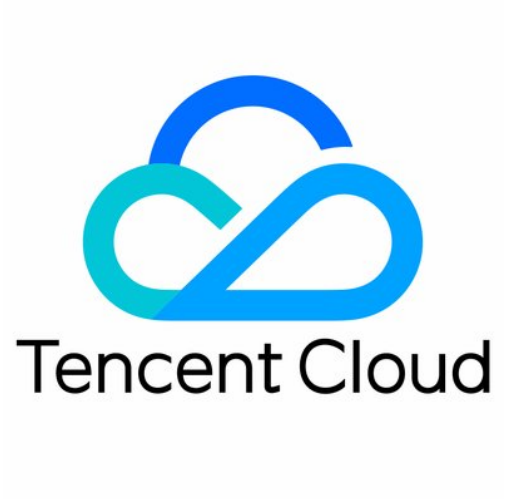
Serverless Cloud Function

Manage Cloud Function Trigger

Product Introduction



Contents

Manage Cloud Function Trigger
What is Function Trigger

Last updated：2018-07-03 10:14:43

Tencent Cloud SCF is a typical event-triggered server-free environment. The core components are SCF and event source. The event source is a Tencent Cloud service that publishes events or user-defined codes. SCF is an event processor. Function trigger manages the relationship between SCF and event source. For example, in the following scenario:

- Image/Video processing: When a user uploads images, he/she crops them to a proper size. When the user uses the application to upload images, it stores these images into COS, creates a thumbnail for each image, and displays these thumbnails on user page. In this scenario, you need to select COS as the event source to publish events to SCF when creating files. The event data provides all the information about buckets and files.
- Data processing: Analyze the data (such as clickstream) collected on the previous day at 0:00 am and generate a report. In this scenario, you need to select Timer as the event source to publish events to SCF at a specified time.
- Custom application: Call the first image processing SCF in one of your applications as a module of the application. In this scenario, you need to allow InvokeFunction to be called automatically in the application to publish events.

These event sources can be one of the following options:

- Internal event source: It is a pre-configured Tencent Cloud service that can be used in combination with SCF. When you configure these event sources to trigger function, the function is called automatically when an event is generated. Relation (i.e. event source mapping) between event source and function is maintained at the event source end. For example, COS provides the API Put Bucket Notification. You can bind Bucket events with the function by using this API.
- Custom application: You can allow the custom application to publish events and call SCF.

Example 1: COS Publishes Event and Calls Function

You can configure the event source mapping of COS to determine what operation (for example, PUT, DELETE an object, etc.) performed by COS can trigger SCF. The event source mapping of COS is stored in COS. You can use bucket notification feature to allow COS to call the function when a specific event is generated:

- Create a COS trigger
- Users create/delete an object in a bucket.
- COS detects an object creation/deletion event.
- COS calls the function automatically, and determines which function should be called based on the event source mapping stored in COS configuration. Pass Bucket and Object information as event data to the function.

Example 2: Timer Publishes Time and Calls Function

The event source mapping of timer is stored in SCF configuration to determine when to trigger the function:

- Create a timer
- The timer calls the function automatically when configuring time

Example 3: Custom Application Calls Function

If you need to call an SCF in a custom application, you do not need to configure a function trigger nor event source mapping. In this case, use Invoke API as the event source.

- The custom application uses InvokeFunction API to call the function, and automatically passes the event data.
- The function receives the trigger request and performs triggering.
- If synchronous call is used, the function returns the result to the application.

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

Notes

1. For limits on the number of triggers supported for a cloud function, please see Quota and Limits.
2. Due to restrictions of different cloud services, event source mapping relation is subject to specific limits. For example, for COS triggers, the same event (such as file upload) of the same COS Bucket cannot trigger multiple functions.

Trigger Types Supported for Tencent Cloud SCF

API Gateway Trigger

Last updated：2018-07-12 16:51:49

You can write an SCF to implement Web backend services and provide these services externally through the API gateway. The API gateway passes the content of the API request as parameters to the SCF, and sends the returned result from the function back to requester as the API response.

The API gateway trigger has the following features:

- **Push model**: When the API gateway receives an API request, the SCF will be triggered if it is configured on the backend of the API gateway. Meanwhile, the API gateway encapsulates the API request's information (such as the specific service receiving the request and API rules, the actual path of the request, the request method, the path, header and query of the request) as the request input parameters, and send it to the triggered function in the form of input parameter event.
- **Synchronous call**: The API gateway synchronously calls the function, and waits for the response of the function until the timeout set in the API gateway occurs. For more information about call types, please see Call Types.

