Elasticsearch Head Kibana IK Logstash

ONE 2022.03.20

目录

一. ElasticSearch 概述

1. ElasticSearch 介绍

ES 是一个开源的高扩展的分布式全文搜索引擎，是整个 Elastic Stack 技术栈的核心。它可以近乎实时的存储，检索数据；本身扩展性很好，可以扩展到上百台服务器，处理 PB 级别的数据。

ElasticSearch 的底层是开源库 Lucene，但是没办法直接用 Lucene，必须自己写代码去调用它的接口，Elastic 是 Lucene 的封装，提供了 REST API 的操作接口，开箱即用。天然的跨平台。

全文检索是在实际项目开发中最常见的需求了，而 ElasticSearch 是目前全文检索引擎的首选，它可以快速的存储，搜索和分析海量的数据，维基百科，GitHub，Stack Overflow 都采用了 ElasticSearch。

2. ElasticSearch 用途

(1) 搜索的数据对象是大量的非结构化的文本数据。
(2) 文件记录达到数十万或数百万个甚至更多。
(3) 支持大量基于交互式文本的查询。
(4) 需求非常灵活的全文搜索查询。
(5) 对高度相关的搜索结果的有特殊需求，但是没有可用的关系数据库可以满足。
(6) 对不同记录类型，非文本数据操作或安全事务处理的需求相对较少的情况。

3. ElasticSearch 基本概念

3.1 索引

索引(indices)可以理解为是文档的集合，同在一个索引中的文档共同建立倒排索引。会把索引类比于 MySQL 中 schema 的概念，但在 ES 中 indices 更加灵活，用起来也更加方便。此外，提交给同一个索引中的文档，最好拥有相同的结构。这样对于 ES 来说，不管是存储还是查询，都更容易优化。

3.2 类型

类型(Type),对应的其实就是数据库中的 Table(数据表)，类型是模拟 mysql 中的 table 概念，一个索引库下可以有不同类型的索引，比如商品索引，订单索引，其数据格式不同。

但是从 Elasticsearch7.X 移除类型(type)。因为 Elasticsearch 设计初期，是直接查考了关系型数据库的设计模式，存在了 type(数据表)的概念。但是，其搜索引擎是基于 Lucene 的，这种 “基因”决定了 type 是多余的。 Lucene 的全文检索功能之所以快，是因为 倒序索引 的存在。而这种倒序索引的生成是基于 index 的，而并非 type。多个 type 反而会减慢搜索的速度。为了保持 Elasticsearch “一切为了搜索” 的宗旨，适当的做些改变(去除 type)也是无可厚非的，也是值得的。

3.3 文档

文档(Document),对应的就是具体数据行(Row)。需要注意的是，这里的文档并非指的是一个纯字符串文本，在 ES 中文档指的是一条 JSON 数据。如果对 MongoDB 有了解的话，这里文档的含义和 MongoDB 中的基本类似。

JSON 数据中可以包含多个字段，这些字段可以类比为 MySQL 中每个表

的字段。

例如:

```
{
  "message": "this is my blog",
  "author": "cyhone"
}
```

后期进行搜索和查询的时候，也可以分别针对 message 字段和 author 字段进行搜索。

3.4 字段

每个 Document 都类似一个 JSON 结构，它包含了许多字段，每个字段都有其对应的值，多个字段组成了一个 Document，可以类比关系型数据库数据表中的字段。 在 Elasticsearch 中，文档（Document）归属于一种类型（Type），而这些类型存在于索引（Index）中，下图展示了 Elasticsearch 与传统关系型数据库的类比：

```
Relational DB  ⇒ Databases ⇒ Tables ⇒ Rows      ⇒ Columns
Elasticsearch  ⇒ Indices   ⇒ Types  ⇒ Documents ⇒ Fields
```

3.5 映射

映射(mapping)是处理数据的方式和规则方面做一些限制，如：某个字段的数据类型、默认值、分析器、是否被索引等等。这些都是映射里面可以设置的，其它就是处理 ES 里面数据的一些使用规则设置也叫做映射，按着最优规则处理数据对性能提高很大，因此才需要建立映射，并且需要思考如何建立映射才能对性能更好。

3.6 分片

一个索引可以存储超出单个节点硬件限制的大量数据。比如，一个具有 10 亿文档数据的索引占据 1TB 的磁盘空间，而任一节点都可能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，每一份就称之为分片。当创建一个索引的时候，可以指定想要的分片的数量。每个分片本身也是一个功能完善并且独立的“索引”，这个“索引”可以被放置到集群中的任何节点上。

分片很重要，主要有两方面的原因：

1）允许水平分割 / 扩展的内容容量。

2）允许在分片之上进行分布式的、并行的操作，进而提高性能/吞吐量。

至于一个分片怎样分布，它的文档怎样聚合和搜索请求，是完全由 Elasticsearch 管理的，对于作为用户的来说，这些都是透明的，无需过分关心。

3.7 副本

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 允许创建分片的一份或多份拷贝，这些拷贝叫做复制分片(副本)。

复制分片之所以重要，有两个主要原因：

(1) 在分片/节点失败的情况下，提供了高可用性。因为这个原因，注意到复制分片从不与原/主要（original/primary）分片置于同一节点上是非常重要的。

(2) 扩展的搜索量/吞吐量，因为搜索可以在所有的副本上并行运行。

总之，每个索引可以被分成多个分片。一个索引也可以被复制 0 次（意思是没有复制）或多次。一旦复制了，每个索引就有了主分片（作为复制源的原来的分片）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。在索引创建之后，可以在任何时候动态地改变复制的数量，但是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 1 个主分片和 1 个复制，这意味着，如果集群中至少有两个节点，索引将会有 1 个主分片和另外 1 个复制分片（1 个完全拷贝），这样的话每个索引总共就有 2 个分片，需要根据索引需要确定分片个数。

3.8 倒排索引

Elasticsearch 是通过 Lucene 的倒排索引技术实现比关系型数据库更快的过滤。特别是它对多条件的过滤支持非常好。倒排索引是搜索引擎的核心。搜索引擎的主要目标是在查找发生搜索条件的文档时提供快速搜索。

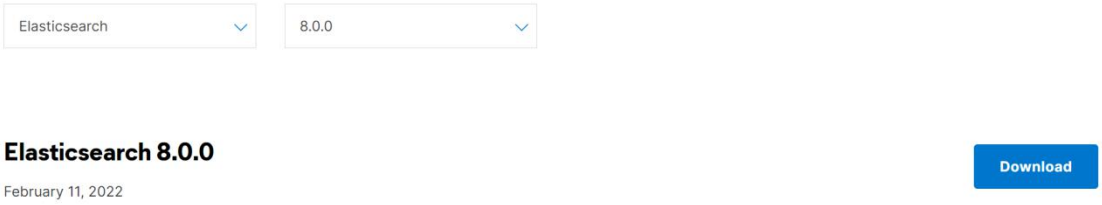
ES 中的倒排索引其实就是 lucene 的倒排索引，区别于传统的正向索引，倒排索引会再存储数据时将关键词和数据进行关联，保存到倒排表中，然后查询时，将查内容进行分词后在倒排表中进行查询，最后匹配数据即可。

二． ElasticSearch 安装

Ubuntu：18.04.6 LTS
ip：192.168.81.100 ,192.168.81.101

1. ElasticSearch 下载

版本：8.0.0
下载地址：https://www.elastic.co/cn/downloads/past-releases#elasticsearch



解压目录 /usr/local

2. ElasticSearch 配置

(1) ElasticSearch 不能以 root 进行启动，为其添加用户权限。

sudo chown -R xxx /usr/local/elasticsearch-8.0.0

(2) 配置 JDK

Elasticsearch 是使用 java 开发的，且高版本的 ES 需要 JDK 版本 1.8 以上，默认安装包带有 jdk 环境，如果系统配置 JAVA_HOME，那么使用系统默认的 JDK，如果没有配置使用自带的 JDK。

安装包自带 JDK 配置：

/usr/local/elasticsearch-8.0.0/bin/elasticsearch-env 增加 JDK 即可

```
JAVA_HOME="/usr/local/elasticsearch-8.0.0/jdk"
```

(3) 修改用户拥有的内存权限

/etc/sysctl.conf 文件最后添加一行

```
vm.max_map_count=655360
```

执行 sysctl -p 使其生效

(4) 线程数修改

/etc/security/limits.conf 最后一行添加

```
* soft nofile 65536
* hard nofile 65536
* soft nproc 32000
* hard nproc 32000
* hard memlock unlimited
* soft memlock unlimited
```

执行 sysctl -p 使其生效

(5) ElasticSearch 集群配置

/usr/local/elasticsearch-8.0.0/config/elasticsearch.yml

```
# ---------------------------------- Cluster -----------------------------------
#es 集群名称，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
#识别集群的标识，同一个集群名字必须相同
cluster.name: my-elasticsearch
# ------------------------------------ Node ------------------------------------
#该节点名称，自定义或者默认
node.name: node-1
#该节点是否可以成为一个 master 节点
node.master: true
#该节点是否存储数据，即是否是一个数据节点，默认 true
node.data: true
#节点的通用属性，用于后期集群进行碎片分配时的过滤
node.attr.rack: r1
# ---------------------------------- Paths -----------------------------------
#配置文件路径，默认 es 安装目录下的 config
path.conf: /path/to/conf
#数据存储路径，默认 es 安装目录下的 data
#可以设置多个存储路径，用逗号隔开
path.data: /usr/local/elasticsearch-8.0.0/data
#日志路径，默认 es 安装目录下的 logs
path.logs: /usr/local/elasticsearch-8.0.0/log
# ---------------------------------- Memory -----------------------------------
#当 JVM 开始写入交换空间时（swapping）ElasticSearch 性能会低下
#设置为 true 来锁住内存,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
bootstrap.memory_lock: true
# ---------------------------------- Network -----------------------------------
#该节点绑定的地址，即对外服务的地址，可以是 IP，主机名
network.host: 0.0.0.0
#该节点对外服务的 http 端口，默认 9200
http.port: 9200
#节点间交互的 tcp 端口,默认 9300
transport.tcp.port: 9300
#HTTP 请求的最大内容，默认 100MB
http.max_content_length: 100MB
#HTTP URL 的最大长度，默认为 4KB
http.max_initial_line_length: 4KB
#允许的标头的最大大小，默认为 8KB
http.max_header_size: 8KB
```

```
#压缩，默认 true
http.compression: true
#压缩级别，有效值:1-9，默认为 3
http.compression_level: 3
#是否开启 http 协议对外提供服务,默认为 true
http.enabled: true
# ---------------------------------- Discovery ----------------------------------
#集群列表
#port 为节点间交互端口，未设置时，默认 9300
discovery.seed_hosts: ["host1:port", "ip2:port"]
#初始主节点列表
cluster.initial_master_nodes: ["node-1", "node-2"]
# ---------------------------------- Gateway -----------------------------------
#gateway 的类型,默认为 local，即为本地文件系统
gateway.type: local
#集群中的 N 个节点启动后,才允许进行恢复处理，默认 3
gateway.recover_after_nodes: 3
#设置初始化恢复过程的超时时间,超时时间从上一个配置中配置的 N 个节点启动后算起
gateway.recover_after_time: 5m
#设置这个集群中期望有多少个节点，一旦这 N 个节点启动，立即开始恢复过程
gateway.expected_nodes: 2
# ---------------------------------- Various -----------------------------------
#删除索引时需要显式名称
action.destructive_requires_name: true
#跨域配置
#action.destructive_requires_name: true
http.cors.enabled: true
http.cors.allow-origin: "*"
```

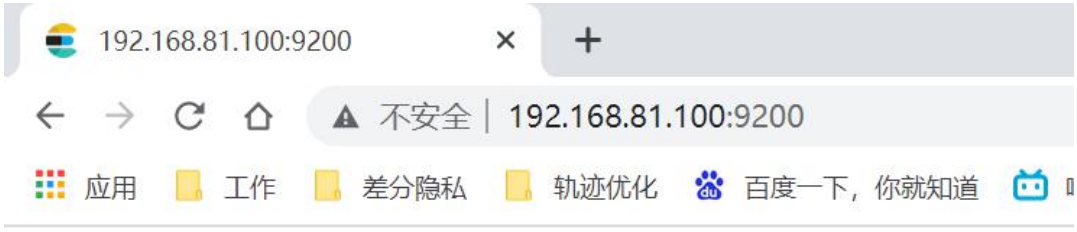
3. ElasticSearch 启动

```
4 bit): Version 8.0.0 (Build 5e85495ea85316) Copyright (c) 2022 Elasticsearch BV
[2022-03-15T09:30:15,400][INFO ][o.e.x.s.Security         ] [node-1] Security is disabled
[2022-03-15T09:30:18,904][INFO ][o.e.t.n.NettyAllocator   ] [node-1] creating NettyAllocator with the following configs:
[name=elasticsearch_configured, chunk_size=1mb, suggested_max_allocation_size=1mb, factors={es.unsafe.use_netty_default_c
hunk_and_page_size=false, g1gc_enabled=true, g1gc_region_size=4mb}]
[2022-03-15T09:30:19,070][INFO ][o.e.d.DiscoveryModule    ] [node-1] using discovery type [zen] and seed hosts providers
[settings]
[2022-03-15T09:30:20,946][INFO ][o.e.n.Node               ] [node-1] initialized
[2022-03-15T09:30:20,947][INFO ][o.e.n.Node               ] [node-1] starting ...
[2022-03-15T09:30:20,997][INFO ][o.e.x.s.c.f.PersistentCache] [node-1] persistent cache index loaded
[2022-03-15T09:30:20,999][INFO ][o.e.x.d.l.DeprecationIndexingComponent] [node-1] deprecation component started
[2022-03-15T09:30:21,332][INFO ][o.e.t.TransportService   ] [node-1] publish_address {192.168.81.100:9300}, bound_address
es {192.168.81.100:9300}
[2022-03-15T09:30:22,243][INFO ][o.e.b.BootstrapChecks    ] [node-1] bound or publishing to a non-loopback address, enfor
cing bootstrap checks
[2022-03-15T09:30:22,245][INFO ][o.e.c.c.Coordinator      ] [node-1] cluster UUID [BFFAf3EBRImVEGph9pqfiA]
[2022-03-15T09:30:23,307][INFO ][o.e.c.s.ClusterApplierService] [node-1] master node changed {previous [], current [{node
-2}{Y8UiA93DTq-F1IkQHXAOvg}{Xv99ivZQRzGHOPO_NCG5vQ}{192.168.81.101}{192.168.81.101:9300}{cdfhilmrstw}]}, added {{node-2}{
Y8UiA93DTq-F1IkQHXAOvg}{Xv99ivZQRzGHOPO_NCG5vQ}{192.168.81.101}{192.168.81.101:9300}{cdfhilmrstw}}, term: 41, version: 20
11, reason: ApplyCommitRequest{term=41, version=2011, sourceNode={node-2}{Y8UiA93DTq-F1IkQHXAOvg}{Xv99ivZQRzGHOPO_NCG5vQ}
{192.168.81.101}{192.168.81.101:9300}{cdfhilmrstw}{ml.machine_memory=2047352832, xpack.installed=true, ml.max_jvm_size=10
23410176}}
[2022-03-15T09:30:23,405][INFO ][o.e.h.AbstractHttpServerTransport] [node-1] publish_address {192.168.81.100:9200}, bound
_addresses {192.168.81.100:9200}
[2022-03-15T09:30:23,406][INFO ][o.e.n.Node               ] [node-1] started
[2022-03-15T09:30:26,212][WARN ][o.e.x.s.i.SetSecurityUserProcessor] [node-1] Creating processor [set_security_user] (tag
 [null]) on field [_security] but authentication is not currently enabled on this cluster  - this processor is likely to
```

