简介

Elasticsearch 是一个分布式、可扩展、实时的搜索与数据分析引擎。能从项目一开始就赋予搜索，分析和探索的能力。

Elasticsearch 不仅仅只是全文搜索，我们还将介绍结构化搜索、数据分析、复杂的语言处理、地理位置和对象间关联关系等

Elasticsearch 也是使用 Java 编写的，它的内部使用 Lucene 做索引与搜索。

Elasticsearch 不仅仅是 Lucene，并且也不仅仅只是一个全文搜索引擎。 它可以被下面这样准确的形容：

* 一个分布式的实时文档存储，*每个字段* 可以被索引与搜索
* 一个分布式实时分析搜索引擎
* 能胜任上百个服务节点的扩展，并支持 PB 级别的结构化或者非结构化数据

安装 elasticSearch

* 1. 安装java环境。
  2. 下载elasticSearch
  3. 启动服务

和elasticSearch交互

节点客户端（Node client）

节点客户端作为一个非数据节点加入到本地集群中。换句话说，它本身不保存任何数据，但是它知道数据在集群中的哪个节点中，并且可以把请求转发到正确的节点。

传输客户端（Transport client）

轻量级的传输客户端可以将请求发送到远程集群。它本身不加入集群，但是它可以将请求转发到集群中的一个节点上。

两个 Java 客户端都是通过 *9300* 端口并使用本地 Elasticsearch *传输* 协议和集群交互。集群中的节点通过端口 9300 彼此通信。如果这个端口没有打开，节点将无法形成一个集群。

Java 客户端作为节点必须和 Elasticsearch 有相同的 主要 版本；否则，它们之间将无法互相理解。

RESTful API with JSON over HTTP[编辑](https://github.com/elasticsearch-cn/elasticsearch-definitive-guide/edit/cn/010_Intro/15_API.asciidoc)

所有其他语言可以使用 RESTful API 通过端口 *9200* 和 Elasticsearch 进行通信，你可以用你最喜爱的 web 客户端访问 Elasticsearch 。事实上，正如你所看到的，你甚至可以使用 curl 命令来和 Elasticsearch 交互。

Elasticsearch 为以下语言提供了官方客户端 --Groovy、JavaScript、.NET、 PHP、 Perl、 Python 和 Ruby--还有很多社区提供的客户端和插件，所有这些都可以在 [Elasticsearch Clients](https://www.elastic.co/guide/en/elasticsearch/client/index.html) 中找到。

一个 Elasticsearch 请求和任何 HTTP 请求一样由若干相同的部件组成：

curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY\_STRING>' -d '<BODY>'

被 < > 标记的部件：

|  |  |
| --- | --- |
| VERB | 适当的 HTTP *方法* 或 *谓词* : GET`、 `POST`、 `PUT`、 `HEAD 或者 `DELETE`。 |
| PROTOCOL | http 或者 https`（如果你在 Elasticsearch 前面有一个 `https 代理） |
| HOST | Elasticsearch 集群中任意节点的主机名，或者用 localhost 代表本地机器上的节点。 |
| PORT | 运行 Elasticsearch HTTP 服务的端口号，默认是 9200 。 |
| PATH | API 的终端路径（例如 \_count 将返回集群中文档数量）。Path 可能包含多个组件，例如：\_cluster/stats 和 \_nodes/stats/jvm 。 |
| QUERY\_STRING | 任意可选的查询字符串参数 (例如 ?pretty 将格式化地输出 JSON 返回值，使其更容易阅读) |
| BODY | 一个 JSON 格式的请求体 (如果请求需要的话) |

面向文档

Elasticsearch 是 面向文档 的，意味着它存储整个对象或文档。lasticsearch 不仅存储文档，而且索引每个文档的内容使之可以被检索你 对文档进行索引、检索、排序和过滤--而不是对行列数据。这是一种完全不同的思考数据的方式，也是 Elasticsearch 能支持复杂全文检索的原因。

