百度智能云

BES 文档

目录

功能发布记录

2024年

发布时间	功能概述
2024-10	• 上线智能巡检功能，提供集群、节点和索引粒度的十余个巡检项，帮助用户自动化观测和分析集群状态，探测潜在风险并提出解决方案，详见智能巡检 • 上线慢查询隔离功能，支持将内存消耗超过阈值的查询进行熔断，避免影响集群整体稳定性，详见 慢查询隔离
2024-08	• 增加读写失败率、读写拒绝率等监控指标，详见监控指标说明 • 发布全新CCR（Cross Cluster Replication）跨集群复制功能，支持任务观测和任务启停，详见 CCR跨集群复制
2024-07	• 支持RRF（Reciprocal Rank Fusion）倒数排序融合的查询方式，便于将多个结果集的结果相互融合获得高质量的排序结果。详见RRF融合查询文档 • 上线智能规模评估工具，便于用户创建集群时自助评估集群规模 • 新增支持 南京 和 成都 地域
	• 支持向量缓存管理优化功能，便于更好实现索引结构常驻内存，提升向量检索的速度。详见向量索引缓存管理文档

2024-06	• 增加新用户服务开通功能，便于新用户初步了解BES服务 • 新增Kibana节点监控指标，便于观测Kibana节点状态；更新集群健康状态指标，增加灰色状态，表示接口超期的情况 • 日志查询功能支持导出，支持用户将集群日志导出到BOS进一步分析。详见日志导出文档 • 定时调度rollover任务支持模糊匹配，支持通过通配符（*、-、？等）和索引名称来进行配置
2024-05	• 集群用户名的密码规则更新，支持更大的密码长度和更多特殊符号。7.10.2版本集群的1.7.3及以上内核版本、7.4.2版本集群的1.5.1及以上内核版本、6.5.3版本集群的1.4.1及以上集群版本开始使用新密码规则，存量集群的用户名密码规则保持不变。详见密码重置文档 • 定时调度rollover任务支持模糊匹配，支持通过通配符（*、-、？等）和索引名称来进行配置
2024-04	• 创建集群、扩容集群、蓝绿迁移等功能增加子网IP数量校验，子网IP数量不足时提示用户需要更换子网或协调子网IP资源 • 定时调度rollover任务支持模糊匹配，支持通过通配符（*、-、？等）和索引名称来进行配置
2024-03	• 支持官网资源管理功能，创建集群时需要选择资源分组，详见资源管理介绍文档 • 接入官网购物车功能，支持加入购物车，批量购买BES集群

2024-02	• 上线节点批量启停功能，支持用户同时开启或暂停多个节点，详见功能文档 • OpenAPI支持标签管理，支持创建集群时绑定绑定标签，通过OpenAPI查看集群标签信息，编辑集群标签
2024-01	• 上线 CCR （Cross Cluster Replication）跨集群复制功能，提升集群灾难恢复及高可用性，便于就近访问数据，降低访问延迟 • 集群和节点状态全面更新

2023年

发布时间	功能概述
2023-12	• 支持变更磁盘类型，升级磁盘类型扩展BES集群性能，提供蓝绿和直调两种升级模式供用户选择
2023-11	• 集群命名规则更新，新命名规则为：支持1-30个英文大小字母、数字、下划线（_）或连字符（-） • 已创建集群支持修改集群名称，便于用户管理和更新集群 • 上线边缘BES服务，支持在边缘节点创建BES集群，支持更多地域 • 支持增强型云磁盘SSD_PL2 ，支持性能预配置，单盘支持最大 10W IOPS 和 750MBps 吞吐

2023-10	• 上线内核版本升级功能，支持用户查看当前集群内核版，平滑升级到最新内核版本
2023-09	• 集群扩缩容功能更新。支持自动化节点数据迁移和节点缩容，详见 集群扩缩容文档 • 上线 全新日志查询功能。支持用户查看主日志、查询慢日志、索引慢日志、GC日志、ES访问日志等 • 向量检索能力升级。优化forcemerge机制，提供向量标量混合查询能力，持续优化读写性能
2023-08	• BES向量数据库能力发布。BES自主开发向量数据库能力，支持多种索引类型和相似度距离算法，构建十亿级向量规模，实现毫秒级延迟，广泛应用于推荐系统、问答系统、语义检索、智能客服等领域，详见文档 • 支持ZSTD压缩算法。提供ZSTD压缩算法插件，具备更高索引压缩比，帮助用户大幅降低索引存储成本
2023-07	• 支持HTTPS加密传输。BES支持用户在创建集群和使用集群时，选择 HTTPS协议，提升访问安全性 • 支持 通过API上传NLP词典。BES提供OpenAPI接口，无需百度智能云控制台，直接调用API，更新NLP分词词典
2023-06	• 新增 集群列表导出功能。控制台集群列表页，支持列表字段导出 • 支持增强型云磁盘ESSD_PL1。创建集群时，数据盘支持选择云磁盘ESSD_PL1。云磁盘性能比较，详见磁盘性能文档

2023-04	• 集群列表优化，支持自定义列 • Logstash托管服务更新。支持logstash-bos-input插件
2023-03	• 发布性能测试文档 • 重置密码功能优化。 支持用户通过控制台重置集群访问密码，用于访问集群和登录Kibana。重置后快速生效，不影响其他功能操作 • 控制台核心操作接入云审计（BCT），便于对云账户进行监管、合规性检查、操作审核和风险审核
2023-02	基于BOS冷热数据分离发布2.0版本。提升性能与易用性，持续帮助用户大幅降低成本 • 查询性能提升，冷数据查询性能 优于 高效云盘 • 增加可观测接口，便于观测冷热数据分布情况 • 提供冷索引预热功能，缓解首次查询延迟 • 清理缓存功能，支持用户使用API清理缓存文件 • 本地缓存支持通过调整配置进行性能调优 • 上线 版本升级功能。提供蓝绿和冷重启两种模式，支持集群 6.x-7.x 版本升级。 • 7.10版本支持 向量检索功能。面向文字搜索、图像视频匹配、语音识别等向量检索场景，提供了linear和hnsw两种算法，对特征向量进行相似度匹配，得到高召回率的检索结果。

2023-01	监控功能更新 ： • 新增节点状态列表，支持观测节点运行状态、CPU使用率、磁盘使用率等基础指标，跳转指标报警 • 新增节点指标，新增IO Util、load_1m、FullGC次数、Field Data内存使用、HeapMemory使用率、search线程池队排队任务数、write线程池队排队任务数、request cache、query cache、段内存等节点指标

2022年

发布时间	功能概述
2022-12	• 上线 限流插件，实现集群和节点级别的读写限流，优化指定索引的读写速度，将流量控制在合适范围内，以提高集群的稳定性 • 集群列表支持检索，支持通过集群名称、集群ID和标签，对已创建的集群进行检索 • SQL解析插件，支持7.10.2版本
2022-11	• 配置变更支持新增节点类型，支持已创建集群添加专用主节点，提升集群稳定性 • 定时调度删除定时任务逻辑优化，缓解大量删除任务导致任务积压问题
2022-10	• 支持SQL解析插件，可使用 SQL 代替 DSL 查询语言

2022-10	• 支持Elasticsearch7.10版本，兼容7.4.2版本特性，支持创建和使用7.10版本集群
2022-09	• 禁售新加坡Region
2022-09	• 支持标签服务，通过对集群添加标签，实现将资源基于不同维度进行分组管理
2022-08	• 发布OpenAPI，支持调用OpenAPI进行配置变更、定时调度、集群重启、集群启停、自动续费等操作
2022-08	• 支持创建双可用区集群，支持将master专用主节点部署在两个可用区中，实现一定的高可用
2022-06	• 支持IDaaSMFA多因素认证，高危操作支持通过短信或动态验证码方式进行二次验证

2022-04	• 发布OpenAPI 支持调用OpenAPI进行创建集群、查看集群列表、删除集群等操作
2022-03	• 开放Exporter端口，支持接入企业统一的Prometheus服务
2022-03	监控功能更新 • 集群指标新增集群总分片数、集群未分配分片数、集群数据节点平均分片数 • 节点指标新增硬盘读iops、硬盘写iops、硬盘读流量、硬盘写流量 • 接入云监控BCM侧边栏，支持BCM实例组功能（批量报警），支持在BCM查看冷数据节点监控
2022-01	• 集群重启功能上线，支持滚动重启和强制重启两种方式

2021年

发布时间	功能概述
2021-12	• 监控功能新增集群健康状态、index写入qps、index增量、index耗时、query qps、query增量、query耗时等集群监控指标，同时支持设置指标报警

2021-08	• 新增索引数据量观测功能
2021-06	• 冷数据节点、协调节点上线，可根据业务场景创建不同类型的节点
2021-05	• 多可用区部署功能上线，支持多个可用区多活容灾 • 跨可用区节点迁移功能上线，支持无缝迁移节点
2021-04	• 配置变更功能上线，支持按需升级节点规格或磁盘容量 • 华东-上海区域开放
2021-02	• 武汉金融专区、香港、新加坡新版节点套餐上线
	• 百度NLP词典动态热更新上线（公测）

2021-01	• 基于BOS的冷热数据分离架构支持限速功能 • 集群日志查询功能上线

2020年

发布时间	功能概述
2020-12	• 定时调度功能上线，支持创建调度任务（公测） • 支持设置专用主节点
2020-11	• 新版节点购买配置上线，支持计算资源和存储资源灵活配置
2020-09	• 基于BOS的冷热数据分离架构上线 • 动态同义词插件上线 • mapper-size插件上线
2020-05	• 向量检索插件上线，支持linear算法和hnsw算法

2020-04	• 支持从DTS支持的数据源向百度智能云Elasticsearch迁移数据
2020-02	• Baidu Elasticsearch 7.4.2 版本上线 • Baidu NLP分词插件上线

2019年

发布时间	功能概述
2019-06	• 增加HybridFS支持，对于nvd, dvd, tim, cfs使用mmap方式访问，其他文件使用niofs，有效提升读取性能 • 修复ik分词插件不能动态更新词库的bug
2019-02	• BES接入IAM系统，新增多用户访问控制功能，支持为子用户设置只读、运维和管理权限. • 支持集群资源评估。用户可以根据索引膨胀、ES内部开销、应急开销和副本的数量预估磁盘空间的总量，并结合所选的套餐来估算节点的数目 • 当IK自身带的词库不满足需求时，用户可以自己定义词库
2019-01	• 集成的插件列表：repository-bos、baidu-pack、anylasis-ik、anylasis-pinyin、ingest-attachment、elasticsearch-sql等 • http.cors.enabled: false http.cors.allow-origin: "*"，一些与ES相关的独立部署的Web服务可能需要跨域支持

2018年

发布时间	功能概述
2018-10	• ik analyzer支持动态设置词库
2018-08	• 修改默认的cfs ratio，从0.1修改为0，即默认在merge下禁用cfs • 修改cluster info service中更新index state和node state的调度周期为5min • 将Index、Bulk、GET、Search 这4个线程池的大小变更为可以动态调整 • 添加插件：拼音分词 • 支持基于BOS对象存储系统的备份和恢复 • 支持基于http basic的权限验证

产品描述

产品介绍

百度智能云Elasticsearch是一项托管服务，让您可以在百度智能云中轻松地部署、操作和扩展 Elasticsearch。Elasticsearch是一款非常流行的开源分布式检索和分析引擎。通过百度智能云管理控制台，只需几分钟就可以创建和配置百度智能云Elasticsearch集群。该服务自动检测并替换出现故障的Elasticsearch节点，减少自行管理基础设施和Elasticsearch软件的相关开销。百度智能云Elasticsearch服务完全兼容Elasticsearch开源 API，便于您现有Elasticsearch业务零成本迁移。同时百度智能云Elasticsearch服务提供权限管理机制，便于您对集群的权限自由配置，进一步保证数据安全。

基本概念

集群（cluster）

Elasticsearch集群（cluster）是一个或多个节点的集合，它们共同存储了整个数据集，并提供了联合索引以及可跨所有节点的搜索能力。多节点组成的集群拥有冗余能力，它可以在一个或几个节点出现故障时保证服务的整体可用性。集群靠其独有的名称进行标识，即用户在百度智能云创建的集群名称。节点靠其集群名称来决定加入哪个Elasticsearch集群，一个节点只能属一个集群。如果不考虑冗余能力等特性，仅有一个节点的Elasticsearch集群一样可以实现所有的存储及搜索功能。

节点（node）

运行了单个实例的Elasticsearch主机称为节点（node），它是集群的一个成员，可以存储数据、参与集群索引及搜索操作。类似于集群，节点靠其名称进行标识。一个节点可以被添加到指定名称的集群中。默认情况下，每个节点会被设置加入到名称为

用户设置的集群名称的集群中，这意味着，如果在同一网络中启动了某些节点（假设这些节点可以发现彼此），它们会自动形成并集群中。

索引（index）

Elasticsearch将数据存储于一个或多个索引（index）中，索引是具有类似特性的文档的集合。类比传统的关系型数据库领域来说，索引相当于SQL中的一个数据库，或者一个数据存储方案（schema）。 索引由其名称（必须为全小写字符）进行标识，并通过引用此名称完成文档的创建、搜索、更新及删除操作。一个Elasticsearch集群中可以按需创建任意数目的索引。

类型（type）

类型（type）是索引内部的逻辑分区。在Elasticsearch 6.x中，Elasticsearch禁止了一个索引中使用多个type；在Elasticsearch 7.x中，Elasticsearch使用`_doc`作为默认的type，并在API中逐步取消type的使用；在将来的Elasticsearch 8中，会彻底移除type概念，详情请参见Elasticsearch官方文档。

文档（document）

文档（document）是Lucene索引和搜索的原子单位，它是包含了一个或多个域的容器，基于JSON格式进行表示。 文档由一个或多个域组成，每个域拥有一个名字及一个或多个值，有多个值的域通常称为多值域。每个文档可以存储不同的域集，但同一类型下的文档至应该有某种程度上的相似之处。相当于关系型数据库中的一行数据。

字段（field）

组成文档的最小单位。相当于关系型数据库中的一列数据。

映射（mapping）

Elasticsearch中，所有的文档在存储之前都要首先进行分析。映射（mapping）被用来定义一个文档以及其所包含的字段如何被存储和索引，例如定义字段的名称和类型，以及选择所使用的分词器等。相当于关系型数据库中的Schema。

分片（shards）

Elasticsearch的分片（shard）机制可将一个索引内部的数据分布地存储于多个节点，它通过将一个索引切分为多个底层物理的Lucene索引完成索引数据的分割存储功能，这每一个物理的Lucene索引称为一个分片。每个分片其内部都是一个全功能且独立的索引，因此可由集群中的任何主机存储。

一个分片可以是主分片（primary shard）或副本分片（replica shard）。Elasticsearch 7.0以下版本默认为一个索引创建5个主分片，并分别为每个主分片创建1个副本分片，7.0及以上版本默认创建1个主分片和1个副本分片。主分片的数量只能在索引创建前指定，并且索引创建后不能动态更改，只能通过reindex或split等方式修改。副本分片的数据可以根据需要动态修改。

关键特性

一键部署

使用百度智能云Elasticsearch管理控制台在几分钟内就可部署好Elasticsearch集群，无需进行复杂的配置。

动态扩容

根据业务需求，通过BCE控制台随时动态扩容集群规模，满足业务增长的同时充分利用集群资源。

安全隔离

每个用户拥有独立集群，自由管理，资源隔离，数据安全。

权限控制

为Elasticsearch集群添加权限管理模块，方便用户对数据表进行权限控制，进一步提高数据安全性。

简单易用

使用 RESTful 接口，HTTP 协议，JSON 格式数据，完全兼容 Elasticsearch 官方 API。方便用户通过任意 HTTP 客户端连接使

用。

高可用

实时监控集群及各节点健康状况，发现问题自动报警并恢复，保证集群高可用性。

强大的数据检索及分析能力

依托开源分布式检索分析引擎 Elasticsearch，为用户提供强大的数据检索与分析功能。

超高性能

百度专家团队深度调优，确保集群性能。

经济高效

按需付费，价格低廉。

产品性能

概述

总览

通过Elasticsearch官方提供的Benchmark rally脚本对不同规格的BES集群实例（7.10.2版本）进行压测，并对相关的压测指标说明。

提供以下规格的BES集群的压测结果：

- 4核8GB 的3个数据节点实例基准性能指标
- 8核16GB 的3个数据节点实例基准性能测试
- 16核32GB 的3个数据节点实例基准性能指标
- 4核16GB 的3个数据节点实例基准性能指标
- 8核32GB 的3个数据节点实例基准性能指标
- 16核64GB 的3个数据节点实例基准性能指标

压测指标说明

在进行压测时，您可以参考下表查看相关指标的具体说明。

指标名称	指标说明
Cumulative indexing time of primary shards	所有主分片索引累积时间总和。该时间总和是指多个indexing线程消耗CPU时间的总和，而不是自然时间。例如有M个indexing线程，运行了N分钟，那么此指标的总时间就是：M*N分钟，而不是N分钟。
Min cumulative indexing time across primary shards	跨主分片索引累积时间的最小值。
Median cumulative indexing time across primary shards	跨主分片索引累积时间的平均值。
Max cumulative indexing time across primary shards	跨主分片索引累积时间的最大值。
Cumulative indexing throttle time of primary shards	所有主分片索引时被限流的累积时间。累计时间是指索引被限流时indexing线程消耗的CPU时间总和，而不是自然时间。
Min cumulative indexing throttle time across primary	跨主分片索引时被限流的累积时间的最小值

throttle time across primary shards	跨主分片索引时被限流的累积时间的最小值。
Median cumulative indexing throttle time across primary shards	跨主分片索引时被限流的累积时间的平均值。
Max cumulative indexing throttle time across primary shards	跨主分片索引时被限流的累积时间的最大值。
Cumulative merge time of primary shards	主分片的累积合并的运行时间，也是指线程消耗CPU的时间总和。
Cumulative merge count of primary shards	主分片合并的累积次数。不是所有的分片都会有合并（merge）动作。
Min cumulative merge time across primary shards	跨主分片索引合并累积时间的最小值。
Median cumulative merge time across primary shards	跨主分片索引合并累积时间的平均值。
Max cumulative merge time across primary shards	跨主分片索引合并累积时间的最大值。
Cumulative merge throttle time of primary shards	主分片的累积合并限制时间，也是指线程消耗CPU的时间总和。
Min cumulative merge throttle time across primary shards	跨主分片索引合并累积的时间最小值，也是指线程消耗CPU的时间总和。
Median cumulative merge throttle time across primary shards	跨主分片索引合并累积的时间平均值，也是指线程消耗CPU的时间总和。
Max cumulative merge throttle time across primary shards	跨主分片索引合并累积的时间最大值，也是指线程消耗CPU的时间总和。
Cumulative refresh time of primary shards	主分片累积刷新的时间，也是指线程消耗CPU的时间总和。
Cumulative refresh count of primary shards	主分片累积刷新的次数。
Min cumulative refresh time across primary shards	跨主分片索引刷新时间的最小值。
Median cumulative refresh time across primary shards	跨主分片索引刷新时间的平均值。
Max cumulative refresh time across primary shards	跨主分片索引刷新时间的最大值。
Cumulative flush time of primary shards	主分片索引事务数据从缓存冲洗到磁盘的累积时间，也是指线程消耗CPU的时间总和。
Cumulative flush count of primary shards	主分片索引事务数据从缓存冲洗到磁盘的累积次数。
Min cumulative flush time across primary shards	跨主分片索引，从缓存冲洗到磁盘累积时间的最小值，也是指线程消耗CPU的时间总和。
Median cumulative flush time	

Median cumulative flush time across primary shards	跨主分片索引，从缓存冲洗到磁盘累积时间的平均值，也是指线程消耗CPU的时间总和。
Max cumulative flush time across primary shards	跨主分片索引，从缓存冲洗到磁盘累积时间的最大值，也是指线程消耗CPU的时间总和。
Store size	索引的大小。不包括 translog和副本分片。
Translog size	传输日志大小。
Heap used for segments	所有主分片索引的segment数占用heap的量。
Heap used for doc values	所有主分片索引的文档数占用heap的量。
Heap used for terms	所有主分片索引的terms因子占用heap的量。
Heap used for norms	所有主分片索引的norms因子占用heap的量。
Heap used for points	所有主分片索引的points占用heap的量。
Heap used for stored fields	所有主分片索引的字段存储使用heap的量。
Segment count	所有主分片索引的segments数。
Total Young Gen GC	整个集群中年轻代垃圾收集器的总运行时间。
Total Old Gen GC	整个集群老年轻代垃圾收集器的总运行时间。
Min Throughput	每个任务的最小吞吐量，即QPS的最小值。
Median Throughput	每个任务的平均吞吐量，即QPS的平均值。
Max Throughput	每个任务的最大吞吐量，即QPS的最大值。
50th percentile latency	50%的完整请求所经历的时间。
90th percentile latency	90%的完整请求所经历的时间。
99th percentile latency	99%的完整请求所经历的时间。
100th percentile latency	100%的完整请求所经历的时间。
50th percentile service time	50%的请求所经历的服务时间。
90th percentile service time	90%的请求所经历的服务时间。
99th percentile service time	99%的请求所经历的服务时间。
100th percentile service time	100%的请求所经历的服务时间。
error rate	索引的每个任务的响应错误率。

压测中 Task 相关说明参考rally官方文档：https://esrally.readthedocs.io/en/stable/track.html#operations

4核16GB512GB通用型的3个数据节点实例基准性能指标

4核16GB 512GB通用型的3个数据节点实例基准性能测试

使用ESRally工具对数据节点规格为4核16GB的BES集群实例（7.10.2版本）进行基准测试。

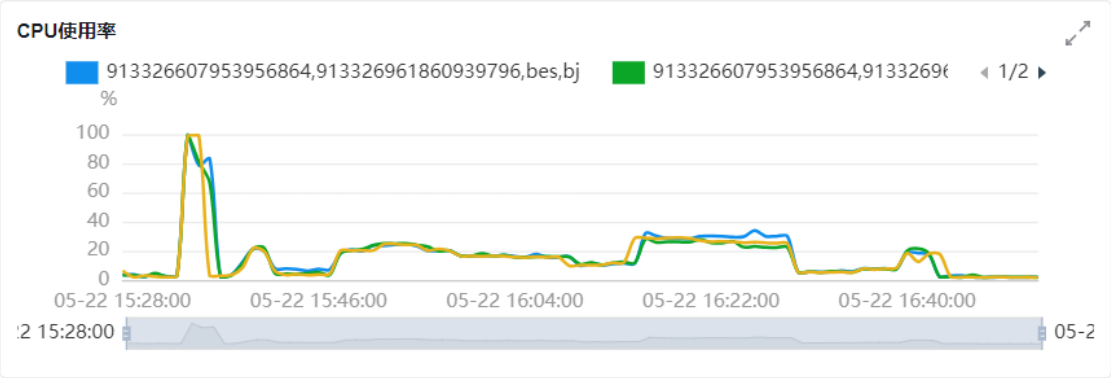
压测配置

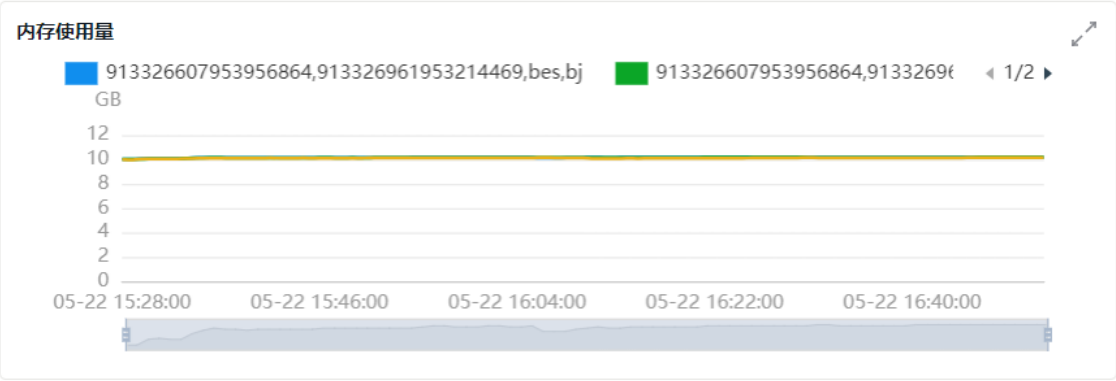
项目	说明
实例配置	版本：BES集群实例（7.10.2版本） 数据节点规格：4核16GB 数据节点存储性能级别：增强型SSD_PL1 数据单节点存储空间：512GB 数据节点数量：3个
ESRally配置	使用ESRally的默认配置tracks。
数据集	使用ESRally的预置数据集Geonames模拟测试场景，数据文档总数11396503。获取数据集，请查看ES Rally Hub获取。
分片数	主分片：6 副本数：0
bulk_size	每次批量操作提交5000个文档。
bulk_indexing_clients	同时执行批量索引操作的客户端数量为8个。
index_append warmup time	从默认的120s修改为30s，当集群配置高时，index_append在预热启动期间就完成了。

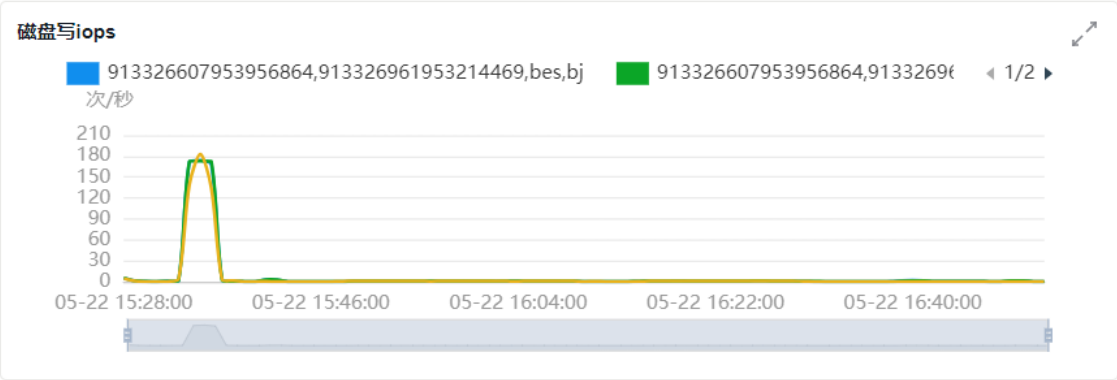
压测结果

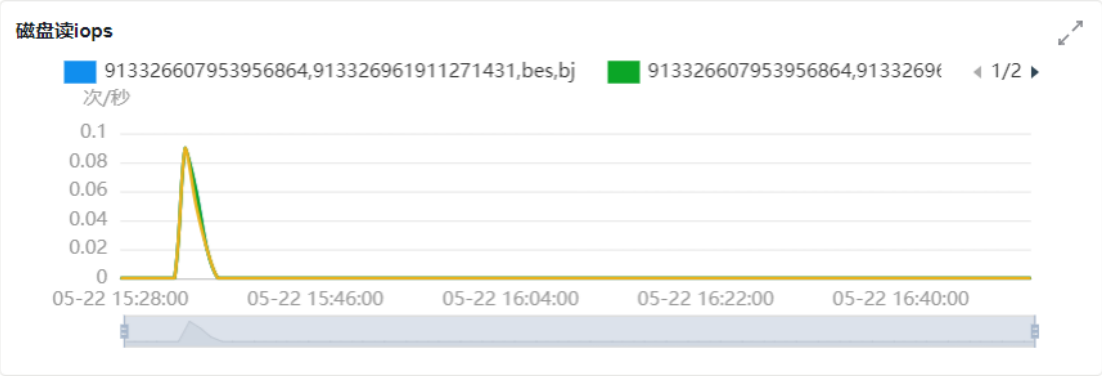
压测结果仅供参考，无法代表实际生产中写入查询情况，建议您结合业务生产数据进行压测。

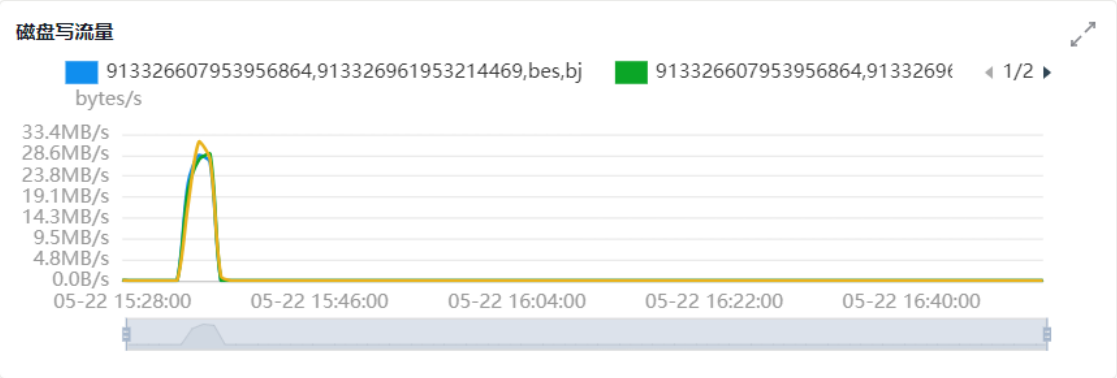
主要压测指标结果

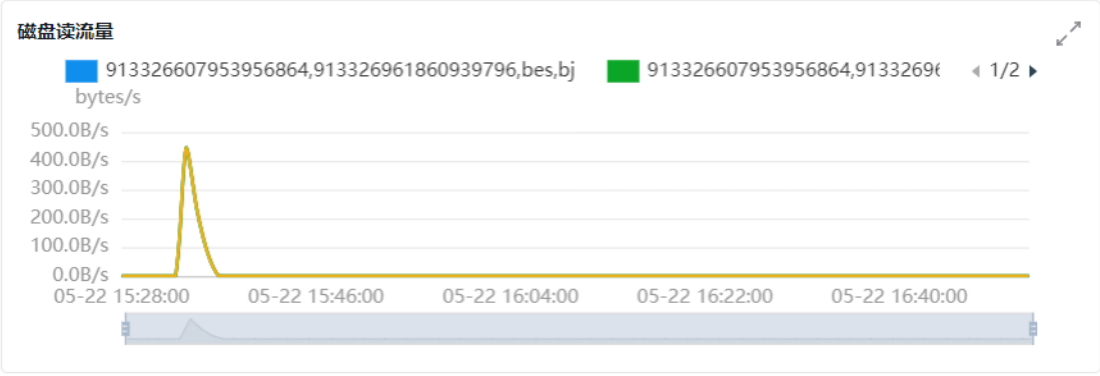


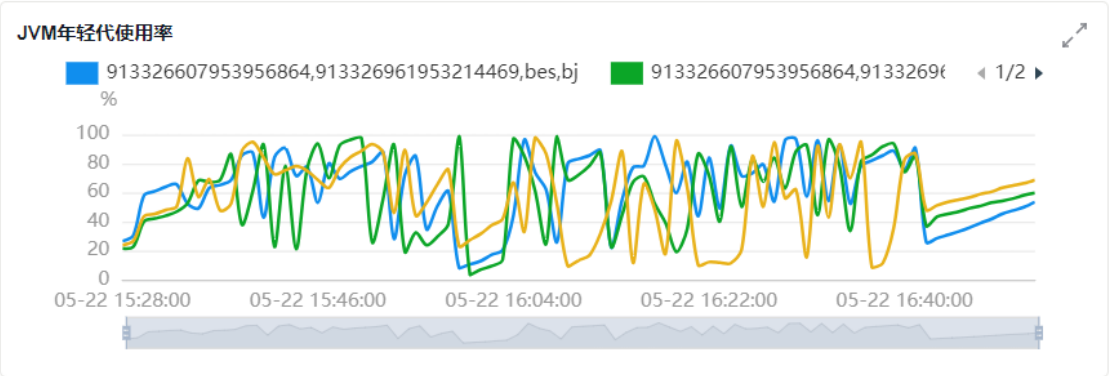


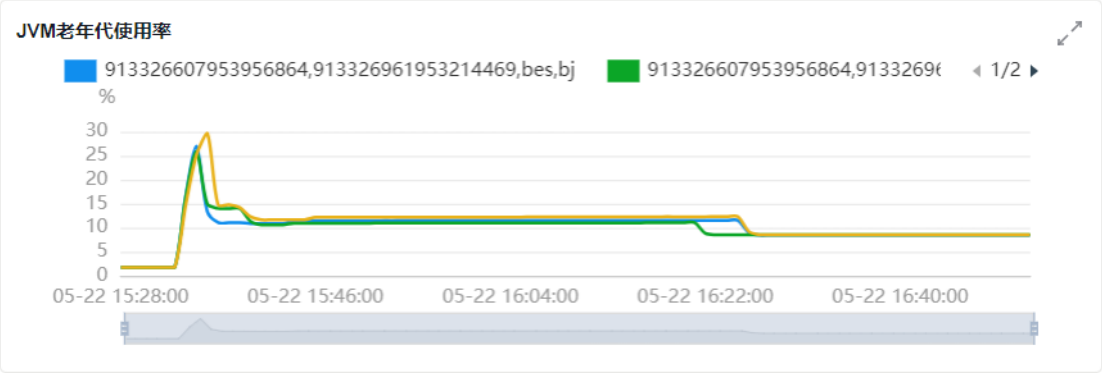

磁盘写iops
913326607953956864,913326961953214469,bes,bj
913326607953956864,9133269(
1/2
次/秒
210
180
150
120
90
60
30
0
05-22 15:28:00
05-22 15:46:00
05-22 16:04:00
05-22 16:22:00
05-22 16:40:00

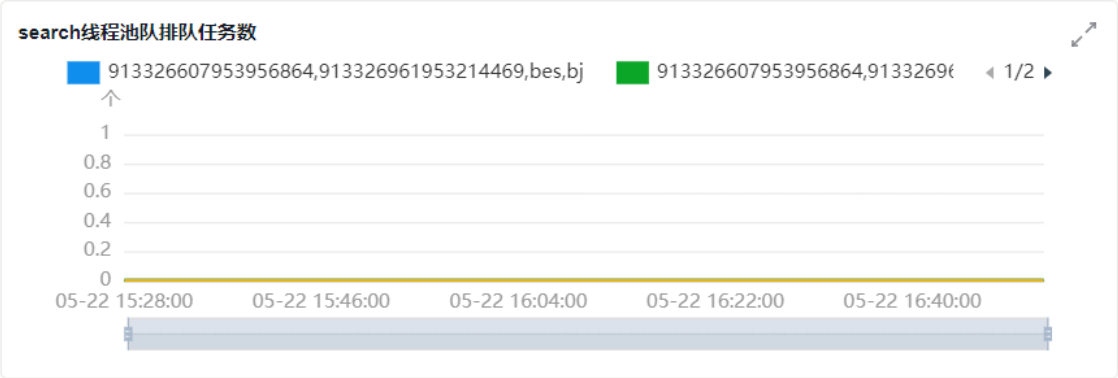

磁盘读iops
913326607953956864,913326961911271431,bes,bj
913326607953956864,9133269(
1/2
次/秒
0.1
0.08
0.06
0.04
0.02
0
05-22 15:28:00
05-22 15:46:00
05-22 16:04:00
05-22 16:22:00
05-22 16:40:00

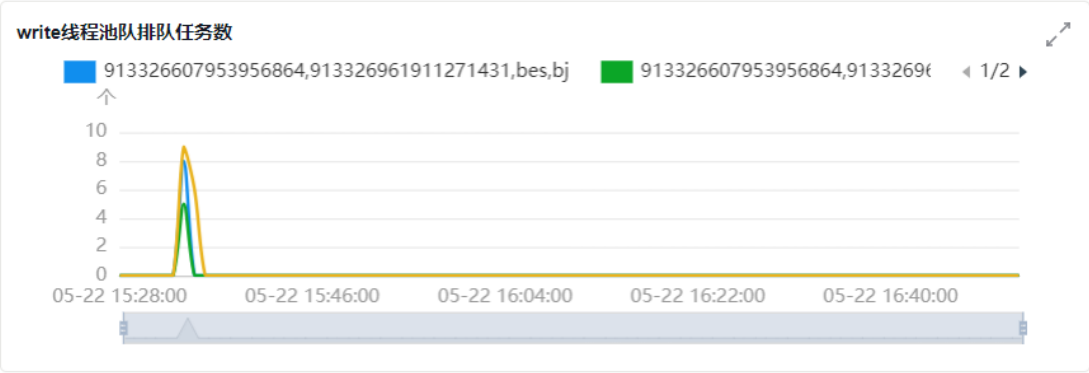

磁盘写流量
913326607953956864,913326961953214469,bes,bj
913326607953956864,9133269(
1/2
bytes/s
33.4MB/s
28.6MB/s
23.8MB/s
19.1MB/s
14.3MB/s
9.5MB/s
4.8MB/s
0.0B/s
05-22 15:28:00
05-22 15:46:00
05-22 16:04:00
05-22 16:22:00
05-22 16:40:00


磁盘读流量
913326607953956864,913326961860939796,bes,bj
913326607953956864,9133269(
1/2
bytes/s
500.0B/s
400.0B/s
300.0B/s
200.0B/s
100.0B/s
0.0B/s
05-22 15:28:00
05-22 15:46:00
05-22 16:04:00
05-22 16:22:00
05-22 16:40:00


JVM年轻代使用率
913326607953956864,913326961953214469,bes,bj
913326607953956864,9133269(
1/2
%
100
80
60
40
20
0
05-22 15:28:00
05-22 15:46:00
05-22 16:04:00
05-22 16:22:00
05-22 16:40:00







基准压测报告

Metric	Task	Value	Unit
Cumulative indexing time of primary shards		14.5433	min
Min cumulative indexing time across primary shards		2.30432	min
Median cumulative indexing time across primary shards		2.43925	min
Max cumulative indexing time across primary shards		2.5264	min
Cumulative indexing throttle time of primary shards		0	min
Min cumulative indexing throttle time across primary shards		0	min
Median cumulative indexing throttle time across primary shards		0	min
Max cumulative indexing throttle time across primary shards		0	min
Cumulative merge time of primary shards		4.85768	min
Cumulative merge count of primary shards		24	
Min cumulative merge time across primary shards		0.649417	min
Median cumulative merge time across primary shards		0.844292	min
Max cumulative merge time across primary shards		0.894633	min
Cumulative merge throttle time of primary shards		1.00215	min

Min cumulative merge throttle time across primary shards		0.146617	min
Median cumulative merge throttle time across primary shards		0.155408	min
Max cumulative merge throttle time across primary shards		0.2133	min
Cumulative refresh time of primary shards		2.37373	min
Cumulative refresh count of primary shards		143	
Min cumulative refresh time across primary shards		0.359867	min
Median cumulative refresh time across primary shards		0.4016	min
Max cumulative refresh time across primary shards		0.423083	min
Cumulative flush time of primary shards		0.196717	min
Cumulative flush count of primary shards		12	
Min cumulative flush time across primary shards		0.0239667	min
Median cumulative flush time across primary shards		0.0248167	min
Max cumulative flush time across primary shards		0.0509833	min
Total Young Gen GC time		9.796	s
Total Young Gen GC count		1464	
Total Old Gen GC time		0	s
Total Old Gen GC count		0	
Store size		2.92598	GB
Translog size		3.07336e-07	GB
Heap used for segments		0.894981	MB
Heap used for doc values		0.0564804	MB
Heap used for terms		0.684509	MB
Heap used for norms		0.0930786	MB
Heap used for points		0	MB
Heap used for stored fields		0.0609131	MB
Segment count		120	
Total Ingest Pipeline count		0	
Total Ingest Pipeline time		0	s
Total Ingest Pipeline failed		0	
Min Throughput	index-append	86946.4	docs/s
Mean Throughput	index-append	87122.7	docs/s
Median Throughput	index-append	87019.4	docs/s
Max Throughput	index-append	87530	docs/s
50th percentile latency	index-append	295.267	ms
90th percentile latency	index-append	616.566	ms
99th percentile latency	index-append	1563.13	ms
100th percentile latency	index-append	1976.91	ms
50th percentile service time	index-append	295.267	ms
90th percentile service time	index-append	616.566	ms
99th percentile service time	index-append	1563.13	ms

100th percentile service time	index-append	1976.91	ms
error rate	index-append	0	%
100th percentile latency	refresh-after-index	60022.4	ms
100th percentile service time	refresh-after-index	60022.4	ms
error rate	refresh-after-index	100	%
100th percentile latency	refresh-after-force-merge	60953.9	ms
100th percentile service time	refresh-after-force-merge	60953.9	ms
error rate	refresh-after-force-merge	100	%
Min Throughput	index-stats	90.02	ops/s
Mean Throughput	index-stats	90.03	ops/s
Median Throughput	index-stats	90.03	ops/s
Max Throughput	index-stats	90.05	ops/s
50th percentile latency	index-stats	3.00853	ms
90th percentile latency	index-stats	3.92021	ms
99th percentile latency	index-stats	5.14182	ms
99.9th percentile latency	index-stats	19.629	ms
100th percentile latency	index-stats	21.6033	ms
50th percentile service time	index-stats	1.72949	ms
90th percentile service time	index-stats	1.96231	ms
99th percentile service time	index-stats	3.50899	ms
99.9th percentile service time	index-stats	14.7519	ms
100th percentile service time	index-stats	16.3908	ms
error rate	index-stats	0	%
Min Throughput	node-stats	89.76	ops/s
Mean Throughput	node-stats	90	ops/s
Median Throughput	node-stats	90.02	ops/s
Max Throughput	node-stats	90.07	ops/s
50th percentile latency	node-stats	3.45345	ms
90th percentile latency	node-stats	4.01962	ms
99th percentile latency	node-stats	5.97835	ms
99.9th percentile latency	node-stats	13.9983	ms
100th percentile latency	node-stats	14.9775	ms
50th percentile service time	node-stats	2.5337	ms
90th percentile service time	node-stats	2.91972	ms
99th percentile service time	node-stats	4.63463	ms
99.9th percentile service time	node-stats	13.3918	ms
100th percentile service time	node-stats	14.3051	ms
error rate	node-stats	0	%
Min Throughput	default	50.02	ops/s
Mean Throughput	default	50.04	ops/s

Median Throughput	default	50.03	ops/s
Max Throughput	default	50.06	ops/s
50th percentile latency	default	3.34432	ms
90th percentile latency	default	4.55729	ms
99th percentile latency	default	8.23355	ms
99.9th percentile latency	default	17.0691	ms
100th percentile latency	default	19.7646	ms
50th percentile service time	default	2.39093	ms
90th percentile service time	default	2.78058	ms
99th percentile service time	default	7.33514	ms
99.9th percentile service time	default	16.4651	ms
100th percentile service time	default	18.552	ms
error rate	default	0	%
Min Throughput	term	99.92	ops/s
Mean Throughput	term	99.95	ops/s
Median Throughput	term	99.96	ops/s
Max Throughput	term	99.97	ops/s
50th percentile latency	term	3.46153	ms
90th percentile latency	term	4.02558	ms
99th percentile latency	term	13.9715	ms
99.9th percentile latency	term	37.4162	ms
100th percentile latency	term	41.96	ms
50th percentile service time	term	2.4764	ms
90th percentile service time	term	2.86928	ms
99th percentile service time	term	6.61472	ms
99.9th percentile service time	term	15.6629	ms
100th percentile service time	term	32.0975	ms
error rate	term	0	%
Min Throughput	phrase	109.83	ops/s
Mean Throughput	phrase	109.9	ops/s
Median Throughput	phrase	109.91	ops/s
Max Throughput	phrase	109.94	ops/s
50th percentile latency	phrase	3.55752	ms
90th percentile latency	phrase	4.09292	ms
99th percentile latency	phrase	23.3495	ms
99.9th percentile latency	phrase	35.3133	ms
100th percentile latency	phrase	40.0299	ms
50th percentile service time	phrase	2.54994	ms
90th percentile service time	phrase	2.9555	ms

99th percentile service time	phrase	8.36362	ms
99.9th percentile service time	phrase	25.5243	ms
100th percentile service time	phrase	27.4925	ms
error rate	phrase	0	%
Min Throughput	country_agg_uncached	3	ops/s
Mean Throughput	country_agg_uncached	3	ops/s
Median Throughput	country_agg_uncached	3	ops/s
Max Throughput	country_agg_uncached	3	ops/s
50th percentile latency	country_agg_uncached	123.319	ms
90th percentile latency	country_agg_uncached	136.326	ms
99th percentile latency	country_agg_uncached	207.354	ms
100th percentile latency	country_agg_uncached	211.931	ms
50th percentile service time	country_agg_uncached	121.843	ms
90th percentile service time	country_agg_uncached	134.62	ms
99th percentile service time	country_agg_uncached	205.798	ms
100th percentile service time	country_agg_uncached	210.959	ms
error rate	country_agg_uncached	0	%
Min Throughput	country_agg_cached	98.82	ops/s
Mean Throughput	country_agg_cached	99.13	ops/s
Median Throughput	country_agg_cached	99.16	ops/s
Max Throughput	country_agg_cached	99.35	ops/s
50th percentile latency	country_agg_cached	2.91306	ms
90th percentile latency	country_agg_cached	3.36731	ms
99th percentile latency	country_agg_cached	9.1527	ms
99.9th percentile latency	country_agg_cached	18.0531	ms
100th percentile latency	country_agg_cached	21.6629	ms
50th percentile service time	country_agg_cached	1.955	ms
90th percentile service time	country_agg_cached	2.209	ms
99th percentile service time	country_agg_cached	4.71351	ms
99.9th percentile service time	country_agg_cached	17.042	ms
100th percentile service time	country_agg_cached	20.7267	ms
error rate	country_agg_cached	0	%
Min Throughput	scroll	20.05	pages/s
Mean Throughput	scroll	20.06	pages/s
Median Throughput	scroll	20.05	pages/s
Max Throughput	scroll	20.07	pages/s
50th percentile latency	scroll	238.026	ms
90th percentile latency	scroll	290.066	ms
99th percentile latency	scroll	302.243	ms
100th percentile latency	scroll	305.624	ms

50th percentile service time	scroll	235.386	ms
90th percentile service time	scroll	287.317	ms
99th percentile service time	scroll	299.288	ms
100th percentile service time	scroll	304.09	ms
error rate	scroll	0	%
Min Throughput	expression	1.5	ops/s
Mean Throughput	expression	1.5	ops/s
Median Throughput	expression	1.5	ops/s
Max Throughput	expression	1.5	ops/s
50th percentile latency	expression	235.018	ms
90th percentile latency	expression	258.815	ms
99th percentile latency	expression	342.362	ms
100th percentile latency	expression	356.749	ms
50th percentile service time	expression	233.551	ms
90th percentile service time	expression	256.986	ms
99th percentile service time	expression	340.774	ms
100th percentile service time	expression	355.308	ms
error rate	expression	0	%
Min Throughput	painless_static	1.4	ops/s
Mean Throughput	painless_static	1.4	ops/s
Median Throughput	painless_static	1.4	ops/s
Max Throughput	painless_static	1.4	ops/s
50th percentile latency	painless_static	318.54	ms
90th percentile latency	painless_static	381.481	ms
99th percentile latency	painless_static	403.241	ms
100th percentile latency	painless_static	438.93	ms
50th percentile service time	painless_static	317.083	ms
90th percentile service time	painless_static	379.599	ms
99th percentile service time	painless_static	401.565	ms
100th percentile service time	painless_static	436.95	ms
error rate	painless_static	0	%
Min Throughput	painless_dynamic	1.4	ops/s
Mean Throughput	painless_dynamic	1.4	ops/s
Median Throughput	painless_dynamic	1.4	ops/s
Max Throughput	painless_dynamic	1.4	ops/s
50th percentile latency	painless_dynamic	267.905	ms
90th percentile latency	painless_dynamic	278.47	ms
99th percentile latency	painless_dynamic	372.193	ms
100th percentile latency	painless_dynamic	380.946	ms
50th percentile service time	painless_dynamic	266.039	ms

50th percentile service time	painless_dynamic	266.039	ms
90th percentile service time	painless_dynamic	276.793	ms
99th percentile service time	painless_dynamic	370.589	ms
100th percentile service time	painless_dynamic	379.55	ms
error rate	painless_dynamic	0	%
Min Throughput	decay_geo_gauss_function_score	1	ops/s
Mean Throughput	decay_geo_gauss_function_score	1	ops/s
Median Throughput	decay_geo_gauss_function_score	1	ops/s
Max Throughput	decay_geo_gauss_function_score	1	ops/s
50th percentile latency	decay_geo_gauss_function_score	299.307	ms
90th percentile latency	decay_geo_gauss_function_score	303.328	ms
99th percentile latency	decay_geo_gauss_function_score	314.982	ms
100th percentile latency	decay_geo_gauss_function_score	320.353	ms
50th percentile service time	decay_geo_gauss_function_score	296.824	ms
90th percentile service time	decay_geo_gauss_function_score	300.809	ms
99th percentile service time	decay_geo_gauss_function_score	312.501	ms
100th percentile service time	decay_geo_gauss_function_score	317.48	ms
error rate	decay_geo_gauss_function_score	0	%
Min Throughput	decay_geo_gauss_script_score	1	ops/s
Mean Throughput	decay_geo_gauss_script_score	1	ops/s
Median Throughput	decay_geo_gauss_script_score	1	ops/s
Max Throughput	decay_geo_gauss_script_score	1	ops/s
50th percentile latency	decay_geo_gauss_script_score	279.925	ms
90th percentile latency	decay_geo_gauss_script_score	283.761	ms
99th percentile latency	decay_geo_gauss_script_score	306.952	ms
100th percentile latency	decay_geo_gauss_script_score	308.536	ms
50th percentile service time	decay_geo_gauss_script_score	278.146	ms
90th percentile service time	decay_geo_gauss_script_score	282.082	ms
99th percentile service time	decay_geo_gauss_script_score	304.703	ms
100th percentile service time	decay_geo_gauss_script_score	306.354	ms
error rate	decay_geo_gauss_script_score	0	%
Min Throughput	field_value_function_score	1.5	ops/s
Mean Throughput	field_value_function_score	1.5	ops/s
Median Throughput	field_value_function_score	1.5	ops/s
Max Throughput	field_value_function_score	1.51	ops/s
50th percentile latency	field_value_function_score	111.885	ms
90th percentile latency	field_value_function_score	120.971	ms
99th percentile latency	field_value_function_score	133.412	ms
100th percentile latency	field_value_function_score	148.74	ms
50th percentile service time	field_value_function_score	110.418	ms

90th percentile service time	field_value_function_score	119.445	ms
99th percentile service time	field_value_function_score	131.805	ms
100th percentile service time	field_value_function_score	146.663	ms
error rate	field_value_function_score	0	%
Min Throughput	field_value_script_score	1.5	ops/s
Mean Throughput	field_value_script_score	1.5	ops/s
Median Throughput	field_value_script_score	1.5	ops/s
Max Throughput	field_value_script_score	1.51	ops/s
50th percentile latency	field_value_script_score	129.386	ms
90th percentile latency	field_value_script_score	133.116	ms
99th percentile latency	field_value_script_score	137.728	ms
100th percentile latency	field_value_script_score	137.875	ms
50th percentile service time	field_value_script_score	126.736	ms
90th percentile service time	field_value_script_score	131.418	ms
99th percentile service time	field_value_script_score	135.124	ms
100th percentile service time	field_value_script_score	135.888	ms
error rate	field_value_script_score	0	%
Min Throughput	large_terms	1.1	ops/s
Mean Throughput	large_terms	1.1	ops/s
Median Throughput	large_terms	1.1	ops/s
Max Throughput	large_terms	1.1	ops/s
50th percentile latency	large_terms	577.719	ms
90th percentile latency	large_terms	607.921	ms
99th percentile latency	large_terms	657.43	ms
100th percentile latency	large_terms	667.958	ms
50th percentile service time	large_terms	570.927	ms
90th percentile service time	large_terms	601.164	ms
99th percentile service time	large_terms	651.043	ms
100th percentile service time	large_terms	661.045	ms
error rate	large_terms	0	%
Min Throughput	large_filtered_terms	1.1	ops/s
Mean Throughput	large_filtered_terms	1.1	ops/s
Median Throughput	large_filtered_terms	1.1	ops/s
Max Throughput	large_filtered_terms	1.1	ops/s
50th percentile latency	large_filtered_terms	580.262	ms
90th percentile latency	large_filtered_terms	618.11	ms
99th percentile latency	large_filtered_terms	649.98	ms
100th percentile latency	large_filtered_terms	653.566	ms
50th percentile service time	large_filtered_terms	572.639	ms
90th percentile service time	large_filtered_terms	610.963	ms

99th percentile service time	large_filtered_terms	642.538	ms
100th percentile service time	large_filtered_terms	646.11	ms
error rate	large_filtered_terms	0	%
Min Throughput	large_prohibited_terms	1.1	ops/s
Mean Throughput	large_prohibited_terms	1.1	ops/s
Median Throughput	large_prohibited_terms	1.1	ops/s
Max Throughput	large_prohibited_terms	1.1	ops/s
50th percentile latency	large_prohibited_terms	559.511	ms
90th percentile latency	large_prohibited_terms	603.712	ms
99th percentile latency	large_prohibited_terms	637.212	ms
100th percentile latency	large_prohibited_terms	651.246	ms
50th percentile service time	large_prohibited_terms	552.202	ms
90th percentile service time	large_prohibited_terms	596.589	ms
99th percentile service time	large_prohibited_terms	629.658	ms
100th percentile service time	large_prohibited_terms	643.627	ms
error rate	large_prohibited_terms	0	%
Min Throughput	desc_sort_population	1.5	ops/s
Mean Throughput	desc_sort_population	1.51	ops/s
Median Throughput	desc_sort_population	1.51	ops/s
Max Throughput	desc_sort_population	1.51	ops/s
50th percentile latency	desc_sort_population	53.548	ms
90th percentile latency	desc_sort_population	64.0532	ms
99th percentile latency	desc_sort_population	74.0357	ms
100th percentile latency	desc_sort_population	93.0945	ms
50th percentile service time	desc_sort_population	51.8361	ms
90th percentile service time	desc_sort_population	62.7609	ms
99th percentile service time	desc_sort_population	72.2838	ms
100th percentile service time	desc_sort_population	91.4106	ms
error rate	desc_sort_population	0	%
Min Throughput	asc_sort_population	1.5	ops/s
Mean Throughput	asc_sort_population	1.51	ops/s
Median Throughput	asc_sort_population	1.51	ops/s
Max Throughput	asc_sort_population	1.51	ops/s
50th percentile latency	asc_sort_population	52.9451	ms
90th percentile latency	asc_sort_population	57.6287	ms
99th percentile latency	asc_sort_population	91.7893	ms
100th percentile latency	asc_sort_population	92.9107	ms
50th percentile service time	asc_sort_population	51.6168	ms
90th percentile service time	asc_sort_population	55.8232	ms
99th percentile service time	asc_sort_population	90.0983	ms

99th percentile service time	asc_sort_population	90.0983	ms
100th percentile service time	asc_sort_population	91.3728	ms
error rate	asc_sort_population	0	%
Min Throughput	asc_sort_with_after_population	1.5	ops/s
Mean Throughput	asc_sort_with_after_population	1.51	ops/s
Median Throughput	asc_sort_with_after_population	1.51	ops/s
Max Throughput	asc_sort_with_after_population	1.51	ops/s
50th percentile latency	asc_sort_with_after_population	75.1255	ms
90th percentile latency	asc_sort_with_after_population	81.9797	ms
99th percentile latency	asc_sort_with_after_population	140.404	ms
100th percentile latency	asc_sort_with_after_population	145.858	ms
50th percentile service time	asc_sort_with_after_population	73.6456	ms
90th percentile service time	asc_sort_with_after_population	79.9653	ms
99th percentile service time	asc_sort_with_after_population	138.833	ms
100th percentile service time	asc_sort_with_after_population	144.456	ms
error rate	asc_sort_with_after_population	0	%
Min Throughput	desc_sort_geonameid	6.01	ops/s
Mean Throughput	desc_sort_geonameid	6.01	ops/s
Median Throughput	desc_sort_geonameid	6.01	ops/s
Max Throughput	desc_sort_geonameid	6.02	ops/s
50th percentile latency	desc_sort_geonameid	7.29081	ms
90th percentile latency	desc_sort_geonameid	8.29137	ms
99th percentile latency	desc_sort_geonameid	17.2185	ms
100th percentile latency	desc_sort_geonameid	21.4305	ms
50th percentile service time	desc_sort_geonameid	6.04519	ms
90th percentile service time	desc_sort_geonameid	7.04462	ms
99th percentile service time	desc_sort_geonameid	15.5322	ms
100th percentile service time	desc_sort_geonameid	19.9791	ms
error rate	desc_sort_geonameid	0	%
Min Throughput	desc_sort_with_after_geonameid	6.01	ops/s
Mean Throughput	desc_sort_with_after_geonameid	6.01	ops/s
Median Throughput	desc_sort_with_after_geonameid	6.01	ops/s
Max Throughput	desc_sort_with_after_geonameid	6.02	ops/s
50th percentile latency	desc_sort_with_after_geonameid	64.1927	ms
90th percentile latency	desc_sort_with_after_geonameid	76.1211	ms
99th percentile latency	desc_sort_with_after_geonameid	120.851	ms
100th percentile latency	desc_sort_with_after_geonameid	136.744	ms
50th percentile service time	desc_sort_with_after_geonameid	63.2755	ms
90th percentile service time	desc_sort_with_after_geonameid	74.9516	ms
99th percentile service time	desc_sort_with_after_geonameid	119.59	ms

100th percentile service time	desc_sort_with_after_geonameid	135.115	ms
error rate	desc_sort_with_after_geonameid	0	%
Min Throughput	asc_sort_geonameid	6.02	ops/s
Mean Throughput	asc_sort_geonameid	6.02	ops/s
Median Throughput	asc_sort_geonameid	6.02	ops/s
Max Throughput	asc_sort_geonameid	6.03	ops/s
50th percentile latency	asc_sort_geonameid	6.46071	ms
90th percentile latency	asc_sort_geonameid	7.1062	ms
99th percentile latency	asc_sort_geonameid	7.71702	ms
100th percentile latency	asc_sort_geonameid	10.3483	ms
50th percentile service time	asc_sort_geonameid	5.28308	ms
90th percentile service time	asc_sort_geonameid	5.6433	ms
99th percentile service time	asc_sort_geonameid	6.56002	ms
100th percentile service time	asc_sort_geonameid	9.29886	ms
error rate	asc_sort_geonameid	0	%
Min Throughput	asc_sort_with_after_geonameid	6.01	ops/s
Mean Throughput	asc_sort_with_after_geonameid	6.02	ops/s
Median Throughput	asc_sort_with_after_geonameid	6.01	ops/s
Max Throughput	asc_sort_with_after_geonameid	6.02	ops/s
50th percentile latency	asc_sort_with_after_geonameid	60.4825	ms
90th percentile latency	asc_sort_with_after_geonameid	70.2862	ms
99th percentile latency	asc_sort_with_after_geonameid	88.596	ms
100th percentile latency	asc_sort_with_after_geonameid	103.853	ms
50th percentile service time	asc_sort_with_after_geonameid	59.2317	ms
90th percentile service time	asc_sort_with_after_geonameid	68.858	ms
99th percentile service time	asc_sort_with_after_geonameid	87.2272	ms
100th percentile service time	asc_sort_with_after_geonameid	103.05	ms
error rate	asc_sort_with_after_geonameid	0	%

4核8GB512GB计算型的3个数据节点实例基准性能测试

4核8GB 512GB计算型的3个数据节点实例基准性能测试

使用ESRally工具对数据节点规格为4核8GB的BES集群实例（7.10.2版本）进行基准测试。

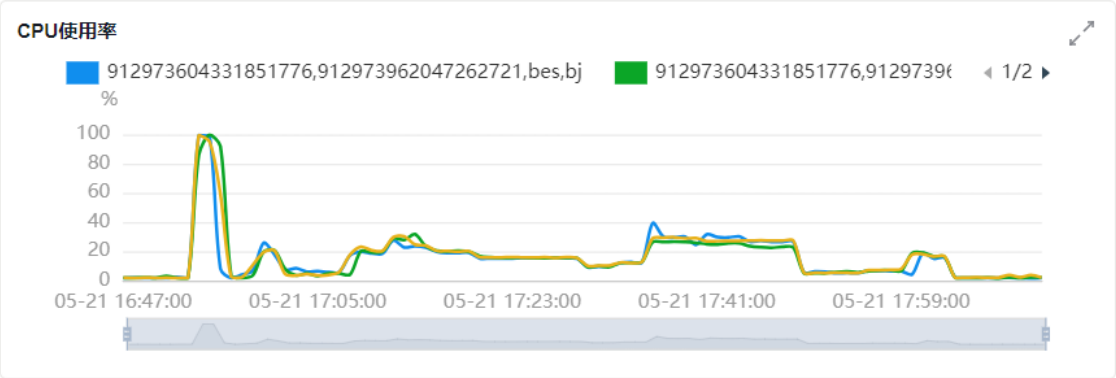
压测配置

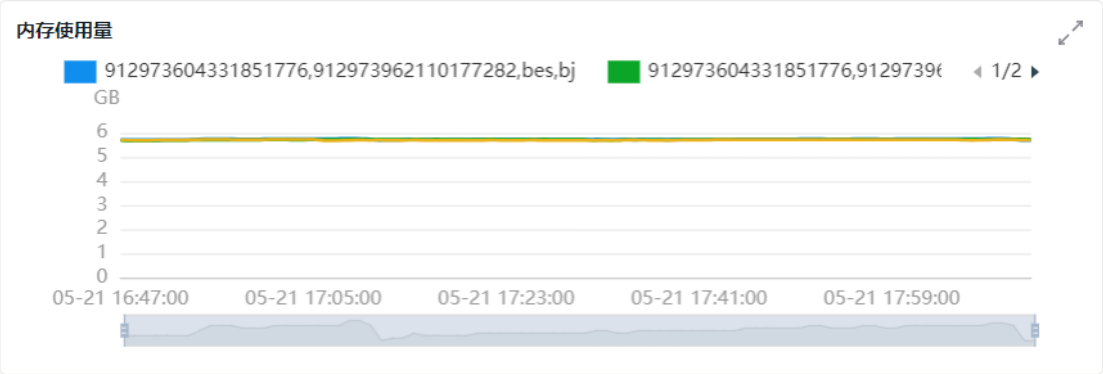
项目	说明
实例配置	版本：BES集群实例（7.10.2版本） 数据节点规格：4核8GB计算型 数据节点存储性能级别：增强型SSD_PL1 数据单节点存储空间：512GB 数据节点数量：3个
ESRally配置	使用ESRally的默认配置tracks。
数据集	使用ESRally的预置数据集Geonames模拟测试场景，文档总数11396503。获取数据集，请查看ES Rally Hub获取。
分片数	主分片：6 副本数：0
bulk_size	每次批量操作提交5000个文档。
bulk_indexing_clients	同时执行批量索引操作的客户端数量为8个。
index_append warmup time	从默认的120s修改为30s，因为当集群配置高时，index_append在预热启动期间就完成了。

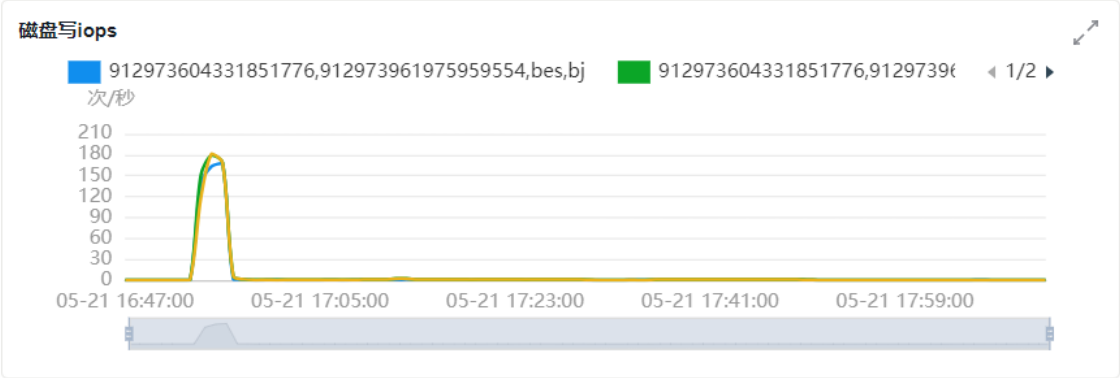
压测结果

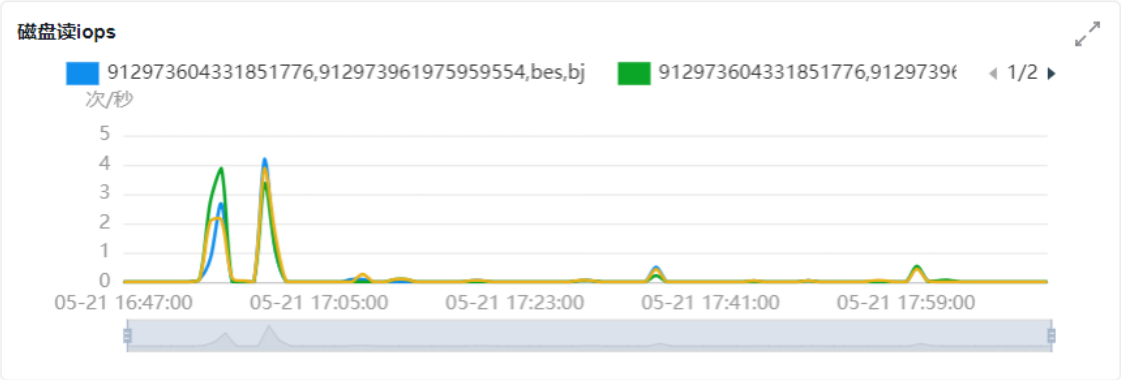
压测结果仅供参考，无法代表实际生产中写入查询情况，建议您结合业务生产数据进行压测。

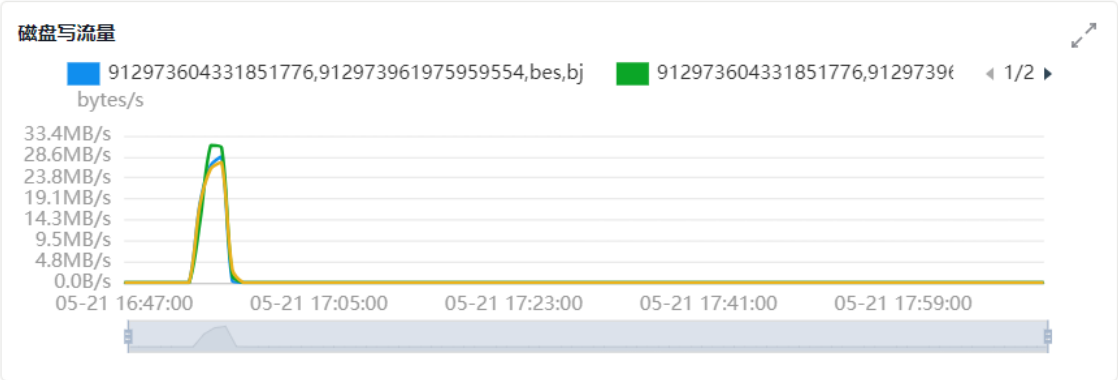
主要压测指标结果

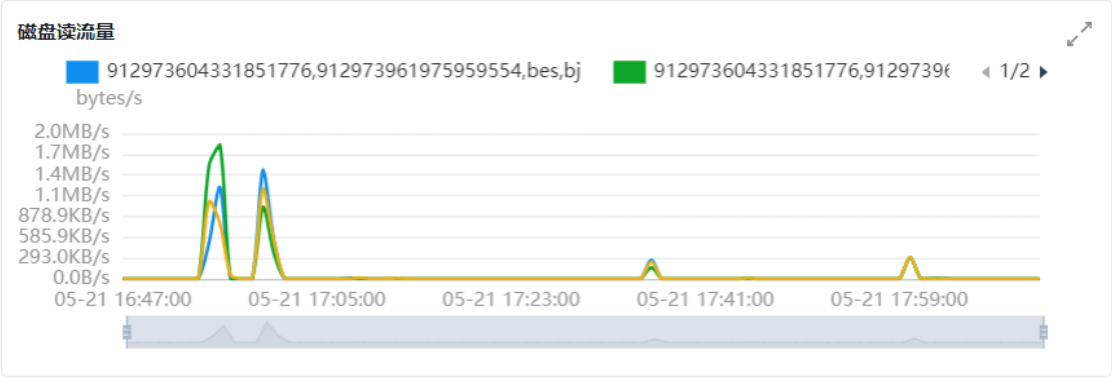


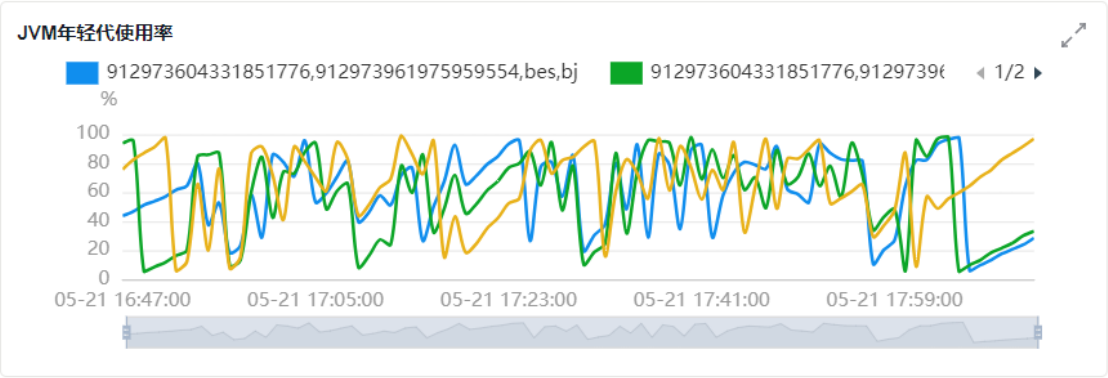


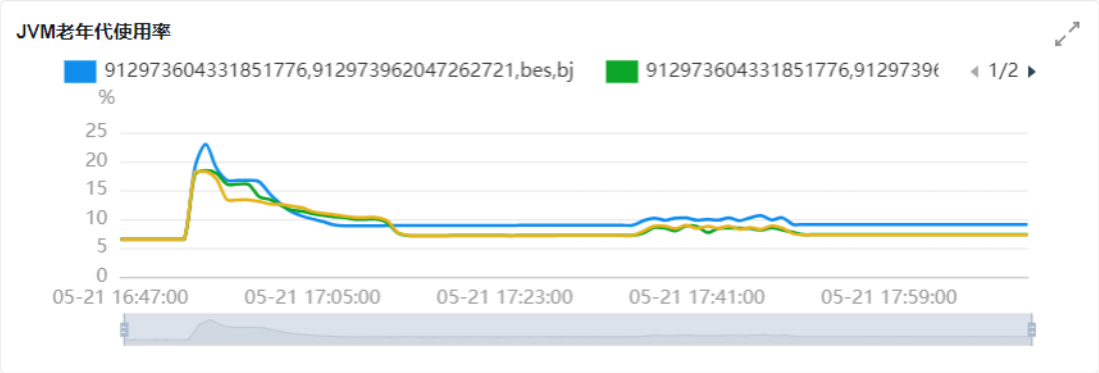

磁盘写iops
912973604331851776,912973961975959554,bes,bj
912973604331851776,9129739(
1/2
次/秒
210
180
150
120
90
60
30
0
05-21 16:47:00
05-21 17:05:00
05-21 17:23:00
05-21 17:41:00
05-21 17:59:00

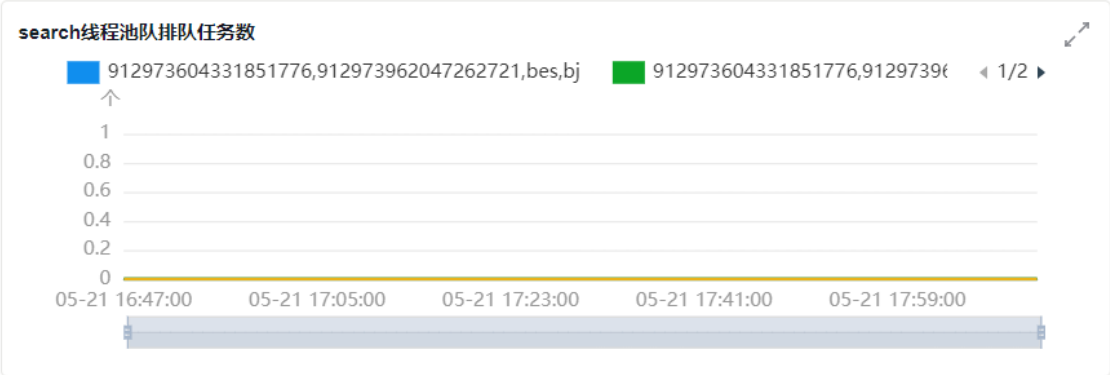

磁盘读iops
912973604331851776,912973961975959554,bes,bj
912973604331851776,9129739(
1/2
次/秒
5
4
3
2
1
0
05-21 16:47:00
05-21 17:05:00
05-21 17:23:00
05-21 17:41:00
05-21 17:59:00

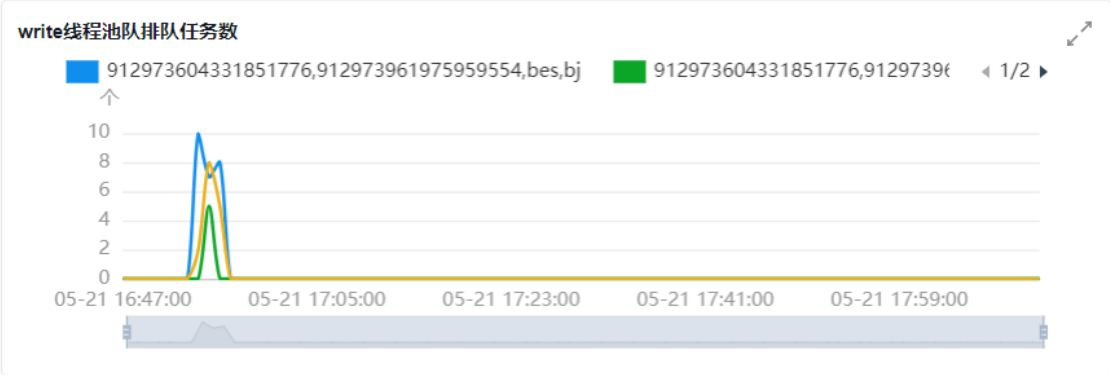

磁盘写流量
912973604331851776,912973961975959554,bes,bj
912973604331851776,9129739(
1/2
bytes/s
33.4MB/s
28.6MB/s
23.8MB/s
19.1MB/s
14.3MB/s
9.5MB/s
4.8MB/s
0.0B/s
05-21 16:47:00
05-21 17:05:00
05-21 17:23:00
05-21 17:41:00
05-21 17:59:00


磁盘读流量
912973604331851776,912973961975959554,bes,bj
912973604331851776,9129739(
1/2
bytes/s
2.0MB/s
1.7MB/s
1.4MB/s
1.1MB/s
878.9KB/s
585.9KB/s
293.0KB/s
0.0B/s
05-21 16:47:00
05-21 17:05:00
05-21 17:23:00
05-21 17:41:00
05-21 17:59:00


JVM年轻代使用率
912973604331851776,912973961975959554,bes,bj
912973604331851776,9129739(
1/2
%
100
80
60
40
20
0
05-21 16:47:00
05-21 17:05:00
05-21 17:23:00
05-21 17:41:00
05-21 17:59:00







基准压测报告 | Metric | Task | Value | Unit | |---:|---:|---:|---:| | Cumulative indexing time of primary shards | | 14.7237 | min | | Min cumulative indexing time across primary shards | | 2.37027 | min | | Median cumulative indexing time across primary shards | | 2.40307 | min | | Max cumulative indexing time across primary shards | | 2.59477 | min | | Cumulative indexing throttle time of primary shards | | 0 | min | | Min cumulative indexing throttle time across primary shards | | 0 | min | | Median cumulative indexing throttle time across primary shards | | 0 | min | | Max cumulative indexing throttle time across primary shards | | 0 | min | | Cumulative merge time of primary shards | | 5.38082 | min | | Cumulative merge count of primary shards | | 25 | | | Min cumulative merge time across primary shards | | 0.8093 | min | | Median cumulative merge time across primary shards | | 0.894842 | min | | Max cumulative merge time across primary shards | | 1.00987 | min | | Cumulative merge throttle time of primary shards | | 1.0408 | min | | Min cumulative merge throttle time across primary shards | | 0.1348 | min | | Median cumulative merge throttle time across primary shards | | 0.159975 | min | | Max cumulative merge throttle time across primary shards | | 0.22275 | min | | Cumulative refresh time of primary shards | | 2.3357 | min | | Cumulative refresh count of primary shards | | 148 | | | Min cumulative refresh time across primary shards | | 0.35635 | min | | Median cumulative refresh time across primary shards | | 0.3918 | min | | Max cumulative refresh time across primary shards | | 0.42765 | min | | Cumulative flush time of primary shards | | 0.210917 | min | | Cumulative flush count of primary shards | | 12 | | | Min cumulative flush time across primary shards | | 0.01755 | min | | Median cumulative flush time across primary shards | | 0.0403667 | min | | Max cumulative flush time across primary shards | | 0.0495 | min | | Total Young Gen GC time | | 14.133 | s | | Total Young Gen GC count | | 2901 | | | Total Old Gen GC time | | 0 | s | | Total Old Gen GC count | | 0 | | | Store size | | 2.90995 | GB | | Translog size | | 3.07336e-07 | GB | | Heap used for segments | | 0.903454 | MB | | Heap used for doc values | | 0.0789986 | MB | | Heap used for terms | | 0.672699 | MB | | Heap used for norms | | 0.0914307 | MB | | Heap used for points | | 0 | MB | | Heap used for stored fields

| | 0.0603256 | MB | | Segment count | | 119 | | | Total Ingest Pipeline count | | 0 | | | Total Ingest Pipeline time | | 0 | s | | Total Ingest Pipeline failed | | 0 | | | Min Throughput | index-append | 85856.2 | docs/s | | Mean Throughput | index-append | 86803.2 | docs/s | | Median Throughput | index-append | 86893.6 | docs/s | | Max Throughput | index-append | 87285.1 | docs/s | | 50th percentile latency | index-append | 336.954 | ms | | 90th percentile latency | index-append | 731.275 | ms | | 99th percentile latency | index-append | 1587.01 | ms | | 100th percentile latency | index-append | 2005.03 | ms | | 50th percentile service time | index-append | 336.954 | ms | | 90th percentile service time | index-append | 731.275 | ms | | 99th percentile service time | index-append | 1587.01 | ms | | 100th percentile service time | index-append | 2005.03 | ms | | error rate | index-append | 0 | % | | 100th percentile latency | refresh-after-index | 60210.2 | ms | | 100th percentile service time | refresh-after-index | 60210.2 | ms | | error rate | refresh-after-index | 100 | % | | 100th percentile latency | refresh-after-force-merge | 60136.1 | ms | | 100th percentile service time | refresh-after-force-merge | 60136.1 | ms | | error rate | refresh-after-force-merge | 100 | % | | Min Throughput | index-stats | 90.03 | ops/s | | Mean Throughput | index-stats | 90.05 | ops/s | | Median Throughput | index-stats | 90.05 | ops/s | | Max Throughput | index-stats | 90.09 | ops/s | | 50th percentile latency | index-stats | 2.88202 | ms | | 90th percentile latency | index-stats | 3.78057 | ms | | 99th percentile latency | index-stats | 4.43655 | ms | | 99.9th percentile latency | index-stats | 10.1314 | ms | | 100th percentile latency | index-stats | 12.89 | ms | | 50th percentile service time | index-stats | 1.56157 | ms | | 90th percentile service time | index-stats | 1.7524 | ms | | 99th percentile service time | index-stats | 3.45842 | ms | | 99.9th percentile service time | index-stats | 8.88811 | ms | | 100th percentile service time | index-stats | 12.2731 | ms | | error rate | index-stats | 0 | % | | Min Throughput | node-stats | 90.03 | ops/s | | Mean Throughput | node-stats | 90.07 | ops/s | | Median Throughput | node-stats | 90.04 | ops/s | | Max Throughput | node-stats | 90.17 | ops/s | | 50th percentile latency | node-stats | 3.23138 | ms | | 90th percentile latency | node-stats | 3.72642 | ms | | 99th percentile latency | node-stats | 9.92562 | ms | | 99.9th percentile latency | node-stats | 21.2769 | ms | | 100th percentile latency | node-stats | 23.0148 | ms | | 50th percentile service time | node-stats | 2.31196 | ms | | 90th percentile service time | node-stats | 2.5836 | ms | | 99th percentile service time | node-stats | 6.9647 | ms | | 99.9th percentile service time | node-stats | 14.0312 | ms | | 100th percentile service time | node-stats | 22.4266 | ms | | error rate | node-stats | 0 | % | | Min Throughput | default | 50.01 | ops/s | | Mean Throughput | default | 50.02 | ops/s | | Median Throughput | default | 50.02 | ops/s | | Max Throughput | default | 50.04 | ops/s | | 50th percentile latency | default | 3.27949 | ms | | 90th percentile latency | default | 4.50475 | ms | | 99th percentile latency | default | 9.26823 | ms | | 99.9th percentile latency | default | 21.1412 | ms | | 100th percentile latency | default | 22.0378 | ms | | 50th percentile service time | default | 2.15493 | ms | | 90th percentile service time | default | 2.49955 | ms | | 99th percentile service time | default | 8.75379 | ms | | 99.9th percentile service time | default | 20.1732 | ms | | 100th percentile service time | default | 20.4612 | ms | | error rate | default | 0 | % | | Min Throughput | term | 99.23 | ops/s | | Mean Throughput | term | 99.53 | ops/s | | Median Throughput | term | 99.56 | ops/s | | Max Throughput | term | 99.7 | ops/s | | 50th percentile latency | term | 3.41989 | ms | | 90th percentile latency | term | 3.8345 | ms | | 99th percentile latency | term | 15.0295 | ms | | 99.9th percentile latency | term | 34.5923 | ms | | 100th percentile latency | term | 35.9145 | ms | | 50th percentile service time | term | 2.33014 | ms | | 90th percentile service time | term | 2.63693 | ms | | 99th percentile service time | term | 6.76757 | ms | | 99.9th percentile service time | term | 22.3215 | ms | | 100th percentile service time | term | 22.4099 | ms | | error rate | term | 0 | % | | Min Throughput | phrase | 108.7 | ops/s | | Mean Throughput | phrase | 109.21 | ops/s | | Median Throughput | phrase | 109.28 | ops/s | | Max Throughput | phrase | 109.5 | ops/s | | 50th percentile latency | phrase | 3.12651 | ms | | 90th percentile latency | phrase | 3.63205 | ms | | 99th percentile latency | phrase | 18.2967 | ms | | 99.9th percentile latency | phrase | 38.804 | ms | | 100th percentile latency | phrase | 39.1649 | ms | | 50th percentile service time | phrase | 2.19042 | ms | | 90th percentile service time | phrase | 2.52546 | ms | | 99th percentile service time | phrase | 8.3174 | ms | | 99.9th percentile service time | phrase | 25.0611 | ms | | 100th percentile service time | phrase | 26.6727 | ms | | error rate | phrase | 0 | % | | Min Throughput | country_agg_uncached | 3 | ops/s | | Mean Throughput | country_agg_uncached | 3 | ops/s | | Median Throughput | country_agg_uncached | 3 | ops/s | | Max Throughput | country_agg_uncached | 3 | ops/s | | 50th percentile latency | country_agg_uncached | 121.606 | ms | | 90th percentile latency | country_agg_uncached | 129.095 | ms | | 99th percentile latency | country_agg_uncached | 139.924 | ms | | 100th percentile latency | country_agg_uncached | 194.642 | ms | | 50th percentile service time | country_agg_uncached | 120.152 | ms | | 90th percentile service time | country_agg_uncached | 127.552 | ms | | 99th percentile service time | country_agg_uncached | 137.707 | ms | | 100th percentile service time | country_agg_uncached | 192.181 | ms | | error rate | country_agg_uncached | 0 | % | | Min Throughput | country_agg_cached | 98.97 | ops/s | | Mean Throughput | country_agg_cached | 99.24 | ops/s | | Median Throughput | country_agg_cached | 99.27 | ops/s | | Max Throughput | country_agg_cached | 99.43 | ops/s | | 50th percentile latency |

country_agg_cached | 2.71151 | ms | | 90th percentile latency | country_agg_cached | 3.12794 | ms | | 99th percentile latency | country_agg_cached | 6.00165 | ms | | 99.9th percentile latency | country_agg_cached | 19.9261 | ms | | 100th percentile latency | country_agg_cached | 24.7657 | ms | | 50th percentile service time | country_agg_cached | 1.72649 | ms | | 90th percentile service time | country_agg_cached | 2.00915 | ms | | 99th percentile service time | country_agg_cached | 4.45331 | ms | | 99.9th percentile service time | country_agg_cached | 14.909 | ms | | 100th percentile service time | country_agg_cached | 15.2913 | ms | | error rate | country_agg_cached | 0 | % | | Min Throughput | scroll | 20.04 | pages/s | | Mean Throughput | scroll | 20.05 | pages/s | | Median Throughput | scroll | 20.05 | pages/s | | Max Throughput | scroll | 20.07 | pages/s | | 50th percentile latency | scroll | 247.022 | ms | | 90th percentile latency | scroll | 254.168 | ms | | 99th percentile latency | scroll | 270.668 | ms | | 100th percentile latency | scroll | 271.919 | ms | | 50th percentile service time | scroll | 244.647 | ms | | 90th percentile service time | scroll | 252.39 | ms | | 99th percentile service time | scroll | 268.858 | ms | | 100th percentile service time | scroll | 269.417 | ms | | error rate | scroll | 0 | % | | Min Throughput | expression | 1.5 | ops/s | | Mean Throughput | expression | 1.5 | ops/s | | Median Throughput | expression | 1.5 | ops/s | | Max Throughput | expression | 1.5 | ops/s | | 50th percentile latency | expression | 227.772 | ms | | 90th percentile latency | expression | 296.556 | ms | | 99th percentile latency | expression | 379.993 | ms | | 100th percentile latency | expression | 392.97 | ms | | 50th percentile service time | expression | 226.091 | ms | | 90th percentile service time | expression | 295.035 | ms | | 99th percentile service time | expression | 378.228 | ms | | 100th percentile service time | expression | 392.017 | ms | | error rate | expression | 0 | % | | Min Throughput | painless_static | 1.4 | ops/s | | Mean Throughput | painless_static | 1.4 | ops/s | | Median Throughput | painless_static | 1.4 | ops/s | | Max Throughput | painless_static | 1.4 | ops/s | | 50th percentile latency | painless_static | 318.866 | ms | | 90th percentile latency | painless_static | 381.073 | ms | | 99th percentile latency | painless_static | 436.901 | ms | | 100th percentile latency | painless_static | 497.268 | ms | | 50th percentile service time | painless_static | 317.356 | ms | | 90th percentile service time | painless_static | 379.478 | ms | | 99th percentile service time | painless_static | 434.922 | ms | | 100th percentile service time | painless_static | 495.764 | ms | | error rate | painless_static | 0 | % | | Min Throughput | painless_dynamic | 1.4 | ops/s | | Mean Throughput | painless_dynamic | 1.4 | ops/s | | Median Throughput | painless_dynamic | 1.4 | ops/s | | Max Throughput | painless_dynamic | 1.4 | ops/s | | 50th percentile latency | painless_dynamic | 261.219 | ms | | 90th percentile latency | painless_dynamic | 271.269 | ms | | 99th percentile latency | painless_dynamic | 283.209 | ms | | 100th percentile latency | painless_dynamic | 284.818 | ms | | 50th percentile service time | painless_dynamic | 259.705 | ms | | 90th percentile service time | painless_dynamic | 269.99 | ms | | 99th percentile service time | painless_dynamic | 281.275 | ms | | 100th percentile service time | painless_dynamic | 283.636 | ms | | error rate | painless_dynamic | 0 | % | | Min Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_function_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_function_score | 284.092 | ms | | 90th percentile latency | decay_geo_gauss_function_score | 306.231 | ms | | 99th percentile latency | decay_geo_gauss_function_score | 327.07 | ms | | 100th percentile latency | decay_geo_gauss_function_score | 334.476 | ms | | 50th percentile service time | decay_geo_gauss_function_score | 282.229 | ms | | 90th percentile service time | decay_geo_gauss_function_score | 304.133 | ms | | 99th percentile service time | decay_geo_gauss_function_score | 325.286 | ms | | 100th percentile service time | decay_geo_gauss_function_score | 332.144 | ms | | error rate | decay_geo_gauss_function_score | 0 | % | | Min Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_script_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_script_score | 278.615 | ms | | 90th percentile latency | decay_geo_gauss_script_score | 284.208 | ms | | 99th percentile latency | decay_geo_gauss_script_score | 294.642 | ms | | 100th percentile latency | decay_geo_gauss_script_score | 298.638 | ms | | 50th percentile service time | decay_geo_gauss_script_score | 276.775 | ms | | 90th percentile service time | decay_geo_gauss_script_score | 282.695 | ms | | 99th percentile service time | decay_geo_gauss_script_score | 292.659 | ms | | 100th percentile service time | decay_geo_gauss_script_score | 296.431 | ms | | error rate | decay_geo_gauss_script_score | 0 | % | | Min Throughput | field_value_function_score | 1.5 | ops/s | | Mean Throughput | field_value_function_score | 1.51 | ops/s | | Median Throughput | field_value_function_score | 1.5 | ops/s | | Max Throughput | field_value_function_score | 1.51 | ops/s | | 50th percentile latency | field_value_function_score | 104.891 | ms | | 90th percentile latency | field_value_function_score | 118.247 | ms | | 99th percentile latency | field_value_function_score | 156.556 | ms | | 100th percentile latency | field_value_function_score | 173.81 | ms | | 50th percentile service time | field_value_function_score | 103.3 | ms | | 90th percentile service time | field_value_function_score | 116.564 | ms | | 99th

percentile service time | field_value_function_score | 154.567 | ms | | 100th percentile service time | field_value_function_score | 171.585 | ms | | error rate | field_value_function_score | 0 | % | | Min Throughput | field_value_script_score | 1.5 | ops/s | | Mean Throughput | field_value_script_score | 1.5 | ops/s | | Median Throughput | field_value_script_score | 1.5 | ops/s | | Max Throughput | field_value_script_score | 1.51 | ops/s | | 50th percentile latency | field_value_script_score | 131.701 | ms | | 90th percentile latency | field_value_script_score | 147.025 | ms | | 99th percentile latency | field_value_script_score | 209.773 | ms | | 100th percentile latency | field_value_script_score | 218.408 | ms | | 50th percentile service time | field_value_script_score | 129.87 | ms | | 90th percentile service time | field_value_script_score | 146.026 | ms | | 99th percentile service time | field_value_script_score | 208.177 | ms | | 100th percentile service time | field_value_script_score | 216.713 | ms | | error rate | field_value_script_score | 0 | % | | Min Throughput | large_terms | 1.1 | ops/s | | Mean Throughput | large_terms | 1.1 | ops/s | | Median Throughput | large_terms | 1.1 | ops/s | | Max Throughput | large_terms | 1.1 | ops/s | | 50th percentile latency | large_terms | 549.582 | ms | | 90th percentile latency | large_terms | 564.981 | ms | | 99th percentile latency | large_terms | 625.616 | ms | | 100th percentile latency | large_terms | 629.699 | ms | | 50th percentile service time | large_terms | 542.805 | ms | | 90th percentile service time | large_terms | 558.597 | ms | | 99th percentile service time | large_terms | 618.715 | ms | | 100th percentile service time | large_terms | 622.72 | ms | | error rate | large_terms | 0 | % | | Min Throughput | large_filtered_terms | 1.1 | ops/s | | Mean Throughput | large_filtered_terms | 1.1 | ops/s | | Median Throughput | large_filtered_terms | 1.1 | ops/s | | Max Throughput | large_filtered_terms | 1.1 | ops/s | | 50th percentile latency | large_filtered_terms | 537.876 | ms | | 90th percentile latency | large_filtered_terms | 562.391 | ms | | 99th percentile latency | large_filtered_terms | 604.459 | ms | | 100th percentile latency | large_filtered_terms | 605.975 | ms | | 50th percentile service time | large_filtered_terms | 531.211 | ms | | 90th percentile service time | large_filtered_terms | 556.294 | ms | | 99th percentile service time | large_filtered_terms | 597.845 | ms | | 100th percentile service time | large_filtered_terms | 599.668 | ms | | error rate | large_filtered_terms | 0 | % | | Min Throughput | large_prohibited_terms | 1.1 | ops/s | | Mean Throughput | large_prohibited_terms | 1.1 | ops/s | | Median Throughput | large_prohibited_terms | 1.1 | ops/s | | Max Throughput | large_prohibited_terms | 1.1 | ops/s | | 50th percentile latency | large_prohibited_terms | 534.651 | ms | | 90th percentile latency | large_prohibited_terms | 564.35 | ms | | 99th percentile latency | large_prohibited_terms | 581.457 | ms | | 100th percentile latency | large_prohibited_terms | 586.575 | ms | | 50th percentile service time | large_prohibited_terms | 528.209 | ms | | 90th percentile service time | large_prohibited_terms | 557.814 | ms | | 99th percentile service time | large_prohibited_terms | 574.667 | ms | | 100th percentile service time | large_prohibited_terms | 580.075 | ms | | error rate | large_prohibited_terms | 0 | % | | Min Throughput | desc_sort_population | 1.5 | ops/s | | Mean Throughput | desc_sort_population | 1.5 | ops/s | | Median Throughput | desc_sort_population | 1.5 | ops/s | | Max Throughput | desc_sort_population | 1.51 | ops/s | | 50th percentile latency | desc_sort_population | 50.7253 | ms | | 90th percentile latency | desc_sort_population | 57.6335 | ms | | 99th percentile latency | desc_sort_population | 75.6575 | ms | | 100th percentile latency | desc_sort_population | 77.1718 | ms | | 50th percentile service time | desc_sort_population | 49.0796 | ms | | 90th percentile service time | desc_sort_population | 55.4331 | ms | | 99th percentile service time | desc_sort_population | 73.7205 | ms | | 100th percentile service time | desc_sort_population | 76.3047 | ms | | error rate | desc_sort_population | 0 | % | | Min Throughput | asc_sort_population | 1.5 | ops/s | | Mean Throughput | asc_sort_population | 1.51 | ops/s | | Median Throughput | asc_sort_population | 1.51 | ops/s | | Max Throughput | asc_sort_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_population | 51.1803 | ms | | 90th percentile latency | asc_sort_population | 64.4503 | ms | | 99th percentile latency | asc_sort_population | 94.2661 | ms | | 100th percentile latency | asc_sort_population | 102.594 | ms | | 50th percentile service time | asc_sort_population | 49.4391 | ms | | 90th percentile service time | asc_sort_population | 61.6553 | ms | | 99th percentile service time | asc_sort_population | 92.323 | ms | | 100th percentile service time | asc_sort_population | 100.32 | ms | | error rate | asc_sort_population | 0 | % | | Min Throughput | asc_sort_with_after_population | 1.5 | ops/s | | Mean Throughput | asc_sort_with_after_population | 1.5 | ops/s | | Median Throughput | asc_sort_with_after_population | 1.5 | ops/s | | Max Throughput | asc_sort_with_after_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_with_after_population | 72.0933 | ms | | 90th percentile latency | asc_sort_with_after_population | 76.2541 | ms | | 99th percentile latency | asc_sort_with_after_population | 78.9059 | ms | | 100th percentile latency | asc_sort_with_after_population | 79.0101 | ms | | 50th percentile service time | asc_sort_with_after_population | 70.6371 | ms | | 90th percentile service time | asc_sort_with_after_population | 74.2892 | ms | | 99th percentile service time | asc_sort_with_after_population | 77.1596 | ms | | 100th percentile service time | asc_sort_with_after_population | 77.2173 | ms | | error rate | asc_sort_with_after_population | 0 | % | | Min Throughput | desc_sort_geonameid | 6.01 | ops/s | | Mean Throughput | desc_sort_geonameid | 6.01 | ops/s | | Median Throughput |

desc_sort_geonameid | 6.01 | ops/s | | Max Throughput | desc_sort_geonameid | 6.02 | ops/s | | 50th percentile latency | desc_sort_geonameid | 6.53949 | ms | | 90th percentile latency | desc_sort_geonameid | 7.71243 | ms | | 99th percentile latency | desc_sort_geonameid | 13.2451 | ms | | 100th percentile latency | desc_sort_geonameid | 14.567 | ms | | 50th percentile service time | desc_sort_geonameid | 5.3374 | ms | | 90th percentile service time | desc_sort_geonameid | 6.38921 | ms | | 99th percentile service time | desc_sort_geonameid | 11.6389 | ms | | 100th percentile service time | desc_sort_geonameid | 13.5502 | ms | | error rate | desc_sort_geonameid | 0 | % | | Min Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | 50th percentile latency | desc_sort_with_after_geonameid | 61.855 | ms | | 90th percentile latency | desc_sort_with_after_geonameid | 69.3716 | ms | | 99th percentile latency | desc_sort_with_after_geonameid | 118.555 | ms | | 100th percentile latency | desc_sort_with_after_geonameid | 119.929 | ms | | 50th percentile service time | desc_sort_with_after_geonameid | 60.9539 | ms | | 90th percentile service time | desc_sort_with_after_geonameid | 67.9422 | ms | | 99th percentile service time | desc_sort_with_after_geonameid | 117.344 | ms | | 100th percentile service time | desc_sort_with_after_geonameid | 118.567 | ms | | error rate | desc_sort_with_after_geonameid | 0 | % | | Min Throughput | asc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | asc_sort_geonameid | 6.02 | ops/s | | Median Throughput | asc_sort_geonameid | 6.02 | ops/s | | Max Throughput | asc_sort_geonameid | 6.03 | ops/s | | 50th percentile latency | asc_sort_geonameid | 6.87588 | ms | | 90th percentile latency | asc_sort_geonameid | 8.01471 | ms | | 99th percentile latency | asc_sort_geonameid | 9.60322 | ms | | 100th percentile latency | asc_sort_geonameid | 13.9739 | ms | | 50th percentile service time | asc_sort_geonameid | 5.55821 | ms | | 90th percentile service time | asc_sort_geonameid | 6.42167 | ms | | 99th percentile service time | asc_sort_geonameid | 8.50887 | ms | | 100th percentile service time | asc_sort_geonameid | 12.8557 | ms | | error rate | asc_sort_geonameid | 0 | % | | Min Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | asc_sort_with_after_geonameid | 6.02 | ops/s | | Median Throughput | asc_sort_with_after_geonameid | 6.02 | ops/s | | Max Throughput | asc_sort_with_after_geonameid | 6.02 | ops/s | | 50th percentile latency | asc_sort_with_after_geonameid | 54.6682 | ms | | 90th percentile latency | asc_sort_with_after_geonameid | 61.0326 | ms | | 99th percentile latency | asc_sort_with_after_geonameid | 85.6821 | ms | | 100th percentile latency | asc_sort_with_after_geonameid | 110.872 | ms | | 50th percentile service time | asc_sort_with_after_geonameid | 53.7899 | ms | | 90th percentile service time | asc_sort_with_after_geonameid | 60.0184 | ms | | 99th percentile service time | asc_sort_with_after_geonameid | 84.6155 | ms | | 100th percentile service time | asc_sort_with_after_geonameid | 110.034 | ms | | error rate | asc_sort_with_after_geonameid | 0 | % |

8核32GB512GB通用型的3个数据节点实例基准性能指标

8核32GB512GB通用型的3个数据节点实例基准性能指标

使用ESRally工具对数据节点规格为8核32GB的BES集群实例（7.10.2版本）进行基准测试。

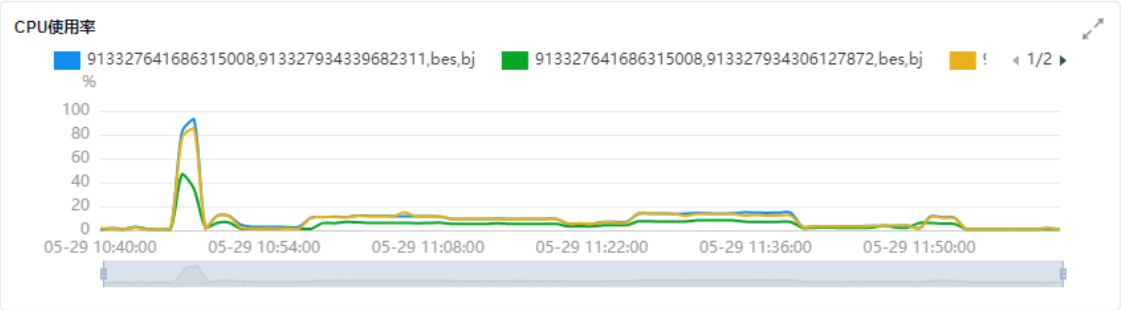
压测配置

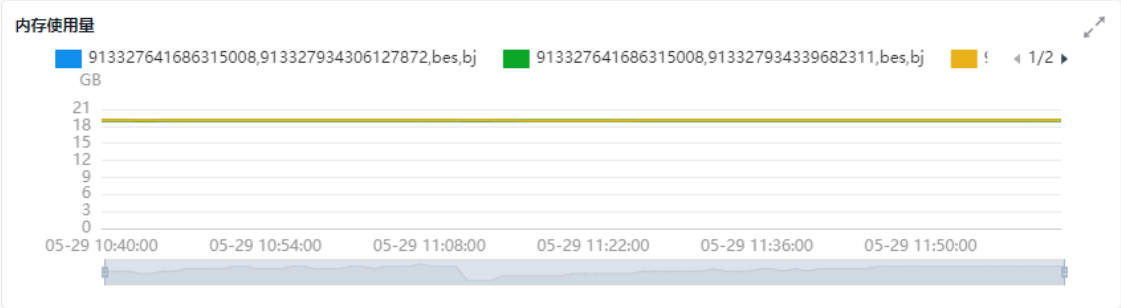
项目	说明
实例配置	版本：BES集群实例（7.10.2版本） 数据节点规格：8核32GB 数据节点存储性能级别：增强型SSD_PL1 数据单节点存储空间：512GB 数据节点数量：3个
ESRally配置	使用ESRally的默认配置tracks。
数据集	使用ESRally的预置数据集Geonames模拟测试场景，数据文档总数11396503。获取数据集，请查看ES Rally Hub获取。
分片数	主分片：6 副本数：0
bulk_size	每次批量操作提交5000个文档。
bulk_indexing_clients	同时执行批量索引操作的客户端数量为8个。
index_append warmup time	从默认的120s修改为30s，当集群配置高时，index_append在预热启动期间就完成了。

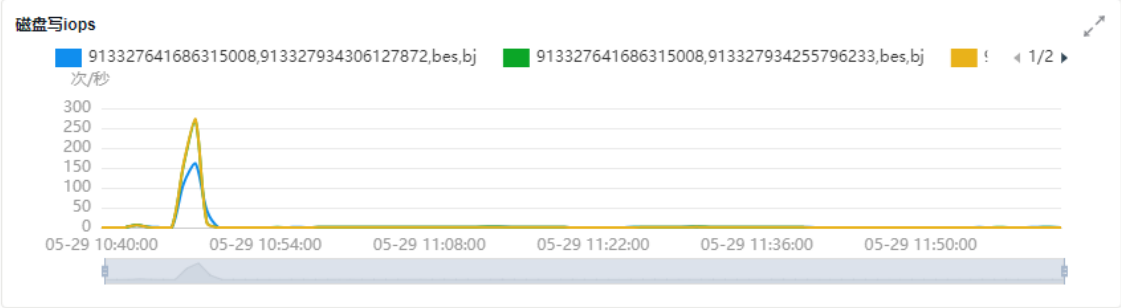
压测结果

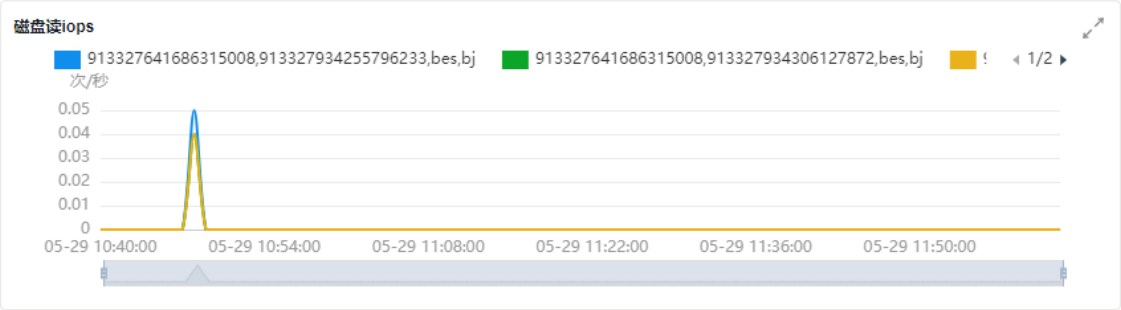
压测结果仅供参考，无法代表实际生产中写入查询情况，建议您结合业务生产数据进行压测。

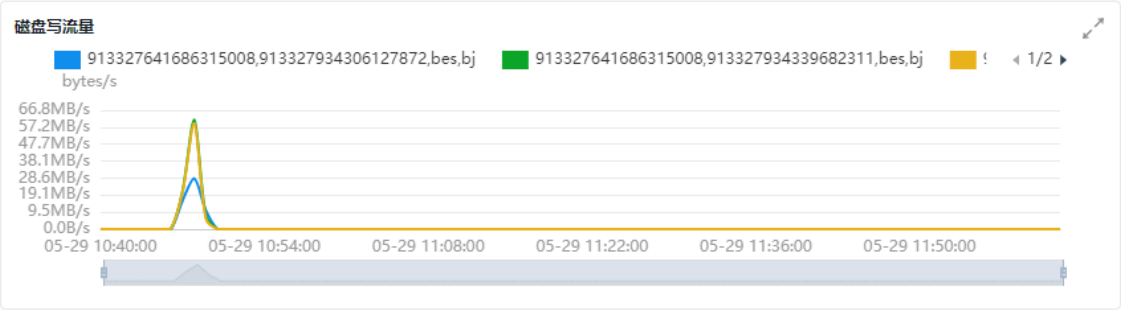
主要压测指标结果

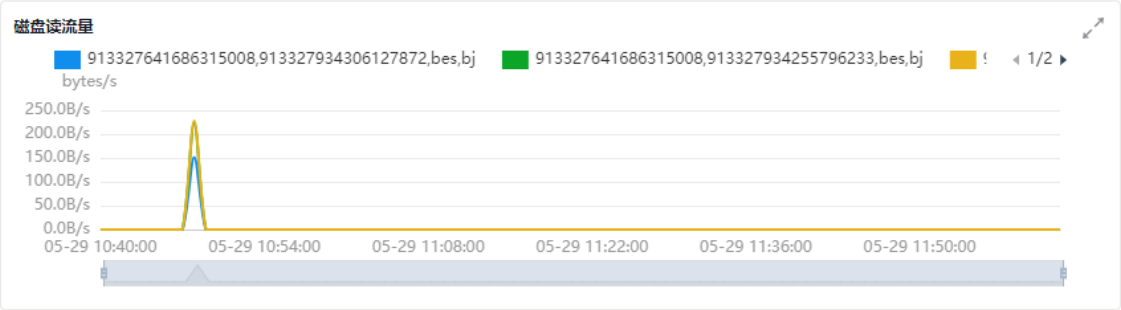


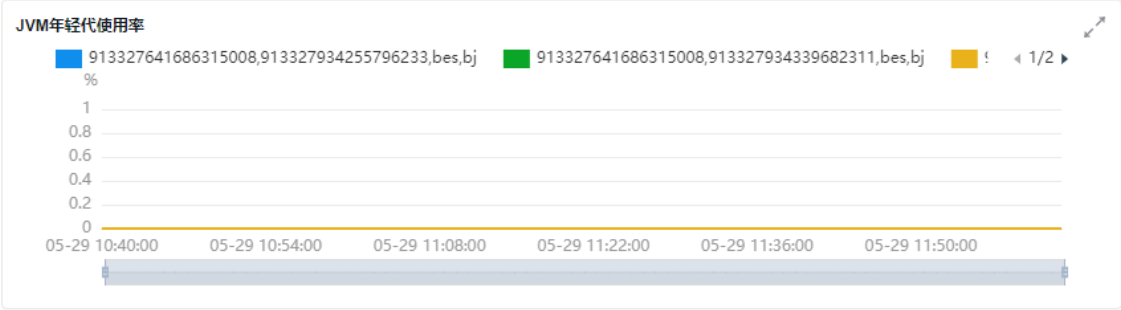


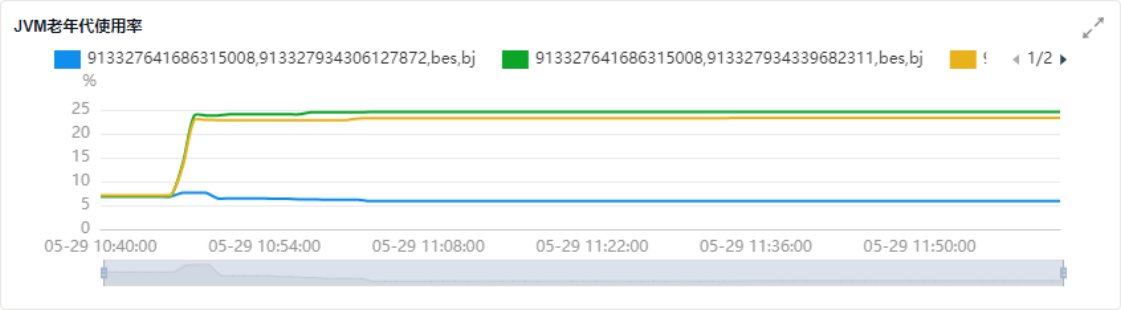


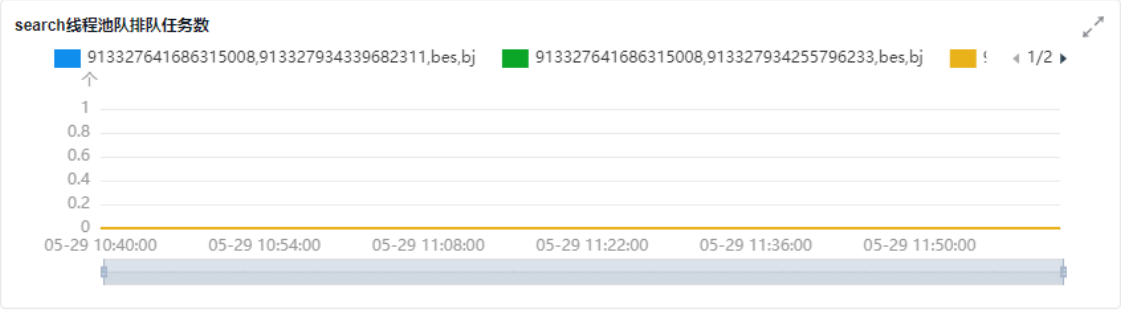

磁盘读iops
913327641686315008,913327934255796233,bes,bj
913327641686315008,913327934306127872,bes,bj
1/2
次/秒
0.05
0.04
0.03
0.02
0.01
0
05-29 10:40:00
05-29 10:54:00
05-29 11:08:00
05-29 11:22:00
05-29 11:36:00
05-29 11:50:00

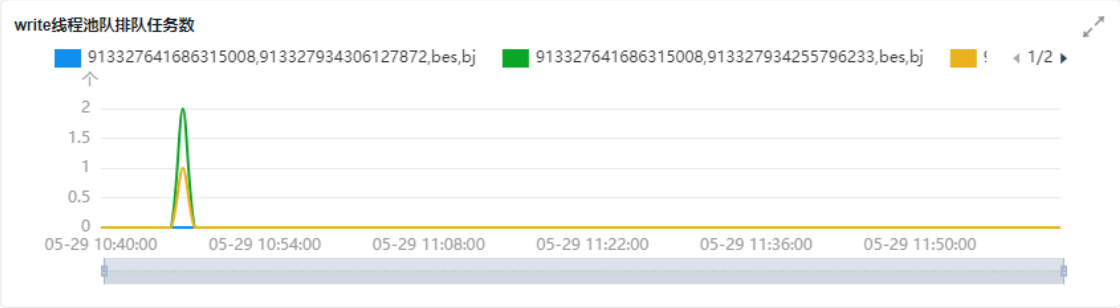

磁盘写流量
913327641686315008,913327934306127872,bes,bj
913327641686315008,913327934339682311,bes,bj
1/2
bytes/s
66.8MB/s
57.2MB/s
47.7MB/s
38.1MB/s
28.6MB/s
19.1MB/s
9.5MB/s
0.0B/s
05-29 10:40:00
05-29 10:54:00
05-29 11:08:00
05-29 11:22:00
05-29 11:36:00
05-29 11:50:00


磁盘读流量
913327641686315008,913327934306127872,bes,bj
913327641686315008,913327934255796233,bes,bj
1/2
bytes/s
250.0B/s
200.0B/s
150.0B/s
100.0B/s
50.0B/s
0.0B/s
05-29 10:40:00
05-29 10:54:00
05-29 11:08:00
05-29 11:22:00
05-29 11:36:00
05-29 11:50:00


JVM年轻代使用率
913327641686315008,913327934255796233,bes,bj
913327641686315008,913327934339682311,bes,bj
1/2
%
1
0.8
0.6
0.4
0.2
0
05-29 10:40:00
05-29 10:54:00
05-29 11:08:00
05-29 11:22:00
05-29 11:36:00
05-29 11:50:00


JVM老年代使用率
913327641686315008,913327934306127872,bes,bj
913327641686315008,913327934339682311,bes,bj
1/2
%
25
20
15
10
5
0
05-29 10:40:00
05-29 10:54:00
05-29 11:08:00
05-29 11:22:00
05-29 11:36:00
05-29 11:50:00


search线程池队排队任务数
913327641686315008,913327934339682311,bes,bj
913327641686315008,913327934255796233,bes,bj
1/2
个
1
0.8
0.6
0.4
0.2
0
05-29 10:40:00
05-29 10:54:00
05-29 11:08:00
05-29 11:22:00
05-29 11:36:00
05-29 11:50:00



基准压测报告 | Metric | Task | Value | Unit | |---|---|---|---| | Cumulative indexing time of primary shards | | 13.4029 | min | | Min cumulative indexing time across primary shards | | 2.18048 | min | | Median cumulative indexing time across primary shards | | 2.71837 | min | | Max cumulative indexing time across primary shards | | 2.94395 | min | | Cumulative indexing throttle time of primary shards | | 0 | min | | Min cumulative indexing throttle time across primary shards | | 0 | min | | Median cumulative indexing throttle time across primary shards | | 0 | min | | Max cumulative indexing throttle time across primary shards | | 0 | min | | Cumulative merge time of primary shards | | 3.59067 | min | | Cumulative merge count of primary shards | | 24 | | | Min cumulative merge time across primary shards | | 0.608417 | min | | Median cumulative merge time across primary shards | | 0.7234 | min | | Max cumulative merge time across primary shards | | 0.834817 | min | | Cumulative merge throttle time of primary shards | | 0.9066 | min | | Min cumulative merge throttle time across primary shards | | 0.1244 | min | | Median cumulative merge throttle time across primary shards | | 0.1927 | min | | Max cumulative merge throttle time across primary shards | | 0.2148 | min | | Cumulative refresh time of primary shards | | 1.01488 | min | | Cumulative refresh count of primary shards | | 90 | | | Min cumulative refresh time across primary shards | | 0.1771 | min | | Median cumulative refresh time across primary shards | | 0.206167 | min | | Max cumulative refresh time across primary shards | | 0.22955 | min | | Cumulative flush time of primary shards | | 0.166533 | min | | Cumulative flush count of primary shards | | 10 | | | Min cumulative flush time across primary shards | | 0.0246333 | min | | Median cumulative flush time across primary shards | | 0.0342167 | min | | Max cumulative flush time across primary shards | | 0.0377333 | min | | Total Young Gen GC time | | 3.982 | s | | Total Young Gen GC count | | 617 | | | Total Old Gen GC time | | 0 | s | | Total Old Gen GC count | | 0 | | | Store size | | 3.04147 | GB | | Translog size | | 2.56114e-07 | GB | | Heap used for segments | | 0.729324 | MB | | Heap used for doc values | | 0.0372009 | MB | | Heap used for terms | | 0.565247 | MB | | Heap used for norms | | 0.0769043 | MB | | Heap used for points | | 0 | MB | | Heap used for stored fields | | 0.0499725 | MB | | Segment count | | 98 | | | Total Ingest Pipeline count | | 0 | | | Total Ingest Pipeline time | | 0 | s | | Total Ingest Pipeline failed | | 0 | | | Min Throughput | index-append | 165743 | docs/s | | Mean Throughput | index-append | 176021 | docs/s | | Median Throughput | index-append | 175847 | docs/s | | Max Throughput | index-append | 191886 | docs/s | | 50th percentile latency | index-append | 179.497 | ms | | 90th percentile latency | index-append | 511.661 | ms | | 99th percentile latency | index-append | 1420.49 | ms | | 99.9th percentile latency | index-append | 2188.58 | ms | | 100th percentile latency | index-append | 2290.8 | ms | | 50th percentile service time | index-append | 179.497 | ms | | 90th percentile service time | index-append | 511.661 | ms | | 99th percentile service time | index-append | 1420.49 | ms | | 99.9th percentile service time | index-append | 2188.58 | ms | | 100th percentile service time | index-append | 2290.8 | ms | | error rate | index-append | 0 | % | | Min Throughput | index-stats | 90.04 | ops/s | | Mean Throughput | index-stats | 90.06 | ops/s | | Median Throughput | index-stats | 90.05 | ops/s | | Max Throughput | index-stats | 90.1 | ops/s | | 50th percentile latency | index-stats | 2.83175 | ms | | 90th percentile latency | index-stats | 3.71149 | ms | | 99th percentile latency | index-stats | 4.1057 | ms | | 99.9th percentile latency | index-stats | 4.27278 | ms | | 100th percentile latency | index-stats | 4.28279 | ms | | 50th percentile service time | index-stats | 1.50329 | ms | | 90th percentile service time | index-stats | 1.7402 | ms | | 99th percentile service time | index-stats | 1.96819 | ms | | 99.9th percentile service time | index-stats | 2.29227 | ms | | 100th percentile service time | index-stats | 2.62388 | ms | | error rate | index-stats | 0 | % | | Min Throughput | node-stats | 89.77 | ops/s | | Mean Throughput | node-stats | 90.05 | ops/s | | Median Throughput | node-stats | 90.05 | ops/s | | Max Throughput | node-stats | 90.21 | ops/s | | 50th percentile latency | node-stats | 3.50882 | ms | | 90th percentile latency | node-stats | 4.02628 | ms | | 99th percentile latency | node-stats | 4.64682 | ms | | 99.9th percentile latency | node-stats | 7.86337 | ms | | 100th percentile latency | node-stats | 15.8098 | ms | | 50th percentile service time | node-stats | 2.55518 | ms | | 90th percentile service time | node-stats | 2.94944 | ms | | 99th percentile service time | node-stats | 3.60089 | ms | | 99.9th percentile service time | node-stats | 3.90836 | ms | | 100th percentile service time | node-stats | 4.72011 | ms | | error rate | node-stats | 0 | % | | Min Throughput | default | 50.03 | ops/s | | Mean Throughput | default | 50.04 | ops/s | | Median

Throughput | default | 50.04 | ops/s | | Max Throughput | default | 50.07 | ops/s | | 50th percentile latency | default | 3.12762 | ms | | 90th percentile latency | default | 4.41481 | ms | | 99th percentile latency | default | 4.84452 | ms | | 99.9th percentile latency | default | 5.33394 | ms | | 100th percentile latency | default | 13.1374 | ms | | 50th percentile service time | default | 2.10249 | ms | | 90th percentile service time | default | 2.35782 | ms | | 99th percentile service time | default | 2.98018 | ms | | 99.9th percentile service time | default | 4.65912 | ms | | 100th percentile service time | default | 12.1674 | ms | | error rate | default | 0 | % | | Min Throughput | term | 100.04 | ops/s | | Mean Throughput | term | 100.06 | ops/s | | Median Throughput | term | 100.05 | ops/s | | Max Throughput | term | 100.09 | ops/s | | 50th percentile latency | term | 3.08206 | ms | | 90th percentile latency | term | 3.60959 | ms | | 99th percentile latency | term | 3.95824 | ms | | 99.9th percentile latency | term | 5.91828 | ms | | 100th percentile latency | term | 11.4106 | ms | | 50th percentile service time | term | 2.12286 | ms | | 90th percentile service time | term | 2.34504 | ms | | 99th percentile service time | term | 2.83201 | ms | | 99.9th percentile service time | term | 5.2732 | ms | | 100th percentile service time | term | 10.8154 | ms | | error rate | term | 0 | % | | Min Throughput | phrase | 110.02 | ops/s | | Mean Throughput | phrase | 110.04 | ops/s | | Median Throughput | phrase | 110.03 | ops/s | | Max Throughput | phrase | 110.06 | ops/s | | 50th percentile latency | phrase | 3.08929 | ms | | 90th percentile latency | phrase | 3.60749 | ms | | 99th percentile latency | phrase | 4.76916 | ms | | 99.9th percentile latency | phrase | 25.1566 | ms | | 100th percentile latency | phrase | 25.7647 | ms | | 50th percentile service time | phrase | 2.17803 | ms | | 90th percentile service time | phrase | 2.45054 | ms | | 99th percentile service time | phrase | 3.21412 | ms | | 99.9th percentile service time | phrase | 17.3833 | ms | | 100th percentile service time | phrase | 24.4362 | ms | | error rate | phrase | 0 | % | | Min Throughput | country_agg_uncached | 3.01 | ops/s | | Mean Throughput | country_agg_uncached | 3.01 | ops/s | | Median Throughput | country_agg_uncached | 3.01 | ops/s | | Max Throughput | country_agg_uncached | 3.01 | ops/s | | 50th percentile latency | country_agg_uncached | 151.068 | ms | | 90th percentile latency | country_agg_uncached | 163.163 | ms | | 99th percentile latency | country_agg_uncached | 166.922 | ms | | 100th percentile latency | country_agg_uncached | 184.322 | ms | | 50th percentile service time | country_agg_uncached | 149.553 | ms | | 90th percentile service time | country_agg_uncached | 161.65 | ms | | 99th percentile service time | country_agg_uncached | 165.51 | ms | | 100th percentile service time | country_agg_uncached | 183.137 | ms | | error rate | country_agg_uncached | 0 | % | | Min Throughput | country_agg_cached | 98.6 | ops/s | | Mean Throughput | country_agg_cached | 99 | ops/s | | Median Throughput | country_agg_cached | 99.05 | ops/s | | Max Throughput | country_agg_cached | 99.29 | ops/s | | 50th percentile latency | country_agg_cached | 2.74603 | ms | | 90th percentile latency | country_agg_cached | 3.22771 | ms | | 99th percentile latency | country_agg_cached | 3.72037 | ms | | 99.9th percentile latency | country_agg_cached | 3.98326 | ms | | 100th percentile latency | country_agg_cached | 4.03721 | ms | | 50th percentile service time | country_agg_cached | 1.82613 | ms | | 90th percentile service time | country_agg_cached | 2.08052 | ms | | 99th percentile service time | country_agg_cached | 2.4266 | ms | | 99.9th percentile service time | country_agg_cached | 2.62101 | ms | | 100th percentile service time | country_agg_cached | 2.85494 | ms | | error rate | country_agg_cached | 0 | % | | Min Throughput | scroll | 20.05 | pages/s | | Mean Throughput | scroll | 20.07 | pages/s | | Median Throughput | scroll | 20.06 | pages/s | | Max Throughput | scroll | 20.08 | pages/s | | 50th percentile latency | scroll | 227.208 | ms | | 90th percentile latency | scroll | 233.554 | ms | | 99th percentile latency | scroll | 245.062 | ms | | 100th percentile latency | scroll | 245.935 | ms | | 50th percentile service time | scroll | 224.803 | ms | | 90th percentile service time | scroll | 230.807 | ms | | 99th percentile service time | scroll | 243.734 | ms | | 100th percentile service time | scroll | 243.813 | ms | | error rate | scroll | 0 | % | | Min Throughput | expression | 1.5 | ops/s | | Mean Throughput | expression | 1.5 | ops/s | | Median Throughput | expression | 1.5 | ops/s | | Max Throughput | expression | 1.5 | ops/s | | 50th percentile latency | expression | 268.439 | ms | | 90th percentile latency | expression | 278.782 | ms | | 99th percentile latency | expression | 288.187 | ms | | 100th percentile latency | expression | 291.137 | ms | | 50th percentile service time | expression | 266.985 | ms | | 90th percentile service time | expression | 277.209 | ms | | 99th percentile service time | expression | 286.597 | ms | | 100th percentile service time | expression | 289.803 | ms | | error rate | expression | 0 | % | | Min Throughput | painless_static | 1.4 | ops/s | | Mean Throughput | painless_static | 1.4 | ops/s | | Median Throughput | painless_static | 1.4 | ops/s | | Max Throughput | painless_static | 1.4 | ops/s | | 50th percentile latency | painless_static | 314.238 | ms | | 90th percentile latency | painless_static | 332.169 | ms | | 99th percentile latency | painless_static | 344.439 | ms | | 100th percentile latency | painless_static | 347.807 | ms | | 50th percentile service time | painless_static | 312.524 | ms | | 90th percentile service time | painless_static | 330.544 | ms | | 99th percentile service time | painless_static | 343.428 | ms | | 100th percentile service time | painless_static | 346.46 | ms | | error rate | painless_static | 0 | % | | Min Throughput | painless_dynamic | 1.4 | ops/s | | Mean Throughput | painless_dynamic | 1.4 | ops/s | | Median Throughput | painless_dynamic | 1.4 | ops/s | | Max Throughput | painless_dynamic | 1.4 | ops/s | |

50th percentile latency | painless_dynamic | 312.25 | ms | | 90th percentile latency | painless_dynamic | 323.291 | ms | | 99th percentile latency | painless_dynamic | 333.411 | ms | | 100th percentile latency | painless_dynamic | 370.609 | ms | | 50th percentile service time | painless_dynamic | 310.713 | ms | | 90th percentile service time | painless_dynamic | 321.704 | ms | | 99th percentile service time | painless_dynamic | 331.562 | ms | | 100th percentile service time | painless_dynamic | 369.53 | ms | | error rate | painless_dynamic | 0 | % | | Min Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_function_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_function_score | 347.399 | ms | | 90th percentile latency | decay_geo_gauss_function_score | 353.831 | ms | | 99th percentile latency | decay_geo_gauss_function_score | 357.212 | ms | | 100th percentile latency | decay_geo_gauss_function_score | 358.705 | ms | | 50th percentile service time | decay_geo_gauss_function_score | 345.587 | ms | | 90th percentile service time | decay_geo_gauss_function_score | 352.563 | ms | | 99th percentile service time | decay_geo_gauss_function_score | 355.432 | ms | | 100th percentile service time | decay_geo_gauss_function_score | 356.649 | ms | | error rate | decay_geo_gauss_function_score | 0 | % | | Min Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_script_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_script_score | 348.383 | ms | | 90th percentile latency | decay_geo_gauss_script_score | 352.959 | ms | | 99th percentile latency | decay_geo_gauss_script_score | 358.822 | ms | | 100th percentile latency | decay_geo_gauss_script_score | 359.47 | ms | | 50th percentile service time | decay_geo_gauss_script_score | 346.826 | ms | | 90th percentile service time | decay_geo_gauss_script_score | 351.324 | ms | | 99th percentile service time | decay_geo_gauss_script_score | 356.757 | ms | | 100th percentile service time | decay_geo_gauss_script_score | 357.435 | ms | | error rate | decay_geo_gauss_script_score | 0 | % | | Min Throughput | field_value_function_score | 1.5 | ops/s | | Mean Throughput | field_value_function_score | 1.5 | ops/s | | Median Throughput | field_value_function_score | 1.5 | ops/s | | Max Throughput | field_value_function_score | 1.51 | ops/s | | 50th percentile latency | field_value_function_score | 119.648 | ms | | 90th percentile latency | field_value_function_score | 123.809 | ms | | 99th percentile latency | field_value_function_score | 130.048 | ms | | 100th percentile latency | field_value_function_score | 130.895 | ms | | 50th percentile service time | field_value_function_score | 118.066 | ms | | 90th percentile service time | field_value_function_score | 122.089 | ms | | 99th percentile service time | field_value_function_score | 128.638 | ms | | 100th percentile service time | field_value_function_score | 129.569 | ms | | error rate | field_value_function_score | 0 | % | | Min Throughput | field_value_script_score | 1.5 | ops/s | | Mean Throughput | field_value_script_score | 1.5 | ops/s | | Median Throughput | field_value_script_score | 1.5 | ops/s | | Max Throughput | field_value_script_score | 1.51 | ops/s | | 50th percentile latency | field_value_script_score | 164.957 | ms | | 90th percentile latency | field_value_script_score | 169.497 | ms | | 99th percentile latency | field_value_script_score | 190.763 | ms | | 100th percentile latency | field_value_script_score | 204.975 | ms | | 50th percentile service time | field_value_script_score | 163.474 | ms | | 90th percentile service time | field_value_script_score | 168.03 | ms | | 99th percentile service time | field_value_script_score | 188.603 | ms | | 100th percentile service time | field_value_script_score | 202.527 | ms | | error rate | field_value_script_score | 0 | % | | Min Throughput | large_terms | 1.1 | ops/s | | Mean Throughput | large_terms | 1.1 | ops/s | | Median Throughput | large_terms | 1.1 | ops/s | | Max Throughput | large_terms | 1.1 | ops/s | | 50th percentile latency | large_terms | 536.34 | ms | | 90th percentile latency | large_terms | 550.509 | ms | | 99th percentile latency | large_terms | 563.634 | ms | | 100th percentile latency | large_terms | 567.994 | ms | | 50th percentile service time | large_terms | 529.501 | ms | | 90th percentile service time | large_terms | 543.495 | ms | | 99th percentile service time | large_terms | 556.978 | ms | | 100th percentile service time | large_terms | 561.595 | ms | | error rate | large_terms | 0 | % | | Min Throughput | large_filtered_terms | 1.1 | ops/s | | Mean Throughput | large_filtered_terms | 1.1 | ops/s | | Median Throughput | large_filtered_terms | 1.1 | ops/s | | Max Throughput | large_filtered_terms | 1.1 | ops/s | | 50th percentile latency | large_filtered_terms | 544.299 | ms | | 90th percentile latency | large_filtered_terms | 558.143 | ms | | 99th percentile latency | large_filtered_terms | 569.945 | ms | | 100th percentile latency | large_filtered_terms | 578.127 | ms | | 50th percentile service time | large_filtered_terms | 537.555 | ms | | 90th percentile service time | large_filtered_terms | 550.804 | ms | | 99th percentile service time | large_filtered_terms | 563.285 | ms | | 100th percentile service time | large_filtered_terms | 571.144 | ms | | error rate | large_filtered_terms | 0 | % | | Min Throughput | large_prohibited_terms | 1.1 | ops/s | | Mean Throughput | large_prohibited_terms | 1.1 | ops/s | | Median Throughput | large_prohibited_terms | 1.1 | ops/s | | Max Throughput | large_prohibited_terms | 1.1 | ops/s | | 50th percentile latency | large_prohibited_terms | 515.312 | ms | | 90th percentile latency | large_prohibited_terms | 526.015 | ms | | 99th percentile latency | large_prohibited_terms | 546.235

| ms | | 100th percentile latency | large_prohibited_terms | 559.464 | ms | | 50th percentile service time | large_prohibited_terms | 508.509 | ms | | 90th percentile service time | large_prohibited_terms | 519.497 | ms | | 99th percentile service time | large_prohibited_terms | 539.616 | ms | | 100th percentile service time | large_prohibited_terms | 553.125 | ms | | error rate | large_prohibited_terms | 0 | % | | Min Throughput | desc_sort_population | 1.5 | ops/s | | Mean Throughput | desc_sort_population | 1.51 | ops/s | | Median Throughput | desc_sort_population | 1.51 | ops/s | | Max Throughput | desc_sort_population | 1.51 | ops/s | | 50th percentile latency | desc_sort_population | 57.5618 | ms | | 90th percentile latency | desc_sort_population | 64.5965 | ms | | 99th percentile latency | desc_sort_population | 76.3824 | ms | | 100th percentile latency | desc_sort_population | 78.3 | ms | | 50th percentile service time | desc_sort_population | 55.8985 | ms | | 90th percentile service time | desc_sort_population | 62.5495 | ms | | 99th percentile service time | desc_sort_population | 74.2887 | ms | | 100th percentile service time | desc_sort_population | 76.1585 | ms | | error rate | desc_sort_population | 0 | % | | Min Throughput | asc_sort_population | 1.5 | ops/s | | Mean Throughput | asc_sort_population | 1.51 | ops/s | | Median Throughput | asc_sort_population | 1.51 | ops/s | | Max Throughput | asc_sort_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_population | 59.833 | ms | | 90th percentile latency | asc_sort_population | 77.361 | ms | | 99th percentile latency | asc_sort_population | 96.5761 | ms | | 100th percentile latency | asc_sort_population | 113.159 | ms | | 50th percentile service time | asc_sort_population | 58.0944 | ms | | 90th percentile service time | asc_sort_population | 75.2932 | ms | | 99th percentile service time | asc_sort_population | 94.741 | ms | | 100th percentile service time | asc_sort_population | 111.651 | ms | | error rate | asc_sort_population | 0 | % | | Min Throughput | asc_sort_with_after_population | 1.5 | ops/s | | Mean Throughput | asc_sort_with_after_population | 1.51 | ops/s | | Median Throughput | asc_sort_with_after_population | 1.51 | ops/s | | Max Throughput | asc_sort_with_after_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_with_after_population | 79.2837 | ms | | 90th percentile latency | asc_sort_with_after_population | 82.592 | ms | | 99th percentile latency | asc_sort_with_after_population | 95.8046 | ms | | 100th percentile latency | asc_sort_with_after_population | 96.5419 | ms | | 50th percentile service time | asc_sort_with_after_population | 77.3989 | ms | | 90th percentile service time | asc_sort_with_after_population | 80.6128 | ms | | 99th percentile service time | asc_sort_with_after_population | 93.9825 | ms | | 100th percentile service time | asc_sort_with_after_population | 94.4302 | ms | | error rate | asc_sort_with_after_population | 0 | % | | Min Throughput | desc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | desc_sort_geonameid | 6.02 | ops/s | | Median Throughput | desc_sort_geonameid | 6.02 | ops/s | | Max Throughput | desc_sort_geonameid | 6.03 | ops/s | | 50th percentile latency | desc_sort_geonameid | 8.4936 | ms | | 90th percentile latency | desc_sort_geonameid | 8.93789 | ms | | 99th percentile latency | desc_sort_geonameid | 9.32758 | ms | | 100th percentile latency | desc_sort_geonameid | 11.0293 | ms | | 50th percentile service time | desc_sort_geonameid | 7.23764 | ms | | 90th percentile service time | desc_sort_geonameid | 7.45754 | ms | | 99th percentile service time | desc_sort_geonameid | 7.87669 | ms | | 100th percentile service time | desc_sort_geonameid | 9.99676 | ms | | error rate | desc_sort_geonameid | 0 | % | | Min Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | desc_sort_with_after_geonameid | 6.02 | ops/s | | 50th percentile latency | desc_sort_with_after_geonameid | 72.8101 | ms | | 90th percentile latency | desc_sort_with_after_geonameid | 86.3111 | ms | | 99th percentile latency | desc_sort_with_after_geonameid | 98.6668 | ms | | 100th percentile latency | desc_sort_with_after_geonameid | 114.476 | ms | | 50th percentile service time | desc_sort_with_after_geonameid | 71.7572 | ms | | 90th percentile service time | desc_sort_with_after_geonameid | 84.9305 | ms | | 99th percentile service time | desc_sort_with_after_geonameid | 97.0551 | ms | | 100th percentile service time | desc_sort_with_after_geonameid | 113.249 | ms | | error rate | desc_sort_with_after_geonameid | 0 | % | | Min Throughput | asc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | asc_sort_geonameid | 6.02 | ops/s | | Median Throughput | asc_sort_geonameid | 6.02 | ops/s | | Max Throughput | asc_sort_geonameid | 6.03 | ops/s | | 50th percentile latency | asc_sort_geonameid | 6.0917 | ms | | 90th percentile latency | asc_sort_geonameid | 6.79385 | ms | | 99th percentile latency | asc_sort_geonameid | 7.6651 | ms | | 100th percentile latency | asc_sort_geonameid | 7.77227 | ms | | 50th percentile service time | asc_sort_geonameid | 4.66351 | ms | | 90th percentile service time | asc_sort_geonameid | 5.64047 | ms | | 99th percentile service time | asc_sort_geonameid | 6.17686 | ms | | 100th percentile service time | asc_sort_geonameid | 6.24999 | ms | | error rate | asc_sort_geonameid | 0 | % | | Min Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | asc_sort_with_after_geonameid | 6.02 | ops/s | | 50th percentile latency | asc_sort_with_after_geonameid | 67.1837 | ms | | 90th percentile latency | asc_sort_with_after_geonameid | 79.7437 | ms | |

