Elasticsearch Service

开发教程

产品文档

腾讯云

文档目录

开发教程
数据迁移

最近更新时间：2020-01-09 10:58:46

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，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：

- elasticsearch-dump
- snapshot
- reindex
- logstash

elasticsearch-dump

适用场景

适用于数据量不大，迁移索引个数不多的场景。

使用方式

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

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

   ```
   npm install elasticdump -g
   ```

2. 主要参数说明

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

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

   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
```

4. 迁移所有索引

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

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
```

snapshot

适用场景

适用数据量大的场景。

使用方式

snapshot api 是 ES 用于对数据进行备份和恢复的一组 api 接口，可以通过 snapshot api 进行跨集群的数据迁移，原理就是从源 ES 集群创建数据快照，然后在目标 ES 集群中进行恢复。

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

1. 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，安装插件 repository-s3。
   - hdfs：快照存放于 hdfs 中，以插件形式支持，安装插件 repository-hdfs。
   - cos：快照存放于腾讯云 COS 对象存储中，以插件形式支持，安装插件 elasticsearch-repository-cos。

   从自建 ES 集群迁移至腾讯云 ES 集群，可直接使用 fs 类型仓库，但需要在 ES 配置文件 elasticsearch.yml 中设置仓库路径。

   ```
   path.repo: ["/usr/local/services/test"]
   ```

   然后调用 snapshot api 创建 repository。

   ```
   curl -XPUT http://172.16.0.39:9200/_snapshot/my_backup -H 'Content-Type: application/json' -d '{
   "type": "fs",
   "settings": {
   "location": "/usr/local/services/test"
   "compress": true
   }
   }'
   ```

   从其它云厂商的 ES 集群迁移至腾讯云 ES 集群，或腾讯云内部的 ES 集群迁移，可使用对应云厂商提供的仓库类型，例如 AWS 的 S3、阿里云的 OSS 和腾讯云的 COS 等。

   ```
   curl -XPUT http://172.16.0.39:9200/_snapshot/my_s3_repository
   {
   "type": "s3",
   "settings": {
   "bucket": "my_bucket_name",
   "region": "us-west"
   }
   }
   ```

2. 源 ES 集群中创建 snapshot
   调用 snapshot api 在创建好的仓库中创建快照。创建快照可以指定索引，也可以指定快照中包含的内容，具体的

api 接口参数可以查阅 官方文档。

```
curl -XPUT http://172.16.0.39:9200/_snapshot/my_backup/snapshot_1?wait_for_completion=true
```

3. 目标 ES 集群中创建 repository
   目标 ES 集群中创建仓库和在源 ES 集群中创建仓库类似，用户可在腾讯云上创建 COS 对象 bucket，把仓库建在 COS 的某个 bucket 下。

4. 移动源 ES 集群 snapshot 至目标 ES 集群的仓库
   把源 ES 集群创建好的 snapshot 上传至目标 ES 集群创建好的仓库中。

5. 从快照恢复

```
curl -XPUT http://172.16.0.20:9200/_snapshot/my_backup/snapshot_1/_restore
```

6. 查看快照恢复状态

```
curl http://172.16.0.20:9200/_snapshot/_status
```

reindex

reindex 是 ES 提供的一个 api 接口，可以把数据从源 ES 集群导入到当前 ES 集群，实现数据的迁移。但仅限于 ES 的实现方式，当前版本不支持 reindex 操作。下面简单介绍 reindex 接口的使用方法：

1. 配置 reindex.remote.whitelist 参数
   需要在目标 ES 集群中配置该参数，指明能够 reindex 的远程集群的白名单。
2. 调用 reindex api
   以下操作表示从源 ES 集群中查询名为 test1 的索引，查询条件为 title 字段为 elasticsearch，将结果写入当前集群的 test2 索引。

```
POST _reindex
{
"source": {
"remote": {
"host": "http://172.16.0.39:9200"
},
"index": "test1",
"query": {
```

```
"match": {
"title": "elasticsearch"
}
}
},
"dest": {
"index": "test2"
}
}
```

