Cloud Search Service

Best Practices

Issue	01
Date	2025-12-25

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property 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 Cloud 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.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Elasticsearch Data Migration

1.1 About Elasticsearch Cluster Migration Solutions

Table 1-1 Elasticsearch cluster migration solutions

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
Migrating data between Huawei Cloud Elasticsearch clusters	Huawei Cloud Logstash	• Migrate data from a Huawei Cloud Elasticsearch cluster of an earlier version to one of a later version. • Migrate data from multiple Huawei Cloud Elasticsearch clusters to a single such cluster to form a unified data platform.	During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration.	**Migrating Data Between Elasticsearch Clusters Using Huawei Cloud Logstash**

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
	Backup and restoration	• Migrate data between Huawei Cloud Elasticsearch clusters in the same region or across regions, and under the same or different accounts. • Migrate data from a Huawei Cloud Elasticsearch cluster of an earlier version to one of a later version. • Migrate data from multiple Huawei Cloud Elasticsearch clusters to a single such cluster to form a unified data platform.	• The version of the destination cluster must not be earlier than that of the source cluster. For details, see **Snapshot version compatibility**. • The number of nodes in the destination cluster must be greater than half of that in the source cluster, and cannot be less than the number of shard replicas in the source cluster. • The CPU, memory, and disk capacities of the destination cluster must not be lower than those of the source cluster.	**Migrating Data Between Huawei Cloud Elasticsearch Clusters Using Backup and Restoration**

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
	Read/ write splitting plugin	• Migrate data between Huawei Cloud Elasticsearch clusters in the same region or across regions, and under the same or different accounts. • Migrate data from multiple Huawei Cloud Elasticsearch clusters to a single such cluster to form a unified data platform.	The versions of the source and destination clusters must both be 7.6.2 or 7.10.2.	**Migrating Data Between Huawei Cloud Elasticsearch Clusters Using the Read/ Write Splitting Plugin**
	Reindex API	Migrate data from multiple Huawei Cloud Elasticsearch clusters to a single such cluster to form a unified data platform.	During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration.	**Migrating Data Between Elasticsearch Clusters Using the Reindex API**

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
	ESM	• Migrate data from a Huawei Cloud Elasticsearch cluster of an earlier version to one of a later version. • Migrate data from multiple Huawei Cloud Elasticsearch clusters to a single such cluster to form a unified data platform.	During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration.	**Migrating Data Between Elasticsearch Clusters Using ESM**
Migrating data from an in-house built Elasticsearch cluster to Huawei Cloud	Huawei Cloud Logstash	• Migrate data from an on-premises Elasticsearch cluster of an earlier version to a Huawei Cloud Elasticsearch cluster of a later version. • Migrate data from multiple on-premises Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform. • Migrate in-house built Elasticsearch workloads to Huawei Cloud.	• During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration. • Ensure that the network between the clusters is connected. Configure public network connectivity for in-house built Elasticsearch cluster.	**Migrating Data Between Elasticsearch Clusters Using Huawei Cloud Logstash**

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
	Backup and restoration	• Migrate data from an on-premises Elasticsearch cluster of an earlier version to a Huawei Cloud Elasticsearch cluster of a later version. • Migrate data from multiple on-premises Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform. • Migrate in-house built Elasticsearch workloads to Huawei Cloud.	• The version of the destination cluster must not be earlier than that of the source cluster. For details, see **Snapshot version compatibility**. • This method does not support incremental data synchronization. You need to pause data update before starting to back up data. • An on-premises Elasticsearch cluster needs public network access to back up snapshots to OBS.	**Migrating Data from an On-premises Elasticsearch Cluster to Huawei Cloud Using the S3 Plugin**
	Reindex API	• Migrate data from multiple on-premises Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform. • Migrate in-house built Elasticsearch workloads to Huawei Cloud.	During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration.	**Migrating Data Between Elasticsearch Clusters Using the Reindex API**

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
	ESM	• Migrate data from an on-premises Elasticsearch cluster of an earlier version to a Huawei Cloud Elasticsearch cluster of a later version. • Migrate data from multiple on-premises Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform. • Migrate in-house built Elasticsearch workloads to Huawei Cloud.	During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration.	**Migrating Data Between Elasticsearch Clusters Using ESM**
	CDM	This is a cloud migration tool provided by Huawei Cloud to migrate data between clusters that belong to different cloud services.	• A VPN or Direct Connect connection needs to be established between the customer's IDC and Huawei Cloud. • During cluster migration, do not delete any of the source cluster's indexes, or something might go wrong.	

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
Migrating data from a third-party Elasticsearch cluster to Huawei Cloud	Huawei Cloud Logstash	• Migrate Elasticsearch clusters across major versions, for example, from 6.X to 7.X. • Migrate data from multiple third-party Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform. • Migrate third-party Elasticsearch workloads to Huawei Cloud.	• During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration. • Ensure that the network between the clusters is connected. A VPN or Direct Connect connection needs to be established between the customer's IDC and Huawei Cloud.	**Migrating Data Between Elasticsearch Clusters Using Huawei Cloud Logstash**

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
	Backup and restoration	• Migrate data from a third-party Elasticsearch cluster of an earlier version to a Huawei Cloud Elasticsearch cluster of a later version. • Migrate data from multiple third-party Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform. • Migrate third-party Elasticsearch workloads to Huawei Cloud.	• The version of the destination cluster must not be earlier than that of the source cluster. For details, see **Snapshot version compatibility**. • This method does not support incremental data synchronization. You need to pause data update before starting to back up data. • Snapshots can be migrated only when public access is enabled for third-party storage repositories.	**Migrating Data from a Third-Party Elasticsearch Cluster to Huawei Cloud Using Backup and Restoration**
	Reindex API	• Migrate data from multiple third-party Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform. • Migrate third-party Elasticsearch workloads to Huawei Cloud.	During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration.	**Migrating Data Between Elasticsearch Clusters Using the Reindex API**

Migration Scenario	Migration Tool/ Method	Applicable Scenario	Constraints	Example
	ESM	• Migrate data from a third-party Elasticsearch cluster of an earlier version to a Huawei Cloud Elasticsearch cluster of a later version. • Migrate data from multiple third-party Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform. • Migrate third-party Elasticsearch workloads to Huawei Cloud.	During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration.	**Migrating Data Between Elasticsearch Clusters Using ESM**
	CDM	This is a cloud migration tool provided by Huawei Cloud to migrate data between clusters that belong to different cloud services.	• A VPN or Direct Connect connection needs to be established between the customer's IDC and Huawei Cloud. • During cluster migration, do not delete any of the source cluster's indexes, or something might go wrong.	

1.2 Migrating Data Between Elasticsearch Clusters Using Huawei Cloud Logstash

You can use a Logstash cluster created using Huawei Cloud's Cloud Search Service (CSS) to migrate data between Elasticsearch clusters.

Scenarios

Huawei Cloud Logstash is a fully managed data ingestion and processing service. It is compatible with open-source Logstash and can be used for data migration between Elasticsearch clusters.

You can use Huawei Cloud Logstash to migrate data from Huawei Cloud Elasticsearch, self-built Elasticsearch, or third-party Elasticsearch to Huawei Cloud Elasticsearch. This solution applies to the following scenarios:

- Cross-version migration: Migrate data between different versions to maintain data availability and consistency in the new version, leveraging the compatibility and flexibility of Logstash. This mode applies to migration scenarios where the Elasticsearch cluster version span is large, for example, from 6.X to 7.X.
- Cluster merging: Utilize Logstash to transfer and consolidate data from multiple Elasticsearch clusters into a single Elasticsearch cluster, enabling unified management and analysis of data from different Elasticsearch clusters.
- Cloud migration: Migrate an on-premises Elasticsearch service to the cloud to enjoy the benefits of cloud services, such as scalability, ease-of-maintenance, and cost-effectiveness.
- Changing the service provider: An enterprise currently using a third-party Elasticsearch service wishes to switch to Huawei Cloud for reasons such as cost, performance, or other strategic considerations.

Solution Architecture

Figure 1-1 Migration process

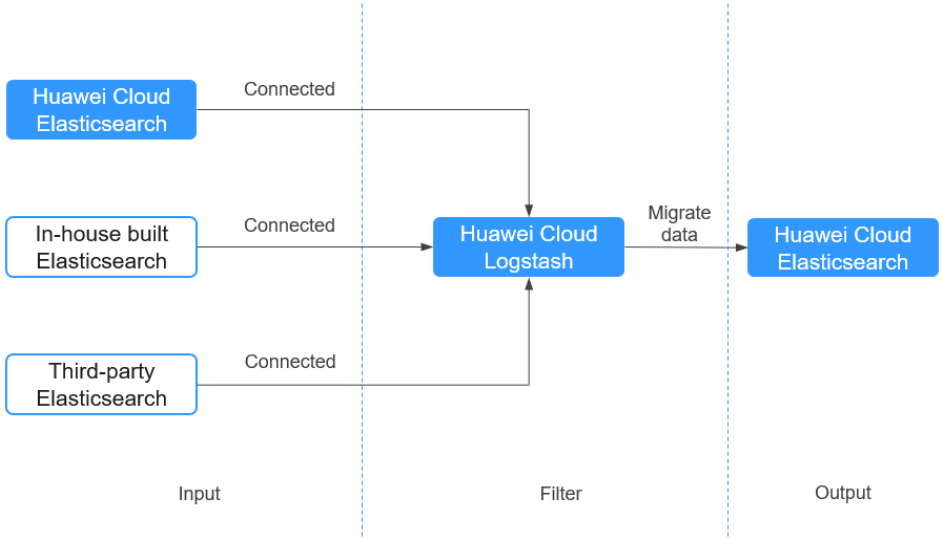


Figure 1-1 shows how to migrate data between Elasticsearch clusters using Huawei Cloud Logstash.

1. Input: Huawei Cloud Logstash receives data from Huawei Cloud Elasticsearch, self-built Elasticsearch, or third-party Elasticsearch.

   NOTE

   The procedure for migrating data from Huawei Cloud Elasticsearch, self-built Elasticsearch, or third-party Elasticsearch to Huawei Cloud Elasticsearch remains the same. The only difference is the access address of the source cluster. For details, see **Obtaining Elasticsearch Cluster Information**.

2. Filter: Huawei Cloud Logstash cleanses and converts data.
3. Output: Huawei Cloud Logstash outputs data to the destination system, for example, Huawei Cloud Elasticsearch.

You can choose between full data migration or incremental data migration based on service requirements.

- **Full data migration**: Utilize Logstash to perform a complete data migration. This method is suitable for the initial phase of migration or scenarios where data integrity must be ensured.
- **Incremental data migration**: Configure Logstash to perform incremental queries and migrate only index data with incremental fields. This method is suitable for scenarios requiring continuous data synchronization or real-time data updates.

Advantages

- Compatibility with later versions: This solution supports the migration of Elasticsearch clusters across different versions.
- Efficient data processing: Logstash supports batch read and write operations, significantly enhancing data migration efficiency.
- Concurrent synchronization technology: The slice concurrent synchronization technology can be utilized to boost data migration speed and performance, especially when handling large volumes of data.
- Simple configuration: Huawei Cloud Logstash offers a straightforward and intuitive configuration process, allowing you to input, process, and output data through configuration files.
- Powerful data processing: Logstash includes various built-in filters to clean, convert, and enrich data during migration.
- Flexible migration policies: You can choose between full migration or incremental migration based on service requirements to optimize storage usage and migration time.

Impact on Performance

Using Logstash for data migration between clusters relies on the Scroll API. This API can efficiently retrieve index data from the source cluster and synchronize the data to the destination cluster in batches. This process may impact the performance of the source cluster. The specific impact depends on how fast data is retrieved from the source cluster, and the data retrieval speed depends on the **size** and **slice** settings of the Scroll API. For details, see the **Reindex API** document.

- If the source cluster has a high resource usage, it is advisable to tune the **size** parameter to slow down the data retrieval speed or perform the migration during off-peak hours, reducing impact on the performance of the source cluster.
- If the source cluster has a low resource usage, keep the default settings of the Scroll API. In the meantime, monitor the load of the source cluster. Tune the **size** and **slice** parameters based on the load conditions of the source cluster, optimizing migration efficiency and resource utilization.

Constraints

During cluster migration, do not add, delete, or modify indexes in the source cluster, or the source and destination clusters will have inconsistent data after the migration.

Prerequisites

- The source and destination Elasticsearch clusters are available.
- You have created a CSS Logstash cluster (for example, **Logstash-ES**) and confirmed that Logstash and the Elasticsearch clusters involved are deployed in the same VPC and the network between them is connected.
  - If the source cluster, Logstash, and destination cluster are in different VPCs, establish a VPC peering connection between them. For details, see .
  - To migrate an on-premises Elasticsearch cluster to Huawei Cloud, you can configure public network access for the on-premises Elasticsearch cluster.
  - To migrate a third-party Elasticsearch cluster to Huawei Cloud, you need to establish a VPN or Direct Connect connection between the third party's internal data center and Huawei Cloud.
- Ensure that **_source** has been enabled for indexes in the cluster.

  By default, **_source** is enabled. You can run the **GET {index}/_search** command to check whether it is enabled. If the returned index information contains **_source**, it is enabled.

Procedure

1. **Obtaining Elasticsearch Cluster Information**.
2. **(Optional) Migrating the Index Structure**: Migrate an Elasticsearch cluster's index template and index structure using scripts.
3. **Verifying Connectivity Between Clusters**: Verify the connectivity between the Logstash cluster and the source Elasticsearch cluster.
4. Migrate the source Elasticsearch cluster using Logstash.
   - **Using Logstash to Perform Full Data Migration** is recommended at the initial stage of cluster migration or in scenarios where data integrity needs to be ensured.
   - **Using Logstash to Incrementally Migrate Cluster Data** is recommended for scenarios that require continuous data synchronization or real-time data updates.
5. **Deleting a Logstash Cluster**: After the cluster migration is complete, release the Logstash cluster in a timely manner.

Obtaining Elasticsearch Cluster Information

Before migrating a cluster, you need to obtain necessary cluster information for configuring a migration task.

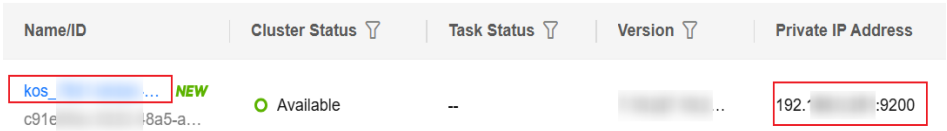
Table 1-2 Required Elasticsearch cluster information

Cluster Source		Required Information	How to Obtain
Source cluster	Huawei Cloud Elasticsearch cluster	• Name of the source cluster • Access address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	• For details about how to obtain the cluster name and access address, see 3. • Contact the service administrator to obtain the username and password.
	Self-built Elasticsearch cluster	• Name of the source cluster. • Public network address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	Contact the service administrator to obtain the information.
	Third-party Elasticsearch cluster	• Name of the source cluster. • Access address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	Contact the service administrator to obtain the information.
Destination cluster	Huawei Cloud Elasticsearch cluster	• Access address of the destination cluster • Username and password for accessing the destination cluster (only for security-mode clusters)	• For details about how to obtain the access address, see 3. • Contact the service administrator to obtain the username and password.

The method of obtaining the cluster information varies depending on the source cluster. This section describes how to obtain information about the Huawei Cloud Elasticsearch cluster.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters** > **Elasticsearch**.
3. In the cluster list, find the destination cluster and obtain the cluster name and address.

Figure 1-2 Obtaining cluster information



(Optional) Migrating the Index Structure

If you plan to manually create an index structure in the destination Elasticsearch cluster, skip this section. This section describes how to migrate an Elasticsearch cluster's index template and index structure using scripts.

1. Create an ECS to migrate the metadata of the source cluster.
   a. Create an ECS. Select CentOS as the OS of the ECS and 2U4G as its flavor. The ECS must be in the same VPC and security group as the CSS cluster.
   b. Test the connectivity between the ECS and the source and destination clusters.

      Run the **curl http:// {ip}:{port}** command on the ECS to test the connectivity. If 200 is returned, the connection is successful.

      IP indicates the access address of the source or destination cluster. **port** indicates the port number. The default port number is **9200**. Use the actual port number of the cluster.
2. Install Python for executing the migration script.

   Select a proper Python installation method based on the ECS (executor) environment.

Table 1-3 Python installation methods

Python Version	Internet Connection	Operation Guide
Python3	Yes	**Installing Python 3 Online**
	No	**Installing Python 3 Offline**
Python2	Yes	**Installing Python 2 Online**
	No	**Installing Python 2 Offline**

- **Install Python 3 online.**

  Install Python 3 using yum and pip.

  i. Install Python 3.

  ```
  [root@ecs opt]# yum install python3
  ```


ii. Install PIP.

```
[root@ecs opt]# yum install python3-pip zlib-devel
```

iii. Install yaml dependencies.

```
[root@ecs opt]# pip3 install pyyaml
```

iv. Install requests dependencies.

```
[root@ecs opt]# pip3 install requests
```

v. Confirm a successful installation.

```
# Check the Python version.
[root@ecs opt]# python3 --version
Python 3.8.0
# Check the pip version.
[root@ecs opt]# pip3 --version
pip 9.0.3 from /usr/lib/python3.8/site-packages (python 3.8)
```

- **Install Python 3 offline.**

  If ECS is not connected to the Internet, download the installation package to the ECS and run the installation commands.

  i. Download the python3 installation package from **https://www.python.org/downloads/release/python-380/**. Download and install the source code.

  Figure 1-3 Downloading the python3 package

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		e18a9d1a0a6d858b9787e03fc6fdaa20	22.8 MB	SIG
XZ compressed source tarball	Source release		dbac8df9d8b9edc678d0f4cacdb7dbb0	17.0 MB	SIG
macOS 64-bit installer	macOS	for OS X 10.9 and later	f5f9ae9f416170c6355cab7256bb75b5	27.7 MB	SIG
Windows help file	Windows		1c33359821033ddb3353c8e5b6e7e003	8.1 MB	SIG

  ii. Use WinSCP to upload the Python installation package to the **opt** directory and install Python.

```
# Extract the Python package.
[root@ecs-52bc opt]# tar -xvf Python-3.8.0.tar.xz
Python-3.8.0/Modules/zlib/crc32.c
Python-3.8.0/Modules/zlib/gzlib.c
Python-3.8.0/Modules/zlib/inffast.c
Python-3.8.0/Modules/zlib/example.c
Python-3.8.0/Modules/python.c
Python-3.8.0/Modules/nismodule.c
Python-3.8.0/Modules/Setup.config.in
...
# Enter the installation directory.
[root@ecs-52bc opt]# cd Python-3.8.0
# Check the file configuration installation path.
[root@ecs-52bc Python-3.8.0]# ./configure --prefix=/usr/local/python3 --with-zlib
...
checking for build directories... checking for --with-computed-gotos... no value specified
checking whether gcc -pthread supports computed gotos... yes
done
checking for ensurepip... no
configure: creating ./config.status
config.status: creating Makefile.pre
config.status: creating Modules/Setup.config
config.status: creating Misc/python.pc
config.status: creating Modules/ld_so_aix
config.status: creating pyconfig.h
creating Modules/Setup
creating Modules/Setup.local
creating Makefile
# Compile Python.
[root@ecs-52bc Python-3.8.0]# make
# Install Python.
[root@ecs-52bc Python-3.8.0]# make install
```


iii. Confirm a successful installation.

```
# Check the Python version.
[root@ecs-52bc Python-3.8.0]# python3 --version
Python 3.8.0
# Check the pip version.
[root@ecs-52bc Python-3.8.0]# pip3 --version
pip 9.0.3 from /usr/lib/python3.8/site-packages (python 3.8)
```

- **Install Python 2 online.**

  Install Python 2 using yum and pip.

  i. Install Python 2.

  ```
  [root@ecs opt]# yum install python2
  ```

  ii. Install PIP.

  ```
  [root@ecs opt]# yum install python-pip
  ```

  iii. Install yaml dependencies.

  ```
  [root@ecs opt]# pip install pyyaml
  ```

  iv. Install requests dependencies.

  ```
  [root@ecs opt]# pip install requests
  ```

  v. Confirm a successful installation.

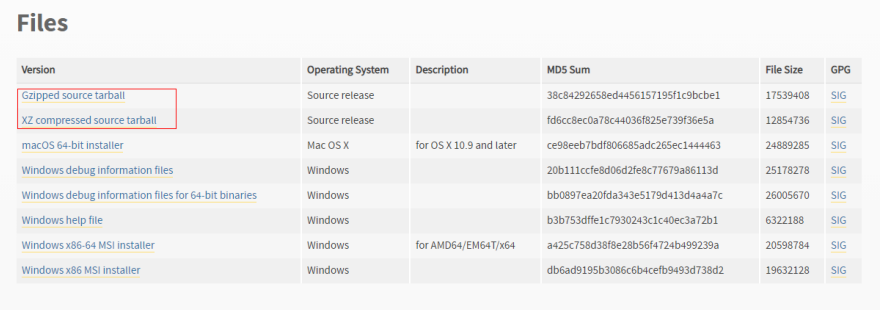
  ```
  # Check the Python version.
  [root@ecs opt]# python --version
  Python 2.7.5
  # Check the pip version.
  [root@ecs opt]# pip --version
  pip 7.1.2 from /usr/lib/python2.7/site-packages/pip-7.1.2-py2.7.egg (python 2.7)
  ```

- **Install Python 2 offline.**

  If the ECS is not connected to the Internet, download the installation package to the ECS and run the installation commands.

  i. Download the python2 installation package from **https://www.python.org/downloads/release/python-2718/**. Download and install the source code.

  Figure 1-4 Downloading the python2 package

  ii. Use WinSCP to upload the Python installation package to the **opt** directory and install Python.

  ```
  # Extract the Python package.
  [root@ecs-52bc opt]# tar -xvf Python-2.7.18.tgz
  Python-2.7.18/Modules/zlib/crc32.c
  Python-2.7.18/Modules/zlib/gzlib.c
  Python-2.7.18/Modules/zlib/inffast.c
  Python-2.7.18/Modules/zlib/example.c
  Python-2.7.18/Modules/python.c
  Python-2.7.18/Modules/nismodule.c
  Python-2.7.18/Modules/Setup.config.in
  ...
  # Enter the installation directory.
  [root@ecs-52bc opt]# cd Python-2.7.18
  ```



```
# Check the file configuration installation path.
[root@ecs-52bc Python-2.7.18]# ./configure --prefix=/usr/local/python2
...
checking for build directories... checking for --with-computed-gotos... no value specified
checking whether gcc -pthread supports computed gotos... yes
done
checking for ensurepip... no
configure: creating ./config.status
config.status: creating Makefile.pre
config.status: creating Modules/Setup.config
config.status: creating Misc/python.pc
config.status: creating Modules/ld_so_aix
config.status: creating pyconfig.h
creating Modules/Setup
creating Modules/Setup.local
creating Makefile
# Compile Python.
[root@ecs-52bc Python-2.7.18]# make
# Install Python.
[root@ecs-52bc Python-2.7.18]# make install
```

iii. Confirm a successful installation.

```
# Check the Python version.
[root@ecs-52bc Python-2.7.18]# python --version
Python 2.7.5
# Check the pip version.
[root@ecs-52bc Python-2.7.18]# pip --version
pip 7.1.2 from /usr/lib/python2.7/site-packages/pip-7.1.2-py2.7.egg (python 2.7)
```

3. Prepare the index migration script of the source Elasticsearch cluster.

   The following provides a sample of the script. Modify it based on service requirements.

   a. Run the **vi migrateConfig.yaml** command. Modify the following content based on site requirements, and run the **wq** command to save the file as **Logstash migration script**: For details about how to obtain the cluster information, see **Obtaining Elasticsearch Cluster Information**.

```
es_cluster_new:
# Name of the source cluster
  clustername: es_cluster_new
# Access address of the source cluster, plus http://.
  src_ip: http://x.x.x.x:9200
# Username and password for accessing the source cluster. For a non-security-mode cluster, set the value to "".
  src_username: ""
  src_password: ""
# Access address of the destination Elasticsearch cluster, plus http://.
  dest_ip: http://x.x.x.x:9200
# Username and password for accessing the destination Elasticsearch cluster. For a non-security-mode cluster, set the value to "".
  dest_username: ""
  dest_password: ""
# only_mapping is an optional parameter with a default value of false. It must be used in conjunction with migrateMapping.py to specify whether to process only the index of the mapping address in the file. When set to true, only the index data that matches the mapping key in the source cluster is migrated. When set to false, all index data, except for .kibana and .*, is migrated from the source cluster.
# During migration, the index name is compared with the provided mapping. If a match is found, the mapping value is used as the index name in the destination cluster. If no match is found, the original index name from the source cluster is retained.
  only_mapping: false
# Set the index to be migrated. key indicates the index name of the source cluster, and value indicates the index name of the destination cluster.
  mapping:
    test_index_1: test_index_1
```

   b. Run the **vi migrateTemplate.py** command. Copy the following script, and run the **wq** command to save it as an **index template migration script**.


```
# -*- coding:UTF-8 -*-
import json
import os
import sys

import requests
import yaml

requests.packages.urllib3.disable_warnings()


def loadConfig(argv):
    if argv is None or len(argv) != 2:
        config_yaml = "migrateConfig.yaml"
    else:
        config_yaml = argv[1]
    config_file = open(config_yaml)
    return yaml.safe_load(config_file)


def put_template_to_target(url, template, cluster, template_name, dest_auth=None):
    headers = {'Content-Type': 'application/json'}
    create_resp = requests.put(url, headers=headers, data=json.dumps(template),
auth=dest_auth, verify=False)
    if not os.path.exists("templateLogs"):
        os.makedirs("templateLogs")
    if create_resp.status_code != 200:
        print("create template " + url + " failed with response: " + str(
            create_resp) + ", source template is " + template_name)
        print(create_resp.text)
        filename = "templateLogs/" + str(cluster) + "#" + template_name
        with open(filename + ".json", "w") as f:
            json.dump(template, f)
        return False
    else:
        return True


def process_template(index_mapping):
    # remove unnecessary keys
    if "settings" in index_mapping and "index" in index_mapping["settings"]:
        # Remove routing configurations.
        if "routing" in index_mapping["settings"]["index"]:
            del index_mapping["settings"]["index"]["routing"]
        if "mapper" in index_mapping["settings"]["index"]:
            del index_mapping["settings"]["index"]["mapper"]
        if "lifecycle" in index_mapping["settings"]["index"]:
            del index_mapping["settings"]["index"]["lifecycle"]
    return index_mapping


def main(argv):
    print("begin to migration template!")
    config = loadConfig(argv)
    src_clusters = config.keys()

    print("begin to process cluster name : " + ','.join(src_clusters))
    print("cluster count : " + str(src_clusters.__len__()))

    for name, value in config.items():
        print("<============================================================>")
        source_user = value["src_username"]
        source_passwd = value["src_password"]
        source_auth = None
        if source_user != "":
            source_auth = (source_user, source_passwd)
        dest_user = value["dest_username"]
        dest_passwd = value["dest_password"]
        dest_auth = None
```



```
    if dest_user != "":
        dest_auth = (dest_user, dest_passwd)

    print("start to process cluster name: " + name)
    source_url = value["src_ip"] + "/_template"

    response = requests.get(source_url, auth=source_auth, verify=False)
    if response.status_code != 200:
        print("[error] get all template failed. resp statusCode:" + str(
            response.status_code) + " response is " + response.text)
        continue
    all_template = response.json()
    migrate_template = []

    for template in all_template.keys():
        if template.startswith(".") or template == "logstash":
            continue
        if "index_patterns" in all_template[template]:
            if any(s.startswith(".") for s in all_template[template]["index_patterns"]):
                print('[skip] skip (.*) template, cluster: %-10s, template %-30s ' % (str(name),
str(template)))
                continue
            template_content = process_template(all_template[template])
            migrate_template.append([template, template_content])

    for template, template_content in migrate_template:
        dest_index_url = value["dest_ip"] + "/_template/" + template
        result = put_template_to_target(dest_index_url, template_content, name, template,
dest_auth)
        if result is True:
            print('[success] migrate success, cluster: %-10s, template %-30s ' % (str(name),
str(template)))
        else:
            print('[failure] migrate failure, cluster: %-10s, template %-30s ' % (str(name),
str(template)))


if __name__ == '__main__':
    main(sys.argv)
```

