链滴

1. Getting Started (入门指南)

作者：felayman

原文链接：https://ld246.com/article/1512755081583

来源网站：链滴

Getting Started（入门指南）

Elasticsearch 是一个高度可扩展且开源的全文检索和分析引擎。它可以让您快速且近实时地存储，检以及分析海量数据。它通常用作那些具有复杂搜索功能和需求的应用的底层引擎或者技术。

下面是 Elasticsearch 一些简单的使用案例：

● 您运行一个可以让您顾客来搜索您所售产品的在线的网络商店。在这种情况下，您可以使用 Elasticsearch 来存储您的整个产品的目录和库存，并且为他们提供搜索和自动完成的建议。

● 您想要去收集日志或交易数据，并且您还想要去分析和挖掘这些数据以来找出趋势，统计，概述，者异常现。在这种情况下，您可以使用 Logstash（Elasticsearch/Logstash/Kibana 技术栈中的一部）来收集，聚合，以及解析数据，然后让 Logstash 发送这些数据到 Elasticsearch。如果这些数据存于 Elasticsearch 中，那么您就可以执行搜索和聚合以挖掘出任何您感兴趣的信息。

● 您运行一个价格警告平台，它允许客户指定精确的价格，如“我感兴趣的是购买指定的电子产品，果任何供应商该产品的价格在未来一个月内低于 $X 这个价钱的话我应该被通知到”。在这种情况下您可以收集供应商的价格，推送它们到 Elasticsearch 中去，然后使用 reverse-search（Percolator（反向搜索（过滤器））功能以匹配客户查询价格的变动，最后如果发现匹配成功就给客户发出通知。

● 您必须分析/商业智能的需求，并希望快速的研究，分析，可视化，并且需要 ad-hoc（即席查询）量数据（像数百万或者数十亿条记录）上的质疑。在这种情况下，您可以使用 Elasticsearch 来存储据，然后使用 Kibana（Elasticsearch/Logstash/Kibana 技术栈中的一部分）以建立一个能够可视化对您很重要的数据方面的定制的 dashboards（面板）。此外，您还可以使用 Elasticsearch 的聚合能对您的数据执行复杂的商业智能查询

对于本教程的其余部分，我将引导您完成 Elasticsearch 的启动和运行的过程，同时了解其原理，并行像 indexing（索引），searching（查询）和 modifing（修改）数据的基础操作。在本教程的最一部分，您应该可以清楚的了解到 Elasticsearch 是什么，它是如何工作的，并有希望获得启发。看如何使用它来构建复杂的搜索应用程序或者从数据中挖掘出想要的信息

基本概念(Basic Concepts)

这里有一些关于 Elasticsearch 的核心概念。从一开始了解这些概念有助于减少学习过程。

Near Realtime（NRT 近实时）

Elasticsearch 是一个近实时的搜索平台。这意味着从您索引一个文档开始直到它可以被查询时会有轻的延迟时间（通常为一秒）。

Cluster（集群）

cluster（集群）是一个或者多个节点的集合，它们一起保存数据并且提供所有节点联合索引以及搜索能。集群存在一个唯一的名字身份且默认为 “elasticsearch”。这个名字非常重要，因为如果节点安时通过它自己的名字加入到集群中的话，那么一个节点只能是一个集群中的一部分。

请确保您在不同环境中不要重复使用相同的集群名字，否则您可能最终会将节点加入到了错误的集群。例如，您可以使用 logging-dev，logging-stage，以及 logging-prod 用于 development（开），staging（演示）和 production（生产）集群。

注意，一个集群如果只有一个结点也是有效的，并且完全可行的。此外，您还可以有多个独立的集群且每个集群都有它自己唯一的 cluster name（集群名）。

Node（节点）

node（节点）是一个单独的服务器，它是集群的一部分，存储数据，参与集群中的索引和搜索功能像一个集群一样，一个节点通过一个在它启动时默认分配的一个随机的 UUID（通用唯一标识符）名来识别。如果您不想使用默认名称您也可自定义任何节点名称。这个名字是要识别网络中的服务器对这在您的 Elasticsearch 集群节点管理的目的是很重要的。

节点可以通过配置 cluster name 来加入到指定的集群中。默认情况下，每个节点安装时都会加入到为 elasticsearch 的集群中，这也就意味着如果您在网络中启动许多节点--假设它们可以发现彼此–它全部将自动的构成并且加入到一个名为 elasticsearch 的单独的集群中。

在一个集群中，你需要多少就可以添加多少结点。此外，如果在当前网络中没有其它 elasticsearch 点在运行，则启动一个结点将会默认形成一个叫 elasticsearch 的单结点集群。

Index（索引）

index（索引）是具有稍微类似特征文档的集合。例如，您有一个消费者数据的索引，一个产品目录索引，和另一个是订单数据的索引。一个索引通过名字（必须全部是小写）来标识，并且该名字在对 ocument（文档）执行 indexing（索引），search（搜索），update（更新）和 delete（删除）作时会涉及到。