索引：对应关系型数据库的的一个数据库。

类型：对应关系型数据库的一张表。

文档：对应着关系型数据库的一条数据 记录。

索引雇员文档

第一个业务需求就是存储雇员数据。 这将会以 **雇员文档**的形式存储：一个文档代表一个雇员。存储数据到 Elasticsearch 的行为叫做 **索引**，但在索引一个文档之前，需要确定将文档存储在哪里。

|  |
| --- |
| 索引（名词）：  如前所述，一个 索引 类似于传统关系数据库中的一个 数据库 ，是一个存储关系型文档的地方。索引 (index) 的复数词为 indices 或 indexes 。  索引（动词）：  索引一个文档 就是存储一个文档到一个 索引 （名词）中以便它可以被检索和查询到。这非常类似于 SQL 语句中的 INSERT 关键词，除了文档已存在时新文档会替换旧文档情况之外。  倒排索引：  关系型数据库通过增加一个 索引 比如一个 B树（B-tree）索引 到指定的列上，以便提升数据检索速度。Elasticsearch 和 Lucene 使用了一个叫做 倒排索引 的结构来达到相同的目的。 |

一个 Elasticsearch 集群可以 包含多个 **索引** ，相应的每个索引可以包含多个 **类型** 。 这些不同的类型存储着多个 **文档** ，每个文档又有 多个 **属性** 。

向ElasticSearch中创建一个索引:

|  |
| --- |
| curl -H "Content-Type:application/json" -XPUT 'http://127.0.0.1:9200/megacorp/employee/1' -d '{ "first\_name":"john","last\_name":"Smith","age":25,"about":" I love to go rock climbing","interests":["spots","music"]} |

注意路径包含了三部分的信息：

Megacorp: 索引名称

Employee：类型名称。

1 ：employee的ID。

检索文档：

执行一个HTTP请求并指定文档的地址---索引库，类型和ID

|  |
| --- |
| curl -XGET 'http://127.0.0.1:9200/megacorp/employee/1' |

返回的数据：

|  |
| --- |
| {"\_index":"megacorp","\_type":"employee","\_id":"1","\_version":1,"found":true,"\_source":  { "first\_name":"john","last\_name":"Smith","age":25,"about":" I love to go rock climbing","interests":["spots","music"]} |

使用Delete命令来删除文档，

使用Head命令来检查文档是否存在

如果想更新已经存在的文档，只需要再次PUT

轻量搜索：、

1. 请求搜索所有的雇员

|  |
| --- |
| curl -XGET 'http://127.0.0.1:9200/megacorp/employee/\_search' |

返回的数据：

|  |
| --- |
| {"took":4,"timed\_out":false,"\_shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{"total":3,"max\_score":1.0,"hits":[{"\_index":"megacorp","\_type":"employee","\_id":"2","\_score":1.0,"\_source":  { "first\_name":"wang","last\_name":"su","age":23,"about":"I love to go rock climbing","interests":["spots","music"]}  },{"\_index":"megacorp","\_type":"employee","\_id":"1","\_score":1.0,"\_source":  { "first\_name":"john","last\_name":"Smith","age":25,"about":" I love to go rock climbing","interests":["spots","music"]}  },{"\_index":"megacorp","\_type":"employee","\_id":"3","\_score":1.0,"\_source":  { "first\_name":"fang","last\_name":"xiaobai","age":23,"about":"I love to go rock climbing","interests":["spots","music"]}  }]}} |

注意：返回结果不仅告知匹配了哪些文档，还包含了整个文档本身：显示搜索结果给最终用户所需的全部信息。

1. 条件查询：

查询姓fang的文档：

|  |
| --- |
| curl -XGET 'http://127.0.0.1:9200/megacorp/employee/\_search?q=first\_name:fang' |

返回的数据：

|  |
| --- |
| {"took":140,"timed\_out":false,"\_shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{"total":1,"max\_score":0.2876821,"hits":[{"\_index":"megacorp","\_type":"employee","\_id":"3","\_score":0.2876821,"\_source":  { "first\_name":"fang","last\_name":"xiaobai","age":23,"about":"I love to go rock climbing","interests":["spots","music"]}  }]}} |

查询姓wang的文档：

|  |
| --- |
| curl -XGET 'http://127.0.0.1:9200/megacorp/employee/\_search?q=first\_name:wang' |

