elastic

2018.11

浅析全文搜索引擎 Elasticsearch

赵知非

2018-11-01

elastic

目录

elastic

关于搜索

为什么需要全文搜索引擎

我要找一本机器学习的书

```
SELECT * FROM bookstore WHERE `book_name` LIKE '%机 器 学 习 %'
```

今天有多少人发微博带上了锦鲤

```
SELECT COUNT(*) FROM weibo_data WHERE `text` LIKE '%锦 鲤 %'
```

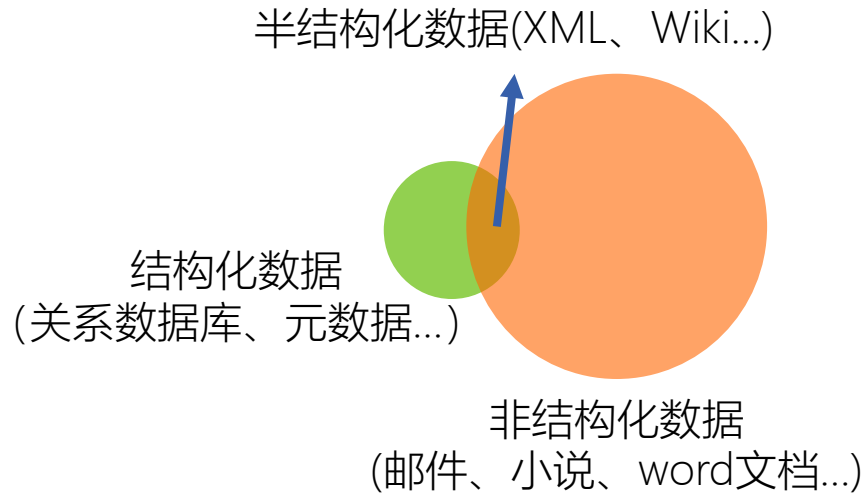
C语言中的printf()函数有漏洞，

检查公司代码库里有多少源代码文件用到了这个函数

```
SELECT * FROM code_library
WHERE `f_name` LIKE '%.c' AND `code` LIKE '%printf%'
```

Or

```
for file in ~/code_library/*.c; do cat $file | grep 'printf'; done;
```



非结构化数据搜索（全文检索）的特点

- 开销线性增长，需遍历所有文本
- 目的不明确，缺乏筛选条件
- 查全率问题：计算机 vs 电脑
- 查准率问题：巴西 vs 古巴西北部

elastic

关于搜索

全文搜索引擎的基石：倒排索引

文档编号	文档数据	
	id	title
0	1	谷歌地图之父跳槽Facebook
1	2	谷歌地图之父加盟Facebook
2	3	谷歌地图创始人拉斯离开谷歌加盟Facebook
3	4	谷歌地图之父跳槽Facebook 与wave项目取消有关
4	5	谷歌地图之父拉斯加盟社交网站Facebook

单词ID	单词	倒排列表（包含该单词的文档ID，DocID）
1	谷歌	0, 1, 2, 3, 4
2	地图	0, 1, 2, 3, 4
3	之父	0, 1, 3, 4
4	跳槽	0, 3
5	Facebook	0, 1, 2, 3, 4
6	加盟	1, 2, 4
7	创始人	2
8	拉斯	2, 4
9	离开	2
10	与	3
11	wave	3
12	项目	3
13	取消	3
14	有关	3
15	社交	4
16	网站	4

