Contents

[ElasticSearch 1](#_Toc460959120)

[Introduction 1](#_Toc460959121)

[install and run 1](#_Toc460959122)

[与Elasticsearch交互 2](#_Toc460959123)

[查询字符串(query string)搜索 3](#_Toc460959124)

[使用DSL语句查询 3](#_Toc460959125)

[全文搜索 4](#_Toc460959126)

[分析 5](#_Toc460959127)

[集群内部工作方式 5](#_Toc460959128)

# ElasticSearch

## Introduction

Elasticsearch是一个**实时分布式搜索和分析引擎**,它用于全文搜索、结构化搜索、分析以及将这三者混合使用

大部分数据库能够通过时间戳或者精确匹配做过滤，但是**无法全文搜索，处理同义词和根据相关性给文档打分，无法根据同一份数据生成分析和聚合**的结果，**无法做到对数据的实时处理**。

Elasticsearch是一个基于Apache Lucene(TM)的开源搜索引擎

Lucene只是一个库。想要使用它，你必须使用Java来作为开发语言并将其直接集成到你的应用中

Elasticsearch也使用Java开发并使用Lucene作为其核心来实现所有索引和搜索的功能，但是它的**目的是通过简单的RESTful API来隐藏Lucene的复杂性，从而让全文搜索变得简单**

Marvel是Elasticsearch的管理和监控工具，在开发环境下免费使用。它包含了一个叫做Sense的交互式控制台，使用户方便的通过浏览器直接与Elasticsearch进行交互。

## install and run

$wget <https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/tar/elasticsearch/2.3.5/elasticsearch-2.3.5.tar.gz>

$tar zxvf elastic…gz

$cd elasti…gz

$tree 查看文件目录结构

./bin/plugin -i elasticsearch/marvel/latest //install marvel

若需要代理:

./bin/plugin -Dhttps.proxyHost=https://165.225.96.34/ -Dhttps.proxyPort=10015 \

install elasticsearch/marvel/latest

$./bin/plugin install -h

//On Linux and Mac

$./bin/plugin -DproxyHost=host\_name -DproxyPort=port\_number \

--install mobz/elasticsearch-head

//On Windows

$ set JAVA\_OPTS="-DproxyHost=host\_name -DproxyPort=port\_number"

$.bin/plugin --install mobz/elasticsearch-head

./bin/elasticsearch -d //run elasticsearch

curl 'http://localhost:9200/?pretty' //client

curl -XPOST 'http://localhost:9200/\_shutdown' //shut down elasticsearch

## 与Elasticsearch交互

Java API

节点客户端(node client)：节点客户端以无数据节点(none data node)身份加入集群，换言之，它自己不存储任何数据，但是它知道数据在集群中的具体位置，并且能够直接转发请求到对应的节点上。

传输客户端(Transport client)：这个更轻量的传输客户端能够发送请求到远程集群。它自己不加入集群，只是简单转发请求给集群中的节点。

两个Java客户端都通过9300端口与集群交互，使用Elasticsearch传输协议(Elasticsearch Transport Protocol)。集群中的节点之间也通过9300端口进行通信。

基于HTTP协议，以JSON为数据交互格式的RESTful API

其他所有程序语言都可以使用RESTful API，通过9200端口的与Elasticsearch进行通信

curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY\_STRING>' -d '<BODY>'

VERB HTTP方法：GET, POST, PUT, HEAD, DELETE

PROTOCOL http或者https协议（只有在Elasticsearch前面有https代理的时候可用）

HOST Elasticsearch集群中的任何一个节点的主机名，如果是在本地的节点，那么就叫localhost

PORT Elasticsearch HTTP服务所在的端口，默认为9200

PATH API路径（例如\_count将返回集群中文档的数量），PATH可以包含多个组件，例如\_cluster/stats或者\_nodes/stats/jvm

