主讲老师：Fox

```
有道云笔记
文档：4. ElasticSearch集群架构实战及其原理...
链接：http://note.youdao.com/noteshare?id=16ca3fcfcdda46a976cfd978e20df4be&sub=C9A58F2702FD42EEA06F413486167F65
```

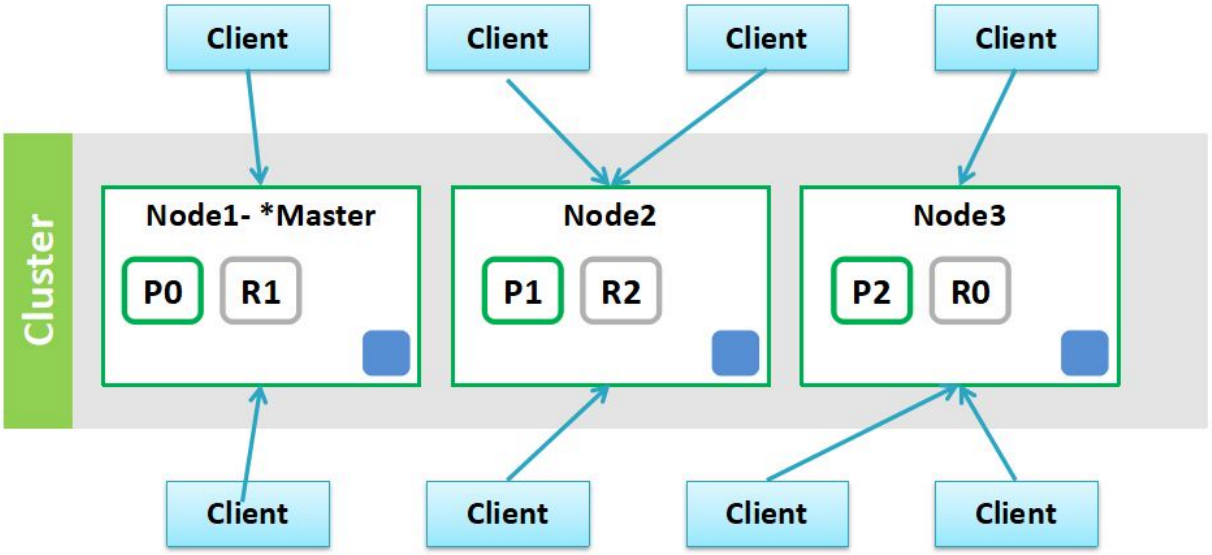
# ES集群架构

分布式系统的可用性与扩展性

- 高可用性
  - 服务可用性-允许有节点停止服务
  - 数据可用性-部分节点丢失，不会丢失数据
- 可扩展性
  - 请求量提升/数据的不断增长(将数据分布到所有节点上)

ES集群架构的优势：

- 提高系统的可用性，部分节点停止服务，整个集群的服务不受影响
- 存储的水平扩容




## 核心概念

### 集群

- 一个集群可以有一个或者多个节点
- 不同的集群通过不同的名字来区分，默认名字“elasticsearch“
- 通过配置文件修改，或者在命令行中 -E cluster.name=es-cluster进行设定

### 节点

- 节点是一个Elasticsearch的实例
  - 本质上就是一个JAVA进程
  - 一台机器上可以运行多个Elasticsearch进程，但是生产环境一般建议一台机器上只运行一个Elasticsearch实例
- 每一个节点都有名字，通过配置文件配置，或者启动时候 -E node.name=node1指定
- 每一个节点在启动之后，会分配一个UID，保存在data目录下

### 节点类型

- Master Node：主节点
- Master eligible nodes：可以参与选举的合格节点
- Data Node：数据节点
- Coordinating Node：协调节点
- 其他节点

| 节点类型 | 配置参数 | 默认值 |
| --- | --- | --- |
| maste eligible | node.master | true |
| data | node.data | true |
| ingest | node.ingest | ture |
| coordinating only | 无 | 每个节点默认都是 coordinating 节点。设置其他类型全部为false, |
| machine learning | node.ml | true (需 enable x-pack) |

**Master eligible nodes和Master Node**

- 每个节点启动后，默认就是一个Master eligible节点
  - 可以设置 node.master: false禁止
- Master-eligible节点可以参加选主流程，成为Master节点
- 当第一个节点启动时候，它会将自己选举成Master节点
- 每个节点上都保存了集群的状态，只有Master节点才能修改集群的状态信息
  - 集群状态(Cluster State)，维护了一个集群中，必要的信息
    - 所有的节点信息
    - 所有的索引和其相关的Mapping与Setting信息
    - 分片的路由信息

**Master Node的职责**

- 处理创建，删除索引等请求，负责索引的创建与删除
- 决定分片被分配到哪个节点
- 维护并且更新Cluster State

**Master Node的最佳实践**

- Master节点非常重要，在部署上需要考虑解决单点的问题
- 为一个集群设置多个Master节点，每个节点只承担Master 的单一角色

**选主的过程**

- 互相Ping对方，Node Id 低的会成为被选举的节点
- 其他节点会加入集群，但是不承担Master节点的角色。一旦发现被选中的主节点丢失，就会选举出新的Master节点

**Data Node & Coordinating Node**

- Data Node
    - 可以保存数据的节点，叫做Data Node，负责保存分片数据。在数据扩展上起到了

至关重要的作用

    - 节点启动后，默认就是数据节点。可以设置node.data: false禁止
    - 由Master Node决定如何把分片分发到数据节点上
    - 通过增加数据节点可以解决数据水平扩展和解决数据单点问题

- Coordinating Node
    - 负责接受Client的请求， 将请求分发到合适的节点，最终把结果汇集到一起
    - 每个节点默认都起到了Coordinating Node的职责

**其他节点类型**

- Hot & Warm Node
    - 不同硬件配置 的Data Node,用来实现Hot & Warm架构，降低集群部署的成本
- Ingest Node
    - 数据前置处理转换节点，支持pipeline管道设置，可以使用ingest对数据进行过滤、转换等操作
- Machine Learning Node
    - 负责跑机器学习的Job，用来做异常检测
- Tribe Node
    - Tribe Node连接到不同的Elasticsearch集群，并且支持将这些集群当成一个单独的集群处理

**分片(Primary Shard & Replica Shard)**

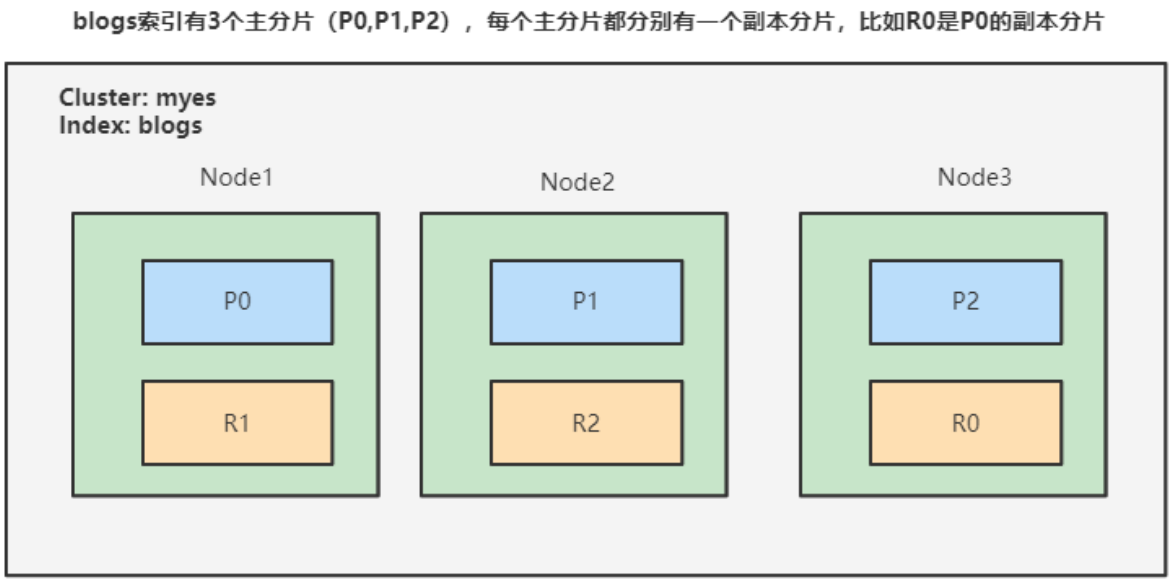
- 主分片（Primary Shard）

- 用以解决数据水平扩展的问题。通过主分片，可以将数据分布到集群内的所有节点之上
- 一个分片是一个运行的Lucene的实例
- 主分片数在索引创建时指定，后续不允许修改，除非Reindex

- 副本分片（Replica Shard）
  - 用以解决数据高可用的问题。 副本分片是主分片的拷贝
  - 副本分片数，可以动态调整
  - 增加副本数，还可以在一定程度上提高服务的可用性(读取的吞吐)

```
# 指定索引的主分片和副本分片数
PUT /blogs
{
 "settings": {
 "number_of_shards": 3,
 "number_of_replicas": 1
 }
}
```

blogs对应的架构




思考：增加一个节点或改大主分片数对系统有什么影响?

