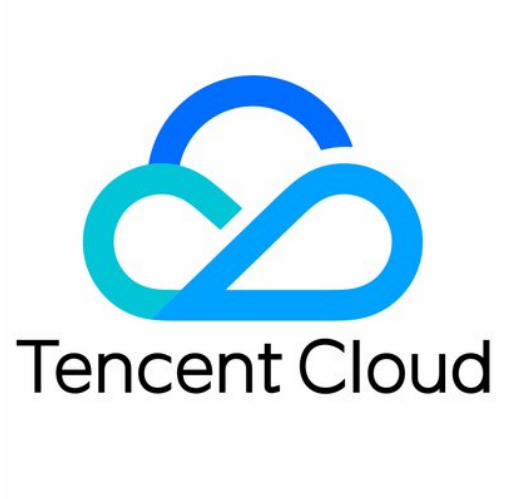
TencentDB for CTSDB

HTTP API References

Product Documentation



Contents

HTTP API References
CTSDB 2.0
Overview

Last updated：2023-03-02 14:39:53

CTSDB 2.0 (still referred to as CTSDB) supports data write, query, and other operations over the HTTP protocol. Its HTTP APIs are RESTful, and the resource request method is to send standard HTTP requests to the corresponding URIs of resources. For example, `GET` is used to get resources, `POST` create (or update) resources, `PUT` update resources, and `DELETE` delete resources.
You can perform almost all data operations through HTTP APIs. CTSDB ensures data security by providing VPC-based network isolation and access authentication with username and password. Data is exchanged through structures in JSON format. Each request will return a standard HTTP response status code and content. If an operation fails, you can get specific error information based on the response content.
For more information on how to connect to instances through RESTful API, see Connecting to Instance.

Naming conventions

The names of metrics, tags, and fields in CTSDB should be concise and clear. For more information on tags and fields, see Creating Metric:
Naming conventions for metrics: A name can contain 1–200 lowercase letters, digits, underscores, hyphens, and dots but cannot start with a dot, underscore, or hyphen.
Naming conventions for tags and fields in metrics: A name can contain 1–255 letters, digits, underscores, hyphens, and dots but cannot start with a dot.

System limits

Limits of tags and fields in metrics: The total number of fields in a metric cannot exceed 1,000.
Limits of write points during bulk writing to metric: We recommend that you limit the number of records in each `bulk` request between 1,000 and 5,000 and the physical size between 1 and 15 MB.

System default rules

Processing of new fields during data write

If there are undefined new fields when you write data to a metric, CTSDB will store the new fields as tags by default. You can also change this by modifying the `default_type` field in the `options` parameter of the metric. For more information on metric modification, see Updating Metric.

If a new field is an integer, CTSDB will store it in `long` type.

If a new field is a decimal number, CTSDB will store it in `float` type.

If a new field is a string, CTSDB will store it in `string` type subject to the value of `max_string_length` and discard the excessive part. `max_string_length` can be customized and is 256 characters by default. For more information on modification, see Updating Metric.

Processing of date and time: Date and time are stored in UTC format in CTSDB. Therefore, for data queries involving time range, specify the time zone with the `time_zone` parameter, which is in the format of ISO 8601 UTC offset (e.g., +01:00 or -08:00). The specific time zone needs to be determined according to the region where the instance resides. Generally, the time zone for the Chinese mainland is UTC+8. For more information, see Common Query Samples.

Instance connection

CTSDB instances currently can be connected to only in VPCs. You can connect to an instance in the console or through RESTful APIs. In the latter case, you need to provide the root account password to ensure security.

Below is a code sample for connecting to an instance and creating a metric through curl. Here, `${user:password}` is the username and password of the instance, `${vip}:${vport}` the IP and port, and `${metric_name}` the name of the created metric.

```
curl -u ${user:password} -H 'Content-Type:application/json' -X PUT
${vip}:${vport}/_metric/${metric_name} -d'
{
  "tags": {
      "region": "string",
      "set":  "long",
      "host": "string"
  },
  "time": {
       "name": "timestamp",
       "format": "epoch_second"
  },
  "fields": {
      "cpu_usage":  "float"
  },
  "options": {
      "expire_day": 7,
      "refresh_interval": "10s",
```

```
        "number_of_shards": 5,
        "number_of_replicas": 1,
        "rolling_period": 1
    }
}'
```

Creating Metric

Last updated：2022-06-09 15:16:07

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_metric/${metric_name}`, where `${metric_name}` is the name of the metric to be created.

Method: PUT

Note:

For the naming limits of metrics, see System limits.

Request parameters

None

Request content

Parameter	Required	Type	Description
tags	Yes	map	Tag, which is used to uniquely identify data. It must contain at least one tag ar (string without tokens), `long`, `integer`, `short`, `byte`, `double` `"long","host": "string"}`
time	Yes	map	Configuration of time column, which is used to store the unique time when dat `"epoch_second"}`. It should be entered completely, where `name` and
fields	No	map	Fields for data storage. We recommend you use data types most suitable for y `long`, `integer`, `short`, `byte`, `double`, `float`, `date`,
options	No	map	Common fine-tuning configuration information, for example, `{"expire_day":7,"refresh_interval":"10s","number_of_sha 256,"default_date_format":"strict_date_optional_time","i`

The `name` of the `time` field is of the `timestamp` type by default. The time formats (`format`) are fully compatible with those in Elasticsearch, such as `epoch_millis` (Unix timestamp in milliseconds), `epoch_second` (Unix timestamp in seconds), `basic_date` (in `yyyyMMdd` format), and `basic_date_time` (in `yyyyMMdd'T'HHmmss.SSSZ` format).

`options` values and their descriptions:

expire_day: Data expiration time in days, which is a non-zero integer. Once expired, the data will be automatically cleared. By default, it is the minimum value -1, which indicates that the data will never expire.

refresh_interval: Data refresh interval, which is 10 seconds by default. The written data can be queried after being refreshed from the memory to disk.

number_of_shards: Number of metric shards, which is a positive integer and 3 by default. This parameter can be ignored for small metrics. A large metric can be divided into shards, and each shard can be up to 25 GB in size.

number_of_replicas: Number of replicas, which is a positive integer; for example, 1 indicates one master and one replica. The default value is 1.

rolling_period: Child metric period (in days), which is a non-zero integer. When CTSDB stores data, to facilitate data expiration and improve query efficiency, it stores data in **child metrics** by the specified time interval, which is subject to the data expiration time by default. The relationships between the default child metric period and data expiration time are as detailed below:

max_string_length: Maximum length of a custom string, which is a positive integer. Its maximum value is 32765, and its default value is 256.

default_date_format: Format of the `date` data type of custom tags and fields, which is `strict_date_optional_time` or `epoch_millis` by default.

indexed_fields: Fields whose indexes need to be retained in the fields. You can use an array to specify multiple fields.

default_type: Default type of new fields. Its valid values are `tag` and `field` , and its default value is `tag` .

Expiration Time	Child Metric Period
≤ 7 days	1 day
> 7 days but ≤ 20 days	3 days
> 20 days but ≤ 49 days	7 days
> 49 days but ≤ 3 months	15 days
> 3 months	30 days
Never expires	30 days

Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.xx.xx.4:9201/_metric/ctsdb_test -d'
{
    "tags":
    {
        "region":"string"
    },
    "time":
    {
        "name":"timestamp",
        "format":"epoch_second"
    },
    "fields":
    {
        "cpuUsage":"float"
    },
    "options":
    {
        "expire_day":7,
        "refresh_interval":"10s",
        "number_of_shards":5
    }
   }
'
```

Response upon success:

```
{
    "acknowledged": true,
    "message": "create ctsdb metric ctsdb_test success!"
   }
```

Response upon failure:

```
  {
    "error": {
    "reason": "table ctsdb_test already exist",
    "type": "metric_exception"
```

```
    },
    "status": 201
}
```

Querying Metric

Last updated：2023-03-02 14:39:32

Getting All Metrics

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_metrics`

Method: GET

Request parameters

None

Request content

None

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_metrics
```

Response:

```
{
  "result":
{
   "metrics":
[
    "ctsdb_test",
    "ctsdb_test1"
   ]
  },
  "status": 200
```

```
}
```

Getting the Specified Metric

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_metric/${metric_name}`, where `${metric_name}` is the metric name.

Method: GET

Request parameters

None

Request content

None

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_metric/ctsdb_test
```

Response:

```
{
   "result":
{
    "ctsdb_test":
    {
        "tags":
        {
            "region": "string"
        },
        "time":
        {
            "name": "timestamp",
```

```
            "format": "epoch_second"
        },
        "fields":
        {
            "cpuUsage": "float"
        },
        "options":
        {
            "expire_day": 7,
            "refresh_interval": "10s",
            "number_of_shards": 5
        }
     }
    },
    "status": 200
}
```

Updating Metric

Last updated：2023-03-02 14:34:58

Request address and method

The address is the instance IP and port, such as `10.13.20.15:9200` , which can be obtained in the console.

Request parameters

None

Request content

The `tags` , `time` , `fields` , and `options` fields are all of the `map` type and optional. For their formats, see Creating Metric. The specific requirements are as detailed below:
tags: You can add tag fields and modify the types of existing ones without deleting them.
time: You cannot modify `name` but can modify `format` .
fields: You can add metric fields and modify the types of existing ones without deleting them.
`options` attributes are as detailed below:

Attribute name	Required	Type	Description
expire_day	No	integer	Data expiration time, which is a non-zero integer. Once expired, the data will be automatically cleared. By default, data never expires.
refresh_interval	No	string	Data refresh interval, which is 10 seconds by default. The written data can be queried after being refreshed from the memory to disk.
number_of_shards	No	integer	Number of metric shards, which is a positive integer and `3` by default. This parameter can be ignored for small metrics. A large metric can be divided into shards, and each shard can be up to 25 GB in size.
number_of_replicas	No	integer	Number of replicas, which is a non-negative integer; for example, 1 indicates one master and one replica. The default value is `1` .

rolling_period	No	integer	Child metric period (in days), which is a non-zero integer. When CTSDB stores data, to facilitate data expiration and improve query efficiency, it stores data into child metrics by the specified time interval, which is subject to the data expiration time by default.
max_string_length	No	integer	Maximum length of a custom string, which is a positive integer. Its maximum value is `32765`, and its default value is `256`.
default_date_format	No	string	Format of the `date` data type of custom tags and fields, which is `strict_date_optional_time` or `epoch_millis` by default.
indexed_fields	No	array	Fields whose indexes need to be retained in the fields. You can use an array to specify multiple fields.
default_type	No	string	Default type of new fields. Its valid values are `tag` and `field`, and its default value is `tag`.

Notes

As the historical data cannot be modified, after fields are updated, the metric information will not immediately change until the next child metric as described in Creating Metric is generated. If you want to confirm whether the update is successful, you can call the `GET /_metric/${metric_name}?v` API.

Only fields of the `short`, `integer`, `float` data types can be changed to another type. The `short` type can be changed to the `integer` or `long` type, `integer` to `long`, and `float` to `double`.

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.xx.xx.4:9201/_metric/ctsdb_test/update -d'
{
    "tags":{
        "set":"string"
```

```
    },
    "time":{
        "name":"timestamp",
        "format":"epoch_second"
    },
    "fields":{
        "diskUsage":"float"
    },
    "options":{
        "expire_day":15,
        "number_of_shards":10
    }
}'
```

Response:

```
{
   "acknowledged": true,
   "message": "update ctsdb metric test111 success!"
}
```

Deleting Metric Field

Last updated：2023-03-02 14:35:20

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_metric/${metric_name}/delete`, where `${metric_name}` is the metric name.
Method: PUT

Request parameters

None

Request content

Parameter	Required	Type	Description
tags	No	Array	Enumeration of the tag fields to be deleted, such as `"tags": ["ip"]`.
fields	No	Array	Enumeration of the data fields to be deleted, such as `"fields": ["diskUsage"]`.

Notes
As historical data cannot be modified, after fields are deleted, the metric information will not change until the next child metric is generated. If you want to check whether the deletion is successful, you can call the `GET /_metric/${metric_name}?v` API. For more information about child metric, see Creating Metric.

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.xx.xx.4:9201/_metric/ctsdb_test1/delete -d'
{
"tags": ["ip"],
"fields": ["cpu"]
}'
```

Response:

```
{
   "acknowledged": true,
   "message": "update ctsdb_test1 metric success!"
}
```

Deleting Metric

Last updated：2023-03-02 14:35:38

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_metric/${metric_name}`, where `${metric_name}` is the name of the metric to be deleted.
Method: DELETE

Request parameters

None

Request content

None

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

Request:

```
DELETE /_metric/ctsdb_test1
```

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X DELETE
172.xx.xx.4:9201/_metric/ctsdb_test1
```

Response:

```
{
   "acknowledged": true,
   "message": "delete metric ctsdb_test1 success!"
}
```

Querying Data

Last updated：2023-03-02 14:36:26

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `${metric_name}/_search`, where `${metric_name}` is the metric name.
Method: GET

Request parameters

You can set the `_routing` parameter during data write to improve the query efficiency. For more information, see the samples.

Request content

Queries mainly include regular queries and aggregate queries. Query requests are fully compatible with Elasticsearch APIs as described in Getting started with Elasticsearch. For the specific request content, see the samples.

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

For specific query samples, see Common Query Samples.

Bulk Querying Data

Last updated：2023-03-02 14:36:47

Request Address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request Path and Method

Path: `/_msearch`
Method: GET

Request Parameters

You can set the `_routing` parameter during data write to improve the query efficiency as instructed in the sample. You can also set the `filter_path` parameter to filter and simplify the returned results. Its value contains multiple JSON paths separated by comma, and each JSON path is constructed by concatenating elements with dots. You can also use the `*` wildcard to match any character, the `**` wildcard to match any path, and the `-` prefix to remove certain returned fields.
For example, `responses._shard.f*` indicates fields starting with "f" in `_shards` in `responses`, and `responses.**._score` indicates all `_score` fields in `responses`.

Request Content

Queries mainly include regular queries and aggregate queries. Query requests are fully compatible with Elasticsearch APIs. For the specific request content, see the samples.

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description. Some common returned fields for queries are as detailed below. You can set the `filter_path` parameter to simplify the returned query results.

Field name	Description
hits	Matched query results. Here, the `total` field indicates the number of matched data records. The `hits` field is an array, which contains the first ten query results if not specified. Each result in the `hits` array contains `_index` (child metric as described in Creating Metric involved in the query). If `docvalue_fields` is specified in the query, the `fields` field will be returned to indicate the value of each field.
took	Time in milliseconds taken by the entire query.
_shards	Number of shards involved in the query. Here, `total` indicates the total number of shards, `successful` the shards that were successfully queried, `failed` the shards that failed to be queried, and `skipped` skipped shards.
timed_out	Indicates whether query timed out. Valid values: `false` , `true` .
status	Status code returned for the query. `2XX` indicates success.

