八六三软件股份有限公司
863 Software Co., Ltd.

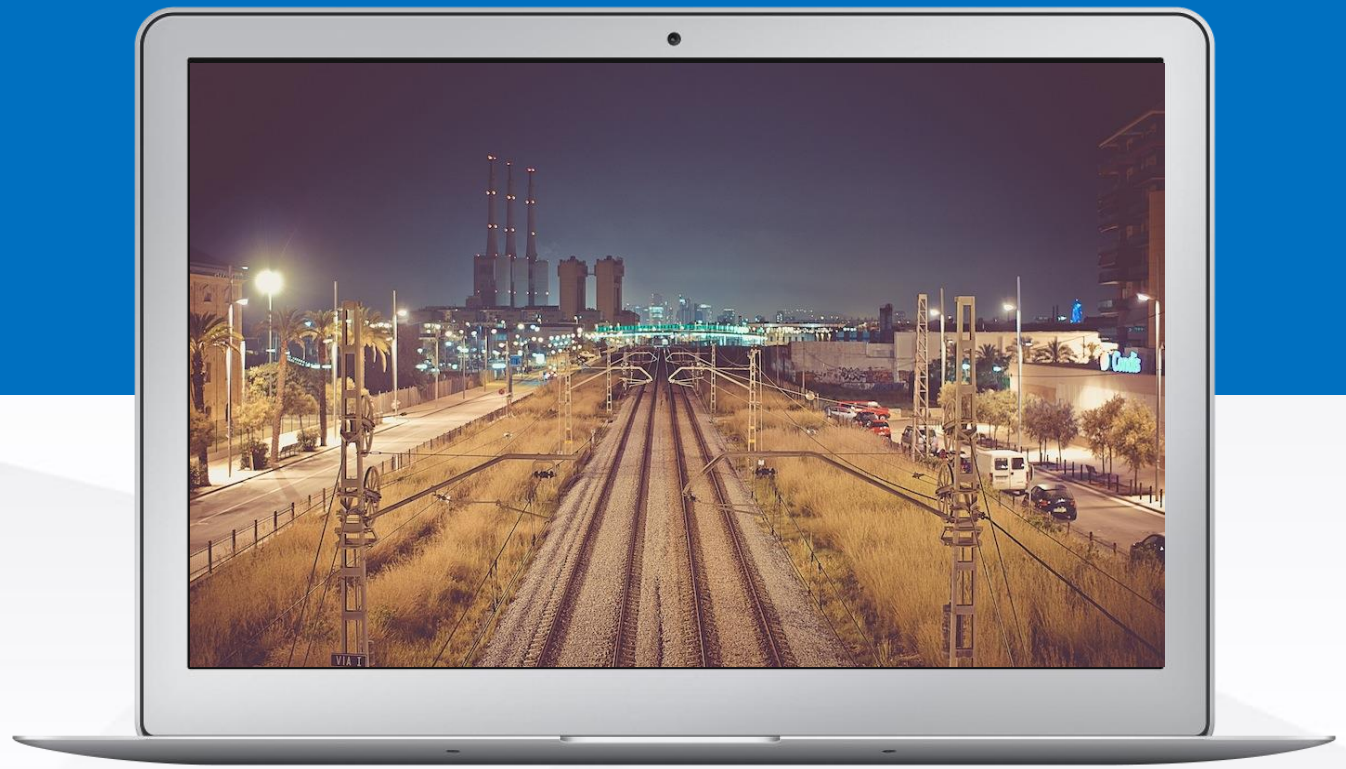
# ElasticSearch

# 目录

八六三软件股份有限公司
863 Software Co., Ltd.

八六三软件股份有限公司
863 Software Co., Ltd.

# 一、ElasticSearch概述



# 什么是搜索



我们比如说想找寻任何的信息的时候，就会上百度去搜索一下。比如说找一部自己喜欢的电影，或者说找一本喜欢的书，或者找一条感兴趣的新闻（提到搜索的第一印象）。

**百度 != 搜索**

1）互联网的搜索：电商网站，招聘网站，新闻网站，APP

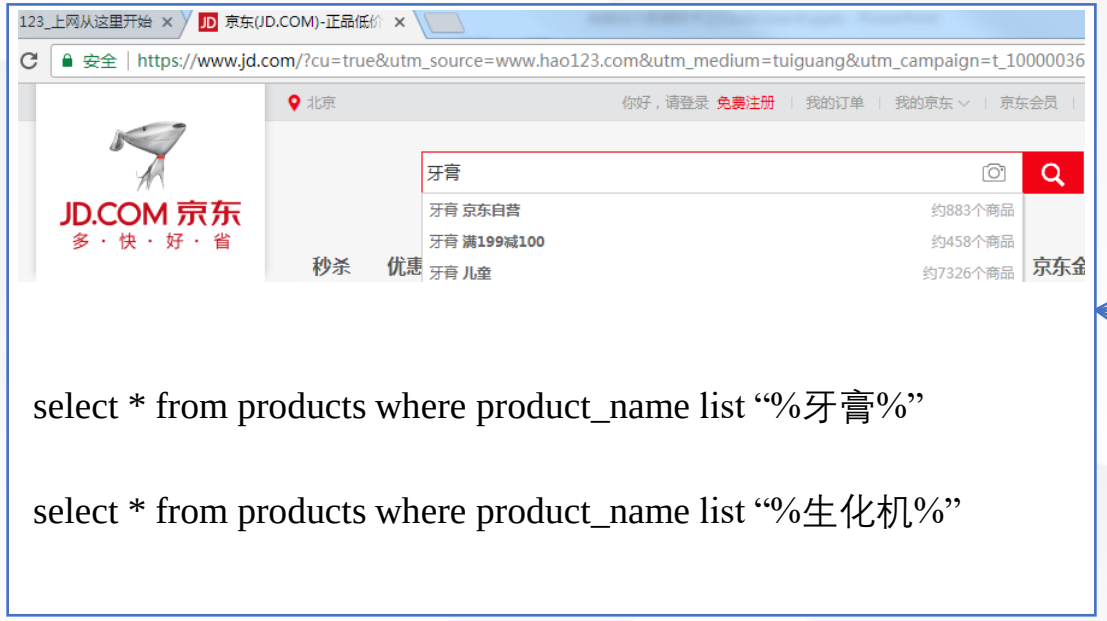
2）IT系统的搜索：OA软件，办公自动化软件，会议管理，日程管理，项目管理。

**搜索，**就是在任何场景下，找寻你想要的信息，这个时候，会输入一段你要搜索的关键字，然后就期望找到**这个关键字相关**的有些信息

# 数据库搜索？

八六三软件股份有限公司
863 Software Co., Ltd.

京东商城搜索框




select * from products where product_name list “%牙膏%”

select * from products where product_name list “%生化机%”

逐条遍历

京东商城后台商品表

| 商品id | 商品名称 | 商品描述 |
| --- | --- | --- |
| 1 | 高露洁牙膏 | 1千字的商品描述 |
| 2 | 中华牙膏 | 1千字的商品描述 |
| 3 | 佳洁士牙膏 | 1千字的商品描述 |
| 4 | 其他牙膏 | 1千字的商品描述 |
| n | … | |

用数据库来实现搜索，显然是不太靠谱的。通俗的说就是性能差，速度慢。

1）比如说“商品描述”字段的长度，有长达数千个，甚至数万个字符，这个时候，每次都要对每条记录的所有文本进行扫描，判断包不包含我指定的这个关键词（比如说“牙膏”），效率非常低。

2）还不能将搜索词拆分开来，尽可能去搜索更多的符合你的期望的结果，比如输入“生化机”，就搜索不出来“生化危机”。

# 什么是全文检索和Lucene?



1）全文检索，倒排索引

**全文检索**是指计算机索引程序通过扫描文章中的每一个词，对每一个词建立一个索引，**指明该词在文章中出现的次数和位置**，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式。这个过程类似于通过字典中的检索字表查字的过程。全文搜索搜索引擎数据库中的数据。

2）lucene

就是一个jar包，里面包含了封装好的各种建立倒排索引，以及进行搜索的代码，包括各种算法。我们就用java开发的时候，引入lucene jar，然后基于lucene的api进行去进行开发就可以了。

# 什么是全文检索和Lucene?

八六三软件股份有限公司
863 Software Co., Ltd.

## 倒排索引原理简介

1 数据库里的数据

商品描述1
商品描述2
商品描述3
商品描述4

2 切词

生化危机电影
生化危机海报
生化危机文章
生化危机新闻

生 化 危 机 电影
生 化 危 机 海报
生 化 危 机 文章
生 化 危 机 新闻

3 倒排索引

| 关键词 | ids |
| --- | --- |
| 生 | 1,2,3,4 |
| 化 | 1,2,3,4 |
| 危 | 1,2,3,4 |
| 机 | 1,2,3,4 |
| 电影 | 1 |
| 海报 | 2 |
| 文章 | 3 |
| 新闻 | 4 |

查找：生化机

返回1,2,3,4商品

总结1：数据库里的数据，一共100万条，按照之前的思路，其实就要扫描100万次，而且每次扫描，都需要匹配文本所有的字符，确认是否包含搜索的关键词，而且还不能将搜索词拆解开来进行检索。

总结2：利用倒排索引，进行搜索的话，假设100万条数据，拆分出来的词语，假设有1000万个词语，那么在倒排索引中，就有1000万行，我们可能并不需要搜索1000万次，很可能，在搜索到第一次的时候，我们就可以找到这个搜索词对应的数据。也可能第100次，或者第1000次。

# 什么是Elasticsearch?

八六三软件股份有限公司 863 Software Co., Ltd.

Elasticsearch 基于 lucene，隐藏复杂性，提供简单易用的Restful API接口、Java API接口（还有其他语言的API接口）。

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。同时又是 Elastic Stack 的核心。

- **全文检索：**将非结构化数据中的一部分信息提取出来,重新组织,使其变得有一定结构,然后对此有一定结构的数据进行搜索,从而达到搜索相对较快的目的。
- **结构化检索：**我想搜索商品分类为日化用品的商品都有哪些，select * from products where category_id='日化用品'
- **数据分析：**电商网站，最近7天牙膏这种商品销量排名前10的商家有哪些；新闻网站，最近1个月访问量排名前3的新闻版块是哪些

Elasticsearch

# 什么是Elasticsearch?

八六三软件股份有限公司
863 Software Co., Ltd.


地址：https://www.elastic.co/cn/products/elasticsearch

# Elasticsearch的适用场景



1）维基百科，类似百度百科，牙膏，牙膏的维基百科，全文检索，高亮，搜索推荐。

2）The Guardian（国外新闻网站），类似搜狐新闻，用户行为日志（点击，浏览，收藏，评论）+ 社交网络数据（对某某新闻的相关看法），数据分析，给到每篇新闻文章的作者，让他知道他的文章的公众反馈（好，坏，热门，垃圾，鄙视，崇拜）。

3）Stack Overflow（国外的程序异常讨论论坛），IT问题，程序的报错，提交上去，有人会跟你讨论和回答，全文检索，搜索相关问题和答案，程序报错了，就会将报错信息粘贴到里面去，搜索有没有对应的答案

4）GitHub（开源代码管理），搜索上千亿行代码。

5）国内：站内搜索（电商，招聘，门户，等等），IT系统搜索（OA，CRM，ERP，等等），数据分析（ES热门的一个使用场景）。

# Elasticsearch特点



1）可以作为一个大型分布式集群（数百台服务器）技术，处理PB级数据，服务大公司；也可以运行在单机上，服务小公司

2）Elasticsearch 不是什么新技术，主要是将全文检索、数据分析以及分布式技术，合并在了一起，才形成了独一无二的ES；lucene（全文检索），商用的数据分析软件（也是有的），分布式数据库（mycat）

3）对开发者而言，是开箱即用的，非常简单，作为中小型的应用，直接3分钟部署ES，就可以作为生产环境的系统来使用了，数据量不大，操作不是太复杂

4）数据库的功能面对很多领域是不够用的（事务，还有各种联机事务型的操作）；特殊的功能，比如全文检索，同义词处理，相关度排名，复杂数据分析，海量数据的近实时处理；Elasticsearch作为传统数据库的一个补充，提供了数据库所不能提供的很多功能

# Elasticsearch学习资料

八六三软件股份有限公司
863 Software Co., Ltd.

