Security 3

Creating the Ranger Plugin for HDF Services

Date of Publish: 2020-12-15

https://docs.cloudera.com/

Contents

Set up NiFi Registry Ranger Plugin

The article describes how to integrate secured NiFi Registry and Ranger environments using Ambari.

Before you begin

- NiFi Registry is installed and SSL is enabled (with certificates manually installed or installed using the NiFi Certificate Authority). Note the keystore and truststore names, locations, aliases, identity (DN) and passwords used when enabling SSL for NiFi Registry.
- Ranger is installed and SSL is enabled. Note the name, location, aliases, identity (DN) and passwords used when creating keystores and truststores for Ranger.
- If you want to use Kerberos, enable it for the HDF cluster before you proceed.

  You need to perform the following steps to set up NiFi Registry Ranger plugin:

Procedure

1. Establish communication between NiFi Registry and Ranger.
2. Enable NiFi Registry Ranger plugin using Ambari.
3. Confirm Ranger configuration.
4. Set up Ranger Policies.

Establish Communication Between NiFi Registry and Ranger

In order for NiFi Registry to communicate with Ranger over SSL, and Ranger to communicate with NiFi Registry over SSL, certificates must be imported from the Ranger host to NiFi Registry and from NiFi Registry to the Ranger host. In this article, same keystore and truststore are used to secure Ranger in order to communicate with NiFi Registry. You can also generate additional keystores and truststores that are dedicated solely to NiFi Registry communication.

About this task

Procedure

1. Create certificate files from the Ranger keystore by using the following command:

   {java.home}/bin/keytool -export -keystore {ranger.keystore.file} -alias {ranger.keystore.alias} -file {ranger.cert.filename}

   For example, use /usr/jdk64/jdk1.8.0_77/bin/keytool -export -keystore /etc/security/certs/ranger/ranger-admin-keystore.jks -alias rangeradmin -file /etc/security/certs/ranger/ranger-admin-trust.cer
2. Import the Ranger server certificate into the NiFi Registry truststore, so that NiFi Registry can access Ranger through HTTPS, by using the following command:

   {java.home}/bin/keytool -import -file {ranger.cert.filename} -alias {ranger.keystore.alias} -keystore {nifi-registry.truststore} -storepass {nifi-registry.truststore.password}
3. Create certificate files for import into the Ranger truststore.
   - If NiFi Certificate Authority is in use, a certificate from the CA can be generated and imported into the Ranger truststore by performing the following steps:

     a. Export NiFi CA certificate, so that other component can import and trust it, by using the following command: {java.home}/bin/keytool -export -keystore {nifi-ca.keystore.file} -alias {nifi-ca.keystore-alias} -file {nifi-ca-cert.filename}

This is required for Ranger. Exporting this certificate once is enough if NiFi and NiFi Registry use the same NiFi CA.

b. Import the NiFi CA certificate into the truststore of Ranger by using the following command: {java.home}/bin/keytool -import -file {nifi-ca.cert.filename} -alias {nifi-ca.keystore.alias} -keystore {ranger.truststore.file} -storepass {ranger.truststore.password}

You need to import each certificate that you generate. Remember that any duplicate alias might need to be changed using changealias command before importing new ones. This is required for Ranger. Exporting this certificate once is enough if NiFi and NiFi Registry use the same NiFi CA.

- If you use an external CA or self-signed certificates and provide manual keystores and truststores for NiFi Registry, then perform the following steps:

a. Create a certificate file from the NiFi Registry keystore by using the following command: {java.home}/bin/keytool -export -keystore {nifi-registry.keystore.file} -alias {nifi-registry.keystore-alias} -file {nifi-registry.cert.filename}

b. Import the certificate file into the Ranger truststore by using the following command: {java.home}/bin/keytool -import -file {nifi-registry.cert.filename} -alias {nifi-registry.keystore.alias} -keystore {ranger.truststore} -storepass {ranger.truststore.password}

Note: Truststore used by Ranger might be default truststore cacerts file located in {java_home}/jre/lib/security/cacerts.

Enable NiFi Registry Ranger Plugin

Enabling the NiFi Registry Ranger plugin leads to Ambari creating a service repository entry in Ranger which stores information for Ranger to communicate with NiFi Registry and the authorized identity of the NiFi Registry that communicates with Ranger.

