Elasticsearch Service

产品文档

目录

Elasticsearch Service 3
- 产品简介 3
  - 产品概述 3
  - 产品功能 4
  - 产品性能 5
  - 产品优势 20
  - 能力与限制说明 21
  - 相关概念 23
  - 应用场景 24
- 快速入门 25
  - 创建集群 25
  - 访问集群 29
    - 通过API访问集群 29
    - 通过客户端访问集群 38
    - 通过Kibana访问集群 45
- 操作指南 53
  - 访问控制 53
    - CAM访问控制配置 53
    - 集群安全管理 56
  - 管理集群 58
    - 集群状态 58
    - 高级配置 59
    - 调整配置 62
    - 重启集群 66
    - 销毁集群 68
  - 冷热分离与索引生命周期管理 69
  - 查询集群日志 75
  - 监控与告警 78
    - 查看监控 78
    - 配置告警 84
  - 数据备份 88
    - 使用对象存储进行备份及恢复 88
    - 自动快照备份 93
  - 升级 96
    - 升级ES集群 96
    - ES版本升级检查 100
  - SQL支持 109
- 最佳实践 110
  - 数据迁移和同步 110
    - 数据迁移 110
    - 数据接入 ES 116
    - MySQL 数据实时同步到 ES 125
  - 节点类型存储配置建议 134
  - 默认索引模板说明和调整 137
  - 监控告警配置建议 141
- 常见问题 142
  - 产品相关问题 142
  - 写入拒绝（Bulk Reject）问题 143
  - 磁盘 readonly 问题 146
  - ES 自研熔断器及常见问题 148

产品简介
产品概述

Elasticsearch Service（ES）是基于开源搜索引擎 Elasticsearch 构建的高可用、可伸缩的云端托管 Elasticsearch 服务。Elasticsearch 是一款分布式的基于 RESTful API 的搜索分析引擎，可以应用于日益增多的海量数据搜索和分析等应用场景。
ES 既集成了云计算在计算、存储、安全等领域的领先技术优势，又保持了 Elasticsearch 本身的兼容与开放，拥有丰富的集群管理功能以及安全、弹性、高可用等特性，可以帮助您简化集群部署、运营管理等基础运维工作，更加聚焦于业务本身。
通过ES，您可以快速构建海量数据存储搜索、实时日志分析等应用，例如网站搜索导航、企业级搜索、服务日志异常监控、点击流分析等。

主要组件

- Elasticsearch
  分布式搜索引擎，可以对海量数据进行存储、全文检索、统计分析等，提供了 RESTful API 以及各类语言客户端，可以灵活地按照业务需求进行开发。
- Kibana
  数据可视化工具，可以方便地对存入 Elasticsearch 集群的数据进行查询和分析。

产品功能

用户可以将 CVM、T DB、容器服务等其他云产品的实时日志或业务的存量及增量业务数据，汇聚传输到 ES 集群，进行数据的分布式存储、查询分析。

数据采集与同步

- 用户通过 Elasticsearch 中的 Beats 功能，可以把数据传输到 Elasticsearch 中进行存储，也可以传输到 Logstash 中进行自定义转换和解析后，再传输到 Elasticsearch 中。
- Elasticsearch 提供了易用的 RESTful API，用户可以自行开发客户端，调用数据存储 API，存储数据到 Elasticsearch 集群中。

数据存储

- ES 提供了多种规格的节点类型和高性能 SSD 磁盘，有效保障数据的读写性能。
- 支持弹性扩展到上百个节点，能达到 PB 级数据的存储，满足用户不同类型的业务场景。
- 支持故障节点探测及替换，保障集群高可用性。
- 具备全文检索功能。

数据查询分析可视化

- Elasticsearch 拥有全文检索、结构化搜索、数据过滤和指标统计等搜索功能，可以应用于信息搜索和数据分析等多种场景。
- Elasticsearch 提供了简单易用的 RESTful API 以及各种语言的客户端，用户可以很方便地构建自己的搜索服务。
- 使用 Kibana，用户可以方便地在浏览器里对集群的数据进行搜索和统计分析。

产品性能

使用 Elasticsearch 官方提供的 rally 压测工具和 geonames 数据集（大小3.2G，共11523468个文档），对 TCloudFinanceZone Elasticsearch Service 构建的两种规格的 ES 集群进行压测，性能结果如下：

2核8G 3个节点：

Metric	Task	Value	Unit
Total indexing time	-	18.5837	min
Min indexing time per shard	-	0.000533333	min
Median indexing time per shard	-	3.4134	min
Max indexing time per shard	-	4.21195	min
Total refresh time	-	2.40147	min
Min refresh time per shard	-	0.0009	min
Median refresh time per shard	-	0.45715	min
Max refresh time per shard	-	0.5655	min
Total flush time	-	0.328317	min
Min flush time per shard	-	0.0004	min
Median flush time per shard	-	0.0643583	min
Max flush time per shard	-	0.0972167	min
Total Young Gen GC	-	127.992	s
Total Old Gen GC	-	1.327	s
Store size	-	3.22145	GB
Translog size	-	2.80E-07	GB
Heap used for segments	-	10.8161	MB
Heap used for doc values	-	0.0831299	MB
Heap used for terms	-	9.62618	MB
Heap used for norms	-	0.0795288	MB
Heap used for points	-	0.217488	MB

Metric	Task	Value	Unit
Heap used for stored fields	-	0.809799	MB
Segment count	-	102	-
Min Throughput	index-append	41895.6	docs/s
Median Throughput	index-append	42562	docs/s
Max Throughput	index-append	43352	docs/s
50th percentile latency	index-append	757.636	ms
90th percentile latency	index-append	1388.11	ms
99th percentile latency	index-append	2650.44	ms
99.9th percentile latency	index-append	5270.13	ms
100th percentile latency	index-append	6535.29	ms
50th percentile service time	index-append	757.636	ms
90th percentile service time	index-append	1388.11	ms
99th percentile service time	index-append	2650.44	ms
99.9th percentile service time	index-append	5270.13	ms
100th percentile service time	index-append	6535.29	ms
error rate	index-append	0	%
Min Throughput	index-stats	99.99	ops/s
Median Throughput	index-stats	100.04	ops/s
Max Throughput	index-stats	100.06	ops/s
50th percentile latency	index-stats	6.04131	ms
90th percentile latency	index-stats	6.56561	ms
99th percentile latency	index-stats	10.7941	ms
99.9th percentile latency	index-stats	22.6768	ms
100th percentile latency	index-stats	24.5623	ms
50th percentile service time	index-stats	5.9341	ms

Metric	Task	Value	Unit
90th percentile service time	index-stats	6.42644	ms
99th percentile service time	index-stats	7.56809	ms
99.9th percentile service time	index-stats	22.5948	ms
100th percentile service time	index-stats	24.4467	ms
error rate	index-stats	0	%
Min Throughput	node-stats	99.93	ops/s
Median Throughput	node-stats	100.06	ops/s
Max Throughput	node-stats	100.33	ops/s
50th percentile latency	node-stats	6.74088	ms
90th percentile latency	node-stats	7.28822	ms
99th percentile latency	node-stats	8.62256	ms
99.9th percentile latency	node-stats	13.271	ms
100th percentile latency	node-stats	13.9379	ms
50th percentile service time	node-stats	6.64634	ms
90th percentile service time	node-stats	7.18403	ms
99th percentile service time	node-stats	8.34209	ms
99.9th percentile service time	node-stats	13.1784	ms
100th percentile service time	node-stats	13.8411	ms
error rate	node-stats	0	%
Min Throughput	default	46.85	ops/s
Median Throughput	default	47.58	ops/s
Max Throughput	default	47.7	ops/s
50th percentile latency	default	1023.59	ms
90th percentile latency	default	1890.67	ms
99th percentile latency	default	2017.58	ms

Metric	Task	Value	Unit
99.9th percentile latency	default	2026.21	ms
100th percentile latency	default	2026.21	ms
50th percentile service time	default	20.2853	ms
90th percentile service time	default	24.3425	ms
99th percentile service time	default	33.3526	ms
99.9th percentile service time	default	54.702	ms
100th percentile service time	default	74.2832	ms
error rate	default	0	%
Min Throughput	term	199.82	ops/s
Median Throughput	term	200.02	ops/s
Max Throughput	term	200.05	ops/s
50th percentile latency	term	4.54844	ms
90th percentile latency	term	9.97587	ms
99th percentile latency	term	18.3546	ms
99.9th percentile latency	term	21.2112	ms
100th percentile latency	term	21.5267	ms
50th percentile service time	term	4.39147	ms
90th percentile service time	term	4.84527	ms
99th percentile service time	term	6.62189	ms
99.9th percentile service time	term	19.4932	ms
100th percentile service time	term	21.4435	ms
error rate	term	0	%
Min Throughput	phrase	199.68	ops/s
Median Throughput	phrase	200.03	ops/s
Max Throughput	phrase	200.11	ops/s

Metric	Task	Value	Unit
50th percentile latency	phrase	3.57488	ms
90th percentile latency	phrase	4.62139	ms
99th percentile latency	phrase	19.6935	ms
99.9th percentile latency	phrase	24.9076	ms
100th percentile latency	phrase	25.0486	ms
50th percentile service time	phrase	3.4742	ms
90th percentile service time	phrase	4.0265	ms
99th percentile service time	phrase	7.57333	ms
99.9th percentile service time	phrase	18.9011	ms
100th percentile service time	phrase	23.8045	ms
error rate	phrase	0	%
Min Throughput	country_agg_uncached	4.99	ops/s
Median Throughput	country_agg_uncached	5	ops/s
Max Throughput	country_agg_uncached	5	ops/s
50th percentile latency	country_agg_uncached	197.534	ms
90th percentile latency	country_agg_uncached	217.842	ms
99th percentile latency	country_agg_uncached	271.988	ms
100th percentile latency	country_agg_uncached	275.963	ms
50th percentile service time	country_agg_uncached	194.061	ms
90th percentile service time	country_agg_uncached	209.086	ms
99th percentile service time	country_agg_uncached	216.432	ms
100th percentile service time	country_agg_uncached	217.275	ms
error rate	country_agg_uncached	0	%
Min Throughput	country_agg_cached	99.97	ops/s
Median Throughput	country_agg_cached	100.05	ops/s

Metric	Task	Value	Unit
Max Throughput	country_agg_cached	100.08	ops/s
50th percentile latency	country_agg_cached	4.9212	ms
90th percentile latency	country_agg_cached	5.44065	ms
99th percentile latency	country_agg_cached	7.15509	ms
99.9th percentile latency	country_agg_cached	16.9407	ms
100th percentile latency	country_agg_cached	17.8111	ms
50th percentile service time	country_agg_cached	4.81515	ms
90th percentile service time	country_agg_cached	5.29377	ms
99th percentile service time	country_agg_cached	6.38482	ms
99.9th percentile service time	country_agg_cached	16.8318	ms
100th percentile service time	country_agg_cached	17.7311	ms
error rate	country_agg_cached	0	%
Min Throughput	scroll	25.02	pages/s
Median Throughput	scroll	25.02	pages/s
Max Throughput	scroll	25.03	pages/s
50th percentile latency	scroll	760.634	ms
90th percentile latency	scroll	794.699	ms
99th percentile latency	scroll	864.897	ms
100th percentile latency	scroll	874.768	ms
50th percentile service time	scroll	760.32	ms
90th percentile service time	scroll	794.397	ms
99th percentile service time	scroll	864.658	ms
100th percentile service time	scroll	874.556	ms
error rate	scroll	0	%
Min Throughput	expression	2	ops/s

Metric	Task	Value	Unit
Median Throughput	expression	2	ops/s
Max Throughput	expression	2	ops/s
50th percentile latency	expression	382.483	ms
90th percentile latency	expression	414.775	ms
99th percentile latency	expression	455.236	ms
100th percentile latency	expression	473.181	ms
50th percentile service time	expression	382.298	ms
90th percentile service time	expression	414.577	ms
99th percentile service time	expression	455.11	ms
100th percentile service time	expression	472.998	ms
error rate	expression	0	%
Min Throughput	painless_static	1.5	ops/s
Median Throughput	painless_static	1.5	ops/s
Max Throughput	painless_static	1.5	ops/s
50th percentile latency	painless_static	480.188	ms
90th percentile latency	painless_static	505.003	ms
99th percentile latency	painless_static	529.066	ms
100th percentile latency	painless_static	547.199	ms
50th percentile service time	painless_static	479.938	ms
90th percentile service time	painless_static	504.731	ms
99th percentile service time	painless_static	528.857	ms
100th percentile service time	painless_static	546.954	ms
error rate	painless_static	0	%
Min Throughput	painless_dynamic	1.5	ops/s
Median Throughput	painless_dynamic	1.5	ops/s

Metric	Task	Value	Unit
Max Throughput	painless_dynamic	1.5	ops/s
50th percentile latency	painless_dynamic	469.434	ms
90th percentile latency	painless_dynamic	508.615	ms
99th percentile latency	painless_dynamic	581.127	ms
100th percentile latency	painless_dynamic	621.998	ms
50th percentile service time	painless_dynamic	469.178	ms
90th percentile service time	painless_dynamic	508.349	ms
99th percentile service time	painless_dynamic	580.819	ms
100th percentile service time	painless_dynamic	621.799	ms
error rate	painless_dynamic	0	%
error rate	large_terms	0	%
Min Throughput	large_filtered_terms	1.52	ops/s
Median Throughput	large_filtered_terms	1.52	ops/s
Max Throughput	large_filtered_terms	1.52	ops/s
50th percentile latency	large_filtered_terms	39664.6	ms
90th percentile latency	large_filtered_terms	46001.3	ms
99th percentile latency	large_filtered_terms	47328	ms
100th percentile latency	large_filtered_terms	47488.6	ms
50th percentile service time	large_filtered_terms	651.731	ms
90th percentile service time	large_filtered_terms	673.319	ms
99th percentile service time	large_filtered_terms	715.941	ms
100th percentile service time	large_filtered_terms	723.06	ms
error rate	large_filtered_terms	0	%
Min Throughput	large_prohibited_terms	1.55	ops/s
Median Throughput	large_prohibited_terms	1.56	ops/s

Metric	Task	Value	Unit
Max Throughput	large_prohibited_terms	1.57	ops/s
50th percentile latency	large_prohibited_terms	35606.8	ms
90th percentile latency	large_prohibited_terms	40847.6	ms
99th percentile latency	large_prohibited_terms	42170.5	ms
100th percentile latency	large_prohibited_terms	42329.2	ms
50th percentile service time	large_prohibited_terms	648.82	ms
90th percentile service time	large_prohibited_terms	672.114	ms
99th percentile service time	large_prohibited_terms	722.666	ms
100th percentile service time	large_prohibited_terms	733.307	ms
error rate	large_prohibited_terms	0	%

4核16G 3个节点：

Metric	Task	Value	Unit
Total indexing time	-	20.1957	min
Min indexing time per shard	-	0.000733333	min
Median indexing time per shard	-	3.77953	min
Max indexing time per shard	-	4.63752	min
Total merge time	-	1.57487	min
Min merge time per shard	-	0	min
Median merge time per shard	-	0.176658	min
Max merge time per shard	-	0.634067	min
Total merge throttle time	-	0.55105	min
Min merge throttle time per shard	-	0	min
Median merge throttle time per shard	-	0.065	min
Max merge throttle time per shard	-	0.217033	min

Metric	Task	Value	Unit
Total refresh time	-	1.41135	min
Min refresh time per shard	-	0.00106667	min
Median refresh time per shard	-	0.269958	min
Max refresh time per shard	-	0.345733	min
Total flush time	-	0.533133	min
Min flush time per shard	-	0.000566667	min
Median flush time per shard	-	0.115592	min
Max flush time per shard	-	0.136683	min
Total Young Gen GC	-	70.747	s
Total Old Gen GC	-	0.92	s
Store size	-	3.31581	GB
Translog size	-	2.80E-07	GB
Heap used for segments	-	11.0486	MB
Heap used for doc values	-	0.100529	MB
Heap used for terms	-	9.84413	MB
Heap used for norms	-	0.0755005	MB
Heap used for points	-	0.216421	MB
Heap used for stored fields	-	0.811981	MB
Segment count	-	97	-
Min Throughput	index-append	74421.1	docs/s
Median Throughput	index-append	75636.9	docs/s
Max Throughput	index-append	76877.4	docs/s
50th percentile latency	index-append	377.922	ms
90th percentile latency	index-append	663.055	ms
99th percentile latency	index-append	3068.99	ms

Metric	Task	Value	Unit
100th percentile latency	index-append	5554.97	ms
50th percentile service time	index-append	377.922	ms
90th percentile service time	index-append	663.055	ms
99th percentile service time	index-append	3068.99	ms
100th percentile service time	index-append	5554.97	ms
error rate	index-append	0	%
Min Throughput	index-stats	99.93	ops/s
Median Throughput	index-stats	100.04	ops/s
Max Throughput	index-stats	100.06	ops/s
50th percentile latency	index-stats	6.62305	ms
90th percentile latency	index-stats	7.35102	ms
99th percentile latency	index-stats	18.0909	ms
99.9th percentile latency	index-stats	24.5381	ms
100th percentile latency	index-stats	24.7431	ms
50th percentile service time	index-stats	6.50957	ms
90th percentile service time	index-stats	7.18652	ms
99th percentile service time	index-stats	9.38455	ms
99.9th percentile service time	index-stats	24.4424	ms
100th percentile service time	index-stats	24.6576	ms
error rate	index-stats	0	%
Min Throughput	node-stats	99.92	ops/s
Median Throughput	node-stats	100.04	ops/s
Max Throughput	node-stats	100.25	ops/s
50th percentile latency	node-stats	7.15655	ms
90th percentile latency	node-stats	7.96104	ms

Metric	Task	Value	Unit
99th percentile latency	node-stats	10.2362	ms
99.9th percentile latency	node-stats	25.7397	ms
100th percentile latency	node-stats	29.1573	ms
50th percentile service time	node-stats	7.04389	ms
90th percentile service time	node-stats	7.84655	ms
99th percentile service time	node-stats	9.13249	ms
99.9th percentile service time	node-stats	10.7357	ms
100th percentile service time	node-stats	29.072	ms
error rate	node-stats	0	%
Min Throughput	default	41.88	ops/s
Median Throughput	default	42.18	ops/s
Max Throughput	default	42.52	ops/s
50th percentile latency	default	3789.31	ms
90th percentile latency	default	5170.79	ms
99th percentile latency	default	5582.03	ms
99.9th percentile latency	default	5610.7	ms
100th percentile latency	default	5618.05	ms
50th percentile service time	default	23.1496	ms
90th percentile service time	default	25.8865	ms
99th percentile service time	default	33.249	ms
99.9th percentile service time	default	45.493	ms
100th percentile service time	default	62.4174	ms
error rate	default	0	%
Min Throughput	term	199.08	ops/s
Median Throughput	term	200	ops/s

Metric	Task	Value	Unit
Max Throughput	term	200.03	ops/s
50th percentile latency	term	5.02391	ms
90th percentile latency	term	21.18	ms
99th percentile latency	term	35.2251	ms
99.9th percentile latency	term	37.4827	ms
100th percentile latency	term	37.6907	ms
50th percentile service time	term	4.61812	ms
90th percentile service time	term	5.10619	ms
99th percentile service time	term	6.8135	ms
99.9th percentile service time	term	22.1183	ms
100th percentile service time	term	25.0033	ms
error rate	term	0	%
Min Throughput	phrase	199.61	ops/s
Median Throughput	phrase	200.04	ops/s
Max Throughput	phrase	200.8	ops/s
50th percentile latency	phrase	3.86572	ms
90th percentile latency	phrase	4.96583	ms
99th percentile latency	phrase	22.5681	ms
99.9th percentile latency	phrase	33.5684	ms
100th percentile latency	phrase	28.2658	ms
50th percentile service time	phrase	3.5689	ms
90th percentile service time	phrase	4.2535	ms
99th percentile service time	phrase	8.6957	ms
99.9th percentile service time	phrase	24.5685	ms
100th percentile service time	phrase	27.6584	ms

Metric	Task	Value	Unit
error rate	phrase	0	%
Min Throughput	country_agg_uncached	4.99	ops/s
Median Throughput	country_agg_uncached	5	ops/s
Max Throughput	country_agg_uncached	5	ops/s
50th percentile latency	country_agg_uncached	182.291	ms
90th percentile latency	country_agg_uncached	201.585	ms
99th percentile latency	country_agg_uncached	257.343	ms
100th percentile latency	country_agg_uncached	267.904	ms
50th percentile service time	country_agg_uncached	181.161	ms
90th percentile service time	country_agg_uncached	196.189	ms
99th percentile service time	country_agg_uncached	216.762	ms
100th percentile service time	country_agg_uncached	267.778	ms
error rate	country_agg_uncached	0	%
Min Throughput	country_agg_cached	99.95	ops/s
Median Throughput	country_agg_cached	100.05	ops/s
Max Throughput	country_agg_cached	100.07	ops/s
50th percentile latency	country_agg_cached	5.57249	ms
90th percentile latency	country_agg_cached	6.47982	ms
99th percentile latency	country_agg_cached	9.33674	ms
99.9th percentile latency	country_agg_cached	27.5319	ms
100th percentile latency	country_agg_cached	32.0567	ms
50th percentile service time	country_agg_cached	5.4601	ms
90th percentile service time	country_agg_cached	6.25153	ms
99th percentile service time	country_agg_cached	7.83564	ms
99.9th percentile service time	country_agg_cached	13.6439	ms

Metric	Task	Value	Unit
100th percentile service time	country_agg_cached	31.9487	ms
error rate	country_agg_cached	0	%
Min Throughput	scroll	25.01	pages/s
Median Throughput	scroll	25.03	pages/s

产品优势

ES 提供了云端托管的服务，您可容易地创建和管理 Elasticsearch 集群，并且能够保障生产环境中的高可用性。以下是对产品核心优势的介绍：

易于部署和管理

您只需通过简单的操作，即可在几分钟内创建一个 ES 集群，免去软硬件部署调试的复杂流程。此外，ES 还提供了便捷的集群运营管理工具、Kibana 页面以及完善的集群监控和告警系统，以满足客户日常的集群运营管理需求。

弹性扩缩容

ES 提供了多种类型的节点规格和存储介质，您可以根据业务需要进行选择。随着业务的发展，您可以动态调整集群的配置扩容或缩容，实时保障业务的发展需要并有效控制成本。

高可用

ES 提供了 COS 数据备份策略可以定时备份数据，保障数据在意外情况下丢失时快速恢复。此外还有为保障集群稳定而进行的内核优化等策略，可以全方位地保障数据的安全和服务的稳定。

安全加固

通过部署在逻辑隔离的私有网络 VPC，客户可以完全掌控自己的环境配置，自定义网络访问控制列表（Access Control List）和安全组，通过 CAM 来管理TCloudFinanceZone账户下的资源的访问权限等，切实保证您云上资源的安全性。

开放与服务集成

支持完整的 ELK 产品体系，兼容标准的开源 RESTful API 和生态组件，可以与对象存储 COS、网络流日志、消息队列等产品集成，为用户提供数据传输和备份能力，以满足不同业务场景需要。

能力与限制说明

ES 是基于开源软件 Elasticsearch 开发的云上 PaaS 服务。通过ES 您可以快速搭建 Elasticsearch 集群服务，开发日志分析、数据搜索等应用。下面介绍产品能力和使用限制。

产品构成

ES 包含核心部件 Elasticsearch 集群和数据可视化分析工具 Kibana 两部分。其中，Elasticsearch为开源版本，不包含任何Xpack相关的功能。数据采集和传输到 ES 集群的部分，您可以根据自己的业务需要，部署数据采集工具如 Beats、Logstash 或开发应用程序，将数据写入 ES 集群。

可选配置

节点规格

参数名称	CPU 核数	内存
ES.S1.SMALL2	1	2G
ES.S1.MEDIUM4	2	4G
ES.S1.MEDIUM8	2	8G
ES.S1.LARGE16	4	16G
ES.S1.2XLARGE32	8	32G
ES.S1.4XLARGE32	16	32G
ES.S1.4XLARGE64	16	64G

其中1核2G仅为测试用，不建议用于生产环境。

存储

SSD 云盘，单个节点，磁盘大小限制为100GB - 2TB。

节点个数

节点个数限制为2 - 50。由于 ES 集群常以分布式多节点的形式构成，因此需要有主节点对集群进行统一的管理。为了防止可能的节点故障导致异常问题，建议您至少选择3个节点构建集群。

配置选择

可参考最佳实践文档的 节点类型存储配置建议。

网络访问

VPC 内网访问

为了保证数据的安全，ES 构建在用户 VPC 内，也只能通过 VPC 访问集群进行数据的写入和查询。如果您需要通过公网访问 ES 集群，进行开发调试，可以通过 VPN 的方式，打通本地 IDC 和云上 VPC，或通过外网代理的形式。同时

也提醒您做好数据安全防护。

Kibana 页面

支持通过公网访问 Kibana 页面，为了保证数据安全，Kibana 页面需要设置密码。

VPC 网络选择

ES 创建完成后，不支持切换 VPC 网络，请在创建集群时，提前规划业务部署。

相关概念

Elasticsearch 集群一般是由多个节点共同组成的分布式集群，节点之间互通，彼此配合，共同对外提供搜索和索引服务（节点之间能够将客户端请求转向到合适的节点）。不同节点负责不同角色，有的负责一个，有的可能负责多个。Elasticsearch 中有多个节点角色，其中有两个比较主要的是数据节点和主节点。

数据节点（Data Node）

主要承担数据存储和数据处理的工作，保存索引分片，处理与数据相关的操作。例如 CRUD、搜索、聚合等各种 I/O、内存和 CPU 密集型操作。集群使用过程中，需要注意监控数据节点的资源使用率，并在服务过载时通过添加更多的数据节点，进行集群扩容，以保障集群的稳定性。

主节点（Master Node）

负责轻量化整个集群范围内的操作，例如创建或删除索引、跟踪哪些节点是集群的一部分以及决定将哪些分片分配给哪些节点。对于集群健康来说，拥有一个稳定的主节点非常重要。

备选主节点（Master-eligible node）

指有资格被选为主节点的节点，任何符合主节点条件的节点（默认情况下为所有节点）都可以通过主选择过程被选为主节点。
默认情况下，所有节点都是数据节点，也都是备选主节点，这对于小型集群来说非常方便。由于数据节点处理索引和搜索数据的请求都是 I/O、内存和 CPU 密集型工作，可能对节点的资源造成压力。随着集群的增长，为了确保主节点稳定且不受压力，保障集群的稳定，需将主节点和数据节点分离。

专用主节点（Dedicated Master Node）

指 Elasticsearch 集群中，设置了只能作为主节点的节点。

专用主节点配置建议

设置专用主节点主要是为保障集群增大时的稳定性，建议专用主节点个数至少为3个。

- 专用主节点个数为1：只有1个备选主节点，discovery.zen.minimum_master_nodes 设置为1，网络发生故障时缺少备份。
- 专用主节点个数为2：有2个备选主节点，minimum_master_nodes 设置为1，有备份节点，但网络发生故障，在重新选主时，有发生脑裂的风险（每个备选主节点都将自己设置为主节点）；minimum_master_nodes 设置为2，发生故障时，可选主节点个数达不到要求，不能选出主节点。
- 专用主节点个数为3：有3个备选主节点，discovery.zen.minimum_master_nodes 设置为2，网络发生故障时，丢失一个可选主节点，在重新选主时，能有效选出主节点。
  其他详细说明，可参考 ES 节点说明。

应用场景

日志分析

网站服务器、移动设备、IOT 传感器等设备产生的日志都存在节点分散、种类多样、规模庞大等问题，这对需要通过日志搜索进行异常问题定位和业务分析等工作造成了很大的挑战。ES 提供了弹性可扩展、实时的集中式存储方案以及全文搜索功能，方便日志的统一管理和查询，帮助用户快速定位和发现问题，提高解决问题的效率。

全文检索

电商商品搜索、移动应用搜索、企业内部信息搜索等海量数据下的站内搜索服务是高效获取信息的必要途径，ES 拥有全文检索功能，对结构化和非结构化数据都有良好的支持，同时还提供了简单易用的 RESTful API 和各种语言的客户端，方便用户快速搭建稳定的搜索服务，整合到已有的业务框架中。

商业智能 BI

在数据驱动运营的行业背景下，电子商务、移动应用、广告媒体等业务都需要借助数据分析和数据挖掘来辅助商业决策，而规模庞大的业务数据对数据的统计分析造成了很大的挑战。ES 拥有结构化查询的能力，支持复杂的过滤和聚合统计功能，帮助客户对海量数据进行高效地个性化统计分析、发现问题与机会、辅助商业决策，让数据产生真正的价值。

快速入门
创建集群

集群是 ES 提供托管 Elasticsearch 服务的基本单元，也是用户使用和管理 Elasticsearch 服务的主要对象。本文为您介绍通过控制台，快速创建 Elasticsearch 集群。

前提条件

已创建账号。

操作步骤

登录控制台

登录 ES 控制台，单击【新建】进入创建购买页。

创建集群

1. 选择集群配置

- 地域：选择希望集群创建的地域。
- Elasticsearch版本：支持7.5.1、5.6.4、6.4.3、6.8.2等版本。
- 网络/可用区及子网：ES 部署在 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。同一个 VPC 内，不同可用区子网之间是互通的。

> ES 集群创建完成后，不支持修改调整 VPC。

> 特殊网络问题说明

跨 VPC 访问：如果需要通过同一个地域内位于其他 VPC 的 CVM 访问 ES 集群，可以考虑 对等连接方案，将同一地域两个不同的 VPC 进行打通。

- 数据节点部署模式：支持单一模式和冷热模式。冷热集群是可以在一个集群内包含冷、热两种属性的节点，对读写性能要求高的热点数据可以存放在热节点，对存储量需求比较大但对读写性能要求较低的索引可以存放在冷节点上，既保证了热数据的读写性能，又可以降低存储成本。详细介绍请参见 冷热分离与索引生命周期管理。
- 节点机型：集群中每个节点机型的规格，不同规格包含不同的 CPU 核数和内存，ES 支持的节点规格类型以及选择建议可参见 节点类型存储配置建议。
- 节点数量：集群中包含的节点数量。

- 节点存储类型：支持高性能云盘和 SSD 云盘。
- 单节点存储：每个节点配置的磁盘容量，整个集群的存储量 = 单个节点存储 * 节点个数。
- 专用主节点：当集群具有一定规模时，可以 配置专用主节点，进一步保障集群稳定性。
- 配置专用主节点：开启。
- 专用主节点机型：专用主节点机型不要求同数据节点机型一致。
- 专用主节点个数：可选3个或5个，奇数个能保证高可用并预防脑裂风险。
- Kibana 节点：系统免费提供，1核2G，暂不支持修改。

2. 设置名称及密码

单击【下一步：设置名称及密码】。

- 集群名：自定义集群名称，不作为全局唯一标示，可以设置为业务相关描述。
- 用户名：默认为 elastic，不能修改。用于 Kibana 页面登录。
- 密码：上一项用户名对应的密码，请按照规则设置，并妥善保存，如果密码忘记，可在详情页重置。

3. 确认配置信息

单击【下一步：确认配置信息】，确认配置。
对按量计费，单击【开通】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。

