

## Security

Date published: 2019-06-26

Date modified: 2025-02-27

# CLOUDERA

# Legal Notice

© Cloudera Inc. 2025. 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

<b>Cluster-level security recommendations.....</b>	<b>5</b>
<b>Identity and policies in Apache NiFi.....</b>	<b>5</b>
<b>TLS/SSL configuration.....</b>	<b>6</b>
Enabling Auto-TLS.....	6
Configuring TLS/SSL manually.....	6
TLS/SSL certificate requirements and recommendations.....	6
Configuring TLS/SSL encryption manually for NiFi and NiFi Registry.....	7
NiFi TLS/SSL properties.....	9
NiFi Registry TLS/SSL properties.....	10
<b>Authentication.....</b>	<b>11</b>
Kerberos authentication.....	11
Customizing Kerberos principal.....	11
LDAP authentication.....	12
SAML authentication.....	14
OpenID Connect authentication.....	15
Identity mapping properties.....	16
<b>Hardening ZooKeeper Znodes for NiFi security.....</b>	<b>17</b>
Preparations.....	17
Reconfiguring NiFi.....	19
Znode hardening.....	19
<b>Authorization.....</b>	<b>21</b>
User group providers.....	21
LDAP integration.....	21
Pairing LDAP with a Composite Group Provider.....	25
Access policies providers.....	25
Ranger authorization.....	25
File-based authorization.....	44
Migrating file-based authorization to Ranger.....	45
Environment variables.....	45
Kerberos credentials.....	45
Local file system access.....	46
<b>Network.....</b>	<b>46</b>
Default ports for NiFi and NiFi Registry.....	46
<b>FIPS 140-2 compliance.....</b>	<b>47</b>
Encrypting NiFi sensitive properties with FIPS 140-2 approved algorithm.....	48

Deploying Cloudera Flow Management on FIPS-enabled clusters.....	48
<b>Integrations.....</b>	<b>50</b>
Integrating NiFi and Atlas.....	50
Manually integrating with Atlas when Auto-TLS is not enabled.....	50
Manually integrating with Atlas when Auto-TLS is enabled.....	51
Integrating NiFi and NiFi Registry with Knox.....	51
<b>Customizing properties in Cloudera Manager.....</b>	<b>52</b>

## Cluster-level security recommendations

Flow management users are authenticated automatically when they log into Cloudera. Other aspects of security such as enabling Auto-TLS, Kerberos, and managing access policies depend on the way the SDX and compute clusters are created.

Cloudera recommends the following security options:

- Enable Auto-TLS.
- Enable Kerberos.
- Use Apache Atlas for dataset level lineage graphs.
- Use Apache Ranger to authorize NiFi and NiFi Registry users.
- Use Knox as a single entry point to securely access all NiFi and NiFi Registry nodes, and switch nodes if one fails.