"谷歌之父" -> "谷歌", "之父" -> {0,1,2,3,4} ∩ {0,1,3,4} -> {0,1,3,4}

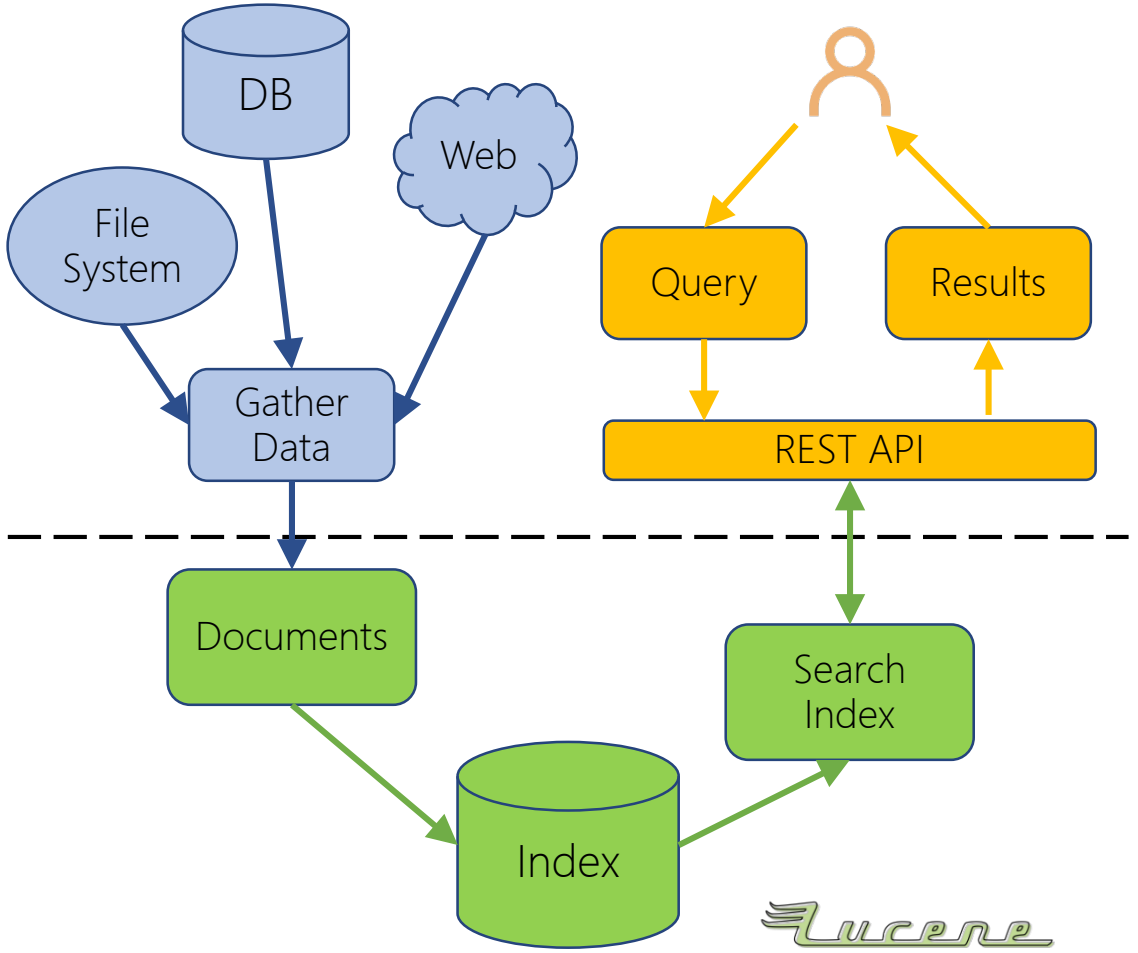
	基于倒排索引	数据库LIKE / grep工具
全文检索效率	对数复杂度	线性复杂度
空间开销	建立或更新索引有额外开销	无
匹配效果	通过词元（term）进行匹配， 效果依赖文本分析与分词算法	基于关键词的简单模糊匹配，词序不可颠倒
匹配度	评分机制，0~1	0 or 1
结果输出	可按匹配度排序后小批量输出，例如Top100的结果	返回所有匹配结果

elastic

Elasticsearch 简介

Elasticsearch 是一个分布式、可扩展、实时的搜索与数据分析引擎。

- 高性能、全功能：基于 Lucene 全文搜索引擎库；
- 自带分布式：实时分发，近实时搜索；
- 面向文档：每个字段都可以被索引；
- 起步简单：RESTful API，开箱即用；
- 易于扩展：胜任上百服务节点的扩展，支持PB级别的结构化或非结构化数据；
- 高可用：冗余存储，变更持久化
- 代码开源：基于Java编写



elastic

Elasticsearch 使用

安装与启用

- Java 8.0 以上
- REST 风格的 HTTP API
- 默认 9200 端口

```
curl -X <VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

```
curl -X GET 'http://localhost:9200/_count?pretty' -d '
{
    "query": {
        "match_all": {}
    }
}
'
```

参数	说明
VERB	HTTP方法：`GET`、 `POST`、 `PUT`、 `HEAD` 或者 `DELETE`
PROTOCOL	http 或者 https
HOST	集群中任意节点的主机名
PORT	端口号，默认是 9200
PATH	API 的终端路径
QUERY_STRING	任意可选的查询字符串参数
BODY	JSON格式的请求体 (如果需要)

安装与启用

- Java 8.0 以上
- REST 风格的 HTTP API
- 默认 9200 端口