About this task

To enable NiFi Registry Ranger plugin, perform the following steps from the Ambari UI:

Procedure

1. Go to **Ranger** > **CONFIGS** > **RANGER PLUGIN**.
2. Switch the **NiFi Registry Ranger Plugin** toggle to **ON**, and click **Save**.
3. Optional. Go to the **Ranger Audit** tab and, if not already enabled, switch the **Audit to Solr** toggle to **ON**.

   It produces options to enter connection properties for a Solr instance.
4. Optional. To use with Ambari Infra (Internal SolrCloud), switch the **SolrCloud** toggle to **ON**, and click **Save**.

   Ambari will pre-populate the zookeeper connection string values and credentials. If an External Solr is used, you need to provide the connection values.
5. Go to **NiFi Registry** > **CONFIGS** > **Advanced ranger-nifi-registry-plugin-properties** from the Ambari UI.

   The Advanced ranger-nifi-registry-plugin-properties section stores all the information needed to support Ranger communication with NiFi Registry.
6. Configure the following properties:

Properties	Description
Ranger repository config password	Confirm that the value is populated. The value refers to the admin password for Ranger and is set by Ambari by default.
Ranger repository config user	Confirm that the value is populated. The value refers to the admin username for Ranger and is set by Ambari by default.

Properties	Description
Authentication	Enter SSL if not populated already by Ambari. It informs Ranger that NiFi Registry is running with SSL.
Keystore for Ranger Service Accessing NiFi Registry	Enter the keystore filename with location path that Ranger uses for SSL communications with NiFi Registry. This corresponds to the keystore used to generate a certificate that you created during establishing communication between Ranger and NiFi Registry.
Keystore password	Enter the password for the keystore.
Keystore Type	Enter the keystore type. For example, enter JKS.
Truststore for Ranger Service Accessing NiFi Registry	Enter the filename with location path of the truststore for the Ranger service.
Truststore password	Enter the password for the truststore.
Truststore Type	Enter the truststore type. For example, enter JKS.
Owner for Certificate	Enter the identity (Distinguished Name or DN) of the certificate used by Ranger.
Policy user for NiFi Registry	Confirm that the value is populated as nifiregistry.
Enable Ranger for NiFi Registry	Confirm that the checkbox is enabled.

7. Go to **Advanced ranger-nifi-registry-policymgr-ssl**.
   This section stores the information NiFi Registry uses to communicate with the secured Ranger service.
8. Configure the following properties:

Properties	Description
owner.for.certificate	Enter the identity (Distinguished Name or DN) of the NiFi Registry to communicate with Ranger. This value is not required if Kerberos is enabled on HDF.
xasecure.policymgr.clientssl.keystore	Enter the keystore location and filename that NiFi Registry uses to communicate with Ranger. This keystore reference must be the same file used to create and import a certificate into Ranger.
xasecure.policymgr.clientssl.keystore.credential.file	This value is populated by default and is used by the plugin to generate a file to store credential information. No change to this value is required.
xasecure.policymgr.clientssl.truststore	Enter the truststore location and filename that NiFi Registry uses to communicate with Ranger.
xasecure.policymgr.clientssl.truststore.credential.file	This value is populated by default and is used by the plugin to generate a file to store credential information. No change to this value is required.
xasecure.policymgr.clientssl.truststore.password	Enter the password for the provided truststore file.

9. Go to **Advanced ranger-nifi-registry-security** and review the following properties:

Properties	Description
ranger.plugin.nifi-registry.policy.rest.ssl.config.file	Check whether it is set to ranger-policymgr-ssl.xml.
ranger.plugin.nifi-registry.policy.rest.url	Check whether it refers to the Ambari variable for Ranger service {{policy_mgr_url}}.

10. Go to **Advanced ranger-nifi-registry-audit** and review the following properties:

Properties	Description
Audit to SOLR	Check whether it is enabled.
xasecure.audit.destination.solr.urls	Check the status of the property. When xasecure.audit.destination.solr.zookeepers is populated, it remains empty.
xasecure.audit.destination.solr.zookeepers	Check whether it is enabled and matches the connection string.
xasecure.audit.is.enabled	Check whether it is set to true.

11. Save all NiFi Registry configuration changes.
12. Restart all required services and ensure that Ambari indicates that the services have been restarted successfully.