c. Run the **vi migrateMapping.py** command. Copy the following script and run the **wq** command to save it as an **index structure migration script**.

```
# -*- coding:UTF-8 -*-
import json
import os
import sys

import requests
import yaml

requests.packages.urllib3.disable_warnings()


def loadConfig(argv):
    if argv is None or len(argv) != 2:
        config_yaml = "migrateConfig.yaml"
    else:
        config_yaml = argv[1]
    config_file = open(config_yaml)
    return yaml.safe_load(config_file)


def process_mapping(index_mapping, dest_index):
    # remove unnecessary keys
    del index_mapping["settings"]["index"]["provided_name"]
    del index_mapping["settings"]["index"]["uuid"]
    del index_mapping["settings"]["index"]["creation_date"]
    del index_mapping["settings"]["index"]["version"]
    # Remove routing configurations.
    if "routing" in index_mapping["settings"]["index"]:
```



```
        del index_mapping["settings"]["index"]["routing"]
    if "lifecycle" in index_mapping["settings"]["index"]:
        del index_mapping["settings"]["index"]["lifecycle"]
    # check alias
    aliases = index_mapping["aliases"]
    for alias in list(aliases.keys()):
        if alias == dest_index:
            print(
                    "source index[" + dest_index + "],alias[" + alias + "]is the same as dest_index
name, will remove this alias.")
            del index_mapping["aliases"][alias]
    # Configure lifecycle policies.
    if "lifecycle" in index_mapping["settings"]["index"]:
        lifecycle = index_mapping["settings"]["index"]["lifecycle"]
        opendistro = {"opendistro": {"index_state_management":
                                     {"policy_id": lifecycle["name"],
                                      "rollover_alias": lifecycle["rollover_alias"]}}}
        index_mapping["settings"].update(opendistro)
        del index_mapping["settings"]["index"]["lifecycle"]

    # replace synonyms_path
    if "analysis" in index_mapping["settings"]["index"]:
        analysis = index_mapping["settings"]["index"]["analysis"]
        if "filter" in analysis:
            filter = analysis["filter"]
            if "my_synonym_filter" in filter:
                my_synonym_filter = filter["my_synonym_filter"]
                if "synonyms_path" in my_synonym_filter:
                    del index_mapping["settings"]["index"]["analysis"]["filter"]["my_synonym_filter"]
["synonyms_path"]
    return index_mapping


def get_index(source, source_auth):
    response = requests.get(source + "/_alias", auth=source_auth)
    if response.status_code != 200:
        print("[error] get all index failed. resp statusCode:" + str(
            response.status_code) + " response is " + response.text)
        exit()
    all_index = response.json()
    system_index = []
    create_index = []
    for index in list(all_index.keys()):
        if index.startswith("."):
            system_index.append(index)
        else:
            create_index.append(index)
    return system_index, create_index


def get_mapping(source, source_index, source_auth):
    source_url = source + "/" + source_index
    response = requests.get(source_url, auth=source_auth, verify=False)
    if response.status_code != 200:
        print("[error] get ElasticSearch message failed. resp statusCode:" + str(
            response.status_code) + " response is " + response.text)
        return None
    return response.json()


def put_mapping_to_target(url, mapping, cluster, source_index, dest_auth=None):
    headers = {'Content-Type': 'application/json'}
    create_resp = requests.put(url, headers=headers, data=json.dumps(mapping),
auth=dest_auth, verify=False)
    if not os.path.exists("mappingLogs"):
        os.makedirs("mappingLogs")
    if create_resp.status_code != 200:
        print("[error] create index " + url + " failed with response: " + str(create_resp) + ", source
index is " + str(
```



```
            source_index))
        print(create_resp.text)
        filename = "mappingLogs/" + str(cluster) + "#" + str(source_index)
        with open(filename + ".json", "w") as f:
            json.dump(mapping, f)
        return False
    else:
        return True


def main(argv):
    print("begin to migrate index mapping!")
    config = loadConfig(argv)
    src_clusters = config.keys()

    print("begin to process cluster name : " + ','.join(src_clusters))
    print("cluster count : " + str(src_clusters.__len__()))

    for name, value in config.items():
        print("<============================================================>")
        source = value["src_ip"]
        source_user = value["src_username"]
        source_passwd = value["src_password"]
        source_auth = None
        if source_user != "":
            source_auth = (source_user, source_passwd)
        dest = value["dest_ip"]
        dest_user = value["dest_username"]
        dest_passwd = value["dest_password"]
        dest_auth = None
        if dest_user != "":
            dest_auth = (dest_user, dest_passwd)

        print("start to process cluster:   " + name)

        indices = []
        # only deal with mapping list
        if 'only_mapping' in value and value["only_mapping"]:
            for source_index, dest_index in value["mapping"].items():
                indices.append([source_index, dest_index])
        else:
            # get all indices
            system_index, create_index = get_index(source, source_auth)
            for index in create_index:
                dest_index = index
                if 'mapping' in value:
                    if index in value["mapping"].keys():
                        dest_index = value["mapping"][index]
                indices.append([index, dest_index])

        success_index = 0
        for source_index, dest_index in indices:
            mapping = get_mapping(source, source_index, source_auth)
            if mapping is None:
                print("[failure] cluster name:" + name + ", " + source_index)
                continue
            index_mapping = process_mapping(mapping[source_index], dest_index)
            dest_url = dest + "/" + dest_index
            result = put_mapping_to_target(dest_url, index_mapping, name, source_index,
dest_auth)
            if result is False:
                print("[failure] migrate mapping cluster name: " + name + ", " + source_index)
                continue
            print("[success] migrate mapping cluster name: " + name + ", " + source_index)
            success_index = success_index + 1
        print("create index mapping success total: " + str(success_index))
```



```
if __name__ == '__main__':
    main(sys.argv)
```

d. Run the **vi checkIndices.py** command. Copy the following script and run the **wq** command to save it as an **index data comparison script**.

```
# -*- coding:UTF-8 -*-
import sys

import requests
import yaml

requests.packages.urllib3.disable_warnings()


def loadConfig(argv):
    if argv is None or len(argv) != 2:
        config_yaml = "migrateConfig.yaml"
    else:
        config_yaml = argv[1]
    config_file = open(config_yaml)
    return yaml.safe_load(config_file)


# get all indices
def get_indices(url, source_auth):
    response = requests.get(url + "/_alias", auth=source_auth)
    if response.status_code != 200:
        print("[error] get all index failed. resp statusCode:" + str(
            response.status_code) + " response is " + response.text)
        exit()
    all_index = response.json()
    system_index = []
    create_index = []
    for index in list(all_index.keys()):
        if (index.startswith(".")):
            system_index.append(index)
        else:
            create_index.append(index)
    return create_index


def get_index_total(url, index, es_auth):
    stats_url = url + "/" + index + "/_stats"
    index_response = requests.get(stats_url, auth=es_auth, verify=False)
    if index_response.status_code != 200:
        print("[error] get ElasticSearch stats message failed. resp statusCode:" + str(
            index_response.status_code) + " response is " + index_response.text)
        return 0
    return index_response.json()


def main(argv):
    print("begin to check index documents count!")
    config = loadConfig(argv)
    src_clusters = config.keys()

    print("begin to process cluster name : " + ','.join(src_clusters))
    print("cluster count : " + str(src_clusters.__len__()))

    for name, value in config.items():
        print("<============================================================>")
        source = value["src_ip"]
        source_user = value["src_username"]
        source_passwd = value["src_password"]
        source_auth = None
        if source_user != "":
            source_auth = (source_user, source_passwd)
        dest = value["dest_ip"]
        dest_user = value["dest_username"]
        dest_passwd = value["dest_password"]
```



```
    dest_auth = None
    if dest_user != "":
        dest_auth = (dest_user, dest_passwd)
    cluster_name = name
    if "clustername" in value:
        cluster_name = value["clustername"]

    print("start to process cluster: " + cluster_name)
    # get all indices
    all_source_index = get_indices(source, source_auth)
    all_dest_index = get_indices(dest, dest_auth)

    print("source indices total      : " + str(all_source_index.__len__()))
    print("destination indices total : " + str(all_dest_index.__len__()))

    for index in all_source_index:
        index_total = get_index_total(value["src_ip"], index, source_auth)
        src_total = index_total["_all"]["primaries"]["docs"]["count"]
        src_size = int(index_total["_all"]["primaries"]["store"]["size_in_bytes"]) / 1024 / 1024
        dest_index = get_index_total(value["dest_ip"], index, dest_auth)
        if dest_index == 0:
            print('[failure] not found, index: %-20s, source total: %-10s size %6sM'
                  % (str(index), str(src_total), src_size))
            continue
        dest_total = dest_index["_all"]["primaries"]["docs"]["count"]
        if src_total != dest_total:
            print('[failure] not consistent, '
                  'index: %-20s, source total: %-10s size %6sM destination total: %-10s '
                  % (str(index), str(src_total), src_size, str(dest_total)))
            continue
        print('[success] compare index total equal : index : %-20s,  total: %-20s '
              % (str(index), str(dest_total)))


if __name__ == '__main__':
    main(sys.argv)
```

4. Run the following commands to migrate the index template and index structure of the Elasticsearch cluster:

```
python migrateTemplate.py
python migrateMapping.py
```

Verifying Connectivity Between Clusters

Before starting a migration task, verify the network connectivity between Logstash and the source Elasticsearch cluster.

1. Go to the **Configuration Center** page of Logstash.
   a. Log in to the **CSS management console**.
   b. In the navigation pane on the left, choose **Clusters > Logstash**.
   c. In the cluster list, click the name of the target cluster. The cluster information page is displayed.
   d. Click the **Configuration Center** tab.
2. On the **Configuration Center** page, click **Test Connectivity**.
3. In the dialog box that is displayed, enter the IP address and port number of the source cluster and click **Test**.

Figure 1-5 Testing connectivity

If **Available** is displayed, the network is connected. If the network is disconnected, configure routes for the Logstash cluster to connect the clusters. For details, see .

Using Logstash to Perform Full Data Migration

At the initial stage of cluster migration, or in scenarios where guaranteeing data integrity takes top priority, it is recommended to use Logstash for full data migration. This approach migrates the entire Elasticsearch cluster's data in one go.

1. Go to the **Configuration Center** page of Logstash.
   a. Log in to the **CSS management console**.
   b. In the navigation pane on the left, choose **Clusters > Logstash**.
   c. In the cluster list, click the name of the target cluster. The cluster information page is displayed.
   d. Click the **Configuration Center** tab.
2. On the **Configuration Center** page, click **Create** in the upper right corner. On the **Create Configuration File** page, edit the configuration file for the full Elasticsearch cluster migration.
   a. **Selecting a cluster template**: Expand the system template list, select **elasticsearch**, and click **Apply** in the **Operation** column.
   b. **Setting the name of the configuration file**: Set **Name**, for example, **es-es-all**.
   c. **Editing the configuration file**: Enter the migration configuration plan of the Elasticsearch cluster in **Configuration File Content**. The following is an example of the configuration file: For details about how to obtain the cluster information, see **Obtaining Elasticsearch Cluster Information**.

```
input{
    elasticsearch{
        # Address for accessing the source Elasticsearch cluster.
        hosts =>  ["xx.xx.xx.xx:9200", "xx.xx.xx.xx:9200"]
# Username and password for accessing the source cluster. You do not need to configure them for a non-security-mode cluster.
        # user => "css_logstash"
        # password => "*****"
# Configure the indexes to be migrated. Use commas (,) to separate multiple indexes. You can use wildcard characters, for example, index*.
        index => "*_202102"
        docinfo => true
        slices => 3
        size => 3000
# If the destination cluster is accessed through HTTPS, you need to configure the following information:
# HTTPS access certificate of the cluster. Retain the following for the CSS cluster:
        # ca_file => "/rds/datastore/logstash/v7.10.0/package/logstash-7.10.0/extend/certs"  # for 7.10.0
```



```
# Whether to enable HTTPS communication. Set this parameter to true for a cluster accessed through HTTPS.
        #ssl => true
    }
}


# Remove specified fields added by Logstash.
filter {
  mutate {
    remove_field => ["@version"]
  }
}

output{
    elasticsearch{
# Access address of the destination Elasticsearch cluster
      hosts => ["xx.xx.xx.xx:9200","xx.xx.xx.xx:9200"]
# Username and password for accessing the destination cluster. You do not need to configure them for a non-security-mode cluster.
      # user => "css_logstash"
      # password => "*****"
# Configure the index of the target cluster. The following configuration indicates that the index name is the same as that of the source end.
      index => "%{[@metadata][_index]}"
      document_type => "%{[@metadata][_type]}"
      document_id => "%{[@metadata][_id]}"
# If the destination cluster is accessed through HTTPS, you need to configure the following information:
# HTTPS access certificate of the cluster. Retain the following for the CSS cluster:
      #cacert => "/rds/datastore/logstash/v7.10.0/package/logstash-7.10.0/extend/certs"  # for 7.10.0
# Whether to enable HTTPS communication. Set this parameter to true for a cluster accessed through HTTPS.
      #ssl => true
# Whether to verify the elasticsearch certificate on the server. Set this parameter to false, indicating that the certificate is not verified.
      #ssl_certificate_verification => false
    }
}
```

Table 1-4 Configuration items for full migration

Item		Description
input	hosts	IP address of the source cluster. If the cluster has multiple access nodes, separate them with commas (,).
	user	Username for accessing the cluster. For a non-security-mode cluster, use # to comment out this parameter.
	password	Password for accessing the cluster. For a non-security-mode cluster, use # to comment out this item.
	index	The source indexes to be fully migrated. Use commas (,) to separate multiple indexes. Wildcard is supported, for example, ***index****.
	docinfo	Indicates whether to re-index the document. The value must be **true**.

Item		Description
	slices	In some cases, it is possible to improve overall throughput by consuming multiple distinct slices of a query simultaneously using sliced scrolls. It is recommended that the value range from 2 to 8.
	size	Maximum number of hits returned for each query
output	hosts	Access address of the destination cluster. If the cluster has multiple nodes, separate them with commas (,).
	user	Username for accessing the cluster. For a non-security-mode cluster, use # to comment out this parameter.
	password	Password for accessing the cluster. For a non-security-mode cluster, use # to comment out this item.
	index	Name of the index migrated to the destination cluster. It can be modified and expanded, for example, Logstash-*%{+yyyy.MM.dd}*.
	document_type	Ensure that the document type on the destination end is the same as that on the source end.
	document_id	Document ID in the index. It is advisable to keep consistent document IDs on the source and destination clusters. If you want to have document IDs automatically generated, use the number sign (#) to comment it out.

d. Click **Next** to configure Logstash pipeline parameters.

In this example, retain the default values. For details about how to set the parameters, see .

e. Click **OK**.

On the **Configuration Center** page, you can check the created configuration file. If its status changes to **Available**, it has been successfully created.

3. Execute the full migration task.

a. In the configuration file list, select configuration file **es-es-all** and click **Start** in the upper left corner.

b. In the **Start Logstash** dialog box, select **Keepalive** if necessary. In this example, **Keepalive** is not enabled.

When Keepalive is enabled, a daemon process is configured on each node. If the Logstash service becomes faulty, the daemon process will try to rectify the fault and restart the service, ensuring that the Logstash

pipelines run efficiently and reliably. You are advised to enable Keepalive for services running long-term. Do not enable it for services running only short-term, or your migration tasks may fail due to a lack of source data.

c. Click **OK** to execute the configuration file and thereby start the Logstash full migration task.

You can view the started configuration file in the pipeline list.

4. After data migration is complete, check data consistency.
   - Method 1: Use PuTTY to log in to the VM used for the migration and run the **python checkIndices.py** command to compare the data.
   - Method 2: Run the **GET _cat/indices** command on the Kibana console of the source and destination clusters, separately, to check whether their indexes are consistent.

Using Logstash to Incrementally Migrate Cluster Data

In scenarios where continuous data synchronization or real-time data is required, it is recommended to use Logstash incremental cluster data migration. This method involves configuring incremental queries in Logstash, allowing only index data with incremental fields to be migrated.

1. Go to the **Configuration Center** page of Logstash.
   a. Log in to the **CSS management console**.
   b. In the navigation pane on the left, choose **Clusters > Logstash**.
   c. In the cluster list, click the name of the target cluster. The cluster information page is displayed.
   d. Click the **Configuration Center** tab.
2. On the **Configuration Center** page, click **Create** in the upper right corner. On the **Create Configuration File** page, edit the configuration file for the incremental migration.
   a. **Selecting a cluster template**: Expand the system template list, select **elasticsearch**, and click **Apply** in the **Operation** column.
   b. **Setting the name of the configuration file**: Set **Name**, for example, **es-es-inc**.
   c. **Editing the configuration file**: Enter the migration configuration plan of the Elasticsearch cluster in **Configuration File Content**. The following is an example of the configuration file:

      The incremental migration configuration varies according to the index and must be provided based on the index analysis. For details about how to obtain the cluster information, see **Obtaining Elasticsearch Cluster Information**.

```
input{
    elasticsearch{
# Access address of the source Elasticsearch cluster. You do not need to add a protocol. If you add the HTTPS protocol, an error will be reported.
        hosts =>  ["xx.xx.xx.xx:9200"]
# Username and password for accessing the source cluster. You do not need to configure them for a non-security-mode cluster.
        user => "css_logstash"
        password => "******"
# Configure incremental migration indexes.
        index => "*_202102"
```



```
# Configure incremental migration query statements.
        query => '{"query":{"bool":{"should":[{"range":{"postsDate":{"from":"2021-05-25
00:00:00"}}}]}}}'
        docinfo => true
        size => 1000
# If the destination cluster is accessed through HTTPS, you need to configure the following
information:
        # HTTPS access certificate of the cluster. Retain the following for the CSS cluster:
        # ca_file => "/rds/datastore/logstash/v7.10.0/package/logstash-7.10.0/extend/certs"  # for
7.10.0
# Whether to enable HTTPS communication. Set this parameter to true for HTTPS-based cluster
access.
        #ssl => true
    }
}

filter {
  mutate {
    remove_field => ["@timestamp", "@version"]
  }
}

output{
    elasticsearch{
# Access address of the destination cluster.
        hosts => ["xx.xx.xx.xx:9200","xx.xx.xx.xx:9200"]
# Username and password for accessing the destination cluster. You do not need to configure
them for a non-security-mode cluster.
        #user => "admin"
        #password => "******"
# Configure the index of the target cluster. The following configuration indicates that the index
name is the same as that of the source end.
        index => "%{[@metadata][_index]}"
        document_type => "%{[@metadata][_type]}"
        document_id => "%{[@metadata][_id]}"
# If the destination cluster is accessed through HTTPS, you need to configure the following
information:
# HTTPS access certificate of the cluster. Retain the default value for the CSS cluster.
        #cacert => "/rds/datastore/logstash/v7.10.0/package/logstash-7.10.0/extend/certs"  # for
7.10.0
# Whether to enable HTTPS communication. Set this parameter to true for HTTPS-based cluster
access.
        #ssl => true
# Whether to verify the elasticsearch certificate on the server. Set this parameter to false,
indicating that the certificate is not verified.
        #ssl_certificate_verification => false
    }

    #stdout { codec => rubydebug { metadata => true }}
}
```

Table 1-5 Incremental migration configuration items

Configuration	Description
hosts	Access addresses of the source and target clusters. If a cluster has multiple nodes, enter all their access addresses.
user	Username for accessing the cluster. For a non-security-mode cluster, use # to comment out this parameter.

Configuration	Description
password	Password for accessing the cluster. For a non-security-mode cluster, use # to comment out this item.
index	Indexes to be incrementally migrated. One configuration file supports the incremental migration of only one index.
query	Identifier of incremental data. Typically, it is a DLS statement of Elasticsearch and needs to be analyzed in advance. **postsDate** indicates the time field in the service. `{"query":{"bool":{"should":[{"range":{"postsDate":{"from":"2021-05-25 00:00:00"}}}]}}}` This command means to migrate data added after 2021-05-25. During multiple incremental migrations, you need to change the log value. If the indexes in the source end Elasticsearch use the timestamp format, convert the data to a timestamp here. The validity of this command must be verified in advance.
scroll	If there is massive data on the source end, you can use the scroll function to obtain data in batches to prevent Logstash memory overflow. The default value is **1m**. The interval cannot be too long. Otherwise, data may be lost.

3. Execute the incremental migration task.
   a. In the configuration file list, select configuration file **es-es-inc** and click **Start** in the upper left corner.
   b. In the **Start Logstash** dialog box, select **Keepalive** if necessary. In this example, **Keepalive** is not enabled.

      When Keepalive is enabled, a daemon process is configured on each node. If the Logstash service becomes faulty, the daemon process will try to rectify the fault and restart the service, ensuring that the Logstash pipelines run efficiently and reliably. You are advised to enable Keepalive for services running long-term. Do not enable it for services running only short-term, or your migration tasks may fail due to a lack of source data.
   c. Click **OK** to execute the configuration file and thereby start the Logstash incremental migration task.

      You can view the started configuration file in the pipeline list.
4. After data migration is complete, check data consistency.
   - Method 1: Use PuTTY to log in to the VM used for the migration and run the **python checkIndices.py** command to compare the data.
   - Method 2: Run the **GET _cat/indices** command on the Kibana console of the source and destination clusters, separately, to check whether their indexes are consistent.

Deleting a Logstash Cluster

After the migration is complete, release the Logstash cluster in a timely manner to save resources and avoid unnecessary fees.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Logstash**.
3. In the Logstash cluster list, select the Logstash cluster **Logstash-ES** and click **More > Delete** in the **Operation** column. In the confirmation dialog box, manually type in **DELETE**, and click **OK**.

1.3 Migrating Data Between Huawei Cloud Elasticsearch Clusters Using Backup and Restoration

Data can be migrated between CSS Elasticsearch clusters by backing up and restoring cluster snapshots.

Scenarios

This option can be used only when both the source and destination clusters are CSS clusters that rely on Object Storage Service (OBS). Typical application scenarios include:

- Cross-region or cross-account migration: Migrate the data of an Elasticsearch cluster in another region or under another account to the current cluster.
- Cross-version migration: Migrate data from a self-built Elasticsearch cluster of an earlier version to a cluster of a later version.
- Cluster merge: Merge the index data of two Elasticsearch clusters.

Solution Architecture

Figure 1-6 Migration procedure

Figure 1-6 shows the process of migrating data between Huawei Cloud Elasticsearch clusters using backup and restoration.

1. Create a snapshot for the source Elasticsearch cluster and store the snapshot in an OBS bucket.
2. Restore the snapshot to the destination Elasticsearch cluster.

Advantages

- Easy operation and management: The cluster snapshot function on the CSS console allows for simple, easy-to-manage, and automatic data backup and restoration.
- Applicable to large-scale data migration: Snapshot backup is suitable for scenarios involving large amounts of data, especially when the data volume reaches GB, TB, or even PB levels.
- Cross-region and cross-account migration: With the cross-region replication function of OBS, data can be migrated across different regions and accounts.
- Controllable restoration process: During data restoration, you can restore specific indexes or all indexes and specify the cluster status to be restored.
- Controllable migration duration: The data migration rate can be configured based on the migration duration evaluation formula. Ideally, the data migration rate matches the file replication rate.

Impact on Performance

This migration method works by copying data directly from the storage layer. It does not rely on any external Elasticsearch APIs. Hence it significantly reduces any impact on the performance of the source cluster. For latency-insensitive workloads, the impact is negligible.

Constraints

- The version of the destination cluster must not be earlier than that of the source cluster. For details, see **Snapshot version compatibility**.
- The number of nodes in the destination cluster must be greater than half of that in the source cluster, and cannot be less than the number of shard replicas in the source cluster.
- The CPU, memory, and disk capacities of the destination cluster must not be lower than those of the source cluster.

Migration Duration

The number of nodes or index shards in the source and destination clusters determines how long the data migration will take. Data migration consists of two phases: data backup and restoration. The backup duration is determined by the source cluster and the restoration duration is determined by the destination cluster. The formula for calculating the total migration duration is as follows:

- If the number of index shards is greater than the number of nodes:

  Total duration (in seconds) = Size of migrated data (in GB)/40 MB (0.04 GB)/(Number of nodes in the source cluster + Number of nodes in the destination cluster) x Number of indexes
- If the number of index shards is smaller than the number of nodes:

  Total duration (in seconds) = Size of migrated data (in GB)/40 MB (0.04 GB)/(Number of index shards in the source cluster + Number of index shards in the destination cluster) x Number of indexes

NOTE

The migration duration estimated using the formula is the minimal duration possible (if each node transmits data at the fastest speed, 40 MB/s). The actual duration also depends on factors such as the network and resources condition.

Prerequisites

- The destination cluster (**Es-2**) and source cluster (**Es-1**) are available. You are advised to migrate a cluster during off-peak hours.
- Ensure that the destination cluster (**Es-2**) and source cluster (**Es-1**) are in the same region.

  If the cluster is deployed across regions or accounts, copy the OBS bucket that stores snapshots for the source cluster to that for the destination cluster. For details, see . Then, restore the snapshots in the **destination cluster**.
- Ensure that an OBS bucket is available for storing snapshots. The OBS bucket must be in the same region as the Elasticsearch clusters, and the storage class must be **Standard**.

Procedure

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, click **Es-1**, the name of the source cluster. The cluster information page is displayed.
4. Select the **Cluster Snapshots** tab. Click **Enable Snapshot**. In the displayed dialog box, configure basic snapshot settings.

Table 1-6 Enabling snapshots

Parameter	Description
OBS Bucket	From the drop-down list, select an OBS bucket for storing snapshots.
Backup Path	Snapshot storage path in the OBS bucket. You can retain the default value.
Maximum Backup Rate (per Second)	The parameter sets the maximum backup rate per node. When it is exceeded, flow control is triggered to prevent excessive resource usage and ensure system stability. The actual backup rate may not reach the configured value, as it depends on factors such as OBS performance and disk I/O. You can use the default value 40 MB.

Parameter	Description

Maximum Recovery Rate (per Second)	The parameter sets the maximum recovery rate per node. When it is exceeded, flow control is triggered to prevent excessive resource usage and ensure system stability. The actual recovery rate may not reach the configured value, as it depends on factors such as OBS performance and disk I/O. The recommended value is 40 MB.
	For Elasticsearch clusters later than 7.6.2, the recovery rate is also limited by the **indices.recovery.max_bytes_per_sec** parameter.
	● If **Maximum Recovery Rate (per Second)** is less than **indices.recovery.max_bytes_per_sec**, the former takes effect.
	● If **Maximum Recovery Rate (per Second)** is greater than **indices.recovery.max_bytes_per_sec**, the latter takes effect.
	NOTE
	● To check the value of **indices.recovery.max_bytes_per_sec**, run the following command:
	`GET _cluster/settings`
	● To modify **indices.recovery.max_bytes_per_sec**, run the following command:
	`PUT _cluster/settings`
	`{`
	`  "transient": {`
	`  "indices.recovery.max_bytes_per_sec": "100mb"`
	`  }`
	`}`

IAM Agency	Select an IAM agency to grant the current account the permission to access and use OBS. To store snapshots to an OBS bucket, you must have the required OBS access permissions. You are advised to use the **css_obs_agency** agency created automatically. If an agency has been created automatically, you can click **One-click Authorization** to grant minimal permissions. **WARNING** The agency name can contain only letters (case-sensitive), digits, underscores (_), and hyphens (-). Otherwise, the backup will fail.
Automatic Snapshot Creation	For a data migration task, you are advised not to enable automatic snapshot creation. This is to avoid occupying storage resources.

5. Click **OK** to enable cluster snapshots.
6. Under **Cluster Snapshot Tasks**, click **Manually Create Snapshot**. In the displayed dialog box, configure the snapshot policy.

Table 1-7 Parameters for manually creating a snapshot

Parameter	Description
Snapshot Name	Set the snapshot name. You can retain the default value.
Index	Specify the name of the index to be backed up. • You can back up a specified index. To specify multiple indexes, use commas (,) to separate them, for example, **index1,index2,index3**. • You can use an asterisk (*) to match multiple indexes. For example, **index*** indicates that all indexes with the prefix **index** will be backed up. If you do not specify this parameter, all indexes in the cluster are backed up by default. The index name cannot contain spaces, uppercase letters, or special characters **"\<|>/?**
Snapshot Description	Add a snapshot description.

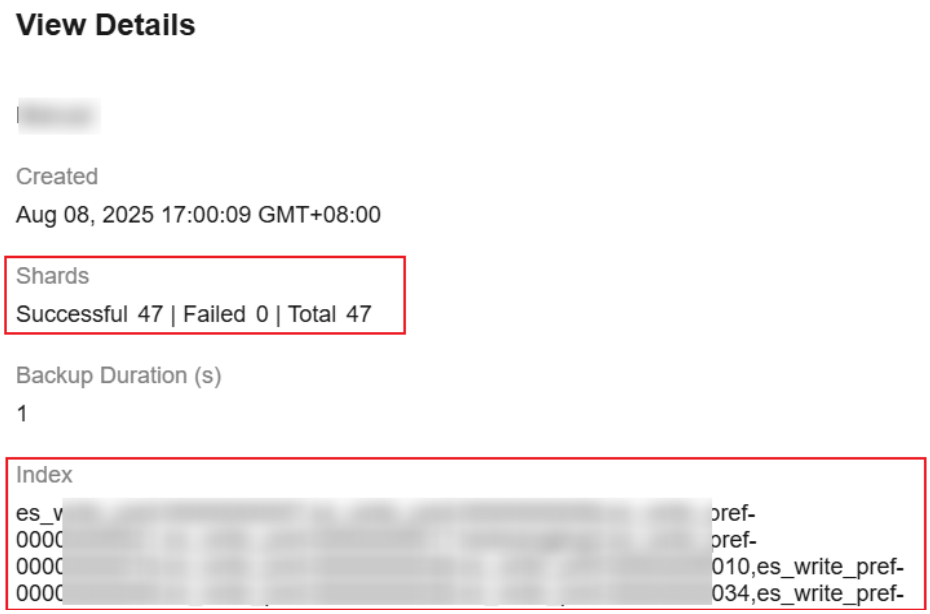
7. Click **OK** to start creating a snapshot for the source cluster.