Sample code for curl

Request without the `filter_path` parameter

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/_msearch -d'
{"index" : "amyli_weather","routing":"host_20"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
{"index" : "ctsdb_test2017","routing":"host_100"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
'
```

Response:

```
{
     "responses":
    [
        {
          "took": 0,
          "timed_out": false,
          "_shards": {
            "total": 0,
            "successful": 0,
            "skipped": 0,
            "failed": 0
```

```
          },
          "hits": {
            "total": 0,
            "max_score": 0,
            "hits": []
          },
          "status": 200
        },
        {
          "took": 1,
          "timed_out": false,
          "_shards": {
            "total": 3,
            "successful": 3,
            "skipped": 0,
            "failed": 0
          },
          "hits": {
            "total": 1,
            "max_score": 1,
            "hits": [
              {
                "_index": "ctsdb_test2017@-979200000_30",
                "_type": "doc",
                "_id": "AV_8fBhlUAkC9PF9L-2t",
                "_score": 1
              }
            ]
          },
          "status": 200
        }
      ]
}
```

Request with the `filter_path` parameter

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/_msearch?
filter_path=responses.\\*\\*.hits,responses._shards.f\\*,-
responses.\\*\\*._score -d'
{"index" : "amyli_weather","routing":"host_20"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
{"index" : "ctsdb_test2017","routing":"host_100"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
'
```

Response:

```
{
      "responses": [
    {
      "_shards": {
        "failed": 0
      }
    },
    {
      "_shards": {
        "failed": 0
      },
      "hits": {
        "hits": [
          {
            "_index": "ctsdb_test2017@-979200000_30",
            "_type": "doc",
            "_id": "AV_8fBhlUAkC9PF9L-2t"
          }
        ]
      }
    }
      ]
}
```

Notes

If both match and filter conditions are set in the `filter_path` parameter, the filter condition will be applied first for the returned results and then the match condition.

Common Query Samples

Last updated：2023-03-02 14:37:13

This document describes how to use common queries and keywords in CTSDB with simple samples for curl. The metric structure used by all samples is as follows:

```
{
    "ctsdb_test" : {
      "tags" : {
        "region" : "string"
      },
      "time" : {
        "name" : "timestamp",
        "format" : "epoch_millis"
      },
      "fields" : {
        "cpuUsage" : "float",
        "diskUsage" : "string",
        "dcpuUsage" : "integer"
      },
      "options" : {
        "expire_day" : 7,
        "refresh_interval" : "10s",
        "number_of_shards" : 5,
        "indexed_fields" : "cpuUsage"
      }
    }
}
```

Combined Query

Combined queries can be used for single queries as well as composite queries. The `query` keyword in the query body can use query domain-specific language (DSL) to define query conditions.
This document describes how to construct and combine filter conditions and process the returned result set.

Common filter conditions

1. Range

`Range` indicates range query, which supports fields of `string`, `long`, `integer`, `short`, `double`, `float`, and `date` types. The parameters that can be contained in a range query are as detailed below:

Parameter	Description
gte	Greater than or equal to
gt	Greater than
lte	Less than or equal to
lt	Less than

Notes

If a range query involves time types, you can use the `format` parameter to specify the time format. For the specific time formats, see Creating Metric.

Sample code of time range query for curl:

```
   curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
       "query": {
           "range": {
               "timestamp": {
                   "gte": "01/01/2018",
                   "lte": "03/01/2018",
                   "format": "MM/dd/yyyy",
                   "time_zone":"+08:00"
                }
            }
         }
    }'
```

Sample code of numeric range query for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "query": {
        "range": {
            "cpuUsage": {
                "gte": 1.0,
                "lte": 10.0
            }
        }
    }
}'
```

2. Terms

The `terms` keyword can be used to match specific fields during query, and its value needs to be enclosed by brackets ([]).
Sample code for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
       "query": {
           "terms": {
               "region": ["sh", "bj"]
           }
       }
   }'
```

Filter condition combination

A composite query usually uses bool keywords to combine multiple query conditions. Common combination keywords used for bool queries include `filter` (similar to `AND`), `must_not` (similar to `NOT`), and `should` (similar to `OR`). To improve the query efficiency, you must add a range query for the `time` field, which always returns a value in `epoch_millis` format no matter how the query is written. The combination of `query-bool-filter` can greatly improve the query performance, so be sure to use it.

1. Sample code of `AND` condition for curl

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
     "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "timestamp": {
              "format": "yyyy-MM-dd HH:mm:ss",
              "gte": "2017-11-06 23:00:00",
              "lt": "2018-03-06 23:05:00",
              "time_zone":"+08:00"
            }

          }
        },
        {
          "terms": {
```

```
            "region": ["sh"]
          }
        },
        {
          "terms": {
            "cpuUsage": ["2.0"]
          }
        }
      ]
    }
  },
      "docvalue_fields": [
    "cpuUsage",
    "timestamp"
       ]
}'
```

Notes

The query conditions are similar to `timestamp>='2017-11-06 23:00:00' AND timestamp<'2018-03-06 23:05:00' AND region=sh AND cpuUsage=2.0` .

2. Sample code of `OR` condition for curl

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
     "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "timestamp": {
              "format": "yyyy-MM-dd HH:mm:ss",
              "gte": "2017-11-06 23:00:00",
              "lt": "2018-11-06 23:05:00",
              "time_zone":"+08:00"
            }
          }
        },
        {
          "term": {
            "region": "gz"
          }
        }
      ],
      "should": [
```

```
          {
            "terms": {
              "cpuUsage": ["2.0"]
            }
          },
          {
            "terms": {
              "cpuUsage": ["2.5"]
            }
          }
        ],
        "minimum_should_match": 1
      }
       },
       "docvalue_fields": [
      "cpuUsage",
      "timestamp"
       ]
  }'
```

Notes

The query conditions are similar to `timestamp>='2017-11-06 23:00:00' AND timestamp<'2018-11-06 23:05:00' AND region='gz' AND (cpuUsage=2.0 or cpuUsage=2.5)`. The `minimum_should_match` parameter is used to set the minimum number of results that should match `cpuUsage=2.0` and `cpuUsage=2.5`, and its default value is 0.

3. Sample code of `NOT` condition for curl

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
{
     "query": {
    "bool": {
      "filter": [
                {
          "range": {
            "timestamp": {
              "format": "yyyy-MM-dd HH:mm:ss",
              "gte": "2017-11-06 23:00:00",
              "lt": "2018-11-06 23:05:00",
              "time_zone":"+08:00"
            }

          }
        },
        {
```

```
            "terms": {
              "region": ["gz"]
            }
          }
        ],
        "must_not": [
         {
            "terms": {
              "cpuUsage": ["2.0"]
            }
          }
        ]
      }
       },
       "docvalue_fields": [
      "cpuUsage",
      "timestamp"
          ]
       }'
```

Notes

The query conditions are similar to `timestamp>=2017-11-06 23:00:00 AND timestamp<2018-11-06 23:05:00 AND region='gz' AND cpuUsage !=2.0` .

Processing of the returned result set

1. From/Size

You can paginate query results by setting the `from` and `size` keywords. The `from` keyword defines the offset of the first data entry in the query results, and `size` the maximum number of returned results. The default values of `from` and `size` are 0 and 10, respectively. The sum of `from` and `size` cannot exceed 65,536 by default. If you want to have more results returned, see the description of query with the `scroll` keyword in this document.

Sample code for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "from": 0,
    "size": 5,
    "query": {
        "bool": {
            "filter": {
                "range": {
                    "timestamp": {
                        "gte": "01/01/2018",
```

```
                        "lte": "03/01/2018",
                        "format": "dd/MM/yyyy",
                        "time_zone":"+08:00"
                    }
                }
            },
            "must_not": {
                "terms": {
                    "region": ["bj"]
                }
            }
        }
    }
}'
```

2. Scroll

A scroll operation is similar to a cursor in a relational database. Therefore, it is suitable for backend batch processing rather than real-time search.

You can divide a scroll operation into two phases: initialization and traversal. During initialization, all search results matching the search conditions will be cached like with a snapshot, from which data will be taken out during traversal. Note that insertion, deletion, and update of metrics after scroll initialization will not affect the traversal result.

During initialization, you can use the `size` keyword to specify the size of the returned result set, whose default value is 10 and maximum value is 65,536. You can also use the `query` and `docvalue_fields` keywords to specify the query conditions and returned fields respectively. The `_scroll_id` field will be returned during both initialization and traversal. You can specify the `_scroll_id` returned by the previous traversal for a new traversal until the returned result is empty. During initialization and traversal, you can also use the `scroll` parameter to set the context retention time of the traversal, after which the `scroll_id` will become invalid. The time format is as detailed below:

Format	Description
d	days
h	hours
m	minutes
s	seconds
ms	milliseconds
micros	microseconds
nanos	nanoseconds

Sample code for curl:

Scroll initialization:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/ctsdb_test/_search?scroll=1m -d'
   {
      "size":5,
      "query": {
      "bool": {
        "filter": [
          {
            "terms": {
              "region": ["gz"]
            }
          }
        ]
      }
    },
   "docvalue_fields": [
         "cpuUsage",
       "region",
       "timestamp"
       ]
}'
```

Response of scroll initialization:

```
   {
     "_scroll_id":
"DnF1ZXJ5VGhlbkZldGNoAwAAAAAADrOFFm5YSEhnMjdnUWNPcndHS1k5Wjc3bHcAAAAAAAz_1RZiRk
ZTcGp4dFRXR18xMGtzSmhEUFJRAAAAAAAP5vQWOXFOR29lc0hROHFWMmFGTkVmSkxmZw==",
     "took": 10641,
     "timed_out": false,
     "_shards": {
      "total": 3,
      "successful": 3,
      "skipped": 0,
      "failed": 0
     },
     "hits": {
       "total": {
            "value":1592072666,
            "relation":"eq"
        },
       "max_score": 0.65708643,
       "hits": [
         {
```

```
      "_index": "ctsdb_test@0_-1",
      "_type": "doc",
      "_id": "oyylNU0U65cZjByyt7sW_JmPPgAACY4Bh0",
      "_score": 0.65708643,
      "_routing": "354d14eb",
      "fields": {
        "region": [
          "gz"
        ],
        "cpuUsage": [
          "2.0"
        ],
        "timestamp": [
          1509909300000
        ]
      }
    },
    {
      "_index": "ctsdb_test@0_-1",
      "_type": "doc",
      "_id": "oyykFN0yd1d9NDPfzjRdrJ8whQAACYsBqc",
      "_score": 0.65708643,
      "_routing": "14dd3277",
      "fields": {
        "region": [
          "gz"
        ],
        "cpuUsage": [
          "1.8"
        ],
        "timestamp": [
          1509908340000
        ]
      }
    },
    {
      "_index": "ctsdb_test@0_-1",
      "_type": "doc",
      "_id": "oyylHLp9jl_3sF4N2rnh67h4SgAABHIBso",
      "_score": 0.65708643,
      "_routing": "1cba7d8e",
      "fields": {
        "region": [
          "gz"
        ],
        "cpuUsage": [
          "2.5"
```



```
            ],
            "timestamp": [
              1509909720000
            ]
          }
        },
        {
          "_index": "ctsdb_test@0_-1",
          "_type": "doc",
          "_id": "oyylH2JsOKnFHGUinQ7jM-ZwkgAAAvcBso",
          "_score": 0.65708643,
          "_routing": "1f626c38",
          "fields": {
            "region": [
              "gz"
            ],
            "cpuUsage": [
              "2.1"
            ],
            "timestamp": [
              1509909720000
            ]
          }
        },
        {
          "_index": "ctsdb_test@0_-1",
          "_type": "doc",
          "_id": "oyylHLp9jl_3sF4N2rnh67h4SgAABGsBso",
          "_score": 0.65708643,
          "_routing": "1cba7d8e",
          "fields": {
            "region": [
              "gz"
            ],
            "cpuUsage": [
              "2.0"
            ],
            "timestamp": [
              1509909720000
            ]
          }
        }
      ]
        }
}
```


Scroll traversal:


```
   curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/_search/scroll  -d'
   {
   "scroll" : "1m",
   "scroll_id" :
"DnF1ZXJ5VGhlbkZldGNoAwAAAAADrOFFm5YSEhnMjdnUWNPcndHS1k5Wjc3bHcAAAAAAAz_1RZiRk
ZTcGp4dFRXR18xMGtzSmhEUFJRAAAAAAAP5vQWOXFOR29lc0hROHFWMmFGTkVmSkxmZw=="
   }'
```

Notes

`scroll_id` in this request is the value of `_scroll_id` returned in scroll initialization. In the next traversal, the `scroll_id` parameter value should be adjusted to the `_scroll_id` value returned in the previous traversal, that is, the `scroll_id` parameter value in each request is the `_scroll_id` value returned in the previous request, and traversal will end until the returned result is empty.

The `_scroll_id` values returned by the two traversals may be the same, so `_scroll_id` cannot be used to redirect to the specified page.

3. Sort

The `sort` keyword is mainly used to sort query results and has two orders: `asc` and `desc`. The default sorting order for custom fields in CTSDB is `asc`. Available sorting modes include `min`, `max`, `sum`, `avg`, and `median`, where `sum`, `avg`, and `median` are suitable only for fields of `array` type that store numbers only.

Sample code for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
       "query": {
           "bool": {
               "must": {
                   "range": {
                       "timestamp": {
                           "gte": "01/01/2018",
                           "lte": "03/01/2018",
                           "format": "MM/dd/yyyy",
                           "time_zone":"+08:00"
                       }
                   }
               }
           }
       },
       "sort": [
           {
```

```
                "cpuUsage": {
                    "order": "asc",
                    "mode": "min"
                }
            },
            {
                "timestamp": {
                    "order": "asc"
                }
            },
            "diskUsage"
        ]
    }'
```

4. docvalue_fields

The `docvalue_fields` keyword specifies the names of the fields to be returned in an array.

Sample code for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
       "query": {
           "terms": {
               "region": ["sh", "bj"]
           }
       },
       "docvalue_fields": ["timestamp", "cpuUsage"]
   }'
```

Aggregate Query

The `agg` keyword is mainly used to construct aggregate queries. You can get the aggregate results in the returned `aggregations` field. The returned aggregate fields are as detailed below. If you want to focus only on the aggregate results, set the `size` parameter to 0 during query.

Field name	Description
hits	Matched query results. Here, the `total` field indicates the number of data records participating in aggregate. The `hits` field is an array, which contains the first 10 query results if not specified. Each result in the `hits` array contains `_index` (child metric as described in Creating Metric involved in the query). If `docvalue_fields` is specified in the query, the `fields` field will be returned to indicate the value of each field.

took	Time in milliseconds taken by the entire query.
_shards	Number of shards involved in the query. Here, `total` indicates the total number of shards, `successful` the shards that were successfully queried, `failed` the shards that failed to be queried, and `skipped` skipped shards.
timed_out	Indicates whether query timed out. Valid values: `false` , `true` .
aggregations	Returned aggregate result.

The following lists some common aggregate modes:

Regular aggregate

For a regular aggregate, you need to specify the aggregate name, mode (common modes include `min` , `max` , `avg` , `value_count` , and `sum`), and target fields.

Sample code for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
       "size":0,
       "query": {
           "terms": {
               "region": ["sh", "bj"]
           }
       },
       "aggs": {
           "myname": {
               "max": {
                   "field": "cpuUsage"
               }
           }
       }
   }'
```

Notes

The above sample aggregates the `cpuUsage` field in `max` mode (you can also use other modes such as `min` and `avg`), and the returned aggregate results are named the alias `myname` field (you can also specify another name).

