云搜索服务

最佳实践

文档版本 06

发布日期 2024-04-18

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 规划集群与索引

云搜索服务（Cloud Search Service，简称CSS），支持灵活选择集群版本、集群架构、存储机型、集群节点数量、存储容量和索引分片数。您可以根据业务的读写请求、数据存算和搜索与分析等需求进行自由组合。

CSS集群选型主要包括以下内容：

- **集群版本**
- **集群架构**
- **存储机型**
- **集群节点数量**
- **存储容量**
- **索引分片数**

集群版本

CSS中的Elasticsearch搜索引擎目前支持7.6.2和7.10.2版本，版本选择建议如下：

1. 新上线的Elasticsearch集群，建议选择7.10.2或7.6.2版本。
2. 如果有Elasticsearch集群迁移与代码改造的需求，建议优先选择7.10.2或7.6.2版本，否则建议和之前的大版本保持一致。

集群架构

CSS支持读写分离、冷热分离、存算分离、角色分离、跨AZ部署等多种架构。架构适用的场景如下所示：

架构	适用场景	用户价值
读写分离	生产业务，读多写少，数据写入后实时可见性要求低(10s+)。	高并发、低时延
冷热分离	日志业务，冷数据查询性能要求低。	低成本
存算分离	日志业务，冷数据不需要更新，并且冷数据查询性能要求低（10s+）；可以和冷热分离结合使用，构建“热-温-冷”3级存储。	低成本

架构	适用场景	用户价值
角色分离	集群的规模较大、集群中的索引数量较多或集群可扩展性要求高。	高可用
跨AZ部署	对可用性要求非常高的生产业务，或采用本地盘时。	高可用

存储机型

CSS支持云盘和本地盘两种机型。

- 云盘机型包括：计算密集型（CPU：内存=1:2）、通用计算型（CPU：内存=1:4）、内存优化型（CPU：内存=1:8）。
- 本地盘机型包括：磁盘增强型（挂载HDD盘）、超高IO型（挂载SSD盘）。

  各机型适用的场景如下：

表 1-1 存储机型适用场景

机型	适用场景
计算密集型	**推荐场景**：数据量较少（单节点<100GB）的搜索场景。
通用计算型	**通用场景**：用于单节点数据量在100-1000GB间的搜索与分析场景，例如中等规模的电商搜索、社交搜索、日志搜索等场景。
内存优化型	**通用场景**：用于单节点数据量在100-2000GB间的搜索与分析场景 **向量检索场景**：大内存有利于提升集群的性能与稳定性。
磁盘增强型	**日志场景**：用于存储冷数据，冷数据的数据查询性能要求低，并且数据需要更新的场景。
超高IO型-鲲鹏	**大型日志场景：**用于存储热数据。
超高IO型-X86	**大型搜索与分析场景**：场景对计算或磁盘IO均有较高要求，例如舆情分析、专利检索、以及部分数据库加速场景。

集群节点数量

当CSS集群的构架与机型确定后，集群的节点数主要由业务对性能的要求决定。

表 1-2 节点数量计算方式

类型	性能基线	节点数量计算方式	示例
写入节点	对于挂载云盘的节点，其单核写入性能基线为1MB/s。 对于超高IO型的节点，其单核写入性能基线为1.5MB/s。	业务峰值时的流量/单节点的核数/单核写入性能基线*副本数	业务峰值写入100MB/s，使用16u64g的节点，预计需要100/16/1*2 = 12个节点。
查询节点	相同节点，不同业务场景下的性能差异非常大，单节点的性能基线难以评估。这里以业务**平均查询响应时间**作为查询的性能基线进行测算。	QPS/{单节点的核数*3/2/平均查询响应时间(s)}*分片数量	查询QPS要求1000，平均查询响应时间100ms，索引规划3个分片，使用16u64g的节点，预计需要1000/{16*3/2/0.1}*3 = 12个节点。
节点数量	/	节点数量= 写入节点数 + 查询节点数	节点数= 写入节点数 + 查询节点数 = 24个节点数。

在同等集群性能的情况下，建议优先选择高配置少节点的集群。例如32C64G*3 节点的集群相比于8C16G*12节点的集群，在集群稳定性和扩容的便捷性上都有一定的优势。因为高配置的集群如果遇到性能瓶颈需要扩容，则只需要横向扩容，即向集群中加入更多同等配置的节点即可；而低配置的集群在扩容节点配置时，则需要纵向扩容。

节点存储容量

CSS集群中每个节点的磁盘空间由数据量、副本数量（一般建议为1）、数据膨胀率、磁盘空间使用率（一般建议为70%）规划等多个因素决定，可以通过以下公式来计算集群的存储容量：

存储容量 = 源数据 * (1 + 副本数量) * 1.25 *（1 + 预留空间）≈ 源数据 * 2 * 1.25 * 1.3 =源数据*3.25

索引分片数

CSS集群的索引分片数，建议按照以下要求规划：

1. 单个分片大小控制在10~50GB。
2. 集群总分片数量控制在3w以内。
3. 1GB的内存空间放置20~30个分片为佳，单节点建议不超过1000个分片。
4. 对于单个索引，索引分片数建议和节点数保持一致，或者为节点数的倍数。

2 配置权限

2.1 为 IAM 用户（子用户）配置创建 CSS 集群的权限

如果您需要对所拥有的云搜索服务（Cloud Search Service，简称CSS）进行细颗粒度的权限管理，可以使用统一身份认证服务（Identity and Access Management，简称IAM），创建独立的IAM用户（子用户）并给IAM用户组授予策略或角色，便可使用这些策略来控制对CSS资源的访问范围。

本章节为您详细介绍如何创建IAM用户，并将IAM用户添加至用户组中，从而使IAM用户拥有创建CSS集群的权限。

步骤一：创建用户组

步骤1 A公司管理员，使用注册的华为账号开通并登录华为云。

图 2-1 登录华为云

步骤2 在华为云首页，单击右上角的“控制台”。

图 2-2 进入控制台

步骤3 在“控制台”页面，鼠标移动至右上方的用户名，在下拉列表中选择“统一身份认证”。

图 2-3 进入统一身份认证

步骤4 A公司管理员登录控制台，并选择“统一身份认证服务”。

步骤5 在统一身份认证服务，左侧导航窗格中，单击“用户组”>“创建用户组”。

图 2-4 创建用户组

步骤6 在“创建用户组”界面，输入“用户组名称”，单击“确定”，完成用户组创建。

图 2-5 输入用户组信息

----结束

步骤二：给用户组授权

步骤1 在用户组列表中，单击新建用户组右侧的“授权”。

图 2-6 授权

步骤2 在用户组选择策略页面中，在搜索框中搜索“CSS FullAccess”勾选并单击“下一步”。

- 一般创建集群的权限有“CSS FullAccess”和“Elasticsearch Administrator”，可根据表2-1的对应的常用操作和系统权限的关系进行对应的配置设置。表2-2包含了CSS所有的系统权限。
- 如果还需要查看资源的消费情况，请在同区域选择“BSS Administrator”权限。

表 2-1 常用操作与系统权限的关系

操作	CSS FullAccess	CSS ReadOnlyAccess	Elasticsearch Administrator	备注
创建集群	√	x	√	-
查询集群列表	√	√	√	-
查询集群详情	√	√	√	-
删除集群	√	x	√	-
重启集群	√	x	√	-
扩容集群	√	x	√	-
扩容实例的数量和存储容量	√	x	√	-
查询指定集群的标签	√	√	√	-
查询所有标签	√	√	√	-
加载自定义词库	√	x	√	依赖OBS和IAM权限
查询自定义词库状态	√	√	√	-
删除自定义词库	√	x	√	-

操作	CSS FullAccess	CSS ReadOnlyAccess	Elasticsearch Administrator	备注
自动设置集群快照的基础配置	√	x	√	依赖OBS和IAM权限
修改集群快照的基础配置	√	x	√	依赖OBS和IAM权限
设置自动创建快照策略	√	x	√	-
查询集群的自动创建快照策略	√	√	√	-
手动创建快照	√	x	√	-
查询快照列表	√	√	√	-
恢复快照	√	x	√	-
删除快照	√	x	√	-
停用快照功能	√	x	√	-
更改规格	√	x	√	-
缩容集群	√	x	√	-

表 2-2 CSS 系统权限

系统角色/策略名称	类别	权限描述	依赖关系
Elasticsearch Administrator	系统角色	CSS服务的所有执行权限。 该角色有依赖，需要在同项目中勾选依赖的Tenant Guest和Server Administrator角色。	• Tenant Guest：全局级角色，在全局项目中勾选。 • Server Administrator：项目级角色，在同项目中勾选。
CSS FullAccess	系统策略	基于策略授权的CSS服务的所有权限，拥有该权限的用户可以完成基于策略授权的CSS服务的所有执行权限。	无
CSS ReadOnlyAccess	系统策略	CSS服务的只读权限，拥有该权限的用户仅能查看CSS服务数据。	无

步骤3 设置最小授权范围。

此处以仅为“华北-北京四”区域的资源设置权限为例，选择授权范围方案为“指定区域项目资源”，并选择“cn-north-4 [华北-北京四]”区域。

图 2-7 设置最小授权范围

步骤4 单击“确定”，完成用户组授权。

----结束

步骤三：创建 IAM 用户并添加到用户组

步骤1 在统一身份认证服务，左侧导航窗格中，单击“用户”>“创建用户”。

步骤2 配置基本信息。在“创建用户”界面填写“用户信息”和“访问方式”。如需一次创建多个用户，可以单击“添加用户”进行批量创建，每次最多可创建10个用户。

图 2-8 配置用户信息

说明

- 用户可以使用此处设置的用户名、邮件地址或手机号任意一种方式登录华为云云服务平台。
- 当用户忘记密码时，可以通过此处绑定的邮箱或手机自行重置密码，如果用户没有绑定邮箱或手机号码，只能由管理员重置密码。

表 2-3 用户信息

用户信息	说明
用户名	必填。IAM用户登录华为云的用户名，此处以“James”和“Alice”为例。
邮件地址	“凭证类型”选择“密码”，并勾选“首次登录时设置”时必填，选择其他时选填。IAM用户绑定的邮件地址，可作为登录凭证，也可由IAM用户自己绑定。

用户信息	说明
手机号	选填。IAM用户绑定的手机号，可作为IAM用户的登录凭证，也可由IAM用户自己绑定。
描述	选填。记录IAM用户相关信息。

图 2-9 配置访问方式

- 编程访问：为IAM用户启用**访问密钥或密码**，支持用户通过API、CLI、SDK等开发工具访问云服务。
- 管理控制台访问：为IAM用户启用**密码**，支持用户登录管理控制台访问云服务。

  说明

  - 如果IAM用户**仅需登录管理控制台访问云服务**，建议访问方式选择**管理控制台访问**，凭证类型为**密码**。
  - 如果IAM用户**仅需编程访问云服务**，建议访问方式选择**编程访问**，凭证类型为**访问密钥**。
  - 如果IAM用户**需要使用密码作为编程访问的凭证**（部分API要求），建议访问方式选择**编程访问**，凭证类型为**密码**。
  - 如果IAM用户使用部分云服务时，需要在其**控制台验证访问密钥**（由IAM用户输入），建议访问方式选择**编程访问和管理控制台访问**，凭证类型为**密码和访问密钥**。例如IAM用户在控制台使用云数据迁移CDM服务创建数据迁移，需要通过访问密钥进行身份验证。

表 2-4 配置凭证类型和登录保护

凭证类型与登录保护		说明
访问密钥		创建用户完成后即可下载本次创建的所有用户的**访问密钥（AK/SK）**。 **一个用户最多拥有两个访问密钥。**
密码	自定义	自定义用户密码，并选择用户首次登录时是否需要重置密码。 如果您是用户的使用主体，建议您选择该方式，设置自己的登录密码，且无需勾选首次登录时重置密码。
	自动生成	系统自动生成IAM用户的登录密码，创建完用户即可下载excel形式的密码文件。将密码文件提供给用户，用户使用该密码登录。 **仅在创建单个用户时适用。**
	首次登录时设置	系统通过邮件发一次性登录链接给用户，用户登录控制台并设置密码。 如果您不是用户的使用主体，建议选择该方式，同时输入用户的邮件地址和手机，用户通过邮件中的一次性链接登录华为云，自行设置密码。该链接**7天**内有效。
登录保护	开启登录保护（推荐）	开启登录保护后，IAM用户登录时，除了在登录页面输入用户名和密码外（第一次身份验证），还需要在登录验证页面输入验证码（第二次身份验证），该功能是一种安全实践，建议开启登录保护，多次身份认证可以提高账号安全性。 您可以选择通过手机、邮箱、虚拟MFA进行登录验证。
	不开启	创建完成后，如需开启登录保护，请参见：**登录保护**。

步骤3 单击“下一步”，将用户加入到**步骤一：创建用户组**中创建的用户组。

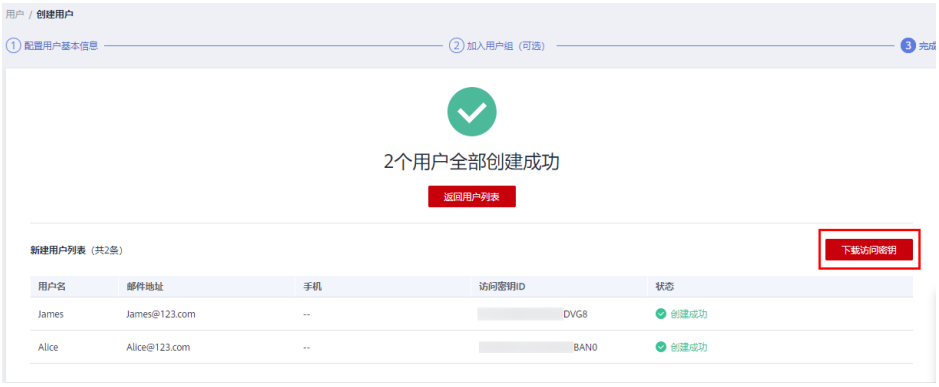
将用户加入用户组，用户将具备用户组的权限，这一过程即给用户授权。

说明

“admin”为系统缺省提供的用户组，具有管理人员以及所有云服务资源的操作权限。

步骤4 单击“创建用户”，IAM用户创建完成，用户列表中显示新创建的IAM用户。如果“访问方式”选择了“编程访问”且**表2 配置凭证类型**凭证类型勾选了“访问密钥”，可在此页面下载访问密钥。

图 2-10 用户创建成功



----结束

步骤四：IAM 用户登录并验证权限

步骤1 在登录页面，单击登录下方的“IAM用户”“子用户登录”，在“IAM用户登录”页面，输入**“租户名/原华为云账号名”**、**“IAM用户名/邮件地址”**和**“密码”**。

图 2-11 IAM 用户登录

- 租户名/原华为云账号名：IAM用户所属的账号。
- IAM用户名/邮件地址：在IAM创建用户时，输入的IAM用户名/邮件地址，例如“James”。如果不知道用户名及初始密码，请向管理员获取。
- IAM用户密码：IAM用户的密码，非账号密码。

步骤2 单击“登录”，登录华为云。

步骤3 在“服务列表”中选择云搜索服务。

步骤4 在云搜索服务总览页面右上角，单击“创建集群”按钮，按照**创建集群**的步骤创建集群，如果创建成功，则表示权限配置成功。

----结束

2.2 新建 Elasticsearch 用户并配置相应索引权限

本章节为您介绍如何在7.6.2版本Elasticsearch安全集群上使用RBAC模型。

背景信息

云搜索服务（Cloud Search Service，简称CSS）用opendistro_security安全插件对外提供安全集群能力，opendistro_security安全插件是基于RBAC（Role-Based Access Control）模型构建。RBAC包括三个重要核心概念：用户（User）、权限（Action）、角色（Role）。RBAC简化了用户和权限的关系，降低了权限管理的难度，方便权限扩展易于维护。三者之前的关系如下图所示：

图 2-12 用户、权限和角色

除了RBAC模型之外，Elasticsearch还有一个重要的概念，叫做Tenant。RBAC能解决各个用户本身授权的问题，Tenant则能解决了不同租户之间的共享信息，通过配置Tenant空间，各个IAM用户（子用户）可以在Tenant空间中共享Dashboard、index_pattern等信息。

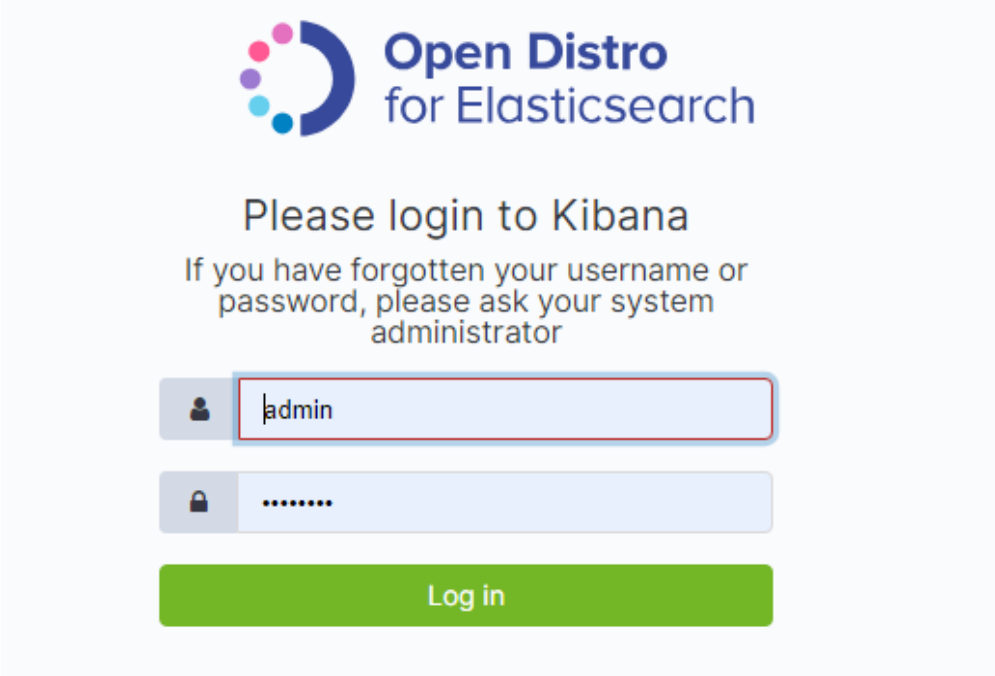
默认情况下，用户只能看见自己Private Tenant空间下的index_pattern、dashboard等信息。新建一个用户“test”，默认会有一个.kibana_xxx_test的索引创建，“test”的private空间的内容会存储在.kibana_xxx_test中。同理，admin账号的private tenant空间内容存储在.kibana_xxx_admin中，如果想要共享当前租户的index_pattern或者Dashboard其他租户，一个是可以创建在global tenant中，其他用户只需要切换到global tenant空间即可。

创建用户并配置相应权限

步骤1 使用Kibana创建用户（User）。

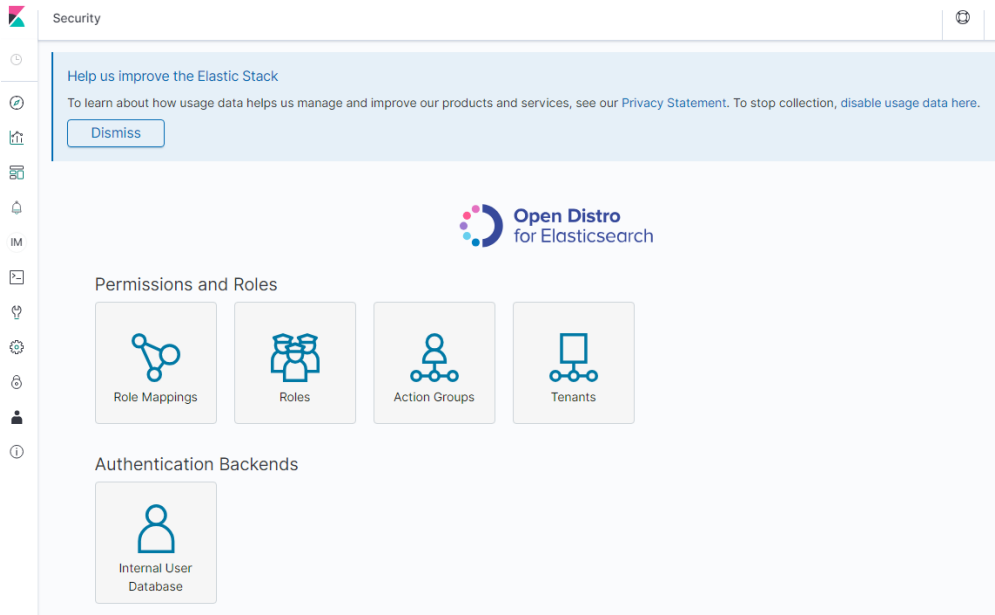
1. 登录云搜索服务控制台。
2. 在集群管理列表，选择对应集群，单击操作列的“Kibana”。
   输入管理员账户名和密码登录Kibana。
   - 账户名：admin（默认管理员账户名）
   - 密码：创建安全模式的集群时，设置的管理员密码。