返回的数据：

|  |
| --- |
| {"took":82,"timed\_out":false,"\_shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{"total":1,"max\_score":0.2876821,"hits":[{"\_index":"megacorp","\_type":"employee","\_id":"2","\_score":0.2876821,"\_source":  { "first\_name":"wang","last\_name":"su","age":23,"about":"I love to go rock climbing","interests":["spots","music"]}  }]}} |

使用查询表达式进行搜索

Elasticsearch 提供一个丰富灵活的查询语言叫做 查询表达式 ， 它支持构建更加复杂和健壮的查询。

领域特定语言 （DSL）， 指定了使用一个 JSON 请求。

查询 姓名为Smith的文档：

|  |
| --- |
| curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'  {  "query" : {  "match" : {  "last\_name" : "Smith"  }  }  }  ' |

返回的数据：

|  |
| --- |
| {  "took" : 6,  "timed\_out" : false,  "\_shards" : {  "total" : 5,  "successful" : 5,  "skipped" : 0,  "failed" : 0  },  "hits" : {  "total" : 1,  "max\_score" : 0.2876821,  "hits" : [  {  "\_index" : "megacorp",  "\_type" : "employee",  "\_id" : "1",  "\_score" : 0.2876821,  "\_source" : {  "first\_name" : "john",  "last\_name" : "Smith",  "age" : 25,  "about" : " I love to go rock climbing",  "interests" : [  "spots",  "music"  ]  }  }  ]  }  } |

更加复杂的搜索

查找last\_name=smith，并且年龄大于30岁的。

|  |
| --- |
| curl -XGET 'localhost:9200/megacorp/employee/\_search' -H 'Content-Type: application/json' -d'  {  "query" : {  "bool" : {  "must" : {  "match":{  "last\_name":"Smith"  }  },  "filter":{  "range":{  "age":{"gt":"30"}  }  }  }  }  }  ' |

返回结果：

|  |
| --- |
| {"took":5,"timed\_out":false,"\_shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{"total":1,"max\_score":0.6931472,"hits":[{"\_index":"megacorp","\_type":"employee","\_id":"4","\_score":0.6931472,"\_source":  { "first\_name":"jane","last\_name":"Smith","age":32,"about":"I love to go rock climbing","interests":["spots","music"]}  }]}} |

全文搜索

|  |
| --- |
| curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'  {  "query" : {  "match" : {  "about" : "rock climbing"  }  }  }  ' |

返回的结果：

|  |
| --- |
| {  "took" : 11,  "timed\_out" : false,  "\_shards" : {  "total" : 5,  "successful" : 5,  "skipped" : 0,  "failed" : 0  },  "hits" : {  "total" : 4,  "max\_score" : 0.5753642,  "hits" : [  {  "\_index" : "megacorp",  "\_type" : "employee",  "\_id" : "1",  "\_score" : 0.5753642,  "\_source" : {  "first\_name" : "john",  "last\_name" : "Smith",  "age" : 25,  "about" : " I love to go rock climbing",  "interests" : [  "spots",  "music"  ]  }  }, |

在hits节点下，\_score是相关性得分.

Elasticsearch 默认按照相关性得分排序，即每个文档跟查询的匹配程度。第一个最高得分的结果很明显：John Smith 的 about 属性清楚地写着 “rock climbing” 。这是一个很好的案例，阐明了 Elasticsearch 如何 在 全文属性上搜索并返回相关性最强的结果。Elasticsearch中的 相关性 概念非常重要，也是完全区别于传统关系型数据库的一个概念，数据库中的一条记录要么匹配要么不匹配。

短语匹配

|  |
| --- |
| curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type:application/json' -d '  {  "query":{  "match\_phrase" : {  "about" : "wangsu"  }  }  }  ' |