```
curl -X <VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

```
curl -X GET 'http://localhost:9200/_count?pretty' -d '
{
    "query": {
        "match_all": {}
    }
}
'
```

返回结果

```
{
  "count" : 0,
  "_shards" : {
    "total" : 0,
    "successful" : 0,
    "skipped" : 0,
    "failed" : 0
  }
}
```

请求简写形式

```
GET /_count
{
    "query": {
        "match_all": {}
    }
}
```

elastic

Elasticsearch 使用

面向文档

- 序列化为JSON结构的实体对象
- string, number, bool, array, object

```
PUT /bookstore/books/1
{
  "name": "天龙八部",
  "writer": "金庸",
  "finished": true,
  "published_date": "1963-09-03",
  "pages": 2130,
  "roles": [
    {
      "name": "乔峰",
      "info": "丐帮帮主，辽国南院大王",
      "features": ["降龙十八掌", "打狗棒法"]
    },
    {
      "name": "虚竹",
      "info": "灵鹫宫尊主，逍遥派掌门，西夏驸马",
      "features": ["小无相功", "天山折梅手", "生死符"]
    },
    {
      "name": "段誉",
      "info": "镇南王世子",
      "features": ["北冥神功", "凌波微步", "六脉神剑"]
    }
  ]
}
```

文档元数据

- 索引（_index）：文档存储的位置
- 类型（_type）：文档所属的类
- 文档id（_id）：文档的唯一标识

Elasticsearch	关系型DB	MongoDB
索引 _index	Database	Database
类型 _type	表名	集合名
文档id _id	行主键	文档 id

```
{
  "_index": "bookstore",
  "_type": "books",
  "_id": "1",
  "_version": 3,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 2,
  "_primary_term": 1
}
```

```
{
  "_index": "bookstore",
  "_type": "books",
  "_id": "1",
  "_version": 3,
  "found": true,
  "_source": {
    "name": "天龙八部",
    "writer": "金庸",
    "finished": true,
    "published_date": "1963-09-03",
    "pages": 2130,
    "roles": [...]
  }
}
```

基础增删改查 – 新增文档

- PUT 方法：指定id
- POST 方法：自增id

```
PUT /{index}/{type}/{id}
{
  "field": "value",
  ...
}

POST /{index}/{type}/
{
  "field": "value",
  ...
}
```

基础增删改查 – 新增文档

- PUT 方法：指定id
- POST 方法：自增id

```
PUT /lab/member/123
{
  "name": "Ni Gang",
  "sex":  "male",
  "birthday":  "1994/10/28"
}
POST /lab/member/
{
  "name": "Bai Chunfei",
  "sex":  "male",
  "birthday":  "1994/10/22"
}
```

冪等操作

```
{
  "_index": "lab",
  "_type": "member",
  "_id": "123",
  "_score": 1,
  "_source": {
    "name": "Ni Gang",
    "sex": "male",
    "birthday": "1994/10/28"
  }
},
```

非冪等操作

```
{
  "_index": "lab",
  "_type": "member",
  "_id": "rkBMzWYBQezJbZJP3Ft_",
  "_score": 1,
  "_source": {
    "name": "Bai Chunfei",
    "sex": "male",
    "birthday": "1994/10/22"
  }
},
{
  "_index": "lab",
  "_type": "member",
  "_id": "sEBNzWYBQezJbZJPfVuO",
  "_score": 1,
  "_source": {
    "name": "Bai Chunfei",
    "sex": "male",
    "birthday": "1994/10/22"
  }
},
{
  "_index": "lab",
  "_type": "member",
  "_id": "r0BNzWYBQezJbZJPY1vg",
  "_score": 1,
  "_source": {
    "name": "Bai Chunfei",
    "sex": "male",
    "birthday": "1994/10/22"
  }
}
```

基础增删改查 – 删除文档

- DELETE 方法
- 惰性删除：不会立即将文档从磁盘中删除，只是将文档标记为已删除状态。

```
DELETE /lab/member/123
```

```
{
  "_index": "lab",
  "_type": "member",
  "_id": "123",
  "_version": 2,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 41,
  "_primary_term": 1
}
```