99th percentile latency | asc_sort_with_after_geonameid | 85.9863 | ms | | 100th percentile latency | asc_sort_with_after_geonameid | 86.6933 | ms | | 50th percentile service time | asc_sort_with_after_geonameid | 65.9387 | ms | | 90th percentile service time | asc_sort_with_after_geonameid | 78.3939 | ms | | 99th percentile service time | asc_sort_with_after_geonameid | 84.669 | ms | | 100th percentile service time | asc_sort_with_after_geonameid | 85.5162 | ms | | error rate | asc_sort_with_after_geonameid | 0 | % |

8核16GB512GB计算型的3个数据节点实例基准性能测试

8核16GB 512GB计算型的3个数据节点实例基准性能测试

使用ESRally工具对数据节点规格为8核16GB的BES集群实例（7.10.2版本）进行基准测试。

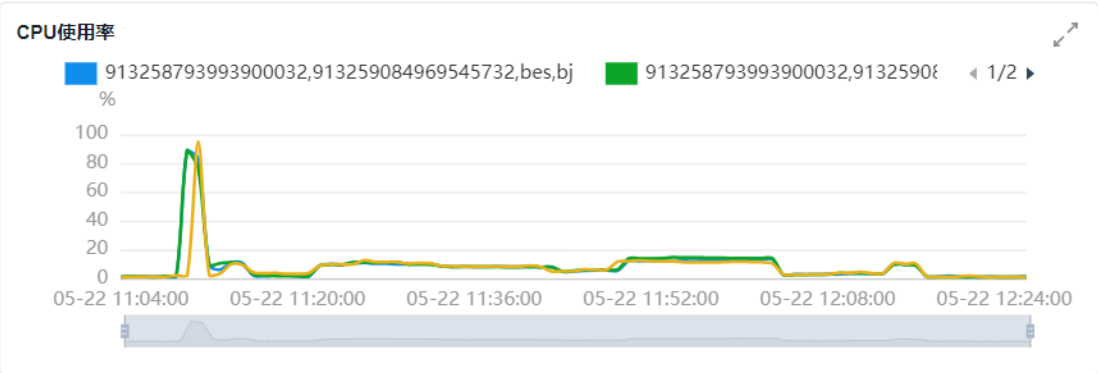
压测配置

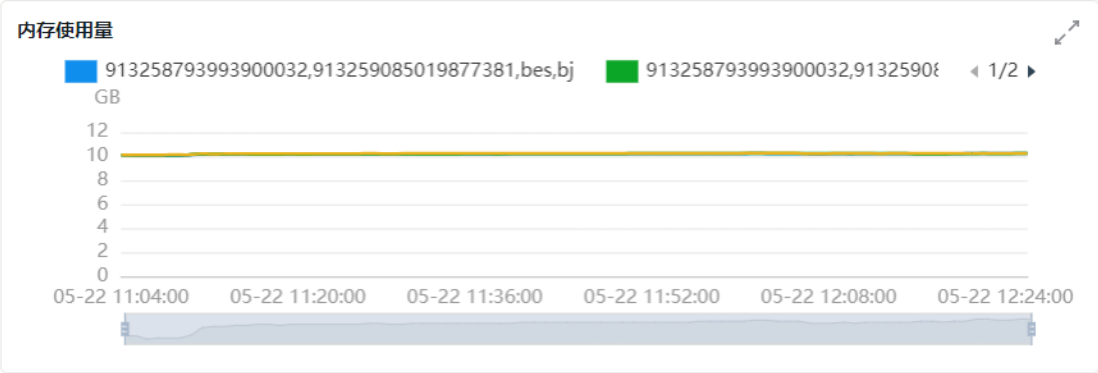
项目	说明
实例配置	版本：BES集群实例（7.10.2版本） 数据节点规格：8核16GB 数据节点存储性能级别：增强型SSD_PL1 数据单节点存储空间：512GB 数据节点数量：3个
ESRally配置	使用ESRally的默认配置tracks。
数据集	使用ESRally的预置数据集Geonames模拟测试场景，数据集大小2.8GB，文档总数11396503。获取数据集，请查看ES Rally Hub获取。
分片数	主分片：6 副本数：0
bulk_size	每次批量操作提交5000个文档。
bulk_indexing_clients	同时执行批量索引操作的客户端数量为8个。
index_append warmup time	从默认的120s修改为30s，当集群配置高时，index_append在预热启动期间就完成了。

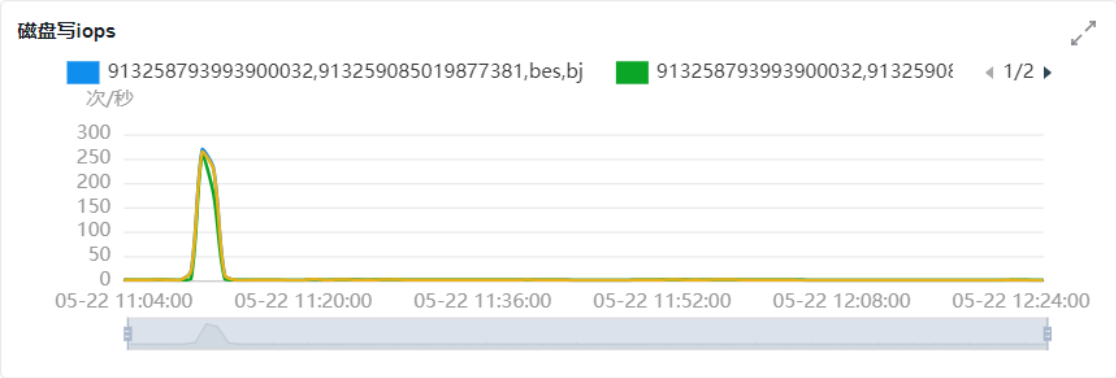
压测结果

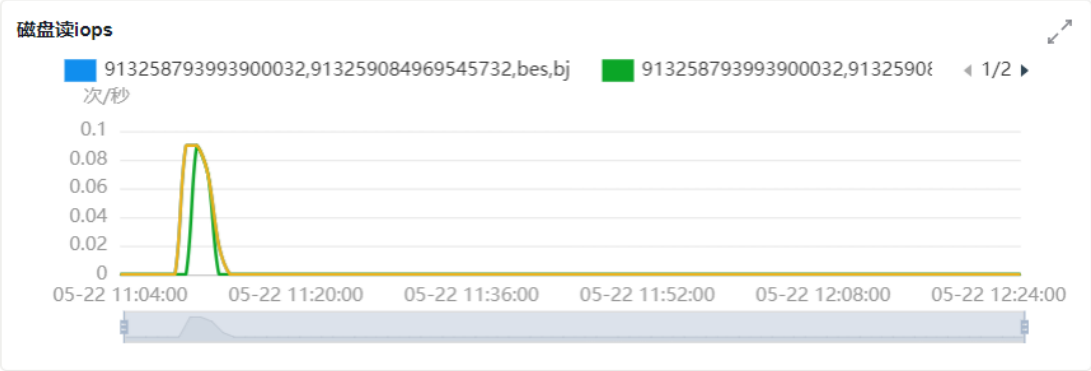
压测结果仅供参考，无法代表实际生产中写入查询情况，建议您结合业务生产数据进行压测。

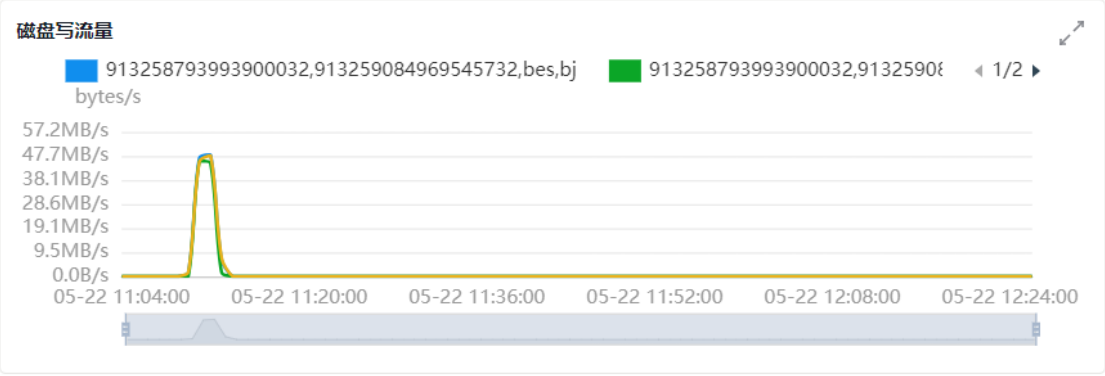
主要压测指标结果

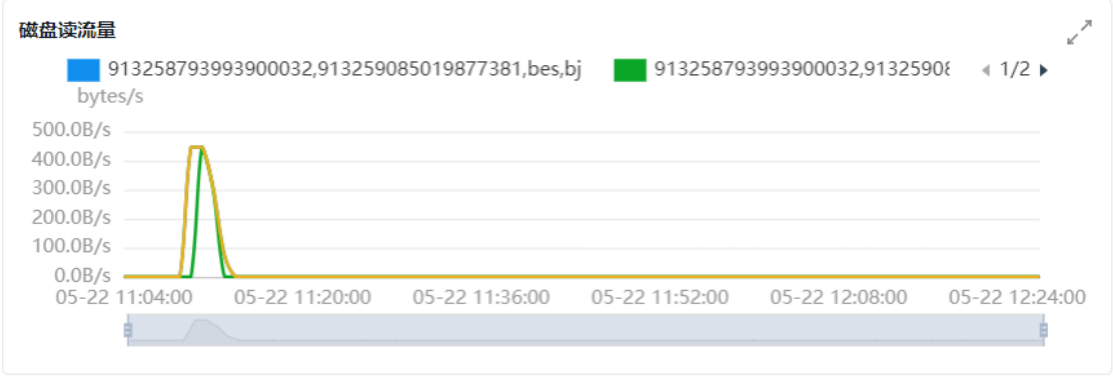


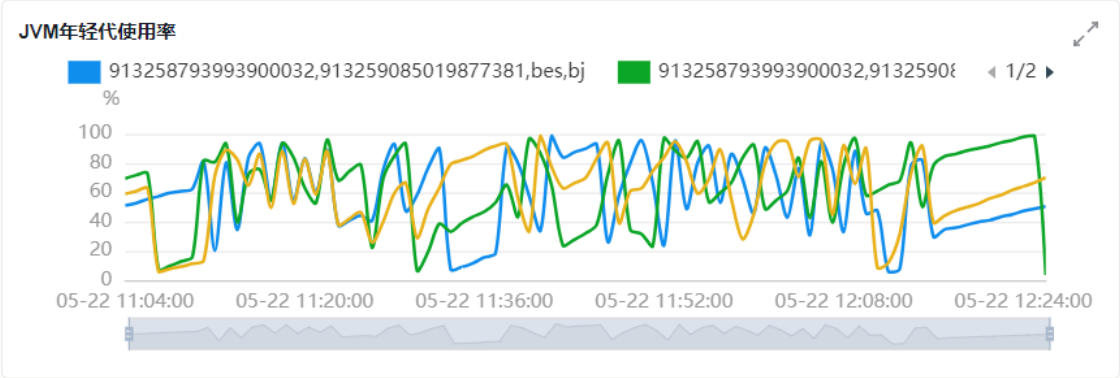

内存使用量
913258793993900032,913259085019877381,bes,bj
913258793993900032,9132590
1/2
GB
12
10
8
6
4
2
0
05-22 11:04:00
05-22 11:20:00
05-22 11:36:00
05-22 11:52:00
05-22 12:08:00
05-22 12:24:00

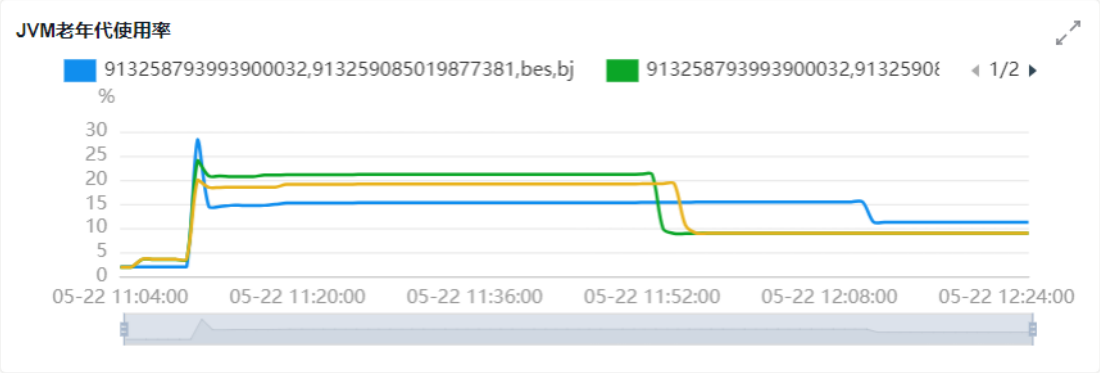

磁盘写iops
913258793993900032,913259085019877381,bes,bj
913258793993900032,9132590
1/2
次/秒
300
250
200
150
100
50
0
05-22 11:04:00
05-22 11:20:00
05-22 11:36:00
05-22 11:52:00
05-22 12:08:00
05-22 12:24:00

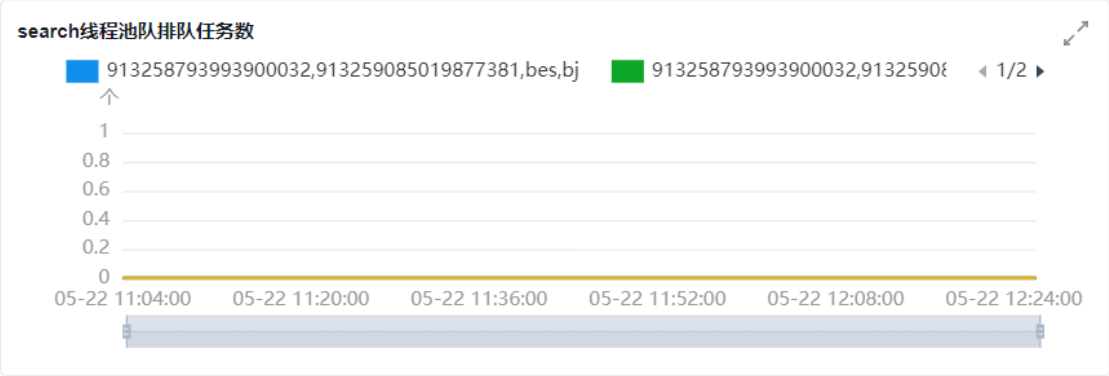

磁盘读iops
913258793993900032,913259084969545732,bes,bj
913258793993900032,9132590
1/2
次/秒
0.1
0.08
0.06
0.04
0.02
0
05-22 11:04:00
05-22 11:20:00
05-22 11:36:00
05-22 11:52:00
05-22 12:08:00
05-22 12:24:00

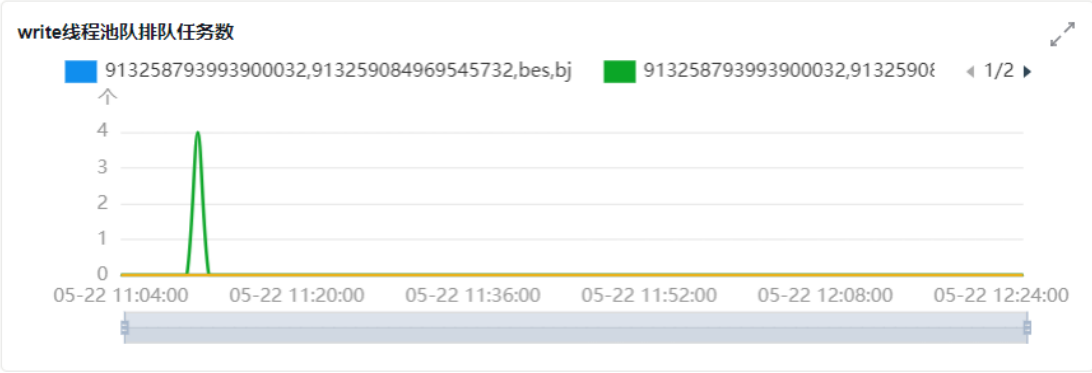

磁盘写流量
913258793993900032,913259084969545732,bes,bj
913258793993900032,9132590
1/2
bytes/s
57.2MB/s
47.7MB/s
38.1MB/s
28.6MB/s
19.1MB/s
9.5MB/s
0.0B/s
05-22 11:04:00
05-22 11:20:00
05-22 11:36:00
05-22 11:52:00
05-22 12:08:00
05-22 12:24:00


磁盘读流量
913258793993900032,913259085019877381,bes,bj
913258793993900032,9132590
1/2
bytes/s
500.0B/s
400.0B/s
300.0B/s
200.0B/s
100.0B/s
0.0B/s
05-22 11:04:00
05-22 11:20:00
05-22 11:36:00
05-22 11:52:00
05-22 12:08:00
05-22 12:24:00









基准压测报告 | Metric | Task | Value | Unit | |---:|---:|---:|---:| | Cumulative indexing time of primary shards | | 13.962 | min | | Min cumulative indexing time across primary shards | | 2.09138 | min | | Median cumulative indexing time across primary shards | | 2.96602 | min | | Max cumulative indexing time across primary shards | | 3.00725 | min | | Cumulative indexing throttle time of primary shards | | 0 | min | | Min cumulative indexing throttle time across primary shards | | 0 | min | | Median cumulative indexing throttle time across primary shards | | 0 | min | | Max cumulative indexing throttle time across primary shards | | 0 | min | | Cumulative merge time of primary shards | | 3.94532 | min | | Cumulative merge count of primary shards | | 28 | | | Min cumulative merge time across primary shards | | 0.716033 | min | | Median cumulative merge time across primary shards | | 0.777367 | min | | Max cumulative merge time across primary shards | | 0.891367 | min | | Cumulative merge throttle time of primary shards | | 0.963583 | min | | Min cumulative merge throttle time across primary shards | | 0.152167 | min | | Median cumulative merge throttle time across primary shards | | 0.186617 | min | | Max cumulative merge throttle time across primary shards | | 0.263517 | min | |

Cumulative refresh time of primary shards | | 1.00345 | min | | Cumulative refresh count of primary shards | | 92 | | | Min cumulative refresh time across primary shards | | 0.1888 | min | | Median cumulative refresh time across primary shards | | 0.197067 | min | | Max cumulative refresh time across primary shards | | 0.221383 | min | | Cumulative flush time of primary shards | | 0.188783 | min | | Cumulative flush count of primary shards | | 10 | | | Min cumulative flush time across primary shards | | 0.0135667 | min | | Median cumulative flush time across primary shards | | 0.0423833 | min | | Max cumulative flush time across primary shards | | 0.0461333 | min | | Total Young Gen GC time | | 5.371 | s | | Total Young Gen GC count | | 1232 | | | Total Old Gen GC time | | 0 | s | | Total Old Gen GC count | | 0 | | | Store size | | 3.02096 | GB | | Translog size | | 2.56114e-07 | GB | | Heap used for segments | | 0.62178 | MB | | Heap used for doc values | | 0.0332794 | MB | | Heap used for terms | | 0.479797 | MB | | Heap used for norms | | 0.0652466 | MB | | Heap used for points | | 0 | MB | | Heap used for stored fields | | 0.043457 | MB | | Segment count | | 84 | | | Total Ingest Pipeline count | | 0 | | | Total Ingest Pipeline time | | 0 | s | | Total Ingest Pipeline failed | | 0 | | | Min Throughput | index-append | 157765 | docs/s | | Mean Throughput | index-append | 170915 | docs/s | | Median Throughput | index-append | 167959 | docs/s | | Max Throughput | index-append | 189560 | docs/s | | 50th percentile latency | index-append | 195.387 | ms | | 90th percentile latency | index-append | 450.982 | ms | | 99th percentile latency | index-append | 1269.04 | ms | | 99.9th percentile latency | index-append | 1749.25 | ms | | 100th percentile latency | index-append | 1960.89 | ms | | 50th percentile service time | index-append | 195.387 | ms | | 90th percentile service time | index-append | 450.982 | ms | | 99th percentile service time | index-append | 1269.04 | ms | | 99.9th percentile service time | index-append | 1749.25 | ms | | 100th percentile service time | index-append | 1960.89 | ms | | error rate | index-append | 0 | % | | Min Throughput | index-stats | 90.03 | ops/s | | Mean Throughput | index-stats | 90.05 | ops/s | | Median Throughput | index-stats | 90.05 | ops/s | | Max Throughput | index-stats | 90.1 | ops/s | | 50th percentile latency | index-stats | 2.83893 | ms | | 90th percentile latency | index-stats | 3.76465 | ms | | 99th percentile latency | index-stats | 4.13091 | ms | | 99.9th percentile latency | index-stats | 4.47985 | ms | | 100th percentile latency | index-stats | 4.62963 | ms | | 50th percentile service time | index-stats | 1.49531 | ms | | 90th percentile service time | index-stats | 1.8023 | ms | | 99th percentile service time | index-stats | 2.0215 | ms | | 99.9th percentile service time | index-stats | 2.24679 | ms | | 100th percentile service time | index-stats | 2.27332 | ms | | error rate | index-stats | 0 | % | | Min Throughput | node-stats | 89.77 | ops/s | | Mean Throughput | node-stats | 90.07 | ops/s | | Median Throughput | node-stats | 90.06 | ops/s | | Max Throughput | node-stats | 90.28 | ops/s | | 50th percentile latency | node-stats | 3.23979 | ms | | 90th percentile latency | node-stats | 3.76533 | ms | | 99th percentile latency | node-stats | 4.62058 | ms | | 99.9th percentile latency | node-stats | 8.79175 | ms | | 100th percentile latency | node-stats | 16.7832 | ms | | 50th percentile service time | node-stats | 2.29505 | ms | | 90th percentile service time | node-stats | 2.51091 | ms | | 99th percentile service time | node-stats | 3.31319 | ms | | 99.9th percentile service time | node-stats | 4.70312 | ms | | 100th percentile service time | node-stats | 5.00144 | ms | | error rate | node-stats | 0 | % | | Min Throughput | default | 50.02 | ops/s | | Mean Throughput | default | 50.04 | ops/s | | Median Throughput | default | 50.04 | ops/s | | Max Throughput | default | 50.06 | ops/s | | 50th percentile latency | default | 3.11821 | ms | | 90th percentile latency | default | 4.42541 | ms | | 99th percentile latency | default | 4.69015 | ms | | 99.9th percentile latency | default | 4.98554 | ms | | 100th percentile latency | default | 5.08235 | ms | | 50th percentile service time | default | 1.95288 | ms | | 90th percentile service time | default | 2.1792 | ms | | 99th percentile service time | default | 2.52395 | ms | | 99.9th percentile service time | default | 3.02195 | ms | | 100th percentile service time | default | 3.50996 | ms | | error rate | default | 0 | % | | Min Throughput | term | 100.02 | ops/s | | Mean Throughput | term | 100.04 | ops/s | | Median Throughput | term | 100.04 | ops/s | | Max Throughput | term | 100.06 | ops/s | | 50th percentile latency | term | 2.97326 | ms | | 90th percentile latency | term | 3.48226 | ms | | 99th percentile latency | term | 3.84898 | ms | | 99.9th percentile latency | term | 9.62081 | ms | | 100th percentile latency | term | 12.7532 | ms | | 50th percentile service time | term | 1.97194 | ms | | 90th percentile service time | term | 2.17499 | ms | | 99th percentile service time | term | 2.61251 | ms | | 99.9th percentile service time | term | 6.498 | ms | | 100th percentile service time | term | 11.9398 | ms | | error rate | term | 0 | % | | Min Throughput | phrase | 110.02 | ops/s | | Mean Throughput | phrase | 110.03 | ops/s | | Median Throughput | phrase | 110.03 | ops/s | | Max Throughput | phrase | 110.06 | ops/s | | 50th percentile latency | phrase | 2.98059 | ms | | 90th percentile latency | phrase | 3.45886 | ms | | 99th percentile latency | phrase | 4.20316 | ms | | 99.9th percentile latency | phrase | 14.7435 | ms | | 100th percentile latency | phrase | 21.5945 | ms | | 50th percentile service time | phrase | 1.99907 | ms | | 90th percentile service time | phrase | 2.2445 | ms | | 99th percentile service time | phrase | 2.91911 | ms | | 99.9th percentile service time | phrase | 13.2573 | ms | | 100th percentile service time | phrase | 20.1639 | ms | | error rate | phrase | 0 | % | | Min Throughput | country_agg_uncached | 3 | ops/s | | Mean Throughput | country_agg_uncached | 3 | ops/s | | Median Throughput | country_agg_uncached | 3 | ops/s | | Max Throughput |

country_agg_uncached | 3 | ops/s | | 50th percentile latency | country_agg_uncached | 149.076 | ms | | 90th percentile latency | country_agg_uncached | 157.751 | ms | | 99th percentile latency | country_agg_uncached | 162.43 | ms | | 100th percentile latency | country_agg_uncached | 163.716 | ms | | 50th percentile service time | country_agg_uncached | 147.889 | ms | | 90th percentile service time | country_agg_uncached | 156 | ms | | 99th percentile service time | country_agg_uncached | 161.547 | ms | | 100th percentile service time | country_agg_uncached | 162.731 | ms | | error rate | country_agg_uncached | 0 | % | | Min Throughput | country_agg_cached | 98.72 | ops/s | | Mean Throughput | country_agg_cached | 99.06 | ops/s | | Median Throughput | country_agg_cached | 99.09 | ops/s | | Max Throughput | country_agg_cached | 99.3 | ops/s | | 50th percentile latency | country_agg_cached | 2.81234 | ms | | 90th percentile latency | country_agg_cached | 3.27198 | ms | | 99th percentile latency | country_agg_cached | 3.70345 | ms | | 99.9th percentile latency | country_agg_cached | 4.05661 | ms | | 100th percentile latency | country_agg_cached | 4.45997 | ms | | 50th percentile service time | country_agg_cached | 1.78307 | ms | | 90th percentile service time | country_agg_cached | 2.01869 | ms | | 99th percentile service time | country_agg_cached | 2.40292 | ms | | 99.9th percentile service time | country_agg_cached | 3.47974 | ms | | 100th percentile service time | country_agg_cached | 3.52695 | ms | | error rate | country_agg_cached | 0 | % | | Min Throughput | scroll | 20.05 | pages/s | | Mean Throughput | scroll | 20.06 | pages/s | | Median Throughput | scroll | 20.06 | pages/s | | Max Throughput | scroll | 20.07 | pages/s | | 50th percentile latency | scroll | 224.553 | ms | | 90th percentile latency | scroll | 230.098 | ms | | 99th percentile latency | scroll | 236.964 | ms | | 100th percentile latency | scroll | 254.53 | ms | | 50th percentile service time | scroll | 221.995 | ms | | 90th percentile service time | scroll | 227.37 | ms | | 99th percentile service time | scroll | 234.79 | ms | | 100th percentile service time | scroll | 251.803 | ms | | error rate | scroll | 0 | % | | Min Throughput | expression | 1.5 | ops/s | | Mean Throughput | expression | 1.5 | ops/s | | Median Throughput | expression | 1.5 | ops/s | | Max Throughput | expression | 1.5 | ops/s | | 50th percentile latency | expression | 269.364 | ms | | 90th percentile latency | expression | 276.925 | ms | | 99th percentile latency | expression | 297.282 | ms | | 100th percentile latency | expression | 303.19 | ms | | 50th percentile service time | expression | 268.006 | ms | | 90th percentile service time | expression | 275.478 | ms | | 99th percentile service time | expression | 295.91 | ms | | 100th percentile service time | expression | 301.678 | ms | | error rate | expression | 0 | % | | Min Throughput | painless_static | 1.4 | ops/s | | Mean Throughput | painless_static | 1.4 | ops/s | | Median Throughput | painless_static | 1.4 | ops/s | | Max Throughput | painless_static | 1.4 | ops/s | | 50th percentile latency | painless_static | 316.759 | ms | | 90th percentile latency | painless_static | 325.201 | ms | | 99th percentile latency | painless_static | 362.284 | ms | | 100th percentile latency | painless_static | 371.81 | ms | | 50th percentile service time | painless_static | 315.386 | ms | | 90th percentile service time | painless_static | 323.56 | ms | | 99th percentile service time | painless_static | 361.033 | ms | | 100th percentile service time | painless_static | 370.653 | ms | | error rate | painless_static | 0 | % | | Min Throughput | painless_dynamic | 1.4 | ops/s | | Mean Throughput | painless_dynamic | 1.4 | ops/s | | Median Throughput | painless_dynamic | 1.4 | ops/s | | Max Throughput | painless_dynamic | 1.4 | ops/s | | 50th percentile latency | painless_dynamic | 312.26 | ms | | 90th percentile latency | painless_dynamic | 316.634 | ms | | 99th percentile latency | painless_dynamic | 324.82 | ms | | 100th percentile latency | painless_dynamic | 359.638 | ms | | 50th percentile service time | painless_dynamic | 310.871 | ms | | 90th percentile service time | painless_dynamic | 315.494 | ms | | 99th percentile service time | painless_dynamic | 322.884 | ms | | 100th percentile service time | painless_dynamic | 357.735 | ms | | error rate | painless_dynamic | 0 | % | | Min Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_function_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_function_score | 351.541 | ms | | 90th percentile latency | decay_geo_gauss_function_score | 355.485 | ms | | 99th percentile latency | decay_geo_gauss_function_score | 358.036 | ms | | 100th percentile latency | decay_geo_gauss_function_score | 361.465 | ms | | 50th percentile service time | decay_geo_gauss_function_score | 349.727 | ms | | 90th percentile service time | decay_geo_gauss_function_score | 353.724 | ms | | 99th percentile service time | decay_geo_gauss_function_score | 356.594 | ms | | 100th percentile service time | decay_geo_gauss_function_score | 359.936 | ms | | error rate | decay_geo_gauss_function_score | 0 | % | | Min Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_script_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_script_score | 356.908 | ms | | 90th percentile latency | decay_geo_gauss_script_score | 359.112 | ms | | 99th percentile latency | decay_geo_gauss_script_score | 363.265 | ms | | 100th percentile latency | decay_geo_gauss_script_score | 364.061 | ms | | 50th percentile service time | decay_geo_gauss_script_score | 355.026 | ms | | 90th percentile service time | decay_geo_gauss_script_score | 357.59 | ms |

| 99th percentile service time | decay_geo_gauss_script_score | 361.629 | ms | | 100th percentile service time | decay_geo_gauss_script_score | 362.214 | ms | | error rate | decay_geo_gauss_script_score | 0 | % | | Min Throughput | field_value_function_score | 1.5 | ops/s | | Mean Throughput | field_value_function_score | 1.5 | ops/s | | Median Throughput | field_value_function_score | 1.5 | ops/s | | Max Throughput | field_value_function_score | 1.51 | ops/s | | 50th percentile latency | field_value_function_score | 123.162 | ms | | 90th percentile latency | field_value_function_score | 133.045 | ms | | 99th percentile latency | field_value_function_score | 149.976 | ms | | 100th percentile latency | field_value_function_score | 150.608 | ms | | 50th percentile service time | field_value_function_score | 121.782 | ms | | 90th percentile service time | field_value_function_score | 131.558 | ms | | 99th percentile service time | field_value_function_score | 148.563 | ms | | 100th percentile service time | field_value_function_score | 149.143 | ms | | error rate | field_value_function_score | 0 | % | | Min Throughput | field_value_script_score | 1.5 | ops/s | | Mean Throughput | field_value_script_score | 1.5 | ops/s | | Median Throughput | field_value_script_score | 1.5 | ops/s | | Max Throughput | field_value_script_score | 1.51 | ops/s | | 50th percentile latency | field_value_script_score | 155.57 | ms | | 90th percentile latency | field_value_script_score | 158.957 | ms | | 99th percentile latency | field_value_script_score | 162.297 | ms | | 100th percentile latency | field_value_script_score | 171.065 | ms | | 50th percentile service time | field_value_script_score | 154.36 | ms | | 90th percentile service time | field_value_script_score | 157.404 | ms | | 99th percentile service time | field_value_script_score | 160.578 | ms | | 100th percentile service time | field_value_script_score | 168.834 | ms | | error rate | field_value_script_score | 0 | % | | Min Throughput | large_terms | 1.1 | ops/s | | Mean Throughput | large_terms | 1.1 | ops/s | | Median Throughput | large_terms | 1.1 | ops/s | | Max Throughput | large_terms | 1.1 | ops/s | | 50th percentile latency | large_terms | 484.578 | ms | | 90th percentile latency | large_terms | 495.362 | ms | | 99th percentile latency | large_terms | 518.507 | ms | | 100th percentile latency | large_terms | 542.268 | ms | | 50th percentile service time | large_terms | 477.213 | ms | | 90th percentile service time | large_terms | 487.98 | ms | | 99th percentile service time | large_terms | 509.848 | ms | | 100th percentile service time | large_terms | 535.922 | ms | | error rate | large_terms | 0 | % | | Min Throughput | large_filtered_terms | 1.1 | ops/s | | Mean Throughput | large_filtered_terms | 1.1 | ops/s | | Median Throughput | large_filtered_terms | 1.1 | ops/s | | Max Throughput | large_filtered_terms | 1.1 | ops/s | | 50th percentile latency | large_filtered_terms | 493.048 | ms | | 90th percentile latency | large_filtered_terms | 508.977 | ms | | 99th percentile latency | large_filtered_terms | 523.258 | ms | | 100th percentile latency | large_filtered_terms | 524.487 | ms | | 50th percentile service time | large_filtered_terms | 486.102 | ms | | 90th percentile service time | large_filtered_terms | 501.647 | ms | | 99th percentile service time | large_filtered_terms | 515.519 | ms | | 100th percentile service time | large_filtered_terms | 518.476 | ms | | error rate | large_filtered_terms | 0 | % | | Min Throughput | large_prohibited_terms | 1.1 | ops/s | | Mean Throughput | large_prohibited_terms | 1.1 | ops/s | | Median Throughput | large_prohibited_terms | 1.1 | ops/s | | Max Throughput | large_prohibited_terms | 1.1 | ops/s | | 50th percentile latency | large_prohibited_terms | 482.855 | ms | | 90th percentile latency | large_prohibited_terms | 496.482 | ms | | 99th percentile latency | large_prohibited_terms | 512.956 | ms | | 100th percentile latency | large_prohibited_terms | 527.143 | ms | | 50th percentile service time | large_prohibited_terms | 475.813 | ms | | 90th percentile service time | large_prohibited_terms | 488.969 | ms | | 99th percentile service time | large_prohibited_terms | 506.16 | ms | | 100th percentile service time | large_prohibited_terms | 520.476 | ms | | error rate | large_prohibited_terms | 0 | % | | Min Throughput | desc_sort_population | 1.5 | ops/s | | Mean Throughput | desc_sort_population | 1.51 | ops/s | | Median Throughput | desc_sort_population | 1.51 | ops/s | | Max Throughput | desc_sort_population | 1.51 | ops/s | | 50th percentile latency | desc_sort_population | 57.536 | ms | | 90th percentile latency | desc_sort_population | 71.9505 | ms | | 99th percentile latency | desc_sort_population | 91.8661 | ms | | 100th percentile latency | desc_sort_population | 100.124 | ms | | 50th percentile service time | desc_sort_population | 55.7684 | ms | | 90th percentile service time | desc_sort_population | 70.5174 | ms | | 99th percentile service time | desc_sort_population | 90.3164 | ms | | 100th percentile service time | desc_sort_population | 98.1669 | ms | | error rate | desc_sort_population | 0 | % | | Min Throughput | asc_sort_population | 1.5 | ops/s | | Mean Throughput | asc_sort_population | 1.51 | ops/s | | Median Throughput | asc_sort_population | 1.51 | ops/s | | Max Throughput | asc_sort_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_population | 58.7567 | ms | | 90th percentile latency | asc_sort_population | 60.5398 | ms | | 99th percentile latency | asc_sort_population | 73.2742 | ms | | 100th percentile latency | asc_sort_population | 73.6542 | ms | | 50th percentile service time | asc_sort_population | 57.0029 | ms | | 90th percentile service time | asc_sort_population | 58.714 | ms | | 99th percentile service time | asc_sort_population | 71.6442 | ms | | 100th percentile service time | asc_sort_population | 71.82 | ms | | error rate | asc_sort_population | 0 | % | | Min Throughput | asc_sort_with_after_population | 1.5 | ops/s | | Mean Throughput | asc_sort_with_after_population | 1.51 | ops/s | | Median Throughput | asc_sort_with_after_population | 1.51 | ops/s | | Max Throughput |

asc_sort_with_after_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_with_after_population | 80.2785 | ms | | 90th percentile latency | asc_sort_with_after_population | 82.6272 | ms | | 99th percentile latency | asc_sort_with_after_population | 94.4026 | ms | | 100th percentile latency | asc_sort_with_after_population | 94.8459 | ms | | 50th percentile service time | asc_sort_with_after_population | 78.5427 | ms | | 90th percentile service time | asc_sort_with_after_population | 80.4041 | ms | | 99th percentile service time | asc_sort_with_after_population | 92.8593 | ms | | 100th percentile service time | asc_sort_with_after_population | 93.0199 | ms | | error rate | asc_sort_with_after_population | 0 | % | | Min Throughput | desc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | desc_sort_geonameid | 6.02 | ops/s | | Median Throughput | desc_sort_geonameid | 6.02 | ops/s | | Max Throughput | desc_sort_geonameid | 6.02 | ops/s | | 50th percentile latency | desc_sort_geonameid | 7.92538 | ms | | 90th percentile latency | desc_sort_geonameid | 8.42276 | ms | | 99th percentile latency | desc_sort_geonameid | 10.2115 | ms | | 100th percentile latency | desc_sort_geonameid | 11.1475 | ms | | 50th percentile service time | desc_sort_geonameid | 6.5583 | ms | | 90th percentile service time | desc_sort_geonameid | 6.89089 | ms | | 99th percentile service time | desc_sort_geonameid | 8.7876 | ms | | 100th percentile service time | desc_sort_geonameid | 10.0496 | ms | | error rate | desc_sort_geonameid | 0 | % | | Min Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | 50th percentile latency | desc_sort_with_after_geonameid | 75.2093 | ms | | 90th percentile latency | desc_sort_with_after_geonameid | 86.6912 | ms | | 99th percentile latency | desc_sort_with_after_geonameid | 90.4317 | ms | | 100th percentile latency | desc_sort_with_after_geonameid | 93.5113 | ms | | 50th percentile service time | desc_sort_with_after_geonameid | 73.79 | ms | | 90th percentile service time | desc_sort_with_after_geonameid | 85.2085 | ms | | 99th percentile service time | desc_sort_with_after_geonameid | 88.932 | ms | | 100th percentile service time | desc_sort_with_after_geonameid | 92.6169 | ms | | error rate | desc_sort_with_after_geonameid | 0 | % | | Min Throughput | asc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | asc_sort_geonameid | 6.02 | ops/s | | Median Throughput | asc_sort_geonameid | 6.02 | ops/s | | Max Throughput | asc_sort_geonameid | 6.03 | ops/s | | 50th percentile latency | asc_sort_geonameid | 5.94919 | ms | | 90th percentile latency | asc_sort_geonameid | 6.45275 | ms | | 99th percentile latency | asc_sort_geonameid | 7.60038 | ms | | 100th percentile latency | asc_sort_geonameid | 7.97628 | ms | | 50th percentile service time | asc_sort_geonameid | 4.61634 | ms | | 90th percentile service time | asc_sort_geonameid | 4.87425 | ms | | 99th percentile service time | asc_sort_geonameid | 6.38694 | ms | | 100th percentile service time | asc_sort_geonameid | 6.79052 | ms | | error rate | asc_sort_geonameid | 0 | % | | Min Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | asc_sort_with_after_geonameid | 6.02 | ops/s | | 50th percentile latency | asc_sort_with_after_geonameid | 70.0619 | ms | | 90th percentile latency | asc_sort_with_after_geonameid | 81.3791 | ms | | 99th percentile latency | asc_sort_with_after_geonameid | 89.1096 | ms | | 100th percentile latency | asc_sort_with_after_geonameid | 90.5249 | ms | | 50th percentile service time | asc_sort_with_after_geonameid | 68.8445 | ms | | 90th percentile service time | asc_sort_with_after_geonameid | 80.2913 | ms | | 99th percentile service time | asc_sort_with_after_geonameid | 87.9724 | ms | | 100th percentile service time | asc_sort_with_after_geonameid | 88.9337 | ms | | error rate | asc_sort_with_after_geonameid | 0 | % |

16核64GB512GB通用型的3个数据节点实例基准性能指标

16核64GB512GB通用型的3个数据节点实例基准性能指标

使用ESRally工具对数据节点规格为16核64GB的BES集群实例（7.10.2版本）进行基准测试。

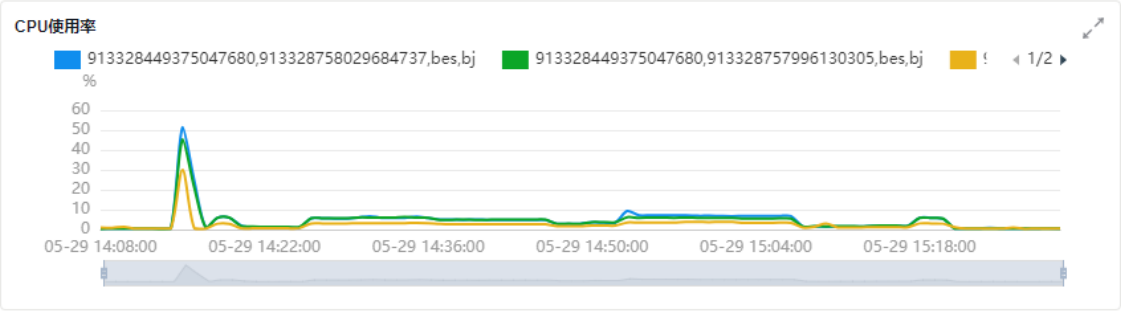
压测配置

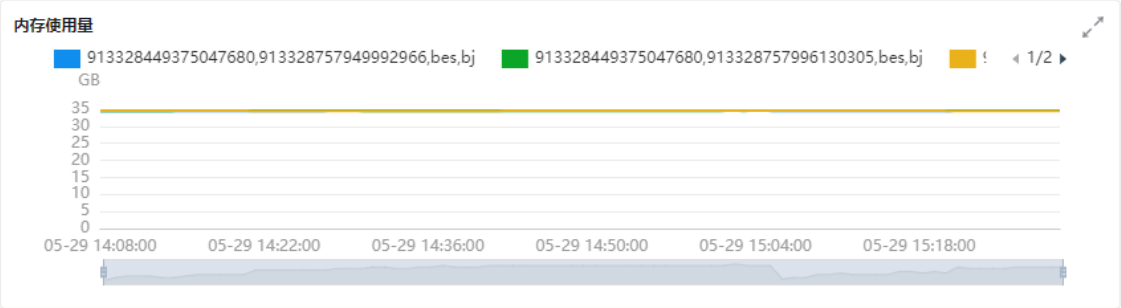
项目	说明
实例配置	版本：BES集群实例（7.10.2版本） 数据节点规格：16核64GB 数据节点存储性能级别：增强型SSD_PL1 数据单节点存储空间：512GB 数据节点数量：3个
ESRally配置	使用ESRally的默认配置tracks。
数据集	使用ESRally的预置数据集Geonames模拟测试场景，数据文档总数11396503。获取数据集，请查看ES Rally Hub获取。
分片数	主分片：6 副本数：0
bulk_size	每次批量操作提交5000个文档。
bulk_indexing_clients	同时执行批量索引操作的客户端数量为8个。
index_append warmup time	从默认的120s修改为30s，当集群配置高时，index_append在预热启动期间就完成了。

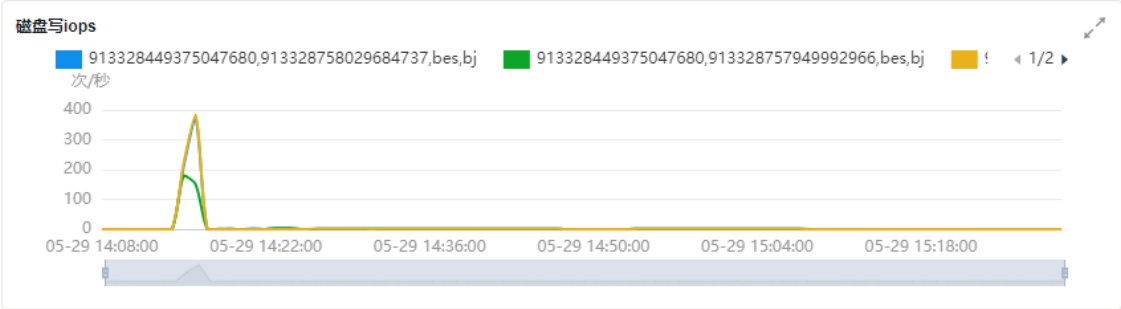
压测结果

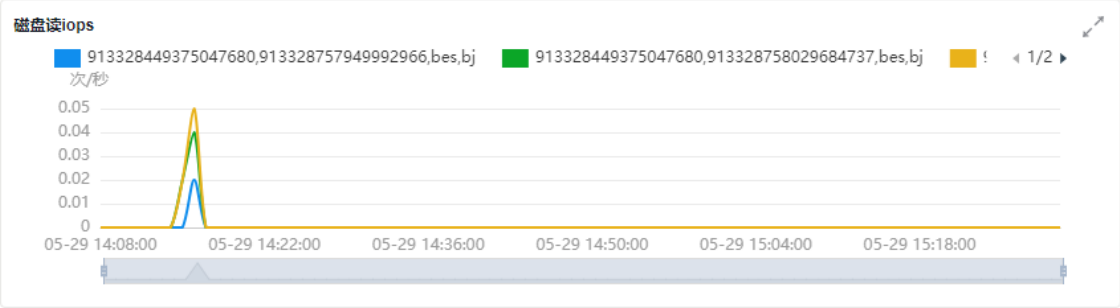
压测结果仅供参考，无法代表实际生产中写入查询情况，建议您结合业务生产数据进行压测。

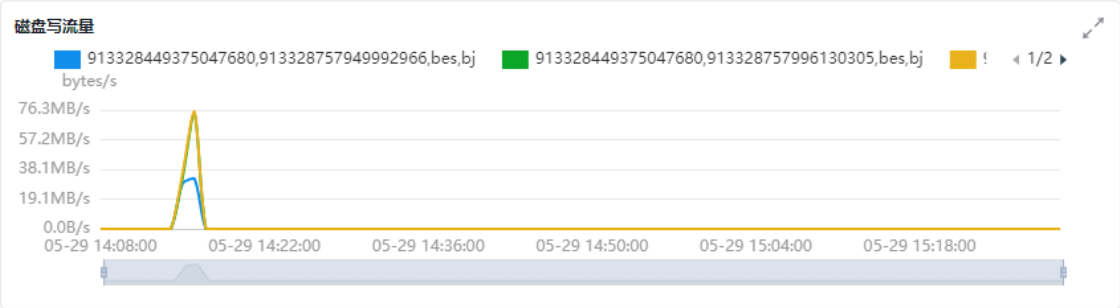
主要压测指标结果

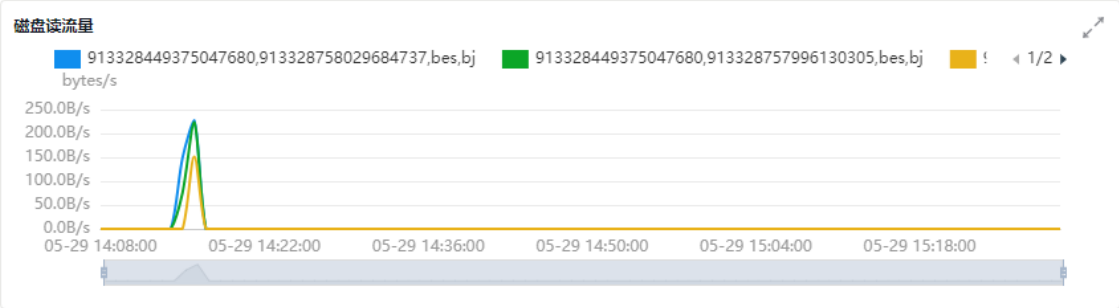


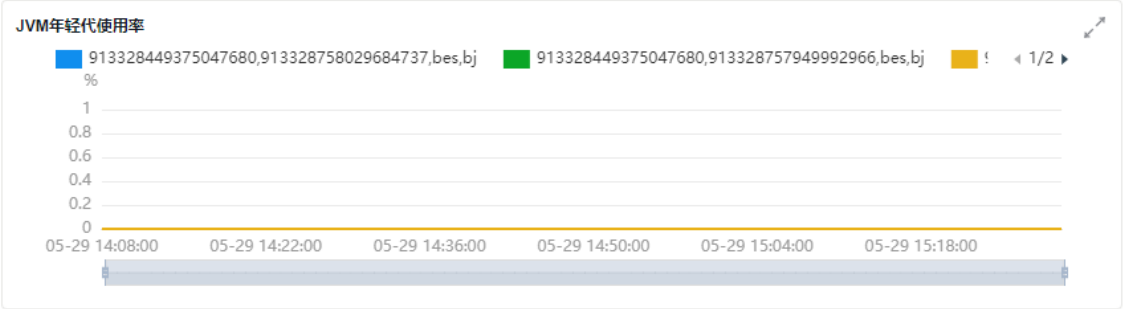


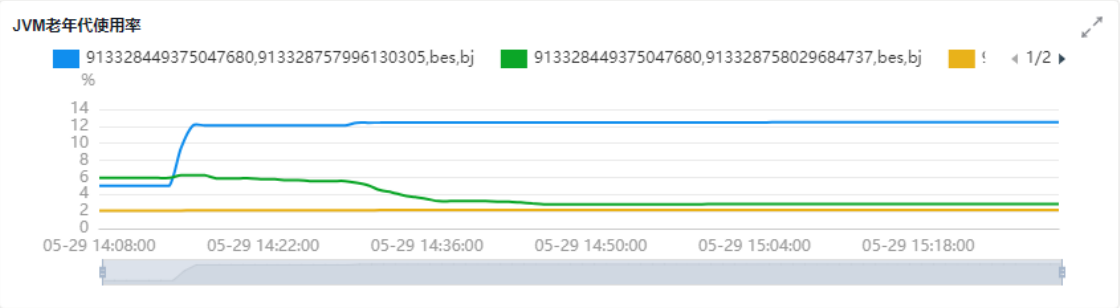


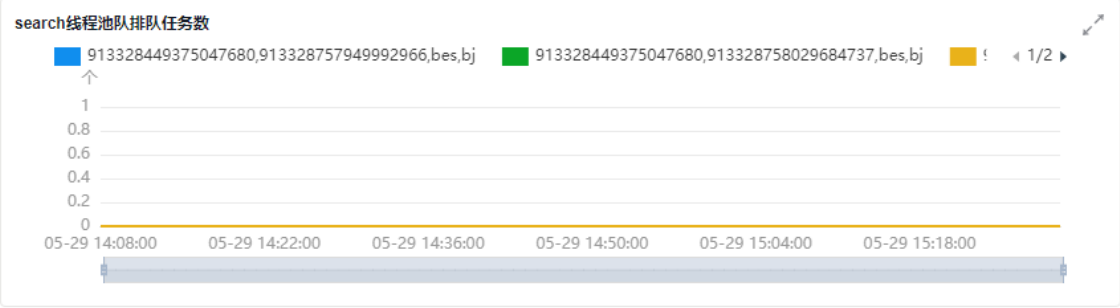

磁盘读iops
913328449375047680,913328757949992966,bes,bj
913328449375047680,913328758029684737,bes,bj
1/2
次/秒
0.05
0.04
0.03
0.02
0.01
0
05-29 14:08:00
05-29 14:22:00
05-29 14:36:00
05-29 14:50:00
05-29 15:04:00
05-29 15:18:00

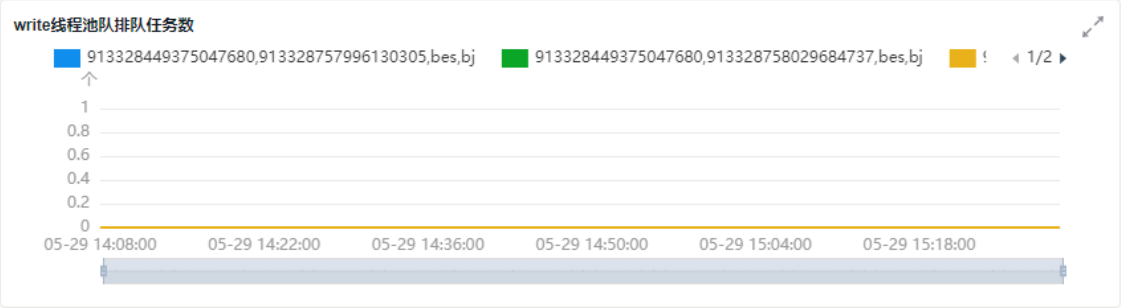

磁盘写流量
913328449375047680,913328757949992966,bes,bj
913328449375047680,913328757996130305,bes,bj
1/2
bytes/s
76.3MB/s
57.2MB/s
38.1MB/s
19.1MB/s
0.0B/s
05-29 14:08:00
05-29 14:22:00
05-29 14:36:00
05-29 14:50:00
05-29 15:04:00
05-29 15:18:00


磁盘读流量
913328449375047680,913328758029684737,bes,bj
913328449375047680,913328757996130305,bes,bj
1/2
bytes/s
250.0B/s
200.0B/s
150.0B/s
100.0B/s
50.0B/s
0.0B/s
05-29 14:08:00
05-29 14:22:00
05-29 14:36:00
05-29 14:50:00
05-29 15:04:00
05-29 15:18:00


JVM年轻代使用率
913328449375047680,913328758029684737,bes,bj
913328449375047680,913328757949992966,bes,bj
1/2
%
1
0.8
0.6
0.4
0.2
0
05-29 14:08:00
05-29 14:22:00
05-29 14:36:00
05-29 14:50:00
05-29 15:04:00
05-29 15:18:00


JVM老年代使用率
913328449375047680,913328757996130305,bes,bj
913328449375047680,913328758029684737,bes,bj
1/2
%
14
12
10
8
6
4
2
0
05-29 14:08:00
05-29 14:22:00
05-29 14:36:00
05-29 14:50:00
05-29 15:04:00
05-29 15:18:00


search线程池队排队任务数
913328449375047680,913328757949992966,bes,bj
913328449375047680,913328758029684737,bes,bj
1/2
1
0.8
0.6
0.4
0.2
0
05-29 14:08:00
05-29 14:22:00
05-29 14:36:00
05-29 14:50:00
05-29 15:04:00
05-29 15:18:00



基准压测报告 | Metric | Task | Value | Unit | |---|---|---|---| | Cumulative indexing time of primary shards | | 12.0343 | min | | Min cumulative indexing time across primary shards | | 2.10367 | min | | Median cumulative indexing time across primary shards | | 2.44152 | min | | Max cumulative indexing time across primary shards | | 2.54422 | min | | Cumulative indexing throttle time of primary shards | | 0 | min | | Min cumulative indexing throttle time across primary shards | | 0 | min | | Median cumulative indexing throttle time across primary shards | | 0 | min | | Max cumulative indexing throttle time across primary shards | | 0 | min | | Cumulative merge time of primary shards | | 3.5264 | min | | Cumulative merge count of primary shards | | 34 | | | Min cumulative merge time across primary shards | | 0.615117 | min | | Median cumulative merge time across primary shards | | 0.6916 | min | | Max cumulative merge time across primary shards | | 0.783883 | min | | Cumulative merge throttle time of primary shards | | 0.917383 | min | | Min cumulative merge throttle time across primary shards | | 0.16 | min | | Median cumulative merge throttle time across primary shards | | 0.162583 | min | | Max cumulative merge throttle time across primary shards | | 0.221383 | min | | Cumulative refresh time of primary shards | | 0.72385 | min | | Cumulative refresh count of primary shards | | 76 | | | Min cumulative refresh time across primary shards | | 0.137933 | min | | Median cumulative refresh time across primary shards | | 0.143717 | min | | Max cumulative refresh time across primary shards | | 0.15445 | min | | Cumulative flush time of primary shards | | 0.10835 | min | | Cumulative flush count of primary shards | | 10 | | | Min cumulative flush time across primary shards | | 0.014 | min | | Median cumulative flush time across primary shards | | 0.0197833 | min | | Max cumulative flush time across primary shards | | 0.0299667 | min | | Total Young Gen GC time | | 1.947 | s | | Total Young Gen GC count | | 327 | | | Total Old Gen GC time | | 0 | s | | Total Old Gen GC count | | 0 | | | Store size | | 3.09734 | GB | | Translog size | | 2.56114e-07 | GB | | Heap used for segments | | 0.636745 | MB | | Heap used for doc values | | 0.0364571 | MB | | Heap used for terms | | 0.489899 | MB | | Heap used for norms | | 0.0666504 | MB | | Heap used for points | | 0 | MB | | Heap used for stored fields | | 0.0437393 | MB | | Segment count | | 85 | | | Total Ingest Pipeline count | | 0 | | | Total Ingest Pipeline time | | 0 | s | | Total Ingest Pipeline failed | | 0 | | | Min Throughput | index-append | 233861 | docs/s | | Mean Throughput | index-append | 241297 | docs/s | | Median Throughput | index-append | 242239 | docs/s | | Max Throughput | index-append | 247904 | docs/s | | 50th percentile latency | index-append | 104.497 | ms | | 90th percentile latency | index-append | 384.387 | ms | | 99th percentile latency | index-append | 1171.61 | ms | | 100th percentile latency | index-append | 1544.05 | ms | | 50th percentile service time | index-append | 104.497 | ms | | 90th percentile service time | index-append | 384.387 | ms | | 99th percentile service time | index-append | 1171.61 | ms | | 100th percentile service time | index-append | 1544.05 | ms | | error rate | index-append | 0 | % | | Min Throughput | index-stats | 90.03 | ops/s | | Mean Throughput | index-stats | 90.05 | ops/s | | Median Throughput | index-stats | 90.05 | ops/s | | Max Throughput | index-stats | 90.1 | ops/s | | 50th percentile latency | index-stats | 2.79773 | ms | | 90th percentile latency | index-stats | 3.62359 | ms | | 99th percentile latency | index-stats | 4.02021 | ms | | 99.9th percentile latency | index-stats | 4.77639 | ms | | 100th percentile latency | index-stats | 4.79167 | ms | | 50th percentile service time | index-stats | 1.46091 | ms | | 90th percentile service time | index-stats | 1.67085 | ms | | 99th percentile service time | index-stats | 2.34065 | ms | | 99.9th percentile service time | index-stats | 2.73045 | ms | | 100th percentile service time | index-stats | 2.9983 | ms | | error rate | index-stats | 0 | % | | Min Throughput | node-stats | 90.04 | ops/s | | Mean Throughput | node-stats | 90.08 | ops/s | | Median Throughput | node-stats | 90.06 | ops/s | | Max Throughput | node-stats | 90.2 | ops/s | | 50th percentile latency | node-stats | 3.24468 | ms | | 90th percentile latency | node-stats | 3.73432 | ms | | 99th percentile latency | node-stats | 4.6546 | ms | | 99.9th percentile latency | node-stats | 6.81514 | ms | | 100th percentile latency | node-stats | 15.1143 | ms | | 50th percentile service time | node-stats | 2.3205 | ms | | 90th percentile service time | node-stats | 2.57523 | ms | | 99th percentile service time | node-stats | 3.40415 | ms | | 99.9th percentile service time | node-stats | 3.90179 | ms | | 100th percentile service time | node-stats | 4.02071 | ms | | error rate | node-stats | 0 | % | | Min Throughput | default | 50.02 | ops/s | | Mean Throughput | default | 50.04 | ops/s | | Median Throughput | default | 50.04 | ops/s | | Max Throughput | default | 50.07 | ops/s | | 50th percentile latency | default | 3.13572

| ms | | 90th percentile latency | default | 4.35137 | ms | | 99th percentile latency | default | 4.7889 | ms | | 99.9th percentile latency | default | 5.1395 | ms | | 100th percentile latency | default | 10.2728 | ms | | 50th percentile service time | default | 2.03777 | ms | | 90th percentile service time | default | 2.30738 | ms | | 99th percentile service time | default | 2.66234 | ms | | 99.9th percentile service time | default | 4.02789 | ms | | 100th percentile service time | default | 9.75839 | ms | | error rate | default | 0 | % | | Min Throughput | term | 100.03 | ops/s | | Mean Throughput | term | 100.06 | ops/s | | Median Throughput | term | 100.05 | ops/s | | Max Throughput | term | 100.1 | ops/s | | 50th percentile latency | term | 2.93054 | ms | | 90th percentile latency | term | 3.44617 | ms | | 99th percentile latency | term | 3.88855 | ms | | 99.9th percentile latency | term | 4.81783 | ms | | 100th percentile latency | term | 5.23642 | ms | | 50th percentile service time | term | 2.01621 | ms | | 90th percentile service time | term | 2.2517 | ms | | 99th percentile service time | term | 2.82973 | ms | | 99.9th percentile service time | term | 3.97545 | ms | | 100th percentile service time | term | 4.20495 | ms | | error rate | term | 0 | % | | Min Throughput | phrase | 110.03 | ops/s | | Mean Throughput | phrase | 110.04 | ops/s | | Median Throughput | phrase | 110.04 | ops/s | | Max Throughput | phrase | 110.06 | ops/s | | 50th percentile latency | phrase | 3.08126 | ms | | 90th percentile latency | phrase | 3.61041 | ms | | 99th percentile latency | phrase | 53.9258 | ms | | 99.9th percentile latency | phrase | 114.888 | ms | | 100th percentile latency | phrase | 121.149 | ms | | 50th percentile service time | phrase | 2.10698 | ms | | 90th percentile service time | phrase | 2.40441 | ms | | 99th percentile service time | phrase | 3.0833 | ms | | 99.9th percentile service time | phrase | 18.6289 | ms | | 100th percentile service time | phrase | 119.816 | ms | | error rate | phrase | 0 | % | | Min Throughput | country_agg_uncached | 3 | ops/s | | Mean Throughput | country_agg_uncached | 3 | ops/s | | Median Throughput | country_agg_uncached | 3 | ops/s | | Max Throughput | country_agg_uncached | 3 | ops/s | | 50th percentile latency | country_agg_uncached | 151.681 | ms | | 90th percentile latency | country_agg_uncached | 170.235 | ms | | 99th percentile latency | country_agg_uncached | 178.553 | ms | | 100th percentile latency | country_agg_uncached | 183.564 | ms | | 50th percentile service time | country_agg_uncached | 150.278 | ms | | 90th percentile service time | country_agg_uncached | 169.046 | ms | | 99th percentile service time | country_agg_uncached | 177.082 | ms | | 100th percentile service time | country_agg_uncached | 181.96 | ms | | error rate | country_agg_uncached | 0 | % | | Min Throughput | country_agg_cached | 98.7 | ops/s | | Mean Throughput | country_agg_cached | 99.04 | ops/s | | Median Throughput | country_agg_cached | 99.08 | ops/s | | Max Throughput | country_agg_cached | 99.28 | ops/s | | 50th percentile latency | country_agg_cached | 2.63408 | ms | | 90th percentile latency | country_agg_cached | 3.04244 | ms | | 99th percentile latency | country_agg_cached | 3.52889 | ms | | 99.9th percentile latency | country_agg_cached | 3.90273 | ms | | 100th percentile latency | country_agg_cached | 4.0691 | ms | | 50th percentile service time | country_agg_cached | 1.65862 | ms | | 90th percentile service time | country_agg_cached | 1.86344 | ms | | 99th percentile service time | country_agg_cached | 2.22166 | ms | | 99.9th percentile service time | country_agg_cached | 2.60855 | ms | | 100th percentile service time | country_agg_cached | 2.6347 | ms | | error rate | country_agg_cached | 0 | % | | Min Throughput | scroll | 20.05 | pages/s | | Mean Throughput | scroll | 20.06 | pages/s | | Median Throughput | scroll | 20.06 | pages/s | | Max Throughput | scroll | 20.08 | pages/s | | 50th percentile latency | scroll | 234.894 | ms | | 90th percentile latency | scroll | 238.723 | ms | | 99th percentile latency | scroll | 251.606 | ms | | 100th percentile latency | scroll | 252.065 | ms | | 50th percentile service time | scroll | 232.651 | ms | | 90th percentile service time | scroll | 236.106 | ms | | 99th percentile service time | scroll | 248.883 | ms | | 100th percentile service time | scroll | 248.996 | ms | | error rate | scroll | 0 | % | | Min Throughput | expression | 1.5 | ops/s | | Mean Throughput | expression | 1.5 | ops/s | | Median Throughput | expression | 1.5 | ops/s | | Max Throughput | expression | 1.5 | ops/s | | 50th percentile latency | expression | 277.092 | ms | | 90th percentile latency | expression | 283.323 | ms | | 99th percentile latency | expression | 301.44 | ms | | 100th percentile latency | expression | 304.69 | ms | | 50th percentile service time | expression | 275.476 | ms | | 90th percentile service time | expression | 281.42 | ms | | 99th percentile service time | expression | 300.013 | ms | | 100th percentile service time | expression | 303.609 | ms | | error rate | expression | 0 | % | | Min Throughput | painless_static | 1.4 | ops/s | | Mean Throughput | painless_static | 1.4 | ops/s | | Median Throughput | painless_static | 1.4 | ops/s | | Max Throughput | painless_static | 1.4 | ops/s | | 50th percentile latency | painless_static | 317.442 | ms | | 90th percentile latency | painless_static | 334.858 | ms | | 99th percentile latency | painless_static | 340.92 | ms | | 100th percentile latency | painless_static | 341.893 | ms | | 50th percentile service time | painless_static | 316.19 | ms | | 90th percentile service time | painless_static | 333.574 | ms | | 99th percentile service time | painless_static | 339.803 | ms | | 100th percentile service time | painless_static | 340.474 | ms | | error rate | painless_static | 0 | % | | Min Throughput | painless_dynamic | 1.4 | ops/s | | Mean Throughput | painless_dynamic | 1.4 | ops/s | | Median Throughput | painless_dynamic | 1.4 | ops/s | | Max Throughput | painless_dynamic | 1.4 | ops/s | | 50th percentile latency | painless_dynamic | 315.694 | ms | | 90th percentile latency | painless_dynamic | 321.423 | ms | | 99th percentile latency |

painless_dynamic | 337.969 | ms | | 100th percentile latency | painless_dynamic | 342.168 | ms | | 50th percentile service time | painless_dynamic | 314.424 | ms | | 90th percentile service time | painless_dynamic | 319.92 | ms | | 99th percentile service time | painless_dynamic | 336.644 | ms | | 100th percentile service time | painless_dynamic | 340.635 | ms | | error rate | painless_dynamic | 0 | % | | Min Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_function_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_function_score | 359.214 | ms | | 90th percentile latency | decay_geo_gauss_function_score | 368.352 | ms | | 99th percentile latency | decay_geo_gauss_function_score | 373.97 | ms | | 100th percentile latency | decay_geo_gauss_function_score | 374.069 | ms | | 50th percentile service time | decay_geo_gauss_function_score | 357.655 | ms | | 90th percentile service time | decay_geo_gauss_function_score | 366.644 | ms | | 99th percentile service time | decay_geo_gauss_function_score | 372.096 | ms | | 100th percentile service time | decay_geo_gauss_function_score | 372.425 | ms | | error rate | decay_geo_gauss_function_score | 0 | % | | Min Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_script_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_script_score | 363.106 | ms | | 90th percentile latency | decay_geo_gauss_script_score | 365.481 | ms | | 99th percentile latency | decay_geo_gauss_script_score | 374.609 | ms | | 100th percentile latency | decay_geo_gauss_script_score | 376.065 | ms | | 50th percentile service time | decay_geo_gauss_script_score | 361.383 | ms | | 90th percentile service time | decay_geo_gauss_script_score | 363.777 | ms | | 99th percentile service time | decay_geo_gauss_script_score | 373.431 | ms | | 100th percentile service time | decay_geo_gauss_script_score | 375.085 | ms | | error rate | decay_geo_gauss_script_score | 0 | % | | Min Throughput | field_value_function_score | 1.5 | ops/s | | Mean Throughput | field_value_function_score | 1.5 | ops/s | | Median Throughput | field_value_function_score | 1.5 | ops/s | | Max Throughput | field_value_function_score | 1.51 | ops/s | | 50th percentile latency | field_value_function_score | 127.949 | ms | | 90th percentile latency | field_value_function_score | 130.053 | ms | | 99th percentile latency | field_value_function_score | 144.821 | ms | | 100th percentile latency | field_value_function_score | 148.107 | ms | | 50th percentile service time | field_value_function_score | 126.362 | ms | | 90th percentile service time | field_value_function_score | 128.247 | ms | | 99th percentile service time | field_value_function_score | 143.593 | ms | | 100th percentile service time | field_value_function_score | 145.336 | ms | | error rate | field_value_function_score | 0 | % | | Min Throughput | field_value_script_score | 1.5 | ops/s | | Mean Throughput | field_value_script_score | 1.5 | ops/s | | Median Throughput | field_value_script_score | 1.5 | ops/s | | Max Throughput | field_value_script_score | 1.51 | ops/s | | 50th percentile latency | field_value_script_score | 163.99 | ms | | 90th percentile latency | field_value_script_score | 172.615 | ms | | 99th percentile latency | field_value_script_score | 185.59 | ms | | 100th percentile latency | field_value_script_score | 188.832 | ms | | 50th percentile service time | field_value_script_score | 162.292 | ms | | 90th percentile service time | field_value_script_score | 171.111 | ms | | 99th percentile service time | field_value_script_score | 183.541 | ms | | 100th percentile service time | field_value_script_score | 187.705 | ms | | error rate | field_value_script_score | 0 | % | | Min Throughput | large_terms | 1.1 | ops/s | | Mean Throughput | large_terms | 1.1 | ops/s | | Median Throughput | large_terms | 1.1 | ops/s | | Max Throughput | large_terms | 1.1 | ops/s | | 50th percentile latency | large_terms | 596.243 | ms | | 90th percentile latency | large_terms | 603.62 | ms | | 99th percentile latency | large_terms | 620.6 | ms | | 100th percentile latency | large_terms | 623.707 | ms | | 50th percentile service time | large_terms | 589.628 | ms | | 90th percentile service time | large_terms | 597.426 | ms | | 99th percentile service time | large_terms | 614.262 | ms | | 100th percentile service time | large_terms | 617.034 | ms | | error rate | large_terms | 0 | % | | Min Throughput | large_filtered_terms | 1.1 | ops/s | | Mean Throughput | large_filtered_terms | 1.1 | ops/s | | Median Throughput | large_filtered_terms | 1.1 | ops/s | | Max Throughput | large_filtered_terms | 1.1 | ops/s | | 50th percentile latency | large_filtered_terms | 599.322 | ms | | 90th percentile latency | large_filtered_terms | 604.024 | ms | | 99th percentile latency | large_filtered_terms | 617.841 | ms | | 100th percentile latency | large_filtered_terms | 619.886 | ms | | 50th percentile service time | large_filtered_terms | 592.66 | ms | | 90th percentile service time | large_filtered_terms | 597.505 | ms | | 99th percentile service time | large_filtered_terms | 610.958 | ms | | 100th percentile service time | large_filtered_terms | 612.832 | ms | | error rate | large_filtered_terms | 0 | % | | Min Throughput | large_prohibited_terms | 1.1 | ops/s | | Mean Throughput | large_prohibited_terms | 1.1 | ops/s | | Median Throughput | large_prohibited_terms | 1.1 | ops/s | | Max Throughput | large_prohibited_terms | 1.1 | ops/s | | 50th percentile latency | large_prohibited_terms | 577.937 | ms | | 90th percentile latency | large_prohibited_terms | 581.529 | ms | | 99th percentile latency | large_prohibited_terms | 596.318 | ms | | 100th percentile latency | large_prohibited_terms | 599.859 | ms | | 50th percentile service time |

large_prohibited_terms | 570.86 | ms | | 90th percentile service time | large_prohibited_terms | 574.387 | ms | | 99th percentile service time | large_prohibited_terms | 588.893 | ms | | 100th percentile service time | large_prohibited_terms | 592.835 | ms | | error rate | large_prohibited_terms | 0 | % | | Min Throughput | desc_sort_population | 1.5 | ops/s | | Mean Throughput | desc_sort_population | 1.51 | ops/s | | Median Throughput | desc_sort_population | 1.51 | ops/s | | Max Throughput | desc_sort_population | 1.51 | ops/s | | 50th percentile latency | desc_sort_population | 58.0235 | ms | | 90th percentile latency | desc_sort_population | 60.1249 | ms | | 99th percentile latency | desc_sort_population | 85.2941 | ms | | 100th percentile latency | desc_sort_population | 85.7457 | ms | | 50th percentile service time | desc_sort_population | 56.389 | ms | | 90th percentile service time | desc_sort_population | 58.1864 | ms | | 99th percentile service time | desc_sort_population | 83.7822 | ms | | 100th percentile service time | desc_sort_population | 83.955 | ms | | error rate | desc_sort_population | 0 | % | | Min Throughput | asc_sort_population | 1.5 | ops/s | | Mean Throughput | asc_sort_population | 1.51 | ops/s | | Median Throughput | asc_sort_population | 1.51 | ops/s | | Max Throughput | asc_sort_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_population | 59.1134 | ms | | 90th percentile latency | asc_sort_population | 60.5708 | ms | | 99th percentile latency | asc_sort_population | 77.2832 | ms | | 100th percentile latency | asc_sort_population | 89.8928 | ms | | 50th percentile service time | asc_sort_population | 57.3594 | ms | | 90th percentile service time | asc_sort_population | 59.0197 | ms | | 99th percentile service time | asc_sort_population | 75.7874 | ms | | 100th percentile service time | asc_sort_population | 88.1853 | ms | | error rate | asc_sort_population | 0 | % | | Min Throughput | asc_sort_with_after_population | 1.5 | ops/s | | Mean Throughput | asc_sort_with_after_population | 1.51 | ops/s | | Median Throughput | asc_sort_with_after_population | 1.51 | ops/s | | Max Throughput | asc_sort_with_after_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_with_after_population | 79.5535 | ms | | 90th percentile latency | asc_sort_with_after_population | 81 | ms | | 99th percentile latency | asc_sort_with_after_population | 105.637 | ms | | 100th percentile latency | asc_sort_with_after_population | 108.186 | ms | | 50th percentile service time | asc_sort_with_after_population | 77.6818 | ms | | 90th percentile service time | asc_sort_with_after_population | 79.3232 | ms | | 99th percentile service time | asc_sort_with_after_population | 103.844 | ms | | 100th percentile service time | asc_sort_with_after_population | 106.852 | ms | | error rate | asc_sort_with_after_population | 0 | % | | Min Throughput | desc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | desc_sort_geonameid | 6.02 | ops/s | | Median Throughput | desc_sort_geonameid | 6.02 | ops/s | | Max Throughput | desc_sort_geonameid | 6.03 | ops/s | | 50th percentile latency | desc_sort_geonameid | 6.38591 | ms | | 90th percentile latency | desc_sort_geonameid | 6.80141 | ms | | 99th percentile latency | desc_sort_geonameid | 7.43864 | ms | | 100th percentile latency | desc_sort_geonameid | 8.4132 | ms | | 50th percentile service time | desc_sort_geonameid | 5.12179 | ms | | 90th percentile service time | desc_sort_geonameid | 5.4492 | ms | | 99th percentile service time | desc_sort_geonameid | 6.54057 | ms | | 100th percentile service time | desc_sort_geonameid | 6.68928 | ms | | error rate | desc_sort_geonameid | 0 | % | | Min Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | desc_sort_with_after_geonameid | 6.02 | ops/s | | 50th percentile latency | desc_sort_with_after_geonameid | 75.4583 | ms | | 90th percentile latency | desc_sort_with_after_geonameid | 82.6942 | ms | | 99th percentile latency | desc_sort_with_after_geonameid | 100.328 | ms | | 100th percentile latency | desc_sort_with_after_geonameid | 107.778 | ms | | 50th percentile service time | desc_sort_with_after_geonameid | 74.2668 | ms | | 90th percentile service time | desc_sort_with_after_geonameid | 81.3509 | ms | | 99th percentile service time | desc_sort_with_after_geonameid | 98.6706 | ms | | 100th percentile service time | desc_sort_with_after_geonameid | 107.126 | ms | | error rate | desc_sort_with_after_geonameid | 0 | % | | Min Throughput | asc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | asc_sort_geonameid | 6.02 | ops/s | | Median Throughput | asc_sort_geonameid | 6.02 | ops/s | | Max Throughput | asc_sort_geonameid | 6.03 | ops/s | | 50th percentile latency | asc_sort_geonameid | 5.44507 | ms | | 90th percentile latency | asc_sort_geonameid | 5.92182 | ms | | 99th percentile latency | asc_sort_geonameid | 8.5025 | ms | | 100th percentile latency | asc_sort_geonameid | 12.0944 | ms | | 50th percentile service time | asc_sort_geonameid | 4.12126 | ms | | 90th percentile service time | asc_sort_geonameid | 4.62526 | ms | | 99th percentile service time | asc_sort_geonameid | 7.17899 | ms | | 100th percentile service time | asc_sort_geonameid | 10.7599 | ms | | error rate | asc_sort_geonameid | 0 | % | | Min Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | asc_sort_with_after_geonameid | 6.02 | ops/s | | 50th percentile latency | asc_sort_with_after_geonameid | 70.3262 | ms | | 90th percentile latency | asc_sort_with_after_geonameid | 73.0334 | ms | | 99th percentile latency | asc_sort_with_after_geonameid | 97.0722 | ms | |

100th percentile latency | asc_sort_with_after_geonameid | 99.2609 | ms | | 50th percentile service time | asc_sort_with_after_geonameid | 69.095 | ms | | 90th percentile service time | asc_sort_with_after_geonameid | 72.0477 | ms | | 99th percentile service time | asc_sort_with_after_geonameid | 95.9471 | ms | | 100th percentile service time | asc_sort_with_after_geonameid | 98.258 | ms | | error rate | asc_sort_with_after_geonameid | 0 | % |

16核32GB512GB计算型的3个数据节点实例基准性能指标

16核32GB512GB计算型的3个数据节点实例基准性能指标

使用ESRally工具对数据节点规格为16核32GB的BES集群实例（7.10.2版本）进行基准测试。

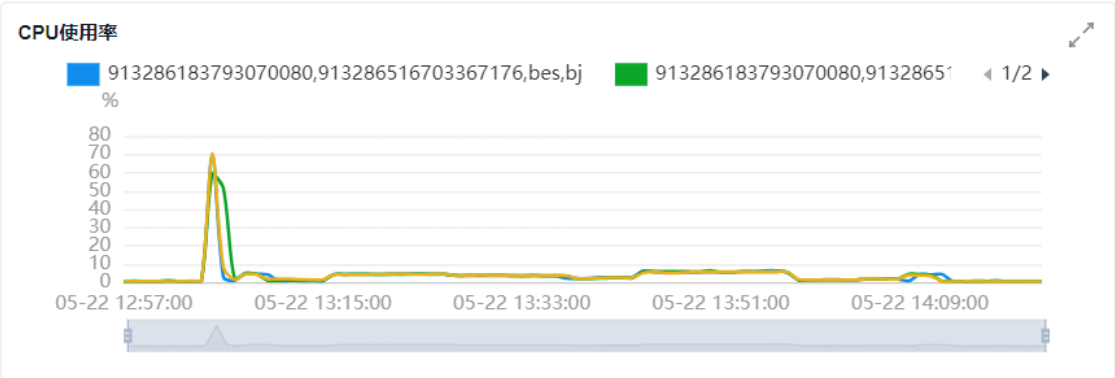
压测配置

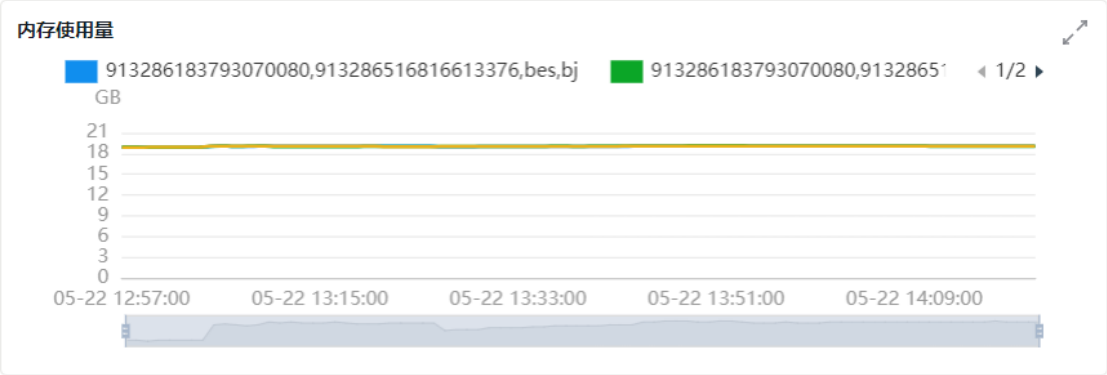
项目	说明
实例配置	版本：BES集群实例（7.10.2版本） 数据节点规格：16核32GB 数据节点存储性能级别：增强型SSD_PL1 数据单节点存储空间：512GB 数据节点数量：3个
ESRally配置	使用ESRally的默认配置tracks。
数据集	使用ESRally的预置数据集Geonames模拟测试场景，数据文档总数11396503。获取数据集，请查看ES Rally Hub获取。
分片数	主分片：6 副本数：0
bulk_size	每次批量操作提交5000个文档。
bulk_indexing_clients	同时执行批量索引操作的客户端数量为8个。
index_append warmup time	从默认的120s修改为30s，当集群配置高时，index_append在预热启动期间就完成了。

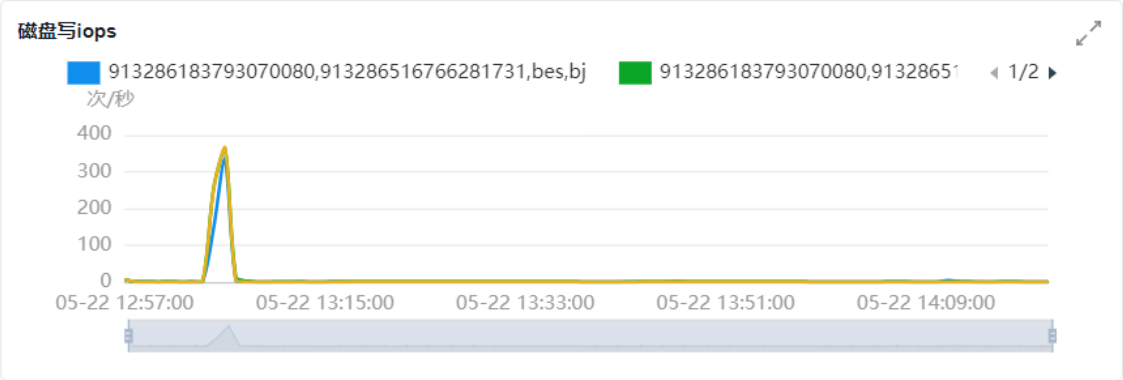
压测结果

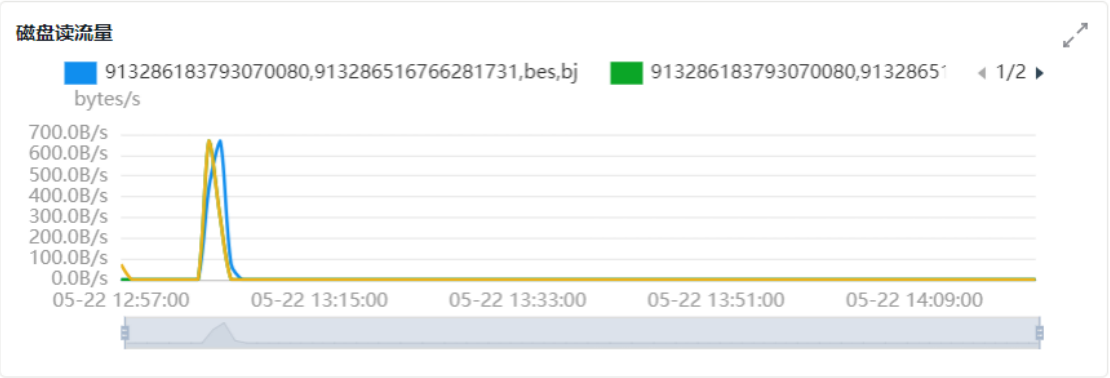
压测结果仅供参考，无法代表实际生产中写入查询情况，建议您结合业务生产数据进行压测。

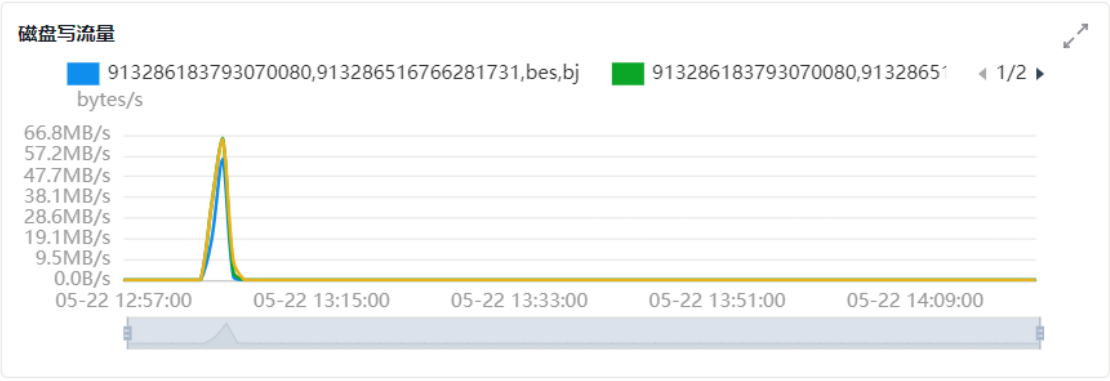
主要压测指标结果

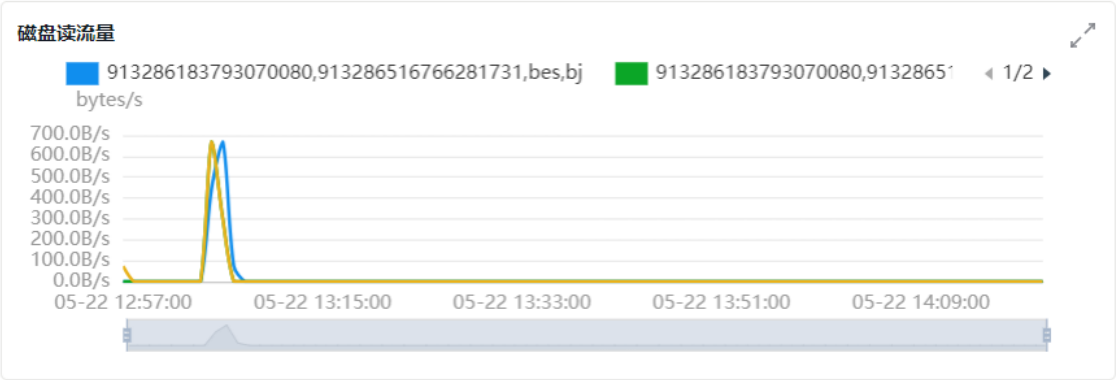


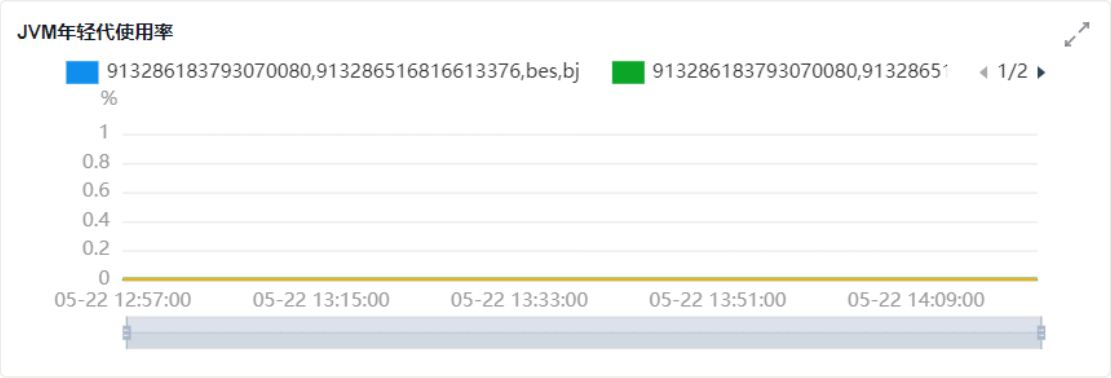

内存使用量
913286183793070080,913286516816613376,bes,bj
913286183793070080,9132865
1/2
GB
21
18
15
12
9
6
3
0
05-22 12:57:00
05-22 13:15:00
05-22 13:33:00
05-22 13:51:00
05-22 14:09:00

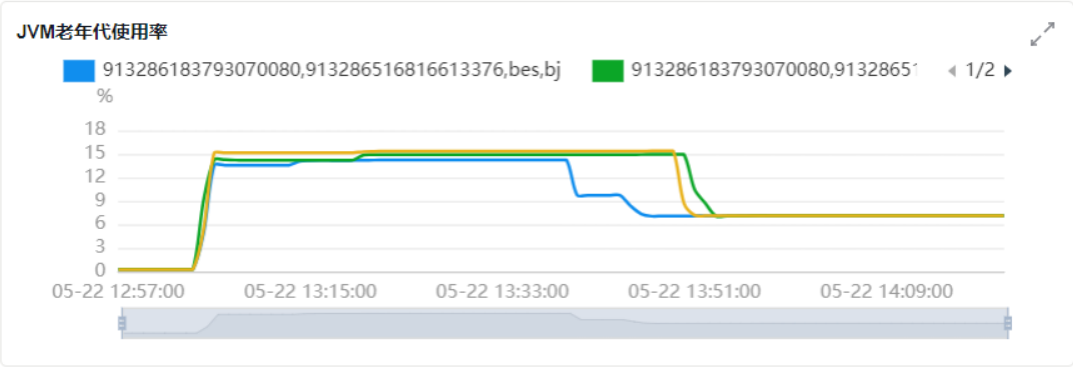

磁盘写iops
913286183793070080,913286516766281731,bes,bj
913286183793070080,9132865
1/2
次/秒
400
300
200
100
0
05-22 12:57:00
05-22 13:15:00
05-22 13:33:00
05-22 13:51:00
05-22 14:09:00

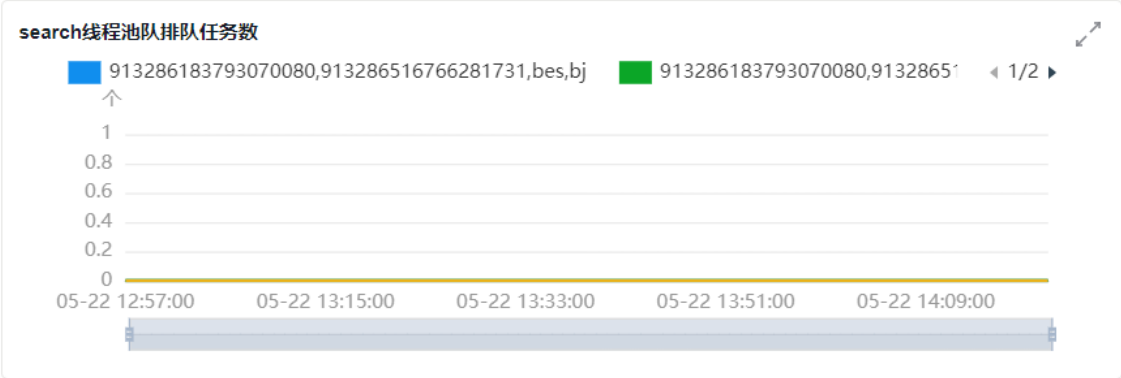

磁盘读流量
913286183793070080,913286516766281731,bes,bj
913286183793070080,9132865
1/2
bytes/s
700.0B/s
600.0B/s
500.0B/s
400.0B/s
300.0B/s
200.0B/s
100.0B/s
0.0B/s
05-22 12:57:00
05-22 13:15:00
05-22 13:33:00
05-22 13:51:00
05-22 14:09:00

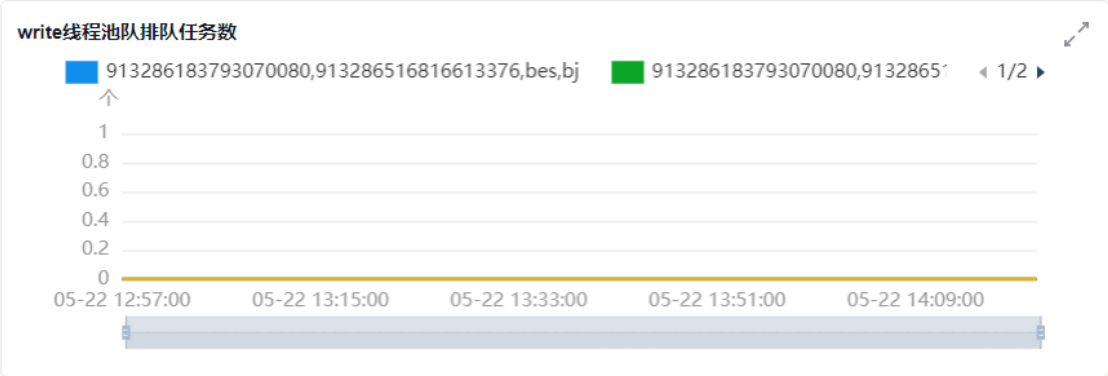

磁盘写流量
913286183793070080,913286516766281731,bes,bj
913286183793070080,9132865
1/2
bytes/s
66.8MB/s
57.2MB/s
47.7MB/s
38.1MB/s
28.6MB/s
19.1MB/s
9.5MB/s
0.0B/s
05-22 12:57:00
05-22 13:15:00
05-22 13:33:00
05-22 13:51:00
05-22 14:09:00


磁盘读流量
913286183793070080,913286516766281731,bes,bj
913286183793070080,9132865
1/2
bytes/s
700.0B/s
600.0B/s
500.0B/s
400.0B/s
300.0B/s
200.0B/s
100.0B/s
0.0B/s
05-22 12:57:00
05-22 13:15:00
05-22 13:33:00
05-22 13:51:00
05-22 14:09:00









基准压测报告 | Metric | Task | Value | Unit | |----------:|----------:|----------:|----------:| | Cumulative indexing time of primary shards | | 10.9337 | min | | Min cumulative indexing time across primary shards | | 1.83442 | min | | Median cumulative indexing time across primary shards | | 2.28088 | min | | Max cumulative indexing time across primary shards | | 2.28988 | min | | Cumulative indexing throttle time of primary shards | | 0 | min | | Min cumulative indexing throttle time across primary shards | | 0 | min | | Median cumulative indexing throttle time across primary shards | | 0 | min | | Max cumulative indexing throttle time across primary shards | | 0 | min | | Cumulative merge time of primary shards | | 3.31318 | min | | Cumulative merge count of primary shards | | 30 | | | Min cumulative merge time across primary shards | | 0.541267 | min | | Median cumulative merge time across primary shards | | 0.689833 | min | | Max cumulative merge time across primary shards | | 0.72115 | min | | Cumulative merge throttle time of primary shards | | 1.07413 | min | | Min cumulative merge throttle time across primary shards | | 0.1705 | min | | Median cumulative merge throttle time across primary shards | | 0.223183 | min | | Max cumulative merge throttle time across primary shards | | 0.252783 | min | |

Cumulative refresh time of primary shards | | 0.733233 | min | | Cumulative refresh count of primary shards | | 76 | | | Min cumulative refresh time across primary shards | | 0.120533 | min | | Median cumulative refresh time across primary shards | | 0.147433 | min | | Max cumulative refresh time across primary shards | | 0.160433 | min | | Cumulative flush time of primary shards | | 0.15545 | min | | Cumulative flush count of primary shards | | 10 | | | Min cumulative flush time across primary shards | | 0.0209167 | min | | Median cumulative flush time across primary shards | | 0.0305333 | min | | Max cumulative flush time across primary shards | | 0.0381667 | min | | Total Young Gen GC time | | 3.28 | s | | Total Young Gen GC count | | 615 | | | Total Old Gen GC time | | 0 | s | | Total Old Gen GC count | | 0 | | | Store size | | 3.10479 | GB | | Translog size | | 2.56114e-07 | GB | | Heap used for segments | | 0.622139 | MB | | Heap used for doc values | | 0.024971 | MB | | Heap used for terms | | 0.487488 | MB | | Heap used for norms | | 0.0663452 | MB | | Heap used for points | | 0 | MB | | Heap used for stored fields | | 0.043335 | MB | | Segment count | | 84 | | | Total Ingest Pipeline count | | 0 | | | Total Ingest Pipeline time | | 0 | s | | Total Ingest Pipeline failed | | 0 | | | Min Throughput | index-append | 251959 | docs/s | | Mean Throughput | index-append | 258644 | docs/s | | Median Throughput | index-append | 258336 | docs/s | | Max Throughput | index-append | 265359 | docs/s | | 50th percentile latency | index-append | 95.672 | ms | | 90th percentile latency | index-append | 260.095 | ms | | 99th percentile latency | index-append | 1130.76 | ms | | 100th percentile latency | index-append | 1534.71 | ms | | 50th percentile service time | index-append | 95.672 | ms | | 90th percentile service time | index-append | 260.095 | ms | | 99th percentile service time | index-append | 1130.76 | ms | | 100th percentile service time | index-append | 1534.71 | ms | | error rate | index-append | 0 | % | | Min Throughput | index-stats | 90.04 | ops/s | | Mean Throughput | index-stats | 90.06 | ops/s | | Median Throughput | index-stats | 90.06 | ops/s | | Max Throughput | index-stats | 90.1 | ops/s | | 50th percentile latency | index-stats | 2.66644 | ms | | 90th percentile latency | index-stats | 3.49003 | ms | | 99th percentile latency | index-stats | 3.81744 | ms | | 99.9th percentile latency | index-stats | 3.94127 | ms | | 100th percentile latency | index-stats | 4.16882 | ms | | 50th percentile service time | index-stats | 1.35649 | ms | | 90th percentile service time | index-stats | 1.48534 | ms | | 99th percentile service time | index-stats | 1.71008 | ms | | 99.9th percentile service time | index-stats | 2.00326 | ms | | 100th percentile service time | index-stats | 2.01654 | ms | | error rate | index-stats | 0 | % | | Min Throughput | node-stats | 89.92 | ops/s | | Mean Throughput | node-stats | 90.08 | ops/s | | Median Throughput | node-stats | 90.06 | ops/s | | Max Throughput | node-stats | 90.28 | ops/s | | 50th percentile latency | node-stats | 2.69308 | ms | | 90th percentile latency | node-stats | 4.06851 | ms | | 99th percentile latency | node-stats | 4.44659 | ms | | 99.9th percentile latency | node-stats | 5.12146 | ms | | 100th percentile latency | node-stats | 11.7301 | ms | | 50th percentile service time | node-stats | 1.78248 | ms | | 90th percentile service time | node-stats | 1.9183 | ms | | 99th percentile service time | node-stats | 2.66963 | ms | | 99.9th percentile service time | node-stats | 3.21213 | ms | | 100th percentile service time | node-stats | 3.3223 | ms | | error rate | node-stats | 0 | % | | Min Throughput | default | 50.03 | ops/s | | Mean Throughput | default | 50.04 | ops/s | | Median Throughput | default | 50.04 | ops/s | | Max Throughput | default | 50.07 | ops/s | | 50th percentile latency | default | 3.13588 | ms | | 90th percentile latency | default | 4.15946 | ms | | 99th percentile latency | default | 4.53467 | ms | | 99.9th percentile latency | default | 4.89849 | ms | | 100th percentile latency | default | 5.43421 | ms | | 50th percentile service time | default | 1.79154 | ms | | 90th percentile service time | default | 2.01733 | ms | | 99th percentile service time | default | 2.32979 | ms | | 99.9th percentile service time | default | 3.18136 | ms | | 100th percentile service time | default | 4.06553 | ms | | error rate | default | 0 | % | | Min Throughput | term | 100.04 | ops/s | | Mean Throughput | term | 100.06 | ops/s | | Median Throughput | term | 100.06 | ops/s | | Max Throughput | term | 100.12 | ops/s | | 50th percentile latency | term | 2.48851 | ms | | 90th percentile latency | term | 2.98729 | ms | | 99th percentile latency | term | 4.01516 | ms | | 99.9th percentile latency | term | 50.5892 | ms | | 100th percentile latency | term | 57.6077 | ms | | 50th percentile service time | term | 1.61853 | ms | | 90th percentile service time | term | 1.82886 | ms | | 99th percentile service time | term | 2.15978 | ms | | 99.9th percentile service time | term | 9.5199 | ms | | 100th percentile service time | term | 56.8846 | ms | | error rate | term | 0 | % | | Min Throughput | phrase | 110.02 | ops/s | | Mean Throughput | phrase | 110.04 | ops/s | | Median Throughput | phrase | 110.04 | ops/s | | Max Throughput | phrase | 110.07 | ops/s | | 50th percentile latency | phrase | 2.60092 | ms | | 90th percentile latency | phrase | 3.06541 | ms | | 99th percentile latency | phrase | 3.74038 | ms | | 99.9th percentile latency | phrase | 14.5582 | ms | | 100th percentile latency | phrase | 18.6079 | ms | | 50th percentile service time | phrase | 1.71315 | ms | | 90th percentile service time | phrase | 1.90559 | ms | | 99th percentile service time | phrase | 2.34588 | ms | | 99.9th percentile service time | phrase | 13.7218 | ms | | 100th percentile service time | phrase | 17.7622 | ms | | error rate | phrase | 0 | % | | Min Throughput | country_agg_uncached | 3.01 | ops/s | | Mean Throughput | country_agg_uncached | 3.01 | ops/s | | Median Throughput | country_agg_uncached | 3.01 | ops/s | | Max Throughput | country_agg_uncached | 3.01 | ops/s | | 50th percentile latency | country_agg_uncached | 130.727 | ms | | 90th

percentile latency | country_agg_uncached | 150.799 | ms | | 99th percentile latency | country_agg_uncached | 160.816 | ms | | 100th percentile latency | country_agg_uncached | 178.802 | ms | | 50th percentile service time | country_agg_uncached | 129.429 | ms | | 90th percentile service time | country_agg_uncached | 149.228 | ms | | 99th percentile service time | country_agg_uncached | 159.495 | ms | | 100th percentile service time | country_agg_uncached | 177.503 | ms | | error rate | country_agg_uncached | 0 | % | | Min Throughput | country_agg_cached | 98.89 | ops/s | | Mean Throughput | country_agg_cached | 99.18 | ops/s | | Median Throughput | country_agg_cached | 99.21 | ops/s | | Max Throughput | country_agg_cached | 99.39 | ops/s | | 50th percentile latency | country_agg_cached | 2.49958 | ms | | 90th percentile latency | country_agg_cached | 3.8216 | ms | | 99th percentile latency | country_agg_cached | 4.71775 | ms | | 99.9th percentile latency | country_agg_cached | 5.29509 | ms | | 100th percentile latency | country_agg_cached | 5.70463 | ms | | 50th percentile service time | country_agg_cached | 1.57903 | ms | | 90th percentile service time | country_agg_cached | 2.80845 | ms | | 99th percentile service time | country_agg_cached | 3.3994 | ms | | 99.9th percentile service time | country_agg_cached | 4.44264 | ms | | 100th percentile service time | country_agg_cached | 4.48943 | ms | | error rate | country_agg_cached | 0 | % | | Min Throughput | scroll | 20.05 | pages/s | | Mean Throughput | scroll | 20.07 | pages/s | | Median Throughput | scroll | 20.06 | pages/s | | Max Throughput | scroll | 20.08 | pages/s | | 50th percentile latency | scroll | 204.928 | ms | | 90th percentile latency | scroll | 209.259 | ms | | 99th percentile latency | scroll | 218.198 | ms | | 100th percentile latency | scroll | 218.331 | ms | | 50th percentile service time | scroll | 202.573 | ms | | 90th percentile service time | scroll | 206.679 | ms | | 99th percentile service time | scroll | 214.707 | ms | | 100th percentile service time | scroll | 216.775 | ms | | error rate | scroll | 0 | % | | Min Throughput | expression | 1.5 | ops/s | | Mean Throughput | expression | 1.5 | ops/s | | Median Throughput | expression | 1.5 | ops/s | | Max Throughput | expression | 1.5 | ops/s | | 50th percentile latency | expression | 243.385 | ms | | 90th percentile latency | expression | 263.782 | ms | | 99th percentile latency | expression | 268.557 | ms | | 100th percentile latency | expression | 271.662 | ms | | 50th percentile service time | expression | 241.65 | ms | | 90th percentile service time | expression | 262.089 | ms | | 99th percentile service time | expression | 267.104 | ms | | 100th percentile service time | expression | 269.75 | ms | | error rate | expression | 0 | % | | Min Throughput | painless_static | 1.4 | ops/s | | Mean Throughput | painless_static | 1.4 | ops/s | | Median Throughput | painless_static | 1.4 | ops/s | | Max Throughput | painless_static | 1.4 | ops/s | | 50th percentile latency | painless_static | 264.972 | ms | | 90th percentile latency | painless_static | 282.138 | ms | | 99th percentile latency | painless_static | 294.578 | ms | | 100th percentile latency | painless_static | 298.941 | ms | | 50th percentile service time | painless_static | 263.489 | ms | | 90th percentile service time | painless_static | 279.358 | ms | | 99th percentile service time | painless_static | 292.5 | ms | | 100th percentile service time | painless_static | 297.875 | ms | | error rate | painless_static | 0 | % | | Min Throughput | painless_dynamic | 1.4 | ops/s | | Mean Throughput | painless_dynamic | 1.4 | ops/s | | Median Throughput | painless_dynamic | 1.4 | ops/s | | Max Throughput | painless_dynamic | 1.4 | ops/s | | 50th percentile latency | painless_dynamic | 267.218 | ms | | 90th percentile latency | painless_dynamic | 284.529 | ms | | 99th percentile latency | painless_dynamic | 294.932 | ms | | 100th percentile latency | painless_dynamic | 318.517 | ms | | 50th percentile service time | painless_dynamic | 265.777 | ms | | 90th percentile service time | painless_dynamic | 283.254 | ms | | 99th percentile service time | painless_dynamic | 293.627 | ms | | 100th percentile service time | painless_dynamic | 317.152 | ms | | error rate | painless_dynamic | 0 | % | | Min Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_function_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_function_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_function_score | 307.989 | ms | | 90th percentile latency | decay_geo_gauss_function_score | 315.452 | ms | | 99th percentile latency | decay_geo_gauss_function_score | 320.052 | ms | | 100th percentile latency | decay_geo_gauss_function_score | 340.011 | ms | | 50th percentile service time | decay_geo_gauss_function_score | 305.711 | ms | | 90th percentile service time | decay_geo_gauss_function_score | 312.513 | ms | | 99th percentile service time | decay_geo_gauss_function_score | 317.589 | ms | | 100th percentile service time | decay_geo_gauss_function_score | 336.846 | ms | | error rate | decay_geo_gauss_function_score | 0 | % | | Min Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Mean Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Median Throughput | decay_geo_gauss_script_score | 1 | ops/s | | Max Throughput | decay_geo_gauss_script_score | 1 | ops/s | | 50th percentile latency | decay_geo_gauss_script_score | 308.468 | ms | | 90th percentile latency | decay_geo_gauss_script_score | 312.12 | ms | | 99th percentile latency | decay_geo_gauss_script_score | 319.508 | ms | | 100th percentile latency | decay_geo_gauss_script_score | 319.726 | ms | | 50th percentile service time | decay_geo_gauss_script_score | 306.34 | ms | | 90th percentile service time | decay_geo_gauss_script_score | 309.919 | ms | | 99th percentile service time | decay_geo_gauss_script_score | 317.574 | ms | | 100th percentile service time |

decay_geo_gauss_script_score | 318.386 | ms | | error rate | decay_geo_gauss_script_score | 0 | % | | Min Throughput | field_value_function_score | 1.5 | ops/s | | Mean Throughput | field_value_function_score | 1.51 | ops/s | | Median Throughput | field_value_function_score | 1.51 | ops/s | | Max Throughput | field_value_function_score | 1.51 | ops/s | | 50th percentile latency | field_value_function_score | 100.511 | ms | | 90th percentile latency | field_value_function_score | 103.967 | ms | | 99th percentile latency | field_value_function_score | 121.622 | ms | | 100th percentile latency | field_value_function_score | 122.2 | ms | | 50th percentile service time | field_value_function_score | 98.7087 | ms | | 90th percentile service time | field_value_function_score | 101.844 | ms | | 99th percentile service time | field_value_function_score | 119.336 | ms | | 100th percentile service time | field_value_function_score | 120.202 | ms | | error rate | field_value_function_score | 0 | % | | Min Throughput | field_value_script_score | 1.5 | ops/s | | Mean Throughput | field_value_script_score | 1.5 | ops/s | | Median Throughput | field_value_script_score | 1.5 | ops/s | | Max Throughput | field_value_script_score | 1.51 | ops/s | | 50th percentile latency | field_value_script_score | 137.987 | ms | | 90th percentile latency | field_value_script_score | 140.154 | ms | | 99th percentile latency | field_value_script_score | 152.326 | ms | | 100th percentile latency | field_value_script_score | 161.77 | ms | | 50th percentile service time | field_value_script_score | 136.351 | ms | | 90th percentile service time | field_value_script_score | 138.457 | ms | | 99th percentile service time | field_value_script_score | 150.885 | ms | | 100th percentile service time | field_value_script_score | 159.668 | ms | | error rate | field_value_script_score | 0 | % | | Min Throughput | large_terms | 1.1 | ops/s | | Mean Throughput | large_terms | 1.1 | ops/s | | Median Throughput | large_terms | 1.1 | ops/s | | Max Throughput | large_terms | 1.1 | ops/s | | 50th percentile latency | large_terms | 545.665 | ms | | 90th percentile latency | large_terms | 549.487 | ms | | 99th percentile latency | large_terms | 557.167 | ms | | 100th percentile latency | large_terms | 560.106 | ms | | 50th percentile service time | large_terms | 538.917 | ms | | 90th percentile service time | large_terms | 542.709 | ms | | 99th percentile service time | large_terms | 550.557 | ms | | 100th percentile service time | large_terms | 553.525 | ms | | error rate | large_terms | 0 | % | | Min Throughput | large_filtered_terms | 1.1 | ops/s | | Mean Throughput | large_filtered_terms | 1.1 | ops/s | | Median Throughput | large_filtered_terms | 1.1 | ops/s | | Max Throughput | large_filtered_terms | 1.1 | ops/s | | 50th percentile latency | large_filtered_terms | 548.928 | ms | | 90th percentile latency | large_filtered_terms | 554.665 | ms | | 99th percentile latency | large_filtered_terms | 562.814 | ms | | 100th percentile latency | large_filtered_terms | 572.404 | ms | | 50th percentile service time | large_filtered_terms | 542.257 | ms | | 90th percentile service time | large_filtered_terms | 548.182 | ms | | 99th percentile service time | large_filtered_terms | 555.781 | ms | | 100th percentile service time | large_filtered_terms | 565.601 | ms | | error rate | large_filtered_terms | 0 | % | | Min Throughput | large_prohibited_terms | 1.1 | ops/s | | Mean Throughput | large_prohibited_terms | 1.1 | ops/s | | Median Throughput | large_prohibited_terms | 1.1 | ops/s | | Max Throughput | large_prohibited_terms | 1.1 | ops/s | | 50th percentile latency | large_prohibited_terms | 537.832 | ms | | 90th percentile latency | large_prohibited_terms | 542.141 | ms | | 99th percentile latency | large_prohibited_terms | 544.209 | ms | | 100th percentile latency | large_prohibited_terms | 550.779 | ms | | 50th percentile service time | large_prohibited_terms | 531.23 | ms | | 90th percentile service time | large_prohibited_terms | 535.368 | ms | | 99th percentile service time | large_prohibited_terms | 537.588 | ms | | 100th percentile service time | large_prohibited_terms | 543.811 | ms | | error rate | large_prohibited_terms | 0 | % | | Min Throughput | desc_sort_population | 1.5 | ops/s | | Mean Throughput | desc_sort_population | 1.51 | ops/s | | Median Throughput | desc_sort_population | 1.51 | ops/s | | Max Throughput | desc_sort_population | 1.51 | ops/s | | 50th percentile latency | desc_sort_population | 60.2142 | ms | | 90th percentile latency | desc_sort_population | 62.2713 | ms | | 99th percentile latency | desc_sort_population | 82.2277 | ms | | 100th percentile latency | desc_sort_population | 83.3576 | ms | | 50th percentile service time | desc_sort_population | 58.5499 | ms | | 90th percentile service time | desc_sort_population | 60.7134 | ms | | 99th percentile service time | desc_sort_population | 80.5677 | ms | | 100th percentile service time | desc_sort_population | 81.5889 | ms | | error rate | desc_sort_population | 0 | % | | Min Throughput | asc_sort_population | 1.5 | ops/s | | Mean Throughput | asc_sort_population | 1.51 | ops/s | | Median Throughput | asc_sort_population | 1.51 | ops/s | | Max Throughput | asc_sort_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_population | 61.8712 | ms | | 90th percentile latency | asc_sort_population | 64.1878 | ms | | 99th percentile latency | asc_sort_population | 84.4655 | ms | | 100th percentile latency | asc_sort_population | 84.7261 | ms | | 50th percentile service time | asc_sort_population | 60.1245 | ms | | 90th percentile service time | asc_sort_population | 62.2853 | ms | | 99th percentile service time | asc_sort_population | 82.7414 | ms | | 100th percentile service time | asc_sort_population | 82.7958 | ms | | error rate | asc_sort_population | 0 | % | | Min Throughput | asc_sort_with_after_population | 1.5 | ops/s | | Mean Throughput | asc_sort_with_after_population | 1.51 | ops/s | | Median Throughput | asc_sort_with_after_population | 1.51 | ops/s | | Max Throughput | asc_sort_with_after_population | 1.51 | ops/s | | 50th percentile latency | asc_sort_with_after_population |

74.7079 | ms | | 90th percentile latency | asc_sort_with_after_population | 83.7405 | ms | | 99th percentile latency | asc_sort_with_after_population | 98.7648 | ms | | 100th percentile latency | asc_sort_with_after_population | 100.996 | ms | | 50th percentile service time | asc_sort_with_after_population | 73.0636 | ms | | 90th percentile service time | asc_sort_with_after_population | 82.1577 | ms | | 99th percentile service time | asc_sort_with_after_population | 97.4383 | ms | | 100th percentile service time | asc_sort_with_after_population | 99.6243 | ms | | error rate | asc_sort_with_after_population | 0 | % | | Min Throughput | desc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | desc_sort_geonameid | 6.02 | ops/s | | Median Throughput | desc_sort_geonameid | 6.02 | ops/s | | Max Throughput | desc_sort_geonameid | 6.03 | ops/s | | 50th percentile latency | desc_sort_geonameid | 5.55858 | ms | | 90th percentile latency | desc_sort_geonameid | 6.32429 | ms | | 99th percentile latency | desc_sort_geonameid | 7.03976 | ms | | 100th percentile latency | desc_sort_geonameid | 7.96389 | ms | | 50th percentile service time | desc_sort_geonameid | 4.28917 | ms | | 90th percentile service time | desc_sort_geonameid | 4.91764 | ms | | 99th percentile service time | desc_sort_geonameid | 5.40451 | ms | | 100th percentile service time | desc_sort_geonameid | 7.0389 | ms | | error rate | desc_sort_geonameid | 0 | % | | Min Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | desc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | desc_sort_with_after_geonameid | 6.02 | ops/s | | 50th percentile latency | desc_sort_with_after_geonameid | 70.4754 | ms | | 90th percentile latency | desc_sort_with_after_geonameid | 76.05 | ms | | 99th percentile latency | desc_sort_with_after_geonameid | 95.7782 | ms | | 100th percentile latency | desc_sort_with_after_geonameid | 95.9687 | ms | | 50th percentile service time | desc_sort_with_after_geonameid | 69.3362 | ms | | 90th percentile service time | desc_sort_with_after_geonameid | 74.731 | ms | | 99th percentile service time | desc_sort_with_after_geonameid | 94.3695 | ms | | 100th percentile service time | desc_sort_with_after_geonameid | 94.4154 | ms | | error rate | desc_sort_with_after_geonameid | 0 | % | | Min Throughput | asc_sort_geonameid | 6.02 | ops/s | | Mean Throughput | asc_sort_geonameid | 6.02 | ops/s | | Median Throughput | asc_sort_geonameid | 6.02 | ops/s | | Max Throughput | asc_sort_geonameid | 6.03 | ops/s | | 50th percentile latency | asc_sort_geonameid | 4.74601 | ms | | 90th percentile latency | asc_sort_geonameid | 5.15122 | ms | | 99th percentile latency | asc_sort_geonameid | 5.76906 | ms | | 100th percentile latency | asc_sort_geonameid | 5.91034 | ms | | 50th percentile service time | asc_sort_geonameid | 3.33382 | ms | | 90th percentile service time | asc_sort_geonameid | 3.90223 | ms | | 99th percentile service time | asc_sort_geonameid | 4.32505 | ms | | 100th percentile service time | asc_sort_geonameid | 4.37321 | ms | | error rate | asc_sort_geonameid | 0 | % | | Min Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Mean Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Median Throughput | asc_sort_with_after_geonameid | 6.01 | ops/s | | Max Throughput | asc_sort_with_after_geonameid | 6.02 | ops/s | | 50th percentile latency | asc_sort_with_after_geonameid | 64.7787 | ms | | 90th percentile latency | asc_sort_with_after_geonameid | 83.7871 | ms | | 99th percentile latency | asc_sort_with_after_geonameid | 88.3394 | ms | | 100th percentile latency | asc_sort_with_after_geonameid | 88.6091 | ms | | 50th percentile service time | asc_sort_with_after_geonameid | 63.1986 | ms | | 90th percentile service time | asc_sort_with_after_geonameid | 81.8991 | ms | | 99th percentile service time | asc_sort_with_after_geonameid | 86.4036 | ms | | 100th percentile service time | asc_sort_with_after_geonameid | 86.629 | ms | | error rate | asc_sort_with_after_geonameid | 0 | % |

产品定价

计费说明

计费项目

百度智能云Elasticsearch仅收取托管Elasticsearch集群的资源费用，其余云上的配套功能例如监控等不收取任何额外费用，Elasticsearch收费项目如下：

计费项	说明
集群节点	- 以节点规格的形式提供，包括CPU、内存和磁盘存储。 - 支持预付费和后付费。

计费模式

Elasticsearch支持预付费（包年包月计费）和后付费（按量计费）两种计费方式，详细的计费规则如下：

计费方式	说明	注意事项
预付费-包年包月计费	- 即包年、包月付费，在创建实例时需要支付费用。 - 适合业务稳定的长期需求，费用比后付费更低，且购买时间越长，折扣越多。	- 预付费实例不支持退订，如需释放请提交工单处理。
后付费－按量计费	- 按分钟计费，根据实例配在北京时间整点扣费并生成账单。 - 出账单时间是当前计费周期结束后1小时内。 - 适合业务瞬间有较大变化的场景，资源使用完即可立即释放，降低成本。	- 后付费实例可随时被释放。 - 购买前需保证账户无欠款，且账户余额和可用代金券总和≥100元。

支付方式

购买Elasticsearch时，您可以选择以下支付方式：

- 百度智能云账户余额，可在 管理控制台 - 财务总览 处查看当前余额。
- 在线支付。
- 代金券支付，请确保代金券的状态为使用中且订单类型为通用。 您可以登录在 管理控制台 - 财务中心 - 代金券 查看当前拥有的代金券列表

计费变更

为了满足用户业务需求，百度智能云Elasticsearch支持计费模式切换服务，您可灵活切换按需计费（后付费）与包年包月计费（预付费）两种计费方式。

预付费转为后付费

计费变更说明

1. 预付费订单需要在到期的24小时之前进行计费方式切换，计费方式变更后将于当前订单到期后生效，到期时间以财务订单为准。
2. 到期前24小时内不允许计费变更。
3. 预付费服务切换为后付费计费，切换后的计费方式将于服务到期后生效。
4. 本功能仅支持未开通自动续费的用户。

> 注：进行预付费转后付费操作时，需要账号余额大于100元方可转换成功，若不足则会出现订单失败作废的情况，则计费方式切换失败。

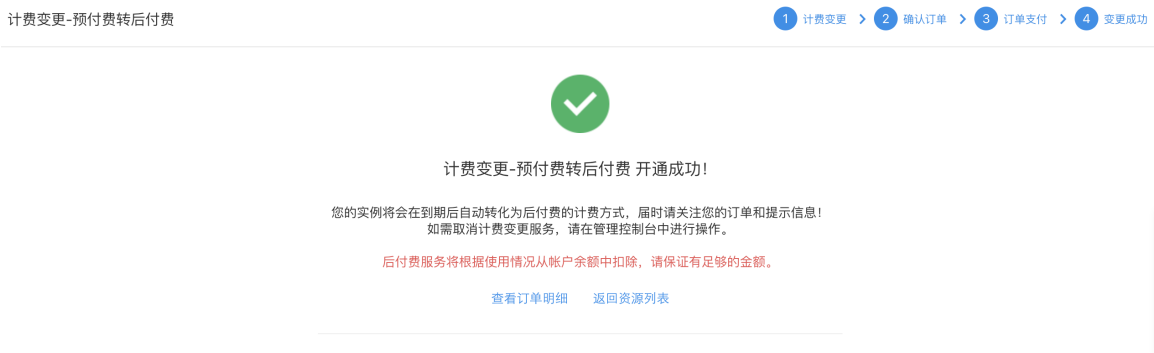
操作步骤

1. 登录控制台，进入“产品服务->Elasticsearch”，进入集群列表页面，点击需要变更计费方式的预付费Elasticsearch集群右侧的操作栏点击“计费变更”。

2. 进入“计费变更”页面，确认预付费转后付费的节点和财务信息。如果没有问题则点击下一步。

3. 确认订单信息，如果没有问题，则点击“下一步”。

4. 预付费转后付费计费变更完成。



5. 完成预付费转后付费的集群实例，目前的支付方式仍显示预付费，待预付费到期后，支付方式自动变更为后付费，这类集群的支付方式旁白你会有一个标志提醒。

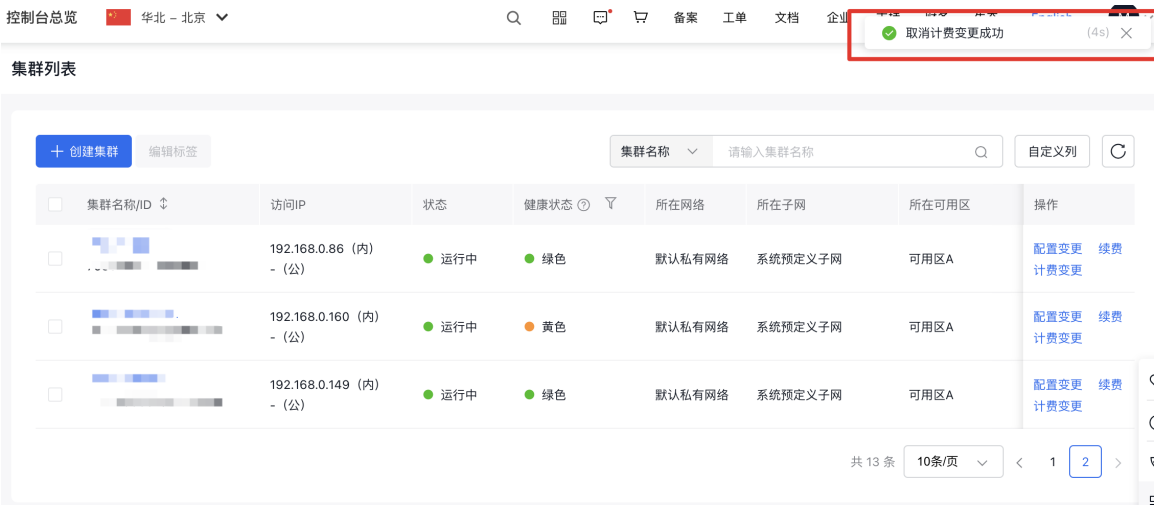
集群列表

6. 在计费变更生效之前，用户随时可以取消之前的计费变更。在对应集群的操作列中选择“取消计费变更”。

集群列表

7. 取消计费变更会弹出确认框，如果确认取消变更，点击“确认”。

8. 确认取消变更成功之后，可以在需要的时候再进行计费变更，但需要注意计费变更要在集群到期24小时前操作，24小时内到期的集群不允许计费变更。



后付费转为预付费

计费变更说明

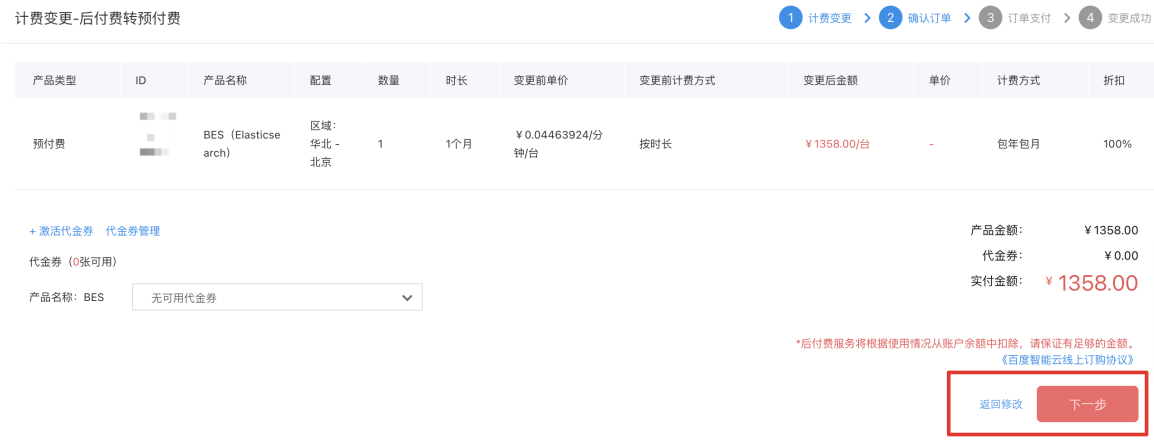
1. 后付费变更预付费随时可以操作；
2. 变更之后，预付费计费立即生效。

操作步骤

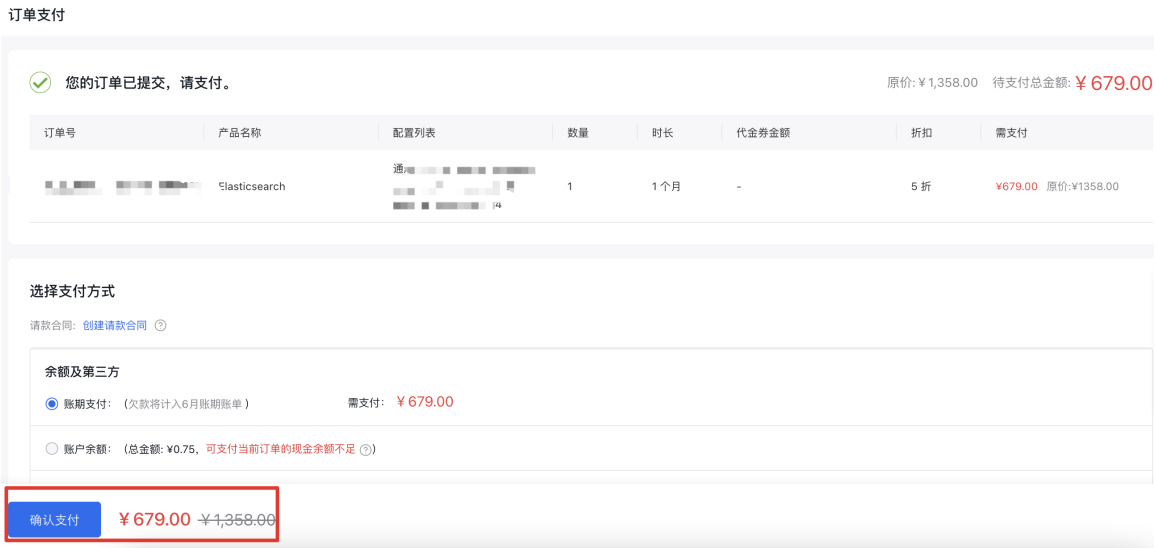
1. 登录控制台，进入“产品服务->Elasticsearch”，进入集群列表页面，点击需要变更计费方式的后付费的Elasticsearch集群右侧操作列表中的计费变更。

2. 在计费变更页面选择预付费的时长，确认之后点击“下一步”。

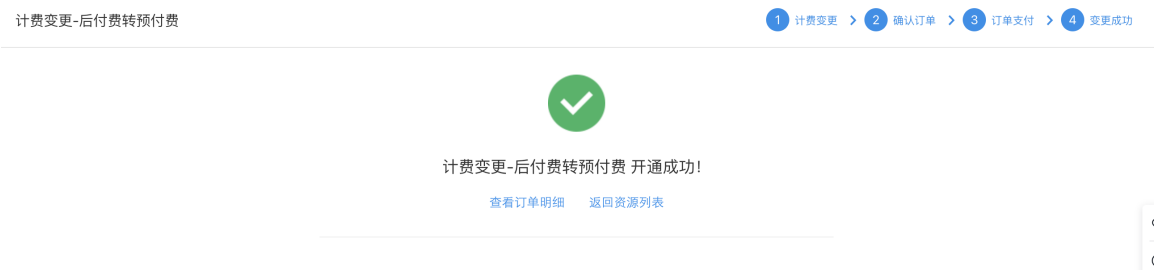
3. 计费变更的预付费支付过程和正常购买集群的过程一致，点击“下一步”进入订单支付页面。



4. 点击“确认支付”。



5. 支付完成之后，计费变更完成。



6. 回到集群管理列表，可以看到之前的后付费集群已经变更为预付费集群，由于后付费转预付费立即生效，集群没有特殊的提示。

预付费

本文档介绍产品预付费的定价、计费规则及到期提醒等内容。

预付费情况下，您可以按月预支付百度云Elasticsearch的使用费用，百度云Elasticsearch会根据您选择的集群节点配置和个数，计算出按月付费的套餐价格。通常，相同时长的预付费价格要远低于按需付费的价格。

> 说明：购买前需保证账户无欠款。

预付费计费公式

百度云Elasticsearch从2021年2月1日开始所有region新购集群执行计算节点和存储磁盘分别计费的价格模式：

以计算节点规格和单节点存储空间为单位，对集群进行计费。

费用 = （节点单价 × 节点个数 + 磁盘单价 × 容量） × 使用时长 × 折扣

对于旧的集群节点续费或者扩容仍然执行以前的节点套餐价格模式：

费用 = 节点套餐单价 × 节点个数 × 使用时长 × 折扣

> 预付费折扣：1年8.3折，2年7折，3年5折

节点价格（新）

华北-北京，华北-保定，华东-苏州，华东-上海，华南-广州

节点类型	规格名称	规格比例	CPU（核）	内存（GB）	单价（元/月/节点）
通用型	bes.g3.c2m8	1:4	2	8	330
-	bes.g3.c4m16	1:4	4	16	620
-	bes.g3.c8m32	1:4	8	32	1201
-	bes.g3.c16m64	1:4	16	64	2363
-	bes.g3.c32m128	1:4	32	128	4686
计算型	bes.c3.c2m4	1:2	2	4	233
-	bes.c3.c4m8	1:2	4	8	427
-	bes.c3.c8m16	1:2	8	16	814
-	bes.c3.c16m32	1:2	16	32	1588
-	bes.c3.c32m64	1:2	32	64	3137
密集计算型	bes.ic3.c4m4	1:1	4	4	330
-	bes.ic3.c8m8	1:1	8	8	620
-	bes.ic3.c12m12	1:1	12	12	912
-	bes.ic3.c16m16	1:1	16	16	1201
内存型	bes.m3.c2m16	1:8	2	16	524
-	bes.m3.c4m32	1:8	4	32	1008
-	bes.m3.c8m64	1:8	8	64	1976

金融华中-武汉

节点类型	规格名称	规格比例	CPU（核）	内存（GB）	单价（元/月/节点）
通用型	bes.g3.c2m8	1:4	2	8	352.9
-	bes.g3.c4m16	1:4	4	16	675.38
-	bes.g3.c8m32	1:4	8	32	1320.33
-	bes.g3.c16m64	1:4	16	64	2610.24
-	bes.g3.c32m128	1:4	32	128	5190.07
计算型	bes.c3.c2m4	1:2	2	4	268.87
-	bes.c3.c4m8	1:2	4	8	507.31
-	bes.c3.c8m16	1:2	8	16	984.2
-	bes.c3.c16m32	1:2	16	32	1937.99
-	bes.c3.c32m64	1:2	32	64	3845.56
密集计算型	bes.ic3.c4m4	1:1	4	4	423.28
-	bes.ic3.c8m8	1:1	8	8	816.14
-	bes.ic3.c12m12	1:1	12	12	1368.9
-	bes.ic3.c16m16	1:1	16	16	1601.86
内存型	bes.m3.c2m16	1:8	2	16	520.96
-	bes.m3.c4m32	1:8	4	32	1011.5
-	bes.m3.c8m64	1:8	8	64	1992.59

