# ELK安装部署手册

## 版本说明

|  |  |
| --- | --- |
| 组件名称 | 版本 |
| jdk | 1.8 |
| elasticsearch |  |
| logstash |  |
| kibana |  |
| kafka |  |
| nginx |  |

## 操作系统配置

建议操作系统配置8G内存，300G以上磁盘

### 句柄，内存，线程数设置

使用root用户执行

vi /etc/security/limits.conf

添加如下配置

\* soft nofile 65535

\* hard nofile 65535

\* soft memlock unlimited

\* hard memlock unlimited

\* soft nproc 2048

\* hard nproc 4096

执行service sshd restart使修改生效

### JVM线程设置

使用root用户执行

Vi /etc/sysctl.conf

新增

vm.max\_map\_count=262144

保持文件后执行

sysctl –p生效修改

### 关闭swap

为了避免内存上的运算移交到了磁盘上运算导致，运算时间变长，可以关闭掉swap，执行

Sudo swapoff –a

如果需要永久的禁用可以在sysctl中配置 vm.swappiness = 1

如果以上的方式都不合适，可以在配置文件中打开mlockall开关，允许JVM锁住内存禁止操作系统交换出去。在elasticsearch.yml中配置bootstrap.mlockall: true

## Elasticsearch安装

### Head插件安装[可选]

### Bigdesk插件安装[可选]

## Elasticsearch优化

### Elasticsearch内存分配策略

Elasticsearch的分词字符串集合运算使用的是堆内存（ES\_HEAP\_SIZE），而lucene segment使用的则是非堆内存（off-heap），因此我们需要合理的分配ES\_HEAP\_SIZE和off-heap内存。

ES\_HEAP\_SIZE越多，内存数据结构操作更快，但是Lucene获取的内存就越少，GC更慢；反之则，GC跟快，Lucene可以获得更多的内存用于缓存数据结构。

因此通常来说，我们建议无论你的机器是高配的还是低配的，内存都需要平分给ES\_HEAP\_SIZE和off-heap。如果是高配的机器，可以选择在一个机器上面安装多个节点。

#### 堆内存ES\_HEAP\_SIZE设置

默认情况下es堆内存设置为1G，但是这个值绝对是太小了，如果使用这个值，集群可能会出问题，但是这个值也不建议大于32G，因为JVM在内存小于32G的时候采用的内存对象指针压缩技术（compressed oops），超过了32G,指针对象会切回普通的对象指针，导致更多地内存浪费，降低CPU的性能，需要GC应对大内存。

修改堆内存的大小有2种方式：

export ES\_HEAP\_SIZE=10g

也可以通过启动命令参数的形式设置

./bin/elasticsearch -Xmx10g -Xms10g

通常来说我们建议设置ES\_HEAP\_SIZE，而不是在启动命令中设置。

### Elasticsearch 分片策略

每个节点的分片数量应该保持在低于每1GB堆内存对应集群分片20-25个之间；分片大小为50G通常被认为是适用于各种用例限制。

分片多少需要考虑数据大小，更新频率，查询频率以及未来的状态。

#### 基本概念

**Lucene段（segment）**：利用操作系统的底层机制来缓存内存数据结构，这些段分别存储在单个文件中，由于段的不可变，这些文件也是不会变化的，操作系统会把这些文件缓存起来，以便快速的访问。

**刷新**：数据写入分片时，会定期的发布到磁盘上新的不可变Lucene段，此时数据可以用于查询

**合并**：随着段segment的增加，这些segment会被定期整合到一个大的segments中。

由于段的不可变，合并的时候，新的段创建，旧的段删除导致使用的磁盘空间会波动，合并的时候会使用较多。

由于大的分片将导致集群故障的时候恢复能力差，因此需要避免大的分片。但是分片的大小没有固定的要求，通常来说50G是各种用例的一个限制。

#### 分片成本

新增分片是会增加资源消耗的，主要表现为：

1. 每个分片本质上就是一个lucene索引，因此会消耗相应的文件句柄，内存和cpu
2. 每个搜索请求会调度到索引的每个分片，如果分片分布在同一个机器上，会导致竞争相同的硬件资源，导致性能的下降
3. Es使用词频统计来计算关联性，这些统计都分布在分片上，如果大量的分片只维护很少的数据，将导致文档的相关性比较差
4. 为组装各个分片的查询，在合并结果的时候，需要消耗大量的IO，因此使用ssd和多核处理器对分片性能有非常大的帮助

#### 堆内存对分片的影响

每个分片都有数据保存在内存中并使用堆空间，这些信息包含在分片级保存的数据结构，也包含segments保存的数据结构。

Segments的开销和segments的大小是成反比的，大的段具有较小的开销。

每个索引和分片都有一定资源的开销。

尽可能的管理和减少堆内存的使用量，节点拥有的堆空间越多，节点可以处理的数据和分片越多。

小的分片导致小的segment,从而增大segment的内存开销，对于基于时间的数据用例，大小为20-40G之间的分片是可取的。

每个分片的消耗取决于segment数量和大小，因此强制小的segment合并成大的segment是可取的，但是，由于合并的消耗是巨大的，因此需要避开高峰时间来做合并。

