Cloud Search Service

Best Practices

Issue 02

Date 2024-04-22

HUAWEI TECHNOLOGIES CO., LTD.

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process.* For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Cluster and Index Planning....1

2 Permission Configuration....5

2.1 Granting IAM Users the Permission to Create CSS Clusters....5

2.2 Creating an Elasticsearch User and Configuring Index Permissions....15

3 Cluster Migration....24

3.1 Migration Solution Overview....24

3.2 Migration from Elasticsearch....26

3.2.1 Using Logstash to Perform Full Data Migration....26

3.2.2 Using Logstash to Perform Incremental Data Migration....42

3.2.3 Migrating Data Through Backup and Restoration (from CSS Elasticsearch)....54

3.2.4 Migrating Data Through Backup and Restoration (from Third-Party Elasticsearch)....57

3.3 Migration from Kafka/MQ....59

3.4 Migration from a Database....61

4 Cluster Access....63

4.1 Overview....63

4.2 Accessing an Elasticsearch Cluster....64

4.3 Accessing a Cluster Using cURL Commands....65

4.4 Accessing a Cluster Using Java....66

4.4.1 Accessing a Cluster Through the Rest High Level Client....66

4.4.2 Accessing a Cluster Through the Rest Low Level Client....76

4.4.3 Accessing the Cluster Through the Transport Client....90

4.4.4 Using Spring Boot to Access a Cluster....91

4.5 Accessing a Cluster Using Python....98

4.6 Using ES-Hadoop to Read and Write Data in Elasticsearch Through Hive....100

4.7 Accessing a Cluster Using Go....107

5 Cluster Performance Tuning....111

5.1 Optimizing Write Performance....111

5.2 Optimizing Query Performance....114

6 Managing the Index Lifecycle....117

6.1 Configuring the Lifecycle to Automate Index Rollover....117

6.2 Configuring the Lifecycle to Decouple Storage and Compute....120

7 Practices.......... 124
7.1 Using CSS to Accelerate Database Query and Analysis.......... 124
7.2 Using CSS to Build a Unified Log Management Platform.......... 129
7.3 Configuring Query Scoring in an Elasticsearch Cluster.......... 133

1 Cluster and Index Planning

In Cloud Search Service (CSS), you can select the cluster version, architecture, storage type, number of cluster nodes, storage capacity, and number of index shards. Configure them based on your service requirements for read and write requests, data storage and computing, and search and analytics.

You can configure the following specifications of a CSS cluster:

- **Cluster Version**
- **Cluster Architecture**
- **Storage Types**
- **Cluster Nodes**
- **Node Storage Capacity**
- **Number of Index Shards**

Cluster Version

CSS supports Elasticsearch versions 7.6.2 and 7.10.2. You are advised to select a version based on the following principles:

1. For a new Elasticsearch cluster, select version 7.10.2 or 7.6.2.
2. If Elasticsearch cluster migration and code reconstruction are required, select 7.10.2 or 7.6.2. Otherwise, you are advised to use the same version as the previous major version.

Cluster Architecture

CSS supports multiple architectures, such as read/write splitting, cold and hot isolation, decoupled storage and computing, role separation, and cross-AZ deployment. The following table shows their applicable scenarios.

Architecture	Scenario	Benefits
Read/ Write splitting	Production services involving many read operations and only a few write operations. After data is written, it does not need to be accessed within 10s.	High concurrency, low latency

Architecture	Scenario	Benefits
Cold and hot data separation	Log services that have low requirements on cold data query performance.	Low costs
Decoupled storage and compute	Log services that have low requirements on cold data query performance (10s+) and do not require cold data update. This architecture can be used together with the cold and hot data separation architecture to build three levels of storage: hot, warm, and cold.	Low costs
Separation of roles	A cluster that is large, has a large number of indexes, or is highly scalable.	High availability
Cross-AZ deployment	Production services that have high requirements on availability or use local disks.	High availability

Storage Types

CSS supports cloud and local disks.

- Cloud disk types include computing-intensive (CPU:memory = 1:2), general computing (CPU:memory = 1:4), and memory-optimized (CPU:memory = 1:8).
- Local disk types include disk-intensive (with HDDs attached) and ultra-high I/O (with SSDs attached).

  The following table shows their applicable scenarios.

Table 1-1 Applicable scenarios of storage types

Type	Scenario
Computing-intensive	**Recommended scenario**: search from a small amount of data (less than 100 GB on a single node)
General computing	**Common scenario**: search and analysis when the data volume on a single node is in the range 100 GB to 1,000 GB, for example, medium-scale e-commerce search, social search, and log search
Memory-optimized	**Common scenario**: search and analysis when the data volume of a single node is in the range 100 GB to 2,000 GB **Vector search**: Large memory helps improve cluster performance and stability.
Disk-intensive	**Logs**: Cold data needs to be stored and updated, and the requirements on cold data query performance is low.

Type	Scenario
Ultra-high I/O - Kunpeng	**Large-scale logs**: hot data storage
Ultra-high I/O - x86	**Large-scale search and analysis**: High computing or disk I/O performance is required, such as public opinion analysis, patent search, and database acceleration.

Cluster Nodes

After the architecture and storage type of a CSS cluster are selected, determine the number of nodes in the cluster based on your performance requirements.

Table 1-2 Node quantity calculation methods

Type	Performance Baseline	Node Quantity Calculation Method	Example
Write node	For a node with a cloud disk, the write performance baseline of a single vCPU is 1 MB/s. For an ultra-high I/O node, the write performance baseline of a single vCPU is 1.5 MB/s.	Traffic during peak hours/Number of vCPUs on a single node/Write performance baseline of a single vCPU x Number of copies	If the peak write rate is 100 MB/s and a node has 16 vCPUs and 64 GB memory, 12 nodes (100/16/1 x 2) are required.
Query node	The performance of the same node varies greatly in different scenarios. It is difficult to evaluate the performance baseline of a single node. The **average query response time** is used as the query performance baseline for calculation.	QPS/(Number of vCPUs on a single node x 3/2/Average query response time per second) x Number of shards	If the query QPS is 1000, the average query response time is 100 ms, three index shards are planned, and a node has 16 vCPUs and 64 GB memory, about 12 nodes (1000/(16 x 3/2/0.1) x 3) are required.
Number of nodes	/	Number of nodes = Number of write nodes + Number of query nodes	Number of nodes = Number of write nodes + Number of query nodes = 24

If two clusters can achieve the same performance, you are advised to select the one using higher specifications and fewer nodes. For example, a cluster using 3 nodes with 32 vCPUs and 64 GB memory achieves the same performance as the one using 12 nodes with 8 vCPUs and 16 GB memory, but the former runs more stable and can be more easily scaled. For a high-specification cluster that reaches the performance bottleneck, you simply need to scale it out (by adding nodes); whereas for a low-specification cluster, you need to scale it up (by changing to higher specifications).

Node Storage Capacity

The disk space of each node in a CSS cluster is determined by multiple factors, such as the data volume, number of copies (often set to 1), data bloat rate, and disk space usage (often set to 70%). You can use the following formula to calculate the storage capacity of a cluster:

Storage capacity = Source data x (1 + Number of copies) x 1.25 x (1 + Reserved space) ≈ Source data x 2 x 1.25 x 1.3 = Source data x 3.25

Number of Index Shards

You are advised to plan the number of index shards in a CSS cluster based on the following principles:

1. The size of a single shard is in the range 10 GB to 50 GB.
2. A cluster has fewer than 30,000 shards.
3. It is recommended that 1 GB memory be used for 20 to 30 shards, and that a single node have no more than 1,000 shards.
4. For a single index, it is recommended that the number of index shards be the same as or a multiple of the number of nodes.

2 Permission Configuration

2.1 Granting IAM Users the Permission to Create CSS Clusters

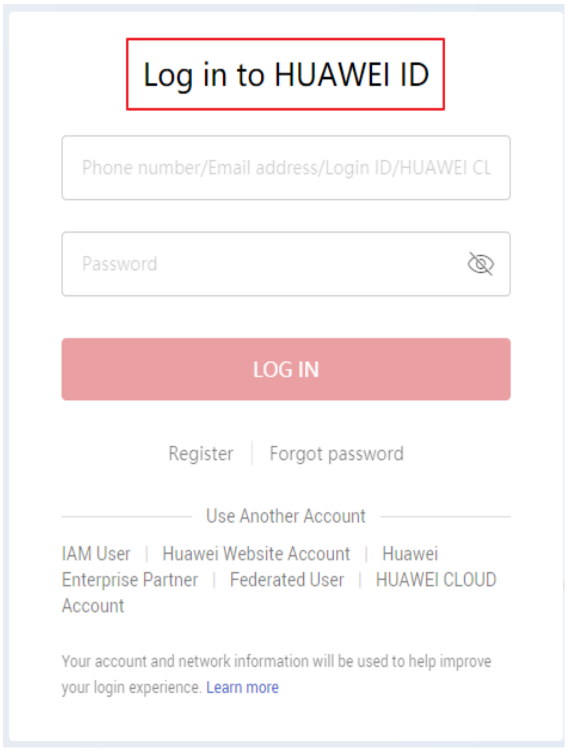
To implement fine-grained permission management for CSS, you can use Identity and Access Management (IAM) to create independent IAM users and assign policies or roles to IAM user groups. The policies and roles can be used to control access to CSS resources.

This section describes how to create an IAM user and add the IAM user to a user group, so that the IAM user has the permission to create CSS clusters.

Step 1: Create a User Group and Assign Policy

Step 1 Use your Huawei ID to enable Huawei Cloud services, and then log in to Huawei Cloud.

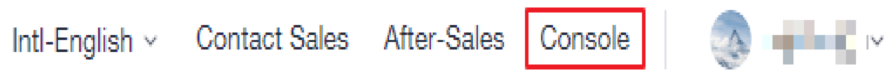
Figure 2-1 Logging in to Huawei Cloud


Log in to HUAWEI ID
Phone number/Email address/Login ID/HUAWEI CL
Password
LOG IN
Register
Forgot password
Use Another Account
IAM User | Huawei Website Account | Huawei Enterprise Partner | Federated User | HUAWEI CLOUD Account

Your account and network information will be used to help improve your login experience. Learn more

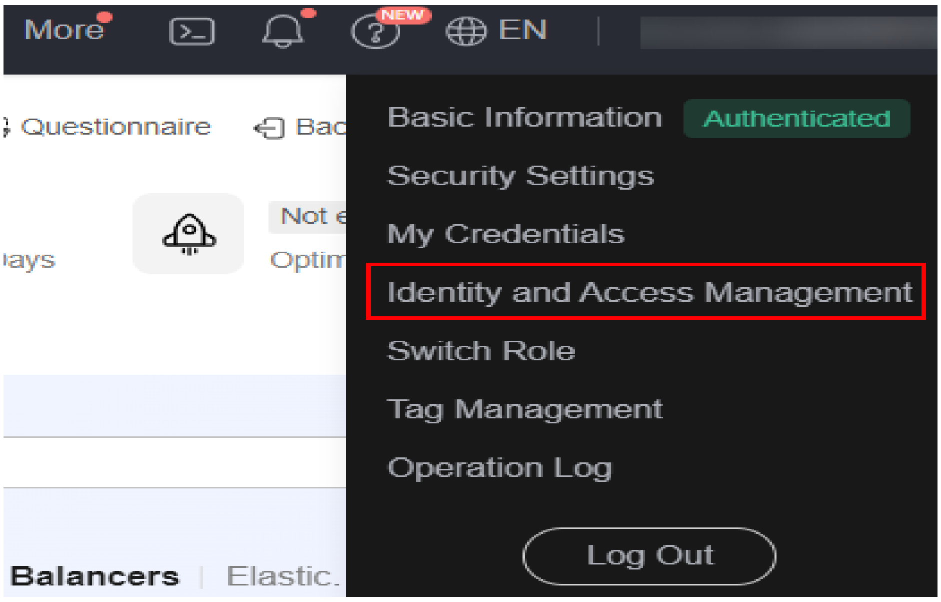
Step 2 Click **Console** in the upper right corner.

Figure 2-2 Accessing the console



Step 3 On the management console, hover the mouse pointer over the username in the upper right corner, and choose **Identity and Access Management** from the drop-down list.

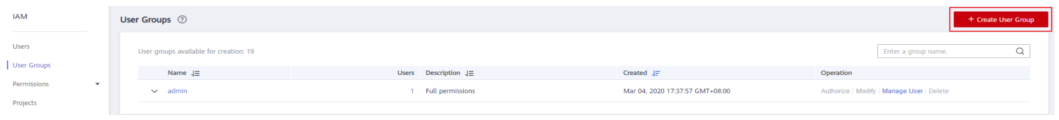
Figure 2-3 Accessing the IAM console



Step 4 Log in to the Huawei Cloud console and choose **Identity and Access Management**.

Step 5 On the IAM console, choose **User Groups** and click **Create User Group**.

Figure 2-4 Creating a user group

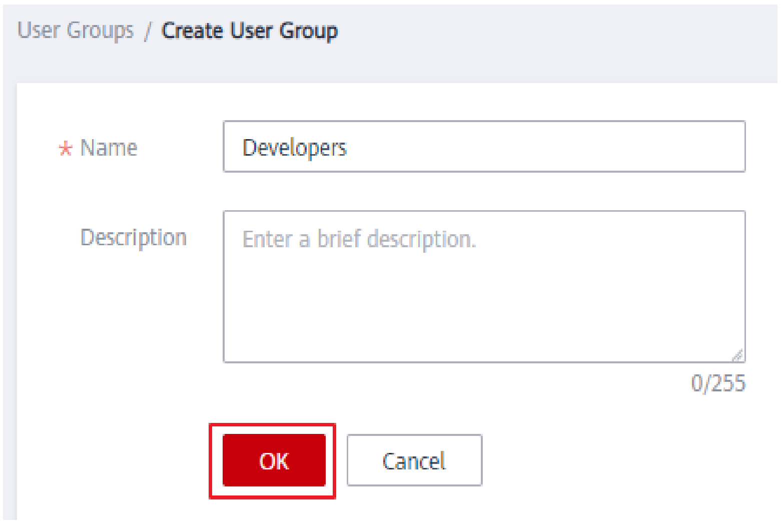


User groups available for creation: 19

Name	Users	Description	Created	Operation
admin	1	Full permissions	Mar 04, 2020 17:37:57 GMT+08:00	Authorize | Modify | Manage User | Delete

Step 6 Enter **Developers** for **Name**, and click **OK**.

Figure 2-5 Setting the user group information



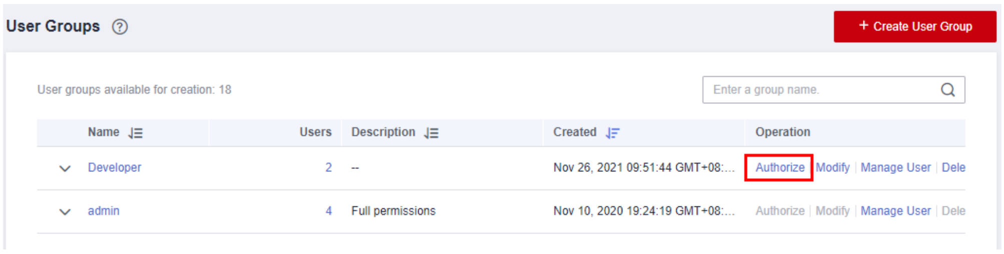
* Name	Developers
Description	

----End

Step 2: Grant Permissions to a User Group

Step 1 In the user group list, click **Authorize** in the row containing the newly created user group.

Figure 2-6 Authorizing a user group



User Groups

+ Create User Group

User groups available for creation: 18

Enter a group name.

	Name	Users	Description	Created	Operation
	Developer	2	--	Nov 26, 2021 09:51:44 GMT+08:...	Authorize | Modify | Manage User | Dele
	admin	4	Full permissions	Nov 10, 2020 19:24:19 GMT+08:...	Authorize | Modify | Manage User | Dele

Step 2 In the **Select Policy/Role** step, search for **CSS FullAccess** in the search box, select it, and click **Next**.

- Generally, the permissions for creating a cluster include **CSS FullAccess** and **Elasticsearch Administrator**. You can configure the permissions based on the relationship between common operations and system permissions in **Table 2-1**. For more information, see **Table 2-2**.
- If users in the group need to view resource usage, attach the **BSS Administrator** role to the group for the same project.

Table 2-1 Common operations supported by each system-defined policy

Operation	CSS FullAccess	CSS ReadOnlyAccess	Elasticsearch Administrator	Remarks
Creating a cluster	√	x	√	-
Querying the cluster list	√	√	√	-
Querying cluster details	√	√	√	-
Deleting a cluster	√	x	√	-
Restarting a cluster	√	x	√	-
Expanding cluster capacity	√	x	√	-
Adding instances and expanding storage	√	x	√	-

Operation	CSS FullAccess	CSS ReadOnlyAccess	Elasticsearch Administrator	Remarks
Querying tags of a specified cluster	√	√	√	-
Querying all tags	√	√	√	-
Loading a custom word dictionary	√	x	√	Depends on OBS and IAM permissions
Querying the status of a custom word dictionary	√	√	√	-
Deleting a custom word dictionary	√	x	√	-
Automatically setting basic configurations of a cluster snapshot	√	x	√	Depends on OBS and IAM permissions
Modifying basic configurations of a cluster snapshot	√	x	√	Depends on OBS and IAM permissions
Setting the automatic snapshot creation policy	√	x	√	-
Querying the automatic snapshot creation policy	√	√	√	-
Manually creating a snapshot	√	x	√	-
Querying the snapshot list	√	√	√	-
Restoring a snapshot	√	x	√	-

Operation	CSS FullAccess	CSS ReadOnlyAccess	Elasticsearch Administrator	Remarks
Deleting a snapshot	√	x	√	-
Disabling the snapshot function	√	x	√	-
Modifying specifications	√	x	√	-
Scaling in clusters	√	x	√	-

Table 2-2 CSS system permissions

Role/Policy Name	Type	Role/Policy Description	Dependencies
Elasticsearch Administrator	System-defined role	Full permissions for CSS. This role depends on the **Tenant Guest** and **Server Administrator** roles in the same project.	● Tenant Guest: A global role, which must be assigned in the global project. ● Server Administrator: A project-level role, which must be assigned in the same project
CSS FullAccess	System policy	Full CSS permissions granted through policies. Users with these permissions can perform all operations on CSS.	None
CSS ReadOnlyAccess	System policy	Read-only permissions for CSS. Users with these permissions can only view CSS data.	None

Step 3 Select a scope.

Take the AP-Singapore region as an example. Set **Scope** to **Region-specific projects** and select **ap-southeast-3 [AP-Singapore]**.

Step 4 Click **OK**.

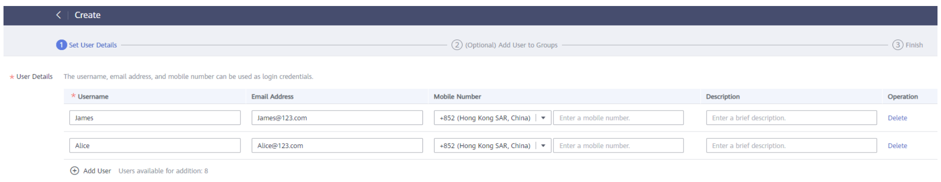
----End

Step 3: Create an IAM User and Add It to the User Group

Step 1 In the navigation pane, choose **Users**. Click **Create User**.

Step 2 Configure basic information. On the **Create User** page, Configure **User Details** and **Access Type**. To create more users, click **Add User**. A maximum of 10 users can be created at a time.

Figure 2-7 Configuring user information



* User Details The username, email address, and mobile number can be used as login credentials.

* Username	Email Address	Mobile Number		Description	Operation
James	James@123.com	+852 (Hong Kong SAR, China)	Enter a mobile number.	Enter a brief description.	Delete
Alice	Alice@123.com	+852 (Hong Kong SAR, China)	Enter a mobile number.	Enter a brief description.	Delete

Add User Users available for addition: 8

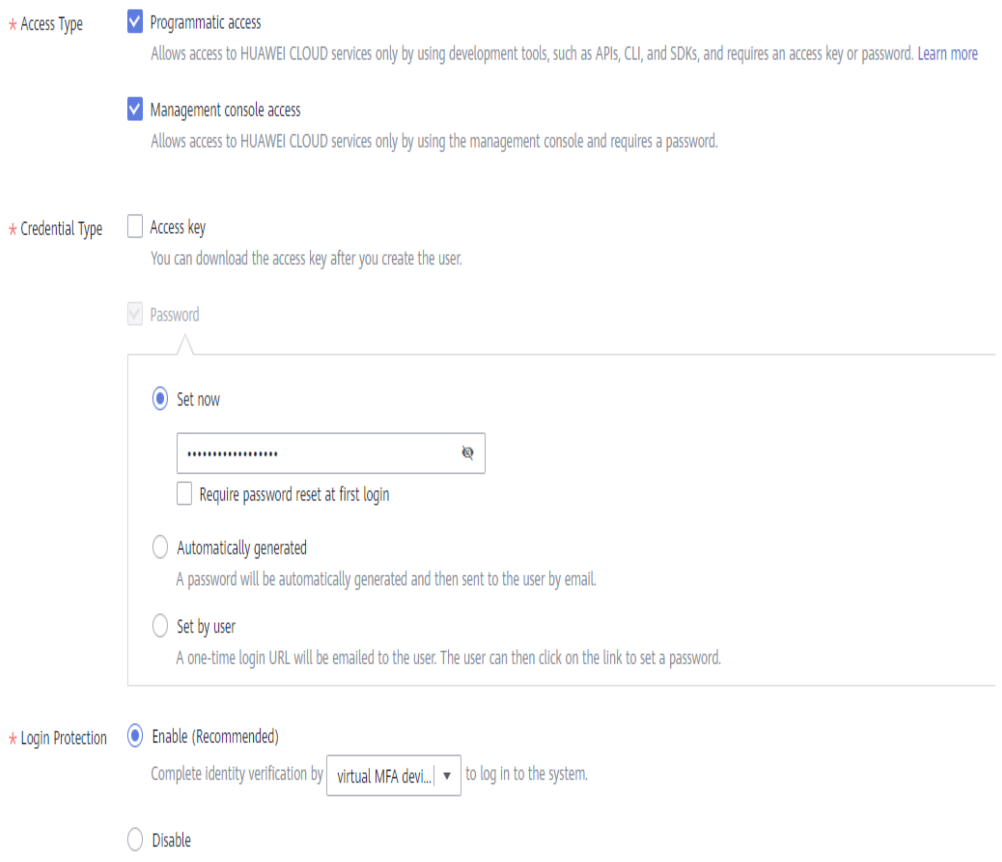
NOTE

- Users can log in to Huawei Cloudthe cloud platform using the username, email address, or mobile number.
- If users forget their password, they can reset it through email address or mobile number verification. If no email addresses or mobile numbers have been bound to users, users need to request the administrator to reset their passwords.

Table 2-3 User information

Parameter	Description
Username	Mandatory. Username that will be used to log in to HUAWEI CLOUD, for example, **James** and **Alice**.
Email Address	This parameter is mandatory if you choose **Credential Type** > **Password** > **Require password reset at first login**. The email address of an IAM user that can be used as a login credential. After IAM users are created, they can also bind email addresses.
Mobile Number	Optional. Mobile phone number of the IAM user to use as a login credential. IAM users can bind mobile numbers by themselves after being created.
Description	Optional. Additional information about the IAM user.

Figure 2-8 Setting the access type


* Access Type
[✓] Programmatic access
Allows access to HUAWEI CLOUD services only by using development tools, such as APIs, CLI, and SDKs, and requires an access key or password. Learn more
[✓] Management console access
Allows access to HUAWEI CLOUD services only by using the management console and requires a password.

* Credential Type
[ ] Access key
You can download the access key after you create the user.
[✓] Password
(●) Set now
••••••••••••••••••
[ ] Require password reset at first login
( ) Automatically generated
A password will be automatically generated and then sent to the user by email.
( ) Set by user
A one-time login URL will be emailed to the user. The user can then click on the link to set a password.

* Login Protection
(●) Enable (Recommended)
Complete identity verification by virtual MFA devi... to log in to the system.
( ) Disable

- **Programmatic access**: Select this option to allow the user to access cloud services using development tools, such as APIs, CLI, and SDKs. You can generate an **access key** or set a **password** for the user.
- **Management console access**: Select this option to allow the user to access cloud services using the management console. You can set or generate a password for the user or request the user to set a password at first login.

NOTE

  – If the user **accesses cloud services only by using the management console**, select **Management console access** for **Access Type** and **Password** for **Credential Type**.
  – If the user **accesses cloud services only through programmatic calls**, select **Programmatic access** for **Access Type** and **Access key** for **Credential Type**.
  – If the user **needs to use a password as the credential for programmatic access** to certain APIs, select **Programmatic access** for **Access Type** and **Password** for **Credential Type**
  – If the user needs to **perform access key verification** when using certain services in the console, select **Programmatic access** and **Management console access** for **Access Type** and **Access key** and **Password** for **Credential Type**. For example, the user needs to perform access key verification when creating a data migration job in the Cloud Data Migration (CDM) console.

