

**Cloud Search Service**

# **Best Practices**

**Issue**            01  
**Date**             2023-03-29



**Copyright © Huawei Technologies Co., Ltd. 2023. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

---

# Contents

---

<b>1 Cluster Migration.....</b>	<b>1</b>
1.1 Migration Solution Overview.....	1
1.2 Migration from Elasticsearch.....	3
1.2.1 Migrating Cluster Data Using Logstash.....	3
1.2.2 Migrating Cluster Data Through Backup and Restoration.....	5
1.3 Migration from Kafka/MQ.....	7
1.4 Migration from a Database.....	8
<b>2 Cluster Access.....</b>	<b>10</b>
2.1 Overview.....	10
2.2 Accessing a Cluster Using cURL Commands.....	11
2.3 Accessing a Cluster Using Java.....	13
2.3.1 Accessing a Cluster Through the Rest High Level Client.....	13
2.3.2 Accessing a Cluster Through the Rest Low Level Client.....	21
2.3.3 Accessing the Cluster Through the Transport Client.....	36
2.4 Accessing a Cluster Using Python.....	37
2.5 Using ES-Hadoop to Read and Write Data in Elasticsearch Through Hive.....	39
<b>3 Cluster Performance Tuning.....</b>	<b>46</b>
3.1 Optimizing Write Performance.....	46
3.2 Optimizing Query Performance.....	49
<b>4 Practices.....</b>	<b>52</b>
4.1 Using CSS to Accelerate Database Query and Analysis.....	52
4.2 Using CSS to Build a Unified Log Management Platform.....	56
4.3 Configuring Query Scoring in an Elasticsearch Cluster.....	60

# 1 Cluster Migration

---

## 1.1 Migration Solution Overview

You can migrate data to a Huawei Cloud Elasticsearch cluster from another Huawei Cloud Elasticsearch cluster, a user-built Elasticsearch cluster, or a third-party Elasticsearch cluster. This section describes the solutions for data migration from different clusters.

### Scenarios

The migration solution varies depending on the data source.

- Migration from an Elasticsearch cluster  
You can use Logstash, CDM, OBS backup and restoration, ESM, or cross-cluster replication plug-ins to migrate data in an Elasticsearch cluster.
  - Logstash: an official data cleaning tool provided by Elasticsearch. It is a part of the Elk ecosystem and provides powerful functions. It can migrate data between different data sources and Elasticsearch, and clean and process data. For details, see [Migrating Cluster Data Using Logstash](#).
  - CDM: a cloud migration tool provided by Huawei Cloud to implement cluster migration between different cloud services. For details, see .
  - Backup and restoration: Elasticsearch provides backup and restoration capabilities. You can back up the data of a cluster to OBS, and restore the data to another cluster. For details, see [Migrating Cluster Data Through Backup and Restoration](#).
- [Migration from Kafka/MQ](#)
- [Migration from a Database](#)

### Solutions

CSS supports migration by backup and restoration, by using the Reindex API or Logstash+ESM, or by data source synchronization. For details, see [Table 1-1](#).

Data source synchronization has fewer constraints and higher performance than the other three solutions. Data source synchronization allows cutover anytime after the synchronization completed, which is more convenient and flexible.

**Table 1-1** Migration solutions

Solution	Description	Constraint	Performance
Backup and restoration	Prepare shared storage that supports the S3 protocol, for example, an OBS bucket. Create a snapshot to back up the data of the source Elasticsearch cluster, synchronize the snapshot to the target cluster, and restore data to the target cluster.	<ul style="list-style-type: none"> <li>Target Elasticsearch version <math>\geq</math> Source Elasticsearch version</li> <li>Number of candidate master nodes of the target Elasticsearch cluster <math>&gt;</math> Half of the number of candidate master nodes of the source Elasticsearch cluster</li> <li>Incremental data synchronization is not supported. You need to stop update before backing up or restoring data.</li> </ul>	The data migration rate is configurable. Ideally, the data migration rate is the same as the file copy rate.
Reindex API	Configure mutual trust between the source and target Elasticsearch clusters, and then migrate data using the Reindex API.	<ul style="list-style-type: none"> <li><b>_source</b> must be enabled for indexes.</li> <li>Real-time synchronization of incremental data is not supported. You need to stop the update and then call the API.</li> </ul>	Batch read and write are supported, but concurrent slicing synchronization is not supported.
Logstash +ESM	Apply for an ECS, deploy and configure Logstash on it, and then start data migration.	<ul style="list-style-type: none"> <li><b>_source</b> must be enabled for indexes.</li> <li>Real-time synchronization of incremental data is not supported. You need to stop the update and then start Logstash.</li> </ul>	Batch read and write are supported, and concurrent slicing synchronization is supported.

Solution	Description	Constraint	Performance
Data source synchronization	Inventory data is migrated using Logstash, and incremental data is automatically synchronized through traffic replication or data links.	None	The inventory migration rate is the same as that of Logstash. An existing tool is reused for incremental migration.

## 1.2 Migration from Elasticsearch

### 1.2.1 Migrating Cluster Data Using Logstash

Logstash is an official data migration tool provided by Elasticsearch.

**Step 1** Apply for an ECS, preferably with at least 8 vCPUs and 16 GB memory.

**Step 2** Install Logstash on the ECS.

1. Install JDK, because Logstash depends on Java. Run the following command to install JDK using **yum**:

```
yum install java
yum install python
```

2. Download Logstash. Choose a Logstash version close to the Elasticsearch version. They do not have to use exactly the same version.

Logstash 7.10.2 OSS is recommended. You can download it from <https://www.elastic.co/downloads/past-releases/logstash-oss-7-10-2>

3. Run the following command to install Logstash using **yum**:

```
yum install logstash-oss-7.10.0-x86_64.rpm
```

Replace *logstash-oss-7.10.0-x86\_64.rpm* with the actual Logstash installation package name.

**Step 3** Modify the JVM configuration of Logstash to improve the cluster data migration efficiency.

Run the following command to modify the JVM configuration. The default heap memory of Logstash is 1 GB. You are advised to change the heap memory to half of the cluster node memory.

```
vim /etc/logstash/jvm.options
-Xms4g
-Xmx4g
```

**Step 4** Modify the **conf** configuration file of Logstash and configure cluster migration settings.

1. Go to the **/etc/logstash/conf.d/** directory where the Logstash configuration file is stored.

```
cd /etc/logstash/conf.d/
```

2. Create the **logstash-es-es-all.conf** file.

```
vim logstash-es-es-all.conf
```

3. Add the following content to the **logstash-es-es-all.conf** file and save the file.

Modify the **hosts**, **user**, **password**, **index** fields as needed.

```
input{
  elasticsearch{
    #IP address of the source cluster
    hosts => ["http://172.16.xxx.xxx:9200", "http://172.16.xxx.xxx:9200"]
    # #For a security cluster, configure the username and password for cluster login. For a non-
security cluster, you can use the number sign (#) to comment out the user and password fields.
    # user => "xxxx"
    # password => "xxxx"
    # #List of indexes to be migrated. Multiple indexes are separated by commas (,). Set this
parameter based on the actual host information. -* indicates that indexes starting with a period (.)
are excluded.
    index => "abmau_edi*,business_test,goods_deploy*, -*"
    # Retain the default values of the following three items, including the number of threads, the
size of migrated data, and Logstash JVM configurations.
    docinfo=>true
    # Retain the default value. To increase the migration speed, you can increase the values of the
following two parameters, but to a proper extent.
    slices => 3
    size => 3000
  }
}

filter {
  # Delete some fields added by Logstash.
  mutate {
    remove_field => ["@timestamp", "@version"]
  }
}

output{
  elasticsearch{
    # Destination cluster address.
    hosts => ["http://10.100.xx.xx:9200", "http://10.100.xx.xx:9200"]
    # Username and password for logging in to the target cluster. If you do not need to configure
them, use the number sign (#) to comment them out.
    user => "admin"
    password => "*****"
    # Index name of the target cluster. The following configurations must be the same as that of
the source cluster.
    index => "%{[@metadata][_index]}"
    # #Index type of the target cluster. The following configurations must be the same as that of
the source cluster.
    document_type => "%{[@metadata][_type]}"
    # _id of the target data. If the original _id does not need to be retained, you can delete it. After
the deletion, the cluster performance can be better.
    document_id => "%{[@metadata][_id]}"
    ilm_enabled => false
    manage_template => false
  }

  # Debugging information. You are advised to delete this information before migration.
  # stdout { codec => rubydebug { metadata => true } }
}
```

### Step 5 Start Logstash to migrate cluster data.

1. Run the following command to start Logstash:  
`/usr/share/logstash/bin/logstash --path.settings /etc/logstash`
2. View the Logstash log file to check the task progress. The Logstash log directory is **/var/log/logstash/**.
3. Wait until the data migration is complete.

----End

## 1.2.2 Migrating Cluster Data Through Backup and Restoration

- To migrate data between Huawei Cloud Elasticsearch clusters, follow the instructions in .
- To migrate data from a user-built or third-party Elasticsearch cluster to a Huawei Cloud Elasticsearch cluster, perform the steps in this section.

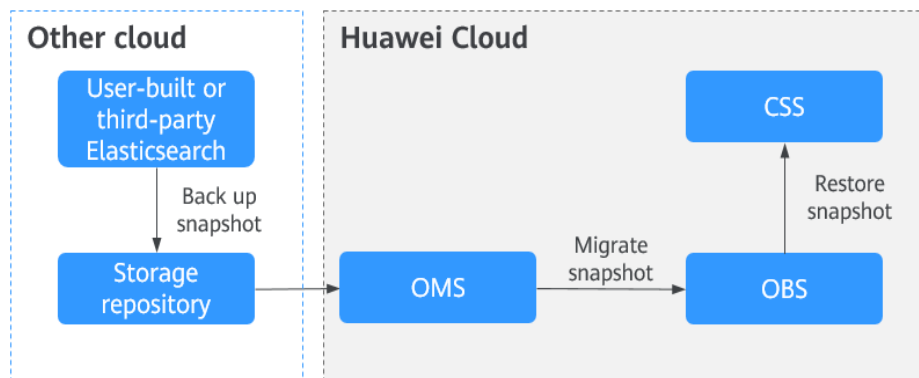
### Prerequisites

- Before using backup and restoration, ensure that:
  - Target Elasticsearch version  $\geq$  Source Elasticsearch version
  - Number of candidate master nodes of the target Elasticsearch cluster  $>$  Half of the number of candidate master nodes of the source Elasticsearch cluster
- Backup and restoration do not support incremental data synchronization. You need to stop data update before backing up data.
- The target Elasticsearch cluster has been created in CSS.

### Migration Process

The following figure shows the cluster migration process when the source is a user-built or third-party Elasticsearch cluster, and the target is an Elasticsearch cluster of CSS.

**Figure 1-1** Migration through backup and restoration



### Procedure

- Step 1** Create a shared repository that supports the S3 protocol, for example, OSS of the Alibaba Cloud.
- Step 2** Create a snapshot backup repository in the user-built or third-party Elasticsearch cluster to store Elasticsearch snapshot data.

For example, create a backup repository named **my\_backup** in Elasticsearch and associate it with the repository OSS.

```
PUT _snapshot/my_backup
{
  # Repository type.
  "type": "oss",
```



```
"settings": {
  # # Private network domain name of the repository in step 1.
  "endpoint": "http://oss-xxx.xxx.com",
  # User ID and password of the repository.
  "access_key_id": "xxx",
  "secret_access_key": "xxx",
  # Bucket name of the repository created in step 1.
  "bucket": "patent-esbak",
  # # Whether to enable snapshot file compression.
  "compress": false,
  # If the size of the uploaded snapshot data exceeds the value of this parameter, the data will be
  uploaded as blocks to the repository.
  "chunk_size": "1g",
  # Start position of the repository. The default value is the root directory.
  "base_path": "snapshot/"
}
```

**Step 3** Create a snapshot for the user-built or third-party Elasticsearch cluster.

- Create a snapshot for all indexes.

For example, create a snapshot named **snapshot\_1**.

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

- Create a snapshot for specified indexes.

For example, create a snapshot named **snapshot\_test** that contains indexes **patent\_analyse** and **patent**.

```
PUT _snapshot/my_backup/snapshot_test
{
  "indices": "patent_analyse,patent"
}
```

**Step 4** View the snapshot creation progress of the cluster.

- Run the following command to view information about all snapshots:

```
GET _snapshot/my_backup/_all
```

- Run the following command to view information about **snapshot\_1**:

```
GET _snapshot/my_backup/snapshot_1
```

**Step 5** Migrate snapshot data from the repository to OBS.

The Object Storage Migration Service (OMS) supports data migration from multiple cloud vendors to OBS. For details, see .

**Step 6** Create a repository in the Elasticsearch cluster of CSS and associate it with OBS. This repository will be used for restoring the snapshot data of the user-built or third-party Elasticsearch cluster.

For example, create a repository named **my\_backup\_all** in the cluster and associate it with the destination OBS.

```
PUT _snapshot/my_backup_all/
{
  "type" : "obs",
  "settings" : {
    # Private network domain name of OBS
    "endpoint" : "obs.xxx.xxx.com",
    "region" : "xxx",
    # Username and password for accessing OBS
    "access_key" : "xxx",
    "secret_key" : "xxx",
    # OBS bucket name, which must be the same as the destination OBS bucket name in the previous step
    "bucket" : "esbak",
    "compress" : "false",
    "chunk_size" : "1g",
    #Note that there is no slash (/) after snapshot.
  }
}
```

```
"base_path" : "snapshot",
"max_restore_bytes_per_sec": "100mb",
"max_snapshot_bytes_per_sec": "100mb"
}
}
```

**Step 7** Restore the snapshot data to the Elasticsearch cluster of CSS.

1. Check information about all snapshots.

```
GET _snapshot
```

2. Restore a snapshot

- Restore all the indexes from a snapshot. For example, to restore all the indexes from **snapshot\_1**, run the following command:

```
POST _snapshot/my_backup_all/snapshot_1/_restore?wait_for_completion=true
```

- Restores some indexes from a snapshot. For example, in the snapshot named **snapshot\_1**, restore only the indexes that do not start with a period (.).

```
POST _snapshot/my_backup/snapshot_1/_restore
```

```
{"indices": "*,-monitoring*,-security*,-kibana*","ignore_unavailable": "true"}
```

- Restore a specified index from a snapshot and renames the index. For example, in **snapshot\_1**, restore **index\_1** to **restored\_index\_1** and **index\_2** to **restored\_index\_2**.

```
POST /_snapshot/my_backup/snapshot_1/_restore
```

```
{
  # Restore only indexes index_1 and index_2 and ignore other indexes in the snapshot.
  "indices": "index_1,index_2"
  # Search for the index that is being restored. The index name must match the provided
  # template.
  "rename_pattern": "index_(.+)",
  # Rename the found index.
  "rename_replacement": "restored_index_$1"
}
```

**Step 8** View the snapshot restoration result.

- Run the following command to view the restoration results of all snapshots:

```
GET /_recovery/
```

- Run the following command to check the snapshot restoration result of a specified index:

```
GET {index_name}/_recovery
```

----End

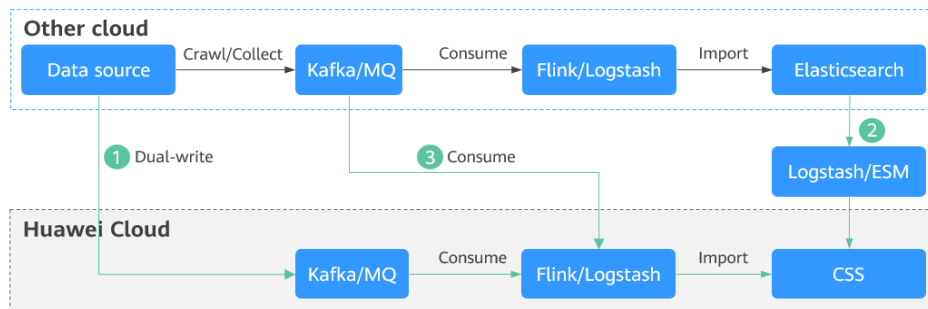
## 1.3 Migration from Kafka/MQ

### Process

In industries dealing with a large amount of data, such as IoT, news, public opinion analysis, and social networking, message middleware such as Kafka and MQ is used to balance traffic in peak and off-peak hours. The tools such as Flink and Logstash are then used to consume data, preprocess data, and import data to the search engine, providing the search service for external systems.