- 官网

https://www.elastic.co/cn/

- ElasticSearch权威指南

https://legacy.gitbook.com/book/looly/elasticsearch-the-definitive-guide-cn/details

- Elastic中文社区

https://elasticsearch.cn/explore/category-18

# Kibana

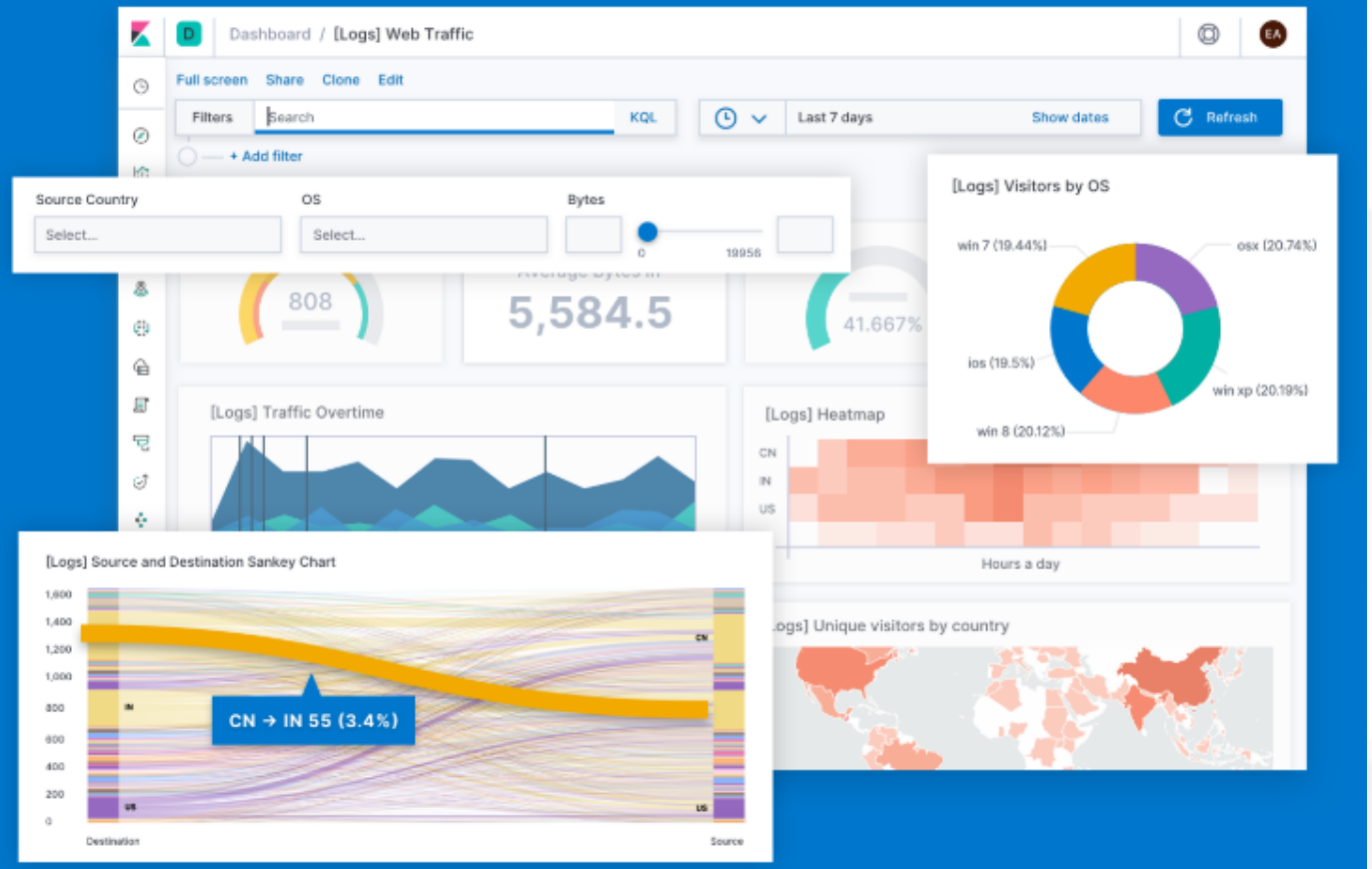
八六三软件股份有限公司
863 Software Co., Ltd.

KIBANA

## 您了解 Elastic Stack 的窗口

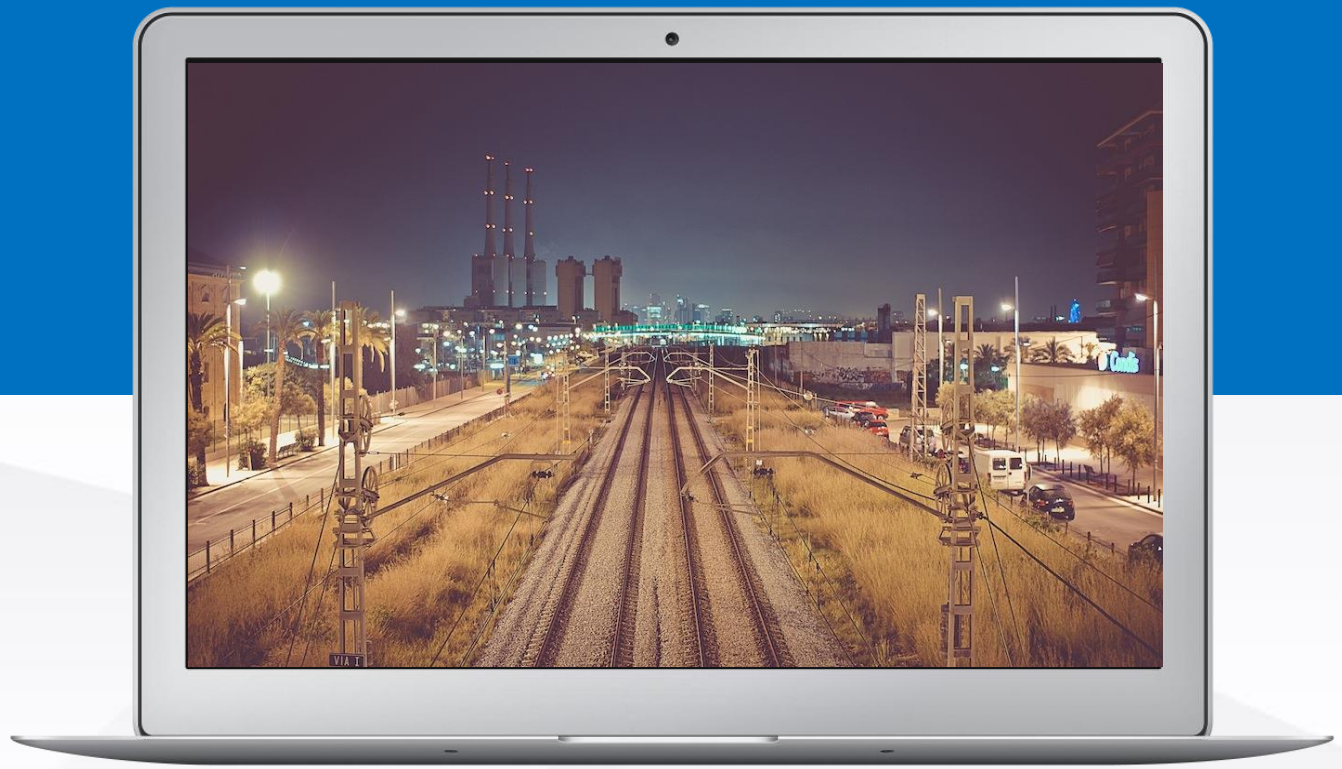
通过 Kibana，您可以对自己的 Elasticsearch 进行可视化，还可以在 Elastic Stack 中进行导航，这样您便可以进行各种操作了，从跟踪查询负载，到理解请求如何流经您的整个应用，都能轻松完成。

亦可下载 Kibana。



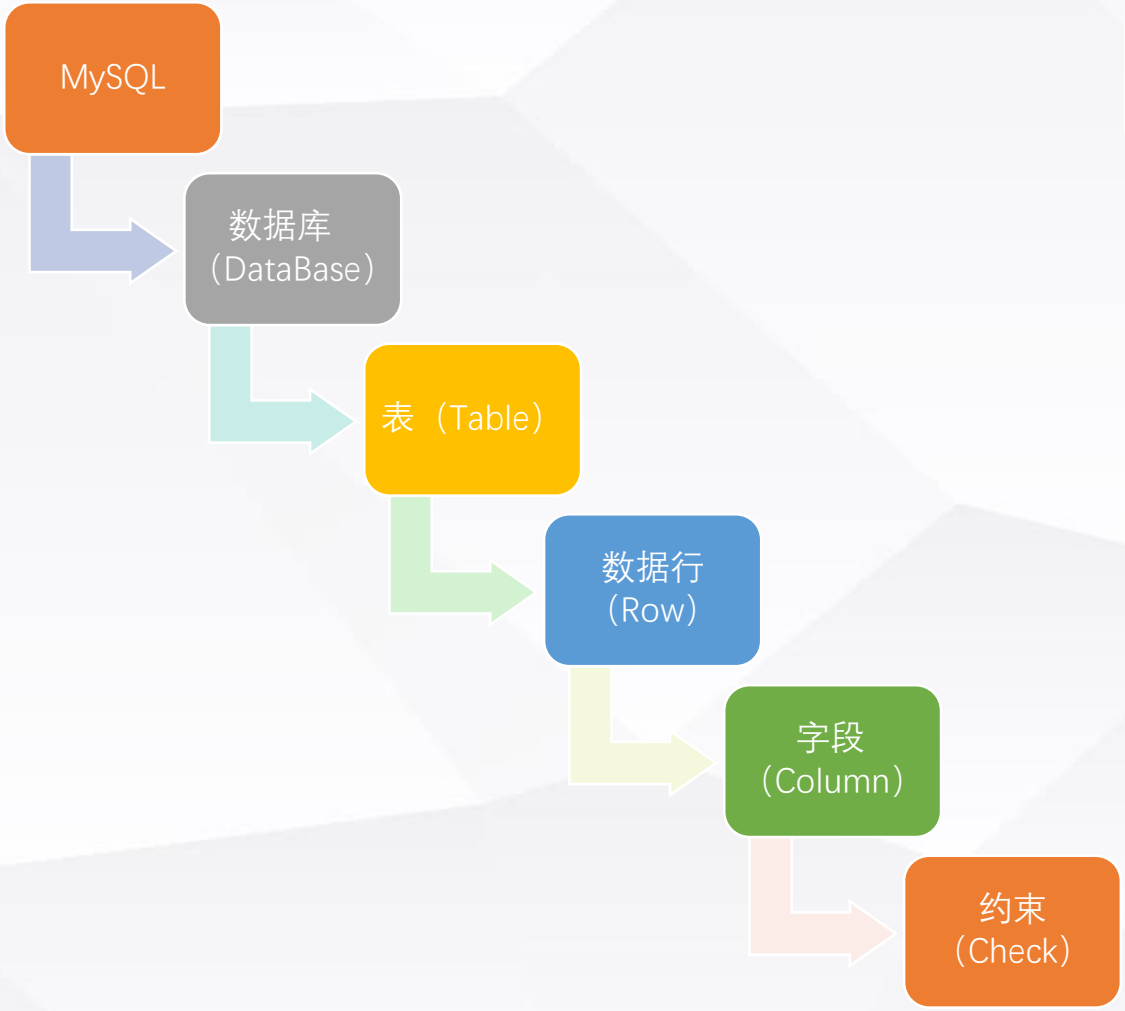

https://www.elastic.co/cn/products/kibana

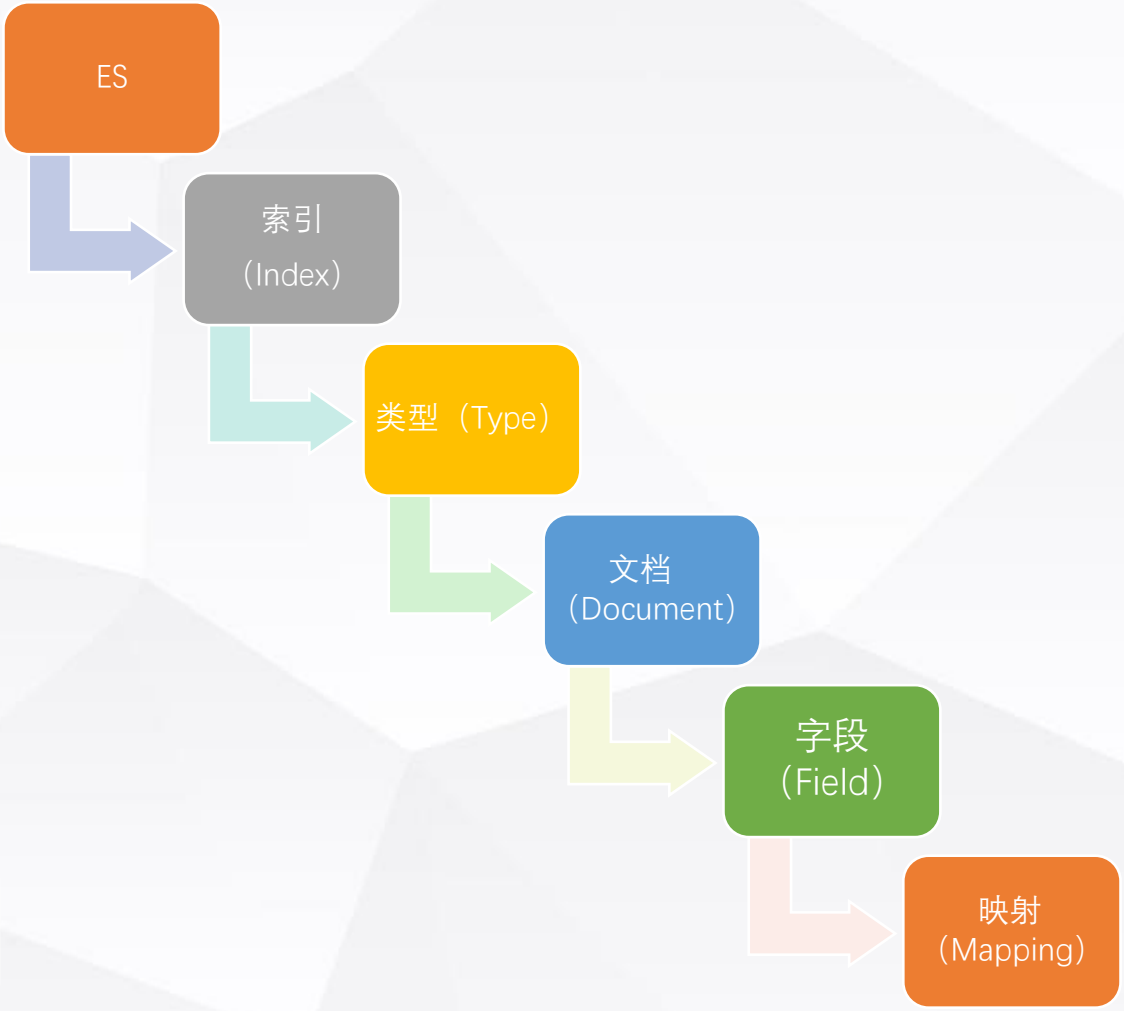
# 二、ElasticSearch架构



# elasticsearch与数据库的类比





# Index（索引-数据库）



索引包含一堆有相似结构的文档数据，比如可以有一个客户索引，商品分类索引，订单索引，索引有一个名称。一个index包含很多document，一个index就代表了一类类似的或者相同的document。

