Elasticsearch Service

FAQs

Product Documentation

Contents

FAQs
Product

Last updated：2020-11-10 16:24:29

What is Elasticsearch?

Elasticsearch is a distributed, scalable, and real-time search and data analysis engine based on Apache Lucene™, a leading full-text search engine library. It supports arbitrary expansion from a single node to hundreds of nodes and enables storage and query of petabytes of structured or unstructured data.

What business scenarios is ES suitable for?

ES fully retains the features of Elasticsearch in terms of massive data search such as full-text search, quasi-real-time search, and structured search. It is widely used in business scenarios like log analysis and in-site search. It enables you to build log service systems for other Tencent Cloud services in use such as CVM and TKE and can also integrate search services into your existing business service frameworks.

What is Elasticsearch Service?

Tencent Cloud ES is a cloud-based cluster service built on the open-source search engine Elasticsearch. It inherits the openness, compatibility, and usability of Elasticsearch and provides a sufficiency of hardware resources, user-friendly graphical creation and management tools, and comprehensive technical and OPS support.

I have an unpaid switch order. Will the order still be valid if I upgrade the cluster configuration?

No. A new purchase order will be generated when the billing mode is switched from pay-as-you-go to monthly subscription, but if you change the configuration of the cluster before paying the order, the amount of the new purchase order will not match the cluster, and the unpaid order cannot be paid.
If you need to switch the billing mode of the cluster, cancel the unpaid order in the Order Center before proceeding with the switch.

Can I change the cloud disk type after a successful purchase?

Currently, switching types of cloud disks is not supported. You can create a snapshot for backup and then use the snapshot to create a cloud disk of your desired type.

How do I choose the appropriate node and disk when creating a cluster?

To evaluate the node specification of the ES service, please see Evaluation of Cluster Specification and Capacity Configuration.

ES Cluster

Cluster Exceptions

Overview

Last updated：2021-08-11 11:17:23

If you encounter errors or problems when using ES, you can find solutions according to the problem types in this section.

Exceptional Cluster Health Status (Red and Yellow)

Cluster Circuit Breaking

Bulk Rejection/Search Rejection

High Cluster CPU Utilization

High Cluster Disk Utilization and read_only Status

Uneven Cluster Load

Exceptional Cluster Health Status (Red and Yellow)

Last updated：2024-11-29 22:01:51

Why is the cluster in an exceptional status?

In the following conditions, the cluster will be in the red or yellow status:

If the cluster has any unassigned **primary index shard**, the cluster status will become red, which affects index reads/writes and thus requires special attention.
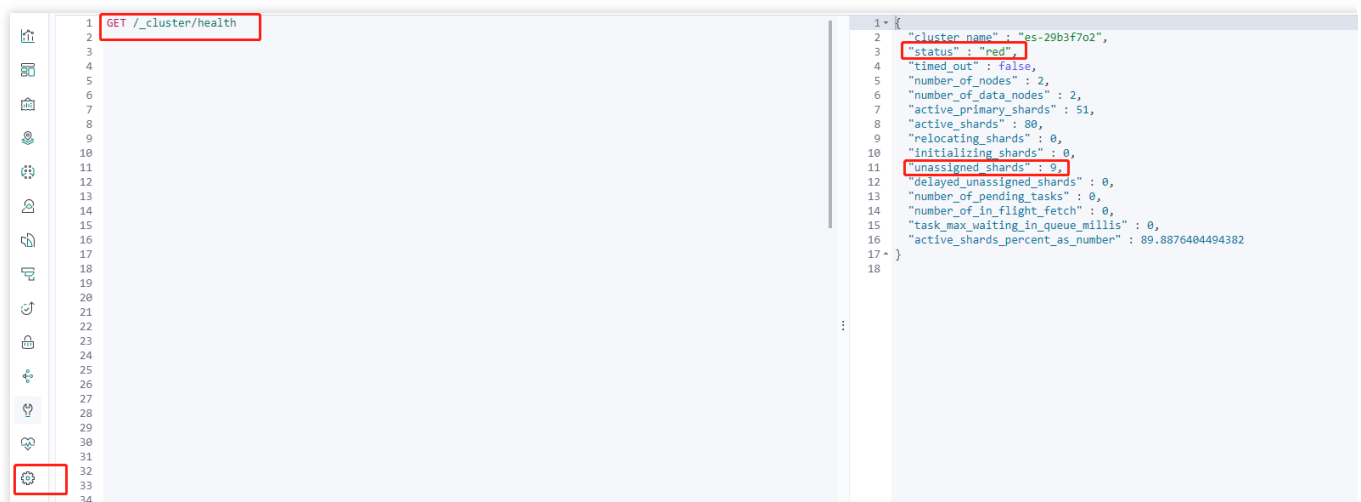
If all primary index shards in the cluster have been assigned, but there are still unassigned **replica index shards**, the cluster status will become yellow, which does not affect index reads/writes and generally can be automatically recovered.