注意：9300 端口为 Elasticsearch 集群间组件的通信端口，9200 端口为浏览

器访问的 http 协议 RESTful 端口。

打开浏览器（推荐使用谷歌浏览器），输入地址：http://localhost:9200，测试结果：




```
{
  "name" : "node-1",
  "cluster_name" : "my-elasticsearch",
  "cluster_uuid" : "BFFAf3EBRImVEGph9pqfiA",
  "version" : {
    "number" : "8.0.0",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "1b6a7ece17463df5ff54a3e1302d825889aa1161",
    "build_date" : "2022-02-03T16:47:57.507843096Z",
    "build_snapshot" : false,
    "lucene_version" : "9.0.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

三． Kibana 安装

1. 简介

Kibana 是一款开源的数据分析和可视化平台，它是 Elastic Stack 成员之一，设计用于和 Elasticsearch 协作。可以使用 Kibana 对 Elasticsearch 索引中的数据进行搜索、查看、交互操作。可以很方便的利用图表、表格及地图对数据进行多元化的分析和呈现。

Kibana 可以使大数据通俗易懂。它很简单，基于浏览器的界面便快速创建和分享动态数据仪表板来追踪 Elasticsearch 的实时数据变化。

2. 下载地址

版本：8.0.0

下载地址：https://www.elastic.co/cn/downloads/past-releases#kibana

解压到 /usr/local

3. Kibana 配置

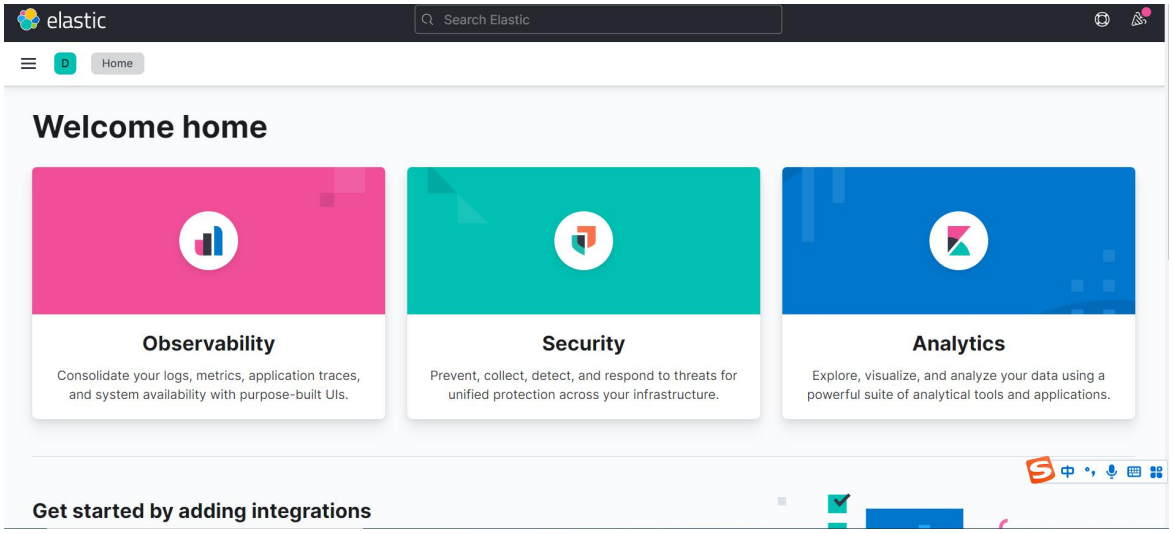
/usr/local/kibana-8.0.0/config/kibana.yml

```
#####----------kibana 服务相关----------#####
#提供服务的端口，监听端口
server.port: 5601
#主机地址，可以是 ip,主机名
server.host: 0.0.0.0
#在代理后面运行，则可以指定安装 Kibana 的路径
#使用 server.rewriteBasePath 设置告诉 Kibana 是否应删除 basePath
#接收到的请求，并在启动时防止过时警告
#此设置不能以斜杠结尾
server.basePath: ""
#指定 Kibana 是否应重写以 server.basePath 为前缀的请求，或者要求它们由反向代理重写，默认 false
server.rewriteBasePath: false
#传入服务器请求的最大有效负载大小,以字节为单位，默认 1048576
server.maxPayloadBytes: 1048576
#该 kibana 服务的名称，默认 your-hostname
server.name: "your-hostname"
#服务的 pid 文件路径，默认/var/run/kibana.pid
pid.file: /var/run/kibana.pid
```

```
#####----------elasticsearch 相关----------#####
#kibana 访问 es 服务器的 URL,就可以有多个，以逗号","隔开
elasticsearch.hosts: ["http://localhost:9200"]
#当此值为 true 时，Kibana 使用 server.host 设定的主机名
#当此值为 false 时，Kibana 使用连接 Kibana 实例的主机的主机名
#默认 ture
elasticsearch.preserveHost: true
#Kibana 使用 Elasticsearch 中的索引来存储已保存的搜索，可视化和仪表板
#如果索引尚不存在，Kibana 会创建一个新索引
#默认.kibana
kibana.index: ".kibana"
#加载的默认应用程序
#默认 home
kibana.defaultAppId: "home"
#kibana 访问 Elasticsearch 的账号与密码(如果 ElasticSearch 设置了的话)
elasticsearch.username: "kibana_system"
elasticsearch.password: "pass"
#从 Kibana 服务器到浏览器的传出请求是否启用 SSL
#设置为 true 时，需要 server.ssl.certificate 和 server.ssl.key
server.ssl.enabled: true
server.ssl.certificate: /path/to/your/server.crt
server.ssl.key: /path/to/your/server.key
#从 Kibana 到 Elasticsearch 启用 SSL 后，ssl.certificate 和 ssl.key 的位置
elasticsearch.ssl.certificate: /path/to/your/client.crt
elasticsearch.ssl.key: /path/to/your/client.key
#PEM 文件的路径列表
elasticsearch.ssl.certificateAuthorities: [ "/path/to/your/CA.pem" ]
#控制 Elasticsearch 提供的证书验证
#有效值为 none，certificate 和 full
elasticsearch.ssl.verificationMode: full
#Elasticsearch 服务器响应 ping 的时间，单位 ms
elasticsearch.pingTimeout: 1500
#Elasticsearch  的响应的时间，单位 ms
elasticsearch.requestTimeout: 30000
#Kibana 客户端发送到 Elasticsearch 的标头列表
#如不发送客户端标头，请将此值设置为空
elasticsearch.requestHeadersWhitelist: []
#Kibana 客户端发往 Elasticsearch 的标题名称和值
elasticsearch.customHeaders: {}
#Elasticsearch 等待分片响应的时间
elasticsearch.shardTimeout: 30000
#Kibana 刚启动时等待 Elasticsearch 的时间，单位 ms，然后重试
elasticsearch.startupTimeout: 5000
#记录发送到 Elasticsearch 的查询
```

```
elasticsearch.logQueries: false
#####----------日志相关----------#####
#kibana 日志文件存储路径，默认 stdout
logging.dest: stdout
#此值为 true 时，禁止所有日志记录输出
#默认 false
logging.silent: false
#此值为 true 时，禁止除错误消息之外的所有日志记录输出
#默认 false
logging.quiet: false
#此值为 true 时，记录所有事件，包括系统使用信息和所有请求
#默认 false
logging.verbose: false
#####----------其他----------#####
#系统和进程取样间隔，单位 ms，最小值 100ms
#默认 5000ms
ops.interval: 5000
#kibana web 语言
#默认 en
i18n.locale: "en"
```

4. Kibana 启动



四． ElasticSearch-Head 安装

1. 简介

elasticsearch-head 被称为是 elasticsearch 集群的 web 前端，head 插件主要是用来和 elastic Cluster 交互的 Web 前端。

2. 下载地址

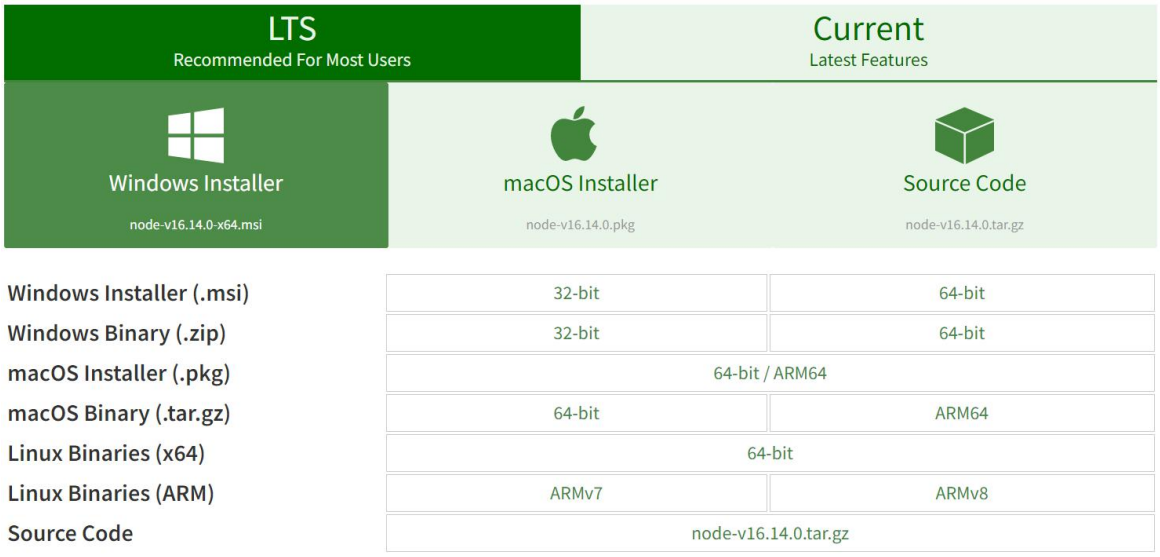
https://github.com/mobz/elasticsearch-head

3. ElasticSearch-Head 配置

ES5 以上的版本中安装 Elasticsearch-Head 必须要安装 NodeJs,然后通过 NodeJS 来启动 Head。

(1) 安装 node.js

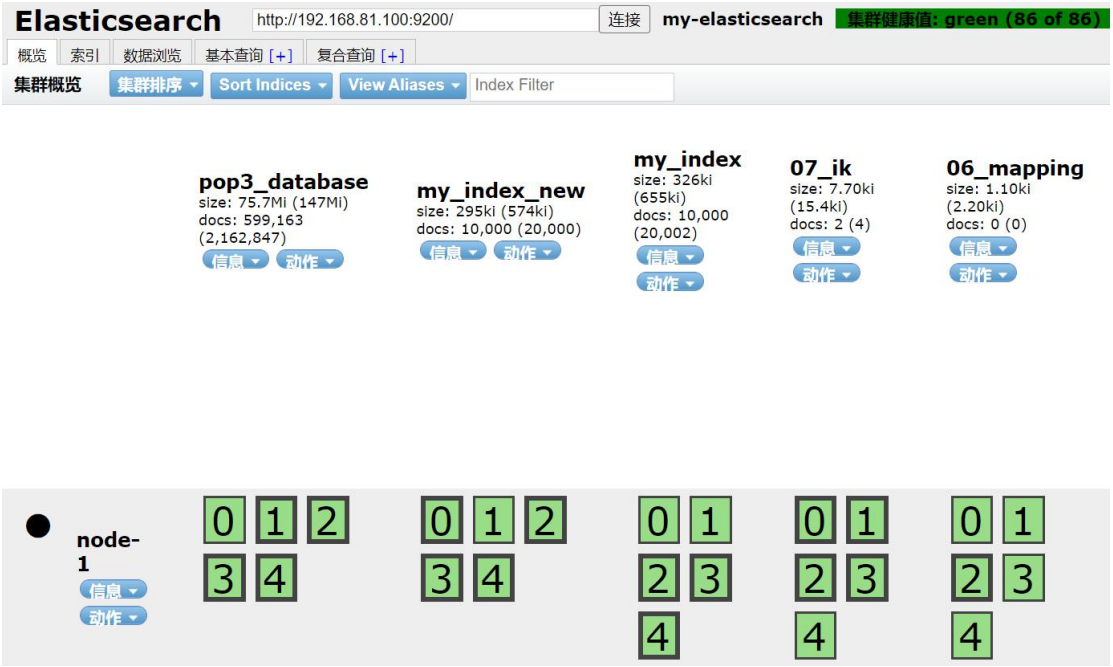
https://nodejs.org/en/download/



(2) Gruntfile.js 配置修改

```
connect: {
        server: {
                options: {
                        hostname:'*',
                        port: 9100,
                        base: '.',
                        keepalive: true
                }
        }
}
```

4. ElasticSearch-Head 启动



五. ElasticSearch 操作

操作方式：Kibana Dev Tools

1. _cat 操作

_cat 接口	说明
GET /_cat/nodes	查看所有节点
GET /_cat/health	查看 ES 健康状况
GET /_cat/master	查看主节点
GET /_cat/indices	查看所有索引信息

/_cat/indices?v 查看所有的索引信息

```
health status index                                uuid                   pri rep docs.count docs.deleted store.size pri.store.size
green  open   02_create_index                      _RrB5TFnQb2rRDBN-I8OBw   5   1          4            0     32.8kb         16.4kb
green  open   my_index_new                        YkkxOh_-TAuvSjprtVgNcw   5   1      10000            0    573.6kb        294.5kb
green  open   07_ik                               EbO68OquQ1WPV2-fAQeBYQ   5   1          2            0     15.4kb          7.7kb
green  open   06_mapping                          okyujDV3Sl2iltLq0qxS_A   5   1          0            0      2.1kb             1kb
green  open   my_index                            jFds1AppQwipaL3eANhXIQ   5   1      10000            1    655.3kb        325.8kb
green  open   04_bank                             8m6hBrhWSKe4hS222oWYdA   5   1       1000            0    884.2kb        442.1kb
green  open   05_ceshi                            K_mCrtdrRKO6FjB-QZWp3Q   1   1          1            0     12.4kb          6.2kb
green  open   .ds-logs-generic-default-2022.03.11-000001 mvD3bI-ARR2Pcbe3KSMc1w   1   1          5            0     49.2kb         24.6kb
green  open   03_doc_operation                    IlkWzyanQ520EE_DYooa0w   1   1          2            1     17.5kb          5.3kb
green  open   04_bank_new                         u1SIWhRPQ126YXX6iiUEOA   5   1       1000            0    541.2kb        270.6kb
green  open   pop3_database                       tmHnaHiWSL-Tiwv7AbjuQg   5   1     599163       442164    146.7mb         75.6mb
```

字段名	含义说明
health	green(集群完整) yellow(单点正常、集群不完整) red(单点不正常)
status	是否能使用
index	索引名主节点几个
uuid	索引统一编号
pri	从节点几个
rep	文档数
docs.count	文档数
docs.deleted	文档被删了多少
store.size	整体占空间大小
pri.store.size	主节点占

2. 索引操作

2.1 索引创建

对比关系型数据库，创建索引就等同于创建数据库

PUT /索引名

参数可选：指定分片及副本，示例分片为 5，副本为 1。

```
"settings":{
        "number_of_shards":5,
        "number_of_replicas":1
    },
```

请求后，服务器返回响应

