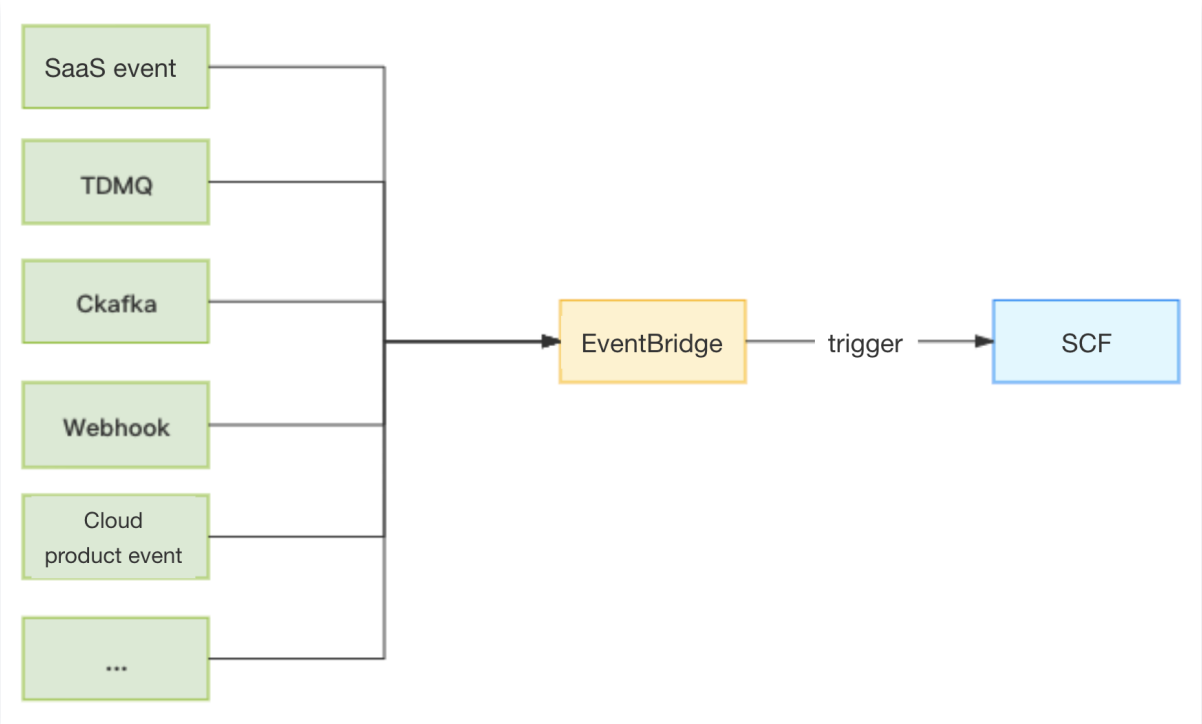
Last updated: 2023-09-28 15:55:29

Overview

Tencent Cloud EventBridge is a secure, stable, and efficient serverless event management platform. The event bus in the event center can receive real-time events and related data streams from your own applications, Software as a Service (SaaS), and Tencent Cloud services, routing these events and data to trigger cloud functions for event processing.

Trigger

EventBridge provides a universal access method for cloud event sources. After cloud service events or product events are matched through EventBridge, they can be directly sent to the specified target function, completing the function trigger and event consumption.



The EventBridge trigger exhibits the following characteristics:

- **Push Model**
  EventBridge actively pushes the received product events to the cloud function, completing the function trigger and invocation.
- **Asynchronous Invocation**
  The EventBridge trigger uses an asynchronous trigger type to invoke functions. For more information about invocation types, please refer to Invocation Types.

> **Note**
> - For execution errors (including user code errors and Runtime errors), the EventBridge trigger

defaults to two retries.
- For system errors, the EventBridge trigger will continue to retry for six hours using an exponential backoff strategy.

Scenarios

- **Common Trigger Scenarios**
  EventBridge expands the trigger scenarios of cloud functions, which can be widely used in event-driven architectures to facilitate communication between applications through event triggers.
- **Cloud-native Driven Scenarios**
  EventBridge, in conjunction with cloud functions, provides a fully lifecycle-managed and completely cloud-native event-driven solution, assisting you in rapidly architecting cloud-native event-driven architectures (EDA) and linking more application scenarios.
- **Status Change Notifications**
  EventBridge can act as a hub, receiving status changes from all applications, and then routing these application status changes to the corresponding cloud functions to complete event handling, storage, alarms, and more.