Table 2-4 Setting the credential type and login protection

Credential Type and Login Protection		**Description**
Access Key		After creating the user, you can download the **access key (AK/SK)** generated for the user. **Each user can have a maximum of two access keys.**
Password	Custom images	Set a password for the user and determine whether to require the user to reset the password at first login. If you are the user, select this option and set a password for login. You do not need to select **Require password reset at first login**.
	Automatically generated	The system automatically generates a login password for the user. After the user is created, download the EXCEL password file and provide the password to the user. The user can then use this password for login. **This option is available only when you create a single user.**
	Set by user	A one-time login URL will be emailed to the user. The user can click the link to log in to the console and set a password. If you do not use the IAM user, select this option and enter the email address and mobile number of the IAM user. The user can then set a password by clicking the one-time login URL sent over email. The login URL is valid for **seven days**.
Login Protection	Enable (Recommended)	If login protection is enabled, the user will need to enter a verification code in addition to the username and password during login. Enable this function for account security. You can select SMS, email, or virtual MFA device for verification during login.
	Disable	To enable login protection for an IAM user after creation, see **Modifying IAM User Information**.

Step 3 Click **Next** and add the user to the user group created in **Step 1: Create a User Group and Assign Policy**.

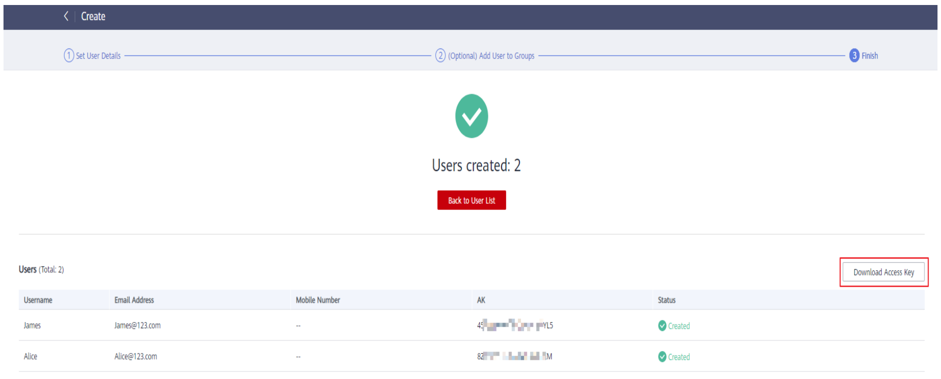
The user will inherit the permissions assigned to the user groups to which the user belongs.

NOTE

The default user group **admin** has the administrator permissions and the permissions required to use all cloud resources.

Step 4 Click **Create**. If you have specified the access type as **Programmatic access** and selected **Access key** for **Credential Type** (see **Table 2-4**), you can download the access keys on the **Finish** page.

Figure 2-9 User created successfully



Users (Total: 2)

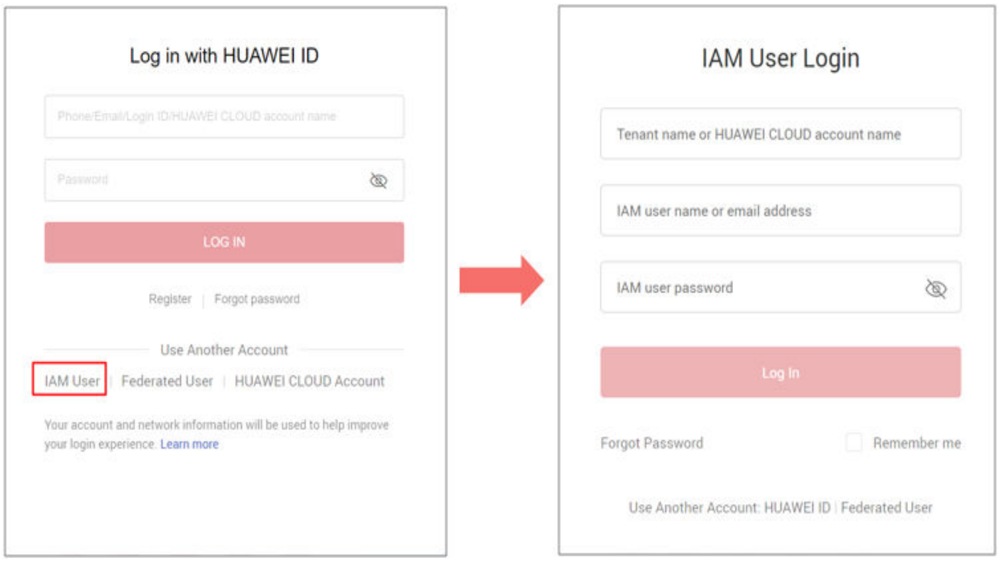
Username	Email Address	Mobile Number	AK	Status
James	James@123.com	--	4[redacted]YL5	✓ Created
Alice	Alice@123.com	--	82[redacted]M	✓ Created

----End

Step 4: Log In as an IAM User and Verify Permissions

Step 1 Click **IAM UserIAM User Login** on the login page, and then enter your **Tenant name or HUAWEI CLOUD account name**, **IAM user name or email address**, and **IAM user password**.

Figure 2-10 IAM user login


Log in with HUAWEI ID
Phone/Email/Login ID/HUAWEI CLOUD account name
Password
LOG IN
Register | Forgot password
Use Another Account
IAM User | Federated User | HUAWEI CLOUD Account
Your account and network information will be used to help improve your login experience. Learn more
IAM User Login
Tenant name or HUAWEI CLOUD account name
IAM user name or email address
IAM user password
Log In
Forgot Password
Remember me
Use Another Account: HUAWEI ID | Federated User

- **Tenant name or Huawei Cloud account name**: the name of the account that was used to create the IAM user.
- **IAM user name or email address**: the username (for example, **James**) or email address of the IAM user. IAM users can obtain their username and password from the administrator.
- **IAM user password**: the password of the IAM user (not the password of the account)

Step 2 Click **Log In**.

Step 3 Click **Service List** and choose **Cloud Search Service**.

Step 4 In the upper right corner of the **Dashboard** page, click **Create Cluster**. Create a cluster by following the steps provided in **Creating a Cluster**. If the cluster can be created, the permissions have taken effect.

----End

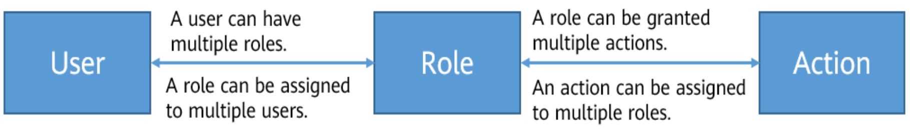
2.2 Creating an Elasticsearch User and Configuring Index Permissions

You can use the Role-Based Access Control (RBAC) model in an Elasticsearch security cluster of version 7.6.2.

Context

CSS uses the opendistro_security plug-in to provide security cluster capabilities. The opendistro_security plug-in is built based on the RBAC model. RBAC involves three core concepts: user, action, and role. RBAC simplifies the relationship between users and actions, simplifies permission management, and facilitates permission expansion and maintenance. The following figure shows the relationship between the three.

Figure 2-11 User, action, and role


User
A user can have multiple roles.
A role can be assigned to multiple users.
Role
A role can be granted multiple actions.
An action can be assigned to multiple roles.
Action

In addition to the RBAC model, Elasticsearch has an important concept called tenant. RBAC is used to manage user authorization, and tenants are used for information sharing across tenants. In a tenant space, IAM users can share information such as dashboard data and index patterns.

By default, users can view only the index patterns and dashboard information in their own private tenant spaces. Create a user named **test**. By default, a **.kibana_xxx_test** index is created to store the content of the private space of **test**. Similarly, the private tenant space of the **admin** account is stored in **.kibana_xxx_admin**. To share the index pattern of the current tenant or another tenant on the dashboard, you can create a global tenant space. Other users can switch to the global tenant space to access shared data.

Creating a User and Assigning Permissions

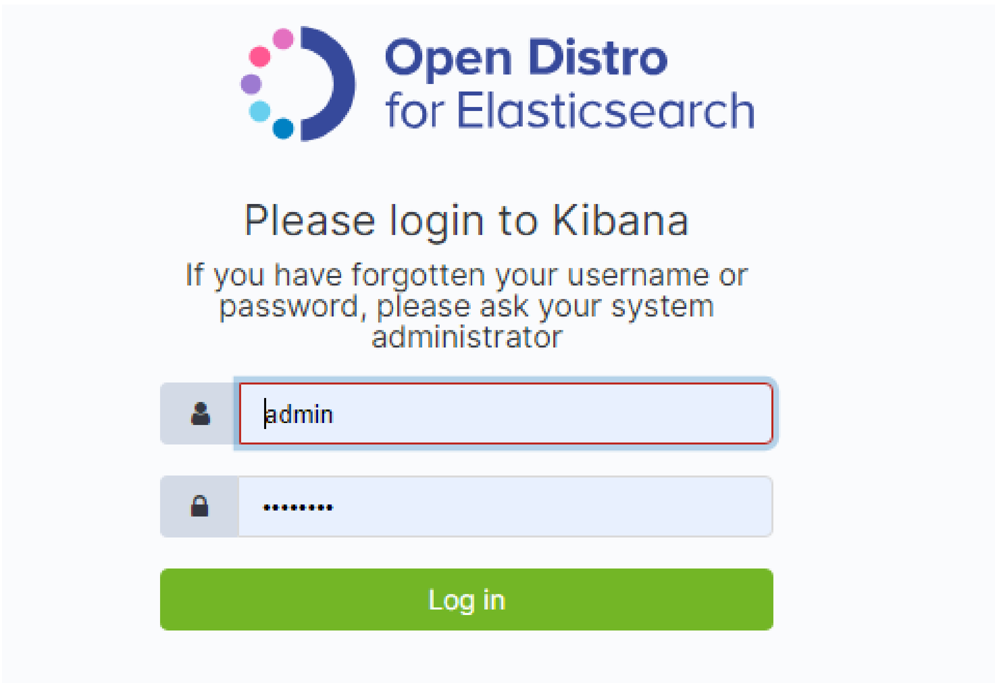
Step 1 Use Kibana to create a user.

1. Log in to the CSS management console.
2. Choose **Clusters** in the navigation pane. On the **Clusters** page, locate the target cluster and click **Access Kibana** in the **Operation** column.

Enter the administrator username and password to log in to Kibana.

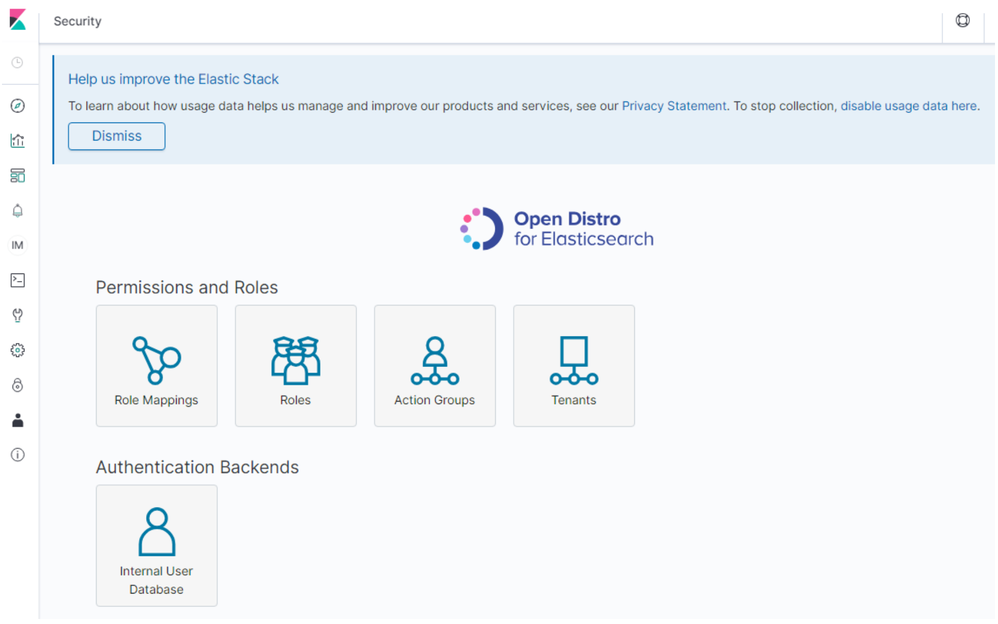
– Username: **admin** (default administrator account name)
– Password: Enter the administrator password you set when creating the cluster in security mode.

Figure 2-12 Login page


Open Distro for Elasticsearch
Please login to Kibana
If you have forgotten your username or password, please ask your system administrator
admin
••••••••
Log in

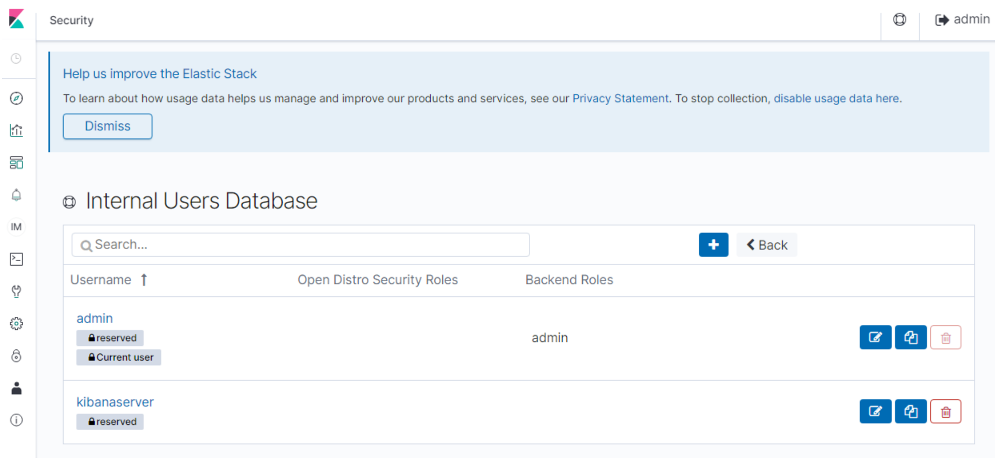
3. Click the **Security** icon on the Kibana operation page.

Figure 2-13 Security page


Security
Help us improve the Elastic Stack
To learn about how usage data helps us manage and improve our products and services, see our Privacy Statement. To stop collection, disable usage data here.
Dismiss
Open Distro for Elasticsearch
Permissions and Roles
Role Mappings
Roles
Action Groups
Tenants
Authentication Backends
Internal User Database

4. On the **Security** page, choose **Authentication Backends** > **Internal Users Database**.

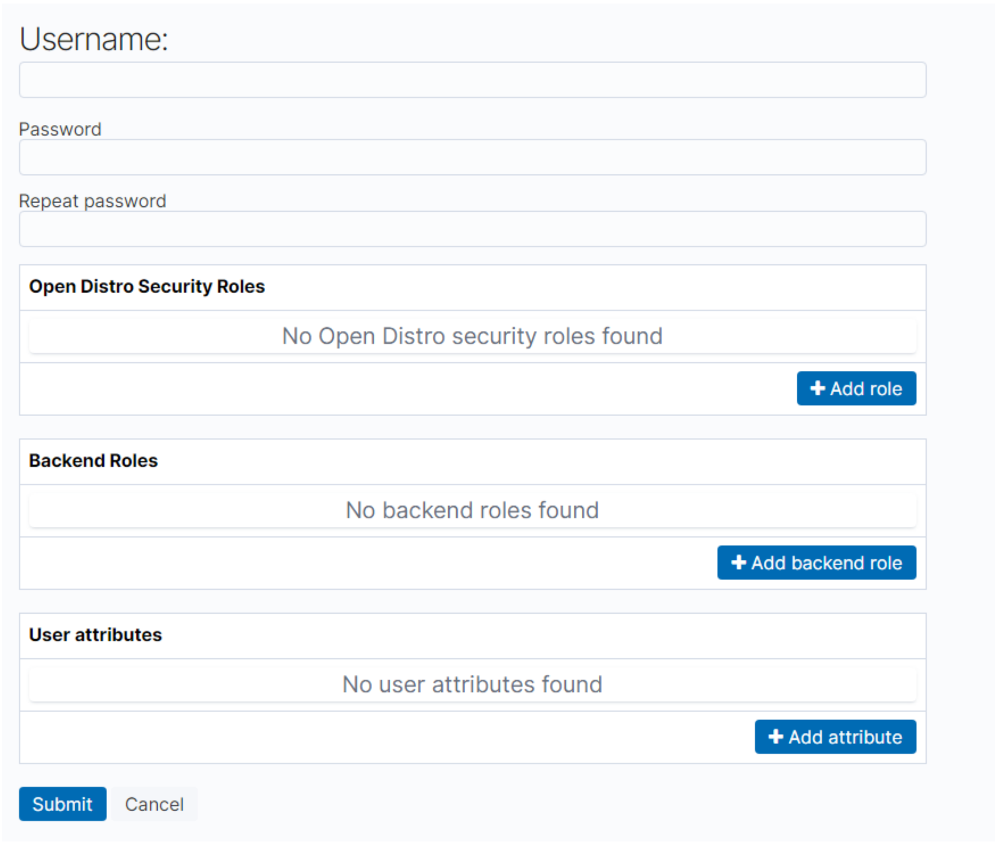
Figure 2-14 Creating a user


Security
admin
Help us improve the Elastic Stack
To learn about how usage data helps us manage and improve our products and services, see our Privacy Statement. To stop collection, disable usage data here.
Dismiss
Internal Users Database
Search...
Back

Username	Open Distro Security Roles	Backend Roles
admin reserved Current user		admin
kibanaserver reserved		

5. On the **Internal Users Database** page, select **+**.
6. On the user creation page, set the username and password, and click **Submit**. The username **test** is used as an example.

Figure 2-15 Adding user information



Username:

Password

Repeat password

Open Distro Security Roles

No Open Distro security roles found

Backend Roles

No backend roles found

User attributes

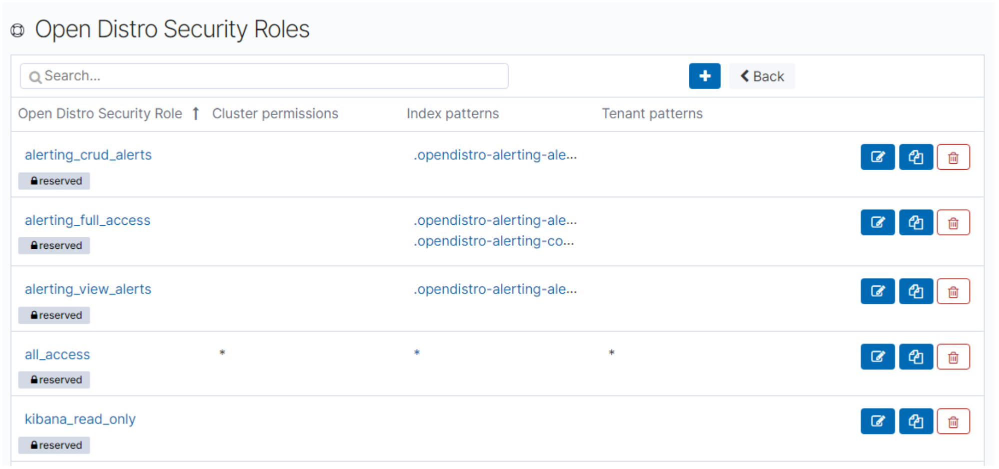
No user attributes found

The user will be displayed in the user list.

Step 2 Create a role and grant permissions to the role.

1. On the **Security** page, click **Roles**.

Figure 2-16 Open Distro Security Roles page


Open Distro Security Roles

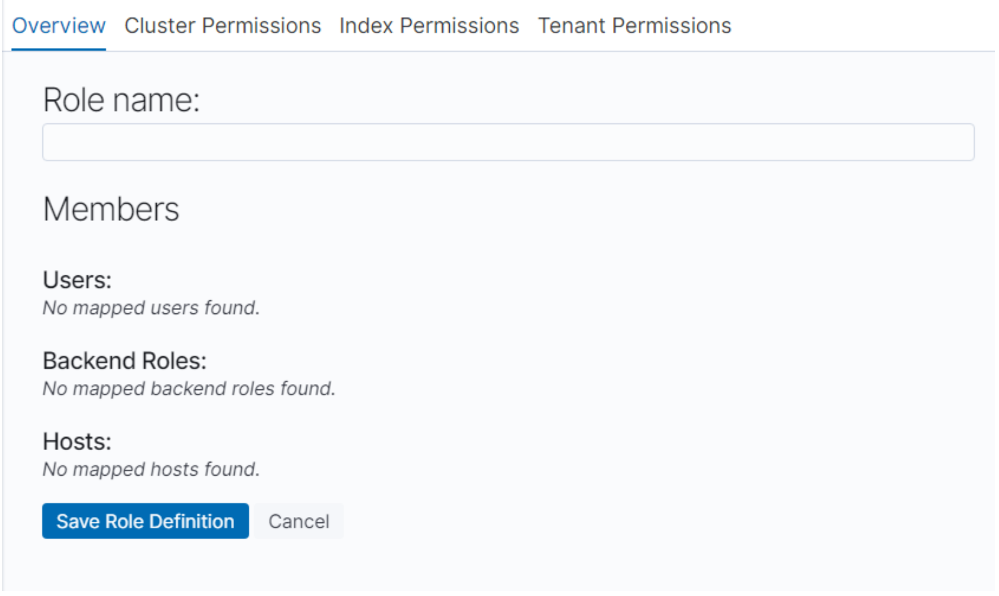
Search...

Back

Open Distro Security Role	Cluster permissions	Index patterns	Tenant patterns
alerting_crud_alerts reserved		.opendistro-alerting-ale...	
alerting_full_access reserved		.opendistro-alerting-ale... .opendistro-alerting-co...	
alerting_view_alerts reserved		.opendistro-alerting-ale...	
all_access reserved	*	*	*
kibana_read_only reserved			

2. On the **Open Distro Security Roles** page, click **+**.
3. Enter a role name on the **Overview** page.

Figure 2-17 Overview page


Overview Cluster Permissions Index Permissions Tenant Permissions

Role name:

Members

Users:
No mapped users found.

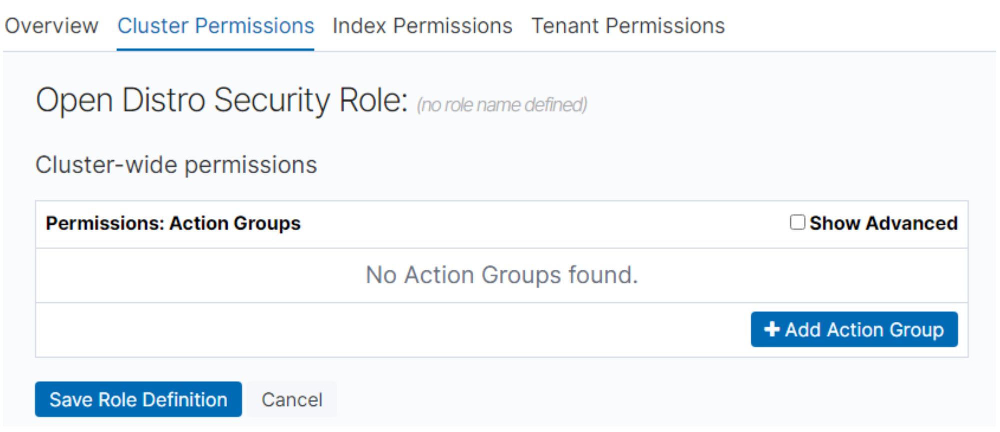
Backend Roles:
No mapped backend roles found.

Hosts:
No mapped hosts found.

Save Role Definition Cancel

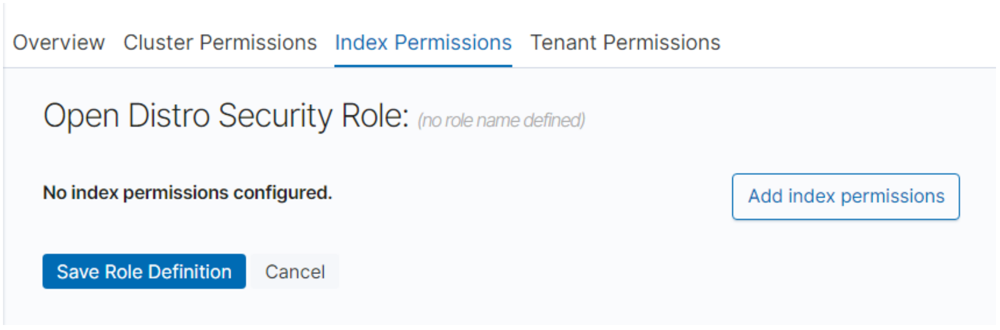
4. On the **Cluster Permissions** tab, configure CSS cluster permissions.

Figure 2-18 Cluster Permissions tab



5. On the **Index Permissions** tab, click **Add index permissions**.

Figure 2-19 Index Permissions tab



   - **Index patterns**: Set this parameter to the name of the index whose permission needs to be configured. For example, **my_store**.
   - Configure **Permissions: Action Groups** as required, for example, select the read-only permission **Search**.

