**ElasticSearch学习使用手册**

**一．介绍**

ElasticSearch是一个基于Lucene的分布式可扩展的开源搜索分析引擎，它能帮你搜索,分析和浏览数据。其使用Java开发并将Lucene作为其核心来实现所有索引和搜索的功能呢个，但是它通过简单的API来隐藏Lucene的复杂性，从而让全文搜索变得更简单。

**1.网络结构**

**1.1 Cluster集群:**

Cluster集群是有相同集群名称的节点Node的集合，集群中有多个节点，其中有一个为主节点，这个主节点是可以通过选举产生的，主从节点是对于集群内部来说的。es的一个概念就是去中心化，字面上理解就是无中心节点，这是对于集群外部来说的，因为从外部来看es集群，在逻辑上是个整体，你与任何一个节点的通信和与整个es集群通信是等价的。

**1.2 Node节点：**

Node节点上有一个ElasticSearch运行的实例，其实就是一个java进程。一般情况下，一个ElasticSearch进程运行在一台机器上。

 一个节点可以通过配置集群名称的方式来加入一个指定的集群。默认情况下，每个节点都会被安排加入到一个叫做“elasticsearch”的集群中，这意味着，如果你在你的网络中启动了若干个节点，并假定它们能够相互发现彼此，它们将会自动地形成并加入到一个叫做“elasticsearch”的集群中。而且，如果当前你的网络中没有运行任何Elasticsearch节点，这时启动一个节点，会默认创建并加入一个叫做“elasticsearch”的集群。

节点分为client，data，master三种类型。

**2.组织结构**

**2.1 Index索引**

Elasticsearch中用来存储数据的逻辑区域，它类似于一个拥有相似特征的文档的集合，类似于关系型数据库中的table概念。一个索引有一个名字来标识（小写字母），当我们对对应于这个索引内的文档进行搜索，更新，删除时都会用到这个索引。一个index可以在一个或者多个shard上面，同时一个shard也可能会有多个replicas。

**2.2 分片（shards）**

一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有10亿文档的索引占据1TB的磁盘空间，而任一节点都没有这样大的磁盘空间；或者单个节点处理搜索请求，响应太慢。

为了解决这个问题，Elasticsearch提供了将索引划分成多份的能力，这些份就叫做分片。es可以把一个完整的索引分成多个分片，这样的好处是可以把一个大的索引拆分成多个，分布到不同的节点上，构成分布式搜索。分片的数量只能在索引创建前指定，并且索引创建后不能更改

优点：- 允许你水平分割/扩展你的内容容量  
          - 允许你在分片（潜在地，位于多个节点上）之上进行分布式的、并行的操作，进而提高性能/吞吐量

**2.3 复制（replicas）**

在一个网络/云的环境里，失败随时都可能发生，在某个分片/节点不知怎么的就处于离线状态，或者由于任何原因消失了，这种情况下，有一个故障转移机制是非常有用并且是强烈推荐的。为此目的，Elasticsearch允许你创建分片的一份或多份拷贝，这些拷贝叫做复制分片，或者直接叫复制。  
 复制之所以重要，有两个主要原因：  
        - 在分片/节点失败的情况下，提供了高可用性。因此，复制分片从不与原分片置于同一节点上。  
        - 扩展你的搜索量/吞吐量，因为搜索可以在所有的复制上并行运行  
    总之，每个索引可以被分成多个分片，也可以被复制0次（没有复制）或多次。一旦复制了，每个索引就有了主分片（作为复制源的分片）和复制分片（主分片的拷贝）之别。分片和复制的数量可以在索引创建的时候指定，索引创建之后，你可以在任何时候动态地改变复制的数量，但是你事后不能改变分片的数量。  
 默认情况下，Elasticsearch中的每个索引被分片5个主分片和1个复制，这意味着，如果你的集群中至少有两个节点，你的索引将会有5个主分片和另外5个复制分片（1个完全拷贝），这样的话每个索引总共就有10个分片。