图 2-13 登录页面



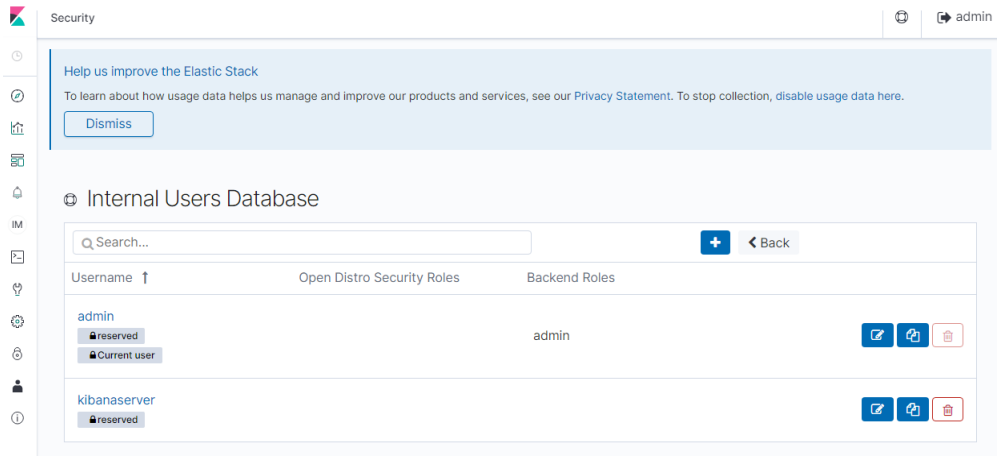
3. 登录成功后，在Kibana操作界面，选择“Security”，进入对应页面。

图 2-14 Security 页面



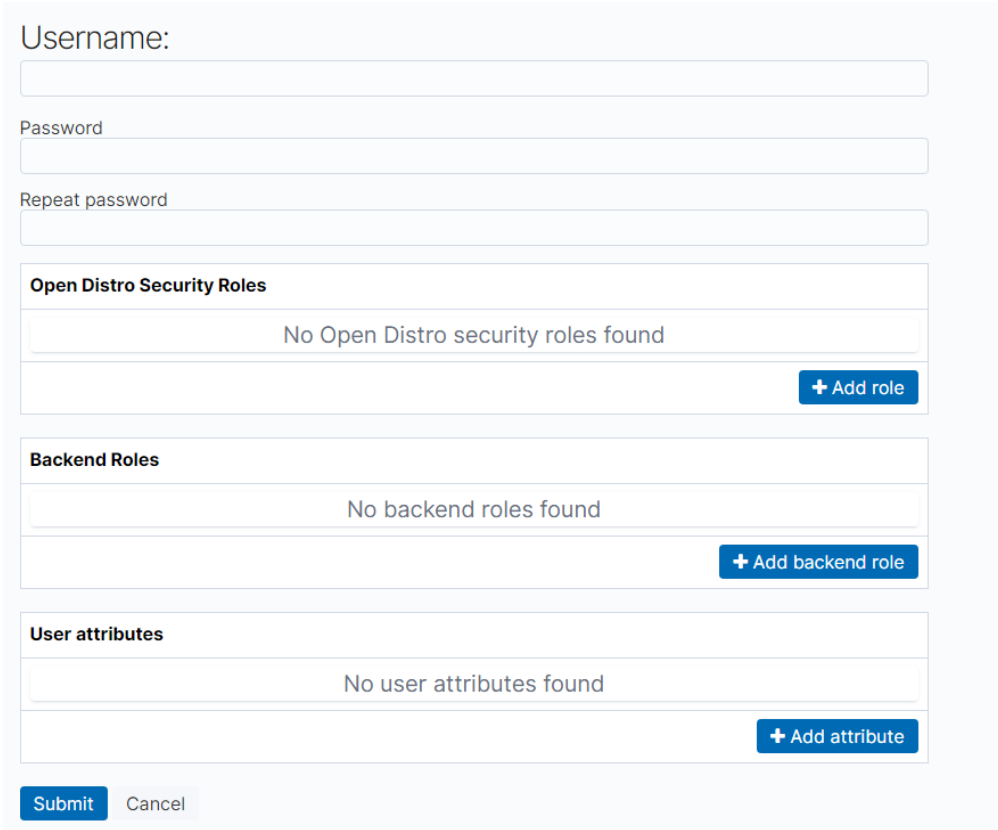
4. 在Security页面，选择“Authentication Backends”>“Internal Users Database”，进入创建用户页面。

图 2-15 创建用户



5. 在“Internal Users Database”页面，选择“+”进入添加用户信息页面。
6. 在创建用户页面，输入“Username”和“Password”，单击“Submit”。本案例已用户名test为示例。

图 2-16 添加用户信息

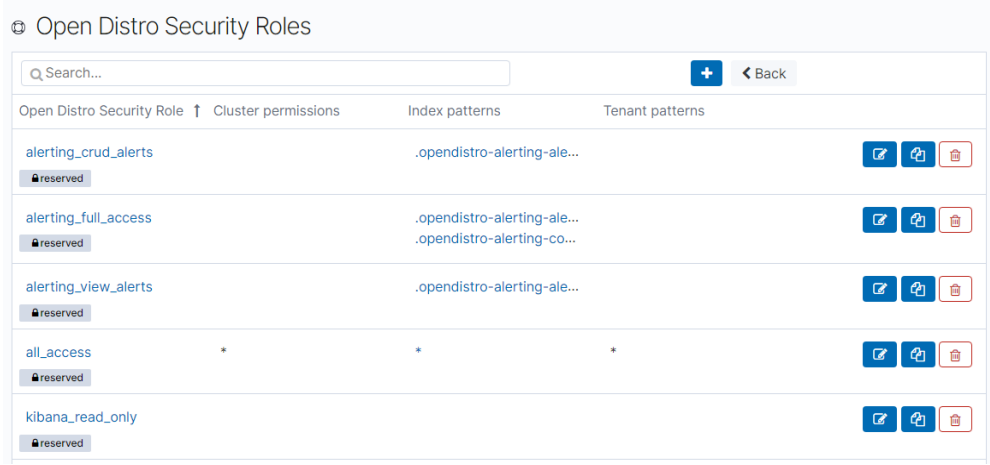


用户创建成功后，可以在列表中看到新创建的用户。

步骤2 创建角色Role，并授予Role相应权限。

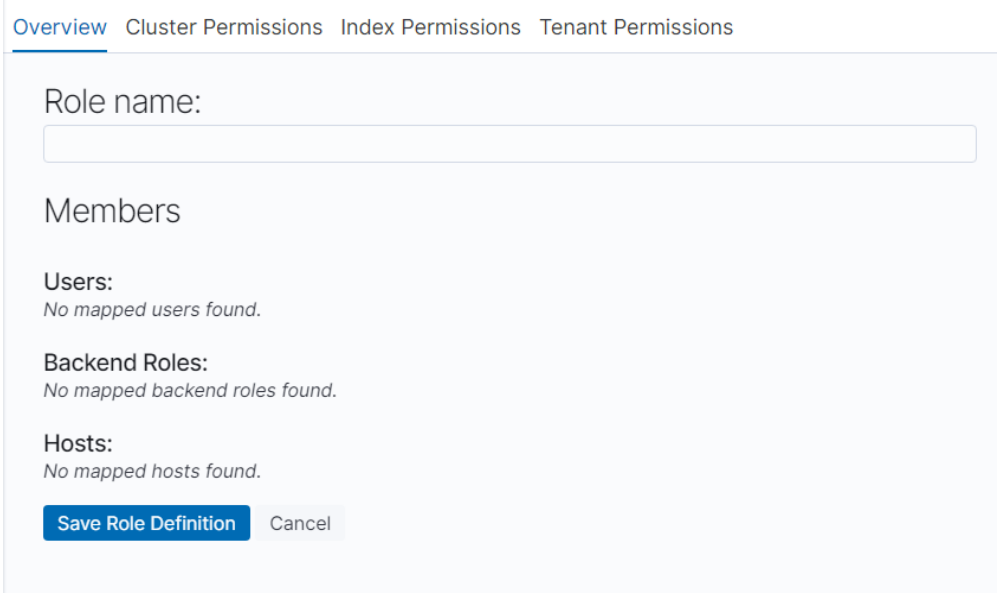
1. 在“Security”中选择“Roles”，进入“Open Distro Security Roles”页面。

图 2-17 Open Distro Security Roles 页面



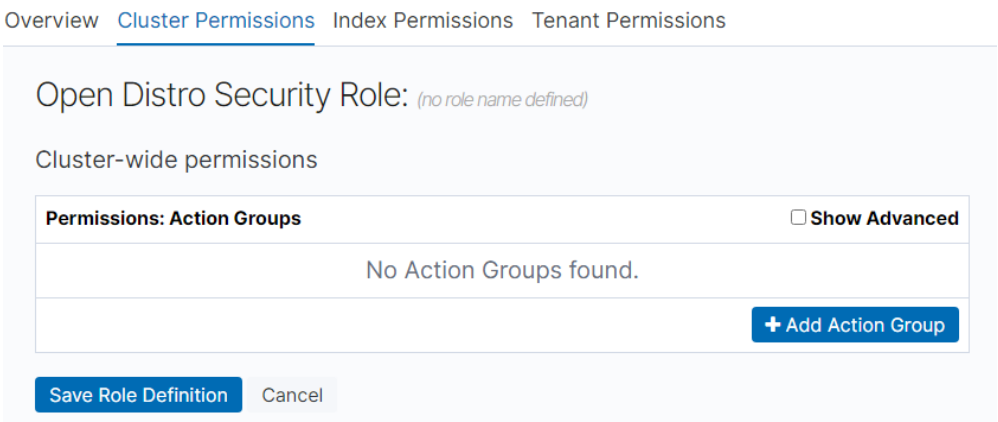
2. 在Open Distro Security Roles页面，单击“+”添加角色权限。
3. 在Overview页面设置角色名。

图 2-18 Overview 页面



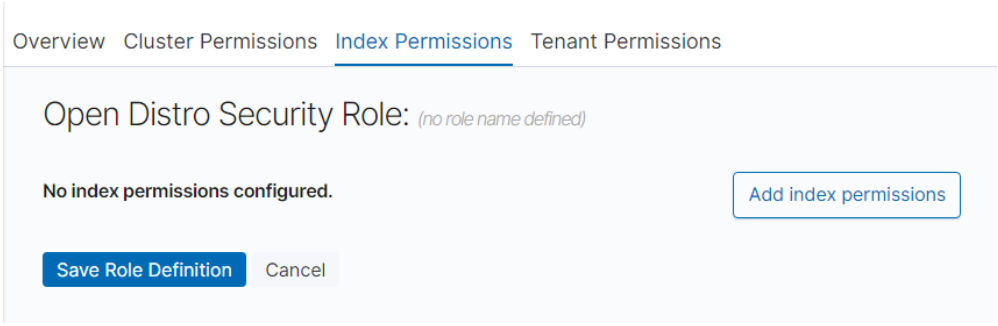
4. 在“Cluster Permissions”页面设置CSS集群权限。

图 2-19 Cluster Permissions 页面



5. 在“Index Permissions”页面，单击“Add index permissions”设置索引权限。

图 2-20 Index Permissions 页面

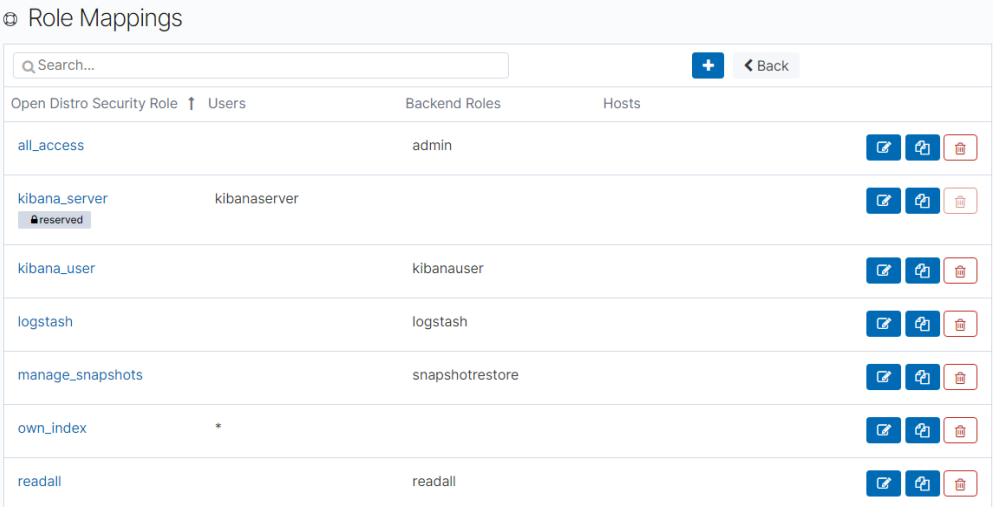


   - Index patterns：配置为需要设置权限的索引名称，例如，索引模板名称为my_store。
   - Permissions: Action Groups根据需要开通的权限设置。例如，只读权限选择Search。
6. 在“Tenant Permissions”页面设置角色权限。
   设置完成后，即可看到设置的角色。

步骤3 角色用户映射，将Role和User绑定。

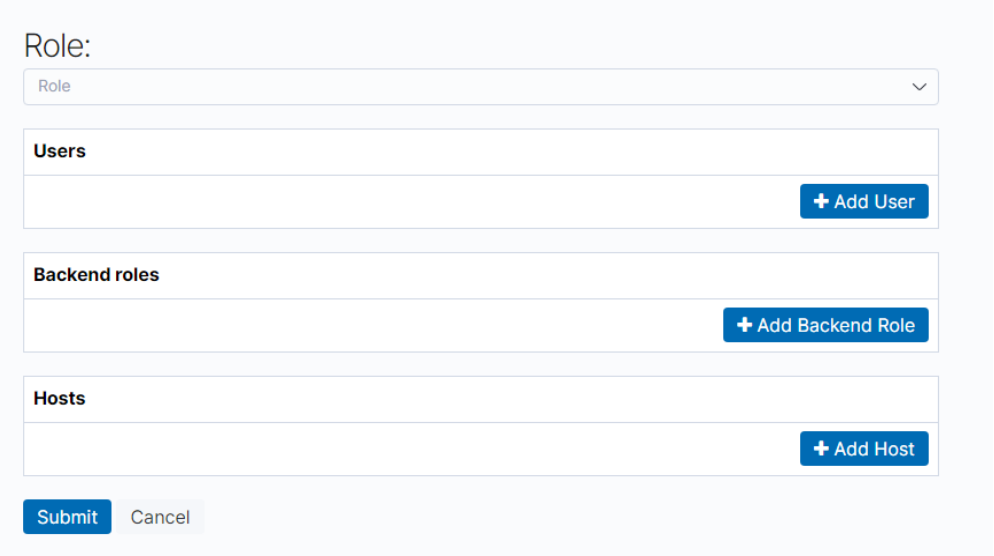
1. 在“Security”中选择“Role Mappings”，进入Role Mappings页面。

图 2-21 Role Mappings 页面



2. 在Role Mappings页面，单击“+”添加用户和角色映射。

图 2-22 添加用户和角色映射



3. 添加完成后，单击“Submit”。
4. 配置完成后，可以在Kibana中进行验证是否生效。

----结束

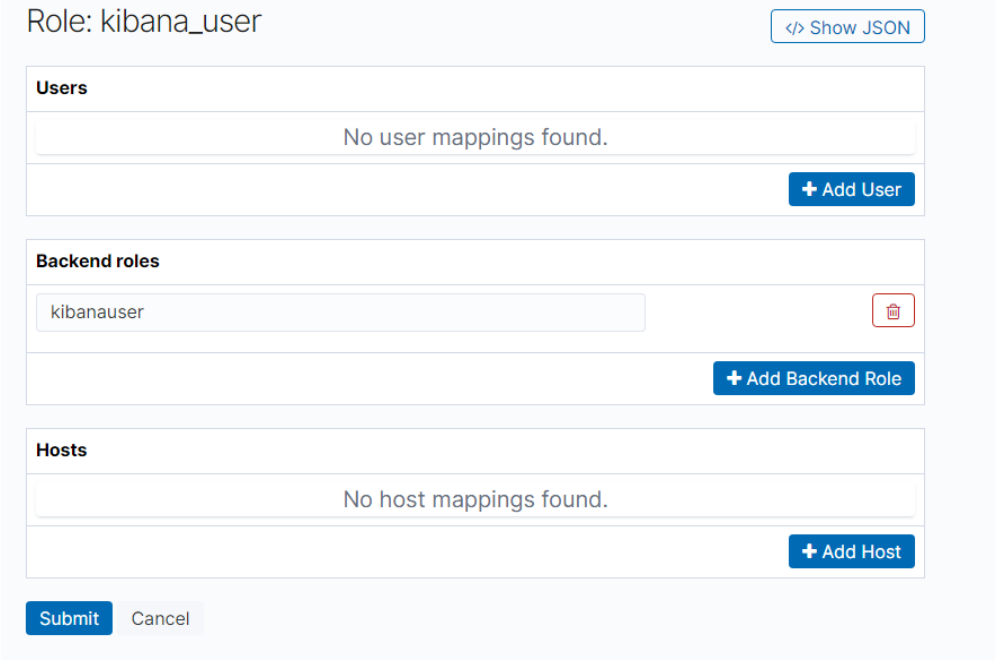
创建一个具有 Kibana 访问权限的用户

步骤1 参考创建用户并配置相应权限中的**使用Kibana创建用户（User）**，创建一个用户名为test的用户。

步骤2 角色用户映射，将Role和User绑定。

1. 在“Security”中选择“Role Mappings”，进入Role Mappings页面。
2. 在Role Mappings页面，单击“kibana_user”角色名称。

图 2-23 kibana_user 角色



kibana_user角色具有.kibana*索引的权限，Kibana界面上面操作的Dashboards以及index_pattern都是保存到.kibana*里面的，将test用户和kibana_user相映射，意味着这个test用户具有了kibana的权限。

3. 单击“+”添加用户和角色映射。
4. 在“Users”区域，选择创建的test用户。
5. 添加完成后，单击“Submit”。

   配置完成后，切换test用户，进行权限验证是否生效。

----结束

给新建用户分配指定索引 index*的权限

新创建的test用户可以访问Kibana，并且可以使用Kibana的index_pattern、Discover、Dashboards等权限，但是这并不意味着test用户能看到任意的.kibana。test用户默认只能看到private tenant和global tenant空间的数据，如果需要访问其他的tenant空间，还需要在role中定义其他的tenant权限。