返回结果：

|  |
| --- |
| {  "took" : 8,  "timed\_out" : false,  "\_shards" : {  "total" : 5,  "successful" : 5,  "skipped" : 0,  "failed" : 0  },  "hits" : {  "total" : 1,  "max\_score" : 0.80259144,  "hits" : [  {  "\_index" : "megacorp",  "\_type" : "employee",  "\_id" : "4",  "\_score" : 0.80259144,  "\_source" : {  "first\_name" : "jane",  "last\_name" : "Smith",  "age" : 32,  "about" : "I love wangsu",  "interests" : [  "spots",  "music"  ]  }  }  ]  }  } |

高亮搜索

许多应用都倾向于在每个搜索结果中 高亮 部分文本片段，以便让用户知道为何该文档符合查询条件。在 Elasticsearch 中检索出高亮片段也很容易。

|  |
| --- |
| curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type:application/json' -d '  {  "query":{  "match\_phrase":{  "about" : "wangsu"  }  },  "highlight" : {  "fields" : {  "about" : {}  }  }  }  ' |

当执行该查询时，返回结果与之前一样，与此同时结果中还多了一个叫做 highlight 的部分。这个部分包含了 about 属性匹配的文本片段，并以 HTML 标签 <em></em> 封装：

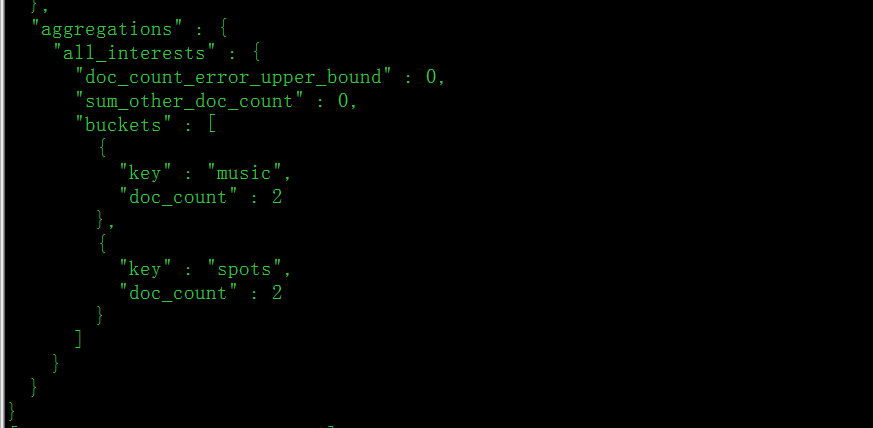
|  |
| --- |
| {  "took" : 103,  "timed\_out" : false,  "\_shards" : {  "total" : 5,  "successful" : 5,  "skipped" : 0,  "failed" : 0  },  "hits" : {  "total" : 1,  "max\_score" : 0.80259144,  "hits" : [  {  "\_index" : "megacorp",  "\_type" : "employee",  "\_id" : "4",  "\_score" : 0.80259144,  "\_source" : {  "first\_name" : "jane",  "last\_name" : "Smith",  "age" : 32,  "about" : "I love wangsu",  "interests" : [  "spots",  "music"  ]  },  "highlight" : {  "about" : [  "I love <em>wangsu</em>"  ]  }  }  ]  }  } |

分析：

支持管理者对雇员目录做分析

Elasticsearch 有一个功能叫聚合（aggregations），允许我们基于数据生成一些精细的分析结果。聚合与 SQL 中的 GROUP BY 类似但更强大。

|  |
| --- |
| **curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d' {**  **"query":{**  **"match":{**  **"last\_name":"smith"**  **}**  **},**  **"aggs": {**  **"all\_interests": {**  **"terms": { "field": "interests.keyword" }**  **}**  **}**  **}**  **'** |



聚合还支持分级汇总 。比如，查询特定兴趣爱好员工的平均年龄：

|  |
| --- |
| curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'  {  "aggs" : {  "all\_interests" : {  "terms" : { "field" : "interests.keyword" },  "aggs" : {  "avg\_age" : {  "avg" : { "field" : "age" }  }  }  }  }  }  ' |