**3.数据存储**

**3.1 Document**

Elasticsearch里面存储的实体数据，类似于关系数据中一个table里面的一行数据。 document由多个field组成，不同的document里面同名的field一定具有相同的类型，document里面field可以重复出现，也就是一个field会有多个值，即multivalued。

**3.2 Document type**

为了查询需要，一个index可能会有多种document，也就是document type，但需要注意，不同document里面同名的field一定要是相同类型的。

**3.3 Mapping**

存储field的相关映射信息，不同document type会有不同的mapping。

**4.相关概念**

**4.1 恢复（重新分布）**

代表数据恢复或叫数据重新分布，es在有节点加入或退出时会根据机器的负载对索引分片进行重新分配，挂掉的节点重新启动时也会进行数据恢复。

**4.2 Gateway**

代表es索引快照的存储方式，es默认是先把索引存放到内存中，当内存满了时再持久化到本地硬盘。gateway对索引快照进行存储，当这个es集群关闭再重新启动时就会从gateway中读取索引备份数据。 es支持多种类型的gateway，有本地文件系统（默认），分布式文件系统，Hadoop的HDFS和amazon的s3云存储服务。

**4.3 Discovery.zen**

代表es的自动发现节点机制，es是一个基于p2p的系统，它先通过广播寻找存在的节点，再通过多播协议来进行节点之间的通信，同时也支持点对点的交互。

**4.4 Transport**

代表es内部节点或集群与客户端的交互方式，默认内部是使用tcp协议进行交互，同时它支持http协议（json格式）、thrift、servlet、memcached、zeroMQ等的传输协议（通过插件方式集成）。

1. **安装及使用**

https://www.elastic.co/guide/cn/index.html

**1.Ubuntu下安装ElasticSearch**

ES依赖于Java，请确保已安装JDK

1)下载并解压方式安装：$wget<https://download.elasticsearch.org/elasticsearch/elasticsearch/elasticsearch-1.3.4.tar.gz> （获取tar包或者去elastic.co官网下载）

$ tar zxvf elasticsearch-1.3.4.tar.gz -C /usr/localelasticsearch使用默认配置即可，默认的cluster name为：elasticsearch；

$ ./bin/elasticsearch -d (用默认参数在后台启动，停止的话只能kill进程)

备用：$ ./bin/elasticsearch (用默认参数在前台启动，ctrl+c即可停止)启动成功后访问本地9200端口即可看到elasticsearch相关信息。

2)命令行安装（未实践，不常用且不确保可行）：

$ wget -qO - https://packages.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add - echo "deb <http://packages.elastic.co/elasticsearch/1.6/>

debian stable main" | sudo tee -a /etc/apt/sources.list

$ sudo apt-get update

$ sudo apt-get install elasticsearch

$ sudo update-rc.d elasticsearch defaults 95 10

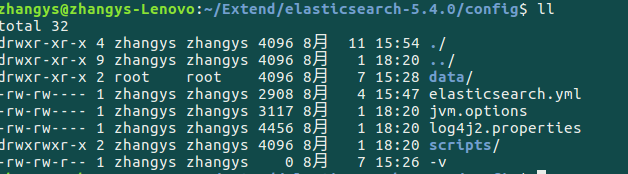
**2.elasticSearch使用及参数配置**

**2.1 elasticsearch 启动参数**

./elasticsearch --cluster.name my\_cluster\_name --node.name my\_node\_name （本地启动elasticsearch时指定集群名称和当前node名称）