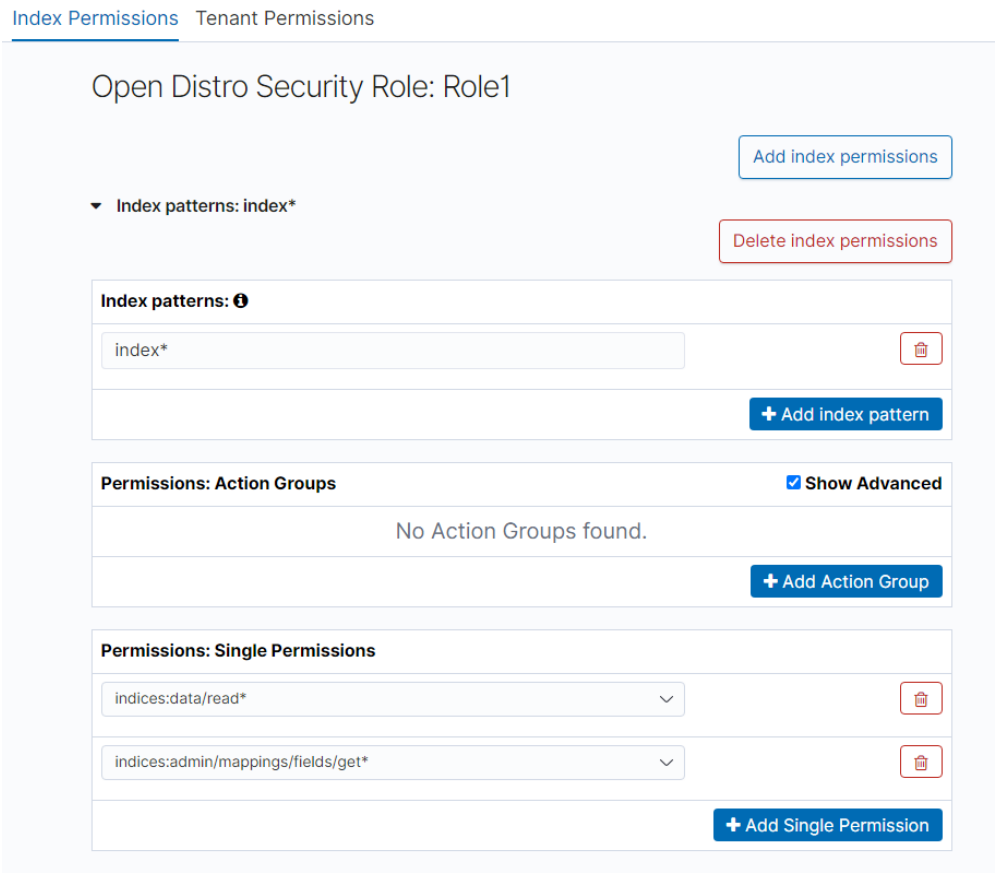
步骤1 在“Security”中选择“Roles”。

步骤2 在“Open Distro Security Roles”页面，单击“+”添加角色权限。

步骤3 在“Overview”页签，设置角色名Role1。

步骤4 在“Index Permissions”页签，单击“Add index permissions”设置索引权限。

图 2-24 设置权限



- “Index patterns”：输入index*。
- “Permissions: Action Groups”：选择对应的权限，如果想要查询，可以选择read。如果想要写入，可以选择write，具体的操作对应的最底层action，可以参考kibana界面的permission模块介绍，以read为例，选择indices:data/read* 和indices:admin/mappings/fields/get*权限。indices:data/read* 包含indices:data/read/下所有的权限，比如indices:data/read/get、indices:data/read/mget、indices:data/read/search等。

步骤5 设置完成后，即可看到设置的角色Role1。

步骤6 在Security中选择“Role Mappings”。

步骤7 在“Role Mappings”页面，单击“+”添加用户test和角色Role1映射。

步骤8 添加完成后，单击“Submit”。

配置完成后，test用户即可拥有index*的read权限。

----结束

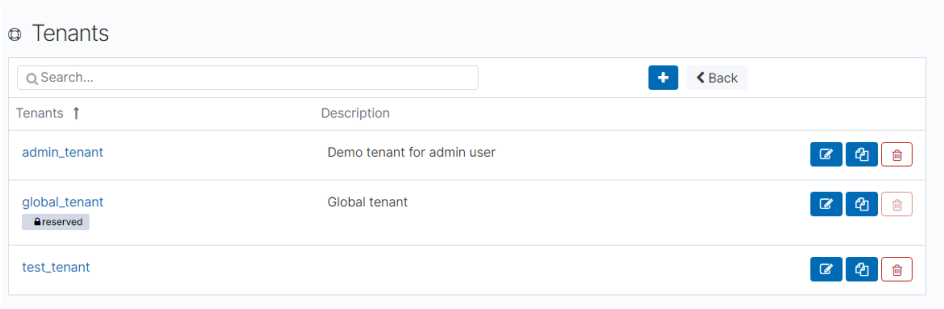
给 test 用户共享 admin 账号的 index_pattern，Dashboards 等信息

一般情况下，新建用户本身并不具备创建index_pattern的权限或者由于业务关系不能够进行数据管理。此时，一般是由admin账号进行index_pattern创建并且管理dashboard等报表信息，然后再将这些信息共享给test用户。

您可以按照以下流程操作：

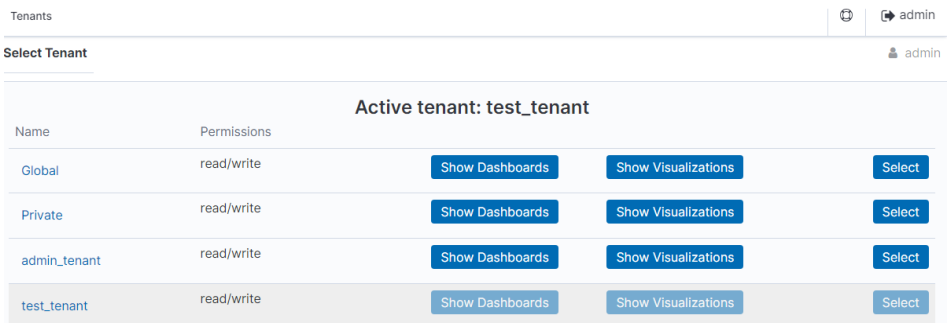
1. 使用admin账户在Global Tenant中创建index_pattern和Dashboards。
2. Global Tenant所有的租户都可以直接访问，但是如果部门过多，这种情况体验较差，可以使用以下方法优化：
   a. 在“Security”中选择“Tenants”。以admin账户建立以部门为粒度的Tenant，比如test_tenant。

   图 2-25 创建 Tenants

   b. 切换到对应部门test_tenant。

   图 2-26 切换 test_tenant

   c. 在test_tenant下创建本部门需要的index_pattern、Dashboards。
   d. 在“Security”中选择“Roles”。单击test用户对应的角色Role1，在Tenant Permissions页签分配test_tenant给Role1。

      保存后切换到test用户，test即可访问test_tenant空间内容。

3 迁移集群

3.1 迁移方案概述

CSS服务的迁移方案适用于华为云ES之间的数据迁移、自建ES到华为云ES之间的数据迁移和第三方友商ES到华为云ES之间的数据迁移。可以根据实际迁移场景，选择合适的集群迁移方案。

迁移场景

不同数据来源的集群，迁移方案会有差别，本章主要介以下场景的迁移方案。

- 源端为Elasticsearch的集群迁移

  Elasticsearch集群的数据迁移有多种方式可以选择，例如使用Logstash、CDM、OBS备份与恢复、ESM、跨集群复制插件等进行数据迁移。

  - Logstash：CSS提供Logstash，可以完成不同数据源和ES数据的迁移，还可以进行数据的清洗和加工。具体操作可以参考**使用云服务Logstash全量迁移集群数据**。
  - CDM：华为云服务提供的云迁移工具，实现不同云服务间的集群迁移能力。具体操作可以参考**Elasticsearch整库迁移到云搜索服务**。
  - 备份与恢复：Elasticsearch提供备份恢复能力，可以把一个集群的数据备份到OBS，在另一个集群恢复数据，完成集群间的数据迁移。CSS Elasticsearch集群之间的数据迁移操作可以参考**使用备份与恢复迁移集群数据（源端为CSS Elasticsearch）**；自建Elasticsearch集群或其他第三方Elasticsearch集群数据迁移到CSS Elasticsearch集群的操作可以参考**使用备份与恢复迁移集群数据（源端为第三方Elasticsearch）**。
- **源端为Kafka/MQ的集群迁移**
- **源端为数据库的集群迁移**

迁移方案

CSS服务支持的迁移方案主要包含备份与恢复、Reindex API、Logstash+ESM和基于数据源同步的迁移方案，各方案的详细信息参见**表3-1**。

其中，基于数据源同步的迁移方案没有明显的限制，且迁移性能方面优于其他3个方案。另外基于数据源同步的迁移方案，在数据同步后随时可割接，更加方便灵活。

表 3-1 迁移方案说明

迁移方案	方案描述	方案限制	迁移性能
备份与恢复	需要有支持s3协议的共享存储，例如OBS桶。先在源端ES集群进行数据备份，再将快照同步到目标集群，最后在目标集群中恢复数据。	• 目的端ES版本≥源端ES版本 • 目的端ES的候选主节点数>源端ES的候选主节点数的一半 • 不支持增量数据同步，需停止更新再进行备份与恢复。	数据迁移速率可配置，理想状态下等于文件复制速率。
Reindex API	先配置源端ES与目的端ES互信，然后通过Reindex API进行数据迁移。	• 索引需开启“_source” • 不支持增量数据实时同步，需停止更新再执行API。	支持批量读写，但不支持slicing并发同步。
Logstash +ESM	先申请ECS部署并配置Logstash，然后启动数据迁移。	• 索引需开启“_source” • 不支持增量数据实时同步，需停止更新再启动Logstash。	支持批量读写，支持slicing并发同步。
基于数据源同步的迁移方案	存量数据采用Logstash迁移，增量数据通过流量复制或数据链路自动同步。	无	存量迁移速率同Logstash，增量迁移复用之前工具。

3.2 源端为 Elasticsearch

3.2.1 使用云服务 Logstash 全量迁移集群数据

Logstash支持全量迁移和增量迁移，首次迁移使用全量迁移，后续增加数据选择增量迁移。本章节介绍如何使用CSS服务的Logstash全量迁移集群数据。

首先请根据**约束和限制**和**准备工作**提前完成迁移前的准备工作。具体步骤如下所示：

- **步骤一：创建Logstash集群**
- **步骤二：验证集群连通性**
- **步骤三：配置Logstash全量迁移任务**
- **步骤四：全量迁移**
- **步骤五：释放Logstash集群**

约束和限制

- Logstash版本约束：

CSS 支持5.5.1，6.3.2，6.5.4，7.1.1，7.6.2，7.10.2多个版本，迁移集群尽量保持大版本一致。

对应ES集群是5.x, 6.x 选择logstash版本5.6.16， 对应ES版本是7.X 选择logstash版本7.10.0。

- 集群迁移过程禁止修改索引，修改索引会导致原数据和目标数据内容不一致。
- 索引大小小于100G可以使用迁移任务不用单独分析索引，简化分析工作。

准备工作

- 创建迁移虚拟机。

  a. 创建迁移虚拟机，用于迁移源集群的元数据。

  i. 创建ECS虚拟机，虚拟机需要创建linux系统，规格选择2U4G。

  ii. 测试虚拟机和源集群和目标集群保持连通性，执行命令**curl http:// {ip}:{port}**可以测试结果。

  IP是源集群和目的集群访问地址，端口默认是9200，如果不是9200使用集群实际端口。

  说明

  如下示例仅适用于非安全集群。

```
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"
}
```

- 准备工具和软件

  在线安装和离线安装以虚拟机是否可以联网判断。如果可以联网直接使用yum和pip安装即可。离线安装需要下载安装包到虚拟机上执行安装命令。

  表 3-2 准备工具和软件

类型	目的	获取位置
Python2	主要用户执行迁移脚本。	**下载地址**，版本选择python 2.7.18。
winscp	Linux上传脚本，跨平台文件传输工具。	**下载Winscp**。

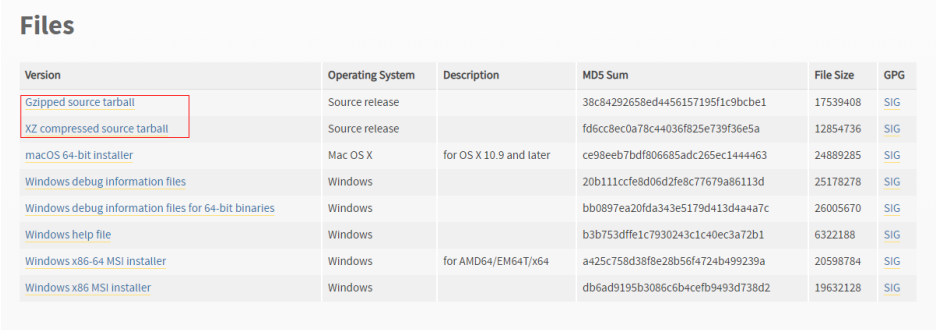
在线安装步骤如下：

a. 执行yum install python2安装python2。
```
[root@ecs opt]# yum install python2
```
b. 执行yum install python-pip安装pip。
```
[root@ecs opt]# yum install python-pip
```
c. 执行pip install pyyaml安装yaml依赖。
d. 执行pip install requests安装requests依赖。

离线安装步骤如下：

a. 下载python2安装包，下载地址**https://www.python.org/downloads/release/python-2718/**。选择源码下载安装。

图 3-1 下载 python2 安装包



b. 使用winscp工具上传python安装包到opt目录下，安装python。
```
# 解压python压缩包
[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
...
# 解压完成进入目录
[root@ecs-52bc opt]# cd Python-2.7.18
# 检查文件配置安装路径
[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
# 编译python
[root@ecs-52bc Python-2.7.18]# make
# 安装python
[root@ecs-52bc Python-2.7.18]# make install
```
c. 安装完成检查，检查python安装结果。
```
# 检查python版本
[root@ecs-52bc Python-2.7.18]# python --version
Python 2.7.5
# 检查pip版本
```



```
[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]#
```

- 准备执行脚本

  a. vi migrateConfig.yaml增加配置文件。

```
es_cluster_new:
# 集群名称
  clustername: es_cluster_new
  # 源端ES集群地址，加上http://,
  src_ip: http://x.x.x.x:9200
  # 没有用户名密码设置为""
  src_username: ""
  src_password: ""
  # 目的端ES集群地址，加上http://
  dest_ip: http://x.x.x.x:9200
  # 没有用户名密码设置为""
  dest_username: ""
  dest_password: ""
  # 可有不定义，默认为false, migrateMapping.py使用
  # 是否只处理这个文件中mapping地址的索引
  # 如果设置成true，则只会将下面的mapping中的索引获取到并在目的端创建
  # 如果设置成false，则会取源端集群的所有索引，除去（.kibana，.*）
  # 并且将索引名称与下面的mapping匹配，如果匹配到使用mapping的value作为目的端的索引名称
  # 如果匹配不到，则使用源端原始的索引名称
  only_mapping: false
  # 要迁移的索引，key为源端的索引名字，value为目的端的索引名字
  mapping:
      test_index_1: test_index_1

  # 可以不定义，默认false, checkIndices.py使用
  # 设置为false会比较所有的索引和文档数量，设置为true只比较索引数量，
  only_compare_index: false
```

  配置文件说明：

配置	说明
clustername	集群名称，集群标识。
src_ip	源集群访问的ip地址，只需要集群一个地址，端口默认是9200，如果集群的访问的端口不是9200，以实际的配置端口为准。
src_username	源集群访问的用户名，如果不需要设置为""。
src_password	源集群访问的密码，如果不需要设置为""。
dest_ip	目标集群访问的ip地址, 只需要集群一个地址，端口默认是9200，如果集群的访问的端口不是9200，以实际的配置端口为准。
dest_username	目标集群访问的用户名，如果不需要设置为""。
dest_password	目标集群访问的密码，如果不需要设置为""。

  b. 执行vi python migrateTemplate.py命令，复制以下脚本到文件生成索引结构迁移脚本。

```
# -*- 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. 执行vi migrateMapping.py命令，复制以下脚本到文件生成索引结构迁移脚本。

```
# -*- 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. 执行vi checkIndices.py命令，复制以下脚本到文件生成索引数据对比脚本。

```
# -*- 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)
```

步骤一：创建 Logstash 集群

说明

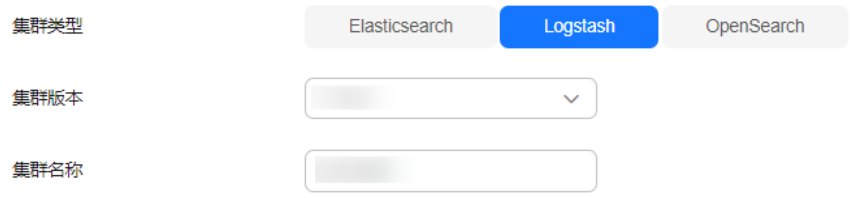
- 迁移数据使用Logstash，创建logstash服务需要费用，默认是按需收费，用户迁移完毕数据及时释放Logstash节省费用。
- 可以基于集群的索引不同创建多个Logstash集群分别配置不同的迁移任务。

1. 登录云搜索服务**管理控制台**。
2. 在“总览”或者“集群管理”页面，选择“Logstash”，进入Logsash类型集群管理页面。
3. 单击“创建集群”，进入“创建集群”页面。
4. 选择“当前区域”和“可用区”。
5. 指定集群基本信息，选择“集群类型”和“集群版本”，并输入“集群名称”。

表 3-3 基本参数说明

参数	说明
集群类型	选择“Logstash”。
集群版本	当前支持5.6.16、7.10.0。 对应ES集群是5.x, 6.x 选择logstash版本5.6.16， 对应ES版本是7.X 选择logstash版本7.10.0。
集群名称	自定义的集群名称，可输入的字符范围为4～32个字符，只能包含数字、字母、中划线和下划线，且必须以字母开头。

图 3-2 基本信息配置



6. 指定集群的主机规格相关参数。“节点数量”设置为“1”。“节点规格”选择“8U16G”，其余参数保持默认值。

图 3-3 设置主机规格

7. 设置集群的企业项目，保持默认值即可。
8. 单击“下一步，网络配置”，设置集群的网络配置。

表 3-4 参数说明

参数	说明
虚拟私有云	VPC即虚拟私有云，是通过逻辑方式进行网络隔离，提供安全、隔离的网络环境。 选择创建集群需要的VPC，单击“查看虚拟私有云”进入VPC服务查看已创建的VPC名称和ID。如果没有VPC，需要创建一个新的VPC。 **说明** 此处您选择的VPC必须包含网段（CIDR），否则集群将无法创建成功。新建的VPC默认包含网段（CIDR）。
子网	通过子网提供与其他网络隔离的、可以独享的网络资源，以提高网络安全。 选择创建集群需要的子网，可进入VPC服务查看VPC下已创建的子网名称和ID。
安全组	安全组是一个逻辑上的分组，为同一个VPC内具有相同安全保护需求并相互信任的弹性云服务器提供访问策略。单击“查看安全组”可了解安全组详情。 **说明** 请确保安全组的“端口范围/ICMP类型”为“Any”或者包含端口9200的端口范围。

图 3-4 设置网络规格

9. 单击“下一步：高级配置”，高级配置可选择默认配置和自定义。此样例保持默认配置即可。

10. 单击“下一步：确认配置”，确认完成后单击“立即创建”开始创建集群。
11. 单击“返回集群列表”，系统将跳转到“集群管理”页面。您创建的集群将展现在集群列表中，且集群状态为“创建中”，创建成功后集群状态会变为“可用”。

步骤二：验证集群连通性

验证Logstash和源集群、目标集群的连通性。

1. 在Logstash集群管理页面，单击**步骤一：创建Logstash集群**中创建的集群名称，进入集群的基本信息页面。选择“配置中心”，进入配置中心页面；或者直接单击目标集群操作列的“配置中心”，进入配置中心页面。
2. 在配置中心页面，选择“连通性测试”。
3. 输入源集群和目的集群的IP地址或域名和端口号，单击“测试”。

图 3-5 连通性测试

步骤三：配置 Logstash 全量迁移任务

1. 在Logstash集群管理页面，单击**步骤一：创建Logstash集群**中创建的集群名称，进入集群的基本信息页面。选择“配置中心”，或者直接单击目标集群操作列的“配置中心”，进入配置中心页面。
2. 单击右上角“创建”，进入创建配置文件页面，选择集群模板，修改ES集群迁移配置文件。

说明

当前案例选用的两个ES集群均未开启https。

- 选择集群模板：此样例是从Elasticsearch类型集群导入数据到Elasticsearch类型，集群模板选择“elasticsearch”，单击操作列的“应用”。基于不同的集群配置增加配置信息。
- 修改配置文件：“名称”填写配置名称，例如es-es-all；“配置文件内容”填写ES集群迁移配置文件，配置文件示例如下。

```
input{
    elasticsearch{
      # 源集群访问地址信息
      hosts =>  ["xx.xx.xx.xx:9200", "xx.xx.xx.xx:9200"]
      # 源集群访问的用户名和密码，没有user和password不填
      # user => "css_logstash"
      # password => "******"
       # 待迁移的索引信息
      index => "*_202102"
      docinfo => true
      slices => 3
      size => 3000
    }
}
```

```
# 移除一些logstash增加的字段
 filter {
   mutate {
     remove_field => ["@metadata", "@version"]
   }
 }

 output{
     elasticsearch{
       # 目标集群访问地址信息
       hosts => ["xx.xx.xx.xx:9200","xx.xx.xx.xx:9200"]
       # 目标集群访问的用户名和密码，没有user和password不填
       # user => "css_logstash"
       # password => "*****"
       index => "%{[@metadata][_index]}"
       document_type => "%{[@metadata][_type]}"
       document_id => "%{[@metadata][_id]}"
     }
 }
```

