

Cloudera Runtime 7.1.7 SP2

Securing Cloudera Search

Date published: 2015-05-05

Date modified: 2024-02-07

CLOUDERA

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Cloudera Search security aspects.....	4
Configure TLS/SSL encryption for Solr.....	4
Additional configuration steps when using a load balancer TLS/SSL for Solr HA.....	5
Cloudera Search authentication.....	6
Set proxy server authentication for clusters using Kerberos.....	6
Configure Kerberos authentication for Solr.....	7
Enable Kerberos authentication in Solr.....	8
Overview of proxy usage and load balancing for Search.....	8
Configuring custom Kerberos principals and custom system users for Solr.....	9
Enable LDAP authentication in Solr.....	11
Enabling Solr clients to authenticate with a secure Solr.....	12
Creating a JAAS configuration file.....	14
Enable Ranger authorization in Solr.....	15
Configuring Ranger authorization for Solr service.....	16
Enable Ranger document-level authorization for a Solr collection.....	17
Modify the schema and solrconfig files to enable document-level authorization.....	18
solrconfig.xml.secure file example.....	20
Configuring LDAP attribute-based document level authorization in Solr.....	41

Cloudera Search security aspects

Cloudera Search security covers the following security aspects:

- Securing network communication

Cloudera Search supports TLS for encrypting communications over a network.

For information on securing communications over a network, see [Encrypting Data in Transit](#).

- Authentication

Cloudera Search supports Kerberos and LDAP for authentication.

For information on enabling Kerberos for Cloudera Search, see [Configuring Authentication in Cloudera Manager](#).

- Authorization

Cloudera Search supports Apache Ranger for authorization.

For information on enabling Ranger for authorization, see [Using Ranger to Provide Authorization in CDP](#).

Related Information

[Enable Kerberos authentication in Solr](#)

[Enable Ranger authorization in Solr](#)

Configure TLS/SSL encryption for Solr

Although Cloudera recommends using AutoTLS, you also have the option to set up TLS manually for Cloudera Search.

Before you begin

Minimum required role: Configurator (Also provided by Cluster Administrator, Full Administrator)

- The Solr service must be running.
- Keystores for Solr must be readable by the solr user. This could be a copy of the Hadoop services' keystore with permissions 0440 and owned by the solr group.
- Truststores must have permissions 0444 (that is, readable by all).
- Specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the Solr service run. Therefore, the paths you choose must be valid on all hosts.
- In case there is a DataNode and a Solr server running on the same host, they can use the same certificate.

For more information on obtaining signed certificates and creating keystores, see [Encrypting Data in Transit](#). You can also view the upstream Solr documentation.

About this task

An additional consideration when configuring TLS/SSL for Solr HA is to allow clients to talk to Solr servers (the target servers) through the load balancer using TLS/SSL. To achieve this, you have to configure the load balancer for TLS/SSL pass-through, which means the load balancer does not perform encryption/decryption but simply passes traffic from clients and servers to the appropriate target host. See the documentation of your load balancer for details.

Procedure

1. Open the Cloudera Manager Admin Console and go to the Solr service.
2. Click the Configuration tab.
3. Select Scope All .

4. In the Search field, type TLS/SSL to show the Solr TLS/SSL properties.
5. Edit the following properties according to your cluster configuration.



Note: These values must be the same for all hosts running the Solr role.

Table 1: Solr TLS/SSL Properties

Property	Description
Enable TLS/SSL for Solr	Check this field to enable TLS for Solr.
Solr TLS/SSL Server JKS Keystore File Location	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Solr is acting as a TLS/SSL server. The keystore must be in JKS format.
Solr TLS/SSL Server JKS Keystore File Password	Password for the Solr JKS keystore.
Solr TLS/SSL Client Trust Store File	Required in case of self-signed or internal CA signed certificates. The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that Solr might connect to. This is used when Solr is the client in a TLS/SSL connection. This truststore must contain the certificate(s) used to sign the service(s) being connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Solr TLS/SSL Client Trust Store Password	The password for the Solr TLS/SSL Certificate Trust Store File. This password is not required to access the truststore: this field can be left blank. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

6. Enter a Reason for Change, and then click Save Changes to commit your changes.
7. Launch the Stale Configuration wizard to restart the Solr service and any dependent services.

What to do next

If Ranger authorization has been enabled for the Solr service, you need to update the Solr Collection URL (for a resource-based policy) or Solr URL (for a resource-based service) from `http://host_ip:8983/solr` to `https://host_ip:8985/solr` on the Ranger Admin Web UI.

Related Information

[Configure a resource-based policy: Solr](#)

[Configure a resource-based service: Solr](#)

[Encrypting Data in Transit](#)

[Enabling SSL](#)

Additional configuration steps when using a load balancer TLS/SSL for Solr HA

About this task

To configure a load balancer:

Procedure

1. Go to the Solr service.
2. Click the Configuration tab.
3. Select Scope Solr .

4. Enter the hostname and port number of the load balancer in the Solr Load Balancer property in the format host name:port number.

**Note:**

When you set this property, Cloudera Manager regenerates the keytabs for Solr roles. The principal in these keytabs contains the load balancer hostname.

If there are services that depend on this Solr service, such as Hue, those services use the load balancer to communicate with Solr.

5. Enter a Reason for change, and then click Save Changes to commit the changes.
6. Restart Solr and any dependent services or restart the entire cluster for this configuration to take effect.

Cloudera Search authentication

Cloudera Search continues to use simple authentication with the anonymous user as the default configuration, but Search also supports changing the authentication scheme to Kerberos. All required packages are installed during the installation or upgrade process. Additional configuration is required before Kerberos is available in your environment.

When authentication is enabled, only specified hosts and users can connect to Solr. Authentication also verifies that clients connect to legitimate servers. This feature prevents spoofing such as impersonation and person-in-the-middle attacks. Search supports Kerberos and LDAP authentication.

Cloudera Search supports a variety of combinations of authentication protocols:

Table 2: Authentication Protocol Combinations

Solr Authentication	Use Case
No authentication	Insecure cluster
Kerberos only	The Hadoop cluster has Kerberos turned on and every user (or client) connecting to Solr has a Kerberos principal.
Kerberos and LDAP	The Hadoop cluster has Kerberos turned on. External Solr users (or clients) do not have Kerberos principals but do have identities in the LDAP server. Client authentication using LDAP requires that Kerberos is enabled for the cluster. Using LDAP alone is not supported.

Once you are finished setting up authentication, configure Ranger authorization. Authorization involves specifying which resources can be accessed by particular users when they connect through Search. For more information, see *Using Ranger to Provide Authorization in CDP*.

Related Information

[Using Ranger to Provide Authorization in CDP](#)

Set proxy server authentication for clusters using Kerberos

In a cluster using Kerberos, applications check host credentials to verify that the host they are connecting to is the same one that is actually processing the request, to prevent person-in-the-middle attacks. To clarify that the load-balancing proxy server is legitimate, you need to perform these extra Kerberos setup steps.

About this task

This procedure assumes you are starting with a Kerberos-enabled cluster.

Procedure

1. Choose the host you will use for the proxy server. Based on the Kerberos setup procedure, it should already have an entry `solr/PROXY_HOST@REALM` in its keytab.
2. Navigate to Solr service Configuration Category Main .
3. Set the value of Solr Load Balancer to `<HOSTNAME>:<PORT>`, specifying the hostname and port of the proxy host.
4. Click Save Changes.
5. Launch the Stale Configuration wizard to restart the Solr service and any dependent services.

Cloudera Manager transparently handles the keytab and dependent service updates by setting `SOLR_AUTHENTICATION_KERBEROS_PRINCIPAL=*` under `/etc/default/solr` and by generating a merged keytab that includes the HTTP principal of the load balancer in addition to the own HTTP principal of the Solr server.

6. You can verify that the merged keytabs have been created and they contain the HTTP principal for both the load balancer and the particular Solr server by checking the process directory of Solr in `/var/run/cloudera-scm-agent/process`:

For example:

```
# klist -kte 291-solr-SOLR_SERVER/solr.keytab
Keytab name: FILE:291-solr-SOLR_SERVER/solr.keytab
KVNO Timestamp          Principal
-----
-----
    2 01/21/20 06:08:05 HTTP/loadbalancer.example.com@EXAMPLE.COM (des3-cbc-sha1)
    2 01/21/20 06:08:05 HTTP/loadbalancer.example.com@EXAMPLE.COM (arcfour-hmac)
    2 01/21/20 06:08:05 HTTP/loadbalancer.example.com@EXAMPLE.COM (des-hmac-sha1)
    2 01/21/20 06:08:05 HTTP/loadbalancer.example.com@EXAMPLE.COM (des-cbc-md5)
    2 01/21/20 06:08:05 HTTP/solrserver1.example.com@EXAMPLE.COM (des3-cbc-sha1)
    2 01/21/20 06:08:05 HTTP/solrserver1.example.com@EXAMPLE.COM (arcfour-hmac)
    2 01/21/20 06:08:05 HTTP/solrserver1.example.com@EXAMPLE.COM (des-hmac-sha1)
    2 01/21/20 06:08:05 HTTP/solrserver1.example.com@EXAMPLE.COM (des-cbc-md5)
    2 01/21/20 06:08:05 solr/solrserver1.example.com@EXAMPLE.COM (des3-cbc-sha1)
    2 01/21/20 06:08:05 solr/solrserver1.example.com@EXAMPLE.COM (arcfour-hmac)
    2 01/21/20 06:08:05 solr/solrserver1.example.com@EXAMPLE.COM (des-hmac-sha1)
    2 01/21/20 06:08:05 solr/solrserver1.example.com@EXAMPLE.COM (des-cbc-md5)
```

Related Information

[Enable Kerberos authentication in Solr](#)
[Stale Configurations](#)

Configure Kerberos authentication for Solr

Solr supports Kerberos authentication. All necessary packages are installed when you install Search.

To enable Kerberos, see [Configuring Authentication in Cloudera Manager](#).

Enable Kerberos authentication in Solr

Secure access to your Solr service by enabling Kerberos authentication.

About this task

Besides securing access to the Solr service, enabling Kerberos authentication is a prerequisite of both configuring LDAP authentication and Ranger authorization.

Before you begin

Solr supports Kerberos authentication. All necessary packages are installed when you install Search.

Kerberos authentication must be configured in Cloudera Manager for the cluster where Solr is deployed. For more information, see [Configuring Authentication in Cloudera Manager](#).