**2.2 elasticsearch.yml 配置文件**

在config/目录下有一个elasticsearch.yml文件，如下图



**cluster.name:** elasticsearch 配置es的集群名称，默认是elasticsearch，es会自动发现在同一网段下的es，如果在同一网段下有多个集群，就可以用这个属性来区分不同的集群。

**node.name:** 当前节点名，默认随机指定一个name列表中名字.

**node.master: true** 指定该节点是否有资格被选举成为master，默认是true，es是默认集群中的第一台机器为master，如果这台机挂了就会重新选举master。

node.data: true 指定该节点是否存储索引数据，默认为true。

index.number\_of\_shards: 5 设置默认索引分片个数，默认为5片。

index.number\_of\_replicas: 1 设置默认索引副本个数，默认为1个副本。

**path.data:** /path/to/data 设置索引数据的存储路径，默认是es根目录下的data文件夹，可以设置多个存储路径，用逗号隔开，例： path.data: /path/to/data1,/path/to/data2

path.work: /path/to/work 设置临时文件的存储路径，默认是es根目录下的work文件夹。

path.logs: /path/to/logs 设置日志文件的存储路径，默认是es根目录下的logs文件夹

path.plugins: /path/to/plugins 设置插件的存放路径，默认是es根目录下的plugins文件夹

bootstrap.mlockall: true 设置为true来锁住内存。因为当jvm开始swapping时es的效率会降低，所以要保证它不swap，可以把ES\_MIN\_MEM和 ES\_MAX\_MEM两个环境变量设置成同一个值，并且保证机器有足够的内存分配给es。同时也要允许elasticsearch的进程可以锁住内存，linux下可以通过`ulimit -l unlimited`命令。

network.bind\_host: 192.168.0.1 设置绑定的ip地址，可以是ipv4或ipv6的，默认为0.0.0.0。

network.publish\_host: 192.168.0.1 设置其它节点和该节点交互的ip地址，如果不设置它会自动判断，值必须是个真实的ip地址。

**network.host:** 192.168.0.1 这个参数是用来同时设置bind\_host和publish\_host上面两个参数。

**transport.tcp.port: 9300** 设置节点间交互的tcp端口，默认是9300。

transport.tcp.compress: true 设置是否压缩tcp传输时的数据，默认为false，不压缩。

**http.port: 9200** 设置对外服务的http端口，默认为9200。 http.max\_content\_length: 100mb 设置内容的最大容量，默认100mb

**http.enabled: false** 是否使用http协议对外提供服务。

gateway.type: local gateway的类型，默认为local即为本地文件系统，可以设置为本地文件系统，分布式文件系统，Hadoop的HDFS，和amazon的s3服务器。

gateway.recover\_after\_nodes: 1 设置集群中N个节点启动时进行数据恢复，默认为1。

gateway.recover\_after\_time: 5m 设置初始化数据恢复进程的超时时间，默认是5分钟。

gateway.expected\_nodes: 2 设置这个集群中节点的数量，默认为2，一旦这N个节点启动，就会立即进行数据恢复。

cluster.routing.allocation.node\_initial\_primaries\_recoveries: 4 初始化数据恢复时，并发恢复线程的个数，默认为4。

cluster.routing.allocation.node\_concurrent\_recoveries: 2 添加删除节点或负载均衡时并发恢复线程的个数，默认为4。

**discovery.zen.minimum\_master\_nodes:** 1 当集群中的节点个数大于这个数字时才选举master节点。默认为1，对于大的集群来说，可以设置大一点的值（2-4）

Discovery.zen.ping\_timeout: 3s 设置集群中自动发现其它节点时ping连接超时时间，默认为3秒，对于比较差的网络环境可以高点的值来防止自动发现时出错。

discovery.zen.ping.multicast.enabled: false (5.0后不存在了)设置是否打开多播发现节点，默认是true。

**discovery.zen.ping.unicast.hosts:** ["host1", "host2:port", "host3[portX-portY]"] 设置集群中master节点的初始列表，可以通过这些节点来自动发现新加入集群的节点。

**3.REST API**

**3.1 api功能**

- 检查你的集群、节点和索引的健康状态、和各种统计信息  
 - 管理你的集群、节点、索引数据和元数据  
      - 对你的索引进行CRUD（创建、读取、更新和删除）和搜索操作  
      - 执行高级的查询操作，像是分页、排序、过滤、脚本编写（scripting）、小平面刻画（faceting）、聚合（aggregations）和许多其它操作