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，具体的配置文件如下：

```
input {
elasticsearch {
hosts => ["http://172.16.0.39:9200"]
index => "*"
docinfo => true
}
}
output {
elasticsearch {
hosts => ["http://172.16.0.20:9200"]
index => "%{[@metadata][_index]}"
}
}
```

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，logstash 的更多功能可查阅 logstash 官方文档。

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无法实现迁移。而使用 snapshot 的方式则没有这个限制，因为 snapshot 方式是完全离线的。因此 elasticsearch-dump 和 logstash 迁移方式更适合于源 ES 集群和目标 ES 集群处于同一网络的情况下进行迁移。而需要跨云厂商的迁移，可以选择使用 snapshot 的方式进行迁移，例如从阿里云 ES 集群迁移至腾讯云 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. elasticsearchdump 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：2020-06-01 15:22:43

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 beat）接入自己的数据。

本文以官方的 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
```

> 说明：
> 请注意 logstash 版本，建议与 Elasticsearch 版本保持一致。

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

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

Docker 中访问 ES 集群

自建 Docker 集群

1. 拉取 logstash 官方镜像。

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

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

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

使用腾讯云容器服务

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。

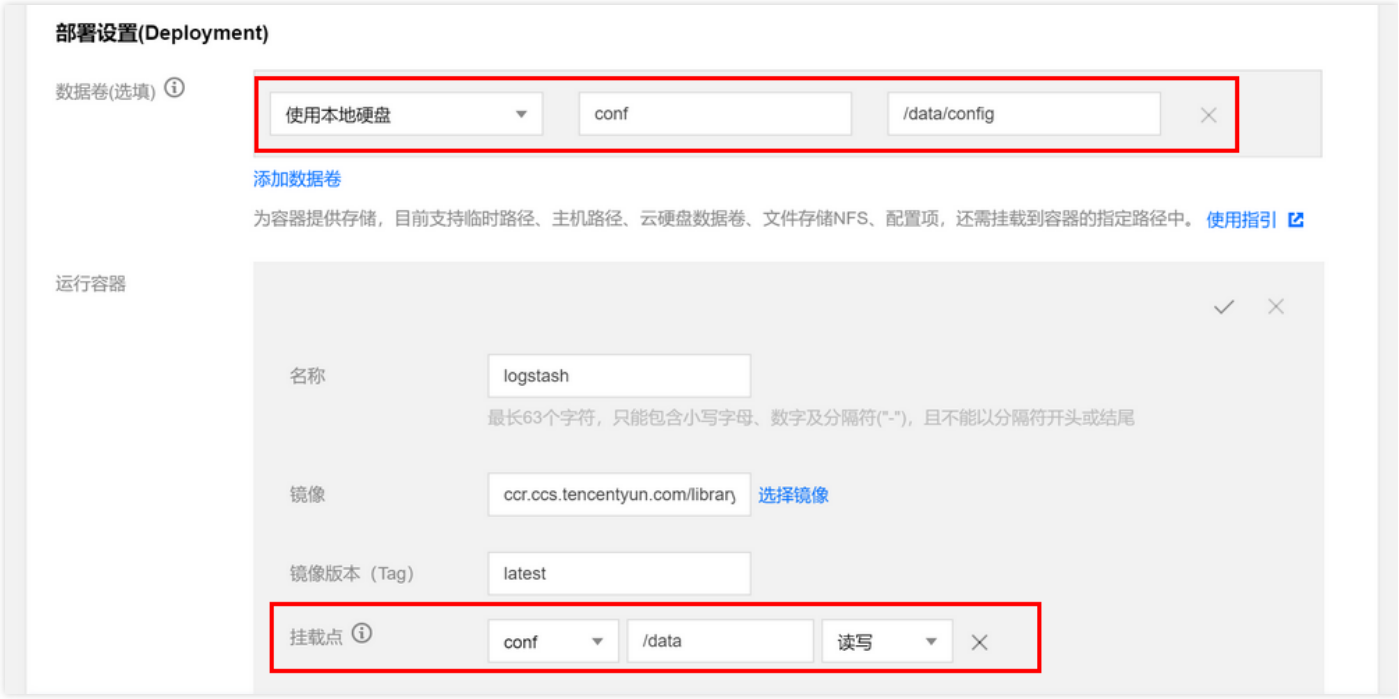
2. 选择左侧菜单栏【服务】，单击【新建】创建服务。

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。


部署设置(Deployment)
数据卷(选填)
使用本地硬盘
conf
/data/config
添加数据卷
为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷、文件存储NFS、配置项，还需挂载到容器的指定路径中。使用指引
运行容器
名称
logstash
最长63个字符，只能包含小写字母、数字及分隔符("-")，且不能以分隔符开头或结尾
镜像
ccr.ccs.tencentyun.com/library
选择镜像
镜像版本（Tag）
latest
挂载点
conf
/data
读写

5. 配置运行参数。


工作目录
指定容器运行后的工作目录，查看详情
运行命令
控制容器运行的输入命令，查看详情
运行参数
-fdata/logstash.conf
传递给容器运行命令的输入参数，查看详情

6. 根据需要配置服务参数并创建服务。

配置文件说明

File 数据源

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

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

Kafka 数据源

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