需要修改的配置如下：

表 3-5 集群配置修改说明

配置		说明
input	hosts	源集群的访问地址，如果集群有多个访问节点请分别填写，使用逗号隔开
	user	集群访问的用户名，如果没有用户名此项使用#注释掉
	password	集群访问的密码，如果没有用户名密码此项使用#注释掉
	index	需要全量迁移的源端索引信息，使用逗号隔开，可以使用通配符设置，index*
	docinfo	是否重新索引文档，必须为true
	slices	配置该参数可以使用切片滚动同时对查询的不同切片，提高整体吞吐量。建议在2-8内
	size	每次查询返回的最大命中数
output	hosts	华为云CSS集群访问地址，如果集群有多个节点，请分别填写，使用逗号隔开
	user	集群访问的用户名，如果没有用户名此项使用 #注释掉
	password	集群访问的密码，如果没有密码此项使用 # 注释掉
	index	迁移到目的端的索引名称，支持扩展修改，如：logstash-%{+yyyy.MM.dd}
	document_type	使目标端文档类型与源端保持一致
	document_id	索引中的文档ID，建议与源端保持一致，如需要自动生成，使用 #注释掉即可

步骤四：全量迁移

步骤1 使用putty登录**准备工作**中创建的Linux虚拟机。

步骤2 执行**python migrateTemplate.py** 迁移索引模板。

步骤3 执行 **python migrateMapping.py**迁移索引。

步骤4 在Logstash集群管理页面，单击**步骤一：创建Logstash集群**中创建的集群名称，进入集群的基本信息页面。选择“配置中心”，或者直接单击目标集群操作列的“配置中心”，进入配置中心页面。

步骤5 选择**步骤三：配置Logstash全量迁移任务**中所创建的配置文件，单击左上角的“启动”。

步骤6 根据界面提示，选择是否启动Logstash服务会立刻开始迁移数据。

步骤7 如果选择“是”，则可以在管道下面看到启动的配置文件。

步骤8 数据迁移完毕检查数据一致性，使用putty登录linux虚拟机，执行**python checkIndices.py** 对比数据结果。

----结束

步骤五：释放 Logstash 集群

集群迁移完毕，释放Logstash集群。

1. 登录云搜索服务管理控制台。
2. 单击“集群管理”>“Logstash”，进入Logstash类型的集群列表界面，在所创建的迁移集群“操作”列中单击“更多>删除”。
3. 在弹出的确认提示框中，再次输入需要删除的集群名称，单击“确定”完成集群删除。

3.2.2 使用云服务 Logstash 增量迁移集群数据

Logstash支持全量迁移和增量迁移，首次迁移使用全量迁移，后续增加数据选择增量迁移。本章节介绍如何使用CSS的Logstash增量迁移集群数据，增量迁移需要索引有时间戳标志，用于识别增量数据。

首先请根据**约束和限制**和**准备工作**提前完成迁移前的准备工作。具体步骤如下所示：

- **步骤一：创建Logstash集群**
- **步骤二：验证集群连通性**
- **步骤三：配置Logstash增量迁移任务**
- **步骤四：增量迁移**
- **步骤五：释放Logstash集群**

约束和限制

- Logstash版本约束：

  CSS 支持5.5.1，6.3.2，6.5.4，7.1.1，7.6.2，7.10.2多个版本，迁移集群尽量保持大版本一致。

对应ES集群是5.x, 6.x 选择logstash版本5.6.16， 对应ES版本是7.X 选择logstash版本7.10.0。

- 集群迁移过程禁止修改索引，修改索引会导致原数据和目标数据内容不一致。
- 索引大小小于100G可以使用迁移任务不用单独分析索引，简化分析工作。

准备工作

- 创建迁移虚拟机。

  a. 创建迁移虚拟机，用于迁移源集群的元数据。

  i. 创建ECS虚拟机，虚拟机需要创建linux系统，规格选择2U4G。

  ii. 测试虚拟机和源集群和目标集群保持连通性，执行命令**curl http://{ip}:{port}**可以测试结果。

  IP是源集群和目的集群访问地址，端口默认是9200，如果不是9200使用集群实际端口。

  说明

  如下示例仅适用于非安全集群。

```
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"
}
```

- 准备工具和软件

  在线安装和离线安装以虚拟机是否可以联网判断。如果可以联网直接使用yum和pip安装即可。离线安装需要下载安装包到虚拟机上执行安装命令。

  在线安装步骤如下：

  a. 执行yum install python2安装python2。

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

  b. 执行yum install python-pip安装pip。

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

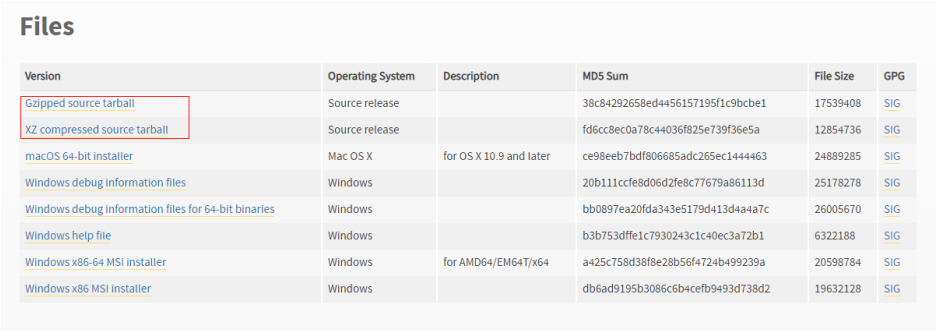
  c. 执行pip install pyyaml安装yaml依赖。

  d. 执行pip install requests安装requests依赖。

  离线安装步骤如下：

  a. 下载python2安装包，下载地址**https://www.python.org/downloads/release/python-2718/**。选择源码下载安装。

图 3-6 下载 python2 安装包



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. 使用winscp工具上传python安装包到opt目录下，安装python。

```
# 解压python压缩包
[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
...
# 解压完成进入目录
[root@ecs-52bc opt]# cd Python-2.7.18
# 检查文件配置安装路径
[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
# 编译python
[root@ecs-52bc Python-2.7.18]# make
# 安装python
[root@ecs-52bc Python-2.7.18]# make install
```

c. 安装完成检查，检查python安装结果。

```
# 检查python版本
[root@ecs-52bc Python-2.7.18]# python --version
Python 2.7.5
# 检查pip版本
[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]#
```

- 准备执行脚本

a. vi migrateConfig.yaml增加配置文件。

```
es_cluster_new:
# 集群名称
  clustername: es_cluster_new
  # 源端ES集群地址，加上http://,
  src_ip: http://x.x.x.x:9200
  # 没有用户名密码设置为""
  src_username: ""
  src_password: ""
  # 目的端ES集群地址，加上http://
```



```
dest_ip: http://x.x.x.x:9200
# 没有用户名密码设置为""
dest_username: ""
dest_password: ""
# 可有不定义，默认为false, migrateMapping.py使用
# 是否只处理这个文件中mapping地址的索引
# 如果设置成true，则只会将下面的mapping中的索引获取到并在目的端创建
# 如果设置成false，则会取源端集群的所有索引，除去（.kibana，.*）
# 并且将索引名称与下面的mapping匹配，如果匹配到使用mapping的value作为目的端的索引名称
# 如果匹配不到，则使用源端原始的索引名称
only_mapping: false
# 要迁移的索引，key为源端的索引名字，value为目的端的索引名字
mapping:
    test_index_1: test_index_1

# 可以不定义，默认false, checkIndices.py使用
# 设置为false会比较所有的索引和文档数量，设置为true只比较索引数量，
only_compare_index: false
```

配置文件说明：

配置	说明
clustername	集群名称，集群标识。
src_ip	源集群访问的ip地址，只需要集群一个地址，端口默认是9200，如果集群的访问的端口不是9200，以实际的配置端口为准。
src_username	源集群访问的用户名，如果不需要设置为""。
src_password	源集群访问的密码，如果不需要设置为""。
dest_ip	目标集群访问的ip地址, 只需要集群一个地址，端口默认是9200，如果集群的访问的端口不是9200，以实际的配置端口为准。
dest_username	目标集群访问的用户名，如果不需要设置为""。
dest_password	目标集群访问的密码，如果不需要设置为""。

b. 执行vi checkIndices.py命令，复制以下脚本到文件生成索引数据对比脚本。

```
# -*- 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)
```

步骤一：创建 Logstash 集群

说明

- 迁移数据使用Logstash，创建logstash服务需要费用，默认是按需收费，用户迁移完毕数据及时释放Logstash节省费用。
- 可以基于集群的索引不同创建多个Logstash集群分别配置不同的迁移任务。

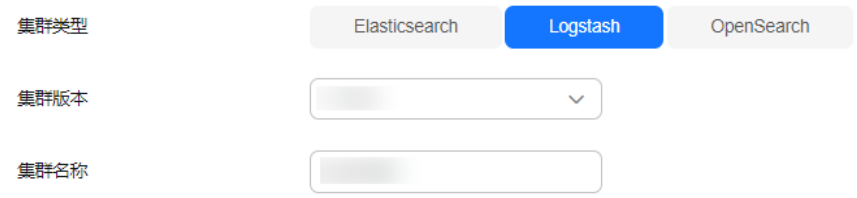
1. 登录云搜索服务**管理控制台**。
2. 在“总览”或者“集群管理”页面，选择“Logstash”，进入Logsash类型集群管理页面。
3. 单击“创建集群”，进入“创建集群”页面。
4. 选择“当前区域”和“可用区”。
5. 指定集群基本信息，选择“集群类型”和“集群版本”，并输入“集群名称”。

表 3-6 基本参数说明

参数	说明
集群类型	选择“Logstash”。

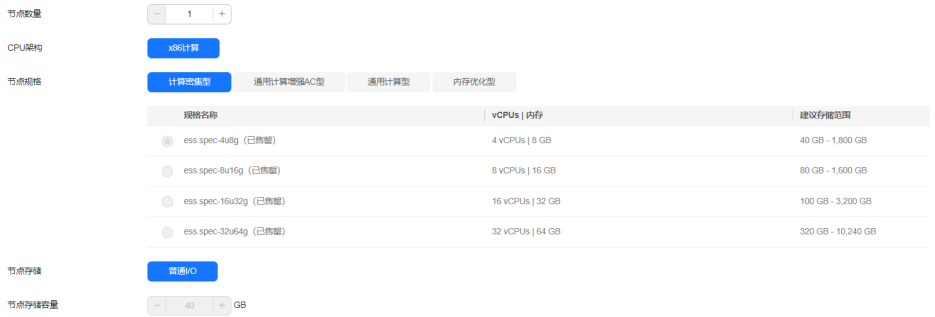
参数	说明
集群版本	当前支持5.6.16、7.10.0。 对应ES集群是5.x, 6.x 选择logstash版本5.6.16， 对应ES版本是7.X 选择logstash版本7.10.0。
集群名称	自定义的集群名称，可输入的字符范围为4～32个字符，只能包含数字、字母、中划线和下划线，且必须以字母开头。

图 3-7 基本信息配置



6. 指定集群的主机规格相关参数。“节点数量”设置为“1”。“节点规格”选择“8U16G”，其余参数保持默认值。

图 3-8 设置主机规格



7. 设置集群的企业项目，保持默认值即可。
8. 单击“下一步，网络配置”，设置集群的网络配置。

表 3-7 参数说明

参数	说明
虚拟私有云	VPC即虚拟私有云，是通过逻辑方式进行网络隔离，提供安全、隔离的网络环境。 选择创建集群需要的VPC，单击“查看虚拟私有云”进入VPC服务查看已创建的VPC名称和ID。如果没有VPC，需要创建一个新的VPC。 **说明** 此处您选择的VPC必须包含网段（CIDR），否则集群将无法创建成功。新建的VPC默认包含网段（CIDR）。

参数	说明
子网	通过子网提供与其他网络隔离的、可以独享的网络资源，以提高网络安全。 选择创建集群需要的子网，可进入VPC服务查看VPC下已创建的子网名称和ID。
安全组	安全组是一个逻辑上的分组，为同一个VPC内具有相同安全保护需求并相互信任的弹性云服务器提供访问策略。单击“查看安全组”可了解安全组详情。 **说明** 请确保安全组的“端口范围/ICMP类型”为“Any”或者包含端口9200的端口范围。

图 3-9 设置网络规格

9. 单击“下一步：高级配置”，高级配置可选择默认配置和自定义。此样例保持默认配置即可。
10. 单击“下一步：确认配置”，确认完成后单击“立即创建”开始创建集群。
11. 单击“返回集群列表”，系统将跳转到“集群管理”页面。您创建的集群将展现在集群列表中，且集群状态为“创建中”，创建成功后集群状态会变为“可用”。

步骤二：验证集群连通性

验证Logstash和源集群、目标集群的连通性。

1. 在Logstash集群管理页面，单击**步骤一：创建Logstash集群**中创建的集群名称，进入集群的基本信息页面。选择“配置中心”，进入配置中心页面；或者直接单击目标集群操作列的“配置中心”，进入配置中心页面。
2. 在配置中心页面，选择“连通性测试”。
3. 输入源集群和目的集群的IP地址或域名和端口号，单击“测试”。

图 3-10 连通性测试

步骤三：配置 Logstash 增量迁移任务

1. 在Logstash集群管理页面，单击**步骤一：创建Logstash集群**中创建的集群名称，进入集群的基本信息页面。选择“配置中心”，或者直接单击目标集群操作列的“配置中心”，进入配置中心页面。
2. 单击右上角“创建”，进入创建配置文件页面，选择集群模板，修改ES集群迁移配置文件。

   当前案例选用的两个ES集群均未开启https。

   - 选择集群模板：此样例是从Elasticsearch类型集群导入数据到Elasticsearch类型，集群模板选择“elasticsearch”，单击操作列的“应用”。基于不同的集群配置增加配置信息。
   - 修改配置文件：“名称”填写配置名称，例如es-es-inc；“配置文件内容”填写ES集群迁移配置文件，配置文件示例如下。

```
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 }}
}
```

   需要修改的配置如下：

表 3-8 集群配置修改说明

配置	说明
hosts	分别填写源集群和目标集群的访问地址，如果集群有多个访问节点请分别填写。
user	集群访问的用户名，如果没有用户名此项使用#注释掉。
password	集群访问的密码，如果没有用户名密码此项使用#注释掉。

配置	说明
index	需要增加迁移的索引信息，一个配置文件只支持一个索引的增量迁移。
query	增量数据的识别标识，一般是es的dls语句，需要提前分析。其中postsDate为业务中时间字段。 {"query":{"bool":{"should":[{"range":{"postsDate":{"from":"2021-05-25 00:00:00"}}}]}}} 此处命令是迁移2021-05-25之后新增加的数据，在多次增量迁移过程中需要修改此处的日志值，如果日期是时间戳方式请转换为时间戳。此处命令需要提前验证有效性。
scroll	当源端数据量过大，为了防止logstash内存溢出，可以使用scroll分批次获取数据。默认为"1m"。间隔时间不要太长，否则可能会丢失数据。

说明

不同的索引的增量迁移配置不同，必须基于索引分析给出增量配置文件迁移命令。

步骤四：增量迁移

步骤1 使用putty登录**准备工作**中创建的Linux虚拟机。

步骤2 在Logstash集群管理页面，单击**步骤一：创建Logstash集群**中创建的集群名称，进入集群的基本信息页面。选择“配置中心”，或者直接单击目标集群操作列的“配置中心”，进入配置中心页面。

步骤3 选择**步骤三：配置Logstash增量迁移任务**中所创建的配置文件，单击左上角的“启动”。

步骤4 根据界面提示，选择“是”启动Logstash服务会立刻开始迁移数据。

步骤5 此时可以在管道下面看到启动的配置文件。

步骤6 数据迁移完毕检查数据一致性，使用putty登录linux虚拟机，执行**python checkIndices.py** 对比数据结果。

----结束

步骤五：释放 Logstash 集群

集群迁移完毕，释放Logstash集群。

1. 登录云搜索服务管理控制台。
2. 单击“集群管理”>“Logstash”，进入Logstash类型的集群列表界面，在所创建的迁移集群“操作”列中单击“更多>删除”。
3. 在弹出的确认提示框中，再次输入需要删除的集群名称，单击“确定”完成集群删除。

3.2.3 使用备份与恢复迁移集群数据（源端为 CSS Elasticsearch）

方案概述

CSS服务的Elasticsearch集群之间的数据迁移，可以通过备份与恢复集群快照功能实现。

适用场景：

- 集群升级：将低版本的集群数据迁移到高版本的集群中。
- 集群合并：将两个集群的索引数据合并到一个集群中。

本案例通过将Elasticsearch集群“Es-1”迁移到“Es-2”为例，介绍如何使用集群快照功能实现集群的备份与恢复。

迁移时长

迁移过程的耗时长短依赖于源集群和目的集群的节点个数或索引shard个数。迁移过程分为备份阶段和恢复阶段，备份阶段耗时由源集群决定，恢复阶段耗时由目的集群决定。迁移总时长的评估公式如下：

- 当索引shard个数大于节点个数时

  $$总时长(s)=(800G\div40MB\div源集群节点个数+800G\div40MB\div目的集群节点个数)\times索引个数$$

- 当索引shard个数小于节点个数时

  $$总时长(s)=(800G\div40MB\div源集群索引shard个数+800G\div40MB\div目的集群索引shard个数)\times索引个数$$

说明

评估公式是基于理想状态下（即单节点以最快速度40MB/s传输）的迁移时长，实际迁移时长还会受到网络、资源等因素影响。

前提条件

- 目的端集群（Es-2）和源端集群（Es-1）处于可用状态。建议在业务空闲期进行集群迁移。
- 确认目的端集群（Es-2）和源端集群（Es-1）在同一个Region下。
- 确认目的端集群（Es-2）的版本大于等于源端集群（Es-1）的版本。
- 确认目的端集群（Es-2）的节点数大于源端集群（Es-1）的节点数的一半。
- 确认目的端集群（Es-2）的节点数大于等于源端集群（Es-1）的shard副本数。
- 确认目的端集群（Es-2）的CPU、MEM和Disk配置大于等于源端集群（Es-1）。

操作步骤

1. 登录云搜索服务管理控制台。
2. 在“集群管理 > Elasticsearch”页面，单击源端集群名称“Es-1”进入集群基本信息页面。
3. 在左侧导航栏选择“集群快照”，打开集群快照开关，设置快照的基础配置。

表 3-9 集群快照基础配置

参数	说明
OBS桶	选择存储集群快照的OBS桶。
备份路径	集群快照在OBS桶中的存放路径。可以保持默认值。
IAM委托	选择IAM委托，授权CSS服务访问或维护存储在OBS中数据。 IAM委托需要具备“全局服务”中“对象存储服务”项目的“OBS Administrator”权限。

4. 完成基础配置后，单击“创建快照”，在弹窗中完成参数配置，单击“确定”启动手动创建快照。

表 3-10 创建快照的配置

参数	说明
快照名称	自定义快照名称，可以保持默认值。
索引	填写需要进行备份的索引名称。索引名称不能包含空格和大写字母，且不能包含"\<|>/?特殊字符，多个索引之间使用英文逗号隔开。如果不填写，则默认备份集群中所有索引。支持使用“*”匹配多个索引，例如：index*，表示备份名称前缀是index的所有索引的数据。
快照描述	描述快照信息。

在快照管理列表中，当“快照状态”为“可用”时表示快照创建成功。

5. 快照创建成功后，在快照管理列表，单击快照操作列的“恢复”，配置恢复参数将数据恢复至目的端集群“Es-2”。

表 3-11 恢复快照的配置

参数	说明
索引	填写需要进行恢复的索引名称。如果不填写，则表示恢复所有的索引数据。支持使用“*”匹配多个索引，比如index*，表示恢复快照中名称前缀是index的所有索引。
索引名称匹配模式	索引名称匹配规则。“索引名称匹配模式”和“索引名称替换模式”必须同时设置才会生效。通过配置这两参数，可对快照中匹配到的索引进行重命名。
索引名称替换模式	索引名称重命名规则。设置“索引名称替换模式”参数时，“索引名称匹配模式”参数和该参数必须同时设置才能生效。 默认值“restored_index_$1”表示在所有恢复的索引名称前面加上“restored_”。