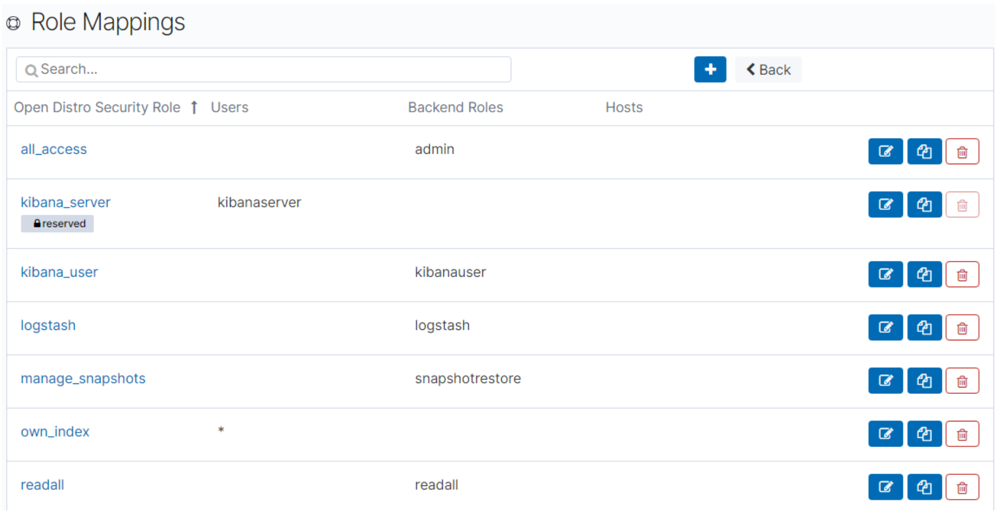
6. On the **Tenant Permissions** page, set role permissions.

   After the configuration is complete, the role will be displayed.

Step 3 Map a user with a role to bind them.

1. On the **Security** page, click **Role Mappings**.

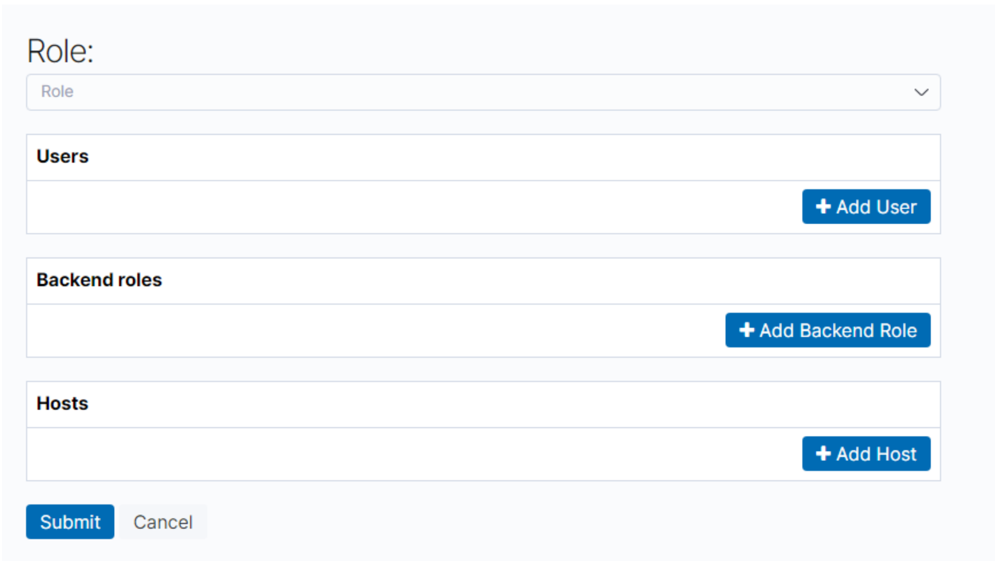
Figure 2-20 Role Mappings page


Role Mappings

Open Distro Security Role	Users	Backend Roles	Hosts
all_access		admin	
kibana_server reserved	kibanaserver		
kibana_user		kibanauser	
logstash		logstash	
manage_snapshots		snapshotrestore	
own_index	*		
readall		readall	

2. Click **+** to add the mapping between users and roles.

Figure 2-21 Adding user-role mappings



3. Click **Submit**.
4. Verify that the configuration takes effect in Kibana.

----End

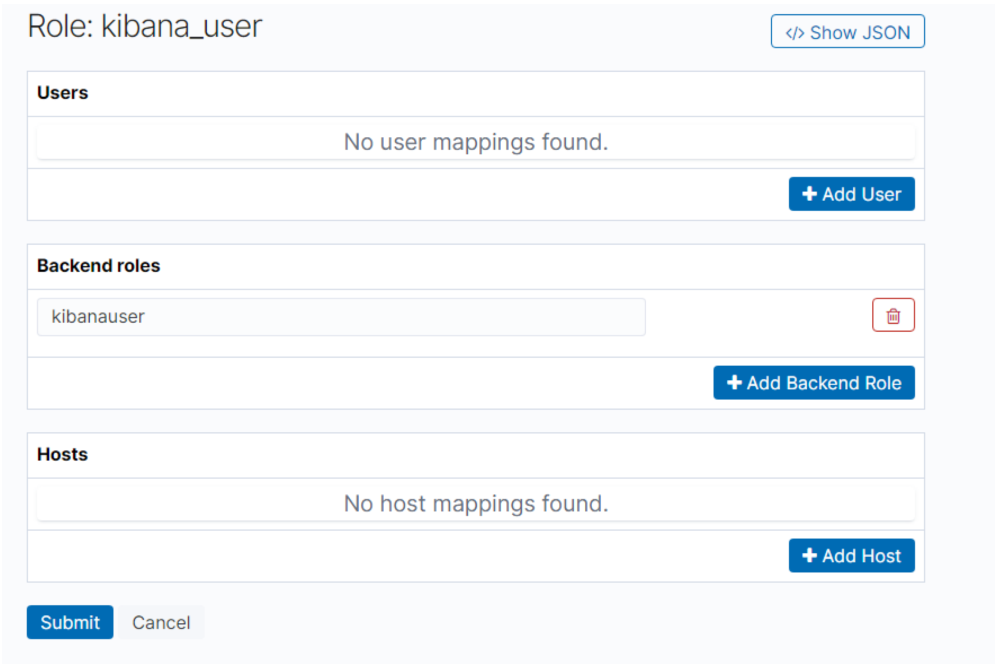
Creating a User Having the Kibana Access Permission

Step 1 Create a user named **test**. For details, see **Step 1**.

Step 2 Map a user with a role to bind them.

1. On the **Security** page, click **Role Mappings**.
2. On the **Role Mappings** page, click **kibana_user**.

Figure 2-22 kibana_user role



The **kibana_user** role has the permission for the **.kibana*** index. The dashboards and index patterns operated on the Kibana page are saved in **.kibana***. The **test** user is mapped to **kibana_user**, indicating that the **test** user has the Kibana permission.

3. Click **+** to add a user-role mapping.
4. In the **Users** area, select the **test** user.
5. Click **Submit**.

   After the configuration is complete, switch to the **test** user to check whether the permission takes effect.

----End

Granting a New User the Permission for the index*

The newly created **test** user can access Kibana and have permissions for the index patterns, Discover, and Dashboards of Kibana. However, this does not mean that the **test** user can view any **.kibana** space. By default, the **test** user can view only the data of its private tenant space and the global tenant space. To access other tenant spaces, you need to define other tenant permissions in the role.

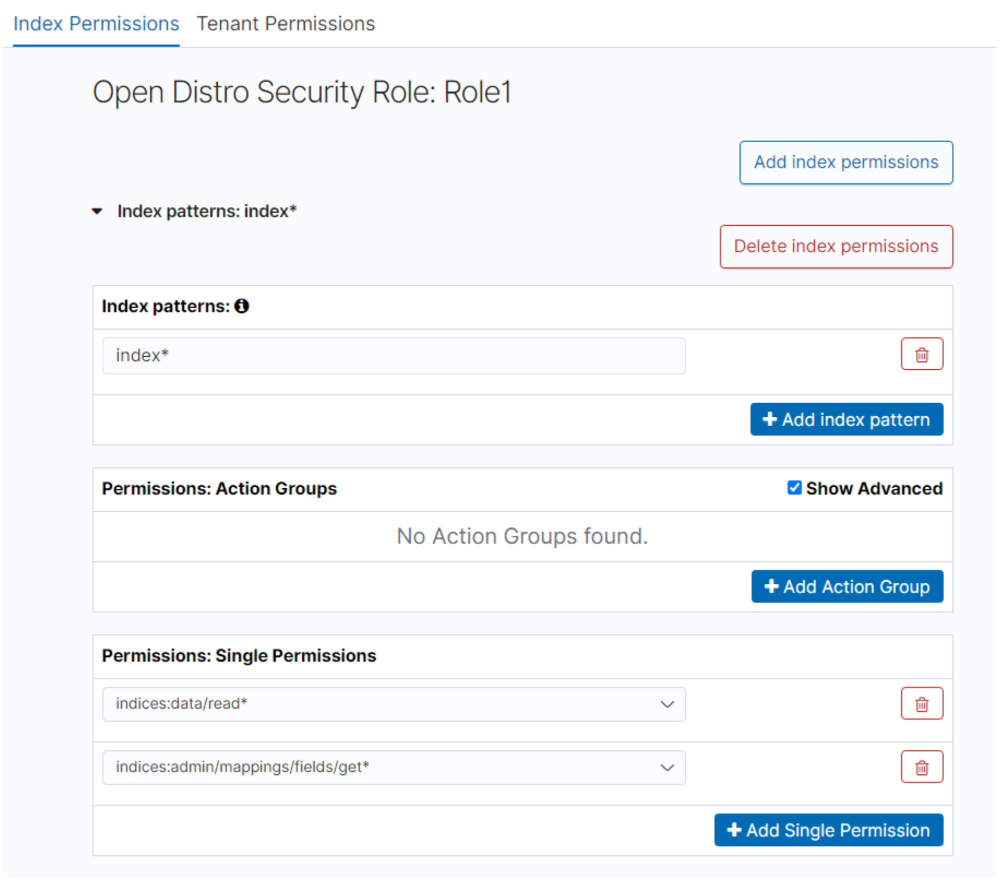
Step 1 Select **Roles** from the **Security** drop-down list box.

Step 2 On the **Open Distro Security Roles** page, click **+** to add role permissions.

Step 3 On the **Overview** tab, set the role name to **Role1**.

Step 4 On the **Index Permissions** tab, click **Add index permissions**.

Figure 2-23 Configuring permissions



Open Distro Security Role: Role1

Index patterns: index*

Index patterns:

index*

Permissions: Action Groups

No Action Groups found.

Permissions: Single Permissions

indices:data/read*

indices:admin/mappings/fields/get*

- **Index patterns**: Enter **index***.
- **Permissions: Action Groups**: Select permissions as needed. For a query permission, select **read**. For a write permission, select **write**. For details about the bottom-layer actions corresponding to operations, see the description of the permission module on the Kibana page. Take **read** as an example. Select **indices:data/read*** and **indices:admin/mappings/fields/get***. **indices:data/read*** contains all permissions in **indices:data/read/**, including **indices:data/read/get**, **indices:data/read/mget**, and **indices:data/read/search**.

Step 5 Complete the settings and check **Role1**.

Step 6 On the **Security** page, click **Role Mappings**.

Step 7 On the **Role Mappings** page, click **+** to add the mapping between the **test** user and the **Role1** role.

Step 8 Click **Submit**.

After the configuration is complete, the **test** user has the read permission on **index***.

----End

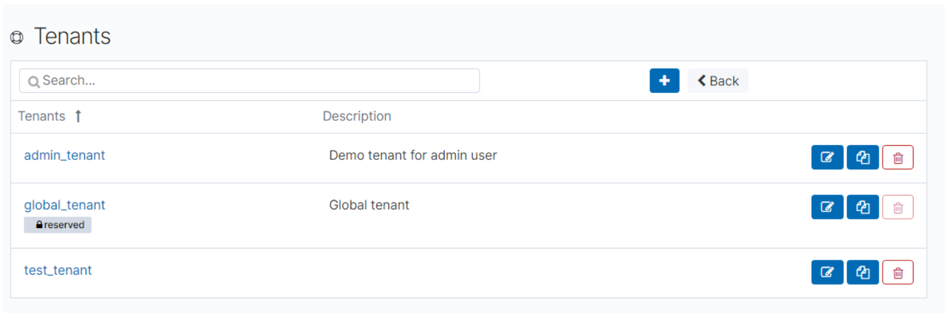
Sharing the Index Pattern and Dashboard Information of the admin Account with the test User

Generally, a new user does not have the permission to create an index pattern, and cannot manage data due to service relationship settings. In this case, the **admin** account creates an index pattern, manages dashboard and other report information, and shares the information with the **test** user.

Perform the following operations:

1. Create an index pattern and a dashboard in **global_tenant** as the **admin** user.
2. All tenants can directly access **global_tenant**. However, if there are too many users from different departments, the access performance may be poor. You can perform the following steps to improve performance:
   a. On the **Security** page, click **Tenants**. Create a tenant by department as the **admin** user, for example, **test_tenant**.

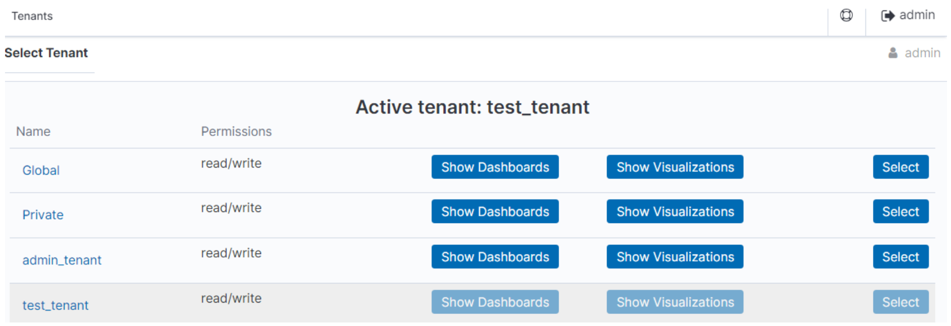
   Figure 2-24 Creating a tenant

Tenants	Description
admin_tenant	Demo tenant for admin user
global_tenant reserved	Global tenant
test_tenant	

   b. Switch to the **test_tenant** department.

   Figure 2-25 Switching to test_tenant

Name	Permissions			
Global	read/write	Show Dashboards	Show Visualizations	Select
Private	read/write	Show Dashboards	Show Visualizations	Select
admin_tenant	read/write	Show Dashboards	Show Visualizations	Select
test_tenant	read/write	Show Dashboards	Show Visualizations	Select

   c. Under **test_tenant**, create index patterns and dashboards as needed.
   d. On the **Security** page, click **Roles**. Click **Role1** corresponding to the **test** user. Assign **test_tenant** to **Role1** on the **Tenant Permissions** tab.

   Save the settings and switch to the **test** user. The **test** user can access the content in the **test_tenant** space.

3 Cluster Migration

3.1 Migration Solution Overview

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

Scenarios

The migration solution varies depending on the data source.

- Migration from an Elasticsearch cluster

  You can use Logstash, CDM, OBS backup and restoration, ESM, or cross-cluster replication plug-ins to migrate data in an Elasticsearch cluster.

  – Logstash: CSS provides Logstash to migrate data from different data sources and Elasticsearch, and clean and process data. For details, see **Using Logstash to Perform Full Data Migration**.

  – CDM: a cloud migration tool provided by Huawei Cloud to implement cluster migration between different cloud services. For details, see **Migrating the Entire Elasticsearch Database to CSS**.

  – Backup and restoration: Elasticsearch provides backup and restoration capabilities. You can back up the data of a cluster to OBS, and restore the data to another cluster. For details about how to migrate data between CSS Elasticsearch clusters, see **Migrating Data Through Backup and Restoration (from CSS Elasticsearch)**. For details about how to migrate data from a user-built Elasticsearch cluster or a third-party Elasticsearch cluster to a CSS Elasticsearch cluster, see **Migrating Data Through Backup and Restoration (from Third-Party Elasticsearch)**.

- **Migration from Kafka/MQ**
- **Migration from a Database**

Solutions

CSS supports migration by backup and restoration, by using the Reindex API or Logstash+ESM, or by data source synchronization. For details, see **Table 3-1**.

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

Table 3-1 Migration solutions

Solution	Description	Constraint	Performance
Backup and restoration	Prepare shared storage that supports the S3 protocol, for example, an OBS bucket. Create a snapshot to back up the data of the source Elasticsearch cluster, synchronize the snapshot to the target cluster, and restore data to the target cluster.	● Target Elasticsearch version ≥ Source Elasticsearch version ● Number of candidate master nodes of the target Elasticsearch cluster > Half of the number of candidate master nodes of the source Elasticsearch cluster ● Incremental data synchronization is not supported. You need to stop update before backing up or restoring data.	The data migration rate is configurable. Ideally, the data migration rate is the same as the file copy rate.
Reindex API	Configure mutual trust between the source and target Elasticsearch clusters, and then migrate data using the Reindex API.	● **_source** must be enabled for indexes. ● Real-time synchronization of incremental data is not supported. You need to stop the update and then call the API.	Batch read and write are supported, but concurrent slicing synchronization is not supported.

Solution	Description	Constraint	Performance
Logstash +ESM	Apply for an ECS, deploy and configure Logstash on it, and then start data migration.	• **_source** must be enabled for indexes. • Real-time synchronization of incremental data is not supported. You need to stop the update and then start Logstash.	Batch read and write are supported, and concurrent slicing synchronization is supported.
Data source synchronization	Inventory data is migrated using Logstash, and incremental data is automatically synchronized through traffic replication or data links.	None	The inventory migration rate is the same as that of Logstash. An existing tool is reused for incremental migration.

3.2 Migration from Elasticsearch

3.2.1 Using Logstash to Perform Full Data Migration

Logstash supports full data migration and incremental data migration. You can select full migration for the first time, and incremental migration for subsequent data migration. This section describes how to use Logstash of CSS to fully migrate cluster data.

Prepare for the migration by referring to **Restrictions** and **Preparations**. The procedure is as follows:

- **Step 1: Creating a Logstash Cluster**
- **Step 2: Verifying Cluster Connectivity**
- **Step 3: Configuring a Logstash Full Data Migration Task**
- **Step 4: Performing a Full Data Migration**
- **Step 5: Deleting the Logstash Cluster**

Restrictions

- Logstash version restrictions:

  CSS supports clusters of versions 5.5.1, 6.3.2, 6.5.4, 7.1.1, 7.6.2, and 7.10.2. Ensure that the major versions of the clusters whose data you want to migrate are the same.

  If the Elasticsearch cluster version is 5.*x*, select Logstash 5.6.16. If the Elasticsearch cluster version is 7.*x*, select Logstash 7.10.0.

- Do not modify indexes during cluster migration. Otherwise, the original data will be inconsistent with the migrated data.
- If there is less than 100 GB indexes, separate index analysis is not required.

Preparations

- Create a VM for data migration.
  a. Create a VM to migrate the metadata of the source cluster.
     i. Create a Linux ECS with 2 vCPUs and 4 GB memory.
     ii. Run the **curl http:// {*IP_address*}:{*port*}** command to test the connectivity between the VM and the source cluster and between the VM and the destination cluster.

        IP_address indicates the access address of the source and destination clusters. Enter the actual port number of the cluster. The default port is **9200**.

        NOTE

        The following example applies only to non-security clusters.

```
curl http://10.234.73.128:9200
{
  "name" : "voc_es_cluster_new-ess-esn-1-1",
  "cluster_name" : "voc_es_cluster_new",
  "cluster_uuid" : "1VbP7-39QNOx_R-llXKKtA",
  "version" : {
    "number" : "6.5.4",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "d2ef93d",
    "build_date" : "2018-12-17T21:17:40.758843Z",
    "build_snapshot" : false,
    "lucene_version" : "7.5.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "Tagline" : "You Know, for Search"
}
```

- Prepare the tools and software.

  The installation method is determined by whether the VM can be connected to the Internet. If VM can be connected to the Internet, use yum and pip to install the software. If VM cannot be connected to the Internet, download the installation package to the VM and run the installation commands.

Table 3-2 Tools and software

Type	Purpose	How to Obtain
Python2	Used to execute data migration scripts.	**Python2**. Select Python 2.7.18.
Winscp	Cross-platform file transfer tool. Upload scripts to Linux.	**Winscp**

The online installation procedure is as follows:

a. Run **yum install python2** to install python2.

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

b. Run **yum install python-pip** to install pip.

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

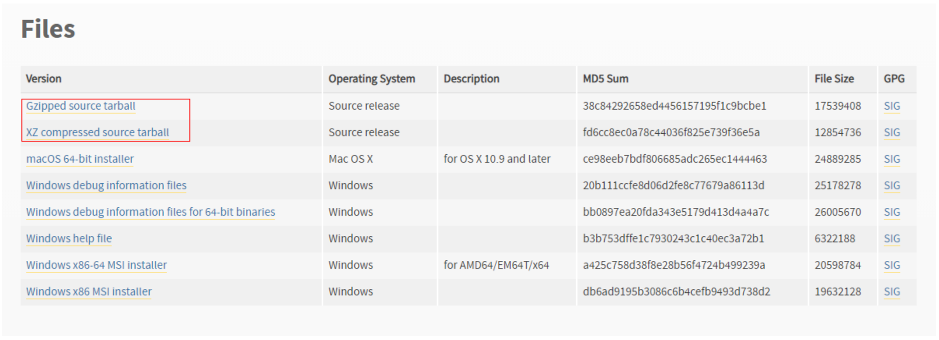
c. Run **pip install pyyaml** to install the YAML dependency.

d. Run **pip install requests** to install the requests dependency.

The online installation procedure is as follows:

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

Figure 3-1 Downloading the python2 package



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		38c84292658ed4456157195f1c9bcbe1	17539408	SIG
XZ compressed source tarball	Source release		fd6cc8ec0a78c44036f825e739f36e5a	12854736	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	ce98eeb7bdf806685adc265ec1444463	24889285	SIG
Windows debug information files	Windows		20b111ccfe8d06d2fe8c77679a86113d	25178278	SIG
Windows debug information files for 64-bit binaries	Windows		bb0897ea20fda343e5179d413d4a4a7c	26005670	SIG
Windows help file	Windows		b3b753dffe1c7930243c1c40ec3a72b1	6322188	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64	a425c758d38f8e28b56f4724b499239a	20598784	SIG
Windows x86 MSI installer	Windows		db6ad9195b3086c6b4cefb9493d738d2	19632128	SIG

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

```
# Decompress 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
...
# After the decompression, go to the 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
```

c. Check the Python installation result.

```
# 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)
[root@ecs-52bc Python-2.7.18]#
```

- Prepare the execution script.

a. Run the **vi migrateConfig.yaml** command to add configuration files.

```
es_cluster_new:
# Cluster name
  clustername: es_cluster_new
  # Address of the source Elasticsearch cluster, with http:// at the beginning
  src_ip: http://x.x.x.x:9200
  # If there is no username, set the password to "".
  src_username: ""
  src_password: ""
  # Address of the target Elasticsearch cluster, with http:// at the beginning
  dest_ip: http://x.x.x.x:9200
  # If there is no username, set the password to "".
  dest_username: ""
  dest_password: ""
  #This parameter is optional. The default value is false. It is used by migrateMapping.py.
  # Indicates whether to process only the mapping address indexes in the file.
  # If this parameter is set to true, only the indexes in the following mappings are obtained and
created in the target.
  # If this parameter is set to false, all the indexes of the source cluster are obtained, excluding
the following: .kibana, .*
  # Match the index names with the following mappings. If an index name matches a mapping,
use the value of the mapping as the index name in the target.
  # If no match is found, the original index name in the source is used.
  only_mapping: false
  # Index to be migrated. key indicates the index name in the source, and value indicates the
index name in the target.
  mapping:
      test_index_1: test_index_1

  # This parameter is optional. The default value is false. It is used by checkIndices.py.
  # If this parameter is set to false, all indexes and the number of documents are compared. If
this parameter is set to true, only the number of indexes is compared.
  only_compare_index: false
```

Configuration file parameters

Configuration	Description
clustername	Cluster name
src_ip	IP address for accessing the source cluster. Only one IP address is required. The default port number is 9200. If the port number for accessing the cluster is not 9200, use the actual port number.
src_username	Username for accessing the source cluster. If this parameter is not required, set it to "".
src_password	Password for accessing the source cluster. If this parameter is not required, set it to "".