```
}

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

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

JDBC 连接的数据库数据源

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

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

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

使用 Beats 接入 ES 集群

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

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

CVM 中访问 ES 集群

1. 安装部署 filebeat。

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

2. 配置 filebeat.yml。
3. 执行 filebeat。

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

Docker 中访问 ES 集群

自建 Docker 集群

1. 拉取 filebeat 官方镜像。

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

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

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

使用腾讯云容器服务

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"]
```

构建日志分析系统

最近更新时间：2020-06-01 15:27:46

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"]
protocol: "http"
username: "elastic"
password: "test"
```

3. 执行 Filebeat

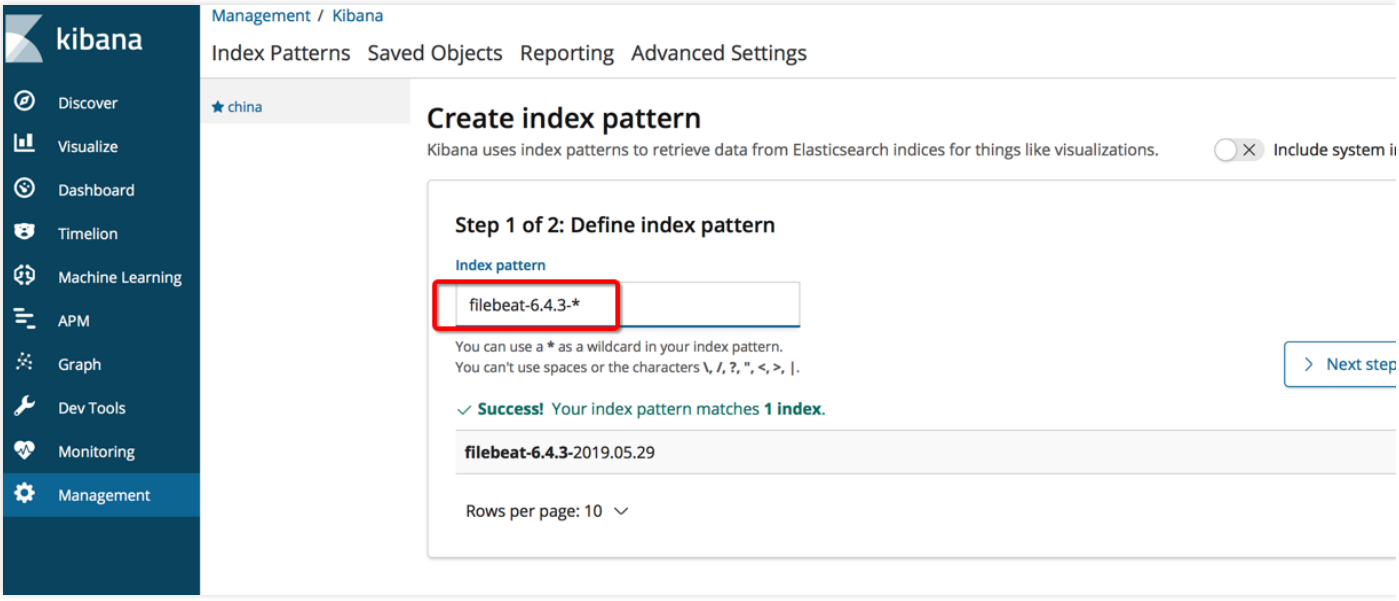
在 filebeat-6.4.3-linux-x86_64 目录中，执行：

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

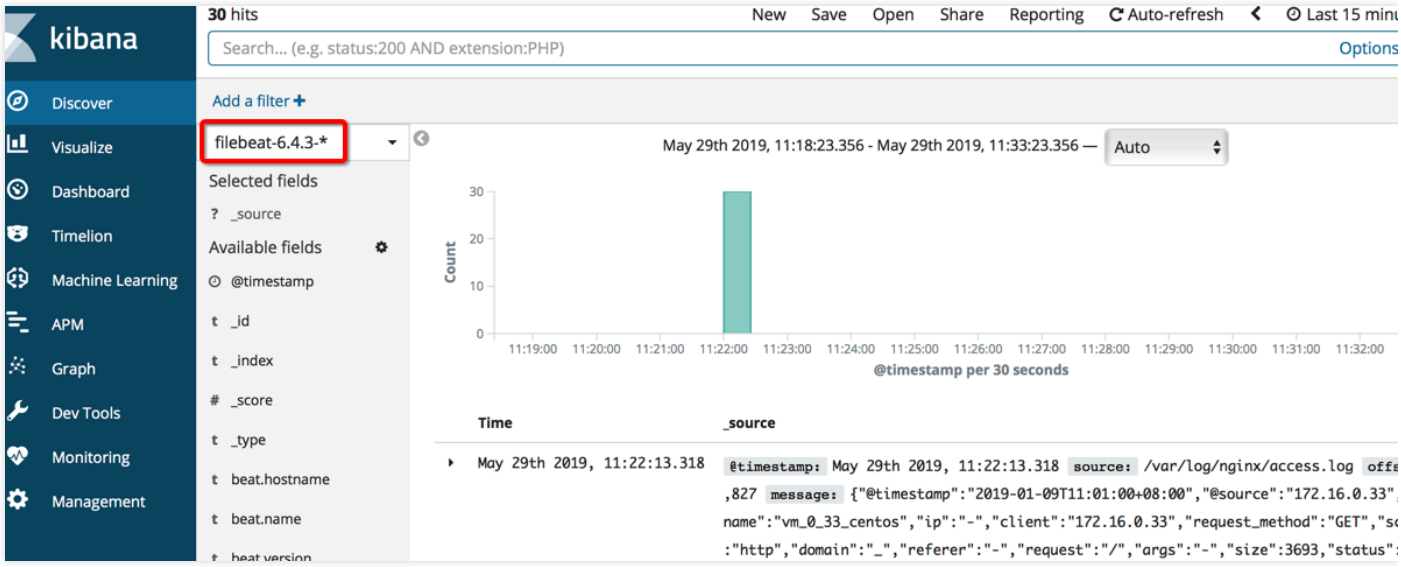
查询日志

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/, :p
```

```
ath=>"/"}
[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_d883144359d3b4f516b37dba51fab2a2", :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，请查看 elastic 官方文档。