The following figure shows the process of migrating data from a Kafka or MQ cluster.

**Figure 1-2** Migration from a Kafka or MQ cluster



This migration solution is convenient and flexible.

- Convenient: Once the data of the two ES clusters becomes consistent, a cutover can be performed at any time.
- Flexible: Data can be added, deleted, modified, and queried on both sides.

## Procedure

- Step 1** Subscribe to incremental data. Create a consumer group in Kafka or MQ, and subscribe to incremental data.
- Step 2** Synchronize inventory data. Use a tool such as Logstash to migrate data from the source Elasticsearch cluster to the CSS cluster. If Logstash is used for data migration, see [Migrating Cluster Data Using Logstash](#).
- Step 3** Synchronize incremental data. After the inventory data is synchronized, enable the incremental consumer group. Based on the idempotence of Elasticsearch operations on data, when the new consumer group catches up with the previous consumer group, the data on both sides will be consistent.

### NOTE

For log migration, data in the source Elasticsearch cluster does not need to be migrated, and you can skip the inventory data synchronization. After the incremental data synchronization is complete, synchronize the data for a period of time (for example, three or seven days), and then directly perform cutover.

----End

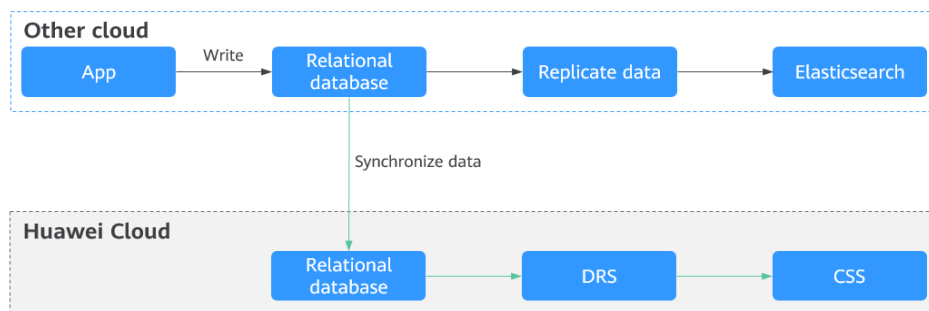
## 1.4 Migration from a Database

### Process

Elasticsearch supports full-text search and ad hoc queries. It is often used as a supplement to relational databases, such as MySQL and GaussDB(for MySQL), to improve the full-text search and high-concurrency ad hoc query capabilities of databases.

The following figure shows the process of migrating data from a database.

**Figure 1-3 Migration from a database**



This migration solution is convenient and flexible.

- Convenient: You can start a cutover while the CSS synchronizes incremental data.
- Flexible: Data can be added, deleted, modified, and queried on both sides.

## Procedure

Data Replication Service (DRS) can be used to migrate and synchronize data between relational databases, such as MySQL databases. For details about the supported database types, see .

- Step 1** Set up the data synchronization link. On Huawei Cloud, set up a synchronization link from a database to CSS.
- Step 2** Synchronize data from the database. Use a third-party tool, such as DRS or DataX, to synchronize data to CSS.

----End

# 2 Cluster Access

## 2.1 Overview

Elasticsearch clusters support multiple connection modes. You can determine how to access an Elasticsearch cluster based on the programming language used for your services. For more information about the clients used for CSS clusters in different security modes (non-security mode, security mode+HTTP, and security mode+HTTPS), see [Table 2-1](#).

- CSS provides visualized Kibana and Cerebro APIs for monitoring and operating clusters. On the CSS console, you can quickly access the Kibana and Cerebro of an Elasticsearch cluster.
- You can access Elasticsearch clusters by using cURL commands, Java clients, and Python clients. You can also use Hadoop clients to develop complex applications. Elasticsearch provides Java clients, including Rest High Level Client, Rest Low Level Client, and Transport Client. To avoid compatibility issues, use the Java client that matches your Elasticsearch cluster version.

**Table 2-1** Support for access from different clients

Client	Cluster in Non-Security Mode	Cluster in Security Mode + HTTP	Cluster in Security Mode + HTTPS
Kibana	Supported by clusters in all the three modes. To log in to Kibana from a cluster in security mode, enter the username and password for authentication. For details, see .		
Cerebro	Supported by clusters in all the three modes. To log in to Cerebro from a cluster in security mode, enter the username and password for authentication. For details, see .		
cURL	Supported by clusters in all the three modes. For details about the commands used for each mode, see <a href="#">Accessing a Cluster Using cURL Commands</a> .		

Client	Cluster in Non-Security Mode	Cluster in Security Mode + HTTP	Cluster in Security Mode + HTTPS
Java (Rest High Level Client)	Supported by clusters in all the three modes. For details about the commands used for each mode, see <a href="#">Accessing a Cluster Through the Rest High Level Client</a> .		
Java (Rest Low Level Client)	Supported by clusters in all the three modes. For details about the commands used for each mode, see <a href="#">Accessing a Cluster Through the Rest Low Level Client</a> .		
Java (Transport Client)	Only clusters in non-security mode are supported. For details, see <a href="#">Accessing the Cluster Through the Transport Client</a> .	Not supported	Not supported
Python	Supported by clusters in all the three modes. For details about the commands used for each mode, see <a href="#">Accessing a Cluster Using Python</a> .		
ES-Hadoop	Supported by clusters in all the three modes. For details about the commands used for each mode, see <a href="#">Using ES-Hadoop to Read and Write Data in Elasticsearch Through Hive</a> .		

## 2.2 Accessing a Cluster Using cURL Commands

If the CSS cluster and ECS are in the same VPC, you can run cURL commands on the ECS to directly access the Elasticsearch cluster. This method is mainly used to check whether the client that accesses the cluster can be connected to Elasticsearch nodes.

### Prerequisites

- The CSS cluster is available.
- An ECS that meets the following requirements is available:
  - The ECS and the CSS cluster must be in the same VPC to ensure network connectivity.
  - The security group of the ECS must be the same as that of the CSS cluster.

If they are different, change the ECS security group, or configure the inbound and outbound rules of the group to allow access from all the security groups of the cluster. For details, see .

For details about how to use the ECS, see .

## Procedure

1. Obtain the private network address of the cluster. It is used to access the cluster.
  - a. In the navigation pane on the left, choose **Clusters**.
  - b. In the cluster list, select a cluster, and obtain and record its **Private Network Address**. Format: `<host>:<port>` or `<host>:<port>,<host>:<port>`  
 If the cluster has only one node, the IP address and port number of only one node are displayed, for example, **10.62.179.32:9200**. If the cluster has multiple nodes, the IP addresses and port numbers of all nodes are displayed, for example, **10.62.179.32:9200,10.62.179.33:9200**.
2. Run one of the following commands on the ECS to access the cluster. The access command varies according to the security mode of the cluster.
  - Cluster in non-security mode  
`curl "http://<host>:<port>"`
  - Cluster in security mode + HTTP  
`curl -u <user>:<password> "http://<host>:<port>"`
  - Cluster in security mode + HTTPS  
`curl -u <user>:<password> -k "https://<host>:<port>"`

**Table 2-2** Variables

Variable	Description
<host>	IP address of each node in the cluster. If the cluster contains multiple nodes, there will be multiple IP addresses. You can use any of them.
<port>	Port number for accessing a cluster node. Generally, the port number is 9200.
<user>	Username for accessing the cluster.
<password>	Password of the user.

An access example is as follows:

```
curl "http://10.62.176.32:9200"
```

Information similar to the following is displayed:

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8
content-length: 513

{
  "name": "xxx-1",
  "cluster_name": "xxx",
  "cluster_uuid": "xxx_uuid",
  "version": {
    "number": "7.10.2",
    "build_flavor": "oss",
    "build_type": "tar",
    "build_hash": "unknown",
    "build_date": "unknown",
    "build_snapshot": true,
    "lucene_version": "8.7.0",
```

```
"minimum_wire_compatibility_version" : "6.7.0",  
"minimum_index_compatibility_version" : "6.0.0-beta1"  
},  
"tagline" : "You Know, for Search"  
}
```

#### NOTE

For more commands, see the [Elasticsearch documentation](#).

## 2.3 Accessing a Cluster Using Java

### 2.3.1 Accessing a Cluster Through the Rest High Level Client

Elasticsearch provides SDK (Rest High Level Client) for connecting to a cluster. This client encapsulates Elasticsearch APIs. You only need to construct required structures to access the Elasticsearch cluster. For details about how to use the Rest Client, see the official document at <https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/master/index.html>.

This section describes how to use the Rest High Level Client to access the CSS cluster. The Rest High Level Client can be connected to the cluster in any of the following ways:

- **Connecting to a Non-Security Cluster Through the Rest High Level Client:** applicable to clusters in non-security mode
- **Connecting to a Security Cluster Through Rest High Level Client (Without Security Certificates):** applicable to clusters in security mode+HTTP, and to clusters in security mode+HTTPS (without using certificates)
- **Connecting to a Security Cluster Through Rest High Level Client (With Security Certificates):** applicable to clusters in security mode+HTTPS

#### Precautions

You are advised to use the Rest High Level Client version that matches the Elasticsearch version. For example, use Rest High Level Client 7.6.2 to access the Elasticsearch cluster 7.6.2. If your Java Rest High Level Client version is later than the Elasticsearch cluster and incompatible with a few requests, you can use **RestHighLevelClient.getLowLevelClient()** to obtain Low Level Client and customize the Elasticsearch request content.

#### Prerequisites

- The CSS cluster is available.
- Ensure that the server running Java can communicate with the CSS cluster.
- Install JDK 1.8 on the server. You can download JDK 1.8 from: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- Declare Java dependencies.

7.6.2 indicates the version of the Elasticsearch Java client.

- Maven mode:

```
<dependency>  
<groupId>org.elasticsearch.client</groupId>
```



```
<artifactId>elasticsearch-rest-high-level-client</artifactId>
<version>7.6.2</version>
</dependency>
<dependency>
  <groupId>org.elasticsearch</groupId>
  <artifactId>elasticsearch</artifactId>
  <version>7.6.2</version>
</dependency>
```

– Gradle mode:

```
compile group: 'org.elasticsearch.client', name: 'elasticsearch-rest-high-level-client', version:
'7.6.2'
```

## Connecting to a Non-Security Cluster Through the Rest High Level Client

You can use the Rest High Level Client to connect to a non-security cluster and check whether the **test** index exists. The sample code is as follows:

```
import org.apache.http.HttpHost;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.GetIndexRequest;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;

/**
 * Use Rest Hive Level to connect to a non-security cluster.
 */
public class Main {
    public static void main(String[] args) throws IOException {
        List<String> host = Arrays.asList("x.x.x.x", "x.x.x.x");
        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, 9200, "http"));
        final RestHighLevelClient client = new RestHighLevelClient(builder);
        GetIndexRequest indexRequest = new GetIndexRequest("test");
        boolean exists = client.indices().exists(indexRequest, RequestOptions.DEFAULT);
        System.out.println(exists);
        client.close();
    }

    /**
     * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }
}
```

*host* indicates the IP address list of each node in the cluster. If there are multiple IP addresses, separate them with commas (,). *test* indicates the index name to be queried.

## Connecting to a Security Cluster Through Rest High Level Client (Without Security Certificates)

You can connect to a cluster in security mode+HTTP or a cluster in security mode + HTTPS (without using certificates).

The sample code is as follows:

```
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
```

```
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.nio.conn.ssl.SSLIOStrategy;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.elasticsearch.action.admin.cluster.health.ClusterHealthRequest;
import org.elasticsearch.action.admin.cluster.health.ClusterHealthResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.GetIndexRequest;
import org.elasticsearch.common.Nullable;

import java.io.IOException;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

/**
 * Connect to a security cluster through Rest High Level (without using certificates).
 */
public class Main {
    /**
     * Create a class for the client. Define the create function.
     */
    public static RestHighLevelClient create(List<String> host, int port, String protocol, int connectTimeout,
int connectionRequestTimeout, int socketTimeout, String username, String password) throws IOException{
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY, new UsernamePasswordCredentials(username,
password));
        SSLContext sc = null;
        try {
            sc = SSLContext.getInstance("SSL");
            sc.init(null, trustAllCerts, new SecureRandom());
        } catch (KeyManagementException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        SSLIOStrategy sessionStrategy = new SSLIOStrategy(sc, new NullHostNameVerifier());
        SecuredHttpClientConfigCallback httpClientConfigCallback = new
SecuredHttpClientConfigCallback(sessionStrategy,
credentialsProvider);

        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, port, protocol))
            .setRequestConfigCallback(requestConfig -> requestConfig.setConnectTimeout(connectTimeout))
            .setConnectionRequestTimeout(connectionRequestTimeout)
            .setSocketTimeout(socketTimeout)
            .setHttpClientConfigCallback(httpClientConfigCallback);
        final RestHighLevelClient client = new RestHighLevelClient(builder);
        logger.info("es rest client build success {} ", client);