创建完成

开通成功后，就可以跳转到 ES 控制台查看刚刚创建的集群，集群将在几分钟内创建完成。

集群开发应用及管理

集群访问

ES 提供了多种类型的客户端访问集群，快速开始，详情请参见 通过 API 访问集群、通过客户端访问集群、通过Kibana访问集群。

集群监控

TCloudFinanceZone ES，提供了丰富的监控指标，可以帮助在集群使用过程中查看集群的状态。详情请参见 查看监控。

集群调整配置

随着业务数据量和访问量的增加，可以弹性地调整集群配置，详情请参见 调整配置。

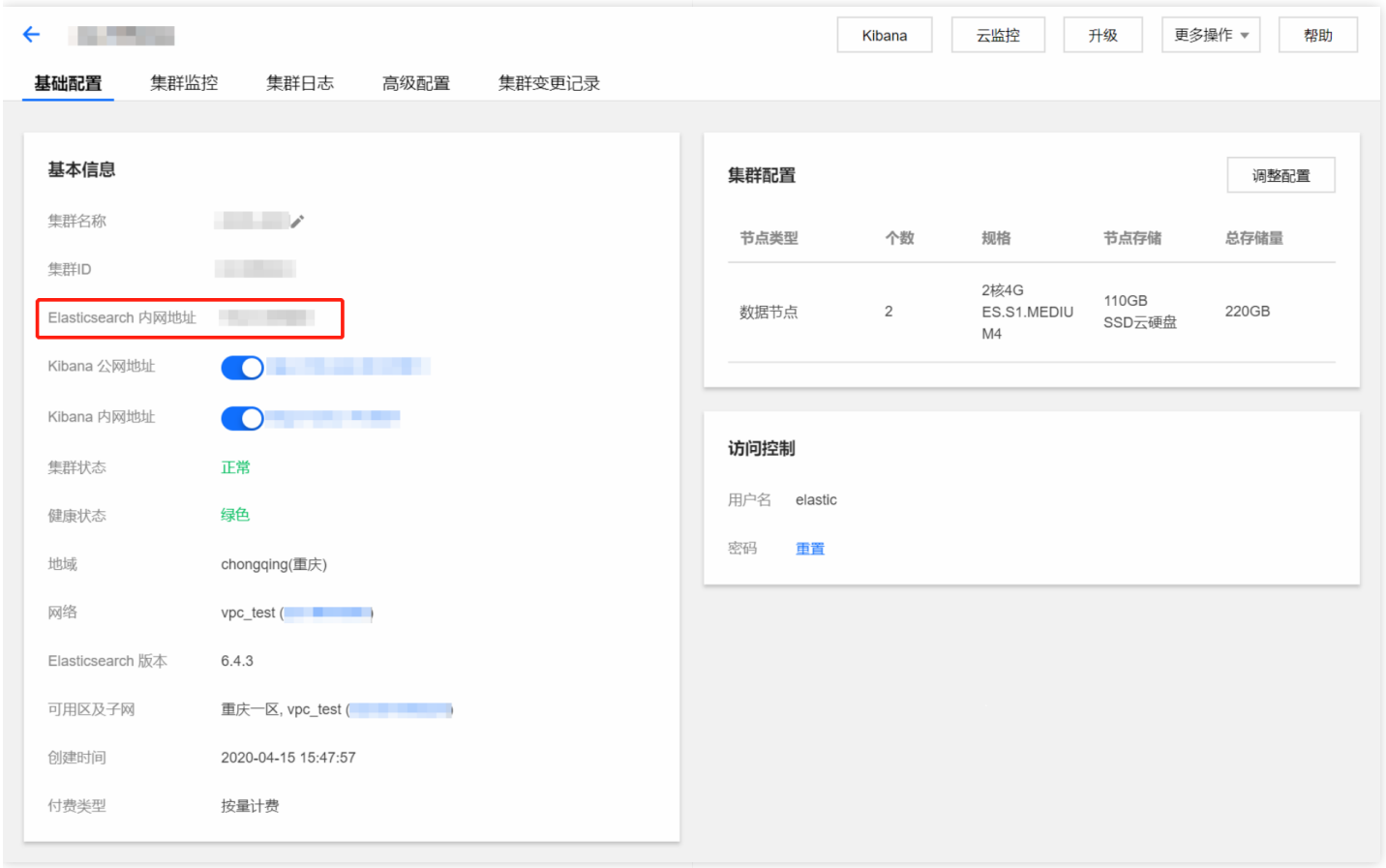
访问集群
通过API访问集群

Elasticsearch 提供了功能全面的 RESTful API 与集群交互，详情请参见 Elasticsearch 官方的 API 文档。

TCloudFinanceZone ES 构建在用户 VPC 内，用户可以通过位于同一 VPC 下的 CVM 作为客户端访问 ES 集群。可通过内网访问方式访问 ES 集群。

查看内网访问地址

在集群列表页，单击【集群 ID】进入详情页：
对于内网地址，在基础配置中可直接查看。



测试访问

可通过 curl 的方式测试访问集群，不支持通过 ping 的方式测试连通性。

测试服务是否可访问

- 输入命令：

```
curl -XGET http://10.0.17.2:9200
```

- 返回如下，表示集群访问正常，具体参数的值会根据集群的版本有所不同：

```
{
  "name": "15589826570000*****",
  "cluster_name": "es-******",
  "cluster_uuid": "NGIm1M_zRw-L3o_gH****",
  "version": {
    "number": "6.4.3",
    "build_flavor": "default",
    "build_type": "zip",
    "build_hash": "fe40335",
    "build_date": "2019-05-17T14:22:47.286024Z",
    "build_snapshot": false,
    "lucene_version": "7.4.0",
    "minimum_wire_compatibility_version": "5.6.0",
    "minimum_index_compatibility_version": "5.0.0"
  },
  "tagline": "You Know, for Search"
}
```

创建文档

创建单个文档

- 输入命令行：

```
curl -XPUT http://10.0.0.2:9200/china/city/beijing -H 'Content-Type: application/json' -d'
{
"name":"北京市",
"province":"北京市",
"lat":39.9031324643,
"lon":116.4010433787,
"x":6763,
"level.range":4,
"level.level":1,
"level.name":"一线城市",
```

```
"y":6381,
"cityNo":1
}
'
```

- 响应如下：

```
{
    "_index":"china",
    "_type":"city",
    "_id":"beijing",
    "_version":1,
    "result":"created",
    "_shards":{
        "total":2,
        "successful":1,
        "failed":0
    },
    "created":true
}
```

创建多个文档

- 输入命令行：

```
curl -XPOST http://10.0.0.2:9200/_bulk -H 'Content-Type: application/json' -d'
{ "index" : { "_index": "china", "_type" : "city", "_id" : "beijing" } }
{"name":"北京市","province":"北京市","lat":39.9031324643,"lon":116.4010433787,"x":6763,"level.range":4,"level.level":1,"level.name":"一线城市","y":6381,"cityNo":1}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "shanghai" } }
{"name":"上海市","province":"上海市","lat":31.2319526784,"lon":121.469443249,"x":7779,"level.range":4,"level.level":1,"level.name":"一线城市","y":4409,"cityNo":2}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "guangzhou" } }
{"name":"广州市","province":"广东省越秀区吉祥路79号","lat":23.1317146641,"lon":113.2595185241,"x":6173,"level.range":4,"level.level":1,"level.name":"一线城市","y":2560,"cityNo":3}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "shenzhen" } }
{"name":"深圳市","province":"广东省福田区新园路37号","lat":22.5455465546,"lon":114.0527779134,"x":6336,"level.range":4,"level.level":1,"level.name":"一线城市","y":2429,"cityNo":4}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "chengdu" } }
{"name":"成都市","province":"四川省锦江区红星路4段-88号-附1号","lat":30.6522796787,"lon":104.0725574128,"x":4387,"level.level":2,"level.range":19,"level.name":"新一线城市","y":4304,"cityNo":5}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "hangzhou" } }
{"name":"杭州市","province":"浙江省拱墅区环城北路316号","lat":30.2753694112,"lon":120.1509063337,"x":7530,"level.level":2,"level.range":19,"level.name":"新一线城市","y":4182,"cityNo":6}
```

'

- 响应如下：

"took":9,"errors":false,"items":[{"index":{"_index":"china","_type":"city","_id":"beijing","_version":4,"result":"updated","_shards":{"total":2,"successful":2,"failed":0},"created":false,"status":200}},{"index":{"_index":"china","_type":"city","_id":"shanghai","_version":2,"result":"updated","_shards":{"total":2,"successful":2,"failed":0},"created":false,"status":200}},{"index":{"_index":"china","_type":"city","_id":"guangzhou","_version":1,"result":"created","_shards":{"total":2,"successful":2,"failed":0},"created":true,"status":201}},{"index":{"_index":"china","_type":"city","_id":"shenzhen","_version":1,"result":"created","_shards":{"total":2,"successful":2,"failed":0},"created":true,"status":201}},{"index":{"_index":"china","_type":"city","_id":"chengdu","_version":2,"result":"updated","_shards":{"total":2,"successful":2,"failed":0},"created":false,"status":200}},{"index":{"_index":"china","_type":"city","_id":"hangzhou","_version":2,"result":"updated","_shards":{"total":2,"successful":2,"failed":0},"created":false,"status":200}}]}

更新文档

重复上文创建单个文档的输入代码，即可更新指定 ID beijing 的文档。 响应如下：

{"_index":"china","_type":"city","_id":"beijing","_version":2,"result":"updated","_shards":{"total":2,"successful":2,"failed":0},"created":false}

查询文档

查询指定 ID

- 输入命令行：

```
curl -XGET 'http://10.0.0.2:9200/china/city/beijing?pretty' -H 'Content-Type: application/json'
```

- 响应如下：

{
  "_index" : "china",
  "_type" : "city",
  "_id" : "beijing",
  "_version" : 4,

```
  "found" : true,
  "_source" : {
    "name" : "北京市",
    "province" : "北京市",
    "lat" : 39.9031324643,
    "lon" : 116.4010433787,
    "x" : 6763,
    "level.range" : 4,
    "level.level" : 1,
    "level.name" : "一线城市",
    "y" : 6381,
    "cityNo" : 1
  }
}
```

查询某个索引

- 输入命令行：

```
curl -XGET 'http://10.0.0.2:9200/china/city/_search?pretty' -H 'Content-Type: application/json'
```

- 响应如下：

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 6,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "china",
        "_type" : "city",
        "_id" : "guangzhou",
        "_score" : 1.0,
        "_source" : {
          "name" : "广州市",
          "province" : "广东省越秀区吉祥路79号",
```

```
            "lat" : 23.1317146641,
            "lon" : 113.2595185241,
            "x" : 6173,
            "level.range" : 4,
            "level.level" : 1,
            "level.name" : "一线城市",
            "y" : 2560,
            "cityNo" : 3
          }
        }]
    },
    ......
}
```

复杂查询

- 模拟 SQL：

```
select * from city where level.level=2
curl -XGET http://10.0.0.2:9200/china/city/_search?pretty -H 'Content-Type: application/json' -d'
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "term" : {
                    "level.level" : 2
                }
            }
        }
    }
}'
```

- 响应如下：

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
```

```
    "total" : 2,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "china",
        "_type" : "city",
        "_id" : "chengdu",
        "_score" : 1.0,
        "_source" : {
          "name" : "成都市",
          "province" : "四川省锦江区红星路4段-88号-附1号",
          "lat" : 30.6522796787,
          "lon" : 104.0725574128,
          "x" : 4387,
          "level.level" : 2,
          "level.range" : 19,
          "level.name" : "新一线城市",
          "y" : 4304,
          "cityNo" : 5
        }
      },
      {
        "_index" : "china",
        "_type" : "city",
        "_id" : "hangzhou",
        "_score" : 1.0,
        "_source" : {
          "name" : "杭州市",
          "province" : "浙江省拱墅区环城北路316号",
          "lat" : 30.2753694112,
          "lon" : 120.1509063337,
          "x" : 7530,
          "level.level" : 2,
          "level.range" : 19,
          "level.name" : "新一线城市",
          "y" : 4182,
          "cityNo" : 6
        }
      }
    ]
  }
}
```


聚合查询

- 模拟 SQL：

```
select level.level, count(1) from city group by level.level
curl -XGET http://10.0.0.2:9200/china/city/_search?pretty -H 'Content-Type: application/json' -d'
{
    "size" : 0,
    "aggs" : {
        "city_level" : {
            "terms" : {
              "field" : "level.level"
            }
        }
    }
}'
```

- 响应如下：

```
{
  "took" : 10,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 7,
    "max_score" : 0.0,
    "hits" : [ ]
  },
  "aggregations" : {
    "city_level" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : 1,
          "doc_count" : 4
        },
        {
          "key" : 2,
          "doc_count" : 3
        }
      ]
```

```
    }
  }
}
```

删除文档

删除单个文档

- 输入命令行：

  ```
  curl -XDELETE 'http://10.0.0.2:9200/china/city/beijing?pretty' -H 'Content-Type: application/json'
  ```

- 响应如下：

  ```
  {
    "found" : true,
    "_index" : "china",
    "_type" : "city",
    "_id" : "beijing",
    "_version" : 5,
    "result" : "deleted",
    "_shards" : {
      "total" : 2,
      "successful" : 2,
      "failed" : 0
    }
  }
  ```

删除类型

```
curl -XDELETE 'http://10.0.0.2:9200/china/city?pretty' -H 'Content-Type: application/json'
```

删除索引

```
curl -XDELETE 'http://10.0.0.2:9200/china?pretty' -H 'Content-Type: application/json'
```

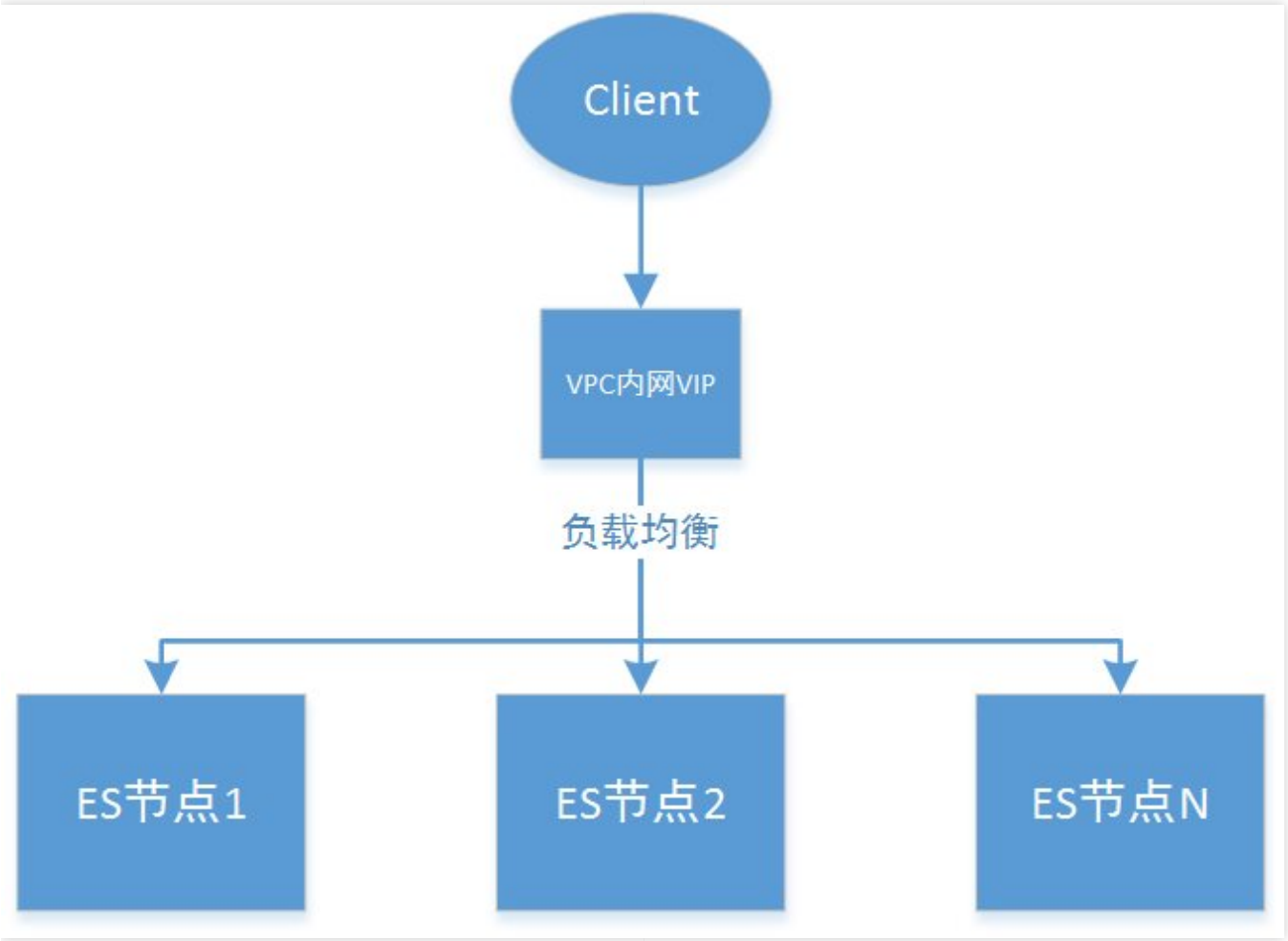
通过客户端访问集群

官方和社区推出了各个语言版本的 SDK 方便用户使用。

- Elasticsearch 提供了各种开发语言的客户端，例如 Java、Python 等，以满足不同开发者的需要。详情请参见 Elasticsearch Client。
- Elasticsearch 从5.6.0版本开始，发布了官方的 Java High Level REST Client，可用于执行 search、index、delete、update and bulk 操作，同 TransportClient 使用的核心 Java classes 一致，并且是为了替代 TransportClient。详情请参见 Java High Level REST Client 说明。

> 版本兼容性上，建议客户端版本同服务端版本保持一致，详情请参见 兼容性说明，TCloudFinanceZone ES 目前提供了多个可选的 Elasticsearch 版本，请注意选择客户端版本。

ES 在用户的 VPC 下提供了一个访问 ES 集群的 VIP，通过负载均衡的方式挂载了 ES 集群内的所有节点。这样做的目的，一方面是为了适配集群的弹性伸缩并保证其高可用性，当 ES 集群的节点发生变化时，VIP 会自动更新节点信息；另一方面，简化用户的操作，用户不用再关注集群节点 IP、port 等信息的变更。



> 不推荐通过 TransportClient 直接连接 Client 和 ES 节点。

ES 的 SDK 支持在创建连接时只配置一个节点的地址，并通过“节点嗅探”功能嗅探出所有的节点来连接使用。这种使用方式有悖于我们推出 VIP 功能的初衷，增加了使用 ES 集群的复杂度。下面我们会简单介绍各个编程语言 SDK 包的使用方法，并说明如何关闭“节点嗅探”功能。

Java 客户端

ES 官方推荐使用 Java REST 客户端连接集群并进行数据操作。Java REST Client 有 Low Level 和 High Level 两种：

- Java Low Level REST Client：使用该客户端需要将 HTTP 请求的 body 手动拼成 JSON 格式，HTTP 响应也必须将返回的 JSON 数据手动封装成对象。
- Java High Level REST Client：该客户端基于 Low Level 客户端实现，提供 API 解决 Low Level 客户端需要手动转换数据格式的问题。

使用 Java High Level REST Client 访问集群，示例步骤与代码如下：

添加 Maven 依赖

> 建议 Java REST 客户端 API 版本，同 ES 集群服务端版本保持一致，TCloudFinanceZone目前提供了不同版本的 ES，请注意选择相应版本的客户端。更多客户端 API 信息，可参考官方 ES API 版本兼容性说明。

```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
    <version>6.4.3</version>
</dependency>
```

> - Client 版本需要与 ES 集群版本保持一致，否则可能会出现兼容性问题。此处的 Demo 适用于 TCloudFinanceZone ES 6.4.3版本，其他版本的使用方法请参考 文档。
> - Java High Level Client 构建于 Java Low Level Client 之上，两种 Client 都是基于 HTTP 协议连接 ES 集群，Java High Level Client 可提供的 API 种类随着版本升级会越来越多，如果当前版本的 Java High Level Client 提供的 API 不满足需求，可以通过升级 ES 集群版本和 Client 版本解决。
> - 使用 TCP 协议连接 ES 集群的 Transport Client 官方已经不再维护，建议使用 HTTP 协议连接集群的 Java High Level Client 或者 Java Low Level Client 。详请参考 elastic 官方文档 由 TransportClient 迁移至 Java High Level Client。

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.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;
import org.elasticsearch.ElasticsearchException;
import org.elasticsearch.action.DocWriteResponse;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.index.IndexResponse;
import org.elasticsearch.action.get.GetRequest;
import org.elasticsearch.action.get.GetResponse;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.rest.RestStatus;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

public class test_es_sdk {
    private static Logger logger = Logger.getLogger(test_es_sdk.class);

    public  static void main(String[]args){
        BasicConfigurator.configure();
        // 设置验证信息，填写账号及密码
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY,
                new UsernamePasswordCredentials("user", "passwd"));
        // 初始化 RestClient, hostName 和 port 填写集群的内网 VIP 地址与端口
        RestClientBuilder builder = RestClient.builder(new HttpHost("xx.xx.xx.xx", 9200, "http"));
        // 设置认证信息
        builder.setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {
                    @Override
                    public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder httpClientBuild
er) {
                        return httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
                    }
                });
        // 设置超时时间
        builder.setMaxRetryTimeoutMillis(10000);
        // 由Low Level Client构造High Level Client
        RestHighLevelClient client = new RestHighLevelClient(builder);

        // 索引文档
        Map<String, Object> jsonMap = new HashMap<String, Object>();
        jsonMap.put("user", "bellen");
        jsonMap.put("name", new Date());
```

```
        jsonMap.put("message", "trying out Elasticsearch");
        IndexRequest indexRequest = new IndexRequest("posts", "doc", "1")
                .source(jsonMap);

        try {
            // 获取响应结果
            IndexResponse indexResponse = client.index(indexRequest);
            String index = indexResponse.getIndex();
            String type = indexResponse.getType();
            String id = indexResponse.getId();
            long version = indexResponse.getVersion();

            if (indexResponse.getResult() == DocWriteResponse.Result.CREATED) {
                logger.info("doc indexed, index: "+ index +", type:"+ type +",id:"+ id+",version:"+version);
            } else if (indexResponse.getResult() == DocWriteResponse.Result.UPDATED) {
                logger.info("doc updated, index: "+ index +", type:"+ type +",id:"+ id+",version:"+version);
            }
        }catch(ElasticsearchException e) {
            if (e.status() == RestStatus.CONFLICT) {
                logger.error("version conflict");

            }
        }catch(Exception e){
            logger.error("execute index api failed, "+ e.toString());
        }

        // 查询文档
        GetRequest getRequest = new GetRequest(
                "posts",
                "doc",
                "1");
        try {
            // 获取响应结果
            GetResponse getResponse = client.get(getRequest);
            String index = getResponse.getIndex();
            String type = getResponse.getType();
            String id = getResponse.getId();
            if (getResponse.isExists()) {
                long version = getResponse.getVersion();
                String sourceAsString = getResponse.getSourceAsString();
                logger.info("get doc, index: "+ index +", type:"+ type +",id:"+ id+",version:"+version +", sour
ce:"+ sourceAsString);
            }
        }catch (ElasticsearchException e) {
            if (e.status() == RestStatus.NOT_FOUND) {
                logger.warn("doc not found");
            }
```

```
        }
        catch(Exception e){
            logger.error("execute get api failed, "+ e.toString());
        }

        // 关闭客户端
        try {
            client.close();
        }catch (Exception e){
            logger.error("close rest client exception:"+ e.toString());
        }
    }
}
```

Python 客户端

pip 安装

```
pip install elasticsearch
```

示例代码

在 Elasticsearch 函数的参数中设置如下3个参数关闭节点嗅探：

- sniff_on_start=False
- sniff_on_connection_fail=False
- sniffer_timeout=None

```
from elasticsearch import Elasticsearch

es = Elasticsearch(["http://xx.xx.xx.xx:9200"],
     http_auth=('user', 'passwd'),
        sniff_on_start=False,
        sniff_on_connection_fail=False,
        sniffer_timeout=None)

res = es.index(index="my_index", doc_type="my_type", id=1, body={"title": "One", "tags": ["ruby"]})
print(res)
res = es.get(index="my_index", doc_type="my_type", id=1)
print(res['_source'])
```

PHP 客户端

为了避免节点的嗅探，您可以设置显示的连接池类为 StaticConnectionPool ，不要使用节点嗅探连接池。

示例代码

```
$client = ClientBuilder::create()
    ->setConnectionPool('\Elasticsearch\ConnectionPool\StaticConnectionPool', ["http://user:passwd@x
x.xx.xx.xx:9200"])
    ->build();
```

Go 客户端

gopkg.in/olivere/elastic 是在 Go 语言中广泛使用的一个由社区贡献的 SDK。此处的 Demo 适用于 TCloudFinanceZone ES 6.4.3版本，其他版本的使用方法请参考 文档。

安装 elastic

```
go get github.com/olivere/elastic
```

示例代码

在 NewClient 函数的参数中设置 elastic.SetSniff(false) 关闭节点嗅探，设置 elastic.SetHealthcheck(false) 关闭对节点的健康检查。

```
import (
    "context"
    "fmt"

    "github.com/olivere/elastic"
)

func main() {
    client, err := elastic.NewClient(elastic.SetURL("http://user:passwd@xx.xx.xx.xx:9200"),
        elastic.SetSniff(false),elastic.SetHealthcheck(false))
    if err != nil {
        panic(err)
    }
    exists, err := client.IndexExists("twitter").Do(context.Background())
    if err != nil {
        panic(err)
    }
```

```
    fmt.Println(exists)
}
```

通过Kibana访问集群

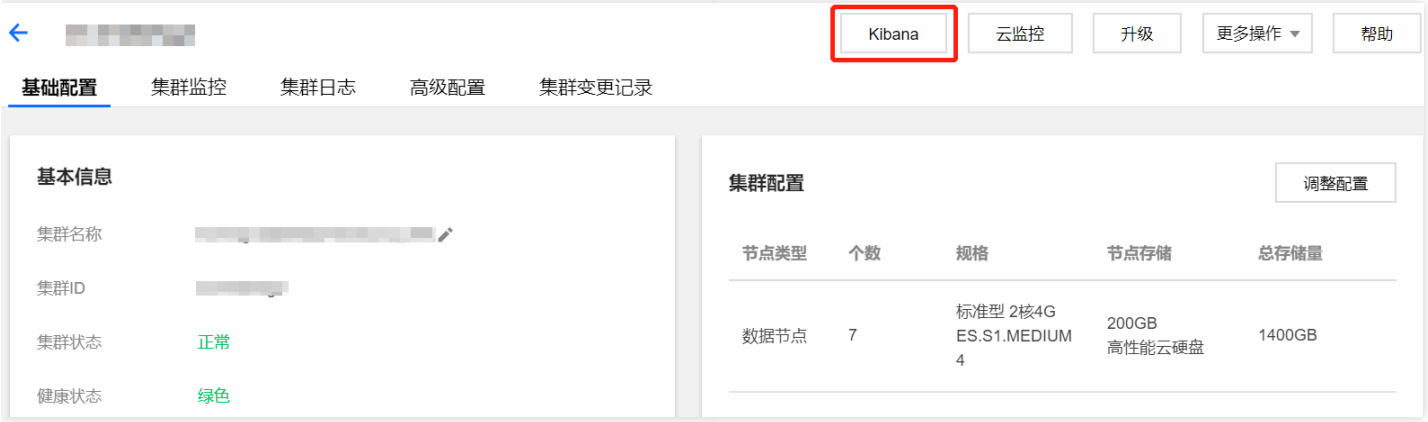
ES 包含 Kibana 模块，用户可以访问集群对应的 Kibana 页面，进行数据的可视化查询分析，以及数据的管理操作，您可以通过本教程开始 Kibana 的快速入门。

访问 Kibana 页面

入口

Kibana 页面有两个入口，分别位于集群列表页和集群详情页，详情如下图。单击对应的入口，会跳转到 Kibana 登录页面。

> 默认情况下，Kibana 通过公网地址访问， 如果您担心通过公网访问 Kibana 会造成安全问题，我们也支持在集群详情页关闭 Kibana 公网地址，开启 Kibana 内网地址来进行访问。

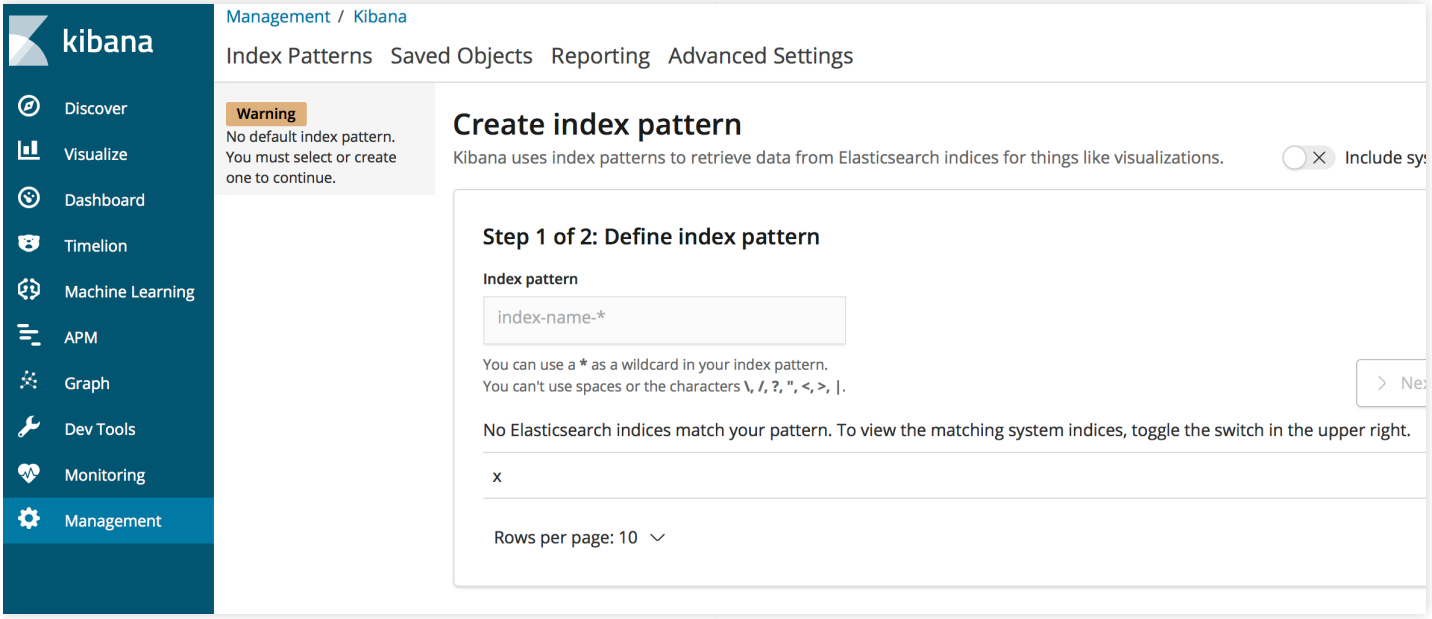


登录

Kibana 页面访问需要登录，帐号为 elastic，密码为用户创建集群时设置的 Kibana 密码。如果忘记密码，可以在集群详情页重置密码。
Kibana 登录页如下图所示：

访问

登录 Kibana 页面后，如果用户是第一次使用，集群尚未存入用户自定义的索引数据，页面会提示用户配置索引，具体参考 索引添加及访问。



索引添加及访问（存储数据）

在 Kibana 页面左侧菜单，单击【Dev Tools】进入开发工具页面，用户可以通过控制台，向集群发送各种操作请求。下面将通过城市信息的数据存储操作的示例，演示如何通过控制台操作集群和存储数据。

添加索引

定义索引的 mapping

索引名称为 china，类型名称为 city，以及详细的字段及类型信息。其中字段 location 的类型是 geo_point，可以表示地理位置信息；level 是对象类型，包含二级字段信息。关于字段类型说明，可查看官方文档 Field Datatypes。