第一次执行

```
{
  "_index": "lab",
  "_type": "member",
  "_id": "123",
  "_version": 3,
  "result": "not_found",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 44,
  "_primary_term": 1
}
```

第二次执行

```
{
  "_index": "lab",
  "_type": "member",
  "_id": "123",
  "_version": 4,
  "result": "not_found",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 45,
  "_primary_term": 1
}
```

第三次执行

```
{
  "_index": "lab",
  "_type": "member",
  "_id": "123",
  "_version": 1,
  "result": "not_found",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 46,
  "_primary_term": 1
}
```

等待几秒钟后执行

基础增删改查 – 更新文档

- 文档**不可变**，实质上是替换旧文档并重建索引
- 局部更新 API：
  1. 检索旧文档（retrieve）
  2. 复制旧文档并修改它（update）
  3. 删除旧文档（delete）
  4. 使用新文档重建索引（reindex）
- 并发冲突控制：
  - 乐观锁机制，基于版本进行并发控制；
  - 若其他进程在检索阶段或重建索引阶段修改了该文档，则 _version 必然不匹配，返回更新失败。

更新整个文档

```
PUT /lab/member/123
{
  "name": "Ni Gang",
  "sex":  "male",
  "birthday":  "1994/10/28",
  "school":  "WHU"
}
```

更新部分文档

```
POST /lab/member/123/_update
{
  "school":  "WHU"
}
```

使用脚本更新部分文档

```
POST /lab/member/123/_update
{
  "script" : "ctx._source.age+=1"
}
```

若文档不存在则创建

```
POST /lab/member/123/_update
{
  "script" : "ctx._source.age+=1",
  "upsert" : {
    "age" : 23
  }
}
```

基础增删改查 – 查询文档

按id查询文档

```
GET /lab/member/123
```

无条件查询文档

```
GET /lab/member/_search
```

跨索引/跨类型查询文档

```
GET /_search   GET /lab/_search
GET /lab1,lab2/member/_search
GET /lab*/member/_search
GET /lab*/doctor,master/_search
```

分页查询文档

```
GET /lab/member/_search?size=10&from=5
```

简单条件查询

```
GET /lab/member/_search
{
  "query": {
    "match": {
      "sex": "male"
    }
  }
}
```

复杂条件查询

```
GET /bookstore/books/_search
{
    "query" :  {
        "bool" : {
            "must" : {
                "match" : {
                    "writer" : "金庸"
                }
            },
            "filter": {
                "range" : {
                    "pages" : { "lte": 500 }
                }
            }
        }
    }
}
```

elastic

Elasticsearch 使用

基础增删改查 – 查询文档

按id查询文档

```
GET /lab/member/123
```

无条件查询文档

```
GET /lab/member/_search
```

跨索引/跨类型查询文档

```
GET /_search   GET /lab/_search
GET /lab1,lab2/member/_search
GET /lab*/member/_search
GET /lab*/doctor,master/_search
```

分页查询文档

```
GET /lab/member/_search?size=10&from=5
```


```
{
  "took": 74,
  "timed_out": false,
  "_shards": {⟺},
  "hits": {
    "total": 5,
    "max_score": 0.2876821,
    "hits": [
      {
        "_index": "lab",
        "_type": "member",
        "_id": "14",
        "_score": 0.2876821,
        "_source": {
          "name": "Ni Gang",
          "sex": "male",
          "birthday": "1994/10/28"
        }
      },
      {⟺},
      {⟺},
      {⟺},
      {⟺}
    ]
  }
}
```


查询耗时74ms

返回结果集大小

匹配度评分结果

搜索技巧 – 查询语言DSL

- 查询子句表达式

```
{
  QUERY_NAME: {
     ARGUMENT: VALUE,
     ARGUMENT: VALUE,
     ...
  }
}
```

- example

```
{ "match": { "first_name": "Jack" } }
```
匹配

```
{ "term": { "age" : 26 }}
```
精确查询

```
{ "terms": { "interests": [ "football", "music" ] }
```

```
{ "range": { "age" : { "gt" : 30, "lte" : 50 } }
```

```
{ "geo_distance_range" : {
    "gte" : "1km",
    "lt" : "2km",
    "location": { "lat":  40.715, "lon": -73.988 }
  }
}
```
范围查询

