

Elasticsearch 6.1官方入门教程 - 程序猿洞穴

2

入门

Elasticsearch是一个高度可伸缩的开源全文搜索和分析引擎。它允许你以近实时的方式快速存储、搜索和分析大量的数据。它通常被用作基础的技术来赋予应用程序复杂的搜索特性和需求。

这里列举了几个Elasticsearch可以用来做的功能例子

- 你有一个在线网上商城，提供用户搜索你所卖的商品功能。在这个例子中，你可以使用Elasticsearch去存储你的全部的商品目录和存货清单并且提供搜索和搜索自动完成以及搜索推荐功能。
- 你想去收集日志或者业务数据，并且去分析并从这些数据中挖掘寻找市场趋势、统计资料、摘要信息或者反常情况。在这个例子中，你可以使用Logstash(part of the Elasticsearch/Logstash/Kibana stack)去收集、聚合并且解析你的数据，然后通过Logstash将数据注入Elasticsearch。一旦数据进入Elasticsearch，你就可以运行搜索和聚集并且从中挖掘任何你感兴趣的数据。
- 你运行一个价格预警平台，它可以让那些对价格精明的客户指定一个规则，比如：“我相中了一个电子产品，并且我想在下个月任何卖家的这个电子产品的价格低于多少钱的时候提醒我”。在这个例子中，你可以抓取所有卖家的价格，把价格放入Elasticsearch并且使用Elasticsearch的反向搜索(过滤器/抽出器)功能来匹配价格变动以应对用户的查询并最终一旦发现匹配结果时给用户弹出提示框。
- 你有分析学/商业情报的需求并且想快速审查、分析并使用图像化进行展示，并且在一个很大的数据集上查询点对点的问题(试想有百万或千万的记录)。在这个例子中，你可以使用Elasticsearch去存储你的数据然后使用Kibana(part of the Elasticsearch/Logstash/Kibana stack)去构建定制化的仪表盘。这样你就可以很直观形象的了解对你重要的数据。此外，你可以使用Elasticsearch的集成功能，靠你的数据去展现更加复杂的商业情报查询。

在剩余的本教程中，你将会被引导去把Elasticsearch安装好并运行起来，并且对它有一个简单的了解。使用基础的操作比如索引，搜索以及修改你的数据。当你完成这个教程的时候，你应该对Elasticsearch有了一个不错的了解，它是怎么工作的，并且我们希望它能够使用它构建出更加复杂精致的搜索应用来挖掘你的数据里的价值。

基础概念

这里有一些Elasticsearch的核心概念。在一开始理解这些概念将会极大的使你的学习过程变得更加轻松。

- 近实时性(Near Realtime[NRT])

Elasticsearch是一个近实时的搜索平台。这意味着当你导入一个文档并把它变成可搜索的时间仅会有轻微的延时。

- 集群(Cluster)

一个集群是由一个或多个节点(服务器)组成的，通过所有的节点一起保存你的全部数据并且提供联合索引和搜索功能的节点集合。每个集群有一个唯一的名称标识，默认是“elasticsearch”。这个名称非常重要，因为一个节点(Node)只有设置了这个名称才能加入集群，成为集群的一部分。

确保你没有在不同的环境下重用相同的名称，否则你最终可能会将节点加入错误的集群。例如你可以使用logging-dev, logging-stage和logging-prod来分别给开发，展示和生产集群命名。

注意，一个集群中只有一个节点是有效的并且是非常好的。所以这样的话，你可能需要部署多个集群并且每个集群有它们唯一的集群名称。

- 节点(Node)

一个节点是一个单一的服务器，是你的集群的一部分，存储数据，并且参与集群的索引和搜索功能。跟集群一样，节点在启动时也会被分配一个唯一的标识名称，这个名称默认是一个随机的UUID(Universally Unique Identifier)。如果你不想用默认的名称，你可以自己定义节点的名称。这个名称对于管理集群节点，识别哪台服务器对应集群中的哪个节点有重要的作用。

一个节点可以通过配置特定的集群名称来加入特定的集群。默认情况下，每个节点被设定加入一个名称为“elasticsearch”的集群，这意味着如果你在你的网络中启动了一些节点，并且假设它们能相互发现，它们将会自动组织并加入一个名称是“elasticsearch”的集群。

在一个集群中，你想启动多少节点就可以启动多少节点。此外，如果没有其它节点运行在当前网络中，只启动一个节点将默认形成一个新的名称为“elasticsearch”单节点集群。

- 索引(Index)

一个索引就是含有某些相似特性的文档的集合。例如，你可以有一个用户数据的索引，一个产品目录的索引，还有其他的有规则数据的索引。一个索引被一个名称(必须都是小写)唯一标识，并且这个名称被用于索引通过文档去执行索引，搜索，更新和删除操作。

在一个集群中，你可以根据自己的需求定义任意多的索引。

- 类型(Type)[Deprecated in 6.0.0]

警告！Type在6.0.0版本中已经不建议使用

一个类型是你的索引中的一个分类或者说是一个分区，它可以让你在同一索引中存储不同类型的文档，例如，为用户建一个类型，为博客文章建另一个类型。现在已不可能在同一个索引中创建多个类型，并且整个类型的概念将会在未来的版本中移除。查看“映射类型的移除[\[https://www.elastic.co/guide/...\]](https://www.elastic.co/guide/...)”了解更多。

- 文档(Document)

一个文档是一个可被索引的数据的基础单元。例如，你可以给一个单独的用户创建一个文档，给单个产品创建一个文档，以及其他的单独的规则。这个文档用JSON格式表现，JSON是一种普遍的网络数据交换格式。

在一个索引或类型中，你可以根据自己的需求存储任意多的文档。注意，虽然一个文档在物理存储上属于一个索引，但是文档实际上必须指定一个在索引中的类型。

- 分片和复制(Shards & Replicas)

我们在一个索引里存储的数据，潜在的情况下可能会超过单节点硬件的存储限制。例如，单个索引有上千万个文档需要占用1TB的硬盘存储空间，但是一台机器的硬盘可能没有这么大，或者是即便有这么大，但是单个节点在提供搜索服务时会响应缓慢。

为了解决这个问题，Elasticsearch提供了分片的能力，它可以将你的索引细分成多个部分。当你创建一个索引的时候，你可以简单的定义你想要的分片的数量。每个分片本身是一个全功能的完全独立的“索引”，它可以部署在集群中的任何节点上。

分片对于以下两个主要原因很重要：

- 它允许你水平切分你的内容卷
- 它允许你通过分片来分布和并行化执行操作来应对日益增长的执行量

一个分片是如何被分配以及文档又是如何被聚集起来以应对搜索请求的，它的实现技术由Elasticsearch完全管理，并且对用户是透明的。

在一个网络环境下或者是云环境下，故障可能会随时发生，有一个故障恢复机制是非常有用并且是高度推荐的，以防一个分片或节点不明原因下线，或者因为一些原因去除了。为了达到这个目的，Elasticsearch允许你制作分片的一个或多个拷贝放入一个叫做复制分片或短暂复制品中。

复制对于以下两个主要原因很重要：

- 高可用。它提供了高可用来以防分片或节点宕机。为此，一个非常重要的注意点是绝对不要将一个分片的拷贝放在跟这个分片相同的机器上。
- 高并发。它允许你的分片可以提供超出自身吞吐量的搜索服务，搜索行为可以在分片所有的拷贝中并行执行。

总结一下，每个索引可以被切分成多个分片，一个索引可以被复制零次(就是没有复制)或多次。一旦被复制，每个索引将会有些主分片(就是那些最原始不是被复制出来的分片)，还有一些复制分片(就是那些通过复制主分片得到的分片)。

主分片和复制分片的数量可以在索引被创建时指定。索引被创建后，你可以随时动态修改复制分片的数量，但是不能修改主分片的数量。

默认情况下，在Elasticsearch中的每个索引被分配5个主分片和一份拷贝，这意味着假设你的集群中至少有两个节点，你的索引将会有5个主分片和5个复制分片(每个主分片对应一个复制分片，5个复制分片组成一个完整拷贝)，总共每个索引有10个分片。

每个Elasticsearch分片是一个Lucene索引。在一个Lucene索引中有一个文档数量的最大值。截至LUCENE-5843，这个限制是2,147,483,519 (= Integer.MAX_VALUE - 128)个文档。你可以使用_cat/shards API监控分片大小。

现在熟悉了概念之后，让我们开始有趣的部分吧...