Viewing Cluster Status

You can use Kibana Dev Tools to view the cluster status:

```
GET /_cluster/health
```

Here, you can see that the current cluster status is red, and there are nine unassigned shards.

Official descriptions of the Elasticsearch health API responses:

Metric	Description

cluster_name	Cluster name
status	Health status of the cluster, based on the state of its primary and replica shards. Statuses are: – green: all shards are assigned </br>– yellow: all primary shards are assigned, but one or more replica shards are unassigned. If a node in the cluster fails, some data may be unavailable until that node is repaired </br>– red: one or more primary shards are unassigned, so some data is unavailable. This can occur briefly during cluster startup as primary shards are assigned
timed_out	If `false`, the response is returned within the period of time that is specified by the `timeout` parameter (30s by default)
number_of_nodes	Number of nodes in the cluster
number_of_data_nodes	Number of nodes that are dedicated data nodes
active_primary_shards	Number of active primary shards
active_shards	Total number of active primary and replica shards
relocating_shards	Number of shards that are under relocation
initializing_shards	Number of shards that are under initialization
unassigned_shards	Number of unassigned shards
delayed_unassigned_shards	Number of shards whose assignment has been delayed by the timeout settings
number_of_pending_tasks	Number of cluster-level changes that have not yet been executed
number_of_in_flight_fetch	Number of unfinished fetches
task_max_waiting_in_queue_millis	Time expressed in milliseconds since the earliest initiated task is waiting for being performed
active_shards_percent_as_number	Ratio of active shards in the cluster expressed as a percentage

Troubleshooting

If a cluster is exceptional, you need to pay attention to the shards that are not assigned properly in `unassigned_shards`. The following is an example:

Finding exceptional index

View the index status and find the exceptional index based on the response.

```
GET /_cat/indices
```


Viewing exception details

```
GET /_cluster/allocation/explain
```


Through the exception information, you can see that:

1. The primary shard is currently in the unassigned status (`current_state`). This problem occurs because the node to which the shard was assigned left the cluster (`unassigned_info.reason`).

2. After the above problem occurs, the shard cannot be automatically assigned because there are no available replicas of the shard in the cluster (`can_allocate`).

3. In addition, more detailed information is provided (`allocate_explanation`).

This problem occurs because the cluster has an offline node, so the primary shard has no available shard data. Currently, the only thing you can do is to wait for the node to recover and join the cluster again.

Note:

In some extreme cases (for example, a shard in a single-replica cluster is corrupted, or the file system fails, causing the node to be removed permanently), you can only accept the fact of data loss and use the reroute command to assign an empty primary shard again. To avoid such cases as much as possible, we recommend you appropriately design index shards and refrain from setting a single replica for the index (single replica is also called zero replica, which means that an index has a primary shard but no replica shards). With the appropriate design of index shards,

you can control the total number of shards in the cluster at a healthy scale, make better use of the distributed cluster characteristics while ensuring high availability, and improve the overall cluster performance.

All possible reasons of unassigned shards (`unassigned_info.reason`)

You can use the following analysis methods to preliminarily figure out the reason why there is an unassigned shard in the cluster. Generally, you can find the reason through the `allocation explain` API.

Note:

If the cluster status hasn't automatically recovered after a long period of time, or you cannot fix the problem, please submit a ticket for assistance.