   In the cluster snapshot task list, if **Snapshot Status** changes to **Available**, the snapshot has been created.

8. Check whether data is successfully backed up.

   In the cluster snapshot task list, click the snapshot name. The **View Details** dialog box is displayed. Check the shards and indexes that have been backed up to see if the backup is successful.

   Figure 1-7 View Details

9. In the cluster snapshot task list, select a snapshot, and click **Restore** in the **Operation** column. In the displayed dialog box, configure necessary settings.

NOTE

- **Retaining the original index name when restoring an index**

  Specify **Index** for the index to be replaced.

- **Renaming an index when restoring it**

  To rename an index upon restoring it, specify **Index**, **Rename Pattern**, and **Rename Replacement**.

Table 1-8 Snapshot restoration parameters

Parameter	Description
Index	Specify the name of the index you want to restore. • The value is a string of 0 to 1024 characters that cannot contain uppercase letters, spaces, or the following special characters: **"\<|>/?**. • When restoring an index whose name is prefixed with **.kibana**, the index name must be specified. • The **.opendistro_security** index cannot be restored. • You can use an asterisk (*) to match multiple indexes. For example, **index*** indicates that all indexes with the prefix **index** will be restored. When an asterisk (*) is used for index matching, the **.opendistro_security** index and any system indexes whose name is prefixed with **.kibana** are filtered out by default. • You can restore indexes by specifying their names, for example, **index1,index2,index3**. By default, this parameter is left blank. That is, no index name is specified, and all indexes will be restored.
Rename Pattern	Index name matching rule. Enter a regular expression. Indexes that match the regular expression will be restored. **Rename Pattern** and **Rename Replacement** take effect only when they are both configured at the same time. The value is a string of 0 to 1024 characters that cannot contain uppercase letters, spaces, or the following special characters: **"\<|>/?,** For example, **index_(.+)** indicates that all indexes whose name starts with **index_** will be renamed upon restoration.
Rename Replacement	Index renaming rule. Upon restoration, matching indexes are renamed according to the defined rule. **Rename Pattern** and **Rename Replacement** take effect only when they are both configured at the same time. The value is a string of 0 to 1024 characters that cannot contain uppercase letters, spaces, or the following special characters: **"\<|>/?,** For example, **restored_index_$1** indicates that **restored_** will prefix the name of all restored indexes.

Parameter	Description
Cluster	Select a destination cluster, for example, **Es-2**.
Overwrite same-name indexes in destination cluster	Whether to overwrite same-name indexes in the destination cluster. We recommend keeping it unselected.

10. Click **OK** to restore data to the destination cluster **Es-2**.

    In the snapshot list, when **Task Status** changes to **Restoration succeeded**, the data migration is complete.

11. After the data migration is complete, check the data consistency between the destination Elasticsearch cluster **Es-2** and source Elasticsearch cluster **Es-1**. For example, run the **_cat/indices** command in the source and destination clusters, separately, to check whether their indexes are consistent.
12. (Optional) After the migration, promptly delete the OBS bucket used for storing snapshots if these snapshots are no longer needed. This is to prevent ongoing storage costs.

1.4 Migrating Data from an On-premises Elasticsearch Cluster to Huawei Cloud Using the S3 Plugin

This topic describes how to use the S3 plugin to migrate data from an on-premises Elasticsearch cluster to a Huawei Cloud Elasticsearch cluster using the snapshot mechanism.

Scenarios

The S3 plugin (repository-s3) is purpose-developed for Elasticsearch. It allows users to store Elasticsearch snapshots to a storage service compatible with S3 APIs, such as Huawei Cloud OBS. The S3 plugin provides an efficient, flexible, and secure way to back up the data of Elasticsearch clusters.

The S3 plugin can be used to migrate data from an on-premises Elasticsearch cluster to a Huawei Cloud Elasticsearch cluster. Typical scenarios include:

- Cloud migration: Migrate an on-premises Elasticsearch service to the cloud to enjoy the benefits of cloud services, such as scalability, ease-of-maintenance, and cost-effectiveness.
- Cross-version migration: Migrate data from an on-premises Elasticsearch cluster of an earlier version to a Huawei Cloud Elasticsearch cluster of a later version.
- Cluster merge: Migrate data from multiple on-premises Elasticsearch clusters to a single Huawei Cloud Elasticsearch cluster to form a unified data platform for simplified management and higher data consistency.

- Unified technology stack: For a simplified, unified technology stack, companies that are already using some services on Huawei Cloud may choose to migrate their on-premises Elasticsearch clusters to Huawei Cloud.

Solution Architecture

Figure 1-8 Migration procedure

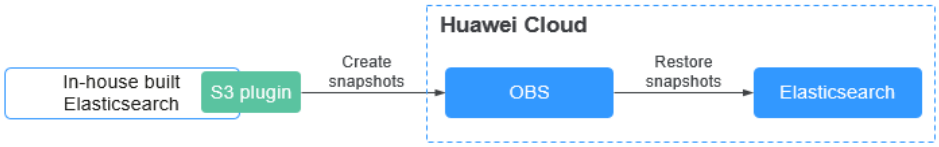


Figure 1-8 describes the procedure for migrating an on-premises Elasticsearch cluster (source cluster) to a Huawei Cloud Elasticsearch cluster (destination cluster) using the S3 plugin.

1. Install the repository-s3 plugin for the on-premises Elasticsearch cluster.
2. Back up the data of the on-premises Elasticsearch cluster to Huawei Cloud OBS.
3. Restore data from Huawei OBS to an Elasticsearch cluster in CSS.

Advantages

- High cross-version compatibility: Data can be migrated between Elasticsearch clusters of different versions, including upgrade from an earlier version to a later version.
- High availability and durability: S3 protects the security of data backups and minimizes the risk of data loss.
- Flexible backup policies: You can choose between full backup or incremental backup based on service requirements to optimize storage utilization and migration time.

Impact on Performance

This migration method works by copying data directly from the storage layer. It does not rely on any external Elasticsearch APIs. Hence it significantly reduces any impact on the performance of the source cluster. For latency-insensitive workloads, the impact is negligible.

Constraints

- The version of the destination cluster must not be earlier than that of the source cluster. For details, see **Snapshot version compatibility**.
- This method does not support incremental data synchronization. You need to pause data update before starting to back up data.
- An on-premises Elasticsearch cluster needs public network access to back up snapshots to OBS.

Prerequisites

- The source and destination Elasticsearch clusters are available.
- The OBS bucket (backup-obs) for storing snapshots has been created. The OBS bucket and the destination Elasticsearch cluster in CSS are in the same **Region**, and the **Storage Class** is **Standard**.

Procedure

Step 1 Install the repository-s3 plugin for the on-premises Elasticsearch cluster.

1. Check the version of the on-premises Elasticsearch cluster and determine the version of the repository-s3 plugin that you need to install.

   Run the following command to obtain cluster information:

   ```
   curl http://x.x.x.x:9200 # Enter the IP address of the Elasticsearch cluster.
   ```

   Find **version.number** in the command output. In the example below, the cluster version is 7.10.2.

   ```
   {
     "name" : "es_cluster_migrate-ess-esn-1-1",
     "cluster_name" : "es_cluster_migrate",
     "cluster_uuid" : "1VbP7-39QNOx_R-llXKKtA",
     "version" : {
       "number" : "7.10.2",
       "build_flavor" : "default",
       "build_type" : "tar",
       "build_hash" : "d2ef93d",
       "build_date" : "2018-12-17T21:17:40.758843Z",
       "build_snapshot" : false,
       "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"
   }
   ```

2. Download the repository-s3 plugin installation package of the required version. 7.10.2 is used as an example here. Change the version number in the download address based on the actual cluster version.

   Download address: **https://artifacts.elastic.co/downloads/elasticsearch-plugins/repository-s3/repository-s3-7.10.2.zip**

3. Log in to an instance node of the on-premises Elasticsearch cluster, and run the **cd** command to go to the plugins directory.

4. Run the following command to extract the repository-s3 plugin installation package to the plugins directory:

   ```
   unzip  repository-s3-7.10.2.zip -d plugins/repository-s3
   ```

5. Add the AK/SK that enables access to Huawei Cloud to the repository-s3 plugin.

   a. Go to the installation path of the on-premises Elasticsearch cluster, and run the following command to create a keystore file: If the keystore file already exists and can be used properly, skip this step. (If you still run the command below, select **N** to avoid overwriting the existing file.)

      ```
      bin/elasticsearch-keystore create s3.keystore
      ```


> **CAUTION**
>
> Use a common user account, rather than the **root** account, to create the keystore file. Otherwise, the configuration will not take effect.

b. Add the AK/SK that enables access to Huawei Cloud to the keystore file.

```
bin/elasticsearch-keystore add s3.client.default.access_key
bin/elasticsearch-keystore add s3.client.default.secret_key
```

access_key indicates the AK, and **secret_key** indicates the SK.

6. After the AK/SK is added, restart the cluster node to apply the change.
7. If the on-premises Elasticsearch cluster has multiple instance nodes, repeat **Step 1.3** to **Step 1.6** to install the S3 plugin and configure the AK/SK for all these nodes.
8. After the restart, run the following command to check whether the S3 plugin has been successfully installed.

```
curl http://x.x.x.x:9200/_cat/plugins # Enter the IP address of the Elasticsearch cluster.
```

If the correct version information of the repository-s3 plugin is returned, as shown below, the plugin has been successfully installed.

```
node-1 repository-s3 7.10.2
```

Step 2 Back up the data of the on-premises Elasticsearch cluster to Huawei Cloud OBS.

1. Log in to the Kibana console of the on-premises Elasticsearch cluster, and choose **Dev Tools** to go to the CLI.
2. Run the following command to create a snapshot repository **my_backup**, and connect it to OBS bucket **backup-obs** for storing snapshots in Huawei Cloud OBS.

```
PUT /_snapshot/my_backup
{
  "type": "s3",
  "settings": {
    # OBS bucket name
    "bucket": "backup-obs",
    # Public network address for accessing OBS
    "endpoint": "obs.xxx.example.com",
    "base_path": "snapshot",
    "max_snapshot_bytes_per_sec": "1000mb"
  }
}
```


CAUTION

If the error information below is returned, the S3 plugin is incompatible with OBS. (OBS buckets do not support path-style requests.) The S3 plugin added support for virtual-hosted–style requests in version 7.4. In prior versions, it only supports path-style requests. The access methods supported by OBS buckets vary with the OBS version version. When this error occurs, choose a different migration solution. We recommend **Migrating Data Between Elasticsearch Clusters Using Huawei Cloud Logstash**.

```
{
   "error": {
      "root_cause": [
         {
            "type": "repository_verification_exception",
            "reason": "[my_backup] path [snapshotqa] is not accessible on master node"
         }
      ],
      "type": "repository_verification_exception",
      "reason": "[my_backup] path [snapshotqa] is not accessible on master node",
      "caused_by": {
         "type": "i_o_exception",
         "reason": "Unable to upload object [snapshotqa/tests-bXFMubHkRc6I6Bi0_MLaVw/
master.dat] using a single upload",
         "caused_by": {
            "type": "amazon_s3_exception",
            "reason": "amazon_s3_exception: Virtual host domain is required while accessing a specific
bucket. (Service: Amazon S3; Status Code: 403; Error Code: VirtualHostDomainRequired; Request
ID: 0000019A06082A2898E7F94824E63CD6; S3 Extended Request ID:
QBhRggr6ok5ah1Vg9wy6v6bdUIR8Z5T/L1HNX765EufYZh6rY5iKccH47xSTX/Ts)"
         }
      }
   },
   "status": 500
}
```

3. Run the following command. If correct information about the snapshot repository is returned, it has been successfully created.

```
GET /_snapshot/my_backup
```

4. Run the following command to create a snapshot.

   Exclude system indexes whose name starts with a period (.), and create a full data snapshot for all other indexes. Setting **wait_for_completion** to **true** means asynchronous task execution.

```
PUT  _snapshot/my_backup/snapshot_all?wait_for_completion=true
{
   "indices": "*,-.*"
}
```

5. After the snapshot is created, run the following command to check the snapshot information:

```
# Display all existing snapshots.
GET _cat/repositories
# Display information about all snapshots.
GET _snapshot/my_backup/_all
# Check the information of a snapshot by specifying its name. snapshot_all indicates the snapshot name.
GET _snapshot/my_backup/snapshot_all
# Check the status of a snapshot by specifying its name. snapshot_all indicates the snapshot name.
GET _snapshot/my_backup/snapshot_all/_status
```

Step 3 Log in to the Kibana console of a CSS Elasticsearch cluster.

1. Log in to the **CSS management console**.

2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
4. In the left navigation pane, choose **Dev Tools**.

   The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.

Step 4 Restore data from OBS to the CSS Elasticsearch cluster.

1. Run the following command on Kibana to create a snapshot repository called **my_backup_all**.

```
PUT _snapshot/my_backup_all/
{
   "type" : "obs",
   "settings" : {
      # Private domain name of OBS.
      "endpoint" : "obs.xxx.example.com",
      "region" : "cn-south-1",
      # 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" : "backup-obs",
      "compress" : "false",
      "chunk_size" : "1g",
      "base_path" : "snapshot",
      "max_restore_bytes_per_sec": "1000mb",
      "max_snapshot_bytes_per_sec": "1000mb"
   }
}
```

2. Run the following command to restore data from the snapshot on OBS to the destination cluster.
   - Check all existing snapshots. In the command below, **my_backup_all** indicates the repository name.

     ```
     GET _snapshot/my_backup_all/_all
     ```

   - Restore data from the **snapshot_all** snapshot in the **my_backup_all** snapshot repository to the destination Elasticsearch cluster.

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

   - Restore specified indexes to the destination Elasticsearch cluster. In this example, **my_index1** and **my_index2** are the names of the indexes to be restored.

     ```
     POST _snapshot/my_backup_all/snapshot_all/_restore?wait_for_completion=true
     {
       "indices": "my_index1,my_index2,-.*",
       "ignore_unavailable": "true"
     }
     ```

Step 5 After the data migration is complete, check the data consistency between the source and destination Elasticsearch clusters. For example, run the **_cat/indices** command in the source and destination clusters, separately, to check whether their indexes are consistent.

Step 6 (Optional) After the migration, promptly delete the OBS bucket used for storing snapshots if these snapshots are no longer needed. This is to prevent ongoing storage costs.

----End

1.5 Migrating Data from a Third-Party Elasticsearch Cluster to Huawei Cloud Using Backup and Restoration

You can use backup and restoration to migrate data from a third-party Elasticsearch cluster to a Huawei Cloud Elasticsearch cluster.

Scenarios

Data migration between a third-party Elasticsearch cluster and a Huawei Cloud Elasticsearch cluster through backup and restoration depends on storage repositories. Typical application scenarios include:

- Changing the service provider: An enterprise currently using a third-party Elasticsearch service wishes to switch to Huawei Cloud for reasons such as cost, performance, or other strategic considerations.
- Cluster merge: Data scattered across different third-party Elasticsearch clusters can be migrated to Huawei Cloud Elasticsearch clusters for centralized management, enabling more efficient data analysis and querying.
- Cross-version migration: Data is migrated from a third-party Elasticsearch cluster of an earlier version to a Huawei Cloud Elasticsearch cluster of a later version.
- Unified technology stack: For a simplified, unified technology stack, companies that are already using some services on Huawei Cloud may choose to migrate their in-house built Elasticsearch clusters to Huawei Cloud.

Solution Architecture

Figure 1-9 Migration procedure

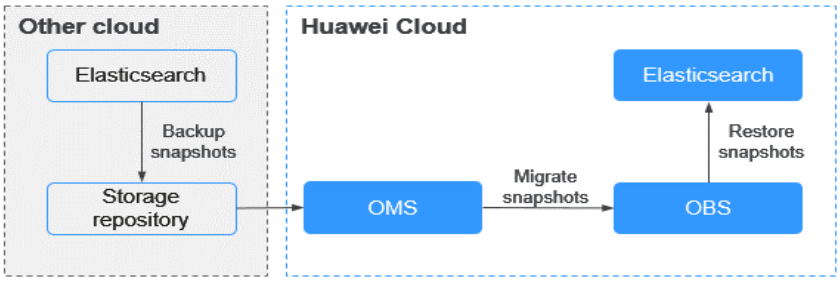


Figure 1-9 shows the process of migrating a third-party Elasticsearch cluster (source cluster) to Huawei Cloud Elasticsearch cluster (destination cluster).

1. Back up third-party Elasticsearch data to a third-party shared repository.
2. Use OMS to migrate data from the shared repository to Huawei Cloud OBS.
3. Use Huawei Cloud OBS to restore data to a Huawei Cloud Elasticsearch cluster.

Advantages

- Easy to use and manage: The snapshot and restoration APIs provided by Elasticsearch are easy to use and manage, and can be used to automate some of the tasks.
- Suitable for large-scale data migration: Snapshots can be used to migrate GB, TB, or even PB of data.
- Controllable restoration process: During data restoration, you can restore specific indexes or all indexes and specify the cluster status to be restored.

Impact on Performance

This migration method works by copying data directly from the storage layer. It does not rely on any external Elasticsearch APIs. Hence it significantly reduces any impact on the performance of the source cluster. For latency-insensitive workloads, the impact is negligible.

Constraints

- The version of the destination cluster must not be earlier than that of the source cluster. For details, see **Snapshot version compatibility**.
- This method does not support incremental data synchronization. You need to pause data update before starting to back up data.
- Snapshots can be migrated only when public access is enabled for third-party storage repositories.

Prerequisites

- The source and destination Elasticsearch clusters are available.
- The OBS bucket (esbak) for storing snapshots has been created. The OBS bucket and the destination Elasticsearch cluster in CSS are in the same **Region**, and the **Storage Class** is **Standard**.

Procedure

Step 1 Log in to the third-party cloud that hosts the Elasticsearch cluster you want to migrate and create a shared storage repository that supports the S3 protocol.

For example, log in to Alibaba Cloud, access the OSS service, and create a directory named **patent-esbak**. Or log in to Tencent Cloud, access the COS service, and create a directory named **patent-esbak**.

Step 2 Create a snapshot backup repository in the 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 OSS repository.

```
PUT _snapshot/my_backup
  {
     # Repository type.
  "type": "oss",
     "settings": {
     # # Private network domain name of the repository in step 1.
     "endpoint": "http://oss-xxx.example.com",
     # User ID and password of the repository. Hard-coded or plaintext access keys (AK/SK) are risky. For
```



```
security purposes, encrypt your access keys and store them in the configuration file or environment variables. In this example, access keys are stored in the environment variables for identity authentication. Before running the code in this example, configure the AK and SK in environment variables.
      "access_key_id": "ak",
      "secret_access_key": "sk",
      # 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 in the 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 in the third-party Elasticsearch 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 Use the Object Storage Migration Service (OMS) to migrate the snapshot data from the repository to the OBS bucket **esbak**.

OMS supports data migration from multiple cloud vendors to OBS. For details, see .

NOTE

When creating a migration task on OMS, set **Object Metadata** to **Migrate**. Otherwise, data migration may be abnormal.

Step 6 Log in to the Kibana console of a CSS Elasticsearch cluster.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
4. In the left navigation pane, choose **Dev Tools**.

   The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.

Step 7 Create a repository in the CSS Elasticsearch cluster and associate it with OBS. This repository will be used for restoring the snapshots of the third-party Elasticsearch cluster.

For example, create a repository named **my_backup_all** in the CSS Elasticsearch cluster and associate it with the OBS bucket **esbak**.

```
PUT _snapshot/my_backup_all/
{
    "type" : "obs",
    "settings" : {
        # Private network domain name of OBS
        "endpoint" : "obs.xxx.example.com",
        "region" : "xxx",
        # Username and password for accessing OBS. Hard-coded or plaintext access keys (AK/SK) are risky. For security purposes, encrypt your access keys and store them in the configuration file or environment variables. In this example, access keys are stored in the environment variables for identity authentication. Before running the code in this example, configure the AK and SK in environment variables.
        "access_key": "ak",
        "secret_key": "sk",
        # 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 8 Restore snapshot data to the CSS Elasticsearch cluster.

1. Check information about all snapshots.
   ```
   GET _snapshot
   ```
2. Restore data using snapshots.
   - 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"
     }
     ```
3. 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
```

Step 9 (Optional) After the migration, promptly delete the OBS bucket used for storing snapshots if these snapshots are no longer needed. This is to prevent ongoing storage costs.

----End

1.6 Migrating Data Between Huawei Cloud Elasticsearch Clusters Using the Read/Write Splitting Plugin

This topic describes how to migrate data between Huawei Cloud Elasticsearch clusters using the read/write splitting plugin.

Scenarios

By default, the read/write splitting plugin of CSS is installed in Elasticsearch 7.6.2 and Elasticsearch 7.10.2 clusters. You can configure read/write splitting to enable near-real-time synchronization of index data between Elasticsearch clusters.

The read/write splitting plugin can be used for data migration only if both the source and destination clusters were created in CSS. Typical application scenarios include:

- Cross-region or cross-account migration: Migrate the data of an Elasticsearch cluster in another region or under another account to the current cluster.
- Cluster merge: Merge the index data of two Elasticsearch clusters.

Solution Architecture

Figure 1-10 Migration procedure

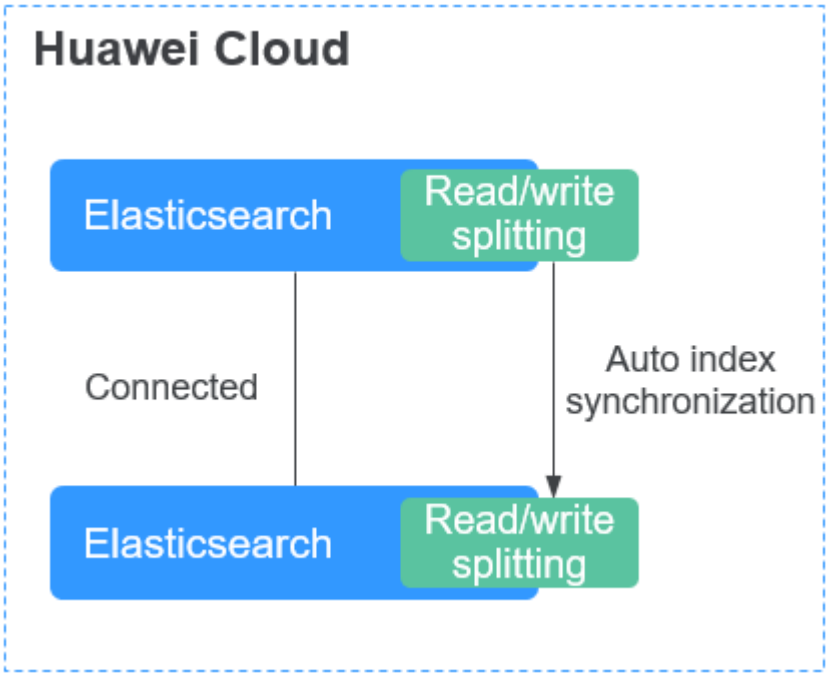


Figure 1-10 illustrates the general procedure for migrating data from one Huawei Cloud Elasticsearch cluster to another using the read/write splitting plugin of CSS.

1. Use the read/write splitting plugin to set up a connection between the source (primary) and destination (secondary) clusters.
2. Configure automatic index synchronization in the destination cluster to have data automatically synchronized from the source to the destination cluster. The synchronization interval is 30 seconds by default and can be changed.
3. Check the status of auto synchronization to see if data migration has completed.

For more information about the read/write splitting feature of CSS, see .

Advantages

- High data consistency: The read/write splitting feature enables data replication between a pair of primary/secondary clusters and ensures data synchronization between different shards.
- Fast migration: The speed of data synchronization depends on the available bandwidth, not the source or destination cluster.
- Configurable synchronization interval: The default data synchronization interval is 30 seconds. You can change it to reduce the delay of data synchronization.

Impact on Performance

When you use the read/write splitting plugin to migrate data between clusters, you are essentially just synchronizing data by copying files at the bottom layer.

This method does not rely on any external Elasticsearch APIs. Hence any impact on the performance of the source cluster is minimized.

Constraints

The versions of the source and destination clusters must both be 7.6.2 or 7.10.2.

Prerequisites

- The source and destination Elasticsearch clusters are available and both have the read/write splitting plugin installed. (By default, the read/write splitting plugin is installed in Elasticsearch 7.6.2 and 7.10.2.)
- The network between the clusters is connected.

  If the source and destination clusters are in different VPCs, establish a VPC peering connection between them. For details, see .

Procedure

Step 1 Obtain information about the Elasticsearch clusters to configure the data migration task.

Table 1-9 Required Elasticsearch cluster information

Cluster		Required Information	How to Obtain
Source cluster (primary cluster)	Huawei Cloud Elasticsearch cluster	• Access address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	• For how to obtain a cluster's access address, see the steps below. • Contact the service administrator to obtain the username and password.
	In-house built Elasticsearch cluster	• Public network address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	Contact the service administrator to obtain the information.
	Third-party Elasticsearch cluster	• Access address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	Contact the service administrator to obtain the information.

Cluster		Required Information	How to Obtain
Destination cluster (secondary cluster)	Huawei Cloud Elasticsearch cluster	• Access address of the destination cluster • Username and password for accessing the destination cluster (only for security-mode clusters)	• For how to obtain a cluster's access address, see the steps below. • Contact the service administrator to obtain the username and password.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, obtain the target cluster's private IP address from the **Private IP 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 this single node are displayed, for example, **10.62.179.32:9200**. If the cluster has multiple nodes and all of them are data nodes, the IP addresses and port numbers of all these nodes are displayed; if some of them are client nodes, only the IP addresses and port numbers of these client nodes are displayed; for example, **10.62.179.32:9200,10.62.179.33:9200**.

Step 2 Log in to the Kibana console of the destination Elasticsearch cluster.

1. In the Elasticsearch cluster list, find the target cluster, and click **Access Kibana** in the **Operation** column to log in to the Kibana console.
2. In the navigation pane on the left, choose **Dev Tools**.

Step 3 Use the read/write splitting plugin to set up a connection between the source and destination clusters.

Run the following command to configure information about the source cluster in the destination cluster:

```
PUT /_cluster/settings
{
  "persistent" : {
    "cluster" : {
      "remote.rest" : {
        "leader1" : {
          "seeds" : [
            "http://10.0.0.1:9200",
            "http://10.0.0.2:9200",
            "http://10.0.0.3:9200"
          ] ,
            "username": "elastic",
            "password": "*****"
        }
      }
    }
  }
}
```


Table 1-10 Request body parameters

Parameter	Description
leader1	Name of the configuration task, which is user-defined and will be used for configuring read/write splitting later.
seeds	Access address of the source cluster. When HTTPS is enabled for the cluster, the URI schema must use HTTPS.
username	Username of the source cluster (primary cluster). This parameter is required only when security mode is enabled for the primary cluster.
password	Password of the source cluster (primary cluster). This parameter is required only when security mode is enabled for the primary cluster.

If the value of **acknowledged** is **true** in the command output, the configuration is successful.

Step 4 Configure automatic index synchronization in the destination cluster to have data automatically synchronized from the source to the destination cluster.

Run the following command in the destination cluster to create a pattern-matching index synchronization policy, which synchronizes matched indexes from the source cluster to the destination cluster.