比如说建立一个product_index，商品索引，里面可能就存放了所有的商品数据，所有的商品 document。

# type（类型-表）



每个索引里都可以有一个或多个type，type是index中的一个逻辑数据分类，一个type下的document，都有相同的field和特殊的field。**在新版本中 type 类型被弱化。**

博客系统，有一个索引，可以定义用户数据type，博客数据type，评论数据type。

电商系统，里面存放了所有的商品数据，商品document。但是商品分很多种类，每个种类的document的field可能不太一样，比如说电器商品，可能还包含一些诸如售后时间范围这样的特殊field；生鲜商品，还包含一些诸如生鲜保质期之类的特殊field。

type，日化商品type，电器商品type，生鲜商品type

日化商品type：商品ID、商品名、描述、分类ID、分类名

电器商品type：商品ID、商品名、描述、分类ID、分类名、售后时间

生鲜商品type：商品ID、商品名、描述、分类ID、分类名、保质期

# document（文档-行）



文档是 ES 中的最小数据单元，一个document可以是一条客户数据，一条商品分类数据，一条订单数据，通常用JSON数据结构表示，每个index下的type中，都可以去存储多个document。

# Field（字段-列）



Field是 ES 的最小单位。一个document里面有多个field，每个field就是一个数据字段。

```
product document
{
  "product_id": "1",
  "product_name": "高露洁牙膏",
  "product_desc": "高效美白",
  "category_id": "2",
  "category_name": "日化用品"
}
```

# mapping（映射-约束）



数据如何存放到索引对象上，需要有一个映射配置，包括：数据类型、是否存储、是否分词等。

创建一个名为blog的Index。Type不用单独创建，在创建Mapping 时指定就可以。

Mapping用来定义Document中每个字段的类型，即所使用的analyzer、是否索引等属性，非常关键等。

```
client.indices.putMapping({
  index : 'blog',
  type : 'article',
  body : {
    article: {
      properties: {
        id: {
          type: 'string',
          analyzer: 'ik',
          store: 'yes',
        },
        content: {
          type: 'string',
          analyzer: 'ik',
          store: 'yes',
        }
      }
    }
  }
});
```

# ElasticSearch物理架构



- Cluster（集群）

集群包含多个节点，每个节点属于哪个集群是通过一个配置（集群名称，默认是elasticsearch）来决定的，对于中小型应用来说，刚开始一个集群一个节点很正常。

- Node（节点）

集群中的一个节点，节点也有一个名称（默认是随机分配的），节点名称很重要（在执行运维管理操作的时候），默认节点会去加入一个名称为“elasticsearch”的集群，如果直接启动一堆节点，那么它们会自动组成一个elasticsearch集群，当然一个节点也可以组成一个elasticsearch集群。

- 分片

当我们的文档量很大时，由于内存和硬盘的限制，同时也为了提高ES的处理能力、容错能力及高可用能力，我们将索引分成若干分片，每个分片可以放在不同的服务器，这样就实现了多个服务器共同对外提供索引及搜索服务。一个搜索请求过来，会分别从各各分片去查询，最后将查询到的数据合并返回给用户。

# ElasticSearch物理架构



- 副本

为了提高ES的高可用同时也为了提高搜索的吞吐量，我们将分片复制一份或多份存储在其它的服务器，这样即使当前的服务器挂掉了，拥有副本的服务器照常可以提供服务。

- 主结点

一个集群中会有一个或多个主结点，主结点的作用是集群管理，比如增加节点，移除节点等，主结点挂掉后ES会重新选一个主结点。

- 结点转发

每个结点都知道其它结点的信息，我们可以对任意一个结点发起请求，接收请求的结点会转发给其它结点查询数据。

# elasticsearch存入和搜索数据机制

八六三软件股份有限公司 863 Software Co., Ltd.

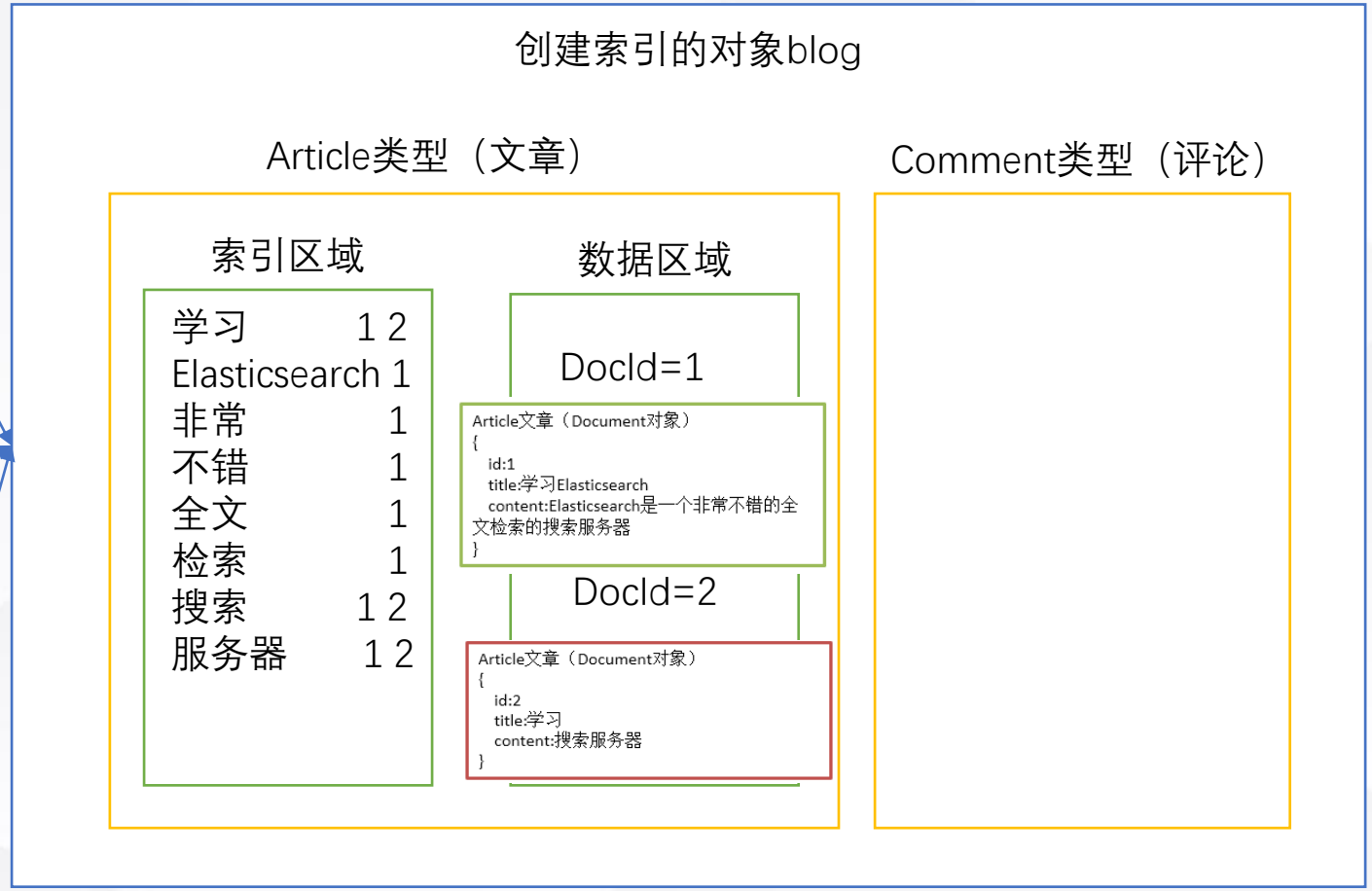
待存储的内容

Article文章（Document对象）
{
id:1
title:学习Elasticsearch
content:Elasticsearch是一个非常不错的全文检索的搜索服务器
}

Article文章（Document对象）
{
id:2
title:学习
content:搜索服务器
}

映射：字段类型、是否存储、是否分词

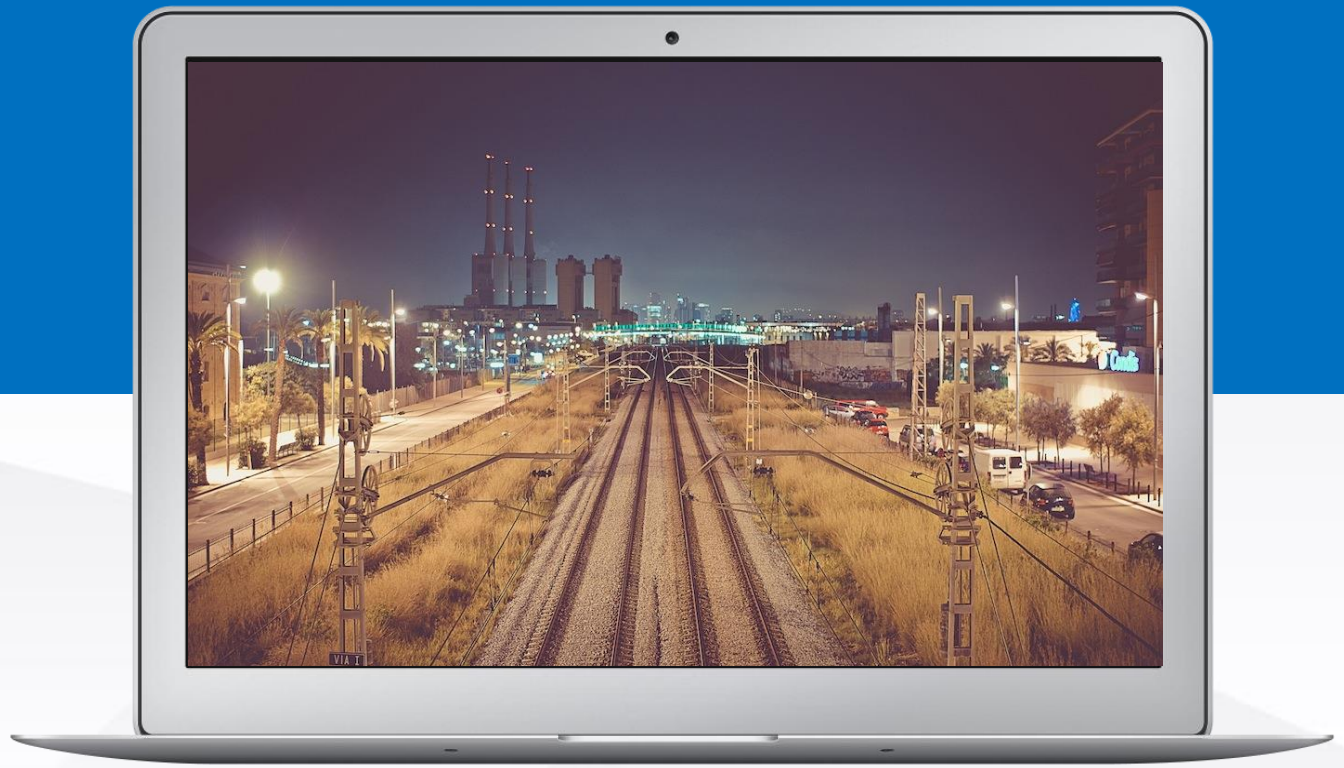
```
Mapping
    index : 'blog',
    type : 'article',
    body : {
        article: {
            properties: {
                id: {
                    type: 'string',
                    analyzer: 'ik',
                    store: 'yes',
```




搜索Elasticsearch，通过倒排索引，先搜索索引区域，查找到对应的DocId，通过DocId,去数据区域查找数据

八六三软件股份有限公司
863 Software Co., Ltd.

