# Elasticsearch

## 入门

Elasticsearch是一个高度可扩展且开源的全文检索与分析引擎。它可以让我们快速且近实时地存储，检索以及分析海量数据。

### 应用场景

l 运行一个在线的网店，你允许客户能够去搜索你销售的商品。在这种情况下，你可以使用ElasticSearch来存储整个商品目录和清单，并且提供搜索和为客户提供自动的建议。

l 你打算去收集日志或者交易数据，并且你想通过分析和挖掘这些数据去发现趋势，统计信息，总体概要或异常情况。在这种情况下，你可以使用Logstash去收集，聚合和解析数据，并且使用Logstash将数据加载到ElasticSearch。一旦数据进入ElasticSearch，你就可以进行搜索和聚合操作，以便去挖掘对你感兴趣的任何信息。

l 你运行一个价格预警系统，它允许对价格精明的客户可以指定一个规则，比如“我对购买电子产品感兴趣，并且在下个月内，如果感兴趣的产品（来自任何的供应商）价格低于X元时我希望自己能够被实时的提醒”。 在这种情况下，你可以把供应商的价格，推送到ElasticSearch中并且使用它的reverse-search（Percolator，Percolator是由谷歌推出的，在海量数据(PB级)上实现增量计算的平台。它使得在已有的结果集上进行小粒度的更新更加快速）能力去匹配价格变动，以此来应对客户查询以及最后把匹配的结果实时推送给客户。

l 你有一个分析/商业智能需求并且想快速调查，分析，可视化和应对海量数据（上百万或几十亿的数据量）的ad-hoc查询。在这种情况下，你可以使用ElasticSearch进行数据存储并且使用Kibana去建立自定义的分析的仪表板，以此实现数据的可视化特性。另外，你对处理的数据使用Elasticsearch聚合函数功能实现复杂的商业智能查询。

### 基础概念

#### 近实时（near realtime）

Es是一个近实时的搜索平台。这意味着从索引一个文档开始直到它可以被查询会有轻微的延迟时间（通常为一秒）

#### 集群（cluster）

集群是一个或者多个节点的集合，它们一起保存数据并且提供所有节点联合索引以及搜索功能。

#### 节点（node）

节点是一个单独的服务器，它是集群的一部分，存储数据，参与集群的索引和搜索功能。

#### 索引（index）

索引是具有稍微类似特征文档的集合（类比数据库的确比较好理解，但是之后就会下意识用数据库的概念来带入）。

#### 类型（type）

在索引中，可以定义一个或多个类型。一个类型是索引中的一个逻辑的种类/分区。

#### 文档（document）

文档是索引信息的基本单位，代表一条记录，尽管文档物理的存在于索引中，实际上一个文档必须被索引/分配给索引内的类型。

#### 分片与副本（shards&replicas）

索引可以存储大量数据，可以超过单个节点的硬件限制。例如，10亿文档占用了1T的磁盘空间的单个索引显然不适合放在单个节点的磁盘上，并且从单个节点服务请求会变慢。

为了解决这个问题，es提供了把索引拆分到多个分片的能力。在创建索引的时候，可以定义分片的数量。每个分片都是一个具有全功能的和独立的”索引”，分片可以存储在集群的任何节点上。

分片的两个优势：一是水平的拆分/扩展；二是分布式和并行跨shard操作（可能在多个节点），从而提高了性能/吞吐量。

分片的分布式机制以及它的文档是如何聚合支持搜索请求是完全由es管理的，并且是对用户透明的。

在网络/云环境中随时可能会故障，无论出于何种原因，在shard/node不知何故会离线或者消失的情况下强烈建议设置故障转移是非常有效的。为了达到这个目的，es可以让你设置一个或者多个索引的shard副本到所谓的副本分片，或者副本中取。

启用副本非常重要的两个理由：

1. 在shard/node故障的情况下提供了高可用性。为了达到这个目的，需要注意的是在原始的主分片被复制时副本的分片不会被分配到相同的节点上。
2. 可以让你水平扩展搜索量/吞吐量，因为搜索可以在所有的副本上并行执行。

