Elasticsearch Service

实践教程

文档目录

实践教程
数据迁移和同步
数据迁移

最近更新时间：2025-04-18 16:22:32

用户在腾讯云上自建的 ES 集群或者在其它云厂商购买的 ES 集群，如果要迁移至腾讯云 ES（适用于大部分普通索引迁移），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：

- COS 快照
- logstash
- elasticsearch-dump

各种迁移方式的对比如下：

迁移方式	适用场景
COS 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

COS 快照

基于 COS 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：

> **注意：**
> 目标 ES 集群的版本号（如5.6.4中的5为主版本号）需要大于等于源 ES 集群的版本号，且不能超过1个大版本。例如：
> - 5.6.4集群创建的快照可以在 6.8.2 版本中恢复。
> - 6.8.3集群创建的快照不能在 6.8.2 版本中恢复。
> - 1.x 版本的集群创建的快照不能在 5.x 版本中恢复。
> - 5.x 版本的集群创建的快照不能在 7.x/8.x 版本中恢复。
> - 8.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。
- cos：快照存放于腾讯云 COS 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 集群迁移至腾讯云的 ES 集群，可以直接使用 COS 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 COS，然后在腾讯云上的 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腾讯云 COS 中，之后在腾讯云上的集群中进行恢复。

通过 COS 快照进行数据迁移时，需要先创建 COS 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：腾讯云账号 APPID。
- access_key_id：腾讯云 API 密钥 SecretId。
- access_key_secret：腾讯云 API 密钥 SecretKey。
- bucket：COS Bucket 名字，**不带 appId 后缀的 bucket 名**。
- region：COS Bucket 地域，必须与 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

> **说明**
> 恢复所有的索引数据，通常会有一些系统索引已经存在，那就会出现异常的情况。

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
```

> **注意**
> 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"
}
```

> **注意**
> 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：

- 需要在和腾讯云上的 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` 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"
        user => "elastic"
        password => "your_password"
        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 Node.js 和 npm，并配置好环境变量。
2. 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
```

3. 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
```

4. 如果集群有安全认证，可以参照下面的方法使用 reindex 集群鉴权。
   在对应的 HTTP 后面，添加 user:password@ 参考样例。

```
elasticsearch-dump --input = http://192.168.1.2:9200/my_index --
output = http://user:password@192.168.1.2:9200/my_index --type = data
```

如果密码包含特殊字符，需要使用在线工具（如 URL Encode ）对密码进行 URL 编码。例如，如果您的原始密码是 P@ssw0rd! ，经过 URL 编码后，它将变为 P%40ssw0rd%21。
如果是 HTTPS 集群，需要 -ca 加上各自集群 `server-certificates.pem` 证书，要在最前面加 `NODE_TLS_REJECT_UNAUTHORIZED=0`

```
NODE_TLS_REJECT_UNAUTHORIZED = 0  elasticsearch-dump --
input = https://elastic:P%40ssw0rd%21@192.168.1.2:9200/my_index -ca /path1/server-certificates.pem --
output = https://elastic:P%40ssw0rd%21@192.168.1.2:9200/my_index  -ca /path2/server-certificates.pem --
type = data
```

5. 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。

> **注意**
> 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://172.16.0.20:9200/companydatabase --type=settings
elasticdump --input=http://172.16.0.39:9200/companydatabase --
output=http://172.16.0.20:9200/companydatabase --type=mapping
elasticdump --input=http://172.16.0.39:9200/companydatabase --
output=http://172.16.0.20:9200/companydatabase --type=data
```


6. 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。

> **注意**
> 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，例如从友商 ES 集群迁移至腾讯云 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

最近更新时间：2024-10-15 21:58:41

腾讯云 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
```

> **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 .//root/logstash-5.6.4/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。

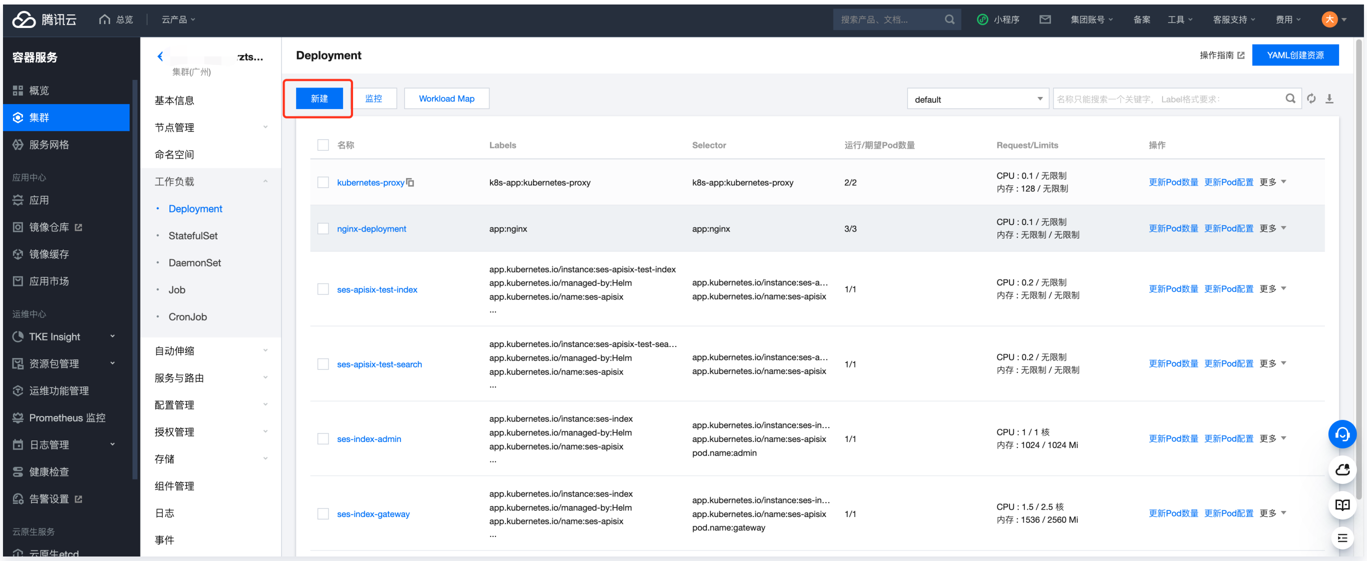
```
docker run --rm -it -v ~/pipeline/:/usr/share/logstash/pipeline/ docker.elastic.co/logstash/logstash:5.6.9
```

使用腾讯云容器服务

腾讯云 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 TencentHub 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 logstash，Beats 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-linux-x86_64.tar.gz
```

2. 配置 filebeat.yml。

   配置 filebeat.yml 参考示例：

```
################### Filebeat Configuration Example #########################

############################# Filebeat ######################################
filebeat:
  # List of prospectors to fetch data.
  prospectors:
    # Each - is a prospector. Below are the prospector specific configurations
    -
      # Paths that should be crawled and fetched. Glob based paths.
      # To fetch all ".log" files from a specific level of subdirectories
      # /var/log/*/*.log can be used.
      # For each file found under this path, a harvester is started.
      # Make sure not file is defined twice as this can lead to unexpected behaviour.
      # 指定要监控的日志，可以指定具体得文件或者目录
      paths:
        - /var/log/*.log  （这是默认的，可以自行修改，例如放在/home/hadoop/app.log里）
        #- c:\programdata\elasticsearch\logs\*

      # Configure the file encoding for reading files with international characters
      # following the W3C recommendation for HTML5 (http://www.w3.org/TR/encoding).
      # Some sample encodings:
      #   plain, utf-8, utf-16be-bom, utf-16be, utf-16le, big5, gb18030, gbk,
      #    hz-gb-2312, euc-kr, euc-jp, iso-2022-jp, shift-jis, ...
      # 指定被监控的文件的编码类型，使用plain和utf-8都是可以处理中文日志的
      #encoding: plain

      # Type of the files. Based on this the way the file is read is decided.
      # The different types cannot be mixed in one prospector
      #
      # Possible options are:
      # * log: Reads every line of the log file (default)
      # * stdin: Reads the standard in
      # 指定文件的输入类型log(默认)或者stdin
      input_type: log

      # Exclude lines. A list of regular expressions to match. It drops the lines that are
```

```
# matching any regular expression from the list. The include_lines is called before
# 在输入中排除符合正则表达式列表的那些行。
# exclude_lines. By default, no lines are dropped.
# exclude_lines: ["^DBG"]

# Include lines. A list of regular expressions to match. It exports the lines that are
# matching any regular expression from the list. The include_lines is called before
# exclude_lines. By default, all the lines are exported.
# 包含输入中符合正则表达式列表的那些行（默认包含所有行），include_lines执行完毕之后会执行exclude_lines
# include_lines: ["^ERR", "^WARN"]

# Exclude files. A list of regular expressions to match. Filebeat drops the files that
# are matching any regular expression from the list. By default, no files are dropped.
# 忽略掉符合正则表达式列表的文件
# exclude_files: [".gz$"]

# Optional additional fields. These field can be freely picked
# to add additional information to the crawled log files for filtering
# 向输出的每一条日志添加额外的信息，比如“level:debug”，方便后续对日志进行分组统计。
# 默认情况下，会在输出信息的fields子目录下以指定的新增fields建立子目录，例如fields.level
# 这个得意思就是会在es中多添加一个字段，格式为 "filelds":{"level":"debug"}
#fields:
#  level: debug
#  review: 1

# Set to true to store the additional fields as top level fields instead
# of under the "fields" sub-dictionary. In case of name conflicts with the
# fields added by Filebeat itself, the custom fields overwrite the default
# fields.
# 如果该选项设置为true，则新增fields成为顶级目录，而不是将其放在fields目录下。
# 自定义的field会覆盖filebeat默认的field
# 如果设置为true，则在es中新增的字段格式为："level":"debug"
#fields_under_root: false

# Ignore files which were modified more then the defined timespan in the past.
# In case all files on your system must be read you can set this value very large.
# Time strings like 2h (2 hours), 5m (5 minutes) can be used.
# 可以指定Filebeat忽略指定时间段以外修改的日志内容，比如2h（两个小时）或者5m(5分钟)。
#ignore_older: 0

# Close older closes the file handler for which were not modified
# for longer then close_older
# Time strings like 2h (2 hours), 5m (5 minutes) can be used.
# 如果一个文件在某个时间段内没有发生过更新，则关闭监控的文件handle。默认1h
#close_older: 1h

# Type to be published in the 'type' field. For Elasticsearch output,
# the type defines the document type these entries should be stored
# in. Default: log
# 设定Elasticsearch输出时的document的type字段 可以用来给日志进行分类。Default: log
#document_type: log

# Scan frequency in seconds.
# How often these files should be checked for changes. In case it is set
# to 0s, it is done as often as possible. Default: 10s
# Filebeat以多快的频率去prospector指定的目录下面检测文件更新（比如是否有新增文件）
# 如果设置为0s，则Filebeat会尽可能快地感知更新（占用的CPU会变高）。默认是10s
#scan_frequency: 10s

# Defines the buffer size every harvester uses when fetching the file
# 每个harvester监控文件时，使用的buffer的大小
#harvester_buffer_size: 16384
```



```
      # Maximum number of bytes a single log event can have
      # All bytes after max_bytes are discarded and not sent. The default is 10MB.
      # This is especially useful for multiline log messages which can get large.
      # 日志文件中增加一行算一个日志事件，max_bytes限制在一次日志事件中最多上传的字节数，多出的字节会被丢弃
      #max_bytes: 10485760

      # Mutiline can be used for log messages spanning multiple lines. This is common
      # for Java Stack Traces or C-Line Continuation
      # 适用于日志中每一条日志占据多行的情况，比如各种语言的报错信息调用栈
      #multiline:

        # The regexp Pattern that has to be matched. The example pattern matches all lines starting with [
        # 多行日志开始的那一行匹配的pattern
        #pattern: ^\[

        # Defines if the pattern set under pattern should be negated or not. Default is false.
        # 是否需要对pattern条件转置使用，不翻转设为true，反转设置为false。  【建议设置为true】
        #negate: false

        # Match can be set to "after" or "before". It is used to define if lines should be append to a
pattern
        # that was (not) matched before or after or as long as a pattern is not matched based on negate.
        # Note: After is the equivalent to previous and before is the equivalent to to next in Logstash
        # 匹配pattern后，与前面（before）还是后面（after）的内容合并为一条日志
        #match: after

        # The maximum number of lines that are combined to one event.
        # In case there are more the max_lines the additional lines are discarded.
        # Default is 500
        # 合并的最多行数（包含匹配pattern的那一行）
        #max_lines: 500

        # After the defined timeout, an multiline event is sent even if no new pattern was found to start a
new event
        # Default is 5s.
        # 到了timeout之后，即使没有匹配一个新的pattern（发生一个新的事件），也把已经匹配的日志事件发送出去
        #timeout: 5s

      # Setting tail_files to true means filebeat starts readding new files at the end
      # instead of the beginning. If this is used in combination with log rotation
      # this can mean that the first entries of a new file are skipped.
      # 如果设置为true，Filebeat从文件尾开始监控文件新增内容，把新增的每一行文件作为一个事件依次发送，
      # 而不是从文件开始处重新发送所有内容
      #tail_files: false

      # Backoff values define how agressively filebeat crawls new files for updates
      # The default values can be used in most cases. Backoff defines how long it is waited
      # to check a file again after EOF is reached. Default is 1s which means the file
      # is checked every second if new lines were added. This leads to a near real time crawling.
      # Every time a new line appears, backoff is reset to the initial value.
      # Filebeat检测到某个文件到了EOF（文件结尾）之后，每次等待多久再去检测文件是否有更新，默认为1s
      #backoff: 1s

      # Max backoff defines what the maximum backoff time is. After having backed off multiple times
      # from checking the files, the waiting time will never exceed max_backoff idenependent of the
      # backoff factor. Having it set to 10s means in the worst case a new line can be added to a log
      # file after having backed off multiple times, it takes a maximum of 10s to read the new line
      # Filebeat检测到某个文件到了EOF之后，等待检测文件更新的最大时间，默认是10秒
      #max_backoff: 10s

      # The backoff factor defines how fast the algorithm backs off. The bigger the backoff factor,
```

```
      # the faster the max_backoff value is reached. If this value is set to 1, no backoff will happen.
      # The backoff value will be multiplied each time with the backoff_factor until max_backoff is reached
      # 定义到达max_backoff的速度，默认因子是2，到达max_backoff后，变成每次等待max_backoff那么长的时间才backoff一次，
      # 直到文件有更新才会重置为backoff
      # 根据现在的默认配置是这样的，每隔1s检测一下文件变化，如果连续检测两次之后文件还没有变化，下一次检测间隔时间变为10s
      #backoff_factor: 2

      # This option closes a file, as soon as the file name changes.
      # This config option is recommended on windows only. Filebeat keeps the files it's reading open. This
can cause
      # issues when the file is removed, as the file will not be fully removed until also Filebeat closes
      # the reading. Filebeat closes the file handler after ignore_older. During this time no new file with
the
      # same name can be created. Turning this feature on the other hand can lead to loss of data
      # on rotate files. It can happen that after file rotation the beginning of the new
      # file is skipped, as the reading starts at the end. We recommend to leave this option on false
      # but lower the ignore_older value to release files faster.
      # 这个选项关闭一个文件,当文件名称的变化。#该配置选项建议只在windows
      #force_close_files: false

    # Additional prospector
    #-
      # Configuration to use stdin input
      #input_type: stdin

  # General filebeat configuration options
  #
  # Event count spool threshold - forces network flush if exceeded
  # spooler的大小，spooler中的事件数量超过这个阈值的时候会清空发送出去（不论是否到达超时时间）
  #spool_size: 2048

  # Enable async publisher pipeline in filebeat (Experimental!)
  # 是否采用异步发送模式（实验功能）
  #publish_async: false

  # Defines how often the spooler is flushed. After idle_timeout the spooler is
  # Flush even though spool_size is not reached.
  # spooler的超时时间，如果到了超时时间，spooler也会清空发送出去（不论是否到达容量的阈值）
  #idle_timeout: 5s

  # Name of the registry file. Per default it is put in the current working
  # directory. In case the working directory is changed after when running
  # filebeat again, indexing starts from the beginning again.
  # 记录filebeat处理日志文件的位置的文件，默认是在启动的根目录下
  #registry_file: .filebeat

  # Full Path to directory with additional prospector configuration files. Each file must end with .yml
  # These config files must have the full filebeat config part inside, but only
  # the prospector part is processed. All global options like spool_size are ignored.
  # The config_dir MUST point to a different directory then where the main filebeat config file is in.
  # 如果要在本配置文件中引入其他位置的配置文件，可以写在这里（需要写完整路径），但是只处理prospector的部分
  #config_dir:

###############################################################################
############################# Libbeat Config ##################################
# Base config file used by all other beats for using libbeat features

############################# Output ##########################################

# Configure what outputs to use when sending the data collected by the beat.
# Multiple outputs may be used.
output:
```

```
  ### Elasticsearch as output
  elasticsearch:                    （这是默认的，filebeat收集后放到es里，可以自行修改，例如，若要filebeat收集后，然后到redis，再到es，就可以注销这行）
    # Array of hosts to connect to.
    # Scheme and port can be left out and will be set to the default (http and 9200)
    # In case you specify and additional path, the scheme is required: http://localhost:9200/path
    # IPv6 addresses should always be defined as: https://[2001:db8::1]:9200
    hosts: ["localhost:9200"]         （这是默认的，filebeat收集后放到es里，可以自行修改，例如，若要filebeat收集后，然后到redis，再到es，就可以注销这行）
```

3. 执行 filebeat。

```
nohup ./filebeat-5.6.4-linux-x86_64/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
```

使用腾讯云容器服务

使用腾讯云容器服务部署 filebeat 的方式和部署 logstash 类似，镜像可以使用腾讯云官方提供的 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"]
```

使用 filebeat 采集 TKE 容器日志

最近更新时间：2024-10-15 21:58:41

背景

当业务使用腾讯云容器服务 TKE 进行部署时，可以通过 filebeat 来采集 TKE 中各个 pod 的日志，写入到下游的 Elasticsearch 集群中，然后在 kibana 上进行日志的查询与分析。本文介绍如何使用 filebeat daemonset 的方式采集容器中的日志。

实战过程

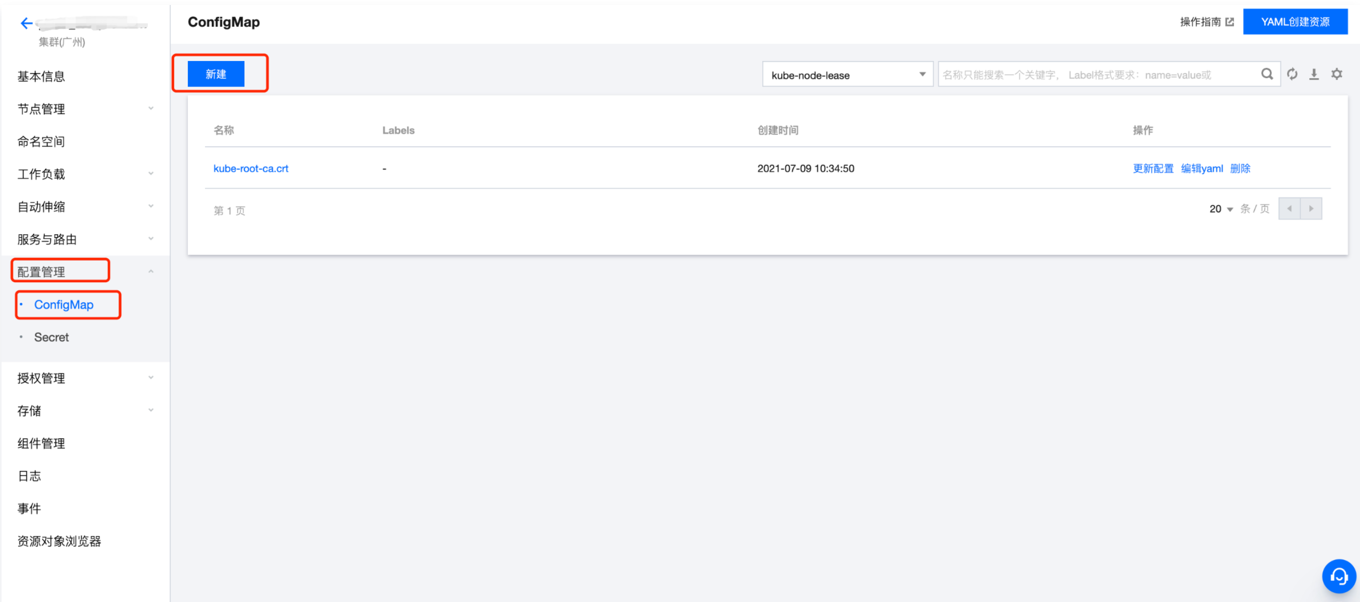
下面以采用运行 containerd 的 TKE 容器集群，以采集 nginx 日志为例，介绍使用 filebeat 采集 nginx pod 上的日志并写入到 ES。

部署 nginx daemonset

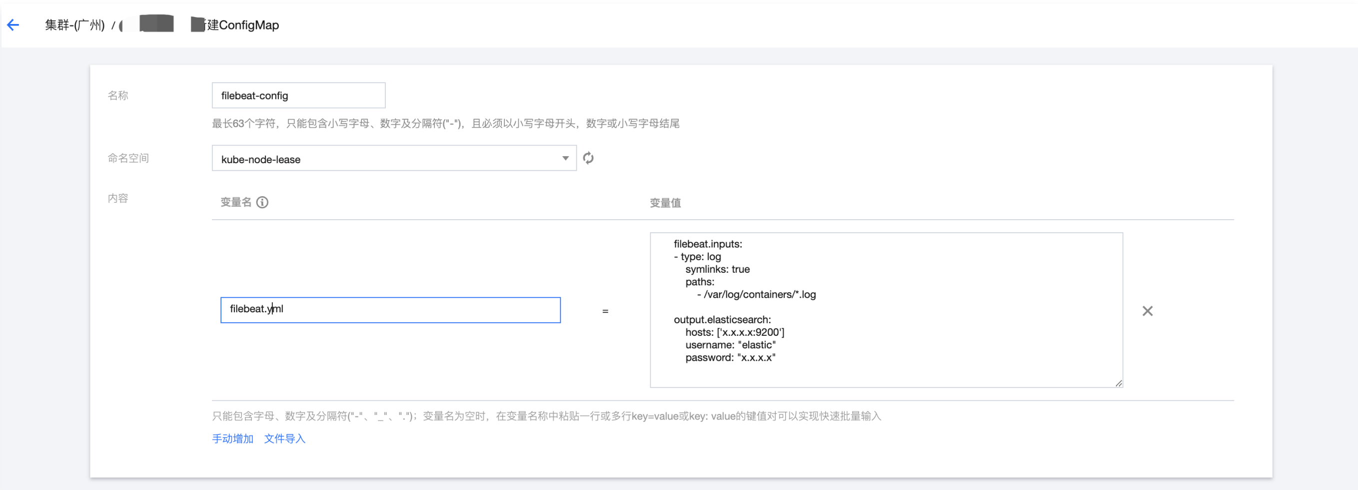
首先使用了一个 Docker Hub 中的 nginx 镜像，部署了一个名为 nginx 的 DaemonSet。详情请参考 创建简单的 Nginx 服务 。

部署 filebeat daemonset

1. 创建 ConfigMap
   1.1 filebeat 以 filebeat.yml 文件为主配置文件，首先创建一个 filebeat.yml 配置的 ConfigMap。
   1.2 进入**配置管理** > **ConfigMap** 页面，单击**新建**。



   1.3 新建的 ConfigMap 中的变量名为 filebeat.yml，内容为具体的配置信息，以下是一个最简单的 filebeat.yml 配置。



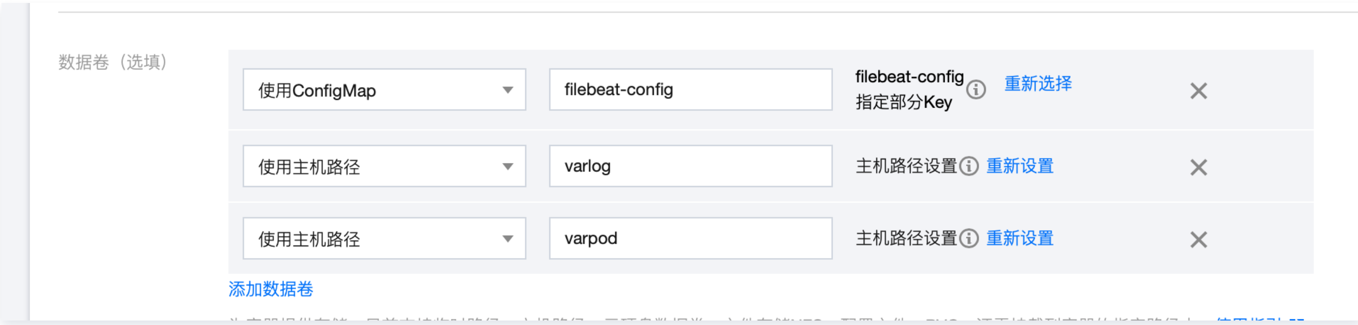

```
filebeat.inputs:
```

```
    - type: log
        symlinks: true
        paths:
            - /var/log/containers/*.log

    output.elasticsearch:
        hosts: ['x.x.x.x:9200']
        username: "elastic"
        password: "x.x.x.x"
```

2. 使用公有镜像库中的 filebeat 镜像，部署 filebeat daemonset。

3. 配置数据卷



4. 配置运行参数

5. 配置挂载点

说明

数据卷和挂载点说明：

1. 使用 ConfigMap 数据卷，使得 filebeat pod 可以读取到自定义的 filebeat.yml 配置。
2. 使用主机路径 `/var/log/containers`，使得 filebeat pod 可以读取到其它pod的日志，因为其它 pod 的日志都会打印在宿主机的 `/var/log/containers` 路径下。
3. 由于主机路径 `/var/log/containers` 下的 pod 日志，都是使用软链接，链接到 `/var/log/pods` 目录下的各个 pod 的日志文件，因此需要将主机路径 `/var/log/containers` 挂载到 filebeat pod 上。这就是在 filebeat.yml 中要定义 symlinks: true 的原因，因为默认情况下，filebeat 不会读取链接文件。

6. 在 kibana 中查看日志

进入到 filebeat.yml 中定义的 ES 集群对应的 kibana 中，查看对应索引是否生成，是否可以正常查看 nginx 日志。

通过以上步骤，即可查看 nginx 的日志可以正常被采集到。但是上述配置采集的是宿主机上所有 pod 的日志，有时需要只采集固定的某几个 pod 的日志，该怎么实现呢?

通过 YML 配置部署一个可以获取到 Pod 元信息的 filebeat daemonSet

在实际的业务场景中，通常需要通过 filebeat 采集部署在相同 host 上的多个 pod 的日志，通常也需要获取到采集到的 pod 的元信息，例如命令空间、pod 名称、标签等信息，以方便进行过滤或者检索。获取到 pod 的元信息需要调用 k8s 的 API, filebeat 内部也实现了这个功能，因此可以直接使用 filebeat 的 container input 以及 add_kubernetes_metadata processors 来实现。

1. 在 TKE 控制台上，单击 **YML 创建资源**按钮，直接使用如下 yml 配置创建 filebeat daemonSet。

```
---
        apiVersion: v1
        kind: ConfigMap
        metadata:
            name: filebeat-config
            namespace: default
            labels:
                k8s-app: filebeat
        data:
            filebeat.yml: |-
                filebeat.inputs:
                - type: container
                    paths:
                        - /var/log/containers/*.log
                    processors:
                        - add_kubernetes_metadata:
                                host: ${NODE_NAME}
                                matchers:
                                - logs_path:
                                        logs_path: "/var/log/containers/"

                processors:
                    - add_cloud_metadata:
                    - add_host_metadata:
                output.elasticsearch:
                    hosts: ['${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_PORT:9200}']
                    username: ${ELASTICSEARCH_USERNAME}
                    password: ${ELASTICSEARCH_PASSWORD}
        ---
        apiVersion: apps/v1
        kind: DaemonSet
        metadata:
            name: filebeat
            namespace: default
            labels:
                k8s-app: filebeat
        spec:
```

```
        selector:
            matchLabels:
                k8s-app: filebeat
        template:
            metadata:
                labels:
                    k8s-app: filebeat
            spec:
                serviceAccountName: filebeat
                terminationGracePeriodSeconds: 30
                containers:
                - name: filebeat
                    image: ccr.ccs.tencentyun.com/tke-market/filebeat:7.10.1
                    args: [
                        "-c", "/etc/filebeat.yml",
                        "-e",
                    ]
                    env:
                    - name: ELASTICSEARCH_HOST
                        value: x.x.x.x
                    - name: ELASTICSEARCH_PORT
                        value: "9200"
                    - name: ELASTICSEARCH_USERNAME
                        value: elastic
                    - name: ELASTICSEARCH_PASSWORD
                        value: Elastic123
                    - name: NODE_NAME
                        valueFrom:
                            fieldRef:
                                fieldPath: spec.nodeName
                    securityContext:
                        runAsUser: 0
                        # If using Red Hat OpenShift uncomment this:
                        #privileged: true
                    resources:
                        limits:
                            memory: 200Mi
                        requests:
                            cpu: 100m
                            memory: 100Mi
                    volumeMounts:
                    - name: config
                        mountPath: /etc/filebeat.yml
                        readOnly: true
                        subPath: filebeat.yml
                    - name: varlibdockercontainers
                        mountPath: /var/lib/docker/containers
                        readOnly: true
                    - name: varlog
                        mountPath: /var/log
                        readOnly: true
                    - name: varpods
                        mountPath: /var/log/pods
                        readOnly: true
                volumes:
                - name: config
                    configMap:
                        defaultMode: 0640
                        name: filebeat-config
                - name: varlibdockercontainers
                    hostPath:
                        path: /var/lib/docker/containers
```

```
            - name: varlog
              hostPath:
                path: /var/log
            - name: varpods
              hostPath:
                path: /var/log/pods
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: filebeat
subjects:
- kind: ServiceAccount
  name: filebeat
  namespace: default
roleRef:
  kind: ClusterRole
  name: filebeat
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: filebeat
  namespace: default
subjects:
  - kind: ServiceAccount
    name: filebeat
    namespace: default
roleRef:
  kind: Role
  name: filebeat
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: filebeat-kubeadm-config
  namespace: default
subjects:
  - kind: ServiceAccount
    name: filebeat
    namespace: default
roleRef:
  kind: Role
  name: filebeat-kubeadm-config
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: filebeat
  labels:
    k8s-app: filebeat
rules:
- apiGroups: [""] # "" indicates the core API group
  resources:
  - namespaces
  - pods
  - nodes
  verbs:
  - get
```

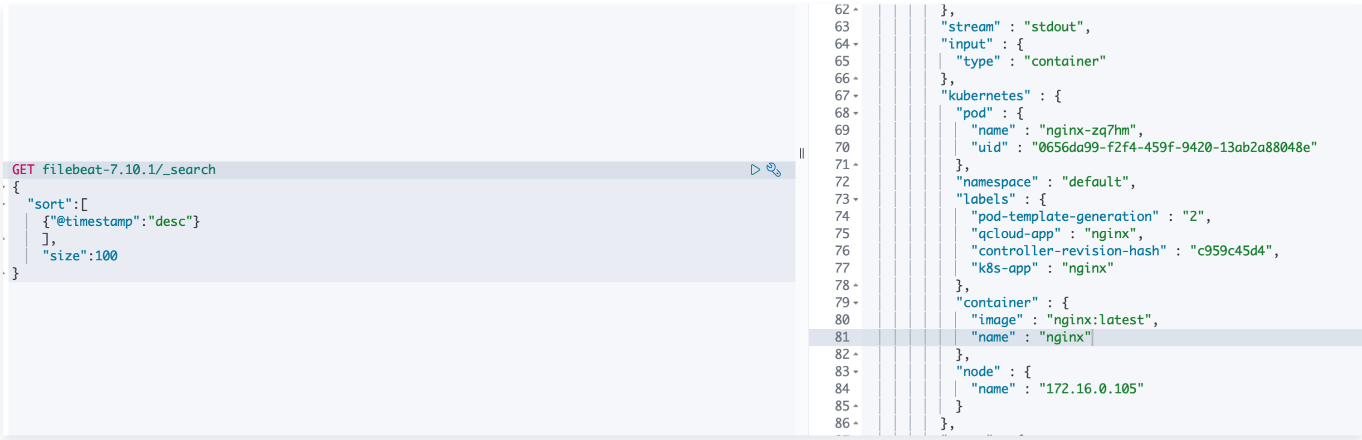
```
          - watch
          - list
      - apiGroups: ["apps"]
          resources:
              - replicasets
          verbs: ["get", "list", "watch"]
      ---
      apiVersion: rbac.authorization.k8s.io/v1
      kind: Role
      metadata:
          name: filebeat
          # should be the namespace where filebeat is running
          namespace: default
          labels:
              k8s-app: filebeat
      rules:
          - apiGroups:
                  - coordination.k8s.io
              resources:
                  - leases
              verbs: ["get", "create", "update"]
      ---
      apiVersion: rbac.authorization.k8s.io/v1
      kind: Role
      metadata:
          name: filebeat-kubeadm-config
          namespace: default
          labels:
              k8s-app: filebeat
      rules:
          - apiGroups: [""]
              resources:
                  - configmaps
              resourceNames:
                  - kubeadm-config
              verbs: ["get"]
      ---
      apiVersion: v1
      kind: ServiceAccount
      metadata:
          name: filebeat
          namespace: default
          labels:
              k8s-app: filebeat
---
```

说明

配置说明：

1. 命令空间为 default，根据需要也可以直接替换。
2. 创建了名为 filebeat 的服务账号，并且授予获取 pods 列表、获取 pod 详情等接口的权限，filebeat 会使用该账号获取到 pods 的元信息。
3. 通过 container input 采集 `/var/log/containers/` 目录下的日志， container input 可采集容器的 stdout 和 stderr。

2. 在 kibana 中查看日志，可以看到每条日志中都包含有 kubernetes 字段。



3. 上述配置通过 container input 直接采集到了 filebeat pod 所在的 node 上的所有 pod 的日志，当然，也包括了 filebeat 自身的日志，在真实的业务场景中，通常只需要采集业务关心的 pod 的日志即可。

- 方式一：通过在 filebeat.yml 中定义 processor，对采集到的所有日志 event 进行过滤，过滤掉不关心的日志 event 即可（例如通过 drop event processor 过滤掉某些不关心的 namespace 和 pod 的日志）；
- 方式二：通过 Autodiscover 定义新的 filebeat.yml 配置文件，通过定义模板只采集固定 pod 的日志，下面是一个简单的 Autodiscover 配置，该配置只采集容器名称为 nginx 的 pod 的日志：

```
filebeat.autodiscover:
            providers:
                - type: kubernetes
                    templates:
                        - condition:
                                equals:
                                    kubernetes.container.name: nginx
                            config:
                                - type: container
                                    paths:
                                        - /var/log/containers/${data.kubernetes.pod.name}_*.log

processors:
            - add_cloud_metadata:
            - add_host_metadata:
output.elasticsearch:
            hosts: ['${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_PORT:9200}']
            username: ${ELASTICSEARCH_USERNAME}
            password: ${ELASTICSEARCH_PASSWORD}
```

4. 通过修改名为 filebeat-config 的 ConfigMap，并重新部署 pod，使得新配置生效，在 kibana 中查看日志，可以看到只有 nginx pod 的日志被采集到了。

MySQL 数据同步到 ES 方案选型

最近更新时间：2024-10-15 21:58:41

当需要把 MySQL 中的数据同步到 Elasticsearch 时，根据应用场景、数据规模、同步延迟需求的不同，可以选择不同的同步方案。本文介绍了常见的同步方案以及适用的场景，帮助您选择合适的方案进行数据同步。

同步方案	基本原理	适用场景	使用限制	相关文档
使用 Oceanus 实现实时同步	使用腾讯云的流计算服务 Oceanus 实现实时同步 mysql 的数据到 Elasticsearch，通过解析 binlog 的方式实现实时同步	实时性要求高的场景	1. MySQL 的 binlog 必须是 ROW 模式（腾讯云 TencentDB for MySQL 产品默认开启） 2. 要同步的 MySQL 数据表必须包含主键 3. 不支持在同步过程中修改正在执行同步的 MySQL 表的结构 4. 要赋予用于连接 MySQL 的账号以 RELOAD 权限、REPLICATION 权限	使用腾讯云 Oceanus 实时同步 MySQL 中的数据到 Elasticsearch
Logstash 实现全量同步与增量同步	Logstash 的 input-jdbc 插件可以实现数据全量地从 MySQL 同步到 Elasticsearch，通过批量拉取 MySQL 表中的数据再批量写入到 ES 的方式实现全量同步； 另外，在 MySQL 表只有新增数据的情况下，可以在 Logstash 的配置中设置定时任务定期地查询最近一段时间内新增的数据实现简单的增量同步	需要进行全表数据导入到 ES 的场景，以及对实时性要求不高的场景	1. 增量同步时需要 MySQL 表中需要有自增字段或者有时间戳字段，例如 UpdateTime 2. 增量同步不能支持数据的删除 3. 增量同步的实时性较差	使用腾讯云 Logstash 同步 MySQL 中的数据到 Elasticsearch

使用流计算 Oceanus 实时同步 MySQL 数据到 ES

最近更新时间：2024-10-30 17:12:01

本文主要介绍了从 mysql 数据库采集数据到流计算服务 Oceanus 进行分析，最后输出到 ElasticSearch 服务的实践。本方案可作为日志搜索场景解决方案使用。使用了云数据库 MySQL、流计算 Oceanus、Elasticsearch、Kibana 和私有网络 VPC。

环境搭建

创建 Oceanus 集群

在 流计算 Oceanus 控制台 的**集群管理 > 新建集群**中新建集群，选择地域、可用区、VPC、日志、存储，设置初始密码等。

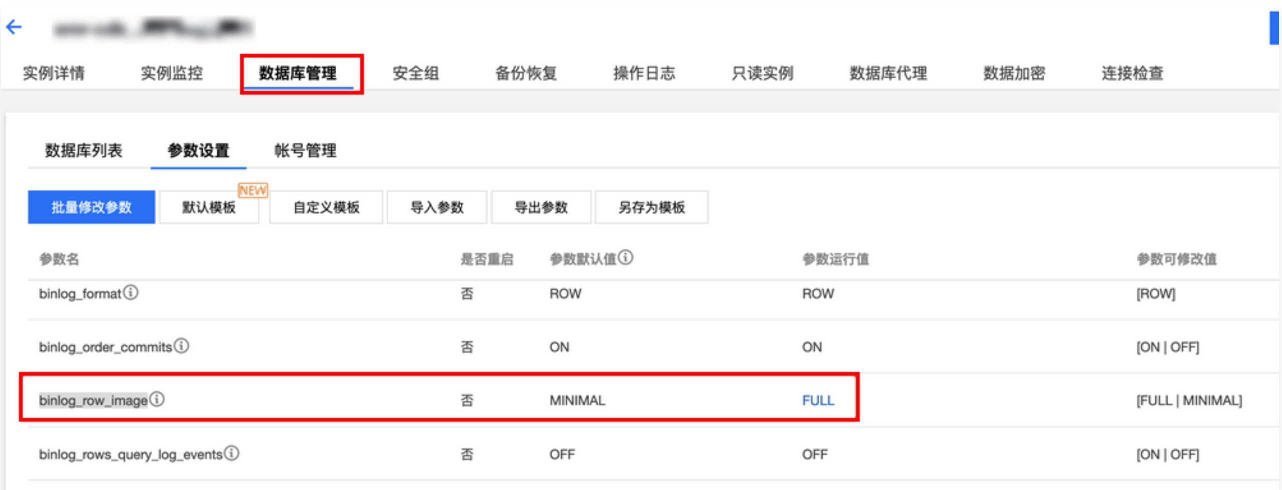
> **说明**
> - 若之前未曾使用过私有网络 VPC、日志 CLS、对象存储 COS 等组件，需要先进行创建。
> - 新建集群时，私有网络 VPC 及子网选择需和下文的 MySQL、ES 集群相同，否则需要手动打通（例如对等连接）。

创建完后的集群如下：

创建 MySQL 集群

在 云数据库 TencentDB 控制台 中，单击**新建**，创建 MySQL 集群。然后在**数据库管理 > 参数设置**中修改如下参数。

```
binlog_row_image=FULL
```



在 MySQL 数据库中创建表

执行如下 sql，或通过可视化页面创建数据库，创建表。

```
-- 创建数据库
create database test;
-- 以学生成绩表为例
CREATE TABLE `cdc_source4es` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '学号',
  `score` int(11) NOT NULL COMMENT '分数',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 COMMENT='create for student score'
```

并在表中插入几条数据。

```
insert into cdc_source4es values(1, 99);
insert into cdc_source4es values(2, 88);
insert into cdc_source4es values(3, 77);
```

创建 ElasticSearch 集群

在 Elasticsearch Service 控制台 中，单击**新建**，创建 Elasticsearch Service 集群。建议选择与 Oceanus 相同地域、可用区和网络。文本以 Elasticsearch 7.5.1 版本的集群为例。

集群创建完成后，可通过 Kibana 查看集群信息。如在 Dev Tools 面板上执行如下命令。

> **说明**
> Elasticsearch Service 中无需提前创建类似表的实体。

```
# 查看集群节点
GET _cat/nodes
```

```
# 返回节点信息则为正常
172.28.1.1 43 99 1 0.06 0.06 0.12 dilm - 1627027760001130832
```

```
172.28.1.2  65 99 3 0.03 0.12 0.13 dilm - 1627027760001130732
172.28.1.3 29 99 3 0.08 0.08 0.12 dilm * 1627027760001130632
```

作业创建

创建 SQL 作业

在 流计算 Oceanus 控制台 的**作业管理 > 新建作业**中新建 SQL 作业，选择在新建的集群中新建作业。然后在作业的**开发调试 > 作业参数**中添加必要的 connector，如 mysql-cdc connector、elasticsearch6/7 connector。

> **注意**
> ES connector 版本要与购买的 ES 组件版本一致。

创建 Source 端

选择 mysql 作为数据源，并将后续的数据持续更新到 ES 中。

```
-- mysql-cdc connector
CREATE TABLE `mysql_source` (
    `id` int,
    `score` int,
    PRIMARY KEY (`id`) NOT ENFORCED -- 如果要同步的数据库表定义了主键, 则这里也需要定义
) WITH (
    'connector' = 'mysql-cdc',       -- 必须为 'mysql-cdc'
    'hostname' = '172.28.28.213',    -- 数据库的 IP
    'port' = '3306',                 -- 数据库的访问端口
    'username' = 'youruser',         -- 数据库访问的用户名（需要提供 SHOW DATABASES, REPLICATION SLAVE, REPLICATION
CLIENT, SELECT, RELOAD 权限）
    'password' = 'yourpassword',     -- 数据库访问的密码
    'database-name' = 'test',    -- 需要同步的数据库
    'table-name' = 'cdc_source4es'       -- 需要同步的数据表名
);
```

创建 Sink 端

Sink 无需在 ES 集群中提前做初始化，可直接写入数据。

```
-- 注意! 如果您启用了 Elasticsearch 的用户名密码鉴权功能, 目前只能使用 Flink 1.10 的旧语法。若无需鉴权, 则可以使用 Flink
1.11 的新语法。
-- 参见 https://ci.apache.org/projects/flink/flink-docs-release-1.10/dev/table/connect.html#elasticsearch-
connector

CREATE TABLE es_old_sink (
```

```
    `id` INT,
    `score` INT
) WITH (
    'connector.type' = 'elasticsearch', -- 输出到 Elasticsearch
    'connector.version' = '7',          -- 指定 Elasticsearch 的版本，例如 '6'，'7'. 注意务必要和所选的内置
Connector 版本一致
    'connector.hosts' = 'http://172.28.1.175:9200',  -- Elasticsearch 的连接地址
    'connector.index' = 'connector-test-index',       -- Elasticsearch 的 Index 名
    'connector.document-type' = '_doc',  -- Elasticsearch 的 Document 类型
    'connector.username' = 'elastic',  -- 可选参数: Elasticsearch 用户名
    'connector.password' = 'yourpassword',  -- 可选参数: Elasticsearch 密码

    'update-mode' = 'upsert',            -- 可选无主键的 'append' 模式，或有主键的 'upsert' 模式
    'connector.key-delimiter' = '$',     -- 可选参数，复合主键的连接字符（默认是 _ 符号，例如 key1_key2_key3)
    'connector.key-null-literal' = 'n/a',  -- 主键为 null 时的替代字符串，默认是 'null'
    'connector.failure-handler' = 'retry-rejected',    -- 可选的错误处理。可选择 'fail' （抛出异常）、'ignore'（忽略
任何错误）、'retry-rejected'（重试）

    'connector.flush-on-checkpoint' = 'true',   -- 可选参数，快照时不允许批量写入（flush），默认为 true
    'connector.bulk-flush.max-actions' = '42',  -- 可选参数，每批次最多的条数
    'connector.bulk-flush.max-size' = '42 mb',  -- 可选参数，每批次的累计最大大小 （只支持 mb)
    'connector.bulk-flush.interval' = '60000',  -- 可选参数，批量写入的间隔 (ms)
    'connector.connection-max-retry-timeout' = '1000',     -- 每次请求的最大超时时间 (ms)
    --'connector.connection-path-prefix' = '/v1'          -- 可选字段，每次请求时附加的路径前缀

    'format.type' = 'json'         -- 输出数据格式，目前只支持 'json'
);
```

算子操作

下面语法中只做了简单的数据插入，没有进行复杂计算。

```
-- Flink SQL 进行运算
INSERT INTO es_old_sink select id, score  from mysql_source;
```

验证总结

在 Kibana 的 Dev Tools 中查询 ES 中的数据是否插入成功。

```
# 查询该索引下所有的数据
GET connector-test-index/_search
```

自建/其他云厂商集群迁移
自建 ES 集群在线融合迁移原理解析及操作指南

最近更新时间：2025-04-21 16:56:42

随着腾讯云 ES 集群稳定性越来越高、产品体验越来越好。有越来越多的外部客户希望将自建的 ES 集群迁移到腾讯云上来。本文将介绍一种腾讯云 ES 在业界独有的业务不停服无感知的迁移方案：在线融合迁移方案。本方案目前已经迁移了上百套客户自建 ES 集群上云。下面将结合我们在给客户迁移过程中总结出的宝贵经验，来详细介绍在线融合迁移技术方案的基本原理、核心优势、迁移步骤和注意事项。

为什么需要在线融合迁移

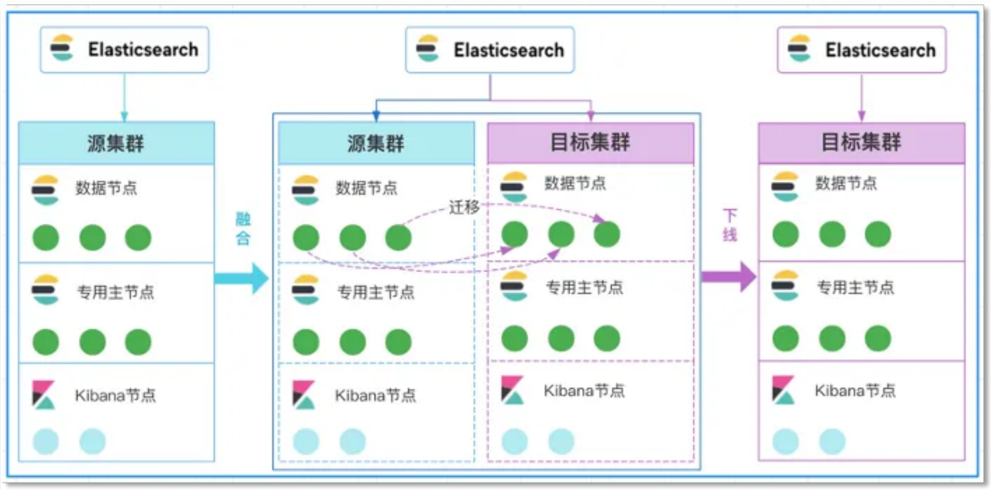
要回答这个问题，我们首先来看下当前业界常用的迁移方案，分别是 Logstash、Reindex、快照备份与恢复和跨集群复制 CCR，如下图所示。

迁移方案	Logstash	Reindex	快照备份与恢复	跨集群复制CCR
迁移原理	通过Logstash管道连通源集群和目标集群，从源集群中读取数据写入到目标集群中。	类似于Logstash迁移，批量读取源集群数据写入到目标集群中。	通过对源集群索引数据打快照，然后在目标集群中进行恢复的方式。	采用类似于MySQL的主从复制机制来实现数据迁移
迁移类型	离线	离线	离线	半在线
是否需要停服	是	是	是	短暂停服
适用场景	集群规模适中，索引个数较少的场景	集群规模较小，且对迁移速度要求不高的场景	集群规模适中，索引个数较多的场景	集群规模中等，对停服时间有严格要求的场景
痛点	需要先在目标集群中创建好索引模版，且需要逐个校验数据一致性，有丢数据风险	离线迁移，业务停服，且只能迁移小规模集群	源集群需要安装COS插件，且需要滚动重启集群	1、有版本限制，低版本不支持 2、切割时需停服且全量重启集群 3、写入压力大时，有性能问题

从图中我们可以看出以上四种迁移方案都有一个共同的特点，那就是都需要业务停服，即都属于离线迁移。离线迁移最大的痛点就是迁移期间业务需要停止写入，且迁移流程非常繁琐，工作量巨大。对于一些核心的业务集群，客户几乎不可能接受任何时刻的停服操作。因此对于负责迁移的同学来说就急需一种平滑的、业务无感知的、高可用的迁移技术方案。这便是腾讯云 ES 在线融合迁移技术方案要解决的问题。

在线融合迁移方案原理

在线融合迁移方案，最体现优势的两个字就是在线，相比于离线迁移，能够真正实现业务不停服、无感知；而最体现原理的两个字就是融合，通过将自建集群和云上集群这两个本身独立的集群融合为一个大集群，并结合ES集群自带的 分片分配、迁移特性 来完成数据的迁移工作。融合迁移示意图如下图所示。



从上图的迁移流程示意图中可以看出，整个迁移主要分为三步，即融合、迁移和下线。

- **融合**：首先我们需要在腾讯云ES控制台上申请一套和自建ES集群同等规模的空集群，即上图2中的目标集群，然后将云上的集群全量重启后加入到自建的ES集群中，使得两个集群融合成一个大集群。
- **迁移**：融合完成后，通过对ES集群cluster/settings设置exclude属性来进行分片的迁移，当执行了如下API后，ES集群就会自动将自建集群节点上的分片逐步驱逐到云上的节点上来。从而完成分片的搬迁和集群数据的迁移工作。

```
PUT _cluster/settings
{
  "persistent": {
```



```
        "cluster.routing.allocation.exclude_name": "{source_node_names}"
        }
}
```

- **下线**：当自建集群节点上的分片都已经全部迁移完成后，再将自建集群节点全部停机下线，这样便完成了整个集群的迁移上云过程。我们可以通过如下API来查看自建节点上的分片数是否为0。

```
GET /_cat/allocation?v=true
```

shards	disk.indices	disk.used	disk.avail	disk.total	disk.percent
11	26.8mb	1.2gb	18.3gb	19.5gb	6
11	8.8mb	1.1gb	18.3gb	19.5gb	6
10	20mb	1.1gb	18.3gb	19.5gb	6

在线融合迁移方案优势

平滑迁移，业务无感知，不停服

相比于 Logstash、快照备份和跨集群复制等离线迁移方案，在线融合迁移真正做到了在线平滑迁移和业务不停服。

集群无需重启，无需做数据一致性校验

在线融合迁移由于是通过全量重启云上ES集群来加入自建集群，因此不会对客户自建集群做任何有侵入性的操作，不同于快照迁移，需要提前安装 COS 插件并重启集群；也不需要像 Logstash 迁移，在迁移前后需要对索引 doc 数量做一致性校验。

对版本要求较低，只需云上集群版本不低于自建集群即可

在线融合迁移对集群版本的要求是只需要云上集群版本不低于自建集群版本即可。而云上目前提供了从5.6.4、6.8.2、7.101、7.14.2和8.16.1多个版本，相信总有一版本能够满足客户需求。

极大节约人力成本和运维成本

之所以说能够极大节约人力成本和运维成本，主要原因是离线迁移方案涉及到的方方面面实在太多，例如需要提前跟业务同学沟通可接受停服时间和停服时长，大部分业务低峰期都是在凌晨，因此离线迁移对迁移时间要求比较苛刻；同时离线迁移还需要做数据一致性校验，这些都是非常耗时耗力的工作。而在线融合迁移方案由于能够做到平滑透明、业务不停服，因此整个迁移过程都不需要任何业务方同学参与，也对迁移时间没有严格要求，更不需要做数据一致性校验。这些都能够极大节约人力和运维成本。

在线融合迁移的限制

虽然在线融合迁移能够彻底解决业务停服的痛点，也能够极大提升迁移效率。但是也并不是所有的自建集群都符合融合迁移的条件的，经过我们大量的测试和实际迁移经验，总结了如下几点限制和需要满足的条件。

自建 ES 集群版本不能大于云上集群版本

如云上 ES 当前提供的最高版本号是7.14.2，因此如果自建集群版本号大于7.14.2，则无法通过在线融合方案迁移；主要原因是云上低版本的集群节点无法加入高版本的自建集群中。另外对于版本号，最好是两个集群大版本号一致，例如自建版本是6.4.6，则云上优先选择6.8.2，自建版本是7.5.2，云上优先选择7.10.1。

自建 ES 集群上不能安装有云上集群不存在的插件

在迁移之前需要先在自建集群节点上执行如下 API 来查看自建集群上都安装了哪些插件。

```
curl  http://127.0.0.1:9200/_cat/plugins
```

如果有发现安装了云上不存在的插件则会使得云上 ES 集群节点融合后，分片分配失败导致集群状态异常问题。下图4是云上 ES 集群支持安装的插件列表。

插件名称	插件说明	状态	操作
analysis-icu	Elasticsearch Unicode文本分析插件	未安装	安装
analysis-ik	Elasticsearch IK分析插件	已安装	更新词典 重装
analysis-kuromoji	Elasticsearch日文分析插件	未安装	安装
analysis-nori	Elasticsearch 韩文分析插件	未安装	安装
analysis-phonetic	Elasticsearch音标分析插件	未安装	安装
analysis-pinyin	Elasticsearch拼音分析插件	已安装	卸载 重装
analysis-qq	Elasticsearch QQ分析插件	未安装	安装
analysis-smartcn	Elasticsearch智能中文分析插件	未安装	安装
analysis-stconvert	Elasticsearch繁简体分析插件	已安装	卸载 重装
ingest-attachment	Apache Tika信息提取插件	未安装	安装
mapper-murmur3	插件用于在创建索引时计算并存储字段的hash值	未安装	安装
mapper-size	插件用于在创建索引时记录文档压缩前的大小	未安装	安装
repository-cos	插件用于存储Elasticsearch Snapshot到腾讯云COS	已安装	卸载 重装
repository-hdfs	插件提供了对Hadoop分布式文件系统（HDFS）存储库的支持	未安装	安装
sql	开源SQL解析插件	已安装	卸载 重装

需确保自建 ES 集群和云上集群网络互联互通

这里说的网络互联互通是要求云上的节点可以直接访问到自建集群的节点，同样自建集群节点也可以直接访问到云上集群节点，中间不可以有任何代理 Proxy。中间有 Proxy 的访问都属于单向网络通信，而不是双向网络通信。下面介绍自建集群两种常见的网络环境和网络通信方案：

1. 腾讯云 CVM 自建 ES 集群迁移到腾讯云 ES：
   由于腾讯云 ES 集群采用了跨租户弹性网卡直接打通了 ES 集群 VPC 和客户 VPC 网络环境，因此对于腾讯云 CVM 自建的 ES 集群，只需要使用自建集群所在的 VPC在云 ES 控制台购买集群即可实现网络双向互通。如果自建集群和腾讯云 ES 集群不在同一个地域，则可以采用云上云联网或者对等连接方案进行打通网络。
2. IDC 自建 ES 集群迁移到腾讯云 ES：
   如果客户的自建 ES 集群是在自己的 IDC 机房或者是其他云厂商云服务器中，则可以通过专线方式打通两边的网络。两个集群网络打通后可以通过在云上集群节点上至下 telnet 自建 ip 9300方式进行验证。

需确保自建 ES 集群没有开启 security

通常情况下，客户的自建集群都是开源版本的，如果是基础版且开启了 security 的话，则需要先将 security 关闭后才能发起融合。并且白金版集群也不支持融合。如果期望上云后集群使用白金版，则可以在迁移结束后再升级到白金版即可。

云上集群名称需和自建集群名称一致

默认情况下，腾讯云 ES 集群会随机生成一个固定格式固定长度的字符串作为集群的名称，即 elasticsearch.yml 配置文件中的 cluster.name，如 cluster.name: "es-cohesszwr"（此集群 ID 配置名为虚构）。
而如果需要和自建集群进行融合，则必须要保证两个集群的 cluster.name 相同才可以。由于是让云上集群去加入自建集群，因此需要保证云上集群的集群名称与自建集群相同。自定义云上集群配置名需要开通白名单支持，开白后会看到如下图所示的填写项。如果集群创建出来后没有设置自定义的集群名称，可以找腾讯云 ES 迁

移同学后台处理。

在线融合迁移基本步骤

在线融合迁移的过程需要客户的运维同学和腾讯云ES团队的迁移同学密切配合才能完成，由于 ES 在7.0版本优化了 Master 选主逻辑，因此7.0以下的版本和7.0以上的版本需要分别采用不同融合策略。下面介绍下不同版本迁移的基本步骤和差异。

7.0以下版本迁移

7.0以下版本的迁移是指客户自建 ES 集群版本和腾讯云 ES 集群版本都是7.0以下的版本，如客户版本是6.4.5，云上版本为6.8.2。

1. 分片锁定（客户侧操作）
   迁移之前，需先在自建集群上执行如下 API，将分片优先锁定在自建集群节点上，这么做的原因是防止两个集群融合后分片自动漂移到云上节点，造成未知风险。

```
curl -H "Content-Type: application/json" -XPUT _cluster/settings -d '{
     "cluster.routing.allocation.include._name" : "自建集群节点名称列表"
}'
```

2. 集群融合（腾讯云侧操作）
   在这一步腾讯云 ES 迁移人员会调用后台接口，将自建 ES 集群节点追加到云上节点中，然后全量重启云上集群。这样在云上集群节点起来后会自动加入到自建集群中。这样就顺利融合成了一个大集群。

```
curl localhost:5100/cluster/update -d '{
"cluster_name": "es-cohesszwr",
"operator": "wr",
"es_config": {
    "discovery.zen.ping.unicast.hosts": "
    [\"10.0.0.xx:9300\",\"10.0.0.xx:9300\",\"10.0.0.xx:9300\",
    \"10.0.0.xx:9300\", \"10.0.0.xx:9300\",, \"10.0.0.xx:9300\"]"
  },
  "restart_type": "full_cluster_restart"
}'
```

3. 数据迁移（腾讯云侧操作）
   集群融合成功后，如果未发现什么异常问题，即可发起正式迁移了，通过执行如下 API。

```
curl -H "Content-Type: application/json" -XPUT _cluster/settings -d '{
"cluster.routing.allocation.include._name" : "云上集群节点名称列表",
"cluster.routing.allocation.exclude._name" : "自建集群节点名称列表"
}'
```

   该 API 的作用就是将自建集群节点上的分片驱逐到云上节点。在融合的中间状态由于分片迁移会消耗一定的性能。因此为了稳定性考虑，可以通过如下 API 将迁移的并发度和迁移速度调小一点。

```
PUT _cluster/settings
{ "persistent": {
         "cluster.routing.allocation.node_concurrent_recoveries": 2,
         "indices.recovery.max_bytes_per_sec": "40mb/s" }
}
```

在迁移的中间状态，业务侧这时候可以将客户端连接ES集群的配置改成腾讯云 ES 的访问 VIP。

4. 集群分离（客户侧操作）
   检查数据都迁移到云上节点后，客户需要将自建集群节点进行停机下线，将两个融合的集群进行分离。下线自建节点前需要修改自建集群配置名称 cluster.name 为其他值，如在原集群配置名称后加上-local，以及检查是否有自动拉起 ES 进程的脚本，如果有也一并删除。另外下线自建节点时，需要逐台下线，且把当选 master 节点安排在最后一个下线。这么做的好处是可以最大程度减少 Master 选主的次数。
5. 配置恢复（腾讯云侧操作）
   腾讯云 ES 迁移同学确认自建集群和云上集群完全分离后，再次和客户侧业务进行确认是否有异常，如果没有异常，则通过调用如下 API 将云上配置进行恢复到融合前。

```
curl localhost:5100/cluster/update -d '{
"cluster_name": "es-cohesszwr",
"operator": "wr",
"es_config": { "discovery.zen.ping.unicast.hosts": "null"},
"restart_type": "no_restart"
}'
```

> **注意**
> 这里的 discovery.seed_hosts 设置为 null，是因为后台接口会自动下发初始节点 IP 到配置中，另外 restart_type 务必设置为 no_restart，不重启方式更新。否则会影响业务稳定性。

7.0以上版本迁移

7.0以上版本的迁移是指客户自建 ES 集群版本和腾讯云 ES 集群版本都是7.0以上的版本或者云上集群是7.0以上版本，如客户版本是6.8.3，云上版本为7.5.1或者自建版本为7.5.2，云上版本为7.10.1。

1. 分片锁定（客户侧操作）
   这一步和7.0以下版本操作是一样的，具体操作详情可参考上面步骤。
2. 集群融合（腾讯云侧操作）
   这一步和7.0以下版本最大的区别就是将自建集群节点列表追加到云上并全量重启云上集群后，云上集群并不能直接加入自建集群，这主要是和7.0以上版本选主算法有关。因此为了让云上集群全量重启后能够正常融合，需要在全量重启后逐步对云上集群节点执行如下命令用于抹除云上集群的元数据。详细原理可参考 官方文档 提供的 elasticsearch-node 命令。
   首先先执行追加自建节点 IP 配置 API：

```
curl localhost:5100/cluster/update -d '{
  "cluster_name": "es-cohesszwr",
  "operator": "wr",
  "es_config": {
  "discovery.seed_hosts": "
      [\"10.0.0.xx:9300\",\"10.0.0.xx:9300\",\"10.0.0.xx:9300\",
       \"10.0.0.xx:9300\", \"10.0.0.xx:9300\",, \"10.0.0.xx:9300\"]"
  },
  "restart_type": "full_cluster_restart"
}'
```