Returned result:

```
{
 "took": 1,
 "timed_out": false,
```

```
  "_shards": {
    "total": 20,
    "successful": 20,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
              "value":7,
              "relation":"eq"
          },
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "myname": {
      "value": 4
    }
  }
}
```

`terms` aggregate

A `terms` aggregate is mainly used to query all the unique values and the number of such values of a field. You can specify the rule of sorting the returned unique values and the number of returned results and perform fuzzy or exact match for data fields participating in the aggregate. For more information, see the sample below. You can use the `filter_path` parameter to customize the returned result fields as instructed in Bulk Querying Data.

Sample code for curl:

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search?filter_path=aggregations -d'
{
    "aggs": {
        "myname": {
            "terms": {"field":"region"}
        }
    }
}'
```

Response:

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
```

```
        "sum_other_doc_count": 0,
        "buckets": [
          {
            "key": "sh",
            "doc_count": 10
          },
          {
            "key": "Motor_sports",
            "doc_count": 6
          },
          {
            "key": "gz",
            "doc_count": 3
          },
          {
            "key": "bj",
            "doc_count": 2
          },
          {
            "key": "cd",
            "doc_count": 2
          },
          {
            "key": "Winter_sports",
            "doc_count": 1
          },
          {
            "key": "water_sports",
            "doc_count": 1
          }
        ]
      }
    }
  }
```

Notes

The above sample returns all the unique values and their numbers of occurrences in the `region` field in `ctsdb_test`. By analyzing the `buckets` field in the returned `aggregations` field, you can find that the `region` field has 7 types of values, i.e., `sh`, `Motor_sports`, `gz`, `bj`, `cd`, `Winter_sports`, and `water_sports`, and the occurrence number of each value of the returned fields is indicated in `doc_count`. You can use the `size` field to specify the number of unique values to be returned; for example, if the `region` field has 7 unique values, you can set the `size` field to 5 to return only the first 5 values. For more information, see the sample.

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search?filter_path=aggregations -d'
{
    "aggs": {
        "myname": {
            "terms": {
               "field":"region",
                "size":5
            }
        }
    }
}'
```

Response:

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 2,
      "buckets": [
        {
          "key": "sh",
          "doc_count": 10
        },
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "gz",
          "doc_count": 3
        },
        {
          "key": "bj",
          "doc_count": 2
        },
        {
          "key": "cd",
          "doc_count": 2
        }
      ]
    }
  }
}
```

You can sort the returned results as instructed in the sample below:

1. Request (the returned results are sorted by number of unique values in descending order):

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search?filter_path=aggregations -d'
   {
     "aggs": {
         "myname": {
             "terms": {
               "field":"region",
               "order":{"_count":"desc"}
             }
         }
     }
   }'
```

Response:

```
  {
     "aggregations": {
       "myname": {
         "doc_count_error_upper_bound": 0,
         "sum_other_doc_count": 0,
         "buckets": [
           {
             "key": "sh",
             "doc_count": 10
           },
           {
             "key": "Motor_sports",
             "doc_count": 6
           },
           {
             "key": "gz",
             "doc_count": 3
           },
           {
             "key": "bj",
             "doc_count": 2
           },
           {
             "key": "cd",
             "doc_count": 2
           },
           {
             "key": "Winter_sports",
             "doc_count": 1
```

```
            },
            {
              "key": "water_sports",
              "doc_count": 1
            }
          ]
        }
      }
    }
```

2. Request (the returned results are sorted alphabetically in ascending order):

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search?filter_path=aggregations -d'
   {
     "aggs": {
         "myname": {
             "terms": {
               "field":"region",
               "order":{"_term":"asc"}
             }
         }
     }
   }'
```

Response:

```
   {
     "aggregations": {
       "myname": {
         "doc_count_error_upper_bound": 0,
         "sum_other_doc_count": 0,
         "buckets": [
           {
             "key": "Motor_sports",
             "doc_count": 6
           },
           {
             "key": "Winter_sports",
             "doc_count": 1
           },
           {
             "key": "bj",
             "doc_count": 2
           },
           {
             "key": "cd",
```

```
            "doc_count": 2
          },
          {
            "key": "gz",
            "doc_count": 3
          },
          {
            "key": "sh",
            "doc_count": 10
          },
          {
            "key": "water_sports",
            "doc_count": 1
          }
        ]
      }
    }
  }
```

You can set a regular expression to fuzzily match fields in the `terms` aggregate or exactly match specified fields as instructed in the sample below:

Sample code of fuzzy match:

Request (only the `region` fields whose values contain `sport` and don't start with `water_` are aggregated and returned):

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search?filter_path=aggregations -d'
{
 "aggs": {
     "myname": {
         "terms": {
           "field":"region",
           "include" : ".*sport.*",
             "exclude" : "water_.*"
         }
     }
 }
}'
```

Response:

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
```

```
      "buckets": [
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        }
      ]
    }
  }
}
```

Sample code of exact match:

Request (the `region` fields whose values are `sh`, `bj`, `cd`, and `gz` are aggregated in `region_zone`, and the `region` fields whose values are not `sh`, `bj`, `cd`, and `gz` are aggregated in `region_sports`):

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search?filter_path=aggregations -d'
{
"aggs" : {
 "region_zone" : {
   "terms" : {
     "field" : "region",
     "include" : ["sh", "bj","cd","gz"]
     }
   },
   "region_sports" : {
    "terms" : {
        "field" : "region",
        "exclude" : ["sh", "bj","cd","gz"]
    }
   }
 }
}'
```

Response:

```
{
  "aggregations": {
    "region_sport": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
```

```
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        },
        {
          "key": "water_sports",
          "doc_count": 1
        }
      ]
    },
    "region_zone": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "sh",
          "doc_count": 10
        },
        {
          "key": "gz",
          "doc_count": 3
        },
        {
          "key": "bj",
          "doc_count": 2
        },
        {
          "key": "cd",
          "doc_count": 2
        }
      ]
    }
  }
}
```

Date histogram aggregate

A date histogram is mainly used to aggregate dates into a histogram.

Sample code for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
```

```
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs": {
            "time_1h_agg": {
                "date_histogram": {
                    "field": "timestamp",
                    "interval": "1h"
                },
                "aggs": {
                    "avgCpuUsage": {
                        "avg": {
                            "field": "cpuUsage"
                        }
                    }
                }
            }
        }
    }'
```

Response:

```
    {
      "took": 5,
      "timed_out": false,
      "_shards": {
        "total": 20,
        "successful": 20,
        "skipped": 0,
        "failed": 0
      },
      "hits": {
        "total": {
             "value":6,
             "relation":"eq"
         },
        "max_score": 0.074107975,
        "hits": [
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2QtGR5xcjRaw2ETf-",
            "_score": 0.074107975,
            "_routing": "sh"
```

```
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2QtGR5xcjRaw2ETf_",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2Q5Xr5xcjRaw2ETgA",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2Q5Xr5xcjRaw2ETgB",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2RGfF5xcjRaw2ETgC",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2RGfF5xcjRaw2ETgD",
          "_score": 0.074107975,
          "_routing": "sh"
        }
      ]
    },
    "aggregations": {
      "time_1h_agg": {
        "buckets": [
          {
            "key_as_string": "1520222400",
            "key": 1520222400000,
            "doc_count": 1,
            "avgCpuUsage": {
              "value": 2.5
```



```
        }
      },
      {
        "key_as_string": "1520226000",
        "key": 1520226000000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520229600",
        "key": 1520229600000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520233200",
        "key": 1520233200000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520236800",
        "key": 1520236800000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520240400",
        "key": 1520240400000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520244000",
        "key": 1520244000000,
        "doc_count": 0,
        "avgCpuUsage": {
```

```
              "value": null
            }
          },
          {
            "key_as_string": "1520247600",
            "key": 1520247600000,
            "doc_count": 1,
            "avgCpuUsage": {
              "value": 2
            }
          },
          {
            "key_as_string": "1520251200",
            "key": 1520251200000,
            "doc_count": 4,
            "avgCpuUsage": {
              "value": 2.25
            }
          }
        ]
      }
    }
  }
```

Notes

The above sample aggregates the `cpuUsage` field in `date_histogram` mode with a granularity of one hour. The total aggregate name of the returned results is `time_1h_agg` (you can specify another name), and the aggregate name in each time interval is `avgCpuUsage` (you can specify another name). The valid time granularities for `interval` include `year`, `quarter`, `month`, `week`, `day`, `hour`, `minute`, and `second`. You can also represent the time granularity as a time unit; for example, `1y` represents one year, and `1h` one hour. The system does not support decimal time units; therefore, you need to convert `1.5h` to `90min` for example.

Percentiles aggregate

You can specify the percentile in percentiles aggregate. The system default percentiles are 1, 5, 25, 50, 75, 95, and 99. You can select other values as needed.

Sample code for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "query": {
        "terms": {
            "region": ["sh", "bj"]
```

```
        }
    },
    "aggs":
    {
        "myname":
        {
            "percentiles":
            {
                "field": "cpuUsage",
                "percents": [1,25,50,70,99]
            }
        }
    }
}'
```

Response:

```
  {
    "took": 18,
    "timed_out": false,
    "_shards": {
      "total": 20,
      "successful": 20,
      "skipped": 0,
      "failed": 0
    },
    "hits": {
      "total": {
           "value":6,
           "relation":"eq"
       },
      "max_score": 0.074107975,
      "hits": [
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2QtGR5xcjRaw2ETf-",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2QtGR5xcjRaw2ETf_",
          "_score": 0.074107975,
          "_routing": "sh"
        },
```

```
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2Q5Xr5xcjRaw2ETgA",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2Q5Xr5xcjRaw2ETgB",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2RGfF5xcjRaw2ETgC",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2RGfF5xcjRaw2ETgD",
          "_score": 0.074107975,
          "_routing": "sh"
        }
      ]
    },
    "aggregations": {
      "myname": {
        "values": {
          "1.0": 2,
          "25.0": 2,
          "50.0": 2.25,
          "70.0": 2.5,
          "99.0": 2.5
        }
      }
    }
  }
```


Notes

The above sample aggregates the `cpuUsage` field in `percentiles` mode, and the selected percentiles are 1, 25, 50, 70, and 99. The returned aggregate results are named the alias `myname` (you can also specify another

name).

Cardinality aggregate

A cardinality aggregate is mainly used to get the number of deduplicated results. By default, if the number of aggregate results is less than or equal to 3,000, the result returned by `cardinality` will be precise; otherwise, the result will be approximate.

Sample code for curl:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "query": {
        "terms": {
            "region": ["sh", "bj"]
        }
    },
    "aggs":
    {
        "myname":
        {
            "cardinality":
            {
                "field": "cpuUsage"
            }
        }
    }
}'
```

Response:

```
   {
     "took": 15,
     "timed_out": false,
     "_shards": {
       "total": 20,
       "successful": 20,
       "skipped": 0,
       "failed": 0
     },
     "hits": {
       "total": {
            "value":6,
            "relation":"eq"
        },
       "max_score": 0.074107975,
       "hits": [
```

```
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2QtGR5xcjRaw2ETf-",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2QtGR5xcjRaw2ETf_",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2Q5Xr5xcjRaw2ETgA",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2Q5Xr5xcjRaw2ETgB",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2RGfF5xcjRaw2ETgC",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2RGfF5xcjRaw2ETgD",
          "_score": 0.074107975,
          "_routing": "sh"
        }
      ]
    },
    "aggregations": {
      "myname": {
        "value": 2
```

```
            }
        }
    }
```

Notes

The above sample aggregates the `cpuUsage` field in `cardinality` mode, and the returned aggregate result is named the alias `myname` (you can also specify another name).

SQL-Like Query Support

Last updated：2022-06-09 10:57:11

CTSDB 2.0 supports using SQL-like statements for querying, which is only for the convenience of beginners. If you are concerned about the query time, we recommend you use CTSDB's native query statements.

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `_nlpcn/sql`.
Method: GET

Request parameters

You can add the `pretty` parameter value when querying to get the response in a sorted format. For details, see the sample.

Request content

Queries mainly include regular queries and aggregate queries. For the specific request content, see the samples.

Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description. If the parameter or metric name is incorrect, you need to check and correct it on your own.

Sample code for curl

For specific queries, see the following samples:
The metric structure used by all samples is as follows:

```
    {
        "ctsdb_test" : {
          "tags" : {
            "region" : "string"
          },
          "time" : {
            "name" : "timestamp",
            "format" : "epoch_millis"
          },
          "fields" : {
```

```
            "cpuUsage" : "float",
            "diskUsage" : "string",
            "dcpuUsage" : "integer"
          },
          "options" : {
            "expire_day" : 7,
            "refresh_interval" : "10s",
            "number_of_shards" : 5
          }
        }
    }
```

Sample code of regular query for curl:

```
curl -u root:le201909 -H 'Content-Type: application/json' -XGET
172.16.345.14:9201/_nlpcn/sql?pretty -d 'select docvalue(cpuUsage) from
ctsdb_test limit 1'
```

Sample response of regular query:

```
{
  "took" : 22,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "ctsdb_test@1646064000000_1",
        "_type" : "_doc",
        "_id" : "XMzBRX8BfqojiIOn8jFa",
        "_score" : 1.0,
        "fields" : {
          "cpuUsage" : [
            10.0
          ]
        }
      }
```

```
    ]
  }
}
```

Note:

During regular query, `docvalue` should be added to the field to be queried, so that the instance can get the data from its columnar storage.

Sample code of regular query with search criteria for curl:

```
curl -u root:le201909 -H 'Content-Type: application/json' -XGET
172.16.345.14:9201/_nlpcn/sql?pretty -d 'select docvalue(cpuUsage) from
ctsdb_test where region="shanghai" limit 1'
```

Sample response of regular query with search criteria:

```
{
  "took" : 13,
  "timed_out" : false,
  "_shards" : {
    "total" : 10,
    "successful" : 10,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 0.0,
    "hits" : [
      {
        "_index" : "ctsdb_test@1646064000000_1",
        "_type" : "_doc",
        "_id" : "XszBRX8BfqojiIOn8jFb",
        "_score" : 0.0,
        "fields" : {
          "cpuUsage" : [
            30.0
          ]
        }
      }
    ]
  }
}
```

Note:

For regular queries with search criteria, there is no need to add `docvalue` to the fields used as criteria.

