# Elastic search基础

# 一 什么是es？

Elasticsearch是一个实时分布式搜索和分析引擎。基于Apache Lucene(TM)，隐藏lucene复杂性，提供restful服务。

特点：

1. 分布式的实时文件存储，每个字段都被索引并可被搜索
2. 分布式的实时分析搜索引擎
3. 可以扩展到上百台服务器，处理PB级结构化或非结构化数据

## 业界场景

维基百科使用Elasticsearch提供全文搜索并高亮关键字、Github使用Elasticsearch检索1300亿行的代码。

## 1.2 es监控工具

Marvel是Elasticsearch的管理和监控工具

## 1.3分布式特性

es自动完成功能：

1. 文档分区到不同的容器或者分片**(shards)**中，它们可以存在于一个或多个节点

中。

1. 将分片均匀的分配到各个节点，对索引和搜索做负载均衡。
2. 冗余每一个分片，防止硬件故障造成的数据丢失。
3. 将集群中任意一个节点上的请求路由到相应数据所在的节点。
4. 无论是增加节点，还是移除节点，分片都可以做到无缝的扩展和迁移。

# 二 分布式集群

主节点(master):: 临时管理集群级别的一些变更，例如新建或

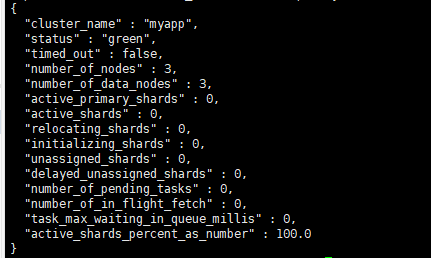
删除索引、增加或移除节点等

## 2.1 集群健康

集群健康分为三种状态:green、yellow或red。

查看es集群状态：

GET /\_cluster/health



|  |  |
| --- | --- |
| 颜色 | 意义 |
| green | **所有主要分片和复制分片都可用** |
| yellow | 所有主分片可用，部分复制分片可用 |
| red | 部分主分片可用 |

## 2.2 添加索引

**索引**：用来指向一个或者多个分片(shards)的”逻辑命名空间(logic namespace)”

### 2.2.1 分片

**分片**: 是一个最小级别的工作单元，只保存索引中所有数据的一部分。分片本质上来说就是一个lucene实例。

分片类型：主分片或复制分片，复制分片是主分片的副本，防止故障导致的数据丢失，同时提供读请求，比如搜索或者从别的分片取回文档。

创建分片

PUT /blogs

{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

}

}

分配3个主分片和一个复制分片，curl具体命令

curl -H "Content-Type: application/json" -XPUT ip地址:9200 -d ‘{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

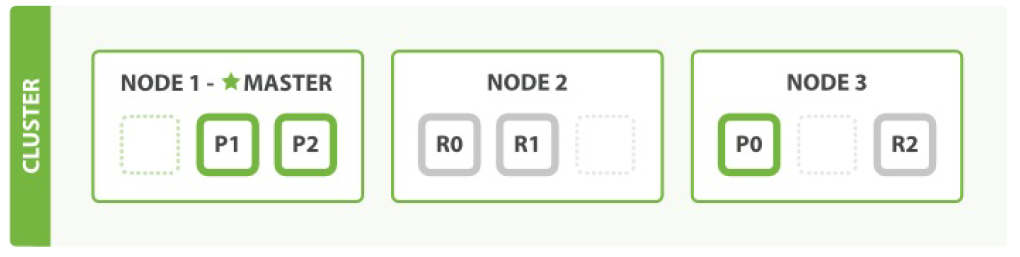
}

}’

注意 大括号前面或后面有引号。

## 2.3 横向扩展

如果集群扩展到三个节点(之前是1个)，那么此时三个分片分布如下:



复制分片的数量在集群中的节点数量发生变化时也能进行改变，数据冗余越多，能处理搜索吞吐量就越大。

PUT /blogs/\_settings

{

"number\_of\_replicas" : 2

}

可以将冗余分片增加到两个

## 2.4 集群增减节点对分片影响

杀掉一个节点的es进程，会发生如下事情

1. 如果节点是主节点，那么集群会重新选举主节点
2. 由于关掉的节点A上有主分片，导致其他节点上对应复制分片没有主分片，这时候，主节点会让节点B、C上对应复制分片有一个提升为主分片，这时候状态是yellow，因为复制分片数量要求是2，这时候集群所有主分片复制分片只有一个，