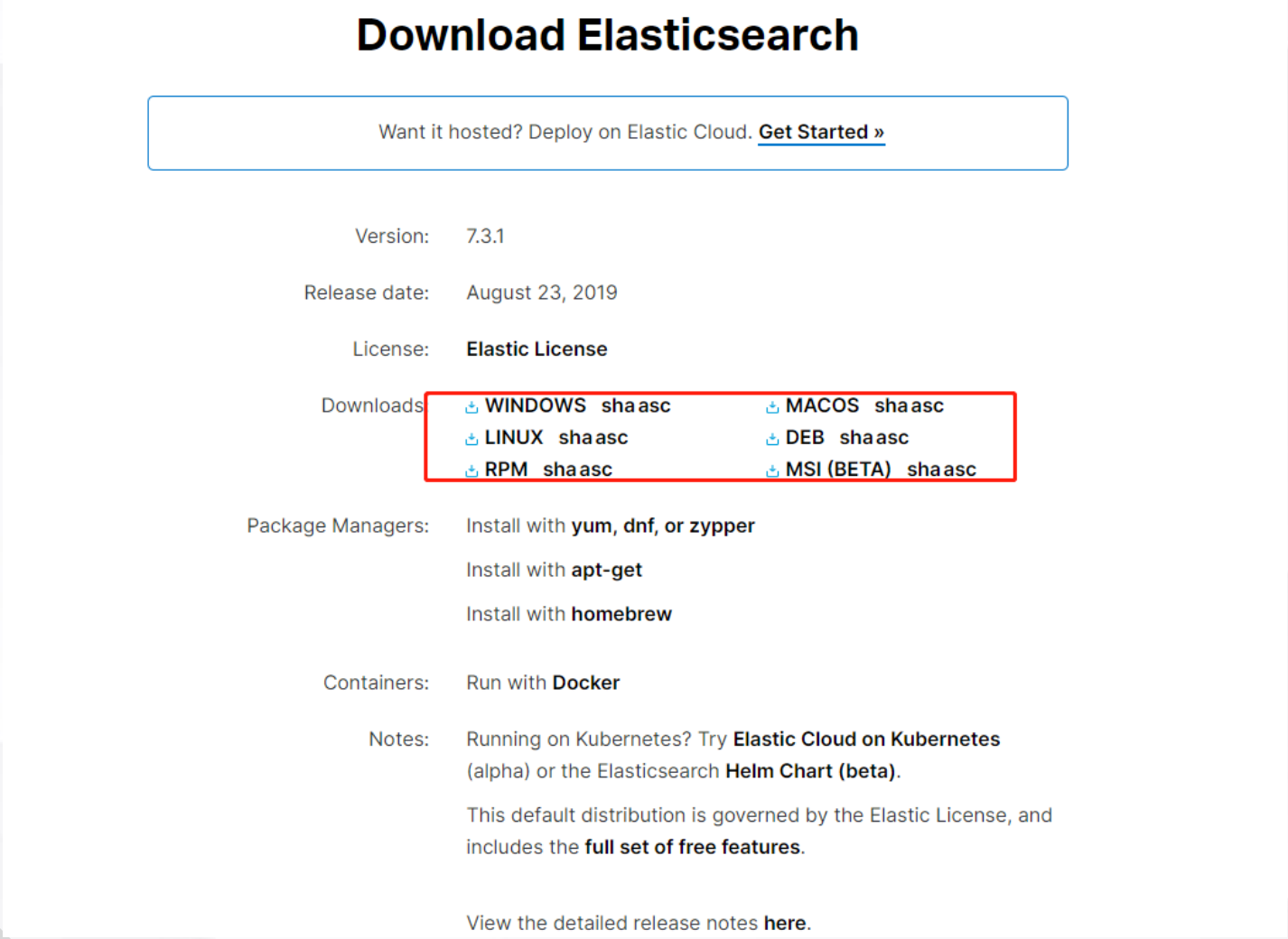
# 三、ElasticSearch部署



# ES下载

八六三软件股份有限公司
863 Software Co., Ltd.

https://www.elastic.co/cn/downloads/elasticsearch



## Download Elasticsearch

Want it hosted? Deploy on Elastic Cloud. **Get Started »**

Version: 7.3.1

Release date: August 23, 2019

License: **Elastic License**

Downloads:
- **WINDOWS** sha asc
- **MACOS** sha asc
- **LINUX** sha asc
- **DEB** sha asc
- **RPM** sha asc
- **MSI (BETA)** sha asc

Package Managers:
Install with **yum, dnf, or zypper**
Install with **apt-get**
Install with **homebrew**

Containers: Run with **Docker**

Notes: Running on Kubernetes? Try **Elastic Cloud on Kubernetes** (alpha) or the Elasticsearch **Helm Chart (beta)**.

This default distribution is governed by the Elastic License, and includes the **full set of free features**.

View the detailed release notes **here**.

Elasticsearch 分为 Windows 和 Linux 两个版本。根据项目需求下载指定版本。

# ES安装（Linux）

八六三软件股份有限公司
863 Software Co., Ltd.

**一、软件解压**

cd /usr/local/soft/

tar -zxvf elasticsearch-7.3.1-linux-x86_64.tar.gz

**二、创建数据和日志文件夹**

cd elasticsearch-7.3.1

mkdir data

mkdir logs

**三、修改配置文件elasticsearch.yml**

cluster.name: my-application

node.name: node-1

path.data: /usr/local/soft/elasticsearch-7.3.1/data

path.logs: /usr/local/soft/elasticsearch-7.3.1/logs

network.host: 0.0.0.0

http.port: 9200

cluster.initial_master_nodes: ["node-1"]

http.cors.enabled: true

http.cors.allow-origin: "*"

**四、修改jvm.options配置文件：**

-Xms200m

-Xmx200m

参考部署文档

**五、授权文件到hadoop用户**

chown hadoop /usr/local/soft/elasticsearch-7.3.1/ -R

**六、limits.conf文件更改**

编辑文件 /etc/security/limits.conf

hadoop soft nofile 65536

hadoop hard nofile 65536

hadoop soft nproc 4096

hadoop hard nproc 4096

**七、20-nproc.conf文件更改**

vi /etc/security/limits.d/20-nproc.conf

hadoop soft nproc 4096

root soft nproc unlimited

**八、sysctl.conf文件更改**

vim 编辑 /etc/sysctl.conf，

在末尾加上：

vm.max_map_count = 655360

执行命令：sysctl -p

# ES启动（Linux）

八六三软件股份有限公司
863 Software Co., Ltd.

**切换用户（root用户不能启动es）：**

su hadoop
cd /usr/local/soft/elasticsearch-7.3.1
./bin/elasticsearch

```
gSkqUKmV3tr6OoA}{yjITBgRdSpWppUSAWL0XwQ}{192.168.9.163}{192.168.9.163:9300}{dim}{ml.machine_memory=1023934464, xpack.insta
lled=true, ml.max_open_jobs=20} elect leader, _BECOME_MASTER_TASK_, _FINISH_ELECTION_], term: 1, version: 1, reason: maste
r node changed {previous [], current [{node-1}{2ERbAylgSkqUKmV3tr6OoA}{yjITBgRdSpWppUSAWL0XwQ}{192.168.9.163}{192.168.9.16
3:9300}{dim}{ml.machine_memory=1023934464, xpack.installed=true, ml.max_open_jobs=20}]}
[2019-08-23T09:24:14,129][INFO ][o.e.c.c.CoordinationState] [node-1] cluster UUID set to [xOeYpXVoQAqjns9AnDcG3g]
[2019-08-23T09:24:14,180][INFO ][o.e.c.s.ClusterApplierService] [node-1] master node changed {previous [], current [{node-
1}{2ERbAylgSkqUKmV3tr6OoA}{yjITBgRdSpWppUSAWL0XwQ}{192.168.9.163}{192.168.9.163:9300}{dim}{ml.machine_memory=1023934464, x
pack.installed=true, ml.max_open_jobs=20}]}, term: 1, version: 1, reason: Publication{term=1, version=1}
[2019-08-23T09:24:14,334][INFO ][o.e.h.AbstractHttpServerTransport] [node-1] publish_address {192.168.9.163:9200}, bound_a
ddresses {[::]:9200}
[2019-08-23T09:24:14,334][INFO ][o.e.n.Node               ] [node-1] started
[2019-08-23T09:24:14,440][INFO ][o.e.g.GatewayService     ] [node-1] recovered [0] indices into cluster_state
[2019-08-23T09:24:14,797][INFO ][o.e.c.m.MetaDataIndexTemplateService] [node-1] adding template [.watch-history-10] for in
dex patterns [.watcher-history-10*]
[2019-08-23T09:24:14,916][INFO ][o.e.c.m.MetaDataIndexTemplateService] [node-1] adding template [.triggered_watches] for i
ndex patterns [.triggered_watches*]
[2019-08-23T09:24:14,987][INFO ][o.e.c.m.MetaDataIndexTemplateService] [node-1] adding template [.watches] for index patte
rns [.watches*]
[2019-08-23T09:24:15,052][INFO ][o.e.c.m.MetaDataIndexTemplateService] [node-1] adding template [.monitoring-logstash] for
 index patterns [.monitoring-logstash-7-*]
[2019-08-23T09:24:15,154][INFO ][o.e.c.m.MetaDataIndexTemplateService] [node-1] adding template [.monitoring-es] for index
 patterns [.monitoring-es-7-*]
[2019-08-23T09:24:15,246][INFO ][o.e.c.m.MetaDataIndexTemplateService] [node-1] adding template [.monitoring-beats] for in
dex patterns [.monitoring-beats-7-*]
[2019-08-23T09:24:15,333][INFO ][o.e.c.m.MetaDataIndexTemplateService] [node-1] adding template [.monitoring-alerts-7] for
 index patterns [.monitoring-alerts-7]
[2019-08-23T09:24:15,428][INFO ][o.e.c.m.MetaDataIndexTemplateService] [node-1] adding template [.monitoring-kibana] for i
ndex patterns [.monitoring-kibana-7-*]
[2019-08-23T09:24:15,495][INFO ][o.e.x.i.a.TransportPutLifecycleAction] [node-1] adding index lifecycle policy [watch-hist
ory-ilm-policy]
[2019-08-23T09:24:15,728][INFO ][o.e.l.LicenseService     ] [node-1] license [0a495f01-1fef-488c-9a0c-68b4712510a9] mode [
basic] - valid
[2019-08-23T09:24:15,729][INFO ][o.e.x.s.s.SecurityStatusChangeListener] [node-1] Active license is now [BASIC]; Security
is disabled
[2019-08-23T09:24:44,267][WARN ][o.e.c.r.a.DiskThresholdMonitor] [node-1] high disk watermark [90%] exceeded on [2ERbAylgS
kqUKmV3tr6OoA][node-1][/usr/local/soft/elasticsearch-7.3.1/data/nodes/0] free: 716.5mb[8%], shards will be relocated away
from this node
[2019-08-23T09:24:44,268][INFO ][o.e.c.r.a.DiskThresholdMonitor] [node-1] rerouting shards: [high disk watermark exceeded
on one or more nodes]
[2019-08-23T09:25:14,271][WARN ][o.e.c.r.a.DiskThresholdMonitor] [node-1] high disk watermark [90%] exceeded on [2ERbAylgS
kqUKmV3tr6OoA][node-1][/usr/local/soft/elasticsearch-7.3.1/data/nodes/0] free: 716mb[8%], shards will be relocated away fr
om this node
[2019-08-23T09:25:44,275][WARN ][o.e.c.r.a.DiskThresholdMonitor] [node-1] high disk watermark [90%] exceeded on [2ERbAylgS
kqUKmV3tr6OoA][node-1][/usr/local/soft/elasticsearch-7.3.1/data/nodes/0] free: 716mb[8%], shards will be relocated away fr
om this node
[2019-08-23T09:25:44,275][INFO ][o.e.c.r.a.DiskThresholdMonitor] [node-1] rerouting shards: [high disk watermark exceeded
on one or more nodes]
[2019-08-23T09:26:14,279][WARN ][o.e.c.r.a.DiskThresholdMonitor] [node-1] high disk watermark [90%] exceeded on [2ERbAylgS
```

# ES验证（Linux）

八六三软件股份有限公司
863 Software Co., Ltd.

curl 127.0.0.1:9200

```
Last login: Fri Aug 23 09:16:29 2019
[root@master ~]#  curl 127.0.0.1:9200
{
  "name" : "node-1",
  "cluster_name" : "my-application",
  "cluster_uuid" : "xOeYpXVoQAqjns9AnDcG3g",
  "version" : {
    "number" : "7.3.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "4749ba6",
    "build_date" : "2019-08-19T20:19:25.651794Z",
    "build_snapshot" : false,
    "lucene_version" : "8.1.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

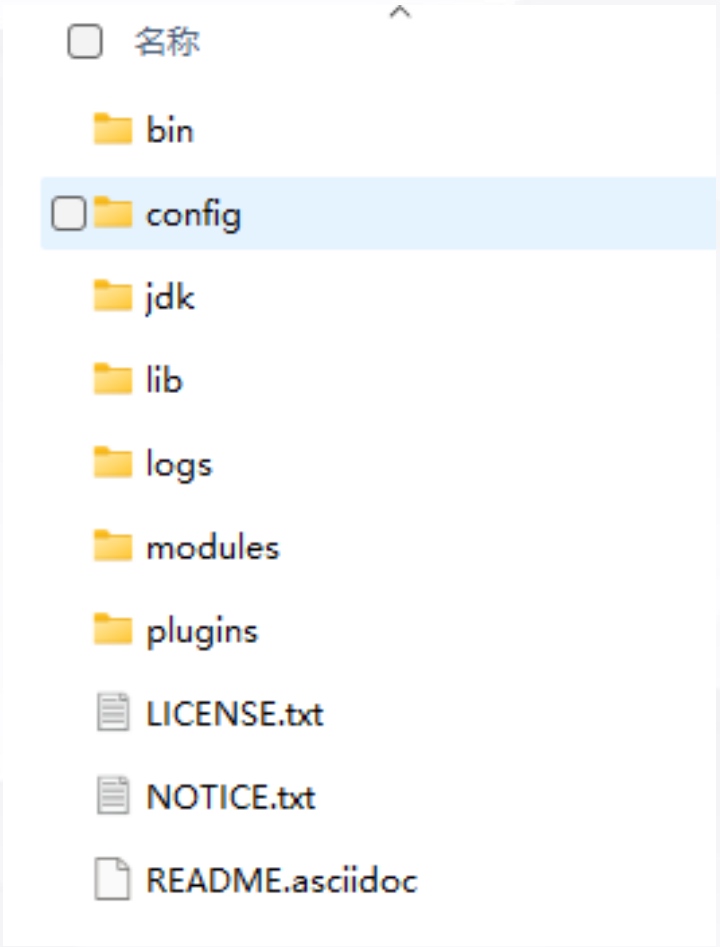
浏览器：http://192.168.9.163:9200/


```
{
  "name" : "node-1",
  "cluster_name" : "my-application",
  "cluster_uuid" : "xOeYpXVoQAqjns9AnDcG3g",
  "version" : {
    "number" : "7.3.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "4749ba6",
    "build_date" : "2019-08-19T20:19:25.651794Z",
    "build_snapshot" : false,
    "lucene_version" : "8.1.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

# ES安装启动（Windows）

八六三软件股份有限公司 863 Software Co., Ltd.