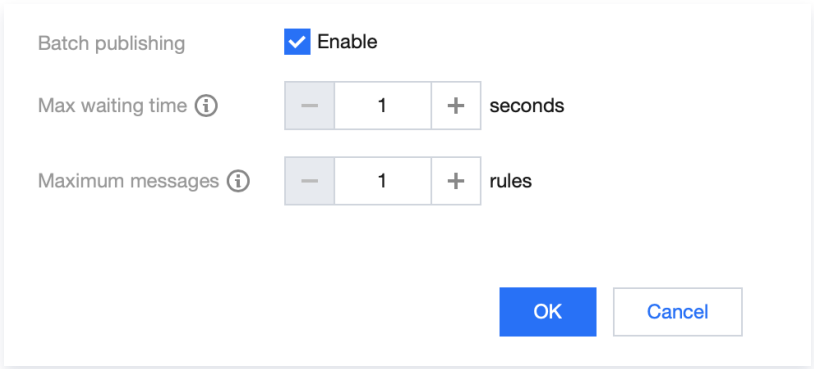
Supported Event Sources

- Tencent Distributed Message Queue (TDMQ). For more information, refer to TDMQ Trigger.
- Message Queue CKafka. For more details, refer to CKafka Trigger.
- SaaS Event Delivery (supported by Enterprise Integration Service EIS). For more details, refer to SaaS Trigger.

Event Consumption

When the delivery target is SCF, EventBridge supports batch delivery. You can choose the delivery method according to your actual business needs.
Parameters for Batch Delivery:

- **Max waiting time:** The longest waiting period for a single trigger. The range for the longest waiting time is 1-60s, with a default value of 1.
- **Maximum messages:** The maximum number of messages that can be pulled and batch delivered to the current SCF. The highest configuration currently supported is 10,000. Considering factors such as message size and write speed, the number of messages delivered each time the SCF is triggered may not necessarily reach the maximum, but fluctuates between 1 and the maximum message count.



> **Note**
> Once the batch delivery feature is enabled, events will be delivered in an array format. Please ensure that the event consumer end is adapted to this format.

Event Format when Batch Delivery is Disabled:

```
{
  "specversion": "1.0.2",
  "id": "13a3f42d-7258-4ada-da6d-023a33**",
  "type": "connector:apigw",
  "source": "apigw.cloud.tencent",
  "subjuect": "qcs::apigw:ap-guangzhou:uid1250000000/appidxxx:Serverid/Appid",
  "time": "1615430559146",
  "region": "ap-guangzhou",
  "datacontenttype": "application/json;charset=utf-8",
  "data":{
        $data_value
      }
}
```

Enabling Batch Delivery, Array Mode:

```
{
  "EventList":[
    {
      "specversion": "1.0.2",
      "id": "13a3f42d-7258-4ada-da6d-023a33**",
      "type": "connector:apigw",
      "source": "apigw.cloud.tencent",
      "subjuect": "qcs::apigw:ap-guangzhou:uid1250000000/appidxxx:Serverid/Appid",
      "time": "1615430559146",
      "region": "ap-guangzhou",
      "datacontenttype": "application/json;charset=utf-8",
      "data":{
          $data_value
       }
    },
    {
      "specversion": "1.0.2",
      "id": "13a3f42d-7258-4ada-da6d-023a33**",
      "type": "connector:apigw",
      "source": "apigw.cloud.tencent",
      "subjuect": "qcs::apigw:ap-guangzhou:uid1250000000/appidxxx:Serverid/Appid",
      "time": "1615430559146",
      "region": "ap-guangzhou",
      "datacontenttype": "application/json;charset=utf-8",
      "data":{
          $data_value
       }
```

```
    }
  ]
}
```

TDMQ Trigger

Last updated: 2023-09-28 15:59:38

Users can write serverless cloud functions to process messages received from the TDMQ message queue through the EventBridge event bus. The serverless cloud function backend module can act as a consumer to consume messages from TDMQ and pass them to the cloud function. This document will guide you on how the cloud function receives and consumes product events from TDMQ through the EventBridge event bus trigger.