集群节点的分片数量和堆内存是成正比的，一般来说1G的内存对应集群的分片数量为20-25.当然，每个分片拥有的内存越多可以保持更好的效果。

#### 分片大小对查询速度的影响

每个查询在es中的每个分片中是单线程执行的，在多个分片中是并行的。可以在同一个分片上执行多个查询和聚合。

查询最小延迟取决于数据，查询类型，分片大小。

在并发数量不是很大的时候，查询小分片的检索速度要更快，在高并发的时候，则不一定了。多个并发查询，很多小碎片也会降低查询的吞吐量。

从查询性能的角度，最大分片的大小确认方式是使用逼真的数据进行基准测试，而不是使用模拟数据。

#### 避免分片的主副本存在同一个机器上

如果分片的主副本都存在同一个机器上面时，副本的高可用性就没有了，因此需要阻止这个行为。

cluster.routing.allocation.same\_shard.host: true

当然，如果每个机器上面只有一个节点的时候，是不需要考虑这个问题的。

### Elasticsearch索引策略

#### 索引时间周期

基于时间的索引，如果使用固定期限的时间索引数据，可以根据时间周期和预期数据量来调整所涵盖的时间范围，以达到目标的分片大小。

基于时间的索引，如果数据量均匀的更新，可以良好的运作，如果索引率快速变化，则很难保持均匀的分片大小。

使用Rollover 和 Shrink Api管理索引和分片的灵活性。

#### 索引有效期

由于segment是不可变的，导致在更新文档的时候，需要先查询到现有的文档并将其标记为已删除，在添加更新的文档，删除操作也是需要先查找到需要删除的文档并标记为删除，在下一次的合并到来之前，被标记为删除的文档依然会占用系统的资源。

由于对单个文档的删除并非实时的释放系统资源，因此，我们不建议去删除单个文档，而是直接删除整个的索引。

由于直接删除索引是最有效的方式，因此，我们的索引最好是基于时间来管理数据的。

基于时间管理，来可以轻松的改变（在生成下一个索引时更新）主分片和副本分片的数量。

#### 索引数量

Es的每个索引，其映射和状态的信息都存储在集群状态，这些信息都存储在内存中，以便快速的访问。如果拥有大量的索引，将导致出现大的集群状态。为保证集群状态的一致性，需要通过单线程来完成，因此更新的速度将变得很慢，如果索引过多的话。

为了避免过多的索引，建议是相同的数据结构存储在相同的索引中，而不是根据数据源来创建索引。

### 定时清理索引

1.编写清除脚本 clear\_index.sh 内容如下：

#!/bin/bash

i=7

echo `date`

while [ $i -le 15 ];

do

indexName=`date +"%Y%m%d" -d"-$i day"`

curl -XDELETE "http://172.30.150.26:29200/\*-$indexName"

i=`expr $i + 1 `

done

2.设置定时执行（每天1点执行）

0 1 \* \* \* /home/es/clear\_index.sh >>/home/es/clear\_index.log

测试可以设置为每一分钟执行一次

\*/1 \* \* \* \* /home/es/clear\_index.sh >>/home/es/clear\_index.log

### 模板使用

我们可以为es创建一些模板，让es在创建索引的时候，安装我们的要求来设置分片，字段内容等。

例如在ES\_HOME/config/templates/log.json写入如下内容:

{

"template": "log-\*",

"order": 1,

"settings": {

"number\_of\_shards": 4,

"number\_of\_replicas": 0,

"analysis": {

"tokenizer": {

"one\_char" :{

"type" : "nGram",

"min\_gram" : 1,

"max\_gram" : 1,

"token\_chars" : [ "letter", "digit", "whitespace", "punctuation", "symbol","math\_symbol","control"]

}

},

"analyzer": {

"substring\_match" : {

"type" : "custom",

"tokenizer" : "pattern",

"filter" : ["lowercase"],

"char\_filter" : []

}

}

}

},

"mappings": {

"\_default\_": {

"\_all": {

"enabled": true

}

},

"log" : {

"\_ttl": {

"enabled": true,

"default": "7d"

},

"properties" : {

"time" : {

"type": "long"

},

"log\_level" : {

"type" : "string",

"index": "not\_analyzed"

},

"machine" : {

"type" : "string",

"index": "not\_analyzed"

},

"app\_id" : {

"type" : "string",

"index": "not\_analyzed"

},

"instance\_id" : {

"type" : "string",

"index": "not\_analyzed"

},

"container\_name" : {

"type" : "string",

"index": "not\_analyzed"

},

"log\_type" : {

"type" : "string",

"index": "not\_analyzed"

},

"log\_data" : {

"type" : "string",

"analyzer": "english"

},

"filename" : {

"type" : "string",

"index": "not\_analyzed"

},

"service\_name" : {

"type" : "string",

"index": "not\_analyzed"

},

"region\_name" : {

"type" : "string",

"index": "not\_analyzed"

},

"region\_id" : {

"type" : "string",

"index": "not\_analyzed"

},

"service\_full\_name" : {

"type" : "string",

"index": "not\_analyzed"

},

"paths" : {

"type" : "string",

"index": "not\_analyzed"

},

"@timestamp" : {

"type" : "string",

"index": "not\_analyzed"

}

}

}

}

}

## Kibana安装

## Logstash安装