Procedure

1. In Cloudera Manager select the Solr service.
2. Select Configuration and find the Solr Secure Authentication property.
3. Select the Kerberos option.
4. Click Save Changes.
5. Restart the Solr service.

Results

Kerberos authentication for Solr is enabled.

Related Information

[Configuring Authentication in Cloudera Manager](#)

Overview of proxy usage and load balancing for Search

See the advantages of configuring a proxy server for the Solr service.

- Applications connect to a single well-known host and port, rather than keeping track of the hosts where the Solr service is running. This is especially useful for non-Java Solr clients such as web browsers or command-line tools such as curl.



Note: The Solr Java client (solrj) can inspect Zookeeper metadata to automatically locate the individual Solr servers, so load-balancing proxy support is not necessary.

- If any host running the Solr service becomes unavailable, application connection requests still succeed because you always connect to the proxy server rather than a specific host running the Solr server.
- Users can configure an SSL terminating proxy for Solr to secure the data exchanged with the external clients without requiring SSL configuration for the Solr cluster itself. This is relevant only if the Solr cluster is deployed on a trusted network and needs to communicate with clients that may not be on the same network. Many of the advantages of SSL offloading are described in [SSL Offloading, Encryption, and Certificates with NGINX](#).
- The "coordinator host" for each Search query potentially requires more memory and CPU cycles than the other hosts that process the query. The proxy server can issue queries using round-robin scheduling, so that each connection uses a different coordinator host. This load-balancing technique lets the hosts running the Solr service share this additional work, rather than concentrating it on a single machine.

Related Information[Set proxy server authentication for clusters using Kerberos](#)

Configuring custom Kerberos principals and custom system users for Solr

In a Kerberos enabled cluster, the Solr service uses the solr principal by default. Changing the default principal and using custom principals is supported. Principals can be configured on a service-wide level in Cloudera Manager with the Kerberos Principal property. To configure a custom system user, you need to modify the System User property.

Before you begin

Make sure you have the following privileges:

- SSH access to the cluster where you want to enable the custom principal
- administrative privileges in Cloudera Manager
- HDFS super user access

About this task

Important: Cloudera Manager configures CDP services to use the default Kerberos principal names. Cloudera recommends that you do not change the default Kerberos principal names. If it is unavoidable to do so, contact Cloudera Professional Services because it requires extensive additional custom configuration.



Important: Currently the names of system users which are impersonating users with Solr should match with the names of their respective Kerberos principals. If changing both the user name and the principal is not possible, you must add the user name you want to associate with the custom Kerberos principal to Solr configuration via the Solr Service Environment Advanced Configuration Snippet (Safety Valve) environment variable in Cloudera Manager.

Procedure

1. Stop the Solr service.
2. Disable ZooKeeper ACL checking temporarily.
 - a) In Cloudera Manager, navigate to **ZooKeeper Configuration**.
 - b) Find the **Java Configuration Options for ZooKeeper Server** property.
 - c) Add the following value:

```
-Dzookeeper.skipACL=yes
```
 - d) Click **Save Changes**.
 - e) Restart the ZooKeeper service.
3. In Cloudera Manager, navigate to **Clusters Solr service Configuration** and find the **Kerberos Principal** property.
4. Provide the custom Kerberos principal.
5. Click **Save Changes**.

6. To be able to interact with the Solr service, you must either change the System User name to match the custom Kerberos principal, or add the existing System User name to Solr Service Environment Advanced Configuration Snippet (Safety Valve).

Select one of the following options:

Option

Change the System User name to match the custom Kerberos principal

- a. In Cloudera Manager, navigate to Clusters Solr service Configuration and find the System User property.
- b. Change the user name to match the custom Kerberos principal you have set.
- c. Click Save Changes.

Keep the original System User name

- a. In Cloudera Manager navigate to Clusters Solr service Configuration and find the Solr Service Environment Advanced Configuration Snippet (Safety Valve) property.
- b. Look for the SOLR_SECURITY_PROXY_JAVA_OPTS key.
- c. Append its value with:


```
-Dsolr.security.proxyuser.[***SYSTEM_USER***].groups=* -Dsolr.security.proxyuser.[***SYSTEM_USER***].hosts=*
```

Replace [***SYSTEM_USER***] with the service user name you want to associate with the custom Kerberos principal.
- d. Click Save Changes.

7. Create a jaas.conf file containing the following:

```
Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=false
    useTicketCache=true
    principal
    ="[***CUSTOM_SOLR_KERBEROS_PRINCIPAL@KERBEROS_REALM_NAME***]";
};
```

Replace [***CUSTOM_SOLR_KERBEROS_PRINCIPAL@KERBEROS_REALM_NAME***] with your Kerberos principal and realm name.

8. Set the LOG4J_PROPS environment variable to a log4j.properties file:

```
export LOG4J_PROPS=/etc/zookeeper/conf/log4j.properties
```

9. Set the ZKCLI_JVM_FLAGS environment variable:

```
export ZKCLI_JVM_FLAGS="-Djava.security.auth.login.config=/path/to/jaas.conf \
-DzkACLProvider=org.apache.solr.common.cloud.SaslZkACLProvider \
-Droot.logger=INFO,console \
-Dsolr.authorization.superuser=[***CUSTOM_SOLR_KERBEROS_PRINCIPAL***]"
```

10. Authenticate as the `[***CUSTOM_SOLR_KERBEROS_PRINCIPAL***]`:

```
kinit [***CUSTOM_SOLR_KERBEROS_PRINCIPAL@KERBEROS_REALM_NAME***]
```

Replace `[***CUSTOM_SOLR_KERBEROS_PRINCIPAL@KERBEROS_REALM_NAME***]` with your Kerberos principal and realm name.

11. Run the `zkcli.sh` script as follows:

```
/opt/cloudera/parcels/CDH/lib/solr/bin/zkcli.sh -zkhost [***ZOOKEEPER_SERVER_HOSTNAME***]:[***ZOOKEEPER_SERVER_PORT***] -cmd updateacls /solr
```

Replace `[***ZOOKEEPER_SERVER_HOSTNAME***]` and `[***ZOOKEEPER_SERVER_PORT***]` with the hostname and port of a ZooKeeper server.

For example:

```
/opt/cloudera/parcels/CDH/lib/solr/bin/zkcli.sh -zkhost zk01.example.com:2181 -cmd updateacls /solr
```

12. Check ACLs in Zookeeper:

```
zookeeper-client -server ${HOSTNAME}:2181 getAcl /solr
```

13. Change ownership of Solr's HDFS Data Directory. Check the value in Cloudera Manager under Solr Configuration HDFS Data Directory .

14. Execute the following command as the HDFS superuser:

```
hdfs dfs -chown -R [***CUSTOM_SOLR_KERBEROS_PRINCIPAL***] [***HDFS_DATA_DIRECTORY***]
```

15. Re-enable ZooKeeper ACL check.

- In Cloudera Manager, navigate to ZooKeeper Configuration .
- Find the Java Configuration Options for ZooKeeper Server property.
- Remove the following value:

```
-Dzookeeper.skipACL=yes
```

- Click Save Changes.
- Restart the ZooKeeper service.

Enable LDAP authentication in Solr

You can configure LDAP-based authentication using Cloudera Manager at the Solr service level.

About this task

Solr supports LDAP authentication for external Solr clients including:

- Command-line tools
- curl
- Web browsers
- Solr Java clients

In some cases, Solr does not support LDAP authentication. Use Kerberos authentication instead in these cases. Solr does not support LDAP authentication with:

- Search indexing components including the MapReduce indexer and Lily HBase indexer.
- Solr internal requests such as those for replication or querying.
- Hadoop delegation token management requests such as GETDELEGATIONTOKEN or RENEWDELEGATIONTOKEN.

Before you begin

- Configuring LDAP authentication requires that Kerberos authentication is already configured and enabled in Solr.
- For secure LDAP connections, it is a prerequisite that TLS/SSL has been configured and enabled in Solr.

Procedure

1. In Cloudera Manager select the Solr service.
2. Click the Configuration tab.
3. Select Scope Solr .
4. Select Category Security .
5. Select Enable LDAP Authentication.
6. Enter the LDAP URL in the LDAP URL property.

To configure a TLS encrypted LDAP connection, select one of the following options:

- `ldaps://<LDAP_SERVER>:<PORT>`

The default port is 636.

OR

- `ldap://<LDAP_SERVER>:<PORT>`

The default port is 389.

Select Enable LDAP TLS. This is not required when using an LDAP URL with prefix `ldaps://`, because that already specifies TLS.

To configure LDAP with unencrypted transmission of usernames and passwords, set `ldap://<ldap_server>:<port>`, without setting Enable LDAP TLS.

7. Configure only one of following mutually exclusive parameters:
 - LDAP BaseDN: Replaces the username with a "distinguished name" (DN) of the form: `uid=userid,ldap_base DN`. Typically used for OpenLDAP server installation.
 - Active Directory Domain: Replaces the username with a string `username@ldap_domain`. Typically used for Active Directory server installation.
8. Launch the Stale Configuration wizard to restart the Solr service and any dependent services.

Related Information

[Stale Configurations Wizard](#)

Enabling Solr clients to authenticate with a secure Solr

As the process of enabling Solr clients to authenticate with a secure Solr is specific to the client,

Cloudera Search supports the following options:

- Using Kerberos and curl
- Using solrctl
- Using a jaas.conf File

- This enables technologies including:
 - Command line solutions
 - Java applications
 - The MapReduceIndexerTool

Secure Solr requires that the CDP components it interacts with are also secure. Secure Solr interacts with HDFS, ZooKeeper and optionally HBase, MapReduce, and NiFi.

Using Kerberos and curl

You can use Kerberos authentication with clients such as curl. To use curl, begin by acquiring valid Kerberos credentials and then run the desired command. For example, you might use commands similar to the following:

```
$ kinit -kt username.keytab username
$ curl --negotiate -u foo:bar http://solrserver:8983/solr/
```



Note: Depending on the tool used to connect, additional arguments may be required. For example, with curl, `--negotiate` and `-u` are required. The username and password specified with `-u` is not actually checked because Kerberos is used. As a result, any value such as `foo:bar` or even just `:` is acceptable. While any value can be provided for `-u`, note that the option is required. Omitting `-u` results in a 401 Unauthorized error, even though the `-u` value is not actually used.