安装

Elasticsearch需要至少Java 8。明确的说，截至本文写作时，推荐使用Oracle JDK 1.8.0_131版本。Java的安装在不同的平台下是不一样的，所以在这里就不再详细介绍。你可以在[Oracle官网](http://www.oracle.com/technetwork/java/javase-downloads-1344955.html)找到官方推荐的安装文档。所以说，当你在安装Elasticsearch之前，请先通过以下命令检查你的Java版本(然后根据需要安装或升级)。

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

一旦Java准备就绪，然后我们就可以下载并运行Elasticsearch。我们可以从这个页面<http://www.elastic.co/downloads> 获取所有发行版本的二进制安装包。每一个版本都对应有zip和tar压缩包，还有deb和rpm安装包，还有Windows下用的msi安装包。

Linux tar包安装示例

为了简单，让我们使用tar包来安装。

使用如下命令下载Elasticsearch 6.1.1的tar包：

```
curl -L -O https://artifacts.elastic.co/...
```

使用如下命令解压：

```
tar -xvf elasticsearch-6.1.1.tar.gz
```

上述操作将会在你的当前目录下创建很多文件和文件夹。然后通过如下命令进入bin目录：

```
cd elasticsearch-6.1.1/bin
```

接下来我们就可以启动我们的单节点集群：

```
./elasticsearch
```

MacOS使用Homebrew安装

在macOS上，我们可以通过Homebrew来安装Elasticsearch：

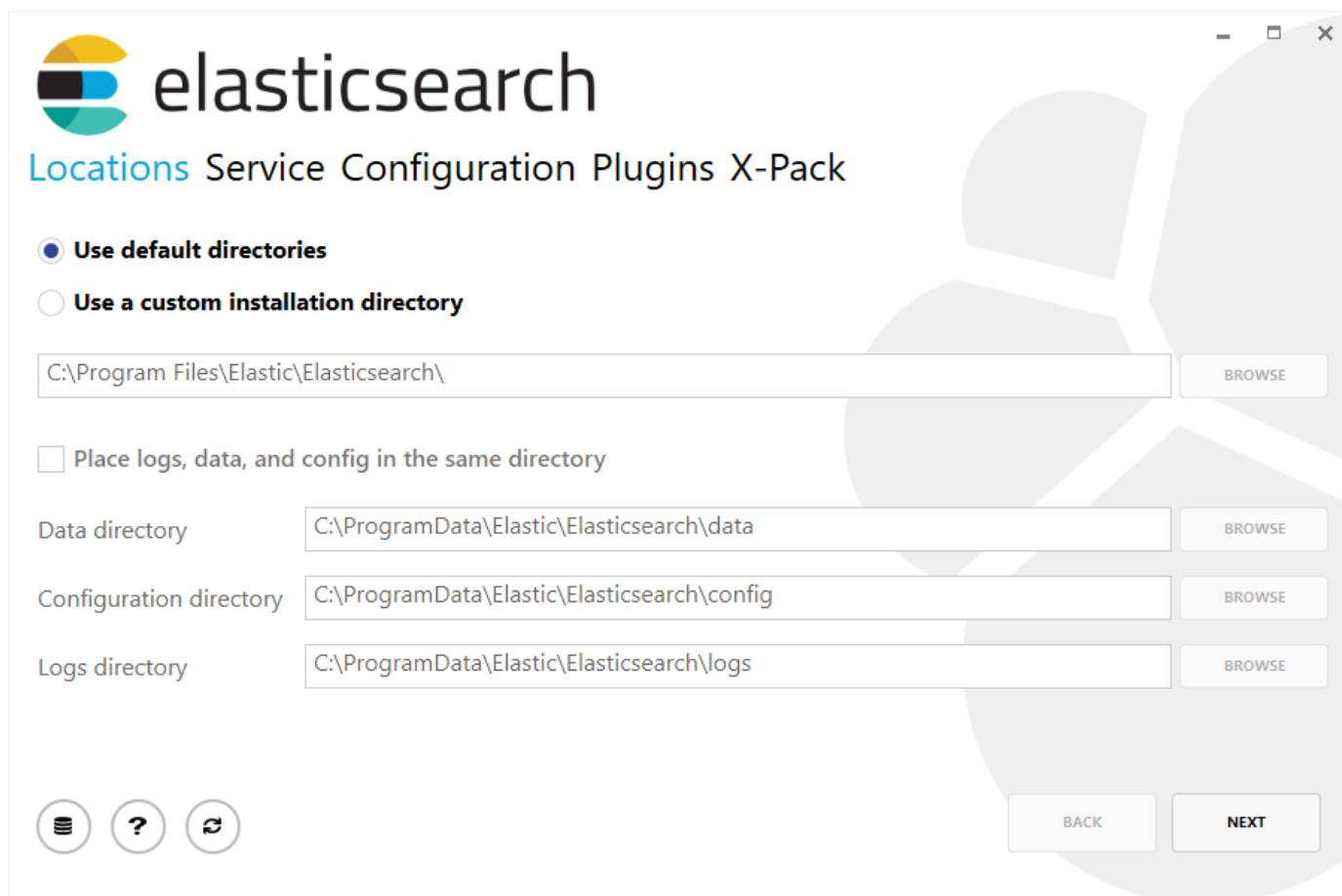
```
brew install elasticsearch
```

Windows上使用MSI安装

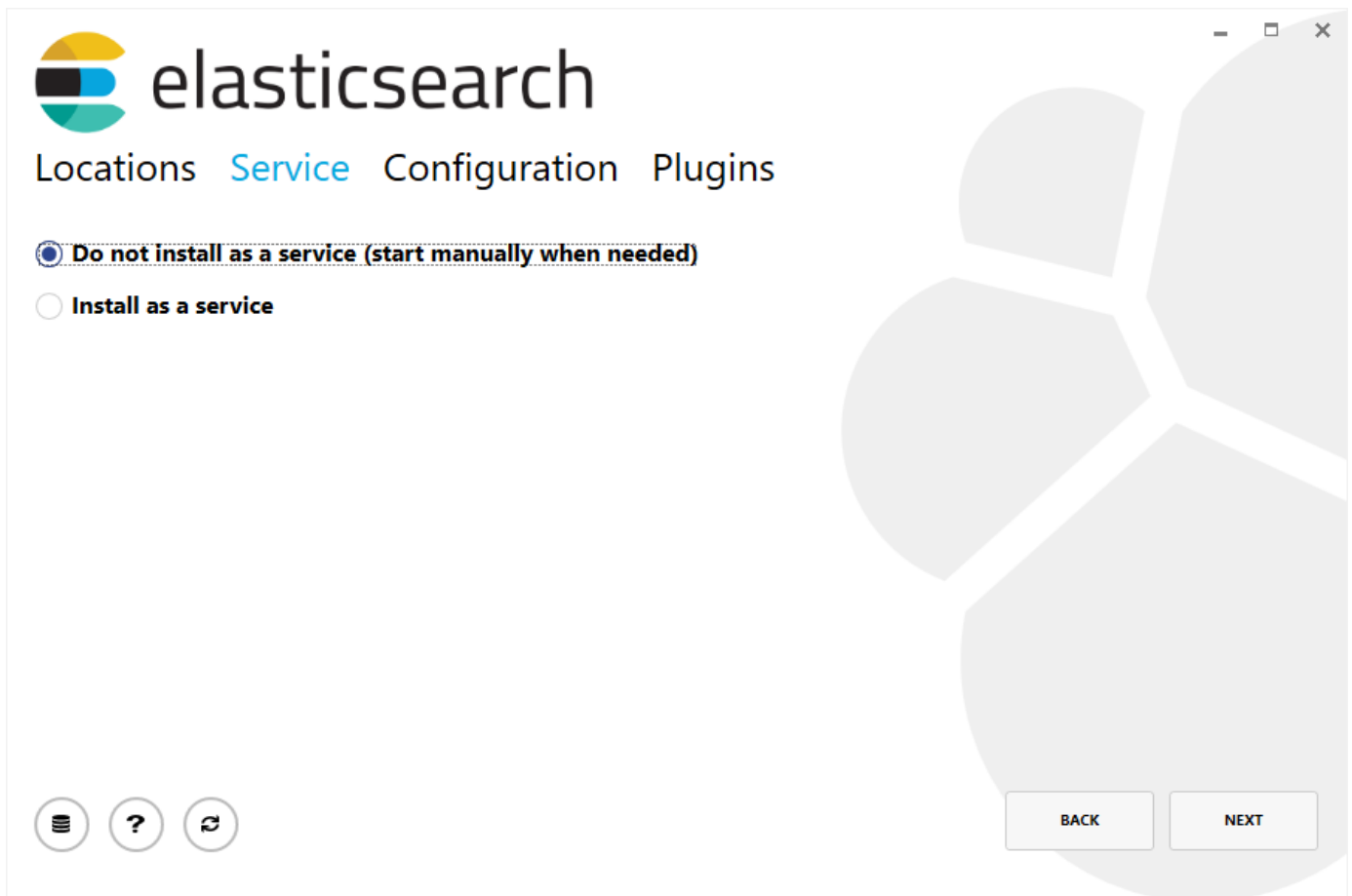
对于Windows用户，我们推荐使用[MSI安装包](#)进行安装。这个安装包使用图形用户界面来引导你进行安装。