> **注意**
> 上面的 API 中 es_config 字段的 key 是 discovery.seed_hosts，而不再是 discovery.zen.ping.unicast.hosts。

随后再通过脚本批量对云上集群每个节点执行如下命令来抹除集群元数据，随后便可以看到云上集群节点就可以成功加入自建集群中了。

```
ps -ef | grep java | grep c_log | awk '{print $2}' |xargs kill -9
cd /data1/containers/*/es/
./bin/elasticsearch-node detach-cluster （选择y）
```

3. 数据迁移（腾讯云侧操作）
   这一步和7.0以下版本操作是一样的，主要目的是让分片从自建集群节点上逐步平滑迁移到云上节点上来，具体操作详情可参考上面步骤。
4. 集群分离（客户侧操作）
   这一步和7.0以下版本操作是一样的，具体操作详情可参考上面步骤。

5. 配置恢复（腾讯云侧操作）
   配置恢复的操作也基本和7.0以下版本一样，唯一的区别同样在于 es_config 参数中的 key，这里应该是 discovery.seed_hosts。同样采用 no_restart 不重启方式进行更新。

```
curl localhost:5100/cluster/update -d '{
     "cluster_name": "es-cohesszwr",
     "operator": "wr",
     "es_config": { "discovery.seed_hosts": "null"},
     "restart_type": "no_restart"
}'
```

在线融合迁移常见问题

自建集群节点切勿同时下线，需逐台停机下线

这里我们对7.0以上版本做过一系列测试，当集群处于融合状态后，然后一次性同时下线所有的自建集群节点后，云上集群会立即处于失主状态，导致集群不可用，业务出现大量报错。这主要是由于集群检测到一半节点（具有 node.master 角色）同时脱离集群后，已经不满足选出 Master 节点的基本条件，从而导致集群长期处于失主状态。
因此为了避免在集群分离时集群不可用风险，切勿同时下线自建节点， 而是需要逐台进行下线，并且把自建节点上当选的 Master 节点放在最后一个下线。否则会导致多次切主，且有可能多切到自建节点上。

禁止同一集群二次融合，否则会有索引丢失风险

在线融合迁移方案需要在集群融合后将所有的自建集群索引迁移到云上，强烈不建议只迁移部分索引，或者分批次迁移，如第一次融合后迁移部分业务索引，然后分离集群，过段时间再进行二次融合，再把剩余索引迁移过来。
结合我们的测试情况来看，当数据迁移完成两个集群分离后，自建的 ES 集群会出现 Red 情况，主要是由于索引都迁移到了云上，而自建集群上只有索引的元信息，因此会出现索引无法分配的情况，这也是集群分离后自建集群 Red 的原因。
通常这种情况下客户会选择删除这些索引以让集群恢复 Green。如果这时候再次将两个集群融合来迁移剩余索引的话，由于是通过全量重启云上集群节点来加入自建集群，因此融合后会以自建集群上当选 Master 上的集群元数据为准。因此一旦融合成功，就会把自建集群的元数据同步给云上集群节点，这时候就会直接删除云上第一次融合迁移过去的索引，从而导致之前迁移的索引数据全部丢失。这个风险的根本原因就是我们第一次分离后将自建集群上 Red 的索引删除了，而这些删除的索引都会记录在自建集群的元数据里，在一个叫索引坟墓的地方。融合后会同步这些元数据给云上集群。详情可参考官方文档 Index tombstones 。
虽然我们强烈不建议甚至禁止同一个集群二次融合，但是如果实在需要二次融合来解决业务不同阶段迁移的问题，可以通过如下解决方案来避免二次融合的数据丢失问题。
二次融合避免数据丢失解决方案：迁移索引 Reopen 方案：
1. **第一步**：第一次融合，在集群分离后，对于自建集群上 Red 的索引，不要进行删除，而是对这些索引执行 close 操作，即 关闭索引 。

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

2. **第二步** 第二次融合，融合成功后，再将之前关闭的索引执行 open 操作，即 打开索引 。

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

经过我们多次测试，迁移索引 Reopen 方案能够解决二次融合场景下数据丢失的问题。

客户端在融合前需确保已关闭嗅探功能

客户端开启嗅探 Sniff 功能是为了能够让 Client 端自动发现集群侧 ES 节点的动态变更。云上 ES 是通过 PrivateLink 打通的客户侧 VPC 和云上集群 VPC 网络，如果融合后客户端将访问配置切换到云上后，同时保持开启了嗅探功能，就会出现嗅探到的节点 IP 访问不同的情况，从而导致 Client 端大量的超时请求，影响业务使用。客户端报错信息可能如下：

```
NoNodeAvailableException[None of the configured nodes are available: [{#transport#-1}
{VTss4h8SRsCpP6EW5PoLxxxrQ}{192.168.106.xxx}{192.168.106.xxx:9300}] ]at
org.elasticsearch.client.transport.TransportClientNodesService.ensureNodesAreAvailable(TransportClientNodesSe
rvice.java:347) at
org.elasticsearch.client.transport.TransportClientNodesService.execute(TransportClientNodesService.java:245)
at org.elasticsearch.client.transport.TransportProxyClient.execute(TransportProxyClient.java:59) ......
```

因此为了保障融合迁移方案对业务的透明和平滑，需要在融合前检查客户端是否开启了 Sniff 嗅探，如果开启需关闭此功能。不同语言客户端接入云上 ES 集群方式可以参考腾讯云 ES 官方文档 通过客户端访问集群 。

腾讯云 Elasticsearch 集群迁移之 Logstash 方案

最近更新时间：2025-02-27 10:57:42

本文介绍了通过 Logstash 工具迁移 ES 集群数据到腾讯云 ES，如果需要迁移数据到腾讯云 Serverless 服务，详情请参见 迁移方案说明 。

迁移目标/范围

腾讯云 ES 是基于开源引擎打造的云端全托管 ELK（ES/Logstash/Kibana）服务，拥有高性能自研内核、支持读写分离、存算分离、自治索引、机器学习等能力，同时也支持自动弹性、零运维、低成本的 Serverless 版，被广泛应用于日志分析、在线搜索、向量检索与 RAG（检索增强生成）等场景。
本方案将详细介绍迁移流程、迁移步骤以及注意事项等方面的内容。通过本方案，用户可以在保证数据完整性、安全性和高效性的前提下，无需进行大规模业务改造，即可顺利平滑完成迁移过程。

迁移目标	迁移方案优点	腾讯云 ES 相对其他日志产品优点
自建 ES（ELK）> 腾讯云 ES 友商云 ES（ELK）> 腾讯云 ES 友商云 Opensearch > 腾讯云 ES	• ES 业务无需大范围改造 • ES 迁移支持全量/增量 • 支持在线不停服迁移（部分场景）	• 自研特性优化，助力客户降本增效 • 支持多级存储，热 > 冷 > COS 标准/低频/归档存储 • 功能丰富，包括日志/指标/APM/告警等全观测能力 • 完全兼容 ELK（ES/Logstash/Kibana），保留原有权限、报表、使用习惯

方案简介

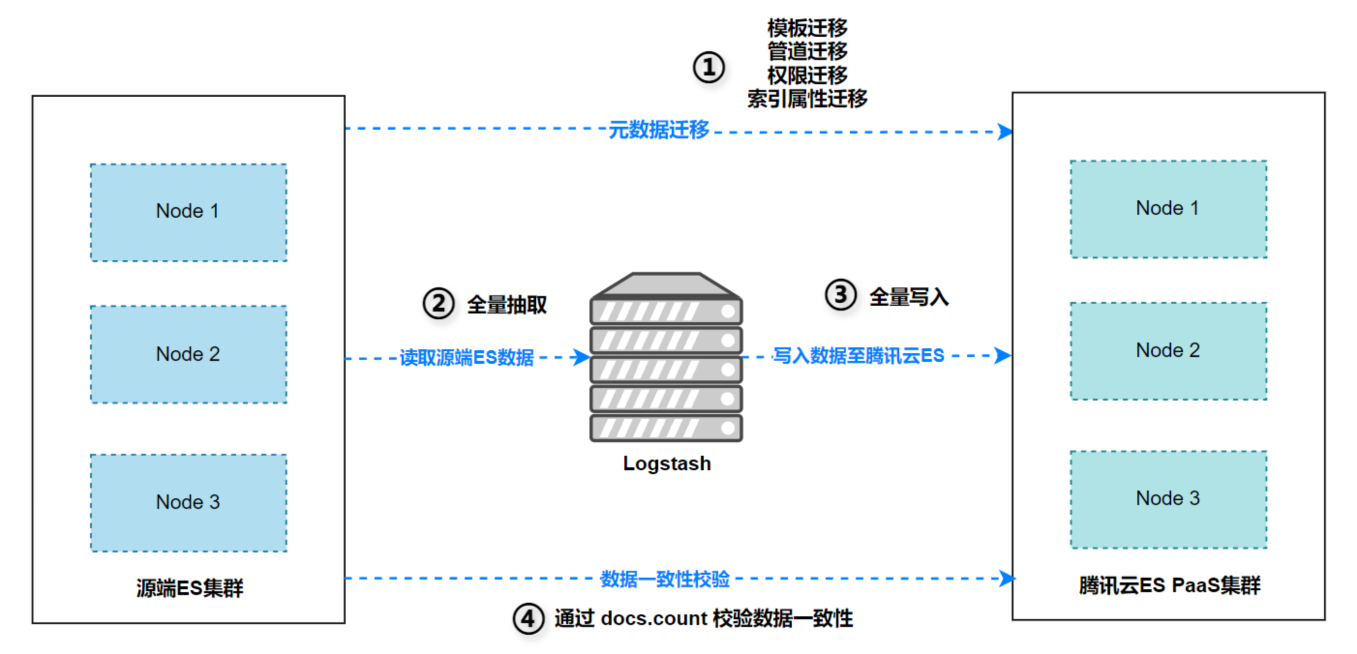
ES 迁移主推 Snapshot 方案，如有其他特殊场景，可参考 Logstash 及在线融合方案。具体方案对比如下：

迁移方案	适用场景	适用产品	停服割接时间	增量	更新	删除	转化	方案实施
Snapshot	可接受短暂停服，对迁移速度有要求的场景	• 自建 ES • AWS/阿里/华为云 ES	通过多次增量快照，保证大部分场景下停服时间<=20min	✓	✓	✓	X	可以根据文档自助迁移
Logstash	源端版本 > 目标端时，或迁移过程中需要定制化改造的场景	• AWS 云 Opensearch • 华为云 Opensearch • AWS/阿里/华为云 ES	Logstash 方案需全程停服，停服时间与集群资源性能、专线带宽等因素有关			X	✓	可以根据文档自助迁移
在线融合（腾讯云 ES 特有）	在线不停服迁移	自建 ES	无需停服	✓	✓	✓	X	需腾讯云 ES 后台配合

概述

本迁移方案旨在为用户提供一种从自建 ES 或友商云 ES 迁移到腾讯云 ES 的实践教程。本方案将详细介绍使用 Logstash 快照方案的迁移流程、迁移步骤以及注意事项等方面的内容。通过本方案，用户可以在保证数据完整性、安全性和高效性的前提下，顺利完成迁移过程。

方案概览



准备工作

账号开通

客户需要开通一个腾讯云账号。

网络打通

客户的 ES 集群需要有一条能访问腾讯云服务的专线网络或 VPN。

- 专线：专线接入
- VPN：建立 VPC 到 IDC 的连接（SPD 策略）

> **注意：**
> 数据迁移务必使用内网，否则默认会走公网进行数据传输，公网传输存在数据泄漏风险和潜在危险。

ES 集群创建

前置确认

确认 ES 集群搭建前基础信息

信息编号	确认信息	确认值（此列需要客户确认）	备注
1	区域	华南地区-广州	ES 集群所在的区域
2	产品版本	ES 7.14.2	ES 集群版本 全文检索建议7.14.2，向量检索推荐8.13.3
3	组件种类	es，logstash，filebeat	ES 集群需要包含的组件
4	鉴权	不开启	白金版不支持关闭鉴权，非白金版支持开启/关闭（需提工单支持）
5	业务场景	全文检索，向量检索	根据业务场景决定集群版本 全文检索建议<=7.14.2，向量检索推荐>8
6	集群网络&可用区	• 集群网络举例：Default-VPC • 子网举例：Default-subnet	集群所在的 VPC，和可用区子网需要提前规划好 有容灾需求可选择多可用区，节点将均匀分布在所选择的多个可用区，可用节点总量不变
7	安全组	安全组举例：sg-xxxx	ES 集群安全组，如需关联安全组，可在集群创建后进行关联

确认 ES 集群资源配置

生产环境的规格参考实际的 ES 业务需求，有特殊机型要求的，需提早申请，以便腾讯侧备货，腾讯云建议基础的 ES 生产环境配置如下：

节点类型	节点说明	推荐配置	节点数量
热数据层	• 默认节点类型，可搭配温数据层实现冷热分离架构； • 数据写入层。通常将经常查询的数据放置在热层，以提升查询与写入效率。	标准型 S1 CPU：8核 内存：32GB 数据盘： 增强型 SSD 云硬盘 500G x 3	N
温数据层	• 可选节点，支持仅开启温数据层，也可搭配热数据层实现冷热分离架构； • 将访问频率较低且更新频率较低的数据存放在温层，以节约成本。非必要，有冷热分层需求才需要开启。	标准型 S1 CPU：4核 内存：16GB 数据盘： 高性能云硬盘 1000G x 3	N
专用主节点	• 可选节点，建议10节点以上或分片规模>5000时启用； • 管理节点，保证集群的调度正常进行。非必要，默认所有数据节点都是候选主节点。当集群节点数>10 或 数据量较大时建议选购专用主节点。	标准型 S1 CPU：4核 内存：16GB	3
协调节点	• 可选节点，默认所有数据节点都是协调节点。建议有特殊需求时候启用； • 协调节点只用于处理客户端请求，不存储数据，能够提升集群处理性能和服务稳定性，适用于高并发读写、多聚合查询等cpu密集型业务场景。	标准型 S1 CPU：4核 内存：16GB	N
机器学习节点	• 可选节点，有推理需求时启用； • 机器学习节点用于创建机器学习任务，自动进行数据分析，识别异常数据。也可在此节点加载向量模型，提升向量生成和向量检索能力。	标准型 S1 CPU：16核 内存：32GB	N
Kibana 节点	默认开启，用于数据可视化，1核2G规格免费，支持购买更高规格。	标准型 S1 CPU：1核 内存：2GB	1

集群购买

购买腾讯云 ES 集群请参见 创建集群。

Logstash 购买

购买腾讯云 Logstash 请参见 创建实例。

插件安装及词典上传（必要）

在进行数据迁移前，需要确认索引是否使用自定义插件及自定义词典，如果有则需要在云上 ES 集群同步，否则相关索引数据恢复无法成功。

插件查询

使用下面命令查询自建集群使用的插件：

```
curl -s -uelastic:'源端集群密码' 'http://源端IP:9200/_cat/plugins?h=component,version' | sort | uniq
```

插件安装

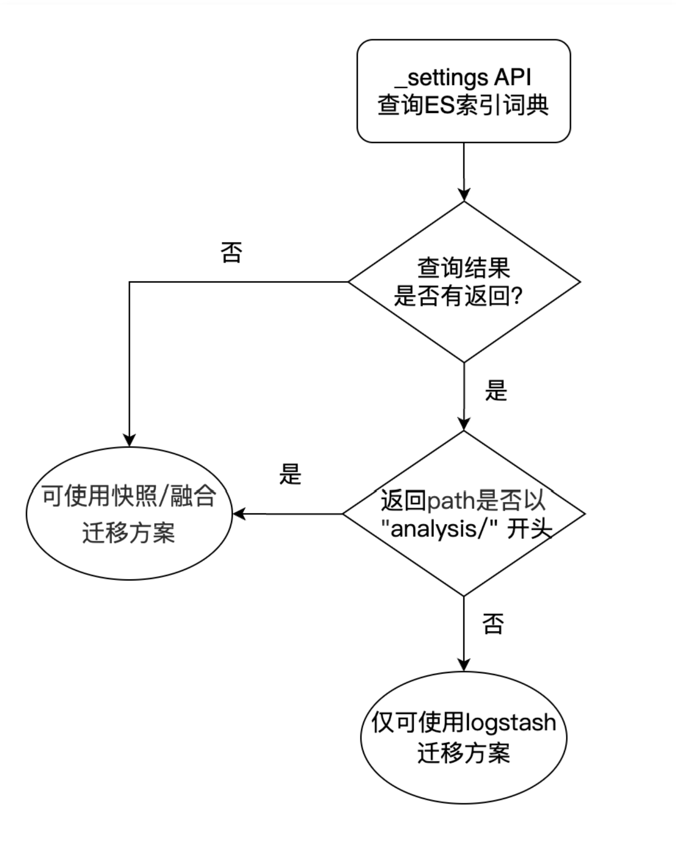
腾讯云 ES 支持的插件列表及插件安装可以参考 腾讯云 Elasticsearch Service 插件列表，如果有另外的自定义插件需要安装可以联系腾讯云侧开通白名单使用此功能。

词典查询

自定义词典查询如下：

```
curl -s -uelastic:'源端集群密码' 'http://源端IP:9200/_settings?pretty' | grep "_path"
```

自定义词典查询流程图



IK 分词/停用词词典查询

```
# 进入到ES主目录，$ES_HOME需替换成实际目录
cd /$ES_HOME/config/analysis-ik

# 进入ik词典目录下，查看配置文件
cat IKAnalyzer.cfg.xml
```

未配置任何 ik 词典则默认内容如下，无需更新 ik 词典。如自定义了 ik 词典，则需上传至腾讯云 ES。


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
        <comment>IK Analyzer 扩展配置</comment>
        <!--用户可以在这里配置自己的扩展字典 -->
        <entry key="ext_dict"></entry>
         <!--用户可以在这里配置自己的扩展停止词字典-->
        <entry key="ext_stopwords"></entry>
        <!--用户可以在这里配置远程扩展字典 -->
        <!-- <entry key="remote_ext_dict">words_location</entry> -->
        <!--用户可以在这里配置远程扩展停止词字典-->
        <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
```

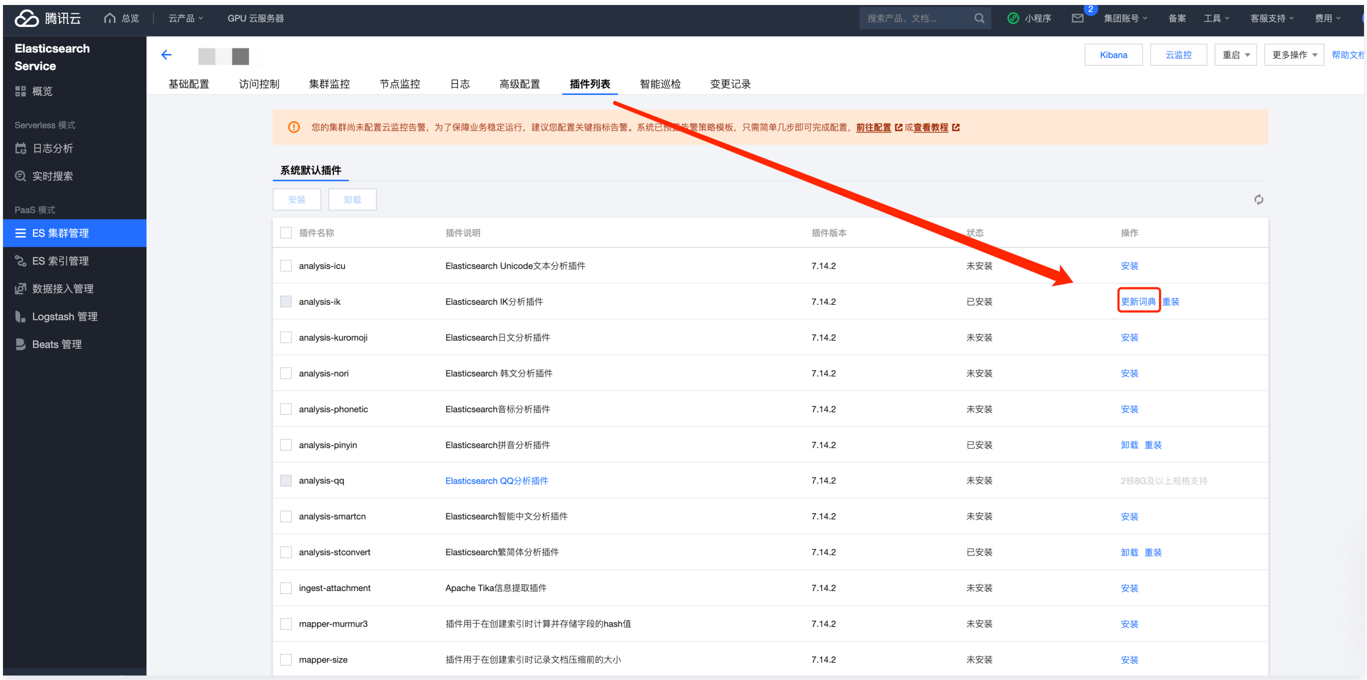
词典上传

- **同义词词典更新**

进入 腾讯云 ES 控制台 > **ES 集群管理** > **ES 实例** > **高级配置**界面，单击**更新词典**，上传自定义词典，配置索引属性即可使用。

- **IK 分词/停用词词典更新**

进入 腾讯云 ES 控制台 > **ES 集群管理** > **ES 实例** > **插件列表**界面，选择 IK 插件单击**更新词典**，上传分词词典后即可。



数据迁移

方案介绍

- **介绍：**Logstash 是一款专门用于应用程序日志、事件的传输、处理、管理的产品。这里我们可以通过 Logstash 完成跨 ES 集群的数据迁移工作。
- **适用场景：**该种迁移方案适用于集群版本跨度较大，能够接受集群一段时间暂停写的场景。

迁移流程介绍

序号	迁移阶段	备注
1	元数据迁移	使用同步工具进行元数据迁移，工具使用说明见实践教程文档。集群相关的元数据包括： • 索引属性迁移 • 模板迁移 • 管道迁移 • 权限迁移
2	全量数据迁移	Logstash 方案需全程停服，停服时间与集群资源性能、专线带宽等因素有关
3	数据一致性校验	数据同步完成后，快速通过 docs.count 来判断数据一致性。

用户及角色迁移

- 如需迁移用户及权限，可使用工具一键迁移：

```
rpm -vih https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/es-security-sync-lastest.x86_64.rpm
```

- 或下载后上传再使用 rpm 安装：

```
rpm -vih es-security-sync-lastest.x86_64.rpm
```

> **说明：**
> security 功能 ES 6 版本才开始支持，ES 8 的 security 索引不允许修改。所以仅支持 6 到 7，以及 7 到 6 的迁移。

迁移示例


```
es-security-sync \
  --src_url http://10.0.xx.xx:9200 \
  --dst_url http://10.0.xx.xx:9400 \
  --src_user log_user \
  --dst_user log_user \
  --src_password ****** \
  --dst_password ******
```

索引模板迁移

- 索引模板为 ES 业务使用中最常用的功能，在迁移前务必先同步索引模板。模板迁移使用工具一键迁移即可：

```
rpm -vih https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/es-template-cp-lastest.x86_64.rpm
```

- 或下载后上传再使用 rpm 安装：

```
rpm -vih es-template-cp-lastest.x86_64.rpm
```

迁移示例

```
es-template-cp \
  --src_url http://源ES IP:9200 \
  --dst_url http://腾讯云ES IP:9200 \
  --src_password '******' \
  --dst_password '腾讯云ES密码'
```

集群管道迁移（按需操作）

ES 管道迁移使用工具一键迁移即可。

```
#下载jar包
#运行环境：JDK-1.8
wget https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/CopyPipelines.jar
```

示范样例

> **注意：**
> 任何参数不要空缺，无密码集群可指定任意密码。

- **样例一**

源端，目标端，集群带密码，跳过目标端已存在的同名管道。

```
java -jar CopyPipelines.jar http://源ES IP:9200 elastic '源端ES密码' http://腾讯云ES IP:9200 elastic '腾讯云ES密码'  skip
```

- **样例二**

源端，目标端，集群带密码，覆盖目标端已存在的同名管道。

```
java -jar CopyPipelines.jar http://源ES IP:9200 elastic '源端ES密码' http://腾讯云ES IP:9200 elastic '腾讯云ES密码' overwrite
```

索引属性同步

- 由于 logstash 不支持迁移 es 索引属性，例如 settings、mapping，所以需要提前同步：

```
rpm -vih https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/es-index-
properties-sync-lastest.x86_64.rpm
```

- 或下载后上传再使用 rpm 安装：

```
rpm -vih es-index-properties-sync-lastest.x86_64.rpm
```

> **说明：**
> 该工具支持修改索引 setting、mapping。

同步示例

```
es-index-properties-sync \
  --src_url http://10.0.xx.xx:9200 \
  --dst_url http://10.0.xx.xx:9200 \
  --src_password ****** \
  --dst_password ****** \
  --indices '*,-.*' \
```

> **注意：**
> 需要同步的索引在目标集群需不存在，否则会自动跳过已存在的索引，建议同步索引属性前清理目标集群索引。

Logstash 全量同步

索引分组

在索引数量比较大的场景下，我们需要将索引分成多个组，以启动多个实例或管道来加速迁移。我们提供了分组工具：

工具安装

```
rpm -vih https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/es-index-
grouping-lastest.x86_64.rpm
```

参数说明

```
--user         default: elastic
--password     default: 默认读取环境变量 export password='***'
--ip           default: 没有默认值，必要参数，不允许为空
--port         default: 9200
--pattern      default: 索引匹配规则，默认为 '*'，支持逗号分隔
--instances    default: 分组数量。没有默认值，必要参数，不允许为空
```

使用示例

- 示例 1

ES 实例没有密码认证，端口为9200，且所有索引都需要进行 分组，分组数为10个。

```
es-index-grouping --ip 源ES IP --instances 10
```

- 示例 2

密码赋值和执行命令写成一条，中间无需分号。

```
password='123456' es-index-grouping --ip 源ES IP --port 9200 --user logUser --pattern '*,-.*' --instances 10
```

将分组后的 index list 根据不同管道填入 index =>，比如我们将全部索引分了10个组，我们则需要创建10个实例或管道。

Logstash 管道配置

1. 进入 Logstash 控制台，点击管道管理>新建管道。

2. 管道配置：单击引用模板可以快捷配置管道，配置完成后，单击**保存并部署**。

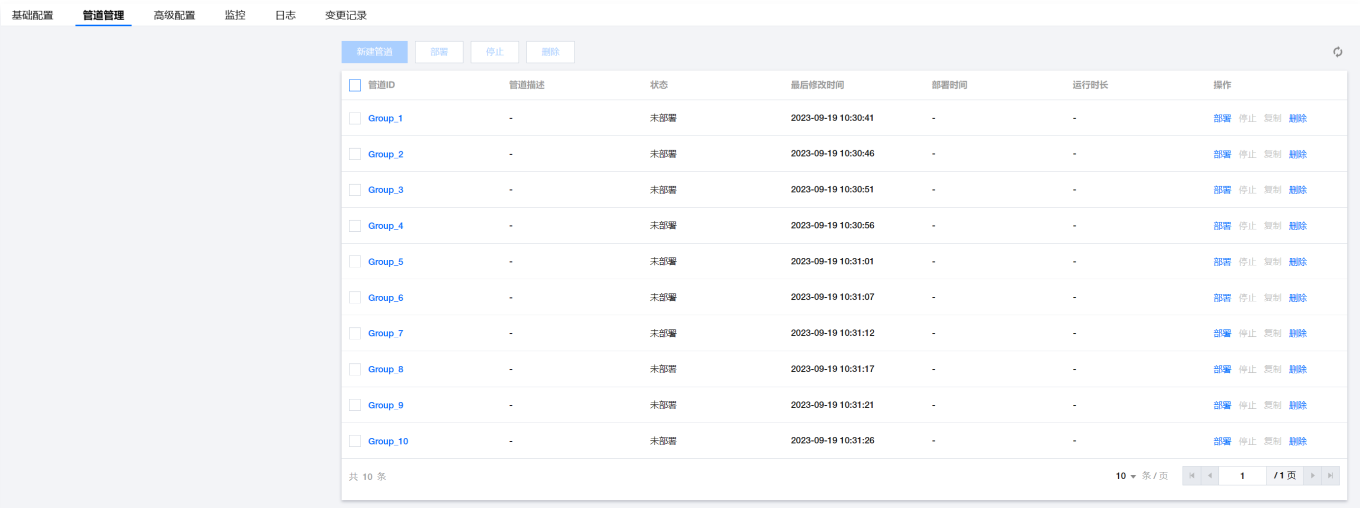
- pipeline.workers：一般为 CPU 数。
- pipeline.batch.size：设置批量执行 event 的最大值，一般3000-5000，不建议超过10000。
- pipeline.batch.delay：批处理的最大等待时长，一般3000-5000ms即可。

Config 配置

```
input {
    elasticsearch {
        hosts => "源ES IP:9200"
        index => "*,-.*"  # 如果是分组同步，这里则填入其中一个组的 index list，支持通配，例如：index => "index-a-*,
index-b-*"
        user => "elastic"
        password => "xxxxxx"
        docinfo => true
        size => 5000
        slices => 4        #建议设置为源端多数索引的分片数量
      }
}

output {
    elasticsearch {
        hosts => ["http://腾讯云 ES IP:9200"]
        user => "elastic"
        password => "腾讯云ES密码"
```

```
            index => "%{[@metadata][_index]}"
            document_type => "%{[@metadata][_type]}"
            document_id => "%{[@metadata][_id]}"
        }
}
```



如果有索引改名的需求，可以使用 filter 插件进行一下关键词替换：

```
filter {
    mutate {
        gsub => ["[@metadata][_index]", "test-", "ceshi-"]
  }
}
```

> **说明：**
> 这里演示的是将 test- 替换为 ceshi-。

Logstash 增量同步

- logstash 本身不具备同步增量的能力，其增量的实现需要业务使用 query 语句完成，且需要业务数据有 update_time 之类的字段，以标识数据的更新时间。
- 如果业务数据只有 create_time 或 @timestamp 这样的字段，则只能做到增量同步，无法做到差异同步（update）。
- logstash 不支持同步删除操作，所以在进行 logstash 增量同步后，目标端可能比源端多出部分数据（即logstash同步过程中源端被删除的数据）。

以下为 logstash input 增量同步示例：

开启索引动态 mapping

```
#为单个索引开启动态mapping
PUT index_name/_mapping
{
  "dynamic": true
}

#为全部索引开启动态mapping
PUT *,-.*/_mapping
{
  "dynamic": true
}
```

业务改造

业务新增、更新数据时需要新增 update_time 字段，标识数据的当前更新时间。

增量同步

在按照本文完成了 ES 全量数据同步后，继续配置新管道进行增量数据同步。

```
input {
    elasticsearch {
        # 源端ES地址。
        hosts =>  ["http://源ES IP:9200"]
        # 安全集群配置登录用户名密码。
        user => "elastic"
        password => "源端ES密码"
        # 需要迁移的索引列表，多个索引使用英文逗号（,）分隔，支持通配。
        index => "sample_data"
        # 按时间范围查询差异数据，以下配置表示查询最近1小时更新的数据。
        query => '{"query":{"range":{"update_time":{"gte":"now-1h/h","lte":"now/m"}}}}'
        # 定时任务，以下配置表示每半小时执行一次。
        schedule => "0 */30 * * * ?"
        scroll => "5m"
        docinfo=>true
        size => 5000
}

output {
    elasticsearch {
        hosts => ["http://腾讯云 ES IP:9200"]
        user => "elastic"
        password => "腾讯云ES密码"
        index => "%{[@metadata][_index]}"
        document_type => "%{[@metadata][_type]}"
        document_id => "%{[@metadata][_id]}"
    }
}
```

特别注意

logstash 增量同步前建议充分调试，将 input 插件里的 query 语句拿到源端执行，看能否查到数据，例如：

```
GET sample_data/_search
{
  "query": {
    "range": {
      "update_time": {
        "gte": "now-1h/h",
        "lte": "now/m"
      }
    }
  },
  "_source": "update_time"
}
```

以下两种情况可能导致查询不到相关数据：

- 源端数据有延迟或者不是流式的，可能是跑批的数据，这样我们则可以把 now-1h 调大一些，比如 now-5h，甚至更大。我们也可以直接获取当前数据的最大时间来判断数据实时性，例如：

```
GET sample_data/_search
{
  "sort": [
    {
      "update_time": {
        "order": "desc"
      }
```

```
    }
  ],
  "_source": "update_time"
}
```

- 源端数据时间字段非 date 类型，而是 keyword 或者 text 类型，那么即便数据是时间字符串，也无法使用时间运算，更无法使用 range 查询，所以结果会为空。
  - 查到的数据比当前时间少8小时，则是符合预期的，因为 ES 默认是使用的 UTC 时间，无需对时间进行额外处理；
  - 如果查到的数据是当前的时间，没有少8小时，则说明数据在写入时候没有使用 UTC 时间，则同步时需要在 logstash 侧使用 now 运算：now + 8h。如果数据本身有延迟，再在8h上额外增加即可。
  - 如果查询报错，则说明使用的时间字段不是时间类型，而是数值类型或其他类型，例如：

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "query_shard_exception",
        "reason" : "failed to create query: For input string: \"now-5m\"",
        "index_uuid" : "mV2rUwq0Q7ajXt6i7gfWcA",
        "index" : "sample_data"
      }
    ],
    "type" : "search_phase_execution_exception",
    "reason" : "all shards failed",
    "phase" : "query",
    "grouped" : true,
    "failed_shards" : [
      {
        "shard" : 0,
        "index" : "sample_data",
        "node" : "iselBn1mQUGLajoaPFuDmw",
        "reason" : {
          "type" : "query_shard_exception",
          "reason" : "failed to create query: For input string: \"now-5m\"",
          "index_uuid" : "mV2rUwq0Q7ajXt6i7gfWcA",
          "index" : "sample_data",
          "caused_by" : {
            "type" : "number_format_exception",
            "reason" : "For input string: \"now-5m\""
          }
        }
      }
    ]
  },
  "status" : 400
}
```

数据校验

可以下载数据条数校验工具进行对比，工具仅对比两端集群索引文档数量，不会校验内容是否相同。

工具安装

```
#下载jar包
#运行环境：JDK-1.8
wget https://tools-release-1253240642.cos.ap-
shanghai.myqcloud.com/elasticsearch/packages/ElasticsearchCompare.jar
```

使用案例

示范样例

共需要如下输入7个参数，任何参数不能空缺，无密码集群可指定任意密码。

参数名	解释	示范样例
srcClusterUrl	源端集群地址	http://127.0.0.1:9200
srcUser	源端集群用户名	elastic
srcPassword	源端集群密码	12345
dstClusterUrl	目标端集群地址	http://127.0.0.1:9201
dstUser	目标端集群用户名	elastic
dstPassword	源目标端集群密码	12345
index pattern	需要校验的索引的匹配规则	"*,-.*"

```
java -jar ESIndexDocumentCountComparison.jar <srcClusterUrl> <srcUser> <srcPassword> <dstClusterUrl>
<dstUser> <dstPassword> <index pattern>
```

样例

源端，目标端，集群带密码，需要校验两端集群全部索引的数据一致性，如果集群无密码则可输入任意密码。

```
java -jar ESIndexDocumentCountComparison.jar http://源端ES IP:9200 elastic '源端ES密码' http://腾讯云ES IP:9200
elastic '腾讯云ES密码'  "*,-.*"
```

返回说明

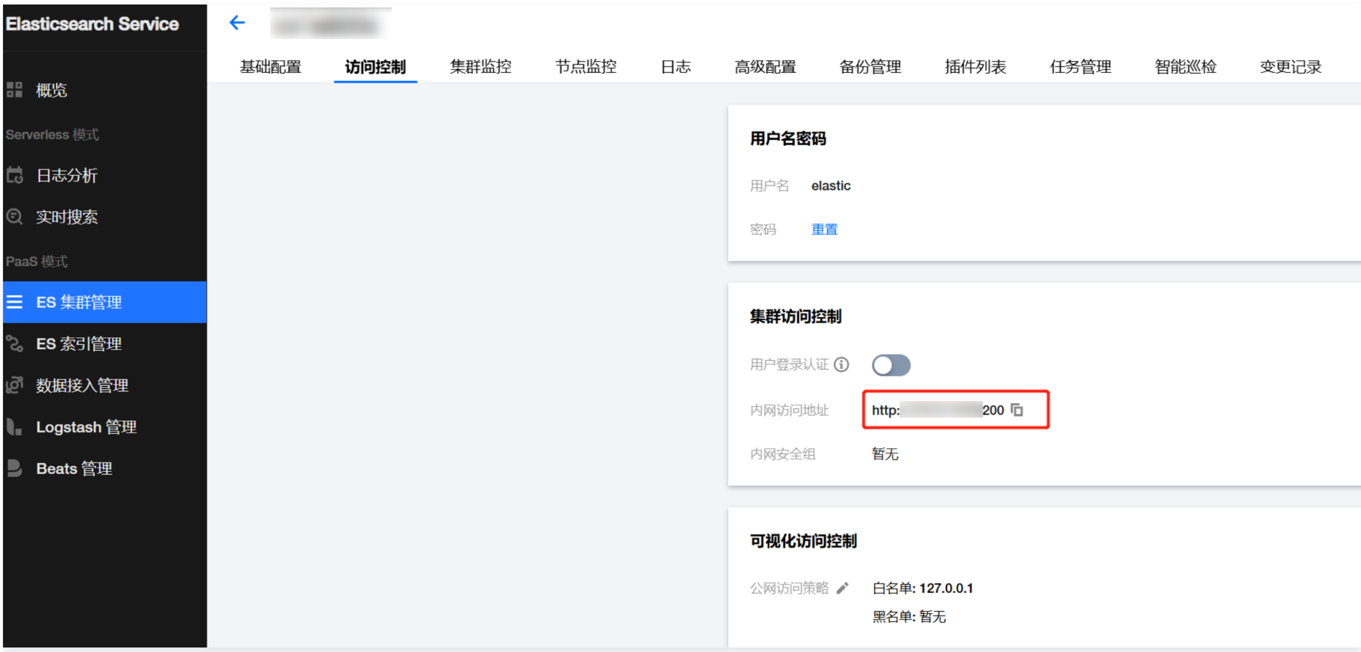
工具首先会返回源端集群的 close 索引，close 状态索引无法检验数据一致性，然后返回对于两边集群数据不一致的索引信息，不返回数据一致的索引信息。

- **数据不一致返回**

- **数据一致返回**

业务切割

数据迁移完成并完成一致性校验，以及完成业务兼容性完全测试，则可以进行业务切割上云，将 es ip 切换为云上内网访问地址即可：


Elasticsearch Service
概览
Serverless 模式
日志分析
实时搜索
PaaS 模式
ES 集群管理
ES 索引管理
数据接入管理
Logstash 管理
Beats 管理
基础配置
访问控制
集群监控
节点监控
日志
高级配置
备份管理
插件列表
任务管理
智能巡检
变更记录
用户名密码
用户名
elastic
密码
重置
集群访问控制
用户登录认证
内网访问地址
http:
200
内网安全组
暂无
可视化访问控制
公网访问策略
白名单: 127.0.0.1
黑名单: 暂无

腾讯云 Elasticsearch 集群迁移之 Snapshot 方案

最近更新时间：2025-05-13 15:27:42

本文介绍了通过快照的方式迁移 ES 集群数据到腾讯云 ES，如果需要迁移数据到腾讯云 Serverless 服务，请参见 迁移方案说明 。

迁移目标/范围

腾讯云 ES 是基于开源引擎打造的云端全托管 ELK（ES/Logstash/Kibana）服务，拥有高性能自研内核、支持读写分离、存算分离、自治索引、机器学习等能力，同时也支持自动弹性、零运维、低成本的 Serverless 版，被广泛应用于日志分析、在线搜索、向量检索与 RAG（检索增强生成）等场景。
本方案将详细介绍迁移流程、迁移步骤以及注意事项等方面的内容。通过本方案，用户可以在保证数据完整性、安全性和高效性的前提下，无需进行大规模业务改造，即可顺利平滑完成迁移过程。

迁移目标	迁移方案优点	腾讯云 ES 相对其他日志产品优点
自建 ES（ELK）> 腾讯云 ES 友商云 ES（ELK）> 腾讯云 ES 友商云 Opensearch > 腾讯云 ES	• ES 业务无需大范围改造 • ES 迁移支持全量/增量 • 支持在线不停服迁移（部分场景）	• 自研特性优化，助力客户降本增效 • 支持多级存储，热 > 冷 > COS 标准/低频/归档存储 • 功能丰富，包括日志/指标/APM/告警等全观测能力 • 完全兼容 ELK（ES/Logstash/Kibana），保留原有权限、报表、使用习惯

方案简介

ES 迁移主推 Snapshot 方案，如有其他特殊场景，可参考 Logstash 及在线融合方案。具体方案对比如下：

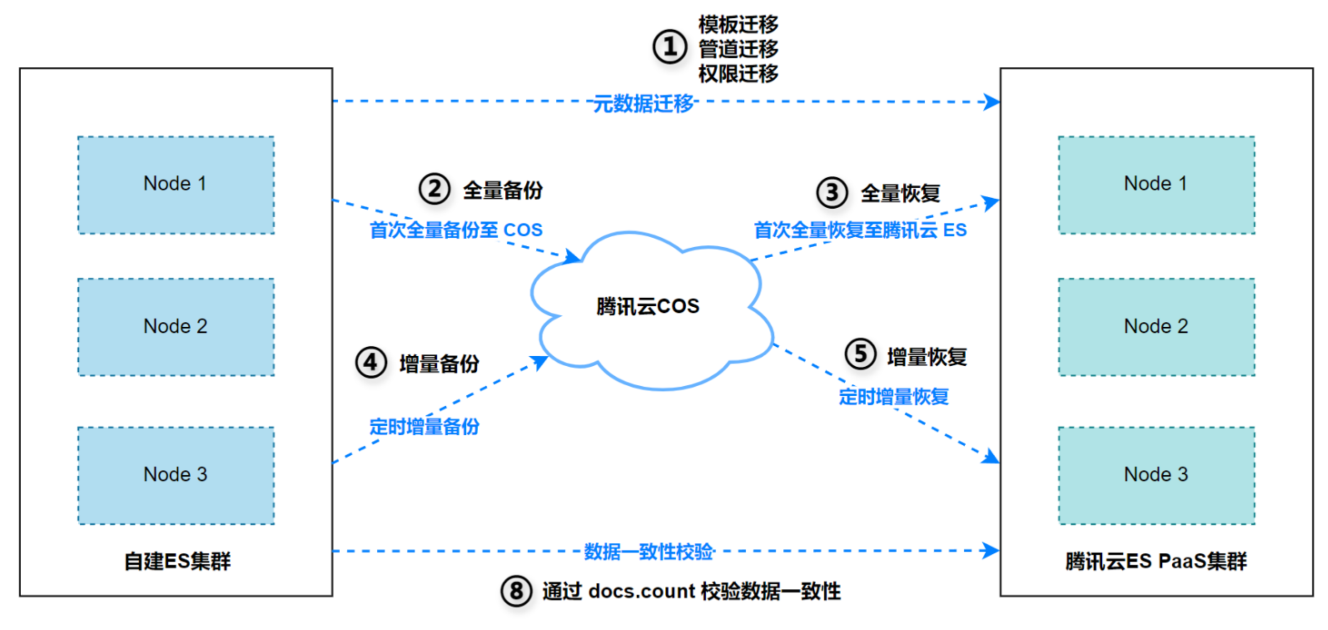
迁移方案	适用场景	适用产品	停服割接时间	增量	更新	删除	清洗	方案实施
Snapshot	可接受短暂停服，对迁移速度有要求的场景	• 自建 ES • AWS/阿里/华为云 ES	通过多次增量快照，保证大部分场景下停服时间<=20min	✓	✓	✓	☓	可以根据文档自助迁移
Logstash	源端版本 > 目标端时，或迁移过程中需要定制化改造的场景	• AWS 云 Opensearch • 华为云 Opensearch • AWS/阿里/华为云 ES	Logstash 方案需全程停服，停服时间与集群资源性能、专线带宽等因素有关			☓	✓	可以根据文档自助迁移
在线融合（腾讯云 ES 特有）	在线不停服迁移	• 自建 ES	无需停服	✓	✓	✓	☓	需腾讯云 ES 后台配合

概述

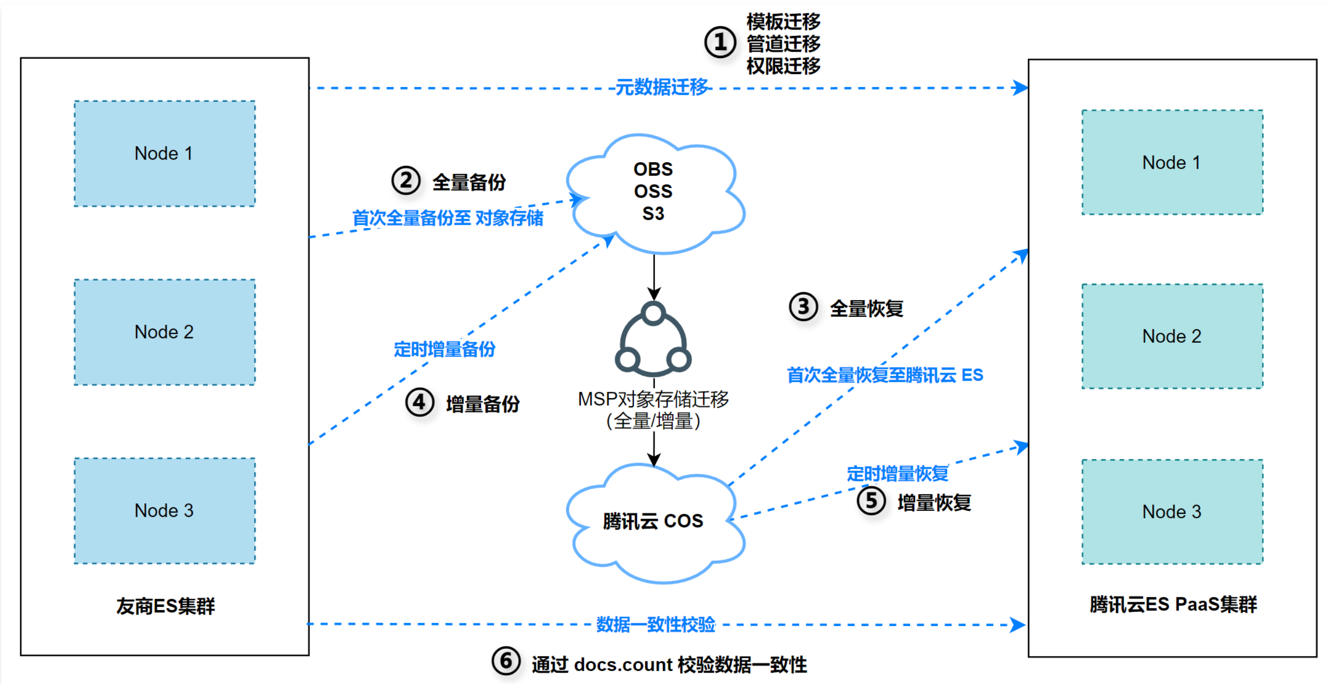
本迁移方案旨在为用户提供一种从自建 ES 或友商云 ES 迁移到腾讯云 ES 的实践教程。本方案将详细介绍使用 Snapshot 快照方案的迁移流程、迁移步骤以及注意事项等方面的内容。通过本方案，用户可以在保证数据完整性、安全性和高效性的前提下，顺利完成迁移过程。

方案概览

- 从自建 ES 迁移至腾讯云 ES 集群方案概览：



- 从友商云 ES 迁移至腾讯云 ES 集群方案概览：



准备工作

账号开通

客户需要开通一个 腾讯云账号 。

网络打通

客户的 ES 集群需要有一条能访问腾讯云服务的专线网络或 VPN。

- 专线：专线接入
- VPN：建立 VPC 到 IDC 的连接（SPD 策略）

> **注意：**
> 数据迁移务必使用内网，否则默认会走公网进行数据传输，公网传输存在数据泄漏风险和潜在危险。

存储桶购买

⚠ **注意：**
- 需要购买和腾讯云 ES 同地域的 COS 存储桶。
- AWS Opensearch 仅支持 Logstash 方案。

ES 集群源平台	备份仓库	目标仓库
自建ES	COS	COS（与源端同仓库）
阿里云 ES 集群	OSS	COS
华为云 ES 集群	OBS	COS
AWS 云 ES 集群	S3	COS

ES 集群创建

前置确认

确认 ES 集群搭建前基础信息

信息编号	确认信息	确认值（此列需要客户确认）	备注
1	区域	华南地区-广州	ES 集群所在的区域
2	产品版本	ES 7.14.2	ES 集群版本 全文检索建议7.14.2，向量检索推荐8.13.3
3	组件种类	es，logstash，filebeat	ES 集群需要包含的组件
4	鉴权	不开启	白金版不支持关闭鉴权，非白金版支持开启/关闭（需提工单支持）
5	业务场景	全文检索，向量检索	根据业务场景决定集群版本 全文检索建议<=7.14.2，向量检索推荐>8
6	集群网络&可用区	• 集群网络举例：Default-VPC • 子网举例：Default-subnet	集群所在的 VPC，和可用区子网需要提前规划好 有容灾需求可选择多可用区，节点将均匀分布在所选择的多个可用区，可用节点总量不变
7	安全组	安全组举例：sg-xxxx	ES 集群安全组，如需关联安全组，可在集群创建后进行关联

确认 ES 集群资源配置

生产环境的规格参考实际的 ES 业务需求，有特殊机型要求的，需提早申请，以便腾讯侧备货，腾讯云建议基础的 ES 生产环境配置如下：

节点类型	节点说明	推荐配置	节点数量
热数据层	• 默认节点类型，可搭配温数据层实现冷热分离架构； • 数据写入层。通常将经常查询的数据放置在热层，以提升查询与写入效率。	标准型 S1 CPU：8核 内存：32GB 数据盘： 增强型 SSD 云硬盘 500G x 3	N
温数据层	• 可选节点，支持仅开启温数据层，也可搭配热数据层实现冷热分离架构； • 将访问频率较低且更新频率较低的数据存放在温层，以节约成本。非必要，有冷热分层需求才需要开启。	标准型 S1 CPU：4核 内存：16GB 数据盘： 高性能云硬盘 1000G x 3	N
专用主节点	• 可选节点，建议10节点以上或分片规模>5000时启用； • 管理节点，保证集群的调度正常进行。非必要，默认所有数据节点都是候选主节点。当集群节点数>10 或 数据量较大时建议选购专用主节点。	标准型 S1 CPU：4核 内存：16GB	3
协调节点	• 可选节点，默认所有数据节点都是协调节点。建议有特殊需求时候启用；	标准型 S1 CPU：4核 内存：16GB	N

	• 协调节点只用于处理客户端请求，不存储数据，能够提升集群处理性能和服务稳定性，适用于高并发读写、多聚合查询等 CPU 密集型业务场景。		
机器学习节点	• 可选节点，有推理需求时启用； • 机器学习节点用于创建机器学习任务，自动进行数据分析，识别异常数据。也可在此节点加载向量模型，提升向量生成和向量检索能力。	标准型 S1 CPU：16核 内存：32GB	N
Kibana 节点	默认开启，用于数据可视化，1核2G规格免费，支持购买更高规格。	标准型 S1 CPU：1核 内存：2GB	1

集群购买

购买腾讯云 ES 集群，请参见 创建集群 。

插件安装及词典上传（必要）

在进行数据迁移前，需要确认索引是否使用自定义插件及自定义词典，如果有则需要在云上 ES 集群同步，否则相关索引数据恢复无法成功。

插件查询

使用下面命令查询自建集群使用的插件：

```
curl -s -uelastic:'源端集群密码' 'http://源端IP:9200/_cat/plugins?h=component,version' | sort | uniq
```

插件安装

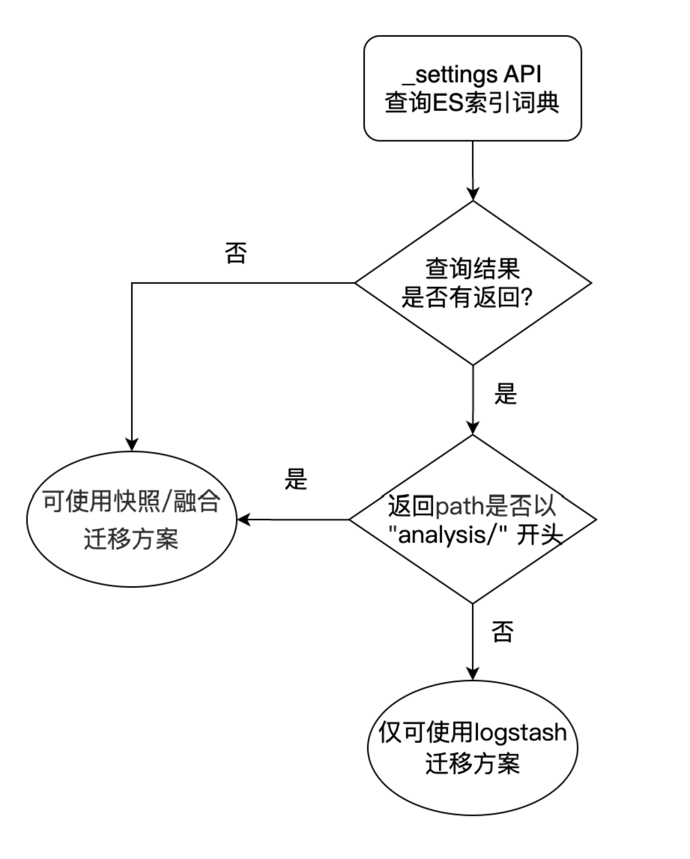
腾讯云 ES 支持的插件列表及插件安装可以参考 腾讯云 Elasticsearch Service 插件列表，如果有另外的自定义插件需要安装可以提工单联系腾讯云侧使用此功能。

词典查询

自定义词典查询

```
PUT _snapshot/my_backucurl -s -uelastic:'源端集群密码' 'http://源端IP:9200/_settings?pretty' | grep "_path"
```

自定义词典查询流程图



IK 分词/停用词词典查询

```
# 进入到ES主目录，$ES_HOME需替换成实际目录
cd /$ES_HOME/config/analysis-ik

# 进入ik词典目录下，查看配置文件
cat IKAnalyzer.cfg.xml
```

未配置任何 ik 词典则默认内容如下，无需更新 ik 词典。如自定义了 ik 词典，则需上传至腾讯云 ES。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
        <comment>IK Analyzer 扩展配置</comment>
        <!--用户可以在这里配置自己的扩展字典 -->
        <entry key="ext_dict"></entry>
         <!--用户可以在这里配置自己的扩展停止词字典-->
        <entry key="ext_stopwords"></entry>
        <!--用户可以在这里配置远程扩展字典 -->
        <!-- <entry key="remote_ext_dict">words_location</entry> -->
        <!--用户可以在这里配置远程扩展停止词字典-->
        <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
```

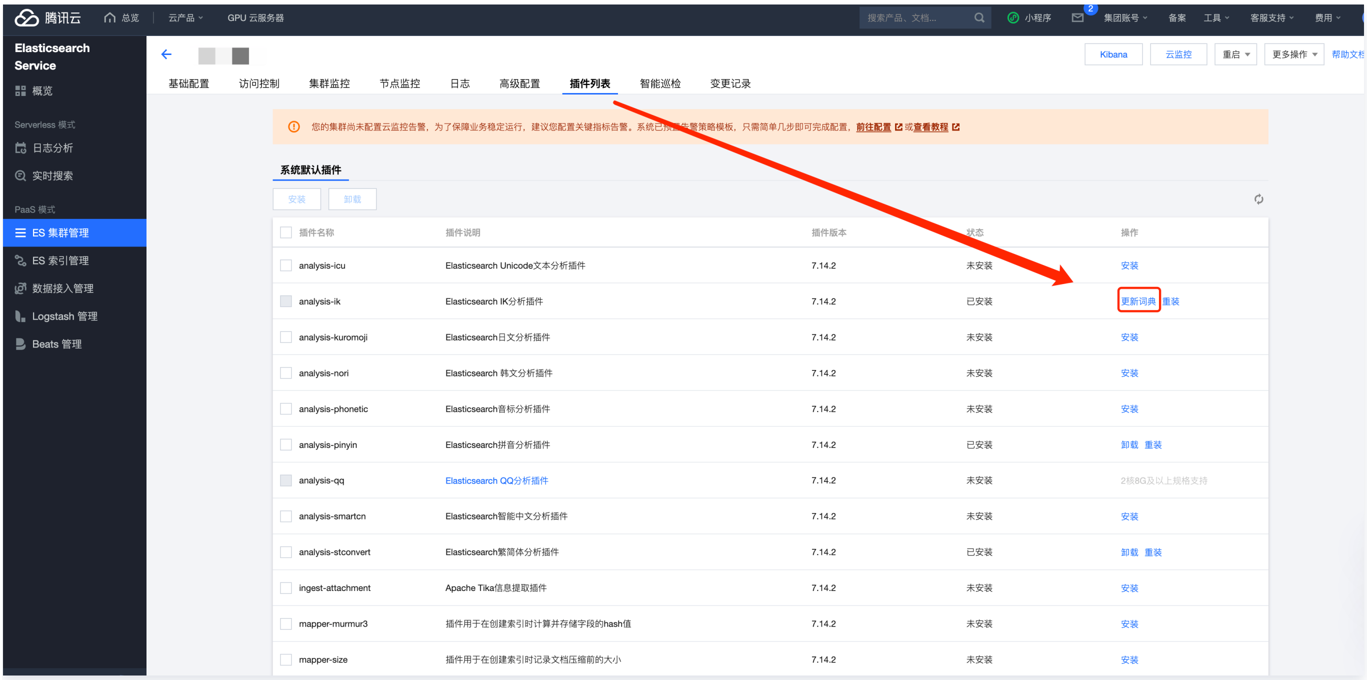
词典上传

- **同义词词典更新**

进入 腾讯云 ES 控制台 > **ES 集群管理** > **ES 实例** > **高级配置**界面，单击**更新词典**，上传自定义词典，配置索引属性即可使用。

- **IK 分词/停用词词典更新**

进入 腾讯云 ES 控制台 > **ES 集群管理** > **ES 实例** > **插件列表**界面，选择 IK 插件单击**更新词典**，上传分词词典后即可。



数据迁移

方案介绍

snapshot api 是 ES 提供的一组用于索引数据快速备份及恢复的 api 接口，详细介绍可查看 官方文档，利用该组 api 接口，我们可以通过将源 ES 集群的数据备份到远程仓库，再在腾讯云 ES 集群上从远程仓库中通过快照恢复的方式来实现跨集群的离线数据迁移。

	Cluster version				
Index creation version	6.8	7.0–7.1	7.2–7.17	8.0–8.2	8.3-8.14
5.0–5.6	✓	✗	✗	✗	✓[1]
6.0–6.7	✓	✓	✓	✗	✓[1]
6.8	✓	✗	✓	✗	✓[1]
7.0–7.1	✗	✓	✓	✓	✓
7.2–7.17	✗	✗	✓	✓	✓
8.0–8.14	✗	✗	✗	✓	✓

详情请参见 快照版本关系说明 。

关于快照迁移的版本限制：

- 快照备份/恢复功能只支持5.x以上。
- 目标 ES 集群的版本号要大于等于源 ES 集群的版本号。例如，客户的源端集群版本号为6.4.3，则腾讯云 ES 集群的版本需>= 6.4.3。
- 目标 ES 集群主版本不能超过源端 ES 集群两个及以上的版本。例如6.x的集群只能恢复到7.x上，再恢复到8.x，不能直接恢复到8.x。

适用场景：

snapshot 迁移适用于集群规模较大、索引个数比较多的场景。

迁移流程介绍：

序号	迁移步骤	详细说明
1	元数据迁移	使用同步工具进行元数据迁移，包括： • 模板迁移 • 管道迁移 • 权限迁移
2	全量备份	• 备份期间业务无明显感知，一般建议在业务低峰进行； • 自建 ES 迁移：直接备份到 COS 进行恢复； • 友商云 ES 迁移，备份到友商对象存储（OBS/OSS/S3），然后通过 MSP 工具迁移到 COS 进行恢复； • 备份/恢复速度与专线带宽、集群负载有关。
3	全量恢复	
4	增量备份	• 需多次执行，保证最短割接窗口； • 除最后一次增量备份外，其他增量备份场景业务无明显感知，一般建议在业务低峰进行； • 最后一次增量备份/恢复需要停服进行，通过多次增量备份/恢复，保证大部分场景下停服时间<=20min。
5	增量恢复	
6	数据一致性校验	数据同步完成后，可以通过 docs.count 来校验数据一致性。

用户及角色迁移

- 如需迁移用户及权限，可使用工具一键迁移：

```
rpm -vih https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/es-security-sync-lastest.x86_64.rpm
```

- 或下载后上传再使用 rpm 安装：

```
rpm -vih es-security-sync-lastest.x86_64.rpm
```

> **说明：**
> security 功能 ES 6 版本才开始支持，ES 8 的 security 索引不允许修改。所以仅支持 6 到 7，以及 7 到 6 的迁移。

迁移示例

```
es-security-sync \
  --src_url http://10.0.xx.xx:9200 \
  --dst_url http://10.0.xx.xx:9400 \
  --src_user log_user \
  --dst_user log_user \
  --src_password ****** \
  --dst_password ******
```

索引模板迁移

- 索引模板为 ES 业务使用中最常用的功能，在迁移前务必先同步索引模板。模板迁移使用工具一键迁移即可：

```
rpm -vih https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/es-template-cp-lastest.x86_64.rpm
```

- 或下载后上传再使用 rpm 安装：

```
rpm -vih es-template-cp-lastest.x86_64.rpm
```


迁移示例

```
es-template-cp \
  --src_url http://源ES IP:9200 \
  --dst_url http://腾讯云ES IP:9200 \
  --src_password '******' \
  --dst_password '腾讯云ES密码'
```

集群管道迁移（按需操作）

ES 管道迁移使用工具一键迁移即可。

```
#下载jar包
#运行环境：JDK-1.8
wget https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/CopyPipelines.jar
```

示范样例

> **注意：**
> 任何参数不要空缺，无密码集群可指定任意密码。

- **样例一**

源端，目标端，集群带密码，跳过目标端已存在的同名管道。

```
java -jar CopyPipelines.jar http://源ES IP:9200 elastic '源端ES密码' http://腾讯云ES IP:9200 elastic '腾讯云ES密码'  skip
```

- **样例二**

源端，目标端，集群带密码，覆盖目标端已存在的同名管道。

```
java -jar CopyPipelines.jar http://源ES IP:9200 elastic '源端ES密码' http://腾讯云ES IP:9200 elastic '腾讯云ES密码' overwrite
```

安装 COS 插件（该步骤为自建 ES，友商云 ES 可跳过）

1. 创建快照前需要先注册仓库，一个仓库可以存储多个数据快照文件。快照存放于 腾讯云 COS 对象存储 中，以插件形式支持，需要安装插件。
2. 其中 COS 插件的 GitHub 地址：elasticsearch-repository-cos releases，根据自建版本下载对应版本的 COS 插件包。对于 GitHub 上没有的版本，可以通过 提交工单 进行索要。
3. 从客户自建 ES 集群中迁移到腾讯云 ES，我们可以直接使用 COS 进行迁移，如果客户集群没有安装过 COS 插件，则需要先安装 COS 插件，必须使用 ES 运行用户执行，且所有节点都需要安装：

   ```
   bin/elasticsearch-plugin install file:///tmp/elasticsearch-cos-x.x.zip
   ```

```
[es@VM-10-2-tencentos ~]$ cd elasticsearch-7.10.1/
[es@VM-10-2-tencentos elasticsearch-7.10.1]$ bin/elasticsearch-plugin install file:///home/es/repository-cos-7.10.1.zip
-> Installing file:///home/es/repository-cos-7.10.1.zip
-> Downloading file:///home/es/repository-cos-7.10.1.zip
[=================================================] 100%
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@     WARNING: plugin requires additional permissions     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
* java.lang.RuntimePermission accessDeclaredMembers
* java.lang.RuntimePermission getClassLoader
* java.net.SocketPermission * connect,resolve
* java.util.PropertyPermission es.allow_insecure_settings read,write
See http://docs.oracle.com/javase/8/docs/technotes/guides/security/permissions.html
for descriptions of what these permissions allow and the associated risks.