在一个单独的集群中，您可以定义您想要的许多索引。

Type（类型）

在 Index（索引）中，可以定义一个或多个类型。一个类型是索引中一个逻辑的种类/分区，它的语义全取决于您自己。一般情况下，一个类型被定义成一组常见字段的文档。例如，假设您运行着一个博平台并且在一个单独的索引中存储了所有的数据。在这个索引中，您也许定义了一个用户数据类型，客数据类型，和评论数据类型。

Document（文档）

document（文档）是索引信息的基本单位。例如，您有一存储 customer（客户）数据的文档，另个是存储 product（产品）数据的文档，还有一个是存储 order（订单）数据的文档。该文档可以使用 JSON 来表示，它是一种无处不在的互联网数据交换格式。

在索引/类型中，您可以存储许多文档。注意，尽管一个文档物理的存在于索引中，实际上一个文档须被 索引/分配 给索引内的类型。

Shards & Replicas（分片 & 副本）

索引可以存储大量数据，可以超过单个节点的硬件限制。例如，十亿个文档占用了 1TB 的磁盘空间单个索引可能不适合放在单个节点的磁盘上，并且从单个节点服务请求会变得很慢。

为了解决这个问题，Elasticsearch 提供了把 Index（索引）拆分到多个 Shard（分片）中的能力。在建索引时，您可以简单的定义 Shard（分片）的数量。每个 Shard 本身就是一个 fully-functional（功能的）和独立的 “Index（索引）”，（Shard）它可以存储在集群中的任何节点上。

Sharding（分片）非常重要两个理由是：

- 水平的拆分/扩展。
- 分布式和并行跨 Shard 操作（可能在多个节点），从而提高了性能/吞吐量。

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

在 网络/云 环境中可能随时会故障，无论出于何种原因，在 shard/node 不知何故会离线或者消失的况下强烈建议设置故障转移是非常有效的。为了达到这个目的，Elasticsearch 可以让您设置一个或多索引的 Shard 副本到所谓的副本分片，或者副本中去。

副本非常重要的两个理由是：

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

总而言之，每个索引可以被拆分成多个分片，一个索引可以设置 0 个（没有副本）或多个副本。开启本后，每个索引将有主分片（被复制的原始分片）和副本分片（主分片的副本）。分片和副本的数量索引被创建时都能够被指定。在创建索引后，您也可以在任何时候动态的改变副本的数量，但是不能改变分片数量。

默认情况下，Elasticsearch 中的每个索引分配了 5 个主分片和 1 个副本，这也就意味着如果您的集至少有两个节点的话，您的索引将会有 5 个主分片和另外 5 个副本分片（1 个完整的副本），每个索共计 10 个分片。

注意：

每个 Elasticsearch 分片是一个 Lucene 索引。在单个 Lucene 索引中有一个最大的文档数量限制。从 LUCENE-5843 的时候开始，该限制为 2,147,483,519（=Interger.MAX_VALUE - 128）个文档。您以使用 _cat/shards api 来监控分片大小。

理解了这些基础概念之后，让我们开始接触更有趣的部分

安装(Installation)

Elasticsearch 需要的 Java 最低版本为 Java 8。在该文档中，建议您使用 Oracle JDK version 1.8.0_31。Java 的安装因平台而异，所以我们不会在这里详细的介绍。在 Oracle 的官网 上可以找到 Oracle 的推荐安装文档。我想说的是，在安装 Elasticsearch 之前，首先请您检查的 Java 版本，运行以下命：

```
java -version
echo $JAVA_HOME
```

如果 Java 安装完成之后，我们可以下载并运行 Elasticsearch。二进制包与所有历史版本都可以从 www.elastic.co/downloads 中获得。对于每一个版本，您可以选择 zip，tar，DEB 或者 RPM 包。为简单期间，我们使用 tar 文件。

让我们下载 Elasticsearch 5.4.0 tar，如下所示（Windows 用户应该下载 zip 包）：

```
curl -L -O https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.4.0.tar.gz
```

然后抽取文件，如下所示（Windows 用户应该 unzip（解压缩）该 zip 包）：

```
tar -xvf elasticsearch-5.4.0.tar.gz
```