Using solrctl

If you are using `solrctl` to manage your deployment in an environment that requires Kerberos authentication, you must have valid Kerberos credentials, which you can get using `kinit`.

Using a jaas.conf File

Some applications, such as those using the SolrJ library, require a Java Authentication and Authorization Service (JAAS) configuration file. You can use a file name other than `jaas.conf`, in the following examples `jaas-client.conf` is used.

Creating a JAAS configuration file:

- If you are authenticating using `kinit` to obtain credentials, you can configure the client to use your credentials cache by creating a `jaas-client.conf` file with the following contents:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=false
  useTicketCache=true
  principal="<USER>@EXAMPLE.COM";
};
```

Replace `<USER>` with your username, and `EXAMPLE.COM` with your Kerberos realm.

- If you want the client application to authenticate using a keytab, modify `jaas-client.conf` as follows:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="/path/to/user.keytab"
  storeKey=true
  useTicketCache=false
  principal="<user>@EXAMPLE.COM";
};
```

```
};
```

Replace `/PATH/TO/USER.KEYTAB` with the keytab file you want to use and `<USER>@EXAMPLE.COM` with the principal in the keytab. If you are using a service principal that includes the hostname, make sure that it is included in the `jaas.conf` file (for example, `solr/solr01.example.com@EXAMPLE.COM`).

Example usage of a JAAS configuration file:

- Command line

Set the property when invoking the program. For example, if you were using a jar, you might use:

```
java -Djava.security.auth.login.config=/home/user/jaas-client.conf -jar app.jar
```

- Java applications

Set the Java system property `java.security.auth.login.config`. For example, if the JAAS configuration file is located on the filesystem as `/home/user/jaas-client.conf`, the Java system property `java.security.auth.login.config` must be set to point to this file. Setting a Java system property can be done programmatically, for example using a call such as:

```
System.setProperty("java.security.auth.login.config", "/home/user/jaas-client.conf");
```

- The MapReduceIndexerTool

The MapReduceIndexerTool uses SolrJ to pass the JAAS configuration file. Using the MapReduceIndexerTool in a secure environment requires the use of the `HADOOP_OPTS` variable to specify the JAAS configuration file. For example, you might issue a command such as the following:

```
HADOOP_OPTS="-Djava.security.auth.login.config=/home/user/jaas-client.conf" \
hadoop jar MapReduceIndexerTool
```

- Configuring the hbase-indexer CLI

Certain hbase-indexer CLI commands such as `replication-status` attempt to read ZooKeeper hosts owned by HBase. To successfully use these commands in Solr in a secure environment, specify a JAAS configuration file with the HBase principal in the `HBASE_INDEXER_OPTS` environment variable. For example, you might issue a command such as the following:

```
HBASE_INDEXER_OPTS="-Djava.security.auth.login.config=/home/user/hbase-jas.conf" \
hbase-indexer replication-status
```

Related Information

[solrctl Reference](#)

Creating a JAAS configuration file

Certain applications, such as those using the SolrJ library, require a Java Authentication and Authorization Service (JAAS) configuration file.

- If you are authenticating using kinit to obtain credentials, you can configure the client to use your credentials cache by creating a JAAS file with the following contents:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=false
```

```
useTicketCache=true
principal="[***USER***]@[***REALM***]";
};
```

- If you want the client application to authenticate using a keytab, create a JAAS file with the following contents:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="[***PATH/TO/USER.KEYTAB***]"
  storeKey=true
  useTicketCache=false
  principal="[***USER***]/[***HOST NAME***]@[***REALM***]";
};
```

[*USER***]**

is a valid user name in your environment

/[*HOST NAME***]**

If you use a service principal that includes the host name, make sure that it is included in the jaas.conf file (for example, solr/solr01.example.com@EXAMPLE.COM).

[*REALM***]**

is your Kerberos realm

[*PATH/TO/USER.KEYTAB***]**

is the path to the keytab file you want to use

Enable Ranger authorization in Solr

Add a Ranger service to enable access control in Solr.

Before you begin

- Ranger authorization requires that Kerberos authentication is enabled in Solr.

About this task

Ranger restrictions are consistently applied regardless of the way users attempt to complete actions. For example, restricting access to data in a collection consistently restricts that access, whether queries come from the command line, from a browser, or through the admin console.

Procedure

1. In Cloudera Manager select the Solr service.
2. Select Configuration and find the RANGER Service property.
3. Select the Ranger service that you want this Solr service to depend on.
4. Click Save Changes.
5. Restart the Solr service.

Results

Ranger authorization for Solr is enabled. The Solr service depends on the selected Ranger service for authorization.

Related Information

[Configure a resource-based service: Solr](#)

[Configure a resource-based policy: Solr](#)

[Enable Ranger authorization on the Solr service used by Ranger for auditing](#)

Configuring Ranger authorization for Solr service

By default, the Solr service is not configured for authorization by Ranger. You can configure authorization manually, using Cloudera Manager.

Before you begin

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

Procedure

1. In the Cloudera Manager Admin Console, go to the **Ranger service**.
2. On the **Service Manager** page, click the Add icon next to Solr.
3. Enter the following information on the **Create Service** page:

Service Details

Field name	Description
Service Name	Assign a name to the Solr service you want to create. Note down the value you define here. You need to enter it later, when specifying the <code>ranger.plugin.solr.service.name</code> parameter in the Solr Service Advanced Configuration Snippet (Safety Valve) for <code>ranger-solr-security.xml</code> option.
Active Status	Enabled
Select Tag Service	Select <code>cm_tag</code> .

Configuration Properties

Field name	Description
Username	Assign a placeholder value. This property is not used in case of Kerberos authentication.
Password	Assign a placeholder value. This property is not used in case of Kerberos authentication.
Solr URL	Assign a placeholder value. This property is not used in case of Kerberos authentication.
Add new configurations	Add the following new configurations: • <code>policy.download.auth.users = solr</code> • <code>tag.download.auth.users = solr</code>



Note:

Do not click Test connection. In this use case, it is not necessary and it does not work.

4. Click Save.
5. Click on the name of the newly added service.
6. Under Action click the Edit icon.

7. In Allow Conditions Select User dropdown select hue.

This is necessary because the Hue service breaks if it has no permission to access Solr.

To keep the cluster accessible to non-admin users, you can add other users to the default policy or you can define additional policies.

8. Grant full admin privileges to users that you want to be able to access the Solr Admin UI.
 - a) Click Add under Allow Conditions to add a new condition, then add the user or users from the Select User drop-down.
 - b) Click Add Permissions then select the Select/Deselect All option.
 - c) Accept the selection, then click Save.
9. In the Cloudera Manager Admin Console, go to the Solr service.
10. Click the Configuration tab.
11. In the Search field start typing 'safety'
12. Click Add under Solr Service Advanced Configuration Snippet (Safety Valve).
13. Define the following:

Key

ranger.plugin.solr.service.name

Value

The Solr Service Name you assigned when creating the Solr service in Ranger.

14. Click Save Changes.
15. Click the Status tab to refresh the window.
16. Click the Stale Configuration: Restart needed indicator on top of the page.
17. Click Restart Stale Services.
18. Click Restart Now.

Enable Ranger document-level authorization for a Solr collection

By default, Ranger authorization works on collection level. Ranger allows you to configure document level security for individual Solr collections. This requires updating the solrconfig.xml file belonging to the particular collection. For the authorization to work on existing collections, you need to update the collection as well, by adding the `ranger_auth` parameter with an appropriate value to individual documents.



Important:

Document-level authorization does not prevent users from modifying documents or performing other update operations on the collection. Update operations are only governed by collection-level authorization.

Document-level authorization can exclusively be used to prevent documents being returned in query results. If users are not granted access to a document, those documents are not returned even if that user submits a query that matches those documents. This does not affect attempted updates.

Consequently, it is possible for a user to not have access to a set of documents based on document-level security, but to still be able to modify the documents using their collection-level authorization update rights. This means that a user can delete all documents in the collection. Similarly, a user might modify all documents, adding their authorization token to each one. After such a modification, the user could access any document using querying.

Therefore, if you restrict access using document-level security, consider granting collection-level update rights only to those users you trust and assume they will be able to access every document in the collection.

Prerequisites

1. Define or edit roles on Ranger UI and assign them to users/groups.
2. Create a new collection configuration or download an existing one for editing.

Enabling document-level authorization

1. Add `ranger_auth` field to the schema file and add hooks to `solrconfig.xml` that trigger document-level authorization.
2. Disable Ranger authorization.
3. Upload the configuration to ZooKeeper.
4. If you updated an existing collection to enable document-level authorization, add appropriate values to the newly created `ranger_auth` field before you turn on Ranger authorization. For example, you can reindex your collection, using one of the batch-indexing options offered by Cloudera Search.
5. To take document-level authorization into use, create a collection using the updated configuration (new collections) or reload your collection (updating an existing collection).
6. Enable Ranger authorization.

Modify the schema and solrconfig files to enable document-level authorization

To enable document-level Ranger authorization, you need to add a field in the schema file determining the roles that can access a particular document in the collection. You also need to edit the `solrconfig` file to include the hooks that trigger Ranger document level authorization based on the schema field values.

Before you begin

1. Disable Ranger authorization.
2. Define or edit roles on Ranger UI and assign them to users/groups.
3. Create a new collection configuration or download an existing one for editing.

Procedure

1. If you are using Kerberos, kinit as a user with sufficient rights to create or modify collections:

```
kinit [***KERBEROS PRINCIPAL***]@[***EXAMPLE.COM***]
```

Replace `[***KERBEROS PRINCIPAL***]@[***EXAMPLE.COM***]` with your Kerberos principal and your Kerberos realm name respectively.

2. Go to the `conf` subdirectory of the newly created/downloaded folder to edit the schema file. (In our example it is `mycollection/conf`)

The file name is either `managed-schema` or `schema.xml` based on the schema factory used.

3. Add a field that determines which roles have access to a particular document. In this example we name this field `ranger_auth`.