查询日志

可参考 查询日志。

更多有关 Kibana 控制台的功能，请查看 elastic 官方文档。

使用 Curator 管理索引

最近更新时间：2020-03-10 15:40:08

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函数的使用方法可以参考官网 文档。

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。

```
curator_cli --config PATH
```

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无服务器云函数的定时触发功能。

使用 API

Python API 的使用可参考 文档。

MySQL 数据实时同步到 ES

最近更新时间：2020-05-14 16:18:37

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

mysql binlog 日志

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

mysql binlog 日志有三种模式，分别为：

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

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

- ROW 模式的 binlog 日志内容示例如下：

```
SET TIMESTAMP=1527917394/*!*/;
BEGIN
/*!*/;
# at 3751
#180602 13:29:54 server id 1 end_log_pos 3819 CRC32 0x8dabdf01 Table_map: `webservice`.`buildin
g` mapped to number 74
# at 3819
#180602 13:29:54 server id 1 end_log_pos 3949 CRC32 0x59a8ed85 Update_rows: table id 74 flags: S
TMT_END_F
BINLOG '
UisSWxMBAAAARAAAAOsOAAAAAEoAAAAAAAEACndlYnNlcnZpY2UACGJ1aWxkaW5nAAYIDwEP
EREG
wACAAQAAAAHfq40=
UisSWx8BAAAAggAAAG0PAAAAAEoAAAAAAAEAAgAG///A1gcAAAAAAAALYnVpbGRpbmctMTAA
DwB3
UkRNbjNLYlV5d1k3ajVbD64WWw+uFsDWBwAAAAAAAAtidWlsZGluZy0xMAEPAHdSRE1uM0tiVXl3
WTdqNVsPrhZbD64Whe2oWQ==
```

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

- STATEMENT 模式下 binlog 日志内容示例如下：

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

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

mysqldump 工具

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

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

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

的文件内容如下：

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

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

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

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

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

LOCK TABLES `building` WRITE;
/*!40000 ALTER TABLE `building` DISABLE KEYS */;
```

```
INSERT INTO `building` VALUES (2000,'building-2',0,'6YFcmntKrNBIeTA','2018-05-30 13:28:31','2018-05-30 13:28:31'),(2001,'building-4',0,'4rY8PcVUZB1vtrL','2018-05-30 13:28:34','2018-05-30 13:28:34'),(2002,'building-5',0,'uyjHVUYrg9KeGqi','2018-05-30 13:28:37','2018-05-30 13:28:37'),(2003,'building-7',0,'DNhyEBO4XEkXpgW','2018-05-30 13:28:40','2018-05-30 13:28:40'),(2004,'building-1',0,'TmtYX6ZC0RNB4Re','2018-05-30 13:28:43','2018-05-30 13:28:43'),(2005,'building-6',0,'t8YQcjeXefWpcyU','2018-05-30 13:28:49','2018-05-30 13:28:49'),(2006,'building-10',0,'WozgBc2IchNyKyE','2018-05-30 13:28:55','2018-05-30 13:28:55'),(2007,'building-3',0,'yJk27cmLOVQLHf1','2018-05-30 13:28:58','2018-05-30 13:28:58'),(2008,'building-9',0,'RSbjotAh8tymfxs','2018-05-30 13:29:04','2018-05-30 13:29:04'),(2009,'building-8',0,'IBOMlhaXV6k226m','2018-05-30 13:29:31','2018-05-30 13:29:31');
/*!40000 ALTER TABLE `building` ENABLE KEYS */;
UNLOCK TABLES;

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

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

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

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

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

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

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

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

使用限制

1. mysql binlog 必须是 ROW 模式（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，否则直接忽略。如果数据表没有主键，UPDATE 和 DELETE 操作就会因为在 ES 中找不到对应的 document 而无法进行同步。

3. 不支持程序运行过程中修改表结构。
4. 要赋予用于连接 MySQL 的账户 RELOAD 权限、REPLICATION 权限。

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

使用方式

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

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

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

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

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

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

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

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

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

```
# pseudo server id like a slave
server_id = 1001

# mysql or mariadb
flavor = "mariadb"

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

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

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

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

# Ignore table without primary key
skip_no_pk_table = false

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

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

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

8. 在控制台输出结果。