```
PUT china
{
  "mappings": {
    "city": {
      "properties":{
        "name":{ "type": "keyword"  },
        "province":{ "type": "keyword"  },
        "location": {"type": "geo_point"},
        "x":{ "type": "integer" },
        "level":{
            "properties":{
                "level":{ "type": "integer" },
                "range":{ "type": "integer" },
                "name":{ "type": "keyword" }
            }
        },
        "y":{ "type": "integer" },
        "cityNo":{ "type": "integer" }
      }
    }
  }
}
```

添加单个文档

PUT china/city/wuhan
{"name":"武汉市","province":"湖北省江岸区沿江大道188号","location":{"lat":30.5952548577,"lon":114.2999398195},"x":6384,"level":{"level":2,"range":19,"name":"新一线城市"},"y":4231,"cityNo":7}

查询单个文档

GET /china/city/wuhan

添加多个文档

POST _bulk
{ "index" : { "_index": "china", "_type" : "city", "_id" : "beijing" } }
{"name":"北京市","province":"北京市","location":{"lat":39.9031324643,"lon":116.4010433787},"x":6763,"level":{"range":4,"level":1,"name":"一线城市"},"y":6381,"cityNo":1}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "shanghai" } }
{"name":"上海市","province":"上海市","location":{"lat":31.2319526784,"lon":121.469443249},"x":7779,"level":{"range":4,"level":1,"name":"一线城市"},"y":4409,"cityNo":2}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "guangzhou" } }
{"name":"广州市","province":"广东省越秀区吉祥路79号","location":{"lat":23.1317146641,"lon":113.2595185241},"x":6173,"level":{"range":4,"level":1,"name":"一线城市"},"y":2560,"cityNo":3}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "shenzhen" } }
{"name":"深圳市","province":"广东省福田区新园路37号","location":{"lat":22.5455465546,"lon":114.0527779134},"x":6336,"level":{"range":4,"level":1,"name":"一线城市"},"y":2429,"cityNo":4}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "chengdu" } }
{"name":"成都市","province":"四川省锦江区红星路4段-88号-附1号","location":{"lat":30.6522796787,"lon":104.0725574128},"x":4387,"level":{"level":2,"range":19,"name":"新一线城市"},"y":4304,"cityNo":5}
{ "index" : { "_index": "china", "_type" : "city", "_id" : "hangzhou" } }
{"name":"杭州市","province":"浙江省拱墅区环城北路316号","location":{"lat":30.2753694112,"lon":120.1509063337},"x":7530,"level":{"level":2,"range":19,"name":"新一线城市"},"y":4182,"cityNo":6}

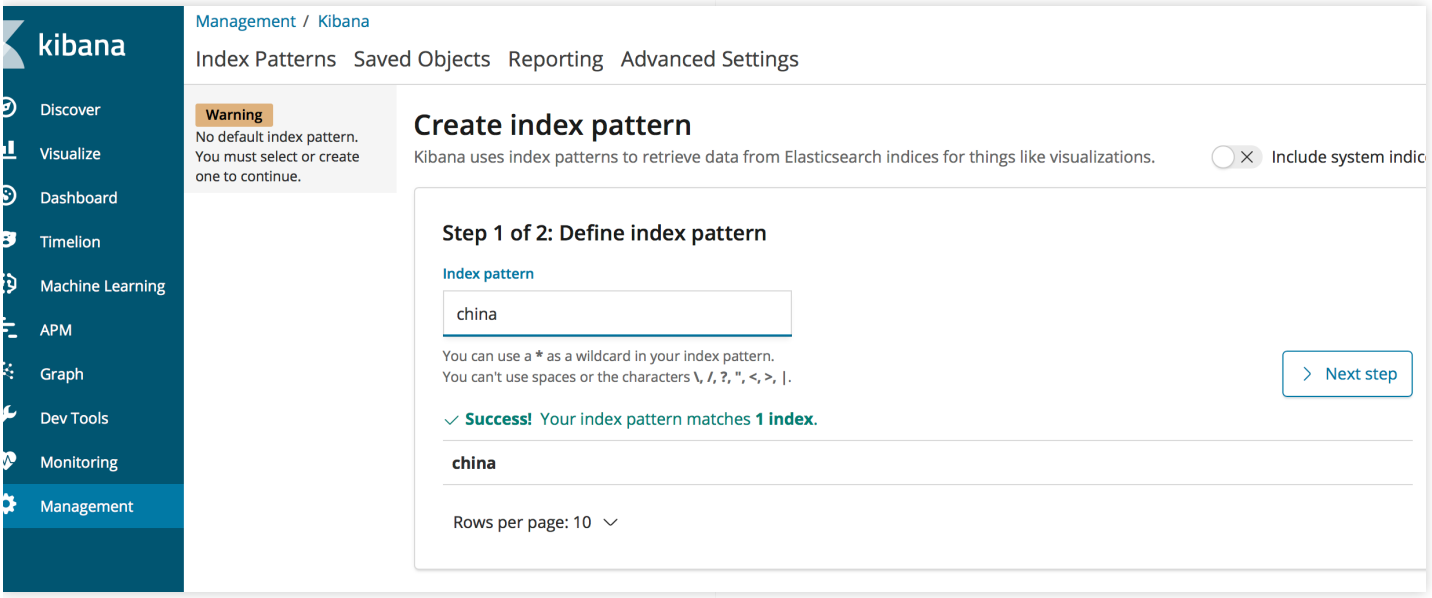
查询多个文档：

```
GET /china/city/_search
```

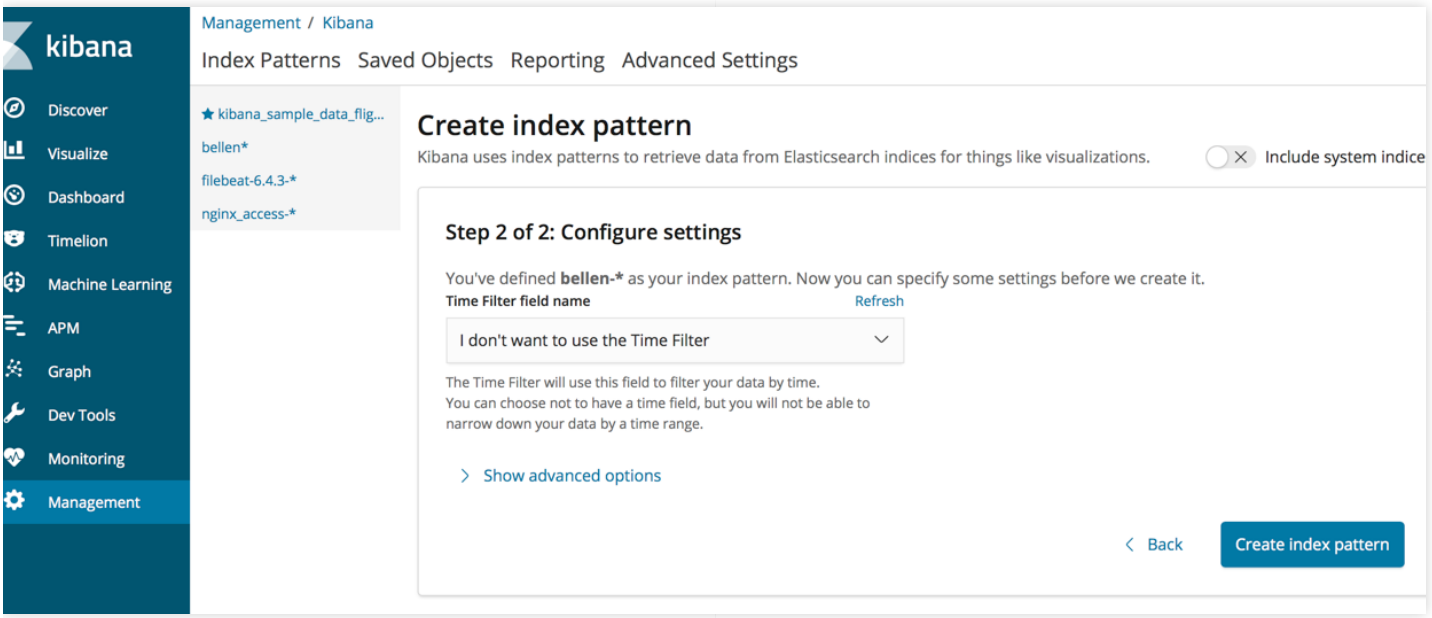
访问索引

配置 Kibana 访问索引

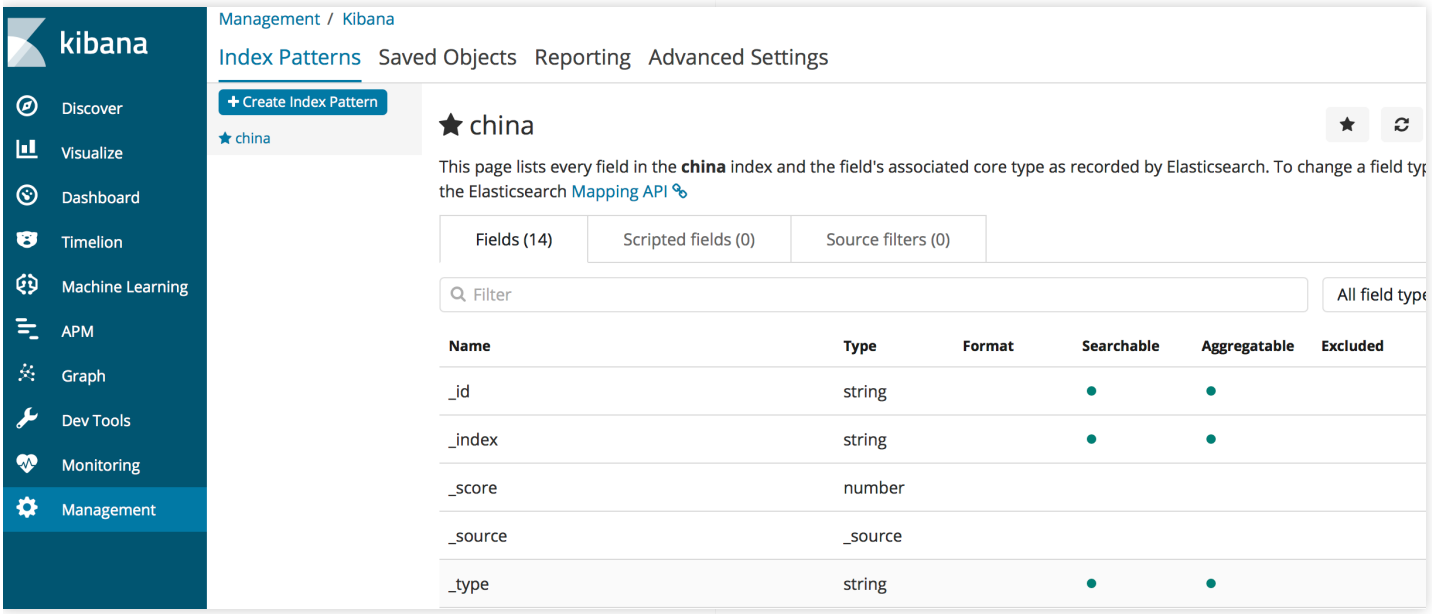
为了使用 Kibana，需要配置至少一个可以匹配到的索引。输入上文创建的索引 china，单击【Next step】进入下一步：



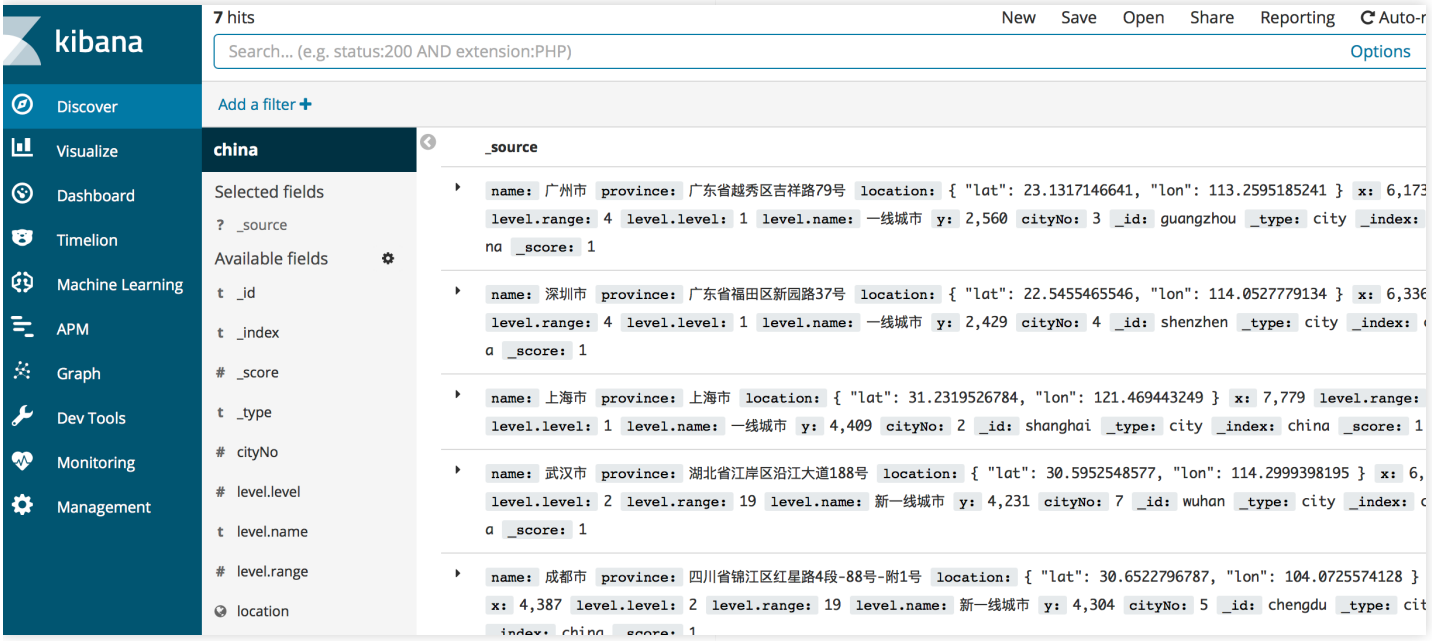
配置时间过滤字段用于通过时间过滤索引中的数据，如果索引中没有表示时间的字段，可以选择不使用时间过滤功能。单击【Create index pattern】创建索引模式：



查看索引对应的字段：

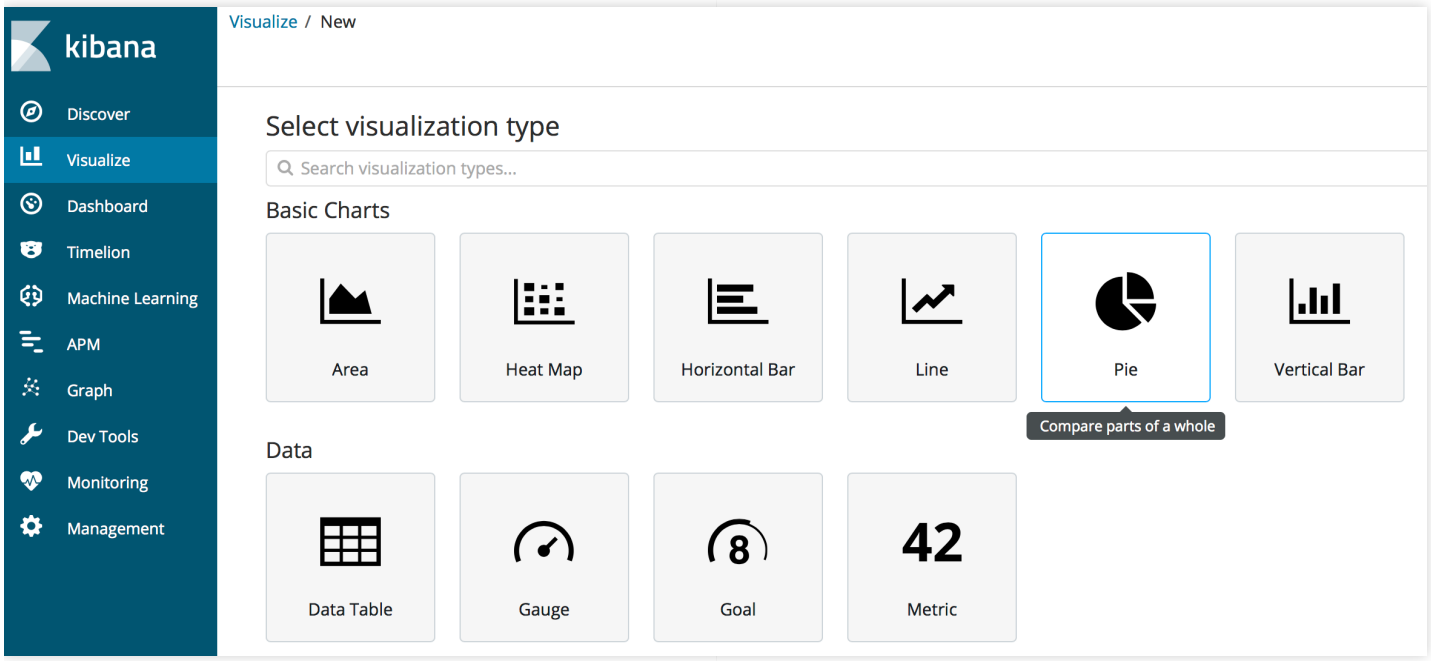


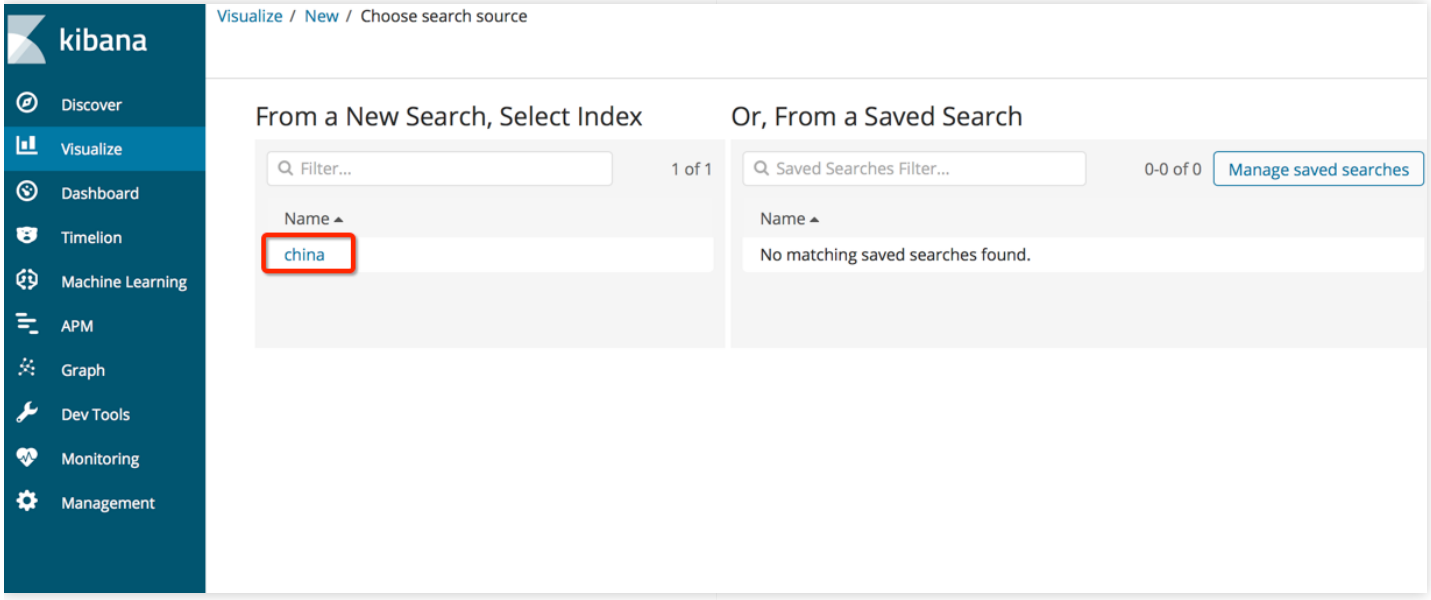
单击左侧菜单【Discover】，查看该索引下已经添加的文档：



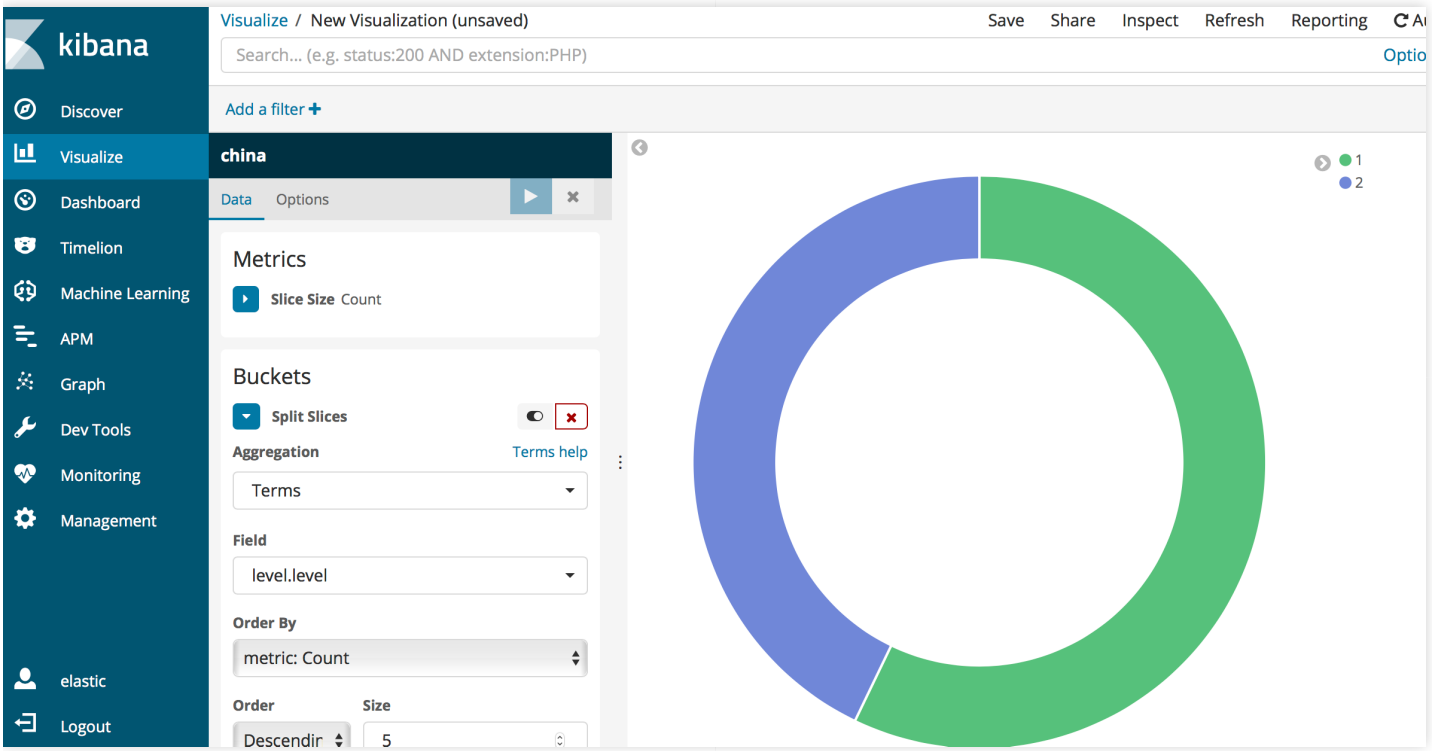
可视化查询分析

Kibana 拥有可视化统计分析数据的能力，单击左侧菜单【Visualize】，可以配置各种可视化的图表进行数据的分析。例如：要统计上文中，china 索引下的不同等级。





配置指标是 count，按字段 level.level 进行分组聚合统计，单击【Save】保存。



Kibana 其他使用方式，可查阅 Kibana 官方文档。

操作指南
访问控制
CAM访问控制配置

ES CAM 简介

访问管理（Cloud Access Management，CAM）是TCloudFinanceZone提供的 Web 服务，主要用于帮助用户安全管理TCloudFinanceZone账户下的资源的访问权限。用户可以通过 CAM 创建、管理和销毁用户(组)，并使用身份管理和策略管理控制其他用户使用TCloudFinanceZone资源的权限，CAM 策略的详细信息及使用请参见 CAM 策略。

ES CAM 策略

通用权限策略

ES 默认提供了两种通用策略：

- 全读写策略 QcloudElasticsearchServiceFullAccess，可以让用户拥有创建和管理 ES 所有集群实例的权限。
- 只读策略 QcloudElasticsearchServiceReadOnlyAccess， 可以让用户拥有查看 ES 集群实例的权限，但是不具有创建、更新、删除等操作的权限。
  您可以登录 策略管理界面，在【服务类型】中选择【Elasticsearch Service】，在列表中会显示默认策略，可对需要授权的账号进行绑定。

如果默认策略不能满足需求，可以单击【新建自定义策略】自定义授权。

自定义权限策略

ES 可授权的资源类型如下：

资源类型	资源描述
instance	qcs::es:$region:$account:instance/*

各 API 支持资源级权限访问控制详情如下：

接口名	描述	是否关联资源	资源描述
获取集群列表、单个集群信息	DescribeInstances	是	qcs::es:${Region}:uin/${ownerUin}:instance/${instanceId}

接口名	描述	是否关联资源	资源描述
创建集群	CreateInstance	否	*
更新集群	UpdateInstance	是	qcs::es:${Region}:uin/${ownerUin}:instance/${instanceId}
重启集群	RestartInstance	是	qcs::es:${Region}:uin/${ownerUin}:instance/${instanceId}
删除集群	DeleteInstance	是	qcs::es:${Region}:uin/${ownerUin}:instance/${instanceId}

支持区域如下（举例）：

区域名称	区域 ID
广州	ap-guangzhou

自定义策略的语法如下：

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "Action"
            ],
            "resource": "Resource",
            "effect": "Effect"
        }
    ]
}
```

- Action：替换成要允许或拒绝的操作。
- Resource：替换成要授权的具体资源。
- Effect：替换成允许或拒绝。

  ES 目前支持除了 DescribeInstances 接口之外的其他操作接口来访问控制管理，您可以将账号下某个集群的更新、重启、删除等操作赋予某个子账号进行管理。