基本操作到此结束。

集群内的原理：

分布式特性：

Elasticsearch 尽可能地屏蔽了分布式系统的复杂性。这里列举了一些在后台自动执行的操作：

* 分配文档到不同的容器 或 *分片* 中，文档可以储存在一个或多个节点中
* 按集群节点来均衡分配这些分片，从而对索引和搜索过程进行负载均衡
* 复制每个分片以支持数据冗余，从而防止硬件故障导致的数据丢失
* 将集群中任一节点的请求路由到存有相关数据的节点
* 集群扩容时无缝整合新节点，重新分配分片以便从离群节点恢复

空集群：

一个运行中的 Elasticsearch 实例称为一个 节点，而集群是由一个或者多个拥有相同 cluster.name 配置的节点组成， 它们共同承担数据和负载的压力。当有节点加入集群中或者从集群中移除节点时，集群将会重新平均分布所有的数据。

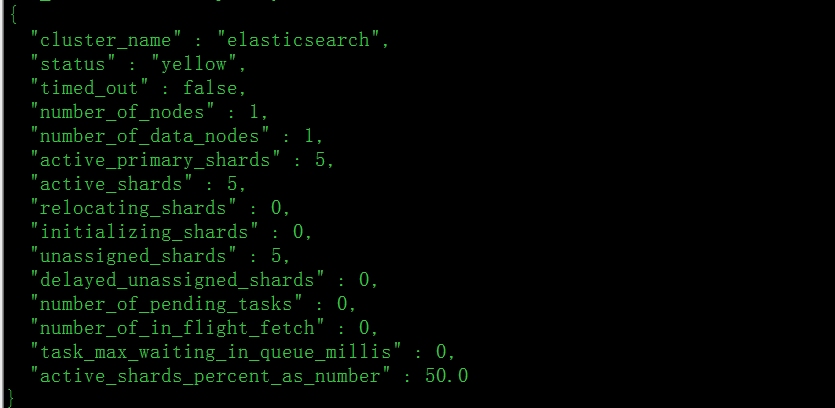
当一个节点被选举成为 主 节点时， 它将负责管理集群范围内的所有变更，例如增加、删除索引，或者增加、删除节点等。 而主节点并不需要涉及到文档级别的变更和搜索等操作，所以当集群只拥有一个主节点的情况下，即使流量的增加它也不会成为瓶颈。 任何节点都可以成为主节点。我们的示例集群就只有一个节点，所以它同时也成为了主节点。

作为用户，我们可以将请求发送到 集群中的任何节点 ，包括主节点。 每个节点都知道任意文档所处的位置，并且能够将我们的请求直接转发到存储我们所需文档的节点。 无论我们将请求发送到哪个节点，它都能负责从各个包含我们所需文档的节点收集回数据，并将最终结果返回給客户端。 Elasticsearch 对这一切的管理都是透明的。

集群健康：

Elasticsearch 的集群监控信息中包含了许多的统计数据，其中最为重要的一项就是 集群健康 ， 它在status 字段中展示为 green 、 yellow 或者 red 。

GET /\_cluster/health?pretty



1. Status

Green: 所有的主分片和副分片都正常运行。

Yellow：所有的主分片都正常运行，但不是所有的副分片都正常运行。

Red: 有主分片没有正常运行。

添加索引

我们往 Elasticsearch 添加数据时需要用到 索引 —— 保存相关数据的地方。 索引实际上是指向一个或者多个物理 分片 的 逻辑命名空间 。

一个 分片 是一个底层的 工作单元 ，它仅保存了 全部数据中的一部分。 在[分片内部机制](https://www.elastic.co/guide/cn/elasticsearch/guide/current/inside-a-shard.html)中，我们将详细介绍分片是如何工作的，而现在我们只需知道一个分片是一个 Lucene 的实例，以及它本身就是一个完整的搜索引擎。 我们的文档被存储和索引到分片内，但是应用程序是直接与索引而不是与分片进行交互

Elasticsearch 是利用分片将数据分发到集群内各处的。分片是数据的容器，文档保存在分片内，分片又被分配到集群内的各个节点里。 当你的集群规模扩大或者缩小时， Elasticsearch 会自动的在各节点中迁移分片，使得数据仍然均匀分布在集群里。