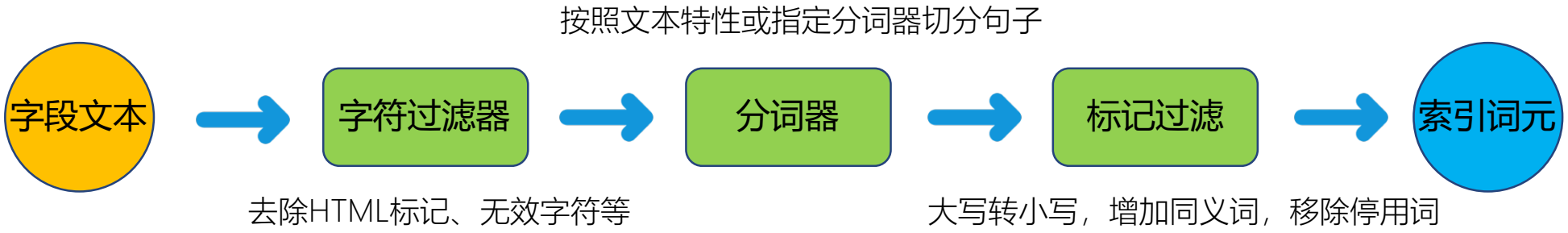
合并查询子句

```
{
    "bool": {
        "must":     { "match": { "first_name": "Jack" }},
        "must_not": { "match": { "last_name":  "mary" }},
        "should":   { "match": { "tweet": "full text" }},
        "filter":   { "range": { "age" : { "gt" : 30 }} }
    }
}
```

对比传统关系型数据库

- must ~ AND
- must_not ~ AND NOT
- should ~ OR
- filter ~ GROUP BY HAVING

全文搜索 – 人类语言处理



原文本	"Set the shape to semi-transparent by calling set_trans(5)"
标准分析器	{set, the, shape, to, semi, transparent, by, calling, set_trans, 5}
英语分析器	{set, shape, semi, transpar, call, set_tran, 5}

全文搜索 – 结果匹配度计算

类TF-IDF算法确定词权重：

- 关键词出现频率越高，匹配度越高（TF）
- 逆文档词频（IDF）
- 字段长度准则：长度越长，匹配度越低

$$词频(TF) = \frac{某个词在文章中的出现次数}{文章的总词数}$$

$$逆文档频率(IDF) = \log(\frac{语料库的文档总数}{包含该词的文档数 + 1})$$

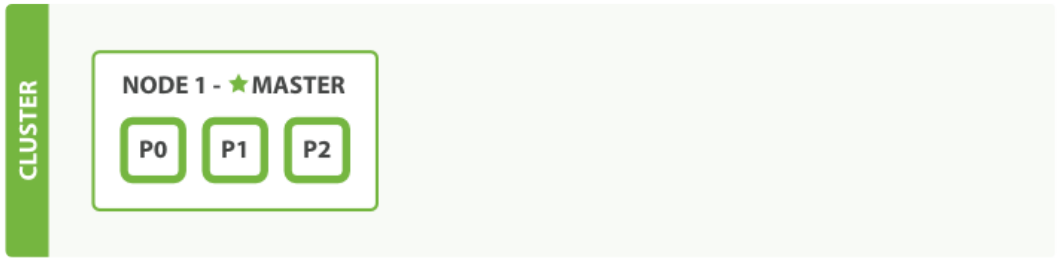
$$TF - IDF = 词频(TF) \times 逆文档频率(IDF)$$

文档编号	文档数据	
	id	title
0	1	谷歌地图之父跳槽Facebook
1	2	谷歌地图之父加盟Facebook
2	3	谷歌地图创始人拉斯离开谷歌加盟Facebook
3	4	谷歌地图之父跳槽Facebook 与wave项目取消有关
4	5	谷歌地图之父拉斯加盟社交网站Facebook