首先，从这里<https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.1.1.msi>下载Elasticsearch 6.1.1的MSI安装包。

然后双击下载好的安装包文件启动图形化安装程序，在第一个界面，选择安装目录：



然后选择是否将Elasticsearch安装为一个系统服务，为了和用tar包安装示例保持一致，我们选择不安装为系统服务，根据自己需要手动启动：



The screenshot shows the 'Service' configuration window in the Elasticsearch installer. The Elasticsearch logo is at the top left. Below it are tabs for 'Locations', 'Service' (which is selected and highlighted in blue), 'Configuration', and 'Plugins'. There are two radio button options: 'Do not install as a service (start manually when needed)' which is selected, and 'Install as a service'. At the bottom left are three circular icons: a list, a question mark, and a refresh. At the bottom right are 'BACK' and 'NEXT' buttons.

elasticsearch

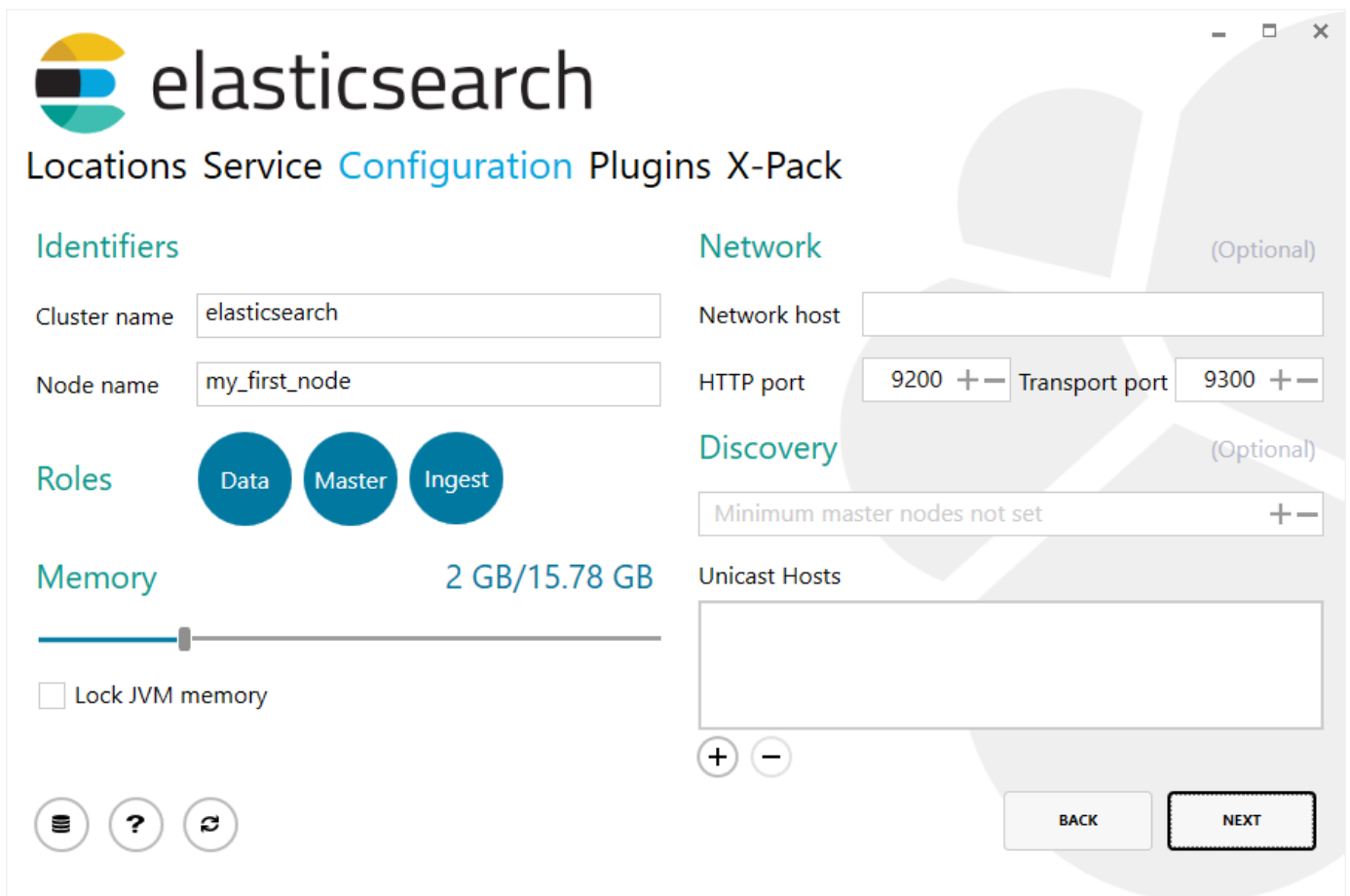
Locations **Service** Configuration Plugins

☒ Do not install as a service (start manually when needed)

☐ Install as a service

BACK NEXT

然后到了配置页面，这里就简单的使用默认的配置值：



The screenshot shows the 'Configuration' window in the Elasticsearch installer. The Elasticsearch logo is at the top left. Below it are tabs for 'Locations', 'Service', 'Configuration' (which is selected and highlighted in blue), 'Plugins', and 'X-Pack'. The window is divided into two main sections: 'Identifiers' on the left and 'Network' on the right. The 'Identifiers' section has input fields for 'Cluster name' (elasticsearch) and 'Node name' (my_first_node). Below these are three circular buttons for 'Roles': 'Data', 'Master', and 'Ingest'. The 'Memory' section shows a slider set to 2 GB/15.78 GB and a checkbox for 'Lock JVM memory'. The 'Network' section has input fields for 'Network host', 'HTTP port' (9200), and 'Transport port' (9300). Below these are 'Discovery' settings, including 'Minimum master nodes not set'. At the bottom right are 'Unicast Hosts' input fields. At the bottom left are three circular icons: a list, a question mark, and a refresh. At the bottom right are 'BACK' and 'NEXT' buttons.

elasticsearch

Locations Service **Configuration** Plugins X-Pack

Identifiers

Cluster name elasticsearch

Node name my_first_node

Roles

Data Master Ingest

Memory 2 GB/15.78 GB

☐ Lock JVM memory

Network (Optional)