Sample code of aggregate query grouped by tag for curl:

```
curl -u root:le201909 -H 'Content-Type: application/json' -XGET
172.16.345.14:9201/_nlpcn/sql?pretty -d 'select max(cpuUsage) from ctsdb_test
group by region'
```

Sample response of aggregate query grouped by tag:

```
{
  "took" : 33,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "region" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "beijing",
          "doc_count" : 1,
          "MAX(cpuUsage)" : {
            "value" : 20.0
          }
        },
        {
          "key" : "chengdu",
          "doc_count" : 1,
          "MAX(cpuUsage)" : {
            "value" : 10.0
          }
        },
        {
```

```
            "key" : "shanghai",
            "doc_count" : 1,
            "MAX(cpuUsage)" : {
              "value" : 30.0
            }
          }
        ]
      }
    }
}
```

Note:

For aggregate queries grouped by tag, there is no need to add `docvalue` to the field used for grouping.

Sample code of aggregate query grouped by time for curl:

```
curl -u root:le201909 -H 'Content-Type: application/json' -XGET
172.16.345.14:9201/_nlpcn/sql?pretty -d 'select max(cpuUsage) from ctsdb_test
GROUP BY date_histogram(field="timestamp","interval"="1d")'
```

Sample response of aggregate query grouped by time:

```
{
  "took" : 30,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "date_histogram(field=timestamp,interval=1d)" : {
      "buckets" : [
        {
          "key_as_string" : "2022-03-01 00:00:00",
          "key" : 1646092800000,
          "doc_count" : 3,
          "MAX(cpuUsage)" : {
            "value" : 30.0
```

```
                }
            }
        ]
    }
  }
}
```

Bulk Writing Data

Last updated：2022-06-09 15:07:42

Bulk Writing Data into One Metric

This API can be used to write data records into a single metric either in bulk or not. To improve the write efficiency, we recommend that you write data in bulk.

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Request path: `/${metric_name}/_doc/_bulk`, where `${metric_name}` is the metric name.
Method: POST
Note:
The `_doc` keyword is the `_type` of the written data and must be added to facilitate subsequent system parsing and upgrade.

Request parameters

You can set the `filter_path` parameter to filter and simplify the returned results. For more information, see Bulk Querying Data.

Request content

You need to bulk write data as structured data in NDJSON format into a metric, which is similar to the following:

```
Metadata\\n
Data to be written\\n
....
Metadata\\n
Data to be written\\n
```

The format of the metadata is as shown below:

```
{
"index" :
{
"_id" : "1",                # Document ID (optional)
}
}
```

The format of the written data is as shown below:

```
{
"field1" : "value1",
"field2" : "value2"
}
```

Response

You should note that the returned result of the bulk data write API is different from those of other APIs. Keep in mind the `errors` (not `error`) field in the JSON result first. If it is `false`, all data records were successfully written; if it is `true`, some data records failed to be written, and you can get the failure details from the `items` field.

The `items` field is an array, where each element corresponds to a write request. You can check whether each element has the `error` field to judge whether the corresponding request is successful. If the `error` field exists, the request failed. The specific error information is in the `error` field. If the `error` field doesn't exist, the request succeeded.

Sample code for curl

Sample response upon success

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/ctsdb_test/_doc/_bulk -d'
 {"index":{"routing": "sh" }}
 {"region":"sh","cpuUsage":2.5,"timestamp":1505294654}
 {"index":{"routing": "sh" }}
 {"region":"sh","cpuUsage":2.0,"timestamp":1505294654}
'
```

Response:

```
{
   "took": 134,
   "errors": false,
   "items":
[
    {
        "index":
        {
            "_index": "ctsdb_test@1505232000000_1",
            "_type": "_doc",
            "_id": "AV_8eeo_UAkC9PF9L-2q",
```

```
            "_version": 1,
            "result": "created",
            "_shards":
             {
                "total": 2,
                "successful": 2,
                "failed": 0
            },
            "created": true,
            "status": 201
            }
        },
        {
        "index":
        {
            "_index": "ctsdb_test2@1505232000000_1",
            "_type": "_doc",
            "_id": "AV_8eeo_UAkC9PF9L-2r",
            "_version": 1,
            "result": "created",
            "_shards":
            {
                "total": 2,
                "successful": 2,
                "failed": 0
            },
            "created": true,
            "status": 201
        }
     }
    ]
}
```


Note:

The `errors` value returned above is `false`, indicating that all data records were successfully written. The `items` array indicates the write result of each record in the same order as in the `bulk` request. For single records in `items`, the `status` of `2XX` indicates that the record was successfully written. `_index` indicates the child metric as described in Creating Metric where the data was written, and `_shards` indicates the replica write status. In the above sample, `total` indicates two replicas, and `successful` indicates that data was successfully written into both replicas.

Sample response upon failure

Request:


```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/hcbs_client_trace/_doc/_bulk -d'
    {"index":{"_id":"5"}}
    {"vol_id":"c57e008c-0ae0-41cd-8da8-
6989d0522fc6","io_type":2,"data_len":4096,"latency":3,"try_times":1,"errcode":0
,"start_time":1503404266}
    {"index":{"_id":"6"}}
    {"vol_id":"c57e008c-0ae0-41cd-8da8-
6989d0522fc6","io_type":"abc","data_len":4096,"latency":1,"try_times":1,"errcod
e":0,"start_time":1503404266}
'
```

Response:

```
{
    "took":71,
    "errors":true,
    "items":[
        {
            "index":{
                "_index":"hcbs_client_trace@1505232000000_1",
                "_type":"_doc",
                "_id":"5",
                "_version":1,
                "result":"created",
                "_shards":{
                    "total":2,
                    "successful":2,
                    "failed":0
                },
                "_seq_no":0,
                "_primary_term":1,
                "status":201
            }
        },
        {
            "index":{
                "_index":"hcbs_client_trace@1505232000000_1",
                "_type":"_doc",
                "_id":"6",
                "status":400,
                "error":{
                    "type":"illegal_argument_exception",
                    "reason":"mapper [io_type] cannot be changed from type
[long] to [keyword]"
                }
```

```
                }
            }
        ]
    }
```

Note:

The `errors` value returned above is `true`, indicating that some data records failed to be written. The `items` array indicates the write result of each record in the same order as in the `bulk` request. For single records in `items`, the `status` of `2XX` indicates that the record was successfully written. The `error` field indicates the detailed error information. `_index` indicates the child metric where the data was written, and `_shards` indicates the replica write status. In the above sample, `total` indicates two replicas, and `successful` indicates that data was successfully written into both replicas.

Bulk Writing Data into Multiple Metrics

This API can be used to write data records into multiple metrics either in bulk or not. To improve the write efficiency, we recommend that you write data in bulk.

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Request path: `_bulk`

Method: PUT

Request parameters

You can set the `filter_path` parameter to filter and simplify the returned results. For more information, see Bulk Querying Data.

Request content

You need to bulk write data as structured data in NDJSON format into a metric, which is similar to the following:

```
Metadata\\n
Data to be written\\n
....
Metadata\\n
Data to be written\\n
```

The format of the metadata is as shown below:

```
{
"index" :
{
    "_index" :"metric_name",    # Metric where data is to be written
    "_id" : "1",                # Document ID (optional)
}
}
```

The format of the written data is as shown below:

```
{
"field1" : "value1",
"field2" : "value2"
}
```

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. The specific error information is in the `error` field. If the request succeeded but the `errors` (not `error`) field is not `false`, the specific data that failed to be written is indicated in the `errors` field.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.xx.xx.4:9201/_bulk -d'
{"index":{"_index" : "ctsdb_test"}}
{"region":"sh","cpuUsage":2.5,"timestamp":1505294654}
{"index":{"_index" : "ctsdb_test2"}}
{"region":"sh","cpuUsage":2.0,"timestamp":1505294654}
'
```

Response:

```
{
    "took":494,
    "errors":false,
    "items":[
        {
            "index":{
                "_index":"ctsdb_test_ay@1505232000000_1",
                "_type":"_doc",
                "_id":"GgJiyXsBE2F__WP8B_ik",
                "_version":1,
```

```
                "result":"created",
                "_shards":{
                    "total":2,
                    "successful":1,
                    "failed":0
                },
                "_seq_no":0,
                "_primary_term":1,
                "status":201
            }
        },
        {
            "index":{
                "_index":"ctsdb_test2_ay@1505232000000_1",
                "_type":"_doc",
                "_id":"GwJiyXsBE2F__WP8B_ik",
                "_version":1,
                "result":"created",
                "_shards":{
                    "total":2,
                    "successful":1,
                    "failed":0
                },
                "_seq_no":0,
                "_primary_term":1,
                "status":201
            }
        }
    ]
}
```


Note:

The `errors` value returned above is `false`, indicating that all data records were successfully written. The `items` array indicates the write result of each record in the same order as in the `bulk` request. For single records in `items`, the `status` of `2XX` indicates that the record was successfully written. `_index` indicates the child metric where the data was written, and `_shards` indicates the replica write status. In the above sample, `total` indicates two replicas, and `successful` indicates that data was successfully written into both replicas.

Deleting Data

Last updated：2023-03-02 14:37:52

Request Address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Request path: `/${metric_name}/_delete_by_query`, where `${metric_name}` is the metric name.
Method: POST

Request Parameters

None

Request Content

Query conditions when a metric is deleted. For more information, see the sample.

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. The error details are in the `error` field.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/ctsdb_test/_delete_by_query -d'
{
"query": {
"bool": {
    "filter": {
        "match_all": {}
```

```
        }
    }
}
}'
```

Response:

```
{
"took": 43,
"timed_out": false,
"total": 1,
"deleted": 1,
"batches": 1,
"version_conflicts": 0,
"noops": 0,
"retries": {
    "bulk": 0,
    "search": 0
   }
}
```

Modifying Data

Last updated：2023-03-02 14:38:30

In CTSDB, you can modify data by deleting it first and then writing it again. For directions on data deletion and write, see Deleting Data and Bulk Writing Data.

Rollup Operations

Last updated：2025-01-03 11:42:16

Creating rollup task

In scenarios with massive amounts of data, a business system can generate petabytes of data per day or even per hour. Time series data is generally massive in amount, time-sensitive, and trending. Therefore, systems using such data (e.g., monitoring or data analysis systems) often only need high-precision data in the most recent time period while downsampling historical data for storage. You can configure a rollup task to periodically aggregate historical data and save it into a new metric. A rollup task not only saves downsampled historical data, but also improves query performance and reduces storage costs. Note that a rollup task automatically creates child metrics inheriting all configurations of the parent metric according to `base_metric`, and the parent metric configurations will be overwritten if `options` is specified.

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

2. Request path and method

Path: `/_rollup/${rollup_task_name}`, where `${rollup_task_name}` is the rollup task name.
Method: PUT

3. Request parameters

None

4. Request content

Parameter	Required	Type	Description
base_metric	Yes	string	Name of the metric (parent metric) depended on by the rollup task
rollup_metric	Yes	string	Name of the metric (child metric) generated by the rollup task
base_rollup	No	string	Rollup task depended on by the current rollup task. Before executing the current task, the system will check whether the dependent task in the corresponding time period has been completed
query	No	string	Query condition for data filtering, which consist of many elements and operations, such as `name:host AND type:max OR region:gz`
group_by	Yes	Array	Tag to be aggregated. Multiple tags can be included

function	Yes	Map	Aggregate name, mode, and fields. The fields can be only from the `fields` field in `base_metric`. If the `fields` field is empty, the rollup cannot be configured. The function can be `sum`, `avg`, `min`, `max`, `set`, `any`, `first`, `last`, `percentiles`, etc., such as `{"cost_total":{"sum":{"field":"cost"}},"cpu_usage_avg":{ "avg": { "field":"cpu_usage"}}}`
interval	Yes	string	Aggregate granularity, such as 1s, 5m, 1h, and 1d
frequency	No	string	Scheduling frequency, such as 5m, 1h, and 1d, which is the same as `interval` by default
delay	No	string	Execution delay. Generally, there should be a certain delay for data write, such as 5m or 1h, in order to avoid data loss
start_time	No	string	Start time of periodic execution of the rollup task, which is the current time by default
end_time	No	string	Scheduling end time, which is the maximum timestamp value by default
options	No	map	`rollup_metric` options, which are the same as the options for metric creation

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

6. Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/_rollup/ctsdb_rollup_task_test -d'
{
   "base_metric": %{base_metric_name},
   "rollup_metric": %{rollup_metric_name},
   "base_rollup": %{base_rollup_name},
   "query" : "name:host AND type:max",
   "group_by": ["host"],
   "function": {
       "cost_total":  {
           "sum": {
               "field": "cost"
           }
```

```
        },
        "cpu_usage_avg": {
            "avg": {
                "field": "cpu_usage"
            }
        },
        "value": {
          "percentiles": {
            "field": "value",
            "percents": [
              95
            ]
          }
        },
        "metricName": {
          "set": {
            "value": "cpu_usage"
          }
        },
        "appid": {
          "any": {
            "field": "appid"
          }
        },
         "first_value": {
          "first": {
            "field": "value"
          }
        },
         "last_value": {
          "last": {
            "field": "value"
          }
        }
    },
    "interval": "1m",
    "frequency": "5m",
    "delay": "1m",
    "start_time": "1502892000",
    "end_time": "2147483647",
    "options": {
        "expire_day": 365
    }
}
'
```

Response:

```
{
   "acknowledged": true,
   "message": "create rollup success"
}
```

Getting names of all rollup tasks

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

2. Request path and method

Path: `/_rollups`

Method: GET

3. Request parameters

None

4. Request content

None

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

6. Sample code for curl

Request:

`curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_rollups`

Response:

```
{
   "result":
{
    "rollups":
    [
        "rollup_jgq_6",
        "rollup_jgq_60"
    ]
   },
```

```
    "status": 200
}
```

Getting rollup task details

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200` .

2. Request path and method

Path: `/_rollup/${rollup_task_name}` , where `${rollup_task_name}` is the rollup task name.
Method: GET

3. Request parameters

You can specify the `v` parameter to view the specific rollup progress. `@last_end_time` in the response structure is the latest rollup progress.

4. Request content

None

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

6. Sample code for curl

Request:

`curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_rollup/rollup_jgq_6?v`

Response:

```
{
 "result": {
   "rollup_jgq_6": {
   "base_metric": "cvm_device-300",
      "rollup_metric": "cvm_device-86400",
      "query": "metricName:cpu_usage AND statType:max",
      "group_by": [
        "vm_uuid"
      ],
      "function": {
```

```
          "value": {
            "percentiles": {
              "field": "value",
              "percents": [
                95
              ]
            }
          },
          "metricName": {
            "set": {
              "value": "cpu_usage"
            }
          },
          "appid": {
            "any": {
              "field": "appid"
            }
          }
        },
        "interval": "1d",
        "delay": "5m",
        "options": {
          "expire_day": 186
        },
        "frequency": "1d",
        "start_time": 1534003200,
        "end_time": 2147483647,
        "@state": "running",              // Running status
        "@timestamp": 1550766085000,      // Rollup task information update time
point
        "@last_end_time": 1550764800      // Rollup task end time point for proper
execution
      }
  },
  "status": 200
}
```

Deleting rollup task

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200` .