Reason	Description
INDEX_CREATED	Unassigned as a result of an API creation of an index
CLUSTER_RECOVERED	Unassigned as a result of a full cluster recovery
INDEX_REOPENED	Unassigned as a result of opening a closed index
DANGLING_INDEX_IMPORTED	Unassigned as a result of importing a dangling index
NEW_INDEX_RESTORED	Unassigned as a result of restoring into a new index
EXISTING_INDEX_RESTORED	Unassigned as a result of restoring into a closed index
REPLICA_ADDED	Unassigned as a result of explicit addition of a replica
ALLOCATION_FAILED	Unassigned as a result of a failed allocation of the shard
NODE_LEFT	Unassigned as a result of the node hosting it leaving the cluster
REROUTE_CANCELLED	Unassigned as a result of explicit cancel reroute command
REINITIALIZED	When a shard moves from started back to initializing
REALLOCATED_REPLICA	A better replica location is identified and causes the existing replica allocation to be canceled

Cluster Circuit Breaking

Last updated：2021-08-11 11:17:23

Circuit Breaking Overview

ES provides a variety of official circuit breakers to prevent ES cluster errors caused by OutOfMemoryError when the memory utilization is too high. It is equipped with various types of child circuit breakers to specify the limit on memory available to specific requests. In addition, there is a parent circuit breaker that is used to specify the total amount of memory available across all child circuit breakers.

Note:

Circuit breaking indicates that the current node's JVM utilization is too high, so the circuit breaker is triggered to prevent OOM. At this point, you can reduce the load of the node by appropriately reducing reads/writes, freeing up the memory, etc. You can also increase the size of the JVM by upgrading the node's memory specification.

Common Memory Cleanup Methods

Clear field data cache: when you perform aggregation and sorting operations on fields of `text` type, the data structure called `fielddata` will be used, which may take up a lot of memory. You can view the memory utilization of field data in an index by running the following command in the **Dev Tools** on the Kibana page:

```
GET /_cat/indices?v&h=index,fielddata.memory_size&s=fielddata.memory_size:desc
```

If field data takes up a lot of memory, you can clear it by running the following command in the **Dev Tools** on the Kibana page:

```
POST /${Indexes whose fielddata takes up a lot of memory}/_cache/clear?
fielddata=true
```

Clear segments: the FST structure of each segment will be loaded into the memory, which will not be recycled by GC. Therefore, a large number of segments in an index will also result in a high memory utilization. You can view the number of segments per node and the amount of memory used by them by running the following command in the **Dev Tools** on the Kibana page:

```
GET /_cat/nodes?v&h=segments.count,segments.memory&s=segments.memory:desc
```

If segments take up a too large amount of memory, you can delete indexes that are not used, close indexes, or regularly merge indexes that are no longer updated.

Scale out the cluster: if the circuit breaker is still triggered frequently after you clean up the memory, your cluster size may be no longer suitable for your business, and we recommend you scale it out. For more information, please see Adjusting Configuration.

Circuit Breaker List

Elasticsearch's official circuit breakers

Parent circuit breaker

The parent circuit breaker limits the total amount of memory used by all child circuit breakers. When it is triggered, the possible log information is as follows:

```
Caused by: org.elasticsearch.common.breaker.CircuitBreakingException: [parent]
Data too large, data for [<transport_request>] would be [1749436147/1.6gb],
which is larger than the limit of [1622605824/1.5gb], real usage:
[1749435872/1.6gb], new bytes reserved: [275/275b]
```

Field data breaker

When you perform aggregation and sorting operations on fields of `text` type, the "fielddata" data structure will be generated. The field data breaker estimates how much data is loaded into the memory. When the memory usage by the estimated data reaches its threshold, it will be triggered. The possible log information is as follows:

```
org.elasticsearch.common.breaker.CircuitBreakingException: [fielddata] Data too
large, data for [_id] would be [943928680/900.2mb], which is larger than the
limit of [255606128/243.7mb]
```

In flight requests circuit breaker

The in flight requests circuit breaker limits the memory usage of all currently active incoming requests at the transport or HTTP level. When it is triggered, the possible log information at this time is as follows:

```
[o.e.x.m.e.l.LocalExporter] [1611816935001404932] unexpected error while
indexing monitoring document
org.elasticsearch.xpack.monitoring.exporter.ExportException:
RemoteTransportException[[1611816935001404732][9.10.153.16:9300]
[indices:data/write/bulk[s]]]; nested:
CircuitBreakingException[[in_flight_requests] Data too large, data for
[<transport_request>] would be [19491363612/18.1gb], which is larger than the
limit of [17066491904/15.8gb]];
```

ES' proprietary circuit breaker

One of the drawbacks of Elasticsearch's official circuit breaking mechanism is that only those requests that are often problematic are tracked to estimate the memory utilization, making it impossible to limit the amount of memory

available to requests or trigger a circuit breaker based on the actual memory utilization on the current node. ES has a proprietary circuit breaker to address this problem with JVM Old memory utilization.