        ClusterHealthRequest request = new ClusterHealthRequest();
        ClusterHealthResponse response = client.cluster().health(request, RequestOptions.DEFAULT);
        logger.info("es rest client health response {} ", response);
        return client;
    }
}
```

```

/**
 * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
 */
public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
    return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
}

/**
 * Configure trustAllCerts to ignore the certificate configuration.
 */
public static TrustManager[] trustAllCerts = new TrustManager[] {
    new X509TrustManager() {
        @Override
        public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
        }

        @Override
        public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
        }

        @Override
        public X509Certificate[] getAcceptedIssuers() {
            return null;
        }
    }
};

private static final Logger logger = LogManager.getLogger(Main.class);

static class SecuredHttpClientConfigCallback implements RestClientBuilder.HttpClientConfigCallback {
    @Nullable
    private final CredentialsProvider credentialsProvider;
    /**
     * The {@link SSLIOSessionStrategy} for all requests to enable SSL / TLS encryption.
     */
    private final SSLIOSessionStrategy sslStrategy;
    /**
     * Create a new {@link SecuredHttpClientConfigCallback}.
     *
     * @param credentialsProvider The credential provider, if a username/password have been supplied
     * @param sslStrategy The SSL strategy, if SSL / TLS have been supplied
     * @throws NullPointerException if {@code sslStrategy} is {@code null}
     */
    SecuredHttpClientConfigCallback(final SSLIOSessionStrategy sslStrategy,
        @Nullable final CredentialsProvider credentialsProvider) {
        this.sslStrategy = Objects.requireNonNull(sslStrategy);
        this.credentialsProvider = credentialsProvider;
    }
    /**
     * Get the {@link CredentialsProvider} that will be added to the HTTP client.
     *
     * @return Can be {@code null}.
     */
    @Nullable
    CredentialsProvider getCredentialsProvider() {
        return credentialsProvider;
    }
    /**
     * Get the {@link SSLIOSessionStrategy} that will be added to the HTTP client.
     *
     * @return Never {@code null}.
     */
    SSLIOSessionStrategy getSSLStrategy() {
        return sslStrategy;
    }
}

```

```

    * Sets the {@linkplain HttpAsyncClientBuilder#setDefaultCredentialsProvider(CredentialsProvider)
credential provider},
    *
    * @param httpClientBuilder The client to configure.
    * @return Always {@code httpClientBuilder}.
    */
    @Override
    public HttpAsyncClientBuilder customizeHttpClient(final HttpAsyncClientBuilder httpClientBuilder) {
        // enable SSL / TLS
        httpClientBuilder.setSSLStrategy(sslStrategy);
        // enable user authentication
        if (credentialsProvider != null) {
            httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
        }
        return httpClientBuilder;
    }
}

public static class NullHostNameVerifier implements HostnameVerifier {
    @Override
    public boolean verify(String arg0, SSLSession arg1) {
        return true;
    }
}

/**
 * The following is an example of the main function. Call the create function to create a client and check
whether the test index exists.
 */
public static void main(String[] args) throws IOException {
    RestHighLevelClient client = create(Arrays.asList("x.x.x.x", "x.x.x.x"), 9200, "https", 1000, 1000, 1000,
"username", "password");
    GetIndexRequest indexRequest = new GetIndexRequest("test");
    boolean exists = client.indices().exists(indexRequest, RequestOptions.DEFAULT);
    System.out.println(exists);
    client.close();
}
}

```

**Table 2-3** Variables

Parameter	Description
host	List of the IP addresses of Elasticsearch nodes (or independent Client node). Multiple IP addresses are separated using commas (,).
port	Access port of the Elasticsearch cluster. The default value is <b>9200</b> .
protocol	Connection protocol, which can be <b>http</b> or <b>https</b> .
connectTimeout	Socket connection timeout period.
connectionRequestTimeout	Timeout period of a socket connection request.
socketTimeout	Timeout period of a socket request.
username	Username for accessing the cluster.
password	Password of the user.

## Connecting to a Security Cluster Through Rest High Level Client (With Security Certificates)

You can use a security certificate to connect to a cluster in security mode + HTTPS.

1. Obtain the security certificate **CloudSearchService.cer**.
  - a. Log in to the CSS management console.
  - b. In the navigation pane, choose **Clusters**. The cluster list is displayed.
  - c. Click the name of a cluster to go to the cluster details page.
  - d. On the **Configuration** page, click **Download Certificate** next to **HTTPS Access**.
2. Convert the security certificate **CloudSearchService.cer**. Upload the downloaded security certificate to the client and use keytool to convert the .cer certificate into a .jks certificate that can be read by Java.
  - In Linux, run the following command to convert the certificate:  
`keytool -import -alias newname -keystore ./truststore.jks -file ./CloudSearchService.cer`
  - In Windows, run the following command to convert the certificate:  
`keytool -import -alias newname -keystore .\truststore.jks -file .\CloudSearchService.cer`

In the preceding command, *newname* indicates the user-defined certificate name.

After this command is executed, you will be prompted to set the certificate password and confirm the password. Securely store the password. It will be used for accessing the cluster.

3. Access the cluster. The sample code is as follows:

```
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.nio.conn.ssl.SSLIOStrategy;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.elasticsearch.action.admin.cluster.health.ClusterHealthRequest;
import org.elasticsearch.action.admin.cluster.health.ClusterHealthResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.GetIndexRequest;
import org.elasticsearch.common.Nullable;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.SecureRandom;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

import javax.net.ssl.SSLContext;
```

```
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;

/**
 * Use Rest Hive Level to connect to a security cluster (using an HTTPS certificate).
 */
public class Main {
    public static RestHighLevelClient create(List<String> host, int port, String protocol, int
connectTimeout, int connectionRequestTimeout, int socketTimeout, String username, String password,
String cerFilePath,
String cerPassword) throws IOException {

        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY, new
UsernamePasswordCredentials(username, password));
        SSLContext sc = null;
        try {
            TrustManager[] tm = {new MyX509TrustManager(cerFilePath, cerPassword)};
            sc = SSLContext.getInstance("SSL", "SunJSSE");
            //You can also use SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
            sc.init(null, tm, new SecureRandom());
        } catch (Exception e) {
            e.printStackTrace();
        }

        SSLIOSessionStrategy sessionStrategy = new SSLIOSessionStrategy(sc, new
NoopHostnameVerifier());
        SecuredHttpClientConfigCallback httpClientConfigCallback = new
SecuredHttpClientConfigCallback(sessionStrategy,
credentialsProvider);

        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, port, protocol))
            .setRequestConfigCallback(requestConfig ->
requestConfig.setConnectTimeout(connectTimeout)
                .setConnectionRequestTimeout(connectionRequestTimeout)
                .setSocketTimeout(socketTimeout))
            .setHttpClientConfigCallback(httpClientConfigCallback);
        final RestHighLevelClient client = new RestHighLevelClient(builder);
        logger.info("es rest client build success {} ", client);

        ClusterHealthRequest request = new ClusterHealthRequest();
        ClusterHealthResponse response = client.cluster().health(request, RequestOptions.DEFAULT);
        logger.info("es rest client health response {} ", response);
        return client;
    }

    /**
     * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }

    /**
     * SecuredHttpClientConfigCallback class definition
     */
    static class SecuredHttpClientConfigCallback implements
RestClientBuilder.HttpClientConfigCallback {
        @Nullable
        private final CredentialsProvider credentialsProvider;

        private final SSLIOSessionStrategy sslStrategy;

        SecuredHttpClientConfigCallback(final SSLIOSessionStrategy sslStrategy,
            @Nullable final CredentialsProvider credentialsProvider) {
            this.sslStrategy = Objects.requireNonNull(sslStrategy);
            this.credentialsProvider = credentialsProvider;
        }
    }
}
```

```
@Nullable
CredentialsProvider getCredentialsProvider() {
    return credentialsProvider;
}

SSLIOSessionStrategy getSSLStrategy() {
    return sslStrategy;
}

@Override
public HttpAsyncClientBuilder customizeHttpClient(final HttpAsyncClientBuilder
httpClientBuilder) {
    httpClientBuilder.setSSLStrategy(sslStrategy);
    if (credentialsProvider != null) {
        httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
    }
    return httpClientBuilder;
}
}

private static final Logger logger = LogManager.getLogger(Main.class);

public static class MyX509TrustManager implements X509TrustManager {
    X509TrustManager sunJSSEX509TrustManager;

    MyX509TrustManager(String cerFilePath, String cerPassword) throws Exception {
        File file = new File(cerFilePath);
        if (!file.isFile()) {
            throw new Exception("Wrong Certification Path");
        }
        System.out.println("Loading KeyStore " + file + "...");
        InputStream in = new FileInputStream(file);
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(in, cerPassword.toCharArray());
        TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509", "SunJSSE");
        tmf.init(ks);
        TrustManager[] tms = tmf.getTrustManagers();
        for (TrustManager tm : tms) {
            if (tm instanceof X509TrustManager) {
                sunJSSEX509TrustManager = (X509TrustManager) tm;
                return;
            }
        }
        throw new Exception("Couldn't initialize");
    }

    @Override
    public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException {

    }

    @Override
    public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {

    }

    @Override
    public X509Certificate[] getAcceptedIssuers() {
        return new X509Certificate[0];
    }
}

/**
 * The following is an example of the main function. Call the create function to create a client and
 * check whether the test index exists.
 */
```

```
public static void main(String[] args) throws IOException {
    RestHighLevelClient client = create(Arrays.asList("xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"), 9200,
    "https", 1000, 1000, 1000, "username", "password", "cerFilePath", "cerPassword");
    GetIndexRequest indexRequest = new GetIndexRequest("test");
    boolean exists = client.indices().exists(indexRequest, RequestOptions.DEFAULT);
    System.out.println(exists);
    client.close();
}
}
```

**Table 2-4** Function parameters

Name	Description
host	List of the IP addresses of Elasticsearch nodes (or independent Client node). Multiple IP addresses are separated using commas (,).
port	Access port of the Elasticsearch cluster. The default value is <b>9200</b> .
protocol	Connection protocol. Set this parameter to <b>https</b> .
connectTimeout	Socket connection timeout period.
connectionRequestTimeout	Timeout period of a socket connection request.
socketTimeout	Timeout period of a socket request.
username	Username for accessing the cluster.
password	Password of the user.
cerFilePath	Certificate path.
cerPassword	Certificate password.

### 2.3.2 Accessing a Cluster Through the Rest Low Level Client

The high-level client is encapsulated based on the low-level client. If the method calls (such as .search and .bulk) in the high-level client cannot meet the requirements or has compatibility issues, you can use the low-level client. You can even use **HighLevelClient.getLowLevelClient()** to directly obtain a low-level client. A low-level client allows you to define the request structure, which is more flexible and supports all the request formats of Elasticsearch, such as GET, POST, DELETE, and HEAD.

This section describes how to use the Rest Low Level Client to access the CSS cluster. The methods are as follows. For each method, you can directly create a REST low-level client, or create a high-level client and then invoke `getLowLevelClient()` to obtain a low-level client.

- **Connecting to a Non-Security Cluster Through the Rest Low Level Client:** applicable to clusters in non-security mode



- **Connecting to a Security Cluster Through Rest Low Level Client (Without Security Certificates)**: applicable to clusters in security mode+HTTP, and to clusters in security mode+HTTPS (without using certificates)
- **Connecting to a Security Cluster Through Rest Low Level Client (With Security Certificates)**: applicable to clusters in security mode+HTTPS

## Precautions

You are advised to use the Rest Low Level Client version that matches the Elasticsearch version. For example, use Rest Low Level Client 7.6.2 to access the Elasticsearch cluster 7.6.2.

## Prerequisites

- The CSS cluster is available.
- Ensure that the server running Java can communicate with the CSS cluster.
- Install JDK 1.8 on the server. You can download JDK 1.8 from: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- Declare the Apache version in Maven mode. The following code uses version 7.6.2 as an example.

7.6.2 indicates the version of the Elasticsearch Java client.

```
<dependency>
  <groupId>org.elasticsearch.client</groupId>
  <artifactId>elasticsearch-rest-client</artifactId>
  <version>7.6.2</version>
</dependency>
<dependency>
  <groupId>org.elasticsearch</groupId>
  <artifactId>elasticsearch</artifactId>
  <version>7.6.2</version>
</dependency>
```

## Connecting to a Non-Security Cluster Through the Rest Low Level Client

- **Method 1: Directly create a Rest Low Level Client.**

```
import org.apache.http.HttpHost;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;

public class Main {

    public static void main(String[] args) throws IOException {
        List<String> host = Arrays.asList("xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx");
        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, 9200, "http"));
        /**
         * Create a Rest Low Level Client.
         */
        RestClient lowLevelClient = builder.build();
        /**
         * Check whether the test index exists. If the index exists, 200 is returned. If the index does not
         exist, 404 is returned.
         */
        Request request = new Request("HEAD", "/test");
```

```

Response response = lowLevelClient.performRequest(request);
System.out.println(response.getStatusLine().getStatusCode());
lowLevelClient.close();
}

/**
 * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
 */
public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
    return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
}
}

```

- **Method 2: Create a high-level client and then call `getLowLevelClient()` to obtain a low-level client.**

```

import org.apache.http.HttpHost;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;

public class Main {

    public static void main(String[] args) throws IOException {
        List<String> host = Arrays.asList("xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx");
        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, 9200, "http"));
        final RestHighLevelClient restHighLevelClient = new RestHighLevelClient(builder);
        /**
         * Create a high-level client and then call getLowLevelClient() to obtain a low-level client.
         * The code differs from the client creation code only in the following line:
         */
        final RestClient lowLevelClient = restHighLevelClient.getLowLevelClient();
        /**
         * Check whether the test index exists. If the index exists, 200 is returned. If the index does not
         * exist, 404 is returned.
         */
        Request request = new Request("HEAD", "/test");
        Response response = lowLevelClient.performRequest(request);
        System.out.println(response.getStatusLine().getStatusCode());
        lowLevelClient.close();
    }

    /**
     * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }
}

```

*host* indicates the IP address list of each node in the cluster. If there are multiple IP addresses, separate them with commas (,). *test* indicates the index name to be queried.

## Connecting to a Security Cluster Through Rest Low Level Client (Without Security Certificates)

- **Method 1: Directly create a Rest Low Level Client.**

```

import org.apache.http.HttpHost;
import org.apache.http.HttpResponse;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;

```

