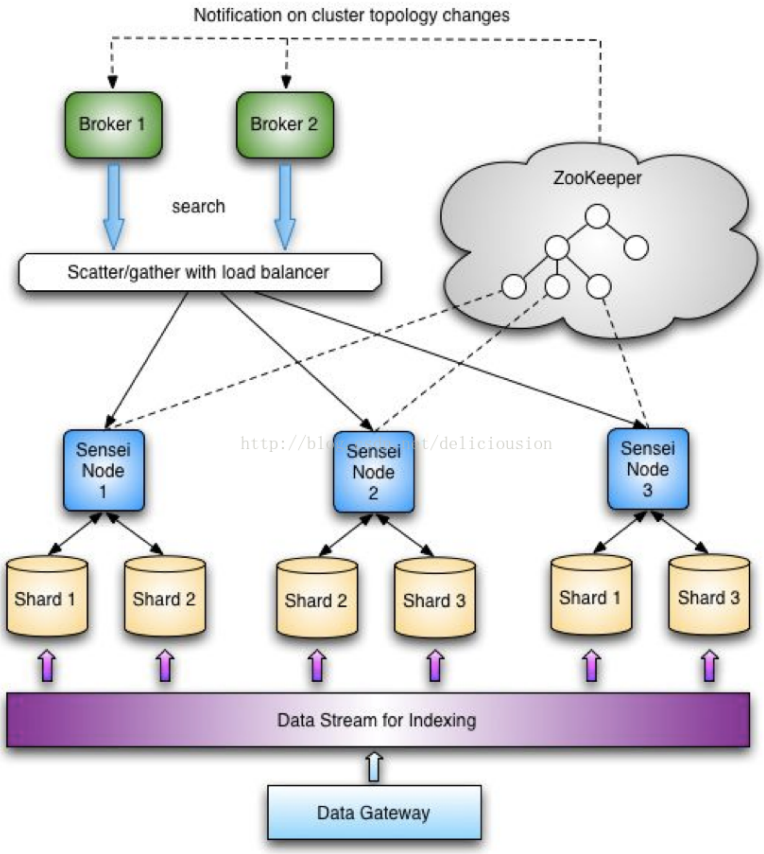
ElasticSearch 搜索中间件介绍及分析

中国人民大学
信息学院
李瀚(2017104169)
2018.10.14

简介

1.基本概念

Elasticsearch（ES）是一个基于 Lucene 构建的开源、分布式、RESTful 接口的全文搜索引擎。Elasticsearch 还是一个分布式文档数据库，其中每个字段均可被索引，而且每个字段的数据均可被搜索，ES 能够横向扩展至数以百计的服务器存储以及处理 PB 级的数据。可以在极短的时间内存储、搜索和分析大量的数据。通常作为具有复杂搜索场景情况下的核心发动机。



ElasticSearch 就是为高可用和可扩展而生的。可以通过购置性能更强的服务器或者升级硬件来完成系统扩展，称为垂直或向上扩展（Vertical Scale/Scaling Up）。另一方面，增加更多的服务器来完成系统扩展，称为水平扩展或者向外扩

展（Horizontal Scale/Scaling Out）。尽管 ES 能够利用更强劲的硬件，垂直扩展毕竟还是有它的极限。真正的可扩展性来自于水平扩展，通过向集群中添加更多的节点来分担负载，增加可靠性。ES 天生就是分布式的：它知道如何管理多个节点来完成扩展和实现高可用性。这也意味你的应用不需要做任何的改动。

2. Elasticsearch 可以做什么

当你经营一家网上商店，你可以让你的客户搜索你卖的商品。在这种情况下，你可以使用 ElasticSearch 来存储你的整个产品目录和库存信息，为客户提供精准搜索，可以为客户推荐相关商品。

当你想收集日志或者交易数据的时候，需要分析和挖掘这些数据，寻找趋势，进行统计，总结，或发现异常。在这种情况下，你可以使用 Logstash 或者其他工具来进行收集数据，当这引起数据存储到 ElasticsSearch 中。你可以搜索和汇总这些数据，找到任何你感兴趣的信息。

对于程序员来说，比较有名的案例是 GitHub，GitHub 的搜索是基于 ElasticSearch 构建的，在 github.com/search 页面，你可以搜索项目、用户、issue、pull request，还有代码。共有 40~50 个索引库，分别用于索引网站需要跟踪的各种数据。虽然只索引项目的主分支（master），但这个数据量依然巨大，包括 20 亿个索引文档，30TB 的索引文件。

关键领域概念

- 集群（Cluster）:

包含一个或多个具有相同 cluster.name 的节点.