2. Request path and method

Path: `/_rollup/${rollup_task_name}` , where `${rollup_task_name}` is the rollup task name.
Method: DELETE

3. Request parameters

None

4. Request content

None

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

6. Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X DELETE 172.xx.xx.4:9201/_rollup/ctsdb_rollup_task_test
```

Response:

```
{
   "acknowledged": true,
   "message": "delete rollup success"
}
```

Updating rollup task

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200` .

2. Request path and method

Path: `/_rollup/${rollup_task_name}/update` , where `${rollup_task_name}` is the rollup task name.
Method: POST

3. Request parameters

None

4. Request content

Parameter	Required	Type	Description
state	Yes	string	Valid values: running, pause
start_time	No	string	Start time of periodic execution of the rollup task, which is the current time by default
end_time	No	string	Scheduling end time, which is the maximum timestamp value by default
options	No	map	Aggregate options, which are the same as the options for metric creation

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

6. Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/_rollup/ctsdb_rollup_task_test/update -d'
{
   "state":"running",
   "start_time": "1511918989",
   "end_time": "1512019765",
   "options":
{
       "expire_day": 365
   }
}'
```

Response:

```
{
   "acknowledged": true,
   "message": "update rollup success"
}
```

CTSDB 1.0
Overview

Last updated：2025-01-03 10:42:08

CTSDB supports data write, query, and other operations over the HTTP protocol. Its HTTP APIs are RESTful, and the resource request method is to send standard HTTP requests to the corresponding URIs of resources. For example, `GET` is used to get resources, `POST` create (or update) resources, `PUT` update resources, and `DELETE` delete resources.
You can perform almost all data operations through HTTP APIs. CTSDB ensures data security by providing VPC-based network isolation and access authentication with username and password. Data is exchanged through structures in JSON format. Each request will return a standard HTTP response status code and content. If an operation fails, you can get specific error information based on the response content.
For more information on how to connect to instances through RESTful API, please see Connecting to Instance.

Naming Conventions

The names of metrics, tags, and fields in CTSDB should be concise and clear. For more information on tags and fields, please see Request Content:
Naming conventions for metrics: a name can contain 1–200 lowercase letters, digits, underscores, hyphens, and dots but cannot start with a dot, underscore, or hyphen.
Naming conventions for tags and fields in metrics: a name can contain 1–255 letters, digits, underscores, hyphens, and dots but cannot start with a dot.

System Limits

Limits of tags and fields in metrics: the total number of fields in a metric cannot exceed 1,000.
Limits of write points during bulk writing to metric: we recommend you limit the number of records in each `bulk` request between 1,000 and 5,000 and the physical size between 1 and 15 MB.

System Default Rules

Processing of new fields during data write: if there are undefined new fields when you write data to a metric, CTSDB will store the new fields as tags by default. You can also change this by modifying the `default_type` field in the `options` parameter of the metric. For more information on metric modification, please see Updating Metric.

If a new field is an integer, CTSDB will store it in `long` type.
If a new field is a decimal number, CTSDB will store it in `float` type.
If a new field is a string, CTSDB will store it in `string` type subject to the value of `max_string_length` and discard the excessive part. `max_string_length` can be customized and is 256 characters by default. For more information on modification, please see Updating Metric.
Processing of date and time: date and time are stored in UTC format in CTSDB. Therefore, for data queries involving time range, please specify the time zone with the `time_zone` parameter, which is in the format of ISO 8601 UTC offset (e.g., +01:00 or -08:00). The specific time zone needs to be determined according to the region where the instance resides. Generally, the time zone for the Chinese mainland is UTC+8. For more information, please see Common Query Samples.

Instance Connection

CTSDB instances currently can be connected to only in VPCs. You can connect to an instance in the console or through RESTful APIs. In the latter case, you need to provide the root account password to ensure security.
Below is a code sample for connecting to an instance and creating a metric through curl. Here, `${user:password}` is the username and password of the instance, `${vip}:${vport}` the IP and port, and `${metric_name}` the name of the created metric.

```
curl -u ${user:password} -H 'Content-Type:application/json' -X PUT
${vip}:${vport}/_metric/${metric_name} -d'
{
    "tags": {
        "region": "string",
        "set":  "long",
        "host": "string"
    },
    "time": {
         "name": "timestamp",
         "format": "epoch_second"
    },
    "fields": {
        "cpu_usage":  "float"
    },
    "options": {
        "expire_day": 7,
        "refresh_interval": "10s",
        "number_of_shards": 5,
        "number_of_replicas": 1,
        "rolling_period": 1
    }
}'
```

Creating Metric

Last updated：2025-01-03 10:41:33

Request Address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request Path and Method

Path: `/_metric/${metric_name}`, where `${metric_name}` is the name of the metric to be created.

Method: PUT

Note:

For the naming limits of metrics, please see System Limits.

Request Parameters

None

Request Content

Parameter	Required	Type	Description
tags	Yes	map	Tag, which is used to uniquely identify data. It must contain at least one tag ar (string without tokens), `long`, `integer`, `short`, `byte`, `double` The format is `{"region": "string","set": "long","host": "s`
time	Yes	map	Configuration of time column, which is used to store the unique time when dat `"epoch_second"}`. If this parameter is left empty, it will be specified as `{`
fields	No	map	Fields for data storage. We recommend you use data types most suitable for y `long`, `integer`, `short`, `byte`, `double`, `float`, `date`, Example: `{"cpu_usage":"float"}`
options	No	map	Common fine-tuning configuration information Example: `{"expire_day":7,"refresh_interval":"10s","number_of_sha`

			`256,"default_date_format":"strict_date_optional_time","i`

The `name` of the `time` field is of the `timestamp` type by default. The time formats (`format`) are fully compatible with those in Elasticsearch, such as `epoch_millis` (Unix timestamp in milliseconds), `epoch_second` (Unix timestamp in seconds), `basic_date` (in `yyyyMMdd` format), and `basic_date_time` (in `yyyyMMdd'T'HHmmss.SSSZ` format).

`options` values and their descriptions:

expire_day: data expiration time in days, which is a non-zero integer. Once expired, the data will be automatically cleared. By default, it is the minimum value -1, which indicates that the data will never expire.

refresh_interval: data refresh interval, which is 10 seconds by default. The written data can be queried after being refreshed from the memory to disk.

number_of_shards: number of metric shards, which is a positive integer and 3 by default. This parameter can be ignored for small metrics. A large metric can be divided into shards, and each shard can be up to 25 GB in size.

number_of_replicas: number of replicas, which is a positive integer; for example, 1 indicates one master and one replica. The default value is 1.

rolling_period: child metric period (in days), which is a non-zero integer. When CTSDB stores data, to facilitate data expiration and improve query efficiency, it stores data in **child metrics** by the specified time interval, which is subject to the data expiration time by default. The relationships between the default child metric period and data expiration time are as detailed below:

max_string_length: maximum length of a custom string, which is a positive integer. Its maximum value is 32765, and its default value is 256.

default_date_format: format of the `date` data type of custom tags and fields, which is `strict_date_optional_time` or `epoch_millis` by default.

indexed_fields: fields whose indexes need to be retained in the fields. You can use an array to specify multiple fields.

default_type: default type of new fields. Its valid values are `tag` and `field` , and its default value is `tag` .

Expiration Time	Child Metric Period
≤ 7 days	1 day
> 7 day and ≤ 20 days	3 days
> 20 days and ≤ 49 days	7 days
> 49 days and ≤ 3 months	15 days
> 3 months	30 days
Never expire	30 days

Response Content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

Sample Code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.16.345.14:9201/_metric/ctsdb_test -d'
 {
        "tags":
        {
                "region":"string"
        },
        "time":
        {
                "name":"timestamp",
                "format":"epoch_second"
        },
        "fields":
        {
                "cpuUsage":"float"
        },
        "options":
        {
                "expire_day":7,
                "refresh_interval":"10s",
                "number_of_shards":5
        }
    }
    '
```

Response upon success:

```
    {
        "acknowledged": true,
        "message": "create ctsdb metric ctsdb_test success!"
    }
```

Response upon failure:

```
   {
```

```
        "error": {
        "reason": "table ctsdb_test already exist",
        "type": "metric_exception"
        },
        "status": 201
    }
```

Querying Metric

Last updated：2025-01-03 10:37:35

Getting All Metrics

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_metrics`

Method: GET

Request parameters

None

Request content

None

Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_metrics
```

Response:

```
{
    "result":
    {
        "metrics":
        [
            "ctsdb_test",
            "ctsdb_test1"
        ]
    },
    "status": 200
```

```
}
```

Getting Specified Metric

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_metric/${metric_name}`, where `${metric_name}` is the metric name.

Method: GET

Request parameters

None

Request content

None

Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_metric/ctsdb_test
```

Response:

```
{
    "result":
    {
        "ctsdb_test":
        {
            "tags":
            {
                "region": "string"
            },
            "time":
            {
                "name": "timestamp",
```

```
                "format": "epoch_second"
            },
            "fields":
            {
                "cpuUsage": "float"
            },
            "options":
            {
                "expire_day": 7,
                "refresh_interval": "10s",
                "number_of_shards": 5
            }
        }
    },
    "status": 200
}
```

Updating Metric

Last updated：2023-03-02 14:41:33

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_metric/${metric_name}/update`, where `${metric_name}` is the metric name.
Method: PUT

Request parameters

None

Request content

The `tags`, `time`, `fields`, and `options` fields are all of the `map` type and optional. For their formats, see Creating Metric. The specific requirements are as detailed below:
tags: You can add tag fields and modify the types of existing ones without deleting them.
time: You cannot modify `name` but can modify `format`.
fields: You can add metric fields and modify the types of existing ones without deleting them.
`options` attributes are as detailed below:

Attribute name	Required	Type	Description
expire_day	No	integer	Data expiration time, which is a non-zero integer. Once expired, the data will be automatically cleared. By default, data never expires.
refresh_interval	No	string	Data refresh interval, which is 10 seconds by default. The written data can be queried after being refreshed from the memory to disk.
number_of_shards	No	integer	Number of metric shards, which is a positive integer and `3` by default. This parameter can be ignored for small metrics. A

			large metric can be divided into shards, and each shard can be up to 25 GB in size.
number_of_replicas	No	integer	Number of replicas, which is a non-negative integer; for example, 1 indicates one master and one replica. The default value is `1` .
rolling_period	No	integer	Child metric period (in days), which is a non-zero integer. When CTSDB stores data, to facilitate data expiration and improve query efficiency, it stores data into child metrics by the specified time interval, which is subject to the data expiration time by default.
max_string_length	No	integer	Maximum length of a custom string, which is a positive integer. Its maximum value is `32765` , and its default value is `256` .
default_date_format	No	string	Format of the `date` data type of custom tags and fields, which is `strict_date_optional_time` or `epoch_millis` by default.
indexed_fields	No	array	Fields whose indexes need to be retained in the fields. You can use an array to specify multiple fields.
default_type	No	string	Default type of new fields. Its valid values are `tag` and `field` , and its default value is `tag` .

Notes

As the historical data cannot be modified, after fields are updated, the metric information will not immediately change until the next child metric as described in Creating Metric is generated. If you want to confirm whether the update is successful, you can call the `GET /_metric/${metric_name}?v` API.

Only fields of the `short` , `integer` , `float` data types can be changed to another type. The `short` type can be changed to the `integer` or `long` type, `integer` to `long` , and `float` to `double` .

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

Request:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.16.345.14:9201/_metric/ctsdb_test/update -d'
    {
        "tags":{
            "set":"string"
        },
        "time":{
            "name":"timestamp",
            "format":"epoch_second"
        },
        "fields":{
            "diskUsage":"float"
        },
        "options":{
            "expire_day":15,
            "number_of_shards":10
        }
    }'
```

Response:

```
{
    "acknowledged": true,
    "message": "update ctsdb metric test111 success!"
}
```

Deleting Metric Field

Last updated：2025-01-03 10:36:57

Request Address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request Path and Method

Path: `/_metric/${metric_name}/delete`, where `${metric_name}` is the metric name.
Method: PUT

Request Parameters

None

Request Content

Parameter	Required	Type	Description
tags	No	Array	Enumeration of the tag fields to be deleted, such as `"tags": ["ip"]`
fields	No	Array	Enumeration of the data fields to be deleted, such as `"fields": ["diskUsage"]`

Note:
As historical data cannot be modified, after fields are deleted, the metric information will not change until the next child metric is generated. If you want to check whether the deletion is successful, you can call the `GET /_metric/${metric_name}?v` API.

Response Content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

Sample Code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.16.345.14:9201/_metric/ctsdb_test1/delete -d'
{
    "tags": ["ip"],
    "fields": ["cpu"]
}'
```

Response:

```
{
    "acknowledged": true,
    "message": "update ctsdb_test1 metric success!"
}
```

Deleting Metric

Last updated：2025-01-03 10:36:24

Request Address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request Path and Method

Path: `/_metric/${metric_name}`, where `${metric_name}` is the name of the metric to be deleted.
Method: DELETE

Request Parameters

None

Request Content

None

Response Content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

Sample Code for curl

Request:

```
DELETE /_metric/ctsdb_test1
```

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X DELETE
172.16.345.14:9201/_metric/ctsdb_test1
```

Response:

```
{
    "acknowledged": true,
    "message": "delete metric ctsdb_test1 success!"
}
```

Querying Data

Last updated：2023-03-02 14:41:56

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `${metric_name}/_search`, where `${metric_name}` is the metric name.
Method: GET

Request parameters

You can set the `_routing` parameter during data write to improve the query efficiency. For more information, see Common Query Samples.

Request content

Queries mainly include regular queries and aggregate queries. Query requests are fully compatible with Elasticsearch APIs as described in Getting started with Elasticsearch. For the specific request content, see Common Query Samples.

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description.

Sample code for curl

For specific query samples, see Common Query Samples.

Batch Querying Data

Last updated：2023-03-02 14:42:21

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Path: `/_msearch`
Method: GET

Request parameters

You can set the `_routing` parameter during data write to improve the query efficiency as instructed in the sample. You can also set the `filter_path` parameter to filter and simplify the returned results. Its value contains multiple JSON paths separated by comma, and each JSON path is constructed by concatenating elements with dots. You can also use the `*` wildcard to match any character, the `* *` wildcard to match any path, and the `-` prefix to remove certain returned fields.
For example, `responses._shards.f*` indicates fields starting with "f" in `_shards` in `responses`, and `responses.* * ._score` indicates all `_score` fields in `responses`.

Request content

Queries mainly include regular queries and aggregate queries. Query requests are fully compatible with Elasticsearch APIs. For the specific request content, see the samples.

Response

You can check whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, see the `error` field description. Some common returned fields for queries are as detailed below. You can set the `filter_path` parameter to simplify the returned query results.