Continue with installation? [y/N]y
-> Installed repository-cos
[es@VM-10-2-tencentos elasticsearch-7.10.1]$
```

4. 所有节点安装并滚动重启完成后，执行 cat api 查看插件情况：

```
curl -s -uelastic:'源端密码' 源端IP:9200/_cat/plugins?h=component,version | grep "repository-cos" | uniq -c
```

5. 查看节点数：

```
curl -s -uelastic:'源端密码' 源端IP:9200/_cat/nodes | wc -l
```

6. 在返回结果中确认 COS 插件个数是否为总节点个数，否则代表有节点未重启或未安装插件。

```
[root@VM-67-54-tencentos ~]#
[root@VM-67-54-tencentos ~]# curl -s -uelastic:****** 127.0.0.1:9200/_cat/plugins?h=component,version | grep "repository-cos" | uniq -c
     23 repository-cos     7.14.2
[root@VM-67-54-tencentos ~]# curl -s -uelastic:****** 127.0.0.1:9200/_cat/nodes | wc -l
23
[root@VM-67-54-tencentos ~]#
```

注意：

- 插件 zip 包无需解压，直接安装即可。
- 安装时指定的 file:/// 协议是三条斜杠，其中前面两个 // 是协议与路径的分隔符，后面一个 / 是绝对路径开头的根目录。
- 待安装的插件zip包不可放在es目录下，可以放在/tmp或者家目录，否则可能会安装失败。
- 执行安装命令时，不可位于 es bin/ 路径下，需要在 es bin/ 路径外面执行，否则可能会安装失败。

ES 集群注册快照仓库

自建 ES（友商/腾讯云/IDC）

打通 COS 内网（必要）

前提条件

自建 ES 在注册快照仓库前，有如下必要操作。

1. 参考文档，打通 COS 内网 内网访问 COS 操作步骤-V2.0，并配置好 hosts 解析。
2. 测试 COS 内网解析，例如 ping。
3. 重启自建 ES 所有节点。

该步骤务必使用内网，否则默认会走公网进行数据传输，公网传输存在数据泄露风险和潜在危险。

4. COS 内网打通并重启ES后，即可开始如下操作：
   4.1 获取 access 密钥信息：登录 API 密钥管理控制台，调用 snapshot 的 api 注册 repository 仓库，cos snapshot API 需要区分 ES<8.x 和 ES>=8.x。
   4.2 region 参数说明：请参见 Elasticsearch Service 地域参数。

ES 集群版本低于8.0注册方式为：

```
PUT _snapshot/my_backup
{
```

```
  "type": "cos",
  "settings": {
    "app_id": "xxxxxxx",
    "access_key_id": "xxxxxx",
    "access_key_secret": "xxxxxxx",
    "end_point": "cos-internal.ap-guangzhou.tencentcos.cn",   //需替换成实际region，例如ap-shanghai
    "bucket": "xxxxxx", // cos bucket后面生成的 -appid 不可以填进来
    "region": "ap-guangzhou",  //需替换成COS桶所在region，例如ap-shanghai
    "base_path": "my_backup",
    "max_restore_bytes_per_sec":"40m",  // 每个节点的恢复速率，可按需调整，建议最大不超过200
    "max_snapshot_bytes_per_sec":"40m",  // 每个节点的备份速率，可按需调整，建议最大不超过200
    "compress": true,
    "chunk_size": "500mb"

  }
}
```

ES 集群版本高于等于8.0注册方式为：

```

PUT _snapshot/my_backup
{
  "type": "cos",
  "settings": {
    "cos": {
      "client": {
        "app_id": "xxxxxxx",
        "access_key_id": "xxxxxx",
        "access_key_secret": "xxxxxxx",
        "end_point": "cos-internal.ap-guangzhou.tencentcos.cn",   //需替换成实际region，例如ap-shanghai
        "bucket": "xxxxxx",
        "region": "ap-guangzhou",  //需替换成COS桶所在region，例如ap-shanghai
        "base_path": "my_backup"
        }
    },
    "max_restore_bytes_per_sec": "40m",     // 每个节点的恢复速率，可按需调整,建议最大不超过200
    "max_snapshot_bytes_per_sec": "40m",    // 每个节点的备份速率，可按需调整，建议最大不超过200
    "compress": true,
    "chunk_size": "500mb"
  }
}
```

返回 true 即为注册成功，友商云 ES 也同理

```
#! Deprecation: cos repository bucket already contain app_id, and app_id will not be supported for the cos repository in future releases
#! Deprecation: cos repository base_path trimming the leading `/`, and leading `/` will not be supported for the cos repository in future releases
{
  "acknowledged" : true
}

```

阿里云 ES

调用 snapshot 的 api 创建 repository 仓库。

```
PUT _snapshot/my_backup/
{
    "type": "oss",
    "settings": {
        "endpoint": "http://xxx.aliyuncs.com", // oss 的 Endpoint 地址
        "access_key_id": "xxxx", // 用户密钥，可通过控制台右上角的头像获取—“AccessKey 管理”
        "secret_access_key": "xxxxxx", // access_key
        "bucket": "xxxxxx", // bucket name，创建 bucket 时的命名
        "compress": true, // 是否压缩源数据大小
        "chunk_size": "500mb", // 数据分块大小
        "base_path": "my_backup", // 仓库起始路径
        "max_restore_bytes_per_sec":"40m",  // 每个节点的恢复速率，可按需调整
        "max_snapshot_bytes_per_sec":"40m"  // 每个节点的备份速率，可按需调整

    }
}
```

华为云 ES

调用 snapshot 的 api 创建 repository 仓库。

```
PUT _snapshot/my_backup/
{
    "type": "obs",
    "settings": {
        "endpoint": "obs.xxx.myhuaweicloud.com", // obs 的 Endpoint 地址
        "access_key": "xxxx", // 用户密钥，可通过控制台右上角的头像获取—"我的凭证"—"访问密钥"
        "secret_key": "xxxxxx", // secret_key
        "bucket": "xxxxxx", // bucket name，创建 bucket 时的命名
        "compress": true, // 是否压缩源数据大小
        "chunk_size": "500mb", // 数据分块大小
        "base_path": "my_backup", // 仓库起始路径
        "max_restore_bytes_per_sec":"40m",  // 每个节点的恢复速率，可按需调整
        "max_snapshot_bytes_per_sec":"40m"  // 每个节点的备份速率，可按需调整
    }
}
```

AWS 云 ES

1. 创建 IAM 角色。
2. 创建 AWS 云 ES 仓库。
3. 调用 snapshot 的 api 创建 repository 仓库。

```
PUT _snapshot/my_backup
{
  "type": "s3",
  "settings": {
    "bucket": "s3-bucket-name",
    "base_path": "my/snapshot/directory",
    "region": "region",
    "role_arn": "arn:aws:iam::账号ID:role/TheSnapshotRole"
  }
}
```

源端 ES 集群执行创建快照

将源端 ES 集群的索引数据备份到 COS 的 my_backup 仓库下，命名为：snapshot_1。

```
PUT _snapshot/my_backup/snapshot_1
{
  "indices":"*,-.*,-ilm-*"
}
```

Body 内容解释

indices 值	含义
*	所有索引
-.*	排除所有以 . 开头的索引，例如，系统索引
-ilm-*	排除所有以 ilm- 开头的索引，例如，索引生命周期系统索引

当备份索引数据较少时，执行完上面的命令后会立即返回，当备份索引数量较大时，该 api 大概率会超时，但不会失败，数据的备份是异步执行的。

备份指定索引

上面备份的是整个索引，如果我们只想将部分索引或分批迁移到云上，也可以通过下面的命令：

```
PUT _snapshot/my_backup/snapshot_1
{
    "indices": "index_1,index_2,log_*,nginx-*"
}
```

如果源端存在 closed 状态的索引，则无法通过通配符进行备份，需要写全索引名称，多个索引以逗号分隔；

查看快照备份进度

我们可以通过如下的命令查看快照执行的状态：

```
GET _snapshot/my_backup/snapshot_1?filter_path=snapshots.state
```

state 若返回的是 "SUCCESS"，则表示快照备份完成。若是 "IN_PROCESS"，则表示快照还在备份中。也可以通过该工具获取详细的备份进度：

```
rpm -vih https://tools-release-1253240642.cos.ap-shanghai.myqcloud.com/elasticsearch/packages/es-backup-progress-lastest.x86_64.rpm
```

或下载后上传再使用 rpm 安装：

```
rpm -vih es-backup-progress-lastest.x86_64.rpm
```

使用方式：

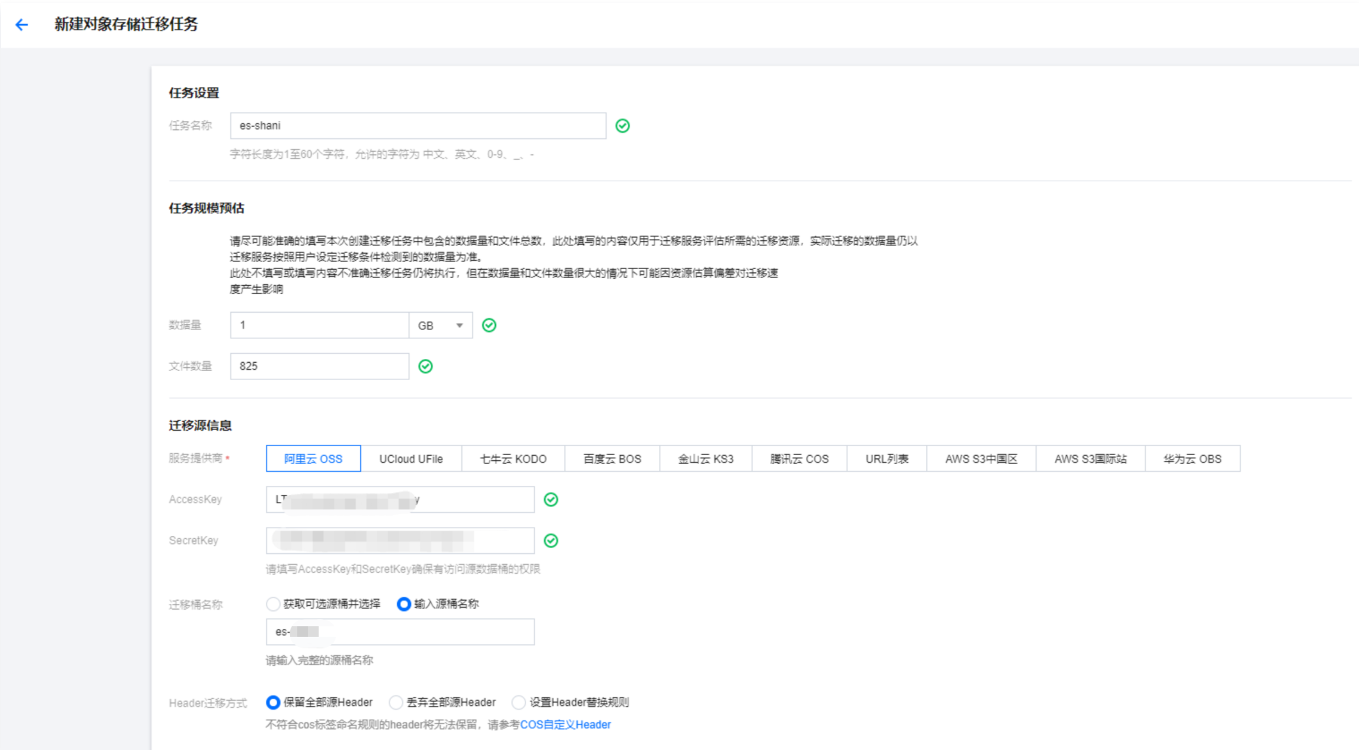
- ES 实例有密码认证，指定用户名。
- ES 实例指定端口。

获取指定仓库以及快照信息。

```
es-backup-progress \
  --ip 源ES IP \
  --port 9201
  --user 'xxx' \
  --password '源ES 密码' \
  --repository my_backup \
  --snapshot s1
```

数据迁移（该步骤为友商云 ES，自建 ES 可跳过）

使用腾讯云提供的传输工具 对象存储迁移，将存储在阿里云 OSS/华为云 OBS/AWS云 S3 中的数据传输至腾讯云 COS：



迁移模式务必选择 Agent 半托管模式，使用内网迁移数据，否则默认会走公网进行数据传输，公网传输存在数据泄露风险和潜在危险，如果需要使用公网迁移数据，请提工单联系腾讯侧。

在腾讯云 ES 集群上注册快照仓库

登录 API 密钥管理控制台 获取 access 密钥信息。

ES 集群版本低于8.0注册方式为：

```
PUT _snapshot/my_backup
{
  "type": "cos",
  "settings": {
    "app_id": "xxxxxxx",
    "access_key_id": "xxxxxx",
    "access_key_secret": "xxxxxxx",
    "bucket": "xxxxxx", // cos bucket后面生成的 -appid 不可以填进来
    "region": "ap-guangzhou",  //需替换成COS桶所在region，例如ap-shanghai
    "compress": true,
    "chunk_size": "500mb",
    "base_path": "my_backup",
    "max_restore_bytes_per_sec":"40m",  // 每个节点的恢复速率，可按需调整
    "max_snapshot_bytes_per_sec":"40m",  // 每个节点的备份速率，可按需调整
  }
```

```
}
```

ES 集群版本高于等于8.0注册方式为：

```
PUT _snapshot/my_backup
{
  "type": "cos",
  "settings": {
    "cos": {
      "client": {
        "app_id": "xxxxxxx",
        "access_key_id": "xxxxxx",
        "access_key_secret": "xxxxxxx",
        "bucket": "xxxxxx",
        "region": "ap-guangzhou",  //需替换成COS桶所在region，例如ap-shanghai
        "base_path": "my_backup"
      }
    },
    "max_restore_bytes_per_sec": "40m",
    "max_snapshot_bytes_per_sec": "40m",
    "compress": true,
    "chunk_size": "500mb"
  }
}
```

- app_id：腾讯云账号 APPID。
- access_key_id：腾讯云 API 密钥 SecretId。
- access_key_secret：腾讯云 API 密钥 SecretKey。
- bucket：COS Bucket 名字，名字不能带 -{appId}后缀。
- region：COS Bucket 地域，此地域必须与 ES 集群为同一地域。地域编码可参考 地域和可用区 。
- base_path：备份目录。

发起快照后快照备份速率与恢复速率可以修改（需版本 7.10.x 后才支持）：

```
PUT _snapshot/my_backup
{
  "type": "cos",
  "settings": {
    "app_id": "xxxxxxx",
    "access_key_id": "xxxxxx",
    "access_key_secret": "xxxxxxx",
    "bucket": "xxxxxx", // cos bucket后面生成的 -appid 不可以填进来
    "region": "ap-guangzhou",  //需替换成COS桶所在region，例如ap-shanghai
    "compress": true,
    "chunk_size": "500mb",
    "base_path": "my_backup",
    "max_restore_bytes_per_sec":"100m",  // 节点恢复速率改为100mb/s
    "max_snapshot_bytes_per_sec":"100m",  // 节点备份速率改为100mb/s
  }
}
```

注意：

- 由于 ES 自身问题，7.14 起不再动态加载仓库的快照列表，当源端仓库发生变化时，目标端仓库需要重新注册，以加载最新的快照列表。
- 注意，该特殊操作只在目标端执行，如下所示：

```
# 第一次同步
PUT _snapshot/my_backup
{
  "type": "cos",
  // 省略
}

# 第二次同步
DELETE _snapshot/my_backup
PUT _snapshot/my_backup
{
  "type": "cos",
  // 省略
}
```

另外，如提前在目标集群注册了仓库，再去源端打快照，也会出现快照无法正常加载的情况。遇到这类问题，重新注册仓库即可。

云上 ES 集群执行恢复命令

本步骤为从 COS 仓库中恢复快照，快照恢复分为无冷热场景和冷热分层场景，选择其中一种执行即可，如果您的 ES 集群未对节点和索引做过任何冷热属性配置，那您的集群属于无冷热分层场景。

无冷热分层场景

```
POST _snapshot/my_backup/snapshot_1/_restore
{
  "indices":"*,-.*,-ilm-*"
}
```

冷热分层场景（如非冷热集群，忽略本操作）

- 指定热节点恢复（如果非冷热集群，忽略本操作）：

```
// 腾讯云 ES < 7.14版本命令如下:
POST _snapshot/my_backup/snapshot_1/_restore
{
  "indices":"{hot_index}*,-.*,-ilm-*",
  "index_settings": {
    "index.routing.allocation.require.temperature": "hot"
    },
  "ignore_index_settings": [
    "index.routing.allocation.require.temperature"
  ]
}

// ES >= 7.14版本命令如下:
POST _snapshot/my_backup/snapshot_1/_restore
{
  "indices":"{hot_index}*,-.*,-ilm-*",
  "index_settings": {
    "index.routing.allocation.include._tier_preference": "data_hot,data_warm,data_cold"
    },
  "ignore_index_settings": [
    "index.routing.allocation.require.temperature"
  ]
}
```

- 指定冷节点恢复（如果非冷热集群，忽略本操作）：

```
// 腾讯云 ES < 7.14版本命令如下：
POST _snapshot/my_backup/snapshot_1/_restore
{
  "indices":"{warm_index}*,-.*,-ilm-*",
  "index_settings": {
    "index.routing.allocation.require.temperature": "warm"
    }
}

// 腾讯云 ES >= 7.14版本命令如下：
POST _snapshot/my_backup/snapshot_1/_restore
{
  "indices":"{warm_index}*,-.*,-ilm-*",
  "index_settings": {
    "index.routing.allocation.include._tier_preference": "data_warm,data_hot,data_cold"
    }
}
```

上面这一步就是将 COS 中刚备份的快照数据恢复到云上的 ES 集群中，执行这一步的前提是需要先在云上的 ES 集群中注册仓库，地址和源端集群上一致。

从快照中恢复指定索引：

```
POST /_snapshot/my_backup/snapshot_1/_restore
{
 "indices": "index_1",
 "index_settings": {
   "index.routing.allocation.include._tier_preference": "data_warm,data_hot,data_cold"
   }
}
```

查看快照恢复进度

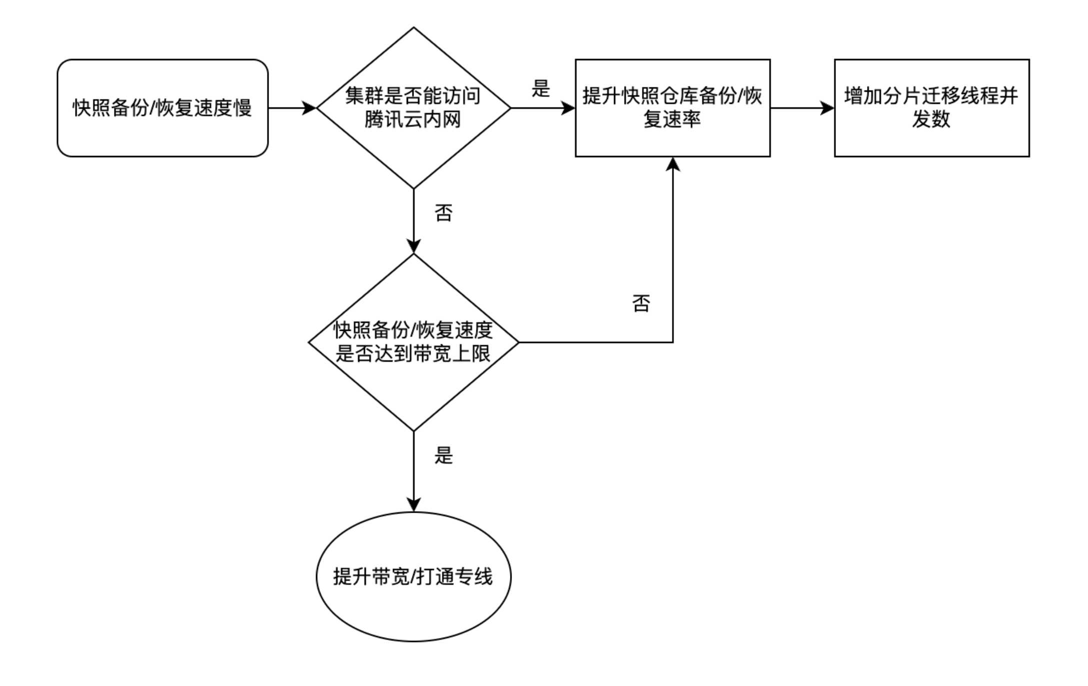
active_shards_percent_as_number 为快照恢复进度百分比。

> **说明：**
> 当集群分片很少，或者分片 size 比较大的时候，这个进度变化会非常缓慢，是正常现象。

```
GET _cluster/health?filter_path=active_shards_percent_as_number
```

备份/恢复加速

- **快照备份/恢复速度慢**



更改恢复速度（谨慎操作），此操作在集群数据恢复时对集群有一定压力：

- **更改快照仓库的备份/恢复速率**

> ⚠ **注意：**
> ES 7.10 之前不支持动态修改，如果备份/恢复已经开始，这个 API 会拒绝请求，直到备份/恢复完成后才允许更新。
> 修改仓库参数时需谨慎，只需调整恢复速度，不要改动其他参数，否则可能会造成恢复异常。

```
PUT _snapshot/my_backup
{
  "type": "cos",
  "settings": {
    "app_id": "xxxxxxx",
    "access_key_id": "xxxxxx",
    "access_key_secret": "xxxxxxx",
    "bucket": "xxxxxx",
    "region": "ap-guangzhou",   //需替换成COS桶所在region，例如ap-shanghai
    "compress": true,
    "chunk_size": "500mb",
    "base_path": "my_backup",
    "max_restore_bytes_per_sec":"100m",   // 每个节点的恢复速率，可按需调整
    "max_snapshot_bytes_per_sec":"100m"   // 每个节点的备份速率，可按需调整
  }
}
```

- **更改集群分片迁移线程并发数**

```
PUT _cluster/settings
{
    "transient":{
        "cluster.routing.allocation.node_concurrent_recoveries":10,
        "cluster.routing.allocation.node_concurrent_incoming_recoveries":10,
        "cluster.routing.allocation.node_initial_primaries_recoveries":10,
        "cluster.routing.allocation.node_concurrent_outgoing_recoveries":10,
        "cluster.routing.allocation.cluster_concurrent_rebalance":10,
        "indices.recovery.max_bytes_per_sec":"100mb"
```

```
    }
}
```

- **恢复完成后，需要回滚加速操作**

```
PUT _cluster/settings
{
    "transient":{
        "cluster.routing.allocation.node_concurrent_recoveries":null,
        "cluster.routing.allocation.node_concurrent_incoming_recoveries":null,
        "cluster.routing.allocation.node_initial_primaries_recoveries":null,
        "cluster.routing.allocation.node_concurrent_outgoing_recoveries":null,
        "cluster.routing.allocation.cluster_concurrent_rebalance":null,
        "indices.recovery.max_bytes_per_sec": null
    }
}
```

增量备份/恢复

增量备份

增量备份就是在原有的快照仓库里，新建一个快照，api 完全一致。

例如：

第一次备份快照

```
PUT _snapshot/my_backup/snapshot_1
{
  "indices":"*,-.*,-ilm-*"
}
```

第二次备份快照

```
PUT _snapshot/my_backup/snapshot_2
{
  "indices":"*,-.*,-ilm-*"
}
```

则第二次备份就是基于第一次全量的增量备份。

增量数据迁移

重复 数据迁移 的步骤。

增量恢复

在目标集群的增量恢复，需要先_close目标需要恢复的索引，**关闭索引后即可进行增量恢复**，在执行增量恢复时会自动 reopen。

关闭全量索引（谨慎操作）

此操作是在目标端集群执行，而非源端：

```
POST *,-.*/_close
```

如果报错不允许模糊匹配，可以开启一下通配权限：

```
{
  "error": {
    "root_cause": [
      {
        "type": "illegal_argument_exception",
        "reason": "Wildcard expressions or all indices are not allowed"
      }
    ],
    "type": "illegal_argument_exception",
    "reason": "Wildcard expressions or all indices are not allowed"
  },
  "status": 400
}
```

```
PUT _cluster/settings
{
  "transient": {
    "action.destructive_requires_name": "false"
  }
}
```

增量恢复

和全量恢复 API 完全一致，只需更改快照名称，将快照名称改为最新的快照名称：

```
POST _snapshot/my_backup/snapshot_2/_restore
{
  "indices":"*,-.*,-ilm-*"
}
```

reopen

1. 确认全量/增量恢复完成后，再补一个全量 _open，以防止有未发起恢复的索引没有被 _open：

```
POST */_open
```

2. 操作完之后需要回滚，由于模糊匹配权限有安全性问题，所以必须关闭：

```
PUT _cluster/settings
{
  "transient": {
    "action.destructive_requires_name": "true"
  }
}
```

注意（重要，重点阅读）

- 快照在进行中时尽量避免 DELETE snapshot，该操作非常重，可能会对集群稳定性造成影响，例如引起索引 shard lock。另外删除快照还有如下问题：
  - 快照在进行中时，DELETE 快照很大几率会超时或无法成功，只能通过重启集群解决。
  - 如已发起快照删除，在删除操作完成之前，无法发起新的快照请求。
- 最后一次增量前的切割操作，建议是业务主动停写。如果是采用 index block write / read only 的方式停写，该索引属性会被同步到快照里，在恢复时也会携带禁写属性，所以在恢复后需要回滚禁写属性，否则会导致业务不可写。
- 在日志场景有 rollover 的情况下，如果 is_write_index 属性发生变化，则会发生恢复失败的问题，具体表现为日志里会抛异常：

```
Caused by: java.lang.IllegalStateException: alias [xxxxx] has more than one write index [xxxxx-2023.10.25-
000001, xxxxx-2023.10.26-000002]
```

解决方案为取消目标端索引旧的 is_write_index 属性：

```
POST _aliases
    {
        "actions" : [
            {
                "add" : {
                     "index" : "xxxx-2023.10.25-000001",
                     "alias" : "xxxx",
                     "is_write_index" : false
                }
            }
        ]
    }
```

- 在日志场景有 rollover 的情况下，建议业务切割上云之后，再同步 ilm 任务，否则会发生目标集群 ilm 提前触发 rollover 的可能，也会导致 is_write_index 属性冲突问题；

数据校验

可以下载数据条数校验工具进行对比，工具仅对比两端集群索引文档数量，不会校验内容是否相同。

工具安装

```
#下载jar包
#运行环境: JDK-1.8
wget https://tools-release-1253240642.cos.ap-
shanghai.myqcloud.com/elasticsearch/packages/ElasticsearchCompare.jar
```

使用案例

示范样例

共需要如下输入7个参数，任何参数不能空缺，无密码集群可指定任意密码。

参数名	解释	示范样例
srcClusterUrl	源端集群地址	http://127.0.0.1:9200
srcUser	源端集群用户名	elastic
srcPassword	源端集群密码	12345
dstClusterUrl	目标端集群地址	http://127.0.0.1:9201
dstUser	目标端集群用户名	elastic
dstPassword	源目标端集群密码	12345
index pattern	需要校验的索引的匹配规则	"*,-.*"

```
java -jar ESIndexDocumentCountComparison.jar <srcClusterUrl> <srcUser> <srcPassword> <dstClusterUrl>
<dstUser> <dstPassword> <index pattern>
```

样例

源端，目标端，集群带密码，需要校验两端集群全部索引的数据一致性，**如果集群无密码则可输入任意密码。**

```
java -jar ESIndexDocumentCountComparison.jar http://源端ES IP:9200 elastic '源端ES密码' http://腾讯云ES IP:9200
elastic '腾讯云ES密码'  "*,-.*"
```

返回说明

工具首先会返回源端集群的 close 索引，close 状态索引无法检验数据一致性，然后返回对于两边集群数据不一致的索引信息，不返回数据一致的索引信息。

- 数据不一致返回

```
(base) [root@VM-10-2-tencentos ~]# java -jar ElasticsearchCompare.jar http://          :9201 elastic any_password
 https://          public.tencentelasticsearch.com:9200 elastic              "*,-.*"

以下是源端集群Close状态的索引列表，无法获取文档数进行数据一致性对比：
[test, test1]

以下是源目两端集群文档数量不一致的索引列表：
索引名称                                    源端集群文档数        目标端集群文档数
tt                                         0                   1
test0911                                   0                   1
(base) [roo                 ~]# █
```

- 数据一致返回

```
(base) [root@VM-10-2-tencentos ~]# java -jar ElasticsearchCompare.jar http://          :9201 elastic any_password
 https://          .public.tencentelasticsearch.com:9200 elastic              "*,-.*"

以下是源端集群Close状态的索引列表，无法获取文档数进行数据一致性对比：
[]

以下是源目两端集群文档数量不一致的索引列表：
索引名称                                    源端集群文档数        目标端集群文档数
(base) [roo                 ~]# █
```

业务切割

数据迁移完成并完成一致性校验，以及完成业务兼容性完全测试，则可以进行业务切割上云，将原 ES 访问地址切换为云上内网访问地址即可：

应用场景构建
使用 Elastic Stack 构建日志分析系统

最近更新时间：2025-06-05 15:48:21

腾讯云 Elasticsearch Service 提供的实例包含 ES 集群和 Kibana 控制台，其中 ES 集群通过在用户 VPC 内的私有网络 VIP 地址 + 端口进行访问，Kibana 控制台提供外网地址供用户在浏览器端访问，至于数据源，当前只支持用户自行接入 ES 集群。
下面以最典型的日志分析架构 Filebeat + Elasticsearch + Kibana 和 Logstash + Elasticsearch + Kibana 为例，介绍如何将用户的日志导入到 ES，并可以在浏览器访问 Kibana 控制台进行查询与分析。

Filebeat + Elasticsearch + Kibana

部署 Filebeat

1. 下载 Filebeat 组件包并解压

> **注意**
> Filebeat 版本应该与 ES 版本保持一致。

```
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-6.4.3-linux-x86_64.tar.gz
tar xvf filebeat-6.4.3-linux-x86_64.tar.gz
```

2. 配置 Filebeat

本示例以 nginx 日志为输入源，输出项配置为 ES 集群的内网 VIP 地址和端口，如果使用的是白金版的集群，output 中需要增加用户名密码验证。
进入 filebeat-6.4.3-linux-x86_64 目录，修改 filebeat.yml 配置文件，文件内容如下：

```
filebeat.inputs:
- type: log
    enabled: true
    paths:
        - /var/log/nginx/access.log
output.elasticsearch:
    hosts: ["10.0.130.91:9200"]  #ES 集群的内网 VIP 地址和端口
    protocol: "http" # ES通信模式，http或者https
    username: "elastic" # 用户名
    password: "test" # 密码
```

3. 执行 Filebeat

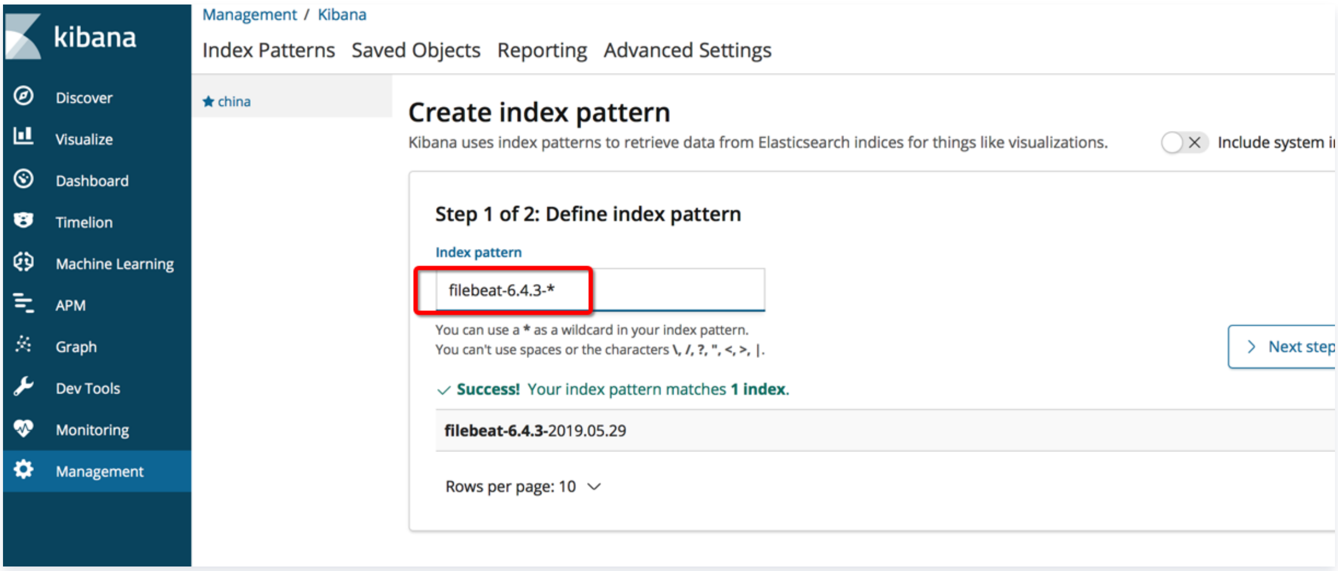
在 filebeat-6.4.3-linux-x86_64 目录中，执行：

```
nohup ./filebeat -c filebeat.yml 2>&1 >/dev/null &
```

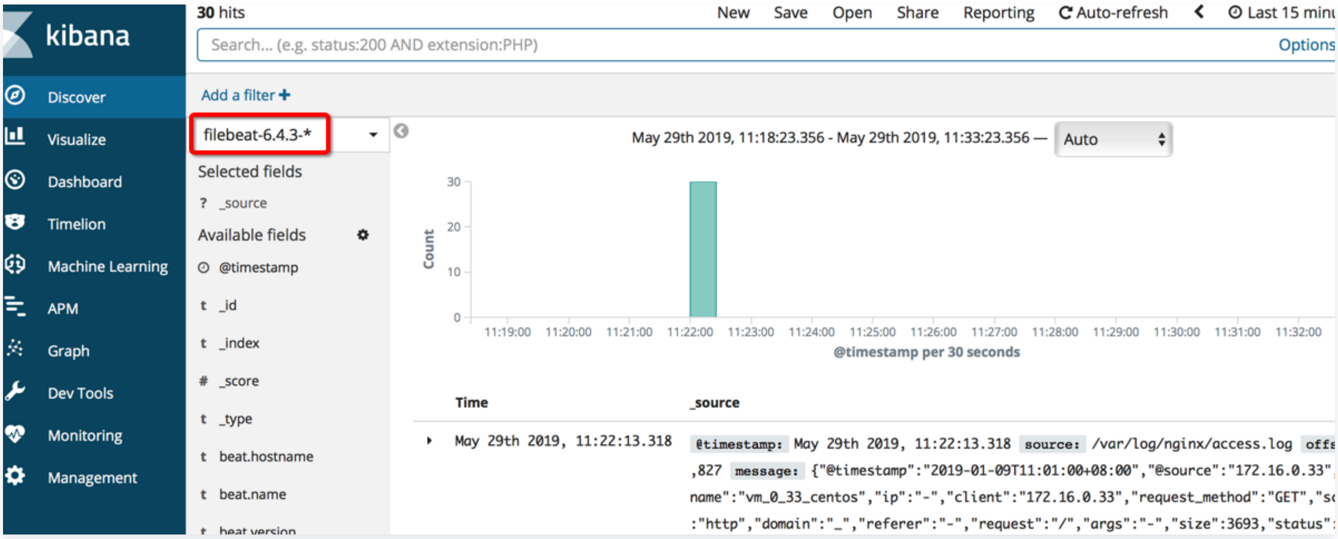
查询日志

1. 在 ES 控制台集群列表页中，选择**操作 > Kibana**，进入 Kibana 控制台。

2. 进入 **Management > Index Patterns**，添加名为 `filebeat-6.4.3-*` 的索引 pattern。



3. 单击 **Discover**，选择 `filebeat-6.4.3-*` 索引项，即可检索到 nginx 的访问日志。



Logstash + Elasticsearch + Kibana

环境准备

- 用户需要创建和 ES 集群在同一 VPC 的 CVM，根据需要可以创建多台 CVM 实例，在 CVM 实例中部署 logstash 组件。
- CVM 需要有2G以上内存。
- 在创建好的 CVM 中安装 Java8 或以上版本。

部署 Logstash

1. 下载 Logstash 组件包并解压

> **注意**
> logstash 版本应该与 ES 版本保持一致。

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-6.4.3.tar.gz
tar xvf logstash-6.4.3.tar.gz
```

2. 配置 Logstash
本示例以 nginx 日志为输入源，输出项配置为 ES 集群的内网 VIP 地址和端口，创建 test.conf 配置文件，文件内容如下：

```
input {
 file {
     path => "/var/log/nginx/access.log" # nginx 访问日志的路径
     start_position => "beginning" # 从文件起始位置读取日志，如果不设置则在文件有写入时才读取，类似于 tail -f
 }
}
filter {
}
output {
 elasticsearch {
     hosts => ["http://172.16.0.145:9200"] # ES 集群的内网 VIP 地址和端口
     index => "nginx_access-%{+YYYY.MM.dd}" # 索引名称， 按天自动创建索引
     user => "elastic" # 用户名
     password => "yinan_test" # 密码
  }
}
```

ES 集群默认开启了允许自动创建索引配置，上述 test.conf 配置文件中的 `nginx_access-%{+YYYY.MM.dd}` 索引会自动创建，除非需要提前设置好索引中字段的 mapping，否则无需额外调用 ES 的 API 创建索引。

3. 启动 logstash
   进入 logstash 压缩包解压目录 logstash-6.4.3 下，执行以下命令，后台运行 logstash，**配置文件路径填写为自己创建的路径。**

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

查看 logstash-6.4.3 目录下的 logs 目录，确认 Logstash 已经正常启动，正常启动的情况下会记录如下日志：

```
Sending Logstash logs to /root/logstash-6.4.3/logs which is now configured via log4j2.properties
[2019-05-29T12:20:26,630][INFO ][logstash.setting.writabledirectory] Creating directory
{:setting=>"path.queue", :path=>"/root/logstash-6.4.3/data/queue"}
[2019-05-29T12:20:26,639][INFO ][logstash.setting.writabledirectory] Creating directory
{:setting=>"path.dead_letter_queue", :path=>"/root/logstash-6.4.3/data/dead_letter_queue"}
[2019-05-29T12:20:27,125][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file
because modules or command line options are specified
[2019-05-29T12:20:27,167][INFO ][logstash.agent           ] No persistent UUID file found. Generating new
UUID {:uuid=>"2e19b294-2b69-4da1-b87f-f4cb4a171b9c", :path=>"/root/logstash-6.4.3/data/uuid"}
[2019-05-29T12:20:27,843][INFO ][logstash.runner          ] Starting Logstash {"logstash.version"=>"6.4.3"}
[2019-05-29T12:20:30,067][INFO ][logstash.pipeline        ] Starting pipeline {:pipeline_id=>"main",
"pipeline.workers"=>1, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>50}
[2019-05-29T12:20:30,871][INFO ][logstash.outputs.elasticsearch] Elasticsearch pool URLs updated
{:changes=>{:removed=>[], :added=>[http://elastic:xxxxxx@10.0.130.91:10880/]}}
[2019-05-29T12:20:30,901][INFO ][logstash.outputs.elasticsearch] Running health check to see if an
Elasticsearch connection is working {:healthcheck_url=>http://elastic:xxxxxx@10.0.130.91:10880/,
:path=>"/"}
[2019-05-29T12:20:31,449][WARN ][logstash.outputs.elasticsearch] Restored connection to ES instance
{:url=>"http://elastic:xxxxxx@10.0.130.91:10880/"}
[2019-05-29T12:20:31,567][INFO ][logstash.outputs.elasticsearch] ES Output version determined
{:es_version=>6}
[2019-05-29T12:20:31,574][WARN ][logstash.outputs.elasticsearch] Detected a 6.x and above cluster: the
`type` event field won't be used to determine the document _type {:es_version=>6}
[2019-05-29T12:20:31,670][INFO ][logstash.outputs.elasticsearch] New Elasticsearch output
{:class=>"LogStash::Outputs::ElasticSearch", :hosts=>["http://10.0.130.91:10880"]}
[2019-05-29T12:20:31,749][INFO ][logstash.outputs.elasticsearch] Using mapping template from {:path=>nil}
[2019-05-29T12:20:31,840][INFO ][logstash.outputs.elasticsearch] Attempting to install template
{:manage_template=>{"template"=>"logstash-*", "version"=>60001, "settings"=>
{"index.refresh_interval"=>"5s"}, "mappings"=>{"_default_"=>{"dynamic_templates"=>[{"message_field"=>
{"path_match"=>"message", "match_mapping_type"=>"string", "mapping"=>{"type"=>"text", "norms"=>false}}},
{"string_fields"=>{"match"=>"*", "match_mapping_type"=>"string", "mapping"=>{"type"=>"text",
"norms"=>false, "fields"=>{"keyword"=>{"type"=>"keyword", "ignore_above"=>256}}}}}], "properties"=>
{"@timestamp"=>{"type"=>"date"}, "@version"=>{"type"=>"keyword"}, "geoip"=>{"dynamic"=>true,
```



```
"properties"=>{"ip"=>{"type"=>"ip"}, "location"=>{"type"=>"geo_point"}, "latitude"=>
{"type"=>"half_float"}, "longitude"=>{"type"=>"half_float"}}}}}}}}
[2019-05-29T12:20:32,094][INFO ][logstash.outputs.elasticsearch] Installing elasticsearch template to
_template/logstash
[2019-05-29T12:20:33,242][INFO ][logstash.inputs.file     ] No sincedb_path set, generating one based on
the "path" setting {:sincedb_path=>"/root/logstash-
6.4.3/data/plugins/inputs/file/.sincedb_d883***********************b2a2", :path=>
["/var/log/nginx/access.log"]}
[2019-05-29T12:20:33,329][INFO ][logstash.pipeline        ] Pipeline started successfully
{:pipeline_id=>"main", :thread=>"#<Thread:0x12bdd65 run>"}
[2019-05-29T12:20:33,544][INFO ][logstash.agent           ] Pipelines running {:count=>1,
:running_pipelines=>[:main], :non_running_pipelines=>[]}
[2019-05-29T12:20:33,581][INFO ][filewatch.observingtail  ] START, creating Discoverer, Watch with file and
sincedb collections
[2019-05-29T12:20:34,368][INFO ][logstash.agent           ] Successfully started Logstash API endpoint
{:port=>9600}
```

有关 Logstash 的更多功能，请参见 elastic 官方文档。

查询日志

更多详情请参见 查询集群日志。
更多有关 Kibana 控制台的功能，请参见 elastic 官方文档。

腾讯云 ES + SCF 快速构建搜索服务

最近更新时间：2025-04-30 17:38:42

搜索服务

搜索服务广泛的存在于我们身边，例如我们生活中用的百度、工作中用的 Wiki 搜索、淘宝时用的商品搜索等。这些场景的数据具有数据量大、结构化、读多写少等特点，而传统的数据库的事务特性在搜索场景并没有很好的使用空间，并且在全文检索方面速度慢（例如 like 语句）。因此，Elasticsearch 应运而生。

Elasticsearch 是一个广泛应用于全文搜索领域的开源搜索引擎，它可以快速地索引、搜索和分析海量的文本数据。腾讯云 ES 是基于 Elasticsearch 构建的高可用、可伸缩的云端托管 Elasticsearch 服务，对结构化和非结构化的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，方便快速搭建稳定的搜索服务。

本文将针对搜索场景，使用腾讯云 ES 官方文档作为语料，介绍如何使用腾讯云 ES + SCF 快速搭建搜索服务。搜索服务界面示例如下：

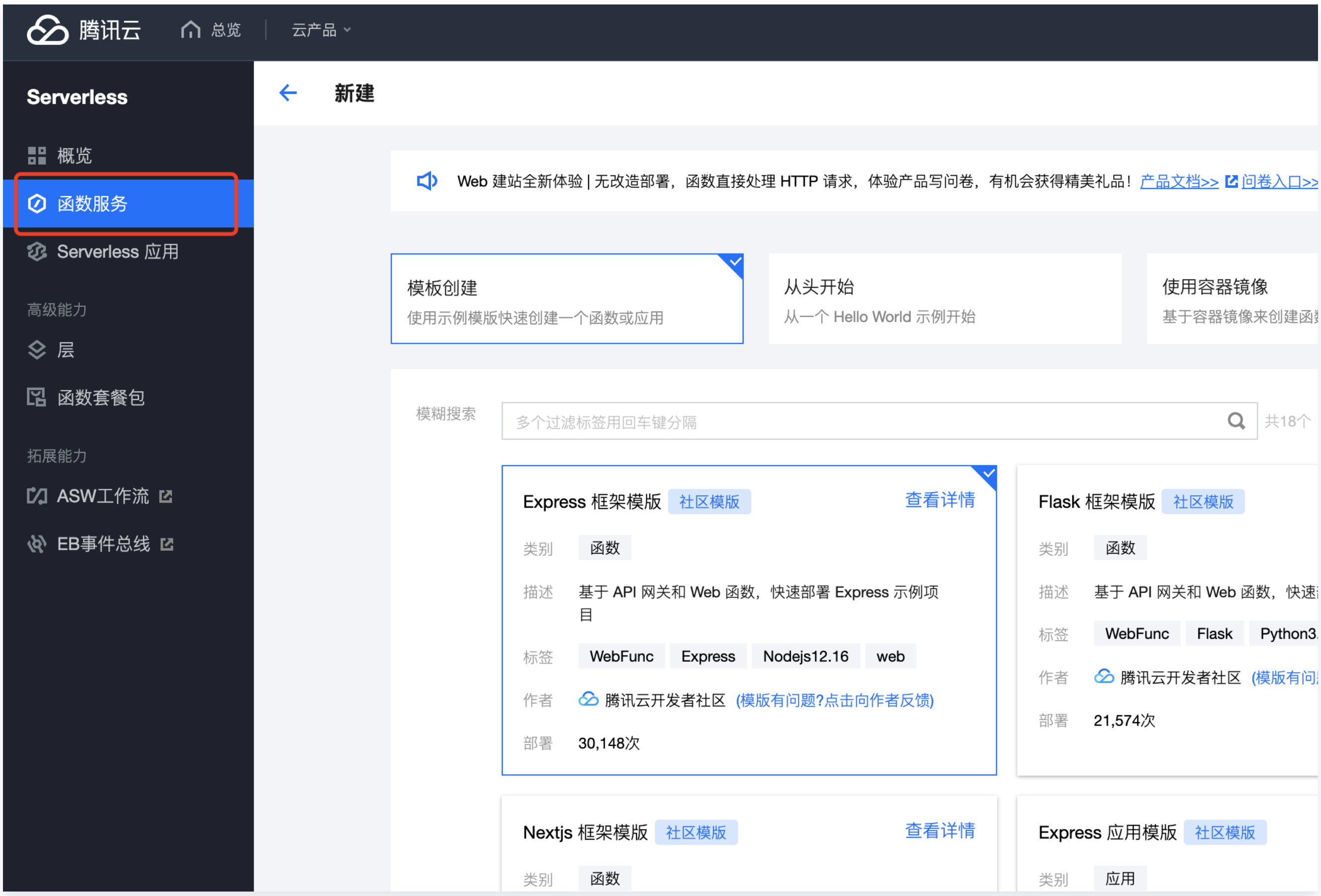
资源准备

只需要一个 ES 集群。在腾讯云购买一个 ES 集群，集群的规模根据搜索服务的 QPS 和存入的文档的数据量而定。具体可参见 集群规格和容量配置评估 。

部署搜索服务

使用腾讯云**免费**的 SCF 工具部署搜索服务的前端界面和后台服务。

1. 在**云函数** > **函数服务** 界面左上角首先选择您购买 ES 集群的地域。

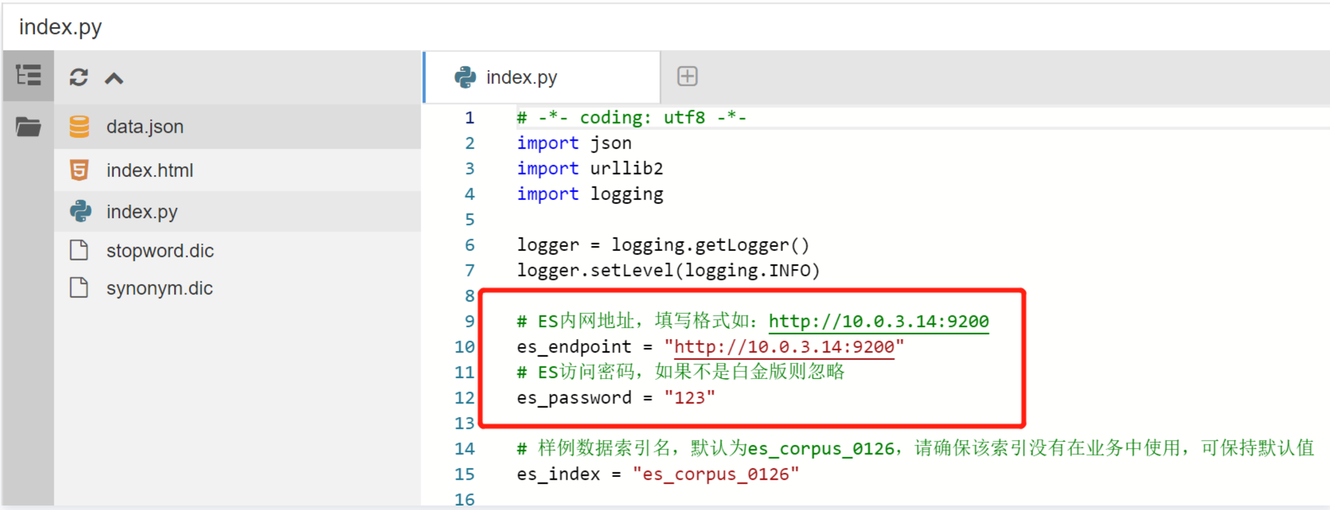


2. 新建一个函数服务，**记住您设置的函数名称其他则可以按照截图内容来选择。**

3. 在创建流程下方的高级配置中，将**私有网络**勾选为启用，VPC 和 ES 集群的 VPC 保持一致。

4. 在函数代码界面，首先将 代码 zip 包 下载到本地。然后在**函数代码 > 提交方法**中，选择**上传本地 zip 包**，并选择刚下载的 zip 包，单击**保存**。

5. 在**函数代码**页修改代码。需要修改的文件有 `index.py` 和 `index.html`：
   - `index.py` 中的 `es_endpoint` 修改为您的 ES 集群的内网地址，填写格式如：`http://10.0.3.14:9200`。
   - `index.py` 中的 `es_password` 修改为白金版 ES 密码，如果不是白金版则不修改。

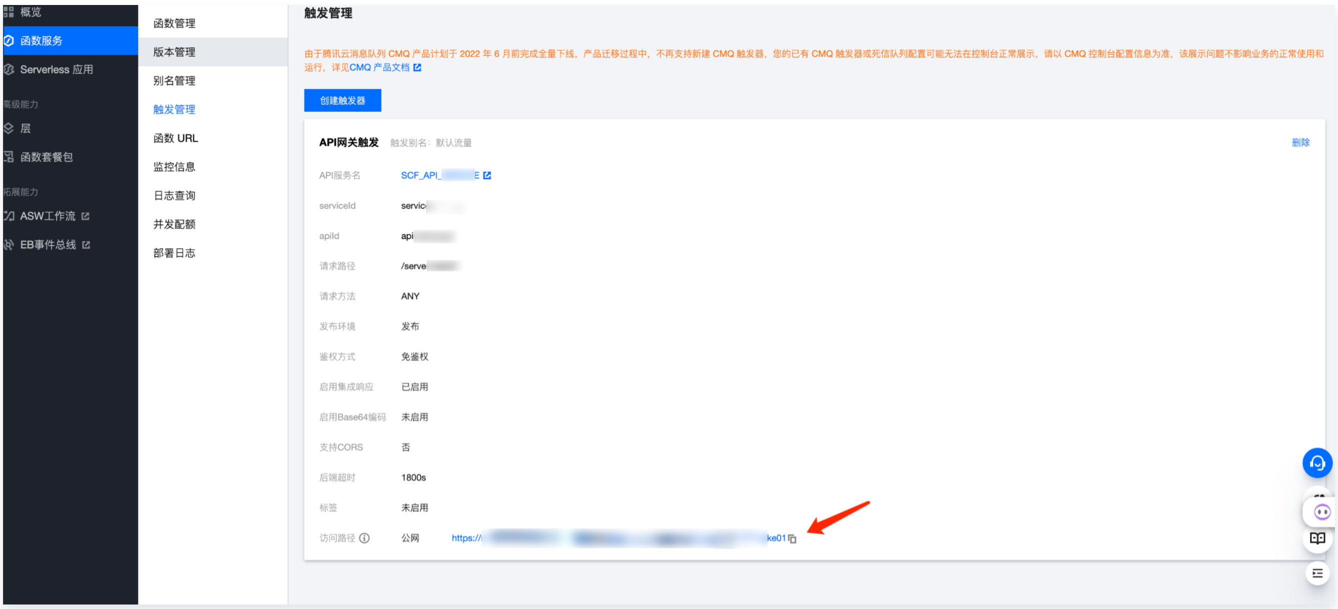


   - `index.html` 中的 `server_name` 修改为您创建的 SCF 函数的函数名称，默认为 `myserver`。

> ⚠ **注意**
> 样例默认使用 `es_corpus_0126` 作为索引名，请确保该索引没有业务在使用。如需修改，可在 `index.py` 中修改 `es_index` 变量。

6. 在触发管理页单击创建**触发方式**，按下图添加 API 网关触发器，并启用集成响应，然后单击**保存**。

7. 可在触发管理中看到函数的**访问路径**，单击此路径即可访问页面。



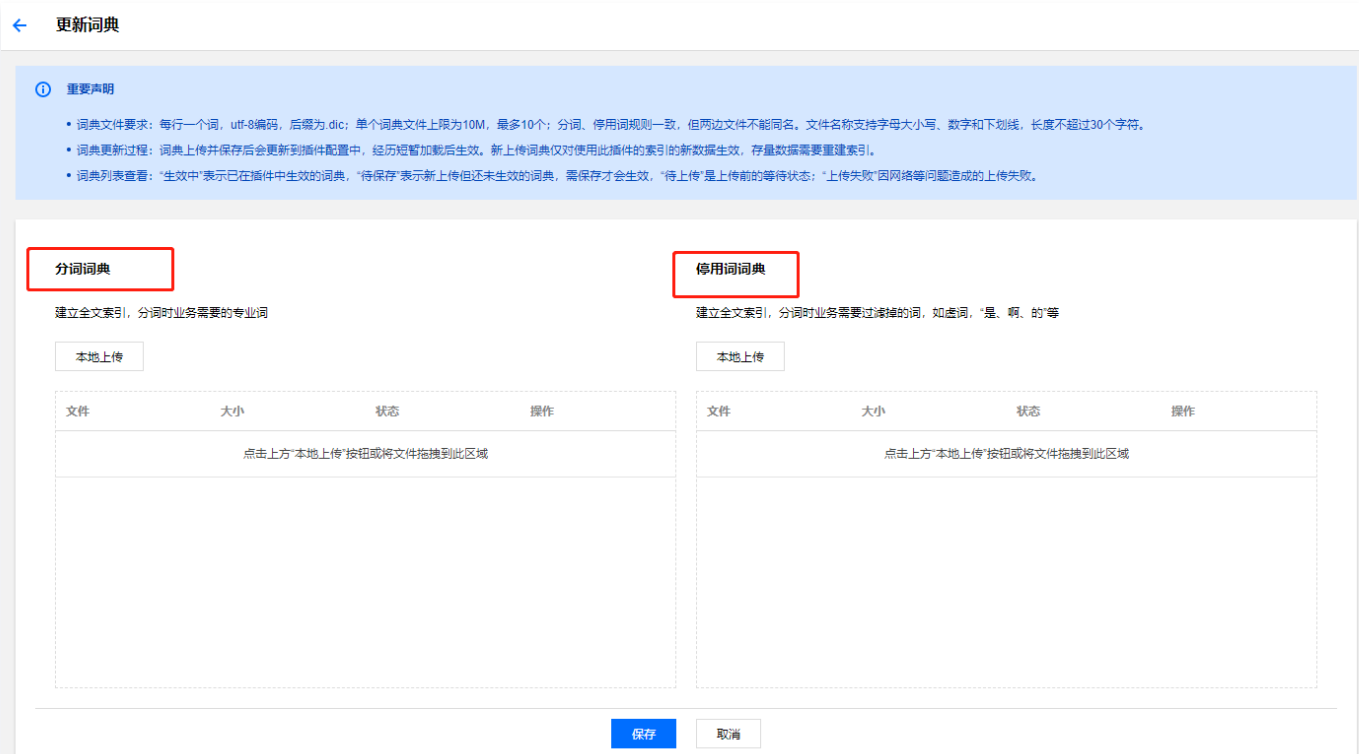
8. 上传 腾讯云 ES 官方文档 样例数据。单击搜索框上方的文字，自动导入数据。

9. 至此，一个简单的基于腾讯云 ES 的问答搜索服务后台已部署完成。

了解更多

停用词和用户词典导入

停用词不会被 ES 检索，用户词典在分词时将保留该词。在上面的案例中，我们导入了默认的 停用词库 和 用户词典，您也可以在 ES 集群详情页的**插件列表** > **更新词典**导入自己的停用词和用户词典。



同义词配置

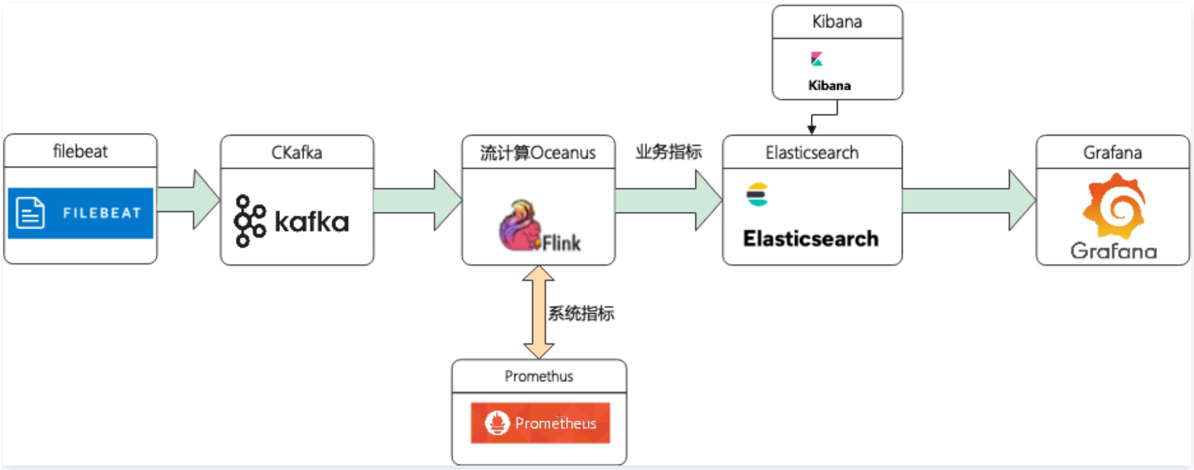
同义词配置需要在创建索引时指定，支持 Solr 和 WordNet 两种同义词格式，具体可参考 Solr synonyms 对格式的介绍。

基于流计算 Oceanus 的实时监控解决方案

最近更新时间：2024-11-20 17:54:32

本方案结合腾讯云 Ckafka、流计算 Oceanus、腾讯云数据库 Elasticsearch、腾讯云可观测平台等，通过 Filebeat 实时监控系统日志和应用日志，将监控数据传输到腾讯云 Ckafka，再将 Kafka 中数据接入流计算 Oceanus，经过简单的业务逻辑处理输出到云数据库 Elasticsearch，利用云 Promethus 监控系统指标，利用云 Grafana 实现对 Oceanus 作业的个性化业务数据监控。

方案架构



前置准备

在使用前，请确保已购买并创建相应的大数据组件。

创建私有网络 VPC

私有网络是一块您在腾讯云上自定义的逻辑隔离网络空间。需要使用对等连接、NAT 网关等方式打通网络。具体创建步骤可参考 创建私有网络 。

> **注意**
> 在构建 Ckafka、Oceanus、Elasticsearch 集群等服务时选择的网络必须保持一致，网络才能互通。

创建 Ckafka 实例

进入 消息队列 CKafka 控制台，在实例列表中单击**新建**，创建 Ckafka 实例详情请参见 创建实例 。购买完成后，再创建 Kafka topic（ `topic-app-info` ），详细操作请参见 创建 Topic 。