Confirm Ranger Configuration

This article describes how to confirm Ranger configuration.

Procedure

1. In Ranger Admin UI, go to **Access Manager** > **Resource Based Policies**.
2. Check that an entry for NiFi Registry exists in the NiFi Registry Service Manager.

   The entry name is dynamically created based on the Ambari cluster name.
3. Click the edit button next to the service repository entry and confirm that the properties from the ranger-nifi-registry-plugin-properties are accurately populated.
4. Check whether the **NiFi Registry URL** is populated.
5. Confirm that the commonNameForCertificate value is the CN value from the Owner for Certificate property from ranger-nifi-registry-plugin-properties.
6. Go to **Audit** > **Plugins**, and check the syncing between NiFi Registry and Ranger policies.
7. If you are not using user sync in Ranger, you can manually create new users in Ranger which correspond to the authentication method used to secure NiFi Registry. For example when using Kerberos authentication in NiFi Registry, ensure that the users created match with the Kerberos principal.
   a) In the Ranger Admin UI, go to **Settings** and select **User/Groups**.
   b) Click the **Add New User** button.
   c) Configure the following properties in the **User Detail** section:
      - User Name. Enter user name. User name is the identity for the appropriate NiFi Registry authentication method. For example, enter Client DN, LDAP DN, or Kerberos principal.
      - New Password. Enter password. This is required by Ranger.
      - Password Confirm. Confirm password.
      - First Name. Enter first name. This is required by Ranger.
      - Last Name. Optional. Enter last name.
      - Email Address. Optional. Enter email address.

- Select Role. Select **User**. Groups are not used by the plugin.

d) Click **Save** to save the new user and repeat for any other users who need access to NiFi Registry.

8. In the Ranger Admin UI, go to **Access Manager** > **Resource Based Policies**, and click the edit button next to the NiFi Registry service repository entry link.
9. Click the **Test Connection** button.
   The **Connected Successfully** message appears.

Set up Ranger Policies

This article describes how to set up Ranger policies.

Procedure

1. In the Ranger Admin UI, go to **Access Manager** > **Resource Based Policies** to configure other user policies.
2. Select the NiFi Registry service repository entry link, and click the edit button next to the **all - nifi-registry-resource** entry.
3. In the **Allow Conditions** section select the users who can access this policy.

   You need to add both **Read** and **Write** permissions.
4. Click **Save** to save the policy with the new settings and confirm that the configured user can access NiFi Registry with given rights by logging into NiFi Registry.
5. Confirm that login access are audited in Ranger using the **Audit** screen and navigating to the **Access** tab.
   Now Ranger can be used to administer policy for NiFi Registry.

Deployment Scenarios for NiFi Registry Ranger Plugin

To integrate secured NiFi Registry and Ranger environments using Ambari, you might need to update or change settings based on you deployment scenarios.

Scenario	Notes
Installing an HDF Cluster	• To use the NiFi Registry Ranger plugin, you must update the value of the **Ranger audit service users** property as {default_ranger_audit_users}, nifiregistry in the Advanced infra-solr-security-json section for Infra Solr service. It enables NiFi Registry to audit logs to Solr. • To configure Ranger policies for NiFi Registry, ensure the following: - NiFi node users must have all permissions for /proxy. - NiFi node users must have read and write permissions for the target bucket or allow all buckets to use /buckets. For example, a bucket resource identifier looks like: /buckets/a4651561-e36f-4dca-8216-330d97043195.
Upgrading an HDF Cluster	• Before executing upgrade mpack command, you must stop the Ambari Server. • You must update the **Template for authorizers.xml** property in the Advanced nifi-registry-authorizers-env section. If you do not update the template and enable the NiFi

Scenario	Notes
	Registry Ranger plugin, NiFi Registry fails to start with the following error message: org.apache.nifi.registry.security.authorization.AuthorizerFactoryExc The specified authorizer 'ranger-authorizer' could not be found. • If Kerberos is enabled in your cluster, then you need to generate additional key tabs for NiFi Registry service. For instructions on generating key tabs in Ambari, see https://docs.cloudera.com/HDPDocuments/Ambari-2.6.2.2/bk_ambari-operations/content/how_to_regenerate_keytabs.html.
Installing HDF Services on an Existing HDP Cluster	• The **NiFi Registry Ranger Plugin** toggle does not appear, by default, in the **Ranger** > **CONFIGS** > **RANGER PLUGIN** tab. You need to enable it in the Advanced ranger-nifi-registry-plugin-properties section from **NiFi Registry** > **CONFIGS** tab. Go to the specified location and select the **Enable Ranger for NiFi Registry** checkbox. • Set the value of the **Authentication** property as SSL in the Advanced ranger-nifi-registry-plugin-properties section for Ranger to access NiFi Registry. Because HDP Ranger service adviser is not updated when you install HDF services on an existing HDP cluster.