Network host

HTTP port 9200 Transport port 9300

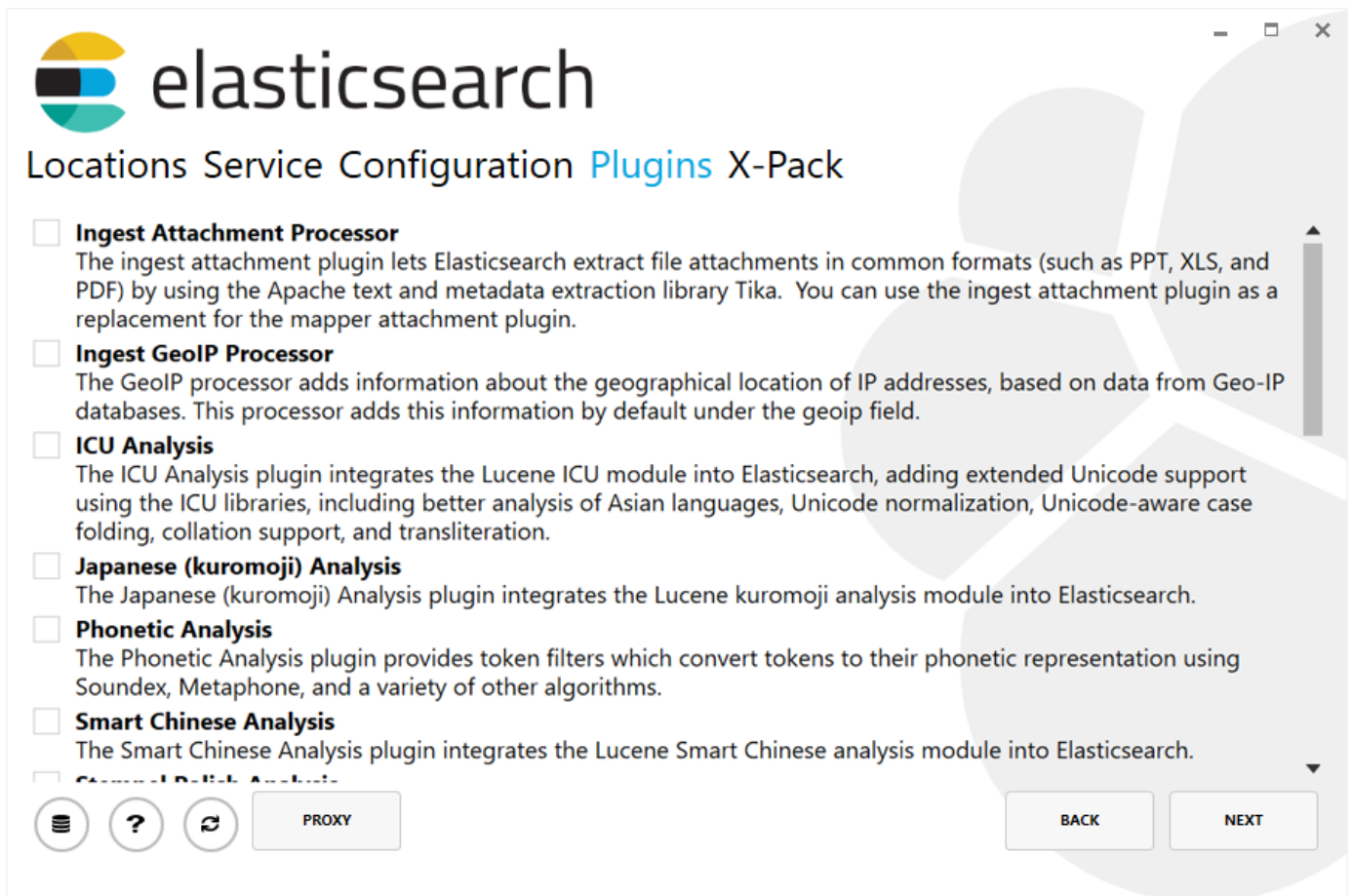
Discovery (Optional)

Minimum master nodes not set

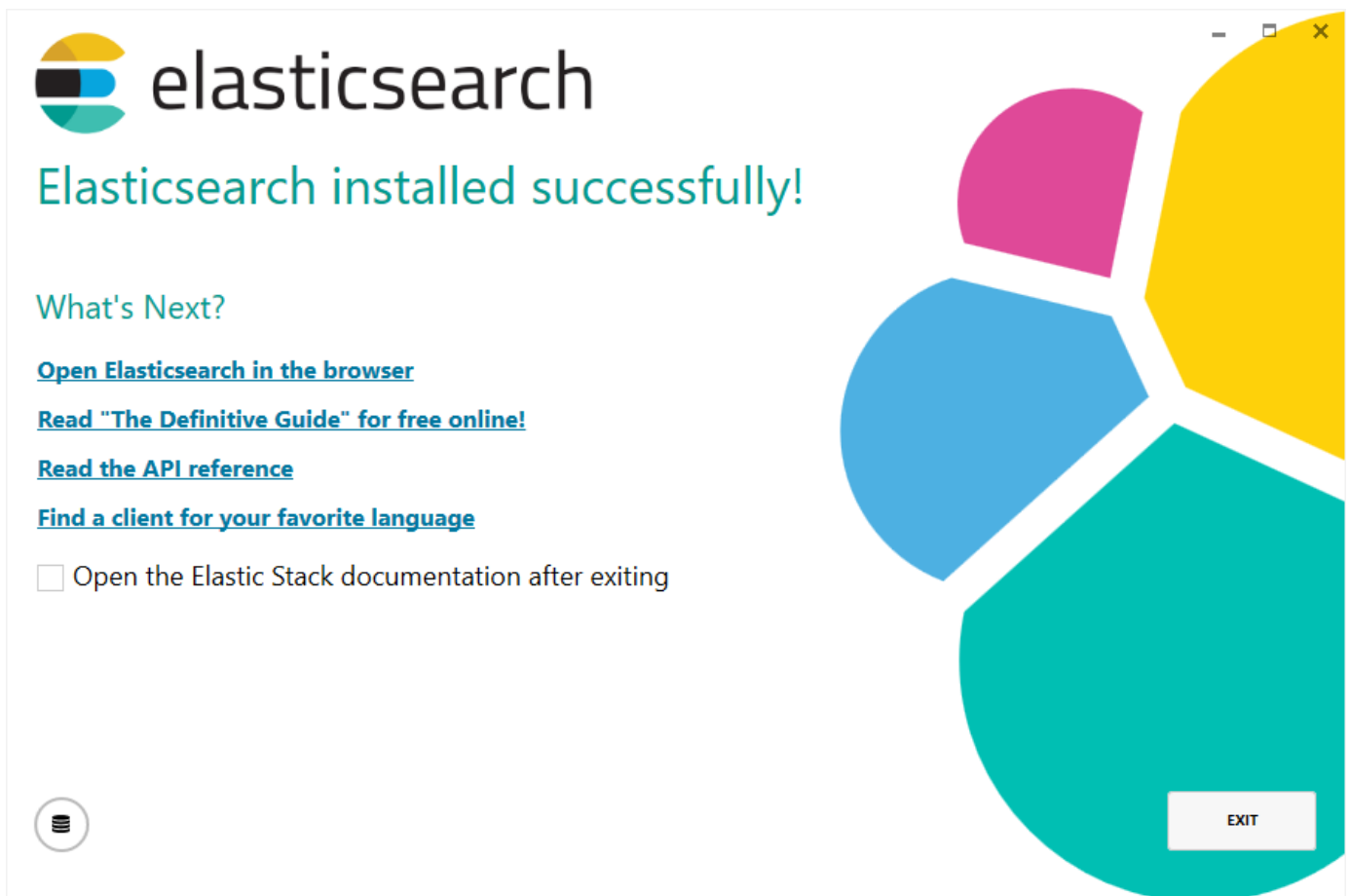
Unicast Hosts

BACK NEXT

进入插件安装页面，同样为了跟tar包安装示例保持一致，将所有的选择去掉，不安装任何插件：



然后点击安装按钮，Elasticsearch将会被安装：



默认情况下，Elasticsearch将会被安装在`%PROGRAMFILES%\Elastic\Elasticsearch`。进入这个目录并且切换到bin目录下：