总而言之，每个索引可以被拆分成多个分片，一个索引可以设置0个或多个副本。开启副本后，每个索引将有主分片和副本分片。分片和副本的数量在索引被创建时能够被指定。在创建索引后，可以在任何时候动态的改变副本的数量，但是不能够改变分片数量。

**Notice：**

每个es分片是一个lucene索引。单个lucene索引中有一个最大的文档数量限制。从lucene-5843的时候，该限制为2147483519（=interger.MAX\_VALUE-128）。可以使用\_cat/shards api来监控分片大小。

### 安装

单机的安装其实没什么内容，主要就是需要安装java环境，一般现在都是jdk1.8以上。之后解压启动即可，或者进行配置下启动，集群的搭建的差异也就在于对配置文件的配置。

直接通过相应的命令启动即可，同时也可以同如下命令修改集群名与节点名。

以上，内容到此为止，接下来的内容都是按照个人的思路进行整理的内容，如需对上述内容延续学习，可参考：<http://cwiki.apachecn.org/pages/viewpage.action?pageId=4260680>

## 查询功能总结

### size

用来指定返回结果数目，可以放在一级查询标签，也可以是term标签里 ，其它的地方的话待发现。

### \_source

查询时，指定返回的\_source字段包含/排除的字段。

"\_source": {

"include": [

"xss"

],

"exclude": [

"result"

]

}

### Fields

字段存储设为true的时候，可以指定返回的filed内容

"fields": [

"angin\_ip"

]

### From

确定返回数据的内容，从多少条开始，后面是size确定取多少条数据。

### Sort

用于返回结果的排序

"sort": [

{

"mem\_used": "desc"

},

{

"net\_send": "desc"

}

]

或者：

"sort": {

"mem\_used": {

"order": "desc"

},

"net\_send": {

"order": "desc"

}

}

### Query

Query内容来完成es的查询功能包括的内容较多。

#### Match

是一个标准查询，如果查询的内容是一个分词字段，会在查询之前用分词器先分析一下查询字符，如果查询的是一个精确值的话，就是进行精确匹配查询。

#### Match\_all

查询所有文档，也是没有查询条件下的默认语句。

#### Match\_phrase

短语匹配，首先解析查询字符串产生一个词条列表，然后搜索所有词条，但是只保留包含所有搜索词条的文档，并且词条的位置要邻接。

#### Multi\_match

允许在match查询的基础上同时搜索多个字段，在多个字段中同时查一个内容。

"query": {

"multi\_match": {

"query": "50",

"fields": [

"net\_send",

"cpu\_used"

]

}

}

<http://www.cnblogs.com/yjf512/p/4897294.html>

#### term

完全匹配，查询的字符串必须是字段里的单独分词，且查询字符必须要小写。

#### Terms

与term类似，但是terms允许指定多个匹配条件。如果字段指定了多个值，那么文档满足一种一项即可。

#### Range

Range让我们能够按照指定范围查找一批数据：

{

"query": {

"range": {

"cpu\_used": {

"from": 30,

"to": 50

}

}

}

}

#### Exists

{

"query": {

"exists": {

"field": "cpu\_used"

}

}

}

{

"query": {

"missing": {

"field": "config\_id"

}

}

}

通过判断是否存在来确定内容。

#### Bool

Bool查询，合并多个查询子句，介绍是分bool查询与bool过滤（差别就是bool过滤里的查询子句是term查询，介绍则是将term查询当做了term过滤，这个是疑惑的地方）。

##### Should

Or

{

"query": {

"bool": {

"should": [

{

"match": {

"cpu\_used": 50

}

},

{

"match": {

"mem\_used": 2.56

}

}

]

}

}

}

同时也有个mininum\_should\_match的属性，最小匹配度的概念，可以取整数，也可以配置百分比。

##### Must

&

##### Must\_not

not(a&b…)

##### filter

在查询前过滤

#### wildcards（通配符）

通配符查询

{

"query": {

"wildcard": {

"message": "se??o?e"

}

}

} ?代表任意字符，\*代表0个或多个字符。

#### Regexp（正则）

{

"query": {

"regexp": {

"message": "s.\*"

}

}

}

#### Prefix（前缀）

{

"query": {

"prefix": {

"message": "i"

}

}

} 在message字段查询i开头的内容。

### Aggs

用于聚合用。

聚合的格式一般如下，想简写可把aggregations换成aggs：

"aggregations" : {

    "<aggregation\_name>" : {

        "<aggregation\_type>" : {

            <aggregation\_body>

        }

        [,"meta" : {  [<meta\_data\_body>] } ]?