This circuit breaker monitors the JVM Old memory utilization. When the utilization exceeds `85%`, write requests will be rejected. If GC still cannot recycle the JVM Old memory, query requests will be rejected when the utilization reaches `90%`. If a request is rejected, the client will receive the following response:

```
{
    "status": 403,
    "error": {
        "root_cause": [{
            "reason": "pressure too high, (smooth) bulk request circuit break",
            "type": "status_exception"
        }],
        "type": "status_exception",
        "reason": "pressure too high, (smooth) bulk request circuit break"
    }
}
```

Bulk Rejection/Search Rejection

Last updated：2024-11-29 22:07:28

Problem Description

In some cases, the bulk rejection or search rejection rate of a cluster increases. Specifically, an error message similar to the following will appear when bulk writes/searches are performed:

Error 1:

```
[2019-03-01 10:09:58][ERROR]rspItemError: {"reason":"rejected execution of
org.elasticsearch.transport.TransportService$7@5436e129 on
EsThreadPoolExecutor[bulk, queue capacity = 1024,
org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@6bd77359[Running,
pool size = 12, active threads = 12, queued tasks = 2390, completed tasks =
20018208656]]","type":"es_rejected_execution_exception"}
```

Error 2:

```
[o.e.a.s.TransportSearchAction] [1590724712002574732] [31691361796] Failed to
execute fetch phase
org.elasticsearch.transport.RemoteTransportException: [1585899116000088832]
[10.0.134.65:22624][indices:data/read/search[phase/fetch/id]]
Caused by:
org.elasticsearch.common.util.concurrent.EsRejectedExecutionException: rejected
execution of org.elasticsearch.common.util.concurrent.TimedRunnable@63779fac on
QueueResizingEsThreadPoolExecutor[name = 1585899116000088832/search, queue
capacity = 1000, min queue capacity = 1000, max queue capacity = 1000, frame
size = 2000, targeted response rate = 1s, task execution EWMA = 2.8ms,
adjustment amount = 50,
org.elasticsearch.common.util.concurrent.QueueResizingEsThreadPoolExecutor@350d
a023[Running, pool size = 49, active threads = 49, queued tasks = 1000,
completed tasks = 57087199564]]
```

You can see that the bulk rejection/search rejection rate has increased in Cloud Monitor.

You can also view the number of bulk writes that are being rejected or have been rejected by running the following command in the Kibana console.

```
GET _cat/thread_pool/bulk?s=queue:desc&v
GET _cat/thread_pool/search?s=queue:desc&v
```

Generally, the default value for a queue is 1024. If there is 1024 under a queue, rejections have occurred on the node.

Troubleshooting

1. Check whether the body size of the bulk request is unreasonable. The size of a single bulk request should be below 10 MB; otherwise, it will take too long to process a single bulk request, causing the queue to fill up. However, if the size is too small, too many bulk requests will be generated, also causing the queue to fill up.

2. Check whether the write QPS matches the cluster configuration. The empirical value is that if shards are distributed evenly, a 4-core 16-GB 3-node cluster can sustain 20,000–30,000 write QPS, but more query requests lead to a lower QPS. Specifically, you can determine the highest write QPS capacity that the cluster can withstand through stress testing and select the appropriate configuration.

3. Check whether the shard data volume is too high. A too large shard size may result in bulk rejections; therefore, we recommend you limit the size of one shard to 20–50 GB. You can view the size of each shard in the index by running the following command in the Kibana console:

```
GET _cat/shards?index=index_name&v
```


Check whether the shards are unevenly distributed

Sometimes, the shards may be unevenly distributed across all the nodes in the cluster. Some nodes are assigned with too many shards, while some too few.

You can check that in **Cluster Monitoring** > **Node Status** on the cluster details page in the ES console. For more information, please see Viewing Monitoring Data.

You can also view the number of shards assigned to each node in the cluster by using ES APIs.

```
GET /_cat/shards?index={index_name}&s=node,store:desc
```

The results are as follows (the first column shows the number of shards, and the second shows the node ID), where some nodes are assigned with one shard, while some eight.

Solutions

1. Set the shard size

The shard size can be configured by using the `number_of_shards` parameter in the index template (after the template is created, it will take effect when you create new indexes, and previous indexes will not be adjusted).

2. Fix the uneven distribution of shards

Temporary solution

If you find that shards are not evenly assigned, you can dynamically adjust a certain index by setting the `routing.allocation.total_shards_per_node` parameter. For more information, please see Total Shards Per Node.