**3.2 常用api**

**运维类命令：**

节点数量查看 curl -XGET 'http://localhost:9200/\_count?pretty'

Node列表查看 curl 'http://localhost:9200/\_cat/nodes?v'

健康检查 curl 'http://localhost:9200/\_cat/health?v'

**索引类命令：**

列举所有索引 curl 'http://localhost:9200/\_cat/indices?v'

创建索引 curl -XPUT 'http://localhost:9200/customer?pretty'

删除索引 curl -XDELETE 'http://localhost:9200/customer?pretty'

**文档类命令：**

创建文档 curl -XPOST 'http://localhost:9200/twitter/user/1?pretty' -d '

> { “name” ： “John”} '

Twitter代表索引，user代表ducument type，1代表id

更新文档 curl -XPOST 'http://localhost:9200/twitter/user/1/\_update?pretty' -d '

> {“doc” ： { “name” ： “Tom”, “age” : 31}} '

删除文档 curl -XDELETE 'http://localhost:9200/twitter/user/[文档id]？Pretty

**查看类命令：**

查看索引内数据 curl -XGET 'http://localhost:9200/twitter/\_search?Pretty

条件索引数据 curl -XPOST 'localhost:9200/twitter/\_search?pretty' -d '

> {

> "query":{"match\_phrase":{"account\_number":20}}

> }'

match\_phrase表示严格匹配，match表示包含匹配

**批处理操作：**

索引两个文档curl -XPOST 'localhost:9200/customer/external/\_bulk?pretty' -d'

> {"index":{"\_id":"6"}}

> {"name":"John Doe"}

> {"index":{"\_id":"7"}}

> {"name":"Jane Doe"}

> '

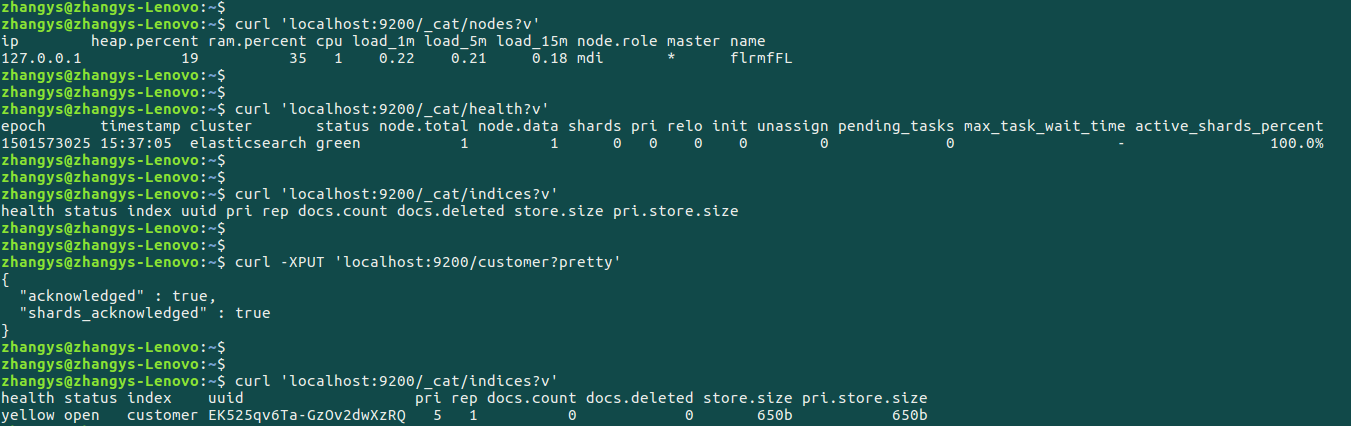
更新然后删除curl -XPOST 'localhost:9200/customer/external/\_bulk?pretty' -d'

> {"update":{"\_id":"6"}}

> {"doc":{"name":"John Doe becomes David.liu"}}

> {"delete":{"\_id":"7"}}

> '