Add the following to the list of fields in your schema, making sure the value of the `type` property is `string`; the values of the `indexed`, `stored`, and `multiValued` properties are `true`:

```
<field name="ranger_auth" type="string" indexed="true" stored="true" required="false" multiValued="true"/>
```

4. Open the `solrconfig.xml` file for editing

5. Locate the section which contains the list of search components. In the default configuration it starts with a comment block similar to this:

```
<!-- Search Components
      Search components are registered to SolrCore and used by
      instances of SearchHandler (which can access them by name)
      ...
-->
```

6. Add a new SearchComponent:

```
<searchComponent name="queryDocAuthorization" class="org.apache.ranger.a
uthorization.solr.authorizer.RangerSolrAuthorizer">
  <str name="enabled">true</str>
  <!-- The field which contains the role or list of roles which are all
owed to query a particular document -->
  <str name="rangerAuthField">ranger_auth</str>
  <!-- If the rangerAuthField contains this value, all roles will be all
owed to query that particular document -->
  <str name="allRolesToken">*</str>
</searchComponent>
```

7. Add queryDocAuthorization to the first-components array of the /query, /get, /browse, /tvrh, /terms, and /elevate request handlers as well, in case they are present in your solrconfig.xml.

Locate the request handler /select section:

```
<requestHandler name="/select" class="solr.SearchHandler">
  <!-- default values for query parameters can be specified, these
will be overridden by parameters in the request
-->
  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>
```

To the end of this section, before the closing tag, insert queryDocAuthorization to the first-components array:

```
<arr name="first-components">
  <str>queryDocAuthorization</str>
</arr>
</requestHandler>
```



Note: /get (real-time get request handler) is implicitly defined even if it does not appear in solrconfig.xml. In case of using document level security, Cloudera recommends to also protect the /get handler. To do that, you need to explicitly add it to the solrconfig.xml and define queryDocAuthorization as a first-component:

```
<requestHandler name="/get" class="solr.RealTimeGetHandler">
  <lst name="defaults">
    <str name="omitHeader">true</str>
    <str name="wt">json</str>
    <str name="indent">true</str>
  </lst>
  <arr name="first-components">
    <str>queryDocAuthorization</str>
  </arr>
</requestHandler>
```

8. Locate the requestParsers section:

```
<requestParsers enableRemoteStreaming="true"
multipartUploadLimitInKB="2048000"
```

```
...
```

Ensure that this section has a boolean attribute called `addHttpRequestToContext` with a value of `true`:

```
<requestParsers enableRemoteStreaming="true"
  multipartUploadLimitInKB="2048000"
  formdataUploadLimitInKB="2048"
  addHttpRequestToContext="true"/>
```

What to do next

1. Upload the configuration metadata to ZooKeeper.
2. Populate the `ranger_auth` field for each document with roles you have defined in Ranger.
3. Create a new collection using the the updated configuration metadata, or update the configuration of an existing collection.
4. Enable Ranger authorization.

Related Information

[Enable Ranger document-level authorization for a Solr collection](#)

solrconfig.xml.secure file example

You can copy this xml file for editing by clicking the Copy to clipboard icon.

solrconfig.xml.secure

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements.  See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License.  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<!--
For more details about configurations options that may appear in
this file, see http://wiki.apache.org/solr/SolrConfigXml.
-->
<config>
  <!-- In all configuration below, a prefix of "solr." for class names
       is an alias that causes solr to search appropriate packages,
       including org.apache.solr.(search|update|request|core|analysis)

       You may also specify a fully qualified Java classname if you
       have your own custom plugins.
  -->
  <!-- Controls what version of Lucene various components of Solr
       adhere to.  Generally, you want to use the latest version to
       get all bug fixes and improvements.  It is highly recommended
       that you fully re-index after changing this setting as it can
```

```

    affect both how text is indexed and queried.
-->
<.luceneMatchVersion>8.4.1</luceneMatchVersion>

<!-- <lib/> directives can be used to instruct Solr to load any Jars
identified and use them to resolve any "plugins" specified in
your solrconfig.xml or schema.xml (ie: Analyzers, Request
Handlers, etc...).

All directories and paths are resolved relative to the
instanceDir.

Please note that <lib/> directives are processed in the order
that they appear in your solrconfig.xml file, and are "stacked"
on top of each other when building a ClassLoader - so if you have
plugin jars with dependencies on other jars, the "lower level"
dependency jars should be loaded first.

If a "./lib" directory exists in your instanceDir, all files
found in it are included as if you had used the following
syntax...
    <lib dir="./lib" />
-->

<!-- A 'dir' option by itself adds any files found in the directory
to the classpath, this is useful for including all jars in a
directory.
When a 'regex' is specified in addition to a 'dir', only the
files in that directory which completely match the regex
(anchored on both ends) will be included.

If a 'dir' option (with or without a regex) is used and nothing
is found that matches, a warning will be logged.

The example below can be used to load a solr-contrib along
with their external dependencies.
-->
<!-- <lib dir="${solr.install.dir:../../../../}/dist/" regex="solr-ltr-
\d.*\.jar" /> -->

<!-- an exact 'path' can be used instead of a 'dir' to specify a
specific jar file. This will cause a serious error to be logged
if it can't be loaded.
-->
<!--
    <lib path="../../../a-jar-that-does-not-exist.jar" />
-->

<!-- Data Directory

Used to specify an alternate directory to hold all index data
other than the default ./data under the Solr home. If
replication is in use, this should match the replication
configuration.
-->
<dataDir>${solr.data.dir:}</dataDir>

<!-- The DirectoryFactory to use for indexes.
solr.StandardDirectoryFactory is filesystem
based and tries to pick the best implementation for the current
JVM and platform. solr.NRTCachingDirectoryFactory, the default,
wraps solr.StandardDirectoryFactory and caches small files in memory
for better NRT performance.

```

```

    One can force a particular implementation via solr.MMapDirectoryFactory,
    solr.NIOFSDirectoryFactory, or solr.SimpleFSDirectoryFactory.
    solr.RAMDirectoryFactory is memory based and not persistent.
    -->

    <directoryFactory name="DirectoryFactory" class="${solr.directoryFactory:org.apache.solr.core.HdfsDirectoryFactory}">
      <str name="solr.hdfs.home">${solr.hdfs.home:}</str>
      <str name="solr.hdfs.confdir">${solr.hdfs.confdir:}</str>
      <str name="solr.hdfs.security.kerberos.enabled">${solr.hdfs.security.kerberos.enabled:false}</str>
      <str name="solr.hdfs.security.kerberos.keytabfile">${solr.hdfs.security.kerberos.keytabfile:}</str>
      <str name="solr.hdfs.security.kerberos.principal">${solr.hdfs.security.kerberos.principal:}</str>
      <bool name="solr.hdfs.blockcache.enabled">${solr.hdfs.blockcache.enabled:true}</bool>
      <!-- Enable/Disable using one global cache for all SolrCores.
      The settings used will be from the first HdfsDirectoryFactory created.
      -->
      <str name="solr.hdfs.blockcache.global">${solr.hdfs.blockcache.global:true}</str>
      <int name="solr.hdfs.blockcache.slab.count">${solr.hdfs.blockcache.slab.count:1}</int>
      <bool name="solr.hdfs.blockcache.direct.memory.allocation">${solr.hdfs.blockcache.direct.memory.allocation:true}</bool>
      <int name="solr.hdfs.blockcache.blocksperbank">${solr.hdfs.blockcache.blocksperbank:16384}</int>
      <bool name="solr.hdfs.blockcache.read.enabled">${solr.hdfs.blockcache.read.enabled:true}</bool>
      <bool name="solr.hdfs.blockcache.write.enabled">${solr.hdfs.blockcache.write.enabled:false}</bool>
      <!-- the buffercount is actually the total size in bytes to used for buffer caching -->
      <int name="solr.hdfs.blockcache.bufferstore.buffercount">${solr.hdfs.blockcache.bufferstore.buffercount:0}</int>
      <bool name="solr.hdfs.nrtcachingdirectory.enable">${solr.hdfs.nrtcachingdirectory.enable:true}</bool>
      <int name="solr.hdfs.nrtcachingdirectory.maxmergesizeemb">${solr.hdfs.nrtcachingdirectory.maxmergesizeemb:16}</int>
      <int name="solr.hdfs.nrtcachingdirectory.maxcachedmb">${solr.hdfs.nrtcachingdirectory.maxcachedmb:192}</int>
      <!-- HDFS Block Locality Reporter can be toggled on and off -->
      <bool name="solr.hdfs.locality.metrics.enabled">${solr.hdfs.locality.metrics.enabled:false}</bool>
    </directoryFactory>
    <!-- The CodecFactory for defining the format of the inverted index.
    The default implementation is SchemaCodecFactory, which is the official Lucene
    index format, but hooks into the schema to provide per-field customization of
    the postings lists and per-document values in the fieldType element (postingsFormat/docValuesFormat). Note that most of the alternative
    implementations
    are experimental, so if you choose to customize the index format, it's a good
    idea to convert back to the official format e.g. via IndexWriter.addIndexes(IndexReader)
    before upgrading to a newer version to avoid unnecessary reindexing.
    A "compressionMode" string element can be added to <codecFactory> to choose

```

```

    between the existing compression modes in the default codec: "BEST_S
PEED" (default)
    or "BEST_COMPRESSION".
    -->
    <codecFactory class="solr.SchemaCodecFactory"/>

    <!-- -----
    Index Config - These settings control low-level behavior of indexing
    Most example settings here show the default value, but are commented
    out, to more easily see where customizations have been made.
    Note: This replaces <indexDefaults> and <mainIndex> from older versi
ons
    -----
    -->
    <indexConfig>
    <!-- maxFieldLength was removed in 4.0. To get similar behavior, include
a
        LimitTokenCountFilterFactory in your fieldType definition. E.g.
        <filter class="solr.LimitTokenCountFilterFactory" maxTokenCount="10000
"/>
    -->
    <!-- Maximum time to wait for a write lock (ms) for an IndexWriter. Defa
ult: 1000 -->
    <!-- <writeLockTimeout>1000</writeLockTimeout> -->

    <!-- Expert: Enabling compound file will use less files for the index,
using fewer file descriptors on the expense of performance decrea
se.
        Default in Lucene is "true". Default in Solr is "false" (since 3.6)
    -->
    <!-- <useCompoundFile>false</useCompoundFile> -->

    <!-- ramBufferSizeMB sets the amount of RAM that may be used by Lucene
indexing for buffering added documents and deletions before they
are
        flushed to the Directory.
        maxBufferedDocs sets a limit on the number of documents buffered
        before flushing.
        If both ramBufferSizeMB and maxBufferedDocs is set, then
        Lucene will flush based on whichever limit is hit first. -->
    <ramBufferSizeMB>128</ramBufferSizeMB>
    <!-- <maxBufferedDocs>1000</maxBufferedDocs> -->