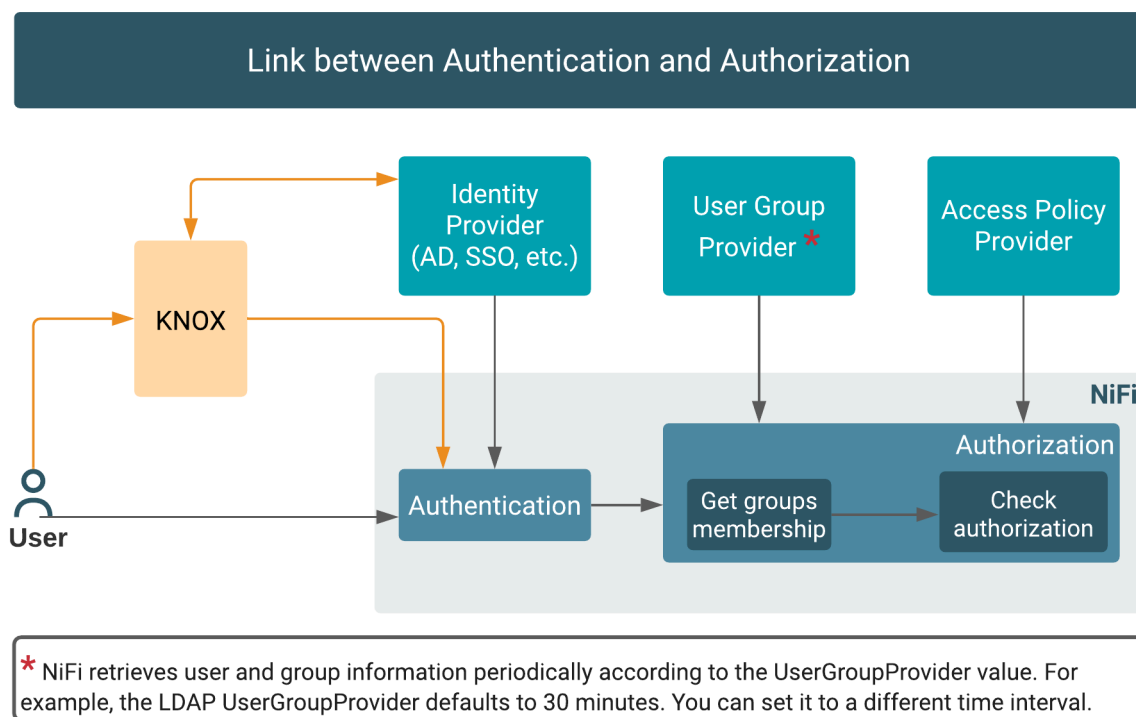


**Important:** Authorization through Apache Ranger is just one element of a secure production cluster: Cloudera supports Ranger only when it runs on a cluster where Kerberos is enabled to authenticate users.

## Identity and policies in Apache NiFi

When a user accesses NiFi, NiFi first determines the identity of the user, then the user group the user belongs to, and then the access policies assigned to the user.

The following image explains the link between authentication and authorization:



When a user accesses NiFi, the following actions take place:

1. NiFi determines the identity of the user:
  - If the user configures a client certificate, the distinguished name associated to the client certificate will be the identity of the user. NiFi nodes use this method to authenticate each other.
  - If the user passes Kerberos credentials along with the access request, the Kerberos principal will be the identity of the user.
  - If the user accesses NiFi through Knox, the authentication (login/password) is done at the Knox level (against the configured identity provider at Knox level) and if the user is allowed to access the NiFi service (Ranger policies defined for Knox), then the user name that is passed to NiFi will be the identity of the user.
2. NiFi determines the group the user belongs to.
3. NiFi determines the policies assigned to the user.

## TLS/SSL configuration

In order to enable user authentication on NiFi, you must first configure Transport Layer Security (TLS).

TLS is an industry standard set of cryptographic protocols for securing communications over a network.

When you configure authentication and authorization for your flow management cluster, Cloudera Flow Management sends sensitive information over the network to cluster hosts, such as Kerberos keytabs and configuration files that contain passwords. TLS encryption keeps these transfers secure.

Configuring TLS involves creating a private key and a public key for use by server and client processes to negotiate an encrypted connection at runtime. In addition, TLS can use certificates to verify the trustworthiness of keys presented during the negotiation to prevent spoofing and mitigate other potential security issues.

In Cloudera Flow Management, you can configure TLS in one of the following ways:

### Enabling Auto-TLS

Auto-TLS greatly simplifies the process of enabling and managing TLS encryption on your cluster.

Auto-TLS automates the creation of an internal certificate authority (CA) and deployment of certificates across all cluster hosts. It can also automate the distribution of existing certificates, such as those signed by a public CA. Adding new cluster hosts or services to a cluster that is Auto-TLS enabled, automatically creates and deploys the required certificates.

In Cloudera, Auto-TLS is enabled by default.



**Note:** Wildcard certificates are not supported. For example, if two nodes, `node1.nifi.apache.org` and `node2.nifi.apache.org`, are assigned the same certificate with a CN or SAN entry of `*.nifi.apache.org`, the certificates will not be supported.

Ensure that you do not generate wildcard certificates for the NiFi nodes.

For more information about Auto-TLS, see *Configuring TLS encryption for Cloudera Manager using Auto-TLS*.

#### Related Information

[Manually Configuring TLS Encryption for Cloudera Manager Using Auto-TLS](#)

### Configuring TLS/SSL manually

If you use your own enterprise-generated certificates, you would need to manually configure TLS.

#### TLS/SSL certificate requirements and recommendations

If you use your own enterprise-generated certificates, you would need to manually configure TLS.

Before you manually configure TLS, ensure that the certificate that you use meets the following requirements.

## Certificate requirements

Verify the following minimum requirements:

- The KeyStore must contain only one PrivateKeyEntry. Using multiple private keys in one KeyStore is not supported.
- The KeyStore password and key/certificate password must be the same or no password should be set on the certificate.
- The unique KeyStores used on each NiFi cluster node must use the same KeyStore password and key/certificate password. Ambari and Cloudera Manager do not support defining unique passwords per NiFi host.
- The X509v3 ExtendedKeyUsages section of the certificate must have the following attributes:
  - clientAuth - This attribute is for TLS web client authentication.
  - serverAuth - This attribute is for TLS web server authentication.
- The signature algorithm used for the certificate must be sha256WithRSAEncryption (SHA-256).
- The certificates must not use wildcards. Each cluster node must have its own certificate. If NiFi or NiFi Registry is behind Knox, do not use wildcard certificates for Knox.
- Subject Alternate Names (SANs) are mandatory and should at least include the FQDN of the host.
- Additional names for the certificate/host can be added to the certificate as SANs.
  - Add the FQDN used for the CN as a DNS SAN entry.
  - If you are planning to use a load balancer for the NiFi service, include the FQDN for the load balancer as a DNS SAN entry.
- The X509v3 KeyUsage section of the certificate must include the following attributes:
  - DigitalSignature
  - Key\_Encipherment

## Cloudera recommendations

Cloudera recommends the following security protocols:

- Use certificates that are signed by a CA. Do not issue self-signed certificates.
- Generate a unique certificate per host.