Authorizers.xml Template for NiFi Registry Ranger Plugin

The authorizers.xml file is used to define the userGroupProviders, accessPolicyProviders, and authorizers to use when running securely.

When you upgrade an HDF cluster, you need to update the **Template for authorizers.xml** property at **NiFi Registry** > **CONFIGS** > **Advanced nifi-registry-authorizers-env**.

```
<!--
  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.
-->
<!--
    This file lists the userGroupProviders, accessPolicyProviders, and
 authorizers to use when running securely. In order
    to use a specific authorizer it must be configured here and its
 identifier must be specified in the nifi.properties file.
    If the authorizer is a managedAuthorizer, it may need to be configured
 with an accessPolicyProvider and an userGroupProvider.
```

```
    This file allows for configuration of them, but they must be configured
 in order:

    ...
    all userGroupProviders
    all accessPolicyProviders
    all Authorizers
    ...
-->
<authorizers>

    <!--
        The FileUserGroupProvider will provide support for managing users
 and groups which is backed by a file
        on the local file system.

        - Users File - The file where the FileUserGroupProvider will store
 users and groups.

        - Initial User Identity [unique key] - The identity of a users and
 systems to seed the Users File. The name of
            each property must be unique, for example: "Initial User
 Identity A", "Initial User Identity B",
            "Initial User Identity C" or "Initial User Identity 1", "Initial
 User Identity 2", "Initial User Identity 3"

            NOTE: Any identity mapping rules specified in nifi.properties
 will also be applied to the user identities,
            so the values should be the unmapped identities (i.e. full DN
 from a certificate).
    -->
    <userGroupProvider>
        <identifier>file-user-group-provider</identifier>

 <class>org.apache.nifi.registry.security.authorization.file.FileUserGroupProvider</
class>
        <property name="Users File">{{nifi_registry_internal_config_dir}}/
users.xml</property>
        <property name="Initial User Identity
 0">{{nifi_registry_initial_admin_id}}</property>
        {{nifi_registry_ssl_config_content | replace("NiFi","Initial
 User")}}
    </userGroupProvider>

    <!--
        The LdapUserGroupProvider will retrieve users and groups from an
 LDAP server. The users and groups
        are not configurable.

        'Authentication Strategy' - How the connection to the LDAP server is
 authenticated. Possible
            values are ANONYMOUS, SIMPLE, LDAPS, or START_TLS.

        'Manager DN' - The DN of the manager that is used to bind to the
 LDAP server to search for users.
        'Manager Password' - The password of the manager that is used to
 bind to the LDAP server to
            search for users.

        'TLS - Keystore' - Path to the Keystore that is used when connecting
 to LDAP using LDAPS or START_TLS.
        'TLS - Keystore Password' - Password for the Keystore that is used
 when connecting to LDAP
            using LDAPS or START_TLS.
```

```
       'TLS - Keystore Type' - Type of the Keystore that is used when
connecting to LDAP using
           LDAPS or START_TLS (i.e. JKS or PKCS12).
       'TLS - Truststore' - Path to the Truststore that is used when
connecting to LDAP using LDAPS or START_TLS.
       'TLS - Truststore Password' - Password for the Truststore that is
used when connecting to
           LDAP using LDAPS or START_TLS.
       'TLS - Truststore Type' - Type of the Truststore that is used when
connecting to LDAP using
           LDAPS or START_TLS (i.e. JKS or PKCS12).
       'TLS - Client Auth' - Client authentication policy when connecting
to LDAP using LDAPS or START_TLS.
           Possible values are REQUIRED, WANT, NONE.
       'TLS - Protocol' - Protocol to use when connecting to LDAP using
LDAPS or START_TLS. (i.e. TLS,
           TLSv1.1, TLSv1.2, etc).
       'TLS - Shutdown Gracefully' - Specifies whether the TLS should be
shut down gracefully
           before the target context is closed. Defaults to false.

       'Referral Strategy' - Strategy for handling referrals. Possible
values are FOLLOW, IGNORE, THROW.
       'Connect Timeout' - Duration of connect timeout. (i.e. 10 secs).
       'Read Timeout' - Duration of read timeout. (i.e. 10 secs).

       'Url' - Space-separated list of URLs of the LDAP servers (i.e.
ldap://<hostname>:<port>).
       'Page Size' - Sets the page size when retrieving users and groups.
If not specified, no paging is performed.
       'Sync Interval' - Duration of time between syncing users and groups.