```
2018/06/02 16:13:21 INFO create BinlogSyncer with config {1001 mariadb 172.16.0.101 3306 bellen utf8 false false <nil> false false 0 0s 0s 0}
```

```
2018/06/02 16:13:21 INFO run status http server 127.0.0.1:12800
2018/06/02 16:13:21 INFO skip dump, use last binlog replication pos (mysql-bin.000001, 120) or GTI
D %!s(<nil>)
2018/06/02 16:13:21 INFO begin to sync binlog from position (mysql-bin.000001, 120)
2018/06/02 16:13:21 INFO register slave for master server 172.16.0.101:3306
2018/06/02 16:13:21 INFO start sync binlog at binlog file (mysql-bin.000001, 120)
2018/06/02 16:13:21 INFO rotate to (mysql-bin.000001, 120)
2018/06/02 16:13:21 INFO rotate binlog to (mysql-bin.000001, 120)
2018/06/02 16:13:21 INFO save position (mysql-bin.000001, 120)
```

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

使用体验

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

使用 mypipe 同步数据到 ES 集群

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

使用限制

1. mysql binlog 必须是 ROW 模式。
2. 要赋予用于连接 MySQL 的账户 REPLICATION 权限。

```
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'elastic'@'%' IDENTIFIED BY 'Elas
tic_123'
```

3. mypipe 只是将 binlog 日志内容解析后编码成 Avro 格式推送到 kafka broker，并不是将数据推送到 kafka。如果需要同步到 ES 集群，可以从 kafka 消费数据后，再写入 ES。
4. 消费 kafka 中的消息（MySQL 的操作日志），需要对消息内容进行 Avro 解析，获取到对应的数据操作内容，进行下一步处理。mypipe 封装了一个 `KafkaGenericMutationAvroConsumer` 类，可以直接继承该类使用，或者自行解析。
5. mypipe 只支持 binlog 同步，不支持存量数据同步，即 mypipe 程序启动后无法对 MySQL 中已经存在的数据进行同步。

使用方式

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

2. 执行命令 `./sbt package` 。

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

```
mypipe {

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

# consumers represent sources for mysql binary logs
consumers {

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

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

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

# pipes join consumers and producers
pipes {

kafka-generic {
enabled = true
consumers = ["localhost"]
producer {
kafka-generic {
metadata-brokers = "172.16.16.22:9092"
}
}
binlog-position-repo {
# saved to a file, this is the default if unspecified
class = "mypipe.api.repo.ConfigurableFileBasedBinaryLogPositionRepository"
config {
file-prefix = "stdout-00" # required if binlog-position-repo is specifiec
```