        [,"aggregations" : { [<sub\_aggregation>]+ } ]?

    }

    [,"<aggregation\_name\_2>" : { ... } ]\*

}

与查询DSL一样，聚合也拥有一种可组合的语法:独立的功能单元可以被混合在一起来满足你的需求，这意味着基本的概念虽然不多，但是其组合的方式是几近无穷的。

聚合主要包括两个主要概念：

桶（buckets）：满足某个条件的文档集合。

指标（metrics）：为某个桶中的文档计算得到的统计信息。

每个聚合都是由一个或者多个桶，零个或者多个指标组合而成。

以sql为例：select count(color) from table1 group by color

这里的count(color)就相当于一个指标。group by color则相当于一个桶。

#### Metrics（指标）

##### Value count aggregation(值计数聚合)

计算从文档中提取/获取的值的数量。这些值可以从文档指定的字段中提取，或者从给定的脚本中生成。

{

    "aggs" : {

        "types\_count" : { "value\_count" : { "field" : "type" } }

    }

}

从脚本中生成:

{

    "aggs" : {

        "type\_count" : {

            "value\_count" : {

                "script" : {

                    "inline" : "doc['type'].value"

                }

            }

        }

    }

}

或者从脚本文件中生成：

{

    "aggs" : {

        "types\_count" : {

            "value\_count" : {

                "script" : {

                    "file": "my\_script",

                    "params" : {

                        "field" : "type"

                    }

                }

            }

        }

    }

}

##### 地理边界聚合&地理重心聚合

参考：<http://cwiki.apachecn.org/pages/viewpage.action?pageId=9406909>

##### 基数聚合

满足条件的bucket里某个field中不同值的数目。

{

"aggs": {

"agg1": {

"cardinality": {

"field": "message"

}

}

}

}

计算message字段中不同值的数目

至于用脚本实现，需要之后了解，参照:

<http://cwiki.apachecn.org/pages/viewpage.action?pageId=5505309>

##### 平均值聚合

{

"size": 0,

"aggs": {

"agg1": {

"avg": {

"field": "mem\_used"

}

}

}

}

聚合求平均值，遇到字段值缺省的时候，可以将其转换

{

"size": 0,

"aggs": {

"agg1": {

"avg": {

"field": "mem\_used",

"missing": 10

}

}

}

}

至于使用脚本的方式，参考：

<http://cwiki.apachecn.org/pages/viewpage.action?pageId=5505218>

##### 扩展统计聚合

这里是在bucket上，获取各种聚合的值，包括count（计数）、min（最小值）、max（最大值）、avg（平均值）、sum（总和）、sum\_of\_squares（平方和）、variance（方差）、std\_deviation(标准差)、std\_deviation\_bounds（平均加/减两个标准差的区间，用于可视化你的数据方差）

"aggs": {

"agg1": {

"extended\_stats": {

"field": "mem\_used"

}

}

}

结果：

"agg1": {

"count": 1000000,

"min": 1,

"max": 3.99,

"avg": 2.4942125499999843,

"sum": 2494212.5499999844,

"sum\_of\_squares": 6971964.064099466,

"variance": 0.7508678195220418,

"std\_deviation": 0.8665262947666631,

"std\_deviation\_bounds": {

"upper": 4.22726513953331,

"lower": 0.7611599604666581

}

脚本内容参考：<http://cwiki.apachecn.org/pages/viewpage.action?pageId=10027518>

##### 最大值聚合

换成max即可，涉及脚本内容参考：

<http://cwiki.apachecn.org/pages/viewpage.action?pageId=9405826>

##### 最小值聚合

换成min即可，涉及脚本内容参考：

<http://cwiki.apachecn.org/pages/viewpage.action?pageId=9405994>

##### 统计聚合

这个统计内容包括count、min、max、avg、sum

##### 百分位法统计

一般的应用场景如下：

运维人员记录每次启动系统所需要的时间，然后对这个时间分布进行统计；

网站记录每次用户访问的页面加载时间，然后对这个时间进行百分位统计。

"agg1": {

"percentiles": {

"field": "mem\_used"