参数	说明
集群	选择要恢复快照的目的端集群，本案例选择“Es-2”。 **须知** 如果目的端集群中存在和源端集群同名的索引，则恢复完成后，目的端集群中的同名索引数据将会被覆盖。

在快照管理列表中，当“任务状态”变更为“恢复成功”时表示源端集群“Es-1”中的数据成功迁移到目的端集群“Es-2”。

3.2.4 使用备份与恢复迁移集群数据（源端为第三方 Elasticsearch）

自建ES到华为云ES之间的数据迁移和第三方友商ES到华为云ES之间的数据迁移，可以参考本章操作指导。

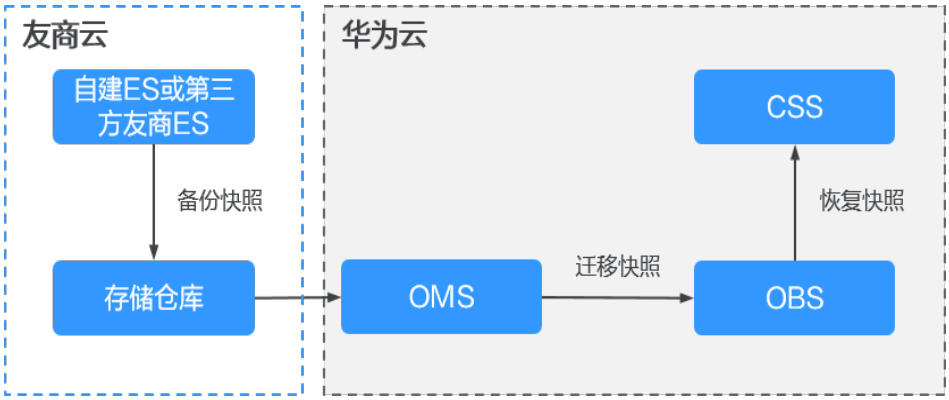
前提条件

- 使用备份与恢复时，需确认如下2点：
  - 目的端ES版本≥源端ES版本
  - 目的端ES的候选主节点数>源端ES的候选主节点数的一半
- 备份与恢复不支持增量数据同步，需停止数据更新后再进行备份。
- CSS服务中已创建好目的端Elasticsearch集群。

迁移流程

当源端是自建ES或第三方友商ES，目的端是CSS服务的ES集群时，集群迁移流程如图3-11所示。

图 3-11 使用备份与恢复迁移的集群迁移流程



操作步骤

步骤1 登录Elasticsearch所在的第三方友商云，创建一个支持s3协议的共享存储仓库，例如登录阿里云的进入OSS服务创建目录patent-esbak， 或者登录腾讯云进入COS服务创建目录patent-esbak。

步骤2 在自建或第三方友商Elasticsearch中创建快照备份仓库，用于存放ES快照数据。

例如，在Elasticsearch中创建一个“my_backup”的备份仓库，关联到存储仓库OSS。

```
PUT _snapshot/my_backup
  {
     # 存储仓库类型。
   "type": "oss",
     "settings": {
     # 步骤1中存储仓库的内网访问域名。
     "endpoint": "http://oss-xxx.xxx.com",
     # 存储仓库的用户ID和密码。认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密, 确保安全；本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量ak和sk。
     "access_key_id": "ak",
     "secret_access_key": "sk",
     # 步骤1创建的存储仓库的bucket名称。
     "bucket": "patent-esbak",
     # 是否打开快照文件的压缩功能。
     "compress": false,
     # 配置此参数可以限制快照数据的分块大小。当上传的快照数据超过这个数值，数据就会被分块上传到存储仓库中。
     "chunk_size": "1g",
     # 仓库的起始位置，默认是根目录。
     "base_path": "snapshot/"
     }
}
```

步骤3 为自建或第三方友商Elasticsearch创建快照。

- 为所有索引创建快照

  例如，创建一个名为“snapshot_1”的快照。

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

- 为指定索引创建快照

  例如，创建一个名为“snapshot_test”的快照，该快照包含索引“patent_analyse”和“patent”。

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

步骤4 查看集群的快照创建进度。

- 执行如下命令可以查看所有快照信息：

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

- 执行如下命令可以查看指定快照“snapshot_1”的信息：

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

步骤5 将快照数据从存储仓库迁移到对象存储服务OBS中。

对象存储迁移服务（OMS）支持多种云服务商数据迁移到对象存储服务OBS中，具体请参见**各云服务商迁移教程**。

步骤6 在CSS服务的Elasticsearch集群中创建一个存储仓库关联到OBS，用于恢复自建或第三方友商Elasticsearch的快照数据。

例如，在集群中创建一个“my_backup_all”的存储仓库，关联上一步数据迁移目的端的OBS。

```
PUT _snapshot/my_backup_all/
{
   "type" : "obs",
   "settings" : {
      # OBS的内网访问域名。
      "endpoint" : "obs.xxx.xxx.com",
      "region" : "xxx",
```



```
    # 访问OBS的用户名和密码。认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险, 建议在配置文件或者环境变量中密文存放, 使用时解密, 确保安全；本示例以ak和sk保存在环境变量中为例, 运行本示例前请先在本地环境中设置环境变量ak和sk。
    "access_key": "ak",
    "secret_key": "sk",
    # OBS的桶名称，和上一步迁移目的端的OBS桶名保持一致。
    "bucket" : "esbak",
    "compress" : "false",
    "chunk_size" : "1g",
    # 注意“snapshot”后面没有/。
    "base_path" : "snapshot",
    "max_restore_bytes_per_sec": "100mb",
    "max_snapshot_bytes_per_sec": "100mb"
  }
}
```

步骤7 在CSS服务的Elasticsearch集群中恢复快照数据。

1. 查看所有快照信息。
   ```
   GET _snapshot
   ```
2. 恢复快照。
   - 恢复某一快照的所有索引。例如恢复名为“snapshot_1”的快照的所有索引数据。
     ```
     POST _snapshot/my_backup_all/snapshot_1/_restore?wait_for_completion=true
     ```
   - 恢复某一快照的部分索引。例如名为“snapshot_1”的快照中只恢复非“.”开头的索引。
     ```
     POST _snapshot/my_backup/snapshot_1/_restore
     {"indices":"*,-.monitoring*,-.security*,-.kibana*","ignore_unavailable":"true"}
     ```
   - 恢复某一快照中的指定索引，并重命名。例如在名为“snapshot_1”的快照中，将索引“index_1”恢复为“restored_index_1”，“index_2”恢复为“restored_index_2”。
     ```
     POST /_snapshot/my_backup/snapshot_1/_restore
     {
       # 只恢复索引“index_1”和“index_2”，忽略快照中的其他索引。
       "indices": "index_1,index_2"
       # 查找正在恢复的索引，该索引名称需要与提供的模板匹配。
       "rename_pattern": "index_(.+)",
       # 重命名查找到的索引。
       "rename_replacement": "restored_index_$1"
     }
     ```

步骤8 查看快照恢复结果。

- 查看所有快照的恢复结果：
  ```
  GET /_recovery/
  ```
- 查看指定索引的快照恢复结果： GET {index_name}/_recovery
  ```
  GET {index_name}/_recovery
  ```

----结束

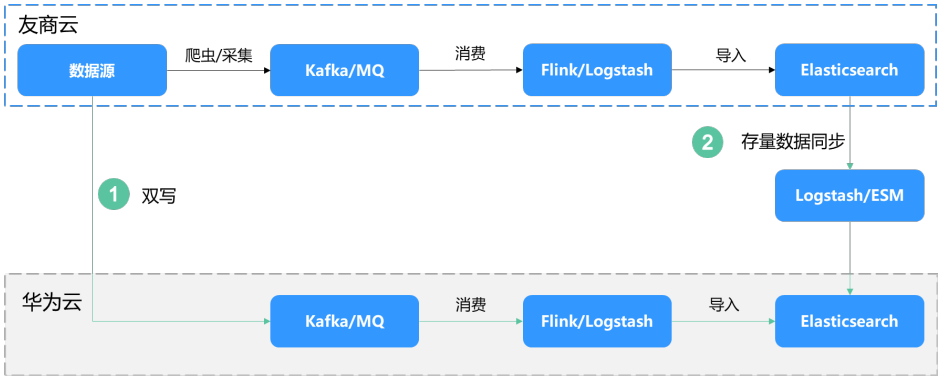
3.3 源端为 Kafka/MQ

迁移流程

在泛IOT、新闻、舆情、社交等数据量较大的行业，通常会引入消息中间件（kafka、MQ等）对流量进行削峰填谷，然后借助Flink/Logstash等工具消费数据并进行数据预处理，最终将数据导入到搜索引擎，对外提供搜索服务。

对于这种源端是kafka/MQ的集群，集群迁移流程如图3-12所示。

图 3-12 源端为 Kafka/MQ 的集群迁移流程



该迁移方案具有割接方便和通用性好的优点。

- 割接方便：一旦两个ES集群的数据保持一致后，随时可割接。
- 通用性好：两边均可同步对客户端的ES进行增删改查操作。

操作步骤

步骤1 订阅增量。在kafka/MQ中新建消费组，订阅增量数据。

步骤2 存量数据同步。使用Logstash等工具将源端ES集群中的数据迁移到CSS集群中。如果使用Logstash进行数据迁移，可以参考**使用Logstash迁移集群数据**。

步骤3 增量数据同步。待存量数据同步完成之后开启增量消费组，利用ES对数据操作的幂等性，等新的消费组追上之前的消费组时，两边的数据就会追平。

说明

关于日志场景，源端ES集群中的数据并不需要迁移，即可跳过存量数据同步的步骤，待增量数据同步完成后，两边持续同步跑一段时间（例如3天或7天），然后直接割接即可。

----结束

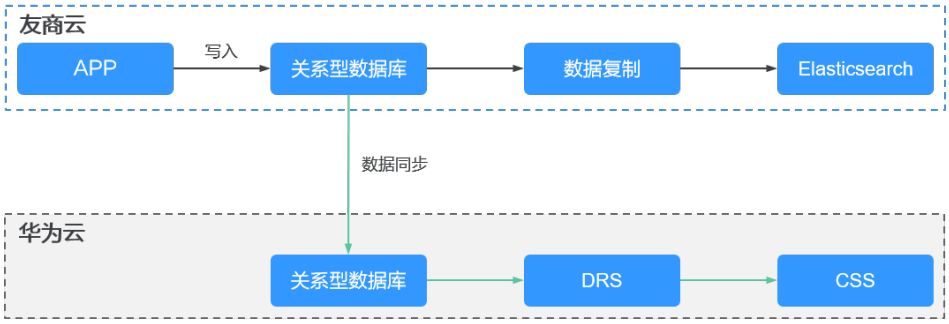
3.4 源端为数据库

迁移流程

因为Elasticsearch支持全文检索和Ad Hoc查询，所以经常作为关系型数据库（例如GaussDB for MySQL等）的补充，以此提升数据库的全文检索能力和高并发的Ad Hoc查询能力。

对于这种源端是数据库的集群，集群迁移流程如**图3-13**所示。

图 3-13 源端为数据库的集群迁移流程



该迁移方案具有割接方便和通用性好的优点。

- 割接方便：当CSS进入增量同步状态，即可启动业务割接操作。
- 通用性好：两边均可同步对客户端的ES进行增删改查操作。

迁移指导

通过数据复制服务（DRS）可以实现GaussDB for MySQL等关系型数据库之间的数据迁移和同步，支持的数据库类型请参见**同步方案概览**。

表3-12为DRS服务所支持的源数据库迁移至CSS服务的具体指导。

表 3-12 数据同步指导

源数据库类型	目标数据库类型	同步模式	相关文档
RDS for MySQL	CSS/ES	全量+增量数据同步	**将MySQL同步到CSS/ES**
• 本地自建MySQL数据库 • ECS自建MySQL数据库	CSS/ES	全量+增量数据同步	**将MySQL同步到CSS/ES**
GaussDB(for MySQL)	CSS/ES	全量+增量数据同步	**将GaussDB(for MySQL)同步到CSS/ES**

4 接入集群

4.1 方案概述

Elasticsearch集群支持多种连接方式，根据业务使用的编程语言可以自行选择接入方式。针对CSS服务的3种不同安全模式的集群（非安全模式的集群、安全模式+HTTP协议的集群、安全模式+HTTPS协议的集群），不同客户端的支持情况请参见**表4-1**。

- CSS提供了可视化的Kibana和Cerebro界面用于监控、使用集群。在CSS服务控制台，可以快速接入每个Elasticsearch集群的Kibana和Cerebro。
- 其他客户端也可以接入并使用Elasticsearch集群，如Curl命令行、Java客户端、Python客户端等形式，亦或是使用Hadoop提供的客户端实现更复杂的应用。Elasticsearch官方提供了的Java客户端，包括Rest High Level Client、Rest Low Level Client和Transport Client。建议使用对应Elasticsearch集群版本的Java客户端，否则可能存在兼容性问题。

表 4-1 不同客户端接入集群的支持情况

客户端	非安全模式的集群	安全模式+HTTP协议的集群	安全模式+HTTPS协议的集群
Kibana	3种安全模式的集群都支持，安全模式的集群登录Kibana时需要输入用户名和密码进行安全认证，接入集群的操作请参见**快速访问Elasticsearch集群**。		
Cerebro	3种安全模式的集群都支持，安全模式的集群登录Cerebro时需要输入用户名和密码进行安全认证，接入集群的操作请参见**快速访问Elasticsearch集群**。		
Curl	3种安全模式的集群都支持，接入命令有差别，具体请参见**通过Curl命令行接入集群**。		
Java（Rest High Level Client）	3种安全模式的集群都支持，接入命令有差别，具体请参见**通过Rest High Level Client接入集群**。		

客户端	非安全模式的集群	安全模式+HTTP协议的集群	安全模式+HTTPS协议的集群
Java（Rest Low Level Client）	3种安全模式的集群都支持，接入命令有差别，具体请参见**通过Rest Low Level Client接入集群**。		
Java（Transport Client）	只支持非安全模式的集群，具体请参见**通过Transport Client接入集群**。	不支持	不支持
Java（Spring Boot）	3种安全模式的集群都支持，接入命令有差别，具体请参见**通过Spring Boot接入集群**。		
Python	3种安全模式的集群都支持，接入命令有差别，具体请参见**通过Python接入集群**。		
ES-Hadoop	3种安全模式的集群都支持，接入命令有差别，具体请参见**通过ES-Hadoop实现Hive读写Elasticsearch数据**。		

4.2 快速访问 Elasticsearch 集群

CSS服务创建的Elasticsearch集群自带Kibana和Cerebro组件，支持一键打开Kibana和Cerebro，快速访问Elasticsearch集群。若需要使用Curl命令行、Java客户端、Python客户端等访问Elasticsearch集群可以参考**接入集群**。

通过 Kibana 访问集群

1. 登录云搜索服务管理控制台。
2. 在“集群管理”页面选择需要登录的集群，单击“操作”列中的“Kibana”进入Kibana登录界面。
   - 非安全模式的集群：将直接进入Kibana操作界面。
   - 安全模式的集群：需要在登录页面输入用户名和密码，单击“Log In”进入Kibana操作界面。用户名默认为admin，密码为创建集群时设置的管理员密码。
3. 登录成功后，可在Kibana界面进行相关操作访问Elasticsearch集群。

通过 Cerebro 访问集群

1. 登录云搜索服务管理控制台。
2. 在“集群管理”页面选择需要登录的集群，单击“操作”列中的“更多 > Cerebro”进入Cerebro登录页面。
   - 非安全模式的集群：单击Cerebro登录页面的集群名称即可进入Cerebro操作界面。
   - 安全模式的集群：单击Cerebro登录页面的集群名称，再输入用户名和密码，单击“Authenticate”进入Cerebro操作界面。用户名默认为admin，密码为创建集群时设置的管理员密码。
3. 登录成功后，可在Cerebro界面进行相关操作访问Elasticsearch集群。

4.3 通过 Curl 命令行接入集群

当CSS集群和弹性云服务器ECS在同一VPC内时，在此ECS中可以通过Curl命令直接访问Elasticsearch集群，此方法多用于前期调试Elasticsearch的连接性，排查访问集群的客户端是否和Elasticserach节点的网络连通。

准备工作

- CSS集群处于可用状态。
- 已经具备满足如下要求的ECS：
  - ECS的VPC需要与CSS集群在同一个VPC中，即保证网络连通。
  - ECS的安全组需要和CSS集群的安全组相同。

    如果不同，请修改ECS安全组或者配置ECS安全组的出入规则，允许集群所有安全组的访问。修改操作请参见**配置安全组规则**。

  ECS的使用请参见**弹性云服务器ECS的操作导航**。

操作步骤

1. 获取集群的内网访问地址。访问集群时，需要输入内网访问地址。
   a. 在云搜索服务管理控制台，单击左侧导航栏的“集群管理”。
   b. 在集群管理列表页面，选择需要访问的集群，获取并记录“内网访问地址”，一般是“<host>:<port>”或“<host>:<port>,<host>:<port>”样式。

      如果集群只有一个节点，此处仅显示1个节点的IP地址和端口号，例如“10.62.179.32:9200”；如果集群有多个节点，此处显示所有节点的IP地址和端口号，例如“10.62.179.32:9200,10.62.179.33:9200”。
2. 在ECS中执行如下命令，访问集群。集群的安全模式不同，访问命令也不同。
   - 非安全模式的集群
     ```
     curl "http://<host>:<port>"
     ```
   - 安全模式+HTTP协议的集群
     ```
     curl -u <user>:<password> "http://<host>:<port>"
     ```
   - 安全模式+HTTPS协议的集群
     ```
     curl -u <user>:<password> -k "https://<host>:<port>"
     ```

表 4-2 变量说明

变量名	说明
<host>	集群中各节点的IP地址，当集群包含多个节点时，会存在多个IP地址，可以任选其中一个发送。
<port>	集群节点的访问端口号，一般为9200。
<user>	访问集群的用户名。
<password>	用户名对应的密码。

访问示例如下：

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

返回结果如下：

```
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"
}
```

说明

更多命令，请参见**Elasticsearch官方文档**。

4.4 通过 Java 接入集群

4.4.1 通过 Rest High Level Client 接入集群

Elasticsearch官方提供了SDK（Rest High level Client）方式连接集群，Rest Client客户端对Elasticsearch的API进行了封装，用户只需要构造对应的结构即可对ES集群进行访问。Rest Client的详细使用请参考官方文档：**https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/master/index.html**

本文介绍通过Rest High level Client访问CSS集群的配置说明。Rest High level Client接入集群有3种方式：

- **通过Rest High Level Client连接非安全集群**：适用于非安全模式的集群
- **通过Rest High Level Client连接安全集群（不使用安全证书）**：适用于安全模式+HTTP协议的集群、安全模式+HTTPS协议的集群（忽略证书）
- **通过Rest High Level Client连接安全集群（使用安全证书）**：适用于安全模式+HTTPS协议的集群

注意事项

建议Rest High Level Client的版本和Elasticsearch的版本保持一致，例如需要访问的ES集群版本是7.6.2，则使用的Rest High Level Client客户端版本建议也是7.6.2。如果您使用相比Elasticsearch集群更高版本的Java Rest High Level Client且存在少量请求的兼容性问题，您可以使用“RestHighLevelClient.getLowLevelClient()”方式直接获取Low Level Client，实现自定义的Elasticsearch请求内容。

准备工作

- CSS集群处于可用状态。
- 确保运行Java代码的服务器与CSS集群的网络是互通的。
- 确认服务器已安装JDK1.8，JDK1.8官网下载地址：**http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html**。
- 引入Java依赖。

  其中*7.6.2*为Elasticsearch Java客户端的版本号。

  - Maven方式引入：

    ```
    <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方式引入：

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

通过 Rest High Level Client 连接非安全集群

通过Rest High Level Client连接非安全集群，并查询*test*索引是否存在。代码示例如下：

```
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;

/**
 * Rest Hive Level 连接非安全集群
 */
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();
    }

    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }
}
```

其中，*host*为集群各个节点的IP地址列表，当存在多个IP地址时，中间用“,”隔开；*test*为查询的索引名称。

通过 Rest High Level Client 连接安全集群（不使用安全证书）

该场景适用于连接2种集群：安全模式+HTTP协议的集群、安全模式+HTTPS协议的集群（忽略证书）。

代码示例如下：

```
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;

/**
 * Rest High Level连接安全集群（不使用证书）
 */
public class Main {
    /**
     * 创建客户端的类，定义create函数用于创建客户端。
     */
    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;
    }

    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }

    /**
     * trustAllCerts忽略证书配置。
     */
    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;
        }
    }

    /**
     * main函数参考如下，调用上面的create函数创建客户端，查询“test”索引是否存在。
     */
    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();
    }
}
```