Note:

A certain buffer should be reserved for `total_shards_per_node` so as to prevent any machine failure from rendering allocation of shards impossible (for example, if there are 10 machines and an index has 20 shards, `total_shards_per_node` should be set to above 2, such as 3).

```
PUT {index_name}/_settings
 {
    "settings": {
      "index": {
        "routing": {
          "allocation": {
            "total_shards_per_node": "3"
          }
```

```
                }
            }
        }
    }
```

Set an index before production: set the number of shards per node through the index template.

```
PUT _template/ {template_name}
{
    "order": 0,
    "template": " {index_prefix@} *",  // Prefix of the index to be adjusted
    "settings": {
      "index": {
        "number_of_shards": "30",   // Specify the number of shards assigned to
the index based on a shard size of about 30 GB
        "routing.allocation.total_shards_per_node":3  // Specify the maximum
number of shards that a node can accommodate
      }
    },
    "aliases": {}
  }
```

High Cluster CPU Utilization

Last updated：2021-08-11 11:17:23

Problem Description

All nodes in the cluster have high CPU utilization, but there are not a lot of reads and writes. The specific problem can be seen on the **Stack Monitoring** page in Kibana:

In addition, you can also see the CPU utilization of each node on the node monitoring page in the ES console:

In this case, because the cluster read rate and write rate are not high, it is difficult to quickly find the root cause from the monitoring perspective. Therefore, you need to observe carefully and find the cause from the details. Below are several possible scenarios and corresponding troubleshooting ideas.

Note:

The situation where the CPU utilization of an individual node is much higher than that of other nodes is quite common. In most cases, this is caused by uneven load due to improper use of the cluster. For more information, please see Uneven Cluster Load.

Troubleshooting

Large query requests cause the CPU utilization to soar

This situation is relatively common, and clues can be found from monitoring. Monitoring data shows that the fluctuation of the query request volume is basically in line with the maximum CPU utilization of the cluster.
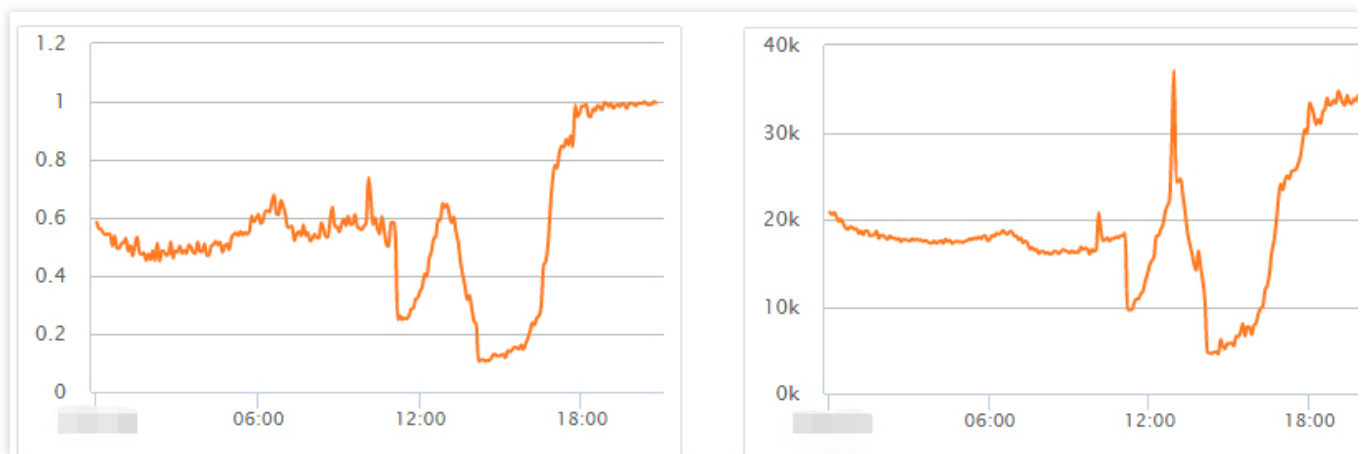
To further identify the problem, you need to enable slow log collection for the cluster. For more information, please see Querying Cluster Logs. You can get more information from the slow logs, such as the indexes that cause slow queries, query parameters, and query content.

Solutions

Try to avoid large text searches and optimize queries.
Use the slow logs to identify indexes where queries are slow. For some indexes with a small amount of data, set a small number of shards and multiple replicas, such as one-shard-multi-replica, to improve the query performance.