然后会在您当前目录中创建一堆文件和文件夹。然后转到 bin 目录中起，如下所示：

```
cd elasticsearch-5.4.0/bin
```

现在我们准备启动我们的节点，单个集群（Windows 用户应该运行 elasticsearch.bat 文件）：

./elasticsearch

如果一切顺利，你应该看到一堆看起来像下面的消息：

```
[2016-09-16T14:17:51,251][INFO ][o.e.n.Node               ] [] initializing ...
[2016-09-16T14:17:51,329][INFO ][o.e.e.NodeEnvironment    ] [6-bjhwl] using [1] data paths, mounts [[/ (/dev/sda1)]], net usable_space [317.7gb], net total_space [453.6gb], spins? [no], type [ext4]
[2016-09-16T14:17:51,330][INFO ][o.e.e.NodeEnvironment    ] [6-bjhwl] heap size [1.9gb], compressed ordinary object pointers [true]
[2016-09-16T14:17:51,333][INFO ][o.e.n.Node               ] [6-bjhwl] node name [6-bjhwl] derived from node ID; set [node.name] to override
[2016-09-16T14:17:51,334][INFO ][o.e.n.Node               ] [6-bjhwl] version[5.0.0], pid[21261], build[f5daa16/2016-09-16T09:12:24.346Z], OS[Linux/4.4.0-36-generic/amd64], JVM[Oracle Corporation/Java HotSpot(TM) 64-Bit Server VM/1.8.0_60/25.60-b23]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [aggs-matrix-stats]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [ingest-common]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [lang-expression]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [lang-groovy]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [lang-mutache]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [lang-painless]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [percolator]
[2016-09-16T14:17:51,968][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [reindex]
[2016-09-16T14:17:51,968][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [transport-netty3]
[2016-09-16T14:17:51,968][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded module [transport-netty4]
[2016-09-16T14:17:51,968][INFO ][o.e.p.PluginsService     ] [6-bjhwl] loaded plugin [mapper-murmur3]
[2016-09-16T14:17:53,521][INFO ][o.e.n.Node               ] [6-bjhwl] initialized
[2016-09-16T14:17:53,521][INFO ][o.e.n.Node               ] [6-bjhwl] starting ...
[2016-09-16T14:17:53,671][INFO ][o.e.t.TransportService   ] [6-bjhwl] publish_address {192.168.8.112:9300}, bound_addresses {{192.168.8.112:9300}
[2016-09-16T14:17:53,676][WARN ][o.e.b.BootstrapCheck     ] [6-bjhwl] max virtual memory areas vm.max_map_count [65530] likely too low, increase to at least [262144]
[2016-09-16T14:17:56,731][INFO ][o.e.h.HttpServer         ] [6-bjhwl] publish_address {192.168.8.112:9200}, bound_addresses {[::1]:9200}, {192.168.8.112:9200}
[2016-09-16T14:17:56,732][INFO ][o.e.g.GatewayService     ] [6-bjhwl] recovered [0] indices into cluster_state
[2016-09-16T14:17:56,748][INFO ][o.e.n.Node               ] [6-bjhwl] started
```

没有过多的细节，我们可以看到我们节点名 “6-bjhwl” （在您的例子中将会不一样）已经启动，并选举它自己作为一个单个集群中的 Master。不用担心目前该 Master 到底意味着什么。这里重要的情是我们已经在一个集群上启动了一个节点。

像前面提到的一样，我们可以覆盖集群或者节点的名称。在命令行中启动 Elasticsearch 时是可以做的，如下所示：

```
./elasticsearch -Ecluster.name=my_cluster_name -Enode.name=my_node_name
```

另外需要注意到 http 那行关于 HTTP 地址（192.168.8.112）和端口（9200）的信息。默认情况下，lasticsearch 使用端口 9200 来访问它的 REST API。如果有必要，该端口也可以配置。

Exploring Your Cluster（探索集群）

The REST API

既然我们已经启动并且运行了我们的节点（和集群），下一步是去了解如何与它通信。幸运的是，Elasticsearch 提供了一个非常全面且强大的 REST API，您可以使用它来与集群进行交互。可以使用 API 完成如下的几件事情：

- 检查集群，节点，和索引的健康，状态和统计信息。
- 管理集群，节点和索引数据以及元数据。
- 针对索引执行 CRUD（Create，Read，Update，和 Delete）和搜索操作。
- 执行高级搜索，例如 paging，sorting，filtering，scripting，aggregations 等等。

Cluster Health（集群健康）

集群健康

让我们从基本的健康检查开始，我们可以用它来看看我们的集群在做什么。我们将使用 curl 来做这件情，当然您也可以使用任何允许您进行 HTTP/REST 调用的工具。假设我们在同一节点上启动了 Elasticsearch 并且打开了另一个命令 shell 窗口。

为了检查集群健康，我们将使用 _cat API。您可以在 Kibana’s Console 通过点击 “VIEW IN CONSOLE” 或者通过点击 “COPY AS CURL” 链接然后粘贴到终端中使用 curl 中运行命令：

```
curl -XGET 'localhost:9200/_cat/health?v&pretty'
```

响应如下：

```
epoch      timestamp cluster       status node.total node.data shards pri relo init unassign pending_tasks max_task_wait_time active_shards_percent
1475247709 17:01:49  elasticsearch green           1         1      0   0    0    0        0             0                  -                100.0%
```

我们可以看到我们名为 “elasticsearch” 的集群与 green 的 status。

无论何时我们请求集群健康，我们可以获得 green，yellow，或者 red 的 status。Green 表示一切常（集群功能齐全），yellow 表示所有数据可用，但是有些副本尚未分配（集群功能齐全），red 味着由于某些原因有些数据不可用。注意，集群是 red，它仍然具有部分功能（例如，它将继续从可的分片中服务搜索请求），但是您可能需要尽快去修复它，因为您已经丢失数据了。