```
PUT auto_sync/pattern/pattern1
{
 "remote_cluster": "leader1",
 "remote_index_patterns": "log*",
 "local_index_pattern": "{{remote_index}}",
 "apply_exist_index": true
}
```

Table 1-11 Request body parameters

Parameter	Description
pattern1	Name of the pattern for index matching.
remote_cluster	Configuration task name, for example, **leader1** in the previous step.
remote_index_patterns	Pattern for matching indexes to be synchronized in the source cluster. The wildcard (*) is supported.
local_index_pattern	Index pattern in the destination cluster. The index template can be replaced. For example, if this parameter is set to **{{remote_index}}**, the index **log1** remains after synchronization.

Parameter	Description
apply_exist_index	Whether to synchronize existing indexes in the source cluster. The default value is **true**.

Step 5 Check the status of automatic synchronization in the destination cluster. The default synchronization interval is 30 seconds.

Run the following command to obtain the synchronization status of the matched indexes:

```
GET auto_sync/stats
```

If the value of **failed_count** is **0** in the command output, the synchronization is complete.

Step 6 After the synchronization is complete, check the data consistency between the source and destination Elasticsearch clusters.

For example, access the Kibana console of the source and destination clusters separately, go to **Dev Tools**, and run the **GET _cat/indices** command to check whether their indexes are consistent.

----End

1.7 Migrating Data Between Elasticsearch Clusters Using the Reindex API

You can use the reindex API to migrate data between Elasticsearch clusters.

Scenario

As an open-source search engine, Elasticsearch provides the reindex API to support index migration between clusters. This API is also provided in CSS to support data migration between Elasticsearch clusters. Below are some scenarios where you might use the reindex API for data migration.

- Cluster merging: The reindex API can be used to merge index data scattered across multiple Elasticsearch clusters into a single cluster for centralized data management and analysis.
- Cloud migration: Migrate an on-premises Elasticsearch service to the cloud to enjoy the benefits of cloud services, such as scalability, ease-of-maintenance, and cost-effectiveness.

The reindex API supports the following:

- Full migration: Migrate the full amount of index data between clusters. During the migration, all writes to the source cluster must be stopped, ensuring data consistency between the source and destination clusters.
- Incremental migration: For indexes that have a timestamp field, the reindex API can be used to execute an incremental migration based on this field. During the workload switchover phase, after the full migration is completed, you must stop all writes to the source cluster, and then use the reindex API to execute a quick incremental migration based on the latest update time. Then you can finally switch all services to the destination cluster.

- Reorganizing indexes: The reindex API can be used to restructure indexes while migrating them, including changing mappings, analyzers, and sharding.

Overview

Figure 1-11 Migration procedure

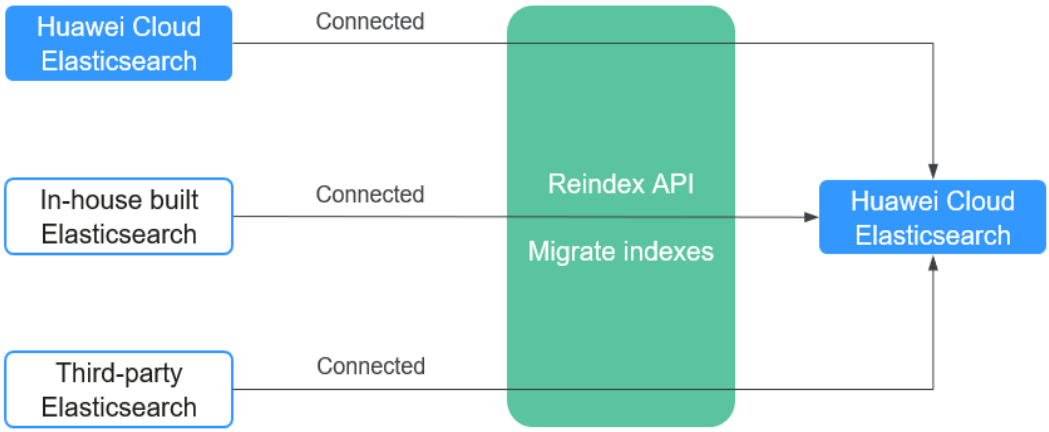


Figure 1-11 shows how to migrate data between Elasticsearch clusters using the reindex API.

1. Configure the reindex remote access whitelist in the destination cluster to connect the source and destination clusters.
2. Use the reindex API to migrate indexes from the source cluster to the destination cluster.

Advantages

- Easy to use: As a built-in function of Elasticsearch, the reindex API offers an easy way to migrate data without complex settings or additional tools.
- Flexible data processing: Indexes can be restructured or rebuilt during migration, such as changing mappings, analyzers, and sharding.
- Performance control: During the migration, you can tune the parameters of the scroll API to control the data migration speed for optimal cluster performance.

Impact on Performance

Using the reindex API for data migration between clusters relies on the scroll API. The scroll API can efficiently retrieve index data from the source cluster and synchronize the data to the destination cluster in batches. This process may impact the performance of the source cluster. The specific impact depends on how fast data is retrieved from the source cluster, and the data retrieval speed depends on the **size** and **slice** settings of the scroll API. For details, see the **Reindex API** document.

Reindex tasks are asynchronous in Elasticsearch clusters, so their impact on the performance of the source cluster is manageable when task concurrency is low. If the source cluster has a high resource usage, it is advisable to tune the size parameter of the scroll API to slow down the data retrieval speed or perform the

migration during off-peak hours, reducing impact on the performance of the source cluster.

Constraints

- During cluster migration, do not add, delete, or modify the index data of the source cluster. Otherwise, the data in the source cluster will be inconsistent with that in the destination cluster after the migration.
- The source and destination clusters must use the same version.

Prerequisites

- The source and destination Elasticsearch clusters are available.
- The network between the clusters is connected.
  - If the source and destination clusters are in different VPCs, establish a VPC peering connection between them. For details, see .
  - To migrate an in-house built Elasticsearch cluster to Huawei Cloud, you can configure public network access for this cluster.
  - To migrate a third-party Elasticsearch cluster to Huawei Cloud, you need to establish a VPN or Direct Connect connection between the third party's internal data center and Huawei Cloud.
- Ensure that **_source** has been enabled for indexes in the cluster.

  By default, **_source** is enabled. You can run the **GET {index}/_search** command to check whether it is enabled. If the returned index information contains **_source**, it is enabled.

Obtaining Information About the Source Elasticsearch Cluster

Before data migration, obtain necessary information about the source cluster for configuring a migration task.

Table 1-12 Required information about the source Elasticsearch cluster

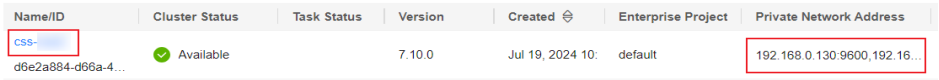
Cluster Type	Required Information	How to Obtain
Huawei Cloud Elasticsearch cluster	• Access address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters) • Index structure	• For details about how to obtain the cluster name and address, see **3**. • Contact the service administrator to obtain the username and password. • For details about how to query the index structure, see **6**.

Cluster Type	Required Information	How to Obtain
In-house built Elasticsearch cluster	• Public network address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters) • Index structure	Contact the service administrator to obtain the information.
Third-party Elasticsearch cluster	• Access address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters) • Index structure	Contact the service administrator to obtain the information.

The method of obtaining the cluster information varies depending on the source cluster. This section describes how to obtain information about a Huawei Cloud Elasticsearch cluster.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters** > **Elasticsearch**.
3. In the Elasticsearch cluster list, obtain the name and address of the target cluster.

   Figure 1-12 Obtaining cluster information

4. Click **Access Kibana** in the **Operation** column to log in to the Kibana console.
5. In the navigation pane on the left, choose **Dev Tools**.
6. Run the following command to query the index structure of the source cluster:

   ```
   GET {index_name}
   ```

   index_name indicates the name of the index to be migrated.

Configuring the Reindex Remote Access Whitelist

In the destination Elasticsearch cluster, configure the reindex whitelist.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, click the name of the target cluster. The cluster information page is displayed.
4. Choose **Cluster Settings** > **Parameter Settings**.

5. On the **Parameter Settings** page, click **Edit**, and then expand **Reindexing**.
   - Set **reindex.remote.whitelist**.
   - Value: Enter the address of the source cluster obtained in **Obtaining Information About the Source Elasticsearch Cluster**.

   If the source Elasticsearch cluster uses HTTPS, expand **Customize**, and add a custom parameter to ignore SSL authentication.
   - Parameter: reindex.ssl.verification_mode
   - Value: none
6. Click **Submit**. In the displayed confirmation dialog box, confirm the parameter settings, select **I understand that the modification will take effect after the cluster is restarted**, then click **OK**.
7. Return to the Elasticsearch cluster list, and locate the destination cluster. Choose **More** > **Restart** in the **Operation** column. In the displayed dialog box, select all nodes to restart the cluster and make the change take effect.

Migrating Indexes Using the Reindex API

1. Log in to the Kibana console of the destination cluster.
   a. Log in to the **CSS management console**.
   b. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
   c. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
   d. In the left navigation pane, choose **Dev Tools**.

      The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.
2. Go to the command execution page. Run the **put {index_name}** command to create an index structure that is identical to that in the source cluster:

   index_name indicates the index name after the migration. For the index structure of the source cluster, see **Obtaining Information About the Source Elasticsearch Cluster**.
3. Run the following command to migrate data using the reindex API:
   - **Full migration**: Migrate the full amount of index data in the source cluster to the destination cluster.

```
POST _reindex?wait_for_completion=false
{
  "source": {
     "remote": {
      "host": "http://xx.xx.xx.xx:9200",    //Address of the source cluster. If the source cluster uses HTTPS, use https://xx.xx.xx.xx:9200.
      "username": "xxx",    //Username for accessing the source cluster. It is needed for a security-mode cluster only.
      "password": "******"    //Password for accessing the source cluster. It is needed for a security-mode cluster only.
    },
    "index": "index_name",    //Index name in the source cluster
    "size": 3000
  },
  "dest": {
    "index": "index_name"    //Index name in the destination cluster
  }
}
```


- **Incremental migration**: Migrate new/changed index data from the source cluster to the destination cluster based on timestamps. This method can also be used to migrate an oversized index one chunk at a time.

```
POST _reindex?wait_for_completion=false
{
  "source": {
    "remote": {
      "host": "http://xx.xx.xx.xx:9200",    //Address of the source cluster. If the source cluster uses HTTPS, use https://xx.xx.xx.xx:9200.
      "username": "xxx",    //Username for accessing the source cluster. It is needed for a security-mode cluster only.
      "password": "******"    //Password for accessing the source cluster. It is needed for a security-mode cluster only.
    },
    "query": {
      "range" : {
        "timestamps" : {    //The time field
          "gte" : "xxx",    //Start time of the incremental data.
          "lte" : "xxx"    //End time of the incremental data.
        }
      }
    },
    "index": "index_name",    //Index name in the source cluster
    "size": 3000
  },
  "dest": {
    "index": "index_name"    //Index name in the destination cluster
  }
}
```

- **Index reorganization in the same cluster**: Use the reindex API to restructure indexes during migration.

```
POST _reindex?wait_for_completion=false
{
  "source": {
    "index": "index_name",    //Index name in the source cluster
    "size": 3000
  },
  "dest": {
    "index": "index_name"    //Index name in the destination cluster
  }
}
```

FAQ: What Do I Do If It Is Slow to Migrate an Oversized Index?

It may take a long time to migrate an oversize index. To speed up the migration, use the following methods:

- The reindex API relies on the scroll API to read data from the source cluster and write data into the destination cluster. You can set the size and slice parameters of the scroll API to increase migration concurrency and speed. For details, see **Reindex API**.
- Before the migration, set the number of replicas to 0 for the destination index. After the migration is completed, change the number of replicas to the original value.
- Use snapshots to migrate large amounts of data. See **Migrating Data Between Huawei Cloud Elasticsearch Clusters Using Backup and Restoration**, **Migrating Data from an On-premises Elasticsearch Cluster to Huawei Cloud Using the S3 Plugin**, and **Migrating Data from a Third-Party Elasticsearch Cluster to Huawei Cloud Using Backup and Restoration** for examples.

- If the index has a time field, use incremental migration to migrate the index one chunk at a time.

1.8 Migrating Data Between Elasticsearch Clusters Using ESM

Scenarios

The Elasticsearch Migration Tool (ESM) is an open-source tool designed for migrating data between Elasticsearch clusters, including between those of different versions. When using ESM to migrate data, you can adjust the migration speed by tuning the parameters of the scroll API to accommodate various network conditions and service requirements. Below are some scenarios where you might use ESM for data migration.

- Cross-version data migration: Use ESM to seamlessly migrate data when upgrading an Elasticsearch cluster to a new version, ensuring data integrity and availability during and after the upgrade.
- Cluster merging: Merge index data scattered across multiple Elasticsearch clusters into a single cluster for centralized data management and analysis.
- Cloud migration: Migrate an on-premises Elasticsearch service to the cloud to enjoy the benefits of cloud services, such as scalability, ease-of-maintenance, and cost-effectiveness.

Overview

Figure 1-13 Migration procedure

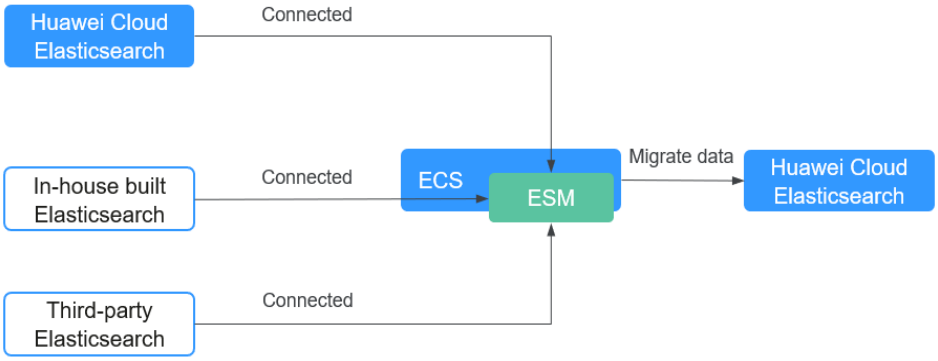


Figure 1-13 shows how to migrate data between Elasticsearch clusters using ESM.

1. Install ESM on a Linux VM.
2. Run ESM commands to migrate indexes from the source cluster to the destination cluster.

Advantages

- Cross-version data migration: You can use ESM to migrate data between Elasticsearch clusters of different versions, including from an earlier version to a later version.

- Easy to use: ESM is easy to use and is developed using the Go language. It quickly becomes available after being installed using an installation package.
- Performance control: During the migration, you can tune the parameters of the scroll API to control the data migration speed for optimal cluster performance.
- Flexible migration options: ESM supports both full migration and incremental migration, accommodating different needs.
- Free: As an open-source tool, the ESM code is hosted on GitHub and accessible to all users for free.

Impact on Performance

Using ESM for data migration between clusters relies on the scroll API. The scroll API can efficiently retrieve index data from the source cluster and synchronize the data to the destination cluster in batches. This process may impact the performance of the source cluster. The impact varies depending on how fast data is read from the source cluster. ESM allows you to adjust the data migration speed by configuring **-w** and **-b**. For details, see the **ESM documentation**.

ESM can quickly read data from the source cluster, potentially impacting its performance. Therefore, it is advisable to perform the data migration during off-peak hours. Additionally, you may need to monitor changes in the CPU and memory metrics of the source cluster. By tuning the migration speed and choosing an appropriate time window for the migration, you can keep the performance impact under control. If you have large amounts to data to migrate or if the source cluster has a high resource usage, you should perform the migration during off-peak hours, reducing impact on the performance of the source cluster.

Constraints

During cluster migration, do not add, delete, or modify the index data of the source cluster. Otherwise, the migration may fail, or data in the source cluster may be inconsistent with that in the destination cluster after the migration.

Prerequisites

- The source and destination Elasticsearch clusters are available.
- The network between the clusters is connected.
  - .
  - To migrate an in-house built Elasticsearch cluster to Huawei Cloud, you can configure public network access for this cluster.
  - To migrate a third-party Elasticsearch cluster to Huawei Cloud, you need to establish a VPN or Direct Connect connection between the third party's internal data center and Huawei Cloud.
- Ensure that **_source** has been enabled for indexes in the cluster.

  By default, **_source** is enabled. You can run the **GET {index}/_search** command to check whether it is enabled. If the returned index information contains **_source**, it is enabled.

Obtaining Elasticsearch Cluster Information

Before data migration, obtain necessary information about the source and destination clusters for configuring a migration task.

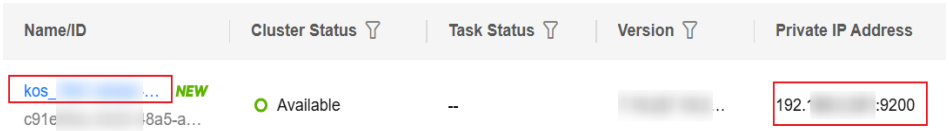
Table 1-13 Required Elasticsearch cluster information

Cluster Type		Required Information	How to Obtain
Source cluster	Huawei Cloud Elasticsearch cluster	• Access address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	• Contact the service administrator to obtain the username and password.
	In-house built Elasticsearch cluster	• Public network address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	Contact the service administrator to obtain the information.
	Third-party Elasticsearch cluster	• Access address of the source cluster • Username and password for accessing the source cluster (only for security-mode clusters)	Contact the service administrator to obtain the information.
Destination cluster	Huawei Cloud Elasticsearch cluster	• Access address of the destination cluster • Username and password for accessing the destination cluster (only for security-mode clusters)	• Contact the service administrator to obtain the username and password.

The method of obtaining the cluster information varies depending on the source cluster. This section describes how to obtain information about a Huawei Cloud Elasticsearch cluster.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters** > **Elasticsearch**.
3. In the cluster list, find the destination cluster and obtain the cluster name and address.

Figure 1-14 Obtaining cluster information



Preparing the VM Used for the Migration

Create an ECS where you install ECM for Elasticsearch cluster migration.

1. Buy a Linux ECS, select the CentOS 7 image, and set the VPC to that of the destination cluster.
2. Test the connectivity between the ECS and the source and destination clusters.

   Run the following command on the ECS to test connectivity. If the correct cluster information is returned, the connection is ready.

```
# For a cluster with the security mode disabled:
curl http://[ip]:9200

# For a security-mode cluster that uses HTTP:
curl -ik http://[ip]:9200 -u [username]:[password]

# For a security-mode cluster that uses HTTPS:
curl -ik https://[ip]:9200 -u [username]:[password]
```

Migrating Data Using ESM

1. Visit **the ESM download address**, and download the **migrator-linux-amd64** installation package.
2. Use SCP to upload the downloaded **migrator-linux-amd64** to a path on the Linux ECS.
3. Run the following command in the above path on the Linux ECS to migrate index structures and data from the source cluster to the destination cluster:

```
# Full migration
./migrator-linux-amd64 -s http://source:9200 -d http://dest:9200 -x index_name -m admin:password -n admin:password --copy_settings --copy_mappings -w 5 -b 10

# Incremental migration
./migrator-linux-amd6 -s http://source:9200 -d http://dest:9200 -x index-test -m admin:password -n admin:password -w 5 -b 10 -q "timestamp:[\"2022-01-17 03:41:20\" TO \"2022-01-22 03:41:20\"]"
```

   For commonly used parameters in the migration command, see **Table 1-14**. For even more information, see **ESM documents**.

Table 1-14 Commonly used parameters in ESM migration commands

Parameter	Example	Description
-s, --source=	http://source:9200	Address for accessing the source Elasticsearch cluster
-d, --dest=	http://dest:9200	Address for accessing the destination Elasticsearch cluster

Parameter	Example	Description
-x, --src_indexes=	index_name index1,index2	The names of the indexes in the source cluster waiting to be migrated. A regular expression can be used to specify indexes. Separate multiple indexes using a comma (,).
-y, --dest_index=	index_name_rename	Index name in a destination cluster. You may specify just a single index name. If left blank, the source index names will be used.
-m, --source_auth=	admin:password	Username and password for accessing the source Elasticsearch cluster (only for security-mode clusters)
-n, --dest_auth=	admin:password	Username and password for accessing the destination Elasticsearch cluster (only for security-mode clusters)
-w, --workers=	5	The number of concurrent threads for bulk data reading. This parameter controls how fast data will be read from the source cluster. Default value: **1**
-b, --bulk_size=	10	The bulk size for data reading. This parameter also controls how fast data will be read from the source cluster. Default value: 5 MB
--sliced_scroll_size	4	Size of sliced scroll. This parameter also controls how fast data will be read from the source cluster. Default value: **1**
--copy_settings	-	Whether to migrate index settings in the source cluster
--copy_mappings	-	Whether to migrate index mappings in the source cluster
--buffer_count=	-	Number of files to be cached in the memory of the VM that hosts ESM. Default value: 100,000

4. After the data migration is complete, run the following commands to obtain the number of documents in the source and destination clusters, respectively, and compare the results to verify data consistency.


```
# For a cluster with the security mode disabled:
curl http://[ip]:9200/[index_name]/_count

# For a security-mode cluster that uses HTTP:
curl -ik -u [username]:[password] http://[ip]:9200/[index_name]/_count

# For a security-mode cluster that uses HTTPS:
curl -ik -u [username]:[password] https://[ip]:9200/[index_name]/_count
```

FAQ

- **How do I handle the error message "out of memory" displayed during migration?**

  The error message "out of memory", if displayed during the migration, indicates an Out Of Memory Exception on the ESM ECS. Handle it using one of the following methods:

  – Use a larger flavor for the ECS.

  – Reduce the value of **buffer_count** used in the ESM migration command to reduce the number of files that can be cached in the memory of the ECS.

- **Why is the total size of index data inconsistent in the source and destination clusters after the migration?**

  This is normal when ESM is used to migrate Elasticsearch clusters. In a typical Elasticsearch cluster, multiple shards are used to store data, and each shard has multiple segments. After data is migrated from the source cluster to the destination cluster using ESM, new segments and shards are generated in the destination cluster. Different numbers of segments and shards in the source and destination clusters lead to different data expansion rates, hence different data sizes. To check data consistency, compare the number of files, rather than the data size.

1.9 Migrating Kibana Saved Objects Between Elasticsearch Clusters

Scenarios

You may want to migrate Kibana's saved objects between Elasticsearch clusters in the following scenarios:

- **Data migration**: When migrating data from one Elasticsearch cluster to another, migrating Kibana's saved objects is a key step to ensure service continuity. Exporting Kibana's saved objects (such as visualizations and dashboards) from the source cluster and importing them to the destination cluster ensures consistency in the user interface and monitoring dashboards.
- **Environment replication**: When replicating an Elasticsearch environment between the development, testing, and production environments, migrating Kibana's saved objects ensures consistency across different environments, thus improving development and testing efficiency.
- **Service failure or data loss**: In the event of a service failure or data loss, migrating Kibana's saved objects to a backup cluster helps to quickly restore data monitoring and analytics capabilities.

- **Multi-cluster management**: In a multi-cluster environment, consolidating data and other objects from different Elasticsearch clusters into a single cluster enables cross-cluster data analysis and management.

These scenarios highlight the importance of migrating Kibana's saved objects between Elasticsearch clusters. It is not just about the movement of data; it's a crucial method to ensure service continuity, improve efficiency, and guarantee compliance.

Solution Architecture

Figure 1-15 Migrating Kibana saved objects between Elasticsearch clusters

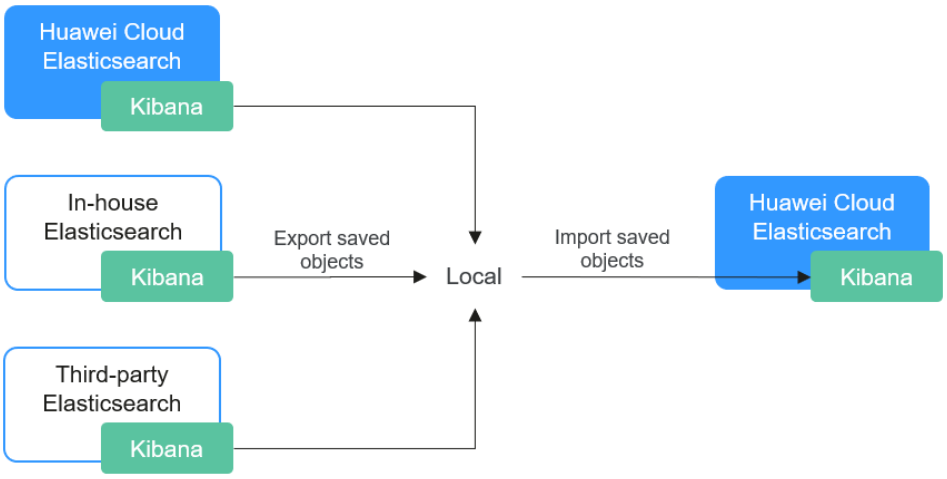


Figure 1-15 illustrates the process of migrating Kibana's saved objects between Elasticsearch clusters.

- The first step is exporting Kibana's saved objects from the source Elasticsearch cluster.
- The second step is importing these objects in the destination Elasticsearch cluster.

Advantages

- **Service continuity**: During a cluster upgrade or migration, migrating Kibana's saved objects ensures the continuity of cluster monitoring and analytics tasks.
- **Cross-environment consistency**: Replicating Kibana's saved objects across different environments (development, testing, and production) helps ensure environment consistency, improving development and testing efficiency.
- **Quick recovery**: In a fault recovery scenario, migrating Kibana's saved objects helps quickly restore cluster monitoring and analytics capabilities, mitigating the impact of any downtime.
- **Data consolidation**: In a multi-cluster environment, consolidating the data from different clusters by migrating Kibana's saved objects enables centralized data management and analysis.

Constraints

To avoid migration errors or failures, the versions of the source and destination Elasticsearch clusters must be close. If a version incompatibility error occurs during

migration, handle it by referring to **FAQ: How Do I Handle a Version Incompatibility Error Reported During the Migration of Kibana's Saved Objects?**.

Prerequisites

The source and destination Elasticsearch clusters are available.

Procedure

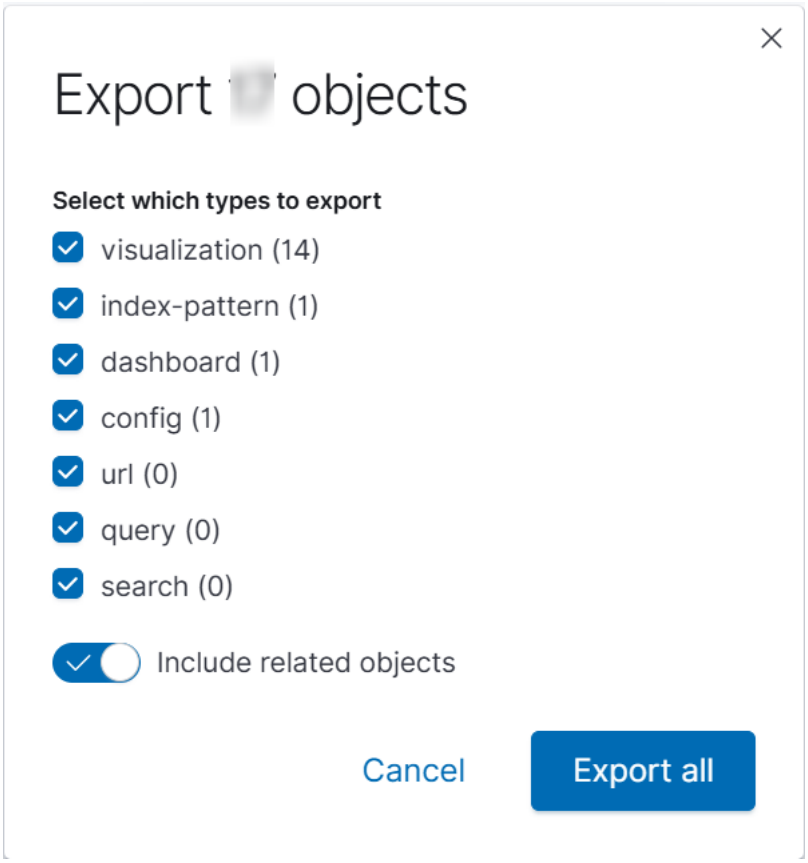
NOTE

The Kibana UI may vary depending on the Kibana version. The following uses version 7.10.2 as an example.

Step 1 Export Kibana's saved objects from the source Elasticsearch cluster to a local PC. In our example, the source cluster is an Elasticsearch cluster on Huawei Cloud.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the displayed cluster list, find the target cluster, and click **Access Kibana** in the **Operation** column to log in to the Kibana console.
4. In the navigation tree on the left, choose **Stack Management** > **Saved Objects**.
5. On the **Saved Objects** page, click **Export xx objects**. In the displayed dialog box, select the types of objects you want to export, and click **Export all**. An **export.ndjson** file is exported to the local PC.

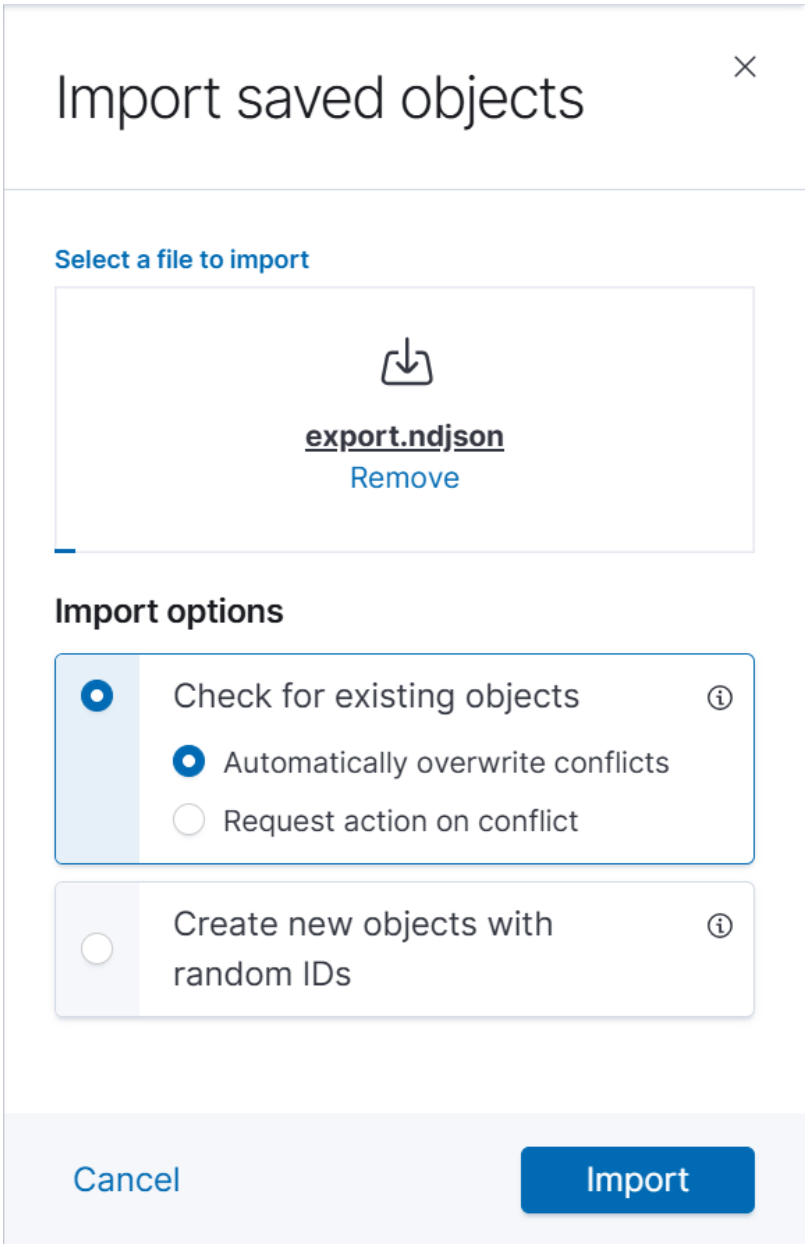
Figure 1-16 Exporting Kibana objects



Step 2 Import Kibana's saved objects exported in the previous step to the destination Elasticsearch cluster.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the displayed cluster list, find the target cluster, and click **Access Kibana** in the **Operation** column to log in to the Kibana console.
4. In the navigation tree on the left, choose **Stack Management** > **Saved Objects**.
5. On the **Saved Objects** page, click **Import**. In the displayed dialog box, select the source cluster's **export.ndjson** file saved on the local PC. Select **Automatically overwrite conflicts** for **Import options**, and click **Import**.

Figure 1-17 Importing Kibana objects



6. Click Done when the import is finished.

----End

FAQ: How Do I Handle a Version Incompatibility Error Reported During the Migration of Kibana's Saved Objects?

If the following error message is displayed when you import Kibana's saved objects, the source and destination clusters have incompatibility issues.