    <!-- Expert: ramPerThreadHardLimitMB sets the maximum amount of RAM that
can be consumed
        per thread before they are flushed. When limit is exceeded, this
        triggers a forced
        flush even if ramBufferSizeMB has not been exceeded.
        This is a safety limit to prevent Lucene's DocumentsWriterPerThread
        from address space
        exhaustion due to its internal 32 bit signed integer based memory
        addressing.
        The specified value should be greater than 0 and less than 2048MB.
        When not specified,
        Solr uses Lucene's default value 1945. -->
    <!-- <ramPerThreadHardLimitMB>1945</ramPerThreadHardLimitMB> -->

    <!-- Expert: Merge Policy
        The Merge Policy in Lucene controls how merging of segments is done
        .
        The default since Solr/Lucene 3.3 is TieredMergePolicy.
        The default since Lucene 2.3 was the LogByteSizeMergePolicy,
        Even older versions of Lucene used LogDocMergePolicy.
    -->

```

```

<!--
  <mergePolicyFactory class="org.apache.solr.index.TieredMergePolicyFactory">
    <int name="maxMergeAtOnce">10</int>
    <int name="segmentsPerTier">10</int>
    <double name="noCFSRatio">0.1</double>
  </mergePolicyFactory>
-->

<!-- Expert: Merge Scheduler
  The Merge Scheduler in Lucene controls how merges are
  performed. The ConcurrentMergeScheduler (Lucene 2.3 default)
  can perform merges in the background using separate threads.
  The SerialMergeScheduler (Lucene 2.2 default) does not.
-->
<!--
  <mergeScheduler class="org.apache.lucene.index.ConcurrentMergeScheduler"/>
-->

<!-- LockFactory

  This option specifies which Lucene LockFactory implementation
  to use.

  single = SingleInstanceLockFactory - suggested for a
    read-only index or when there is no possibility of
    another process trying to modify the index.
  native = NativeFSLockFactory - uses OS native file locking.
    Do not use when multiple solr webapps in the same
    JVM are attempting to share a single index.
  simple = SimpleFSLockFactory - uses a plain file for locking

  Defaults: 'native' is default for Solr3.6 and later, otherwise
    'simple' is the default

  More details on the nuances of each LockFactory...
  http://wiki.apache.org/lucene-java/AvailableLockFactories
-->
<lockType>${solr.lock.type:hdfs}</lockType>

<!-- Commit Deletion Policy
  Custom deletion policies can be specified here. The class must
  implement org.apache.lucene.index.IndexDeletionPolicy.

  The default Solr IndexDeletionPolicy implementation supports
  deleting index commit points on number of commits, age of
  commit point and optimized status.

  The latest commit point should always be preserved regardless
  of the criteria.
-->
<!--
  <deletionPolicy class="solr.SolrDeletionPolicy">
-->
<!-- The number of commit points to be kept -->
<!-- <str name="maxCommitsToKeep">1</str> -->
<!-- The number of optimized commit points to be kept -->
<!-- <str name="maxOptimizedCommitsToKeep">0</str> -->
<!--
  Delete all commit points once they have reached the given age.
  Supports DateMathParser syntax e.g.
-->
<!--

```

```

    <str name="maxCommitAge">30MINUTES</str>
    <str name="maxCommitAge">1DAY</str>
  -->
<!--
</deletionPolicy>
-->

<!-- Lucene InfoStream

    To aid in advanced debugging, Lucene provides an "InfoStream"
    of detailed information when indexing.

    Setting The value to true will instruct the underlying Lucene
    IndexWriter to write its debugging info the specified file
  -->
  <!-- <infoStream file="INFOSTREAM.txt">false</infoStream> -->
</indexConfig>

<!-- JMX

    This example enables JMX if and only if an existing MBeanServer
    is found, use this if you want to configure JMX through JVM
    parameters. Remove this to disable exposing Solr configuration
    and statistics to JMX.

    For more details see http://wiki.apache.org/solr/SolrJmx
  -->
<jmx />
<!-- If you want to connect to a particular server, specify the
    agentId
  -->
<!-- <jmx agentId="myAgent" /> -->
<!-- If you want to start a new MBeanServer, specify the serviceUrl -->
<!-- <jmx serviceUrl="service:jmx:rmi:///jndi/rmi://localhost:9999/solr"/>
  -->

<!-- The default high-performance update handler -->
<updateHandler class="solr.DirectUpdateHandler2">

  <!-- Enables a transaction log, used for real-time get, durability, and
    and solr cloud replica recovery. The log can grow as big as
    uncommitted changes to the index, so use of a hard autoCommit
    is recommended (see below).
    "dir" - the target directory for transaction logs, defaults to the
    solr data directory.
    "numVersionBuckets" - sets the number of buckets used to keep
    track of max version values when checking for re-ordered
    updates; increase this value to reduce the cost of
    synchronizing access to version buckets during high-volume
    indexing, this requires 8 bytes (long) * numVersionBuckets
    of heap space per Solr core.
  -->
  <updateLog>
    <str name="dir">${solr.ulog.dir}</str>
    <int name="tlogDfsReplication">${solr.ulog.tlogDfsReplication:3}</int>
    <int name="numVersionBuckets">${solr.ulog.numVersionBuckets:65536}</
int>
  </updateLog>

  <!-- AutoCommit
    Perform a hard commit automatically under certain conditions.
    Instead of enabling autoCommit, consider using "commitWithin"
    when adding documents.

```

```

http://wiki.apache.org/solr/UpdateXmlMessages

maxDocs - Maximum number of documents to add since the last
         commit before automatically triggering a new commit.

maxTime - Maximum amount of time in ms that is allowed to pass
         since a document was added before automatically
         triggering a new commit.
openSearcher - if false, the commit causes recent index changes
               to be flushed to stable storage, but does not cause a new
               searcher to be opened to make those changes visible.

If the updateLog is enabled, then it's highly recommended to
have some sort of hard autoCommit to limit the log size.
-->
<autoCommit>
  <maxTime>${solr.autoCommit.maxTime:60000}</maxTime>
  <openSearcher>false</openSearcher>
</autoCommit>
<!-- softAutoCommit is like autoCommit except it causes a
      'soft' commit which only ensures that changes are visible
      but does not ensure that data is synced to disk. This is
      faster and more near-realtime friendly than a hard commit.
-->

<autoSoftCommit>
  <maxTime>${solr.autoSoftCommit.maxTime:15000}</maxTime>
</autoSoftCommit>

<!-- Update Related Event Listeners

Various IndexWriter related events can trigger Listeners to
take actions.

postCommit - fired after every commit or optimize command
postOptimize - fired after every optimize command
-->
<!-- The RunExecutableListener executes an external command from a
hook such as postCommit or postOptimize.

exe - the name of the executable to run
dir - dir to use as the current working directory. (default=".")
wait - the calling thread waits until the executable returns.
      (default="true")
args - the arguments to pass to the program. (default is none)
env - environment variables to set. (default is none)
-->
<!-- This example shows how RunExecutableListener could be used
with the script based replication...
http://wiki.apache.org/solr/CollectionDistribution
-->
<!--
  <listener event="postCommit" class="solr.RunExecutableListener">
    <str name="exe">solr/bin/snapshooter</str>
    <str name="dir">.</str>
    <bool name="wait">true</bool>
    <arr name="args"> <str>arg1</str> <str>arg2</str> </arr>
    <arr name="env"> <str>MYVAR=val1</str> </arr>
  </listener>
-->

</updateHandler>

```

```

<!-- IndexReaderFactory

Use the following format to specify a custom IndexReaderFactory,
which allows for alternate IndexReader implementations.

** Experimental Feature **

Please note - Using a custom IndexReaderFactory may prevent
certain other features from working. The API to
IndexReaderFactory may change without warning or may even be
removed from future releases if the problems cannot be
resolved.

** Features that may not work with custom IndexReaderFactory **
The ReplicationHandler assumes a disk-resident index. Using a
custom IndexReader implementation may cause incompatibility
with ReplicationHandler and may cause replication to not work
correctly. See SOLR-1366 for details.

-->
<!--
<indexReaderFactory name="IndexReaderFactory" class="package.class">
  <str name="someArg">Some Value</str>
</indexReaderFactory >
-->

<!-- ~~~~~~
Query section - these settings control query time things like caches
~~~~~
-->
<query>

<!-- Maximum number of clauses allowed when parsing a boolean query string.

This limit only impacts boolean queries specified by a user as part
of a query string,
and provides per-collection controls on how complex user specified
boolean queries can
be. Query strings that specify more clauses then this will result
in an error.

If this per-collection limit is greater then the global `maxBooleanClauses` limit
specified in `solr.xml`, it will have no effect, as that setting al
so limits the size
of user specified boolean queries.
-->
<maxBooleanClauses>${solr.max.booleanClauses:1024}</maxBooleanClauses>
<!-- Solr Internal Query Caches

There are two implementations of cache available for Solr,
LRUCache, based on a synchronized LinkedHashMap, and
FastLRUCache, based on a ConcurrentHashMap.

FastLRUCache has faster gets and slower puts in single
threaded operation and thus is generally faster than LRUCache
when the hit ratio of the cache is high (> 75%), and may be
faster under other scenarios on multi-cpu systems.
-->

<!-- Filter Cache

Cache used by SolrIndexSearcher for filters (DocSets),

```

unordered sets of **all** documents that match a query. When a new searcher is opened, its caches may be prepopulated or "autowarmed" using data from caches in the old searcher. `autowarmCount` is the number of items to prepopulate. For `LRUCache`, the autowarmed items will be the most recently accessed items.