表 4-3 函数中的变量说明

参数	描述
host	ES集群各个节点（或者独立Client节点）的IP地址列表，当存在多个IP地址时，中间用“,”隔开。
port	ES集群的连接端口，默认是“9200”。
protocol	连接协议，“http”或者“https”，根据集群实际情况填写。
connectTimeout	socket连接超时时间。
connectionRequestTimeout	socket连接请求超时时间。
socketTimeout	socket请求超时时间。

参数	描述
username	访问集群的用户名。
password	用户名对应的密码。

通过 Rest High Level Client 连接安全集群（使用安全证书）

该场景适用于使用安全证书连接安全模式+HTTPS协议的集群。

1. 获取安全证书（CloudSearchService.cer）。
   a. 登录云搜索服务控制台。
   b. 选择“集群管理”进入集群列表。
   c. 单击对应集群的名称，进入集群基本信息页面。
   d. 在“基本信息”页面，单击“HTTPS访问”后面的“下载证书”。

   图 4-1 下载证书

2. 转换安全证书（CloudSearchService.cer）。将下载的安全证书上传到客户端机器上，使用keytool工具将“.cer”证书转换成Java可以读取的“.jks”证书格式。

- 在Linux系统中，执行如下命令转换证书。
  ```
  keytool -import -alias newname -keystore ./truststore.jks -file ./CloudSearchService.cer
  ```
- 在Windows系统中，执行如下命令转换证书。
  ```
  keytool -import -alias newname -keystore .\truststore.jks -file .\CloudSearchService.cer
  ```

其中，*newname*是由用户自定义的证书名称。

该命令执行后，会提示设置证书密码，并确认密码。请保存该密码，后续接入集群会使用。

3. 接入集群。代码示例如下：

```
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;

/**
 * Rest Hive Level连接安全集群（使用https证书）
 */
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");
            //也可以使用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;
    }

    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    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类定义。
     */
    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];
        }
    }

    /**
     * main函数参考如下，调用上面的create函数创建客户端，查询“test”索引是否存在。
     */
    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();
    }
}
```

表 4-4 函数中的参数说明

参数	描述
host	ES集群各个节点（或者独立Client节点）的IP地址列表，当存在多个IP地址时，中间用“,”隔开。
port	ES集群的连接端口，默认是“9200”。
protocol	连接协议，此处填写“https”。
connectTimeout	socket连接超时时间。

参数	描述
connectionRequestTimeout	socket连接请求超时时间。
socketTimeout	socket请求超时时间。
username	访问集群的用户名。
password	用户名对应的密码。
cerFilePath	证书路径。
cerPassword	证书密码。

4.4.2 通过 Rest Low Level Client 接入集群

High Level Client是在Low Level Client基础上进行封装的，如果High Level Client中的方法调用（例如.search，.bulk）不能满足使用需求，或存在兼容性问题，可以选择使用Low Level Client方式，甚至可以使用“HighLevelClient.getLowLevelClient()”方式直接获取Low Level Client。使用Low Level Client发送请求时需要自己定义请求结构，使用上更为灵活，能满足所有Elasticsearch支持的请求格式，例如GET、POST、DELETE、HEAD等。

本文介绍通过Rest Low Level Client访问CSS集群的配置说明。Rest Low Level Client接入集群有3种方式，每一种方式又分为直接创建Rest Client（即Rest Low Level Client）和通过创建High Level Client再调用getLowLevelClient()获取Low Level Client。

- **通过Rest Low Level Client连接非安全集群**：适用于非安全模式的集群
- **通过Rest Low Level Client连接安全集群（不使用安全证书）**：适用于安全模式+HTTP协议的集群、安全模式+HTTPS协议的集群（忽略证书）
- **通过Rest Low Level Client连接安全集群（使用安全证书）**：适用于安全模式+HTTPS协议的集群

注意事项

建议Rest Low Level Client的版本和Elasticsearch的版本保持一致，例如需要访问的ES集群版本是7.6.2，则使用的Rest Low Level Client客户端版本建议也是7.6.2。

准备工作

- CSS集群处于可用状态。
- 确保运行Java代码的服务器与CSS集群的网络是互通的。
- 确认服务器已安装JDK1.8，JDK1.8官网下载地址：**http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html**。
- 通过Maven方式引入apache版本。代码示例以7.6.2版本为例。

  其中*7.6.2*为Elasticsearch Java客户端的版本号。

```
<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>
```

通过 Rest Low Level Client 连接非安全集群

- **方式一：直接创建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"));
        /**
         * 创建Rest Low Level Client。
         */
        RestClient lowLevelClient = builder.build();
        /**
         * 查询“test”索引是否存在。当索引存在时返回200，不存在时返回404。
         */
        Request request = new Request("HEAD", "/test");
        Response response = lowLevelClient.performRequest(request);
        System.out.println(response.getStatusLine().getStatusCode());
        lowLevelClient.close();
    }

    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }
}
```

- **方式二：创建High Level Client再调用getLowLevelClient()获取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);
        /**
         * 创建High Level Client再调用getLowLevelClient()获取Low Level Client，即client创建仅下面这一行代码存在差别。
         */
        final RestClient lowLevelClient = restHighLevelClient.getLowLevelClient();
        /**
```

```
        * 查询“test”索引是否存在。当索引存在时返回200，不存在时返回404。
        */
        Request request = new Request("HEAD", "/test");
        Response response = lowLevelClient.performRequest(request);
        System.out.println(response.getStatusLine().getStatusCode());
        lowLevelClient.close();
    }

    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }
}
```

其中，*host*为集群各个节点的IP地址列表，当存在多个IP地址时，中间用“,”隔开；*test*为查询的索引名称。

通过 Rest Low Level Client 连接安全集群（不使用安全证书）

- **方式一：直接创建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 {

    /**
     * 创建客户端的类，定义create函数用于创建客户端。
     */
    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;
    }

    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }

    /**
     * trustAllCerts忽略证书配置。
     */
    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;
            }
        }
    };

    /**
     * CustomConnectionKeepAliveStrategy函数设置连接的保活时间，主要应对大量短连接的情况和数据
请求不高的场景。
     */
    public static class CustomConnectionKeepAliveStrategy extends
DefaultConnectionKeepAliveStrategy {
        public static final CustomConnectionKeepAliveStrategy INSTANCE = new
CustomConnectionKeepAliveStrategy();

        private CustomConnectionKeepAliveStrategy() {
            super();
        }

        /**
         * 最大keep alive的时间（分钟）
         * 这里默认为10分钟，可以根据实际情况设置。可以观察客户端机器状态为TIME_WAIT的TCP连接
数，如果太多，可以增大此值。
         */
```



```
        private final long MAX_KEEP_ALIVE_MINUTES = 10;

        @Override
        public long getKeepAliveDuration(HttpResponse response, HttpContext context) {
            long keepAliveDuration = super.getKeepAliveDuration(response, context);
            // <0 为无限期keepalive
            // 将无限期替换成一个默认的时间
            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;
        }
    }

    /**
     * main函数参考如下，调用create函数创建Rest Low Level Client客户端，查询“test”索引是否存在。
     */
    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();
    }
}
```

- **方式二：创建High Level Client再调用getLowLevelClient()获取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 {

    /**
     * 创建客户端的类，定义create函数用于创建客户端。
     */
    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;
    }

    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    public static HttpHost[] constructHttpHosts(List<String> host, int port, String protocol) {
        return host.stream().map(p -> new HttpHost(p, port, protocol)).toArray(HttpHost[]::new);
    }

    /**
     * trustAllCerts忽略证书配置。
     */
    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;
            }
        }
    };

    /**
     * CustomConnectionKeepAliveStrategy函数设置连接的保活时间，主要应对大量短连接的情况和数据
请求不高的场景。
     */
    public static class CustomConnectionKeepAliveStrategy extends
DefaultConnectionKeepAliveStrategy {
        public static final CustomConnectionKeepAliveStrategy INSTANCE = new
CustomConnectionKeepAliveStrategy();

        private CustomConnectionKeepAliveStrategy() {
            super();
```



```
        }

        /**
         * 最大keep alive的时间（分钟）
         * 这里默认为10分钟，可以根据实际情况设置。可以观察客户端机器状态为TIME_WAIT的TCP连接
数，如果太多，可以增大此值。
         */
        private final long MAX_KEEP_ALIVE_MINUTES = 10;

        @Override
        public long getKeepAliveDuration(HttpResponse response, HttpContext context) {
            long keepAliveDuration = super.getKeepAliveDuration(response, context);
            // <0 为无限期keepalive
            // 将无限期替换成一个默认的时间
            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;
        }
    }

    /**
     * main函数参考如下，调用create函数创建High Level Client再调用getLowLevelClient()获取Low
Level Client，并查询“test”索引是否存在。
     */
    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();
    }
}
```

表 4-5 函数中的变量说明

参数	描述
host	ES集群各个节点（或者独立Client节点）的IP地址列表，当存在多个IP地址时，中间用“,”隔开。
port	ES集群的连接端口，默认是“9200”。
protocol	连接协议，“http”或者“https”，根据集群实际情况填写。
connectTimeout	socket连接超时时间。
connectionRequestTimeout	socket连接请求超时时间。
socketTimeout	socket请求超时时间。
username	访问集群的用户名。
password	用户名对应的密码。

通过 Rest Low Level Client 连接安全集群（使用安全证书）

- **方式一：直接创建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);

    /**
     * 创建客户端的类，定义create函数用于创建客户端。
     */
    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");
            //也可以使用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;
    }
```

```
    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    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];
        }
    }

    /**
     * main函数参考如下，调用create函数创建Rest Low Level Client，查询“test”索引是否存在。
     */
    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();
    }
}
```

- **方式二：创建High Level Client再调用getLowLevelClient()获取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);

    /**
     * 创建客户端的类，定义create函数用于创建客户端。
     */
    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");
            //也可以使用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;
    }

    /**
     * constructHttpHosts函数转换host集群节点ip列表。
     */
    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];
    }
}

/**
 * main函数参考如下，调用create函数创建High Level Client再调用getLowLevelClient()获取Low
Level Client，并查询“test”索引是否存在。
 */
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();
}
}
```

表 4-6 函数中的参数说明

参数	描述
host	ES集群各个节点（或者独立Client节点）的IP地址列表，当存在多个IP地址时，中间用“,”隔开。
port	ES集群的连接端口，默认是“9200”。
protocol	连接协议，此处填写“https”。
connectTimeout	socket连接超时时间。
connectionRequestTimeout	socket连接请求超时时间。
socketTimeout	socket请求超时时间。
username	访问集群的用户名。
password	用户名对应的密码。
cerFilePath	证书路径。
cerPassword	证书密码。

4.4.3 通过 Transport Client 接入集群

本文介绍通过Transport Client访问CSS服务非安全集群的配置说明。如果是安全模式的集群，建议通过Rest High Level Client访问Elasticsearch集群。

注意事项

建议Transport Client的版本和Elasticsearch的版本保持一致，例如需要访问的ES集群版本是7.6.2，则使用的Transport Client客户端版本建议也是7.6.2。

准备工作

- CSS集群处于可用状态。
- 确保运行Java代码的服务器与CSS集群的网络是互通的。
- 确认服务器已安装JDK1.8，JDK1.8官网下载地址：**http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html**。
- 引入Java依赖：

  其中*7.6.2*为Elasticsearch Java客户端的版本号。

```
<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>
```

操作步骤

以下介绍Transport Client连接Elasticsearch集群并查询*test*索引是否存在的代码示例。

```
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());
    }
}
```

其中，*cluster_name*为集群的名称；*host1*、*host2*为集群节点IP地址，可通过**GET _cat/nodes**命令查看节点的IP地址。

4.4.4 通过 Spring Boot 接入集群

本文介绍通过Spring Boot访问CSS集群的配置说明。Spring Boot接入集群有3种方式：

- **通过Spring Boot接入HTTP集群**：适用于非安全模式的集群、安全模式+HTTP协议的集群
- **使用Spring Boot接入HTTPS集群（不使用安全证书）**：适用于安全模式+HTTPS协议的集群
- **使用Spring Boot接入HTTPS集群（使用安全证书）**：适用于安全模式+HTTPS协议的集群

说明

Spring Boot的具体使用方式请参见官方文档：**https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/**。

注意事项

- 建议Elasticsearch Rest High Level Client的版本和Elasticsearch的版本保持一致，例如需要访问的Elasticsearch集群版本是7.10.2，则使用的Elasticsearch Rest High Level Client客户端版本建议也是7.10.2。
- 本章节以2.5.5版本Spring Boot为例介绍Spring Boot接入集群的方式，对应的spring data elasticsearch版本是4.2.x。

准备工作

- CSS集群处于可用状态。
- 确保运行Java代码的服务器与CSS集群的网络是互通的。
- 确认服务器已安装JDK1.8，JDK1.8官网下载地址：**http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html**。
- 创建SpringBoot项目。
- 引入Java依赖。

  其中*7.10.2*为Elasticsearch Java客户端的版本号。

  - Maven方式引入：

```
<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>
```

通过 Spring Boot 接入 HTTP 集群

该场景适用于连接非安全模式的集群或是安全模式+HTTP协议的集群。

配置文件：

```
elasticsearch.url=host1:9200,host2:9200
//非安全集群不用配置如下两行。
elasticsearch.username=username
elasticsearch.password=password
```

表 4-7 参数说明

参数	描述
host	Elasticsearch集群节点的IP地址。
username	访问集群的用户名。
password	用户名对应的密码。

配置代码：

说明

- com.xxx为项目目录，例如com.company.project。
- com.xxx.repository为仓库目录，通过extends org.springframework.data.elasticsearch.repository.ElasticsearchRepository进行具体定义。

```
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;

    //非安全集群不用配置如下两个参数。
    @Value("${elasticsearch.username}")
    public String elasticsearchUsername;

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

    @Override
    @Bean
    public RestHighLevelClient elasticsearchClient() {
        final ClientConfiguration clientConfiguration = ClientConfiguration.builder()
            .connectedTo(elasticsearchUrl)
            //非安全集群不用配置“.withBasicAuth”。
            .withBasicAuth(elasticsearchUsername, elasticsearchPassword)
            .build();

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

使用 Spring Boot 接入 HTTPS 集群（不使用安全证书）

该场景适用于不使用安全证书连接安全模式+HTTPS协议的集群。

配置文件：

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

表 4-8 参数说明

参数	描述
host	Elasticsearch集群节点的IP地址。
username	访问集群的用户名。
password	用户名对应的密码。

配置代码：

说明

- com.xxx为项目目录，例如com.company.project。
- com.xxx.repository为仓库目录，通过extends org.springframework.data.elasticsearch.repository.ElasticsearchRepository进行具体定义。

```
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;
        }
    }
}
```

使用 Spring Boot 接入 HTTPS 集群（使用安全证书）

该场景适用于使用安全证书连接安全模式+HTTPS协议的集群。

1. 获取安全证书（CloudSearchService.cer）。
   a. 登录云搜索服务控制台。
   b. 选择“集群管理”进入集群列表。
   c. 单击对应集群的名称，进入集群基本信息页面。
   d. 在“基本信息”页面，单击“HTTPS访问”后面的“下载证书”。

图 4-2 下载证书

2. 转换安全证书（CloudSearchService.cer）。将下载的安全证书上传到客户端机器上，使用keytool工具将“.cer”证书转换成Java可以读取的“.jks”证书格式。
   - 在Linux系统中，执行如下命令转换证书。
     keytool -import -alias *newname* -keystore ./truststore.jks -file ./CloudSearchService.cer
   - 在Windows系统中，执行如下命令转换证书。
     keytool -import -alias *newname* -keystore .\truststore.jks -file .\CloudSearchService.cer

   其中，*newname*是由用户自定义的证书名称。

   该命令执行后，会提示设置证书密码，并确认密码。请保存该密码，后续接入集群会使用。
3. application.properties配置文件：

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

表 4-9 参数说明

参数	描述
host	Elasticsearch集群节点的IP地址。
username	访问集群的用户名。
password	用户名对应的密码。

4. 配置代码：

   说明

   - com.xxx为项目目录，例如com.company.project。
   - com.xxx.repository为仓库目录，通过extends org.springframework.data.elasticsearch.repository.ElasticsearchRepository进行具体定义。

```
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;
        }
    }
}
```

其中，*cerFilePath*和*cerPassword*是生成的.jks证书的存放路径及其密码。

4.5 通过 Python 接入集群

本文介绍通过Python语言访问CSS集群的配置说明。

准备工作

- CSS集群处于可用状态。
- 确保运行Python代码的服务器与CSS集群的网络是互通的。

操作步骤

1. 安装Elasticsearch Python客户端，建议和Elasticsearch的版本保持一致，例如需要访问的集群版本是7.6.2，则安装7.6的Elasticsearch Python客户端。
   ```
   pip install Elasticsearch==7.6
   ```
2. 创建Elasticsearch客户端并查看是否存在索引“test”。根据集群安全模式参考对应的示例代码。
   - 非安全模式的集群
     ```
     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'))
     ```
   - 安全模式+HTTP协议的集群
     ```
     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'))
```

- 安全模式+HTTPS协议的集群

```
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'))
```

表 4-10 函数中的变量说明

参数	描述
host	ES集群各个节点（或者独立Client节点）的IP地址列表，当存在多个IP地址时，中间用“,”隔开。
port	ES集群的连接端口，填写“9200”。
username	访问集群的用户名。
password	用户名对应的密码。

3. 通过Elasticsearch客户端创建集群索引。

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

4. 通过Elasticsearch客户端查询上一步创建的索引。


```
body = {
    "query": {
        "match": {
            "查询字段":  "查询内容"
        }
    }
}
result = es.search(index=index, body=body)
```

4.6 通过 ES-Hadoop 实现 Hive 读写 Elasticsearch 数据

Elasticsearch-Hadoop (ES-Hadoop) 连接器将Hadoop海量的数据存储和深度加工能力与Elasticsearch实时搜索和分析功能结合在一起。它能够让您快速深入了解大数据，并让您在Hadoop生态系统中更好地开展工作。

本文通过MRS的ES-Hadoop与CSS集群连接作为示例，你可以配置其他任何需要使用ES集群的应用。如有需要，也可以参考本文在其他服务中使用Elasticsearch，前提是要保证客户端与Elasticsearch集群网络连通。

准备工作

- CSS集群处于可用状态。
- 确保客户端与CSS集群的网络是互通的。
- 确保CSS集群和MRS集群在同一个区域、可用区、虚拟私有云和子网。

图 4-3 CSS 集群的区域等信息

操作步骤

1. 获取集群的内网访问地址。访问集群时，需要输入内网访问地址。
   a. 在云搜索服务管理控制台，单击左侧导航栏的“集群管理”。
   b. 在集群管理列表页面，选择需要访问的集群，获取并记录“内网访问地址”，一般是“<host>:<port>”或“<host>:<port>,<host>:<port>”样式。
      如果集群只有一个节点，此处仅显示1个节点的IP地址和端口号，例如“10.62.179.32:9200”；如果集群有多个节点，此处显示所有节点的IP地址和端口号，例如“10.62.179.32:9200,10.62.179.33:9200”。

2. 登录MRS集群节点，操作步骤请参见**登录集群节点**。
3. 在MRS集群节点上通过curl命令检查网络连通性，需要确保MRS集群的每个节点都能连通CSS集群。
   - 非安全模式的集群
     ```
     curl -X GET http://<host>:<port>
     ```
   - 安全模式+HTTP协议的集群
     ```
     curl -X GET http://<host>:<port> -u <user>:<password>
     ```
   - 安全模式+HTTPS协议的集群
     ```
     curl -X GET https://<host>:<port> -u <user>:<password> -ik
     ```

   表 4-11 变量说明

变量名	说明
<host>	集群中各节点的IP地址，当集群包含多个节点时，会存在多个IP地址，可以任选其中一个发送。
<port>	集群节点的访问端口号，一般为9200。
<user>	访问集群的用户名。
<password>	用户名对应的密码。