}

}

"aggregations": {

"agg1": {

"values": {

"1.0": 1.02,

"5.0": 1.1432445286195285,

"25.0": 1.744648896918908,

"50.0": 2.4924286334797316,

"75.0": 3.2444725293818104,

"95.0": 3.8480331043098057,

"99.0": 3.97

}

}

}

结果一般都是这样分布。

##### 百分位指定范围统计

上面的key从最小值到最大值，如果想要统计指定value的百分比，则需要百分位指定范围统计。

"agg1": {

"percentile\_ranks": {

"field": "mem\_used",

"values": [

1.5,

2.5

]

}

}

"agg1": { "values": { "1.5": 16.852496009536146, "2.5": 50.188323245922106 } }

#### Bucket（桶）

##### Filter

先过滤后聚合，下例就是通过filter得出cpu\_used为2.56的文档集（桶），在此基础上再求net\_send的平均值。

"agg1": {

"filter": {

"term": {

"cpu\_used": 2.56

}

},

"aggs": {

"agg2": {

"avg": {

"field": "net\_send"

}

}

}

}

##### Range

根据数据的分布情况取桶，直接进行统计，并可以在此基础上进行指标聚合。

"agg1": {

"range": {

"field": "cpu\_used",

"ranges": [

{

"from": 2.56

},

{

"to": 60

}

]

},

"aggs": {

"agg2": {

"avg": {

"field": "net\_send"

}

}

}

}

##### Missing

找出某个字段为空的文档数，并可以在此（桶）基础上进行聚合。

"agg1": {

"missing": {

"field": "config\_id"

},

"aggs": {

"agg2": {

"avg": {

"field": "create\_by"

}

}

}

}

##### Terms

针对某个字段内容排重后统计个数（根据字段内的value来，如果是分词字段，会取分词后的结果进行排重统计）。

"agg1": {

"terms": {

"field": "cpu\_used"

},

"aggs": {

"agg2": {

"avg": {

"field": "net\_send"

}

}

}

}

##### Date range

对时间字段取时间范围的记录数，并可以在此基础上进行聚合。

"agg1": {

"date\_range": {

"field": "date",

"format": "yyyy-MM-dd",

"ranges": [

{

"from": "2016-7-21"

}

]

},

"aggs": {

"agg2": {

"max": {

"field": "date"

}

}

}

}

##### Global aggregation

默认的聚合都是在查询的结果之上，先取桶结果，然后再聚合。这里指定聚合的作用域与查询的作用域没有关联。因此返回结果中query命中的文档，与聚合的统计结果是没有关联的。

{

"query" : {

"match" : { "title" : "shirt" }

},

"aggs" : {

"all\_products" : {

"global" : {},

"aggs" : {

"avg\_price" : { "avg" : { "field" : "price" } }

}

}

}

}

##### Histogram

跟range类似，不过histogram不需要你指定统计区间，只需要提供一个间隔区间的值。"agg1": {

"histogram": {

"field": "cpu\_used",

"interval": 20

}

}

"buckets": [  
        {  
          "key": 0,  
          "doc\_count": 622  
        },  
        {  
          "key": 20,  
          "doc\_count": 702  
        },  
        {  
          "key": 40,  
          "doc\_count": 694  
        },  
        {  
          "key": 60,  
          "doc\_count": 714  
        },  
        {  
          "key": 80,  
          "doc\_count": 746  
        }  
      ]

##### Date histogram

与histogram类似，不过针对的是时间字段。

"aggs1" : {

"date\_histogram" : {

"field" : "date",

"interval" : "1M",

"format" : "yyyy-MM-dd"

}

}

##### Ipv4 range

"aggs1" : {

"ip\_range" : {

"field" : "ip",

"ranges" : [

{ "to" : "10.0.0.5" },

{ "from" : "10.0.0.5" }

]

}

}