集群内节点协同工作，共享数据，并共同分担工作负荷。

由于节点是从属集群的，集群会自我重组来均匀地分发数据.

cluster Name 是很重要的，因为每个节点只能是群集的一部分，当该节点被设置为相同的名称时，就会自动加入群集。

集群中通过选举产生一个 mater 节点，它将负责管理集群范畴的变更，例如创建或删除索引，添加节点到集群或从集群删除节点。master 节点无需参与文档层面的变更和搜索，这意味着仅有一个 master 节点并不会因流量增长而成为瓶颈。任意一个节点都可以成为 master 节点。我们例举的集群只有一个节点，因此它会扮演 master 节点的角色。

作为用户，我们可以访问包括 master 节点在内的集群中的任一节点。每个节点都知道各个文档的位置，并能够将我们的请求直接转发到拥有我们想要的数据的节点。无论我们访问的是哪个节点，它都会控制从拥有数据的节点收集响应的过程，并返回给客户端最终的结果。这一切都是由 Elasticsearch 透明管理的

- 节点(node):

一个节点是一个逻辑上独立的服务，可以存储数据，并参与集群的索引和搜索功能, 一个节点也有唯一的名字，群集通过节点名称进行管理和通信.

- 索引（Index)：

索引与关系型数据库实例(Database)相当。

索引只是一个 逻辑命名空间，它指向一个或多个分片(shards)，内部用 Apache Lucene 实现索引中数据的读写

- 文档类型（Type)：

相当于数据库中的 table 概念。

每个文档在ElasticSearch中都必须设定它的类型。文档类型使得同一个索引中在存储结构不同文档时，只需要依据文档类型就可以找到对应的参数映射(Mapping)信息，方便文档的存取

- 文档（Document）：

相当于数据库中的row， 是可以被索引的基本单位。例如，你可以有一个的客户文档，有一个产品文档，还有一个订单的文档。文档是以JSON格式存储的。在一个索引中，您可以存储多个的文档。请注意，虽然在一个索引中有多分文档，但这些文档的结构是一致的，并在第一次存储的时候指定， 文档属于一种 类型(type)，各种各样的类型存在于一个 索引 中。

类比：

关系数据库 ⇒ 数据库 ⇒ 表 ⇒ 行 ⇒ 列(Columns)

Elasticsearch ⇒ 索引 ⇒ 类型 ⇒ 文档 ⇒ 字段(Fields)

- Mapping：

相当于数据库中的 schema，用来约束字段的类型，不过 Elasticsearch 的 mapping 可以自动根据数据创建

- 分片(shard)：

是 工作单元(worker unit) 底层的一员，用来分配集群中的数据，它只负责保存索引中所有数据的一小片。

分片是一个独立的Lucene实例，并且它自身也是一个完整的搜索引擎。

文档存储并且被索引在分片中，但是我们的程序并不会直接与它们通信。取而代之，它们直接与索引进行通信的

把分片想象成一个数据的容器。数据被存储在分片中，然后分片又被分配在集群

的节点上。当你的集群扩展或者缩小时，elasticsearch 会自动的在节点之间迁移分配分片，以便集群保持均衡

分片分为 主分片(primary shard) 以及 从分片(replica shard) 两种。在你的索引中，每一个文档都属于一个主分片。从分片只是主分片的一个副本，它用于提供数据的冗余副本，在硬件故障时提供数据保护，同时服务于搜索和检索这种只读请求。

索引中的主分片的数量在索引创建后就固定下来了，但是从分片的数量可以随时改变。

一个索引默认设置了 5 个主分片，每个主分片有一个从分片对应。

应用场景

- 站内搜索：主要和 Solr 竞争，属于后起之秀
- NoSQL json 文档数据库：主要抢占 Mongo 的市场，它在读写性能上优于 Mongo ，同时也支持地理位置查询，还方便地理位置和文本混合查询，属于歪打正着 （对比测试参见：http://blog.quarkslab.com/mongodb-vs-elasticsearch-the-quest-of-the-holy-performances.html）
- 监控：统计以及日志类时间序的数据的存储和分析以及可视化，这方面是引领者