```
{
    "acknowledged"【响应结果】: true, # true 操作成功
    "shards_acknowledged"【分片结果】: true, #  分片操作成功
```

"index"【索引名称】: "demo"
}
# 注意：创建索引库的分片数默认 1 片，在 7.0.0 之前的 Elasticsearch 版本中，默认 5 片
如果重复添加索引，会返回错误信息


```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "resource_already_exists_exception",
        "reason" : "index [demo/RImvGu4RQIqfr0KXMS33vg] already exists"
        ,
        "index_uuid" : "RImvGu4RQIqfr0KXMS33vg",
        "index" : "demo"
      }
    ],
    "type" : "resource_already_exists_exception",
    "reason" : "index [demo/RImvGu4RQIqfr0KXMS33vg] already exists",
    "index_uuid" : "RImvGu4RQIqfr0KXMS33vg",
    "index" : "demo"
  },
  "status" : 400
}
```


2.2 索引信息查看

GET /索引名


```
{
  "demo" : {
    "aliases" : { },
    "mappings" : { },
    "settings" : {
      "index" : {
        "routing" : {
          "allocation" : {
            "include" : {
              "_tier_preference" : "data_content"
            }
          }
        },
        "number_of_shards" : "5",
        "provided_name" : "demo",
        "creation_date" : "1647397518404",
        "number_of_replicas" : "1",
        "uuid" : "RImvGu4RQIqfr0KXMS33vg",
        "version" : {
          "created" : "8000099"
        }
      }
    }
  }
}
```


{
"shopping"【索引名】: {
"aliases"【别名】: {},

```
        "mappings"【映射】: {},
        "settings"【设置】: {
            "index"【设置-索引】: {
                "creation_date"【设置-索引-创建时间】: "1614265373911",
                "number_of_shards"【设置-索引-主分片数量】: "5",
                "provided_name"【设置-索引-名称】: "demo",
                "number_of_replicas"【设置-索引-副分片数量】: "1",
                "uuid"【设置-索引-唯一标识】: "RImvGu4RQIqfr0KXMS33vg",
                "version"【设置-索引-版本】: {
                "created": "8000099"
                },
                        }
        }
    }
}
```

可以使用 * 来查询所有索引具体信息


```
{
  ".ds-logs-generic-default-2022.03.11-000001" : {⟺},
  "02_create_index" : {⟺},
  "03_doc_operation" : {⟺},
  "04_bank" : {⟺},
  "04_bank_new" : {⟺},
  "05_ceshi" : {⟺},
  "06_mapping" : {⟺},
  "07_ik" : {⟺},
  "demo" : {⟺},
  "my_index" : {⟺},
  "my_index_new" : {⟺},
  "pop3_database" : {⟺}
}
```


2.3 索引删除

DELETE /索引名称


```
{
  "acknowledged" : true
}
```


重新访问索引时，服务器返回响应：索引不存在

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "index_not_found_exception",
        "reason" : "no such index [demo]",
        "resource.type" : "index_or_alias",
        "resource.id" : "demo",
        "index_uuid" : "_na_",
        "index" : "demo"
      }
    ],
    "type" : "index_not_found_exception",
    "reason" : "no such index [demo]",
    "resource.type" : "index_or_alias",
    "resource.id" : "demo",
    "index_uuid" : "_na_",
    "index" : "demo"
  },
  "status" : 404
}
```


3. 文档操作

3.1 文档创建

创建文档，并添加数据。这里的文档可以类比为关系型数据库中的表数据，添加的数据格式为 JSON 格式。

提交方式	描述
PUT	提交的 id 如果不存在就是新增操作，如果存在就是更新操作，id 不能为空
POST	如果不提供 id 会自动生成一个 id,如果 id 存在就更新，如果 id 不存在就新增

方式一：**PUT /索引名称/类型名/编号**

请求体内容为：

```
{
  "name":"one",
  "age":25
}
```


```
{
  "_index" : "demo",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```


```
{
    "_index"【索引】: "demo",
```

```
    "_id"【唯一标识】: "1", #可以类比为 MySQL 中的主键，随机生成
    "_version"【版本】: 1,
    "result"【结果】: "created", #这里的 create 表示创建成功
    "_shards"【分片】: {
        "total"【分片 - 总数】: 2,
        "successful"【分片 - 成功】: 2,
        "failed"【分片 - 失败】: 0
    },
    "_seq_no": 0,
    "_primary_term": 1
}
```

方式二：**POST /索引名称/类型名/编号**

```
{
  "_index" : "demo",
  "_id" : "2Cm-kH8BSvtzLdjqm8dQ",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "_seq_no" : 1,
  "_primary_term" : 1
}
```

由于没有指定数据唯一性标识（ID），默认情况下，ES 服务器会随机生成一个。

3.2 文档查看

查看文档时，需要指明文档的唯一性标识，类似于 MySQL 中数据的主键查询。

GET /索引/类型/id

```
{
  "_index" : "demo",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "one",
    "age" : 25
  }
}
```

字段	含义
_index	索引名称
_id _version	记录 id 版本号
_seq_no	并发控制字段，每次更新都会+1，用来实现乐锁

_primary_term	同上，主分片重新分配，如重启，就会发生变化
found	找到结果
_source	真正的数据内容

3.3 文档更新

POST /索引/类型/id

和新增文档一样，输入相同的 URL 地址请求，如果请求体变化，会将原有的数据内容覆盖。

请求体内容为:

```
{
   "name":"one",
   "age":24
}
```

```
{
  "_index" : "demo",
  "_id" : "1",
  "_version" : 2,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "_seq_no" : 2,
  "_primary_term" : 1
}
```

"_version"【版本】: 2,

"result"【结果】: "updated", # updated 表示数据被更新

3.4 文档批量插入

ES 提供了 Bulk API 支持批量操作，当我们有大量的写任务时，可以使用 Bulk 来进行批量写入。

通用的策略如下：Bulk 默认设置批量提交的数据量不能超过 100M。数据条数一般是根据文档的大小和服务器性能而定的，但是单次批处理的数据大小应从 5MB～15MB 逐渐增加，当性能没有提升时，把这个数据量作为最大值。

_bulk 批量操作，语法格式

{action:{metadata}}

{request body }

{action:{metadata}}

{request body }

测试数据：demo.json

```
POST /04_bank/_bulk
{"index":{"_id":"1"}}
{"account_number":1,"balance":39225,"firstname":"Amber","lastname"
  :"Duke","age":32,"gender":"M","address":"880 Holmes Lane","employer"
  :"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state"
  :"IL"}
{"index":{"_id":"6"}}
{"account_number":6,"balance":5686,"firstname":"Hattie","lastname"
  :"Bond","age":36,"gender":"M","address":"671 Bristol Street"
  ,"employer":"Netagy","email":"hattiebond@netagy.com","city":"Dante"
  ,"state":"TN"}
{"index":{"_id":"13"}}
{"account_number":13,"balance":32838,"firstname":"Nanette","lastname"
  :"Bates","age":28,"gender":"F","address":"789 Madison Street"
  ,"employer":"Quility","email":"nanettebates@quility.com","city"
  :"Nogal","state":"VA"}
{"index":{"_id":"18"}}
{"account_number":18,"balance":4180,"firstname":"Dale","lastname"
  :"Adams","age":33,"gender":"M","address":"467 Hutchinson Court"
  ,"employer":"Boink","email":"daleadams@boink.com","city":"Orick"
  ,"state":"MD"}
{"index":{"_id":"20"}}
{"account_number":20,"balance":16418,"firstname":"Elinor","lastname"
  :"Ratliff","age":36,"gender":"M","address":"282 Kings Place"
```

3.5 文档删除

DELETE /索引/类型/id

删除一个文档不会立即从磁盘上移除，它只是被标记成已删除（逻辑删除）。

```
{
  "_index" : "demo",
  "_id" : "1",
  "_version" : 2,
  "result" : "deleted",
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "_seq_no" : 1,
  "_primary_term" : 1
}
```

"_version"【版本】: 2, #对数据的操作，都会更新版本

"result"【结果】: "deleted", # deleted 表示数据被标记为删除

4. 映射操作

索引库(index)中的映射类似于数据库(database)中的表结构(table)。创建数据库表需要设置字段名称，类型，长度，约束等；索引库也一样，需要知道这个类型下有哪些字段，每个字段有哪些约束信息。

4.1 映射创建

PUT /索引库名/_mapping/类型名称

```
{
  "properties": {
```

```
      "name":{
        "type": "text",
        "index": true
      },
      "age":{
        "type": "long",
        "index": false
      }
    }
}
```

1. 字段名：任意填写，下面指定许多属性，例如：title、subtitle、images、price
2. type：类型，Elasticsearch 中支持的数据类型非常丰富，说几个关键的：

(1) String 类型，又分两种：

text：可分词

keyword：不可分词，数据会作为完整字段进行匹配

(2) Numerical：数值类型，分两类

基本数据类型：long、integer、short、byte、double、float、half_float

浮点数的高精度类型：scaled_float

(3) Date：日期类型

(4) Array：数组类型

(5) Object：对象

3. index：是否索引，默认为 true，也就是说不进行任何配置，所有字段都会被索引。

true：字段会被索引，则可以用来进行搜索

false：字段不会被索引，不能用来搜索

4. store：是否将数据进行独立存储，默认为 false

原始的文本会存储在_source 里面，默认情况下其他提取出来的字段都不是独立存储的，是从_source 里面提取出来的。当然可以独立的存储某个字段，只要设置"store": true 即可，获取独立存储的字段要比从_source 中解析快得多，但是也会占用更多的空间，所以要根据实际业务需求来设置。

5. analyzer：分词器，这里的 ik_max_word 即使用 ik 分词器。

4.2 映射查看

GET/索引库名/_mapping

```
{
  "demo" : {
    "mappings" : {
      "properties" : {
        "age" : {
          "type" : "long"
        },
        "name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        }
      }
    }
  }
}
```


4.3 映射新增

第一个就是先删除索引，然后调整后再新建索引映射，二在已有的基础上新增。

索引一旦创建，是无法修改里边的内容的，不如说修改索引字段的名称。但是是可以向索引中添加其他字段的。

PUT /06_mapping/_mapping

```
{
  "properties":{
    "sex":{
      "type":"text"
    }
  }
}
```


```
{
  "demo" : {
    "mappings" : {
      "properties" : {
        "age" : {
          "type" : "long"
        },
        "name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "sex" : {
          "type" : "text"
        }
      }
    }
  }
}
```

4.4 删除映射

delete /demo/_mapping

4.5 常用数据类型

(1) text、keyword、number、array、range、boolean、date、geo_point、ip、nested、object。
(2) text：默认会进行分词，支持模糊查询（5.x 之后版本 string 类型已废弃，请大家使用 text）。
(3) keyword：不进行分词，默认开启 doc_values 来加速聚合排序操作，占用了大量磁盘 io 如非必须可以禁用 doc_values。
(4) number：如果只有过滤场景 用不到 range 查询的话，使用 keyword 性能更佳，另外数字类型的 doc_values 比字符串更容易压缩。
(5) range：对数据的范围进行索引，目前支持 number range、date range 、ip range。
(6) array：es 不需要显示定义数组类型，只需要在插入数据时用'[]'表示即可。[] 中的元素类型需保持一致。
(7) boolean: 只接受 true、false，也可以是字符串类型的"true"、"false"。
(8) date：支持毫秒、根据指定的 format 解析对应的日期格式，内部以 long 类型存储。
(9) geo_point：存储经纬度数据对。
(10) ip：将 ip 数据存储在这种数据类型中，方便后期对 ip 字段的模糊与范围查询。
(11) ested：嵌套类型，一种特殊的 object 类型，存储 object 数组，可检索内部子项。
(12) object：嵌套类型，不支持数组。

5. 高级查询

Elasticsearch 提供了基于 JSON 提供完整的查询 DSL 来定义查询

5.1 RESTful

REST 指的是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是 RESTful。Web 应用程序最重要的 REST 原则是，客户端和服务器之间的交互在请求之间是无状态的。从客户端到服务器的每个请求都必须包含理解请求所必需的信息。如果服务器在请求之间的任何时间点重启，客户端不会得到通知。此外，无状态请求可以由任何可用服务器回答，这十分适合云计算之类的环境。客户端可以缓存数据以改进性能。

在服务器端，应用程序状态和功能可以分为各种资源。资源是一个有趣的概念实体，它向客户端公开。资源的例子有：应用程序对象、数据库记录、算法等等。每个资源都使用 URI (Universal Resource Identifier) 得到一个唯一的地址。所有资源都共享统一的接口，以便在客户端和服务器之间传输状态。使用的是标准的 HTTP 方法，比如 GET、PUT、POST 和 DELETE。

在 REST 样式的 Web 服务中，每个资源都有一个地址。资源本身都是方法调用的目标，方法列表对所有资源都是一样的。这些方法都是标准方法，包括 HTTP GET、POST、PUT、DELETE，还可能包括 HEAD 和 OPTIONS。简单的理解就是，如果想要访问互联网上的资源，就必须向资源所在的服务器发出请求，请求体中必须包含资源的网络路径，以及对资源进行的操作(增删改查)。

5.2 查询所有文档

GET bank/_search

```
{
  "query": {
    "match_all": {}
  }
}
# "query"：这里的 query 代表一个查询对象，里面可以有不同的查询属性
# "match_all"：查询类型，例如：match_all(代表查询所有)， match，term ， range 等等
# {查询条件}：查询条件会根据类型的不同，写法也有差异
```

```
{
  "took" : 38,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "04_bank",
        "_id" : "13",
        "_score" : 1.0,
        "_source" : {
          "account_number" : 13,
          "balance" : 32838,
          "firstname" : "Nanette",
```

```
{
  "took【查询花费时间，单位毫秒】" : 38,
  "timed_out【是否超时】" : false,
  "_shards【分片信息】" : {
    "total【总数】" : 5,
    "successful【成功】" : 5,
    "skipped【忽略】" : 0,
    "failed【失败】" : 0
  },
  "hits【搜索命中结果】" : {
      "total"【搜索条件匹配的文档总数】: {
          "value"【总命中计数的值】: 1000,
          "relation"【计数规则】: "eq" # eq 表示计数准确， gte 表示计数不准确
      },
    "max_score【匹配度分值】" : 1.0,
    "hits【命中结果集合】" : [
        。。。
```

```
        }
      ]
    }
}
```

5.3 匹配查询

match 匹配类型查询，会把查询条件进行分词，然后进行查询，多个词条之间是 or 的关系。

```
{
  "query": {
    "match": {
      "address":"mail lane"
    }
  }
}
```


```
{
  "took" : 181,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 16,
      "relation" : "eq"
    },
    "max_score" : 4.8202815,
    "hits" : [
```


响应结果有 16 条

5.4 字段匹配查询

multi_match 与 match 类似，不同的是它可以在多个字段中查询。

```
"query": {
    "multi_match": {
      "query": "666",
      "fields": ["gender","address"]
    }
```

```
{
  "took" : 6,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 4.9972124,
    "hits" : [
      {
        "_index" : "04_bank",
        "_id" : "988",
        "_score" : 4.9972124,
        "_source" : {
          "account_number" : 988,
          "balance" : 17803,
          "firstname" : "Lucy",
```


5.5 关键字精确查询

term 查询，精确的关键词匹配查询，不对查询条件进行分词。

```
{
        "query": {
          "term": {
            "balance": {
              "value": 17803
            }
          }
        }
    }
```