自定义权限示例

授予某账号指定集群更新操作权限，策略语法如下：

```
{
```

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "es:Describe*"
            ],
            "resource": [
                "qcs::es:ap-guangzhou:uin/$uin:instance/$instanceID"
            ],
            "effect": "allow"
        },
        {
            "action": [
                "vpc:Describe*",
                "vpc:Inquiry*",
                "vpc:Get*"
            ],
            "resource": "*",
            "effect": "allow"
        },
        {
            "action": [
                "monitor:*",
                "cam:ListUsersForGroup",
                "cam:ListGroups",
                "cam:GetGroup"
            ],
            "resource": "*",
            "effect": "allow"
        },
        {
            "action": [
                "es:Update*"
            ],
            "resource": [
                "qcs::es:ap-guangzhou:uin/$uin:instance/$instanceID"
            ],
            "effect": "allow"
        }
    ]
}
```

集群安全管理

ES 集群部署在逻辑隔离的私有网络 VPC 中，客户可以完全掌控自己的环境配置，自定义网络访问控制列表（Access Control List）和安全组。在此基础上，我们还提供了更丰富的能力来切实保证云上资源的安全性，包括：

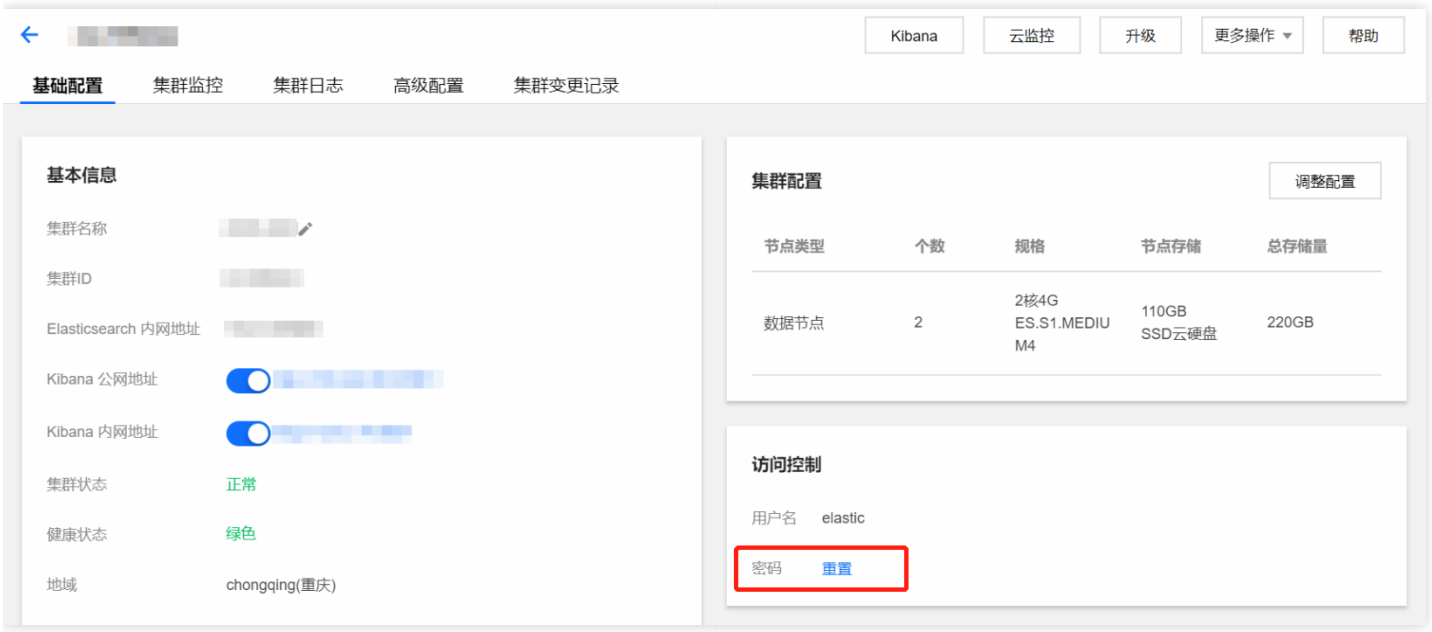
- 对TCloudFinanceZone帐户下资源的 CAM 访问管理（参见 CAM 访问控制配置）
- Kibana 访问 ES 集群密码
- 设置 Kibana 外网访问 IP 黑白名单，或限制 Kibana 仅能通过内网访问

设置 ES 集群访问密码

在创建ES 集群时，会要求用户设置默认用户 elastic 的密码，该帐号和密码用于 Kibana 页面登录。

重置 ES 集群访问密码

用户需要调整 ES 集群访问密码时，可以通过集群详情页的密码重置功能对 ES 集群 elastic 账号的密码进行重置。



设置 Kibana 外网访问 IP 黑白名单

由于 Kibana 页面是通过公网访问，在进行密码校验的基础上，ES 还为 Kibana 访问提供了 IP 黑白名单功能，进一步保障用户集群的访问安全性。

- 配置规则：支持多个 IP，IP 之间以英文逗号分隔，格式为 192.168.0.1 、 192.168.0.0/24 ，最多支持10个。
- 黑白名单设置：客户可以设置任意一个，如果黑白名单都配置，以白名单为准。

限制 Kibana 仅能通过内网访问

如果用户担心公网访问安全性，也可以关闭外网访问，设置仅允许内网访问。

管理集群
集群状态

在集群列表页和集群详情页的基本信息中，均可查看到集群的状态信息。

集群状态是反映集群是否在变更中或正常使用的状态，包括正常、处理中等，具体含义如下：

状态	含义
正常	集群创建完成，且没有配置变更、集群重启等动作，可以正常访问和使用的状态。
处理中	集群创建、集群配置变更、集群重启等操作，需要一定的时间进行处理的状态，期间部分服务访问会受到影响，如 Kibana、数据存储和查询等。

其中健康状态是 ES 集群众多监控信息中非常重要的一个，用来表征集群总体上是否工作正常。健康状态种类如下：

颜色	健康状态
绿色（green）	所有的主分片和副本分片都正常运行。
黄色（yellow）	所有的主分片都正常运行，但不是所有的副本分片都正常运行。
红色（red）	有主分片没有正常运行。

详情可查看 ES 集群健康状态。

高级配置

本部分提供了集群部分高级特性的配置管理功能，包括 IK 中文分词插件自定义词典， elasticsearch.yml 参数修改。
在集群列表页，单击集群 ID 进入集群详情页，单击【高级配置】，即可进入集群高级配置管理页面。

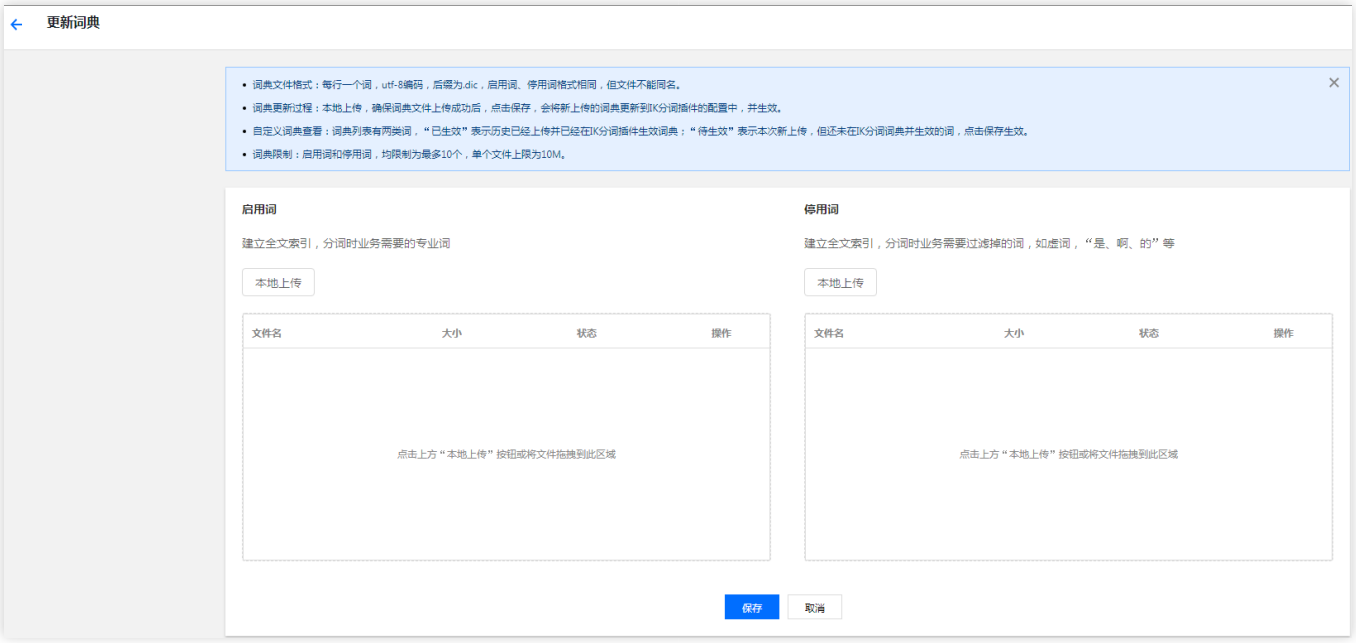
插件配置

IK 中文分词插件

在插件配置部分，可以查看到集群已经预装了的 IK 中文分词插件。
关于 IK 中文分词插件的介绍，详情可查看 IK Analysis for Elasticsearch ，您可以利用该插件对存到 ES 集群数据中的业务的中文关键词建立索引，实现搜索功能。

操作步骤：

1. 单击【更新词典】，进入更新词典页面，有启用词和停用词两项。
2. 单击【本地上传】，选择您需要更新的词典文件。
3. 单击【保存】，即可热更新词典（不需要重启集群）。



关于词典文件的要求及说明：

- 词典类型：有两类词，“启用词”和“停用词”。“启用词”词典中的词是用户在向 ES 集群存入数据，建立索引的时候，指定 IK 作为分词工具。如果存入的数据中时有这类词，就会建立索引，并能通过关键词查询搜索到。“停用词”则会刻意回避不建立索引的词。
- 限制要求：对于词典文件，也有一些限制和要求，需要一行一个词，utf-8 编码。为了避免混淆，启用词和停用词文件名不能重复。另外，因为词典文件会加载到内存中，所以对文件的大小和个数也做了一定的限制，单个文件最大为10M，上传文件总数最多为10个。
- 更新过程：列表会展示历史已经更新上传的词典。新上传的词典，如果不符合要求，会直接限制上传。上传完成后，词典文件会显示成“待生效”状态。所有需要更新的词典上传完成后，单击【保存】，会保存到用户的集群并生效。如果有上传失败的文件，或不是 utf-8 格式的文件，会提示失败，需要删除失败的文件后，才能单击保存生效。

YML 配置

查看配置项

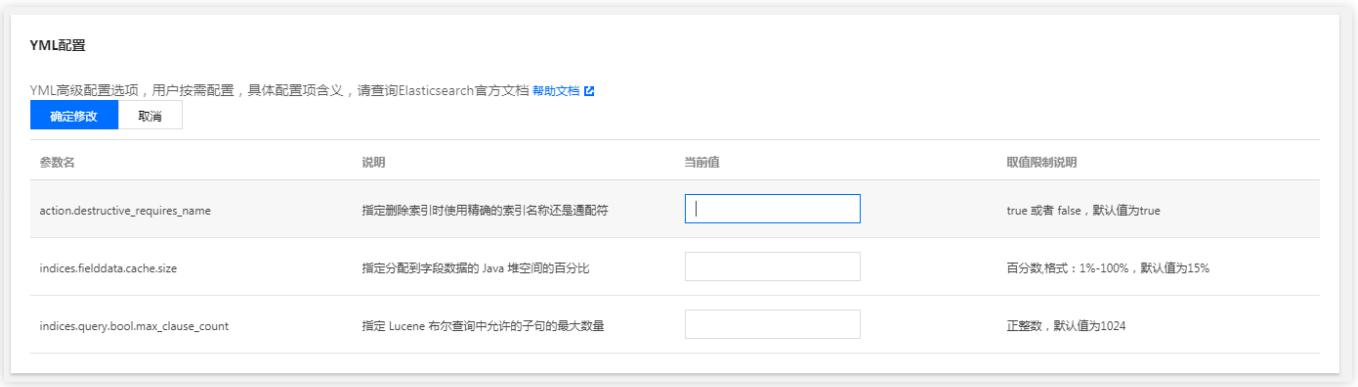
在集群详情中单击【高级配置】中可查看配置项。

配置项	说明	取值限制
action.destructive_requires_name	删除索引时是否需要明确索引名称	true 或者 false，默认值为 true
indices.fielddata.cache.size	指定分配到字段数据的 Java 堆空间的百分比	百分数，格式：1% - 100%，默认值为15%
indices.query.bool.max_clause_count	指定 Lucene 布尔查询中允许的子句的最大数量	正整数，默认值为1024

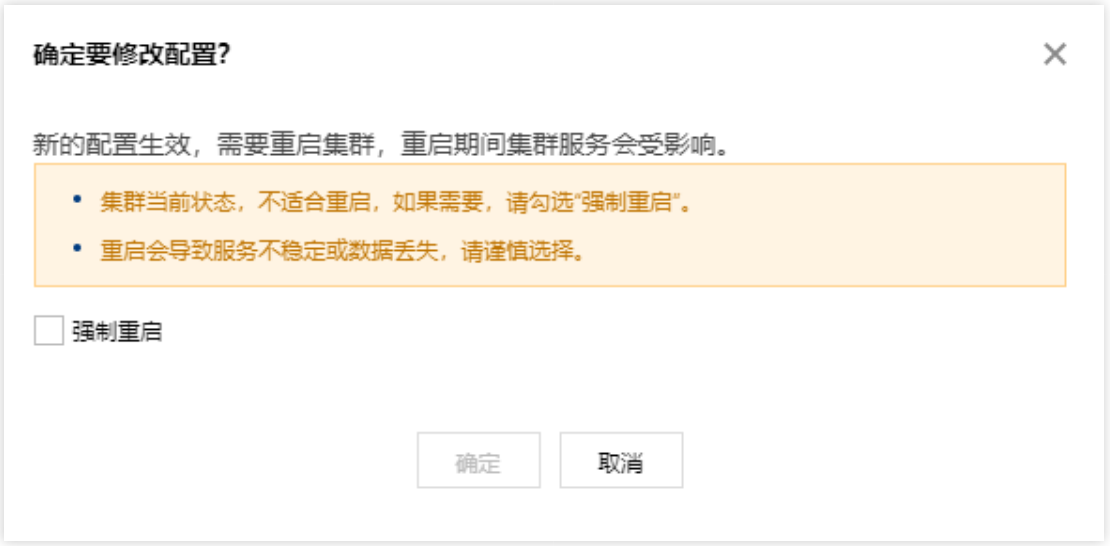
具体配置项详细含义，请查询 Elasticsearch 官方文档。

修改配置项

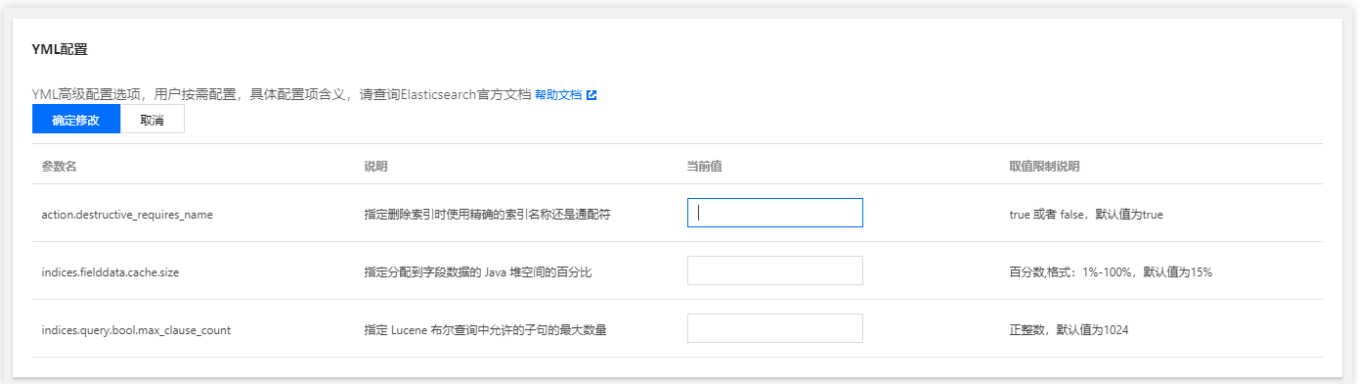
1. 单击【批量修改】，可对相应的配项进行修改，配置项的输入限制，见上文的说明。



- 如果集群健康状态为 YELLOW 或 RED，或集群存在无副本索引时，修改集群配置项对话框会有强制重启的提示和选项框，此时进行更新配置操作有较大风险，建议修复集群状态，为所有索引添加副本，再进行修改配置操作。
- 如果用户已了解该操作风险，仍要进行更新配置操作，可以勾选【强制重启】选项框，单击【确定】进行重启。



2. 单击【批量修改】，可对相应的配项就行修改，配置项的输入限制，见上文的说明。



3. 如果有其他配置项自定义设置需求，请通过提交工单向我们进行反馈。

调整配置

应用场景

在业务发展过程中，集群的数据量和访问量是动态的。随着业务的发展，数据量和访问量都在增大。当集群规模跟实际业务需求不太匹配时，可以动态调整集群的配置，对集群进行扩容；也可以在数据量和访问量不大时，暂时调小集群的规模。ES 支持对节点个数、节点类型和单个数据节点的存储量等进行配置调整。此外，也支持对集群的专用主节点进行调整，具体见以下说明。

缩容目前限制开放，您可以通过工单申请。

操作步骤

1. 登录 ES 控制台，进入集群列表页。
2. 集群【调整配置】有两个入口，分别位于集群列表页和集群详情页。
    - 在集群列表页，选择需要扩容的集群，在【操作】列选择【更多操作】>【调整配置】。
    - 单击集群名称进入集群详情页，在右上角选择【更多操作】>【调整配置】。
3. 在弹出的【调整配置】对话框，根据业务需求对集群节点机型或节点数量进行调整，单击【确定】。
   一次操作仅支持对节点类型（节点机型和存储）或节点数量中的一项进行扩容或缩容调整，不支持同时调整。
4. 配置调整开始后，集群状态为【处理中】，待集群状态变为【正常】，即可正常使用。
5. 可以在集群详情页，查看集群的变更进度。

    - 调整数据节点配置。

调整配置
调整配置，可以扩容，也可以缩容，一次操作只支持对节点类型（节点规格和磁盘容量）或节点个数中的一项进行调整，不支持同时调整。
磁盘使用率或集群负载较高时，建议调大集群配置。
磁盘使用率较低或负载不高时，可以调小配置。如果磁盘使用率较高，想调整配置，需要先进行数据清理。
缩容情况产生的退款，退款金额将 按购买使用的现金和赠送金支付比例 返还到您的 账户，但是，若购买时使用折扣或代金券，折扣和代金券不予退还。
集群
当前配置
ID: es-6f79xjtf
名称: test-opack
地域: 华南地区(广州)
可用区: 广州四区
节点机型: ES.S1.SMALL2（1核2G）
节点存储量: 100GB SSD云硬盘
节点个数: 2
集群总存储量: 200GB
专用主节点: 未配置
节点机型
ES.S1.SMALL2-1核2G
节点数量
3
单节点存储
100
GB
(步长为10GB)
0GB
660GB
1330GB
2000GB
配置专用主节点
原配置费用
0.87元/小时
新配置费用
元/小时
确定
取消

- 调整主节点配置。

调整配置
调整配置，可以扩容，也可以缩容，一次操作只支持对节点类型（节点规格和磁盘容量）或节点个数中的一项进行调整，不支持同时调整。
磁盘使用率或集群负载较高时，建议调大集群配置。
磁盘使用率较低或负载不高时，可以调小配置。如果磁盘使用率较高，想调整配置，需要先进行数据清理。
缩容情况产生的退款，退款金额将 按购买使用的现金和赠送金支付比例 返还到您的 云账户，但是，若购买时使用折扣或代金券，折扣和代金券不予退还。
集群
当前配置
ID: es-6f79xjtf
名称: test-opack
地域: 华南地区(广州)
可用区: 广州四区
节点机型: ES.S1.SMALL2（1核2G）
节点存储量: 100GB SSD云硬盘
节点个数: 2
集群总存储量: 200GB
专用主节点: 未配置
节点机型
ES.S1.SMALL2-1核2G
节点数量
2
单节点存储
100
GB
(步长为10GB)
0GB
660GB
1330GB
2000GB
配置专用主节点
专用主节点机型
ES.S1.MEDIUM4-2核4G
专用主节点个数
3
原配置费用
0.87元/小时
新配置费用
确定
取消

- 查看集群变更进度。

es-0g
Kibana
云监控
更多操作
帮助
基础配置
集群监控
集群日志
高级配置
集群变更记录
您的集群尚未配置云监控告警，为了保障业务稳定运行，建议您配置关键指标告警。系统已预置告警策略模板，只需简单几步即可完成配置，点击前往（查看教程）
2019-05-16 20:04:27 至 2019-05-23 20:04:27
时间
操作
详情
进度
2019-05-23 19:23:05
变配
原：数据节点机型: ES.S1.SMALL2;数据节点磁盘容量: 110
新：数据节点机型: ES.S1.2XLARGE32;数据节点磁盘容量: 400
100%（收起详情）
准备资源
进度 100%
2019-05-23 19:23:34
集群节点变更
进度 100%
2019-05-23 19:27:31
配置负载均衡
进度 100%
2019-05-23 19:31:25

">

重启集群

操作场景

ES 为用户提供了集群重启功能，用户可以根据集群的情况使用该功能对集群进行重启操作。

注意事项

- 重启集群为耗时操作，建议用户在非必要情况下不要进行该操作。
- 重启过程会灰度重启各节点，因此用户应保证集群至少有1副本，否则会导致重启时集群变为 RED 状态，甚至有丢失数据的风险。
- 当集群状态为 YELLOW 或 RED，或集群存在无副本索引等情况下，用户进行重启操作会提示需要进行强制重启。强制重启操作有较高风险，存在丢失数据的可能，在此种情况下建议用户先恢复集群状态为 GREEN，并为所有索引创建至少一个副本，然后再进行重启操作。若当前集群状态无法恢复，用户在了解强制重启风险的前提下仍要进行重启操作，可以勾选强制重启选项进行重启。

操作步骤

1. 登录 ES 控制台。
2. 重启集群有两个入口，分别位于集群列表页和集群详情页。
    - 在集群列表页，选择相应的集群，选择【操作】>【更多】>【重启】。
    - 单击集群名称进入集群详情页，在右上角选择【更多】>【重启】。
3. 正常集群在单击【重启】后出现如下界面，单击【确定】，集群状态变为处理中，等待集群状态恢复为正常重启操作即完成。

4. 当集群状态为 YELLOW 或 RED，或集群存在无副本索引的情况下，界面中会出现强制重启提示，例如下图只有勾选了强制重启才能对集群发起重启操作。在此种情况下用户可参考 注意事项 第三条中的描述进行操作。重启操作发起后，集群状态会变为处理中，等待集群状态恢复为正常，重启操作即完成。

销毁集群

因为业务发展需要，当 ES 集群无法满足您的需求，需要退货时，您可以在控制台对集群进行销毁，避免服务继续运行而产生费用。如果是集群配置无法满足需求，您也可以通过调整集群配置把集群调整到合适的规格，操作方法可查看 调整集群配置。
集群被销毁后，数据无法恢复，请谨慎操作。

操作步骤

1. 登录 ES 控制台。
2. 销毁集群有两个入口，分别位于集群列表页和集群详情页。
    - 在集群列表页，选择相应的集群，在【操作】列选择【更多操作】>【销毁】。
    - 单击集群名称进入集群详情页，在右上角选择【更多操作】>【销毁】
3. 在销毁集群对话框，单击【确定】，系统将清空集群数据，并回收资源，数据清空后，无法恢复。

冷热分离与索引生命周期管理

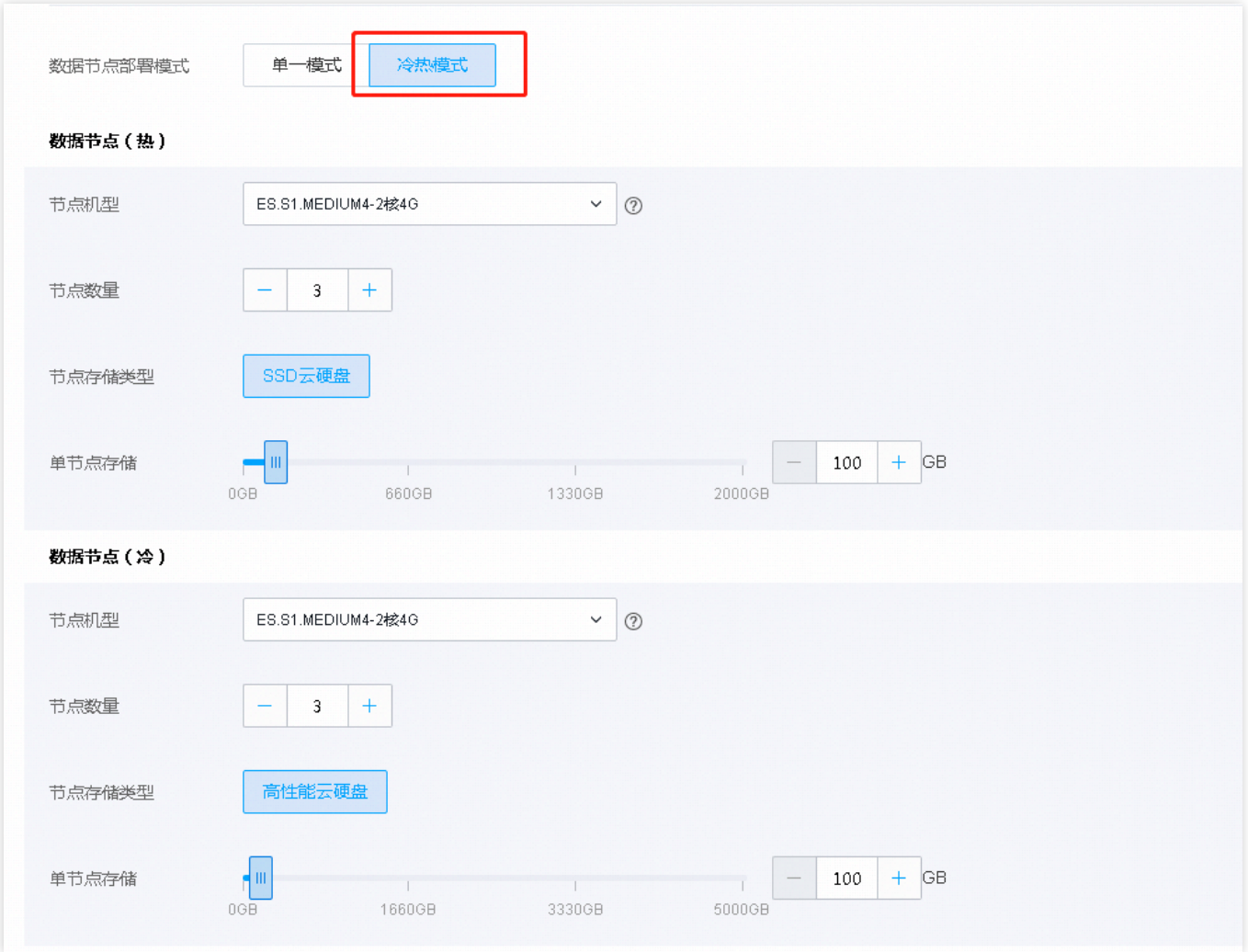
Elasticsearch 主要用于海量数据的存储和检索，若将所有数据都放在 SSD 硬盘上，成本会非常高。可通过冷热分离来解决这个问题，冷热集群可以在一个集群内包含冷、热两种属性的节点，从而兼顾性能和容量之间的矛盾：

- 对读写性能要求比较高的热数据（例如7天内的日志）可以在热节点上以 SSD 磁盘存储。
- 对存储量需求比较大但对读写性能要求较低的索引（例如1个月甚至更长时间的日志）可以在冷节点上以 SATA 磁盘存储。
  ES 提供了快速配置构建冷热集群的能力，用户可以在TCloudFinanceZone官网根据业务需要指定冷热节点规格，快速建立一个冷热分离架构的 ES 集群。

创建冷热集群

在购买集群时直接创建

1. 进入Elasticsearch Service 创建集群 页面，在页面填写所需创建集群的相关信息。
2. 【数据节点部署方式】选择【冷热模式】，并选择冷热节点的规格，



3. 进一步设置集群的其他参数，确认并支付即可。

将现有集群变配为冷热集群

在【集群列表】页面，单击右上角的【更多操作】，在下拉菜单中选择【调整配置】，选择【冷热模式】，根据需要设置冷热节点的规格和相关配置，将现有集群变配为冷热集群。

使用冷热集群

节点角色查看

验证节点冷热属性，命令如下：

```
GET _cat/nodeattrs?v&h=node,attr,value&s=attr:desc

node       attr        value
node1   temperature     hot
node2   temperature     hot
node3   temperature     warm
node4   temperature     hot
node5   temperature     warm
...
```

指定索引冷热属性

业务方可以根据实际情况决定索引的冷热属性。

- 对于热数据，设置索引如下：

```
PUT hot_data_index/_settings
{
  "index.routing.allocation.require.temperature": "hot"
}
```

- 对于冷数据，设置索引如下：

```
PUT hot_data_index/_settings
{
  "index.routing.allocation.require.temperature": "warm"
}
```

验证设置的索引

1. 创建索引。

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

2. 查看分片分配，可以看到分片均匀分配在五个节点上。

```
GET _cat/shards/hot_warm_test_index?v&h=index,shard,prirep,node&s=node
index          shard prirep node
hot_data_index 1     p      node1
hot_data_index 0     r      node1
hot_data_index 2     r      node2
hot_data_index 2     p      node3
hot_data_index 1     r      node4
hot_data_index 0     p      node5
```

3. 设置索引。

- 设置索引为热索引。

```
PUT hot_warm_test_index/_settings
{
      "index.routing.allocation.require.temperature": "hot"
}
```

查看分片分配，分片均分配在热节点上。

```
GET _cat/shards/hot_warm_test_index? v&h=index,shard,prirep,node&s=node
index          shard prirep node
hot_data_index 1     p      node1
hot_data_index 0     r      node1
hot_data_index 0     p      node2
hot_data_index 2     r      node2
hot_data_index 2     p      node4
hot_data_index 1     r      node4
```

- 设置索引为冷索引。

```
PUT hot_warm_test_index/_settings
{
```

```
        "index.routing.allocation.require.temperature": "warm"
    }
```

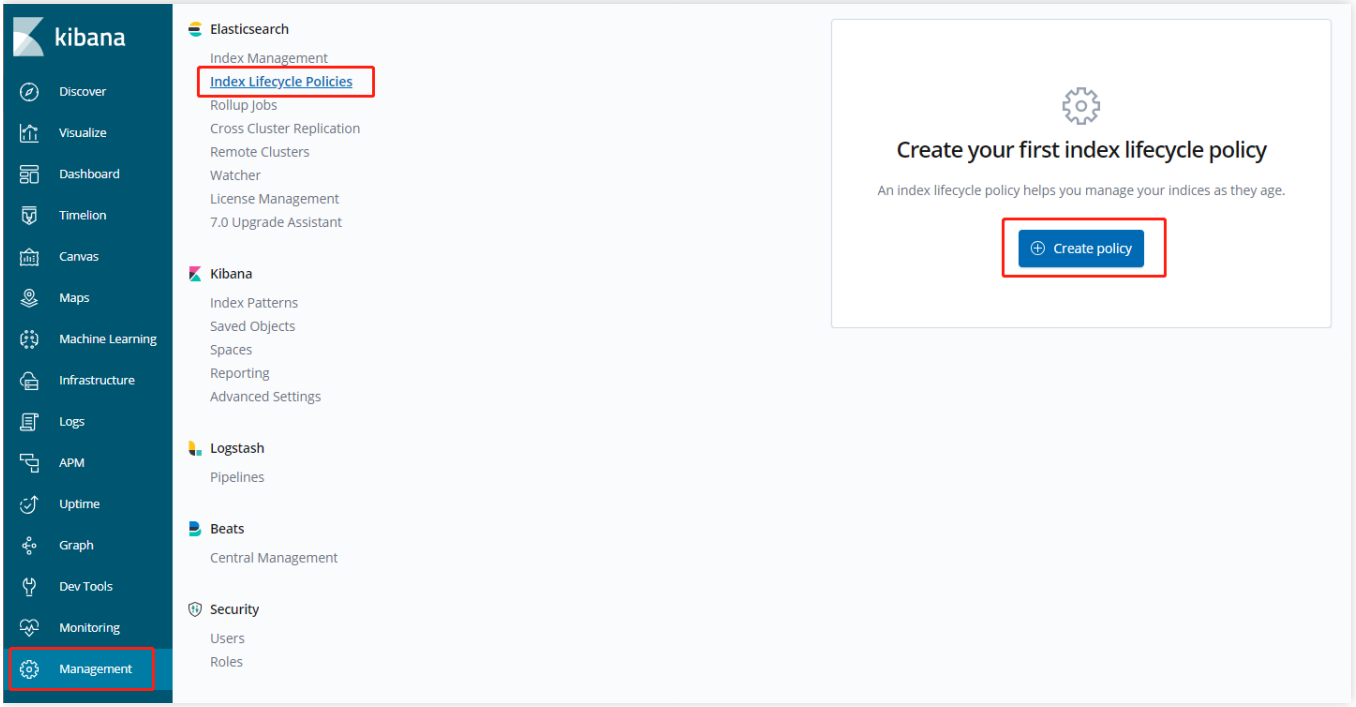
查看分片分配，分片均分配到冷节点上。