Field name	Description
hits	Matched query results. Here, the `total` field indicates the number of matched data records. The `hits` field is an array, which contains the first ten query results if not specified. Each result in the `hits` array contains `_index` (child metric as described in Creating Metric involved in the query). If `docvalue_fields` is specified in the query, the `fields` field will be returned to indicate the value of each field.
took	Time in milliseconds taken by the entire query.
_shards	Number of shards involved in the query. Here, `total` indicates the total number of shards, `successful` the shards that were successfully queried, `failed` the shards that failed to be queried, and `skipped` skipped shards.
timed_out	Indicates whether query timed out. Valid values: `false` , `true` .
status	Status code returned for the query. `2XX` indicates success.

Sample code for curl

Request without the `filter_path` parameter:

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/_msearch -d'
{"index" : "amyli_weather","routing":"host_20"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
{"index" : "ctsdb_test2017","routing":"host_100"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
'
```

Response:

```
{
      "responses":
        [
            {
              "took": 0,
              "timed_out": false,
              "_shards": {
                "total": 0,
                "successful": 0,
                "skipped": 0,
                "failed": 0
```

```
            },
            "hits": {
              "total": 0,
              "max_score": 0,
              "hits": []
            },
            "status": 200
          },
          {
            "took": 1,
            "timed_out": false,
            "_shards": {
              "total": 3,
              "successful": 3,
              "skipped": 0,
              "failed": 0
            },
            "hits": {
              "total": 1,
              "max_score": 1,
              "hits": [
                {
                  "_index": "ctsdb_test2017@-979200000_30",
                  "_type": "doc",
                  "_id": "AV_8fBhlUAkC9PF9L-2t",
                  "_score": 1
                }
              ]
            },
            "status": 200
          }
        ]
    }
```

Request with the `filter_path` parameter:

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/_msearch?
filter_path=responses.\\*\\*.hits,responses._shards.f\\*,-
responses.\\*\\*._score -d'
{"index" : "amyli_weather","routing":"host_20"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
{"index" : "ctsdb_test2017","routing":"host_100"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
'
```

Response:

```
{
       "responses": [
     {
       "_shards": {
         "failed": 0
       }
     },
     {
       "_shards": {
         "failed": 0
       },
       "hits": {
         "hits": [
           {
             "_index": "ctsdb_test2017@-979200000_30",
             "_type": "doc",
             "_id": "AV_8fBhlUAkC9PF9L-2t"
           }
         ]
       }
     }
       ]
     }
```

Notes

If both match and filter conditions are set in the `filter_path` parameter, the filter condition will be applied first for the returned results and then the match condition.

Common Query Samples

Last updated：2025-01-03 10:35:42

This document describes how to use common queries and keywords in CTSDB with simple samples for curl. The metric structure used by all samples is as follows:

```
{
    "ctsdb_test" : {
      "tags" : {
        "region" : "string"
      },
      "time" : {
        "name" : "timestamp",
        "format" : "epoch_millis"
      },
      "fields" : {
        "cpuUsage" : "float",
        "diskUsage" : "string",
        "dcpuUsage" : "integer"
      },
      "options" : {
        "expire_day" : 7,
        "refresh_interval" : "10s",
        "number_of_shards" : 5
      }
    }
}
```

Combined Query

Combined queries can be used for single queries as well as composite queries. The `query` keyword in the query body can use query domain-specific language (DSL) to define query conditions.
This document describes how to construct and combine filter conditions and process the returned result set.

Common filter conditions

1. Range

`Range` indicates range query, which supports fields of `string`, `long`, `integer`, `short`, `double`, `float`, and `date` types. The parameters that can be contained in a range query are as detailed below:

Parameter	Description

gte	Greater than or equal to
gt	Greater than
lte	Less than or equal to
lt	Less than

Note:

If a range query involves time types, you can use the `format` parameter to specify the time format. For the specific time formats, see Creating Metric.

Sample code of time range query for curl:

```
    curl -u root:1e201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "query": {
            "range": {
                "timestamp": {
                    "gte": "01/01/2018",
                    "lte": "03/01/2018",
                    "format": "MM/dd/yyyy",
                    "time_zone":"+08:00"
                  }
              }
          }
     }'
```

Sample code of numeric range query for curl:

```
    curl -u root:1e201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "query": {
            "range": {
                "cpuUsage": {
                    "gte": 1.0,
                    "lte": 10.0
                }
            }
        }
    }'
```

2. Terms

The `terms` keyword can be used to match specific fields during query, and its value needs to be enclosed by brackets ([]).
Sample code for curl:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        }
    }'
```

Filter condition combination

A composite query usually uses bool keywords to combine multiple query conditions. Common combination keywords used for bool queries include `filter` (similar to `AND`), `must_not` (similar to `NOT`), and `should` (similar to `OR`). To improve the query efficiency, you must add a range query for the `time` field, which always returns a value in `epoch_millis` format no matter how the query is written. The combination of `query-bool-filter` can greatly improve the query performance, so be sure to use it.

1. Sample code of `AND` condition for curl

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
      "query": {
        "bool": {
          "filter": [
            {
              "range": {
                "timestamp": {
                  "format": "yyyy-MM-dd HH:mm:ss",
                  "gte": "2017-11-06 23:00:00",
                  "lt": "2018-03-06 23:05:00",
                  "time_zone":"+08:00"
                }

              }
            },
            {
              "terms": {
                "region": ["sh"]
              }
```

```
            },
            {
              "terms": {
                "cpuUsage": ["2.0"]
              }
            }
          ]
        }
      },
       "docvalue_fields": [
        "cpuUsage",
        "timestamp"
        ]
    }'
```

Note:

The query conditions are similar to `timestamp>='2017-11-06 23:00:00' AND timestamp<'2018-03-06 23:05:00' AND region=sh AND cpuUsage=2.0`.

2. Sample code of `OR` condition for curl

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
      "query": {
        "bool": {
          "filter": [
            {
              "range": {
                "timestamp": {
                  "format": "yyyy-MM-dd HH:mm:ss",
                  "gte": "2017-11-06 23:00:00",
                  "lt": "2018-11-06 23:05:00",
                  "time_zone":"+08:00"
                }
              }
            },
            {
              "term": {
                "region": "gz"
              }
            }
          ],
          "should": [
            {
              "terms": {
```

```
                "cpuUsage": ["2.0"]
              }
            },
            {
              "terms": {
                "cpuUsage": ["2.5"]
              }
            }
          ],
          "minimum_should_match": 1
        }
      },
      "docvalue_fields": [
        "cpuUsage",
        "timestamp"
      ]
    }'
```

Note:

The query conditions are similar to `timestamp>='2017-11-06 23:00:00' AND timestamp<'2018-11-06 23:05:00' AND region='gz' AND (cpuUsage=2.0 or cpuUsage=2.5)`. The `minimum_should_match` parameter is used to set the minimum number of results that should match `cpuUsage=2.0` and `cpuUsage=2.5`, and its default value is 0.

3. Sample code of `NOT` condition for curl

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
 172.16.345.14:9201/ctsdb_test/_search -d'
    {
      "query": {
        "bool": {
          "filter": [
                    {
            "range": {
              "timestamp": {
                "format": "yyyy-MM-dd HH:mm:ss",
                "gte": "2017-11-06 23:00:00",
                "lt": "2018-11-06 23:05:00",
                "time_zone":"+08:00"
              }

            }
          },
          {
            "terms": {
              "region": ["gz"]
```

```
                }
              }
            ],
            "must_not": [
             {
                "terms": {
                  "cpuUsage": ["2.0"]
                }
              }
            ]
          }
        },
        "docvalue_fields": [
          "cpuUsage",
          "timestamp"
            ]
        }'
```

Note:

The query conditions are similar to `timestamp>=2017-11-06 23:00:00 AND timestamp\\<2018-11-06 23:05:00 AND region='gz' AND cpuUsage !=2.0`.

Processing of returned result set

1. From/Size

You can paginate query results by setting the `from` and `size` keywords. The `from` keyword defines the offset of the first data entry in the query results, and `size` the maximum number of returned results. The default values of `from` and `size` are 0 and 10, respectively. The sum of `from` and `size` cannot exceed 65,536 by default. If you want to have more results returned, see the description of query with the `scroll` keyword in this document.

Sample code for curl:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "from": 0,
        "size": 5,
        "query": {
            "bool": {
                "filter": {
                    "range": {
                        "timestamp": {
                            "gte": "01/01/2018",
                            "lte": "03/01/2018",
                            "format": "dd/MM/yyyy",
```

```
                            "time_zone":"+08:00"
                        }
                    }
                },
                "must_not": {
                    "terms": {
                        "region": ["bj"]
                    }
                }
            }
        }
    }'
```

2. Scroll

A scroll operation is similar to a cursor in a relational database. Therefore, it is suitable for backend batch processing rather than real-time search.

You can divide a scroll operation into two phases: initialization and traversal. During initialization, all search results matching the search conditions will be cached like with a snapshot, from which data will be taken out during traversal. Note that insertion, deletion, and update of metrics after scroll initialization will not affect the traversal result.

During initialization, you can use the `size` keyword to specify the size of the returned result set, whose default value is 10 and maximum value is 65,536. You can also use the `query` and `docvalue_fields` keywords to specify the query conditions and returned fields respectively. The `_scroll_id` field will be returned during both initialization and traversal. You can specify the `_scroll_id` returned by the previous traversal for a new traversal until the returned result is empty. During initialization and traversal, you can also use the `scroll` parameter to set the context retention time of the traversal, after which the `scroll_id` will become invalid. The time format is as detailed below:

Format	Description
d	days
h	hours
m	minutes
s	seconds
ms	milliseconds
micros	microseconds
nanos	nanoseconds

Sample code for curl:

Scroll initialization:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/ctsdb_test/_search?scroll=1m -d'
    {
       "size":5,
       "query": {
       "bool": {
         "filter": [
           {
             "terms": {
               "region": ["gz"]
             }
           }
         ]
       }
     },
    "docvalue_fields": [
          "cpuUsage",
        "region",
        "timestamp"
        ]
    }'
```

Response of scroll initialization:

```
    {
      "_scroll_id":
"DnF1ZXJ5VGhlbkZldGNoAwAAAAADrOFFm5YSEhnMjdnUWNPcndHS1k5Wjc3bHcAAAAAAAz_1RZiRk
ZTcGp4dFRXR18xMGtzSmhEUFJRAAAAAAAP5vQWOXFOR29lc0hROHFWMmFGTkVmSkxmZw==",
      "took": 10641,
      "timed_out": false,
      "_shards": {
       "total": 3,
       "successful": 3,
       "skipped": 0,
       "failed": 0
      },
      "hits": {
        "total": 1592072666,
        "max_score": 0.65708643,
        "hits": [
          {
            "_index": "ctsdb_test@0_-1",
            "_type": "doc",
```

```
        "_id": "oyylNU0U65cZjByyt7sW_JmPPgAACY4Bh0",
        "_score": 0.65708643,
        "_routing": "354d14eb",
        "fields": {
          "region": [
            "gz"
          ],
          "cpuUsage": [
            "2.0"
          ],
          "timestamp": [
            1509909300000
          ]
        }
      },
      {
        "_index": "ctsdb_test@0_-1",
        "_type": "doc",
        "_id": "oyykFN0yd1d9NDPfzjRdrJ8whQAACYsBqc",
        "_score": 0.65708643,
        "_routing": "14dd3277",
        "fields": {
          "region": [
            "gz"
          ],
          "cpuUsage": [
            "1.8"
          ],
          "timestamp": [
            1509908340000
          ]
        }
      },
      {
        "_index": "ctsdb_test@0_-1",
        "_type": "doc",
        "_id": "oyylHLp9jl_3sF4N2rnh67h4SgAABHIBso",
        "_score": 0.65708643,
        "_routing": "1cba7d8e",
        "fields": {
          "region": [
            "gz"
          ],
          "cpuUsage": [
            "2.5"
          ],
          "timestamp": [
```



```
                1509909720000
              ]
            }
          },
          {
            "_index": "ctsdb_test@0_-1",
            "_type": "doc",
            "_id": "oyylH2JsOKnFHGUinQ7jM-ZwkgAAAvcBso",
            "_score": 0.65708643,
            "_routing": "1f626c38",
            "fields": {
              "region": [
                "gz"
              ],
              "cpuUsage": [
                "2.1"
              ],
              "timestamp": [
                1509909720000
              ]
            }
          },
          {
            "_index": "ctsdb_test@0_-1",
            "_type": "doc",
            "_id": "oyylHLp9jl_3sF4N2rnh67h4SgAABGsBso",
            "_score": 0.65708643,
            "_routing": "1cba7d8e",
            "fields": {
              "region": [
                "gz"
              ],
              "cpuUsage": [
                "2.0"
              ],
              "timestamp": [
                1509909720000
              ]
            }
          }
        ]
          }
    }
```


Scroll traversal:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/_search/scroll  -d'
    {
    "scroll" : "1m",
    "scroll_id" :
"DnF1ZXJ5VGhlbkZldGNoAwAAAAAADrOFFm5YSEhnMjdnUWNPcndHS1k5Wjc3bHcAAAAAAAz_1RZiRk
ZTcGp4dFRXR18xMGtzSmhEUFJRAAAAAAAP5vQWOXFOR29lc0hROHFWMmFGTkVmSkxmZw=="
    }'
```

Note:

`scroll_id` in this request is the value of `_scroll_id` returned in scroll initialization. In the next traversal, the `scroll_id` parameter value should be adjusted to the `_scroll_id` value returned in the previous traversal, that is, the `scroll_id` parameter value in each request is the `_scroll_id` value returned in the previous request, and traversal will end until the returned result is empty.

The `_scroll_id` values returned by the two traversals may be the same, so `_scroll_id` cannot be used to redirect to the specified page.

3. Sort

The `sort` keyword is mainly used to sort query results and has two orders: `asc` and `desc`. The default sorting order for custom fields in CTSDB is `asc`. Available sorting modes include `min`, `max`, `sum`, `avg`, and `median`, where `sum`, `avg`, and `median` are suitable only for fields of `array` type that store numbers only.

Sample code for curl:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "query": {
            "bool": {
                "must": {
                    "range": {
                        "timestamp": {
                            "gte": "01/01/2018",
                            "lte": "03/01/2018",
                            "format": "MM/dd/yyyy",
                            "time_zone":"+08:00"
                        }
                    }
                }
            }
        },
        "sort": [
```

```
            {
                "cpuUsage": {
                    "order": "asc",
                    "mode": "min"
                }
            },
            {
                "timestamp": {
                    "order": "asc"
                }
            },
            "diskUsage"
        ]
    }'
```

4. docvalue_fields

The `docvalue_fields` keyword specifies the names of the fields to be returned in an array.