Creation Process

Step 1. Create a function

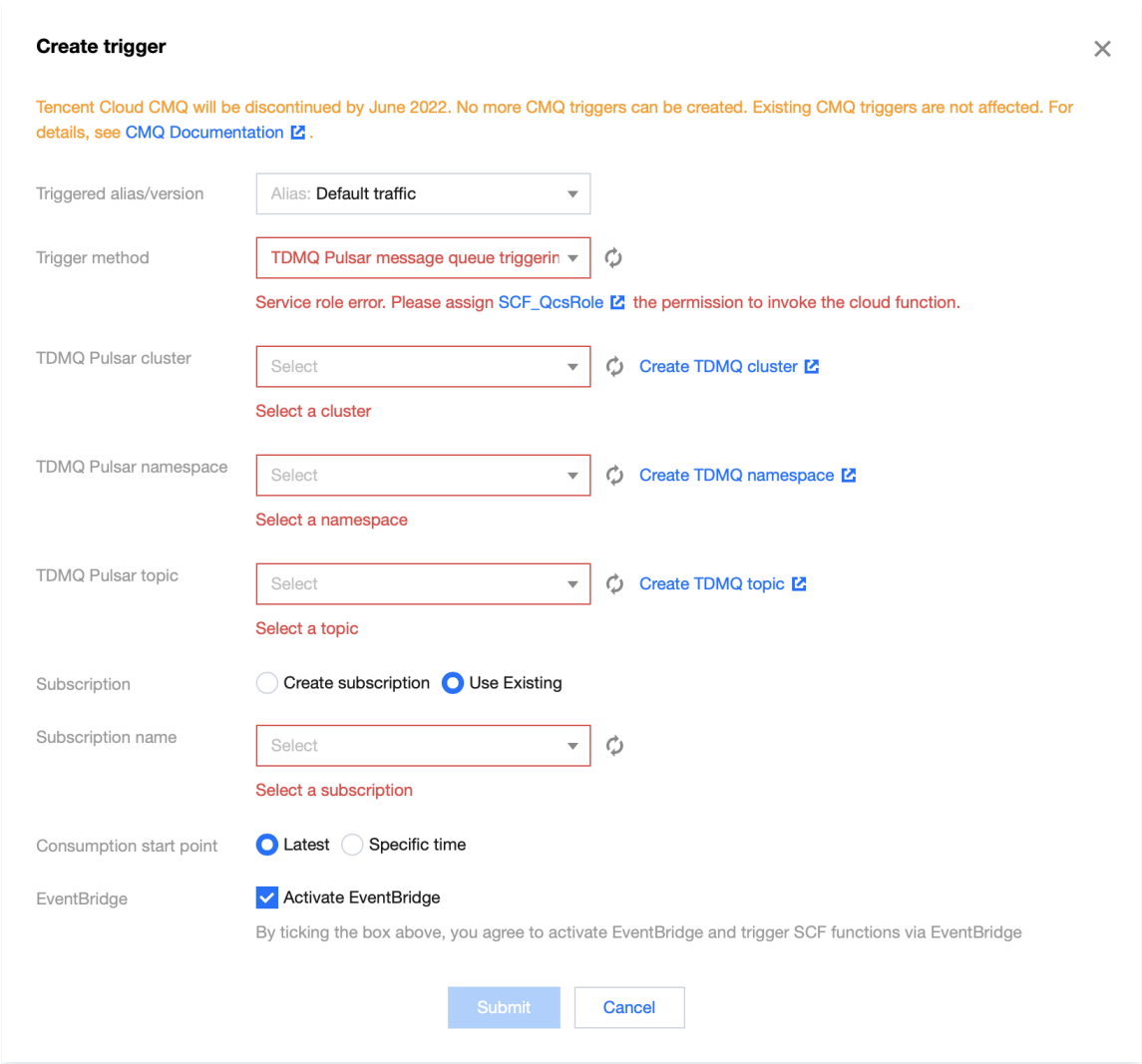
Log in to the Serverless Console, complete the upload and deployment of your function code on the new function page. For more details, refer to Creating an Event Function Using the Console.