另外，从上面的响应中，我们可以看到共计 1 个 node（节点）和 0 个 shard（分片），因为我们还有放入数据的。注意，因为我们使用的是默认的集群名称（elasticsearch），并且 Elasticsearch 默情况下使用 unicast network（单播网络）来发现同一机器上的其它节点。有可能您不小心在您的电上启动了多个节点，然后它们全部加入到了单个集群。在这种情况下，你会在上面的响应中看到不止 1 个 node（节点）。

我们也可以获取我们集群的节点列表，如下所示：

```
curl -XGET 'localhost:9200/_cat/nodes?v&pretty'
```

响应如下：

```
ip        heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
127.0.0.1           10          5   5    4.46                    mdi      *      PB2SGZY
```

在这里，我们可以到我名为 “I8hydUG” 的节点名，这是目前在我们集群中的单个节点。

List All Indices（列出所有索引）

现在，让我们来看一看我们的索引：

```
curl -XGET 'localhost:9200/_cat/indices?v&pretty'
```

响应如下：

```
health status index uuid pri rep docs.count docs.deleted store.size pri.store.size
```

这意味着我们的集群中还没有索引。

Create an Index（创建索引）

现在让我们创建一个名为 “customer” 的索引，然后再列出所有索引：

```
curl -XPUT 'localhost:9200/customer?pretty&pretty'
curl -XGET 'localhost:9200/_cat/indices?v&pretty'
```

这是第一个使用 PUT 动作命令创建名为 “customer” 的索引。我们简单的添加 pretty 到调用命令末尾，来告诉它漂亮的打印成 JSON 响应（如果有的话）。

响应如下：

```
health status index    uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   customer 95SQ4TSUT7mWBT7VNHH67A   5   1          0            0       260b           260b
```

第一个命令的结果告诉我们现在已经有 1 个名为 customer 的索引，并且它有 5 个主分片和 1 个副（默认）以及它包含了 0 文档在索引中。

您可以也注意到 customer 索引有一个 yellow 标记在索引中。回想下我们先前讨论的 yellow 意味着些副本没有被分配。该索引发生这种情的原因是因为 Elasticsearch 默认为该索引创建了 1 个副本。为目前我们只有一个节点在运行，这一个副本不能够被分配（为了高可用性），直到稍候其它节点加到集群。如果副本被分配到第二个节点，该索引的 heath status（健康状态）将会转换成 green。

Index and Query a Document（索引和查询文档）

现在让我们放入一些东西动我们的 customer 索引中去。回想一下前面的内容，为了索引一个文档，们必须告诉 Elasticsearch

在索引中应该使用哪种类型。

让我们索引一个简单的 customer 文档到 customer 索引中， “external” 类型，与一个为 1 的 ID

如下所示：

```
curl -XPUT 'localhost:9200/customer/external/1?pretty&pretty' -d'
{
  "name": "John Doe"
}'
```

响应如下：

```
{
  "_index" : "customer",
  "_type" : "external",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "created" : true
}
```

从上面我们可以看到一个新的 customer 文档被成功的创建到 customer 索引和 external 类型中。文档还有一个为 1 的内部 ID，它是我们在索引时指定的。

需要注意的是，在您可以索引文档到 Elasticsearch 之前时，它不需要您首先就明确的创建一个索引在前面的例子中，Elasticsearch 将自动的创建 customer 索引（如果它事先不存在）。

现在让我们检索我们刚刚索引的文档：

```
curl -XGET 'localhost:9200/customer/external/1?pretty&pretty'
```

响应如下：

```
{
  "_index" : "customer",
  "_type" : "external",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : { "name": "John Doe" }
}
```

除了一个字段 found 之外，没有什么特别的东西，它指出了我们使用 ID 为 1 请求的状态，另一个字，_source，它返回了我们先前步骤中索引的全部的 JSON 文档。

Delete an Index（删除索引）

现在让我们删除我们刚才创建的索引并且再次列出所有索引：

```
curl -XDELETE 'localhost:9200/customer?pretty&pretty'
curl -XGET 'localhost:9200/_cat/indices?v&pretty'
```

响应如下：

```
health status index uuid pri rep docs.count docs.deleted store.size pri.store.size
```

这意味着索引被成功的删除，并且我们现在又回到了集群中什么都没有的时候了。

在我们继续之前，让我们仔细的看一看一些我们迄今为止学习的 API 命令：

```
curl -XPUT 'localhost:9200/customer?pretty'
curl -XPUT 'localhost:9200/customer/external/1?pretty' -d'
{
  "name": "John Doe"
}'
curl -XGET 'localhost:9200/customer/external/1?pretty'
curl -XDELETE 'localhost:9200/customer?pretty'
```

如果我们仔细的研究上面的命令，我们可以清楚的看到，我们如何访问 Elasticsearch 中的数据的 pattern（模式）。该 pattern（模式）可以概括如下：

```
<REST Verb> /<Index>/<Type>/<ID>
```

在所有的 API 命令中这个 REST 访问模式是很常见的，如果您可以简单的记住它，在掌握 Elasticsearch 时您会有一个良好的开端。

Modifying Your Data（修改数据）

Elasticsearch 提供了近实时的操纵数据和搜索的能力。默认情况下，从您 index/update/delete 数时到搜索结果中出现会有 1 秒的延迟（刷新间隔）。这是从其他平台例如 SQL 其中一个事务完成后据是立即可用的一个重要区别。