使用命令提示符：

```
cd %PROGRAMFILES%ElasticElasticsearchbin
```

使用PowerShell:

```
cd $env:PROGRAMFILESElasticElasticsearchbin
```

接下来我们就可以启动我们的单节点集群了:

```
.elasticsearch.exe
```

成功运行节点

如果安装过程顺利的话, 你应该会看到如下的信息输出:

```
[2016-09-16T14:17:51,251][INFO ][o.e.n.Node                ] [] initializing ...
[2016-09-16T14:17:51,329][INFO ][o.e.e.NodeEnvironment ] [6-bjhw1] using [1] data paths, mounts
[[/ (dev/sda1)], net usable_space [317.7gb], net total_space [453.6gb], spins? [no], types [ext4]
[2016-09-16T14:17:51,330][INFO ][o.e.e.NodeEnvironment ] [6-bjhw1] heap size [1.9gb], compressed
ordinary object pointers [true]
[2016-09-16T14:17:51,333][INFO ][o.e.n.Node                ] [6-bjhw1] node name [6-bjhw1] derived
from node ID; set [node.name] to override
[2016-09-16T14:17:51,334][INFO ][o.e.n.Node                ] [6-bjhw1] version[6.1.1], 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-bjhw1] loaded module [aggs-matrix-
stats]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded module [ingest-common]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded module [lang-expression]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded module [lang-mustache]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded module [lang-painless]
[2016-09-16T14:17:51,967][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded module [percolator]
[2016-09-16T14:17:51,968][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded module [reindex]
[2016-09-16T14:17:51,968][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded module [transport-
netty3]
[2016-09-16T14:17:51,968][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded module [transport-
netty4]
[2016-09-16T14:17:51,968][INFO ][o.e.p.PluginsService     ] [6-bjhw1] loaded plugin [mapper-murmur3]
[2016-09-16T14:17:53,521][INFO ][o.e.n.Node                ] [6-bjhw1] initialized
[2016-09-16T14:17:53,521][INFO ][o.e.n.Node                ] [6-bjhw1] starting ...
[2016-09-16T14:17:53,671][INFO ][o.e.t.TransportService   ] [6-bjhw1] 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-bjhw1] 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-bjhw1] 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-bjhw1] recovered [0] indices into
cluster_state
[2016-09-16T14:17:56,748][INFO ][o.e.n.Node                ] [6-bjhw1] started
```


安装过程中我们没有关注过多的细节，可以看到我们名称叫做“6-bjhw1”（在你自己的示例中可能是别的名称）的节点已经启动并且选举了它自己作为单点集群的主节点(master)。不用担心此时的master是什么意思。这里我们主要关心的重点是我们启动了一个单节点的集群。

在前面我们提到过，我们可以覆盖集群或者是节点的名称。这个操作可以通过如下方式启动Elasticsearch完成。

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

还有就是通过上面启动时的输出信息我们可以看到，我们可以通过IP地址(192.168.8.112)和端口号(9200)来访问我们的节点。默认情况下，Elasticsearch使用9200端口提供REST API访问。这个端口可以根据需要自定义。

注意！为安全起见，Elasticsearch被设置为不允许使用root用户运行。所以运行之前首先需要创建新用户并赋予权限。

探究你的集群

REST API

既然我们的节点(集群)已经安装成功并且已经启动运行，那么下一步就是去了解如何去操作它。幸运的是，Elasticsearch提供了非常全面和强大的REST API，我们可以通过它去跟集群交互。通过API我们可以完成如下的功能：

- 检查集群，节点和索引的健康状况，状态和统计数据
- 管理集群，节点和索引的数据和原数据
- 执行CRUD(增删改查)操作，依靠索引进行搜索
- 执行高级搜索操作，比如分页，排序，过滤，脚本化，聚集等等

集群健康监控

让我们从一个简单的健康检查开始，通过这个我们可以了解我们集群的运行情况。我们将使用curl工具来做这个测试，当然你可以使用任何可以发送HTTP/REST请求的工具。让我们假设我们依然在之前已启动的Elasticsearch节点上并且打开了另一个shell窗口。

我们将使用 `cat API` 去检查集群健康状态。HTTP请求内容为：

```
GET /_cat/health?v
```

你可以通过点击[VIEW IN Console](#)在[Kibana Console](#)中运行命令，或者直接执行如下curl命令：

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

响应结果为：

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%							

我们可以看到我们的名称为“elasticsearch”的集群正在运行，状态标识为green。

无论何时查看集群健康状态，我们会得到green、yellow、red中的任何一个。

- Green - 一切运行正常(集群功能齐全)
- Yellow - 所有数据是可以获取的, 但是一些复制品还没有被分配(集群功能齐全)
- Red - 一些数据因为一些原因获取不到(集群部分功能不可用)

注意: 当一个集群处于red状态时, 它会通过可用的分片继续提供搜索服务, 但是当有未分配的分片时, 你需要尽快的修复它。

另外, 从上面的返回结果中我们可以看到, 当我们里面没有数据时, 总共有1个节点, 0个分片。注意当我们使用默认的集群名称(elasticsearch)并且当Elasticsearch默认使用单播网络发现在同一台机器上的其它节点时, 很可能你会在你电脑上不小心启动不止一个节点并且他们都加入了一个集群。在这种情况下, 你可能会从上面的返回结果中看到不止一个节点。

我们也可以通过如下请求获取集群中的节点列表:

Http请求体

```
GET /_cat/nodes?v
```

Curl命令

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

Kibana Console

返回结果为:

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

这里, 我们可以看到我们的一个节点名称叫做 “PB2SGZY”, 它是目前我们集群中的唯一的节点。

列出所有的索引

现在让我们来大概看一看我们的索引:

Http请求内容:

```
GET /_cat/indices?v
```

Curl命令

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

Kibana Console

得到的返回结果为:

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

这个返回结果只是一个表头, 简单说就是我们的集群中还没有任何索引。

创建一个索引

现在让我们创建一个索引, 名称为 “customer”, 然后再一次列出所有的索引:

Http请求内容:

Curl命令

Kibana Console

第一个命令使用`PUT`方法创建了一个名为“customer”的索引。我们简单的在请求后面追加`pretty`参数来使返回值以格式化过美观的JSON输出(如果返回值是JSON格式的话)。

然后它的返回结果为:

第一个命令:

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "customer"
}
```

第二个命令:

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								

第二个命令的返回结果告诉我们,我们现在有1个名称为“customer”的索引,并且有5个主分片和1个拷贝(默认情况),并且里面包含0个文档。

你可能也注意到,这个customer索引的健康状态是yellow,回忆我们之前讨论过的,yellow的意思是有一些拷贝还没有被分配。索引发生这种情况的原因是Elasticsearch默认为当前索引创建一个拷贝。但是当前我们只启动了一个节点,这个拷贝直到一段时间后有另一个节点加入集群之前,不会被分配(为了高可用,拷贝不会与索引分配到同一个节点上)。一旦拷贝在第二个节点上获得分配,这个索引的健康状态就会变成green。

索引和文档查询

现在让我们往customer索引中放点东西。如下请求将一个简单的顾客文档放入customer索引中,这个文档有一个ID为1:

Http请求内容:

```
PUT /customer/doc/1?pretty
{
  "name": "John Doe"
}
```

Curl命令

```
curl -XPUT 'localhost:9200/customer/doc/1?pretty&pretty' -H 'Content-Type: application/json' -d '{ "name": "John Doe" }'
```

Kibana Console

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_index_and_query_a
_document/1.json
```

返回结果为:

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

从上面我们可以看到, 一个新的顾客文档已经在customer索引中成功创建。同时这个文档有一个自己的id, 这个id就是我们在将文档加入索引时指定的。

这里有一个重要的注意点, 你不需要在将一个文档加入一个索引前明确的将这个索引预先创建好。在上面我们创建文档的例子中, 如果这个customer索引事先不存在, Elasticsearch会自动创建customer索引。

现在让我们获取刚刚加入索引的文档:

Http请求体:

```
GET /customer/doc/1?pretty
```

Curl命令

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

Kibana Console

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_index_and_query_a
_document/2.json
```

返回结果为:

```
{
  "_index" : "customer",
  "_type" : "doc",
```

```
{
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : { "name": "John Doe" }
}
```

这里没有什么不寻常的，除了一个属性`found`，这个`found`属性表示我们通过请求ID为1发现了一个文档，还有另一个属性`_source`，`_source`属性返回我们在上一步中加入索引的完整JSON文档内容。

删除一个索引

现在让我们删除刚刚创建的索引并且再次列出所有的索引：

Http请求内容：

```
DELETE /customer?pretty
GET /_cat/indices?v
```

Curl命令

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

Kibana Console

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_delete_an_index/1
.json
```