Sample code for curl:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "docvalue_fields": ["timestamp", "cpuUsage"]
    }'
```

Aggregate Query

The `agg` keyword is mainly used to construct aggregate queries. You can get the aggregate results in the returned `aggregations` field. The returned aggregate fields are as detailed below. If you want to focus only on the aggregate results, set the `size` parameter to 0 during query.

Field	Description
hits	Matched query results. Here, the `total` field indicates the number of data records participating in aggregate. The `hits` field is an array, which contains the first 10 query results if not specified. Each result in the `hits` array contains `_index` (child metric

	involved in the query). If `docvalue_fields` is specified in the query, the `fields` field will be returned to indicate the value of each field.
took	Time in milliseconds taken by the entire query.
_shards	Number of shards involved in the query. Here, `total` indicates the total number of shards, `successful` the shards that were successfully queried, `failed` the shards that failed to be queried, and `skipped` skipped shards.
timed_out	Indicates whether query timed out. Valid values: false, true.
aggregations	Returned aggregate result.

The following lists some common aggregate modes:

Regular aggregate

For a regular aggregate, you need to specify the aggregate name, mode (common modes include `min` , `max` , `avg` , `value_count` , and `sum`), and target fields.

Sample code for curl:

```
    curl -u root:1e201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "size":0,
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs": {
            "myname": {
                "max": {
                    "field": "cpuUsage"
                }
            }
        }
    }'
```

Note:

The above sample aggregates the `cpuUsage` field in `max` mode (you can also use other modes such as `min` and `avg`), and the returned aggregate results are named the alias `myname` field (you can also specify another name).

Response:

```
{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 20,
    "successful": 20,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 7,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "myname": {
      "value": 4
    }
  }
}
```

`terms` aggregate

A `terms` aggregate is mainly used to query all the unique values and the number of such values of a field. You can specify the rule of sorting the returned unique values and the number of returned results and perform fuzzy or exact match for data fields participating in the aggregate. For more information, see the sample below. You can use the `filter_path` parameter to customize the returned result fields as instructed in Batch Querying Data.

Sample code for Curl:

Request:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search?filter_path=aggregations -d'
    {
        "aggs": {
            "myname": {
                "terms": {"field":"region"}
            }
        }
    }'
```

Response:

```
    {
      "aggregations": {
```

```
        "myname": {
          "doc_count_error_upper_bound": 0,
          "sum_other_doc_count": 0,
          "buckets": [
            {
              "key": "sh",
              "doc_count": 10
            },
            {
              "key": "Motor_sports",
              "doc_count": 6
            },
            {
              "key": "gz",
              "doc_count": 3
            },
            {
              "key": "bj",
              "doc_count": 2
            },
            {
              "key": "cd",
              "doc_count": 2
            },
            {
              "key": "Winter_sports",
              "doc_count": 1
            },
            {
              "key": "water_sports",
              "doc_count": 1
            }
          ]
        }
      }
    }
```


Note:

The above sample returns all the unique values and their numbers of occurrences in the `region` field in `ctsdb_test`. By analyzing the `buckets` field in the returned `aggregations` field, you can find that the `region` field has 7 types of values, i.e., `sh`, `Motor_sports`, `gz`, `bj`, `cd`, `Winter_sports`, and `water_sports`, and the occurrence number of each value of the returned fields is indicated in `doc_count`. You can use the `size` field to specify the number of unique values to be returned; for example, if the `region` field has 7 unique values, you can set the `size` field to 5 to return only the first 5 values. For more information, see

the sample.

Request:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search?filter_path=aggregations -d'
    {
        "aggs": {
            "myname": {
                "terms": {
                   "field":"region",
                    "size":5
                }
            }
        }
    }'
```

Response:

```
    {
      "aggregations": {
        "myname": {
          "doc_count_error_upper_bound": 0,
          "sum_other_doc_count": 2,
          "buckets": [
            {
              "key": "sh",
              "doc_count": 10
            },
            {
              "key": "Motor_sports",
              "doc_count": 6
            },
            {
              "key": "gz",
              "doc_count": 3
            },
            {
              "key": "bj",
              "doc_count": 2
            },
            {
              "key": "cd",
              "doc_count": 2
            }
          ]
        }
```

```
        }
      }
```

You can sort the returned results as instructed in the sample below:

1. Request (the returned results are sorted by number of unique values in descending order):

```
 curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search?filter_path=aggregations -d'
 {
   "aggs": {
       "myname": {
           "terms": {
             "field":"region",
             "order":{"_count":"desc"}
           }
       }
   }
 }'
```

Response:

```
    {
       "aggregations": {
         "myname": {
           "doc_count_error_upper_bound": 0,
           "sum_other_doc_count": 0,
           "buckets": [
             {
               "key": "sh",
               "doc_count": 10
             },
             {
               "key": "Motor_sports",
               "doc_count": 6
             },
             {
               "key": "gz",
               "doc_count": 3
             },
             {
               "key": "bj",
               "doc_count": 2
             },
             {
               "key": "cd",
```

```
              "doc_count": 2
            },
            {
              "key": "Winter_sports",
              "doc_count": 1
            },
            {
              "key": "water_sports",
              "doc_count": 1
            }
          ]
        }
      }
    }
```

2. Request (the returned results are sorted alphabetically in ascending order):

```
 curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search?filter_path=aggregations -d'
 {
   "aggs": {
       "myname": {
           "terms": {
             "field":"region",
             "order":{"_term":"asc"}
           }
       }
   }
 }'
```

Response:

```
    {
      "aggregations": {
        "myname": {
          "doc_count_error_upper_bound": 0,
          "sum_other_doc_count": 0,
          "buckets": [
            {
              "key": "Motor_sports",
              "doc_count": 6
            },
            {
              "key": "Winter_sports",
              "doc_count": 1
            },
```

```
            {
              "key": "bj",
              "doc_count": 2
            },
            {
              "key": "cd",
              "doc_count": 2
            },
            {
              "key": "gz",
              "doc_count": 3
            },
            {
              "key": "sh",
              "doc_count": 10
            },
            {
              "key": "water_sports",
              "doc_count": 1
            }
          ]
        }
      }
    }
```

You can set a regular expression to fuzzily match fields in the `terms` aggregate or exactly match specified fields as instructed in the sample below:

Sample code of fuzzy match:

Request (only the `region` fields whose values contain `sport` and don't start with `water_` are aggregated and returned):

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search?filter_path=aggregations -d'
    {
      "aggs": {
          "myname": {
              "terms": {
                "field":"region",
                "include" : ".*sport.*",
                "exclude" : "water_.*"
              }
          }
      }
    }'
```

Response:

```
    {
      "aggregations": {
        "myname": {
          "doc_count_error_upper_bound": 0,
          "sum_other_doc_count": 0,
          "buckets": [
            {
              "key": "Motor_sports",
              "doc_count": 6
            },
            {
              "key": "Winter_sports",
              "doc_count": 1
            }
          ]
        }
      }
    }
```

Sample code of exact match:

Request (the `region` fields whose values are `sh`, `bj`, `cd`, and `gz` are aggregated in `region_zone`, and the `region` fields whose values are not `sh`, `bj`, `cd`, and `gz` are aggregated in `region_sports`):

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search?filter_path=aggregations -d'
    {
    "aggs" : {
      "region_zone" : {
        "terms" : {
          "field" : "region",
          "include" : ["sh", "bj","cd","gz"]
          }
        },
        "region_sports" : {
         "terms" : {
             "field" : "region",
             "exclude" : ["sh", "bj","cd","gz"]
         }
        }
      }
    }'
```

Response:

```
{
  "aggregations": {
    "region_sport": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        },
        {
          "key": "water_sports",
          "doc_count": 1
        }
      ]
    },
    "region_zone": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "sh",
          "doc_count": 10
        },
        {
          "key": "gz",
          "doc_count": 3
        },
        {
          "key": "bj",
          "doc_count": 2
        },
        {
          "key": "cd",
          "doc_count": 2
        }
      ]
    }
  }
}
```


Date histogram aggregate

A date histogram is mainly used to aggregate dates into a histogram.

Sample code for curl:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs": {
            "time_1h_agg": {
                "date_histogram": {
                    "field": "timestamp",
                    "interval": "1h"
                },
                "aggs": {
                    "avgCpuUsage": {
                        "avg": {
                            "field": "cpuUsage"
                        }
                    }
                }
            }
        }
    }'
```

Response:

```
    {
      "took": 5,
      "timed_out": false,
      "_shards": {
        "total": 20,
        "successful": 20,
        "skipped": 0,
        "failed": 0
      },
      "hits": {
        "total": 6,
        "max_score": 0.074107975,
        "hits": [
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2QtGR5xcjRaw2ETf-",
```

```
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2QtGR5xcjRaw2ETf_",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2Q5Xr5xcjRaw2ETgA",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2Q5Xr5xcjRaw2ETgB",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2RGfF5xcjRaw2ETgC",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2RGfF5xcjRaw2ETgD",
            "_score": 0.074107975,
            "_routing": "sh"
          }
        ]
      },
      "aggregations": {
        "time_1h_agg": {
          "buckets": [
            {
              "key_as_string": "1520222400",
              "key": 1520222400000,
              "doc_count": 1,
```



```
        "avgCpuUsage": {
          "value": 2.5
        }
      },
      {
        "key_as_string": "1520226000",
        "key": 1520226000000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520229600",
        "key": 1520229600000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520233200",
        "key": 1520233200000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520236800",
        "key": 1520236800000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520240400",
        "key": 1520240400000,
        "doc_count": 0,
        "avgCpuUsage": {
          "value": null
        }
      },
      {
        "key_as_string": "1520244000",
        "key": 1520244000000,
```

```
            "doc_count": 0,
            "avgCpuUsage": {
              "value": null
            }
          },
          {
            "key_as_string": "1520247600",
            "key": 1520247600000,
            "doc_count": 1,
            "avgCpuUsage": {
              "value": 2
            }
          },
          {
            "key_as_string": "1520251200",
            "key": 1520251200000,
            "doc_count": 4,
            "avgCpuUsage": {
              "value": 2.25
            }
          }
        ]
      }
    }
  }
```

Note:

The above sample aggregates the `cpuUsage` field in `date_histogram` mode with a granularity of one hour. The total aggregate name of the returned results is `time_1h_agg` (you can specify another name), and the aggregate name in each time interval is `avgCpuUsage` (you can specify another name). The valid time granularities for `interval` include `year`, `quarter`, `month`, `week`, `day`, `hour`, `minute`, and `second`. You can also represent the time granularity as a time unit; for example, `1y` represents one year, and `1h` one hour. The system does not support decimal time units; therefore, you need to convert `1.5h` to `90min` for example.

Percentiles aggregate

You can specify the percentile in percentiles aggregate. The system default percentiles are 1, 5, 25, 50, 75, 95, and 99. You can select other values as needed.

Sample code for curl:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "query": {
```

```
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs":
        {
            "myname":
            {
                "percentiles":
                {
                    "field": "cpuUsage",
                    "percents": [1,25,50,70,99]
                }
            }
        }
    }'
```

Response:

```
    {
      "took": 18,
      "timed_out": false,
      "_shards": {
        "total": 20,
        "successful": 20,
        "skipped": 0,
        "failed": 0
      },
      "hits": {
        "total": 6,
        "max_score": 0.074107975,
        "hits": [
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2QtGR5xcjRaw2ETf-",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2QtGR5xcjRaw2ETf_",
            "_score": 0.074107975,
            "_routing": "sh"
          },
```

```
            {
              "_index": "ctsdb_test@1520092800000_3",
              "_type": "doc",
              "_id": "AWH2Q5Xr5xcjRaw2ETgA",
              "_score": 0.074107975,
              "_routing": "sh"
            },
            {
              "_index": "ctsdb_test@1520092800000_3",
              "_type": "doc",
              "_id": "AWH2Q5Xr5xcjRaw2ETgB",
              "_score": 0.074107975,
              "_routing": "sh"
            },
            {
              "_index": "ctsdb_test@1520092800000_3",
              "_type": "doc",
              "_id": "AWH2RGfF5xcjRaw2ETgC",
              "_score": 0.074107975,
              "_routing": "sh"
            },
            {
              "_index": "ctsdb_test@1520092800000_3",
              "_type": "doc",
              "_id": "AWH2RGfF5xcjRaw2ETgD",
              "_score": 0.074107975,
              "_routing": "sh"
            }
          ]
        },
        "aggregations": {
          "myname": {
            "values": {
              "1.0": 2,
              "25.0": 2,
              "50.0": 2.25,
              "70.0": 2.5,
              "99.0": 2.5
            }
          }
        }
      }
```


Note:

The above sample aggregates the `cpuUsage` field in `percentiles` mode, and the selected percentiles are 1, 25, 50, 70, and 99. The returned aggregate results are named the alias `myname` (you can also specify another

name).

Cardinality aggregate

A cardinality aggregate is mainly used to get the number of deduplicated results. By default, if the number of aggregate results is less than or equal to 3,000, the result returned by `cardinality` will be precise; otherwise, the result will be approximate.

Sample code for curl:

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET
172.16.345.14:9201/ctsdb_test/_search -d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs":
        {
            "myname":
            {
                "cardinality":
                {
                    "field": "cpuUsage"
                }
            }
        }
    }'
```

Response:

```
    {
      "took": 15,
      "timed_out": false,
      "_shards": {
        "total": 20,
        "successful": 20,
        "skipped": 0,
        "failed": 0
      },
      "hits": {
        "total": 6,
        "max_score": 0.074107975,
        "hits": [
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
```

```
            "_id": "AWH2QtGR5xcjRaw2ETf-",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2QtGR5xcjRaw2ETf_",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2Q5Xr5xcjRaw2ETgA",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2Q5Xr5xcjRaw2ETgB",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2RGfF5xcjRaw2ETgC",
            "_score": 0.074107975,
            "_routing": "sh"
          },
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2RGfF5xcjRaw2ETgD",
            "_score": 0.074107975,
            "_routing": "sh"
          }
        ]
      },
      "aggregations": {
        "myname": {
          "value": 2
        }
      }
    }
```

Note:

The above sample aggregates the `cpuUsage` field in `cardinality` mode, and the returned aggregate result is named the alias `myname` (you can also specify another name).

Bulk Writing Data

Last updated：2025-01-03 11:30:40

Bulk Writing Data into Single Metric

This API can be used to write data records into a single metric either in bulk or not. To improve the write efficiency, we recommend you write data in bulk.

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Request path: `/${metric_name}/doc/_bulk`, where `${metric_name}` is the metric name.
Method: POST
Note:
The `doc` keyword is the `_type` of the written data and must be added to facilitate subsequent system parsing and upgrade.

Request parameters

You can set the `filter_path` parameter to filter and simplify the returned results. For more information, please see Batch Querying Data.

Request content

You need to bulk write data as structured data in NDJSON format into a metric, which is similar to the following:

```
Metadata\\n
Data to be written\\n
....
Metadata\\n
Data to be written\\n
```

The format of the metadata is as shown below:

```
{
    "index" :
    {
        "_id" : "1",                        # Document ID (optional)
        "_routing": "sh"                        # Routing value (optional)
    }
```

```
}
```

The format of the written data is as shown below:

```
{
    "field1" : "value1",
    "field2" : "value2"
}
```

Note:

When writing data, you can set the `_routing` parameter to specify the shards where data to be written. This parameter is optional and can be specified to any value. You can specify the same `_routing` value for different data records to route them into the same shard. In addition, if you specify an already set `_routing` parameter value during a query, the system will query data in the specified shard, which can greatly accelerate the query. You need to add a line break at the end of the request body.