**分片的设定**

对于生产环境中分片的设定，需要提前做好容量规划

- 分片数设置过小
    - 导致后续无法增加节点实现水平扩展
    - 单个分片的数据量太大，导致数据重新分配耗时
- 分片数设置过大，7.0 开始，默认主分片设置成1，解决了over-sharding（分片过度）的问题
    - 影响搜索结果的相关性打分，影响统计结果的准确性
    - 单个节点上过多的分片，会导致资源浪费，同时也会影响性能

```
#查看集群的健康状况
GET _cluster/health
```

**集群status**

- Green: 主分片与副本都正常分配
- Yellow: 主分片全部正常分配，有副本分片未能正常分配
- Red: 有主分片未能分配。例如，当服务器的磁盘容量超过85%时,去创建了一个新的索引

**CAT API查看集群信息：**

```

GET /_cat/nodes?v #查看节点信息
GET /_cat/health?v #查看集群当前状态：红、黄、绿
GET /_cat/shards?v #查看各shard的详细情况
GET /_cat/shards/{index}?v #查看指定分片的详细情况
GET /_cat/master?v #查看master节点信息
GET /_cat/indices?v #查看集群中所有index的详细信息
GET /_cat/indices/{index}?v #查看集群中指定index的详细信息
```

# 搭建三节点ES集群

**系统环境**

操作系统: CentOS7，准备用户es

elasticsearch：elasticsearch-7.17.3

切换到root用户，修改/etc/hosts

```
vim /etc/hosts
192.168.65.174 es-node1
192.168.65.192 es-node2
192.168.65.204 es-node3
```

**修改elasticsearch.yml**

```
# 指定集群名称3个节点必须一致
cluster.name: es-cluster
#指定节点名称，每个节点名字唯一
node.name: node-1
#是否有资格为master节点，默认为true
node.master: true
#是否为data节点，默认为true
node.data: true
# 绑定ip,开启远程访问,可以配置0.0.0.0
 network.host: 0.0.0.0
 #指定web端口
 #http.port: 9200
 #指定tcp端口
 #transport.tcp.port: 9300
 #用于节点发现
 discovery.seed_hosts: ["es-node1", "es-node2", "es-node3"]
 #7.0新引入的配置项,初始仲裁，仅在整个集群首次启动时才需要初始仲裁。
 #该选项配置为node.name的值，指定可以初始化集群节点的名称
 cluster.initial_master_nodes: ["node-1","node-2","node-3"]
 #解决跨域问题
 http.cors.enabled: true
 http.cors.allow-origin: "*"
```

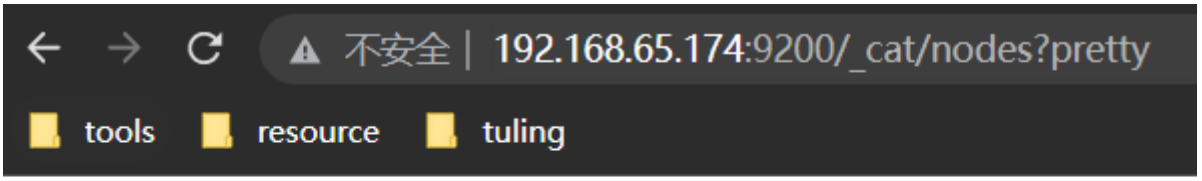
三个节点配置如下：

```
#192.168.65.174的配置
cluster.name: es-cluster
node.name: node-1
node.master: true
node.data: true
network.host: 0.0.0.0
discovery.seed_hosts: ["es-node1", "es-node2", "es-node3"]
cluster.initial_master_nodes: ["node-1","node-2","node-3"]
http.cors.enabled: true
 http.cors.allow-origin: "*"

 #192.168.65.192的配置
 cluster.name: es-cluster
 node.name: node-3
 node.master: true
 node.data: true
```

```
network.host: 0.0.0.0
discovery.seed_hosts: ["es-node1", "es-node2", "es-node3"]
cluster.initial_master_nodes: ["node-1","node-2","node-3"]
http.cors.enabled: true
http.cors.allow-origin: "*"

#192.168.65.204的配置
cluster.name: es-cluster
node.name: node-2
node.master: true
node.data: true
network.host: 0.0.0.0
discovery.seed_hosts: ["es-node1", "es-node2", "es-node3"]
cluster.initial_master_nodes: ["node-1","node-2","node-3"]
http.cors.enabled: true
http.cors.allow-origin: "*"
```

验证集群

http://192.168.65.174:9200/_cat/nodes?pretty




```
192.168.65.204 13 50 59 24.33 24.45 24.85 cdfhilmrstw - node-2
192.168.65.174  6 98 59 12.47 17.79 21.78 cdfhilmrstw * node-1
192.168.65.192 31 55 58 13.29 18.73 22.11 cdfhilmrstw - node-3
```

**安装Cerebro客户端**

**Cerebro介绍**

Cerebro 可以查看分片分配和通过图形界面执行常见的索引操作。 完全开源，并且它允许添加用户，密码或 LDAP 身份验证问网络界面。

Cerebro 基于 Scala 的Play 框架编写，用于后端 REST 和 Elasticsearch 通信。 它使用通过 AngularJS 编写的单页应用程序（SPA）前端。