```
{
  "took" : 5,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "04_bank",
        "_id" : "988",
        "_score" : 1.0,
        "_source" : {
          "account_number" : 988,
          "balance" : 17803,
          "firstname" : "Lucy",
```


5.6 范围查询

range 查询找出那些落在指定区间内的数字或者时间。range 查询允许以下字符

操作符	说明
gt	大于>
gte	大于等于>=
lt	小于<
lte	小于等于<=

```
{
        "query": {
            "range": {
                "account_number": {
                    "gte": 995,
                    "lte": 1000
                }
            }
        },
        "sort": [
            {
                "account_number": {
                    "order": "asc"
                }
            }
        ]
    }
```

5.7 过滤查询

并不是所有的查询都需要产生分数，特别是那些仅用于"filtering"的文档，为了不计算分数，ElasticSearch 会自动检查场景并且优化查询的执行。

```
{
  "query": {
    "bool": {
      "must": [
        {
          "match_all": {}
        }
      ],
      "filter": [
        {
          "range": {
            "balance": {
              "gte": 100,
              "lte": 1032
            }
          }
        }
      ]
    }
  }
}
```

结果会过滤出 balance 大于 100 小于 1032 的值。

5.8 组合查询

bool 把各种其它查询通过 must（必须 ）、must_not（必须不）、should（应该）的方式进行组合。

must：必须匹配每个子查询，类似“与”，参与算分

should：选择性匹配子查询，类似“或”，参与算分

must_not：必须不匹配，不参与算分，类似“非”

filter：必须匹配，不参与算分

```
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "address": "mail lane"
          }
        }
      ],
      "must_not": [
        {
```

```
                "range": {
                    "age": {
                        "gte": 10,
                        "lte": 30
                    }
                }
            }
        ],
        "filter": [
            {
                "term": {
                    "balance"{
                        "value":45975
                    }
                }
            }
        ]
    }
  }
}
```

5.9 排序查询

elasticsearch 默认是根据相关度算分（_score）来排序，但是也支持自定义方式对搜索结果排序。可以排序字段类型有：keyword 类型、数值类型、地理坐标类型、日期类型等。

```
{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "age": {
        "order": "desc" #  先按照 age 进行降序排列
      },
      "account_number": {
        "order": "asc" # age 相同情况下按照序号升序排列
      }
    }
  ]
}
```

5.10 分页查询

from：当前页的起始索引，默认从 0 开始。 from = (pageNum - 1) * size

size：每页显示多少条

```
{
  "query": {
```

```
        "match_all": {}
      },
      "sort": [
        {
          "age": {
            "order": "desc"
          }
        }
      ],
      "from": 0,
      "size": 2
    }
```

结果只会显示从第 1 条开始显示 2 条。

5.11 深度分页

由于 es 是分布式的，数据都是分布在多个分片上。当页数越深，查询的节点的数量越大，自然占用的内存也越多。ES 为了避免深度分页带来的内存开销，ES 默认限定只能查询 10000 个文档。

demo：报错

```
POST /pop3_database/_search
{
  "from" : 0 ,
  "size" : 10001 ,
  "query" : {
    "match_all":{}
}
}
```

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "illegal_argument_exception",
        "reason" : "Result window is too large, from + size must be
          less than or equal to: [10000] but was [10001]. See the
          scroll api for a more efficient way to request large data
          sets. This limit can be set by changing the [index
          .max_result_window] index level setting."
      }
    ],
    "type" : "search_phase_execution_exception",
    "reason" : "all shards failed",
    "phase" : "query",
    "grouped" : true,
    "failed_shards" : [
      {
        "shard" : 0,
        "index" : "pop3_database",
```

from + size 这种方式不适用于深度分页场景，其它两种分页方式：

(1) **Search After**：只能一页一页往下查询。search_after 分页的方式是根据上一页的最后一条数据来确定下一页的位置，同时在分页请求的过程中，如果有

索引数据的增删改查，这些变更也会实时的反映到游标上。

测试数据集：

{ "index" : { "_index" : "twitter", "_id": 1} }
{"user":"双榆树-张三", "DOB":"1980-01-01", "message":"今儿天气不错啊，出去转转去","uid":2,"age":20,"city":"北京","province":"北京","country":"中国","address":"中国北京市海淀区","location":{"lat":"39.970718","lon":"116.325747"}}
{ "index" : { "_index" : "twitter", "_id": 2 }}
{"user":"东城区-老刘", "DOB":"1981-01-01", "message":"出发， 下一站云南！","uid":3,"age":30,"city":"北京","province":"北京","country":"中国","address":"中国北京市东城区台基厂三条 3 号","location":{"lat":"39.904313","lon":"116.412754"}}
{ "index" : { "_index" : "twitter", "_id": 3} }
{"user":"东城区-李四", "DOB":"1982-01-01", "message":"happy birthday!","uid":4,"age":30,"city":"北京","province":"北京","country":"中国","address":"中国北京市东城区","location":{"lat":"39.893801","lon":"116.408986"}}
{ "index" : { "_index" : "twitter", "_id": 4} }
{"user":"朝阳区-老贾","DOB":"1983-01-01", "message":"123,gogogo","uid":5,"age":35,"city":"北京","province":"北京","country":"中国","address":"中国北京市朝阳区建国门","location":{"lat":"39.718256","lon":"116.367910"}}
{ "index" : { "_index" : "twitter", "_id": 5} }
{"user":"朝阳区-老王","DOB":"1984-01-01", "message":"Happy BirthDay My Friend!","uid":6,"age":50,"city":"北京","province":"北京","country":"中国","address":"中国北京市朝阳区国贸","location":{"lat":"39.918256","lon":"116.467910"}}
{ "index" : { "_index" : "twitter", "_id": 6} }
{"user":"虹桥-老吴", "DOB":"1985-01-01", "message":"好友来了都今天我生日，好友来了,什么 birthday happy 就成!","uid":7,"age":90,"city":"上海","province":"上海","country":"中国","address":"中国上海市闵行区","location":{"lat":"31.175927","lon":"121.383328"}}

检索第一页的前 2 条数据：

```
{
  "size": 2,
  "query": {
    "match": {
      "city": "北京"
    }
  },
  "sort": [
    {
      "DOB": {
        "order": "asc"
      }
    },
    {
      "user.keyword": {
        "order": "asc"
      }
    }
  ]
}
```

显示结果：


```
{
        "_index" : "twitter",
        "_id" : "1",
        "_score" : null,
        "_source" : {
          "user" : "双榆树-张三",
          "DOB" : "1980-01-01",
          "message" : "今儿天气不错啊，出去转转去",
          "uid" : 2,
          "age" : 20,
          "city" : "北京",
          "province" : "北京",
          "country" : "中国",
          "address" : "中国北京市海淀区",
          "location" : {
            "lat" : "39.970718",
            "lon" : "116.325747"
          }
        },
        "sort" : [
          315532800000,
          "双榆树-张三"
        ]
      },
      {
        "_index" : "twitter",
```

```
      "_id" : "2",
      "_score" : null,
      "_source" : {
        "user" : "东城区-老刘",
        "DOB" : "1981-01-01",
        "message" : "出发，下一站云南！",
        "uid" : 3,
        "age" : 30,
        "city" : "北京",
        "province" : "北京",
        "country" : "中国",
        "address" : "中国北京市东城区台基厂三条 3 号",
        "location" : {
          "lat" : "39.904313",
          "lon" : "116.412754"
        }
      },
      "sort" : [
        347155200000,
        "东城区-老刘"
      ]
    }
```


上述请求的结果包括每个文档的 sort 值数组。这些 sort 值可以与 search_after 参数一起使用，以开始返回在这个结果列表之后的任何文档。

```
GET twitter/_search
{
  "size": 2,
  "query": {
    "match": {
      "city": "北京"
    }
  },
  "search_after": [
    347155200000,
    "东城区-老刘"
  ],
  "sort": [
    {
      "DOB": {
        "order": "asc"
      }
    },
    {
      "user.keyword": {
        "order": "asc"
      }
    }
  ]
```

把上一个搜索结果的 sort 值放进来。显示的结果为：

```
{
        "_index" : "twitter",
        "_id" : "3",
        "_score" : null,
        "_source" : {
          "user" : "东城区-李四",
          "DOB" : "1982-01-01",
          "message" : "happy birthday!",
          "uid" : 4,
          "age" : 30,
          "city" : "北京",
          "province" : "北京",
          "country" : "中国",
          "address" : "中国北京市东城区",
          "location" : {
            "lat" : "39.893801",
            "lon" : "116.408986"
          }
        },
        "sort" : [
          378691200000,
          "东城区-李四"
        ]
      },
      {
        "_index" : "twitter",
        "_id" : "4",
        "_score" : null,
        "_source" : {
          "user" : "朝阳区-老贾",
          "DOB" : "1983-01-01",
          "message" : "123,gogogo",
          "uid" : 5,
          "age" : 35,
          "city" : "北京",
          "province" : "北京",
          "country" : "中国",
          "address" : "中国北京市朝阳区建国门",
          "location" : {
            "lat" : "39.718256",
            "lon" : "116.367910"
          }
        },
        "sort" : [
```

```
            410227200000,
            "朝阳区-老贾"
        ]
    }
```

官方推荐使用 _uid 作为全局唯一值，其实使用业务层的 id 也可以。

(2) **Scroll Api**：滚动深度分页。这个分页的用法，不是为了实时查询数据，而是为了一次性查询大量的数据(甚至是全部的数据)。

scroll 相当于维护了一份当前索引段的快照信息，这个快照信息是你执行这个 scroll 查询时的快照。在这个查询后的任何新索引进来的数据，都不会在这个快照中查询到。在遍历时，从这个快照里取数据，也就是说，在初始化后，对索引插入、删除、更新数据都不会影响遍历结果。

第一次查询：查询出 3 条数据，scroll 指定为 5 分钟。

```
POST /twitter/_search?scroll=5m
{
  "size": 2,
  "query": {
    "match": {
      "city": "北京"
    }
  },
  "sort": [
    {
      "DOB": {
        "order": "asc"
      }
    },
    {
      "user.keyword": {
        "order": "asc"
      }
    }
  ]
```

显示信息：

```
{
    "_scroll_id" : "FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmV0Y2gBFlNLanNiVWxZUkttSE5zRkc4TFVxelEAAAAAAAAAGARZaWUw2R1A2cFJZdUVJeXJiZ1J3ZVZ3",
    "took" : 1,
    "timed_out" : false,
    "_shards" : {
      "total" : 1,
      "successful" : 1,
      "skipped" : 0,
      "failed" : 0
    },
    "hits" : {
      "total" : {
```

```
    "value" : 5,
    "relation" : "eq"
  },
  "max_score" : null,
  "hits" : [
    {
      "_index" : "twitter",
      "_id" : "1",
      "_score" : null,
      "_source" : {
        "user" : "双榆树-张三",
        "DOB" : "1980-01-01",
        "message" : "今儿天气不错啊，出去转转去",
        "uid" : 2,
        "age" : 20,
        "city" : "北京",
        "province" : "北京",
        "country" : "中国",
        "address" : "中国北京市海淀区",
        "location" : {
          "lat" : "39.970718",
          "lon" : "116.325747"
        }
      },
      "sort" : [
        315532800000,
        "双榆树-张三"
      ]
    },
    {
      "_index" : "twitter",
      "_id" : "2",
      "_score" : null,
      "_source" : {
        "user" : "东城区-老刘",
        "DOB" : "1981-01-01",
        "message" : "出发，下一站云南！",
        "uid" : 3,
        "age" : 30,
        "city" : "北京",
        "province" : "北京",
        "country" : "中国",
        "address" : "中国北京市东城区台基厂三条 3 号",
        "location" : {
```

```
            "lat" : "39.904313",
            "lon" : "116.412754"
          }
        },
        "sort" : [
          347155200000,
          "东城区-老刘"
        ]
      }
    ]
  }
}
```

第二次查询：在第一次查询的结果中，返回了执行的_scroll_id，复制下来，在第二次查询中带上这个_scroll_id 是快照的 id，这里面记录了上一次查询的信息，如查询排序，size 等。

```
GET /_search/scroll
{
  "scroll_id" : "FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmV0Y2gBFlNLan
    NiVWxZUkttSE5zRkc4TFVxelEAAAAAAAAFjxZaWUw2R1A2cFJZdUVJeXJiZ1J3ZVZ3"
    ,
  "scroll":"5m"
}
```

会显示接下 2 个的信息。


```
{
  "_scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmV0Y2gBFlNLanNiVW
xZUkttSE5zRkc4TFVxelEAAAAAAAAGRRZaWUw2R1A2cFJZdUVJeXJiZ1J3ZV
Z3",
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
      {
        "_index" : "twitter",
```

```
    "_id" : "3",
    "_score" : null,
    "_source" : {
      "user" : "东城区-李四",
      "DOB" : "1982-01-01",
      "message" : "happy birthday!",
      "uid" : 4,
      "age" : 30,
      "city" : "北京",
      "province" : "北京",
      "country" : "中国",
      "address" : "中国北京市东城区",
      "location" : {
        "lat" : "39.893801",
        "lon" : "116.408986"
      }
    },
    "sort" : [
      378691200000,
      "东城区-李四"
    ]
  },
  {
    "_index" : "twitter",
    "_id" : "4",
    "_score" : null,
    "_source" : {
      "user" : "朝阳区-老贾",
      "DOB" : "1983-01-01",
      "message" : "123,gogogo",
      "uid" : 5,
      "age" : 35,
      "city" : "北京",
      "province" : "北京",
      "country" : "中国",
      "address" : "中国北京市朝阳区建国门",
      "location" : {
        "lat" : "39.718256",
        "lon" : "116.367910"
      }
    },
    "sort" : [
      410227200000,
      "朝阳区-老贾"
```

```
        ]
      }
    ]
  }
}
```

第三次搜索：重复第二次，下面的 N 次搜索只需要重复第二次搜索的操作即可。结果显示最后一个结果。


```
{
  "_scroll_id" : "FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmV0Y2gBFlNLanNiVWxZUkttSE5zRkc4TFVxelEAAAAAAAAAGRRZaWUw2R1A2cFJZdUVJeXJiZ1J3ZVZ3",
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
      {
        "_index" : "twitter",
        "_id" : "5",
        "_score" : null,
        "_source" : {
          "user" : "朝阳区-老王",
          "DOB" : "1984-01-01",
          "message" : "Happy BirthDay My Friend!",
          "uid" : 6,
          "age" : 50,
          "city" : "北京",
          "province" : "北京",
          "country" : "中国",
          "address" : "中国北京市朝阳区国贸",
          "location" : {
            "lat" : "39.918256",
            "lon" : "116.467910"
```

```
            }
          },
          "sort" : [
            441763200000,
            "朝阳区-老王"
          ]
        }
      ]
    }
  }
```