第一个命令的返回结果为：

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

第二个命令的返回结果为：

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

以上结果意味着我们的索引已经被删除，并且我们回到了刚开始集群中什么都没有的地方。

在我们继续前进之前，让我们来仔细看一下到目前为止学习的这些API命令：

```
PUT /customer
PUT /customer/doc/1
{
  "name": "John Doe"
}
```

```

}
GET /customer/doc/1
DELETE /customer

```

如果我们在学习上面的命令时非常仔细的话，我们一定会发现在Elasticsearch中访问数据的模式。这个模式可以总结为以下形式：

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

这种REST访问模式遍布所有的API命令，如果简单的记住它，你将会在掌握Elasticsearch的过程中有一个很好的开端。

如下是我在上述章节实际做的操作：

```

[root@bogon elasticsearch-6.1.1]# curl -XGET 'localhost:9200/_cat/health?v&pretty'
epoch      timestamp cluster      status node.total node.data shards pri relo init unassign
pending_tasks max_task_wait_time active_shards_percent
1514269983 14:33:03  elasticsearch green          1          1      0  0  0  0      0
0              -              100.0%
[root@bogon elasticsearch-6.1.1]# curl -XGET 'localhost:9200/_cat/health?v&pretty'
epoch      timestamp cluster      status node.total node.data shards pri relo init unassign
pending_tasks max_task_wait_time active_shards_percent
1514270109 14:35:09  elasticsearch green          1          1      0  0  0  0      0
0              -              100.0%
[root@bogon elasticsearch-6.1.1]# curl -XGET 'localhost:9200/_cat/nodes?v&pretty'
ip          heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
127.0.0.1      7          93    0    0.00   0.01    0.05 mdi      *     sEicoNR
[root@bogon elasticsearch-6.1.1]# curl -XGET 'localhost:9200/_cat/indices?v&pretty'
health status index uuid pri rep docs.count docs.deleted store.size pri.store.size
[root@bogon elasticsearch-6.1.1]# curl -XPUT 'localhost:9200/customer?pretty&pretty'
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "customer"
}
[root@bogon elasticsearch-6.1.1]# curl -XGET 'localhost:9200/_cat/indices?v&pretty'
health status index      uuid                                pri rep docs.count docs.deleted store.size
pri.store.size
yellow open    customer Azxs-a4FqGKgAj0zdWXxQ      5   1          0          0      1.1kb
1.1kb
[root@bogon elasticsearch-6.1.1]# curl -XPUT 'localhost:9200/customer/doc/1?pretty&pretty' -H
'Content-Type: application/json' -d '{"name": "John Doe"}'
{
  "_index" : "customer",
  "_type" : "doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,

```

```
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
[root@bogon elasticsearch-6.1.1]# curl -XGET 'localhost:9200/customer/doc/1?pretty&pretty'
{
  "_index" : "customer",
  "_type" : "doc",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : {
    "name" : "John Doe"
  }
}
[root@bogon elasticsearch-6.1.1]# curl -XDELETE 'localhost:9200/customer?pretty&pretty'
{
  "acknowledged" : true
}
[root@bogon elasticsearch-6.1.1]# curl -XGET 'localhost:9200/_cat/indices?v&pretty'
health status index uuid pri rep docs.count docs.deleted store.size pri.store.size
[root@bogon elasticsearch-6.1.1]#
```

修改你的数据

Elasticsearch提供了近实时的数据操作和搜索能力。默认情况下，从你开始索引/更新/删除你的数据到出现搜索结果的时间会有一秒的延时(刷新间隔)。这个与其它的SQL数据库平台在一个事务完成后立即获取到数据这一点上有很大的优势。

将文档放入索引/替换索引中的文档

我们之前已经看过如何将一个文档放入索引中，让我们再次回忆一下那个命令：

Http请求：

```
PUT /customer/doc/1?pretty
{
  "name": "John Doe"
}
```

curl命令：

```
curl -XPUT 'localhost:9200/customer/doc/1?pretty&pretty' -H 'Content-Type: application/json' -d'
{
  "name": "John Doe"
}
```

```
}  
,
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?  
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_modifying_your_data/1.json
```

再次补充说明一下，上面的请求将会将一个ID为1的文档加入customer索引。如果我们再次执行上面的请求，以相同的文档内容或者是不同的，Elasticsearch将会用这个新文档替换之前的文档(就是以相同的ID重新加入索引)。

Http请求内容:

```
PUT /customer/doc/1?pretty  
{  
  "name": "Jane Doe"  
}
```

curl命令:

```
curl -XPUT 'localhost:9200/customer/doc/1?pretty&pretty' -H 'Content-Type: application/json' -d'  
{  
  "name": "Jane Doe"  
}'
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?  
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_modifying_your_data/2.json
```

上述操作将ID为1的文档的name属性从“John Doe”改成了“Jane Doe”。设想另一种场景，我们使用一个不同的ID，这样的话将会创建一个新的文档，而之前的文档还是保持原样。

Http请求:

```
PUT /customer/doc/2?pretty  
{  
  "name": "Jane Doe"  
}
```

curl命令:

```
curl -XPUT 'localhost:9200/customer/doc/2?pretty&pretty' -H 'Content-Type: application/json' -d'  
{  
  "name": "Jane Doe"  
}'
```



```
}  
,
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?  
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_modifying_your_data/3.json
```

上述操作将一个ID为2的文档加入索引。

当将文档加入索引时，ID部分并不是必须的。如果没有指定，Elasticsearch将会生产一个随机的ID，然后使用它去索引文档。实际Elasticsearch生成的ID(或者是我们明确指定的)将会在API调用成功后返回。

如下这个例子演示如何使用隐式的ID将一个文档加入索引：

Http请求：

```
POST /customer/doc?pretty  
{  
  "name": "Jane Doe"  
}
```

curl命令：

```
curl -XPOST 'localhost:9200/customer/doc?pretty&pretty' -H 'Content-Type: application/json' -d'  
{  
  "name": "Jane Doe"  
}  
,
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?  
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_modifying_your_data/4.json
```

注意在上面的例子中，当我们没有明确指定ID的时候，我们需要使用POST方法代替PUT来发送请求。

更新文档

除了能够新增和替换文档，我们也可以更新文档。注意虽然Elasticsearch在底层并没有真正更新文档，而是当我们更新文档时，Elasticsearch首先去删除旧的文档，然后加入新的文档。

如下的例子演示如何去更新我们的之前ID为1的文档，在这里将name属性改为“Jane Doe”：

Http请求内容：

```
POST /customer/doc/1/_update?pretty  
{
```

```
{
  "doc": { "name": "Jane Doe" }
}
```

curl命令:

```
curl -XPOST 'localhost:9200/customer/doc/1/_update?pretty&pretty' -H 'Content-Type: application/json' -d'
{
  "doc": { "name": "Jane Doe" }
},
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_updating_documents/1.json
```

如下示例演示如何更新我们之前ID为1的文档，修改name属性为 "Jane Doe" ，并同时添加新的age属性：

Http请求内容:

```
POST /customer/doc/1/_update?pretty
{
  "doc": { "name": "Jane Doe", "age": 20 }
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_updating_documents/2.json
```

更新操作也可以使用简单的脚本来执行。如下的示例使用一个脚本将age增加了5:

Http请求:

```
POST /customer/doc/1/_update?pretty
{
```

```
"script" : "ctx._source.age += 5"
}
```

curl命令:

```
curl -XPOST 'localhost:9200/customer/doc/1/_update?pretty&pretty' -H 'Content-Type: application/json' -d'
{
  "script" : "ctx._source.age += 5"
},
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_updating_documents/3.json
```

在上面的示例中, `ctx._source`指代的是当前需要被更新的source文档。

Elasticsearch提供了一次更新多个文档的功能, 通过使用查询条件(比如SQL的UPDATE-WHERE语句)。详情查看[docs-update-by-query API](#)

删除文档

删除一个文档操作相当的直截了当。如下的示例演示了如何删除我们之前ID为2的文档:

Http请求内容:

```
DELETE /customer/doc/2?pretty
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_deleting_documents/1.json
```

查看[delete_by_query API](#)去删除匹配特定条件的所有的文档。有一个值得注意的地方是, 直接删除整个索引比通过Query API删除索引中的所有文档更高效。