国外：

Wikipedia 使用 ES 提供全文搜索并高亮关键字、StackOverflow 结合全文搜索与地理位置查询、Github 使用 Elasticsearch 检索 1300 亿行的代码

国内：

百度（在 casio、云分析、网盟、预测、文库、直达号、钱包、风控等业务上都应用了 ES，单集群每天导入 30TB+数据，总共每天 60TB+）、新浪，阿里巴巴、腾讯等公司均有对 ES 的使用

使用比较广泛的平台 ELK(ElasticSearch, Logstash, Kibana)

竞品比较

Solr 是 Apache Lucene 项目的开源企业搜索平台。其主要功能包括全文检索、命中标示、分面搜索、动态聚类、数据库集成，以及富文本（如 Word、PDF）的处理。

Solr 是高度可扩展的，并提供了分布式搜索和索引复制。Solr 是最流行的企业级搜索引擎，Solr4 还增加了 NoSQL 支持。

Solr 是用 Java 编写、运行在 Servlet 容器（如 Apache Tomcat 或 Jetty）的一个独立的全文搜索服务器。 Solr 采用了 Lucene Java 搜索库为核心的全文索引和搜索，并具有类似 REST 的 HTTP/XML 和 JSON 的 API。

Solr 强大的外部配置功能使得无需进行 Java 编码，便可对 其进行调整以适应多种类型的应用程序。Solr 有一个插件架构，以支持更多的高级定制

- Elasticsearch 与 Solr 的比较总结

二者安装都很简单

Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能。

Solr 支持更多格式的数据，而 Elasticsearch 仅支持 json 文件格式。

Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供。

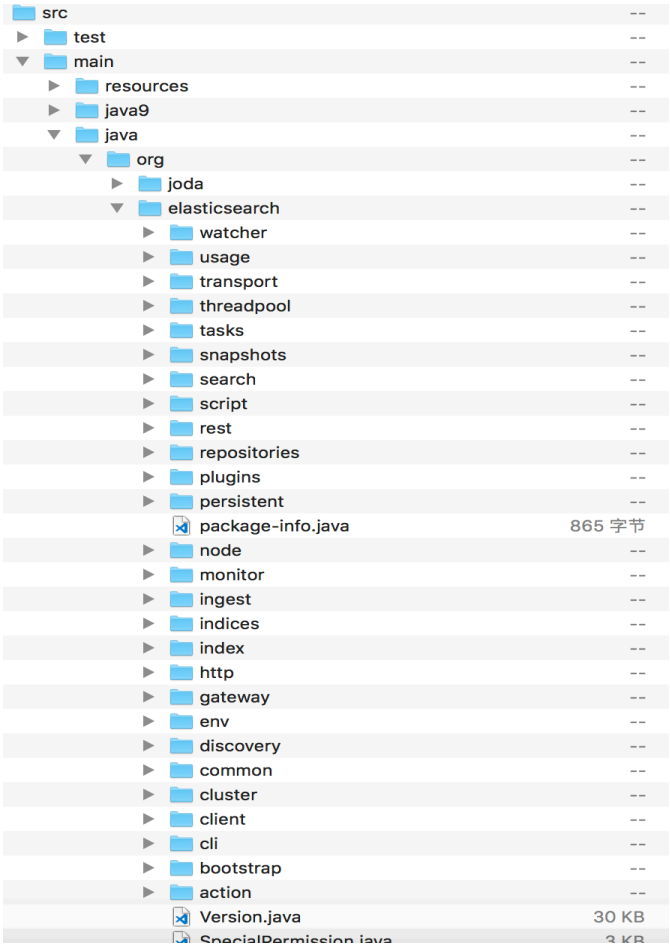
Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于 Elasticsearch。