ES7 版本变更

参照：在 7.*版本中，ES 官方不再推荐使用 Scroll 方法来进行深分页，而是推荐使用带 PIT 的 search_after 来进行查询；

从 7.*版本开始，您可以使用 SEARCH_AFTER 参数通过上一页中的一组排序值检索下一页命中。

使用 SEARCH_AFTER 需要多个具有相同查询和排序值的搜索请求。

如果这些请求之间发生刷新，则结果的顺序可能会更改，从而导致页面之间的结果不一致。

为防止出现这种情况，可以创建一个时间点(PIT)来在搜索过程中保留当前索引状态。

```
{
  "id" : "46ToAwEHdHdpdHRlchZDeEkxTzQyaFNuU3hyT1dHWHpHcE5BABZaWUw2R1A2cF
    JZdUVJeXJiZ1J3ZVZ3AAAAAAAAAApgFlNLanNiVWxZUkttSE5zRkc4TFVxelEAARZDeE
    kxTzQyaFNuU3hyT1dHWHpHcE5BAAA="
}
```

```
GET /_search
{
  "size": 5,
  "query": {
    "match": {
      "city": "北京"
    }
  },
  "pit": {
    "id":  "46ToAwEHdHdpdHRlchZDeEkxTzQyaFNuU3hyT1dHWHpHcE5BABZaWUw2R1
      A2cFJZdUVJeXJiZ1J3ZVZ3AAAAAAAAAApgFlNLanNiVWxZUkttSE5zRkc4TFVxel
      EAARZDeEkxTzQyaFNuU3hyT1dHWHpHcE5BAAA=",
    "keep_alive": "5m"
  },
  "sort": [
    {
      "DOB": {
        "order": "asc"
      }
    },
    {
      "user.keyword": {
        "order": "asc"
```

5.12 聚合查询

聚合可以极其方便的实现对数据的统计、分析。es 里面的聚合跟 sql 里面的统计求和最大值分组类似。例如：

什么品牌的手机最受欢迎？

这些手机的平均价格、最高价格、最低价格？

这些手机每月的销售情况如何？

实现这些统计功能的比数据库的 sql 要方便的多，而且查询速度非常快，可以实现实时搜索效果。

Elasticsearch 中的聚合，包含多种类型，最常用的两种，一个叫桶，一个叫度量。

(1) 桶（bucket）

桶的作用，是按照某种方式对数据进行分组，每一组数据在 ES 中称为一个桶。

Elasticsearch 中提供的划分桶的方式有很多：

① Date Histogram Aggregation：根据日期阶梯分组，例如给定阶梯为周，会自动每周分为一组。

② Histogram Aggregation：根据数值阶梯分组，与日期类似。

③ Terms Aggregation：根据词条内容分组，词条内容完全匹配的为一组。

④ Range Aggregation：数值和日期的范围分组，指定开始和结束，然后按段分组。

bucket aggregations 只负责对数据进行分组，并不进行计算，因此往往 bucket 中往往会嵌套另一种聚合：metrics aggregations 即度量

(2) 度量（metrics）

分组完成以后，一般会对组中的数据进行聚合运算，例如求平均值、最大、最小、求和等，这些在 ES 中称为度量

比较常用的一些度量聚合方式：

① Avg Aggregation：求平均值

② Max Aggregation：求最大值

③ Min Aggregation：求最小值

④ Percentiles Aggregation：求百分比

⑤ Stats Aggregation：同时返回 avg、max、min、sum、count 等

⑥ Sum Aggregation：求和

⑦ Top hits Aggregation：求前几

⑧ Value Count Aggregation：求总数

demo1:搜索 address 中包含 mill 的所有人的年龄分布以及平均年龄

```
{
  "query": { # 搜索 address 中包含 mill 的所有人
    "match": {
      "address": "mill"
    }
  },
  "aggs": { # Aggregation 聚合缩写
    "ageAggs": { # 聚合函数名称 自己进行定义即可
```

```
            "terms": { # 聚合的类型，进行分组 terms
              "field": "age", # 分组的字段
              "size": 10 # 分组的个数 例如 100 分为 10 组 每组 10 个
            }
          },
          "ageavg":{ # 第二个分组名称 平均年龄
            "avg": { # avg 聚合函数中求平均
              "field": "age" # 进行分组的字段
            }
          }
        },
        "size": 0 # 不对 query 查询结果进行输出，方便查看
}
```

demo2:按照年龄聚合，并且请求这些年龄段的这些人的平均薪资

```
    {
      "query": { # 查询全部的信息
        "match_all": {}
      },
      "aggs": {
        "ageaggs": { # 按照年龄进行分组，分为 50 个分组
          "terms": {
            "field": "age",
            "size": 50
          },
          "aggs": { # 在年龄的分组基础上在求每个分组的平均工资，这里使用
嵌套。。。
            "blanceavg": {
              "avg": {
                "field": "balance" # 字段为工资
              }
            }
          }
        }
      },
      "size": 0
    }
```

demo3 查出所有年龄分布，并且这些年龄段中 M 的平均薪资和 F 的平均薪资以及这个年龄段的总体平均薪资。

```
    {
      "query": {
        "match_all": {}
      },
```

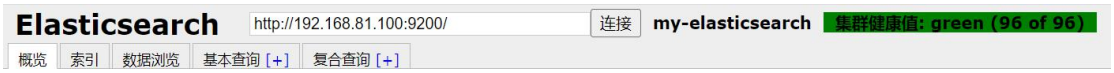
```
    "aggs": {
      "ageagg": { # 按照年龄进行分组
        "terms": {
          "field": "age",
          "size": 50
        },
        "aggs": {
          "genderagg": { # 嵌套，在年龄组里面在进行性别分组
            "terms": {
              "field": "gender.keyword",
              "size": 10
            },
            "aggs": {
              "blanceavg": { # 同样是嵌套，在上述分组中在求出 balance 的
大小
                "avg": {
                  "field": "balance"
                }
              }
            }
          }
        }
      },
      "size": 0
    }
```

六．ElasticSearch-Head 操作

1. 集群健康

```
{
    "settings" : {
        "number_of_shards" : 5,
        "number_of_replicas" : 1
    }
}
```

通过 elasticsearch-head 插件查看集群情况

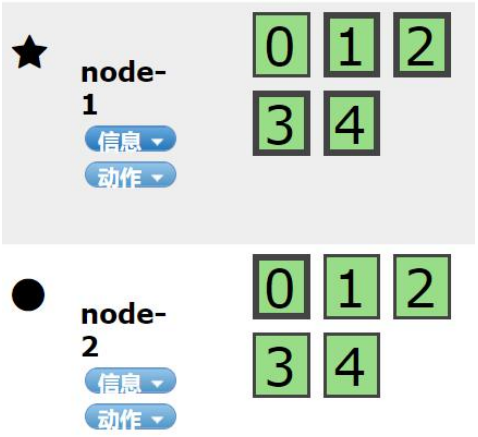


集群健康值:yellow(3 of 6)：表示当前集群的全部主分片都正常运行，但是副本分片没有全部处在正常状态。

集群健康值:yellow(3 of 6)：表示当前集群的全部主分片都正常运行，副本分片处在正常状态。

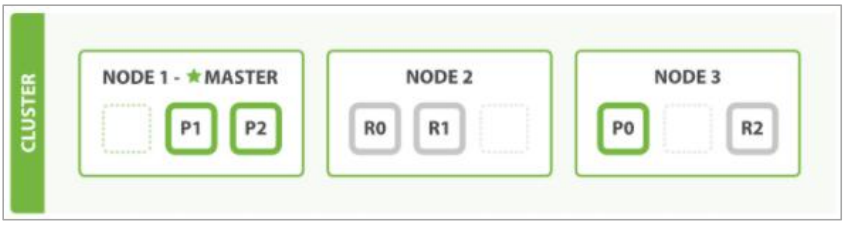
当第二个节点加入到集群后，3 个副本分片将会分配到这个节点上——每个主分片对应一个副本分片。这意味着当集群内任何一个节点出现问题时，数据都完好无损。所有新近被索引的文档都将会保存在主分片上，然后被并行的复制到对应的副本分片上。这就保证了既可以从主分片又可以从副本分片上获得文档。

分片示意图：

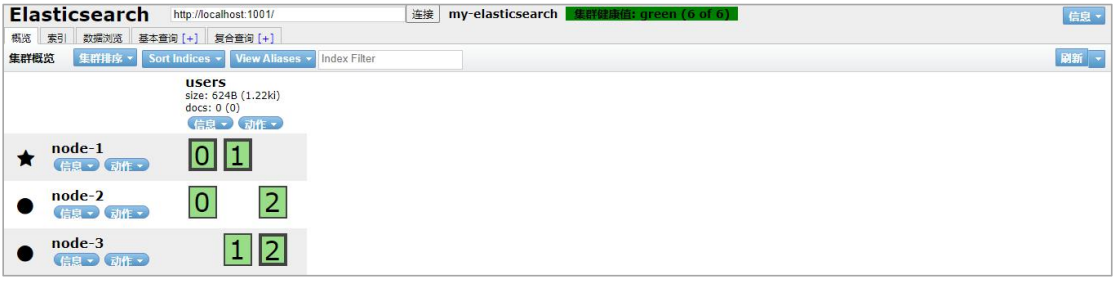


2. 水平扩容

当启动了第三个节点，集群将会拥有三个节点的集群：为了分散负载而对分片进行重新分配。



通过 elasticsearch-head 插件查看集群情况



集群健康值:green(6 of 6): 表示所有 6 个分片（包括 3 个主分片和 3 个副本分片）都在正常运行。

Node 1 和 Node 2 上各有一个分片被迁移到了新的 Node 3 节点，现在每个节点上都拥有 2 个分片，而不是之前的 3 个。 这表示每个节点的硬件资源（CPU, RAM, I/O）将被更少的分片所共享，每个分片的性能将会得到提升。分片是一个功能完整的搜索引擎，它拥有使用一个节点上的所有资源的能力。这个拥有 6 个分片（3 个主分片和 3 个副本分片）的索引可以最大扩容到 6 个节点，每个节点上存在一个分片，并且每个分片拥有所在节点的全部资源。

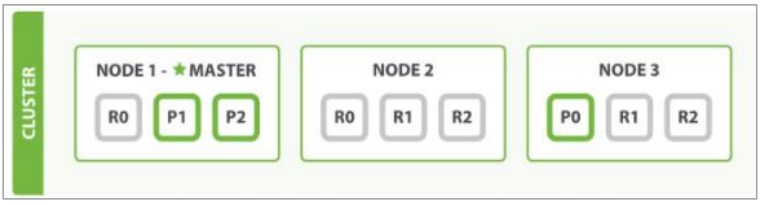
扩容超过 6 个节点：

主分片的数目在索引创建时就已经确定了下来。实际上，这个数目定义了这个索引能够存储的最大数据量。（实际大小取决于数据、硬件和使用场景。） 但是，读操作——搜索和返回数据——可以同时被主分片或副本分片所处理，所以当拥有越多的副本分片时，也将拥有越高的吞吐量。

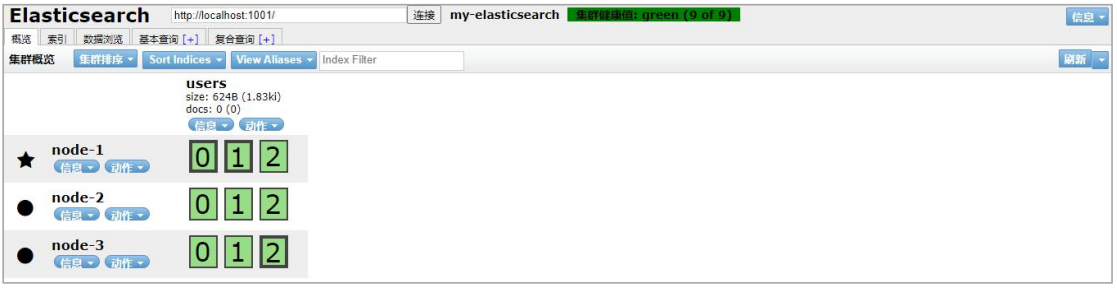
在运行中的集群上是可以动态调整副本分片数目的，可以按需伸缩集群。把副本数从默认的 1 增加到 2

```
{
    "number_of_replicas" : 2
}
```

users 索引现在拥有 9 个分片：3 个主分片和 6 个副本分片。 这意味着可以将集群扩容到 9 个节点，每个节点上一个分片。相比原来 3 个节点时，集群搜索性能可以提升 3 倍。



通过 elasticsearch-head 插件查看集群情况



当然，如果只是在相同节点数目的集群上增加更多的副本分片并不能提高性能，因为每个分片从节点上获得的资源会变少。需要增加更多的硬件资源来提升吞吐量。

但是更多的副本分片数提高了数据冗余量：按照上面的节点配置，可以在失去 2 个节点的情况下不丢失任何数据。

3. 路由计算

当索引一个文档的时候，文档会被存储到一个主分片中。 Elasticsearch 如何知道一个文档应该存放到哪个分片中呢？当创建文档时，它如何决定这个文档应当被存储在分片 1 还是分片 2 中呢？首先这肯定不会是随机的，否则将来要获取文档的时候就不知道从何处寻找了。实际上，这个过程是根据下面这个公式决定的：

```
shard = hash(routing) % number_of_primary_shards
```

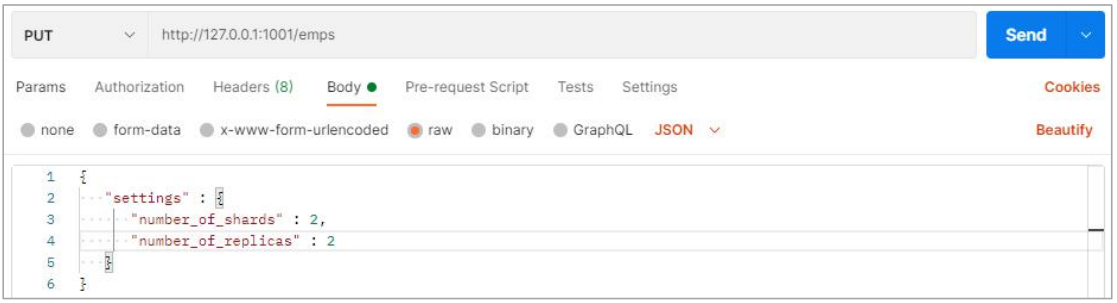
routing 是一个可变值，默认是文档的_id ，也可以设置成一个自定义的值。routing 通过 hash 函数生成一个数字，然后这个数字再除以 number_of_primary_shards（主分片的数量）后得到余数。这个分布 0 到 number_of_primary_shards-1 之间的余数，就是所寻求的文档所在分片的位置。

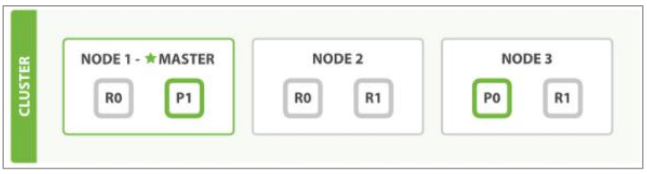
这就解释了为什么要在创建索引的时候就确定好主分片的数量 并且永远不会改变这个数量：因为如果数量变化了，那么所有之前路由的值都会无效，文档也再也找不到了。

所有的文档 API(get、index、delete、bulk、update 以及 mget)都接受一个叫做 routing 的路由参数，通过这个参数可以自定义文档到分片的映射。一个自定义的路由参数可以用来确保所有相关的文档——例如所有属于同一个用户的文档——都被存储到同一个分片中。