Configuration	Description
dest_ip	IP address for accessing the target cluster. Only one IP address is required. The default port number is 9200. If the port number for accessing the cluster is not 9200, use the actual port number.
dest_username	Username for accessing the target cluster. If this parameter is not required, set it to "".
dest_password	Password for accessing the target cluster. If this parameter is not required, set it to "".

b. Run the **vi python migrateTemplate.py** command. Copy the following script to generate the index structure migration script:

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


def printDividingLine():
    print("<===========================================================>")


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)
    # config = yaml.load(config_file, Loader=yaml.FullLoader)
    return yaml.load(config_file)


def put_tempalte_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 main(argv):
    requests.packages.urllib3.disable_warnings()
    print("begin to migration template!")
    config = loadConfig(argv)
    src_clusters = config.keys()
    print("process cluster name:")
    for name in src_clusters:
        print(name)
    print("cluster total number:" + str(src_clusters.__len__()))

    for name, value in config.items():
        printDividingLine()
        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("*** get all template failed. resp statusCode:" + str(
                response.status_code) + " response is " + response.text)
            continue
        all_template = response.json()
        migrate_itemplate = []

        for template in all_template.keys():
            if template.startswith(".") or template == "logstash":
                continue
            if "index_patterns" in all_template[template]:
                for t in all_template[template]["index_patterns"]:
                    # if "kibana" in template:
                    if t.startswith("."):
                        continue
                    migrate_itemplate.append(template)

        for template in migrate_itemplate:
            dest_index_url = value["dest_ip"] + "/_template/" + template
            result = put_tempalte_to_target(dest_index_url, all_template[template], name,
template, dest_auth)
            if result is True:
                print('[success] delete success, cluster: %-10s, template %-10s ' % (str(name),
str(template)))
            else:
                print('[failure] delete failure, cluster: %-10s, template %-10s ' % (str(name),
str(template)))


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

c. Run the **vi migrateMapping.py** command. Copy the following script to generate the index structure migration script:

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


def printDividingLine():
    print("<===============================================================>")


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)
    # config = yaml.load(config_file, Loader=yaml.FullLoader)
    return yaml.load(config_file)


def get_cluster_version(url, auth=None):
    response = requests.get(url, auth=auth)
    if response.status_code != 200:
        print("*** get ElasticSearch message failed. resp statusCode:" + str(
            response.status_code) + " response is " + response.text)
        return False
    cluster = response.json()
    version = cluster["version"]["number"]

    return True


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"]

    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]
    # if index_mapping["settings"]["index"].has_key("lifecycle"):
    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)
        # index_mapping["settings"]["opendistro"]["index_state_management"]["rollover_alias"] =
lifecycle["rollover_alias"]
        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:
                    index_mapping["settings"]["index"]["analysis"]["filter"]["my_synonym_filter"][
                        "synonyms_path"] = "/rds/datastore/elasticsearch/v7.10.2/package/
elasticsearch-7.10.2/plugins/analysis-dynamic-synonym/config/synonyms.txt"
    return index_mapping


def getAlias(source, source_auth):
    # get all indices
    response = requests.get(source + "/_alias", auth=source_auth)
    if response.status_code != 200:
        print("*** 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 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(
            "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):
    requests.packages.urllib3.disable_warnings()
    print("begin to migrate index mapping!")
    config = loadConfig(argv)
    src_clusters = config.keys()

    print("begin to process cluster name :")
    for name in src_clusters:
        print(name)
    print("cluster count:" + str(src_clusters.__len__()))

    for name, value in config.items():
        printDividingLine()
        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)
        # only deal with mapping list
        if 'only_mapping' in value and value["only_mapping"]:
            for source_index, dest_index in value["mapping"].iteritems():
                print("start to process source index" + source_index + ", target index: " + dest_index)
                source_url = source + "/" + source_index
                response = requests.get(source_url, auth=source_auth)
                if response.status_code != 200:
                    print("*** get ElasticSearch message failed. resp statusCode:" + str(
                        response.status_code) + " response is " + response.text)
                    continue
                mapping = response.json()
                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("cluster name:" + name + ", " + source_index + ":failure")
                    continue
                print("cluster name:" + name + ", " + source_index + ":success")
        else:
            # get all indices
            system_index, create_index = getAlias(source, source_auth)
            success_index = 0
            for index in create_index:
                source_url = source + "/" + index
                index_response = requests.get(source_url, auth=source_auth)
                if index_response.status_code != 200:
                    print("*** get ElasticSearch message failed. resp statusCode:" + str(
                        index_response.status_code) + " response is " + index_response.text)
                    continue
                mapping = index_response.json()

                dest_index = index
                if 'mapping' in value:
                    if index in value["mapping"].keys():
                        dest_index = value["mapping"][index]
                index_mapping = process_mapping(mapping[index], dest_index)

                dest_url = dest + "/" + dest_index
                result = put_mapping_to_target(dest_url, index_mapping, name, index, dest_auth)
                if result is False:
                    print("[failure]: migrate mapping cluster name: " + name + ", " + index)
                    continue
                print("[success]: migrate mapping cluster name: " + name + ", " + 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 to generate the index data comparison script:

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


def printDividingLine():
    print("<===============================================================>")


def get_cluster_version(url, auth=None):
    response = requests.get(url, auth=auth)
    if response.status_code != 200:
        print("*** get ElasticSearch message failed. resp statusCode:" + str(
            response.status_code) + " response is " + response.text)
        return False
    cluster = response.json()
    version = cluster["version"]["number"]
    return True


# get all indices
```

```
def get_indices(url, source_auth):
    response = requests.get(url + "/_alias", auth=source_auth)
    if response.status_code != 200:
        print("*** 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_mapping(url, _auth, index):
    source_url = url + "/" + index
    index_response = requests.get(source_url, auth=_auth)
    if index_response.status_code != 200:
        print("*** get ElasticSearch message failed. resp statusCode:" + str(
            index_response.status_code) + " response is " + index_response.text)
        return "[failure] --- index is not exist in destination es. ---"
    mapping = index_response.json()
    return mapping


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("*** get ElasticSearch stats message failed. resp statusCode:" + str(
            index_response.status_code) + " response is " + index_response.text)
        return 0
    return index_response.json()


def get_indices_stats(url, es_auth):
    endpoint = url + "/_cat/indices"
    indicesResult = requests.get(endpoint, es_auth)
    indicesList = indicesResult.split("\n")
    indexList = []
    for indices in indicesList:
        indexList.append(indices.split()[2])
    return indexList


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)
    # python3
    # return yaml.load(config_file, Loader=yaml.FullLoader)
    return yaml.load(config_file)


def main(argv):
    requests.packages.urllib3.disable_warnings()
    print("begin to migrate index mapping!")
    config = loadConfig(argv)
    src_clusters = config.keys()

    print("begin to process cluster name :")
    for name in src_clusters:
        print(name)
    print("cluster count:" + str(src_clusters.__len__()))
```

```
for name, value in config.items():
    printDividingLine()
    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)

    if not os.path.exists("mappingLogs"):
        os.makedirs("mappingLogs")
    filename = "mappingLogs/" + str(cluster_name) + "#indices_stats"
    with open(filename + ".json", "w") as f:
        json.dump("cluster name: " + cluster_name, f)
        f.write("\n")
        json.dump("source indices: ", f)
        f.write("\n")
        json.dump(all_source_index, f, indent=4)
        f.write("\n")
        json.dump("destination indices : ", f)
        f.write("\n")
        json.dump(all_dest_index, f, indent=4)
        f.write("\n")

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

    filename_src = "mappingLogs/" + str(cluster_name) + "#indices_source_mapping"
    filename_dest = "mappingLogs/" + str(cluster_name) + "#indices_dest_mapping"
    with open(filename_src + ".json", "a") as f_src:
        json.dump("cluster name: " + cluster_name, f_src)
        f_src.write("\n")
    with open(filename_dest + ".json", "a") as f_dest:
        json.dump("cluster name: " + cluster_name, f_dest)
        f_dest.write("\n")
    for index in all_source_index:
        mapping = get_mapping(source, source_auth, index)
        with open(filename + ".json", "a") as f_src:
            json.dump("========================", f_src)
            f_src.write("\n")
            json.dump(mapping, f_src, indent=4)
            f_src.write("\n")
        with open(filename_src + ".json", "a") as f_src:
            json.dump("========================", f_src)
            f_src.write("\n")
            json.dump(mapping, f_src, indent=4)
            f_src.write("\n")

        mapping = get_mapping(dest, dest_auth, index)
        with open(filename + ".json", "a") as f_dest:
            json.dump("========================", f_dest)
            f_dest.write("\n")
            json.dump(mapping, f_dest, indent=4)
            f_dest.write("\n")
            with open(filename_dest + ".json", "a") as f_src:
                json.dump("========================", f_src)
                f_src.write("\n")
                json.dump(mapping, f_src, indent=4)
                f_src.write("\n")

        print("source indices write file success,      file: " + filename_src)
        print("destination indices write file success, file: " + filename_dest)

        if "only_compare_index" in value and value["only_compare_index"]:
            print("[success] only compare mapping, not compare index count.")
            continue

        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 is 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)
```

Step 1: Creating a Logstash Cluster

NOTE

- Logstash clusters are used to migrate data. By default, Logstash clusters are charged in pay-per-use mode. After data migration is complete, you are advised to delete the Logstash cluster to save costs.
- If there are multiple cluster indexes, you can create multiple Logstash clusters and configure different migration tasks for them.

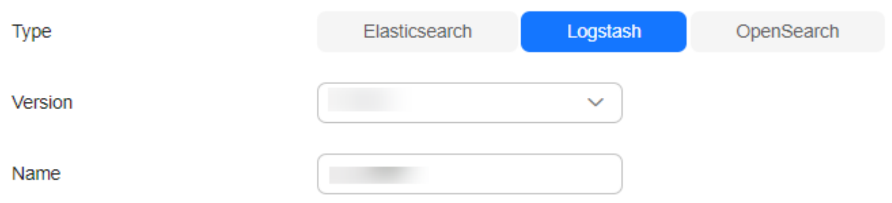
1. Log in to the CSS **management console**.
2. On the **Dashboard** or **Clusters** page, choose **Logstash** in the navigation pane on the left.
3. Click **Create Cluster**. The **Create Cluster** page is displayed.
4. Specify **Region** and **AZ**.
5. Specify the basic cluster information, select the cluster type and cluster version, and enter the cluster name.

Table 3-3 Basic parameters

Parameter	Description
Cluster Type	Select **Logstash**.

Parameter	Description
Version	**5.6.16** and **7.10.0** are supported. If the Elasticsearch cluster version is 5.*x*, select Logstash 5.6.16. If the Elasticsearch cluster version is 7.*x*, select Logstash 7.10.0.
Name	Cluster name, which contains 4 to 32 characters. Only letters, numbers, hyphens (-), and underscores (_) are allowed and the value must start with a letter.

Figure 3-2 Configuring basic information



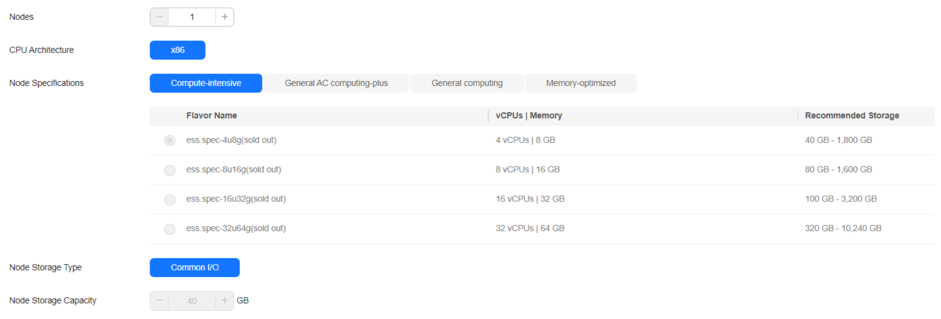
Type: Elasticsearch | [✓ Logstash] | OpenSearch

Version

Name

6. Set host specifications of the cluster. Set the number of **Nodes** to **1**. Set **Node Specifications** to 8 vCPUs | 16 GB and retain the default values for other parameters.

Figure 3-3 Configuring host specifications



Nodes: 1

CPU Architecture: [✓ x86]

Node Specifications: [✓ Compute-intensive] | General AC computing-plus | General computing | Memory-optimized

	Flavor Name	vCPUs | Memory	Recommended Storage
●	ess.spec-4u8g(sold out)	4 vCPUs | 8 GB	40 GB - 1,800 GB
○	ess.spec-8u16g(sold out)	8 vCPUs | 16 GB	80 GB - 1,600 GB
○	ess.spec-16u32g(sold out)	16 vCPUs | 32 GB	100 GB - 3,200 GB
○	ess.spec-32u64g(sold out)	32 vCPUs | 64 GB	320 GB - 10,240 GB

Node Storage Type: [✓ Common I/O]

Node Storage Capacity: 40 GB

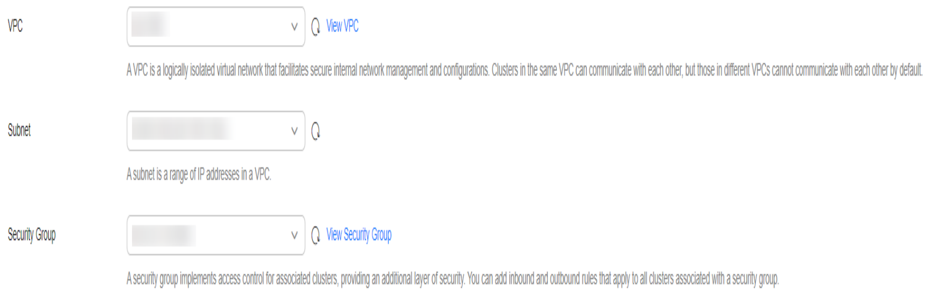
7. Set the enterprise project. Retain the default value.
8. Click **Next: Configure Network**. Configure the cluster network.

Table 3-4 Parameter description

Parameter	Description
VPC	A VPC is a secure, isolated, and logical network environment. Select the target VPC. Click **View VPC** to enter the VPC management console and view the created VPC names and IDs. If no VPCs are available, create one. **NOTE** The VPC must contain CIDRs. Otherwise, cluster creation will fail. By default, a VPC will contain CIDRs.

Parameter	Description
Subnet	A subnet provides dedicated network resources that are isolated from other networks, improving network security. Select the target subnet. You can access the VPC management console to view the names and IDs of the existing subnets in the VPC.
Security Group	A security group implements access control for ECSs that have the same security protection requirements in a VPC. To view more details about the security group, click **View Security Group**. **NOTE** Ensure that **Port Range/ICMP Type** is **Any** or a port range includes port **9200** for the selected security group.

Figure 3-4 Configuring network specifications



A VPC is a logically isolated virtual network that facilitates secure internal network management and configurations. Clusters in the same VPC can communicate with each other, but those in different VPCs cannot communicate with each other by default.

A subnet is a range of IP addresses in a VPC.

A security group implements access control for associated clusters, providing an additional layer of security. You can add inbound and outbound rules that apply to all clusters associated with a security group.

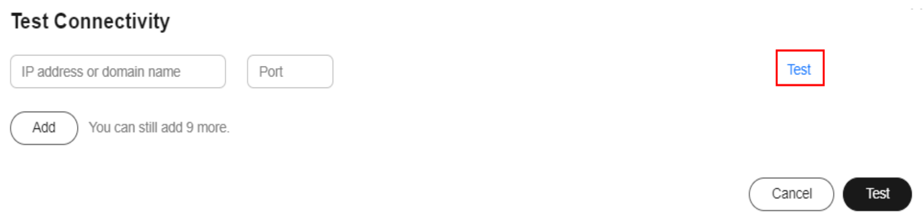
9. Click **Next: Configure Advanced Settings**. You can select **Default** or **Custom** for **Advanced Settings**. Retain the default settings in this example.
10. Click **Next: Confirm**. Check the configuration and click **Next** to create a cluster.
11. Click **Back to Cluster List** to switch to the **Clusters** page. The cluster you created is listed on the displayed page and its status is **Creating**. If the cluster is successfully created, its status will change to **Available**.

Step 2: Verifying Cluster Connectivity

Verify the connectivity between Logstash and the source and destination clusters.

1. On the Logstash clusters page, click the name of the Logstash cluster created in **Step 1: Creating a Logstash Cluster**. The **Cluster Information** page is displayed. Choose **Configuration Center** in the navigation pane on the left to go to the configuration center page. Alternatively, click **Configuration Center** in the **Operation** column of the target cluster to go to the configuration center page.
2. On the **Configuration Center** page, click **Test Connectivity**.
3. Enter the IP addresses or domain names and port numbers of the source and destination clusters, and click **Test**.

Figure 3-5 Testing the connectivity



Step 3: Configuring a Logstash Full Data Migration Task

1. On the Logstash clusters page, click the name of the Logstash cluster created in **Step 1: Creating a Logstash Cluster**. The **Cluster Information** page is displayed. Choose **Configuration Center**, or click **Configuration Center** in the **Operation** column of the target cluster. The **Configuration Center** page is displayed.
2. Click **Create** in the upper right corner. On the configuration file creation page that is displayed, select a cluster template and modify the migration configuration file of the Elasticsearch cluster.

   NOTE

   In this example, HTTPS is not enabled for the two Elasticsearch clusters.

   – Select a cluster template: In this example, data is imported from an Elasticsearch cluster to an Elasticsearch cluster. Locate the **elasticsearch** row and click **Apply** in the **Operation** column. Add configuration information based on different cluster configurations.
   – Modify the configuration file. Specify the configuration name, for example, es-es-all. Specify the migration configuration file of the Elasticsearch cluster. The following is an example of the configuration file:

```
input{
     elasticsearch{
        # IP address of the source cluster
        hosts =>  ["xx.xx.xx.xx:9200", "xx.xx.xx.xx:9200"]
        # Username and password for accessing the source cluster. Leave them blank if there is no
username or password.
        # user => "css_logstash"
        # password => "*****"
         # Information about the indexes to be migrated
        index => "*_202102"
        docinfo => true
        slices => 3
        size => 3000
    }
 }


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

 output{
     elasticsearch{
       # IP address of the destination cluster
       hosts => ["xx.xx.xx.xx:9200","xx.xx.xx.xx:9200"]
        # Username and password for accessing the destination cluster. Leave them blank if there
is no username or password.
       # user => "css_logstash"
```

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

Modify the following configurations:

Table 3-5 Modification of cluster configurations

Configuration		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. If there are no usernames, use the number sign (#) to comment out this item.
	password	Password for accessing the cluster. If there are no usernames or passwords, use the number sign (#) 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**.
	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	IP address for accessing the Huawei Cloud CSS cluster. If the cluster has multiple nodes, separate them with commas (,).
	user	Username for accessing the cluster. If there are no usernames, use the number sign (#) to comment out this item.
	password	Password for accessing the cluster. If there are no passwords, use the number sign (#) 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}*.

Configuration		Description
	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 recommended that the document ID be the same as that on the source end. If you need to automatically generate the document ID, use the number sign (#) to comment it out.

Step 4: Performing a Full Data Migration

Step 1 Use PuTTY to log in to the Linux VM created in **Preparations**.

Step 2 Run the **python migrateTemplate.py** command to migrate the index template.

Step 3 Run the **python migrateMapping.py** command to migrate indexes.

Step 4 On the Logstash clusters page, click the name of the Logstash cluster created in **Step 1: Creating a Logstash Cluster**. The **Cluster Information** page is displayed. Choose **Configuration Center**, or click **Configuration Center** in the **Operation** column of the target cluster. The **Configuration Center** page is displayed.

Step 5 Select the configuration file created in section **Step 3: Configuring a Logstash Full Data Migration Task** and click **Start** in the upper left corner.

Step 6 Determine whether to start data migration immediately once the Logstash service is started.

Step 7 If you enable Logstash, you can view the startup configuration file under the pipe.

Step 8 After the data migration is complete, use PuTTY to log in to the Linux VM and run the **python checkIndices.py** command to compare the data.

----End

Step 5: Deleting the Logstash Cluster

After the cluster migration is complete, delete the Logstash cluster.

1. Log in to the CSS management console.
2. Choose **Clusters** > **Logstash**. On the displayed page, locate the row that contains the target cluster and click **More** > **Delete** in the **Operation** column.
3. In the displayed dialog box, enter the name of the cluster to be deleted again and click **OK**.

3.2.2 Using Logstash to Perform Incremental Data Migration

Logstash supports full data migration and incremental data migration. You can select full migration for the first time, and incremental migration for subsequent data migration. This section describes how to use Logstash to incrementally migrate cluster data. Incremental migration requires indexes with timestamps to identify incremental data.

Prepare for the migration by referring to **Restrictions** and **Preparations**. The procedure is as follows:

- **Step 1: Creating a Logstash Cluster**
- **Step 2: Verifying Cluster Connectivity**
- **Step 3: Configuring a Logstash Incremental Data Migration Task**
- **Step 4: Performing an Incremental Data Migration**
- **Step 5: Deleting the Logstash Cluster**

Restrictions

- Logstash version restrictions:

  CSS supports clusters of versions 5.5.1, 6.3.2, 6.5.4, 7.1.1, 7.6.2, and 7.10.2. Ensure that the major versions of the clusters whose data you want to migrate are the same.

  If the Elasticsearch cluster version is 5.*x*, select Logstash 5.6.16. If the Elasticsearch cluster version is 7.*x*, select Logstash 7.10.0.
- Do not modify indexes during cluster migration. Otherwise, the original data will be inconsistent with the migrated data.
- If there is less than 100 GB indexes, separate index analysis is not required.

Preparations

- Create a VM for data migration.
  a. Create a VM to migrate the metadata of the source cluster.
     i. Create a Linux ECS with 2 vCPUs and 4 GB memory.
     ii. Run the **curl http:// {*IP_address*}:{*port*}** command to test the connectivity between the VM and the source cluster and between the VM and the destination cluster.

        IP_address indicates the access address of the source and destination clusters. Enter the actual port number of the cluster. The default port is **9200**.

        NOTE

        The following example applies only to non-security clusters.

```
curl http://10.234.73.128:9200
{
  "name" : "voc_es_cluster_new-ess-esn-1-1",
  "cluster_name" : "voc_es_cluster_new",
  "cluster_uuid" : "1VbP7-39QNOx_R-llXKKtA",
  "version" : {
    "number" : "6.5.4",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "d2ef93d",
    "build_date" : "2018-12-17T21:17:40.758843Z",
    "build_snapshot" : false,
    "lucene_version" : "7.5.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "Tagline" : "You Know, for Search"
}
```

- Prepare the tools and software.

  The installation method is determined by whether the VM can be connected to the Internet. If VM can be connected to the Internet, use yum and pip to install the software. If VM cannot be connected to the Internet, download the installation package to the VM and run the installation commands.

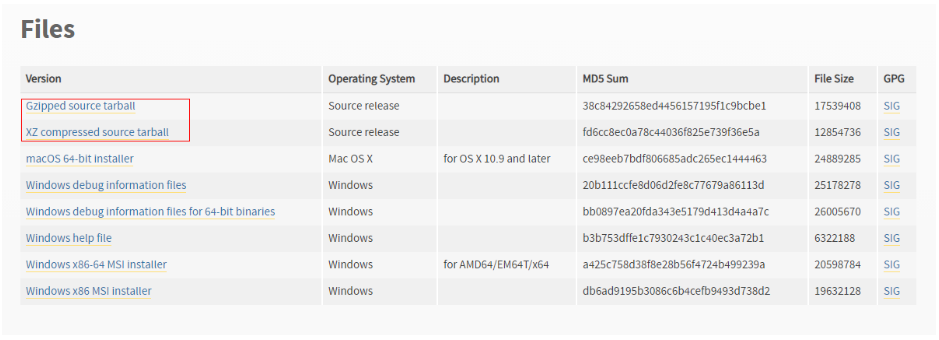
  The online installation procedure is as follows:

  a. Run **yum install python2** to install python2.
  ```
  [root@ecs opt]# yum install python2
  ```
  b. Run **yum install python-pip** to install pip.
  ```
  [root@ecs opt]# yum install python-pip
  ```
  c. Run **pip install pyyaml** to install the YAML dependency.
  d. Run **pip install requests** to install the requests dependency.

  The online installation procedure is as follows:

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

  Figure 3-6 Downloading the python2 package



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		38c84292658ed4456157195f1c9bcbe1	17539408	SIG
XZ compressed source tarball	Source release		fd6cc8ec0a78c44036f825e739f36e5a	12854736	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	ce98eeb7bdf806685adc265ec1444463	24889285	SIG
Windows debug information files	Windows		20b111ccfe8d06d2fe8c77679a86113d	25178278	SIG
Windows debug information files for 64-bit binaries	Windows		bb0897ea20fda343e5179d413d4a4a7c	26005670	SIG
Windows help file	Windows		b3b753dffe1c7930243c1c40ec3a72b1	6322188	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64	a425c758d38f8e28b56f4724b499239a	20598784	SIG
Windows x86 MSI installer	Windows		db6ad9195b3086c6b4cefb9493d738d2	19632128	SIG

  b. Use WinSCP to upload the Python installation package to the **opt** directory and install Python.
  ```
  # Decompress 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
  ...
  # After the decompression, go to the 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
```

c. Check the Python installation result.

```
# 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)
[root@ecs-52bc Python-2.7.18]#
```

- Prepare the execution script.

a. Run the **vi migrateConfig.yaml** command to add configuration files.