Solr 是传统搜索应用的有力解决方案，但 Elasticsearch 更适用于新兴的实时搜索应用。

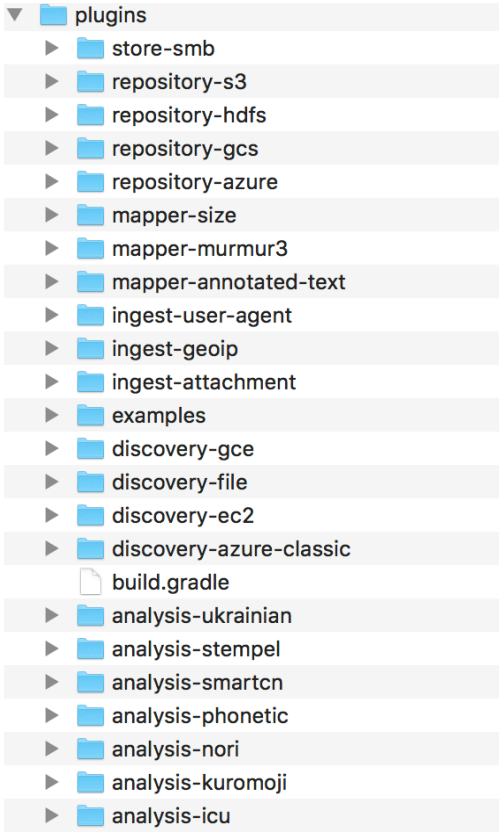
架构介绍

代码结构：

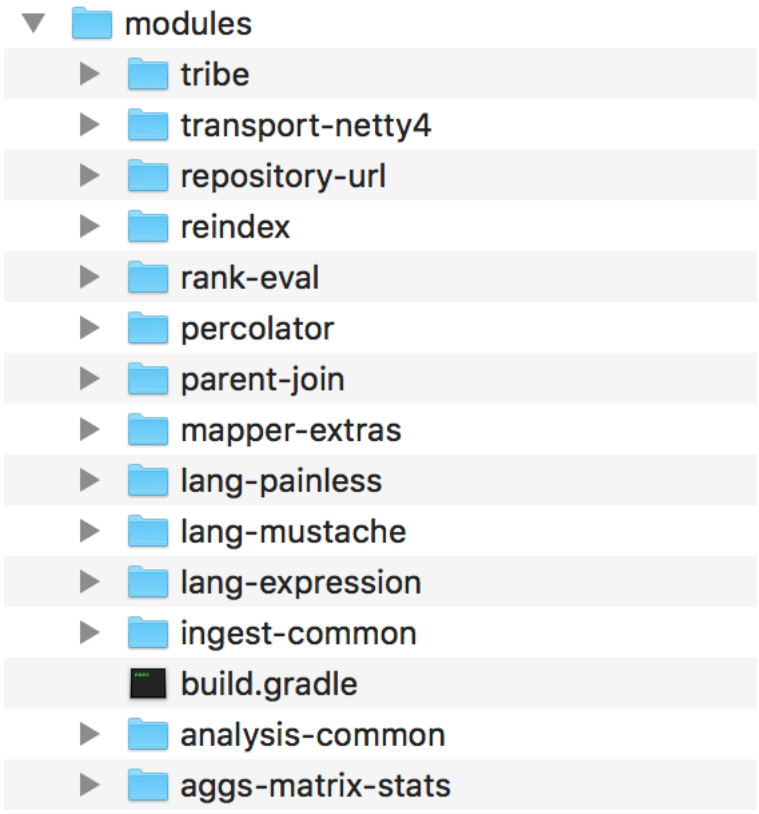
Server



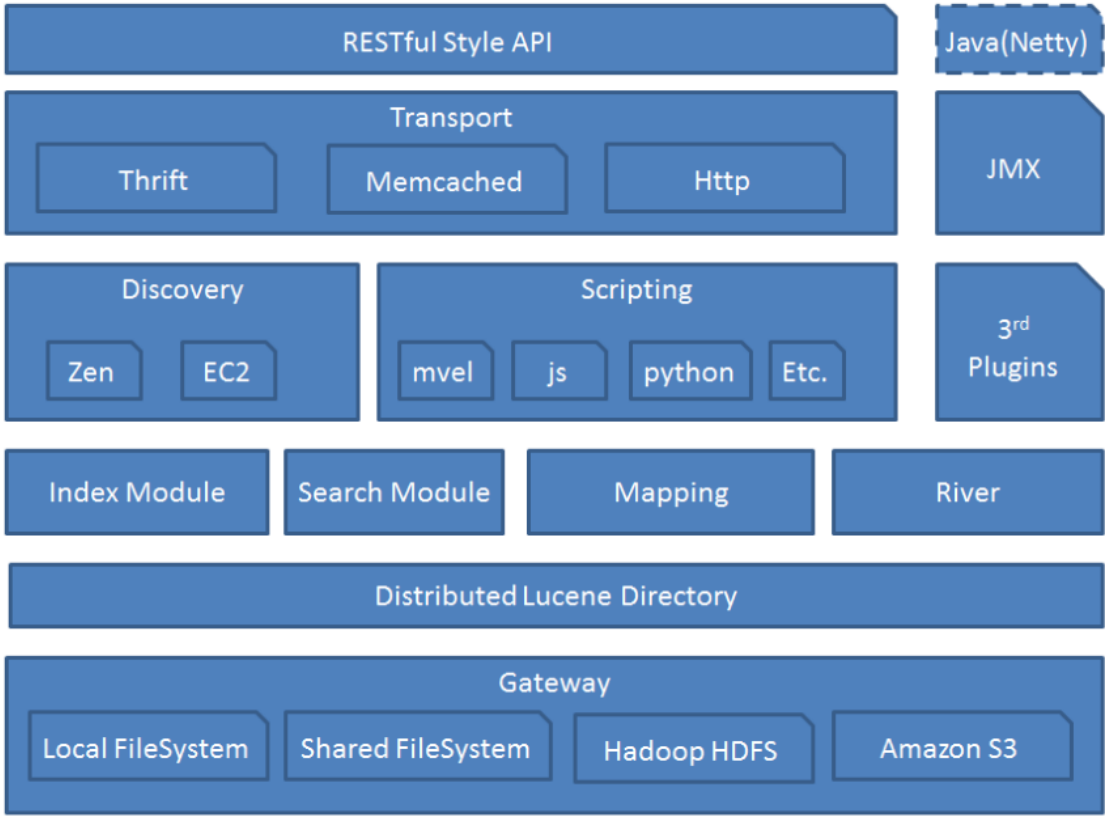
2. Plugin：



3. 第三方模块：



模块结构图如下



- Gateway:

代表 ES 的持久化存储方式,包含索引信息,ClusterState(集群信息),mapping,索引碎片信息，以及 transaction log 等

1．对于分布式集群来说，当一个或多个节点 down 掉了，能够保证我们的数据不能丢，最通用的解放方案就是对失败节点的数据进行复制，通过控制复制的份数可以保证集群有很高的可用性,复制这个方案的精髓主要是保证操作的时候没有单点，对一个节点的操作会同步到其他的复制节点上去。