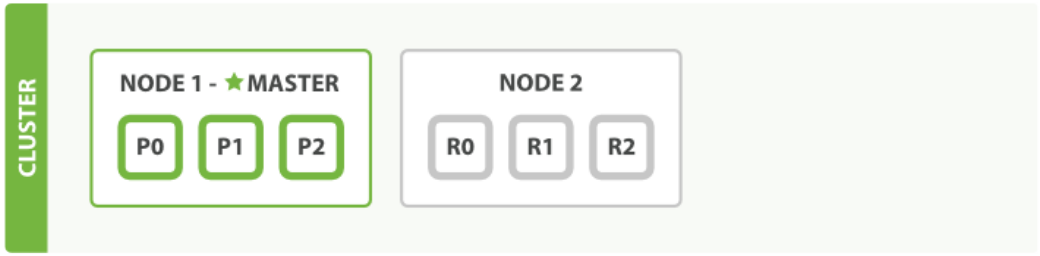
单词ID	单词	倒排列表（包含该单词的文档ID，DocID）
1	谷歌	0, 1, 2, 3, 4
2	地图	0, 1, 2, 3, 4
3	之父	0, 1, 3, 4
4	跳槽	0, 3
5	Facebook	0, 1, 2, 3, 4
6	加盟	1, 2, 4
7	创始人	2
8	拉斯	2, 4
9	离开	2
10	与	3
11	wave	3
12	项目	3
13	取消	3
14	有关	3
15	社交	4
16	网站	4

ES集群概念 - 集群、节点、分片

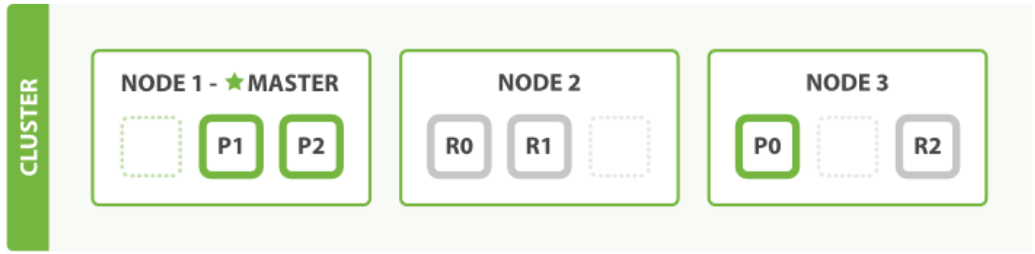
- 分片 (Shard): 分片就是一个Lucene实例，是一个完整的搜索引擎，一个索引（逻辑概念）可能对应一个或多个分片（物理概念）；
- 节点 (Node): 节点是一个运行中的 Elasticsearch 实例；一个节点会包含多个分片；
- 集群 (Cluster): 多个节点组成一个集群，并由主节点（Master）来管理整个集群。