```
GET _cat/shards/hot_warm_test_index?v&h=index,shard,prirep,node&s=node
index          shard prirep node
hot_data_index 1     p      node3
hot_data_index 0     r      node3
hot_data_index 2     r      node3
hot_data_index 0     p      node5
hot_data_index 2     p      node5
hot_data_index 1     r      node5
```

索引生命周期管理

目前已提供6.8.2版本的集群，该版本 Elasticsearch(>=6.6) 提供了索引生命周期管理功能。索引生命周期管理可以通过 API 或者 kibana 界面配置，详情可参考 index-lifecycle-management。下文将通过 kibana 界面来演示，如何使用索引生命周期管理，结合冷热分离架构，实现索引数据的动态管理。

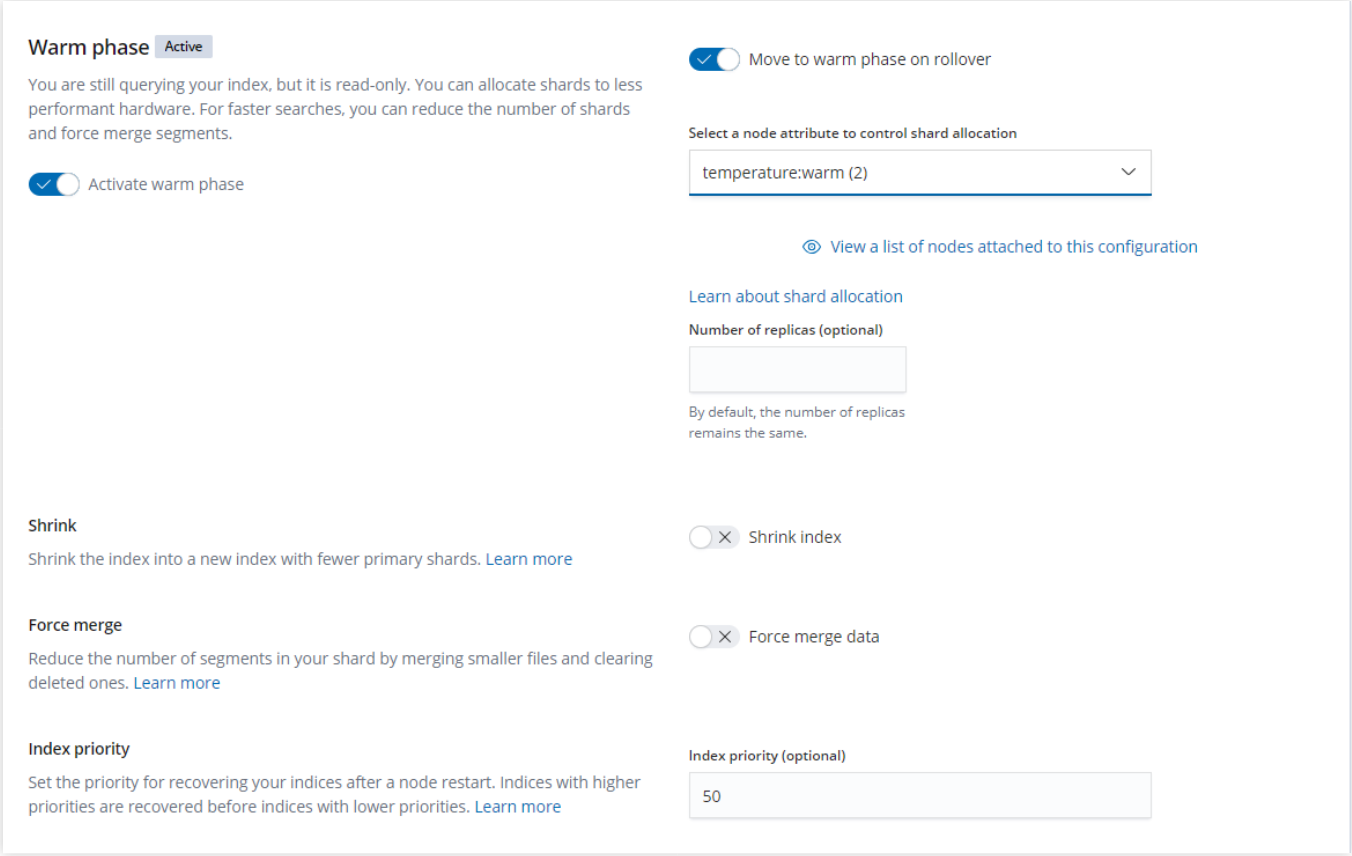
kibana 中的索引生命周期管理位置（版本6.8.2）：



单击【Create policy】进入配置界面。索引的生命周期为 Hot phase 、 Warm phase 、 Cold phase 、 Delete phase 四个阶段。

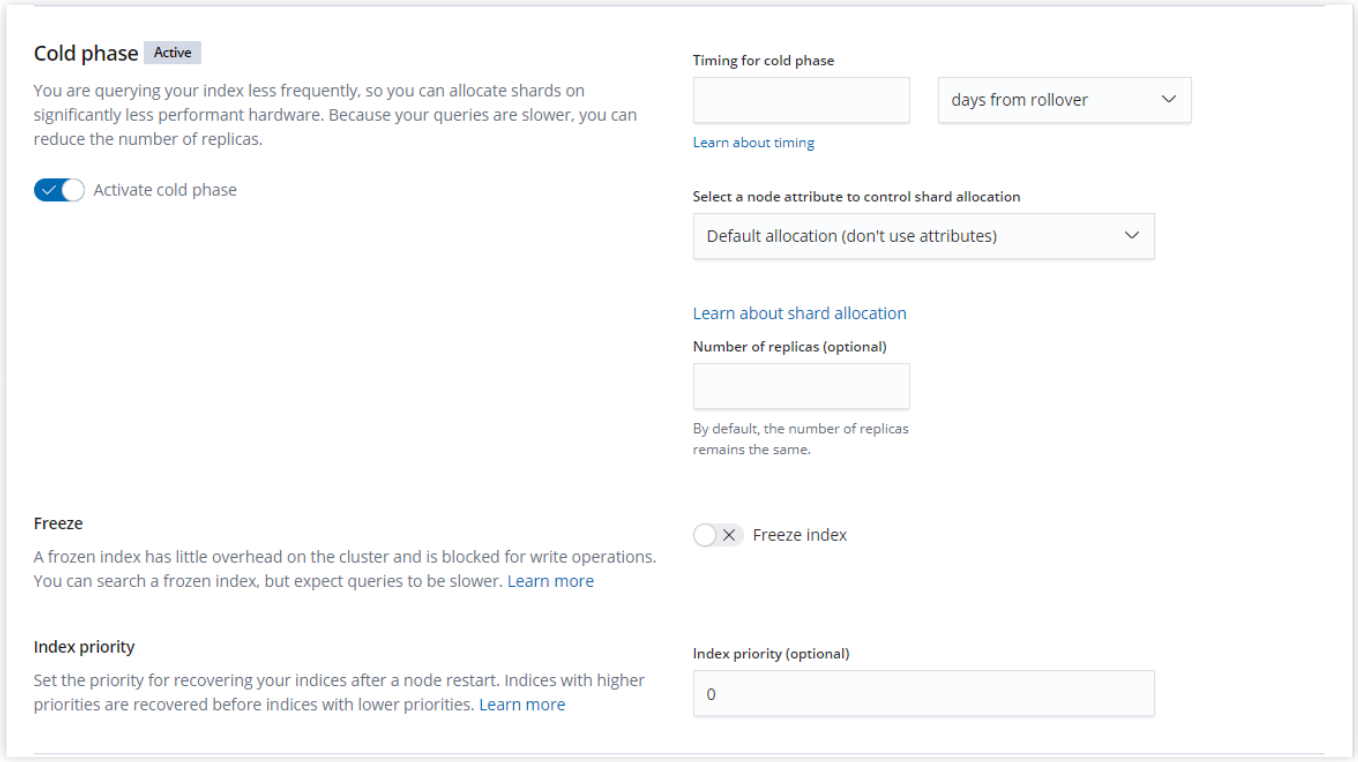
- Hot phase：此阶段可以根据索引的文档数、大小、时长决定是否调用 rollover API 来滚动索引，详情可参考 indices-rollover-index。因与本文关系不大，这里不再赘述。

- Warm phase：当一个索引在 Hot phase 被 rollover 后便会进入 Warm phase ，进入该阶段的索引会被设置为 read-only。用户可为此索引设置要使用的 attribute，例如对于冷热分离策略，这里可选择 temperature: warm 属性。另外还可以对索引进行 forceMerge、shrink 等操作，这两个操作具体可以参考 shrink API 和 force merge 官方文档。



- Cold phase：可以设置当索引 rollover 一段时间后进入 cold 阶段，这个阶段也可以设置一个属性。从冷热分离架构可以看出冷热属性是具备扩展性的，不仅可以指定 hot、warm，也可以扩展增加 hot、warm、cold、freeze 等多个冷热属性。如果想使用三层的冷热分离，可指定为 temperature: cold 。同时还支持对索引的 freeze 操作，详情参考 freeze API 官方文档。

- Delete phase：可以设置索引 rollover 一段时间后进入 delete 阶段，进入该阶段的索引会自动被删除。


Cold phase
Active
You are querying your index less frequently, so you can allocate shards on significantly less performant hardware. Because your queries are slower, you can reduce the number of replicas.
Activate cold phase
Timing for cold phase
days from rollover
Learn about timing
Select a node attribute to control shard allocation
Default allocation (don't use attributes)
Learn about shard allocation
Number of replicas (optional)
By default, the number of replicas remains the same.
Freeze
A frozen index has little overhead on the cluster and is blocked for write operations. You can search a frozen index, but expect queries to be slower. Learn more
Freeze index
Index priority
Set the priority for recovering your indices after a node restart. Indices with higher priorities are recovered before indices with lower priorities. Learn more
Index priority (optional)
0

查询集群日志

介绍 ES 集群运行日志的使用说明。用户可以通过集群的运行日志，了解集群的运行状况，定位问题，辅助集群的应用开发和运维。

查询集群日志

1. 登录 ES 控制台，单击集群名称，进入在集群详情页。
2. 选择【集群日志】页面，可查看集群的运行日志。
   - ES 共有四种类型的日志：主日志、搜索慢日志、索引慢日志和 GC 日志，日志内容包括日志时间、日志级别，以及具体信息等。
   - ES 默认提供集群7天内的运行日志，按时间倒序展示，用户可以按时间和关键字进行查询。此外，用户也可调用 ES API 调整日志相关配置，例如对于慢日志，设定查询或索引数据时，认为响应较慢的时间阈值。
3. 在日志页面的搜索框，可以按照时间范围和关键字查询相关日志，关键字查询语法同 lucene 查询语法一致。
   - 输入关键词查询，例如：“YELLOW”。
   - 指定字段设置关键词，例如：“message:YELLOW”。
     多个条件组合：“level:INFO and ip:10.0.1.1”，可以查询相关日志。

日志说明

主日志

展示集群运行产生日志的时间、级别、信息等，有 INFO、WARN、DEBUG 等不同级别。

2019-2-14 08:00:00 10.0****
INFO

```
[o.e.c.r.a.AllocationService] [1550199698000783811] Cluster health status changed from [YELLOW] to [GREEN] (reas
on: [shards started [[filebeat-2019.02.19][2]] ...]).

2019-2-14 02:30:02 10.0****
DEBUG
[o.e.a.a.i.d.TransportDeleteIndexAction] [1550199698000783811] failed to delete indices [[[filebeat-2019.02.09/7VZ
M6Fa-Twmj8pVaAyyrxg]]]
org.elasticsearch.index.IndexNotFoundException: no such index
    at org.elasticsearch.cluster.metadata.MetaData.getIndexSafe(MetaData.java:475) ~[elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.cluster.metadata.MetaDataDeleteIndexService.lambda$deleteIndices$0(MetaDataDeleteInde
xService.jav
```

慢日志

慢日志的目的是捕获超过指定时间阈值的查询和索引请求，用于追踪分析由用户产生的很慢的请求。

```
2018-10-28 12:04:17
WARN
[index.indexing.slowlog.index] [1540298502000001009] [pmc/wCALr6BfRm-sr3qOQuGX
Xw] took[18.6ms], took_millis[18], type[articles], id[AWa41-J9c0s1mOPvR6F3], routing[] , source[]
```

开启和调整慢日志

默认情况，慢日志不开启。开启慢日志需要定义具体动作（query、fetch 或 index）、期望的事件记录等级（INFO、WARN 、DEBUG 等）、以及时间阈值。用户可以根据业务场景，开启和调整相关配置。

如需开启慢日志，可在集群详情页的右上角单击【Kibana】进入 Kibana 页面，通过 Dev Tools 调用 Elasticsearch 相关 API，或通过客户端调用配置修改 API。

配置所有索引：

```
PUT */_settings
{
"index.indexing.slowlog.threshold.index.debug" : "5ms",
"index.indexing.slowlog.threshold.index.info" : "50ms",
"index.indexing.slowlog.threshold.index.warn" : "100ms",
"index.search.slowlog.threshold.fetch.debug" : "10ms",
"index.search.slowlog.threshold.fetch.info" : "50ms",
"index.search.slowlog.threshold.fetch.warn" : "100ms",
"index.search.slowlog.threshold.query.debug" : "100ms",
"index.search.slowlog.threshold.query.info" : "200ms",
"index.search.slowlog.threshold.query.warn" : "1s"
}
```

配置单个索引：

```
PUT /my_index/_settings
{
"index.indexing.slowlog.threshold.index.debug" : "5ms",
"index.indexing.slowlog.threshold.index.info" : "50ms",
"index.indexing.slowlog.threshold.index.warn" : "100ms",
"index.search.slowlog.threshold.fetch.debug" : "10ms",
```

```
"index.search.slowlog.threshold.fetch.info" : "50ms",
"index.search.slowlog.threshold.fetch.warn" : "100ms",
"index.search.slowlog.threshold.query.debug" : "100ms",
"index.search.slowlog.threshold.query.info" : "200ms",
"index.search.slowlog.threshold.query.warn" : "1s"
}
```

GC 日志

ES 默认开启 GC 日志，以下两条具体的 GC 日志，分别展示日志的时间、节点 IP、日志级别等。

```
2019-2-14 20:48:22
10.0.***
INFO
[o.e.m.j.JvmGcMonitorService] [1550199698000783711] [gc][380573] overhead, spent [307ms] collecting in the last [1s]
2019-2-14 10:04:09
10.0.***
WARN
[o.e.m.j.JvmGcMonitorService] [1550199698000784111] [gc][341943] overhead, spent [561ms] collecting in the last [1s]
```

监控与告警
查看监控

操作场景

ES 对运行中的 ES 集群，提供了多项监控指标，用以监测集群的运行情况，如存储、IO、CPU、内存使用率等。您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 ES 控制台查看集群监控的操作。

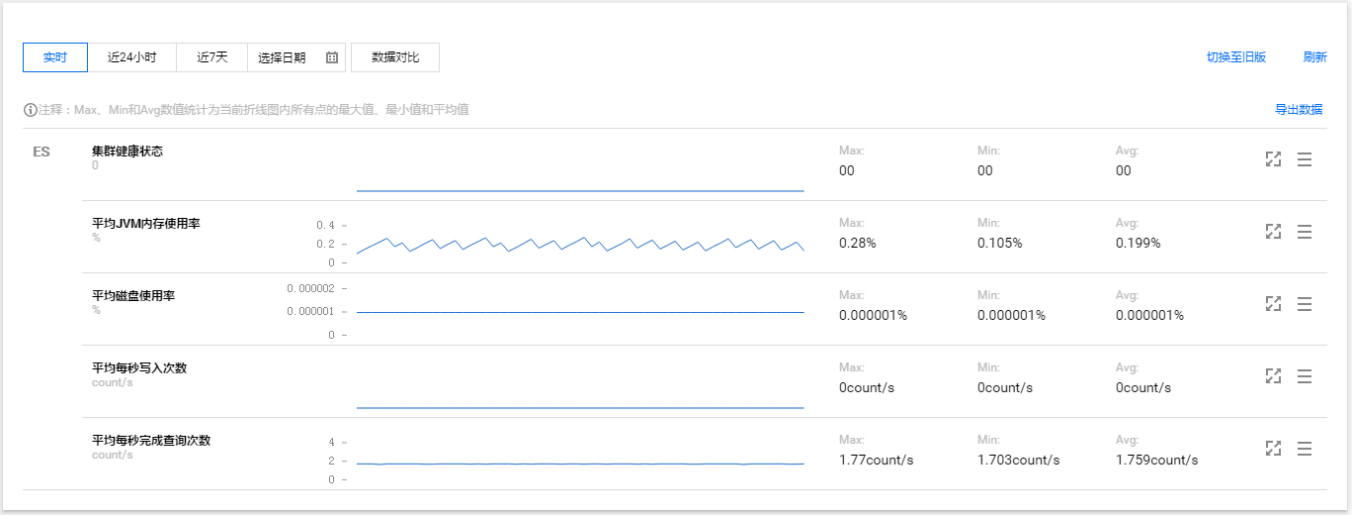
操作步骤

1. 登录 ES 控制台，在集群列表单击集群 ID/名称，进入集群详情页。
2. 选择【集群监控】页，可查看集群的运行情况，包括集群整体运行情况和各个节点运行情况。

集群状态

- 实时运行情况
  展示集群运行需要关注的一些实时的核心指标，您可以通过该部分数据，了解到当前集群的整体运行状态。

- 历史运行情况
  展示详细的、随时间变化的指标，可以了解集群过去一段时间内的运行情况。

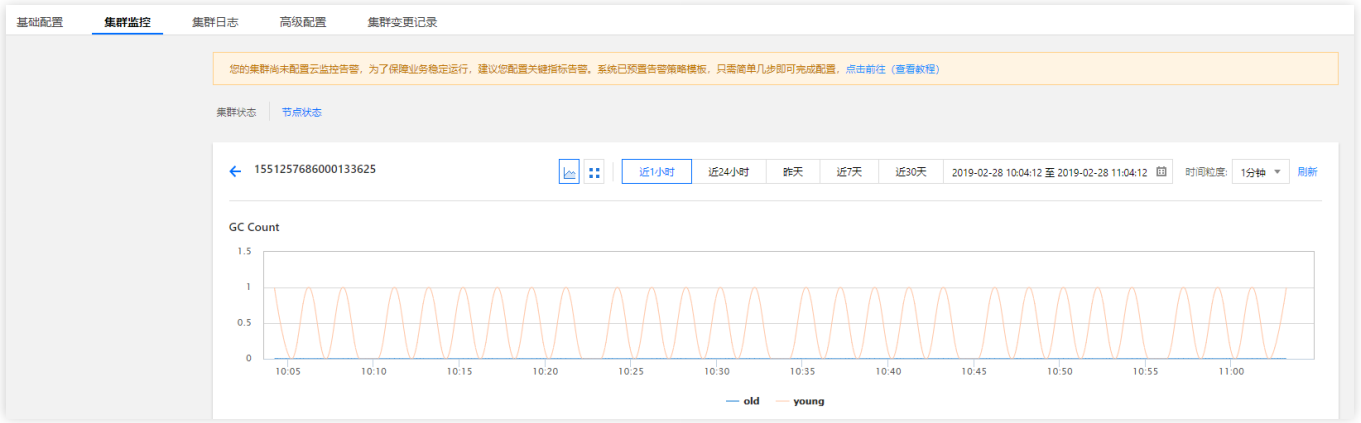


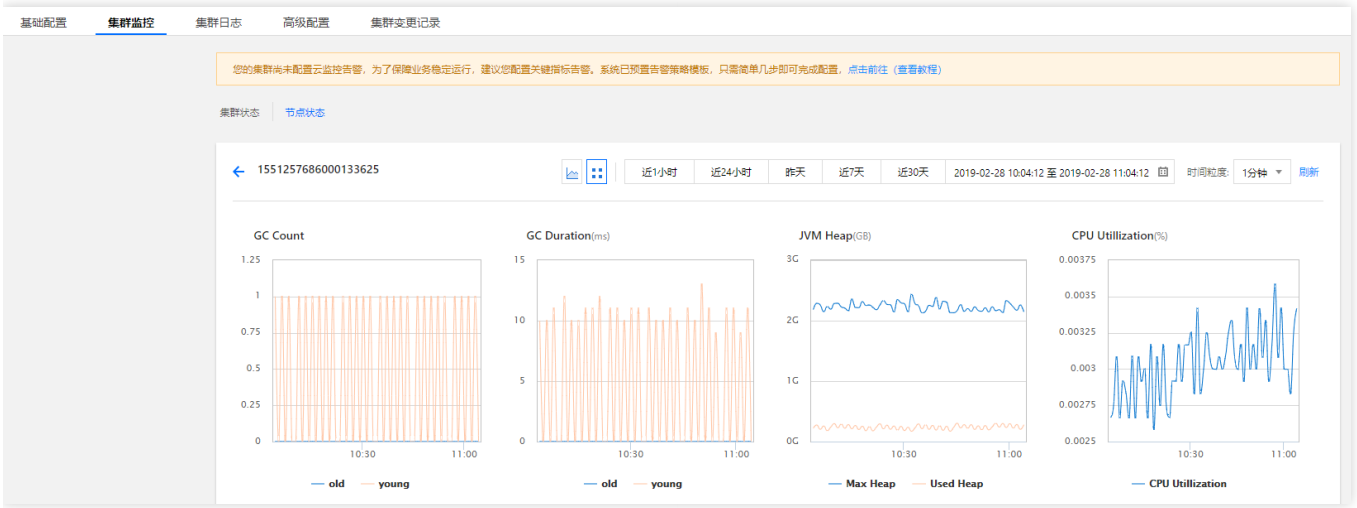
也可通过 云监控控制台 查看 ES 集群完整的监控指标。

节点状态

- 节点列表
  展示集群各个节点部分实时运行指标。

- 单个节点运行详情
  提供每个节点，各项指标详细的历史运行情况。





部分指标含义及说明

ES 集群一般由多个节点构成，为反映集群整体的运行情况，部分监控指标提供了两类值：平均值、最大值。
平均值表示集群所有节点该指标值的平均数，最大值表示集群所有节点该指标的最大值。所有指标的统计周期均为1分钟，即每1分钟对集群的指标采集1次。具体各指标含义说明如下：

监控指标	统计方式	详情
集群健康状态	ES 集群健康状态： 0：表示绿色，集群正常； 1：表示黄色，告警，部分副本分片不可用； 2：表示红色，异常，部分主分片不可用。	green：表示所有的主分片和副本分片都可用，集群处于最健康的状态。 yellow：表示所有的主分片均可用，但部分副本分片不可用，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，数据面临较高的丢失风险。在集群健康状态变为 yellow 后，应及时调查和定位问题，并修复，防止数据丢失。 red：表示至少一个主分片以及

监控指标	统计方式	详情
		它的全部副本均不可用。集群处于 red 状态意味着已有部分数据丢失：搜索只能返回部分数据，而分配到丢失分片上的写入请求会返回异常。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。
平均磁盘使用率	每单位统计周期内（1分钟），集群各个节点的磁盘使用率的平均值。	磁盘使用率过高会导致数据无法正常写入。解决方法： 及时清理无用的索引。 对集群进行扩容，增加单节点的磁盘容量或增加节点个数。
最大磁盘使用率	每单位统计周期内（1分钟），集群各个节点中最大磁盘使用率。	-
平均 JVM 内存使用率	每单位统计周期内（1分钟），集群各个节点的 JVM 内存使用率的平均值。	该值过高会导致集群节点 GC 频繁，甚至有出现 OOM。 导致该值过高的原因，一般是节点上 ES 处理任务超出节点 JVM 的负载能力。您需要注意观察集群正在执行的任务，或调整集群的配置。
最大 JVM 内存使用率	每单位统计周期内（1分钟），集群各个节点中最大 JVM 内存使用率。	-
平均 CPU 使用率	每单位统计周期内（1分钟），集群各个节点的 CPU 使用率的平均值。	当集群各节点处理的读写任务超出节点 CPU 的负载能力时，该指标就会过高，CPU 使用率过高会导致集群节点处理能力下降，甚至宕机。您可以从以下几点解决平均 CPU 使用率过高的问题： 观察该指标是持续性较高，还是临时飙升。若是临时飙升，确定是否有临时性复杂任务正在执行。 若该指标持续较高，分析业务对集群的读写操作是否可以优化，降低读写频率，减小数据量，从而减轻节点负载。 对于节点配置无法满足业务吞吐量的情况，建议对集群节点进行纵向扩容，提高单节点的负载能力。

监控指标	统计方式	详情
最大 CPU 使用率	每单位统计周期内（1分钟），集群各个节点中最大 CPU 使用率。	-
集群1分钟平均负载	集群1分钟所有节点的平均负载 load_1m，指标来源：ES 节点状态 api：_nodes/stats/os/cpu/load_average/1m。	load_1m 过高时，建议降低集群负载或调大集群节点规格。
集群1分钟最大负载	集群1分钟所有节点的最大平均负载 load_1m。	-
平均写入延迟	写入延迟（index_latency），指单次 index 请求耗时（ms/次），集群平均写入延迟是统计周期内（1分钟）所有节点单次 index 请求耗时的平均值。 节点单次 index 请求耗时计算规则：每隔一个统计周期（1分钟）记录一次节点的两个指标，节点历史 index 总次数（_nodes/stats/indices/indexing/index_total），节点历史 index 总耗时（_nodes/stats/indices/indexing/index_time_in_millis），取相邻两次记录的差值，即一个周期内的绝对值并进行计算：index 耗时 ÷ index 次数，得出统计周期内（1分钟）单次 index 平均耗时。	写入延迟，是指单个文档写入平均耗时。集群平均写入延迟，是指统计周期内，所有节点的写入用时的平均值。 写入延迟过高时，建议调大节点规格或增加节点个数。
最大写入延迟	写入延迟（index_latency），指单次 index 请求耗时（ms/次），集群最大写入延迟是统计周期内（1分钟）所有节点中单次 index 请求耗时的最大值。 节点单次 index 请求耗时计算规则：见平均写入延迟。	-
平均查询延迟	查询延迟（search_latency），指单次查询请求耗时（ms/次），集群平均查询延迟是统计周期内（1分钟）所有节点单次查询请求耗时的平均值。 节点单次查询请求耗时计算规则：每隔一个统计周期（1分钟）记录一次节点的两个指标，节点历史查询总次数（_nodes/stats/indices/search/query_total），节点历史查询总耗时（_nodes/stats/indices/search/query_time_in_millis），取相邻两次记录的差值，即一个周期内的绝对值并进行计算：query 耗时 ÷ query 次数，得出统计周期内（1分钟）单次查询平均耗时。	查询延迟，是指单个查询平均耗时。集群平均查询延迟，就是统计周期内，所有节点查询用时的平均值。 写入延迟过高时，建议调大节点规格或增加节点个数。
最大查询延迟	查询延迟（search_latency），指单次查询请求耗时（ms/次），集群最大查询延迟是统计周期内（1分钟）所有节点中单次查询请求耗时的最大值。 节点单次查询请求耗时计算规则：见平均查询延迟。	-
平均每秒写入次数	集群所有节点接收到的每秒 index 请求次数的平均值。节点每秒 index 请求次数计算规则：每隔一个统计周期（1分钟）记录一次节点历史 index 总次数（_nodes/stats/indices/indexing/index_total），取相邻两次记录的差值，即一个周期内的绝对值并进行计算：index 次数 ÷ 60秒，得出统计周期内每秒 index 请求次数的平均值	-

监控指标	统计方式	详情
平均每秒完成查询次数	集群所有节点接收到的每秒查询请求次数的平均值。节点每秒查询请求次数计算规则：每个统计周期（1分钟）记录一次节点历史查询总次数（_nodes/stats/indices/search/query_total），取相邻两次记录的差值，即一个周期内的绝对值并进行计算：query 次数 ÷ 60秒，得到统计周期内每秒查询请求次数的平均值。	-
写入拒绝率	单位周期内，集群写入请求被拒绝次数 ÷ 总写入请求次数，得到的比率。具体计算规则：每隔一个统计周期采集两个指标：历史写入请求被拒绝次数（5.6.4版本：_nodes/stats/thread_pool/bulk/rejected，6.4.3及以后版本： _nodes/stats/thread_pool/write/rejected）、历史写入请求次数（5.6.4版本：_nodes/stats/thread_pool/bulk/completed，6.4.3及以后版本：_nodes/stats/thread_pool/write/completed），取相邻两次记录的差值，即一个周期内的绝对值并进行计算，写入请求拒绝次数 ÷ 写入请求完成次数。	写入 QPS 过大，CPU、内存、磁盘使用率过高时，可能会造成集群写入拒绝率会增加。一般地，是集群当前配置无法满足业务写操作需求。对于节点配置过低的场景，可以通过提高节点规格或降低写入操作次数来解决。对于磁盘使用率过高的情况，可以通过扩容集群磁盘或删除无用数据来解决。
查询拒绝率	单位周期内，集群查询请求被拒绝次数 ÷ 总查询请求次数，得到的比率。具体计算规则：每隔一个统计周期采集两个指标：历史查询请求被拒绝次数（_nodes/stats/thread_pool/search/rejected）、历史查询请求次数（_nodes/stats/thread_pool/search/completed），取相邻两次记录的差值，即一个周期内的绝对值并进行计算：查询请求拒绝次数 ÷ 查询请求完成次数。	写入 QPS 过大，CPU、内存使用率过高，可能会造成造成集群查询拒绝率上升。一般地，是集群当前配置无法满足业务读操作需求，该值过高时建议对集群调大集群节点配置，提高集群节点的处理能力。
集群总文档数	集群写入的 doc 总数，计算规则：ES 集群文档数 API：_cluster/stats/indices/docs/count。	-
自动快照备份状态	集群开启自动快照备份后的备份结果： 0：未开启自动备份； 1：自动备份正常； -1：自动备份失败。	自动快照备份，会把集群的数据定时备份到 COS，以便在需要的时候可以对数据进行恢复，从而更全面保障数据安全，建议开启，更多信息可查看：自动快照备份

配置告警

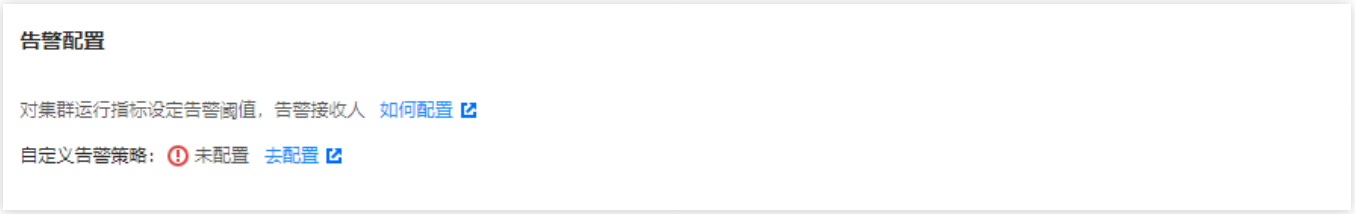
操作场景

TCloudFinanceZone ES 提供一些关键指标的配置告警功能，配置告警可帮助您及时发现集群问题并进行处理。本文为您介绍通过控制台配置告警的操作。

操作步骤

查看集群是否已配置告警

1. 登录 ES 控制台，在集群列表单击集群 ID/名称，进入集群详情页。
2. 选择【集群监控】页，在【告警配置】区域，可查看集群是否已经配置了告警，如果没有，强烈建议您配置告警策略，以便及时获取并处理集群运行的状况及风险，保障服务的稳定。

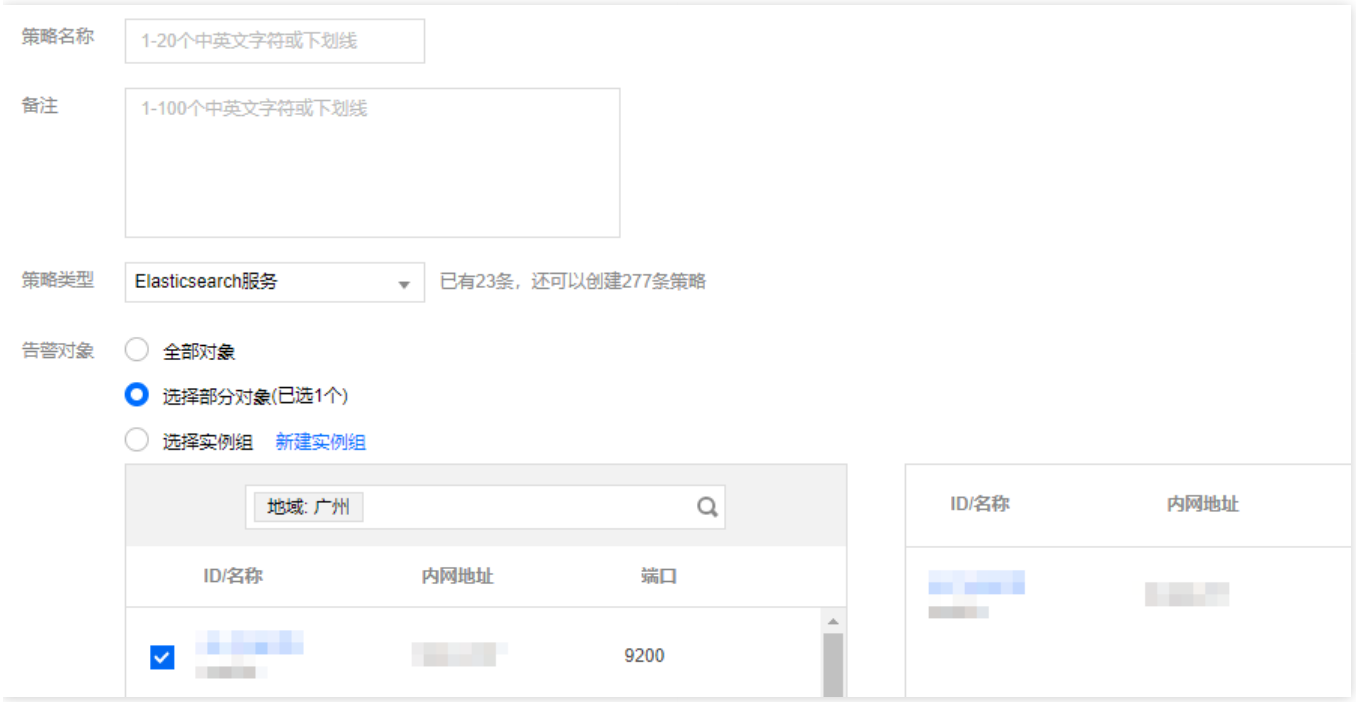


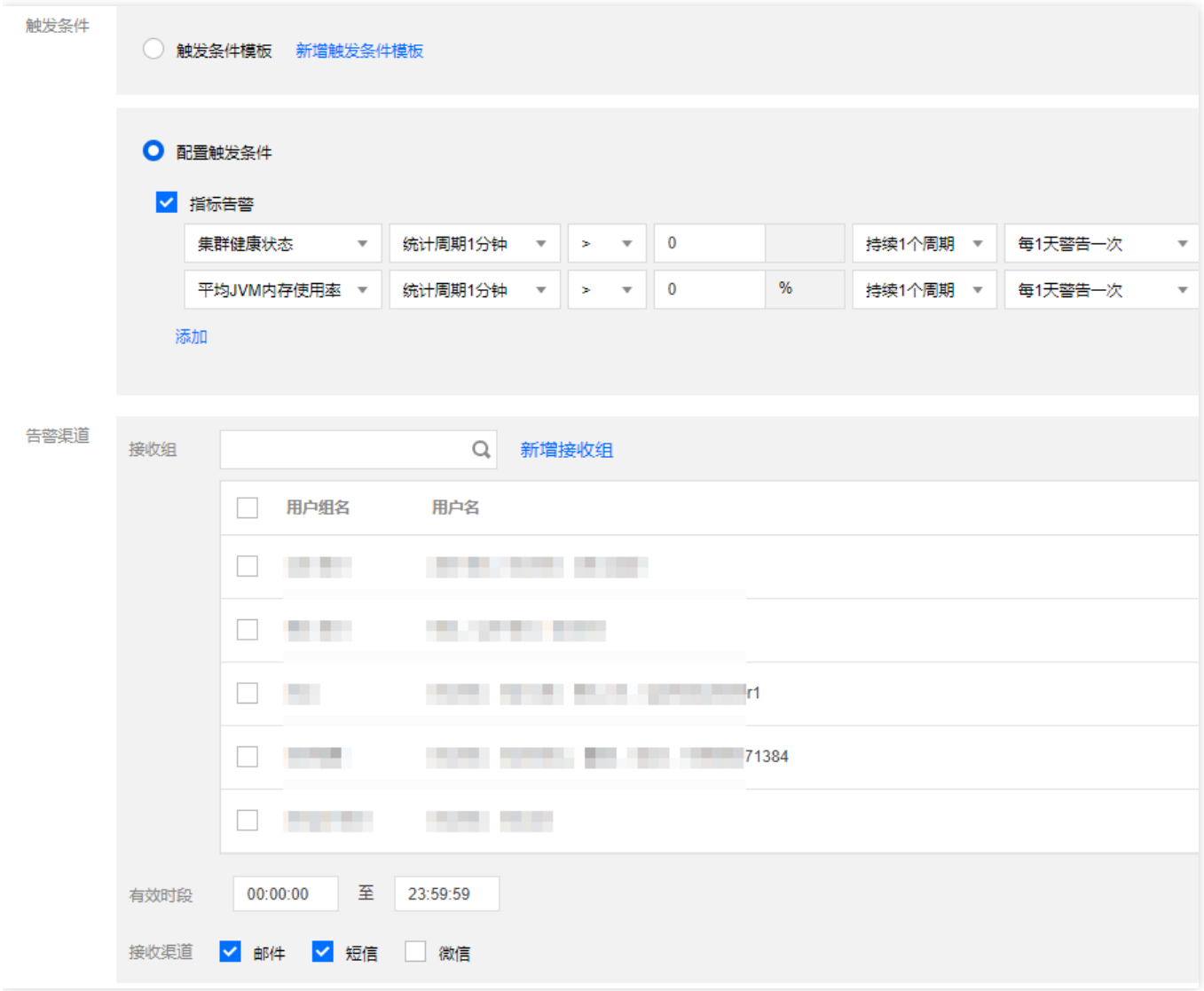
也可登录云监控控制台 ，在【告警配置】>【告警策略】，通过筛选策略和产品，查询某个集群是否已经配置了告警策略。

自定义告警配置

1. 登录云监控控制台，在【告警策略】页，单击【新建】。
2. 在新建策略页面，配置策略参数。
   - 策略类型：选择【Elasticsearch 服务】。
   - 告警对象：选择需要配置告警策略的集群。
   - 触发条件：支持【触发条件模板】和【配置触发条件】，默认选择自定义配置触发条件，自定义配置参见以下说明，新建模板参见下文 “新建触发条件模板”。
   - 指标：例如“CPU 使用率”，统计周期为1分钟或5分钟。因 ES 集群的各项指标都是1分钟采集1次，所以选择统计周期是1分钟时，当集群出现一次超过阈值就会触发告警，如果选择5分钟，则5分钟内，连续超过阈值才会触发告警。
   - 告警频次：例如“每30分钟警告一次”，指每30分钟内，连续多个统计周期指标都超过了阈值，如果有一次告警，30分钟内就不会再次进行告警，直到下一个30分钟，如果指标依然超过阈值，才会再次告警。
   - 告警渠道：选择接收组、有效时段、接收渠道。配置方法可参见 创建和管理告警接收组。
3. 配置完成后，单击【完成】，返回到【告警策略】列表，即可查看到刚配置的告警策略。

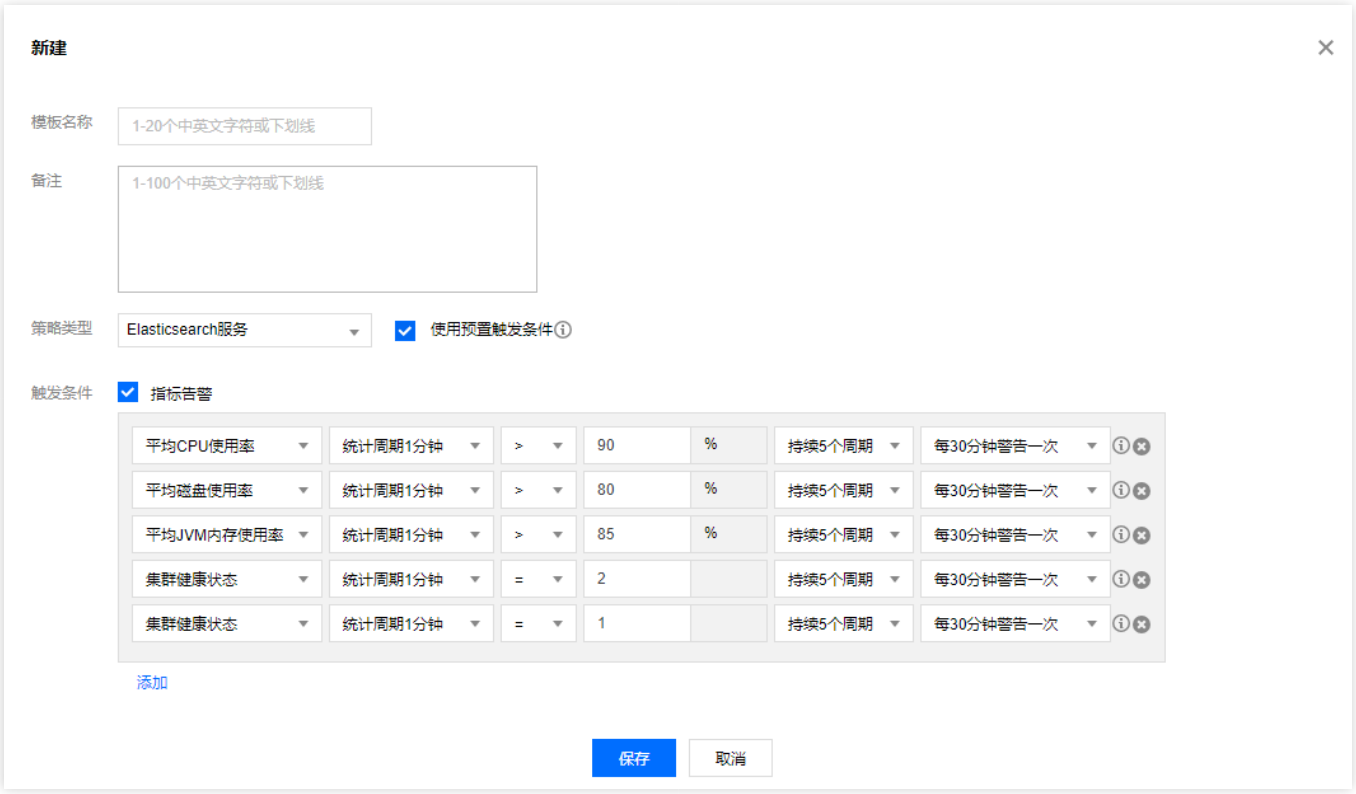
告警策略，更详细配置教程可参见 云监控告警配置。


策略名称
1-20个中英文字符或下划线
备注
1-100个中英文字符或下划线
策略类型
Elasticsearch服务
已有23条，还可以创建277条策略
告警对象
全部对象
选择部分对象(已选1个)
选择实例组
新建实例组
地域: 广州
ID/名称
内网地址
端口
9200
ID/名称
内网地址



新建触发条件模板

1. 在【触发条件】单击【新增触发条件模板】。
2. 在触发条件模板页单击【新建】。
3. 在新建模板页，配置策略类型。
    - 策略类型：选择【Elasticsearch 服务】。
    - 使用预置触发条件：勾选此选项，会出现系统建议的告警策略。
4. 确认无误后，单击【保存】。



5. 返回新建告警策略页，单击【刷新】，就会出现刚配置的告警策略模板。

数据备份
使用对象存储进行备份及恢复

创建仓库

您可以通过如下命令创建仓库：