2. ES 一个索引会拆分成多个碎片，每个碎片可以拥有一个或多个副本（创建索引的时候可以配置），这里有个例子，每个索引分成 3 个碎片，每个碎片有 2 个副本，如下：

```
$ curl -XPUT http://localhost:9200/twitter/ -d '
index :
```

number_of_shards : 3

number_of_replicas : 2

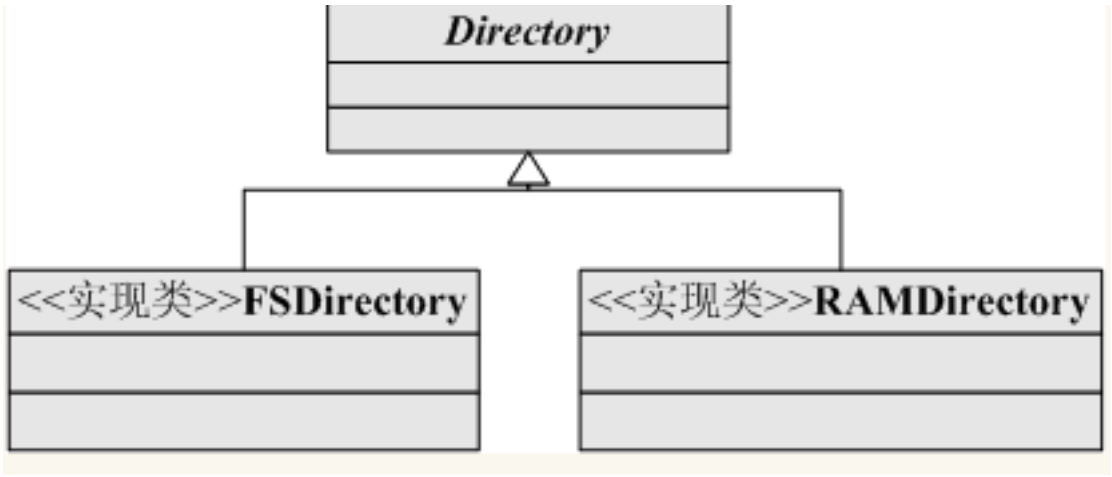
每个操作会自动路由主碎片所在的节点，在上面执行操作，并且同步到其他复制节点，通过使用“non blocking IO”模式所有复制的操作都是并行执行的，也就是说如果你的节点的副本越多，你网络上的流量消耗也会越大。复制节点同样接受来自外面的读操作，意义就是你的复制节点越多，你的索引的可用性就越强，对搜索的可伸缩行就更好，能够承载更多的操作。

