> 主讲老师：Fox
> 有道云笔记地址：https://note.youdao.com/s/UostSRP4

1. 节点角色配置方案

节点角色介绍

如果你的Elasticsearch集群是7.9之前的版本，在配置节点的时候，则只会涉及节点类型的知识。

- 主节点：负责集群管理和元数据维护，确保集群正常运行。
- 数据节点：负责存储、检索和处理数据，提供搜索和聚合功能。
- 协调节点：处理客户端请求，协调数据节点工作，优化分布式搜索。
- ingest节点：即预处理节点，负责数据预处理，如过滤、转换等，准备好数据再将其索引到数据节点。

Elasticsearch 7.9版本开始引入节点角色的概念。节点角色划分的目的是让不同角色的节点各司其职，共同确保集群功能的稳定和性能的高可用。

Elasticsearch早期版本（以7.1版本为例）中，如果配置仅候选主节点类型，那么极端情况下需要的配置如下：

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

这是非常烦琐的配置，其逻辑类似于“若我要说明自己是主节点，则要先说明我不是数据节点、不是ingest节点、不是XXX节点……”。而节点角色的出现“革命性”地解决了这个问题。利用节点角色，我们只需要说明“我是XXX”即可，而不需要卖力解释“我不是XXX”。

```
node.roles: [data,master]
```

以Elasticsearch 8.X版本集群为例，如果我们不手动设置节点角色，则默认节点角色为cdfhilmrstw。

对默认节点角色cdfhilmrstw的解释如下表所示：

当集群规模比较大之后（比如集群节点数大于6个），就需要手动设定、配置节点角色。

一个节点只承担一个角色的配置

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

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

增加节点的场景

- 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

2. 高可用场景部署方案

读写分离架构

Hot & Warm 架构

热节点存放用户最关心的热数据；温节点存放用户关心优先级低的暖数据；冷节点存放用户不太关心的冷数据。

典型的应用场景

在成本有限的前提下，让客户关注的实时数据和历史数据硬件隔离，最大化解决客户反应的响应时间慢的问题。

业务场景描述：每日增量6TB日志数据，高峰时段写入及查询频率都较高，集群压力较大，查询ES时，常出现查询缓慢问题。

- ES集群的索引写入及查询速度主要依赖于磁盘的IO速度，冷热数据分离的关键为使用SSD磁盘存储热数据，提升查询效率。
- 若全部使用SSD，成本过高，且存放冷数据较为浪费，因而使用普通SATA磁盘与SSD磁盘混搭，可做到资源充分利用，性能大幅提升的目标。

ES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

```
# 标记一个 Hot 节点
node.attr.my_node_type: hot

# 标记一个 warm 节点
node.attr.my_node_type: warm

# 查看节点
GET /_cat/nodeattrs?v
```

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
```

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
```

3. 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

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

1）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

2）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"
}

```

3）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
      }
    }
  }
}
```

4. 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)

做容量规划之前应该先对业务的性能需求做一个评估。

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

常见用例：

- 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1:10-20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: 固定大小的数据集

场景：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

相关的场景：

- 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

创建timed-base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
- 备份和删除的效率高

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

- 创建索引，每天/每周/每月
- 在索引的名字中增加时间信息

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

#可以每天晚上定时执行
POST /_aliases
{
  "actions": [
    {
      "add": {
        "index": "logs_2022-05-27",
        "alias": "logs_write"
      }
    },
    {
      "remove": {
        "index": "logs_2022-05-26",
        "alias": "logs_write"
      }
    }
  ]
}

GET /logs_write
```

5. 如何设计和管理分片

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

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。
  - Lucene Indices / File descriptors / RAM/ CPU
  - 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