```
PUT _snapshot/my_cos_backup
{
    "type": "cos",
    "settings": {
        "app_id": "xxxxxxx",
        "access_key_id": "xxxxxx",
        "access_key_secret": "xxxxxxx",
        "bucket": "xxxxxx",
        "region": "ap-guangzhou",
        "compress": true,
        "chunk_size": "500mb",
        "base_path": "/"
    }
}
```

- app_id：TCloudFinanceZone账号 APPID。
- access_key_id：TCloudFinanceZone API 密钥 SecretId。
- access_key_secret：TCloudFinanceZone API 密钥 SecretKey。
- bucket：对象存储 Bucket 名字，名字不能带 -{appId} 后缀。
- region：对象存储 Bucket 地域，建议与 ES 集群同地域。
- base_path：备份目录。

列出仓库信息

您可通过 GET _snapshot 获取仓库信息，也可使用 GET _snapshot/my_cos_backup 获取指定的仓库信息。

创建快照备份

备份所有索引

将 ES 集群内所有索引备份到 my_cos_backup 仓库下，并命名为 snapshot_1 。

```
PUT _snapshot/my_cos_backup/snapshot_1
```

这个命令会立刻返回，并在后台异步执行直到结束。如果希望创建快照命令阻塞执行，可以添加 wait_for_completion 参数。命令执行的时间与索引大小相关。

```
PUT _snapshot/my_cos_backup/snapshot_1?wait_for_completion=true
```

备份指定索引

您可以在创建快照的时候指定要备份的索引。参数 indices 的值为多个索引的时候，需要用，隔开且不能有空格。

```
PUT _snapshot/my_cos_backup/snapshot_2
{
    "indices": "index_1,index_2"
}
```

查询快照

查询单个快照信息：

```
GET _snapshot/my_cos_backup/snapshot_1
```

该命令会返回快照的相关信息：

```
{
    "snapshots": [
        {
            "snapshot": "snapshot_1",
            "uuid": "zUSugNiGR-OzH0CCcgcLmQ",
            "version_id": 5060499,
            "version": "5.6.4",
            "indices": [
                "index_1",
                "index_2"
            ],
            "state": "SUCCESS",
            "start_time": "2018-05-04T11:44:15.975Z",
            "start_time_in_millis": 1525434255975,
            "end_time": "2018-05-04T11:45:29.395Z",
            "end_time_in_millis": 1525434329395,
            "duration_in_millis": 73420,
            "failures": [],
```

```
            "shards": {
                "total": 3,
                "failed": 0,
                "successful": 3
            }
        }
    ]
}
```

删除快照

删除指定的快照：

```
DELETE _snapshot/my_cos_backup/snapshot_1
```

> 注意：
>
> 如果存在还未完成的快照，删除快照命令依旧会执行，并取消未完成快照的创建进程。

从快照恢复

1. 将快照中备份的所有索引都恢复到 ES 集群中。

   ```
   POST _snapshot/my_cos_backup/snapshot_1/_restore
   ```

   - 如果 snapshot_1 包括5个索引，则这5个索引都会被恢复到 ES 集群中。
   - 您还可以使用附加的选项对索引进行重命名。该选项允许您通过模式匹配索引名称，并通过恢复进程提供一个新名称。如果您想在不替换现有数据的前提下，恢复旧数据来验证内容或进行其他操作，则可以使用该选项。

2. 从快照里恢复单个索引并提供一个替换的名称。

   ```
   POST /_snapshot/my_cos_backup/snapshot_1/_restore
   {
     "indices": "index_1",
     "rename_pattern": "index_(.+)",
     "rename_replacement": "restored_index_$1"
   }
   ```

   - indices：只恢复 index_1 索引，忽略快照中存在的其他索引。
   - rename_pattern：查找所提供的模式能匹配上的正在恢复的索引。

- rename_replacement：将匹配的索引重命名成替代的模式。

查询快照恢复状态

可通过执行 _recovery 命令，查看快照恢复的状态并监控快照恢复的进度。

1. 您可以在恢复的指定索引中单独调用下述 API。

```
GET index_1/_recovery
```

2. 该命令会返回指定索引各分片的恢复状况。


```
{
   "sonested": {
      "shards": [
         {
            "id": 1,
            "type": "SNAPSHOT",
            "stage": "INDEX",
            "primary": true,
            "start_time_in_millis": 1525766148333,
            "total_time_in_millis": 8718,
            "source": {
               "repository": "my_cos_backup",
               "snapshot": "snapshot",
               "version": "5.6.4",
               "index": "index_1"
            },
            "target": {
               "id": "TlzmxJHwSqyv4rhyQfRkow",
               "host": "10.0.0.6",
               "transport_address": "10.0.0.6:9300",
               "ip": "10.0.0.6",
               "name": "node-1"
            },
            "index": {
               "size": {
                  "total_in_bytes": 1374967573,
                  "reused_in_bytes": 0,
                  "recovered_in_bytes": 160467084,
                  "percent": "11.7%"
               },
               "files": {
```

```
                "total": 132,
                "reused": 0,
                "recovered": 20,
                "percent": "15.2%"
            },
            "total_time_in_millis": 8716,
            "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": 0
        },
        "verify_index": {
            "check_index_time_in_millis": 0,
            "total_time_in_millis": 0
        }
    }
  ]
 }
}
```


- type：描述恢复的本质，该分片是在从一个快照恢复。
- source：哈希描述作为恢复来源的特定快照和仓库。
- percent：描述恢复的状态。该特定分片目前已经恢复了94%的文件，即将全部恢复。

输出会列出所有目前正在恢复的索引，以及这些索引里的所有分片。每个分片里会有启动/停止时间、持续时间、恢复百分比、传输字节数等统计值。

取消快照恢复

```
DELETE /restored_index_1
```

如果 restored_index_1 正在恢复中，则该删除命令会停止恢复，同时删除所有已经恢复到集群中的数据。

自动快照备份

ES 提供自动备份，自动创建集群的主索引分片的快照并备份到 COS，进而根据需要将备份数据恢复到集群。

备份说明

- ES 自动快照备份，只保留最近7天的快照数据。
- 自动快照备份的数据只能用于恢复到原集群，如果需要跨集群恢复，请参考手动快照。
- 默认情况下，自动快照备份在凌晨进行，建议您根据业务需求选择集群访问压力不大的时间进行。
- 集群第一个快照是集群数据的完整拷贝，执行时间视具体数据量而定，之后的快照保留的是已存快照和新数据之间的增量差异。
- 集群健康状态为红色时，自动快照服务将停止创建，建议您关注集群的健康状态。

操作步骤

开启自动快照备份

1. 登录 ES 控制台，在【集群列表】单击【集群名称】进入集群详情页。
2. 在【高级配置】页，可以开启自动快照备份以及配置自动快照备份的时间。

查看快照存储库

在集群详情页单击【Kibana】进入 Kibana 控制台，在“Dev Tools”页可以通过 ES 的 API 查看集群的所有快照存储库。

- 查看集群的快照存储库。

```
GET _snapshot?pretty
```

如果只有自动快照，将得到如下的返回信息：

```
{
  "ES_AUTO_BACKUP": {
    "type": "cos",
    "settings": {
      "bucket": "es-ap-guangzhou",
      "base_path": "/es_backup/es-2s8x1b9u",
      "chunk_size": "500mb",
      "region": "ap-guangzhou",
      "compress": "true"
    }
  }
}
```

- 查看自动快照存储库中的快照信息。

```
GET _snapshot/ES_AUTO_BACKUP/_all?pretty
```

返回结果如下：

如果没有快照，列表为空。

```
{
  "snapshots": []
}
{
  "snapshots": [
    {
      "snapshot": "es-2s8x1b9u_20181220",
      "uuid": "gsXPyWb1SNOlTuj3eNs2gA",
      "version_id": 5060499,
      "version": "5.6.4",
      "indices": [
        ".kibana"
      ],
      "state": "SUCCESS",
      "start_time": "2018-12-20T08:00:12.336Z",
      "start_time_in_millis": 1545292812336,
      "end_time": "2018-12-20T08:00:12.945Z",
```

```
      "end_time_in_millis": 1545292812945,
      "duration_in_millis": 609,
      "failures": [],
      "shards": {
        "total": 1,
        "failed": 0,
        "successful": 1
      }
    }
  ]
}
```

恢复数据

```
POST _snapshot/ES_AUTO_BACKUP/es-2s8x1b9u_20181220/_restore
```

升级
升级ES集群

ES 提供了版本升级的功能，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。

支持的升级操作

Elasticsearch 版本升级

源 Elasticsearch 版本	目标 Elasticsearch 版本
低版本（如5.6版本）	高版本（如6.8版本）

关于升级的重启

对于升级重启情况，在执行升级操作时，集群可能：

- 不重启
- 滚动重启（节点依次重启，期间服务可以正常访问，但性能可能受到部分影响，建议在集群负载不高时进行）

根据您的选择，升级界面对这些情况均会给出提示说明。

升级注意事项

Elasticsearch 5.x版本与6.x版本兼容及使用说明

1. index 多 type。
   Elasticsearch 从 6.x 开始，不再支持一个索引多个 type。从5.x版本升级到6.x版本后，新建多 type 的索引将会报错，原有在5.x版本创建的多 type 索引，不受影响，可以正常写入。

2. curl 访问集群。
   用 curl 访问集群时，需要增加请求 header -H 'Content-Type: application/json' 。

```
curl -XPUT http://10.0.0.2:9200/china/city/beijing -H 'Content-Type: application/json' -d'
{
    "name":"北京市",
    "province":"北京市",
    "lat":39.9031324643,
    "lon":116.4010433787,
    "x":6763,
```

```
    "level.range":4,
    "level.level":1,
    "level.name":"一线城市",
    "y":6381,
    "cityNo":1
}
```


3. 配置项兼容。
   Elasticsearch 不同版本之间会存在一些不兼容的配置，如果您有设置，升级后可能会影响集群的使用。ES 升级功能提供了配置项的检查流程，也提供了调整说明，见下文 升级检查。

4. 更多说明请参考 Breaking changes in 6.0。

升级处理流程说明

Elasticsearch 版本升级，需要先进行升级检查、数据备份两个步骤。前两项成功后，才会开始执行升级操作。

1. 升级检查。
   只有 Elasticsearch 版本升级会有。
   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，您可以查看具体的检查项和相应的解决方案。具体请参考 升级检查。您也可以在升级操作前，只选择升级检查操作，查看集群是否满足升级条件。
2. 快照备份。
   只有 Elasticsearch 版本升级会有。

升级操作前，ES 会先对您的集群进行快照备份，以防升级操作失败时，可以用快照还原集群。所以，如果快照这一步失败，升级流程也会终止。快照备份耗时同集群数据量有关，如果集群未开启自动快照备份，且数据量较大，第一次快照时间会比较长。

集群升级操作步骤

1. 登录 ES 控制台，进入集群详情页，单击右上角【升级】。

2. 在升级对话框中可以选择升级 Elasticsearch 版本。

升级 Elasticsearch 版本

1. 在升级对话框的【升级类型】中选择【升级 Elasticsearch 版本】。

2. 在【Elasticsearch 版本】下拉框中选择要升级到的版本。

3. 我们在升级之前会先对集群状态进行检查，判断集群是否可以升级，包括检查集群的配置、集群的状态等。您可以勾选【仅做升级检查】，在单击【确定】后，仅做升级检查，不执行升级命令，您可以在详情页【集群变更记录】中查看检查结果。

> ES主版本升级，例如5.x版本到6.x版本，有些集群级别的配置项和索引级别的配置项不兼容，需要通过【升级检查】判断集群是否可升级。整个升级检查会检查一些配置项，错误的配置项需要您进行调整，告警的配置项可以选择性调整，具体可查看 。

4. 阅读升级须知，同意并勾选后单击【确定】开始版本升级（如勾选【仅做升级检查】则开始版本升级检查）。

升级

版本升级说明

- 集群版本升级，由于集群状态、版本兼容问题，需要先检测是否能支持当前版本升级到目标版本；
- 如果支持升级，升级操作时，会首先对集群进行快照备份，建议在集群负载比较低时执行；
- 开始升级后，可以在“操作日志”查看升级进度。

集群ID	集群名称	当前Elasticsearch版本
[illegible]	[illegible]	6.4.3

升级类型 ◉ 升级 Elasticsearch 版本

Elasticsearch 版本 6.8.2

仅做升级检查 ☐ 勾选后，只做升级检查判断集群是否可以升级，不执行升级命令，可在详情页【集群变更记录】查看检查结果。

操作提示 ☐ 开始升级检查或开始升级后，流程无法取消，期间可以继续向集群写入和读取数据，不停服，但不能修改其他配置，如扩容操作等。

确定 取消

ES版本升级检查

Elasticsearch 不同版本之间有部分不兼容的配置，如果您已设置，升级后，使用集群可能会受到影响。您可通过升级检查的功能，检查是否有不兼容的配置，并进行调整。以下是 ES 对 Elasticsearch 版本升级时检查的配置说明。
升级检查操作入口，在 控制台 详情页，单击右上角【升级】，操作步骤详见 升级 ES 集群。
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，例如，6.8版本升级到7.5版本后默认无法创建包含 type 的索引，详情见下文 需人工自查的不兼容配置。

Elasticsearch 5.x 版本升级到 6.x 版本配置检查

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	集群快照设置（Snapshot settings）	CRITICAL	集群设置 cluster.routing.allocation.snapshot.relocation_enabled ，在6.0版本开始被废弃，详见 Breaking changes in 6.0
2	集群级别	集群存储限流设置（Store throttling settings）	CRITICAL	集群设置 indices.store.throttle.type 和 indices.store.throttle.max_bytes_per_sec ，在6.0版本开始被废弃，详见 Breaking changes in 6.0
3	索引级别	索引相似性设置（Similarity settings）	WARNING	索引设置 index.similarity.base ，在6.0版本开始被废弃，详见 Breaking changes in 6.0
4	索引级别	索引影子副本设置（Shadow Replicas settings）	CRITICAL	索引设置 index.shared_filesystem 和 index.shadow_replicas ，在6.0版本开始被废弃，详见 Breaking changes in 6.0
5	索引级别	索引存储设置（Index Store settings）	CRITICAL	索引设置 index.store.type 为 default，在6.0版本开始被废弃，详见 Breaking changes in 6.0
6	索引级别	索引存储限流设置（Index Store throttling settings）	CRITICAL	索引设置 index.store.throttle.type 和 index.store.throttle.max_bytes_per_sec ，在6.0版本开始被废弃，详见 Breaking changes in 6.0
7	索引级别	索引 Mapping 参数 include_in_all	WARNING	索引 mapping 参数 include_in_all ，在6.0版本之后创建的索引中无法使用（5.x 版本创建包含此设置的索引在升级 6.x 版本后可以兼容），详见 Breaking changes in 6.0
8	索引级别	索引创建版本	CRITICAL	索引创建版本 index.version.created 不允许跨 ES 主版本，例如，无法将在 5.x 版本创建的索引直接升级到 7.x 版本，需要将失败的索引 reindex 迁移到新索引并删除后再进行升级。
9	索引模板级别	索引模板相似性设置（Similarity settings）	CRITICAL	索引设置 index.similarity.base ，在6.0版本开始被废弃，详见 Breaking changes in 6.0，模板中存在该设置将导致版本升级后该模板无法用于创建新索引。
10	索引模板级别	索引模板影子副本设置（Shadow Replicas settings）	CRITICAL	索引设置 index.shared_filesystem 和 index.shadow_replicas ，在6.0版本开始被废弃，详见 Breaking changes in 6.0，模板中存在该设置将导致版本升级后该模板无法用于创建新索引。
11	索引模板级别	索引模板存储设置（Index Store settings）	CRITICAL	索引设置 index.store.type 为 default，在6.0版本开始被废弃，详见 Breaking changes in 6.0，模板中存在该设置将导致版本升级后该模板无法用于创建新索引。
12	索引模板级别	索引模板存储限流设置（Index Store throttling settings）	CRITICAL	索引设置 index.store.throttle.type 和 index.store.throttle.max_bytes_per_sec ，在6.0版本开始被废弃，详见 Breaking changes in 6.0，模板中存在该设置将导致版本升级后该模板无法用于创建新索引。
13	索引模板级别	索引模板 Mapping 参数 include_in_all	CRITICAL	索引 Mapping 参数 include_in_all ，在6.0版本开始被废弃，详见 Breaking changes in 6.0，模板中存在该设置将导致版本升级后该模板无法用于创建新索引。

序号	配置维度	配置信息	兼容性	详细说明
14	索引模板级别	索引模板 Mapping 元字段 _all	CRITICAL	索引 Mapping 元字段 _all ，在6.0版本开始被废弃，详见 Breaking changes in 6.0，模板中存在该设置将导致版本升级后该模板无法用于创建新索引。
15	索引模板级别	索引模板 Mapping 多个 Types	CRITICAL	索引 Mapping 包含多个 Types，在6.0版本开始被废弃，详见 Removal of mapping types，模板中存在该设置将导致版本升级后该模板无法用于创建新索引。

- 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

- 集群快照设置（Snapshot settings）
  通过 ES 集群设置更新接口 PUT _cluster/settings 取消此设置（包括 persistent 和 transient）：

```
PUT _cluster/settings
{
   "persistent": {
     "cluster.routing.allocation.snapshot.relocation_enabled": null
   },
   "transient": {
     "cluster.routing.allocation.snapshot.relocation_enabled": null
   }
}
```

- 集群存储限流设置（Store throttling settings）
  通过 ES 集群设置更新接口 PUT _cluster/settings 取消此设置（包括 persistent 和 transient）：

```
PUT _cluster/settings
{
   "persistent": {
     "indices.store.throttle.type": null,
     "indices.store.throttle.max_bytes_per_sec": null
   },
   "transient": {
     "indices.store.throttle.type": null,
     "indices.store.throttle.max_bytes_per_sec": null
   }
}
```

索引级别

- 索引相似性设置（Similarity settings）
  此设置在升级 6.x 后被忽略，但不会影响升级。如需取消设置，可按照如下步骤操作：

- 此设置需要关闭索引后修改，关闭后的索引无法被读写。关闭索引：

```
POST my_index/_close
```

- 使用 ES 索引设置更新接口取消此设置：

```
PUT my_index/_settings
{
     "index.similarity.base.*": null
}
```

- 最后打开索引：

```
POST my_index/_open
```

- 索引影子副本设置（Shadow Replicas settings）
- 此设置需要关闭索引后修改，关闭后的索引无法被读写。关闭索引：

```
POST my_index/_close
```

- 使用 ES 索引设置更新接口取消此设置：

```
PUT my_index/_settings
{
    "index.shared_filesystem": null,
    "index.shadow_replicas": null
}
```

- 最后打开索引：

```
POST my_index/_open
```

- 索引存储设置（Index Store settings）
- 此设置需要关闭索引后修改，关闭后的索引无法被读写。关闭索引：

```
POST my_index/_close
```

- 使用 ES 索引设置更新接口取消此设置：

```
PUT my_index/_settings
{
    "index.store.type": null
}
```

- 最后打开索引：

```
POST my_index/_open
```

- 索引存储限流设置（Index Store throttling settings）
  使用 ES 索引设置更新接口取消此设置：

```
PUT my_index/_settings
{
  "settings": {
    "index.store.throttle.type": null,
    "index.store.throttle.max_bytes_per_sec": null
  }
}
```

- 索引 Mapping 参数 include_in_all
  对已创建的包含此参数的索引，升级后可以兼容，无需修复。

索引模板级别

1. 使用 API GET _template/my_template 获取不兼容模板 my_template 。存在以下三个不兼容设置：索引模板存储设置（Index Store settings）、索引模板 Mapping 参数 include_in_all 、索引模板 Mapping 元字段 _all 。

```
{
  "my_template": {
    "order": 0,
    "template": "my_*",
    "settings": {
      "index": {
        "store": {
          "throttle": {
            "max_bytes_per_sec": "10m"
          }
        }
      }
    },
    "mappings": {
      "my_type": {
        "_all": {
          "enabled": true
        },
        "properties": {
          "my_field": {
            "type": "text",
            "include_in_all": true
          }
        }
      }
    },
    "aliases": {}
  }
}
```

2. 拷贝并去除模板中不兼容配置后，使用 API PUT _template/my_template 更新模板。

```
PUT _template/my_template
{
    "order": 0,
    "template": "my_*",
    "settings": {
    },
    "mappings": {
      "my_type": {
        "properties": {
          "my_field": {
            "type": "text"
          }
        }
      }
    },
    "aliases": {}
}
```

Elasticsearch 6.8 版本升级到 7.x 版本配置检查

需人工自查的不兼容配置">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 ：

> = 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"
        }
      }
    }
  }
}
```

可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	WARNING	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	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
6	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
7	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
8	索引级别	索引名	WARNING	索引名不能包含":"，详见 Breaking changes in 7.0
9	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
10	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
11	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
12	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
13	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

序号	配置维度	配置信息	兼容性	详细说明
14	索引级别	索引 Mapping 日期格式	WARNING	Joda-Time 格式改为 Java Time，详见 Breaking changes in 7.0
15	索引模板级别	Delimited payload 语汇单元过滤器（Token filter）索引模板设置	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
16	索引模板级别	索引模板设置（index.percolator.map_unmapped_fields_as_string）	CRITICAL	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 Breaking changes in 7.0
17	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
18	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
19	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
20	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
21	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
22	索引模板级别	索引模板 Mapping 日期格式	WARNING	Joda-Time 格式改为 Java Time，详见 Breaking changes in 7.0