中国香港

节点类型	规格名称	规格比例	CPU（核）	内存（GB）	单价（元/月/节点）
通用型	bes.g3.c2m8	1:4	2	8	476.64
-	bes.g3.c4m16	1:4	4	16	923.04
-	bes.g3.c8m32	1:4	8	32	1815.84
-	bes.g3.c16m64	1:4	16	64	3601.44
-	bes.g3.c32m128	1:4	32	128	7172.64
计算型	bes.c3.c2m4	1:2	2	4	327.84
-	bes.c3.c4m8	1:2	4	8	625.44
-	bes.c3.c8m16	1:2	8	16	1220.64
-	bes.c3.c16m32	1:2	16	32	2411.04
-	bes.c3.c32m64	1:2	32	64	4791.84
密集计算型	bes.ic3.c4m4	1:1	4	4	476.64
-	bes.ic3.c8m8	1:1	8	8	923.04
-	bes.ic3.c12m12	1:1	12	12	1369.44
-	bes.ic3.c16m16	1:1	16	16	1815.84
内存型	bes.m3.c2m16	1:8	2	16	774.24
-	bes.m3.c4m32	1:8	4	32	1518.24
-	bes.m3.c8m64	1:8	8	64	3006.24

新加坡

节点类型	规格名称	规格比例	CPU（核）	内存（GB）	单价（元/月/节点）
通用型	bes.g3.c2m8	1:4	2	8	496.32
-	bes.g3.c4m16	1:4	4	16	960.96
-	bes.g3.c8m32	1:4	8	32	1890.24
-	bes.g3.c16m64	1:4	16	64	3748.8
-	bes.g3.c32m128	1:4	32	128	7465.92
计算型	bes.c3.c2m4	1:2	2	4	341.44
-	bes.c3.c4m8	1:2	4	8	651.2
-	bes.c3.c8m16	1:2	8	16	1270.72
-	bes.c3.c16m32	1:2	16	32	2509.76
-	bes.c3.c32m64	1:2	32	64	4987.84
密集计算型	bes.ic3.c4m4	1:1	4	4	496.32
-	bes.ic3.c8m8	1:1	8	8	960.96
-	bes.ic3.c12m12	1:1	12	12	1425.6
-	bes.ic3.c16m16	1:1	16	16	1890.24
内存型	bes.m3.c2m16	1:8	2	16	806.08
-	bes.m3.c4m32	1:8	4	32	1580.48
-	bes.m3.c8m64	1:8	8	64	3129.28

磁盘价格

华北-北京，华北-保定，华东-苏州，华东-上海，华南-广州

磁盘类型	计费项	元/GB/月	元/GB/年（8.3折）	元/GB/2年（7折）	元/GB/3年（5折）
SSD云磁盘	存储空间	0.9	8.964	15.12	16.2
高性能云磁盘	存储空间	0.35	3.486	5.88	6.3

金融华中-武汉

磁盘类型	计费项	元/GB/月	元/GB/年（8.3折）	元/GB/2年（7折）	元/GB/3年（5折）
SSD云磁盘	存储空间	1.17	11.6532	19.656	21.06
高性能云磁盘	存储空间	0.455	4.5318	7.644	8.19

中国香港

磁盘类型	计费项	元/GB/月	元/GB/年（8.3折）	元/GB/2年（7折）	元/GB/3年（5折）
SSD云磁盘	存储空间	1.26	12.5496	21.168	22.68
高性能云磁盘	存储空间	0.49	4.8804	8.232	8.82

新加坡

磁盘类型	计费项	元/GB/月	元/GB/年（8.3折）	元/GB/2年（7折）	元/GB/3年（5折）
SSD云磁盘	存储空间	1.44	14.34	24.19	25.92
高性能云磁盘	存储空间	0.56	5.58	9.41	10.08

节点价格（旧）

华北-北京，华北-保定，华东-苏州，华南-广州

套餐名称	节点类型	CPU（核）	内存（GB）	硬盘	单价（元/月/节点）
存储1型	Elasticsearch Node	4	16	500G SSD云磁盘	1200
存储2型	Elasticsearch Node	8	32	1T SSD云磁盘	2300
存储3型	Elasticsearch Node	16	64	2T SSD云磁盘	4500
计算1型	Elasticsearch Node	4	32	100G SSD云磁盘	1100
计算2型	Elasticsearch Node	16	64	200G SSD云磁盘	2500
计算3型	Elasticsearch Node	32	128	500G SSD云磁盘	5100

金融华中-武汉

套餐名称	节点类型	CPU（核）	内存（GB）	硬盘	单价（元/月/节点）
存储1型	Elasticsearch Node	4	16	500G SSD云磁盘	1560
存储2型	Elasticsearch Node	8	32	1T SSD云磁盘	2990
存储3型	Elasticsearch Node	16	64	2T SSD云磁盘	5850
计算1型	Elasticsearch Node	4	32	100G SSD云磁盘	1430
计算2型	Elasticsearch Node	16	64	200G SSD云磁盘	3250
计算3型	Elasticsearch Node	32	128	500G SSD云磁盘	6630

中国香港

套餐名称	节点类型	CPU（核）	内存（GB）	硬盘	单价（元/月/节点）
存储1型	Elasticsearch Node	4	16	500G SSD云磁盘	1580
存储2型	Elasticsearch Node	8	32	1T SSD云磁盘	3000
存储3型	Elasticsearch Node	16	64	2T SSD云磁盘	5800
计算1型	Elasticsearch Node	4	32	100G SSD云磁盘	1500
计算2型	Elasticsearch Node	16	64	200G SSD云磁盘	3500
计算3型	Elasticsearch Node	32	128	500G SSD云磁盘	7200

新加坡

套餐名称	节点类型	CPU（核）	内存（GB）	硬盘	单价（元/月/节点）
存储1型	Elasticsearch Node	4	16	500G SSD云磁盘	1508
存储2型	Elasticsearch Node	8	32	1T SSD云磁盘	3027
存储3型	Elasticsearch Node	16	64	2T SSD云磁盘	6033
计算1型	Elasticsearch Node	4	32	100G SSD云磁盘	1496
计算2型	Elasticsearch Node	16	64	200G SSD云磁盘	3505
计算3型	Elasticsearch Node	32	128	500G SSD云磁盘	7126

> 说明：针对不同类型、地域的用户所能选择的配置可能有所差异，以Console显示实际所能购买的配置为准。

到期提醒

- Elasticsearch服务到期前7天，系统会给您发送即将到期提醒通知。
- 到期后立即停服，系统会发送欠费停服通知。停服7天后，如果还没有续买服务，系统会自动删除服务并释放资源。

后付费

后付费即按使用量付费，在此模式下，您可以按使用时长支付Elasticsearch的使用费用，可随时停止服务释放集群。

> 购买集群前需保证账户无欠款，且保证账户余额和可用代金券总和大于或等于100元。

后付费计费公式

百度云Elasticsearch从2021年2月1日开始所有region新购集群执行计算节点和存储磁盘分别计费的价格模式：

以计算节点规格和单节点存储空间为单位，对集群进行计费。

费用 = （节点单价 × 节点个数 + 磁盘单价 × 容量） × 使用时长

旧的集群节点仍然执行以前的节点套餐价格模式：：

费用 = 节点套餐单价 × 节点个数 × 使用时长

节点价格（新）

华北-北京，华北-保定，华东-苏州，华东-上海，华南-广州

节点类型	规格名称	规格比例	CPU（核）	内存（GB）	单价（元/分钟/节点）
通用型	bes.g3.c2m8	1:4	2	8	0.01127931
-	bes.g3.c4m16	1:4	4	16	0.02197531
-	bes.g3.c8m32	1:4	8	32	0.04336731
-	bes.g3.c16m64	1:4	16	64	0.08615131
-	bes.g3.c32m128	1:4	32	128	0.17171931
计算型	bes.c3.c2m4	1:2	2	4	0.00768131
-	bes.c3.c4m8	1:2	4	8	0.01477931
-	bes.c3.c8m16	1:2	8	16	0.02897531
-	bes.c3.c16m32	1:2	16	32	0.05736731
-	bes.c3.c32m64	1:2	32	64	0.11415131
密集计算型	bes.ic3.c4m4	1:1	4	4	0.01118131
-	bes.ic3.c8m8	1:1	8	8	0.02177931
-	bes.ic3.c12m12	1:1	12	12	0.03237731
-	bes.ic3.c16m16	1:1	16	16	0.04297531
内存型	bes.m3.c2m16	1:8	2	16	0.01847531
-	bes.m3.c4m32	1:8	4	32	0.03636731
-	bes.m3.c8m64	1:8	8	64	0.07215131

金融华中-武汉

节点类型	规格名称	规格比例	CPU（核）	内存（GB）	单价（元/分钟/节点）
通用型	bes.g3.c2m8	1:4	2	8	0.01190137
-	bes.g3.c4m16	1:4	4	16	0.02309853
-	bes.g3.c8m32	1:4	8	32	0.04549285
-	bes.g3.c16m64	1:4	16	64	0.09028149
-	bes.g3.c32m128	1:4	32	128	0.17985877
计算型	bes.c3.c2m4	1:2	2	4	0.00898365
-	bes.c3.c4m8	1:2	4	8	0.01726309
-	bes.c3.c8m16	1:2	8	16	0.03382197
-	bes.c3.c16m32	1:2	16	32	0.06693973
-	bes.c3.c32m64	1:2	32	64	0.13317525
密集计算型	bes.ic3.c4m4	1:1	4	4	0.01434537
-	bes.ic3.c8m8	1:1	8	8	0.02798653
-	bes.ic3.c12m12	1:1	12	12	0.07746401
-	bes.ic3.c16m16	1:1	16	16	0.05526885
内存型	bes.m3.c2m16	1:8	2	16	0.01773681
-	bes.m3.c4m32	1:8	4	32	0.03476941
-	bes.m3.c8m64	1:8	8	64	0.06883461

中国香港

节点类型	规格名称	规格比例	CPU（核）	内存（GB）	单价（元/分钟/节点）
通用型	bes.g3.c2m8	1:4	2	8	0.01417603
-	bes.g3.c4m16	1:4	4	16	0.02771043
-	bes.g3.c8m32	1:4	8	32	0.05477923
-	bes.g3.c16m64	1:4	16	64	0.10891683
-	bes.g3.c32m128	1:4	32	128	0.21719203
计算型	bes.c3.c2m4	1:2	2	4	0.00965283
-	bes.c3.c4m8	1:2	4	8	0.01866403
-	bes.c3.c8m16	1:2	8	16	0.03668643
-	bes.c3.c16m32	1:2	16	32	0.07273123
-	bes.c3.c32m64	1:2	32	64	0.14482083
密集计算型	bes.ic3.c4m4	1:1	4	4	0.01414083
-	bes.ic3.c8m8	1:1	8	8	0.02764003
-	bes.ic3.c12m12	1:1	12	12	0.04113923
-	bes.ic3.c16m16	1:1	16	16	0.05463843
内存型	bes.m3.c2m16	1:8	2	16	0.02322243
-	bes.m3.c4m32	1:8	4	32	0.04580323
-	bes.m3.c8m64	1:8	8	64	0.09096483

新加坡

节点类型	规格名称	规格比例	CPU（核）	内存（GB）	单价（元/分钟/节点）
通用型	bes.g3.c2m8	1:4	2	8	0.01754137
-	bes.g3.c4m16	1:4	4	16	0.03434937
-	bes.g3.c8m32	1:4	8	32	0.06796537
-	bes.g3.c16m64	1:4	16	64	0.13519737
-	bes.g3.c32m128	1:4	32	128	0.26966137
计算型	bes.c3.c2m4	1:2	2	4	0.01188737
-	bes.c3.c4m8	1:2	4	8	0.02304137
-	bes.c3.c8m16	1:2	8	16	0.04534937
-	bes.c3.c16m32	1:2	16	32	0.08996537
-	bes.c3.c32m64	1:2	32	64	0.17919737
密集计算型	bes.ic3.c4m4	1:1	4	4	0.01738737
-	bes.ic3.c8m8	1:1	8	8	0.03404137
-	bes.ic3.c12m12	1:1	12	12	0.05069537
-	bes.ic3.c16m16	1:1	16	16	0.06734937
内存型	bes.m3.c2m16	1:8	2	16	0.02884937
-	bes.m3.c4m32	1:8	4	32	0.05696537
-	bes.m3.c8m64	1:8	8	64	0.11319737

磁盘价格

华北-北京，华北-保定，华东-苏州，华东-上海，华南-广州

磁盘类型	计费项	元/GB/分钟	元/GB/小时	元/GB/天
SSD云磁盘	存储空间	0.0000208	0.001248	0.03
高性能云磁盘	存储空间	0.0000081	0.000486	0.0117

金融华中-武汉

磁盘类型	计费项	元/GB/分钟	元/GB/小时	元/GB/天
SSD云磁盘	存储空间	0.00002708	0.0016248	0.0389952
高性能云磁盘	存储空间	0.000010533	0.00063198	0.01516752

中国香港

磁盘类型	计费项	元/GB/分钟	元/GB/小时	元/GB/天
SSD云磁盘	存储空间	0.000029167	0.00175002	0.042
高性能云磁盘	存储空间	0.00001135	0.000681	0.016344

新加坡

磁盘类型	计费项	元/GB/分钟	元/GB/小时	元/GB/天
SSD云磁盘	存储空间	0.000033333	0.00199998	0.04799952
高性能云磁盘	存储空间	0.000012967	0.00077802	0.01867248

节点价格(旧)

华北-北京，华北-保定，华东-苏州，华南-广州

套餐名称	节点类型	CPU（核）	内存（GB）	硬盘	单价（元/分钟/节点）
存储1型	Elasticsearch Node	4	16	500G SSD云磁盘	0.08
存储2型	Elasticsearch Node	8	32	1T SSD云磁盘	0.15
存储3型	Elasticsearch Node	16	64	2T SSD云磁盘	0.3
计算1型	Elasticsearch Node	4	32	100G SSD云磁盘	0.06
计算2型	Elasticsearch Node	16	64	200G SSD云磁盘	0.15
计算3型	Elasticsearch Node	32	128	500G SSD云磁盘	0.3

金融华中-武汉

套餐名称	节点类型	CPU（核）	内存（GB）	硬盘	单价（元/分钟/节点）
存储1型	Elasticsearch Node	4	16	500G SSD云磁盘	0.1
存储2型	Elasticsearch Node	8	32	1T SSD云磁盘	0.2
存储3型	Elasticsearch Node	16	64	2T SSD云磁盘	0.39
计算1型	Elasticsearch Node	4	32	100G SSD云磁盘	0.08
计算2型	Elasticsearch Node	16	64	200G SSD云磁盘	0.2
计算3型	Elasticsearch Node	32	128	500G SSD云磁盘	0.39

中国香港

套餐名称	节点类型	CPU（核）	内存（GB）	硬盘	单价（元/分钟/节点）
存储1型	Elasticsearch Node	4	16	500G SSD云磁盘	0.11
存储2型	Elasticsearch Node	8	32	1T SSD云磁盘	0.21
存储3型	Elasticsearch Node	16	64	2T SSD云磁盘	0.4
计算1型	Elasticsearch Node	4	32	100G SSD云磁盘	0.06
计算2型	Elasticsearch Node	16	64	200G SSD云磁盘	0.15
计算3型	Elasticsearch Node	32	128	500G SSD云磁盘	0.3

新加坡

套餐名称	节点类型	CPU（核）	内存（GB）	硬盘	单价（元/分钟/节点）
存储1型	Elasticsearch Node	4	16	500G SSD云磁盘	0.1454
存储2型	Elasticsearch Node	8	32	1T SSD云磁盘	0.091
存储3型	Elasticsearch Node	16	64	2T SSD云磁盘	0.1815
计算1型	Elasticsearch Node	4	32	100G SSD云磁盘	0.0522
计算2型	Elasticsearch Node	16	64	200G SSD云磁盘	0.123
计算3型	Elasticsearch Node	32	128	500G SSD云磁盘	0.2488

> 说明：针对不同类型、地域的用户所能选择的配置可能有所差异，以Console显示实际所能购买的配置为准。

计费规则

百度Elasticsearch根据您选择的实例机型配置和个数，按使用时长（分钟级）计费。具体计费规则如下：

- 按分钟计费， 不足1分钟按1分钟计。
- 订单提交并且集群实例已经分配时，开始计费。
- 删除集群时立即释放所有资源并停止计费。
- 集群处于停止状态时仍然会计费，因为此时资源并没有释放；如果要停止计费，必须删除集群，释放资源。
- 外网流出流量目前不计费。

余额不足和欠费

余额不足提醒

- 根据您最近3天的账单金额来判断您的账户余额（含可用代金券）是否足够支付未来3天的费用，若不足以支付，系统发送续费提醒。
- 根据您最近1天的账单金额来判断您的账户余额（含可用代金券）是否足以支付未来1天的费用，若不足以支付，系统发送续费提醒。

欠费处理

- 北京时间整点检查您的账户余额是否足以支付本次Elasticsearch账单的费用（如北京时间11点整检查账户余额是否足以支付10点至11点的账单费用），若不足以支付，即为欠费，欠费时系统会发送欠费通知。
- 欠费后立即停止服务，系统会发送欠费停服通知。为了不影响您的服务，建议您在使用服务时要保证账户中的金额足够支付。
- 欠费7天后，如果还没有充值，系统会自动删除服务并释放资源。

配置变更费用说明

百度智能云Elasticsearch的节点配置和数量可以根据业务的需求方便的进行调整，配置的变更会产生相应的费用，以下对配置变更的几种情况进行费用说明。

预付费配置升级

集群配置升级包括节点规格的升级、节点数量增加和磁盘容量增加，配置的升级会产生差价费用。

费用规则

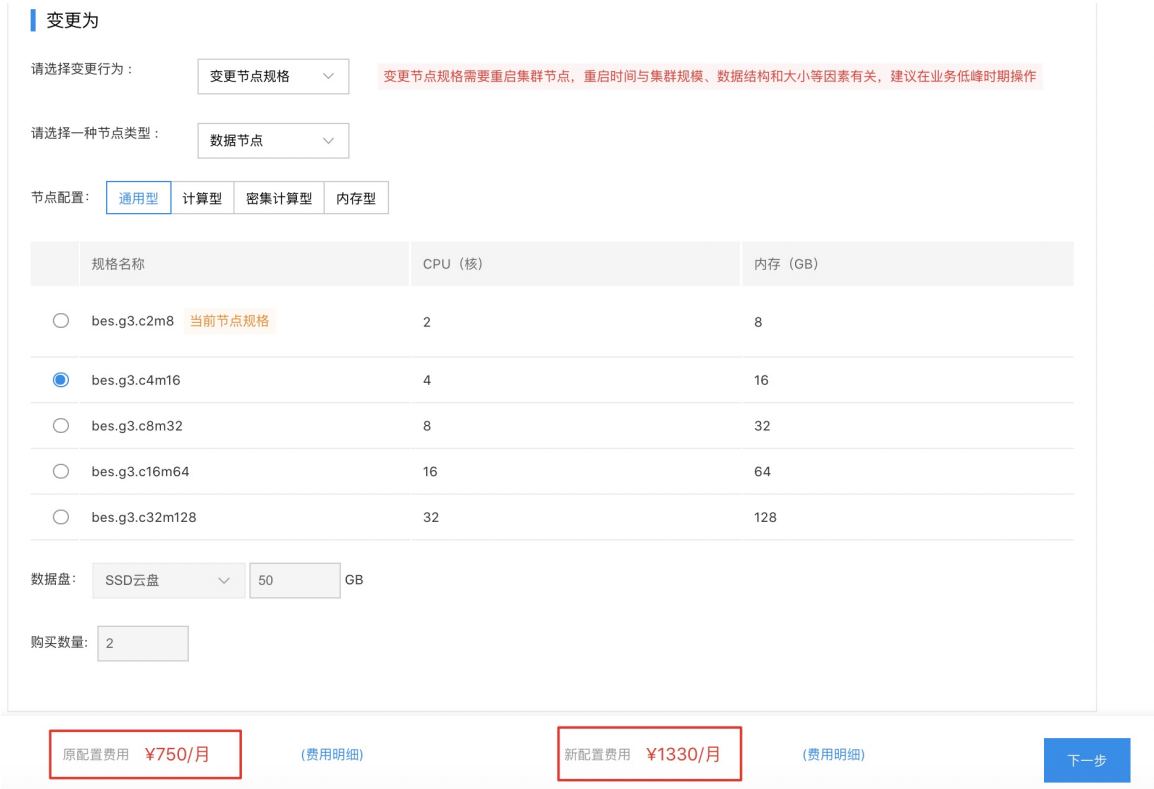
- 升配费用 = 月配置差价 × （合同剩余天数/合同总天数) × 合同月数
- 月配置差价 = （新配置价格 - 原配置价格）
- 配置升级不影响资源到期时间

升配场景案例

节点规格升级

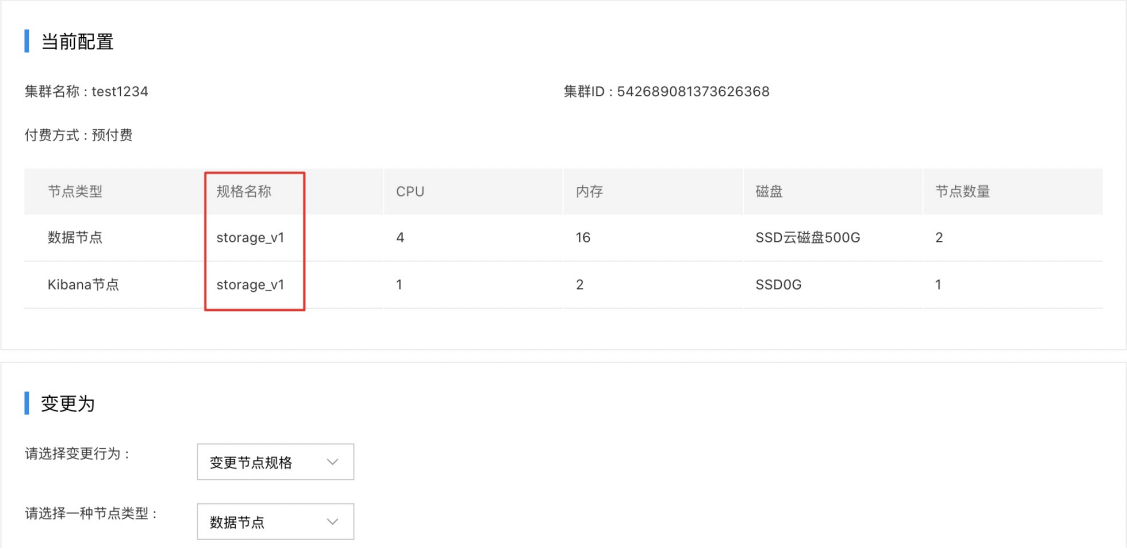
例子1：2021年8月4日将一个从`2021-07-29`到`2021-08-29`的预付费集群节点变更为配置更高规格的节点。

- 原配置750元/月，新配置1330元/月，差价580元/月
- 总合同天数31天，剩余24天
- 升配费用 = 580元/月 × （24/31) × 1月 = 449.03元



例子2：2021年8月4日将一个从`2021-08-03`到`2022-08-03`的预付费集群节点变更为配置更高规格的节点。

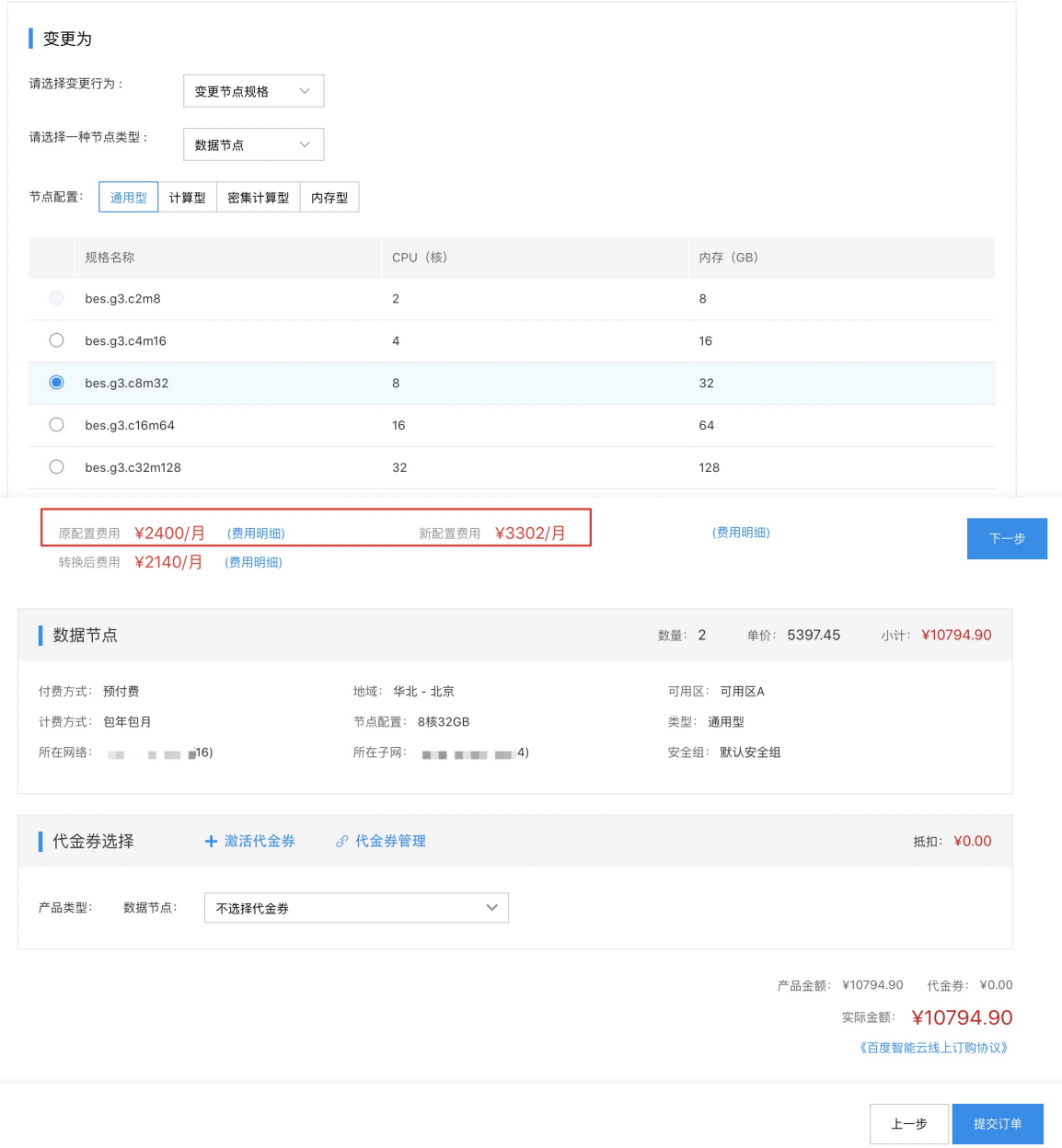
> 此案例为旧节点集群升级节点规格。旧节点规格是形如`storage_v1`、`calculate_v1`等形式的节点。新旧节点规格查看此处。



- 旧节点的规格升级，会按照先转换为新规格的方式实现配置升级。

- 由于新旧节点的价格模式不同，会产生拆分后同规格的节点价格不一致的情况，一般来说价格会更加优惠。
- 原配置2400元/月，新配置3302元/月，差价902元/月

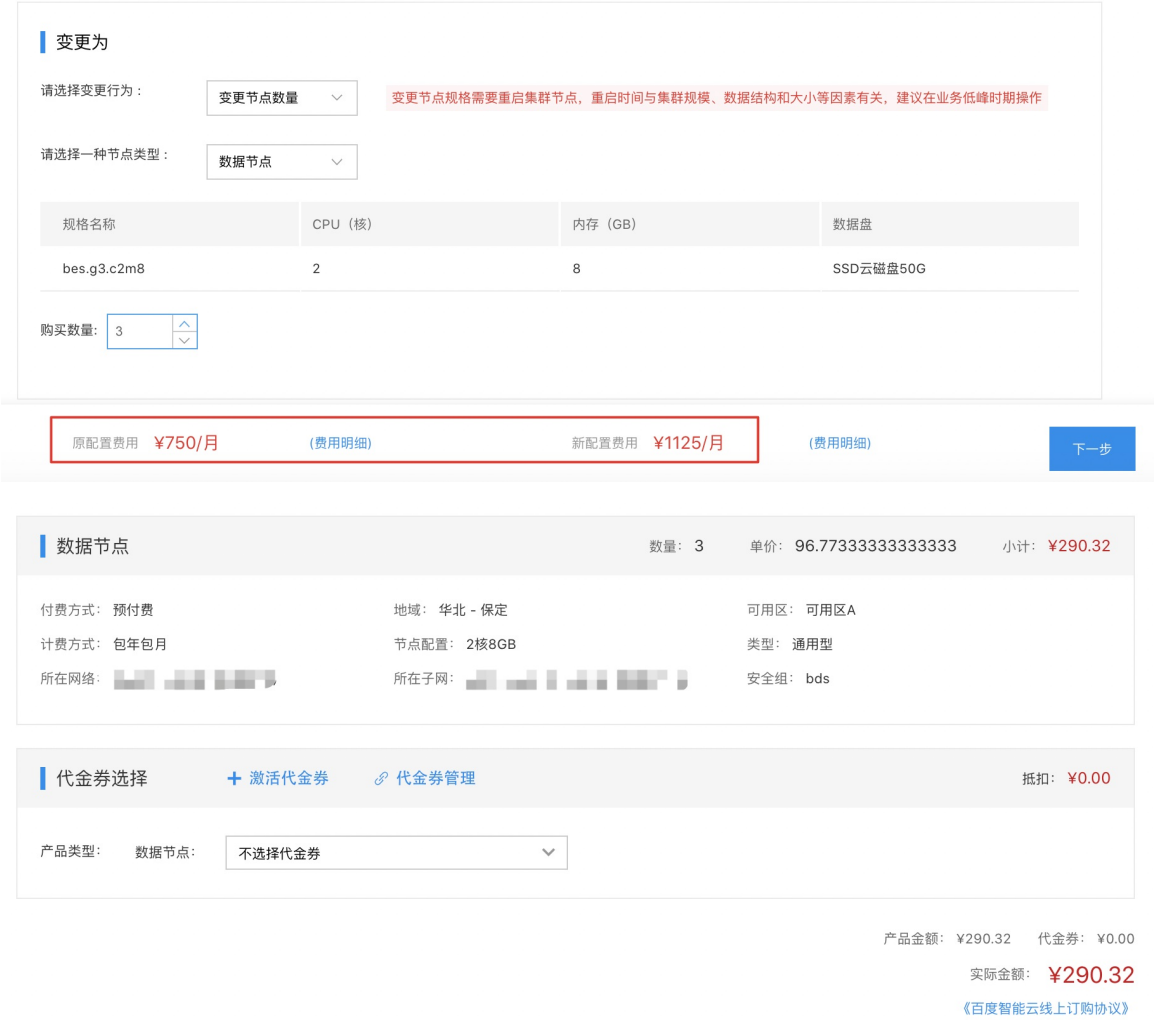
- 总合同天数365天，剩余364天
- 升配费用 = 902元/月 × （364/365) × 12月 = 10794元



节点数量增加

例子1：2021年8月4日将一个从`2021-07-29`到`2021-08-29`的预付费集群数据节点数量从2变更3个数据节点。

- 原配置750元/月，新配置1125元/月，差价375元/月
- 总合同天数31天，剩余24天
- 升配费用 = 375元/月 × （24/31) × 1月 = 290.32元



磁盘容量增加

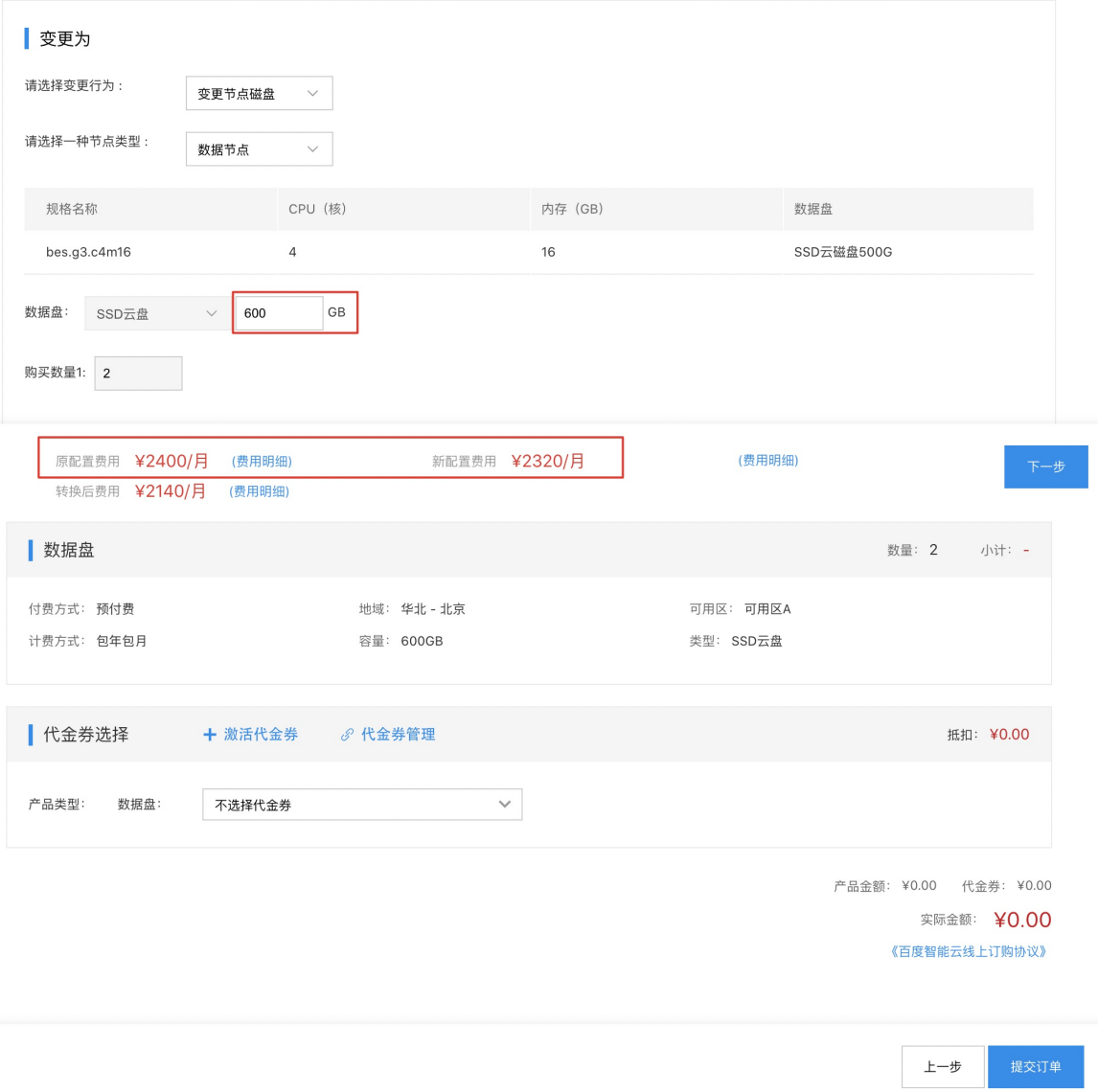
例子1：2021年8月4日将一个从`2021-07-29`到`2021-08-29`的预付费集群数据节点磁盘容量从SSD 50G变更为 SSD 200G。

- 原配置750元/月，新配置1020元/月，差价270元/月
- 总合同天数31天，剩余24天
- 升配费用 = 270元/月 × （24/31）× 1月 = 209.03元

例子2：2021年8月4日将一个从`2021-08-03`到`2022-08-03`的预付费集群节点磁盘容量从SSD512G变更SSD600G。

> 此案例为旧节点集群升级节点磁盘容量。

- 原配置2400元/月，新配置2320元/月，差价-80元/月
- 总合同天数365天，剩余364天
- 升配费用 = -80元/月 × （364/365) × 12月 = -957.37元 < 0，不退费，不额外收费。



后付费配置升级

后付费配置的升级费用比较简单，配置升级一般情况下单价会提高，但是部分地域的节点价格可能有过降价调整，同配置或者更高配置的节点可能会比以前的旧规格节点更优惠。后付费的集群不涉及退费和补差价问题，调整配置之后，从调整配置时间开始，按照新配置费用进行计费即可。

向量检索特性

能力简介

百度Elasticsearch团队自主开发向量引擎，专用于存储、检索、分析多维向量数据。支持多种索引类型和相似度计算方法，支持构建十亿级向量规模，实现毫秒级延迟。不仅能为文心等大模型提供外部知识库能力，提高大模型回答的准确性和时效性，还可广泛应用于推荐系统、问答系统、语义检索、智能客服等领域。

相关文档链接

- 快速入门 https://cloud.baidu.com/doc/BES/s/wllejzxis
- 资源评估 https://cloud.baidu.com/doc/BES/s/tllek0mxh
- 算法介绍 https://cloud.baidu.com/doc/BES/s/zllejszu7

关键概念

非结构化数据

非结构化数据是指无法存储在预定义格式中或数据模型的数据，如图像、视频、音频等。与结构化数据相比，非结构化数据的处理和分析更加复杂。

向量

向量（vector）是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。

向量检索

向量检索是在指定向量数据集中，查找与目标向量最相似的结果的过程。相似的向量通常具有相近的原始数据，通过向量检索可以挖掘出原始数据之间的联系。

KNN

KNN （K-Nearest Neighbor Search）指的是最近邻搜索。它的原理是：计算待查询向量与数据库中所有向量之间的距离，然后按照距离从小到大排序，选择距离最近的 K 个向量作为查询结果。KNN 算法的优点是可以保证精确的结果，但是对于大规模的向量数据，计算量会非常大，效率较低。

ANN

ANN（Approximate Nearest Neighbor Search） 表示近似最近邻搜索，是一种用于高维数据空间中快速查找最近邻点的方法。与KNN（精确最近邻搜索——相比，ANN 牺牲了一定的精度以换取更高的搜索速度，因此在处理大规模数据集时具有较高的效率。ANN 方法通常会对数据进行预处理，从而在查询时减少计算距离的次数。ANN 算法的优点是速度快、效率高，但是相对于 KNN 算法来说，其结果可能不够精确。

HNSW

HNSW（Hierarchical Navigable Small World） 是一种基于图的高维向量相似性搜索算法。通过构建一张图来表示向量之间的相似度关系，并使用一些优化策略来加速搜索过程。

算法介绍

算法介绍

百度Elasticsearch向量引擎提供了两种向量检索算法： linear 算法和 hnsw、hnsw_sq8、hnsw_qp、ivf、ivf_pq、ivf_hnsw、ivf_hnsw_pq 等 ANN 算法（Approximate Nearest Neighbor，近似最近邻算法）。

算法	含义	适用场景	缺点	支持距离算法
linear	线性计算所有向量数据	召回率100% 查询时间与数据量成正比 通常用于小数据量场景或效果对照	大数据量下效率较低 查询大量消耗cpu	欧式距离（l2） 余弦距离（cosine） 内积（innerproduct）
hnsw	基于hnsw算法对数据进行近似计算	单机数据量小 对召回率要求高 对查询速度要求高	全内存 消耗内存大	欧式距离（l2） 余弦距离（cosine） 内积（innerproduct）
hnsw_sq8	基于hnsw算法对数据进行近似计算，对向量进化标量量化	对召回率要求较高 对查询速度要求高 需求内存占用比较小	全内存 有精度损失，召回率比hnsw低	欧式距离（l2） 内积（innerproduct）
hnsw_pq	基于hnsw算法对数据进行近似计算，对向量进化乘积量化	对召回率要求适中 对查询速度要求高 需求内存占用比较小	全内存 精度较低，召回率比hnsw低，往往也比hnsw_sq8低，不过内存消耗更少	欧式距离（l2） 内积（innerproduct）
ivf	基于ivf算法对数据进行近似计算	单机数据量小 对召回率要求较高 对查询速度要求适中	全内存 消耗内存较大 查询计算量大，大数据量下效率较低	欧式距离（l2） 内积（innerproduct）
ivf_pq	基于ivf算法对数据进行近似计算，对向量进行乘积量化	对查询速度要求较高 对召回率要求适中 需求内存占用比较小	全内存 精度较低 查询计算量较大，大数据量下效率不高	欧式距离（l2） 内积（innerproduct）
ivf_hnsw	基于ivf算法对数据进行近似计算，分桶内构建hnsw索引	单机数据量小 对召回率要求较高 对查询速度要求适中	全内存 消耗内存较大 查询计算量大，大数据量下效率较低	欧式距离（l2） 内积（innerproduct）
ivf_hnsw_pq	基于ivf算法对数据进行近似计算，分桶内构建hnsw索引，对向量进行乘积量化	对查询速度要求高 对召回率要求较低 需求内存占用比较小	全内存 精度较低	欧式距离（l2） 内积（innerproduct）

Elasticsearch Score详解

默认情况下，ES的检索结果会按照相关性分数排序并返回（由高到低），相关性分数体现了document和检索的相关程度，通过一个大于0的浮点数来体现。向量检索的场景下，会基于向量之间的距离为基础进行计算，不过为了满足相关性分数大于0的约束，ES检索结果的_source字段不一定会直接返回距离计算结果。

向量检索引擎支持的距离算法包括：余弦距离（cosine）、欧式距离（l2）、内积（innerproduct），距离公式和相关下分数计算逻辑如下：

linear

距离算法	距离公式	Elasticsearch Score
l2	$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$	$\frac{1}{1+\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}}$
cosine	$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$	$\frac{1+\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}}{2}$
innerproduct	$sum(Ai * Bi)$	$x = \begin{cases} sum(Ai * Bi) + 1 & \text{if } sum(Ai * Bi) > 0 \\ 1/(1 - sum(Ai * Bi)) & \text{if } sum(Ai * Bi) <= 0 \end{cases}$

hnsw

距离算法	距离公式	Elasticsearch Score
l2	$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$	$\frac{1}{1+\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}}$
cosine	$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$	$\frac{1}{2-\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}}$
innerproduct	$sum(Ai * Bi)$	$x = \begin{cases} sum(Ai * Bi) + 1 & \text{if } sum(Ai * Bi) > 0 \\ 1/(1 - sum(Ai * Bi)) & \text{if } sum(Ai * Bi) <= 0 \end{cases}$

资源规划

使用BES向量数据库前，用户需要优先结合业务需求和所存数据，评估集群所需的资源容量。评估方式如下：

资源估算

CPU资源

CPU资源决定了向量数据库查询的QPS能力，以基于标准embedding的768维数据为例，单节点100万条数据的情况下，数据节点的CPU数量与查询QPS能力关系如下：

```
CPU核数 = 预估QPS / 80
```

上述核数估计方式是经验公式，实际生产中一般需要通过压测方式评估最佳CPU核数选择。

内存资源

BES的数据节点默认会预留一半内存给 JVM（JVM最大占用30G内存），向量模块默认分配剩余内存的 50% 为向量数据构建索引。

```
 向量索引的最大内存占用比例可以通过`circuit_breaker_limit`参数调节，默认设置为 50%。调节方式如下：
PUT /_cluster/settings
{
   "persistent" : {
      "bpack.knn.memory.circuit_breaker.limit" : "70%"
   }
}
```

快速评估内存

我们对于单节点容纳常用维度向量的能力进行了预算；提供不同节点规格，单节点支持常用维度向量的最大数据数量级。

如果用户使用的向量维度，算法以及向量模块分配内存比例与我们计算条件相匹配，可以通过下表进行快速集群规模计算。

基于标准embedding的768维数据，以HNSW算法存储，`circuit_breaker_limit` 参数为 50% 情况下，单节点支持最大向量数据量级如下表所示：

BES数据节点内存规格	JVM堆内存	向量索引占用的最大内存	单节点支持的最大向量数据量
32G	16G	8G	2,000,000
64G	30G	17G	4,000,000
128G	30G	49G	14,000,000

基于标准embedding的384维数据，以HNSW算法存储，`circuit_breaker_limit` 参数为 50% 情况下，单节点支持最大向量数据量级如下表所示：

BES数据节点内存规格	JVM堆内存	向量索引占用的最大内存	单节点支持的最大向量数据量
32G	16G	8G	4,000,000
64G	30G	17G	9,000,000
128G	30G	49G	26,000,000

上述单节点支持的最大向量数据量，均在不存在副本情况计算所得；如果存在k个副本，则单节点支持的最大向量数据量需要除以（1+k）

> 例如基于标准embedding的768维数据，以HNSW算法存储，`circuit_breaker_limit` 参数为 50% 为情况下，如果存在3个副本，32G规格节点单节点最大支持向量的数据量为500000

内存详细估算方式

如果用户使用的向量维度、算法或者内存分配比例参数与上述预算条件不同，可以通过以下方式计算所需的集群规模。

以下为一种通过估算集群的内存用量来计算集群规模的方法。需要考虑的因素包括算法的选择、给向量索引分配内存比例、向量维度和数量以及副本数。

HNSW 内存预估公式

HNSW 所需的存放向量索引内存估算公式为：

```
(1.1 * (4 * dims + 8 * M) * num of vectors ) bytes
```

注：其中M值表示构造HNSW索引期间为每个新元素创建的双向链接数。M值越高，消耗内存、存储空间以及构造时间越多但是准确率更高。

建议根据min(向量维度 * 1.5, 32)取值，一般12-48可以满足大多数场景的需求。

假设用户有100万个维度为 768 的向量，且 M 设置为 32，内存需求可以估计如下：

```
1.1 * (4 * 768 + 8 * 32) * 1,000,000 ≈ 3.409 GB
```

IVF 内存预估公式

IVF 所需的存放向量索引内存估算公式为：

```
1.1 * (((4 * dims) * num of vectors) + (4 * nlist * dims)) bytes.
```

假设用户有100万个维度为256的向量，且 nlist 设置为128，内存需求可以估计如下：

```
1.1 * (((4 * 256) * 1,000,000) + (4 * 128 * 256))  ≈ 1.126 GB
```

节点规模试算

以HNSW算法为例，1亿条768维向量(默认 M 为32)，副本数为1，节点规格选择8核64G，则使用节点数为

```
 向量占用的内存总量 (1.1 * (4 * dims + 8 * M) * num of vectors ) bytes =  （1.1 * (4 * 768 + 8 * 32) * 10kw）= 366080000000 byte ≈ 340.939 GB
考虑到有1个副本，所需要提供给向量的总内存为 340.939 GB * (1 + 1) = 681.878 GB
使用默认circuit_breaker_limit = 50%，则规格为64G的节点，单节点可以提供给向量数据的最大内存为（64G-30G）* 50% = 17G
因此需要节点数为 681.878 GB / 17G = 40.011台 ≈ 41 台
```

如果使用16核128G节点，并且调整 `circuit_breaker_limit` 为70%，则使用节点数为

```
 向量占用的内存总量 (1.1 * (4 * dims + 8 * M) * num of vectors ) bytes =  （1.1 * (4 * 768 + 8 * 32) * 10kw）= 366080000000 byte ≈ 340.939 GB
考虑到有1个副本，所需要提供给向量的总内存为 340.939 GB * (1 + 1) = 681.878 GB
使用默认circuit_breaker_limit = 50%，则规格为128G的节点，单节点可以提供给向量数据的最大内存为（128G-30G）* 70% = 68.6G
因此需要节点数为 681.878 GB / 68.6G = 42.617台 ≈ 9.940 台 ≈ 10 台
```

磁盘资源

数据节点所需存储容量计算方式如下：

```
 存储容量 = 源数据大小 * (1 + 副本数量) * (1 + 数据膨胀) * (1 + 内部其他开销) * (1 + 预留空间)

源数据大小 = 标量数据量 + 向量数据量
标量数据量 = 原始标量数据的字节数
向量数据量 ≈ (4 * dims) * num of vectors * 4
```

- 副本数量: 推荐的副本数量为1
- 数据膨胀: 一般情况下源数据会膨胀10%-50%，用于 ES 存储索引、列存数据等 (_all参数等未计算)
- 内部其他开销: 需要预留20%，用于 segment 合并、ES Translog 等内部操作
- 安全阈值: 在上述基础上，通常至少再额外预留30%的安全阈值

因此最终的所需容量计算公式为:

```
 存储容量 ≈ 源数据大小 * (1 + 副本数量) * 1.7
```

具体原理详见 https://cloud.baidu.com/doc/BES/s/Vke3oar2l

以8核64G和16核128G的BES节点为例，数据量和选购磁盘的关系如下：

标量数据	向量数据量 (基于标准embedding的768维数据)	副本分片数	节点规格	建议节点数	所需磁盘总大小	建议选购单机规格
1000G	10,000,000	1	8核64G	5	3520G	8核64G + 700G磁盘
10,000G	100,000,000	1	8核64G	41	35200G	8核64G + 900G磁盘
1000G	10,000,000	1	16核128G	2	3520G	16核128G + 1.8T磁盘
10,000G	100,000,000	1	16核128G	10	35200G	16核128G + 3.5T磁盘

快速入门

用户可以参考本文档，快速使用BES向量数据库的hnsw算法进行数据写入和向量检索。

前提条件

已创建百度智能云账号。创建账号参考百度云账号注册流程。

创建集群

创建BES集群，请参考文档：创建集群。

如需规划集群的资源规格及数量，请参考文档：资源规划。

访问集群

参考文档：访问Elasticsearch服务。

快速体验

1、创建一个hnsw的索引，详细参数含义可以参考文档：创建索引。

```
PUT my_index
{
  "settings": {
    "index": {
      "knn": true
    }
  },
  "mappings": {
    "properties": {
      "image-vector": {
        "type": "bpack_vector",
        "dims": 4,
        "index_type":  "hnsw",
        "space_type": "l2",
        "parameters": {
            "ef_construction": 200,
            "m": 32
        }
      },
      "id": {
        "type": "integer"
      },
      "title": {
        "type": "keyword"
      }
    }
  }
}
```

2、写入向量数据

```
POST my_index/_bulk
{"index":{}}
{"id":1, "title":"t1", "image-vector":[3.5,4.5,6.5,6.5]}
{"index":{}}
{"id":2, "title":"t2", "image-vector":[3.5,4.5,6.5,6.5]}
{"index":{}}
{"id":3, "title":"t1", "image-vector":[5.5,6.5,6.5,6.5]}
{"index":{}}
{"id":4, "title":"t2", "image-vector":[3.5,6.5,6.5,6.5]}
{"index":{}}
{"id":5, "title":"t1", "image-vector":[5.5,2.5,4.5,4.5]}
```

3、索引构建完成后，即可检索数据

```
GET my_index/_search
{
  "size": 2,
  "query": {
    "knn": {
      "image-vector": {
        "vector": [3, 4, 5, 6],
        "k": 2,
        "ef": 100,
        "filter": { // 基于标量数据进行过滤，不需要的话可以删掉filter
          "term": {
            "title": "t1"
          }
        }
      }
    }
  }
}
```

操作指南

创建索引

用户可以参考本文档，来进行向量索引创建。

创建索引

向量索引在创建的时候，需要指定 `settings.index.knn：true`，以开启向量相关的功能。同时，可以在创建索引的时候指定分片数量`number_of_shards` 和副本数量`number_of_replicas`。

- 分片数量在索引创建后无法修改，增加分片数量，可以提高索引的整体容量，建议单个分片存储100万-500万向量，存储容量不超过50GB。分片数量不宜过高，因为对索引的查询需要汇总所有分片上的子查询结果，所以分片数量会增加查询消耗。因此，增加分片数量不会提高查询QPS。
- 副本数量定义了每个分片有多少份数据副本，可以随时修改，取值范围[0, 节点数-1]。提高副本数量，可以扩展查询可用副本，提高整体QPS，不过会增加数据冗余、存储成本、写入和查询时的资源消耗等。目标副本数超过当前节点数时，集群需要扩容。

BES支持hnsw、hnsw_sq8、hnsw_pq、ivf、ivf_qp、ivf_hnsw、ivf_hnsw_pq等多种不同的算法，用户可以根据自己的业务场景进行选择，算法的说明参考：算法介绍。

用户在创建向量索引的时候，可以为每个字段指定需要的索引算法和算法相关的参数，详情请参考以下示例和参数说明：

```
PUT my_index
{
  "settings": {
    "index": {
      "knn": true,
      "number_of_shards": 5, // 索引的分片数量，创建后无法修改，建议单个分片存储100万-500万向量，建议单个分片存储容量不超过50GB
      "number_of_replicas": 1 // 分片的副本数量，取值范围[0, 节点数-1]，可随时修改
    }
  },
  "mappings": {
    "properties": {
      "id": {
        "type": "long"
      },
      "file-type": {
        "type": "keyword"
```

```
    },
    "field_hnsw": {
      "type": "bpack_vector",
      "index_type": "hnsw", // 构建hnsw索引
      "dims": 768, // 设置实际的向量维度
      "space_type": "l2",
      "parameters": { // 指定hnsw参数
        "m": 32,
        "ef_construction": 256
      }
    },
    "field_hnsw_default": { // 只配置必填参数，其他参数均为默认值
      "type": "bpack_vector",
      "index_type": "hnsw", // 构建hnsw索引
      "dims": 768
    },
    "field_linear": { // 只想用linear算法的字段，可以把build_index参数置为false，代表不会实际构建向量索引结构
      "type": "bpack_vector",
      "index_type": "hnsw",
      "dims": 768,
      "build_index": false // 默认为true
    },
    "field_hnsw_sq8": {
      "type": "bpack_vector",
      "dims": 768,
      "index_type": "hnsw_sq8", // 构建hnsw_sq8索引
      "space_type": "l2",
      "parameters": { // 指定hnsw参数
          "ef_construction": 256,
          "m": 32
      }
    },
    "field_hnsw_pq": {
      "type": "bpack_vector",
      "dims": 768,
      "index_type": "hnsw_pq", // 构建hnsw_pq索引
      "space_type": "l2",
      "parameters": { // 指定hnsw_pq参数
          "ef_construction": 256,
          "m": 32,
          "pqm": 128
      }
    },
    "field_ivf": {
      "type": "bpack_vector",
      "dims": 768,
      "index_type": "ivf", // 构建ivf索引
      "space_type": "l2",
      "parameters": { // 指定ivf参数
          "nlist": 100
      }
    },
    "field_ivf_pq": {
      "type": "bpack_vector",
      "dims": 768,
      "index_type": "ivf_pq", // 构建ivf_pq索引
      "space_type": "l2",
      "parameters": { // 指定ivf_pq参数
          "nlist": 100,
          "pqm": 128
      }
    },
    "field_ivf_hnsw": {
```

```
        "field_ivf_hnsw": {
          "type": "bpack_vector",
          "dims": 768,
          "index_type": "ivf_hnsw", // 构建ivf_hnsw索引
          "space_type": "l2",
          "parameters": { // 指定ivf_hnsw参数
            "nlist": 100,
            "ef_construction": 256,
            "m": 32
          }
        },
        "field_ivf_hnsw_pq": {
          "type": "bpack_vector",
          "dims": 768,
          "index_type": "ivf_hnsw_pq", // 构建ivf_hnsw_pq索引
          "space_type": "l2",
          "parameters": { // 指定ivf_hnsw_pq参数
            "nlist": 100,
            "ef_construction": 256,
            "m": 32,
            "pqm": 128
          }
        }
      }
    }
  }
```

参数

向量字段相关参数

参数	类型	是否必填	默认值	创建后是否可修改	说明
type	string	是	bpack_vector	否	字段类型，向量字段类型固定为 bpack_vector
dims	int	是	无	否	向量维数，必填项
index_type	string	是	hnsw	否	向量检索算法，必填项，取值参考下方详细说明
space_type	string	否	cosine	否	向量检索空间距离算法，取值参考下方详细说明
parameters	object	否	null	否	向量检索算法的具体参数，取值参考下方详细说明

index_type 参数说明

参数值	支持的距离算法（space_type）	说明
hnsw	l2，cosinesimil（cosine），innerproduct（dot_prod）	hnsw算法
hnsw_sq8	l2，cosinesimil（cosine），innerproduct（dot_prod）	hnsw算法+sq8量化
hnsw_pq	l2，cosinesimil（cosine），innerproduct（dot_prod）	hnsw算法+pq量化
ivf	l2，innerproduct（dot_prod）	ivf算法
ivf_pq	l2，innerproduct（dot_prod）	ivf算法+pq量化
ivf_hnsw	l2，innerproduct（dot_prod）	ivf算法+聚簇内构建hnsw索引
ivf_hnsw_pq	l2，innerproduct（dot_prod）	ivf算法+聚簇内构建hnsw索引+pq量化

parameters 参数说明

参数	类型	是否必填	默认值	创建后是否可修改	说明
m	int	否	16	否	hnsw算法参数。此参数表示构造期间为每个新元素创建的双向链接数，主要影响内存、存储消耗以及召回率。m值越高，意味着更高消耗的内存和存储，更慢的索引构建时间，以及更好的召回率。建议根据min(向量维度 * 1.5, 32)取值，以保证性能，12-48可以满足大多数场景的需求。
ef_construction	int	否	200	否	hnsw算法参数。此参数表示在索引构建过程中，最近邻居的动态扫描区域大小。该值越大，越不容易陷入局部最优解，召回率更高，但是索引构建越慢，取值范围为[2,+∞]。需要注意，如果用户业务需求要求检索top k，那设置的ef_construction值需要大于k。
pqm	int	否	1	否	pq量化算法参数，需要可以被向量维数整除。pq算法的字向量量化器数量，也即子向量数量。一般建议设置为向量维数的1/4；m 取值越小内存占用越小，查询速度越快，精度也更低。
nlist	int	否	4	否	ivf算法参数。向量划分的桶数，较高的值意味着向量数据会被划分到更多的桶，每个桶内到数据更少，查询单个桶的向量数据的资源消耗和时间都会减少，但是查询时相同数量的桶包含的向量数量变少了，召回率可能会下降。查询时需要设置合理的 nprobes 参数以保证召回率。一般建议 nlist = C * sqrt(N)，C为用户自选的常数，可以根据具体情况调整，比如可以从C=4开始尝试，再根据业务所需要的召回率来调整测试。

写入和查询数据

写入数据

bulk写入 用户创建向量索引后，即可使用ES BULK API写入数据。不同的index_type与space_type，在数据写入时没有区别。在以下示例中，我们向my_index索引的field_hnsw和field_linear字段分别写入样例向量数据。

```
POST my_index/_bulk
{"index":{}}
{"field_hnsw":[3.5,4.5,6.5,6.5], "field_linear":[3.5,4.5,6.5,6.5]}
{"index":{}}
{"field_hnsw":[3.5,4.5,6.5,6.5], "field_linear":[3.5,4.5,6.5,6.5]}
{"index":{}}
{"field_hnsw":[5.5,6.5,6.5,6.5], "field_linear":[3.5,4.5,6.5,6.5]}
...
```

集群参数配置 数据写入以后，BES会在后台为向量字段构建向量索引，用户可以通过bpack.knn.index_thread_qty参数设置构建向量索引的并发度，默认值是1，取值范围[1, 32]。并发度越高，构建索引的速度越快，不过CPU消耗也会越高，需要注意过高的bpack.knn.index_thread_qty值可能会导致CPU资源耗尽，影响同时间的检索请求。

bpack.knn.index_thread_qty参数的设置方式如下：

```
PUT /_cluster/settings
{
  "persistent" : {
    "bpack.knn.index_thread_qty" : 4
  }
}
```

查询数据

hnsw、hnsw_sq8、hnsw_pq 查询示例 因为hnsw、hnsw_sq8、hnsw_pq 类型字段的查询参数相同，所以我们用一个示例进行说明。

在以下示例中，我们基于 my_index 索引的 field_hnsw 和file-type字段进行检索，其中file-type字段是keyword类型，存储了图片类型；field_hnsw字段是bpack_vector类型，存储了图像的embedding向量。

```
GET my-index/_search
{
  "size": 3, // 期望的查询结果集大小限制
  "_source": ["id"], // 期望返回的字段列表
  "query": {
    "knn": {
      "field_hnsw": { // 要检索的向量字段
        "vector": [2, 3, 5, 6], // 查询向量
        "k": 10, // 查询的最近邻的数量
        "ef": 256, // hnsw算法的ef_search参数，表示检索最近邻时的动态扫描队列大小
        "filter": { // 填写所需的filter，不需要filter则省略
          "term": {
            "file-type": "png"
          }
        }
      }
    }
  }
}
```

参数说明

参数	类型	是否必填	说明
ef	int	否	hnsw算法的ef_search参数，表示检索最近邻时的动态扫描队列大小。提高ef_search参数可以降低检索陷入局部最优解的概率，提高召回率，相应的，也会消耗更多cpu和内存资源，使查询性能降低。取值需大于k，一般建议200以上。

hnsw + filter 查询示例 以下示例展示了一个较复杂的filter过滤条件+hnsw检索的查询DSL，其中file-type字段是keyword类型，存储了图片类型；description是text类型，存储了图片的描述；size 是long类型，存储了图片大小；user_id是keyword类型，存储了图片的用户标识；field_hnsw字段是bpack_vector类型，存储了图像的embedding向量。

```
GET my-index/_search
{
  "size": 3, // 期望的查询结果集大小限制
  "_source": ["id"], // 期望返回的字段列表
  "query": {
    "knn": {
      "field_hnsw": { // 要检索的向量字段
        "vector": [2, 3, 5, 6], // 查询向量
        "k": 10, // 查询的最近邻的数量
        "ef": 256, // hnsw算法的ef_search参数，表示检索最近邻时的动态扫描队列大小
        "filter": { // 填写用户所需的filter，不需要filter则省略
          "bool": {
            "must": [
              {
                "term": {
                  "file-type": "png"
                }
              },
              {
                "match": {
                  "description": {
                    "query": "little cat"
                  }
                }
              },
              {
                "range": {
                  "size":{
                    "gte": 1024
                  }
                }
              }
            ],
            "must_not": [
              {
                "terms": {
                  "user_id": ["aaa", "bbb"]
                }
              }
            ]
          }
        }
      }
    }
  }
}
```

ivf、ivf_pq 查询示例 因为ivf、ivf_pq 类型字段的查询参数相同，所以我们用一个示例进行说明。

在以下示例中，我们基于 `my_index` 索引的 `field_ivf` 字段进行检索，`field_ivf`字段是`bpack_vector`类型，存储了图像的embedding向量。

```
GET my-index/_search
{
  "size": 3, // 期望的查询结果集大小限制
  "_source": ["id"], // 期望返回的字段列表
  "query": {
    "knn": {
      "field_ivf": { // 要检索的向量字段
        "vector": [2, 3, 5, 6], // 查询向量
        "k": 10, // 查询的最近邻的数量
        "nprobes": 10 // ivf系索引检索时扫描的桶数量
      }
    }
  }
}
```

参数说明

参数	类型	是否必填	说明
nprobes	int	否	ivf系索引检索时扫描的桶数量，nprobes越大，recall越高，但查询性能越差。可以从 nlist / 32 开始调整。再参照用户目标召回率来调整，如果取值等于nlist，则查询性能会跟暴力检索效果相似。

ivf_hnsw、ivf_hnsw_pq 查询示例 因为ivf_hnsw、ivf_hnsw_pq 类型字段的查询参数相同，同时包括ivf索引的参数和hnsw索引的参数，所以我们用一个示例进行说明。

在以下示例中，我们基于 `my_index` 索引的 `field_ivf_hnsw` 进行检索，`field_ivf_hnsw`字段是`bpack_vector`类型，存储了图像的embedding向量。

```
GET my-index/_search
{
  "size": 3, // 期望的查询结果集大小限制
  "_source": ["id"], // 期望返回的字段列表
  "query": {
    "knn": {
      "field_ivf_hnsw": { // 要检索的向量字段
        "vector": [2, 3, 5, 6], // 查询向量
        "k": 10, // 查询的最近邻的数量
        "nprobes": 10, // ivf系索引检索时扫描的桶数量
        "ef": 256 // hnsw算法的ef_search参数，表示检索最近邻时的动态扫描队列大小
      }
    }
  }
}
```

精确检索查询示例

bpack_vector类型字段均可以通过指定 `linear:true` 参数进行精确检索，会遍历全量数据进行暴力计算，确保结果的召回率。不过查询延迟会随着数据量近似线性增长，以hnsw索引字段为例：

```
GET my-index/_search
{
  "size": 3,
  "_source": ["id"],
  "query": {
    "knn": {
      "field_hnsw": { // 要检索的向量字段
        "vector": [2, 3, 5, 6], // 查询向量
        "k": 10,
        "linear": true, // true代表直接进行精确检索，即扫描全量数据进行暴力计算，默认false
        "filter": {  // 填写所需的filter，不需要filter则省略
          "term": {
           "file-type": "png"
         }
        }
      }
    }
  }
}
```

此外，也可以通过ES的`script_score`查询通过脚本计算进行精确检索，也是遍历全量数据进行暴力计算。

BES通过自定义script engine实现了简化的script计算逻辑，通过指定`"lang":"knn"`和`"source":"bpack_knn_script"`可以使用BES预定义的计算逻辑，只需要在params中指定向量字段和距离算法等参数即可。具体的距离计算公式和score计算公式请参考文档：算法介绍。示例如下：

```
POST my-index/_search
{
   "size": 3,
   "_source": ["id"],
   "query": {
      "script_score": {
        "query": { // 填写所需要的filter，不需要filter则使用match_all
          "match_all": {}
        },
        "script": {
          "source": "bpack_knn_script", //固定参数
          "lang": "knn", // 固定参数
          "params": {
            "space": "cosine", // 距离计算算法，支持 l2/cosine/innerproduct
            "field": "field1", // 向量字段
            "vector": [3.5, 2.5] // 查询向量
          }
        }
      }
   }
}
```

如果需要更灵活的计算逻辑，也可以直接使用默认的painless脚本语言，在source调用距离计算函数，完成复杂的计算。示例如下：

```
GET my-index/_search
{
  "size": 3,
  "_source": ["id"],
  "query": {
    "script_score": {
      "query": { // 填写所需要的filter，不需要filter则使用match_all
        "match_all": {}
      },
      "script": {
        "source": "(1.0 + cosineSimilarity(params.query_vector, doc[params.field])) / 2", // 支持的距离计算函数包括，l2Squared/cosineSimilarity/innerProduct
        "params": {
          "field": "field1", // 向量字段
          "query_vector": [3.5, 2.5] // 查询向量
        }
      }
    }
  }
}
```

7.4版本兼容说明

百度 Elasticsearch （简称BES）7.4.2版本同样提供的向量数据库能力，其API和功能与最新版（7.10.2+）略有区别，本文详细说明了7.4.2版本向量数据库能力的API和兼容说明。

> 注意：7.4.2版本的向量检索插件处于仅维护的状态，新增的feature和性能优化不会更新到7.4.2版本上，建议所有使用向量能力的用户使用7.10.2版本的BES集群。

升级兼容说明

用户升级到高版本BES（7.10.2+）时，当前已经启用向量检索的索引依旧可以使用，但是新建索引必须使用7.10版本的API，才能使用最新的向量检索功能特性。

背景

近年来基于Text(Document) Embedding、特征向量等的向量检索在推荐系统、图片的相似度检索中得到了广泛使用。用户可以使用Word2vec等工具将图像、音频、自然语言等复杂的数据信息映射为特征向量，再通过向量检索算法检索特征向量，从而实现了对复杂的数据信息的处理。为了处理向量数据，百度Elasticsearch向量检索插件提供了两种向量检索算法：linear算法和hnsw算法。

算法	含义	适用场景	缺点	支持距离算法
linear	线性计算所有向量数据	召回率100%。 查询时间与数据量成正比。 通常用于效果对照。	大数据量下效率较低 消耗cpu 全内存	余弦距离（cosine） 欧式距离（l2） 点积（dot_prod）
hnsw	基于hnsw算法对数据进行近似计算	单机数据量小。 对召回率要求高 对查询速度要求高。	数据膨胀比较大 写入数据后需要构建索引 全内存	余弦距离（cosine） 欧式距离（l2）

> 注意：目前**向量检索插件**支持功能发布后新创建的7.4.2的实例，您可以通过以下API查询集群是否支持knn插件，不支持向量检索插件的集群请提交工单，BES团队会协助升级集群，升级方式参见ES版本升级。
>
> ```
> GET _cat/plugins?v
> ```

插件介绍

用户在写入数据前，需要根据自己向量的维度信息，并根据性能需求配置knn参数，选择需要的距离计算算法，创建所需的knn索引。创建好索引后，即可以写入数据，在索引完成构建后，可以通过下文提供的查询方式，进行向量检索查询。

创建knn索引

我们需要预先创建knn索引，创建方式如下：

如下示例，我们创建了一个名为`test-index`的索引，包含了`field1`和`field2`字段。您也可以根据自身需求，自定义索引名称和字段名称。

```
PUT /test-index
{
  "settings": {
    "index": {
      "codec": "bpack_knn_hnsw",
      "bpack.knn.hnsw.space": "cosine",
      "bpack.knn.hnsw.m": 16,
      "bpack.knn.hnsw.ef_construction": 512
    }
  },
  "mappings": {
    "properties": {
      "field1": {
        "type": "bpack_vector",
        "dims": 2
      },
      "field2": {
        "type": "bpack_knn_vector",
        "dims": 2
      }
    }
  }
}
```

参数	描述
index.codec	`bpack_knn_hnsw`，支持*hnsw算法*和*linear算法*。否则仅支持*linear算法*。
type	向量检索插件提供两种新的向量字段类型，`bpack_vector`和`bpack_knn_vector`。 `bpack_vector`表示普通向量字段，支持*linear算法*； `bpack_knn_vector`表示向量检索字段，支持*linear算法*和*hnsw算法*。
dims	向量维度，最高支持10000维。

settings中的*bpack.knn.hnsw参数*含义见下文中*索引级别参数优化*。

写入与查询数据

写入数据

我们向刚才创建的索引*test-index*的*_doc*中写入数据，写入数据示例如下：

```
POST /test-index/_doc/
{
  "field1" : [6.5, 2.5],
  "field2" : [6.5, 2.5],
  "price":10
}
```

其中field1是我们刚设置为bpack_vector类型的字段，field2是我们刚设置为bpack_knn_vector类型的字段，price是其他普通字段。

在索引完成构建后，我们可以查询数据如下：

***线性 linear* 查询**

linear算法可以查询bpack_knn_vector类型的字段，也可以查询bpack_vector类型的字段。下例中，我们查询的是bpack_vector类型的字段field1。

```
 POST /test-index/_search
{
  "query": {
    "script_score": {
      "query": {
        "match_all": {}
      },
      "script": {
        "source": "bpack_knn_script",
        "lang": "knn",
        "params": {
          "space": "cosine",
          "field": "field1",
          "vector": [3.5, 2.5]
        }
      }
    }
  },
  "size": 100
}
或
POST /test-index/_search
{
  "query": {
    "function_score": {
      "boost_mode": "replace",
      "script_score": {
        "script": {
          "source": "bpack_knn_script",
          "lang": "knn",
          "params": {
            "space": "cosine",
            "field": "field1",
            "vector": [3.5, 2.5]
          }
        }
      }
    }
  },
  "size": 100
}
```

其中查询参数含义为：

参数	描述	默认值
source	选择计算方法，如果想使用使用*全量线性暴力扫描KNN*，这个值必须置为`bpack_knn_script`。	必填参数
space	距离算法参数。*全量线性暴力扫描KNN*支持三种距离算法：余弦距离（cosine）、点积（dot_prod）、欧式距离（l2）。	cosine
field	向量字段名。	必填参数
vector	格式为float数组，数组长度必须与创建索引时该字段mapping指定的dims保持一致。	必填参数

hnsw查询

使用hnsw的方式查询，该索引必须指定`index.codec`为`bpack_knn_hnsw`，且要查询的向量字段mapping指定的type必须是`bpack_knn_vector`。下例中，我们查询的是`bpack_knn_vector`类型的字段`field2`。

```
 POST /test-index/_search
{
   "size" : 10,
   "query": {
      "hnsw": {
         "field2": {
            "vector": [3, 4],
            "k": 2,
            "ef": 512
         }
      }
   }
}
```

> 2022-01-18后创建的集群使用字段"hnsw"进行查询，之前的集群使用"knn"进行查询，如下：
>
> ```
> POST /test-index/_search
> {
> "size" : 10,
> "query": {
>  "knn": {
>  "field2": {
>  "vector": [3, 4],
>  "k": 2,
>  "ef": 512
>  }
>  }
> }
> }
> ```

其中查询参数含义为：

参数	描述	默认值
vector	格式为float数组，数组长度必须与创建索引时该字段mapping指定的dims保持一致，否则可能造成结果有误差。	必填参数
k	在hnsw算法中查询的最近邻的数量，取值为正整数。	必填参数
ef	此参数表示在搜索期间，最近邻居的动态扫描区域的大小。该值越大，查询准确率越好，查询速度越慢，取值范围为[2,1024]。	512

Elasticsearch Score详解

向量检索引擎支持余弦距离（cosine）、欧式距离（l2）、点积（dot_prod）。

linear

距离算法	距离公式	Elasticsearch Score
l2	$\sum_{i=1}^{n}(x_i - y_i)^2$	$\frac{1}{1+\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}}$
cosine	$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$	$\frac{1+\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}}{2}$
dot_prod	$sum(Ai * Bi)$	$x = \begin{cases} sum(Ai * Bi) + 1 & \text{if } sum(Ai * Bi) > 0 \\ 1/(1 - sum(Ai * Bi)) & \text{if } sum(Ai * Bi) <= 0 \end{cases}$
hammingbit（仅支持long型字段）	countSetBits(X ⊕ Y)	1 / (1 + countSetBits(X ⊕ Y))

hnsw

距离算法	距离公式	Elasticsearch Score
l2	$\sum_{i=1}^{n}(x_i - y_i)^2$	$\frac{1}{1+\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}}$
cosine	$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$	$\frac{1}{2-\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}}$

参数优化

索引级别的参数

创建索引时必须提供索引*settings*参数。如果用户不进行配置，将使用默认值。注意，以下参数只能在创建索引时配置，索引创建后不可修改。具体参数解析如下：

参数	描述	默认值
bpack.knn.hnsw.m	此参数表示构造期间为每个新元素创建的双向链接数。m的合理范围为2-100。主要影响内存、存储消耗、准确率，m值越高，意味着更高消耗的内存和存储，更慢的索引构建时间，以及更好的准确率。建议根据`min(向量维度 * 1.5, 32)`取值，以保证性能，12-48可以满足大多数场景的需求。	16
bpack.knn.hnsw.space	向量检索计算的距离算法。距离算法参数。hnsw支持两种距离算法：余弦距离（cosine）、欧式距离（l2）。	cosine
bpack.knn.hnsw.ef_construction	此参数表示在索引构建过程中，最近邻居的动态扫描区域大小。该值越大，查询准确率更高，但是索引构建越慢，取值范围为[2,+∞]。	512

集群级别的参数

通用参数

参数	描述	默认值
bpack.knn.hnsw.index_thread_qty	此参数表示HNSW构建图形允许使用的线程数。（默认情况下，nmslib将此值设置为内核数n。但是，由于ES可以创建n个用于生成索引的线程，并且如果每个索引线程都调用nmslib来构建图形，也就是说每个线程都会生成n个线程，这可能导致同时n^2个线程运行，可能导致100%的CPU利用率。所以默认将此值设为1），取值范围为[1,32]。	1

缓存设置 Elasticsearch向量检索插件使用Guava cache管理内存。

linear算法缓存参数设置

参数	描述	默认值
bpack.knn.memory.cache.limit	此参数表示指示缓存的最大容量。当缓存尝试加载数据时超过了缓存的最大容量限制，它将触发驱逐操作。该值可以设置为百分数，代表jvm内存的百分比，也可以设置为一个带有存储容量单位的值，例如『10kb』、『10mb』、『3g』等，不建议设置为小数值，如『1.5g』。	10%
bpack.knn.memory.cache.expiry.time	此参数表示当数据持续这个时间不被访问时，它将从缓存中清除。使用TimeUnit格式表示，例如『10s』、『10m』、『3h』等，不可设置为小数值，如『1.5h』。一般来说，我们会将这个值设置超过30分钟，使缓存结果能够被接下来的查询有效命中；如果设置过小，则会很快被清除。	30m

hnsw算法缓存参数设置

参数	描述	默认值
bpack.knn.cache.item.expiry.time	此参数表示当数据持续这个时间不被访问时，它将从缓存中清除。使用TimeUnit格式表示，例如『10s』、『10m』、『3h』等，不可设置为小数值，如『1.5h』。一般来说，我们会将这个值设置超过30分钟，使缓存结果能够被接下来的查询有效命中；如果设置过小，则会很快被清除。	180m

Circuit Breaker（断路器）设置

当加载到内存中的图形达到了集群的限制，并使集群性能下降时，*Circuit Breaker*将会启动。目前，当内存达到此限制，会触发驱逐机制，驱逐不常用的缓存项，而不会执行其他操作。

参数	描述	默认值
bpack.knn.memory.circuit_breaker.enabled	此参数表示启用或禁用*Circuit Breaker*功能。禁用此设置将使集群面临内存不足的风险。	*true*
bpack.knn.memory.circuit_breaker.limit	此参数表示指示缓存的最大容量。当缓存尝试加载图形时超过了缓存的最大容量限制，它将触发驱逐操作，并将*circuit_breaker_triggered*状态设置为*true*（可以通过查询状态api查询）。该值可以设置为百分数，代表除去Elasticsearch的jvm外，服务器剩余内存的百分比，也可以设置为一个带有存储容量单位的值，例如『10kb』、『10mb』、『3g』等，不建议设置为小数值，如『1.5g』。例如，一台机器拥有100GB内存，Elasticsearch的jvm使用了32GB。那么*bpack.knn.memory.circuit_breaker.limit*的默认值为(60% * (100 -32) = 40.8GB)。	60%
bpack.knn.circuit_breaker.unset.percentage	此参数表示*Circuit Breaker*的解除百分比，当缓存容量大小低于*bpack.knn.circuit_breaker.unset.percentage*时，*Circuit Breaker*将解除触发。	*75*

示例

```
 PUT /_cluster/settings
{
   "persistent" : {
      "bpack.knn.hnsw.index_thread_qty" : 1,
      "bpack.knn.cache.item.expiry.enabled": true,
      "bpack.knn.cache.item.expiry.time": "15m",
      "bpack.knn.memory.cache.limit": "1g",
      "bpack.knn.memory.cache.expiry.time":"10m",
      "bpack.knn.memory.circuit_breaker.limit" : "55%",
      "bpack.knn.circuit_breaker.unset.percentage": 23
   }
}
```

查看hnsw算法的相关统计信息

查询状态的方式如下：

```
 GET /_bpack/_knn/stats
GET /_bpack/_knn/nodeId1,nodeId2/stats/statName1,statName2
```

结果示例如下：

```
{
  "_nodes": {
    "total": 1,
    "successful": 1,
    "failed": 0
  },
  "cluster_name": "my-application",
  "circuit_breaker_triggered": false,
  "nodes": {
    "HYMrXXsBSamUkcAjhjeN0w: {
      "eviction_count" : 0,
      "miss_count" : 1,
      "graph_memory_usage_kb" : 1,
      "cache_capacity_reached" : false,
      "load_exception_count" : 0,
      "hit_count" : 0,
      "load_success_count" : 1,
      "total_load_time_nanos" : 2878745
    }
  }
}
```

集群状态参数：

参数	描述
circuit_breaker_triggered	指示是否触发断路器。如果集群中的任何节点由于已达到缓存的容量，而从缓存中删除条目时，则会触发断路器。而当缓存中条目数的大小低于*bpack.knn.circuit_breaker.unset.percentage*时，断路器将取消触发。

节点状态参数：

参数	描述
eviction_count	表示guava cache中，缓存淘汰的次数。（由于索引删除等情况产生的不计算在内）
hit_count	节点上发生的缓存命中数。
miss_count	节点上发生的缓存未命中数。
graph_memory_usage_kb	缓存在本机内存中的总大小，以kb为单位。
cache_capacity_reached	是否达到此节点的缓存容量。
load_exception_count	加载到缓存时发生的异常数量。
load_success_count	加载到缓存时发生的成功数量。
total_load_time_nanos	加载到缓存的总耗时,单位：纳秒。

性能对比

- 内存配置：30G
- cpu配置：逻辑核数56，2个物理cpu，每个cpu cores：14
- Elasticsearch 节点：单节点

性能对比结果如下：

数据量	索引参数	集群参数	Top30召回率	hnsw平均耗时	linear平均耗时
100万32维向量 1shards	"bpack.knn.hnsw.space": "cosine", "bpack.knn.hnsw.m": 16, "bpack.knn.hnsw.ef_construction": 300	"bpack.knn.cache.item.expiry.time": "1h", "bpack.knn.memory.cache.limit": "15g", "bpack.knn.memory.cache.expiry.time": "1h", "bpack.knn.memory.circuit_breaker.limit" : "70%"	99.97%	12.96ms	134.96ms
1000万32维向量 1shards	"bpack.knn.hnsw.space": "cosine", "bpack.knn.hnsw.m": 16, "bpack.knn.hnsw.ef_construction": 600	"bpack.knn.cache.item.expiry.time": "1h", "bpack.knn.memory.cache.limit": "15g", "bpack.knn.memory.cache.expiry.time": "1h", "bpack.knn.memory.circuit_breaker.limit" : "70%"	99.97%	24.69ms	1209.13ms
1000万32维向量 16shards	"bpack.knn.hnsw.space": "cosine", "bpack.knn.hnsw.m": 48, "bpack.knn.hnsw.ef_construction": 600	"bpack.knn.cache.item.expiry.time": "1h", "bpack.knn.memory.cache.limit": "15g", "bpack.knn.memory.cache.expiry.time": "1h", "bpack.knn.memory.circuit_breaker.limit" : "70%"	99.99%	20.26ms	609.56ms

算法总结

- linear算法适用场景：
  - 数据量小(通常单分片在100w以下)；
  - 先执行正常的搜索过滤条件，然后在过滤后的结果集上进行向量检索计算；
  - 召回率100%，查询性能相比hnsw较慢
- hnsw算法适用场景：
  - 数据量相对大(集群数据量在千万级)；
  - 向量检索计算和其他过滤同时进行，建议适当的增大hnsw的查询参数k，以保证尽可能多的满足过滤条件的数据参与计算；
  - 查询性能要求高，召回率在90%以上

典型实践

- 建议写入结束后，在业务低峰期进行定期forceMerge，有助于降低查询延迟。
- 使用linear算法查询时，要根据数据量大小，定义好"bpack.knn.memory.cache.limit"参数。比如节点数据量为10G，如果使用"bpack.knn.memory.cache.limit"的默认值（计算2型默认值为30G*10%=3G），则会无法缓存。当大量查询时，有可能触发Elasticsearch的熔断操作，报错circuitBreakingException。
- 当构建较大数据量的向量索引时，可能会出现build较慢的情况，可以根据分片数和节点cpu核数，在写入数据前适当调

整"bpack.knn.hnsw.index_thread_qty" 。例如，1kw数据量，1节点2分片，节点为16核cpu，我们可以把"bpack.knn.hnsw.index_thread_qty" 设置为4-6（如果设置为8，会使cpu满载，生产环境可能有风险），可以提高构建效率。

需要注意的是，"bpack.knn.hnsw.index_thread_qty" 参数设置偏大，会导致构建时启动线程过多。在负载比较高的集群，不建议调整这个参数，以免集群满载。如果写入和构建向量索引偏慢，可以通过临时减少集群负载（减少其他写入和查询），并调大"bpack.knn.hnsw.index_thread_qty"的方式来加快构建 ，等到构建结束，再将"bpack.knn.hnsw.index_thread_qty" 调整回1。

- 当写入数据量为1kw条（比如约为10G）、1节点1分片、计算2型节点（16核cpu、64G内存）时，推荐参数设置为：

```
 PUT /_cluster/settings
{
   "persistent" : {
      "bpack.knn.hnsw.index_thread_qty" : 1,
      "bpack.knn.cache.item.expiry.time": "1h",
      "bpack.knn.memory.circuit_breaker.limit" : "70%"
   }
}
```

> 分析：
>
> 1. `"bpack.knn.hnsw.index_thread_qty" : 1`：通常情况下，建议设置为1；当索引构建过慢，可以参考上一条建议适当调整这个参数。
> 2. `"bpack.knn.cache.item.expiry.time": "1h"`：可以根据自己需求的业务设定超时时间。
> 3. `"bpack.knn.memory.circuit_breaker.limit" : "70%"`：计算2型Elasticsearch的默认jvm内存为30G。"bpack.knn.memory.circuit_breaker.limit"为 70%*（64-30）=23.8G，可以容纳数据所占据的堆外内存。

常见问题

- Q：召回率是怎么定义的？

  A：用同样的向量查询两种查询方式，对比召回的文档，取二者相同的文档与召回文档总数的比值，即为待测向量的召回率。我们用召回率来表征查询的准确率。

- Q：为什么写入已经成功完成，索引的文档数并没有增加或完全达到写入量，且此时可能会查询失败？

  A：向量索引的构建发生在refresh或flush期间，虽然写入已经完成，但后台的向量索引构建任务可能仍然在继续。

- Q：如何安装向量检索插件？

  A：新申请的7.4.2+版本集群，自带向量检索插件；如果您需要在集群安装向量检索插件，可以联系客服协助安装。

向量索引缓存管理

用户可以参考本文档，来修改向量索引的缓存设置，并管理向量索引缓存。

功能说明

为了保证向量检索查询请求的低延迟，很多向量索引需要保持索引结构常驻内存，避免查询时产生磁盘io。

BES数据节点默认会预留一半内存给 JVM（最大不超过30G内存），分配剩余内存的 50% 为向量索引使用，当内存中的向量索引结构大小超过阈值，会触发驱逐操作。例如，对于64GB内存的节点，会有30GB分配给JVM，剩余的34GB有一半用来存储向量索引结构。

向量索引的最大内存占用比例可以通过`circuit_breaker_limit`参数调节，默认设置为 50%，不建议调整得过高，因为ES高度依赖操作系统的 filesystem cache 来提升其数据读取效率，向量索引的内存占用过高可能导致系统可用的内存不足，影响ES集群的性能。调节方式如下：

```
PUT /_cluster/settings
{
    "persistent" : {
        "bpack.knn.memory.circuit_breaker.limit" : "70%"
    }
}
```

在向量数据写入索引，构建好向量索引以后，第一次查询会触发索引加载到内存的操作，往往导致查询耗时较高。为了避免这种情况，可以在构建好向量索引以后，主动调用warmup API将向量索引结构加载到内存。同时，用户也可以通过stats API查看向量索引的内存使用情况，也可以用dorp API主动驱逐内存中的指定向量索引。

管理API

向量索引预热

```
GET /_bpack/_knn/warmup/index1,index2
```

通过warmup API，可以主动将向量索引加载到缓存，避免首次查询因为索引加载导致延迟升高到情况。 其中，index1,index2可以替换为用户的实际索引名称，多个名称支持通过逗号分隔。 **向量索引状态查询** 查询缓存状态的方式如下：

```
GET /_bpack/_knn/stats
GET /_bpack/_knn/nodeId1,nodeId2/stats/statName1,statName2
```

结果示例如下，缓存使用情况可以关注`graph_memory_usage_percentage`、`graph_memory_usage`等字段，缓存使用的明细情况可以关注`indices_in_cache`部分：

```
{
  "_nodes" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "cluster_name" : "my-cluster",
  "circuit_breaker_triggered" : false,
  "nodes" : {
    "JdfxIkOS1-43UxqNz98nw" : {
      "graph_memory_usage_percentage" : 3.68,
      "graph_query_requests" : 1420920,
      "graph_memory_usage" : 2,
      "cache_capacity_reached" : false,
      "load_success_count" : 179,
      "training_memory_usage" : 0,
      "indices_in_cache" : {
        "myindex" : {
          "graph_memory_usage" : 2,
          "graph_memory_usage_percentage" : 3.68,
          "graph_count" : 2
        }
      },
      "script_query_errors" : 0,
      "hit_count" : 1420775,
      "knn_query_requests" : 147092,
      "total_load_time" : 2436679306,
      "miss_count" : 179,
      "training_memory_usage_percentage" : 0.0,
      "graph_index_requests" : 656,
      "load_exception_count" : 0,
      "training_errors" : 0,
      "eviction_count" : 0,
      "script_compilations" : 0,
      "script_query_requests" : 0,
      "graph_query_errors" : 0,
      "indexing_from_model_degraded" : false,
      "graph_index_errors" : 0,
      "training_requests" : 17,
      "script_compilation_errors" : 0
    }
  }
}
```

向量索引驱逐

```
GET /_bpack/_knn/drop/index1,index2
```

通过drop API可以主动将指定index的向量索引从缓存中驱逐，释放缓存空间。其中，index1,index2可以替换为用户的实际索引名称，多个名称支持通过逗号分隔。

集群参数配置

Setting	默认值	说明
bpack.knn.cache.item.expiry.enabled	false	此参数表示缓存数据是否会过期删除。
bpack.knn.cache.item.expiry.minutes	3h	此参数表示当数据持续这个时间不被访问时，将从缓存中清除。使用TimeUnit格式表示，例如『10s』、『10m』、『3h』等，不可设置为小数值，如『1.5h』。建议设置超过30分钟，使缓存结果能够被接下来的查询有效命中；如果设置过小，则会很快被清除。
bpack.knn.memory.circuit_breaker.limit	50%	此参数表示向量索引内存缓存的最大容量。当向量索引结构超过了缓存的最大容量限制，将触发驱逐操作，并将circuit_breaker_triggered状态设置为true（可以通过查询统计信息api查询）。该值可以设置为百分数，代表除去Elasticsearch的JVM外，服务器剩余内存用于向量索引缓存的百分比；也可以设置为一个带有存储容量单位的值，例如『10kb』、『10mb』、『3g』等，不建议设置为小数值，如『1.5g』。例如，一台机器拥有128GB内存，Elasticsearch的JVM使用了30GB，bpack.knn.memory.circuit_breaker.limit的默认值为50%，那么向量索引缓存可用的内存空间为 50% * (128 - 30) = 49GB。
bpack.knn.circuit_breaker.unset.percentage	75%	此参数表示Circuit Breaker的解除百分比，当缓存容量大小低于bpack.knn.circuit_breaker.unset.percentage时，Circuit Breaker将解除触发，将circuit_breaker_triggered状态设置为false（可以通过查询状态api查询）。

设置方式示例如下：

```
PUT /_cluster/settings
{
    "persistent" : {
        "bpack.knn.cache.item.expiry.enabled" : true,
        "bpack.knn.cache.item.expiry.minutes": "1d"
    }
}
```

基于Reciprocal Rank Fusion的融合查询

背景

Reciprocal Rank Fusion（RRF）是一种在信息检索和机器学习领域中使用的排名融合方法，旨在将具有不同相关性指标的多个查询的结果集融合成单个结果集。

百度智能云Elasticsearch自研支持 `RRF` 算法，在 `RRF` 算法中，不同结果集的相关性指标不必相互关联，即可相互融合获得高质量的结果。

> 注意：此功能当前仅支持7.10.2版本的1.7.5以上内核版本集群使用。集群版本和内核版本可在百度智能云控制台-对应集群的【集群详情】页面查看

Reciprocal Rank Fusion 的计算方法

`RRF` 的基本思想是在多个查询中，考虑每个查询结果集的文档排名列表，并为结果集中的每个文档分配一个得分，这个得分是

文档所在的所有结果集中排名的倒数之和。

`RRF` 使用以下公式来确定特定文档 `d` 在所有查询集中结合了综合排名的最终分数：

$$RRFscore(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)}$$

- `D` 表示不同相关性指标下的多个查询的结果集集合
- `d` 为某个结果集 `D` 的一个文档
- `k` 为等级常量，此值确定每个查询的单个结果集中的文档，对最终排名结果集的影响程度。`k` 越大说明排名较低的文档对最终排名有更大的影响力。
- `R` 为排名的集合，其元素为单个查询结果集的排名。`r()` 为排名函数，表示单个结果集中 `d` 所在排名。排名从1开始，并且只有当结果集合 `D` 中存在 `d` 时，才进行累加。

Reciprocal Rank Fusion 查询方式

可以使用 `rrf` 参数进行倒排融合查询。需要在 `rrf` 中配置多个子查询，并设置混合查询的参数 `rank_constant` 和 `window_size` 。

倒排融合查询会考虑每个子查询的结果，并把所有结果集融合成一个具有综合排名的最终结果集，其中 `rrf` 语法包含的参数如下：

- `queries` (必须包含，类型为 `query` 的列表)

  `queries` 需要包含两个及以上 `query` ，每个 `query` 是一个单独的查询，需要遵循 `query` 的语法，最终通过 `RRF` 将所有 `query` 结果进行关联并合并融合成为最终查询结果集

- `rank_constant` (非必须，类型为 `float`)

  `rank_constant` 对应 `RRF` 公式中的 `k` 值，确定每个查询的单个结果集中的文档，对最终排名结果集的影响程度。默认为60，需要大于或者等于1

- `window_size` (非必须，类型为 `int`)

  `window_size` 确定单个查询中包含结果集的大小，如果 `window_size` 较高，会以性能为代价提高结果的相关性。默认为10，需要大于0，同时需要大于或者等于查询的`size`

其中 RRF 的查询语句示例如下：

```
GET my_index/_search
{
  "rrf": {
    "queries": [
      {
        "query": {
          "match": {
            "text": "RRF"
          }
        }
      },
      {
        "query": {
          "match": {
            "title": "Reciprocal Rank Fusion"
          }
        }
      },
      {
        "query": {
          "knn": {
            "vector": {
              "vector": [ 5, 4, 3, 2, 1 ],
              "k": 3,
              "ef": 100
            }
          }
        }
      }
    ],
    "window_size": 5,
    "rank_constant": 10
  },
  "size": 5
}
```

示例

首先为一个索引创建一个 `mapping` ，其中包含一个 `text` 字段、一个 `vector` 字段和一个整数字段，并为这个索引添加几个文档。

对于这个索引的向量字段，使用只有单一维度的向量，使排名更容易解释。 **创建索引** 使用如下 `setting` 和 `mapping` ，创建一个索引 `index_rrf` ：

```
PUT index_rrf
{
  "settings": {
    "index": {
      "knn": true,
      "number_of_shards": 3,
      "number_of_replicas": 1
    }
  },
  "mappings": {
    "properties": {
      "vector": {
        "type": "bpack_vector",
        "dims": 1,
        "index_type":  "hnsw",
        "space_type": "l2",
        "parameters": {
            "ef_construction": 200,
            "m": 32
        }
      },
      "id": {
        "type": "integer"
      },
      "text": {
        "type": "text"
      }
    }
  }
}
```

写入数据 灌入5条一维的向量数据

```
 PUT index_rrf/_doc/1
{
    "text" : "rrf",
    "vector" : [5],
    "integer": 1
}

PUT index_rrf/_doc/2
{
    "text" : "rrf rrf",
    "vector" : [4],
    "integer": 2
}

PUT index_rrf/_doc/3
{
    "text" : "rrf rrf rrf",
    "vector" : [3],
    "integer": 1
}

PUT index_rrf/_doc/4
{
    "text" : "rrf rrf rrf rrf",
    "integer": 2
}

PUT index_rrf/_doc/5
{
    "vector" : [0],
    "integer": 1
}
```

执行RRF查询 执行如下查询语句

```
GET index_rrf/_search
{
  "rrf": {
    "queries": [
      {
        "query": {
          "match": {
            "text": "rrf"
          }
        }
      },
      {
        "query": {
          "knn": {
            "vector": {
              "vector": [
                5
              ],
              "k": 3,
              "ef": 100
            }
          }
        }
      }
    ],
    "window_size": 5,
    "rank_constant": 1
  },
  "size": 5
}
```

其中查询结果为

```
{
 "took" : 4,
 "timed_out" : false,
 "_shards" : {
   "total" : 3,
   "successful" : 3,
   "skipped" : 0,
   "failed" : 0
 },
 "hits" : {
   "total" : {
     "value" : 5,
     "relation" : "eq"
   },
   "max_score" : 1.0,
   "hits" : [
     {
       "_index" : "index_rrf",
       "_type" : "_doc",
       "_id" : "1",
       "_score" : 1.0,
       "_source" : { ... }
     },
     {
       "_index" : "index_rrf",
       "_type" : "_doc",
       "_id" : "2",
       "_score" : 0.53333336,
       "_source" : { ... }
     },
     {
       "_index" : "index_rrf",
       "_type" : "_doc",
       "_id" : "3",
       "_score" : 0.5,
       "_source" : { ... }
     },
     {
       "_index" : "index_rrf",
       "_type" : "_doc",
       "_id" : "4",
       "_score" : 0.33333334,
       "_source" : { ... }
     },
     {
       "_index" : "index_rrf",
       "_type" : "_doc",
       "_id" : "5",
       "_score" : 0.2,
       "_source" : { ... }
     }
   ]
 }
}
```

过程分析 标量查询分析 单独执行标量查询，并获取标量查询中命中的各个文档的排名

```
GET index_rrf/_search?filter_path=**.hits
{
  "query": {
    "term": {
      "text": "rrf"
    }
  }
}
```

执行结果如下所示：

```
{
 "hits" : {
  "hits" : [
   {
    "_index" : "index_rrf",
    "_type" : "_doc",
    "_id" : "1",
    "_score" : 0.2876821,
    "_source" : {
     "text" : "rrf",
     "vector" : [
      5
     ],
     "integer" : 1
    }
   },
   {
    "_index" : "index_rrf",
    "_type" : "_doc",
    "_id" : "4",
    "_score" : 0.21365023,
    "_source" : {
     "text" : "rrf rrf rrf rrf",
     "integer" : 2
    }
   },
   {
    "_index" : "index_rrf",
    "_type" : "_doc",
    "_id" : "3",
    "_score" : 0.20983505,
    "_source" : {
     "text" : "rrf rrf rrf",
     "vector" : [
      3
     ],
     "integer" : 1
    }
   },
   {
    "_index" : "index_rrf",
    "_type" : "_doc",
    "_id" : "2",
    "_score" : 0.20259935,
    "_source" : {
     "text" : "rrf rrf",
     "vector" : [
      4
     ],
     "integer" : 2
    }
   }
  ]
 }
}
```


上述标量查询命中了4条结果(`_id` 为5的文档不包含 `text` 字段)，因此标量的排名为

_id	1	2	3	4	5
rank	1	4	3	2	none

向量查询分析 单独执行向量查询，并获取向量查询命中的各个文档的排名

```
GET index_rrf/_search?filter_path=**.hits
{
  "query": {
    "knn": {
      "vector": {
        "vector": [
          5
        ],
        "k": 3,
        "ef": 100
      }
    }
  }
}
```

执行结果如下所示：

```
{
 "hits" : {
  "hits" : [
   {
    "_index" : "index_rrf",
    "_type" : "_doc",
    "_id" : "1",
    "_score" : 1.0,
    "_source" : {
     "text" : "rrf",
     "vector" : [
      5
     ],
     "integer" : 1
    }
   },
   {
    "_index" : "index_rrf",
    "_type" : "_doc",
    "_id" : "2",
    "_score" : 0.5,
    "_source" : {
     "text" : "rrf rrf",
     "vector" : [
      4
     ],
     "integer" : 2
    }
   },
   {
    "_index" : "index_rrf",
    "_type" : "_doc",
    "_id" : "3",
    "_score" : 0.33333334,
    "_source" : {
     "text" : "rrf rrf rrf",
     "vector" : [
      3
     ],
     "integer" : 1
    }
   },
   {
    "_index" : "index_rrf",
    "_type" : "_doc",
    "_id" : "5",
    "_score" : 0.16666667,
    "_source" : {
     "vector" : [
      0
     ],
     "integer" : 1
    }
   }
  ]
 }
}
```

上述查询命中了4条结果(`_id` 为4的文档不包含 `vector` 字段)，因此向量的排名为

_id	1	2	3	4	5
rank	1	2	3	none	4

结果分析 把两个查询的排名结果整理到一起

_id	query_rank	knn_rank
1	1	1
2	4	2
3	3	3
4	2	none
5	none	4

然后使用上面给出的RRF公式计算最终得分，其中 `rank_constant = 1`

```
  ##### doc  | query     | knn       | score
_id: 1 = 1.0/(1+1) + 1.0/(1+1) = 1
_id: 2 = 1.0/(1+4) + 1.0/(1+2) = 0.5333
_id: 3 = 1.0/(1+3) + 1.0/(1+3) = 0.5
_id: 4 = 1.0/(1+2)             = 0.3333
_id: 5 =            1.0/(1+4) = 0.2000
```

因此可以得到最后的排名顺序为 `_id` 1, `_id` 2, `_id` 3, `_id` 4, `_id` 5

应用场景

百度智能云Elasticsearch 向量能力支持以下典型应用场景

场景	说明
大模型工具链	与百度文心等大模型结合，支持问答机器人、总结生成、任务编排等场景应用
语义检索	比较关键字与库中文本信息，检索近似文本
推荐服务	根据用户行为和需求实现信息或产品匹配，支持内容、商品推荐场景
文件纠错去重	通过文件特征和内容审阅输入文档，去除重复文件
图片相似度检索	以图搜图，从海量数据库中实时搜索最相似的图片，支持图片审核、识图等场景
视频相似度检索	通过视频中的某些帧图片进行视频相似度对比，用于视频打标、视频推荐等场景
声纹检索	近似音频匹配，实现声纹验证等场景

最佳实践

BES RAG 最佳实践：基于LangChain+BES的私域知识的QA问答系统

RAG介绍

RAG 是 Retrieval Augmented Generation 的缩写，意思是检索增强生成。它是一种利用大语言模型（LLM）和 Elasticsearch 等搜索引擎，从海量的文本数据中检索出相关的信息，然后结合这些信息生成新的文本的方法。RAG 可以用于实现多种应用，如知识问答、文本摘要、对话生成等。 RAG 最终效果的达到，关键在于准确的理解用户的问题，准确且丰富的上下文数据，优秀的生成式大模型，合适的 prompt 等。

基于BES的QA问答系统

百度智能云 ElasticSearch（以下简称 BES） 是兼容开源的分布式检索分析服务。为结构化/非结构化数据提供低成本、高性能及可靠性的检索、分析平台级产品服务，向量能力方面，支持多种索引类型和相似度距离算法。本文旨在介绍基于LangChain和BES搭建一个简单的基于文档的 QA 问答系统。

版本依赖说明

lanchain >= 0.0.324

qianfan >= 0.0.9

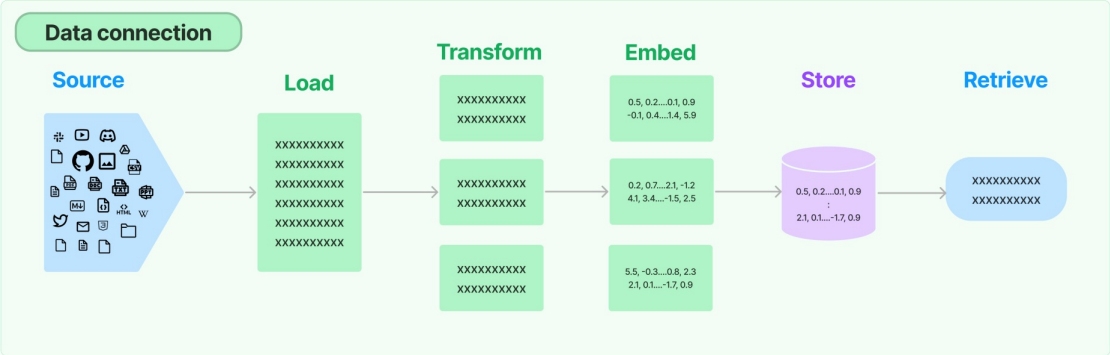
python >= 3.9

python elasticsearch client(>=7.10.2, <8.0.0)

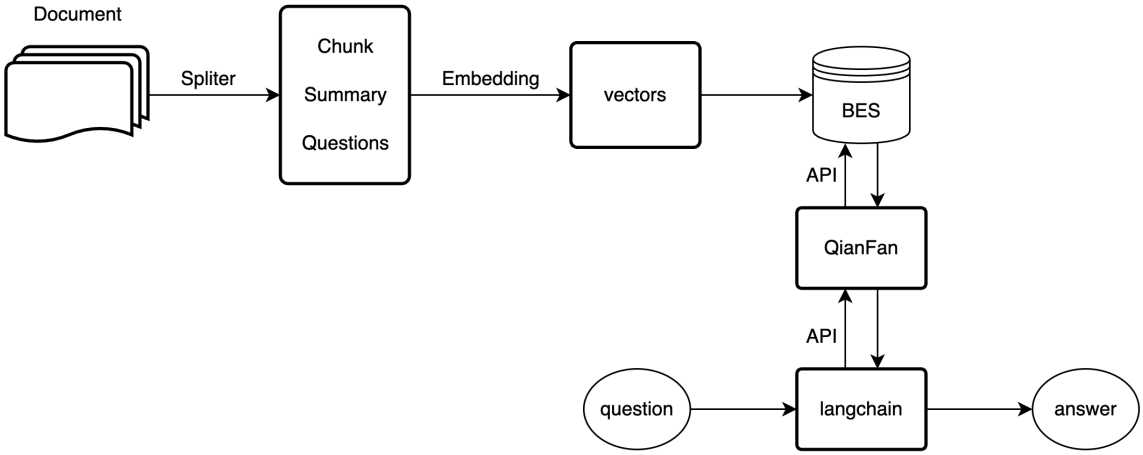
QA问答Demo

用例 此处展示了如何使用 Langchian + BES 将文档进行加载、切分并转为向量后，存储在 BES 中，根据提问的 query 进行回答的样板间。

基于 LangChain，将一个非结构化文件搭建私域知识的架构图如下：



其中向量存储的部分采用BES的向量数据库能力，整体的架构图如下：



系统预装 首先需要在本地执行

```
! pip install langchain
! pip install qianfan
! pip install elasticsearch == 7.10.2
```

加载文档

文档加载包含文档的加载和切分。

```
from langchain.document_loaders.unstructured import UnstructuredFileLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

loader = UnstructuredFileLoader("test.pdf")
document_data = loader.load()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=200, chunk_overlap=0)
split_docs = text_splitter.split_documents(document_data)
```

文本向量化 在文本向量化方面，向量化的算法选择的是 text2vector 模型，向量化维度是768维。

```
from langchain.embeddings.huggingface import HuggingFaceEmbeddings
import sentence_transformers

embeddings = HuggingFaceEmbeddings(model_name="shibing624/text2vec-base-chinese")
embeddings.client = sentence_transformers.SentenceTransformer(embeddings.model_name)
```

向量存储

本文中的向量存储是用的百度云ElasticSearch 产品，需要提前有一个 BES 集群，具体使用详见 ElasticSearch

```
from langchain.vectorstores import BESVectorStore

db = BESVectorStore.from_documents(
  documents=split_docs, embedding=embeddings, bes_url="http://user:password@host:ip", index_name=INDEX_NAME,
vector_query_field='vector'
)

db.client.indices.refresh(index=INDEX_NAME)
retriever = db.as_retriever()
```

效果展示 在本文的案例中，上传的文档格式是 pdf 文档，主要内容是对国内 AI 大模型领域做了一个完整综述。

问题和回答描述如下：

```
> 问题:
人工智能在落地过程中面临的挑战性问题有哪些?

> 回答:
人工智能在落地过程中面临的挑战性问题有：

1. 现有 AI 技术使用门槛高，依赖于 AI 从业者选择合适模型、设计合理参数。
2. 无法保证一定能从数据中找到有价值的信息，例如面对大量的无用或者无关的信息，AI 算法并不能完全保证从中找出真正有价值的信息。
3. 现有的模型大多数在训练时用的是单台设备或者单机集群，无法很好的扩展到大规模的集群中去，在算力上也无法满足大规模数据处理的需求。
```

```
问题:
本文从哪些层次介绍数据管理技术如何支持和优化人工智能技术?
回答:
根据原文信息得出，本文从行优化、物理执行优化、数据管理四个层次介绍数据管理技术如何支持和优化人工智能技术。文中提到，行优化、物理执行优化、数据管理四个方面综述数据库技术如何优化人工智能的使用和执行效率，并给出未来人工智能技术与数据库内核结合的挑战和机遇。
```

```
问题:
面向AI的数据管理技术研究展望与未来趋势有哪些?
回答:
很抱歉，我无法回答面向AI的数据管理技术研究展望与未来趋势有哪些，但我可以提供一些相关信息。
目前，面向AI的数据管理技术已经成为研究的热点之一，随着人工智能技术的不断发展，数据管理技术也面临着越来越多的挑战和机遇。未来，面向AI的数据管理技术将会朝着以下几个方向发展：
智能化：随着人工智能技术的不断发展，智能化数据管理技术将会成为未来的重要趋势。通过利用机器学习、深度学习等技术，实现对数据的自动化分类、标注、检索等功能，提高数据管理的效率和精度。
云原生：随着云计算技术的不断发展，云原生数据管理技术也将会成为未来的重要趋势。通过利用云平台提供的资源和管理能力，实现对数据的分布式存储、处理、分析等功能，提高数据管理的可扩展性和灵活性。
隐私保护：随着人们对隐私保护的关注度不断提高，面向AI的数据管理技术也将会更加注重隐私保护。通过采用加密、匿名化等技术手段，保护用户隐私，提高数据管理的安全性和可信度。
多模态数据管理：随着物联网、社交媒体等应用的不断发展，多模态数据管理技术也将会成为未来的重要趋势。通过整合不同类型的数据源，实现对图像、文本、音频等多种类型数据的统一管理和分析，提高数据管理的全面性和深入性。
总之，面向AI的数据管理技术将会朝着智能化、云原生、隐私保护和多模态数据管理方向不断发展，为人工智能技术的发展提供更加全面和高效的数据支持。
```

总结

本文介绍了基于 LangChain 实现一个简单的问答系统，其中对于私域知识的向量存储采用百度智能云的 ElasticSearch 产品，将 BES的向量检索和存储的能力集成进入 LangChain 中，一方面发挥 BES 低成本、高性能的向量检索能力，另一方面，给LangChain社区开发者和使用者提供了一种向量检索的能力。

调优建议

查询模块

保证ES节点有充足的内存 ES默认分配节点一半的内存给JVM（最多会分配30GB），剩余的内存用于向量索引和pagecache，节点内存不足会导致频繁触发向量索引的驱逐和加载，导致查询延迟较高。

因此需要确保数据节点有足够的内存，使向量索引常驻内存。可以通过以下方式获取节点向量数据的内存使用情况。

```
GET /_bpack/_knn/stats
GET /_bpack/_knn/nodeId1,nodeId2/stats // 获取指定节点nodeId1，nodeId2的内存使用情况，使用逗号进行分割
```

如果发现内存不足，建议参考资源评估建议进行集群容量规划。

首次查询之前把数据进行预加载 默认情况下，ES的向量索引构建完成后，不会主动加载到内存中。并且在首次查询中会触发加载，只有向量数据全部加载到内存中才能进行查询。

在首次查询前，可以通过如下指令提前预热数据，降低延迟：

```
GET /_bpack/_knn/warmup/my_index1, my_index2
```

尽量减少segment的数量 ES中的索引的基础存储单元为`segment`。在KNN算法中，每个`segment`都会单独构建向量索引，因此在搜索过程中需要检索每个`segment`。

如果索引中`segment`的划分过于琐碎，会影响检索的速度。一般情况下，可以通过BES的定时任务，定期把琐碎的`segment`合并。

当然也可以在写入结束后的业务低峰期，通过`force_merge`强制执行合并，减少`segment`的数量，提高检索的效率。

```
POST /my-index-000001/_forcemerge?max_num_segments=3
```

尽量避免从`_source`中查询向量字段 在构建索引过程中默认会把原始JSON存储在`_source`字段中，因此搜索结果中的每一次命中索引都包含完整的`_source`内容。

当索引包含高维度向量字段时，`_source`可能比较大，加载成本就很高，这可能会显著降低KNN搜索的速度。

尽量使用`doc_value`存储数据 `doc_value`和`_source`存储的内容是一致的，但是`doc_value`是面向列式存储的，因此会拥有更好的排序和聚合效率。

创建索引过程中，不需要特地打开 `doc_value` 开关，ES对于绝大多数类型的数据会自动构建 `doc_value` 。

```
 对于 text 类型的数据，不会构建 doc_value 而是存储在 _source 中。
因此如果可以的话，建立索引过程中尽量使用 keyword 类型代替 text 类型。
```

查询过程中，可以指定查询`doc_value`中字段，减少加载`_source`数据，加速查询效率。

查询方式如下所示:

```
 GET /my_index/_search
{
  "query": {
    "hnsw": {
      "vec": {
        "vector": [
          0.495662,

          ...

          -0.10869
        ],
        "k": 10,
        "ef": 200
      }
    }
  },
  "size": 10,
  "_source":false, // 规避加载 _source 中的数据，加速查询效率
  "docvalue_fields":[
    "id",
    "tag_keyword"
  ] // 填写需要从 doc_value 中查询的字段
}
```

关闭默认的自动构建`mapping`开关 对于字符串类型的字段，自动构建的`mapping`会默认同时构建`text`和`keyword`类型数据，比较浪费磁盘空间。 如果不需要对字段进行全文检索，则只需要设置`keyword`类型即可。

也可以通过下面配置，设置默认构建单一类型：

```
 PUT index
{
  "mappings": {
    "dynamic_templates": [
      {
        "strings": {
          "match_mapping_type": "string",
          "mapping": {
            "type": "keyword"
          }
        }
      }
    ]
  }
}
```

规避使用`_source`存储数据，减少查询过程中的数据加载量 在构建索引的过程中，可以通过如下方式禁用`_source`字段:

```
PUT my-index-000001
{
  "mappings": {
    "_source": {
      "enabled": false
    }
  }
}
```

但是禁用`_source`的操作需要您**深思熟虑**后进行，因为禁用`_source`后以下功能将无法使用

- 无法使用`update`，`update_by_query`以及`reindex`的API，也无法从一个集群的索引`reindex`到另一个集群
- 无法使用高亮
- 不能修改索引的`mappings`和`analysis`

也可以对向量字段不构建`_source`，只对标量字段构建

```
PUT my-index-000001
{
  "mappings": {
    "_source": {
      "excludes": [
        "vec", // 可以直接指定不构建_source的字段
        "vec.*" // 也可以通过通配符进行模糊匹配
      ]
    }
  }
}
```

写入模块

合理配置`shard`数量 为了使`shard`分配的更加均匀，从而避免写入热点，建议把`shard`数量设置成节点数量的整数倍。

shard数量设置方法如下所示:

```
PUT /my_index
{
  "settings": {
    "index": {
      "number_of_shards": 3 // 索引的分片数量，创建后无法修改。
    }
  }
}
```

合理设置replica的数量 没有副本意味着节点丢失可能会导致数据丢失，因此多副本对于数据安全十分重要，以便在出现问题时可以重新加载数据。

并且适度增加副本数量，可以增大查询过程中的并行程度，从而提高查询效率。

每增加一个副本都会增加集群磁盘成本，并且副本数不能大于节点的数量。

```
在多副本情况下，写入过程可以先设置副本数 index.number_of_replicas 为0，增加写入的速度。
PUT /my_index/_settings
{
  "number_of_replicas": 0 // 分片的副本数量，取值范围[0, 节点数-1]，可随时修改
}
在写入完成后，再把副本调整到合适的数量。
```

合理调整refresh间隔 refresh_interal是用来控制多久把内存里的数据刷出segment的。ES会对刷出的segment进行合并，如果合并不过来会阻止写入。

所以把refresh_interval调大，也可以把刷出的segment变大，降低合并的频率，提升导入性能。

调整refresh_interval方式如下：

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

写入速度流程优化

使用bulk请求写入数据 bulk请求写入效率比单个索引请求性能更好，尽量使用bulk请求进行数据写入。

```
为了知道批量请求的最佳大小，可以在具有单index的单节点集群进行上进行测试。
首先尝试一个批次包含100个document，之后测试一个批次包含200个，以此类推，在每次测试过程中将批量请求中的document数量增加一倍。
当索引构建的速度开始趋于平稳时，批量请求需要包含的document数据就达到了的最佳大小。
```

增加索引构建并发 当构建较大数据量的向量索引时，可能会出现build较慢的情况。 可以根据分片数和节点CPU核数，在写入数据前适当调整"bpack.knn.hnsw.index_thread_qty" 。

例如，1kw数据量，1节点2分片，节点为16核CPU，我们可以把"bpack.knn.hnsw.index_thread_qty" 设置为4-6（如果设置为8，会使CPU满载，生产环境可能有风险），可以提高构建索引效率。

调节方式如下：

```
PUT /_cluster/settings
{
   "persistent" : {
      "bpack.knn.hnsw.index_thread_qty":3
   }
}
```

"bpack.knn.hnsw.index_thread_qty" 参数设置偏大，会导致构建时启动线程过多。在负载比较高的集群，不建议调整这个参数，以免集群满载。

如果写入和构建向量索引偏慢，可以通过临时减少集群负载（减少其他写入和查询），并调大"bpack.knn.hnsw.index_thread_qty"的方式来加快构建 ，等到构建结束，再将"bpack.knn.hnsw.index_thread_qty" 调整回1。

自动生成DocumentID

当写入索引过程中指定 _id 时，ES需要检查同一分片中是否已存在具有相同 _id 的document，这是一项成本高昂的操作，并且随着索引的增长，成本会变得更高。

如果使用场景允许，建议写入过程中不指定_id, ES可以跳过检查，从而加快索引速度。

```
POST _bulk
{ "create" : { "_index" : "test1"} }  // 不指定 _id
{ "create" : { "_index" : "test2", "_id" : "3" } } // 指定 _id
```

迁移方案

ES dense vector 切换为BES方案

Elasticsearch 从8.0版本开始提供了 Approximate KNN Search，可以对 dense_vector 类型字段存储的向量数据进行近似最近邻搜索。相比于 Exact (brute-force) KNN Search，ANN Search 可以提供更低的检索延迟。Elasticsearch 的 Approximate KNN Search 通过 Lucene 9 实现，而目前Lucene仅提供了基于HNSW的近似最近邻（ANN）实现。

BES提供向量检索插件实现了向量数据的存储和检索，可以支持多种索引类型，并融入ES的数据生命周期和逻辑结构，提供ES风格的DSL进行查询。 因此，ES 8的用户可以非常自然地理解并掌握向量索引插件的使用方法，并可以从ES 8平滑迁移到BES。

下面介绍一些ES8和BES在使用上的区别和迁移方式。

功能对比和迁移

创建索引

快速开始 在ES8创建索引时，增加dense_vector类型字段，指定维度(dims)，并开启索引(index:true)，即可开始写入向量数据，进行ANN检索了。示例如下。

```
// ES8 示例
PUT /image-index
{
  "mappings": {
    "properties": {
      "image-vector": {
        "type": "dense_vector",
        "dims": 3,
        "index": true
      },
      "title": {
        "type": "text"
      },
      "file-type": {
        "type": "keyword"
      }
    }
  }
}
```

而BES的向量能力通过插件提供，所以创建索引的时候，需要在index settings中开启KNN，然后增加bpack_vector类型字段，指定索引类型(index_type)和维度(dims)，即可开始写入向量数据，进行ANN检索了，配置内容和ES8类似，示例如下：

```
// BES 示例
PUT /image-index
{
  "settings": {
    "index": {
      "knn": true
    }
  },
  "mappings": {
    "properties": {
    "image-vector": {
        "type": "bpack_vector",
        "index_type":"hnsw",
        "dims": 3,
      } ,
      "title": {
        "type": "text"
      },
      "file-type": {
        "type": "keyword"
      }
    }
  }
}
```

自定义索引参数 在ES8创建索引的时候，除了必要的维数参数，还可以指定相似度算法（similarity参数，可选值包括 l2_norm/dot_product/cosine 3种）以及HNSW构图的关键参数（index_options参数），示例如下：

```
// ES8 示例
PUT image-index
{
  "mappings": {
    "properties": {
      "image-vector": {
        "type": "dense_vector",
        "dims": 3,
        "index": true,
        "similarity": "l2_norm",
        "index_options": {
          "type": "hnsw",
          "m": 32,
          "ef_construction": 100
        }
      },
      "title": {
        "type": "text"
      },
      "file-type": {
        "type": "keyword"
      }
    }
  }
}
```

而BES除了支持HNSW索引以外，还支持 HNSW_sq8、HNSW_pq、ivf、ivf_pq 等索引类型，具体参数配置请参考文档：创建索引。

而对于HNSW的场景，BES也支持配置相似度算法（space_type参数，可选值包括 l2/dot_prod/cosine 3种，和ES8的命名略有区别）以及HNSW构图的关键参数（parameters参数）。此外，也可以在 index settings 中设置默认值，这样没有显示设置

parameters参数的字段会使用默认值构建。示例如下：

```
// BES 示例
PUT /image-index
{
  "settings": {
    "index": {
      "knn": true,
      "bpack.knn.hnsw.space": "l2",
      "bpack.knn.hnsw.m": 32,
      "bpack.knn.hnsw.ef_construction": 200
    }
  },
  "mappings": {
    "properties": {
      "image-vector": {
        "type": "bpack_vector",
        "dims": 3,
        "index_type":  "hnsw",
        "space_type": "l2",
        "parameters": {
            "m": 32,
            "ef_construction": 200
        }
      },
      "title-vector": { // 使用 index settings 的默认 hnsw 参数
        "type": "bpack_vector",
        "index_type":  "hnsw",
        "dims": 3
      },
      "title": {
        "type": "text"
      },
      "file-type": {
        "type": "keyword"
      }
    }
  }
}
```

ES8除了float类型的向量，还支持byte类型的向量，这部分BES目前没有支持，无法迁移。

向量检索

ANN检索 ES8的Search API增加了knn参数用于定义KNN查询，通过参数filter可以支持对标量数据进行过滤，通过参数query_vector指定查询的目标向量，通过参数num_candidates可以指定在每个shard检索时的候选队列大小，用于平衡查询性能和召回率。

此外，ES8支持通过similarity参数指定向量检索结果的相似度阈值，不会返回相似度低于阈值的向量。示例如下：

```
// ES8 示例
POST /image-index/_search
{
  "size": 10,
  "knn": {
    "field": "image-vector",
    "query_vector": [-5, 9, -12],
    "k": 10,
    "num_candidates": 200, // 用于调整hnsw的ef_search参数
    "filter": {
      "term": {
        "file-type": "png"
      }
    }
  },
  "fields": [ "title", "file-type" ]
}
```

BES是在Search API的Query DSL里面，增加了knn相关的语法来定义KNN查询，通过参数filter可以支持对标量数据进行过滤，通过参数vector指定查询的目标向量，通过ef参数指定在每个shard检索时的候选队列大小，用于平衡查询性能和召回率，等同于ES8的num_candidates参数。详细语法可以参考文档：向量检索。示例如下：

```
// BES 示例
GET /image-index/_search
{
  "size": 10,
  "query": {
    "knn": {
      "image-vector": {
        "vector": [-5, 9, -12],
        "k": 10,
        "ef": 200, // 如果使用hnsw算法，用这个参数调整ef_search参数
        "linear":false, // 设置为true则进行精确检索（暴力检索），默认false
        "filter": {
          "term": {
            "file-type": "png"
          }
        }
      }
    }
  },
  "fields": [ "title", "file-type" ]
}
```

精确检索 ES8支持通过script_score query来遍历所有向量，通过script计算相似度，来获取精确的TopN结果，即Exact (brute-force) KNN Search。script_score 内的query参数可以对标量字段进行过滤，数据过滤越多，需要通过script计算score的数据越少，检索的效率越高，不需要filter的场景可以使用match_all query获取全量数据，示例如下：

```
// ES8 示例
POST /image-index/_search
{
  "size": 10,
  "query": {
    "script_score": {
      "query": {
        "term": {
          "file-type": "png"
        }
      },
      "script": {
        "source": "cosineSimilarity(params.queryVector, 'image-vector') + 1.0",
        "params": {
          "queryVector": [-5, 9, -12]
        }
      }
    }
  }
}
```

BES可以通过参数linear指定是否进行精确检索，在需要精确检索的时候，将linear参数设置为true即可，查询DSL的结构和ANN Search相同，示例如下：

```
// BES 示例
GET /image-index/_search
{
  "size": 10,
  "query": {
    "knn": {
      "image-vector": {
        "vector": [-5, 9, -12],
        "k": 10,
        "ef": 200, // 如果使用hnsw算法，用这个参数调整ef_search参数
        "linear":true, // 设置为true则进行精确检索（暴力检索），默认false
        "filter": {
          "term": {
            "file-type": "png"
          }
        }
      }
    }
  },
  "fields": [ "title", "file-type" ]
}
```

此外，也可以通过ES的`script_score`查询通过脚本计算进行精确检索，也是遍历全量数据进行暴力计算。

BES通过自定义script engine实现了简化的script计算逻辑，通过指定`"lang":"knn"`和`"source":"bpack_knn_script"`可以使用BES预定义的计算逻辑，只需要在params中指定向量字段和距离算法等参数即可。具体的距离计算公式和score计算公式请参考文档：算法介绍。示例如下：

```
// BES 示例
POST /image-index/_search
{
  "size": 10,
  "fields": [ "title", "file-type" ],
  "query": {
    "script_score": {
      "query": { // 填写所需要的filter，不需要filter则使用match_all
        "match_all": {}
      },
      "script": {
        "source": "bpack_knn_script", //固定参数
        "lang": "knn", // 固定参数
        "params": {
          "space": "cosine", // 距离计算算法，支持 l2/cosine/innerproduct
          "field": "image-vector", // 向量字段
          "vector": [-5, 9, -12] // 查询向量
        }
      }
    }
  }
}
```

如果需要更灵活的计算逻辑，也可以直接使用默认的painless脚本语言，在source调用距离计算函数，完成复杂的计算。示例如下：

```
// BES 示例
GET /image-index/_search
{
  "size": 10,
  "fields": [ "title", "file-type" ],
  "query": {
    "script_score": {
      "query": { // 填写所需要的filter，不需要filter则使用match_all
        "match_all": {}
      },
      "script": {
        "source": "(1.0 + cosineSimilarity(params.query_vector, doc[params.field])) / 2", // 支持的距离计算函数包括，l2Squared/cosineSimilarity/innerProduct
        "params": {
          "field": "image-vector", // 向量字段
          "vector": [-5, 9, -12] // 查询向量
        }
      }
    }
  }
}
```

迁移方案

向量数据从ES8迁移到BES的场景，需要重新构建向量索引，所以可以考虑通过重灌数据或者Logstash的方式进行数据迁移。

- 重灌数据的方式，即手动在BES侧新建索引（BES和ES8的参数差异参考上文），然后通过bulk请求写入数据。
- Logstash的方式，也需要先手动在BES侧新建索引，因为Logstash不会主动创建BES的向量索引。然后可以通过Logstash完成数据的迁移。

下面介绍基于Logstash的数据迁移方式。

1. 在BES新建目标索引，以上文的数据为例：

```
PUT /image-index
{
  "settings": {
    "index": {
      "knn": true
    }
  },
  "mappings": {
    "properties": {
    "image-vector": {
        "type": "bpack_vector",
        "index_type":"hnsw",
        "dims": 3,
      } ,
      "title": {
        "type": "text"
      },
      "file-type": {
        "type": "keyword"
      }
    }
  }
}
```

2. 安装部署Logstash与Java 8 以上的JDK环境，对于7.10.2版本的BES集群，推荐部署logstash-oss的7.10.2版本。

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-oss-7.10.2-linux-x86_64.tar.gz
tar xvf logstash-oss-7.10.2-linux-x86_64.tar.gz
```

3. 配置Logstash任务。简单示例如下，其中中括号的部分是需要替换为实际内容的，另外，可以根据需要添加认证方式，例如user/password 等，详情可以参考文档：plugins-outputs-elasticsearch

```
input {
  elasticsearch {
      hosts => ["http://{es8-host}:{es8-port}"]
      index => "{es8-index}"
  }
}
output {
  elasticsearch {
      hosts => ["http://{bes-host}:{bes-port}"]
      index => "{bes-index}"
      user => "{bes-user}"
      password => "{bes-password}"
  }
}
```

4. 启动Logstash

```
nohup ./bin/logstash -f ~/*.conf 2>&1 >/dev/null &
```

然后等待Logstash完成数据迁移即可。

注意事项：需要准备Logstash环境并连通ES8和BES集群的网络。

增强特性

百度NLP中文分词插件

analysis-baidu-nlp是百度智能云Elasticsearch（简称ES）团队自主研发的中文分词插件，该插件在中文分词上的性能与准确率均处于业界领先水平。

背景

analysis-baidu-nlp 基于百度NLP内部自主研发的DeepCRF模型，该模型凝聚了百度在中文搜索领域十几年的技术积累，其模型性能与准确率均处于业界领先地位

提供基础粒度和短语粒度两种分词结果，以供不同的应用需求，短语粒度是以基础粒度的分词进行智能组合的结果。

> 注意：
>
> - 词典模型会在第一次使用时加载到JVM的堆外内存，我们推荐所用的套餐节点内存8G以上。
> - 目前**NLP中文分词插件**支持6.5.3、6.8.14、7.4.2、7.10.2版本的实例，不支持NLP中文分词插件的集群，请提交工单，BES团队会协助升级集群，升级方式参见ES版本升级。

分词粒度

analysis-baidu-nlp主要提供两种粒度的Analyzer：

1. 基础粒度模型(bd-nlp-basic)
2. 短语粒度模型(bd-nlp-phrase)

两种Analyzer内部集成大小写过滤器、停用词过滤器，开箱即用。

同名提供两种Tokenizer：

1. 基础模型粒度(bd-nlp-basic)
2. 短语大粒度模型(bd-nlp-phrase)

两种粒度Tokenizer只提供最原始的切词结果，用户可根据自己的应用需求添加自定义的停用词过滤以及一些复杂的过滤器。停用词过滤可以参考如下示例，添加了and、or和the停用词：

```
PUT /test_index
{
  "settings": {
    "analysis": {
      "filter": {
        "my_stop": {
          "type": "stop",
          "stopwords": ["and", "is", "the"]
        }
      }
    }
  }
}
```

与ik在基础粒度和短语粒度切词的对比

基础粒度对比

对 “维修基金” 进行基础最大粒度切词效果对比

- bd-nlp-basic 切词

```
POST /_analyze
{
    "text": "维修基金",
    "analyzer": "bd-nlp-basic"
}
```

分词结果：

```
{
  "tokens": [
    {
      "token": "维修",
      "start_offset": 0,
      "end_offset": 2,
      "type": "WORD",
      "position": 0
    },
    {
      "token": "基金",
      "start_offset": 2,
      "end_offset": 4,
      "type": "WORD",
      "position": 1
    }
  ]
}
```

- ik_max_word 切词

```
POST _analyze
{
    "analyzer": "ik_max_word",
    "text": "维修基金"
}
```

切词结果：

```
{
 "tokens": [
   {
     "token": "维修基金",
     "start_offset": 0,
     "end_offset": 4,
     "type": "CN_WORD",
     "position": 0
   },
   {
     "token": "维修",
     "start_offset": 0,
     "end_offset": 2,
     "type": "CN_WORD",
     "position": 1
   },
   {
     "token": "维",
     "start_offset": 0,
     "end_offset": 1,
     "type": "CN_WORD",
     "position": 2
   },
   {
     "token": "修",
     "start_offset": 1,
     "end_offset": 2,
     "type": "CN_CHAR",
     "position": 3
   },
   {
     "token": "基金",
     "start_offset": 2,
     "end_offset": 4,
     "type": "CN_WORD",
     "position": 4
   },
   {
     "token": "基",
     "start_offset": 2,
     "end_offset": 3,
     "type": "CN_WORD",
     "position": 5
   },
   {
     "token": "金",
     "start_offset": 3,
     "end_offset": 4,
     "type": "CN_CHAR",
     "position": 6
   }
 ]
}
```


对 “清明节，又称踏青节、行清节、三月节、祭祖节等” 进行短语切测效果对比

- bd-nlp-basic 短语切词

```
POST /_analyze
{
    "text": "清明节，又称踏青节、行清节、三月节、祭祖节等",
    "analyzer": "bd-nlp-phrase"
}
```

短语切分结果：

```
{
 "tokens": [
   {
     "token": "清明节",
     "start_offset": 0,
     "end_offset": 3,
     "type": "WORD",
     "position": 0
   },
   {
     "token": "又称",
     "start_offset": 4,
     "end_offset": 6,
     "type": "WORD",
     "position": 2
   },
   {
     "token": "踏青节",
     "start_offset": 6,
     "end_offset": 9,
     "type": "WORD",
     "position": 3
   },
   {
     "token": "行清节",
     "start_offset": 10,
     "end_offset": 13,
     "type": "WORD",
     "position": 5
   },
   {
     "token": "三月节",
     "start_offset": 14,
     "end_offset": 17,
     "type": "WORD",
     "position": 7
   },
   {
     "token": "祭祖",
     "start_offset": 18,
     "end_offset": 20,
     "type": "WORD",
     "position": 9
   },
   {
     "token": "节",
     "start_offset": 20,
     "end_offset": 21,
     "type": "WORD",
     "position": 10
   }
 ]
}
```

- ik_smart 智能切词

```
POST _analyze
{
  "analyzer": "ik_smart",
  "text": "清明节，又称踏青节、行清节、三月节、祭祖节等"
}
```

切词结果：

```
{
  "tokens": [
    {
      "token": "清明节",
      "start_offset": 0,
      "end_offset": 3,
      "type": "CN_WORD",
      "position": 0
    },
    {
      "token": "又称",
      "start_offset": 4,
      "end_offset": 6,
      "type": "CN_WORD",
      "position": 1
    },
    {
      "token": "踏青",
      "start_offset": 6,
      "end_offset": 8,
      "type": "CN_WORD",
      "position": 2
    },
    {
      "token": "节",
      "start_offset": 8,
      "end_offset": 9,
      "type": "CN_WORD",
      "position": 3
    },
    {
      "token": "行",
      "start_offset": 10,
      "end_offset": 11,
      "type": "CN_WORD",
      "position": 4
    },
    {
      "token": "清",
      "start_offset": 11,
      "end_offset": 12,
      "type": "CN_CHAR",
      "position": 5
    },
    {
      "token": "节",
      "start_offset": 12,
      "end_offset": 13,
      "type": "CN_WORD",
      "position": 6
    },
    {
      "token": "三月",
```

```
        "start_offset": 14,
        "end_offset": 16,
        "type": "CN_WORD",
        "position": 7
      },
      {
        "token": "节",
        "start_offset": 16,
        "end_offset": 17,
        "type": "COUNT",
        "position": 8
      },
      {
        "token": "祭祖",
        "start_offset": 18,
        "end_offset": 20,
        "type": "CN_WORD",
        "position": 9
      },
      {
        "token": "节",
        "start_offset": 20,
        "end_offset": 21,
        "type": "CN_WORD",
        "position": 10
      }
    ]
  }
```

Analyze API使用

基础模型粒度分词