```
es_cluster_new:
# Cluster name
  clustername: es_cluster_new
  # Address of the source Elasticsearch cluster, with http:// at the beginning
  src_ip: http://x.x.x.x:9200
  # If there is no username, set the password to "".
  src_username: ""
  src_password: ""
  # Address of the target Elasticsearch cluster, with http:// at the beginning
  dest_ip: http://x.x.x.x:9200
  # If there is no username, set the password to "".
  dest_username: ""
  dest_password: ""
  #This parameter is optional. The default value is false. It is used by migrateMapping.py.
  # Indicates whether to process only the mapping address indexes in the file.
  # If this parameter is set to true, only the indexes in the following mappings are obtained and
created in the target.
  # If this parameter is set to false, all the indexes of the source cluster are obtained, excluding
the following: .kibana, .*
  # Match the index names with the following mappings. If an index name matches a mapping,
use the value of the mapping as the index name in the target.
  # If no match is found, the original index name in the source is used.
  only_mapping: false
  # Index to be migrated. key indicates the index name in the source, and value indicates the
index name in the target.
  mapping:
      test_index_1: test_index_1

  # This parameter is optional. The default value is false. It is used by checkIndices.py.
  # If this parameter is set to false, all indexes and the number of documents are compared. If
this parameter is set to true, only the number of indexes is compared.
  only_compare_index: false
```

Configuration file parameters

Configuration	Description
clustername	Cluster name
src_ip	IP address for accessing the source cluster. Only one IP address is required. The default port number is 9200. If the port number for accessing the cluster is not 9200, use the actual port number.
src_username	Username for accessing the source cluster. If this parameter is not required, set it to "".

Configuration	Description
src_password	Password for accessing the source cluster. If this parameter is not required, set it to "".
dest_ip	IP address for accessing the target cluster. Only one IP address is required. The default port number is 9200. If the port number for accessing the cluster is not 9200, use the actual port number.
dest_username	Username for accessing the target cluster. If this parameter is not required, set it to "".
dest_password	Password for accessing the target cluster. If this parameter is not required, set it to "".

b. Run the **vi checkIndices.py** command. Copy the following script to generate the index data comparison script:

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


def printDividingLine():
    print("<=============================================================>")


def get_cluster_version(url, auth=None):
    response = requests.get(url, auth=auth)
    if response.status_code != 200:
        print("*** get ElasticSearch message failed. resp statusCode:" + str(
            response.status_code) + " response is " + response.text)
        return False
    cluster = response.json()
    version = cluster["version"]["number"]
    return True


# get all indices
def get_indices(url, source_auth):
    response = requests.get(url + "/_alias", auth=source_auth)
    if response.status_code != 200:
        print("*** 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_mapping(url, _auth, index):
    source_url = url + "/" + index
    index_response = requests.get(source_url, auth=_auth)
    if index_response.status_code != 200:
        print("*** get ElasticSearch message failed. resp statusCode:" + str(
            index_response.status_code) + " response is " + index_response.text)
        return "[failure] --- index is not exist in destination es. ---"
    mapping = index_response.json()
    return mapping


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("*** get ElasticSearch stats message failed. resp statusCode:" + str(
            index_response.status_code) + " response is " + index_response.text)
        return 0
    return index_response.json()


def get_indices_stats(url, es_auth):
    endpoint = url + "/_cat/indices"
    indicesResult = requests.get(endpoint, es_auth)
    indicesList = indicesResult.split("\n")
    indexList = []
    for indices in indicesList:
        indexList.append(indices.split()[2])
    return indexList


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)
    # python3
    # return yaml.load(config_file, Loader=yaml.FullLoader)
    return yaml.load(config_file)


def main(argv):
    requests.packages.urllib3.disable_warnings()
    print("begin to migrate index mapping!")
    config = loadConfig(argv)
    src_clusters = config.keys()

    print("begin to process cluster name :")
    for name in src_clusters:
        print(name)
    print("cluster count:" + str(src_clusters.__len__()))

    for name, value in config.items():
        printDividingLine()
        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)

        if not os.path.exists("mappingLogs"):
            os.makedirs("mappingLogs")
        filename = "mappingLogs/" + str(cluster_name) + "#indices_stats"
        with open(filename + ".json", "w") as f:
            json.dump("cluster name: " + cluster_name, f)
            f.write("\n")
            json.dump("source indices: ", f)
            f.write("\n")
            json.dump(all_source_index, f, indent=4)
            f.write("\n")
            json.dump("destination indices : ", f)
            f.write("\n")
            json.dump(all_dest_index, f, indent=4)
            f.write("\n")

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

        filename_src = "mappingLogs/" + str(cluster_name) + "#indices_source_mapping"
        filename_dest = "mappingLogs/" + str(cluster_name) + "#indices_dest_mapping"
        with open(filename_src + ".json", "a") as f_src:
            json.dump("cluster name: " + cluster_name, f_src)
            f_src.write("\n")
        with open(filename_dest + ".json", "a") as f_dest:
            json.dump("cluster name: " + cluster_name, f_dest)
            f_dest.write("\n")
        for index in all_source_index:
            mapping = get_mapping(source, source_auth, index)
            with open(filename + ".json", "a") as f_src:
                json.dump("========================", f_src)
                f_src.write("\n")
                json.dump(mapping, f_src, indent=4)
                f_src.write("\n")
            with open(filename_src + ".json", "a") as f_src:
                json.dump("========================", f_src)
                f_src.write("\n")
                json.dump(mapping, f_src, indent=4)
                f_src.write("\n")

            mapping = get_mapping(dest, dest_auth, index)
            with open(filename + ".json", "a") as f_dest:
                json.dump("========================", f_dest)
                f_dest.write("\n")
                json.dump(mapping, f_dest, indent=4)
                f_dest.write("\n")
            with open(filename_dest + ".json", "a") as f_src:
                json.dump("========================", f_src)
                f_src.write("\n")
                json.dump(mapping, f_src, indent=4)
                f_src.write("\n")

        print("source indices write file success,      file: " + filename_src)
        print("destination indices write file success, file: " + filename_dest)

        if "only_compare_index" in value and value["only_compare_index"]:
            print("[success] only compare mapping, not compare index count.")
            continue

        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 is 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)
```

Step 1: Creating a Logstash Cluster

NOTE

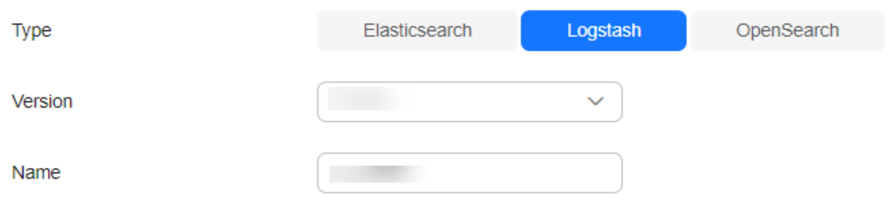
- Logstash clusters are used to migrate data. By default, Logstash clusters are charged in pay-per-use mode. After data migration is complete, you are advised to delete the Logstash cluster to save costs.
- If there are multiple cluster indexes, you can create multiple Logstash clusters and configure different migration tasks for them.

1. Log in to the CSS **management console**.
2. On the **Dashboard** or **Clusters** page, choose **Logstash** in the navigation pane on the left.
3. Click **Create Cluster**. The **Create Cluster** page is displayed.
4. Specify **Region** and **AZ**.
5. Specify the basic cluster information, select the cluster type and cluster version, and enter the cluster name.

Table 3-6 Basic parameters

Parameter	Description
Cluster Type	Select **Logstash**.
Version	**5.6.16** and **7.10.0** are supported. If the Elasticsearch cluster version is 5.*x*, select Logstash 5.6.16. If the Elasticsearch cluster version is 7.*x*, select Logstash 7.10.0.
Name	Cluster name, which contains 4 to 32 characters. Only letters, numbers, hyphens (-), and underscores (_) are allowed and the value must start with a letter.

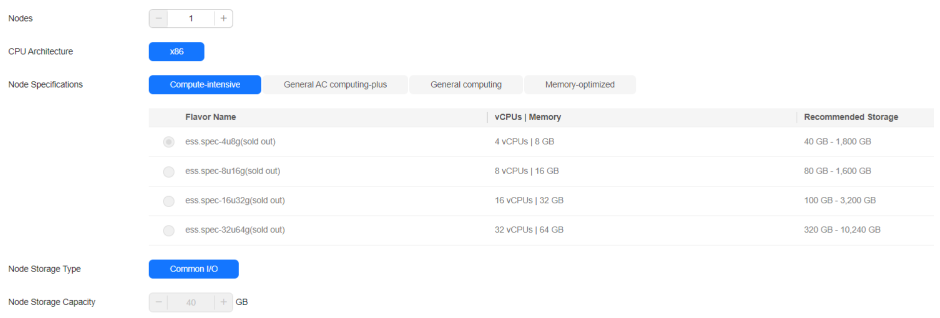
Figure 3-7 Configuring basic information



Type	Elasticsearch [✓ Logstash] OpenSearch
Version	[redacted]
Name	[redacted]

6. Set host specifications of the cluster. Set the number of **Nodes** to **1**. Set **Node Specifications** to 8 vCPUs | 16 GB and retain the default values for other parameters.

Figure 3-8 Configuring host specifications



Nodes	1
CPU Architecture	[✓ x86]
Node Specifications	[✓ Compute-intensive] General AC computing-plus General computing Memory-optimized

Flavor Name	vCPUs | Memory	Recommended Storage
ess.spec-4u8g(sold out)	4 vCPUs | 8 GB	40 GB - 1,800 GB
ess.spec-8u16g(sold out)	8 vCPUs | 16 GB	80 GB - 1,600 GB
ess.spec-16u32g(sold out)	16 vCPUs | 32 GB	100 GB - 3,200 GB
ess.spec-32u64g(sold out)	32 vCPUs | 64 GB	320 GB - 10,240 GB

Node Storage Type	[✓ Common I/O]
Node Storage Capacity	40 GB

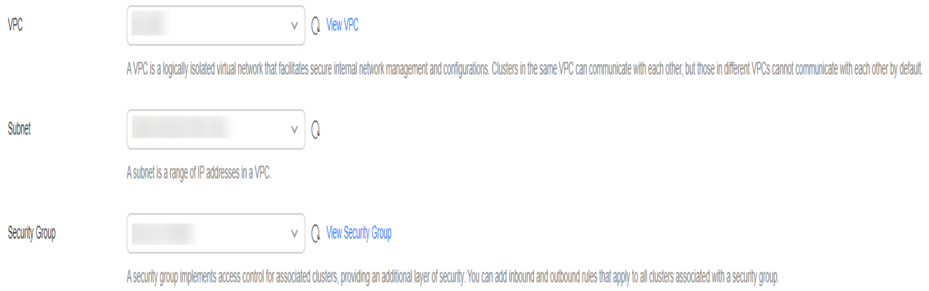
7. Set the enterprise project. Retain the default value.
8. Click **Next: Configure Network**. Configure the cluster network.

Table 3-7 Parameter description

Parameter	Description
VPC	A VPC is a secure, isolated, and logical network environment. Select the target VPC. Click **View VPC** to enter the VPC management console and view the created VPC names and IDs. If no VPCs are available, create one. **NOTE** The VPC must contain CIDRs. Otherwise, cluster creation will fail. By default, a VPC will contain CIDRs.
Subnet	A subnet provides dedicated network resources that are isolated from other networks, improving network security. Select the target subnet. You can access the VPC management console to view the names and IDs of the existing subnets in the VPC.

Parameter	Description
Security Group	A security group implements access control for ECSs that have the same security protection requirements in a VPC. To view more details about the security group, click **View Security Group**. **NOTE** Ensure that **Port Range/ICMP Type** is **Any** or a port range includes port **9200** for the selected security group.

Figure 3-9 Configuring network specifications



VPC

View VPC

A VPC is a logically isolated virtual network that facilitates secure internal network management and configurations. Clusters in the same VPC can communicate with each other, but those in different VPCs cannot communicate with each other by default.

Subnet

A subnet is a range of IP addresses in a VPC.

Security Group

View Security Group

A security group implements access control for associated clusters, providing an additional layer of security. You can add inbound and outbound rules that apply to all clusters associated with a security group.

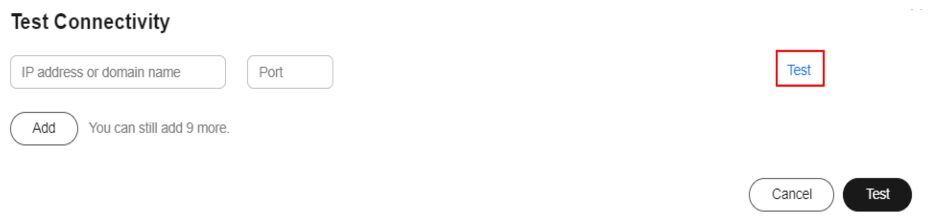
9. Click **Next: Configure Advanced Settings**. You can select **Default** or **Custom** for **Advanced Settings**. Retain the default settings in this example.
10. Click **Next: Confirm**. Check the configuration and click **Next** to create a cluster.
11. Click **Back to Cluster List** to switch to the **Clusters** page. The cluster you created is listed on the displayed page and its status is **Creating**. If the cluster is successfully created, its status will change to **Available**.

Step 2: Verifying Cluster Connectivity

Verify the connectivity between Logstash and the source and destination clusters.

1. On the Logstash clusters page, click the name of the Logstash cluster created in **Step 1: Creating a Logstash Cluster**. The **Cluster Information** page is displayed. Choose **Configuration Center** in the navigation pane on the left to go to the configuration center page. Alternatively, click **Configuration Center** in the **Operation** column of the target cluster to go to the configuration center page.
2. On the **Configuration Center** page, click **Test Connectivity**.
3. Enter the IP addresses or domain names and port numbers of the source and destination clusters, and click **Test**.

Figure 3-10 Testing the connectivity



Test Connectivity

IP address or domain name

Port

Test

Add

You can still add 9 more.

Cancel

Test

Step 3: Configuring a Logstash Incremental Data Migration Task

1. On the Logstash clusters page, click the name of the Logstash cluster created in **Step 1: Creating a Logstash Cluster**. The **Cluster Information** page is displayed. Choose **Configuration Center**, or click **Configuration Center** in the **Operation** column of the target cluster. The **Configuration Center** page is displayed.
2. Click **Create** in the upper right corner. On the configuration file creation page that is displayed, select a cluster template and modify the migration configuration file of the Elasticsearch cluster.

   In this example, HTTPS is not enabled for the two Elasticsearch clusters.

   – Select a cluster template: In this example, data is imported from an Elasticsearch cluster to an Elasticsearch cluster. Locate the **elasticsearch** row and click **Apply** in the **Operation** column. Add cluster configurations as required.
   – Modify the configuration file. Specify the configuration name, for example, es-es-inc. Specify the migration configuration file of the Elasticsearch cluster. The following is an example of the configuration file:

```
input{
     elasticsearch{
         hosts =>  ["xx.xx.xx.xx:9200"]
         user => "css_logstash"
         password => "******"
         index => "*_202102"
         query => '{"query":{"bool":{"should":[{"range":{"postsDate":{"from":"2021-05-25
00:00:00"}}}]}}}'
         docinfo => true
         size => 1000
         #scroll => "5m"

     }
 }

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

 output{
     elasticsearch{
       hosts => ["xx.xx.xx.xx:9200","xx.xx.xx.xx:9200"]
         user => "admin"
         password => "******"
         index => "%{[@metadata][_index]}"
         document_type => "%{[@metadata][_type]}"
         document_id => "%{[@metadata][_id]}"
     }

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

   Modify the following configurations:

Table 3-8 Modification of cluster configurations

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. If there are no usernames, use the number sign (#) to comment out this item.
password	Password for accessing the cluster. If there are no usernames or passwords, use the number sign (#) 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. Generally, it is the 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.

NOTE

The incremental migration configuration varies according to the index and must be provided based on the index analysis.

Step 4: Performing an Incremental Data Migration

Step 1 Use PuTTY to log in to the Linux VM created in **Preparations**.

Step 2 On the Logstash clusters page, click the name of the Logstash cluster created in **Step 1: Creating a Logstash Cluster**. The **Cluster Information** page is displayed. Choose **Configuration Center**, or click **Configuration Center** in the **Operation** column of the target cluster. The **Configuration Center** page is displayed.

Step 3 Select the configuration file created in section **Step 3: Configuring a Logstash Incremental Data Migration Task** and click **Start** in the upper left corner.

Step 4 Start data migration immediately once the Logstash service is started as prompted .

Step 5 You can view the startup configuration file under the pipe.

Step 6 After the data migration is complete, use PuTTY to log in to the Linux VM and run the **python checkIndices.py** command to compare the data.

----End

Step 5: Deleting the Logstash Cluster

After the cluster migration is complete, delete the Logstash cluster.

1. Log in to the CSS management console.
2. Choose **Clusters** > **Logstash**. On the displayed page, locate the row that contains the target cluster and click **More** > **Delete** in the **Operation** column.
3. In the displayed dialog box, enter the name of the cluster to be deleted again and click **OK**.

3.2.3 Migrating Data Through Backup and Restoration (from CSS Elasticsearch)

Overview

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

Application scenarios:

- Cluster upgrade: Migrate data from a cluster of an earlier version to a cluster of a later version.
- Cluster merge: Merge the index data of two clusters.

This section describes how to take a snapshot of a cluster and restore it to another cluster. Take Elasticsearch cluster **Es-1** and **Es-2** as an example.

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 (s) = (800 GB/40 MB/Number of source cluster nodes + 800 GB/40 MB/Number of destination cluster nodes) x Number of indexes

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

  Total duration (s) = (800 GB/40 MB/Number of shards of the source cluster index + 800 GB/40 MB/Number of shards of the destination cluster index) 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.
- Ensure that the version of the destination cluster (**Es-2**) is later than or same as that of the source cluster (**Es-1**).
- Ensure that the number of nodes in the destination cluster (**Es-2**) is greater than half of the number of nodes in the source cluster (**Es-1**).
- Ensure that the number of nodes in the destination cluster (**Es-2**) is greater than or equal to the number of shards in the source cluster (**Es-1**).
- Ensure that the CPU, memory, and disk configurations of the target cluster (**Es-2**) are greater than or equal to those of the source cluster (**Es-1**).

Procedure

1. Log in to the Cloud Search Service management console.
2. Choose **Clusters** > **Elasticsearch**. On the displayed page, click the source cluster name **Es-1** to go to the basic information page.
3. In the navigation pane, choose **Cluster Snapshots**, and set basic snapshot configurations.

Table 3-9 Basic configurations for a cluster snapshot

Parameter	Description
OBS Bucket	Select an OBS bucket for storing cluster snapshots.
Backup Path	Storage path of the cluster snapshot in the OBS bucket. You can retain the default value.
IAM Agency	Select an IAM agency to authorize CSS to access or maintain data stored in OBS. The IAM agency must have the **OBS Administrator** permission for project **OBS** in region **Global service**.

4. Click **Create**. In the dialog box that is displayed, configure the parameters and click **OK** to manually create a snapshot.

Table 3-10 Snapshot creation parameters

Parameter	Description
Snapshot Name	User-defined snapshot name. You can retain the default value.
Index	Enter the name of the index to be backed up. Use commas (,) to separate multiple indexes. Uppercase letters, spaces, and the following special characters are not allowed: "\<|>/? If you do not specify this parameter, data of all indexes in the cluster is backed up by default. You can use the asterisk (*****) to back up data of certain indexes. For example, if you enter **index***, then data of indexes with the name prefix of **index** will be backed up.
Description	Snapshot description.

In the snapshot management list, if the snapshot status is **Available**, the snapshot has been created.

5. In the snapshot management list, click **Restore** in the **Operation** column of the snapshot and configure restoration parameters to restore data to destination cluster **Es-2**.

Table 3-11 Snapshot restoration parameters

Parameter	Description
Index	Enter the name of the index to be restored. If this parameter is not specified, all index data will be restored. You can use the asterisk (*) to match multiple indexes. For example, **index*** indicates that all indexes with the prefix **index** in snapshots are restored.
Rename Pattern	Index name matching rule. The **Rename Pattern** and **Rename Replacement** take effect only when they are configured at the same time. You can configure them to rename matched indexes in snapshots.
Rename Replacement	Rule for renaming an index name. The **Rename Pattern** and **Rename Replacement** take effect only when they are configured at the same time. The default value **restored_index_$1** indicates that **restored_** is added in front of the names of all restored indexes.
Cluster	Select a destination cluster, for example, **Es-2**. **NOTICE** If the source and destination clusters have indexes with the same names, the indexes in the destination cluster will be overwritten by those in the source cluster after the restoration.

In the snapshot management list, if **Task Status** changes to **Restoration succeeded**, data in source cluster **Es-1** is successfully migrated to destination cluster **Es-2**.

3.2.4 Migrating Data Through Backup and Restoration (from Third-Party Elasticsearch)

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

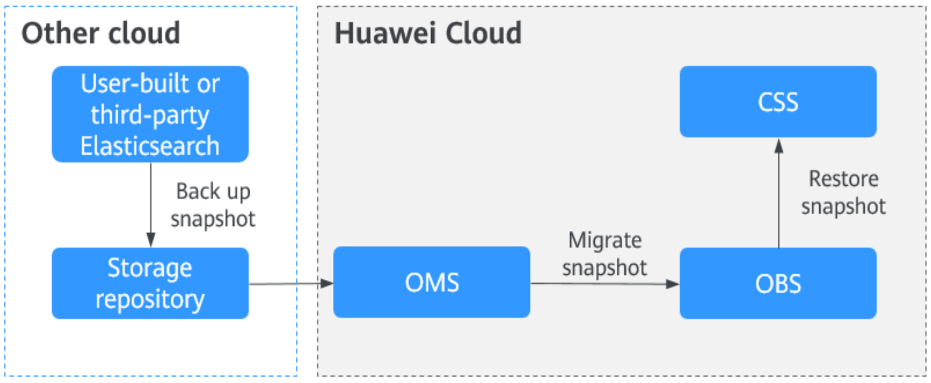
Prerequisites

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

Migration Process

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

Figure 3-11 Migration through backup and restoration



Procedure

Step 1 Log in to the third-party cloud where Elasticsearch is located and create a shared repository that supports the S3 protocol.

Step 2 Create a snapshot backup repository in the user-built or third-party Elasticsearch cluster to store Elasticsearch snapshot data.

For example, create a backup repository named **my_backup** at Elasticsearch and associate it with the repository OSS.

```
PUT _snapshot/my_backup
   {
      # Repository type.
   "type": "oss",
      "settings": {
      # # Private network domain name of the repository in step 1.
      "endpoint": "http://oss-xxx.xxx.com",
      # User ID and password of the repository. 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 for the user-built or third-party Elasticsearch cluster.

- Create a snapshot for all indexes.

  For example, create a snapshot named **snapshot_1**.

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

- Create a snapshot for specified indexes.

  For example, create a snapshot named **snapshot_test** that contains indexes **patent_analyse** and **patent**.

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

Step 4 View the snapshot creation progress of the cluster.

- Run the following command to view information about all snapshots:
  ```
  GET _snapshot/my_backup/_all
  ```
- Run the following command to view information about **snapshot_1**:
  ```
  GET _snapshot/my_backup/snapshot_1
  ```

Step 5 Migrate snapshot data from the repository to OBS.