官网下载的安装包为压缩包，下载之后解压即可，目录如下




**bin：可执行的文件，ES 的启动文件就在其中；**

config：ES 的配置文件目录；

jdk：ES 自带的 JDK 环境，因为 ES 对 JDK 有要求；

lib：ES 自身依赖的库；

logs：ES 日志目录；

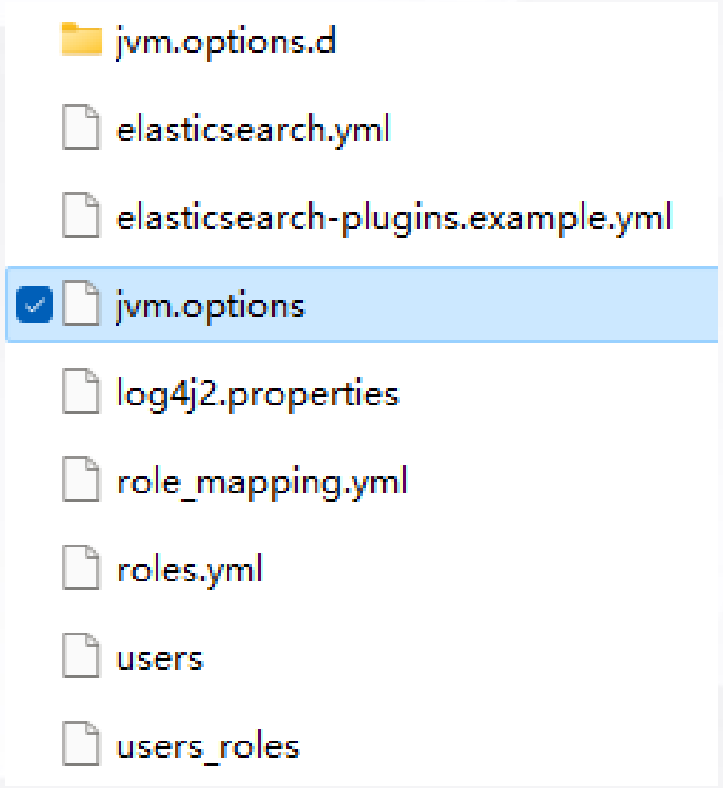
modules：ES 模块目录；

plugins：ES 的插件目录；

# ES安装启动（Windows）

八六三软件股份有限公司
863 Software Co., Ltd.

打开 config 目录




使用文本编辑器，打开 jvm.options 文件，修改 JDK 的虚拟内存大小。

**默认情况下 ES 启动会消耗大量的内存，可以配置自定一大小。**

**如下：**

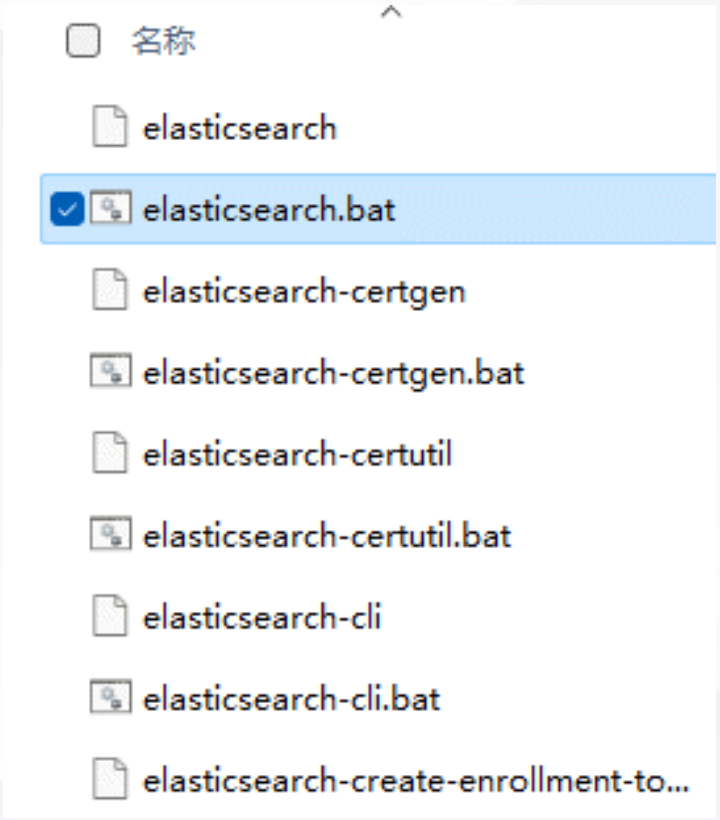
**-Xms1g**

**-Xmx1g**

# ES安装启动（Windows）

八六三软件股份有限公司
863 Software Co., Ltd.

打开 bin 目录




双击 elasticsearch.bat 即可运行 ES。

PS：如果本地JDK不能使用时，可以使用ES 自带JDK，需要在环境变量中配置。

/config/elasticsearch.yml 添加如下配置，可以绕过ES登录认证：

xpack.security.enabled: false

# ES安装启动（Windows）

八六三软件股份有限公司
863 Software Co., Ltd.

```
[shards started [[.security-7][0]]]." previous.health="YELLOW" reason="shards started [[.security-7][0]]"



----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
-> Elasticsearch security features have been automatically configured!
-> Authentication is enabled and cluster connections are encrypted.

->  Password for the elastic user (reset with `bin/elasticsearch-reset-password -u elastic`):
  AeWN1Th556j5y2*W_E7N

->  HTTP CA certificate SHA-256 fingerprint:
  0375b8f1c12c446b996824f8fa690515b3b8a3ad80309d3508b84e0a3419184e

->  Configure Kibana to use this cluster:
* Run Kibana and click the configuration link in the terminal when Kibana starts.
* Copy the following enrollment token and paste it into Kibana in your browser (valid for the next 30 minutes):
  eyJ2ZXIiOiI4LjMuMyIsImFkciI6WyIxMC44Ny4yMjEuMjQ2OjkyMDAiXSwiZmdyIjoiMDM3NWI4ZjFjMTJjNDQ2Yjk5NjgyNGY4ZmE2OTA1MTViM2I4YTNhZDgwMzA5ZDM1MDhiODR1MGEzNDE5MTg0ZSIsImt1eSI6Ij1mT3JwSU1CeXpaazFOUnFLcDBmOk5pb3hxVVk5UWRLXzQyemxGeVg3SkEifQ==

->  Configure other nodes to join this cluster:
* On this node:
  - Create an enrollment token with `bin/elasticsearch-create-enrollment-token -s node`.
  - Uncomment the transport.host setting at the end of config/elasticsearch.yml.
  - Restart Elasticsearch.
* On other nodes:
  - Start Elasticsearch with `bin/elasticsearch --enrollment-token <token>`, using the enrollment token that you generated.
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
```

登录 ES 的密码
账号是：elastic

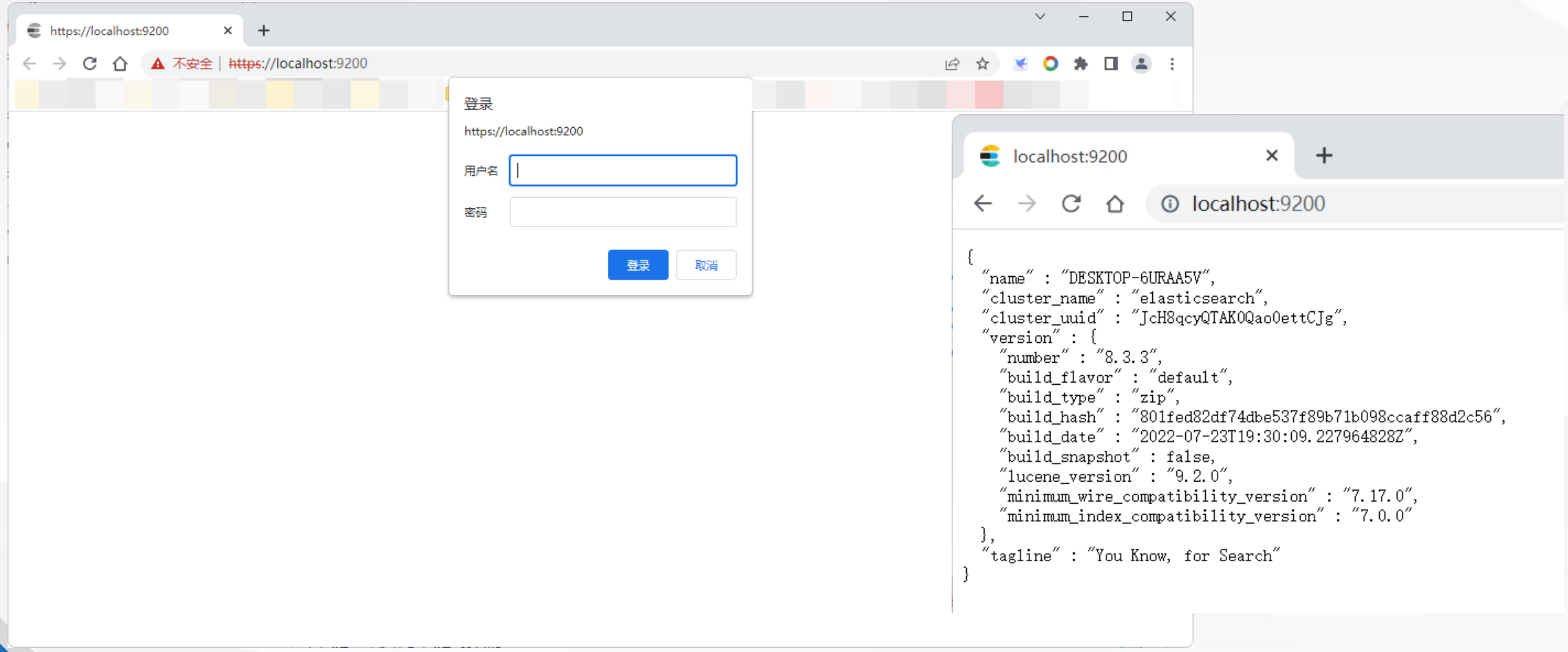
Kibana 的认证
Token

# ES安装启动（Windows）

八六三软件股份有限公司
863 Software Co., Ltd.

打开浏览器，请求 https://localhost:9200 地址，出现如下信息表示成功。

**PS：这里是 https 安全协议。如果已经绕过安全登录使用 http 。**



# Kibana下载

八六三软件股份有限公司
863 Software Co., Ltd.

## Download Kibana

Want it hosted? Deploy on Elastic Cloud. **Get Started »**

| | |
|---|---|
| Version: | 7.3.1 |
| Release date: | August 23, 2019 |
| License: | **Elastic License** |
| Downloads: | **WINDOWS** sha asc<br>**LINUX 64-BIT** sha asc<br>**DEB 64-BIT** sha asc<br>**MAC** sha asc<br>**RPM 64-BIT** sha asc |
| Package Managers: | Install with **yum, dnf, or zypper**<br>Install with **apt-get**<br>Install with **homebrew** |

**https://www.elastic.co/cn/downloads/kibana**

# Kibana安装（Linux）

八六三软件股份有限公司
863 Software Co., Ltd.

**一、解压**

cd /usr/local/soft/

tar -zxvf  kibana-7.3.1-linux-x86_64.tar.gz

**二、编辑kibana.yml文件**

server.port: 5601

server.host: "192.168.9.163"

elasticsearch.hosts: ["http://192.168.9.163:9200"]

**三、修改权限**

chown hadoop /usr/local/soft/kibana-7.3.1-linux-x86_64 –R

**四、切换到hadoop用户：**

su hadoop

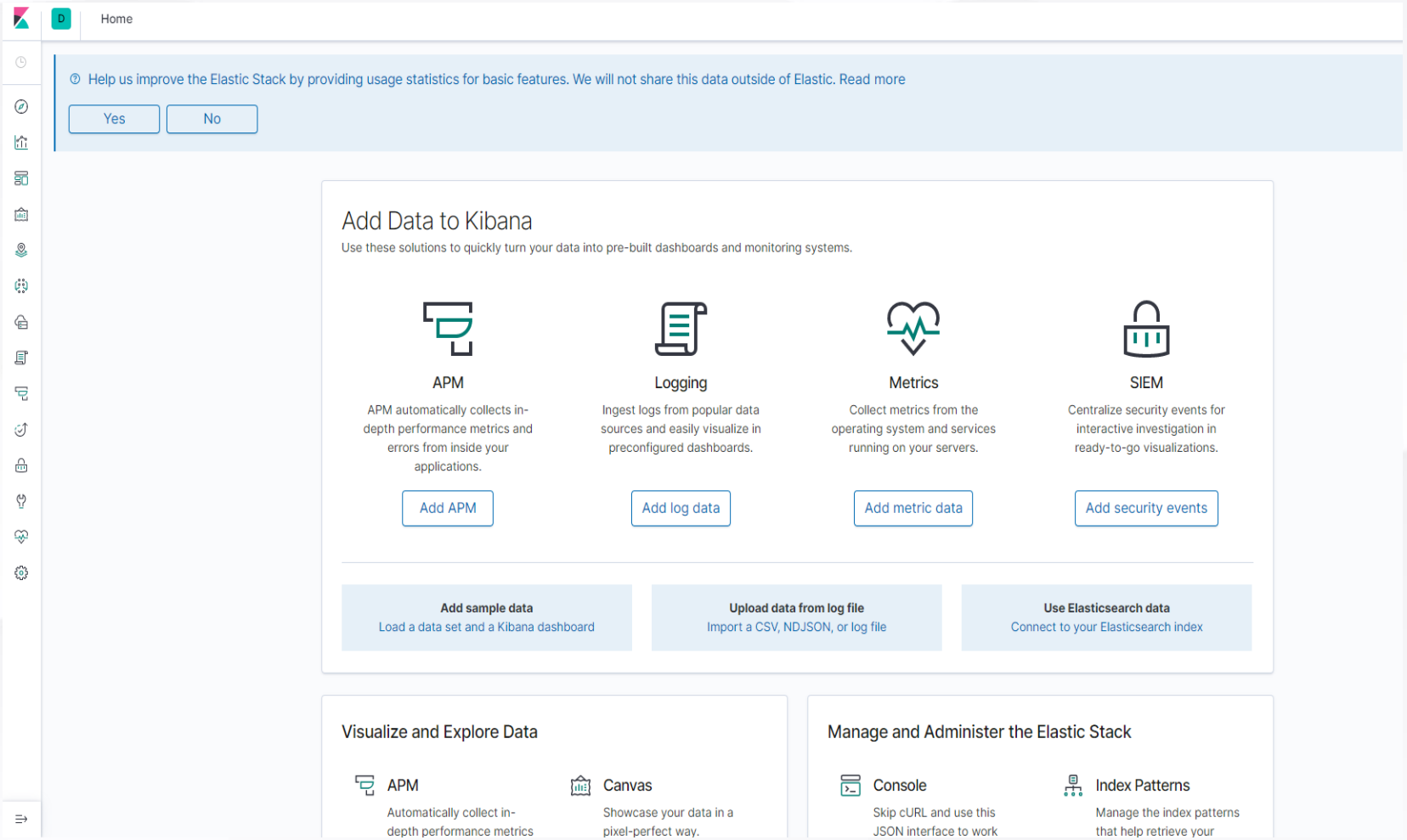
cd /usr/local/soft/kibana-7.3.1-linux-x86_64/bin/

./kibana &

# Kibana验证（Linux）

八六三软件股份有限公司
863 Software Co., Ltd.