```
 POST /_analyze
{
  "analyzer": "bd-nlp-basic",
  "text": "去年我们和他们展开了炉际竞赛，第一回合赢了，第二回合和第三回合却败下阵来。"
}
```

分词结果：

```
 {
  "tokens": [
    {
      "token": "去年",
      "start_offset": 0,
      "end_offset": 2,
      "type": "WORD",
      "position": 0
    },
    {
      "token": "我们",
      "start_offset": 2,
      "end_offset": 4,
      "type": "WORD",
      "position": 1
    },
    {
      "token": "和",
      "start_offset": 4,
```

```
        "end_offset": 5,
        "type": "WORD",
        "position": 2
    },
    {
        "token": "他们",
        "start_offset": 5,
        "end_offset": 7,
        "type": "WORD",
        "position": 3
    },
    {
        "token": "展开",
        "start_offset": 7,
        "end_offset": 9,
        "type": "WORD",
        "position": 4
    },
    {
        "token": "炉际",
        "start_offset": 10,
        "end_offset": 12,
        "type": "WORD",
        "position": 6
    },
    {
        "token": "竞赛",
        "start_offset": 12,
        "end_offset": 14,
        "type": "WORD",
        "position": 7
    },
    {
        "token": "第一",
        "start_offset": 15,
        "end_offset": 17,
        "type": "WORD",
        "position": 9
    },
    {
        "token": "回合",
        "start_offset": 17,
        "end_offset": 19,
        "type": "WORD",
        "position": 10
    },
    {
        "token": "赢",
        "start_offset": 19,
        "end_offset": 20,
        "type": "WORD",
        "position": 11
    },
    {
        "token": "第二",
        "start_offset": 22,
        "end_offset": 24,
        "type": "WORD",
        "position": 14
    },
    {
        "token": "回合",
        "start_offset": 24,
```

```
        "start_offset": 24,
        "end_offset": 26,
        "type": "WORD",
        "position": 15
      },
      {
        "token": "和",
        "start_offset": 26,
        "end_offset": 27,
        "type": "WORD",
        "position": 16
      },
      {
        "token": "第三",
        "start_offset": 27,
        "end_offset": 29,
        "type": "WORD",
        "position": 17
      },
      {
        "token": "回合",
        "start_offset": 29,
        "end_offset": 31,
        "type": "WORD",
        "position": 18
      },
      {
        "token": "败",
        "start_offset": 32,
        "end_offset": 33,
        "type": "WORD",
        "position": 20
      },
      {
        "token": "下",
        "start_offset": 33,
        "end_offset": 34,
        "type": "WORD",
        "position": 21
      },
      {
        "token": "阵",
        "start_offset": 34,
        "end_offset": 35,
        "type": "WORD",
        "position": 22
      },
      {
        "token": "来",
        "start_offset": 35,
        "end_offset": 36,
        "type": "WORD",
        "position": 23
      }
    ]
}
```

短语模型大粒度分词

```
POST /_analyze
{
  "analyzer": "bd-nlp-phrase",
  "text": "去年我们和他们展开了炉际竞赛，第一回合赢了，第二回合和第三回合却败下阵来。"
}
```

分词结果：

```
{
 "tokens": [
   {
     "token": "去年",
     "start_offset": 0,
     "end_offset": 2,
     "type": "WORD",
     "position": 0
   },
   {
     "token": "我们",
     "start_offset": 2,
     "end_offset": 4,
     "type": "WORD",
     "position": 1
   },
   {
     "token": "和",
     "start_offset": 4,
     "end_offset": 5,
     "type": "WORD",
     "position": 2
   },
   {
     "token": "他们",
     "start_offset": 5,
     "end_offset": 7,
     "type": "WORD",
     "position": 3
   },
   {
     "token": "展开",
     "start_offset": 7,
     "end_offset": 9,
     "type": "WORD",
     "position": 4
   },
   {
     "token": "炉际竞赛",
     "start_offset": 10,
     "end_offset": 14,
     "type": "WORD",
     "position": 6
   },
   {
     "token": "第一回合",
     "start_offset": 15,
     "end_offset": 19,
     "type": "WORD",
     "position": 8
   },
   {
     "token": "赢",
```

```
      "start_offset": 19,
      "end_offset": 20,
      "type": "WORD",
      "position": 9
    },
    {
      "token": "第二回合",
      "start_offset": 22,
      "end_offset": 26,
      "type": "WORD",
      "position": 12
    },
    {
      "token": "和",
      "start_offset": 26,
      "end_offset": 27,
      "type": "WORD",
      "position": 13
    },
    {
      "token": "第三",
      "start_offset": 27,
      "end_offset": 29,
      "type": "WORD",
      "position": 14
    },
    {
      "token": "回合",
      "start_offset": 29,
      "end_offset": 31,
      "type": "WORD",
      "position": 15
    },
    {
      "token": "败",
      "start_offset": 32,
      "end_offset": 33,
      "type": "WORD",
      "position": 17
    },
    {
      "token": "下",
      "start_offset": 33,
      "end_offset": 34,
      "type": "WORD",
      "position": 18
    },
    {
      "token": "阵",
      "start_offset": 34,
      "end_offset": 35,
      "type": "WORD",
      "position": 19
    },
    {
      "token": "来",
      "start_offset": 35,
      "end_offset": 36,
      "type": "WORD",
      "position": 20
    }
  ]
```

```
}
```

索引指定Analyzer

```
 PUT test
{
  "mappings": {
    "doc": {
      "properties": {
        "k1": {
          "type": "text",
          "analyzer": "bd-nlp-basic" // 使用基础粒度模型
        },
        "k2": {
          "type": "text",
          "analyzer": "bd-nlp-phrase" // 使用短语粒度模型
        }
      }
    }
  },
  "settings": {
    "index": {
      "number_of_shards": "1",
      "number_of_replicas": "0"
    }
  }
}
```

索引指定Tokenizer

```
 PUT /test
{
   "settings":{
      "analysis":{
         "analyzer":{
            "my_analyzer":{
               "tokenizer":"bd-nlp-basic",  // 自定义一个analyzer
               "filter":[
                  "lowercase"              // 添加应用需要的过滤器
               ]
            }
         }
      }
   },
   "mappings":{
      "properties":{
         "k2":{
            "type":"text",
            "analyzer":"my_analyzer"        // 将自定义analyzer应用到对应字段上
         }
      }
   }
}
```

准确率与召回率

百度内部大数据集测试结果：

模型	准确率	召回率	F值
analysis-baidu-nlp	98.8%	98.9%	98.8%

百度NLP中文分词词典动态更新

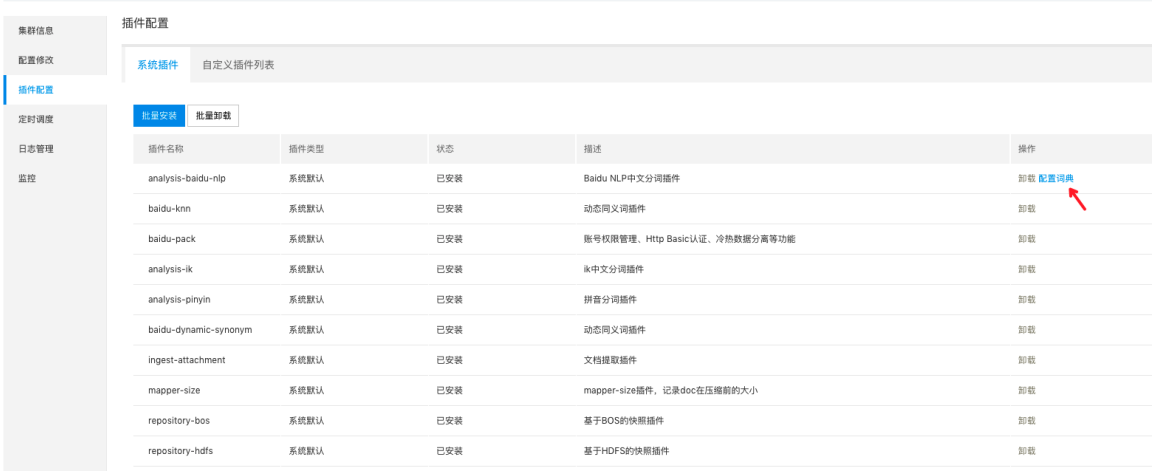
百度智能云Elasticsearch的NLP中文分词插件支持用户添加自定义词典干预NLP模型，从而进行分词词典动态热更新。

用户可以根据需求，通过**上传词典文件**或**输入文本**两种方式添加自定义词典。

> 注意：**NLP中文分词词典动态更新**目前支持功能发布后新创建的7.4.2版本的实例（即2021年01月20日后创建的7.4.2版本的实例），不支持的集群请提交工单，BES团队会协助升级集群，升级方式参见ES版本升级。

使用方式

1. 登录百度智能云Elasticsearch控制台，并点击集群名称，进入集群详情界面。
2. 在左侧导航栏，单击**插件配置**。



3. 点击“配置词典”，在弹出的页面添加自定义词典。用户可以通过**上传词典文件**或**输入文本**两种方式添加自定义词典。

- 通过**上传文件**的方式：

- 通过**直接输入文本**的方式：

4. 上传成功后，即完成词典更新，可以使用以下语句测试自定义词典的效果。

```
POST /_analyze
{
  "analyzer": "bd-nlp-basic",
  "text": "测试语句"
}
```

自定义词典

词典格式说明

1. 干预规则按行分割（`\n`）。

2. **上传词典文件**时，可以选择使用空格或使用TAB（`\t`）来进行干预规则内的词典分词；**直接输入文本**的方式则只能使用空格作为分隔符。

3. 干预规则内可以使用`[ ]`对`basic`模型和`phrase`模型同时干预。

干预规则说明

用户可以按照以下格式对自定义词典进行配置：

```
 中华    人民    共和国
中华    [人民    共和国]
中华
人民
共和国
```

例如：未进行干预前，`bd-nlp-basic`(`basic`模型)和`bd-nlp-phrase`(`phrase`模型)对`中华人民共和国`分词的结果是:

```
 中华人民共和国
```

干预 basic 模型分词

干预词典:

带`上下文关系`的干预规则:

```
 中华  人民  共和国
```

不带`上下文关系`的干预规则:

```
 中华
人民
共和国
```

干预后两种力度的切词结果一样:

```
 {
   "tokens":[
      {
          "token":"中华",
          "position":0
      },
      {
          "token":"人民",
          "position":1
      },
      {
          "token":"共和国",
          "position":2
      }
   ]
}
```

注意：

- `上下文关系` 表示是不是切出这个词是由这个词的`前后词`来决定，例如用户如果将干预规则配置为：

```
中华 人民 共和国
```

则只会对`中华人民共和国`进行按照该规则进行分词，而不会对`中华人民`或`人民共和国`进行分词；而如果将干预规则配置为：

```
中华
人民
共和国
```

则可以对`中华人民`或`人民共和国`进行分词。

- 干预基础粒度分词，会有一定概率影响phrase粒度分词结果，因为phrase模型是对basic结果进行智能组合，会有无法重组回原粒度切词结果的情况出现

同时干预 basic 模型和 phrase 模型

干预词典规则:

```
中华 [人民 共和国]
```

`bd-nlp-basic`分词结果:

```
{
  "tokens":[
     {
        "token":"中华",
        "position":0
     },
     {
        "token":"人民",
        "position":1
     },
     {
        "token":"共和国",
        "position":2
     }
  ]
}
```

`bd-nlp-phrase`分词结果:

```
{
  "tokens":[
     {
        "token":"中华",
        "position":0
     },
     {
        "token":"人民共和国",
        "position":1
     }
  ]
}
```

`phrase`模型会对`[]`内的词(`basic`可以分词)重组成一个词。

利用上下文语义进行干预 basic 和 phrase 模型

1. `[]`内的词是对`phrase`模型进行干预，`phrase`模型会对`[]`内的词(`basic`模型可以切出来)智能组合
2. 没有`[]`只会根据上下文语义对`basic`模型进行干预

干预词典规则：

```
云原生 [数仓 创始]
```

原始文本内容:

```
云原生数仓创始和数仓创始
```

规则生效前:

`basic`模型分词结果:

```
[云原生, 数, 仓, 创始, 和, 数, 仓, 创始]
```

`phrase`模型分词结果:

```
[云原生, 数, 仓, 创始, 和, 数, 仓, 创始]
```

规则生效后:


```
{
  "tokens":[
     {
        "token":"云原生",
        "position":0
     },
     {
        "token":"数仓",
        "position":1
     },
     {
        "token":"创始",
        "position":2
     },
     {
        "token":"和",
        "position":3
     },
     {
        "token":"数",
        "position":4
     },
     {
        "token":"仓",
        "position":5
     },
     {
        "token":"创始",
        "position":6
     }
  ]
}
```

分词结果为:

```
[云原生, 数仓, 创始, 和, 数, 仓, 创始]
```

可以看到文本中第二个的`数仓创始`，由于不满足*上下文关系*（该词前面的词不是`云原生`），所以不会切出`数仓`。

`nlp-bd-phrase`分词结果：

```
{
  "tokens":[
     {
        "token":"云原生",
        "position":0
     },
     {
        "token":"数仓创始",
        "position":1
     },
     {
        "token":"和",
        "position":2
     },
     {
        "token":"数",
        "position":3
     },
     {
        "token":"仓",
        "position":4
     },
     {
        "token":"创始",
        "position":5
     }
  ]
}
```

切词结果:

```
[云原生, 数仓创始, 和, 数, 仓, 创始]
```

可以看到`数仓`和`创始`被`phrase`模型智能组合为`数仓创始`。

说明： `云原生`可以放在干预词典规则的前面和后面, 具体可以根据业务的`上下文语义`需求。

注意

- 不是所有的干预规则都会成功，会有一个模型匹配的概率问题。
- 上传的文件用**UTF-8编码**，扩展名为**txt**。
- 上传的词典文件最大为**10M**。

基于BOS的冷热数据分离

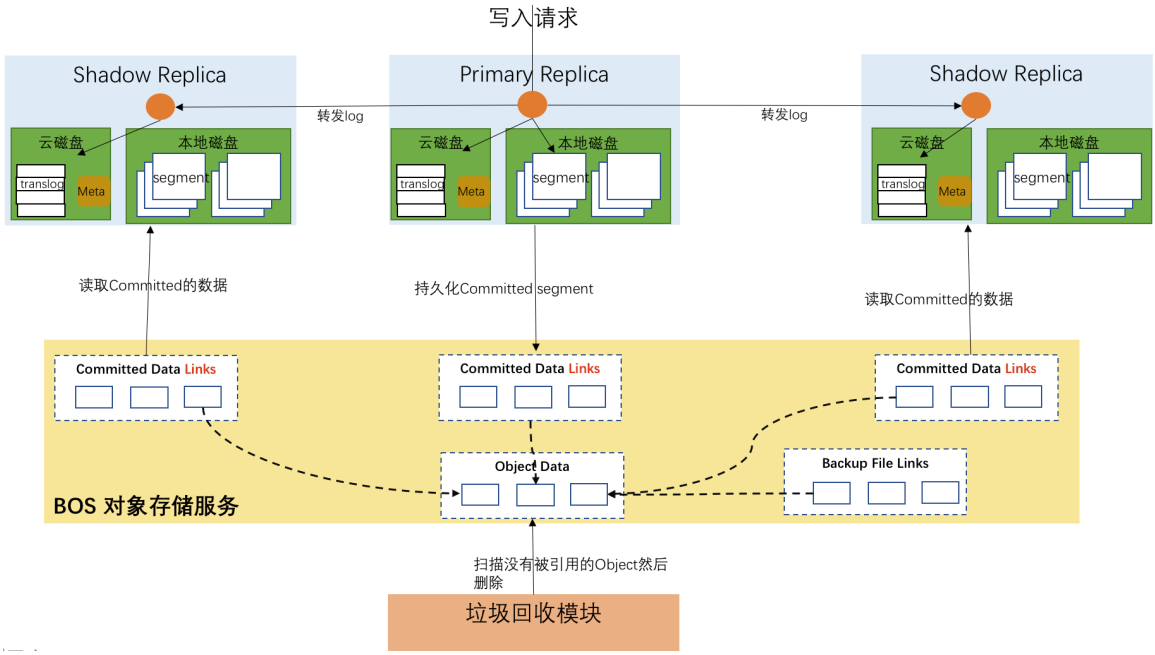
百度智能云Elasticsearch（以下简称为ES）使用不同的存储介质来存储数据，达到冷热数据分离的目的：

- 对于读写性能要求比较高的“热数据”，使用SSD云磁盘存储，保障了高效的查询性能。

- 对于存储量需求比较大但对查询性能要求较低的“冷数据”，使用价格更低廉的百度对象存储（BOS）存储数据，大幅度降低存储成本。

> 注意：目前**冷热数据分离架构**仅支持功能发布后新创建的7.4.2、7.10.2版本的实例。无法使用冷热数据分离功能的集群请提交工单，BES团队会协助升级集群，升级方式参见ES版本升级。

实现原理



优势

- 冷热数据分离：热数据使用SSD云磁盘存储，保证性能，冷数据使用价格低廉的BOS存储，更经济的满足用户的需求。
- 高性价比：使用BOS作为存储介质，扩大了ES单节点的存储容量，用户可以使用较少的ES节点存储大量的数据，大幅度降低存储成本。

适用场景

- 历史日志分析场景：历史数据量大，查询频率相对低，使用冷热数据分离可以大幅降低存储成本。

使用方法

最开始使用基于BOS冷热分离架构时，首先创建BOS Bucket的并配置，然后将BOS相关信息配置到ES中。最后就可以在ES中创建冷热数据分离Index，在业务需要时 手动 或通过管控平台 定时调度功能 进行索引置冷，将Index迁移到BOS。

创建BOS Bucket

冷热数据分离使用BOS来做冷数据的共享存储，因此在使用冷热数据分离架构前，需要先新建BOS Bucket。

用户可以通过 Console 控制台创建BOS Bucket：

1. 登录 对象存储 BOS 管理控制台
2. 在控制台左侧的导航栏中找到并点击 + 按钮（“新建 Bucket”按钮），在弹出框中按照提示创建Bucket。
3. 新建Bucket配置项说明：

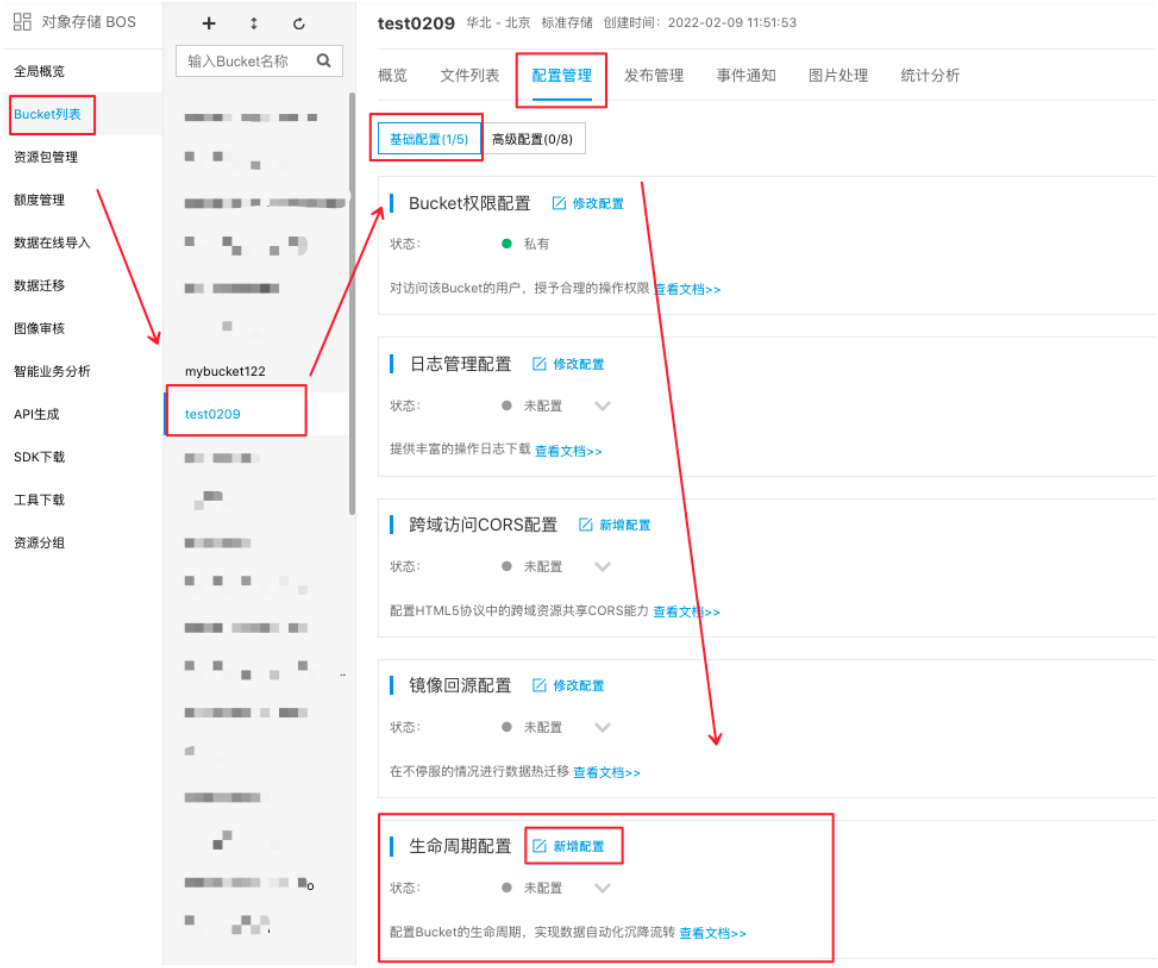
- **默认存储类型**：选择“**标准存储**”
- **CDN官方加速：关闭**

其他配置项用户可根据业务需求选择。创建Bucket详细方式参见：创建Bucket

Bucket生命周期配置

使用BOS创建Bucket后，**强烈建议**配置Bucket生命周期。

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧Bucket列表中，选择需要设置权限的 Bucket，点击 Bucket 名称进入 Bucket 管理目录。
3. 在上方导航栏选择 **配置管理** 标签。
4. 在 **配置管理** 页面中选择 **基础配置**，在 **生命周期配置** 区域点击 **新增配置** ，对该Bucket的生命周期进行配置。默认情况下该功能不开启。



5. 配置规则：

- 不可启用`删除Object`规则，避免冷数据被删除
- 启用`删除Part`规则，保存天数设为`1`天，防止未上传完整的数据块占用存储空间，造成多余存储费用

配置BOS信息到BES

通过如下API可以设置BOS存储信息：

```
PUT /_cluster/settings
{
  "persistent": {
    "bpack.remote_storage.bos.access_key": "xxxxxxxx",
    "bpack.remote_storage.bos.secret_key": "xxxxxxxx",
    "bpack.remote_storage.bos.bucket": "es_remote_bucket",
    "bpack.remote_storage.bos.endpoint": "bj.bcebos.com",
    "bpack.remote_storage.bos.base_path": "es_shared_data"
  }
}
```

参数说明：

参数	含义
bpack.remote_storage.bos.access_key	访问BOS需要的access key, 可以在百度智能云的控制台上获得。
bpack.remote_storage.bos.secret_key	访问BOS需要的secret access key，可以在百度智能云的控制台上获得。
bpack.remote_storage.bos.bucket	BOS上建立的bucket的名称，需要提前创建。
bpack.remote_storage.bos.endpoint	BOS的访问地址，一般情况下每个地区有不同的地址，您可以参考BOS的使用文档来获得每个地域的地址。
bpack.remote_storage.bos.base_path	百度智能云Elasticsearch存放数据的目录。

> 冷热分离引擎会定期请求BOS，扫描并清理BOS上的过期数据，因此配置的AK、SK需要拥有该bucket的write、read、list权限。

创建冷热数据分离Index

在创建Index时，需要特别指定*index.store.type*为`bosfs`，这样百度智能云Elasticsearch才会使用冷热分离的架构来管理Index。

```
PUT /{index_name}
{
  "settings": {
    "index.store.type": "bosfs"
  }
}
```

在以上操作结束后，用户可以和访问普通Index一样来对冷热数据分离的Index进行写入、查询数据。

对已存在的Index启用冷热数据分离架构

对于之前已经建好的索引，也可以使用基于bos的冷热分离功能。

前提条件：完成BOS Bucket创建和配置，已经将BOS信息配置到BES。

操作说明：

首先关闭索引:

```
POST /{index_name}/_close
```

然后将*store.type*设置为`bosfs`：

```
put /{index_name}/_settings
{
    "index.store.type": "bosfs"
}
```

再重新打开索引:

```
POST /{index_name}/_open
```

在以上操作结束后，就将此Index设置成了冷热数据分离的Index。

冷数据迁移的触发条件

冷热数据分离的Index中的数据在满足以下2种情况之一时，会触发数据迁移到BOS。

手动触发迁移

当这个Index中的数据不再写入，并且读取频率比较低的时候，用户可以把Index设置为Cold状态，手动触发数据迁移。设置方式如下：

```
POST /_bpack/migrate/{index_name}/cold
```

> 注意：Cold状态的Index为只读状态，不能再写入数据。已经置为Cold状态的Index，不可以再改为普通状态。

定时调度触发迁移

用户可以使用定时调度-索引置冷来触发迁移，在定时置冷任务执行后，Index会置为Cold，并触发数据迁移，效果与上文手动触发迁移一致。

> 建议用户使用手动触发迁移或定时调度触发迁移来进行索引置冷。

本地缓存淘汰策略

当本地磁盘使用率超过了*缓存最大磁盘限制*，本地数据会被删除。*缓存最大磁盘限制*默认为75%，用户可以通过修改设置项bpack.bosfs.cache.disk.limit来修改*缓存最大磁盘限制*，修改方式如下：

```
 PUT /_cluster/settings
{
    "persistent": {
        "bpack.bosfs.cache.disk.limit" : "50%"
    }
}
or
PUT /_cluster/settings
{
    "persistent": {
        "bpack.bosfs.cache.disk.limit" : "0.5"
    }
}
```

本地缓存配置

数据置冷后，为了提升ES的查询性能，冷热分离架构将会缓存部分数据，减少对BOS的访问请求。在检索时，第一次获取数据会直接访问BOS，之后将获取到的数据缓存在本地，后续访问会先检查是否有缓存。

缓存的配置

> 仅在7.10.2版本支持

配置	说明	默认值	类型
bpack.migrate.max_bytes_per_sec	置冷数据时数据上传bos速率限制值	40m	字符串
bpack.bosfs.cache.custom_cache_names	缓存系统支持多级缓存，分别用来缓存不同访问粒度的数据。此配置列出所有缓存的名字，即使不配置，系统也会默认有2个缓存，名字为default和dic。如果自定义配置，可以配置除default和dic外的其他缓存类型。	["default","dic"]	字符串数组
bpack.bosfs.cache.{cache_name}.expiry_time	缓存中的文件在最后一次读或者写入后开始计时，在指定的时间后过期。使用TimeUnit格式表示，例如『10s』、『10m』、『3h』等，不可设置为小数值，如『1.5h』。	60m	字符串
bpack.bosfs.cache.{cache_name}.disk_limit	缓存的最大容量。当缓存尝试加载数据时超过了缓存的最大容量限制，它将触发驱逐操作。该值可以设置为一个带有存储容量单位的值，例如『10kb』、『10mb』、『3g』等，不建议设置为小数值，如『1.5g』。	disk的10%	字符串
bpack.bosfs.cache.{cache_name}.file.types	文件缓存的文件后缀名。	-	字符串数组
bpack.bosfs.cache.default.expiry_time	默认文件缓存的过期时间	60m	字符串
bpack.bosfs.cache.default.disk_limit	默认文件缓存的最大容量	disk的30%	字符串
bpack.bosfs.cache.default.file.types	默认文件缓存的文件后缀名。会缓存除其他缓存外的所有文件。	-	字符串数组
bpack.bosfs.cache.dic.expiry_time	默认文件缓存的过期时间	120m	字符串
bpack.bosfs.cache.dic.disk_limit	默认文件缓存的最大容量	disk的10%	字符串
bpack.bosfs.cache.dic.file.types	默认文件缓存的文件后缀名。会缓存除其他缓存外的所有文件。	["tvx", "si", "tip", "tim", "tvb", "fdx", "doc", "dim"]	字符串数组

数据观测

提供数据观测接口，方便使用者观察数据分布情况。

1、索引级观测接口

接口命令：

```
GET /_bpack/migration/stats?v
```

响应结果说明：

属性	说明
index	索引名称
uuid	索引uuid
status	bosfs类型索引的冷热状态
active.shard	索引活跃分片数
finished.shard	完成bos数据上传的分片数
waiting.shard	等待bos数据上传的分片数
running.shard	进行bos数据上传的分片数
hot.data	本地索引数据大小
cold.data	bos上索引数据大小
cache.data	本地索引缓存数据大小

响应示例：

```
1 index                                uuid status active.shard finished.shard waiting.shard running.shard hot.data cold.data cache.data
2 [redacted]_test_[redacted]           [redacted]A    HOT           12             0             0             0   83.6gb        0b        0b
3 [redacted]_test_[redacted]           [redacted]     COLD          12             0             0            12  491.8gb    76.7gb   345.8mb
4 [redacted]_test[redacted]            [redacted]Q    COLD          12            12             0             0       0b   568.5gb      8gb
```

2、节点级观测接口

> 仅在7.10.2版本支持

接口命令：

```
GET /_bpack/node/migration/stats?v
```

响应结果说明：

属性	说明
id	es节点id
ip	es节点ip
shards	节点上bosfs索引的分片的总量
hot.data	本地索引数据大小
cold.data	bos上索引数据大小
cache.data	本地索引缓存数据大小

响应示例：

```
id                     ip            shards hot.data cold.data cache.data
[redacted]             [redacted]    404    198.9gb    18.4tb    249.7gb
[redacted]             [redacted]    404    193.8gb    18.4tb    248.4gb
[redacted]             [redacted]    404    197.2gb    18.4tb    248.6gb
```

3、集群级观测接口

> 仅在7.10.2版本支持

接口命令：

```
GET /_bpack/cluster/migration/stats?v
```

响应结果说明：

属性	说明
shards	节点上bosfs索引的分片的总量
hot.data	本地索引数据大小
cold.data	bos上索引数据大小
cache.data	本地索引缓存数据大小

响应示例：

```
health shards hot.data cold.data cache.data
green  1212    588.6gb    55.4tb    746.7gb
```

常用API

> 仅在7.10.2版本支持

1、冷索引预热

在初次收到搜索请求时，由于冷数据均保存在BOS，初次请求会存在较高延迟。为了避免这种情况，用户可以使用冷索引预热APi对目标索引进行预热，下载对应文件进入缓存，加快检索效率。

```
POST /_bpack/bosfs/{index}/_pre_hot
```

2、清理缓存

用户可以使用清理缓存APi手动清理缓存文件。

```
POST /_bpack/bosfs/_clear_cache
POST /_bpack/bosfs/{index}/_clear_cache
```

性能测试

- 测试工具：Elasticsearch 官方的esrally工具
- 数据集：pmc
- 数据量：
  - 总文档数：14,354,975
  - 存储总量：约500GB
- 集群规格：
  - 节点数：3
  - shard数：12
  - 副本数：0

针对不同的查询种类，分别设置一定的吞吐量进行压测，测试结果如下：

			高性能云盘集群	冷热架构集群
节点配置			cpu：8 内存：32G	cpu：8 内存：32G
term查询	吞吐量(ops/s)		397.25	396.06
	延迟(ms)	90%分位值	32.04	16.11
		平均值	14.61	12.05
	查询耗时(ms)	90%分位值	12.05	13.70
		平均值	11.16	11.02
phrase查询	吞吐量(ops/s)		199.21	199.14
	延迟(ms)	90%分位值	30.88	31.78
		平均值	25.83	27.25
	查询耗时(ms)	90%分位值	26.16	30.53
		平均值	24.24	26.27
agg查询	吞吐量(ops/s)		9.98	9.98
	延迟(ms)	90%分位值	425.30	462.42
		平均值	388.61	428.12
	查询耗时(ms)	90%分位值	424.45	461.20
		平均值	387.63	426.92
scroll查询	吞吐量(pages/s)		37.64	37.65
	延迟(ms)	90%分位值	396.26	350.29
		平均值	375.37	340.43
	查询耗时(ms)	90%分位值	394.86	348.52
		平均值	373.68	338.66

结论

在集群配置相同的情况下，在进行各种查询时，冷热数据架构的耗时与高性能云盘查询耗时基本持平。其中，在term查询和scroll查询时，冷热数据架构的查询性能更高。

价格方面，冷热数据架构使用BOS作为存储介质，单价比高性能云盘存储低很多，而且是按量付费，无须提前创建存储容量。

因此，冷热数据架构可以做到更好的做到降本增效。

常见问题

- Q：已经置为`bosfs`的Index还能改为其他的存储类型（如hybridfs、niofs等）吗？

  A：如果改为其他存储类型，由于数据存储目录不同，数据将失效。所以**不可以**改变为其他存储类型。

- Q：对于已经存在的Index可以启用基于BOS的冷热分离结构吗？

  A：可以，操作方式如上文`《对已存在的Index启用冷热数据分离架构》`章节。

- Q：已经置为Cold状态的Index还能改回普通状态吗？

  A：不可以。且Cold状态的Index是只读状态，不能再写入数据了。

- Q：没有设置BOS信息，或BOS信息有误，可以创建`bosfs`的Index吗？

  A：可以创建成功，但Index不可用。更正BOS信息后，需要重新创建索引。

基于段文件的主从复制

概述

主从复制模式

- Elasticsearch的主从复制模式，是基于“文档复制”（document replication）：当有增删改文档操作时，操作请求会先转发到主分片（primary shard），主分片在其本地构建索引，然后把这个操作请求转发给各副本分片（replica shard）所在节点，每个节点分别构建各自的索引数据。
- 百度云Elasticsearch提供一种基于“段复制”（segment replication）的主从同步机制：只有主分片会执行构建索引数据的过程，构建完成后，把新增的索引数据直接复制给各个副本分片。由于Elasticsearch的索引数据底层是lucene的段（segment）文件，因此这种数据复制模式被称为“段复制”。
- 您可根据使用需求，在创建新索引时选择主从复制模式为“文档复制”或“段复制”。

优势

- “文档复制”每个副本所在节点各自构建索引，因此集群整体的cpu开销更大。相比而言，“段复制”会节省这部分cpu压力，从而增加集群的写入吞吐量。不过，“段复制”也会加大集群内部的网络通信压力，以及搜索滞后性。
- 详细对比数据可参考本页末的性能测试。

适用场景

1. 写入QPS较大、CPU使用率较高的集群。
2. 业务对搜索滞后性不敏感。典型使用场景为日志检索。

使用方式

创建基于段复制的索引

主从复制模式可基于集群粒度或索引粒度来配置。配置只影响新创建的索引，且索引一旦创建后，该索引的复制模式不能再修改。

	配置项	类型	说明
集群粒度	cluster.indices.replication.strategy	String	取值"DOCUMENT"或"SEGMENT"（区分大小写），默认值为"DOCUMENT"
索引粒度	index.replication.type	String	取值"DOCUMENT"或"SEGMENT"（区分大小写），默认值为"DOCUMENT"

- 集群粒度设置使用示例：

```
PUT /_cluster/settings
{
  "persistent": {
    "cluster.indices.replication.strategy": "SEGMENT"
  }
}
```

此后新创建的索引，如果不显式指定索引粒度的复制模式，则以集群粒度的配置为准。

- 索引粒度设置使用示例：

```
PUT /my-index1
{
  "settings": {
    "index.replication.type": "SEGMENT"
  }
}
```

查看段复制的执行状态

您可查看开启了段复制的索引的执行状态，并可以指定索引：

```
GET _cat/segment_replication?v
GET _cat/segment_replication/{index}?v
```

请求参数（可选）：

参数名	类型	说明
shards	String	指定要查看的shard id列表，如"0,1,2"。默认不填，即查看全部shard。
active_only	Boolean	只查看正在执行段复制的shard。默认false。

响应结果说明：

参数名	说明
shardId	shard id
target_node	复制目标节点名
target_host	复制目标节点ip
checkpoints_behind	副本分片落后主分片的checkpoint数（一个checkpoint对应一个lucene段）
bytes_behind	副本分片落后主分片的字节数
current_lag	自主分片刷新（但目前副本尚未同步）到当前的时间
last_completed_lag	上一次主从段复制过程的执行耗时
rejected_requests	拒绝请求数

相关优化项

如果启用了“段复制”模式，推荐您了解以下的集群粒度设置，可能会帮助您更好地使用集群。

依据primary数量进行shard分配

Elasticsearch默认的shard分配策略，是使各数据节点的shard数量趋于相同，但不区分primary shard和replica shard。

而如果您开启了段复制功能，且集群内全部索引或绝大部分索引都是基于段复制的，那么仅primary shard有较大的写入压力，因此我们提供了一种根据primary shard数量分配的策略配置：

配置项	类型	说明
cluster.routing.allocation.balance.prefer_primary	Boolean	默认值false。如果设置为true，新分配shard的策略会尽量使得各节点的primary shard数量均匀。

不过，这个设置只影响新建索引时的shard分配策略。如果开启设置前，所有shard已经被分配均衡，不会因为primary shard的分布不均衡而触发重分配。在其它场景下，比如故障恢复中，该设置也不会影响shard的重分配。

背压机制

我们提供一种分片级背压机制：当副本分片落后于主分片超过一定阈值时，可以动态地拒绝对这个分片的写入请求，直到下降到阈值以下时，又可以恢复写入。如果开启集群的段复制背压机制，则所有段复制模式的索引都会启用背压机制。

配置项如下：

配置项	类型	说明
segrep.pressure.enabled	Boolean	集群背压机制的开关。默认值true。
segrep.pressure.time.limit	TimeUnit	副本分片从主分片复制所等待的最长时间。默认值5分钟。
segrep.pressure.checkpoint.limit	Integer	副本分片落后于主分片的checkpoint（一个checkpoint对应一个lucene段）的最大数量。默认为4。
segrep.pressure.replica.stale.limit	Float	陈旧副本的比例阈值。当一个副本的"segrep.pressure.time.limit"和"segrep.pressure.checkpoint.limit"两者都达到了阈值，则认为这个副本为“陈旧的”（stale）。当一个分片的各个副本中，陈旧比例达到阈值，则这个分片开始拒绝写入。默认值0.5，即50%。

拷贝限速

节点间的段复制过程中，可能存在网络带宽占用较大的问题，默认开启一定的速度限制。而如果您需要尽可能降低主从复制的延迟，可以提高限速阈值，或取消限速。

配置项	类型	说明
indices.replication.max_bytes_per_sec	String	每个节点的段复制传输速度限制。默认值"200mb"。如果置为"0mb"则不限速。

内核版本要求

Elasticsearch版本7.10.2，内核版本1.7.0及以上。

性能测试

测试集群包括5个es数据节点（bes.g3.c8m32），机型详细规格如下：

配置项	类型
核数	8
内存	32GB
磁盘类型	增强型ssd pl1
磁盘容量	512GB
网卡带宽	3Gbps

使用esrally测试工具，数据集为nyc_taxis，每组测试写入8000万条数据（单shard存储占用约16GB）。写入时，每bulk请求10000条文档，并发数为10。

测试结果如下：

primary	replica	err-rate(%)	文档复制mean throughput(docs/s)	段复制mean throughput(docs/s)	Percent difference	文档复制mean service time(ms)	段复制mean service time(ms)	Percent difference
5	1	0	219864.67	315401.21	43.45%	496.78	339.61	-31.64%
10	1	0	251130.60	321112.98	27.87%	437.10	330.37	-24.42%
20	1	0	247969.20	326191.10	31.55%	446.63	318.40	-28.71%
40	1	0	252686.17	329312.61	30.32%	417.75	306.47	-26.64%

结论：

在上述集群配置条件下，与文档复制相比，段复制可以明显提升集群写入性能，写入吞吐量可提升约30%。

慢查询隔离

百度智能云Elasticsearch提供慢查询隔离的功能，支持用户配置固定大小的资源隔离池，当单条查询运行时申请的内存大小超过设定的阈值时，将该查询任务放到隔离池中管理。当隔离池的内存总量超过设定阈值时，通过可插拔的优先级策略熔断资源消耗最大的查询。避免单条查询导致的集群异常问题，提升整体服务稳定性。

> 注意：目前仅7.10.2版本的Elasticsearch集群支持慢查询隔离。

使用方法

开启慢查询隔离和全局查询超时功能

慢查询隔离和全局查询超时特性默认关闭，使用时需要先开启功能，配置后立即生效，两个开关是独立功能。

```
 PUT /_cluster/settings
{
    "persistent": {
  "search.isolator.enabled": "true",
  "search.isolator.time.enabled": "true"
 }
}
```

配置慢查询隔离

分为三种情况，设置单个分片查询任务被定义为慢查询任务的阈值、设置触发隔离池内查询任务中断的阈值和设置中断隔离池中某一条查询任务的选取策略。

设置单个分片查询任务被定义为慢查询任务的阈值

```
 PUT /_cluster/settings
{
    "persistent": {
  "search.isolator.trigger.task.mem_cost": "50m",
  "search.isolator.trigger.task.latency": "10s"
 }
}
```

参数说明

参数名	数据类型	参数说明
search.isolator.trigger.task.mem_cost	String	查询请求内存使用阈值，当单个查询任务使用内存超过阈值时，系统会将其放入慢查询隔离池中。默认100mb，取值范围：0b~节点最大堆内存。
search.isolator.trigger.task.latency	String	查询请求延迟阈值，当单个查询任务耗时超过阈值时，系统会将其放入慢查询隔离池中。默认10s，取值范围：≥ 0ms。

设置触发隔离池内查询任务中断的阈值

```
PUT /_cluster/settings
{
 "persistent": {
  "search.isolator.total.mem.limit": "60%",
  "search.isolator.total.heap.usage.limit": "75%",
  "search.isolator.total.tasks.limit": "1000"
 }
}
```

参数说明

参数名	类型	说明
search.isolator.total.mem.limit	String	慢查询隔离池堆内存占用集群的比例阈值。默认60%，表示占用率达到60%时，通过优先级策略熔断隔离池中资源消耗最大的查询。取值范围：0.0 ~ 100.0%。
search.isolator.total.heap.usage.limit	String	整体堆内存使用率阈值。默认75%，表示当堆内存整体使用率达到75%时，通过优先级策略熔断隔离池中资源消耗最大的查询。取值范围：0.0 ~ 100.0%。
search.isolator.total.tasks.limit	Integer	慢查询隔离池最多允许处理的任务数。默认1000，表示隔离池中同时处理的查询数超过1000时，通过优先级策略熔断隔离池中资源消耗最大的查询。取值范围：10 ~ 50000。

设置中断隔离池中某一条查询任务的选取策略

```
PUT /_cluster/settings
{
 "persistent": {
  "search.isolator.strategy": "fair",
  "search.isolator.strategy.fair.fair_ratio": "1%"
 }
}
```

参数说明

参数名	类型	说明
search.isolator.strategy	String	中断隔离池中任务的选取策略，可选值：fair、mem-first、time-first。 - mem-first：中断堆内存使用最大的任务。 - time-first：中断运行时间最长的任务。 - fair：综合内存与时间。当任务堆内存差异小于“最大堆内存 * ratio”时，优先中断时间长的任务，否则优先中断内存大的任务。默认值：fair。
search.isolator.strategy.fair_ratio	String	fair策略的内存阈值，适用于search.isolator.strategy为fair时。相差内存小于此阈值时，中断运行时间长的任务，反之则中断大内存任务。默认值：1%，范围：0.0%-100.0%。

配置全局超时特性

```
PUT /_cluster/settings
{
 "persistent": {
  "search.isolator.time.limit": "10s"
 }
}
```

参数说明

参数名	数据类型	参数说明
search.isolator.time.limit	String	全局查询任务超时设置。当全局查询超时功能开启时，所有已创建的查询任务超过此时长将被取消执行。默认值：120s，取值范围：≥ 0ms

监控

- 获取隔离池任务

```
GET /_tasks/isolator
GET /_cat/isolator_tasks
```

- 获取隔离池统计信息

```
GET /_nodes/isolator_stats
```

快速入门

集群资源评估

使用百度智能云Elasticsearch前，需要优先结合业务需求和所存数据，评估集群所需的资源容量，包括磁盘容量、单机规格、shard大小和数量等。评估方式如下：

节点数量评估

建议至少选择3个节点，保证ES集群具有较高的节点故障容错能力和更好的可用性。

磁盘评估

磁盘类型选择

SSD云磁盘：适合对性能要求高的场景，具有高IO和高吞吐，适合数据响应度较高的在线分析和搜索场景。核心业务系统、大型线上查询业务**建议**选择SSD云磁盘。

高性能云磁盘：性能比SSD云磁盘低，价格相对低廉。适合线下测试、大规模冷数据存储分析等对性能要求不敏感的场景。

磁盘容量评估

用户考虑磁盘空间时主要考虑以下几方面：

- 影响ES磁盘空间大小的因素包括：
  - 副本分片数量：副本可以增加数据的可靠性和查询并发量，建议副本分片数量至少设置为1(number_of_replicas=1)。
  - 索引数据膨胀开销：除原始数据外，ES 需要存储索引、列存数据等，通常比源数据大10%-50%（`_all`参数等未计算）。
  - 内部其他开销：用于 segment 合并、ES Translog、日志等内部操作，预留20%。

- 安全阈值：通常至少预留20%~30%的安全阈值。

因此，数据保存在 ES 中实际数据量大概为：

```
实际数据量 = 源数据 * (1 + 副本数量) * (1 + 数据膨胀) * (1 + 内部其他开销)
    ≈ 源数据 * (1 + 副本数量) * 1.4
```

计算副本数量和安全阈值预留空间后，建议申请的存储容量为：

```
存储容量 = 源数据 * (1 + 副本数量) * 1.4 * (1 + 预留空间)
    ≈ 源数据 * (1 + 1) * 1.7
    ≈ 源数据 * 3.4
```

> - 索引数据膨胀具体开销可以使用“mapper-size”插件测试，使用方式：创建索引mapping时设置“_size:{"enable":true}”。
> - `_all`参数默认禁用，如果业务上没有特殊需求，建议无需开启。
> - 开启`_all`参数，磁盘容量的开销也会增大，在评估磁盘空间时需要相应增加。

单机规格评估

单机规格的大小会影响百度智能云Elasticsearch集群性能。建议参考以下几点来评估集群单机规格。

- 如果需要使用NLP中文分词或向量检索插件，建议单节点内存在8G以上。
- 使用场景不同，单节点磁盘最大容量也会不同。Elasticsearch实例能够容载的数据量与JVM的堆内存大小有关，一般JVM的堆内存会使用50%左右用于处理数据，建议单节点磁盘最大容量按照JVM内存来评估，建议比例如下：

场景	说明	单节点磁盘最大容量
搜索场景	大量查询聚合、数据计算的业务场景	JVM的堆内存（G）* 0.5 * 50
日志场景	大量日志写入、离线分析的业务场景	JVM的堆内存（G）* 0.5 * 100

除了小内存机型外，通常情况下BES将JVM的堆内存设置为节点内存的一半，最大不超过30G。集群规格参考示例如下：

规格名称	CPU（核）	内存（GB）	单节点磁盘最大容量
bes.g3.c2m8	2	8	搜索场景：100 GB 日志场景：200 GB
bes.g3.c4m16	4	16	搜索场景：200 GB 日志场景：500 GB
bes.g3.c8m32	8	32	搜索场景：500 GB 日志场景：1 TB
bes.g3.c16m64	16	64	搜索场景：1 TB 日志场景：2 TB

shard 相关评估

在进行shard规划前，用户需要考虑:

> - 存量数据和增量数据的量级

> - 数据的写入TPS和查询QPS
> - 集群规模和单节点规格

基于以上考虑，对shard规划提供了一些参考建议。

- 主分片数目在创建索引后不可动态更改，所以建议在创建索引之前就对主分片数量进行合理规划。
- 建议shard的个数（包括副本）要尽可能等于数据节点数，或者是数据节点数的整数倍，保证shard可以在各个节点间均匀分布。
- 通常业务场景下建议单个shard大小保持在**10-30GB**，日志场景可以保持在20-50GB。分片不宜过大或过小：过大可能使 ES 的故障恢复速度变慢、查询性能降低；过小可能导致非常多的分片，分片越多，ES性能开销也越大，从而导致读写性能差、内存不足等问题。
- 建议单节点上的shard数目控制在**500个以下**，7.x版本的实例默认的单节点shard数量上限为1000个（官方不建议调整）。如果规划中单节点shard数量超过上限，建议调整shard规划或扩容节点来降低单节点的shard数量。
- 建议单个节点上同一索引的shard个数不要超5个。
- 当数据量和写入量较大时，为了降低 ES 压力，建议选择多主分片+单副本分片的模式；当数据量较小，且写入量也比较小时，建议使用单主分片+多副本分片或者单主分片+单副本分片的模式。

创建集群

前提条件

已创建百度智能云账号。创建账号参考百度云账号注册流程。

功能入口

进入百度智能云首页。

在搜索框中输入”Elasticsearch“，点击进入百度智能云 Elasticsearch 产品首页。

点击“立即购买”，进入创建集群页。

操作步骤

接下来将具体说明创建集群流程。

集群和节点配置

详细说明创建集群中的配置项，包括付费及地域、基础配置、网络与可用区、节点配置和其他配置。

付费及地域

- 当前地域：百度智能云Elasticsearch 已在多个地域上线，其中金融华中-武汉需要联系 售前服务 咨询详情。
- 付费方式：支持预付费和后付费两种模式。
  - 预付费:一般为包年包月的购买形式，先付费后使用服务。用户根据自身对云资源的使用需求，一次性支付一个月、多个月或多年的费用，支付成功后，云资源将被系统分配给用户使用，直到到期后用户未继续续费而被回收。预付费模式价格相较于后付费模式更低。
  - 后付费：按量计费，即先按需申请资源使用，结算时会按用户的实际资源使用量和使用周期收取费用。结算时如果账户余额不足，客户账号将进入欠费状态，需要在约定时间完成缴费，超过约定时间，所使用资源将被关停并收回资源。后付费模式按分钟计费，按小时扣费。不需要提前支付费用，但需要保证账户余额大于100元。后付费比预付费模式价格高一些。

基础配置

- 集群名称：用户自定义，支持1-30个英文大小字母、数字、下划线（_）或连字符（-）。
- 管理员名称：默认为superuser，不支持更改，用于访问BES集群和登录Kibana。创建集群后，可使用suoeruser账号新增集群管理员和用户。
- 管理员密码：管理员密码请按照规则设置，并妥善保存，如果密码忘记，可在集群详情页重置。
- HTTPS协议：若不开启则集群默认使用HTTP协议，启用后将应用HTTPS协议提升数据安全

网络与可用区

- 可用区部署数量：可选单可用区、双可用区和三可用区。
  - 单可用区指 ES 集群部署在一个可用区内；
  - 双可用区指 ES 集群部署在同一个区域的两个可用区内。双可用区部署方式下可选为专用主节点单独部署一个可用区，同时需要单独配置一个子网；
  - 三可用区是ES集群部署在同一个地域的三个可用区中。在索引配置了副本的情况下，当某一可用区出现故障时，剩余的可用区依然可以不间断地提供服务，可用区的部署方式可以提高集群的容灾能力，保障线上业务的稳定性。
- 网络类型：ES部署在私有网络VPC内，只有在同一个 VPC下才能访问 ES 集群，因此为保证内网顺利访问，建议选择已有云上业务的区域位置所在 VPC。多可用区部署模式下，也是选择同一个 VPC。VPC创建详见私有网络VPC。
- 可用区与子网：根据在上方选择的可用区部署数量，选择可用区，每个可用区配置一个子网，同一个 VPC 内，不同可用区子网之间是互通的。子网创建详见子网。若所选子网的IP数量不足，无法正常创建集群。请保证子网IP数量充足，再创建集群。

节点配置

- ES版本：百度智能云支持7.10.2、7.4.2、6.8.14、6.5.3、5.6.16、5.5.0版本。推荐选择7.10.2版本。页面可选7.10.2、7.4.2、6.8.14、6.5.3版本，如需其他版本，请提工单开通。
- 类型：百度智能云ES提供数据节点、Kibana节点、专用主节点、冷数据节点和协调节点可供选择。
  - 数据节点：数据节点用于储存索引分片，进行数据处理相关工作。默认选中3个数据节点，用户可根据业务规模修改配置。

    > 说明：一个集群最多可以配置150个数据节点，单可用区和双可用区集群最少配置2个数据节点，三可用区集群最少配置3个数据节点。

  - Kibana节点：Kibana节点提供 Kibana 服务。百度智能云 ES 默认为用户赠送一个1核2 GB 规格的 Kibana 节点。也支持用户购买更高规格的 Kibana 节点。说明：由于1核2GB的套餐资源量小，可能存在稳定性问题，建议用户根据对 Kibana 的依赖程度选择更高规格的套餐。
  - 专用主节点：专用主节点是ES集群一种特定类型的节点，不存储数据，只作为主节点，用于保障集群稳定性。专用主节点默认未配置，购买时先启用专用主节点，再继续进行节点资源配置。
  - 冷数据节点：冷数据节点用于量比较大但对查询性能要求较低的“冷”数据，帮助用户节省存储成本。当业务索引具备明显的冷热区分度，建议创建冷数据节点，分离冷热数据。冷数据节点默认未配置，请先启用冷数据节点，再进行节点资源配置。功能详见基于节点磁盘介质的冷热数据分离。
  - 协调节点：专用协调节点用于接收API请求，分担数据节点的CPU开销，从而提高处理性能和服务稳定性。如果您的业务是CPU密集型的业务，建议购买协调节点。协调节点默认未配置，请先启用协调节点，再继续进行节点资源配置。
- 节点配置：支持通用型、计算型、密集计算型和内存型，用户可根据业务场景进行选择不同规格，选择建议可参见集群资源

评估。不同规格售卖价格参见产品定价。

- 数据盘：
  - 数据节点用户可选增强型SSD_PL2，增强型SSD_PL1，SSD云盘和高性能云盘，并根据业务需要配置资源量。详细性能对比参考磁盘性能。

    > 注意：增强型SSD_PL2 仅支持CPU大于等于16G的部分套餐。用户如有需要，请 提工单 开通。增强型SSD_PL2支持性能预配置，用户可以配置额外性能

  - 冷数据节点用户可配置高效云盘资源量。
  - 专用主节点可选20GB的增强型SSD_PL1或者SSD云盘。
  - 协调节点配有20GB的高效云盘。
  - Kibana节点不支持配置数据盘。
- 购买数量：默认选中3个数据节点和1个Kibana节点，Kibana节点数量不可更改， 为保障服务高可用，建议在生产环境中，数据节点数量至少为3。其他节点的购买数据支持用户根据业务需求进行选择。

百度智能云ES支持在创建集群后进行配置变更。

购买时长

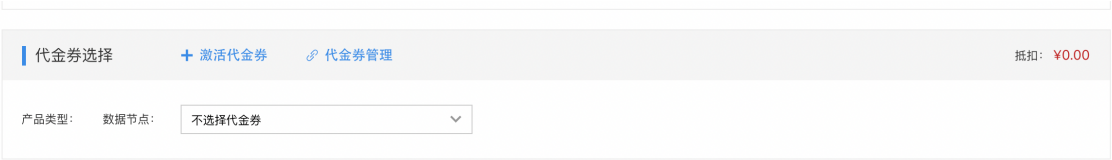
- 购买时长：预付费模式可选择购买时长。默认购买时长为1个月，可选：1~9月、1~3年。
- 自动续费：预付费模式支持设置自动续费功能（默认关闭），可在购买页面通过勾选到期自动续费开启。更多功能说明详见自动续费。

其他配置

- 绑定标签：标签支持按各种标准（如用途、所有者或项目）对资源进行分类；每个标签包含键和值两部分。
- 资源分组：支持按照其企业组织、业务场景、项目等不同维度统一管理资源；资源分组不可为空，默认选中为【默认分组】，可通过下拉框选择其他已有分组。如需新的资源分组，资源分组创建详见资源管理创建。

确认订单

点击“下一步”，进入订单确认页面，用户需要确认一下订单价格，如果有代金券在此处添加代金券。



在线支付

点击“提交订单”后，进入下一步：

- 预付费方式，点击“提交订单”后进入订单支付页，核对订单并选择支付方式，确认无误后点击“确认支付”进行支持。返回 ES 控制台，在集群列表页可以查看到创建的集群。
- 后付费方式，点击“提交订单”后直接创建集群，不需要确认订单支付。在使用过程中，会按小时进行计费及扣费。

创建成功

大约1分钟之后，集群创建工作完成，点击“管理控制台”即可进入集群管理界面。点击集群名称，可以查看集群的详细信息。

后续操作

可以通过控制台对集群进行各种管理操作，包括删除集群、配置变更、修改密码等。

进入集群详情页可以查看各节点的运行情况以及集群的HTTP/HTTPS的连接URL等信息。云上不提供基于tcp transport方式的访问形式，只能使用http/https方式的restful api访问。

同时百度智能云ES也提供配置修改、插件管理、定时调度、日志管理、监控、操作历史和数据量观测等丰富功能，便于用户对ES集群进行高效管控。

访问Elasticsearch服务

Elasticsearch集群搭建好之后，用户可以通过Kibana访问集群，也可以通过curl命令访问集群，然后进行数据的导入或查询。在使用Elasticsearch服务前，用户需要注意以下几点：

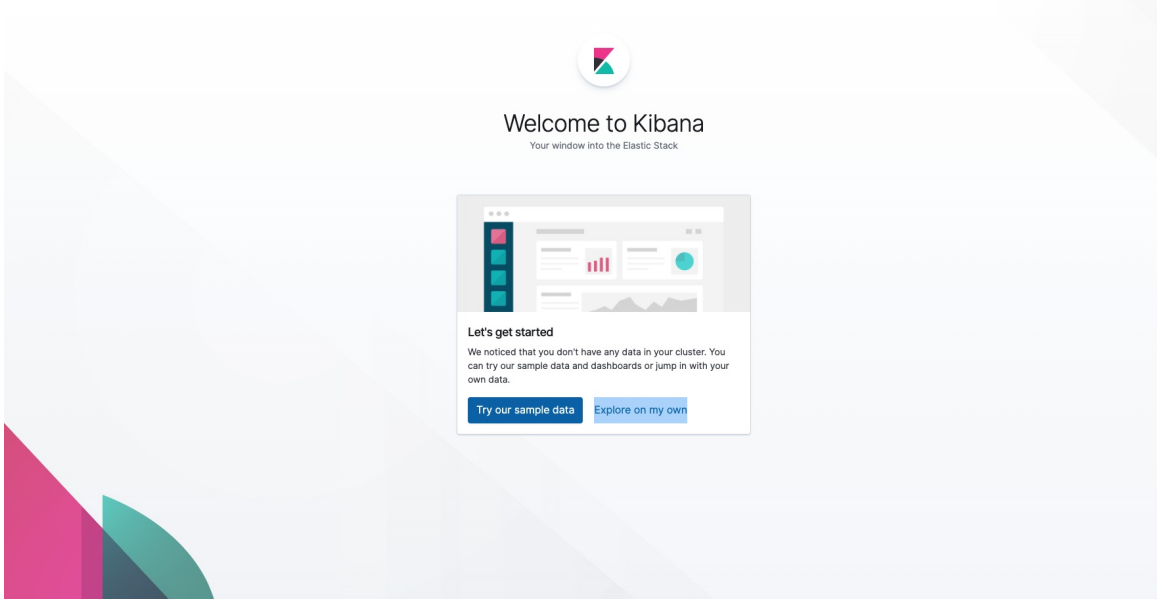
- 由于Elasticsearch集群创建的时候在用户定义的VPC内，所以只有属于同一个VPC内的bcc才能直接访问Elasticsearch服务，而不同VPC内的节点则需要通过对等连接等方式进行访问。
- Elasticsearch集群默认只有一个superuser用户，superuser用户的密码就是用户创建集群时指定的管理员密码。您也可以创建新的Elasticsearch用户，创建方法参见权限管理。
- 连接Elasticsearch服务的方式可参见Elasticsearch Restful API。

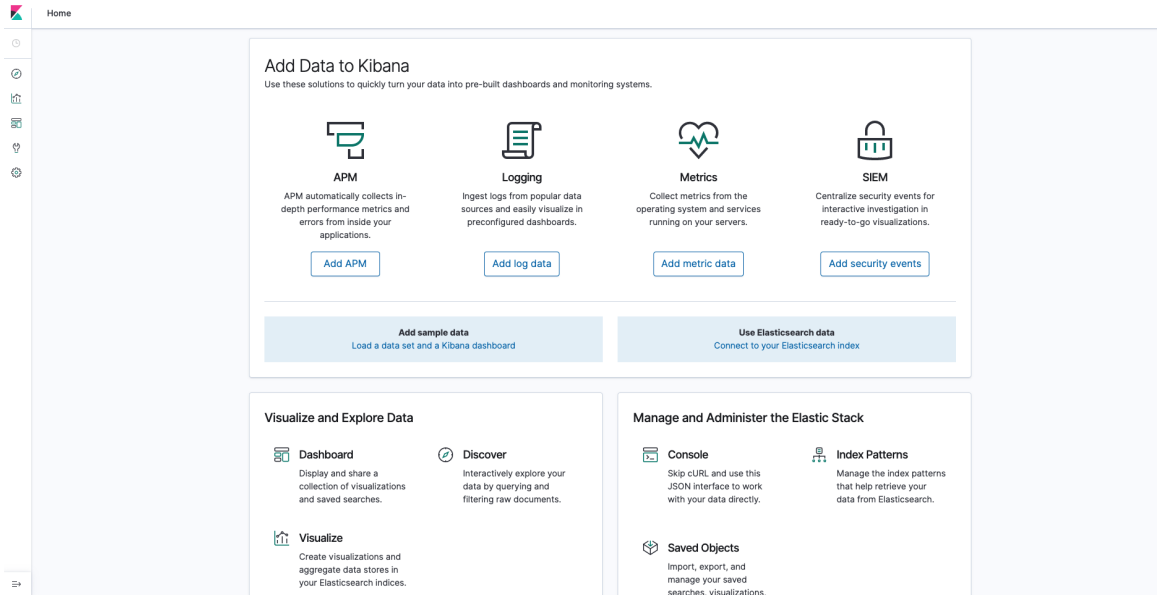
通过Kibana访问

下面介绍如何通过Kibana来访问Elasticsearch服务。

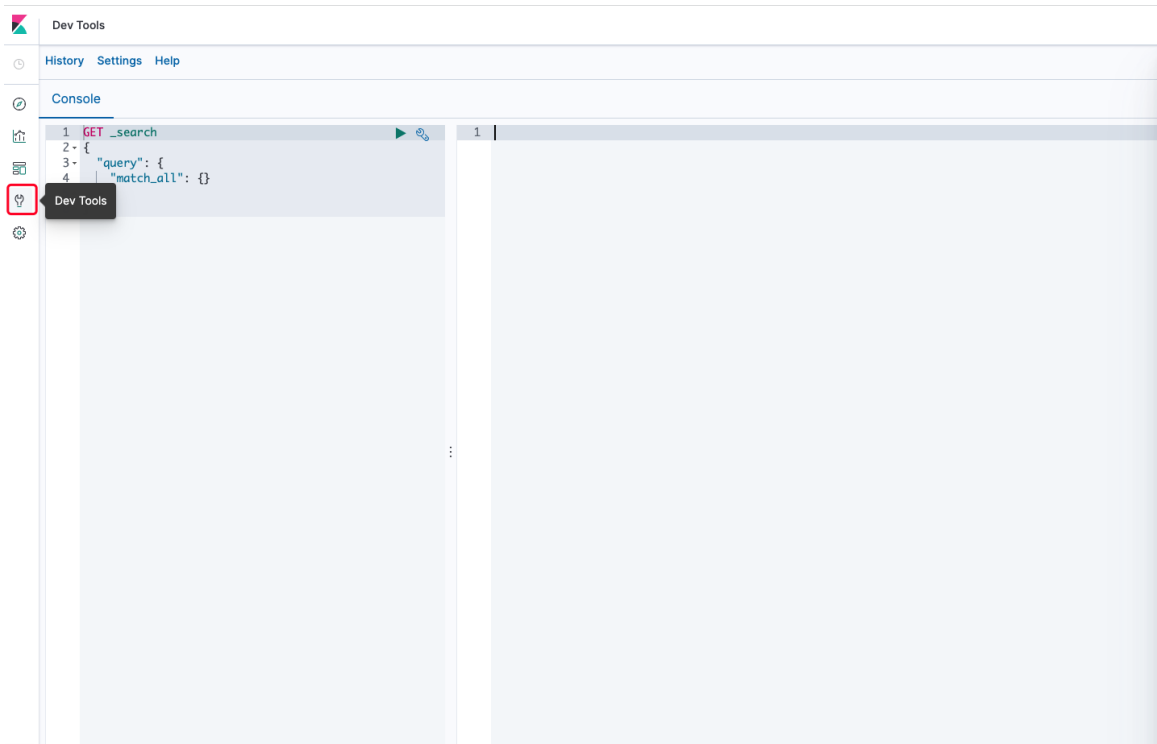
创建百度智能云Elasticsearch集群会默认附赠一个Kibana节点。等待实例状态变成运行中，就可以通过Kibana访问集群，具体访问步骤如下：

1. 在集群列表中，单击目标集群名称进入集群详情页。
2. 在集群详情页左侧导航栏默认选中集群信息，集群信息的内容区域展示了Kibana的地址信息。
3. 在浏览器打开上述Kibana地址，然后登录Kibana（Kibana用户名密码与百度智能云Elasticsearch服务的用户名密码相同）。
4. 登录成功之后，点击Explore on my own，就可以进入Kibana可视化管理页面。



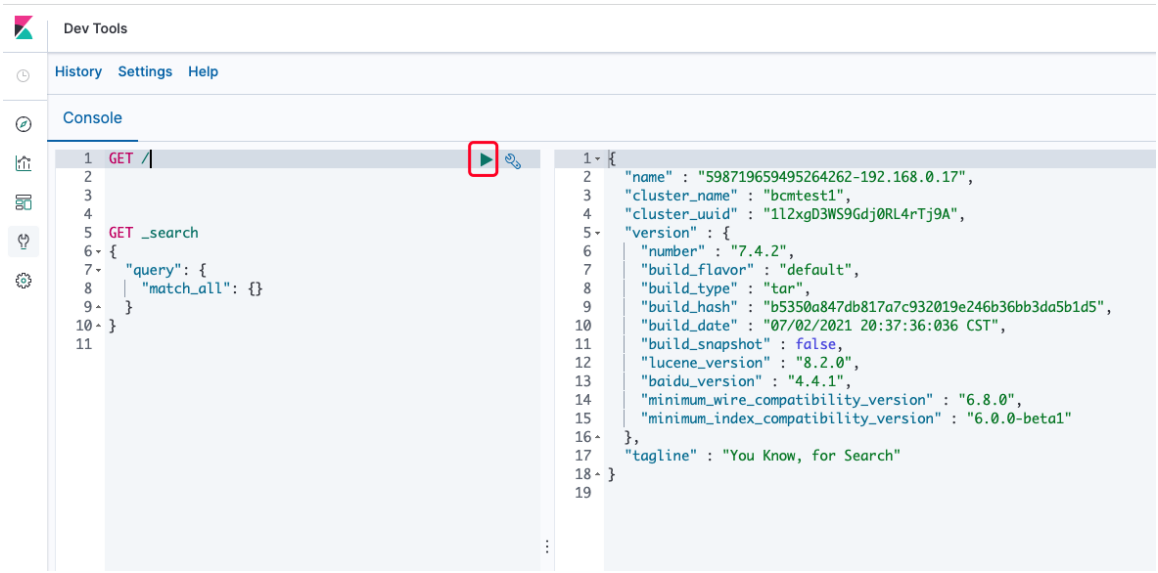


5. 在Kibana可视化管理页面，单击Dev Tools（开发工具，位于左侧导航栏中），若需要输入密码，输入上述相同的用户名和密码。



6. 在Console下执行如下命令可以访问Elasticsearch集群，获取结果在右侧栏展示。

```
GET /
```



通过curl命令访问

下面以Linux下的curl命令为例来介绍如何访问Elasticsearch服务。

认证方式

例如使用 Http Basic认证方式访问集群：

```
curl -u {username}:{password} --header 'Content-Type: application/json' -XGET '{host}:{port}/'
```

使用URL参数认证：

```
curl -XGET '{host}:{port}/?username={username}&password={password}'
```

创建 index

通过Elasticsearch的创建 index API 带上用户名和密码来创建新的 index：

```
curl -u {username}:{password} --header 'Content-Type: application/json'  -XPUT '{host}:{port}/index_name' -d'
{
  "settings": {
    "index": {
      "number_of_shards": 1,
      "number_of_replicas": 9
    }
  }
}'
```

导入数据

实时导入：

指定id：

```
curl -u {username}:{password} --header 'Content-Type: application/json'  -XPOST '{host}:{port}/index_name/_doc/id' -d '
{
    "field": "value"
}'
```

不指定id：

```
curl -u {username}:{password} --header 'Content-Type: application/json'  -XPOST '{host}:{port}/index_name/_doc/' -d '
{
    "field": "value"
}'
```

id为该条数据的唯一id，如果不指定id，则会自动随机生成一个id。

批量导入：

```
curl -u {username}:{password} --header 'Content-Type: application/json'  -XPOST '{host}:{port}/index_name/_doc/_bulk?pretty' -d '
{"index":{"_id":"1"}}
{"name": "John Doe" }
{"index":{"_id":"2"}}
{"name": "Jane Doe" }
'
```

每两行为一条数据，必须用换行符分隔。

使用如下命令导入文件：

```
curl -u {username}:{password} --header 'Content-Type: application/json'  -XPOST '{host}:{port}/index_name/_doc/_bulk' --data-binary @import.json
```

其中*import.json*为文件名，单个文件不能超过100M，大文件可分割为小文件并行导入。

查询数据

列出集群中所有的 index：

```
curl -u {username}:{password} --header 'Content-Type: application/json'  -XGET '{host}:{port}/_cat/indices?v'
```

查询 index 中的数据：

```
curl -u {username}:{password} --header 'Content-Type: application/json'  -XGET '{host}:{port}/testindex/_search?pretty' -d '
{
    "query": { "match_all": {} }
}'
```

关于Elasticsearch其他详细API介绍，请参考Elasticsearch官方文档：

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

导入数据并搜索

前提条件

开通百度智能云Elasticsearch服务（参考创建集群）且可访问Elasticsearch集群（参考访问Elasticsearch服务）后，可以通过

附赠的kibana节点访问Elasticsearch集群，然后可以创建索引、导入数据和搜索数据。

创建索引

在导入数据之前，可以评估shard和创建索引，以下示例创建一个名称为index_name的索引。

```
PUT index_name
{
  "settings": {
    "index": {
      "number_of_shards": 1,
      "number_of_replicas": 1
    }
  }
}
```

其中，number_of_shards指主分片数，number_of_replicas指每个主分片有多少个副本分片数。Elasticsearch 7.0以下版本默认创建5个主分片和1个副本分片，Elasticsearch 7.0及以上版本默认创建1个主分片和1个副本分片。主分片数只能在索引创建前指定，且索引创建后不能动态修改，只能通过reindex或split等方式达到修改目的，副本分片数可以动态修改。

创建成功，返回如下结果。

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "index_name"
}
```

返回结果各个字段的含义如下所示。

返回字段	含义
acknowledged	索引在集群中是否创建成功
shards_acknowledged	每个分片组在超时之前是否激活了所需数量的分片
index	索引名称

注：返回值中acknowledged或者shards_acknowledged为false时，索引创建操作仍然可能成功，这两个值只是表示操作是否在超时之前完成。假设acknowledged为false，新创建的索引更新集群状态时超时，创建索引操作仍然可能会在之后执行成功。假设shards_acknowledged为false，新创建的索引成功更新集群状态，但激活所需数量的分片超时，默认只启动主分片。

导入数据

导入单条数据 以下示例创建单个文档并导入数据。

```
PUT index_name/_doc/1
{
  "name": "Amber Adams",
  "age": 32
}
```

导入数据成功，返回如下结果。

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

返回结果各个字段的含义如下所示。

返回字段	含义
index	索引名称
_type	文档类型
_id	文档唯一标识符
_version	文档版本，每次更新文档，版本都会增加
result	操作结果
_shards	被操作的分片总数量、成功数量、失败数量
_seq_no	文档序列号，序列号用于确保旧版本文档不会覆盖新版本
_primary_term	主项

可以使用如下命令查询数据来进行验证。

```
GET index_name/_search
{
"query": { "match_all": {} }
}
```

导入多条数据 以下示例使用_bulk API创建多个文档并导入数据。

```
POST _bulk
{"index":{"_index": "index_name", "_id": 2}}
{"name": "Fulton Rich","age": 40}
{"index":{"_index": "index_name", "_id": 3}}
{"name": "Virginia Ayala", "age": 12}
{"index":{"_index": "index_name", "_id": 4}}
{"name": "Mcgee Mooney", "age": 23}
{"index":{"_index": "index_name", "_id": 5}}
{"name": "Burton Meyers", "age": 60}
```

可以使用如下命令查询插入的数据来进行验证。

```
GET index_name/_search
{
"query": { "match_all": {} }
}
```

搜索数据

向集群导入数据之后，可以进行数据搜索，比如查询索引、全文搜索、按条件搜索等。 **查询索引** 如下命令查询集群的所有索引。

```
GET _cat/indices?v
```

查询结果如下所示。

```
health status index          uuid                   pri rep docs.count docs.deleted store.size pri.store.size
green open   index_name      GKL71ejiT0CyqJZgVgkMjA   3   1          5            0     11.2kb         11.2kb
```

查询结果各个字段的含义如下所示。

返回字段	含义
health	索引健康状态
status	索引的开启状态
index	索引名称
uuid	索引uuid
pri	索引主分片数
rep	索引副本分片数量
docs.count	索引中文档总数
docs.deleted	索引中删除状态的文档
store.size	主分片+副本分片的大小
pri.store.size	主分片的大小

全文搜索 如下命令搜索name包含Ayala的文档数据。

```
GET index_name/_search
{
  "query": {
    "match": {
      "name": "Ayala"
    }
  }
}
```

搜索结果如下所示。

```
{
 "took" : 10,
 "timed_out" : false,
 "_shards" : {
   "total" : 3,
   "successful" : 3,
   "skipped" : 0,
   "failed" : 0
 },
 "hits" : {
   "total" : {
     "value" : 1,
     "relation" : "eq"
   },
   "max_score" : 0.9808292,
   "hits" : [
     {
       "_index" : "index_name",
       "_type" : "_doc",
       "_id" : "3",
       "_score" : 0.9808292,
       "_source" : {
         "name" : "Virginia Ayala",
         "age" : 12
       }
     }
   ]
 }
}
```

搜索结果各个字段的含义如下所示。

返回字段	含义
took	执行搜索耗时，单位为毫秒
time_out	搜索是否超时
_shards	多少分片被搜索，成功多少，跳过多少，失败多少
hits	搜索结果展示
hits.total	匹配条件的文档总数
hits.max_score	最大匹配得分
hits._score	返回文档的匹配得分（得分越高，匹配程度越高，越靠前）
hits.hits	返回结果展示

按条件搜索 如下命令搜索age在20-30岁之间(闭区间)的文档数据。

```
GET index_name/_search
{
 "query": {
   "range": {
     "age": {
       "gte": 20,
       "lte": 30
     }
   }
 }
}
```

搜索结果如下所示。

```
{
 "took" : 18,
 "timed_out" : false,
 "_shards" : {
   "total" : 3,
   "successful" : 3,
   "skipped" : 0,
   "failed" : 0
 },
 "hits" : {
   "total" : {
     "value" : 1,
     "relation" : "eq"
   },
   "max_score" : 1.0,
   "hits" : [
     {
       "_index" : "index_name",
       "_type" : "_doc",
       "_id" : "4",
       "_score" : 1.0,
       "_source" : {
         "name" : "Mcgee Mooney",
         "age" : 23
       }
     }
   ]
 }
}
```

搜索结果各个字段的含义同上。

Elasticsearch

用户手册

集群管理

集群创建

前提条件

已创建百度智能云账号。创建账号参考百度云账号注册流程。

功能入口

进入百度智能云首页。

在搜索框中输入”Elasticsearch“，点击进入百度智能云 Elasticsearch 产品首页。

点击“立即购买”，进入创建集群页。

操作步骤

接下来将具体说明创建集群流程。

集群和节点配置

详细说明创建集群中的配置项，包括付费及地域、基础配置、网络与可用区、节点配置和其他配置。

付费及地域

- 当前地域：百度智能云Elasticsearch 已在多个地域上线，其中金融华中-武汉需要联系 售前服务 咨询详情。
- 付费方式：支持预付费和后付费两种模式。
  - 预付费:一般为包年包月的购买形式，先付费后使用服务。用户根据自身对云资源的使用需求，一次性支付一个月、多个月或多年的费用，支付成功后，云资源将被系统分配给用户使用，直到到期后用户未继续续费而被回收。预付费模式价格相较于后付费模式更低。
  - 后付费：按量计费，即先按需申请资源使用，结算时会按用户的实际资源使用量和使用周期收取费用。结算时如果账户余额不足，客户账号将进入欠费状态，需要在约定时间完成缴费，超过约定时间，所使用资源将被关停并收回资源。后付费模式按分钟计费，按小时扣费。不需要提前支付费用，但需要保证账户余额大于100元。后付费比预付费模式价格高一些。

基础配置

- 集群名称：用户自定义，支持1-30个英文大小字母、数字、下划线（_）或连字符（-）。
- 管理员名称：默认为superuser，不支持更改，用于访问BES集群和登录Kibana。创建集群后，可使用suoeruser账号新增集群管理员和用户。
- 管理员密码：管理员密码请按照规则设置，并妥善保存，如果密码忘记，可在集群详情页重置。
- HTTPS协议：若不开启则集群默认使用HTTP协议，启用后将应用HTTPS协议提升数据安全

网络与可用区

- 可用区部署数量：可选单可用区、双可用区和三可用区。
  - 单可用区指 ES 集群部署在一个可用区内；
  - 双可用区指 ES 集群部署在同一个区域的两个可用区内。双可用区部署方式下可选为专用主节点单独部署一个可用区，同时需要单独配置一个子网；
  - 三可用区是ES集群部署在同一个地域的三个可用区中。在索引配置了副本的情况下，当某一可用区出现故障时，剩余的可用区依然可以不间断地提供服务，可用区的部署方式可以提高集群的容灾能力，保障线上业务的稳定性。
- 网络类型：ES部署在私有网络VPC内，只有在同一个 VPC下才能访问 ES 集群，因此为保证内网顺利访问，建议选择已有云上业务的区域位置所在 VPC。多可用区部署模式下，也是选择同一个 VPC。VPC创建详见私有网络VPC。
- 可用区与子网：根据在上方选择的可用区部署数量，选择可用区，每个可用区配置一个子网，同一个 VPC 内，不同可用区子网之间是互通的。子网创建详见子网。若所选子网的IP数量不足，无法正常创建集群。请保证子网IP数量充足，再创建集群。

节点配置

- ES版本：百度智能云支持7.10.2、7.4.2、6.8.14、6.5.3、5.6.16、5.5.0版本。推荐选择7.10.2版本。页面可选7.10.2、7.4.2、6.8.14、6.5.3版本，如需其他版本，请提工单开通。
- 类型：百度智能云ES提供数据节点、Kibana节点、专用主节点、冷数据节点和协调节点可供选择。
  - 数据节点：数据节点用于储存索引分片，进行数据处理相关工作。默认选中3个数据节点，用户可根据业务规模修改配置。

    > 说明：一个集群最多可以配置150个数据节点，单可用区和双可用区集群最少配置2个数据节点，三可用区集群最少配置3个数据节点。

  - Kibana节点：Kibana节点提供 Kibana 服务。百度智能云 ES 默认为用户赠送一个1核2 GB 规格的 Kibana 节点。也支

持用户购买更高规格的 Kibana 节点。说明：由于1核2GB的套餐资源量小，可能存在稳定性问题，建议用户根据对 Kibana 的依赖程度选择更高规格的套餐。

- 专用主节点：专用主节点是ES集群一种特定类型的节点，不存储数据，只作为主节点，用于保障集群稳定性。专用主节点默认未配置，购买时先启用专用主节点，再继续进行节点资源配置。
- 冷数据节点：冷数据节点用于量比较大但对查询性能要求较低的“冷”数据，帮助用户节省存储成本。当业务索引具备明显的冷热区分度，建议创建冷数据节点，分离冷热数据。冷数据节点默认未配置，请先启用冷数据节点，再进行节点资源配置。功能详见基于节点磁盘介质的冷热数据分离。
- 协调节点：专用协调节点用于接收API请求，分担数据节点的CPU开销，从而提高处理性能和服务稳定性。如果您的业务是CPU密集型的业务，建议购买协调节点。协调节点默认未配置，请先启用协调节点，再继续进行节点资源配置。

- 节点配置：支持通用型、计算型、密集计算型和内存型，用户可根据业务场景进行选择不同规格，选择建议可参见集群资源评估。不同规格售卖价格参见产品定价。
- 数据盘：
  - 数据节点用户可选增强型SSD_PL2，增强型SSD_PL1，SSD云盘和高性能云盘，并根据业务需要配置资源量。详细性能对比参考磁盘性能。

    > 注意：增强型SSD_PL2 仅支持CPU大于等于16G的部分套餐。用户如有需要，请 提工单 开通。增强型SSD_PL2支持性能预配置，用户可以配置额外性能

  - 冷数据节点用户可配置高效云盘资源量。
  - 专用主节点可选20GB的增强型SSD_PL1或者SSD云盘。
  - 协调节点配有20GB的高效云盘。
  - Kibana节点不支持配置数据盘。
- 购买数量：默认选中3个数据节点和1个Kibana节点，Kibana节点数量不可更改， 为保障服务高可用，建议在生产环境中，数据节点数量至少为3。其他节点的购买数据支持用户根据业务需求进行选择。

百度智能云ES支持在创建集群后进行配置变更。

购买时长

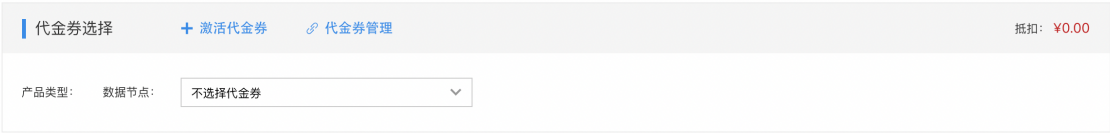
- 购买时长：预付费模式可选择购买时长。默认购买时长为1个月，可选：1~9月、1~3年。
- 自动续费：预付费模式支持设置自动续费功能（默认关闭），可在购买页面通过勾选到期自动续费开启。更多功能说明详见自动续费。

其他配置

- 绑定标签：标签支持按各种标准（如用途、所有者或项目）对资源进行分类；每个标签包含键和值两部分。
- 资源分组：支持按照其企业组织、业务场景、项目等不同维度统一管理资源；资源分组不可为空，默认选中为【默认分组】，可通过下拉框选择其他已有分组。如需新的资源分组，资源分组创建详见资源管理创建。

确认订单

点击“下一步”，进入订单确认页面，用户需要确认一下订单价格，如果有代金券在此处添加代金券。



在线支付

点击“提交订单”后，进入下一步：

- 预付费方式，点击“提交订单”后进入订单支付页，核对订单并选择支付方式，确认无误后点击“确认支付”进行支持。返回 ES 控制台，在集群列表页可以查看到创建的集群。
- 后付费方式，点击“提交订单”后直接创建集群，不需要确认订单支付。在使用过程中，会按小时进行计费及扣费。

创建成功

大约1分钟之后，集群创建工作完成，点击“管理控制台”即可进入集群管理界面。点击集群名称，可以查看集群的详细信息。

后续操作

可以通过控制台对集群进行各种管理操作，包括删除集群、配置变更、修改密码等。

进入集群详情页可以查看各节点的运行情况以及集群的HTTP/HTTPS的连接URL等信息。云上不提供基于tcp transport方式的访问形式，只能使用http/https方式的restful api访问。

同时百度智能云ES也提供配置修改、插件管理、定时调度、日志管理、监控、操作历史和数据量观测等丰富功能，便于用户对ES集群进行高效管控。

ES服务区域代码

区域	代码
北京	bj
保定	bd
苏州	su
广州	gz
香港	hkg
金融云武汉专区	fwh
新加坡	sin

集群重启

百度智能云 Elasticsearch 提供集群重启功能，支持用户通过控制台重启 Elasticsearch 实例。

前提条件

对于集群重启功能，百度智能云 Elasticsearch 提供以下操作建议：

- 如果选择滚动重启方式，请确保：
  - 集群健康状态为绿色。
  - 集群中不存在异常节点。
  - 除Kibana节点外，集群至少包括3个节点。
- 集群重启前，建议维持集群资源利用率可控（节点CPU利用率为80%左右），同时索引至少包含1个副本。

操作步骤

1. 登录百度智能云 Elasticsearch 控制台。
2. 在集群列表中，单击目标集群名称，进入集群详情页。
3. 在集群详情页上方，单击“重启”按钮。

4. 在重启弹窗中，确认集群ID和集群名称，并选择重启方式。BES提供两种重启方式：滚动重启和强制重启。请用户根据业务需求进行选择。

- **滚动重启：** 通过对各个节点依次重启实现集群重启。相比强制重启，滚动重启更稳定，如果集群整体负载不高且索引存在副本分片，则滚动重启不会影响集群服务。滚动重启耗时较长（单个节点恢复时间约为5-20min，具体视集群规模而定）。
- **强制重启：** 通过对所有节点全停全启方式实现集群重启。相比滚动重启，强制重启耗时较短（集群恢复时间约为5-20min，具体视集群规模而定），但会导致短暂停服。适用于对需要快速重启并允许短暂停服的业务。

注意： 当集群健康状态为黄色（yellow）或红色（red）、集群中存在异常节点或集群中只有两个节点（除Kibana节点外）时，不支持滚动重启操作，只支持强制重启。

5. 完成重启方式选择后，单击“重启”按钮，开始重启操作。自动跳转“操作历史”页面，可查看重启进度。

6. 重启过程中，集群状态为“正在重启”，健康状态为黄色（yellow）。

集群名称/ID	创建时间	状态	健康状态	支付方式	运行时间	操作
estest 600579701253214208	2022-01-10 14:49:23	正在重启	黄色	后付费	0 小时30 分钟	配置变更 续费

重启成功后，集群状态为“运行中”，健康状态为绿色（green）。

集群名称/ID	创建时间	状态	健康状态	支付方式	运行时间	操作
estest 600579701253214208	2022-01-10 14:49:23	运行中	绿色	后付费	0 小时46 分钟	配置变更 续费

注意事项

- 集群正在进行跨可用区迁移、配置变更、节点启停、计费方式变更等操作时，无法进行重启。请先将集群状态恢复为正常，再进行重启操作。
- 集群重启过程中，无法进行其他变更操作，如跨可用区迁移、配置变更、节点启停、计费方式变更等。
- 如果集群整体负载不高且索引存在副本分片，一般情况下滚动重启过程中可对外持续提供服务。但如果强制重启并发度高、集群负载很高并且已经存在集群访问不可用的情况、没有副本分片、在重启或强制重启过程中存在大量的写入和查询等场景，重启过程中可能会出现访问超时，建议重启前先在客户端设计好重试机制并且在业务低峰期进行操作。
- 当集群健康状态为黄色（yellow）或红色（red）、集群中存在异常节点、集群中只有两个节点（除Kibana节点外）时，不支持滚动重启操作，需要选择强制重启。

集群删除

如果您不再需要当前集群，可以进行删除集群操作。

注意：删除集群后数据将全部丢失且不可恢复，需要提前备份数据！

具体删除集群步骤如下：

1. 进入集群详情页，鼠标移动到更多操作出现删除集群的按钮，点击删除集群。

2. 点击删除集群后，会弹出确认删除的对话框，点确定即可删除集群。集群删除后将不再计费且不再出现在集群列表中。

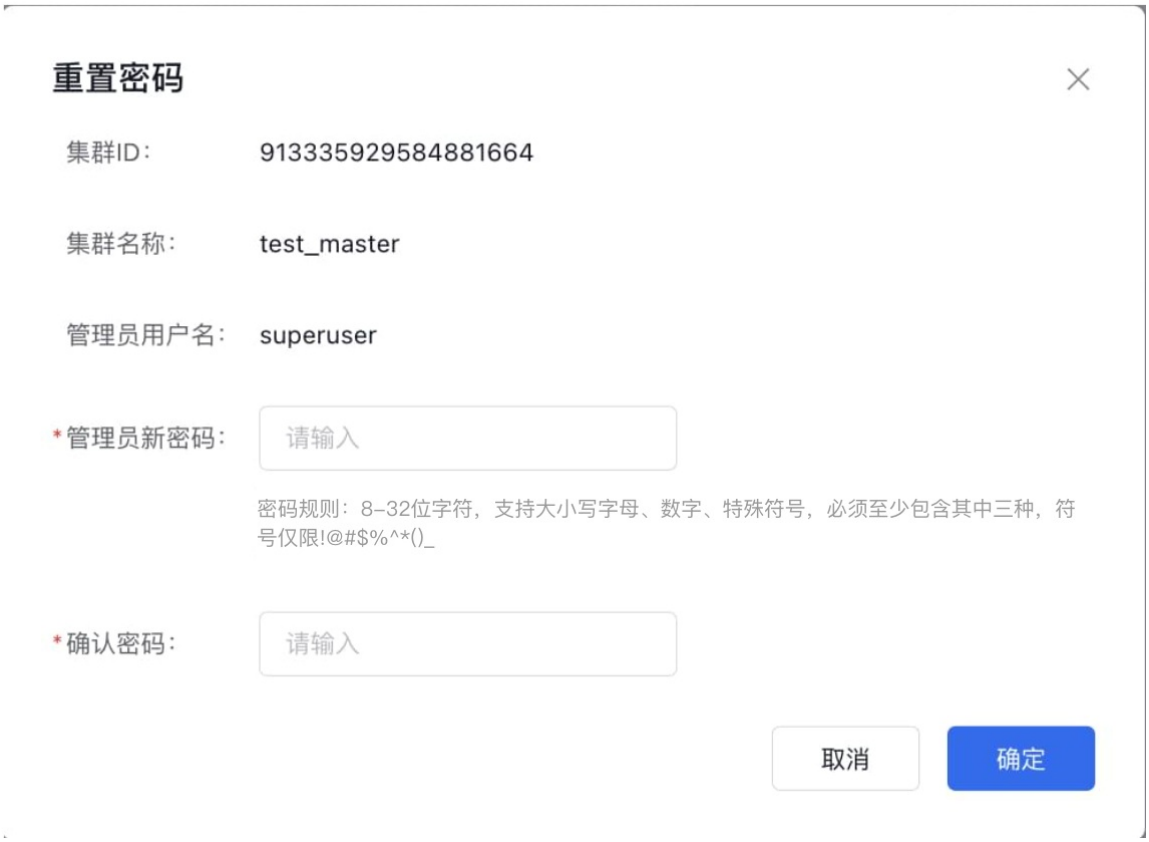
密码重置

百度Elasticsearch支持重置管理员 superuser 的密码。用户可以在控制台进行密码重置，也可以通过百度Elasticsearch的权限管理 API 完成密码重置，详情查看权限管理。

控制台重置密码具体步骤如下：

1. 进入集群详情页面，点击操作栏的 "重置密码"。

2. 在窗口中输入新的管理员密码，再次输入确认密码。



3. 点击确定后，需要通过手机短信进行身份验证，验证成功后密码重置完成。

> **注意：**
>
> 1. 新建集群的管理员密码需遵循新密码规则。新密码规则如下：8-32位字符，支持大小写字母、数字、特殊符号，必须至少包含其中三种，符号仅限!@#$%^*()_
> 2. 存量集群的管理员密码不受影响，密码规则不更新，集群更新到对应的内核版本才会更新密码规则。存量集群密码规则如下：支持小写字母(a-z)、大写字母(A-Z)、数字(0-9)、下划线(_)，并且至少包含其中三种，长度8-12位。

自动续费

预付费集群有两种续费方式：手动续费和自动续费。

> **注意：** 只有预付费集群支持续费，后付费集群不支持续费。

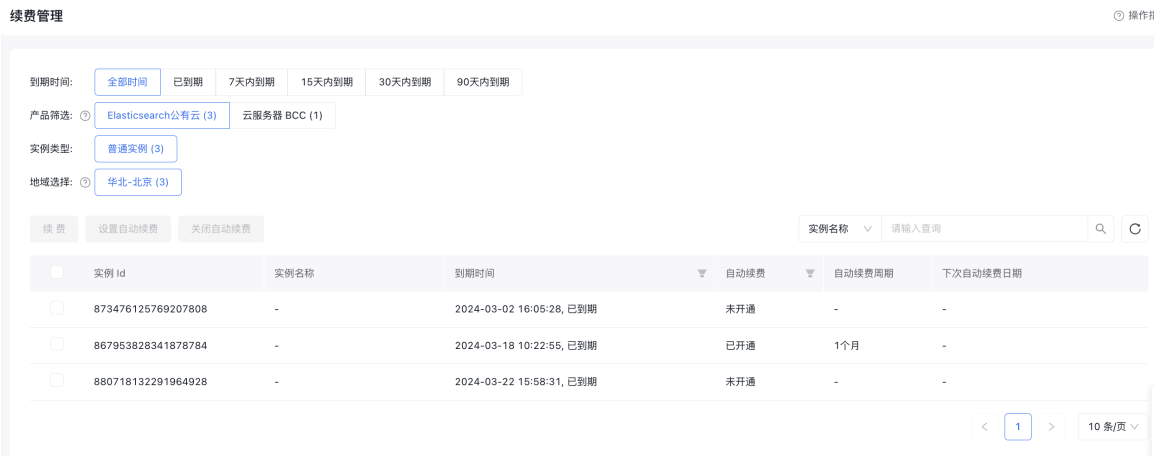
自动续费 自动续费支持在创建集群时开启，也可以在续费管理中开启。 **创建集群**

在创建集群时，选择付费方式为预付费后，在“购买时长”模板可以开启自动续费，设置续费周期，可以选择以月为单位和以年为单位进行自动续费。

续费管理

集群创建后如果还需要开启自动续费功能，可以从控制台操作栏进入续费管理页面。

选择Elasticsearch后可看到自己所有的预付费集群。



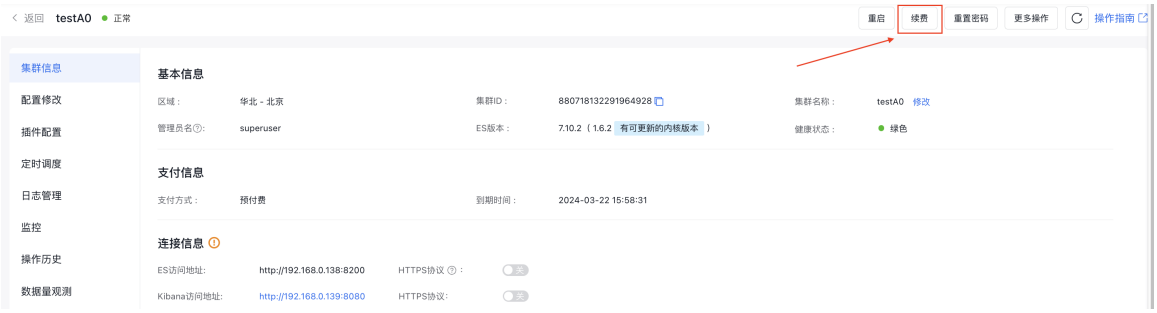
选中某个集群后点击设置自动续费。

选择自动续费的时长后点击下一步确认，最终开启自动续费。设置完毕后，该集群会在上次订单到期的7天之内自动续费，用户需确保账户金额足够。

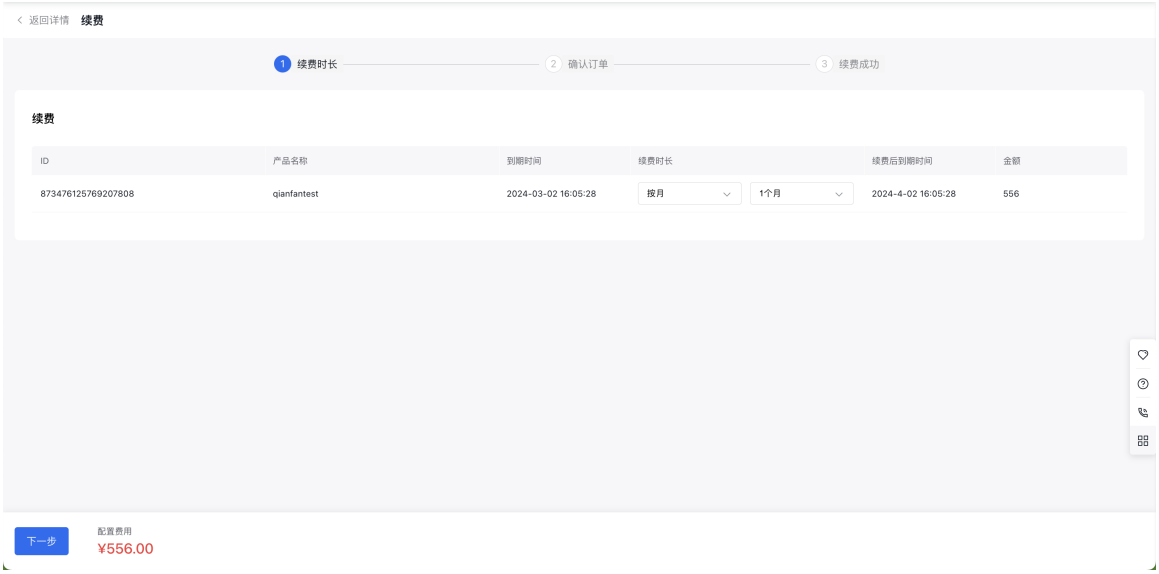
手动续费 预付费集群还支持手动续费，进入续费页面有两个入口：

- 集群列表页面，在“操作列”选择目标集群的集群点击“续费”。

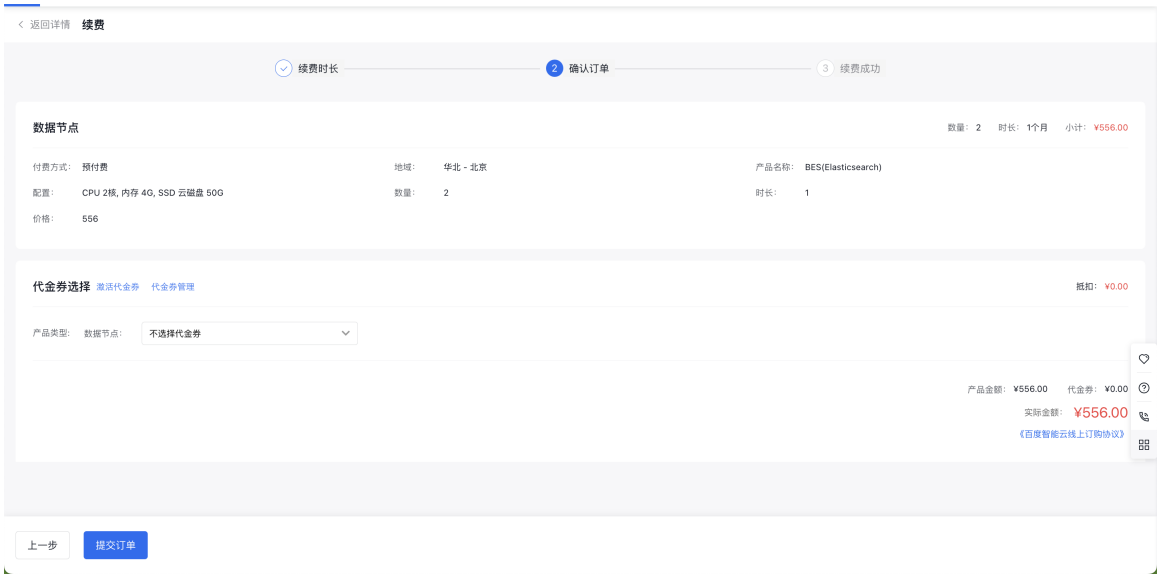
- 进入集群详情页面，在上方操作栏点击“续费”。



进入续费页面，可以选择续费的时间，支持以月为单位和以年为单位进行续费，右下角会展示续费所需支付的价格。



点击下一步，进入确定订单页面，可以选择使用代金券进行折扣。



提交订单后，集群续费成功。

公网绑定

百度智能云Elasticsearch支持通过外网访问Elasticsearch或Kibana。

操作步骤

1. 首先进入弹性公网IP产品页面创建一个公网IP。
2. EIP创建成功后，回到BES。点击需要进行绑定的集群名称，进入集群详情页。在集群信息的网络配置与可用区模块中，可以选择将创建的公网IP绑定到“公网访问IP”或“ Kibana 公网访问IP”上。

3. 完成绑定后，等待一段时间刷新页面，即可成功通过外网访问该集群或Kibana。

> **注意**
>
> 开通公网访问会大大降低安全性，因为一旦 Elasticsearch 的用户名密码泄露，任何人都能通过公网地址使用您的 Elasticsearch 服务。

跨可用区迁移集群

百度智能云Elasticsearch提供了跨可用区迁移集群的功能。用户在遇到可用区资源不足的情况或由于业务架构变更需要更换可用区时，可以将集群迁移到其他可用区。

使用方式

准备工作

1. 需要确保目标可用区已经创建好相应的子网。
2. 确保集群处于健康状态(`green`)，如图所示。

3. 确保集群内**没有close的索引**，查询方式如下：

```
GET /_cat/indices?v
```

```
health status index          uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   test_bos       Wsd_K_sZRpST6tNx9DVssQ   1   1         19            0      7.7kb          7.7kb
yellow open   test           YNjlxgyxQX-W9e8nfm6V-w   1   1          1            0      3.4kb          3.4kb
green  close  my_index2      _khJauHgQcKg9q9_HETjdw   5   0
yellow open   simpleauth     2o7Lml9vRGuGH32KhzWDFw   1   1          1            0      8.3kb          8.3kb
yellow open   .bes_schedules BdPRWFNoRLy-_J-r4nM9EQ   1   1          4            0     31.7kb         31.7kb
```

若存在close索引，则会阻塞集群迁移过程，需要手动启动或删除：

- 启动索引：

```
POST /<index_name>/_open
```

- 删除索引：

> **警告** 索引删除后不可恢复，请谨慎操作。

```
DELETE /<index_name>
```

操作步骤

1. 登录百度智能云Elasticsearch控制台，并点击集群名称，进入集群详情界面。
2. 查看集群架构图，点击可用区图标，选择迁移按钮。

3. 选择可用区和子网，点击确认，则开始迁移集群。

> **说明：** 若选择迁移的目标可用区子网IP数量不足，无法迁移集群。请保证子网IP数量充足，再进行迁移。

注意

- 多可用区部署集群暂不支持迁移可用区。
- 跨可用区迁移需要目标可用区的资源充足才能正常创建节点，建议迁移前在百度云控制台提交工单确认目标可用区资源是否充足。
- 迁移过程中要保证集群健康状态为`green`，否则会造成迁移过程阻塞。
- 在可用区迁移过程中，由于涉及网络传输，可能对性能造成影响，建议业务低峰进行迁移操作。

变更HTTPS协议用户指南

概述 HTTPS（Hyper Text Transfer Protocol over Secure Socket Layer）是一种能够保障数据安全的HTTP通道，通过在HTTP协议的基础上增加传输层安全tls（Transport Layer Security）来保障数据传输的安全性。本文介绍如何使用HTTPS协议。 **前提条件**

- 创建百度云Elasticsearch（BES）集群。具体操作请参见创建集群。
- 变更访问百度云Elasticsearch（BES）集群的客户端代码。不变更会导致无法使用客户端程序访问集群。

以官方Elasticsearch的Rest Client访问方式为例，开启HTTPS后，HttpHost中需要修改http参数为https，或者加上https参数，例如new HttpHost("192.168.0.248", 8200, "https"));，同时配置ssl上下文，用于构造client。示例代码如下。

开启HTTPS协议前的示例代码：

```
 import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
public class EsRestClient {
    public static void main(String[] args) throws Exception {
        CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY,
                new UsernamePasswordCredentials("username", "password"));
        RestClient client= RestClient.builder(
                        new HttpHost("hostname", 8200, "http")).setRequestConfigCallback(
                        requestConfigBuilder -> requestConfigBuilder
                                .setSocketTimeout(30000))
                .setHttpClientConfigCallback(httpClientBuilder ->
                        httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider))
                .build();
        Request request = new Request("GET", "/");
        Response response = client.performRequest(request);
    }
}
```

开启HTTPS协议后的示例代码：

```
 import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
import java.security.cert.X509Certificate;
public class EsRestClient {
    public static void main(String[] args) throws Exception {
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(null, new TrustManager[]{new X509TrustManager() {
            @Override
            public void checkClientTrusted(X509Certificate[] x509Certificates, String s) {}
            @Override
            public void checkServerTrusted(X509Certificate[] x509Certificates, String s) {}
            @Override
            public X509Certificate[] getAcceptedIssuers() {return new X509Certificate[0];}
        }}, null);
        CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY,
                new UsernamePasswordCredentials("username", "password"));
        RestClient client= RestClient.builder(
                        new HttpHost("hostname", 8200, "https")).setRequestConfigCallback(
                        requestConfigBuilder -> requestConfigBuilder
                                .setSocketTimeout(30000))
                .setHttpClientConfigCallback(httpClientBuilder -> httpClientBuilder.setSSLContext(sslContext)
                        .setSSLHostnameVerifier(NoopHostnameVerifier.INSTANCE)
                        .setDefaultCredentialsProvider(credentialsProvider))
                .build();
        Request request = new Request("GET", "/");
        Response response = client.performRequest(request);
    }
}
```

操作步骤

- 登录百度云Elasticsearch（BES）控制台，选择地区，点击集群ID/名称，进入目标实例。
- 在集群详情页面的连接信息模块，打开使用HTTPS协议开关。

注：

1. 请确认访问BES集群的代码已经修改，否则将无法用变更后的地址访问BES集群。
2. 变更HTTPS协议前要保证运行的专有主节点数量大于总的专有主节点数量的一半。否则会造成选主失败，集群重启不成功。（如果没有专有主节点，就需要保证运行中的数据节点数量大于总的数据节点数量的一半）
3. 启用和关闭HTTPS服务会中断服务，并且会触发集群全量重启，为保证您的业务不受影响，请确认后操作。

- 点击确定后，连接信息模块访问地址立刻变更，并跳转到操作历史页面。当进度条完成后可使用新地址访问BES集群。
- 协议变更过程中，集群重启、配置变更等操作不可进行。

修改集群名称

Elasticsearch集群成功创建后，依然支持修改集群名称。

操作步骤

1. 登录百度智能云Elasticsearch控制台，进入集群列表。
2. 选中想要修改的集群，进入该集群详情页。
3. 在集群信息里，点击集群名称右侧的“修改”按钮，弹出修改框。

4. 依照编辑框下方命名规则提示，输入新名称，点击”确定“按钮，完成集群名称修改。修改后立刻生效，无需手动刷新页面。

> 重要
>
> - 修改Elasticsearch集群名称，Elasticsearch集群的cluster_name字段始终不会被修改。Elasticsearch集群通过cluster_name参数将所有节点正确地连接到同一个集群，以确保节点能够协同工作并共享数据。
> - 新建集群的cluster_name与集群ID保持一致，存量集群的cluter_name与创建集群时设定的集群名称保持一致。

5. 在**操作历史**里可以查看修改记录，包括可以操作时间，操作行为，操作详情和进度。操作行为为修改集群名称。操作详情分别显示原集群名称和新集群名称。进度包括修改集群名称和操作完成两个环节。

操作历史

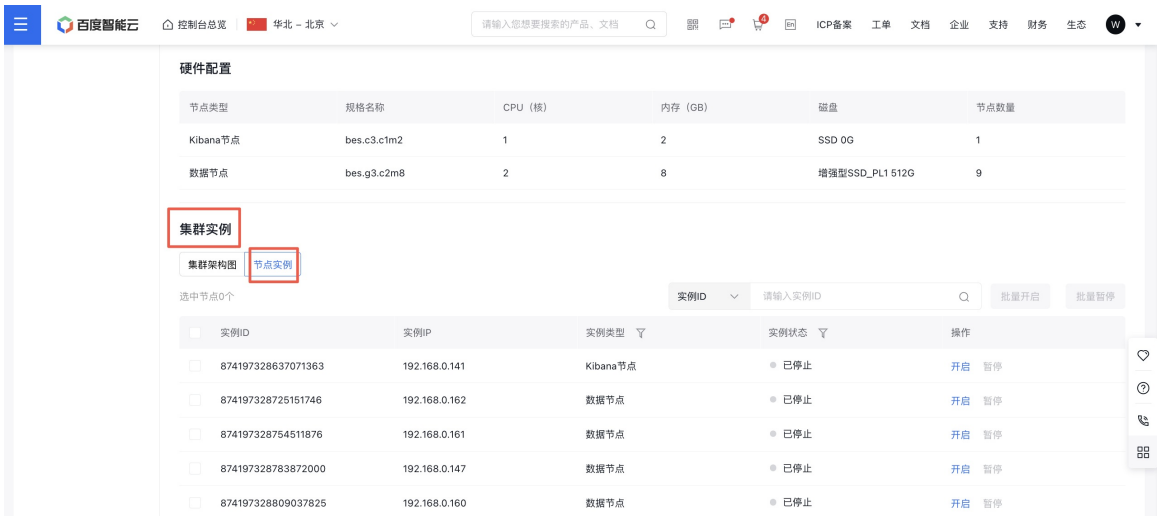
时间	操作行为	操作详情	进度
2023/11/14 11:41:26	修改集群名称	配置： [原集群名称] test_ES [新集群名称] ES_cluster_testcase01	修改集群名称 操作完成 0h0m0s -

节点启停

百度智能云 Elasticsearch 提供节点启停功能，支持用户通过控制台启停 Elasticsearch 集群的节点实例。

操作步骤

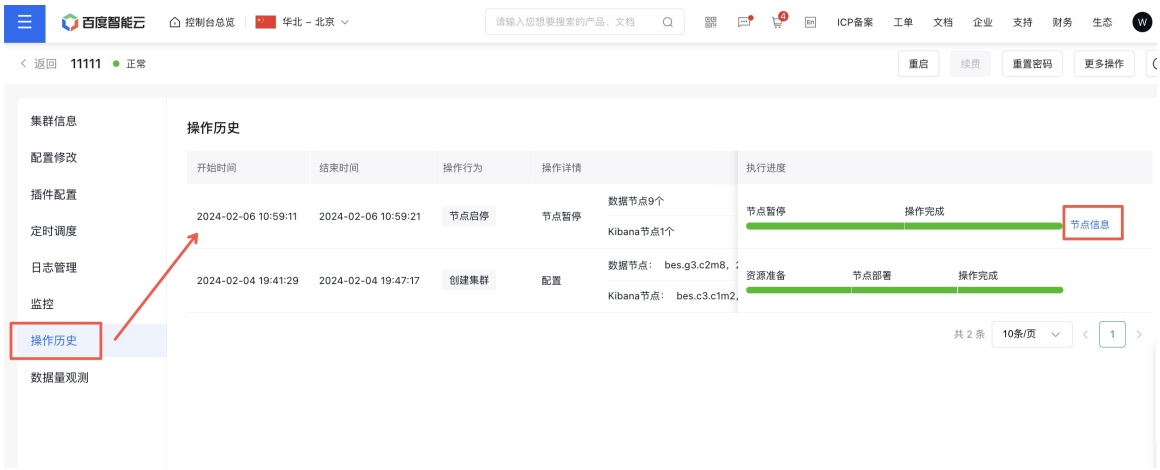
1. 登录百度智能云 Elasticsearch 控制台。
2. 在集群列表中，单击目标集群名称，进入集群详情页。
3. 在集群详情页，集群实例里点击“节点实例”按钮，显示该集群的全部节点。



4. 选中目标节点，执行开启或暂停操作。支持对单个节点进行启停操作，也支持对多个节点进行批量启停的操作

> 注意：
>
> - 进行启停操作时，集群状态会变更为生效中，待启停操作完成后，集群状态恢复为正常；若暂停全部节点，集群状态会变更为失效。
> - 不建议暂停主节点和存在数据的节点，暂停可能会导致集群健康状态变为yellow或red，影响集群正常读写。
> - 节点暂停仍计费，如需删除节点请参考文档进行操作。

5. 在操作历史里可以查看操作记录，包括开始时间、结束时间、操作行为、操作详情、执行进度和节点信息。



点击“节点信息”按钮，可以查看启停节点的详细信息，包括节点ID和节点IP。

集群列表

百度云Elasticsearch的集群列表展示了集群的名称/id、访问ip、运行状态、健康状态、所在网络、所在子网、所在可用区、标签、资源分组、版本、支付方式、创建时间、运行时间等信息，并提供了创建集群、集群配置变更、续费、计费变更、编辑资源分组、搜索集群、导出集群列表等功能入口。

列表说明

功能入口

登录百度云Elasticsearch控制台，进入Elasticsearch的集群列表页面。集群列表页面展示了当前地域您账号下的所有百度云Elasticsearch集群，并提供了以下功能。

创建集群

点击左上角“创建集群”，进入集群创建页面。详细信息，请参见集群创建。

刷新集群

点击右上角搜索栏右侧的刷新图标，可获取集群的实时状态。集群创建和配置变更后，需一段时间生效，状态为“生效中”， 可点击刷新图标查看实例的最新状态，当状态变为“运行中”，即可正常使用集群。

搜索集群

当集群太多时，用户可以通过搜索功能快速定位到目标集群。百度云Elasticsearch支持按照集群ID、集群名称和标签搜索集群。点击搜索栏左侧的下拉菜单选择需要的搜索类别，搜索时支持模糊匹配。资源分组搜索不支持模糊匹配，在下拉菜单中选择分组进行定位。

查看集群基本信息

点击集群名称进入集群详情页面，该页面展示了对应集群的基本信息，支付信息，网络信息和节点信息。详细信息，请参见[查看集群基本信息]。

实例配置变更

在“操作”列点击目标集群的“配置变更”按钮直接进入配置变更界面或点击集群名称进入集群详情页面，通过右上方的“配置变更”按钮进入。详情功能请参见集群扩缩容。

删除集群

点击集群名称进入集群详情页面，点击右上方的“更多操作”按钮选择删除集群即可释放集群。详情信息，请参见集群删除。

续费

在“操作”列点击目标集群的“续费”按钮进入集群续费页面（只有支付方式为预付费的集群可以实现续费功能）。续费时长可

选“按年”和“按月”，并选择需要续费的年长或月长。点击下一步，确认订单信息无误后点击提交订单，选择付款方式付款，完成续费。

计费变更

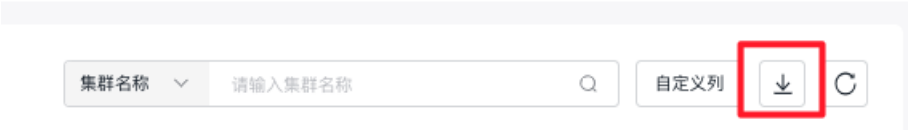
在“操作”列点击目标集群的“计费变更”按钮进入计费变更页面，详情信息，请参见计费说明。

编辑资源分组

在“操作”列点击目标集群的“编辑资源分组”按钮弹出编辑资源分组的对话框。选中不同资源分组树下的子级分组，点击“确认”，添加目标集群至该资源分组中。一个目标集群可以属于多个资源分组树，但在一个资源分组树下只能属于其中的一个分支。如需删除原资源分组，最少保留一个资源分组。详情信息请参见资源管理。

导出集群列表

鼠标放到右上方的下载的icon按钮，会气泡提示“导出列表”，点击此按钮可以将用户该地区的集群列表全部导出为Excel表格，导出表格中的字段和列表中字段一致



查看集群信息

概述 当您需要查看集群的基本信息、支付信息、连接信息等详情信息时，可以通过实例的集群信息页面获取。 **操作步骤**

1. 登陆百度智能云Elasticsearch 控制台。
2. 在集群列表中，单击目标集群名称，进入集群信息页，可以查看集群的基本信息和运行状态。

集群信息说明

类别	名称	说明
基本信息	区域	集群所在地域。
	集群ID	集群唯一标识。
	集群名称	集群名称，支持修改名称。
	管理员名	用于访问Elasticsearch和登录Kibana
	ES版本	包括集群版本和内核版本。
	健康状态	集群的状态。支持绿色（集群的所有主分片和副本分片都可用）、黄色（所有主分片可用，但副本分片有缺失）、红色（有主分片缺失）和灰色（未知状态）。
支付信息	支付方式	支持预付费和后付费两种方式。
	到期时间	支付方式为预付费时集群的到期具体时间。
连接信息	ES访问地址	支持在专有网络下，通过指定的私网地址访问百度智能云Elasticsearch集群。
	Kibana访问地址	支持在专有网络下，通过指定的私网地址访问Kibana。
	HTTPS协议	能够保障数据安全的HTTP通道，通过在HTTP协议的基础上增加传输层安全tls来保障数据传输的安全性。
	Exporter监控接入地址	用于访问Exporter监控服务，查看监控数据。
	是否开启Grafana 监控	支持Grafana监控功能开启与关闭。
	Grafana 监控URL	用于访问Grafana服务，查看监控数据。
网络配置与可用区	可用区部署数量	集群所部属的可用区数量，支持单可用区、双可用区和三可用区。
	所在网络(VPC)	集群所在的虚拟网络地址。
	可用区及子网	集群所在地域、可用区以及子网IP。
	公网访问IP	支持在公网环境下，通过指定的公网地址访问Elasticsearch实例。需要自行绑定eip。
	Kibana公网访问IP	支持在公网环境下，通过指定的公网地址访问Kibana。需要自行绑定eip。
硬件配置		列表展示集群的节点类型、规格名称、CUP、内存、磁盘以及节点数量等信息。
集群实例	集群架构图	可视化展示集群的地域、可用区以及每一个可用区下的节点信息。
	节点实例	列表详细展示每一个好节点实例信息，包括实例ID、实例IP、实例类型、实例状态和操作。

配置变更

集群扩缩容

应用场景 在业务发展过程中，数据量和访问量都在变化，当现有集群规模无法满足实际业务需求时，可以对集群进行扩容以满足新的数据量和访问量使用场景。

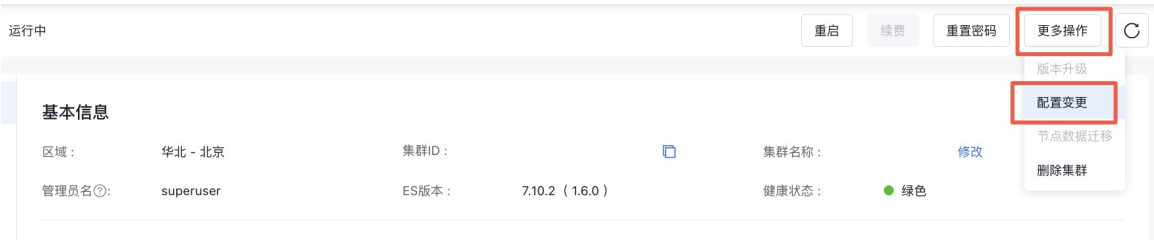
目前BES支持支持对节点规格、节点数量、节点磁盘和节点类型进行配置调整。整体集群规格评估请参考集群容量规格评估。

操作步骤

1. 登录百度智能云Elasticsearch 控制台，进入百度云Elasticsearch的集群列表。
2. 集群配置变更的操作有两个入口：

- 集群列表页面，在“操作列”选择目标集群的集群点击“配置变更”直接进入；

- 进入集群详情页面，在上方操作栏点击“更多操作”，在下方操作中选择“配置变更”。



3. 进入配置变更页面，选择变更操作和节点类型进行配置。

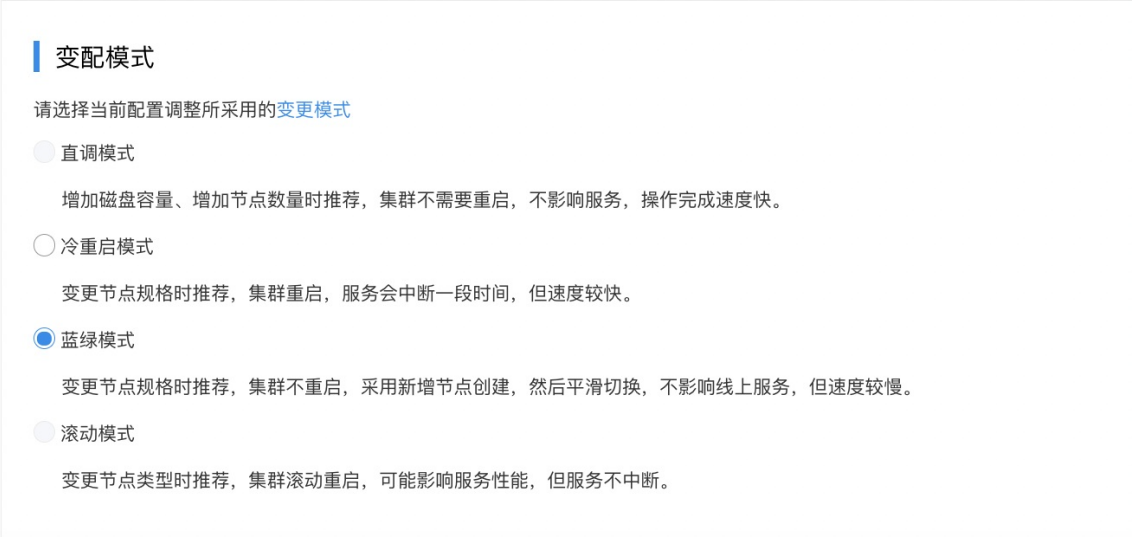
> **说明：**
>
> 1. 目前一次变更操作仅支持对节点数量、节点规格、节点磁盘和节点类型的其中一种操作进行变更调整，且每次仅支持升级一种节点类型，暂不支持多种变更行为或多种节点同时调整。如果有多种变更需求，请执行多次操作。
> 2. 集群变配目前仅支持存储计算资源分离的新套餐节点进行操作，旧集群暂时不支持修改节点规格，仅能增加节点数量。如果想进行其他变更操作，请先将旧套餐升级为新套餐，套餐升级操作入口也在配置变更页。
> 3. 当集群状态为 red 或 yellow 时，不支持进行变更节点规格、变更节点磁盘和删除节点等操作。请将集群状态恢复 green，再进行以上操作。

变更节点规格

需要变更节点的CPU和内存配置时，变更行为选择“变更节点规格”，选择需要变更的节点类型，然后在下方选择需要变更的新的规格配置。

变更节点规格，变配模式可选择冷重启模式或蓝绿模式，冷重启模式变更速度快，但会重启集群，服务会中断一段时间；蓝绿

模式无需重启集群，可实现平滑切换，不影响线上服务，变配速度取决于集群的数据量大小，数据量越大速度会越慢。用户可根据业务场景进行选择。



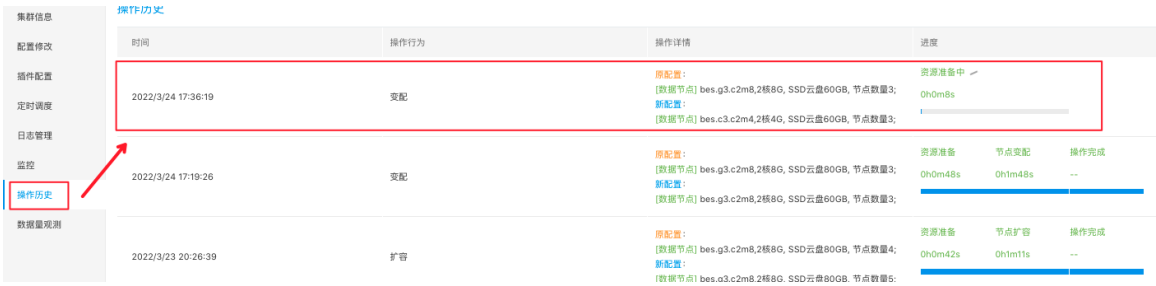
> **说明**：当变更模式为蓝绿模型时，若所在集群的子网IP数量不足，无法变更节点规格。请保证子网IP数量充足，再进行节点规格变更。

变更到新的配置后，下方会展示新的配置价格，确认之后点击“下一步”。

“下一步”会展示本次变更的详细信息。确认无误之后，点击“提交订单”，提交配置变更操作。

配置变更操作提交之后，集群会进入“正在变配”状态。待集群状态变为“运行中”，即可正常使用。

可以在集群操作页，查看集群的变更进度。



> **说明：**
>
> 当集群状态为 red 或 yellow 时，不支持变更节点规格。请将集群状态恢复为green，再进行节点规格变更操作。

新增节点类型

新增节点类型支持为已创建集群添加新的节点类型。BES目前支持已创建集群添加专用主节点。

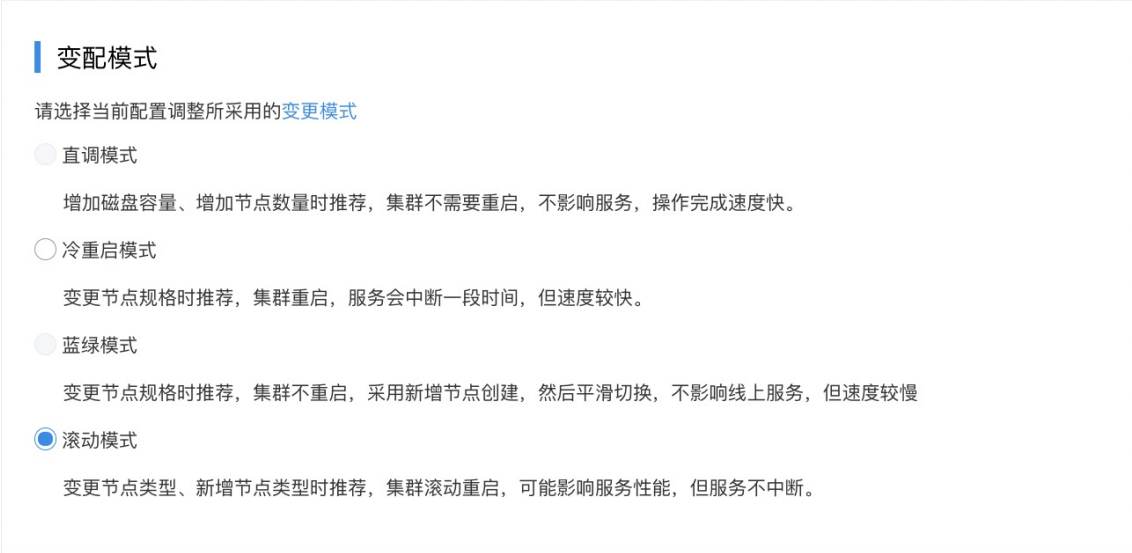
> **说明：**
>
> 当集群状态为 red 或 yellow 时，不支持新增节点类型。请将集群状态恢复为green，再进行新增节点类型操作。

专有主节点的主要功能是对集群进行操作，例如创建或删除索引，跟踪哪些节点是集群的一部分，并决定哪些分片分配给相关的节点。稳定的主节点对集群的健康非常重要。如果创建BES集群时，没有选择专用主节点，任一数据节点会被选为主节点。

随着业务的发展，索引数据和搜索查询等操作会占用大量的CPU、内存和IO资源。为了确保集群的稳定性，建议用户添加专有主节点，分离主节点和数据节点，缓解集群读写压力，提升集群稳定性。

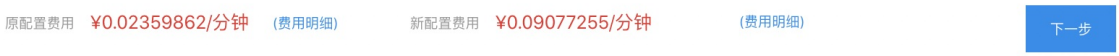
变更行为选择“新增节点类型”，节点类型选择“专用主节点”。根据业务场景，在节点配置中选中专用主节点的规格。BES免费为用户提供40G的系统盘。数据盘默认为20G的SSD云盘，不支持变更。节点数量可选3个或5个。

新增专用主节点，变配模式可选择冷重启模式或滚动模式，冷重启模式变更速度快，但会重启集群，服务会中断一段时间；滚动模式对集群中节点逐个滚动重启完成变配，期间系统服务不间断，可能影响线上访问性能。用户可根据业务场景进行选择。



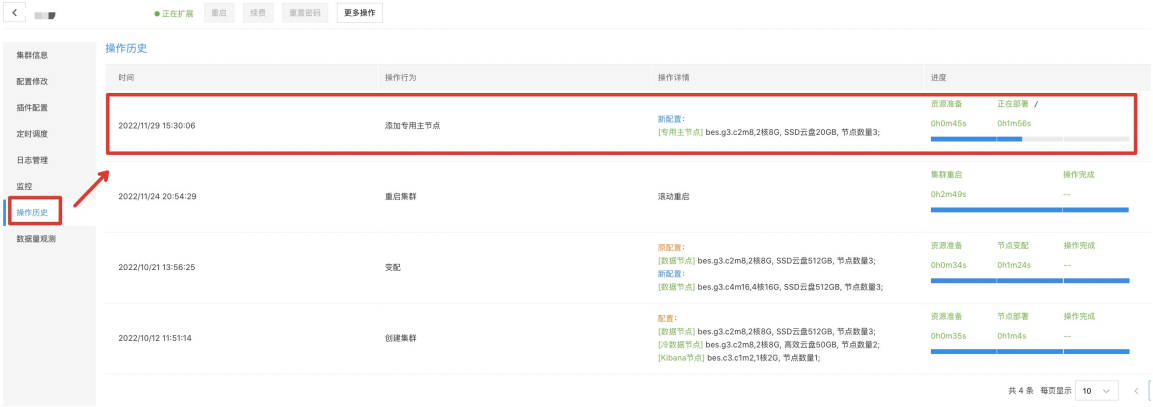
> **说明**：若所在集群的子网IP数量不足，无法新增节点类型。请保证子网IP数量充足，再进行新增节点类型。

底端展示新的配置价格，确认之后点击“下一步”。确认本次变更的详细信息，无误之后，点击“提交订单”。



配置变更操作提交之后，集群会进入“正在扩展”状态。待集群状态变为“运行中”，即可正常使用。

可以在集群操作页，添加专用主节点的变更进度。



变更节点磁盘 变更节点磁盘目前仅支持对数据节点的磁盘类型和磁盘大小进行变更。变更磁盘类型的同时，支持调整磁盘大小。 **变更磁盘类型**

为更好满足用户弹性使用资源的需求，BES支持用户按照实际业务需求来变更磁盘类型。业务高峰期可以升级磁盘类型，业务低谷期可以降低磁盘类型。

> **说明：** 目前降低磁盘类型只对白名单用户开放。如有需要，请提工单开通。

在变更行为中选择“变更节点磁盘”，在数据盘下拉框中选择想要变更的磁盘类型，填写磁盘容量。轻触或点击“磁盘性能对比”，展示磁盘性能对比信息。

变配模式支持直调模式和蓝绿模式。升级节点磁盘类型时，默认直调模式；降低节点磁盘类型时，默认选择蓝绿模式。 详细信息可参考变配模式。

> **说明：** 当变更模式为蓝绿模式时，若所在集群的子网IP数量不足，无法变更磁盘。请保证子网IP数量充足，再进行磁盘变更。

> **重要：** 若您降低磁盘类型时，选择直调模式，系统为了数据安全设置了磁盘类型变更的窗口保护期，持续12小时。该窗口保护期开始于发起变配操作的时刻，结束于 12 小时后。需等待保护期结束才能再次发起新的变配操作。若12小时内变更失败，由于偶然外部因素，可立即发起新的变更操作，不受保护期限制。若磁盘异常导致变更时间超过12小时，请提工单处理。

底端自动展示新的配置价格，确认配置信息之后，点击“下一步”。确认本次变更的详细信息，无误之后，点击“提交订单”。

配置变更操作提交之后，集群会进入“正在变配”状态。待集群状态变为“运行中”，即可正常使用。

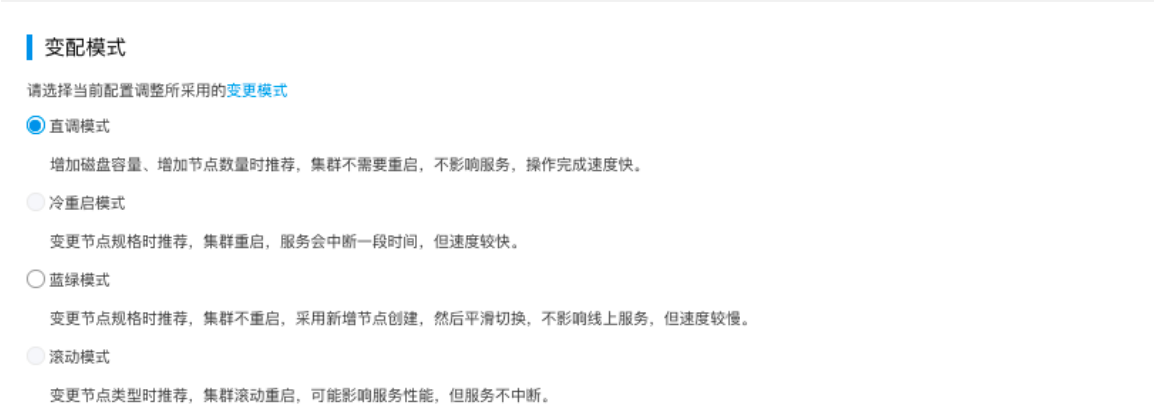
可以在操作历史页，查看变配进度。

> **说明：**
>
> 1. 增强型SSD_PL2仅支持CPU大于等于16的部分套餐。用户如有需要，请提工单开通。
> 2. 老套餐不支持变更磁盘类型。
> 3. 集群处于初始化、正在变配、正在删除等状态，集群健康状态为red或yellow时，不可变更磁盘类型。

变更磁盘大小

当节点的计算资源足够，而存储空间不足或多余时，可以变更磁盘的容量。在变更行为中选择“变更节点磁盘”，然后输入相应的磁盘容量数值进行调整。

变更节点磁盘时，扩容磁盘可选择直调模式和蓝绿模式，直调模式无需重启集群，完成速度快；缩容磁盘可以选择蓝绿模式，蓝绿模式无需重启集群，不影响线上服务，速度取决于数据量大小，数据量越大速度会越慢。用户可根据业务场景进行选择。



> **说明**： 当变更模式为蓝绿模式时，若所在集群的子网IP数量不足，无法变更磁盘。请保证子网IP数量充足，再进行磁盘变更。

底端自动展示新的配置价格，确认配置信息之后，点击“下一步”。确认本次变更的详细信息，无误之后，点击“提交订单”。

配置变更操作提交之后，集群会进入“正在变配”状态。待集群状态变为“运行中”，即可正常使用。

可以在操作历史页，查看集群的变更进度。

> **说明**：
>
> 1. 如有磁盘缩容或降配需求，请提交工单咨询，此操作需要产品服务方进行评估。
> 2. 磁盘扩缩容的范围为50GB-6TB。
> 3. 当缩容后的磁盘规格 < 当下数据量/75%时，缩容后磁盘容量无法容纳当前数据量，不支持存储缩容。
> 4. 当集群状态为 red 或 yellow 时，不支持磁盘缩容操作，请将集群状态恢复为green，再进行操作。 当集群状态为 red 或 yellow 时，磁盘扩容支持通过直调模式实现。
> 5. 由于磁盘扩容技术升级，对于2021年3月19日0点之后创建的集群，支持磁盘在线热扩容。之前的集群暂只能支持磁盘滚动扩容。
> 6. 增强型SSD_PL2支持的最小磁盘容量为461GB，不可再减少。