Indexing/Replacing Documents（索引/替换 文档）

我们先前看到过我们是如何索引单个文档的。让我们再次调用该命令：

```
curl -XPUT 'localhost:9200/customer/external/1?pretty&pretty' -d'
{
  "name": "John Doe"
}'
```

同样，上面的命令将索引指定的文档到 customer 索引，external 类型，ID 为 1 中。如果我们稍候用不同的（或者相同的）文档来执行上面的命令，Elasticsearch 将在已存在的 ID 为 1 的文档替换（如，reindex）成一个新的文档。

```
curl -XPUT 'localhost:9200/customer/external/1?pretty&pretty' -d'
{
  "name": "Jane Doe"
}'
```

上面将 ID 为 1 文档的名字从 “John Doe” 改成 “Jane Doe”。如果，在另一方面，我们使用不的 ID，一个新的文档将会被索引并且已存在的文档仍然在索引中保持不变。

```
curl -XPUT 'localhost:9200/customer/external/2?pretty&pretty' -d'
{
  "name": "Jane Doe"
}'
```

上面将索引一个 ID 为 2 的新文档。

在索引时，ID 是可选项的一部分。如果不指定，Elasticsearch 将生产一个随机 ID，然后使用它去索文档。Elasticsearch 生成的真实的 ID（或者我们在先前的例子中明确的指定）将作为索引 API 调用一部分返回。

该例子演示了在没有明确的 ID 的情况下如何去索引一个文档：

```
curl -XPOST 'localhost:9200/customer/external?pretty&pretty' -d'
{
  "name": "Jane Doe"
}'
```

注意，在上述的情况下，我们使用了 POST 动作，而不是 PUT，因为我们没有指定 ID。

Updating Documents（更新文档）

除了可以索引和替换文档之外，我们也可以更新文档。不过，请注意尽管 Elasticsearch 没有在后台上更新。每当我们做一次更新，Elasticsearch 删除旧的文档，然后一次性应用更新索引一个新文档。

这个例子演示了如何去更新我们先前的文档（ID 为 1），通过修改 name 字段的值为 “Jane Doe”：

```
curl -XPOST 'localhost:9200/customer/external/1/_update?pretty&pretty' -d'
{
  "doc": { "name": "Jane Doe" }
}'
```

这个例子演示了如何去更新我们先前的文档（ID 为 1），通过修改 name 字段的值为 “Jane Doe” 并且同时添加 age 字段：

```
curl -XPOST 'localhost:9200/customer/external/1/_update?pretty&pretty' -d'
{
  "doc": { "name": "Jane Doe", "age": 20 }
}'
```

更新也是通过使用简单的简本来执行。这个例子演示了使用简本来将 age 加 5：

```
curl -XPOST 'localhost:9200/customer/external/1/_update?pretty&pretty' -d'
{
  "script" : "ctx._source.age += 5"
}'
```

在上面的例子中，ctx._source 代表当前将被更新的源文档。

请注意，在编写该文档时，在同一时间中更新只能够被执行一个文档上。在将来，Elasticsearch 也许提供给定一个查询条件（像 SQL UPDATE-WHERE 语句）来更新多个文档的功能。

Deleting Documents（删除文档）

删除文档是很简单的，这个例子演示了如何去删除我们先前 ID 为 2 的 customer（的文档）。

```
curl -XDELETE 'localhost:9200/customer/external/2?pretty&pretty'
```

请看 Delete By Query API 以删除指定查询匹配到的所有文档。值得注意的是，它是删除一个完整的引更有效的方式，而不是使用 Delete By Query API 来删除所有的文档。

Batch Processing（批处理）

除了可以索引，更新，和删除单个的文档之外，Elasticsearch 还提供了使用 _bulk API 来执行上面任操作的批量方式的能力。这个功能是非常重要，它提供了一种非常有效的机制，以尽可能减少网络返且尽可能快的执行多个操作。

作为一个简单的例子，在一个批量操作中下面调用且索引了两个文档（ID 1 - John Doe 和 ID 2 - Jan Doe）：

```
curl -XPOST 'localhost:9200/customer/external/_bulk?pretty&pretty' -d'
{"index":{"_id":"1"}}
{"name": "John Doe" }
{"index":{"_id":"2"}}
{"name": "Jane Doe" }
'
```

这个例子使用一个批量操作更新了第一个文档（ID 为 1），然后删除了第二个文档（ID 为 2）：

```
curl -XPOST 'localhost:9200/customer/external/_bulk?pretty&pretty' -d'
{"update":{"_id":"1"}}
{"doc": { "name": "John Doe becomes Jane Doe" } }
{"delete":{"_id":"2"}}
'
```

请注意上面的删除操作，在它后面没有相关的源文档，因为删除操作只需要文档的 ID 来删除。

该 bulk API 以此按顺序执行所有的 action（动作）。如果一个单个的动作因任何原因而失败，它将续处理它后面剩余的动作。当 bulk API 返回时，它将提供每个动作的状态（与发送的顺序相同），以您可以检查是否一个指定的动作是不是失败了。