项目网址：https://github.com/lmenezes/cerebro

**安装 Cerebro**

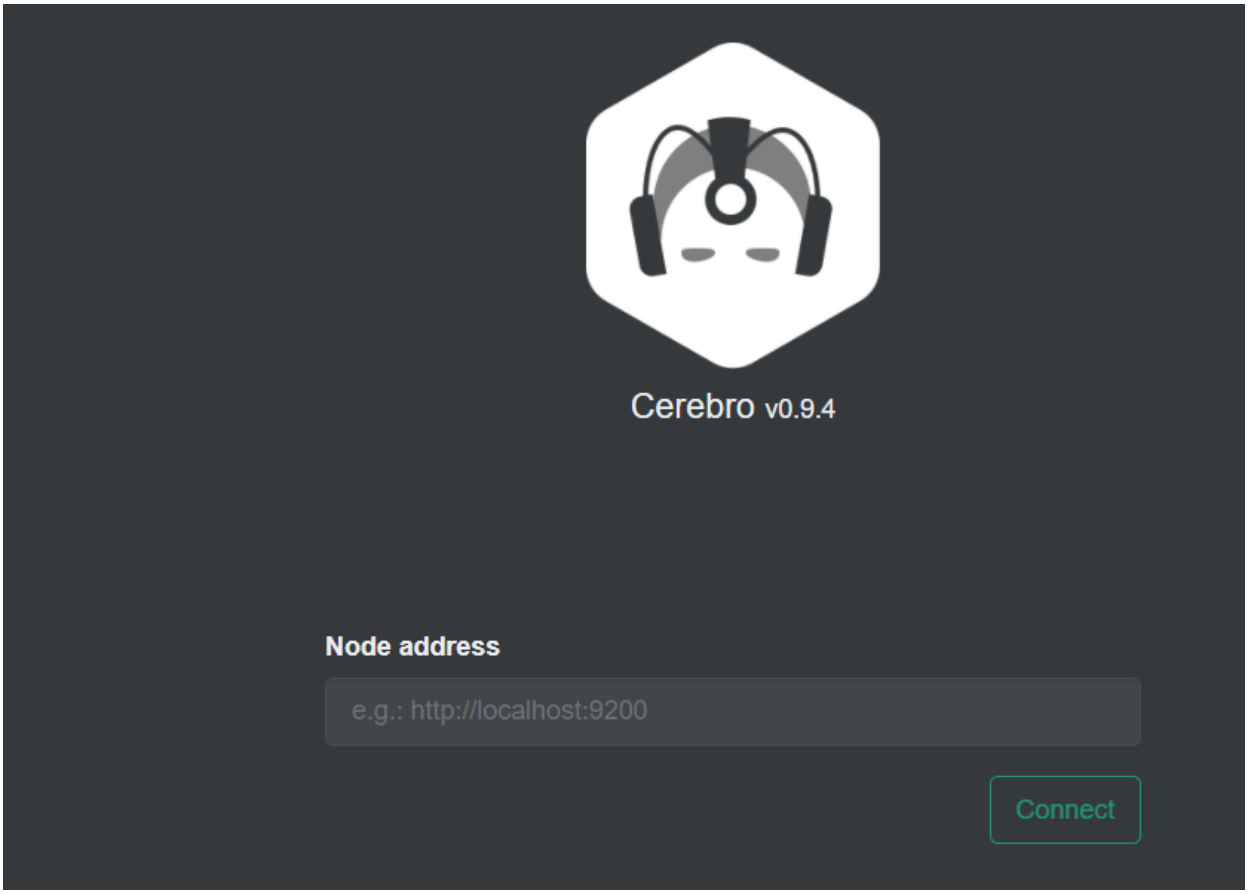
下载地址：

https://github.com/lmenezes/cerebro/releases/download/v0.9.4/cerebro-0.9.4.zip

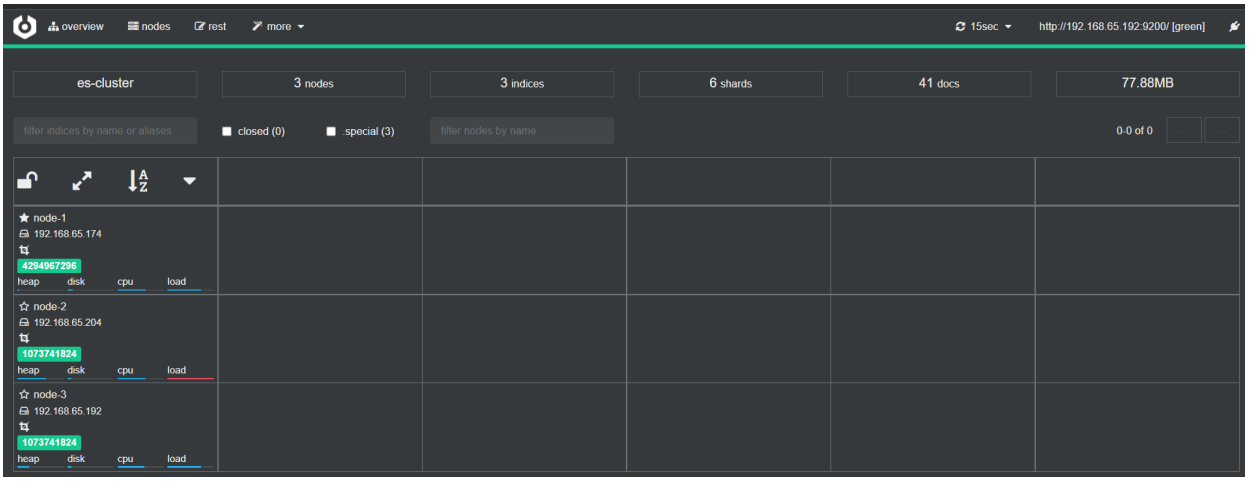
## 运行 cerebro

```
cerebro-0.9.4/bin/cerebro

#后台启动
nohup bin/cerebro > cerebro.log &
```

访问：http://192.168.65.174:9000/




输入ES集群节点：http://192.168.65.192:9200，建立连接：




## 安装kibana

修改kibana配置

```
vim config/kibana.yml
```

```
2
3 server.port: 5601
4 server.host: "192.168.65.174"
5 elasticsearch.hosts: ["http://192.168.65.174:9200","http://192.168.65.19
2:9200","http://192.168.65.204:9200"]
6 i18n.locale: "zh-CN"
```

**运行Kibana**

提示：Kibana对外的 tcp 端口是5601，使用netstat -tunlp|grep 5601即可查看进程

```
1 #后台启动
2 nohup bin/kibana &
```

访问Kibana: http://192.168.65.174:5601/

## ES安全认证

参考文档:

https://www.elastic.co/guide/en/elasticsearch/reference/7.17/configuring-stack-security.html

**ES敏感信息泄露的原因**

- Elasticsearch在默认安装后，不提供任何形式的安全防护
- 不合理的配置导致公网可以访问ES集群。比如在elasticsearch.yml文件中,server.host配置为0.0.0.0

**免费的方案**

- 设置nginx反向代理
- 安装免费的Security插件
  - Search Guard： https://search-guard.com/
  - readonlyrest： https://readonlyrest.com/
- X-Pack的Basic版
  - 从ES 6.8开始，Security纳入x-pack的Basic版本中，免费使用一些基本的功能

**集群内部安全通信**

ElasticSearch集群内部的数据是通过9300进行传输的，如果不对数据加密，可能会造成数据被抓包，敏感信息泄露。

解决方案： 为节点创建证书

TLS 协议要求Trusted Certificate Authority (CA) 签发x.509的证书。证书认证的不同级别：

- Certificate ——节点加入需要使用相同CA签发的证书
- Full Verification——节点加入集群需要相同CA签发的证书，还需要验证Host name 或IP地址
- No Verification——任何节点都可以加入，开发环境中用于诊断目的

### 1）生成节点证书

```
# 为集群创建一个证书颁发机构
bin/elasticsearch-certutil ca
# 为集群中的每个节点生成证书和私钥
bin/elasticsearch-certutil cert --ca elastic-stack-ca.p12
# 移动到config目录下
mv *.p12 config/
```

将如上命令生成的两个证书文件拷贝到另外两个节点作为通信依据。

```
# 拷贝到192.168.65.192
scp *.p12 es@192.168.65.192:/home/es/elasticsearch-7.17.3/config
```

### 2）配置节点间通信

三个ES节点增加如下配置：

```
## elasticsearch.yml 配置
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: certificate
xpack.security.transport.ssl.client_authentication: required
xpack.security.transport.ssl.keystore.path: elastic-certificates.p12
xpack.security.transport.ssl.truststore.path: elastic-certificates.p12
```

### 开启并配置X-Pack的认证

1）修改elasticsearch.yml配置文件，开启xpack认证机制

```
xpack.security.enabled: true # 开启xpack认证机制
```

测试：

```
#使用Curl访问ES，返回401错误
curl 'localhost:9200/_cat/nodes?pretty'
```

```
[es@redis elasticsearch-7.17.3]$ curl 'localhost:9200/_cat/nodes?pretty'
{
  "error" : {
    "root_cause" : [
      {
        "type" : "security_exception",
        "reason" : "missing authentication credentials for REST request [/_cat/nodes?pretty]",
        "header" : {
          "WWW-Authenticate" : "Basic realm=\"security\" charset=\"UTF-8\""
        }
      }
    ],
    "type" : "security_exception",
    "reason" : "missing authentication credentials for REST request [/_cat/nodes?pretty]",
    "header" : {
      "WWW-Authenticate" : "Basic realm=\"security\" charset=\"UTF-8\""
    }
  },
  "status" : 401
}
```

浏览器访问http://192.168.65.174:9200/需要输入用户名密码


2）为内置账号添加密码

ES中内置了几个管理其他集成组件的账号即：

apm_system, beats_system, elastic, kibana,

logstash_system, remote_monitoring_user，使用之前，首先需要添加一下密码。

```
bin/elasticsearch-setup-passwords interactive
```

- interactive：给用户手动设置密码。
- auto：自动生成密码。

```
[es@redis elasticsearch-7.17.3]$ bin/elasticsearch-setup-passwords interactive
Initiating the setup of passwords for reserved users elastic,apm_system,kibana,kibana_system,logstash_system,beats_system,remote_monitoring_user.
You will be prompted to enter passwords as the process progresses.
Please confirm that you would like to continue [y/N]y


Enter password for [elastic]:                    设置密码123456
Reenter password for [elastic]:
Enter password for [apm_system]:
Reenter password for [apm_system]:
Enter password for [kibana_system]:
Reenter password for [kibana_system]:
Enter password for [logstash_system]:
Reenter password for [logstash_system]:
Enter password for [beats_system]:
Reenter password for [beats_system]:
Enter password for [remote_monitoring_user]:
Reenter password for [remote_monitoring_user]:
Changed password for user [apm_system]
Changed password for user [kibana_system]
Changed password for user [kibana]
Changed password for user [logstash_system]
Changed password for user [beats_system]
Changed password for user [remote_monitoring_user]
Changed password for user [elastic]
```

测试

```
curl -u elastic 'localhost:9200/_cat/nodes?pretty'
```

```
[es@redis elasticsearch-7.17.3]$ curl -u elastic 'localhost:9200/_cat/nodes?pretty'
Enter host password for user 'elastic':
192.168.3.100 42 94 4 0.16 0.16 0.21 cdfhilmrstw * redis
```

3）配置Kibana

开启了安全认证之后，kibana连接es以及访问es都需要认证。

修改kibana.yml

```
elasticsearch.username: "kibana_system"
elasticsearch.password: "123456"
```

启动kibana服务

```
nohup bin/kibana &
```

4）配置cerebro

修改配置文件

```
vim conf/application.conf

hosts = [
 {
 host = "http://192.168.65.174:9200"
 name = "es-cluster"
 auth = {
 username = "elastic"
 password = "123456"
  }
  }
 ]

```

```
# A list of known hosts
hosts = [
  #{
  #  host = "http://192.168.65.174:9200"
  #  name = "Localhost cluster"
  #  headers-whitelist = [ "x-proxy-user", "x-proxy-roles", "X-Forwarded-For
  #}
  # Example of host with authentication
  {
    host = "http://192.168.65.174:9200"
    name = "es-cluster"
    auth = {
      username = "elastic"
      password = "123456"
    }
  }
]
```

启动cerebro服务

```
nohup bin/cerebro > cerebro.log &
```

## 生产环境常见集群部署方式

不同角色的节点：Master eligible / Data / Ingest / Coordinating /Machine Learning

在开发环境中，一个节点可承担多种角色。

在生产环境中：

- 根据数据量，写入和查询的吞吐量，选择合适的部署方式
- 建议设置单一角色的节点

| 节点类型 | 配置参数 | 默认值 |
| --- | --- | --- |
| maste eligible | node.master | true |
| data | node.data | true |
| ingest | node.ingest | ture |
| coordinating only | 无 | 每个节点默认都是 coordinating 节点。<br>设置其他类型全部为false, |
| machine learning | node.ml | true (需 enable x-pack) |

一个节点只承担一个角色的配置

```
#Master节点
node.master: true
node.ingest: false
node.data: false

#data节点
node.master: false
```

```
node.ingest: false
node.data: true

#ingest 节点
node.master: false
node.ingest: true
node.data: false

#coordinate节点
node.master: false
node.ingest: false
node.data: false
```