Write requests causes the CPU utilization to soar

If monitoring data shows that the CPU utilization surge is related to writes, then enable slow log collection for the cluster, identify slow write requests, and optimize them. You can also get the `hot_threads` information to identify which thread is consuming the CPU:

```
curl http://9.15.49.78:9200/_nodes/hot_threads
```

```
100.5% (502.4ms out of 500ms) cpu usage by thread 'elasticsearch[1604483193000692932][write][T#14]'
  4/10 snapshots sharing following 35 elements
    java.util.regex.Pattern$CharProperty$1.isSatisfiedBy(Unknown Source)
    java.util.regex.Pattern$CharProperty.match(Unknown Source)
    java.util.regex.Pattern$Curly.match0(Unknown Source)
    java.util.regex.Pattern$Curly.match(Unknown Source)
--
 99.9% (499.7ms out of 500ms) cpu usage by thread 'elasticsearch[1604483193000692932][write][T#9]'
  10/10 snapshots sharing following 33 elements
    java.util.regex.Pattern$Curly.match0(Unknown Source)
    java.util.regex.Pattern$Curly.match(Unknown Source)
    java.util.regex.Pattern$GroupHead.match(Unknown Source)
    java.util.regex.Pattern$Start.match(Unknown Source)
--
 98.8% (493.9ms out of 500ms) cpu usage by thread 'elasticsearch[1604483193000692932][write][T#16]'
  3/10 snapshots sharing following 34 elements
    java.util.regex.Pattern$BmpCharProperty.match(Unknown Source)
    java.util.regex.Pattern$Curly.match0(Unknown Source)
    java.util.regex.Pattern$Curly.match(Unknown Source)
    java.util.regex.Pattern$GroupHead.match(Unknown Source)
--
 99.9% (499.2ms out of 500ms) cpu usage by thread 'elasticsearch[1604483193000693132][write][T#4]'
  7/10 snapshots sharing following 35 elements
    java.util.regex.Pattern$CharProperty$1.isSatisfiedBy(Unknown Source)
    java.util.regex.Pattern$CharProperty.match(Unknown Source)
    java.util.regex.Pattern$Curly.match0(Unknown Source)
    java.util.regex.Pattern$Curly.match(Unknown Source)
--
 99.8% (499ms out of 500ms) cpu usage by thread 'elasticsearch[1604483193000693132][write][T#2]'
  6/10 snapshots sharing following 35 elements
    java.util.regex.Pattern$CharProperty$1.isSatisfiedBy(Unknown Source)
    java.util.regex.Pattern$CharProperty.match(Unknown Source)
    java.util.regex.Pattern$Curly.match0(Unknown Source)
    java.util.regex.Pattern$Curly.match(Unknown Source)
--
 99.7% (498.3ms out of 500ms) cpu usage by thread 'elasticsearch[1604483193000693132][write][T#7]'
  3/10 snapshots sharing following 35 elements
    java.util.regex.Pattern$CharProperty$1.isSatisfiedBy(Unknown Source)
    java.util.regex.Pattern$CharProperty.match(Unknown Source)
:
```

For example, it is found here that there are a lot of `ingest pipeline` operations, and such operations are very resource intensive.

```
java.util.regex.Matcher.find(Unknown Source)
java.util.regex.Matcher.replaceAll(Unknown Source)
org.elasticsearch.ingest.common.GsubProcessor.process(GsubProcessor.java:55)
org.elasticsearch.ingest.common.GsubProcessor.process(GsubProcessor.java:32)
org.elasticsearch.ingest.common.AbstractStringProcessor.execute(AbstractStringProcessor.java:69)
org.elasticsearch.ingest.Processor.execute(Processor.java:50)
```

Solutions

If you encounter the above problems, you need to optimize as appropriate on the business side. The key point of troubleshooting such problems is to make good use of the cluster's monitoring metrics to quickly locate the problems and then use the cluster logs together to identify the root causes, so that the problems can be solved quickly.

High Cluster Disk Utilization and read_only Status

Last updated：2024-11-29 22:14:22

Problem Description

When the disk utilization exceeds 85% or reaches 100%, the ES cluster or Kibana cannot provide services normally, and the following problems may occur:

When an index request is made, an error similar to `{[FORBIDDEN/12/index read-only/allow delete(api)];","type":"cluster_block_exception"}` is returned.