Configuration of the API Gateway Trigger

The API gateway trigger is configured in the API gateway **instead of the SCF**. When you configure API rules in the API gateway, select Cloud Function for the backend configuration, and then you can select the SCF that shares the same region with the API service.

When the SCF is configured in the API gateway, the timeout is also configured. The request timeout in the API gateway and the SCF execution timeout take effect respectively, as described below:

- API gateway timeout > SCF timeout: SCF timeout takes effect first. A response of 200 HTTP code is returned with an error code indicating that the SCF has timed out.
- API gateway timeout < SCF timeout: API gateway timeout takes effect first. A response of 5xx HTTP code is returned, indicating that the request has timed out.

Binding Limits on API Gateway Trigger

In the API gateway, an API rule can only be bound to an SCF, but an SCF can be bound with multiple API rules as the backend. Besides, the API gateway trigger can only be bound to SCFs in the same region, i.e., those created in the Guangzhou region. These SCFs can only be bound to and triggered by the rules of the API services created in the Guangzhou region. To trigger an SCF with API gateway configuration in the specified region, you can create a function in this region.

Event Message Structure of API Gateway Trigger

When receiving a request, the API gateway sends the following event data in JSON format to the bound SCF.

```
{
  "requestContext": {
    "serviceName": "testsvc",
    "path": "/test/{path}",
    "httpMethod": "POST",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "identity": {
      "secretId": "abdcdxxxxxxxsdfs"
    },
    "sourceIp": "10.0.2.14",
    "stage": "prod"
  },
  "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": "test"
  },
```

```
  "path": "/test/value",
  "query": "foo=bar&bob=alice",
  "httpMethod": "POST"
}
```

The data structure is described as follows:

Name	Content
requestContext	The configuration information of the API gateway sending the request, request ID, verification information and source information. serviceName, path and httpMethod represent the API gateway service, the request path and method of the API, respectively. stage represents the environment where the request source API is located. requestId represents the unique ID of the current request. identity represents user's verification method and the information to be verified. sourceIp represents the request source IP.
path	Records the full path for the actual request
httpMethod	Records the HTTP method for the actual request
query	Records the full Query for the actual request
body	Records the full Body for the actual request
headers	Records the full Header for the actual request
pathParameters	Records the parameter path configured in the API gateway and its actual value
queryStringParameters	Records the parameter Query configured in the API gateway and its actual value
headerParameters	Records the parameter Header configured in the API gateway and its actual value

Note:
1. As the API gateway iterates, more parameters will be added to requestContext. Parameters **in** the data
2. Parameters **in** the actual request may appear **in** multiple locations **and** can be selected based on busir

How Does the API Gateway Trigger Process the Response of SCF

Since the SCF is called synchronously, the API gateway triggers the SCF until it is successfully executed, and then sends the returned result from SCF to the API request initiator as the API response. Therefore, the SCF can be used to implement API backend services. It processes the API request and returns the result which will then be sent to the API requester.

COS Trigger

Last updated：2018-07-03 10:15:15

You can write an SCF to process the object creation and deletion events in a COS Bucket. COS publishes events to the SCF and call the function using the event data as parameters. You can add the bucket notification configuration in the COS Bucket, which identifies the type of an event that can trigger a function, the name of a function to be called, and other information. For more information, please see the API PutBucketNotification.

COS trigger has the following features:

- **Push model**: COS monitors the specified Bucket action (event type) and calls a relevant function to push event data to the SCF. Use the Bucket notification in the push model to store the event source mapping of COS.
- **Asynchronous call**: COS always calls the function asynchronously, and does not return the result to the caller. For more information about call types, please see Call Types.

Attributes of COS Trigger

- (Required) COS Bucket: Configured COS Bucket. Only the COS bucket in the same region is supported.
- (Required) Event type: "File Upload" and "File Deletion" are supported to determine when the trigger triggers the function. For example, if "File Upload" is selected, the function is triggered when a file is uploaded to the COS Bucket.

Use Limits of COS Trigger

To prevent errors in COS event generation and delivery, COS sets a limit to bind only one function that can be triggered to each event (such as file upload/file deletion) in a bucket. Therefore, when you create a COS trigger, do not bind multiple function triggers to the same event of the same COS Bucket.