> **说明**
> - 私有网络和子网选择上一步创建的网络和子网。
> - Kafka 建议选择最新的版本，和 Filebeat 采集工具兼容性较好。

创建 Oceanus 集群

流计算 Oceanus 服务兼容原生的 Flink 任务。在 流计算 Oceanus 控制台 中**计算资源 > + 新建**创建集群，选择地域、可用区、VPC、日志、存储、设置初始密码等。**VPC 及子网选择刚创建好的网络**，具体创建步骤可参考 创建独享集群。创建完后 Flink 的集群如下：

创建 Elasticsearch 实例

进入 Elasticsearch Service 控制台，在ES集群管理中单击新建**集群，需选择之前创建好的私有网络和子网**，并设置账户和密码，具体操作可参考 创建集群。

创建腾讯云可观测平台服务

为了展示自定义系统指标，需购买 Prometheus 服务。若只需要自定义业务指标，可以省略此步骤。
进入腾讯云可观测平台页面，单击左侧 Prometheus 监控，单击**新建**，**选择之前的私有网络和子网**，并设置实例名称和 Grafana 密码，具体操作可参考 创建实例 。

创建独立 Grafana 资源

在 Grafana 管理页面 进行单独购买实现业务监控指标的展示。单击**新建**进入 Grafana 可视化服务购买页，详情请参见 创建实例 ，购买时仍需选择与其他资源相同的 VPC 网络。

安装配置 Filebeat

Filebeat 是一款轻量级日志数据采集的工具，通过监控指定位置的文件收集信息。在该 VPC 下给需要监控主机信息和应用信息的 CVM 上安装 Filebeat(配置一般会在filebeat.yml中)。

- 方式一：下载 Filebeat 并安装 Filebeat 下载地址 。
- 方式二：采用 **Elasticsearch 管理页面 > beats 管理**中提供的 Filebeat。本示例中采用了方式一。
  下载到 CVM 中并配置 Filebeat，在 filebeat.yml 文件中添加如下配置项：```shell。

监控日志文件配置

- type: log
  enabled: true
  paths:
  - /tmp/test.log
    #- c:\programdata\elasticsearch\logs*

```
```shell
# 监控数据输出项配置
output.kafka:
version: 2.0.0                          # kafka版本号
hosts: ["xx.xx.xx.xx:xxxx"]             # 请填写实际的IP地址+端口
topic: 'topic-app-info'                   # 请填写实际的topic
```

请根据实际业务需求配置相对应的 filebeat.yml 文件，参考 Filebeat 官方文档。

> **注意**
> 示例选用2.4.1的 Ckafka 版本，这里配置 version: 2.0.0。版本对应不上可能出现 `ERROR   [kafka] kafka/client.go:341     Kafka (topic=topic-app-info): dropping invalid message` 错误。

方案实现

接下来通过案例介绍如何通过流计算 Oceanus 实现个性化监控。

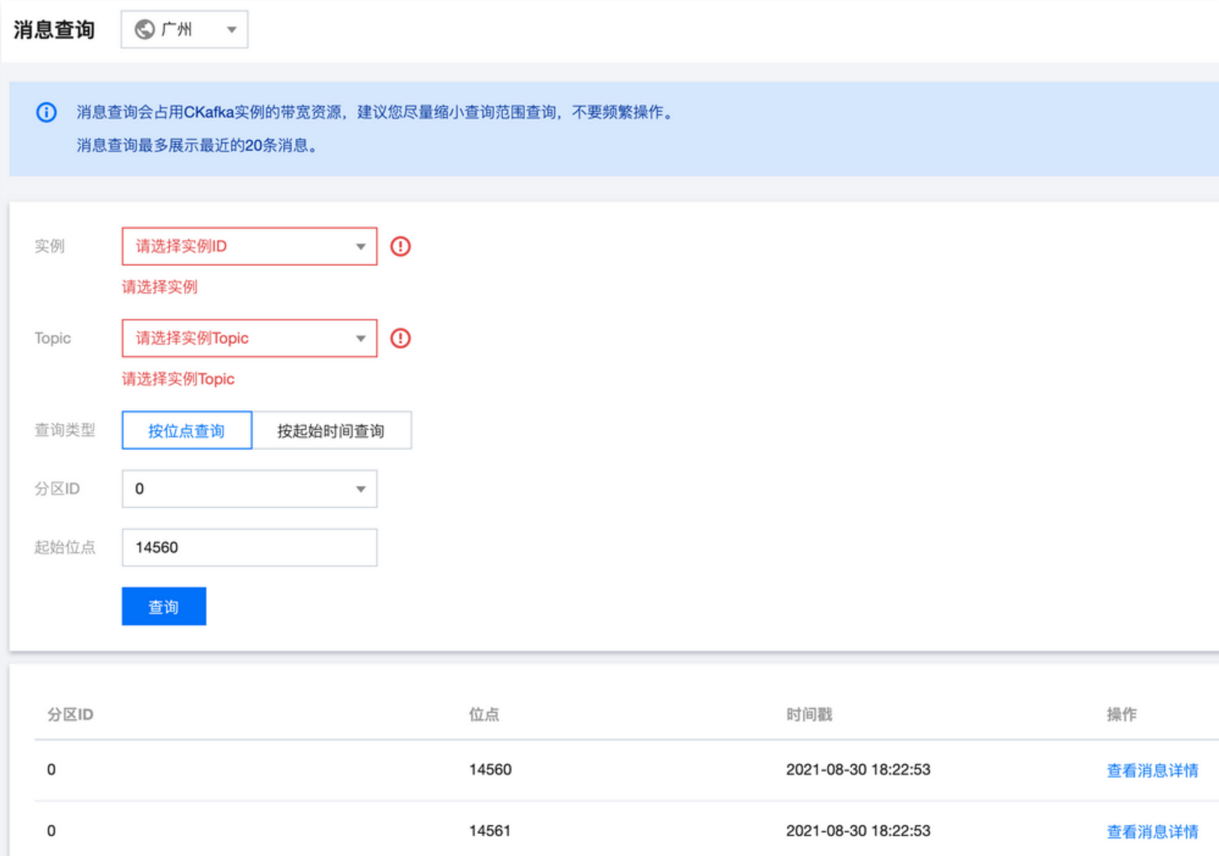
Filebeat 数据传输

1. 进入到 Filebeat 根目录下，并启动 Filebeat 进行数据采集。示例中采集了 top 命令中显示的 CPU、内存等信息，也可以采集 jar 应用的日志、JVM 使用情况、监听端口等，详情可参考 Filebeat 官网。

```
# filebeat启动
./filebeat -e -c filebeat.yml

# 监控系统信息写入test.log文件
top -d 10 >>/tmp/test.log
```
```

2. 进入 Ckafka 页面，单击**消息查询**，查询对应 topic 消息，验证是否采集到数据。



filebeat 采集到的数据格式：

```
{
  "@timestamp": "2021-08-30T10:22:52.888Z",
  "@metadata": {
 "beat": "filebeat",
 "type": "_doc",
 "version": "7.14.0"
  },
  "input": {
 "type": "log"
  },
  "host": {
 "ip": ["xx.xx.xx.xx", "xx::xx:xx:xx:xx"],
 "mac": ["xx:xx:xx:xx:xx:xx"],
 "hostname": "xx.xx.xx.xx",
 "architecture": "x86_64",
 "os": {
   "type": "linux",
   "platform": "centos",
   "version": "7(Core)",
   "family": "redhat",
   "name": "CentOSLinux",
   "kernel": "3.10.0-1062.9.1.el7.x86_64",
   "codename": "Core"
 },
 "id": "0ea7************************9dfc",
 "name": "xx.xx.xx.xx",
 "containerized": false
  },
  "agent": {
 "name": "xx.xx.xx.xx",
```

```
  "type": "filebeat",
  "version": "7.14.0",
  "hostname": "xx.xx.xx.xx",
  "ephemeral_id": "6c0922a6-17af-4474-9e88-1fc3b1c3b1a9",
  "id": "6b23463c-0654-4f8b-83a9-84ec75721311"
   },
   "ecs": {
  "version": "1.10.0"
   },
   "log": {
  "offset": 2449931,
  "file": {
    "path": "/tmp/test.log"
  }
   },
   "message": "�(B�[m16root0-20000S0.00.00:00.00kworker/1:0H�(B�[m�[39;49m�[K"
 }
```

SQL 作业编写

在流计算 Oceanus 中，对 Kafka 接入的数据进行加工处理，并存入 Elasticsearch 中。

1. 定义 source

按照 Filebeat 中 json 消息的格式，构造 Flink Source。

```
 CREATE TABLE DataInput (
     `@timestamp` VARCHAR,
     `host`        ROW<id VARCHAR,ip ARRAY<VARCHAR>>,
     `log`         ROW<`offset` INTEGER,file ROW<path VARCHAR>>,
     `message`     VARCHAR
 ) WITH (
     'connector' = 'kafka',    -- 可选 'kafka','kafka-0.11'. 注意选择对应的内置  Connector
     'topic' = 'topic-app-info',  -- 替换为您要消费的 Topic
     'scan.startup.mode' = 'earliest-offset', -- 可以是 latest-offset / earliest-offset / specific-offsets /
group-offsets 的任何一种
     'properties.bootstrap.servers' = '10.0.0.29:9092',  -- 替换为您的 Kafka 连接地址
     'properties.group.id' = 'oceanus_group2',  -- 必选参数, 一定要指定 Group ID
     -- 定义数据格式 (JSON 格式)
     'format' = 'json',
     'json.ignore-parse-errors' = 'true',      -- 忽略 JSON 结构解析异常
     'json.fail-on-missing-field' = 'false'   -- 如果设置为 true, 则遇到缺失字段会报错 设置为 false 则缺失字段设置为
null
 );
```

2. 定义 sink

```
CREATE TABLE es_output (
    `id` VARCHAR,
    `ip` ARRAY<VARCHAR>,
    `path` VARCHAR,
    `num` INTEGER,
    `message` VARCHAR,
    `createTime` VARCHAR
) WITH (
    'connector.type' = 'elasticsearch', -- 输出到 Elasticsearch
    'connector.version' = '6',           -- 指定 Elasticsearch 的版本, 例如 '6', '7'.
    'connector.hosts' = 'http://10.0.0.175:9200',  -- Elasticsearch 的连接地址
    'connector.index' = 'oceanus_test2',        -- Elasticsearch 的 Index 名
    'connector.document-type' = '_doc',  -- Elasticsearch 的 Document 类型
    'connector.username' = 'elastic',
    'connector.password' = 'yourpassword',
```

```
    'update-mode' = 'upsert',              -- 可选无主键的 'append' 模式，或有主键的 'upsert' 模式
    'connector.key-delimiter' = '$',       -- 可选参数，复合主键的连接字符 (默认是 _ 符号，例如 key1_key2_key3)
    'connector.key-null-literal' = 'n/a',   -- 主键为 null 时的替代字符串，默认是 'null'
    'connector.failure-handler' = 'retry-rejected',    -- 可选的错误处理。可选择 'fail' （抛出异常）、'ignore'（忽略任何错误）、'retry-rejected'（重试）

    'connector.flush-on-checkpoint' = 'true',    -- 可选参数，快照时不允许批量写入（flush），默认为 true
    'connector.bulk-flush.max-actions' = '42',   -- 可选参数，每批次最多的条数
    'connector.bulk-flush.max-size' = '42 mb',   -- 可选参数，每批次的累计最大大小 （只支持 mb）
    'connector.bulk-flush.interval' = '60000',   -- 可选参数，批量写入的间隔 （ms）
    'connector.connection-max-retry-timeout' = '1000',      -- 每次请求的最大超时时间 （ms）
    --'connector.connection-path-prefix' = '/v1'           -- 可选字段，每次请求时附加的路径前缀
    'format.type' = 'json'         -- 输出数据格式，目前只支持 'json'
);
```

3. 业务逻辑

```
INSERT INTO es_output
SELECT
host.id as `id`,
host.ip as `ip`,
log.file.path as `path`,
log.`offset` as `num`,
message,
`@timestamp` as `createTime`
from DataInput;
```

4. ES 数据查询

在 ES 控制台的 Kibana 页面查询数据，或者进入某台相同子网的 CVM 下，使用以下命令进行查询：

```
# 查询索引  username:password请替换为实际账号密码
curl -XGET -u username:password http://xx.xx.xx.xx:xxxx/oceanus_test2/_search -H 'Content-Type:
application/json' -d'
{
    "query": { "match_all": {}},
    "size":  10
}
'
```

更多访问方式请参考 访问 ES 集群。

系统指标监控

本章节主要实现系统信息监控，对 Flink 作业运行状况进行监控告警。
Prometheus 是一个非常灵活的时序数据库，通常用于监控数据的存储、计算和告警。流计算 Oceanus 建议用户使用腾讯云可观测平台提供的 Prometheus 服务，以免去部署、运维开销；同时它还支持腾讯云的通知模板，可以通过短信、电话、邮件、企业微信机器人等方式，将告警信息轻松触达不同的接收方。

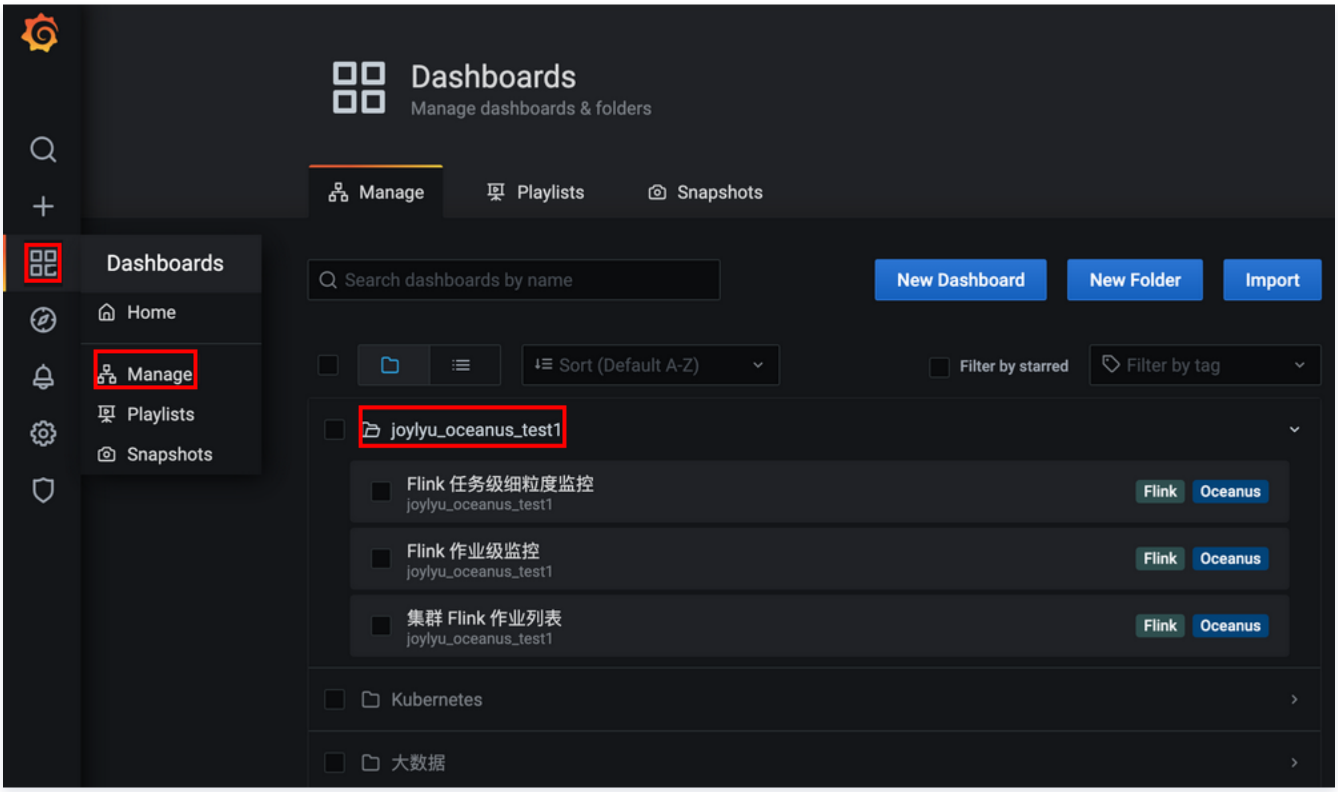
监控配置（Oceanus 作业监控）

除了 Oceanus 控制台 自带的监控信息，还可以配置目前已经支持了任务级细粒度监控、作业级监控和集群 Flink 作业列表监控。

1. Oceanus 作业详情页面，单击**作业参数**，在**高级参数**处添加如下配置：

```
pipeline.max-parallelism: 2048
metrics.reporters: promgateway
metrics.reporter.promgateway.host: xx.xx.xx.xx              # Prometheus实例地址
metrics.reporter.promgateway.port: 9090                     # Prometheus实例端口
metrics.reporter.promgateway.needBasicAuth: true
metrics.reporter.promgateway.password: xxxxxxxxxxx          # Prometheus实例密码
metrics.reporter.promgateway.interval: 10 SECONDS
```

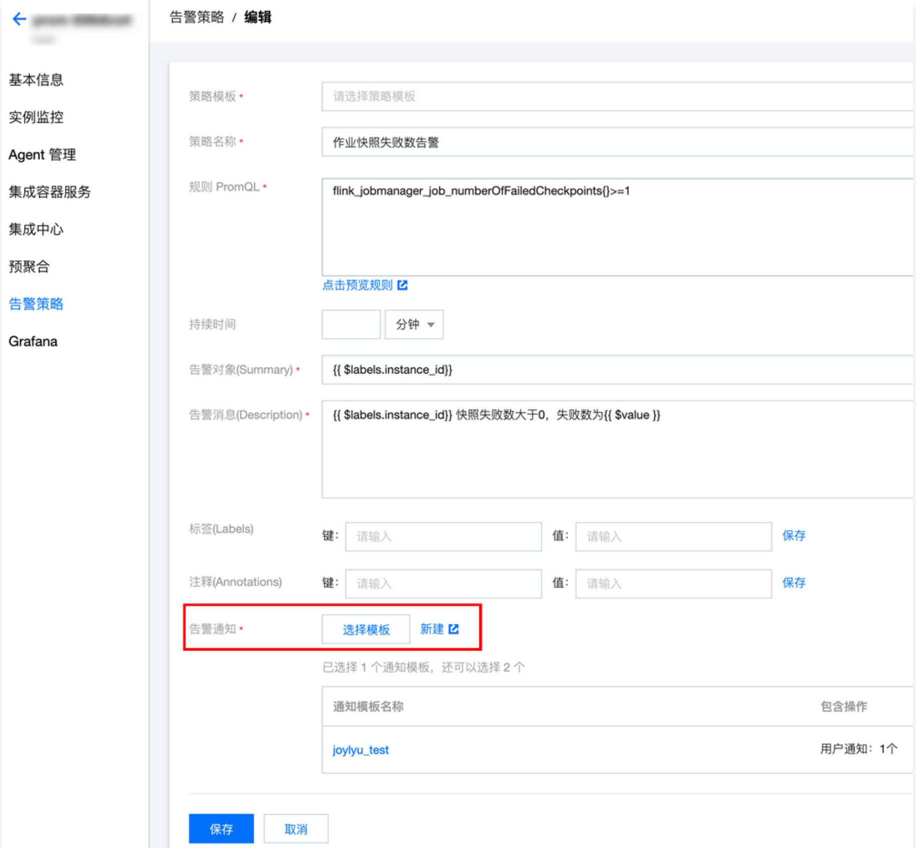
2. 在任一 Oceanus 作业中，单击**云监控**进入云 Prometheus 实例，点击链接进入 Grafana（灰度中的 Grafana 不能由此进入），导入 json 文件，详情请参见 [接入 Prometheus 自定义监控]。



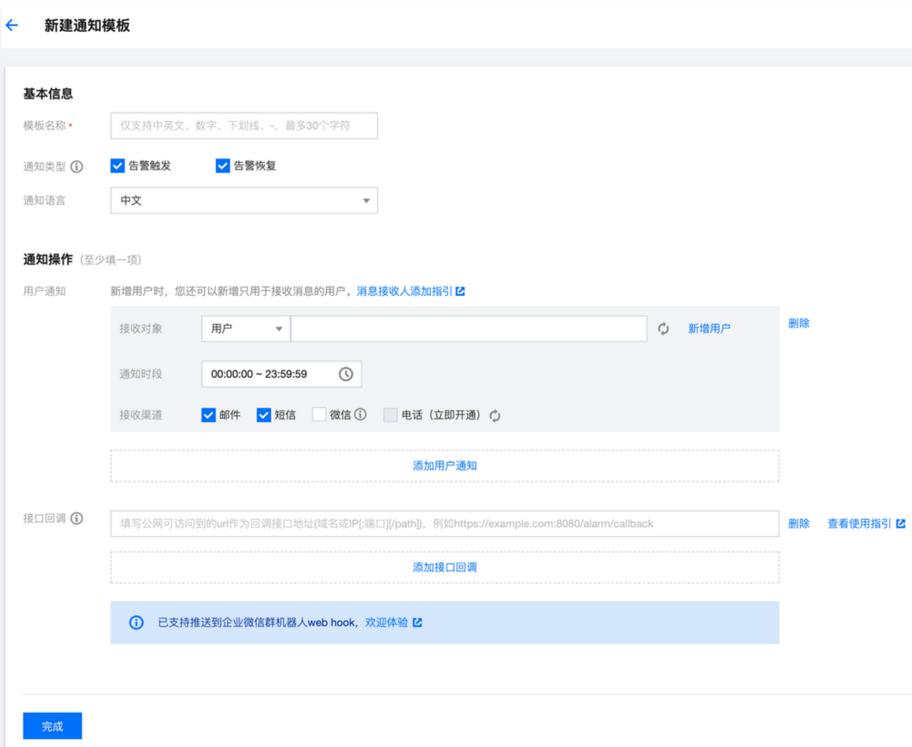
3. 展现出来的 flink 任务监控效果如下，用户也可以单击 **Edit** 设置不同 Panel 来优化展现效果。

告警配置

1. 进入腾讯云可观测平台界面，单击 **Prometheus 监控**，点击已购买的实例进入服务管理页面，然后选择**告警策略 > 新建**，配置相关信息。具体操作参考 [接入 Prometheus 自定义监控]。



2. 设置告警通知。选择**选择模板**或**新建**，设置通知模板。

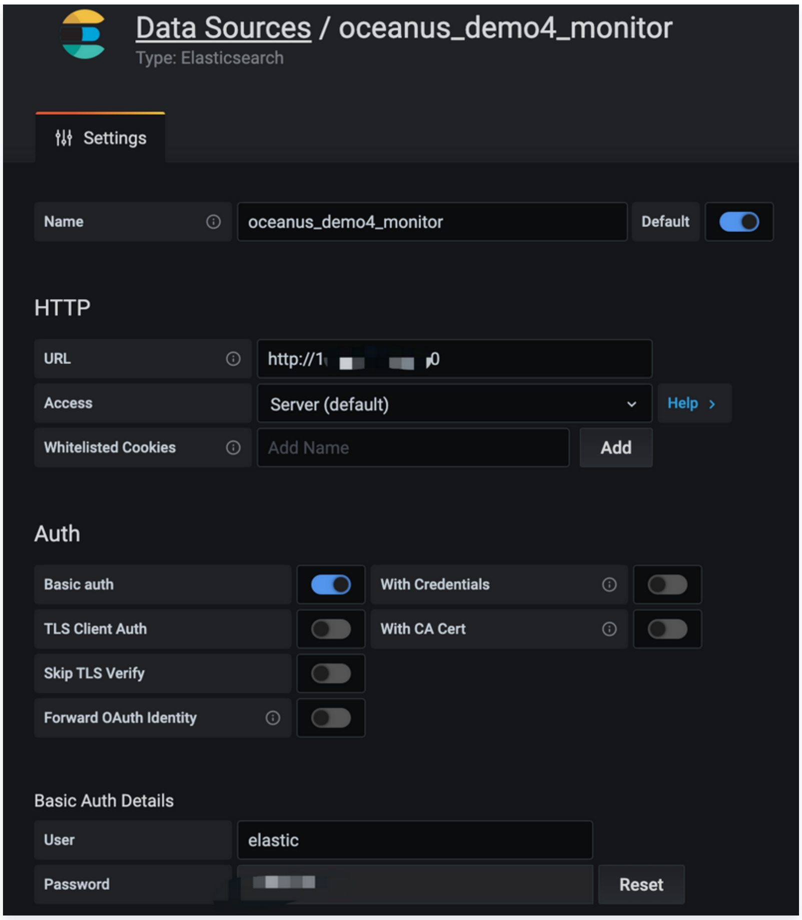


3. 短信通知消息。

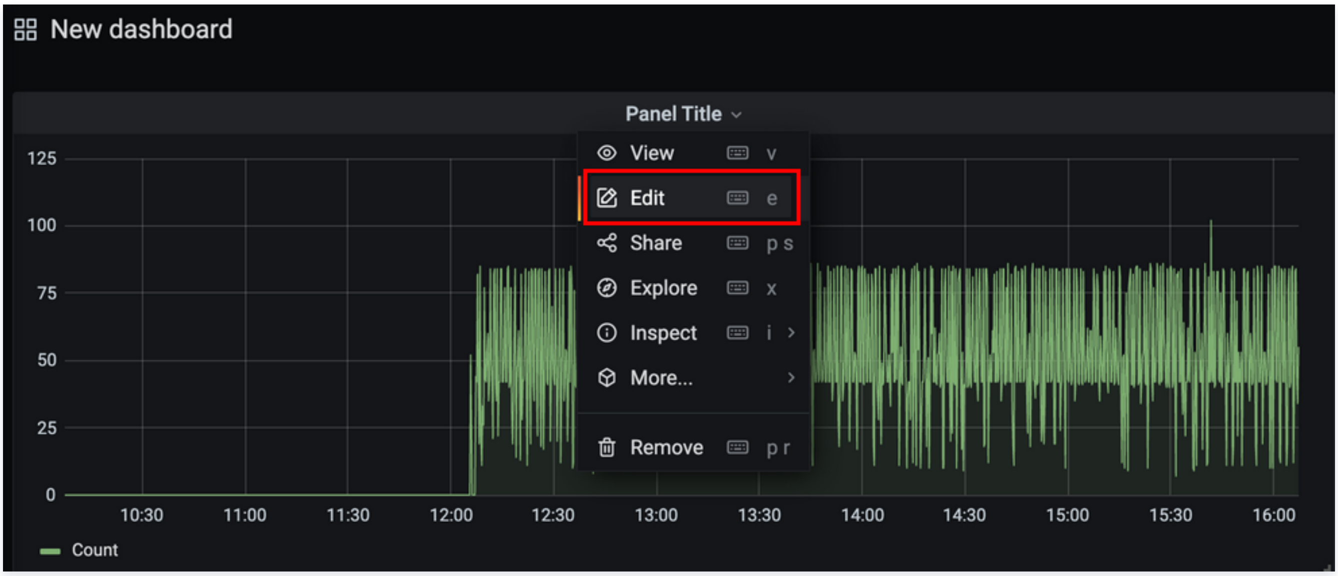
业务指标监控

通过 Filebeat 采集到应用业务数据，经过 Oceanus 服务的加工处理已经被存入 ES，可以通过 ES + Grafana 来实现业务数据的监控。

1. Grafana 配置 ES 数据源。进入灰度发布中的 Grafana 控制台，进入刚创建的 Grafana 服务，找到外网地址并打开。Grafana 账号为 admin，登录后选择 **Configuration > Add Source**，搜索 `elasticsearch`，填写相关 ES 实例信息，添加数据源。



2. 选择左侧 **Dashboards > Manage**，单击右上角 **New Dashboard**，新建面板。



展现效果如下：

- 总数据量写入实时监控 ：对写入数据源的总数据量进行监控。
- 数据来源实时监控 ：对来源于某个特定 log 的数据写入量进行监控。
- 字段平均值监控 ：对某个字段的平均值进行监控。
- `num`字段最大值监控 ：对 num 字段的最大值进行监控。

> **说明**
>
> 此处仅作为示例，无实际业务。

总结

本方案中对系统监控指标和业务监控指标2种方式都进行尝试。若只需要对业务指标进行监控，可省略 Promethus 相关操作。此外，需要注意的是：

1. Ckafka 的版本和开源版本 Kafka 并没有严格对应，方案中 Ckafka2.4.1 和开源 Filebeat-1.14.1 版本能够调试成功。
2. 腾讯云可观测平台中的 Promethus 服务已经嵌入了 Grafana 监控服务。但不支持自定义数据源，该嵌入的 Grafana 只能接入 Promethus，需使用独立灰度发布的 Grafana 才能完成 ES 数据接入 Grafana。

索引设置
默认索引模板说明和调整

最近更新时间：2024-10-15 22:03:51

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。腾讯云 ES 在集群创建时提供了一个默认的索引模板，您可以在控制台内选择 ES 集群管理 ，点击对应集群操作列的 Kibana 按钮，进入 Kibana 界面，并在左侧的 Management 中选择 Dev Tools，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

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。

> **注意**
> 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 order表示模板的优先级，值越大表示优先级越高
    "settings" : {
        "index": {
            "number_of_shards" : 1 number_of_shards表示索引的分片数
        }
    }
}
```

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" //type表示字段的数据类型
        }
```

```
    }
  }
}
```

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 order表示模板的优先级，值越大表示优先级越高
    "settings" : {
        "index": {
            "refresh_interval": "10s"
        }
    }
}
```

使用 Curator 管理索引

最近更新时间：2024-10-16 11:20:11

Curator 是 Elastic 官方发布的一个管理 Elasticsearch 索引的工具，可以完成许多索引生命周期的管理工作，例如清理创建时间超过7天的索引、每天定时备份指定的索引、定时将索引从热节点迁移至冷节点等等。更多 Curator 支持的操作，可查看官方文档的 功能 列表。

Curator 提供了一个命令行 CLI 工具，可以通过参数配置要执行的任务。Curator 还提供了完善的 Python API，这样就可以和腾讯云无服务云函数做结合，例如 使用 Curator 在腾讯云 Elasticsearch 中自动删除过期数据，腾讯云无服务云函数配置了 Curator 的模板，用户应用模板后进行简单的参数配置即可运行。更多的腾讯云无服务器云函数的使用方法可以参考 云函数 。

Curator 用法示例

下面将以定时删除历史过期索引为例，展示如何配置运行 Curator。

安装

在腾讯云 Elasticsearch 集群对应的 VPC 下购买一台 CVM，通过 pip 安装 curator 包。

```
pip install elasticsearch-curator
```

以命令行参数方式运行

下面的命令会过滤索引名称匹配 logstash-20xx-xx-xx 格式且时间为7天前的索引，然后将这些索引删除。

> **注意**
> 示例代码会执行删除操作清除您的数据，请谨慎确认上述语句已经在非生产环境中进行了测试。可以增加 `--dry-run` 参数进行测试，避免实际删除数据。

```
curator_cli --host 10.0.0.2:9200 --http_auth 'user:passwd' delete-indices --filter_list '[{"filtertype":
"pattern", "kind": "prefix", "value": "logstash-"}, {"filtertype": "age", "source": "name", "direction":
"older", "timestring": "%Y.%m.%d", "unit": "days", "unit_count": 7}]'
```

以配置文件方式运行

如您的操作比较复杂，参数太多或不想使用命令行参数，可以将参数放在配置文件中执行。

在指定的 config 目录下，需要编辑 config.yml 和 action.yml 两个配置文件。 并加入到自己指定的路径下。

```
curator_cli --config PATH(PATH替换成自己的config实际目录)
```

定时执行

如需要定时执行，可以将命令配置到 Linux 系统的 crontab 中，也可以直接使用上面提到的腾讯云云函数的定时触发功能。

使用 API

Python API 的使用可参见 文档 。

热温分离与索引生命周期管理

最近更新时间：2025-02-28 11:49:02

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。可通过热温分离来解决这个问题，热温集群可以在一个集群内包含温、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）可以在温节点上以 SATA 磁盘存储。

腾讯云 ES 提供了快速配置构建热温集群的能力，用户可以在腾讯云官网根据业务需要指定温热节点规格，快速建立一个温热分离架构的 ES 集群。

创建热温集群

在购买集群时直接创建

1. 进入腾讯云 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**温数据节点**，并选择温数据节点规格，如下图：

3. 进一步设置集群的其他参数，确认并支付即可。

将现有集群变配为热温集群

在集群详情页面，选择右上角**更多操作 > 调整配置**，在调整配置页面，选择**数据节点部署模式为温热模式**，根据需要设置温热节点的规格和相关配置，将现有集群变配为热温集群。

使用热温集群

节点角色查看

验证节点温热属性，命令如下：

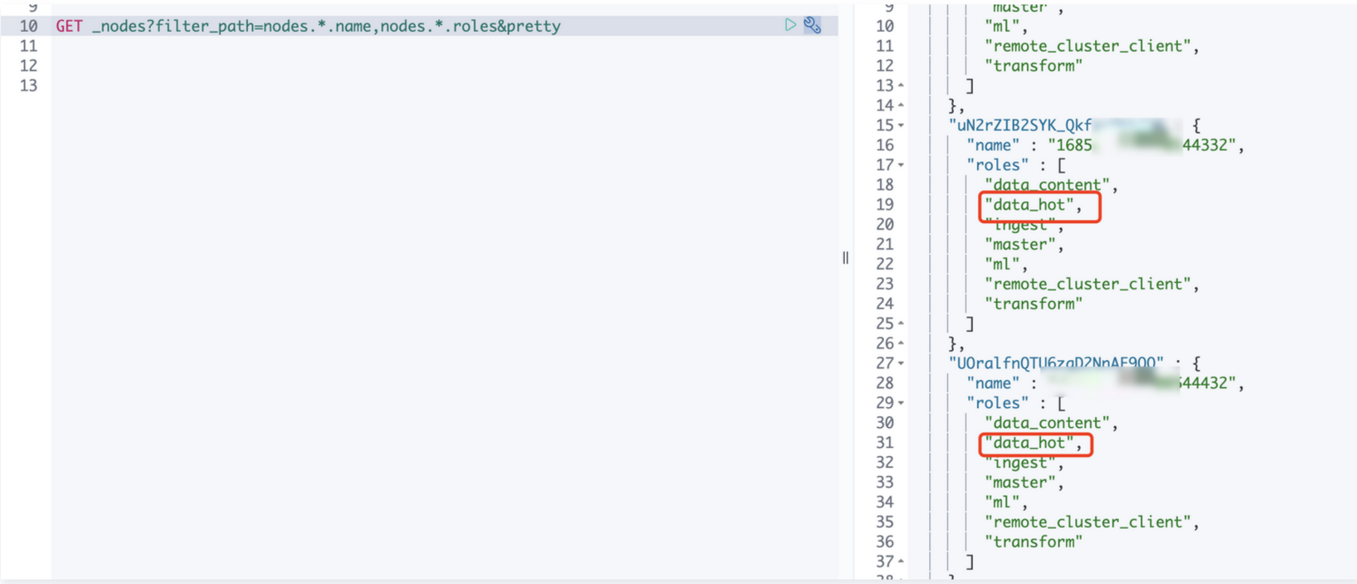
```
GET _cat/nodeattrs?v&h=node,attr,value&s=attr:desc

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

```
node5    temperature     warm
...
```

7.14.2 版本中查看节点属性使用如下命令：

```
GET _nodes?filter_path=nodes.*.name,nodes.*.roles&pretty
```



指定索引温热属性

业务方可以根据实际情况决定索引的温热属性。

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"
}
//7.14.2版本中更改索引为热属性命令如下:
```

```
PUT hot_warm_test_index/_settings
{
 "index.routing.allocation.include._tier_preference": "data_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
```

- 设置索引为温索引。

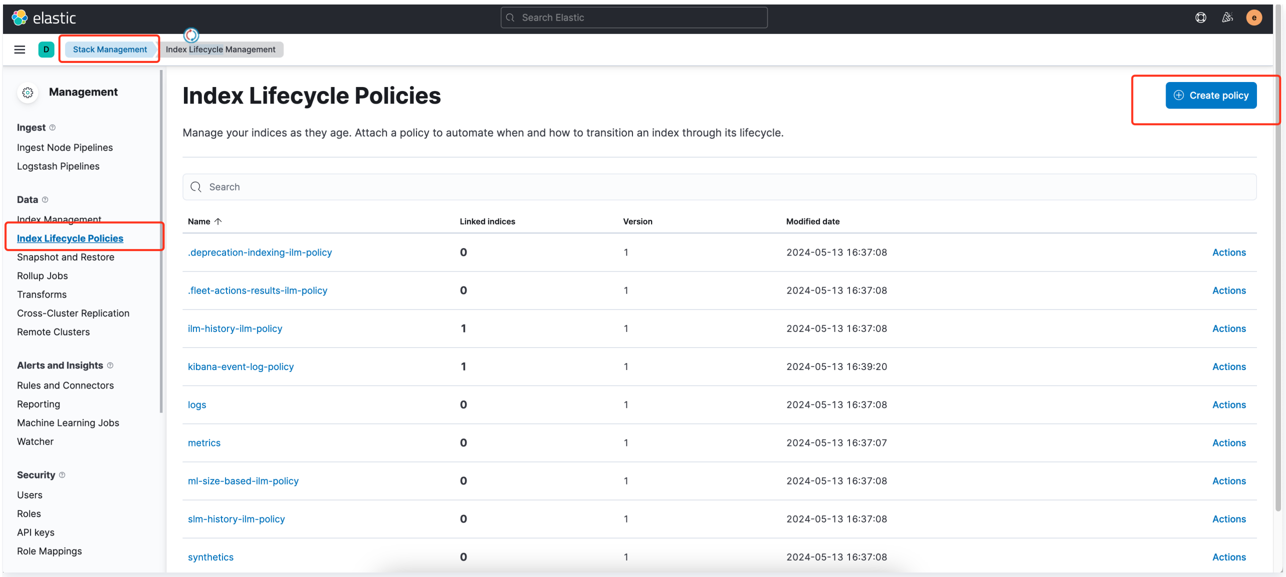
```
PUT hot_warm_test_index/_settings
{
 "index.routing.allocation.require.temperature": "warm"
}
//7.14.2版本中更改索引为温属性命令如下：
PUT hot_warm_test_index/_settings
{
 "index.routing.allocation.include._tier_preference": "data_warm"
}
```

查看分片分配，分片均分配到温节点上。

```
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
```

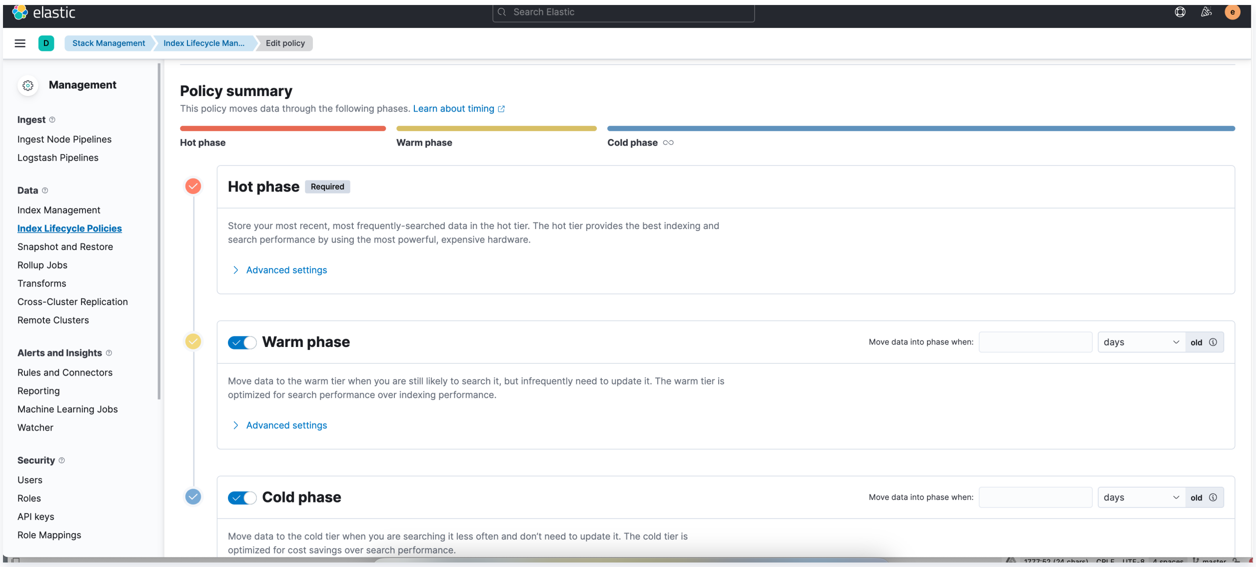
索引生命周期管理

腾讯云目前已提供 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，详情请参见 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，结合温热分离架构，实现索引数据的动态管理。
kibana 中的索引生命周期管理位置如下图（版本 7.14.2）：



单击 **Create policy**，进入配置界面。索引的生命周期为 `Hot phase` 、 `Warm phase` 、 `Cold phase` 三个阶段。

- 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，详情请参见 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，例如对于温热分离策略，这里可选择 `temperature: warm` 属性。另外还可以对索引进行 forceMerge、shrink 等操作，这两个操作具体请参见 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。从温热分离架构可以看出温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 等多个温热属性。如果想使用三层的温热分离，可指定为 `temperature: cold` 。同时还支持对索引的 freeze 操作，详情请参见 freeze API 官方文档。

RAG 应用实践
如何基于非结构化文档实现一套标准的 RAG 应用

最近更新时间：2025-06-11 17:22:42

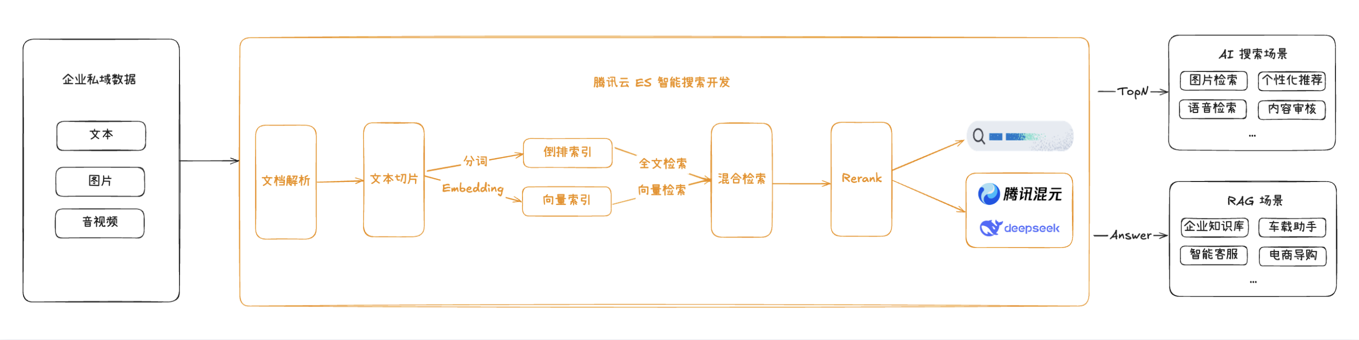
一、引言

本文主要面向已掌握 RAG 基础概念的读者，因此不再重复阐述 RAG 的基本定义及其核心价值。在阅读这篇最佳实践之前，先回答一个问题：在上千万 tokens 上下文窗口模型出现后，为什么我们仍然需要 RAG?

其实很简单，将所有数据都加载到模型上下文中是否能行得通？我将列出以下问题：
1. **可扩展性**：企业知识库以 TB 或 PB 为单位，而不是以 token 为单位，即使有1000万个 tokens 上下文窗口，仍然只能看到可用信息的一小部分。
2. **准确性**：实际的上下文窗口与产品发布时宣传的差异巨大，研究 一致表明，早在模型达到官方宣传极限之前，准确性就已经下降到不可用的情况。
3. **延迟**：将所有内容加载到模型的上下文中会导致响应时间显著延长。对于面向用户的应用程序来说，用户在得到答案之前就会放弃交互，体验很差。
4. **效率**：每次回答一个简单的问题，您都要翻阅整本教科书吗？

在 RAG 中，高效精准的检索其实是非常关键的，混合搜索以及结构化数据支持方面的进步，有助于在知识库中找到正确信息，从而解决上述的问题。同时，RAG 与**长上下文、微调、MCP**这些概念并不排斥，他们可以通过最佳的协同方式去解决前沿模型的局限性：
- RAG 提供对模型知识之外的信息的访问。
- 微调可以改善信息的处理和应用方式，让大模型更加智能、专业。
- 更长的上下文允许检索更多信息，以供模型进行推理。
- MCP 简化了代理与 RAG 系统（和其他工具）的集成。

最近腾讯云 ES 发布的新特性**智能搜索开发**，以 AI 搜索增强版内核为底座，进一步优化了对全文与向量混合搜索的能力支持，从原始文档解析、向量化等原子能力，到查询性能、混合排序效率、搜索结果精准度等提供了全方位的支持和优化，让搜索有了更多想象空间。 在此基础上，还可以与混元、DeepSeek 等大语言模型无缝集成，从而帮助企业进一步高效、灵活地构建知识问答等 RAG 应用，在关键环节也可介入调优，整体流程如下：



接下来具体看下搭建过程。

二、AI 助手搭建

您只需要将下面的文件运行成功，即可快速构建一个AI 问答应用。
- **数据写入：（二选一）**
  - 如果您有大批量文档或正式生产：运行 data_input.py 。
  - 如果您想快速搭 demo 体验：运行 data_input_streamlit.py 。
- **在线问答：** search.py 。

如需将上述文件运行成功，您需要获得以下信息，可以参见 前置条件 章节进行获取。

```
# 腾讯云API客户端配置，获取方式见【3.获取API密钥】
SECRET_ID = "AKID******"     # 替换为您的云API SecretId
SECRET_KEY = "******"        # 替换为您的云API SecretKey

# es集群相关配置，获取方式见【1.购买集群】【2.获取ES访问地址】
# 用户名为 elastic、密码在创建集群时设置。用本地环境测试时，可开启公网访问，实际生产时，建议使用内网访问地址。
```

```
ES_ENDPOINT = "https://es-******.tencentelasticsearch.com:9200"
ES_USERNAME = "elastic"  # Elasticsearch集群用户名
ES_PASSWORD = "******"  # Elasticsearch集群密码
INDEX_NAME = "索引名称"  # 在写入时要保证索引未被创建过，会默认创建索引并设置mapping，写入数据和在线问答的索引保持一致

# 数据入库时，选择cos方式需设置您的cos文件路径
# 要处理的文档URL（推荐将文件存放到cos存储，并开启「公有读私有写」权限）
document_url = "https://xxx.cos.xxx/sample.doc"
```

当然，如果您需要深入了解代码细节，或想调整相关参数达到匹配业务的效果，请参见下一章节 核心代码解析 。

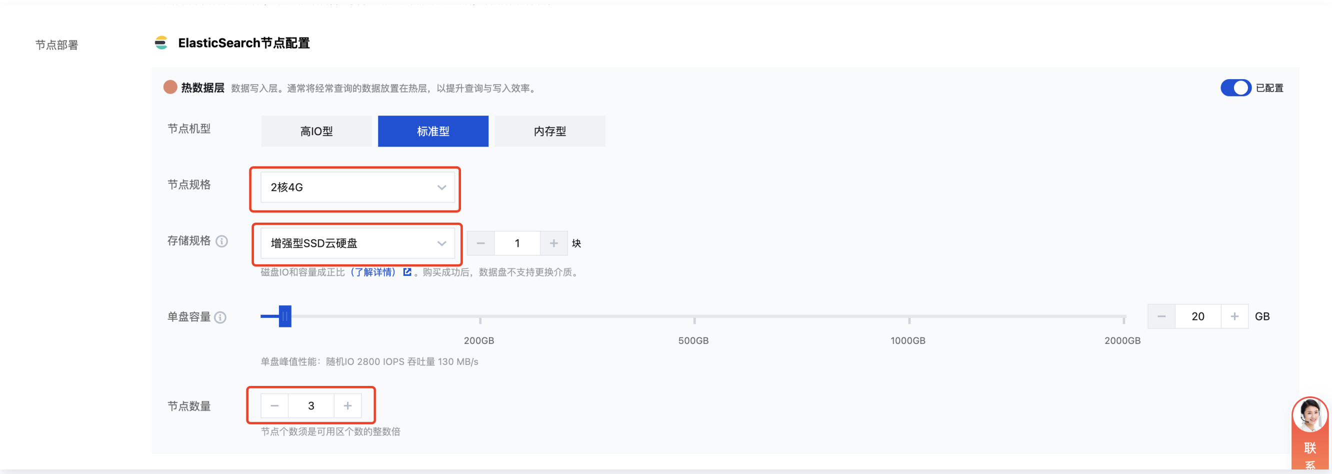
前置条件

1. 购买 ES 集群

1. 登录腾讯云 ES 控制台，进入 ES 集群管理 页面，单击**新建**按钮。

2. 进入新建集群界面，填写以下信息：
   - 用于测试验证计费模式可选**按量计费**。
   - 产品版本选择 **AI 搜索增强版**，专用于对搜索能力、查询性能和稳定性有极高要求的搜索和 RAG 场景。
   - Elasticsearch 版本选择 **8.16.1**，向量能力主要集中在 **8.x** 以上版本。

3. ES 节点配置，测试验证可选择为：
   - 节点规格：**2核4G**。
   - 节点数：默认为 **3**。
   - 磁盘：为**增强型 SSD云硬盘**。
   - 磁盘容量：为 **20GB**。



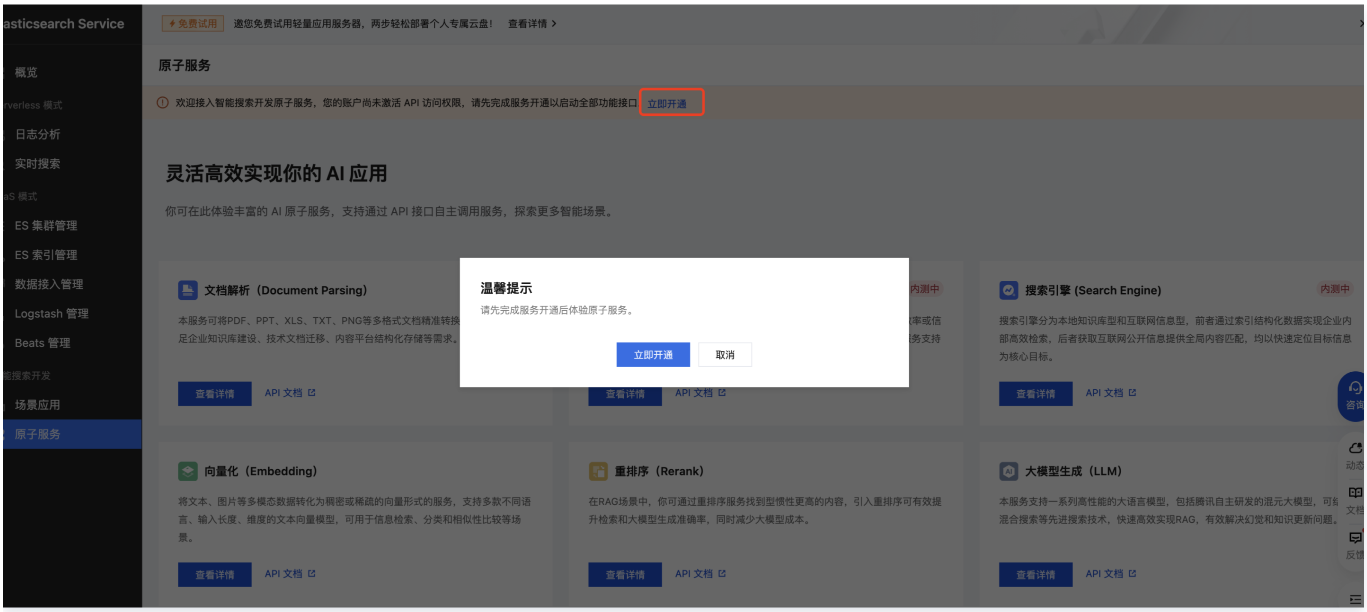
4. 其余配置选择默认即可，注意保存集群访问密码。

2. 获取 ES 访问地址

对应集群 访问控制 下可获取相关信息，用户名为 elastic，密码在创建集群时所设置，如已忘记可以重置。使用本地 mac 测试时，可开启公网访问，实际生产时，建议使用内网访问地址。

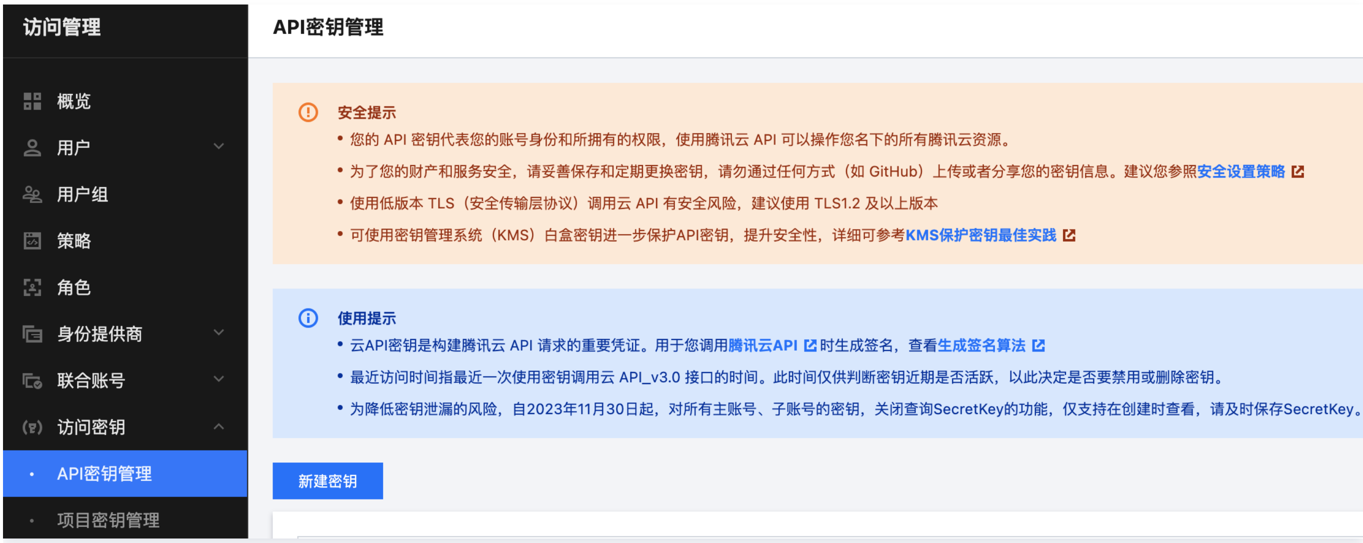
3. 开通服务

现在集群创建好了，在数据入库以及查询时需要用到很多模型服务，腾讯云 ES 智能搜索开发已经集成了文档解析、文本切片、Embedding、Rerank、LLM 服务，只需要在 控制台 开通服务即可，单击**立即开通**，确认免费开通。



4. 获取 API 密钥

进入 API 密钥管理 界面，获取 SecretID、SecretKey。



搭建步骤

1. 安装依赖

```
# Python版本：3.9及以上
pip install elasticsearch==8.16.0
pip install tencentcloud-sdk-python
pip install urllib3 certifi
# 安装文档处理包
pip install python-docx pypdf2 lxml
pip install streamlit     #使用cos数据入库时，无需安装
```

2. 数据入库

这里提供了如下两种方式：

- 如果有大批量的文档需要进行处理，或者您需要上生产，那建议按 COS 上传 方式入库，这种方式更加稳定和安全。
- 如果您只是简单体验一下 AI 问答 demo，并通过界面可视化的方式操作，那可以按 本地上传 的方式入库。

方式一：COS上传（可选）

1. 先将需要处理的文档拖入对应 COS 桶，具体操作步骤：进入 对象存储控制台，单击**存储桶列表**，在对应文件存储桶，单击**上传文件**，出现以下弹框，拖拽需要上传的文件即可。

2. 然后设置文档权限，这里需要对文档进行解析切片处理，所以需要开放**公有读私有写**权限，具体操作：进入 对象存储控制台，单击**存储桶列表**，找到对应文件存储桶，单击**文件详情**，开启**公有读私有写**权限。

3. 在 data_input.py 文件中，修改 main 方法中的 document_url 为您上传的 COS 文件地址。

```
# 要处理的文档URL（推荐将文件存放到COS存储，并开启「公有读私有写」权限。
document_url = "https://xxx.cos.xxx/sample.doc"
```

4. 然后，在终端运行该文件：

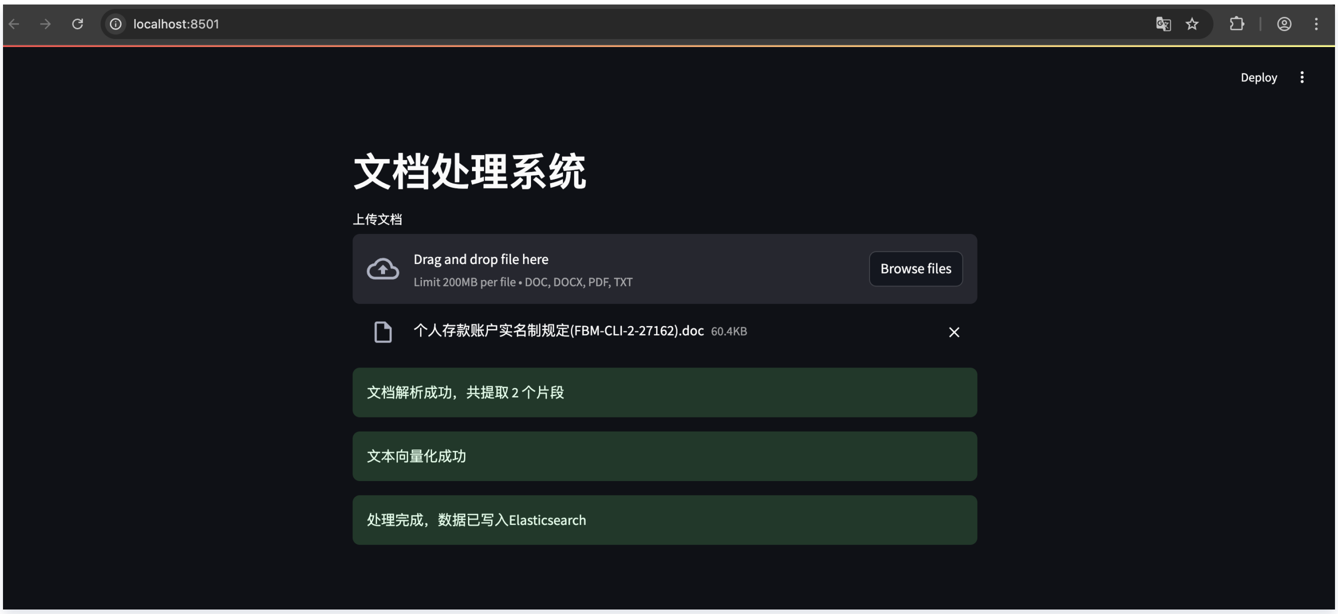
```
python data_input.py
```

方式二：本地上传（可选）

1. 在终端运行 data_input_streamlit.py 文件，在终端执行如下命令。

```
streamlit run data_input_streamlit.py
```

2. 将要上传的文档拖拽进去，或者单击 **Browse files** 按钮，选择文件上传，效果如下：



3. 在线问答

该步骤主要是运行 search.py 文件，在终端执行如下命令：

```
streamlit run search.py
```

在输入框中提出相关问题，系统会进行**混合搜索 > 重排序 > 组装 Prompt > 大模型回答**一系列动作，整个流程中间内容可见，最后生成回答内容。效果如下：

详情参见 录屏文件。

三、核心代码解析

1. 数据入库

对应 data_input.py 和 data_input_streamlit.py 文件，主要讲述一个非结构化文档从解析、切片、到 Embedding 后写入 ES 索引的过程，具体如下：

1.1. 文档解析/切片

企业大部分数据都是以文档形式存在的非结构化数据，如何让文档数据变得可读，是实现 RAG 的第一步。我们通过调用 文档切片 服务对文档数据先进行文档识别转 MD 格式，然后进行语义切片预处理。

常量 MAX_CHUNK_SIZE 即为文本切片单条文本最大字符数，不小于1000，可以根据您的具体业务，确定合适的最大字符数。在 Embedding 过程中，选定的模型 Tokens 限制应大于您设定的最大字符数，否则会出现截断的可能。向量化模型参数请参见 向量化（Embedding）模型列表 。

常量 DOCUMENT_CHUNK_MODEL_NAME 即为选择的文档解析切片模型，“doc-tree-chunk”包含了文档解析和切片部分，这里可以直接用。

```
MAX_CHUNK_SIZE = 2000  # 文本切片单条文本最大字符个数（根据具体选用的Embedding模型tokens设定，最小值不小于1000）
DOCUMENT_CHUNK_MODEL_NAME = "doc-tree-chunk"

# 调用ChunkDocumentAsync接口创建文档切片任务
params = {
    "Document": {
        "FileType": file_type,
        "FileUrl": file_url,
        "FileName": os.path.basename(urlparse(file_url).path)
    },
    "Config": {
        "MaxChunkSize": MAX_CHUNK_SIZE  # 可根据需要调整切片大小
    },
    "ModelName": DOCUMENT_CHUNK_MODEL_NAME
}

response = retry_request(client.call_json, "ChunkDocumentAsync", params)
task_id = response['Response']['TaskId']
logger.info(f"Created document chunking task with ID: {task_id}")
```

因为文档解析/切片是个异步任务，所以需要通过 查询接口 获取结果。
解压获取到的结果，切片好的数据存放在 xxx.jsonl 文件内，对应 page_content 即为切片数据。

```
try:
    with zipfile.ZipFile(io.BytesIO(response.content)) as zip_file:
        # 检查zip中是否有.jsonl文件
        jsonl_files = [f for f in zip_file.namelist() if f.endswith('.jsonl')]
        if not jsonl_files:
            error_msg = "No .jsonl file found in the zip archive"
            logger.error(error_msg)
            raise Exception(error_msg)

        # 处理每个jsonl文件
        for file_name in jsonl_files:
            with zip_file.open(file_name) as f:
                # 逐行读取jsonl文件
                for line in f:
                    try:
                        data = json.loads(line.decode('utf-8'))
                        if 'page_content' in data:
                            doc_list.append(data['page_content'])
                    except json.JSONDecodeError as e:
                        logger.warning(f"Failed to parse JSON line: {str(e)}")
                        continue
```

1.2. 切片 Embedding

在上面已经将原始文档处理成切片了，这是检索的最小单元，我们需要将切片向量化。相关详情请参见 Embedding 模型信息 。

如果文本内容过长，分片个数较大时，建议分批次获取 Embedding，建议每次设置最大元素个数在1 - 20之间，可以保证更高的效率。