```
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.client.DefaultConnectionKeepAliveStrategy;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.nio.conn.ssl.SSLIOSessionStrategy;
import org.apache.http.protocol.HttpContext;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.common.Nullable;

import java.io.IOException;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.concurrent.TimeUnit;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;import javax.net.ssl.X509TrustManager;

public class Main {

    /**
     * Create a class for the client. Define the create function.
     */
    public static RestClient create(List<String> host, int port, String protocol, int connectTimeout, int
connectionRequestTimeout, int socketTimeout, String username, String password) throws
IOException {
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY, new
UsernamePasswordCredentials(username, password));
        SSLContext sc = null;
        try {
            sc = SSLContext.getInstance("SSL");
            sc.init(null, trustAllCerts, new SecureRandom());
        } catch (KeyManagementException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        SSLIOSessionStrategy sessionStrategy = new SSLIOSessionStrategy(sc, new
NullHostNameVerifier());
        SecuredHttpClientConfigCallback httpClientConfigCallback = new
SecuredHttpClientConfigCallback(sessionStrategy,
credentialsProvider);

        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, port, protocol))
            .setRequestConfigCallback(requestConfig ->
requestConfig.setConnectTimeout(connectTimeout)
                .setConnectionRequestTimeout(connectionRequestTimeout)
                .setSocketTimeout(socketTimeout))
            .setHttpClientConfigCallback(httpClientConfigCallback);
        final RestClient client = builder.build();
        logger.info("es rest client build success {} ", client);
        return client;
    }

    /**
     * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }
}
```

```
}

/**
 * Configure trustAllCerts to ignore the certificate configuration.
 */
public static TrustManager[] trustAllCerts = new TrustManager[] {
    new X509TrustManager() {
        @Override
        public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
        }

        @Override
        public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
        }

        @Override
        public X509Certificate[] getAcceptedIssuers() {
            return null;
        }
    }
};

/**
 * The CustomConnectionKeepAliveStrategy function is used to set the connection keepalive time when
there are a large number of short connections or when the number of data requests is small.
 */
public static class CustomConnectionKeepAliveStrategy extends
DefaultConnectionKeepAliveStrategy {
    public static final CustomConnectionKeepAliveStrategy INSTANCE = new
CustomConnectionKeepAliveStrategy();

    private CustomConnectionKeepAliveStrategy() {
        super();
    }

    /**
     * Maximum keep alive time (minutes)
     * The default value is 10 minutes. You can set it based on the number of TCP connections in
TIME_WAIT state. If there are too many TCP connections, you can increase the value.
     */
    private final long MAX_KEEP_ALIVE_MINUTES = 10;

    @Override
    public long getKeepAliveDuration(HttpResponse response, HttpContext context) {
        long keepAliveDuration = super.getKeepAliveDuration(response, context);
        // <0 indicates that the keepalive period is unlimited.
        // Change the period from unlimited to a default period.
        if (keepAliveDuration < 0) {
            return TimeUnit.MINUTES.toMillis(MAX_KEEP_ALIVE_MINUTES);
        }
        return keepAliveDuration;
    }
}

private static final Logger logger = LogManager.getLogger(Main.class);

static class SecuredHttpClientConfigCallback implements
RestClientBuilder.HttpClientConfigCallback {
    @Nullable
    private final CredentialsProvider credentialsProvider;
    /**
     * The {@link SSLIOStrategy} for all requests to enable SSL / TLS encryption.
     */
    private final SSLIOStrategy sslStrategy;
    /**
     * Create a new {@link SecuredHttpClientConfigCallback}.
     */
}
```

```

    * @param credentialsProvider The credential provider, if a username/password have been
    supplied
    * @param sslStrategy The SSL strategy, if SSL / TLS have been supplied
    * @throws NullPointerException if {@code sslStrategy} is {@code null}
    */
    SecuredHttpClientConfigCallback(final SSLIOSessionStrategy sslStrategy,
        @Nullable final CredentialsProvider credentialsProvider) {
        this.sslStrategy = Objects.requireNonNull(sslStrategy);
        this.credentialsProvider = credentialsProvider;
    }
    /**
    * Get the {@link CredentialsProvider} that will be added to the HTTP client.
    *
    * @return Can be {@code null}.
    */
    @Nullable
    CredentialsProvider getCredentialsProvider() {
        return credentialsProvider;
    }
    /**
    * Get the {@link SSLIOSessionStrategy} that will be added to the HTTP client.
    *
    * @return Never {@code null}.
    */
    SSLIOSessionStrategy getSSLStrategy() {
        return sslStrategy;
    }
    /**
    * Sets the {@linkplain
    HttpAsyncClientBuilder#setDefaultCredentialsProvider(CredentialsProvider) credential provider},
    *
    * @param httpClientBuilder The client to configure.
    * @return Always {@code httpClientBuilder}.
    */
    @Override
    public HttpAsyncClientBuilder customizeHttpClient(final HttpAsyncClientBuilder
    httpClientBuilder) {
        // enable SSL / TLS
        httpClientBuilder.setSSLStrategy(sslStrategy);
        // enable user authentication
        if (credentialsProvider != null) {
            httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
        }
        return httpClientBuilder;
    }
}

public static class NullHostNameVerifier implements HostnameVerifier {
    @Override
    public boolean verify(String arg0, SSLSession arg1) {
        return true;
    }
}

/**
 * The following is an example of the main function. Call the create function to create a Rest Low
 * Level Client and check whether the test index exists.
 */
public static void main(String[] args) throws IOException {
    RestClient lowLevelClient = create(Arrays.asList("xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"), 9200, "http",
    1000, 1000, 1000, "username", "password");
    Request request = new Request("HEAD", "/test");
    Response response = lowLevelClient.performRequest(request);
    System.out.println(response.getStatusLine().getStatusCode());
    lowLevelClient.close();
}
}

```

- **Method 2: Create a high-level client and then call getLowLevelClient() to obtain a low-level client.**

```
import org.apache.http.HttpHost;
import org.apache.http.HttpResponse;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.client.DefaultConnectionKeepAliveStrategy;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.nio.conn.ssl.SSLIOSessionStrategy;
import org.apache.http.protocol.HttpContext;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.common.Nullable;

import java.io.IOException;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.concurrent.TimeUnit;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;import javax.net.ssl.X509TrustManager;

import org.elasticsearch.client.RestHighLevelClient;

public class Main13 {

    /**
     * Create a class for the client. Define the create function.
     */
    public static RestHighLevelClient create(List<String> host, int port, String protocol, int
connectTimeout, int connectionRequestTimeout, int socketTimeout, String username, String
password) throws IOException {

        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY, new
UsernamePasswordCredentials(username, password));
        SSLContext sc = null;
        try {
            sc = SSLContext.getInstance("SSL");
            sc.init(null, trustAllCerts, new SecureRandom());
        } catch (KeyManagementException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        SSLIOSessionStrategy sessionStrategy = new SSLIOSessionStrategy(sc, new
NullHostNameVerifier());
        SecuredHttpClientConfigCallback httpClientConfigCallback = new
SecuredHttpClientConfigCallback(sessionStrategy,
credentialsProvider);

        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, port, protocol))
.setRequestConfigCallback(requestConfig ->
requestConfig.setConnectTimeout(connectTimeout)
.setConnectionRequestTimeout(connectionRequestTimeout)
.setSocketTimeout(socketTimeout))
.setHttpClientConfigCallback(httpClientConfigCallback);
        final RestHighLevelClient client = new RestHighLevelClient(builder);
        logger.info("es rest client build success {}", client);
    }
}
```

```
        return client;
    }

    /**
     * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }

    /**
     * Configure trustAllCerts to ignore the certificate configuration.
     */
    public static TrustManager[] trustAllCerts = new TrustManager[] {
        new X509TrustManager() {
            @Override
            public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
            }

            @Override
            public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
            }

            @Override
            public X509Certificate[] getAcceptedIssuers() {
                return null;
            }
        }
    };

    /**
     * The CustomConnectionKeepAliveStrategy function is used to set the connection keepalive time when
     * there are a large number of short connections or when the number of data requests is small.
     */
    public static class CustomConnectionKeepAliveStrategy extends
DefaultConnectionKeepAliveStrategy {
        public static final CustomConnectionKeepAliveStrategy INSTANCE = new
CustomConnectionKeepAliveStrategy();

        private CustomConnectionKeepAliveStrategy() {
            super();
        }

        /**
         * Maximum keep alive time (minutes)
         * The default value is 10 minutes. You can set it based on the number of TCP connections in
         * TIME_WAIT state. If there are too many TCP connections, you can increase the value.
         */
        private final long MAX_KEEP_ALIVE_MINUTES = 10;

        @Override
        public long getKeepAliveDuration(HttpResponse response, HttpContext context) {
            long keepAliveDuration = super.getKeepAliveDuration(response, context);
            // <0 indicates that the keepalive period is unlimited.
            // Change the period from unlimited to a default period.
            if (keepAliveDuration < 0) {
                return TimeUnit.MINUTES.toMillis(MAX_KEEP_ALIVE_MINUTES);
            }
            return keepAliveDuration;
        }
    }

    private static final Logger logger = LogManager.getLogger(Main.class);

    static class SecuredHttpClientConfigCallback implements
RestClientBuilder.HttpClientConfigCallback {
        @Nullable
```

```

private final CredentialsProvider credentialsProvider;
/**
 * The {@link SSLIOSessionStrategy} for all requests to enable SSL / TLS encryption.
 */
private final SSLIOSessionStrategy sslStrategy;
/**
 * Create a new {@link SecuredHttpClientConfigCallback}.
 *
 * @param credentialsProvider The credential provider, if a username/password have been
supplied
 * @param sslStrategy The SSL strategy, if SSL / TLS have been supplied
 * @throws NullPointerException if {@code sslStrategy} is {@code null}
 */
SecuredHttpClientConfigCallback(final SSLIOSessionStrategy sslStrategy,
 @Nullable final CredentialsProvider credentialsProvider) {
    this.sslStrategy = Objects.requireNonNull(sslStrategy);
    this.credentialsProvider = credentialsProvider;
}
/**
 * Get the {@link CredentialsProvider} that will be added to the HTTP client.
 *
 * @return Can be {@code null}.
 */
@Nullable
CredentialsProvider getCredentialsProvider() {
    return credentialsProvider;
}
/**
 * Get the {@link SSLIOSessionStrategy} that will be added to the HTTP client.
 *
 * @return Never {@code null}.
 */
SSLIOSessionStrategy getSSLStrategy() {
    return sslStrategy;
}
/**
 * Sets the {@linkplain
HttpAsyncClientBuilder#setDefaultCredentialsProvider(CredentialsProvider) credential provider},
 *
 * @param httpClientBuilder The client to configure.
 * @return Always {@code httpClientBuilder}.
 */
@Override
public HttpAsyncClientBuilder customizeHttpClient(final HttpAsyncClientBuilder
httpClientBuilder) {
    // enable SSL / TLS
    httpClientBuilder.setSSLStrategy(sslStrategy);
    // enable user authentication
    if (credentialsProvider != null) {
        httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
    }
    return httpClientBuilder;
}
}

public static class NullHostNameVerifier implements HostnameVerifier {
    @Override
    public boolean verify(String arg0, SSLSession arg1) {
        return true;
    }
}

/**
 * The following is an example of the main function. Call the create function to create a high-level
client, call the getLowLevelClient() function to obtain a low-level client, and check whether the test
index exists.
 */
public static void main(String[] args) throws IOException {
    RestHighLevelClient client = create(Arrays.asList("xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"), 9200,

```



```

"http", 1000, 1000, 1000, "username", "password");
    RestClient lowLevelClient = client.getLowLevelClient();
    Request request = new Request("HEAD", "test");
    Response response = lowLevelClient.performRequest(request);
    System.out.println(response.getStatusLine().getStatusCode());
    lowLevelClient.close();
}
}

```

**Table 2-5 Variables**

Parameter	Description
host	List of the IP addresses of Elasticsearch nodes (or independent Client node). Multiple IP addresses are separated using commas (,).
port	Access port of the Elasticsearch cluster. The default value is <b>9200</b> .
protocol	Connection protocol, which can be <b>http</b> or <b>https</b> .
connectTimeout	Socket connection timeout period.
connectionRequestTimeout	Timeout period of a socket connection request.
socketTimeout	Timeout period of a socket request.
username	Username for accessing the cluster.
password	Password of the user.

## Connecting to a Security Cluster Through Rest Low Level Client (With Security Certificates)

- **Method 1: Directly create a Rest Low Level Client.**

```

import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.nio.conn.ssl.SSLIOStrategy;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.common.Nullable;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.SecureRandom;

```

```
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

import javax.net.ssl.SSLContext;import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;

public class Main13 {

    private static final Logger logger = LogManager.getLogger(Main.class);

    /**
     * Create a class for the client. Define the create function.
     */
    public static RestClient create(List<String> host, int port, String protocol, int connectTimeout, int
connectionRequestTimeout, int socketTimeout, String username, String password, String cerFilePath,
String cerPassword) throws IOException {
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY, new
UsernamePasswordCredentials(username, password));
        SSLContext sc = null;
        try {
            TrustManager[] tm = {new MyX509TrustManager(cerFilePath, cerPassword)};
            sc = SSLContext.getInstance("SSL", "SunJSSE");
            //You can also use SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
            sc.init(null, tm, new SecureRandom());
        } catch (Exception e) {
            e.printStackTrace();
        }

        SSLIOSessionStrategy sessionStrategy = new SSLIOSessionStrategy(sc, new
NoopHostnameVerifier());
        SecuredHttpClientConfigCallback httpClientConfigCallback = new
SecuredHttpClientConfigCallback(sessionStrategy,
credentialsProvider);

        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, port, protocol))
            .setRequestConfigCallback(requestConfig ->
requestConfig.setConnectTimeout(connectTimeout)
                .setConnectionRequestTimeout(connectionRequestTimeout)
                .setSocketTimeout(socketTimeout))
            .setHttpClientConfigCallback(httpClientConfigCallback);
        final RestClient client = builder.build();
        logger.info("es rest client build success {} ", client);
        return client;
    }

    /**
     * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);}

    static class SecuredHttpClientConfigCallback implements
RestClientBuilder.HttpClientConfigCallback {
        @Nullable
        private final CredentialsProvider credentialsProvider;

        private final SSLIOSessionStrategy sslStrategy;

        SecuredHttpClientConfigCallback(final SSLIOSessionStrategy sslStrategy,
            @Nullable final CredentialsProvider credentialsProvider) {
            this.sslStrategy = Objects.requireNonNull(sslStrategy);
            this.credentialsProvider = credentialsProvider;
        }
    }
}
```

```
@Nullable
CredentialsProvider getCredentialsProvider() {
    return credentialsProvider;
}

SSLIOSessionStrategy getSSLStrategy() {
    return sslStrategy;
}

@Override
public HttpAsyncClientBuilder customizeHttpClient(final HttpAsyncClientBuilder
httpClientBuilder) {
    httpClientBuilder.setSSLStrategy(sslStrategy);
    if (credentialsProvider != null) {
        httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
    }
    return httpClientBuilder;
}

public static class MyX509TrustManager implements X509TrustManager {
    X509TrustManager sunJSSEX509TrustManager;