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

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

For example, create a repository named **my_backup_all** in the cluster and associate it with the destination OBS.

```
PUT _snapshot/my_backup_all/
{
   "type" : "obs",
   "settings" : {
      # Private network domain name of OBS
      "endpoint" : "obs.xxx.xxx.com",
```

```
        "region" : "xxx",
        # Username and password for accessing OBS. 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 7 Restore the snapshot data to the Elasticsearch cluster of CSS.

1. Check information about all snapshots.
   ```
   GET _snapshot
   ```
2. Restore a snapshot
   – Restore all the indexes from a snapshot. For example, to restore all the indexes from **snapshot_1**, run the following command:
     ```
     POST _snapshot/my_backup_all/snapshot_1/_restore?wait_for_completion=true
     ```
   – Restores some indexes from a snapshot. For example, in the snapshot named **snapshot_1**, restore only the indexes that do not start with a period (.).
     ```
     POST _snapshot/my_backup/snapshot_1/_restore
     {"indices":"*,-.monitoring*,-.security*,-.kibana*","ignore_unavailable":"true"}
     ```
   – Restore a specified index from a snapshot and renames the index. For example, in **snapshot_1**, restore **index_1** to **restored_index_1** and **index_2** to **restored_index_2**.
     ```
     POST /_snapshot/my_backup/snapshot_1/_restore
     {
         # Restore only indexes index_1 and index_2 and ignore other indexes in the snapshot.
         "indices": "index_1,index_2"
         # Search for the index that is being restored. The index name must match the provided 
     template.
         "rename_pattern": "index_(.+)",
         # Rename the found index.
         "rename_replacement": "restored_index_$1"
     }
     ```

Step 8 View the snapshot restoration result.

- Run the following command to view the restoration results of all snapshots:
  ```
  GET /_recovery/
  ```
- Run the following command to check the snapshot restoration result of a specified index:
  ```
   GET {index_name}/_recovery
  ```

----End

3.3 Migration from Kafka/MQ

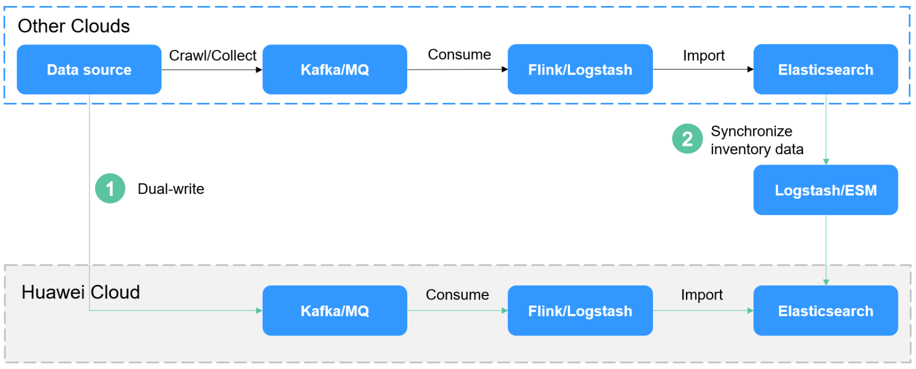
Process

In industries dealing with a large amount of data, such as IoT, news, public opinion analysis, and social networking, message middleware such as Kafka and

MQ is used to balance traffic in peak and off-peak hours. The tools such as Flink and Logstash are then used to consume data, preprocess data, and import data to the search engine, providing the search service for external systems.

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

Figure 3-12 Migration from a Kafka or MQ cluster



This migration solution is convenient and flexible.

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

Procedure

Step 1 Subscribe to incremental data. Create a consumer group in Kafka or MQ, and subscribe to incremental data.

Step 2 Synchronize inventory data. Use a tool such as Logstash to migrate data from the source Elasticsearch cluster to the CSS cluster. If Logstash is used for data migration, see **Deploying Logstash and Migrating Cluster Data**.

Step 3 Synchronize incremental data. After the inventory data is synchronized, enable the incremental consumer group. Based on the idempotence of Elasticsearch operations on data, when the new consumer group catches up with the previous consumer group, the data on both sides will be consistent.

NOTE

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

----End

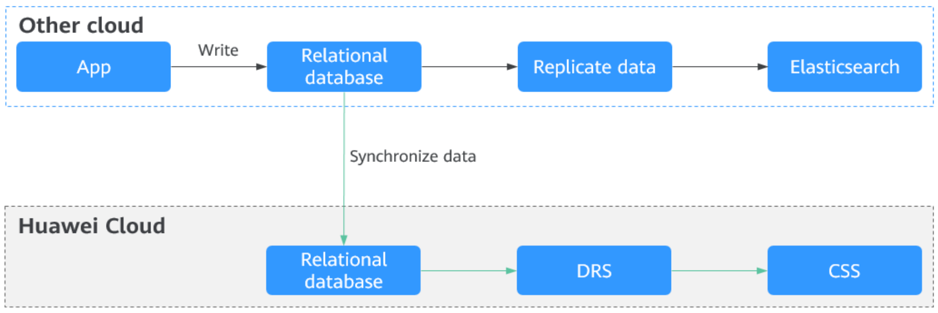
3.4 Migration from a Database

Process

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

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

Figure 3-13 Migration from a database



This migration solution is convenient and flexible.

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

Migration Guide

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

Table 3-12 provides guidance for migrating source databases to CSS.

Table 3-12 Data synchronization guide

Source DB Type	Destination DB Type	Synchronization Mode	Related Documents
RDS for MySQL	CSS/ES	Full + incremental synchronization	**From MySQL to CSS/ES**

Source DB Type	Destination DB Type	Synchronization Mode	Related Documents
• Local user-built MySQL databases • User-built MySQL databases on ECS	CSS/ES	Full + incremental synchronization	**From MySQL to CSS/ES**
GaussDB(for MySQL)	CSS/ES	Full + incremental synchronization	**From GaussDB(for MySQL) to CSS/ES**

4 Cluster Access

4.1 Overview

Elasticsearch clusters support multiple connection modes. You can determine how to access an Elasticsearch cluster based on the programming language used for your services. For more information about the clients used for CSS clusters in different security modes (non-security mode, security mode+HTTP, and security mode+HTTPS), see **Table 4-1**.

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

Table 4-1 Support for access from different clients

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

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

4.2 Accessing an Elasticsearch Cluster

Elasticsearch clusters have built-in Kibana and Cerebro components. You can quickly access an Elasticsearch cluster through Kibana and Cerebro. For details about how to access the Elasticsearch clusters using cURL commands, Java clients, or Python clients, see **Access Cluster**.

Access a Cluster Through Kibana

1. Log in to the CSS management console.
2. On the **Clusters** page, locate the target cluster and click **Access Kibana** in the **Operation** column to go to the Kibana login page.
   – Non-security cluster: The Kibana console is displayed.
   – Security cluster: Enter the username and password on the login page and click **Log In** to go to the Kibana console. The default username is **admin** and the password is the one specified during cluster creation.
3. After the login is successful, you can access the Elasticsearch cluster through Kibana.

Accessing a Cluster Through Cerebro

1. Log in to the CSS management console.
2. On the **Clusters** page, locate the target cluster and click **More** > **Cerebro** in the **Operation** column to go to the Cerebro login page.
   - Non-security cluster: Click the cluster name on the Cerebro login page to go to the Cerebro console.
   - Security cluster: Click the cluster name on the Cerebro login page, enter the username and password, and click **Authenticate** to go to the Cerebro console. The default username is **admin** and the password is the one specified during cluster creation.
3. After the login is successful, you can access the Elasticsearch cluster through Cerebro.

4.3 Accessing a Cluster Using cURL Commands

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

Prerequisites

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

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

  For details about how to use the ECS, see **ECS Hands-On Tutorials**.

Procedure

1. Obtain the private network address of the cluster. It is used to access the cluster.
   a. In the navigation pane on the left, choose **Clusters**.
   b. In the cluster list, select a cluster, and obtain and record its **Private Network Address**. Format: *<host>***:***<port>* or *<host>***:***<port>***,***<host>***:***<port>*

      If the cluster has only one node, the IP address and port number of only one node are displayed, for example, **10.62.179.32:9200**. If the cluster has multiple nodes, the IP addresses and port numbers of all nodes are displayed, for example, **10.62.179.32:9200,10.62.179.33:9200**.
2. Run one of the following commands on the ECS to access the cluster. The access command varies according to the security mode of the cluster.

– Cluster in non-security mode
```
curl "http://<host>:<port>"
```
– Cluster in security mode + HTTP
```
curl -u <user>:<password> "http://<host>:<port>"
```
– Cluster in security mode + HTTPS
```
curl -u <user>:<password> -k "https://<host>:<port>"
```

Table 4-2 Variables

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

An access example is as follows:
```
curl "http://10.62.176.32:9200"
```
Information similar to the following is displayed:
```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8
content-length: 513

{
   "name" : "xxx-1",
   "cluster_name" : "xxx",
   "cluster_uuid" : "xxx_uuid",
   "version" : {
      "number" : "7.10.2",
      "build_flavor" : "oss",
      "build_type" : "tar",
      "build_hash" : "unknown",
      "build_date" : "unknown",
      "build_snapshot" : true,
      "lucene_version" : "8.7.0",
      "minimum_wire_compatibility_version" : "6.7.0",
      "minimum_index_compatibility_version" : "6.0.0-beta1"
   },
   "tagline" : "You Know, for Search"
}
```

NOTE

For more commands, see the **Elasticsearch documentation**.

4.4 Accessing a Cluster Using Java

4.4.1 Accessing a Cluster Through the Rest High Level Client

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

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

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

Precautions

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

Prerequisites

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

  7.6.2 indicates the version of the Elasticsearch Java client.

  – Maven mode:
  ```
  <dependency>
      <groupId>org.elasticsearch.client</groupId>
      <artifactId>elasticsearch-rest-high-level-client</artifactId>
      <version>7.6.2</version>
  </dependency>
  <dependency>
      <groupId>org.elasticsearch</groupId>
      <artifactId>elasticsearch</artifactId>
      <version>7.6.2</version>
  </dependency>
  ```
  – Gradle mode:
  ```
  compile group: 'org.elasticsearch.client', name: 'elasticsearch-rest-high-level-client', version: '7.6.2
  ```

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

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

```
import org.apache.http.HttpHost;
import org.elasticsearch.client.RequestOptions;
```

```
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.GetIndexRequest;

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

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

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

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

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

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

The sample code is as follows:

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

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

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

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


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

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

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

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

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

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

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

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

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

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

Table 4-3 Variables

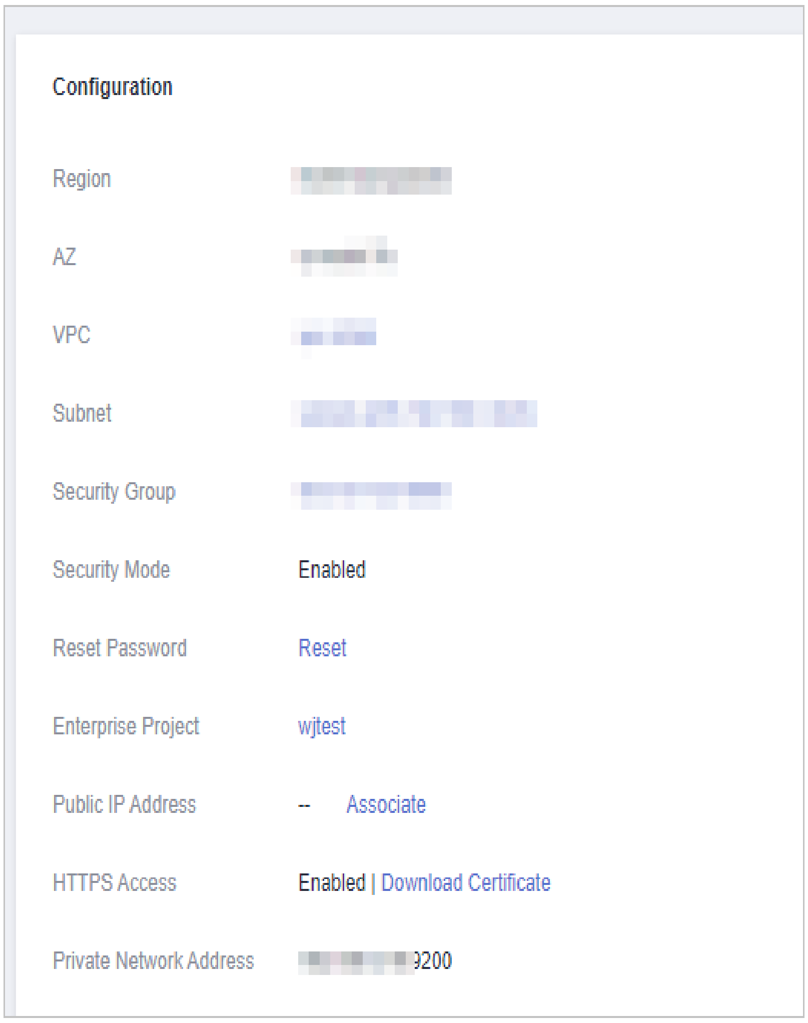
Parameter	Description
host	List of the IP addresses of Elasticsearch nodes (or independent Client node). Multiple IP addresses are separated using commas (,).
port	Access port of the Elasticsearch cluster. The default value is **9200**.
protocol	Connection protocol, which can be **http** or **https**.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

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

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

1. Obtain the security certificate **CloudSearchService.cer**.
   a. Log in to the CSS management console.
   b. In the navigation pane, choose **Clusters**. The cluster list is displayed.
   c. Click the name of a cluster to go to the cluster details page.
   d. On the **Configuration** page, click **Download Certificate** next to **HTTPS Access**.

Figure 4-1 Downloading a certificate



Configuration

Region	
AZ	
VPC	
Subnet	
Security Group	
Security Mode	Enabled
Reset Password	Reset
Enterprise Project	wjtest
Public IP Address	-- Associate
HTTPS Access	Enabled | Download Certificate
Private Network Address	9200

2. Convert the security certificate **CloudSearchService.cer**. Upload the downloaded security certificate to the client and use keytool to convert the .cer certificate into a .jks certificate that can be read by Java.
   - In Linux, run the following command to convert the certificate:
     ```
     keytool -import -alias newname -keystore ./truststore.jks -file ./CloudSearchService.cer
     ```
   - In Windows, run the following command to convert the certificate:
     ```
     keytool -import -alias newname -keystore .\truststore.jks -file .\CloudSearchService.cer
     ```

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

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

3. Access the cluster. The sample code is as follows:
   ```
   import org.apache.http.HttpHost;
   import org.apache.http.auth.AuthScope;
   import org.apache.http.auth.UsernamePasswordCredentials;
   import org.apache.http.client.CredentialsProvider;
   import org.apache.http.conn.ssl.NoopHostnameVerifier;
   import org.apache.http.impl.client.BasicCredentialsProvider;
   import org.apache.http.impl.nio.client.HttpAsyncClientBuilder;
   import org.apache.http.nio.conn.ssl.SSLIOSessionStrategy;
   import org.apache.logging.log4j.LogManager;
   ```

```
import org.apache.logging.log4j.Logger;
import org.elasticsearch.action.admin.cluster.health.ClusterHealthRequest;
import org.elasticsearch.action.admin.cluster.health.ClusterHealthResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.GetIndexRequest;
import org.elasticsearch.common.Nullable;

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

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

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

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

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

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

        ClusterHealthRequest request = new ClusterHealthRequest();
        ClusterHealthResponse response = client.cluster().health(request, RequestOptions.DEFAULT);
        logger.info("es rest client health response {} ", response);
        return client;
    }
    /**
     * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }

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

        private final SSLIOSessionStrategy sslStrategy;

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

        @Nullable
        CredentialsProvider getCredentialsProvider() {
            return credentialsProvider;
        }

        SSLIOSessionStrategy getSSLStrategy() {
            return sslStrategy;
        }

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

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

    public static class MyX509TrustManager implements X509TrustManager {
        X509TrustManager sunJSSEX509TrustManager;

        MyX509TrustManager(String cerFilePath, String cerPassword) throws Exception {
            File file = new File(cerFilePath);
            if (!file.isFile()) {
                throw new Exception("Wrong Certification Path");
            }
            System.out.println("Loading KeyStore " + file + "...");
            InputStream in = new FileInputStream(file);
            KeyStore ks = KeyStore.getInstance("JKS");
            ks.load(in, cerPassword.toCharArray());
            TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509", "SunJSSE");
            tmf.init(ks);
            TrustManager[] tms = tmf.getTrustManagers();
            for (TrustManager tm : tms) {
                if (tm instanceof X509TrustManager) {
                    sunJSSEX509TrustManager = (X509TrustManager) tm;
                    return;
                }
            }
            throw new Exception("Couldn't initialize");
        }
        @Override
        public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException {

        }

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

        }

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

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

Table 4-4 Function parameters

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

4.4.2 Accessing a Cluster Through the Rest Low Level Client

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

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

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

Precautions

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

Prerequisites

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

  7.6.2 indicates the version of the Elasticsearch Java client.

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

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

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

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

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

public class Main {

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

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

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

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

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

public class Main {

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

    /**
```

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

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

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

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

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

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

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

public class Main {

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

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

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

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

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

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

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

        private CustomConnectionKeepAliveStrategy() {
            super();
        }

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

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

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

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

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

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

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

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

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

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

import org.elasticsearch.client.RestHighLevelClient;

public class Main13 {

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

        final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
        credentialsProvider.setCredentials(AuthScope.ANY, new
UsernamePasswordCredentials(username, password));
```

```
        SSLContext sc = null;
        try {
            sc = SSLContext.getInstance("SSL");
            sc.init(null, trustAllCerts, new SecureRandom());
        } catch (KeyManagementException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        SSLIOSessionStrategy sessionStrategy = new SSLIOSessionStrategy(sc, new
NullHostNameVerifier());
        SecuredHttpClientConfigCallback httpClientConfigCallback = new
SecuredHttpClientConfigCallback(sessionStrategy,
            credentialsProvider);

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

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

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

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

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

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

        private CustomConnectionKeepAliveStrategy() {
            super();
        }

        /**
         * Maximum keep alive time (minutes)
         * The default value is 10 minutes. You can set it based on the number of TCP connections in
```

```
TIME_WAIT state. If there are too many TCP connections, you can increase the value.
         */
        private final long MAX_KEEP_ALIVE_MINUTES = 10;

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

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

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

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

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

Table 4-5 Variables

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

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

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

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

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

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

public class Main13 {

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

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

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

        RestClientBuilder builder = RestClient.builder(constructHttpHosts(host, port, protocol))
            .setRequestConfigCallback(requestConfig ->
requestConfig.setConnectTimeout(connectTimeout)
                .setConnectionRequestTimeout(connectionRequestTimeout)
                .setSocketTimeout(socketTimeout))
            .setHttpClientConfigCallback(httpClientConfigCallback);
        final RestClient client = builder.build();
        logger.info("es rest client build success {} ", client);
        return client;
    }
    /**
     * Use the constructHttpHosts function to convert the node IP address list of the host cluster.
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);}

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

        private final SSLIOSessionStrategy sslStrategy;

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

        @Nullable
        CredentialsProvider getCredentialsProvider() {
            return credentialsProvider;
        }

        SSLIOSessionStrategy getSSLStrategy() {
            return sslStrategy;
        }

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

    public static class MyX509TrustManager implements X509TrustManager {
        X509TrustManager sunJSSEX509TrustManager;

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

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

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

        }

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

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

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

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

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

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

public class Main {

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

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

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

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

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

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

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

        private final SSLIOSessionStrategy sslStrategy;

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

        @Nullable
        CredentialsProvider getCredentialsProvider() {
            return credentialsProvider;
        }

        SSLIOSessionStrategy getSSLStrategy() {
            return sslStrategy;
        }

        @Override
        public HttpAsyncClientBuilder customizeHttpClient(final HttpAsyncClientBuilder
httpClientBuilder) {
```

```
            httpClientBuilder.setSSLStrategy(sslStrategy);
            if (credentialsProvider != null) {
                httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider);
            }
            return httpClientBuilder;
        }}

    public static class MyX509TrustManager implements X509TrustManager {
        X509TrustManager sunJSSEX509TrustManager;

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

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

        }

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

        }

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

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

Table 4-6 Function parameters

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

4.4.3 Accessing the Cluster Through the Transport Client

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

Precautions

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

Prerequisites

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

  7.6.2 indicates the version of the Elasticsearch Java client.

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

Procedure

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

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

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

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

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

4.4.4 Using Spring Boot to Access a Cluster

You can access a CSS cluster using Spring Boot. Spring Boot can connect to a cluster in any of the following ways:

- **Accessing an HTTP Cluster Through Spring Boot**: applicable to clusters in non-security mode and clusters in Security mode+HTTP
- **Using Spring Boot to Access an HTTPS Cluster (Without Using Any Security Certificate)**: applicable to clusters in security mode+HTTPS
- **Using Spring Boot to Access an HTTPS Cluster (Using a Security Certificate)**: applicable to clusters in security mode+HTTPS

NOTE

For details about how to use Spring Boot, see the official document: **https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/**

Precautions

- You are advised to use the Elasticsearch Rest High Level Client version that matches the Elasticsearch version. For example, use Rest High Level Client 7.10.2 to access the Elasticsearch cluster 7.10.2.
- This section uses Spring Boot 2.5.5 as an example to describe how to connect Spring Boot to a cluster. The corresponding Spring Data Elasticsearch version is 4.2.*x*.

Prerequisites

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

  7.10.2 indicates the version of the Elasticsearch Java client.

  – Maven mode:
```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.5</version>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
    </dependency>
    <dependency>
        <groupId>org.elasticsearch.client</groupId>
        <artifactId>elasticsearch-rest-high-level-client</artifactId>
        <version>7.10.2</version>
    </dependency>
</dependencies>
```

Accessing an HTTP Cluster Through Spring Boot

This scenario applies to clusters in non-security mode or clusters in security mode +HTTP.

Configuration file:

```
elasticsearch.url=host1:9200,host2:9200
// You do not need to configure the following two lines for a non-security cluster.
elasticsearch.username=username
elasticsearch.password=password
```

Table 4-7 Parameter description

Parameter	Description
host	IP address of the Elasticsearch cluster node.
username	Username for accessing the cluster.
password	Password of the user.

Code:

NOTE

- **com.xxx** indicates the project directory, for example, **com.company.project**.
- **com.xxx.repository** is the repository directory, which is defined by **extends org.springframework.data.elasticsearch.repository.ElasticsearchRepository**.

```
package com.xxx.configuration;

import org.elasticsearch.client.RestHighLevelClient;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.elasticsearch.client.ClientConfiguration;
import org.springframework.data.elasticsearch.client.RestClients;
import org.springframework.data.elasticsearch.config.AbstractElasticsearchConfiguration;
import org.springframework.data.elasticsearch.repository.config.EnableElasticsearchRepositories;

@Configuration
@EnableElasticsearchRepositories(basePackages = "com.xxx.repository")
@ComponentScan(basePackages = "com.xxx")
public class Config extends AbstractElasticsearchConfiguration {

    @Value("${elasticsearch.url}")
    public String elasticsearchUrl;

    // You do not need to set the following two parameters for a non-security cluster.
    @Value("${elasticsearch.username}")
    public String elasticsearchUsername;

    @Value("${elasticsearch.password}")
    public String elasticsearchPassword;

    @Override
    @Bean
    public RestHighLevelClient elasticsearchClient() {
        final ClientConfiguration clientConfiguration = ClientConfiguration.builder()
            .connectedTo(elasticsearchUrl)
            // For a non-security cluster, you do not need to configure .withBasicAuth.
            .withBasicAuth(elasticsearchUsername, elasticsearchPassword)
            .build();

        return RestClients.create(clientConfiguration).rest();
    }
}
```

Using Spring Boot to Access an HTTPS Cluster (Without Using Any Security Certificate)

You can connect to a cluster in Security mode + HTTPS without using any security certificate.

Configuration file:

```
elasticsearch.url=host1:9200,host2:9200
elasticsearch.username=username
elasticsearch.password=password
```

Table 4-8 Parameter description

Parameter	Description
host	IP address of the Elasticsearch cluster node.
username	Username for accessing the cluster.
password	Password of the user.

Code:

NOTE

- **com.xxx** indicates the project directory, for example, **com.company.project**.
- **com.xxx.repository** is the repository directory, which is defined by **extends org.springframework.data.elasticsearch.repository.ElasticsearchRepository**.

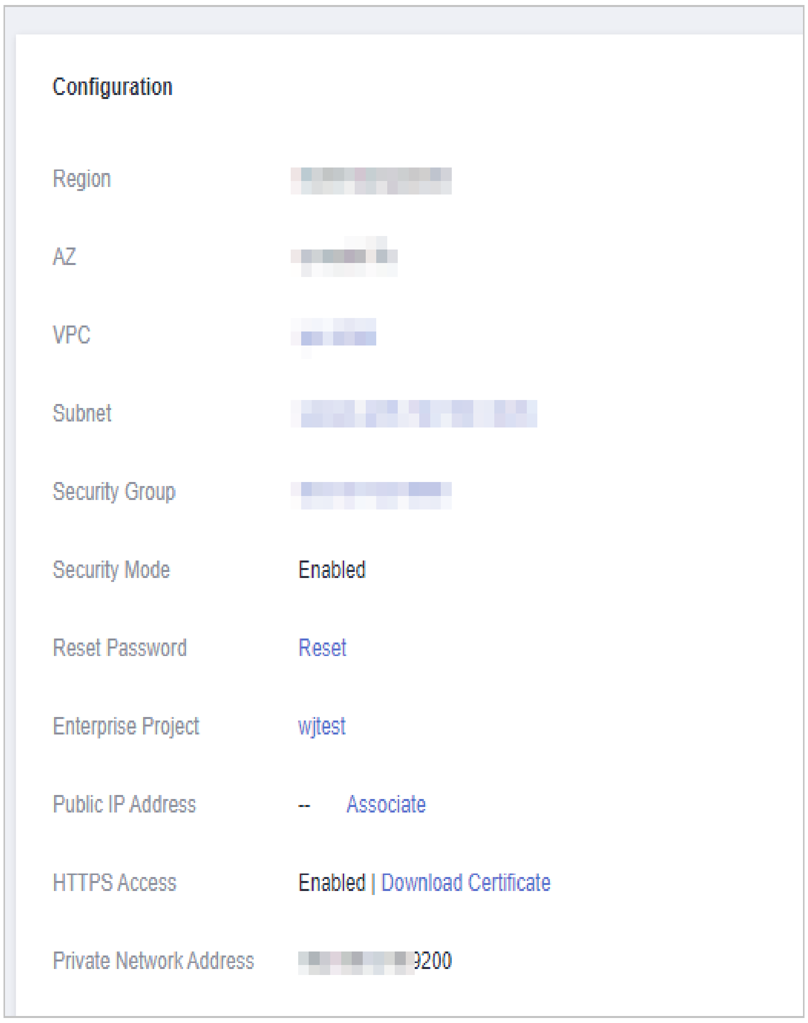
```
package com.xxx.configuration;
import org.elasticsearch.client.RestHighLevelClient;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.elasticsearch.client.ClientConfiguration;
import org.springframework.data.elasticsearch.client.RestClients;
import org.springframework.data.elasticsearch.config.AbstractElasticsearchConfiguration;
import org.springframework.data.elasticsearch.repository.config.EnableElasticsearchRepositories;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;
@Configuration
@EnableElasticsearchRepositories(basePackages = "com.xxx.repository")
@ComponentScan(basePackages = "com.xxx")
public class Config extends AbstractElasticsearchConfiguration {
    @Value("${elasticsearch.url}")
    public String elasticsearchUrl;
    @Value("${elasticsearch.username}")
    public String elasticsearchUsername;
    @Value("${elasticsearch.password}")
    public String elasticsearchPassword;
    @Override
    @Bean
    public RestHighLevelClient elasticsearchClient() {
        SSLContext sc = null;
        try {
            sc = SSLContext.getInstance("SSL");
            sc.init(null, trustAllCerts, new SecureRandom());
        } catch (KeyManagementException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        final ClientConfiguration clientConfiguration = ClientConfiguration.builder()
            .connectedTo(elasticsearchUrl)
            .usingSsl(sc, new NullHostNameVerifier())
            .withBasicAuth(elasticsearchUsername, elasticsearchPassword)
            .build();
        return RestClients.create(clientConfiguration).rest();
    }
    public static TrustManager[] trustAllCerts = new TrustManager[] {
        new X509TrustManager() {
            @Override
            public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException
{
            }
            @Override
            public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
            }
            @Override
            public X509Certificate[] getAcceptedIssuers() {
                return null;
            }
        }
    };
    public static class NullHostNameVerifier implements HostnameVerifier {
        @Override
        public boolean verify(String arg0, SSLSession arg1) {
            return true;
        }
    }
}
```

Using Spring Boot to Access an HTTPS Cluster (Using a Security Certificate)

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

1. Obtain the security certificate **CloudSearchService.cer**.
    a. Log in to the CSS management console.
    b. In the navigation pane, choose **Clusters**. The cluster list is displayed.
    c. Click the name of a cluster to go to the cluster details page.
    d. On the **Configuration** page, click **Download Certificate** next to **HTTPS Access**.