增加一个节点es进程后，集群会将复制分片分配到这个节点上达到保持复制分片数量。

# 三 数据

## 文档

Elasticsearch是一个分布式的文档(document)存储引擎。它可以实时存储并检索复杂数据结构——序列化的JSON文档。

### 文档元数据

保存文档信息，三个必须的元数据节点是：

|  |  |
| --- | --- |
| 节点 | 说明 |
| \_index | 文档存储的地方 |
| \_type | 文档代表的对象的类 |
| \_id | 文档的唯一标志 |

### \_index

索引(index)类似于关系型数据库里的“数据库”，用于存储和索引关联数据。

数据被存储和索引在分片(shards)中，索引知识一个把一个或多个分片分组在一起的逻辑空间。

## \_type

类型用来表示多个事物具有相同特征，在es中，相同类型的文档具有相同的数据结构，存储在一个索引下。每个类型(type)都有自己的映射或者结构定义。类型的映射告诉es不同的文档如何被索引。

### \_id

Id仅是一个字符串，用来在es中唯一标识一个文档。

## 索引一个文档

使用自己的ID建立文档：

PUT /{index}/{type}/{id}

{

"field": "value",

...

}

例如我们的索引叫做 “website” ，类型叫做 “blog” ，我们选择的ID是 “123” ，那么这个索引

请求就像这样：

PUT /website/blog/123

{

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/01"

}

Elasticsearch的响应：

{

"\_index": "website",

"\_type": "blog",

"\_id": "123",

"\_version": 1,

"created": true

}

\_version:每当文档发生变化，\_version都会增加

如果在添加文档没有指定id，那么es会给我们随机提供提个唯一id，长度22个字符，或者叫UUIDs。

## 检索一个文档

使用GET请求，向es发起请求：

**GET /website/blog/123?pretty**

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 1,

"found" : true,

"\_source" : {

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/01"

}

}

发现增加了\_source元数据字段。

pretty: 代表JSON文档以好看的方式打印出来。

### 3.3.1 检索文档一部分

GET /website/blog/123?\_source=title,text

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 1,

"exists" : true,

"\_source" : {

"title": "My first blog entry" ,

"text": "Just trying this out..."

}

}

只返回\_source字段不反悔其他的元数据

GET /website/blog/123/\_source

{

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/01"

}

## 检查文档是否存在

使用HEAD方法代替GET即可。

curl -i -XHEAD <http://localhost:9200/website/blog/123>

若文档存在，将返回200 OK状态

HTTP/1.1 200 OK

Content-Type: text/plain; charset=UTF-8

Content-Length: 0

若不存在

HTTP/1.1 404 Not Found

Content-Type: text/plain; charset=UTF-8

Content-Length: 0

## 更新整个文档

PUT /website/blog/123

{

title": "My first blog entry",

"text": "I am starting to get the hang of this...",

"date": "2014/01/02"

}

响应

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 2,

"created": false <1>

}

可以看到es将\_version增加了。created标志为false，是因为同索引、同类型已存在同ID文档。

Update API过程：

1. 从旧文档中检索JSON
2. 修改它
3. 删除旧文档
4. 索引新文档

## 创建新文档

\_index、\_type、\_id三者唯一确定衣蛾文档。最简单方式通过POST方法让ES自动生成唯一ID:

**POST /website/blog/**

**{ ... }**

**其次，自己指定id**

PUT /website/blog/123/\_create

{ ... }

如果成功，则响应状态码是201 created.若失败，返回409 Conflict状态码。

## 删除文档

使用DELETE方法：

DELETE /website/blog/123

如果成功，es返回200 OK状态码和以下相应体。

{

"found" : true,

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 3

}

如果文档未找到，将得到404 Not Found状态码，响应体是:

{

"found" : false,

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 4

}

尽管found为false，\_version还是增加了，主要是为了确保多节点之间不同操作能够保持正确顺序。

## 并发控制

Es并发问题

Es在并发修改的时候可能会出现数据不一致的问题，比如两个web端同时使用一个es，同时库存减一，最后发现es中库存剩余没有减2，反而只少了一个。

**悲观并发控制**:

将访问es行资源加锁，只有获取锁的线程可以修改这行数据。

**乐观并发控制**:

读写过程中进行检测，数据发生变化没有，如果发生变化，更新操作失败。可以重新尝试更新，刷新数据或者直接反馈给用户。

Es提供了\_version参数，这个参数代表document的版本号，可用于控制并发读写的乐观控制。

### 3.8.1 内部版本控制

Es提供了乐观写，

PUT /website/blog/1?version=1 <1>

{

"title": "My first blog entry",

"text": "Starting to get the hang of this..."