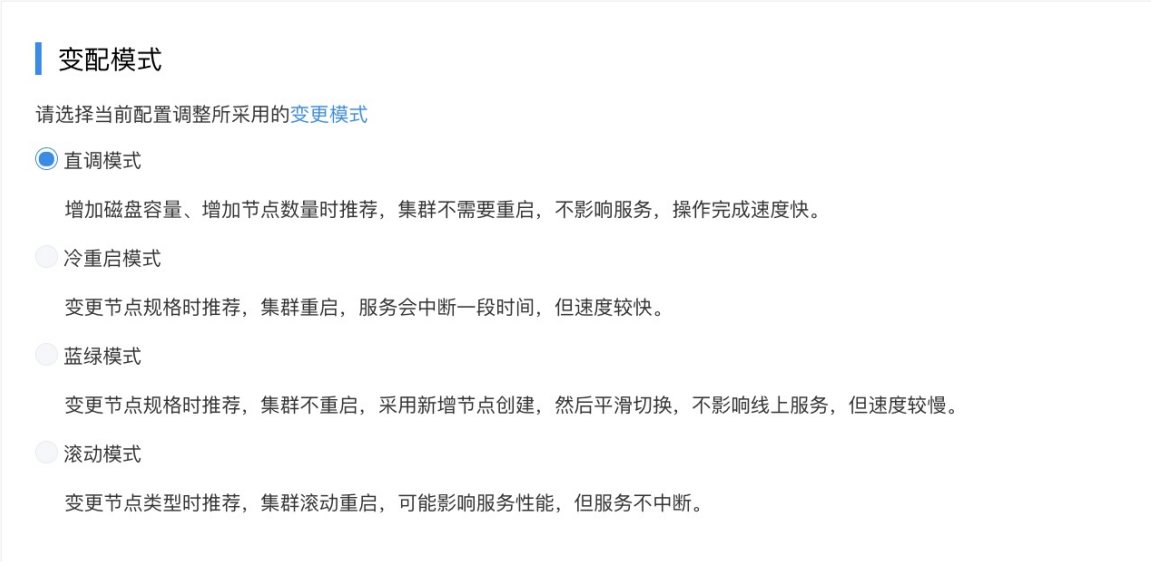
变更节点数量

增加节点数量

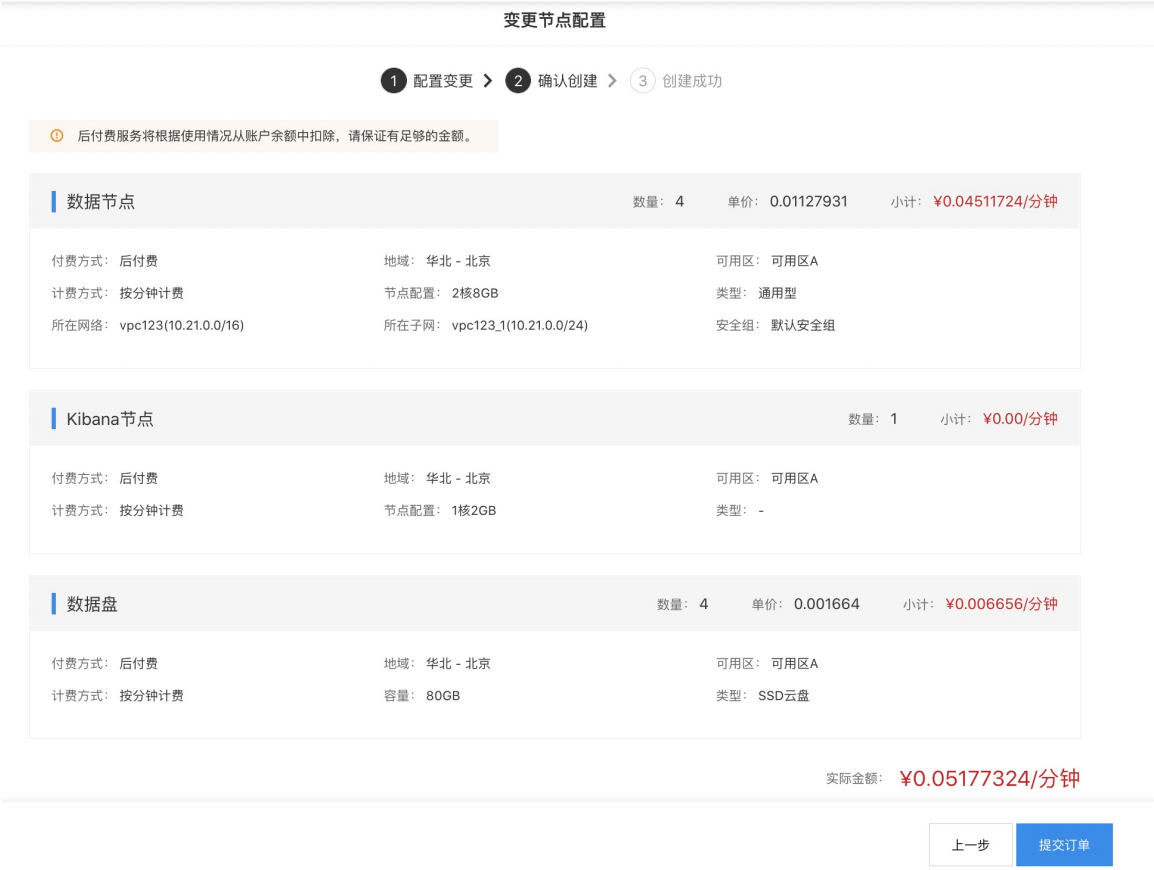
在配置变更页，选择“变更节点数量”，然后在“节点操作”一栏选择扩容节点，下方数字调节框确定要增加的节点数量。

> **注意：** 一个集群中数据节点最多有150个，扩容后的节点数也不可超过150个。有150个数据节点的集群，不支持扩容。

增加节点数量时，变配模式建议选择直调模式，变更速度快，并且无需重启集群，不影响线上服务。



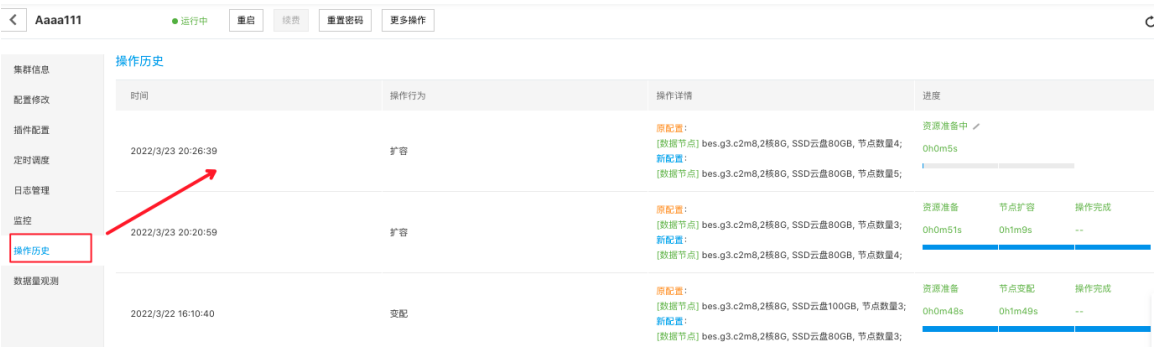
底端展示新的配置价格，确认之后点击“下一步”。确认本次变更的详细信息，无误之后，点击“提交订单”。



> **说明：** 若所在集群的子网IP数量不足，无法增加节点数量。请保证子网IP数量充足，再增加节点数量。

配置变更操作提交之后，集群会进入“正在扩展”状态。待集群状态变为“运行中”，即可正常使用。

可以在操作历史页，查看集群的变更进度。



删除节点

在配置变更页，选择“变更节点数量”，然后在“节点操作”一栏选择缩容节点，节点类型选择需要删除的节点类型，下方会显示该集群当前选择的节点类型下的所有节点。

注：仅支持删除数据节点、协调节点、冷数据节点。

勾选表格中需要删除的节点，可用区右边的图标可以辅助筛选可用区。若选中的节点中还存有数据，则会提示“监测到选中节点上有遗存数据，建议先进行节点数据迁移再继续节点缩容”，提供数据迁移服务入口。

注：

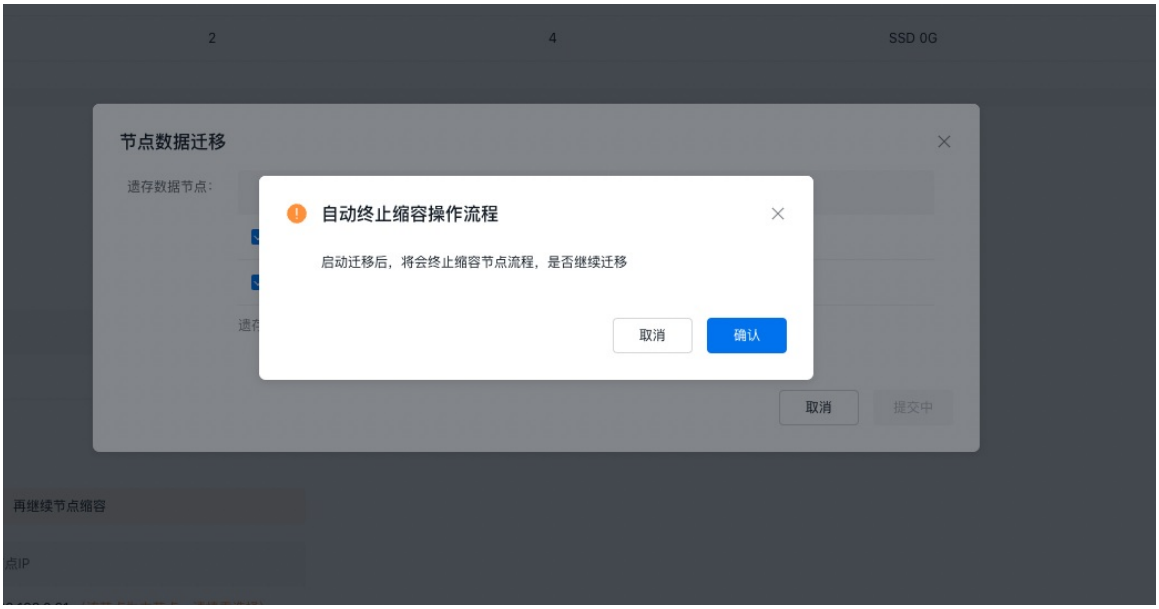
1. 单可用区集群：缩容节点个数的上限为【当前集群此类节点总数-1】。
2. 多可用区集群：缩容节点个数的上限是，每个可用区剩余的节点大于等于1且各个可用区剩余的同类型节点数要相同。

用户可以选择进行节点数据迁移再删除节点也可不做迁移直接删除。

- 若用户点击“节点数据迁移”，则弹出弹窗，显示已选中节点中仍遗存数据的节点，点击确定，会提示用户集群将先进行节点数据迁移，等迁移结束后用户需重新从配置变更页面进入进行缩容操作。再次点击确定，页面跳转到操作历史，节点开始进行数据迁移。

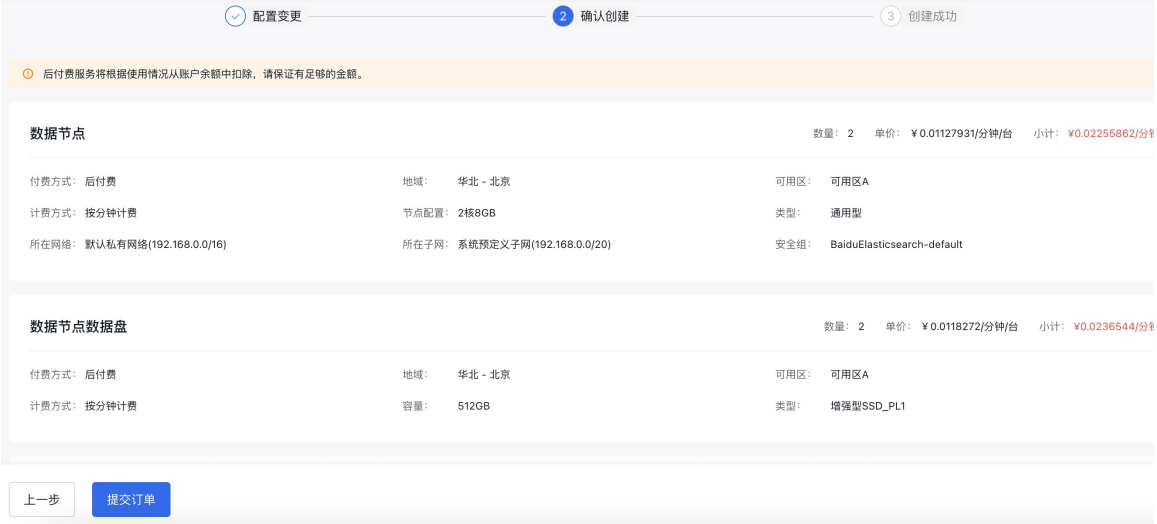
注：

1. 如果用户想终止迁移，在迁移操作完成前，可点击“迁移回滚”终止迁移，将已经迁移的数据重新分配到节点中。



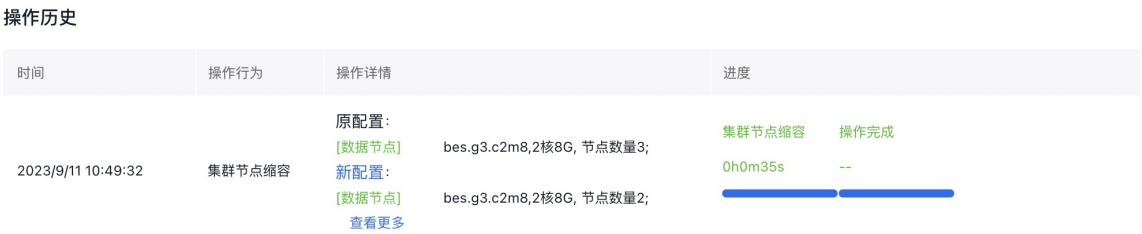
2. 数据迁移的时候，集群状态显示正在迁移，当集群状态变为运行中时，用户可重新进行删除节点操作。

- 如果用户只是想删除节点并不想迁移数据，也可忽略提示，直接点击左下角的下一步。确认本次变更的详细信息，无误之后，点击“提交订单”。



配置变更操作提交之后，集群会进入“正在扩展”状态。待集群状态变为“运行中”，即可正常使用。

可以在集群操作页，查看集群的变更进度。



变配模式说明

不同的变更操作会通过不同的过程来执行，因此变更操作对应多种变更模式，用户可以根据业务的情况进行选择。

- **直调模式**：是指集群直接进行配置调整，集群不需要重启，不影响服务，磁盘容量热扩、增加节点数量时会采用直调模式，操作完成速度快。
- **冷重启模式**：是指集群会进行重新启动，服务会中断一段时间，但重启时间较短，整个操作的执行过程较快。节点规格变更、新增节点类型等操作，如果需要迅速生效，服务允许在中断的情况下选择冷重启模式。
- **蓝绿模式**：是指集群通过蓝绿节点进行服务切换，将数量相等且更高规格的新节点加入集群，然后将老节点上的数据全部迁移至新节点，最后删除全部的旧节点，实现集群配置的升级。蓝绿模式变更集群配置时不需要重启集群，可以实现平滑扩容，不会影响在线业务，但是整个执行过程较慢。
- **滚动模式**：是指逐个对集群中的节点通过重启的方式完成配置的变更操作，变配时间与节点个数成正比，整个操作过程不会中断服务，但是会影响服务性能。新增节点类型和不支持直调模式的磁盘扩容可以支持滚动模式扩容，其他变更操作暂不支持滚动模式。

调整配置费用说明

不同计费模式下的集群，调整配置时，费用结算方式会有所不同。

- 预付费包年包月的集群，在调整配置时，会根据集群剩余有效期以及新的配置的定价，计算需要的费用，具体可以参考 调整配置费用说明。
- 后付费按量计费的集群，计费周期为秒，当配置调整完成后，下一秒费用按新的配置定价进行结算。
- 缩容操作，现金券代金券均不退回；按活动价购买集群，缩容不退款。

集群变配说明与建议

百度智能云Elasticsearch(BES)服务为用户提供灵活扩缩容的变配能力，用户可以根据业务的增长变化随时调整集群的配置规模。BES提供了多种变配模式，以下将详细介绍各种变配模式的原理和操作建议。

变配模式介绍

直调模式

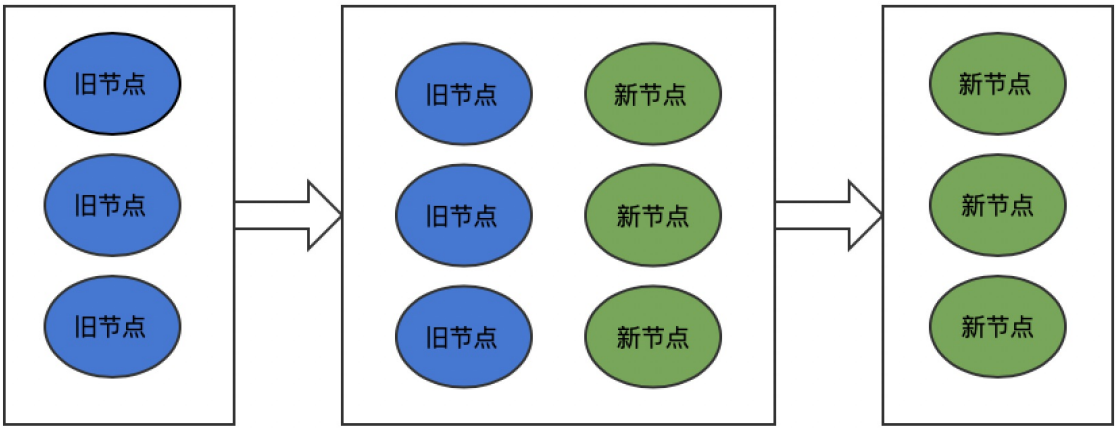
直调模式适合对线上业务无影响，且速度较快的变配操作，例如集群添加节点、增加磁盘容量等。

冷重启模式

直接将集群中所有节点重启并完成变配，期间系统服务中断，会影响线上访问，但总体变配持续时间较短，适合需要短时间变更配置的操作。

蓝绿模式

不会重启集群，在原集群中添加相同数量的配置不同的新节点。添加节点后进行数据迁移，待数据迁移完成后，无缝切换并去除掉原节点。



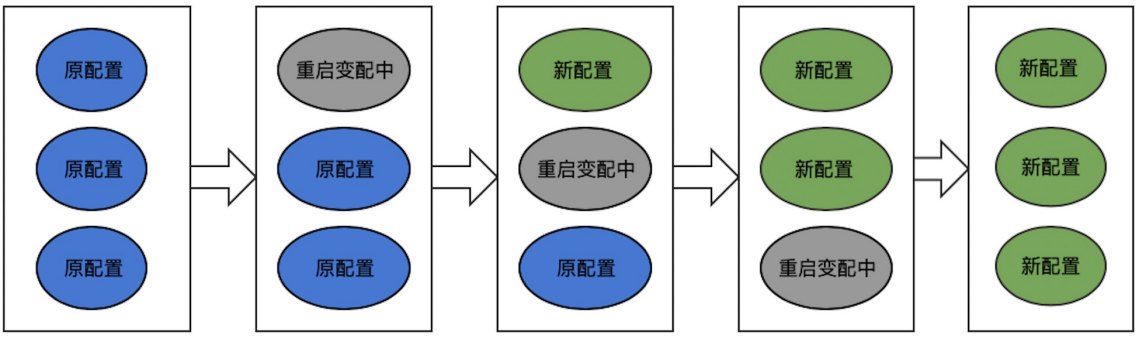
注意：

- 蓝绿变换整个过程业务端无感知，且扩容平滑，适合对集群可用性要求较高的场景。
- 蓝绿变换完成后节点 IP 会发生变化。
- 由于涉及数据迁移，蓝绿变换耗时和集群数据量有关，总体变配持续时间从数分钟到数小时或更长。如果急需提升数据迁移的速度，可以通过如下命令修改 ES 的配置：

```
 PUT _cluster/settings
{
  "persistent": {
    "cluster.routing.allocation.node_concurrent_recoveries": "8",
    "indices.recovery.max_bytes_per_sec": "80mb"
  }
}
```

滚动模式

对集群中节点逐个滚动重启完成变配，期间服务不会中断，但可能对线上性能造成影响。总体变配持续时间与节点个数、集群数据量有关。



注意：

- 由于不需要迁移数据，总体变配过程相比于蓝绿模式能够快速完成。适合于集群遇到性能瓶颈，期望快速完成扩容变配的场景。
- 每重启一个节点后，会等待ES集群恢复为`green`状态后，才会重启下一个节点。
- 由于需要滚动重启集群的每一个节点，所以当集群存在无副本索引的情况下会造成集群健康状态变成`red`，且会有丢失数据的风险。因此**不建议当集群存在无副本索引的情况下选择滚动模式**。
- 滚动模式过程中会对集群的性能有一定的影响，因此**不建议在业务高峰期间选择滚动模式**。

变配行为与变配模式选择

变配行为	适用场景	变配模式	简介	特点
变更节点数量	节点规格较高，但集群整体计算资源不足	直调模式	直接向集群中加入更多相同规格的节点。	扩容平滑 无需停服 流程时间短，不受节点数量影响，预计5 - 10分钟完成
变更节点磁盘	计算资源充足，磁盘存储空间不足	直调模式 滚动模式/冷重启模式	直接增加集群中节点上挂载的磁盘扩容容量。 对于2021年3月19日0点之前的集群暂只能支持滚动变更，其中总节点个数小于2的集群仅支持冷重启变更。	直调模式变更平滑扩容不停服，流程时间短，预计2- 5分钟完成。 滚动模式会对集群性能造成影响，但服务不会中断，变配时间与节点数量、集群数据量有关。冷重启会导致服务短时间暂时中断。
变更节点规格	节点规格较低，集群整体计算资源不足	冷重启模式 蓝绿模式	可选择冷重启模式或蓝绿模式。	冷重启模式会造成服务短时间暂时中断，变更规格后会恢复。 蓝绿模式的总持续时间跟集群数据量有关。

节点数据迁移

概述 当用户需要缩容节点、调整分片分布、更换虚机时，可通过百度智能云Elasticsearch的节点数据迁移功能，调整节点上数据的数量。

操作入口 登录百度智能云Elasticsearch控制台，单击集群列表中的目标集群ID进入在基本信息页面。点击更多操作>节点数据迁移，跳出节点数据迁移弹窗。

操作流程 选择需要迁移节点的节点类型。 注：只支持迁移数据节点和冷数据节点。

节点数据迁移支持智能推荐节点和自定义节两种选择。 **智能推荐节点** 确定迁移节点的个数后，系统将根据数据分布和节点状态智能推荐迁移节点，系统无法评估数据重要性，需要用户根据实际业务情况进行确认。

自定义节点 用户可自主选择要迁移的节点，平台会提醒用户主节点慎重选择。

用户确定好需要迁移的节点后，勾选下方知悉数据迁移服务协议，点击确定，跳转到操作历史页面，集群开始进行节点数据迁移。

权限管理

权限管理包括两个层面的控制：

- URI级别的权限控制：控制特定用户可以访问哪些URI

  场景说明：管理员可以让某个用户有管理集群动态配置参数的权限，需要将 /_cluster/setting 写(POST)权限授权给这个用户

- index级别的权限控制：控制特定用户对index的读写操作权限

  场景说明：禁止某些用户访问特定index或读写权限分配给不同的用户进行操作，使得不同的用户获得对index不同的操作权限

创建用户

```
 curl -u username:password -XPOST -H "Content-Type: application/json" 'host:port/_user/create' -d '
{
 "username" : "test",
 "password" : "test123",
 "ip_whitelist": ["x.x.x.x"],
 "get_path" : ["/test*", "/*/_bulk", "/*/_search"],
 "put_path" : ["/test*"],
 "post_path" : ["/test*"],
 "del_path" : ["/test*"],
 "head_path" : ["/test*"],
 "read_index": ["test*", "online-"],
 "write_index": ["test*"]
}'
```

参数	意义
username	用户名，支持字母(a-z及A-Z)、数字(0-9)，长度小于20个字符
password	用户密码，支持字母(a-z及A-Z)、数字(0-9)，长度6-20个字符
ip_whitelist	访问白名单
{HTTP_METHOD}_path	对不同HTTP类型的请求的权限控制，{HTTP_METHOD}为put、post、get、delete、head
read_index	拥有读权限的index列表
write_index	拥有写权限的index列表

注意：

> 1. 只有superuser可以执行create user的指令，其他用户在任何情况下都不能执行
> 2. {HTTP_METHOD}_path、read_index、write_index中的内容支持通配符匹配和前缀匹配

通配符支持 * 和 ？ ，*代表任意个字符，?代表一个字符

例子	说明
*st	匹配test、tst，不能匹配 tweet
*st	匹配test、tst，不能匹配 tweet
a?d	匹配abd，acd, 不能匹配 abcd
a??d	匹配abcd，acad, 不能匹配 abd
/*/test	匹配 /abc/test,/abc/ddd/test 不能匹配 /abc/cde
*	匹配所有字符串
/*	匹配所有以/开头的URI
/*/_settings	匹配所有以_settings结尾的URI

如果只想赋予user某个index的权限，只需要设置read_index、write_index以及将get_path设置为"/"即可，示例如下：

```
 curl -u username:password -XPOST -H "Content-Type: application/json" 'host:port/_user/create' -d '
{
 "username" : "test",
 "password" : "test123",
 "ip_whitelist": ["x.x.x.x"],
 "get_path" : ["/"]
 "read_index": ["test*"],
 "write_index": ["test*"]
}'
```

修改用户

```
 curl -u username:password -XPOST -H "Content-Type: application/json" 'host:port/_user/alter' -d '
{
 "username" : "test",
 "password" : "test321",
 "get_path" : ["/test*", "/_cat/health"],
 "read_index": ["test*"]
}'
```

> 1. 修改用户的访问权限，同时也可以修改用户的密码
> 2. 覆盖方式：出现的字段将会覆盖原有内容，未出现的字段会保留
> 3. 只有root和superuser可以执行这个API

删除用户

```
 curl -u username:password -XPOST -H "Content-Type: application/json" 'host:port/_user/delete' -d '
{
 "username" : "test"
}'
```

查看用户

```
 curl -u username:password -XPOST 'host:port/_user/show’
{
  "size" : 20
}
```

> 1. 该API需要在创建普通用户时加入其权限路径，这样普通用户才可查看自己的用户信息
> 2. 如果执行该API的是root或者superuser，显示所有的用户
> 3. size表示返回的用户数，默认返回10

重置密码

```
 curl -u username:password -XPOST 'host:port/_user/resetpasswd’
{
 "password" : "test123"
}
```

> 1. 该API需要在创建普通用户时加入其权限路径，这样普通用户才可修改自己的密码
> 2. 管理员修改普通用户密码时请使用更新用户的API

使用示例

创建可使用kibana的只读用户

```
 curl -u username:password -XPOST -H "Content-Type: application/json" 'host:port/_user/create' -d '
{
 "username" : "xxxxxx",
 "password" : "xxxxxx",
 "ip_whitelist": ["x.x.x.x"]
 "get_path" : ["/"],
 "put_path" : ["/"],
 "post_path" : ["/"],
 "del_path" : ["/"],
 "head_path" : ["/"],
 "read_index": ["test*",".kibana"],
 "write_index": [".kibana"]
}'
```

kibana会向ES发送一些状态检查、存储图表信息等请求，所以请求权限需要提供所有权限；如果希望创建的用户只拥有只读权限，可以在`write_index`中只添加`.kibana`，在`read_index`中增加用户希望读取的索引。

基于BOS的快照与恢复

介绍如何用百度智能云Elasticsearch创建快照与恢复，如何将您的数据快照至百度对象存储BOS(Baidu Object Storage)中。快照和备份的概念基本上是一致的，快照更强调point-in-time即强调某一时刻。

> 注意：目前**基于BOS的快照功能**支持所有版本的Elasticsearch实例。

创建快照

主要包括两个步骤

- 创建基于BOS的仓库
- 创建数据的快照（手动触发或者定时调度）

创建基于BOS的仓库

- 创建仓库前的准备，需要在您的BOS中创建相应的bucket，并保证创建快照的用户具有相应的权限，这里的用户是用百度智能云的access_key、secrect_key来标识的，bucket的存储类型目前**仅支持标准存储和低频存储**（暂不支持多AZ的存储类型），您可以根据需求选择，推荐选用标准存储
- **BOS对应的bucket您应该保证和您的Elasticsearch集群在同一个region中**
- es_repo为您设置的仓库名字，您可以根据您自己的业务需求起不同的名字

```
PUT /_snapshot/es_repo
{
    "type": "bos",
    "settings": {
        "access_key": "your access_key",
        "secret_key": "your secret_key",
        "endpoint": "s3.bj.bcebos.com",
        "bucket": "es-repo",
        "base_path": ""
    }
}
```

相关参数意义：

参数	作用
type	仓库的类型，在这里您应该填`bos`
access_key	百度智能云的access_key，可以在百度智能云的console中看到
secret_key	百度智能云的secret_key，可以在百度智能云的console中看到
endpoint	BOS对应各个region的服务域名
bucket	BOS的bucket，务必保证ak/sk对应的用户身份有读写bucket权限
base_path	仓库的起始位置，默认为根目录;开始使用后不要修改，会导致快照查询异常
chunk_size	大文件将会被拆分成多个part，默认1G，最小5M，最大5T
max_snapshot_bytes_per_sec	每个节点快照的最大速度，默认`40mb/s`
max_restore_bytes_per_sec	每个节点恢复的最大速度，默认`40mb/s`

BOS对应各个region的服务域名

区域	访问Endpoint
BJ	s3.bj.bcebos.com
BD	s3.bd.bcebos.com
GZ	s3.gz.bcebos.com
SU	s3.su.bcebos.com

创建完仓库后，您如果需要修改对应的参数，使用POST方法,假如我们上传的数据非常大，我们可以限制snapshot过程中分块的大小，超过这个大小，数据将会被分块上传到BOS中。

```
POST /_snapshot/es_repo
{
    "type": "bos",
    "settings": {
        "access_key": "your access_key",
        "secret_key": "your secret_key",
        "endpoint": "s3.bj.bcebos.com",
        "bucket": "es-repo",
        "chunk_size": "1g",
        "base_path": ""
    }
}
```

列出所有仓库信息

```
GET /_snapshot
```

查看具体仓库的信息

```
GET /_snapshot/{您设置的仓库名}
```

快照(snpashot)

一个仓库可以包含多个快照，每个快照都是一系列索引的集合，可以是单个索引，一部分索引，所有索引。当创建快照的时候，您可以指定您需要快照的索引，再不指定的情况下将快照集群所有打开着的索引，您需要给快照取一个唯一的名字，这个名字最好有一定的意义如snapshot_2018_07_01表示2018年7月1日的一个快照，这样您恢复的时候可以根据您对数据的要求进行恢复。

发起一次快照：

```
PUT /_snapshot/es_repo/snapshot_2018_07_01?wait_for_completion=false
```

这个请求会将集群中所有打开的索引快照到`es_repo`仓库下，并将这次快照命名为`snapshot_2018_07_01`，这个请求将会在snapshot初始化完成后会立即返回，快照过程将在您的集群后台运行。

`wait_for_completion`参数用来告诉请求是否在snapshot初始化完成后或snapshot完成后返回，默认为`false`即snpashot初始化完成后即返回；设置为`true`则是在完成备份快照后才会返回结果，这可能需要很长时间，一般不建议设置为`true`。

在snapshot初始化的时候，关于之前所有的snapshots的信息都会被加载到内存中，这就意味着即使wait_for_completion设置为false，面对非常大的仓库可能需要花费数秒更甚至数分钟。

默认情况下，集群中所有`打开的(open)`和`开始的(started)`的索引都会被创建snapshot，可以通过在快照请求中指定那些索引被快照：

```
PUT /_snapshot/es_repo/snapshot_2018_07_01
{
  "indices": "index1,index2",
  "ignore_unavailable": true,
  "include_global_state": false
}
```

参数	作用
indices	需要被包含在snapshot中的index列表，支持`multi index syntax`
ignore_unavailable	设置为true时，忽略indices中不存在的index.默认情况下没有设置，如果index，不存在会报错
include_global_state	设置为false时，阻止`cluster global state`被snapshot

`cluster global state`是指Elasticsearch维护的集群全局元数据信息，具体解释可参考：https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster-state.html。

snapshot有以下几个特性：

- snapshot是增量的,一个snapshot代表的是index的`point-in-time` 视图(从snapshot创建开始，开始后添加的records不在这个snapshot中不可见)
- 除了创建这个index所有主分片(primary)的snapshot，snapshot还可以快照 `global cluster metadata`（包括持久化的cluster setting和templates等）
- 对于集群而言，任意时刻只能有一个snapshot在进行，所有的快照请求都必须在上一个快照完成后才能被执行，否则会被直接拒绝

如果您希望周期性地进行快照，可以通过BES的定时调度-备份数据功能定时执行快照备份数据，保证数据的安全性。

查询快照

一旦一个snapshot被创建，关于这次snapshot的信息可以通过直接对仓库和快照名字发起一次`GET`请求，获得特定的快照信息。

基本格式: GET /_snapshot/{your_repo_name}/{your_snapshot_name}，具体如下：

```
GET /_snapshot/es_repo/snapshot_2018_07_01
```

返回的响应中包含了快照相关的所有信息：

```
{
 "snapshots": [
   {
     "snapshot": "snapshot_2018_07_19",
     "uuid": "TWKo55e7TSy1Sq4WLxMVrQ",
     "version_id": 5050099,
     "version": "5.5.0",
     "indices": [
       "snapindex"
     ],
     "state": "SUCCESS",
     "start_time": "2018-07-19T10:53:17.543Z",
     "start_time_in_millis": 1531997597543,
     "end_time": "2018-07-19T10:53:21.795Z",
     "end_time_in_millis": 1531997601795,
     "duration_in_millis": 4252,
     "failures": [],
     "shards": {
       "total": 1,
       "failed": 0,
       "successful": 1
     }
   }
 ]
}
```

可以同时获得多个快照的信息，`GET`请求支持通配符方式匹配多个快照信息，这时候快照的名字代表的意义就起了作用如：

```
GET /_snapshot/es_repo/snapshot_order_*
```

可以使用`_all`参数获取一个仓库中所有快照的完整列表信息：

```
GET /_snapshot/es_repo/_all
```

停止或删除快照

百度智能云提供的Elasticsearch服务没有单独的停止快照API，停止快照和删除快照是一个语义，如果发现snapshot执行有错误或耗时非常长，可以通过删除这个snapshot来停止后台运行的snapshot：

```
DELETE /_snapshot/es_repo/snapshot_2018_07_01
```

从仓库中删除：

```
DELETE /_snapshot/es_repo/snapshot_2018_07_01
```

一个仓库亦可以直接被删除：

```
DELETE /_snapshot/es_repo
```

> 注意：删除一个snapshot或删除一个仓库，ES只是删除了集群对仓库或snapshot的位置引用，真实的物理文件等是需要用户自己处理的。当确定所有的快照都不再适用的时候，可以先在Elasticsearch中删除仓库的元数据，然后登录百度智能云BOS控制台手动直接删除仓库。
>
> 需要额外注意的是，**请不要手动修改或删除BOS仓库中的任何快照文件**，手动删除一个快照文件后，将会导致快照处于不可用的状态，恢复的时候会出现不可挽回的损失。

查看快照进度

可以通过`status`接口查看一个快照的进度信息：

```
GET /_snapshot/es_repo/snapshot_2018_07_19/_status
```

下面是`status`接口返回的详细统计信息：

```
{
 "snapshots": [
   {
     "snapshot": "snapshot_2018_07_19",
     "repository": "es_repo",
     "uuid": "TWKo55e7TSy1Sq4WLxMVrQ",
     "state": "SUCCESS",  ..................  [A]
     "shards_stats": {
       "initializing": 0,
       "started": 0,
       "finalizing": 0,
       "done": 1,
       "failed": 0,
       "total": 1
     },
     "stats": {
       "number_of_files": 16,
       "processed_files": 16,
       "total_size_in_bytes": 18639,
       "processed_size_in_bytes": 18639,
       "start_time_in_millis": 1531997598051,
       "time_in_millis": 2782
     },
     "indices": {
       "snapindex": {
         "shards_stats": {
           "initializing": 0,
           "started": 0,
           "finalizing": 0,
           "done": 1,  ..................... [B]
           "failed": 0,
           "total": 1
         },
         "stats": {
           "number_of_files": 16,
           "processed_files": 16,
           "total_size_in_bytes": 18639,
           "processed_size_in_bytes": 18639,
           "start_time_in_millis": 1531997598051,
           "time_in_millis": 2782
         },
         "shards": {
           "0": {
             "stage": "DONE",............... [C]
             "stats": {
               "number_of_files": 16,
               "processed_files": 16,
               "total_size_in_bytes": 18639,
               "processed_size_in_bytes": 18639,
               "start_time_in_millis": 1531997598051,
               "time_in_millis": 2782
             }
           }
         }
       }
     }
   }
 ]
}
```

相应中包括快照的所有信息例如快照的开始时间、总大小、文件总数和已经处理完的文件总数等，详细记录了被快照的所有的

index的当前状态以及index下的所有shard的状态等。

- [A]表示这个快照已经成功完成，显示`SUCCESS`状态，正在运行的快照会显示`IN_PROGRESS`
- [B]表示这个index的所有分片(shard)完成快照
- [C]表示这个index对应的分片(shard)完成快照

不同的状态值表征不同的含义：

状态值	意义
INIT	快照还没有开始，正在做初始化工作
STARTED	快照正在拷贝index文件
FINALIZE	快照的元数据正在写入远端仓库
DONE	快照成功完成
FAILURE	快照失败，失败的原因都可以在`status` API中看到

恢复快照

一个快照可以通过下面的命令被恢复(restore):

```
POST /_snapshot/es_repo/snapshot_2018_07_19/_restore
```

缺省情况下，指定快照中所有的index都会被恢复，可以通过在请求体中添加`indices`和`include_global_state`来指定特定的`index`和`global cluster state`:

```
POST /_snapshot/es_repo/snapshot_2018_07_19/_restore
{
  "indices": "snapindex",
  "ignore_unavailable": true,
  "include_global_state": true,
  "rename_pattern": "snap(.+)",
  "rename_replacement": "restore$1"
}
```

`rename_pattern`和`rename_replacement`可以用来对index重新命名，大多数的index的设置可以被重新设置，如下：

```
POST /_snapshot/repo/snapshot_wyf_2018_01_29/_restore
{
  "indices": "wyf",
  "index_settings": {
    "index.number_of_replicas": 0
  },
  "ignore_index_settings": [
    "index.refresh_interval"
  ]
}
```

需要注意的是：**一些设置如index.number_of_shards是不可以在restore的时候重新设置的**，可以restore到另外一个集群，新的cluster的version必须和做snapshot的集群一样或者更大(仅仅只能大1个major version).例如可以restore一个1.x的snapshot到2.x，但是不能到5.x。

和快照一样，`restore`请求在检查完快照信息和验证快照中的index信息后立即返回，恢复将在集群后台执行，可以在请求后面追加`wait_for_completion`参数阻塞请求之道恢复完成：

```
POST /_snapshot/es_repo/snapshot_2018_07_19/_restore?wait_for_completion=true
```

监控快照的恢复

从BOS仓库恢复数据利用了Elasticsearch内部的恢复机制，从内部实现原理来讲，从仓库恢复数据和一个节点从另一个节点上恢复数据是完全等价，Elasticsearch的内部恢复包括原地恢复(existing_store recovery)、远端恢复(peer recovery)和仓库恢复(snapshot recovery)。

恢复进度可以通过`recovery`API来查看：

```
GET /{index}/_recovery

GET snapindex/_recovery
```

这个接口返回的响应如下：

```
{
  "snapindex": {
    "shards": [
      {
        "id": 0,
        "type": "SNAPSHOT", ........................ [A]
        "stage": "DONE", ........................... [B]
        "primary": true,
        "start_time_in_millis": 1532065843418,
        "stop_time_in_millis": 1532065845773,
        "total_time_in_millis": 2354,
        "source": { ................................ [C]
          "repository": "es_repo",
          "snapshot": "snapshot_2018_07_19",
          "version": "5.5.0",
          "index": "snapindex"
        },
        "target": {
          "id": "8wR8Z38USImEeSO0SZ1_hA",
          "host": "192.168.16.5",
          "transport_address": "192.168.16.5:9300",
          "ip": "192.168.16.5",
          "name": "8wR8Z38"
        },
        "index": {
          "size": {
            "total_in_bytes": 18668,
            "reused_in_bytes": 0,
            "recovered_in_bytes": 18668,
            "percent": "100.0%" ................. [D]
          },
          "files": {
            "total": 16,
            "reused": 0,
            "recovered": 16,
            "percent": "100.0%"
          },
          "total_time_in_millis": 2148,
          "source_throttle_time_in_millis": 0,
          "target_throttle_time_in_millis": 0
        },
        "translog": {
          "recovered": 0,
          "total": 0,
          "percent": "100.0%",
          "total_on_start": 0,
          "total_time_in_millis": 158
        },
        "verify_index": {
          "check_index_time_in_millis": 0,
          "total_time_in_millis": 0
        }
      }
    ]
  }
}
```

- [A] `type`表示是从远端仓库快照中进行恢复
- [B] `stage`字段表示这个恢复已经完成

- [C] source字段表示这个恢复已经完成
- [D] percent字段表示恢复完成的百分比

取消正在进行的恢复

可以通过删除正在恢复的index，取消对这个index的恢复：

```
DELETE /snapindex
```

快照需要注意

- 一个集群同时只能有一个快照运行
- 有删除快照的操作时，不能进行快照

恢复需要注意

- restore的index 可以不存在，如果存在必须是closed状态。
- restore的index会完全覆盖之前的文件：即使文件完全相同，也会先删除原来old文件，然后在创建新new文件。
- restore过程会跳过translog recovery过程，创建出新的translog。
- 如果恢复的目标不是做snapshot的ES集群，而是一个新集群的话，需要在新集群里也创建一下repo，并且将readonly 参数指定为true。

集群配置

配置同义词

百度智能云Elasticsearch支持上传自定义的同义词文件。此外，百度智能云Elasticsearch也支持动态同义词文件更新，见动态同义词。

上传方式

1. 登录百度智能云Elasticsearch控制台，并点击集群名称，进入集群详情界面。
2. 在左侧导航栏，单击**配置修改**。
3. 点击“选择文件”，在弹出的页面选择需要上传的同义词文件，并“确认上传”。
4. 在词典列表里可以看见相应文件，即完成同义词词典上传。

注意事项

- 同义词文件需要为utf-8编码，扩展名为txt。
- 要求每行一个同义词表达式（表达式支持 Solr 规则 和 WordNet 规则），例如：

```
 肯德基,KFC => 肯德基
elasticsearch,ES
```

- 新上传/新变更的同义词文件对旧索引不生效，只对新建索引生效。
- 若旧索引需要使用新上传的同义词文件，需要重建索引（reindex）；已经被使用的同义词文件如果发生变更，需reopen索引（先将索引close，随后open索引），重新载入新的同义词配置。（但旧数据依旧没有使用新的同义词配置，需要reindex）
- 同义词文件单个文件最大为10M，上传文件总数最多为10个。

使用方式

上传同义词文件后，用户可以创建索引，验证同义词配置，并可以写入测试数据进行搜索测试。创建索引时需要配置**settings**和**mapping**，并且需要在**settings**中配置`"synonyms_path": "analysis/{your_dict_name}.txt"`。

具体使用方式如下：

上传文件

在百度智能云Elasticsearch控制台，上传文件`test_synonym.txt`，文件内容为：

```
肯德基,KFC => 肯德基
```

创建索引

```
PUT /test
{
  "settings": {
    "index": {
      "analysis": {
        "analyzer": {
          "my_analyzer": {
            "type": "custom",
            "tokenizer": "ik_smart",
            "filter": ["my_synonym"]
          }
        },
        "filter": {
          "my_synonym": {
            "type": "synonym",
            "synonyms_path": "analysis/test_synonym.txt"
          }
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "title": {
        "type": "text",
        "analyzer": "my_analyzer",
        "search_analyzer": "my_analyzer"
      }
    }
  }
}
```

验证同义词配置

```
GET test/_analyze
{
  "analyzer": "my_analyzer",
  "text": "KFC"
}
```

结果为：

```
 {
  "tokens": [
    {
      "token": "肯德基",
      "start_offset": 0,
      "end_offset": 3,
      "type": "SYNONYM",
      "position": 0
    }
  ]
}
```

写入数据测试

写入数据如下：

```
 POST test/_doc/1
{
   "title":"我喜欢吃肯德基"
}
```

查询语句如下：

```
 GET /test/_doc/_search
{
   "query":{
     "match": {
       "title": "KFC"
     }
   }
}
```

查询结果为：

```
{
 "took": 20,
 "timed_out": false,
 "_shards": {
   "total": 1,
   "successful": 1,
   "skipped": 0,
   "failed": 0
 },
 "hits": {
   "total": {
     "value": 1,
     "relation": "eq"
   },
   "max_score": 0.2876821,
   "hits": [
     {
       "_index": "test",
       "_type": "_doc",
       "_id": "1",
       "_score": 0.2876821,
       "_source": {
         "title": "我喜欢吃肯德基"
       }
     }
   ]
 }
}
```

详细信息见Elasticsearch官方配置同义词文档。

插件管理

Elasticsearch插件列表

百度智能云Elasticsearch支持开源Elasticsearch的所有插件，在此基础上又自研了部分插件，为您提供丰富的插件功能。本文介绍百度智能云Elasticsearch目前支持的系统插件。

插件类型

百度智能云Elasticsearch支持系统默认插件和自定义插件，自定义插件详情可查看上传与安装自定义插件。

系统默认插件

百度智能云Elasticsearch预置系统默认插件，可根据需求直接进行使用。

支持的默认插件及适用版本如下：

插件名称	插件描述	适用版本	是否默认安装
baidu-metrics	baidu-metrics插件，提供写入拒绝率、查询拒绝率、写入失败率、查询失败率等监控指标的采集功能。	5.5.0、5.6.16、6.5.3、6.8.14、7.4.2、7.10.2	是
analysis-baidu-nlp	Baidu NLP中文分词插件是百度智能云提供的基于自然语言理解的分词插件，具备更高的分词性能与准确率，同时还支持词典动态热更新。	6.5.3、6.8.14、7.4.2、7.10.2	是
baidu-knn	向量检索插件	7.4.2、7.10.2	是
baidu-pack	账号权限管理、Http Basic认证、冷热数据分离等功能	全部	是
analysis-ik	ik中文分词插件	全部	是
analysis-pinyin	拼音分词插件	全部	是
baidu-dynamic-synonym	动态同义词插件	7.4.2、7.10.2	是
mapper-size	mapper-size插件，记录doc在压缩前的大小	7.4.2、7.10.2	是
compression-zstd	zstandard（zstd）压缩插件	6.5.3、7.4.2、7.10.2	否
sql	sql解析插件	7.4.2、7.10.2	否
ccr	CCR（Cross-Cluster Replication 跨集群复制）插件	7.10.2	否
baidu-rate-limiting	限流插件	7.4.2、7.10.2	否
ingest-attachment	文档提取插件	除5.6.16版本外，其他版本均支持	是
repository-bos	基于BOS的快照插件	全部	是
repository-hdfs	基于HDFS的快照插件	除5.6.16版本外，其他版本均支持	是
analysis-stconvert	简繁体中文转换插件	除7.10.2版本外，其他版本均支持	否
analysis-jieba	结巴中文分词	6.5.3	否

插件安装 没有安装的系统插件，需要用户根据需求自选安装。

安装后需要用户操作重启集群，从而使插件生效。

BES支持两种重启方式，强制重启和滚动重启，详见文档，用户可以根据自己的场景进行选择

上传与安装自定义插件

当用户需要使用自研插件或百度智能云Elasticsearch默认插件中不包含的开源插件时，可通过百度智能云Elasticsearch的自定义插件上传与安装功能，安装并使用插件。本文介绍具体的操作方法。

前提条件

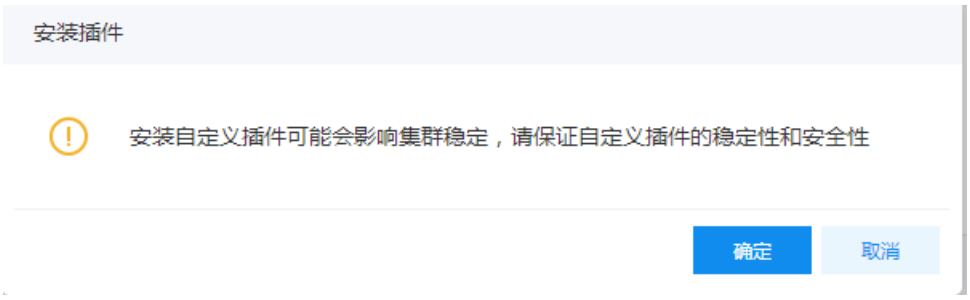
- 准备待上传的插件，并确保插件的可用性和安全性。
- 插件文件格式要求：文件名只能包含大写字母、小写字母、数字、连接线（-）或点（.），文件后缀名必须是.zip。
- 建议在上传插件前，先在本地自建Elasticsearch集群（与百度智能云Elasticsearch相同版本）上进行测试，成功后再进行上传。具体安装插件方式请参见 Installing Plugins。
- 只能上传安装百度智能云Elasticsearch集群默认插件以外的插件，您可以通过以下API查询当前集群已安装插件。

```
GET _cat/plugins?v
```

操作步骤

1. 登录百度智能云Elasticsearch。
2. 在集群列表中选择需要上传自定义插件的集群，点击进入**集群详情**。
3. 选择**插件配置**，进入插件列表。
4. 在**插件配置**页面，进入**自定义插件列表**，单击**选择文件**进行上传。

5. 在本地选择插件后点击**上传文件**按钮进行上传。
6. 上传完成后可在**自定义插件列表**中看到插件信息。
7. 点击**安装**并仔细阅读提示框中的内容，确认插件稳定后点击**确认**，等待插件**状态**显示为**已安装**即可。



8. 依次点击右上角的**停止**和**启动**，重启集群等待插件生效。

> 注：如果插件安装失败集群会一直停在**正在启动**的状态，可以通过**删除**或**卸载**插件使集群恢复正常。

9. 如果希望卸载插件，可单击插件右侧的**卸载**或**删除**，卸载此插件，然后**重启**集群生效。

系统默认插件

analysis-pinyin拼音分词插件

analysis-pinyin拼音分词插件是百度智能云Elasticsearch默认安装的分词插件。

> 注意：目前**analysis-pinyin拼音分词插件**支持所有版本的Elasticsearch实例。

使用方法

创建索引并使用自定义拼音分析器

```
PUT /test_index/
{
  "settings" : {
    "analysis" : {
      "analyzer" : {
        "pinyin_analyzer" : {
          "tokenizer" : "my_pinyin"
          }
      },
      "tokenizer" : {
        "my_pinyin" : {
          "type" : "pinyin",
          "keep_separate_first_letter" : false,
          "keep_full_pinyin" : true,
          "keep_original" : true,
          "limit_first_letter_length" : 16,
          "lowercase" : true,
          "remove_duplicated_term" : true
        }
      }
    }
  }
}
```

其中参数含义如下：

参数	含义	默认值	备注
keep_first_letter	保留拼音首字母分词，例如：`刘德华->ldh`。	true	
keep_separate_first_letter	拆分保留拼音首字母，例如：`刘德华->l,d,h`。	false	分词过细可能导致查询结果比较模糊
limit_first_letter_length	限制首字母最大长度	16	
keep_full_pinyin	拆分保留全拼，例如：`刘德华-> [liu,de,hua]`	true	
keep_joined_full_pinyin	合并保留全拼，例如：`刘德华-> liudehua`	false	
keep_none_chinese	保留非中文字符	true	
keep_none_chinese_together	保留非中文字符，并将非中文字符放在一起。例如：当设置为true，则`DJ音乐家 -> DJ,yin,yue,jia`；当设置为false，`DJ音乐家 -> D,J,yin,yue,jia`	true	需要先设置`keep_none_chinese`为true
keep_none_chinese_in_first_letter	将非中文字符放在首字母分词中，例如：`刘德华AT2016->ldhat2016`	true	
keep_none_chinese_in_joined_full_pinyin	将非中文字符放在全拼分词中，例如：`刘德华AT2016->liudehua2016`	false	
none_chinese_pinyin_tokenize	如果非中文字母是拼音的话，把它们分成单独的拼音。例如：`liudehuaalibaba13zhuanghan -> liu,de,hua,a,li,ba,ba,13,zhuang,han`	true	需要先设置`keep_none_chinese`和`keep_no`为true
keep_original	在分词的同时，保留原始输入内容	false	
lowercase	将非中文字符转化成小写字母	true	
trim_whitespace	去除空格	true	
remove_duplicated_term	删除重复的term，例如：`de的>de`	false	

测试拼音分词效果

```
GET /test_index/_analyze
{
  "text": ["刘德华"],
  "analyzer": "pinyin_analyzer"
}
```

```
 {
  "tokens" : [
   {
     "token" : "liu",
     "start_offset" : 0,
     "end_offset" : 1,
     "type" : "word",
     "position" : 0
   },
   {
     "token" : "de",
     "start_offset" : 1,
     "end_offset" : 2,
     "type" : "word",
     "position" : 1
   },
   {
     "token" : "hua",
     "start_offset" : 2,
     "end_offset" : 3,
     "type" : "word",
     "position" : 2
   },
   {
     "token" : "刘德华",
     "start_offset" : 0,
     "end_offset" : 3,
     "type" : "word",
     "position" : 3
   },
   {
     "token" : "ldh",
     "start_offset" : 0,
     "end_offset" : 3,
     "type" : "word",
     "position" : 4
   }
  ]
}
```

创建索引mapping

```
 POST /test_index/_mapping
{
        "properties": {
            "name": {
                "type": "keyword",
                "fields": {
                    "pinyin": {
                        "type": "text",
                        "store": false,
                        "term_vector": "with_offsets",
                        "analyzer": "pinyin_analyzer",
                        "boost": 10
                    }
                }
            }
        }

}
```

写入数据

```
POST /test_index/_doc
{"name":"刘德华"}
```

查询测试

```
POST test_index/_search?q=name.pinyin:刘德华
POST test_index/_search?q=name.pinyin:刘德
POST test_index/_search?q=name.pinyin:liu
POST test_index/_search?q=name.pinyin:ldh
POST test_index/_search?q=name.pinyin:de+hua
```

IK中文分词插件与动态更新词典

IK中文分词插件（英文名为analysis-ik）是百度智能云Elasticsearch默认安装的中文分词插件。

本文介绍了IK 中文分词插件的**使用方法**和**动态更新IK词典**的方法。

> 注意：目前**IK中文分词插件**支持所有版本的Elasticsearch实例。

使用方法

分词粒度

analysis-ik主要提供两种粒度的`Analyzer`：

1. ik_max_word：会将文本做最细粒度的拆分；尽可能多的拆分出词语
2. ik_smart：会做最粗粒度的拆分；已被分出的词语将不会再次被其它词语占有

同名提供两种`Tokenizer`：

1. ik_max_word
2. ik_smart

切词效果

ik_max_word的切词效果如下：

```
POST /_analyze
{
  "analyzer": "ik_max_word",
  "text":    "百度校园夏令营"
}
```

```
{
  "tokens": [
    {
      "token": "百度",
      "start_offset": 0,
      "end_offset": 2,
      "type": "CN_WORD",
      "position": 0
    },
    {
      "token": "百",
```

```
        "start_offset": 0,
        "end_offset": 1,
        "type": "TYPE_CNUM",
        "position": 1
      },
      {
        "token": "度",
        "start_offset": 1,
        "end_offset": 2,
        "type": "COUNT",
        "position": 2
      },
      {
        "token": "校园",
        "start_offset": 2,
        "end_offset": 4,
        "type": "CN_WORD",
        "position": 3
      },
      {
        "token": "夏令营",
        "start_offset": 4,
        "end_offset": 7,
        "type": "CN_WORD",
        "position": 4
      },
      {
        "token": "夏令",
        "start_offset": 4,
        "end_offset": 6,
        "type": "CN_WORD",
        "position": 5
      },
      {
        "token": "夏",
        "start_offset": 4,
        "end_offset": 5,
        "type": "CN_WORD",
        "position": 6
      },
      {
        "token": "令",
        "start_offset": 5,
        "end_offset": 6,
        "type": "CN_CHAR",
        "position": 7
      },
      {
        "token": "营",
        "start_offset": 6,
        "end_offset": 7,
        "type": "CN_CHAR",
        "position": 8
      }
    ]
  }
```

ik_smart的切词效果如下：

```
POST /_analyze
{
  "analyzer": "ik_smart",
  "text":    "百度校园夏令营"
}
```

```
{
  "tokens": [
    {
      "token": "百度",
      "start_offset": 0,
      "end_offset": 2,
      "type": "CN_WORD",
      "position": 0
    },
    {
      "token": "校园",
      "start_offset": 2,
      "end_offset": 4,
      "type": "CN_WORD",
      "position": 1
    },
    {
      "token": "夏令营",
      "start_offset": 4,
      "end_offset": 7,
      "type": "CN_WORD",
      "position": 2
    }
  ]
}
```

动态更新词典

配置词典

当IK自身带的词库不满足需求时，用户可以自定义词典。步骤如下：

1. 把编辑好的词典放到一个http服务器上，可以选择以下两种方案。

- 用户可以把词典文件上传到BOS，并获取链接地址。**需要注意的是，务必保证Elasticsearch所在地区与BOS Bucket的地区相同，否则无法访问词典。**

1.建议将词典文件的访问权限设置为`公共读`，确保链接地址永久有效，且不会发生变化。

2.若不设置为公共读，则需要按业务需求修改BOS链接的有效时间，如需永久有效，可将有效时间设为-1。**注意**，链接有效期过后或词典文件如果发生修改（删除后重新上传），链接地址会发生变化，此时需要重新在ES中配置链接地址。

- 用户也可以自行部署http服务器，将词库放在服务器上。**需要注意的是，务必保证Elasticsearch所在节点能够访问该http服务，否则无法下载安装。**

2. 在Elasticsearch里配置ik词库的http地址，比如自定义词库的配置文件是baidu.dict，停用词的配置文件是baidu_stop.dic时，向Elasticsearch发送的命令如下:

```
PUT /_cluster/settings
{
  "persistent": {
    "bpack.ik_analyzer.remote_ext_dict":"http://ip:port/baidu.dic",
    "bpack.ik_analyzer.remote_ext_stopwords":"http://ip:port/baidu_stop.dic"
  }
}
```

- Elasticsearch会每隔60s检测一下setting中http url指向的词库文件是否发生变化，如果变化了，那么es就会自动下载这个文件，然后加载到ik中。

验证词库是否生效

当配置好后，用户可以通过`POST /_analyze`来检测词库是否生效，比如：

- 在没有配置词库之前，分词效果为：

```
POST /_analyze
{
  "analyzer" : "ik_smart",
  "text" : ["赵小明明真帅"]
}
```

Elasticsearch的返回结果是：

```
{
  "tokens": [
    {
      "token": "赵",
      "start_offset": 0,
      "end_offset": 1,
      "type": "CN_WORD",
      "position": 0
    },
    {
      "token": "小明",
      "start_offset": 1,
      "end_offset": 3,
      "type": "CN_WORD",
      "position": 1
    },
    {
      "token": "明",
      "start_offset": 3,
      "end_offset": 4,
      "type": "CN_WORD",
      "position": 2
    },
    {
      "token": "真帅",
      "start_offset": 4,
      "end_offset": 6,
      "type": "CN_WORD",
      "position": 3
    }
  ]
}
```

- 然后配置自定义词典，其中正常词库里只包含 "赵小明明"， 停用词词库中包含 "真帅"，配置完之后，再次调用`POST /_analyze`，结果如下：

```
{
  "tokens": [
    {
      "token": "赵小明明",
      "start_offset": 0,
      "end_offset": 4,
      "type": "CN_WORD",
      "position": 0
    }
  ]
}
```

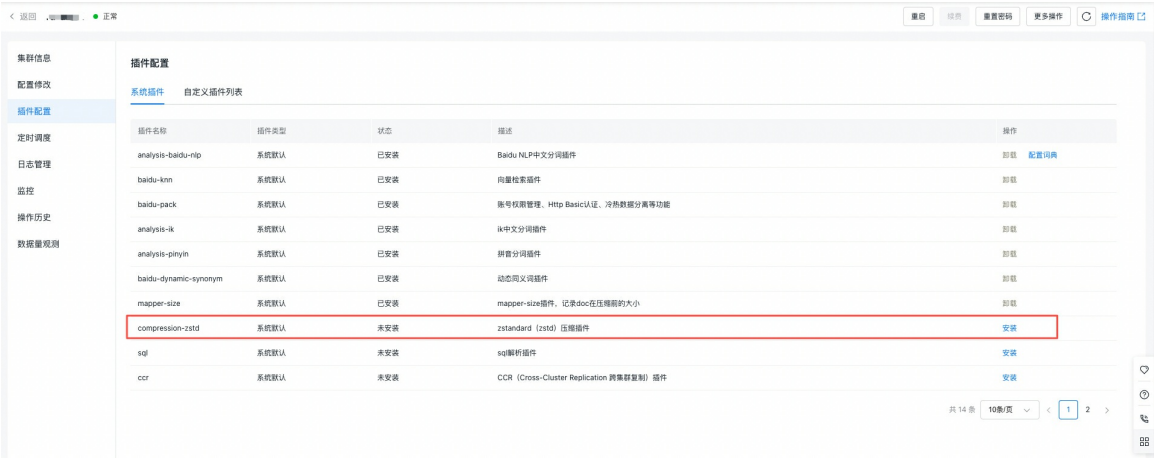
结果显示，“赵小明明” 被当做一个单独的词，而 "真帅" 由于是停用词所以没有被分词。

compression-zstd插件

背景说明 Elasticsearch提供的数据存储方式，默认使用LZ4算法压缩，压缩速度较快，但压缩率较低。也可选择使用Deflate算法获得更好的压缩率，但牺牲了压缩速度。

BES提供了业内先进的压缩算法 Zstandard（ZSTD），在提升压缩率的同时，保证了压缩速度。

使用方式 1.在控制台安装BES提供的compression-zstd插件，点击安装后需要重启集群使插件生效。



2.在创建索引时，如需指定压缩算法，做如下配置即可：

```
PUT my_index
{
  "settings": {
    "index.codec": "zstd"
  }
}
```

配置参数与压缩算法的对应关系：

index.codec取值	压缩算法
default	LZ4
best_compression	Deflate
zstd	Zstandard

性能参数 在同等规格配置的集群内，对上述三种压缩算法，分别使用esrally工具写入pmc数据集进行写入性能测试，结果如下：

序号	压缩算法	分片数	存储空间（force merge后）	平均写入速率
1	Zstandard	1	14.6GB	843.25doc/s
2	LZ4（default，ES默认）	1	18.8GB	779.21doc/s
3	Deflate（best_compression）	1	14.1GB	655.51doc/s

- 结论：

  在压缩率方面，Zstandard方式的数据存储空间比默认LZ4方式节省22.34%，比Deflate多3.55%。

  在压缩速度方面，Zstandard方式的数据写入速率比默认LZ4方式快8.22%，比Deflate快28.64%。

  综合来看，Zstandard方式兼具了优秀的压缩率和压缩速度。

CCR插件

插件介绍 CCR（Cross Cluster Replication）插件是BES提供的跨集群复制插件，支持将一个BES集群（Leader集群）中的数据实时复制到另一个集群（Follower集群）。

> **注意**：只在7.10.2版本提供

插件安装

在控制台的集群详情页面，【插件配置】-> 【系统插件】中提供了CCR插件，找到CCR插件，点击安装即可完成插件安装。

> **注意1**：安装后需要重启集群插件才可正常使用
>
> **注意2**：主集群（Leader集群）、从集群（Follower集群）都需要安装

使用说明 使用前置条件：主集群（leader集群）、从集群（follower集群）都需要安装CCR插件，且集群间网络互通。

使用CCR启动、停止索引同步的步骤是：1.从集群设置连接的主集群信息、2.从集群配置同步索引、3.从集群停止同步索引，具体的操作指令与说明详见下述内容。

一、从集群设置连接的主集群信息： 该步骤是使用需要配置的第一个步骤，用于从集群（Follower集群）中配置需要连接的主集群（Leader集群）地址信息。一个从集群可以配置多个主集群（Leader集群），具体的指令与说明详见下述内容。

格式说明

```
 PUT /_cluster/settings
{
  "persistent": {
    "cluster": {
      "remote": {
        "{主集群1-自定义名称}": {
          "seeds": [
            "{主集群1节点blb ip:tcp端口}","{主集群1节点blb ip:tcp端口}","{主集群1节点blb ip:tcp端口}"
          ]
        },
        "{主集群2-自定义名称}": {
          "seeds": [
            "{主集群2节点ip:tcp端口}","{主集群2节点ip:tcp端口}","{主集群2节点ip:tcp端口}"
          ]
        }
      }
    }
  }
}
```

> **说明**：
>
> 主集群ip有两种配置方式
>
> 使用vpc_blb_ip：推荐使用该方式配置，如上述配置的主集群1， 配置简单且蓝绿变配后对同步无影响，由于是新增配置方式，部分存量集群可能不支持，可提工单人工处理 ，tcp端口为 9200
>
> 使用vpc_ip：如上述配置的主集群2，需要把主集群上所有vpc_ip全部配置到从集群， 且如果集群有蓝绿操作，需要重新配置，影响同步任务，tcp端口为 9200

使用举例

```
PUT _cluster/settings
{
  "persistent": {
    "cluster": {
      "remote": {
        "leader-cluster1": {
          "seeds": [
            "192.1.1.1:9200"
          ]
        },
        "leader-cluster2": {
          "seeds": [
            "192.1.2.4:9200", "192.1.2.5:9200", "192.1.2.6:9200"
          ]
        }
      }
    }
  }
}
```

> **说明**：上述ip地址为示例地址，实际使用时请填写对应主集群的 vpc_blb_ip或vpc_ip

二、从集群配置同步索引： 完成上述步骤操作后，既可以在从集群（Follower集群）进行同步索引的配置操作，开启数据同步过程。

格式说明

```
PUT /_ccr/_replication/{从集群索引名}/_start
{
  "leader_alias": "{主集群1-自定义名称}",
  "leader_index": "{主集群1索引名}"
}
```

> **注意**：新建从集群同步索引名，不能是从集群已存在索引的名称
>
> **说明1**：一次同步索引配置操作只能设置一个从索引
>
> **说明2**：新建从集群索引和主集群关联索引的名称不需要保持一致
>
> **说明3**：新建从集群同步索引，先将主集群存量索引同步到从集群中，后持续同步增量索引

使用举例

```
PUT /_ccr/_replication/follower_index/_start
{
  "leader_alias": "leader-cluster1",
  "leader_index": "leader_index"
}
```

三、修改从索引配置： 索引建立同步关系后从索引的配置不可修改而是随着主索引的变化而变化，如果需要修改从索引的配置则在从集群上执行以下命令 **格式说明**

```
PUT /_ccr/_replication/{从集群索引名}/_update
{}
```

使用举例 修改从索引副本分片数

```
PUT /_ccr/_replication/follow_index/_update
{
  "settings":{
    "index.number_of_replicas": 2
  }
}
```

四、从集群停止同步索引： 完成上述步骤后，如果需要停止同步索引，可以在从集群（Follower集群）通过下述指令完成操作。

格式说明

```
POST /_ccr/_replication/{从集群索引名}/_stop
{}
```

使用举例

```
POST /_ccr/_replication/follower_index/_stop
{}
```

五、从集群暂停/恢复同步索引： 同步关系建立后如果想暂停同步，可以在从集群执行以下命令 **格式说明**

```
POST /_ccr/_replication/{从集群索引名}/_pause
{}
```

> **注意**：
>
> **说明1**：同步关系的暂停时间不能超过12小时，否则需要重新进行同步

暂停同步后如果想要恢复，可以在从集群上执行以下命令 **格式说明**

```
POST /_ccr/_replication/{从集群索引名}/_resume
{}
```

六、创建/删除auto-follow自动同步 如果想要按特定的索引名称规则去自动同步主集群上的索引，可以使用自动同步功能，在从集群执行以下命令 **格式说明**

```
POST /_ccr/_replication/_autofollow
{
   "leader_alias": "{主集群1-自定义名称}",
   "name": "{同步规则名称}",
   "pattern": "{主集群1索引匹配规则}"
}
```

> **注意**：
>
> **说明1**：按照设置的匹配规则同步主集群索引时，如果从集群上存在同名索引，则从集群会跳过该索引
>
> **说明2**：可以使用相同命令更新索引匹配规则

使用举例

```
POST /_ccr/_replication/_autofollow
{
  "leader_alias": "leader-cluster1",
  "name": "auto-follow-pattern-1",
  "pattern": "test*"
}
```

如果想要停止按名称规则去同步主集群索引，需要删除该规则，在从集群上执行以下命令 **格式说明**

```
DELETE /_ccr/_replication/_autofollow
{
  "leader_alias" : "{主集群1-自定义名称}",
  "name": "{同步规则名称}"
}
```

使用举例

```
DELETE /_ccr/_replication/_autofollow
{
  "leader_alias" : "leader-cluster1",
  "name": "auto-follow-pattern-1"
}
```

七、查看同步规则

设置同步规则之后，可以在从集群通过下述命令查看配置的所有规则 **格式说明**

```
GET /_ccr/_replication/autofollow_stats
```

响应示例

```
{
  "num_success_start_replication": 2,
  "num_failed_start_replication": 0,
  "num_failed_leader_calls": 0,
  "failed_indices":[

  ],
  "autofollow_stats":[
    {
      "name":"auto-follow-pattern-1",
      "pattern":"test*",
      "num_success_start_replication": 2,
      "num_failed_start_replication": 0,
      "num_failed_leader_calls": 0,
      "failed_indices":[

      ]
    }
  ]
}
```

八、查看同步状态 支持查看集群级、索引级、分片级的同步状态，在从集群执行命令

集群级同步状态 格式说明

```
GET /_ccr/_replication/follower_stats
```

响应示例

```
{
  "num_syncing_indices": 2,
  "num_bootstrapping_indices": 0,
  "num_paused_indices": 0,
  "num_failed_indices": 0,
  "num_shard_tasks": 2,
  "num_index_tasks": 2,
  "operations_written": 3,
  "operations_read": 3,
  "failed_read_requests": 0,
  "throttled_read_requests": 0,
  "failed_write_requests": 0,
  "throttled_write_requests": 0,
  "follower_checkpoint": 1,
  "leader_checkpoint": 1,
  "total_write_time_millis": 2290,
  "index_stats":{
     "follower-index-1":{
        "operations_written": 2,
        "operations_read": 2,
        "failed_read_requests": 0,
        "throttled_read_requests": 0,
        "failed_write_requests": 0,
        "throttled_write_requests": 0,
        "follower_checkpoint": 1,
        "leader_checkpoint": 1,
        "total_write_time_millis": 1355
     },
     "follower-index-2":{
        "operations_written": 1,
        "operations_read": 1,
        "failed_read_requests": 0,
        "throttled_read_requests": 0,
        "failed_write_requests": 0,
        "throttled_write_requests": 0,
        "follower_checkpoint": 0,
        "leader_checkpoint": 0,
        "total_write_time_millis": 935
     }
  }
}
```

> **注意**：
>
> **说明1**：第一层的follower_checkpoint和leader_checkpoint代表集群上所有建立同步关系索引的同步进度之和，相等则代表该集群上所有索引均已完全同步

索引级同步状态 格式说明

```
GET /_ccr/_replication/{从集群索引名}/_status
```

响应示例

```
{
 "status" : "SYNCING",
 "reason" : "User initiated",
 "leader_alias" : "leader",
 "leader_index" : "leader-index",
 "follower_index" : "follower-index",
 "syncing_details" : {
   "leader_checkpoint" : 19,
   "follower_checkpoint" : 19,
   "seq_no" : 0
 }
}
```

> **注意**：
>
> **说明1**：leader_checkpoint和follower_checkpoint如果值相同代表索引完全同步

分片级同步状态 如果想进一步看分片级同步状态，则需要在上述命令后加上&verbose=true参数 **格式说明**

```
GET /_ccr/_replication/{从集群索引名}/_status&verbose=true
```

响应示例

```
{
 "status" : "SYNCING",
 "reason" : "User initiated",
 "leader_alias" : "leader",
 "leader_index" : "leader-index",
 "follower_index" : "follower-index",
 "syncing_details" : {
   "leader_checkpoint" : 7,
   "follower_checkpoint" : 7,
   "seq_no" : 8
 },
 "shard_replication_details": [
     {
        "shard_id": "[follower-index][0]",
        "syncing_task_details": {
           "leader_checkpoint": 7,
           "follower_checkpoint": 7,
           "seq_no": 8
        }
     }
  ]
}
```

限流插件

百度智能云Elasticsearch（以下简称为ES）提供了限流插件，目的是为了提高集群的稳定性。该插件能够实现集群级别的读写限流，在关键时刻把指定索引的流量控制在合适范围内。例如当上游业务无法进行流量控制时，尤其对于读请求业务，可根据限速插件设置的规则进行限流，来保护Elasticsearch服务的稳定性。

> 注意：目前限流插件支持7.4.2、7.10.2版本的Elasticsearch实例。

插件安装 百度智能云ElasticSearch在控制台的集群详情页面，【插件配置】-> 【系统插件】中提供了限流插件，找到限流插件

（baidu-rate-limiting），点击安装即可完成插件安装。

baidu-rate-limiting	系统默认	未安装	BES 限流插件	安装

> 注意：需要重启集群后插件才可正常使用

限流规则

用户可先参考如下规则评估限流规则：

限流请求类型

- 查询请求

> 查询请求的限流阈值 = 客户端查询请求到达Elasticsearch的QPS（Query Per Second）

> **注意**：QPS指查询请求到达ES集群的每秒请求数。

- 写入请求

> 写入请求的限流阈值的计算规则与查询请求类似，可以限制Bulk每秒写入的总字节数，也可以限制单次写入的最大字节数。

限流级别

- 节点级别

每个节点都会按照当前设定的限流规则进行限流，比如设定查询限流QPS为 100/s，则每个节点可以承受100/s的查询请求。默认限流级别为`节点级别`。

- 集群级别

整个集群按照当前设定的限流规则进行限流，比如设定查询限流QPS为 100/s，则所有节点总计可以承受100/s的查询请求。

> **注意**：集群级别限流需要在集群的主节点进行流量统计，而大型在线业务使用集群级别限流会导致主节点计算压力过大。**因此大型在线业务限流建议使用节点级别，集群级别限流仅建议测试集群使用**。

使用方法

开启限流功能

限流插件的限流功能默认关闭，使用时需要先开启该功能。

```
 PUT _cluster/settings
{
  "persistent" : {
    "bpack.ratelimit.enabled":"true"
  }
}
```

设置限流级别

- 设置为节点级别（默认为节点级别）

```
put _cluster/settings
{
  "persistent": {
    "bpack.ratelimit.level": "NODE"
  }
}
```

- 设置为集群级别

```
put _cluster/settings
{
  "persistent": {
    "bpack.ratelimit.level": "CLUSTER"
  }
}
```

设置查询限流

通过定义`index_patterns`设置目标索引的查询QPS（Query Per Second），支持索引和索引通配符。

设置单个索引的查询QPS

```
PUT _bpack/_ratelimit/<limitName>
{
  "type" : "search",
  "index_patterns" : "twitter",
  "max_queries_per_sec" : 1000
}
```

设置指定名称前缀的索引的查询QPS

```
PUT _bpack/_ratelimit/<limitName>
{
  "type" : "search",
  "index_patterns" : "nginx-log-*",
  "max_queries_per_sec" : 1000
}
```

设置所有索引的查询QPS

```
PUT _bpack/_ratelimit/<limitName>
{
  "type" : "search",
  "index_patterns" : "*",
  "max_queries_per_sec" : 1000
}
```

限流示例

当设置完成后，如果查询QPS超过`max_queries_per_sec`设置的值，系统会显示如下报错信息。请根据报错信息适当减少查询QPS。

```
{
  "error": {
    "root_cause": [
      {
        "type": "rate_limiting_exception",
        "reason": "search request indices:data/read/search, limited by [search limiting:aaa:1.0]."
      }
    ],
    "type": "rate_limiting_exception",
    "reason": "search request indices:data/read/search, limited by [search limiting:aaa:1.0]."
  },
  "status": 429
}
```

设置写入限流

设置Bulk每秒写入大小

通过设置Bulk每秒写入的总字节数，限制协调节点每秒接收的写入字节数。

```
PUT _bpack/_ratelimit/<limitName>
{
  "type" : "write",
  "index_patterns": "nginx-log-*",
  "max_throughput_in_bytes" : 1000000
}
```

设置Bulk单次请求大小

通过设置Bulk单次请求的最大值，限制ES集群接收单次请求的写入字节数。

```
PUT _bpack/_ratelimit/<limitName>
{
  "type" : "write",
  "index_patterns": "nginx-log-*",
  "max_request_size_in_bytes" : 1000
}
```

限流示例

当设置完成后，如果每秒写入的字节数超过`max_throughput_in_bytes`的值或单次写入的字节数超过`max_request_size_in_bytes`的值，系统会显示如下报错信息。请根据报错信息适当减少每秒写入的字节数。

限制每秒写入大小：

```
{
  "error": {
    "root_cause": [
      {
        "type": "rate_limiting_exception",
        "reason": "write request indices:data/write/bulk, limited by [write limiting:nginx-log-1:100.0]."
      }
    ],
    "type": "rate_limiting_exception",
    "reason": "write request indices:data/write/bulk, limited by [write limiting:nginx-log-1:100.0]."
  },
  "status": 413
}
```

限制单次请求大小：

```
{
  "error": {
    "root_cause": [
      {
        "type": "rate_limiting_exception",
        "reason": "write request indices:data/write/bulk, limited by [write limiting:nginx-log-1:max_size(10), but 73]."
      }
    ],
    "type": "rate_limiting_exception",
    "reason": "write request indices:data/write/bulk, limited by [write limiting:nginx-log-1:max_size(10), but 73]."
  },
  "status": 413
}
```

查看限流配置

- 获取所有限流配置

```
GET _bpack/_ratelimit
```

- 获取单个指定的限流配置

```
GET _bpack/_ratelimit/<limitName>
```

- 获取多个指定的限流配置

```
GET _bpack/_ratelimit/<limitName1,limitName2>
```

删除限流配置

```
DELETE _bpack/_ratelimit/<limitName>
```

关闭限流功能

```
 PUT _cluster/settings
{
  "persistent" : {
    "bpack.ratelimit.enabled":"false"
  }
}
或
PUT _cluster/settings
{
  "persistent" : {
    "bpack.ratelimit.enabled":null
  }
}
```

动态同义词插件

背景

词干提取是通过简化他们的词根形式来扩大搜索的范围，同义词通过相关的观念和概念来扩大搜索范围。 也许没有文档匹配查询 “英国女王“ ，但是包含 “英国君主” 的文档可能会被认为是很好的匹配。用户搜索 “美国” 并且期望找到包含 美利坚合众国 、

美国 、 美洲 、或者 USA 的文档。

百度智能云Elasticsearch提供的动态同义词插件可以满足上述需求，同义词词典主要包括一个可选的内置的汉语同义词词典和用户提供的远端可更新的同义词词库，用户需要在创建索引的时候对需要进行同义展开的字段进行设定。

> 注意：目前**动态同义词插件**支持7.4.2、7.10.2版本的Elasticsearch实例。

使用方式

创建索引并自定义分词器

```
 PUT synonym_test
{
  "settings": {
    "index": {
      "number_of_replicas": 0,
      "number_of_shards": 1,
      "analysis": {
        "analyzer": {
          "sy": {
            "type": "custom",
            "tokenizer": "bd-nlp-phrase",
            "filter": [
              "remote_synonym",
              "lowercase"
            ]
          }
        },
        "filter": {
          "remote_synonym": {
            "type": "bd_synonym",
            "expand": true,
            "synonyms_path": "http://{ip}:{port}/synonym.txt",
            "reload_interval": 30
          }
        }
      }
    }
  }
}
```

参数说明：

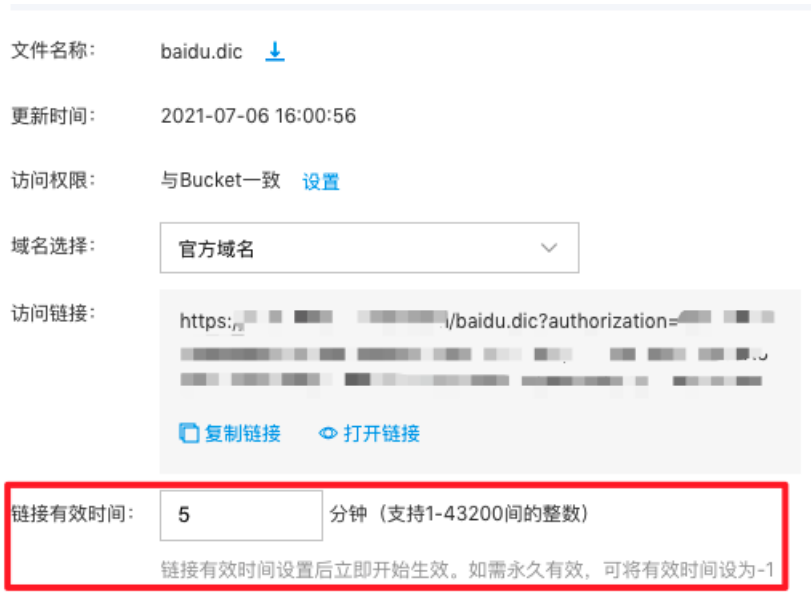
参数	说明
type	固定为bd_synonym
expand	对同义词词进行取代替换或展开，默认为`true`即展开
synonyms_path	远程词典的地址，需要用户配置并且保证ES集群可以访问（可以上传到BOS获取BOS访问链接，详见下文）
reload_interval	检测远程词典变化的时间间隔，默认为30s
need_tokenize	是否对同义词词典中的单个词进行分词，默认为false并采用`keyword` tokenizer
local	是否启用内置的本地同义词库，默认为true，我们提供了一个基础的中文同义词词典但并不保证效果
format	支持`solr`和`wordnet`类型的同义词词典，建议设置值采用默认值`solr`，设定其他值按`solr`处理
analysis_mode	同义语义作用的分析阶段，`all`、`index`、`search`，默认为`all`
lenient	同义词词典中如果有行解析错误是否跳过，默认为true

上传词典到BOS

用户可以把词典文件上传到BOS，并获取链接地址。**需要注意的是，务必保证Elasticsearch所在地区与BOS Bucket的地区相同，否则无法访问词典。**

1.建议将词典文件的访问权限设置为公共读，确保文件永久有效，且链接地址不会发生变化。

2.若不设置为公共读，则需要按业务需求修改BOS链接的有效时间，如需永久有效，可将有效时间设为-1。**注意**，链接有效期过后或词典文件如果发生修改（删除后重新上传），链接地址会发生变化，此时需要重新在Elasticsearch中配置链接地址。



词典说明

同义词最简单的表达形式是用逗号分隔：

"jump,leap,hop"

如果遇到这些词项中的任何一项，则将其替换为所有列出的同义词

```
  原始词项  →   取代
———————————————————————————————————————————————————————————
jump            → (jump,leap,hop)
leap            → (jump,leap,hop)
hop             → (jump,leap,hop)
```

或者, 使用 => 语法，可以指定一个词项列表（在左边），和一个或多个替换（右边）的列表：

```
  "usa,united states,united states of america => usa"
"gb,gb,great britain => britain,england,scotland,wales"
```

```
  原始词项  →   取代
———————————————————————————————————————————————————————————
u s a           → (usa)
united states   → (usa)
great britain   → (britain,england,scotland,wales)
```

如果多个规则指定同一个同义词，它们将被合并在一起，且顺序无关，否则使用最长匹配

一个简单的中文同义词词典样例:

```
  肯德基,KFC,开封菜
America, Unitied States, 美国, 美利坚合众国
中国,china,中华人民共和国
elasticsearch,ES
公有云,云计算
AI,人工智能,智能
机器人,robot,机械机构
```

说明:

- 每行是一个同义词组
- 用英文逗号进行分割

_analyze API测试

```
  POST test/_analyze
{
    "text": "KFC",
    "analyzer": "sy"
}
```

分词结果:

```
{
 "tokens": [
   {
     "token": "kfc",
     "start_offset": 0,
     "end_offset": 3,
     "type": "WORD",
     "position": 0
   },
   {
     "token": "金拱门",
     "start_offset": 0,
     "end_offset": 3,
     "type": "SYNONYM",
     "position": 0
   },
   {
     "token": "开封菜",
     "start_offset": 0,
     "end_offset": 3,
     "type": "SYNONYM",
     "position": 0
   }
 ]
}
```

注意

创建索引时会以同步的方式请求远端词典，所以创建索引的时间可能会比较长。

简繁体转换插件

插件介绍 简繁体转换插件可以帮助用户完成简体中文与繁体中文的转换。用户可以通过该插件的转换功能，使用中文繁体关键字搜索出包含对应中文简体的索引数据，同样也可以使用中文简体关键字搜索出包含对应中文繁体的索引数据。

> 注意：目前**简繁体转换插件**支持除7.10.2版本外所有版本的Elasticsearch实例。

插件安装

百度云ElasticSearch在控制台的集群详情页面，【插件配置】-> 【系统插件】中提供了简繁体转换插件，找到简繁体转换插件，点击安装即可完成插件安装。

> **注意**：需要重启集群后插件才可正常使用

使用说明及示例

简繁体转换插件的转换类型包含两种：

s2t：将简体中文转换为繁体中文。

t2s：将繁体中文转换为简体中文。

使用示例： 创建索引

```
PUT /stconvert/
{
  "settings": {
    "analysis": {
      "analyzer": {
        "st_ik": {
          "tokenizer": "ik_smart",
          "char_filter": [
            "tsconvert",
            "stconvert"
          ]
        }
      },
      "char_filter": {
        "tsconvert": {
          "type": "stconvert",
          "convert_type": "t2s"
        },
        "stconvert": {
          "type": "stconvert",
          "convert_type": "s2t"
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "desc": {
        "type": "text",
        "analyzer": "st_ik"
      }
    }
  }
}
```

写入数据

```
POST /stconvert/_doc/1
{
   "desc": "簡繁體轉換插件"
}
```

数据搜索

```
GET /stconvert/_search
{
   "query": {
      "match": {
        "desc": "转换"
      }
   }
}
```

搜索结果

```
{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 0.5753642,
    "hits": [
      {
        "_index": "stconvert",
        "_type": "_doc",
        "_id": "1",
        "_score": 0.5753642,
        "_source": {
          "desc": "簡繁體轉換插件"
        }
      }
    ]
  }
}
```

可以看到使用简繁体插件转换后，我们可以通过简体关键字查询繁体结果。

分词结果

使用以下命令查看我们自定义分词器的分词结果

```
POST /stconvert/_analyze
{
  "analyzer": "st_ik",
  "text": "简繁体转换插件"
}
```

返回结果

```
{
 "tokens": [
   {
     "token": "簡",
     "start_offset": 0,
     "end_offset": 1,
     "type": "CN_CHAR",
     "position": 0
   },
   {
     "token": "繁",
     "start_offset": 1,
     "end_offset": 2,
     "type": "CN_WORD",
     "position": 1
   },
   {
     "token": "體",
     "start_offset": 2,
     "end_offset": 3,
     "type": "CN_CHAR",
     "position": 2
   },
   {
     "token": "轉",
     "start_offset": 3,
     "end_offset": 4,
     "type": "CN_CHAR",
     "position": 3
   },
   {
     "token": "換",
     "start_offset": 4,
     "end_offset": 5,
     "type": "CN_CHAR",
     "position": 4
   },
   {
     "token": "插件",
     "start_offset": 5,
     "end_offset": 7,
     "type": "CN_WORD",
     "position": 5
   }
 ]
}
```

可以看到，实际搜索时已经将简体转换为繁体进行分词并执行搜索，与直接使用繁体进行分词搜索的结果是一样的。

ingest attachment插件

简介

Elasticsearch的ingest attachment插件可以将常用格式的文件作为附件写入Index。ingest attachment插件通过使用*Apache Tika*来提取文件，支持的文件格式有TXT、DOC、PPT、XLS和PDF等。

> 注意：
>
> - 源字段必须是base64编码的二进制。

- 目前**ingest attachment插件**支持除5.6.16版本外所有版本的Elasticsearch实例。

ingest attachment的pipeline参数含义

Name	是否必须	Default	Description
field	yes	-	从这个字段中获取base64编码
target_field	no	attachment	用于保留attachment信息，主要用于多附件的情况
indexed_chars	no	100000	限制字段的最大保存字符数。`-1`为无限制。
indexed_chars_field	no	-	可以从数据中设定的字段取到`indexed_chars`限制的值。
properties	no	全属性	选择需要存储的属性。例如 `content`, `title`, `name`, `author`, `keywords`, `date`, `content_type`, `content_length`, `language`
ignore_missing	no	false	如果使用`true`，并且 `field` 不存在, 则会忽略附件直接写入doc；否则则会报错。

使用方法：

单附件

1、创建pipeline

```
PUT _ingest/pipeline/attachment
{
  "description" : "Extract attachment information",
  "processors" : [
    {
      "attachment" : {
        "field" : "data"
      }
    }
  ]
}
```

2、向Elasticsearch中写入数据

我们可以通过两种方式将附件写入Elasticsearch：

（1）直接写入附件的base64编码

```
POST my_index/_doc?pipeline=attachment
{
  "filename": "test.docx",
  "data": "5rWL6K+V5paH5Lu25pWw5o2udGVzdA=="
}
```

其中，`5rWL6K+V5paH5Lu25pWw5o2udGVzdA==`是附件内容转化为base64的结果。

（2）通过Linux的curl命令直接将附件写入Elasticsearch

```
curl -u user:pwd -H 'Content-Type: application/json' -XPOST IP:PORT/my_index/_doc?pipeline=attachment -d'
{
  "filename": "test.docx",
  "data": "'`base64 -w 0 /test_path/test.docx | perl -pe 's/\n/\\n/g'`'"
}'
```

通过这种方式，`test.docx`文件可以直接写入Elasticsearch。

3、查询数据

```
GET my_index/_search
{
   "query": {
      "match": {
         "attachment.content": "test"
      }
   }
}
```

得到结果如下：

```
 {
  "took": 292,
  "timed_out": false,
  ...
       {
          "_index": "my_index",
          "_type": "_doc",
          "_id": "RL_1CXMByfR8Yao01Qs9",
          "_score": 1.9692057,
          "_source": {
            "filename": "test.docx",
            "data": "5rWL6K+V5paH5Lu25pWw5o2udGVzdA==",
            "attachment": {
              "content_type": "text/plain; charset=UTF-8",
              "language": "lt",
              "content": "测试文件数据test",
              "content_length": 10
            }
          }
       }
  ...
}
```

限制附件内容长度

1、创建pipeline

```
 PUT _ingest/pipeline/attachment
{
  "description" : "Extract attachment information",
  "processors" : [
    {
      "attachment" : {
        "field" : "data",
        "indexed_chars" : 6,
        "indexed_chars_field" : "my_field_size"
      }
    }
  ]
}
```

2、写入数据

```
 POST /my_index/_doc?pipeline=attachment
{
  "filename": "test.docx",
  "data": "5rWL6K+V5paH5Lu25pWw5o2udGVzdA=="
}
POST /my_index/_doc?pipeline=attachment
{
  "filename": "test2.docx",
  "my_field_size": 2
  "data": "5rWL6K+V5paH5Lu25pWw5o2udGVzdA=="
}
```

我们写入两个带有附件的doc，附件内容都是：*测试文件数据test*。其中第二个doc，我们将`indexed_chars_field`设定的字段`my_field_size`的值设定为*2*。

注：这里只列举了直接写入base64的方式。

3、查询数据

```
 GET my_index/_search
{
   "query": {
      "match_all": {}
   }
}
```

得到结果如下：

```
{
 "took": 292,
 "timed_out": false,
 ...
     {
      "_index": "my_index",
      "_type": "_doc",
      "_id": "RL_1CXMByfR8Yao01Qs9",
      "_score": 1.9692057,
      "_source": {
        "filename": "test.docx",
        "data": "5rWL6K+V5paH5Lu25pWw5o2udGVzdA==",
        "attachment": {
          "content_type": "text/plain; charset=UTF-8",
          "language": "lt",
          "content": "测试文件数据",
          "content_length": 6
        }
      }
    },
    {
      "_index": "my_index",
      "_type": "_doc",
      "_id": "RL_1CXMByfR8Yao01Qs9",
      "_score": 1.9692057,
      "_source": {
        "filename": "test2.docx",
        "my_field_size": 2,
        "data": "5rWL6K+V5paH5Lu25pWw5o2udGVzdA==",
        "attachment": {
          "content_type": "text/plain; charset=UTF-8",
          "language": "lt",
          "content": "测试",
          "content_length": 2
        }
      }
    }
 ...
}
```

可以看到，第一个doc，附件内容保留了6个字符，而第二个文档，保留了2个字符。

多附件

1、创建pipeline

```
PUT _ingest/pipeline/attachment
{
  "description" : "Extract attachment information from arrays",
  "processors" : [
    {
      "foreach": {
        "field": "attachments",
        "processor": {
          "attachment": {
            "target_field": "_ingest._value.attachment",
            "field": "_ingest._value.data"
          }
        }
      }
    }
  ]
}
```

需要注意的是，多附件的情况下，`field`和`target_field`必须要写成`_ingest._value.*`，否则不能匹配正确的字段。

2、写入多附件数据

```
POST /my_index/_doc?pipeline=attachment
{
  "attachments" : [
    {
      "filename" : "ipsum.txt",
      "data" : "dGhpcyBpcwpqdXN0IHNvbWUgdGV4dAo="
    },
    {
      "filename" : "test.txt",
      "data" : "VGhpcyBpcyBhIHRlc3QK"
    }
  ]
}
```

注：这里只列举了直接写入base64的方式。

3、查询数据

```
GET my_index/_search
{
   "query": {
      "match_all": {}
   }
}
```

得到结果如下：

```
{
 "took": 292,
 "timed_out": false,
 ...
    {
      "_index": "my_index",
      "_id": "RL_1CXMByfR8Yao01Qs9",
      "_version": 1,
      "_seq_no": 50,
      "_primary_term": 1,
      "found": true,
      "_source": {
        "attachments": [
          {
            "filename": "ipsum.txt",
            "data": "dGhpcyBpcwpqdXN0IHNvbWUgdGV4dAo=",
            "attachment": {
              "content_type": "text/plain; charset=ISO-8859-1",
              "language": "en",
              "content": "this is\njust some text",
              "content_length": 24
            }
          },
          {
            "filename": "test.txt",
            "data": "VGhpcyBpcyBhIHRlc3QK",
            "attachment": {
              "content_type": "text/plain; charset=ISO-8859-1",
              "language": "en",
              "content": "This is a test",
              "content_length": 16
            }
          }
        ]
      }
    }
 ...
}
```

SQL插件

7.4.2、7.10.2版本中我们提供Open Distro for Elasticsearch SQL插件，允许用户使用SQL而不是Elasticsearch查询域特定语言（DSL）编写查询语句。

基本操作:

要使用该功能，需要将请求发送到_opendistro/_sqlURI。用户可以使用请求参数或请求正文（推荐）。

```
GET https://<host>:<port>/_opendistro/_sql?sql=select * from my-index limit 50
```

```
POST https://<host>:<port>/_opendistro/_sql
{
 "query": "SELECT * FROM my-index LIMIT 50"
}
```

还可以使用curl命令：

```
curl -XPOST  -u username:password -k -d '{"query": "SELECT * FROM my-index LIMIT 10"}' -H 'Content-Type: application/json'
```

默认情况下，查询返回JSON，但您也可以选择CSV格式返回数据，需要对format参数进行设置：

```
POST _opendistro/_sql?format=csv
{
  "query": "SELECT * FROM my-index LIMIT 50"
}
```

CSV格式返回数据时，每行对应一个文档，每列对应一个字段。

```
data
2
1
3
```

format也支持json参数（格式与默认方式不同），如下所示：

```
POST _opendistro/_sql?format=json
{
    "query":"select * from my-index limit 10"
}
响应：
{
  "total": 3,
  "size": 3,
  "columns": [
    {
      "name": "data",
      "type": "long"
    }
  ],
  "rows": [
    [
      2
    ],
    [
      1
    ],
    [
      3
    ]
  ],
  "status": 200
}
```

支持操作 支持的SQL操作包括声明、条件、聚合函数、Include和Exclude、常用函数、连接join和展示等操作。

- 声明statements

Statement	Example
Select	SELECT * FROM my-index
Delete	DELETE FROM my-index WHERE _id=1
Where	SELECT * FROM my-index WHERE ['field']='value'
Order by	SELECT * FROM my-index ORDER BY _id asc
Group by	SELECT * FROM my-index GROUP BY range(age, 20,30,39)
Limit	SELECT * FROM my-index LIMIT 50 (default is 200)
Union	SELECT *FROM my-index1 UNION SELECT* FROM my-index2
Minus	SELECT *FROM my-index1 MINUS SELECT* FROM my-index2

说明：与任何复杂查询一样，大型UNION和MINUS语句可能会使集群资源紧张甚至崩溃。

- 条件Conditions

Condition	Example
Like	SELECT * FROM my-index WHERE name LIKE 'j%'
And	SELECT * FROM my-index WHERE name LIKE 'j%' AND age > 21
Or	SELECT * FROM my-index WHERE name LIKE 'j%' OR age > 21
Count distinct	SELECT count(distinct age) FROM my-index
In	SELECT * FROM my-index WHERE name IN ('alejandro', 'carolina')
Not	SELECT * FROM my-index WHERE name NOT IN ('jane')
Between	SELECT * FROM my-index WHERE age BETWEEN 20 AND 30
Aliases	SELECT avg(age) AS Average_Age FROM my-index
Date	SELECT * FROM my-index WHERE birthday='1990-11-15'
Null	SELECT * FROM my-index WHERE name IS NULL

- 聚合函数Aggregation

Aggregation	Example
avg()	SELECT avg(age) FROM my-index
count()	SELECT count(age) FROM my-index
max()	SELECT max(age) AS Highest_Age FROM my-index
min()	SELECT min(age) AS Lowest_Age FROM my-index
sum()	SELECT sum(age) AS Age_Sum FROM my-index

- Include和Exclude字段

Pattern	Example
include()	SELECT include('a*'), exclude('age') FROM my-index
exclude()	SELECT exclude('*name') FROM my-index

- 函数Functions

Function	Example
floor	SELECT floor(number) AS Rounded_Down FROM my-index
trim	SELECT trim(name) FROM my-index
log	SELECT log(number) FROM my-index
log10	SELECT log10(number) FROM my-index
substring	SELECT substring(name, 2,5) FROM my-index
round	SELECT round(number) FROM my-index
sqrt	SELECT sqrt(number) FROM my-index
concat_ws	SELECT concat_ws(' ', age, height) AS combined FROM my-index
/	SELECT number / 100 FROM my-index
%	SELECT number % 100 FROM my-index
date_format	SELECT date_format(date, 'Y') FROM my-in

说明：

1.字符串函数使用：

必须在文档映射中启用fielddata，字符串函数（例如：substring、concat_ws、trim）才能正常工作。

2.date_format使用说明：

date_format函数只能针对date类型的字段使用，下面是一个完整的示例：

```
  1.创建包含date类型字段的索引

PUT my_index
{
  "mappings": {
    "properties": {
      "date": {
        "type": "date"
      }
    }
  }
}
```

```
  2.写入数据
PUT my_index/_doc/2
{ "date": "2022-10-26T10:46:30Z" }

3.函数使用
POST /_opendistro/_sql?format=json
{
  "query": "select DATE_FORMAT(date, 'Y-M-d') from my_index"
}
```

- 连接操作Joins

Join	Example
Inner join	SELECT s.firstname, s.lastname, s.gender, sc.name FROM student s JOIN school sc ON sc.name = s.school_name WHERE s.age > 20
Left outer	joinSELECT s.firstname, s.lastname, s.gender, sc.name FROM student s LEFT JOIN school sc ON sc.name = s.school_name
Cross join	SELECT s.firstname, s.lastname, s.gender, sc.name FROM student s CROSS JOIN school sc

- 展示Show

展示show操作与索引模式匹配的索引和映射。您可以使用*或%使用通配符。

Show	Example
Show tables like	SHOW TABLES LIKE logs-*
Show tables	SHOW TABLES

- 连接操作Joins

支持inner joins, left outer joins,和cross joins。Join操作有许多约束：

您只能加入两个参数。

您必须为索引使用别名（例如people p）。

在ON子句中，您只能使用AND条件。

在WHERE语句中，不要将包含多个索引的树组合在一起。例如，以下语句有效

```
WHERE (a.type1 > 3 OR a.type1 < 0) AND (b.type2 > 4 OR b.type2 < -1)
```

以下声明无效：

```
WHERE (a.type1 > 3 OR b.type2 < 0) AND (a.type1 > 4 OR b.type2 < -1)
```

您不能使用GROUP BY或ORDER BY来获得结果。

LIMIT和OFFSET不支持一起使用（例如LIMIT 25 OFFSET 25）。

- JDBC驱动

Java数据库连接（JDBC）驱动程序允许您将Open Distro for Elasticsearch与您的商业智能（BI）应用程序集成。

有关下载和使用JAR文件的信息，请参阅GitHub仓库。

- 注意：

Sql插件在使用时，需要对线上使用的语句，需进行充分完整的测试。

数据迁移

用户在接入百度智能云Elasticsearch服务时，如需要百度智能云Elasticsearch服务间进行数据迁移或自建Elasticsearch服务数据迁移至百度智能云 Elasticsearch，可以根据自己的业务需求选择合适的迁移方案。本文介绍各迁移方案适用的场景，帮助您根据业务选择合适的场景进行迁移。

- BOS快照迁移

- 在线reindex
- Logstash
- 通过Spark迁移数据
- HDFS快照迁移

BOS快照迁移

snapshot api 是Elasticsearch用于对数据进行备份和恢复的一组 api 接口，可以通过 snapshot api 进行跨集群的数据迁移，原理就是从源 Elasticsearch 集群创建数据快照，然后在目标 Elasticsearch 集群中进行恢复。

百度智能云Elasticsearch基于百度存储BOS(Baidu Object Storage)，实现了snapshot api，能够帮助您在本集群内和多集群间进行快照备份与恢复。

具体使用方式参照基于BOS的快照与恢复。

适用场景

- 源端数据量较大（GB、TB、PB级别）的场景
- 恢复数据过程中需要关闭目的集群index
- 源Elasticsearch集群和目的集群Elasticsearch版本符合快照备份与恢复场景的匹配

Snapshot version → new Cluster version	2.x	5.x	6.x	7.x	8.x
1.x →	✔				
2.x →	✔	✔			
5.x →		✔	✔		
6.x →			✔	✔	
7.x →				✔	✔

使用方式

1. 两个集群分别创建基于BOS的仓库

源Elasticsearch集群需要添加BOS信息来创建仓库。创建命令如下：

```
POST /_snapshot/es_repo
{
  "type": "bos",
  "settings": {
    "access_key": "your access_key",
    "secret_key": "your secret_key",
    "endpoint": "s3.bj.bcebos.com",
    "bucket": "es-repo",
    "base_path": "test_base_path"
  }
}
```

相关参数意义：

参数	作用
type	仓库的类型，在这里您应该填`bos`
access_key	百度智能云的access_key，可以在百度智能云的console中看到
secret_key	百度智能云的secret_key，可以在百度智能云的console中看到
endpoint	BOS对应各个region的服务域名
bucket	BOS的bucket，务必保证对应的用户身份有读写bucket权限
base_path	仓库的起始位置，默认为根目录

BOS对应各个region的服务域名：

区域	访问Endpoint
北京	s3.bj.bcebos.com
保定	s3.bd.bcebos.com
苏州	s3.su.bcebos.com
广州	s3.gz.bcebos.com
香港	s3.hkg.bcebos.com
金融云武汉专区	s3.fwh.bcebos.com

在目的Elasticsearch集群创建仓库时，除了添加BOS信息外，还需要添加*readonly*参数，以保证只有一个集群可以对仓库进行操作。创建命令如下：

```
POST /_snapshot/es_repo
{
    "type": "bos",
    "settings": {
        "access_key": "your access_key",
        "secret_key": "your secret_key",
        "endpoint": "s3.bj.bcebos.com",
        "bucket": "es-repo",
        "base_path": "test_base_path",
        "readonly": true
    }
}
```

2. 在源Elasticsearch集群创建数据的快照

```
PUT /_snapshot/es_repo/snapshot_2020_01_01?wait_for_completion=true
{
  "indices": "index_1,index_2",
  "ignore_unavailable": true,
  "include_global_state": false
}
```

3. 在目的Elasticsearch集群恢复数据快照

```
POST /_snapshot/es_repo/snapshot_2020_01_01/_restore
{
  "indices": "index_1,index_2",
  "include_global_state": false
}
```

具体参数详解参照基于BOS的快照备份恢复。

注意事项

- 需要开通BOS服务并且创建bucket。
- 并且源集群和目的集群能够访问BOS网络。
- 源集群进行快照时，同时只能有一个快照运行，且有删除快照的操作时，不能进行快照。
- 目的集群创建仓库时，需要添加*readonly*参数，以保证只有一个集群可以对仓库进行写操作。
- 目的集群恢复数据时，恢复的index可以不存在，如果存在必须是`closed`状态。
- 对源集群和目的集群的版本匹配有严格要求，具体要求参见上文表格。
- 使用BOS跨region迁移数据时，可能会产生外网流量费用。
- 对索引做快照的时候，对索引的变更操作不会在快照中出现，用户应该避免这种情况。如果必须变更数据，需要在迁移完后在新的集群进行回放变更操作。同理，其他迁移方式也要注意这点。

在线reindex

reindex from a remote cluster api是Elasticsearch提供的一个 api接口，可以把数据从源Elasticsearch集群导入到当前Elasticsearch集群，实现数据的迁移。原理是从源 Elasticsearch 集群中查询数据，然后写入到目标 Elasticsearch 集群中。

适用场景

- 在线迁移数据
- 对迁移速度要求不高
- 可以对源端集群数据进行查询

使用方式

- 在目的集群的elasticsearch.yml配置reindex白名单

```
reindex.remote.whitelist: {IP}:{port}
```

- 在目的集群创建index（如果不需要特殊处理分片数和mapping等，可以省略这一步）
- 做reindex操作

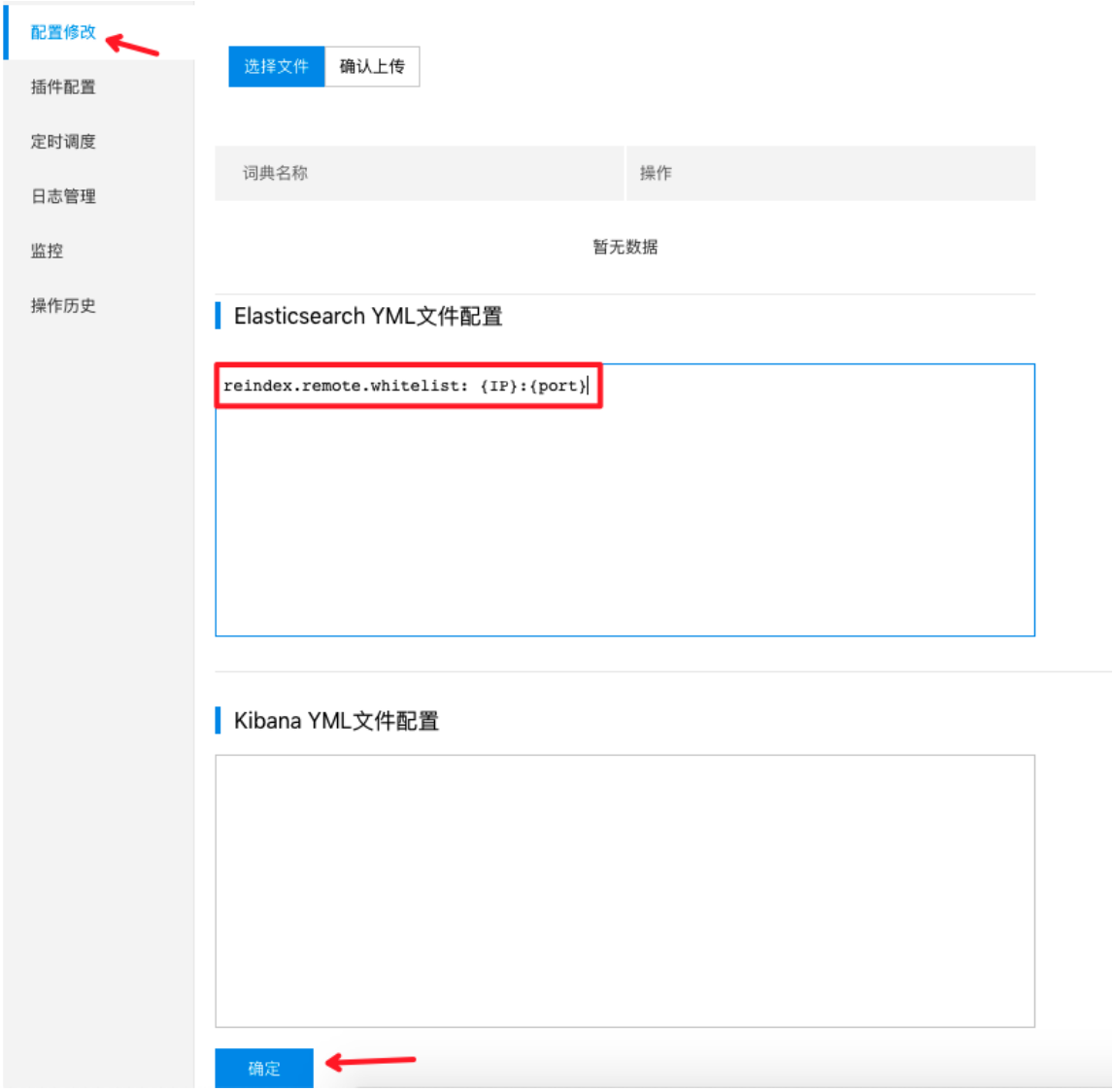
```
POST _reindex
{
  "source": {
    "remote": {
      "host": "http://{oldhost}:{oldport}",
      "username": "{username}",
      "password": "{password}"
    },
    "index": "source_index",
    "query": {
      "match_all": {}
    }
  },
  "dest": {
    "index": "dest_index"
  }
}
```

详细使用方式参照reindex from a remote cluster api。

注意事项

- 目的集群需要连通源集群的网络。
- 需要重启目的集群配置reindex白名单。

> 白名单修改可以在百度云页面【配置修改】进行修改，
> 注意：
> 1、修改配置后，需要重启集群才生效。
> 2、配置格式为yml格式。注意，在【冒号】后【配置值】前要有一个“空格”。



- 迁移数据时，最好不要对索引有变更操作；如果必须变更数据，需要在迁移完后在新的集群进行回放变更操作。

Logstash

适用场景

- 在线迁移数据
- 可以对源端集群数据进行查询
- 请注意 Logstash 版本的版本是否与Elastic相匹配，建议与 Elasticsearch 版本保持一致。版本兼容性说明请参见Elasticsearch系列产品兼容性。

使用方式

- 安装部署Logstash与Java8以上的jdk环境。

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-7.4.2.tar.gz
tar xvf logstash-7.4.2.tar.gz
```

- 根据数据源类型自定义配置文件`*.conf`，配置文件内容如下：

```
input {
  elasticsearch {
      hosts => ["http://{oldhost}:{oldport}"]
      index => "*"
      docinfo => true
  }
}
output {
  elasticsearch {
      hosts => ["http://{newhost}:{newport}"]
      index => "%{[@metadata][_index]}"
  }
}
```

- 执行Logstash。

```
nohup ./bin/logstash -f ~/*.conf 2>&1 >/dev/null &
```

上述配置文件将源Elasticsearch集群的所有索引同步到目标集群中，也可以设置只同步指定的索引，Logstash 的更多功能可查阅 Logstash 官方文档。

注意事项

- 需要准备Logstash环境并连通Elasticsearch集群的网络。

通过Spark迁移数据

Spark支持从一个Elasticsearch集群中读取数据然后写入到另一个Elasticsearch集群。

使用方式

Spark代码参考如下：

```
import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.elasticsearch.spark.rdd.api.java.JavaEsSpark;

import java.util.HashMap;
import java.util.Map;

import static org.elasticsearch.spark.rdd.api.java.JavaEsSpark.esRDD;

public class MigrateESToES {

    public static void main(String[] args) {

        SparkConf conf = new SparkConf();

        conf.setAppName("Baidu-Cloud-Migrate-ES-Example");
        conf.set("es.index.auto.create", "true");
```

```
        SparkContext sc = new SparkContext();

        // source cluster configuration
        Map<String, String> inputCfg = new HashMap<>();
        // node ip:port
        inputCfg.put("es.nodes", "{Source Elasticseach Host}:8200");
        // 要迁移的索引和文档类型，例如test_size索引，文档类型_doc
        inputCfg.put("es.resource", "test_size/_doc");
        // 如果没有开启认证的话，这两个选项可以不写
        // username
        // inputCfg.put("es.net.http.auth.user", "superuser");
        // password
        // inputCfg.put("es.net.http.auth.pass", "xxxx");
        // es_version，条件可以的话建议写上
        inputCfg.put("es.internal.es.version", "7.4.2");
        // 读取_version 字段
        // inputCfg.put("es.read.metadata.version", "true");
        // 读取_matedata 字段
        // inputCfg.put("es.read.metadata", "true");

        // dst cluster configuration
        Map<String, String> outputCfg = new HashMap<>();
        outputCfg.put("es.nodes", "{Dst Cluster}:8200");
        outputCfg.put("es.net.http.auth.user", "root");
        outputCfg.put("es.net.http.auth.pass", "root");
        outputCfg.put("es.internal.es.version", "7.4.2");
        // 目标索引和文档类型
        outputCfg.put("es.resource", "test_size/_doc");

        // 指定写入version type 为外部指定
        // outputCfg.put("es.mapping.version.type", "external");
        // 指定 index id
        // outputCfg.put("es.mapping.id", "_metadata._id");
        // 指定 routing id
        // outputCfg.put("es.mapping.routing", "_metadata._id");
        // 指定 doc version
        // outputCfg.put("es.mapping.version", "_metadata._version");
        // 关闭refresh
        outputCfg.put("es.batch.write.refresh", "false");
        // 单次写15mb
        outputCfg.put("es.batch.size.bytes", "15mb");
        //size (in entries) for batch writes using Elasticsearch bulk API
        outputCfg.put("es.batch.size.entries", "5000");
        // 异常不处理，打log，避免conflict 导致task failed
        outputCfg.put("es.write.rest.error.handlers", "log");
        // 异常日志前缀
        outputCfg.put("es.write.rest.error.handler.log.logger.name", "es_error_handler");
        // outputCfg.put("es.mapping.exclude", "_metadata");

        // create spark context
        JavaSparkContext jsc = new JavaSparkContext(sc);

        JavaRDD<Map<String, Object>> sourceRDD = esRDD(jsc, inputCfg).values();

        JavaEsSpark.saveToEs(sourceRDD, outputCfg);
        sc.stop();
    }
}
```

注意事项

- 需要准备Spark环境并连通Elasticsearch集群的网络。
- 需要自行实现Spark业务逻辑。

HDFS快照迁移

使用方式

BES已预装了repository-hdfs插件。您可以通过HDFS进行快照备份并恢复，使用方式与基于BOS的快照备份与恢复相类似，只是中转媒介换成了HDFS。

创建快照仓库的命令示例：

```
PUT _snapshot/my_hdfs_repository
{
  "type": "hdfs",
  "settings": {
    "uri": "hdfs://192.168.1.101:8020",
    "path": "/tmp/hadoop/es_repo"
  }
}
```

配置详情可参考ES官网介绍repository-hdfs-config。

注意事项

- 需要准备资源足够的HDFS环境并连通网络。
- BES已预装了repository-hdfs插件。
- 目的集群恢复数据时，需要关闭相应index。
- 对源集群和目的集群的版本匹配有严格要求，具体要求参见上文表格。

账户管理

多用户访问控制

介绍

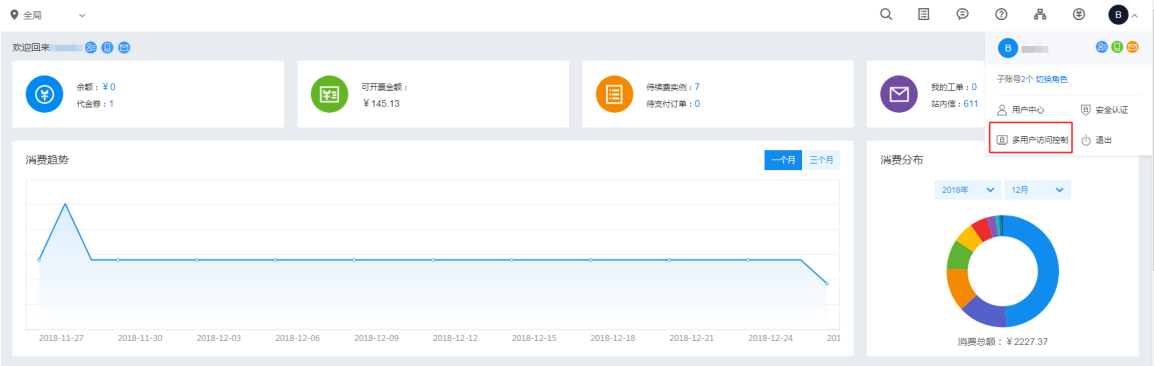
多用户访问控制，主要用于帮助用户管理云账户下资源的访问权限，适用于企业内的不同角色，可以对不同的工作人员赋予使用产品的不同权限，当您的企业存在多用户协同操作资源时，推荐您使用多用户访问控制。

适用于下列使用场景：

- 中大型企业客户：对公司内多个员工授权管理；
- 偏技术型vendor或SAAS的平台商：对代理客户进行资源和权限管理；
- 中小开发者或小企业：添加项目成员或协作者，进行资源管理。

创建用户

1. 主账号用户登录后在控制台选择“多用户访问控制”进入用户管理页面。



2. 在左侧导航栏点击“用户管理”，在“子用户管理列表”页，点击“新建用户”。

3. 在弹出的“新建用户”对话框中，完成填写“用户名”和确认，返回“子用户管理列表”区可以查看到刚刚创建的子用户。

配置策略

BES支持系统策略和用户自定义策略两种，分别实现BES的产品级权限和实例级权限控制。

- 系统策略：百度智能云系统为管理资源而预定义的权限集，这类策略可直接为子用户授权，用户只能使用而不能修改。
- 自定义策略：由用户自己创建，更细化的管理资源的权限集，可以针对单个实例配置权限，更加灵活的满足账户对不同用户的差异化权限管理。

系统策略

系统策略包含只读、运维和管理操作权限，权限范围详细如下：

策略名称	权限说明	权限范围
BESReadAccessPolicy	只读BES的权限	只读权限包括查看集群详情的权限。
BESOperateAccessPolicy	运维操作BES的权限	运维权限包含“只读”权限，还具有启动、停止、修改密码、绑定和解绑EIP等权限的功能。
BESFullControlAccessPolicy	完全控制管理BES的权限	管理权限包含“运维”权限，还具有创建、删除、集群伸缩等功能。

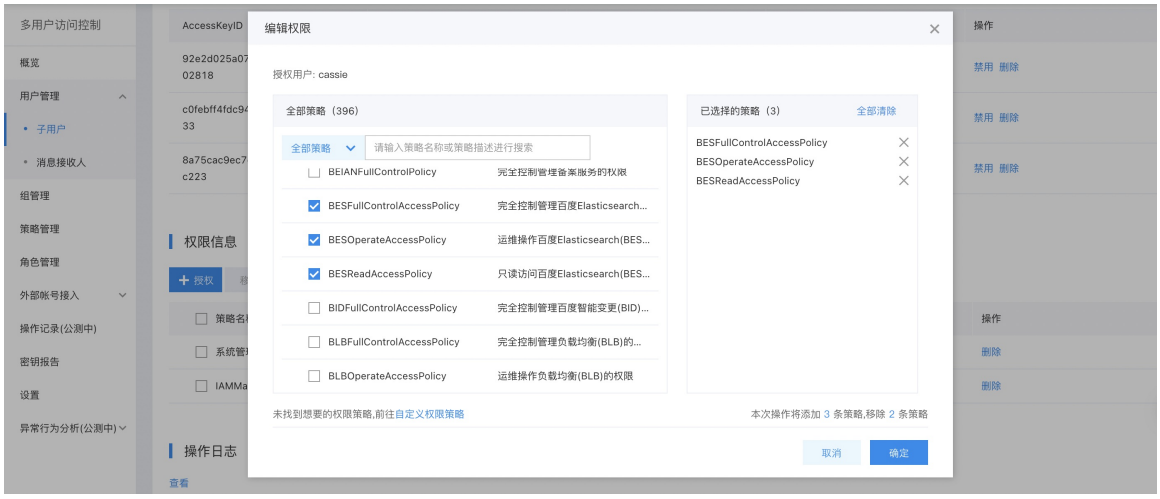
自定义策略

自定义策略是从单个集群维度进行授权，与系统策略不同，只对选定的集群生效。

子用户先通过左侧导航栏进入【策略管理】，然后点击“创建策略”，用户填写策略名称并选择服务类型为“百度Elasticsearch BES”，其中策略生成方式默认为策略生成器，不需要修改。

用户授权

在“用户管理->子用户管理列表页”的对应子用户的“操作”列选择“添加权限”，并为用户选择系统权限或自定义策略进行授权。



> **说明：**如果在不修改已有策略规则的情况下修改某子用户的权限，只能通过删除已有的策略并添加新的策略来实现，不能取消勾选已经添加过的策略权限。

相关服务授权

为使子用户能够正常使用BES服务，还需要为其授予百度智能云内其他相关服务的权限，相关服务的权限策略及影响范围如下：

相关服务策略名称	描述	影响范围
FCFullControlPolicy	完全控制订单服务权限	BES管理、运维权限
VPCFullControlAccessPolicy	完全控制VPC服务权限	BES管理权限
SUBNETFullControlAccessPolicy	完全控制子网服务权限	BES管理权限
EIPFullControlAccessPolicy	完全控制管理EIP的权限	BES管理权限
SECURITYGROUPFullControlAccessPolicy	完全控制安全组的权限	BES管理权限

子用户登录

主账号完成对子用户的授权后，可以将链接发送给子用户；子用户可以通过IAM用户登录链接登录主账号的管理控制台，根据

被授权的策略对主账户资源进行操作和查看。

其他详细操作参考：多用户访问控制。

定时调度

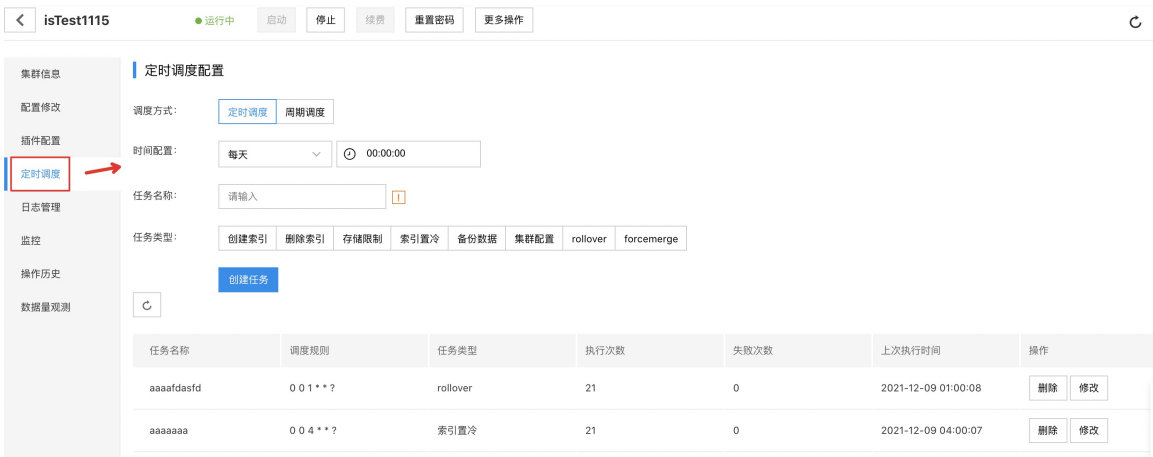
百度智能云Elasticsearch（BES）提供定时调度功能，使用此功能可实现：

- 定时创建索引
- 定时删除索引
- 对索引存储进行限制
- 对冷热数据分离的索引定时置冷
- 定时备份数据
- 定时修改集群配置
- 定时rollover
- 定时forcemerge

可以按照业务场景创建调度任务，更方便地使用和管理百度智能云Elasticsearch（BES）。

创建任务

1. 登录百度智能云Elasticsearch控制台，并点击集群名称，进入集群详情界面。
2. 在左侧导航栏，单击**定时调度**进入定时调度界面。



3. 在定时调度配置界面，选择调度方式、时间配置、任务名称、和任务类型，根据不同任务类型，配置各类任务信息。最后点击“创建任务“按钮，完成任务创建。

创建索引

定时创建索引支持用户指定索引的命名规则，以及settings、mappings等信息，并按照设定的时间策略创建索引。

用户可根据业务需求配置调度方式、任务名称、任务类型、索引名称前缀、索引名称后缀和索引信息。

使用说明

- **调度方式：**
  - 定时调度：可以配置定时调度，指定每天、每周、每月的确定时间执行任务。
  - 周期调度：可以配置周期调度，指定每间隔一段时间执行一次任务。
- **任务名称：** 用户自行填写。名称规则支持字母(a-z及A-Z)、数字(0-9)、下划线(_)或连字符(-)，长度6~24个字符。
- **任务类型：**在任务类型中选择“创建索引”。
- **索引名称前缀：**索引名称仅能由数字、小写字符、下划线（`_`）和横线（`-`）组成。
- **索引名称后缀：**可选5种时间格式，“yyyy.MM.dd.HH.mm”、“yyyy.MM.dd”、“yyyy-MM-dd-HH-mm”、“yyyy-MM-dd“和”yyyy-MM“。系统会将索引名称前缀和索引名称后缀相结合成索引名称，并按指定时间创建索引。
- **索引信息：**索引信息为JSON格式，可以设置索引的settings和mappings，详见创建索引。非必填。

建议在创建索引前先创建模板。

例如：用户需求每周一零点创建一个新索引，命名规则为`test_index_2020.12.01`，则可以配置一个**创建索引**任务。配置方式如下图：

删除索引

用户可以使用通配符等方式匹配索引，并按照策略定期删除索引。

使用说明

- **调度方式：**
  - 定时调度：可以配置定时调度，指定每天、每周、每月的确定时间执行任务。
  - 周期调度：可以配置周期调度，指定每间隔一段时间执行一次任务。
- **任务名称：** 用户自行填写。名称规则支持字母(a-z及A-Z)、数字(0-9)、下划线(_)或连字符(-)，长度6~24个字符。
- **任务类型：**在任务类型中选择“删除索引”。
- **索引匹配规则：**可以使用通配符、索引名称匹配索引或使用减号排除索引。
  - 使用通配符（*）来匹配多个索引，如`test*`代表匹配以`test`为前缀的所有索引；
  - 直接使用索引名称匹配，如`test`代表匹配名称为`test`的索引；也可以输入多个索引名称用逗号分隔，如`test1,test2`；
  - 可以使用减号（-）排除索引，例如，`test*,-test`，代表匹配以`test`为前缀且除去名称为`test`的所有索引。
- **索引最小年龄：**执行定时任务的时间减去索引创建时间称为索引年龄，只有年龄超过索引最小年龄的索引才会执行定时任务。最小年龄可以配置为0，代表符合索引匹配规则的索引都可以执行定时任务。

例如：用户需要在每天凌晨0点删除7天前创建的，并且名称符合 `test*` 的索引，则可以配置一个**创建索引**任务。配置方式如下图：

注意事项

- 索引删除**不可恢复**，请提前确定索引数据可删除，再进行任务配置。

存储限制

用户可以使用通配符等方式匹配索引，定期检查索引占据的存储空间是否超过设定值，如果超过设定值，则将索引置为只读模式。

使用说明

- **调度方式：**
  - 定时调度：可配置定时调度，指定每天、每周、每月的确定时间执行任务。
  - 周期调度：可配置周期调度，指定每间隔一段时间执行一次任务。
- **任务名称：** 用户自行填写。名称规则支持字母(a-z及A-Z)、数字(0-9)、下划线(_)或连字符(-)，长度6~24个字符。

- **任务类型：**在任务类型中选择“存储限制”。
- **索引匹配规则：**可以使用通配符、索引名称匹配索引或使用减号排除索引。
  - 使用通配符（*）来匹配多个索引，如`test*`代表匹配以`test`为前缀的所有索引；
  - 直接使用索引名称匹配，如`test`代表匹配名称为`test`的索引；也可以输入多个索引名称用逗号分隔，如`test1,test2`；
  - 可以使用减号（-）排除索引，例如，`test*,-test`，代表匹配以`test`为前缀且除去名称为`test`的所有索引。
- **索引最大存储量：**索引允许的最大存储量，最小为1，只支持整数。当索引存储数据量超过此处设定值，会将索引设置为只读。

例如：用户需要名称符合`test*`的索引均不允许超过100GB，则可以配置一个**存储限制**任务。配置方式如下图：

如图代表，每十分钟对集群中所有符合`test*`的索引进行检测，如果存储超过100GB，则置为只读模式。

注意事项

- 此处索引占据的存储空间指的是主分片占据的存储空间，您可以通过以下接口查询**pri.store.size**字段来查看主分片大小。

```
get /_cat/indices?v
```

```
health status index     uuid                   pri rep docs.count docs.deleted store.size pri.store.size
green  open   test_size Dnkj89rzQDizlL6eOjAuEQ   1   1         99            0     64.4kb         32.2kb
```

- 用户可以将调度周期设置得更短，如“每隔1分钟执行一次”，这样可以更精细地对索引存储进行限制。
- 只读模式可以通过以下API恢复。注意：若索引依旧超过限制的最大存储量，即使恢复后，依旧会再次被置为只读模式。