Figure 4-2 Downloading a certificate



Configuration

Region	[redacted]
AZ	[redacted]
VPC	[redacted]
Subnet	[redacted]
Security Group	[redacted]
Security Mode	Enabled
Reset Password	Reset
Enterprise Project	wjtest
Public IP Address	-- Associate
HTTPS Access	Enabled | Download Certificate
Private Network Address	[redacted]9200

2. Convert the security certificate **CloudSearchService.cer**. Upload the downloaded security certificate to the client and use keytool to convert the .cer certificate into a .jks certificate that can be read by Java.
   - In Linux, run the following command to convert the certificate:
     ```
     keytool -import -alias newname -keystore ./truststore.jks -file ./CloudSearchService.cer
     ```
   - In Windows, run the following command to convert the certificate:
     ```
     keytool -import -alias newname -keystore .\truststore.jks -file .\CloudSearchService.cer
     ```

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

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

3. **application.properties** configuration file:
   ```
   elasticsearch.url=host1:9200,host2:9200
   elasticsearch.username=username
   elasticsearch.password=password
   ```

Table 4-9 Parameter description

Parameter	Description
host	IP address of the Elasticsearch cluster node.
username	Username for accessing the cluster.
password	Password of the user.

4. Code:

NOTE

- **com.xxx** indicates the project directory, for example, **com.company.project**.
- **com.xxx.repository** is the repository directory, which is defined by **extends org.springframework.data.elasticsearch.repository.ElasticsearchRepository**.

```
package com.xxx.configuration;
import org.elasticsearch.client.RestHighLevelClient;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.elasticsearch.client.ClientConfiguration;
import org.springframework.data.elasticsearch.client.RestClients;
import org.springframework.data.elasticsearch.config.AbstractElasticsearchConfiguration;
import org.springframework.data.elasticsearch.repository.config.EnableElasticsearchRepositories;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.SecureRandom;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;
@Configuration
@EnableElasticsearchRepositories(basePackages = "com.xxx.repository")
@ComponentScan(basePackages = "com.xxx")
public class Config extends AbstractElasticsearchConfiguration {
    @Value("${elasticsearch.url}")
    public String elasticsearchUrl;
    @Value("${elasticsearch.username}")
    public String elasticsearchUsername;
    @Value("${elasticsearch.password}")
    public String elasticsearchPassword;
    @Override
    @Bean
    public RestHighLevelClient elasticsearchClient() {
        SSLContext sc = null;
        try {
            TrustManager[] tm = {new MyX509TrustManager(cerFilePath, cerPassword)};
            sc = SSLContext.getInstance("SSL", "SunJSSE");
            sc.init(null, tm, new SecureRandom());
        } catch (Exception e) {
            e.printStackTrace();
        }
        final ClientConfiguration clientConfiguration = ClientConfiguration.builder()
            .connectedTo(elasticsearchUrl)
            .usingSsl(sc, new NullHostNameVerifier())
            .withBasicAuth(elasticsearchUsername, elasticsearchPassword)
            .build();
        return RestClients.create(clientConfiguration).rest();
    }
    public static class MyX509TrustManager implements X509TrustManager {
        X509TrustManager sunJSSEX509TrustManager;
        MyX509TrustManager(String cerFilePath, String cerPassword) throws Exception {
            File file = new File(cerFilePath);
            if (!file.isFile()) {
                throw new Exception("Wrong Certification Path");
            }
            System.out.println("Loading KeyStore " + file + "...");
            InputStream in = new FileInputStream(file);
            KeyStore ks = KeyStore.getInstance("JKS");
            ks.load(in, cerPassword.toCharArray());
            TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509", "SunJSSE");
            tmf.init(ks);
            TrustManager[] tms = tmf.getTrustManagers();
            for (TrustManager tm : tms) {
                if (tm instanceof X509TrustManager) {
                    sunJSSEX509TrustManager = (X509TrustManager) tm;
                    return;
                }
            }
            throw new Exception("Couldn't initialize");
        }
        @Override
        public void checkClientTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
        }
        @Override
        public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {
        }
        @Override
        public X509Certificate[] getAcceptedIssuers() {
            return new X509Certificate[0];
        }
    }
    public static class NullHostNameVerifier implements HostnameVerifier {
        @Override
        public boolean verify(String arg0, SSLSession arg1) {
            return true;
        }
    }
}
```

In the preceding command, *cerFilePath* and *cerPassword* indicate the path and password of the .jks certificate, respectively.

4.5 Accessing a Cluster Using Python

You can access a CSS cluster using Python.

Prerequisites

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

Procedure

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

```
pip install Elasticsearch==7.6
```

2. Create an Elasticsearch client and check whether the **test** index exists. The examples for clusters in different security modes are as follows:
   - Cluster in non-security mode

```
from elasticsearch import Elasticsearch

class ElasticFactory(object):

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

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

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

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

   - Cluster in security mode + HTTP

```
from elasticsearch import Elasticsearch

class ElasticFactory(object):

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

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

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

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

   - Cluster in security mode + HTTPS

```
from elasticsearch import Elasticsearch
import ssl

class ElasticFactory(object):

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

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

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

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

Table 4-10 Variables

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

3. Create a cluster index through the Elasticsearch client.

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

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

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

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

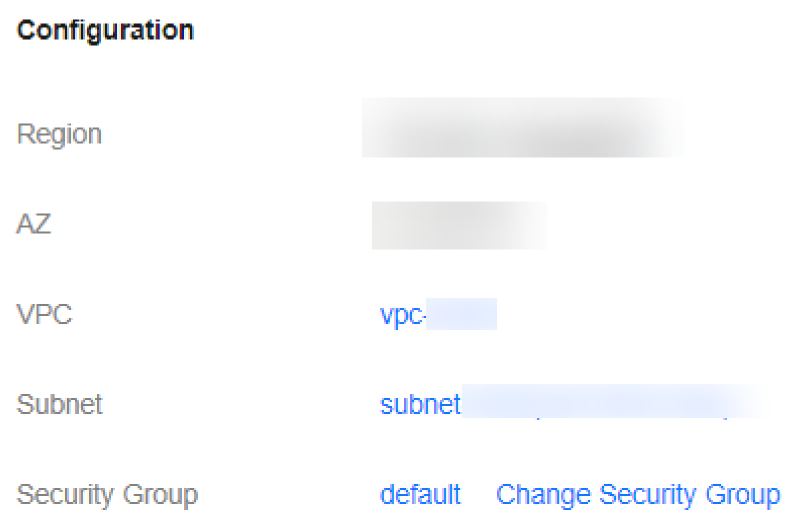
The Elasticsearch-Hadoop (ES-Hadoop) connector combines the massive data storage and in-depth processing capabilities of Hadoop with the real-time search and analysis capabilities of Elasticsearch. It allows you to quickly get to know big data and work better in the Hadoop ecosystem.

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

Prerequisites

- The CSS cluster is available.
- The client can communicate with the CSS cluster.
- The CSS and MRS clusters are in the same region, AZ, VPC, and subnet.

Figure 4-3 CSS cluster information



Configuration

Region	
AZ	
VPC	vpc-
Subnet	subnet
Security Group	default Change Security Group

Procedure

1. Obtain the private network address of the cluster. It is used to access the cluster.
   a. In the navigation pane on the left, choose **Clusters**.
   b. In the cluster list, select a cluster, and obtain and record its **Private Network Address**. Format: *<host>*:*<port>* or *<host>*:*<port>*,*<host>*:*<port>*

      If the cluster has only one node, the IP address and port number of only one node are displayed, for example, **10.62.179.32:9200**. If the cluster has multiple nodes, the IP addresses and port numbers of all nodes are displayed, for example, **10.62.179.32:9200,10.62.179.33:9200**.
2. Log in to an MRS cluster node. For details, see **Logging In to an ECS**.
3. Run the cURL command on an MRS cluster node to check the network connectivity. Ensure every node in the MRS cluster can connect to the CSS cluster.
   – Cluster in non-security mode
```
curl -X GET http://<host>:<port>
```

- Cluster in security mode + HTTP
  ```
  curl -X GET http://<host>:<port> -u <user>:<password>
  ```
- Cluster in security mode + HTTPS
  ```
  curl -X GET https://<host>:<port> -u <user>:<password> -ik
  ```

Table 4-11 Variables

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

4. Download the **ES-Hadoop lib package** and decompress it to obtain the **elasticsearch-hadoop-x.x.x.jar** file. The version must be the same as the CSS cluster version. For example, if the CSS cluster version is 7.6.2, you are advised to download **elasticsearch-hadoop-7.6.2.zip**.
5. Download the httpclient dependency package **commons-httpclient:commons-httpclient-3.1.jar**. In the package name, **3.1** indicates the version number. Select the package of the version you need.
6. Install the MRS client. If the MRS client has been installed, skip this step. For details, see **Installing a Client (MRS 3.x or Later)**.
7. Log in to the MRS client. Upload the JAR dependency packages of ES-Hadoop and httpclient to the MRS client.
8. Create an HDFS directory on the MRS client. Upload the ES-Hadoop lib package and the httpclient dependency package to the directory.
   ```
   hadoop fs -mkdir /tmp/hadoop-es
   hadoop fs -put elasticsearch-hadoop-x.x.x.jar /tmp/hadoop-es
   hadoop fs -put commons-httpclient-3.1.jar /tmp/hadoop-es
   ```
9. Log in to the Hive client from the MRS client. For details, see **Using a Hive Client**.
10. On the Hive client, add the ES-Hadoop lib package and the httpclient dependency package. This command is valid only for the current session.

    Enter **beeline** or **hive** to go to the execution page and run the following commands:
    ```
    add jar hdfs:///tmp/hadoop-es/commons-httpclient-3.1.jar;
    add jar hdfs:///tmp/hadoop-es/elasticsearch-hadoop-x.x.x.jar;
    ```
11. On the Hive client, create a Hive foreign table.
    - Cluster in non-security mode
      ```
      CREATE EXTERNAL table IF NOT EXISTS student(
        id BIGINT,
        name STRING,
        addr STRING
      )
      STORED BY 'org.elasticsearch.hadoop.hive.EsStorageHandler'
      TBLPROPERTIES(
         'es.nodes' = 'xxx.xxx.xxx.xxx:9200',
      ```

```
    'es.port' = '9200',
    'es.net.ssl' = 'false',
    'es.nodes.wan.only' = 'false',
    'es.nodes.discovery'='false',
    'es.input.use.sliced.partitions'='false',
    'es.resource' = 'student/_doc'
);
```

– Cluster in security mode + HTTP

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

– Cluster in security mode + HTTPS

i. Obtain the security certificate **CloudSearchService.cer**.

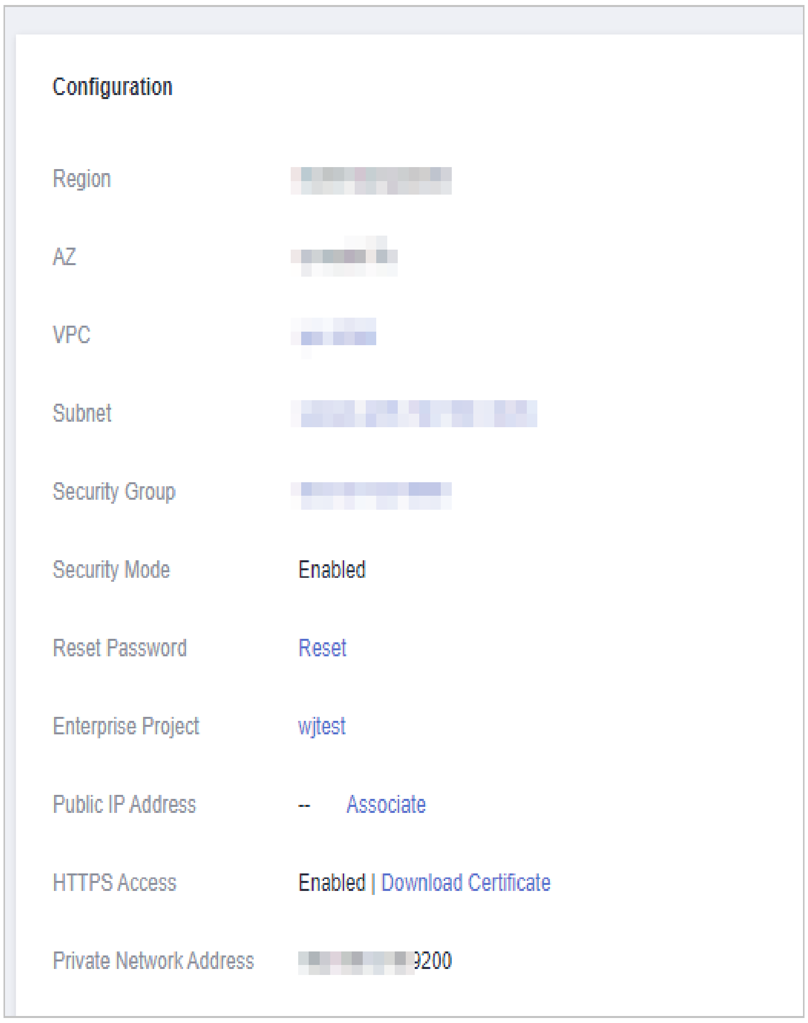
1) Log in to the CSS management console.

2) In the navigation pane, choose **Clusters**. The cluster list is displayed.

3) Click the name of a cluster to go to the cluster details page.

4) On the **Configuration** page, click **Download Certificate** next to **HTTPS Access**.

Figure 4-4 Downloading a certificate



Configuration

Region	
AZ	
VPC	
Subnet	
Security Group	
Security Mode	Enabled
Reset Password	Reset
Enterprise Project	wjtest
Public IP Address	-- Associate
HTTPS Access	Enabled | Download Certificate
Private Network Address	9200

ii. Convert the security certificate **CloudSearchService.cer**. Upload the downloaded security certificate to the client and use keytool to convert the .cer certificate into a .jks certificate that can be read by Java.

  - In Linux, run the following command to convert the certificate:
```
keytool -import -alias newname -keystore ./truststore.jks -file ./
CloudSearchService.cer
```
  - In Windows, run the following command to convert the certificate:
```
keytool -import -alias newname -keystore .\truststore.jks -
file .\CloudSearchService.cer
```

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

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

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

iv. Create a Hive foreign table.

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

Table 4-12 ES-Hadoop parameters

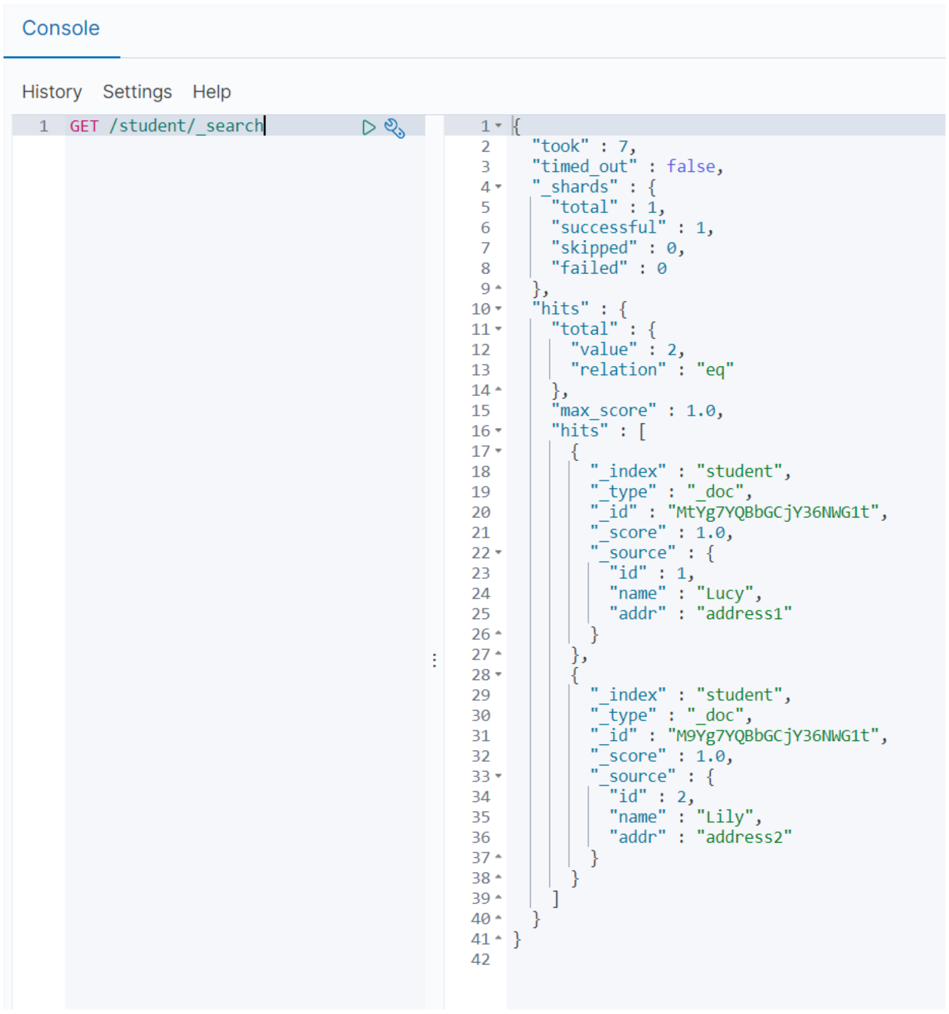
Parameter	Default Value	Description
es.nodes	localhost	Address for accessing the CSS cluster. You can view private network address in the cluster list.
es.port	9200	Port number for accessing a cluster. Generally, the port number is 9200.
es.nodes.wan.only	false	Whether to perform node sniffing.
es.nodes.discovery	true	Whether to disable node discovery.
es.input.use.sliced.partitions	true	Whether to use slices. Its value can be: ● **true** ● **false** **NOTE** If this parameter is set to **true**, the index prefetch time may be significantly prolonged, and may even be much longer than the data query time. You are advised to set this parameter to **false** to improve query efficiency.
es.resource	NA	Specifies the index and type to be read and written.
es.net.http.auth.user	NA	Username for accessing the cluster. Set this parameter only if the security mode is enabled.

Parameter	Default Value	Description
es.net.http.auth.pass	NA	Password of the user. Set this parameter only if the security mode is enabled.
es.net.ssl	false	Whether to enable SSL. If SSL is enabled, you need to configure the security certificate information.
es.net.ssl.truststore.location	NA	Path of the .jks certificate file, for example, **file:///tmp/truststore.jks**.
es.nodes.client.only	false	Check whether the IP address of an independent Client node is configured for **es.nodes** (that is, whether the Client node is enabled during Elasticsearch cluster creation). If yes, change the value to **true**, or an error will be reported, indicating that the data node cannot be found.
es.net.ssl.truststore.pass	NA	Password of the .jks certificate file.

For details about ES-Hadoop configuration items, see the **official configuration description**.

12. On the Hive client, insert data.
```
INSERT INTO TABLE student VALUES (1, "Lucy", "address1"), (2, "Lily", "address2");
```
13. On the Hive client, run a query.
```
select * from student;
```
The query result is as follows:
```
+-------------+---------------+---------------+
| student.id  | student.name  | student.addr  |
+-------------+---------------+---------------+
| 1           | Lucy          | address1      |
| 2           | Lily          | address2      |
+-------------+---------------+---------------+
2 rows selected (0.116 seconds)
```
14. Log in to the CSS console and choose **Clusters**. Locate the target cluster and click **Access Kibana** in the **Operation** column.
15. On the **Dev Tools** page of Kibana, run a query and view the result.
```
GET /student/_search
```

Figure 4-5 Kibana query result



```
GET /student/_search
```

```
{
  "took" : 7,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "student",
        "_type" : "_doc",
        "_id" : "MtYg7YQBbGCjY36NWG1t",
        "_score" : 1.0,
        "_source" : {
          "id" : 1,
          "name" : "Lucy",
          "addr" : "address1"
        }
      },
      {
        "_index" : "student",
        "_type" : "_doc",
        "_id" : "M9Yg7YQBbGCjY36NWG1t",
        "_score" : 1.0,
        "_source" : {
          "id" : 2,
          "name" : "Lily",
          "addr" : "address2"
        }
      }
    ]
  }
}
```

4.7 Accessing a Cluster Using Go

This section describes how to access a CSS cluster using Go.

Preparations

- The CSS cluster is available.
- Ensure that the server running Go can communicate with the CSS cluster.
- Ensure that Go has been installed on the server. You can download Go from the official website: https://go.dev/dl/.

Connecting to a Non-Security Cluster

Connect to a non-security cluster. The sample code is as follows:

```
package main

import (
    "github.com/elastic/go-elasticsearch/v7"
    "log"
)
```

```
func main() {
    cfg := elasticsearch.Config{
        Addresses: []string{
            "http://HOST:9200/",
        },
    }

    es, _ := elasticsearch.NewClient(cfg)
    log.Println(es.Info())
}
```

Connecting to a Security Cluster

- Connect to a security cluster with HTTPS disabled. The sample code is as follows:

```
package main

import (
    "github.com/elastic/go-elasticsearch/v7"
    "log"
)

func main() {
    cfg := elasticsearch.Config{
        Addresses: []string{
            "http://HOST:9200/",
        },
        Username: "USERNAME",
        Password: "PASSWORD",
    }

    es, _ := elasticsearch.NewClient(cfg)
    log.Println(es.Info())
}
```

- Connect to a security cluster that has enabled HTTPS and does not use certificates. The sample code is as follows:

```
package main

import (
    "crypto/tls"
    "github.com/elastic/go-elasticsearch/v7"
    "log"
    "net/http"
)

func main() {
    cfg := elasticsearch.Config{
        Addresses: []string{
            "https://HOST:9200/",
        },
        Username: "USERNAME",
        Password: "PASSWORD",
        Transport: &http.Transport{
            TLSClientConfig: &tls.Config{
                InsecureSkipVerify: true,
            },
        },
    }

    es, _ := elasticsearch.NewClient(cfg)
    log.Println(es.Info())
}
```

- Connect to a security cluster that has enabled HTTPS and uses certificates. The sample code is as follows:

```
package main

import (
    "crypto/tls"
    "crypto/x509"
    "flag"
    "github.com/elastic/go-elasticsearch/v7"
    "io/ioutil"
    "log"
    "net"
    "net/http"
    "time"
)

func main() {
    insecure := flag.Bool("insecure-ssl", false, "Accept/Ignore all server SSL certificates")
    flag.Parse()

    // Get the SystemCertPool, continue with an empty pool on error
    rootCAs, _ := x509.SystemCertPool()
    if rootCAs == nil {
        rootCAs = x509.NewCertPool()
    }

    // Read in the cert file
    certs, err := ioutil.ReadFile("/tmp/CloudSearchService.cer")
    if err != nil {
        log.Fatalf("Failed to append %q to RootCAs: %v", "xxx", err)
    }

    // Append our cert to the system pool
    if ok := rootCAs.AppendCertsFromPEM(certs); !ok {
        log.Println("No certs appended, using system certs only")
    }

    config := elasticsearch.Config{
        Addresses: []string{
            "https://HOST:9200/",
        },
        Username: "USERNAME",
        Password: "PASSWORD",
        Transport: &http.Transport{
            MaxIdleConnsPerHost:   10,
            ResponseHeaderTimeout: time.Second,
            DialContext: (&net.Dialer{
                Timeout:   30 * time.Second,
                KeepAlive: 30 * time.Second,
            }).DialContext,
            TLSClientConfig: &tls.Config{
                InsecureSkipVerify: *insecure,
                RootCAs:            rootCAs,
            },
        },
    }
    es, _ := elasticsearch.NewClient(config)
    log.Println(elasticsearch.Version)
    log.Println(es.Info())
}
```

Running Code

Write the preceding code to the **EsTest.gc** file based on the cluster type and save the file to an independent directory. Run the following command in the directory:

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://repo.huaweicloud.com/repository/goproxy/
go env -w GONOSUMDB=*

go mod init test
go mod tidy
go run EsTest.go
```

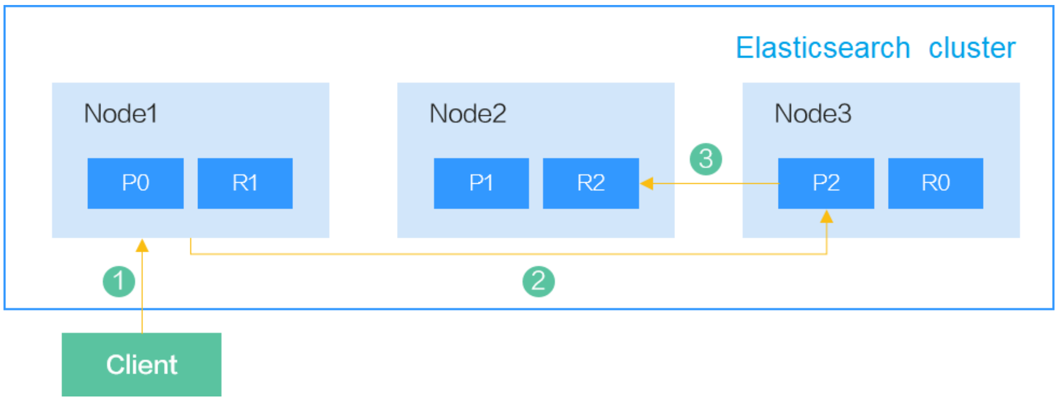
5 Cluster Performance Tuning

5.1 Optimizing Write Performance

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

Data Write Process

Figure 5-1 Data write process

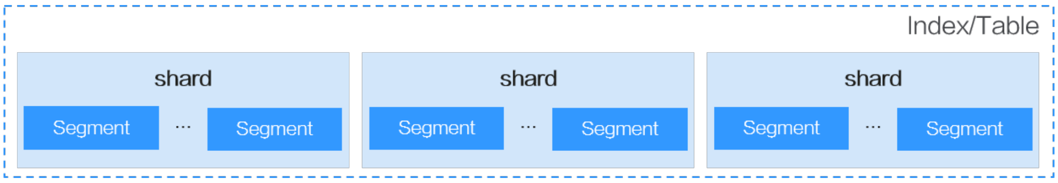


The process of writing data from a client to Elasticsearch is as follows:

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

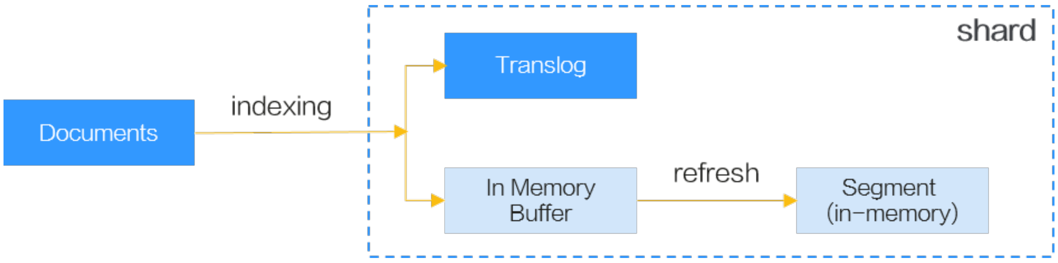
An index in Elasticsearch consists of one or more shards. Each shard contains multiple segments, and each segment is an inverted index.