}

<1>代表只希望文档的\_version是1时更新文档才会有效。

如果请求成功，响应体中\_version会增加到2.

{

"\_index": "website",

"\_type": "blog",

"\_id": "1",

"\_version": 2

"created": false

}

### 3.8.2 外部版本控制

由外部程序自己制定版本号，使用方式:

PUT /website/blog/2?version=5&version\_type=external

{

"title": "My first external blog entry",

"text": "Starting to get the hang of this..."

}

增加文档时后面添加 version=版本号&version\_type=external

## 文档局部更新update

Es文档是不可变的，不能被修改只能被替换，es在使用update API处理数据时也是经过了检索-》修改-》重建索引的过程。

使用以下请求为博客

添加一个 tags 字段和一个 views 字段：

POST /website/blog/1/\_update

{

"doc" : {

"tags" : [ "testing" ],

"views": 0

}

}

更新阶段发生冲突如何解决？

Es重试机制，es提供了如下请求方式，乐观失败重试：

POST /website/pageviews/1/\_update?retry\_on\_conflict=5 <1>

{

"script" : "ctx.\_source.views+=1",

"upsert": {

"views": 0

}

}

## 检索多个文档

POST /\_mget

{

"docs" : [

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : 2

},

{

"\_index" : "website",

"\_type" : "pageviews",

"\_id" : 1,

"\_source": "views"

}

]

}

## 3.11 批量更新

使用bulk命令时，REST API以\_bulk结尾,批量操作写在json文件中，官网给出的语法格式：

action\_and\_meta\_data\n

optional\_source\n

action\_and\_meta\_data\n

optional\_source\n

....

action\_and\_meta\_data\n

optional\_source\n

命令操作类型占一行 数据另起一行

如 create 格式 案例

{"index":{"\_id":"17"}} // 需要插入的数据类型 action

{"name":"cddd","id":17} // 带插入的action 对应的数据内容

1. **更新中的所有子请求异步，不互相干扰，其中一个子请求失败，整个请求会标志位失败。**
2. **批量更新请求会占用节点内存空间，需要把握好数据量大小，不能太大。一个好的批次可以保持在5-15MB之间。**

# 四 分布式增删改查

## 路由

文档分布式储存，多个节点可以为一个请求提供服务。Es提供了路由机制用于分发请求。这一功能就是路由。

文档存储到哪一个主机,是根据一个简单的算法决定的:

**shard 等于 hash(routing) % number\_of\_primary\_shards**

routing值是一个任意字符串,吗，默认是\_id也可以自定义。Routing字符串通过哈希函数生成一个数字，然后除以主切片数量得到一个余数。余数决定文档存储的主机位置。

Es规定主分片数量在创建索引时定义且不能修改，这是因为如果主分片数量发生改变，那么一个请求过来就找不到原来的分片位置了。

## 主分片和路由分片之间的交互

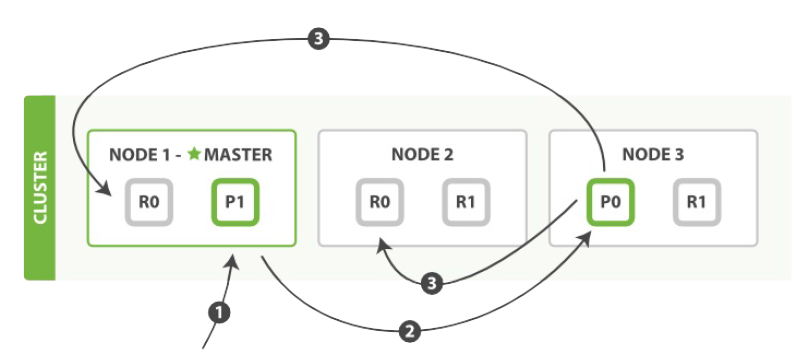
假设有三个节点的集群。它包含一个叫做 bblogs 的索引并拥有两个主分片。每个主分片有两个复制分片。相同的分片不会放在同一个节点上，所以我们的集群是这样的：