Parameters:

- `class` - the `SolrCache` implementation `LRUCache` or `(LRUCache or FastLRUCache)`
- `size` - the maximum number of entries in the cache
- `initialSize` - the initial capacity (number of entries) of the cache. (see `java.util.HashMap`)
- `autowarmCount` - the number of entries to prepopulate from and old cache.
- `maxRamMB` - the maximum amount of RAM (in MB) that this cache is allowed to occupy. Note that when this option is specified, the `size` and `initialSize` parameters are ignored.

```
-->
<filterCache class="solr.FastLRUCache"
  size="512"
  initialSize="512"
  autowarmCount="0"/>

<!-- Query Result Cache

Caches results of searches - ordered lists of document ids
(DocList) based on a query, a sort, and the range of documents requested.
Additional supported parameter by LRUCache:
  maxRamMB - the maximum amount of RAM (in MB) that this cache is allowed
to occupy
-->
<queryResultCache class="solr.LRUCache"
  size="512"
  initialSize="512"
  autowarmCount="0"/>

<!-- Document Cache

Caches Lucene Document objects (the stored fields for each document). Since Lucene internal document ids are transient, this cache will not be autowarmed.
-->
<documentCache class="solr.LRUCache"
  size="512"
  initialSize="512"
  autowarmCount="0"/>

<!-- custom cache currently used by block join -->
<cache name="perSegFilter"
  class="solr.search.LRUCache"
  size="10"
  initialSize="0"
  autowarmCount="10"
  regenerator="solr.NoOpRegenerator" />

<!-- Field Value Cache

Cache used to hold field values that are quickly accessible by document id. The fieldValueCache is created by default even if not configured here.
```

```

-->
<!--
  <fieldValueCache class="solr.FastLRUCache"
                  size="512"
                  autowarmCount="128"
                  showItems="32" />

-->

<!-- Custom Cache
  Example of a generic cache. These caches may be accessed by
  name through SolrIndexSearcher.getCache(),cacheLookup(), and
  cacheInsert(). The purpose is to enable easy caching of
  user/application level data. The regenerator argument should
  be specified as an implementation of solr.CacheRegenerator
  if autowarming is desired.
-->
<!--
  <cache name="myUserCache"
         class="solr.LRUCache"
         size="4096"
         initialSize="1024"
         autowarmCount="1024"
         regenerator="com.mycompany.MyRegenerator"
         />

-->

<!-- Lazy Field Loading

  If true, stored fields that are not requested will be loaded
  lazily. This can result in a significant speed improvement
  if the usual case is to not load all stored fields,
  especially if the skipped fields are large compressed text
  fields.

-->
<enableLazyFieldLoading>true</enableLazyFieldLoading>

<!-- Use Filter For Sorted Query

  A possible optimization that attempts to use a filter to
  satisfy a search. If the requested sort does not include
  score, then the filterCache will be checked for a filter
  matching the query. If found, the filter will be used as the
  source of document ids, and then the sort will be applied to
  that.

  For most situations, this will not be useful unless you
  frequently get the same search repeatedly with different sort
  options, and none of them ever use "score"
-->
<!--
  <useFilterForSortedQuery>true</useFilterForSortedQuery>
-->

<!-- Result Window Size
  An optimization for use with the queryResultCache. When a search
  is requested, a superset of the requested number of document ids
  are collected. For example, if a search for a particular query
  requests matching documents 10 through 19, and queryWindowSize is
50,
  then documents 0 through 49 will be collected and cached. Any fu
rther
  requests in that range can be satisfied via the cache.
-->
<queryResultWindowSize>20</queryResultWindowSize>

```

```

<!-- Maximum number of documents to cache for any entry in the
      queryResultCache.
-->
<queryResultMaxDocsCached>200</queryResultMaxDocsCached>
<!-- Query Related Event Listeners
      Various IndexSearcher related events can trigger Listeners to
      take actions.

      newSearcher - fired whenever a new searcher is being prepared
      and there is a current searcher handling requests (aka
      registered). It can be used to prime certain caches to
      prevent long request times for certain requests.

      firstSearcher - fired whenever a new searcher is being
      prepared but there is no current registered searcher to handle
      requests or to gain autowarming data from.

-->
<!-- QuerySenderListener takes an array of NamedList and executes a
      local query request for each NamedList in sequence.
-->
<listener event="newSearcher" class="solr.QuerySenderListener">
  <arr name="queries">
    <!--
      <lst><str name="q">solr</str><str name="sort">price asc</str></
lst>
      <lst><str name="q">rocks</str><str name="sort">weight asc</str></
lst>
    -->
  </arr>
</listener>
<listener event="firstSearcher" class="solr.QuerySenderListener">
  <arr name="queries">
    <!--
      <lst>
        <str name="q">static firstSearcher warming in solrconfig.xml</str>
      </lst>
    -->
  </arr>
</listener>

<!-- Use Cold Searcher

      If a search request comes in and there is no current
      registered searcher, then immediately register the still
      warming searcher and use it. If "false" then all requests
      will block until the first searcher is done warming.
-->
<useColdSearcher>false</useColdSearcher>
<!-- Slow Query Request Logging

      Any queries that take longer than the specified threshold
      will be logged as "slow" queries.
      To disable slow request logging for this Solr config,
      set the value to -1
-->
<slowQueryThresholdMillis>5000</slowQueryThresholdMillis>

</query>

<!-- Request Dispatcher

```

This section contains instructions for how the SolrDispatchFilter should behave when processing requests for this SolrCore.

```
-->
<requestDispatcher>
  <!-- Request Parsing

    These settings indicate how Solr Requests may be parsed, and
    what restrictions may be placed on the ContentStreams from
    those requests
    enableRemoteStreaming - enables use of the stream.file
    and stream.url parameters for specifying remote streams.

    multipartUploadLimitInKB - specifies the max size (in KiB) of
    Multipart File Uploads that Solr will allow in a Request.

    formdataUploadLimitInKB - specifies the max size (in KiB) of
    form data (application/x-www-form-urlencoded) sent via
    POST. You can use POST to pass request parameters not
    fitting into the URL.

    addHttpRequestToContext - if set to true, it will instruct
    the requestParsers to include the original HttpServletRequest
    object in the context map of the SolrQueryRequest under the
    key "httpRequest". It will not be used by any of the existing
    Solr components, but may be useful when developing custom
    plugins.
    *** WARNING ***
    Before enabling remote streaming, you should make sure your
    system has authentication enabled.

  -->
  <requestParsers enableRemoteStreaming="true"
    multipartUploadLimitInKB="2048000"
    formdataUploadLimitInKB="2048"
    addHttpRequestToContext="true"/>

  <!-- HTTP Caching

    Set HTTP caching related parameters (for proxy caches and clients).

    The options below instruct Solr not to output any HTTP Caching
    related headers
  -->
  <httpCaching never304="true" />
  <!-- If you include a <cacheControl> directive, it will be used to
    generate a Cache-Control header (as well as an Expires header
    if the value contains "max-age=")
    By default, no Cache-Control header is generated.
    You can use the <cacheControl> option even if you have set
    never304="true"
  -->
  <!--
  <httpCaching never304="true" >
    <cacheControl>max-age=30, public</cacheControl>
  </httpCaching>
  -->
  <!-- To enable Solr to respond with automatically generated HTTP
    Caching headers, and to response to Cache Validation requests
    correctly, set the value of never304="false"

    This will cause Solr to generate Last-Modified and ETag
    headers based on the properties of the Index.
```

The following options can also be specified to affect the values of these headers...

`lastModFrom` - the default value is `"openTime"` which means the Last-Modified value (and validation against If-Modified-Since requests) will all be relative to when the current Searcher was opened. You can change it to `lastModFrom="dirLastMod"` if you want the value to exactly correspond to when the physical index was last modified.

`etagSeed="..."` is an option you can change to force the ETag header (and validation against If-None-Match requests) to be different even if the index has not changed (ie: when making significant changes to your config file)
(`lastModifiedFrom` and `etagSeed` are both ignored if you use the `never304="true"` option)

```
-->
<!--
  <httpCaching lastModifiedFrom="openTime"
               etagSeed="Solr">
    <cacheControl>max-age=30, public</cacheControl>
  </httpCaching>
-->
</requestDispatcher>

<!-- Request Handlers

  http://wiki.apache.org/solr/SolrRequestHandler

  Incoming queries will be dispatched to a specific handler by name
  based on the path specified in the request.

  If a Request Handler is declared with startup="lazy", then it will
  not be initialized until the first request that uses it.

  -->
  <!-- SearchHandler
    http://wiki.apache.org/solr/SearchHandler

    For processing Search Queries, the primary Request Handler
    provided with Solr is "SearchHandler" It delegates to a sequent
    of SearchComponents (see below) and supports distributed
    queries across multiple shards
  -->

<requestHandler name="/get" class="solr.RealTimeGetHandler">
  <lst name="defaults">
    <str name="omitHeader">true</str>
    <str name="wt">json</str>
    <str name="indent">true</str>
  </lst>
  <arr name="first-components">
    <str>queryDocAuthorization</str>
  </arr>
</requestHandler>

<requestHandler name="/select" class="solr.SearchHandler">
  <!-- default values for query parameters can be specified, these
       will be overridden by parameters in the request
  -->
  <lst name="defaults">
```

```

<str name="echoParams">explicit</str>
<int name="rows">10</int>
<!-- Default search field
  <str name="df">text</str>
-->
<!-- Change from JSON to XML format (the default prior to Solr 7.0)
  <str name="wt">xml</str>
-->
</lst>
<!-- In addition to defaults, "appends" params can be specified
  to identify values which should be appended to the list of
  multi-val params from the query (or the existing "defaults").
-->
<!-- In this example, the param "fq=instock:true" would be appended to
  any query time fq params the user may specify, as a mechanism for
  partitioning the index, independent of any user selected filtering
  that may also be desired (perhaps as a result of faceted searching
  ).

  NOTE: there is absolutely nothing a client can do to prevent these
  "appends" values from being used, so don't use this mechanism
  unless you are sure you always want it.
-->
<!--
  <lst name="appends">
    <str name="fq">instock:true</str>
  </lst>
-->
<!-- "invariants" are a way of letting the Solr maintainer lock down
  the options available to Solr clients. Any params values
  specified here are used regardless of what values may be specified
  in either the query, the "defaults", or the "appends" params.

  In this example, the facet.field and facet.query params would
  be fixed, limiting the facets clients can use. Faceting is
  not turned on by default - but if the client does specify
  facet=true in the request, these are the only facets they
  will be able to see counts for; regardless of what other
  facet.field or facet.query params they may specify.

  NOTE: there is absolutely nothing a client can do to prevent these
  "invariants" values from being used, so don't use this mechanism
  unless you are sure you always want it.
-->
<!--
  <lst name="invariants">
    <str name="facet.field">cat</str>
    <str name="facet.field">manu_exact</str>
    <str name="facet.query">price:[* TO 500]</str>
    <str name="facet.query">price:[500 TO *]</str>
  </lst>
-->
<!-- If the default list of SearchComponents is not desired, that
  list can either be overridden completely, or components can be
  prepended or appended to the default list. (see below)
-->
<!--
  <arr name="components">
    <str>nameOfCustomComponent1</str>
    <str>nameOfCustomComponent2</str>
  </arr>
-->