Response content

You should note that the returned result of the bulk data write API is different from those of other APIs. Please pay attention to the `errors` (not `error`) field in the JSON result first. If it is `false`, all data records were successfully written; if it is `true`, some data records failed to be written, and you can get the failure details from the `items` field.

The `items` field is an array, where each element corresponds to a write request. You can check whether each element has the `error` field to judge whether the corresponding request is successful. If the `error` field exists, the request failed. The specific error information is in the `error` field. If the `error` field doesn't exist, the request succeeded.

Sample code for curl

Sample response upon success:

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/ctsdb_test/doc/_bulk -d'
{"index":{"_routing": "sh" }}
{"region":"sh","cpuUsage":2.5,"timestamp":1505294654}
{"index":{"_routing": "sh" }}
{"region":"sh","cpuUsage":2.0,"timestamp":1505294654}
'
```

Response:

```
{
    "took": 134,
    "errors": false,
```

```
    "items":
    [
        {
            "index":
            {
                "_index": "ctsdb_test@1505232000000_1",
                "_type": "doc",
                "_id": "AV_8eeo_UAkC9PF9L-2q",
                "_version": 1,
                "result": "created",
                "_shards":
                 {
                    "total": 2,
                    "successful": 2,
                    "failed": 0
                },
                "created": true,
                "status": 201
                }
            },
            {
            "index":
            {
                "_index": "ctsdb_test2@1505232000000_1",
                "_type": "doc",
                "_id": "AV_8eeo_UAkC9PF9L-2r",
                "_version": 1,
                "result": "created",
                "_shards":
                {
                    "total": 2,
                    "successful": 2,
                    "failed": 0
                },
                "created": true,
                "status": 201
            }
        }
    ]
}
```

Note:

The `errors` value returned above is `false`, indicating that all data records were successfully written. The `items` array indicates the write result of each record in the same order as in the `bulk` request. For single records in `items`, the `status` of `2XX` indicates that the record was successfully written. `_index` indicates the child metric where the data was written, and `_shards` indicates the replica write status. In the above

sample, `total` indicates two replicas, and `successful` indicates that data was successfully written into both replicas.

Sample response upon failure:

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/hcbs_client_trace/doc/_bulk -d'
{"index":{"_type":"type"}}
{"vol_id":"c57e008c-0ae0-41cd-8da8-
6989d0522fc6","io_type":2,"data_len":4096,"latency":3,"try_times":1,"errcode":0
,"start_time":1503404266}
{"index":{"_type":"type"}}
{"vol_id":"c57e008c-0ae0-41cd-8da8-
6989d0522fc6","io_type":"abc","data_len":4096,"latency":1,"try_times":1,"errcod
e":0,"start_time":1503404266}
'
```

Response:

```
{
  "took": 7,
  "errors": true,
  "items": [
    {
      "index": {
        "_index": "hcbs_client_trace",
        "_type": "type",
        "_id": "AWMe9r9lNifptzIWMVPT",
        "_version": 1,
        "result": "created",
        "_shards": {
          "total": 2,
          "successful": 2,
          "failed": 0
        },
        "created": true,
        "status": 201
      }
    },
    {
      "index": {
        "_index": "hcbs_client_trace",
        "_type": "type",
        "_id": "AWMe9r9lNifptzIWMVPU",
        "status": 400,
        "error": {
```

```
            "type": "mapper_parsing_exception",
            "reason": "failed to parse [io_type]",
            "caused_by": {
              "type": "number_format_exception",
              "reason": "For input string: \\"abc\\""
            }
          }
        }
      }
    ]
  }
```

Note:

The `errors` value returned above is `true`, indicating that some data records failed to be written. The `items` array indicates the write result of each record in the same order as in the `bulk` request. For single records in `items`, the `status` of `2XX` indicates that the record was successfully written. The `error` field indicates the detailed error information. `_index` indicates the child metric where the data was written, and `_shards` indicates the replica write status. In the above sample, `total` indicates two replicas, and `successful` indicates that data was successfully written into both replicas.

Bulk Writing Data into Multiple Metrics

This API can be used to write data records into multiple metrics either in bulk or not. To improve the write efficiency, we recommend you write data in bulk.

Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request path and method

Request path: `_bulk`

Method: PUT

Note:

The `doc` keyword is the `_type` of the written data and must be added to facilitate subsequent system parsing and upgrade.

Request parameters

You can set the `filter_path` parameter to filter and simplify the returned results. For more information, please see Batch Querying Data.

Request content

You need to bulk write data as structured data in NDJSON format into a metric, which is similar to the following:

```
Metadata\\n
Data to be written\\n
....
Metadata\\n
Data to be written\\n
```

The format of the metadata is as shown below:

```
{
    "index" :
    {
        "_index" :"metric_name",        # Metric where data is to be written
        "_type" : "doc",                        # Document type of the data to
be written
        "_id" : "1",                       # Document ID (optional)
        "_routing": "sh"                        # Routing value (optional)
    }
}
```

The format of the written data is as shown below:

```
 {
    "field1" : "value1",
    "field2" : "value2"
}
```

Note:

When writing data, you can set the `_routing` parameter to specify the shards where data to be written. This parameter is optional and can be specified to any value. You can specify the same `_routing` value for different data records to route them into the same shard. In addition, if you specify an already set `_routing` parameter value during a query, the system will query data in the specified shard, which can greatly accelerate the query. Note that you need to add a line break at the end of the request body.

Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. The specific error information is in the `error` field. Note: if the request succeeded but the `errors` (not `error`) field is not `false`, the specific data that failed to be written is indicated in the `errors` field.

Sample code for curl

Request:


```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.16.345.14:9201/_bulk -d'
{"index":{"_index" : "ctsdb_test", "_type" : "doc", "_routing": "sh" }}
{"region":"sh","cpuUsage":2.5,"timestamp":1505294654}
{"index":{"_index" : "ctsdb_test2", "_type" : "doc", "_routing": "sh" }}
{"region":"sh","cpuUsage":2.0,"timestamp":1505294654}
'
```

Response:

```
{
    "took": 134,
    "errors": false,
    "items":
    [
        {
            "index":
            {
                "_index": "ctsdb_test@1505232000000_1",
                "_type": "doc",
                "_id": "AV_8eeo_UAkC9PF9L-2q",
                "_version": 1,
                "result": "created",
                "_shards":
                 {
                    "total": 2,
                    "successful": 2,
                    "failed": 0
                },
                "created": true,
                "status": 201
                }
            },
            {
            "index":
            {
                "_index": "ctsdb_test2@1505232000000_1",
                "_type": "doc",
                "_id": "AV_8eeo_UAkC9PF9L-2r",
                "_version": 1,
                "result": "created",
                "_shards":
                {
                    "total": 2,
                    "successful": 2,
                    "failed": 0
```

```
                },
                "created": true,
                "status": 201
            }
        }
    ]
}
```

Note:

The `errors` value returned above is `false`, indicating that all data records were successfully written. The `items` array indicates the write result of each record in the same order as in the `bulk` request. For single records in `items`, the `status` of `2XX` indicates that the record was successfully written. `_index` indicates the child metric where the data was written, and `_shards` indicates the replica write status. In the above sample, `total` indicates two replicas, and `successful` indicates that data was successfully written into both replicas.

Deleting Data

Last updated：2025-01-03 10:32:49

Request Address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

Request Path and Method

Request path: `/${metric_name}/_delete_by_query`, where `${metric_name}` is the metric name.
Method: POST

Request Parameters

None

Request Content

Query conditions when a metric is deleted. For more information, please see the sample.

Response Content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. The specific error information is in the `error` field.

Sample Code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/ctsdb_test/_delete_by_query -d'
{
"query": {
```

```
    "bool": {
        "filter": {
            "match_all": {}
            }
        }
    }
}'
```

Response:

```
{
    "took": 43,
    "timed_out": false,
    "total": 1,
    "deleted": 1,
    "batches": 1,
    "version_conflicts": 0,
    "noops": 0,
    "retries": {
        "bulk": 0,
        "search": 0
    }
}
```

Modifying Data

Last updated：2025-01-03 10:32:19

In CTSDB, you can modify data by deleting it first and then writing it again. For directions on data deletion and write, please see Deleting Data and Bulk Writing Data.

Rollup Operations

Last updated：2025-01-03 10:31:22

Creating Rollup Task

In scenarios with massive amounts of data, a business system can generate petabytes of data per day or even per hour. Time series data is generally massive in amount, time-sensitive, and trending. Therefore, systems using such data (e.g., monitoring or data analysis systems) often only need high-precision data in the most recent time period while downsampling historical data for storage. You can configure a rollup task to periodically aggregate historical data and save it into a new metric. A rollup task not only saves downsampled historical data, but also improves query performance and reduces storage costs. Note that a rollup task automatically creates child metrics inheriting all configurations of the parent metric according to `base_metric`, and the parent metric configurations will be overwritten if `options` is specified.

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

2. Request path and method

Path: `/_rollup/${rollup_task_name}`, where `${rollup_task_name}` is the rollup task name.
Method: PUT

3. Request parameters

None

4. Request content

Parameter	Required	Type	Description
base_metric	Yes	string	Name of the metric (parent metric) depended on by the rollup task
rollup_metric	Yes	string	Name of the metric (child metric) generated by the rollup task
base_rollup	No	string	Rollup task depended on by the current rollup task. Before executing the current task, the system will check whether the dependent task in the corresponding time period has been completed
query	No	string	Query condition for data filtering, which consist of many elements and operations, such as `name:host AND type:max OR region:gz`
group_by	Yes	Array	Tag to be aggregated. Multiple tags can be included

function	Yes	Map	Aggregate name, mode, and fields. The fields can be only from the `fields` field in `base_metric`. If the `fields` field is empty, the rollup cannot be configured. The function can be `sum`, `avg`, `min`, `max`, `set`, `any`, `first`, `last`, `percentiles`, etc., such as `{"cost_total":{"sum":{"field":"cost"}},"cpu_usage_avg":{ "avg": { "field":"cpu_usage"}}}`
interval	Yes	string	Aggregate granularity, such as 1s, 5m, 1h, and 1d
frequency	No	string	Scheduling frequency, such as 5m, 1h, and 1d, which is the same as `interval` by default
delay	No	string	Execution delay. Generally, there should a certain delay for data write, such as 5m or 1h, in order to avoid data loss
start_time	No	string	Start time of periodic execution of the rollup task, which is the current time by default
end_time	No	string	Scheduling end time, which is the maximum timestamp value by default
options	No	map	`rollup_metric` options, which are the same as the options for metric creation

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

6. Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.16.345.14:9201/_rollup/ctsdb_rollup_task_test -d'
{
    "base_metric": %{base_metric_name},
    "rollup_metric": %{rollup_metric_name},
    "base_rollup": %{base_rollup_name},
    "query" : "name:host AND type:max",
    "group_by": ["host"],
    "function": {
        "cost_total":  {
            "sum": {
                "field": "cost"
            }
```

```
        },
        "cpu_usage_avg": {
            "avg": {
                "field": "cpu_usage"
            }
        },
        "value": {
          "percentiles": {
            "field": "value",
            "percents": [
              95
            ]
          }
        },
        "metricName": {
          "set": {
            "value": "cpu_usage"
          }
        },
        "appid": {
          "any": {
            "field": "appid"
          }
        },
         "first_value": {
          "first": {
            "field": "value"
          }
        },
         "last_value": {
          "last": {
            "field": "value"
          }
        }
    },
    "interval": "1m",
    "frequency": "5m",
    "delay": "1m",
    "start_time": "1502892000",
    "end_time": "2147483647",
    "options": {
        "expire_day": 365
    }
}
'
```

Response:

```
{
    "acknowledged": true,
    "message": "create rollup success"
}
```

Getting Names of All Rollup Tasks

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200`, which can be obtained in the console.

2. Request path and method

Path: `/_rollups`

Method: GET

3. Request parameters

None

4. Request content

None

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

6. Sample code for curl

Request:

`curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_rollups`

Response:

```
{
    "result":
    {
        "rollups":
        [
            "rollup_jgq_6",
            "rollup_jgq_60"
```

```
            ]
        },
        "status": 200
    }
```

Getting Rollup Task Details

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200` .

2. Request path and method

Path: `/_rollup/${rollup_task_name}` , where `${rollup_task_name}` is the rollup task name.

Method: GET

3. Request parameters

You can specify the `v` parameter to view the specific rollup progress. `@last_end_time` in the response structure is the latest rollup progress.

4. Request content

None

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

6. Sample code for curl

Request:

`curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_rollup/rollup_jgq_6?v`

Response:

```
{
  "result": {
    "rollup_jgq_6": {
    "base_metric": "cvm_device-300",
       "rollup_metric": "cvm_device-86400",
       "query": "metricName:cpu_usage AND statType:max",
       "group_by": [
         "vm_uuid"
```

```
        ],
        "function": {
          "value": {
            "percentiles": {
              "field": "value",
              "percents": [
                95
              ]
            }
          },
          "metricName": {
            "set": {
              "value": "cpu_usage"
            }
          },
          "appid": {
            "any": {
              "field": "appid"
            }
          }
        },
        "interval": "1d",
        "delay": "5m",
        "options": {
          "expire_day": 186
        },
        "frequency": "1d",
        "start_time": 1534003200,
        "end_time": 2147483647,
        "@state": "running",              // Running status
        "@timestamp": 1550766085000,      // Rollup task information update time
point
        "@last_end_time": 1550764800      // Rollup task end time point for
proper execution

    }
  },
  "status": 200
}
```

Deleting Rollup Task

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200` .

2. Request path and method

Path: `/_rollup/${rollup_task_name}` , where `${rollup_task_name}` is the rollup task name.

Method: DELETE

3. Request parameters

None

4. Request content

None

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

6. Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X DELETE 172.16.345.14:9201/_rollup/ctsdb_rollup_task_test
```

Response:

```
{
    "acknowledged": true,
    "message": "delete rollup success"
}
```

Updating Rollup Task

1. Request address

The address is the instance IP and port, such as `10.13.20.15:9200` .

2. Request path and method

Path: `/_rollup/${rollup_task_name}/update` , where `${rollup_task_name}` is the rollup task name.

Method: POST

3. Request parameters

None

4. Request content

Parameter	Required	Type	Description
state	Yes	string	Valid values: running, pause
start_time	No	string	Start time of periodic execution of the rollup task, which is the current time by default
end_time	No	string	Scheduling end time, which is the maximum timestamp value by default
options	No	map	Aggregate options, which are the same as the options for metric creation

5. Response content

You need to judge whether a request is successful based on the `error` field. If the response content contains the `error` field, the request failed. For the error details, please see the `error` field description.

6. Sample code for curl

Request:

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/_rollup/ctsdb_rollup_task_test/update -d'
{
    "state":"running",
    "start_time": "1511918989",
    "end_time": "1512019765",
    "options":
    {
        "expire_day": 365
    }
}'
```

Response:

```
{
    "acknowledged": true,
    "message": "update rollup success"
}
```