QUERY\_STRING 一些可选的查询请求参数，例如?pretty参数将使请求返回更加美观易读的JSON数据

BODY 一个JSON格式的请求主体（如果请求需要的话）

Elasticsearch是面向文档(document oriented)的，这意味着它可以存储整个对象或文档(document)。然而它不仅仅是存储，还会索引(index)每个文档的内容使之可以被搜索。在Elasticsearch中，你可以对文档（而非成行成列的数据）进行索引、搜索、排序、过滤

**Relational DB -> Databases -> Tables -> Rows -> Columns**

**Elasticsearch -> Indices -> Types -> Documents -> Fields**

curl -i -XPUT 'http://localhost:9200/megacorp/employee/1?pretty' -d '{}' 插入文档

curl -i -XGET 'http://localhost:9200/megacorp/employee/1?pretty' 获取文档

搜索全部:默认情况下搜索会返回前10个结果

curl -i -XGET 'http://localhost:9200/megacorp/employee/\_search?pretty'

### 查询字符串(query string)搜索

curl -i -XGET 'http://localhost:9200/megacorp/employee/\_search?pretty&q=last\_name:Smith'

### 使用DSL语句查询

不再使用查询字符串(query string)做为参数，而是使用请求体代替

curl -i -XGET 'http://localhost:9200/megacorp/employee/\_search?pretty' -d

'{

"query": {

"match": {

"last\_name": "Smith"

}

}

}'

找到姓氏为“Smith”的员工，and年龄大于30岁的员工

GET /megacorp/employee/\_search

{

"query" : {

"filtered" : {

"filter" : {

"range" : {

"age" : { "gt" : 30 } <1>

}

},

"query" : {

"match" : {

"last\_name" : "smith" <2>

}

}

}

}

}

### 全文搜索

默认情况下，Elasticsearch根据结果相关性评分来对结果集进行排序，所谓的「结果相关性评分」就是文档与查询条件的匹配程度.相关性(relevance)的概念在Elasticsearch中非常重要，而这个概念在传统关系型数据库中是不可想象的，因为传统数据库对记录的查询只有匹配或者不匹配。

GET /megacorp/employee/\_search

{

"query" : {

"match" : { //匹配rock or climbing

"about" : "rock climbing"

}

}

}

GET /megacorp/employee/\_search

{

"query" : {

"match\_phrase" : { //匹配rock climbing

"about" : "rock climbing"

}

}

}

//从每个搜索结果中高亮(highlight)匹配到的关键字，这样用户可以知道为什么这些文档和查询相匹配

GET /megacorp/employee/\_search

{

"query" : {

"match\_phrase" : {

"about" : "rock climbing"

}

},

"highlight": {

"fields" : {

"about" : {}

}

}

}

### 分析

Elasticsearch有一个功能叫做聚合(aggregations)，它允许你在数据上生成复杂的分析统计

聚合也允许分级汇总

统计每种兴趣下职员的平均年龄

GET /megacorp/employee/\_search

{

"aggs" : {

"all\_interests" : {

"terms" : { "field" : "interests" },

"aggs" : {

"avg\_age" : {

"avg" : { "field" : "age" }

}

}

}

}

}

## 集群内部工作方式

集群和节点

节点(node)是一个运行着的Elasticsearch实例。集群(cluster)是一组具有相同cluster.name的节点集合，他们协同工作，共享数据并提供故障转移和扩展功能

你最好找一个合适的名字来替代cluster.name的默认值，比如你自己的名字，这样可以防止一个新启动的节点加入到相同网络中的另一个同名的集群中。

Elasticsearch用于构建高可用和可扩展的系统。扩展的方式可以是购买更好的服务器(纵向扩展(vertical scale or scaling up))或者购买更多的服务器（横向扩展(horizontal scale or scaling out)）。

对于大多数数据库而言，横向扩展意味着你的程序将做非常大的改动才能利用这些新添加的设备。对比来说，Elasticsearch天生就是分布式的：它知道如何管理节点来提供高扩展和高可用。这意味着你的程序不需要关心这些。