```
BATCH_EMBEDDING_SIZE = 20  # 单次向量化入参设置的最大元素个数（建议1-20）
document_embedding_list = batch_get_text_embeddings(
    client=tencent_cloud_client,
    docs=chunk_list,
    batch_size=BATCH_EMBEDDING_SIZE,  # 根据API限制调整单个批次获取向量化的元素个数
    max_workers=2  # 根据服务器CPU核心数调整
)

# 调用GetTextEmbedding进行向量化。模型选择参考：https://cloud.tencent.com/document/product/845/118215
EMBEDDING_MODEL_NAME = "bge-base-zh-v1.5"
response = retry_request(
    func=client.call_json,
    action="GetTextEmbedding",
    params={
        "ModelName": EMBEDDING_MODEL_NAME,
        "Texts": docs
    }
)
```

1.3. 文本&向量数据写入

创建向量索引，定义索引 Schema，注意向量维度需与上述 Embedding 模型维度保持一致。相关详情请参见 模型维度信息 。

```
# 定义索引映射
mappings = {
    "mappings": {
        "properties": {
            "content_embedding": {
                "type": "dense_vector",
                "dims": EMBEDDING_MODEL_DIMS,
                "index": True,
                "similarity": "cosine",  # 指定相似度计算方式为余弦相似度
                "index_options": {
                    "type": "int8_hnsw",  # 使用int8量化的HNSW（Hierarchical Navigable Small World）算法进行索引
                    "m": 16,  # HNSW参数，表示每个节点在图中连接的邻居数，影响索引的构建和搜索性能
                    "ef_construction": 100  # HNSW参数，表示构建索引时考虑的候选列表大小，影响索引质量和构建时间
                }
            },
            "content": {
                "type": "text",
                "index": True
            }
        }
    }
}
```

通过 Bulk API 批量写入数据。原始文档信息和向量化之后的数据均进行存储，方便 ES 混合搜索，提高召回率。
原始文档写入 content 字段，向量写入 content_embedding 字段。

```
actions = [
    {
        "_index": index_name,
        "_id": str(doc["index"]),
        "_source": {
            "content": doc["content"],  # 原始文本内容
            "content_embedding": doc["content_embedding"]  # 向量
        }
    }
    for doc in embeddings
```

```
]
```

2. 在线问答

对应 search.py 文件，主要讲述用户的一个 Query 语句进来后，经过向量化，从 ES 中进行混合检索召回 TopK，然后经过 Rerank 再精排过滤后，结合 Prompt 工程喂给大模型进行总结回答，具体如下：

2.1. Query 向量化

对 Query 语句进行向量化，具体调用方式本质上是与切片向量化一致，不再赘述。

2.2. 文本&向量混合检索

指定向量字段，对 Query 向量进行向量相似度检索，获取前 k 条向量数据，同时进行全文检索，在最后将两者进行融合排序。
关于文本与向量的权重可以通过"weights"设置，可根据具体业务数据调试，取效果最优的权重即可。
通用场景参考如下：

- 电商搜索：提升关键词权重（权重比 3:1 或更大）
- 内容推荐：增强语义相关性（权重比 1:2 或更大）
- 知识库检索：平衡语义与关键词（权重比 1:1 微调）

```
# 使用8.16集群的混合搜索语法(并行召回：返回 query 和 knn 召回结果的并集)
mix_query = {
    "retriever": {
        "rank_fusion": {
            "retrievers": [
                {
                    "standard": {
                        "query": {"match": {"content": query}}
                    }
                },
                {
                    "knn": {
                        "field": "content_embedding",
                        "query_vector": [...],     # 向量字段
                        "k": 5,
                        "num_candidates": 10
                    }
                }
            ],
            "weights": [2, 1],
            "rank_constant": 20
        }
    }
}
```

2.3. 召回结果重排序

调用 Rerank 模型进行重排序。

- Rerank 模型请参见 Rerank 模型列表 。
- TopN 即为排序返回的 top 文档数量。
  - 如果没有指定，则返回全部候选 doc。
  - 如果指定的 TopN 值大于输入的候选 doc 数量，返回全部 doc。

```
RERANK_MODEL_NAME = "bge-reranker-large"
json_object = {
    "ModelName": RERANK_MODEL_NAME,
    "Query": query,
    "ReturnDocuments": True,
    "Documents": docs,
    "TopN":5     # 排序返回的top文档数量, 如果没有指定则返回全部候选doc，如果指定的TopN值大于输入的候选doc数量，返回全部doc
}
```

```

```

2.4. 组装 Prompt

设置自己的 Prompt 模板，注意预留 user_query 和 documents_str 占位符，分别代表问题与检索到最相关的切片上下文。

```
PROMPT_TEMPLATE = """
# 任务说明
您是一位专业的AI助手，需要基于提供的参考文档准确回答用户问题。请严格遵循以下要求：

# 用户问题
{user_query}

# 参考文档（已按相关性排序）
{documents_str}

# 回答要求
   - 简明扼要的核心答案（1-2句话）
   - 详细解释或分点说明（3-5点，视问题复杂度而定）
   - 可选的补充信息或建议（如适用）

   - 对于事实性问题：提供准确数据或事实
   - 对于观点性问题：呈现不同角度的分析
   - 对于操作性问题：给出步骤清晰的指导
   - 对于比较性问题：进行客观对比分析

   "根据现有资料，我暂时无法提供令人满意的答案。建议您补充更多相关信息或重新提问。"

# 特别注意
1.保持回答专业、准确且流畅
2.如文档中有数据或研究结果，请优先引用
3.根据问题复杂度控制回答长度在200-400字之间
4.对于专业术语，必要时提供简单解释
5.避免使用"根据文档"等冗余表述，直接整合信息到回答中
"""
```

2.5. 调用 DeepSeek 回答

调用 deepseek-v3 大模型进行总结回答，当然您可以切换不同 hunyuan 以及 deepseek 大模型进行体验。如果期望返回值是流式响应时，则设置 Stream 为 True。

```
# 设置deepseek-v3模型（这里可以设置为您想要的模型，参考：https://cloud.tencent.com/document/api/845/117810）
LLM_MODEL_NAME = "deepseek-v3"

# 调用大模型回答，content即为build_llm_prompt返回组装后的prompt
def generate_llm_response(content: str) -> Dict[str, Any]:
    """
    调用大模型生成回答

    Args:
        content: 要发送给大模型的prompt内容

    Returns:
        大模型的响应结果

    Raises:
        Exception: 如果调用大模型过程中出现错误
    """
    if not content.strip():
        raise ValueError("空内容无法生成回答")
```

```
    json_object = {
        "Messages": [{"Role": "user", "Content": content}],
        "ModelName": LLM_MODEL_NAME,
        "Stream": False    # 可以设置成True，流式输出
    }

    try:
        response = tencent_cloud_client.call_json("ChatCompletions", json_object)
        return response
    except Exception as e:
        logger.error(f"大模型调用失败: {e}")
        raise
```

四、总结

腾讯云 ES 凭借其在传统 PB 级日志和海量搜索场景中积累的丰富经验，通过深度重构底层系统，成功地将多年的性能优化、索引构建和运营管理经验应用于 RAG 领域，并积极探索向量召回与传统搜索技术的融合之道，旨在充分发挥两者的优势，为用户提供更加精准、高效的搜索体验。未来，腾讯云 ES 将持续深耕智能检索领域，在成本、性能、稳定性等方面持续提升，帮助客户降本增效的同时实现业务价值持续增长。

其实，用 ES Inference API 构建 RAG 应用会更加便捷，相关能力马上推出，欢迎关注下一篇实践文章。

腾讯云 ES RAG 实践教程：百行代码轻松实现 ES 帮助文档的智能问答

最近更新时间：2024-12-13 17:19:22

引言

信息化技术的飞速发展，使得海量数据爆发式增长。一方面，越来越多的数据可以为我们的生活带来便利；另一方面，软件开发带来巨大的挑战——图片、声音、视频等不同结构的数据越来越多地出现，为搜索分析带来巨大的挑战，传统的关键词搜索，搜索结果局限于输入的关键词，用户体验较差。
因此，向量检索的出现，给我们提供了一个新的思路，向量数据库将非结构化、半结构化甚至是结构化等数据以向量形式存储，实现相似度搜索、聚类、降维等操作，结合机器学习模型，为用户更加智能的搜索服务。

然而，集**文本搜索** + **向量检索** + **AI 能力**于一身，且成熟稳定、值得信赖的云端搜索引擎，在业内屈指可数，腾讯云大数据 Elasticsearch Service 近期首发上线的 ES 8.8.1版本，该版本具有以下优势：

- 提供了强大的云端 AI 增强与向量检索能力；
- 支持在端到端搜索与分析平台中实现 NLP 服务、向量搜索以及与大模型的集成；
- 10亿级向量检索平均响应延迟控制在毫秒级，助力客户实现由 AI 驱动的高级搜索能力，为搜索与分析带来全新的前沿体验。

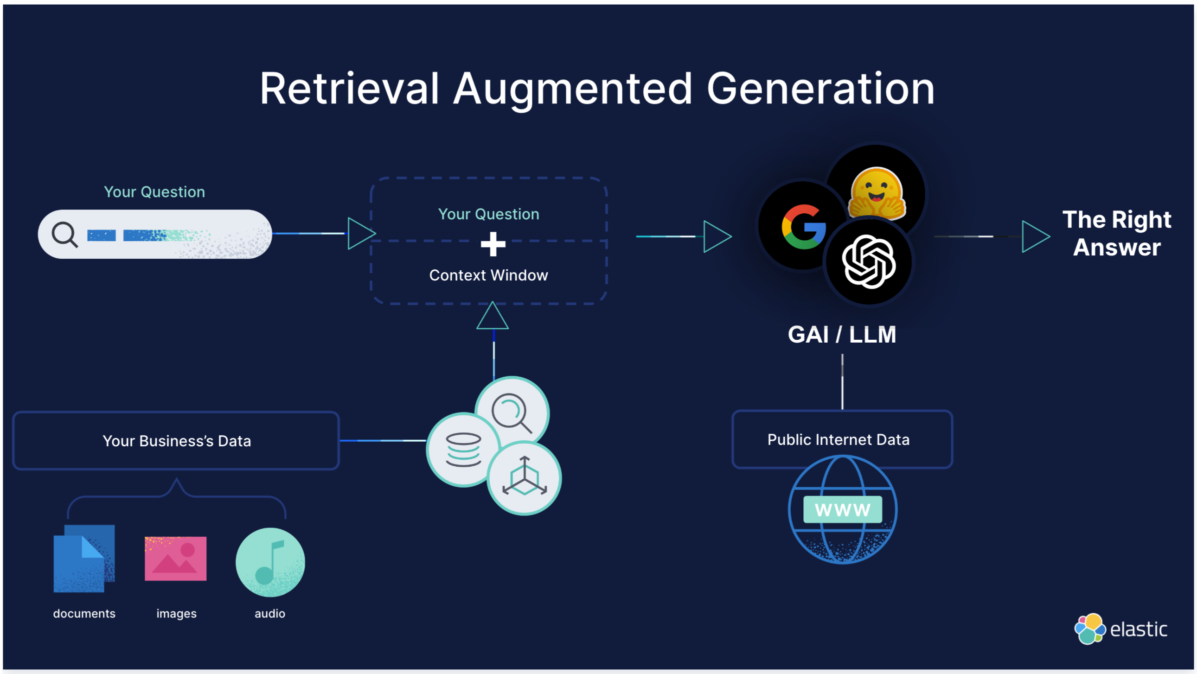
本文将结合腾讯云 ES 与 ChatGPT，演示如何通过百行代码实现 AI 问答。

RAG 介绍

在进入正式的介绍之前，我们先来介绍 RAG，随着搜索引擎技术的不断发展，我们对于查询的需求也日益提高。传统的关键词搜索已经无法满足用户对于查询准确性和效率的要求。为此，我们引入了语义搜索技术。通过使用先进的 NLP 服务技术，语义搜索能够更好地理解用户的查询意图，并返回更相关的搜索结果。而随着机器学习技术的持续发展，特别是 ChatGPT 等生成式大模型的火爆，一个新的技术方向应运而生 —— RAG。
但是真正的理解什么是 RAG 并不容易，实现 RAG 就更难。现状是大多数时候用户会简单地把实现 RAG 理解为在企业中加入一个向量数据库。但 RAG 是一个复杂的概念，它不仅仅是一个向量数据库，实现 RAG 需要对业务场景有深入的理解，并且需要进行大量的数据处理和算法优化，用户的行为的理解和反馈也是最终效果达成的重要关键。因此，我们需要的更多地是一个解决方案，而非仅仅一个高性能的向量数据库。

什么是 RAG?

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



为什么要做 RAG?

举个例子，在官网文档上，目前的文本检索方式存在以下问题：

- **词汇不匹配（lexical mismatch）：** 当文档和查询使用不同的词语来表达相同或相似的意思时，传统的搜索方法可能无法找到相关的文档。例如，对于ELK技术栈来说，“logstash”和“ls”在语境中是一个意思，但是它们在词汇上是不同的。

- **语义不匹配（semantic mismatch）：** 当文档和查询使用相同或相似的词语来表达不同或不相关的意思时，传统的搜索方法可能会找到不相关的文档。例如，“Elastic”可以指弹性，也可以指代一家科技公司，但是它们在语义上是不同的。
- **语言不匹配（language mismatch）：** 当文档和查询使用不同的语言时，传统的搜索方法可能无法找到相关的文档。例如，“Serverless 架构”和“无服务器架构”都可以指代同一种架构，但是它们在语言上是不同的。

因此，我们需要通过语义搜索的方式解决以上问题，找到对应的文档。但客户实际的需求可能会更复杂，客户的问题可能是：

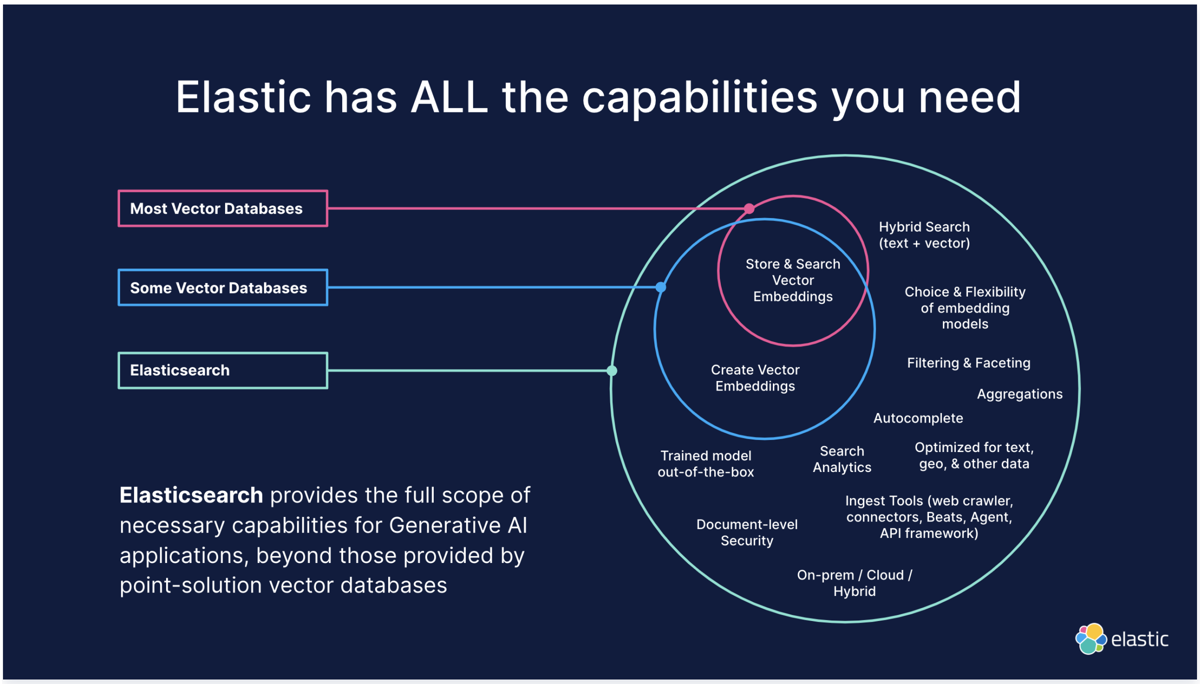
- ES Serverless 服务如何收费？请给我一个详细的例子。
- 如何销毁 ls 实例，给我具体的步骤。

在这种需要学习文档并给出指导性意见的场景中，仅仅依靠全文检索、语义搜索或混合搜索是无法达到令人满意的效果的。为了实现这一目标，我们需要结合大模型，并在企业私有数据的基础上，通过检索增强生成（RAG）来提升效率和准确性，同时给出指导性的步骤。

使用腾讯云 ES 进行 RAG

腾讯云 Elasticsearch Service（ES）是云端全托管海量数据检索分析服务，拥有高性能自研内核，集成 X-Pack。ES 支持通过自治索引、存算分离、集群巡检等特性轻松管理集群，也支持免运维、自动弹性、按需使用的 Serverless 模式。使用 ES 您可以高效构建信息检索、日志分析、运维监控等服务，它独特的向量检索还可助您构建基于语义、图像的AI深度应用。

腾讯云 ES 近期上线的 8.8.1 版本，包含了实现 RAG 所需的所有功能的综合性引擎，从向量存储和搜索到向量生成和混合搜索排序，甚至还包括了专有的向量模型：



为了给您展示完整的使用 Elasticsearch 进行 RAG 的过程，我们将通过 腾讯云 ES 和 ChatGPT，在腾讯云 ES 的官网文档实现一个案例。 步骤如下：

- 创建一个 ES 集群，并配置相关的参数和插件。
- 收集并了解我们的数据，在本例中，腾讯云ES帮助文档将作为我们的知识库，用来研究如何通过语义搜索并结合大模型来增强。
- 处理数据，使其能进行语义搜索。使用 BAAI/bge-base-zh 模型将文本转换为向量，并将向量和文本一起存储到 ES 中。
- 使用 ES 提供的 API 或者 Kibana 工具来进行向量+文本的混合搜索。
- 使用大模型提供的 API 或者 SDK 来进行 RAG 的对话生成。

下面我们将详细介绍每个步骤的具体操作。

创建 ES 集群

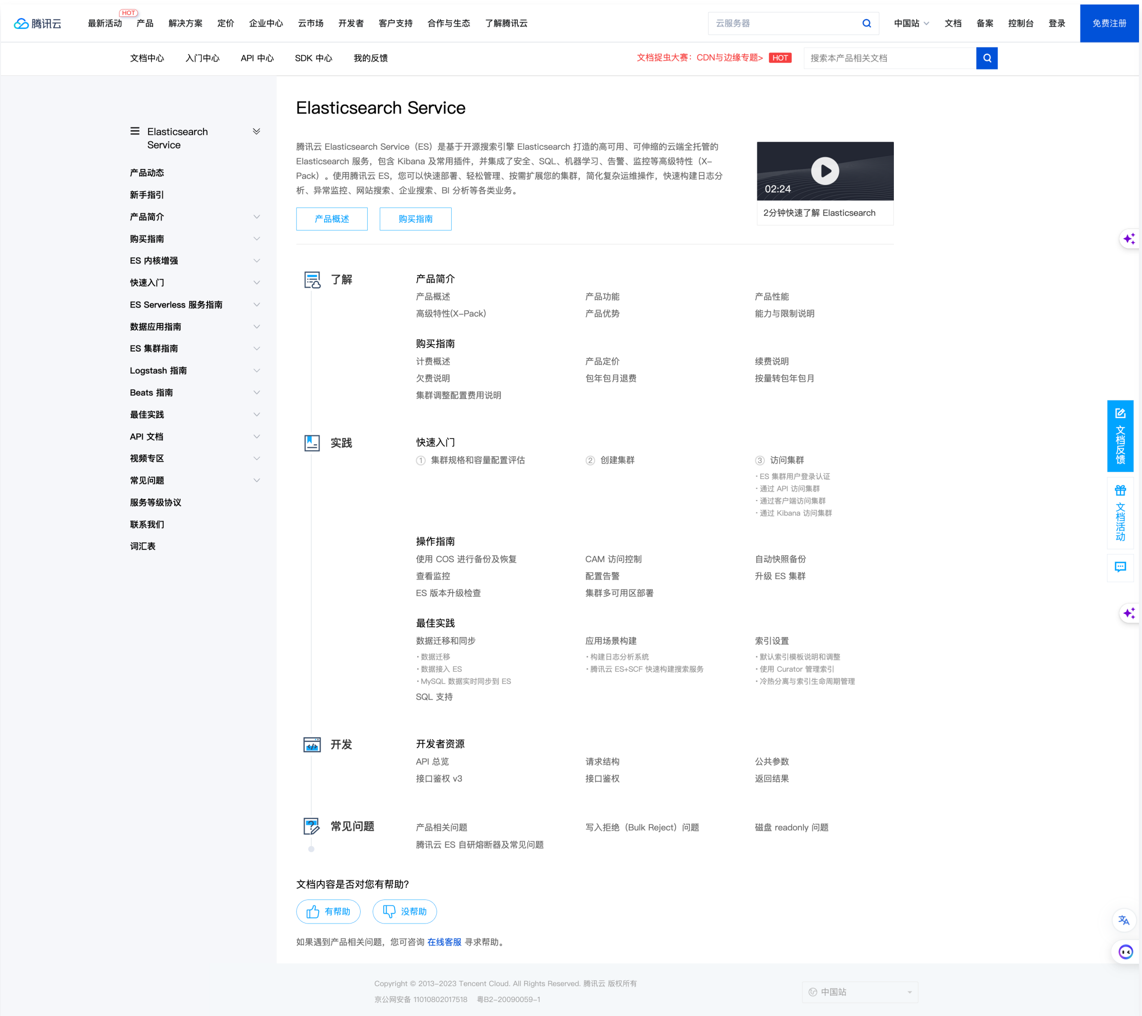
在 腾讯云 ES 上申请 ES 8.8.1 集群。

注意：
由于需要部署 embedding 模型，我们尽量保证所选足够的内存。如需实现本例中的内容，建议选择配置8G及以上内存的数据节点。

收集并了解我们的数据

要在具体的场景下实现 RAG，带给我们的第一个问题是我们要如何收集该场景下可能会用到的数据。很不幸的是，这些数据大多数时候并非是管理良好的结构化数据，例如最新版的监管文档、企业内部的员工手册、传感器数据和日志和促销记录等。这些数据可能存放在 Wiki、网盘、邮件、日志里，或者分散在多个不同的数据库中。如何采集和理解它们，并且如何将其转换为搜索比重不同的字段，就会存在很大的问题，而且是留给开发者自己处理的问题。

以 腾讯云 ES 的帮助文档 为例子，文档是以 Markdown 的格式存储在 github 上的。并且会随时更新。在其上做 RAG ，您需要联系对应的业务团队，获取数据的同步权限，而且还需要定期同步更新。另外，这是一个典型的半结构化数据，虽然有 HTML 和 Markdown 的标签，但实际上，每一篇文章的内容和格式是不固定的，如何提取合适的 “context window”，需要我们在搜索引擎中重新有效的组织数据。



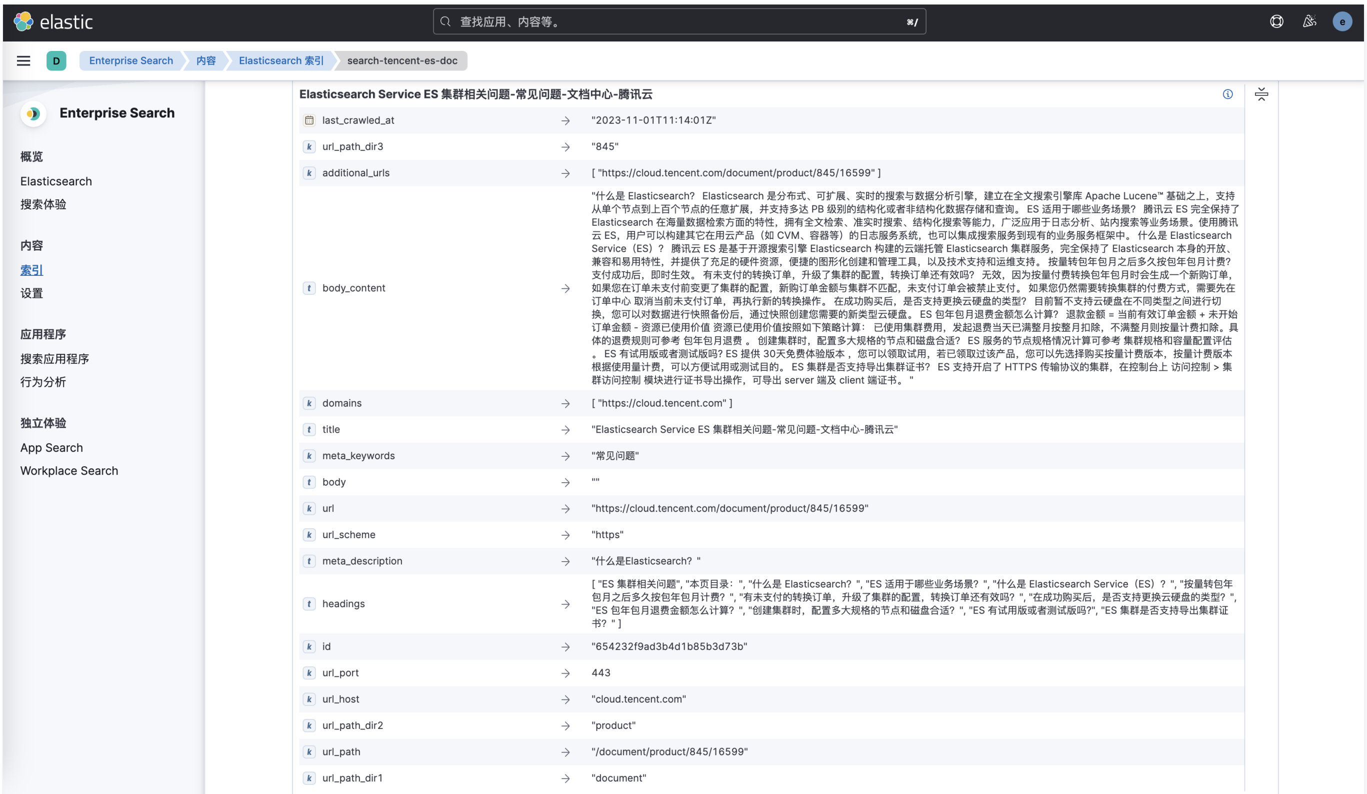
面对这些问题，ES 提供的企业搜索功能，为我们最大程度的减少了对应的工作。从沟通，到采集，以及信息的提取，再到数据的理解。

为了减少阅读的时间，这里制作了一个短视频，方便您快速了解，我们是如何通过爬虫，抓取页面数据的，以及抓取下来的数据，如何通过管道进行处理和清洗。具体参考如下视频演示：

观看视频

> **说明：**
> 如果您需要使用企业搜索功能进行数据摄入，请联系我们进行相关操作。

完成数据的采集，以及简单的处理后，我们可以直接在 Kibana 中了解我们场景中的数据：



可以看到，核心的内容包含在 `body_content` ， `headings` ， `title` 等三个字段中。如果要以语义搜索的方式来支持 RAG，那么，需要为这三个字段生成向量，并且需保留原始的文本字段用于页面展示引用，并发送给大模型。 特别是其中的 `headings` 字段（该字段由爬虫自动生成），包含了文档中所有章节的标题。这种有效的概括和总结性的文字，特别适合在不进行chunk、不分段存储的情况下，作为文本内容的向量表示，示例如下所示：

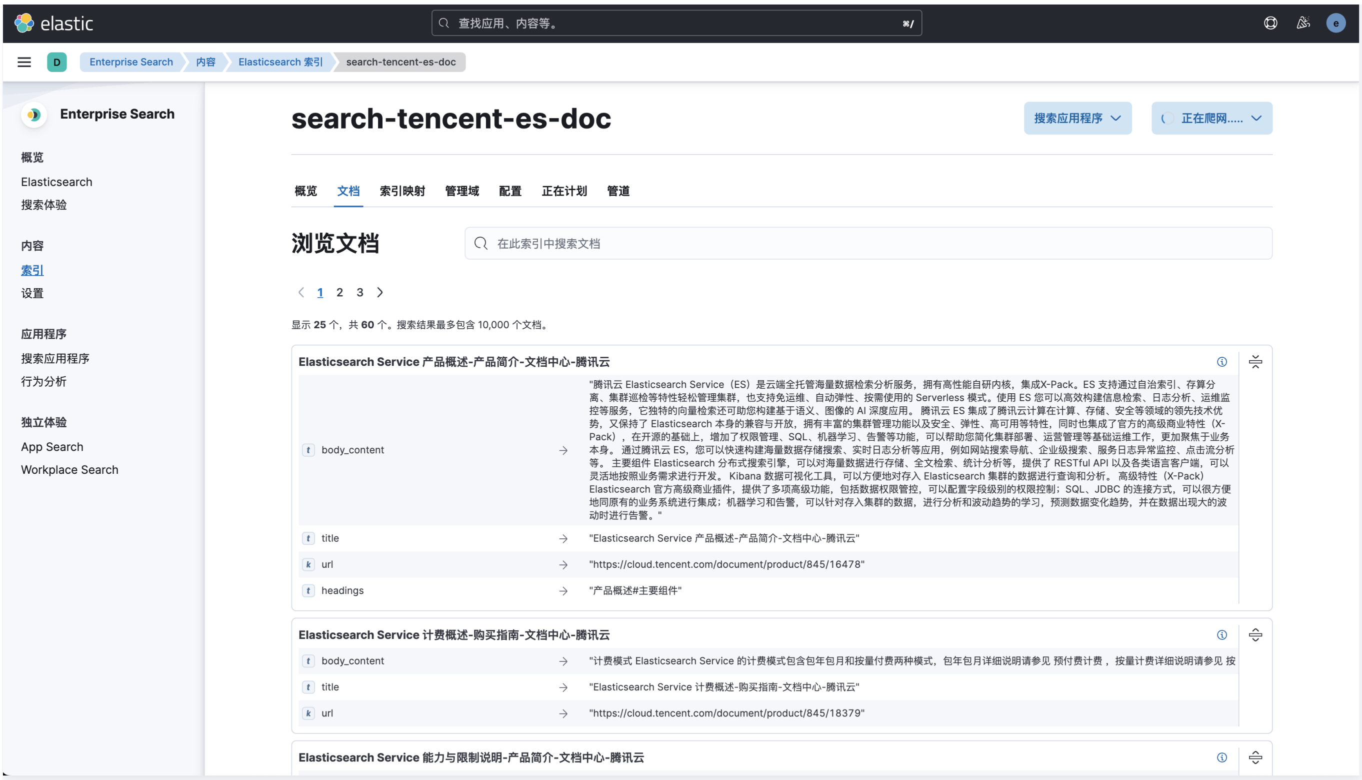
```
"headings":"集群规格和容量配置评估#本页目录：#存储容量评估#计算资源评估#实例类型选择及测试#分片数量评估"
```

而剩余的字段中， `url` 则可用于链接到源文档。剩下的字段均可删除。 而因为是数组类型，需要转化为 text 才能被转化为向量，因此，通过以下数据管道后：

```
PUT _ingest/pipeline/search-tencent-es-doc@custom
{
  "version": 1,
  "description": "Enterprise Search customizable ingest pipeline for the 'search-tencent-es-doc' index",
  "processors": [
    {
      "rename": {
        "field": "doc_content",
        "target_field": "body_content",
        "ignore_missing": true
      }
    },
    {
      "join": {
        "field": "headings",
        "separator": "#"
      }
    },
    {
      "remove": {
        "keep": [
          "headings",
          "body_content",
          "url",
          "title"
        ],
```

```
        "ignore_missing": true,
        "description": "移除多余字段"
      }
    }
  ]
}
```

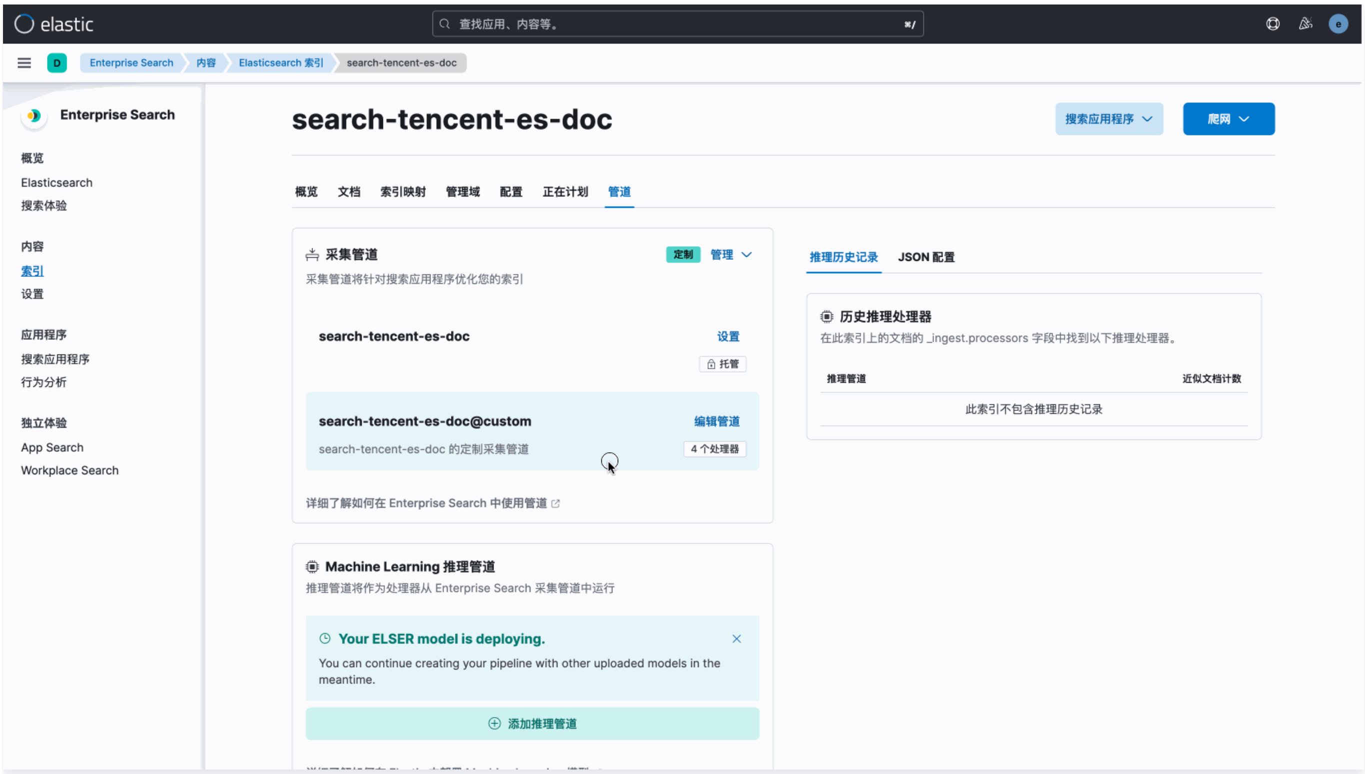
清洗后的数据为：



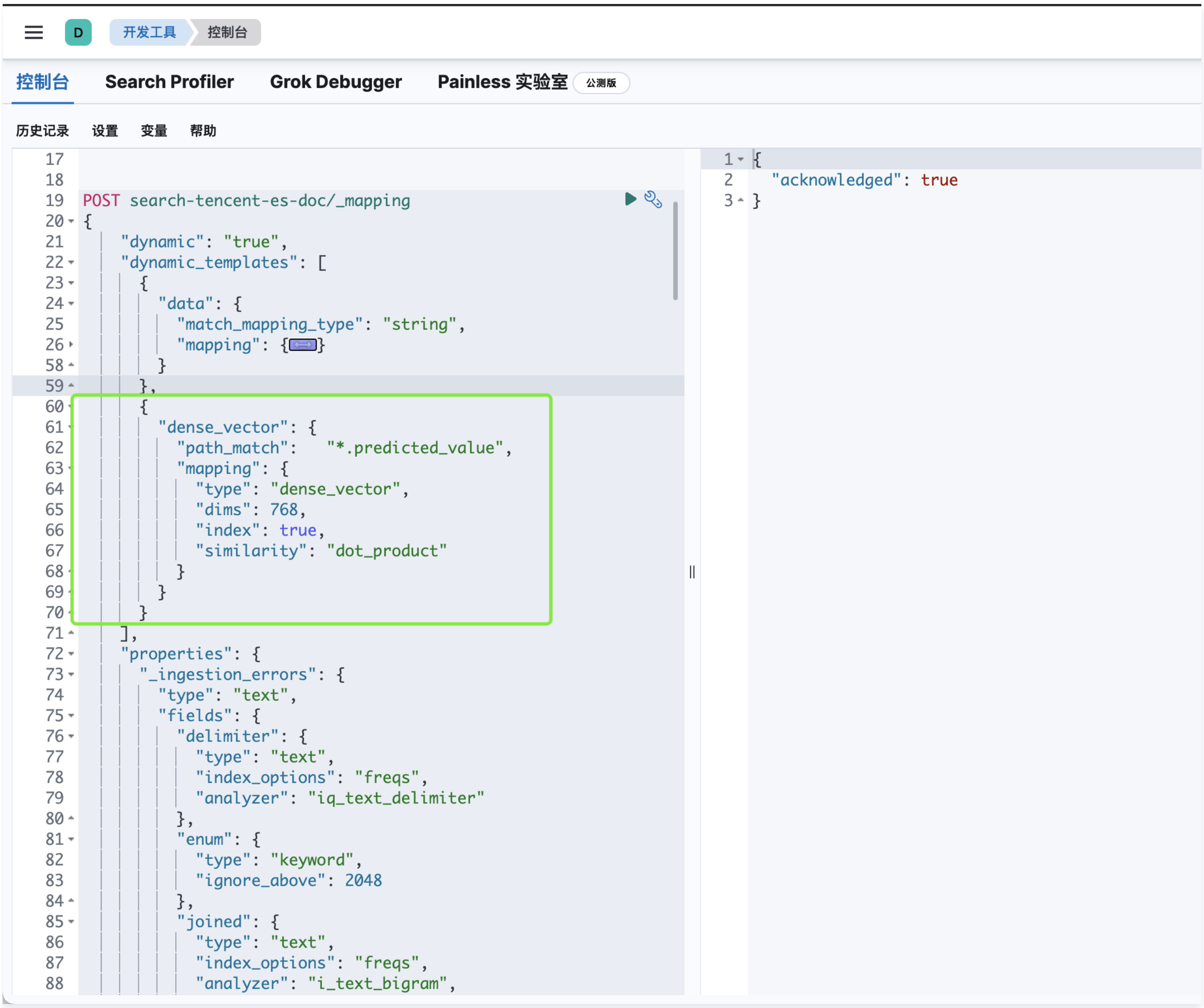
处理数据，使其能进行语义搜索

因为在一个 RAG 系统中，用户是通过自然语言对话的方式进行提问的，因此，检索本地数据的引擎必须支持语义搜索。通常，我们可以稀疏向量检索和密集向量检索两种方式来实现语义搜索。

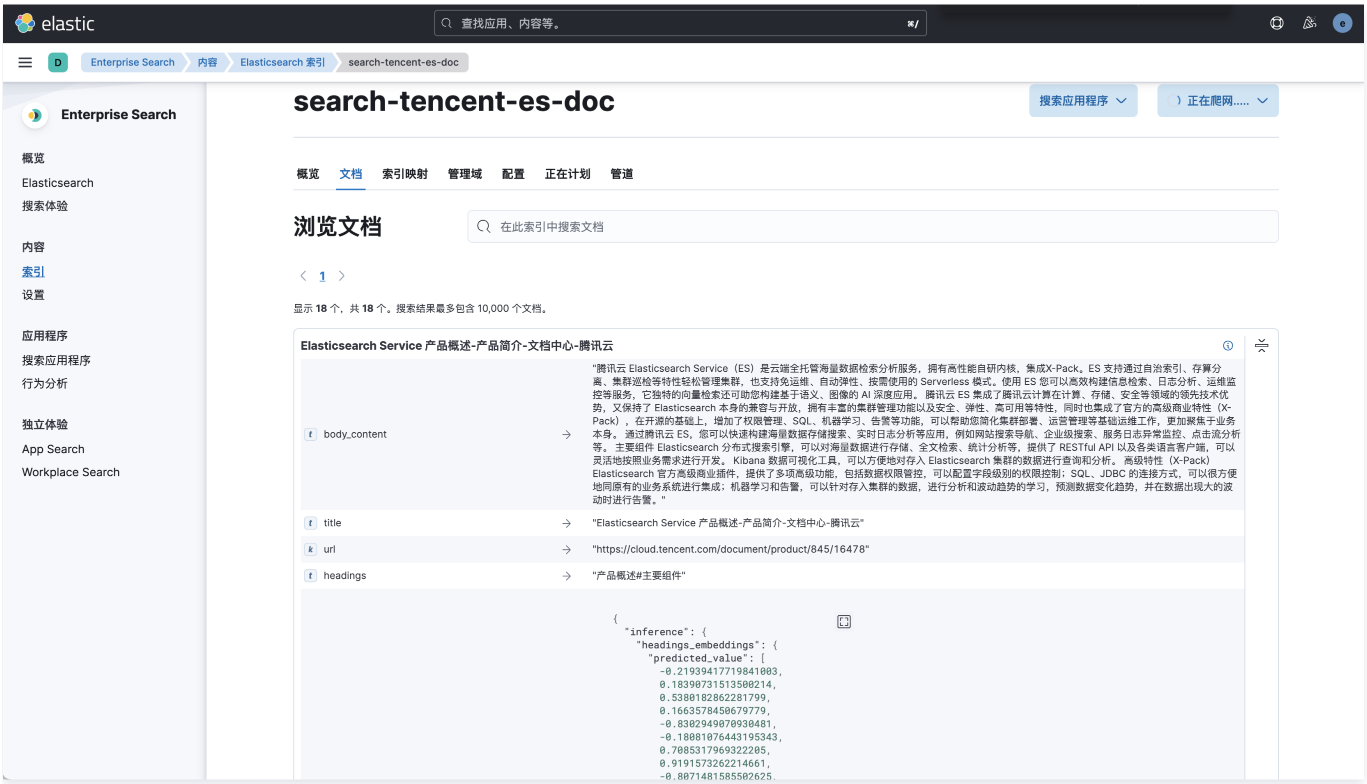
ES上支持两种方式，并且为稀疏向量检索提供了 ELSER 专有模型。但在本例中，我们将使用密集向量检索作为演示。 为此，我们先添加一个机器学习的推理管道，用于向量的生成：



这里，我们使用的模型是 BAAI/bge-base-zh（但实际上，我们可以在ES中部署多个 embedding 模型，通过调试及更换模型，在不同的场景上可以获得不同的效果），该模型将输入转化为768维的向量，因此，我们需要定义一个密集向量字段，用以存储该模型生成的数据:



最后的包含向量的数据为：



使用 ES 进行向量检索 + 文本搜索的混合搜索

在常见的一些教程中，要进行语义搜索还需要结合 Langchain 等封装好的模块工具，才能快速的开展语义搜索。而使用腾讯云 ES，我们只需要短短几行代码就能实现。

> **说明：**
> - **到目前为止，我们还没有写任何一行代码**，所有的数据采集和清洗，以及数据的向量生成，均可在 UI 界面通过提供的工具来完成。
> - **对于一个熟练能使用 ES 上各种工具的程序员来说，上面的过程整体花费可能不超过30分钟**。这使得我们能够更快速地进入到向量模型和大模型调试的阶段。

而在进行混合搜索的阶段，得益于 ES 将向量检索功能与原有的全文检索进行有效的整合，整个过程也几乎不需要花费太长时间。从代码上看，写一个函数就能进行测试：

```
from elasticsearch import Elasticsearch


# Search ElasticSearch index and return
def search(es, index_name, embedding_model, query_text):

    source_fields = ["body_content", "url", "title"]
    query = {
        "bool": {
            "must": [{
                "match": {
                    "title": {
                        "query": query_text,
                        "boost": 1
                    }
                }
            }]
        }
    }
    knn = [{
```

```
        "field": "ml.inference.headings_embeddings.predicted_value",
        "query_vector_builder": {
            "text_embedding": {
                "model_id": embedding_model,
                "model_text": query_text
            }
        },
        "k": 5,
        "num_candidates": 10,
        "boost": 24
    }, {
        "field": "ml.inference.body_content_embeddings.predicted_value",
        "query_vector_builder": {
            "text_embedding": {
                "model_id": embedding_model,
                "model_text": query_text
            }
        },
        "k": 5,
        "num_candidates": 10,
        "boost": 24
    },
    ]
    resp = es.search(
        index= index_name,
        fields=source_fields,
        query=query,
        knn=knn,
        size=3,
        source=False)

    return resp
```

关于以上代码，重点需要关注的地方在于：

1. 我们通常需要结合混合搜索的方式才能获得最佳效果。
2. 同时，在执行向量搜索的时候，我们可以在多字段上进行向量检索。
3. 而我们将 `embedding_model` 作为变量传入，使得我们可以在调试的过程中，选择不同的模型组合来查看语义检索的性能，看看哪些文档会被召回。
4. 因为我们需要将检索出来的内容交给大模型来理解，受限于模型 token的限制，我们通常只召回1~3篇文档。**这种场景下，精确率比召回率重要！**

通过语义搜索，对比一下原来的全文检索，同样是搜索 “如何销毁ls节点” 我们可以查到原先无法获得的信息：

```
es = Elasticsearch(hosts=["https://es-7cu6zx9m.public.tencentelasticsearch.com:9200"],
                   http_auth=(username, password))
print(search(es,"moka-ai__m3e-base","如何销毁ls节点？"))
>>>>>>>>>>>>>
{'took': 79, 'timed_out': False, '_shards': {'total': 2, 'successful': 2, 'skipped': 0, 'failed': 0}, 'hits':
{'total': {'value': 24, 'relation': 'eq'}, 'max_score': 47.651337,
'hits': [{'_index': 'search-tencent-es-doc', '_id': '6542334ead3b4de80cb57864', '_score': 47.651337,
'fields': {
'body_content': ['操作场景 当 Logstash 实例无法满足您的需求，需要退货时，您可以在 Elasticsearch Service 控制台对实例进行
销毁，以避免服务继续运行而产生费用。如果是实例配置无法满足需求，您也可以通过调整实例配置把实例调整到合适的规格，详情可参见 实例扩缩
容 。 不同计费模式退费说明 不同计费模式下的实例，销毁实例的条件如下： 预付费包年包月的实例，如果实例还未到期，需要提前销毁时，可参
见 包年包月退费 。 后付费按量计费的实例，根据使用量计费，可以随时销毁实例，销毁后，就不再产生费用。 注意 实例被销毁后，数据无法恢
复，请谨慎操作。 操作步骤 1. 登录 Elasticsearch Service 控制台 ，在左侧导航栏单击 Logstash 管理 ，进入 Logstash 实例列表
页。 \ufeff 2. 在实例列表页，选择需要销毁的实例，选择 操作 > 更多 > 销毁 进行销毁；或单击实例 ID/名称 进入实例基本信息页，选择
右上角 更多操作 > 销毁 进行销毁。 \ufeff 3. 在销毁实例页面中，单击 确定 ，系统将清空实例数据，并回收资源， 数据清空后，无法恢复
。包年包月的费用退还方式，可参见 包年包月退费 。'],
'title': ['Elasticsearch Service 销毁实例-Logstash 指南-文档中心-腾讯云'],
'url': ['https://cloud.tencent.com/document/product/845/55114']}},
```

如果是搜索 “无服务器ES”，也能找到相关的结果，而不会因为语言的原因无法识别 “Serverless” 和 “无服务器” 之间的关系。

使用大模型进行 RAG 的对话生成

但正如我们之前所说的，客户实际的需求可能会更复杂，客户的问题可能是：

- 无服务器 es 如何收费？请给我一个详细的例子。
- 如何销毁 ls 实例，给我具体的步骤。

我们需要把查出来的文档交给大模型进行指导性内容的生成，受限于大模型 token 的限制，我们不可能无限的把召回的内容交给大模型去理解和学习；同时，召回的不相关的文档，也可能会导致产生幻觉；而根据场景，设定良好的 prompt，也是大模型能够按照我们的期望为我们提供答案的关键。 因此，其实现如下：

```
def truncate_text(text, max_tokens):
    tokens = text.split()
    if len(tokens) <= max_tokens:
        return text
    return ' '.join(tokens[:max_tokens])
# Generate a response from ChatGPT based on the given prompt
def chat_gpt(prompt, model="gpt-3.5-turbo", max_tokens=1024, max_context_tokens=4000, safety_margin=5):
    # Truncate the prompt content to fit within the model's context length
    truncated_prompt = truncate_text(prompt, max_context_tokens - max_tokens - safety_margin)
    print(truncated_prompt)
    response = openai.ChatCompletion.create(engine="gpt-35-turbo",
                                        messages=[{"role": "system",
                                                   "content": "你现在是腾讯云Elasticsearch Service的专家."},
                                                  {"role": "user", "content": truncated_prompt}])
    return response["choices"][0]["message"]["content"]

negResponse = "根据检索出来的文档，我无法回答这个问题。"
resp = search(es,embedding_model,query)
retrival_result = []
for hit in resp['hits']['hits']:
    #避免产生幻觉
    if hit['score'] > 30:
        retrival_result.append(hit['fields']['body_content'][0])
# 组合召回的内容，也可以只选择 top 1
combine_result = ";".join(retrival_result)
print(combine_result)
prompt = f"回答此问题：{query}\n 如果所提供的文档中没有答案，请回复： '{negResponse}'\n 回答时，参考来自腾讯云ES的帮助文
档：{combine_result} "
answer = chat_gpt(prompt)
```

> **注意：**
>
> 关于以上代码，重点需要关注的地方如下：
>
> - 大模型的限制，所以定义了 `truncate_text()` 函数与 `max_context_tokens` 等参数，避免因超过 token 而调用失败。
> - 限制交给大模型的上下文信息。通过 `if hit['score']>30` 避免不相关的文档误导大模型的理解。
> - 设置合理的 Prompt，让大模型理解自己的角色，以及角色该如何处理问题：
>   - `"role":"system","content":"你现在是腾讯云 Elasticsearch Service 的专家."`
>   - `role":"user","content":""回答此问题：{query}\n 如果所提供的文档中没有答案，请回复： '{negResponse}'\n 回答时，参考来自腾讯云 ES 的帮助文档：{combine_result} "`

简单的运行上面的代码，我们可以得到如下结果：

```python
query = "无服务器es如何收费?给我一个详细的例子"

resp = search(es,"moka-ai__m3e-base",query)

retrival_results = []
for hit in resp['hits']['hits']:
    #避免产生幻觉
    if hit['_score'] > 30:
        retrival_results.append(hit['fields']['body_content'][0])
# 组合召回的内容，也可以只选择 top 1
combine_result = ";".join(retrival_results)


prompt = f"回答此问题: {query}\n 如果所提供的文档中没有答案，请回复: '{negResponse}'\n 回答时，参考来自腾讯云ES的帮助文档: {comb
answer = chat_gpt(prompt)
print(answer)
```

PROBLEMS 6 OUTPUT DEBUG CONSOLE PORTS 6 TERMINAL

```
(ElasticDocs_GPT) [root@VM-0-10-centos ElasticDocs_GPT]# /root/ElasticDocs_GPT/bin/python /root/test.py
ES Serverless是一种无服务器的Elasticsearch服务，目前仅支持按量计费模式，其费用由计算流量、数据存储和接口调用三个方面组成
。

以广州地域的ES Serverless服务为例，假设一个应用的平均访问日志为100字节/条，每天1亿条日志，数据存储时间为7天，且对所有字段
构建索引。则其费用如下：