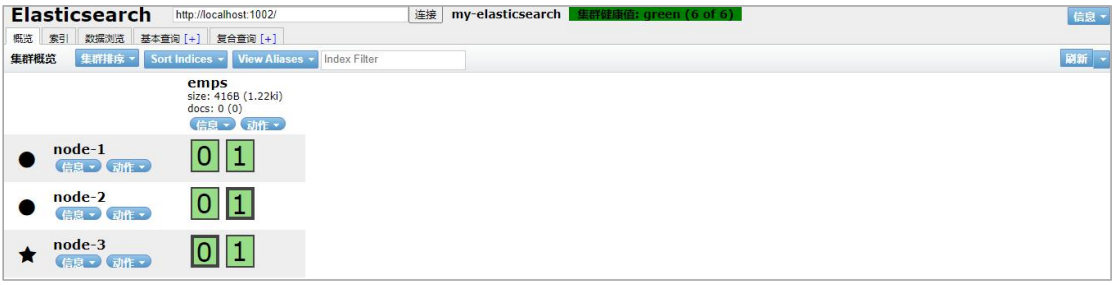
4. 分片控制

假设有一个集群由三个节点组成。 它包含一个叫 emps 的索引，有两个主分片，每个主分片有两个副本分片。相同分片的副本不会放在同一节点。





通过 elasticsearch-head 插件查看集群情况，所以集群是一个有三个节点和一个索引的集群。

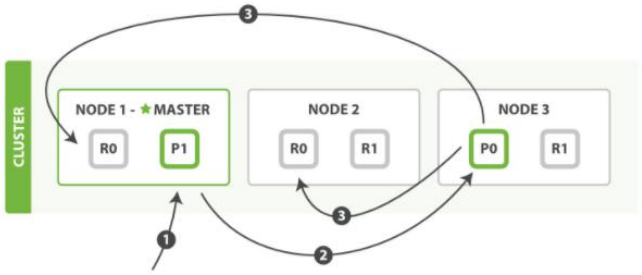


可以发送请求到集群中的任一节点。 每个节点都有能力处理任意请求。 每个节点都知道集群中任一文档位置，所以可以直接将请求转发到需要的节点上。在下面的例子中，将所有的请求发送到 Node 1，将其称为协调节点(coordinating node) 。

当发送请求的时候，为了扩展负载，更好的做法是轮询集群中所有的节点。

5. 写流程

新建、索引和删除请求都是写操作，必须在主分片上面完成之后才能被复制到相关的副本分片。



新建，索引和删除文档所需要的步骤顺序：

(1) 客户端向 Node 1 发送新建、索引或者删除请求。

(2) 节点使用文档的 _id 确定文档属于分片 0 。请求会被转发到 Node 3，因为分片 0 的主分片目前被分配在 Node 3 上。

(3) Node 3 在主分片上面执行请求。如果成功了，它将请求并行转发到 Node 1 和 Node 2 的副本分片上。一旦所有的副本分片都报告成功, Node 3 将向协调节点报告成功，协调节点向客户端报告成功。

在客户端收到成功响应时，文档变更已经在主分片和所有副本分片执行完成，变更是安全的。

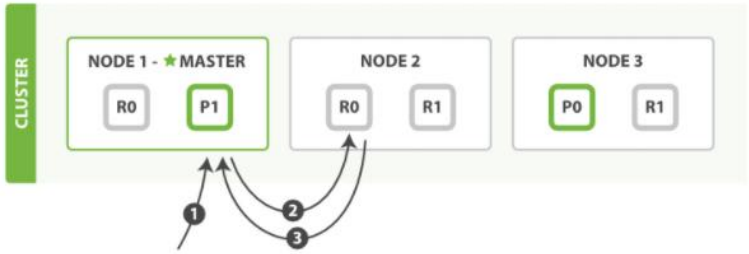
有一些可选的请求参数允许影响这个过程，可能以数据安全为代价提升性能。这些选项很少使用，因为 Elasticsearch 已经很快，但是为了完整起见，参考下面表格：

参数	含义
consistency	consistency，即一致性。在默认设置下，即使仅仅是在试图执行一个_写_操作之前，主分片都会要求 必须要有 规定数量(quorum)（或者换种说法，也即必须要有大多数）的分片副本处于活跃可用状态，才会去执行_写_操作(其中分片副本可以是主分片或者副本分片)。这是为了避免在发生网络分区故障（network partition）的时候进行_写_操作，进而导致数据不一致。_规定数量_即： int((primary + number_of_replicas) / 2) + 1 consistency 参数的值可以设为 one （只要主分片状态 ok 就允许执行_写_操作）,all（必须要主分片和所有副本分片的状态没问题才允许执行_写_操作），或 quorum 。默认值为 quorum，即大多数的分片副本状态没问题就允许执行_写_操作。 注意，规定数量 的计算公式中 number_of_replicas 指的是在索引设置中的设定副本分片数，而不是指当前处理活动状态的副本分片数。如果索引设置中指定了当前索引拥有三个副本分片，那规定数量的计算结果即： int((primary + 3 replicas) / 2) + 1 = 3 如果此时只启动两个节点，那么处于活跃状态的分片副本数量就达不到规定数量，也因此将无法索引和删除任何文档。
timeout	如果没有足够的副本分片会发生什么？ Elasticsearch 会等待，希望更多的分片出现。默认情况下，它最多等待 1 分钟。 如果需要，可以使用 timeout 参数 使它更早终止：100 100 毫秒，30s 是 30 秒。

新索引默认有 1 个副本分片，这意味着为满足规定数量应该需要两个活动的分片副本。 但是，这些默认的设置会阻止在单一节点上做任何事情。为了避免这个问题，要求只有当 number_of_replicas 大于 1 的时候，规定数量才会执行。

6. 读流程

可以从主分片或者从其它任意副本分片检索文档。



(1) 从主分片或者副本分片检索文档的步骤顺序：客户端向 Node 1 发送获取请求。

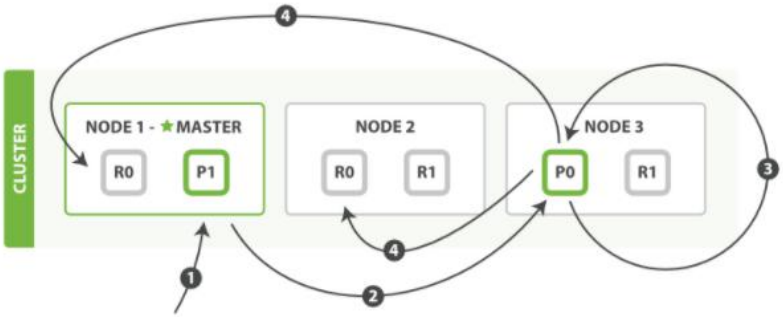
(2) 节点使用文档的 _id 来确定文档属于分片 0 。分片 0 的副本分片存在于所有的三个节点上。 在这种情况下，它将请求转发到 Node 2 。

(3) Node 2 将文档返回给 Node 1 ，然后将文档返回给客户端。在处理读取请求时，协调结点在每次请求的时候都会通过轮询所有的副本分片来达到负载均

衡。在文档被检索时，已经被索引的文档可能已经存在于主分片上但是还没有复制到副本分片。 在这种情况下，副本分片可能会报告文档不存在，但是主分片可能成功返回文档。 一旦索引请求成功返回给用户，文档在主分片和副本分片都是可用的。

7. 更新流程

部分更新一个文档结合了先前说明的读取和写入流程：



部分更新一个文档的步骤如下：

(1) 客户端向 Node 1 发送更新请求。

(2) 它将请求转发到主分片所在的 Node 3 。

(3) Node 3 从主分片检索文档，修改 _source 字段中的 JSON ，并且尝试重新索引主分片的文档。 如果文档已经被另一个进程修改，它会重试步骤 3 ，超过 retry_on_conflict 次后放弃。

(4) 如果 Node 3 成功地更新文档，它将新版本的文档并行转发到 Node 1 和 Node 2 上的副本分片，重新建立索引。一旦所有副本分片都返回成功，Node 3 向协调节点也返回成功，协调节点向客户端返回成功。

当主分片把更改转发到副本分片时，它不会转发更新请求。相反，它转发完整文档的新版本。这些更改将会异步转发到副本分片，并且不能保证它们以发送它们相同的顺序到达。如果 Elasticsearch 仅转发更改请求，则可能以错误的顺序应用更改，导致得到损坏的文档。

七．IK 分词器

所谓的分词就是通过 tokenizer(分词器)将一个字符串拆分为多个独立的 tokens(词元-独立的单词)， 然后输出为 tokens 流的过程。

1. 内置分词器

Elasticsearch 还附带了可以直接使用的预包装的分析器。接下来我们会列出最重要的分析器。为了证明它们的差异，我们看看每个分析器会从下面的字符串得到哪些词条：

"Set the shape to semi-transparent by calling set_trans(5)"

(1) 标准分析器

标准分析器是 Elasticsearch 默认使用的分析器。它是分析各种语言文本最常用的选择。它根据 Unicode 联盟 定义的 单词边界 划分文本。删除绝大部分标点。最后，将词条小写。它会产生：

set, the, shape, to, semi, transparent, by, calling, set_trans, 5

(2) 简单分析器

简单分析器在任何不是字母的地方分隔文本，将词条小写。它会产生：

set, the, shape, to, semi, transparent, by, calling, set, trans

(3) 空格分析器

空格分析器在空格的地方划分文本。它会产生：

Set, the, shape, to, semi-transparent, by, calling, set_trans(5)

(4) 语言分析器

特定语言分析器可用于很多语言。它们可以考虑指定语言的特点。例如，英语分析器附带了一组英语无用词（常用单词，例如 and 或者 the ，它们对相关性没有多少影响），它们会被删除。由于理解英语语法的规则，这个分词器可以提取英语单词的词干 。

英语分词器会产生下面的词条：

set, shape, semi, transpar, call, set_tran, 5

transparent、 calling 和 set_trans 已经变为词根格式

2. 测试分词器

例如"my name is one"这样一个字符串就会被默认的分词器拆分为[my,name,is,one]。ElasticSearch 中提供了很多默认的分词器。

```
{
  "tokens" : [
    {
      "token" : "my",
      "start_offset" : 0,
      "end_offset" : 2,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "name",
      "start_offset" : 3,
      "end_offset" : 7,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "is",
      "start_offset" : 8,
      "end_offset" : 10,
      "type" : "<ALPHANUM>",
      "position" : 2
    },
    {
      "token" : "one",
      "start_offset" : 11,
      "end_offset" : 14,
      "type" : "<ALPHANUM>",
      "position" : 3
    }
  ]
}
```


3. IK 中文分词器

测试中文分词功能：

```
{
   "analyzer": "standard",
   "text": "我是中国人，我热爱我的祖国"
}
```


```
{
  "tokens" : [
    {
      "token" : "我",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "<IDEOGRAPHIC>",
      "position" : 0
    },
    {
      "token" : "是",
      "start_offset" : 1,
      "end_offset" : 2,
      "type" : "<IDEOGRAPHIC>",
      "position" : 1
    },
    {
      "token" : "中",
      "start_offset" : 2,
      "end_offset" : 3,
      "type" : "<IDEOGRAPHIC>",
      "position" : 2
    },
    {
```

分词之后：【我 是 中 国 人 ， 我 热 爱 我 的 祖 国】

显然在 ElasticSearch 中提供的分词器对中文的分词效果都不好，所以需要下载 ES 对应版本的中文分词器。

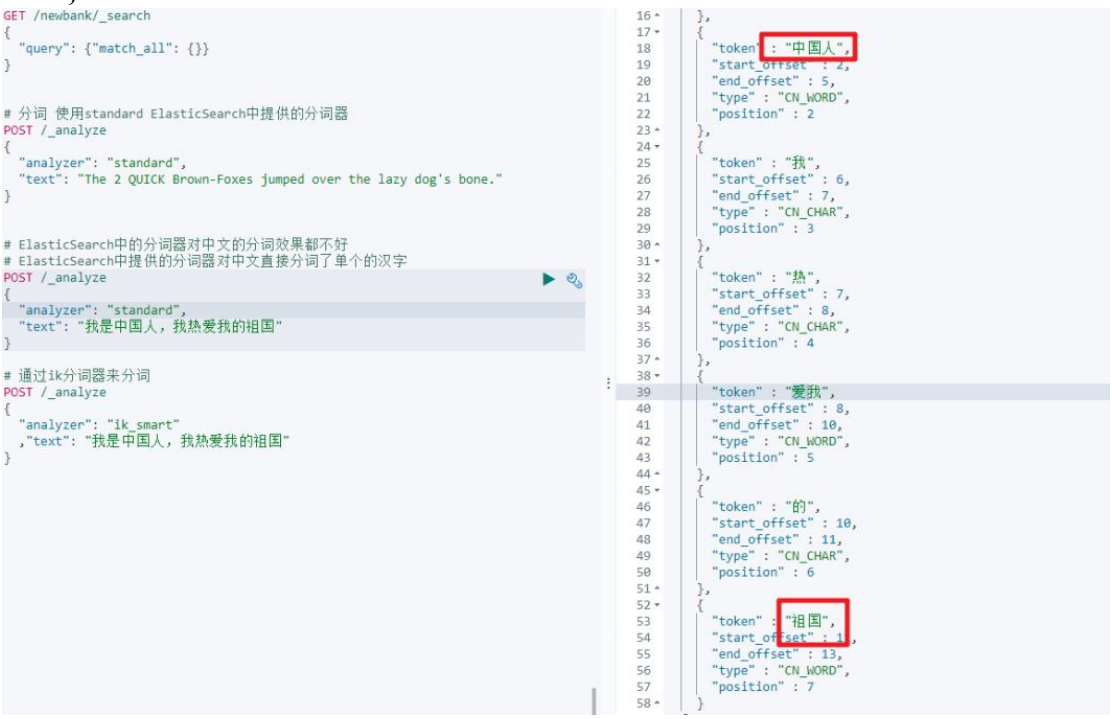
下载地址：https://github.com/medcl/elasticsearch-analysis-ik/releases

将解压后的后的文件夹放入 ES 根目录下的 plugins 目录下，重启 ES 即可使用。

- ik_max_word：会将文本做最细粒度的拆分
- ik_smart：会将文本做最粗粒度的拆分

(1) ik_smart 分词

{
"analyzer": "ik_smart" ,
"text": "我是中国人，我热爱我的祖国"
}



(2) ik_max_word

{
"analyzer": "ik_max_word" ,
"text": "我是中国人，我热爱我的祖国"
}

```
POST /_analyze
{
  "analyzer": "standard",
  "text": "The 2 QUICK Brown-Foxes jumped over the lazy dog's bone."
}


# ElasticSearch中的分词器对中文的分词效果都不好
# ElasticSearch中提供的分词器对中文直接分词了单个的汉字
POST /_analyze
{
  "analyzer": "standard",
  "text": "我是中国人，我热爱我的祖国"
}

# 通过ik分词器来分词 ik_smart 分词器
POST /_analyze
{
  "analyzer": "ik_smart"
  ,"text": "我是中国人，我热爱我的祖国"
}

# ik_max_word 分词

POST /_analyze
{
  "analyzer": "ik_max_word"
  ,"text": "我是中国人，我热爱我的祖国"
}
```

```
    "start_offset" : 1,
    "end_offset" : 2,
    "type" : "CN_CHAR",
    "position" : 1
  },
  {
    "token" : "中国人",
    "start_offset" : 2,
    "end_offset" : 5,
    "type" : "CN_WORD",
    "position" : 2
  },
  {
    "token" : "中国",
    "start_offset" : 2,
    "end_offset" : 4,
    "type" : "CN_WORD",
    "position" : 3
  },
  {
    "token" : "国人",
    "start_offset" : 3,
    "end_offset" : 5,
    "type" : "CN_WORD",
    "position" : 4
  },
  {
    "token" : "我",
    "start_offset" : 6,
    "end_offset" : 7,
    "type" : "CN_CHAR",
    "position" : 5
  },
  {
    "token" : "热爱",
    "start_offset" : 7,
    "end_offset" : 9,
    "type" : "CN_WORD",
    "position" : 6
  },
  {
    "token" : "爱我",
    "start_offset" : 8,
    "end_offset" : 10,
    "type" : "CN_WORD",
```