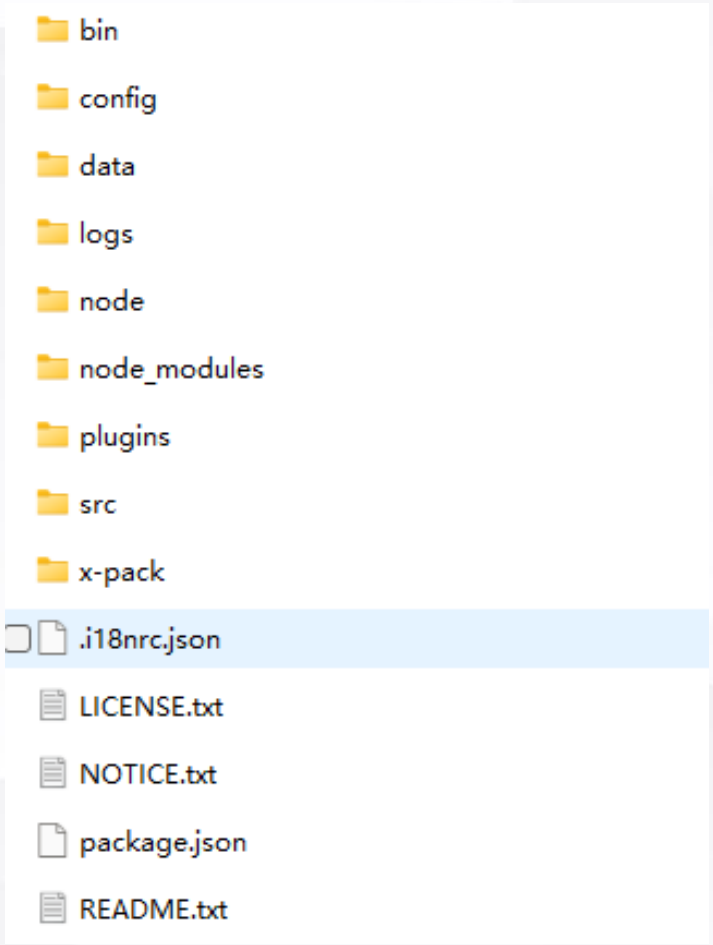
**浏览器中输入：**http://localhost:5601



# Kibana安装启动（Windows）

八六三软件股份有限公司 863 Software Co., Ltd.

官网下载的安装包为压缩包，下载之后解压即可，目录如下




目录内容和 ES 类似。

进入 bin 目录，双击 kibana.bat 启动。

PS：可以配置中文界面。

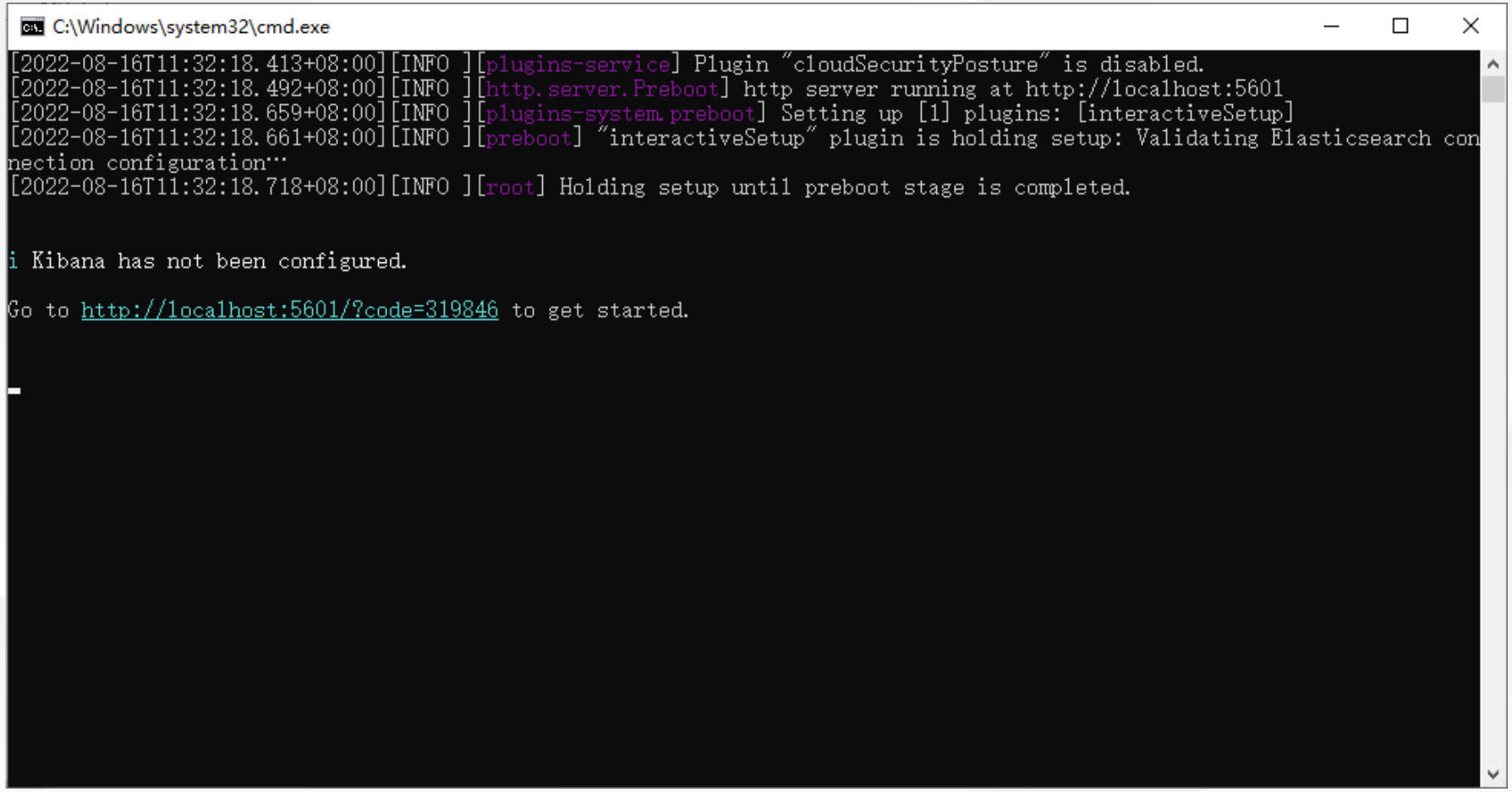

打开 kibana.yml，修改国际化配置 i18n.locale: "zh-CN"

# Kibana安装启动（Windows）

八六三软件股份有限公司
863 Software Co., Ltd.

打开浏览器：http://localhost:5601

PS：这里是 http 普通协议。

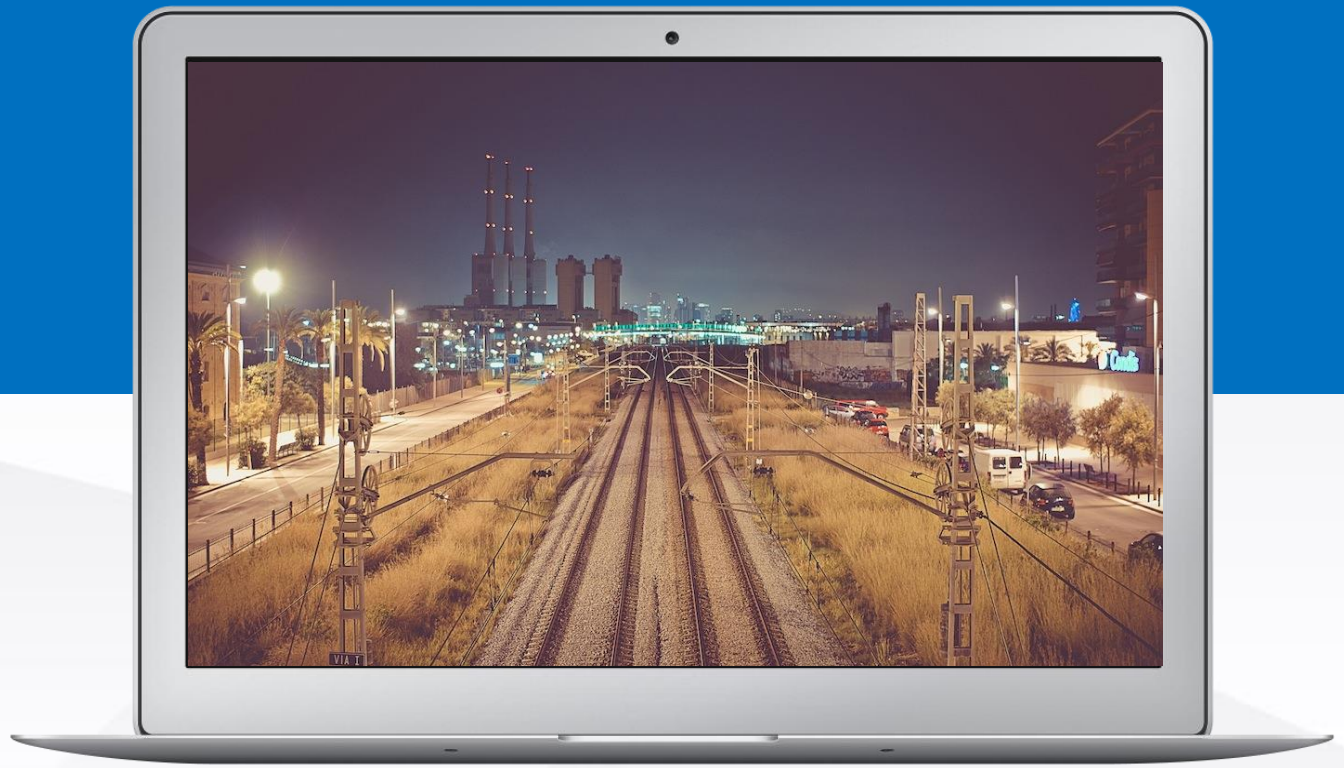


# Kibana安装启动（Windows）

八六三软件股份有限公司
863 Software Co., Ltd.

八六三软件股份有限公司
863 Software Co., Ltd.

# 四、CURL命令使用



# 查询集群配置

八六三软件股份有限公司 863 Software Co., Ltd.

Shell: curl  http://192.168.9.163:9200/_cat
浏览器：http://192.168.9.163:9200/_cat

```
[root@master ~]# curl  192.168.9.163:9200/_cat
=^.^=
/_cat/allocation
/_cat/shards
/_cat/shards/{index}
/_cat/master
/_cat/nodes
/_cat/tasks
/_cat/indices
/_cat/indices/{index}
/_cat/segments
/_cat/segments/{index}
/_cat/count
/_cat/count/{index}
/_cat/recovery
/_cat/recovery/{index}
/_cat/health
/_cat/pending_tasks
/_cat/aliases
/_cat/aliases/{alias}
/_cat/thread_pool
/_cat/thread_pool/{thread_pools}
/_cat/plugins
/_cat/fielddata
/_cat/fielddata/{fields}
/_cat/nodeattrs
/_cat/repositories
/_cat/snapshots/{repository}
/_cat/templates
```

# 查询集群状态

八六三软件股份有限公司
863 Software Co., Ltd.