第一次启动的时候，它会去持久化设备读取集群的状态信息（创建的索引，配置等）然后执行应用它们（创建索引，创建 mapping 映射等），每一次 shard 节点第一次实例化加入复制组，它都会从长持久化存储里面恢复它的状态信息

- Lucence Directory:

是 lucene 的框架服务发现以及选主 ZenDiscovery: 用来实现节点自动发现，还有 Master 节点选取，假如 Master 出现故障，其它的这个节点会自动选举，产生一个新的 Master。

它是Lucene存储的一个抽象，由此派生了两个类:FSDirectory和RAMDirectory，用于控制索引文件的存储位置。使用 FSDirectory 类，就是存储到硬盘；使用 RAMDirectory 类，则是存储到内存



一个 Directory 对象是一份文件的清单。文件可能只在被创建的时候写一次。一旦文件被创建，它将只被读取或者删除。在读取的时候进行写入操作是允许的。

- Discovery

1. 节点启动后先 ping（这里的 ping 是 Elasticsearch 的一个 RPC 命令。如果 discovery.zen.ping.unicast.hosts 有设置，则 ping 设置中的 host，否则尝试 ping localhost 的几个端口， Elasticsearch 支持同一个主机启动多个节点）。

2.Ping 的 response 会包含该节点的基本信息以及该节点认为的 master 节点

3.选举开始，先从各节点认为的 master 中选，规则很简单，按照 id 的字典序排序，取第一个。

4.如果各节点都没有认为的 master，则从所有节点中选择，规则同上。这里有个限制条件就是 discovery.zen.minimum_master_nodes，如果节点数达不到最小值的限制，则循环上述过程，直到节点数足够可以开始选举

5.最后选举结果是肯定能选举出一个 master，如果只有一个 local 节点那就选出的是自己

6.如果当前节点是 master，则开始等待节点数达到 minimum_master_nodes，然

后提供服务，如果当前节点不是 master，则尝试加入 master.

7.ES 支持任意数目的集群（1-N）,所以不能像 Zookeeper/Etcd 那样限制节点必须是奇数，也就无法用投票的机制来选主，而是通过一个规则，只要所有的节点都遵循同样的规则，得到的信息都是对等的，选出来的主节点肯定是一致的.但分布式系统的问题就出在信息不对等的情况，这时候很容易出现脑裂（Split-Brain）的问题，大多数解决方案就是设置一个 quorum 值，要求可用节点必须大于 quorum（一般是超过半数节点），才能对外提供服务。而 Elasticsearch 中，这个 quorum 的配置就是 discovery.zen.minimum_master_nodes 。

- memcached

1.通过 memecached 协议来访问 ES 的接口,支持二进制和文本两种协议.通过一个名为 transport-memcached 插件提供

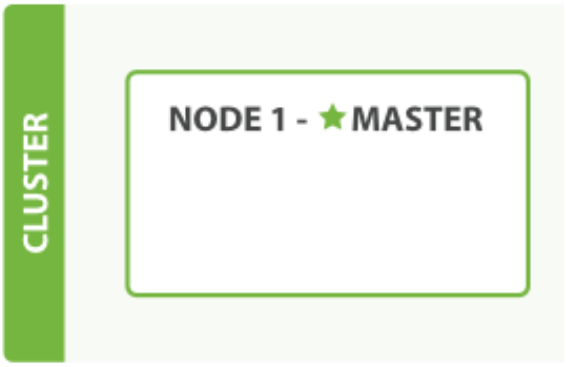
2.Memcached 命令会被映射到 REST 接口，并且会被同样的 REST 层处理，memcached 命令列表包括：get/set/delete/quit