(i.e. 30 mins).

       'User Search Base' - Base DN for searching for users (i.e.
ou=users,o=nifi). Required to search users.
       'User Object Class' - Object class for identifying users (i.e.
person). Required if searching users.
       'User Search Scope' - Search scope for searching users (ONE_LEVEL,
OBJECT, or SUBTREE). Required if searching users.
       'User Search Filter' - Filter for searching for users against the
'User Search Base' (i.e. (memberof=cn=team1,ou=groups,o=nifi) ). Optional.
       'User Identity Attribute' - Attribute to use to extract user
identity (i.e. cn). Optional. If not set, the entire DN is used.
       'User Group Name Attribute' - Attribute to use to define group
membership (i.e. memberof). Optional. If not set
           group membership will not be calculated through the users. Will
rely on group membership being defined
           through 'Group Member Attribute' if set.
       'User Group Name Attribute - Referenced Group Attribute' - If blank,
the value of the attribute defined in 'User Group Name Attribute'
           is expected to be the full dn of the group. If not blank, this
property will define the attribute of the group ldap entry that
           the value of the attribute defined in 'User Group Name
Attribute' is referencing (i.e. name). Use of this property requires that
           'Group Search Base' is also configured.

       'Group Search Base' - Base DN for searching for groups (i.e.
ou=groups,o=nifi). Required to search groups.
       'Group Object Class' - Object class for identifying groups (i.e.
groupOfNames). Required if searching groups.
       'Group Search Scope' - Search scope for searching groups (ONE_LEVEL,
OBJECT, or SUBTREE). Required if searching groups.
       'Group Search Filter' - Filter for searching for groups against the
'Group Search Base'. Optional.
```

```
        'Group Name Attribute' - Attribute to use to extract group name
 (i.e. cn). Optional. If not set, the entire DN is used.
        'Group Member Attribute' - Attribute to use to define group
 membership (i.e. member). Optional. If not set
            group membership will not be calculated through the groups. Will
 rely on group member being defined
            through 'User Group Name Attribute' if set.
        'Group Member Attribute - Referenced User Attribute' - If blank, the
 value of the attribute defined in 'Group Member Attribute'
            is expected to be the full dn of the user. If not blank, this
 property will define the attribute of the user ldap entry that
            the value of the attribute defined in 'Group Member Attribute'
 is referencing (i.e. uid). Use of this property requires that
            'User Search Base' is also configured. (i.e. member: cn=User
 1,ou=users,o=nifi-registry vs. memberUid: user1)

        NOTE: Any identity mapping rules specified in nifi.properties will
 also be applied to the user identities.
            Group names are not mapped.
    -->
    <!-- To enable the ldap-user-group-provider remove 2 lines. This is 1 of
 2.
    <userGroupProvider>
        <identifier>ldap-user-group-provider</identifier>

 <class>org.apache.nifi.registry.security.ldap.tenants.LdapUserGroupProvider</
class>
        <property name="Authentication Strategy">START_TLS</property>

        <property name="Manager DN"></property>
        <property name="Manager Password"></property>

        <property name="TLS - Keystore"></property>
        <property name="TLS - Keystore Password"></property>
        <property name="TLS - Keystore Type"></property>
        <property name="TLS - Truststore"></property>
        <property name="TLS - Truststore Password"></property>
        <property name="TLS - Truststore Type"></property>
        <property name="TLS - Client Auth"></property>
        <property name="TLS - Protocol"></property>
        <property name="TLS - Shutdown Gracefully"></property>

        <property name="Referral Strategy">FOLLOW</property>
        <property name="Connect Timeout">10 secs</property>
        <property name="Read Timeout">10 secs</property>

        <property name="Url"></property>
        <property name="Page Size"></property>
        <property name="Sync Interval">30 mins</property>

        <property name="User Search Base"></property>
        <property name="User Object Class">person</property>
        <property name="User Search Scope">ONE_LEVEL</property>
        <property name="User Search Filter"></property>
        <property name="User Identity Attribute"></property>
        <property name="User Group Name Attribute"></property>
        <property name="User Group Name Attribute - Referenced Group