这种单一角色职责分离的好处：

- 单一 master eligible nodes：负责集群状态(cluster state)的管理
  - 使用低配置的CPU,RAM和磁盘
- 单一 data nodes: 负责数据存储及处理客户端请求
  - 使用高配置的CPU,RAM和磁盘
- 单一ingest nodes: 负责数据处理
  - 使用高配置CPU; 中等配置的RAM; 低配置的磁盘
- 单一Coordinating Only Nodes(Client Node)
  - 使用高配置CPU; 高配置的RAM; 低配置的磁盘

生产环境中，建议为一些大的集群配置Coordinating Only Nodes

- 扮演Load Balancers，降低Master和 Data Nodes的负载
- 负责搜索结果的Gather/Reduce
- 有时候无法预知客户端会发送怎么样的请求。比如大量占用内存的操作，一个深度聚合可能会引发OOM

**单一 master eligible nodes**

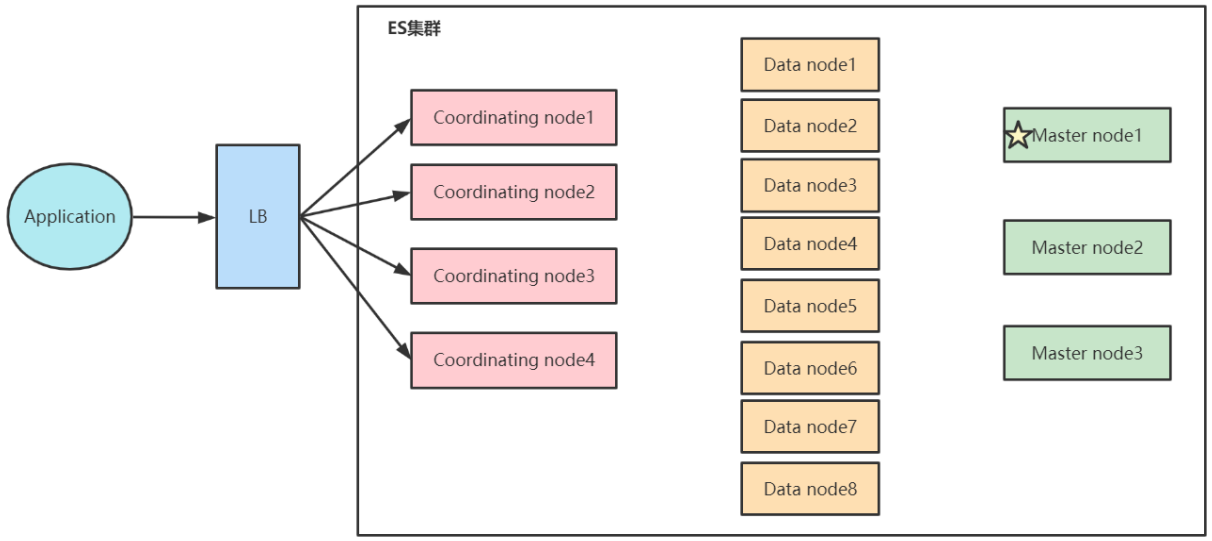
从高可用&避免脑裂的角度出发：

- 一般在生产环境中配置3台
- 一个集群只有1台活跃的主节点（master node）
  - 负责分片管理，索引创建，集群管理等操作
- 如果和数据节点或者Coordinate节点混合部署
  - 数据节点相对有比较大的内存占用
  - Coordinate节点有时候可能会有开销很高的查询，导致OOM

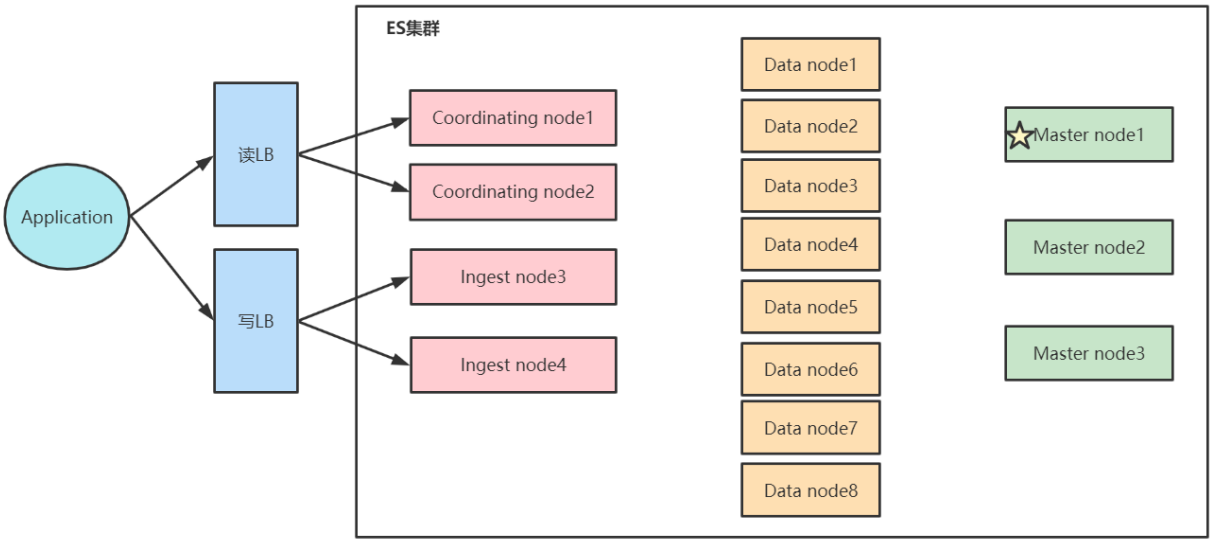
- 这些都有可能影响Master节点，导致集群的不稳定

**增加节点水平扩展场景**

- 当磁盘容量无法满足需求时，可以增加数据节点；
- 磁盘读写压力大时，增加数据节点
- 当系统中有大量的复杂查询及聚合时候，增加Coordinating节点，增加查询的性能

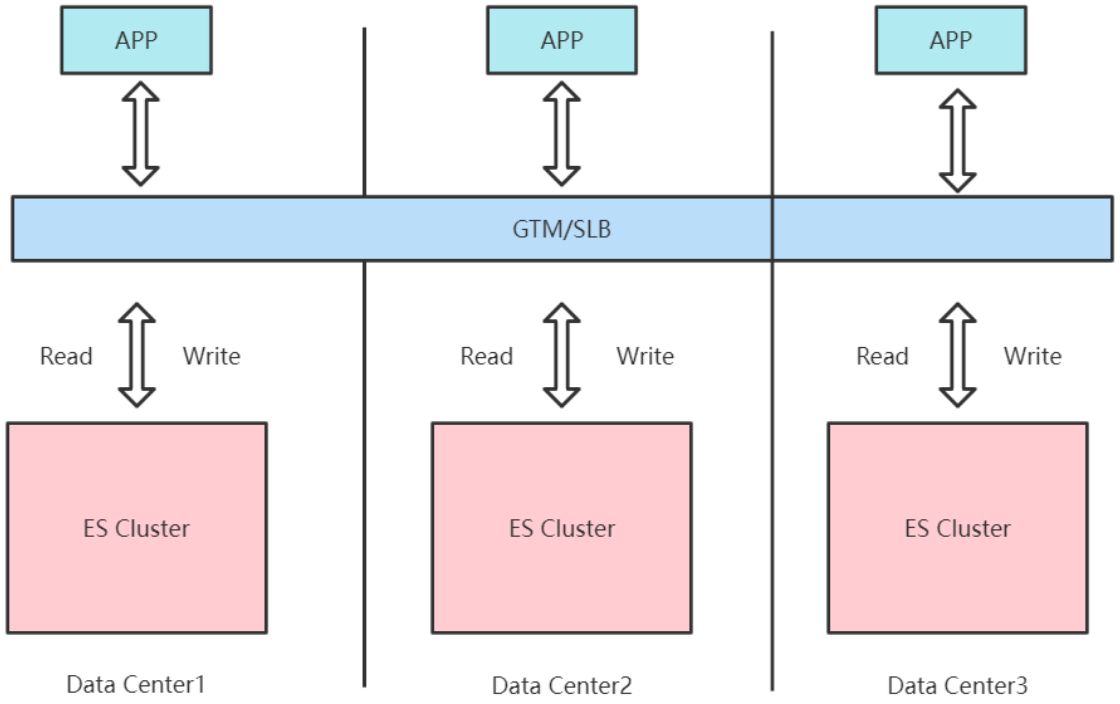



**读写分离架构**




**异地多活架构**

集群处在三个数据中心，数据三写，GTM分发读请求




> 全局流量管理（GTM）和负载均衡（SLB）的区别：
> GTM 是通过DNS将域名解析到多个IP地址，不同用户访问不同的IP地址，来实现应用服务流量的分配。同时通过健康检查动态更新DNS解析IP列表，实现故障隔离以及故障切换。最终用户的访问直接连接服务的IP地址，并不通过GTM。而 SLB 是通过代理用户访问请求的形式将用户访问请求实时分发到不同的服务器，最终用户的访问流量必须要经过SLB。 一般来说，相同Region使用SLB进行负载均衡，不同region的多个SLB地址时，则可以使用GTM进行负载均衡。

ES 跨集群复制 （Cross-Cluster Replication）是ES 6.7的的一个全局高可用特性。CCR允许不同的索引复制到一个或多个ES 集群中。
https://www.elastic.co/guide/en/elasticsearch/reference/7.17/ccr-apis.html

**Hot & Warm 架构**

为什么要设计Hot & Warm 架构?