- River ：

代表 es 的一个数据源，也是其它存储方式（如：数据库）同步数据到 es 的一个方法。它是以插件方式存在的一个 es 服务，通过读取 river 中的数据并把它索引到 es 中，官方的 river 有 couchDB 的，RabbitMQ 的，Twitter 的，Wikipedia 的，river 这个功能将会在后面的文件中重点说到

分片示例

启用一个既没有数据，也没有索引的单一节点，如下图



在空的单节点集群中上创建一个叫做 blogs 的索引，设置 3 个主分片和一组从分片（每个主分片有一个从分片对应），代码如下：

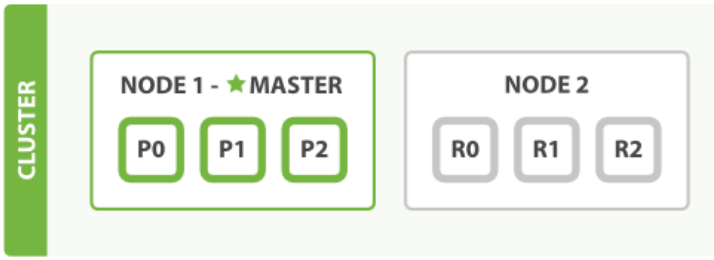
```
PUT /blogs
{
   "settings" : {
      "number_of_shards" : 3,
      "number_of_replicas" : 1
   }
}
```

集群示例图如下：（此时集群健康状态为： yellow 三个从分片还没有被分配到节点上）

```
{
   "cluster_name":          "elasticsearch",
   "status":                "yellow", <1>
   "timed_out":             false,
   "number_of_nodes":       1,
   "number_of_data_nodes":  1,
   "active_primary_shards": 3,
   "active_shards":         3,
   "relocating_shards":     0,
   "initializing_shards":   0,
   "unassigned_shards":     3 <2>
}
```

1.主分片(primary shards) 启动并且运行了，这时集群已经可以成功的处理任意请求，但是 从分片(replica shards) 没有完全被激活。事实上，当前这三个从分片都处于 unassigned（未分配）的状态，它们还未被分配到节点上。在同一个节点上保存相同的数据副本是没有必要的，如果这个节点故障了，就等同于所有的数据副本也丢失了。

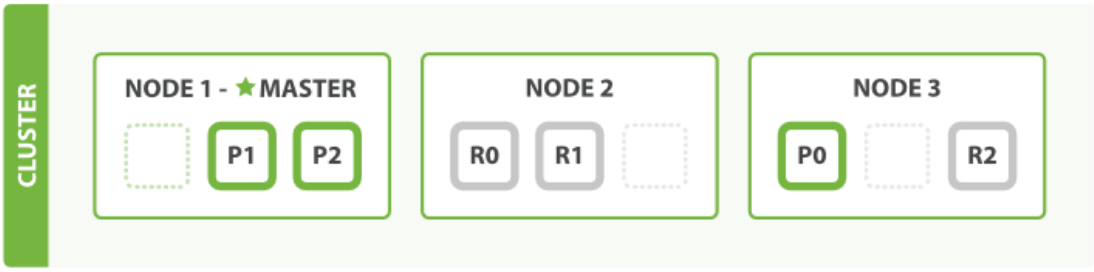
2.启动第二个节点，配置第二个节点与第一个节点的 cluster.name 相同（./config/elasticsearch.yml 文件中的配置），它就能自动发现并加入到第一个节点的集群中,如下图：



```
{
   "cluster_name":          "elasticsearch",
   "status":                "green", <1>
   "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
}
```

3.cluster-health 的状态为 green，这意味着所有的 6 个分片（三个主分片和三个从分片）都已激活，文档在主节点和从节点上都能被检索。

随着应用需求的增长，启动第三个节点进行横向扩展，集群内会自动重组，如图



1.在 Node 1 和 Node 2 中分别会有一个分片被移动到 Node 3 上，这样一来，每个节点上就都只有两个分片了。这意味着每个节点的硬件资源(CPU、RAM、I/O)被更少的分片共享，所以每个分片就会有更好的性能表现