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。

索引级别">索引级别

- 索引创建版本
  不允许6.0版本之前创建的索引升级到7.x版本，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"
}
```

- 通过修改 ES YML 配置文件提升最大字段数设置 indices.query.bool.max_clause_count （后续支持）。

- 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。

SQL支持

TCloudFinanceZone Elasticsearch 支持使用 SQL 代替 DSL 查询语言。对于从事产品运营、数据分析等工作以及初次接触 ES 的人，使用 SQL 语言进行查询，将会降低他们使用 ES 的学习成本。

ES 所有的开源版本，均预装了开源社区提供的 SQL 解析插件。

开源 SQL 解析插件

```
POST /_sql
"select * from my_index"
```

更多 SQL 插件的 API 及使用方法请参见 文档。

最佳实践
数据迁移和同步
数据迁移

用户在TCloudFinanceZone上自建的 ES 集群或者在其它云厂商购买的 ES 集群，如果要迁移至TCloudFinanceZone ES，用户可以根据自己的业务需要选择合适的迁移方案。如果业务可以停服或者可以暂停写操作，可以使用以下几种方式进行数据迁移：

- 对象存储快照
- logstash
- elasticsearch-dump

  各种迁移方式的对比如下：

迁移方式	适用场景
对象存储快照	数据量大的场景（GB、TB、PB 级别） 对迁移速度要求较高的场景
logstash	迁移全量或增量数据，且对实时性要求不高的场景 需要对迁移的数据通过 es query 进行简单的过滤的场景 需要对迁移的数据进行复杂的过滤或处理的场景 版本跨度较大的数据迁移场景，如 5.x 版本迁移到 6.x 版本或 7.x 版本
elasticsearch-dump	数据量较小的场景

对象存储快照

基于对象存储快照的迁移方式是使用 ES 的 snapshot api 接口进行迁移，基本原理就是从源 ES 集群创建索引快照，然后在目标 ES 集群中进行恢复。通过 snapshot 方式进行数据迁移时，特别需要注意 ES 的版本问题：

> 目标 ES 集群的主版本号（如5.6.4中的5为主版本号）要大于等于源 ES 集群的主版本号。
> 1.x 版本的集群创建的快照不能在 5.x 版本中恢复。

在源 ES 集群中创建 repository

创建快照前必须先创建 repository 仓库，一个 repository 仓库可以包含多份快照文件，repository 主要有以下几种类型。

- fs：共享文件系统，将快照文件存放于文件系统中。
- url：指定文件系统的 URL 路径，支持协议：http、https、ftp、file、jar。

- s3：AWS S3 对象存储,快照存放于 S3 中，以插件形式支持，安装该插件请参考 repository-s3。
- hdfs：快照存放于 hdfs 中，以插件形式支持，安装该插件请参考 repository-hdfs。
- 对象存储：快照存放于TCloudFinanceZone 对象存储中，以插件形式支持，安装该插件请参考 cos-repository。
  如果需要从自建 ES 集群迁移至TCloudFinanceZone的 ES 集群，可以直接使用对象存储类型仓库。但需要先在自建 ES 集群上安装 cos-repository 插件（安装插件后需要重启集群才能使用），先把自建 ES 集群中的数据先备份到对象存储，然后在TCloudFinanceZone上的 ES 集群中恢复出来，以完成数据的迁移。

如果自建 ES 的集群不方便安装 cos-repository 插件，但是已经安装 repository-s3 或者 repository-hdfs 插件，则可以先把数据备份到 S3 或者 HDFS 中，然后把 S3 或者 HDFS 中备份好的文件上传到TCloudFinanceZone对象存储中，之后在TCloudFinanceZone上的集群中进行恢复。

通过对象存储快照进行数据迁移时，需要先创建对象存储仓库，您可以通过如下命令创建仓库：

```
PUT _snapshot/my_cos_backup
{
    "type": "cos",
    "settings": {
        "app_id": "xxxxxxx",
        "access_key_id": "xxxxxx",
        "access_key_secret": "xxxxxxx",
        "bucket": "xxxxxx",
        "region": "ap-guangzhou",
        "compress": true,
        "chunk_size": "500mb",
        "base_path": "/"
    }
}
```

- app_id：TCloudFinanceZone账号 APPID。
- access_key_id：TCloudFinanceZone API 密钥 SecretId。
- access_key_secret：TCloudFinanceZone API 密钥 SecretKey。
- bucket：对象存储 Bucket 名字，注意不要带'-{appId}'后缀。
- region：对象存储 Bucket 地域，建议与 ES 集群同地域。
- base_path：备份目录。

在源 ES 集群中创建 snapshot

调用 snapshot api 创建快照以备份索引数据，创建快照时可以指定只对部分索引进行备份，也可以备份所有的索引，具体的 api 接口参数可以查阅 官方文档。

备份所有索引

将源 ES 集群中的所有索引备份到 my_cos_backup 仓库下，并命名为 snapshot_1 ：

```
PUT _snapshot/my_cos_backup/snapshot_1
```

这个命令会立刻返回，并在后台异步执行直到结束。如果希望创建快照命令阻塞执行，可以添加 wait_for_completion 参数：

```
PUT _snapshot/my_cos_backup/snapshot_1?wait_for_completion=true
```

> 命令执行的时间与索引大小相关。

备份指定索引

您可以在创建快照的时候指定要备份的索引：

```
PUT _snapshot/my_cos_backup/snapshot_2
{
  "indices": "index_1,index_2"
}
```

> 参数 indices 的值为多个索引的时候，需要用，隔开且不能有空格。

查看快照状态

通过以下命令检查快照是否备份完成，返回结果中的 state 字段为 SUCCESS 则说明快照已经备份成功：

```
GET _snapshot/my_cos_backup/snapshot_1
```

在目标 ES 集群中创建 repository

在目标 ES 集群中创建仓库和在源 ES 集群中创建仓库完全相同。

从快照恢复

将快照中备份的所有索引都恢复到 ES 集群中：

```
POST _snapshot/my_cos_backup/snapshot_1/_restore
```

如果 snapshot_1 包括5个索引，则这5个索引都会被恢复到 ES 集群中。您还可以使用附加的选项对索引进行重命名。该选项允许您通过模式匹配索引名称，并通过恢复进程提供一个新名称。如果您想在不替换现有数据的前提下，恢复旧数据来验证内容或进行其他操作，则可以使用该选项。从快照里恢复单个索引并提供一个替换的名称：

```
POST /_snapshot/my_cos_backup/snapshot_1/_restore
{
  "indices": "index_1",
  "rename_pattern": "index_(.+)",
  "rename_replacement": "restored_index_$1"
```

```
}
```

- indices：只恢复 index_1 索引，忽略快照中存在的其他索引。
- rename_pattern：查找所提供的模式能匹配上的正在恢复的索引。
- rename_replacement：将匹配的索引重命名成替代的模式。

查看索引恢复状态

您可以通过调用 _recovery API，查看指定索引恢复的进度：

```
GET index_1/_recovery
```

另外可以通过调用以下 API，查看指定索引的状态，返回结果中 status 为 green 则说明索引已经完全恢复：

```
GET _cluster/health/index_1
```

logstash

logstash 支持从一个 ES 集群中读取数据然后写入到另一个 ES 集群，因此可以使用 logstash 进行数据迁移，使用 logstash 进行迁移前，需要注意以下几点：

- 需要在和TCloudFinanceZone上的 ES 集群相同的 VPC 下创建 CVM，部署 logstash，同时保证该 CVM 能够访问到源 ES 集群。
- 用于部署 logstash 的 CVM 最好选择比较高的配置，例如 CPU 为16核，内存为32GB。
- logstash 应该和目标 ES 集群的主版本号相同，例如目标 ES 集群为6.8.2版本，则 logstash 也需要使用6.8版本。
- 需要特别注意索引 type 的问题，因为 ES 的不同版本对索引 type 的约束不同，跨大版本迁移 ES 集群时可能出现因为索引的 type 而导致写入目标集群失败等的问题。具体可参考 logstash-output-elasticsearch 插件中对 document_type> document_type 参数的说明。
  一个常用的使用 logstash 进行跨集群数据迁移的配置文件如下：

```
input {
  elasticsearch {
    hosts => "1.1.1.1:9200"
    index => "*"
    docinfo => true
    size => 5000
    scroll => "5m"
  }
}
```

```
output {
  elasticsearch {
    hosts => ["http://2.2.2.2:9200"]
    user => "elastic"
    password => "your_password"
    index => "%{[@metadata][_index]}"
    document_type => "%{[@metadata][_type]}"
    document_id => "%{[@metadata][_id]}"
  }
}
```

上述配置文件将源 ES 集群的所有索引同步到目标集群中，同时也可以设置只同步指定的索引，利用 logstash 进行迁移的更多功能可查阅 logstash-input-elasticsearch 和 logstash-output-elasticsearch。

elasticsearch-dump

elasticsearch-dump 是一款开源的 ES 数据迁移工具，github 地址。

1. 安装 elasticsearch-dump。
   elasticsearch-dump 使用 node.js 开发，可使用 npm 包管理工具直接安装：

   ```
   npm install elasticdump -g
   ```

2. 主要参数说明。

   --input: 源地址，可为 ES 集群 URL、文件或 stdin,可指定索引，格式为：{protocol}://{host}:{port}/{index}
   --input-index: 源 ES 集群中的索引
   --output: 目标地址，可为 ES 集群地址 URL、文件或 stdout，可指定索引，格式为：{protocol}://{host}:{port}/{index}
   --output-index: 目标 ES 集群的索引
   --type: 迁移类型，默认为 data，表明只迁移数据，可选 settings, analyzer, data, mapping, alias

3. 迁移单个索引。
   以下操作通过 elasticdump 命令将集群172.16.0.39中的 companydatabase 索引迁移至集群172.16.0.20。

   > 第一条命令先将索引的 settings 先迁移，如果直接迁移 mapping 或者 data 将失去原有集群中索引的配置信息如分片数量和副本数量等，当然也可以直接在目标集群中将索引创建完毕后再同步 mapping 与 data。

   ```
   elasticdump --input=http://172.16.0.39:9200/companydatabase --output=http://172.16.0.20:920
   ```

```
0/companydatabase --type=settings
elasticdump --input=http://172.16.0.39:9200/companydatabase --output=http://172.16.0.20:920
0/companydatabase --type=mapping
elasticdump --input=http://172.16.0.39:9200/companydatabase --output=http://172.16.0.20:920
0/companydatabase --type=data
```

4. 迁移所有索引。
   以下操作通过 elasticdump 命令将集群172.16.0.39中的所有索引迁移至集群172.16.0.20。

   > 此操作并不能迁移索引的配置，例如分片数量和副本数量，必须对每个索引单独进行配置的迁移，或者直接在目标集群中将索引创建完毕后再迁移数据。

```
elasticdump --input=http://172.16.0.39:9200 --output=http://172.16.0.20:9200
```

总结

1. elasticsearch-dump 和 logstash 做跨集群数据迁移时，都要求用于执行迁移任务的机器可以同时访问到两个集群，因为网络无法连通的情况下就无法实现迁移。而使用 snapshot 的方式则没有这个限制，因为 snapshot 方式是完全离线的。因此 elasticsearch-dump 和 logstash 迁移方式更适合于源 ES 集群和目标 ES 集群处于同一网络的情况下进行迁移。而需要跨云厂商的迁移，可以选择使用 snapshot 的方式进行迁移，例如从阿里云 ES 集群迁移至TCloudFinanceZone ES 集群，也可以通过打通网络实现集群互通，但是成本较高。
2. elasticsearch-dump 工具和 MySQL 数据库用于做数据备份的工具 mysqldump 类似，都是逻辑备份，需要将数据一条一条导出后再执行导入，所以适合数据量小的场景下进行迁移。
3. snapshot 的方式适合数据量大的场景下进行迁移。

数据接入 ES

Elasticsearch 服务提供在用户 VPC 内通过私有网络 VIP 访问集群的方式，用户可通过 Elasticsearch REST Client 编写代码访问集群并将自己的数据导入到集群中，也可以通过官方提供的组件（如 logstash 和 beats）接入自己的数据。

本文以官方提供的组件 logstash 和 beats 为例，介绍不同类型的数据源接入 ES 的方式。

准备工作

因访问 ES 集群需要在用户 VPC 内进行，因此用户需要创建一台和 ES 集群相同 VPC 下的 CVM 实例或者 Docker 集群。

使用 logstash 接入 ES 集群

CVM 中访问 ES 集群

1. 安装部署 logstash 与 java8。

   ```
   wget https://artifacts.elastic.co/downloads/logstash/logstash-5.6.4.tar.gz
   tar xvf logstash-5.6.4.tar.gz
   yum install java-1.8.0-openjdk  java-1.8.0-openjdk-devel -y
   ```
   > 请注意 logstash 版本，建议与 Elasticsearch 版本保持一致。

2. 根据数据源类型自定义配置文件 *.conf ，配置文件内容可参考数据源配置文件说明。

3. 执行 logstash。

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

Docker 中访问 ES 集群

自建 Docker 集群

1. 拉取 logstash 官方镜像。

   ```
   docker pull docker.elastic.co/logstash/logstash:5.6.9
   ```

2. 根据数据源类型自定义配置文件 *.conf ，放置在 /usr/share/logstash/pipeline/ 目录下，目录可自定义。

3. 运行 logstash。

```
docker run --rm -it -v ~/pipeline/:/usr/share/logstash/pipeline/ docker.elastic.co/logstash/logstash:5.6.9
```

使用TCloudFinanceZone容器服务

TCloudFinanceZone Docker 集群运行于 CVM 实例上，所以需要先在容器服务控制台上创建 CVM 集群。

1. 登录 容器服务控制台，选择左侧菜单栏【集群】>【新建】创建集群。

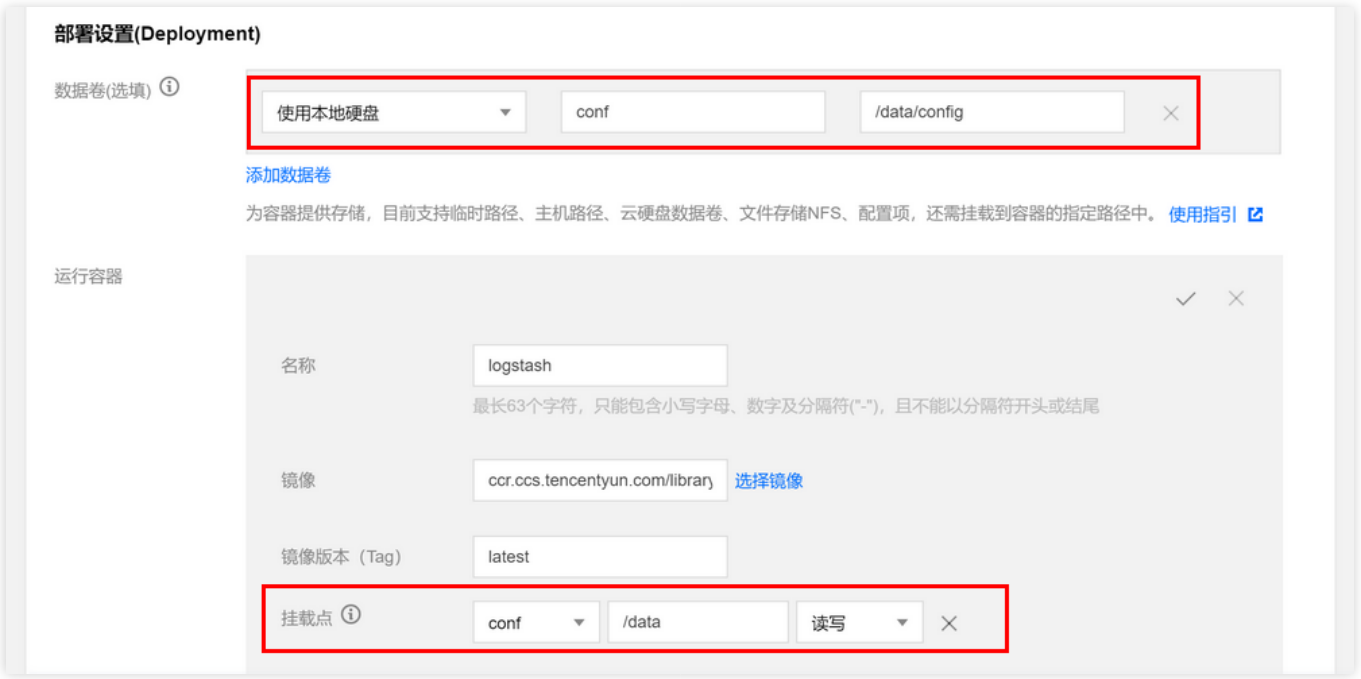
2. 选择左侧菜单栏【服务】，单击【新建】创建服务。

3. 选取 logstash 镜像。

   本例中使用 Hub 镜像仓库提供的 logstash 镜像，用户也可以自行创建 logstash 镜像。

4. 创建数据卷。

   创建存放 logstash 配置文件的数据卷，本例中在 CVM 的 /data/config 目录下添加了名为 logstash.conf 的配置文件，并将其挂在到 Docker 的 /data 目录下，从而使得容器启动时可以读取到 logstash.conf 文件。



5. 配置运行参数。

6. 根据需要配置服务参数并创建服务。

配置文件说明

File 数据源

```
input {
  file {
    path => "/var/log/nginx/access.log" # 文件路径
    }
}
filter {
}
output {
  elasticsearch {
    hosts => ["http://172.16.0.89:9200"] # Elasticsearch 集群的内网 VIP 地址和端口
    index => "nginx_access-%{+YYYY.MM.dd}" # 自定义索引名称，以日期为后缀，每天生成一个索引
  }
}
```

更多有关 File 数据源的接入，请参见官方文档 file input plugin。

Kafka 数据源

```
input{
    kafka{
      bootstrap_servers => ["172.16.16.22:9092"]
      client_id => "test"
      group_id => "test"
      auto_offset_reset => "latest" #从最新的偏移量开始消费
      consumer_threads => 5
      decorate_events => true #此属性会将当前 topic、offset、group、partition 等信息也带到 message 中
      topics => ["test1","test2"] #数组类型，可配置多个 topic
      type => "test" #数据源标记字段
    }
}

output {
  elasticsearch {
    hosts => ["http://172.16.0.89:9200"] # Elasticsearch 集群的内网 VIP 地址和端口
```

```
    index => "test_kafka"
  }
}
```

更多有关 kafka 数据源的接入，请参见官方文档 kafka input plugin。

JDBC 连接的数据库数据源

```
input {
  jdbc {
    # mysql 数据库地址
    jdbc_connection_string => "jdbc:mysql://172.16.32.14:3306/test"
    # 用户名和密码
    jdbc_user => "root"
    jdbc_password => "Elastic123"
    # 驱动 jar 包，如果自行安装部署 logstash 需要下载该 jar，logstash 默认不提供
    jdbc_driver_library => "/usr/local/services/logstash-5.6.4/lib/mysql-connector-java-5.1.40.jar"
    # 驱动类名
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_paging_enabled => "true"
    jdbc_page_size => "50000"
    # 执行的sql 文件路径+名称
    #statement_filepath => "test.sql"
    # 执行的sql语句
    statement => "select * from test_es"
    # 设置监听间隔  各字段含义（由左至右）分、时、天、月、年，全部为*默认含义为每分钟都更新
    schedule => "* * * * *"
    type => "jdbc"
  }
}

output {
  elasticsearch {
    hosts => ["http://172.16.0.30:9200"]
    index => "test_mysql"
    document_id => "%{id}"
  }
}
```

更多有关 JDBC 数据源的接入，请参见官方文档 jdbc input plugin。

使用 Beats 接入 ES 集群

Beats 包含多种单一用途的采集器，这些采集器比较轻量，可以部署并运行在服务器中收集日志、监控等数据，相对 logstashBeats 占用系统资源较少。

Beats 包含用于收集文件类型数据的 FileBeat、收集监控指标数据的 MetricBeat、收集网络包数据的 PacketBeat 等，用户也可以基于官方的 libbeat 库根据自己的需求开发自己的 Beats 组件。

CVM 中访问 ES 集群

1. 安装部署 filebeat。

   ```
   wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-5.6.4-linux-x86_64.tar.gz
   tar xvf filebeat-5.6.4.tar.gz
   ```

2. 配置 filebeat.yml。

3. 执行 filebeat。

   ```
   nohup ./filebeat 2>&1 >/dev/null &
   ```

Docker 中访问 ES 集群

自建 Docker 集群

1. 拉取 filebeat 官方镜像。

   ```
   docker pull docker.elastic.co/beats/filebeat:5.6.9
   ```

2. 根据数据源类型自定义配置文件 *.conf ，放置在 /usr/share/logstash/pipeline/ 目录下，目录可自定义。

3. 运行 filebeat。

   ```
   docker run docker.elastic.co/beats/filebeat:5.6.9
   ```

使用TCloudFinanceZone容器服务

使用TCloudFinanceZone容器服务部署 filebeat 的方式和部署 logstash 类似，镜像可以使用TCloudFinanceZone官方提供的 filebeat 镜像。

配置文件说明

配置 filebeat.yml 文件，内容如下：

```
// 输入源配置
filebeat.prospectors:
- input_type: log
    paths:
    - /usr/local/services/testlogs/*.log

// 输出到 ES
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["172.16.0.39:9200"]
```

MySQL 数据实时同步到 ES

当需要把 MySQL 的数据实时同步到 ES 时，为了实现低延迟的检索到 ES 中的数据或者进行其它数据分析处理。本文给出以同步 mysql binlog 的方式实时同步数据到 ES 的思路，实践并验证该方式的可行性，以供参考。

mysql binlog 日志

MySQL 的 binlog 日志主要用于数据库的主从复制和数据恢复。binlog 中记录了数据的增删改查操作，主从复制过程中，主库向从库同步 binlog 日志，从库对 binlog 日志中的事件进行重放，从而实现主从同步。

mysql binlog 日志有三种模式，分别为：

- ROW：记录每一行数据被修改的情况，但是日志量太大。
- STATEMENT：记录每一条修改数据的 SQL 语句，减少了日志量，但是 SQL 语句使用函数或触发器时容易出现主从不一致。
- MIXED：结合了 ROW 和 STATEMENT 的优点，根据具体执行数据操作的 SQL 语句选择使用 ROW 或者 STATEMENT 记录日志。

  要通过 mysql binlog 将数据同步到 ES 集群，只能使用 ROW 模式，因为只有 ROW 模式才能知道 mysql 中的数据的修改内容。下文为以 UPDATE 操作为例，ROW 模式和 STATEMENT 模式的 binlog 日志内容。

- ROW 模式的 binlog 日志内容示例如下：

```
SET TIMESTAMP=1527917394/*!*/;
BEGIN
/*!*/;
# at 3751
#180602 13:29:54 server id 1  end_log_pos 3819 CRC32 0x8dabdf01  Table_map: `webservice`.`building` mapped to number 74
# at 3819
#180602 13:29:54 server id 1  end_log_pos 3949 CRC32 0x59a8ed85        Update_rows: table id 74 flags: STMT_END_F
BINLOG '
UisSWxMBAAAARAAAAOsOAAAAAEoAAAAAAAEACndlYnNlcnZpY2UACGJ1aWxkaW5nAAYIDwEPEREG
wACAAQAAAAHfq40=
UisSWx8BAAAAggAAAG0PAAAAAEoAAAAAAAEAAgAG///A1gcAAAAAAAALYnVpbGRpbmctMTAADwB3
UkRNbjNLYlV5d1k3ajVbD64WWw+uFsDWBwAAAAAAAAtidWlsZGluZy0xMAEPAHdSRE1uM0tiVX
```

```
l3
WTdqNVsPrhZbD64Whe2oWQ==
'/*!*/;
### UPDATE `webservice`.`building`
### WHERE
###   @1=2006 /* LONGINT meta=0 nullable=0 is_null=0 */
###   @2='building-10' /* VARSTRING(192) meta=192 nullable=0 is_null=0 */
###   @3=0 /* TINYINT meta=0 nullable=0 is_null=0 */
###   @4='wRDMn3KbUywY7j5' /* VARSTRING(384) meta=384 nullable=0 is_null=0 */
###   @5=1527754262 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
###   @6=1527754262 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
### SET
###   @1=2006 /* LONGINT meta=0 nullable=0 is_null=0 */
###   @2='building-10' /* VARSTRING(192) meta=192 nullable=0 is_null=0 */
###   @3=1 /* TINYINT meta=0 nullable=0 is_null=0 */
###   @4='wRDMn3KbUywY7j5' /* VARSTRING(384) meta=384 nullable=0 is_null=0 */
###   @5=1527754262 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
###   @6=1527754262 /* TIMESTAMP(0) meta=0 nullable=0 is_null=0 */
# at 3949
#180602 13:29:54 server id 1  end_log_pos 3980 CRC32 0x58226b8f  Xid = 182
COMMIT/*!*/;
```

- STATEMENT 模式下 binlog 日志内容示例如下：

```
SET TIMESTAMP=1527919329/*!*/;
update building set Status=1 where Id=2000
/*!*/;
# at 688
#180602 14:02:09 server id 1  end_log_pos 719 CRC32 0x4c550a7d    Xid = 200
COMMIT/*!*/;
```

从 ROW 模式和 STATEMENT 模式下 UPDATE 操作的日志内容可以看出，ROW 模式完整地记录了要修改的某行数据更新前以及更改后所有字段的值，而 STATEMENT 模式只记录了 UPDATE 操作的 SQL 语句。我们要将 MySQL 的数据实时同步到 ES，只能选择 ROW 模式的 binlog，获取并解析 binlog 日志的数据内容，执行 ES document api，将数据同步到 ES 集群中。

mysqldump 工具

mysqldump 是一个对 MySQL 数据库中的数据进行全量导出的一个工具。 mysqldump 的使用方式如下：

```
mysqldump -uelastic -p'Elastic_123' --host=172.16.32.5 -F webservice > dump.sql
```

上述命令表示从远程数据库 172.16.32.5:3306 中导出 database:webservice 的所有数据，写入到 dump.sql 文件中，指定 -F 参数表示在导出数据后重新生成一个新的 binlog 日志文件以记录后续的所有数据操作。 dump.sql 中的文件内容如下：

```
-- MySQL dump 10.13  Distrib 5.6.40, for Linux (x86_64)
--
-- Host: 172.16.32.5    Database: webservice
-- ------------------------------------------------------
-- Server version       5.5.5-10.1.9-MariaDBV1.0R012D002-20171127-1822

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `building`
--

DROP TABLE IF EXISTS `building`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `building` (
  `Id` bigint(20) unsigned NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `BuildingId` varchar(64) NOT NULL COMMENT '虚拟建筑Id',
  `Status` tinyint(4) NOT NULL DEFAULT '0' COMMENT '虚拟建筑状态：0、处理中；1、正常；-1，停止；-2，销毁中；-3，已销毁',
  `BuildingName` varchar(128) NOT NULL DEFAULT '' COMMENT '虚拟建筑名称',
  `CreateTime` timestamp NOT NULL DEFAULT '2017-12-03 16:00:00' COMMENT '创建时间',
  `UpdateTime` timestamp NOT NULL DEFAULT '2017-12-03 16:00:00' COMMENT '更新时间',
  PRIMARY KEY (`Id`),
  UNIQUE KEY `BuildingId` (`BuildingId`)
) ENGINE=InnoDB AUTO_INCREMENT=2010 DEFAULT CHARSET=utf8 COMMENT='虚拟建筑表';
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `building`
--

LOCK TABLES `building` WRITE;
```

```
/*!40000 ALTER TABLE `building` DISABLE KEYS */;
INSERT INTO `building` VALUES (2000,'building-2',0,'6YFcmntKrNBIeTA','2018-05-30 13:28:31','2018-0
5-30 13:28:31'),(2001,'building-4',0,'4rY8PcVUZB1vtrL','2018-05-30 13:28:34','2018-05-30 13:28:34'),(20
02,'building-5',0,'uyjHVUYrg9KeGqi','2018-05-30 13:28:37','2018-05-30 13:28:37'),(2003,'buildin
g-7',0,'DNhyEBO4XEkXpgW','2018-05-30 13:28:40','2018-05-30 13:28:40'),(2004,'building-1',0,'TmtYX6Z
C0RNB4Re','2018-05-30 13:28:43','2018-05-30 13:28:43'),(2005,'building-6',0,'t8YQcjeXefWpcyU','201
8-05-30 13:28:49','2018-05-30 13:28:49'),(2006,'building-10',0,'WozgBc2IchNyKyE','2018-05-30 13:28:5
5','2018-05-30 13:28:55'),(2007,'building-3',0,'yJk27cmLOVQLHf1','2018-05-30 13:28:58','2018-05-30 1
3:28:58'),(2008,'building-9',0,'RSbjotAh8tymfxs','2018-05-30 13:29:04','2018-05-30 13:29:04'),(2009,'buil
ding-8',0,'IBOMlhaXV6k226m','2018-05-30 13:29:31','2018-05-30 13:29:31');
/*!40000 ALTER TABLE `building` ENABLE KEYS */;
UNLOCK TABLES;

/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2018-06-02 14:23:51
```

从以上内容可以看出，mysqldump 导出的 sql 文件包含 create table、drop table 以及插入数据的 sql 语句，但是不包含 create database 建库语句。

使用 go-mysql-elasticsearch 开源工具同步数据到 ES

go-mysql-elasticsearch 是用于同步 MySQL 数据到 ES 集群的一个开源工具，项目 github 地址。

go-mysql-elasticsearch 的基本原理：如果是第一次启动该程序，首先使用 mysqldump 工具对源 MySQL 数据库进行一次全量同步，通过 elasticsearch client 执行操作写入数据到 ES；然后实现了一个 mysql client，作为 slave 连接到源 MySQL，源 MySQL 作为 master 会将所有数据的更新操作通过 binlog event 同步给 slave，通过解析 binlog event 就可以获取到数据的更新内容，写入到 ES。

另外，该工具还提供了操作统计的功能，每当有数据增删改操作时，会将对应操作的计数加1，程序启动时会开启一个 HTTP 服务，通过调用 HTTP 接口可以查看增删改操作的次数。

使用限制

1. mysql binlog 必须是 ROW 模式（TCloudFinanceZone DB for MySQL 产品默认开启）。

2. 要同步的 MySQL 数据表必须包含主键，否则直接忽略。如果数据表没有主键，UPDATE 和 DELETE 操作就会因为在 ES 中找不到对应的 document 而无法进行同步。

3. 不支持程序运行过程中修改表结构。

4. 要赋予用于连接 MySQL 的账户 RELOAD 权限、REPLICATION 权限。

```
GRANT REPLICATION SLAVE ON *.* TO 'elastic'@'172.16.32.44';
GRANT RELOAD ON *.* TO 'elastic'@'172.16.32.44';
```

使用方式

1. 安装 Go1.10+ 版本，可以直接安装最新版的 Go，然后设置 GOPATH 环境变量。

2. 执行命令 go get github.com/siddontang/go-mysql-elasticsearch 。

3. 执行命令 cd $GOPATH/src/github.com/siddontang/go-mysql-elasticsearch 。

4. 执行 make 进行编译，编译成功后 go-mysql-elasticsearch/bin 目录下会生成名为 go-mysql-elasticsearch 的可执行文件。

5. 执行命令 vi etc/river.toml 修改配置文件，同步 172.16.0.101:3306 数据库中的 webservice.building 表到 ES 集群 172.16.32.64:9200 的 building index（更详细的配置文件说明请参考 项目文档）。

```
# MySQL address, user and password
# user must have replication privilege in MySQL.
my_addr = "172.16.0.101:3306"
my_user = "bellen"
my_pass = "Elastic_123"
my_charset = "utf8"