- ES数据通常不会有 Update操作;
- 适用于Time based索引数据，同时数据量比较大的场景。
- 引入 Warm节点，低配置大容量的机器存放老数据，以降低部署成本

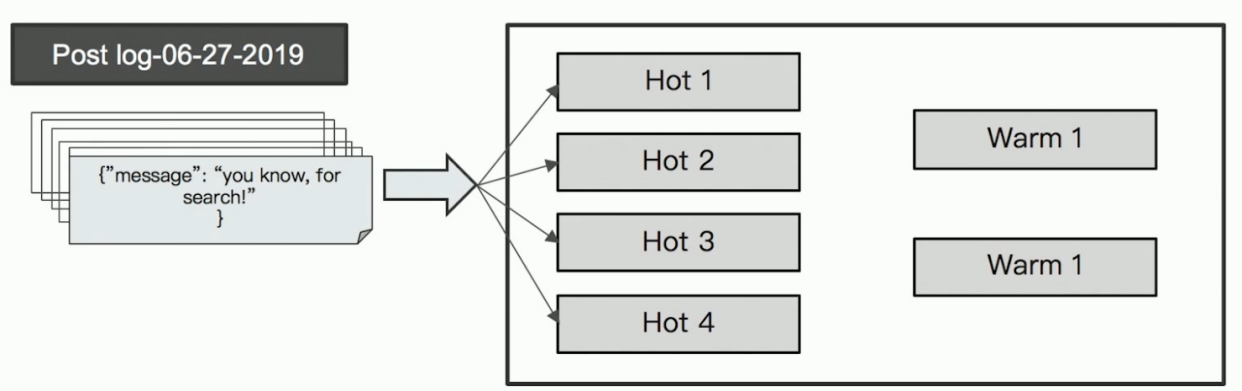
两类数据节点，不同的硬件配置：

- Hot节点(通常使用SSD)：索引不断有新文档写入。
- Warm 节点（通常使用HDD)：索引不存在新数据的写入，同时也不存在大量的数据查询

**Hot Nodes**

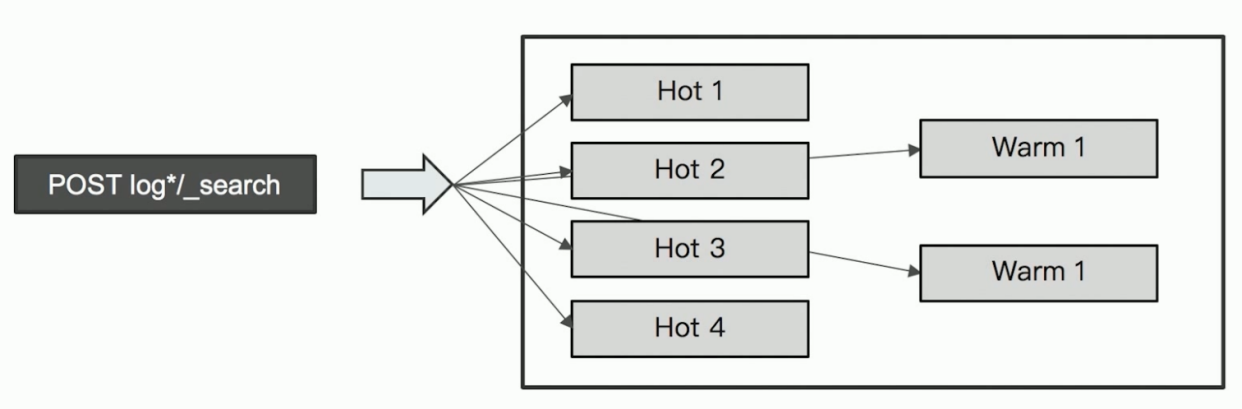
用于数据的写入：

- Indexing 对 CPU和IO都有很高的要求，所以需要使用高配置的机器
- 存储的性能要好，建议使用SSD




**Warm Nodes**

用于保存只读的索引，比较旧的数据。通常使用大容量的磁盘




**配置Hot & Warm 架构**

**使用Shard Filtering实现Hot&Warm node间的数据迁移**

- node.attr来指定node属性：hot或是warm。
- 在index的settings里通过index.routing.allocation来指定索引（index)到一个满足要求的node

| 设置 | 分配索引到节点，节点的属性规则 |
| --- | --- |
| index.routing.allocation.include.{attr} | 至少包含一个值 |
| index.routina.allocation.exclude.{attr} | 不能包含任何一个值 |
| index.routina.allocation.require. {attr} | 所有值都需要包含 |

使用 Shard Filtering，步骤分为以下几步:

- 标记节点(Tagging)
- 配置索引到Hot Node
- 配置索引到 Warm节点

**1) 标记节点**

需要通过“node.attr”来标记一个节点

- 节点的attribute可以是任何的key/value
- 可以通过elasticsearch.yml 或者通过-E命令指定

```
# 标记一个 Hot 节点
elasticsearch.bat -E node.name=hotnode -E cluster.name=tulingESCluster -E http.port=9200 -E path.data=hot_data -E node.attr.my_node_type=hot

# 标记一个 warm 节点
elasticsearch.bat -E node.name=warmnode -E cluster.name=tulingESCluster -E http.port=9201 -E path.data=warm_data -E node.attr.my_node_type=warm

# 查看节点
GET /_cat/nodeattrs?v
```

```
node     host      ip        attr               value
warmnode 127.0.0.1 127.0.0.1 ml.machine_memory 17019777024
warmnode 127.0.0.1 127.0.0.1 ml.max_open_jobs  512
warmnode 127.0.0.1 127.0.0.1 xpack.installed   true
warmnode 127.0.0.1 127.0.0.1 ml.max_jvm_size   1073741824
warmnode 127.0.0.1 127.0.0.1 my_node_type      warm
warmnode 127.0.0.1 127.0.0.1 transform.node    true
hotnode  127.0.0.1 127.0.0.1 ml.machine_memory 17019777024
hotnode  127.0.0.1 127.0.0.1 xpack.installed   true
hotnode  127.0.0.1 127.0.0.1 my_node_type      hot
hotnode  127.0.0.1 127.0.0.1 transform.node    true
hotnode  127.0.0.1 127.0.0.1 ml.max_open_jobs  512
hotnode  127.0.0.1 127.0.0.1 ml.max_jvm_size   1073741824
```

**2）配置Hot数据**

创建索引时候，指定将其创建在hot节点上

```
# 配置到 Hot节点
PUT /index-2022-05
{
 "settings":{
 "number_of_shards":2,
 "number_of_replicas":0,
 "index.routing.allocation.require.my_node_type":"hot"
 }
}

POST /index-2022-05/_doc
{
```

```
 "create_time":"2022-05-27"
}

#查看索引文档的分布
GET _cat/shards/index-2022-05?v
```

```
index          shard prirep state   docs store ip        node
index-2022-05 1     p      STARTED    1 3.3kb 127.0.0.1 hotnode
index-2022-05 0     p      STARTED    0  226b 127.0.0.1 hotnode
```

**3）旧数据移动到Warm节点**

Index.routing.allocation是一个索引级的dynamic setting,可以通过API在后期进行设定

```
# 配置到 warm 节点
PUT /index-2022-05/_settings
{
 "index.routing.allocation.require.my_node_type":"warm"
}
GET _cat/shards/index-2022-05?v
```

```
index          shard prirep state   docs store ip        node
index-2022-05 1     p      STARTED    1 3.3kb 127.0.0.1 warmnode
index-2022-05 0     p      STARTED    0  226b 127.0.0.1 warmnode
```

**如何对集群的容量进行规划**

一个集群总共需要多少个节点?一个索引需要设置几个分片？规划上需要保持一定的余量，当负载出现波动，节点出现丢失时，还能正常运行。

做容量规划时，一些需要考虑的因素：

- 机器的软硬件配置
- 单条文档的大小 | 文档的总数据量 | 索引的总数据量（(Time base数据保留的时间)|副本分片数
- 文档是如何写入的(Bulk的大小)
- 文档的复杂度，文档是如何进行读取的(怎么样的查询和聚合)

评估业务的性能需求：

- 数据吞吐及性能需求
  - 数据写入的吞吐量，每秒要求写入多少数据?
  - 查询的吞吐量?