```

```

    <arr name="first-components">
      <str>queryDocAuthorization</str>
    </arr>
  </requestHandler>

  <!-- A request handler that returns indented JSON by default -->
  <requestHandler name="/query" class="solr.SearchHandler">
    <lst name="defaults">
      <str name="echoParams">explicit</str>
      <str name="wt">json</str>
      <str name="indent">true</str>
    </lst>
    <arr name="first-components">
      <str>queryDocAuthorization</str>
    </arr>
  </requestHandler>

  <initParams path="/update/**,/query,/select,/spell">
    <lst name="defaults">
      <str name="df">_text_</str>
    </lst>
  </initParams>

  <!-- Search Components

  Search components are registered to SolrCore and used by
  instances of SearchHandler (which can access them by name)

  By default, the following components are available:

  <searchComponent name="query"          class="solr.QueryComponent" />
  <searchComponent name="facet"          class="solr.FacetComponent" />
  <searchComponent name="mlt"            class="solr.MoreLikeThisComponent" />
  <searchComponent name="highlight"      class="solr.HighlightComponent" />
  <searchComponent name="stats"           class="solr.StatsComponent" />
  <searchComponent name="debug"           class="solr.DebugComponent" />

  Default configuration in a requestHandler would look like:

  <arr name="components">
    <str>query</str>
    <str>facet</str>
    <str>mlt</str>
    <str>highlight</str>
    <str>stats</str>
    <str>debug</str>
  </arr>

  If you register a searchComponent to one of the standard names,
  that will be used instead of the default.

  To insert components before or after the 'standard' components, use:

  <arr name="first-components">
    <str>myFirstComponentName</str>
  </arr>
  <arr name="last-components">
    <str>myLastComponentName</str>
  </arr>

  NOTE: The component registered with the name "debug" will
  always be executed after the "last-components"

```

```

-->
<searchComponent name="queryDocAuthorization" class="org.apache.ranger.a
uthorization.solr.authorizer.RangerSolrAuthorizer">
  <str name="enabled">true</str>
  <!-- The field which contains the role or list of roles which are allow
ed to query a particular document -->
  <str name="rangerAuthField">ranger_auth</str>
  <!-- If the rangerAuthField contains this value, all roles will be allo
wed to query that particular document -->
  <str name="allRolesToken">*</str>
</searchComponent>

<!-- Spell Check

    The spell check component can return a list of alternative spelling
    suggestions.

    http://wiki.apache.org/solr/SpellCheckComponent
-->
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
  <str name="queryAnalyzerFieldType">text_general</str>

  <!-- Multiple "Spell Checkers" can be declared and used by this
    component
  -->

  <!-- a spellchecker built from a field of the main index -->
  <lst name="spellchecker">
    <str name="name">default</str>
    <str name="field">_text</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <!-- the spellcheck distance measure used, the default is the internal
levenshtein -->
    <str name="distanceMeasure">internal</str>
    <!-- minimum accuracy needed to be considered a valid spellcheck su
ggestion -->
    <float name="accuracy">0.5</float>
    <!-- the maximum #edits we consider when enumerating terms: can be 1
or 2 -->
    <int name="maxEdits">2</int>
    <!-- the minimum shared prefix when enumerating terms -->
    <int name="minPrefix">1</int>
    <!-- maximum number of inspections per result. -->
    <int name="maxInspections">5</int>
    <!-- minimum length of a query term to be considered for correction --
>
    <int name="minQueryLength">4</int>
    <!-- maximum threshold of documents a query term can appear to be co
nsidered for correction -->
    <float name="maxQueryFrequency">0.01</float>
    <!-- uncomment this to require suggestions to occur in 1% of the do
cuments
    <float name="thresholdTokenFrequency">.01</float>
  -->
</lst>

  <!-- a spellchecker that can break or combine words. See "/spell" handl
er below for usage -->
  <!--
  <lst name="spellchecker">
    <str name="name">wordbreak</str>
    <str name="classname">solr.WordBreakSolrSpellChecker</str>
    <str name="field">name</str>

```

```

    <str name="combineWords">true</str>
    <str name="breakWords">true</str>
    <int name="maxChanges">10</int>
  </lst>
-->
</searchComponent>

<!-- A request handler for demonstrating the spellcheck component.

NOTE: This is purely as an example. The whole purpose of the
SpellCheckComponent is to hook it into the request handler that
handles your normal user queries so that a separate request is
not needed to get suggestions.
IN OTHER WORDS, THERE IS REALLY GOOD CHANCE THE SETUP BELOW IS
NOT WHAT YOU WANT FOR YOUR PRODUCTION SYSTEM!
See http://wiki.apache.org/solr/SpellCheckComponent for details
on the request parameters.
-->
<requestHandler name="/spell" class="solr.SearchHandler" startup="lazy">
  <lst name="defaults">
    <!-- Solr will use suggestions from both the 'default' spellchecker
    and from the 'wordbreak' spellchecker and combine them.
    collations (re-written queries) can include a combination of
    corrections from both spellcheckers -->
    <str name="spellcheck.dictionary">default</str>
    <str name="spellcheck">on</str>
    <str name="spellcheck.extendedResults">true</str>
    <str name="spellcheck.count">10</str>
    <str name="spellcheck.alternativeTermCount">5</str>
    <str name="spellcheck.maxResultsForSuggest">5</str>
    <str name="spellcheck.collate">true</str>
    <str name="spellcheck.collateExtendedResults">true</str>
    <str name="spellcheck.maxCollationTries">10</str>
    <str name="spellcheck.maxCollations">5</str>
  </lst>
  <arr name="last-components">
    <str>spellcheck</str>
  </arr>
</requestHandler>
<!-- Terms Component
http://wiki.apache.org/solr/TermsComponent

A component to return terms and document frequency of those
terms
-->
<searchComponent name="terms" class="solr.TermsComponent"/>

<!-- A request handler for demonstrating the terms component -->
<requestHandler name="/terms" class="solr.SearchHandler" startup="lazy">
  <lst name="defaults">
    <bool name="terms">true</bool>
    <bool name="distrib">false</bool>
  </lst>
  <arr name="components">
    <str>terms</str>
  </arr>
</requestHandler>

<!-- Highlighting Component
http://wiki.apache.org/solr/HighlightingParameters
-->
<searchComponent class="solr.HighlightComponent" name="highlight">
  <highlighting>
    <!-- Configure the standard fragmenter -->

```

```

<!-- This could most likely be commented out in the "default" case -->
<fragmenter name="gap"
            default="true"
            class="solr.highlight.GapFragmenter">
  <lst name="defaults">
    <int name="hl.fragsize">100</int>
  </lst>
</fragmenter>
<!-- A regular-expression-based fragmenter
      (for sentence extraction)
-->
<fragmenter name="regex"
            class="solr.highlight.RegexFragmenter">
  <lst name="defaults">
    <!-- slightly smaller fragsizes work better because of slop -->
    <int name="hl.fragsize">70</int>
    <!-- allow 50% slop on fragment sizes -->
    <float name="hl.regex.slop">0.5</float>
    <!-- a basic sentence pattern -->
    <str name="hl.regex.pattern">[-\w ,/\n\&quot;&apos;]{20,200}</str>
  </lst>
</fragmenter>

<!-- Configure the standard formatter -->
<formatter name="html"
           default="true"
           class="solr.highlight.HtmlFormatter">
  <lst name="defaults">
    <str name="hl.simple.pre"><![CDATA[<em>]]></str>
    <str name="hl.simple.post"><![CDATA[</em>]]></str>
  </lst>
</formatter>

<!-- Configure the standard encoder -->
<encoder name="html"
         class="solr.highlight.HtmlEncoder" />

<!-- Configure the standard fragListBuilder -->
<fragListBuilder name="simple"
                class="solr.highlight.SimpleFragListBuilder"/>

<!-- Configure the single fragListBuilder -->
<fragListBuilder name="single"
                class="solr.highlight.SingleFragListBuilder"/>

<!-- Configure the weighted fragListBuilder -->
<fragListBuilder name="weighted"
                default="true"
                class="solr.highlight.WeightedFragListBuilder"/>

<!-- default tag FragmentsBuilder -->
<fragmentsBuilder name="default"
                 default="true"
                 class="solr.highlight.ScoreOrderFragmentsBuilder">
  <!--
  <lst name="defaults">
    <str name="hl.multiValuedSeparatorChar">/</str>
  </lst>
  -->
</fragmentsBuilder>

<!-- multi-colored tag FragmentsBuilder -->
<fragmentsBuilder name="colored"
                 class="solr.highlight.ScoreOrderFragmentsBuilder">

```

```

        <lst name="defaults">
          <str name="hl.tag.pre"><![CDATA[
            <b style="background:yellow">,<b style="background:lawgreen"
>,
            <b style="background:aquamarine">,<b style="background:magenta">,
            <b style="background:palegreen">,<b style="background:coral">
            ,
            <b style="background:wheat">,<b style="background:khaki">,<b style="background:lime">,<b style="background:deepskyblue"
>]]></str>
          <str name="hl.tag.post"><![CDATA[</b>]]></str>
        </lst>
      </fragmentsBuilder>

      <boundaryScanner name="default"
        default="true"
        class="solr.highlight.SimpleBoundaryScanner">
        <lst name="defaults">
          <str name="hl.bs.maxScan">10</str>
          <str name="hl.bs.chars">., !? &#9;&#10;&#13;</str>
        </lst>
      </boundaryScanner>

      <boundaryScanner name="breakIterator"
        class="solr.highlight.BreakIteratorBoundaryScanner">
        <lst name="defaults">
          <!-- type should be one of CHARACTER, WORD(default), LINE and SENTENCE -->
          <str name="hl.bs.type">WORD</str>
          <!-- language and country are used when constructing Locale object. -->
          <!-- And the Locale object will be used when getting instance of BreakIterator -->
          <str name="hl.bs.language">en</str>
          <str name="hl.bs.country">US</str>
        </lst>
      </boundaryScanner>
    </highlighting>
  </searchComponent>

  <!-- Update Processors
  Chains of Update Processor Factories for dealing with Update
  Requests can be declared, and then used by name in Update
  Request Processors
  http://wiki.apache.org/solr/UpdateRequestProcessor
  -->
  <!-- Add unknown fields to the schema

  Field type guessing update processors that will
  attempt to parse string-typed field values as Booleans, Longs,
  Doubles, or Dates, and then add schema fields with the guessed
  field types. Text content will be indexed as "text_general" as
  well as a copy to a plain string version in *_str.

  These require that the schema is both managed and mutable, by
  declaring schemaFactory as ManagedIndexSchemaFactory, with
  mutable specified as true.

  See http://wiki.apache.org/solr/GuessingFieldTypes
  -->
  <updateProcessor class="solr.UUIDUpdateProcessorFactory" name="uuid"/>