```
The file could not be processed due to error: "Unprocessable Entity: Document "7.1.1" has property "config" which belongs to a more recent version of Kibana [7.13.0]. The last known version is [7.9.0]"
```

In this case, you can modify the version information in the **export.ndjson** file on the local PC to keep the versions consistent. In this example, you need to change [7.13.0] to [7.9.0] in the code. Then, save the change and import the file again. If

the import still fails, you will have to manually rebuild the objects in the destination cluster.

1.10 Using Elasticsearch Pipelines for Incremental Data Migration

Elasticsearch ingest pipelines let you perform common transformations on your data before indexing. This capability makes it easier to identify incremental data updates, enabling reliable incremental data migration.

Scenario

If you encounter the following issues when using tools such as Logstash and ESM to incrementally migrate data between Elasticsearch clusters, you may consider using this solution:

- The indexes do not contain an incremental update time field that enables easy identification of incremental data updates.
- Due to complex service logic, incremental data updates cannot be identified reliably.
- Indexes use different incremental update time fields. A unified update method is needed.
- The incremental update time fields are not updated along with the data.

This solution uses Elasticsearch pipelines to automatically add an incremental update time field to indexes when data is written in, enabling more efficient and flexible incremental data migration.

Solution Architecture

Use Elasticsearch pipelines to automatically add an incremental update time field when data is written in. Based on this field, a data migration tool fetches and migrates incremental data.

1. Configure a pipeline in the source Elasticsearch cluster to automatically add an incremental update time field.
2. Associate an index with this pipeline, which automatically updates the incremental update time field when data is written into this index.
3. Use a migration tool to migrate incremental data.

Highlights

- Simplified operations: There is no need to analyze incremental update time fields for indexes. An ingest pipeline automatically generates a unified field, reducing configuration complexity.
- Flexibility: This solution supports different indexes and service scenarios. There is no need to modify the index structure.
- Compatibility: This solution can be used with different data migration tools, such as Logstash, ESM, and Reindex.

Constraints

During incremental data migration, do not delete index data in the source cluster. Otherwise, incremental data may be lost, and after the migration, data in the destination cluster may be inconsistent with that in the source cluster.

Prerequisites

- The source and destination Elasticsearch clusters are available, and the cluster version is later than 6.x. Otherwise, they may not support Elasticsearch ingest pipelines.
- The network between the clusters is connected.
  - If the source and destination clusters are in different VPCs, establish a VPC peering connection between them. For details, see .
  - To migrate an in-house built Elasticsearch cluster to Huawei Cloud, you need to enable public network access for this cluster.
  - To migrate a third-party Elasticsearch cluster to Huawei Cloud, you need to establish a VPN or Direct Connect connection between the third party's internal data center and Huawei Cloud.
- Ensure that **_source** has been enabled for indexes in the cluster.

  By default, **_source** is enabled. You can run the **GET {index}/_search** command to check whether it is enabled. If the returned index information contains **_source**, it is enabled.

Procedure

Step 1 Log in to the Kibana console of the source Elasticsearch cluster.

The login method varies depending on the source cluster. If Kibana is not installed, you can run cURL commands to configure the cluster. This section uses a CSS Elasticsearch cluster as an example to describe the procedure.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
4. In the left navigation pane, choose **Dev Tools**.

   The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.

Step 2 Add an incremental update time field for an index in the source cluster.

```
PUT /{index_name}/_mapping
{
  "properties": {
    "@migrate_update_time": {
      "type": "date"
    }
  }
}
```

Add an incremental update time field (for example, @migrate_update_time) and set the field type to **date**. {index_name} indicates the index name.

Step 3 Create a pipeline that automatically updates the incremental update time field.

```
PUT _ingest/pipeline/migrate_update_time
{
  "description": "Adds update_time timestamp to documents",
  "processors": [
    {
      "set": {
        "field": "_source.@migrate_update_time",
        "value": "{{_ingest.timestamp}}"
      }
    }
  ]
}
```

Pipeline processors will read the current machine time and write it to the incremental update time field (for example, @migrate_update_time) of the index.

Step 4 Associate the index and pipeline. The pipeline automatically updates the incremental update time field when data is written into this index.

```
PUT {index_name}/_settings
{
  "index.default_pipeline": "migrate_update_time"
}
```

Configure the default pipeline of the index. When index data is added or updated, the pipeline will always update the incremental update time field @migrate_update_time.

Step 5 Query the incremental update time field @migrate_update_time to identify incremental data.

```
GET /{index_name}/_search
{
  "query": {
    "range": {
      "@migrate_update_time": {
        "gte": "2025-01-01T00:00:00"
      }
    }
  }
}
```

Query documents whose incremental update time field (@migrate_update_time) is later than or equal to the specified time. The time format must match the field mapping type (**date**).

Step 6 Use a migration tool to perform incremental data migration.

For example, with ESM, run the following command:

```
./migrator-linux-amd6 -s http://source:9200 -d http://dest:9200 -x {index_name} -m admin:password -n admin:password -w 5 -b 10 -q "@migrate_update_time:[\"2025-04-08T00:00:00\" TO \"2030-01-01T00:00:00\"]"
```

For more about data migration commands and other migration tools, see **About Elasticsearch Cluster Migration Solutions**.

Step 7 Check data consistency.

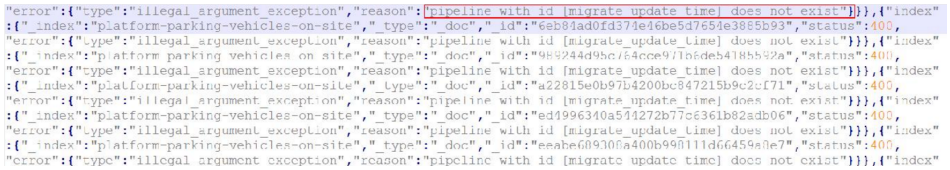
After the data migration is complete, run the **GET {index_name}/_count** command in Kibana of both the source and destination clusters to check index consistency.

----End

Common Issue: Pipeline (migrate_update_time) Does Not Exist

- **Symptom**: The error shown in **Figure 1-18** is reported during migration.

  Figure 1-18 Pipeline does not exist

- **Cause**: During index structure migration, pipelines created in the source Elasticsearch cluster are migrated along with indexes to the destination Elasticsearch cluster. However, if no such pipelines are created in the destination cluster, this error is reported.
- **Solution**: Cancel such pipelines in the destination Elasticsearch cluster.

  Log in to the Kibana console of the destination Elasticsearch cluster, go to **Dev Tools**, and run the following command:

  ```
  PUT {index_name}/_settings
  {
     "index.default_pipeline":  null
  }
  ```


2 Optimizing the Performance of Elasticsearch Clusters

2.1 Optimizing the Write Performance of Elasticsearch Clusters

Before using an Elasticsearch cluster in CSS, you are advised to optimize the cluster's write performance to improve efficiency.

Data Write Process

Figure 2-1 Data write process

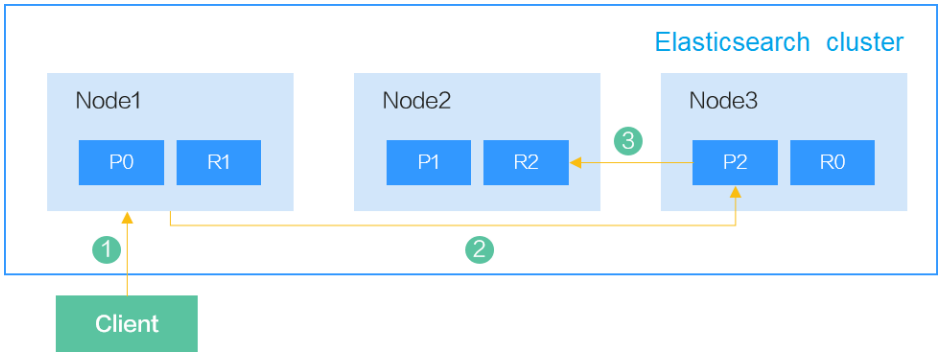


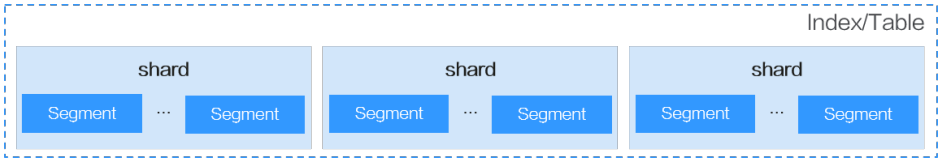
Figure 2-1 shows how a client writes data to an Elasticsearch cluster. In the preceding figure, **P** indicates the primary shard, and **R** indicates the replica shard. The primary and replica shards are randomly allocated in data nodes, but cannot be in the same node.

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.

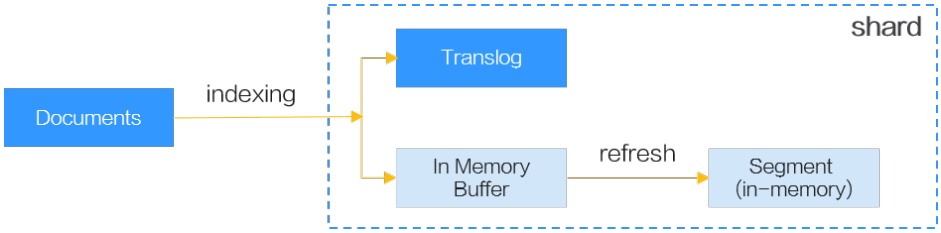
An index in Elasticsearch consists of one or more shards. Each shard contains multiple segments, and each segment is an inverted index.

Figure 2-2 Elasticsearch index composition



As shown in **Figure 2-3**, when a document is inserted into Elasticsearch, it is written to Buffer and Translog, and then periodically flushed to Segment. The refresh frequency is specified by the **refresh_interval** parameter. By default, data is refreshed every second. For more information about write performance, see **Near Real-Time Search**.

Figure 2-3 Process of inserting a document into Elasticsearch



Improving Write Performance

The following solutions can be used to improve Elasticsearch's write performance:

Table 2-1 Improving write performance

Solution	Description
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.
Use Bulk APIs.	The client writes data in batches. You are advised to write 1 MB to 10 MB data in each batch.
Randomly generate **_id**.	If **_id** 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 **_id**, you are advised to use a randomly generated **_id**.
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.

Solution	Description

Close replicas.	Data write and query are performed in off-peak hours. Close data copies during writing and open them afterwards.
	The command for disabling replicas in Elasticsearch 7.x is as follows:
	PUT *{index}*/_settings
	{
	"number_of_replicas": 0
	}

Adjust the index refresh frequency.	During batch data writing, you can set **refresh_interval** to a large value or **-1** (indicating no refresh), improving the write performance by reducing refresh.
	In Elasticsearch 7.x, run the following command to set the update time to 15s:
	PUT *{index}*/_settings
	{
	"refresh_interval": "*15s*"
	}

Change the number of write threads and the size of the write queue.	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. In Elasticsearch 7.x, you can modify the following parameters to optimize write performance: **thread_pool.write.size** and **thread_pool.write.queue_size**

Set a proper field type.	Specify the type of each field in the index, so that Elasticsearch will not assume by default that all fields are a combination of keywords and texts, which unnecessarily increase data volume. Keywords are used for keyword search, and texts used for full-text search.
	For the fields that do not require indexes, you are advised to set **index** to **false**.
	In Elasticsearch 7.x, run the following command to set **index** to **false** for **field1**:
	PUT *{index}*
	{
	"mappings": {
	"properties": {
	"field1":{
	"type": "text",
	"index": false
	}
	}
	}
	}

Solution	Description

Optimize the shard balancing policy.	By default, Elasticsearch uses the load balance policy based on 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 **routing.allocation.total_shards_per_node** 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.
	Run the following command to modify the setting of an existing index:
	PUT *{index}*/_settings
	{
	"index": {
	"routing.allocation.total_shards_per_node": 2
	}
	}

2.2 Optimizing the Query Performance of Elasticsearch Clusters

Before using an Elasticsearch cluster in CSS, you are advised to optimize the cluster's query performance to improve efficiency.

Data Query Process

Figure 2-4 Data query process

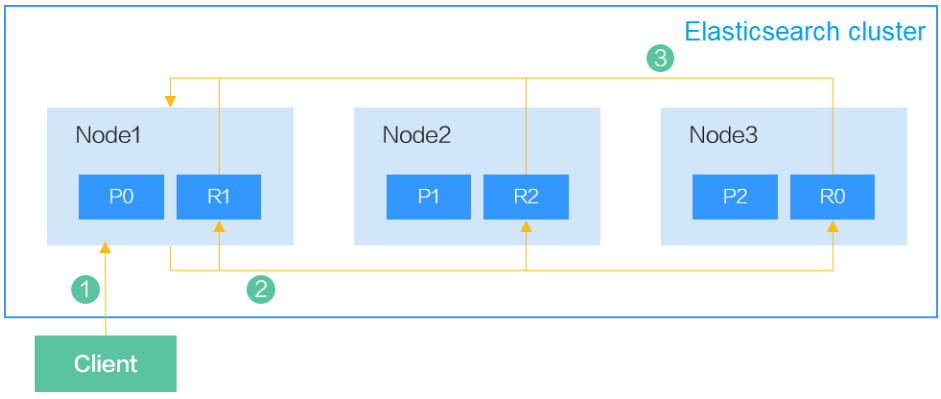


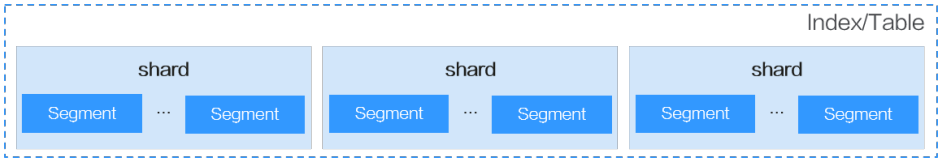
Figure 2-4 shows how a client sends a query request to an Elasticsearch cluster. In the preceding figure, **P** indicates the primary shard, and **R** indicates the replica shard. The primary and replica shards are randomly allocated in data nodes, but cannot be in the same node.

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.

3. 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 2-5 Composition of an Elasticsearch index



Improving Query Performance

The following solutions can be used to improve Elasticsearch's query performance:

Table 2-2 Improving query performance

Solution	Description

Use **_routing** to reduce the number of shards scanned during retrieval.	During data import, configure **routing** to route data to a specific shard instead of all the shards of the related index, improving the overall throughput of the cluster.
	In Elasticsearch 7.x, run the following commands:
	• Insert data based on a specified routing.
	`PUT /{index}/_doc/1?routing=user1`
	`{`
	`  "title": "This is a document"`
	`}`
	• Query data based on a specified routing.
	`GET /{index}/_doc/1?routing=user1`

Use index sorting to reduce the number of segments to be scanned.	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).
	For example, in Elasticsearch 7.x, run the following commands:
	`// Assume the date field needs to be frequently used for range query.`
	`PUT {index}`
	`{`
	`  "settings": {`
	`  "index": {`
	`  "sort.field": "date",`
	`  "sort.order": "desc"`
	`  }`
	`  },`
	`  "mappings": {`
	`  "properties": {`
	`  "date": {`
	`  "type": "date"`
	`  }`
	`  }`
	`  }`
	`}`

Solution	Description
Add query cache to improve cache hit.	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. You can add query cache by increasing the value of **indices.queries.cache.size**. Restart the cluster for the modification to take effect.

Perform forcemerge in advance to reduce the number of segments to be scanned.	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 **deleted**.
	In Elasticsearch 7.x, a configuration example is as follows:
	`// Assume the number of segments after index forcemerge is set to 10.`
	`POST /{index}/_forcemerge?max_num_segments=10`

3 Using Logstash to Synchronize Data to Elasticsearch

3.1 Synchronizing Data from RDS for MySQL to Elasticsearch Through Logstash

Scenarios

The logstash-input-jdbc plugin is installed by default in CSS's Logstash clusters. This plugin allows Logstash to fetch data from the relational database RDS for MySQL and push the data to Elasticsearch for deeper analysis. All you need to do is to configure the Logstash configuration file to define the JDBC input and Elasticsearch output. This solution can be used for the following purposes:

- **Real-time data update and synchronization**: Synchronizes an RDS for MySQL database with Elasticsearch in real time to leverage the latter's powerful search and analytics capabilities.
- **Log analytics and search**: Synchronizes log data from RDS for MySQL to Elasticsearch for quick search and analytics.
- **Application performance monitoring**: Synchronizes application performance data stored in RDS for MySQL to Elasticsearch through Logstash for real-time performance monitoring and performance analysis.
- **Data backup and recovery**: Uses Logstash to back up the data in RDS for MySQL to Elasticsearch, allowing for quick recovery in case of data loss or corruption.

Solution Architecture

Figure 3-1 Synchronizing RDS for MySQL with Elasticsearch

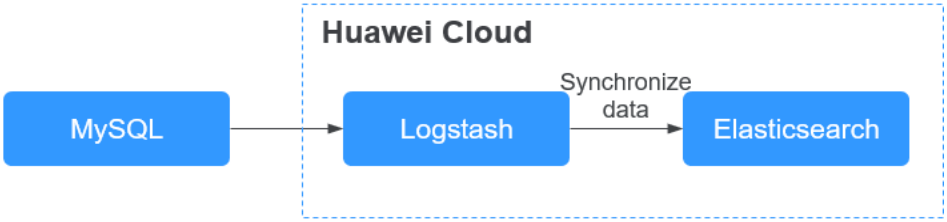


Figure 3-1 illustrates the process of synchronizing RDS for MySQL with Elasticsearch through Logstash.

The logstash-input-jdbc plugin enables both full data migrations and incremental data updates from RDS for MySQL to Elasticsearch.

Advantages

- **Flexible**: Logstash offers versatile data collection, transformation, optimization, and output capabilities, meeting diverse data synchronization needs.
- **Real-time**: Logstash supports near-real time data synchronization, accommodating the needs of most scenarios.
- **Easy-to-use**: The Logstash configuration file can be configured to meet various synchronization needs. The configuration is straightforward and requires no coding.

Constraints

- The same ID fields must be used in both Elasticsearch and RDS for MySQL.

  This is to establish a direct mapping between MySQL records and Elasticsearch indexes. For instance, when a record is updated in RDS for MySQL, the synchronization task overwrites the corresponding index in Elasticsearch with the same ID.
- New or updated records in RDS for MySQL must include a field indicating the insertion or update time.

  Logstash records the latest timestamp for each record during each polling cycle. During the next poll, it only processes records with timestamps later than the previously recorded ones.
- The RDS for MySQL database, Logstash cluster, and Elasticsearch cluster must be in the same time zone; otherwise, there may be time zone-related errors after the synchronization.

Prerequisites

- An RDS for MySQL database is available. This topic uses an RDS for MySQL instance on Huawei Cloud as an example. For details, see .
- A Logstash cluster used for data synchronization has been created. For details, see . This topic uses a Logstash 7.10.0 cluster as an example.
- An Elasticsearch cluster has been created. For details, see . This topic uses an Elasticsearch 7.10.2 cluster as an example.

The RDS for MySQL DB, Logstash cluster, and Elasticsearch cluster are in the same VPC.

If an in-house built or third-party MySQL database is used, check whether the database driver is MariaDB.

- Yes: Proceed to data synchronization configuration.
- No: Upload a SQL JDBC driver that is compatible with the RDS database version in use to the Logstash cluster. For details, see **FAQ: What Do I Do If the MySQL Driver Is Incompatible?**.

Procedure

Step 1 Test connectivity between the Logstash cluster and data sources.

1. Go to the **Configuration Center** page.
   a. Log in to the **CSS management console**.
   b. In the navigation pane on the left, choose **Clusters > Logstash**.
   c. In the cluster list, click the name of the target cluster. The cluster information page is displayed.
   d. Click the **Configuration Center** tab.
2. On the **Configuration Center** page, click **Test Connectivity**.
3. In the **Test Connectivity** dialog box, enter the IP address and port number of the data source and destination, and click **Test**.

   You can test a maximum of 10 IP addresses at a time. You can click **Add** to add more IP addresses and click **Test** at the bottom to test connectivity to multiple IP addresses at a time.

   If **Available** is displayed, the network is connected. If the network is disconnected, configure routes for the Logstash cluster to connect the clusters. For details, see .

Step 2 Create a Logstash configuration file for data synchronization.

1. On the Configuration Center page of the Logstash cluster, click **Create** in the upper right corner. On the **Create Configuration File** page, edit the configuration file.

   Table 3-1 Creating a Logstash configuration file

Parameter	Description
Name	User-defined configuration file name. It can contain only letters, digits, hyphens (-), and underscores (_), and must start with a letter. The minimum length is 4 characters.
Configuration File Content	Configure the configuration file by referring to the code example below. **NOTE** The size of each configuration file cannot exceed 100 KB.

Parameter	Description
Hidden Content	For items that you enter in this box, the corresponding strings will be replaced with *** in the configurations. This configuration is not required in our example here.

```
input {
  jdbc{
    # Configure the JDBC driver.
    jdbc_driver_library => "/rds/datastore/logstash/v7.10.0/package/logstash-7.10.0/extend/jars/
mariadb-java-client-2.7.0.jar"
    jdbc_driver_class => "org.mariadb.jdbc.Driver"
    jdbc_connection_string => "jdbc:mariadb://xxx.xxx.xxx.xxx:port/cms?
useUnicode=true&characterEncoding=utf8mb4&autoReconnect=true&allowMultiQueries=true"
    jdbc_user => "root"
    jdbc_password => "xx"
    # Retain the default values.
    jdbc_paging_enabled => "true"
    jdbc_page_size => "50000"
    # SQL statement for data migration.
    statement => "select a.user_code AS doctor_id,a.record_status from cluster "
    # Scheduled task, which repeats itself every 5 minutes. This interval can be customized.
    schedule => "*/5 * * * *"
  }
}
filter {

}
output {
  elasticsearch {
    hosts => ["xxx.xxx.xxx.xxx:port","xxx.xxx.xxx.xxx:port","xxx.xxx.xxx.xxx:port"]
    # Set the index name.
    index => "rds_doctor_index"
    user => "admin"
    password => "xx"
    # Document ID in the index. Keep it consistent with a primary key of the target table in RDS for
MySQL.
    document_id => "%{primary_id}"
    # Configure a certificate only if the destination Elasticsearch cluster uses HTTPS.
    ssl => true
    ssl_certificate_verification => false
    cacert => "/rds/datastore/logstash/v7.10.0/package/logstash-7.10.0/extend/certs"
    # Retain the default values.
    manage_template => false
    ilm_enabled => false
  }
}
```


Table 3-2 Configuration items

Configuration Item		Mandatory	Description
input	jdbc_driver_library	Yes	Path of the JDBC driver library. – If the database driver is MariaDB, set the value to **/rds/datastore/logstash/v7.10.0/package/logstash-7.10.0/extend/jars/mariadb-java-client-2.7.0.jar**. – If the database driver is a SQL JDBC driver compatible with the RDS database version in use, contact technical support to set the value. For how to set JDBC driver parameters, see **Jdbc input plugin**.
	jdbc_driver_class	Yes	Class path of the driver library. – If the database driver is MariaDB, set the value to **org.mariadb.jdbc.Driver**. – If the database driver is a SQL JDBC driver compatible with the RDS database version in use, set the value to **com.mysql.jdbc.Driver**.
	jdbc_connection_string	Yes	Address for accessing MySQL JDBC. – If the database driver is MariaDB, set the value to **jdbc:mariadb://xxx.xxx.xxx.xxx:port/cms?useUnicode=true&characterEncoding=utf8mb4&autoReconnect=true&allowMultiQueries=true**. – If the database driver is a SQL JDBC driver compatible with the RDS database version in use, set the value to **jdbc:mysql://xxx.xxx.xxx.xxx:port/cms**. Replace **xxx.xxx.xxx.xxx:port** with the database address plus port number.

Configuration Item		Mandatory	Description
	jdbc_user	Yes	Username for accessing MySQL JDBC.
	jdbc_password	Yes	Password for accessing MySQL JDBC.
	statement	Yes	SQL statement for data migration.
	schedule	Yes	Scheduled task. The data synchronization interval is customizable.
output	hosts	Yes	Address for accessing the Elasticsearch cluster.
	index	Yes	Index to which data is loaded.
	user	No	Username for accessing the Elasticsearch cluster, which is required only for security-mode clusters.
	password	No	Password for accessing the Elasticsearch cluster, which is required only for security-mode clusters.
	document_id	Yes	Document ID in the index. Keep it consistent with a primary key used in the target table in RDS for MySQL.
	ssl	No	Whether to enable HTTPS communication. If HTTPS access is enabled for the Elasticsearch cluster, set this parameter to **true**. Otherwise, do not set this parameter.
	ssl_certificate_verification	No	Whether to verify the server-end Elasticsearch certificate. Set this parameter only when **ssl** is set to **true**. – **true**: Verify the certificate. – **false**: Ignore the certificate.
	cacert	No	HTTPS access certificate. Retain the default for a CSS cluster.

2. Click **Next** to configure Logstash pipeline parameters. In this example, retain the default settings.
3. After the configuration is complete, click **Create**.

   On the **Configuration Center** page, you can check the created configuration file. If its status changes to **Available**, it has been successfully created.

Step 3 Start the Logstash configuration file.

1. In the configuration file list, select the configuration file you want to start, and click **Start** in the upper left corner.
2. In the **Start Logstash** dialog box, select **Keepalive** if necessary.
3. Click **OK** to start the configuration file and hence the Logstash migration task.

   You can check the started configuration file in the pipeline list.

Step 4 Verify that the database is now synchronized with the Elasticsearch cluster.

1. On the CSS management console, choose **Clusters** > **Elasticsearch**.
2. In the Elasticsearch cluster list, locate the destination cluster, and click **Access Kibana** in the **Operation** column to log in to the Kibana console.
3. In the navigation tree on the left, choose **Dev Tools**.
4. Run the following command to query index data:

```
GET rds_doctor_index/_count
{
  "query": {"match_all": {}}
}
```

   If the value of **count** is not 0 in the command output, data synchronization is successful.

----End

FAQ: What Do I Do If the MySQL Driver Is Incompatible?

If the Logstash pipeline does not function properly after the Logstash configuration file is started in the Logstash cluster, click **Run Log**. If the log contains error information similar to the following, the MySQL driver is incompatible.