Exploring Your Data（探索数据）

Sample Dataset（样本数据集）

现在我们已经学会了基础知识，让我们尝试在更真实的数据集上操作。我准备了一份顾客银行账户信的虚构的 JSON 文档样本。每个文档都有下列的

schema（模式）：

```
{
    "account_number": 0,
    "balance": 16623,
    "firstname": "Bradshaw",
    "lastname": "Mckenzie",
    "age": 29,
    "gender": "F",
    "address": "244 Columbus Place",
    "employer": "Euron",
    "email": "bradshawmckenzie@euron.com",
    "city": "Hobucken",
    "state": "CO"
}
```

如果您对这份数据有兴趣，我从 www.json-generator.com/ 生成的这份数据，因为这些都是随机生

的，所以请忽略实际的值和数据的语义。

Loading the Sample Dataset（加载样本数据集）

您可以从这里下载这份样本数据集（accouts.json）。抽取它到我们当前的目录，然后加载它到我们群中，如下所示：

```
curl -XPOST 'localhost:9200/bank/account/_bulk?pretty&refresh' --data-binary "@accounts.json"
curl 'localhost:9200/_cat/indices?v'
```

响应如下：

```
health status index uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   bank  l7sSYV2cQXmu6_4rJWVIww   5   1       1000            0    128.6kb        128.6kb
```

这样的响应表示我们刚才成功的批量索引了 1000 份文档到 bank 索引（account 类型下）。

The Search API（搜索 API）

现在让我们从一些简单的搜索开始。这里两个运行搜索的基本方法：一个是通过使用 REST request URI 发送搜索参数，另一个是通过使用 REST request body 来发送它们。请求体的方法可以让您更具表现力，并且可以在一个更可读的 JSON 格式中定义您的搜索。我们会尝试使用一个 REST request URI 的示例，但是在本教程的其它部分，我们将只使用 REST request body 的方法。

搜索的 REST API 从 _search 的尾部开始。这个示例返回了 bank 索引中的所有文档：

```
curl -XGET 'localhost:9200/bank/_search?q=*&sort=account_number:asc&pretty'
```

首先让我们切开搜索的调用。我们在 bank 索引中执行搜索（_search 尾部），然后 q=* 参数命令 Elasticsearch 去匹配索引中所有的文档。pretty 参数，再一次告诉 Elasticsearch 去返回打印漂亮的 JSON 结果。

响应如下（部分）：

```
{
  "took" : 63,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1000,
    "max_score" : null,
    "hits" : [ {
      "_index" : "bank",
      "_type" : "account",
      "_id" : "0",
      "sort": [0],
      "_score" : null,
      "_source" : {"account_number":0,"balance":16623,"firstname":"Bradshaw","lastname":"Mcknzie","age":29,"gender":"F","address":"244 Columbus Place","employer":"Euron","email":"brad
```

hawmckenzie@euron.com","city":"Hobucken","state":"CO"}
    }, {
      "_index" : "bank",
      "_type" : "account",
      "_id" : "1",
      "sort": [1],
      "_score" : null,
      "_source" : {"account_number":1,"balance":39225,"firstname":"Amber","lastname":"Duke","
ge":32,"gender":"M","address":"880 Holmes Lane","employer":"Pyrami","email":"amberduke@
yrami.com","city":"Brogan","state":"IL"}
    }, ...
    ]
  }
}

在响应中，我们可以看到以下几个部分：

- took - Elasticsearch 执行搜索的时间（毫秒）
- time_out - 告诉我们搜索是否超时
- _shards - 告诉我们多少个分片被搜索了，以及统计了成功/失败的搜索分片
- hits - 搜索结果
- hits.total - 搜索结果
- hits.hits - 实际的搜索结果数组（默认为前 10 的文档）
- sort - 结果的排序 key（键）（没有则按 score 排序）
- score 和 max_score

现在暂时忽略这些字段

这里是上面相同的搜索，使用了 REST request body 方法：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match_all": {} },
  "sort": [
    { "account_number": "asc" }
  ]
}'
```

这里不同的地方是而不是在 URL 中传递 q=*，我们 POST 一个 JSON 风格的查询请求体到 _search PI。我们将在下一部分讨论这个 JSON 查询。

需要了解，一旦您搜索的结果被返回，Elasticsearch 完成了这次请求，并且不会维护任何服务端的资或者结果的 cursor（游标）。这与其它的平台形成了鲜明的对比，例如 SQL，您可能首先获得查询果的子集，如果您想要使用一些服务端有状态的 cursor（光标）来抓取（或者通过分页）其它的结果然后您必须再次回到服务器。

Introducing the Query Language（介绍查询语言）

Elasticsearch 提供了一个可以执行查询的 Json 风格的 DSL（domain-specific language 领域特定言）。这个被称为 Query DSL。该查询语言非常全面，并且刚开始的时候感觉有点复杂，真正学好它

方法是从一些基础的示例开始的。

回到我们上个例子，我们执行了这个查询：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match_all": {} }
}'
```