> **Note**
> Currently, TDMQ is only available in the Beijing, Shanghai, and Guangzhou regions.

Step 2. Configure a trigger

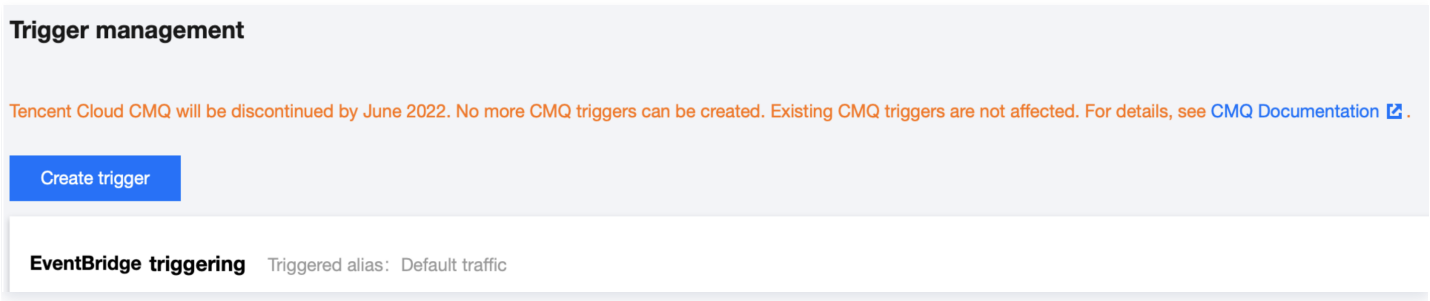
In the Configure Trigger step, select **TDMQ Pulsar message queue triggering**. Follow the guide to sequentially select your TDMQ cluster, topic, and other information to specify the trigger event source and

consumption location:

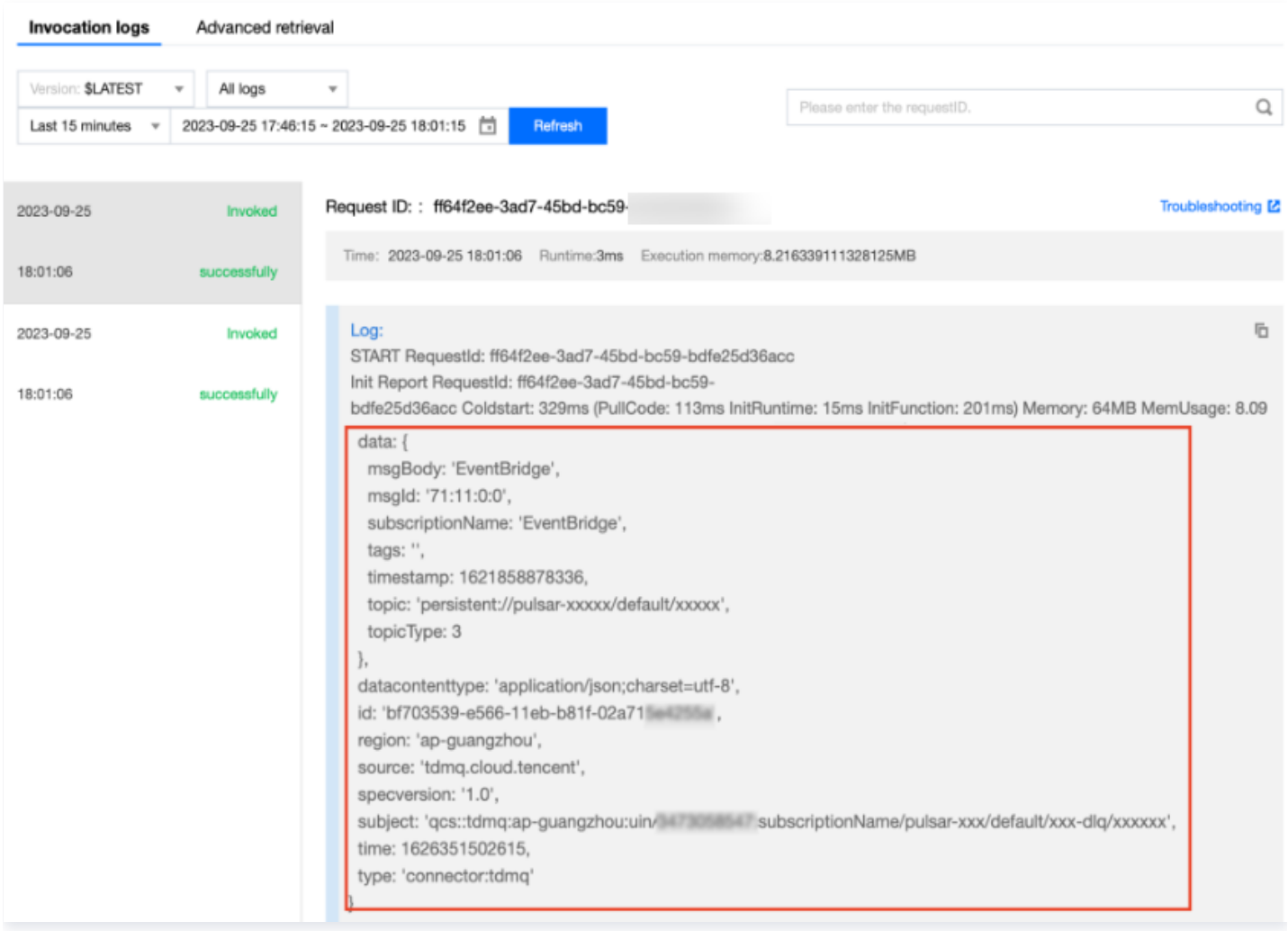


Step 3. Manage the trigger

Upon completion, you can view the created trigger information on the "Trigger management" page. Click to enter the EventBridge console to manage event sets, event sources, and other information. For more details, please refer to the EventBridge Documentation .



By sending information to the specified TDMQ message queue, you can see that the function is called normally:



Event Structure

```
{
  {
   "specversion": "0",
   "id": "13a3f42d-7258-4ada-da6d-023a333b4662",
   "type": "connector:tdmq",
   "source": "tdmq.cloud.tencent",
   "subjuect":
"qcs::tdmq:$region:$account:topicName/$topicSets.clusterId/$topicSets.environmentId/$topicSets.t
opicName/$topicSets.subscriptionName",
   "time": "1615430559146",
   "region": "ap-guangzhou",
   "datacontenttype": "application/json;charset=utf-8",
   "data": {
              "topic":  "persistent://appid/namespace/topic-1",
         "tags": "testtopic",
              "TopicType": 0,
              "subscriptionName": "xxxxxx",
```

```
            "toTimestamp": "1603352765001",
        "partitions": "0",
            "msgId": "123345346",
            "msgBody": "Hello from TDMQ!"
    }
}
```

The parameters are described as follows:

Category	Description
topic	Complete Topic Path `persistent://appid/namespace/topic-1` .
subscriptionName	Subscription Name.
timestamp	Timestamp, accurate to the millisecond.
tags	TDMQ Tag.
msgId	TDMQ Message ID.
msgBody	TDMQ Message Body.
topictype	Topic Type Description: 0: Standard message. 1: Globally sequential message. 2: Locally sequential message. 3: Retry Queue. 4: Dead Letter Queue.

Trigger Configuration Description

Last updated: 2023-09-27 19:49:09

When invoking the trigger interface CreateTrigger, the corresponding `TriggerDesc` parameter serves as the trigger description. Refer to this document for usage.

Timer Trigger

Please directly enter a cron expression for this parameter. For more information, please see Timer Trigger Description.