4. 下载**ES-Hadoop的lib包**，并解压zip包获取“elasticsearch-hadoop-x.x.x.jar”文件。版本需要与CSS集群版本一致，例如CSS集群是7.6.2版本，则建议下载elasticsearch-hadoop-7.6.2.zip。
5. 下载httpclient依赖包**commons-httpclient:commons-httpclient-3.1.jar**，其中*3.1*为版本号，建议用户根据实际需要选择。
6. 安装MRS客户端，如果已经安装可以跳过该步骤，未安装的请参见**安装客户端（3.x及之后版本）**。
7. 登录MRS客户端，将下载的ES-Hadoop和httpclient的jar依赖包上传到MRS客户端。
8. 在MRS客户端创建HDFS目录，将ES-Hadoop lib包和httpclient依赖包上传到该目录下。
   ```
   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. 从MRS客户端登录到Hive客户端，具体操作请参见**使用Hive客户端**。
10. 在Hive客户端，添加ES-Hadoop lib包和httpclient依赖包。该命令只对当前会话有效。

    输入**beeline**或**hive**进入到执行界面，执行如下命令：

    ```
    add jar hdfs:///tmp/hadoop-es/commons-httpclient-3.1.jar;
    add jar hdfs:///tmp/hadoop-es/elasticsearch-hadoop-x.x.x.jar;
    ```
11. 在Hive客户端，创建Hive外表。
    - 非安全模式的集群
      ```
      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'
);
```

- 安全模式+HTTP协议的集群

```
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'
);
```

- 安全模式+HTTPS协议的集群

  i. 获取安全证书“CloudSearchService.cer”。

     1) 登录云搜索服务控制台。
     2) 选择“集群管理”进入集群列表。
     3) 单击对应集群的名称，进入集群基本信息页面。
     4) 在“基本信息”页面，单击“HTTPS访问”后面的“下载证书”。

图 4-4 下载证书

ii. 转换安全证书（CloudSearchService.cer）。将下载的安全证书上传到客户端机器上，使用keytool工具将“.cer”证书转换成Java可以读取的“.jks”证书格式。

- 在Linux系统中，执行如下命令转换证书。
  ```
  keytool -import -alias newname -keystore ./truststore.jks -file ./CloudSearchService.cer
  ```
- 在Windows系统中，执行如下命令转换证书。
  ```
  keytool -import -alias newname -keystore .\truststore.jks -file .\CloudSearchService.cer
  ```

其中，*newname*是由用户自定义的证书名称。

该命令执行后，会提示设置证书密码，并确认密码。请保存该密码，后续接入集群会使用。

iii. 将“.jks”文件分发到MRS集群的每个节点的相同路径，如“/tmp”，可以使用**scp**命令进行文件传输。同时，要确保**omm**用户有权限读取该文件，设置权限可以参考如下命令：

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

iv. 创建Hive外表。

```
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'
);
```

表 4-12 ES-Hadoop 参数说明

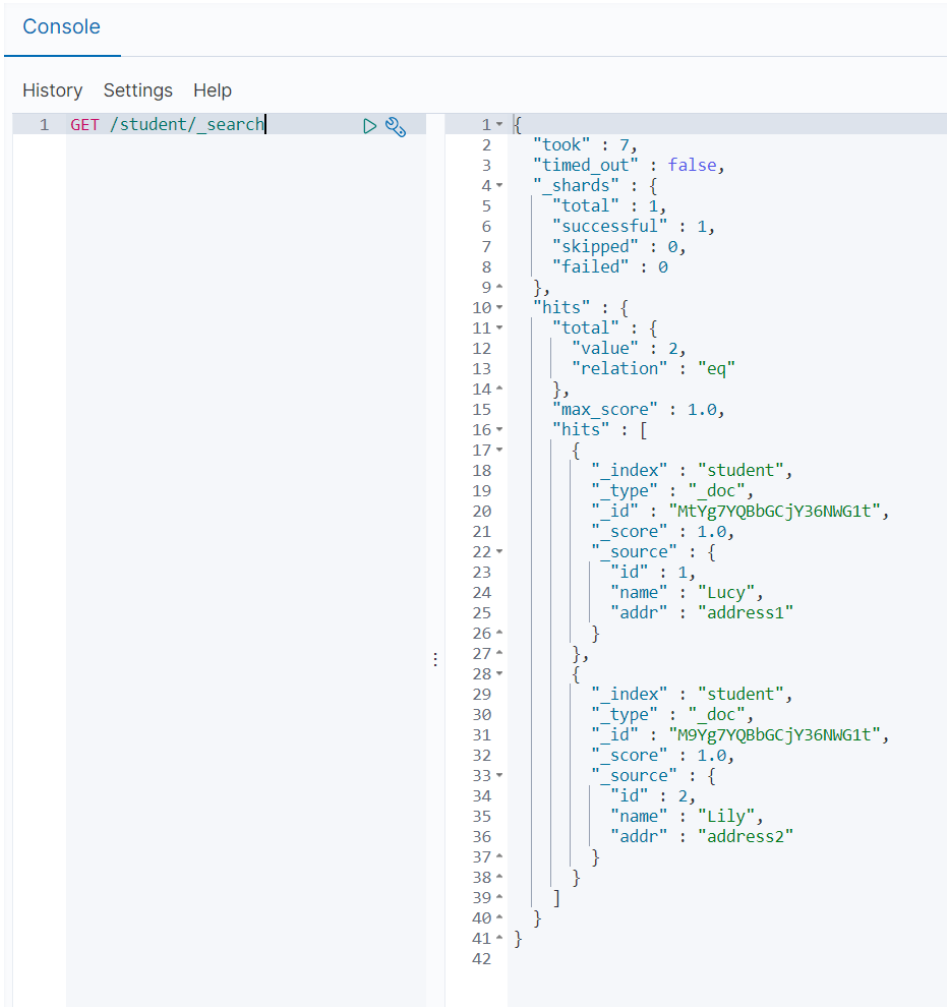
参数	默认值	描述
es.nodes	localhost	CSS集群的访问地址，可以集群列表页面查看“内网访问地址”。
es.port	9200	集群的访问端口号，一般为“9200”。
es.nodes.wan.only	false	是否进行节点嗅探。
es.nodes.discovery	true	是否禁用节点发现。
es.input.use.sliced.partitions	true	是否使用slice分区： ● true：使用。 ● false：不使用。 **说明** 设置为true，可能会导致索引在预读阶段的时间明显变长，有时会远远超出查询数据所耗费的时间。建议设置为false，以提高查询效率。
es.resource	NA	指定要读写的Index和type。
es.net.http.auth.user	NA	访问集群的用户名，只有启用了安全模式才需要配置此值。
es.net.http.auth.pass	NA	用户名所对应的密码，只有启用了安全模式才需要配置此值。
es.net.ssl	false	是否启用SSL，启用后需要配置安全证书信息。
es.net.ssl.truststore.location	NA	“.jks”证书文件的路径，如“file:///tmp/truststore.jks”。
es.nodes.client.only	false	是否配置了独立的Client节点的IP地址给es.nodes（即创建Elasticsearch集群时是否“启用Client节点”）。如果是，则需要将该值改为“true”，否则会报错找不到data节点。

参数	默认值	描述
es.net.ssl.truststore.pass	NA	“.jks”证书文件的密码。

更多ES-Hadoop配置项说明请参见**官方配置说明**。

12. 在Hive客户端，插入数据。
```
INSERT INTO TABLE student VALUES (1, "Lucy", "address1"), (2, "Lily", "address2");
```
13. 在Hive客户端，执行查询。
```
select * from student;
```
查询结果如下：
```
+-------------+---------------+---------------+
| student.id  | student.name  | student.addr  |
+-------------+---------------+---------------+
| 1           | Lucy          | address1      |
| 2           | Lily          | address2      |
+-------------+---------------+---------------+
2 rows selected (0.116 seconds)
```
14. 登录CSS控制台，在“集群管理”页面，单击集群操作列“Kibana”，登录Kibana界面。
15. 进入Kibana的“Dev Tools”页面执行查询命令，查看查询结果。
```
GET /student/_search
```

图 4-5 kibana 查询结果



4.7 通过 Go 接入集群

本文介绍通过Go语言访问CSS集群的配置说明。

准备工作

- CSS集群处于可用状态。
- 确保运行Go代码的服务器与CSS集群的网络是互通的。
- 确认服务器已安装Go，Go官网下载地址：https://go.dev/dl/。

连接非安全集群

连接非安全集群，示例代码如下：

```
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())
}
```

连接安全集群

- 连接未开启https的安全集群，示例如下：

  ```
  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())
  }
  ```

- 连接开启https的安全集群，不使用证书，示例代码如下：

  ```
  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())
}
```

- 连接开启https的安全集群，使用证书，示例代码如下：

```
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())
}
```

运行代码

根据集群类型将以上适配的代码写入到EsTest.gc文件并存放到一个单独的目录，在该目录执行以下命令：

```
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
```

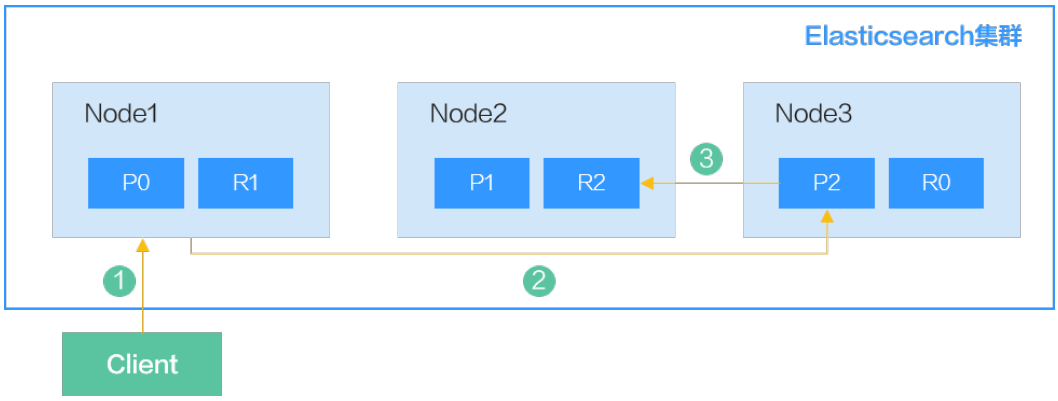
5 优化集群性能

5.1 写入性能优化

CSS集群在使用前，建议参考本文进行集群的写入性能优化，便于提高集群的写入性能，提升使用效率。

数据写入流程

图 5-1 数据写入流程

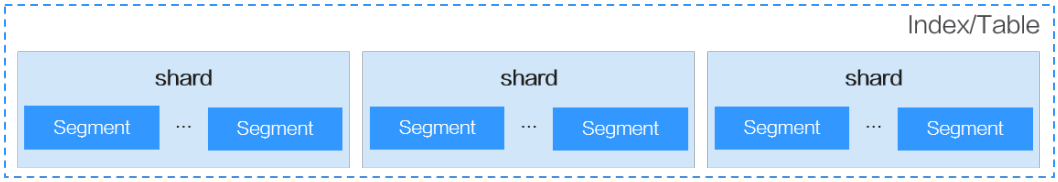


当从客户端往Elasticsearch中写入数据时，写入流程如下：

1. 客户端向Node1发送写数据请求，此时Node1为协调节点。
2. 节点Node1根据数据的_id将数据路由到分片2，此时请求会被转发到Node3，并执行写操作。
3. 当主分片写入成功后，它将请求转发到Node2的副本分片上。当副本写入成功后，Node3将向协调节点报告写入成功，协调节点向客户端报告写入成功。

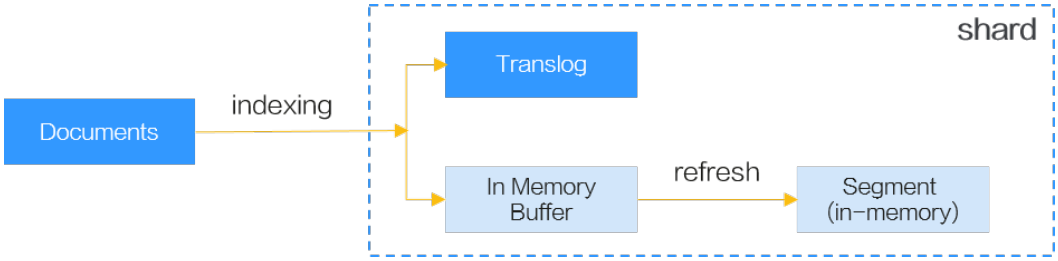
Elasticsearch中的单个索引由一个或多个分片(shard)组成，每个分片包含多个段（Segment），每一个Segment都是一个倒排索引。

图 5-2 Elasticsearch 的索引组成



将文档插入Elasticsearch时，文档首先会被写入缓冲区中，然后在刷新时定期从该缓冲区刷新到Segment中。刷新频率由refresh_interval参数控制，默认每1秒刷新一次。

图 5-3 文档插入 Elasticsearch 的流程



写入性能优化

基于Elasticsearch的数据写入流程分析，有以下几种性能优化方案。

表 5-1 写入性能优化

序号	优化方案	方案说明
1	使用SSD盘或升级集群配置	使用SSD盘可以大幅提升数据写入与merge操作的速度，对应到CSS服务，建议选择“超高IO型”存储，或者超高IO型主机。
2	采用Bulk API	客户端采用批量数据的写入方式，每次批量写入的数据建议在1~10MB之间。
3	随机生成_id	如果采用指定_id的写入方式，数据写入时会先触发一次查询操作，进而影响数据写入性能。对于不需要通过_id检索数据的场景，建议使用随机生成的_id。
4	设置合适的分片数	分片数建议设置为集群数据节点的倍数，且分片的大小控制在50GB以内。

5	关闭副本	数据写入与查询错峰执行，在数据写入时关闭数据副本，待数据写入完成后再开启副本。
		Elasticsearch 7.x版本中关闭副本的命令如下：
		`PUT {index}/_settings`
		`{`
		`  "number_of_replicas": 0`
		`}`

序号	优化方案	方案说明

6	调整索引的刷新频率	数据批量写入时，可以将索引的刷新频率“refresh_interval”设置为更大的值或者设置为“-1”（表示不刷新），通过减少分片刷新次数提高写入性能。
		Elasticsearch 7.x版本中，将更新时间设置为15s的命令如下：
		PUT *{index}*/_settings
		{
		"refresh_interval": "*15s*"
		}

7	优化写入线程数与写入队列大小	为应对突发流量，可以适当地提升写入线程数与写入队列的大小，防止突发流量导致出现错误状态码为429的情况。 Elasticsearch 7.x版本中，可以修改如下自定义参数实现写入优化：thread_pool.write.size，thread_pool.write.queue_size；

8	设置合适的字段类型	指定集群中各字段的类型，防止Elasticsearch默认将字段猜测为keyword和text的组合类型，增加不必要的数据量。其中keyword用于关键词搜索，text用于全文搜索。
		对于不需要索引的字段，建议“index”设置为“false”。
		Elasticsearch 7.x版本中，将字段“field1”设置为不建构索引的命令如下：
		PUT *{index}*
		{
		"mappings": {
		"properties": {
		"field1":{
		"type": "text",
		"index": false
		}
		}
		}
		}

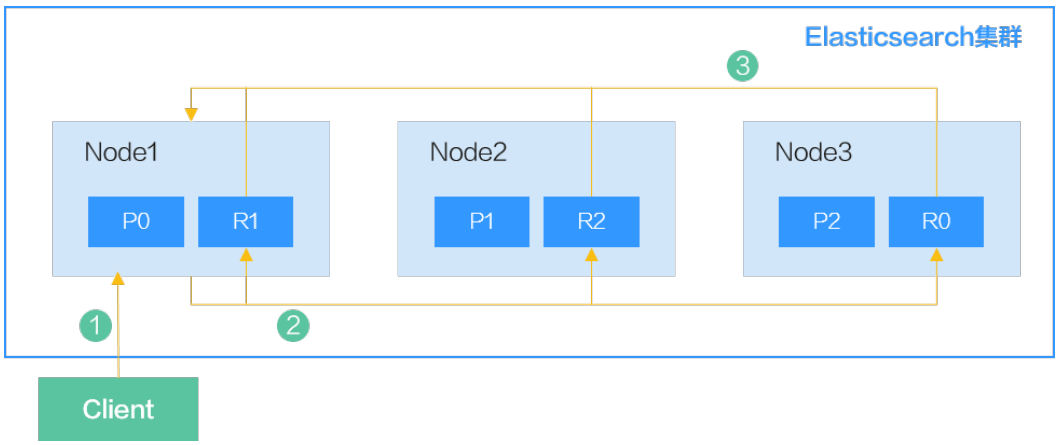
9	优化shard均衡策略	Elasticsearch默认采用基于磁盘容量大小的Load balance策略，多节点时，尤其是在新扩容的节点上，可能出现shard在各节点上分配不均的问题。为避免这类问题，可以通过设置索引级别的参数“routing.allocation.total_shards_per_node”控制索引分片在各节点的分布情况。此参数可以在索引模板中配置，也可以修改已有索引的setting生效。
		修改已有索引的setting的命令如下：
		PUT *{index}*/_settings
		{
		"index": {
		"routing.allocation.total_shards_per_node": 2
		}
		}

5.2 查询性能优化

CSS集群在使用前，建议参考本文进行集群的查询性能优化，便于提高集群的查询性能，提升使用效率。

数据查询流程

图 5-4 数据查询流程

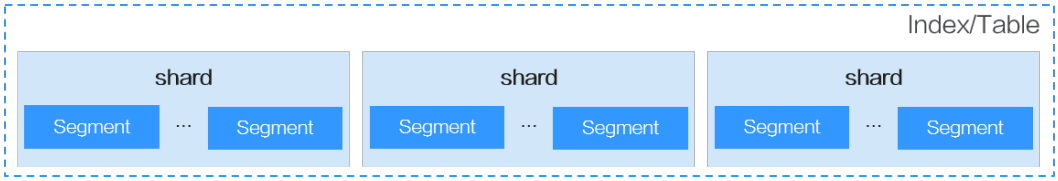


当从客户端往Elasticsearch发送查询请求时，查询流程如下：

1. 客户端向Node1发送查询请求，此时Node1为协调节点。
2. 节点Node1根据查询请求的索引以及其分片分布，进行分片选择；然后将请求转发到Node1、Node2、Node3。
3. 各分片分别执行查询任务；当各分片查询成功后，将查询结果汇聚到Node1，然后协调节点向客户端返回查询结果。

对于某个查询请求，其在节点上默认可并行查询5个分片，多于5个分片时将分批进行查询；在单个分片内，通过逐个遍历各个Segment的方式进行查询。

图 5-5 Elasticsearch 的索引组成



查询性能优化

基于Elasticsearch的数据查询流程分析，有以下几种性能优化方案。

表 5-2 查询性能优化

序号	优化方案	方案说明

1	通过_routing减少检索扫描的分片数	在数据入库时指定routing值，将数据路由到某个特定的分片，查询时通过该routing值将请求转发到某个特定的分片，而不是相关索引的所有分片，进而提升集群整体的吞吐能力。
		Elasticsearch 7.x版本中，设置命令如下：
		• 指定routing值插入数据
		`PUT /{index}/_doc/1?routing=user1`
		`{`
		`  "title": "This is a document"`
		`}`
		• 根据routing值去查询数据
		`GET /{index}/_doc/1?routing=user1`

2	采用index sorting减少检索扫描的Segments数	当请求落到某个分片时，会逐个遍历其Segments，通过使用index sorting，可以使得范围查询、或者排序查询在段内提前终止(early-terminate)。
		Elasticsearch 7.x版本中，示例命令如下：
		`//假设需要频繁使用字段date做范围查询。`
		`PUT {index}`
		`{`
		`  "settings": {`
		`    "index": {`
		`      "sort.field": "date",`
		`      "sort.order": "desc"`
		`    }`
		`  },`
		`  "mappings": {`
		`    "properties": {`
		`      "date": {`
		`        "type": "date"`
		`      }`
		`    }`
		`  }`
		`}`

3	增加query cache提升缓存命中的概率	当filter请求在段内执行时，会通过bitset保留其刷选结果，当下一个类似的查询过来时，就可以复用之前查询的结果，以此减少重复查询。 增加query cache可以通过修改集群的参数配置实现，将自定义缓存参数“indices.queries.cache.size”设置为更大的值。具体操作请参见**参数配置**，修改参数配置后一定要重启集群使参数生效。

4	提前Forcemerge，减小需要扫描的Segments数	对于定期滚动后的只读索引，可以定期执行forcemerge，将小的Segments合并为大的Segments，并将标记为“deleted”状态的索引彻底删除，提升查询效率。
		Elasticsearch 7.x版本中，配置示例如下：
		`//假设配置索引forcemerge后segments数量为10个。`
		`POST /{index}/_forcemerge?max_num_segments=10`