```
data-dir = "/tmp/mypipe/data" # defaults to mypipe.data-dir if not present
}
}
}
}
}
```

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

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

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

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

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

8. 消费到没有经过解析的数据如下：

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

使用体验

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

腾讯云 ES+SCF 快速构建搜索服务

最近更新时间：2020-06-05 15:19:00

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（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。具体可参考 节点类型存储配置建议。

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。

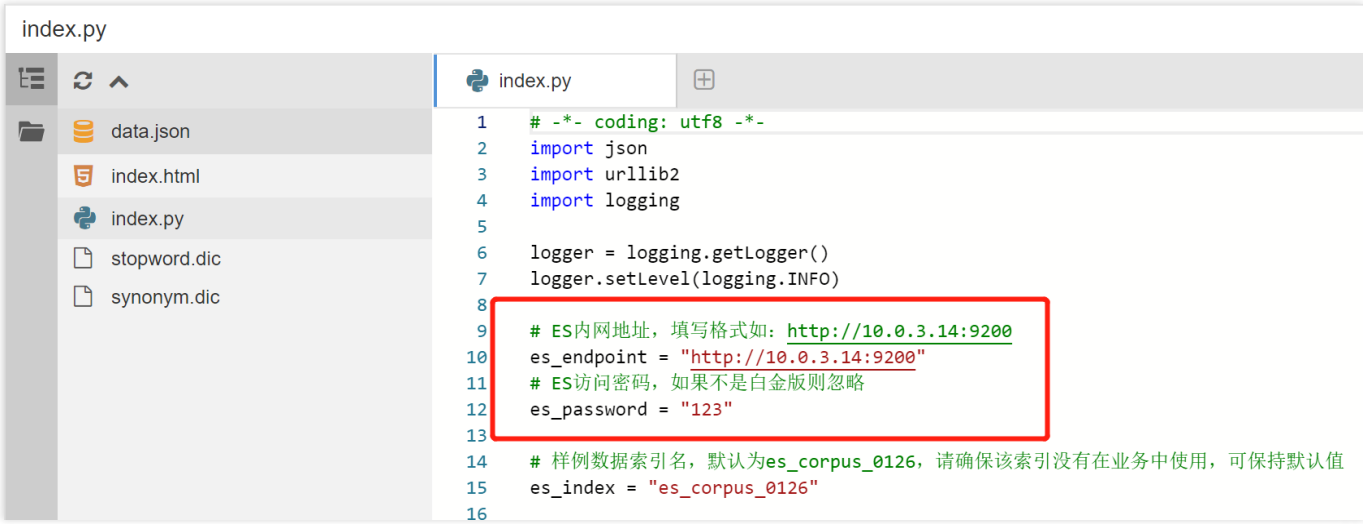
1. 在【云函数】>【函数服务】界面左上角首先选择您购买 ES 集群的地域。

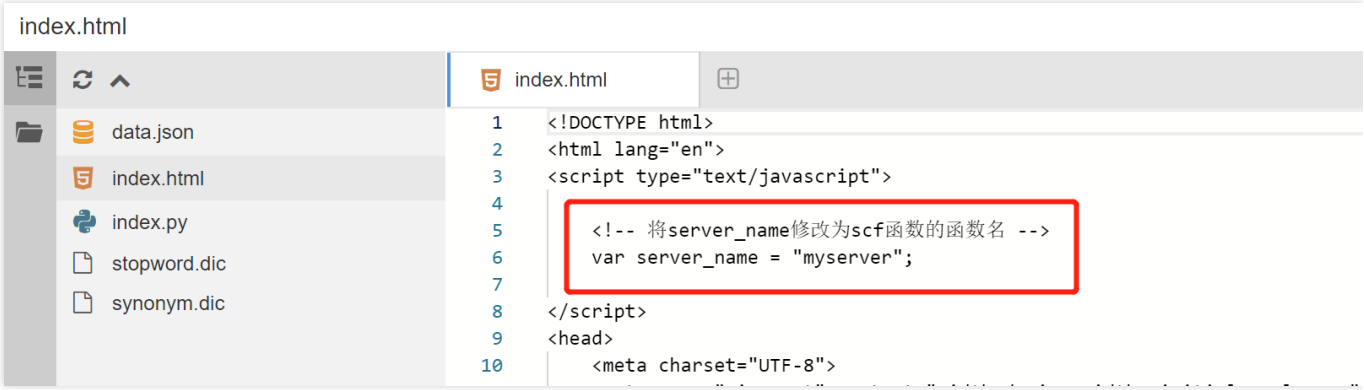
2. 新建一个函数服务，**记住您设置的函数名称**，然后选择【下一步】>【完成】。

3. 在**函数配置**页单击右上角的【编辑】，开启**内网访问**，并选择您创建 ES 所选的 VPC，然后单击【保存】。

4. 首先将 代码 zip 包 下载到本地。然后在**函数代码**页的**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**触发方式**页单击【添加触发方式】，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**触发方式**中看到函数的"访问路径"，单击此路径即可访问页面。

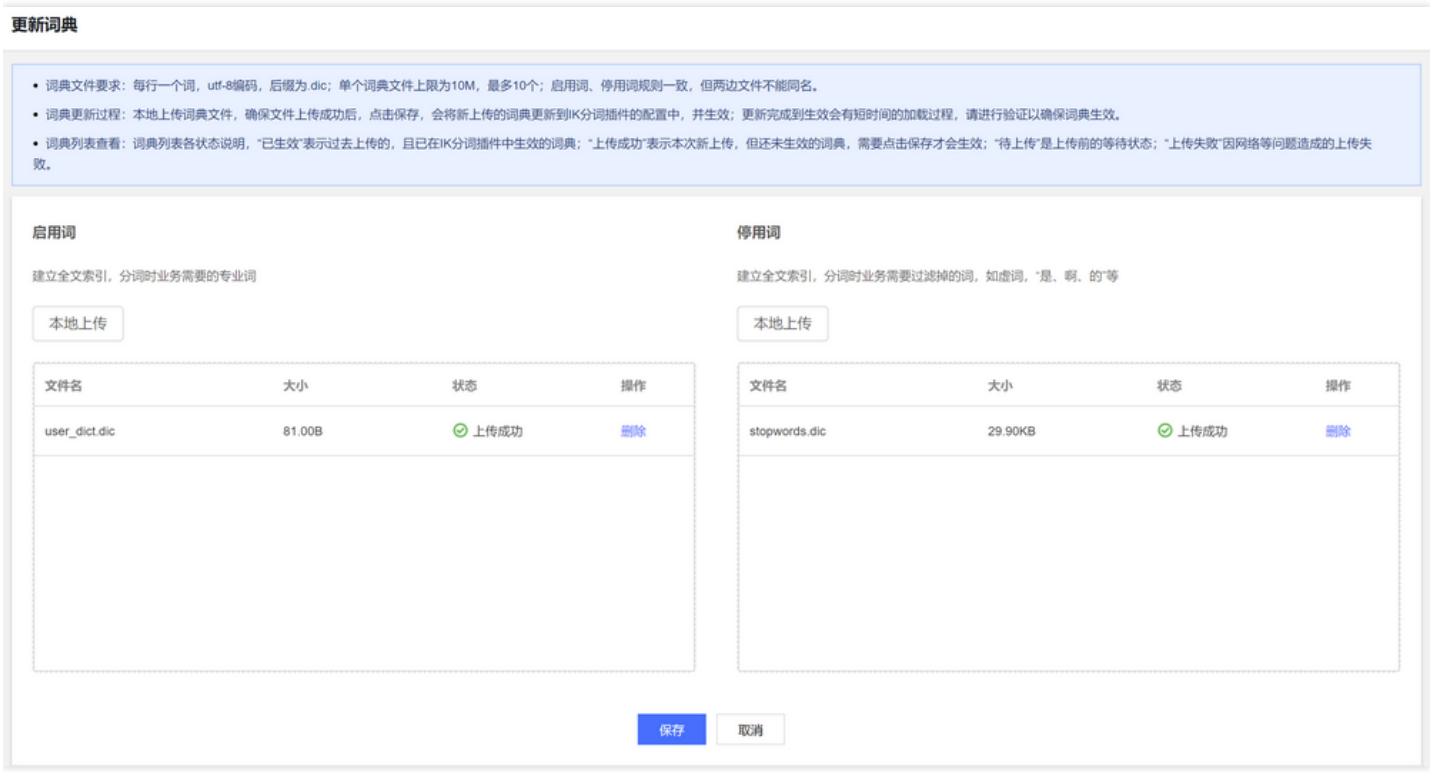
8. 上传 腾讯云 ES 官方文档 样例数据。单击搜索框上方的文字，自动导入数据。

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，用户词典在分词的时候将保留该词。在上面的案例中，我们导入了默认的 停用词库 和 用户词典，您也可以通过 ES 集群详情页的【高级配置】>【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。