计算流量：
索引字段占比为100%，则计算流量大小为（0.1+100%）*9.3=10.23GB，价格为0.29元/GB，费用为2.97元。

数据存储：
索引字段占比为100%，则数据存储大小为（0.12+100%）*9.3*7=72.9GB，单位价格为0.01元/GB/天，费用为0.729元。

接口调用：
bulk size为5,000条/次，1亿条文档/bulk size=20,000次，单位价格为0.1元/百万次，费用为0.002元。

因此，该应用每天的费用约为3.7元。

需要注意的是，计算流量和数据存储的大小与数据中的字段以及字段长度有关。并且，索引字段占比是估算构建索引产生的流量与存储消
耗的比例，可能与实际情况不完全一致。详细的价格计算可以通过腾讯云的价格计算器进行估算。
(ElasticDocs_GPT) [root@VM-0-10-centos ElasticDocs_GPT]#
```

上图中，我们可通过提问 “无服务器 es 如何收费？给我一个详细的例子”，我们得到了一个详细的，由大模型生成的例子。成功实现 RAG！ 结合 streamList 等简单的 Python 框架，我们可以非常轻松的构建出一个智能助手程序：

总结

在本文中，我们介绍了如何通过腾讯云 ES 与 ChatGPT 快速搭建一个检索增强的生成式人工智能助手，通过腾讯云 ES 的数据采集、数据清洗、数据分析等功能，以及向量生成、向量存储、向量检索等向量库的功能，再结合原有的全文检索能力，我们可以快速实现与大模型的结合，只需区区百行代码即可完成一个带用户界面的人工智能助手。

不仅仅是玩票性质的应用开发，对于规模更大的企业级生产环境来说，腾讯云 ES 能够带来更高的效能和更稳定的表现。多年的历程使得腾讯云 ES 经过了千锤百炼的考验，成为一个值得信赖的平台。

在未来，我们将继续优化腾讯云 ES，以满足企业级生产环境的更多需求，并为用户提供更好的使用体验。我们将不断努力，为人工智能助手的开发和应用带来更多创新和便利。

如果您对相关能力感兴趣，欢迎扫描下方二维码加入腾讯云 ES AI 增强与向量检索交流群！

让企业知识触手可及，乐享基于腾讯云 ES 的 RAG 应用实践

最近更新时间：2025-01-07 11:44:12

腾讯乐享利用腾讯云 ES 的一站式文本检索和向量检索混合搜索能力，打造面向企业级知识管理的智能问答模块。通过精准的信息检索、严格的权限管理和全流程的知识管理，帮助企业内部员工和外部伙伴高效挖掘企业和组织的知识金矿。
本文介绍了乐享基于腾讯云 ES 的一站式 RAG 解决方案实现乐享智能问答功能的实践。

腾讯乐享的业务背景和挑战

腾讯乐享是一款集知识管理、企业培训、文化建设于一体的智能化组织学习协作平台，拥有知识库、问答、课堂、考试等功能，旨在帮助企业建立高效灵活的知识共享和流动机制，提升组织能力。
随着大模型技术的发展，乐享看到了前所未有的机会来提升信息获取的效率和准确性，来推动业务模式和服务方式的创新。
然而，乐享也面临着新的挑战，尤其是解决大模型"幻觉"问题，确保数据安全和隐私保护的情况下保证千人千面的回答效果。企业的"私有知识"不能直接用于大模型的预训练和精调，这也对知识管理提出了更高的要求。
如何用大模型这个"新瓶"，来装企业知识管理的"旧酒"，左手客户价值最大化，右手产品体验商业化，便成了乐享团队必须要解决的课题。

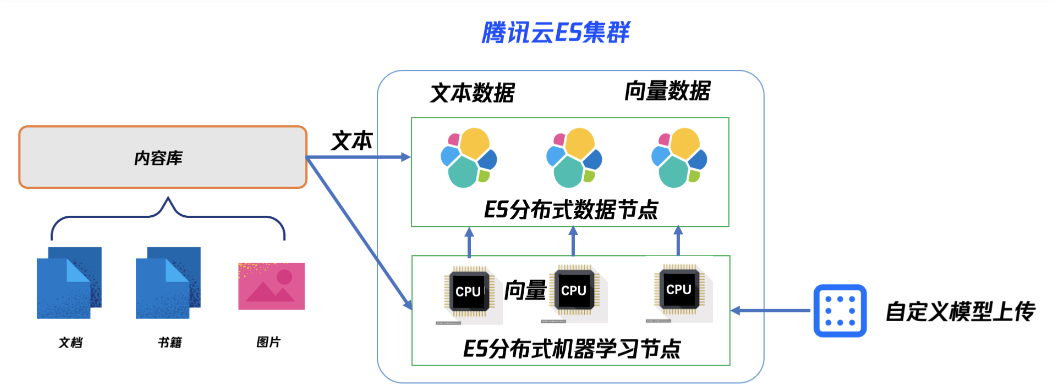
什么是 RAG?

随着搜索引擎技术的发展，传统的关键词搜索已无法满足用户需求。RAG（Retrieval-Augmented Generation，检索增强生成）结合向量检索和文本生成，通过检索模型查找相关文档，将其作为生成模型的上下文信息生成准确的答案。这种方法将外部知识与生成模型结合，提升了内容的流畅性和准确性，广泛应用于知识问答、文本摘要和对话生成等场景。

腾讯云 ES 一站式 RAG 解决方案

一站式向量检索能力

基于腾讯云 ES 提供的能力，我们可以直接完成**向量生成** > **向量索引/存储** > **向量检索**的全流程操作，具体能力如下：
- 内置开箱即用的 ELSER 语义模型。
- 支持上传自定义模型或从 Hugging Face 等开源社区直接上传模型。
- 支持专有机器学习节点，用于向量推理（可选，也可跟数据节点混部）。
- 支持向量索引（HNSW 算法）、向量存储与 KNN 检索。



独有混合搜索能力

向量搜索尽管可以实现语义查询，但在处理特定字符串如"8XLARGE64"时，可能召回的结果不相关。结合向量搜索和文本搜索的优势，混合搜索通过同时执行文本和向量搜索并融合结果，提升召回率。腾讯云 ES 支持一站式混合搜索，能够在单次查询中同时执行文本和向量搜索，并自动对结果进行排序融合。

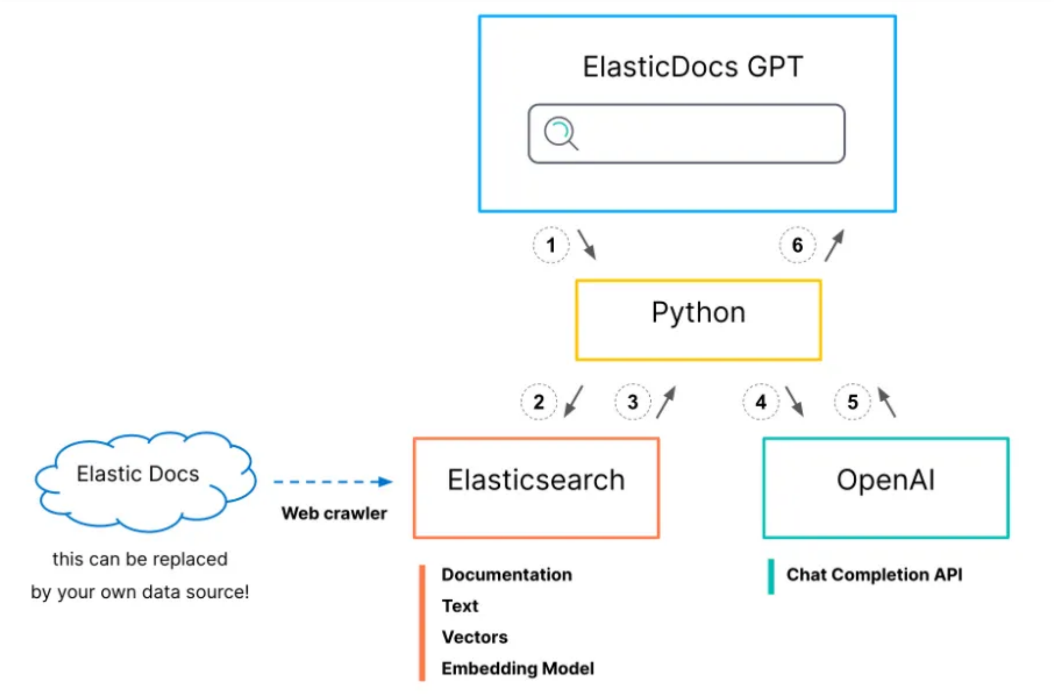
倒数排序融合（RRF）

在多路召回中，系统评分机制不同，需要归一化评分来公平比较和融合。腾讯云 ES 内置的倒数排序融合（RRF）算法通过对排名赋予权重，计算各系统排名的倒数之和，生成最终融合排名列表。RRF 的优点包括：

- **简单性：**无需复杂归一化，只需知道每个文档在各系统中的排名。
- **鲁棒性：**对不同评分尺度和分布的敏感性较低。
- **公平性：**通过相同公式赋予权重，确保融合过程公平性。
- **易于实现：**实现相对简单，无需复杂参数调整或训练过程。
- **适应性：**可以适应新的系统或数据，不依赖特定评分机制。

与 LLM 大模型的集成

腾讯云 ES 可与 Langchain 等第三方管道以及 LLM 大模型等无缝集成，帮助构建生成式应用程序。

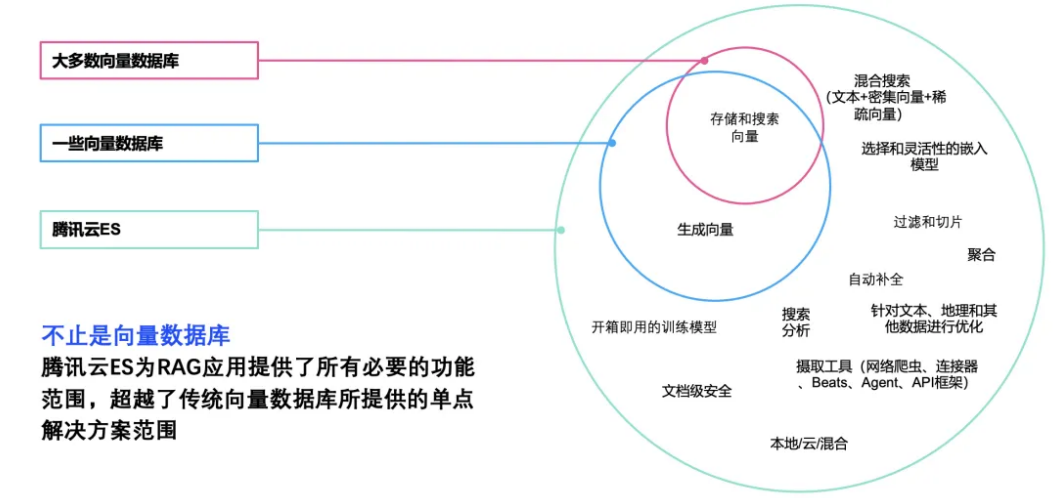


腾讯云 ES RAG 方案优势

腾讯云 ES 提供的一站式 RAG 解决方案，提供了 RAG 应用范围内的所有功能，超越了传统向量数据库所提供的单点解决方案范围，具体如下：

- 支持直接上传自定义 embedding 模型，或者 hugging face 上的 embedding 模型，在 ES 内部完成模型管理、部署、数据向量化等。

- 支持通过文本 + 向量的混合搜索能力，灵活应对各种需求。
- 支持通过设置文本、向量的权重，甚至字段级别的词权重，进行打分排序，同时也可以直接使用内置的 RRF 能力，直接实现排序融合，后续也支持通过引入 LTR 模型，对粗排结果进行重排，满足不同业务场景的需求。
- 在某些权限敏感的场景，ES 支持基于角色设置文档设置字段级别的权限管控，精准进行权限控制。



除此之外，RAG 应用一般服务的是在线业务场景，稳定性也是极为重要的一环，腾讯云 ES 支持多副本设置、多可用区部署、CCR 跨集群复制、自动数据备份、弹性伸缩等能力，切实保障在线业务的稳定性。同时，通过智能巡检、多维立体监控告警、审计日志等能力，帮助企业降低运维与管理成本，实现降本增效。

腾讯云 ES 高精度高性能自研内核

腾讯云 ES 不仅提供了相对于传统向量数据库的更全面的 RAG 解决方案，同时在性能方面也有着独特的优势。 腾讯云 ES RAG 方案凭借紫霄 GPU 的强大性能和深度优化的自研内核，大幅提升了向量检索的效率和精度。

自研内核优化

腾讯云 ES 在内核层面，针对典型向量场景特点做了深度优化，例如分片架构优化，查询并行化，lucene 查询缓存锁改造等，10亿级向量检索平均响应延迟控制在毫秒级，整体查询性能提升3 - 10倍：

- **混合搜索：**基于向量检索与 BM25 算法的实时联合检索和多路召回，并对结果进行融合，大幅提升结果的召回率，同时保持了较高的性能。
- **向量的标量量化：**将浮点数 float8 转为 int8 型表示，在保证召回率不受影响的前提下，大幅降低了 CPU、堆内存资源消耗，并降低了搜索的延迟。
- **自适应副本策略：**ES 分布式系统网络调用多，如果遇到跨地区/AZ 调用或者慢节点分片副本调用，就会导致查询延迟高。腾讯云 ES 通过改进的本地自适应副本选择策略，其核心原理是：通过计算协调节点跟数据分片副本的查询平均响应时间、查询队列、查询成功率，不断地调整选择延迟最低的副本进行查询，来大幅降低搜索的延迟，并保障负载均衡。
- **查询裁剪：**ES 查询模型是将查询请求拆分成分片级的子请求转发给各个分片并行执行，最后在协调节点合并各个分片的结果，在每个分片内部有多个 segment。腾讯云 ES 通过对列存、数值索引、Terms 等维度对 segment 进行提前裁剪跟合并收敛，减少随机 IO，优化查询性能。
- **查询并行化：**通过压榨空闲 CPU 资源，将 ES 的单个分片级请求拆分成3 - 5个子请求并行处理该分片下的 segment 或者 docs，根据 docs 或者 segment 切分，每个线程只处理一部分 docs 或者 segment，在数据节点合并每个线程的结果后再返回给协调节点，协调节点合并各个分片的结果返回给客户端，从而达到性能倍数级的提升。
- **查询缓存优化：**通过 CBO 策略，避免了查询 Cache 操作导致查询耗时10+倍的毛刺，并通过最小粒度的读写锁提升了 LRU 缓存性能2倍+，并提交到官方 Elasticsearch、Lucene 社区，得到社区的认可跟点赞。

紫霄 GPU 支持

腾讯云 ES 是全球唯一支持 GPU 的 ES 服务，它可以与腾讯自研 “芯” 技术紫霄软硬结合，充分利用 GPU 的性能优势，提高向量生成和检索的效率。紫霄 V1 具有高能效、高吞吐、高带宽等特点，设计算力规格 NVIDIA A10 相当，显存带宽比 A10 高30%，最高可比 A10 性能高50% - 100%。 在常见中小模型上，紫霄相比 NVIDIA T4 通常有100%以上的性能提升，相比 NVIDIA A10 有20%+的性能优势。

腾讯云 ES 与 Milvus 性能对比

我们基于开源工具 ann-benchmark 进行了详细性能测试，通过检索不同维度的数据集在召回率达到99%的情况下，获取最相似的 Top10 的文档，对比 Milvus、与腾讯云 ES 的向量检索的 QPS 数据。

公共数据集	维度	配置	召回率	QPS	
				Milvus	腾讯云 ES

sift-128-euclidean	128	3节点4C8G，HNSW（m=16, efConstruction=200）	99%	3653	19479
gist-960-euclidean	960			480	4136

可以看到，无论是在高维还是低维的向量检索场景，腾讯云 ES 有接近5 - 8倍的性能领先优势，用更低的成本、更好的性能帮助业务发展，降本增效。

乐享基于腾讯云 ES 的 RAG 实践

乐享智能搜索模块的核心目标是帮助企业内部员工和外部伙伴高效挖掘企业沉淀的知识金矿。虽然智能搜索的交互并不复杂，但要把这个搜索结果做好，并没有那么容易。首先要解决的是大模型“幻觉”问题，确切的说把“幻觉”降到最低。对企业客户来说，诚实地说“我不知道”，比起一个明显错误的答案，前者的影响范围要小的多。并且这些企业的“私有知识”，显然是不能直接用在大模型预训练（Pre Trained）和精调（Fine Tuning）中的。针对这类问题，乐享采用的方案是 RAG（Retrieval Augmented Generation，索引增强生成）方案。

在实现 RAG 的过程中，乐享采用了腾讯云 ES 提供的向量检索跟精确文本匹配的混合搜索，结合其内置的倒数排序融合（RRF）算法，完成了多路召回、混合搜索和混合排序，最终实现高召回率、高准确率的数据检索。

实践结果

通过实施腾讯云 ES 的 RAG 解决方案，乐享智能搜索模块实现了以下成果：

- **高召回率和准确率：**通过混合搜索技术，乐享有效提升了向量检索的召回率和准确性，确保用户查询到的结果更加相关和精确。
- **用户满意度提升：**智能搜索模块减少了大模型的“幻觉”现象，提高了回答的准确性和可靠性，用户满意度显著提升。
- **业务效率提高：**企业内部员工和外部伙伴可以更快速地获取所需知识，提升了工作效率和协作效果。
- **数据安全保障：**通过严格的权限管理，乐享确保了企业数据的安全性和隐私保护，增强了用户对系统的信任。

总结

腾讯云 ES 的一站式 RAG 解决方案在智能检索领域表现卓越，帮助企业降本增效，实现业务价值持续增长。
乐享通过大模型技术，将企业多年积累的数据转化为有价值的知识，助力企业构建智能学习社区。
未来，腾讯云 ES 将继续深耕智能检索领域，在成本、性能、稳定性等方面持续提升，帮助客户实现更高的业务价值。感谢您的支持，欢迎持续关注腾讯云 ES 与乐享！

基于腾讯云 ES 混合搜索与 DeepSeek，十分钟构建专属 RAG 应用

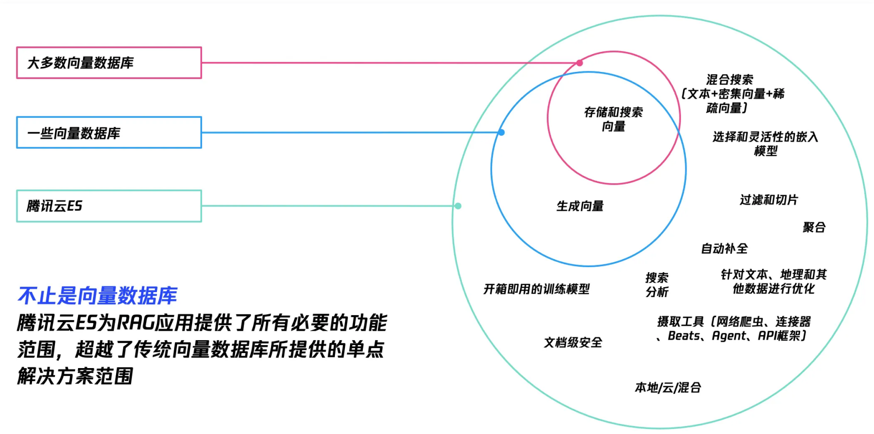
最近更新时间：2025-01-22 11:40:22

背景概述

随着数据智能技术的不断发展，以大语言模型（LLM）驱动的 AIGC 为代表的内容生成技术已经成为企业数据智能能力中不可或缺的一部分，但传统的内容生成技术存在信息更新不及时、垂直领域知识匮乏、模型幻觉等问题，如何推进大模型在各行业、各业务场景落地是各方普遍关注的问题，而检索增强生成（Retrieval-Augmented Generation，RAG）技术则为此提供了有效的解决方案，成为数据智能时代的一大趋势。
RAG 是一种结合了检索和大语言模型内容生成的技术方案，它通过引用外部知识库，在用户输入 Query 时检索出知识，然后让模型基于可信的知识进行用户回答。RAG 具有较高的可解释性和定制能力，可大幅降低大语言模型的幻觉，适用于问答系统、文档生成、智能助手等多种自然语言处理任务。本文将通过介绍腾讯云ES 一站式 RAG 方案，演示如何通过结合腾讯云 ES 与 DeepSeek 大模型，快速构建 RAG 应用。

腾讯云 ES 一站式 RAG 方案

腾讯云 ES 是云端全托管海量数据检索分析服务，拥有高性能自研内核，集成 X-Pack，支持通过自治索引、存算分离、集群巡检等特性轻松管理集群，也支持免运维、自动弹性、按需使用的 Serverless 模式。在自研内核方面，腾讯云 ES 依托腾讯内外部海量业务的运营经验，针对 ES 内核进行了成本、性能、稳定性、拓展性等方面的优化，是全球第三方企业开源贡献第一的团队，使用腾讯云 ES 可以高效构建在线搜索、向量检索、日志分析、运维监控、智能问答等服务。
在 RAG 方面，腾讯云 ES 支持了一站式向量检索、文本 + 向量混合搜索、倒数排序融合、与大模型集成、GPU 高性能推理、字段级别权限控制等能力，同时针对查询性能做了大量优化，有效的提升了数据检索效率，目前已落地微信读书“AI 问书”、乐享智能搜索等大型应用中。



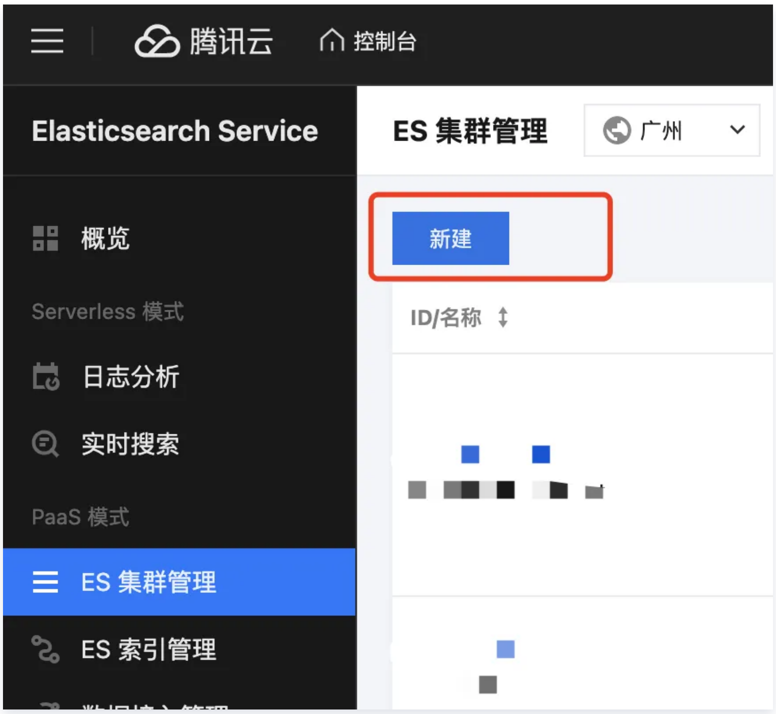
同时，作为国内公有云首个从自然语言处理、到向量生成/存储/检索、并与大模型集成的端到端一站式技术平台，腾讯云 ES 作为核心参编单位参与了由中国信通院发起的的 RAG 标准制定，并成为首个通过 RAG 权威认证的企业。

知识库构建能力	知识检索能力	内容生成能力	质量评估能力	平台能力
数据读取	查询优化	Prompt优化	评估指标	用户管理
数据预处理	检索能力	大模型生成	评估方法	数据权限管理
内容增强	检索结果优化	生成内容优化	评估数据集	模型权限管理
索引构建				资源及任务管理
知识库管理				日志管理

AI 助手构建

购买 ES 集群

1. 登录 腾讯云 ES。
2. 进入 ES 集群管理页面，单击**新建**。

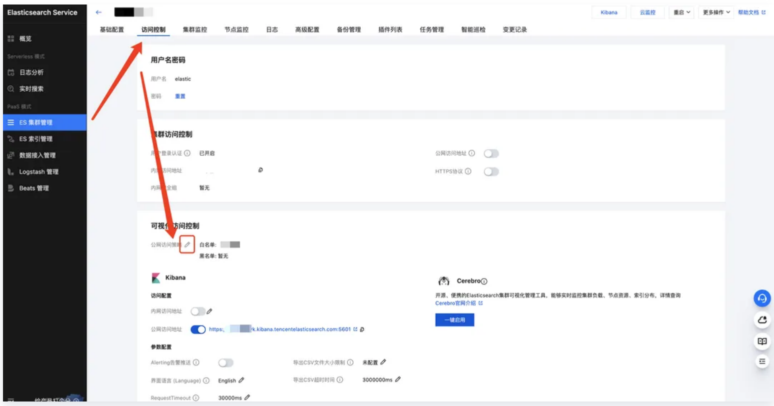


3. 计费模式为按量计费，产品版本为标准版、ES 版本为 8.13.3、商业特性为白金版。

4. ES 节点配置，测试环境可选择为 ES.S1(2核4G)，节点数为2，磁盘为通用型 SSD，磁盘容量为20GB。

登录 Kibana

1. 在集群列表单击对应集群名称，进入详情页后，单击**访问控制**，设置公网访问策略。



2. 获取当前 IP 地址并设置到 IP 白名单中。

3. 单击 Kibana 公网访问地址访问 Kibana。

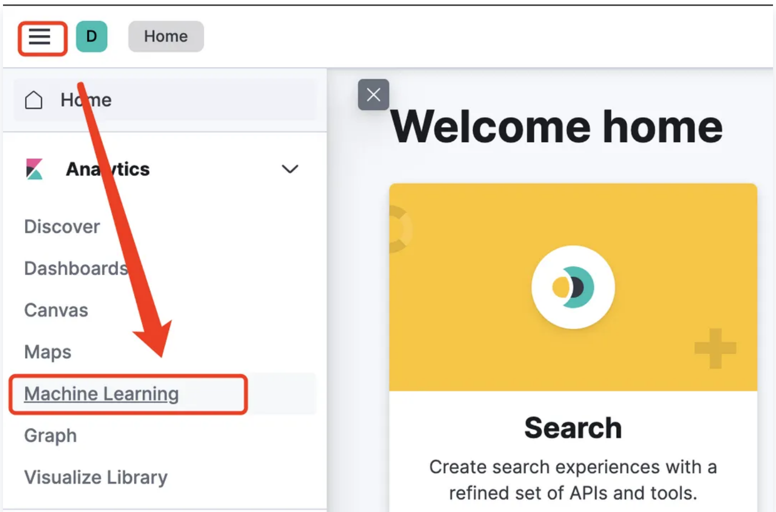
部署 embedding 模型

集群购买完成后，前往 Kibana 部署 Embedding 模型、创建知识库索引与向量化管道。

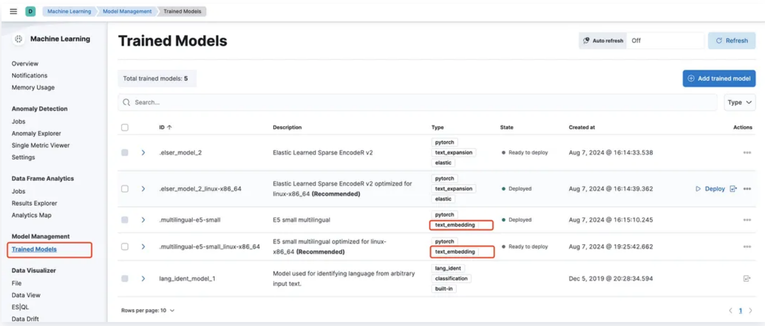
1. 开启**节点出站访问**，仅开启数据节点即可，如有专用主节点，仅开启专用主节点即可。（该功能为白名单，请联系 工单 处理）

> **注意：**
> 如需上传自定义模型或第三方平台（例如 Huggingface）模型，可参考 GitHub – elastic/eland: Python Client and Toolkit for DataFrames, Big Data, Machine Learning and ETL in Elasticsearch 。

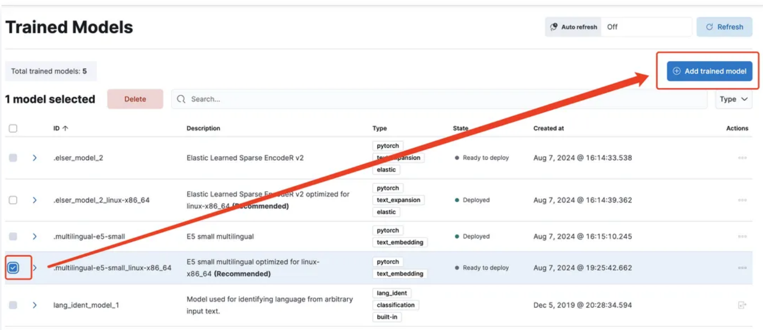
2. 登录 Kibana 后，在左侧导航栏找到 **Machine Learning** 功能。

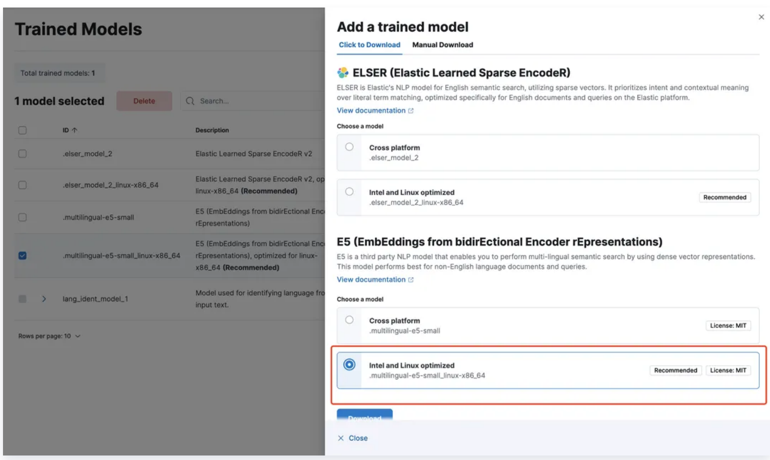


3. 进入模型管理页面，并找到类型为 **text_embedding** 的模型。

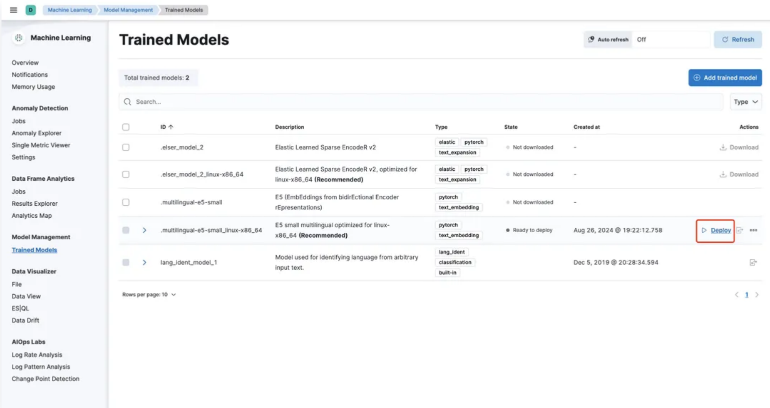


4. 如为未下载状态，选中模型，并单击 **Add trained model**，本次演示我们使用 **.multilingual-e5-small_linux-x86_64** 模型，这是一个384维的多语言模型：

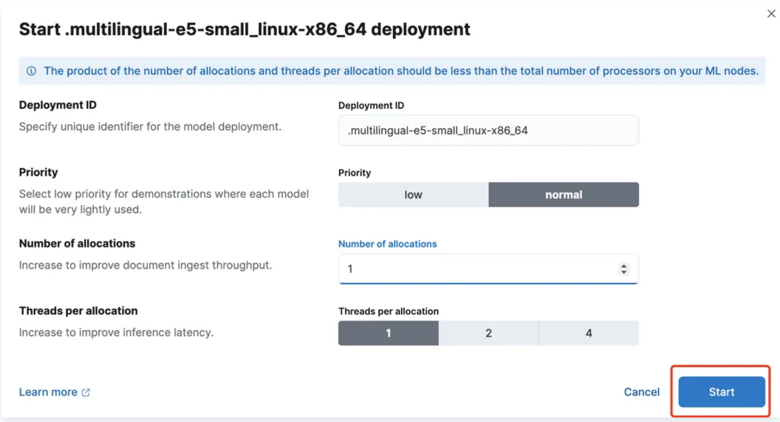




5. 下载完成后，单击部署。

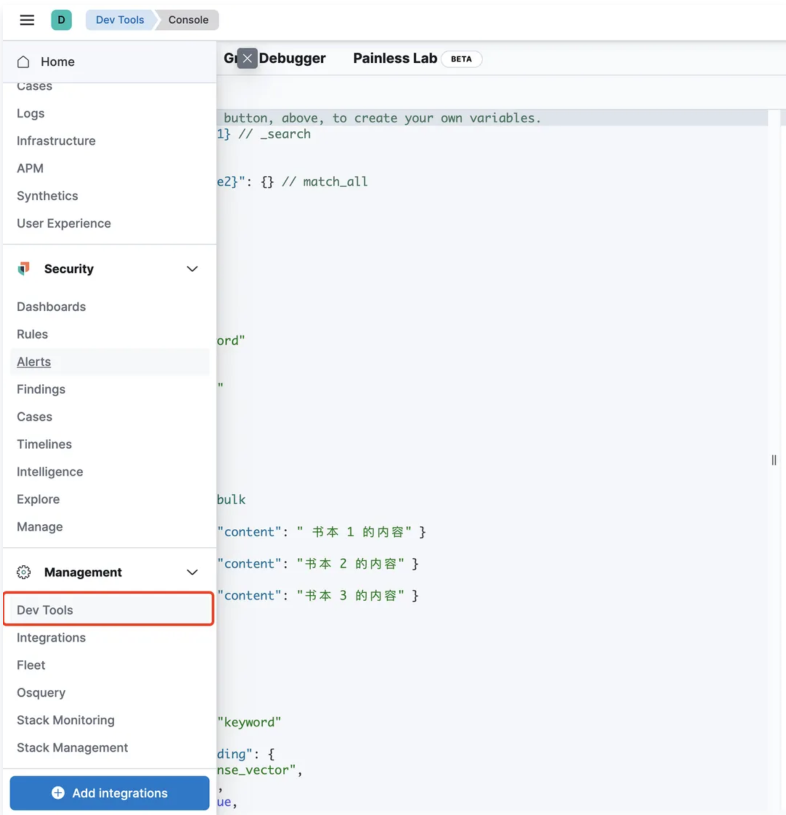


6. 为快速体验，配置使用默认值即可。



创建索引与向量化管道

1. 单击进入 Dev tools。



2. 创建知识库索引：index-name 为索引名称，实际可按需命名。

```
PUT /index-name
{
  "mappings": {
    "properties": {
    "title": {
        "type": "keyword"
      },
      "content": {
        "type": "text"
      },
      "url": {
        "type": "keyword"
      },
      "content_embedding": {
        "type": "dense_vector",
        "dims": 384
      }
    }
  }
}
```

3. 创建推理管道，该管道可用于写入数据时进行数据向量化。

```
PUT /_ingest/pipeline/index-name-pipeline
{
  "description": "Text embedding pipeline",
  "processors": [
    {
      "inference": {
        "model_id": ".multilingual-e5-small_linux-x86_64",
        "input_output": [
          {
            "input_field": "content",
            "output_field": "content_embedding"
          },
          {
            "input_field": "title",
            "output_field": "title_embedding"

          }
        ]
      }
    }
  ]
}
```

上述管道，将字段 content、title 的内容，调用 .multilingual-e5-small_linux-x86_64 模型向量化之后存储到新的字段中。

写入知识库数据

通过 Bulk API 批量写入数据，可将 title 、content 、url 的内容替换为实际的知识库数据。

```
POST index-name/_bulk?pipeline=index-name-pipeline&refresh
{ "index" : {} }
{ "title" : "标题 1","content": "内容 1","url": "https: url1"  }
{ "index" : {} }
{ "title" : "标题 2","content": "内容 2","url": "https: url2"  }
```

调用 DeepSeek 大模型

1. python 文件命名为 deepseek.py。

2. 获取 DeepSeek，当前可免费使用500万 tokens。

3. deepseek.py 内容如下：

```
import json
import requests
from typing import Dict, Any

class DeepSeekAPI:
    def __init__(self, api_key: str):
        self.api_key = api_key
        self.base_url = " https://api.deepseek.com/v1"
        self.headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json"
        }

    def chat_completion(self, system_prompt: str, user_message: str) -> Dict[str, Any]:
        url = f"{self.base_url}/chat/completions"
        payload = {
            "model": "deepseek-chat",
            "messages": [
                {"role": "system", "content": system_prompt},
                {"role": "user", "content": user_message}
            ],
            "temperature": 0.7,
            "max_tokens": 2000
        }

        try:
            response = requests.post(
                url,
                headers=self.headers,
                json=payload,
                timeout=30
            )
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            return {"error": str(e)}

def deepseek_chat(system_prompt: str, content: str):
    api_key = "输入申请的 api_key"
    deepseek = DeepSeekAPI(api_key)
    return deepseek.chat_completion(system_prompt, content)
```


智能问答系统构建

1. 安装 streamlit。

```
pip install streamlit
```

2. 获取 ES 访问地址：

   用户名为 elastic、密码在创建集群时设置，用本地 mac 测试时，可开启公网访问，实际生产时，建议使用内网访问地址。

3. 运行如下代码（ 可命名为 web.py，需与 deepseek.py 在一个目录下 ）

```
import streamlit as st
from elasticsearch import Elasticsearch
from deepseek import deepseek_chat

es_client = Elasticsearch(
    "ES集群访问地址",basic_auth=("用户名", "密码"))

def get_elasticsearch_results(query):
    es_query = {
        "knn": {
            "field": "content_embedding",
            "num_candidates": 100,
            "query_vector_builder": {
                "text_embedding": {
                    "model_id": ".multilingual-e5-small_linux-x86_64",
                    "model_text": query
                }
            }
        },
        "query":{
            "match":{
                "content":query
            }
        },
        "rank":{
            "rrf":{
                "window_size":100,
                "rank_constant":20
            }
        }
    }
    result = es_client.search(index="index-name", body=es_query)
    return result["hits"]["hits"]

def create_deepseek_prompt(results,question):
    context = ""
    for hit in results:
        source_field = hit["_source"]["content"]
        hit_context = source_field
        context += f"{hit_context}\n"
    prompt = f"""
```

```
Instructions:

回答此问题：{question}
回答时，你只能参考文档{context} 生成答案，如果无法生成答案，请回复：对不起，该问题我无法回答

"""
    return prompt

def generate_deepseek_completion(user_prompt, question):
    response = deepseek_chat(user_prompt, question)
    if 'error' in response:
        return f"Error: {response['error']}"
    return response['choices'][0]['message']['content']

def format_result(hit):
    title = hit["_source"]["title"]
    return f'[{title}]'

def main():
    st.title("我的专属AI助手")

    # 创建输入框和查询按钮
    question = st.text_input("请输入您的问题：")
    if st.button("查询并生成答案"):
        elasticsearch_results = get_elasticsearch_results(question)
        user_prompt = create_deepseek_prompt(elasticsearch_results,question)
        system_prompt = f"""
你是一个问答任务的助手。使用呈现的上下文真实、实事求是地回答问题。
"""

        openai_completion = generate_deepseek_completion(system_prompt, user_prompt)

        # 显示结果
        st.write(openai_completion)

        # 展示Elasticsearch查询结果
        st.write("大模型参考的文档：")
        for hit in elasticsearch_results:
            st.markdown(format_result(hit))

if __name__ == "__main__":
    main()
```

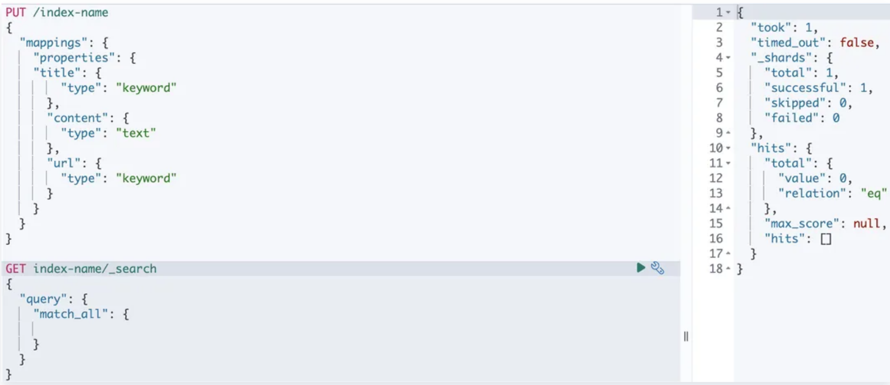
4. 在上述 python 文件的目录下，使用如下命令运行系统：

```
streamlit run web.py
```

5. 生成的界面如下：

私域数据问答测试

1. 索引中无数据。



2. AI 助手无法回答。

3. 写入相关数据。

```
POST index-name/_bulk?pipeline=index-name-pipeline&refresh
{ "index" : {} }
{ "title" : "腾讯云ES的产品经理","content": "腾讯云ES的产品经理是Kevin","url": "https: url1"  }
```

4. 回答如下:

总结

本文通过介绍腾讯云 ES 一站式 RAG 方案，演示如何通过结合腾讯云 ES 与 DeepSeek 大模型，快速构建 RAG 应用。腾讯云 ES 凭借其在传统 PB 级日志和海量搜索场景中积累的丰富经验，通过深度重构底层系统，成功地将多年的性能优化、索引构建和运营管理经验应用于 RAG 领域，并积极探索向量召回与传统搜索技术的融合之道，旨在充分发挥两者的优势，为用户提供更加精准、高效的搜索体验。未来，腾讯云 ES 将持续深耕智能检索领域，在成本、性能、稳定性等方面持续提升，帮助客户降本增效的同时实现业务价值持续增长，欢迎持续关注！

RAG 实践：基于腾讯云 ES 与混元，十分钟构建你的专属 AI 助手

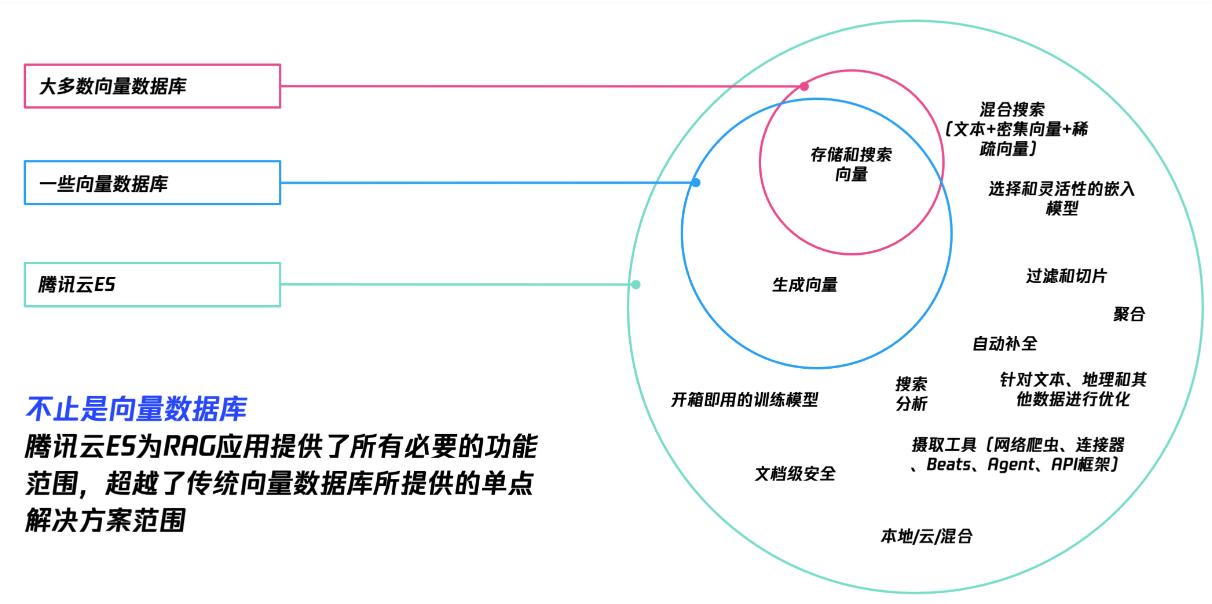
最近更新时间：2024-10-21 09:06:32

背景概述

随着数据智能技术的不断发展，以大语言模型（LLM）驱动的 AIGC 为代表的内容生成技术已经成为企业数据智能能力中不可或缺的一部分，但传统的内容生成技术存在信息更新不及时、垂直领域知识匮乏、模型幻觉等问题，如何推进大模型在各行业、各业务场景落地是各方普遍关注的问题，而检索增强生成（Retrieval-Augmented Generation，RAG）技术则为此提供了有效的解决方案，成为数据智能时代的一大趋势。
RAG 是一种结合了检索和大语言模型内容生成的技术方案，它通过引用外部知识库，在用户输入 Query 时检索出知识，然后让模型基于可信的知识进行用户回答。RAG 具有较高的可解释性和定制能力，可大幅降低大语言模型的幻觉，适用于问答系统、文档生成、智能助手等多种自然语言处理任务中。本文将通过介绍腾讯云 ES 一站式 RAG 方案，演示如何通过结合腾讯云 ES 与混元大模型，快速构建 RAG 应用。

腾讯云 ES 一站式 RAG 方案

腾讯云 ES 是云端全托管海量数据检索分析服务，拥有高性能自研内核，集成 X-Pack，支持通过自治索引、存算分离、集群巡检等特性轻松管理集群，也支持免运维、自动弹性、按需使用的 Serverless 模式。在自研内核方面，腾讯云 ES 依托腾讯内外部海量业务的运营经验，针对 ES 内核进行了成本、性能、稳定性、拓展性等方面的优化，是亚太地区开源贡献第一的团队，使用腾讯云 ES 您可以高效构建在线搜索、向量检索、日志分析、运维监控、智能问答等服务。
在 RAG 方面，腾讯云 ES 支持了一站式向量检索、文本 + 向量混合搜索、倒数排序融合、与大模型集成、GPU 高性能推理、字段级别权限控制等能力，同时针对查询性能做了大量优化，有效的提升了数据检索效率。



同时，作为国内公有云首个从自然语言处理、到向量生成/存储/检索、并与大模型集成的端到端一站式技术平台，腾讯云 ES 也作为核心参编单位参与了由信通院组织的 RAG 标准制定，同时成为首个通过 RAG 权威认证的企业。

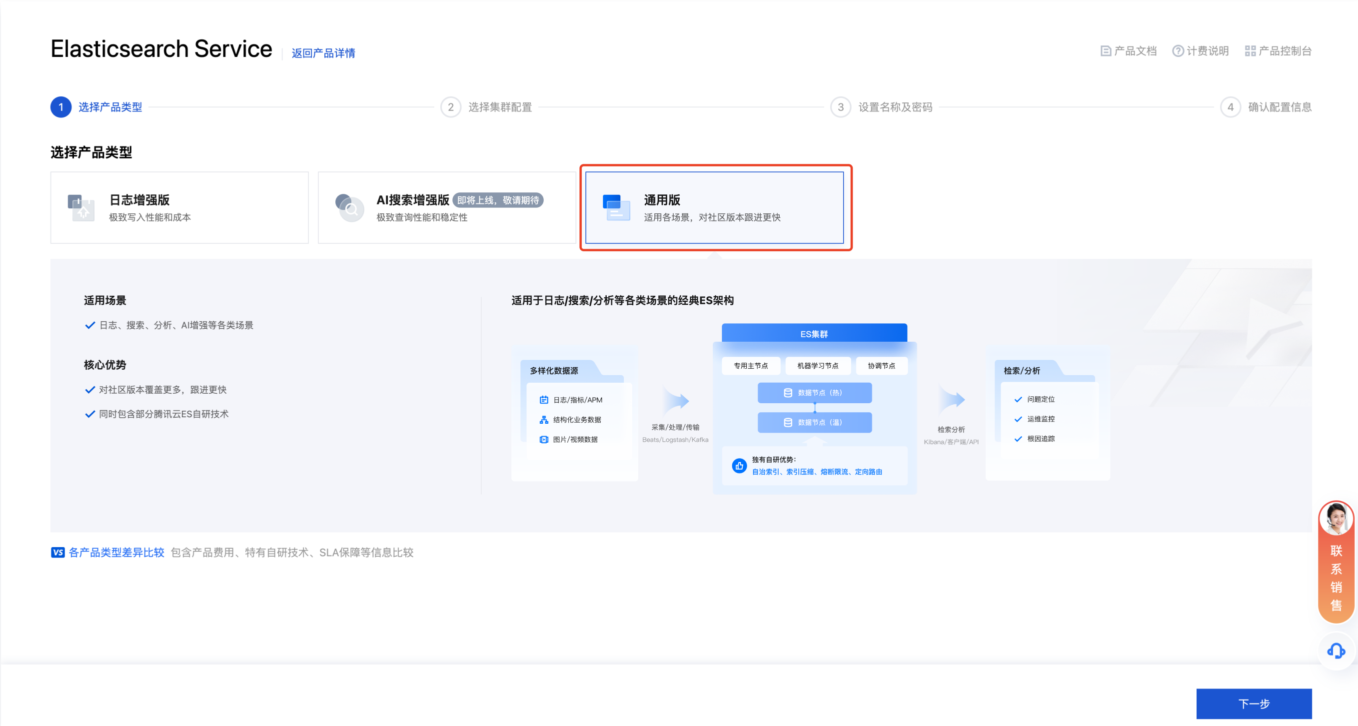
知识库构建能力	知识检索能力	内容生成能力	质量评估能力	平台能力
数据读取	查询优化	Prompt优化	评估指标	用户管理
数据预处理	检索能力	大模型生成	评估方法	数据权限管理
内容增强	检索结果优化	生成内容优化	评估数据集	模型权限管理
索引构建				资源及任务管理
知识库管理				日志管理

AI 助手构建

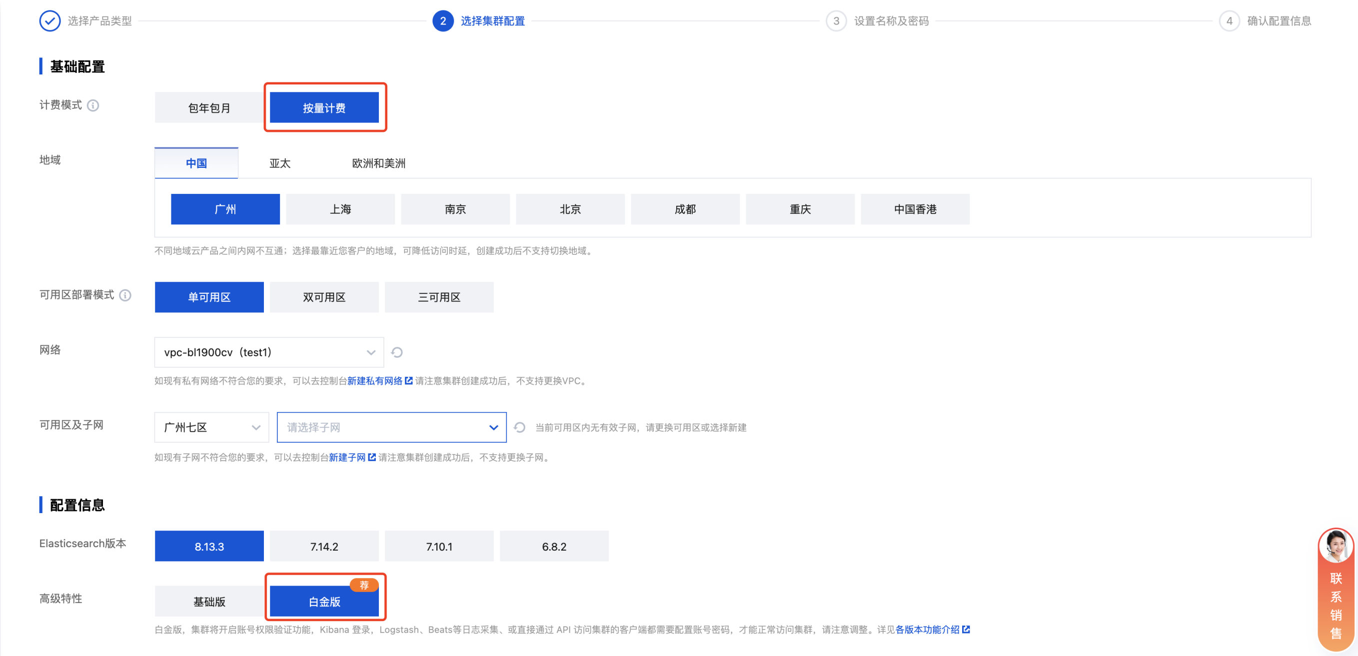
购买 ES 集群

1. 登录腾讯云 ES 控制台，进入 ES 集群管理 页面，单击**新建**。

2. 进入新建页面，产品类型选择为：**通用版**。



3. 选择版本：计费模式为**按量计费**、ES 版本为 **8.13.3**、商业特性为**白金版**。

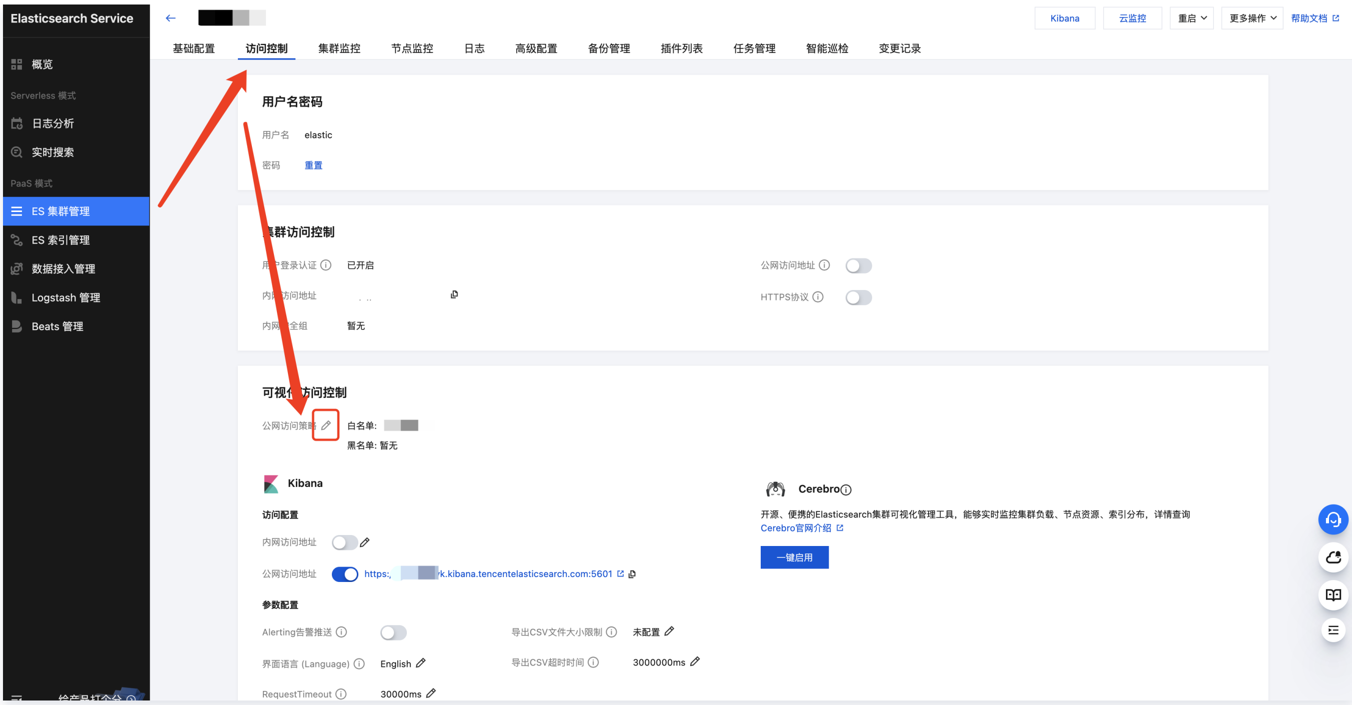


4. ES 节点配置，测试环境可选择为 **ES.S1(2核4G)**，节点数为**2**，磁盘为**通用型 SSD**，磁盘容量为**20GB**。

5. 其余配置选择默认即可。

登录 Kibana

1. 访问 Kibana，设置公网访问策略。



2. 单击 Kibana 公网访问地址访问 Kibana。

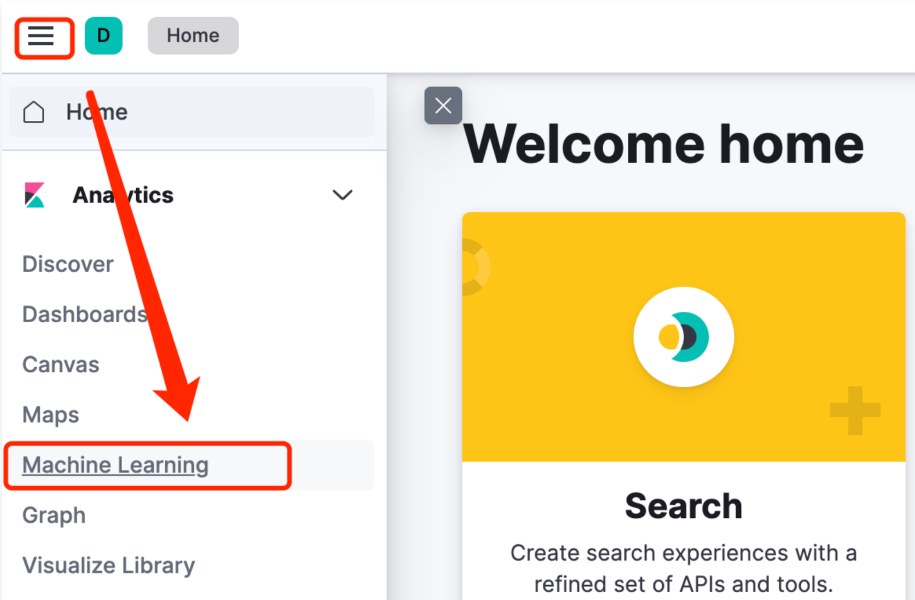
部署 embedding 模型

集群购买完成后，前往 Kibana 部署 Embedding 模型、创建知识库索引与向量化管道。

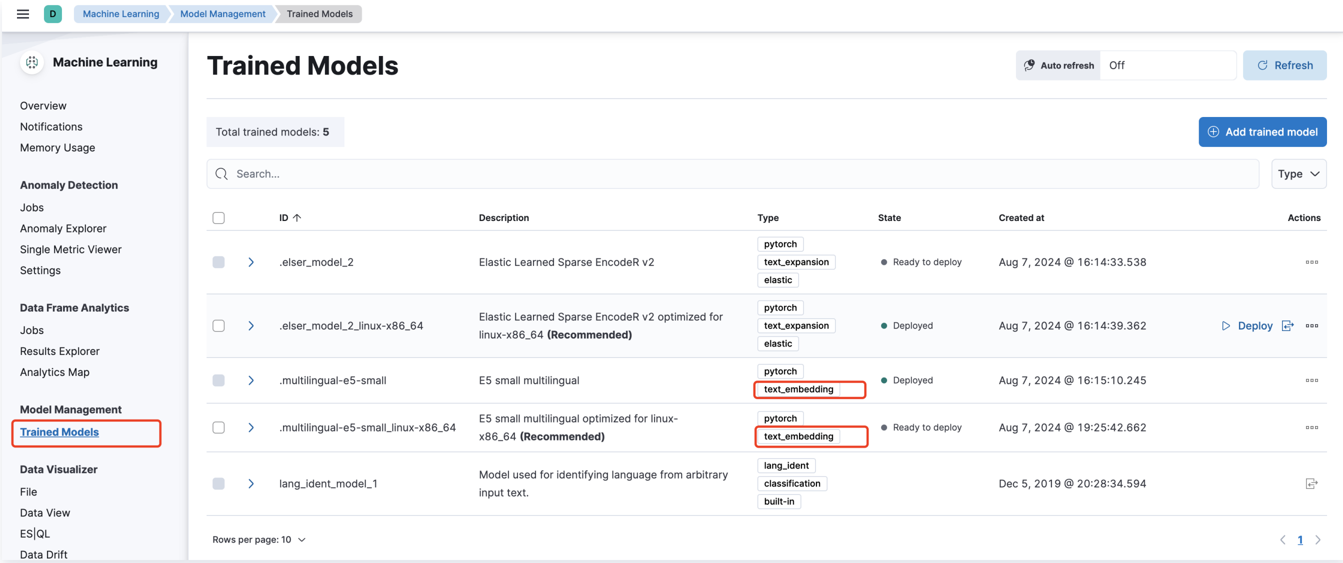
1. 开启**节点出站访问**，仅开启数据节点即可，如有专用主节点，仅开启专用主节点即可。（该功能为白名单，请联系 工单 处理）

> ⚠ **注意：**
> 如需上传自定义模型或第三方平台（例如 Huggingface）模型，可参见 文档。

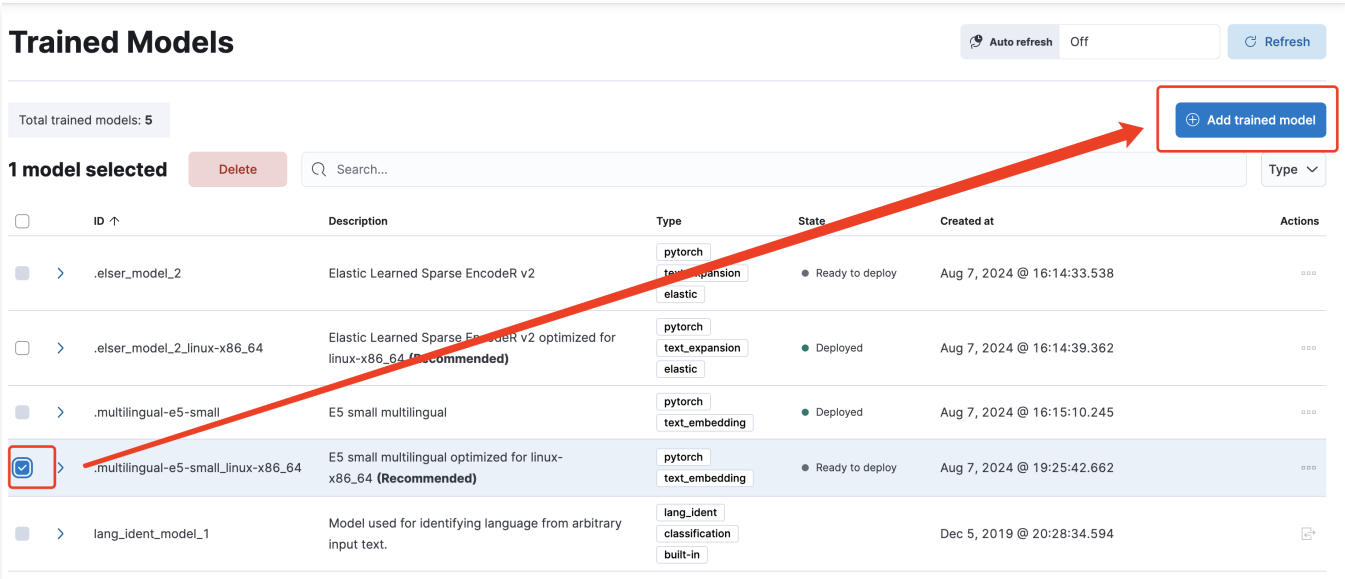
2. 登录 Kibana 后，在左侧导航栏找到 **Machine Learning** 功能。

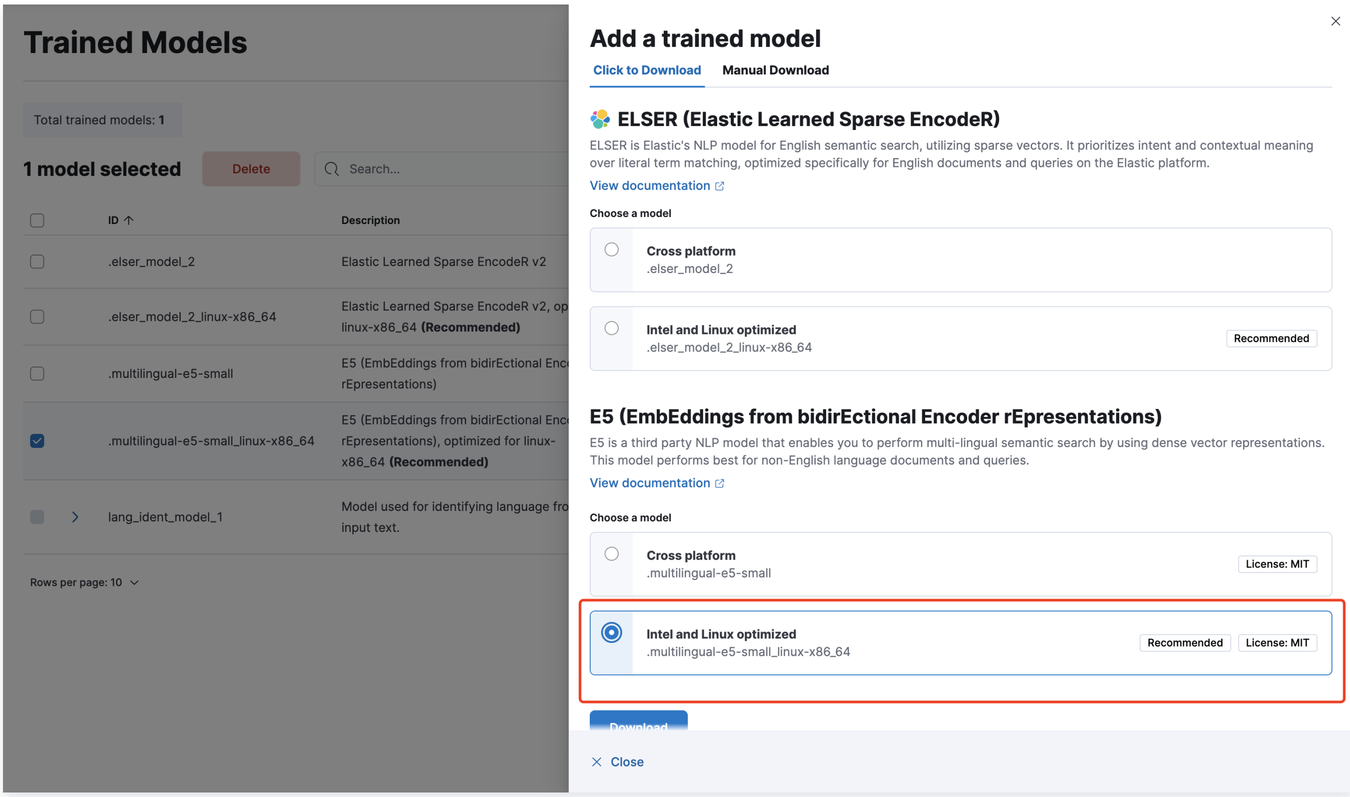


3. 进入模型管理页面，并找到类型为 **text_embedding** 的模型。

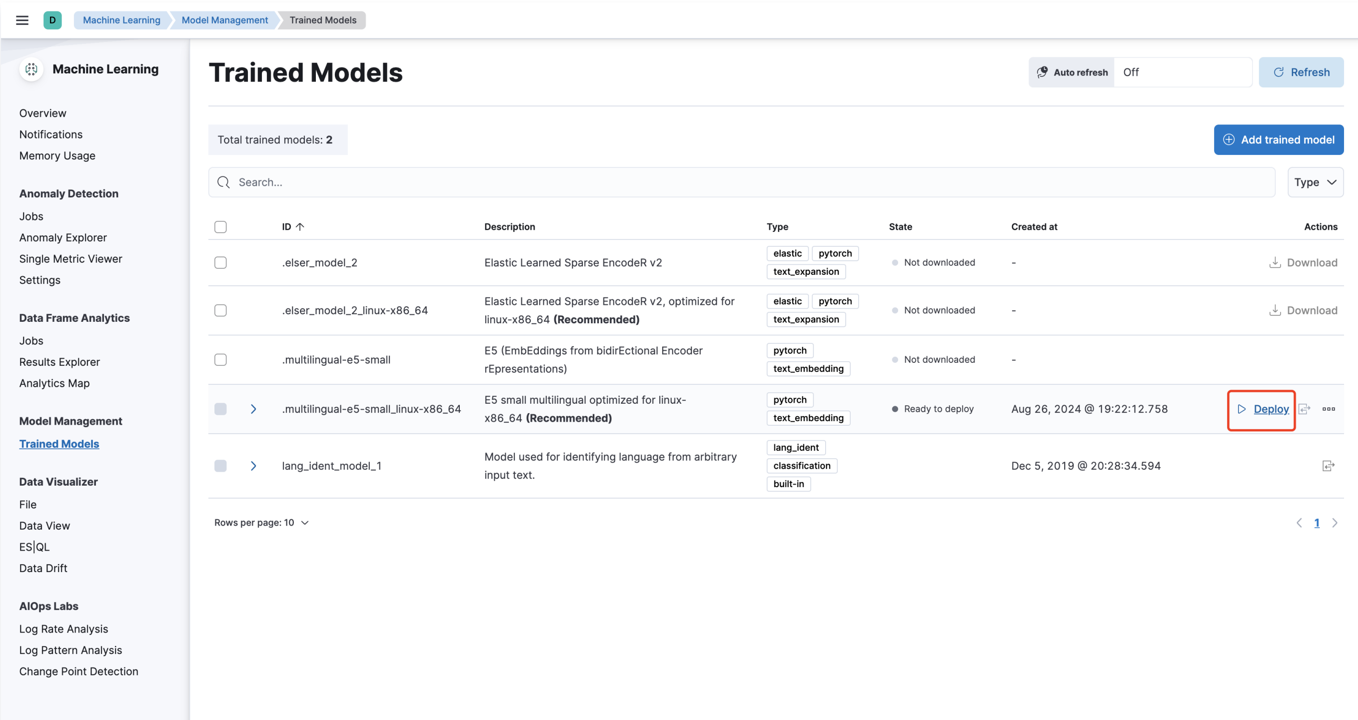


4. 如为未下载状态，选中模型，并单击 **Add trained model**，本次演示我们使用 **.multilingual-e5-small_linux-x86_64** 模型。

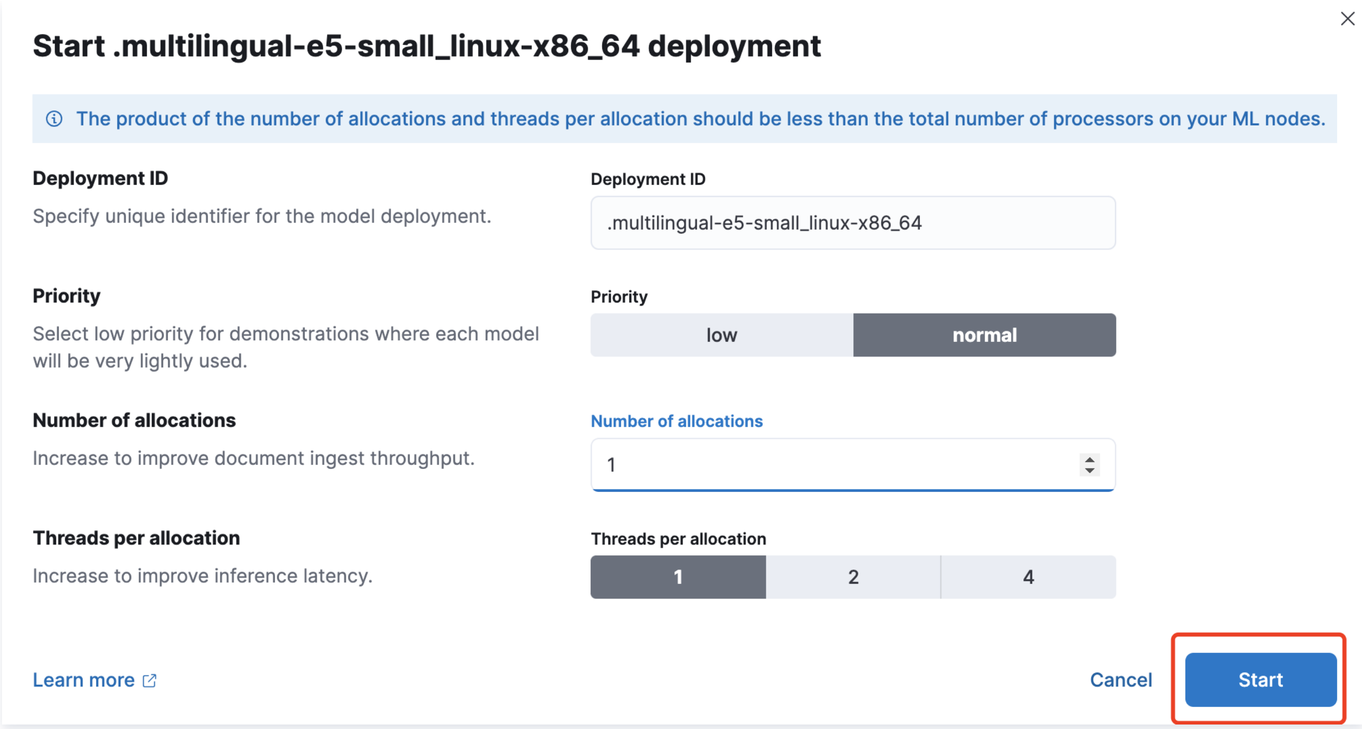




5. 下载完成后，单击**部署**。

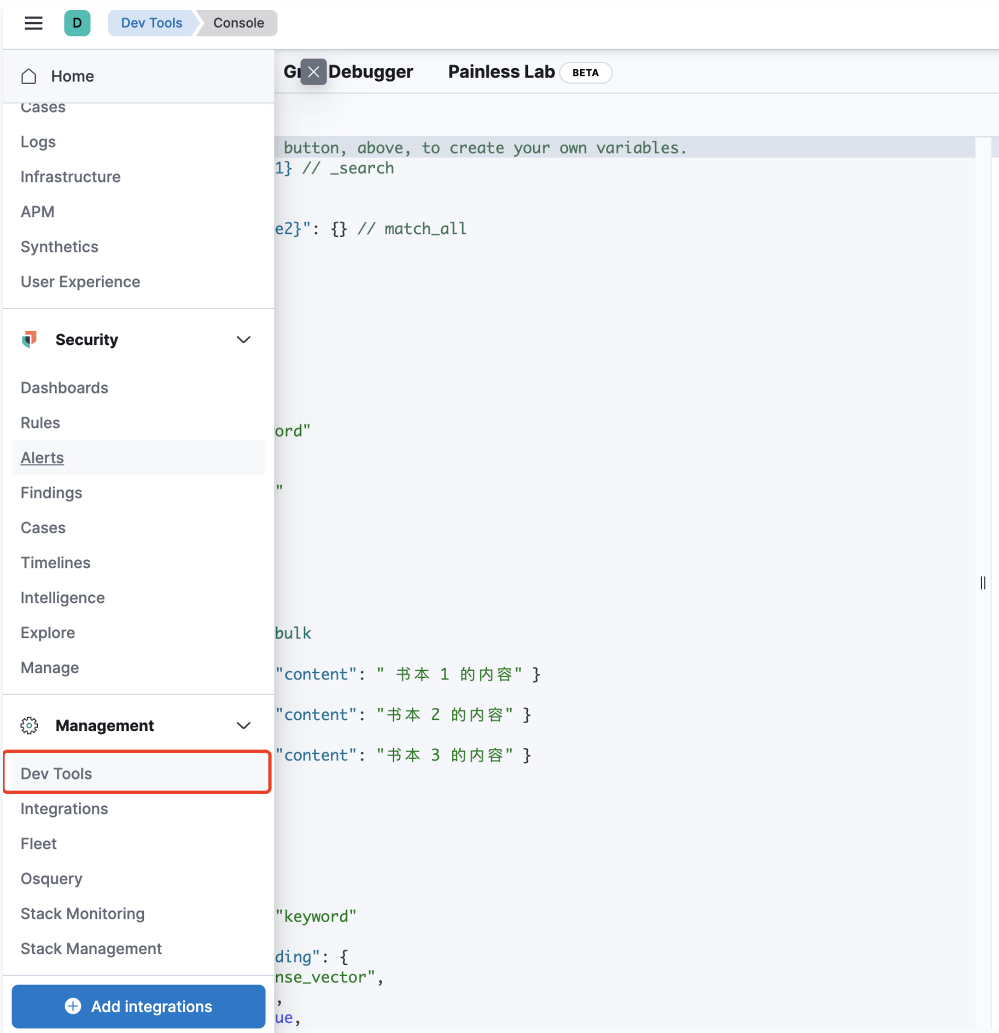


6. 为快速体验，配置使用默认值即可。



创建索引与向量化管道

1. 单击进入 Dev tools 页面。



2. 创建知识库索引

index-name 为索引名称，实际可按需命名。