    MyX509TrustManager(String cerFilePath, String cerPassword) throws Exception {
        File file = new File(cerFilePath);
        if (!file.isFile()) {
            throw new Exception("Wrong Certification Path");
        }
        System.out.println("Loading KeyStore " + file + "...");
        InputStream in = new FileInputStream(file);
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(in, cerPassword.toCharArray());
        TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509", "SunJSSE");
        tmf.init(ks);
        TrustManager[] tms = tmf.getTrustManagers();
        for (TrustManager tm : tms) {
            if (tm instanceof X509TrustManager) {
                sunJSSEX509TrustManager = (X509TrustManager) tm;
                return;
            }
        }
        throw new Exception("Couldn't initialize");
    }

    @Override
    public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException {

    }

    @Override
    public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {

    }

    @Override
    public X509Certificate[] getAcceptedIssuers() {
        return new X509Certificate[0];
    }
}

/**
 * The following is an example of the main function. Call the create function to create a Rest Low
 * Level Client and check whether the test index exists.
 */
public static void main(String[] args) throws IOException {
    RestClient lowLevelClient = create(Arrays.asList("xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"), 9200,
"https", 1000, 1000, 1000, "username", "password", "cerFilePath", "cerPassword");
    Request request = new Request("HEAD", "test");
}
```

```
Response response = lowLevelClient.performRequest(request);
System.out.println(response.getStatusLine().getStatusCode());
lowLevelClient.close();
}
}
```

- **Method 2: Create a high-level client and then call `getLowLevelClient()` to obtain a low-level client.**

```
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
import org.apache.http.nio.conn.ssl.SSLIOSessionStrategy;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.elasticsearch.action.admin.cluster.health.ClusterHealthRequest;
import org.elasticsearch.action.admin.cluster.health.ClusterHealthResponse;
import org.elasticsearch.client.Request;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.Response;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.common.Nullable;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.SecureRandom;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;

public class Main {

    private static final Logger logger = LogManager.getLogger(Main.class);

    /**
     * Create a class for the client. Define the create function.
     */
    public static RestHighLevelClient create(List<String> host, int port, String protocol, int
connectTimeout, int connectionRequestTimeout, int socketTimeout, String username, String password,
String cerFilePath, String cerPassword) throws IOException {
        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY, new
UsernamePasswordCredentials(username, password));
        SSLContext sc = null;
        try {
            TrustManager[] tm = {new MyX509TrustManager(cerFilePath, cerPassword)};
            sc = SSLContext.getInstance("SSL", "SunJSSE");
            //You can also use SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
            sc.init(null, tm, new SecureRandom());
        } catch (Exception e) {
            e.printStackTrace();
        }

        SSLIOSessionStrategy sessionStrategy = new SSLIOSessionStrategy(sc, new
NoopHostnameVerifier());
        SecuredHttpClientConfigCallback httpClientConfigCallback = new
```

```
SecuredHttpClientConfigCallback(sessionStrategy,
    credentialsProvider);

    RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, port, protocol))
        .setRequestConfigCallback(requestConfig ->
requestConfig.setConnectTimeout(connectTimeout)
        .setConnectionRequestTimeout(connectionRequestTimeout)
        .setSocketTimeout(socketTimeout))
        .setHttpClientConfigCallback(httpClientConfigCallback);
    final RestHighLevelClient client = new RestHighLevelClient(builder);
    logger.info("es rest client build success {}", client);

    ClusterHealthRequest request = new ClusterHealthRequest();
    ClusterHealthResponse response = client.cluster().health(request, RequestOptions.DEFAULT);
    logger.info("es rest client health response {}", response);
    return client;
}

/**
 * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
 */
public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
    return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);}

static class SecuredHttpClientConfigCallback implements
RestClientBuilder.HttpClientConfigCallback {
    @Nullable
    private final CredentialsProvider credentialsProvider;

    private final SSLIOSessionStrategy sslStrategy;

    SecuredHttpClientConfigCallback(final SSLIOSessionStrategy sslStrategy,
        @Nullable final CredentialsProvider credentialsProvider) {
        this.sslStrategy = Objects.requireNonNull(sslStrategy);
        this.credentialsProvider = credentialsProvider;
    }

    @Nullable
    CredentialsProvider getCredentialsProvider() {
        return credentialsProvider;
    }

    SSLIOSessionStrategy getSSLStrategy() {
        return sslStrategy;
    }

    @Override
    public HttpAsyncClientBuilder customizeHttpClient(final HttpAsyncClientBuilder
httpClientBuilder) {
        httpClientBuilder.setSSLStrategy(sslStrategy);
        if (credentialsProvider != null) {
            httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
        }
        return httpClientBuilder;
    }
}

public static class MyX509TrustManager implements X509TrustManager {
    X509TrustManager sunJSSEX509TrustManager;

    MyX509TrustManager(String cerFilePath, String cerPassword) throws Exception {
        File file = new File(cerFilePath);
        if (!file.isFile()) {
            throw new Exception("Wrong Certification Path");
        }
        System.out.println("Loading KeyStore " + file + "...");
        InputStream in = new FileInputStream(file);
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(in, cerPassword.toCharArray());
        TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509", "SunJSSE");
```

```

tmf.init(ks);
TrustManager[] tms = tmf.getTrustManagers();
for (TrustManager tm : tms) {
    if (tm instanceof X509TrustManager) {
        sunJSSEX509TrustManager = (X509TrustManager) tm;
        return;
    }
}
throw new Exception("Couldn't initialize");
}

@Override
public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException {

}

@Override
public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {

}

@Override
public X509Certificate[] getAcceptedIssuers() {
    return new X509Certificate[0];
}
}

/**
 * The following is an example of the main function. Call the create function to create a high-level
 * client, call the getLowLevelClient() function to obtain a low-level client, and check whether the test
 * index exists.
 */
public static void main(String[] args) throws IOException {
    RestHighLevelClient client = create(Arrays.asList("xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"), 9200,
    "https", 1000, 1000, 1000, "username", "password", "cerFilePath", "cerPassword");
    RestClient lowLevelClient = client.getLowLevelClient();
    Request request = new Request("HEAD", "test");
    Response response = lowLevelClient.performRequest(request);
    System.out.println(response.getStatusLine().getStatusCode());
    lowLevelClient.close();
}
}

```

**Table 2-6** Function parameters

Name	Description
host	List of the IP addresses of Elasticsearch nodes (or independent Client node). Multiple IP addresses are separated using commas (,).
port	Access port of the Elasticsearch cluster. The default value is <b>9200</b> .
protocol	Connection protocol. Set this parameter to <b>https</b> .
connectTimeout	Socket connection timeout period.
connectionRequestTimeout	Timeout period of a socket connection request.

Name	Description
socketTimeout	Timeout period of a socket request.
username	Username for accessing the cluster.
password	Password of the user.
cerFilePath	Certificate path.
cerPassword	Certificate password.

### 2.3.3 Accessing the Cluster Through the Transport Client

You can use Transport Client to access a CSS cluster in non-security mode. If the cluster is in security mode, you are advised to use Rest High Level Client to access the Elasticsearch cluster.

#### Precautions

You are advised to use the Transport Client version that matches the Elasticsearch version. For example, use Transport Client 7.6.2 to access the Elasticsearch cluster 7.6.2.

#### Prerequisites

- The CSS cluster is available.
- Ensure that the server running Java can communicate with the CSS cluster.
- Install JDK 1.8 on the server. You can download JDK 1.8 from: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- Declare Java dependencies.

7.6.2 indicates the version of the Elasticsearch Java client.

```
<dependency>
  <groupId>org.elasticsearch.client</groupId>
  <artifactId>transport</artifactId>
  <version>7.6.2</version>
</dependency>
<dependency>
  <groupId>org.elasticsearch</groupId>
  <artifactId>elasticsearch</artifactId>
  <version>7.6.2</version>
</dependency>
```

#### Procedure

The following code is an example of using Transport Client to connect to the Elasticsearch cluster and check whether the **test** index exists.

```
import org.elasticsearch.action.ActionFuture;
import org.elasticsearch.action.admin.indices.exists.indices.IndicesExistsRequest;
import org.elasticsearch.action.admin.indices.exists.indices.IndicesExistsResponse;
import org.elasticsearch.client.transport.TransportClient;
import org.elasticsearch.common.settings.Settings;
import org.elasticsearch.common.transport.TransportAddress;
```

```
import org.elasticsearch.transport.client.PreBuiltTransportClient;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.concurrent.ExecutionException;

public class Main {
    public static void main(String[] args) throws ExecutionException, InterruptedException,
UnknownHostException {
        String cluster_name = "xxx";
        String host1 = "x.x.x.x";
        String host2 = "y.y.y.y";
        Settings settings = Settings.builder()
            .put("client.transport.sniff",false)
            .put("cluster.name", cluster_name)
            .build();
        TransportClient client = new PreBuiltTransportClient(settings)
            .addTransportAddress(new TransportAddress(InetAddress.getByName(host1), 9300))
            .addTransportAddress(new TransportAddress(InetAddress.getByName(host2), 9300));
        IndicesExistsRequest indicesExistsRequest = new IndicesExistsRequest("test");
        ActionFuture<IndicesExistsResponse> exists = client.admin().indices().exists(indicesExistsRequest);
        System.out.println(exists.get().isExists());
    }
}
```

In the preceding information, *cluster\_name* indicates the cluster name, and *host1* and *host2* indicate the IP addresses of the cluster nodes. You can run the **GET \_cat/nodes** command to view the IP addresses of the nodes.

## 2.4 Accessing a Cluster Using Python

You can access a CSS cluster using Python.

### Prerequisites

- The CSS cluster is available.
- Ensure that the server running Python can communicate with the CSS cluster.

### Procedure

1. Install the Elasticsearch Python client. You are advised to use the client version that matches the Elasticsearch version. For example, if the cluster version is 7.6.2, install the Elasticsearch Python client 7.6.  

```
pip install Elasticsearch==7.6.2
```
2. Create an Elasticsearch client and check whether the **test** index exists. The examples for clusters in different security modes are as follows:

- Cluster in non-security mode  
from elasticsearch import Elasticsearch

```
class ElasticFactory(object):

    def __init__(self, host: list, port: str, username: str, password: str):
        self.port = port
        self.host = host
        self.username = username
        self.password = password

    def create(self) -> Elasticsearch:
        addrs = []
        for host in self.host:
            addr = {'host': host, 'port': self.port}
            addrs.append(addr)
```



```

if self.username and self.password:
    elasticsearch = Elasticsearch(addr, http_auth=(self.username, self.password))
else:
    elasticsearch = Elasticsearch(addr)
return elasticsearch

es = ElasticFactory(["xxx.xxx.xxx.xxx"], "9200", None, None).create()
print(es.indices.exists(index='test'))

```

- **Cluster in security mode + HTTP**

```

from elasticsearch import Elasticsearch

class ElasticFactory(object):

    def __init__(self, host: list, port: str, username: str, password: str):
        self.port = port
        self.host = host
        self.username = username
        self.password = password

    def create(self) -> Elasticsearch:
        addr = []
        for host in self.host:
            addr = {'host': host, 'port': self.port}
            addr.append(addr)

        if self.username and self.password:
            elasticsearch = Elasticsearch(addr, http_auth=(self.username, self.password))
        else:
            elasticsearch = Elasticsearch(addr)
        return elasticsearch

es = ElasticFactory(["xxx.xxx.xxx.xxx"], "9200", "username", "password").create()
print(es.indices.exists(index='test'))

```

- **Cluster in security mode + HTTPS**

```

from elasticsearch import Elasticsearch
import ssl

class ElasticFactory(object):

    def __init__(self, host: list, port: str, username: str, password: str):
        self.port = port
        self.host = host
        self.username = username
        self.password = password

    def create(self) -> Elasticsearch:
        context = ssl.create_unverified_context()

        addr = []
        for host in self.host:
            addr = {'host': host, 'port': self.port}
            addr.append(addr)

        if self.username and self.password:
            elasticsearch = Elasticsearch(addr, http_auth=(self.username, self.password),
            scheme="https", ssl_context=context)
        else:
            elasticsearch = Elasticsearch(addr)
        return elasticsearch

es = ElasticFactory(["xxx.xxx.xxx.xxx"], "9200", "username", "password").create()
print(es.indices.exists(index='test'))

```

**Table 2-7** Variables

Name	Description
host	List of the IP addresses of Elasticsearch nodes (or independent Client node). Multiple IP addresses are separated using commas (,).
port	Access port of the Elasticsearch cluster. Enter <b>9200</b> .
username	Username for accessing the cluster.
password	Password of the user.

3. Create a cluster index through the Elasticsearch client.

```
mappings = {
  "settings": {
    "index": {
      "number_of_shards": number_of_shards,
      "number_of_replicas": 1,
    },
  },
  "mappings": {
    properties
  }
}
result = es.indices.create(index=index, body=mappings)
```

4. Query the index created in the previous step through the Elasticsearch client.

```
body = {
  "query": {
    "match": {
      "Query field": "Query content"
    }
  }
}
result = es.search(index=index, body=body)
```

## 2.5 Using ES-Hadoop to Read and Write Data in Elasticsearch Through Hive

The Elasticsearch-Hadoop (ES-Hadoop) connector combines the massive data storage and in-depth processing capabilities of Hadoop with the real-time search and analysis capabilities of Elasticsearch. It allows you to quickly get to know big data and work better in the Hadoop ecosystem.

This section uses the ES-Hadoop of MRS as an example to describe how to connect to a CSS cluster. You can configure any other applications that need to use the Elasticsearch cluster. Ensure the network connection between the client and the Elasticsearch cluster is normal.

### Prerequisites

- The CSS cluster is available.
- The client can communicate with the CSS cluster.

- The CSS and MRS clusters are in the same region, AZ, VPC, and subnet.

**Figure 2-1** CSS cluster information

**Configuration**

Region	[Redacted]
AZ	[Redacted]
VPC	vpc-[Redacted]
Subnet	subnet-[Redacted]
Security Group	[Redacted]

**Procedure**

1. Obtain the private network address of the cluster. It is used to access the cluster.
  - a. In the navigation pane on the left, choose **Clusters**.
  - b. In the cluster list, select a cluster, and obtain and record its **Private Network Address**. Format: *<host>:<port>* or *<host>:<port>,<host>:<port>*  
 If the cluster has only one node, the IP address and port number of only one node are displayed, for example, **10.62.179.32:9200**. If the cluster has multiple nodes, the IP addresses and port numbers of all nodes are displayed, for example, **10.62.179.32:9200,10.62.179.33:9200**.
2. Log in to an MRS cluster node. For details, see .
3. Run the cURL command on an MRS cluster node to check the network connectivity. Ensure every node in the MRS cluster can connect to the CSS cluster.
  - Cluster in non-security mode  
`curl -X GET http://<host>:<port>`
  - Cluster in security mode + HTTP  
`curl -X GET http://<host>:<port> -u <user>:<password>`
  - Cluster in security mode + HTTPS  
`curl -X GET https://<host>:<port> -u <user>:<password> -ik`

**Table 2-8** Variables

Variable	Description
<host>	IP address of each node in the cluster. If the cluster contains multiple nodes, there will be multiple IP addresses. You can use any of them.
<port>	Port number for accessing a cluster node. Generally, the port number is 9200.

Variable	Description
<user>	Username for accessing the cluster.
<password>	Password of the user.

4. Download the [ES-Hadoop lib package](#) and decompress it to obtain the **elasticsearch-hadoop-x.x.x.jar** file. The version must be the same as the CSS cluster version. For example, if the CSS cluster version is 7.6.2, you are advised to download **elasticsearch-hadoop-7.6.2.zip**.
5. Download the httpclient dependency package [commons-httpclient:commons-httpclient-3.1.jar](#). In the package name, **3.1** indicates the version number. Select the package of the version you need.
6. Install the MRS client. If the MRS client has been installed, skip this step. For details, see .
7. Log in to the MRS client. Upload the JAR dependency packages of ES-Hadoop and httpclient to the MRS client.
8. Create an HDFS directory on the MRS client. Upload the ES-Hadoop lib package and the httpclient dependency package to the directory.