```

```

<updateProcessor class="solr.RemoveBlankFieldUpdateProcessorFactory" name="remove-blank"/>
<updateProcessor class="solr.FieldNameMutatingUpdateProcessorFactory" name="field-name-mutating">
  <str name="pattern">[^\w-\.]</str>
  <str name="replacement">_</str>
</updateProcessor>
<updateProcessor class="solr.ParseBooleanFieldUpdateProcessorFactory" name="parse-boolean"/>
<updateProcessor class="solr.ParseLongFieldUpdateProcessorFactory" name="parse-long"/>
<updateProcessor class="solr.ParseDoubleFieldUpdateProcessorFactory" name="parse-double"/>
<updateProcessor class="solr.ParseDateFieldUpdateProcessorFactory" name="parse-date">
  <arr name="format">
    <str>yyyy-MM-dd['T'[HH:mm[:ss[.SSS]]][z</str>
    <str>yyyy-MM-dd['T'[HH:mm[:ss[,SSS]]][z</str>
    <str>yyyy-MM-dd HH:mm[:ss[.SSS]]][z</str>
    <str>yyyy-MM-dd HH:mm[:ss[,SSS]]][z</str>
    <str>[EEE, ]dd MMM yyyy HH:mm[:ss] z</str>
    <str>EEEE, dd-MMM-yy HH:mm:ss z</str>
    <str>EEE MMM ppd HH:mm:ss [z ]yyyy</str>
  </arr>
</updateProcessor>
<updateProcessor class="solr.AddSchemaFieldsUpdateProcessorFactory" name="add-schema-fields">
  <lst name="typeMapping">
    <str name="valueClass">java.lang.String</str>
    <str name="fieldType">text_general</str>
    <lst name="copyField">
      <str name="dest">*_str</str>
      <int name="maxChars">256</int>
    </lst>
    <!-- Use as default mapping instead of defaultFieldType -->
    <bool name="default">true</bool>
  </lst>
  <lst name="typeMapping">
    <str name="valueClass">java.lang.Boolean</str>
    <str name="fieldType">booleans</str>
  </lst>
  <lst name="typeMapping">
    <str name="valueClass">java.util.Date</str>
    <str name="fieldType">pdates</str>
  </lst>
  <lst name="typeMapping">
    <str name="valueClass">java.lang.Long</str>
    <str name="valueClass">java.lang.Integer</str>
    <str name="fieldType">plongs</str>
  </lst>
  <lst name="typeMapping">
    <str name="valueClass">java.lang.Number</str>
    <str name="fieldType">pdoubles</str>
  </lst>
</updateProcessor>

<!-- The update.autoCreateFields property can be turned to false to disable schemaless mode -->
<updateRequestProcessorChain name="add-unknown-fields-to-the-schema" default="${update.autoCreateFields:true}"
  processor="uuid,remove-blank,field-name-mutating,parse-boolean,parse-long,parse-double,parse-date,add-schema-fields">
  <processor class="solr.LogUpdateProcessorFactory"/>
  <processor class="solr.DistributedUpdateProcessorFactory"/>

```

```

<processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
<!-- Deduplication

    An example dedup update processor that creates the "id" field
    on the fly based on the hash code of some other fields. This
    example has overwriteDups set to false since we are using the
    id field as the signatureField and Solr will maintain
    uniqueness based on that anyway.
-->
<!--
<updateRequestProcessorChain name="dedupe">
  <processor class="solr.processor.SignatureUpdateProcessorFactory">
    <bool name="enabled">true</bool>
    <str name="signatureField">id</str>
    <bool name="overwriteDups">false</bool>
    <str name="fields">name, features, cat</str>
    <str name="signatureClass">solr.processor.Lookup3Signature</str>
  </processor>
  <processor class="solr.LogUpdateProcessorFactory" />
  <processor class="solr.RunUpdateProcessorFactory" />
</updateRequestProcessorChain>
-->

<!-- Response Writers

    http://wiki.apache.org/solr/QueryResponseWriter

    Request responses will be written using the writer specified by
    the 'wt' request parameter matching the name of a registered
    writer.

    The "default" writer is the default and will be used if 'wt' is
    not specified in the request.
-->
<!-- The following response writers are implicitly configured unless
    overridden...
-->
<!--
  <queryResponseWriter name="xml"
                      default="true"
                      class="solr.XMLResponseWriter" />
  <queryResponseWriter name="json" class="solr.JSONResponseWriter"/>
  <queryResponseWriter name="python" class="solr.PythonResponseWriter"/>
  <queryResponseWriter name="ruby" class="solr.RubyResponseWriter"/>
  <queryResponseWriter name="php" class="solr.PHPResponseWriter"/>
  <queryResponseWriter name="phps" class="solr.PHPSerializedResponseWriter"/>
  <queryResponseWriter name="csv" class="solr.CSVResponseWriter"/>
  <queryResponseWriter name="schema.xml" class="solr.SchemaXmlResponseWriter"/>
-->
  <queryResponseWriter name="json" class="solr.JSONResponseWriter">
    <!-- For the purposes of the tutorial, JSON responses are written as
    plain text so that they are easy to read in *any* browser.
    If you expect a MIME type of "application/json" just remove this override.
    -->
    <str name="content-type">text/plain; charset=UTF-8</str>
  </queryResponseWriter>
<!-- Query Parsers

    https://lucene.apache.org/solr/guide/query-syntax-and-parsing.html

```

```

    Multiple QParserPlugins can be registered by name, and then
    used in either the "defType" param for the QueryComponent (used
    by SearchHandler) or in LocalParams
    -->
<!-- example of registering a query parser -->
<!--
    <queryParser name="myparser" class="com.mycompany.MyQParserPlugin"/>
    -->

<!-- Function Parsers
    http://wiki.apache.org/solr/FunctionQuery
    Multiple ValueSourceParsers can be registered by name, and then
    used as function names when using the "func" QParser.
    -->
<!-- example of registering a custom function parser -->
<!--
    <valueSourceParser name="myfunc"
                      class="com.mycompany.MyValueSourceParser" />
    -->

<!-- Document Transformers
    http://wiki.apache.org/solr/DocTransformers
    -->
<!--
    Could be something like:
    <transformer name="db" class="com.mycompany.LoadFromDatabaseTransform
er" >
        <int name="connection">jdbc://...</int>
    </transformer>

    To add a constant value to all docs, use:
    <transformer name="mytrans2" class="org.apache.solr.response.transfor
m.ValueAugmenterFactory" >
        <int name="value">5</int>
    </transformer>
    If you want the user to still be able to change it with _value:somethi
ng_ use this:
    <transformer name="mytrans3" class="org.apache.solr.response.transfor
m.ValueAugmenterFactory" >
        <double name="defaultValue">5</double>
    </transformer>

    If you are using the QueryElevationComponent, you may wish to mark d
ocuments that get boosted. The
    EditorialMarkerFactory will do exactly that:
    <transformer name="qecBooster" class="org.apache.solr.response.trans
form.EditorialMarkerFactory" />
    -->
</config>

```

Configuring LDAP attribute-based document level authorization in Solr

You can enable Ranger-Solr plugin to support LDAP attribute-based document level authorization.

About this task

In order to enable this feature, you need to make some configuration changes on both Solr and Ranger sides.

Procedure

1. Configure Usersync to sync additional attributes for users/groups from AD/LDAP.

- a) In Cloudera Manager, go to Ranger Configuration .
- b) Add the following safety valve properties in the ugsync-site.xml file:
 - For users:
 - `ranger.usersync.ldap.user.otherattributes` - Takes "," separated values of the attribute names of the users from AD/LDAP. Example values can be `userPrincipalName`, `displayName`, `objectGUID`, and so on.
 - `ranger.usersync.ldap.user.otherattributes.<attribute name>datatype` - This configuration provides flexibility to specify the datatype of specific attribute value. Default data type for all the attribute values is "String". For example, value for `objectGUID` returned by AD is `byte[]`. Hence the corresponding configuration must be `ranger.usersync.ldap.user.otherattributes.objectGUIDdatatype = byte[]`.
 - For groups:
 - `ranger.usersync.ldap.group.otherattributes` - Takes "," separated values of the attribute names of the groups from AD/LDAP.
 - `ranger.usersync.ldap.group.otherattributes.<attribute name>datatype` - Same as user attribute datatype.

2. Restart Usersync.

3. Update Solr service definition configuration to download user/group attributes to Solr service.

- a) Get the existing service definition from your cluster by using the curl command.

For example,

```
curl -u admin:admin123 -H "Accept: application/json" -H "Content-Type: application/json" -X GET http://sp-718-1.sp-718.root.hwx.site:6080/service/public/v2/api/servicedef/8
```

- b) Save the output to file and update the contextEnricher section with the following information:

```
"contextEnrichers": [{ "itemId": 1, "name": "UserEnricher", "enricher": "org.apache.ranger.plugin.contextenricher.RangerUserStoreEnricher", "enricherOptions": { "userStoreRetrieverClassName": "org.apache.ranger.plugin.contextenricher.RangerAdminUserStoreRetriever", "userStoreRefresherPollingInterval": "60000" } }],
```

- c) Update the service definition by using the following curl command:

```
curl -u admin:admin123 -H "Accept: application/json" -H "Content-Type: application/json" -X PUT http://sp-718-1.sp-718.root.hwx.site:6080/service/public/v2/api/servicedef/name/solr -d @/tmp/solrServiceDef.json
```

4. Update Solr service repo configuration to allow Solr service user (solr) to download user or group attributes from Ranger admin.

- a) Login to Ranger UI as admin user.
- b) Go to Access Manager.
- c) Click the edit icon for the `cm_solr` plugin.
- d) On the Edit Service page, click the + icon for Add New Configurations.
- e) Add `userstore.download.auth.users` with the value `solr`.
- f) Click Save.