- 单条查询可接受的最大返回时间?
- 了解你的数据
  - 数据的格式和数据的Mapping
  - 实际的查询和聚合长的是什么样的

ES集群常见应用场景:

- 搜索: 固定大小的数据集
  - 搜索的数据集增长相对比较缓慢
- 日志: 基于时间序列的数据
  - 使用ES存放日志与性能指标。数据每天不断写入，增长速度较快
  - 结合Warm Node 做数据的老化处理

硬件配置:

- 选择合理的硬件，数据节点尽可能使用SSD
- 搜索等性能要求高的场景，建议SSD
  - 按照1：10的比例配置内存和硬盘
- 日志类和查询并发低的场景，可以考虑使用机械硬盘存储
  - 按照1:50的比例配置内存和硬盘
- 单节点数据建议控制在2TB以内，最大不建议超过5TB
- JVM配置机器内存的一半，JVM内存配置不建议超过32G
- 不建议在一台服务器上运行多个节点

内存大小要根据Node 需要存储的数据来进行估算

- 搜索类的比例建议: 1:16
- 日志类: 1:48——1:96之间

假设总数据量1T，设置一个副本就是2T总数据量

- 如果搜索类的项目，每个节点31*16 = 496 G，加上预留空间。所以每个节点最多400G数据，至少需要5个数据节点
- 如果是日志类项目，每个节点31*50= 1550 GB，2个数据节点即可

部署方式:

- 按需选择合理的部署方式
- 如果需要考虑可靠性高可用，建议部署3台单一的Master节点
- 如果有复杂的查询和聚合，建议设置Coordinating节点

集群扩容:

- 增加Coordinating / Ingest Node
    - 解决CPU和内存开销的问题
- 增加数据节点
    - 解决存储的容量的问题
    - 为避免分片分布不均的问题，要提前监控磁盘空间，提前清理数据或增加节点

**容量规划案例1：产品信息库搜索**

特性：

- 被搜索的数据集很大，但是增长相对比较慢(不会有大量的写入)。更关心搜索和聚合的读取性能
- 数据的重要性与时间范围无关。关注的是搜索的相关度

估算索引的的数据量，然后确定分片的大小：

- 单个分片的数据不要超过20 GB
- 可以通过增加副本分片，提高查询的吞吐量

思考：如果单个索引数据量非常大，如何优化提升查询性能?

拆分索引

- 如果业务上有大量的查询是基于一个字段进行Filter，该字段又是一个数量有限的枚举值。
    - 例如订单所在的地区。可以考虑以地区进行索引拆分

如果在单个索引有大量的数据，可以考虑将索引拆分成多个索引：

- 查询性能可以得到提高
- 如果要对多个索引进行查询，还是可以在查询中指定多个索引得以实现
- 如果业务上有大量的查询是基于一个字段进行Filter，该字段数值并不固定
    - 可以启用Routing 功能，按照filter 字段的值分布到集群中不同的shard，降低查询时相关的shard数提高CPU利用率

```
es分片路由的规则：
shard_num = hash(_routing) % num_primary_shards
_routing字段的取值，默认是_id字段，可以自定义。

PUT /users
{
 "settings": {
 "number_of_shards":2
```

```
}
}
POST /users/_create/1?routing=fox
{
 "name":"fox"
}
```

**容量规划案例2: 基于时间序列的数据**

相关场景:

- 日志/指标/安全相关的事件
- 舆情分析

特性:

- 每条数据都有时间戳，文档基本不会被更新(日志和指标数据)
- 用户更多的会查询近期的数据，对旧的数据查询相对较少
- 对数据的写入性能要求比较高

创建基于时间序列的索引:

- 在索引的名字中增加时间信息
- 按照每天/每周/每月的方式进行划分

这样做的好处：更加合理的组织索引，例如随着时间推移，便于对索引做的老化处理。

- 可以利用Hot & Warm 架构
- 备份和删除以及删除的效率高。(Delete By Query执行速度慢，底层也不会立刻释放空间)

基于Date Math方式建立索引

比如：假设当前日期 2022-05-27

| | |
|---|---|
| <indexName-{now/d}> | indexName-2022.05.27 |
| <indexName-{now{YYYY.MM}}> | indexName-2022.05 |

```
# PUT /<logs-{now/d}
PUT /%3Clogs-%7Bnow%2Fd%7D%3E

# POST /<logs-{now/d}>/_search
POST /%3Clogs-%7Bnow%2Fd%7D%3E/_search
```

基于Index Alias索引最新的数据

```
PUT /logs_2022-05-27
PUT /logs_2022-05-26
```

```
3
4 #可以每天晚上定时执行
5 POST /_aliases
6 {
7  "actions": [
8  {
9  "add": {
10  "index": "logs_2022-05-27",
11  "alias": "logs_write"
12  }
13  },
14  {
15  "remove": {
16  "index": "logs_2022-05-26",
17  "alias": "logs_write"
18  }
19  }
20  ]
21 }
22
23 GET /logs_write
```

## ES跨集群搜索（CCS）

### ES水平扩展存在的问题

- 单集群水平扩展时，节点数不能无限增加
    - 当集群的meta 信息(节点，索引，集群状态)过多会导致更新压力变大，单个Active Master会成为性能瓶颈，导致整个集群无法正常工作
- 早期版本，通过Tribe Node可以实现多集群访问的需求，但是还存在一定的问题
    - Tribe Node会以Client Node的方式加入每个集群，集群中Master节点的任务变更需要Tribe Node 的回应才能继续。
    - Tribe Node 不保存Cluster State信息，一旦重启，初始化很慢

- 当多个集群存在索引重名的情况时，只能设置一种 Prefer 规则

**跨集群搜索实战**

早期Tribe Node 的方案存在一定的问题，现已被弃用。Elasticsearch 5.3引入了跨集群搜索的功能(Cross Cluster Search)，推荐使用

- 允许任何节点扮演联合节点，以轻量的方式，将搜索请求进行代理
- 不需要以Client Node的形式加入其他集群

**配置集群**

```
//启动3个集群
elasticsearch.bat -E node.name=cluster0node -E cluster.name=cluster0 -E path.data=cluster0_data -E discovery.type=single-node -E http.port=9200 -E transport.port=9300
elasticsearch.bat -E node.name=cluster1node -E cluster.name=cluster1 -E path.data=cluster1_data -E discovery.type=single-node -E http.port=9201 -E transport.port=9301
elasticsearch.bat -E node.name=cluster2node -E cluster.name=cluster2 -E path.data=cluster2_data -E discovery.type=single-node -E http.port=9202 -E transport.port=9302

//在每个集群上设置动态的设置
PUT _cluster/settings
{
 "persistent": {
  "cluster": {
  "remote": {
  "cluster0": {
  "seeds": [
  "127.0.0.1:9300"
  ],
  "transport.ping_schedule": "30s"
  },
  "cluster1": {
  "seeds": [
  "127.0.0.1:9301"
  ],
  "transport.compress": true,
  "skip_unavailable": true
  },
```

```
  "cluster2": {
  "seeds": [
  "127.0.0.1:9302"
  ]
  }
  }
  }
  }
}

```

CCS的配置：

1）seeds

配置的远程集群的remote cluster的一个node。

2）connected

如果至有少一个到远程集群的连接则为true。

3）num_nodes_connected

远程集群中连接节点的数量。

4）max_connections_per_cluster

远程集群维护的最大连接数。

5）transport.ping_schedule

设置了tcp层面的活性监听

6）skip_unavailable

设置为true的话，当这个remote cluster不可用的时候，就会忽略，默认是false，当对应的remote cluster不可用的话，则会报错。

7）cluster.remote.connections_per_cluster

gateway nodes数量，默认是3

8）cluster.remote.initial_connect_timeout

节点启动时等待远程节点的超时时间，默认是30s

9）cluster.remote.node.attr：

一个节点属性，用于过滤掉remote cluster中 符合gateway nodes的节点，比如设置cluster.remote.node.attr=gateway，那么将匹配节点属性node.attr.gateway: true 的node才会被该node连接用来做CCS查询。

10）cluster.remote.connect：

默认情况下，群集中的任意节点都可以充当federated client并连接到remote cluster，cluster.remote.connect可以设置为 false（默认为true）以防止某些节点连接到remote cluster

11）在使用api进行动态设置的时候每次都要把seeds带上

**创建测试数据**

```
#在不同集群上执行
# cluster0 localhost:9200
POST /users/_doc
{
    "name":"fox",
    "age":"30"
}

#cluster1 localhost:9201
 POST /users/_doc
 {
     "name":"monkey",
     "age":"33"
 }

 #cluster2 localhost:9202
 POST /users/_doc
 {
     "name":"mark",
     "age":"35"
 }

```

**查询**

```
#查询结果获取到所有集群符合要求的数据
GET /users,cluster1:users,cluster2:users/_search
{
  "query": {
  "range": {
  "age": {
  "gte": 30,
  "lte": 40
  }
   }
   }
}
```

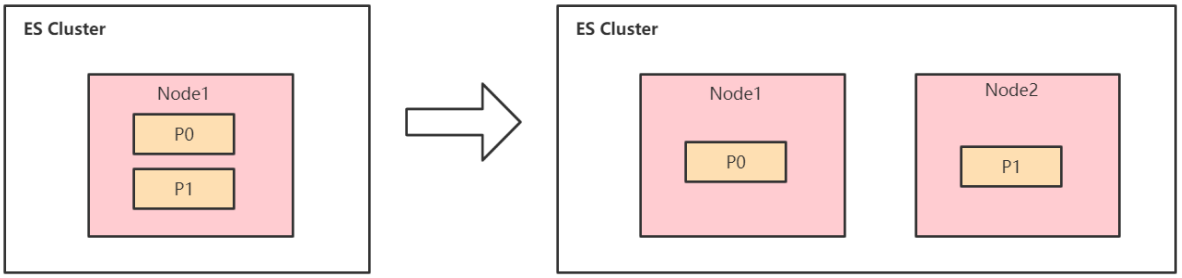
## 分片的设计和管理

### 单个分片

- 7.0开始，新创建一个索引时，默认只有一个主分片。单个分片，查询算分，聚合不准的问题都可以得以避免
- 单个索引，单个分片时候，集群无法实现水平扩展。即使增加新的节点，无法实现水平扩展