```
hadoop fs -mkdir /tmp/hadoop-es
hadoop fs -put elasticsearch-hadoop-x.x.x.jar /tmp/hadoop-es
hadoop fs -put commons-httpclient-3.1.jar /tmp/hadoop-es
```

9. Log in to the Hive client from the MRS client. For details, see .
10. On the Hive client, add the ES-Hadoop lib package and the httpclient dependency package. This command is valid only for the current session. Enter **beeline** or **hive** to go to the execution page and run the following commands:

```
add jar hdfs:///tmp/hadoop-es/commons-httpclient-3.1.jar;
add jar hdfs:///tmp/hadoop-es/elasticsearch-hadoop-x.x.x.jar;
```

11. On the Hive client, create a Hive foreign table.

- Cluster in non-security mode

```
CREATE EXTERNAL table IF NOT EXISTS student(
  id BIGINT,
  name STRING,
  addr STRING
)
STORED BY 'org.elasticsearch.hadoop.hive.EsStorageHandler'
TBLPROPERTIES(
  'es.nodes' = 'xxx.xxx.xxx.xxx:9200',
  'es.port' = '9200',
  'es.net.ssl' = 'false',
  'es.nodes.wan.only' = 'false',
  'es.nodes.discovery'='false',
  'es.input.use.sliced.partitions'='false',
  'es.resource' = 'student/_doc'
);
```

- Cluster in security mode + HTTP

```
CREATE EXTERNAL table IF NOT EXISTS student(
  id BIGINT,
  name STRING,
  addr STRING
)
STORED BY 'org.elasticsearch.hadoop.hive.EsStorageHandler'
TBLPROPERTIES(
  'es.nodes' = 'xxx.xxx.xxx.xxx:9200',
  'es.port' = '9200',
```

```
'es.net.ssl' = 'false',
'es.nodes.wan.only' = 'false',
'es.nodes.discovery'='false',
'es.input.use.sliced.partitions'='false',
'es.nodes.client.only'='true',
'es.resource' = 'student/_doc',
'es.net.http.auth.user' = 'username',
'es.net.http.auth.pass' = 'password'
);
```

- Cluster in security mode + HTTPS
  - i. Obtain the security certificate **CloudSearchService.cer**.
    - 1) Log in to the CSS management console.
    - 2) In the navigation pane, choose **Clusters**. The cluster list is displayed.
    - 3) Click the name of a cluster to go to the cluster details page.
    - 4) On the **Configuration** page, click **Download Certificate** next to **HTTPS Access**.
  - ii. Convert the security certificate **CloudSearchService.cer**. Upload the downloaded security certificate to the client and use **keytool** to convert the .cer certificate into a .jks certificate that can be read by Java.

- o In Linux, run the following command to convert the certificate:
 

```
keytool -import -alias newname -keystore ./truststore.jks -file ./CloudSearchService.cer
```

- o In Windows, run the following command to convert the certificate:
 

```
keytool -import -alias newname -keystore .\truststore.jks -file .\CloudSearchService.cer
```

In the preceding command, *newname* indicates the user-defined certificate name.

After this command is executed, you will be prompted to set the certificate password and confirm the password. Securely store the password. It will be used for accessing the cluster.

- iii. Put the .jks file to the same path of each node in the MRS cluster, for example, **/tmp**. You can run the **scp** command to transfer the file. Ensure user **omm** has the permission to read the file. You can run the following command to set the permission:

```
chown -R omm truststore.jks
```

- iv. Create a Hive foreign table.

```
CREATE EXTERNAL table IF NOT EXISTS student(
  id BIGINT,
  name STRING,
  addr STRING
)
STORED BY 'org.elasticsearch.hadoop.hive.EsStorageHandler'
TBLPROPERTIES(
  'es.nodes' = 'https://xxx.xxx.xxx.xxx:9200',
  'es.port' = '9200',
  'es.net.ssl' = 'true',
  'es.net.ssl.truststore.location' = 'cerFilePath',
  'es.net.ssl.truststore.pass' = 'cerPassword',
  'es.nodes.wan.only' = 'false',
  'es.nodes.discovery'='false',
  'es.nodes.client.only'='true',
  'es.input.use.sliced.partitions'='false',
  'es.resource' = 'student/_doc,
```

```
'es.net.http.auth.user' = 'username',
'es.net.http.auth.pass' = 'password'
);
```

**Table 2-9** ES-Hadoop parameters

Parameter	Default Value	Description
es.nodes	localhost	Address for accessing the CSS cluster. You can view private network address in the cluster list.
es.port	9200	Port number for accessing a cluster. Generally, the port number is 9200.
es.nodes.wan.only	false	Whether to perform node sniffing.
es.nodes.discovery	true	Whether to disable node discovery.
es.input.use.sliced.partitions	true	Whether to use slices. Its value can be: <ul style="list-style-type: none"> <li>• <b>true</b></li> <li>• <b>false</b></li> </ul> <b>NOTE</b> If this parameter is set to <b>true</b> , the index prefetch time may be significantly prolonged, and may even be much longer than the data query time. You are advised to set this parameter to <b>false</b> to improve query efficiency.
es.resource	NA	Specifies the index and type to be read and written.
es.net.http.auth.user	NA	Username for accessing the cluster. Set this parameter only if the security mode is enabled.
es.net.http.auth.pass	NA	Password of the user. Set this parameter only if the security mode is enabled.
es.net.ssl	false	Whether to enable SSL. If SSL is enabled, you need to configure the security certificate information.
es.net.ssl.truststore.location	NA	Path of the .jks certificate file, for example, <b>file:///tmp/truststore.jks</b> .

Parameter	Default Value	Description
es.nodes.client.only	false	Check whether the IP address of an independent Client node is configured for <b>es.nodes</b> (that is, whether the Client node is enabled during Elasticsearch cluster creation). If yes, change the value to <b>true</b> , or an error will be reported, indicating that the data node cannot be found.
es.net.ssl.truststore.pass	NA	Password of the .jks certificate file.

For details about ES-Hadoop configuration items, see the [official configuration description](#).

- On the Hive client, insert data.

```
INSERT INTO TABLE student VALUES (1, "Lucy", "address1"), (2, "Lily", "address2");
```

- On the Hive client, run a query.

```
select * from student;
```

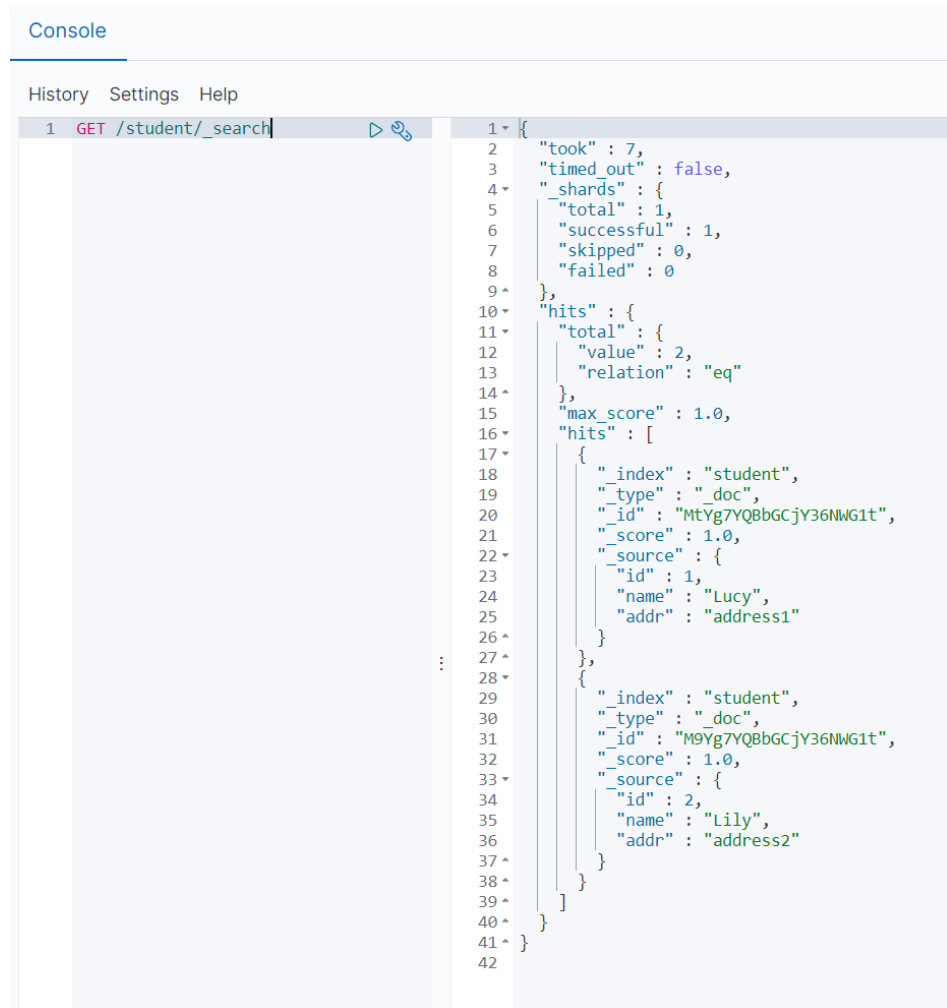
The query result is as follows:

```
+-----+-----+-----+
| student.id | student.name | student.addr |
+-----+-----+-----+
| 1          | Lucy         | address1     |
| 2          | Lily        | address2     |
+-----+-----+-----+
2 rows selected (0.116 seconds)
```

- Log in to the CSS console and choose **Clusters**. Locate the target cluster and click **Access Kibana** in the **Operation** column.
- On the **Dev Tools** page of Kibana, run a query and view the result.

```
GET /student/_search
```

Figure 2-2 Kibana query result



The screenshot shows the Kibana console interface. At the top, there is a 'Console' tab. Below it, there are links for 'History', 'Settings', and 'Help'. The main area displays a search query: '1 GET /student/\_search'. To the right of the query, there is a 'Run' button (a play icon) and a 'Copy' button (a document icon). The search results are displayed in a JSON format, showing the following structure:

```
1 {
2   "took" : 7,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 2,
13      "relation" : "eq"
14    },
15    "max_score" : 1.0,
16    "hits" : [
17      {
18        "_index" : "student",
19        "_type" : "_doc",
20        "_id" : "MtYg7YQBbGCjY36NWG1t",
21        "_score" : 1.0,
22        "_source" : {
23          "id" : 1,
24          "name" : "Lucy",
25          "addr" : "address1"
26        }
27      },
28      {
29        "_index" : "student",
30        "_type" : "_doc",
31        "_id" : "M9Yg7YQBbGCjY36NWG1t",
32        "_score" : 1.0,
33        "_source" : {
34          "id" : 2,
35          "name" : "Lily",
36          "addr" : "address2"
37        }
38      }
39    ]
40  }
41 }
```



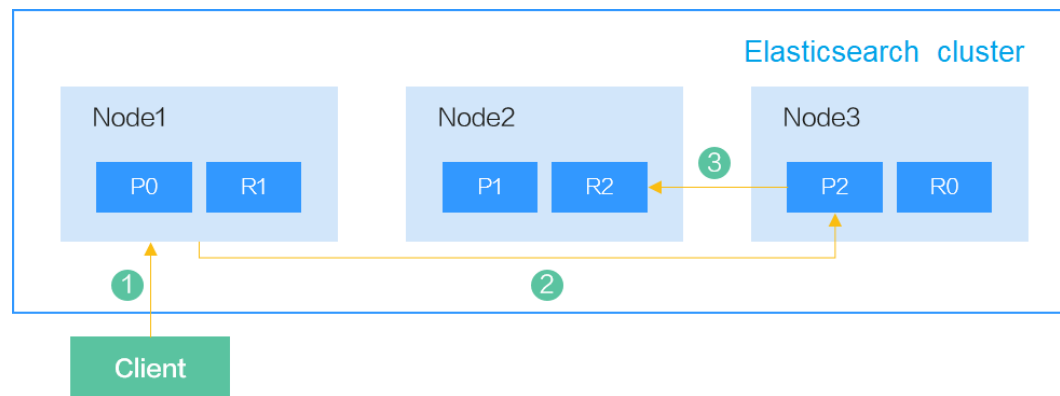
# 3 Cluster Performance Tuning

## 3.1 Optimizing Write Performance

Before using a CSS cluster, you are advised to optimize the write performance of the cluster to improve efficiency.

### Data Write Process

Figure 3-1 Data write process



The process of writing data from a client to Elasticsearch is as follows:

1. The client sends a data write request to Node1. Here Node1 is the coordinator node.
2. Node1 routes the data to shard 2 based on the `_id` of the data. In this case, the request is forwarded to Node3 and the write operation is performed.
3. After data is written to the primary shard, the request is forwarded to the replica shard of Node2. After the data is written to the replica, Node3 reports the write success to the coordinator node, and the coordinator node reports it to the client.

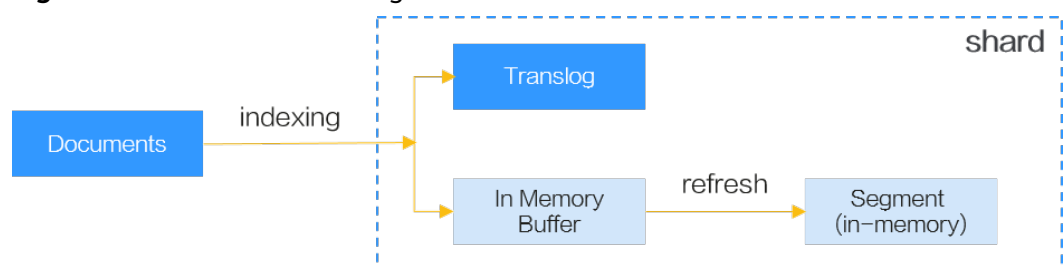
An index in Elasticsearch consists of one or more shards. Each shard contains multiple segments, and each segment is an inverted index.

**Figure 3-2** Elasticsearch index composition



When a document is inserted into Elasticsearch, the document is first written to the buffer and then periodically refreshed from the buffer to the segment. The refresh frequency is specified by the **refresh\_interval** parameter. By default, data is refreshed every second.

**Figure 3-3** Process of inserting a document into Elasticsearch



## Improving Write Performance

In the Elasticsearch data write process, the following solutions can be used to improve performance:

**Table 3-1** Improving write performance

N o.	Solution	Description
1	Use SSDs or improve cluster configurations.	Using SSDs can greatly speed up data write and merge operations. For CSS, you are advised to select the ultra-high I/O storage or ultra-high I/O servers.
2	Use Bulk APIs.	The client writes data in batches. You are advised to write 1 MB to 10 MB data in each batch.
3	Randomly generate <b>_id</b> .	If <b>_id</b> is specified, a query operation will be triggered before data is written, affecting data write performance. In scenarios where data does not need to be retrieved using <b>_id</b> , you are advised to use a randomly generated <b>_id</b> .
4	Set a proper number of segments.	You are advised to set the number of shards to a multiple of the number of cluster data nodes. Ensure each shard is smaller than 50 GB.

No.	Solution	Description
5	Close replicas.	<p>Data write and query are performed in off-peak hours. Close data copies during writing and open them afterwards.</p> <p>The command for disabling replicas in Elasticsearch 7.x is as follows:</p> <pre>PUT {index}_settings {   "number_of_replicas": 0 }</pre>
6	Adjust the index refresh frequency.	<p>During batch data writing, you can set <b>refresh_interval</b> to a large value or <b>-1</b> (indicating no refresh), improving the write performance by reducing refresh.</p> <p>In Elasticsearch 7.x, run the following command to set the update time to 15s:</p> <pre>PUT {index}_settings {   "refresh_interval": "15s" }</pre>
7	Change the number of write threads and the size of the write queue.	<p>You can increase the number of write threads and the size of the write queue, or error code 429 may be returned for unexpected traffic peaks.</p> <p>In Elasticsearch 7.x, you can modify the following parameters to optimize write performance: <b>thread_pool.write.size</b> and <b>thread_pool.write.queue_size</b></p>
8	Set a proper field type.	<p>Specify the type of each field in the cluster, so that Elasticsearch will not regard the fields as a combination of keywords and texts, which unnecessarily increase data volume. Keywords are used for keyword search, and texts used for full-text search.</p> <p>For the fields that do not require indexes, you are advised to set <b>index</b> to <b>false</b>.</p> <p>In Elasticsearch 7.x, run the following command to set <b>index</b> to <b>false</b> for <b>field1</b>:</p> <pre>PUT {index} {   "mappings": {     "properties": {       "field1": {         "type": "text",         "index": false       }     }   } }</pre>

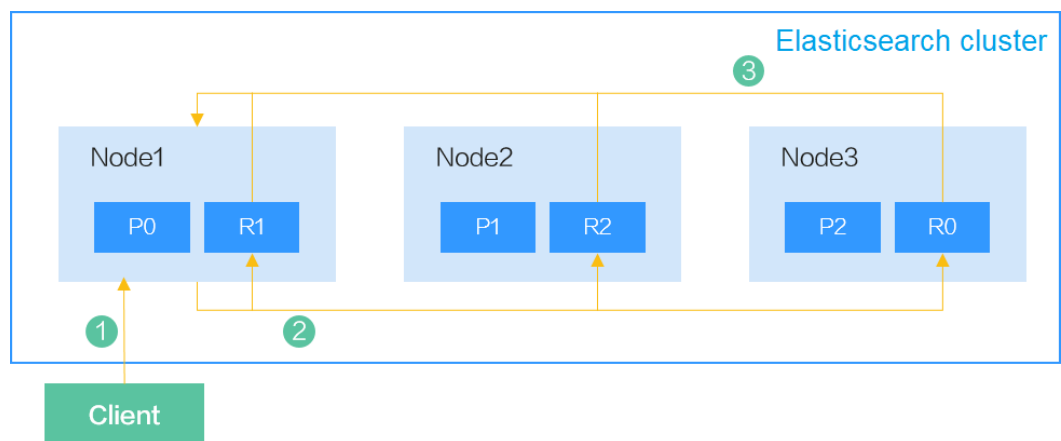
No.	Solution	Description
9	Optimize the shard balancing policy.	<p>By default, Elasticsearch uses the load balance policy based on the disk capacity. If there are multiple nodes, especially if some of them are newly added, shards may be unevenly allocated on the nodes. To avoid such problems, you can set the index-level parameter <b>routing.allocation.total_shards_per_node</b> to control the distribution of index shards on each node. You can set this parameter in the index template, or modify the setting of an existing index to make the setting take effect.</p> <p>Run the following command to modify the setting of an existing index:</p> <pre data-bbox="651 705 1428 862">PUT {index}/_settings {   "index": {     "routing.allocation.total_shards_per_node": 2   } }</pre>

### 3.2 Optimizing Query Performance

Before using a CSS cluster, you are advised to optimize the query performance of the cluster to improve efficiency.

#### Data Query Process

Figure 3-4 Data query process



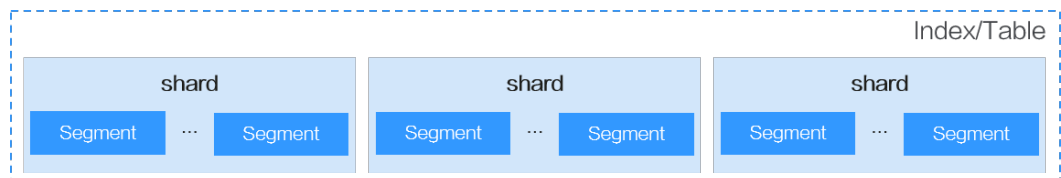
When a client sends a query request to Elasticsearch, the query process is as follows:

1. The client sends a data query request to Node1. Here Node1 is the coordinator node.
2. Node1 selects a shard based on the shard distribution and the index specified in the query, and then forwards the request to Node1, Node2, and Node3.

- Each shard executes the query task. After the query succeeds on the shards, the query results are aggregated to Node1, which returns the results to the client.

For a query request, five shards can be queried concurrently on a node by default. If there are more than five shards, the query will be performed in batches. In a single shard, the query is performed by traversing each segment one by one.

**Figure 3-5** Elasticsearch index composition



## Improving Query Performance

In the Elasticsearch data query process, the following solutions can be used to improve performance:

**Table 3-2** Improving query performance

No.	Solution	Description
1	Use <b>_routing</b> to reduce the number of shards scanned during retrieval.	<p>During data import, configure <b>routing</b> to route data to a specific shard instead of all the shards of the related index, improving the overall throughput of the cluster.</p> <p>In Elasticsearch 7.x, run the following commands:</p> <ul style="list-style-type: none"> <li>Insert data based on a specified routing.  <pre>PUT /{index}/_doc/1?routing=user1 {   "title": "This is a document" }</pre> </li> <li>Query data based on a specified routing.  <pre>GET /{index}/_doc/1?routing=user1</pre> </li> </ul>

No.	Solution	Description
2	Use index sorting to reduce the number of segments to be scanned.	<p>When a request is processed on a shard, the segments of the shard are traversed one by one. By using index sorting, the range query or sorting query can be terminated in advance (early-terminate).</p> <p>For example, in Elasticsearch 7.x, run the following commands:</p> <pre>// Assume the <b>date</b> field needs to be frequently used for range query. PUT {index} {   "settings": {     "index": {       "sort.field": "date",       "sort.order": "desc"     }   },   "mappings": {     "properties": {       "date": {         "type": "date"       }     }   } }</pre>
3	Add query cache to improve cache hit.	<p>When a filter request is executed in a segment, the bitset is used to retain the result, which can be reused for later similar queries, thus reducing the overall query workloads.</p> <p>You can add query cache by increasing the value of <b>indices.queries.cache.size</b>. For details, see . Restart the cluster for the modification to take effect.</p>
4	Perform forcemerge in advance to reduce the number of segments to be scanned.	<p>For read-only indexes that are periodically rolled, you can periodically execute forcemerge to combine small segments into large segments and permanently delete indexes marked as <b>deleted</b>.</p> <p>In Elasticsearch 7.x, a configuration example is as follows:</p> <pre>// Assume the number of segments after index forcemerge is set to 10. POST /{index}/forcemerge?max_num_segments=10</pre>

# 4 Practices

## 4.1 Using CSS to Accelerate Database Query and Analysis

### Overview

Elasticsearch is used as a supplement to relational databases, such as MySQL and GaussDB(for MySQL), to improve the full-text search and high-concurrency ad hoc query capabilities of the databases.

This chapter describes how to synchronize data from a MySQL database to CSS to accelerate full-text search and ad hoc query and analysis. The following figure shows the solution process.

**Figure 4-1** Using CSS to accelerate database query and analysis



1. Service data is stored in the MySQL database.
2. DRS synchronizes data from MySQL to CSS in real time.
3. CSS is used for full-text search and data query and analysis.

### Prerequisites

- A CSS cluster and a MySQL database in security mode have been created, and they are in the same VPC and security group.
- Data to be synchronized exists in the MySQL database. This section uses the following table structure and initial data as an example.
  - a. Create a student information table in MySQL.

```
CREATE TABLE `student` (  
  `dsc` varchar(100) COLLATE utf8mb4_general_ci DEFAULT NULL,  
  `age` smallint unsigned DEFAULT NULL,  
  `name` varchar(32) COLLATE utf8mb4_general_ci NOT NULL,  
  `id` int unsigned NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

```
b. Insert the initial data of three students into the MySQL database.
INSERT INTO student (id,name,age,dsc)
VALUES
('1','Jack Ma Yun','50','Jack Ma Yun is a Chinese business magnate, investor and philanthropist. '),
('2','will smith','22','also known by his stage name the Fresh Prince, is an American actor, rapper,
and producer.'),
('3','James Francis Cameron','68','the director of avatar');
```

- Indexes have been created in the CSS cluster and match the table indexes in the MySQL database.

The following is an example of the indexes in the cluster in this chapter:

```
PUT student
{
  "settings": {
    "number_of_replicas": 0,
    "number_of_shards": 3
  },
  "mappings": {
    "properties": {
      "id": {
        "type": "keyword"
      },
      "name": {
        "type": "short"
      },
      "age": {
        "type": "short"
      },
      "desc": {
        "type": "text"
      }
    }
  }
}
```

Configure **number\_of\_shards** and **number\_of\_replicas** as needed.

## Procedure

- Step 1** Use DRS to synchronize MySQL data to CSS in real time. For details, see .

In this example, configure the parameters by following the suggestions in [Table 4-1](#).

**Table 4-1** Synchronization parameters

Module	Parameter	Suggestion
<b>Create Synchronization Instance &gt; Synchronize Instance Details</b>	<b>Network Type</b>	Select <b>VPC</b> .
	<b>Source DB Instance</b>	Select the RDS for MySQL instance to be synchronized, that is, the MySQL database that stores service data.
	<b>Synchronization Instance Subnet</b>	Select the subnet where the synchronization instance is located. You are advised to select the subnet where the database instance and the CSS cluster are located.
<b>Configure Source and Destination</b>	<b>VPC and Subnet</b>	Select the VPC and subnet of the CSS cluster.



Module	Parameter	Suggestion
<b>Databases &gt; Destination Database</b>	<b>IP Address or Domain Name</b>	Enter the IP address of the CSS cluster. For details, see <a href="#">Obtaining the IP address of a CSS cluster</a> .
	<b>Database Username and Database Password</b>	Enter the administrator username ( <b>admin</b> ) and password of the CSS cluster.
	<b>Encryption Certificate</b>	Select the security certificate of the CSS cluster. If <b>SSL Connection</b> is not enabled, you do not need to select any certificate. For details, see <a href="#">Obtaining the security certificate of a CSS cluster</a> .
<b>Set Synchronization Task</b>	<b>Flow Control</b>	Select <b>No</b> .
	<b>Synchronization Object Type</b>	Deselect <b>Table structure</b> , because the indexes matching MySQL tables have been created in the CSS cluster.
	<b>Synchronization Object</b>	Select <b>Tables</b> . Select the database and table name corresponding to CSS. <b>NOTE</b> Ensure the type name in the configuration item is the same as the index name, that is, <b>_doc</b> .
<b>Process Data</b>	-	Click <b>Next</b> .

After the synchronization task is started, wait until the **Status** of the task changes from **Full** synchronization to **Incremental**, indicating real-time synchronization has started.

**Step 2** Check the synchronization status of the database.

1. Verify full data synchronization.

Run the following command in Kibana of CSS to check whether full data has been synchronized to CSS:

```
GET student/_search
```

2. Insert new data in the source cluster and check whether the data is synchronized to CSS.

For example, insert a record whose **id** is **4** in the source cluster.

```
INSERT INTO student (id,name,age,dsc)
```

```
VALUES
```

```
('4','Bill Gates','50','Gates III is an American business magnate, software developer, investor, author, and philanthropist.')
```

Run the following command in Kibana of CSS to check whether new data is synchronized to CSS:

```
GET student/_search
```

- Update data in the source cluster and check whether the data is synchronized to CSS.

For example, in the record whose **id** is **4**, change the value of **age** from **50** to **55**.

```
UPDATE student set age='55' WHERE id=4;
```

Run the following command in Kibana of CSS to check whether the data is updated in CSS:

```
GET student/_search
```

- Delete data from the source cluster and check whether the data is deleted synchronously from CSS.