批处理

除了在单个文档上执行索引, 更新和删除操作外, Elasticsearch还提供了批操作的功能, 通过使用 [bulk API](#)完成。这个功能非常重要, 因为它提供了一种非常高效的机制去通过更少的网络切换尽可能快的执行多个操作。

作为一个快速入门示例, 如下请求在一个批操作中创建了两个文档:

Http请求内容:

```
POST /customer/doc/_bulk?pretty
{"index":{"_id":"1"}}
{"name": "John Doe" }
{"index":{"_id":"2"}}
{"name": "Jane Doe" }
```

curl命令:

```
curl -XPOST 'localhost:9200/customer/doc/_bulk?pretty&pretty' -H 'Content-Type: application/json' -d'
{"index":{"_id":"1"}}
{"name": "John Doe" }
{"index":{"_id":"2"}}
{"name": "Jane Doe" }
,
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_batch_processing/
1.json
```

如下的示例在一个批操作中首先更新ID为1的文档，然后删除ID为2的文档:

Http请求内容:

```
POST /customer/doc/_bulk?pretty
{"update":{"_id":"1"}}
{"doc": { "name": "John Doe becomes Jane Doe" } }
{"delete":{"_id":"2"}}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_batch_processing/
2.json
```

注意上面的删除操作，删除时只需要指定被删除的文档的ID即可，不需要指定对应的source内容。

批处理API不会因为单条指令失败而全部失败。如果里面有单条指令因为一些原因失败了，那么整个批处理还会继续执行它后面剩余的指令。当批处理API返回时，它会提供每条指令的执行状态(以发送时的顺序)，以便你可以检查一个特定的指令是否失败。

探究你的数据

样本数据集

既然我们已经了解了基础知识，让我们来尝试操作一些更真实的数据集。我已经预先准备好了一些虚拟的顾客银行账户信息JSON文档样本。每一个文档都有如下的机构：

```
{
  "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"
}
```

出于好奇，这些数据是在www.json-generator.com生成的。所有请忽略这些数据值的实际意义，因为它们都是随机生成的。

加载样本数据集

你可以从[这里](#)下载样本数据集(accounts.json)。把它放到我们当前的目录下，然后使用如下的命令把它加载到我们得集群中：

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

返回结果为：

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	bank	17sSYV2cQXmu6_4rJWVIww	5	1	1000	0	128.6kb	128.6kb

这意味着我们刚刚成功将1000个文档批量放入bank索引中(在account类型下)。

搜索API

现在，让我们从一些简单的搜索指令开始。执行搜索有两种基础的方式，一种是在请求的URL中加入参数来实现，另一种方式是将请求内容放到请求体中。使用请求体可以让你的JSON数据以一种更加可读和更加富有展现力的方式发送。我们将会在一开始演示一次使用请求URI的方式，然后在本教程剩余的部分，我们将统一使用请求体的方式发送。

REST API可以使用 `_search` 端点来实现搜索。如下的示例将返回bank索引的所有的文档：

HTTP请求：

```
GET /bank/_search?q=*&sort=account_number:asc&pretty
```

Curl命令：

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

Kibana Console：

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_the_search_api/1.
json
```

让我们首先来详细分析一下这个搜索请求。这个请求在bank索引中进行搜索(使用 `_search` 端点)，然后 `q=*` 参数命令Elasticsearch匹配索引中的全部文档。`sort=account_number:asc` 参数表示按 `account_number` 属性升序排列返回的结果。`pretty` 参数之前已经提到过，就是将返回结果以美观的格式返回。

返回结果为(展示部分)：

```
{
  "took" : 63,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "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":"Mckenzie","age":29,"gender":"F",
      "address":"244 Columbus
      Place","employer":"Euron","email":"bradshawmckenzie@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","age":32,"gender":"M","address":"880 Holmes Lane","employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state":"IL"}
  }, ...
]
}
}

```

关于返回结果，我们看到了如下的部分：

- `took` - Elasticsearch执行此次搜索所用的时间(单位：毫秒)
- `timed_out` - 告诉我们此次搜索是否超时
- `shards` - 告诉我们搜索了多少分片，还有搜索成功和搜索失败的分片数量
- `hits` - 搜索结果
- `hits.total` - 符合搜索条件的文档数量
- `hits.hits` - 实际返回的搜索结果对象数组(默认只返回前10条)
- `hits.sort` - 返回结果的排序字段值(如果是按score进行排序，则没有)
- `hits._score` 和 `max_score` - 目前先忽略这两个字段

如下是相同效果的另一种将数据放入请求体的方式：

HTTP请求内容：

```

GET /bank/_search
{
  "query": { "match_all": {} },
  "sort": [
    { "account_number": "asc" }
  ]
}

```

curl命令：

```

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

```

Kibana Console：

```

http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_the_search_api/2.
json

```


这里的不同点在于我们使用一个JSON格式的请求体代替了在URI中的 `q=*` 参数。我们将在下一节详细讨论这种JSON格式的查询方式。

一旦你得到了返回结果，Elasticsearch就完全执行结束，不会保持任何的服务器资源或者往你的结果里加入开放的游标，理解这一点是非常重要的。这同很多其他的平台比如SQL数据库的一些特性形成了鲜明的对比，比如在SQL数据库中你可能在查询时，会首先得到查询结果的一部分，然后你需要通过一些有状态的服务端游标不断地去请求服务端来取得剩余的查询结果。

介绍查询语言

Elasticsearch提供了一种JSON格式的领域特定语言，你可以使用它来执行查询。这个通常叫做[Query DSL](#)。这门查询语言相当的全面以至于你第一次看到它时会被它吓住，不过学习它最好的方式就是从一些简单的示例程序开始。

回到我们上个例子，我们执行了这个查询：

HTTP请求内容：

```
GET /bank/_search
{
  "query": { "match_all": {} }
}
```

curl命令：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": { "match_all": {} }
},
'
```

Kibana Console：

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_introducing_the_query_language/1.json
```

分析以上查询，`query` 部分告诉我们我们的查询定义是什么，`match_all` 部分简单指定了我们想去执行的查询类型，意思就是在索引中搜索所有的文档。

除了`query`参数，我们还可以通过其他的参数影响搜索结果。在上一节的示例中我们使用了`sort`来指定搜索结果的顺序，这里我们指定`size`来指定返回的结果数量：

HTTP请求内容：

```
GET /bank/_search
{
  "query": { "match_all": {} },
  "size": 1
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_introducing_the_q
uery_language/2.json
```

注意如果size没有指定，它默认为10。

如下的示例使用match_all并返回了11到20的文档:

HTTP请求内容:

```
GET /bank/_search
{
  "query": { "match_all": {} },
  "from": 10,
  "size": 10
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_introducing_the_q
uery_language/3.json
```

from 参数(从0开始)指定了从哪个文档索引开始, size 参数指定了从from指定的索引开始返回多少个文档。这个特性在实现分页搜索时很有用。注意如果from参数没有指定, 它默认为0。

如下示例使用match_all并且按账户的balance值进行倒序排列后返回前10条文档:

HTTP请求内容:

```
GET /bank/_search
{
  "query": { "match_all": {} },
  "sort": { "balance": { "order": "desc" } }
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_introducing_the_q
uery_language/4.json
```

执行搜索

既然我们已经了解了一些基础的搜索参数，那就让我们来深入学习一下Query DSL吧。首先，我们来关注一下返回的文档属性。默认情况下，文档会作为搜索结果的一部分返回所有的属性值。这个文档的JSON内容被称为source(返回结果中的hits的_source属性值)。如果我们不需要返回所有的source文档属性，我们可以在请求体中加入我们需要返回的属性名。

如下的示例演示了如何返回两个属性，`account_number` 和 `balance` (在`source`中):

HTTP请求内容:

```
GET /bank/_search
{
  "query": { "match_all": {} },
  "_source": ["account_number", "balance"]
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/1.json
```

注意上面的例子仅仅只是减少了 `source` 里的属性。它仍然会返回 `source` 属性，只不过 `source` 属性中之包含 `account_number` 和 `balance` 两个属性。

如果你之前学过SQL，上面的示例有点像SQL中的 `SELECT` `FROM` 中指定返回的字段列表。

现在，让我们的视线转到查询部分。之前我们已经看到如何使用 `match_all` 来匹配所有的文档。现在让我们介绍一个新的查询叫做 `match` 查询，它可以被认为是基本的属性搜索查询(就是通过特定的一个或多个属性来搜索)。

如下的示例返回 `account_number` 为20的文档：

HTTP请求内容：