分析上面的查询，query 部分告诉我们查询是如何定义的，match_all 部分就是我们要运行查询的简的类型。该 match_all 查询简单的搜索了指定所有的所有文档。

除了 query 参数之外，我们也可以传递其它的参数以改变查询结果。在上部分的例子中我们传递了 so t，这里我们将传递 size：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match_all": {} },
  "size": 1
}'
```

注意，如果不指定 size，默认为 10.

下面的例子做了一个 match_all 并且返回了

11 到 20 的文档。

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match_all": {} },
  "from": 10,
  "size": 10
}'
```

该 from 参数（0 为基础）指定了文档开始的编号，size 参数指定了从 from 参数开始有多少个文档返回。在实现搜索结果分页时这个功能是很有用的。注意，如果不指定 from 参数，默认为 0。

下面的例子做了一个 match_all，以及对结果通过账户余额按降序排序，并且返回了 top 10（默认大）的文档。

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match_all": {} },
  "sort": { "balance": { "order": "desc" } }
}'
```

Executing Searches（执行查询）

现在我们已经了解了一下基本的搜索参数，让我们深入探讨更多的 DSL。我们首先看一下返回的文档字段。默认情况下，完成的 JSON 文档被作为所有搜索的一部分返回。这被称为源（在搜索 hits 中的 source 字段）。如果我们不希望整个源文档返回，我们可以只请求源中的一些字段返回。

下面的例子演示了如何返回两个字段，account_number 和 balance（在 _source 之内），从 search 开始，如下所示：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match_all": {} },
  "_source": ["account_number", "balance"]
}'
```

注意，上面的例子减少了 _source 字段，它将仍然只返回一个名为 _source 的字段，但是不在它里，仅包括 account_number 和 balance 字段。

如果您拥有 SQL 基础，上面的地方与 SQL SELECT FROM 字段列表的概念相似。

现在让我们继续查询部分。此前，我们了解了 match_all 是如何匹配所有文档的。我们现在介绍一个为 match query 的新的查询，可以认为它是搜索查询的一个基础的字段（例如，针对一个指定的字或一组字段来搜索）。

这个例子返回了账户编号为 20 的文档：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match": { "account_number": 20 } }
}'
```

这个例子返回了所有在 address 中包含 term 为 “mill” 的账户：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match": { "address": "mill" } }
}'
```

这个例子返回了所有在 address 中包含了 term 为 “mill” 或 “lane” 的账户：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match": { "address": "mill lane" } }
}'
```

这个例子是 match（match_phrase）的另一种方式，它返回了在 address 中所有包含 phrase 为 “ill lane” 的账户：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": { "match_phrase": { "address": "mill lane" } }
}'
```

现在我们介绍 bool(ean) query。该 bool 查询可以让我们使用 boolean 逻辑构建较小的查询到更大查询中去。

这个例子构建两个 match 查询，并且回了所有在 address 中包含了 “mill” 和 “lane” 的账户：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
```

```
      ]
    }
  }
}'
```

在上面的例子中，bool must 语句指定了所有查询必须为 true 时将匹配到文档。

相反，这个例子构建两个 match 查询，并且回了所有在 address 中包含了 “mill” 或 “lane” 的户：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": {
    "bool": {
      "should": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'
```

在上面的例子中，bool should 语句指定了一个查询列表，两者中的一个为 true 时将匹配到文档。

这个例子构建两个 match 查询，并且回了所有在 address 中既不包含 “mill” 也不包含 “lane” 账户：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": {
    "bool": {
      "must_not": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'
```

在上面的例子中，bool must_not 语句指定了一个查询列表，都不为 true 将匹配到文档。

我们可以在 bool 查询中同时联合 must，should，和 must_not 语句。

此外，我们可以在任何一个 bool 语句内部构建 bool 查询，从而模拟出任何复杂的多级别的 boolean 逻辑。

这个例子返回了 age 为 40 但是 state 不为 ID 的账户：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
```

```
      "must_not": [
        { "match": { "state": "ID" } }
      ]
    }
  }
}'
```

Executing Filters（执行过滤）

在上一节中，我们跳过了一些名为文档分数（在搜索结果中的 _score 字段）的细节。score（分数是一个我们指定搜索后匹配的文档的相对的效果评估的数值型的值。分数越高，文档的相关度更高，数越低，文档的相关度越低。

并不是所有的查询都需要产生分数，特别是那些仅用于 “filtering” （过滤）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。

在上一节中我们介绍的 bool query 也支持 filter 语句，它可以使用一个查询来限制会被其它语句匹的文档，而不改变分数是如何计算出来的。举个例子，让我们介绍下 range query，它可以让我们通一系列的值过滤文档。这通常用于数字或者日期过滤。

这个例子使用了一个 bool 查询来返回余额在 20000 ~ 30000 直接的账户（包含 20000 和 30000）换言之，我们想要去找出余额大于或等于 20000 且小于或等于 30000 的账户。