一个分片可以是 主 分片或者 副本 分片。 索引内任意一个文档都归属于一个主分片，所以主分片的数目决定着索引能够保存的最大数据量。

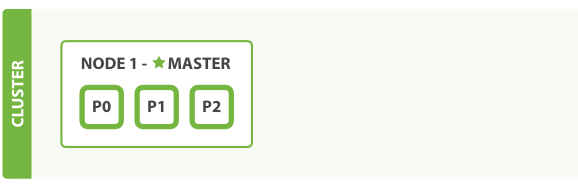
技术上来说，一个主分片最大能够存储 Integer.MAX\_VALUE - 128 个文档，但是实际最大值还需要参考你的使用场景：包括你使用的硬件， 文档的大小和复杂程度，索引和查询文档的方式以及你期望的响应时长。

一个副本分片只是一个主分片的拷贝。 副本分片作为硬件故障时保护数据不丢失的冗余备份，并为搜索和返回文档等读操作提供服务。

在索引建立的时候就已经确定了主分片数，但是副本分片数可以随时修改。

让我们在包含一个空节点的集群内创建名为 blogs 的索引。 索引在默认情况下会被分配5个主分片， 但是为了演示目的，我们将分配3个主分片和一份副本（每个主分片拥有一个副本分片）：

拥有一个索引的单节点集群：



添加故障转移

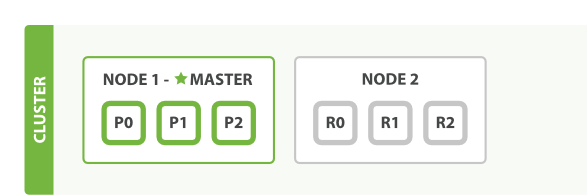
当集群中只有一个节点在运行时，意味着会有一个单点故障问题——没有冗余。 幸运的是，我们只需再启动一个节点即可防止数据丢失。

**启动第二个节点**

为了测试第二个节点启动后的情况，你可以在同一个目录内，完全依照启动第一个节点的方式来启动一个新节点（参考[安装并运行 Elasticsearch](https://www.elastic.co/guide/cn/elasticsearch/guide/current/running-elasticsearch.html)）。多个节点可以共享同一个目录。

当你在同一台机器上启动了第二个节点时，只要它和第一个节点有同样的 cluster.name 配置，它就会自动发现集群并加入到其中。 但是在不同机器上启动节点的时候，为了加入到同一集群，你需要配置一个可连接到的单播主机列表。 详细信息请查看[最好使用单播代替组播](https://www.elastic.co/guide/cn/elasticsearch/guide/current/important-configuration-changes.html#unicast)

**图 3. 拥有两个节点的集群——所有主分片和副本分片都已被分配**



当第二个节点加入到集群后，3个 副本分片 将会分配到这个节点上——每个主分片对应一个副本分片。 这意味着当集群内任何一个节点出现问题时，我们的数据都完好无损。

所有新近被索引的文档都将会保存在主分片上，然后被并行的复制到对应的副本分片上。这就保证了我们既可以从主分片又可以从副本分片上获得文档。

cluster-health 现在展示的状态为 green ，这表示所有6个分片（包括3个主分片和3个副本分片）都在正常运行。

{

"cluster\_name": "elasticsearch",

"status": "green",

"timed\_out": false,

"number\_of\_nodes": 2,

"number\_of\_data\_nodes": 2,

"active\_primary\_shards": 3,

"active\_shards": 6,

"relocating\_shards": 0,

"initializing\_shards": 0,

"unassigned\_shards": 0,

"delayed\_unassigned\_shards": 0,

"number\_of\_pending\_tasks": 0,

"number\_of\_in\_flight\_fetch": 0,

"task\_max\_waiting\_in\_queue\_millis": 0,

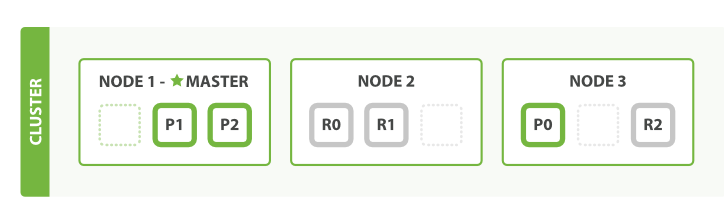
"active\_shards\_percent\_as\_number": 100