COS trigger supports triggering SCF with COS Bucket events in the same region, that is, when you configure a COS trigger for an SCF created in Guangzhou region, you can only choose a COS Bucket in the Guangzhou region (South China). To trigger an SCF with COS Bucket events in the specified region, you can create a function in this region.

Event Message Structure of COS Trigger

When an object creation or object deletion event is generated in the specified COS Bucket, COS sends the following event data in JSON format to the bound SCF.

```
{
  "Records":[
    {
     "event": {
      "eventVersion":"1.0",
      "eventSource":"qcs::cos",
      "eventName":"cos:ObjectCreated:*,
      "eventTime":"1970-01-01T00:00:00.000Z",
      "eventQueue":"qcs:0:cos:gz:1251111111:cos",
      "requestParameters":{
        "requestSourceIP": "111.111.111.111",
        "requestHeaders":{
          "Authorization": "Example"
        }
       }
      },
      "cos":{
        "cosSchemaVersion":"1.0",
        "cosNotificationId":"Configured or returned ID",
        "cosBucket":{
          "name":"bucketname",
          "appid":"1251111111",
          "region":"gz",
        },
        "cosObject":{
          "key":"/test.jpg",
          "size":"1024",
          "meta":{
            "Content-Type": "text/plain",
            "x-cos-meta-test": "Custom meta",
            "x-image-test": "Custom meta"
          },
          "url": "Origin server URL for accessing files"
        }
      }
    }
  ]
}
```

The data structure is described as follows:

Name	Content
Records	List structure. Multiple messages may be merged into the list
event	Records event information, including event version, event source, event name, time, queue information
cos	Records the COS information of the event
cosBucket	Records the bucket of the specific event, including bucket name, region, user APPID
cosObject	Records the object of the specific event, including object path, size, custom metadata, access URL
subscriptionName	Records the subscription name of SCF under the topic

Timed Trigger

Last updated：2018-07-31 11:25:08

You can write an SCF to process timed tasks. Timer can automatically trigger the SCF at a specified time. Timer trigger has the following features:

- **Pull model**: Timer directly calls the Invoke API of a function at a specified time to trigger the function. The event source mapping relation is stored in SCF.
- **Asynchronous call**: Timer always calls the function asynchronously, and does not return the result to the caller. For more information about call types, please see Call Types.

Attribute of Timer Trigger

- (Required) Timer name: Its length should be limited to 60 characters. `a-z`, `A-Z`, `0-9`, `-` and `_` are supported. It must begin with a letter. Multiple timer triggers with the same name are not supported for a function.
- (Required) Triggering cycle: A specified time at which the function is triggered. You can use the default value on the console, or choose a custom standard CRON expression to determine when to trigger the function. For more information on the CRON expression, please see below.

CRON Expression

When creating a timer trigger, you can use a standard CRON expression to customize the time to trigger function.

CRON Expression Syntax

CRON expression contains five required fields separated by spaces.

First	Second	Third	Fourth	Fifth
Minute	Hour	Day	Month	Week

Each field corresponds to a value range:

Field	Value	Wildcard

Field	Value	Wildcard
Minute	0-59	, - * /
Hour	0-23	, - * /
Day	1-31	, - * /
Month	1-12 or JAN-DEC	, - * /
Week	1-7 or MON-SUN	, - * /

The wildcard has the following meaning:

Wildcard	Meaning
, (comma)	A set of characters separated by commas. For example, in the "Hour" field, "1, 2, 3" refers to 1, 2 and 3 am/pm
- (dash)	Includes all values within a specified range. For example, in the "Day" field, "1-15" contains the 1st to 15th days of a specified month
* (asterisk)	All values. In the "Hour" field, * refers to each hour
/ (slash)	Specified increment. In the "Minute" field, entering 1/10 means to repeat every 10 minutes from the first minute. For example, the 11th minute, 21st minute, 31st minute, and so on

Note

- When "Day" and "Week" fields are specified at the same time in a CRON expression, the relation between them is "OR", and the function is triggered when either of them is met.

Example

The following examples show some CRON expressions and their meanings:

- `15 23 * * *` Run at 23:15 every night
- `0 18 * * MON-FRI` Run at 6:00 pm on each work day
- `0 8 1 * *` Run at 8:00 am on the 1st day of each month
- `0/15 * * * *` Run every 15 minutes

Input Parameters of Timer Trigger

No input parameter is specified when a timer trigger triggers a function, which means the event is empty. This allows you to determine whether the trigger source is a timer trigger.