For example, delete the record whose **id** is **4**.

```
DELETE FROM student WHERE id=4;
```

Run the following command in Kibana of CSS to check whether the data is deleted synchronously from CSS:

```
GET student/_search
```

### Step 3 Verify the full-text search capability of the database.

For example, run the following command to query the data that contains **avatar** in **dsc** in CSS:

```
GET student/_search
{
  "query": {
    "match": {
      "dsc": "avatar"
    }
  }
}
```

### Step 4 Verify the ad hoc query capability of the database.

For example, query **philanthropist** whose age is greater than **40** in CSS.

```
GET student/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "dsc": "philanthropist"
          }
        },
        {
          "range": {
            "age": {
              "gte": 40
            }
          }
        }
      ]
    }
  }
}
```

### Step 5 Verify the statistical analysis capability of the database.

For example, use CSS to collect statistics on the age distributions of all users.

```
GET student/_search
{
  "size": 0,
```

```
"query": {
  "match_all": {}
},
"aggs": {
  "age_count": {
    "terms": {
      "field": "age",
      "size": 10
    }
  }
}
}
```

----End

## Other Operations

- **Obtaining the IP address of a CSS cluster**
  - a. In the navigation pane on the left, choose **Clusters**.
  - b. In the cluster list, locate a cluster, and obtain the IP address of the CSS cluster from the **Private Network Address** column. Generally, the IP address format is *<host>:<port>* or *<host>:<port>,<host>:<port>*.  
If the cluster has only one node, the IP address and port number of only one node are displayed, for example, **10.62.179.32:9200**. If the cluster has multiple nodes, the IP addresses and port numbers of all nodes are displayed, for example, **10.62.179.32:9200,10.62.179.33:9200**.
- **Obtaining the security certificate of a CSS cluster**
  - a. Log in to the CSS management console.
  - b. In the navigation pane, choose **Clusters**. The cluster list is displayed.
  - c. Click the name of a cluster to go to the cluster details page.
  - d. On the **Configuration** page, click **Download Certificate** next to **HTTPS Access**.

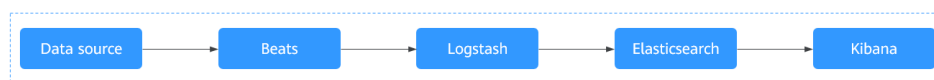
## 4.2 Using CSS to Build a Unified Log Management Platform

A unified log management platform built using CSS can manage logs in real time in a unified and convenient manner, enabling log-driven O&M and improving service management efficiency.

### Overview

Elasticsearch, Logstash, Kibana, and Beats (ELKB) provides a complete set of log solutions and is a mainstream log system. The following figure shows its framework.

**Figure 4-2** Unified log management platform framework



- Beats is a lightweight log collector, comprising Filebeat and Metricbeat.
- Logstash collects and preprocesses logs. It supports multiple data sources and ETL processing modes.
- Elasticsearch is an open-source distributed search engine that collects, analyzes, and stores data. CSS allows you to create Elasticsearch clusters.
- Kibana is a visualization tool used to perform web-based visualized query and make BI reports.

This section describes how to use CSS, Filebeat, Logstash, and Kibana to build a unified log management platform. Filebeat collects ECS logs and sends the logs to Logstash for data processing. The processing results are stored in CSS, and can be queried, analyzed, and visualized using Kibana.

For details about the version compatibility of ELKB components, see [https://www.elastic.co/support/matrix#matrix\\_compatibility](https://www.elastic.co/support/matrix#matrix_compatibility).

## Prerequisites

- A CSS cluster in non-security mode has been created.
- You have applied for an ECS and installed the Java environment on it.

## Procedure

### Step 1 Deploy and configure Filebeat.

1. Download Filebeat. The recommended version is 7.6.2. Download it at <https://www.elastic.co/downloads/past-releases#filebeat-oss>.
2. Configure the Filebeat configuration file **filebeat.yml**.

For example, to collect all the files whose names end with **log** in the **/root/** directory, configure the **filebeat.yml** file is as follows:

```
filebeat.inputs:
- type: log
  enabled: true
  # Path of the collected log file
  paths:
  - /root/*.log

filebeat.config.modules:
  path: ${path.config}/modules.d/*.yml
  reload.enabled: false
# Logstash hosts information
output.logstash:
  hosts: ["192.168.0.126:5044"]

processors:
```

### Step 2 Deploy and configure Logstash.

#### NOTE

To achieve better performance, you are advised to set the JVM parameter in Logstash to half of the ECS or docker memory.

1. Download Logstash. The recommended version is 7.6.2. Download it at <https://www.elastic.co/downloads/past-releases#logstash-oss>.
2. Ensure that Logstash can communicate with the CSS cluster.
3. Configure the Logstash configuration file **logstash-sample.conf**.

The content of the `logstash-sample.conf` file is as follows:

```
input {
  beats {
    port => 5044
  }
}
# Split data.
filter {
  grok {
    match => {
      "message" => "[%{GREEDYDATA:timemaybe}] [%{WORD:level}] %{GREEDYDATA:content}"
    }
  }
  mutate {
    remove_field => ["@version","tags","source","input","prospector","beat"]
  }
}
# CSS cluster information
output {
  elasticsearch {
    hosts => ["http://192.168.0.4:9200"]
    index => "%{[@metadata][beat]}-%{+YYYY.MM.dd}"
    #user => "xxx"
    #password => "xxx"
  }
}
```

#### NOTE

You can use Grok Debugger (<http://grokdebug.herokuapp.com/>) to configure the **filter** mode of Logstash.

### Step 3 Configure the index template of the CSS cluster on Kibana or via API.

For example, create an index template. Let the index use three shards and no replicas. Fields such as **@timestamp**, **content**, **host.name**, **level**, **log.file.path**, **message** and **timemaybe** are defined in the index.

```
PUT _template/filebeat
{
  "index_patterns": ["filebeat*"],
  "settings": {
    # Define the number of shards.
    "number_of_shards": 3,
    # Define the number of copies.
    "number_of_replicas": 0,
    "refresh_interval": "5s"
  },
  # Define a field.
  "mappings": {
    "properties": {
      "@timestamp": {
        "type": "date"
      },
      "content": {
        "type": "text"
      },
      "host": {
        "properties": {
          "name": {
            "type": "text"
          }
        }
      },
      "level": {
        "type": "keyword"
      },
      "log": {
```

```

    "properties": {
      "file": {
        "properties": {
          "path": {
            "type": "text"
          }
        }
      }
    },
    "message": {
      "type": "text"
    },
    "timemaybe": {
      "type": "date",
      "format": "yyyy-MM-dd HH:mm:ss|epoch_millis"
    }
  }
}

```

**Step 4** Prepare test data on ECS.

Run the following command to generate test data and write the data to **/root/tmp.log**:

```
bash -c 'while true; do echo [$(date)] [info] this is the test message; sleep 1; done;' >> /root/tmp.log &
```

The following is an example of the generated test data:

```
[Thu Feb 13 14:01:16 CST 2020] [info] this is the test message
```

**Step 5** Run the following command to start Logstash:

```
nohup ./bin/logstash -f /opt/pht/logstash-6.8.6/logstash-sample.conf &
```

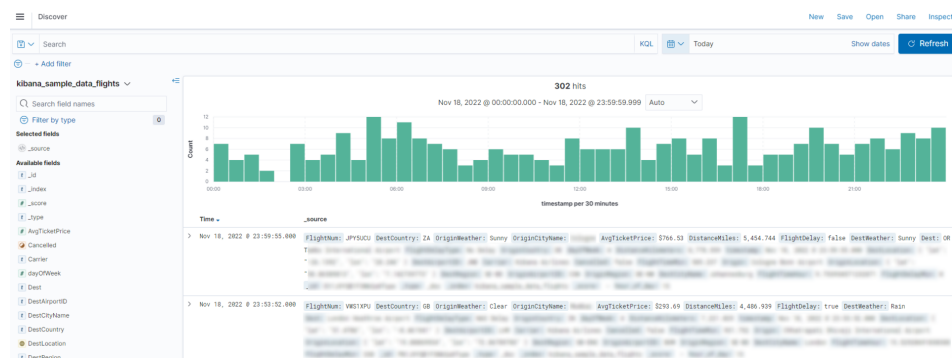
**Step 6** Run the following command to start Filebeat:

```
./filebeat
```

**Step 7** Use Kibana to query data and create reports.

1. Go to the Kibana page of the CSS cluster.
2. Click **Discover** and perform query and analysis, as shown in the following figure.

**Figure 4-3** Discover page



----End

## 4.3 Configuring Query Scoring in an Elasticsearch Cluster

You can score matched documents in an Elasticsearch cluster. This section describes how to configure query scoring.

### Overview

You can score a query in either of the following ways:

- Calculate the final scores (**new\_score**) of query results based on **vote** and sort the results in descending order.

**new\_score** = **query\_score** x (**vote** x **factor**)

- **query\_score**: calculated based on the total number of search keywords found in a record. A record earns 1 point for each keyword it contains.
- **vote**: vote of a record.
- **factor** : user-defined weight of **vote**.

- Calculate the final scores (**new\_score**) of query results based on **inline** and sort the results in descending order.

**new\_score** = **query\_score** x **inline**

- **query\_score**: calculated based on the total number of search keywords found in a record. A record earns 1 point for each keyword it contains.
- **vote**: vote of a record.
- **inline**: Configure two value options for this parameter and a threshold for **vote**. One option is used if **vote** exceeds the threshold, and the other is used if **vote** is smaller than or equal to the threshold. In this way, the query accuracy will not be affected by abnormal **vote** values.

### Prerequisites

An Elasticsearch cluster has been created on the CSS management console and is available.

### Procedure

#### NOTE

The code examples in this section can only be used for clusters Elasticsearch 7.x or later.

1. Log in to the CSS management console.
2. In the navigation pane on the left, click **Clusters** to go to the Elasticsearch cluster list.
3. Click **Access Kibana** in the **Operation** column of a cluster.
4. In the navigation tree on the left of Kibana, choose **Dev Tools**. The command execution page is displayed.
5. Create an index and specify a custom mapping to define the data type.  
For example, the content of the **tv.json** file is as follows:

```
{
  "tv":[
    { "name": "tv1", "description": "USB, DisplayPort", "vote": 0.98 }
    { "name": "tv2", "description": "USB, HDMI", "vote": 0.99 }
    { "name": "tv3", "description": "USB", "vote": 0.5 }
    { "name": "tv4", "description": "USB, HDMI, DisplayPort", "vote": 0.7 }
  ]
}
```

Run the following command to create the **mall** index and specify the user-defined mapping to define the data type:

```
PUT /mall?pretty
{
  "mappings": {
    "properties": {
      "name": {
        "type": "text",
        "fields": {
          "keyword": {
            "type": "keyword"
          }
        }
      },
      "description": {
        "type": "text",
        "fields": {
          "keyword": {
            "type": "keyword"
          }
        }
      },
      "vote": {
        "type": "float"
      }
    }
  }
}
```

## 6. Import data.

Run the following command to import data in the **tv.json** file to the **mall** index:

```
POST /mall/_bulk?pretty
{ "index": { "_id": "1" } }
{ "name": "tv1", "description": "USB, DisplayPort", "vote": 0.98 }
{ "index": { "_id": "2" } }
{ "name": "tv2", "description": "USB, HDMI", "vote": 0.99 }
{ "index": { "_id": "3" } }
{ "name": "tv3", "description": "USB", "vote": 0.5 }
{ "index": { "_id": "4" } }
{ "name": "tv4", "description": "USB, HDMI, DisplayPort", "vote": 0.7 }
```

## 7. Query data by using custom scoring. The query results can be scored based on **vote** or **inline**.

Assume a user wants to query TVs with USB, HDMI, and/or DisplayPort ports. The final query score can be calculated in the following ways and used for sorting:

- Scoring based on **vote**

The score is calculated using the formula **new\_score = query\_score x (vote x factor)**. Run the following command:

```
GET /mall/_doc/_search?pretty
{
  "query":{
    "function_score":{
      "query":{
        "bool":{
```



```

    "should": [
      {"match": {"description": "USB"}},
      {"match": {"description": "HDMI"}},
      {"match": {"description": "DisplayPort"}}
    ]
  },
  "field_value_factor": {
    "field": "vote",
    "factor": 1
  },
  "boost_mode": "multiply",
  "max_boost": 10
}
}
}

```

The query results are displayed in descending order of the score. The command output is as follows:

```

{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 0.8388366,
    "hits" : [
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "4",
        "_score" : 0.8388366,
        "_source" : {
          "name" : "tv4",
          "description" : "USB, HDMI, DisplayPort",
          "vote" : 0.7
        }
      },
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "2",
        "_score" : 0.7428025,
        "_source" : {
          "name" : "tv2",
          "description" : "USB, HDMI",
          "vote" : 0.99
        }
      },
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 0.7352994,
        "_source" : {
          "name" : "tv1",
          "description" : "USB, DisplayPort",
          "vote" : 0.98
        }
      },
      {
        "_index" : "mall",

```

```
  "_type": "_doc",
  "_id": "3",
  "_score": 0.03592815,
  "_source": {
    "name": "tv3",
    "description": "USB",
    "vote": 0.5
  }
}
```

– Scoring based on **inline**

The score is calculated using the formula **new\_score = query\_score x inline**. In this example, if **vote** > 0.8, the value of **inline** is 1. If **vote** ≤ 0.8, the value of **inline** is 0.5. Run the following command:

```
GET /mall/_doc/_search?pretty
{
  "query": {
    "function_score": {
      "query": {
        "bool": {
          "should": [
            {"match": {"description": "USB"}},
            {"match": {"description": "HDMI"}},
            {"match": {"description": "DisplayPort"}}
          ]
        }
      },
      "script_score": {
        "script": {
          "params": {
            "threshold": 0.8
          },
          "inline": "if (doc[\"vote\"].value > params.threshold) {return 1;} return 0.5;"
        }
      },
      "boost_mode": "multiply",
      "max_boost": 10
    }
  }
}
```

The query results are displayed in descending order of the score. The command output is as follows:

```
{
  "took": 4,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 4,
      "relation": "eq"
    },
    "max_score": 0.75030553,
    "hits": [
      {
        "_index": "mall",
        "_type": "_doc",
        "_id": "1",
        "_score": 0.75030553,
        "_source": {
```

```
"name" : "tv1",
"description" : "USB, DisplayPort",
"vote" : 0.98
},
{
  "_index" : "mall",
  "_type" : "_doc",
  "_id" : "2",
  "_score" : 0.75030553,
  "_source" : {
    "name" : "tv2",
    "description" : "USB, HDMI",
    "vote" : 0.99
  }
},
{
  "_index" : "mall",
  "_type" : "_doc",
  "_id" : "4",
  "_score" : 0.599169,
  "_source" : {
    "name" : "tv4",
    "description" : "USB, HDMI, DisplayPort",
    "vote" : 0.7
  }
},
{
  "_index" : "mall",
  "_type" : "_doc",
  "_id" : "3",
  "_score" : 0.03592815,
  "_source" : {
    "name" : "tv3",
    "description" : "USB",
    "vote" : 0.5
  }
}
]
}
```