```
PUT {index_name}/_settings
{
   "index.blocks.write":null
}
```

索引置冷

百度智能云ES支持通过定时调度功能将索引制冷。用户可以使用通配符等方式匹配索引，定时置为冷数据索引。可选将索引迁移到冷数据节点或对象存储BOS。

使用说明

- **调度方式：**
  - 定时调度：可配置定时调度，指定每天、每周、每月的确定时间执行任务。
  - 周期调度：可配置周期调度，指定每间隔一段时间执行一次任务。
- **任务名称：** 用户自行填写。名称规则支持字母(a-z及A-Z)、数字(0-9)、下划线(_)或连字符(-)，长度6~24个字符。
- **任务类型：**在任务类型中选择“索引置冷”。
- **索引存储位置：**可选冷数据节点或对象存储BOS，用户可根据自身集群进行选择。索引存储位置不影响调度任务其他配置。
  - 冷数据节点：如果集群中包含冷数据节点，可选择将索引变为只读索引且数据存储在冷数据节点。
  - 对象存储BOS： 如果集群支持并开启了基于BOS的冷热数据分离功能，可将索引变为只读索引且数据上传到BOS。非冷热数据分离的index则不受影响。
- **索引匹配规则：**可以使用通配符、索引名称匹配索引或使用减号排除索引。
  - 使用通配符（*）来匹配多个索引，如`test*`代表匹配以`test`为前缀的所有索引；
  - 直接使用索引名称匹配，如`test`代表匹配名称为`test`的索引；也可以输入多个索引名称用逗号分隔，如`test1,test2`；
  - 可以使用减号（-）排除索引，例如，`test*,-test`，代表匹配以`test`为前缀且除去名称为`test`的所有索引。
- **索引最小年龄：**执行定时任务的时间减去索引创建时间称为索引年龄，只有年龄超过索引最小年龄的索引才会执行定时任务。最小年龄可以配置为0，代表符合索引匹配规则的索引都可以执行定时任务。

例如：用户需要将创建超过1周且名称符合`test*`的冷热数据分离索引置为冷索引，存放在冷数据节点，每周一0点执行。则可以配置一个**索引置冷**任务。配置方式如下图：

再例如：用户需要将创建超过3天且名称符合`test*`的冷热数据分离索引置为冷索引，存放在对象存储BOS，每天0点执行。则可以配置一个**索引置冷**任务。配置方式如下图：

注意事项

- 索引存储位置为对象存储BOS的功能只支持基于BOS冷热数据分离的集群使用，详见基于BOS的冷热数据分离。
- 索引存储位置为冷数据节点的功能只有在创建了冷数据节点的集群才能使用。相见基于节点磁盘介质的冷热数据分离。

备份数据

用户可以通过基于BOS的快照与恢复来备份数据。在创建BOS仓库后，用户可以设置需要备份的索引和其他快照策略，系统会根据配置，定时执行快照备份数据，保证数据的安全性。

使用方式

- **调度方式：**
  - 定时调度：可配置定时调度，指定每天、每周、每月的确定时间执行任务。
  - 周期调度：可配置周期调度，指定每间隔一段时间执行一次任务。

快照任务的最短间隔是1h，若配置较小的时间间隔，也会以1h的时间间隔备份数据。

- **任务名称：** 用户自行填写。名称规则支持字母(a-z及A-Z)、数字(0-9)、下划线(_)或连字符(-)，长度6~24个字符。
- **任务类型：**在任务类型中选择“备份数据”。
- **仓库名称：**repository名称，snapshot使用的仓库, **快照仓库必须在备份数据前创建。** 创建方式参见基于BOS的快照与恢复。
- **快照名称：**snapshot名称，在执行snapshot时，该名称作为前缀执行。具体snapshot名称为：[snapshot_time_uuid]。
- **索引列表：**需要被包含在snapshot中的index列表，默认为备份全部index；格式为index1,index2,index3
- **快照过期时间：**快照过期时间，默认为90天。快照过期之后将会被删除。

例如：用户需要每周一将集群中`my_index1,my_index2,my_index3`等几个索引的数据备份到BOS，则可以配置一个**备份数据**任务。配置方式如下图：

集群配置

用户可以根据需要定期修改集群配置。

使用说明

- **调度方式：**
  - 定时调度：可配置定时调度，指定每天、每周、每月的确定时间执行任务。
  - 周期调度：可配置周期调度，指定每间隔一段时间执行一次任务。
- **任务名称：** 用户自行填写。名称规则支持字母(a-z及A-Z)、数字(0-9)、下划线(_)或连字符(-)，长度6~24个字符。
- **任务类型：**在任务类型中选择“集群配置”。
- **settings：**setting支持JSON格式配置。

例如：用户需要每天21：00关闭配置`action.destructive_requires_name`，以便能够使用通配符删除索引，而在每天的23：00完成删除，并开启`action.destructive_requires_name`，禁止使用通配符删除索引，以防止误操作导致数据丢失。则配置方式如下图：

rollover

rollover API可以根据索引的大小、文档数、创建时长等规则翻转到新索引。当rollover触发后，将创建新索引，并将写入别名（write alias）指向新索引，后续写入都将写入新索引。用户可以配置索引的大小、文档数、创建时长等策略定时翻转索引。

使用方式

- **调度方式：**

  - 定时调度：可配置定时调度，指定每天、每周、每月的确定时间执行任务。
  - 周期调度：可配置周期调度，指定每间隔一段时间执行一次任务。
- **任务名称：** 用户自行填写。名称规则支持字母(a-z及A-Z)、数字(0-9)、下划线(_)或连字符(-)，长度6~24个字符。
- **任务类型：**在任务类型中选择“rollover”。
- **索引别名：**索引别名可由数字，大小写字符和（_,-,+）组成，不能以（_,-,+）开头，可以使用通配符、索引名称匹配索引别名或使用减号排除索引别名。
- **max_size：**当索引主分片存储达到这个参数设定的量级，即可触发rollover，设置为0表示此项不生效。
- **max_age：**当索引在创建后达到这个参数设定的年龄，即可触发rollover，设置为0表示此项不生效。
- **max_docs：**当索引文档数达到这个参数设定的数目，即可触发rollover，设置为0表示此项不生效。

用户同时设置了max_size、max_age和max_docs，只要其中一项达到设定值就会触发rollover。

例如：用户需要每周一对别名`test_a`的索引进行rollover，则可以配置一个**rollover**任务（rollover使用参见rollover API）。

配置方式如下图：

forcemerge

用户可以使用通配符等方式匹配索引，并按照策略定期合并segment。forcemerge一般适用于**不再更新或只有定时批量更新**的索引，且执行频率不易过高，建议在没有写入的时间段进行。

使用方式

- **调度方式：**
  - 定时调度：可配置定时调度，指定每天、每周、每月的确定时间执行任务。
  - 周期调度：可配置周期调度，指定每间隔一段时间执行一次任务。
- **任务名称：** 用户自行填写。名称规则支持字母(a-z及A-Z)、数字(0-9)、下划线(_)或连字符(-)，长度6~24个字符。

- **任务类型：**在任务类型中选择“forcemerge”。
- **索引匹配规则：**可以使用通配符、索引名称匹配索引或使用减号排除索引。
- **最大Segment数量：**forcemerge合并segment的最大数量。
- **索引最小年龄：**定时任务的执行时间减去索引创建时间称为索引年龄，只有年龄超过索引最小年龄的索引才会执行定时任务。最小年龄可以配置为0，代表符合索引匹配规则的索引都可以执行定时任务。

例如：用户需要每天凌晨0点对创建超过10天且名称符合`test*`的索引进行segment合并，且将segment合并为10个，则可以配置一个**forcemerge**任务。配置方式如下图：

查看任务列表

在定时调度配置界面下方，用户可以看到已经创建的任务列表，以及定时任务的执行次数和失败次数。

任务名称	调度规则	任务类型	执行次数	失败次数	上次执行时间	操作
test_index_	0 0 12 * * ?	创建索引	3	0	2021-12-16 12:00:09	删除 修改
delete index	0 0 0 * * ?	删除索引	0	0		删除 修改
index limit	0 0 0 * * ?	存储限制	0	0		删除 修改
cold_index	0 0 18 * * ?	索引置冷	2	0	2021-12-15 18:00:10	删除 修改
limitindex	0 0 0 * * ?	存储限制	0	0		删除 修改

修改任务

点击任务列表操作列中修改按钮，在右侧弹出页面中可修改对应定时任务。

删除任务

点击任务列表中的删除按钮，确认后即可删除对应定时任务。

账号使用说明

百度智能云Elasticsearch(BES)账号、监控服务Grafana账号、百度云账号及其子账号、以及百度云AK/SK的使用说明如下所示。

百度智能云Elasticsearch账号

用于访问BES集群，以及Kibana服务。

说明：

1.BES管理员账号(superuser)可以直接访问BES集群、Kibana服务，其密码在创建时指定，也可以对密码进行重置，详见：集群创建、密码重置；

2.非管理员账号访问Kibana服务，需要添加Kibana服务相关索引的读写权限。下面是创建一个有test索引读写权限，并可以访问Kibana服务的普通账号例子（BES账号的管理与权限配置的内容，详见权限管理）；

```
 curl -u username:password -XPOST -H "Content-Type: application/json" 'host:port/_user/create' -d '
{
 "username" : "username",
 "password" : "password",
 "get_path" : ["/"],
 "put_path" : ["/"],
 "post_path" : ["/"],
 "del_path" : ["/"],
 "head_path" : ["/"],
 "read_index": ["test",".kibana"],
 "write_index": ["test",".kibana"]
}'
```

监控服务Grafana账号

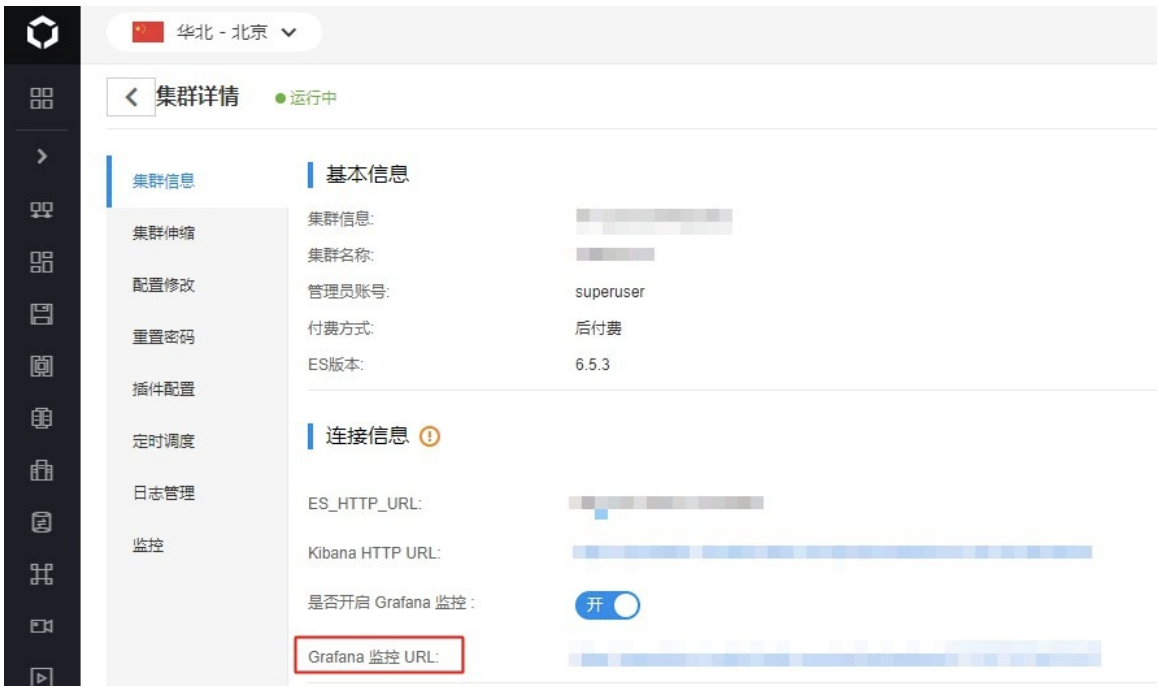
用于访问Grafana服务，查看监控数据。

说明：

1.Grafana服务在创建时，默认使用BES管理员账号(superuser)和密码进行初始化，后续可以单独修改，修改方式参考下面的说明。

> 注意：2024年11月之前创建的集群使用用户名：admin，默认密码：admin 访问 Grafana 服务。

2.Grafana的ip地址与Kibana的ip地址相同，端口为8082；在百度云ES Console的集群详情页，可以查看Grafana监控URL，如下图所示：



3.修改Grafana账号密码的命令，如下所示：

```
curl -X PUT -H "Content-Type: application/json" http://user:oldpass@kibana_ip:8082/api/user/password -d '{
 "oldPassword": "oldpass",
 "newPassword": "newpass",
 "confirmNew": "newpass"
}'
```

以默认账号密码修改为例，具体的执行命令，如下所示：

```
curl -X PUT -H "Content-Type: application/json" http://admin:admin@kibana_ip:8082/api/user/password -d '{
 "oldPassword": "admin",
 "newPassword": "newpass",
 "confirmNew": "newpass"
}'
```

4.使用管理员账号添加新账号的命令，如下所示：

```
curl -XPOST -H "Content-Type: application/json" http://admin:admin@kibana_ip:8082/api/admin/users -d '{
 "login":"user",
 "password":"userpassword"
}'
```

其它账号说明

1.百度云账号及其子账号，通过访问百度云ES Console，对ES集群资源进行管理。百度云子账号的管理，详见：[多用户访问控制]

2.百度云AK/SK，通过Open Api的方式，对ES集群资源进行管理。具体使用详见：BES-Java-SDK、BES-Python-SDK 中的说明；

数据量观测

百度智能云Elasticsearch（BES）提供索引数据量观测功能，用户可以观察时序性数据索引的数据量走势来分析业务数据，并且可以直接判断集群是否需要扩容、变配，提高集群稳定性。

> 索引数据量观测功能是基于索引名称和时间格式共同来匹配索引，所以需要集群中索引命名是带有时间格式命名。

使用方式

1. 登录百度智能云Elasticsearch控制台，并点击集群名称，进入集群详情界面。
2. 在左侧导航栏，单击**数据量观测**进入数据量观测界面。

3. 按照业务需求填写参数后，点击**搜索**，即可查询走势图。

参数说明

索引前缀

索引前缀是用户希望匹配到的索引的共同命名前缀，如用户希望匹配索引命名为【data-2021-08-01】，索引前缀就可以填写`data`。如果不填写，则匹配所有索引。最大可以填写128个字符。

时间格式

索引数据量观测功能使用索引前缀和时间格式共同来匹配索引，如果用户希望匹配索引命名为【data-年-月-日】（如data-2021-08-01），索引前缀就可以填写`data`，时间格式可以填写`yyyy-MM-dd`；如果用户希望匹配索引命名为【data.年.月.日】（如data.2021.08.01），索引前缀就可以填写`data`，时间格式可以填写`yyyy.MM.dd`。

> **注意**：时间格式要注意匹配符号的大小写，具体含义如下表。

> 常用格式：
>
> | 符号 | 含义 |
> | --- | --- |
> | yyyy | 年份 |
> | MM | 月份 |
> | dd | 日期 |
> | HH | 24时制小时 |
> | mm | 分钟 |
> | ss | 秒 |
>
> 易错格式（**慎用**）：
>
> | 符号 | 含义 |
> | --- | --- |
> | YYYY | week-based-year，代表所在周所处的年份 |
> | DD | day in year，代表当天处于当年的第几天 |
> | hh | 12时制小时 |

时间范围

用户可以选择观测最近x天、小时、分钟、月、年的索引数据量，最大取值为1000。

> **注意**：选择的时间单位要和时间格式**匹配**，如：用户选择观测最近7天的数据量，则时间格式需要精确到天，如`yyyy-MM-dd`；用户选择观测最近7个月的数据量，则时间格式需要精确到月，如`yyyy-MM`。

多可用区部署

百度智能云Elasticesarch支持多可用区部署，提升集群的容灾能力。

适用场景

多可用区部署支持单可用区、双可用区、三可用区三种部署方式，各部署方式的适用场景如下：

- 单可用区：普通部署模式，适用于非关键任务型的工作（默认）。
- 双可用区：跨可用区容灾部署模式，适用于生产型的工作。
- 三可用区：高可用部署模式，适用于具有更高可用性要求的生产型的工作。

创建多可用区集群

登录百度智能云Elasticsearch 控制台，选择【创建集群】。

可用区与子网配置 在【网络与可用区】模块，【可用区部署数量】可以选择单可用区、双可用区或三可用区集群。并在下方选择相应的【可用区及子网】。

双可用区部署

创建双可用区集群，支持选择两个可用区或三个可用区，前两个可用区为必选可用区，第三个可用区为补充可用区，选填。

- 如果选择三个可用区，数据节点会被部署在前两个可用区，第三个可用区会用来单独部署一个专用主节点。保证当任意一个可用区出现故障时，集群依然可以选出主节点。
- 如果选择两个可用区，数据节点和专用主节点都会部署在两个可用区。当存在半数以下的专用主节点的可用区出现故障时，集群依然可以选出主节点；当存在超过半数专用主节点的可用区出现故障时，集群无法选主，集群将不可用。

> **说明：**
>
> 1.由于要开启多可用区容灾的集群，必须选择专用主节点，用户可自行选择专有主节点数量3或5个。
>
> 2.如果业务上有更高的可靠性要求，建议选择三个可用区。

三可用区部署

三可用区集群，数据节点和专用主节点都会均匀地部署在三个可用区中。在一个可用区发生不可用的情况下，不会出现超过一半的专用主节点不可用的情况，始终保持集群有超过法定的主节点选择个数，保证了集群的可靠性。

> **说明:** 在创建集群时，会校验子网。如果所选子网的IP数量不足，无法成功创建集群，请根据提示进行对应操作。

节点配置 确定好可用区及子网信息后，继续进行【节点配置】。

- 节点配置中，数据节点、冷数据节点及协调节点的个数必须为可用区个数的整数倍，例如双可用区，数据节点个数应为2、4、6...
- 由于要开启多可用区容灾的集群，必须选择专用主节点，用户可自行选择专有主节点数量3或5个。

其他参数设置与单可用区基本一致。

集群多可用区容灾原理

数据节点

> **注意：**
>
> 为了保证多可用区容灾的功能生效，用户请注意以下原则：
>
> 1.创建集群所选择的数据节点、冷数据节点及协调节点个数必须为可用区个数的整数倍，例如双可用区，数据节点个数应为2、4、6...
>
> 2.索引分片至少设置1个副本，保证集群始终有两份以上的数据。

百度智能云Elasticsearch会自动将数据节点均匀的部署在用户所选的可用区中，且数据节点可以感知其部署的可用区 ，使数据的副本平均分布到多个可用区中。

百度智能云Elasticsearch提供了负载均衡功能，通过控制台提供的 BLB 进行访问，请求会平均分配到集群的各个数据节点上，当某个节点或者某个可用区不可用时，BLB 会自动过滤问题节点，保证集群稳定性。

专用主节点 为提高Elasticsearch集群可靠性，用户在使用多可用区部署时最少要创建三个专用主节点。当一个可用区不可用时，如果集群依然有半数以上主节点参与选举，集群可正常选主。

配置说明 此处用户无需自己配置，创建时，系统会自动为多可用区实例配置与分片分配策略相关的集群配置。 例如，多可用区实例部署在广州可用区A和可用区B，则集群配置相关的参数说明如下：

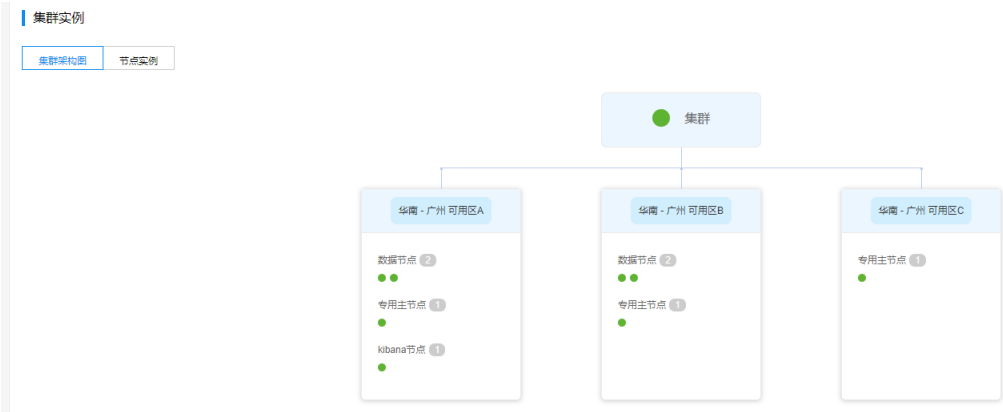
参数	说明	示例
cluster.routing.allocation.awareness.attributes	配置用于设置分配策略的节点属性，例如配置为zone_id时，各节点通过zone_id这个节点属性来确定分配策略，多可用区实例通过在启动参数中加入node.attr.zone_id来标识该节点的可用区，例如广州A可用区的节点启动参数中加入node.attr.zone_id=cn-gz-a，该参数默认为zone_id。	zone_id
cluster.routing.allocation.awareness.force.zone_id.values	配置多可用区部署强制均分shard。例如索引中有1个主分片(primary)，3个副本分片(number_of_replicas=3)，共4个副本，按照分配策略，会在可用区A和B各分配两个副本。如果指定了该参数，当cn-gz-a不可用时，该可用区的两个分片不会自动迁移到cn-gz-b。该参数默认为当前部署的所有可用区，即不允许分片自动迁移。	["cn-gz-a", "cn-gz-b"]

节点分布

用户可以通过控制台集群详情查看各个可用区的节点分布情况:

- 双可用区集群4节点部署在2个可用区情况下的节点，分布图如下

- 双可用区集群4节点部署在3个可用区情况下的节点，分布图如下



- 三可用区3节点情况下的节点，分布图如下

集群监控报警

配置报警

实例组（批量观测及批量报警）

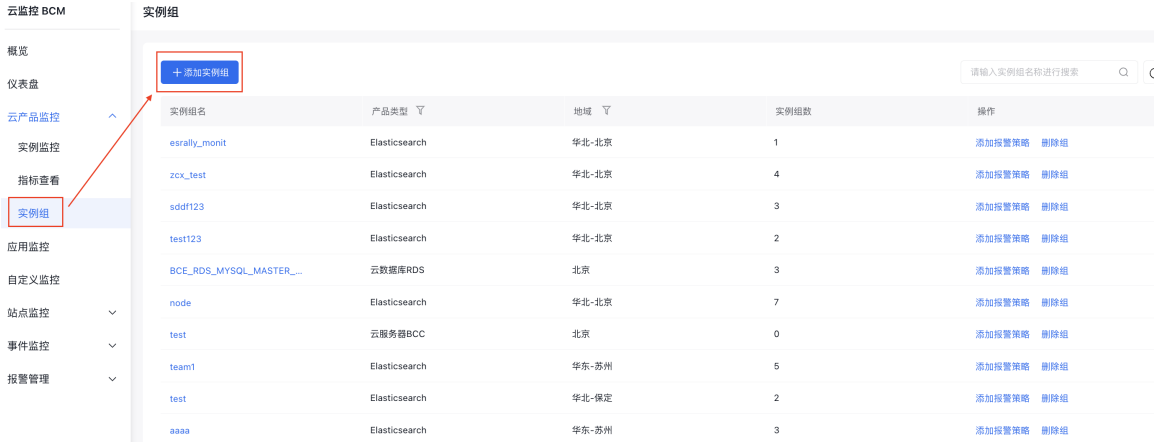
添加实例组

目前百度智能云 Elasticsearch 实例组只支持节点维度，可选择多个节点（实例）组成实例组，按组对节点（实例）进行监控和配置报警策略。

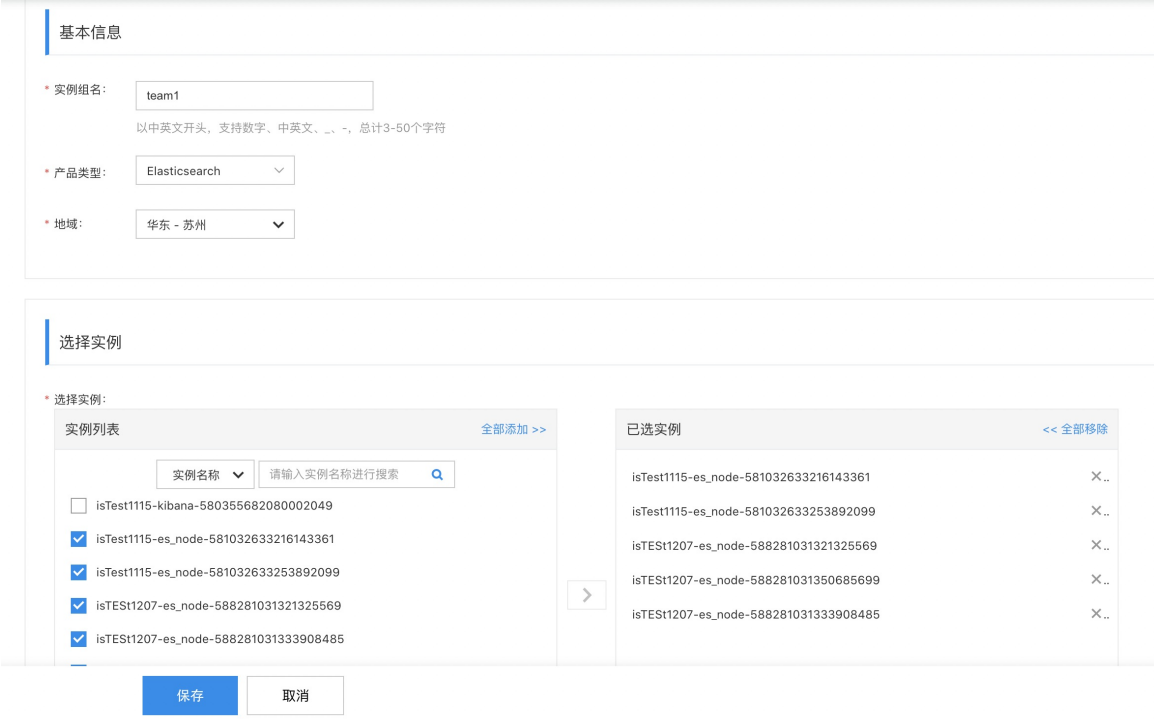
1.进入云监控 BCM 管控平台 云监控BCM。

2.在侧边栏中点击“实例组”。

3.点击“添加实例组”按钮，创建新的实例组。

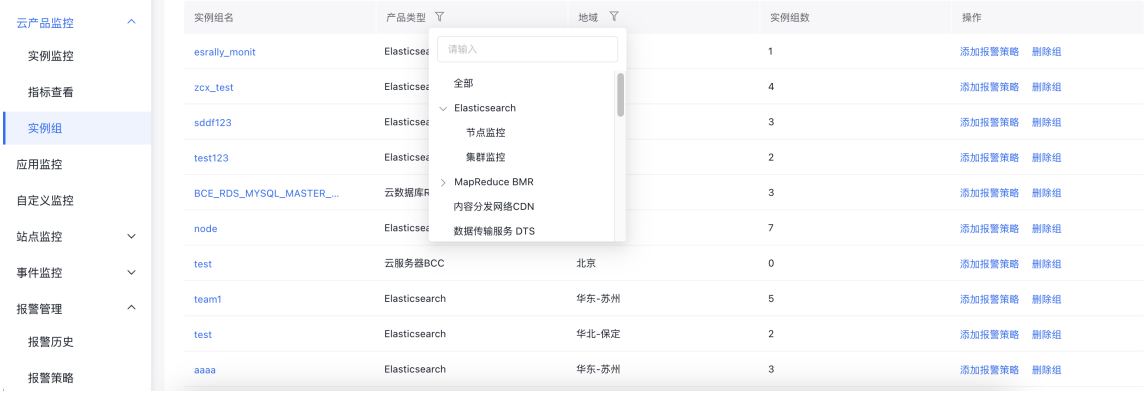


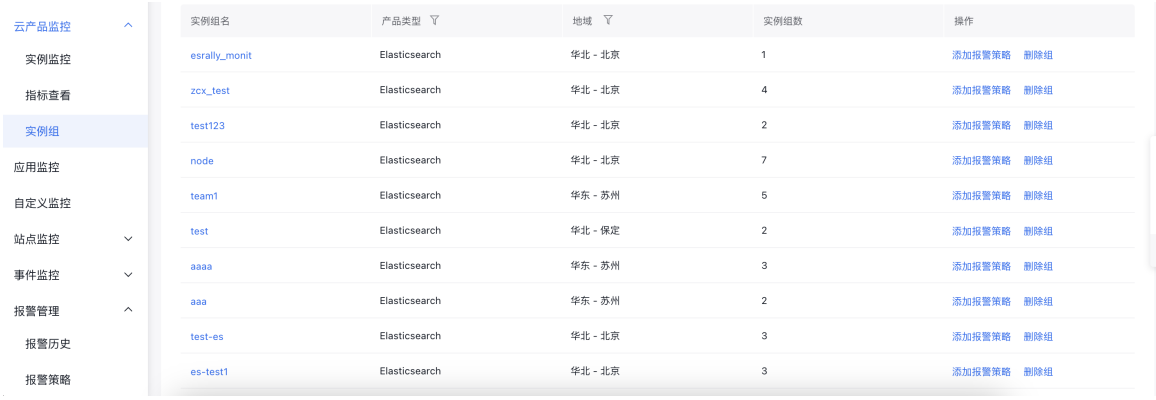
4.填写基本信息，选择想添加到实例组的多个实例，点击保存即可。



查看和管理实例组

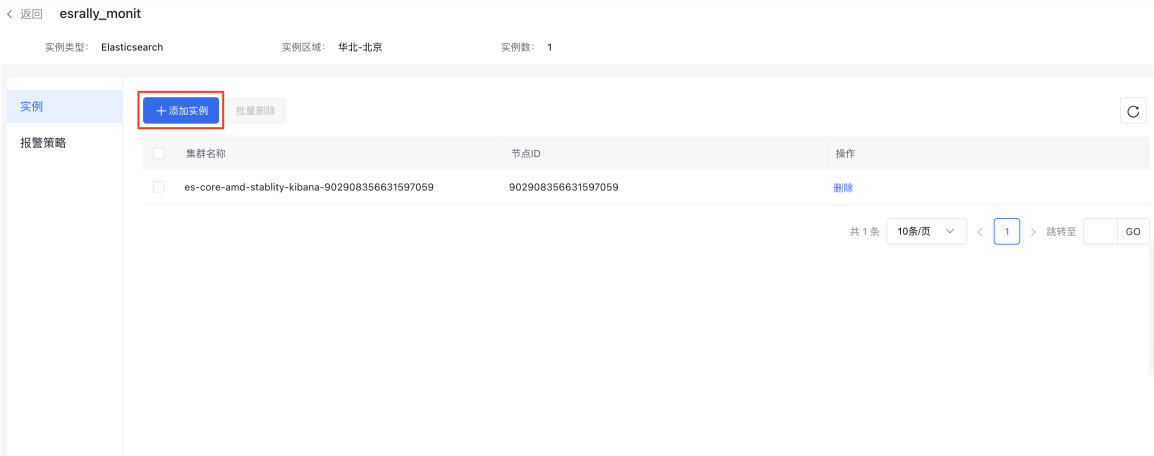
1.在实例组列表中，选择产品类型为“Elasticsearch”，可查看现有Elasticsearch实例组。





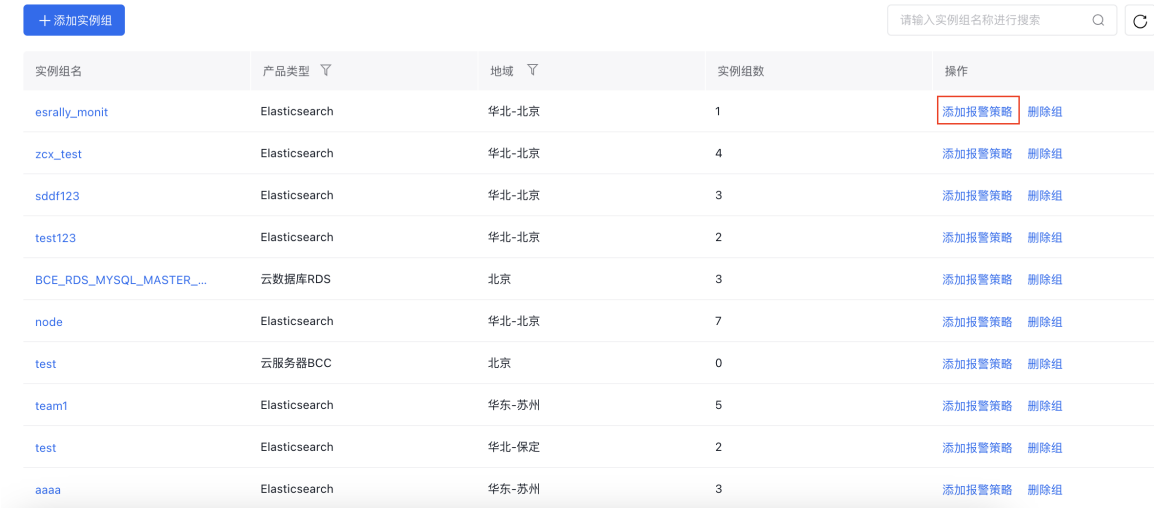
2.点击操作列中的“删除组”按钮，可删除实例组。

3.点击实例组名称，可修改实例组中实例内容。

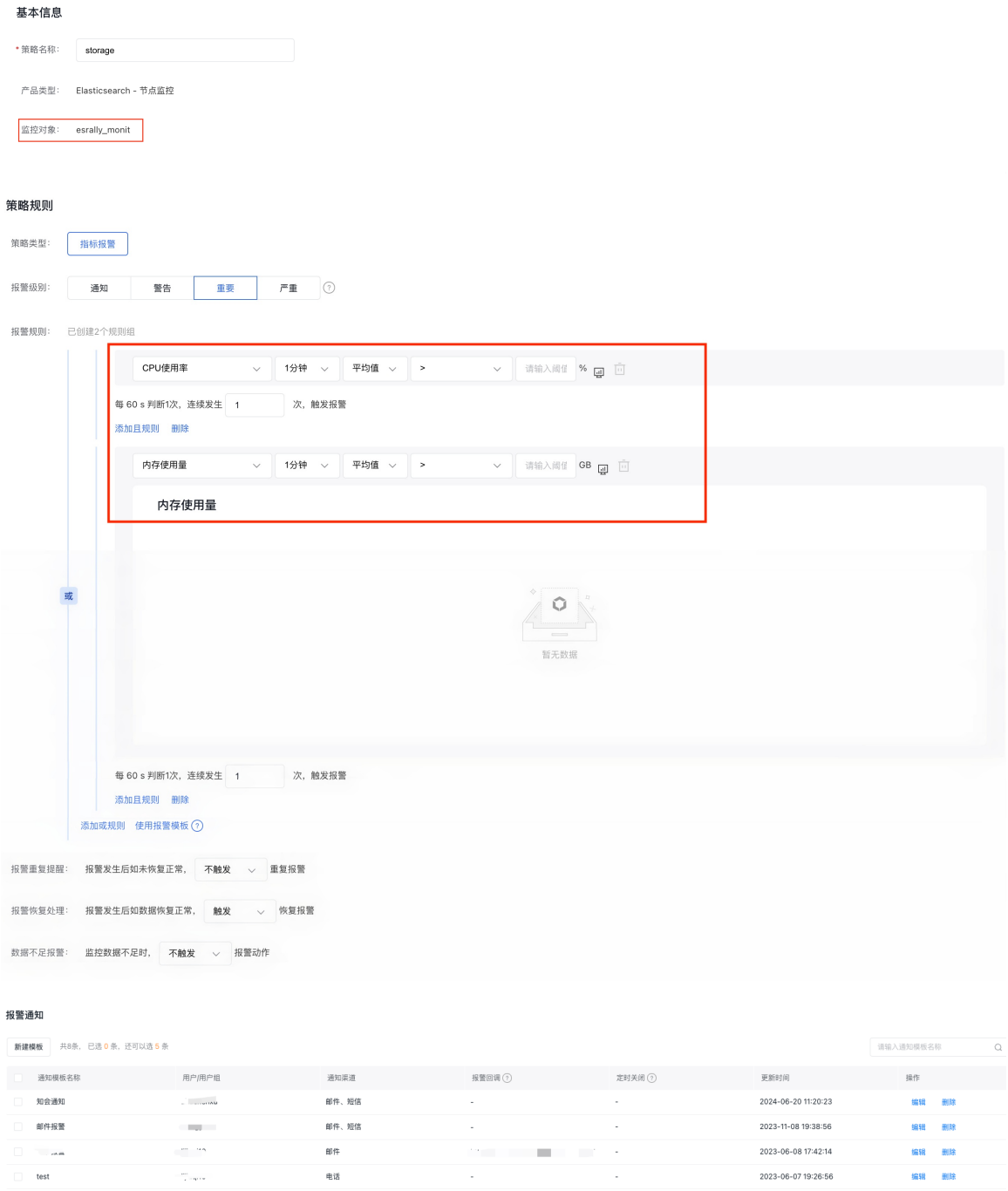


添加报警策略 为实例组添加报警策略，实现多节点（实例）监控指标批量报警。

1.实例组列表，点击操作列的“添加报警策略”按钮。



2.填写基本信息，配置策略规则和报警通知。



3.配置后点击保存，完成实例组报警策略配置。

当实例组中的任一实例的策略规则超过了阈值，就会触发警报。

查看和编辑报警策略

1.点击 Elasticsearch 实例组名称。

2.在详情页左侧侧边栏中选择“报警策略”，可查看此实例组已配置的报警策略。

3.根据列表中的“是否开启通知”功能，开启和关闭报警策略。点击操作列中的“编辑”按钮，可继续编辑报警策略。操作列中的“删除”按钮，可删除对应报警策略。

4.左上角“创建策略”按钮，支持为实例组创建新的报警策略。

更多实例组功能说明，可参考实例组

监控查看及指标说明

百度智能云Elasticsearch 对运行中的 BES 集群，提供了多项监控指标，用以监测集群的运行情况。用户可以根据这些指标实时了解集群服务的运行状况，针对可能存在的风险及时处理，保障集群的稳定运行。本文为您介绍通过 BES 控制台查看集群监控的操作。

操作步骤

1.登录百度智能云 Elasticsearch 控制台。

2.在集群列表中点击目标集群ID。

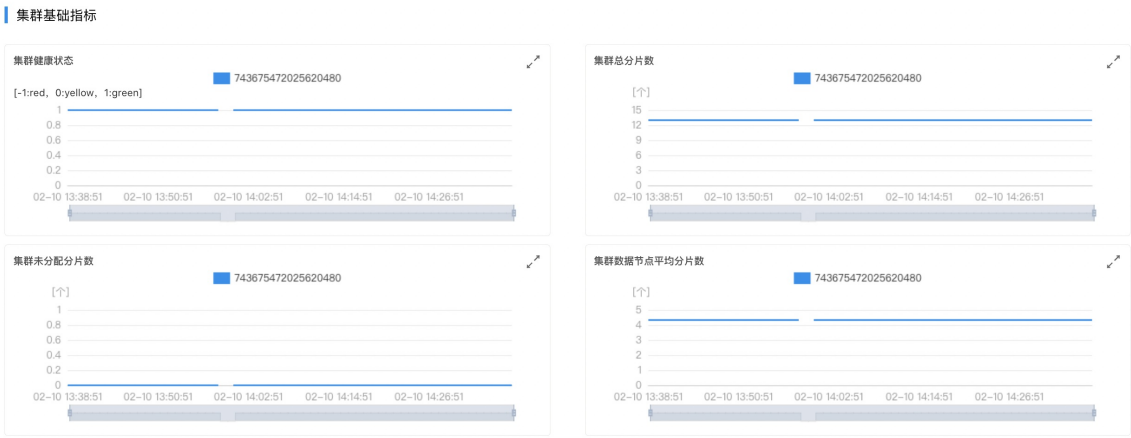
3.在集群信息页左侧的导航栏中选择**监控**。

BES提供两大类指标，集群监控指标和节点监控指标。默认展示集群监控指标。可以查看集群整体运行情况。上方tab可切换为节点监控指标，查看集群内各节点的运行情况和性能指标。

集群监控

在集群监控页，可以看到集群的监控数据信息，可通过选择不同的时间范围查看集群基础指标、集群性能指标和负载均衡指标。

集群基础指标：**集群健康状态、集群总分片数、集群数据节点平均分片数、集群未分配分片数等**



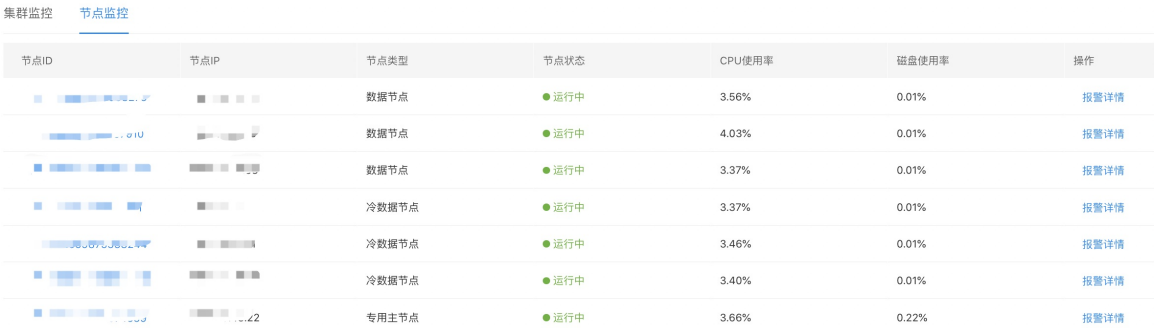
集群性能指标：**集群写入QPS、集群查询QPS、集群写入增量、集群查询增量、集群平均写入耗时、集群平均查询耗时等。**

负载均衡端口指标：**负载均衡服务端口健康检查、负载均衡服务端口网络流量、负载均衡服务端口网络数据包和负载均衡服务端口并发连接数。**

页面右上角可跳转到BCM进行告警策略设置。

节点监控

节点列表 展示集群各个节点基本信息和部分运行指标。



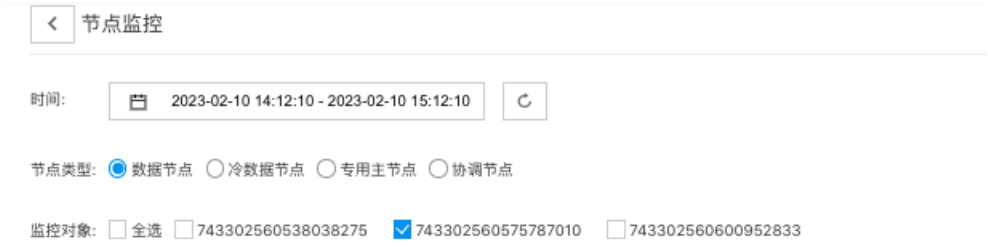
集群监控 节点监控

节点ID	节点IP	节点类型	节点状态	CPU使用率	磁盘使用率	操作
		数据节点	运行中	3.56%	0.01%	报警详情
		数据节点	运行中	4.03%	0.01%	报警详情
		数据节点	运行中	3.37%	0.01%	报警详情
		冷数据节点	运行中	3.37%	0.01%	报警详情
		冷数据节点	运行中	3.46%	0.01%	报警详情
		冷数据节点	运行中	3.40%	0.01%	报警详情
		专用主节点	运行中	3.66%	0.22%	报警详情

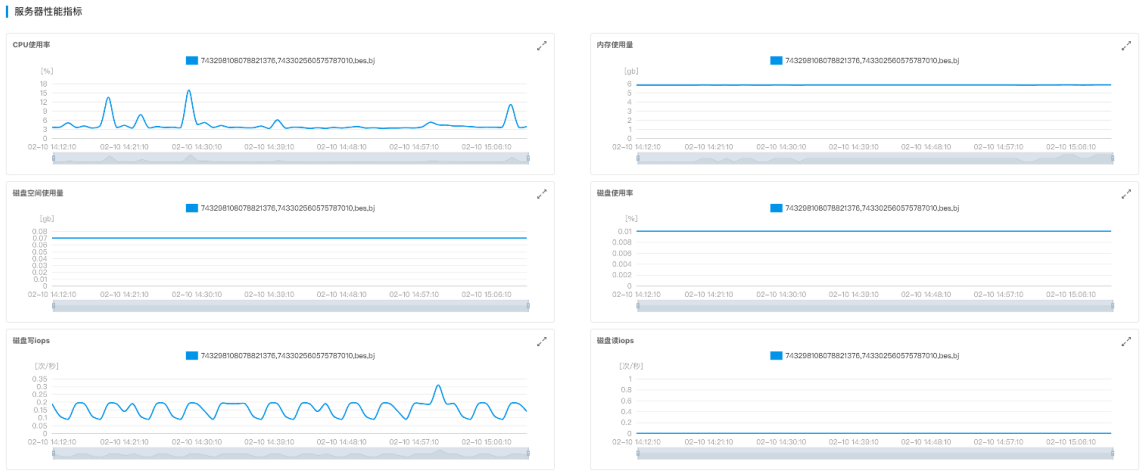
操作列【报警详情】按钮，点击跳转BCM配置对应节点的告警策略配置。

节点状态指标 节点列表页，点击节点ID，进入监控指标页，查看节点各项指标的详细运行情况。包括服务器性能指标和节点性能指标。

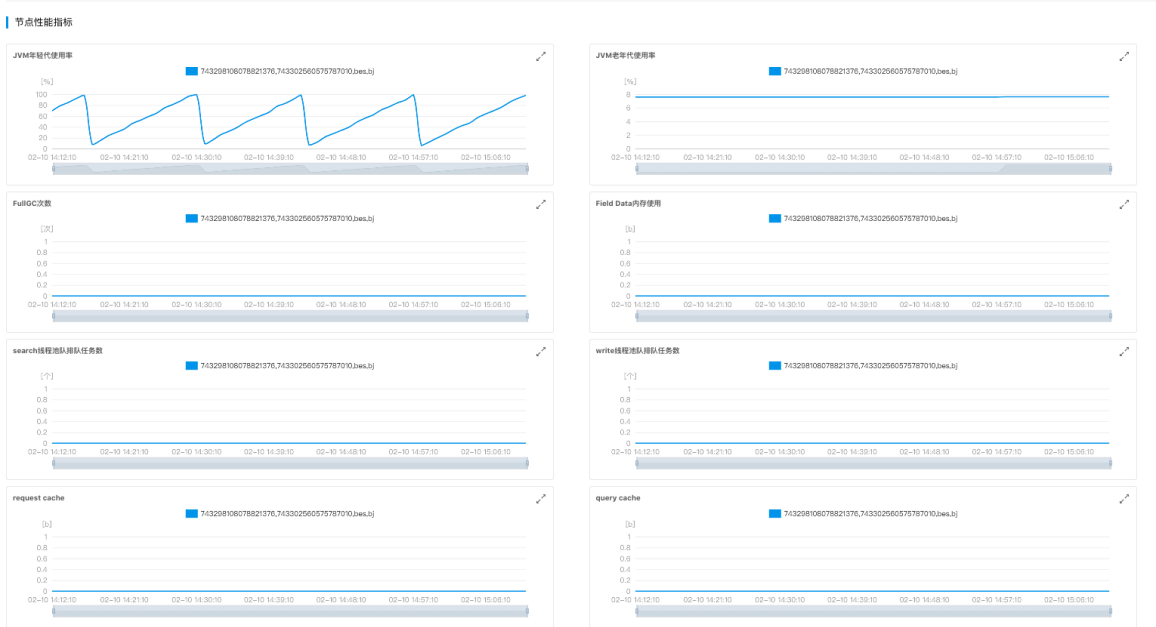
支持选择不同的相对和绝对时间范围、节点ID和节点类型查看指标。



服务器性能指标：**CPU使用率、内存使用量、磁盘空间使用量、磁盘使用率、磁盘写iops、磁盘读iops、磁盘写流量、磁盘读流量、load_1m、IO Util（平均值和最大值）等。**



节点性能指标：**JVM年轻代使用率、JVM老年代使用率、FullGC次数、Field Data内存使用、search线程池队排队任务数、write线程池队排队任务数、request cache、query cache、HeapMemory使用率、段内存等。**



部分指标含义及说明

对部分集群指标和节点指标进行说明。具体指标含义说明以及告警阈值建议如下

> 注意，为便于用户配置合理的监控指标项告警，在此分为3档提供推荐配置阈值：
>
> ★★★为强烈推荐用户去配置此监控项的告警
>
> ★★和★推荐程度逐次递减
>
> 没有★的指标项，用户可视具体业务情况而定

集群指标说明 指标的统计周期均为60秒，即每60秒对集群的指标采集1次。具体各指标含义说明如下：

指标名称	指标说明	推荐配置告警	建议告警阈值-低风险	建议告警阈值-高风险
集群健康状态	BES集群的健康状态。 "1" 表示 green ，表示所有的主分片和副本分片都可用，集群处于最健康状态。 "0" 表示 yellow ， 表示所有的主分片均可用，但部分副本分片未分配（unassigned）。此时搜索结果仍然是完整的。但集群的高可用性在一定程度上受到影响。在集群健康状态变为 yellow 后，建议及时调查和定位问题并修复，防止数据丢失。 "-1" 表示 red ，集群异常状态，表示该集群中某个或某几个索引的主分片未分配（unassigned）。在集群健康状态变为 red 后，应及时定位异常分片，并进行修复。 "-2" 表示 gray ，表示未知状态。	配置告警推荐程度： ★★★ 直接反映了集群可用性，需要高度关注	1分钟最大值<=0	1分钟最大值<=-1
集群总分片数	BES集群已分配shard总数。			
集群数据节点平均分片数	BES集群已分配shard总数/数据节点总数。阈值由用户自行设定，建议不超过1000，超过阈值后请及时清理数据。			
集群未分配分片数	BES集群未分配的分片总数。	配置告警推荐程度：★★ 达到限制后无法创建新索引，ES官方推荐数据节点的分片数不超过1000个，如果调整过设置，需要根据设置调整告警阈值	1分钟最大值>=800	1分钟最大值>=1000
集群写入QPS	BES集群在统计周期内（60秒）每秒写入文档的数量的平均值。 如果在1秒内，客户端向BES集群发送1个文档的写入请求，对应写入QPS为1。			
集群查询QPS	BES集群在统计周期内（60秒）每秒query的数量的平均值。查询QPS数量与待查询索引的分片个数有关，客户端的1个查询可能涉及多个分片。 如果在1秒内，客户端向BES集群发送1个查询请求，被查询的分片有3个，对应查询query的QPS为3。			
集群平均写入耗时	BES集群在统计周期内（60秒）所有节点单次 index 请求耗时的平均值			
集群平均查询耗时	BES集群在统计周期内（60秒）所有节点单次查询请求耗时的平均值。			
集群写入增量	BES集群在统计周期内（60秒）写入文档的增加的数量。集群写入增量和副本数相关。 如果用户设置1个副本，在60秒内，客户端共向BES集群发送1个文档的写入请求，对应写入增量为2.			
集群查	BES集群在统计周期内（60秒）查询query的数量。集群查询增量与待查询索引的分片个数有关，客户端的1个查询可能涉及多个分片。			

询增量	如果在60秒内，客户端共向BES集群发送1个查询请求，被查询的分片有3个，对应查询query增量为3。			
集群写入拒绝率	BES集群在统计周期内（60秒），被拒绝的写入请求数/总写入请求数。具体计算规则：根据 GET /_bpack/metrics/nodes 接口，统计总写入请求数和被拒绝的写入请求数。总写入请求数是es的action层接收到的总体写入请求数，返回状态码为429的作为拒绝数。取相邻两次记录的差值，然后汇聚计算每个统计周期内的绝对值。 说明：2024年8月20日之后新创建的集群默认支持统计该指标，存量集群需要安装对应插件（baidu-metrics），从而支持该指标统计。插件安装详见文档。	配置告警推荐程度：★ 有数据写入拒绝代表写入压力大	1分钟最大值>0%	1分钟最大值>10%
集群查询拒绝率	BES集群群在统计周期内（60秒），被拒绝的查询请求数/总查询请求数。具体计算规则：根据 GET /_bpack/metrics/nodes 接口，统计总查询请求数和被拒绝的查询请求数。总查询请求数是es的action层接收到的总体查询请求数，返回状态码为429的作为拒绝数。取相邻两次记录的差值，然后汇聚计算每个统计周期内的绝对值。 说明：2024年8月20日之后新创建的集群默认支持统计该指标，存量集群需要安装对应插件（baidu-metrics），从而支持该指标统计。插件安装详见文档。	配置告警推荐程度：★ 有数据查询拒绝代表查询压力大	1分钟最大值>0%	1分钟最大值>10%
集群写入失败率	BES集群群在统计周期内（60秒），失败的写入请求数量/总写入请求数。具体计算规则：根据 GET /_bpack/metrics/nodes 接口，统计总写入请求数和失败的写入请求数。总写入请求数是es的action层接收到的总体写入请求数，所有返回状态码非2XX的作为失败数。取相邻两次记录的差值，然后汇聚计算每个统计周期内的绝对值。 说明：2024年8月20日之后新创建的集群默认支持统计该指标，存量集群需要安装对应插件（baidu-metrics），从而支持该指标统计。插件安装详见文档。	★★ 数据写入有失败情况	1分钟最大值>0%	1分钟最大值>10%
集群查询失败率	BES集群群在统计周期内（60秒），失败的查询请求数量/总查询请求数。具体计算规则：根据 GET /_bpack/metrics/nodes 接口，统计总查询请求数和失败的查询请求数。总查询请求数是es的action层接收到的总体查询请求数，所有返回状态码非2XX的作为失败数。取相邻两次记录的差值，然后汇聚计算每个统计周期内的绝对值。 说明：2024年8月20日之后新创建的集群默认支持统计该指标，存量集群需要安装对应插件（baidu-metrics），从而支持该指标统计。插件安装详见文档。	★★ 数据查询有失败情况	1分钟最大值>0%	1分钟最大值>10%
负载均衡服务端口健康检查	包含3个指标，该集群的负载均衡（BLB）实例总数、正常实例数和异常实例数量。	★★ 存在异常探针数量代表不能通过BLB正常访问的BES节点数量	5分钟异常探针数量平均值>0	5分钟异常探针数量平均值>1
负载均衡服务端口网络流量	包含2个指标，负载均衡端口的网络输入流量和网络输出流量。可参考业务实际流量对指标进行观测。			
负载均衡服务端口网络数据包	包含2个指标，负载均衡端口的输入数据包数和输出数据包数。可参考业务收发数据包数量对指标进行观测。			

负载均衡服务端口并发连接数	BES负载均衡端口的并发连接数。			

节点指标说明 指标的统计周期均为60秒，即每60秒对集群的指标采集1次。具体各指标含义说明以及告警阈值建议如下

> 注意，为便于用户配置合理的监控指标项告警，在此分为3档提供推荐配置阈值：
>
> ★★★为强烈推荐用户去配置此监控项的告警
>
> ★★和★推荐程度逐次递减
>
> 没有★的指标项，用户可视具体业务情况而定

指标名称	指标说明	推荐配置告警	建议告警阈值-低风险	建议告警阈值-高风险
节点状态	统计周期内（60秒），节点的运行状态。监控指标中的节点状态和集群信息-集群架构图中的节点状态一致。 "1" 对应颜色为 green 绿色 ，表示节点正常运行中。 "0" 对应颜色为 blue 蓝色，表示节点正在生效中，生效中的节点不可进行暂停和启动等操作。节点启停功能详见节点启停文档。 "-1" 对应颜色为 gray 灰色，表示节点已经停止，已经停止的节点不可进行暂停的操作，可以进行节点启动操作。 "-2" 对应颜色为 orange 橙色，表示节点未知状态，未知状态的节点不可以进行暂停和启动等操作。	★★★	10分钟最大值<=0	10分钟最大值<=-1
CPU使用率	统计周期内（60秒），节点的CPU使用率百分比。	★★	5分钟平均值>80%	5分钟平均值>90%
内存使用量	统计周期内（60秒），节点的内存（mem）使用量。			
磁盘空间使用量	统计周期内（60秒），节点的磁盘使用量。			
磁盘空间使用率	统计周期内（60秒），节点的磁盘使用率。	★★★ 磁盘使用率超过es的水位线可能影响索引创建、数据写入	1分钟最大值>75%	1分钟最大值>85%
磁盘写iops	统计周期内（60秒），磁盘每秒io写次数（次/秒）			
磁盘读iops	统计周期内（60秒），磁盘每秒io读次数（次/秒）			
磁盘写流量	统计周期内（60秒），磁盘每秒io写速率（Kb/秒）	★ 可视具体磁盘类型而定。各种磁盘的吞吐		

		（见附录）		
磁盘读流量	统计周期内（60秒），磁盘每秒io读速率（Kb/秒）	★ 可视具体磁盘类型而定。各种磁盘的吞吐（见附录）		
load_1m	在统计周期内（60秒），节点在1分钟内的负载情况，表示各节点的系统繁忙程度。该指标的正常数值，应该低于对应节点规格的CPU核数。load_1m 过高时，建议降低集群负载或调大集群节点规格。	★★	10分钟平均值>节点cpu核数*0.7	10分钟平均值>节点cpu核数*1.5
IO Util	统计周期内（60秒），节点的IO使用率。此指标提供平均值和最大值，平均值表示60秒内节点IO使用率的平均值，最大值表示60秒内节点IO使用率的最大值。默认展示平均值，可切换成最大值。建议将报警阈值设置在90%。此指标的最大值可能出现毛刺现象，建议拉长报警间隔，持续观测。	★	5分钟IOUtilAvg平均值>80%	5分钟IOUtilAvg平均值>90%
JVM年轻代使用率	统计周期内（60秒），节点的 JVM 年轻代内存使用率。			
JVM老年代使用率	统计周期内（60秒），节点的 JVM 老年代内存使用率。	★	1分钟最大值>80%	1分钟最大值>90%
FullGC次数	统计周期内（60秒），节点的gc总次数。	★★ 频繁GC会影响集群性能和稳定性	10分钟和值>=2	10分钟和值>=5
Field Data内存使用	统计周期内（60秒），节点的fielddata内存占用情况，监控曲线越高，说明堆内存存在大量的fielddata数据缓存，过大的fielddata内存占用会触发fielddata内存熔断，影响集群稳定性。			
search线程池队排队任务数	统计周期内（60秒），search线程池中的队列数。	★ 可视具体线程池大小而定		
write线程池队排队任务数	统计周期内（60秒），write线程池中的队列数。	★ 可视具体线程池大小而定		
request cache	统计周期内（60秒），request 缓存大小。			
query cache	统计周期内（60秒），query 缓存大小。			
HeapMemory使用率	统计周期内（60秒），节点的HeapMemory使用率百分比。当HeapMemory使用率较高或存在较大的内存对象时，会影响集群服务，也会自动触发gc操作。建议报警阈值为75%。	★★	1分钟最大值>75%	1分钟最大值>85%
段内存	统计周期内（60秒），为了提高搜索效率而缓存在内存中的段的数据。建议报警阈值为（堆内存 * 30%)。			

附录 磁盘容量为512GB时，各类磁盘吞吐对比，详细可参考 https://cloud.baidu.com/doc/CDS/s/hketf8fyr

类型	随机IOPS	吞吐量(MBps)	访问延时(ms)
增强型SSD_PL2	27400	376	小于2
增强型SSD_PL1	27400	350	小于2
SSD云盘	17160	300	小于2
高效云盘	5000	140	小于3

告警功能详见配置告警文档或BCM云监控文档。

版本升级

升级版本

百度智能云 Elasticsearch 提供版本升级的功能，支持 ES 从低版本到高版本的升级。可根据业务需要对集群进行升级，实现业务的无缝过渡。

升级版本和升级模式说明

升级版本

升级版本包括对集群版本进行升级，对内核版本进行升级。

- 当前支持通过升级功能的集群版本。

当前集群版本	目标集群版本	可选升级模式
6.5.3	6.8.14、7.4.2、7.10.2	• 6.5.3版本 升级到 6.8.14版本，支持蓝绿模式和冷重启模式 • 6.5.3版本 升级到 7.4.2版本 或 7.10.2版本，只支持冷重启模式
6.8.14	7.4.2、7.10.2	支持蓝绿模式和冷重启模式
7.4.2	7.10.2	支持蓝绿模式和冷重启模式

- 当前内核版本升级仅支持从低版本向高版本进行升级，并且不可以跨集群版本进行升级，升级模式支持蓝绿模式和冷重启模式。内核版本说明

> **例外**
>
> Elasticsearch 7.10.2 版本的内核 1.0.0 版本冷热分离索引升级到高版本仅支持冷重启模式，不支持蓝绿模式。

升级模式

升级模式包括蓝绿模式和冷重启模式。

升级模式	原理	风险和建议
蓝绿模式	在集群中先增加高版本节点，再将低版本节点的数据迁移到高版本节点，最后删除低版本节点。	• 升级操作无法取消，升级期间可以继续向集群写入数据或从集群读取数据，但不能进行其他变更操作，建议在流量低峰期进行。 • 节点的IP地址会发生变化。如果在集群配置或客户端访问集群的代码中指定了节点IP地址，需要在升级后更新配置或代码。
冷重启模式	所有节点完全关闭后，一次性全量重启。	• 通过冷重启模式进行升级期间，集群停服，无法读写。 • 冷重启后，集群节点的IP地址不会发生变化，用户无需对实例进行配置变更。

> **说明：** 选择在蓝绿模式下进行版本升级时，若所在集群的子网IP数量不足，无法实现升级。请保证子网IP数量充足，再进行升级。

升级流程

Elasticsearch 版本升级，需要先进行数据备份、升级检查两个步骤。数据备份虽然非必须，但是强烈建议进行数据备份。升级检查成功后，才会开始执行升级操作。

1. 快照备份

升级操作前，需要用户手动对集群进行快照备份，以防升级操作失败时，可以用快照还原集群。

2. 升级检查

   检查升级前后两个版本是否有不兼容的配置，如果检查不通过，流程就会终止。如果升级过程中，遇到升级检查不通过的情况，用户可以查看具体的检查项和相应的解决方案，详情可参见升级检查。

   用户也可以在升级操作前，选择单独进行升级检查，检查集群是否满足升级条件。

3. 升级过程和集群重启

   BES提供冷重启和蓝绿两种升级模式，用户可根据业务场景选择。具体升级时长与用户集群中的数据大小、数据结构、集群规格等都有关。方式说明如下：

- 蓝绿模式，无需重启集群，升级期间服务可以继续向集群写入数据或从集群读取数据，但不能进行其他变更操作。需要拷贝数据从而升级速度较慢，建议在流量低峰期进行。
- 冷重启模式，需要重启集群，速度较快，重启期间集群会停服，服务不可访问，升级完成后才能正常访问。

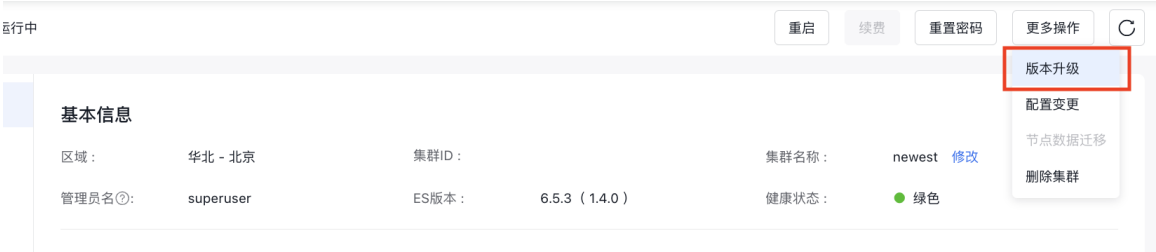
> **重要**
>
> - 升级操作无法回退，建议用户提前备份数据。

操作步骤

集群版本升级

1. 登录百度智能云 Elasticsearch 控制台。
2. 进入集群列表界面，选中想要升级的集群，进入目标集群详情页。
3. 点击左上角**更多操作-“版本升级”**按钮，进入版本升级页面。

用户在**集群信息-基本信息**里可以查看集群名称，集群ID，当前的ES版本以及内核版本。



> 注意
>
> - 若ES集群版本或内核版本都不可进行升级时，更多操作的“版本升级”按钮无法点击。

用户在**升级配置**里可以对升级类型，目标版本，操作类型和升级模式进行选择。

升级类型选择**集群版本升级**。

> 注意
>
> - 目标版本默认为最新集群版本，用户可下拉菜单选择其他版本。

操作可以选择“仅做升级检查，暂不升级”或“执行升级操作”。

> - 仅做升级检查，暂不升级
>
> 升级检查，系统会对配置兼容性、集群状态、集群快照备份以及基础资源进行检查。用户可以在操作历史页查看检查结果。检查完成后，请根据页面提示处理异常结果。
>
> - 执行升级操作
>
> 选择“执行升级操作”，需要用户选择升级模式，蓝绿模式或冷重启模式。系统会进行先升级检查然后执行升级操作。

升级模式包括“蓝绿模式”和“冷重启模式”。

4. 选择完成后，单击“确定”按钮开始执行对应的升级操作。升级时间一般持续1个小时左右。待升级完成后，可在集群的基本信息页面，查看升级后的集群版本。

5. 查看升级记录，**操作历史**里可以查看操作时间，操作行为，操作详情和进度。

操作行为为版本升级。操作详情分别显示原ES版本和新ES版本。进度包括集群状态检查，配置兼容检查，升级BES集群，升级重启 Kibana 和操作完成五个环节。

内核版本升级

1. 登录百度智能云 Elasticsearch 控制台。

2. 进入集群列表界面，选中想要升级的集群，进入目标集群详情页。

3. 点击左上角**更多操作**-**“版本升级”**按钮，进入版本升级页面。

用户在**集群信息**-**基本信息**里可以查看集群名称，集群ID，当前集群版本以及当前内核版本。

> **注意**
>
> - 若内核版本可进行升级时，在版本号右侧，会出现“有可更新的内核版本”提示文案。
> - 若ES集群版本或内核版本都不可进行升级时，更多操作的“版本升级”按钮无法点击。

用户在**升级配置**里可以对升级类型，目标版本，操作类型和升级模式进行选择。

升级类型选择**更新内核版本**。

> 注意
>
> - 目标内核版本默认为最新内核版本，用户可下拉菜单选择其他版本。

操作可以选择“仅做升级检查，暂不升级”或“执行升级操作”。

> - 仅做升级检查，暂不升级
>
> 升级检查，系统会对配置兼容性、集群状态、集群快照备份以及基础资源进行检查。用户可以在操作历史页查看检查结果。检查完成后，请根据页面提示处理异常结果。
>
> - 执行升级操作
>
> 选择“执行升级操作”，需要用户选择升级模式，蓝绿模式或冷重启模式。系统会进行先升级检查然后执行升级操作。

升级模式包括蓝绿模式和冷重启模式。

4. 选择完成后，单击“确定”按钮开始执行对应的升级操作。升级时间一般持续1个小时左右。待升级完成后，可在集群的基本信息页面，查看升级后的内核版本。
5. 查看升级记录，**操作历史**里可以查看操作时间，操作行为，操作详情和进度。

操作行为为更新内核补丁。操作详情分别显示原内核版本和新内核版本。进度包括集群状态检查，配置兼容检查，更新内核补丁和操作完成四个环节。

> **注意**
>
> - 老套餐暂不支持更新内核版本功能。
> - 5.x版本新老套餐暂不支持更新内核版本功能。

升级检查

在升级百度智能云 Elasticsearch 集群版本前，用户需要通过升级检查，检查集群状态是否正常、是否存在不兼容的配置等，并进行调整。

本文介绍在版本升级前需要完成的人工检查、集群状态检查和系统兼容性检查，以及兼容性检查失败时的调整方法。

背景信息

- 升级检查操作入口，在 百度智能云 Elasticsearch 控制台-**集群详情**，单击版本升级。版本升级操作步骤详见版本升级。
- 本文的命令都可在Kibana控制台上执行，关于如何登录Kibana控制台，请参见Kibana使用指南。

> **注意：**

> 需特别留意本文中需要人工检查的部分（无法自动检查），并在升级前完成代码兼容性改造，避免升级后造成集群无法正常访问。

通用检查项 每个版本升级之前，系统都会自动进行以下几个通用检查项的检查。

集群健康状态检查

序号	配置维度	配置信息	兼容性	详细说明
1	集群级别	集群健康状态	CRITICAL	集群健康状态

备份检查

序号	配置维度	配置信息	兼容性	详细说明
1	集群级别	快照备份	WARNING	详细介绍参考 快照备份文档

系统插件检查

序号	配置维度	配置信息	兼容性	详细说明
1	集群级别	集群插件	CRITICAL	详细介绍参考 插件说明文档

> **说明：**
>
> - 自定义插件不支持直接升级，请先卸载自定义插件，再进行版本升级操作。升级成功后可重新安装自定义插件。

Elasticsearch 6.5 版本升级到 6.8 版本配置检查

自动检查的配置项列表

以下均为升级检查中自动执行的系统检查项，用户也可手动检查并修改。

序号	配置维度	配置信息	兼容性	详细说明
1	集群级别	集群设置（node.store.allow_mmapfs）	CRITICAL	集群设置`node.store.allow_mmapfs`，在6.8版本改名为`node.store.allow_mmap`，详见 Breaking changes in 6.8
2	索引级别	索引设置（geo_shape）	WARNING	索引设置tree, precision, tree_levels, distance_error_pct, points_only, 和strategy被废弃被废弃，详见 Breaking changes in 6.8
3	索引模板级别	索引模板设置（geo_shape）	WARNING	索引设置tree, precision, tree_levels, distance_error_pct, points_only, 和strategy被废弃被废弃，详见 Breaking changes in 6.8

> **说明：**
>
> - 警告（WARNING）：检查失败时仍可以升级。此类型检查项对应设置在升级后将被忽略。
> - 错误（CRITICAL）：检查失败时无法升级。此类型检查项对应设置在目标版本无法兼容。

配置不兼容调整方法

集群级别

- 集群设置（node.store.allow_mmapfs） 在**配置修改**页面Elasticsearch YML文件配置中删除设置后重启集群

- 索引设置（geo_shape） 通过 ES 索引设置更新接口取消此设置。

索引级别

- 索引设置（geo_shape） 通过 ES 索引设置更新接口取消此设置。

索引模板级别

1. 使用 API `PUT _template/***` 更新模板。

Elasticsearch 6.x 版本升级到 7.x 版本配置检查

需人工自查的不兼容配置

- 对于超过1024个字段数的索引，查询时如果不指定字段会非常消耗性能。建议设置默认查询字段 `index.query.default_field`。如果仍要查询所有字段，那么可以通过提高“布尔查询最大子句数”设置 `indices.query.bool.max_clause_count` 来实现。
- `include_type_name` 默认为 false，升级后默认将无法创建包含 type 的索引，需要显式指定参数 `include_type_name`：

> 注意：
>
> 用户需重点关注集群是否有此项不兼容配置，避免升级后造成集群无法正常访问。

```
PUT my_index?include_type_name
{
  "mappings": {
    "type1": {
      "properties": {
        "name": {
          "type": "text"
        }
      }
    }
  }
}
```

自动检查的配置项列表

序号	配置维度	配置信息	兼容性	详细说明
1	集群级别	用户代理预处理器（User agent processor）设置	WARNING	用户代理预处理器（User agent processor）的格式默认值变为`ecs`，详见 Breaking changes in 7.0
2	集群级别	集群分片数	WARNING	每个节点最大 shard 数`cluster.max_shards_per_node`默认1000，shard 数超过在升级后将无法创建新分片，详见 Breaking changes in 7.0
3	集群级别	集群设置（discovery.zen.no_master_block）	WARNING	集群设置 `discovery.zen.no_master_block` 改名为 `cluster.no_master_block`，详见 Breaking changes in 7.0
4	集群级别	预处理器管道（Ingest pipeline）日期格式	CRITICAL	预处理器管道（Ingest pipeline）`date`或`date_index_name`处理器使用废弃的日期格式，详见 Breaking changes in 7.0
5	集群级别	集群设置（thread_pool.index）	CRITICAL	`thread_pool.index.*` 设置被废弃，详见 Breaking changes in 7.0
6	集群级别	集群设置（thread_pool.bulk）	CRITICAL	`thread_pool.bulk.*` 设置被废弃，详见 Breaking changes in 7.0

	别	（thread_pool.bulk）	L	
7	集群级别	集群设置（pidfile）	CRITICAL	集群设置`pidfile` 改名为`node.pidfile`，详见 Breaking changes in 7.0
8	集群级别	集群设置（processors）	CRITICAL	集群设置`processors` 改名为`node.processors`，详见 Breaking changes in 7.0
9	集群级别	集群设置（http.tcp_no_delay）	CRITICAL	集群设置`http.tcp_no_delay` 改名为`http.tcp.no_delay`，详见 Breaking changes in 7.0
10	集群级别	集群设置（network.tcp.connect_timeout）	CRITICAL	集群设置`network.tcp.connect_timeout` 被废弃，请使用`transport.connect_timeout`，详见 Breaking changes in 7.0
11	索引级别	索引创建版本	CRITICAL	不允许6.0版本之前创建的索引升级到7.x版本，详见 Breaking changes in 7.0
12	索引级别	多Type索引	CRITICAL	一个索引不能有多个Type，详见 Breaking changes in 7.0
13	索引级别	Delimited payload 词汇单元过滤器（Token filter）索引设置	WARNING	Delimited payload 语汇单元过滤器（Token filter）从 `delimited_payload_filter` 改名为 `delimited_payload`，详见 Breaking changes in 7.0
14	索引级别	索引设置（index.percolator.map_unmapped_fields_as_string）	CRITICAL	索引设置`index.percolator.map_unmapped_fields_as_string` 改名为 `index.percolator.map_unmapped_fields_as_text`，详见 Breaking changes in 7.0
15	索引级别	索引名	WARNING	索引名不能包含“:”，详见 Breaking changes in 7.0
16	索引级别	索引设置（index.unassigned.node_left.delayed_timeout）	CRITICAL	索引设置 `index.unassigned.node_left.delayed_timeout` 负数值被废弃，详见 Breaking changes in 7.0
17	索引级别	索引设置（index.shard.check_on_startup）	CRITICAL	索引设置 `index.shard.check_on_startup` 的值为`fix`被废弃，详见 Breaking changes in 7.0
18	索引级别	经典相似性（classic similarity）索引 Mapping 参数	WARNING	经典相似性（classic similarity）索引 Mapping 参数被废弃，详见 Breaking changes in 7.0
19	索引级别	经典相似性（classic similarity）索引设置	WARNING	经典相似性（classic similarity）索引设置被废弃，详见 Breaking changes in 7.0
20	索引级别	索引字段数	WARNING	对于超过1024个字段的索引，建议设置默认查询字段（`index.query.default_field`）或提升最大字段数设置（`indices.query.bool.max_clause_count`），否则升级后将导致不指定字段的查询无法正常使用，详见 Breaking changes in 7.0
21	索引级别	索引 Mapping 日期格式	CRITICAL	Joda-Time 格式改为 Java Time，详见 Breaking changes in 7.0
22	索引模板级别	Delimited payload 语汇单元过滤器（Token filter）索引模板设置	CRITICAL	Delimited payload 语汇单元过滤器（Token filter）从 `delimited_payload_filter` 改名为 `delimited_payload`，详见 Breaking changes in 7.0
23	索引模	索引模板设置（index.percolator.ma	CRITICA	索引模板设置 `index.percolator.map_unmapped_fields_as_string` 改名为 `index.percolator.map_unmapped_fields_as_text`，升级后不兼容，详见

	板级别	p_unmapped_fields_as_string）	L	Breaking changes in 7.0
24	索引模板级别	索引模板设置（index.unassigned.node_left.delayed_timeout）	CRITICAL	索引模板设置 `index.unassigned.node_left.delayed_timeout` 负数值被废弃，升级后不兼容，详见 Breaking changes in 7.0
25	索引模板级别	索引模板设置（index.shard.check_on_startup）	CRITICAL	索引模板设置 `index.shard.check_on_startup` 的值为`fix`被废弃，升级后不兼容，详见 Breaking changes in 7.0
26	索引模板级别	经典相似性（classic similarity）索引模板 Mapping 参数	CRITICAL	经典相似性（classic similarity）索引模板 Mapping 参数被废弃，升级后不兼容，详见 Breaking changes in 7.0
27	索引模板级别	经典相似性（classic similarity）索引模板设置	CRITICAL	经典相似性（classic similarity）索引模板设置被废弃，升级后不兼容，详见 Breaking changes in 7.0
28	索引模板级别	索引模板字段数	CRITICAL	对于超过1024个字段的索引模板，建议设置默认查询字段（`index.query.default_field`）或提升最大字段数设置（`indices.query.bool.max_clause_count`），否则升级后将导致不指定字段的查询无法正常使用，详见 Breaking changes in 7.0
29	索引模板级别	索引模板 Mapping 日期格式	CRITICAL	Joda-Time 格式改为 Java Time，详见 Breaking changes in 7.0

> **说明：**
>
> - 警告（WARNING）：检查失败时仍可以升级。此类型检查项对应设置在升级后将被忽略。
> - 错误（CRITICAL）：检查失败时无法升级。此类型检查项对应设置在目标版本无法兼容。

配置不兼容调整方法

集群级别

- 用户代理预处理器（User agent processor）设置 使用非`ecs`格式的 user agent 将被废弃，应调整为：

```
PUT _ingest/pipeline/my_pipeline1
{
  "processors" : [
    {
      "user_agent" : {
        "field" : "agent",
        "ecs": true
      }
    }
  ]
}
```

- 集群分片数 对于超过分片数限制的集群，有两种调整方法：
- 降低集群分片数（通过清理过期索引、shrink 索引 等）。
- 修改集群设置提高分片数限制。

```
PUT _cluster/settings
{
  "persistent": {
    "cluster.max_shards_per_node": 5000
  }
}
```

- 集群设置（discovery.zen.no_master_block） 对于使用了集群设置 `discovery.zen.no_master_block` 的集群，在**升级完成后**调整为 `cluster.no_master_block` ，例如：

```
PUT _cluster/settings
{
  "persistent": {
    "cluster.no_master_block": "write"
  }
}
```

- 预处理器管道（Ingest pipeline）日期格式 可参考 Joda based date formatters are replaced with java ones 进行调整。
- 其他集群设置 在**配置修改**页面 Elasticsearch YML文件配置中删除设置后重启集群生效。

索引级别

- 索引创建版本和多Type索引 不允许6.0版本之前创建的索引升级到7.x版本，一个索引不能有多个Type，对于需要保留的索引，需要在升级前执行 reindex ， 例如：

```
POST _reindex
{
  "source": {
    "index": "my_index"
  },
  "dest": {
    "index": "new_my_index"
  }
}
```

- Delimited payload 语汇单元过滤器（Token filter）索引模板设置 将索引设置中的 `delimited_payload_filter` 改为 `delimited_payload` ，例如原始索引如下：

```
PUT my_index
{
  "settings": {
    "analysis": {
      "analyzer": {
        "whitespace_plus_delimited": {
          "tokenizer": "whitespace",
          "filter": [ "plus_delimited" ]
        }
      },
      "filter": {
        "plus_delimited": {
          "type": "delimited_payload_filter",
          "delimiter": "+",
          "encoding": "int"
        }
      }
    }
  }
}
```

修改后的索引：

```
PUT my_index
{
  "settings": {
    "analysis": {
      "analyzer": {
        "whitespace_plus_delimited": {
          "tokenizer": "whitespace",
          "filter": [ "plus_delimited" ]
        }
      },
      "filter": {
        "plus_delimited": {
          "type": "delimited_payload",
          "delimiter": "+",
          "encoding": "int"
        }
      }
    }
  }
}
```

- 索引设置（index.percolator.map_unmapped_fields_as_string） 将索引设置 `index.percolator.map_unmapped_fields_as_string` 调整为 `index.percolator.map_unmapped_fields_as_text`（此设置调整前需要 close 索引）。

```
PUT my_index/_settings
{
  "index.percolator.map_unmapped_fields_as_text": true
}
```

- 索引名 对于包含“:”的索引，需执行 reindex，例如：

```
POST _reindex
{
  "source": {
    "index": "my_index"
  },
  "dest": {
    "index": "new_my_index"
  }
}
```

- 索引设置（index.unassigned.node_left.delayed_timeout） 将索引设置 `index.unassigned.node_left.delayed_timeout` 调整为正整数：

```
PUT my_index/_settings
{
  "index.unassigned.node_left.delayed_timeout": 0
}
```

- 索引设置（index.shard.check_on_startup） 将索引设置 `index.shard.check_on_startup` 值为 `fix` 调整为其它值，例如：

```
PUT my_index/_settings
{
  "index.shard.check_on_startup": true
}
```

- 经典相似性（classic similarity）索引 Mapping 参数、经典相似性（classic similarity）索引设置 对于包含经典相似性（classic similarity）的索引设置或 Mapping，调整为其他类型，例如原始索引如下：

```
PUT my_index
{
  "mappings": {
    "doc": {
      "properties": {
        "field1": {
          "properties": {
            "field4": {
              "type": "text",
              "similarity": "classic"
            }
          }
        }
      }
    }
  }
}
```

修改后的索引为：

```
PUT my_index
{
  "mappings": {
    "doc": {
      "properties": {
        "field1": {
          "properties": {
            "field4": {
              "type": "text",
              "similarity": "BM25"
            }
          }
        }
      }
    }
  }
}
```

- 索引字段数 对于超过1024个字段的索引，有两种调整方法：
- 设置默认查询字段设置 `index.query.default_field`。

```
PUT my_index/_settings
{
  "index.query.default_field": "field1"
}
```

- 在**配置修改**页面 Elasticsearch YML文件配置中，提升最大字段数设置 `indices.query.bool.max_clause_count`
- 索引 Mapping 日期格式 可参考 Joda based date formatters are replaced with java ones 进行调整。

索引模板级别

索引模板级别的不兼容设置调整请参考 索引级别 。

Elasticsearch 7.4 版本升级到 7.10 版本配置检查 自动检查的配置项列表

序号	配置维度	配置信息	兼容性	详细说明
1	集群级别	集群设置（thread_pool.listener.queue_size）	CRITICAL	集群设置thread_pool.listener.queue_size被废弃，详见 Breaking changes in 7.10
2	集群级别	集群设置（thread_pool.listener.size）	CRITICAL	集群设置thread_pool.listener.size被废弃，详见 Breaking changes in 7.10
3	集群级别	集群设置（cluster.remote.connect）	CRITICAL	集群设置cluster.remote.connect改名为node.remote_cluster_client，详见 Breaking changes in 7.10
4	集群级别	集群设置（node.local_storage）	CRITICAL	集群设置node.local_storage被废弃，详见 Breaking changes in 7.10
5	集群级别	集群设置（script.cache.max_size）	CRITICAL	集群设置script.cache.max_size改名为script.context.cache_max_size，详见 Breaking changes in 7.10
6	集群级别	集群设置（script.cache.expire）	CRITICAL	集群设置script.cache.expire改名为script.context.cache_expire，详见 Breaking changes in 7.10
7	集群级别	集群设置（script.max_compilations_rate）	CRITICAL	集群设置script.max_compilations_rate改名为script.context.max_compilations_rate，详见 Breaking changes in 7.10
8	集群级别	集群设置（.tcp.keep_idle和.tcp.keep_interval）	CRITICAL	集群设置*.tcp.keep_idle和*.tcp.keep_interval不能超过300s，详见 Breaking changes in 7.10
9	集群级别	集群设置（cluster.join.timeout）	CRITICAL	集群设置cluster.join.timeout被废弃，详见 Breaking changes in 7.10
10	集群级别	集群设置（cluster.routing.allocation.disk.include_relocations）	CRITICAL	集群设置cluster.routing.allocation.disk.include_relocations被废弃，详见 Breaking changes in 7.10
11	索引级别	索引设置（boost）	CRITICAL	mappings中的boost被废弃，详见 Breaking changes in 7.10
12	索引级别	索引设置（index.translog.retention.size和index.translog.retention.age）	WARNING	index.translog.retention.size和index.translog.retention.age被废弃，详见 Breaking changes in 7.10
13	索引模板级别	索引模板设置（boost）	CRITICAL	mappings中的boost被废弃，详见 Breaking changes in 7.10
14	索引模板级别	索引模板设置（index.translog.retention.size和index.translog.retention.age）	WARNING	index.translog.retention.size和index.translog.retention.age被废弃，详见 Breaking changes in 7.10

说明：

- 警告（WARNING）：检查失败时仍可以升级。此类型检查项对应设置在升级后将被忽略。
- 错误（CRITICAL）：检查失败时无法升级。此类型检查项对应设置在目标版本无法兼容。

配置不兼容调整方法

集群级别

- 集群设置 在**配置修改**页面 Elasticsearch YML文件配置中删除设置后重启集群生效。

索引级别

- 索引设置（boost） 去掉设置
- 索引设置（index.translog.retention.size和index.translog.retention.age） 去掉设置

索引模板级别

- 索引模板设置（boost） 去掉设置
- 索引模板设置（index.translog.retention.size和index.translog.retention.age） 去掉设置

内核版本升级检查项

> **重要**
>
> 此检查只针对于Elastichsearch7.10.2版本，其他集群版本内核升级时不进行检查。

自动检查的配置项列表

序号	配置维度	配置信息	兼容性	详细说明
1	集群级别	向量检索模板设置	WARNING	低版本升级到1.2及以上版本，升级过程终中读写会受影响，升级完毕，读写功能恢复正常。
2	索引级别	向量检索设置	WARNING	低版本升级到1.2及以上版本，升级过程终中读写会受影响，升级完毕，读写功能恢复正常。
3	索引级别	BOS冷热索引检查	CRITICAL	内核1.0.0版本冷热分离索引升级到高版本仅支持冷重启模式，不支持蓝绿模式。

> **说明：**
>
> - 警告（WARNING）：检查失败时仍可以升级。此类型检查项对应设置在升级后将被忽略。
> - 错误（CRITICAL）：检查失败时无法升级。此类型检查项对应设置在目标版本无法兼容。

内核版本说明

在完全兼容开源Elasticsearch内核的所有特性基础上，百度云Elasticsearch在冷热分离、向量检索、查询与写入性能优化等诸多方面，自主研发BES内核引擎。基于百度云Elasticsearch团队在多种应用场景下所积累的丰富经验，提升集群稳定性、增强性能、优化成本和丰富功能。本文介绍BES的内核版本及重点优化的特性。

7.10.2

内核版本	发布说明
1.7.7	**bug修复** 1、修复了执行cloneapi过程中元数据丢失的问题 2、优化了向量索引写入流程，解决了内存使用超出限制时副本间数据不一致问题，优化了写入阻塞问题
1.7.6	**新增、优化特性** 1、提供慢查询隔离功能，降低异常查询对集群稳定性的影响
	新增、优化特性

1.7.5	1、新增了倒排融合查询的插件 2、优化了索引恢复流程，增大了索引恢复过程中simpleauth索引恢复的优先级
1.7.4	**bug修复** 1、修复了一个index包含多个同义词filter时的词典更新问题
1.7.3	**新增、优化特性** 1、优化了superuser和自定义用户的密码规则 • 新密码规则如下：8-32位字符，支持大小写字母、数字、特殊符号，必须至少包含其中三种，符号仅限!@#$%^*()_ • 原密码规则如下：支持小写字母(a-z)、大写字母(A-Z)、数字(0-9)、下划线(_)，并且至少包含其中三种, 长度8-12位 • 如果保持内核低于此版本，存量集群的密码规则不变
1.7.2	**新增、优化特性** 1、优化了hnsw算法带filter场景的实现，提高了该场景的召回率
1.7.1	**新增、优化特性** 1、优化了向量模块内存管理机制，当向量模块内存使用超出断路器限制后，不会再阻止向量数据写入
1.7.0	**新增、优化特性** 1、增加了基于段复制的主从同步机制 2、优化了使用量化算法的向量索引的带filter检索执行效率 **bug修复** 1、修复了部分队列任务阻塞的问题
1.6.3	**bug修复** 1、完善了cosine相似度算法实现对不同处理器的兼容性
1.6.2	**bug修复** 1、修复了script计算读取向量数据偶现的错误 2、优化了knn向量插件的实现，降低了gc消耗
1.6.1	**bug修复** 1、修复了大数据量下读取向量数据偶现的io错误
1.6.0	**新增、优化特性** 1、支持了通过SIMD指令加速script距离计算 2、优化了读取向量数据的内存消耗
1.5.1	**bug修复** 1、修复了hnsw索引带filter检索场景，在有nested字段的情况下影响召回的问题
1.5.0	**新增、优化特性** 1、优化source filter实现，提高_source解析效率 **bug修复** 1、完善了内核升级版本对mapping元数据的兼容性支持
1.4.1	**bug修复** 1、修复了稀疏向量存储实现导致的查询io错误
1.4.0	**新增、优化特性** 1、基于BOS冷热分离底层的文件读取类IndexInput采用mmap读取文件的方式实现。改变了冷热索引底层读取本地文件的方式，提升性能与稳定性 2、支持hnsw_sq8、hsnw_pq、ivf_hnsw、vf_hnsw_pq等索引算法，支持基于以上算法后台自动构建向量索引 3、hnsw支持通过mmap加载索引数据 4、优化了lucene的lz4的解压缩效率

1.3.0	**新增、优化向量特性** 1、上线基于自研引擎的hnsw算法 2、支持ivf、ivf_pq索引算法 3、支持基于hnsw的filter过滤能力，并通过智能修改查询计划得到更好性能表现 4、支持流式写入，优化向量写入内存消耗，优化后写入内存降低一倍 5、优化向量序列化格式，提高读写效率 6、优化forcemerge逻辑，支持直接merge到1个segment，不会有中间状态 **Bug修复** 1、修复hnsw写入阻塞问题和knn带 term filter 查询报数组越界错误的问题 2、修复shard的本地文件已存在导致shard启动错误的问题 3、修复向量功能should条件有不存在的字段的term检索时报错的问题
1.2.0	**优化特性** 1、 优化向量检索linear索引算法逻辑
1.1.0	**优化特性** 1、增加30秒延时释放句柄的逻辑 2、基于BOS冷热分离功能禁用序列化过程
1.0.0	**版本更新** 1、发布7.10.2版本 **优化特性** 1、升级了基于BOS冷热分离功能逻辑，保证了blk大小改动的情况下集群的稳定性 2、BOS冷热分离支持二级缓存功能 3、优化在blk size修改逻辑，修改后旧的缓存文件不会初始化到集群中 **Bug修复** 1、修复基于BOS冷热分离功能下，索引合并删除文件导致下次合并读取失败的问题

7.4.2

内核版本	发布说明
1.5.1	**优化特性** 1、优化了superuser和自定义用户的密码规则 ● 新密码规则如下：8-32位字符，支持大小写字母、数字、特殊符号，必须至少包含其中三种，符号仅限!@#$%^*()_ ● 原密码规则如下：支持小写字母(a-z)、大写字母(A-Z)、数字(0-9)、下划线(_)，并且至少包含其中三种, 长度8-12位 ● 如果保持内核低于此版本，存量集群的密码规则不变
1.5.0	**优化特性** 1、优化了TaskBatcher的并发性能，以解决高并发put mapping时导致的节点离群问题
1.4.0	**优化特性** 1、基于BOS冷热分离功能本地缓存清理优化 **Bug修复** 1、修复了删除数据由于解析不了导致metadata文件卡住的问题
1.3.0	**Bug修复** 1、修复了向量检索中linear检索导致fullgc的问题 2、基于BOS冷热分离功能的清理缓存机制，避免出现too many open files 3、修复log4j问题
1.2.0	**优化特性** 1、基于BOS冷热分离支持可观测功能 **Bug修复** 1、启动脚本中增加expression_policy的配置，防止使用painless等脚本时报错
1.1.0	**优化特性** 1、数据BOS迁移限速 2、bos_upload线程池粒度改为shard **Bug修复** 1、更新基于BOS冷热分离功能bug，热数据迁移不再通过BOS
1.0.0	**版本更新** 1、发布7.4.2版本 **Bug修复** 1、更新动态同义词插件bug，被删除的index的动态更新线程不会被停掉

6.8.14

内核版本	发布说明
1.1.1	**优化特性** 1、优化了superuser和自定义用户的密码规则 • 新密码规则如下：8-32位字符，支持大小写字母、数字、特殊符号，必须至少包含其中三种，符号仅限!@#$%^*()_ • 原密码规则如下：支持小写字母(a-z)、大写字母(A-Z)、数字(0-9)、下划线(_)，并且至少包含其中三种, 长度8-12位 • 如果保持内核低于此版本，存量集群的密码规则不变
1.1.0	**Bug修复** 1、修复log4j问题
1.0.0	**版本更新** 1、发布6.8.14版本

6.5.3

内核版本	发布说明
1.4.1	**优化特性** 1、优化了superuser和自定义用户的密码规则 • 新密码规则如下：8-32位字符，支持大小写字母、数字、特殊符号，必须至少包含其中三种，符号仅限!@#$%^*()_ • 原密码规则如下：支持小写字母(a-z)、大写字母(A-Z)、数字(0-9)、下划线(_)，并且至少包含其中三种, 长度8-12位 • 如果保持内核低于此版本，存量集群的密码规则不变
1.4.0	**优化特性** 1、6.5.3版本支持zstd压缩
1.3.0	**优化特性** 1、ik分词停用词加载授权
1.2.0	**Bug修复** 1、修复log4j问题
1.1.0	**优化特性** 1、NLP中文分词支持自定义词库
1.0.0	**版本更新** 1、发布6.5.3版本

集群列表

百度云Elasticsearch的集群列表展示了集群的名称/id、访问ip、运行状态、健康状态、所在网络、所在子网、所在可用区、标签、资源分组、版本、支付方式、创建时间、运行时间等信息，并提供了创建集群、集群配置变更、续费、计费变更、编辑资源分组、搜索集群、导出集群列表等功能入口。

列表说明

功能入口

登录 百度云Elasticsearch控制台，进入Elasticsearch的集群列表页面。集群列表页面展示了当前地域您账号下的所有百度云Elasticsearch集群，并提供了以下功能。

创建集群

点击左上角“创建集群”，进入集群创建页面。详细信息，请参见 集群创建。

刷新集群

点击右上角搜索栏右侧的刷新图标，可获取集群的实时状态。集群创建和配置变更后，需一段时间生效，状态为“生效中”， 可点击刷新图标查看实例的最新状态，当状态变为“运行中”，即可正常使用集群。

搜索集群

当集群太多时，用户可以通过搜索功能快速定位到目标集群。百度云Elasticsearch支持按照集群ID、集群名称和标签搜索集群。点击搜索栏左侧的下拉菜单选择需要的搜索类别，搜索时支持模糊匹配。资源分组搜索不支持模糊匹配，在下拉菜单中选择分组进行定位。

查看集群基本信息

点击集群名称进入集群详情页面，该页面展示了对应集群的基本信息，支付信息，网络信息和节点信息。详细信息，请参见[查看集群基本信息]。

实例配置变更

在“操作”列点击目标集群的“配置变更”按钮直接进入配置变更界面或点击集群名称进入集群详情页面，通过右上方的“配置变更”按钮进入。详情功能请参见 集群扩缩容。

删除集群

点击集群名称进入集群详情页面，点击右上方的“更多操作”按钮选择删除集群即可释放集群。详情信息，请参见 删除集群。

续费

在“操作”列点击目标集群的“续费”按钮进入集群续费页面（只有支付方式为后付费的集群可以实现续费功能）。续费时长可选“按年”和“按月”，并选择需要续费的年长或月长。点击下一步，确认订单信息无误后点击提交订单，选择付款方式付款，完成续费。

计费变更

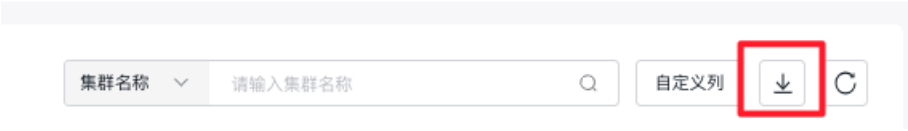
在“操作”列点击目标集群的“计费变更”按钮进入计费变更页面，详情信息，请参见 计费说明.

编辑资源分组

在“操作”列点击目标集群的“编辑资源分组”按钮弹出编辑资源分组的对话框。选中不同资源分组树下的子级分组，点击“确认”，添加目标集群至该资源分组中。一个目标集群可以属于多个资源分组树，但在一个资源分组树下只能属于其中的一个分支。如需删除原资源分组，最少保留一个资源分组。详情信息请参见资源管理。

导出集群列表

鼠标放到右上方的下载的icon按钮，会气泡提示“导出列表”，点击此按钮可以将用户该地区的集群列表全部导出为Excel表格，导出表格中的字段和列表中字段一致



YML参数配置

概述

用户可以通过百度智能云Elasticsearch的配置修改功能，修改常用的参数配置。

操作步骤

1. 登录百度智能云Elasticsearch控制台。
2. 点击集群名称/ID进入目标集群。在左侧导航栏，选择配置修改进入Elasticsearch YML文件配置页面。
3. 在Elasticsearch YML文件配置的框中以示例格式输入，然后点击确定，即可完成YML参数的修改。

参数修改示例：

action.destructive_requires_name: true

4. 回到集群详情页面，手动重启集群，重启完成后，新的配置参数将生效。

支持参数

查看elasticsearch.yml参数的内容，可登录Kibana控制台执行以下命令

```
GET /_cluster/settings?include_defaults&flat_settings=true
```

配置索引 | 参数 | 默认值 | 说明 | | --- | --- | --- | | action.auto_create_index | true | 自动创建索引 当Elasticsearch集群接收到新文档后，如果没有对应索引，是否允许系统自动创建索引。true: 启用。允许系统自动创建索引。false: 不启用。禁止创建索引。 | | action.destructive_requires_name | false | 在删除索引时是否需要明确指定索引名称。默认删除或关闭时索引名称时不需要明确指定索引名称，支持用通配符进行批量删除索引。如果需要删除时明确指定索引名称，请将false改为true | **配置CORS访问**

参数	默认值	说明
http.cors.allow-credentials	false	Elasticsearch是否允许其他域资源下的浏览器向其发送请求： true：启用。Elasticsearch会处理OPTIONS CORS请求。如果发送请求中的域信息已在http.cors.allow-origin中声明，那么Elasticsearch会在头信息中附加Access-Control-Allow-Origin，以响应跨域请求。 false：不启用。Elasticsearch会忽略请求头中的域信息，将不会使用Access-Control-Allow-Origin信息头应答。如果客户端不支持发送附加域信息头的preflight请求，或者不校验从服务端返回的报文的头信息中的Access-Control-Allow-Origin信息，那么跨域安全访问将受到影响。如果关闭CORS支持，则客户端只能尝试通过发送OPTIONS请求，以了解此响应信息是否存在。
http.cors.allow-headers	*	请求头信息配置项。
http.cors.allow-origin	*	域资源配置项，可设置接受来自哪些域名的请求。
http.cors.enabled	true	凭证信息配置项目，设置是否允许响应头中返回Access-Control-Allow-Credentials信息： true：允许 false：不允许
http.cors.max-age	1728000	浏览器可发送OPTIONS请求以获取CORS配置信息，此配置项可设置获取的信息在浏览器中的缓存时间，单位为秒。
http.cors.allow-methods	OPTIONS,HEAD,GET,POST,PUT,DELETE	请求方法配置项。

配置reindex白名单 通过当前集群调用reindex API，从远程集群迁移索引数据前，需要先配置reindex白名单。用户可以在YML文件配置中，配置reindex白名单，支持配置的参数如下。

参数	默认值	说明
reindex.remote.whitelist	[]	设置远程Elasticsearch集群的访问地址，将其添加到当前集群的远程访问白名单中。配置语法：reindex.remote.whitelist: {IP}:{port}。

配置queue大小 通过自定义queue大小，调整文档写入和搜索的队列大小。用户可以在YML文件配置中，配置queue大小。以下示例配置文档写入和搜索queue大小为500和1000，实际业务中请根据具体情况自行调整。

参数	默认值	说明
thread_pool.search.queue_size	1000	文档搜索队列大小，默认的文档搜索队列最大最小都是1000.
thread_pool.write.queue_size	5000	文档写入队列大小

日志查询

日志查询（新）

概述

通过BES的日志功能，用户可以输入关键字和设置时间范围，锁定需要查询的日志内容，快速定位集群问题，辅助集群运维。

使用限制

1. 仅华北-北京、华北-保定、华南-广州、华东-苏州的7.x版本实例支持在控制台查看日志。

2. 存量集群

- 未开启日志查看功能的存量集群

  用户点击日志管理功能，直接显示更新后的新页面。

- 对于开启日志查看功能的存量集群

  用户点击左侧导航栏的日志管理后，会显示更新后的页面，但用户仍然可以在Kibana，BES中查询日志，左上方可以关闭日志查看按钮。关闭后，该按钮，文字和icon消失，无法再开启老功能，新产生的日志将不会再记录到BES中。若用户没有手动关闭，该功能将三个月后强制关闭。

操作步骤

1. 登录BES控制台。进入目标集群，单击左侧导航栏日志管理，查看集群的运行日志。

2. BES主要支持的日志类型包括：主日志、查询慢日志、索引慢日志、GC日志、ES访问日志。各类日志的说明和使用场景如下。

日志类别	说明	使用场景
主日志	集群的健康运行状态及索引查询写入日志。	当用户需要查看集群中各节点的运行状况及查询写入情况，例如节点之间的连通性、Full GC情况、创建或删除索引情况、集群级别的查询报错等信息时，可查看主日志进行排查。
查询慢日志	慢查询日志。当查询耗时超过指定阈值时，将在慢查询日志中打印相关信息。	当用户的业务出现查询耗时久的情况时，可查看查询慢日志进行排查。查询耗时越久，集群资源消耗越大。当日志存在大量的慢日志，请排查集群资源及负载情况，获取瓶颈项，根据瓶颈项及时扩充对应资源或使用集群限流插件（baidu-rate-limiting）进行限流，以保证集群的稳定性。
索引慢日志	慢写入日志。当写入耗时超过指定阈值时，将在慢写入日志中打印相关信息。	当用户的业务出现写入耗时久的情况时，可查看索引慢日志进行排查。写入耗时越久，对集群的资源消耗越大。当日志存在大量的慢日志，请排查集群资源及负载情况，获取瓶颈项，根据瓶颈项及时扩充对应资源或集群限流插件（baidu-rate-limiting）进行限流，以保证集群的稳定性。
GC日志	垃圾回收器日志。显示所有JVM堆内存占用触发的垃圾回收情况，通过GC日志可获取详细的垃圾回收信息。	当集群出现性能瓶颈时，可通过GC日志获取详细的GC回收信息，查看是否存在耗时长或操作频繁的GC。如果存在，需要及时扩充集群资源或使用集群限流插件（baidu-rate-limiting）进行限流，以保证集群的稳定性。
ES访问日志	集群的访问日志。显示Elasticsearch集群接收到DSL查询、写入、bug等所有请求访问ES的详细信息。	当用户需要排查哪些客户端在向Elasticsearch集群发送查询、写入等请求时，可通过ES访问日志获取。

3. 在日志页面的搜索框中，输入查询条件，选择开始时间和结束时间，单击搜索。日志查询只支持使用match语法，可点击搜索框右边的icon，查询更多语法规则。

注：BES 默认提供集群从当前时间点为止近7天内的运行日志，且最大支持返回10000条日志，按时间倒序展示。

4. 查询慢日志、索引慢日志、ES访问日志功能默认关闭，用户如需查看查看相关日志，请点击右上角日志设置开启相关日志功能。

注：默认关闭的日志 开启后才开始记录日志信息。

日志查询（旧）

百度智能云Elasticsearch支持配置集群日志采集及日志定期清理，支持的日志类型包括索引慢日志及查询慢日志。

> 注意：目前仅支持的ES版本为7.4.2+的集群进行配置使用

集群日志配置

1. 登录百度智能云Elasticsearch控制台，并点击集群名称，进入集群详情界面。
2. 在左侧导航栏，单击**日志管理**。

3. 点击开启日志查看功能开关，并进行日志定期清理配置，目前可配置1~7天的保留时间，集群日志超过保留时间后会被自动清理。

4. 完成配置后集群将会收集索引及查询慢日志存储到您的ES集群中，生成前缀为 filebeat-7.4.2-* 的索引。

配置开启慢日志

默认情况，慢日志是关闭的。开启慢日志需要日志记录等级（INFO、WARN、DEBUG 等），定义义具体动作（query、fetch 或 index）以及时间阈值。

举例如下：

1. 配置集群慢日志级别为INFO

```
PUT /_cluster/settings
{
  "transient" : {
    "logger.index.search.slowlog" : "INFO",
    "logger.index.indexing.slowlog" : "INFO"
  }
}
```

2. 配置单个index索引慢日志时间阈值

```
PUT /{index_name}/_settings
{
  "index.indexing.slowlog.threshold.index.debug": "10ms",
  "index.indexing.slowlog.threshold.index.info": "50ms",
  "index.indexing.slowlog.threshold.index.warn": "100ms"
}
```

3. 配置所有日志索引慢日志时间阈值

```
PUT /_all/_settings
{
  "index.indexing.slowlog.threshold.index.debug": "10ms",
  "index.indexing.slowlog.threshold.index.info": "50ms",
  "index.indexing.slowlog.threshold.index.warn": "100ms"
}
```

集群日志查询

此处展示通过Kibana查看的方式，您也可以通过连接ES自行查看

登录该集群的Kibana，找到 filebeat-7.4.2-* 索引，这里查看的是索引慢日志

```
health status index                            uuid                   pri rep docs.count docs.deleted store.size pri.store.size
green  open   filebeat-7.4.2-slowlog-2021.01.06 GKItlPCaS2i8UwdXFsG8Jg   1   0          1            0     17.3kb         17.3kb
green  open   simpleauth                       E801jggHRTeFFkfhuhrCzg   1   0          1            0      4.7kb          4.7kb
green  open   .kibana_1                        2zd8IdkKQ16vssRe3T8fTA   1   0          1            0      3.7kb          3.7kb
green  open   .bes_schedules                    97UsGGRjQqyez8pBONPZYA   1   0          1            0      7.5kb          7.5kb
```

直接通过search api方式查询 例如：查询世界时间范围为 2021-01-06 8点到9点的慢查询日志

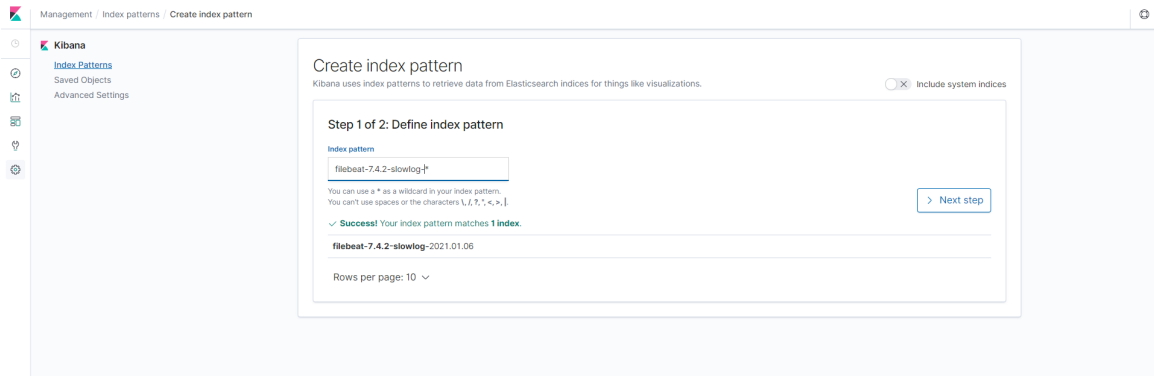
```
GET /filebeat-7.4.2-slowlog-2021.01.06/_search
{
  "query": {
    "range": {
      "@timestamp": {
        "gte": "2021-01-06T08:00:00Z",
        "lte": "2021-01-06T09:00:00Z"
      }
    }
  }
}
```


```
    {
      "_index" : "filebeat-7.4.2-slowlog-2021.01.06",
      "_type" : "_doc",
      "_id" : "lq3H1nYBHnbKINBlRYee",
      "_score" : 1.0,
      "_source" : {
        "took_millis" : "3",
        "shard_num" : "1",
        "query_dsl" : "{}",
        "shard" : "0",
        "took" : "3ms",
        "level" : "INFO",
        "index" : "kkkk",
        "message" : "[2021-01-06T16:19:04,911][INFO ][index.search.slowlog.fetch] [466879576610574336-192.168.6.169][kkkk][0] took[3ms], took_millis[3], total_hits[1 hits], types[],
          stats[], search_type[QUERY_THEN_FETCH], total_shards[1], source[{}], id[], ",
        "search_type" : "QUERY_THEN_FETCH",
        "hits" : "1",
        "node" : "466879576610574336-192.168.6.169",
        "@timestamp" : "2021-01-06T08:19:06.410Z",
        "time" : "2021-01-06T16:19:04,911",
        "fields" : {
          "type" : "slowlog"
        }
      }
    }
  ]
}
}
```

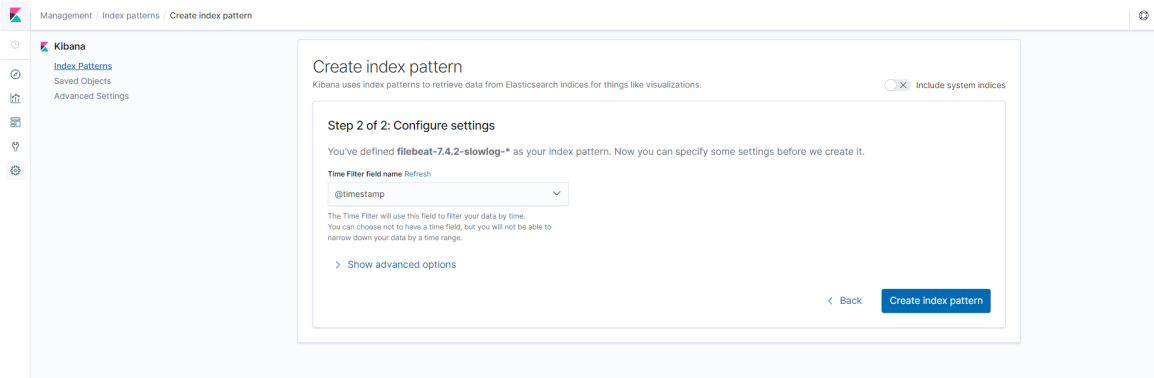

使用仪表盘展示数据

以集群慢查询日志为例，使用Kibana把数据在仪表盘中展示:

创建索引模式Index Patterns

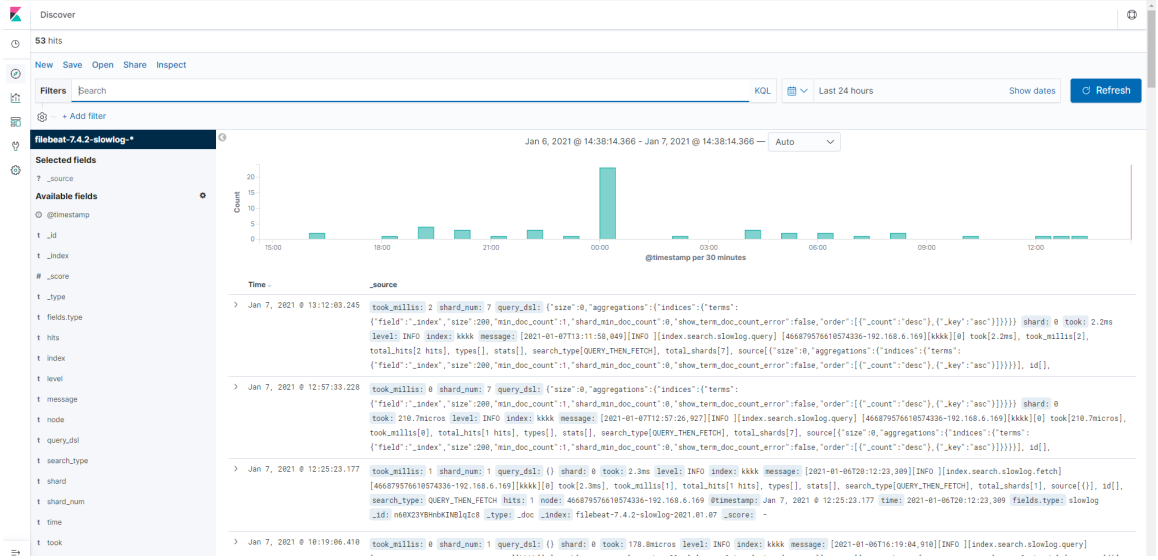


选择@timestamp为Time Filter field，Time Filter将使用此字段按时间筛选数据



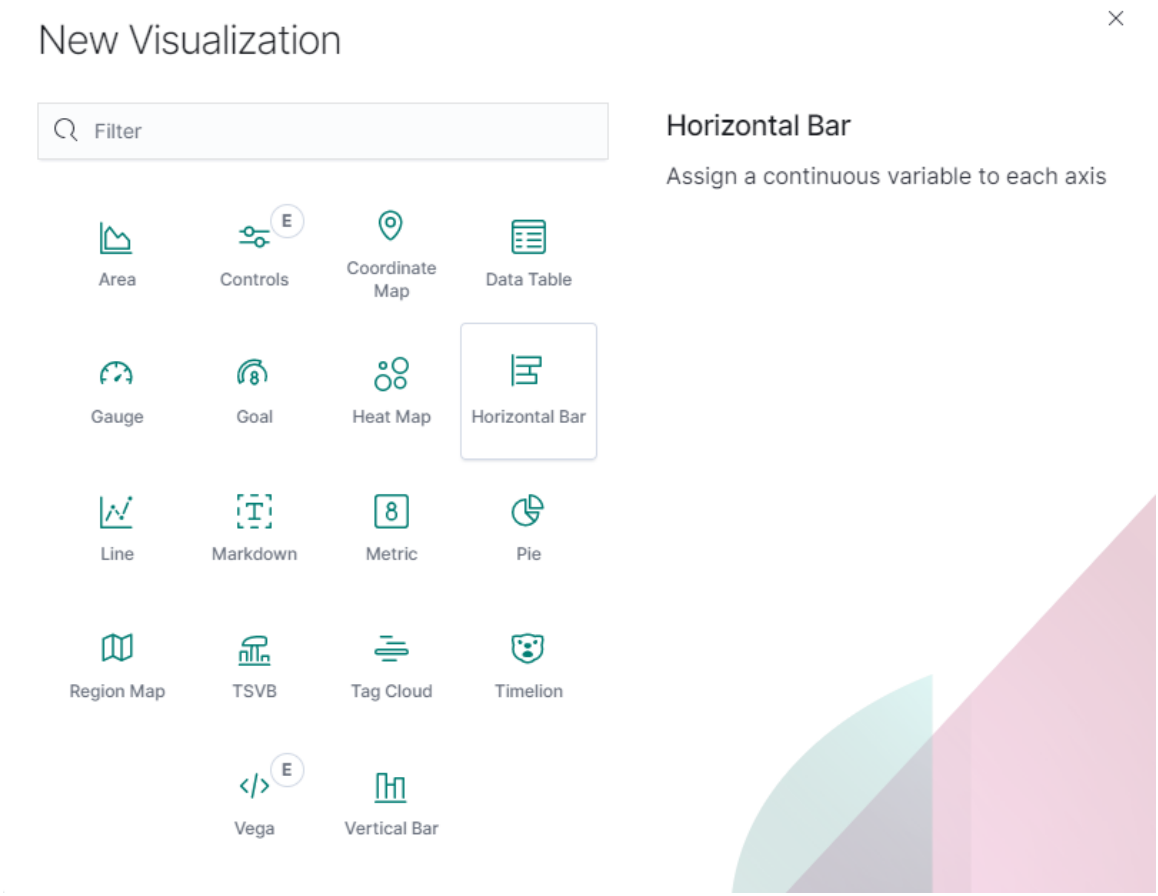
使用Discover检索数据

单击侧面导航中的 Discover 进入 Kibana 的数据探索功能：

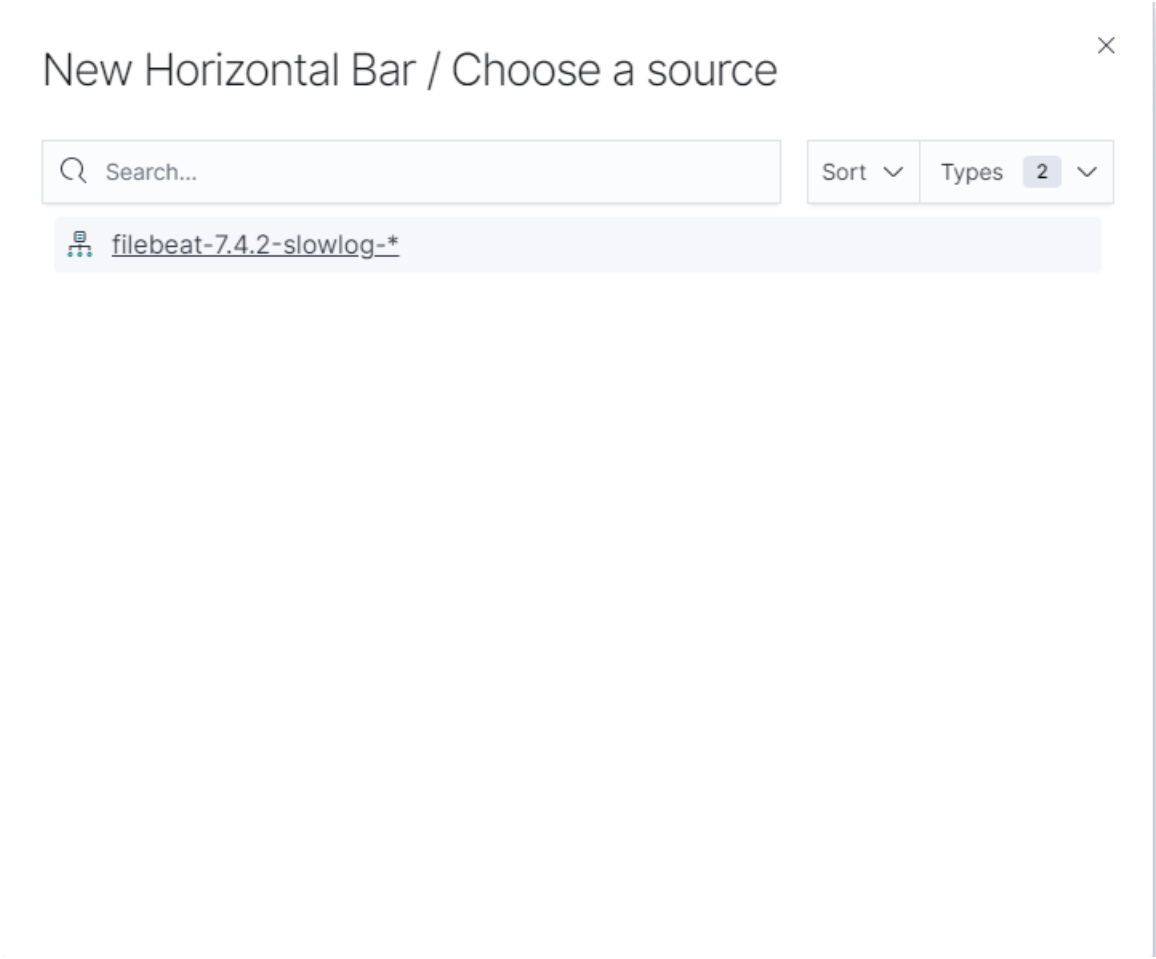


创建Horizontal Bar（水平条形图）

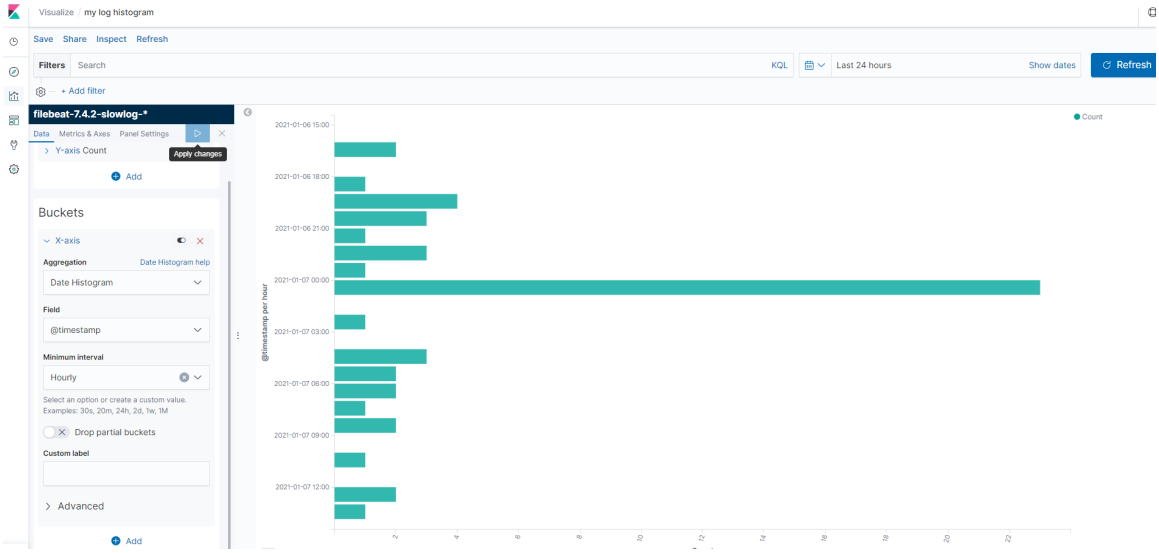
1. 在 New Visualization 页面中选择 Horizontal Bar 选项；



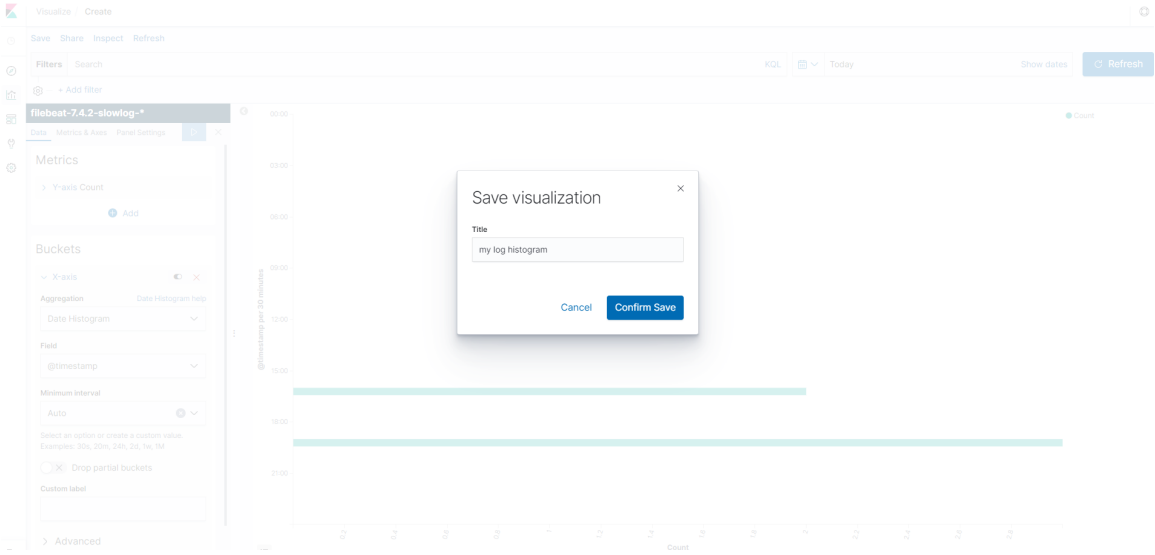
2. 选择配置的Index Pattern；

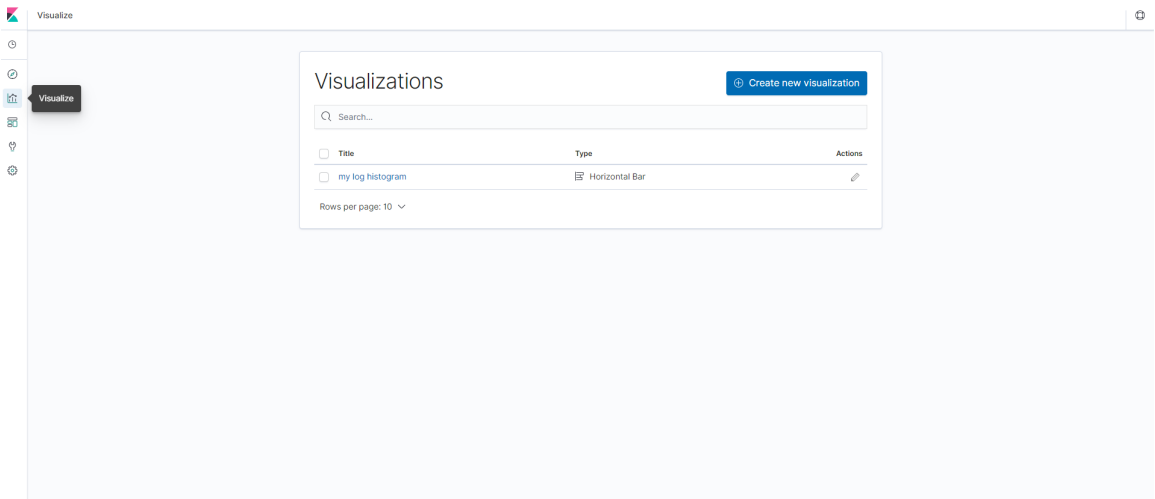


3. 如图，可以清楚的看出不同的时间内慢日志条数之间的对比，有助于排查慢查询问题。 例如此处可以清晰的看出查询超时大多集中在 1月7日 0点 之后



制作完成后，用户可以点击页面左上角 Save 然后输入名称 （如图示例my log histogram） 来保存这个图表供以后使用。





更多Kibana使用可参考Kibana官方文档

查询语法

日志查询功能，关键词查询的语法关键词是match，检索语句写在 match 关键字之后，和 match 关键字之间通过空格分隔。形如： `match 检索语句`

支持的查询类型

- **数值查询：** 指定字段和查询条件，所指定字段符合条件的日志会作为结果返回。例如 `match status=200 and time_cost in [100 200)` ，表示查询 status 等于 200 且 timecost 大于等于 100 且 timecost 小于 200 的日志
- **关键字查询：** 指定字段和关键字，所指定字段包含该关键字的日志会作为结果返回。如果不指定字段，将对日志原文进行查询。如果需要查询的关键字中包含检索语法运算符或空格，可以将关键字用双引号（""）包裹，表示将双引号中的内容作为多个关键字查询。例如 `match method:GET and msg:"service not avaliable"` 表示查询 method 是 GET 并且 msg 包含关键字 service、not 和 avaliable 的日志，等价于查询 `match method:GET and msg:service and msg:"not" and msg:avaliable`
- **短语查询：** 如果需要查询特定的关键字组合，可以将关键字用单引号('')包裹作为短语，表示按照单引号中的关键字顺序进行匹配查询。如果不指定字段，将对日志原文进行查询。例如 `match msg:'service not avaliable'`表示查询 msg 包含短语 service not avaliable 的日志，且组成短语的关键字之前不包括其他关键字, service is temporarily not avaliable 和 not service avaliable 都不会命中查询
- **模糊查询：** 在关键字中的中间或末尾加上模糊查询关键字，即 `*` 和`?`，日志服务会在所有日志中为您查询符合条件的日志。例如`match httpuseragent:andr*` 表示在所有日志中查找 http_user_agent 字段包含以 andr 开头的词的日志

限制说明：

- 查询时必须指定前缀，即 * 和 ? 不能出现在词的开头

- 指定的词越精确，查询结果越精确。

字段对应的查询类型

- 对于 text 类型的字段，可以进行普通查询、短语查询、模糊查询
- 对于 float 和 long 类型的字段，可以指定数值范围进行查询

例如，查询语句为 `match method:GET and status >= 500`，表示查询 method 字段值是 GET，且 status 字段值大于等于 500 的日志

运算符简介

运算符	说明
and	双目运算符。格式为 query1 and query2，表示 query1 和 query2 查询结果的交集。如果多个单词间没有语法关键词，默认是 and 的关系。
or	双目运算符。格式为 query1 or query2，表示 query1 和 query2 查询结果的并集。
not	双目运算符。格式为 query1 not query2，那么表示符合 query1 但不符合 query2 的结果，即 query1 - query2。如果省略 query1 ，只有 not query2 ，表示从全部日志中选取不包含 query2 的结果。
(,)	括号用于把一个或多个子查询合并成一个查询条件，用于提高括号内查询条件的优先级。
:	用于 key-value 对的查询。key:term 构成一个 key-value 对。表示在 key 的索引上执行 term 查询。如果 key 或者 value 内有空格（ ）、冒号（:）等保留字符，需要用双引号 "" 把整个 key 或者 value 包裹起来。左右引号内部的任何一个 term 都会被查询，而不会当成语法关键词。
*	模糊查询关键字，用于替代 0 个或多个字符，例如：andr*，会返回 andr 开头的所有命中词。
?	模糊查询关键字，用于替代一个字符，例如 andr?id，会返回以 andr 开头，以 id 结尾，并且中间还有一个字符的所有命中的词。
"	转义符号，表示把一个关键词转换成普通的查询字符。左右引号内部的任何一个 term 都会被查询，而不会当成语法关键词。引号内的模糊查询关键字 (*,?) 会被当做字符常量处理。
'	转义符号，表示关键字短语。左右引号内部的内容会被当做一个关键字短语进行短语查询。引号内的模糊查询关键字 (*,?) 会被当做字符常量处理。
\	转义符。用于转义引号，转义后的引号表示符号本身，不会当成转义字符，例如 "\""。
>	查询某个字段下大于某个数值的日志，例如 status > 500。
>=	查询某个字段下大于或等于某个数值的日志，例如 status >= 500。
<	查询某个字段下小于某个数值的日志，例如 status < 400。
<=	查询某个字段下小于或等于某个数值的日志，例如 status <= 400。
=	查询某个字段下等于某个数值的日志，例如 status = 404。
in	查询某个字段处于某个范围内的日志，使用中括号表示闭区间，使用小括号表示开区间，括号中间使用两个数字，数字中间为若干个空格。例如 status in [500 600] 或 status in [200 400)。
is null	查询某个字段值为空或不包含该字段的日志。例如 error is null。
is not null	查询某个字段值不为空的日志。例如 error is not null。
is true	查询某个布尔类型的字段值为 true 的日志，例如 success is true。
is false	查询某个布尔类型的字段值为 false 的日志，例如 success is false。
is not true	查询某个布尔类型字段值不为 true 的日志。需要注意，若字段内容为空或日志中没有该字段，则默认字段值为null，满足字段值不为 true 的条件。例如 success is not true。如果需要查询有 success 字段，且值不为 true 的日志，可以使用 success is not null and success is not true 进行查询。
is not false	查询某个布尔类型字段值不为 false 的日志。需要注意，若字段内容为空或日志中没有该字段，则默认字段值为 null，满足字段值不为 false 的条件。例如 success is not false。

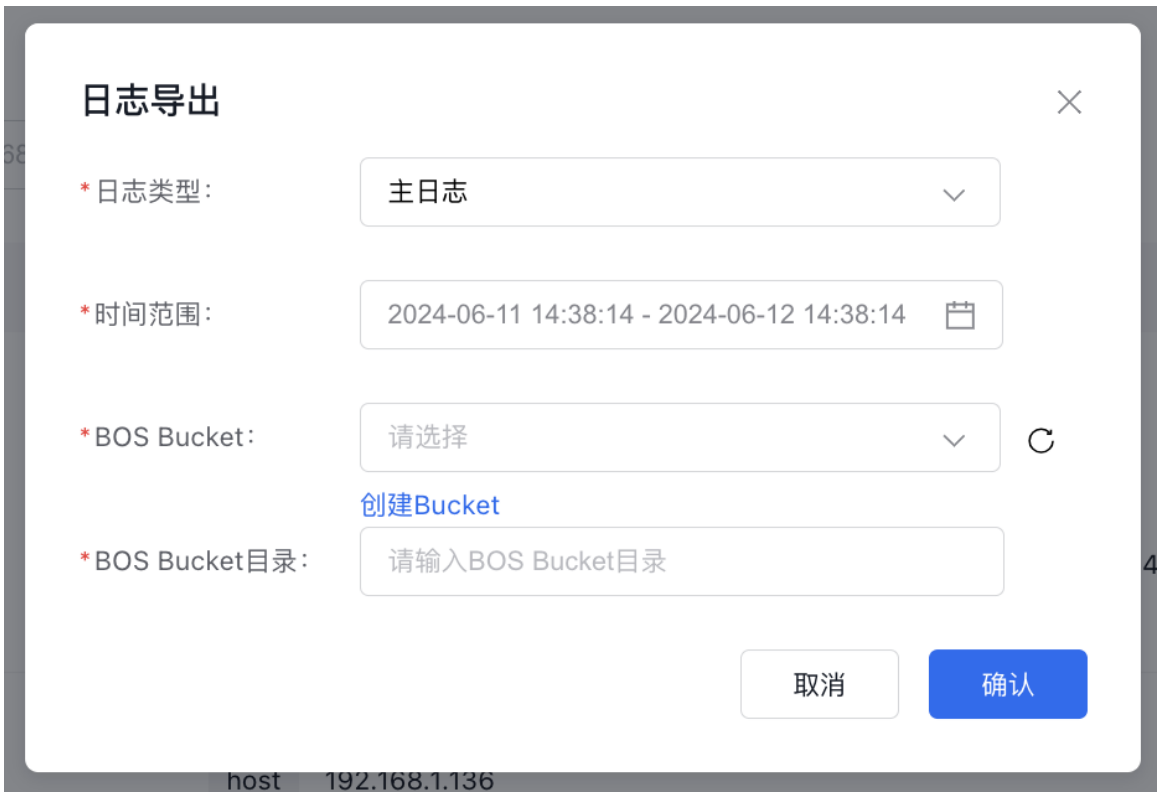
日志导出

概述 百度智能云Elasticsearch支持将日志数据存到BOS Bucket，通过BOS下载数据，进一步分析集群运行情况。

操作步骤

1. 登陆百度智能云Elasticsearch控制台。
2. 进入集群列表页面，点击目标集群，单击左侧导航栏日志管理。
3. 点击日志导出。

任务设置 每一次日志导出任务，需要填写日志类型、日志时间范围、BOS Bucket以及BOS Bucket目录。



日志类型 百度智能云Elasticsearch支持导出集群的主日志，查询慢日志、索引慢日志、GC日志和ES访问日志。具体日志信息详情查看日志查询。

> **注意：**
>
> 1. 每次日志导出任务支持选择多个日志类型。
> 2. 如果未开启查询慢日志、索引慢日志和ES访问日志，则无法选择导出该类日志信息。开启日志详情查看日志查询。

时间范围 日志导出任务支持导出最近7天的日志，默认选中最近1天的日志。

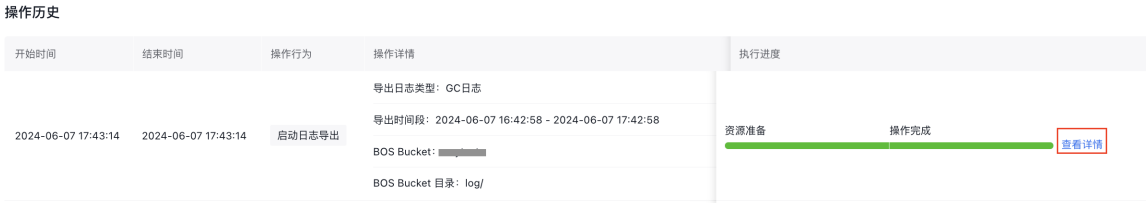
BOS Bucket 您可以通过下拉选择已有的BOS Bucket。下拉列表中也提供搜索框，支持根据输入的内容进行模糊搜索，在搜索的结果中进行选择。

若您暂未开通BOS，下拉列表中无法提供对应选项，可以通过输入框下方的“创建Bucket”按钮进入BOS产品页面进行开通和创建。开通BOS详情查看开始使用BOS。

BOS Bucket目录 您可以自定义编辑目录，目录名称支持1-63个字符，不支持输入连续斜杠，例如//。默认目录为“bes_log_repository/集群名称_集群id”。

任务进度 完成任务配置后点击确认，会自动生成导出任务。

如需查看任务进度，可跳转至操作历史页面查看。操作历史中会生成一条新的记录，为避免影响其他操作的进度，该条记录只会记录“开启日志导出”任务进度。如需查看导出的详细进度，可以点击进度条后方“查看详情”按钮。



导出进度窗口会展示日志导出的实际进度。

注意事项

1. 日志导出功能目前只支持7.x版本。
2. 导出任务一旦创建，无法中断或删除。如果想删除数据，可以通过BOS找到对应Bucket操作。
3. 日志导出操作一旦开始，无法暂停。

智能巡检

智能巡检是百度智能云Elasticsearch提供的自动化运维工具，提供集群、节点、索引维度的多个巡检项。自动归纳巡检结果生成报告，帮助用户持续分析集群状态，监测集群潜在风险，高效定位集群问题，并提供优化建议，提升运维效率

开启智能巡检

智能巡检需要访问用户集群的基本信息和日志，因此当首次启用智能巡检服务时，需要用户对访问授权

操作步骤

1. 登录百度智能云Elasticsearch控制台，在集群列表中单击需要开启智能巡检的**集群ID/名称**，进入集群详情页
2. 在集群详情页，侧边栏点击**智能巡检**，进入智能巡检功能，开始初始化流程
3. 在初始化页面中，阅读相关说明和注意事项

4. 确认无误后，点击【授权】按钮，即可开启智能巡检功能

查看巡检概览

巡检概览统计并展示了集群的巡检结果及其趋势变化，方便用户查看集群近期的健康状况。巡检结果通过高风险、低风险、安全和失败，一共4种状态来展示各巡检项的结果：

高风险：表示集群已经出现了严重的问题或隐患，已经影响集群可用性，需要立即处理，否则会导致数据丢失、集群故障等问题

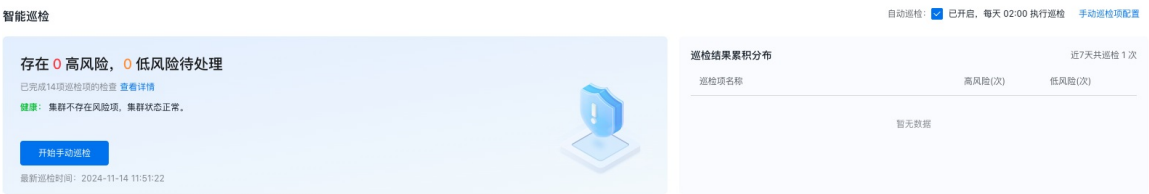
低风险：表示集群存在较严重的问题或隐患，可能会影响集群可用性，建议尽快处理

安全：表示集群健康

失败：表示巡检项巡检失败，建议用户检查对应索引或提工单咨询

操作步骤

1. 登录百度智能云Elasticsearch控制台，在集群列表中单击需要开启智能巡检的**集群ID/名称**，进入集群详情页
2. 在集群详情页，侧边栏点击**智能巡检**，进入智能巡检功能。页面上方展示了最新巡检结果摘要和近7天巡检结果累计分布
   - 最新巡检结果摘要：会展示最近一次的集群巡检结果的关键指标，包括最近一次巡检时间、高风险巡检项个数、低风险巡检项个数等
   - 近7天巡检结果累计分布：展示最近7天内全部巡检次数累计的巡检结果，帮助用户判断巡检项的健康状况



巡检集群

开启智能巡检功能后，支持巡检集群，并产生巡检报告。支持自动巡检和手动巡检，用户可选择需要巡检的索引和巡检项。每次巡检约耗时约几分钟

- 自动巡检：系统将在每天凌晨2点进行巡检，默认选择全部巡检项并生成巡检报告，每天一次
- 手动巡检：除了每天的自动巡检外，支持手动随时触发巡检任务，用户可以指定部分巡检项和索引，并立即生成巡检报告，无次数限制

操作步骤

1. 登录百度智能云Elasticsearch控制台，在集群列表中单击需要开启智能巡检的**集群ID/名称**，进入集群详情页
2. 在集群详情页，侧边栏点击**智能巡检**，进入智能巡检功能。自动巡检和手动巡检的操作步骤将分别介绍

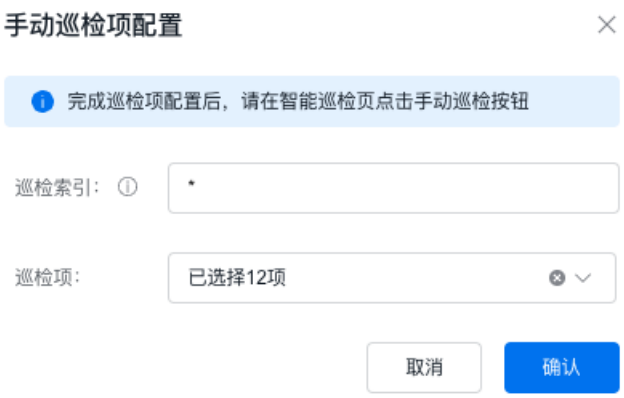
自动巡检

智能巡检页面右上角提供自动巡检开关，智能巡检初始化完成后，自动巡检功能默认开启，用户可勾选关闭

手动巡检

智能巡检页面的最新巡检结果摘要模块，提供手动巡检按钮，点击后即刻进行手动巡检，生成巡检报告

另外手动巡检支持用户配置巡检项和索引，用户可配置业务上关注重点的巡检项和索引。功能入口在智能巡检页面右上角-**手动巡检项配置**，单击后在弹窗中配置。手动巡检项配置完成后，点击**确认**，巡检项的配置将被保存。再点击**开始手动巡检**，配置的巡检项和索引会在手动巡检生成的报告中生效



查看巡检报告

巡检报告中包含巡检项说明、巡检结果、巡检建议以及详情，手动巡检报告内容由用户选择的巡检项构成。最多支持查看最近7天的历史巡检报告。

操作步骤

1. 登录百度智能云Elasticsearch控制台，在集群列表中单击需要开启智能巡检的**集群ID/名称**，进入集群详情页
2. 在集群详情页，侧边栏点击**智能巡检**，进入智能巡检功能。
3. 在智能巡检页面下方的**巡检报告**模块，提供报告列表功能，可以查看不同日期的历史报告。在列表中点击**巡检报告名称**或**操作**列的**查看报告**，在右侧查看对应的历史巡检报告

巡检报告支持下载，在巡检报告右上角点击**下载报告**，支持下载巡检报告pdf版

关闭智能巡检

当用户不再需要使用智能巡检功能时，可关闭该服务，关闭后，系统将不会再定时巡检集群并生成新的巡检报告

操作步骤

1. 登录百度智能云Elasticsearch控制台，在集群列表中单击需要开启智能巡检的**集群ID/名称**，进入集群详情页
2. 在集群详情页，侧边栏点击**智能巡检**，进入智能巡检功能。
3. 智能巡检页面右上角提供**自动巡检**开关，用户勾选关闭后，功能关闭

Logstash

实例管理

创建实例

前提条件

1、已创建百度智能云账号。创建账号请参考百度云账号注册流程

2、创建Logstash实例之前必须完成实名认证。请参考实名认证操作方法完成认证。

功能入口

入口一：通过百度智能云Elasticsearch控制台，进入Logstash 实例列表页。

点击左上角【新建】按钮，进入实例创建页。

入口二：在Elasticsearch集群创建页，点击“新建Elasticsearch集群”右边的下拉箭头，选择“新建Logstash实例”进入Logstash新建流程。

操作步骤

接下来将具体说明创建实例流程

1. 实例配置

详细说明创建实例中的配置项，包括付费及地域、基础配置、网络配置、节点配置和其他配置。

- 付费及地域

  1)当前地域：目前百度智能云Logstash已在四个地域上线，灰色部分不可选择。

  2)可用区：用户可根据需求进行可用区选择，不同区域包含的可用区数量可能不同，当一个可用区出现故障后，不会影响其他可用区的使用，保护用户的应用程序或数据库不受单一位置故障影响，并且可用区能提供比单个数据中心更强的可用性、容错能力以及可扩展性。

  3)付费方式：

预付费：一般为包年包月的购买形式，先付费后使用服务。用户根据自身对云资源的使用需求，一次性支付一个月、多个月或多年的费用，支付成功后，云资源将被系统分配给用户使用，直到到期后用户未继续续费而被回收。预付费模式价格相较于后付费模式更低。

后付费：按量计费，即先按需申请资源使用，结算时会按用户的实际资源使用量和使用周期收取费用。结算时如果账户余额不足，客户账号将进入欠费状态，需要在约定时间完成缴费，超过约定时间，所使用资源将被关停并收回资源。后付费模式按分钟计费，按小时扣费。不需要提前支付费用，但需要保证账户余额大于100元。后付费比预付费模式价格高一些。

- 基础配置

  1)Logstash版本：目前提供7.10.2和7.4.2两种版本选择，默认为7.10.2版本。建议和BES版本选择相对应。

  2)实例名称：用户自定义，长度为1~50个字符，以大小字母、数字或中文开头，可包含大小字母、数字、中文、下划线（_）或连字符（-）。

- 网络配置

网络类型： Logstash部署在私有网络VPC内，只有在同一个 VPC 的同一个安全组内才能访问Logstash实例，因此为保证内网顺利访问，建议选择已有云上业务的区域位置所在 VPC。选择对应可用区的子网，子网创建详见子网。若需要将 Logstash 实例与Elasticsearch集群搭配使用，请选择 Elasticsearch集群所在的网络。VPC创建详见私有网络VPC。

- 节点配置（类型，节点配置，数据盘，购买数量）

  1)类型：Logstash目前只提供Logstash节点。

  2)节点配置：目前支持通用型、计算型，用户可根据业务场景进行选择不同规格。

  3)数据盘：用户可选高性能云盘和SSD云盘，并根据业务需要配置资源量。

  4)购买数量：默认选中1个Logstash节点，用户可根据业务需求调整购买数量。

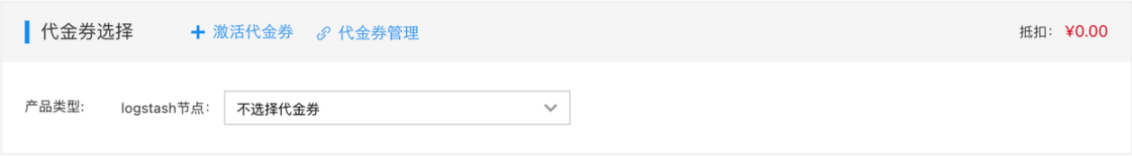
- 其他配置（购买时长，自动续费）

  1)标签：标签支持按各种标准（如用途、所有者或项目）对资源进行分类；每个标签包含键和值两部分。

  2)购买时长：预付费模式可选择购买时长。默认购买时长为1个月，可选：1~9月、1~3年。预付费模式支持设置自动续费功能（默认关闭），可在购买页面通过勾选“到期自动续费开启“。更多功能说明详见自动续费。

2. 确认订单

点击“下一步”，进入订单确认页面，用户需要确认一下订单价格，若选择预付款则会出现“代金券选择”，如果有代金券在此处添加代金券。



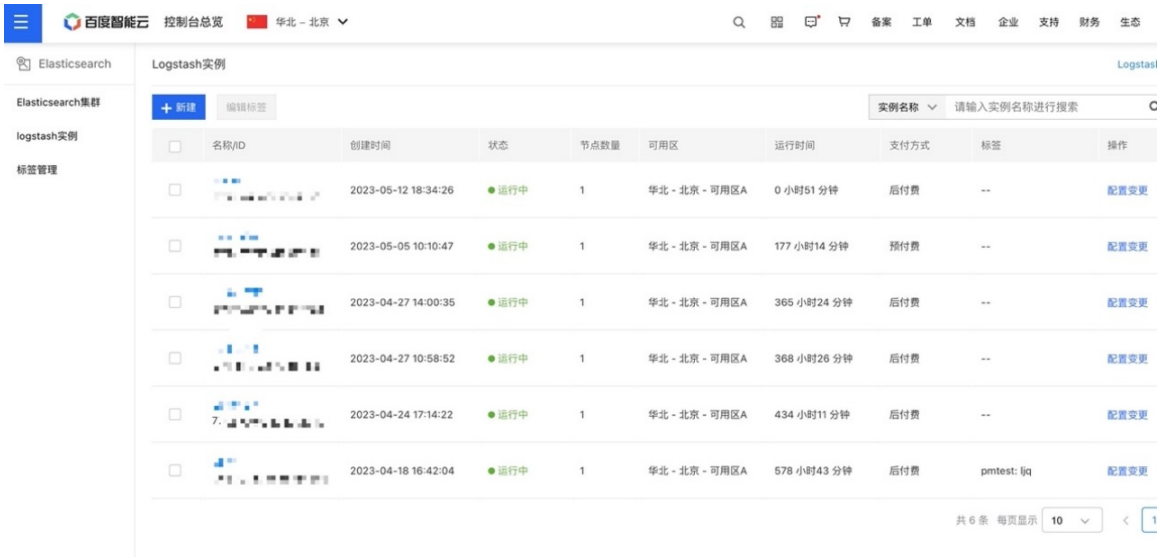
若选择后付款，如果用户有代金券，产生账单后会优先从可用代金券中扣除；如果有折扣，会在实际产生的账单中体现。

3. 在线支付

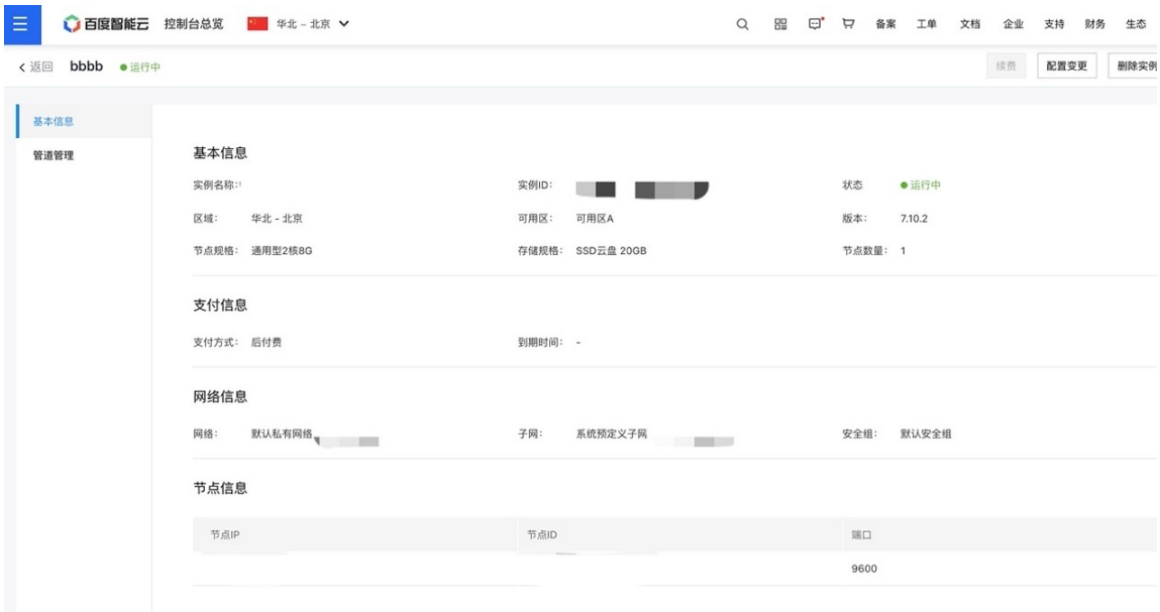
点击“提交订单”后，进入下一步：

- 预付费方式，点击“提交订单”后进入订单支付页，核对订单并选择支付方式，确认无误后点击“确认支付”进行支持。返回 ES 控制台，点击“logstash实例”在实例列表页可以查看到创建的实例。
- 后付费方式，点击“提交订单”后直接创建实例，不需要确认订单支付。在使用过程中，会按小时进行计费及扣费。

4. 后续操作 大约1分钟后，实例创建工作完成，点击“管理控制台”即可进入默认集群管理界面，点击“logstash实例”即可进入实例管理界面：



点击实例名称，可以查看实例的详细信息：



可以通过控制台对实例进行各种管理操作，包括配置变更、删除实例等。

实例详情信息

概述

实例详情页面包含实例基本信息、支付信息、网路信息和节点信息。当用户需要访问实例的节点地址、端口号、版本等信息时，可进入实例详情页面获取。

基本信息

1．实例名称：需用户自定义实例名称。实例名称可被用来实现实例搜索功能。

2．实例ID：实例唯一标识，系统自动生成，实例ID可被用来实现实例搜索功能。

3．状态：实例的运行状态。

- “运行中”（绿色）表示实例正常运行。
- “生效中”（黄色）包括配置变更、重启中、重启中和删除中几种情况。
- “已欠费“（灰色）表示欠费但实例未释放。
- “错误”（红色）表示实例创建失败、节点无法访问、节点宕机等情况。

4．区域：实例所在地域。

5．可用区：实例所在可用区。

6．版本：实例的选择版本。目前实例支持7.4.2和7.10.2两个版本。

7．节点规格：实例中包含的Logstash节点的CPU和内存配比。

8．存储规格：实例中包含的Logstash节点的存储类型和数据盘容量。数据盘支持高效云盘和SSD云盘，SSD云盘支持最大6TB的存储空间，高效云盘支持最大2TB的存储空间。

9．节点数量：实例中包含的Logstash节点的数量。

支付信息

1.支付方式：用户选择的付费方式，支持“预付费”和“后付费”两种方式。

2.到期时间：用户选择的购买时长决定到期时间。仅预付费会显示“到期时间”。后付费时，到期时间为“-”。

网络信息

1.网络：实例所属的网络。

2.子网：实例所属的子网。

节点信息

1.节点IP：实例中各个节点的IP地址。

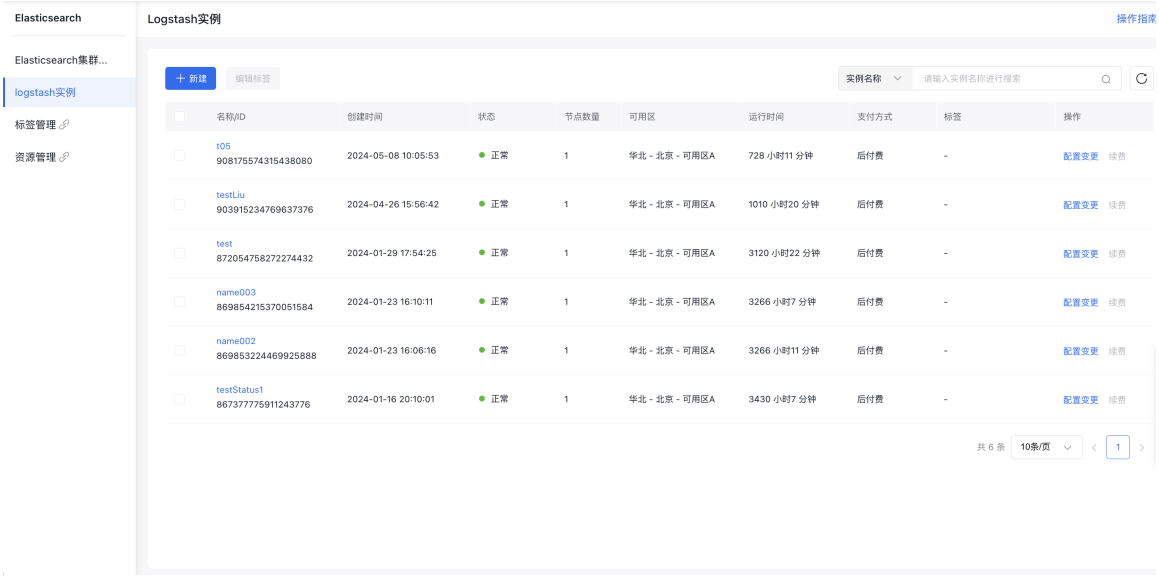
2.节点ID：节点唯一标识。

3.端口：实例固定为9600。

实例列表

实例列表概览

百度云Logstash的实例列表展示了实例的名称/id、创建时间、运行状态、节点数量、可用区、运行时间、支付方式等信息，并提供了新建实例、更改配置，续费、搜索实例等功能入口。



功能详情

- 创建实例

点击左上角“新建”，进入实例创建页面或点击实例名称进入可创建实例。详细信息，请参见创建实例。

- 刷新实例

点击搜索栏右侧的刷新图标，刷新可获取实例的实时状态。实例创建和配置变更后，需一段时间生效，状态为“生效中”， 可点击刷新图标查看实例的最新状态。当状态变为“运行中”，即可正常使用实例。

- 搜索实例

当实例数量较多时，用户可以通过搜索功能快速定位到目标实例。百度云Logstash支持按照实例ID、实例名称和标签搜索实例，点击搜索栏的下拉菜单选择需要的搜索类别，搜索时支持模糊匹配。

- 查看实例基本信息

点击实例名称进入实例详情页面，该页面展示了对应实例的基本信息，支付信息，网络信息和节点信息。详细信息，请参见查看实例详情信息。

- 实例配置变更

1. 点击目标实例操作列的“配置变更”按钮直接进入配置变更界面。

2. 点击目标实例名称进入实例详情页面，通过右上方的“配置变更”按钮进入配置变更页面。详情功能请参见实例扩缩容。

- 删除实例

点击实例名称进入实例详情页面，点击右上方的“删除实例”可释放实例。详情信息，请参删除实例。

- 续费

点击目标实例操作列的“续费”按钮进入实例续费页面。续费时长分“按年”和“按月”两种方式，选择特定的方式后需要确认续费的时长。点击下一步，确认订单信息无误后点击提交订单，选择付款方式，完成续费。

注：支付方式为预付费的才可执行续费操作。

实例扩缩容

概述

在业务发展中，数据量始终在变化，当现有的实例规格无法满足实际业务需求或者业务变更需降低成本，可以对实例扩缩容以满足需采集并转换更多数据的使用场景。目前Logstash支持对节点数量，节点规格，磁盘容量进行配置调整。

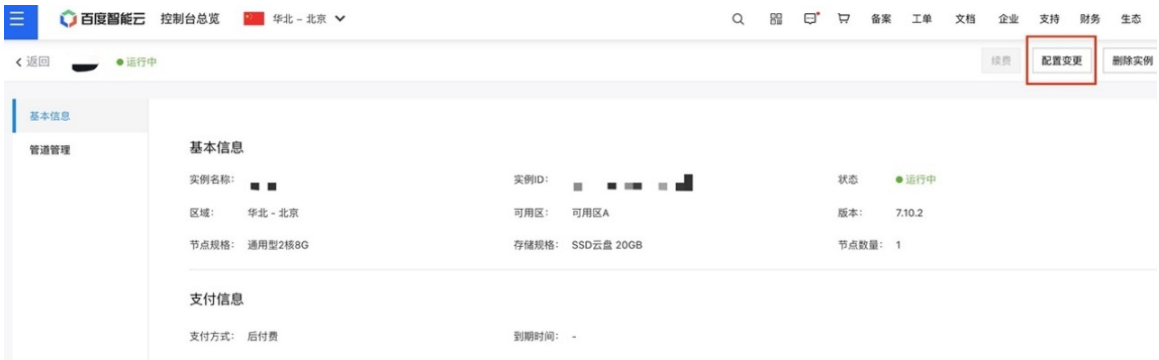
操作步骤

1.登录百度智能云Elasticsearch 控制台，进入百度云Elasticsearch的logstash实例列表。

2.集群配置变更操作入口：

- 实例列表页面，点击目标实例操作列的“配置变更”直接进入。

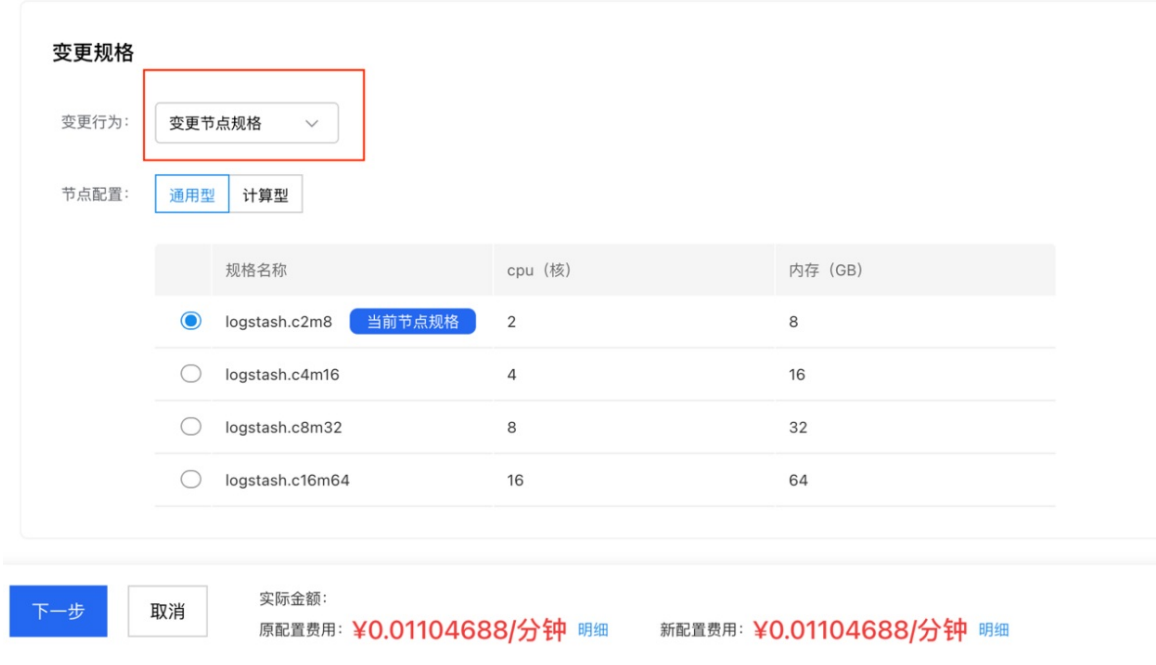
- 点击实例名称进入实例详情页面，点击右上方的“配置变更”进入页面。



3.进入配置变更页面后，选择变更行为和节点配置进行配置。

- 变更节点规格（cpu，内存）

需要变更节点的CPU和内存配置时，变更行为选择“变更节点规格”，选择需要变更的节点类型，在下方选择需要变更的新的规格配置。



变更到新的配置后，底部会展示新的配置价格，确认后点击“下一步”。

确认订单会展示本次变更的详细信息。确认无误之后，点击“提交订单”，提交配置变更操作。

实例变配重启过程中，Logstash服务暂时无法使用，当实例状态变为“运行中”，即可正常使用。

- 变更节点磁盘

当节点的计算资源足够，而存储空间不足或多余时，可以选择仅变更磁盘的容量。在变更行为中选择“变更节点容量”，对节点容量进行调整。底端展示新的配置价格，确认容量之后点击下一步确认订单即可。

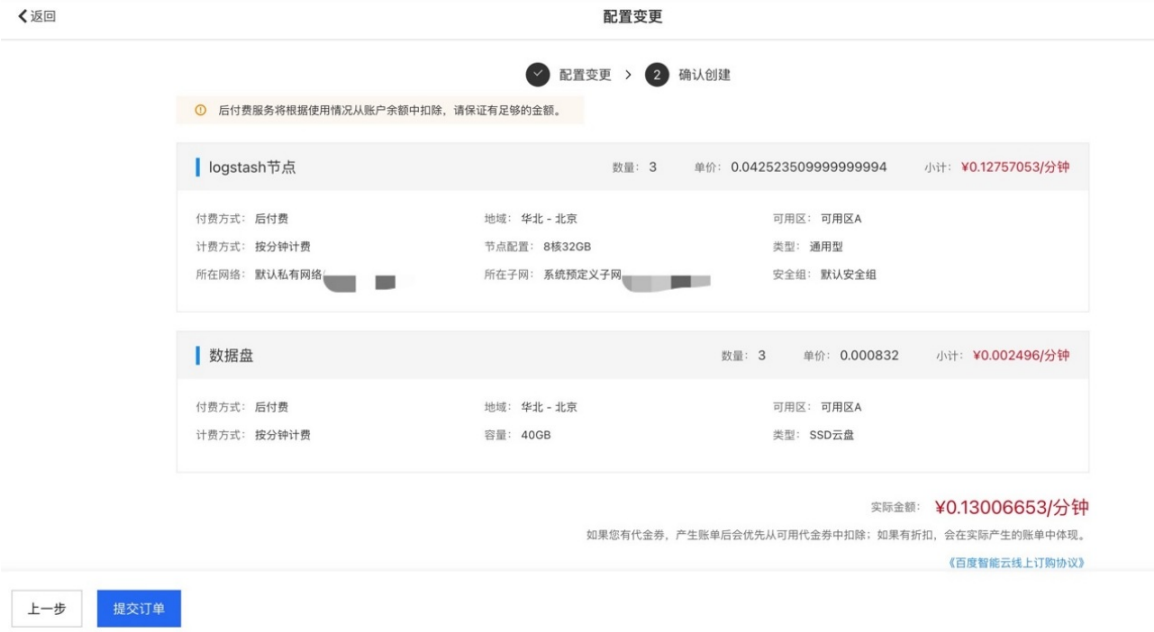
确认本次变更的详细信息，无误之后，点击“提交订单”。

配置变更操作提交之后，实例会进入“生效中”状态。实例变配、重启过程中，Logstash服务暂时无法使用，待实例状态变为“运行中”，即可正常使用。

- 变更节点数量

在配置变更页，选择“变更节点数量”，在“节点数量”一栏确定变更后的节点数。底端展示新的配置价格，确认之后点击“下一步”。

确认订单页面会展示本次变更的详情信息。确认无误之后，点击“提交订单”提交配置变更操作。



实例变配、重启过程中，Logstash服务暂时无法使用，待实例状态变为“运行中”，即可正常使用。

说明：

(1) 目前一次变更操作仅支持对节点数量、节点规格和磁盘容量的其中一种进行变更调整。如果有多种变更需求，请执行多次操作。

(2) 当实例状态为 red 或 yellow 时，不支持进行变更节点规格、缩容磁盘和删除节点等操作。请将集群状态恢复为 green，再进行以上操作。

(3) 目前Logstash实例不支持磁盘缩容，仅支持增加磁盘容量，且不支持磁盘类型变更。

(4) 目前Logstash实例不支持减少节点数量。所有节点会显示在实例详情页面，但不支持节点删除。

4. 调整配置费用说明

不同计费模式下的集群，调整配置时，费用结算方式有所不同。

- 预付费:包年包月的集群。在调整配置时，会根据集群剩余有效期以及新的配置的定价，计算变更后的费用，具体可以参考调整配置费用说明。
- 后付费:按量计费的集群，计费周期为秒。当配置调整完成后，下一秒费用按新的配置定价进行结算。

删除实例

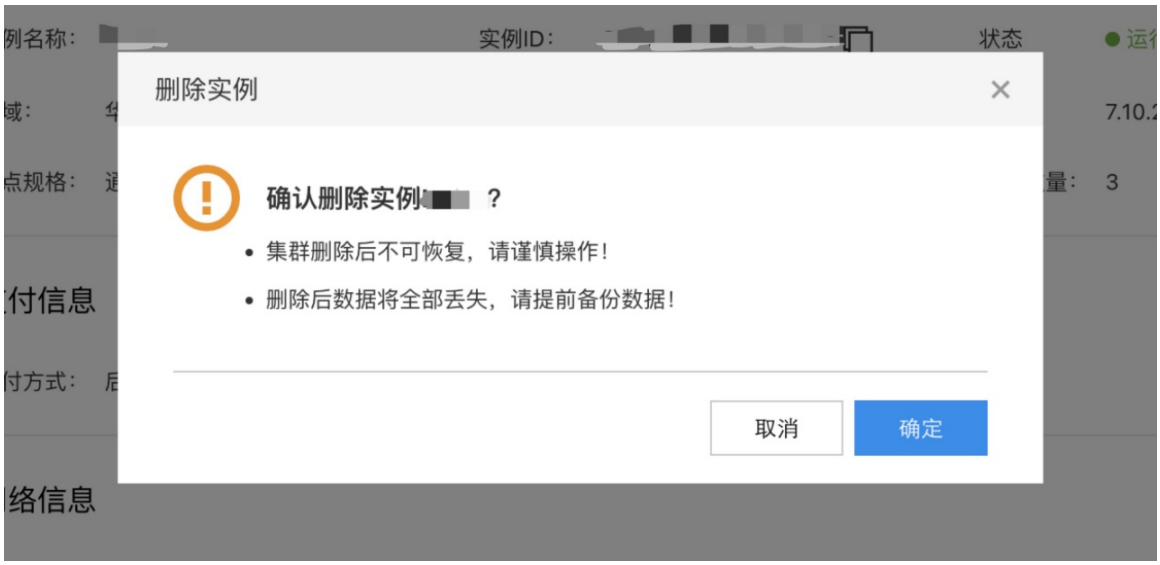
概述

用户不在需要当前实例或需要释放实例时，可以进行删除实例的操作。

操作步骤

1.通过实例名称进入实例详情页，点击右上方“删除实例”按钮。

2.点击删除实例后，会弹出确认删除的对话框，点击确定即可删除集群。实例删除后将不再计费且不再出现在实例列表中。



注：

1. 删除实例后数据将全部丢失且不可恢复，需要提前备份数据。
2. 进行创建实例、实例配置变更和管道配置等操作时不可删除实例。

管道管理

前提条件

1. 创建百度云Elasticsearch集群，具体操作，请参见创建集群
2. 创建Logstash实例,具体操作，请参见创建实例

创建管道

功能入口

登陆百度云Elasticsearch控制台，点击logstash实例，通过logstash名称进入详情页面，点击左侧导航栏的管道管理进入管道列表，点击左上角的“新增管道”按钮进入管道创建页面。

管道配置

1. Config配置

输入规则代码，**注意如需监听本地端口，请在8700-8800之间选择** 进行Config配置，配置示例如下：