# Set true when elasticsearch use https
#es_https = false
# Elasticsearch address
es_addr = "172.16.32.64:9200"
# Elasticsearch user and password, maybe set by shield, nginx, or x-pack
es_user = ""
es_pass = ""

# Path to store data, like master.info, if not set or empty,
# we must use this to support breakpoint resume syncing.
# TODO: support other storage, like etcd.
```

```
data_dir = "./var"

# Inner Http status address
stat_addr = "127.0.0.1:12800"

# pseudo server id like a slave
server_id = 1001

# mysql or mariadb
flavor = "mariadb"

# mysqldump execution path
# if not set or empty, ignore mysqldump.
mysqldump = "mysqldump"

# if we have no privilege to use mysqldump with --master-data,
# we must skip it.
#skip_master_data = false

# minimal items to be inserted in one bulk
bulk_size = 128

# force flush the pending requests if we don't have enough items >= bulk_size
flush_bulk_time = "200ms"

# Ignore table without primary key
skip_no_pk_table = false

# MySQL data source
[[source]]
schema = "webservice"
tables = ["building"]
[[rule]]
schema = "webservice"
table = "building"
index = "building"
type = "buildingtype"
```

6. 在 ES 集群中创建 building index，因为该工具并没有使用 ES 的 auto create index 功能，如果 index 不存在会报错 。

7. 执行命令 ./bin/go-mysql-elasticsearch -config=./etc/river.toml 。

8. 在控制台输出结果。

```
2018/06/02 16:13:21 INFO  create BinlogSyncer with config {1001 mariadb 172.16.0.101 3306
bellen   utf8 false false <nil> false false 0 0s 0s 0}
2018/06/02 16:13:21 INFO  run status http server 127.0.0.1:12800
2018/06/02 16:13:21 INFO  skip dump, use last binlog replication pos (mysql-bin.000001, 120)
or GTID %!s(<nil>)
2018/06/02 16:13:21 INFO  begin to sync binlog from position (mysql-bin.000001, 120)
2018/06/02 16:13:21 INFO  register slave for master server 172.16.0.101:3306
2018/06/02 16:13:21 INFO  start sync binlog at binlog file (mysql-bin.000001, 120)
2018/06/02 16:13:21 INFO  rotate to (mysql-bin.000001, 120)
2018/06/02 16:13:21 INFO  rotate binlog to (mysql-bin.000001, 120)
2018/06/02 16:13:21 INFO  save position (mysql-bin.000001, 120)
```

9. 测试：向 MySQL 中插入、修改、删除数据，都可以反映到 ES 中。

使用体验

- go-mysql-elasticsearch 完成了最基本的 MySQL 实时同步数据到 ES 的功能，业务如果需要更深层次的功能如允许运行中修改 MySQL 表结构，可以进行自行定制化开发。
- 异常处理不足，解析 binlog event 失败直接抛出异常。
- 据作者描述，该项目并没有被其应用于生产环境中，所以使用过程中建议通读源码，知其利弊。

使用 mypipe 同步数据到 ES 集群

mypipe 是一个 mysql binlog 同步工具，在设计之初是为了能够将 binlog event 发送到 kafka，根据业务的需要也可以将数据同步到任意的存储介质中。mypipe github 地址。

使用限制

1. mysql binlog 必须是 ROW 模式。

2. 要赋予用于连接 MySQL 的账户 REPLICATION 权限。

```
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'elastic'@'%' IDENTIFIED BY 'Elastic
_123'
```

3. mypipe 只是将 binlog 日志内容解析后编码成 Avro 格式推送到 kafka broker，并不是将数据推送到 kafka。如果需要同步到 ES 集群，可以从 kafka 消费数据后，再写入 ES。

4. 消费 kafka 中的消息（MySQL 的操作日志），需要对消息内容进行 Avro 解析，获取到对应的数据操作内容，进行下一步处理。mypipe 封装了一个 KafkaGenericMutationAvroConsumer 类，可以直接继承该类使用，

或者自行解析。

5. mypipe 只支持 binlog 同步，不支持存量数据同步，即 mypipe 程序启动后无法对 MySQL 中已经存在的数据进行同步。

使用方式

1. 执行命令 git clone https://github.com/mardambey/mypipe.git 。

2. 执行命令 ./sbt package 。

3. 配置 mypipe-runner/src/main/resources/application.conf 。

```
mypipe {

 # Avro schema repository client class name
 schema-repo-client = "mypipe.avro.schema.SchemaRepo"

 # consumers represent sources for mysql binary logs
 consumers {

  localhost {
   # database "host:port:user:pass" array
   source = "172.16.0.101:3306:elastic:Elastic_123"
  }
 }

 # data producers export data out (stdout, other stores, external services, etc.)
 producers {

  kafka-generic {
   class = "mypipe.kafka.producer.KafkaMutationGenericAvroProducer"
  }
 }

 # pipes join consumers and producers
 pipes {

  kafka-generic {
   enabled = true
   consumers = ["localhost"]
   producer {
    kafka-generic {
     metadata-brokers = "172.16.16.22:9092"
    }
```

```
    }
    binlog-position-repo {
      # saved to a file, this is the default if unspecified
      class = "mypipe.api.repo.ConfigurableFileBasedBinaryLogPositionRepository"
      config {
        file-prefix = "stdout-00"     # required if binlog-position-repo is specifiec
        data-dir = "/tmp/mypipe/data" # defaults to mypipe.data-dir if not present
      }
    }
  }
 }
}
```

4. 配置 mypipe-api/src/main/resources/reference.conf ，修改 include-event-condition 选项，指定需要同步的 database 和 table。

   ```
   include-event-condition = """ db == "webservice" && table =="building" """
   ```

5. 在 kafka broker 端创建 topic: webservice_building_generic ，默认情况下 mypipe 以 ${db}_${table}_generic 为 topic 名，向该 topic 发送数据。

6. 执行命令 ./sbt "project runner" "runMain mypipe.runner.PipeRunner" 。

7. 测试：向 mysql building 表中插入数据，写一个简单的 consumer 消费 mypipe 推送到 kafka 中的消息。

8. 消费到没有经过解析的数据如下：

   ```
   ConsumerRecord(topic=u'webservice_building_generic', partition=0, offset=2, timestamp=None, timestamp_type=None, key=None, value='\x00\x01\x00\x00\x14webservice\x10building\xcc\x01\x02\x91,\xae\xa3fc\x11\xe8\xa1\xaaRT\x00Z\xf9\xab\x00\x00\x04\x18BuildingName\x06xxx\x14BuildingId\nId-10\x00\x02\x04Id\xd4%\x00', checksum=128384379, serialized_key_size=-1, serialized_value_size=88)
   ```

使用体验

- mypipe 相比 go-mysql-elasticsearch 更成熟，支持运行时 ALTER TABLE，同时解析 binlog 异常发生时，可通过配置不同的策略处理异常。
- mypipe 不能同步存量数据，如果需要同步存量数据可通过其它方式先全量同步后，再使用 mypipe 进行增量同步。
- mypipe 只同步 binlog，若要同步数据到 ES，则需另行开发。

节点类型存储配置建议

TCloudFinanceZone Elasticsearch Service（ES），是分布式多节点形式的集群，每个节点均是有计算和存储两部分构成。如何根据业务的需求，选择合适的配置，我们根据实际运营经验，在此提供一些 ES 常见使用场景下，配置选择的建议。您可以根据业务需要进行参考，当然，最好的方法还是需要您在业务的实际使用过程中逐步去探索。依托TCloudFinanceZone ES 提供的弹性伸缩机制，在业务规模增大，性能遇到瓶颈的时候，您可以随时扩容，调整得到合适的集群规格。

存储容量评估

影响TCloudFinanceZone ES 服务存储容量的主要因素如下：

- 副本数量：副本有利于增加数据的可靠性，但同时会增加存储成本。默认和建议的副本数量为1，对于部分可以承受异常情况导致数据丢失的场景，可考虑设置副本数量为0。
- 数据膨胀：除原始数据外，ES 需要存储索引、列存数据等，在应用编码压缩等技术后，一般膨胀10%。
- 内部任务开销：ES 占用约**20%**的磁盘空间，用于 segment 合并、ES Translog、日志等。
- 操作系统预留：Linux 操作系统默认为 root 用户预留5%的磁盘空间，用于关键流程处理、系统恢复、防止磁盘碎片化问题等。
  因此，数据在 ES 中占用的实际空间可通过下面公式估算：

```
实际空间 = 源数据 * (1 + 副本数量) * (1 + 数据膨胀) / (1 - 内部任务开销) / (1 - 操作系统预留)
      ≈ 源数据 * (1 + 副本数量) * 1.45
```

为保证服务的稳定运行，建议至少预留50%的存储空间，因此建议申请的存储容量为：

```
存储容量 = 源数据 * (1 + 副本数量) * 1.45 * （1 + 预留空间）
      ≈ 源数据 * (1 + 副本数量) * 2.2
```

> ES 广泛应用于日志、站点检索、Metrics、APM 等场景，上述存储容量评估方法较为泛化，您可以根据自身需求高度优化 ES，降低存储成本。

计算资源评估

ES 的计算资源主要消耗在写入和查询过程，而不同业务场景在写入和查询方面的复杂度不同、比重不同，导致计算资源相比存储资源较难评估。但一般情况下，存储资源会较早成为瓶颈，因此建议您优先评估存储资源量，然后参考TCloudFinanceZone ES 节点类型初步选择计算资源，在测试过程中确认计算资源是否足够。下面也针对几种常见使用场景，介绍计算资源评估过程中的一些经验：

- 日志场景：日志属于典型的写多读少类场景，计算资源主要消耗在写入过程中。我们在日志场景的经验是：2核

8GB内存的资源最大可支持0.5w/s的写入能力，但注意不同业务场景可能有偏差。由于实例性能基本随计算资源总量呈线性扩容，您可以按实例资源总量估算写入能力。例如6核24GB内存的资源可支持1.5w/s的写入能力，40核160GB内存的资源可支持10w/s的写入能力。

- Metric 及 APM 等结构化数据场景：这也是写多读少类场景，但相比日志场景计算资源消耗较小，2核8GB内存的资源一般可支持1w/s的写入能力，您可参照日志场景线性扩展的方式，评估不同规格实例的实际写入能力。
- 站内搜索及应用搜索等搜索场景：此类为读多写少类场景，计算资源主要消耗在查询过程，由于查询复杂度在不同使用场景差别非常大，计算资源也最难评估，建议您结合存储资源初步选择计算资源，然后在测试过程中验证、调整。

实例类型选择及测试

在完成存储、计算资源评估后，您可以初步参考 TCloudFinanceZone ES 节点类型选择实例类型，这里包含节点规格和节点数量两方面。选择实例类型的常用建议如下：

- 我们建议您至少选择3个节点，避免 ES 实例出现脑裂问题，保证 ES 实例具有较高的节点故障容错能力。
- 如果您有非常大的存储容量需求，建议选择高规格的节点，避免大量低规格节点，这对大实例的性能、稳定性等有较大好处。例如如果您有40核160GB内存5TB存储容量需求，建议选择8核32GB内存1TB * 5节点的实例。同理，当您需要对实例扩容时，建议优先进行纵向扩容，把节点扩容到8核32GB或16核64GB的规格，然后再考虑横向扩容增加节点个数。
- 当完成实例类型的初步选择后，您可以使用真实数据进行测试，通过观察 CPU 使用率、写入指标（性能、拒绝率）、查询指标（QPS、拒绝率）等监控信息，进一步确认实例类型是否合适。另外，建议针对上述监控信息配置告警，方便在线上使用时，及时发现资源不足等问题。

分片数量评估

每个 ES 索引被分为多个分片，数据按哈希算法打散到不同的分片中。由于索引分片的数量影响读写性能、故障恢复速度，且通常无法轻松更改，需要提前考虑。这里给出配置分片数量的一些常用建议：

- 建议单个分片大小保持在10GB - 50GB之间，您可以据此初步确定 Index 的分片数量。分片不宜过大或过小：过大可能使 ES 的故障恢复速度变慢；过小可能导致非常多的分片，但因为每个分片使用一些数量的 CPU 和内存，从而导致读写性能、内存不足等问题。
- 在测试阶段，可以根据每个 Index 的实际大小、预期未来增长情况，适当调整分片数量。
- 当分片数量超过数据节点数量时，建议分片数量接近数据节点的整数倍，方便分片在所有数据节点均匀分布。
- 对于日志、Metric 等场景中，建议使用 ES 自带的 Rollover Index 功能，持续滚动产生新的 Index。方便在发现分片大小不合理时，通过该功能及时调整分片数量。
  例如， 假设实例有5个数据节点，Index 当前大小为150GB，预期半年后增长50%。如果我们控制每个单分片为30GB，则大约需要150GB * （1 + 50%） / 30 ≈ 7个分片，考虑到有两个数据节点支撑2/7的数据压力，节点间压力相对不均匀，我们把分片数量调整到10个。

默认索引模板说明和调整

默认模板说明

索引模板是预先定义好的在创建新索引时自动应用的模板，主要包括索引设置、映射和模板优先级等配置。TCloudFinanceZone ES 在集群创建时提供了一个默认的索引模板，您可以在 Kibana 界面的【Dev Tools】中通过命令 GET _template/default@template 查看这个模板。下面是默认模板及其中配置的一些说明，可以根据需求适当调整这些配置。

```
{
  "default@template": {
    "order": 1, // 模板优先级，数值越大优先级越高
    "index_patterns": [ // 模板应用的索引
      "*"
    ],
    "settings": {
      "index": {
        "max_result_window": "65536", // 最大查询窗口，如果查询的窗口超过该大小，会报 Result window is too large 错误，需要调大这个配置
        "routing": {
          "allocation": {
            "include": {
              "temperature": "hot"
            }
          }
        },
        "refresh_interval": "30s", // 索引刷新间隔，被索引的文档在该间隔后才能被查询到，如果对于查询实时性要求较高，可以适当调小该值，但是值过小将影响写入性能
        "unassigned": {
          "node_left": {
            "delayed_timeout": "5m"
          }
        },
        "translog": {
          "sync_interval": "5s", // translog 刷盘间隔，值过小将影响写入性能
          "durability": "async"
        },
        "number_of_replicas": "1" // 副本分片数
      }
    },
    "mappings": {
      "_default_": {
        "_all": {
```

```
        "enabled": false // 建议禁用，_all 字段会包含所有其他字段形成一个大字符串，会占用较多磁盘空间，也会影响写入性能
      },
      "dynamic_templates": [ // 动态模板
       {
        "message_full": { // 将名为 message_full 的字段动态映射为 text 和 keyword 类型
         "match": "message_full",
         "mapping": {
          "type": "text",
          "fields": {
           "keyword": {
            "type": "keyword",
            "ignore_above": 2048
           }
          }
         }
        }
       },
       {
        "message": { // 将名为 message 的字段动态映射为 text 类型
         "match": "message",
         "mapping": {
          "type": "text"
         }
        }
       },
       {
        "strings": { // 将 string 类型字段动态映射为 keyword 类型
         "match_mapping_type": "string",
         "mapping": {
          "type": "keyword"
         }
        }
       }
      ]
     }
    },
    "aliases": {}
  }
}
```

模板调整

您可以在 Kibana 界面的【Dev Tools】中通过命令 PUT _template/my_template 自定义自己的索引模板，并通过

设置模板优先级 order 的数值大于默认模板优先级来覆盖默认的索引模板中的配置。

> 索引模板仅在索引创建时应用，因此模板调整不会对已有的索引产生影响。

调整主分片个数

在 Elasticsearch 5.6.4版本和6.4.3版本中，默认的索引主分片个数为5个。对于数据规模较小、索引个数较多的场景，建议调小主分片个数，以减轻索引元数据对堆内存的压力。您可以参考下面模板调整主分片个数：

```
{
  "index_patterns" : ["*"],
    "order" : 2, // 请确保模板中 order 字段的值大于1
    "settings" : {
        "index": {
            "number_of_shards" : 1
        }
    }
}
```

调整字段类型

在默认模板中，我们将 string 类型字段动态映射为 keyword 类型，以防止对所有文本类型数据都进行全文索引。您可以根据业务需求，修改指定 string 类型字段为 text，使其可以全文索引：

```
{
  "index_patterns" : ["*"],
    "order" : 2, // 请确保模板中 order 字段的值大于1
    "mappings": {
      "properties": {
        "字段名": {
          "type":  "text"
        }
    }
  }
}
```

其他业务场景

例如，您希望让索引的文档在10s之后就能被搜索到，并应用于所有的 search-* 索引，那么您可以新建一个如下的模板：

```
{
    "index_patterns" : ["search-*"],
    "order" : 2, // 请确保模板中 order 字段的值大于1
    "settings" : {
```

```
        "index": {
            "refresh_interval": "10s"
        }
    }
}
```

监控告警配置建议

TCloudFinanceZone ES 不仅为运行中的 ES 集群提供了多项监控指标，用于监测集群的运行情况，还提供了一些关键指标的配置告警功能，帮助您及时发现集群问题并进行处理。具体使用方法参考 查看监控 和 配置告警。
本文为您介绍在使用 ES 集群过程中需要重点关注的一些指标及其告警建议配置：

指标	告警建议配置	详细说明
集群健康状态	统计周期1分钟，>=1，持续5个周期，每30分钟告警一次	集群健康状态取值为： • 0：绿色，表示集群所有主分片和副本分片都可用，集群处于最健康的状态。 • 1：黄色，表示所有的主分片均可用，但存在不可用副本分片。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，数据面临较高的丢失风险。 • 2：红色，表示至少一个主分片以及它的全部副本分片均不可用。集群处于红色状态意味着已有部分数据不可用，搜索只能返回部分数据，而分配到丢失分片上的请求会返回异常。 集群健康状态是集群当前运行情况的最直接体现，当集群处于黄色或红色状态时，应立即排查产生原因，并及时修复，防止数据丢失和服务不可用。
平均磁盘使用率	统计周期1分钟，>80%，持续5个周期，每30分钟告警一次	平均磁盘使用率表示集群各节点磁盘使用率的平均值。磁盘使用率过高会导致节点没有足够的磁盘空间容纳分配到该节点上的分片，从而导致创建索引，添加文档等基本操作执行失败。建议在平均磁盘使用率超过75%时及时清理数据或扩容集群。
平均 JVM 内存使用率	统计周期1分钟，>85%，持续5个周期，每30分钟告警一次	平均 JVM 内存使用率表示集群各节点 JVM 内存使用率的平均值。JVM 内存使用率过高会导致读写操作被拒绝，集群 GC 频繁，甚至出现 OOM 等问题。当发现 JVM 内存使用率超过阈值时，建议通过纵向扩容的方式提高集群节点的规格。
平均 CPU 使用率	统计周期1分钟，>90%，持续5个周期，每30分钟告警一次	平均 CPU 使用率表示集群各节点 CPU 使用率的平均值。该值过高会导致集群节点处理能力下降，甚至宕机。发现 CPU 过高时，应根据集群当前节点配置情况和业务情况，提高节点规格或降低业务请求量。
bulk 拒绝率	统计周期1分钟，>0%，持续1个周期，每30分钟告警一次	bulk 拒绝率表示单周期内集群执行 bulk 操作被拒绝次数占 bulk 总操作次数的百分比。当 bulk 拒绝率大于0%，即出现 bulk 拒绝时，说明集群已经达到了 bulk 操作处理能力的上限，或集群出现异常，应及时排除出现 bulk 拒绝的原因并及时解决，否则会影响业务的 bulk 操作，甚至出现数据丢失情况。
查询拒绝率	统计周期1分钟，>0%，持续1个周期，每30分钟告警一次	查询拒绝率表示单周期内集群执行查询操作被拒绝次数占查询总操作数的百分比。当查询拒绝率大于0%，即出现查询拒绝时，说明集群已经达到了查询操作处理能力的上限，或集群出现异常，因及时排查出现查询拒绝的原因并及时解决，否则会影响业务的查询操作。

常见问题
产品相关问题

什么是 Elasticsearch？

Elasticsearch 是分布式、可扩展、实时的搜索与数据分析引擎，建立在全文搜索引擎库 Apache Lucene™ 基础之上，支持从单个节点到上百个节点的任意扩展，并支持多达 PB 级别的结构化或者非结构化数据存储和查询。

ES 适用于哪些业务场景？

ES 完全保持了 Elasticsearch 在海量数据检索方面的特性，拥有全文检索、准实时搜索、结构化搜索等能力，广泛应用于日志分析、站内搜索等业务场景。使用TCloudFinanceZone ES，用户可以构建其它在用云产品（如 CVM、容器等）的日志服务系统，也可以集成搜索服务到现有的业务服务框架中。

什么是 Elasticsearch Service（ES）？

ES 是基于开源搜索引擎 Elasticsearch 构建的云端托管 Elasticsearch 集群服务，完全保持了 Elasticsearch 本身的开放、兼容和易用特性，并提供了充足的硬件资源，便捷的图形化创建和管理工具，以及技术支持和运维支持。

写入拒绝（Bulk Reject）问题

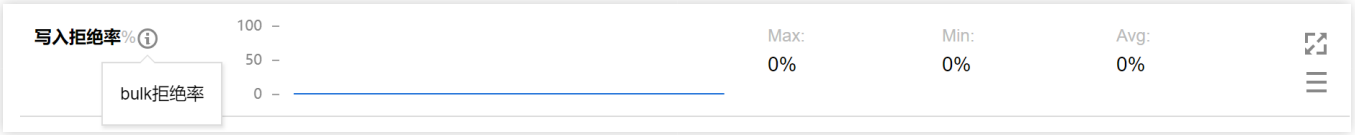
出现写入拒绝（Bulk Reject）问题如何解决？

问题现象

集群在某些情况下会出现写入拒绝率增大（bulk reject）的现象，具体表现为，bulk 写入的时候，会有以下类似报错：

> [2019-03-01 10:09:58][ERROR]rspItemError: {"reason":"rejected execution of org.elasticsearch.transport.TransportService$7@5436e129 on EsThreadPoolExecutor[bulk, queue capacity = 1024, org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@6bd77359[Running, pool size = 12, active threads = 12, queued tasks = 2390, completed tasks = 20018208656]]","type":"es_rejected_execution_exception"}

- 可在云监控看到集群写入拒绝率增大。



- 也可在 kibana 控制台，通过命令查看正在拒绝或者历史拒绝的个数。

```
GET _cat/thread_pool/bulk?s=queue:desc&v
```

一般默认队列是1024，如果 queue 下有1024，说明这个节点有 reject 的现象。

	node_name	name	active	queue	rejected
2	1536026850017169411	bulk	0	0	4813800
3	1536026850017169311	bulk	0	0	4731663
4	1536026850017169611	bulk	1	0	4539142
5	1536026850017169111	bulk	2	0	2812537
6	1528894127000000311	bulk	0	0	1208871
7	1536026850017169511	bulk	1	0	132380
8	1528894127000000611	bulk	3	0	100724

问题定位

引起 bulk reject 的大多原因是 shard 容量过大或 shard 分配不均，具体可通过以下方法进行定位分析。

1. 检查分片（shard）数据量是否过大 分片数据量过大，有可能引起 Bulk Reject，建议单个分片大小控制在20GB - 50GB左右。可在 kibana 控制台，通过命令查看索引各个分片的大小。

```
GET _cat/shards?index=index_name&v
```

```
GET _cat/shards?index=filebeat-6.5.1-2019.02.22&v
```

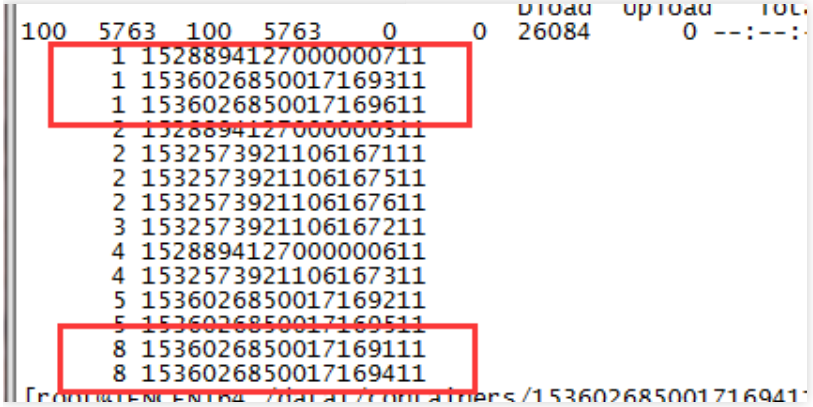
```
index                       shard prirep state      docs   store ip            node
filebeat-6.5.1-2019.02.22 2     r      STARTED 3862018 792.3mb 10.249.[redacted] node-2
filebeat-6.5.1-2019.02.22 2     p      STARTED 3862018 790.5mb 10.249.[redacted] node-1
filebeat-6.5.1-2019.02.22 1     p      STARTED 3858668 792.6mb 10.249.[redacted] node-3
filebeat-6.5.1-2019.02.22 1     r      STARTED 3858668 791.8mb 10.249.[redacted] node-1
filebeat-6.5.1-2019.02.22 0     p      STARTED 3857101 792.1mb 10.249.[redacted] node-2
filebeat-6.5.1-2019.02.22 0     r      STARTED 3857101 791.5mb 10.249.[redacted] node-3
```

2. 检查分片数是否分布不均匀 集群中的节点分片分布不均匀，有的节点分配的 shard 过多，有的分配的 shard 少。

- 可在 ES 控制台集群详情页的【集群监控】>【节点状态】查看，具体操作可参见 查看监控。
- 也可通过 curl 客户端，查看集群各个节点的分片个数。

  curl "$p:$port/_cat/shards?index={index_name}&s=node,store:desc" | awk '{print $8}' | sort | uniq -c | sort

结果如下图（第一列为分片个数，第二列为节点 ID），有的节点分片为1，有的为8，分布极不均匀。



解决方案

1. 设置分片大小。
   分片大小可以通过 index 模版下的 number_of_shards 参数进行配置（模板创建完成后，再次新创建索引时生效，老的索引不能调整）。
2. 调整分片数据不均匀

- 临时解决方案： 如果发现集群有分片分配不均的现象，可以通过设置 routing.allocation.total_shards_per_node 参数，动态调整某个 index 解决，参考地址。

  > 注意：
  >
  > total_shards_per_node 要留有一定的 buffer，防止机器故障导致分片无法分配（例如10台机器，索引有20个分片，则 total_shards_per_node 设置要大于2，可以取3）。

- 参考命令：

```
PUT {index_name}/_settings
 {
    "settings": {
      "index": {
        "routing": {
          "allocation": {
            "total_shards_per_node": "3"
          }
        }
      }
    }
}
```

- 索引生产前设置： 通过索引模板，设置其在每个节点上的分片个数。

```
PUT _template/ { template_name }
{
    "order": 0,
    "template": " { index_prefix@ } *",  //要调整的index前缀
    "settings": {
      "index": {
        "number_of_shards": "30",  //指定index分配的shard数，可以根据一个shard 30GB左右的空间来分配
        "routing.allocation.total_shards_per_node":3  //指定一个节点最多容纳的shards数
      }
    },
    "aliases": {}
  }
```

磁盘 readonly 问题

磁盘 readonly 的产生原因和解决办法是什么？

Elasticsearch 5.6.4版本没有磁盘容量阈值以防止磁盘被写满，因此，如果您使用 Elasticsearch 5.6.4版本请关注磁盘使用量情况，及时删除无用索引或扩容磁盘。

Elasticsearch 6.4.3版本设置了磁盘水位（watermark）阈值，当磁盘使用率到达 99% 时，在该磁盘上有分片的索引将变为 readonly 状态，此时无法向这些索引写入数据。因此，我们建议当您发现磁盘使用率较高的时候就及时地清理索引或扩容磁盘。

产生原因

1. 创建 index pattern 超时 当您的集群磁盘状态变为 readonly 时，将无法在 kibana 上创建 index pattern。这是因为当集群磁盘写满时，ES 会自动把索引设置的 block 级别设为 readonly_allow_delete 状态。

2. 写入索引出错 当您的集群磁盘状态变为 readonly 时，清理索引或扩容磁盘仍无法将数据写入索引，错误如下：

    ○ 索引只读错误：

```
Elasticsearch exception [
    type=cluster_block_exception,
    reason=blocked by: [FORBIDDEN/12/index read-only / allow delete (api)];
]
```

- 集群只读错误：

```
Elasticsearch exception [
    type=cluster_block_exception,
    reason=blocked by: [FORBIDDEN/13/cluster read-only / allow delete (api)];
]
```

解决办法

您需要先清理无用索引释放空间或扩容磁盘，然后在 Kibana 界面的【Dev Tools】中使用如下命令：

- 关闭索引只读状态：

```
PUT _all/_settings
```

```
{
"index.blocks.read_only_allow_delete": null
}
```

- 关闭集群只读状态：

```
PUT _cluster/settings
{
"persistent": {
"cluster.blocks.read_only_allow_delete": null
}
}
```

ES 自研熔断器及常见问题

Elasticsearch 提供了多种官方的熔断器（circuit breaker），用于防止内存使用过高导致 ES 集群因为 OutOfMemoryError 而出现问题。Elasticsearch 设置有各种类型的子熔断器，负责特定请求处理的内存限制。此外，还有一个父熔断器，用于限制所有子熔断器上使用的内存总量。

TCloudFinanceZone ES 自研熔断器的熔断机制是什么？

官方熔断机制的一个不足是仅跟踪那些经常会出问题的请求来预估内存的使用，而无法根据当前节点的实际内存使用状态，来限制请求的内存使用或触发熔断。在TCloudFinanceZone ES 中，开发了针对 JVM OLD 区内存使用率的自研熔断器来解决这个问题。

TCloudFinanceZone ES 的自研熔断器监控 JVM OLD 区的使用率，当使用率超过 85% 时开始拒绝写入请求，若 GC 仍无法回收 JVM OLD 区中的内存，在使用率到达 90% 时将拒绝查询请求。当请求被拒绝时，客户端将收到如下的响应：

```
{
    "status": 403,
    "error": {
        "root_cause": [{
            "reason": "pressure too high, (smooth) bulk request circuit break",
            "type": "status_exception"
        }],
        "type": "status_exception",
        "reason": "pressure too high, (smooth) bulk request circuit break"
    }
}
```

上面的错误提示表明当前 JVM OLD 区负载较高，需要清理 JVM 中部分内存后重试。

常用清理内存的方法是什么？

- 清理 fielddata cache：在 text 类型的字段上进行聚合和排序时会使用 fileddata 数据结构，可能占用较大内存。可以在 Kibana 界面的【Dev Tools】中使用如下命令查看索引的 fielddata 内存占用：

  ```
  GET /_cat/indices?v&h=index,fielddata.memory_size&s=fielddata.memory_size:desc
  ```

  若 fielddata 占用内存过高，可以在 Kibana 界面的【Dev Tools】中使用如下命令清理 fielddata：

  ```
  POST /${fielddata占用内存较高的索引}/_cache/clear?fielddata=true
  ```

- 清理 segment：每个 segment 的 FST 结构都会被加载到内存中，并且这些内存是不会被 GC 回收的。因此如

果索引的 segment 数量过大，也会导致内存使用率较高。可以在 Kibana 界面的【Dev Tools】中使用如下命令查看各节点的 segment 数量和占用内存大小：

```
GET /_cat/nodes?v&h=segments.count,segments.memory&s=segments.memory:desc
```

若 segment 占用内存过高，可以通过删除部分不用的索引，关闭索引，或定期合并不再更新的索引等方式缓解。

- 扩容集群：如果您清理内存后，仍频繁触发熔断，说明您的集群规模已经不匹配于您的业务负载，最好的方式是扩大集群规模，您可以参考 扩容集群。