|  |  |
| --- | --- |
|  | 集群 status 值为 green 。 |

我们的集群现在不仅仅是正常运行的，并且还处于 *始终可用* 的状态。

、

**图 4. 拥有三个节点的集群——为了分散负载而对分片进行重新分配**



Node 1 和 Node 2 上各有一个分片被迁移到了新的 Node 3 节点，现在每个节点上都拥有2个分片，而不是之前的3个。 这表示每个节点的硬件资源（CPU, RAM, I/O）将被更少的分片所共享，每个分片的性能将会得到提升。

分片是一个功能完整的搜索引擎，它拥有使用一个节点上的所有资源的能力。 我们这个拥有6个分片（3个主分片和3个副本分片）的索引可以最大扩容到6个节点，每个节点上存在一个分片，并且每个分片拥有所在节点的全部资源。

### 更多的扩容[编辑](https://github.com/elasticsearch-cn/elasticsearch-definitive-guide/edit/cn/020_Distributed_Cluster/30_Scale_more.asciidoc)

但是如果我们想要扩容超过6个节点怎么办呢？

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

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

PUT /blogs/\_settings

{

"number\_of\_replicas" : 2

}

**拷贝为 CURL**[在 SENSE 中查看](http://localhost:5601/app/sense/?load_from=https://www.elastic.co/guide/cn/elasticsearch/guide/current/snippets/020_Distributed_Cluster/30_Replicas.json)

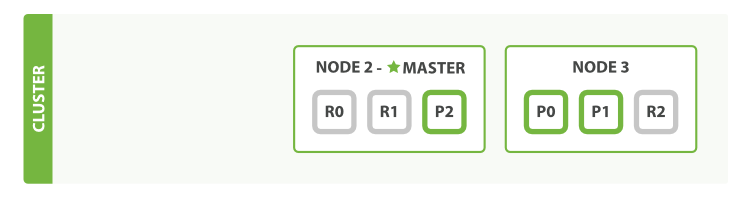
如[图 5 “将参数 number\_of\_replicas 调大到 2”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_scale_horizontally.html#cluster-three-nodes-two-replicas)所示， blogs 索引现在拥有9个分片：3个主分片和6个副本分片。 这意味着我们可以将集群扩容到9个节点，每个节点上一个分片。相比原来3个节点时，集群搜索性能可以提升 3 倍。

**图 5. 将参数 number\_of\_replicas 调大到 2**

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

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

**关闭了一个节点后的集群**



我们关闭的节点是一个主节点。而集群必须拥有一个主节点来保证正常工作，所以发生的第一件事情就是选举一个新的主节点： Node 2 。

在我们关闭 Node 1 的同时也失去了主分片 1 和 2 ，并且在缺失主分片的时候索引也不能正常工作。 如果此时来检查集群的状况，我们看到的状态将会为 red ：不是所有主分片都在正常工作。

幸运的是，在其它节点上存在着这两个主分片的完整副本， 所以新的主节点立即将这些分片在 Node 2 和Node 3 上对应的副本分片提升为主分片， 此时集群的状态将会为 yellow 。 这个提升主分片的过程是瞬间发生的，如同按下一个开关一般。

为什么我们集群状态是 yellow 而不是 green 呢？ 虽然我们拥有所有的三个主分片，但是同时设置了每个主分片需要对应2份副本分片，而此时只存在一份副本分片。 所以集群不能为 green 的状态，不过我们不必过于担心：如果我们同样关闭了 Node 2 ，我们的程序 依然 可以保持在不丢任何数据的情况下运行，因为 Node 3 为每一个分片都保留着一份副本。

如果我们重新启动 Node 1 ，集群可以将缺失的副本分片再次进行分配，那么集群的状态也将如[图 5 “将参数 number\_of\_replicas 调大到 2”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_scale_horizontally.html#cluster-three-nodes-two-replicas)所示。 如果 Node 1 依然拥有着之前的分片，它将尝试去重用它们，同时仅从主分片复制发生了修改的数据文件。