```
PUT /index-name
{
  "mappings": {
    "properties": {
    "title": {
        "type": "keyword"
      },
      "content": {
        "type": "text"
      },
      "url": {
        "type": "keyword"
      }
    }
  }
}
```

3. 创建推理管道，该管道可用于写入数据时进行数据向量化。

```
PUT /_ingest/pipeline/index-name-pipeline
{
  "description": "Text embedding pipeline",
  "processors": [
    {
      "inference": {
        "model_id": ".multilingual-e5-small_linux-x86_64",
        "input_output": [
          {
            "input_field": "content",
            "output_field": "content_embedding"
          },
          {
            "input_field": "title",
            "output_field": "title_embedding"

          }
        ]
      }
    }
  ]
}
```

上述管道，将字段 content、title 的内容，调用 .multilingual-e5-small_linux-x86_64 模型向量化后存储到新的字段中。

写入知识库数据

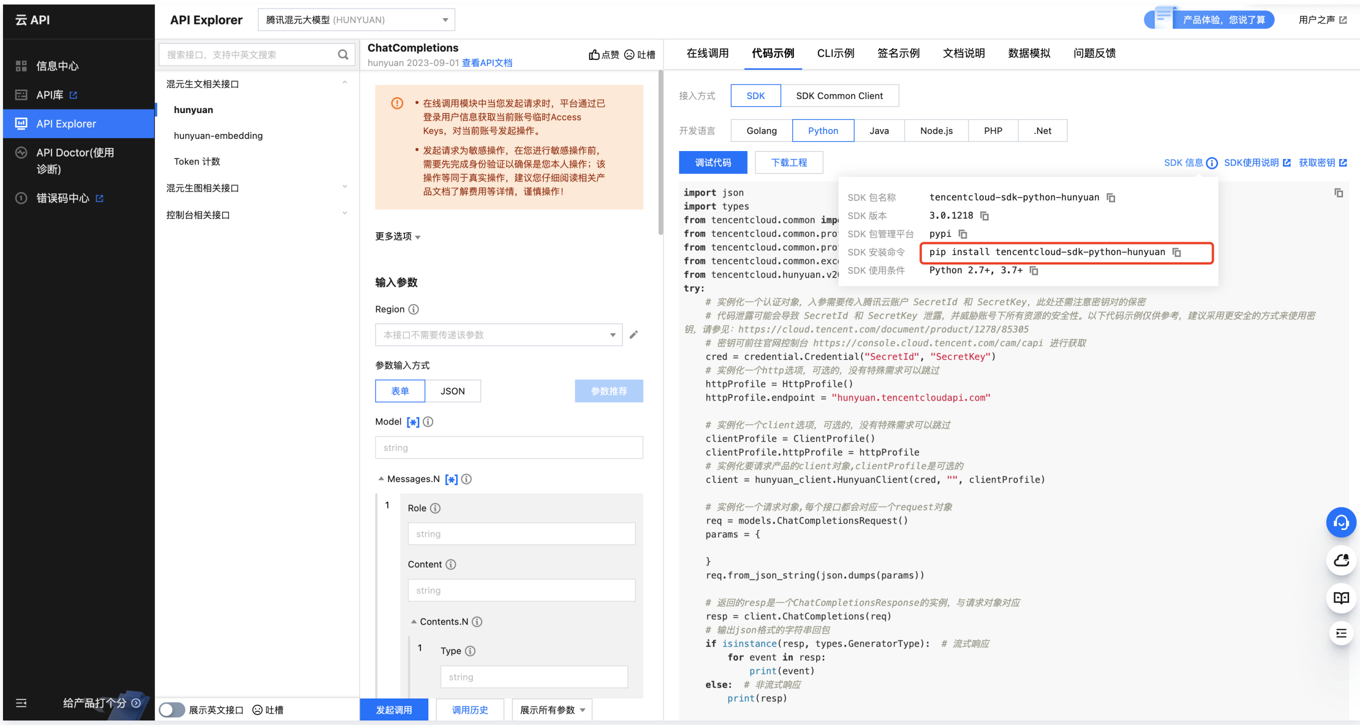
通过 Bulk API 批量写入数据，可将 title 、content 、url 的内容替换为您实际的知识库数据。

```
POST index-name/_bulk?pipeline=index-name-pipeline&refresh
{ "index" : {} }
{ "title" : "标题 1","content": "内容 1","url": "https: url1"  }
{ "index" : {} }
{ "title" : "标题 2","content": "内容 2","url": "https: url2"  }
```

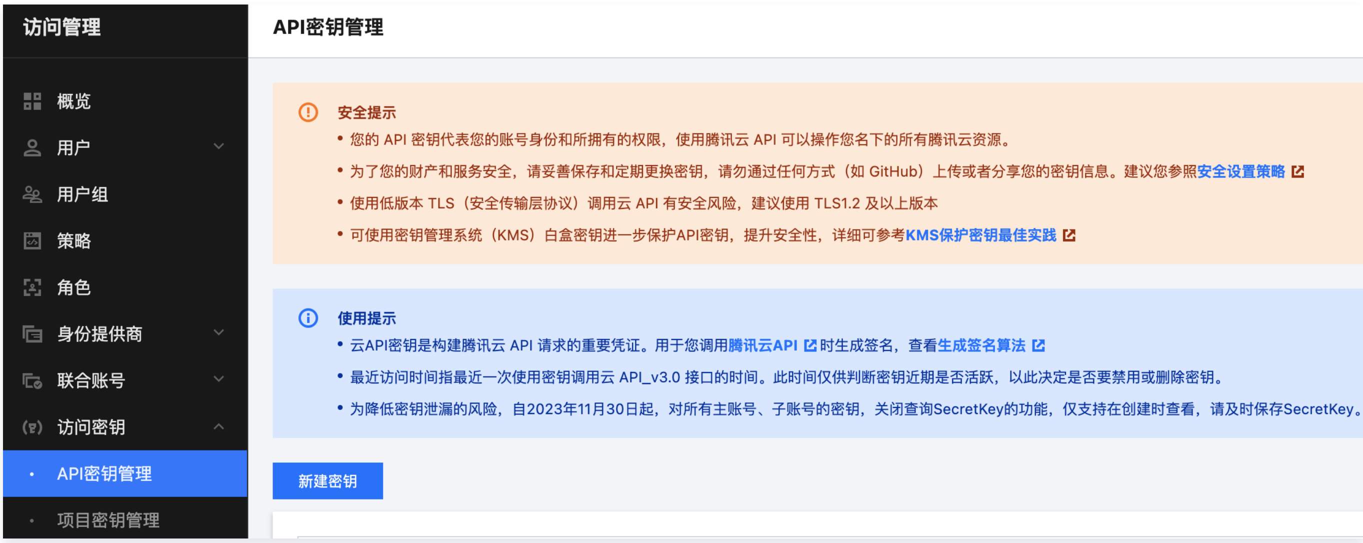
调用混元大模型

1. Python 文件命名为 **hunyuan.py**。

2. 进入 API Explorer 页面，安装相关库，参考如下：



3. 进入 API 密钥管理 界面，获取 SecretID、SecretKey。




```
import json
from tencentcloud.common import credential
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudSDKException
from tencentcloud.hunyuan.v20230901 import hunyuan_client, models

def hunyuan_gpt(system_prompt, content):
    try:
        # 密钥可前往官网控制台 https://console.cloud.tencent.com/cam/capi 进行获取
        cred = credential.Credential("SecretId", "SecretKey")
        # 实例化一个http选项，可选的，没有特殊需求可以跳过
        httpProfile = HttpProfile()
        httpProfile.endpoint = "hunyuan.tencentcloudapi.com"

        # 实例化一个client选项，可选的，没有特殊需求可以跳过
```

```
        clientProfile = ClientProfile()
        clientProfile.httpProfile = httpProfile

        # 实例化要请求产品的client对象,clientProfile是可选的
        client = hunyuan_client.HunyuanClient(cred, "", clientProfile)

        # 实例化一个请求对象,每个接口都会对应一个request对象
        req = models.ChatCompletionsRequest()
        params = {
            "Model": "hunyuan-standard",
            "Messages": [
                {"Role": "system", "Content": system_prompt},
                {"Role": "user", "Content": content}
            ]
        }
        req.from_json_string(json.dumps(params))

        # 返回的resp是一个ChatCompletionsResponse的实例，与请求对象对应
        resp = client.ChatCompletions(req)

        return resp

    except TencentCloudSDKException as err:
        return str(err)
```

智能问答系统构建

1. 安装 streamlit。

```
pip install streamlit
```

2. 获取 ES 访问地址。
   用户名为 elastic、密码在创建集群时设置，使用本地 mac 测试时，可开启公网访问，实际生产时，建议使用内网访问地址。

3. 运行如下代码，文件命名为 demo.py（需与 hunyuan.py 在一个目录下）。

```
import streamlit as st
from elasticsearch import Elasticsearch
from hunyuan import hunyuan_gpt

es_client = Elasticsearch(
    "ES 集群访问地址",basic_auth=("用户名", "密码"))

def get_elasticsearch_results(query):
```

```
    es_query = {
        "knn": {
            "field": "content_embedding",
            "num_candidates": 100,
            "query_vector_builder": {
                "text_embedding": {
                    "model_id": ".multilingual-e5-small_linux-x86_64",
                    "model_text": query
                }
            }
        },
        "query":{
            "match":{
                "content":query
            }
        },
        "rank":{
            "rrf":{
                "window_size":100,
                "rank_constant":20
            }
        }
    }
    result = es_client.search(index="index-name", body=es_query)
    return result["hits"]["hits"]

def create_hunyuan_prompt(results,question):
    context = ""
    for hit in results:
        source_field = hit["_source"]["content"]
        hit_context = source_field
        context += f"{hit_context}\n"
    prompt = f"""
Instructions:

回答此问题：{question}
回答时，你只能参考文档{context} 生成答案，如果无法生成答案，请回复：对不起，该问题我无法回答

"""
    return prompt

def generate_hunyuan_completion(user_prompt, question):
    response = hunyuan_gpt(user_prompt, question)
    return response.Choices[0].Message.Content

def format_result(hit):
    title = hit["_source"]["title"]
    return f'[{title}]'

def main():
    st.title("我的专属AI助手")

    # 创建输入框和查询按钮
    question = st.text_input("请输入您的问题：")
    if st.button("查询并生成答案"):
        # 获取Elasticsearch结果并创建HunYuan提示
        elasticsearch_results = get_elasticsearch_results(question)
        user_prompt = create_hunyuan_prompt(elasticsearch_results,question)
        system_prompt = f"""
你是一个问答任务的助手。使用呈现的上下文真实、实事求是地回答问题。
"""
```

```
        # 使用HunYuan模型生成回答
        openai_completion = generate_hunyuan_completion(system_prompt, user_prompt)

        # 显示结果
        st.write(openai_completion)

        # 展示Elasticsearch查询结果
        st.write("大模型参考的文档：")
        for hit in elasticsearch_results:
            st.markdown(format_result(hit))

if __name__ == "__main__":
    main()
```

4. 在上述 python 文件的目录下，使用如下命令运行系统：

```
streamlit run demo.py
```

5. 生成的界面如下：

总结

本文通过介绍腾讯云 ES 一站式 RAG 方案，演示如何通过结合腾讯云 ES 与混元大模型，快速构建 RAG 应用。腾讯云 ES 凭借其在传统 PB 级日志和海量搜索场景中积累的丰富经验，通过深度重构底层系统，成功地将多年的性能优化、索引构建和运营管理经验应用于 RAG 领域，并积极探索向量召回与传统搜索技术的融合之道，旨在充分发挥两者的优势，为用户提供更加精准、高效的搜索体验。未来，腾讯云 ES 将持续深耕智能检索领域，在成本、性能、稳定性等方面持续提升，帮助客户降本增效的同时实现业务价值持续增长，欢迎持续关注！

SQL 支持

最近更新时间：2024-10-16 11:20:11

腾讯云 Elasticsearch Service 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 提供了两种 SQL 解析器。ES 所有的开源版本，均预装了开源社区提供的 SQL 解析插件。ES 6.4.3及以上版本，包括基础版和白金版，支持使用 ES 原生的 SQL 解析器。

原生 SQL 解析器

使用 SQL 的 API 进行简单的查询。

```
POST /_xpack/sql?format=txt
{
    "query": "SELECT * FROM my_index"
}
```

> **说明**
> 使用时将以上示例中的 my_index 替换成已有的索引。

更多原生 SQL 解析器的 API 及使用方法请参见 官方文档。

开源 SQL 解析插件

- 7.5.1及以上版本：

```
POST /_nlpcn/sql
{
      "sql":"select * from test_index"
}
```

- 其他版本：

```
POST /_sql
{
      "sql":"select * from test_index"
}
```

> **说明**
> 使用时将以上示例中 test_index 替换成已有的索引。

更多 SQL 插件的 API 及使用方法请参见 文档。

SQL JDBC 访问

ES 6.4.3及以上的白金版中，支持通过 JDBC 访问 ES 集群。您首先需要下载 JDBC 驱动，JDBC 驱动可以 点击下载，或在 Maven 中将如下内容粘贴到 pom 文件，添加依赖来下载：

```
<dependency>
  <groupId>org.elasticsearch.plugin</groupId>
  <artifactId>x-pack-sql-jdbc</artifactId>
  <version>6.4.3</version>
</dependency>
```

SQL JDBC 访问示例代码：

```
import java.sql.*;
```

```java
import java.util.Properties;

public class Main {

    public static void main(String[] args) {
        try {
            Class.forName("org.elasticsearch.xpack.sql.jdbc.jdbc.JdbcDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            return;
        }
        String address = "jdbc:es://http://YOUR_ES_VIP:9200";
        Properties properties = new Properties();
        properties.put("user", "elastic");
        properties.put("password", "YOUR_PASS");

        Connection connection = null;
        try {
            connection = DriverManager.getConnection(address, properties);
            Statement statement = connection.createStatement();
            ResultSet results = statement.executeQuery("select FlightNum from kibana_sample_data_flights
limit 10");
            while (results.next()) {
                System.out.println(results.getString(1));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (connection != null && !connection.isClosed()) {
                    connection.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```


企业微信机器人接收 Watcher 告警

最近更新时间：2024-11-14 16:38:32

腾讯云 Elasticsearch Service 白金版中支持了 X-Pack Watcher 特性，通过添加触发器、操作等配置，可以实现当条件满足时执行某些特定操作。例如当检测到索引中出现错误日志时自动发送告警。本文介绍如何配置企业微信机器人接收 Watcher 发出的告警。

> **注意**
> - X-Pack Watcher 特性仅在白金版中提供。
> - 由于腾讯云 Elasticsearch Service 网络架构调整，仅2020年6月及之后创建的实例支持配置企业微信机器人接收 Watcher 告警。

背景信息

一个 Watcher 由4部分组成，具体如下：
- Trigger：定义了何时 Watcher 开始执行，在配置 Watcher 时必须设置。支持的触发器详情参见 Schedule Trigger 。
- Input：对监控的索引执行的查询条件，同时在触发 Watcher 时，Input 将数据加载到执行上下文中，在后续的 Watcher 执行阶段，可访问这个上下文。详情请参见 Inputs 。
- Condition：执行 Actions 需要满足的条件。
- Actions：当条件发生时，执行的具体操作。例如本文介绍的 Webhook Action。

操作步骤

1. 准备一台与 ES 集群同 VPC 的并且可以访问 Webhook 地址的 CVM（如通过外网访问）。
2. 在 CVM 中安装 Nginx，具体安装方法参见 Nginx 安装 。
3. 配置 Nginx 代理转发。使用以下配置替换 nginx.conf 文件中 Server 部分的配置。

- Nginx 服务的默认端口是80，若您需要更改其端口，则需要登录控制台 安全组 放行此端口。
- <企业微信机器人 Webhook 地址>：需替换为接收报警消息的企业微信机器人的 Webhook 地址。

```
server {
         listen       80;
         server_name  localhost;
         index index.html index.htm index.php;
         root /usr/local/nginx/html;
         #charset koi8-r;

         #access_log  logs/host.access.log  main;
         location ~ .*\.(php|php5)?$
         {
             fastcgi_pass 127.0.0.1:9000;
             fastcgi_index index.php;
             include fastcgi.conf;
         }
         location ~ .*\.(gif|jpg|jpeg|png|bmp|swf|ico)$
         {
             expires 30d;
         # access_log off;
         }
         location / {
             proxy_pass <企业微信机器人的wehbook地址>;
         }
         location ~ .*\.(js|css)?$
         {
             expires 15d;
             # access_log off;
         }
         access_log off;

         #error_page  404              /404.html;
```

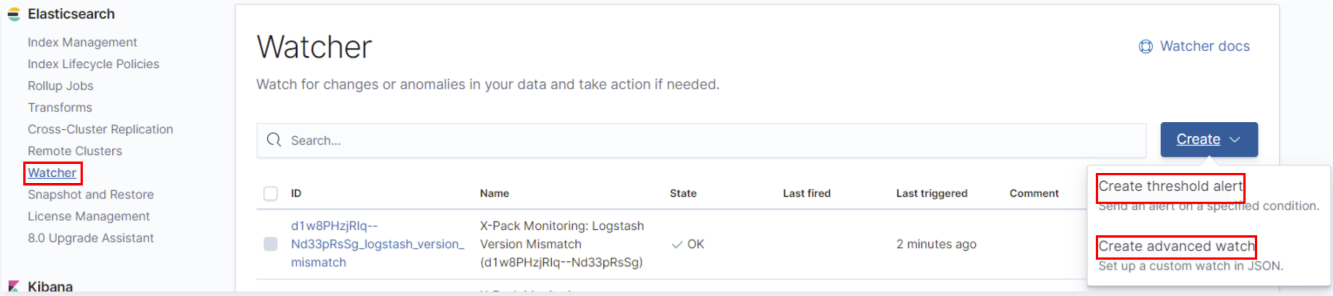
```
        # redirect server error pages to the static page /50x.html

        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
                root   html;
        }
}
```

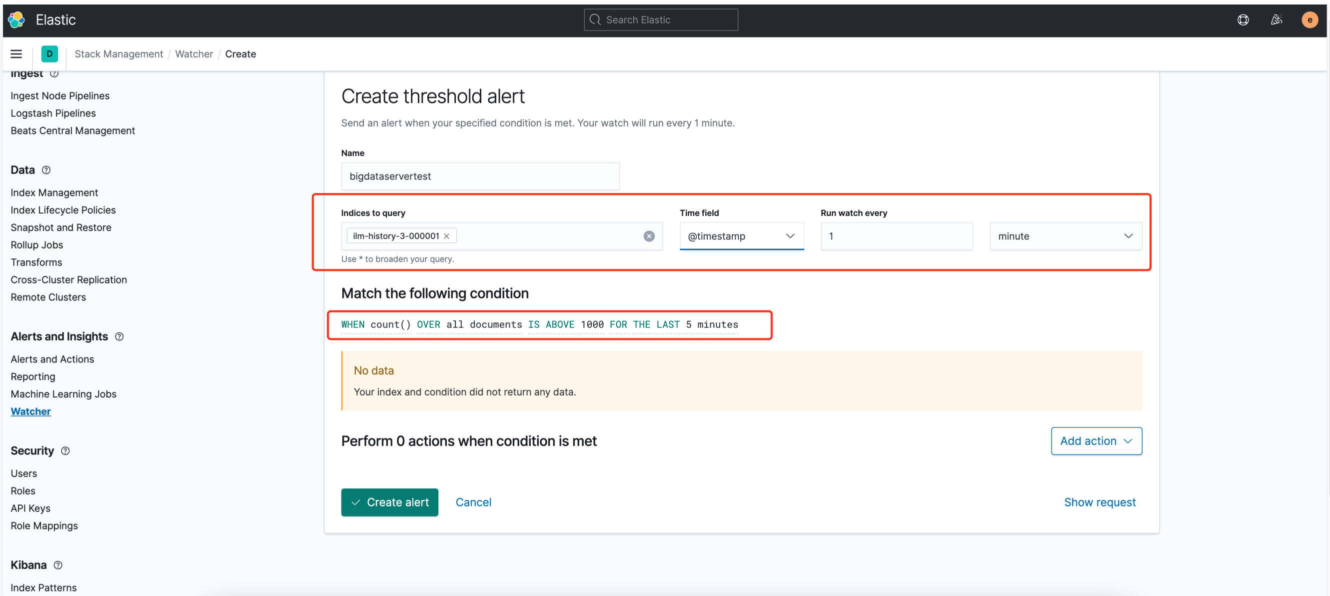
4. 加载修改后的配置文件并重启 Nginx。

```
/usr/local/webserver/nginx/sbin/nginx -s reload
/usr/local/webserver/nginx/sbin/nginx -s reopen
```

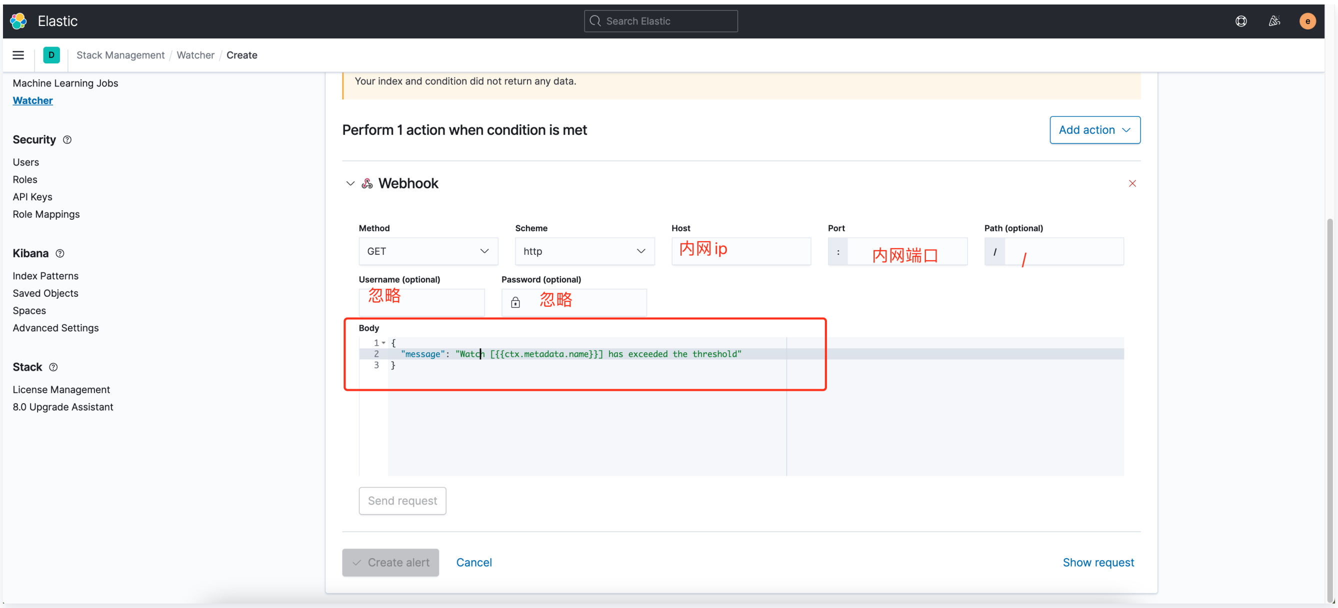
5. 配置 Watcher 报警规则。**此步骤可以在 Kibana 界面 Management > Watcher 选项中进行图形化操作。**



- `Create threshold alert` 在界面进行阈值告警设置。可以针对某索引的特定条件进行监控告警，例如 CPU 使用率、文档个数等，可以在下面的 Condition 选项做更细节的设置，参考如下：

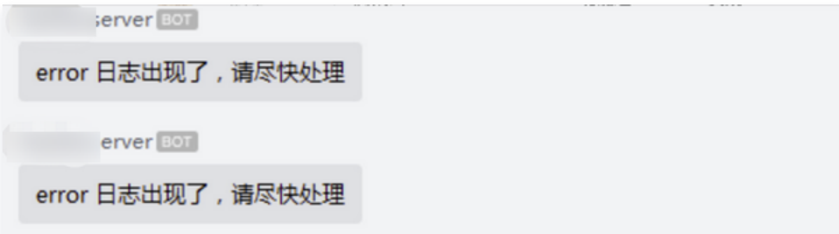


- 单击右上角的 **Add action**, 选择 “Wehook”，相关设置如下：



- 单击 **Send request** 可以进行测试，然后单击 **Create alert** 即可。
- `Create advanced watch` 通过 API 设置 Watcher 各参数，API 详情请参见 PUT Watch。

6. 以上步骤配置完成后，即可在自己创建的企业微信群中接收到机器人发来的告警信息。



HTTPS 集群访问通信实践教程

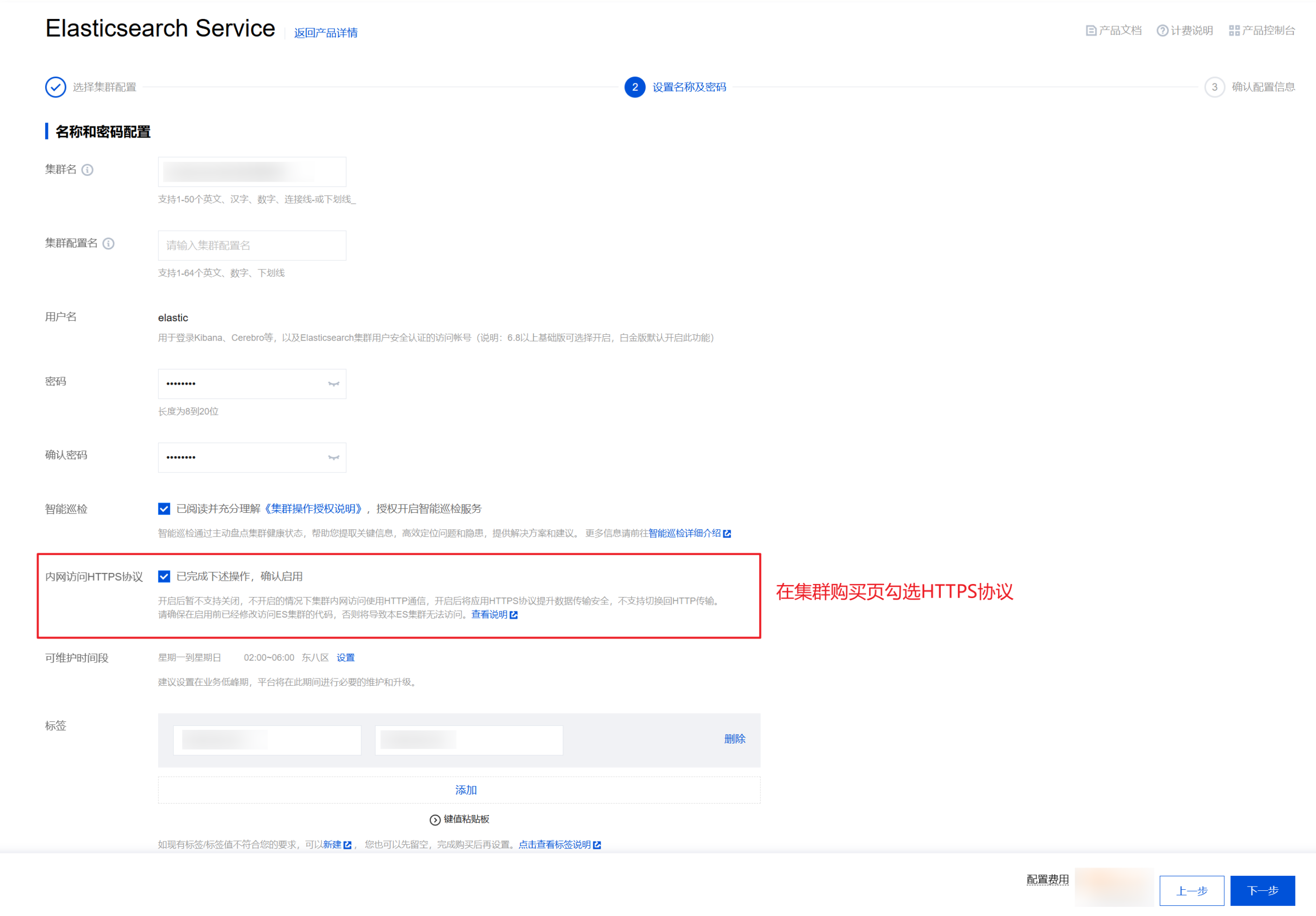
最近更新时间：2025-06-23 17:38:12

HTTPS 协议是一种以安全为目的的 HTTP 通道，在 HTTP 的基础上通过传输加密和身份认证等机制，可保障数据传输过程中的安全性。本文将基于腾讯云 ES 集群环境，演示 Beats、Logstash、Kibana 和 Java Client 等客户端访问连接开启了 HTTPS 协议的 ES 集群。

HTTPS 集群环境准备

创建 HTTPS 协议集群

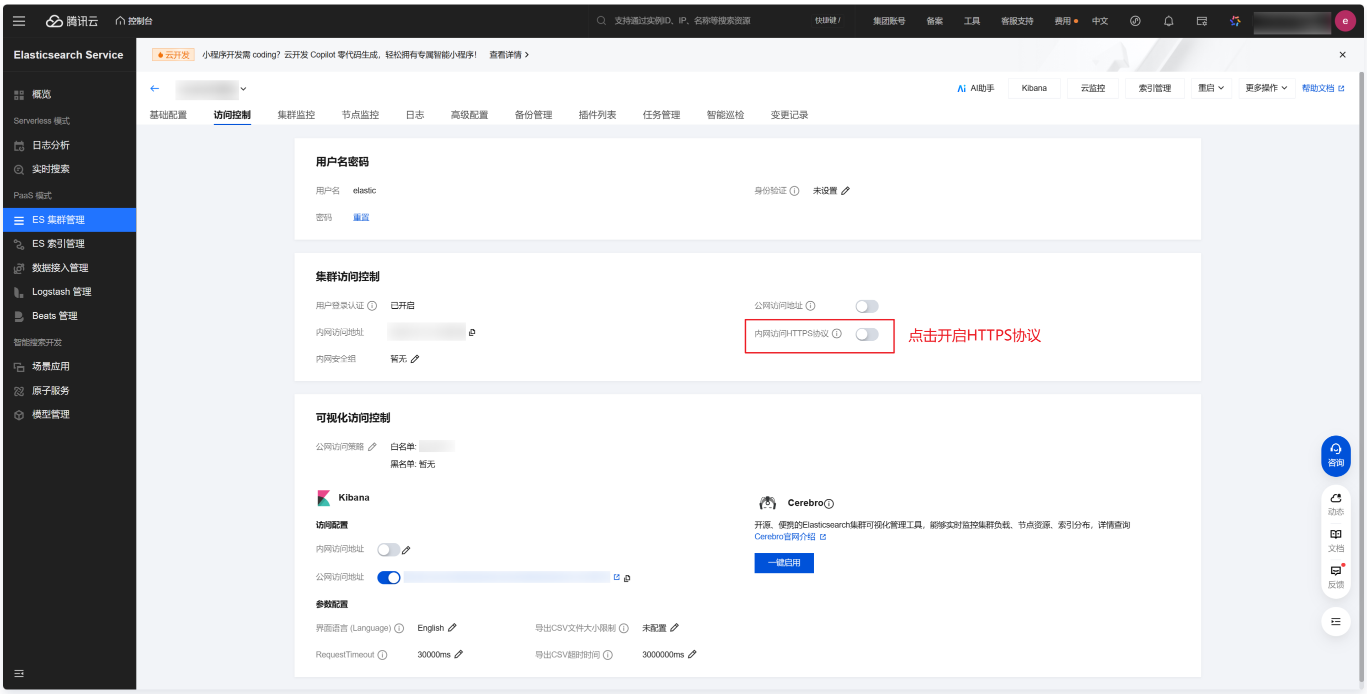
- 首先在腾讯云 ES 控制台创建出一个 HTTPS 集群，在购买页勾选 HTTPS 协议，目前开启 HTTPS 协议后暂不支持关闭。



其中，ES 通过在 elasticsearch.yml 配置文件中设置特定参数来启用 HTTPS 协议的，参数如下所示：

```
xpack.security.http.ssl.enabled: true
xpack.security.http.ssl.keystore.path: certs/ces-certificates.p12
xpack.security.http.ssl.truststore.path: certs/ces-certificates.p12
```

- 如果在创建集群页没有开启 HTTPS 协议，可以通过在 ES 集群管理界面中点击访问控制，在集群访问控制模块中开启内网访问 HTTPS 协议。



获取 pem 证书文件

集群创建成功后，可在 ES 控制台 访问控制页面上的集群访问控制模块，进行相关证书导出。

此时腾讯云 ES 侧会提供如下两个证书文件：

文件名称	用途
client-certificates.pem	用于 Beats Logstash、Java Client 连接 ES 集群
server-certificates.pem	用于 Kibana 连接 ES 集群

下面将详细介绍 Beats、Logstash、Kibana 和 Java 等客户端连接 HTTPS 集群的配置方式。

Beats 输出到 HTTPS 集群

CVM Metricbeat 输出到 ES

1. 我们首先在腾讯云 CVM 控制台 创建一个和 ES 集群同 VPC 下的 CVM，创建好后，将得到的 pem 鉴权文件上传到该 CVM 上，这里的存放路径为：/usr/local/service/https-certs。随后到腾讯云 ES 控制台 的 Beats 管理页，创建一个 Metricbeat。

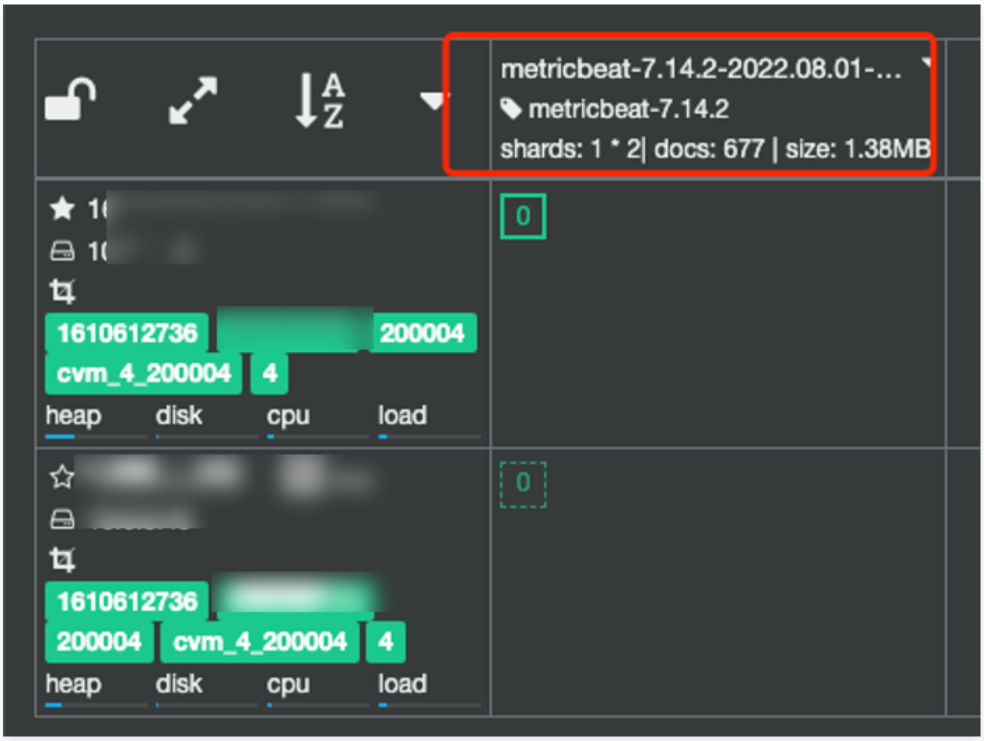
2. 其中，最核心的步骤是在 metricbeat.yml 配置文件中进行如下配置。

```
output.elasticsearch:
    hosts: ["https://ES-VIP:9200"]
    username: "elastic"
    password: "changeme"
    ssl.certificate_authorities: ["/usr/local/service/https-certs/client-certificates.pem"]
    ssl.verification_mode: certificate
```

配置信息说明：

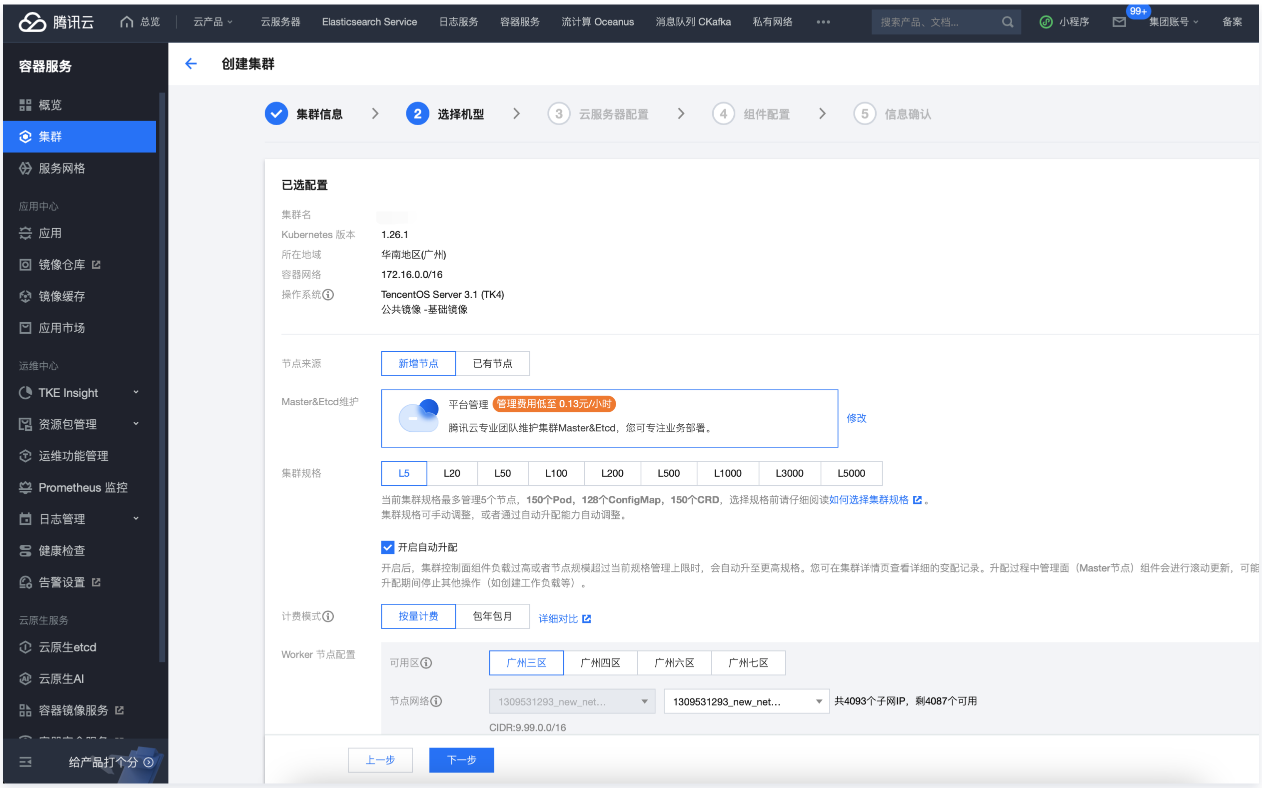
配置项	说明
hosts	ES 集群的 VIP，如 `https://10.0.X.29:9200` ，以 https 开头
username/password	ES 集群的用户名密码
ssl.certificate_authorities	连接 HTTPS 集群所需的 pem 鉴权证书文件路径
ssl.verification_mode	服务器证书认证模式，有四种模式，分别是 full，strict，certificate 和 none。我们这里以 certificate 模式进行认证，即只认证 CA 证书，不认证主机名信息。详情可参考官方文档。

3. 配置完成后，即可在 ES 集群的 Cerebro 或者 Kibana 中看到自动创建了一个 metricbeat-7.14.20-*开头的索引，到此 CVM 中 Metricbeat 连接 HTTPS 的 ES 集群配置完成。

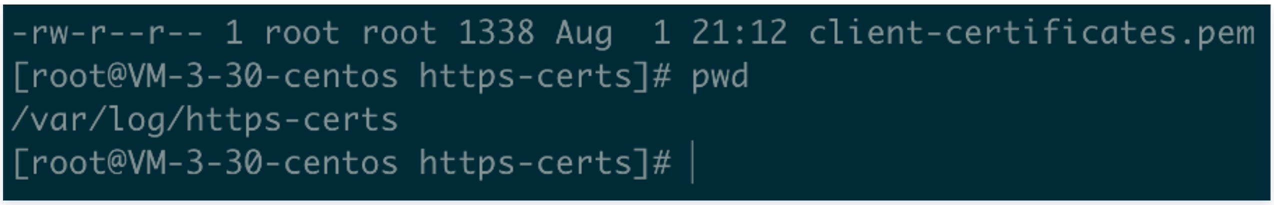


TKE Filebeat 日志采集器输出到 ES

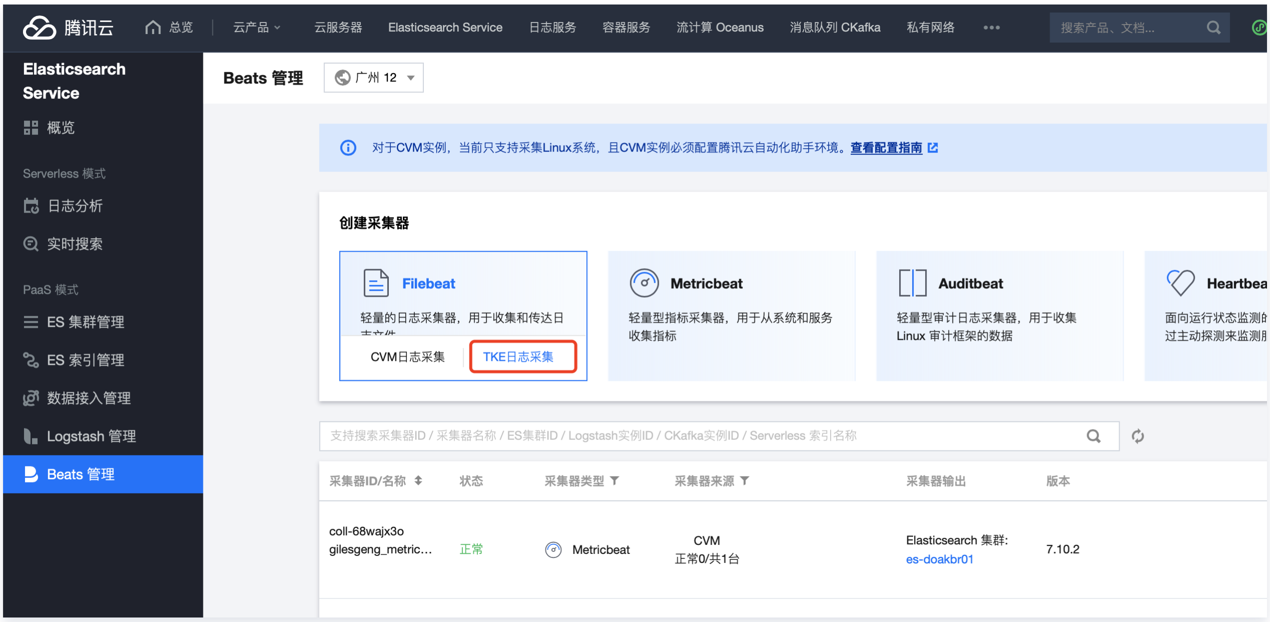
1. TKE Filebeat 日志采集器输出到 HTTPS 的 ES 集群流程和 CVM 的 Metricbeat 输出一样，首先我们将 pem 文件上传到创建 TKE 集群时自动创建的 Worker 所在的 CVM 节点上，如 /var/log/https-certs 目录下。



2. client-certificates.pem 文件存放位置：



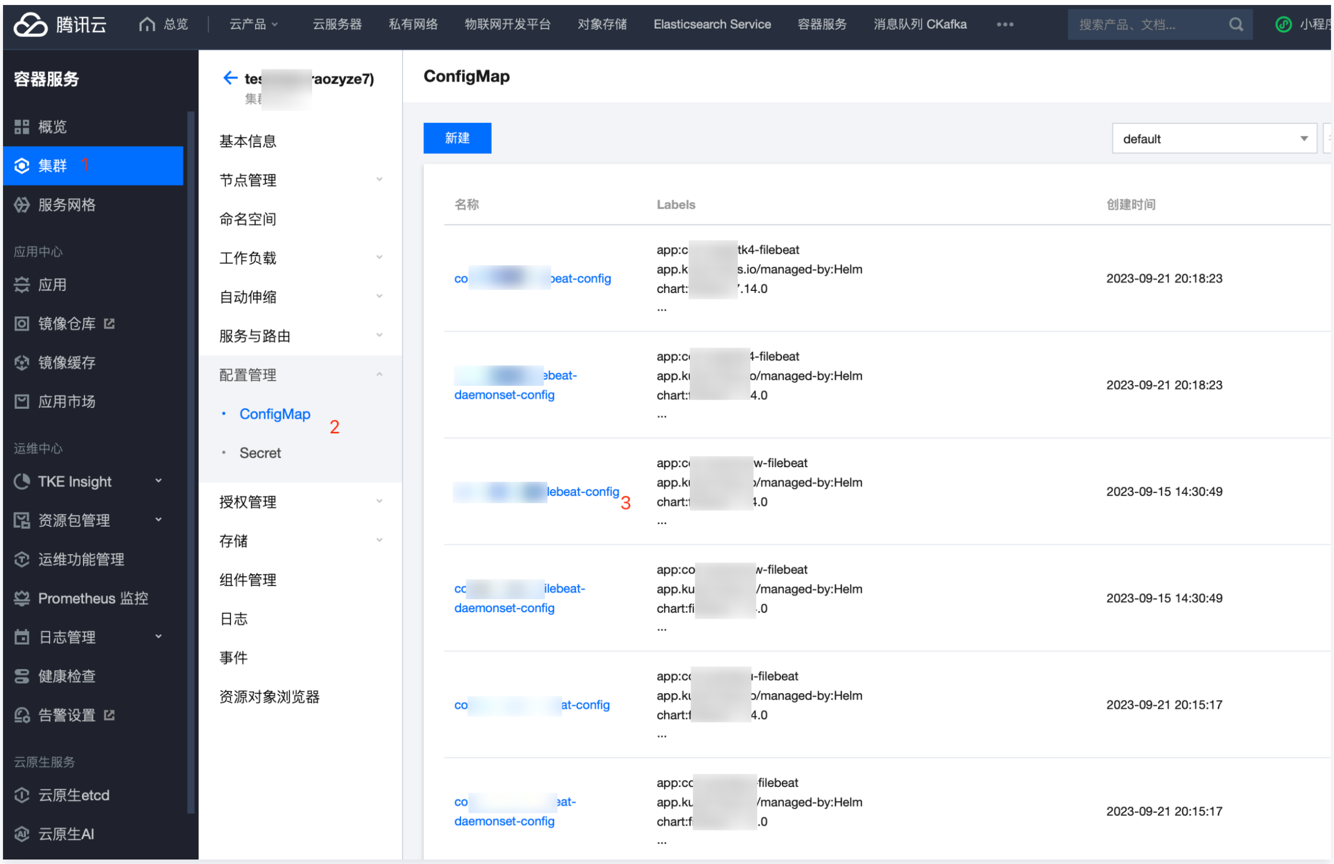
3. 随后我们在腾讯云 ES 控制台新建 Fliebeat 采集器，这里选择 TKE 日志采集：



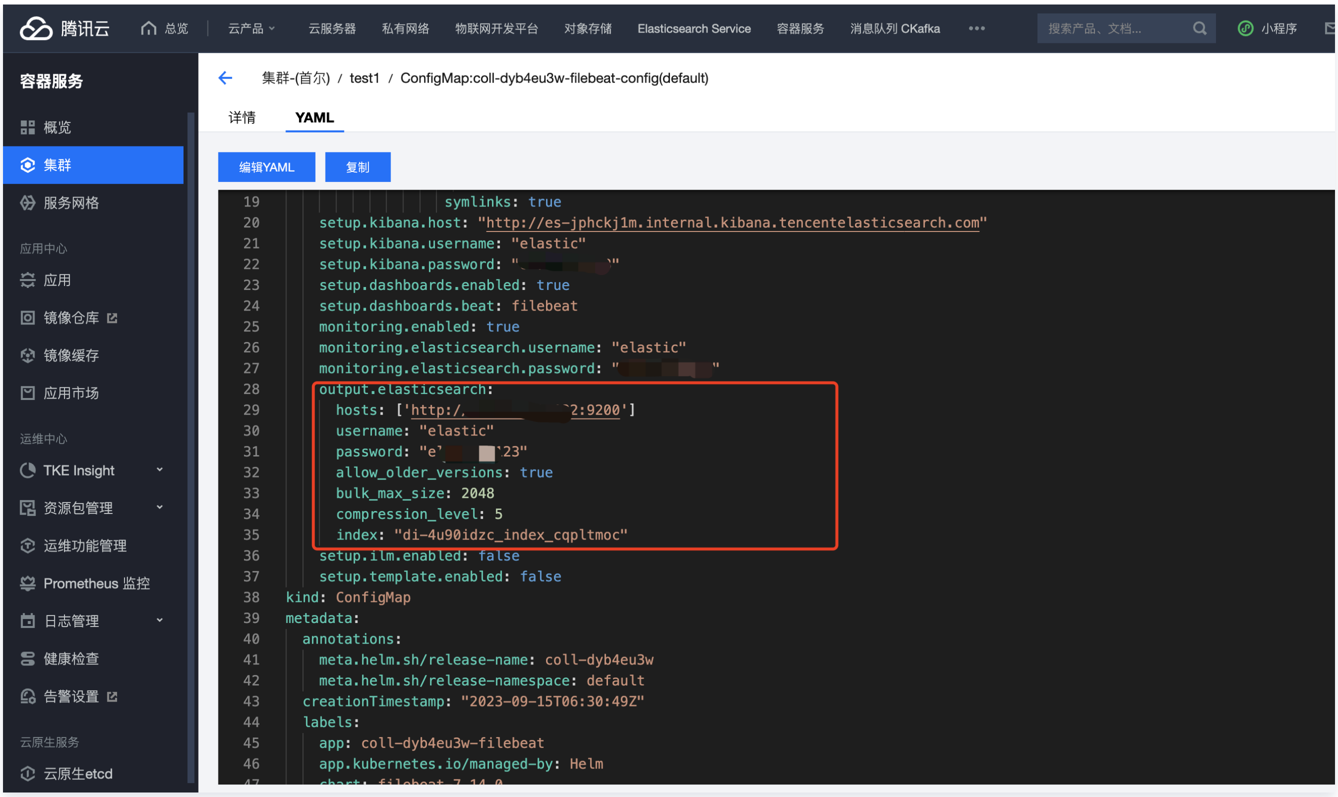
4. 下面进行 Filebeat 采集器的基本信息配置，选择版本号、采集器输出 ES 集群。如下图所示：

5. 开始配置采集来源，如下图所示：

6. 创建好 Filebeat TKE 容器日志采集器后，随后我们在 TKE 集群的详情页，找到配置管理中的 ConfigMap，然后找到对应 beats 的 config 文件：



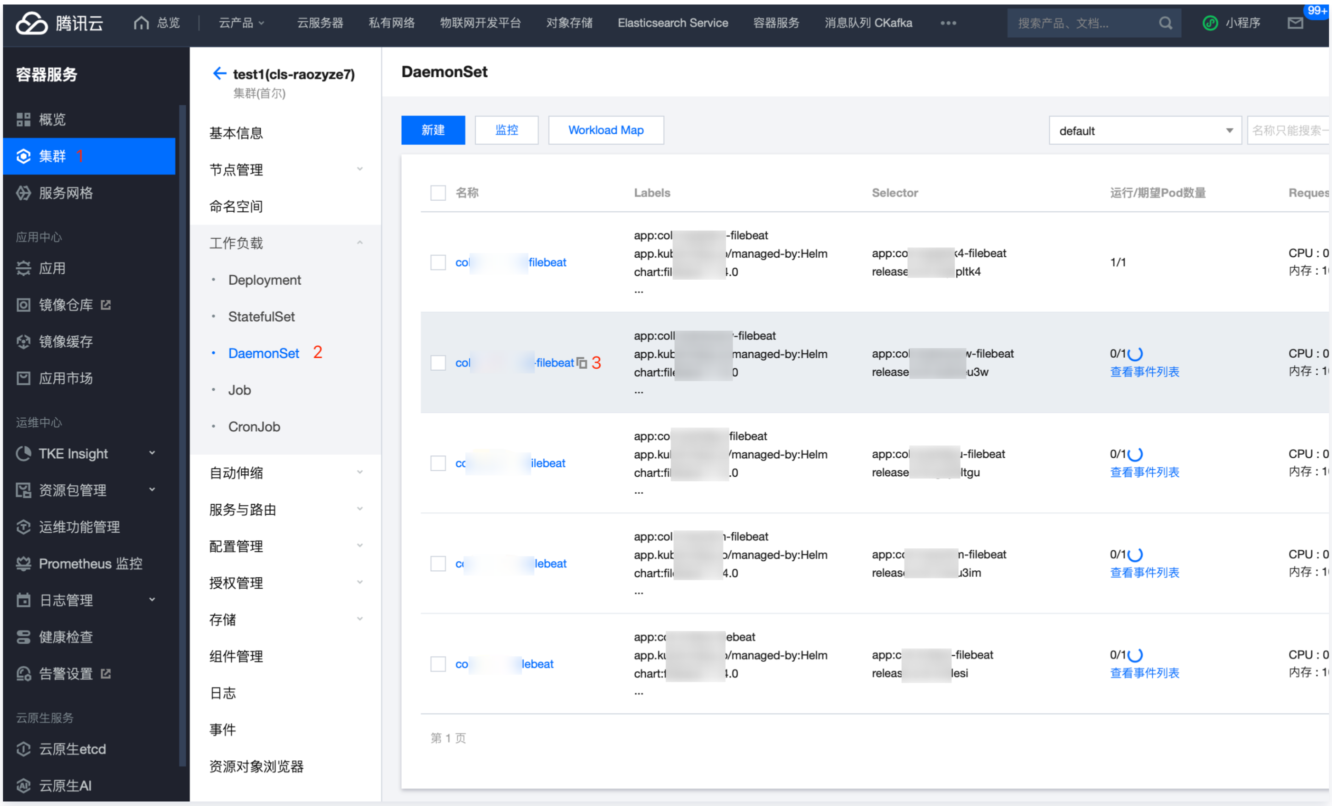
7. 单击**编辑 yaml** 后，对 output.elasticsearch 配置项进行修改，如下图所示：



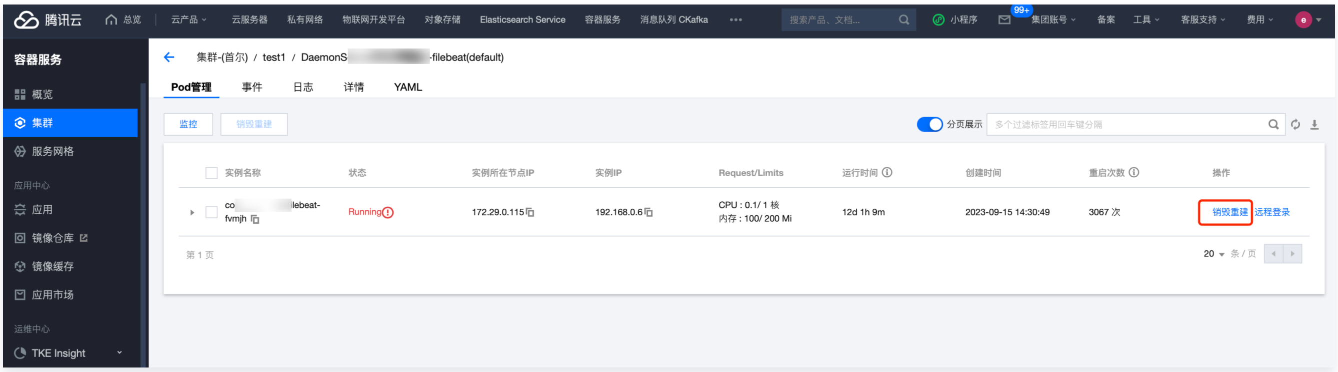
8. 具体配置信息如下，配置信息说明同 Metricbeat 一致。

```
output.elasticsearch:
      hosts: ['https://ES-VIP:9200']
      username: "elastic"
      ssl.certificate_authorities: ["/var/log/https-certs/client-certificates.pem"]
      ssl.verification_mode: "certificate"
      password: "changeme"
      indices:
        - index: "filebeat-tke-%{+yyyy.MM.dd}"
          when.equals:
            tke_collector_target_name: "logs_to_https_es"
```

9. 然后在 TKE 集群中的工作负载中，找到 DaemonSet，然后再到对应 beats 的 DaemonSet，单击进去，对它的 pod 进行销毁重建。

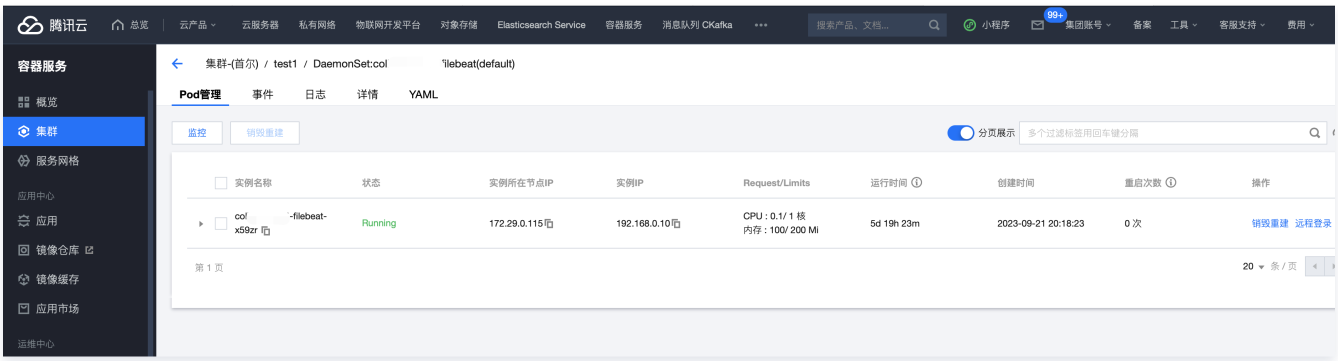


10. 进入到 Pod 管理页，选择对应的 Pod 销毁重建。

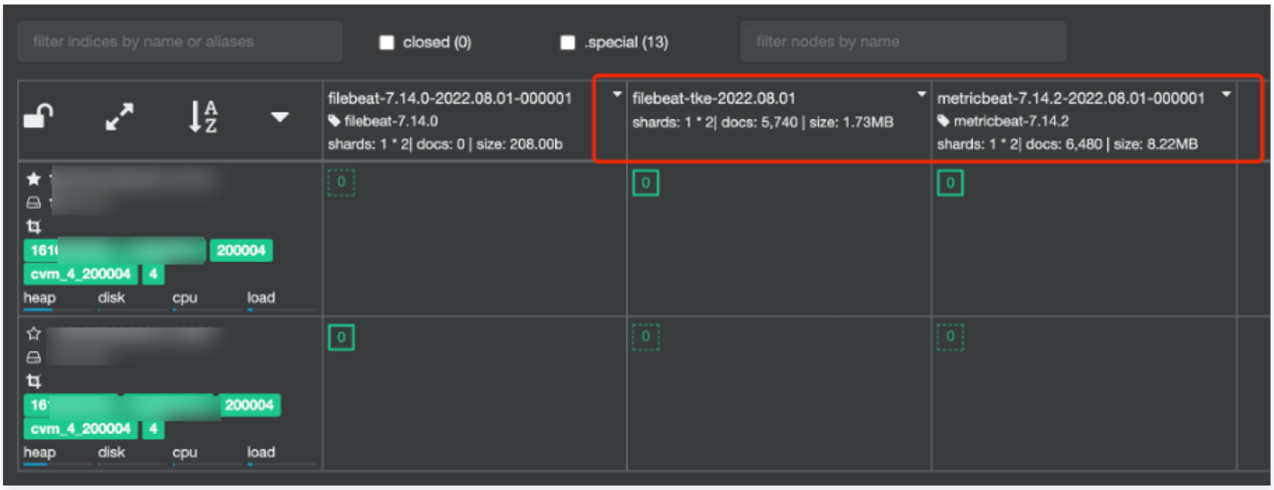


11. 销毁重建后，Pod 的运行状态变成 Running，如下图所示。

> **注意：**
> 如果多次对 pod 进行销毁重建，仍是红色 Running，有可能是 beats 的 yml 配置出现问题，可以在日志里面查看具体问题，或者重新检查一下 yml 配置是否错误。