### 两个分片

集群增加一个节点后，Elasticsearch 会自动进行分片的移动，也叫 Shard Rebalancing




### 算分不准的原因

相关性算分在分片之间是相互独立的，每个分片都基于自己的分片上的数据进行相关度计算。这会导致打分偏离的情况，特别是数据量很少时。当文档总数很少的情况下，如果主分片大于1，主分片数越多，相关性算分会越不准

### Demo

```
PUT /blogs
{
 "settings":{
 "number_of_shards" : "3"
 }
}

POST /blogs/_doc/1?routing=fox
{
  "content":"Cross Cluster elasticsearch Search"
 }

POST /blogs/_doc/2?routing=fox2
{
  "content":"elasticsearch Search"
 }
```

```
17
18 POST /blogs/_doc/3?routing=fox3
19 {
20  "content":"elasticsearch"
21 }
22
23 GET /blogs/_search
24 {
25  "query": {
26  "match": {
27  "content": "elasticsearch"
28  }
29  }
30 }
31
32 #解决算分不准的问题
33 GET /blogs/_search?search_type=dfs_query_then_fetch
34 {
35  "query": {
36  "match": {
37  "content": "elasticsearch"
38  }
39  }
40 }
41
```

解决算分不准的方法：

- 数据量不大的时候，可以将主分片数设置为1。当数据量足够大时候，只要保证文档均匀分散在各个分片上，结果一般就不会出现偏差
- 使用DFS Query Then Fetch
    - 搜索的URL中指定参数“_search?search_type=dfs_query_then_fetch"
    - 到每个分片把各分片的词频和文档频率进行搜集，然后完整的进行一次相关性算分，

耗费更加多的CPU和内存，执行性能低下，一般不建议使用

**如何设计分片数**

当分片数>节点数时

- 一旦集群中有新的数据节点加入，分片就可以自动进行分配
- 分片在重新分配时，系统不会有downtime

多分片的好处: 一个索引如果分布在不同的节点，多个节点可以并行执行

- 查询可以并行执行
- 数据写入可以分散到多个机器

**案例1**

- 每天1GB的数据，一个索引一个主分片，一个副本分片
- 需保留半年的数据，接近360 GB的数据量，360个分片

**案例2**

- 5个不同的日志，每天创建一个日志索引。每个日志索引创建10个主分片
- 保留半年的数据
- 5*10* 30* 6 = 9000个分片

**分片过多所带来的副作用**

Shard是Elasticsearch 实现集群水平扩展的最小单位。过多设置分片数会带来一些潜在的问题：

- 每个分片是一个Lucene的索引，会使用机器的资源。过多的分片会导致额外的性能开销。
- 每次搜索的请求,需要从每个分片上获取数据
- 分片的Meta 信息由Master节点维护。过多，会增加管理的负担。经验值，控制分片总数在10W以内

**如何确定主分片数**

从存储的物理角度看：

- 搜索类应用，单个分片不要超过20 GB
- 日志类应用，单个分片不要大于50 GB

为什么要控制分片存储大小：

- 提高Update 的性能
- 进行Merge 时，减少所需的资源
- 丢失节点后，具备更快的恢复速度

- 便于分片在集群内 Rebalancing

**如何确定副本分片数**

副本是主分片的拷贝：

- 提高系统可用性：响应查询请求，防止数据丢失
- 需要占用和主分片一样的资源

对性能的影响：

- 副本会降低数据的索引速度: 有几份副本就会有几倍的CPU资源消耗在索引上
- 会减缓对主分片的查询压力，但是会消耗同样的内存资源。如果机器资源充分，提高副本数，可以提高整体的查询QPS

ES的分片策略会尽量保证节点上的分片数大致相同，但是有些场景下会导致分配不均匀：

- 扩容的新节点没有数据，导致新索引集中在新的节点
- 热点数据过于集中，可能会产生性能问题

可以通过调整分片总数，避免分配不均衡

- "index.routing.allocation.total_shards_per_node"，index级别的，表示这个index每个Node总共允许存在多少个shard，默认值是-1表示无穷多个；
- "cluster.routing.allocation.total_shards_per_node"，cluster级别，表示集群范围内每个Node允许存在有多少个shard。默认值是-1表示无穷多个。

如果目标Node的Shard数超过了配置的上限，则不允许分配Shard到该Node上。注意：index级别的配置会覆盖cluster级别的配置。

思考：5个节点的集群。索引有5个主分片，1个副本，index.routing.allocation.total_shards_per_node应该如何设置?

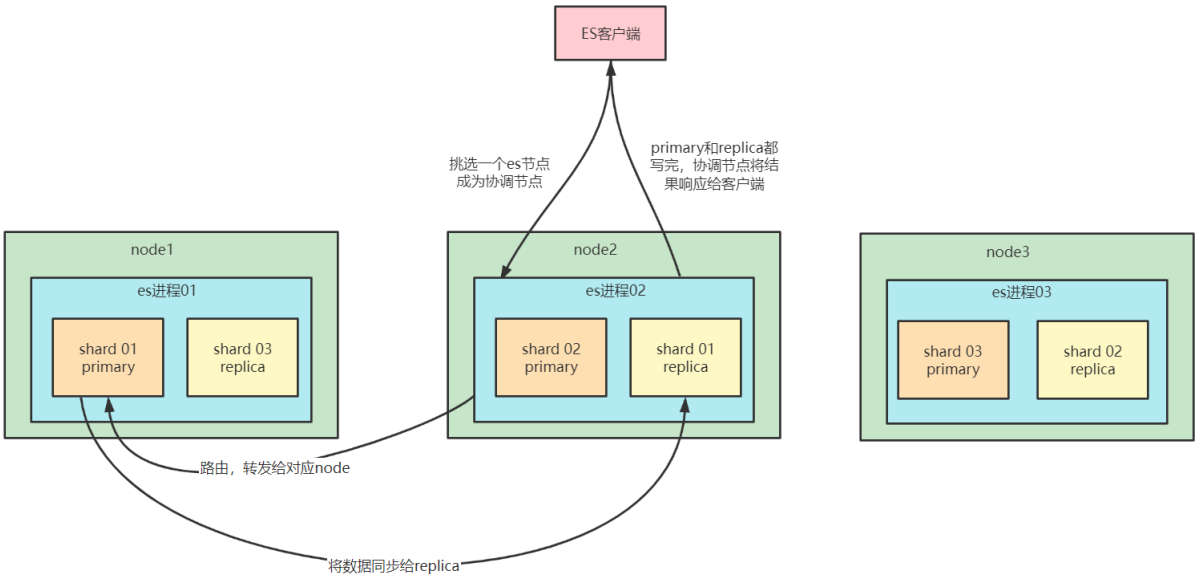
- (5+5)/ 5= 2
- 生产环境中要适当调大这个数字，避免有节点下线时，分片无法正常迁移

## ES底层读写工作原理

写请求是写入 primary shard，然后同步给所有的 replica shard；读请求可以从 primary shard 或 replica shard 读取，采用的是随机轮询算法。

**ES写入数据的过程**

1. 客户端选择一个node发送请求过去，这个node就是coordinating node (协调节点)
2. coordinating node，对document进行路由，将请求转发给对应的node
3. node上的primary shard处理请求，然后将数据同步到replica node
4. coordinating node如果发现primary node和所有的replica node都搞定之后，就会返回请求到客户端




**ES读取数据的过程**

**根据id查询数据的过程**

根据 doc id 进行 hash，判断出来当时把 doc id 分配到了哪个 shard 上面去，从那个 shard 去查询。

1. 客户端发送请求到任意一个 node，成为 coordinate node 。
2. coordinate node 对 doc id 进行哈希路由，将请求转发到对应的 node，此时会使用 round-robin 随机轮询算法，在 primary shard 以及其所有 replica 中随机选择一个，让读请求负载均衡。
3. 接收请求的 node 返回 document 给 coordinate node 。
4. coordinate node 返回 document 给客户端。

**根据关键词查询数据的过程**

- 客户端发送请求到一个 coordinate node 。
- 协调节点将搜索请求转发到所有的 shard 对应的 primary shard 或 replica shard ，都可以。
- query phase：每个 shard 将自己的搜索结果返回给协调节点，由协调节点进行数据的合并、排序、分页等操作，产出最终结果。