做为用户，我们能够与集群中的任何节点通信，包括主节点。每一个节点都知道文档存在于哪个节点上，它们可以转发请求到相应的节点上。我们访问的节点负责收集各节点返回的数据，最后一起返回给客户端。这一切都由Elasticsearch处理。

curl -XGET 'http://localhost:9200/\_cluster/health' //集群健康

集群健康有三种状态：green、yellow或red。

green 所有主要分片和复制分片都可用

yellow 所有主要分片可用，但不是所有复制分片都可用

red 不是所有的主要分片都可用

为了将数据添加到Elasticsearch，我们需要索引(index)——一个存储关联数据的地方。实际上，索引只是一个用来指向一个或多个分片(shards)的“逻辑命名空间(logical namespace)”.

分片就是一个Lucene实例，并且它本身就是一个完整的搜索引擎。我们的文档存储在分片中，并且在分片中被索引，但是我们的应用程序不会直接与它们通信，取而代之的是，直接与索引通信。

你索引中的每个文档属于一个单独的主分片，所以主分片的数量决定了索引最多能存储多少数据。

复制分片只是主分片的一个副本，它可以防止硬件故障导致的数据丢失，同时可以提供读请求，比如搜索或者从别的shard取回文档。

当索引创建完成的时候，主分片的数量就固定了，但是复制分片的数量可以随时调整。

默认情况下，一个索引被分配5个主分片

调整：分配3个主分片和一个复制分片（每个主分片都有一个复制分片）

PUT /blogs

{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

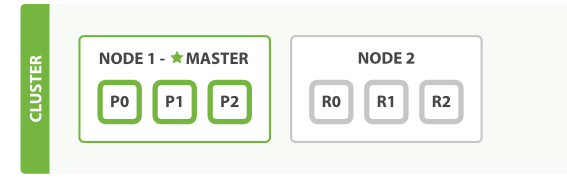
}

}

集群的健康状态yellow表示所有的主分片(primary shards)启动并且正常运行了——集群已经可以正常处理任何请求——但是复制分片(replica shards)还没有全部可用。事实上所有的三个复制分片现在都是unassigned状态——它们还未被分配给节点。在同一个节点上保存相同的数据副本是没有必要的，如果这个节点故障了，那所有的数据副本也会丢失。

增加故障转移

只要第二个节点与第一个节点有相同的cluster.name（请看./config/elasticsearch.yml文件），它就能自动发现并加入第一个节点所在的集群。如果没有，检查日志找出哪里出了问题。这可能是网络广播被禁用，或者防火墙阻止了节点通信。

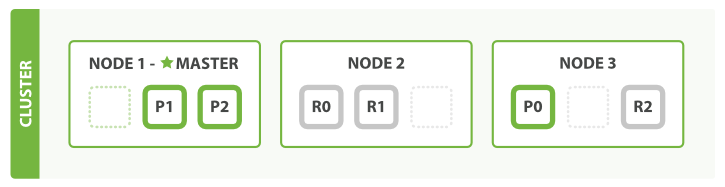


文档的索引将首先被存储在主分片中，然后并发复制到对应的复制节点上。这可以确保我们的数据在主节点和复制节点上都可以被检索。

curl -XGET 'http://localhost:9200/\_cluster/health' //集群健康

横向扩展

启动第三个节点，我们的集群会重新组织自己



Node3包含了分别来自Node 1和Node 2的一个分片，这样每个节点就有两个分片，和之前相比少了一个，这意味着每个节点上的分片将获得更多的硬件资源（CPU、RAM、I/O）。

分片本身就是一个完整的搜索引擎，它可以使用单一节点的所有资源。我们拥有6个分片（3个主分片和三个复制分片），最多可以扩展到6个节点，每个节点上有一个分片，每个分片可以100%使用这个节点的资源。

复制分片的数量可以在运行中的集群中动态地变更，这允许我们可以根据需求扩大或者缩小规模。让我们把复制分片的数量从原来的1增加到2：

PUT /blogs/\_settings

{

"number\_of\_replicas" : 2

}