4. 自定义词库

```
{
  "analyzer": "ik_smart",
  "text":"潘嘎之交"
}
```

```
{
  "tokens" : [
    {
      "token" : "潘",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "CN_CHAR",
      "position" : 0
    },
    {
      "token" : "嘎",
      "start_offset" : 1,
      "end_offset" : 2,
      "type" : "CN_CHAR",
      "position" : 1
    },
    {
      "token" : "之交",
      "start_offset" : 2,
      "end_offset" : 4,
      "type" : "CN_WORD",
      "position" : 2
    }
  ]
```

仅仅可以得到每个字的分词结果，需要做的就是使分词器识别到潘嘎之交也是一个词语。

Ik 分词器配置：

(1) 本地字典配置

首先进入 ES 根目录中的 plugins 文件夹下的 ik 文件夹，进入 config 目录，创建 wangluo.dic 文件，写入潘嘎之交。同时打开 IKAnalyzer.cfg.xml 文件，将新建的 wangluo.dic 配置其中，重启 ES 服务器。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
        <comment>IK Analyzer 扩展配置</comment>
        <!--用户可以在这里配置自己的扩展字典 -->
        <entry key="ext_dict">wangluo.dic</entry>
         <!--用户可以在这里配置自己的扩展停止词字典-->
        <entry key="ext_stopwords"></entry>
        <!--用户可以在这里配置远程扩展字典 -->
        <!-- <entry key="remote_ext_dict">words_location</entry> -->
        <!--用户可以在这里配置远程扩展停止词字典-->
        <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
~
~
```

```
{
  "tokens" : [
    {
      "token" : "潘嘎之交",
      "start_offset" : 0,
      "end_offset" : 4,
      "type" : "CN_WORD",
      "position" : 0
    }
  ]
}
```

(2) Nginx 远程字典配置：

实现方式：同将远程词库放在 Nginx 中当静态资源使用，这样添加的词语不用重启两个服务都就可以直接使用。

需要的配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
        <comment>IK Analyzer 扩展配置</comment>
        <!--用户可以在这里配置自己的扩展字典 -->
        <entry key="ext_dict"></entry>
        <!--用户可以在这里配置自己的扩展停止词字典-->
        <entry key="ext_stopwords"></entry>
        <!--用户可以在这里配置远程扩展字典 -->
        <entry key="remote_ext_dict">http://192.168.81.100:8080/fenci/wangluo.txt</entry>
        <!--用户可以在这里配置远程扩展停止词字典-->
        <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
```

修改/usr/local/nginx/conf 中的 nginx.conf

```
server {
            listen    80;
            server_name    192.168.81.100;
            location /fenci/ {
                root es;
        }
```

}

并且在/usr/local/nginx/ 建/es/fenci/目录，目录下加 wangluo.txt

wangluo.txt 中编写关键词，每一行代表一个词。

潘嘎之交


```
{
  "tokens" : [
    {
      "token" : "潘嘎之交",
      "start_offset" : 0,
      "end_offset" : 4,
      "type" : "CN_WORD",
      "position" : 0
    }
  ]
}
```


5. 自定义分词器

虽然 Elasticsearch 带有一些现成的分析器，然而在分析器上 Elasticsearch 真正的强大之处在于，你可以通过在一个适合你的特定数据的设置之中组合字符过滤器、分词器、词汇单元过滤器来创建自定义的分析器。在分析与分析器我们说过，一个分析器就是在一个包里面组合了三种函数的一个包装器， 三种函数按照顺序被执行:

(1) 字符过滤器

字符过滤器用来整理一个尚未被分词的字符串。例如，如果我们的文本是 HTML 格式的，它会包含像 <p> 或者 这样的 HTML 标签，这些标签是我们不想索引的。我们可以使用 html 清除 字符过滤器 来移除掉所有的 HTML 标签，并且像把 Á 转换为相对应的 Unicode 字符 Á 这样，转换 HTML 实体。一个分析器可能有 0 个或者多个字符过滤器。

(2) 分词器

一个分析器必须有一个唯一的分词器。 分词器把字符串分解成单个词条或者词汇单元。标准分析器里使用的标准分词器 把一个字符串根据单词边界分解成单个词条，并且移除掉大部分的标点符号，然而还有其他不同行为的分词器存在。

例如，关键词分词器完整地输 接收到的同样的字符串，并不做任何分词。空格分词器只根据空格分割文本。正则分词器 据匹配正则表达式来分割文本。

(3) 词单元过滤器

经过分词，作为结果的词单元流 会按照指定的顺序通过指定的词单元过滤器。

词单元过滤器可以修改、添加或者移除词单元。我们已经提到过 lowercase 和 stop 词过滤器，但是在 Elasticsearch 里面还有很多可供选择的词单元过滤器。

词干过滤器把单词遏制为词干。ascii_folding 过滤器移除变音符，把一个像 "très" 这样的词转换为 "tres"。ngram 和 edge_ngram 词单元过滤器可以产生适合用于部分匹配或者自动补全的词单元。

```
{"index":{"_id":"1"}}
{"text":"我爱双截棍"}
{"index":{"_id":"2"}}
{"text":"我爱双棍"}

{
"query": {
"match": {
  "text": "双截棍"
}
}
}
```

发现查找之后还是按照默认 stand 进行分词的。

删除索引(已经确定的索引值是没有办法进行更改的),建立索引时对分词进行赋值。

```
{
"settings":{
      "number_of_shards":5,
      "number_of_replicas":1
},
"mappings": {
"properties": {
  "text":{
    "type": "text",
    "analyzer": "ik_smart" # "analyzer"分析器使用 ik_smasrt
  }
}
}
}
```

```
{
  "took" : 7,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 0.2876821,
    "hits" : [
      {
        "_index" : "07_ik",
        "_id" : "1",
        "_score" : 0.2876821,
        "_source" : {
          "text" : "我爱双截棍"
        }
      }
```


此时只会显示一次数据。

八. Logstash

1. 简介

Logstash 主要是用来日志的搜集、分析、过滤日志的工具，支持大量的数据获取方式。一般工作方式为 c/s 架构，client 端安装在需要收集日志的主机上，server 端负责将收到的各节点日志进行过滤、修改等操作在一并发往 elasticsearch 上去。

logstash 从输入源接受数据，直接发送达目的地，或者对数据进行过滤后在传输到目的地。

(1) Input：输入源

可以从文件中、存储中、数据库中抽取数据，Input 有两种选择一个是交给 Filter 进行过滤、修剪。另一个是直接交给 Output。logstash 支持多种输入源，如：stdin、beats、elasticsearch、file、jdbc、redis 等。

(2) Filter：过滤器

能够动态地转换和解析数据。可以通过自定义的方式对数据信息过滤、修剪。logstash 提供多种过滤器，过滤器可不使用，如：grok、geoip、date 等。

(3) Output：输出源

提供众多输出选择，您可以将数据发送到您要指定的地方，并且能够灵活地解锁众多下游用例。logstash 支持多种输出源，如：elasticsearch、redis、stdout 等。

2. 标准输入输出

下载地址：https://www.elastic.co/cn/downloads/past-releases#logstash

logstash 启动：

/usr/local/logstash-8.0.0//logstash -e 'input { stdin {} } output { stdout {} }'

启动 Logstash 后，等到看到“Pipeline main started”，然后 hello world 在命令提示符处输入。

Logstash 将时间戳和 IP 地址信息添加到消息中。通过在运行 Logstash 的 shell 中发出 CTRL-D 命令退出 Logstash。

```
hello world
{
      "@version" => "1",
         "event" => {
        "original" => "hello world"
    },
    "@timestamp" => 2022-03-18T02:12:31.884677Z,
          "host" => {
        "hostname" => "ubuntu"
    },
       "message" => "hello world"
}
```

一般不会这样子进行启动 需要写的配置太多，放到脚本即可。

3. 日志采集

3.1 输出到文件

./logstash -e 'input { stdin {} } output { file { path => "/tmp/logstash-test-%{+YYYY.MM.dd}.log"} }'

```
hello world
[2022-03-18T10:49:13,213][INFO ][logstash.outputs.file    ][main][176baa844bf910442e22f3c027ffa5be94fa99fa949eb7156c1c50ceb38f1f1a] Opening file {:path=>"/tmp/logstash-test-2022.03.18.log"}
omg may asd
asddasd
kasjdk
[2022-03-18T10:49:54,141][INFO ][logstash.outputs.file    ][main][176baa844bf910442e22f3c027ffa5be94fa99fa949eb7156c1c50ceb38f1f1a] Closing file /tmp/logstash-test-2022.03.18.log
```

```
{"@version":"1","@timestamp":"2022-03-18T02:49:13.007947Z","event":{"original":"hello world"},"host":{"hostname":"ubuntu"},"message":"hello world"}
{"@version":"1","@timestamp":"2022-03-18T02:49:21.218503Z","event":{"original":"omg may asd"},"host":{"hostname":"ubuntu"},"message":"omg may asd"}
{"@version":"1","@timestamp":"2022-03-18T02:49:24.795053Z","event":{"original":"asddasd"},"host":{"hostname":"ubuntu"},"message":"asddasd"}
{"@version":"1","@timestamp":"2022-03-18T02:49:26.325521Z","event":{"original":"kasjdk "},"host":{"hostname":"ubuntu"},"message":"kasjdk "}
```

开启 gzip 压缩输出

./logstash -e 'input { stdin {} } outpu{ file { path => "/tmp/logstash-test-%{+YYYY.MM.dd}.log.tar.gz" gzip => true } }'

```
>1.7}
[2022-03-18T10:57:35,145][INFO ][logstash.javapipeline    ][main] Pipeline started {"pipeline.id"=>"main"}
The stdin plugin is now waiting for input:
[2022-03-18T10:57:35,279][INFO ][logstash.agent           ] Pipelines running {:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
hello world
[2022-03-18T10:57:50,495][INFO ][logstash.outputs.file    ][main][4a95db19ae726174af240b5194fa7a0e8dead0848effe610f79489f428900c52] Opening file {:path=>"/tmp/logstash-test-2022.03.18.log.tar.gz"}
one
[2022-03-18T10:58:10,187][INFO ][logstash.outputs.file    ][main][4a95db19ae726174af240b5194fa7a0e8dead0848effe610f79489f428900c52] Closing file /tmp/logstash-test-2022.03.18.log.tar.gz
```

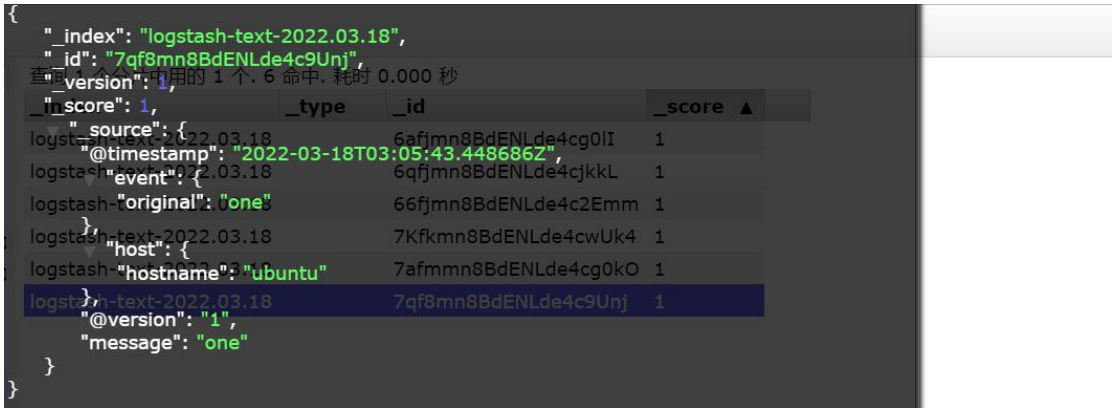
```
one@ubuntu:/tmp$ ls
elasticsearch-11166316894667076354  jruby-47366
elasticsearch-12897126256234516094  logstash-test-2022.03.18.log
elasticsearch-13053451586737421 67  logstash-test-2022.03.18.log.tar.gz
elasticsearch-13261701368657135795  systemd-private-8ef329627e7d4a6c9e5b24314b5eb0a8-bolt.service-f3QPBx
elasticsearch-14184027838900112747  systemd-private-8ef329627e7d4a6c9e5b24314b5eb0a8-colord.service-wAJsLx
elasticsearch-2819123109524670160   systemd-private-8ef329627e7d4a6c9e5b24314b5eb0a8-ModemManager.service-yDuL8l
elasticsearch-3702912289647090729   systemd-private-8ef329627e7d4a6c9e5b24314b5eb0a8-rtkit-daemon.service-ygFX8r
elasticsearch-4635856041783498029   systemd-private-8ef329627e7d4a6c9e5b24314b5eb0a8-systemd-resolved.service-JxxM90
elasticsearch-7210717250902930549   systemd-private-8ef329627e7d4a6c9e5b24314b5eb0a8-systemd-timesyncd.service-R2GazL
elasticsearch-873578878826287209    VMwareDnD
elasticsearch-9769924299984003963   vmware-root_894-2730693566
```

3.2 输出到 ES

./logstash -e 'input { stdin {} } output { elasticsearch { hosts => ["192.168.81.100:9200"] index => "logstash-test-%{+YYYY.MM.dd}" } }'

输出日志到 ES 中并且按照日期创建索引 logstash-test-%{+YYYY.MM.dd}" }

```
ers"=>4, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>50, "pipeline.max_inflight"=>500, "pipeline.sources"=>["config string"], :thread=>"#<Thread:0x62a7d7c0 run>"}
[2022-03-18T11:05:13,771][INFO ][logstash.javapipeline    ][main] Pipeline Java execution initialization time {"seconds"=>1.38}
[2022-03-18T11:05:13,875][INFO ][logstash.javapipeline    ][main] Pipeline started {"pipeline.id"=>"main"}
The stdin plugin is now waiting for input:
[2022-03-18T11:05:13,966][INFO ][logstash.agent           ] Pipelines running {:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
one
```



3.3 指定文件输出

```
input {
        file{
                path => "/export/logs/gateway/gateway-provider.%{+YYYY-MM-dd}"
                 type => "elasticsearch-java-log"
                 start_position => "beginning"
                 stat_interval => "2"
                 codec => multiline {
                         pattern => "^\["     #以"["开头进行正则匹配
                         negate => "true"   #正则匹配成功
                         what => "previous"   #和前面的内容进行合并
                 }
        }
}
output {
        if [type] == "elasticsearch-java-log" {
                 elasticsearch {
                         hosts => ["10.159.42.37:9200"]
                         index => "gateway-log-%{+YYYY.MM.dd}"
                 }
        }
}
```

后台运行脚本

nohup /usr/local/logstash/bin/logstash -f /etc/logstash/conf.d/logstash.conf -w 8 -b 1000 > /dev/null 2>&1 &