- fetch phase：接着由协调节点根据 doc id 去各个节点上拉取实际的 document 数据，最终返回给客户端。

## 写数据底层原理

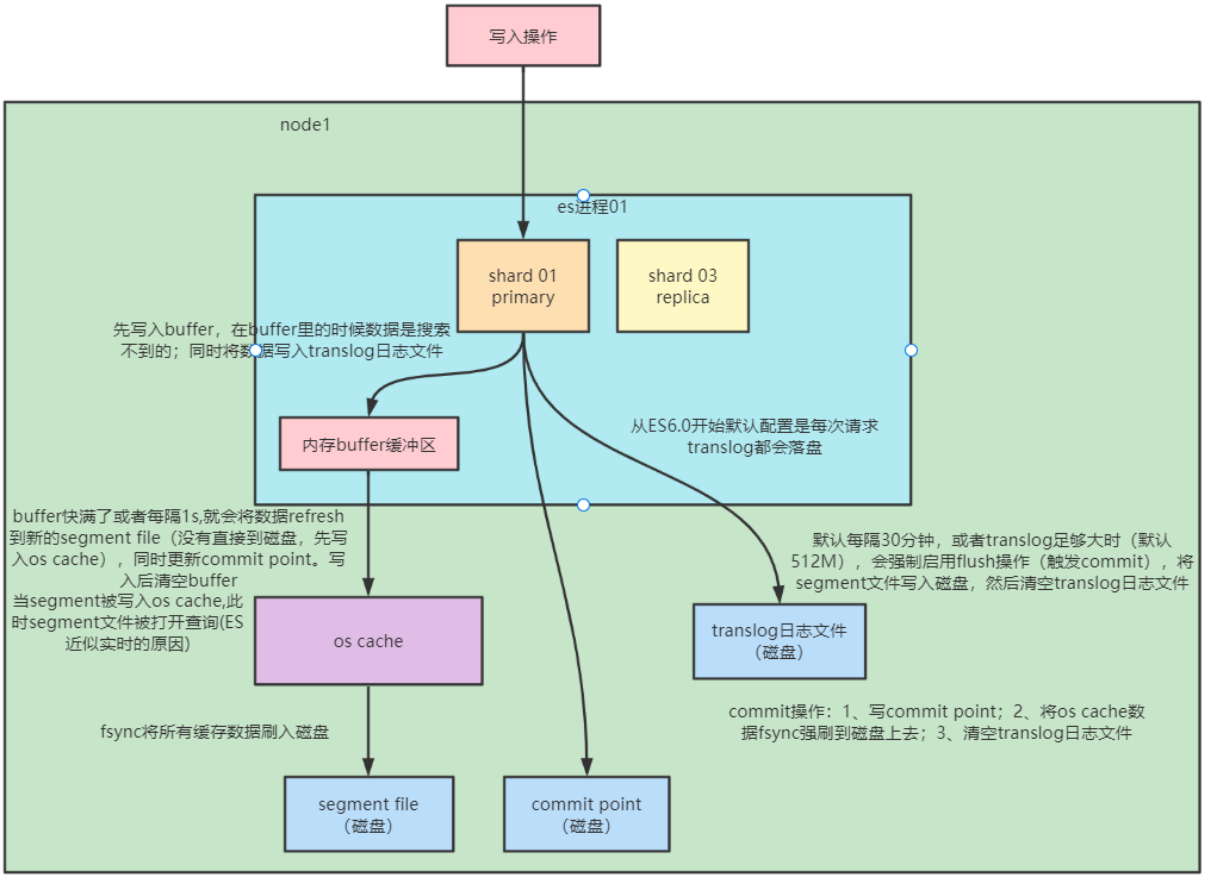
**核心概念**

**segment file**: 存储倒排索引的文件，每个segment本质上就是一个倒排索引，每秒都会生成一个segment文件，当文件过多时es会自动进行segment merge（合并文件），合并时会同时将已经标注删除的文档物理删除。

**commit point**: 记录当前所有可用的segment，每个commit point都会维护一个.del文件，即每个.del文件都有一个commit point文件（es删除数据本质是不属于物理删除），当es做删改操作时首先会在.del文件中声明某个document已经被删除，文件内记录了在某个segment内某个文档已经被删除，当查询请求过来时在segment中被删除的文件是能够查出来的，但是当返回结果时会根据commit point维护的那个.del文件把已经删除的文档过滤掉

**translog日志文件**: 为了防止elasticsearch宕机造成数据丢失保证可靠存储，es会将每次写入数据同时写到translog日志中。

**os cache**：操作系统里面，磁盘文件其实都有一个东西，叫做os cache，操作系统缓存，就是说数据写入磁盘文件之前，会先进入os cache，先进入操作系统级别的一个内存缓存中去



**Refresh**

- 将文档先保存在Index buffer中，以refresh_interval为间隔时间，定期清空buffer，生成 segment,借助文件系统缓存的特性，先将segment放在文件系统缓存中，并开放查询，以提升搜索的实时性

**Translog**

- Segment没有写入磁盘，即便发生了当机，重启后，数据也能恢复，从ES6.0开始默认配置是每次请求都会落盘

**Flush**

- 删除旧的translog 文件
- 生成Segment并写入磁盘 | 更新commit point并写入磁盘。ES自动完成，可优化点不多

## 如何提升集群的读写性能

### 提升集群读取性能的方法

#### 数据建模

- 尽量将数据先行计算，然后保存到Elasticsearch 中。尽量避免查询时的 Script 计算

```
#避免查询时脚本
GET blogs/_search
{
 "query": {
 "bool": {
 "must": [
 {"match": {
 "title": "elasticsearch"
 }}
  ],

  "filter": {
  "script": {
  "script": {
  "source": "doc['title.keyword'].value.length()>5"
  }
  }
  }
  }
```

```
  }
}
```

- 尽量使用Filter Context，利用缓存机制，减少不必要的算分
- 结合profile，explain API分析慢查询的问题，持续优化数据模型
- 避免使用*开头的通配符查询

```
GET /es_db/_search
{
 "query": {
 "wildcard": {
 "address": {
 "value": "*白云*"
 }
 }
 }
}
```

**优化分片**

- 避免Over Sharing
  - 一个查询需要访问每一个分片，分片过多，会导致不必要的查询开销
- 结合应用场景，控制单个分片的大小
  - Search: 20GB
  - Logging: 40GB
- Force-merge Read-only索引
  - 使用基于时间序列的索引，将只读的索引进行force merge，减少segment数量

```
#手动force merge
POST /my_index/_forcemerge
```

**提升写入性能的方法**

- 写性能优化的目标: 增大写吞吐量，越高越好
- 客户端: 多线程，批量写
  - 可以通过性能测试，确定最佳文档数量

    - 多线程: 需要观察是否有HTTP 429（Too Many Requests）返回，实现 Retry以及线程数量的自动调节
- 服务器端: 单个性能问题，往往是多个因素造成的。需要先分解问题，在单个节点上进行调整并且结合测试，尽可能压榨硬件资源,以达到最高吞吐量
    - 使用更好的硬件。观察CPU / IO Block
    - 线程切换 | 堆栈状况

**服务器端优化写入性能的一些手段**

- 降低IO操作
    - 使用ES自动生成的文档Id
    - 一些相关的ES 配置，如Refresh Interval
- 降低 CPU 和存储开销
    - 减少不必要分词
    - 避免不需要的doc_values
    - 文档的字段尽量保证相同的顺予，可以提高文档的压缩率
- 尽可能做到写入和分片的均衡负载，实现水平扩展
    - Shard Filtering / Write Load Balancer
- 调整Bulk 线程池和队列

注意：ES 的默认设置，已经综合考虑了数据可靠性，搜索的实时性，写入速度，一般不要盲目修改。一切优化，都要基于高质量的数据建模。

**建模时的优化**

- 只需要聚合不需要搜索，index设置成false
- 不要对字符串使用默认的dynamic mapping。字段数量过多，会对性能产生比较大的影响
- Index_options控制在创建倒排索引时，哪些内容会被添加到倒排索引中。

如果需要追求极致的写入速度，可以牺牲数据可靠性及搜索实时性以换取性能：

- 牺牲可靠性: 将副本分片设置为0，写入完毕再调整回去
- 牺牲搜索实时性：增加Refresh Interval的时间
- 牺牲可靠性: 修改Translog的配置

**降低 Refresh的频率**

- 增加refresh_interval 的数值。默认为1s ，如果设置成-1，会禁止自动refresh
    - 避免过于频繁的refresh，而生成过多的segment 文件
    - 但是会降低搜索的实时性

```
PUT /my_index/_settings
{
    "index" : {
        "refresh_interval" : "10s"
    }
}
```

- 增大静态配置参数indices.memory.index_buffer_size
    - 默认是10%，会导致自动触发refresh

**降低Translog写磁盘的频率，但是会降低容灾能力**

- Index.translog.durability：默认是request，每个请求都落盘。设置成async，异步写入
- Index.translog.sync_interval：设置为60s，每分钟执行一次
- Index.translog.flush_threshod_size: 默认512 m，可以适当调大。当translog超过该值，会触发flush

**分片设定**

- 副本在写入时设为0，完成后再增加
- 合理设置主分片数，确保均匀分配在所有数据节点上
- Index.routing.allocation.total_share_per_node:限定每个索引在每个节点上可分配的主分片数

**调整Bulk 线程池和队列**

- 客户端
    - 单个bulk请求体的数据量不要太大，官方建议大约5-15m
    - 写入端的 bulk请求超时需要足够长，建议60s 以上
    - 写入端尽量将数据轮询打到不同节点。
- 服务器端
    - 索引创建属于计算密集型任务，应该使用固定大小的线程池来配置。来不及处理的放入队列，线程数应该配置成CPU核心数+1，

避免过多的上下文切换

- 队列大小可以适当增加，不要过大，否则占用的内存会成为GC的负担
- ES线程池设置：https://blog.csdn.net/justlpf/article/details/103233215

```
DELETE myindex
PUT myindex
{
 "settings": {
 "index": {
 "refresh_interval": "30s", #30s一次refresh
 "number_of_shards": "2"
 },
 "routing": {
  "allocation": {
  "total_shards_per_node": "3" #控制分片，避免数据热点
  }
 },
 "translog": {
 "sync_interval": "30s",
 "durability": "async" #降低translog落盘频率
 },
 "number_of_replicas": 0
 },
 "mappings": {
 "dynamic": false, #避免不必要的字段索引，必要时可以通过update by query
索引必要的字段
 "properties": {}
 }
}
```