查询集群状态

curl http://192.168.9.163:9200

curl http://192.168.202.133:9200/_cluster/health?pretty

curl -XGET http://192.168.9.163:9200/_cluster/stats?pretty=true


```
{
  "cluster_name" : "my-application",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 8,
  "active_shards" : 8,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 2,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 80.0
}
```



```
{
  "_nodes" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "cluster_name" : "my-application",
  "cluster_uuid" : "_yjBL1FtRSqhK9hNL9D6ow",
  "timestamp" : 1567073692672,
  "status" : "yellow",
  "indices" : {
    "count" : 8,
    "shards" : {
      "total" : 8,
      "primaries" : 8,
      "replication" : 0.0,
      "index" : {
        "shards" : {
          "min" : 1,
          "max" : 1,
          "avg" : 1.0
        },
        "primaries" : {
          "min" : 1,
          "max" : 1,
          "avg" : 1.0
        },
        "replication" : {
          "min" : 0.0,
          "max" : 0.0,
          "avg" : 0.0
        }
      }
    },
```

# 查询索引信息



查询所有索引库信息
curl http://192.168.9.163:9200/_search?pretty
查看索引信息
curl -XGET http://192.168.9.163:9200/_cat/indices?v


```
{
  "cluster_name" : "my-application",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 8,
  "active_shards" : 8,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 2,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 80.0
}
```

```
{
  "_nodes" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "cluster_name" : "my-application",
  "cluster_uuid" : "_yjBL1FtRSqhK9hNL9D6ow",
  "timestamp" : 1567073692672,
  "status" : "yellow",
  "indices" : {
    "count" : 8,
    "shards" : {
      "total" : 8,
      "primaries" : 8,
      "replication" : 0.0,
      "index" : {
        "shards" : {
          "min" : 1,
          "max" : 1,
          "avg" : 1.0
        },
        "primaries" : {
          "min" : 1,
          "max" : 1,
          "avg" : 1.0
        },
        "replication" : {
          "min" : 0.0,
          "max" : 0.0,
          "avg" : 0.0
        }
      }
    },
```

# 创建索引



创建索引:
curl -XPUT http://192.168.9.163:9200/bigdata

```
[root@master ~]# curl -XPUT http://192.168.9.163:9200/bigdata
{"acknowledged":true,"shards_acknowledged":true,"index":"bigdata"}
```

删除索引:
curl -XDELETE http://192.168.9.163:9200/customer
curl -XDELETE http://master:9200/bigdata/product/4/

```
[root@master ~]# curl -XDELETE http://master:9200/bigdata/product/4/
{"_index":"bigdata","_type":"product","_id":"4","_version":2,"result":"deleted","_shards":{"total":2,"successful":1,"failed
":0},"_seq_no":5,"_primaGET http://192.168.9.163:9200/bigdata/product/_search?size=2&from=0&prettyct/4/
```

# 添加文档



```
curl -XPOST http://192.168.9.163:9200/bigdata/product/ -H "Content-Type: application/json" -d
'{"author" : "Doug Cutting"}'
curl -XPOST http://192.168.9.163:9200/bigdata/product -H "Content-Type: application/json" -d '{"id" :
"2","author" : "Doug Cutting"}‘
curl -XPUT http://192.168.9.163:9200/bigdata/product/4? -H "Content-Type: application/json" -d
'{"author" : "Doug Cutting"}'
```

提示:
-X 指定http的请求方法 有HEAD GET POST PUT DELETE
-d 指定要传输的数据
-H 指定http请求头信息

PUT一般用户更新，POST一般用于新增比较。

ES创建索引库和索引时的注意点

1)索引库名称必须要全部小写，不能以下划线开头，也不能包含逗号

2)如果没有明确指定索引数据的ID，那么es会自动生成一个随机的ID，需要使用POST参数

# 查询具体索引



curl http://192.168.9.163:9200/bigdata/product/_search
curl http://192.168.9.163:9200/bigdata/product/_search?pretty
提示：在url后面加上一个pretty则会对返回结果进行格式化

```
[root@master ~]# curl http://192.168.9.163:9200/bigdata/product/_search
{"took":0,"timed_out":false,"_shards":{"total":1,"successful":1,"skipped":0,"failed":0},"hits":{"total":{"value":2,"relatio
n":"eq"},"max_score":1.0,"hits":[{"_index":"bigdata","_type":"product","_id":"6vTz3GwBINrLhTu4qZil","_score":1.0,"_source":
{"author" : "Doug Cutting"}},{"_index":"bigdata","_type":"product","_id":"FvT83GwBINrLhTu4B5z3","_score":1.0,"_source":{"au
thor" : "Doug Cutting"}}]}}[root@master ~]# curl http://192.168.9.163:9200/bigdata/product/_search?pretty
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
```

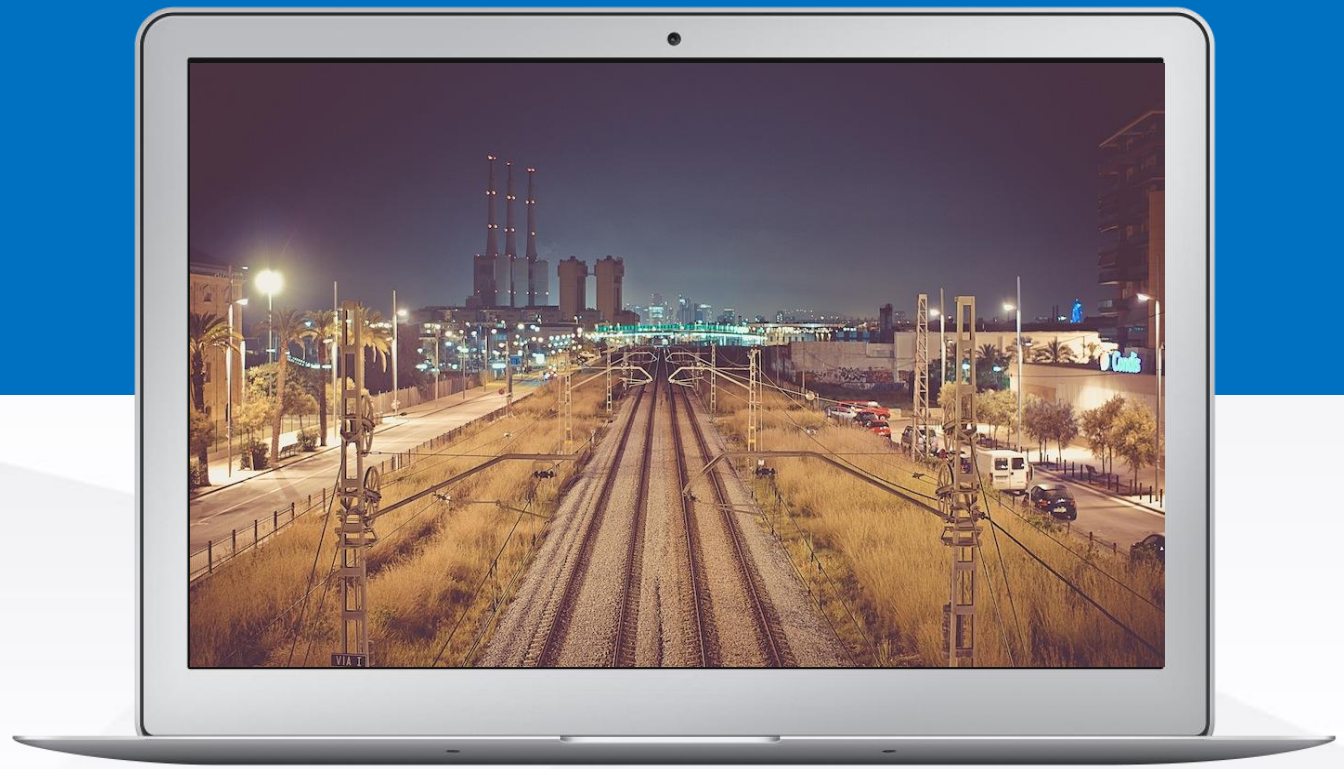
# 数据查询

八六三软件股份有限公司
863 Software Co., Ltd.

1、查询所有数据:
curl -XGET http://192.168.9.163:9200/_search?pretty
2、查询某个索引库所有数据:
curl http://192.168.9.163:9200/bigdata/_search?pretty
3、查询某个索引库某个类型的所有数据:
curl http://192.168.9.163:9200/bigdata/product/_search?pretty
4、查询具体的一条记录（通过_id）
curl -XGET http://192.168.9.163:9200/bigdata/product/2?pretty
5、查询多个字段
curl -XGET http://192.168.9.163:9200/bigdata/product/_search?pretty&_source （获取所有字段）
curl -XGET http://192.168.9.163:9200/bigdata/product/_search?_source=id
curl -XGET http://192.168.9.163:9200/bigdata/product/_search?_source=id,author （多个字段,号隔开）
6、根据条件查询
curl -XGET http://192.168.9.163:9200/bigdata/product/_search?q=id:"2“
7、多索引、多类型查询
curl -XGET http://192.168.9.163:9200/bigdata,bigdata1/product,product1/_search?pretty
curl -XGET http://192.168.9.163:9200/_all/product,product1/_search?pretty
8、分页查询
curl -XGET http://192.168.9.163:9200/bigdata/product/_search?size=2&from=0

八六三软件股份有限公司
863 Software Co., Ltd.

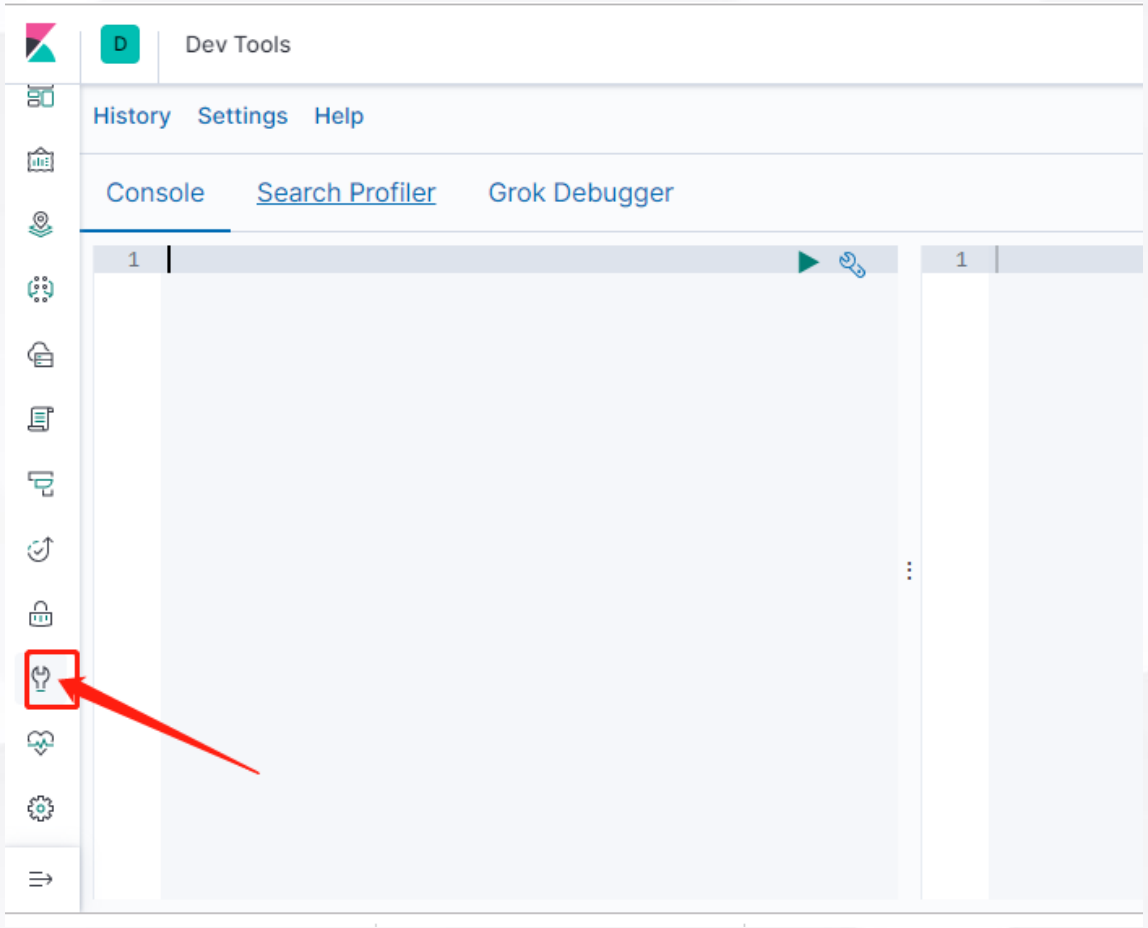
# 五、Kibana开发工具使用



# Kibana开发工具

八六三软件股份有限公司
863 Software Co., Ltd.

http://localhost:5601/app/kibana#/dev_tools/console?_g=()



# 索引操作



```
# 添加索引，索引单词小写
# 语法结构：PUT 索引名称
PUT shop_index

# ES 不支持修改索引

# 判断是否有索引，返回值是HTTP的状态码（200：有；404：没有；）
HEAD shop_index

# 查询单个索引
# 语法结构：GET 索引名称
GET shop_index

# 查询所有的索引
GET _cat/indices

# 删除索引
# 语法结构：DELETE 索引名称
DELETE shop_index
```

# 文档操作



```
# 添加文档
# 语法结构：PUT 索引名称/_doc/唯一ID
# 唯一ID 为手动创建
PUT shop_index/_doc/1001
{
    "id" : 1001,
    "name" : "TV",
    "price" : 6999
}

# 语法结构：POST 索引名称/_doc/
# 唯一ID 为自动创建
POST shop_index/_doc
{
    "id" : 1002,
    "name" : "TV-MI",
    "price" : 3699
}
```

```
# 查询文档
# 语法结构：GET 索引名称_doc/唯一ID
GET shop_index/_doc/1001

# 查询当前索引中所有的文档数据
GET shop_index/_search

# 删除文档
# 语法结构：DELETE 索引名称/_doc/唯一ID
DELETE shop_index/_doc/1003
```