Figure 5-2 Elasticsearch index composition



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

Figure 5-3 Process of inserting a document into Elasticsearch



Improving Write Performance

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

Table 5-1 Improving write performance

No.	Solution	Description
1	Use SSDs or improve cluster configurations.	Using SSDs can greatly speed up data write and merge operations. For CSS, you are advised to select the ultra-high I/O storage or ultra-high I/O servers.
2	Use Bulk APIs.	The client writes data in batches. You are advised to write 1 MB to 10 MB data in each batch.
3	Randomly generate **_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**.
4	Set a proper number of segments.	You are advised to set the number of shards to a multiple of the number of cluster data nodes. Ensure each shard is smaller than 50 GB.

No.	Solution	Description

5	Close replicas.	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`
		`}`

6	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"`
		`}`

7	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**

8	Set a proper field type.	Specify the type of each field in the cluster, so that Elasticsearch will not regard the fields as a combination of keywords and texts, which unnecessarily increase data volume. Keywords are used for keyword search, and texts used for full-text search.
		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`
		`      }`
		`    }`
		`  }`
		`}`

No.	Solution	Description

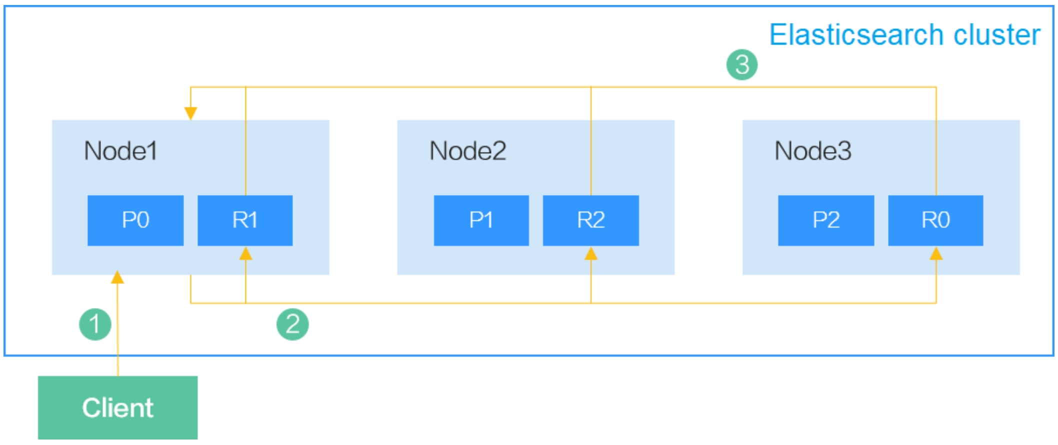
9	Optimize the shard balancing policy.	By default, Elasticsearch uses the load balance policy based on the disk capacity. If there are multiple nodes, especially if some of them are newly added, shards may be unevenly allocated on the nodes. To avoid such problems, you can set the index-level parameter **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`
		`    }`
		`}`

5.2 Optimizing Query Performance

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

Data Query Process

Figure 5-4 Data query process



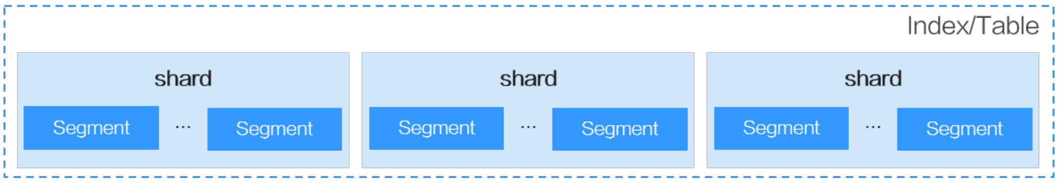
When a client sends a query request to Elasticsearch, the query process is as follows:

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

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 5-5 Elasticsearch index composition



Improving Query Performance

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

Table 5-2 Improving query performance

No.	Solution	Description

1	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`

No.	Solution	Description

2	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"`
		`      }`
		`    }`
		`  }`
		`}`

3	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**. For details, see **Configuring Parameters**. Restart the cluster for the modification to take effect.

4	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`

6 Managing the Index Lifecycle

6.1 Configuring the Lifecycle to Automate Index Rollover

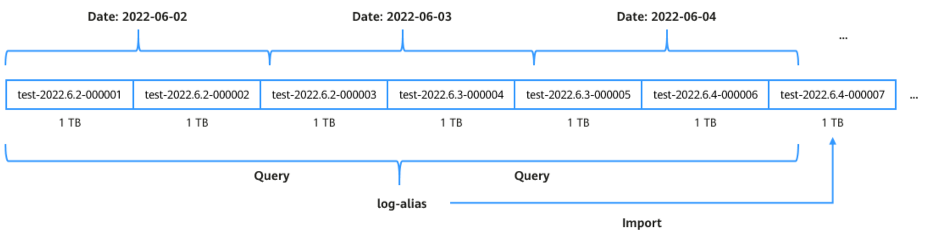
Overview

Time series data is continuously written and increases index size. You can configure the lifecycle to periodically roll over to new indexes and delete old indexes.

In this section, a lifecycle policy is configured. If the size of an index reaches 1 TB or the index has been created for more than one day, a new index will be automatically generated. Seven days after the index is created, the data copy will be disabled. Thirty days after the index is created, the index will be deleted.

Assume that an index generates about 2.4 TB data every day. The index alias is **log-alias**. The following figure shows the data format in Elasticsearch. During read, it points to all indexes starting with **test**. During write, it points to the latest index.

Figure 6-1 log-alias format



NOTE

The one day in the rollover time refers to 24 hours after the index creation time, not a calendar day.

Prerequisites

- The CSS cluster is available.
- The cluster version is Elasticsearch 7.6.2 or later.

Procedure

1. Log in to the CSS management console.
2. In the navigation pane on the left, choose **Clusters** to go to the Elasticsearch cluster list.
3. Click **Access Kibana** in the **Operation** column of a cluster.
4. In the navigation tree on the left of Kibana, choose **Dev Tools**. The command execution page is displayed.
5. Create a rollover lifecycle policy named **rollover_workflow**.

   Policy description: When the size of an index reaches 1 TB or the index has been created for more than one day, the index rollover is performed. When the index has been created for seven days, the data copy is disabled. When the index has been created for 30 days, the index is deleted.

```
PUT _opendistro/_ism/policies/rollover_workflow
{
  "policy": {
    "description": "rollover test",
    "default_state": "hot",
    "states": [
      {
        "name": "hot",
        "actions": [
          {
            "rollover": {
              "min_size": "1tb",
              "min_index_age": "1d"
            }
          }
        ],
        "transitions": [
          {
            "state_name": "warm",
            "conditions": {
              "min_index_age": "7d"
            }
          }
        ]
      },
      {
        "name": "warm",
        "actions": [
          {
            "replica_count": {
              "number_of_replicas": 0
            }
          }
        ],
        "transitions": [
          {
            "state_name": "delete",
            "conditions": {
              "min_index_age": "30d"
            }
          }
        ]
      },
      {
        "name": "delete",
        "actions": [
          {
            "delete": {}
          }
        ]
      }
    ]
  }
}
```

After a lifecycle policy is created, run the following command to query the policy details:

```
GET _opendistro/_ism/policies/rollover_workflow
```

6. Create the index template **template_test**.

   Template description: All the new indexes starting with **test** are automatically associated with the rollover lifecycle policy **rollover_workflow**. The alias **log_alias** is used during rollover.

```
PUT _template/template_test
{
  "index_patterns": "test*",
  "settings": {
    "number_of_replicas": 1,
    "number_of_shards": 1,
    "opendistro.index_state_management.policy_id": "rollover_workflow",
    "index.opendistro.index_state_management.rollover_alias": "log_alias"
  },
  "mappings": {
    "properties": {
      "name": {
        "type": "text"
      }
    }
  }
}
```

Table 6-1 Parameter description

Parameter	Description
number_of_shards	Number of index shards
number_of_replicas	Number of index shard replicas
opendistro.index_state_management.policy_id	Lifecycle policy name
index.opendistro.index_state_management.rollover_alias	Index alias for rollover

After an index template is created, you can run the following command to query the template details:

```
GET _template/template_test
```

7. Create an index, specify **aliases**, and set **is_write_index** to **true**. The index template **template_test** is automatically used for the index and is associated with the lifecycle policy **rollover_workflow** based on the index template configuration. In this way, when the index size reaches 1 TB or the index is

created for more than one day, the rollover automatically starts. After an index is created for seven days, the data copy is disabled. After an index is created for 30 days, the index is deleted.

The following index is the URL code of **<test-{now/d}-000001>**. By default, an index name contains the creation date. For example, if an index is created on 2022-06-02, the index name is **test-2022.06.02-000001**.

```
PUT %3Ctest-%7Bnow%2Fd%7D-000001%3E
{
  "aliases": {
    "log_alias": {
      "is_write_index": true
    }
  }
}
```

8. The alias **log_alias** is used to during data write, and **log_alias** always points to the last index.

```
POST log_alias/_bulk
{"index":{}}
{"name":"name1"}
{"index":{}}
{"name":"name2"}
{"index":{}}
{"name":"name3"}
{"index":{}}
{"name":"name4"}
{"index":{}}
{"name":"name5"}
{"index":{}}
{"name":"name6"}
```

9. Query data and check whether the rollover takes effect.
   - One day after the indexes are created, check the indexes starting with **test**.

     ```
     GET _cat/indices/test*?s=i
     ```

     There are supposed to be at least two indexes, for example:

     ```
     green open test-<Date>-000001 r8ab5NX6T3Ox_hoGUanogQ 1 1 6 0 416b 208b
     green open test-<Date>-000002 sfwkVgy8RSSEw7W-xYjM2Q 1 1 0 0 209b 209b
     ```

     In the preceding information, **test-<Date>-000001** is the index created in 7, and **test-<Date>-000002** is the index generated in rollover.
   - To query the index associated with the alias **log_alias**, run the following command:

     ```
     GET _cat/aliases/log_alias?v
     ```

     The alias is supposed to point to multiple indexes, for example:

     ```
     alias     index                filter routing.index routing.search is_write_index
     log_alias test-<Date>-000001   -      -             -              false
     log_alias test-<Date>-000002   -      -             -              true
     ```

6.2 Configuring the Lifecycle to Decouple Storage and Compute

Overview

CSS supports decoupled storage and compute. That is, indexes can be frozen in OBS to reduce the storage cost of cold data. This document describes how to use index lifecycle management to automatically freeze indexes at a specific time to decouple storage and compute.

In this section, a lifecycle policy is configured to automatically freeze an index three days after it is created and dump data to OBS. The index will be deleted seven days after it is created.

Prerequisites

- The CSS cluster is available.
- The cluster version is Elasticsearch 7.6.2 or later.

Procedure

1. Log in to the CSS management console.
2. In the navigation pane on the left, choose **Clusters** to go to the Elasticsearch cluster list.
3. Click **Access Kibana** in the **Operation** column of a cluster.
4. In the navigation tree on the left of Kibana, choose **Dev Tools**. The command execution page is displayed.
5. Create a lifecycle policy named **hot_warm_policy**.

   Policy description: Three days after an index is created, the API for freezing the index is automatically called to dump data to OBS. Seven days after an index is created, the index is deleted.

```
PUT _opendistro/_ism/policies/hot_warm_policy
{
  "policy": {
    "description": "hot warm delete workflow",
    "error_notification": null,
    "default_state": "hot",
    "states": [
      {
        "name": "hot",
        "actions": [],
        "transitions": [
          {
            "state_name": "warm",
            "conditions": {
              "min_index_age": "3d"
            }
          }
        ]
      },
      {
        "name": "warm",
        "actions": [
          {
            "freeze_low_cost": {}
          }
        ],
        "transitions": [
          {
            "state_name": "delete",
            "conditions": {
              "min_index_age": "7d"
            }
          }
        ]
      },
      {
        "name": "delete",
        "actions": [
          {
            "delete": {}
          }
        ],
        "transitions": []
      }
    ]
  }
}
```

6. Create the index template **template_hot_warm**.

   Template description: All the new indexes starting with **data** are automatically associated with the lifecycle policy **hot_warm_policy**.

```
PUT _template/template_hot_warm
{
  "index_patterns": "data*",
  "settings": {
    "number_of_replicas": 5,
    "number_of_shards": 1,
    "opendistro.index_state_management.policy_id": "hot_warm_policy"
  },
  "mappings": {
    "properties": {
      "name": {
        "type": "text"
      }
    }
  }
}
```

Table 6-2 Parameter description

Parameter	Description
number_of_shards	Number of index shards
number_of_replicas	Number of index shard replicas
opendistro.index_state_management.policy_id	Lifecycle policy name

7. Create the **data-2022-06-06** index. The index automatically uses the **template_hot_warm** template and associates the index template with the lifecycle policy **hot_warm_policy**. In this way, the index is frozen three days after creation and is deleted seven days after creation.

```
POST data-2022-06-06/_bulk
{"index":{}}
{"name":"name1"}
{"index":{}}
{"name":"name2"}
{"index":{}}
{"name":"name3"}
{"index":{}}
{"name":"name4"}
{"index":{}}
{"name":"name5"}
{"index":{}}
{"name":"name6"}
```

8. Query data and check whether storage and compute is automatically decoupled.

   – Three days after the index is created, check the frozen index.

```
GET _cat/freeze_indices?s=i&v
```

The index generated three days ago is expected to be frozen.

```
health status index              uuid                  pri rep docs.count docs.deleted store.size pri.store.size
green  open   data-2022-06-06  x8ab5NX6T3Ox_xoGUanogQ    1   1          6            0      7.6kb          3.8kb
```

– Seven days after the index is created, check the frozen index. The index is expected to be deleted.

7 Practices

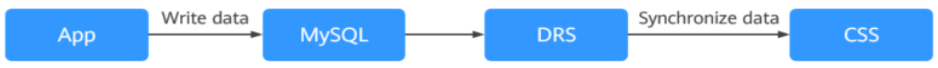
7.1 Using CSS to Accelerate Database Query and Analysis

Overview

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

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

Figure 7-1 Using CSS to accelerate database query and analysis



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

Prerequisites

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

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

b. Insert the initial data of three students into the MySQL database.

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

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

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

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

  Configure **number_of_shards** and **number_of_replicas** as needed.

Procedure

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

In this example, configure the parameters by following the suggestions in **Table 7-1**.

Table 7-1 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 CSS cluster are located.

Module	Parameter	Suggestion
Configure Source and Destination Databases > **Destination Database**	**VPC** and **Subnet**	Select the VPC and subnet of the CSS cluster.
	IP Address or Domain Name	Enter the IP address of the CSS cluster. For details, see **Obtaining the IP address of a CSS cluster**.
	Database Username and **Database Password**	Enter the administrator username (**admin**) and password of the CSS cluster.
	Encryption Certificate	Select the security certificate of the CSS 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 CSS cluster.
	Synchronization Object	Select **Tables**. Select the database and table name corresponding to CSS. **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 Check the synchronization status of the database.

1. Verify full data synchronization.

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

   ```
   GET student/_search
   ```

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

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

   ```
   INSERT INTO student (id,name,age,dsc)
   VALUES
   ('4','Bill Gates','50','Gates III is an American business magnate, software developer, investor, author,
   and philanthropist.')
   ```

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

```
GET student/_search
```

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

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

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

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

   ```
   GET student/_search
   ```

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

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

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

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

   ```
   GET student/_search
   ```

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

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

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

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

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

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

Step 5 Verify the statistical analysis capability of the database.

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

```
GET student/_search
{
  "size": 0,
  "query": {
    "match_all": {}
  },
  "aggs": {
    "age_count": {
      "terms": {
        "field": "age",
        "size": 10
      }
    }
  }
}
```

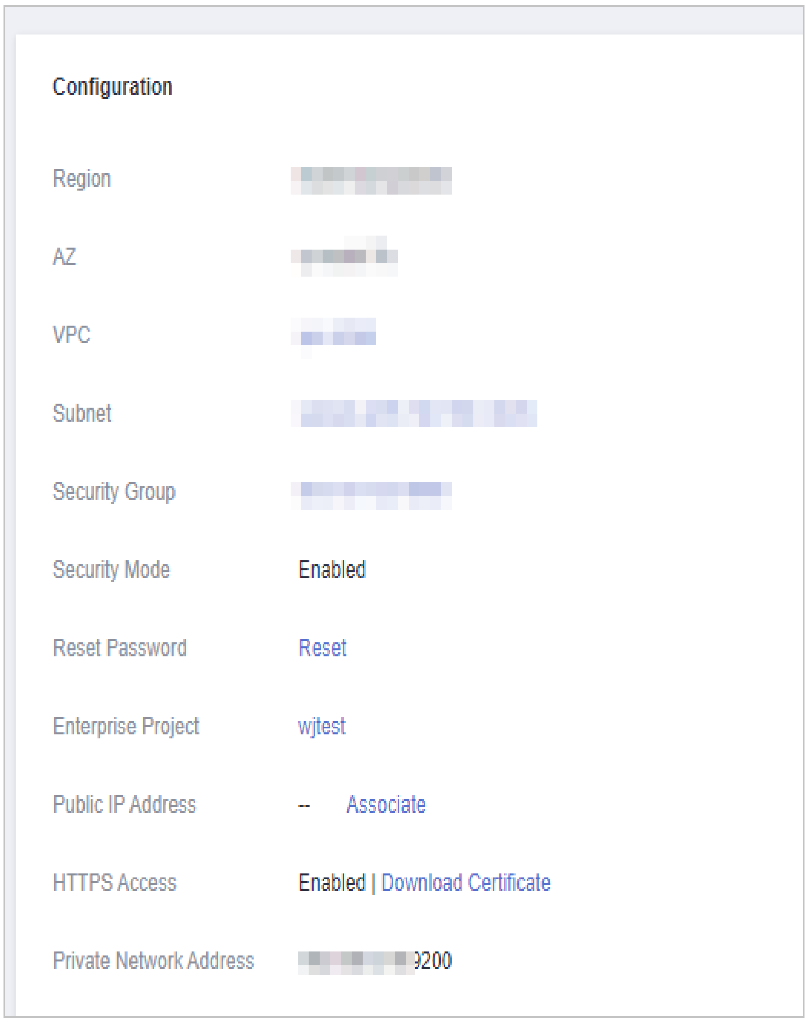
----End

Other Operations

- **Obtaining the IP address of a CSS cluster**
  a. In the navigation pane on the left, choose **Clusters**.
  b. In the cluster list, locate a cluster, and obtain the IP address of the CSS cluster from the **Private Network Address** column. Generally, the IP address format is *<host>***:***<port>* or *<host>***:***<port>***,***<host>***:***<port>*.

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

Figure 7-2 Downloading a certificate



Configuration

Region	
AZ	
VPC	
Subnet	
Security Group	
Security Mode	Enabled
Reset Password	Reset
Enterprise Project	wjtest
Public IP Address	-- Associate
HTTPS Access	Enabled | Download Certificate
Private Network Address	9200

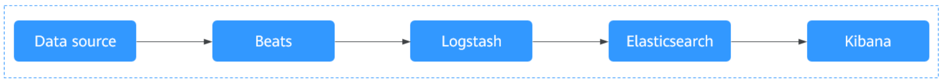
7.2 Using CSS to Build a Unified Log Management Platform

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

Overview

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

Figure 7-3 Unified log management platform framework



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

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

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

Prerequisites

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

Procedure

Step 1 Deploy and configure Filebeat.

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

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

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

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

processors:
```

Step 2 Deploy and configure Logstash.

NOTE

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

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

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

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

NOTE

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

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

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

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

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

Step 4 Prepare test data on ECS.

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

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

The following is an example of the generated test data:

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

Step 5 Run the following command to start Logstash:

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

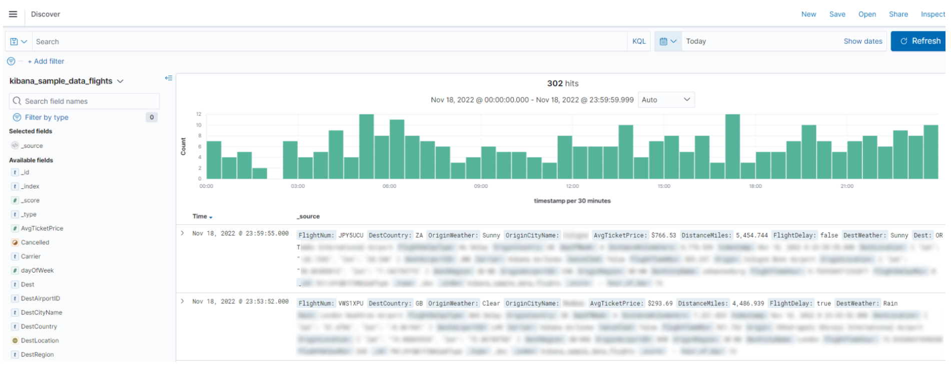
Step 6 Run the following command to start Filebeat:

```
./filebeat
```

Step 7 Use Kibana to query data and create reports.

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

Figure 7-4 Discover page



----End

7.3 Configuring Query Scoring in an Elasticsearch Cluster

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

Overview

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

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

  new_score = **query_score** x (**vote** x **factor**)

  – **query_score**: calculated based on the total number of search keywords found in a record. A record earns 1 point for each keyword it contains.

  – **vote**: vote of a record.

  – **factor**: user-defined weight of **vote**.

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

  new_score = **query_score** x **inline**

  – **query_score**: calculated based on the total number of search keywords found in a record. A record earns 1 point for each keyword it contains.

  – **vote**: vote of a record.

  – **inline**: Configure two value options for this parameter and a threshold for **vote**. One option is used if **vote** exceeds the threshold, and the other is used if **vote** is smaller than or equal to the threshold. In this way, the query accuracy will not be affected by abnormal **vote** values.

Prerequisites

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

Procedure

NOTE

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

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

   For example, the content of the **tv.json** file is as follows:

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

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

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

6. Import data.

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

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

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

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

   – Scoring based on **vote**

     The score is calculated using the formula **new_score** = **query_score** x (**vote** x **factor**). Run the following command:

```
GET /mall/_doc/_search?pretty
{
  "query":{
    "function_score":{
      "query":{
        "bool":{
          "should":[
            {"match": {"description": "USB"}},
            {"match": {"description": "HDMI"}},
            {"match": {"description": "DisplayPort"}}
          ]
        }
      },
      "field_value_factor":{
        "field":"vote",
        "factor":1
      },
      "boost_mode":"multiply",
      "max_boost":10
    }
  }
}
```

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

```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 0.8388366,
    "hits" : [
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "4",
        "_score" : 0.8388366,
        "_source" : {
          "name" : "tv4",
          "description" : "USB, HDMI, DisplayPort",
          "vote" : 0.7
        }
      },
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "2",
        "_score" : 0.7428025,
        "_source" : {
          "name" : "tv2",
          "description" : "USB, HDMI",
          "vote" : 0.99
        }
      },
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 0.7352994,
        "_source" : {
          "name" : "tv1",
          "description" : "USB, DisplayPort",
          "vote" : 0.98
        }
      },
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "3",
        "_score" : 0.03592815,
        "_source" : {
          "name" : "tv3",
          "description" : "USB",
          "vote" : 0.5
        }
      }
    ]
  }
}
```

– Scoring based on **inline**

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

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

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

```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 0.75030553,
    "hits" : [
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 0.75030553,
        "_source" : {
          "name" : "tv1",
          "description" : "USB, DisplayPort",
          "vote" : 0.98
        }
      },
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "2",
        "_score" : 0.75030553,
        "_source" : {
          "name" : "tv2",
          "description" : "USB, HDMI",
          "vote" : 0.99
        }
      },
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "4",
        "_score" : 0.599169,
        "_source" : {
          "name" : "tv4",
          "description" : "USB, HDMI, DisplayPort",
          "vote" : 0.7
        }
      },
      {
        "_index" : "mall",
        "_type" : "_doc",
        "_id" : "3",
        "_score" : 0.03592815,
        "_source" : {
          "name" : "tv3",
          "description" : "USB",
          "vote" : 0.5
        }
      }
    ]
  }
}
```