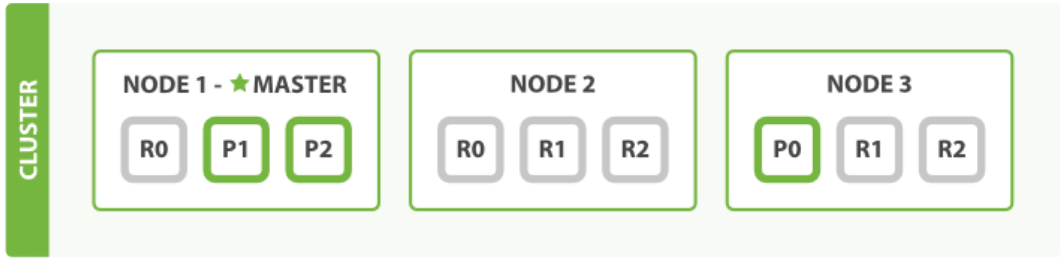
单节点集群



双节点集群



负载均衡-三节点集群



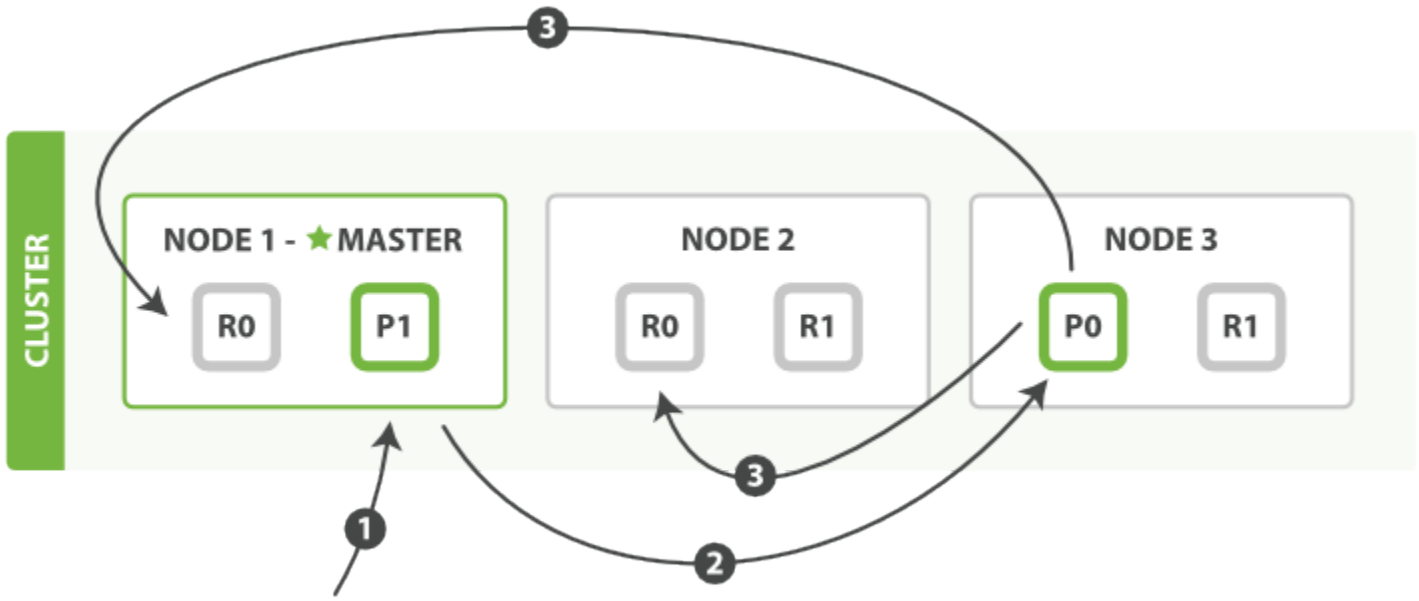
双复制分片-三节点集群

分布式增删改查

- 路由方式：shard = hash(routing) % number_of_primary_shards
- 负载均衡：每个节点都有能力处理任意请求。每个节点都知道任意文档所在的节点，每个节点都可以将请求转发到需要的节点。

分布式增删改

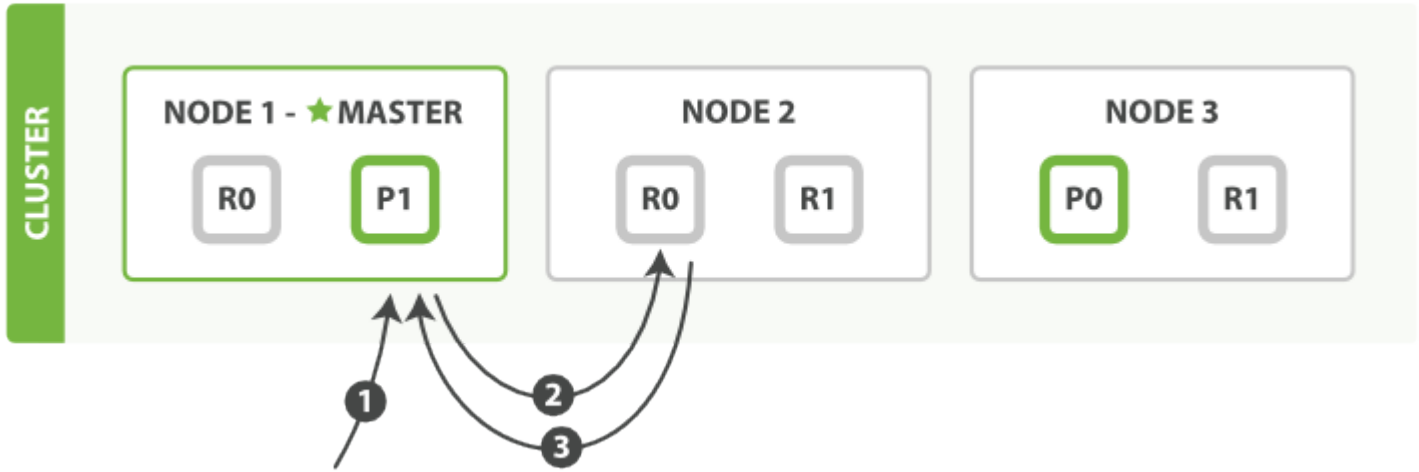
1. 客户端向 Node 1 发送新建、修改或者删除请求。
2. Node 1 根据文档的 _id 确定文档属于分片 0 。请求会被转发到 Node 3（主分片P0 目前被分配在 Node 3 上）
3. Node 3 在主分片上面执行请求。若成功则将请求并转发到 Node 1 和 Node 2 的副本分片上。
4. 一旦所有的副本分片都报告成功，Node 3 将向原请求节点 Node 1 报告成功，Node 1 向客户端报告成功。