```
[2024-05-21T11:31:00,196][ERROR][logstash.inputs.jdbc     ] Java::JavaSql::SQLSyntaxErrorException: (conn=-1409730930) You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '"T1" LIMIT 1' at line 1: SELECT count(*) AS "COUNT" FROM (select * from logstash_broker where updatetime+30000 > 0 order by updatetime) AS "T1" LIMIT 1
```

Solution:

1. Stop the Logstash configuration file.
2. Download a SQL JDBC driver compatible with the RDS database version in use, for example, **mysql-connector-java-8.0.11.tar.gz**. Extract **mysql-connector-java-8.0.11.jar** from it.

   Download address: **https://downloads.mysql.com/archives/c-j/**
3. Contact technical support to upload the .jar file to the Logstash cluster used for data synchronization.
4. Modify the Logstash configuration file.

   Change the values of **jdbc_driver_class** and **jdbc_connection_string**. Replace **xxx.xxx.xxx.xxx:port** with the database address plus port number.


```
jdbc_driver_class => "com.mysql.jdbc.Driver"
# Enter the MySQL JDBC URL.
jdbc_connection_string => "jdbc:mysql://xxx.xxx.xxx.xxx:port/cms"
```

5. Restart the configuration file.

3.2 Using Logstash to Sync Kafka Data to Elasticsearch

This topic describes how to use CSS Logstash to sync data from Kafka to CSS Elasticsearch. Logstash is an open-source data processing pipeline that ingests data from various sources, transforms it, and then sends it to your desired destination. A key component of the ELK Stack—Elasticsearch, Logstash, and Kibana—Logstash supports a multitude of data sources and ETL (Extract, Transform, and Load) methods, enabling efficient log management and services.

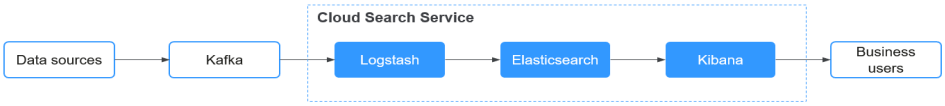
Scenarios

You may need to use CSS Logstash to sync data from Kafka to CSS Elasticsearch in the following scenarios:

- Real-time log management: real-time ingestion, processing, and storage of log data.
- Data preprocessing: cleaning, transformation, and enriching of raw log data.
- Unified log management: a unified log management platform for log storage, analysis, and visualization.

Solution Architecture

Figure 3-2 Solution architecture



1. Kafka as messaging middleware stores log data to be processed.
2. Logstash consumes data from Kafka topics, performs ETL, and writes the processed data to Elasticsearch.
3. Elasticsearch stores processed log data to support efficient query and analytics.
4. Kibana provides data visualization, facilitating data analysis and reporting.

Advantages

- Efficient data processing: Logstash supports high throughput and low latency in data processing.
- Flexible configuration: A variety of data sources and destinations are supported.
- Horizontal scalability: The solution can be used for large-scale data processing.
- Real-time: Data is consumed from Kafka topics and ingested into Elasticsearch in real time.

Constraints

The networks between Logstash and Kafka and between Logstash and Elasticsearch must be connected.

Prerequisites

- An Elasticsearch cluster has been created on the CSS console for storing and analyzing data. The cluster status is **Running**. You have obtained and recorded the cluster address. In the case of a security-mode cluster, you have obtained the administrator account and password.

  To create an Elasticsearch cluster, see .
- A Logstash cluster has been created on the CSS console for data processing.

  To create a Logstash cluster, see .
- Data has been ingested into the specified Kafka topics. You have obtained and recorded the IP address and port number of Kafka.

  If Distributed Message Service (DMS) for Kafka is used, see for how to create a Kafka topic.

Step 1: Create an Index Template in Elasticsearch

An index template is a way to tell Elasticsearch how to configure an index when it is created. It defines settings, mappings, and aliases that can be applied automatically to new indexes. The settings include the number of shards, number of replicas, and field types. Make sure they are consistent with how data is actually stored in Elasticsearch.

1. Log in to the Kibana console of the destination cluster.
   a. Log in to the **CSS management console**.
   b. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
   c. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
   d. In the left navigation pane, choose **Dev Tools**.

      The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.
2. Run the following command to create an index template.

   For example, create an index template that specifies three shards and no replicas per index and defines the **@timestamp** field in indexes.

```
PUT _template/filebeat
{
  "index_patterns": ["*topic*"],
  "settings": {
    # Define the number of shards.
    "number_of_shards": 3,
    # Define the number of replicas.
    "number_of_replicas": 0,
    "refresh_interval": "5s"
  },
  # Define fields.
  "mappings": {
        "properties": {
          "@timestamp": {
```



```
        "type": "date"
      }
    }
  }
}
```

Step 2: Test the Network Connectivity Between Logstash and Kafka

Test the network connectivity between Logstash and Kafka, making sure data can be correctly transmitted from Kafka to Logstash and then ingested into Elasticsearch.

1. Go to the **Configuration Center** page.
   a. Log in to the **CSS management console**.
   b. In the navigation pane on the left, choose **Clusters > Logstash**.
   c. In the cluster list, click the name of the target cluster. The cluster information page is displayed.
   d. Click the **Configuration Center** tab.
2. On the **Configuration Center** page, click **Test Connectivity**.
3. In the **Test Connectivity** dialog box, enter the IP address and port number of Kafka, and click **Test**.

   If **Available** is displayed, the network is connected.

   NOTE

   - If Logstash and Kafka are on the same internal network but are not connected, connect them by referring to .
   - If Kafka is on an external network and the network cannot be reached, connect them by referring to .

Step 3: Create a Logstash Configuration File

A Logstash configuration file defines the rules for data input, processing, and output, ensuring data can be correctly consumed from Kafka topics, processed, and ingested into Elasticsearch.

1. On the **Configuration Center** page of the Logstash cluster, click **Create** in the upper right corner to edit the configuration file.
   a. Applying a cluster template: Expand the system template list, select **kafka**, and click **Apply** in the **Operation** column.
   b. Setting the configuration file name: Set **Name**, for example, **kafka-es**.
   c. Editing the configuration file: In **Configuration File Content**, set the parameters by referring to comments. The following provides an example of the configuration file content. Modify the content as needed.
      - Configure a single Logstash pipeline with multiple Kafka inputs (one per topic) rather than using a single input to consume all topics.
      - For different Kafka topics, configure **group_id**, **client_id**, and **type** to ensure proper isolation between consumer groups.
      - Configure **consumer_threads** based on the number of partitions in topics. The number of Logstash nodes multiplied by

consumer_threads must be greater than or equal to the number of partitions.

- Logstash processes timestamps in UTC. To enable daily index rollover at 00:00 (UTC+8), modify the Ruby code to add 8 hours to the timestamps.

```
input {
  kafka  {
     # Read data in JSON format.
     codec => "json"
     # Topic name
     topics => ["topic-nginx"]
     # Kafka IP address
     bootstrap_servers => "192.168.0.1:9092,192.168.0.2:9092"
     # Kafka heartbeat settings. Keep the default.
     max_poll_interval_ms => "3000000"
     session_timeout_ms => "90000"
     heartbeat_interval_ms => "30000"
     # Number of consumer threads per Logstash node. Configure this based on the number of partitions in Kafka topics.
     consumer_threads => 5
     max_poll_records => "3000"
     auto_offset_reset => "latest"
     # Consumer group and type
     group_id => "topic-nginx-elk-hw"
     client_id => "topic-nginx-elk-hw"
     type => "topic-nginx"
   }
}
input {
  kafka  {
     # Read data in text format.
     codec => "plain"
     # Topic name
     topics => ["topic-gateway"]
     # Kafka IP address
     bootstrap_servers => "192.168.0.1:9092,192.168.0.2:9092"
     # Kafka heartbeat settings. Keep the default.
     max_poll_interval_ms => "3000000"
     session_timeout_ms => "90000"
     heartbeat_interval_ms => "30000"
     # Number of consumer threads per Logstash node. Configure this based on the number of partitions in Kafka topics.
     consumer_threads => 5
     max_poll_records => "3000"
     auto_offset_reset => "latest"
     # Consumer group and type
     group_id => "topic-gateway-elk-hw"
     client_id => "topic-gateway-elk-hw"
     type => "topic-gateway"
   }
}

# Split data.
filter {
   mutate {
     remove_field => ["@version","tags","source","input","prospector","beat"]
   }
# Align daily index rollover to Beijing time (UTC+8).
 ruby {
   code => " event.set('[@metadata][localdate]', (event.get('@timestamp').time.localtime + 8 * 60 * 60).strftime('%Y.%m.%d'))"
 }
}

# CSS cluster information
output {
 elasticsearch {
```



```
    hosts => ["http://192.168.0.4:9200"]
    index  => "%{type}-%{[@metadata][localdate]}"
    #user => "xxx"
    #password => "xxx"
  }
}
```

d. **Hidden Content**: Enter the list of strings you want to hide. Then press **Enter**. In the configurations returned, the specified strings will be replaced with asterisks (*). (You can enter a maximum of 20 strings, each with a maximum length of 512 bytes.)

2. Click **Next** to configure Logstash pipeline parameters.
   - **pipeline.workers**: Set the value to match the number of CPU cores on your Logstash node.
   - **pipeline.batch.size**: If there are only a small number of topics in a pipeline, set it to a value greater than 1000. If there are a large number of topics in a pipeline and the data size of each topic varies significantly, set the value to less than 1000, preventing the pipeline from being jammed by a single topic.
   - **pipeline.batch.delay**: Use the default value.
   - **queue type**: Select **memory**.
3. Click **Create**.

   The newly created configuration file is displayed on the **Configuration Center** page. If its status changes to **Available**, it is created successfully.

Step 4: Start the Logstash Configuration File

Start the Logstash configuration file to have Logstash connect to Kafka, pull data from Kafka topics, process the data, and write the processed data to Elasticsearch.

1. On the **Configuration Center** page of the Logstash cluster, select the newly created configuration file, and click **Start**.
2. In the **Start Logstash** dialog box, select **Keepalive** to ensure that the Logstash configuration file keeps running upon service restart, Logstash consumption errors, or any other incident, so that data processing will not be interrupted.

   When Keepalive is enabled, a daemon process is configured on each node. If the Logstash service becomes faulty, the daemon process will try to rectify the fault and restart the service, ensuring that the log management system runs efficiently and reliably
3. Click **OK** to start the configuration file.

   In the pipeline list, you can check the configuration file status and monitor data migration, ensuring proper data consumption and ingestion.

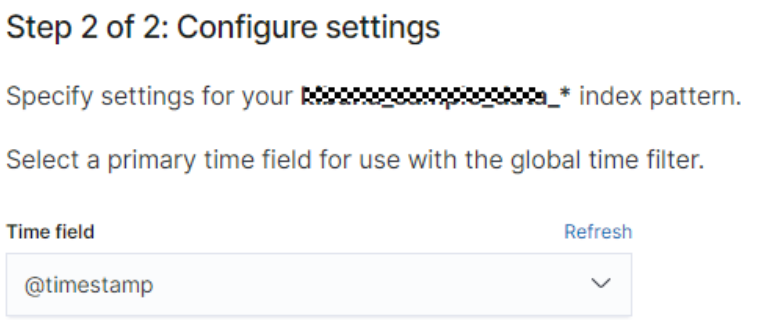
Step 5: Create an Index Pattern in Kibana

An index pattern defines the Elasticsearch indexes you want to visualize in Kibana. In our example, when creating an index pattern, we set the time field to @timestamp, so that Kibana can match data by time ranges.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.

3. In the displayed cluster list, find the target cluster, and click **Access Kibana** in the **Operation** column to log in to the Kibana console.
4. In the left navigation pane on the Kibana console, choose **Stack Management**.
5. Choose **Index patterns**, and click **Create index pattern**.
6. On the Create index pattern page, set **Index pattern name**, for example, to **topic-gateway***. Click **Next step**, and set **Time field** to **@timestamp** set for the index template created in **Step 1: Create an Index Template in Elasticsearch**.

   Figure 3-3 Index pattern configuration page

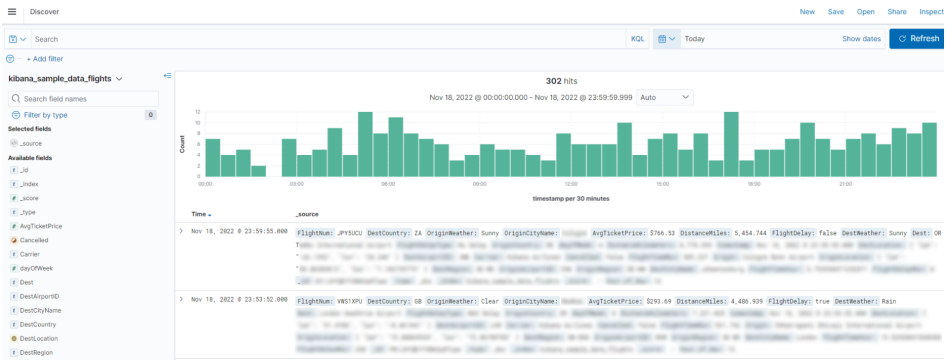
7. Click **Create index pattern**. The new index pattern is displayed in the index pattern list.

Step 6: Query and Analyze Data

Kibana allows you to visualize and analyze data stored in Elasticsearch, making it easy to create intuitive dashboards for insights and reporting.

In the left navigation pane, choose **Discover** to query and analyze data. **Figure 3-4** shows the data visualization effect.

Figure 3-4 Discover page



3.3 Using Logstash to Ingest Data from OBS into Elasticsearch

You can use CSS Logstash to ingest data stored in OBS to Elasticsearch for efficient data migration, search, and analytics.

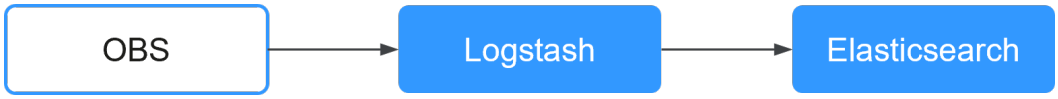
Scenarios

OBS is used to store massive amounts of data. When you need to perform quick search and analysis on such data, you can use CSS Logstash to ingest the data from OBS into an Elasticsearch cluster. Common application scenarios include:

- Data search and analytics: Synchronizes data (such as logs and service data) stored in OBS periodically or in real time to Elasticsearch for quick full-text search, aggregation, and visualization.
- OBS bucket log analysis: Synchronizes OBS bucket access logs to Elasticsearch for auditing, behavior analysis, and anomaly detection.

Solution Architecture

Figure 3-5 Solution architecture



Logstash can ingest data from a variety of data sources. For data stored in OBS, you can use Logstash's **logstash-input-s3** plugin to read objects from OBS and then use the **logstash-output-elasticsearch** plugin to synchronize the processed data to the destination Elasticsearch cluster.

1. Input: The logstash-input-s3 plugin connects to an OBS bucket, monitors specified files, and read their data.
2. Processing: Filters can be configured in the Logstash configuration file to clean, transform, and structure the data (for example, use the Grok filter to parse log formats).
3. Output: The logstash-output-elasticsearch plugin connects to the destination Elasticsearch cluster and indexes the data.

Advantages

- High compatibility: Supports multiple data formats, such as JSON, CSV, and OBS logs.
- High scalability: Filters can be configured to clean data, extract specific fields, and convert data formats.
- High flexibility: Supports different destinations, such as CSS Elasticsearch, In-house built Elasticsearch, and third-party Elasticsearch services.

Prerequisites

- The target data has been uploaded to the OBS bucket, which is in the same region as the destination Elasticsearch cluster. The following information about the OBS bucket has been obtained: bucket name, endpoint, and region.
- The destination Elasticsearch cluster has been created, and its access address (host) has been obtained. For a security-mode cluster, its username and password have been obtained.
- A CSS Logstash cluster has been created. It is in the same VPC as the destination Elasticsearch cluster and the two are connected.
- You have obtained the AK/SK of your account. For details, see **How Do I Obtain an Access Key (AK/SK)?**.

Procedure

Step 1 Access the CSS Logstash cluster.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Logstash**.

Step 2 Test network connectivity. If Logstash and Elasticsearch are deployed in the same VPC, skip this step.

1. In the Logstash cluster list, find the target cluster and click **Configuration Center** in the **Operation** column.
2. On the **Configuration Center** page, click **Test Connectivity**.
3. In the **Test Connectivity** dialog box, enter the IP address and port number of the destination Elasticsearch, and click **Test**.

   If **Available** is displayed, the network is connected.

   NOTE

   - If Logstash and Elasticsearch are on the same internal network but are not connected, connect them by referring to .
   - If Elasticsearch is on an external network and the network cannot be reached, connect them by referring to .

Step 3 Prepare the Logstash configuration file.

1. In the Logstash cluster list, find the target cluster and click **Configuration Center** in the **Operation** column.
2. On the Configuration Center page, click **Create** in the upper-right corner, edit the configuration file, and save the change.

   An example of the configuration file content (modify it based on service requirements):

```
input {
  s3 {
    access_key_id => "YOUR_AK"          # Access Key ID of the account
    secret_access_key => "YOUR_SK"      # Secret Access Key of the account
    bucket => "log_obs_access"          # OBS bucket name
    prefix => "test/access_log"         # (Optional) File name prefix that includes the bucket path. If no path is specified, it indicates all files matching the specified prefix in the bucket. In this example, all files whose name starts with access_log in the test path of the log_obs_access bucket will be fetched. Regular expression matching is not supported.
    endpoint => "https://OBS_Endpoint"  # OBS access address (the domain name here is the OBS VPC endpoint)
    region => "REGION"                  # Region where OBS is located
```



```
    watch_for_new_files => false          # (Optional) Whether to actively watch for newly created files. The default value is true.
    backup_to_bucket => "backup_log_obs_access"  # (Optional) Backup bucket, used to back up fetched files. Backup will not be performed if it is not set.
    backup_add_prefix => "backup"        # (Optional) Prefix of the path that stores backup files, which is used together with the prefix parameter. In this example, the actual backup path is /backup/test/access_log.
  }
}

filter {
  mutate {
    remove_field => ["@version"]         # (Optional) Remove redundant fields
  }
}

output {
  elasticsearch {
    hosts => ["192.168.0.xxx:9200"]        # Elasticsearch cluster address
    index => "delivery_events_log_alias"  # Index name
    manage_template => false              # Do not manage index templates
    ilm_enabled => false                 # Disable ILM policies (not supported in CSS)

    user => "USERNAME"                # Elasticsearch username (only for a security-mode cluster)
    password => "YOUR_PASSWORD"       # Elasticsearch password (only for a security-mode cluster)

    cacert => "/rds/datastore/logstash/v7.10.0/package/logstash-7.10.0/extend/certs"  # Elasticsearch cluster certificate (only for a cluster with HTTPS enabled)
    ssl => true                       # Whether to enable SSL (only for a cluster with HTTPS enabled)
    ssl_certificate_verification => false # Whether to perform certificate authentication (only for a cluster with HTTPS enabled)
  }
}
```

NOTE

- The filter part can be configured to customize the pattern for segmenting and transforming the data of a target file. For example, when analyzing OBS bucket access logs, you can use Grok for pattern matching. For an example, see **FAQ: How Do I Check OBS Bucket Access Logs?**. In addition to Grok, other processors, such as Dissect, KV, mutate, and split, are also supported. For more details, see **Filter plugins**.
- When using Logstash to ingest OBS data multiple times, you can further configure parameters like **watch_for_new_files**, **delete**, and **backup_to_bucket** to avoid duplicate data. For an example, see **FAQ: What Do I Do If Duplicate Data Is Read from OBS?**.

Step 4 Start a Logstash pipeline task.

1. On the **Configuration Center** page, select the newly created configuration file, and click **Start** in the upper-left corner.
2. In the **Start Logstash** dialog box, select **Keepalive** to ensure that the Logstash configuration file keeps running upon service restart or any incidents, so that data processing will not be interrupted.

   When Keepalive is enabled, a daemon process is configured on each node. If the Logstash service becomes faulty, the daemon process will try to rectify the fault and restart the service, ensuring that the Logstash pipelines run efficiently and reliably.
3. Click **OK** to start the configuration file.

   In the pipeline list, you can check the configuration file status and monitor data migration.

Step 5 Verify data synchronization.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the displayed cluster list, find the target cluster, and click **Access Kibana** in the **Operation** column to log in to the Kibana console.
4. In the navigation pane on the left, choose **Dev Tools**.
5. Run the following index query command. Check whether the expected number of records is returned for the target index.

```
GET delivery_events_log_alias/_count
{
  "query": {"match_all": {}}
}
```

   If the value of **count** in the result is not 0, data has been synchronized.

----End

FAQ: How Do I Check OBS Bucket Access Logs?

If logging is enabled for an OBS bucket, Logstash can read the OBS bucket's access logs and write the log data to Elasticsearch, where you can then analyze operations performed on the OBS bucket.

A sample of the OBS bucket log:

```
15e02840b2784ffb9dcac293afc01a75 genean-hot-test [02/Feb/2024:07:51:17 +0000] 58.250.177.72 15e02840b2784ffb9dcac293afc01a75 0000018D68CCEF1AD3296F98BB7B4255 REST.GET.OBJECT 00002c9d-172a-495d-b0a3-4aca8c1f86cc "GET /genean-hot-test/00002c9d-172a-495d-b0a3-4aca8c1f86cc?AWSAccessKeyId=H0N1CQY4D1ZBSHABB3NF&Expires=1706860576&response-content-disposition=attachment&response-content-type=application/octet-stream&x-amz-security-token=*****&Signature=DeC2lkZVFB4CjDjKe147ZOl8sKY%3D HTTP/1.1" 200 - 82509 82509 84 84 "https://console.example.com/" "Mozilla/5.0 (xx.xx.xx; xx; xx) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36 Edg/120.0.0.0" - - STANDARD - "-" efe6d963779e4475bb81c5cb1f5f5368
```

For more information, see **Using Logging to Record OBS Logs**.

You can add filters to the Logstash configuration file to customize OBS logs. See the following for an example. In this example, Grok is used for data matching.

```
filter {
 grok {
   match => {
     "message" => '(?<BucketOwner>[^ ]+) (?<Bucket>[^ ]+) \[%{HTTPDATE:Time}\] (?<RemoteIP>[^ ]+) (?<Requester>[^ ]+) (?<RequestID>[^ ]+) (?<Operation>[^ ]+) (?<Key>[^ ]+) \"(?<RequestURI>[^"]+)\" (?<HTTPStatus>[^ ]+) (?<ErrorCode>[^ ]+) (?<BytesSent>[^ ]+) (?<ObjectSize>[^ ]+) (?<TotalTime>[^ ]+) (?<Turn-AroundTime>[^ ]+) \"(?<Referer>[^"]+)\" \"(?<User-Agent>[^"]+)\" (?<VersionID>[^ ]+) (?<STSLogUrn>[^ ]+) (?<StorageClass>[^ ]+) (?<TargetStorageClass>[^ ]+) \"(?<DentryName>[^ ]+)\" (?<IAMUserID>[^ ]+)'
   }
   timeout_millis => 3000
   timeout_scope => "event"
 }
  mutate {
   remove_field => ["@version","message"]
 }
}
```

Once OBS bucket logs are ingested into Elasticsearch, you can search and analyze the data in Elasticsearch.

FAQ: What Do I Do If Duplicate Data Is Read from OBS?

The logstash-input-s3 plugin uses the **watch_for_new_files** parameter to determine whether to actively watch for file changes.

- When **watch_for_new_files** is set to **false**, the task reads the data in the OBS bucket path configured in the configuration file when it is run for the first time. After that, it records the last modification time of the files to the sincedb file and exits. If you restart the Logstash data consumption task, the files with the same last modification time will be consumed again, and this will lead to duplicate data. To avoid it, you are advised to avoid restarting the Logstash consumption task.
- When **watch_for_new_files** is set to **true**, the task periodically reads data from the OBS bucket, and records the file modification time in the sincedb file. Next time, the task will read new data based on the recorded file modification time. Files with an unchanged last modification time are read again, leading to duplicate data consumption. To avoid this, you can configure the **delete** and **backup_to_bucket** parameters to back up consumed files to a new OBS bucket and then delete them from the original bucket. The following is an example:

```
input {
  s3 {
    access_key_id => "YOUR_AK"
    secret_access_key => "YOUR_SK"
    bucket => "log_obs_access"
    prefix => "logstash"
    endpoint => "https://OBS_Endpoint"
    region => "REGION"
    delete => true                          # Whether to delete consumed files from the original bucket. The default value is false. If this parameter is set to true, the consumed files will be deleted.
    watch_for_new_files => true             # Whether to actively watch for file changes. The default value is true, in which case, new files added to the bucket are automatically consumed.
    backup_to_bucket => "backup_log_obs_access"  # Backup bucket name. Consumed files are backed up to this OBS bucket. By default, consumed files are not backed up.
    backup_add_prefix => "backup/"          # Prefix of the storage path of the backup files. It is used together with the prefix parameter. In this example, the actual backup path is /backup/logstash.
  }
}
```


4 Elasticsearch Vector Search

4.1 Testing the Performance of CSS's Elasticsearch Vector Search

Scenarios

CSS's vector search engine provides a fully managed, high-performance distributed vector database service. To facilitate performance/pressure testing for the vector search service and provide accurate references for product selection and resource configuration, this document describes the performance testing solutions for CSS's Elasticsearch vector search service based on open-source datasets and open-source pressure testing tools.

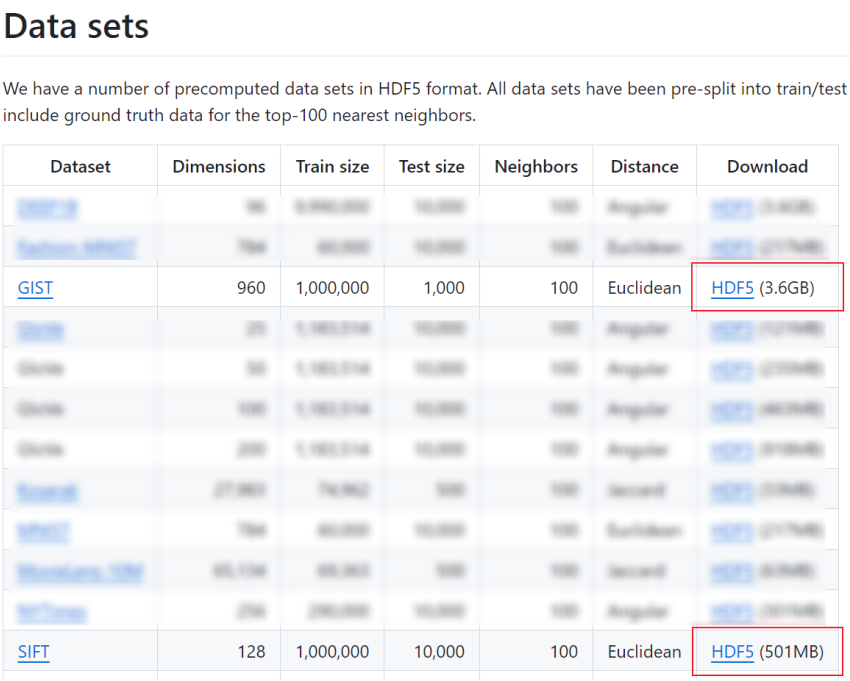
Preparations

- Create an Elasticsearch vector database. For details, see .

  Set the node quantity to 3, and node specifications to 4 vCPUs | 16GB under General computing. (The test data volume is relatively small, and the CPU specifications need to be kept close to those used in third-party performance benchmarking.) Select ultra-high I/O for node storage, and keep the security mode disabled.
- Obtain test datasets.
  - **sift-128-euclidean**: 128 dimensions, 1 million base records, measured using the Euclidean distance.
  - **cohere-768-cosine**: 768 dimensions, 1 million base records, measured using the cosine distance.
  - **gist-960-euclidean**: 960 dimensions, 1 million base records, measured using the Euclidean distance.

  You can download **sift-128-euclidean** and **gist-960-euclidean** at **https://github.com/erikbern/ann-benchmarks**. To use the **cohere-768-cosine** dataset, submit a service ticket.

Figure 4-1 Downloading sift-128-euclidean and gist-960-euclidean



Data sets

We have a number of precomputed data sets in HDF5 format. All data sets have been pre-split into train/test and include ground truth data for the top-100 nearest neighbors.

Dataset	Dimensions	Train size	Test size	Neighbors	Distance	Download
GIST	960	1,000,000	1,000	100	Euclidean	HDF5 (3.6GB)
SIFT	128	1,000,000	10,000	100	Euclidean	HDF5 (501MB)

- Prepare the testing tools.
  - Prepare the data writing and recall testing scripts. For details, see **Script base_test_example.py**.
  - Download the open-source pressure testing tool wrk at **https://github.com/wg/wrk/tree/master**.

Performance Testing Procedure

1. Create an ECS for installing the pressure testing tool and executing test scripts. For details, see .
   - This ECS and the Elasticsearch cluster created for the testing purpose must be within the same VPC and security group.
   - You may use another client server instead. No matter what server you use, ensure that it is in the same VPC as the Elasticsearch cluster.
2. Upload the test dataset to the ECS.
3. Upload the data writing and recall testing scripts to the ECS, and run the following command:

   ```
   pip install h5py
   pip install elasticsearch==7.6

   python3 base_test_example.py
   ```

   This command creates a vector index for testing, writes in the test data, and returns the average recall for the queries.
4. Install wrk on the ECS.
5. On the ECS, prepare the query request file used for the pressure testing to simulate real-world traffic. See **Script prepare_query.py** for an example.

   ```
   pip install h5py

   python3 prepare_query.py
   ```


6. Prepare the wrk pressure testing configuration script on the ECS. See **Script perf.lua** for an example. Modify the query request file name, cluster address, and index name in the script as needed.
7. Run the following command on the ECS to perform pressure testing on CSS's vector search service:

```
wrk -c60 -t60 -d10m -s perf.lua http://x.x.x.x:9200
```

   - **t** indicates the number of pressure testing threads.
   - **c** indicates the number of server connections.
   - **d** indicates the pressure testing duration. **10m** indicates 10 minutes.
   - **s** indicates the pressure testing configuration script for wrk.
   - **x.x.x.x** indicates the address of the Elasticsearch cluster.

   Obtain the test result from the command output, where **Requests/sec** indicates the query throughput in QPS.

   Figure 4-2 Test result example

```
root@ecs-          :vdb:~/workspaces/luvector# wrk -c60 -t60 -d5m -s perf.lua
Running 5m test @ http://              :9200
  60 threads and 60 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     4.16ms    2.70ms 125.82ms   89.86%
    Req/Sec   259.93     38.66     0.92k    68.65%
  4670330 requests in 5.00m, 4.92GB read
Requests/sec:  15562.60
Transfer/sec:     16.80MB
```

Performance Testing Solutions

NOTE

The following testing solutions list only the values of the compared parameters. You are advised to retain the default values for parameters that are not listed here.

- **GRAPH indexes**

  For a database of millions of records, GRAPH indexing is recommended.

  - **Testing solution 1**: Use datasets of varying numbers of data dimensions to test the maximum QPS supported by the vector database when the top-10 recall rate reaches 99%. Perform the test on each dataset using both default parameters and performance-tuned ones. By tuning the build parameters, you can optimize the graph index structure to enhance query performance while maintaining the same recall rate.

    Test result:

Table 4-1 GRAPH index test result 1

Dataset	Build Parameters		Query Parameters		Metrics	
	efc	shrink	ef	max_scan_num	QPS	Recall
sift-128-euclidean	200	1.0	84	10000	15562	0.99
	500	0.8	50	10000	17332	0.99
cohere-768-cosine	200	1.0	154	10000	3232	0.99
	500	0.95	106	10000	3821	0.99
gist-960-euclidean	200	1.0	800	19000	860	0.99
	500	0.9	400	15000	1236	0.99

Conclusion: On all these datasets, the vector search service can reach a recall rate of 99% or higher using default parameters. By further tuning the build and query parameters, the index building overhead increases slightly, but this also results in improved query performance.

- **Testing solution 2**: Use the same dataset to test the vector search service's query performance under different recall rates by tuning index parameters. This solution uses the Cohere dataset to test the maximum QPS of the cluster under different top-10 recall rates—99%, 98%, and 95%, respectively.

  Test result:

Table 4-2 GRAPH index test result 1

Dataset	Build Parameter	Query Parameter	Metrics	
	efc	ef	QPS	Recall
cohere-768-cosine	500	128	3687	0.99
	500	80	5320	0.98
	500	36	9028	0.95

Conclusion: With fixed index building parameters for the same cluster, tuning the ef parameter can achieve different query precisions. Slightly sacrificing the recall rate can significantly boost QPS.

- **GRAPH_PQ indexes**

  Graph-based indexing typically requires residual memory to ensure query performance. When dealing with a large number of vector dimensions or high data volumes, memory resources become a crucial factor affecting costs and performance. Specifically, high-dimensional vectors and large datasets demand significantly more memory, increasing storage costs and directly impacting the efficiency and response times of the indexing algorithm. In this scenario, the GRAPH_PQ indexing algorithm is recommended.

  Testing solution: Use the COHERE and GIST datasets with high data dimensions to test the cluster's maximum QPS under a top-10 recall rate of 95%. Compare the residual memory overhead with that of GRAPH indexes.

  Test result:

  Table 4-3 GRAPH_PQ index test result

Dataset	Build Parameters		Query Parameters		Metrics		Memory Overhead	
	efc	fragment_num	ef	topk	QPS	Recall	GRAPH_PQ	GRAPH
cohere-768-cosine	200	64	85	130	8723	0.95	332MB	3.3GB
gist-960-euclidean	200	120	200	360	4267	0.95	387MB	4.0GB

  Conclusion: The result shows that GRAPH_PQ indexing can achieve the same or similar precision and QPS as GRAPH indexing while cutting the memory overhead more than 10 times. By integrating graph indexing and quantization, the GRAPH_PQ indexing algorithm used by CSS's vector search service significantly reduces the memory overhead and increases per-node data capacity.

Table 4-4 describes the indexing parameters used above. For more information about the build parameters, see . For more information about the query parameters, see .

Table 4-4 Description of index parameters

Type	Parameter	Description
Build parameter	efc	Queue size of the neighboring node during HNSW build. The default value is **200**. A larger value indicates a higher precision and slower build speed.
	shrink	Cropping coefficient during HNSW build. The default value is **1.0f**.
	fragment_num	Number of fragments. The default value is **0**. The plugin automatically sets the number of fragments based on the vector length.
Query parameter	ef	Queue size of the neighboring node during the query. A larger value indicates a higher query precision and slower query speed. The default value is **200**.
	max_scan_num	Maximum number of scanned nodes. A larger value indicates a higher query precision and slower query speed. The default value is **10000**.
	topk	The number of top-k records returned for a query.

Script base_test_example.py

```
# -*- coding: UTF-8 -*-
import json
import time

import h5py
from elasticsearch import Elasticsearch
from elasticsearch import helpers

def get_client(hosts: list, user: str = None, password: str = None):
    if user and password:
        return Elasticsearch(hosts, http_auth=(user, password), verify_certs=False, ssl_show_warn=False)
    else:
        return Elasticsearch(hosts)

# For more information about the index parameters, see .
def create(es_client, index_name, shards, replicas, dim, algorithm="GRAPH",
           metric="euclidean", neighbors=64, efc=200, shrink=1.0):
    index_mapping = {
        "settings": {
            "index": {
                "vector": True
            },
            "number_of_shards": shards,
            "number_of_replicas": replicas,
        },
        "mappings": {
```



```
            "properties": {
                "id": {
                    "type": "integer"
                },
                "vec": {
                    "type": "vector",
                    "indexing": True,
                    "dimension": dim,
                    "algorithm": algorithm,
                    "metric": metric,
                    "neighbors": neighbors,
                    "efc": efc,
                    "shrink": shrink,
                }
            }
        }
    }
    es_client.indices.create(index=index_name, body=index_mapping)
    print(f"Create index success! Index name: {index_name}")

def write(es_client, index_name, vectors, bulk_size=1000):
    print("Start write! Index name: " + index_name)
    start = time.time()
    for i in range(0, len(vectors), bulk_size):
        actions = [{
            "_index": index_name,
            "id": i + j,
            "vec": v.tolist()
        } for j, v in enumerate(vectors[i: i + bulk_size])]
        helpers.bulk(es_client, actions, request_timeout=180)
    print(f"Write success! Docs count: {len(vectors)}, total cost: {time.time() - start:.2f} seconds")
    merge(es_client, index_name)

def merge(es_client, index_name, seg_cnt=1):
    print(f"Start merge! Index name: {index_name}")
    start = time.time()
    es_client.indices.forcemerge(index=index_name, max_num_segments=seg_cnt, request_timeout=7200)
    print(f"Merge success! Total cost: {time.time() - start:.2f} seconds")

# For more information about the query parameters, see .
def query(es_client, index_name, queries, gts, size=10, k=10, ef=200, msn=10000):
    print("Start query! Index name: " + index_name)
    i = 0
    precision = []
    for vec in queries:
        hits = set()
        dsl = {
            "size": size,
            "stored_fields": ["_none_"],
            "docvalue_fields": ["id"],
            "query": {
                "vector": {
                    "vec": {
                        "vector": vec.tolist(),
                        "topk": k,
                        "ef": ef,
                        "max_scan_num": msn
                    }
                }
            }
        }
        res = es_client.search(index=index_name, body=json.dumps(dsl))
        for hit in res['hits']['hits']:
            hits.add(int(hit['fields']['id'][0]))
        precision.append(len(hits.intersection(set(gts[i, :size]))) / size)
        i += 1
    print(f"Query complete! Average precision: {sum(precision) / len(precision)}")

def load_test_data(src):
```

```
    hdf5_file = h5py.File(src, "r")
    base_vectors = hdf5_file["train"]
    query_vectors = hdf5_file["test"]
    ground_truths = hdf5_file["neighbors"]
    return base_vectors, query_vectors, ground_truths

def test_sift(es_client):
    index_name = "index_sift_graph"
    vectors, queries, gts = load_test_data(r"sift-128-euclidean.hdf5")
    # Adjust the number of shards and replicas, indexing algorithm, and index parameters based on the testing requirements. In our example here, one shard and two replicas are configured for performance testing.
    create(es_client, index_name, shards=1, replicas=2, dim=128)
    write(es_client, index_name, vectors)
    query(es_client, index_name, queries, gts)

if __name__ == "__main__":
    # Change the value to the address of the CSS cluster.
    client = get_client(['http://x.x.x.x:9200'])
    test_sift(client)
```

Script prepare_query.py

```
import base64
import json
import struct

import h5py

def prepare_query(src, dst, size=10, k=10, ef=200, msn=10000, metric="euclidean", rescore=False, use_base64=True):
    """
This function is used to read query vectors from the source data files in HDF5 format and generate a complete query request body for performance testing.
    :param src: path of the source data files in HDF5 format.
    :param dst: destination file path.
    :param size: number of query results returned.
    :param k: number of top-k similar results returned by querying a segment-level index.
    :param ef: specifies the queue size used during a query.
    :param msn: specifies max_scan_num.
    :param metric: metric used for result rescoring, such as euclidean, cosine, and inner_product.
    :param rescore: whether to use rescoring. You can enable rescoring for GRAPH_PQ indexes.
    :param use_base64: whether to use Base64-encoded vector data.
    """
    hdf5_file = h5py.File(src, "r")
    query_vectors = hdf5_file["test"]
    with open(dst, "w", encoding="utf8") as fw:
        for vec in query_vectors:
            query_template = {
                "size": size,
                "stored_fields": ["_none_"],
                "docvalue_fields": ["id"],
                "query": {
                    "vector": {
                        "vec": {
                            "vector": vec.tolist() if not use_base64 else floats2base64(vec),
                            "topk": k,
                            "ef": ef,
                            "max_scan_num": msn,
                        }
                    }
                }
            }
            if rescore:
                query_template["query"]["rescore"] = {
                    "window_size": k,
                    "vector_rescore": {
                        "field": "vec",
                        "vector": vec.tolist() if not use_base64 else floats2base64(vec),
```

```
                    "metric": metric
                }
            }
        fw.write(json.dumps(query_template))
        fw.write("\n")

def floats2base64(vector):
    data = struct.pack('<{}f'.format(len(vector)), *vector)
    return base64.b64encode(data).decode()

if __name__ == "__main__":
    # Change the value to the data file address.
    prepare_query(r"/path/to/sift-128-euclidean.hdf5", r"requests.txt")
```

Script perf.lua

```
local random = math.random

local reqs = {}
local cnt = 0

-- Rename the query request file used for pressure testing as needed.
for line in io.lines("requests.txt") do
    table.insert(reqs, line)
    cnt = cnt + 1
end


local addrs = {}
local counter = 0
function setup(thread)
  local append = function(host, port)
    for i, addr in ipairs(wrk.lookup(host, port)) do
      if wrk.connect(addr) then
        addrs[#addrs+1] = addr
      end
    end
  end

  if #addrs == 0 then
    -- Change the value to the cluster address.
    append("x.x.x.x", 9200)
    append("x.x.x.x", 9200)
    append("x.x.x.x", 9200)
  end

  local index = counter % #addrs + 1
  counter = counter + 1
  thread.addr = addrs[index]
end

-- Change the index name as needed.
wrk.path = "/index_sift_graph/_search?request_cache=false&preference=_local"
wrk.method = "GET"
wrk.headers["Content-Type"] = "application/json"


function request()
    return wrk.format(wrk.method, wrk.path, wrk.headers, reqs[random(cnt)])
end
```


4.2 Using the GRAPH Algorithm to Implement Vector Search

The GRAPH algorithm (a deeply optimized implementation of HNSW) enables vector search featuring high recall and low latency based on a memory-optimized cluster. It also supports hybrid queries that combine both vector similarity search and scalar field filters.

Scenario

Vector databases that use the GRAPH indexing algorithm support vector search applications that demand high query performance and recall, such as image search, recommendation systems, and semantic search.

Solution Procedure

1. Creating vector indexes: The GRAPH algorithm is used to create vector indexes. Optimizations like edge cutting, connectivity enhancement, and SIMD acceleration are supported.
2. Associating with scalar fields: Hybrid queries that combine both vector similarity search and scalar field filters (such as labels and classes) are supported.
3. Query engine: Enables efficient vector similarity search.

Highlights

- Enhanced performance: Compared with open-source algorithms, optimizations like edge cutting, connectivity enhancement, and SIMD acceleration enhance query performance and recall.
- Flexible filtering: Hybrid queries that combine both vector and scalar fields enhance search accuracy.
- Higher accuracy: The HNSW-optimized implementation enables more accurate vector similarity matching.

Constraints

Indexes created using the GRAPH algorithm require resident memory. The optimal query performance can be achieved only when there is sufficient memory.

Prerequisites

A CSS Elasticsearch vector database has been created. The cluster version is 7.10.2, and the cluster node flavor is **memory-optimized**.

Procedure

Step 1 Log in to Kibana and go to the command execution page. Elasticsearch clusters support multiple access methods. This topic uses Kibana integrated by CSS as an example to describe the operation procedures.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
4. In the left navigation pane, choose **Dev Tools**.

   The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.

Step 2 Create a GRAPH vector index.

Create an index named **my_index** that contains a vector field **my_vector** and a label field **my_label**.

```
PUT my_index
{
  "settings": {
    "index": {
      "vector": true
    }
  },
  "mappings": {
    "properties": {
      "my_vector": {
        "type": "vector",
        "dimension": 2,
        "indexing": true,
        "algorithm": "GRAPH",
        "metric": "euclidean"
      },
      "my_label": {
        "type": "keyword"
      }
    }
  }
}
```

Step 3 Ingest vector data.

Run the following command to write the full vector data to the new GRAPH index.

- Write a single record:

  ```
  POST my_index/_doc
  {
    "my_vector": [1.0, 2.0],
    "my_label": "red"
  }
  ```

- Write multiple records at the same time:

  ```
  POST my_index/_bulk
  {"index": {}}
  {"my_vector": [1.0, 2.0], "my_label": "red"}
  {"index": {}}
  {"my_vector": [2.0, 2.0], "my_label": "green"}
  {"index": {}}
  {"my_vector": [2.0, 3.0], "my_label": "red"}
  ```

Step 4 Query vector data.

Run the following command to perform a vector search:

- Pure vector similarity search:

  ```
  POST my_index/_search
  {
    "size":3,
  ```



```
  "_source": {"excludes": ["my_vector"]},
  "query": {
    "vector": {
      "my_vector": {
        "vector": [1, 1],
        "topk":3
      }
    }
  }
}
```

- Vector+scalar hybrid search (pre-filtering query):

```
POST my_index/_search
{
  "size":3,
  "_source": {"excludes": ["my_vector"]},
  "query": {
    "vector": {
      "my_vector": {
        "vector": [1, 1],
        "topk":3,
        "filter": {
          "term": {
            "my_label": "red"
          }
        }
      }
    }
  }
}
```

If the result is returned, the query is successful.

----End

Related Documents

- For more information about the CSS vector database, see **About Vector Search**.
- To learn how to quickly get started with CSS's vector search service, see **Using Elasticsearch for Vector Search**.

4.3 Using the IVF_GRAPH_PQ Algorithm to Implement Vector Search

By combining shared, pre-built centroid indexes (GRAPH_PQ) with IVF_GRAPH_PQ vector indexes, CSS vector databases enable high-performance and low-cost search through tens of billions of vector records.

Scenario

IVF_GRAPH_PQ is a composite indexing algorithm. Its core components include:

- IVF (Inverted File): generates centroid vectors through clustering, thus partitioning the dataset into clusters by centroid. This prunes the search space in graph indexes and enhances query efficiency.
- PQ (Product Quantization): quantizes and compresses high-dimensional vectors to reduce storage and computing overheads.
- Centroid sharing: Multiple shards or segments reuse the same set of centroids, avoiding redundant clustering computations.

Vector databases that use the IVF_GRAPH_PQ algorithm are designed to handle large-scale vector search, supporting datasets exceeding 10 billion records with high-throughput ingestion (e.g., 100 million new vectors daily). Examples of such applications include image recognition, recommendation systems, and natural language processing, which all involve vector similarity search.

Solution Procedure

1. Generate centroid vectors through clustering or sampling and create a GRAPH_PQ centroid index.
2. Use the IVF_GRAPH_PQ algorithm to create a vector index and associate it with the centroid index.
3. Ingest vector data.
4. Enable efficient vector similarity search.

Highlights

- Centroid sharing: Indexes share centroid vectors across shards or segments, avoiding redundant clustering computations.
- Graph index acceleration: Search space is reduced for graph indexes, thus enhancing query performance.
- Cost reduction with PQ: Index storage, query, and computation overheads are reduced.
- Resource optimization: Indexes do not require resident memory, cutting resource costs.

Prerequisites

- A CSS Elasticsearch vector database has been created. The cluster version is 7.10.2, and the cluster node flavor is **memory-optimized**.
- Centroid vector data (accounting for 0.001% to 0.01% of the total data size) has been obtained through clustering or random sampling.

Procedure

Step 1 Log in to Kibana and go to the command execution page. Elasticsearch clusters support multiple access methods. This topic uses Kibana integrated by CSS as an example to describe the operation procedures.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
4. In the left navigation pane, choose **Dev Tools**.

   The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.

Step 2 Create a centroid index.

Run the following command to create a centroid index, for example, **my_dict**.

number_of_shards must be set to **1**, and **algorithm** must be set to **GRAPH_PQ**.

```
PUT my_dict
{
  "settings": {
    "index": {
      "vector": true
    },
    "number_of_shards": 1,
    "number_of_replicas": 0
  },
  "mappings": {
    "properties": {
      "my_vector": {
        "type": "vector",
        "dimension": 4,
        "indexing": true,
        "algorithm": "GRAPH_PQ",
        "metric": "euclidean"
      }
    }
  }
}
```

Step 3 Import centroid vectors.

Write the centroid vectors obtained through sampling or clustering into the newly created centroid index.

- Write a single record:

  ```
  POST my_dict/_doc
  {
    "my_vector": [1.0, 1.0, 1.0, 1.0]
  }
  ```

- Write multiple records at the same time.

  ```
  POST my_dict/_bulk
  {"index": {}}
  {"my_vector": [2.0, 2.0, 2.0, 2.0]}
  {"index": {}}
  {"my_vector": [3.0, 3.0, 3.0, 3.0]}
  {"index": {}}
  {"my_vector": [4.0, 4.0, 4.0, 4.0]}
  ```

Step 4 Register the centroid index.

Run the following command to register the centroid index as a Dict object with a globally unique name (dict_name). **dict_name** is configurable.

```
PUT _vector/register/my_dict
{
  "dict_name": "my_dict_name"
}
```

Step 5 Create an IVF_GRAPH_PQ index.

Run the following command to create an IVF_GRAPH_PQ vector index (for example, my_index) and associate it with the centroid index registered in the previous step through **dict_name**.

```
PUT my_index
{
  "settings": {
    "index": {
      "vector": true,
      "sort.field": "my_vector.centroid"  # Set the centroid subfield of each vector field as a ranking field.
    }
  },
  "mappings": {
```



```
  "properties": {
    "my_vector": {
      "type": "vector",
      "indexing": true,
      "algorithm": "IVF_GRAPH_PQ",
      "dict_name": "my_dict_name",
      "offload_ivf": true
    }
  }
 }
}
```

Step 6 Import the full vector data.

Run the following command to write the full vector data to the new IVF_GRAPH_PQ index.

- Write a single record:

```
POST my_index/_doc
{
  "my_vector": [2.0, 3.0, 4.0, 5.0]
}
```

- Write multiple records at the same time:

```
POST my_index/_bulk
{"index": {}}
{"my_vector": [2.0, 3.0, 4.0, 5.0]}
{"index": {}}
{"my_vector": [1.0, 2.0, 3.0, 4.0]}
{"index": {}}
{"my_vector": [3.0, 4.0, 5.0, 6.0]}
```

Step 7 Query vector data.

Run the following command to perform a vector search:

```
POST my_index/_search
{
  "size":2,
  "query": {
    "vector": {
      "my_vector": {
        "vector": [1.0, 1.0, 1.0, 1.0],
        "topk":2
      }
    }
  }
}
```

If the two most relevant results are returned, the query is successful.

----End

Related Documents

- For more information about the CSS vector database, see **About Vector Search**.
- To learn how to quickly get started with CSS's vector search service, see **Using Elasticsearch for Vector Search**.

5 Using Open-Source Logstash to Export Data in Batches from a CSS Elasticsearch Cluster

Use open-source Logstash to efficiently export data from a CSS Elasticsearch cluster for purposes like backup, migration, and analysis.

Scenarios

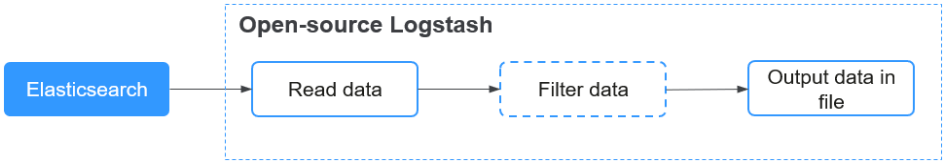
You may use this solution for the following purposes:

- Data backup: Regularly perform full backups on all indexes to ensure data security.
- Data migration: Migrate data to other storage systems (such as an object storage service or database).
- Data cleaning: Export data and preprocess it, including filtering by fields and converting formats.
- Data analysis: Export data to JSON files for offline analysis.

To export only a small amount of data (less than 10 MB), use Kibana. For details, see **Can I Export Data from Kibana in CSS?**.

Solution Architecture

Figure 5-1 Using open-source Logstash to export data from Elasticsearch



This solution consists of three key components:

- Input (reading data): The logstash-input-elasticsearch plugin connects to a CSS cluster to read data from specified indexes.

- Filter (filtering data): This one is optional. It is used for data cleaning (such as filtering data by field and converting formats).
- Output (outputting data in file): The logstash-output-file plugin writes data to a local JSON file.

Advantages

- Efficient batch processing: This solution supports both full data exporting and incremental data exporting based on filters.
- Flexible configuration: You can customize filters using Elasticsearch query DSL.
- Ecosystem compatibility: This solution fits perfectly into the Elasticsearch ecosystem. There are no additional development costs.
- Low cost: This solution uses an open-source tool.

Constraints

- The data exporting speed is affected by the load of the target CSS cluster and the performance (CPU and memory) of the ECS where the data is exported to.
- Make sure the CSS Elasticsearch cluster and Logstash versions are compatible. This example uses Logstash 7.10.2, in which case, Elasticsearch 7.10.2 is recommended.

Prerequisites

- You have obtained the address, username, and password for accessing the CSS Elasticsearch cluster.
- The ECS where you plan to export data to has been deployed and can communicate with the CSS Elasticsearch cluster (for example, they are in the same VPC and security group).

Step 1: Deploying Logstash on the Client

Deploy open-source Logstash on the ECS.

1. Log in to the ECS.
   ```
   ssh username@<ECS_IP>
   ```
2. Download and install open-source Logstash.
   ```
   cd ~
   wget https://artifacts.elastic.co/downloads/logstash/logstash-7.10.2-linux-x86_64.tar.gz
   tar -zxvf logstash-7.10.2-linux-x86_64.tar.gz
   ```
   If the ECS cannot access the Internet, download the Logstash installation package and upload it to the ECS beforehand. Download Logstash at **https://artifacts.elastic.co/downloads/logstash/logstash-7.10.2-linux-x86_64.tar.gz**.
3. Modify the Logstash memory configuration.

   Change the JVM heap size for Logstash. The default value is 1 GB. We recommend that you set it to half of the ECS memory.

   ```
   cd logstash-7.10.2
   vi config/jvm.options
   ```

   Figure 5-2 shows an example.

Figure 5-2 Changing the JVM heap size for Logstash

```
## JVM configuration

# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space

-Xms2g
-Xmx2g

################################################################
## Expert settings
################################################################
##
## All settings below this section are considered
## expert settings. Don't tamper with them unless
## you understand what you are doing
##
################################################################
```

4. Optimize Logstash's batch processing performance.

   Modify the **pipelines.yml** file and change the value of **pipeline.batch.size** to **5000**.

   ```
   vi config/pipelines.yml
   ```

   Figure 5-3 shows an example.

   Figure 5-3 Modifying pipelines.yml

   ```
   #   # How many worker threads execute the Filters+Outputs stage of the pipeline
   #   pipeline.workers: 1 (actually defaults to number of CPUs)
   #
   #   # How many events to retrieve from inputs before sending to filters+workers
     pipeline.batch.size: 5000
   #
   #   # How long to wait in milliseconds while polling for the next event
   #   # before dispatching an undersized batch to filters+outputs
   #   pipeline.batch.delay: 50
   ```

5. Test connectivity.

   Run a curl command to test the connectivity between the ECS and the CSS Elasticsearch cluster.

   ```
   curl -ik http://<CSS_IP>:9200    #No user authentication
   curl -ik https://<CSS_IP>:9200 -u <Username>:<password>   # HTTPS authentication
   ```

   If the cluster information is returned, the two are connected.

Step 2: Exporting Elasticsearch Data

1. Create a Logstash configuration file.

   Create a configuration file, for example, **es2file_all.conf**, in the config directory under the Logstash installation path.

   ```
   vi config/es2file_all.conf
   ```

2. Modify the Logstash configuration file and save the change.

   Example:

   ```
   input {
     elasticsearch {
       hosts => ["http://<CSS_IP>:9200"]
       user => "<Username>"
       password => "<Password>"
       index => "kibana_sample_data_logs"
       query => '{"query":{"range":{"@timestamp":{"gte":"now-5m","lte":"now/m"}}}}'
       docinfo => true
       size => 5000
   ```



```
  }
}

filter {
  mutate {
    remove_field => ["@version"]
  }
}

output {
  file {
    path => "./test_inc-%{+YYYY-MM-dd}.json"
  }
}
```

Table 5-1 Description of key configuration items

Configuration Item	Mandatory (Yes/No)	Description
input	Yes	The input plugin reads data from the Elasticsearch cluster.
hosts	Yes	Address for accessing the Elasticsearch cluster.
user	No	Username for accessing the Elasticsearch cluster. This parameter is mandatory for a security-mode cluster. Without it, the Elasticsearch cluster cannot be accessed.
password	No	Password for accessing the Elasticsearch cluster. This parameter is mandatory for a security-mode cluster. Without it, the Elasticsearch cluster cannot be accessed.
index	Yes	Name of the index to be exported. Wildcards (such as *) can be used to match multiple indexes. If you specify multiple index names, use a comma (,) to separate them.

Configuration Item	Mandatory (Yes/No)	Description
query	No	Use Elasticsearch query DSL to set filters. To export all data, there is no need to set this parameter. To export only part of the data, use this parameter to set filters. When setting this parameter, make sure the fields used in the filters already exist in the target indexes. Otherwise, data cannot be matched. In the example above, data of the last 5 minutes is fetched. **now-5m** indicates the last 5 minutes prior to the current time, and **now/m** indicates the current minute rounded down.
docinfo	No	Whether to include document metadata (such as index names and IDs). • **true**: to include document metadata. Such data can be useful for subsequent association analysis. • **false**: not to include document metadata. If you only need the document content, set this parameter to **false** to reduce the output size.
slices	No	Number of concurrent slices. The default value is 1. To accelerate data exporting, you can increase the value (for example, to 5 to 10), so long as your cluster has sufficient resources. Set this parameter based on the cluster load. Too many slices will increase the cluster load.

Configuration Item	Mandatory (Yes/No)	Description
size	No	Number of documents read from Elasticsearch for each request. A larger value indicates higher efficiency, but at the cost of higher memory usage. The maximum value is constrained by the Elasticsearch cluster. You are advised to set the value (for example, between 1000 and 10000) based on the ECS performance and Elasticsearch cluster load.
filter	No	The filter plugin used for data filtering. In this example, the filter plugin only deletes fields automatically added by Logstash to avoid exporting too much redundant data.
output	Yes	The output plugin can be used to specify the output file path and file naming rule. You are advised to specify an absolute path to ensure file accessibility.

3. Execute the Logstash configuration file to start the export task.