CMQ Topic Trigger

Last updated：2018-07-03 10:16:11

You can write an SCF to process messages received by CMQ Topic. CMQ Topic can pass messages to the SCF and call the function using the message content and relevant information as parameters.

CMQ Topic trigger has the following features:

- **Push model**: After receiving messages, CMQ Topic pushes them to all subscribers who subscribe to the Topic. If an SCF is configured, it is also used as a subscriber to receive the messages pushed from the queue. In the push model, CMQ Topic stores the event source mapping of the SCF.
- **Asynchronous call**: CMQ Topic always calls the function asynchronously, and does not return the result to the caller. For more information about call types, please see Call Types.

Attributes of CMQ Topic Trigger

- (Required) CMQ Topic: Configured CMQ Topic. Only CMQ in the same region is supported.

Binding Limit on CMQ Topic Trigger

For CMQ Topic, a maximum of 100 subscribers are supported under a single topic. Therefore, if this limit is reached, binding of SCF triggers may fail. A topic can bind with multiple SCFs before this limit is reached.

CMQ Topic trigger supports triggering SCF with CMQ Topic messages in the same region, that is, when you configure a CMQ Topic trigger for an SCF created in Guangzhou region, you can only choose a CMQ Topic in the Guangzhou region (South China). To trigger an SCF with CMQ Topic messages in the specified region, you can create a function in this region.

Event Message Structure of CMQ Topic Trigger

When receiving a message, the specified CMQ Topic sends the following event data in JSON format to the bound SCF.

```
{
  "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"]
      }
    }
  ]
}
```

The data structure is described as follows:

Name	Content
Records	List structure. Multiple messages may be merged into the list
CMQ	Identifies the data structure source as CMQ
type	Determines whether the message source is topic or queue
topicOwner	Records the topic owner account ID
topicName	Records the topic name
subscriptionName	Records the subscription name of SCF under the topic
publishTime	Records the time when the message is published
msgId	Records the unique ID of the message
requestId	Records the ID of the request for message push
msgBody	Records the message content
msgTag	Records the message tag via the list structure

CKafka Trigger

Last updated：2018-07-03 10:17:42

You can write an SCF to process messages received by CKafka. The SCF backend module can be used as a consumer to consume the messages in CKafka and then send them to SCF.

CKafka trigger has the following features:

- **Pull model**: The backend module of SCF, as a consumer, connects to the CKafka instance and consumes messages. After receiving messages, the backend module encapsulates them into the data structure and calls the specified function to pass the data structure to the SCF.
- **Asynchronous call**: CKafka trigger always calls the function asynchronously, and does not return the result to the caller. For more information about call types, please see Call Types.

Attributes of CKafka Trigger

- CKafka instance: CKafka instance configured for connection. Only instance in the same region is supported.
- Topic: Supports topics already created in the CKafka instance.
- Maximum number of messages: The maximum number of messages pulled and sent in batches to the current SCF. The number of messages that are sent every time the SCF is triggered may not reach the limit depending on the size of massages and write speed. So it is a variable ranging from 1 to the maximum number.

How to Consume and Send CKafka Messages

Since CKafka messages cannot be pushed automatically, but must be pulled by consumers for consumption. Therefore, after a CKafka trigger is configured, the SCF backend launches a CKafka consumer module as the consumer to consume messages in an independent consumer group it created in CKafka.

After consuming messages, the SCF backend consumer module encapsulates them into an event structure according to timeout, number and size of accumulated messages and maximum number of messages, and then sends it to the SCF. During the consumption, 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 configured is too large, the number of messages in the event structure will never reach the limit.

After obtaining the event content, the SCF can process messages on a loop to make sure every message is processed, rather than assuming that the number of message passed each time is constant.

Event Message Structure of CKafka Trigger

When the specified CKafka Topic receive a message, the SCF backend consumer module consumes the message and encapsulates it into the following event structure in JSON format, then triggers the bound function and passes it the data content as input parameters.

```
{
  "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 structure is described as follows:

Name	Content
Records	List structure. Multiple messages may be merged into the list
Ckafka	Identifies the event source as CKafka
topic	Message source topic
partition	Partition ID of message source

Name	Content
offset	Consumption offset number
msgKey	Message key
msgBody	Message content