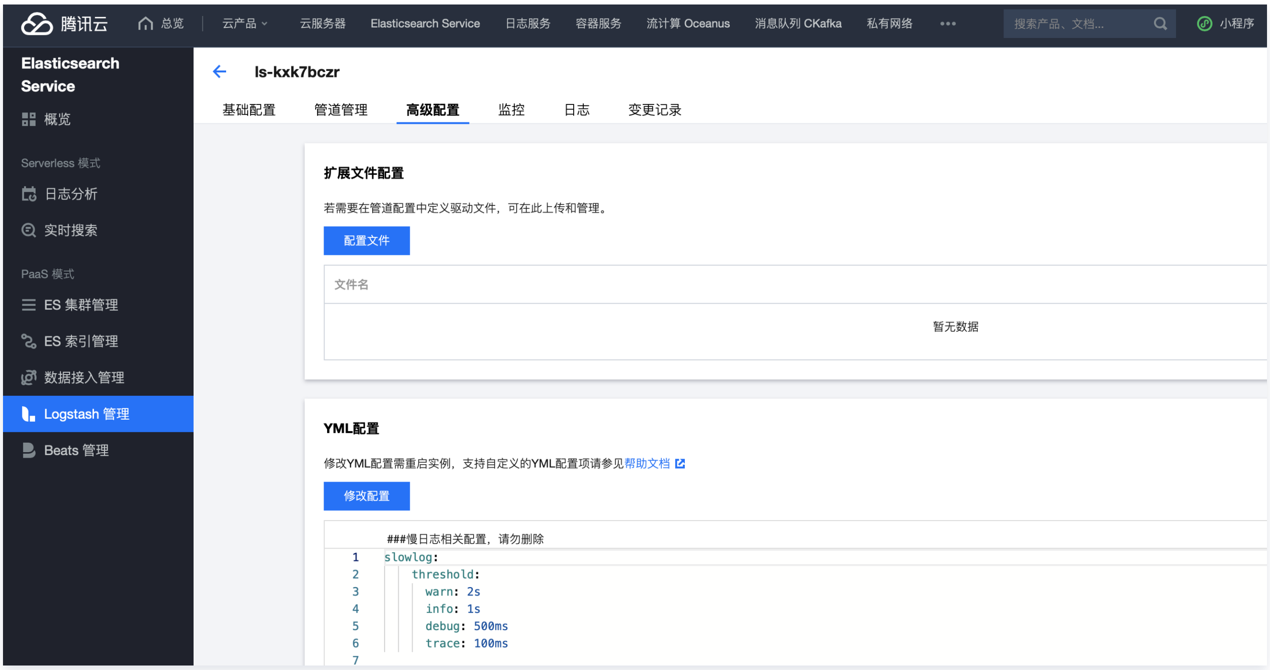


12. 随后我们到 ES 集群中，可以看到自动创建了以 filebeat-tke-*开头的索引。表明 TKE 日志顺利输出到 HTTPS 集群中了。



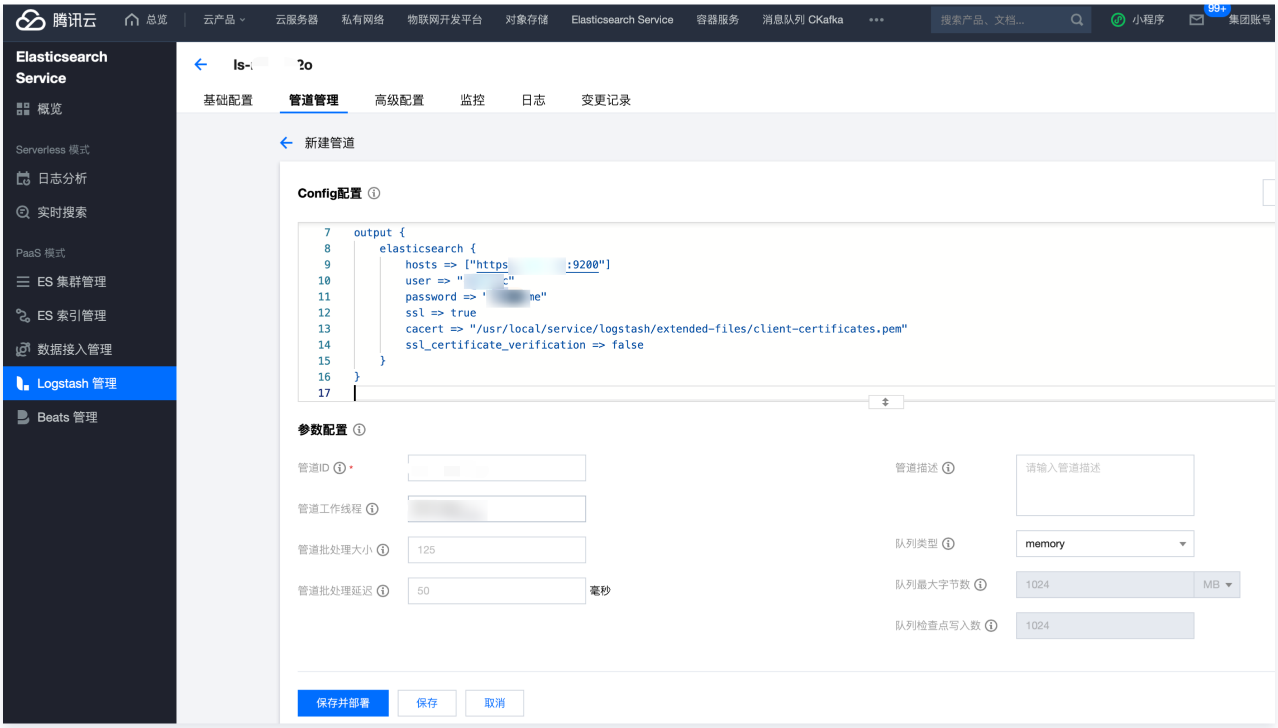
Logstash 输出到 HTTPS 集群

1. 腾讯云 Logstash 是一款全托管的产品，因此我们首先需要在腾讯云 Logstash 控制台将 pem 文件以扩展文件方式进行上传，如下图所示。



2. 进入 **Logstash 管理** > **高级配置**界面，单击**本地上传**按钮。

3. 随后我们在管道管理 Tab 下，单击**新建管道**按钮，新建一个管道，在管道的 Config 配置编辑框里配置上 ES 的连接信息和证书路径。



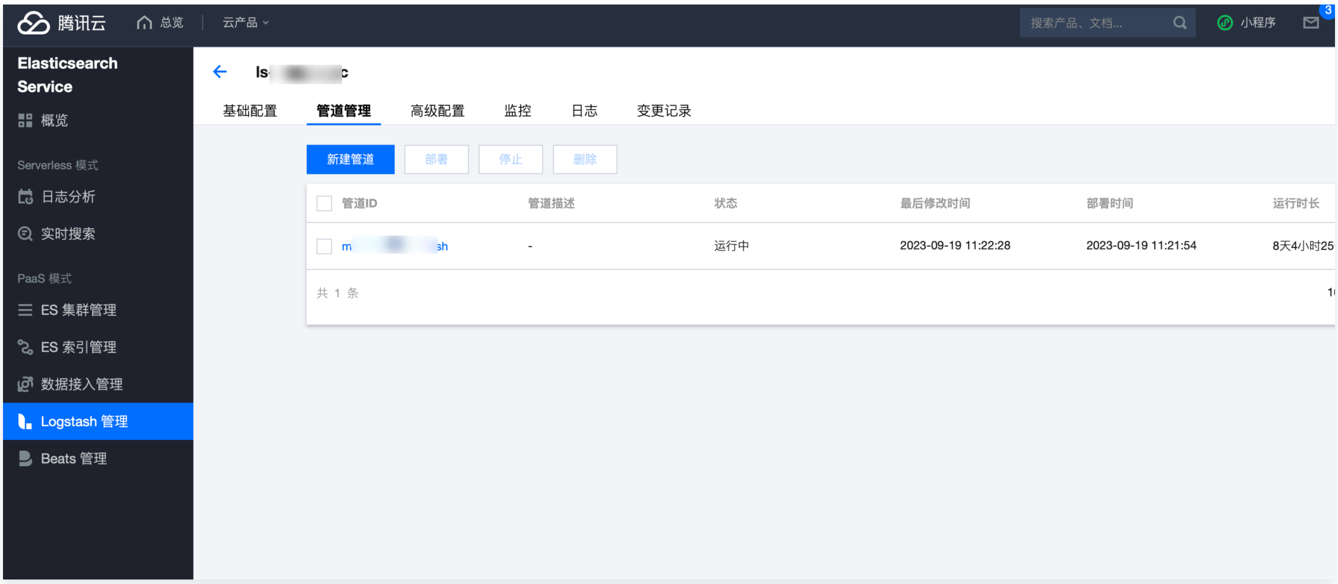
4. 管道详细配置文本信息如下：

```
output {
    elasticsearch {
        hosts => ["https://ES-VIP:9200"]
        user => "elastic"
        password => "changeme"
        ssl => true
```

```
        cacert => "/usr/local/service/logstash/extended-files/client-certificates.pem"
        ssl_certificate_verification => false
    }
}
```

   - 其中 cacert 即可我们上一步上传的扩展文件 pem 的路径，固定路径为：`/usr/local/service/logstash/extended-files/client-certificates.pem`。
   - 其他配置说明同 Metricbeat 一致。

5. 单击**保存**并部署管道后，即可在管道列表查看我们刚新建管道信息，此时状态为运行中。



6. 这时候我们再到 HTTPS 集群中就可以看到有数据从 input 的集群中写入进来了。

Kibana 连接 HTTPS 集群

1. 腾讯云 ES 集群默认自带 Kibana 访问能力，因此一般情况下，客户是不需要对 Kibana 进行任何配置的。这里介绍自建 Kibana 连接 HTTPS 集群的配置方式。和 Beats、Logstash 等使用的客户端鉴权证书不一样，Kibana 使用的是 server-certificates.pem，腾讯云 ES 侧在 CVM 中生成证书命令如下：

```
openssl pkcs12 -in ces-certificates.p12 -password pass:xxxxxx -nokeys -cacerts -out server-certificates.pem
```

2. 由于前面我们已经拿到了 pem 证书文件。因此，我们将 server-certificates.pem 文件拷贝到 Kibana 所在节点的如下路径：`/usr/local/service/https-certs`。然后修改 kibana.yml 配置文件如下：

```
elasticsearch.hosts: ["https://ES-VIP:9200"]
elasticsearch.username: "elastic"
elasticsearch.password: "changeme"
elasticsearch.ssl.verificationMode: certificate
elasticsearch.ssl.certificateAuthorities: ["/usr/local/service/https-certs/server-certificates.pem"]
xpack.encryptedSavedObjects.encryptionKey: "dfed624ca4014135f6180**********"
xpack.fleet.registryUrl: "https://epr.elastic.co"
```

3. 配置项说明：

- elasticsearch.ssl.certificateAuthorities： pem 文件的存放路径。
- elasticsearch.ssl.verificationMode： 证书鉴权模式，certificate 采用只鉴权 CA 证书，不鉴权主机名称的模式。
- xpack.fleet.registryUrl： （非必选配置）Fleet 集成模块需要访问的公网仓库，这里需要 Kibana 节点具备公网访问能力。
- xpack.encryptedSavedObjects.encryptionKey： value 可以通过Kibana上执行 `bin/kibana-encryption-keys generate` 命令获得。

```
[root@VM-48-212-centos /usr/local/service/kibana]# bin/kibana-encryption-keys generate
## Kibana Encryption Key Generation Utility

The 'generate' command guides you through the process of setting encryption keys for:

xpack.encryptedSavedObjects.encryptionKey
    Used to encrypt stored objects such as dashboards and visualizations
    https://www.elastic.co/guide/en/kibana/current/xpack-security-secure-saved-objects.html#xpack-security-secure-saved-ol

xpack.reporting.encryptionKey
    Used to encrypt saved reports
    https://www.elastic.co/guide/en/kibana/current/reporting-settings-kb.html#general-reporting-settings

xpack.security.encryptionKey
    Used to encrypt session information
    https://www.elastic.co/guide/en/kibana/current/security-settings-kb.html#security-session-and-cookie-settings


Already defined settings are ignored and can be regenerated using the --force flag.  Check the documentation links for in
.
Definitions should be set in the kibana.yml used configure Kibana.

Settings:
xpack.encryptedSavedObjects.encryptionKey: 1fb56afd8128f4da65ee056b839c7b9f
xpack.security.encryptionKey: 16027cc53a6f269857251b1d4720cea5

[root@VM-48-212-centos /usr/local/service/kibana]#
```

4. 重新启动 Kibana 后，即可正常访问 HTTPS 的 ES 集群了。

Java Client 连接 HTTPS 集群

1. 本文档演示在 Spring 项目中访问 HTTPS 集群的配置方式，首先需要添加如下 maven 依赖：

```
        <dependency>
            <groupId>org.elasticsearch.client</groupId>
            <artifactId>elasticsearch-rest-client</artifactId>
            <version>7.10.1</version>
        </dependency>
        <dependency>
            <groupId>org.elasticsearch.client</groupId>
            <artifactId>elasticsearch-rest-high-level-client</artifactId>
            <version>7.10.1</version>
        </dependency>
        <dependency>
            <groupId>org.elasticsearch</groupId>
            <artifactId>elasticsearch</artifactId>
            <version>7.10.1</version>
        </dependency>
```

2. 随后我们在 ElasticsearchConfig 自定义类中实现对 Elasticsearch 连接的配置。

```
@Configuration
public class ElasticsearchConfig {
    @Value("${es.username}")
    private String userName;
    @Value("${es.password}")
    private String password;
    @Value("${es.scheme}")
    private String scheme;
    @Value("${es.domain}")
    private String domain;
    @Value("${es.https.cert}")
    private String certFile;
    @Value("${es.port}")
    private int port;
    private RestHighLevelClient client;
    @Bean
    public RestHighLevelClient EsConnectInit() throws CertificateException, IOException, KeyStoreException,
NoSuchAlgorithmException, KeyManagementException {
```

```
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY,new
UsernamePasswordCredentials(userName,password));
        RestClientBuilder builder;
        if (scheme.equals("https")) {
            Path caCertificatePath = Paths.get(certFile);
            System.out.println(certFile);
            CertificateFactory factory = CertificateFactory.getInstance("X.509");
            Certificate trustedCa;
            try(InputStream is = Files.newInputStream(caCertificatePath)) {
                trustedCa = factory.generateCertificate(is);
            }
            KeyStore trustStore = KeyStore.getInstance("pkcs12");
            trustStore.load(null,null);
            trustStore.setCertificateEntry("ca",trustedCa);
            SSLContextBuilder sslContextBuilder = SSLContexts.custom().loadTrustMaterial(trustStore, new
TrustStrategy() {
                @Override
                public boolean isTrusted(X509Certificate[] x509Certificates, String s) throws
CertificateException {
                    return true;
                }
            });
            final SSLContext sslContext = sslContextBuilder.build();
            final HostnameVerifier hostnameVerifier = new HostnameVerifier() {
                public boolean verify(String hostname, SSLSession session) {
                    return true;
                }
            };
            builder = RestClient.builder(new HttpHost(domain, port, scheme)).setRequestConfigCallback(new
RestClientBuilder.RequestConfigCallback() {
                @Override
                public RequestConfig.Builder customizeRequestConfig(RequestConfig.Builder
requestConfigBuilder) {
                    requestConfigBuilder.setConnectTimeout(-1);
                    requestConfigBuilder.setSocketTimeout(-1);
                    requestConfigBuilder.setConnectionRequestTimeout(-1);
                    return requestConfigBuilder;
                }
            }).setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {
                @Override
                public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder httpClientBuilder)
{
                    httpClientBuilder.disableAuthCaching();
                    httpClientBuilder.setSSLHostnameVerifier(hostnameVerifier);
                    return httpClientBuilder.setSSLContext(sslContext)
                            .setDefaultCredentialsProvider(credentialsProvider);

                }
            });
        } else {
             builder = RestClient.builder(new HttpHost(domain, port, scheme))
                    .setRequestConfigCallback(new RestClientBuilder.RequestConfigCallback() {
                        @Override
                        public RequestConfig.Builder customizeRequestConfig(RequestConfig.Builder
requestConfigBuilder) {
                        requestConfigBuilder.setConnectTimeout(-1);
                            requestConfigBuilder.setSocketTimeout(-1);
                            requestConfigBuilder.setConnectionRequestTimeout(-1);
                            return requestConfigBuilder;
                        }
                    }).setHttpClientConfigCallback(new RestClientBuilder.HttpClientConfigCallback() {
```

```
                        @Override
                        public HttpAsyncClientBuilder customizeHttpClient(HttpAsyncClientBuilder
httpClientBuilder) {                            httpClientBuilder.disableAuthCaching();
                            return httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
                        }
                    });
        }
        client = new RestHighLevelClient(builder);
        return client;
    }
}
```

3. 其中 Elasticsearch 的配置信息如下：

```
es.scheme=https # 协议https或者http
es.port=9200 # ES集群VIP的端口
es.domain=9.10.1.X  # ES集群的VIP
es.username=elastic # ES集群的用户名
es.password=changeme # ES集群的密码
es.https.cert=/usr/local/services/certs/client-certificates.pem # 连接HTTPS集群所需的pem鉴权证书文件路径
```

通过如上配置后，即可在 Spring 项目中成功连接上开启了 HTTPS 协议的 ES 集群了。

十亿级高性能向量检索实践教程

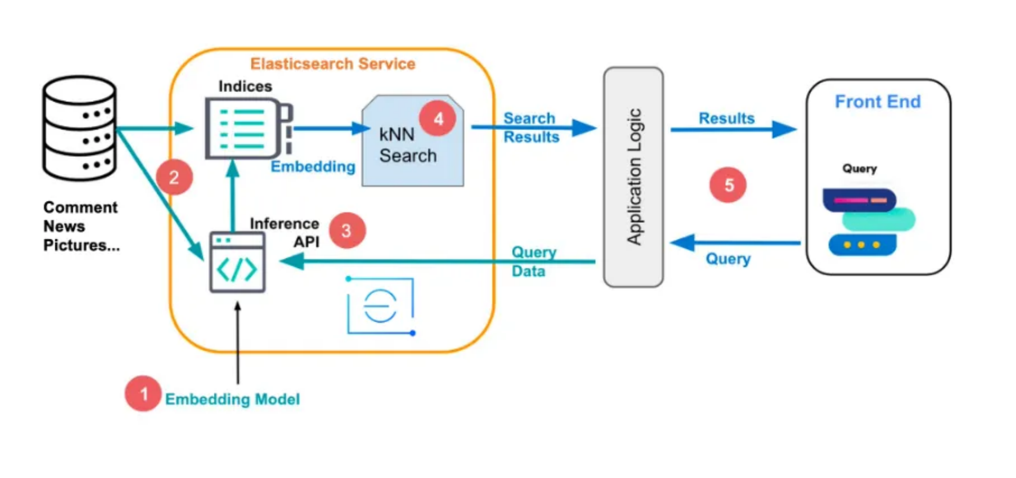
最近更新时间：2025-05-21 16:05:22

前言

Elasticsearch（ES）是一款基于 Lucene 的分布式搜索引擎，可以快速地存储、搜索和分析大量的数据。ES 以其高性能、可扩展性和丰富的功能使得它广泛应用于各种领域，例如企业搜索、日志分析、安全分析、商业智能、向量检索等。腾讯云大数据 Elasticsearch Service 是云端全托管 Elastic Stack 服务，集成 X-Pack 特性，并独有腾讯自研优化的内核特性。近期首发上线的 ES 8.16.1 版本，集成了 HNSW 算法来提高向量检索的计算速度，经过我们调优后，支持高达十亿级向量检索。本文旨在介绍如何使用集文本搜索、向量检索和 AI 能力于一身的 ES 8.16.1，为您带来搜索与分析全新的前沿体验。

总体架构

ES 8.16.1 提供的是一个从自然语言处理，到向量化，再到向量搜索，并能与大模型集成的端到端的搜索与分析平台。与 ES 7 相比，ES 8.16.1 引入了 HNSW 算法来给向量建立图索引，并且集成到 Lucene 的工作方式（倒排索引，BKD 树，缓存等）来优化检索性能。所以，ES 8.16.1 的向量搜索可以跟任何 ES 支持的任意查询相结合，并与聚合、文档级安全性、字段级安全性、索引排序等兼容，并同时兼具良好的性能体验。



总体流程简介如下：
向量化数据：用户可以上传提前训练好的自定义大数据模型，例如词嵌入模型（Word Embeddings）或深度学习模型（如 BERT），让 ES 做实时的读写 Embedding。业务也可以直接将向量数据上报到 ES。
索引向量数据：将向量化的数据存储到 ES 中。在索引过程中，将向量数据分布式地存储在多个节点上，以提高数据的可靠性和可扩展性。
发起向量检索请求：通过使用 ES 的搜索 API，发起向量检索请求。在请求中，需要指定待检索的向量以及其他相关参数，例如调用模型、相似度算法、返回结果数量等。
执行向量检索：ES 使用倒排索引来加速向量检索，快速定位包含特定向量的文档。在向量检索过程中， ES 会根据查询向量的特征，通过倒排索引匹配相似的向量。一旦匹配到倒排索引中的文档， ES 会计算查询向量与匹配文档向量之间的相似度。常用的相似度计算方法包括余弦相似度、欧几里得距离等。
返回检索结果：根据相似度计算的结果， ES 会返回与查询向量最相似的文档或结果。可以根据需求设置返回结果的数量和排序方式。同时，在召回后，可选择将 TOP 结果传入 LLM 大语言模型（例如混元、GPT）等，对信息进行对话式结果整合，最终返回给用户，实现对话式搜索。

操作实践

1. 在腾讯云 Elasticsearch Service 上申请 ES 8.16.1 集群。

2. **在 ES 中上传部署自定义 NLP 模型（可选）。**
   如果您需要提前将原始数据转化成向量，可以通过一些大数据模型执行推理任务，切割提取文本中的信息，将其转为向量。腾讯云 ES 8.16.1 最大的区别在于，支持用户无需重启集群即可本地上传提前训练好的自定义大数据模型，例如，从 HuggingFace 上下载的 transformer 模型，并在 Kibana 界面上管理和测试模型。最后通过在管道中集成不同的 Processor，灵活的处理数据。详情可参见 如何将本地 transformer 模型部署到 Elasticsearch 。
3. **在 ES 中创建向量索引结构。**
   命令范例以及一些参数介绍如下：

```
PUT my_vector_index
{
"mappings": {
   "properties": {
        "my_vector_field": {
           "type": "dense_vector",
            "dims": 768,              // 最高支持2048维度
            "element_type": "float",  // 支持 float, byte
            "index": true,
            "similarity": "cosine",   // 支持 cosine, dot_product, l2_norm
            "index_options": {        // hnsw 高级参数配置
                          "type": "int8_hnsw",
                          "m": 16,
                          "ef_construction": 100
            }
         }
       }
    }
}
```

4. **导入向量数据。**
   如果您的数据已经完成向量化数据，可以通过以下 ES Bulk/Index API 直接写入ES。

```
POST my_vector_index/_doc
{ "my_vector_field": [54, 10, -2],  "ip": "1.1.1.1", "name": "john" }
```

   如果您的数据需要做向量化，通过指定 ES Ingest Pipeline 配置之前上传的 NLP 模型，可以在实时写入过程中执行推理任务，可以随时添加、删除或更新新向量。
5. **进行向量检索**
   ES 8.16.1 提供的向量检索功能既可以进行单独向量检索，也可以支持跟任意的查询组合，实现高效预过滤、高效混合搜索。并且跟与聚合、文档级安全性、字段级安全性、索引排序等兼容，在执行向量搜索之后还能再进行 ES 聚合操作。

```
POST my_vector_index/_search
{
```



```
  "query": {                    // 可选，混合多路评分检索
      "match": {
        "name":  "john"
        }
     },
  "knn": {
    "field": "my_vector_field",
    "query_vector": [54, 10, -2],
    "k": 10,
    "num_candidates": 100,
    "query_vector_builder": {   // 可选，调用模型进行Embedding
    "text_embedding": {
      "model_id": "sentence-transformers__msmarco-minilm-l-12-v3",
      "model_text": "How is the weather in Jamaica?"
      }
    },
    "filter": {                 // 可选，前置过滤
      "term": {
        "file-type": "png"
      }
    }
}
"size": 100
}
```

ES 的数据是以分片的形式散落在多台机器节点上的。向量搜索的流程如下：

1. 先在每个分片上查找 num_candidates 个近似最近邻候选者向量。
2. 计算这些候选向量与查询向量的相似度，从每个分片中选择 k 个最相似的结果。
3. 协调节点会归并每个分片的结果，以返回全局前 size 最终结果返回给用户。

所以，用户可以增加 num_candidates 以提升查询结果的准度，相应的，查询耗时也会增加，时间换取准度。如果您的查询还未做向量化，也可以在查询语句中指定已经上传好的大模型进行向量化再做查询。

优化实践

我们在支持客户的大型在线搜索业务的接入过程中，业务持续导入了超过亿条向量，并做了最终性能压测，ES 8.16.1 轻松扛住了最高十亿级向量，并且可以做到查询结果平均响应时延在毫秒级。下面我们会具体介绍该业务接入的优化实践。

1. **使用段合并大幅提升向量检索性能。**

   ES 的写入模型采用的是类似 LSM-Tree 的存储结构。ES 实时写入的数据都先放在 Lucene 内存 buffer 中，同时依赖写入 translog 保证数据的可靠性。当积攒到一定程度后，将他们批量写入一个新的段（Segment）。ES 对向量也是采用相同的处理方式，所以可以做到向量实时的插入跟检索。但这种方式，也会产生较多的小段，不同段之间的向量无法连接起来，导致查询效率低下。ES 会在集群运行过程中缓慢地持续进行底层段的 Merge，同时我们也建议业务在低峰期可以定期调度一些段合并（forcemerge）任务以减少底层段数量，实测至少可以提升向量检索性能3 - 5倍以上。
   其压测采用的配置为：6台高 IO 型 SSD 节点，16核64G，亿级向量，96维。最终压测结果如下：

是否做段合并优化	优化前(30个段)	优化后(2个段)
查询 qps	622	3112
查询平均延迟	307 ms	81 ms

   需要注意的是 forcemerge 需要全量重新构建图索引，对资源占用比较高，建议可以在业务低峰期进行。ES 集群支持横向无限拓展，业务如果性能不满足还可以通过横向增加节点跟分片数提升 qps。

2. **使用 ES 8.16.1 同时调大 Merge 线程池，以提升 forcemerge 效率。**

   ES 8.16.1 相对于之前的版本，Lucene 对合并 HNSW 图进行了重大改进，会尽可能复用现有最大的 HNSW 图。因此，Lucene 不再像以前那样从一个空图开始，而是利用之前完成的所有工作来构建现有最大的段。合并较大的段时，这个改进带来的提升是巨大的。在基准测试中，段合并时间减少了40%。腾讯云 ES 自研内核新增合并策略，集群闲时自动 Merge 清理超过阈值的段，不影响集群整体性能的同时降低存储成本、提升查询性能。如果用户需要做forcemerge 的话，可以根据机器核数配置，调大 forcemerge 线程池配置来加速 forcemerge 过程。

   ```
   thread_pool.force_merge.size: 8          // 初始默认值为2，比较小
   ```

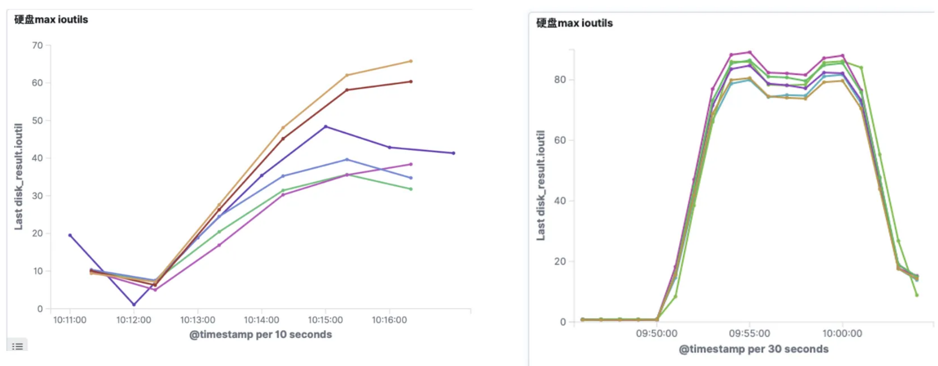
   云 ES 控制台上也支持用户设置定时任务做 forcemerge。

3. **尽量分配足够的 RAM 和高 IO 的硬盘。**

   ES 将向量数据存储在磁盘上，同时使用倒排索引来加速向量查询跟匹配，所以ES 将允许数据集大于本地主机上可用的 RAM 总量，但是随着页面缓存中可容纳的 HNSW 数据比例的降低，性能将会下降。注重性能的用户需要根据数据集的大小来调整 RAM 大小，同时选用高 IO 的硬盘，例如 SSD，以维持良好的查询性能。

4. **分片数、ES 查询参数调优。**

   用户在性能压测过程中，遇到了压测负载上不去，负载不均等问题，我们通过对分片数、分发等 ES 查询参数进行调优，解决了这些问题。



优化前：负载不均衡且资源利用率上不去　　优化后：负载均衡且资源接近用满

5. **召回率测试。**

   我们基于不同 ef_construction 参数，跟暴力计算相对比来计算召回率，结果如下。

efConstruction	100	200	500
top100 召回率	>=96%	=100%	=100%
top200 召回率	>=94.5%	>99.9%	=100%
top500 召回率	>=90%	>=99.2%	>99%

总结

腾讯云大数据 Elasticsearch Service 近期首发上线的 ES 8.16.1 版本，提供了强大的云端 AI 增强与向量检索能力，支持在端到端搜索与分析平台中实现自然语言处理、向量搜索以及与大模型的集成，10亿级向量检索平均响应延迟控制在毫秒级，助力客户实现由 AI 驱动的高级搜索能力，为搜索与分析带来全新的前沿体验。

腾讯云 ES Serverless x TKE，分钟级低门槛实现一套可观测日志分析

最近更新时间：2025-01-07 11:44:12

引言

作为云原生技术的爱好者，笔者活跃在各种开发者社群，最近，腾讯云 Elasticsearch Serverless 凭借其自动弹性、免运维的特性，丰富的产品能力，以及能分钟级实现容器服务的日志采集与可观测分析的能力，深深吸引到了笔者。为了让您也能免费体验到这款开箱即用的好产品，笔者再三争取，最终领取到了免费体验券 + 资源包特惠1元购 + 开发者课程等福利，具体领取方式可见文末相关链接。

云原生与容器

随着容器、Serverless、微服务等技术快速发展，云原生已逐步构建出繁荣的技术体系。如今云原生凭借降本增效、提高持续交付能力、易于开发等优势，正在不断激活应用构建范式，引起企业系统架构、生产方式、商业模式等发生变革，毋庸置疑，云原生已成为企业数字化转型的最短路径。
腾讯云容器服务（TKE）是基于原生 kubernetes 提供以容器为核心的、高度可扩展的高性能容器管理服务，本文将基于腾讯云 ES Serverless 服务，构建 TKE 容器服务日志分析体系，实现分钟级落地，帮助企业轻量化构建容器服务可观测分析的同时实现降本提效。

腾讯云 ES Serverless 介绍

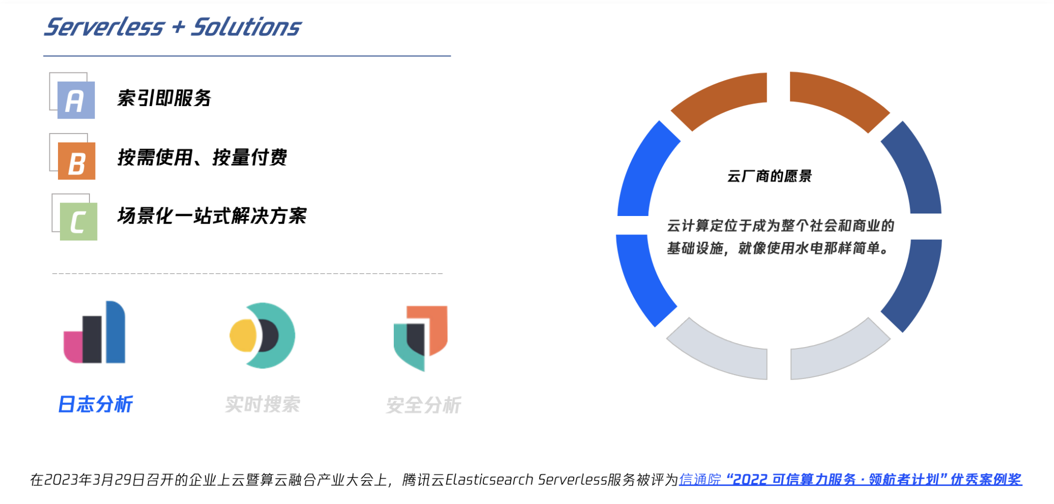
笔者认真学习了腾讯云《ES Serverless 一站式日志分析从入门到精通》课程，简单而言，该产品的设计理念、产品能力与优势如下：

设计理念

让算力像自来水一样按需使用一直都是各大云厂商的目标，Serverless 概念的设计思想提出，极大的影响了近几年云计算的发展。在云计算的发展初期，侧重的是能够快速平滑迁移上云，例如，把 IDC 资源换成云服务器以及云硬盘等。而现在，客户的需求已经从上好云逐步到转变到用好云，即在释放运维与管理成本的同时，能够更加聚焦于业务，提高效率，实现降本增效。
为了帮助企业能够更好的聚焦于业务，ES Serverless 服务在设计开发中始终坚持以下目标：

1. **索引即服务**，无集群概念，可按需创建与使用索引。
2. **按需使用**，按量付费，按照实际使用的计算与存储资源按量计费。
3. 不止索引，还是**场景化的一站式解决方案**，针对每个场景提供相应的开箱即用的能力，提升整体的使用效率，当前已支持**日志分析**场景。

值得一提的是，在2023年3月29日召开的企业上云暨算云融合产业大会上，腾讯云 ES Serverless 服务被评为「信通院 “2022 可信算力服务 · 领航者计划” 优秀案例奖」。



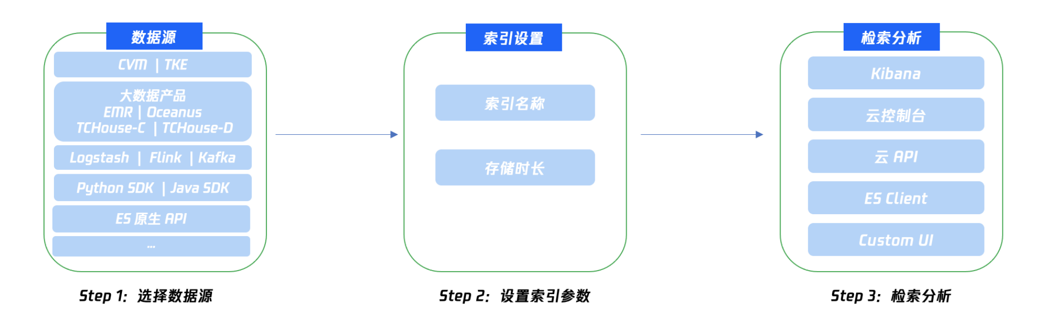
设计思路

从设计思路上，ES Serverless 服务坚持以下几个点：

1. 提升易用性，除了提供开箱即用的一站式场景解决方案外，还实现了集群免运维、索引免运维以及数据链路免运维。
2. 降低成本，基于自研存算分离技术，实现自动伸缩，去除冗余成本，同时，完全按照使用量计费。
3. 提升稳定性，集群、索引等由平台统一运维调优，避免因使用不当造成的故障。
4. 完全兼容原生 ES 生态与 API，实现无缝迁移。

产品能力

从产品架构上，面向日志场景，ES Serverless 服务提供**自动弹性、完全免运维**的一站式日志分析解决方案，在使用时，用户仅需关注数据源以及检索分析等业务逻辑即可，中间的流量调度、链路调度以及资源调度全部由 ES Serverless 服务完成，不需要再关注底层的数据链路、消息队列、集群运维以及索引配置等，平台提供端到端的 SLA 保障。
除了支持原生的 ES API 写入方式之外，控制台已支持**云服务器 CVM、容器服务 TKE、EMR、腾讯云数据仓库 TCHouse** 等云产品的一站式数据采集分析，同时也支持通过 Logstash、Flink 以及 Kafka 等将数据投递到 ES Serverless 服务的索引中。简单而言，只需3步，我们即可完成一站式日志分析场景的落地。
在索引管理方面，则提供了配置管理、指标监控、用户管理以及告警管理等能力，可方便我们快速完成数据应用。而在检索分析能力方面，除了支持原生的Kibana 能力，ES Serverless 服务也在控制台内嵌了 Kibana 的相关核心能力，无需外链访问，即可快速进行检索分析。



产品优势

自动弹性：索引粒度的自动弹性伸缩，从容应对突发流量增长，在保证业务连续性的同时降低日志分析、可观测性等场景下业务波峰波谷导致的运维与管理成本高等问题 。
完全免运维：内置分片自动调优、智能生命周期管理以及故障自愈等能力，用户可按需创建与使用索引，无需关心底层的资源配置、集群扩缩容和索引设置等问题，整个使用过程完全免运维。
灵活易用：提供端到端从数据接入、到数据管理、再到数据分析探索的一站式产品能力，极大降低业务上云门槛，可在分钟级实现业务落地。
成本低廉：自研低成本、高性能、高可用的存算分离架构，根据实际访问与存储量计费，实现业务负载与资源动态匹配的按需付费，减少闲置资源导致的冗余成本支出，大幅降低成本。
开放集成：完全兼容 Elastic Stack 生态，保留用户原有使用习惯，实现无缝迁移，助力业务快速上云；同时，打通云上数据源（如云服务器 CVM、容器服务 TKE），降低数据接入门槛，实现分钟级业务落地。
稳定可靠：集群配置、读写性能由后台统一优化，减少由于使用不当带来的故障问题，提升稳定性，为业务保驾护航。

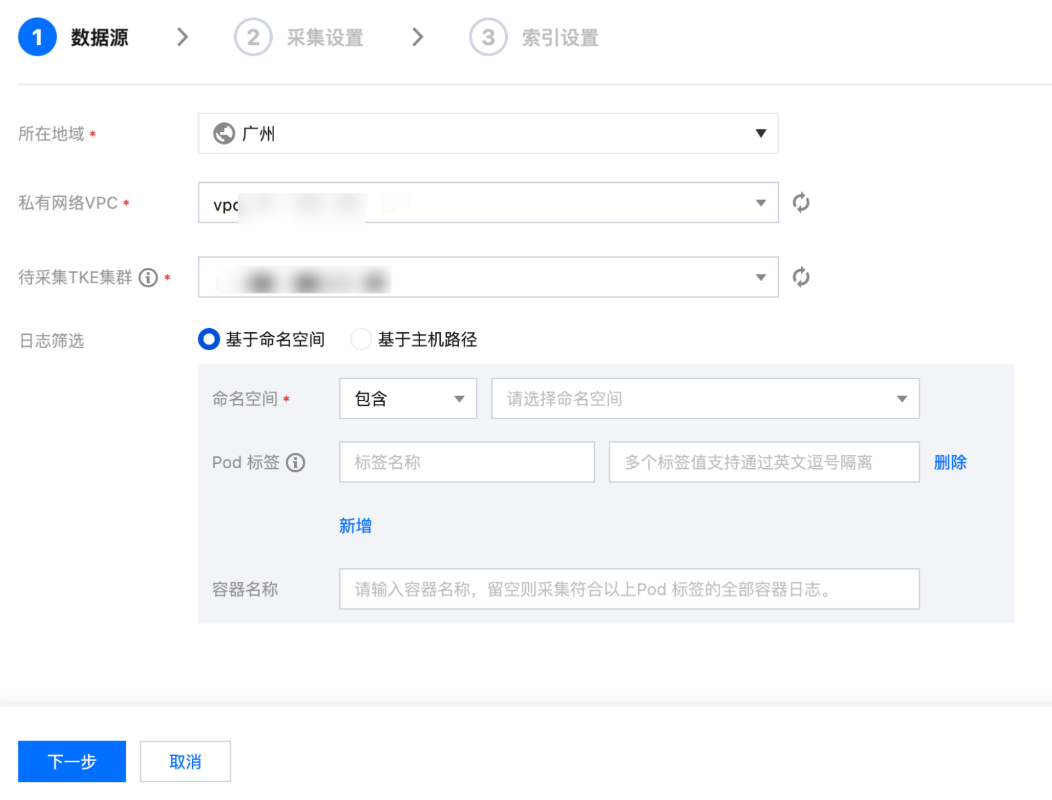
日志分析实践

日志采集

ES Serverless 服务目前已在控制台提供了一站式的容器服务日志采集，只需简单的选择需采集的容器集群，设置相关采集参数，同时设置数据保存时长等，即可快速完成 TKE 日志接入。参考如下：

1. 在 ES Serverless 控制台选择**容器服务 TKE**：

2. 选择 TKE 集群，目前支持基于命名空间、Pod 标签、容器名称或是主机路径进行日志筛选：



3. 如需对上报到的日志进行解析，可设置采集解析，当前已支持全文日志、JSON 格式以及分隔符解析，如有更丰富的解析需求，也可自行编写 Processors，实现自定义解析。此处我们选择**全文日志**：

解析模式	描述
全文日志	日志数据不做任何键值提取，日志内容将存储到名称为"message"的字段中，您可通过自动分词能力进行检索分析。
JSON	对于内容为标准 JSON 格式的日志，按照日志中的 Key：Value 提取对应的字段。
分隔符	对于固定分隔符分割内容的日志，我按照指定分隔符对日志进行键值提取，分隔符支持单个字符或者是字符串，可在控制台选择或输入分隔符。

4. 设置索引名称以及数据存储时长，同时需设置时间字段，该字段指在实际数据中类型为 date 的字段，通过设置该字段，可有助于提升查询效率。

通过以上简单几步，我们即可快速将容器日志采集到 ES Serverless 的索引中，单击**确认创建**，等待约5 - 10秒即可。

报表建设

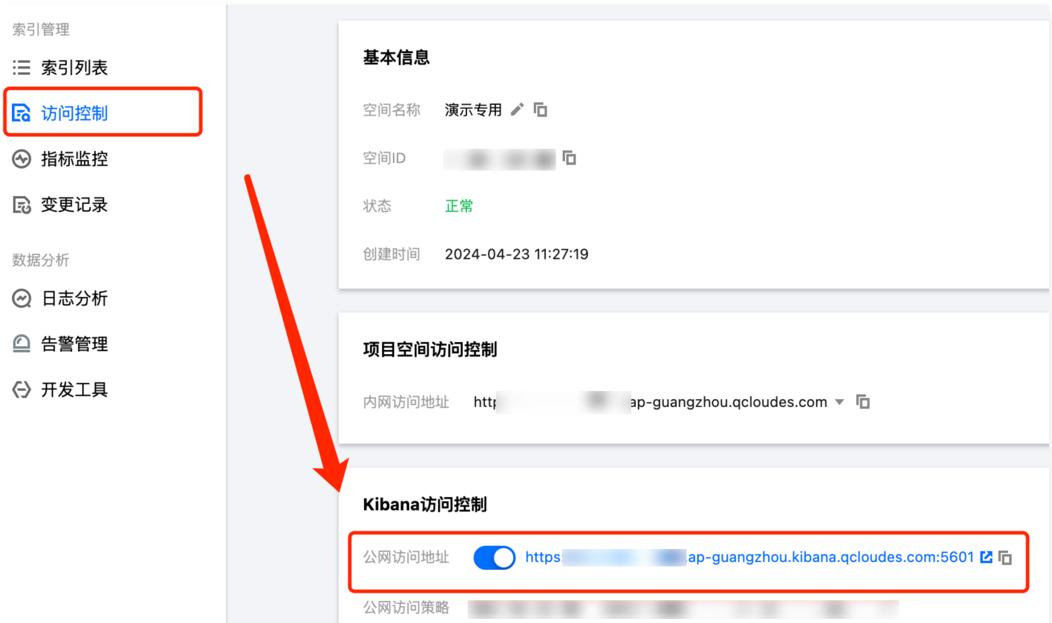
ES Serverless 目前已在控制台集成了日志检索、开发工具以及监控告警等能力，可帮助我们快速实现检索分析。作为二流“资深”ES 使用者，笔者习惯使用 Kibana 进行分析，方便通过外链进行分享交流。

监控告警

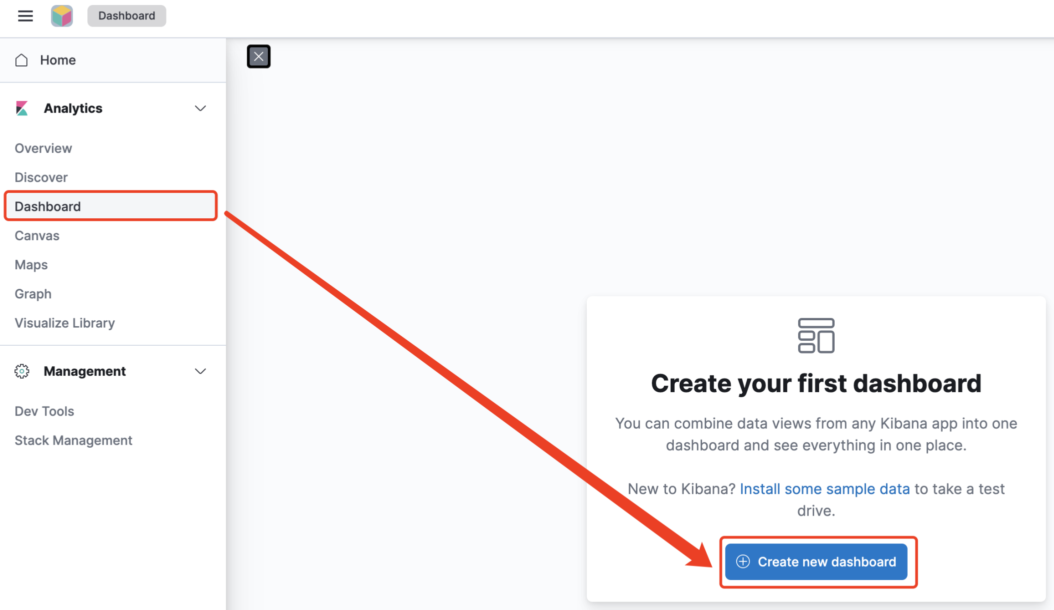
为了在错误日志出现时能及时发现与处理，我们可先在控制台配置监控告警，例如，通过统计五分钟内来自 /var/log/error.log 路径的日志条数，快速识别异常，当超出阈值时，告警到我们的邮箱与企业微信。

可观测分析

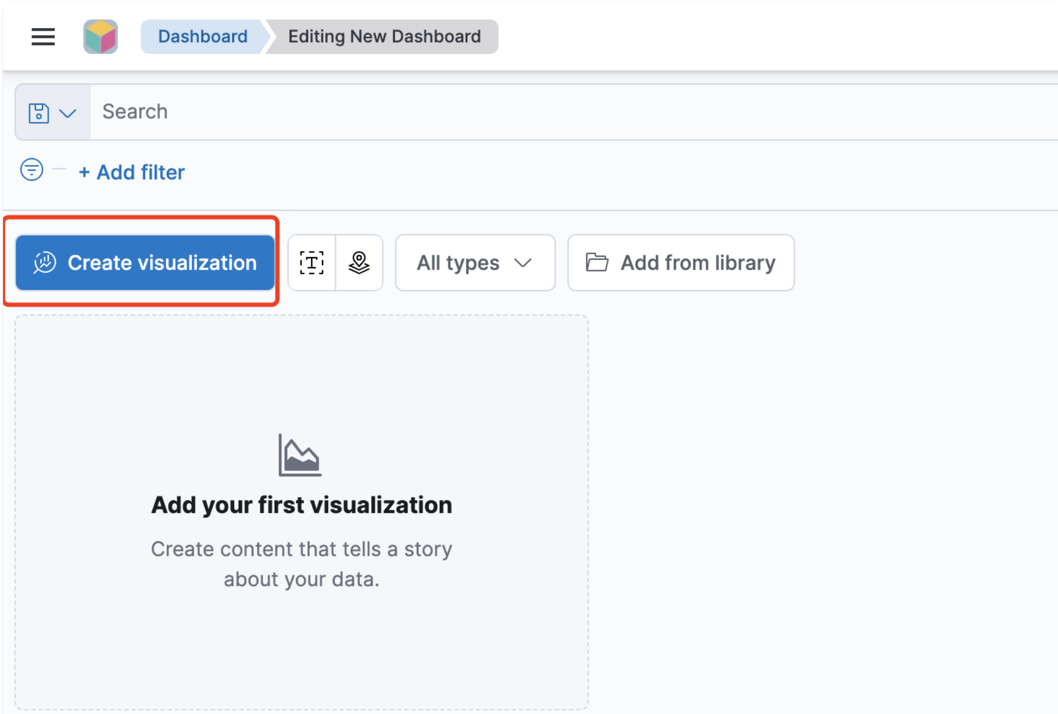
1. 首先登录 Kibana，我们找到 Kibana 的公网访问地址，为确保公网访问安全性，在访问前，需将我们的 IP 地址设置到公网访问策略中。



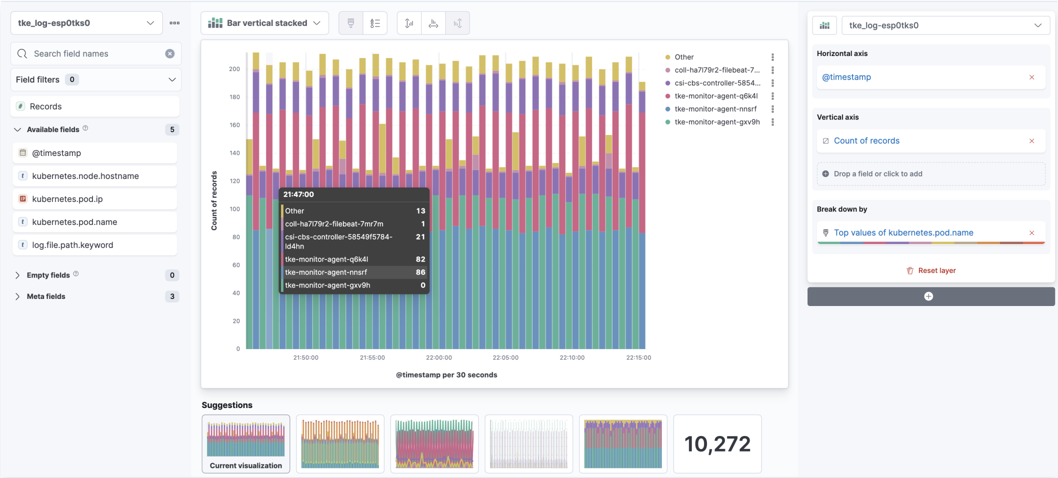
2. 单击 Dashboard，我们可快速创建图表：



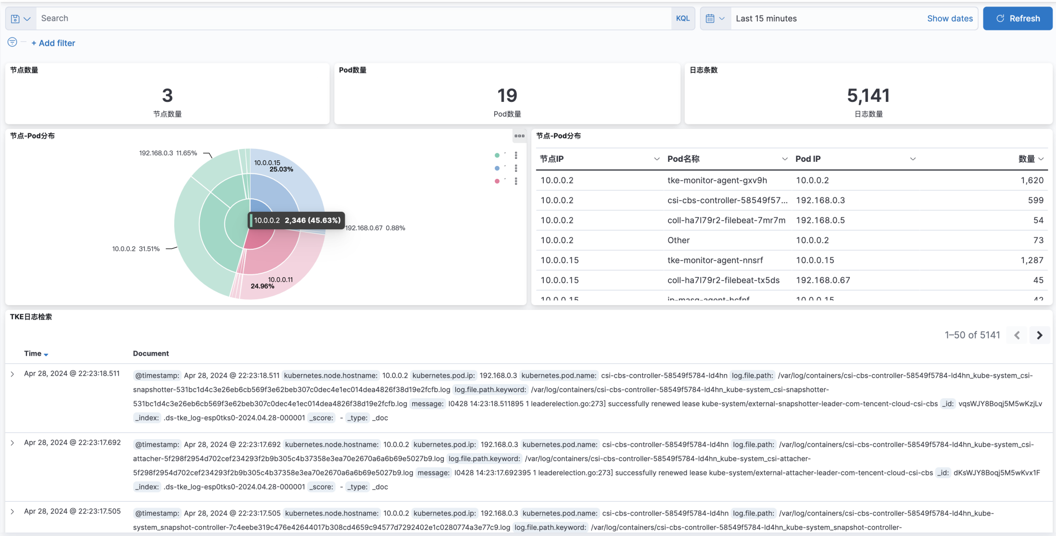
3. 单击 **Create visualization**：



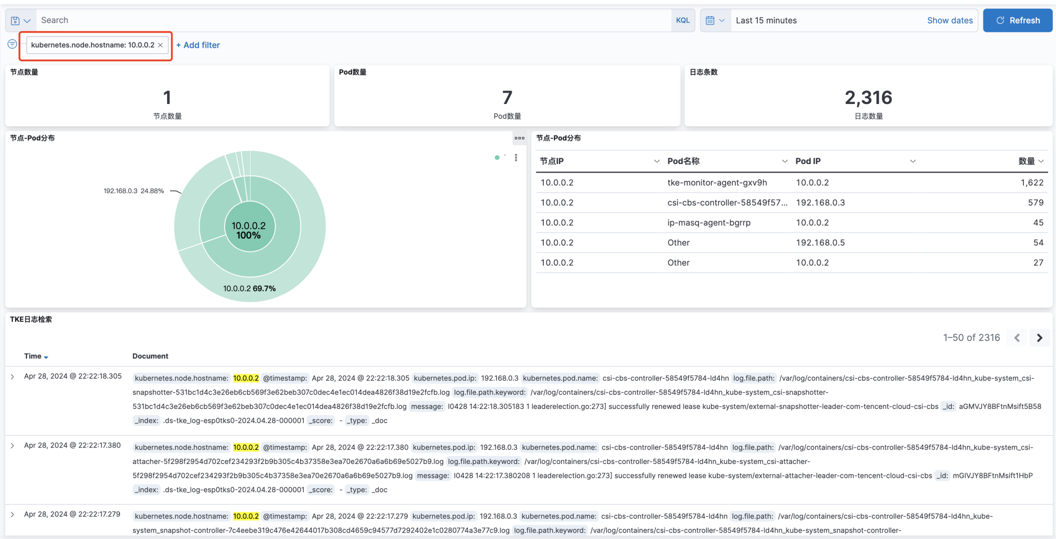
4. Kibana 的 Lens 功能非常易用跟强大，仅需将光标移动到左侧的字段，并拖拽到中间，即可生成图表，同时下方还提供了其他图表建议，方便我们快速进行分析。如下图，我们将 **@timestamp** 以及 **pod.name** 两个字段拖拽到了中间，Lens 快速生成了柱状图，通过该图，我们可快速了解每个时间段每个 Pod 上报的日志数量。



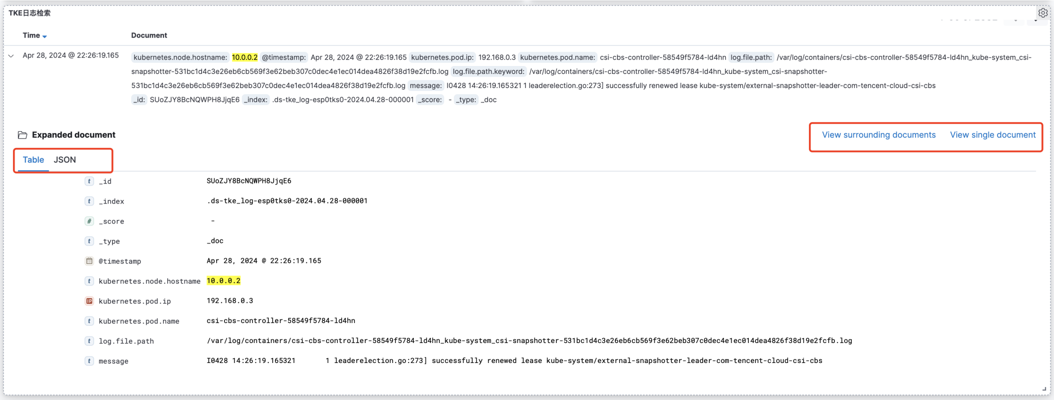
4.1 首先第一行，是各种总览数据，如节点数量、Pod 数量以及日志条数；第二行是节点-Pod分布；第三行则是具体的日志数据。即整体按照**总览 > 多维分布 > 日志层层下钻**。



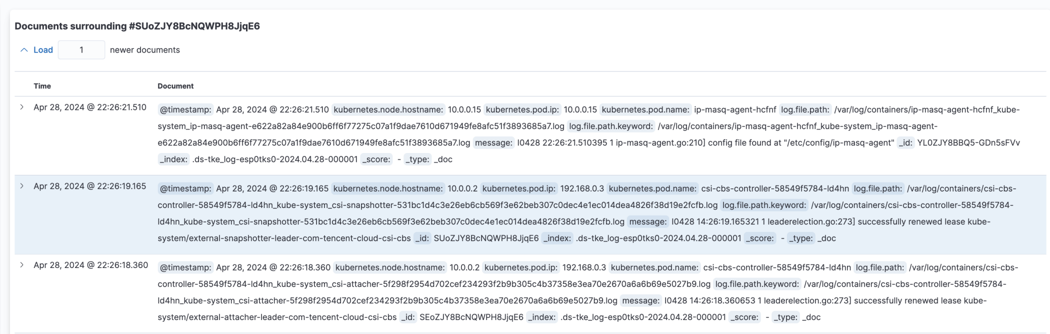
4.2 从上图中，我们看到10.0.0.2这个节点的日志数量占比将近一半，那我们可以逮着它来分析，只需在上图的**节点-Pod** 多维分析图中，单击该节点所在的区域，即可自动生成筛选条件，将该节点相关的数据全部筛选出来并在日志中高亮。



4.3 当想改变日志展示风格时，下方支持了表单模式跟 JSON 模式，任君选择；当发现当前日志发生的问题可能需要上下文才能辅助分析，单击 **view surrounding documents** 即可。



4.4 例如，查看该日志的前一条与后一条日志：



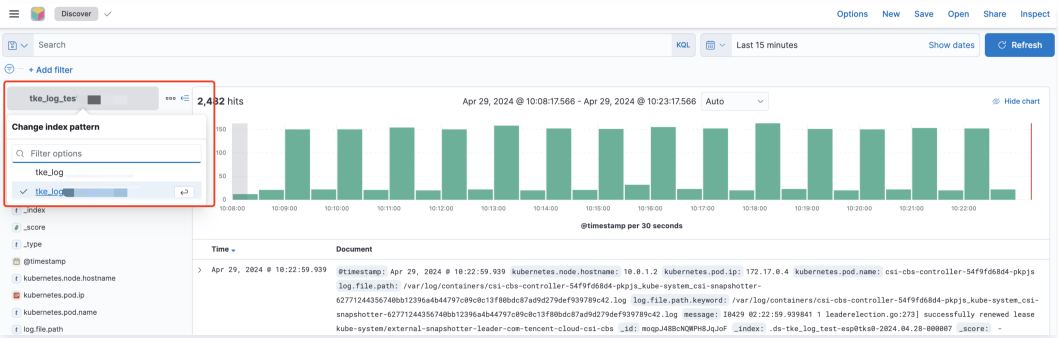
权限控制

当然，在实际业务应用场景中，我们难免需要将权限分享给团队的其他同事，这时候，权限管控的就显得尤为重要，例如对于 A 同事，仅分配查看某些索引的读权限，而对于 B 同事，则支持读写所有索引。

关于这个场景的能力，ES Serverless 也充分为用户考虑，进入 ES 控制台，在用户管理页面，单击**新建用户**，即可开始创建用户。

权限类型目前支持**只读与读写**，权限范围包含所有索引以及指定索引，填写完成后，单击**确定**即可。

例如，我设置 **hd_man** 用户仅对 TKE 相关的索引有只读权限，那么设置完成后，我通过该用户登录 Kibana，即只能查看到 TKE 日志相关的索引：



总结

本文简单介绍了基于腾讯云 ES Serverless 服务实现一站式容器服务可观测分析，在实践中，笔者只花了几分钟，轻松完成了日志接入与可观测分析报表建设，从这点上来说，ES Serverless 服务真正做到了轻量、零门槛。当然，看了这么多，是时候展示笔者跪求小编拿到的优惠了，链接如下：

- 免费体验券与资源包1元购：

腾讯云大数据 Elasticsearch Serverless 服务首发上线

- 外部开发者课程：

ES Serverless 一站式日志分析入门到精通学习课程_ES Serverless 一站式日志分析入门到精通视频教程-腾讯云开发者社区

如对该产品感兴趣，也可扫码加入交流群，ES Serverless 服务的产品小哥非常 nice～