```
input {
    elasticsearch {

      hosts => ["[redacted]"] # ["http://es目标数据ip:es目标数据端口"]
      index => "test1"
      user => "s[redacted]ser"  #用户名
      password => "[redacted]" #密码
      docinfo => true #设置为true，将会提取ES文档的元数据信息，例如index、type和id
    }

}
filter {

}
output {
    elasticsearch {
      hosts => ["[redacted]"]
      user => "[redacted]r"
      password => "[redacted]"
      index => "copy-of-production.%{[@metadata][_index]}" #output中设置为%{[@metadata][_index]}，表示匹配元数据中的index，
      document_type => "%{[@metadata][_type]}" #表示匹配元数据中索引的type，即迁移前后ES服务器的索引类型相同
      document_id => "%{[@metadata][_id]}"#表示匹配元数据中文档的id，即迁移前后ES服务器的文档id相同

    }

}
```

2. 参数配置

一共包括6个配置项，管道描述非必填，其他5个配置项必填。当所有必选项填完后，点击左下角按钮“保存并部署”或“保存”按钮。

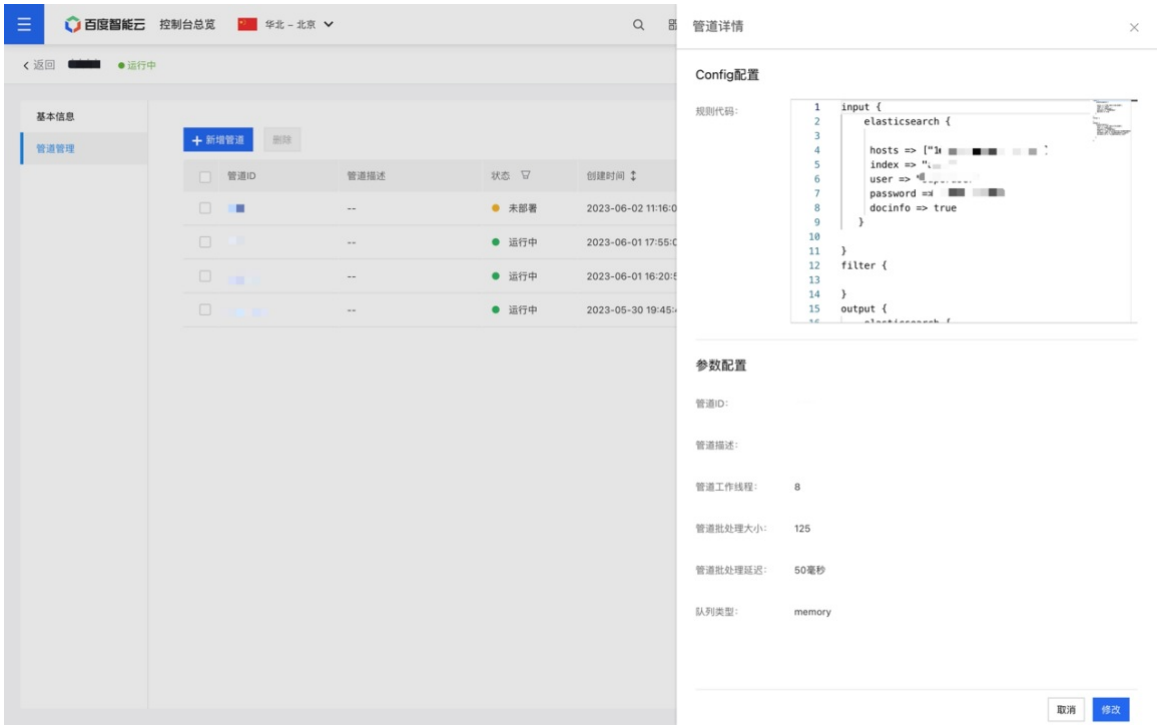
- 管道ID：管道的唯一标识，不可重复。支持1-25个大小写字母、数字、下划线。
- 管道描述：管道的描述信息，支持0-25个中文、字母、数字、下划线”_”或连接符"-"。非必填。
- 管道工作线程：管道的工作线程数量，可根据需求调节数量。
- 管道批处理大小：每个批次处理的最大事件数量。可根据需求调节。
- 管道批处理延迟：当管道批处理大小不满足时，每个批次最大的等待时间。可根据自身实际情况调节。
- 队列类型：用于事件缓冲的排队模型，可选值为 memory（基于内存的内列）或者 persisted（基于磁盘的持久化队列）。

3. 部署管道

配置管道完成后，可在创建管道页面点击“保存并部署”按钮，进行管道部署。也可以在管道列表页，点击“部署”按钮，部署已保存未部署的管道任务。点击“部署”按钮，弹窗进行二次确认。

4. 查看管道

管道列表页，点击管道ID可进入管道信息详情页面。可查看管道的Config配置和参数配置。



5. 修改管道

点击管道列表页的操作栏的“修改”按钮，可进入修改页面。或通过点击管道ID可进入管道信息详情页面，再点击右下角的“修改”按钮进入修改页面。

所有内容均支持修改（但管道ID要求不能重复）。运行中的管道，修改完成后，可以“保存并部署”或“取消”按钮。未运行的管道，修改后，可以“保存并部署”、“保存”或“取消”按钮。

6. 停止管道

点击管道列表页的操作列的【停止】按钮，跳出弹窗，点击确认后，停止管道任务运行。正在运行的管道任务会被中断。

7. 删除管道

点击管道列表页操作列的【删除】按钮，跳出弹窗，点击确认后，删除管道任务。正在运行的管道任务会被中断。已删除的管道无法恢复。

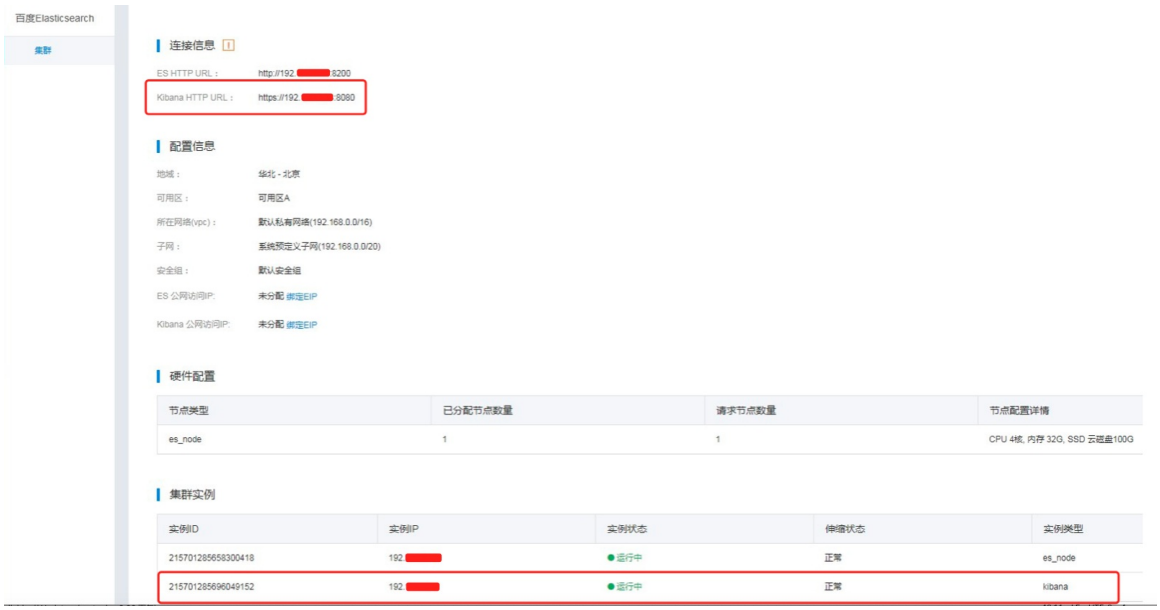
Kibana

Kibana使用指南

Kibana 是一款开源的可视化数据分析工具，可以使用 Kibana 对 Elasticsearch 中的数据进行搜索、分析，可以很方便的利用图表、表格及地图对数据进行多样的可视化展现。

创建与访问Kibana

在百度智能云选择Elasticsearch服务并完成创建集群后，除了已购买的Elasticsearch节点，默认会附赠一个Kibana节点，如图所示：



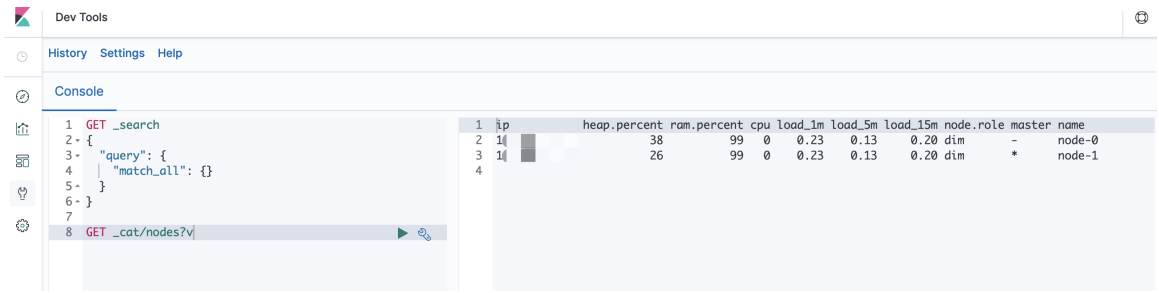
通过上图*连接信息*中所示的*Kibana HTTP URL*可以直接访问Kibana服务，**Kibana用户名密码与百度智能云Elasticsearch服务的用户名密码相同**。

使用Kibana

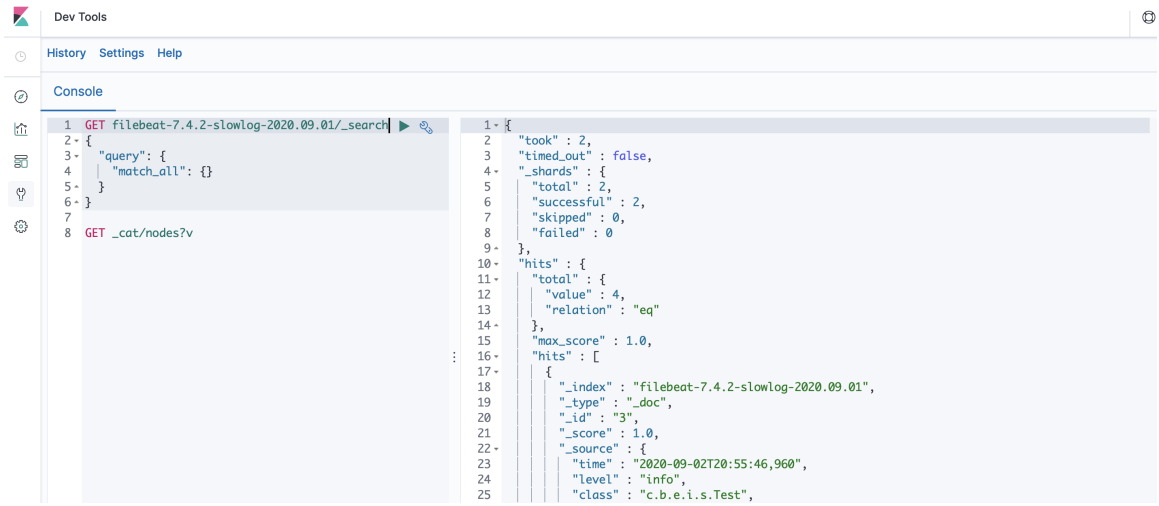
使用Console快速访问Elasticsearch

使用Dev Tools中的Console界面，可以直接使用REST API快速访问Elasticsearch。使用方式如下：

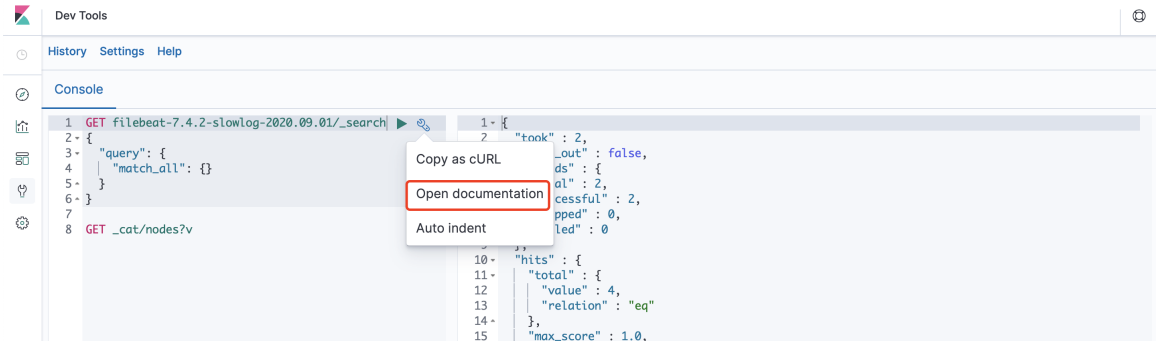
- 调用`GET _cat/nodes`，快速得到节点信息。



- 使用search API，可以直接检索索引*filebeat-7.4.2-slowlog-2020.09.01*中的数据。



- 直接点击如图所示*Open documentation*，可以查看Elasticsearch API文档。



使用仪表盘展示数据

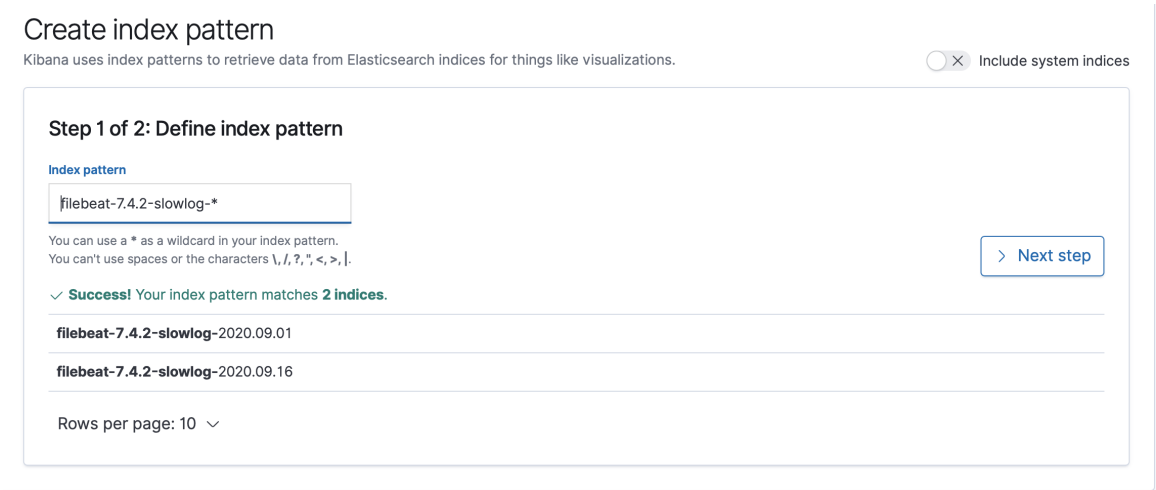
接下来本文用Elasticsearch慢查询日志为例，体验Kibana把数据在仪表盘中展示的过程。

在索引`filebeat-7.4.2-slowlog-2020.09.16`中，保存有使用filebeat导入的慢查询日志数据，数据结构如下所示：

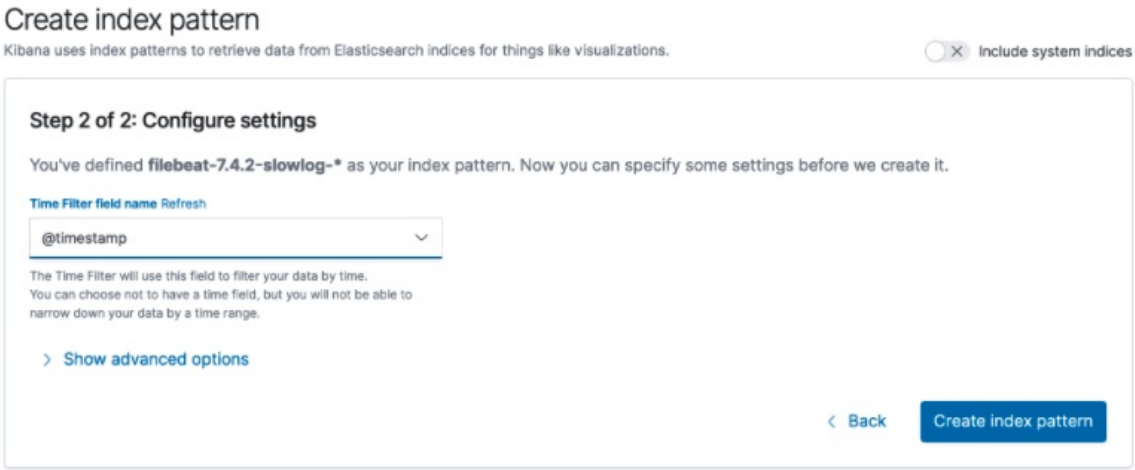
```
{
 "took_millis": "4",
 "shard_num": "2",
 "query_dsl": "{}",
 "hostname": "node-1",
 "shard": "0",
 "class": "index.search.slowlog.fetch",
 "took": "4.9",
 "level": "INFO",
 "index": "filebeat_test",
 "message": "[2020-09-16T20:05:12,000][INFO ][index.search.slowlog.fetch] [node-0][filebeat_test][0] took[4.9ms], took_millis[4], total_hits[5132 hits], types[], stats[], search_type[QUERY_THEN_FETCH], total_shards[2], source[{}], id[], ",
 "hits": "5132",
 "node": "node-0",
 "@timestamp": "2020-09-16T12:05:12.333Z",
 "time": "2020-09-16T20:05:12,000"
}
```

创建索引模式Index Patterns

加载到 Elasticsearch 的每组数据都有一个索引模式（Index Pattern）。 在这里为了检索`filebeat-7.4.2-slowlog-2020.09.16`，可以设置Index Pattern为 `filebeat-7.4.2-slowlog-*` ，如图所示：



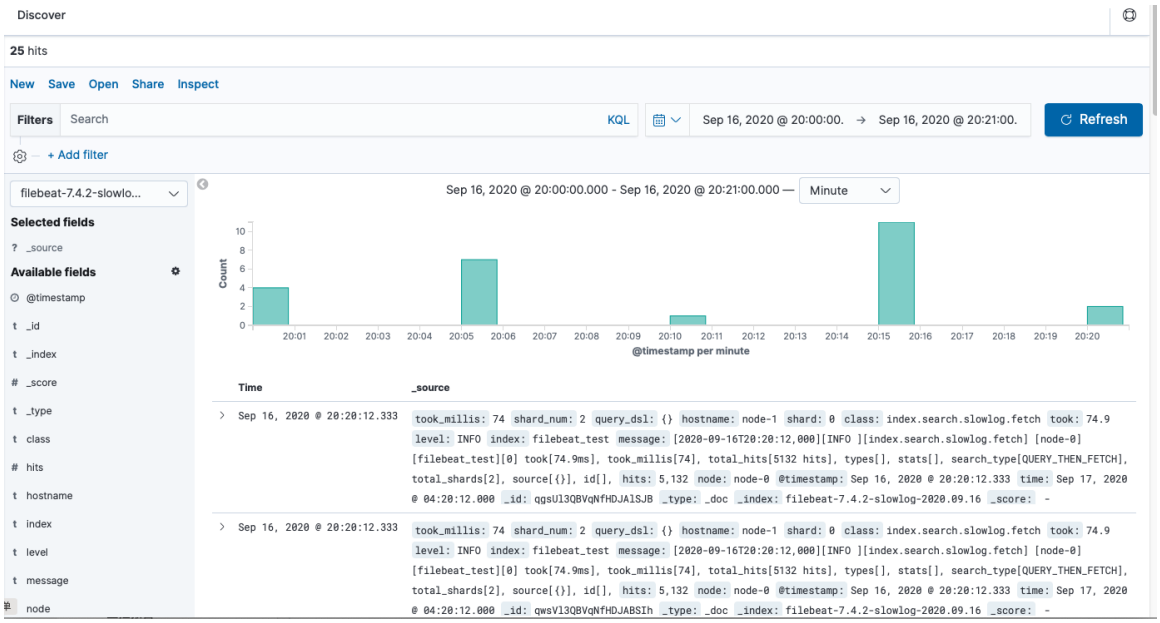
选择`@timestamp`为Time Filter field，Time Filter将使用此字段按时间筛选数据。



点击 **Create Index Pattern** 后成功创建Index Pattern。

使用Discover检索数据

单击侧面导航中的 **Discover** 进入 Kibana 的数据探索功能：



在查询框Filters里，用户可以输入Kibana查询语句来搜索上文保存的慢查询日志。在 Discover 页面下可以查看搜索结果并在 Visualize 页面下生成已保存搜索的可视化效果。

使用Visualize可视化数据

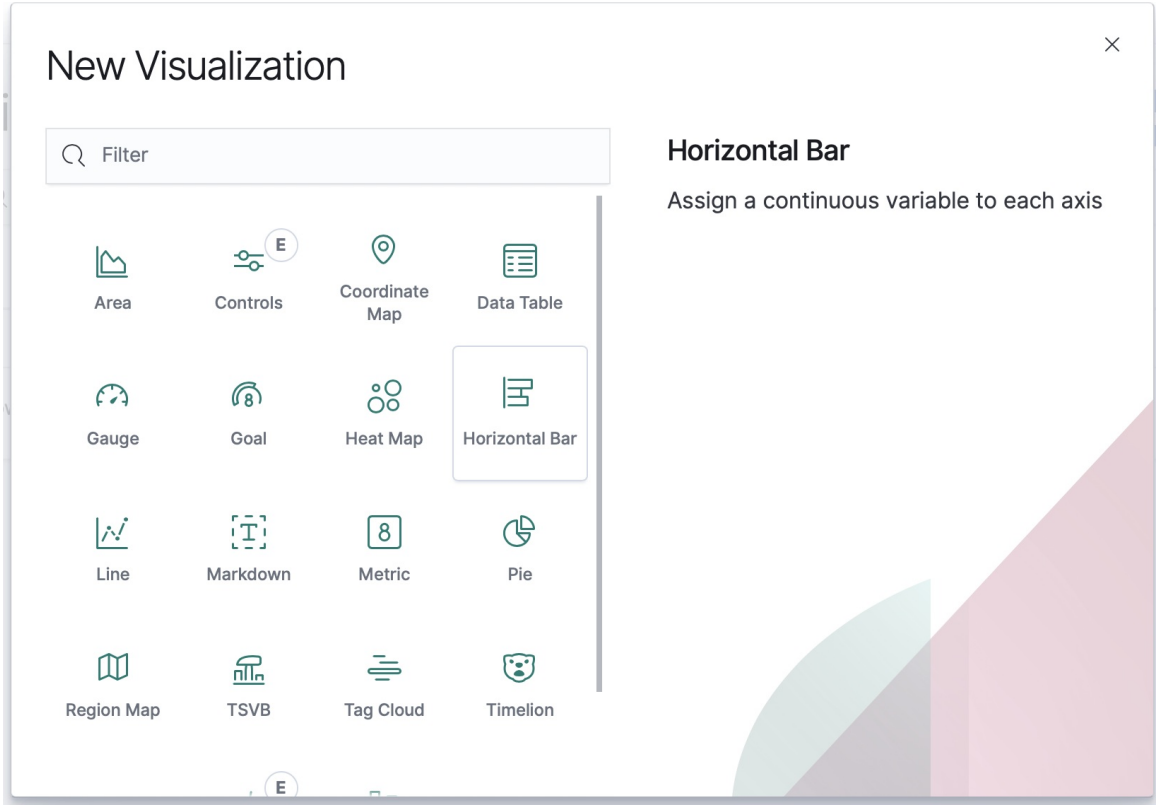
在侧边导航栏点击 **Visualize** 开始可视化慢查询日志数据。

在**New Visualize**中可以选择合适的图表来可视化数据，在这里我们制作一个**Horizontal Bar**（水平条形图）和一个**Pie**（饼图）。

创建一个Horizontal Bar图（水平条形图）

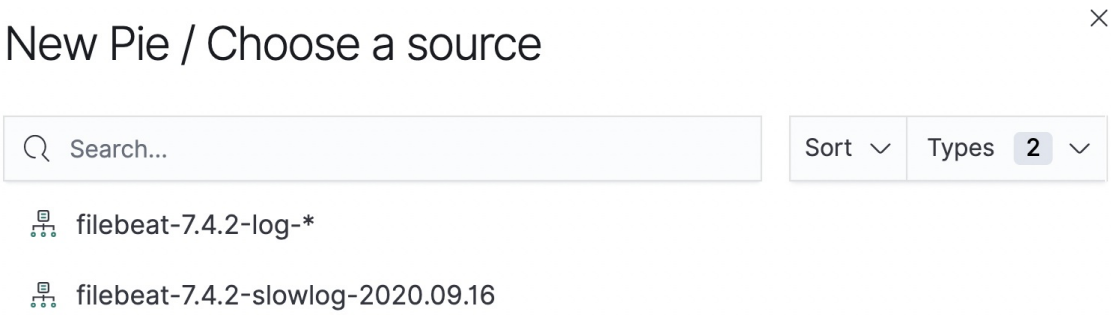
Horizontal Bar(水平条形图)一般用来比较不同分组的数据差异性。在日志分析的例子中，本文使用**Horizontal Bar(水平条形图)**来比较不同时间段内的慢日志条数。下面是创建Horizontal Bar图（水平条形图）的例子。

1.在 New Visualization 页面中选择 Horizontal Bar 选项；



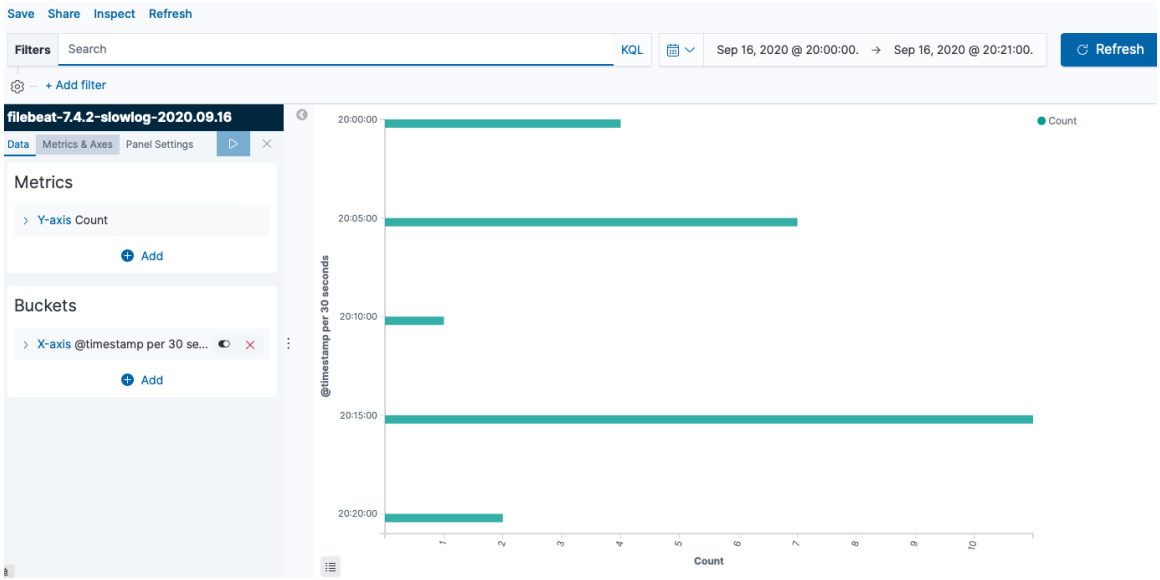
2.选择

配置的Index Pattern；



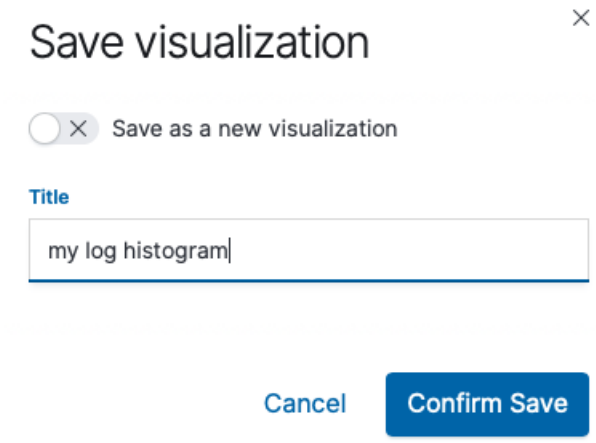
3.指标

项（Metrics）的y轴可以选择为Count（日志条数），Buckets选择Date Histogram（日期矩形图），得到的统计图如下所示：



如图，可以清楚的看出不同的时间内慢日志条数之间的对比，有助于排查慢查询问题。

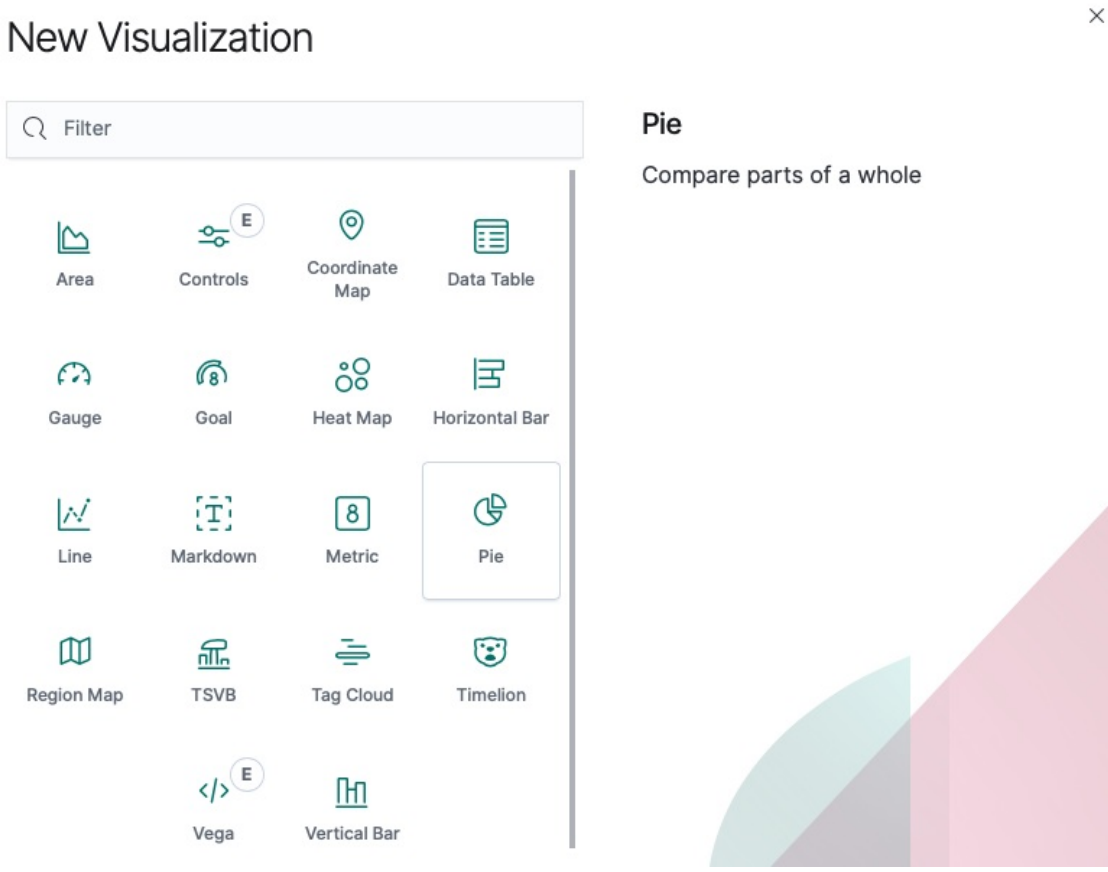
制作完成后，用户可以点击页面左上角 **Save** 然后输入名称 （如图示例`my log histogram`） 来保存这个图表供以后使用。



创建Pie(饼图)

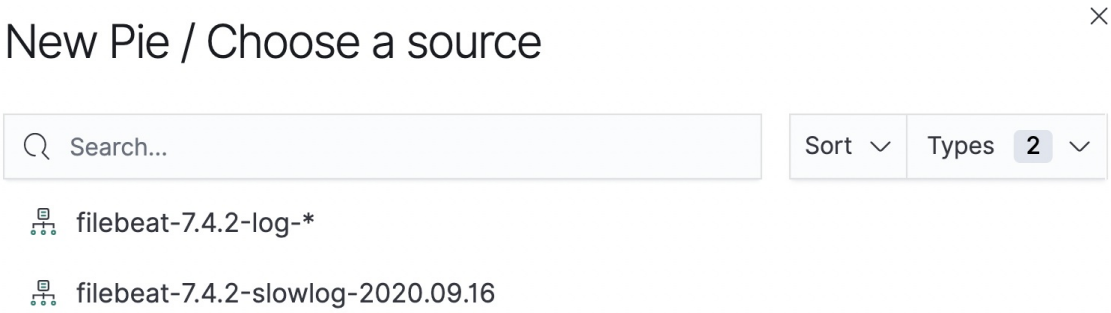
Pie(饼图)一般用于强调各项数据占总体的占比，提现了个体和整体的比较。在日志分析的例子中，我们可以使用**Pie(饼图)**来比较慢查询日志的查询耗时区间的分布情况。下面是创建Pie图(饼图)的例子。

1.在 New Visualization 页面中选择 Pie 选项；



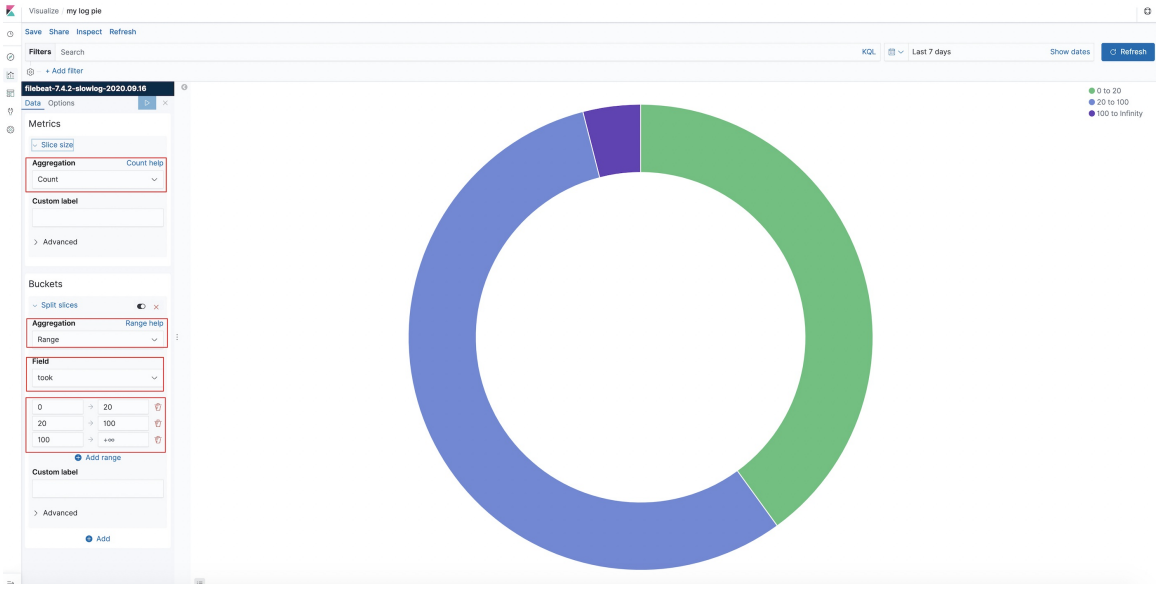
2.选择

配置的Index Pattern；



3.

对took字段进行Metrics的Count聚合，并设定了若干统计区间，得到的统计图如下所示：



如图可以看出，耗时20ms内和耗时20ms-100ms的查询占据了绝大部分，而100ms以上的查询只有极少的分布，可以更加直观的分析慢查询的情况。

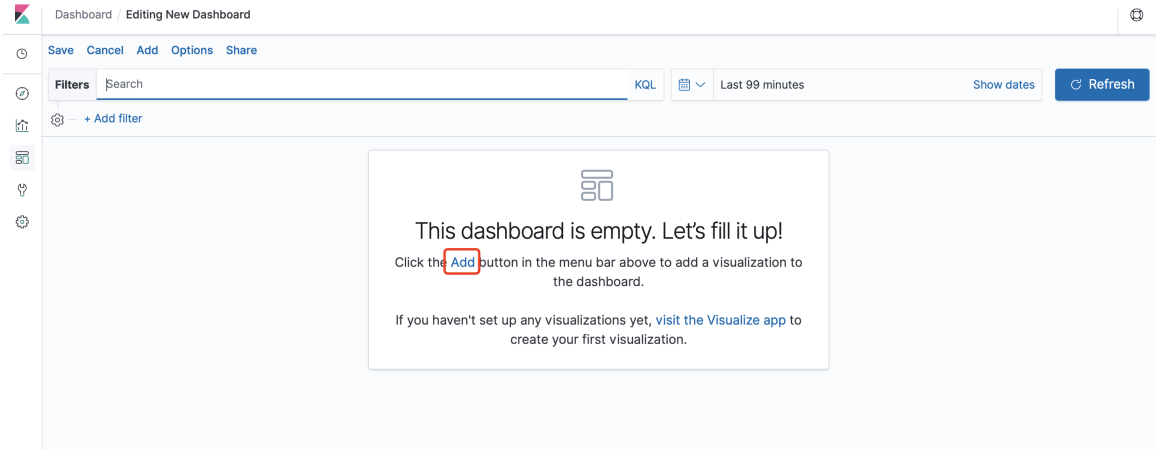
同上，制作完成后，点击页面左上角 **Save** 然后输入名称 （`my log pie`） 来保存这个图表供以后使用。

更多**Visualize**的用法可参见Kibana官方网站。

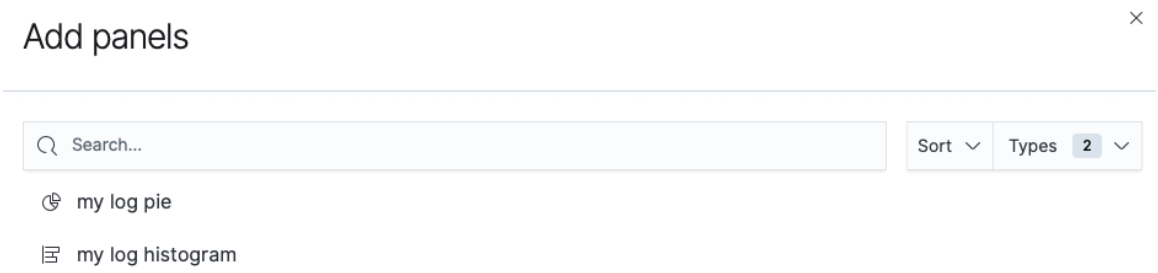
使用Dashboard仪表盘汇总数据

仪表板可用于集中管理和分享可视化控件集合。构建一个仪表板可以配置在**Visualize**中已保存的可视化控件，方法如下：

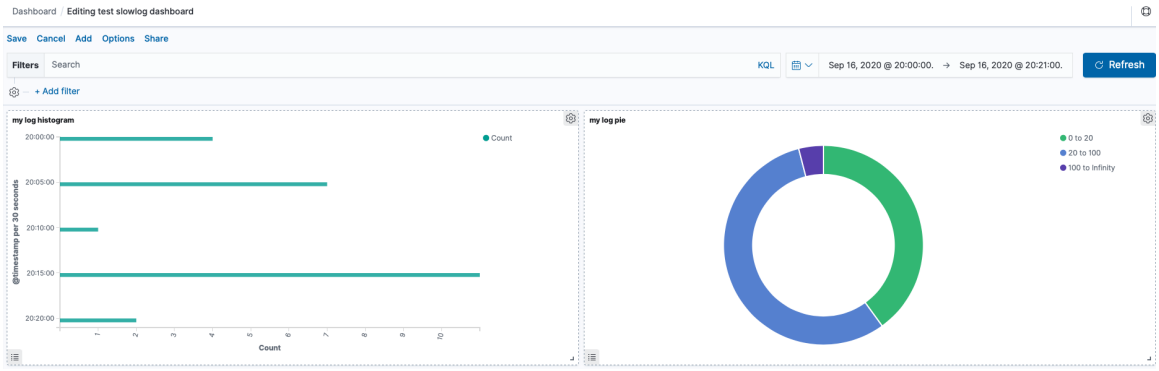
1. 在侧边导航栏点击 **Dashboard** 。
2. 点击 **Add** 显示已保存的可视化控件列表。



3. 单击`my log histogram`和`my log pie`，然后刚刚制作的Visualize就增加到了Dashboard中。



将鼠标悬停在一个可视化控件上可以显示允许编辑、移动、删除和调整它的控制器。



制作完成后，点击页面左上角 **Save** 然后输入名称 来保存Dashboard。

要获取分享链接或将仪表板嵌入到网页中的 HTML 代码，请保存仪表板并点击 **Share** 。

更多**Dashboard**的用法可参见Kibana官方网站。

API文档

概述

本文提供百度智能云检索与分析服务(Baidu Elasticsearch)的的API接口基本说明。

在线调试

用户可以通过可视化API调试工具，进行学习与调用。

服务域名

百度智能云开放多区域，BES支持的区域及对应域名请查看获取访问域名。

区域	访问EndPoint	支持协议
华北-北京	bes.bj.baidubce.com	HTTP,HTTPS
华北-保定	bes.bd.baidubce.com	HTTP,HTTPS
华北-阳泉	bes.yq.baidubce.com	HTTP,HTTPS
华东-苏州	bes.su.baidubce.com	HTTP,HTTPS
华东-上海	bes.fsh.baidubce.com	HTTP,HTTPS
华南-广州	bes.gz.baidubce.com	HTTP,HTTPS
中国香港	bes.hkg.baidubce.com	HTTP,HTTPS
华中金融-武汉	bes.fwh.baidubce.com	HTTP,HTTPS

集群相关接口

启动集群

接口说明

启动BES集群

请求URI

POST /api/bes/cluster/start

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	否	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	572979545519558656

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	String	常量字符串	

请求示例

```
 POST /api/bes/cluster/start
<公共请求头>
x-Region: bd


{
 "clusterId": "644734225693675520"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

重启集群

接口说明

重启集群

请求URI

POST /api/bes/cluster/restart

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	572979545519558656
mode	String	是	重启方式，full_restart：全量重启，roll_restart：滚动重启	full_restart

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	String	常量字符串	

请求示例

```
 POST /api/bes/cluster/restart
<公共请求头>
x-Region: bd


{
 "clusterId":"644734225693675520",
 "mode":"full_restart"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

扩容集群

接口说明

扩容集群 包含以下三种功能： 1、扩容节点数量 支持对数据节点、冷数据节点、协调节点进行节点数量变更，选直调模式 2、扩容节点规格 支持对数据节点、Kibana节点、冷数据节点、协调节点、专用主节点进行规格扩容，可选冷重启模式和蓝绿模式 3、扩容节点磁盘 支持对数据节点、冷数据节点进行磁盘扩容，可选直调模式和蓝绿模式

只填写需要修改的必填参数，不填写的参数默认为修改前参数，如果变配参数和修改前参数一致，会返回resize no change;

缩容集群 包含缩容节点规格和缩容磁盘 kibana仅支持缩规格，数据节点和冷数据节点支持缩规格、磁盘(冷数据节点仅支持缩磁盘大小)、节点数量，专有主节点支持缩规格，协调节点支持缩规格、节点数量

注意：缩节点数量调用的不是本接口而是/api/bes/cluster/v2/instance/delete删除节点v2接口

缩容节点规格时除了kibana节点外仅支持蓝绿和冷重启，kibana节点仅支持冷重启；缩容磁盘类型时仅支持冷重启和蓝绿，缩容磁盘大小和cdsIO时仅支持蓝绿

注意：缩cdsIO仅支持pl2磁盘

请求URI

POST /api/bes/cluster/resize

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID，超过规定18位会返回500错误	570900694777860096
modules	List	是	模板信息	
paymentType	String	是	计费方式	postpay表示后付费、prepay表示预付费
coupon	String	否	可用代金券，缩容和后付费不用传	123456
resizeMode	String	是	扩容、升配方式，如果resizeMode为空会报"Http Message Not Readable"。SCROLL：滚动模式，SMOOTH：蓝绿模式，COLD：冷重启模式，HOT：直调模式	SCROLL
isShrink	Boolean	否	升配false，降配true，默认为false	false

ModulesItem字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
type	String	否	节点类型，es_node：数据节点，kibana：kibana节点，es_dedicated_master：专用主节点，es_cold_tier_node：冷数据节点，es_coordinate_node：协调节点	kibana
version	String	否	集群版本	7.4.2
desireInstanceNum	Integer	否	节点数量	1
slotType	String	否	机型	bes.c3.c1m2
diskSlotInfo	DiskSlotInfo	否	磁盘信息	

DiskSlotInfo字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
type	String	否	磁盘类型；磁盘类型；ssd表示高效云盘、premium_ssd表示ssd云盘	premium_ssd
size	BigDecimal	否	磁盘大小；单位G 最小为50G premium_ssd最大为6144G ssd最大为20480G	50
cdsExtraIo	BigDecimal	否	预配置IO	100

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
orderId	String	订单ID	186f2566f2884534a65ed8cdc2499d82

请求示例

```
 POST /api/bes/cluster/resize
<公共请求头>
x-Region: bd
```

1. 扩容节点数量

支持对数据节点、冷数据节点、协调节点进行节点数量变更，选直调模式

示例：

```
{
 "clusterId" : "570900694777860096",
 "isShrink": false,
 "modules": [
  {
    "type": "es_node",
    "desireInstanceNum": 5
  }
 ],
 "paymentType": "postpay",
 "resizeMode": "HOT"
}
```

2. 扩容节点规格

支持对数据节点、Kibana节点、冷数据节点、协调节点、专用主节点进行规格扩容，可选冷重启模式和蓝绿模式

示例：

```
{
 "clusterId" : "570900694777860096",
 "isShrink": false,
 "modules": [
  {
    "type": "es_node",
    "slotType": "bes.g3.c4m16"
  }
 ],
 "paymentType": "postpay",
 "resizeMode": "SMOOTH"
}
```

3. 扩容节点磁盘

支持对数据节点、冷数据节点进行磁盘扩容，可选直调模式和蓝绿模式

示例：

```
{
 "clusterId" : "570900694777860096",
 "isShrink": false,
 "modules": [
  {
    "type": "es_node",
    "diskSlotInfo": {
  "size": 100
 }
  }
 ],
 "paymentType": "postpay",
 "resizeMode": "SMOOTH"
}
```

4. 变更磁盘类型

支持对数据节点变更磁盘类型，升配磁盘类型的同时，支持调整磁盘大小，可选直调模式和蓝绿模式

示例：

```
  {
   "clusterId" : "570900694777860096",
   "isShrink": false,
   "modules": [
     {
       "type": "es_node",
       "diskSlotInfo": {
       "type":"enhanced_ssd_pl2",
    "size": 500,
    "cdsExtraIo": 100
   }
     }
   ],
   "paymentType": "postpay",
   "resizeMode": "SMOOTH"
 }
```

响应示例

```
  HTTP/1.1 200 OK
 <公共响应头>

 {
   "result" : {
     "orderId" : "186f2566f2884534a65ed8cdc2499d82"
   },
   "success" : true,
   "status" : 200
 }
```

错误码

请参考通用错误码

停止集群

接口说明

停止BES集群

请求URI

POST /api/bes/cluster/stop

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	572979545519558656

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	String	常量字符串	

请求示例

```
 POST /api/bes/cluster/stop
<公共请求头>
x-Region: bd


{
 "clusterId": "644734225693675520"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

用户可用代金券列表

接口说明

用户可用代金券列表

请求URI

POST /api/bes/cluster/coupon/avail

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	代金券结果	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
coupons	List	代金券数据	

CouponsItem字段数据结构说明

参数名称	参数类型	描述	示例值
id	BigDecimal	代金券id	1189112
name	String	代金券号	DN1EIM18VLLYBN8
couponType	String	优惠券类型	CASH_COUPON
productType	String	产品类型	COMMON
productTypes	List	多产品类型	["COMMON"]
productRuleDescription	String	产品描述	通用（除域名/短信/专线/商标知产/证书/工商财税）
totalAmount	BigDecimal	总额	500.0
balance	String	余额	500.00
amountOrDiscount	String	金额/折扣	500.00
usedAmount	String	已使用金额	
beginTime	BigDecimal	生效时间	1641744000000
endTime	BigDecimal	过期时间	1665331200000
couponStatus	String	状态	NORMAL
status	String	筛选状态	USED_UP
region	String	适用地域	global
conditionArgsMap	ConditionArgsMap	特殊场景参数	
effectArgsMap	EffectArgsMap	效果参数	
conditionEffectDescription	String	特殊场景描述	通用

请求示例

```
 POST /api/bes/cluster/coupon/avail
<公共请求头>
x-Region: bd


{ }
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "coupons" : [ {
      "productRuleDescription" : "通用（除域名/短信/专线/商标知产/证书/工商财税）",
      "effectArgsMap" : { },
      "amountOrDiscount" : "500.00",
      "usedAmount" : "usedAmount",
      "productTypes" : "[\"COMMON\"]",
      "conditionEffectDescription" : "通用",
      "totalAmount" : 500.00,
      "balance" : "500.00",
      "conditionArgsMap" : { },
      "couponType" : "CASH_COUPON",
      "name" : "DN1EIM18VLLYBN8",
      "couponStatus" : "NORMAL",
      "id" : 1189112,
      "beginTime" : 1641744000000,
      "endTime" : 1665331200000,
      "region" : "global",
      "productType" : "COMMON",
      "status" : "USED_UP"
    }, {
      "productRuleDescription" : "通用（除域名/短信/专线/商标知产/证书/工商财税）",
      "effectArgsMap" : { },
      "amountOrDiscount" : "500.00",
      "usedAmount" : "usedAmount",
      "productTypes" : "[\"COMMON\"]",
      "conditionEffectDescription" : "通用",
      "totalAmount" : 500.00,
      "balance" : "500.00",
      "conditionArgsMap" : { },
      "couponType" : "CASH_COUPON",
      "name" : "DN1EIM18VLLYBN8",
      "couponStatus" : "NORMAL",
      "id" : 1189112,
      "beginTime" : 1641744000000,
      "endTime" : 1665331200000,
      "region" : "global",
      "productType" : "COMMON",
      "status" : "USED_UP"
    } ]
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

创建集群

接口说明

创建集群

节点配置类型：参考https://cloud.baidu.com/doc/BES/s/tki03tbio

version说明： sdk支持的版本是'7.4.2'

集群中节点类型说明

序号	类型值	说明	是否必须
1	es_node	es数据节点	是
2	kibana	kibana节点	是
3	es_dedicated_master	专有主节点	否
4	es_coordinate_node	协调节点	否
5	es_cold_tier_node	冷节点	否

集群中节点磁盘类型说明

序号	类型值	说明
1	premium_ssd	ssd云盘
2	ssd	高效云盘
3	enhanced_ssd_pl1	增强型SSD_PL1

集群类节点支持的计算资源规格说明

数据节点	Kibana	专用主节点	冷数据节点	协调节点
bes.g3.c2m8	bes.g3.c2m8	bes.g3.c2m8	bes.g3.c2m8	bes.g3.c2m8
bes.g3.c4m16	bes.g3.c4m16	bes.g3.c4m16	bes.g3.c4m16	bes.g3.c4m16
bes.g3.c8m32	bes.g3.c8m32	bes.g3.c8m32	bes.g3.c8m32	bes.g3.c16m64
bes.g3.c16m64	bes.c3.c2m4	bes.g3.c16m64	bes.g3.c16m64	bes.g3.c16m64
bes.g3.c32m128	bes.c3.c4m8	bes.g3.c16m64	bes.g3.c16m64	bes.g3.c32m128
bes.c3.c2m4	bes.c3.c8m16	bes.c3.c32m64	bes.g3.c32m128	bes.c3.c2m4
bes.c3.c4m8		bes.c3.c8m16	bes.c3.c2m4	bes.c3.c4m8
bes.c3.c8m16		bes.c3.c16m32	bes.c3.c4m8	bes.c3.c4m8
bes.c3.c16m32		bes.m3.c4m32	bes.c3.c16m32	bes.c3.c8m16
bes.c3.c32m64		bes.m3.c2m16	bes.c3.c32m64	bes.c3.c16m32
bes.ic3.c4m4		bes.m3.c4m32	bes.ic3.c4m4	bes.c3.c32m64
bes.ic3.c8m8		bes.m3.c8m64	bes.ic3.c8m8	bes.ic3.c4m4
bes.ic3.c12m12			bes.ic3.c12m12	bes.ic3.c8m8
bes.ic3.c16m16			bes.ic3.c16m16	bes.ic3.c12m12
bes.m3.c2m16			bes.m3.c2m16	bes.ic3.c16m16
bes.m3.c4m32			bes.m3.c4m32	bes.m3.c4m32
bes.m3.c8m64			bes.m3.c8m64	bes.m3.c8m64

集群各类节点支持的存储资源说明：

序号	节点类型	说明
1	es数据节点	支持ssd云盘、高效云盘，最小50GB
2	Kibana	无；不需要设置
3	专用主节点	ssd云盘，大小20GB
4	冷数据节点	高性能云盘，最小50GB
5	协调节点	高性能云盘，大小20GB

请求URI

POST /api/bes/cluster/create

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
name	String	是	集群名称	postpaynew1
password	String	是	集群密码	123456aA
securityGroupId	String	是	安全组ID	g-s77w7h0ierks
subnetUuid	String	是	子网ID	sbn-38zqasty2i8n
availableZone	String	是	可用区	cn-bd-b
vpcId	String	是	虚拟网络ID	vpc-5dd5bib4h0vc
isOldPackage	Boolean	否	区分新老套餐 (true表示老套餐、false表示新套餐)；默认是false	false
version	String	是	集群版本	7.4.2
modules	List	是	节点信息	
billing	Billing	是	集群计费信息	
tags	List	否	选择标签	[{"testKey":"key",testValue:"value'}]
enableSSL	Boolean	否	是否开启https	false
resGroupId	String	否	资源分组id	RESG-44pvmsruNiy

ModulesItem字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
instanceNum	Integer	是	节点数量	2
slotType	String	是	资源配置类型；	bes.c3.c2m4
diskSlotInfo	DiskSlotInfo	是	磁盘信息（新套餐必需，老套餐不需要）	
type	String	是	节点类型	kibana

DiskSlotInfo字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
size	Integer	是	磁盘大小；单位G 最小为50G premium_ssd最大为6144G ssd最大为20480G	50
type	String	是	磁盘类型；磁盘类型；ssd表示高效云盘、premium_ssd表示ssd云盘	premium_ssd
cdsExtraIo	Integer	是	预配置io，只有pl2磁盘可配置	11

Billing字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
paymentType	String	是	计费方式;postpay表示后付费、prepay表示预付费	postpay
time	Integer	否	购买时长（后付费时time参数设置为0），单位是月	0
enableAutoRenew	Boolean	否	是否开启自动续费	true
autoRenewInfo	AutoRenewInfo	否	自动续费信息	
coupon	String	否	可用代金券	123456

AutoRenewInfo字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
renewTimeUnit	String	否	自动续费单位，month：月，year：年	month
renewTime	BigDecimal	否	自动续费时间，可填1-9个月、1-3年	1

TagsItem字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
field_2	String	是		
field_3	String	是		

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/create
<公共请求头>
x-Region: bd


示例：
{
  "name": "postpaynew1",
  "password": "123456aA",
  "securityGroupId": "g-s77w7h0ierks",
  "subnetUuid": "sbn-38zqasty2i8n",
  "availableZone": "cn-bd-b",
```

```
  "vpcId": "vpc-5dd5bib4h0vc",
  "isOldPackage": false,
  "version": "7.4.2",
  "modules": [
    {
      "instanceNum": 2,
      "slotType": "bes.g3.c2m8",
      "diskSlotInfo": {
        "size": 50,
        "type": "premium_ssd"
      },
      "type": "es_node"
    }, {
      "instanceNum": 3,
      "slotType": "bes.g3.c2m8",
      "diskSlotInfo": {
        "size": 20,
        "type": "premium_ssd"
      },
      "type": "es_dedicated_master"
    }, {
      "instanceNum": 3,
      "slotType": "bes.g3.c2m8",
      "diskSlotInfo": {
        "size": 50,
        "type": "ssd"
      },
      "type": "es_cold_tier_node"
    }, {
      "instanceNum": 1,
      "slotType": "bes.g3.c2m8",
      "diskSlotInfo": {
        "size": 20,
        "type": "ssd"
      },
      "type": "es_coordinate_node"
    },
    {
      "instanceNum": 1,
      "slotType": "bes.c3.c2m4",
      "type": "kibana"
    }
  ],
  "billing": {
    "paymentType": "postpay",
    "time": 0,
    "enableAutoRenew":true,
    "autoRenewInfo":{
        "renewTimeUnit":"month",
        "renewTime":1
    }
  },
  "tags":[
   {
    "tagKey":"key",
    "tagValue":"value"
   }
   ]
}
```


响应示例

```
  HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

查看集群列表

接口说明

查看集群列表，优化集群状态

请求URI

POST /api/bes/cluster/v2/list

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	否	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
pageNo	Integer	是	页数（从1开始取值）	1
pageSize	Integer	是	页大小	10

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
page	Page	集群信息列表	

Page字段数据结构说明

参数名称	参数类型	描述	示例值
pageNo	Integer	页数	0
pageSize	Integer	页大小	10
totalCount	Integer	集群个数	1

ResultItem字段数据结构说明

参数名称	参数类型	描述	示例值
clusterId	String	集群ID	572979545519558656
clusterName	String	集群名称	postpaynew1
createTime	String	创建时间	2021-10-26 10:56:12
actualStatus	String	状态,active：正常，activating：生效中，invalid：失效	active
runningTime	String	运行时间	12 小时23 分钟
region	String	区域	bj
version	String	版本	7.10.2
clusterHealth	ClusterHealth		
billing	Billing	集群计费信息	
tags	List	标签信息	
resGroupList	List	资源分组	
innerAccessIp	String	内部访问ip	1.1.1.1
accessEip	String	访问的eip	1.1.1.1
vpcId	String	vpcId	xxx-xxx-xxx-xxx
subnet	List	子网信息	
network	List	网络信息	

ClusterHealth字段数据结构说明

参数名称	参数类型	描述	示例值
status	String	健康状态	GREEN(健康)、YELLOW(亚健康)、RED(不健康)

Billing字段数据结构说明

参数名称	参数类型	描述	示例值
paymentType	String	计费类型	postpay

TagsItem字段数据结构说明

参数名称	参数类型	描述	示例值
tagKey	String	标签key	test
tagValue	String	标签value	111

ResGroupListItem字段数据结构说明

参数名称	参数类型	描述	示例值
groupId	String	资源分组id	RESG-44pvmsruNiy
groupName	String	资源分组名称	默认分组

SubnetItem字段数据结构说明

参数名称	参数类型	描述	示例值
availableZone	String	可用区名称	zoneA
subnetName	String	字网名称	系统预定义子网

NetworkItem字段数据结构说明

参数名称	参数类型	描述	示例值
subnetId	String	子网id	xxx-xxx-xxx-xxx-xxx

请求示例

```
 POST /api/bes/cluster/v2/list
<公共请求头>
x-Region: bd


{
  "pageNo" : 1,
  "pageSize" : 10
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "success" : true,
  "page" : {
    "result" : [ {
      "resGroupList" : [ {
        "groupName" : "默认分组",
        "groupId" : "RESG-44pvmsruNiy"
      }, {
        "groupName" : "默认分组",
        "groupId" : "RESG-44pvmsruNiy"
      } ],
      "subnet" : [ {
        "availableZone" : "zoneA",
        "subnetName" : "系统预定义子网"
      }, {
        "availableZone" : "zoneA",
        "subnetName" : "系统预定义子网"
      } ],
      "accessEip" : "1.1.1.1",
      "clusterId" : "572979545519558656",
      "runningTime" : "12 小时23 分钟",
      "clusterHealth" : {
        "status" : "GREEN(健康)、YELLOW(亚健康)、RED(不健康)"
      },
      "version" : "7.10.2",
      "billing" : {
        "paymentType" : "postpay"
      },
      "tags" : [ {
        "tagValue" : "111",
        "tagKey" : "test"
      }, {
        "tagValue" : "111",
        "tagKey" : "test"
      } ],
      "network" : [ {
        "subnetId" : "xxx-xxx-xxx-xxx-xxx"
      }, {
        "subnetId" : "xxx-xxx-xxx-xxx-xxx"
      } ],
```

```
      "actualStatus" : "active",
      "createTime" : "2021-10-26 10:56:12",
      "innerAccessIp" : "1.1.1.1",
      "clusterName" : "postpaynew1",
      "vpcId" : "xxx-xxx-xxx-xxx",
      "region" : "bj"
    }, {
      "resGroupList" : [ {
        "groupName" : "默认分组",
        "groupId" : "RESG-44pvmsruNiy"
      }, {
        "groupName" : "默认分组",
        "groupId" : "RESG-44pvmsruNiy"
      } ],
      "subnet" : [ {
        "availableZone" : "zoneA",
        "subnetName" : "系统预定义子网"
      }, {
        "availableZone" : "zoneA",
        "subnetName" : "系统预定义子网"
      } ],
      "accessEip" : "1.1.1.1",
      "clusterId" : "572979545519558656",
      "runningTime" : "12 小时23 分钟",
      "clusterHealth" : {
        "status" : "GREEN(健康)、YELLOW(亚健康)、RED(不健康)"
      },
      "version" : "7.10.2",
      "billing" : {
        "paymentType" : "postpay"
      },
      "tags" : [ {
        "tagValue" : "111",
        "tagKey" : "test"
      }, {
        "tagValue" : "111",
        "tagKey" : "test"
      } ],
      "network" : [ {
        "subnetId" : "xxx-xxx-xxx-xxx-xxx"
      }, {
        "subnetId" : "xxx-xxx-xxx-xxx-xxx"
      } ],
      "actualStatus" : "active",
      "createTime" : "2021-10-26 10:56:12",
      "innerAccessIp" : "1.1.1.1",
      "clusterName" : "postpaynew1",
      "vpcId" : "xxx-xxx-xxx-xxx",
      "region" : "bj"
    } ],
    "pageNo" : 0,
    "pageSize" : 10,
    "totalCount" : 1
  },
  "status" : 200
}
```


错误码

请参考通用错误码

EIP解绑

请求URI

POST /api/bes/cluster/eip/unbind

请求参数

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	必填	类型	描述	示例值
deployId	是	String	集群id，即clusterId	1
moduleTemplateName	是	String	节点类型	es_node 或者 kibana

响应体参数

响应体字段数据结构说明

参数名称	类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	String	常量字符串	

请求示例

```
 POST /api/bes/cluster/eip/unbind
<公共请求头>
x-Region: bd


{
  "moduleTemplateName" : "es_node 或者 kibana",
  "deployId" : "1"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

EIP绑定

请求URI

POST /api/bes/cluster/eip/bind

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	必填	类型	描述	示例值
instanceId	是	String	集群id，即clusterId	218185657699405824
instanceType	是	String	节点类型	es_node 或者 kibana
eip	是	String	eip地址	

响应体参数

响应体字段数据结构说明

参数名称	类型	描述	示例值
success	boolean	请求是否成功	true
status	integer	状态码	200
result	String	常量字符串	

请求示例

```
 POST /api/bes/cluster/eip/bind
<公共请求头>
x-Region: bd


{
  "instanceId" : "218185657699405824",
  "instanceType" : "es_node 或者 kibana",
  "eip" : "eip"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

查看集群详情信息

接口说明

查看集群详情信息，集群/节点状态优化

请求URI

POST /api/bes/cluster/v2/detail

请求头参数

参数名	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	必填	参数类型	描述	示例值
clusterId	是	String	集群id，超过规定18位会返回500错误	12345

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	参数类型
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result	集群详细信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
expireTime	ModelNull	终止时间	
clusterId	String	集群ID	572979545519558656
clusterName	String	集群名称	postpaynew1
adminUsernam e	String	管理员名称	superuser
actualStatus	String	实际状态，active：正常，activating：生效中，invalid：失效	active
desireStatus	String	目标状态	active
esUrl	String	ES服务器访问路径	http: //172.16.16.6: 8200
kibanaUrl	String	Kibana服务器访问路径	https: //172.16.16.7: 8080
exporterUrl	String	Exporter监控接入地址	http://172.16.16.7:8081
enableEsSSL	Boolean	es是否开启https	false
enableKibanaS SL	Boolean	kibana是否开启https	false
esEip	String	String	
kibanaEip	String	Kibana公网访问IP	
modules	List	节点信息列表	
instances	List	2	
region	String	区域	bd
inspectAuthoriz ed	String	是否已开通巡检授权	true
vpc	String	虚拟网络	默认私有网络(172.16.0.0/16)
vpcId	String	虚拟网络ID	1b9c28ef-0874-414f-8a0c-aa85796ffffb
subnet	String	子网	系统预定义子网B(172.16.16.0/20)
availableZone	String	可用区	zoneB
securityGroup	String	安全组	默认安全组
clusterHealth	ClusterHealt h		
billing	Billing	计费信息	
network	List	网络信息	
tags	List	标签信息	
log	Log	当前日志设置	
resGroupList	List	资源分组	
bdVersion	String	内核版本	1.7.6

ModulesItem字段数据结构说明

参数名称	参数类型	描述	示例值
type	String	type表示节点的类型；es_node表示es节点、kibana表示kibana节点	kibana
version	String	集群版本	7.4.2
slotType	String	资源配置类型	bes.c3.c1m2
slotDescription	String	资源配置类型描述	CPU 1核, 内存 2G, SSD 0G
actualInstanceNum	Integer	实例数量	1

InstancesItem字段数据结构说明

参数名称	参数类型	描述	示例值
instanceId	String	节点实例id	572979729246851074
status	String	状态	Running
moduleType	String	节点类型	kibana
moduleVersion	String	版本	7.4.2
hostIp	String	ip地址	172.16.16.5

ClusterHealth字段数据结构说明

参数名称	参数类型	描述	示例值
status	String	健康状态	GREEN(健康)、YELLOW(亚健康)、RED(不健康)
totalDiskSize	String	磁盘总大小	1TB
usedDiskSize	String	磁盘已使用大小	1GB
nodeDiskMaxSizeInGB	NodeDiskMaxSizeInGB	节点所占磁盘	
indexDiskSize	String	索引大小	1MB

NodeDiskMaxSizeInGB字段数据结构说明

参数名称	参数类型	描述	参数类型
es_node	String	数据节点	25
es_cold_tier_node	String	冷数据节点	0

Billing字段数据结构说明

参数名称	参数类型	描述	示例值
paymentType	String	计费类型	postpay

NetworkItem字段数据结构说明

参数名称	参数类型	描述	示例值
subnetId	String	子网ID	e6c8bc3a-0b17-43ac-a748-98a37e44b39d
subnet	String	子网	系统预定义子网B(172.16.16.0/20)
availableZone	String	可用区	zoneB

TagsItem字段数据结构说明

参数名称	参数类型	描述	
tagKey	String	标签key	test
tagValue	String	标签value	111

Log字段数据结构说明

参数名称	参数类型	描述	示例值
mainLog	Boolean	主日志是否开启	true
gcLog	Boolean	gc日志是否开启	true
indexSlowLog	Boolean	索引慢日志是否开启	true
searchSlowLog	Boolean	查询慢日志是否开启	true
auditLog	Boolean	访问日志是否开启	true

ResGroupListItem字段数据结构说明

参数名称	参数类型	描述	示例值
groupId	String	资源分组id	RESG-44pvmsruNiy
groupName	String	资源分组名称	默认分组

请求示例

```
 POST /api/bes/cluster/v2/detail
<公共请求头>
x-Region: bd


{
  "clusterId" : "12345"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "subnet" : "系统预定义子网B(172.16.16.0/20)",
    "kibanaEip" : "kibanaEip",
    "instances" : [ {
      "instanceId" : "572979729246851074",
      "moduleType" : "kibana",
      "hostIp" : "172.16.16.5",
      "moduleVersion" : "7.4.2",
      "status" : "Running"
    }, {
      "instanceId" : "572979729246851074",
      "moduleType" : "kibana",
      "hostIp" : "172.16.16.5",
      "moduleVersion" : "7.4.2",
      "status" : "Running"
    } ],
    "log" : {
      "gcLog" : true,
```

```
    "mainLog" : true,
    "indexSlowLog" : true,
    "auditLog" : true,
    "searchSlowLog" : true
  },
  "clusterId" : "572979545519558656",
  "billing" : {
    "paymentType" : "postpay"
  },
  "network" : [ {
    "subnetId" : "e6c8bc3a-0b17-43ac-a748-98a37e44b39d",
    "subnet" : "系统预定义子网B(172.16.16.0/20)",
    "availableZone" : "zoneB"
  }, {
    "subnetId" : "e6c8bc3a-0b17-43ac-a748-98a37e44b39d",
    "subnet" : "系统预定义子网B(172.16.16.0/20)",
    "availableZone" : "zoneB"
  } ],
  "actualStatus" : "active",
  "clusterName" : "postpaynew1",
  "vpcId" : "1b9c28ef-0874-414f-8a0c-aa85796ffffb",
  "inspectAuthorized" : "true",
  "enableKibanaSSL" : false,
  "enableEsSSL" : false,
  "desireStatus" : "active",
  "resGroupList" : [ {
    "groupName" : "默认分组",
    "groupId" : "RESG-44pvmsruNiy"
  }, {
    "groupName" : "默认分组",
    "groupId" : "RESG-44pvmsruNiy"
  } ],
  "bdVersion" : "1.7.6",
  "esUrl" : "http: //172.16.16.6: 8200",
  "kibanaUrl" : "https: //172.16.16.7: 8080",
  "vpc" : "默认私有网络(172.16.0.0/16)",
  "securityGroup" : "默认安全组",
  "clusterHealth" : {
    "totalDiskSize" : "1TB",
    "usedDiskSize" : "1GB",
    "nodeDiskMaxSizeInGB" : {
      "es_node" : "25",
      "es_cold_tier_node" : "0"
    },
    "indexDiskSize" : "1MB",
    "status" : "GREEN(健康)、YELLOW(亚健康)、RED(不健康)"
  },
  "modules" : [ {
    "slotDescription" : "CPU 1核, 内存 2G, SSD 0G",
    "slotType" : "bes.c3.c1m2",
    "actualInstanceNum" : 1,
    "type" : "kibana",
    "version" : "7.4.2"
  }, {
    "slotDescription" : "CPU 1核, 内存 2G, SSD 0G",
    "slotType" : "bes.c3.c1m2",
    "actualInstanceNum" : 1,
    "type" : "kibana",
    "version" : "7.4.2"
  } ],
  "tags" : [ {
    "tagValue" : "111",
    "tagKey" : "test"
```

```
      "tagKey" : "test"
    }, {
      "tagValue" : "111",
      "tagKey" : "test"
    } ],
    "adminUsername" : "superuser",
    "availableZone" : "zoneB",
    "expireTime" : "",
    "exporterUrl" : "http://172.16.16.7:8081",
    "esEip" : "esEip",
    "region" : "bd"
  },
  "success" : true,
  "status" : 200
}
```


错误码

请参考通用错误码

新增节点类型

接口说明

新增类型节点，比如新增专有主节点。

请求URI

POST /api/bes/cluster/addModule

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	57090069477786009 6
modules	List	是	模板信息	
resizeMode	String	是	添加方式，如果resizeMode为空会报"Http Message Not Readable"。SCROLL：滚动模式，SMOOTH：蓝绿模式，COLD：冷重启模式	SCROLL
paymentType	String	是	计费方式	postpay表示后付费、prepay表示预付费

ModulesItem字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
type	String	是	节点类型，es_node：数据节点，kibana：kibana节点，es_dedicated_master：专用主节点，es_cold_tier_node：冷数据节点，es_coordinate_node：协调节点	es_dedicated_master
version	String	是	集群版本	7.4.2
desireInstanceNum	Integer	是	节点数量	3
slotType	String	是	机型	bes.c3.c1m2
diskSlotInfo	DiskSlotInfo	是	磁盘信息	

DiskSlotInfo字段数据结构说明

参数名称	必填	参数类型	描述	示例值
type	是	String	磁盘类型；ssd表示高效云盘、premium_ssd表示ssd云盘	premium_ssd
size	是	BigDecimal	磁盘大小；单位G，最小为50G，premium_ssd最大为6144G， ssd最大为20480G	50

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
orderId	String	订单ID	186f2566f2884534a65ed8cdc2499d82

请求示例

```
 POST /api/bes/cluster/addModule
<公共请求头>
x-Region: bd


{"modules":
[{"type":"es_dedicated_master","version":"7.4.2","desireInstanceNum":3,"slotType":"bes.g3.c2m8","diskSlotInfo":
{"type":"premium_ssd","size":20,"share":false,"instanceNum":3}}],"paymentType":"postpay","resizeMode":"SCROLL","clus
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "orderId" : "186f2566f2884534a65ed8cdc2499d82"
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

删除集群

接口说明

删除集群

请求URI

POST /api/bes/cluster/v2/delete

请求头参数

参数名	类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824
refundReason	String	否	退款原因，预付费集群必填	业务测试结束

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result	结果	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
orderId	String	预付费退款订单id，后付费为空	9afda69e8e1f4811a6ff554f115162e6

请求示例

```
POST /api/bes/cluster/v2/delete
<公共请求头>
x-Region: bd


{
  "refundReason" : "业务测试结束",
  "clusterId" : "218185657699405824"
}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "orderId" : "9afda69e8e1f4811a6ff554f115162e6"
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

上传NLP分词词典

接口描述

支持用户通过api配置NLP分词词典

上传词典接口

请求路径 POST /api/bes/cluster/nlp_dict/update **请求参数 Header参数**

参数名	类型	是否必须	描述	示例值
x-Region	String	是	区域	bd
authorization	String	是	认证字符串，计算方法请参考生成认证字符串。	bce-auth-v1/{accessKey}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}

Body参数

参数名	类型	是否必须	描述	示例值
clusterId	是	String	集群id	5729795455195586 56
clusterId	是	String	代表nlp词典的配置模式，可选参数：upload_dict（上传词典文件），text（输入文本），reset_dict（重置词典）。	
fileName	否	String	上传的词典文件名称，当配置模式选择上传词典文件（upload_dict）时必须填写。	test.txt
bucketName	否	String	BOS的Bucket的名称，当配置模式选择上传词典文件（upload_dict）时必须填写。	test_bucket
bucketKey	否	String	词典文件在BOS Bucket中的存储路径，当配置模式选择上传词典文件（upload_dict）时必须填写。	/test_path/.../test.txt
separator	否	String	上下文分隔符规则，默认值为tab。可选参数：tab、space(空格)。例如：（1）干预规则为tab： 中华(tab)民(空格)国 -- 切词结果：中华，民(空格)国。（2）干预规则为空格：中华(tab)民(空格)国 -- 切词结果：中华，民，国	
dictContent	否	String	直接输入文本作为词典，当配置方式选择输入文本（text）时必须填写。内容中可以使用"\n"代表分行符，使用"[]"作为短语干预符，使用"\t"、空格来代表上下文分隔符，且限制1000000个字符。	中华 人民 共和国\n中华 [人民 共和国]\n中华\n人民\n共和国

请求示例

```
POST /api/bes/cluster/nlp_dict/update
{
  "clusterId": 1111111,
  "mode": "upload_dict",
  "fileName": "test.txt",
  "bucketName": "test_bucket",
  "bucketKey": "/test_path/.../test.txt",
  "separator": "tab"
}
```

响应参数

参数	必填	类型	描述
success	是	Boolean	标识请求是否成功，true/false
status	是	String	状态码
result	是	Object	词典详情

Result字段数据结构说明

参数	必填	类型	描述
code	是	String	错误码
message	是	String	请求详情

响应示例

```
{
  "success": true,
  "status": 200,
  "result": {
      "code": "Success",
      "message": ""
  }
}
```

错误详情

错误码（code）	请求详情（message）	描述
SUCCESS	-	请求成功
INVALID_ENTITY_TOO_LARGE	The dictContent exceeds the 1000000 character limit.	当配置方式选择输入文本（text）时，dictContent内容超过1000000字符限制
INVALID_ENTITY_TOO_LARGE	The dict file is too large.	当配置模式选择上传词典文件（upload_dict）时，词典文件超过10M
INVALID_CLUSTER_ID	The cluster id is not valid.	集群id不存在
CLUSTER_NEED_UPGRADE	The ES Version need more than 7.x.	ES至少需要7.x版本以上
CLUSTER_NEED_UPGRADE	The ES Node memory requires at least 64g.	ES节点内存配置至少需要64G
INVALID_PARAMETER	The value of request parameter is invalid, Please check the required field .	请求参数有误，请检查必填项或字段格式
INVALID_NLP_DICT_FILE	The dictionary file is not a txt file.	词典文件不是txt文件
PERMISSION_DENY	Permission deny.	权限不足，需要BES OPERATE权限
INVALID_FILE	The bucket or file not exist or banned.	bucket或文件不存在或不允许访问
UNKNOW	Unknow error.	未知错误

查看词典接口

请求路径 POST /api/bes/cluster/nlp_dict/update **请求参数 Header参数**

参数名	类型	是否必须	描述	示例值
x-Region	String	是	区域	bd
authorization	String	是	认证字符串，计算方法请参考生成认证字符串。	bce-auth-v1/{accessKey}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}

Body参数

参数名	类型	是否必须	描述	示例值
clusterId	String	是	集群id	572979545519558656

请求示例

```
  POST /api/bes/cluster/nlp_dict/display
{
  "clusterId": 1111111
}
```

响应参数

参数	必填	类型	描述
success	是	Boolean	标识请求是否成功，true/false
status	是	String	状态码
result	是	Object	词典详情

Result字段数据结构说明

参数	必填	类型	描述
operationType	是	String	upload_dict（上传词典文件），text（输入文本）
separator	是	String	上下文分隔符规则
fileName	是	String	若使用上传词典的方式，则此字段有值，代表上传词典的名称
dictContent	是	String	若使用输入文本的方式，则此字段有值，代表详细词典内容

响应示例

```
  {
    "success": true,
    "status": 200,
    "result": {
        "operationType": "upload_dict",
        "separator": "tab",
        "fileName": "test.txt",
        "dictContent": null
    }
}
```

获取操作历史信息

请求URI

POST /api/bes/cluster/cluster_tasks

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	否	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	807521220013723648

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	
result	Result		

Result字段数据结构说明

参数名称	参数类型	描述	示例值
clusterId	String	集群id	
appTasks	List	主任务列表	

AppTasksItem字段数据结构说明

参数名称	参数类型	描述	示例值
subTasks	List	子任务列表	
type	String	主任务类型	RESTART
detail	Detail	主任务详情	
progress	Integer	主任务进度	100
status	String	主任务状态	NOT_START/RUNNING/FAILED/FINISH
start_timestamp	String	主任务开始时间戳	1692098199445
end_timestamp	String	主任务结束时间戳	1692098232160

SubTasksItem字段数据结构说明

参数名称	参数类型	描述	示例值
type	String	子任务类型	SCALE_IN
detail	Detail	子任务详情	
progress	BigDecimal	子任务进度	50
status	String	子任务状态	NOT_START/RUNNING/FAILED/FINISH
taskList	List		
start_timestamp	String	子任务开始时间戳	1692098199445
end_timestamp	String	子任务结束时间戳	1692098232160

TaskListItem字段数据结构说明

参数名称	参数类型	描述	示例值
name	String	检查项名称	clusterRemoteConnectSettingCheck
type	String	类型	cluster
level	String	级别	1
status	String	状态	1 通过；-1不通过
finishTime	String	结束时间	1699264859155
message	String	提示	cluster has no snapshot
failedIndices	List	失败索引	

请求示例

```
 POST /api/bes/cluster/cluster_tasks
<公共请求头>
x-Region: bj


{
    "clusterId":"807521220013723648"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "appTasks" : [ {
      "end_timestamp" : "1692098232160",
      "start_timestamp" : "1692098199445",
      "subTasks" : [ {
        "end_timestamp" : "1692098232160",
        "start_timestamp" : "1692098199445",
        "progress" : 50,
        "taskList" : [ {
          "finishTime" : "1699264859155",
          "level" : "1",
          "name" : "clusterRemoteConnectSettingCheck",
          "failedIndices" : [ "failedIndices", "failedIndices" ],
          "type" : "cluster",
          "message" : "cluster has no snapshot",
          "status" : "1 通过；-1不通过"
        }, {
          "finishTime" : "1699264859155",
          "level" : "1",
          "name" : "clusterRemoteConnectSettingCheck",
          "failedIndices" : [ "failedIndices", "failedIndices" ],
          "type" : "cluster",
          "message" : "cluster has no snapshot",
          "status" : "1 通过；-1不通过"
        } ],
        "detail" : { },
        "type" : "SCALE_IN",
        "status" : "NOT_START/RUNNING/FAILED/FINISH"
      }, {
        "end_timestamp" : "1692098232160",
        "start_timestamp" : "1692098199445",
        "progress" : 50,
        "taskList" : [ {
          "finishTime" : "1699264859155",
          "level" : "1",
          "name" : "clusterRemoteConnectSettingCheck",
          "failedIndices" : [ "failedIndices", "failedIndices" ],
          "type" : "cluster",
          "message" : "cluster has no snapshot",
          "status" : "1 通过；-1不通过"
        }, {
          "finishTime" : "1699264859155",
          "level" : "1",
          "name" : "clusterRemoteConnectSettingCheck",
          "failedIndices" : [ "failedIndices", "failedIndices" ],
          "type" : "cluster",
```

```
      "message" : "cluster has no snapshot",
      "status" : "1 通过；-1不通过"
    } ],
    "detail" : { },
    "type" : "SCALE_IN",
    "status" : "NOT_START/RUNNING/FAILED/FINISH"
  } ],
  "progress" : 100,
  "detail" : { },
  "type" : "RESTART",
  "status" : "NOT_START/RUNNING/FAILED/FINISH"
}, {
  "end_timestamp" : "1692098232160",
  "start_timestamp" : "1692098199445",
  "subTasks" : [ {
    "end_timestamp" : "1692098232160",
    "start_timestamp" : "1692098199445",
    "progress" : 50,
    "taskList" : [ {
      "finishTime" : "1699264859155",
      "level" : "1",
      "name" : "clusterRemoteConnectSettingCheck",
      "failedIndices" : [ "failedIndices", "failedIndices" ],
      "type" : "cluster",
      "message" : "cluster has no snapshot",
      "status" : "1 通过；-1不通过"
    }, {
      "finishTime" : "1699264859155",
      "level" : "1",
      "name" : "clusterRemoteConnectSettingCheck",
      "failedIndices" : [ "failedIndices", "failedIndices" ],
      "type" : "cluster",
      "message" : "cluster has no snapshot",
      "status" : "1 通过；-1不通过"
    } ],
    "detail" : { },
    "type" : "SCALE_IN",
    "status" : "NOT_START/RUNNING/FAILED/FINISH"
  }, {
    "end_timestamp" : "1692098232160",
    "start_timestamp" : "1692098199445",
    "progress" : 50,
    "taskList" : [ {
      "finishTime" : "1699264859155",
      "level" : "1",
      "name" : "clusterRemoteConnectSettingCheck",
      "failedIndices" : [ "failedIndices", "failedIndices" ],
      "type" : "cluster",
      "message" : "cluster has no snapshot",
      "status" : "1 通过；-1不通过"
    }, {
      "finishTime" : "1699264859155",
      "level" : "1",
      "name" : "clusterRemoteConnectSettingCheck",
      "failedIndices" : [ "failedIndices", "failedIndices" ],
      "type" : "cluster",
      "message" : "cluster has no snapshot",
      "status" : "1 通过；-1不通过"
    } ],
    "detail" : { },
    "type" : "SCALE_IN",
    "status" : "NOT_START/RUNNING/FAILED/FINISH"
```

```
      } ],
      "progress" : 100,
      "detail" : { },
      "type" : "RESTART",
      "status" : "NOT_START/RUNNING/FAILED/FINISH"
    } ],
    "clusterId" : "clusterId"
  },
  "success" : "true",
  "status" : 0
}
```

错误码

请参考通用错误码

获取数据量观测数据

请求URI

POST /api/bes/cluster/data_size_tendency

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	111111111111111
indexPrefix	String	否	索引前缀(默认为空字符串)	index
datePattern	String	是	日期格式	yyyy-MM-dd
times	Integer	是	时间范围	7
timeUnit	String	是	时间单位(枚举)	HOURS/MINUTES/DAYS/MONTHS/YEARS

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	是否请求成功	true
status	Integer	状态码	200
result	Result		

Result字段数据结构说明

参数名称	参数类型	描述	示例值
clusterId	String	集群id	
data	List		

DataItem字段数据结构说明

参数名称	参数类型	描述	示例值
byteSize	String	总数据字节数	11059
indices	List	索引信息	
size	String	总数据量大小	10.7KB
time	String	具体时间	20230919
timeMills	String	时间戳	1695115967995

IndicesItem字段数据结构说明

参数名称	参数类型	描述	示例值
byteSize	String	总数据字节数	11059
index	String	索引名称	index
size	String	总数据量大小	10.7KBindex

请求示例

```
 POST /api/bes/cluster/data_size_tendency
<公共请求头>
x-Region: bj


{
    "clusterId":"811064791824207872",
    "indexPrefix":"",
    "datePattern":"yyyyMMdd",
    "times":7,
    "timeUnit":"DAYS"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "data" : [ {
      "indices" : [ {
        "byteSize" : "11059",
        "size" : "10.7KB",
        "index" : "index"
      }, {
        "byteSize" : "11059",
        "size" : "10.7KB",
        "index" : "index"
      } ],
      "byteSize" : "11059",
      "size" : "10.7KB",
      "timeMills" : "1695115967995",
      "time" : "20230919"
    }, {
      "indices" : [ {
        "byteSize" : "11059",
        "size" : "10.7KB",
        "index" : "index"
      }, {
        "byteSize" : "11059",
        "size" : "10.7KB",
        "index" : "index"
      } ],
      "byteSize" : "11059",
      "size" : "10.7KB",
      "timeMills" : "1695115967995",
      "time" : "20230919"
    } ],
    "clusterId" : "clusterId"
  },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

重置密码

请求URI

POST /api/bes/cluster/password/reset

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	443755843180171264
newPassword	String	是	新密码	Kic123456
confirmPassword	String	是	确认密码	Kic123456

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/password/reset
<公共请求头>
x-Region: bj


{
    "clusterId":"443755843180171264",
    "newPassword":"Kic123456",
    "confirmPassword":"Kic123456"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

是否开启Grafana监控

请求URI

POST /api/bes/cluster/monitor_state

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	807521220013723648
enableMonitor	Boolean	是	是否开启监控	false

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/monitor_state
<公共请求头>
x-Region: bj


{
    "clusterId":"807521220013723648",
    "enableMonitor":true
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

开启和关闭https

接口说明

开启https、关闭https

请求URI

POST /api/bes/cluster/v2/esProtocolTransform

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824
enableSSL	Boolean	是	true：开启https，false：关闭https	true

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	String	常量字符串	

请求示例

```
 POST /api/bes/cluster/v2/esProtocolTransform
<公共请求头>
x-Region: bd


{
  "enableSSL" : true,
  "clusterId" : "218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

集群blb信息

接口说明

查询集群blb ip信息

请求URI

POST /api/bes/cluster/blbInfo

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	否	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	807521220013723648

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result	blb信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
clusterId	String	集群ID	572979545519558656
moduleInfoList	List	blb mobule信息	

ModuleInfoListItem字段数据结构说明

参数名称	参数类型	描述	示例值
moduleType	String	module 类型	es_node
enableSSL	Boolean	是否开启https	false
blbInfoList	List	blb 信息	

BlbInfoListItem字段数据结构说明

参数名称	参数类型	描述	示例值
blbInternalIp	String	blb 访问ip	
blbFloatingIp	String	blb floatingip	

请求示例

```
 POST /api/bes/cluster/blbInfo
<公共请求头>
x-Region: bd



{
 "clusterId": "644734225693675520"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "clusterId" : "572979545519558656",
    "moduleInfoList" : [ {
      "enableSSL" : false,
      "moduleType" : "es_node",
      "blbInfoList" : [ {
        "blbFloatingIp" : "blbFloatingIp",
        "blbInternalIp" : "blbInternalIp"
      }, {
        "blbFloatingIp" : "blbFloatingIp",
        "blbInternalIp" : "blbInternalIp"
      } ]
    }, {
      "enableSSL" : false,
      "moduleType" : "es_node",
      "blbInfoList" : [ {
        "blbFloatingIp" : "blbFloatingIp",
        "blbInternalIp" : "blbInternalIp"
      }, {
        "blbFloatingIp" : "blbFloatingIp",
        "blbInternalIp" : "blbInternalIp"
      } ]
    } ]
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

实例相关接口

启动实例

接口说明

启动实例V2

请求URI

POST /api/bes/cluster/v2/instance/start

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	218185657699405824
instanceId	String	是	实例id	276316903033671680

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/v2/instance/start
<公共请求头>
x-Region: bd


{
  "instanceId" : "276316903033671680",
  "clusterId" : "218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

停止实例

接口说明

停止实例V2

请求URI

POST /api/bes/cluster/v2/instance/stop

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	218185657699405824
instanceId	String	是	实例id	276316903033671680

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/v2/instance/stop
<公共请求头>
x-Region: bd


{
  "instanceId" : "276316903033671680",
  "clusterId" : "218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

删除实例

请求URI

POST /api/bes/cluster/v2/instance/delete

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	218185657699405824
instanceIdList	List	是	需要删除的实例id列表，缩容后最少有两个节点	
moduleType	String	是	缩容节点的类型，容节点数量仅支持 es_cold_tier_node：冷数据节点 es_node：数据节点 es_coordinate_node:协调节点	es_node

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	BigDecimal	状态码	200
result	Result		

Result字段数据结构说明

参数名称	参数类型	描述	示例值
orderId	String	缩容订单id	186f2566f2884534a65ed8cdc2499d82

请求示例

```
 POST /api/bes/cluster/v2/instance/delete
<公共请求头>
x-Region: bj


{
"clusterId":"111",
"instanceIdList":["222"],
"moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "orderId" : "186f2566f2884534a65ed8cdc2499d82"
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

批量启动实例

接口说明

批量启动实例

请求URI

POST /api/bes/cluster/v2/instance/batchStart

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	218185657699405824
instanceId	String	是	实例id	276316903033671680

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/v2/instance/batchStart
<公共请求头>
x-Region: bd


{
  "instanceIdList" : ["276316903033671680"],
  "clusterId" : "218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

查看缩容节点列表

请求URI

POST /api/bes/cluster/v2/scalein/instances

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	218185657699405824
moduleType	String	是	节点类型，缩容节点数量仅支持 es_cold_tier_node：冷数据节点 es_node：数据节点 es_coordinate_node:协调节点	es_cold_tier_node

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	BigDecimal	状态码	200
result	Result		

Result字段数据结构说明

参数名称	参数类型	描述	示例值
instanceInfoList	List	可缩容的节点列表	

InstanceInfoListItem字段数据结构说明

参数名称	参数类型	描述	示例值
availableZone	String	可用区	zoneA
instanceList	List	节点数据	

InstanceListItem字段数据结构说明

参数名称	参数类型	描述	示例值
instanceId	String	实例id	798832817483157504
hostIp	String	实例ip	x.x.x.x
isMaster	Boolean	是否为主节点	true
isEmpty	Boolean	节点数据是否为空	true

请求示例

```
 POST /api/bes/cluster/v2/scalein/instances
<公共请求头>
x-Region: bj


{
"clusterId":"111",
"moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "instanceInfoList" : [ {
      "availableZone" : "zoneA",
      "instanceList" : [ {
        "instanceId" : "798832817483157504",
        "hostIp" : "x.x.x.x",
        "isMaster" : true,
        "isEmpty" : true
      }, {
        "instanceId" : "798832817483157504",
        "hostIp" : "x.x.x.x",
        "isMaster" : true,
        "isEmpty" : true
      } ]
    }, {
      "availableZone" : "zoneA",
      "instanceList" : [ {
        "instanceId" : "798832817483157504",
        "hostIp" : "x.x.x.x",
        "isMaster" : true,
        "isEmpty" : true
      }, {
        "instanceId" : "798832817483157504",
        "hostIp" : "x.x.x.x",
        "isMaster" : true,
        "isEmpty" : true
      } ]
    } ]
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

数据迁移节点列表

请求URI

POST /api/bes/cluster/migrate/v1/instances

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	21818565769940582 4
moduleType	String	是	节点类型，数据迁移仅支持 es_cold_tier_node：冷数据节点 es_node：数据节点	es_cold_tier_node

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	BigDecimal	状态码	200
result	Result		

Result字段数据结构说明

参数名称	参数类型	描述	示例值
instanceInfoList	List	可迁移的节点列表	

InstanceInfoListItem字段数据结构说明

参数名称	参数类型	描述	示例值
availableZone	String	可用区	zoneA
instanceList	List	节点数据	

InstanceListItem字段数据结构说明

参数名称	参数类型	描述	示例值
instanceId	String	实例id	798832817483157504
hostIp	String	实例ip	x.x.x.x
isMaster	Boolean	是否为主节点	true

请求示例

```
 POST /api/bes/cluster/migrate/v1/instances
<公共请求头>
x-Region: bj


{
"clusterId":"111",
"moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "instanceInfoList" : [ {
      "availableZone" : "zoneA",
      "instanceList" : [ {
        "instanceId" : "798832817483157502",
        "hostIp" : "x.x.x.x",
        "isMaster" : true
      }, {
        "instanceId" : "798832817483157502",
        "hostIp" : "x.x.x.x",
        "isMaster" : true
      } ]
    }, {
      "availableZone" : "zoneA",
      "instanceList" : [ {
        "instanceId" : "798832817483157502",
        "hostIp" : "x.x.x.x",
        "isMaster" : true
      }, {
        "instanceId" : "798832817483157502",
        "hostIp" : "x.x.x.x",
        "isMaster" : true
      } ]
    } ]
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

数据迁移系统建议

请求URI

POST /api/bes/cluster/migrate/v1/suggestInstances

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	218185657699405824
moduleType	String	是	节点类型，数据迁移仅支持 es_cold_tier_node：冷数据节点 es_node：数据节点	es_cold_tier_node
migrateCount	Integer	是	迁移节点数量，节点数量需小于当前节点数量	2

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	BigDecimal	状态码	200
result	Result		

Result字段数据结构说明

参数名称	参数类型	描述	示例值
suggestList	List	建议迁移节点	

SuggestListItem字段数据结构说明

参数名称	参数类型	描述	示例值
availableZone	String	可用区	zoneA
instanceList	List	节点列表	

InstanceListItem字段数据结构说明

参数名称	参数类型	描述	示例值
instanceId	String	实例id	798832817483157502
hostIp	String	实例ip	x.x.x.x

请求示例

```
 POST /api/bes/cluster/migrate/v1/suggestInstances
<公共请求头>
x-Region: bj



{
"clusterId":"111",
"moduleType":"es_node",
"migrateCount":2
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "suggestList" : [ {
      "availableZone" : "zoneA",
      "instanceList" : [ {
        "instanceId" : "798832817483157502",
        "hostIp" : "x.x.x.x"
      }, {
        "instanceId" : "798832817483157502",
        "hostIp" : "x.x.x.x"
      } ]
    }, {
      "availableZone" : "zoneA",
      "instanceList" : [ {
        "instanceId" : "798832817483157502",
        "hostIp" : "x.x.x.x"
      }, {
        "instanceId" : "798832817483157502",
        "hostIp" : "x.x.x.x"
      } ]
    } ]
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

数据迁移

请求URI

POST /api/bes/cluster/migrate/v1/confirm

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	218185657699405824
moduleType	String	是	节点类型,数据迁移仅支持 es_cold_tier_node：冷数据节点 es_node：数据节点	es_cold_tier_node
instanceIdList	List	是	实例id列表	

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	BigDecimal	状态码	200
result	Result	数据	

请求示例

```
 POST /api/bes/cluster/migrate/v1/confirm
<公共请求头>
x-Region: bj


{
"clusterId":"111",
"instanceIdList":["222"],
"moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

数据迁移回滚

请求URI

POST /api/bes/cluster/migrate/v1/rollback

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	218185657699405824
requestId	String	是	调用数据迁移后的请求id，过滤哪个请求(可以在操作历史接口拿到)	1689582177835

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	BigDecimal	状态码	200
result	Result	数据	

请求示例

```
 POST /api/bes/cluster/migrate/v1/rollback
<公共请求头>
x-Region: bj


{
"clusterId":"111",
"requestId":"222"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

定时调度相关接口

创建和更新定时调度任务

接口说明

创建定时调度任务，请求参数参加api文档https://cloud.baidu.com/doc/BES/s/ol6uozs08

请求URI

POST /api/bes/cluster/schedule/create

请求头参数

参数名	类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	必填	类型	描述	示例值
clusterId	是	String	集群ID	570900694777860096
schedule	是	String	任务的定期或周期调度	0 0/5 * * *？*每5min执行一次任务，0 30 1 * ?*：每天的1:30am执行任务
scheduleName	是	String	定时调度任务名称，支持字母(a-z及A-Z)和数字(0-9),长度6~24个字符	test123
taskType	是	String	定时调度任务类型	CREATE_INDEX
task	是	Task	任务具体参数，不同任务类型配置相应的详细配置	参见api文档https://cloud.baidu.com/doc/BES/s/ol6uozs08

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	String	返回信息	

请求示例

```
 POST /api/bes/cluster/schedule/create
<公共请求头>
x-Region: bd


以创建索引为例
{
 "clusterId": "665018667943202816",
 "schedule": "0 1 0 * * ?",
 "scheduleName": "CreateIndexApi1",
 "taskType": "CREATE_INDEX",
 "task": {
  "indices": [{
   "indexPrefix": "apiindex1-",
   "indexSuffix": "yyyy.MM.dd.HH.mm",
   "indexInfo": {
    "settings": {
     "index": {
      "number_of_shards": "5",
      "number_of_replicas": "1"
     }
    }
   }
  }]
 }
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

查看定时调度任务

请求URI

POST /api/bes/cluster/schedule/list

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID，超过规定18位会返回500错误	570900694777860096

响应体参数

参数名称	参数类型	描述	示例值
success	boolean	请求是否成功	true
status	integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
clusterId	String	集群ID	570900694777860096
schedule	String	任务的定期或周期调度	0 0/5 * * *？：每5min执行一次任务，0 30 1 * ？：每天的1:30am执行任务
scheduleName	String	定时调度任务名称，支持字母(a-z及A-Z)和数字(0-9),长度6~24个字符	test123
taskType	String	定时调度任务类型	CREATE_INDEX
taskRunTimes	String	执行次数	
taskFailTimes	String	失败次数	
lastExecuteMillis	String	上次执行时间	
modifiedMillis	String	修改时间	
scheduleId	String	任务id	
modifiedVersion	String	修改版本	
status	String	状态	
task	Task	任务具体参数	

请求示例

```
 POST /api/bes/cluster/schedule/list
<公共请求头>
x-Region: bd


{
 "clusterId": "665018667943202816"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
 "success": true,
 "status": 200,
 "result": {
  "schedules": [{
   "schedule": "0 4 0 * * ?",
   "scheduleName": "IndexColdApi100",
   "taskType": "MIGRATE_COLD",
   "task": {
    "indexPatterns": ["apiindex*"],
    "minIndexAge": "48h",
    "coldStorage": "BOS"
   },
   "taskRunTimes": 0
```

```
  "taskRunTimes": 0,
  "taskFailTimes": 0,
  "lastExecuteMillis": 0,
  "modifiedMillis": 1658912883143,
  "modifiedVersion": 1,
  "status": "RUNNING"
 }, {
  "schedule": "0 7 0 * * ?",
  "scheduleName": "SettingApi100",
  "taskType": "CLUSTER_SETTINGS",
  "task": {
   "persistent": {
    "bpack.migrate.max_bytes_per_sec": "40MB"
   },
   "transient": {}
  },
  "taskRunTimes": 0,
  "taskFailTimes": 0,
  "lastExecuteMillis": 0,
  "modifiedMillis": 1658914322424,
  "modifiedVersion": 1,
  "status": "RUNNING"
 }, {
  "schedule": "0 5 0 * * ?",
  "scheduleName": "RolloverApi100",
  "taskType": "ROLLOVER",
  "task": {
   "alias": ["apiindex1"],
   "maxSize": "10kb",
   "maxDocs": 100000,
   "maxAge": "100h"
  },
  "taskRunTimes": 0,
  "taskFailTimes": 0,
  "lastExecuteMillis": 0,
  "modifiedMillis": 1658914129397,
  "modifiedVersion": 1,
  "status": "RUNNING"
 }, {
  "schedule": "0 3 0 * * ?",
  "scheduleName": "ForcemergeApi100",
  "taskType": "FORCEMERGE",
  "task": {
   "indexPatterns": ["apiindex*"],
   "minIndexAge": "100h",
   "maxNumSegments": 1
  },
  "taskRunTimes": 0,
  "taskFailTimes": 0,
  "lastExecuteMillis": 0,
  "modifiedMillis": 1658911986313,
  "modifiedVersion": 1,
  "status": "RUNNING"
 }, {
  "schedule": "0 8 0 * * ?",
  "scheduleName": "BackupApi100",
  "taskType": "SNAPSHOT",
  "task": {
   "repository": "esRes",
   "snapshot": "snaptest",
   "retention": {
    "expireAfter": "90d",
    "minCount": "0",
```

```
    "maxCount": "2147483647"
   },
   "taskParams": {
    "indices": "apiIndex1"
   }
  },
  "taskRunTimes": 0,
  "taskFailTimes": 0,
  "lastExecuteMillis": 0,
  "modifiedMillis": 1658914384638,
  "modifiedVersion": 1,
  "status": "RUNNING"
 }, {
  "schedule": "0 2 0 * * ?",
  "scheduleName": "DelIndexApi100",
  "taskType": "DELETE",
  "task": {
   "indexPatterns": ["apiindex*"],
   "minIndexAge": "24h"
  },
  "taskRunTimes": 0,
  "taskFailTimes": 0,
  "lastExecuteMillis": 0,
  "modifiedMillis": 1658911939815,
  "modifiedVersion": 1,
  "status": "RUNNING"
 }, {
  "schedule": "0 6 0 * * ?",
  "scheduleName": "StoreLimitApi100",
  "taskType": "INDEX_STORE_MANAGER",
  "task": {
   "indexPatterns": ["apiindex*"],
   "maxSize": "100gb"
  },
  "taskRunTimes": 0,
  "taskFailTimes": 0,
  "lastExecuteMillis": 0,
  "modifiedMillis": 1658915010970,
  "modifiedVersion": 1,
  "status": "RUNNING"
 }, {
  "schedule": "0 1 0 * * ?",
  "scheduleName": "CreateIndexApi100",
  "taskType": "CREATE_INDEX",
  "task": {
   "indices": [{
    "indexPrefix": "apiindex1-",
    "indexSuffix": "yyyy.MM.dd.HH.mm",
    "indexInfo": {
     "settings": {
      "index": {
       "number_of_shards": "1",
       "number_of_replicas": "1"
      }
     }
    }
   }]
  },
  "taskRunTimes": 0,
  "taskFailTimes": 0,
  "lastExecuteMillis": 0,
  "modifiedMillis": 1658911611317,
```

```
    "modifiedVersion": 1,
    "status": "RUNNING"
   }]
  }
 }
```

错误码

请参考通用错误码

删除定时调度任务

请求URI

POST /api/bes/cluster/schedule/delete

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID，超过规定18位会返回500错误	570900694777860096
scheduleName	String	是	定时调度任务名称，支持字母(a-z及A-Z)和数字(0-9),长度6~24个字符	test123

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	String	返回信息	

请求示例

```
 POST /api/bes/cluster/schedule/delete
<公共请求头>
x-Region: bd


{
 "clusterId": "665018667943202816",
 "scheduleName": "BackupApi1"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

自动续费相关接口

续费列表

接口说明

获取续费列表

请求URI

POST /api/bes/cluster/renew/list

请求头参数

参数名称	参数类型	是否必填	描述	示例值
x-Region	String	是	区域	bd

请求体参数 请求体字段数据结构说明

参数名称	参数类型	是否必填	描述	示例值
order	String	是	排序规则	desc
orderBy	String	是	排序字段	expireTime
pageNo	Integer	是	页数	1
pageSize	Integer	是	页大小	15
daysToExpiration	Integer	是	与当前时间比较，还有几天过期。-1为不限制	60

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
page	Page	集群列表	

Page字段数据结构说明

参数名称	参数类型	描述	示例值
orderBy	String	排序字段	expireTime
order	String	排序方式	desc
pageNo	Integer	页数	1
pageSize	Integer	页大小	15
totalCount	Integer	集群数量	1
result	List	集群信息	