# 文档操作



```
# 修改文档，PUT 和 POST 都支持文档存在时修改，不存在时创建。
# 语法结构：PUT 索引名称/_doc/唯一ID
PUT shop_index/_doc/1001
{
    "id" : 1001,
    "name" : "Phone",
    "price" : 6999,
    "type" : "apple"
}
```

```
# 语法结构：POST 索引名称/_doc/唯一ID
POST shop_index/_doc/1001
{
    "id" : 1001,
    "name" : "Phone",
    "price" : 6999,
    "type" : "mi"
}
```

```
# 局部修改文档
# 语法结构：POST 索引名称/_update/唯一ID
POST shop_index/_update/1005
{
    "doc":{
        "price": 6599.00
    }
}
```

# 文档检索



```
# 提前准备数据，使用批量添加
PUT shop_index/_bulk
{"index":{"_index":"shop_index", "_id":"1002"}}
{"id":1002, "name":"iPhone", "price":5000}
{"index":{"_index":"shop_index", "_id":"1003"}}
{"id":1003, "name":"iPhone13", "price":5500}
{"index":{"_index":"shop_index", "_id":"1004"}}
{"id":1004, "name":"iPhone 13", "price":5700}
{"index":{"_index":"shop_index", "_id":"1005"}}
{"id":1005, "name":"iPhone 13 Pro", "price":6000}
{"index":{"_index":"shop_index", "_id":"1006"}}
{"id":1006, "name":"iPhone 13 Max", "price":6000}
{"index":{"_index":"shop_index", "_id":"1007"}}
{"id":1007, "name":"iPhone 13 Pro Max", "price":9000}
{"index":{"_index":"shop_index", "_id":"1008"}}
{"id":1008, "name":"小米", "price":3959}
{"index":{"_index":"shop_index", "_id":"1009"}}
{"id":1009, "name":"红米", "price":2999}
{"index":{"_index":"shop_index", "_id":"1010"}}
{"id":1010, "name":"华为", "price":5999}
```

# 文档检索



```
# 分词查询
GET shop_index/_search
{
    "query": {
        "match": {
            "name": "米"
        }
    }
}
```

```
# 分词查询（等值）
GET shop_index/_search
{
    "query": {
        "match_phrase": {
            "name": "米"
        }
    }
}
```

```
# 等值查询（精确）
GET shop_index/_search
{
    "query": {
        "term": {
            "name": {
                "value": "米"
            }
        }
    }
}
```

**match、 match_phrase、 term区别：准备数据 [ 张三，张丰，张三丰，李三 ]；**

match：分词查询，将输入的参数拆开之后再匹配分词；输入张三后，都可以检索到。

match_phrase：等值查询，按照输入的参数匹配分词；输入张三后，展示张三和，张三丰；

term：精确查询，按照输入的参数精确匹配分词；输入张三后，无法检索到数据；

# 文档检索



```
# 高亮
GET shop_index/_search
{
    "query": {
        "match_phrase": {
            "name": "米"
        }
    },
    "highlight": {
        "fields": {
            "name": {}
        }
    }
}
```

```
# 对查询结构进行限制，展示文档中的某些字段
GET shop_index/_search
{
    "_source": ["price"],
    "query": {
        "match": {
            "name": "iphone 13"
        }
    }
}
```

# 文档检索



```
# 组合条件（或者）
GET shop_index/_search
{
    "query": {
        "bool": {
            "should": [
                {
                    "match": {
                        "name": "max"
                    }
                },
                {
                    "match": {
                        "price": 5000
                    }
                }
            ]
        }
    }
}
```

```
# 组合条件（并且）
GET shop_index/_search
{
    "query": {
        "bool": {
            "must": [
                {
                    "match": {
                        "name": "max"
                    }
                },
                {
                    "match": {
                        "price": 9000
                    }
                }
            ]
        }
    }
}
```

# 文档检索



```
# 范围查询
GET shop_index/_search
{
    "query": {
        "bool": {
            "must": [
                {
                    "match": {
                        "name": "max"
                    }
                }
            ],
            "filter": [
                {
                    "range": {
                        "price": {
                            "gte": 7000,
                            "lte": 10000
                        }
                    }
                }
            ]
        }
    }
}
```

```
# 查询结果排序
GET shop_index/_search
{
    "query": {
        "match": {
            "name": "max"
        }
    },
    "sort": [
        {
            "price": {
                "order": "desc"
            }
        }
    ]
}
```

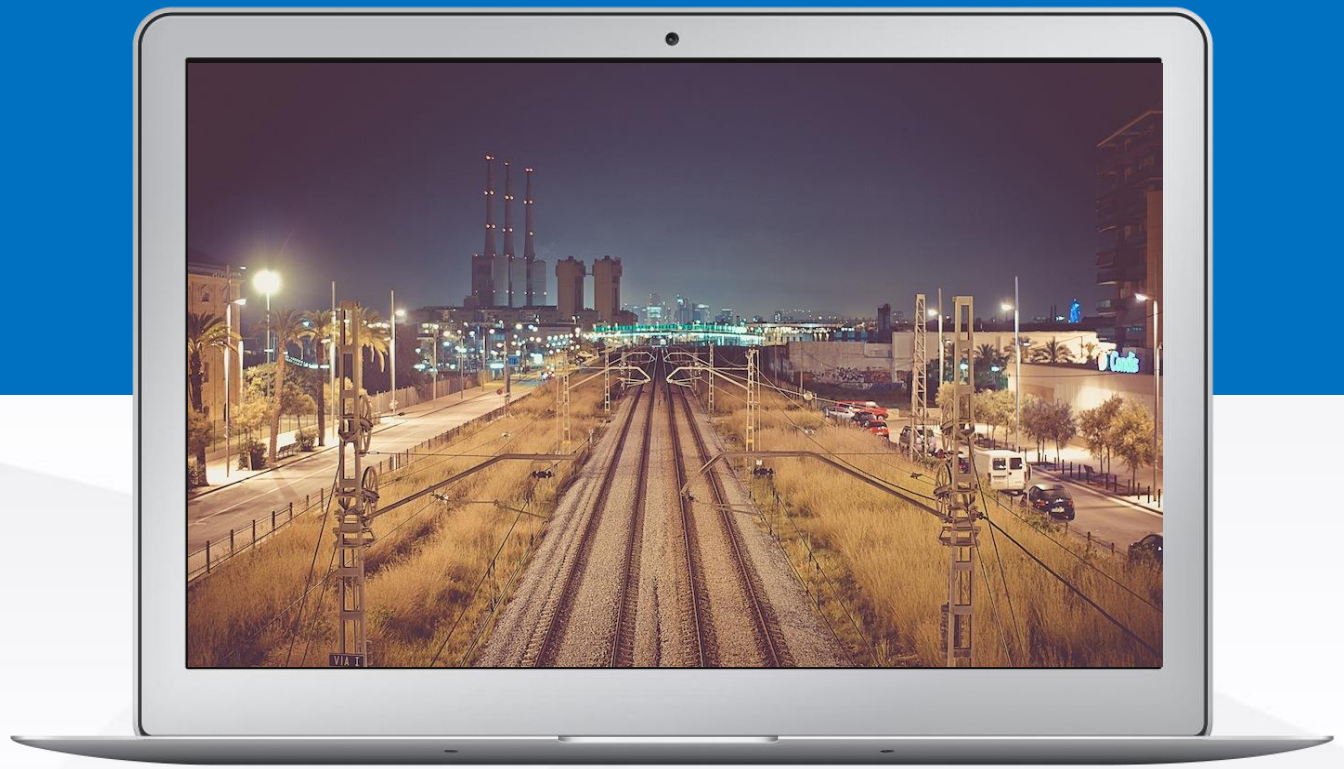
# 文档检索



```
# 分页查询
# from = (pageNum - 1) * size
GET shop_index/_search
{
    "query": {
        "match_all": {}
    },
    "sort": [
        {
            "price": {
                "order": "desc"
            }
        }
    ],
    "from": 0,
    "size": 2
}
```

八六三软件股份有限公司
863 Software Co., Ltd.

# 五、ElasticSearch之编程



# pom.xml 依赖添加



ES 7 版本

```
<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>7.3.1</version>
</dependency>
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>transport</artifactId>
    <version>7.3.1</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.9.0</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
```

ES8 版本

```
<dependency>
    <groupId>co.elastic.clients</groupId>
    <artifactId>elasticsearch-java</artifactId>
    <version>8.3.3</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.3</version>
</dependency>
```

# ES客户端连接



```
@Test
public void init() throws IOException {
    /*
        1、创建 Rest 客户端，应为 ES 使用的是 Rest 风格的请求
        Rest 请求是一个 HTTP 请求，所以要配置请求的地址（ES 服务器地址）
        HttpHost 连接服务端的主机配置
        构造器参数1：服务端的IP地址，此案例的 ES 是安装在本地的，所以可以写 localhost
        构造器参数2：ES 端口号
     */
    HttpHost httpHost = new HttpHost("localhost", 9200);
    RestClient restClient = RestClient.builder(httpHost).build();
    /*
        2、使用 RestClientTransport 进行请求的转换（表现层内容转换）
        JSON 数据，通过 restClient 客户端发送，进行请求转换
     */
    ElasticsearchTransport transport = new RestClientTransport(restClient, new JacksonJsonpMapper());
    // 3、创建 ES 客户端
    ElasticsearchClient client = new ElasticsearchClient(transport);
    // 4、通过 ES 客户端操作
    // 5、关闭ES客户端
    transport.close();
    restClient.close();
}
```

# 索引操作



- **1、创建索引，返回状态**

```
CreateIndexResponse createIndexResponse = client.indices().create(c -> c.index("索引名字"));
boolean acknowledged = createIndexResponse.acknowledged();
```

- **2、查询索引**

```
GetIndexResponse getIndexResponse = client.indices().get(builder -> builder.index("索引名字"));
getIndexResponse.result();
```

- **3、删除索引**

```
DeleteIndexResponse deleteIndexResponse = client.indices().delete(builder -> builder.index(("索引名字"));
deleteIndexResponse.acknowledged();
```

# 文档操作



- **添加数据**

```
// index中参数为文档所属的索引名，id中参数为当文档的id，document为文档数据，新版本支持直接传入java对象。
CreateResponse createResponse = client.create(builder -> builder.index("索引名字").id(ID).document(数据对象));
createResponse.result();
```

- **批量添加数据**

```
// 构建一个批量数据集合
List<BulkOperation> list = new ArrayList<>();
list.add(new BulkOperation.Builder().create(d -> d.document(数据对象1).id(ID).index("索引名字")).build());
list.add(new BulkOperation.Builder().create(d -> d.document(数据对象2).id(ID).index("索引名字")).build());

// 调用bulk方法执行批量插入操作
BulkResponse bulkResponse = client.bulk(builder -> builder.index("user_test").operations(list));
bulkResponse.items();
```

# 文档操作



- **查询数据**

```
// 如果查不到控制台抛出NullPointerException
GetResponse<T> getResponse = client.get(builder -> builder.index("索引名字").id(ID), T.class);
getResponse.source();
```

- **修改数据**

```
// 使用map集合封装需要修改的内容
Map<String, Object> map = new HashMap<>();
map.put("field", newValue);
UpdateResponse<T> updateResponse = client.update(builder -> builder.index("索引名字").id(ID).doc(map), T.class);
updateResponse.result();
```

# 文档操作



- **删除数据**

```
DeleteResponse deleteResponse = client.delete(builder -> builder.index("索引名字").id(ID));
deleteResponse.result();
```

- **批量删除数据**

```
// 构建一个批量数据集合
List<BulkOperation> list = new ArrayList<>();
list.add(new BulkOperation.Builder().delete(builder -> builder.id(ID1).index("索引名字")).build());
list.add(new BulkOperation.Builder().delete(builder -> builder.id(ID2).index("索引名字")).build());
// 调用bulk方法执行批量删除操作
BulkResponse bulkResponse = client.bulk(builder -> builder.index("索引名字").operations(list));
bulkResponse.items();
```

# 文档操作



- **全量查询**

```
SearchResponse<T> searchResponse = client.search(b -> b.index("索引名字").query(q -> q.matchAll(m -> m)), T.class);
HitsMetadata<T> hits = searchResponse.hits();
for (Hit<T> hit : hits.hits()) {
    System.out.println("T = " + hit.source().toString());
}
System.out.println(searchResponse.hits().total().value());
```

- **分页查询**

```
SearchResponse<T> searchResponse = client.search(
    b -> b.index("索引名字").query(q -> q.matchAll(m -> m)).from(2).size(2),
    T.class
);
searchResponse.hits().hits().forEach(h -> System.out.println(h.source().toString()));
```

# 文档操作



- **排序查询**

```
SearchResponse<T> searchResponse = client.search(
    b -> b.index("索引名字").query(q -> q.matchAll(m -> m)).sort(o -> o.field(f -> f.field("属性").order(SortOrder.Asc))),
    T.class);
searchResponse.hits().hits().forEach(h -> System.out.println(h.source().toString()));
```

- **条件组合查询（并且）**

```
SearchResponse<T> searchResponse = client.search(
    b -> b.index("索引名字").query(q -> q.bool(b -> b
            .must(m -> m.match(u -> u.field("属性1").query(值)))
            .must(m -> m.match(u -> u.field("属性2").query(值)))
            .mustNot(m -> m.match(u -> u.field("属性3").query(值)))
        )
    ),
    T.class
);
searchResponse.hits().hits().forEach(h -> System.out.println(h.source().toString()));
```

# 文档操作



- **条件组合查询（或者）**

```
SearchResponse<T> searchResponse = client.search(
    b -> b.index("索引名字").query(q -> q.bool(b -> b
            .should(h -> h.match(u -> u.field("属性1").query(值1)))
            .should(h -> h.match(u -> u.field("属性2").query(值)))
        )
    )
    , T.class
);
searchResponse.hits().hits().forEach(h -> System.out.println(h.source().toString()));
```