 Attribute"></property>

        <property name="Group Search Base"></property>
        <property name="Group Object Class">group</property>
        <property name="Group Search Scope">ONE_LEVEL</property>
        <property name="Group Search Filter"></property>
        <property name="Group Name Attribute"></property>
```

```
        <property name="Group Member Attribute"></property>
        <property name="Group Member Attribute - Referenced User
 Attribute"></property>
    </userGroupProvider>
    To enable the ldap-user-group-provider remove 2 lines. This is 2 of 2.
 -->

    <!--
        The CompositeUserGroupProvider will provide support for retrieving
 users and groups from multiple sources.

        - User Group Provider [unique key] - The identifier of user group
 providers to load from. The name of
            each property must be unique, for example: "User Group Provider
 A", "User Group Provider B",
            "User Group Provider C" or "User Group Provider 1", "User Group
 Provider 2", "User Group Provider 3"

            NOTE: Any identity mapping rules specified in nifi.properties
 are not applied in this implementation. This behavior
            would need to be applied by the base implementation.
    -->
    <!-- To enable the composite-user-group-provider remove 2 lines. This is
 1 of 2.
    <userGroupProvider>
        <identifier>composite-user-group-provider</identifier>

 <class>org.apache.nifi.registry.security.authorization.CompositeUserGroupProvider</
class>
        <property name="User Group Provider 1"></property>
    </userGroupProvider>
    To enable the composite-user-group-provider remove 2 lines. This is 2 of
 2. -->

    <!--
        The CompositeConfigurableUserGroupProvider will provide support for
 retrieving users and groups from multiple sources.
        Additionally, a single configurable user group provider is required.
 Users from the configurable user group provider
        are configurable, however users loaded from one of the User Group
 Provider [unique key] will not be.

        - Configurable User Group Provider - A configurable user group
 provider.

        - User Group Provider [unique key] - The identifier of user group
 providers to load from. The name of
            each property must be unique, for example: "User Group Provider
 A", "User Group Provider B",
            "User Group Provider C" or "User Group Provider 1", "User Group
 Provider 2", "User Group Provider 3"

            NOTE: Any identity mapping rules specified in nifi.properties
 are not applied in this implementation. This behavior
            would need to be applied by the base implementation.
    -->
    <!-- To enable the composite-configurable-user-group-provider remove 2
 lines. This is 1 of 2.
    <userGroupProvider>
        <identifier>composite-configurable-user-group-provider</identifier>

 <class>org.apache.nifi.registry.security.authorization.CompositeConfigurableUserGroupPr
class>
```

```
        <property name="Configurable User Group Provider">file-user-group-
provider</property>
        <property name="User Group Provider 1"></property>
    </userGroupProvider>
    To enable the composite-configurable-user-group-provider remove 2 lines.
 This is 2 of 2. -->

    {% if not (has_ranger_admin and enable_ranger_nifi_registry) %}
    <!--
        The FileAccessPolicyProvider will provide support for managing
 access policies which is backed by a file
        on the local file system.

        - User Group Provider - The identifier for an User Group Provider
 defined above that will be used to access
            users and groups for use in the managed access policies.

        - Authorizations File - The file where the FileAccessPolicyProvider
 will store policies.

        - Initial Admin Identity - The identity of an initial admin user
 that will be granted access to the UI and
            given the ability to create additional users, groups, and
 policies. The value of this property could be
            a DN when using certificates or LDAP. This property will only be
 used when there
            are no other policies defined.

            NOTE: Any identity mapping rules specified in nifi.properties
 will also be applied to the initial admin identity,
            so the value should be the unmapped identity. This identity must
 be found in the configured User Group Provider.

        - Node Identity [unique key] - The identity of a NiFi cluster node.
 When clustered, a property for each node
            should be defined, so that every node knows about every other
 node. If not clustered these properties can be ignored.
            The name of each property must be unique, for example for a