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "query": {
    "bool": {
      "must": { "match_all": {} },
      "filter": {
        "range": {
          "balance": {
            "gte": 20000,
            "lte": 30000
          }
        }
      }
    }
  }
}'
```

分析上面的结构，bool 号查询包含了一个 match_all 查询（查询部分），和一个 range（范围）查（过滤部分）。我们可以将查询和过滤部分替换成任何其它的查询。在上述情况下，范围查询是非常理的，因为文档落入的所有匹配范围是 “equally” （相等的）。例如，没有文档与其它的东西相关。

除了 match_all，match，bool 和 range 查询之外 ，还有很多在这里我们我没有用到的其它有用的询类型。既然我们对于它们是如何工作的方式有了一个基本的了解，在学习和尝试其它的查询类型中用这些知识应该不是很难。

Executing Aggregations（执行聚合）

聚合提供了从数据中分组和提取数据的能力。最简单的聚合方法大致等于 SQL GROUP BY 和 SQL 合函数。在 Elasticsearch 中，您有执行搜索返回 hits（命中结果），并且同时返回聚合结果，把一

响应中的所有 hits（命中结果）分隔开的能力。这是非常强大且有效的，您可以执行查询和多个聚合并且在一次使用中得到各自的（任何一个的）返回结果，使用一次简洁和简化的 API 来避免网络往返。

作为开始，找这个例子按 state 将所有的 account 给分组了，然后按 count 降序（默认）排序返回 top 10（默认）的 state：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state.keyword"
      }
    }
  }
}'
```

在 SQL 中，上面的聚合概念类似下面：

```
SELECT state, COUNT(*) FROM bank GROUP BY state ORDER BY COUNT(*) DESC
```

响应如下（部分显示）：

```
{
  "took": 29,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits" : {
    "total" : 1000,
    "max_score" : 0.0,
    "hits" : [ ]
  },
  "aggregations" : {
    "group_by_state" : {
      "doc_count_error_upper_bound": 20,
      "sum_other_doc_count": 770,
      "buckets" : [ {
        "key" : "ID",
        "doc_count" : 27
      }, {
        "key" : "TX",
        "doc_count" : 27
      }, {
        "key" : "AL",
        "doc_count" : 25
      }, {
        "key" : "MD",
        "doc_count" : 25
      }, {
        "key" : "TN",
```

```
      "doc_count" : 23
    }, {
      "key" : "MA",
      "doc_count" : 21
    }, {
      "key" : "NC",
      "doc_count" : 21
    }, {
      "key" : "ND",
      "doc_count" : 21
    }, {
      "key" : "ME",
      "doc_count" : 20
    }, {
      "key" : "MO",
      "doc_count" : 20
    } ]
  }
 }
}
```

我们可以看到上面在 ID（Idaho）中有 27 个 account（账户），接着有 27 个 account（账户）在 T（Texas）中，25 个账户在 AL（Alabama）中，等等。

注意我们设置了 size=0 以不显示搜索的 hits（命中数量），因为我们只希望在响应中看聚合结果。

基于前面的聚合，这个例子按 state 计算了平均的账户余额（再一次按 count 降序（默认）排序返回 op 10（默认）的 state ）：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
 "size": 0,
 "aggs": {
  "group_by_state": {
   "terms": {
    "field": "state.keyword"
   },
   "aggs": {
    "average_balance": {
     "avg": {
      "field": "balance"
     }
    }
   }
  }
 }
}'
```

注意，我们是如何内嵌 average_balance 聚合到 group_by_state 聚合的内部的。这是所有聚合中见的模式。您可以嵌入聚合到随意的聚合中以从您需要的数据中开窗汇总。

基于前面的聚合，现在让我们在 average balance（平均余额）上降序排序：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -d'
{
```

```
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state.keyword",
        "order": {
          "average_balance": "desc"
        }
      },
      "aggs": {
        "average_balance": {
          "avg": {
            "field": "balance"
          }
        }
      }
    }
  }
}'
```

这个例子演示了我们如何按年龄段（20-29岁，30-39岁，和 40-49岁），然后按性别，最后获得每年龄段，每个性别的平均账户余额：

```
GET /bank/_search
{
  "size": 0,
  "aggs": {
    "group_by_age": {
      "range": {
        "field": "age",
        "ranges": [
          {
            "from": 20,
            "to": 30
          },
          {
            "from": 30,
            "to": 40
          },
          {
            "from": 40,
            "to": 50
          }
        ]
      },
      "aggs": {
        "group_by_gender": {
          "terms": {
            "field": "gender.keyword"
          },
          "aggs": {
            "average_balance": {
              "avg": {
                "field": "balance"
```

```
            }
          }
        }
      }
    }
  }
}
```

还有一些我们这里没有细讲的其它的聚合功能。如果您想要做进一步的实验，聚合参考指南是一个比好的起点。

Conclusion（总结）

Elasticsearch 既是一个简单，又是一个复杂的产品。我们现在学会了它的基础部分，如何去看它内部以及如何使用一些 REST API 来操作它。我希望本教程可以让您更好的了解 Elasticsearch 是什么，重要的是，可以促使你进一步尝试它更强大的功能。