分布式增删改查

分布式查询

1. 客户端向 Node 1 发送获取请求。
2. 节点使用文档的 _id 来确定文档属于分片 0 。分片 0 的三个节点上都存在。 在这种情况下，它将请求转发到 Node 2 。
3. Node 2 将文档返回给 Node 1 ， 然后将文档返回给客户端。

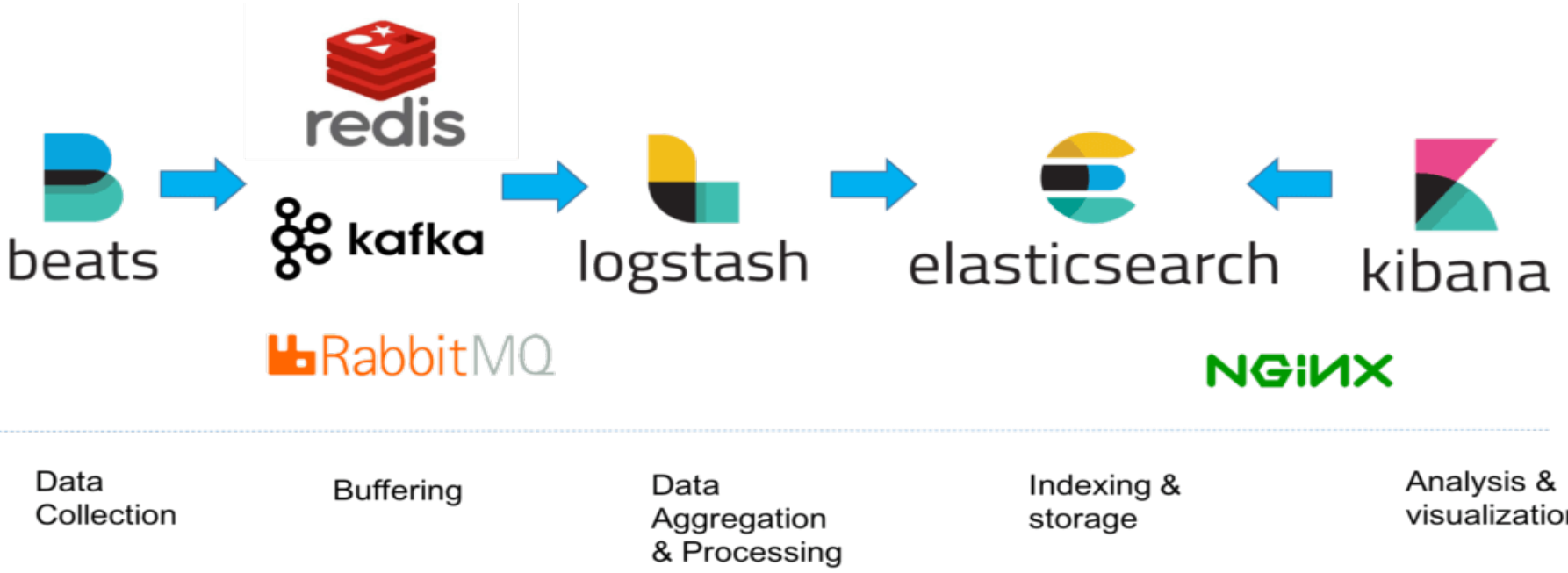


通过轮询所有的副本分片来达到负载均衡

elastic

相关拓展：ELK Stack

- Elasticsearch：分布式搜索和分析引擎，作为应用的基础搜索引擎，使其具有复杂的搜索功能；
- Logstash：数据收集引擎。
- Kibana：数据分析和可视化平台。
- Filebeat：轻量级开源日志文件数据搜集器



参考
Conference

Elasticsearch 文档 & 源码：

- https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
- https://github.com/elastic/elasticsearch

Apache Lucene 文档 & 源码：

- https://github.com/apache/lucene-solr
- http://lucene.apache.org/core/documentation.html

THANKS