 three node cluster:
            "Node Identity A", "Node Identity B", "Node Identity C" or "Node
 Identity 1", "Node Identity 2", "Node Identity 3"

            NOTE: Any identity mapping rules specified in nifi.properties
 will also be applied to the node identities,
            so the values should be the unmapped identities (i.e. full DN
 from a certificate). This identity must be found
            in the configured User Group Provider.
    -->
    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>

 <class>org.apache.nifi.registry.security.authorization.file.FileAccessPolicyProvider</
class>
        <property name="User Group Provider">file-user-group-provider</
property>
        <property name="Authorizations
 File">{{nifi_registry_internal_config_dir}}/authorizations.xml</property>
        <property name="Initial Admin
 Identity">{{nifi_registry_initial_admin_id}}</property>
        {{nifi_registry_ssl_config_content}}
    </accessPolicyProvider>

    <!--
```

```
        The StandardManagedAuthorizer. This authorizer implementation must
 be configured with the
        Access Policy Provider which it will use to access and manage users,
 groups, and policies.
        These users, groups, and policies will be used to make all access
 decisions during authorization
        requests.

        - Access Policy Provider - The identifier for an Access Policy
 Provider defined above.
    -->
    <authorizer>
        <identifier>managed-authorizer</identifier>

 <class>org.apache.nifi.registry.security.authorization.StandardManagedAuthorizer</
class>
        <property name="Access Policy Provider">file-access-policy-
provider</property>
    </authorizer>

    {% else %}
    <authorizer>
        <identifier>ranger-authorizer</identifier>
        <class>org.apache.nifi.registry.ranger.RangerAuthorizer</class>
        <property name="Ranger Service Type">nifi-registry</property>

        <property name="User Group Provider">file-user-group-provider</
property>

        <!-- Specify Ranger service name to use -->
        <property name="Ranger Application Id">{{repo_name}}</property>

        <!--
            Specify configuration file paths for Ranger plugin.
            See the XML files bundled with this extension for further
 details.
         -->
        <property name="Ranger Security Config
 Path">{{nifi_registry_config_dir}}/ranger-nifi-registry-security.xml</
property>
        <property name="Ranger Audit Config
 Path">{{nifi_registry_config_dir}}/ranger-nifi-registry-audit.xml</
property>

        <!--
            Specify user identity that is used by Ranger to access NiFi
 Registry.
            This property is used by NiFi Registry for Ranger to get
 available NiFi Registry policy resource identifiers.
            The configured user can access NiFi Registry /policies/resources
 REST endpoint regardless of configured access policies.
            Ranger uses available policies for user input suggestion at
 Ranger policy editor UI.
        -->
        <property name="Ranger Admin Identity">{{ranger_admin_identity}}</
property>

        <!--
            Specify if target Ranger is Kerberized.
            If set to true, NiFi Registry will use the principal and keytab
 defined at nifi-registry.properties:
            - nifi.registry.kerberos.service.principal
            - nifi.registry.kerberos.service.keytab.location
```

```
            The specified credential is used to access Ranger API, and to
 write audit logs into HDFS (if enabled).

            At Ranger side, the configured user needs to be added to
 'policy.download.auth.users' property, see Ranger configuration section
 below.

            Also, ranger-nifi-registry-security.xml needs additional
 "xasecure.add-hadoop-authorization = true" configuration.
        -->
        {% if security_enabled %}
        <property name="Ranger Kerberos Enabled">true</property>
        {% else %}
        <property name="Ranger Kerberos Enabled">false</property>
        {% endif %}

    </authorizer>
    {% endif %}

</authorizers>
```

Troubleshooting

If there are problems in communication between NiFi Registry and Ranger, review the xa_secure.log and nifi-registry-app.log to determine the source of the issue. This might be due to certificates not being imported into the truststore of Ranger. Also, if Kerberos is not enabled, the commonNameForCertificate property value might be inaccurate.

The problem could also be due to certificates not being imported into NiFi Registry or the Ranger certificate not appropriately being identified. In addition to the previously mentioned logs, review the nifi-registry-user.log.