6 管理索引生命周期

6.1 使用生命周期实现自动滚动索引

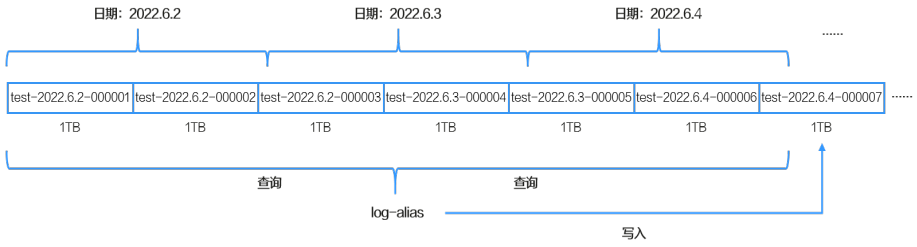
方案概述

对于时间序列数据，随着时间推移数据持续写入，索引会越来越大，通过生命周期管理来定期将数据滚动到新索引，并将历史老索引删除，实现自动滚动索引。

本案例通过配置生命周期策略，当索引的大小达到1TB或索引创建超过1天时，自动滚动生成新索引；当索引创建7天后，关闭数据副本；当索引创建30天后，删除该索引。

假设某个索引，其每天约产生2.4TB的数据，索引别名“log-alias”，其数据在Elasticsearch中的组织形态如下图所示。查询时，指向所有test开头的索引；写入时，指向最新的索引。

图 6-1 log-alias 的组织形态



说明

索引的滚动条件1天是以索引的创建时间来计算的，并不是完整自然日区分的。

前提条件

- CSS集群处于可用状态。
- 集群版本为Elasticsearch 7.6.2及以上。

操作步骤

1. 登录云搜索服务管理控制台。
2. 在左侧导航栏，选择“集群管理”，进入Elasticsearch集群列表页面。
3. 在集群列表页面中，单击集群操作列的“Kibana”登录Kibana页面。
4. 在Kibana的左侧导航中选择“Dev Tools”，进入命令执行页面。
5. 创建Rollover生命周期策略“rollover_workflow”。

   策略定义：当索引的大小达到1TB或索引创建超过1天时，自动进行滚动；当索引创建7天后，关闭数据副本；当索引创建30天后，删除该索引。

```
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": {}
          }
        ]
      }
    ]
  }
}
```

   当生命周期策略创建完成后，执行如下命令可以查询策略详情：


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

6. 新建索引模板“template_test”。

   模板定义：新建的所有“test”开头的索引自动关联上Rollover生命周期策略“rollover_workflow”，并且Rollover时使用“log_alias”作为别名。

```
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"
      }
    }
  }
}
```

表 6-1 参数说明

参数	说明
number_of_shards	索引分片数
number_of_replicas	索引分片副本数
opendistro.index_state_management.policy_id	生命周期的策略名
index.opendistro.index_state_management.rollover_alias	rollover的索引别名

当索引模板创建完成后，可以通过如下命令查询模板详情：

```
GET _template/template_test
```

7. 新建一个索引，指定“aliases”，并配置“is_write_index”为“true”。该索引会自动应用索引模板“template_test”，并通过索引模板的配置与生命周期策略“rollover_workflow”相关联，实现当索引的大小达到1TB或索引创建超过1天时，自动进行滚动；当索引创建7天后，关闭数据副本；当索引创建30天后，删除该索引。

   如下索引是<test-{now/d}-000001>的URL编码，其创建时默认会带上当天的时间，例如当天日期是“2022.6.02”，创建出来的索引名称为“test-2022.06.02-000001”。

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

8. 使用别名“log_alias”写入数据，且写入时“log_alias”始终指向最后一个索引。

```
POST log_alias/_bulk
{"index":{}}
```



```
{"name":"name1"}
{"index":{}}
{"name":"name2"}
{"index":{}}
{"name":"name3"}
{"index":{}}
{"name":"name4"}
{"index":{}}
{"name":"name5"}
{"index":{}}
{"name":"name6"}
```

9. 查询数据，确认数据是否实现滚动索引。
   - 在索引创建一天后查看"test"开头的索引：
     ```
     GET _cat/indices/test*?s=i
     ```
     正常情况下会显示至少有两个索引，如下所示：
     ```
     green open test-<日期>-000001 r8ab5NX6T3Ox_hoGUanogQ 1 1 6 0 416b 208b
     green open test-<日期>-000002 sfwkVgy8RSSEw7W-xYjM2Q 1 1 0 0 209b 209b
     ```
     其中，“test-<日期>-000001”为7创建的索引，“test-<日期>-000002”为滚动生成的索引。
   - 查询别名“log_alias”关联的索引情况：
     ```
     GET _cat/aliases/log_alias?v
     ```
     正常情况下会显示该别名指向多个索引：
     ```
     alias     index               filter routing.index routing.search is_write_index
     log_alias test-<日期>-000001     -      -             -              false
     log_alias test-<日期>-000002     -      -             -              true
     ```

6.2 使用生命周期实现自动存算分离

方案概述

CSS支持存算分离，即将索引冻结到OBS来降低冷数据的存储成本。本文介绍如何使用索引生命周期管理，在特定的时间自动冻结索引，实现存算分离。

本案例通过配置生命周期策略，实现索引在创建3天后，自动被冻结，数据转储到OBS中；当索引创建7天后，删除该索引。

前提条件

- CSS集群处于可用状态。
- 集群版本为Elasticsearch 7.6.2及以上。

操作步骤

1. 登录云搜索服务管理控制台。
2. 在左侧导航栏，选择“集群管理”，进入Elasticsearch集群列表页面。
3. 在集群列表页面中，单击集群操作列的“Kibana”登录Kibana页面。
4. 在Kibana的左侧导航中选择“Dev Tools”，进入命令执行页面。
5. 创建生命周期策略“hot_warm_policy”。

   策略定义：当索引创建3天后，自动调用冻结索引API将数据转储到OBS；索引创建7天后，删除该索引。

   ```
   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. 新建索引模板“template_hot_warm”。

   模板定义：新建的所有“data”开头的索引会自动关联上生命周期策略“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"
   }
  }
 }
}
```


表 6-2 参数说明

参数	说明
number_of_shards	索引分片数
number_of_replicas	索引分片副本数
opendistro.index_state_management.policy_id	生命周期的策略名

7. 新建一个索引“data-2022-06-06”，该索引会自动应用索引模板“template_hot_warm”，并通过索引模板的配置与生命周期策略“hot_warm_policy”相关联，实现索引在创建3天后冻结，7天后删除。

```
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. 查询数据，确认数据是否实现自动存算分离。
   - 在索引创建三天后查看冻结的索引：

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

     正常情况下会显示3天以前的索引已经被冻结：

     ```
     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
     ```

   - 在索引创建7天后，查看索引情况，正常情况下7天以前的索引已经被删除。

7 实践案例

7.1 使用 CSS 加速数据库的查询分析

方案架构

关系型数据库（例如MySQL、GaussDB for MySQL等）受限于全文检索和Ad Hoc查询能力，因此会将Elasticsearch作为关系型数据库的补充，以此提升数据库的全文检索能力和高并发的Ad Hoc查询能力。

本章主要介绍，如何将MySQL数据库中的数据同步到云搜索服务CSS，通过CSS实现数据库的全文检索与Ad Hoc查询分析加速。方案架构图如图7-1所示。

图 7-1 CSS 加速数据库的查询分析方案

1. 用户业务数据存储到MySQL。
2. 通过数据复制服务DRS将MySQL中的数据实时同步到CSS。
3. 通过CSS进行全文检索与数据查询分析。

前提条件

- 已创建好安全模式的CSS集群和MySQL数据库，且两者在同一个VPC与安全组内。
- MySQL数据库中已经有待同步的数据。本章以如下表结构和初始数据举例。
  a. MySQL中创建一个学生信息表：
  ```
  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. MySQL中插入3个学生的初始数据：
  ```
  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');
```

- CSS集群中已完成索引创建，与MySQL中表相对应。

  本章集群的索引示例如下：

```
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"
        }
    }
  }
}
```

  其中的number_of_shards与number_of_replicas需根据具体业务场景进行配置。

操作步骤

步骤1 通过DRS将MySQL数据实时同步到CSS。具体操作步骤请参见**将MySQL同步到CSS/ES**。

在本章案例中，**表7-1**中的同步任务配置参数需要按建议填写。

表 7-1 同步任务参数说明

配置模块	参数名称	填写建议
同步实例 > 同步实例信息	“网络类型”	选择“VPC网络”。
	“源数据库实例”	选择需要同步的RDS for MySQL实例，即存储用户业务数据的MySQL。
	“同步实例所在子网”	选择同步实例所在的子网，建议跟数据库实例以及CSS集群所在子网保持一致。
源库及目标库 > 目标库信息	“VPC”和“子网”	选择和CSS集群一致的VPC与子网。
	“IP地址或域名”	填写CSS集群的IP地址，获取方式请参见**获取CSS集群的IP地址**。
	“数据库用户名”和“数据库密码”	填写CSS集群的管理员账户名（admin）和密码。

配置模块	参数名称	填写建议
	“加密证书”	选择CSS集群的安全证书，如果未启用“SSL安全链接”，则不用选择。获取方式请参见**获取CSS集群的安全证书**。
设置同步	“流速模式”	选择“不限速”。
	“同步对象类型”	不勾选“同步表结构”，因为已经预先在CSS集群中创建了与MySQL中表相对应的索引。
	“同步对象”	选择“表级同步”，选择与CSS对应的数据库以及表名。 **说明** 配置项中type名称需要与索引名称一样，都是“_doc”，如果不一致请修改。
数据加工	-	直接“下一步”。

启动同步任务后，等待任务“状态”从“全量同步”变成“增量同步”，表示数据进入实时同步状态。

步骤2 验证数据库的同步状态。

1. 全量数据同步验证。

   在CSS的Kibana中执行如下命令，确认全量数据是否同步到CSS。

   ```
   GET student/_search
   ```

2. 源端插入新数据，验证数据是否会同步到CSS。

   例如，源端插入“id”为“4”的新数据。

   ```
   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.')
   ```

   在CSS的Kibana中执行如下命令，确认新数据是否同步到CSS。

   ```
   GET student/_search
   ```

3. 源端更新数据，验证数据是否会同步更新到CSS。

   例如，更新“id”为“4”这条数据的“age”字段，从“50”改成“55”。

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

   在CSS的Kibana中执行如下命令，确认数据是否同步更新到CSS。

   ```
   GET student/_search
   ```

4. 源端删除数据，验证CSS里的数据是否同步删除。

   例如，删除“id”为“4”的数据。

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

   在CSS的Kibana中执行如下命令，确认CSS里的数据是否被同步删除。

   ```
   GET student/_search
   ```

步骤3 验证数据库的全文检索能力。

例如，在CSS查询“dsc”中包含“avatar”的数据。

```
GET student/_search
{
```



```
  "query": {
    "match": {
      "dsc": "avatar"
    }
  }
}
```

步骤4 验证数据库的Ad Hoc查询能力。

例如，在CSS查询年龄大于40的philanthropist。

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

步骤5 验证数据库的统计分析能力。

例如，在CSS统计所有人的年龄分布。

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

----结束

其他操作

- **获取CSS集群的IP地址**
  a. 在云搜索服务管理控制台，单击左侧导航栏的“集群管理”。
  b. 在集群管理列表页面，选择需要访问的集群，在“内网访问地址”列获取CSS集群的IP地址，一般是“<host>:<port>”或“<host>:<port>,<host>:<port>”样式。

     如果集群只有一个节点，此处仅显示1个节点的IP地址和端口号，例如“10.62.179.32:9200”；如果集群有多个节点，此处显示所有节点的IP地址和端口号，例如“10.62.179.32:9200,10.62.179.33:9200”。

- **获取CSS集群的安全证书**
  a. 登录云搜索服务控制台。
  b. 选择“集群管理”进入集群列表。
  c. 单击对应集群的名称，进入集群基本信息页面。
  d. 在“基本信息”页面，单击“HTTPS访问”后面的“下载证书”。

  图 7-2 下载证书

7.2 使用 CSS 搭建统一日志管理平台

使用CSS搭建的统一日志管理平台可以实时地、统一地、方便地管理日志，让日志驱动运维、运营等，提升服务管理效率。

方案架构

ELKB（Elasticsearch、Logstash、Kibana、Beats）提供了一整套日志场景解决方案，是目前主流的一种日志系统。其框架如图所示。

图 7-3 统一日志管理平台框架

- **Beats**是一个轻量级日志采集器，包括Filebeat、Metricbeat等。
- **Logstash**用于对日志进行搜集与预处理，支持多种数据源与ETL处理方式。
- **Elasticsearch**是个开源分布式搜索引擎，提供搜集、分析、存储数据等主要功能，CSS云搜索服务可以创建Elasticsearch集群。
- **Kibana**是一个可视化工具，可以基于Kibana进行Web可视化查询，并制作BI报表。

本章以CSS、Filebeat、Logstash和Kibana为例，搭建一个统一日志管理平台。使用Filebeat采集ECS中的日志，发送到Logstash进行数据处理，再存储到CSS中，最后通过Kibana进行日志的可视化查询与分析。

ELKB系统中各组件的版本兼容性请参见 **https://www.elastic.co/support/matrix#matrix_compatibility**。

前提条件

- 已创建好非安全模式的CSS集群。
- 已申请了ECS虚拟机，并安装了Java环境。

操作步骤

步骤1 部署并配置Filebeat。

1. 下载Filebeat，版本建议选择7.6.2。下载地址：**https://www.elastic.co/downloads/past-releases#filebeat-oss**
2. 配置Filebeat的配置文件“filebeat.yml”。

   例如，采集“/root/”目录下以“log”结尾的所有文件，配置文件“filebeat.yml”中的内容如下：

```
filebeat.inputs:
- type: log
  enabled: true
  # 采集的日志文件路径。
  paths:
    - /root/*.log

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

processors:
```

步骤2 部署并配置Logstash。

说明

为了获得更优的性能，Logstash中的JVM参数建议配置为ECS/docker中内存的一半。

1. 下载Logstash，版本建议选择7.6.2。下载地址：**https://www.elastic.co/downloads/past-releases#logstash-oss**

2. 确保Logstash与CSS集群的网络互通。
3. 配置Logstash的配置文件“logstash-sample.conf”。
   配置文件“logstash-sample.conf”中的内容如下：

```
input {
  beats {
    port => 5044
  }
}
# 对数据的切割与截取信息，
filter {
    grok {
        match => {
            "message" => '\[%{GREEDYDATA:timemaybe}\] \[%{WORD:level}\] %{GREEDYDATA:content}'
        }
    }
    mutate {
      remove_field => ["@version","tags","source","input","prospector","beat"]
    }
}
# CSS集群的信息
output {
  elasticsearch {
    hosts => ["http://192.168.0.4:9200"]
    index => "%{[@metadata][beat]}-%{+YYYY.MM.dd}"
    #user => "xxx"
    #password => "xxx"
  }
}
```

说明

Logstash的“filter”进行模式配置时，可以借助Grok Debugger（**http://grokdebug.herokuapp.com/**）。

步骤3 在Kibana上或通过API配置CSS集群的索引模板。

例如，创建一个索引模板，配置索引默认采用3分片、0副本，索引中定义了@timestamp、content、host.name、level、log.file.path、message、timemaybe等字段。

```
PUT _template/filebeat
{
  "index_patterns": ["filebeat*"],
  "settings": {
    # 定义分片数。
    "number_of_shards": 3,
    # 定义副本数。
    "number_of_replicas": 0,
    "refresh_interval": "5s"
  },
  # 定义字段。
  "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||EEE MMM dd HH:mm:ss zzz yyyy"
        }
      }
    }
}
```

步骤4 在ECS上准备测试数据。

执行如下命令，生成测试数据，并将数据写到“/root/tmp.log”中：

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

生成的测试数据样例如下：

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

步骤5 执行如下命令，启动Logstash。

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

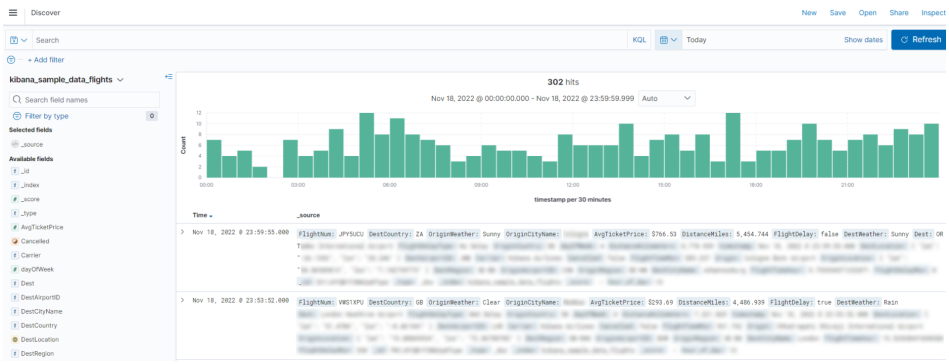
步骤6 执行如下命令，启动Filebeat。

```
./filebeat
```

步骤7 通过Kibana进行查询并制作报表。

1. 进入CSS集群的Kibana页面。
2. 选择“Discover”进行查询与分析，类似的效果如下：

图 7-4 Discover 界面示例



----结束

7.3 使用 Elasticsearch 集群自定义评分查询

通过Elasticsearch集群可以对匹配的文档进行评分。本文介绍如何进行自定义评分查询。

方案概述

自定义评分查询有两种方式。

- 用**绝对好评率**计算总分，按照总分由高到低的顺序排列出查询结果。
  总分 = 匹配得分 *（好评率 * 绝对因子）
  - 匹配得分：根据查询结果计分，内容匹配记1分，否则记0分，得分之和即为匹配得分。
  - 好评率：从匹配项的数据内容中获取好评率的值，一般指单条数据的评分。
  - 绝对因子：自定义的好评比例。
- 用**相对好评率**计算总分，按照总分由高到低的顺序排列查询结果。
  总分 = 匹配得分 *（好评率 * 相对分数）
  - 匹配得分：根据查询结果计分，内容匹配记1分，否则记0分，得分之和即为匹配得分。
  - 好评率：从匹配项的数据内容中获取好评率的值，一般指单条数据的评分。
  - 相对分数：自定义一个好评率阈值，当好评率大于阈值时，返回一个自定义的相对分数；当好评率小于等于阈值时，返回另一个自定义的相对分数。通过这种方式可以避免异常好评率对查询结果的影响。

前提条件

已经在云搜索服务管理控制台创建好Elasticsearch集群，且集群处于可用状态。

操作步骤

说明

本文的代码示例仅适用于Elasticsearch 7.x及以上版本的集群。

1. 登录云搜索服务管理控制台。
2. 在左侧导航栏，选择“集群管理”，进入Elasticsearch集群列表页面。
3. 在集群列表页面中，单击集群操作列的“Kibana”登录Kibana页面。
4. 在Kibana的左侧导航中选择“Dev Tools”，进入命令执行页面。
5. 创建索引，并指定自定义映射来定义数据类型。
   例如，数据文件“tv.json”的内容如下所示。

```
{
"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 }
]
}
```

   可以执行如下命令，创建索引“mall”，并指定自定义映射来定义数据类型。


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

6. 导入数据。

   执行如下命令，将“tv.json”文件中的数据导入到“mall”索引中。

```
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. 自定义评分查询数据。分别列举了绝对好评率和相对好评率查询方式。

   假设用户想要查询有USB接口、HDMI接口、DisplayPort接口的电视机，并根据好评率计算各款电视机的总分，根据总分由高到低的顺序排列结果。

   - 用绝对好评率计算总分

     总分的计算公式为“new_score = query_score * (vote * factor)”，执行的命令如下：

```
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
    }
  }
}
```


返回结果如下所示，按照总分由高到低的顺序排列查询结果。


```
{
  "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
        }
      }
    ]
  }
}
```


- 用相对好评率计算总分

总分的计算公式为“new_score = query_score * inline”，本示例中设置的好评率阈值为0.8，当vote>0.8时，inline取值为1；当vote≤0.8时，inline取值为0.5。执行命令如下：

```
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
    }
  }
}
```

返回结果如下所示，按照总分由高到低的顺序排列查询结果。

```
{
  "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
      }
    }
  ]
 }
}
```