4. 过滤器

4.1 rok 正则捕获

grok 是一个十分强大的 logstash filter 插件，可以通过正则解析任意文本，将非结构化日志数据弄成结构化和方便查询的结构。是目前 logstash 中解析非结构化日志数据最好的方式。在配置文件中 grok 其实是使用正则表达式来进行过滤的。

grok 的语法规则是：

%{语法：语义}

“语法”指的是匹配的模式。例如使用 NUMBER 模式可以匹配出数字，IP 模式则会匹配出 192.168.81.100 这样的 IP 地址。

过滤 ip：

```
input {
        stdin {
              }
}
filter{
        grok{
                match => {"message" => "%{IPV4:ip}"}
        }
output {
        stdout {
                }
}
```

```
192.168.81.100 [07/Feb/2018:16:24:19 +0800]\"GET /HTTP/1.1\" 403 5039
{
      "@version" => "1",
    "@timestamp" => 2022-03-18T06:46:39.984752Z,
         "event" => {
      "original" => "192.168.81.100 [07/Feb/2018:16:24:19 +0800]\\\"GET /HTTP/1.1\\\" 403 5039"
    },
           "ip" => "192.168.81.100",
         "host" => {
      "hostname" => "ubuntu"
    },
      "message" => "192.168.81.100 [07/Feb/2018:16:24:19 +0800]\\\"GET /HTTP/1.1\\\" 403 5039"
}
```

过滤时间戳：

```
input {
        stdin {
              }
}
filter{
        grok{
                match => {"message" => "%{IPV4:ip}\ \[%{HTTPDATE:timestamp}\]"}
        }
}
output {
        stdout {
                }
}
```

```
192.168.81.100 [07/Feb/2018:16:24:19 +0800]\"GET /HTTP/1.1\" 403 5039
{
           "ip" => "192.168.81.100",
     "@version" => "1",
   "@timestamp" => 2022-03-18T06:56:09.808364Z,
    "timestamp" => "07/Feb/2018:16:24:19 +0800",
         "host" => {
      "hostname" => "ubuntu"
    },
        "event" => {
      "original" => "192.168.81.100 [07/Feb/2018:16:24:19 +0800]\\\"GET /HTTP/1.1\\\" 403 5039"
    },
      "message" => "192.168.81.100 [07/Feb/2018:16:24:19 +0800]\\\"GET /HTTP/1.1\\\" 403 5039"
}
```

过滤报头：

```
input {
        stdin {
               }
}
filter{
        grok{
                match => {"message" => "\ %{QS:referrer}\ "}
        }
}
output {
        stdout {
                }
}
```

```
192.168.81.100 - - [07/Feb/2018:16:24:19 +0800] "GET /HTTP/1.1" 403 5039
{
      "@version" => "1",
          "host" => {
        "hostname" => "ubuntu"
    },
      "referrer" => "\"GET /HTTP/1.1\"",
         "event" => {
      "original" => "192.168.81.100 - - [07/Feb/2018:16:24:19 +0800] \"GET /HTTP/1.1\" 403 5039"
    },
       "message" => "192.168.81.100 - - [07/Feb/2018:16:24:19 +0800] \"GET /HTTP/1.1\" 403 5039",
    "@timestamp" => 2022-03-18T07:09:53.551357Z
}
```

4.2 date 插件

timestamp 字段，表示取出日志中的时间。但是在显示的时候除了显示你指定的 timestamp 外，还有一行是@timestamp 信息，这两个时间是不一样的，@timestamp 表示系统当前时间。两个时间并不是一回事，在 ELK 的日志处理系统中，@timestamp 字段会被 elasticsearch 用到，用来标注日志的生产时间，如此一来，日志生成时间就会发生混乱，要解决这个问题，需要用到另一个插件，即 date 插件，这个时间插件用来转换日志记录中的时间字符串，变成 Logstash::Timestamp 对象，然后转存到@timestamp 字段里面。

```
input {
        stdin {
               }
}
filter{
        grok{
                match => {"message" => "%{IPV4:ip}\ \[%{HTTPDATE:timestamp}\]"}
        }
        date{
                match => ["timestamp","dd/MMM/yyyy:HH:mm:ss Z"]
        }
}
output {
        stdout {
                }
}
```

```
192.168.81.100 [17/Mar/2022:15:28:19 +0800] "GET /HTTP/1.1" 403 5039
{
    "@timestamp" => 2022-03-17T07:28:19Z,
         "event" => {
      "original" => "192.168.81.100 [17/Mar/2022:15:28:19 +0800] \"GET /HTTP/1.1\" 403 5039"
    },
      "@version" => "1",
          "host" => {
        "hostname" => "ubuntu"
    },
            "ip" => "192.168.81.100",
     "timestamp" => "17/Mar/2022:15:28:19 +0800",
       "message" => "192.168.81.100 [17/Mar/2022:15:28:19 +0800] \"GET /HTTP/1.1\" 403 5039"
}
```

会发现@timestamp 时间转换成功，还有一点就是在时间少 8 个小时。

date 插件对于排序事件和回填旧数据尤其重要，它可以用来转换日志记录中的时间字段，变成 Logstash：：timestamp 对象，然后转存到@timestamp 字段里面。

一方面由于 Logstash 会给收集到的每条日志自动打上时间戳（即@timestamp），但是这个时间戳记录的是 input 接收数据的时间，而不是日志生成的时间（因为日志生成时间与 input 接收的时间肯定不同），这样就可能导致搜索数据时产生混乱。

另一方面，在上面那段 rubydebug 编码格式的输出中，@timestamp 字段虽然已经获取了 timestamp 字段的时间，但是仍然比北京时间晚了 8 个小时，这是因为在 Elasticsearch 内部，对时间类型字段都是统一采用 UTC 时间，而日志统一采用 UTC 时间存储，是国际安全、运维界的一个共识。其实这并不影响什么，因为 ELK 已经给出了解决方案，那就是在 Kibana 平台上，程序会自动读取浏览器的当前时区，然后在 web 页面自动将 UTC 时间转换为当前时区的时间。

mutate 插件是 logstash 另一个非常重要的插件，它提供了丰富的基础类型数据处理能力，包括重命名、删除、替换、修改日志事件中的字段。我们这里举几个常用的 mutate 插件：字段类型转换功能 covert、正则表达式替换字段功能 gsub、分隔符分隔字符串为数值功能 split、重命名字段功能 rename、删除字段功能 remove_field。

字段类型转换 convert

```
input {
        stdin {
              }
}
filter{
        grok{
                match => {"message" => "%{IPV4:ip}"}
                remove_field => ["message"]
        }
        mutate{
                convert => ["ip","string"]
        }
}
output {
        stdout {
                }
}
```

```
192.168.81.100 [17/Mar/2022:15:28:19 +0800] "GET /HTTP/1.1" 403 5039
{
    "@timestamp" => 2022-03-18T08:38:15.032968Z,
         "event" => {
        "original" => "192.168.81.100 [17/Mar/2022:15:28:19 +0800] \"GET /HTTP/1.1\" 403 5039"
    },
      "@version" => "1",
          "host" => {
        "hostname" => "ubuntu"
    },
            "ip" => "192.168.81.100"
}
```

ip 变成字符串类型。

4.3 remove_field 的用法

remove_field 的作用就是去重。不管是我们要输出什么样子的信息，都是有两份数据，即 message 里面是一份，HTTPDATE 或者 IP 里面也有一份，这样子就造成了重复，过滤的目的就是筛选出有用的信息，重复的去除。

```
input {
        stdin {
              }
}
filter{
        grok{
                match => {"message" => "%{IP:ip_address}"}
                remove_field => ["message"]
        }

}
output {
        stdout {
                }
}
```

```
192.168.81.100 [17/Mar/2022:15:28:19 +0800] "GET /HTTP/1.1" 403 5039
{
      "@version" => "1",
         "event" => {
        "original" => "192.168.81.100 [17/Mar/2022:15:28:19 +0800] \"GET /HTTP/1.1\" 403 5039"
    },
          "host" => {
        "hostname" => "ubuntu"
    },
    "ip_address" => "192.168.81.100",
    "@timestamp" => 2022-03-18T07:51:06.155081Z
}
```

信息全部显示

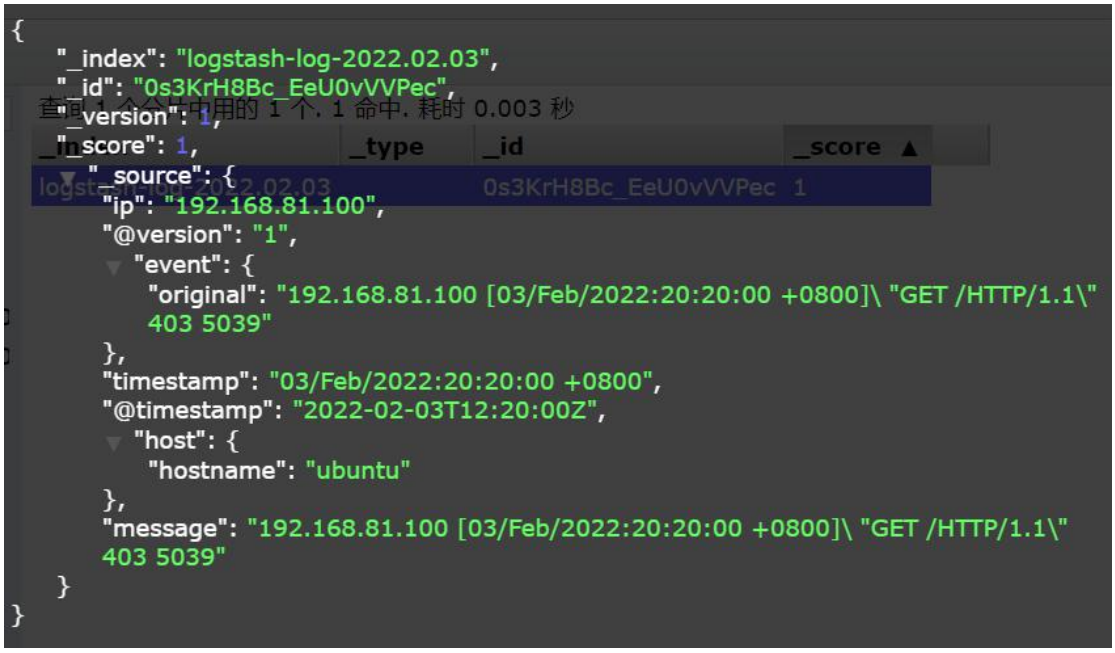
```
input {
        stdin {
              }
}
filter{
        grok{
            match => {"message" => "%{IP:ip_address}\ \[%{HTTPDATE:timestamp}\]\ %{QS:referrer}\ %{NUMBER:status}\ %{NUMB
ER:bytes}"}
        }
        date{
            match => ["timestamp","dd/MMM/yyyy:HH:mm:ss Z"]
        }
        mutate{
            remove_field => ["message","timestamp"]
        }

}
output {
        stdout {
              }
}
```

```
192.168.81.100 [17/Mar/2022:15:28:19 +0800] "GET /HTTP/1.1" 403 5039
{
        "status" => "403",
         "bytes" => "5039",
      "@version" => "1",
    "@timestamp" => 2022-03-17T07:28:19Z,
      "referrer" => "\"GET /HTTP/1.1\"",
         "event" => {
        "original" => "192.168.81.100 [17/Mar/2022:15:28:19 +0800] \"GET /HTTP/1.1\" 403 5039"
    },
    "ip_address" => "192.168.81.100",
          "host" => {
        "hostname" => "ubuntu"
    }
}
```

4.4 过滤器解析日志存到 es

```
input {
        stdin {}
}
filter{
        grok{
                match => {"message" => "%{IPV4:ip}\ \[%{HTTPDATE:timestamp}\]"}
        }
        date{
                match => ["timestamp","dd/MMM/yyyy:HH:mm:ss Z"]
        }
}
output {
        elasticsearch {
                        hosts => "192.168.81.100"
                        index => "logstash-log-%{+YYYY.MM.dd}"
                         }
        stdout { codec => rubydebug }
}
```



5. 数据迁移

Logstash 是一个开源数据收集引擎，具有实时管道功能。Logstash 可以动态地将来自不同数据源的数据统一起来，并将数据标准化到你所选择的目的地。

使用 Logstash 将 MySQL 数据迁移到 ElasticSearch 中。

5.1 配置文件

需要 mysql-connector-java 驱动程序

mysql-connector-java 下载地址

https://cloud.tencent.com/developer/article/1688598

```
input {
    jdbc {
        # 设置 MySql/MariaDB 数据库 url 以及数据库名称
        jdbc_connection_string => "jdbc:mysql://192.168.81.100:3306/imap_database?useUnicode=true&allowMultiQue
```

```
rie=true&characterEncoding=utf-8&serverTimezone=UTC&useSSL=false"
        # 用户名和密码
        jdbc_user => "root"
        jdbc_password => "123456"
        # 数据库驱动 mysql-connector-java-8.0.19.jar 所在位置，可以是绝对路径或者相对路径
        jdbc_driver_library => "/usr/local/logstash-8.0.0/my/mysql-connector-java-8.0.28/mysql-connector-java-8.0.28.jar"
        # 驱动类名
        jdbc_driver_class => "com.mysql.cj.jdbc.Driver"
         # 是否开启分页,ture 为开启
        jdbc_paging_enabled => true
         # 分页每页数量
        jdbc_page_size => "50000"
        # 设置时区
        jdbc_default_timezone =>"Asia/Shanghai"
        # 执行的 sql 文件路径
        statement_filepath => "/usr/local/logstash-8.0.0/my/jdbc.sql"
        #使用这个可以直接写 sql 语句，但是复杂的语句最好是写在文件内
        #statement =>
        # 设置定时任务间隔  含义：分、时、天、月、年，全部为*默认含义为每分钟跑一次任务
        schedule => "* * * * *"
        #是否需要记录某个字段值,如果为 true,我们可以自定义要记录的数据库某个字段值，例如 id 或 date 字段。如果为 false，记录的是上次执行的标记，默认是一个 timestamp
        use_column_value => true
      #如果 use_column_value 为真,需配置此参数. 指定增量更新的字段名。当然该字段必须是递增的，比如 id 或 date 字段。
        tracking_column => "id"
        #记录上次执行字段值路径。我们可以在 sql 语句中这么写：WHERE ID > :last_sql_value。其中 :sql_last_value 取得就是该文件中的值，这个 last_id 会以文件形式存在
        # last_run_metadata_path => "/usr/local/nbin/logstash-7.12.0/my/jdbc.sql"
        # tracking_column 对应字段的类型，只能选择 timestamp 或者 numeric(数字类型)，默认 numeric，所以可以不写这个配置
        tracking_column_type => "numeric"
       #如果为 true，每次会记录所更新的字段的值,并保存到 last_run_metadata_path 指定的文件中
        record_last_run => true
        # 是否清除 last_run_metadata_path 的记录，true 则每次都从头开
```

始查询所有的数据库记录

```
                clean_run => false
                 # 是否将字段名称转小写。默认是 true。这里注意 Elasticsearch 是区分大小写的
                lowercase_column_names => false
            }
        }
        output {
            elasticsearch {
                 # es 地址  集群数组 hosts => ["127.0.0.1:9200","127.0.0.1:9201"]
                hosts => ["192.168.81.100:9200","192.168.81.101:9200"]
               # 同步的索引名必须要有@timestamp    不然 yyyyMM 不起效
               index => "pop3_database"
                # 设置_docID 和数据相同
                document_id => "%{id}"
            }
          # 日志输出形式设置
            stdout {
                codec => json
                #codec => rubydebug
            }
        }
```

5.2 配置查询语句

```
SELECT * FROM mail_2022_01
```

5.3 Logstash 检查并启动

```
./logstash -f /usr/local/logstash-8.0.0/my/jdbc.conf -t
```