**4.elasticsearch-head插件**

**4.1介绍**

该插件提供了对elasticsearch进行操作的界面

**4.2 安装及使用**

**参考：http://blog.csdn.net/napoay/article/details/53896348**

下载5.0以后版本的tar包，解压

安装 node

安装 npm

进入elasticsearch-head文件夹运行npm-install

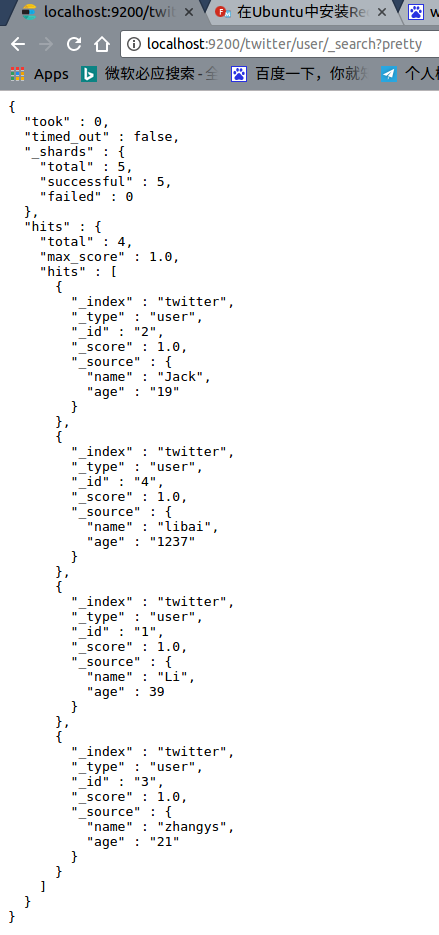
修改Gruntfile.js文件，添加ip地址

**或者**

直接去dockerhub上下载elasticsearch-head镜像启动即可

1. **Go SDK（elastic）**

下图为elasticsearch内数据格式（用curl命令手动添加的数据）



以下是一个go代码通过es sdk操作es的示例（在https://github.com/olivere/elastic基础上稍加修改）

package main

import (

"fmt"

"time"

"encoding/json"

elastic "gopkg.in/olivere/elastic.v5"

"golang.org/x/net/context"

)

type Member struct {

Name string

Age string

}

func main(){

fmt.Println("开始工作")

for a := 0; a<6; a++{

time.Sleep(150000000)

print(" - ")

}

println()

ctx := context.Background()

//建立连接

client, err := elastic.NewClient()

if err != nil {

panic(err) //抛出异常

}

//创建索引

// \_表示忽略这个返回值

/\*

\_, err = client.CreateIndex("index1").Do(ctx)

if err != nil {

panic(err)

}

\*/

//添加document

/\*

memberA := `{ "name": "libai", "age": "1237" }`

fmt.Println(memberA)

\_,err = client.Index().

Index("twitter").

Type("user").

Id("4").

BodyJson(memberA).

Refresh("true").

Do(ctx)

if err != nil {

println(err)

panic(err)

}

\*/

//查找

termQuery := elastic.NewTermQuery("name", "jack")

// termQuery := `{"query":{"match":{"name":"Li"}}}`

searchResult,errs := client.Search().

Index("twitter"). //制定index

Query(termQuery). //指定查询语句

// Sort("name", true).//根据field分类

From(0).Size(10). //获取documents 0-9

Pretty(true).

Do(ctx)

if errs != nil {

panic(errs)

}

fmt.Printf("查询花费 %d 秒\n", searchResult.TookInMillis)

fmt.Printf("共查询到 %d 条数据\n",searchResult.Hits.TotalHits)

if searchResult.Hits.TotalHits > 0 {

for \_, hit := range searchResult.Hits.Hits {

var member Member

err = json.Unmarshal(\*hit.Source, &member)

if err != nil { fmt.Printf("转化异常") }

fmt.Printf("result:%s\n", \*hit.Source)

fmt.Printf("result(name) %s\n", member.Name)

}

}else{

fmt.Println("无结果")

}

}