上面三个节点都可以提供服务。

### 4.2.1 新建、索引和删除文档

新建、索引和删除请求都是写(write)操作，它们必须在主分片上成功完成才能复制到相关的复制分片上。



新建、索引和删除请求途径的步骤：

1. 客户端给 Node 1 发送新建、索引或删除请求。
2. 节点使用文档\_id确定文档属于分片0.它转发请求到Node3，分片0位于这个节点上。
3. Node3在主分片上执行请求，如果成功，将转发请求到相应的位于Node1和Node2的复制节点上。当所有复制节点报告成功，Node3报告成功到请求的节点，请求的节点再报告给客户端。

### 4.2.2 replication

复制默认值是sync，代表主分片要得到所有复制分片的响应才会返回。

如果设置replication为async，请求在主分片上被执行后就会立即响应给客户端，主分片依旧会把请求转发给复制节点，但是客户端是不清楚复制节点执行请求的成功与失败的。async

复制可能会让其他机器未就绪的情况下发送过多请求而使elasticsearch负载。

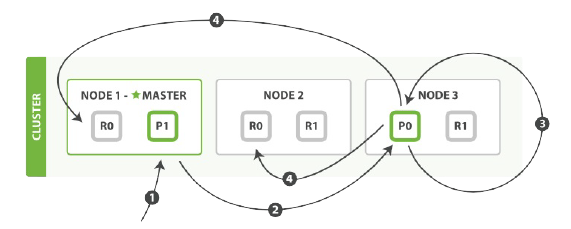
### 4.3.3 一致性

主分片尝试写入时规定数量或者过半的分片可用。防止数据被写入到错的网络分区。规定数量计算公式：

Int(primary+number\_of\_replicas)/2)+1

分片不足时，es会等待更多分片出现。默认等待一分钟。

## 4.4局部更新文档



下面给出必要的顺序步骤:

1. 客户端给任意节点发送请求
2. 如果这个节点上没有目的主分片。将请求转发到目的主分片所在主机。如果有则不用转发
3. 目的主分片所在主机修改\_source字段JSON，然后在主分片上重建索引。如果有其他进程修改了文档。它以retry\_on\_conflict设置的次数重复步骤3，都未成功则放弃。
4. 如果成功则更新文档，同时转发文档的新版本到其他复制分片所在主机，来重建索引。当所有复制节点报告成功，主分片所在节点返回成功给请求节点。请求节点返回给客户端。

# 五 搜索

## 空搜索

GET /\_search

响应内容：

{

"hits" : {

"total" : 14,

"hits" : [

{

"\_index": "us",

"\_type": "tweet",

"\_id": "7",

"\_score": 1,

"\_source": {

"date": "2014-09-17",

"name": "John Smith",

"tweet": "The Query DSL is really powerful and flexible",

"user\_id": 2

}

},

... 9 RESULTS REMOVED ...

],

"max\_score" : 1

},

"took" : 4,

"\_shards" : {

"failed" : 0,

"successful" : 10,

"total" : 10

},

"timed\_out" : false

}

### hits

响应中最重要的是hits。包含total字段表示匹配到的文档总数，页包含了搜索匹配到的文档。

Hits数组中每个结果都包含\_index、\_type和文档\_id字段，被加入到\_source字段中意味着在搜索结果中可以直接使用全部文档。

每个文档中有\_score字段，这是相关性得分(relevance score)。衡量了文档与查询的匹配程度。

### took

took代表整个搜索花费的毫秒数。

### shards

\_shards节点代表参与查询分片数量，有多少是成功的，多少是失败的。

### timeout

查询超时与否，一般，搜索请求不会超时。

## 分页

Es使用from和size参数：

from参数： 跳过开始的结果数，默认0

size: 结果数，默认10

使用方式:

GET /\_search?size=5

GET /\_search?size=5&from=5

GET /\_search?size=5&from=10

六