Sample TriggerDesc

Triggered once every five minutes:

```
0 */5 * * * * *
```

API Gateway Trigger

Name	Local Disk Types	Required	Description
api	ApigwApi	Not required	API configuration of the created API gateway
service	ApigwService	Not required	Service configuration of the created API gateway
release	ApigwRelease	Not required	Release environment for the created API gateway

ApigwApi

Name	Local Disk Types	Required	Description
authRequired	String	Not required	Whether authentication is required. Valid values: TRUE, FALSE. Default value: FALSE
requestConfig	ApigwApiRequestConfing	Not required	Configuration of request backend API
isIntegratedResponse	String	Not required	Whether to use integrated response. Valid values: TRUE, FALSE. Default value: FALSE
IsBase64Encoded	String	Not required	Whether to enable Base64-encoding. Valid values: TRUE, FALSE. Default value: FALSE

ApigwApiRequestConfing

Name	Local Disk Types	Required	Description
method	String	Not required	Method configuration of request backend API. Valid

			values: `ANY` , `GET` , `HEAD` , `POST` , `PUT` , `DELETE`

ApigwService

Name	Local Disk Types	Required	Description
serviceId	String	Not required	Apigw Service ID (if this parameter is not passed in, a new service will be created)

ApigwRelease

Name	Local Disk Types	Required	Description
environmentName	String	Supported	Release environment. Valid values: `release` , `test` , `prepub` . If this parameter is left empty, `release` will be used by default

Sample TriggerDesc

```
{
 "api":{
  "authRequired":"FALSE",
  "requestConfig":{
   "method":"ANY"
  },
  "isIntegratedResponse":"FALSE"
 },
 "service":{
  "serviceName":"SCF_API_SERVICE"
 },
 "release":{
  "environmentName":"release"
 }
}
```

CKafka Trigger

Name	Local Disk Types	Required	Description
maxMsgNum	String	Supported	A function invocation will be triggered once every time `maxMsgNum` CKafka messages are aggregated within 5 seconds
offset	String	Supported	Offset refers to the starting position for consuming CKafka messages. Currently, three methods are

			supported: `latest`, `earliest`, and `millisecond-level timestamp`
retry	String	Supported	Maximum number of retries when the function reports an error

Sample TriggerDesc

```
{"maxMsgNum":100,"offset":"latest","retry":10000}
```

```
{"maxMsgNum":999,"offset":"1595927203000","retry":10}
```

API request description

When creating a CKafka trigger via API request, the TriggerName field should be defined as the instanceId and topicName information of the CKafka to be dumped.
 `[instanceId]-[topicName]`. Here is an example of a request:

```
TriggerName: "ckafka-8tfxzia3-test"
```

COS Trigger

Name	Local Disk Types	Required	Description
event	String	Supported	COS event type
filter	CosFilter	Supported	COS filename filter

CosFilter

Name	Local Disk Types	Required	Description
Prefix	String	Not required	Prefix rule of file filter
Suffix	String	Not required	Suffix rule of file filter, which must begin with `.`

COS event conflict rules

- Core concept: an event can trigger a function invocation at most once. If an event is bound to another product, it cannot be bound to a function.
- Set at most one prefix filter and one suffix filter.
- If the `cos:ObjectCreated:*` event is set but no prefix/suffix is set, subsequent binding to any event that starts with `cos:ObjectCreated` will fail.
- The filter will be valid only if both the prefix and suffix are matched, and if there are conflicts with both the prefix and suffix, subsequent binding will fail.

Sample TriggerDesc

```
{"event":"cos:ObjectCreated:*","filter":{"Prefix":"","Suffix":""}}
```

> **Note**
> When `TriggerDesc` is used as a trigger description, the JSON string must be continuous with no spaces contained.

API request description

To create a COS trigger by using an API request, the `TriggerName` field needs to be defined as the XML API access domain name of the target COS bucket. Below is an example:

```
TriggerName: "xxx.cos.ap-guangzhou.myqcloud.com"
```

> **Note**
> Please view the access domain name in the Object Storage Console under **Bucket List > Basic Configuration > Basic Information**.

CMQ Trigger

Name	Local Disk Types	Required	Description
filterType	String	Not required	Message filtering type, where `1` represents tag type and `2` signifies route matching type.
filterKey	String	Not required	When `filterType` is `1`, it signifies message filtering tag, whereas when `filterType` is `2`, it represents Binding Key.

Sample TriggerDesc

```
{"filterType":1,"filterKey":["test"]}
```

```
{"filterType":2,"filterKey":["#test"]}
```

API request description

To create a CMQ trigger by using an API request, the `TriggerName` field needs to be defined as `CMQ Topic`. Below is an example:

```
TriggerName: "Tabortest"
```