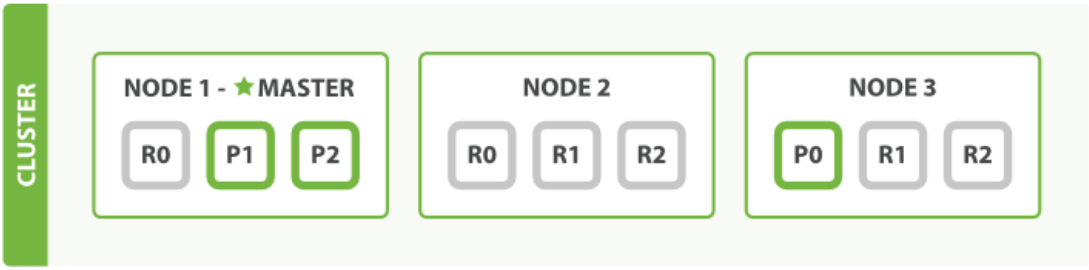
2.一共有 6 个分片(3 个主分片和 3 个从分片)，因此最多可以扩展到 6 个节点，每个节点上有一个分片，这样每个分片都可以使用到所在节点 100%的资源了

主分片的数量在索引创建的时候就已经指定了，实际上，这个数字定义了能存储到索引中的数据最大量(具体的数量取决于你的数据，硬件的使用情况)。例如，读请求——搜索或者文档恢复就可以由主分片或者从分片来执行，所以当你拥有更多份数据的时候，你就拥有了更大的吞吐量

从分片的数量可以在运行的集群中动态的调整，这样我们就可以根据实际需求扩展或者缩小规模。接下来，我们来增加一下从分片组的数量：

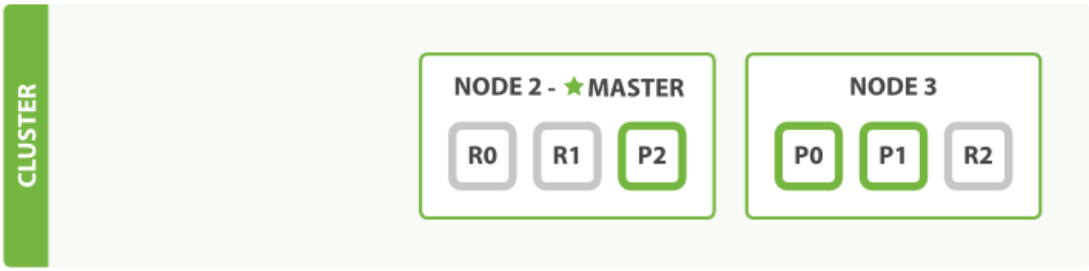
```
PUT /blogs/_settings
{
   "number_of_replicas" : 2
}
```

现在 blogs 的索引总共有 9 个分片：3 个主分片和 6 个从分片， 又会变成一个节点一个分片的状态了，最终得到了三倍搜索性能的三节点集群



说明：仅仅是在同样数量的节点上增加从分片的数量是根本不能提高性能的，因为每个分片都有访问系统资源的权限。你需要升级硬件配置以提高吞吐量。

尝试一下，把第一个节点杀掉，我们的集群就会如下图所示：



- 被杀掉的节点是主节点，主分片 1 和 2 在我们杀掉 Node 1 后就丢失了，我们的索引在丢失主节点的时候是不能正常工作的。如果我们在这个时候检查集群健康状态，将会显示 red：存在不可用的主节点

- 而为了集群的正常工作必须需要一个主节点，所以首先进行的进程就是从各节点中选择了一个新的主节点：Node 2
- 新的主节点所完成的第一件事情就是将这些在 Node 2 和 Node 3 上的从分片提升为主分片，然后集群的健康状态就变回至 yellow。这个提升的进程是瞬间完成了，就好像按了一下开关
- 如果再次杀掉 Node 2 的时候，我们的程序依旧可以在没有丢失任何数据的情况下运行，因为 Node 3 中依旧拥有每个分片的备份
- 如果我们重启 Node 1，集群就能够重新分配丢失的从分片，这样结果就会与三节点两从集群一致。如果 Node 1 依旧还有旧节点的内容，系统会尝试重新利用他们，并只会复制在故障期间的变更数据。

安装和使用

见官方帮助文档(https://github.com/elasticsearch)

总结

ElasticSearch 作为现在业内广泛使用的搜索引擎构建依赖服务，也被与 logstash，kibana 一起形成 ELK 日志分析监控中间件服务而被大家广泛使用。凭借上手容易，功能强大的特点，受到很多互联网公司和学习者追捧，社区迭代也很快，加之其开放的 plugin 架构，天然和现在的大数据技术亲和，在未来的发展中也将有更大的前途。