```
GET /bank/_search
{
  "query": { "match": { "account_number": 20 } }
}
```

curl命令：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": { "match": { "account_number": 20 } }
},
'
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/2.json
```

如下示例返回所有的 `address` 字段中包含 “mill” 这个单词的账户文档：

HTTP请求内容：

```
GET /bank/_search
{
  "query": { "match": { "address": "mill" } }
}
```

curl命令：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": { "match": { "address": "mill" } }
},
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/3.json
```

如下示例返回所有的address字段中包含 “mill” 或者是 “lane” 的账户文档:

HTTP请求内容:

```
GET /bank/_search
{
  "query": { "match": { "address": "mill lane" } }
}
```

curl命令:

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": { "match": { "address": "mill lane" } }
},
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/4.json
```

如下示例是`match`的一种变体(`match_phrase`), 这个将返回所有address中包含 “mill lane” 这个短语的账户文档:

HTTP请求内容:

```
GET /bank/_search
{
  "query": { "match_phrase": { "address": "mill lane" } }
}
```

curl命令:

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
```

```
{
  "query": { "match_phrase": { "address": "mill lane" } }
},
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/5.json
```

现在让我们介绍 [bool 查询](#)。bool 查询允许我们使用布尔逻辑将小的查询组成大的查询。

如下的示例组合两个 [match](#) 查询并且返回所有address属性中包含 “mill” 和 “lane” 的账户文档:

HTTP请求内容:

```
GET /bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/6.json
```

在上述示例中，`bool must` 子句指定了所有匹配文档必须满足的条件。

相比之下，如下的示例组合两个`match`查询并且返回所有`address`属性中包含 “mill” 或 “lane” 的账户文档：

HTTP请求内容：

```
GET /bank/_search
{
  "query": {
    "bool": {
      "should": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}
```

curl命令：

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

Kibana Console：

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/7.json
```

在上述的例子中，`bool should` 子句指定了匹配文档只要满足其中的任何一个条件即可匹配。

如下示例组合两个`match`查询并且返回所有`address`属性中**既不**包含 “mill” **也不**包含 “lane” 的账户文档：

HTTP请求内容：

```
GET /bank/_search
{
  "query": {
    "bool": {
      "must_not": [
```



```
{ "match": { "address": "mill" } },
{ "match": { "address": "lane" } }
]
}
}
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/8.json
```

在上述例子中, `bool must_not` 子句指定了其中的任何一个条件都不满足时即可匹配。

我们可以在一个 `bool` 查询中同时指定 `must`, `should` 和 `must_not` 子句。此外, 我们也可以在一個 `bool` 子句中组合另一个 `bool` 来模拟任何复杂的多重布尔逻辑。

如下的示例返回所有age属性为40, 并且state属性不为ID的账户文档:

HTTP请求内容:

```
GET /bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "ID" } }
      ]
    }
  }
}
```

curl命令:

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "ID" } }
      ]
    }
  }
},
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_searches/9.json
```

执行过滤

在之前的章节中,我们跳过了一个叫做文档得分(在搜索结果中的`score`属性)的小细节。这个得分是一个数值,它是一个相对量,用来衡量搜索结果跟我们指定的关键字的相关程度。分数越高,说明这个文档的相关性越大,分数越低,说明这个文档的相关性越小。

但是一些查询结果并不总是需要产生得分,尤其是当他们仅仅被用来过滤文档集的时候。Elasticsearch会检测这种情况并自动优化查询以免计算无用的分数。

我们在前面章节介绍的`bool 查询`也支持 `filter` 子句,它允许我们可以在不改变得分计算逻辑的情况下限制其他子句匹配的查询结果。为了示例说明,让我们介绍一下`range 查询`,它允许我们通过一个值区间来过滤文档。这个通常用在数值和日期过滤上。

如下的示例使用bool查询返回所有余额在20000到30000之间的账户(包含边界)。换句话说,我们想查询账户余额大于等于20000并且小于等于30000的用户。

HTTP请求内容:

```
GET /bank/_search
{
  "query": {
    "bool": {
      "must": { "match_all": {} },
      "filter": {
        "range": {
```

```
        "balance": {
          "gte": 20000,
          "lte": 30000
        }
      }
    }
  }
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_filters
/1.json
```

仔细分析一下上面的例子，bool查询在查询部分使用`match_all`，在过滤部分使用`range`。我们可以使用任何的查询来代替查询部分和过滤部分。在上面的例子中，range查询让结果更加合乎情理，因为文档在这个区间中一定是符合的，就是说，没有比这些相关性更大的了。

除了`match_all`，`match`，`bool`，和`range`查询之外，还有很多其他的查询类型，在这里我们就不一一介绍了。当我们对这些基础的理解了之后，再去学习和使用其他的查询类型应该是不会太难了。

执行聚合

聚合提供了功能可以分组并统计你的数据。理解聚合最简单的方式就是可以把它粗略的看做SQL的GROUP BY操作和SQL的聚合函数。在Elasticsearch中，你可以在执行搜索后在一个返回结果中同时返回搜索结果和聚合结果。你可以使用简洁的API执行搜索和多个聚合操作，并且可以一次拿到所有的结果，避免网络切换，就此而言，这是一个非常强大和高效功能。

作为开始，如下的例子将账户按state进行分组，然后按count降序(默认)返回前10组(默认)states。

HTTP请求内容：

```
GET /bank/_search
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state.keyword"
      }
    }
  }
}
```

curl命令：

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

Kibana Console：

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_aggrega
tions/1.json
```

上面的聚合的例子跟如下的SQL类似：

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

返回结果为(展示部分)：

```
{
  "took": 29,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
```

```

    "skipped" : 0,
    "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
      } ]
    }
  }
}

```

我们可以看到有27个账户在ID(爱达荷州)，然后27个在TX(得克萨斯州)，还有25个在AL(亚拉巴马州)，等等。

注意我们设置了size=0来不显示hits搜索结果，因为我们这里只关心聚合结果。

如下示例我们在上一个聚合的基础上构建，这个示例计算每个state分组的平均账户余额(还是使用默认按count倒序返回前10个):

HTTP请求内容:

```
GET /bank/_search
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state.keyword"
      },
      "aggs": {
        "average_balance": {
          "avg": {
            "field": "balance"
          }
        }
      }
    }
  }
}
```

curl命令:

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

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_aggrega
tions/2.json
```

注意我们是怎么嵌套`average_balance`聚合到`group_by_state`聚合中的。

这是一个适用于所有聚合操作的通用模式。你可以任意嵌套聚合，从你的数据中提取你需要的主题汇总。

如下例子依然是在之前的聚合上构建，我们现在来按平均余额倒序排列：

HTTP请求内容：

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

curl命令：

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state.keyword",
        "order": {
          "average_balance": "desc"
        }
      },
    },
    "aggs": {
      "average_balance": {
        "avg": {
```



```
        "field": "balance"
      }
    }
  }
}
},
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_aggrega
tions/3.json
```

如下示例演示我们如何按年龄区间分组(20-29, 30-39, 40-49), 然后按性别, 最后获取每个年龄区间, 每个性别的平均账户余额:

HTTP请求内容:

```
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"
      }
    }
  }
}
}
```

curl命令:

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'
{
  "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"
            }
          }
        }
      }
    }
  }
}
```

```
}  
,
```

Kibana Console:

```
http://localhost:5601/app/kibana#/dev_tools/console?  
load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/_executing_aggrega  
tions/4.json
```

还有很多其它的聚合功能在这里我们就不去详细介绍了。如果你想了解更多，可以参考[聚合参考手册](#)。

结论

Elasticsearch是一个既简单又复杂的产品。我们到目前为止已经学习了基础的知识，知道了它是什么，它内部的实现原理，以及如何使用REST API去操作它。希望此教程能帮助你理解Elasticsearch以及更重要的东西，鼓励你去实践它剩余的更多的特性！

-