ResultItem字段数据结构说明

参数名称	参数类型	描述	示例值
clusterId	String	集群ID	573343274681634816
clusterName	String	集群名称	postpaynew2
region	String	区域	bd
expiredTime	String	到期时间	2021-11-27T03: 03: 54Z
clusterStatus	String	集群状态	RUNNING

请求示例

```
 POST /api/bes/cluster/renew/list
<公共请求头>
x-Region: bd


{
  "pageNo" : 1,
  "daysToExpiration" : 60,
  "orderBy" : "expireTime",
  "pageSize" : 15,
  "order" : "desc"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "success" : true,
  "page" : {
    "result" : [ {
      "clusterName" : "postpaynew2",
      "clusterId" : "573343274681634816",
      "region" : "bd",
      "expiredTime" : "2021-11-27T03: 03: 54Z",
      "clusterStatus" : "RUNNING"
    }, {
      "clusterName" : "postpaynew2",
      "clusterId" : "573343274681634816",
      "region" : "bd",
      "expiredTime" : "2021-11-27T03: 03: 54Z",
      "clusterStatus" : "RUNNING"
    } ],
    "pageNo" : 1,
    "orderBy" : "expireTime",
    "pageSize" : 15,
    "totalCount" : 1,
    "order" : "desc"
  },
  "status" : 200
}
```

错误码

请参考通用错误码

续费操作

请求URI

POST /api/bes/cluster/renew?orderType=RENEW

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	
time	Integer	是	续费时长，单位:月	

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
orderId	String	订单ID	186f2566f2884534a65ed8cdc2499d82

请求示例

```
 POST /api/bes/cluster/renew?orderType=RENEW
<公共请求头>
x-Region: 区域


{
  "clusterId" : "clusterId",
  "time" : 0
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "orderId" : "186f2566f2884534a65ed8cdc2499d82"
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

创建自动续费规则

接口说明

创建自动续费规则

请求URI

POST /api/bes/cluster/auto_renew_rule/create

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterIds	List	是	集群ID列表	
renewTimeUnit	String	是	续费时间单位 支持month，year	month
renewTime	Integer	是	续费时长	1
serviceType	String	是	服务类型	BES

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
orderId	String	订单ID	186f2566f2884534a65ed8cdc2499d82

请求示例

```
 POST /api/bes/cluster/auto_renew_rule/create
<公共请求头>
x-Region: bd


{
  "serviceType" : "BES",
  "renewTime" : 1,
  "renewTimeUnit" : "month",
  "clusterIds" : [ "443755843180171264" ]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "orderId" : "186f2566f2884534a65ed8cdc2499d82"
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

查看自动续费规则列表

接口说明

获取自动续费规则列表

请求URI

POST /api/bes/cluster/auto_renew_rule/list

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	否	集群id	123
serviceType	String	是	服务类型	BES

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	List	返回信息	

ResultItem字段数据结构说明

参数名称	参数类型	描述	示例值
uuid	String	续费规则的ID	rule-f38743879d554275a71d2d737362ab40
userId	String	用户ID	9485f297f8a64d8da79dde31791c08a6
clusterId	String	集群ID	573343274681634816
region	String	区域	bd
renewTimeUnit	String	续费时间单位 支持month，year	month
renewTime	Integer	续费时长	1
createTime	String	创建时间	2021-10-27T06: 51: 18Z
updateTime	String	修改时间	1971-01-01T00: 00: 01Z

请求示例

```
 POST /api/bes/cluster/auto_renew_rule/list
<公共请求头>
x-Region: bd


{
  "serviceType" : "BES",
  "clusterId" : "123"
}
```

响应示例

```
  HTTP/1.1 200 OK
<公共响应头>

{
  "result" : [ {
    "renewTime" : 1,
    "createTime" : "2021-10-27T06: 51: 18Z",
    "renewTimeUnit" : "month",
    "updateTime" : "1971-01-01T00: 00: 01Z",
    "clusterId" : "573343274681634816",
    "region" : "bd",
    "uuid" : "rule-f38743879d554275a71d2d737362ab40",
    "userId" : "9485f297f8a64d8da79dde31791c08a6"
  }, {
    "renewTime" : 1,
    "createTime" : "2021-10-27T06: 51: 18Z",
    "renewTimeUnit" : "month",
    "updateTime" : "1971-01-01T00: 00: 01Z",
    "clusterId" : "573343274681634816",
    "region" : "bd",
    "uuid" : "rule-f38743879d554275a71d2d737362ab40",
    "userId" : "9485f297f8a64d8da79dde31791c08a6"
  } ],
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

查看自动续费

接口说明

查看自动续费

请求URI

POST /api/bes/cluster/auto_renew_rule/detail

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	443755843180171264

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
clusterId	String	集群ID	573343274681634816
renewTimeUnit	String	续费时间单位 支持month，year	month
renewTime	BigDecimal	续费时长	1
expireTime	BigDecimal	自动续费时间	1658216735000
nextRenewTime	BigDecimal	下次自动续费时间	1658216735000

请求示例

```
 POST /api/bes/cluster/auto_renew_rule/detail
<公共请求头>
x-Region: bd


{
  "clusterId" : "443755843180171264"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "renewTime" : 1,
    "expireTime" : 1658216735000,
    "renewTimeUnit" : "month",
    "clusterId" : "573343274681634816",
    "nextRenewTime" : 1658216735000
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

更新自动续费规则

接口说明

修改自动续费规则

请求URI

POST /api/bes/cluster/auto_renew_rule/update

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	443755843180171264
renewTimeUnit	String	是	续费时间单位 支持month，year	month
renewTime	Integer	是	续费时长	1

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	String	常量字符串	

请求示例

```
 POST /api/bes/cluster/auto_renew_rule/update
<公共请求头>
x-Region: bd


{
  "renewTime" : 1,
  "renewTimeUnit" : "month",
  "clusterId" : "443755843180171264"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

删除自动续费规则

接口说明

删除自动续费规则

请求URI

POST /api/bes/cluster/auto_renew_rule/delete

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	否	集群ID	443755843180171264

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result		

Result字段数据结构说明

参数名称	参数类型	描述	示例值
expireTime	BigDecimal	自动续费时间	1658216735000

请求示例

```
 POST /api/bes/cluster/auto_renew_rule/delete
<公共请求头>
x-Region: bd


{
  "clusterId" : "443755843180171264"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "expireTime" : 1658216735000
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

配置修改相关接口

配置修改

请求URI

POST /api/bes/cluster/config/update

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	否	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	807521220013723648
esConfigsMap	EsConfigsMap	是	es节点配置(全量更新)	{"cluster.publish.timeout": “180s”}
kibanaConfigsMap	KibanaConfigsMap	是	kibana节点配置(全量更新)	{"ops.interval": 100}

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/config/update
<公共请求头>
x-Region: bj


{
    "clusterId": "644734225693675520",
    "esConfigsMap":{
        "cluster.publish.timeout": "180s"
    },
    "kibanaConfigsMap":{
        "ops.interval": 100
    }
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

上传同义词配置文件

请求URI

POST /api/bes/cluster/{clusterId}/synonym_dict/upload

参数名称	参数类型	是否必须	描述	示例值	参数位置
clusterId	String	是	集群id	807521220013723648	Path

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
file	File	是	自定义插件文件(只支持.txt文件)	xxx.txt

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/807521220013723648/synonym_dict/upload
<公共请求头>
x-Region: bj


{
    "file":""
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

删除同义词配置文件

请求URI

POST /api/bes/cluster/synonym_dict/delete

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	807521220013723648
synonymDictName	String	是	要删除的文件	xxx.txt

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/synonym_dict/delete
<公共请求头>
x-Region: bj


{
 "clusterId": "644734225693675520",
 "synonymDictName":"xxx.txt"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

获取同义词文件列表

请求URI

POST /api/bes/cluster/synonym_dict/list

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	807521220013723648

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	List	同义词列表	

ResultItem字段数据结构说明

参数名称	参数类型	描述	示例值
synonym_dict_name	String	同义词配置名称	xxx.txt

请求示例

```
 POST /api/bes/cluster/synonym_dict/list
<公共请求头>
x-Region: bj


{
 "clusterId": "644734225693675520"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : [ {
    "synonym_dict_name" : "xxx.txt"
  }, {
    "synonym_dict_name" : "xxx.txt"
  } ],
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

查看集群配置

请求URI

POST /api/bes/cluster/config/info

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	807521220013723648

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result	config信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
esNodeExtraConfig	String	es节点信息	cluster.publish.timeout: 180s
kibanaExtraConfig	String	kibana节点信息	ops.interval: 100

请求示例

```
 POST /api/bes/cluster/config/info
<公共请求头>
x-Region: bj


{
 "clusterId": "644734225693675520"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "kibanaExtraConfig" : "ops.interval: 100",
    "esNodeExtraConfig" : "cluster.publish.timeout: 180s"
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

插件配置接口

获取默认和自定义插件列表

请求URI

POST /api/bes/cluster/plugin/info

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	807521220013723648

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	操作是否成功	true
status	Integer	状态码	200
result	Result		

Result字段数据结构说明

参数名称	参数类型	描述	示例值
plugins	List	自定义插件列表	
defaultPlugins	List	系统默认插件列表	

PluginsItem字段数据结构说明

参数名称	参数类型	描述	示例值
moduleType	String	模块类型	es_node
pluginDesc	String	插件描述	Baidu NLP中文分词插件
pluginName	String	插件名称	xxx.zip
pluginStatus	String	插件状态	UNINSTALLED

DefaultPluginsItem字段数据结构说明

参数名称	参数类型	描述	示例值
moduleType	String	模块类型	es_node
pluginDesc	String	插件描述	custom
pluginName	String	插件名称	xxx.zip
pluginStatus	String	插件状态	UNINSTALLED
pluginOperation	PluginOperation	插件操作	

PluginOperation字段数据结构说明

参数名称	参数类型	描述	示例值
enable	Boolean	是否可操作	true
operation	String	操作类型	uninstall
text	String	操作名称	卸载

请求示例

```
 POST /api/bes/cluster/plugin/info
<公共请求头>
x-Region: bj


{
 "clusterId": "644734225693675520"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "defaultPlugins" : [ {
      "pluginDesc" : "custom",
      "moduleType" : "es_node",
      "pluginName" : "xxx.zip",
      "pluginOperation" : {
        "enable" : true,
        "text" : "卸载",
        "operation" : "uninstall"
      },
      "pluginStatus" : "UNINSTALLED"
    }, {
      "pluginDesc" : "custom",
      "moduleType" : "es_node",
      "pluginName" : "xxx.zip",
      "pluginOperation" : {
        "enable" : true,
        "text" : "卸载",
        "operation" : "uninstall"
      },
      "pluginStatus" : "UNINSTALLED"
    } ],
    "plugins" : [ {
      "pluginDesc" : "Baidu NLP中文分词插件",
      "moduleType" : "es_node",
      "pluginName" : "xxx.zip",
      "pluginStatus" : "UNINSTALLED"
    }, {
      "pluginDesc" : "Baidu NLP中文分词插件",
      "moduleType" : "es_node",
      "pluginName" : "xxx.zip",
      "pluginStatus" : "UNINSTALLED"
    } ]
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

安装系统默认插件

请求URI

POST /api/bes/cluster/default_plugin/install

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	807521220013723648
pluginName	String	是	插件名称	baidu-rate-limiting
moduleType	String	是	模块类型	es_node

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	String	返回信息	

请求示例

```
 POST /api/bes/cluster/default_plugin/install
<公共请求头>
x-Region: bj


{
    "clusterId":"644734225693675520",
    "pluginName":"baidu-rate-limiting",
    "moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 0
}
```

错误码

请参考通用错误码

卸载系统默认插件

请求URI

POST /api/bes/cluster/default_plugin/uninstall

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	否	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	807521220013723648
pluginName	String	是	插件名称	baidu-rate-limiting
moduleType	String	是	模块类型	es_node

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	String	返回信息	

请求示例

```
 POST /api/bes/cluster/default_plugin/uninstall
<公共请求头>
x-Region: bj


{
    "clusterId":"644734225693675520",
    "pluginName":"baidu-rate-limiting",
    "moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 0
}
```

错误码

请参考通用错误码

安装自定义插件

请求URI

POST /api/bes/cluster/plugin/install

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群ID	807521220013723648
pluginName	String	是	插件名称(只支持zip)	xxx.zip
moduleType	String	是	模块类型	es_node

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/plugin/install
<公共请求头>
x-Region: bj


{
    "clusterId":"644734225693675520",
    "pluginName":"xxx.zip",
    "moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

卸载自定义插件

请求URI

POST /api/bes/cluster/plugin/uninstall

请求头参数

参数名	类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id	807521220013723648
pluginName	String	是	插件名称(只支持zip)	xxx.zip
moduleType	String	是	模块类型	es_node

响应体参数

响应体字段数据结构说明

参数	类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/plugin/uninstall
<公共请求头>
x-Region: bj


{
    "clusterId":"644734225693675520",
    "pluginName":"xxx.zip",
    "moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

删除自定义插件

请求URI

POST /api/bes/cluster/plugin/delete

请求头参数

参数名	类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id	807521220013723648
pluginName	String	是	插件名称(只支持zip)	xxx.zip
moduleType	String	是	模块类型	es_node

响应体参数

响应体字段数据结构说明

参数	类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	String	返回信息	

请求示例

```
 POST /api/bes/cluster/plugin/delete
<公共请求头>
x-Region: bj


{
    "clusterId":"644734225693675520",
    "pluginName":"xxx.zip",
    "moduleType":"es_node"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

上传自定义插件

请求URI

POST /api/bes/cluster/plugin/upload/{clusterId}/{moduleType}

参数名	类型	是否必须	描述	示例值	参数位置
clusterId	String	是	集群id	807521220013723648	Path
moduleType	String	是	模块类型	es_node	Path

请求头参数

参数名	类型	是否必须	描述	示例值
x-Region	String	否	区域	bj

请求体参数

参数名称	参数类型	是否必须	描述	示例值
file	File	是	自定义插件文件(只支持zip)	

响应体参数

响应体字段数据结构说明

参数	类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	String	返回信息	

请求示例

```
 POST /api/bes/cluster/plugin/upload/807521220013723648/es_node
<公共请求头>
x-Region: bj


{
    "file":""
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

上传nlp词典

接口说明

上传nlp词典时如果上传的是一个txt文件，则调用v2接口直接上传文件，其他情况下调用原来的nlp上传接口

请求URI

POST /api/bes/cluster/v2/nlp_dict/update

请求头参数

参数名	类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id	57297954551955 8656
separator	String	是	上下文分隔符规则，默认值为tab。 可选参数：tab、space(空格)。 例如：干预规则为tab： 中华(tab)民(空格)国 切词结果：中华，民(空格)国 干预规则为空格：中华(tab)民(空格)国 切词结果：中华，民，国	
file	File	是	nlp词典文件	

响应体参数

响应体字段数据结构说明

参数	类型	描述	示例值	
success	Boolean	是否成功	true	
status	String	状态码	200	
result	String	请求结果		

请求示例

```
 POST /api/bes/cluster/v2/nlp_dict/update
<公共请求头>
x-Region: bj


{
    "clusterId":"11111111",
    "separator":"tab",
    "file":""
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : "200"
}
```

错误码

请参考通用错误码

查看nlp词典

请求URI

POST /api/bes/cluster/nlp_dict/display

请求头参数

参数名	类型	是否必须	描述	示例值
x-Region	String	否	区域	bj

请求体参数

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id	11111111

响应体参数

响应体字段数据结构说明

参数	参数类型	描述	示例值
success	Boolean	是否成功	true
status	String	状态码	200
result	Result	请求结果	

Result字段数据结构说明

参数	参数类型	描述	示例值
operationType	String	upload_dict（上传词典文件），text（输入文本）	upload_dict
separator	String	上下文分隔符规则	tab
fileName	String	若使用上传词典的方式，则此字段有值，代表上传词典的名称	test.txt
dictContent	String	若使用输入文本的方式，则此字段有值，代表详细词典内容	

请求示例

```
 POST /api/bes/cluster/nlp_dict/display
<公共请求头>
x-Region: bj


{
  "clusterId": "1111111"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
    "success": true,
    "status": 200,
    "result": {
        "operationType": "upload_dict",
        "separator": "tab",
        "fileName": "test.txt",
        "dictContent": null
    }
}
```

错误码

请参考通用错误码

日志管理相关接口

查看日志导出任务记录

请求URI

POST /api/bes/cluster/es_log_view_new/export_record

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	否	集群id	807521220013723648
taskId	String	否	任务id	123456

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
progress	Integer	导出进度百分比。取值为0-100	85
state	String	下载状态。取值为：等待执行/进行中/导出完成/导出失败	导出完成
config	Config	导出任务的配置	

Config字段数据结构说明

参数名称	参数类型	描述	示例值
logType	String	日志类型	gcLog
bosBucket	String	bos桶	mybucket
bosPath	String	bos路径	myPath
startTime	String	日志起始时间戳	1582098232160
endTime	String	日志终止时间戳	1582098233160
region	String	地域	bj

请求示例

```
 POST /api/bes/cluster/es_log_view_new/export_record
<公共请求头>
x-Region: bj


{
    "clusterId":"811453865437302784",
    "taskId":"123456"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "success" : true,
  "status" : 200,
  "progress" : 85,
  "state": "进行中",
  "config": {
      "logType":"gcLog",
      "bosBucket":"mybucket",
      "startTime":1582098232160,
      "endTime":1582098232160,
      "region":"bj",
      "bosPath":"bes_log_repository"
  }
}
```

错误码

请参考通用错误码

日志设置

请求URI

POST /api/bes/cluster/es_log_view_new/settings

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id	807521220013723648
indexSlowLog	Boolean	是	是否开启索引慢日志功能	true
searchSlowLog	Boolean	是	是否开启查询慢日志功能	true
auditLog	Boolean	是	是否开启ES访问日志功能	true

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	String	返回信息	

请求示例

```
 POST /api/bes/cluster/es_log_view_new/settings
<公共请求头>
x-Region: bj


{
 "clusterId":"811453865437302781",
 "auditLog":true,
 "indexSlowLog":false,
 "searchSlowLog":false
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : "result",
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

日志查询

请求URI

POST /api/bes/cluster/es_log_view_new/search

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id	807521220013723648
pageNo	Integer	是	页码	1
pageSize	Integer	是	每页展示数量	10
keyword	String	否	查询关键字	""
startTime	String	是	日志查询起始时间(格式为yyyy-MM-dd HH:mm:ss，起始时间不能超过结束时间，且最小为结束时间前七天)	2023-11-02 14:24:21
endTime	String	是	日志查询结束时间(格式为yyyy-MM-dd HH:mm:ss，不能早于起始时间，且不能晚于当前时间)	2023-11-02 14:24:21
type	String	是	查询日志类型(主日志:mainLog,查询慢日志: searchSlowLog,索引慢日志:indexSlowLog,GC日志:gcLog,ES访问日志:auditLog)	mainLog

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
level	String	日志级别	WARN
host	String	ip	0.0.0.0
time	String	日志时间	2023-11-09T17:11:43,013
content	String	日志内容	xxxx

请求示例

```
 POST /api/bes/cluster/es_log_view_new/search
<公共请求头>
x-Region: bj


{
 "clusterId":"811453865437302784",
 "endTime":"2023-11-09 14:24:21",
 "keyword":"",
 "pageNo":1,
 "pageSize":10,
 "startTime":"2023-11-02 14:24:21",
 "type":"mainLog"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "level" : "WARN",
  "host" : "0.0.0.0",
  "time" : "2023-11-09T17:11:43,013",
  "content" : "xxxx"
}
```

错误码

请参考通用错误码

创建日志导出任务

请求URI

POST /api/bes/cluster/es_log_view_new/export

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	否	集群id	807521220013723648
startTime	String	否	日志查询起始时间(格式为yyyy-MM-dd HH:mm:ss，起始时间不能超过结束时间，且最小为结束时间前七天)	2023-11-02 14:24:21
endTime	String	否	日志查询结束时间(格式为yyyy-MM-dd HH:mm:ss，不能早于起始时间，且不能晚于当前时间)	2023-11-02 14:24:21
types	List	否	导出日志类型列表	["mainLog", "gcLog"]
bosBucket	String	否	bos bucket	my-bucket
bosPath	String	否	bos路径,若不传参，则使用默认值"bes_log_repository"	my-repo1

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
tasks	List	导出任务列表	[{"logType":"mainLog", "taskId":"123456"}]

TasksItem字段数据结构说明

参数名称	参数类型	描述	示例值
logType	String	导出日志的类型	mainLog
taskId	String	任务id	123456

请求示例

```
 POST /api/bes/cluster/es_log_view_new/export
<公共请求头>
x-Region: bj


{
    "clusterId":"811453865437302784",
    "types": ["mainLog"],
    "startTime":"2023-11-02 14:24:21",
    "endTime":"2023-11-03 14:24:21",
    "bosBucket":"my-bucket",
    "bosPath":"my-path"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "success" : true,
  "status" : 200,
  "tasks" : [
     {"logType": "mainLog", "taskId": "12345"}
  ]
}
```

错误码

请参考通用错误码

智能巡检接口

提交手动巡检任务

请求URI

POST /api/bes/cluster/inspect/create_manual

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/inspect/create_manual
<公共请求头>
x-Region: bd


{
    "clusterId":"218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

查询今日已执行完成的手动巡检次数

请求URI

POST /api/bes/cluster/inspect/manual_count

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
count	Integer	今日已执行完成的手动巡检次数	3

请求示例

```
 POST /api/bes/cluster/inspect/manual_count
<公共请求头>
x-Region: bd


{
    "clusterId":"218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "count" : 3
  },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

查看某巡检任务的执行状态和结果

请求URI

POST /api/bes/cluster/inspect/get_task

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824
taskId	String	是	巡检任务id	10

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
state	String	任务的执行状态。"doing"，"success"，“error”，对应“执行中”、“执行成功”、“执行失败”	doing
type	String	巡检任务类型。"manual"、"auto"，对应“手动”、“自动”	manual
indices	String	巡检索引，*代表全部索引	index1
totalCount	Integer	巡检项数量	14
execTime	String	巡检任务执行时间。以北京时间表示	2024-07-20 18:31:42
groups	List	巡检结果按照诊断等级（"safe"，“lowRisk”，“highRisk”，“error”）的分组。	

GroupsItem字段数据结构说明

参数名称	参数类型	描述	示例值
grade	String	巡检项诊断等级。目前包括"safe","lowRisk","highRisk","error"，分别对应"安全"，“低风险”，“高风险”，“巡检失败”	lowRisk
itemCount	Integer	组内巡检项的数量	0
results	List	当state="success"，则results有内容。表示执行结果，JSON字符串。	

ResultsItem字段数据结构说明

参数名称	参数类型	描述	示例值
id	String	巡检项id，即英文名	ClusterHealth
type	String	巡检项类型。包括"cluster"，"node"，"index"，"search"	cluster
conclusion	String	巡检项结果	低风险：当前集群黄色（YELLOW）
hasLowRiskAdvice	Boolean	是否包含低风险的巡检建议(高低风险同时存在时为true)	false
notExistIndex	Boolean	巡检索引是否存在(索引级巡检项失败时用来判断是否是索引不存在导致的失败)	false

请求示例

```
 POST /api/bes/cluster/inspect/get_task
<公共请求头>
x-Region: bd


{
    "clusterId":"218185657699405824",
    "taskId":10
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "indices" : "index1",
    "execTime" : "2024-07-20 18:31:42",
    "groups" : [ {
      "grade" : "lowRisk",
      "results" : [ {
        "conclusion" : "低风险：当前集群黄色（YELLOW）",
        "notExistIndex" : false,
        "id" : "ClusterHealth",
        "type" : "cluster",
        "hasLowRiskAdvice" : false
      }, {
        "conclusion" : "低风险：当前集群黄色（YELLOW）",
        "notExistIndex" : false,
        "id" : "ClusterHealth",
        "type" : "cluster",
        "hasLowRiskAdvice" : false
      } ],
      "itemCount" : 0
    }, {
      "grade" : "lowRisk",
      "results" : [ {
        "conclusion" : "低风险：当前集群黄色（YELLOW）",
        "notExistIndex" : false,
        "id" : "ClusterHealth",
        "type" : "cluster",
        "hasLowRiskAdvice" : false
      }, {
        "conclusion" : "低风险：当前集群黄色（YELLOW）",
        "notExistIndex" : false,
        "id" : "ClusterHealth",
        "type" : "cluster",
        "hasLowRiskAdvice" : false
      } ],
      "itemCount" : 0
    } ],
    "state" : "doing",
    "type" : "manual",
    "totalCount" : 14
  },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

最新一次巡检概况

请求URI

POST /api/bes/cluster/inspect/overview/latest

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
taskId	String	任务id	11
taskStatus	String	任务状态，成功：success，失败：error，只返回成功和失败任务，如果是首次授权，该字段为null	success
countSafe	Integer	安全项个数	1
countLowRisk	Integer	低风险项个数	2
countHighRisk	Integer	高风险项个数	1
countError	Integer	失败项个数	1
execTime	String	巡检任务执行时间。以北京时间表示	2024-07-20 18:31:42

请求示例

```
 POST /api/bes/cluster/inspect/overview/latest
<公共请求头>
x-Region: bd


{
    "clusterId":"218185657699405824"
}
```

响应示例

```
  HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "countLowRisk" : 2,
    "countError" : 1,
    "execTime" : "2024-07-20 18:31:42",
    "countSafe" : 1,
    "taskId" : "11",
    "taskStatus" : "success",
    "countHighRisk" : 1
  },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

近七天巡检概况

请求URI

POST /api/bes/cluster/inspect/overview/weekly

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
count	Integer	近7天已执行完成的巡检次数	1
riskItems	List	有风险的巡检项列表	

RiskItemsItem字段数据结构说明

参数名称	参数类型	描述	示例值
id	String	巡检项id，即英文名称	ClusterHealth
highRiskCount	Integer	7天累计的高风险次数	1
lowRiskCount	Integer	7天累计的低风险次数	1

请求示例

```
 POST /api/bes/cluster/inspect/overview/weekly
<公共请求头>
x-Region: bd



{
    "clusterId":"218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "riskItems" : [ {
      "highRiskCount" : 1,
      "lowRiskCount" : 3,
      "id" : "ClusterHealth"
    }, {
      "highRiskCount" : 1,
      "lowRiskCount" : 3,
      "id" : "ClusterHealth"
    } ],
    "count" : 1
  },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

修改手动巡检任务的配置

接口说明

可选巡检项包括：集群健康状态(ClusterHealth)、集群负载(ClusterPayload)、节点分片数(NodeShardNum)、节点负载(NodePayload)、节点计算资源(NodeCompute)、节点堆内存(NodeHeap)、节点存储资源(NodeDisk)、数据节点负载均衡(NodePayloadBalance)、节点离群(NodeOffline)、集群写入拒绝率(ClusterWriteReject)、查询队列堆积(NodeSearchQueue)、索引副本合理性(IndexReplicaNum)、索引分片合理性(IndexShardSize)、查询耗时(SearchTime)。

请求URI

POST /api/bes/cluster/inspect/update_manual_conf

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824
indices	String	是	索引匹配字符串。填"*"则匹配全部索引	*
items	List	否	巡检项列表。不允许填空列表。不填，则表示不修改巡检项配置	["ClusterHealth","ClusterPayload"]

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/inspect/update_manual_conf
<公共请求头>
x-Region: bd


{
  "indices" : "*",
  "clusterId" : "218185657699405824",
  "items" : "[\"ClusterHealth\",\"ClusterPayload\"]"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

查看手动巡检任务的配置

请求URI

POST /api/bes/cluster/inspect/get_manual_conf

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
indices	String	索引匹配字符串。如果是匹配全部，则返回"*"	*
items	List		

请求示例

```
 POST /api/bes/cluster/inspect/get_manual_conf
<公共请求头>
x-Region: bd



{
    "clusterId":"218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "indices" : "*",
    "items" : [ "[\"ClusterHealth\", \"ClusterPayload\"]", "[\"ClusterHealth\", \"ClusterPayload\"]" ]
  },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

开启或关闭自动巡检

请求URI

POST /api/bes/cluster/inspect/switch_auto

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824
switchOn	Boolean	是	开启/ 关闭自动巡检。true为开启，false为关闭	true

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/inspect/switch_auto
<公共请求头>
x-Region: bd


{
    "clusterId":"218185657699405824",
    "switchOn":true
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

查看集群是否开启自动巡检

请求URI

POST /api/bes/cluster/inspect/check_auto

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
switchOn	Boolean	是否已启用自动巡检	true

请求示例

```
 POST /api/bes/cluster/inspect/check_auto
<公共请求头>
x-Region: bd


{
    "clusterId":"218185657699405824"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "switchOn" : true
  },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

列举所有可选巡检项

接口说明

可选巡检项包括：集群健康状态(ClusterHealth)、集群负载(ClusterPayload)、节点分片数(NodeShardNum)、节点负载(NodePayload)、节点计算资源(NodeCompute)、节点堆内存(NodeHeap)、节点存储资源(NodeDisk)、数据节点负载均衡(NodePayloadBalance)、节点离群(NodeOffline)、集群写入拒绝率(ClusterWriteReject)、查询队列堆积(NodeSearchQueue)、索引副本合理性(IndexReplicaNum)、索引分片合理性(IndexShardSize)、查询耗时(SearchTime)

请求URI

POST /api/bes/cluster/inspect/list_items

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
items	List	巡检项列表	

ItemsItem字段数据结构说明

参数名称	参数类型	描述	示例值
id	String	巡检项id，即英文名	ClusterHealth
type	String	巡检项类型。目前包括"cluster"、“node”、“index”、“search”	cluster

请求示例

```
POST /api/bes/cluster/inspect/list_items
<公共请求头>
x-Region: bd



{ }
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>
```

```
{
    "success": true,
    "status": 200,
    "result": {
        "items": [
            {
                "id": "ClusterHealth",
                "type": "cluster"
            },
            {
                "id": "ClusterPayload",
                "type": "cluster"
            },
            {
                "id": "NodeShardNum",
                "type": "node"
            },
            {
                "id": "NodePayload",
                "type": "node"
            },
            {
                "id": "NodeCompute",
                "type": "node"
            },
            {
                "id": "NodeHeap",
                "type": "node"
            },
            {
                "id": "NodeDisk",
                "type": "node"
            },
            {
                "id": "NodePayloadBalance",
                "type": "node"
            },
            {
                "id": "NodeOffline",
                "type": "node"
            },
            {
                "id": "ClusterWriteReject",
                "type": "cluster"
            },
            {
                "id": "NodeSearchQueue",
                "type": "node"
            },
            {
                "id": "IndexReplicaNum",
                "type": "index"
            },
            {
                "id": "IndexShardSize",
                "type": "index"
            },
            {
                "id": "SearchTime",
                "type": "search"
            }
        ]
```

```
    }
}
```

错误码

请参考通用错误码

查询是否可以提交巡检任务

请求URI

POST /api/bes/cluster/inspect/check_busy

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bd

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id，超过规定18位会返回500错误	218185657699405824

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	String	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

Result字段数据结构说明

参数名称	参数类型	描述	示例值
busy	Boolean	是否已启用true表示当前集群正在执行巡检任务，处于繁忙状态，不能再提交巡检任务。巡检	true

请求示例

```
 POST /api/bes/cluster/inspect/check_busy
<公共请求头>
x-Region: bd



{
    "clusterId":"218185657699405824"
}
```

响应示例

```
  HTTP/1.1 200 OK
<公共响应头>

{
  "result" : {
    "busy" : true
  },
  "success" : "true",
  "status" : 200
}
```

错误码

请参考通用错误码

标签接口

单个集群更新标签

请求URI

POST /api/bes/cluster/updateTags

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterId	String	是	集群id	
tags	List	是	标签组(可为空数组但不可为null)	[{"tagKey":"key","tagValue":"value"}]

TagsItem字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
tagKey	String	是	标签key	key
tagValue	String	是	标签value	value

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/updateTags
<公共请求头>
x-Region: bj


{
    "clusterId":"1111111",
    "tags":[{
        "tagKey":"key",
        "tagValue":"value"
    }]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

批量新增标签

请求URI

POST /api/bes/cluster/batchInsertTags

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

请求体字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
clusterIdList	List	是	集群id数组(至少为两个集群id)	["111","222"]
insertTags	List	是	新增标签组(不可为null和空数组)	[{"tagKey":"key","tagValue":"value"}]

InsertTagsItem字段数据结构说明

参数名称	参数类型	是否必须	描述	示例值
tagKey	String	是	标签key	key
tagValue	String	是	标签value	value

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	Result	返回信息	

请求示例

```
 POST /api/bes/cluster/batchInsertTags
<公共请求头>
x-Region: bj


{
    "clusterIdList":["111","222"],
    "insertTags":[{
        "tagKey":"key",
        "tagValue":"value"
    }]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : { },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

查询标签列表

请求URI

POST /api/bes/cluster/tagList

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	bj

请求体参数

无

响应体参数

响应体字段数据结构说明

参数名称	参数类型	描述	示例值
success	Boolean	请求是否成功	true
status	Integer	状态码	200
result	List	返回信息	

ResultItem字段数据结构说明

参数名称	参数类型	描述	示例值
tagKey	String	标签key	
tagValue	String	标签value	

请求示例

无

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "result" : [ {
    "tagValue" : "tagValue",
    "tagKey" : "tagKey"
  }, {
    "tagValue" : "tagValue",
    "tagKey" : "tagKey"
  } ],
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码

开发指南

Elasticsearch Restful API

Elasticsearch采用REST API， 所有的操作都可通过HTTP API完成，例如增删改查、别名配置等。本文介绍Restful API的使用方法。

Elasticsearch官方参考文档

- 单文档 API：
  - Index
  - Get
  - Delete
  - Update
- 多文档 API：

- Multi Get
- Bulk
- Delete By Query
- Update By Query
- Reindex

使用REST Client交互

客户端访问仅支持HTTP / TCP方式，建议您采用Elasticsearch官方提供的Elasticsearch client。

说明： Elasticsearch官方还提供了多种程序语言的客户端，例如E`Groovy`、`Javascript`、`.NET`、`PHP`、`Perl`、`Python`以及`Ruby`以及很多由社区提供的客户端和插件，您可以在Community Contributed Clients中获取。

使用Java API交互

Elasticsearch为Java用户提供了内置客户端，包括TransportClient和Java REST Client。Elasticsearch 7.0中已经弃用TransportClient，在8.0中完全移除它。因此在实际开发中建议您使用Java REST Client。

- Java Low Level REST Client：Elasticsearch的官方低级客户端，允许通过http与Elasticsearch集群进行通信，需要使用者对http的请求进行编解码。它与所有Elasticsearch版本兼容。
- Java High Level REST Client：Elasticsearch的官方高级客户端，基于Low Level REST Client开发，它提供了更多的API，接受请求对象作为参数并返回响应对象，由客户端自己处理编码和解码。**使用时需要注意，客户端版本需要与Elasticsearch版本相同。**

RESTful API请求参数说明

您可以使用RESTful API或者curl命令，通过8200端口与Elasticsearch进行通信。

curl命令（HTTP）由以下几部分组成。

```
curl -u <USER:PWD> -H<HEADER> -X<METHOD> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

参数	说明
`USER:PWD`	`-u`参数为baidu Elasticsearch的用户名密码，例如：`superuser:mypassword`，如果使用HTTP Basic认证方式则不需要加`-u`参数
`HEADER`	HTTP请求头，例如：`"Content-Type: application/json"`
`METHOD`	HTTP方法，包括`GET`、`POST`、`PUT`、`HEAD`、`DELETE`等。
`PROTOCOL`	`http`或者`https`协议。
`HOST`	Elasticsearch的主机名。
`PORT`	Elasticsearch HTTP服务所在的端口，默认为`8200`。
`PATH`	API路径（例如`/_cluster/health`或者`_cat/nodes`等）。
`QUERY_STRING`	可选的查询请求参数，例如`?pretty`参数可使请求返回的JSON数据更加美观易读。
`BODY`	JSON格式的请求主体（可选）。

RESTful API（HTTP）请求示例

通过以下命令，统计Elasticserach集群中doc的数量。

```
curl -u user:password --header 'Content-Type: application/json' -XGET 'http://host:port/_count' -d '
{
  "query": {
    "match_all": {}
  }
}'
```

返回结果如下。

```
{
  "count": 1646413284,
  "_shards": {
    "total": 1147,
    "successful": 1147,
    "skipped": 0,
    "failed": 0
  }
}
```

Elasticsearch Java客户端

Java REST Client

本章节介绍Elasticsearch的Java Client的原理、版本兼容性以及使用示例，帮助您快速地使用Java客户端与Baidu Elasticsearch集群进行连接，完成查询分析等相关业务。

Java REST Client

Java REST Client有两种类型：

- Java Low Level REST Client：Elasticsearch Client低级别客户端。它允许通过HTTP请求与ES集群进行通信。API本身不负责数据的编码解码，由用户去编码解码。它与所有的ES版本兼容。
- Java High Level REST Client：Elasticsearch Client官方高级客户端。基于低级客户端，主要目标是为了暴露各API特定的方法。Java High Level REST Client依赖于ES核心项目，将Request对象作为参数，返回一个Response对象。所有API都可以同步或异步调用。

REST客户端的请求调用方式有以下两种：

- 同步调用方法，请求后会直接返回一个Response对象，对其进行处理。
- 异步调用方法，一般方法名以async结尾，依赖于监听实现。当有请求返回或是错误返回时，该监听会通知到对应的方法进行处理。

本章节为您讲解Java Client的用法（建议优先使用REST Client），包括：

- High Level REST Client
- Low Level REST Client

High Level REST Client

本文基于Java High Level REST Client 7.x版本，为您介绍Elasticsearch Java API的用法。

准备工作

- 安装Java，要求JDK版本为1.8及以上。
- 创建Baidu Elasticsearch实例，版本7.4.2。

  注意 High Level Client能够向上兼容，但为了保证最大程度地使用最新版客户端的特性，推荐High Level Client版本与集群版

本一致。

- 创建Java Maven工程，并将如下的*pom依赖*添加到Java工程的pom.xml文件中。

pom依赖

```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
    <version>7.4.2</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.7</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.7</version>
</dependency>
```

示例

以下代码为使用Create Index API创建索引，使用异步调用的方式写入文档，删除文档，使用Delete API删除该索引等操作的示例，可供参考。

```
import java.io.IOException;
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.elasticsearch.action.ActionListener;
import org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest;
import org.elasticsearch.action.delete.DeleteRequest;
import org.elasticsearch.action.delete.DeleteResponse;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.index.IndexResponse;
import org.elasticsearch.action.support.master.AcknowledgedResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.CreateIndexRequest;
import org.elasticsearch.client.indices.CreateIndexResponse;
import org.elasticsearch.common.settings.Settings;
import org.elasticsearch.common.xcontent.XContentBuilder;
import org.elasticsearch.common.xcontent.XContentFactory;
import org.elasticsearch.common.xcontent.XContentType;

public class RestClientTest742 {

    public static void main(String[] args) {
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        //访问用户名和密码为您创建Elasticsearch实例时设置的用户名和密码。
        credentialsProvider.setCredentials(AuthScope.ANY, new UsernamePasswordCredentials("{访问用户名}", "{访问密
码}"));
        // 通过builder创建rest client，配置http client的HttpClientConfigCallback。
        // ES HTTP URL 在Baidu Elasticsearch界面中可以查询
```

```
RestClientBuilder builder = RestClient.builder(new HttpHost("{ES HTTP URL}", 8200))
        .setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {
            @Override
            public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder httpClientBuilder) {
                return httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
            }
        });

// RestHighLevelClient实例通过REST low-level client builder进行构造。
RestHighLevelClient client = new RestHighLevelClient(builder);

try {
    //创建索引
    CreateIndexRequest createRequest = new CreateIndexRequest("my_index");

    //创建的每个索引都可以有与之关联的特定设置。
    createRequest.settings(Settings.builder()
            .put("index.number_of_shards", 3)
            .put("index.number_of_replicas", 1)
            .put("index. refresh_interval", "10s")
    );
    // 创建索引时创建文档类型映射
    createRequest.mapping("{\n" +
                "    \"message\": {\n" +
                "      \"properties\": {\n" +
                "        \"message\": {\n" +
                "          \"type\": \"text\"\n" +
                "        }\n" +
                "      }\n" +
                "    }\n" +
                "  }",
            XContentType.JSON);
    // 同步执行创建索引操作
    CreateIndexResponse createIndexResponse = client.indices().create(createRequest, RequestOptions.DEFAULT);
    System.out.println("create index successfully! response : " + createIndexResponse.toString());

    // 写入文档
    // 以XContentBuilder对象的形式提供，Elasticsearch 内部会帮我们生成JSON内容
    XContentBuilder jsonBuilder = XContentFactory.jsonBuilder();
    jsonBuilder.startObject();
    {
        jsonBuilder.field("user", "Kobe Bryant");
        jsonBuilder.field("number", 24);
        jsonBuilder.field("message", "One of the greatest basketball players since the 21st century");
    }
    jsonBuilder.endObject();
    IndexRequest indexRequest = new IndexRequest("my_index").id("1").source(jsonBuilder);
    //异步执行写入文档操作，方法不会阻塞并立即返回。
    ActionListener<IndexResponse> listener = new ActionListener<IndexResponse>() {
        @Override
        public void onResponse(IndexResponse indexResponse) {
            //执行成功时调用。 Response以参数方式提供
            System.out.println("Index doc successfully! response : " + indexResponse.toString());
        }

        @Override
        public void onFailure(Exception e) {
            //在失败的情况下调用。 引发的异常以参数方式提供
            System.out.println("Index doc fail!");
            e.printStackTrace();
        }
    };
```

```
            //异步执行索引请求需要将IndexRequest实例和ActionListener实例传递给异步方法：
            client.indexAsync(indexRequest, RequestOptions.DEFAULT, listener);

            // 删除文档
            DeleteRequest deleteRequest = new DeleteRequest("my_index").id("1");
            // 执行删除文档操作
            DeleteResponse deleteResponse = client.delete(deleteRequest, RequestOptions.DEFAULT);
            System.out.println("Delete doc successfully! \n" + deleteResponse.toString() + "\n" + deleteResponse.status());

            // 指定要删除的索引名称
            DeleteIndexRequest deleteIndexRequest = new DeleteIndexRequest("my_index");
            // 执行删除索引操作
            AcknowledgedResponse deleteIndexResponse = client.indices().delete(deleteIndexRequest,
RequestOptions.DEFAULT);
            System.out.println("delete index successfully! response : " + deleteIndexResponse.toString());
        } catch (IOException ioException) {
            // 异常处理。
            ioException.printStackTrace();
        } finally {
            // 在不需要使用client后，将client关闭。
            try {
                client.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

以上示例代码中带{}的参数需要替换为您具体业务的参数，详情请参见代码注释。

更多Java High Level REST Client的使用特性，请参见Java High Level REST Client官方文档。

Low Level REST Client

本文基于Java Low Level REST Client 7.x版本，为您介绍Elasticsearch Java API的用法。

准备工作

- 安装Java，要求JDK版本为1.8及以上。
- 创建Baidu Elasticsearch实例，版本7.4.2。

  注意 Low Level Client能够与任何版本的Elasticsearch兼容，因此客户端版本可以为任何版本，本文以7.4.2版本为例。
- 创建Java Maven工程，并将如下的*pom依赖*添加到Java工程的pom.xml文件中。

pom依赖

```
 <dependency>
   <groupId>org.elasticsearch.client</groupId>
   <artifactId>rest</artifactId>
   <version>7.4.2</version>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
   <artifactId>httpasyncclient</artifactId>
   <version>4.1.4</version>
</dependency>
<dependency>
   <groupId>org.apache.httpcomponents</groupId>
   <artifactId>httpcore-nio</artifactId>
   <version>4.4.13</version>
</dependency>
<dependency>
   <groupId>org.apache.httpcomponents</groupId>
   <artifactId>httpclient</artifactId>
   <version>4.5.6</version>
</dependency>
<dependency>
   <groupId>org.apache.httpcomponents</groupId>
   <artifactId>httpcore</artifactId>
   <version>4.4.12</version>
</dependency>
<dependency>
   <groupId>commons-codec</groupId>
   <artifactId>commons-codec</artifactId>
   <version>1.10</version>
</dependency>
<dependency>
   <groupId>commons-logging</groupId>
   <artifactId>commons-logging</artifactId>
   <version>1.2</version>
</dependency>
```

示例

以下代码为使用Java Low Level REST ClientI对Elasticsearch进行了查询，可供参考。

```
import java.io.IOException;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.entity.ContentType;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.nio.entity.NStringEntity;
import org.apache.http.util.EntityUtils;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;

public class RestClientTest742 {
    public static void main(String[] args) {
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        //访问用户名和密码为您创建Elasticsearch实例时设置的用户名和密码。
        credentialsProvider.setCredentials(AuthScope.ANY, new UsernamePasswordCredentials("{访问用户名}", "{访问密
码}"));
        // 通过builder创建rest client，配置http client的HttpClientConfigCallback。
        // ES HTTP URL 在Baidu Elasticsearch界面中可以查询
        RestClient restClient = RestClient.builder(new HttpHost("{ES HTTP URL}", 8200))
                .setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {
                    @Override
                    public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder httpClientBuilder) {
                        return httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
                    }
                }).build();
        try {
            // 创建request
            Request request = new Request(
                    "POST",
                    "/{index_name}/_search");
            // 配置请求体
            request.setEntity(new NStringEntity("{\"query\":{\"match_all\":{}}}", ContentType.APPLICATION_JSON));
            Response response = restClient.performRequest(request);
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

以上示例代码中带{}的参数需要替换为您具体业务的参数，详情请参见代码注释。

更多Java Low Level REST Client的使用特性，请参见Java Low Level REST Client官方文档。

Rest Client操作向量索引示例

本文基于Java High Level REST Client 7.x 和 Java Low Level REST Client 7.x 版本，为您介绍Elasticsearch Java API的用法。

准备工作

- 安装Java，要求JDK版本为1.8及以上。
- 创建Baidu Elasticsearch实例，版本7.10.2。

  采用 High Level Client 进行创建索引，写入数据，refresh 以及 删除索引操作， 采用 Low Level Client 执行检索数据操作。

注意 High Level Client能够向上兼容，但为了保证最大程度地使用最新版客户端的特性，推荐High Level Client版本与集群版本一致。

注意 Low Level Client能够与任何版本的Elasticsearch兼容，因此客户端版本可以为任何版本，本文以7.10.2版本为例。

- 创建Java Maven工程，并将如下的*pom依赖*添加到Java工程的pom.xml文件中。

pom依赖

```
<dependency>
  <groupId>org.elasticsearch.client</groupId>
  <artifactId>elasticsearch-rest-high-level-client</artifactId>
  <version>7.10.2</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.7</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.7</version>
</dependency>
<dependency>
  <groupId>org.elasticsearch.client</groupId>
  <artifactId>elasticsearch-rest-client</artifactId>
  <version>7.10.2</version>
</dependency>
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpasyncclient</artifactId>
  <version>4.1.4</version>
</dependency>
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpcore-nio</artifactId>
  <version>4.4.13</version>
</dependency>
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.6</version>
</dependency>
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpcore</artifactId>
  <version>4.4.12</version>
</dependency>
```

示例

以下示例采用 High Level Client 执行创建索引操作、写入数据、refresh 操作以及删除索引操作， 采用 Low Level Client 执行检索数据操作，可供参考。

```
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.entity.ContentType;
import org.apache.http.impl.client.BasicCredentialsProvider;
```

```
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.nio.entity.NStringEntity;
import org.apache.http.util.EntityUtils;
import org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest;
import org.elasticsearch.action.admin.indices.refresh.RefreshRequest;
import org.elasticsearch.action.admin.indices.refresh.RefreshResponse;
import org.elasticsearch.action.bulk.BulkRequest;
import org.elasticsearch.action.bulk.BulkResponse;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.support.master.AcknowledgedResponse;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.indices.CreateIndexRequest;
import org.elasticsearch.client.indices.CreateIndexResponse;
import org.elasticsearch.common.settings.Settings;
import org.elasticsearch.common.xcontent.XContentBuilder;
import org.elasticsearch.common.xcontent.XContentFactory;
import org.elasticsearch.common.xcontent.XContentType;

import java.io.IOException;
import java.util.Random;

public class RestClientTest710 {

    public static void main(String[] args) {
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        //访问用户名和密码为您创建Elasticsearch实例时设置的用户名和密码。
        credentialsProvider.setCredentials(AuthScope.ANY, new UsernamePasswordCredentials("{访问用户名}", "{访问密
码}"));
        // 通过builder创建high level rest client，配置http client的HttpClientConfigCallback。
        // ES HTTP URL 在Baidu Elasticsearch界面中可以查询
        RestClientBuilder builder = RestClient.builder(new HttpHost("{ES HTTP URL}", 8200))
                .setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {
                    @Override
                    public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder httpClientBuilder) {
                        return httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
                    }
                });

        // RestHighLevelClient实例通过REST low-level client builder进行构造。
        RestHighLevelClient client = new RestHighLevelClient(builder);

        // 通过 restClient 进行查询操作
        RestClient restClient = RestClient.builder(new HttpHost("{ES HTTP URL}", 8200))
                .setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {
                    @Override
                    public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder httpClientBuilder) {
                        return httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
                    }
                }).build();

        try {
            //创建索引
            CreateIndexRequest createRequest = new CreateIndexRequest("my_index_vec");

            //创建的每个索引都可以有与之关联的特定设置。
            createRequest.settings(Settings.builder()
                    .put("index.number_of_shards", 2)
                    .put("index.number_of_replicas", 1)
```

```
        .put("index.number_of_replicas", 1)
        .put("index.refresh_interval", "10s")
        .put("knn", true)
    );
    // 创建索引时创建文档类型映射
    createRequest.mapping(
            "{\n" +
            "   \"properties\": {\n" +
            "   \"id\": {\n" +
            "       \"type\": \"long\"\n" +
            "   },\n" +
            "   \"field_hnsw\": {\n" +
            "       \"type\": \"bpack_vector\",\n" +
            "       \"index_type\": \"hnsw\",\n" +
            "       \"dims\": 8,\n" +
            "       \"space_type\": \"l2\",\n" +
            "       \"parameters\": {\n" +
            "           \"m\": 32,\n" +
            "           \"ef_construction\": 256\n" +
            "           }\n" +
            "       }\n" +
            "   }\n" +
            "}",
        XContentType.JSON);

    // 同步执行创建索引操作
    CreateIndexResponse createIndexResponse = client.indices().create(createRequest, RequestOptions.DEFAULT);
    System.out.println("create index successfully! response : " + createIndexResponse.toString());

    // 批次写入文档
    BulkRequest bulkRequest = new BulkRequest();
    for (int i = 0; i < 100; i++) {
        // 以XContentBuilder对象的形式提供，Elasticsearch 内部会帮我们生成JSON内容
        XContentBuilder jsonBuilder = XContentFactory.jsonBuilder();

        jsonBuilder.startObject();
        {
            jsonBuilder.field("id", i);
            jsonBuilder.field("field_hnsw", RandomVector(8));
        }
        jsonBuilder.endObject();
        bulkRequest.add(new IndexRequest("my_index_vec").source(jsonBuilder));
    }
    BulkResponse bulkResponse = client.bulk(bulkRequest, RequestOptions.DEFAULT);
    System.out.println("bulk success ! response : " + bulkResponse.toString());

    //数据写入后会过一定时间触发自动的refresh构建向量索引，手动执行refresh保证可以立即查询
    RefreshRequest refreshRequest = new RefreshRequest("my_index_vec");
    RefreshResponse refreshResponse = client.indices().refresh(refreshRequest, RequestOptions.DEFAULT);
    System.out.println("refresh success ! response : " + refreshResponse.toString());

    // 检索向量数据
    // 创建检索request
    Request request = new Request(
            "POST",
            "/my_index_vec/_search");
    // 配置请求体
    request.setEntity(new NStringEntity("{\n" +
            "  \"size\": 5, \n" +
            "  \"_source\": [\"id\"], \n" +
            "  \"query\": {\n" +
            "    \"knn\": { \n" +
            "      \"field_hnsw\": { \n" +
```

```
                "        \"vector\": [0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9 ,0.1], \n" +
                "        \"k\": 10, \n" +
                "        \"ef\": 256 \n" +
                "      }\n" +
                "    }\n" +
                "  }\n" +
                "}", ContentType.APPLICATION_JSON));
            // 发送检索请求
            Response response = restClient.performRequest(request);
            System.out.println(EntityUtils.toString(response.getEntity()));

            // 指定要删除的索引名称
            DeleteIndexRequest deleteIndexRequest = new DeleteIndexRequest("my_index_vec");
            // 执行删除索引操作
            AcknowledgedResponse deleteIndexResponse = client.indices().delete(deleteIndexRequest,
RequestOptions.DEFAULT);
            System.out.println("delete index successfully! response : " + deleteIndexResponse.toString());
        } catch (IOException ioException) {
            // 异常处理。
            ioException.printStackTrace();
        } finally {
            // 在不需要使用client后，将client关闭。
            try {
                client.close();
                restClient.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    // 生成指定维度随机向量
    public static float[] RandomVector(int dim) {
        float[] vec = new float[dim];
        Random random = new Random();

        // 随机生成数组元素
        for (int i = 0; i < dim; i++) {
            vec[i] = random.nextFloat(1);
        }
        return vec;
    }
}
```

以上示例代码中带{}的参数需要替换为您具体业务的参数，详情请参见代码注释。

更多Java High Level REST Client的使用特性，请参见Java High Level REST Client官方文档。 更多Java Low Level REST Client的使用特性，请参见Java Low Level REST Client官方文档。

通过其他客户端访问Elasticsearch

本文介绍使用PHP、Python和Go客户端访问Baidu Elasticsearch的方法，并提供了示例代码和注意事项供您参考。

PHP语言

> **警告**
>
> Elasticsearch的PHP客户端提供的默认连接池并不适合云上环境，可能在长连接断开时出现连接异常的问题。推荐PHP客户端访问程序使用SimpleConnectionPool作为连接池，并做好失败重连的机制。

通过PHP客户端访问Elasticsearch进行测试，示例代码如下，详情请参见elasticsearch-php。

```
<?php
require 'vendor/autoload.php';
use Elasticsearch\ClientBuilder;

$client = ClientBuilder::create()->setHosts([
  [
    'host'   => '<HOST>',
    'port'   => '8200',
    'scheme' => 'http',
    'user'   => '<USER_NAME>',
    'pass'   => '<PASSWORD>'
  ]
])->setConnectionPool('\Elasticsearch\ConnectionPool\SimpleConnectionPool', [])
  ->setRetries(10)->build();

$indexParams = [
  'index'  => 'my_index',
  'type'   => 'my_type',
  'id'     => '1',
  'body'   => ['testField' => 'abc'],
  'client' => [
    'timeout'         => 10,
    'connect_timeout' => 10
  ]
];
$indexResponse = $client->index($indexParams);
print_r($indexResponse);

$searchParams = [
  'index'  => 'my_index',
  'type'   => 'my_type',
  'body'   => [
    'query' => [
      'match' => [
        'testField' => 'abc'
      ]
    ]
  ],
  'client' => [
    'timeout'         => 10,
    'connect_timeout' => 10
  ]
];
$searchResponse = $client->search($searchParams);
print_r($searchResponse);
?>
```

参数	说明
HOST	ES HTTP URL，在Baidu Elasticsearch界面中可以查询
USER_NAME	用户名。
PASSWORD	密码。

Python语言

通过Python客户端访问Elasticsearch进行测试，示例代码如下，详情请参见elasticsearch-py。

```
 from elasticsearch import Elasticsearch, RequestsHttpConnection
import certifi
es = Elasticsearch(
    ['<HOST>'],
    http_auth=('<USER_NAME>', '<PASSWORD>'),
    port=8200,
    use_ssl=False
)
res = es.index(index="my_index", doc_type="my_type", id=1, body={"title": "One", "tags": ["ruby"]})
print(res['created'])
res = es.get(index="my_index", doc_type="my_type", id=1)
print(res['_source'])
```

Go语言

通过Go语言访问Elasticsearch进行测试，示例代码如下：

- 安装Go编译环境，详情请参见官方文档。以下示例代码使用Go1.13.4版本。
- 创建Baidu Elasticsearch实例。

```
 package main

import (
    "context"
    "gopkg.in/olivere/elastic.v5"
    "gopkg.in/olivere/elastic.v5/config"
    "log"
)

const (
    url      = "http://<HOST>:8200"     //<1>
    username = "<USER_NAME>"        //<2>
    password = "<PASSWORD>"       //<3>
)

func main() {
    var sniff = false     //<4>
    cfg := &config.Config{
        URL:      url,
        Username: username,
        Password: password,
    }

    cfg.Sniff = &sniff
    var client, err = elastic.NewClientFromConfig(cfg)
    if err != nil {
        log.Println(err)
        return
    }

    exists, err := client.IndexExists("index_test").Do(context.Background())   //<5>
    if err != nil {
        log.Println(err)
    }
    log.Println(exists)
}
```

参数	说明
HOST	ES HTTP URL，在Baidu Elasticsearch界面中可以查询
USER_NAME	用户名。
PASSWORD	密码。

连接成功后，返回如下结果。

```
2019/11/29 10:07:37 true
```

更多语言

更多语言示例请参见HTTP/REST Clients and Security。

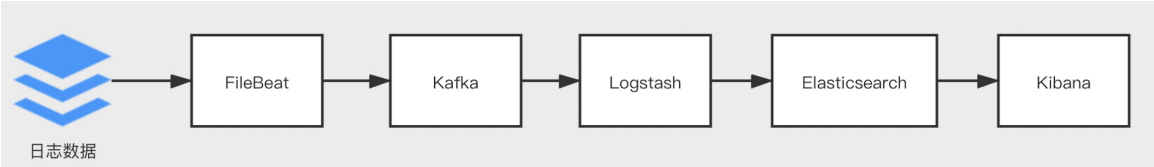
ELK

基于ELK构建日志分析系统

介绍

日志是一个系统不可缺少的组成部分，日志记录系统产生的各种行为，基于日志信息可以对系统进行问题排查，性能优化，安全分析等工作。 使用ELK(Elasticsearch、Logstash、Kibana) 结合Filebeat与Kafka构建日志分析系统，可以方便的对日志数据进行采集解析存储，以及近实时的查询分析。

各组件在日志系统中的职责与关系如下所示：



1. Filebeat：采集原始日志数据，输出数据到Kafka；
2. Kafka：作为消息队列，存储原始日志数据；
3. Logstash：消费Kafka中数据，解析过滤后，将其存储到Elasticsearch中；
4. Elasticsearch：存储用于分析的日志数据，并提供查询、聚合的底层能力；
5. Kibana：提供日志分析的可视化界面；

操作流程：

1. 环境准备

   包括：创建百度Elasticsearch（简称BES）集群、Kibana服务，安装Filebeat、Logstash、Kafka。

2. 配置Filebeat：

   配置Filebeat的input为系统日志，output为Kafka，将日志数据采集到Kafka的指定topic中。

3. 配置Logstash：

   配置Logstash的input为Kafka，output为BES，使用Logstash消费topic中的数据并传输到BES中。

4. 查看日志消费状态：

   在消息队列Kafka中查看日志数据的消费的状态，验证日志数据是否采集成功。

5. 通过Kibana查看日志数据：

在Kibana控制台的Discover页面，查询采集的日志数据。

环境准备

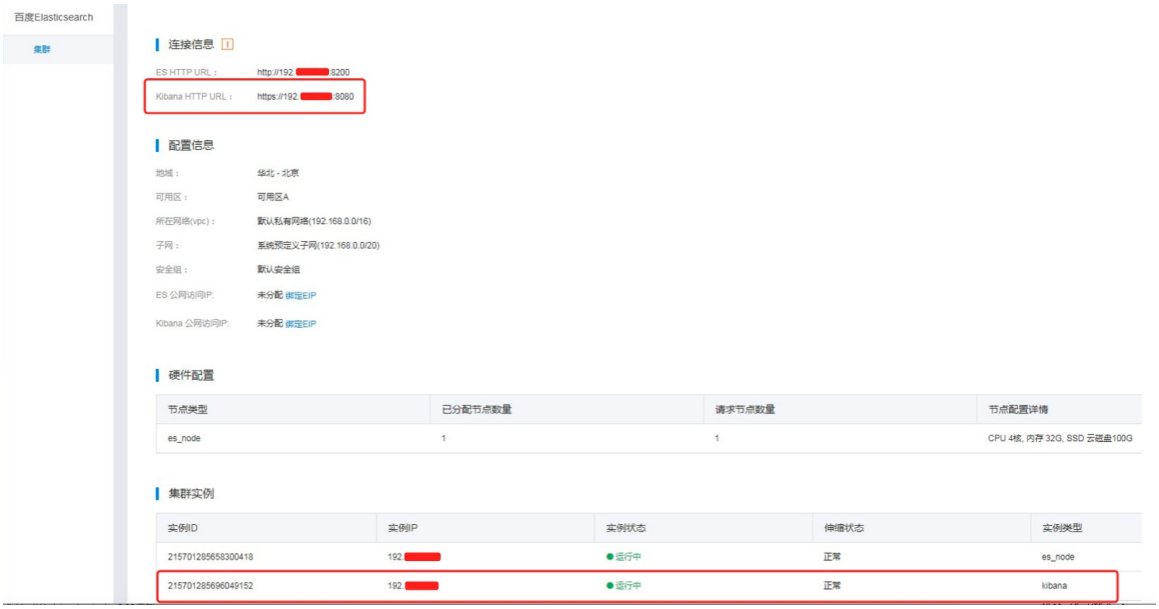
1. 创建BES集群

   参考集群创建中的内容，完成BES集群的创建。

   本文档中创建的BES集群选择7.4.2版本，Filebeat组件、Logstash组件使用的也是7.4.2版本。

2. 创建Kibana服务

   在完成BES集群创建后，除了已购买的BES节点，默认会附赠一个单节点的Kibana服务（Kibana节点支持按需自定义配置），如图所示：



   通过上图连接信息中所示的Kibana HTTP URL可以直接访问Kibana服务。

   Kibana用户名密码与BES服务的用户名密码相同，详见账号使用说明

3. 安装Filebeat

   下载7.4.2的oss版本Filebeat；

```
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-oss-7.4.2-linux-x86_64.tar.gz
```

   解压安装Filebeat；

```
tar -zxvf filebeat-oss-7.4.2-linux-x86_64.tar.gz
```

4. 安装Logstash

   下载7.4.2的oss版本Logstash；

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-oss-7.4.2.tar.gz
```

   解压安装Logstash；

```
tar -zxvf logstash-oss-7.4.2.tar.gz
```

5. 安装、配置Kafka服务

   5.1 下载Zookeeper（Kafka服务依赖Zookeeper）；

```
wget https://mirrors.tuna.tsinghua.edu.cn/apache/zookeeper/zookeeper-3.6.4/apache-zookeeper-3.6.4-bin.tar.gz
```

   5.2 解压安装Zookeeper；

```
tar -zxvf apache-zookeeper-3.6.4-bin.tar.gz
```

5.3 配置Zookeeper；

```
vi conf/zoo.cfg
```

zoo.cfg内容如下所示：

```
tickTime=2000
dataDir=/your_path/apache-zookeeper-3.6.4-bin/data
clientPort=2181
initLimit=5
syncLimit=2
admin.serverPort=8888
```

配置参数说明，如下所示：

参数名称	参数说明
tickTime	服务器之间或客户端与服务器之间维持心跳的时间间隔
dataDir	数据文件目录
clientPort	客户端连接端口
initLimit	Leader-Follower初始通信时限
syncLimit	Leader-Follower同步通信时限
admin.serverPort	内置Jetty绑定的port

5.4 启动Zookeeper；

```
bin/zkServer.sh start
```

5.5 下载Kafka；

```
wget https://archive.apache.org/dist/kafka/2.0.1/kafka_2.12-2.0.1.tgz
```

5.6 解压安装Kafka；

```
tar -zxvf kafka_2.12-2.0.1.tgz
```

5.7 配置Kafka；

```
cd kafka_2.12-2.0.1
vi config/server.properties
```

server.properties内容如下所示：

```
broker.id=0
zookeeper.connect=localhost:2181
```

配置参数说明，如下所示：

参数名称	参数说明
broker.id	Kafka集群实例的唯一标识
zookeeper.connect	Zookeeper的连接地址

5.8 启动Kafka；

```
bin/kafka-server-start.sh config/server.properties
```

5.9 创建名称为 `logdata` 的 topic，用于接收采集数据，命令示例如下所示：

```
bin/kafka-topics.sh --create --zookeeper 127.0.0.1:2181 --replication-factor 2 --partitions 20 --topic logdata
```

命令参数说明，如下所示：

参数名称	参数说明
zookeeper	Zookeeper的连接地址
topic	topic的名称
replication-factor	topic的副本数
partitions	topic的分区数量

配置Filebeat

1.创建Filebeat采集配置文件；

```
 cd filebeat-oss-7.4.2-linux-x86_64
vi filebeat.kafka.yml
```

filebeat.kafka.yml内容如下：

```
 filebeat.inputs:
 - type: log
   enabled: true
   paths:
       - /your_path/*.log

output.kafka:
   hosts: ["**.**.**.**:9092"]
   topic: logdata
   version: 2.0.1
```

配置参数说明，如下所示：

参数名称	参数说明
type	输入类型，设置为log，表示输入源为日志
enabled	input处理器的开关
paths	需要监控的日志文件的路径
hosts	消息队列Kafka的地址
topic	日志输出到消息队列Kafka的topic
version	Kafka的版本

2.启动Filebeat；

```
./filebeat -e -c filebeat.kafka.yml
```

配置Logstash

1.创建配置文件logstash.conf；

```
 cd logstash-oss-7.4.2
vi logstash.conf
```

logstash.conf内容如下：

```
 input {
  kafka {
    bootstrap_servers => "127.0.0.1:9092"
    group_id => "log-data"
    topics => ["logdata"]
    codec => json
  }
}
filter {
}
output {
  elasticsearch {
    hosts => "http://es_ip:es_http_port"
    user =>"your username"
    password =>"your password"
    index => "kafka-%{+YYYY.MM.dd}"
  }
}
```

配置参数说明，如下所示：

参数名称	参数说明
bootstrap_servers	Kafka的地址
group_id	Kafka消费者组id
topics	Kafka的topic名称
codec	数据格式
hosts	BES服务地址
user	BES账号
password	BES账号的密码
index	数据写入的索引

2.启动Logstash；

```
bin/logstash -f logstash.conf
```

更多Logstash的使用说明，详见Logstash使用指南。

查看日志消费状态

使用如下命令查看消费状态；

```
bin/kafka-consumer-groups.sh  --bootstrap-server 127.0.0.1:9092 --group log-data --describe
```

命令参数说明，如下所示：

参数名称	参数说明
bootstrap_servers	Kafka的地址
group	Kafka消费者组

执行结果如下所示：

```
TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID
logdata         15         0               0               0               logstash-0-
logdata         3          0               0               0               logstash-0-
logdata         8          0               0               0               logstash-0-
logdata         0          0               0               0               logstash-0-
logdata         7          1               1               0               logstash-0-
logdata         4          0               0               0               logstash-0-
logdata         13         0               0               0               logstash-0-
logdata         12         1               1               0               logstash-0-
logdata         17         0               0               0               logstash-0-
logdata         1          1               1               0               logstash-0-
logdata         18         0               0               0               logstash-0-
logdata         14         0               0               0               logstash-0-
logdata         2          0               0               0               logstash-0-
logdata         16         1               1               0               logstash-0-
logdata         19         0               0               0               logstash-0-
logdata         9          0               0               0               logstash-0-
logdata         5          1               1               0               logstash-0-
logdata         11         1               1               0               logstash-0-
logdata         6          0               0               0               logstash-0-
logdata         10         0               0               0               logstash-0-
```

结果属性说明，如下所示：

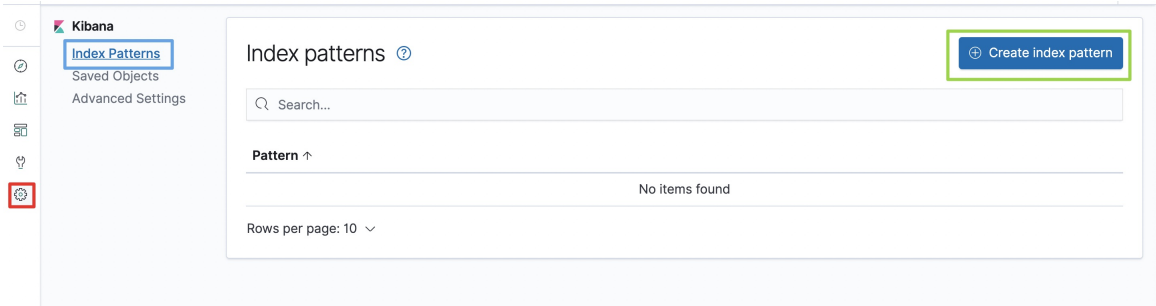
参数名称	参数说明
topic	Kafka的topic名称
partition	topic的分区id
current-offset	consumer消费的具体位移
log-end-offset	partition的最高位移
lag	堆积量
consumer-id	消费者id

通过Kibana查看日志数据

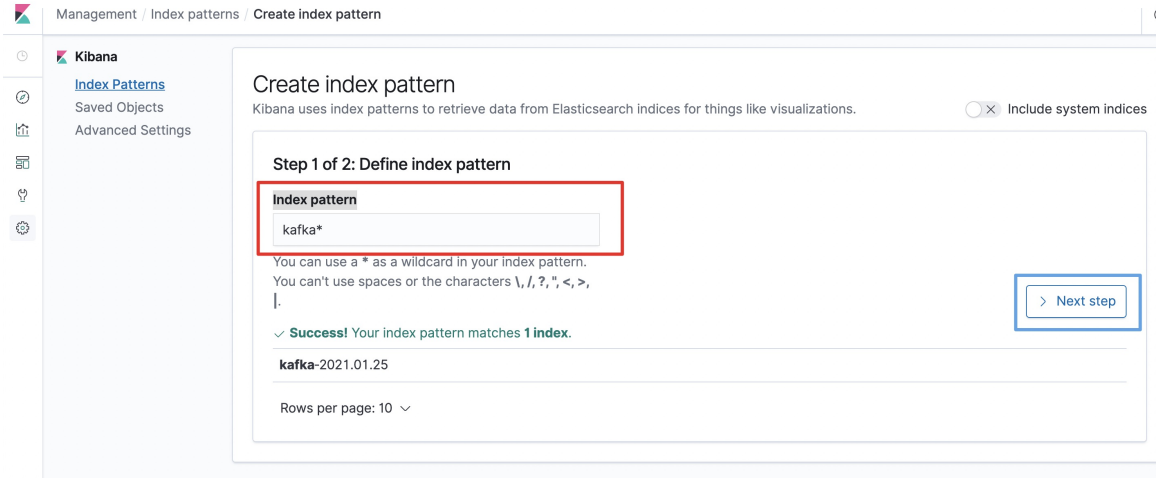
完成以下的配置后，即可通过Kibana查看日志数据。

1.创建索引模式

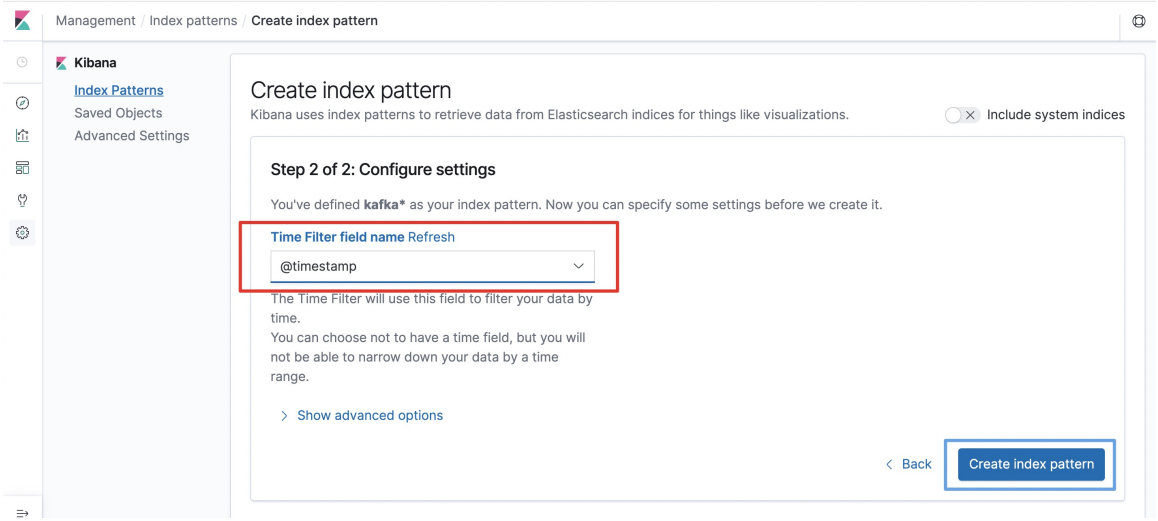
在左侧导航栏，点击Management（下图中红框位置）后，在Kibana区域点击Index Patterns（下图中蓝框位置），然后再点击Create index pattern按钮（下图绿框位置）



在Index pattern处（下图红框位置）填入kafka*，然后点击点击Next Step按钮（下图中蓝框位置）



为Time Filter field name选择@timestamp字段（下图中红框位置），然后点击Create index pattern按钮（下图中蓝框位置）



2.查看日志数据

在左侧导航栏，点击Discover（下图中红框位置）后，即可以查询日志数据

更多Kibana的使用说明，详见Kibana使用指南中的内容。

典型实践

使用BSC将BOS中的数据导入Es

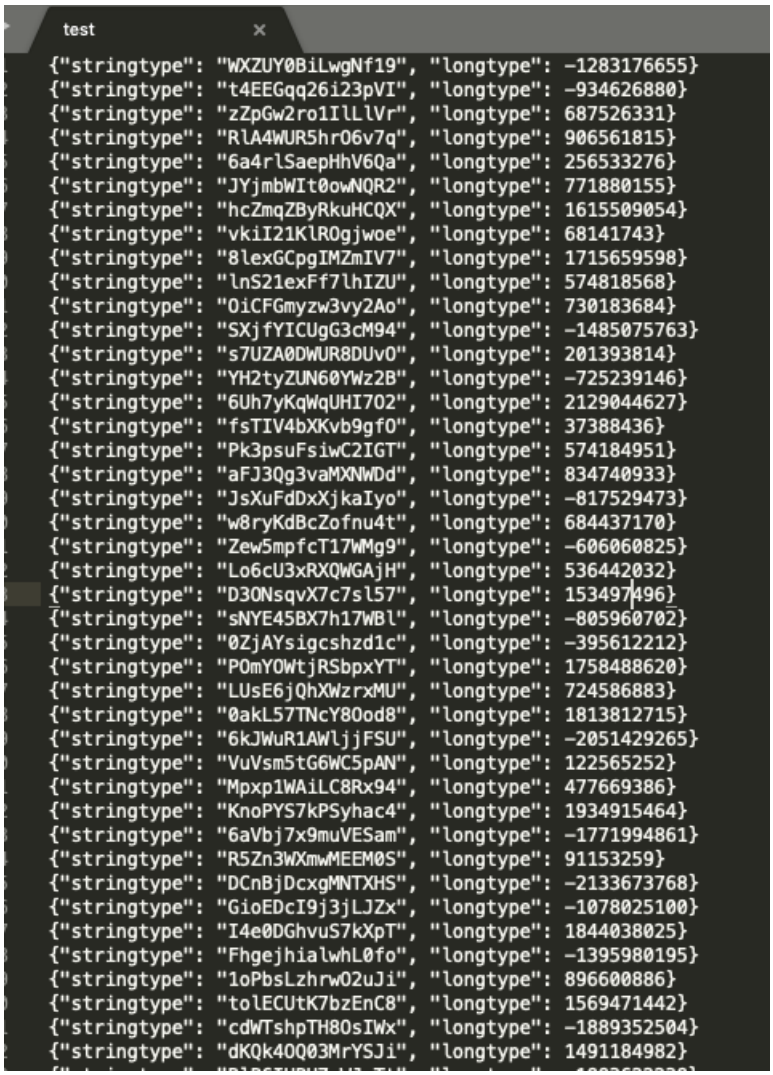
介绍

本文主要介绍通过 BSC【百度流式计算服务】将数据从BOS【百度对象存储系统】中导入到Es中。

上传数据到BOS

登录管理控制台，进入BOS产品界面，创建bucket并上传测试文件：

测试文件的内容如下：



```
test
{"stringtype": "WXZUY0BiLwgNf19", "longtype": -1283176655}
{"stringtype": "t4EEGqq26i23pVI", "longtype": -934626880}
{"stringtype": "zZpGw2ro1IlLlVr", "longtype": 687526331}
{"stringtype": "RlA4WUR5hrO6v7q", "longtype": 906561815}
{"stringtype": "6a4rlSaepHhV6Qa", "longtype": 256533276}
{"stringtype": "JYjmbWIt0owNQR2", "longtype": 771880155}
{"stringtype": "hcZmqZByRkuHCQX", "longtype": 1615509054}
{"stringtype": "vkiI21KlROgjwoe", "longtype": 68141743}
{"stringtype": "8lexGCpgIMZmIV7", "longtype": 1715659598}
{"stringtype": "lnS21exFf7lhIZU", "longtype": 574818568}
{"stringtype": "OiCFGmyzw3vy2Ao", "longtype": 730183684}
{"stringtype": "SXjfYICUgG3cM94", "longtype": -1485075763}
{"stringtype": "s7UZA0DWUR8DUvO", "longtype": 201393814}
{"stringtype": "YH2tyZUN60YWz2B", "longtype": -725239146}
{"stringtype": "6Uh7yKqWqUHI7O2", "longtype": 2129044627}
{"stringtype": "fsTIV4bXKvb9gfO", "longtype": 37388436}
{"stringtype": "Pk3psuFsiwC2IGT", "longtype": 574184951}
{"stringtype": "aFJ3Qg3vaMXNWDd", "longtype": 834740933}
{"stringtype": "JsXuFdDxXjkaIyo", "longtype": -817529473}
{"stringtype": "w8ryKdBcZofnu4t", "longtype": 684437170}
{"stringtype": "Zew5mpfcT17WMg9", "longtype": -606060825}
{"stringtype": "Lo6cU3xRXQWGAjH", "longtype": 536442032}
{"stringtype": "D3ONsqvX7c7sl57", "longtype": 153497496}
{"stringtype": "sNYE45BX7h17WBl", "longtype": -805960702}
{"stringtype": "0ZjAYsigcshzd1c", "longtype": -395612212}
{"stringtype": "POmYOWtjRSbpxYT", "longtype": 1758488620}
{"stringtype": "LUsE6jQhXWzrxMU", "longtype": 724586883}
{"stringtype": "0akL57TNcY8Ood8", "longtype": 1813812715}
{"stringtype": "6kJWuR1AWljjFSU", "longtype": -2051429265}
{"stringtype": "VuVsm5tG6WC5pAN", "longtype": 122565252}
{"stringtype": "Mpxp1WAiLC8Rx94", "longtype": 477669386}
{"stringtype": "KnoPYS7kPSyhac4", "longtype": 1934915464}
{"stringtype": "6aVbj7x9muVESam", "longtype": -1771994861}
{"stringtype": "R5Zn3WXmwMEEM0S", "longtype": 91153259}
{"stringtype": "DCnBjDcxgMNTXHS", "longtype": -2133673768}
{"stringtype": "GioEDcI9j3jLJZx", "longtype": -1078025100}
{"stringtype": "I4e0DGhvuS7kXpT", "longtype": 1844038025}
{"stringtype": "FhgejhialwhL0fo", "longtype": -1395980195}
{"stringtype": "1oPbsLzhrwO2uJi", "longtype": 896600886}
{"stringtype": "tolECUtK7bzEnC8", "longtype": 1569471442}
{"stringtype": "cdWTshpTH8OsIWx", "longtype": -1889352504}
{"stringtype": "dKQk4OQ03MrYSJi", "longtype": 1491184982}
```

编辑BSC 作业

创建BOS Source

进入BSC编辑作业界面，创建bos source table, sql代码如下

```
 CREATE table source_table_bos(
    stringtype STRING,
    longtype LONG
) with(
    type = 'BOS',
    path = 'bos://es-sink-test/test',
    encode = 'json'
);
```

其中path为上图中红框中指定的bos路径，并在路径前增加"bos://"前缀。

创建Es Sink Table

sql代码如下

```
 create table sink_table_es(
    stringtype String,
    longtype Long
)with(
    type = 'ES',
    es.net.http.auth.user = 'superuser',
    es.net.http.auth.pass = 'bbs_2016',
    es.resource = 'bsc_test_2/doc_type',
    es.clusterId = '296245916518715392',
    es.region = 'bd',
    es.port = '8200',
    es.version = '6.5.3'
);
```

其中：

- es.resource对应es的索引与类型，es会在bsc写入数据时自动创建指定索引
- es.clusterId对应es的集群ID
- es.region 表示 Es服务所在的地区的代码，可以参考 Es服务区域代码 中查询区域与代码的对应关系。

编写导入语句

sql语句如下：

```
 insert into
 sink_table_es(stringtype, longtype) outputmode append
select
    stringtype,
    longtype
from
    source_table_bos;
```

保存作业并发布运行作业

查看Es中的数据

```
$ curl -u superuser:[redacted] -XGET '[redacted]:9200/bsc_test_2/doc_type/_search?pretty'
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 47,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "bsc_test_2",
        "_type" : "doc_type",
        "_id" : "Z9QrXm0BOYiXc3ZprchG",
        "_score" : 1.0,
        "_source" : {
          "stringtype" : "fyCYBmBfDQOgzuk",
          "longtype" : -1345471890
        }
      },
      {
        "_index" : "bsc_test_2",
        "_type" : "doc_type",
        "_id" : "adQrXm0BOYiXc3ZprchG",
        "_score" : 1.0,
        "_source" : {
          "stringtype" : "wr74uHSIXKfx2D8",
          "longtype" : 2008439108
        }
      }
```

使用BSC将Kafka中的数据导入Es

介绍

本文主要介绍通过 BSC【百度流式计算服务】将数据从Kafka中导入到Es中。

创建集群

在将数据导入Es之前需要再百度云上创建Es集群，假定创建的集群信息如下：

这里主要记录以下信息：

- 集群ID： 296245916518715392
- 创建集群时的密码： bbs_2016

创建Kafka Topic

登录百度云管理控制台，进入kafka产品界面，创建topic，并向topic中灌入数据。示例中我们灌入的数据如下，包括两个json字

段，样例数据如下：


```
{"stringtype": "WXZUY0BiLWgNf19", "longtype": -1283176655}
{"stringtype": "t4EEGqq26i23pVI", "longtype": -934626880}
{"stringtype": "zZpGw2ro1IlLlVr", "longtype": 687526331}
{"stringtype": "RlA4WUR5hrO6v7q", "longtype": 906561815}
{"stringtype": "6a4rlSaepHhV6Qa", "longtype": 256533276}
{"stringtype": "JYjmbWIt0owNQR2", "longtype": 771880155}
{"stringtype": "hcZmqZByRkuHCQX", "longtype": 1615509054}
{"stringtype": "vkiI21KlROgjwoe", "longtype": 68141743}
{"stringtype": "8lexGCpgIMZmIV7", "longtype": 1715659598}
{"stringtype": "lnS21exFf7lhIZU", "longtype": 574818568}
{"stringtype": "OiCFGmyzw3vy2Ao", "longtype": 730183684}
{"stringtype": "SXjfYICUgG3cM94", "longtype": -1485075763}
{"stringtype": "s7UZA0DWUR8DUvO", "longtype": 201393814}
{"stringtype": "YH2tyZUN60YWz2B", "longtype": -725239146}
{"stringtype": "6Uh7yKqWqUHI7O2", "longtype": 2129044627}
{"stringtype": "fsTIV4bXKvb9gfO", "longtype": 37388436}
{"stringtype": "Pk3psuFsiwC2IGT", "longtype": 574184951}
{"stringtype": "aFJ3Qg3vaMXNWDd", "longtype": 834740933}
{"stringtype": "JsXuFdDxXjkaIyo", "longtype": -817529473}
{"stringtype": "w8ryKdBcZofnu4t", "longtype": 684437170}
{"stringtype": "Zew5mpfcT17WMg9", "longtype": -606060825}
{"stringtype": "Lo6cU3xRXQWGAjH", "longtype": 536442032}
{"stringtype": "D3ONsqvX7c7sl57", "longtype": 153497496}
{"stringtype": "sNYE45BX7h17WBl", "longtype": -805960702}
{"stringtype": "0ZjAYsigcshzd1c", "longtype": -395612212}
{"stringtype": "POmYOWtjRSbpxYT", "longtype": 1758488620}
{"stringtype": "LUsE6jQhXWzrxMU", "longtype": 724586883}
{"stringtype": "0akL57TNcY8Ood8", "longtype": 1813812715}
{"stringtype": "6kJWuR1AWljjFSU", "longtype": -2051429265}
{"stringtype": "VuVsm5tG6WC5pAN", "longtype": 122565252}
{"stringtype": "Mpxp1WAiLC8Rx94", "longtype": 477669386}
{"stringtype": "KnoPYS7kPSyhac4", "longtype": 1934915464}
{"stringtype": "6aVbj7x9muVESam", "longtype": -1771994861}
{"stringtype": "R5Zn3WXmwMEEM0S", "longtype": 91153259}
{"stringtype": "DCnBjDcxgMNTXHS", "longtype": -2133673768}
{"stringtype": "GioEDcI9j3jLJZx", "longtype": -1078025100}
{"stringtype": "I4e0DGhvuS7kXpT", "longtype": 1844038025}
{"stringtype": "FhgejhialwhL0fo", "longtype": -1395980195}
{"stringtype": "1oPbsLzhrwO2uJi", "longtype": 896600886}
{"stringtype": "tolECUtK7bzEnC8", "longtype": 1569471442}
{"stringtype": "cdWTshpTH8OsTWx", "longtype": -1889352504}
```


示例中创建topic：a15fdd9dd5154845b32f7c74ae155ae3__demo_test 并且确保该topic下有对应的证书，将证书下载到本地。

编辑BSC 作业

创建Kafka Source

进入BSC编辑作业界面，创建kafka source table, sql代码如下

```
 CREATE table source_table_kafka(
stringtype STRING,
longtype LONG
) with(
    type = 'BKAFKA',
    topic = 'a15fdd9dd5154845b32f7c74ae155ae3__demo_test',
    kafka.bootstrap.servers = 'kafka.bj.baidubce.com:9091',
    sslFilePath = 'kafka_key.zip',
    encode = 'json'
);
```

其中sslFilePath = 'kafka-key.zip'，为上一步下载到本地的kafka证书。

上传Kafka证书

点击高级设置，上传kafka证书

上传之后如下图

创建Es Sink Table

sql代码如下

```
 create table sink_table_es(
   stringtype String,
   longtype Long
)with(
   type = 'ES',
   es.net.http.auth.user = 'superuser',
   es.net.http.auth.pass = 'bbs_2016',
   es.resource = 'bsc_test/doc_type',
   es.clusterId = '296245916518715392',
   es.region = 'bd',
   es.port = '8200',
   es.version = '6.5.3'
);
```

其中：

- es.resource对应es的索引与类型，es会在bsc写入数据时自动创建指定索引
- es.clusterId对应es的集群ID
- es.region 表示 Es服务所在的地区的代码，可以参考 Es服务区域代码 中查询区域与代码的对应关系。

编写导入语句

sql语句如下：

```
 insert into
   sink_table_es(stringtype, longtype) outputmode append
select
   stringtype,
   longtype
from
   source_table_kafka;
```

保存作业并发布运行作业

查看Es中的数据

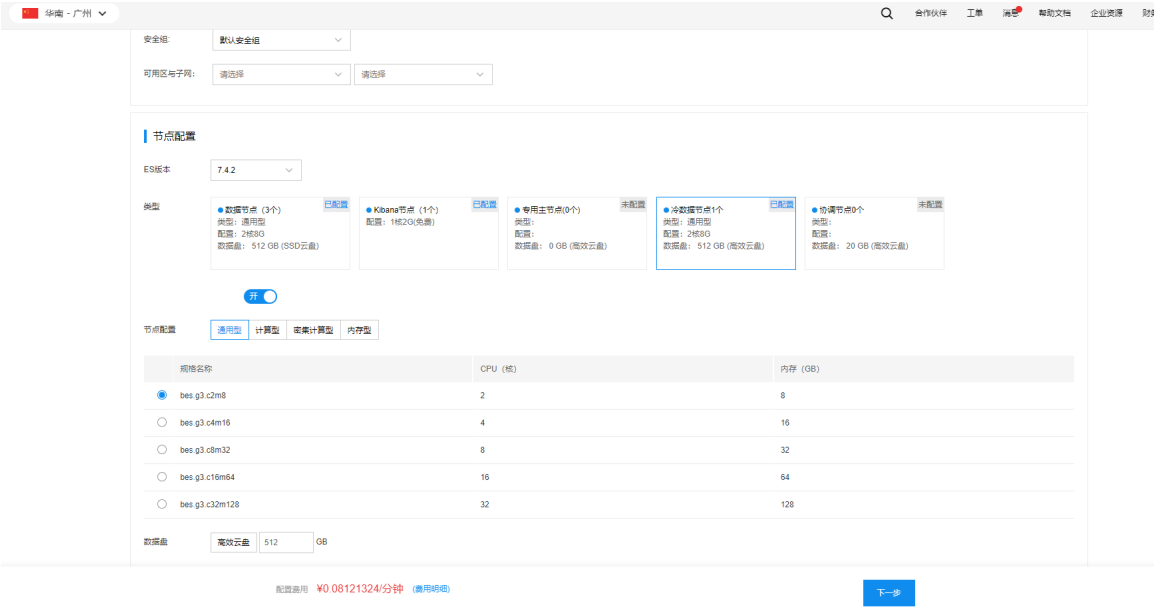
```
[redacted]~]$ curl -u superuser:[redacted] -XGET [redacted]8200/bsc_test/doc_type/_search?pretty'
{
  "took" : 7,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 2946,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "bsc_test",
        "_type" : "doc_type",
        "_id" : "3tQOXm0BOYiXc32ppryK",
        "_score" : 1.0,
        "_source" : {
          "stringtype" : "BHFyWqFOUj9KutL",
          "longtype" : 518597404
        }
      },
      {
        "_index" : "bsc_test",
        "_type" : "doc_type",
        "_id" : "5NQOXm0BOYiXc32ppryK",
        "_score" : 1.0,
        "_source" : {
          "stringtype" : "uPbnh9pmS1hFlgF",
          "longtype" : 1772078079
        }
      },
```

基于节点磁盘介质的冷热数据分离

创建带冷数据节点集群

冷数据节点集群创建

登录百度智能云Elasticsearch 控制台，选择【创建集群】



开启冷数据节点，并选择对应的计算资源及磁盘配置进行创建

冷热数据节点对比

节点类型	磁盘类型	参数配置	是否绑定百度云blb接受用户请求
热数据节点	建议使用ssd云磁盘	node.attr.data_tier: hot	是
冷数据节点	高性能云磁盘	node.attr.data_tier: warm	否

创建成功后可以通过以下命令查看节点属性：

```
GET /_cat/nodeattrs?v&s=value
```

例如创建单个热节点及单个冷节点则节点配置如下：

```
node                                 host        ip          attr      value
524849072495005696-172.16.0.14 172.16.0.14 172.16.0.14 data_tier hot
524849072750858240-172.16.0.12 172.16.0.12 172.16.0.12 data_tier warm
```

冷热数据分离的集群建议使用热数据节点存储近期数据(如近7天数据)来保证热数据的查询效率，使用冷数据节点来存储历史数据（如7天之前的数据）来降低存储成本。

> **注意**：创建集群时，只有启用了冷数据节点，数据节点才会变成热数据节点。

索引操作冷热变更

冷热数据属性说明

`index.routing.allocation.require.data_tier`属性用于设置索引为热数据(hot)还是冷数据(warm)。

创建热索引:

```
PUT hot_warm_data
{
  "settings" : {
    "index.routing.allocation.require.data_tier": "hot",
    "number_of_shards": 5
  }
}
```

```
index         shard prirep state   docs store ip          node
hot_warm_data 2     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
hot_warm_data 4     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
hot_warm_data 1     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
hot_warm_data 3     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
hot_warm_data 0     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
```

设置索引数据为冷数据

```
PUT hot_warm_data/_settings
{
  "index.routing.allocation.require.data_tier": "warm"
}
```

数据开始由热数据节点向冷数据节点迁移，迁移进度可以通过以下命令查看：

```
GET /_cat/recovery?v&active_only=true
```

```
index         shard time type stage source_host source_node                     target_host target_node                     repository snapshot files files_recovered files_percent files_total bytes bytes_recovered bytes_percent bytes_total translog_ops
translog_ops_recovered translog_ops_percent
hot_warm_data 1     1.2s peer index 172.16.0.14 524849072495005696-172.16.0.14 172.16.0.12 524849072750858240-172.16.0.12 n/a        n/a      1     0             0.0%          1          230   0             0.0%          230        10          0
0.0%
```

迁移完成后shard分布情况，数据已被迁移到冷数据节点

```
index          shard prirep state   docs store ip           node
hot_warm_data 4     p      STARTED    0  230b 172.16.0.12 524849072750858240-172.16.0.12
hot_warm_data 2     p      STARTED    0  230b 172.16.0.12 524849072750858240-172.16.0.12
hot_warm_data 1     p      STARTED    0  230b 172.16.0.12 524849072750858240-172.16.0.12
hot_warm_data 3     p      STARTED    0  230b 172.16.0.12 524849072750858240-172.16.0.12
hot_warm_data 0     p      STARTED    0  230b 172.16.0.12 524849072750858240-172.16.0.12
```

设置索引数据为热数据

```
PUT hot_warm_data/_settings
{
  "index.routing.allocation.require.data_tier": "hot"
}
```

数据开始由冷数据节点向热数据节点迁移，迁移进度可以通过以下命令查看：

```
GET /_cat/recovery?v&active_only=true
```

```
index          shard time type stage source_host source_node                    target_host target_node                    repository snapshot files files_recovered files_percent files_total bytes bytes_recovered bytes_percent bytes_total translog_ops translog_ops_recovered
translog_ops_percent
hot_warm_data 1     1.4s peer index 172.16.0.12 524849072750858240-172.16.0.12 172.16.0.14 524849072495005696-172.16.0.14 n/a       n/a     1     0            0.0%        1         261   0            0.0%        261        0          0                    100
.0%
```

迁移完成后shard分布情况，数据已被迁移到热数据节点

```
index          shard prirep state   docs store ip           node
hot_warm_data 2     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
hot_warm_data 4     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
hot_warm_data 1     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
hot_warm_data 3     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
hot_warm_data 0     p      STARTED    0  230b 172.16.0.14 524849072495005696-172.16.0.14
```

基于CCR实现多集群跨地域高可用

一、CCR功能介绍

功能概述：

CCR（Cross Cluster Replication，跨集群复制）是BES（百度Elasticsearch）高效的数据同步解决方案，旨在提高数据的可用性、降低访问延迟，并实现数据的灵活迁移。

CCR功能以数据订阅的方式工作，一个集群的数据可以被多个集群订阅，即可以被复制到多个集群上。 在CCR中有两种角色：Leader和Follower，Leader表示源数据方，Follower表示数据订阅方，其中Leader生产数据，Follower订阅Leader后同步数据，以只读的方式提供数据服务。

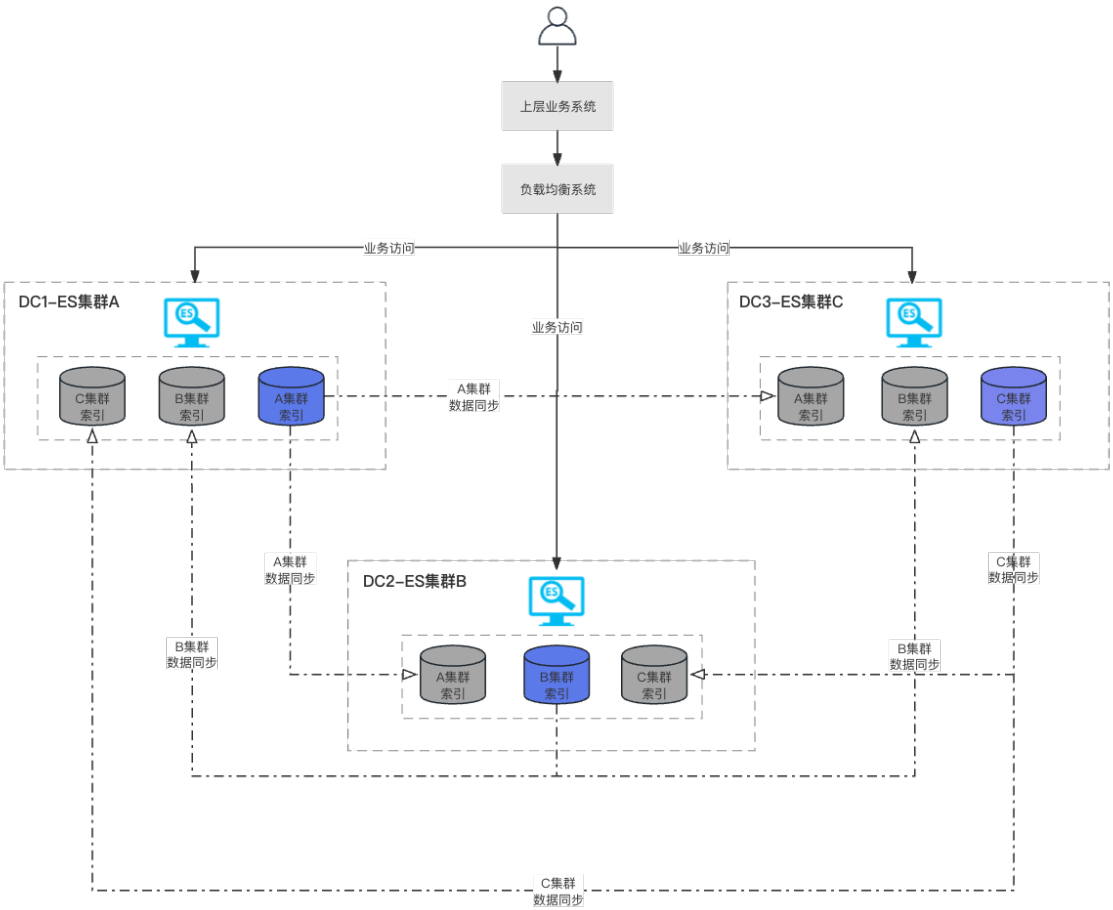
工作机制：

在BES中，索引数据被分割成多个分片（Shard），每个分片负责物理存储和管理索引的一部分数据。CCR就是在索引分片维度上实现的，其采用轮询的方式从Leader索引的主分片拉取数据到Follower索引主分片，单向进行数据同步，并保证一致性与及时性，其中分片中数据同步方式如下所示：

序号	数据类型	说明	同步方式	特点
1	分片存量数据	构建同步关系前Leader索引已经存在的数据	索引文件拉取	只涉及IO操作，同步速率快
2	分片增量数据	构建同步关系后Leader索引生成的数据	索引数据记录拉取	基于索引数据记录同步，延迟低

二、多地域高可用场景实践

CCR支持索引数据的单向同步，也支持多订阅模式，可以用于构建多地域的高可用场景。以三地域（包含BES集群：A、B、C）的高可用BES集群为例，三个地域的BES集群，每个集群均订阅另外两个集群中的数据，保证每个地域的BES集群都具备完整数据并都提供读写服务，如下图所示：



当上述任何一个地域故障时，负载均衡系统会切断故障机房流量，上层业务系统仍可以正常使用ES集群对故障无感。

2.1 集群环境准备

在三个地域准备三个7.10.2的BES集群，如下所示：

序号	集群名称	集群地域	版本
1	集群A	北京	7.10.2
2	集群B	保定	7.10.2
3	集群C	广州	7.10.2

2.2 基于CCR搭建高可用环境

2.2.1 网络设置

进行高可用环境的配置，先要进行各个集群所属VPC间的网络配置，保证各个区域的BES集群网络互通。

2.2.2 同步索引配置

2.2.2.1 插件安装 在控制台的集群详情页面，【插件配置】-> 【系统插件】中找到CCR插件，点击安装即可完成插件安装（注意：安装后需要重启BES集群，插件才能生效）。 **2.2.2.2 主索引配置** 主索引不需要进行特殊的设置来支持CCR同步功能，但使用别名机制可以方便的进行索引切换，避免业务程序因操作索引的名称变化而进行重新部署。 以名称为leader_a_index的索引为例，加入读写别名的操作指令如下所示：

```
POST /_aliases
{
  "actions": [
    {
      "add": {
        "index": "leader_a_index",
        "alias": "leader_a_index_read_alias"
      }
    },
    {
      "add": {
        "index": "leader_a_index",
        "alias": "leader_a_index_write_alias",
        "is_write_index":true
      }
    }
  ]
}
```

> 说明：上述指令可以在Kibana的Dev Tools执行，也可以通过SDK或直接调用Restful API执行，下文所有指令不再进行额外说明。

2.2.2.3 同步索引配置 在三个BES集群上，配置同步索引的操作步骤是相似的，下面以一个集群的配置操作举例说明如何进行操作配置。 在一个BES集群中，配置索引同步的步骤是：

1. 从集群设置连接的主集群信息
2. 从集群配置同步索引

具体的操作指令与说明详见下述内容。 **2.2.2.3.1 从集群设置连接的主集群信息** 该步骤是使用需要配置的第一个步骤，用于从集群（Follower集群）中配置需要连接的主集群（Leader集群）地址信息。一个从集群可以配置 多个主集群（Leader集群），以A集群配置B、C集群的同步索引为例，指令与说明详见下述内容。 操作指令说明:

```
PUT /_cluster/settings
{
  "persistent": {
    "cluster": {
      "remote": {
        "{主集群B-自定义名称}": {
          "seeds": [
            "{集群B节点ip:tcp端口}","{主集群B节点ip:tcp端口}","{主集群B节点ip:tcp端口}"
          ]
        },
        "{主集群C-自定义名称}": {
          "seeds": [
            "{主集群C节点ip:tcp端口}","{主集群C节点ip:tcp端口}","{主集群C节点ip:tcp端口}"
          ]
        }
      }
    }
  }
}
```

> 说明：上述节点ip为主集群节点的 vpc ip ，tcp端口为 9200

2.2.2.3.2 从集群配置同步索引 完成上述步骤操作后，既可以在从集群（Follower集群）进行同步索引的配置操作，开启数据同步过程。 操作指令说明:

```
PUT /_ccr/_replication/{从集群索引名}/_start
{
  "remote_cluster": "{主集群-自定义名称}",
  "remote_index": "{主集群索引名}"
}
```

使用示例：

```
PUT /_ccr/_replication/follower_index/_start
{
 "remote_cluster": "leader-cluster-a",
 "remote_index": "leader_a_index"
}
```

2.2.2.3.3 同步索引别名操作 同步索引没有索引别名操作的限制，当需要将查询从leader_a_index同步索引，切换为leader_a_index_new同步索引时，更换别名的操作指令如下所示：

```
POST /_aliases
{
  "actions": [
    {
      "remove": {
        "index": "leader_a_index",
        "alias": "leader_a_index_read_alias"
      }
    },
    {
      "add": {
        "index": "leader_a_index_new",
        "alias": "leader_a_index_read_alias"
      }
    }
  ]
}
```

三、方案优势

与传统的基于消息中间件多订阅模式的高可用模式相比，基于CCR插件的高可用方案具有如下优势：

方案	系统复杂度	数据实时性	运维复杂度
CCR方案	低	高	低
消息中间件方案	高	普通	高

3.1 系统复杂度

基于消息中间件多订阅模式的高可用方案需要Kafka（或者其他消息队列）以及多个消费者程序，依赖的外服服务多；CCR方案以ES插件的方式提供服务，无第三方服务的依赖，因此基于CCR的方案复杂度更低。

3.2 数据实时性

对于存量数据的同步：

基于消息中间件的方案需要使用数据索引的方式进行同步，操作复杂度高（需要reindex或者将全部数据读取到消息队列，使用索引程序读取构 建）耗时久，而CCR方案使用的是索引文件读取方式进行同步，速率更快。

对于增量数据：

基于消息中间件的方案同步数据时，还需要经过消息中间件传导数据，而CCR方案是分片到分片的直接同步，速率也更有优势。

3.3 运维复杂度

基于消息中间件的方案的运维成本更高，因为增加了第三方的服务依赖，需要更多运维工作来保障消息中间件及生产消费程序的可用性；在进行故障恢复时，需要的操作也更多。

综上，基于CCR的BES高可用方案，系统复杂度低、数据同步实时性高、系统维护成本也更低，是更好的方案选择。

常见问题

常见问题总览

Elasticsearch系统常见问题

- 如何查看Es安装了哪些插件
- 线程池队列满导致错误
- Too Many Open Files的错误
- Es 中一个分片一般设置多大
- 当集群为red或者yellow的时候怎么办
- 如何cancel掉慢查询
- PageCache 在查询中的作用很大
- 禁用权限验证
- 支持的Client的类型
- Es是否支持Spark和Hadoop来写入或者读取数据
- JVM FULL GC 的几种情形
- 如何提升导入性能
- 集群配置问题
- 集群重启问题
- 禁用_field_names
- 导入数据发现越来越慢的几种情形
- 控制index在节点上的数量
- 控制索引的分片数量和副本数量
- 当集群处于恢复状态的时候，恢复速度可能会比较慢

Spark访问Es常见问题

- 说明
- 版本号检测异常

- 数据节点发现
- 用户权限问题
- 配置Bulk导入有错误的处理方式
- 如何拿到除_source外的其他文档元数据
- 导入的时候指定id、version的方法
- spark默认写ES的时候refresh的会在每次bulk结束的时候调用
- 控制每次Bulk写入的量
- 读相关设置
- 控制需要写入的文档字段
- 执行upsert操作

Elasticsearch系统常见问题

本文档主要目的：

1. 用来积累用户常见的问题。
2. 文档中的操作可通过 Kibana 或在chrome浏览器中安装sense插件完成。用户也可以通过curl的方式自行实现。

如何查看Es安装了哪些插件

可以使用下面这个API，会列出每个节点安装的插件列表。

```
GET /_cat/plugins
```

线程池队列满导致错误

在这种场景下ES抛出的异常是

> ```
> rejected execution of org.elasticsearch.transport.TransportService$4@c8998f4
> on EsThreadPoolExecutor[bulk, queue capacity = 50,
> org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@553aee29
> [Running, pool size = 4, active threads = 4, queued tasks = 50, completed tasks = 0]]
> ```

ES内部有很多线程池，比如index，search，bulk是我们能够看到的3个典型的线程池，如果系统的压力特别大，后台线程处理不过来的时候，用户发起的任务会在线程池的队列里堆积，如果达到队列的上限就会抛出对应的异常，遇到这种错误需要做以下两步：

- 检查系统的CPU和IO的利用情况，如果系统的IO和CPU的利用率比较高，这说明系统遇到资源瓶颈了，已经不能通过优化系统的参数来避免这种错误发生了，云上的用户可以在ES的控制台查看ES的CPU利用率，也可以通过ES自带的命令来查看，通过下面这个命令可以看到ES各个节点的CPU利用率以及负载： GET /_cat/nodes?v
- 如果资源没有问题，那么检查当先线程池的配置，比如上面这个错误就需要检查bulk的线程池的配置，在sense里执行以下命令： GET /_cluster/settings

  结果如下

```
    "thread_pool": {
      "bulk": {
        "type": "fixed",
        "size": "4",
        "queue_size": "50"
      }
    }
```

这个结果表示处理bulk任务的线程池有4个执行线程，队列数为50. 根据我们的经验看，这个值还是比较小的，所以可以直接用以下操作处理：

ES 5.5.0 版本：

```
 PUT /_cluster/settings
{
 "persistent": {
  "thread_pool.bulk.size": 32,
  "thread_pool.bulk.queue_size": 300
 }
}
```

ES 6.5.3+ 集群：

```
 PUT /_cluster/settings
{
 "persistent": {
  "thread_pool.write.size": 32,
  "thread_pool.write.queue_size": 300
 }
}
```

Too Many Open Files的错误

在es的日志中如果出现这个错误，一般都是打开的文件太多了，ES 建议文件句柄的限制至少为65536个，用户可以通过修改/etc/security/limits.conf来修改，或者用ulimit这个命令来修改。 es里每个shard都是一个单独的lucene index writer，每个shard由多个segment组成，每个segment有多个文件，所以打开的文件的数目= shard数目 *segment数目* 每个segment包含的文件数量，所以我们建议一个物理机节点上shard的数目在1000个左右，不建议有太多的shard。另外lucene使用compound file格式也能有效的减少每个segment里的文件的数量。

Es 中一个分片一般设置多大

ES 的每个分片（shard）都是lucene的一个index，而lucene的一个index只能存储20亿个文档，所以一个分片也只能最多存储20亿个文档。 另外，我们也建议一个分片的大小在10G-50G之间，太大的话查询时会比较慢，另外在做副本修复的时，耗时比较多；分片太小的话，会导致一个索引的分片数目很多，查询时带来的 fanout 太大。

当集群为red或者yellow的时候怎么办

集群为RED表示集群中有primary shard没有分配，yellow表示有replica没有分配，我们建议你用下面这个API来看shard为什么没有被分配到某个节点上。 GET _cluster/allocation/explain

根据我们的使用经验，有以下几种情况导致shard没有被分配：

- 没有节点上有存储空间能够放下这个shard。
- 如果shard是replica，那么可能是primary shard未分配或者处于initializing状态。

分片长时间处于未分配状态 ES内部会对一个unassigned 分片尝试5次进行分配,失败后不再尝试进行分配，这时候需要调用进行手动控制集群处理 unassigned 分片：

```
POST /_cluster/reroute?retry_failed=true
```

如何cancel掉慢查询

用户发送一个查询可能导致一个集群非常慢，CPU利用率非常高，所以用户有的时候想把占用资源非常多的查询cancel 掉。在ES 5.0之后es提供了cancel查询的命令。 es内部把所有的执行任务都封装成了task，可以通过task api来查看一个节点在执行的task任务列表，也可以使用task api来取消task。比如我们要查询所有在执行search类型的task，可以使用如下API： GET /_tasks?actions=*search

取消所有在执行的search任务：

```
POST _tasks/_cancel?actions=*search
```

更多的使用方式可以参考官方网站的介绍。

PageCache 在查询中的作用很大

我们建议如果条件可以的话应该给ES留尽量多的pagecache，这能极大的优化我们的查询速度，如果pagecache不足够多，那么ES每次查询【fetch 文档，拿posting list】都会读取磁盘，此时系统就会变慢。 用户可以使用iostat 来查看一下系统的IO 信息，也可在 GET _nodes/state 返回的信息里搜索 "io_stats"查看。如果iops比较高的话，说明系统的io比较高了，可能就是pagecache小的原因。

禁用权限验证

有的时候业务系统原有的ES服务没有权限验证，但是云上的ES服务是有权限验证的，当业务系统迁移的时候不希望改代码，那么可以先把权限验证关掉，这样就能平滑迁移了。操作的方式是：

```
 PUT /_cluster/settings
{
   "persistent": {
      "simpleauth.enable":false
   }
}
```

支持的Client的类型

目前我们云上的产品只支持基于http 的restful api，不支持基于tcp的transport client这种api。 这个设置主要原因是transport client跟集群运行的版本深度绑定，当集群升级的时候需要前端业务也跟着升级才可以。

Es是否支持Spark和Hadoop来写入或者读取数据

支持，需要到es官方网站下载es-hadoop包放到spark或者hadoop中就可以用spark或者hadoop读写es了。

JVM FULL GC 的几种情形

Scroll 导致FullGC

一些用户使用scroll做分页查询或者用scroll导出数据的时候，经常把scroll的超时时间设置的比较长，比如设置为1天，在这种情况下es后端会为这个scroll一直保存对应的search context，每个search context都对应了lucene的searcher，此时searcher一直不释放导致lucene merge完的文件也不删除，一些leafreader， fst等都长期在JVM 里导致最终随着search context越来越多导致了FullGC。用户可以使用以下2个API 来查看和清除这些Context。

```
 GET /_nodes/stats/indices/search
DELETE /_search/scroll/_all
```

查询导致FullGC

用户在查询时将结果集的from+size设置的太大，比如size=Integer.MAX_VALUE导致的，目前ES会根据设置的这个from+size开辟一个priority queue，当并发量大时，内存会分配不过来这么多非常大的queue，导致FULL GC，甚至OOM。

aggregation导致FullGC

用户在执行类似terms agg时，如果不同的值非常多，最终会导致产生很多bucket，比如几千万个bucket，这些bucket也会在内存里，最终导致fullgc。

如何提升导入性能

减少副本数，增加refresh间隔

```
 PUT /index_name/_settings
{
  "index.number_of_replicas": 0,
  "index.refresh_interval": "10s"
}
```

ES 的多副本机制在写入时会向多个副本都发送原始的json文档，然后在多个副本上分别进行分词，建立索引等操作。由于导入是CPU密集型操作，所以把replica数目改成0，可以减少CPU使用率，当导入完毕后，把replica数目改回，这样就是直接拷贝物理文件了，速度会比较快。

refresh interal是用来控制多久把内存里的数据刷出segment的，es会对刷出的segment进行merge，如果merge不过来es会阻止写入。所以把refresh interval调大，也可以把刷出的segment变大，降低merge的频率，提升导入性能。 **增大index的导入速度限制**

```
 PUT /_cluster/settings
{
    "persistent" : {
        "indices.store.throttle.max_bytes_per_sec" : "200mb"
    }
}
```

ES 在写入数据的时候会有速度的限制，防止占用过多的磁盘IO，如果集群的导入比较大而查询比较少，那么可以把这个速度限制调大。

集群配置问题

- 需要使用 oracle JDK 1.8 以上版本。
- 设置最大文件数：

```
 修改 /etc/security/limits.conf ：

*     soft     nofile     65536

*     hard     nofile     65536
```

- 增加 mmap counts ： 修改 /etc/sysctl.conf ：

```
 vm.max_map_count=262144

然后执行： sysctl -p
```

集群重启问题

在一些情况下（比如修改配置文件），我们希望重启集群，重启集群可以是一台台的重启，也可以是整个集群重启，重启Es的

时候，可能会引起数据的重分布，下面就这两种情况分别介绍如何重启服务。

整个集群重启

- 把整个集群设置为只读状态

```
 PUT /_cluster/settings
{
    "persistent": {
        "cluster.blocks.read_only":true
    }
}
```

- 把节点内存的数据全部flush到硬盘上

```
 POST /_flush/synced
```

- 把所有的es节点重启
- 当集群green之后，把集群修改为可写入状态

```
 PUT /_cluster/settings
{
    "persistent": {
        "cluster.blocks.read_only":false
    }
}
```

一台台重启

这种方式重启服务不会中断，适用于线上服务。

- 禁止分片分配，这样我们关闭一台Es服务的时候，分片不会重分布。 `PUT /_cluster/settings`
  `{`
  `"transient" : {`
  `"cluster.routing.allocation.enable" : "none"`
  `}`
  `}`
- 关闭单个节点，修改配置或者替换jar包，启动节点
- 开启分片重分布

```
 PUT /_cluster/settings
{
    "transient" : {
        "cluster.routing.allocation.enable" : "all"
    }
}
 ```
```

- 等待集群green后，重复执行1-3 步，直到所有的节点都修改完配置为止。

禁用_field_names

`_field_names`字段是Elasticsearch的内部的一个元数据字段。该字段会索引文档内的每个字段的名字(除了字段值为null的字段名字);这个字段存在的意义主要是执行Elasticsarech `exists` query，Elasticsearch对该字段只做了索引处理，没有存储该字段，6.3版本以后该字段只会索引没有禁掉doc_value和norms的字段，建议业务不使用`exists` 查询的情况下disable该字段，能
```

够少量的减少倒排表的占用的存储空间，可能会适当增强pagecache的利用

```
PUT index
{
  "mappings": {
    "_doc": {
      "_field_names": {
        "enabled": false
      }
    }
  }
}
```

导入数据发现越来越慢的几种情形

导入数据中包含update

Es的update实际上是先读取数据然后更改后，再写入的，当写入的数据越来越多的时候，读取数据就会比较慢，写入也就逐渐变慢了。

控制index在节点上的数量

默认情况下，ES集群会尽可能将所有节点上的index和shard的数量进行balance，但是一些特殊情况下，可能会造成某一个index的shard过多的集中在少量的节点上，这时候可以通过设置集群中每个节点存放index shard的个数：

```
PUT {index名字}/_settings
{
  "index.routing.allocation.total_shards_per_node": 10
}
```

控制索引的分片数量和副本数量

不修改参数的情况下，一个index 一共有5个分片，2个副本（包括主分片）,可以通过修改index的参数来控制：

```
PUT /{index名字}
{
   "settings": {
       "number_of_shards": 20,
       "number_of_replicas": 2
   }
}
```

```
number_of_shards: 分片个数，创建完index后不可修改，需在创建的时候指定
number_of_replicas： 副本个数，不包括主分片
```

当集群处于恢复状态的时候，恢复速度可能会比较慢

当前正在恢复的索引分片可以通过

```
GET /_recovery?active_only=true
```

查看，默认情况一个节点同时恢复的个数为4，包括2个作为source节点，2个作为target节点，当分片个数非常多的时候可能会恢复的很慢，恢复的时候默认是有限速的最大40mb,这时候可以通过设置集群参数：

```
curl -XPUT "host:port/_cluster/settings" -d'
{
    "transient": {
        "cluster.routing.allocation.node_concurrent_recoveries": 8,
        "indices.recovery.max_bytes_per_sec": "120mb"
    }
}'
```

```
indices.recovery.max_bytes_per_sec： 节点恢复的最大带宽，这个设置应该小于当前网络带宽，避免影响其他网络服务
cluster.routing.allocation.node_concurrent_recoveries： 节点作为source node或target node同时恢复的最大个数
```

磁盘满了之后如何恢复

当Es的DataNode的磁盘使用率达到一定的阈值（95%）之后，Es会阻止继续写入，Es会在所有的Index上加一个block，当用户继续写入的时候会收到以下错误

```
cluster_block_exception [FORBIDDEN/12/index read-only / allow delete (api)];
```

此时用户必须释放磁盘空间才能解决问题，释放磁盘空间有2种办法：

- 删除不用的Index
- 降低Index的副本的数目，比如把replica数目从2降到1.

当释放完毕磁盘空间之后，Es并不会自动把block去掉，此时用户仍然无法写入数据，需要执行以下命令：

```
curl -XPUT "host:port/_all/_settings" -d '
{
    "index.blocks.read_only_allow_delete": null
}'
```

Spark访问Es常见问题

说明

本文档主要介绍了通过`elasticsearch-hadoop`中的`Spark`访问ES时常见配置项意义。本文中的`es-spark`是`elasticsearch-hadoop`中和`Spark`相关联的包，用户通过自己的`Spark`集群读写ES集群，`elasticsearch-hadoop`基本上兼容了目前ES所有的版本

版本号检测异常

es-spark 运行时通常会自动检测ES集群的版本号，获取的版本号主要是用来对不同集群的版本做API的兼容处理

一般情况下用户不用关注ES版本号，但是在云上有时候自动检测集群的版本号会发生一些莫名其妙的检测不到的错误，可以通过配置解决：

配置项：

```
es.internal.es.version："6.5.3"
```

在一些较新版本的`es-spark`包中同样需要配置：

```
es.internal.es.cluster.name："Your Cluter Name"
```

实现原理：

配置完以后，es-spark不会请求 / 目录，解析version，会直接使用用户配置的version：

```
 INTERNAL_ES_VERSION = "es.internal.es.version"
INTERNAL_ES_CLUSTER_NAME = "es.internal.es.cluster.name"

public static EsMajorVersion discoverEsVersion(Settings settings, Log log) {
    return discoverClusterInfo(settings, log).getMajorVersion();
}

// 不同版本的elasticsearch-hadoop可能会有差异
public static ClusterInfo discoverClusterInfo(Settings settings, Log log) {
        ClusterName remoteClusterName = null;
        EsMajorVersion remoteVersion = null;
        // 尝试从配置中获取集群名字
        String clusterName = settings.getProperty(InternalConfigurationOptions.INTERNAL_ES_CLUSTER_NAME);
        // 尝试从配置中获取集群UUID
        String clusterUUID = settings.getProperty(InternalConfigurationOptions.INTERNAL_ES_CLUSTER_UUID);
        // 尝试从配置中获取ES version
        String version = settings.getProperty(InternalConfigurationOptions.INTERNAL_ES_VERSION);
        // 如果集群名字和版本号没有从配置文件中拿到，则发起网络请求(请求根目录)
        if (StringUtils.hasText(clusterName) && StringUtils.hasText(version)) {
            if (log.isDebugEnabled()) {
                log.debug(String.format("Elasticsearch cluster [NAME:%s][UUID:%s][VERSION:%s] already present in
configuration; skipping discovery",
                        clusterName, clusterUUID, version));
            }
            remoteClusterName = new ClusterName(clusterName, clusterUUID);
            remoteVersion = EsMajorVersion.parse(version);
            return new ClusterInfo(remoteClusterName, remoteVersion);
        }
      ....
}
```

如果启用集群名、版本号自动检测功能，需要保证分配给es-spark的用户有访问根目录`/`的`GET`权限

```
 GET /
```

数据节点发现

配置项：

```
 es.nodes.wan.only： false  默认为false
es.nodes.discovery: true   默认为true
```

百度云ES集群前面有一个BLB负载均衡，配置`es.nodes`的时候写这个BLB的地址即可，需要保证`es-spark`可以访问BLB的地址

- `es.nodes.wan.only: false`，`es.nodes.discovery: true`: Spark会通过访问`es.nodes`中指定的`host`(可以为多个) 得到ES集群所有开启`HTTP`服务节点的`ip`和`port`，后续对数据的访问会直接访问分片数据所在的节点上（需要保证ES集群所有节点都能够被`Spark`集群访问到）
- `es.nodes.wan.only: true`，`es.nodes.discovery: false`或不设置：Spark发送给ES的所有请求都需要通过这个节点进行转发,效率相对比较低

具体代码逻辑：

```
 ES_NODES_DISCOVERY = "es.nodes.discovery"
ES_NODES_WAN_ONLY = "es.nodes.wan.only"
ES_NODES_WAN_ONLY_DEFAULT = "false"

InitializationUtils#discoverNodesIfNeeded
    public static List<NodeInfo> discoverNodesIfNeeded(Settings settings, Log log) {
        if (settings.getNodesDiscovery()) { // 需要读取配置项
            RestClient bootstrap = new RestClient(settings);

            try {
                List<NodeInfo> discoveredNodes = bootstrap.getHttpNodes(false);
                if (log.isDebugEnabled()) {
                    log.debug(String.format("Nodes discovery enabled - found %s", discoveredNodes));
                }

                SettingsUtils.addDiscoveredNodes(settings, discoveredNodes);
                return discoveredNodes;
            } finally {
                bootstrap.close();
            }
        }

        return null;
    }

public boolean getNodesDiscovery() {
        // by default, if not set, return a value compatible with the WAN setting
        // otherwise return the user value.
        // this helps validate the configuration
        return Booleans.parseBoolean(getProperty(ES_NODES_DISCOVERY), !getNodesWANOnly()); //默认值
是!getNodesWANOnly()
    }

public boolean getNodesWANOnly() {
        return Booleans.parseBoolean(getProperty(ES_NODES_WAN_ONLY, ES_NODES_WAN_ONLY_DEFAULT));
    }
```

用户权限问题

如果启动节点发现功能，需要保证es-spark使用的用户有访问

```
 GET /_nodes/http
GET /{index}/_search_shards
```

的权限，否则会导致整个Job失败

配置Bulk导入有错误的处理方式

Spark 写ES的时候发生错误且经过几次尝试后会中断job，默认的情况下会直接中断当前Job，导致整个任务失败，如果只是想把导入失败的文档打印到日志中，可以通过如下配置解决：

配置错误发生错误的处理机制

```
 es.write.rest.error.handlers = log
es.write.rest.error.handler.log.logger.name： es_error_handler
```

设置后当出现bulk写错误的时候，Job不会中断，会以log的形式输出到日志中，日志前缀为`es_error_handler`

如何拿到除_source外的其他文档元数据

正常情况下，我们通过调用ES的_search API返回的每条数据如下：

```
        {
         "_index": "d_index",
         "_type": "doc",
         "_id": "51rrB2sBaX4YjyPY-2EG",
         "_score": 1,
         "_source": {
            "A": "field A value",
            "B": "field B value"
         }
       }
```

但是用Spark去读ES的时候，默认是不读除`_source`字段意外的其他字段，如`_id` `_version`，在一些场景下，业务可能需要拿到`_id`，可以通过如下配置：

```
  es.read.metadata:true  //这个配置默认是false
es.read.metadata.field: "_id" // 配置需要读取的元数据字段
es.read.metadata.version: 默认false 读取es的版本号
```

文档的元数据字段信息会放在一个`_metadata`的字段里面

导入的时候指定id、version的方法

做数据迁移的时候，比如从一个低版本的ES集群迁移到高版的ES集群，我们可以用es-spark边度边写，如果需要指定`_id`，`_rouring`，`_version` 这些信息，可以设置：

```
  es.mapping.id：”_meta._id” 指定json中id的路径
es.mapping.routing：”_meta._rouring” 指定json中id的路径
es.mapping.version：”_meta._version” 指定json中id的路径
```

`_meta.xxx`是所需的字段在Json文档中的路径

spark默认写ES的时候refresh的会在每次bulk结束的时候调用

我们建议设置为false，有ES内部`index`的 `refresh_interval`来控制`refresh`，否则ES集群会有大量的线程在`refesh`会带来很大的CPU和磁盘压力

```
  es.batch.write.refresh: false 默认是true
```

控制每次Bulk写入的量

```
  es.batch.size.bytes:1mb 每次bulk写入文档的大小，默认1mb
es.batch.size.entries:1000 每次bulk写入的文档数，默认是1000
```

用户可根据ES集群套餐合理设置

读相关设置

```
  es.scroll.size: 50, 默认是50 这个值相对来说比较小，可以适当增大至1000-10000
es.input.use.sliced.partitions：  true  为了提高并发，es会进行scroll-slice进行切分
es.input.max.docs.per.partition： 100000 根据这个值进行切分slice
```

es-spark 读取 ES集群数据的时候，会按照每个分片的总数进行切分做`scroll-slice`处理：

```
int numPartitions = (int) Math.max(1, numDocs / maxDocsPerPartition);
```

numDocs为单个分片的文档总数，如果文档有5千万，这时候会切分成500个sclice，会对后端的线上ES集群造成巨大的CPU压力，所以一般建议关闭`scroll-slice`,以避免影响在线业务

建议的参数：

```
es.scroll.size: 2000 //尽量根据文档的大小来选择
es.input.use.sliced.partitions： false
```

控制需要写入的文档字段

有时候业务导入数据的时候，希望有一些字段不被写如ES，可以设置：

```
es.mapping.exclude：默认是none
// 多个字段用逗号进行分割，直接 . * 表达
es.mapping.exclude = *.description
es.mapping.include = u*, foo.*
```

执行upsert操作

示例：

```
String update_params = "parmas:update_time";
String update_script = "ctx._source.update_time = params.update_time";
// 设置sparkConfig
SparkConf sparkConf = new SparkConf()
        .setAppName("YourAppName”)
        .set("es.net.http.auth.user", user)
        .set("es.net.http.auth.pass", pwd)
        .set("es.nodes", nodes)
        .set("es.port", port)
        .set("es.batch.size.entries", "50")
        .set("es.http.timeout","5m")
        .set("es.read.metadata.field", "_id")
        .set("es.write.operation","upsert")
        .set("es.update.script.params", update_params)
        .set("es.update.script.inline", update_script)
        .set("es.nodes.wan.only", "true");
```

注：

```
es.update.script.params 为执行更新需要的参数列表
es.update.script.inline 为执行update使用script脚本
```

服务等级协议SLA

BES服务等级协议SLA（V1.0）

协议生效时间：2020 年 9 月 1 日

本服务等级协议（Service Level Agreement，以下简称 “SLA”）规定了百度智能云向用户提供的Elasticsearch的服务可用性等级指标及赔偿方案。

1. 定义

服务周期：一个服务周期为一个自然月。

服务周期总分钟数：按照服务周期内的总天数 * 24（小时）* 60（分钟）计算。

服务不可用分钟数：当用户所有试图与指定的Elasticsearch集群建立连接（如数据写入、查询操作）的连续尝试均失败，且状态持续超过5分钟以上，则视为该分钟内该Elasticsearch集群服务不可用。 在一个服务周期内Elasticsearch集群服务不可用分钟数之和（相同分钟内的不同操作不叠加）即服务不可用分钟数。

月度服务费用：用户在一个自然月中就单个Elasticsearch集群所支付的服务费用总额，如果用户一次性支付了多个月份的服务费用，则将按照所购买的月数分摊计算月度服务费用。

2.服务可用性

2.1 服务可用性计算公式

服务可用性 = （1-服务不可用分钟数/服务周期总分钟数）* 100%

2.2 服务可用性承诺

Elasticsearch服务可用性不低于99.90%，如Elasticsearch未达到上述可用性承诺，客户可以根据本协议第3条约定获得赔偿。

3. 服务赔偿条款

3.1 赔偿标准

根据客户某一百度智能云账号下某一百度Elasticsearch集群的月度服务可用性，按照下表中的标准计算赔偿金额，赔偿方式仅限于用于购买百度Elasticsearch的代金券，且赔偿总额不超过未达到服务可用性承诺的当月客户就该账号下该集群的月度服务费。

服务可用性	赔偿代金券金额
低于99.90%但等于或高于99.00%	月度服务费用的10%
低于99.00%但等于或高于95.00%	月度服务费用的25%
低于95.00%	月度服务费用的100%

3.2 赔偿申请时限

客户可以在每月第五（5）个工作日后对上个月没有达到可用性的服务提出赔偿申请。赔偿申请必须限于在消息服务没有达到可用性的相关月份结束后两（2）个月内提出。超出申请时限的赔偿申请将不被受理。

3.3 除外情形

赔偿范围不包括以下原因所导致的不可用：

(1) 百度智能云预先通知用户后进行系统维护所引起的，包括割接、维修、升级和模拟故障演练；

(2) 任何百度智能云所属设备以外的网络、设备故障或配置调整引起的；

(3) 用户的应用程序或数据信息受到黑客攻击而引起的；

(4) 用户维护不当或保密不当致使数据、口令、密码等丢失或泄漏所引起的；

(5) 用户的疏忽或由用户授权的操作所引起的；

(6) 客户未遵循百度智能云产品使用文档或使用建议引起的，如客户集群基本状态为非正常运行状态（即非绿状态）、CPU利用率过高、内存负载过高、磁盘占用过满时（以上值均建议保持在50%以下），仍然进行频繁数据读写与变更的操作引起的不可用；

(7) 本地盘出现宕机数据会被擦除，依赖本地盘及本地盘中数据作为启动依赖项而导致的不可用；

(8) 因Elasticsearch人为配置操作引起的服务不可用；

(9) Elasticsearch实例在进行实例变更重启、强制重启操作过程中所引起的服务不可用；

(10) 不可抗力以及意外事件或其他非百度智能云原因所造成的百度Elasticsearch无法正常使用引起的。

4. 其他说明

(1) 在法律法规允许的范围内，百度智能云负责对本协议进行解释说明。

(2) 本协议一经公布立即生效，百度智能云有权对本SLA条款作出修改。如本SLA条款有任何修改，百度智能云将以网站公示或发送邮件的方式通知您。如您不同意百度智能云对SLA所做的修改，您有权停止使用Elasticsearch服务，如您继续使用Elasticsearch服务，则视为您接受修改后的SLA。

(3) 本协议项下百度智能云对于用户所有的通知均可通过网页公告、站内信、电子邮件、手机短信或其他形式等方式进行；该等通知于发送之日视为已送达收件人。因用户未及时获知百度智能云的服务变更或终止条款遭受损失的，百度智能云不承担任何责任。

(4) 本协议的订立、执行和解释及争议的解决均应适用中国法律并受中国法院管辖。如双方就本协议内容或其执行发生任何争议，双方应尽量友好协商解决；协商不成时，任何一方均可向北京市海淀区人民法院提起诉讼。

(5) 本协议构成双方对本协议之约定事项及其他有关事宜的完整协议，除本协议规定的之外，未赋予本协议各方其他权利。

(6) 如本协议中的任何协议无论因何种原因完全或部分无效或不具有执行力，本协议的其余协议仍应有效并且有约束力。

(7) 关于用户约束条款，详见《用户服务协议》中的"用户的权利与义务"相关条款内容。

(8) 关于服务商免责条款，详见《用户服务协议》中的"免责声明"相关条款内容。

数据迁移服务协议

数据迁移服务协议

当用户需要进行百度智能云Elasticsearch集群数据节点缩容操作、虚拟机更换或分片分布调整等情况，百度智能云Elasticsearch服务将协助迁移目标数据节点中的数据。由于该项操作本身具有一定的风险，请确保用户已了解并知悉该项操作的步骤及以下后果，并且同意本服务协议，再进行操作。

1. 由于百度智能云Elasticsearch集群本身的特性、数据节点的特性、网络传输的不确定性，数据迁移过程可能会受到集群健康状态、节点存储情况等多方面因素影响。迁移不成功（集群健康状态不佳、迁移过程中断或出现类似后果的情形）可能会导致源数据和目标数据丢失，或迁移成功但迁移前后数据不一致。
2. 百度智能云Elasticsearch不会干预有关集群数据节点的数据迁移过程，数据是否迁移成功以及迁移前后数据是否一致并非是百度云的责任。迁移是用户自主进行的一项操作，用户需要自行承担与迁移操作本身以及后果相关的责任。
3. 百度智能云郑重提示用户，请确保在用户已了解并知悉该项操作的步骤及后果的情况下进行操作；请做好迁移的相关准备以及灾备措施，包括但不限于在迁移前进行数据备份；确保迁移成功且数据一致后，再进行备份数据的删除等操作。

API3.0

集群相关接口

删除集群

接口说明

删除集群

请求URI

POST /api/bes/cluster/v2/delete

请求头参数

参数名称	参数类型	是否必须	描述	示例值
x-Region	String	是	区域	xRegion_example

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **clusterId** | **String** | 是|集群id，超过规定18位会返回500错误 | 218185657699405824 **refundReason** | **String** | 否|退款原因，预付费集群必填 | 业务测试结束

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **success** | **Boolean** | 操作是否成功 | true **status** | **Integer** | 状态码 | 200 **result** | **Result** | 返回信息 | result

Result字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **orderId** | **String** | 预付费退订订单ID，后付费为空 | 186f2566f2884534a65ed8cdc2499d82

请求示例

```
POST /api/bes/cluster/v2/delete
<公共请求头>
x-Region: xRegion_example


{
  "clusterId" : "218185657699405824"
}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{
  "result": {
  "orderId": "7954130a3961416f8eeec4f83021470c"
  },
  "success" : true,
  "status" : 200
}
```

错误码

请参考通用错误码