```
cd logstash-7.10.2
bin/logstash -f config/es2file_all.conf
```

4. Verify the export result.

   Go to the output file path, and open the JSON file (for example, **test_all-2025-06-05.json**). Check that the exported data meets your expectation.

6 Using Elasticsearch to Accelerate Query and Analysis for Relational Databases

This section explains how to synchronize data from a MySQL database to an Elasticsearch cluster in CSS to enable full-text search, ad hoc queries, and statistical analysis of the database.

CAUTION

The Data Replication Service (DRS) no longer supports Elasticsearch.

Scenario

Using Elasticsearch to accelerate relational databases can overcome some limitations of relational databases and enable more efficient and intelligent data processing and analysis. It is typically used in the following scenarios:

- E-commerce platform: Quickly search for commodities, provide personalized recommendations, and monitor user behavior and transaction data in real time.
- Content management system: Efficiently retrieve a large number of documents and contents, supporting complex queries and data analysis.
- Financial services: Monitor transaction data in real time for risk analysis and fraud detection.
- Social media analysis: Perform sentiment analysis, trend analysis, and influence evaluation on user-generated content.
- Customer relationship management: Quickly search for customer information, analyze customer behavior, and provide customized services.
- Log and event monitoring: Collect and analyze a large amount of log data to monitor system status and security events in real time.
- Healthcare records: Quickly retrieve and analyze patient records to support clinical decision-making and research.

Overview

Figure 6-1 Architecture of the Elasticsearch-accelerated relational database solution

1. Service data is stored in the MySQL database.
2. DRS synchronizes data from MySQL databases to the CSS Elasticsearch cluster in real time.
3. Users perform full-text search, Ad Hoc query, and statistics analysis in the Elasticsearch cluster.

Advantages

The advantages of using Elasticsearch to accelerate relational databases are as follows:

- Improved full-text search capability: Elasticsearch is a search engine that provides powerful full-text search functions. Relational databases are usually not good at full-text retrieval, but Elasticsearch can effectively solve this problem.
- High-concurrency Ad Hoc query: Elasticsearch is designed to process a large number of concurrent query requests, especially in Ad Hoc query scenarios. It can provide fast response to meet query requirements in high-concurrency environments.
- Real-time data synchronization: Data Replication Service (DRS) synchronizes data from the MySQL database to Elasticsearch in real time, ensuring data consistency and real-time performance.
- Simplified data migration and index creation: In an Elasticsearch cluster, you can create indexes that match the MySQL database table structure, simplifying data migration and index management.
- Flexible query language: Elasticsearch provides flexible query languages to support complex query construction, such as range queries, fuzzy queries, and aggregation queries. This may require more complex SQL statements in relational databases.
- Statistical analysis: The aggregation function of Elasticsearch can quickly collect and analyze data, such as age distribution statistics, which may require more computing resources and time in relational databases.
- Security and stability: You can configure security-mode Elasticsearch clusters and MySQL databases and use SSL connections between them to ensure data transmission security and system stability.
- Easy monitoring and maintenance: Elasticsearch provides various monitoring tools and APIs to facilitate system maintenance and performance monitoring.
- Scalability: The Elasticsearch cluster can be horizontally expanded based on service requirements. More nodes can be added to process larger data volumes and query loads.

These advantages make the Elasticsearch cluster an effective supplement to relational databases in processing full-text retrieval and high-concurrency Ad Hoc queries.

Prerequisites

- A MySQL database and an Elasticsearch cluster in security mode are available, and they are in the same VPC and security group.
- Data to be synchronized exists in the MySQL database.

  This document 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 business magnate, investor and philanthropist.'),
  ('2','will smith','22','also known by his stage name the Fresh Prince, is an actor, rapper, and producer.'),
  ('3','James Francis Cameron','68','the director of avatar');
  ```

- An index has been created in the Elasticsearch cluster and matches the table indexes in the MySQL database.

  Run the following command to create an index for the Elasticsearch cluster:

  ```
  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.

In this example, configure the parameters by following the suggestions in **Table 6-1**.

Table 6-1 Synchronization parameters

Module	Parameter	Suggestion
Create Synchronization Instance > **Synchronize Instance Details**	Network Type	Select **VPC**.
	Source DB Instance	Select the RDS for MySQL instance to be synchronized, that is, the MySQL database that stores service data.
	Synchronization Instance Subnet	Select the subnet where the synchronization instance is located. You are advised to select the subnet where the database instance and the Elasticsearch cluster are located.
Configure Source and Destination Databases > **Destination Database**	VPC	Select the same VPC as the Elasticsearch cluster.
	Subnet	Select the same subnet as the Elasticsearch cluster.
	IP Address or Domain Name	Enter the IP address of the Elasticsearch cluster. For details, see **Obtaining the IP address of a CSS cluster**.
	Database Username	Enter the administrator username (**admin**) and password of the Elasticsearch cluster.
	Database Password	Enter the administrator password of the Elasticsearch cluster.
	Encryption Certificate	Select the security certificate of the Elasticsearch cluster. If **SSL Connection** is not enabled, you do not need to select any certificate. For details, see **Obtaining the security certificate of a CSS cluster**.
Set Synchronization Task	Flow Control	Select **No**.
	Synchronization Object Type	Deselect **Table structure**, because the indexes matching MySQL tables have been created in the Elasticsearch cluster.
	Synchronization Object	Select **Tables**. Select the database and table name corresponding to the Elasticsearch cluster. **NOTE** Ensure the type name in the configuration item is the same as the index name, that is, **_doc**.
Process Data	-	Click **Next**.

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 Log in to Kibana and go to the command execution page.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
4. In the left navigation pane, choose **Dev Tools**.

   The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.

Step 3 Check the synchronization status of the database.

1. Verify full data synchronization.

   Run the following command in Kibana of the Elasticsearch cluster 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 Elasticsearch.

   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 a business magnate, software developer, investor, author, and philanthropist.')
   ```

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

   ```
   GET student/_search
   ```

3. Update data in the source cluster and check whether the data is synchronized to Elasticsearch.

   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 the Elasticsearch cluster to check whether the data is updated in CSS:

   ```
   GET student/_search
   ```

4. Delete data from the source cluster and check whether the data is deleted synchronously from Elasticsearch.

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

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

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

   ```
   GET student/_search
   ```

Step 4 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 the Elasticsearch cluster:

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

Step 5 Verify the ad hoc query capability of the database.

For example, query **philanthropist** whose age is greater than **40** in the Elasticsearch cluster.

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

Step 6 Verify the statistical analysis capability of the database.

For example, collect statistics on the age distributions of all users in the Elasticsearch cluster.

```
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. Log in to the **CSS management console**.

b. In the navigation pane on the left, choose **Clusters > Elasticsearch**.

c. In the cluster list, obtain the target cluster's private IP address from the **Private IP 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 this single node are displayed, for example, **10.62.179.32:9200**. If the cluster has multiple nodes and all of them are data nodes, the IP addresses and port numbers of all these nodes are displayed; if some of them are client nodes, only the IP addresses and port numbers of these client nodes are displayed; for example, **10.62.179.32:9200,10.62.179.33:9200**.

- **Obtaining the security certificate of a CSS cluster**

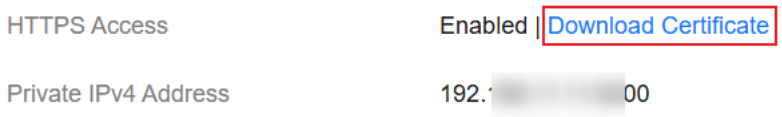
a. Log in to the **CSS management console**.

b. In the navigation pane on the left, choose **Clusters > Elasticsearch**.

c. In the cluster list, click the name of the target cluster. The cluster information page is displayed.

d. Click the **Overview** tab. In the **Configuration** area, click **Download Certificate** next to **HTTPS Access**.

Figure 6-2 Downloading a security certificate



7 Using Elasticsearch, In-House Built Logstash, and Kibana to Build a Log Management Platform

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

Scenarios

This document utilizes Elasticsearch, Filebeat, Logstash, and Kibana to illustrate the construction of a unified log management platform. Filebeat collects ECS logs and forwards them to Logstash for data processing. The processed data is stored in Elasticsearch and can be queried, analyzed, and visualized using Kibana. This solution is applicable in the following scenarios:

- Log Management: Centrally manage application and system logs to quickly identify faults.
- Security Monitoring: Detect and respond to security threats, detect intrusions, and analyze abnormal behaviors.
- Service Analysis: Analyze user behaviors to optimize products and services.
- Performance Monitoring: Monitor system and application performance in real-time to detect bottlenecks.

Overview

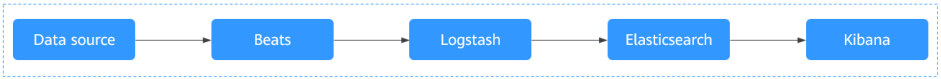
Elasticsearch, Logstash, Kibana, and Beats (ELKB) provides a complete set of log solutions and is a mainstream log system.

- Elasticsearch is an open-source, distributed search and analytics engine used to store, search, and analyze large volumes of data.
- Logstash is a server-side data pipeline that collects, parses, and enriches data before sending it to Elasticsearch.
- Kibana provides an open-source data analysis and visualization platform for Elasticsearch, enabling users to search, view, and interact with the data stored in Elasticsearch.

- Beats, such as Filebeat and Metricbeat, are lightweight data collectors installed on servers to collect and forward data to Logstash.

Figure 7-1 shows the architecture of the log management platform using Elasticsearch and Logstash.

Figure 7-1 ELKB architecture



1. Collection
   - As a data collector, Beats gather data from various sources and send it to Logstash.
   - Logstash can independently collect data or receive it from Beats to filter, transform, and enhance the data.
2. Data processing

   Before sending data to Elasticsearch, Logstash performs necessary processing, such as parsing structured logs and filtering out irrelevant information.
3. Data storage

   As a core storage component, Elasticsearch indexes and stores data from Logstash, providing quick search and data retrieval functions.
4. Data analysis and visualization

   Kibana is used to analyze and visualize data in Elasticsearch, allowing users to create dashboards and reports to visualize the data.

For details about the version compatibility of ELKB components, see **https://www.elastic.co/support/matrix#matrix_compatibility**.

Advantages

- Real-time: Provide real-time data collection and analysis capabilities.
- Flexibility: Support various data sources and flexible data processing flows.
- Ease of use: The user-friendly interface simplifies data operations and visualization.
- Scalability: Offer strong horizontal expansion capabilities, enabling the processing of petabyte-level data.

Prerequisites

- You have created an Elasticsearch cluster in non-security mode.
- You have applied for an ECS and installed the Java environment.

Procedure

Step 1 Log in to the ECS, 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 in-house.

NOTE

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

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 and the CSS cluster are connected. Run the **curl http://{ip}:{port}** command on the VM to test the connectivity between the VM and the Elasticsearch cluster. If 200 is returned, they are connected.
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 (**https://grokdebugger.com/**) to configure the **filter** mode of Logstash.

Step 3 Configure an index template in an Elasticsearch cluster.

1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the displayed cluster list, find the target cluster, and click **Access Kibana** in the **Operation** column to log in to the Kibana console.
4. In the navigation pane on the left, choose **Dev Tools**.
5. Create an index template.

   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||strict_date_optional_time||epoch_millis||EEE MMM dd HH:mm:ss zzz yyyy"
        }
      }
    }
}
```

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 &
```

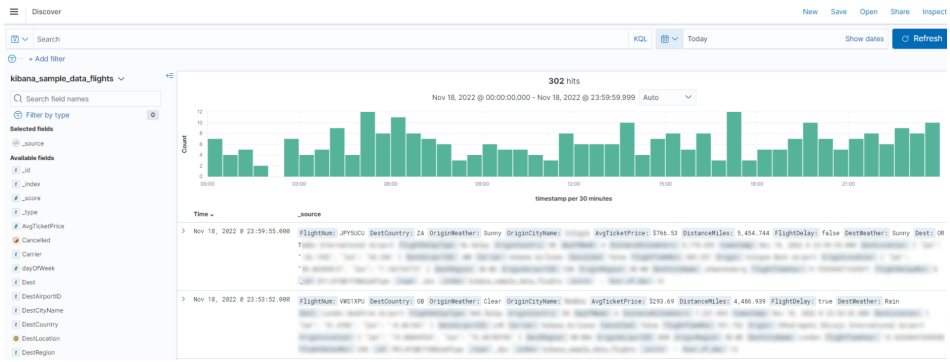
Step 6 Run the following command to start Filebeat:

```
./filebeat
```

Step 7 Use Kibana to query data and create reports.

1. Enter the Kibana page of the Elasticsearch cluster.
2. Click **Discover** in the navigation tree on the left, as shown in **Figure 7-2**.

Figure 7-2 Discover page



----End

8 Using Kibana Discover for Time Series Data Visualization

On Kibana Discover, you can create index patterns to define which Elasticsearch indexes to explore, and visualize and query time-series data using specified time fields.

Scenario

Kibana serves as the visualization layer for Elasticsearch data. On Kibana Discover, you can query time-series data using Elasticsearch's search capabilities and visualize results through dynamic charts and histograms. Typical applications include:

- Log analytics: Filter server logs by time to identify and diagnose abnormal events.
- Service monitoring: Monitor real-time changes and trends in key metrics, such as order volume and user activity.
- Data insights: Quickly browse data and verify data ingestion results.

Solution Architecture

Discover serves as Kibana's primary interface for interactive data exploration. It directly interacts with the underlying Elasticsearch cluster.

1. User requests: Users set filters, rank data, and select time ranges on the Discover page.
2. Kibana: Converts user requests into Elasticsearch query DSL.
3. Elasticsearch: Executes queries and returns the results.
4. Kibana: Visualizes the results using charts and tables (for example, showing document counts using a horizontal bar chart).

For more information, see **Discover**.

Highlights

- Easy to use: An intuitive GUI enables quick data query and browsing without complex settings.

- Flexible: Kibana's intuitive KQL (Kibana Query Language) syntax and flexible filters enable precise, real-time data discovery across your Elasticsearch indexes.
- Real-time: Data is updated in real time. You can set time fields to query the latest data.

Prerequisites

- A CSS Elasticsearch cluster has been created. For a security-mode cluster, make sure its username and password have been obtained.
- The target data for query or analytics (such as logs and monitoring metrics) has been ingested into your Elasticsearch cluster. The data must contain a time field (for example, @timestamp).

Procedure

This section uses Kibana 7.10.2 as an example to describe how to use Discover for time-series data visualization. The Kibana UI varies slightly depending on the Kibana version.

Step 1 Log in to Kibana and go to the command execution page.

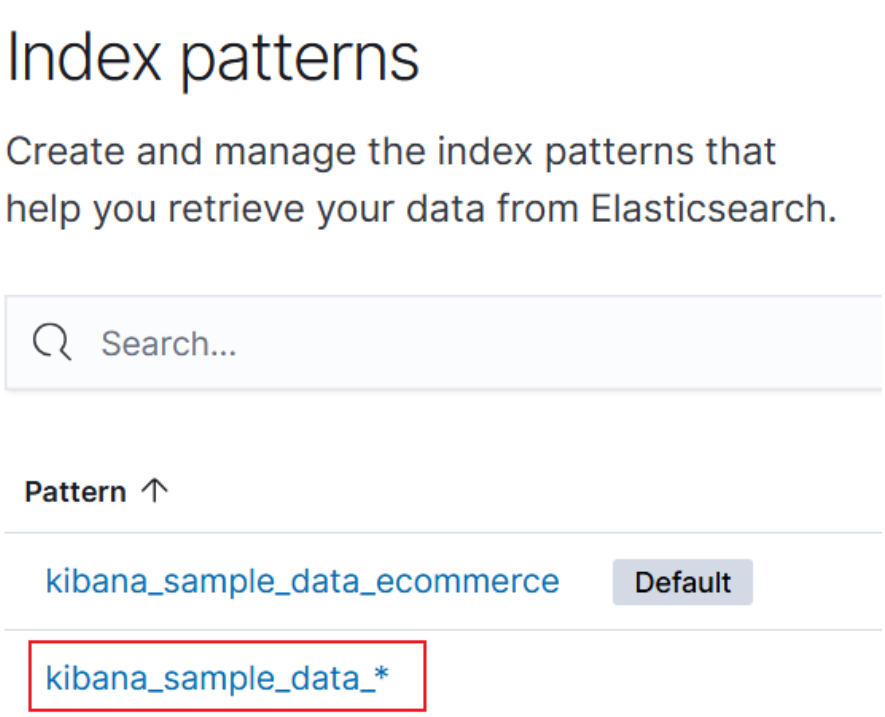
1. Log in to the **CSS management console**.
2. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
3. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
4. In the left navigation pane, choose **Dev Tools**.

   The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.

Step 2 Create an index pattern.

1. On the Kibana console, click the menu icon in the upper-left corner and choose **Management > Stack Management**.
2. On the **Stack Management** page, choose **Index Patterns** and click **Create index pattern**.
3. In the **Index pattern name** text box, enter an index name prefix (for example, **kibana_sample_data***).
4. Click **Next step**.
5. Select a time field (for example, **@timestamp**) from the **Time field** drop-down list.
6. Click **Create index pattern** to create an index pattern.

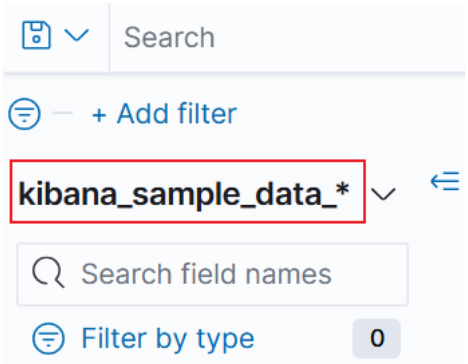
Figure 8-1 Creating an index pattern



Step 3 Go to the Discover page.

1. On the Kibana console, click the menu icon in the upper-left corner, and choose **Kibana > Discover**.
2. Select the index pattern created in the previous step (for example, **kibana_sample_data***) from the **Index patterns** drop-down list.

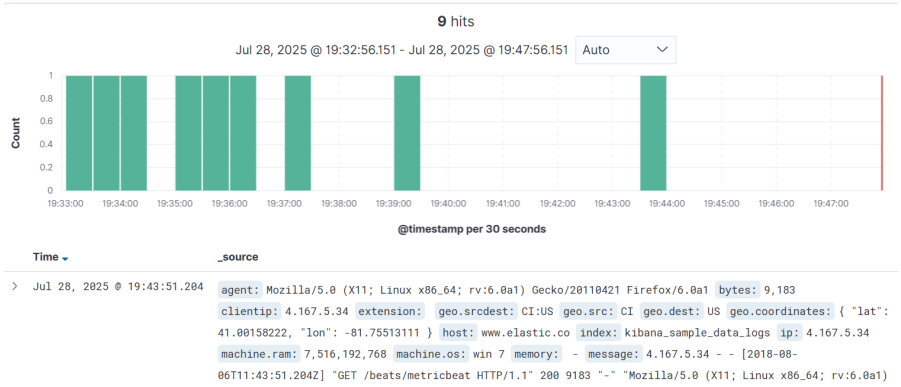
Figure 8-2 Selecting an index pattern



Step 4 Browse data.

- Bar chart: shows document counts in chronological order (aggregated every 15 minutes by default).
- Document list: shows 500 matching documents by default (you can scroll down to load more).

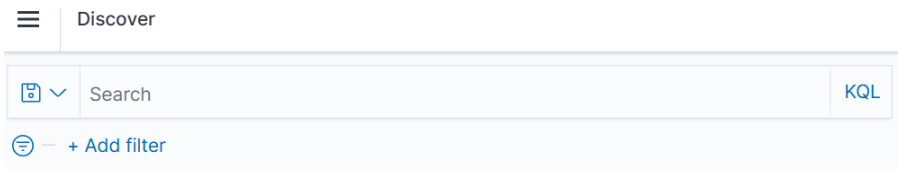
Figure 8-3 Browsing data



Step 5 Search for data.

- Add filters: In the **Add filter** area, select a field and enter a value (for example, **response.keyword: 200**) to filter the data.
- Use KQL to query data: Enter a Kuery query statement (for example, request.keyword: "/elasticsearch") in the search box and press **Enter**. For more information, see **Kibana Query Language**.

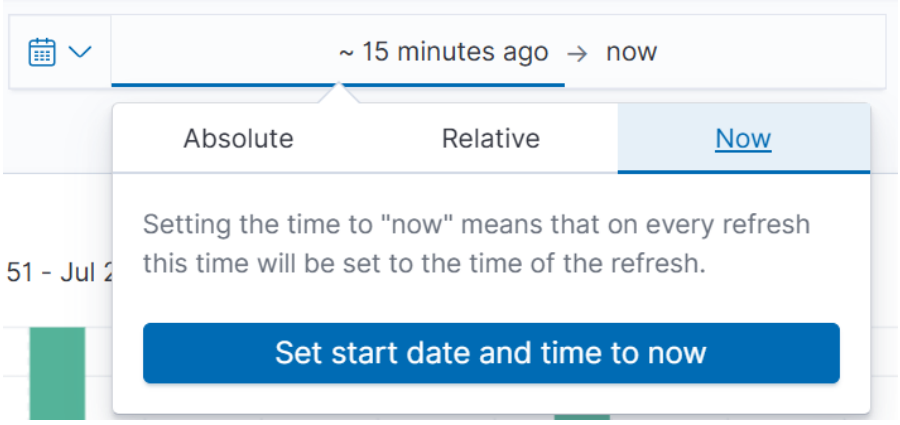
Figure 8-4 Searching for data



Step 6 Set a time filter.

1. Adjust the time range. Click the time filter in the upper-right corner and select one of the following:
   - **Absolute**: Specify the start time and end time (for example, from 2024-03-01 00:00 to 2024-03-02 00:00).
   - **Relative**: Set an offset from the current time (for example, Last 1 hour or Last 7 days).
   - **Now**: Always update to the latest refresh time.

Figure 8-5 Time filter



2. Perform an interactive query on a histogram. Click a time range in the bar chart to quickly locate documents that fall into this range.

----End

9 Ranking Search Results Using Elasticsearch Custom Rules

You can use the Elasticsearch cluster to sort the search results based on customized rules.

Scenario

Elasticsearch is a highly scalable open-source search and analysis engine. It allows users to sort search results based on customized rules. Custom sorting enables the definition of specific sorting rules based on service requirements, optimizing the relevance of search results and enhancing user experience. This solution is applicable in the following scenarios:

- E-commerce: Sort offerings based on factors such as sales volume, user comments, and prices.
- Content Management: Sort articles or blog entries based on the number of views and publishing time.
- Financial Services: Sort transaction records based on transaction amount, frequency, or risk score.
- Customer Support: Sort customer requests based on the urgency or opening time of service tickets.

Solution Architecture

The sorting API in Elasticsearch is used to sort search results according to customized rules. By calling the sorting API, you can query and sort data based on customized rules.

You can query with customized rules using either of the following methods:

- 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.

Advantages

- Flexibility: Customized sorting rules can meet various complex service requirements.
- Scalability: The distributed nature of Elasticsearch supports horizontal expansion to accommodate increasing data volumes.
- Performance: Elasticsearch's optimization mechanisms ensure efficient sorting operations, maintaining good performance even with large-scale datasets.
- Real-time: The near real-time search capability of Elasticsearch ensures the timeliness of sorting results.

Prerequisites

An Elasticsearch cluster 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 Kibana and go to the command execution page.
   a. Log in to the **CSS management console**.
   b. In the navigation pane on the left, choose **Clusters > Elasticsearch**.
   c. In the cluster list, find the target cluster, and click **Kibana** in the **Operation** column to log in to the Kibana console.
   d. In the left navigation pane, choose **Dev Tools**.
      The left part of the console is the command input box, and the triangle icon in its upper-right corner is the execution button. The right part shows the execution result.
2. 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"
      }
    }
  }
}
```

3. 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 }
   ```

4. Query data based on customized rules. 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
      }
    }
  ]
 }
}
```


10 Cloud Search Service Security Best Practices

CSS offers a fully-managed, distributed search service built on open-source Elasticsearch and OpenSearch. It enables efficient search, analysis, and visualization of both structured and unstructured data, making it an ideal choice for log analytics, data-driven operations and maintenance, and intelligent search applications. This section provides actionable best practices for enhancing CSS security. Based on them, you can continuously evaluate the security posture of your CSS resources, and enhance their security by combining multiple security features provided by CSS. This way, you protect your data stored in CSS against leakage and tampering—both at rest and in transit.

To secure your data and workloads on CSS, we recommend that you follow the best practices below:

- **Configuring Security Settings**: Lower the risk of cyber attacks on clusters.
- **Improving Account and Password Security**: Lower the risk of data breaches.
- **Enhancing Permission Management**: Lower security risks.
- **Enabling Security Audit Logs**: Enable post-event review and backtracking.
- **Enabling Data Backup**: Improve data reliability.
- **Encrypting Data at Rest**: Enhance data security.
- **Upgrading Your Clusters to the Latest Version**: Get better user experience and enhanced security.

Configuring Security Settings

- Creating security-mode clusters

  CSS provides a security mode for Elasticsearch and OpenSearch clusters. A security-mode enabled cluster requires user authentication using a username and password, while a non-security mode cluster does not require that. You should not create clusters with the security mode disabled unless you are in a tightly controlled environment or your data is of minor importance. For details about how to change a cluster's security mode, see or .

- Enabling HTTPS access

  Without the use of Secure Sockets Layer (SSL), data transmitted between a CSS client and server is in plaintext, making it vulnerable to eavesdropping,

tampering, and man-in-the-middle attacks. To improve data security in transit, you are advised to create a cluster by enabling the security mode as well as HTTPS, which uses the SSL protocol to encrypt data. Enabling HTTPS introduces encryption and decryption overhead, which may reduce cluster performance by approximately 20%. Before enabling HTTPS access, carefully weigh its security benefits against potential performance losses. For details about how to switch between HTTPS and HTTP for a cluster, see or .

- Avoiding exposing a cluster to the Internet via an EIP

  Avoid deploying CSS clusters on the public network or in a DMZ. Instead, deploy them on your company's internal network, where they are protected by routers or firewalls. To prevent unauthorized access and reduce the likelihood of DDoS attacks, avoid exposing them to the Internet by binding an EIP directly to them. When possible, disable public network access to your clusters. If public network access is unavoidable, use a dedicated load balancer for your clusters and apply strict security group rules to the load balancer. For details, see or .

Improving Account and Password Security

- Resetting the administrator password periodically

  Security-mode enabled Elasticsearch and OpenSearch clusters have a default administrator account **admin**, which has full permissions on the clusters. You are advised to reset its password periodically. Doing so improves account security as well as the security of sensitive data stored in the clusters. For details, see

- Using more complex passwords

  As distributed search and analytics engines, CSS's Elasticsearch and OpenSearch clusters are desirable targets for cyber attacks. You must keep your accounts and passwords secure. Additionally, you should use more complex passwords, instead of weak ones. CSS checks the complexity of administrator passwords set by users. The password must contain at least 12 characters, and must be a combination of uppercase letters, lowercase letters, digits, and special characters (such as !@#$%). This should make your password strong enough.

Enhancing Permission Management

- Configuring IAM users for fine-grained permission management

  CSS allows you to use IAM to implement fine-grained user permission management and better resource isolation by department or project. For more information, see .

- Avoiding using the administrator account to access clusters

  Security-mode enabled Elasticsearch and OpenSearch clusters have a default administrator account **admin**, which has full permissions on the clusters. To access data stored in clusters, you are advised to a non-administrator account. Additionally, you can configure index access permissions for common users through Kibana or OpenSearch Dashboards. You can configure user permissions based on users, permissions, and roles to suit your needs. For more information, see or .

Enabling Security Audit Logs

Security audit logs record all user operations on data and indexes. They can be used to analyze user behavior, generate compliance reports, and trace the root cause of an incident. For more information, see

Enabling Data Backup

CSS's Elasticsearch and OpenSearch clusters support both automatic and manual data backup. Depending on data importance, you are advised to perform backup every day or every week and retain multiple backups. In the event of a cluster failure or data corruption, you can use one of the backups to restore data. For more information, see or .

Encrypting Data at Rest

When creating a CSS cluster, you are advised to enable disk encryption, which uses the Key Management Service (KMS) to encrypt data before storing it on disks. This reduces the risk of data leakage in the event of stolen physical disks or unauthorized access. To enable disk encryption, submit a .

Upgrading Your Clusters to the Latest Version

Based on newly discovered vulnerabilities disclosed in the open-source community, CSS may release new kernel versions for Elasticsearch and OpenSearch to incorporate the needed fixes to these vulnerabilities. To improve the ease-of-use and security of your CSS clusters, you are advised to check for new cluster software versions quarterly, install security patches in a timely manner, and always upgrade your clusters to the latest images. For details, see or .