1. When an operation is performed on the cluster, an error similar to `[FORBIDDEN/13/cluster read-only / allow delete (api)]` is returned.

2. The cluster is in the red status. In severe cases, the node may not join the cluster (which can be viewed through the `GET _cat/allocation?v` command), and there are unassigned shards (which can be viewed through the `GET _cat/allocation?v` command).

3. The node monitoring page in the ES console shows that the disk utilization of cluster nodes has reached or approached 100%.

Troubleshooting

The above problems are caused by high disk utilization. The disk utilization of data nodes has the following three thresholds. Exceeding them may affect Elasticsearch or Kibana services.

When the cluster disk utilization exceeds 85%, new shards cannot be assigned.

When the cluster disk utilization exceeds 90%, Elasticsearch will try to migrate the shards on the corresponding node to other data nodes with lower disk utilization.

When the cluster disk utilization exceeds 95%, the system will forcibly set the `read_only_allow_delete` attribute for each index on the corresponding node in the Elasticsearch cluster. At this time, the node cannot write data to any index; instead, it can only read and delete the corresponding indexes.

Solutions

Clearing expired cluster data

1. You can access **Kibana** > **Dev Tools** to delete expired indexes to free up disk space in the following steps:

Warning:

Data cannot be recovered after deletion; therefore, please do so with caution. You can also choose to keep the data, but you need to expand the disk space.

Step 1. Enable batch operation for cluster indexes.

```
PUT _cluster/settings
 {
        "persistent": {
         "action.destructive_requires_name": "false"
        }
}
```

Step 2. Delete data, such as `DELETE NginxLog-12*` .

```
DELETE index-name-*
```

2. After performing the above step, if the version of your ES is below 7.5.1, you also need to run the following commands in **Dev Tools** on the Kibana page:

Disable the read-only status of the index by running the following command:

```
PUT _all/_settings
{
        "index.blocks.read_only_allow_delete": null
}
```

Disable the read-only status of the cluster by running the following command:

```
PUT _cluster/settings
{
        "persistent": {
            "cluster.blocks.read_only_allow_delete": null
        }
}
```

3. Check whether the cluster index is still in the `read_only` status and whether the index can be written to.

4. If the cluster is still in the red status, run the following command to check whether there are unassigned shards in the cluster.

```
GET /_cluster/allocation/explain
```

5. After the distribution of the shards is completed, check the cluster status. If the cluster status is still red, please submit a ticket for assistance.

6. In order to avoid high disk utilization from affecting Elasticsearch services, we recommend you enable disk utilization monitoring alarms, check the alarm messages in time, and take precautionary measures. For more information, please see Suggestions for Configuring Monitors and Alarms.

Expanding cloud disk space

If you don't want to clean cluster data, you can also expand the disk space on the **Cluster Configuration** page in the ES console in the following steps:

1. In the cluster list, click the **ID/Name** of the instance to be configured to enter the instance details page. Then, click **Adjust Configuration** on the **Basic Configuration** tab.
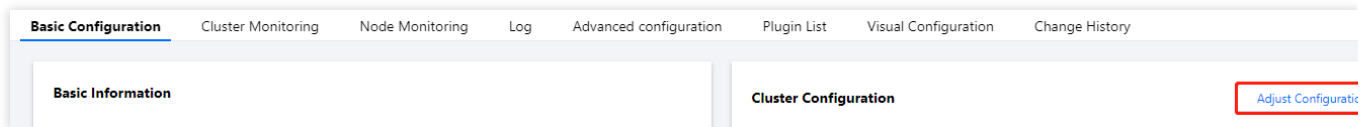
2. On the **Adjust Configuration** page, select the desired disk space for **Single-node Data Disk** and click **Next** to submit the task.
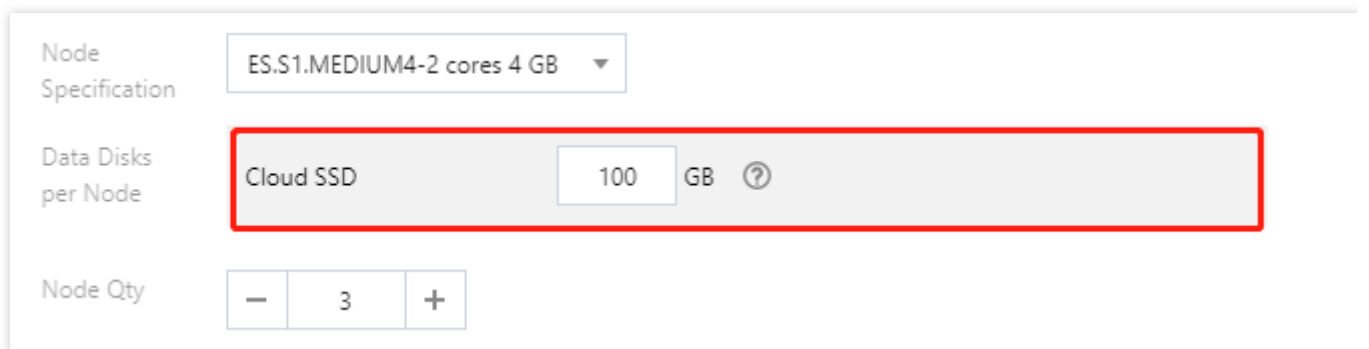
3. If the version of your ES is below 7.5.1, you also need to run the following commands in **Dev Tools** on the Kibana page:

Disable the read-only status of the index by running the following command:

```
PUT _all/_settings
  {
        "index.blocks.read_only_allow_delete": null
  }
```

Disable the read-only status of the cluster by running the following command:

```
PUT _cluster/settings
{
        "persistent": {
        "cluster.blocks.read_only_allow_delete": null
        }
}
```

Uneven Cluster Load

Last updated：2021-08-11 11:17:23

Problem Description

In some cases, the CPU utilization of certain nodes in a cluster is much higher than that of other nodes, which can be clearly observed in node monitoring in the ES console.

Causes

The design of index shards is inappropriate.

The sizes of segments are uneven.

There are typical scenarios with requirements for storage of hot and warm data.

Troubleshooting

Inappropriate shard settings

1. Log in to the Kibana console and run the following command in Dev Tools to view the index shard information. If nodes with a high load have more index shards, the shards are unevenly assigned:

```
GET _cat/shards?v
```

2. Log in to the Kibana console and run the following command in Dev Tools to view the index information. Check whether the node shards are unevenly assigned based on the cluster configuration:

```
GET _cat/indices?v
```

3. Assign shards again and appropriately plan them to ensure that the total number of primary and replica shards is an integer multiple of the number of data nodes in the cluster.

Note:

Elasticsearch also searches `.del` files during search and filters documents marked with `.del` , which reduces the search efficiency and wastes the specification resources. We recommend you force a merge during off-peak hours. For more information, please see Force merge API.

Suggestions for shard planning

The shard size and quantity are two important factors affecting the stability and performance of an Elasticsearch cluster. All indexes in the cluster require appropriate shard planning; otherwise, large shards in unspecific businesses may incur excessive Elasticsearch performance overheads. The following are some suggestions for shard planning: Keep the size of a single shard of an index between 20 and 50 GB.

1. Add a time suffix to the index, so you can implement scrolling by time for easier management.

2. While following the principle of single shard design, predict the final index size and estimate the number of index shards based on the number of cluster nodes so as to distribute the shards among the nodes as evenly as possible.

Note:

More primary shards are not necessarily better, as the more the primary shards, the higher the Elasticsearch performance overheads. We recommend you keep the total number of shards per node 30 times of the node memory. If there are too many shards, file handles are very likely to be used up, causing cluster failures.

Uneven segment sizes

1. Add `"profile": true` in the query body to check whether the `test` index has a shard whose query time is longer than that of other shards.

2. Specify `preference=_primary` and `preference=_replica` in the query separately and add `"profile": true` in the body to view the time it takes the primary and replica shards to make the query respectively. Check whether the primary or replica shard uses more time.

3. Log in to the Kibana console, run the following command in Dev Tools to view the shards, analyze the problem based on the segment information, and check whether the uneven loads are related to the uneven segment sizes.

```
GET _cat/segments/index?v&amp;h=shard,segment,size,size.momery,ip
GET _cat/shards?v
```

4. Solve the problem in either of the following methods:

Force a merge during off-peak hours. For more information, please see Force merge API. Delete `delete.doc` in the cache completely and merge small segments into big ones.

Restart the node where the primary shard resides to trigger promoting the replica shard to the primary shard and generate a new replica shard. Data in the new primary shard is replicated to the new replica shard to ensure that they have the same segments.

Typical scenarios with requirements for storage of hot and warm data

If you add `routing` in a query or query hotspot data with a high query frequency, uneven data load will definitely occur.