## Configuring TLS/SSL encryption manually for NiFi and NiFi Registry

If you do not want to enable Auto-TLS because for example, you need to use your own enterprise-generated certificates, you can manually enable TLS for NiFi and NiFi Registry.

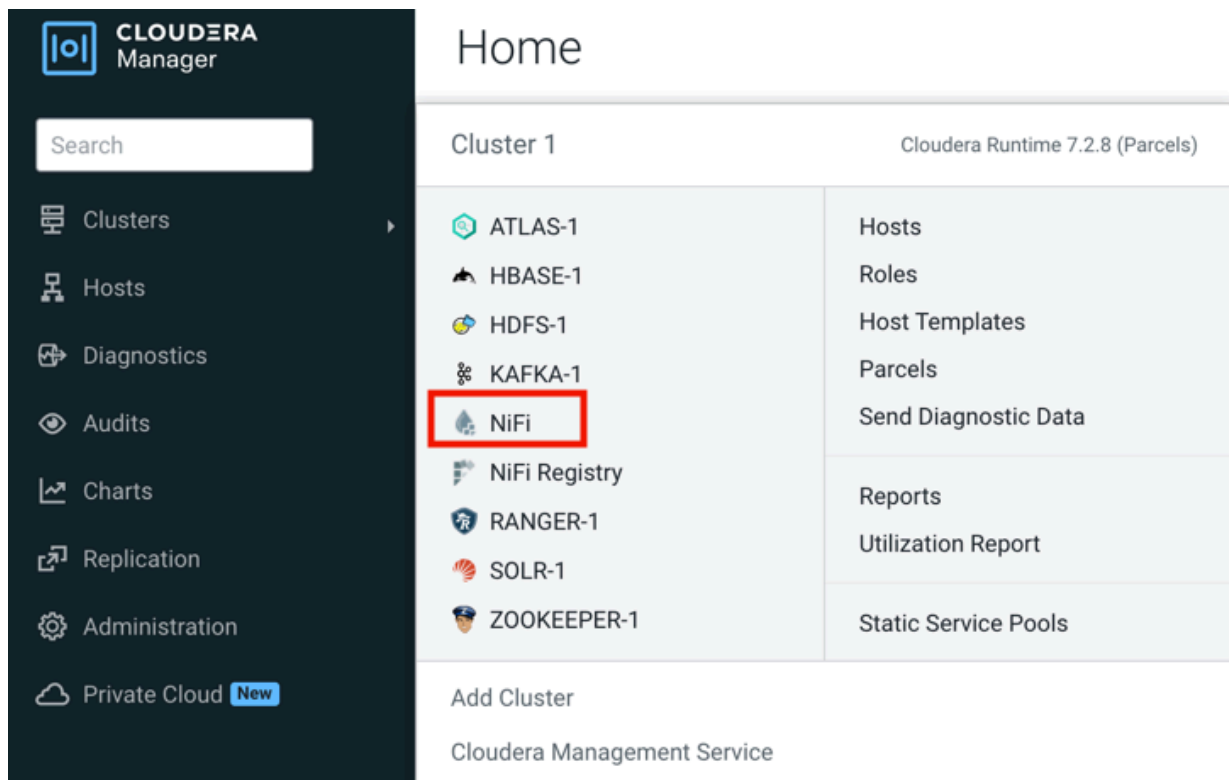
### Before you begin

Ensure you have set up TLS for Cloudera Manager:

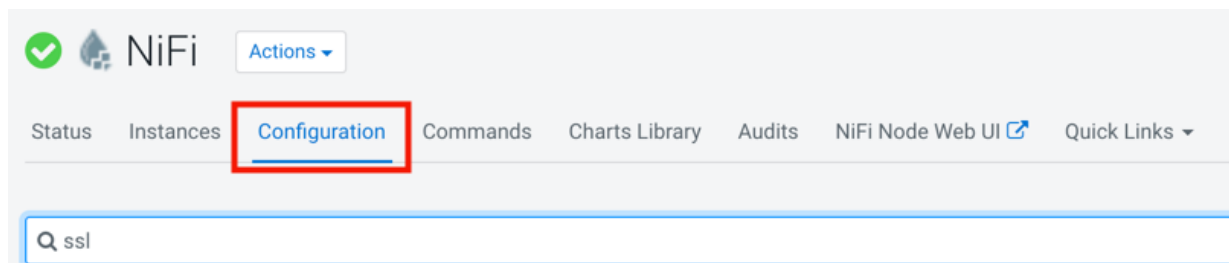
1. Review the requirements and recommendations for the certificates. For more information, see *TLS certificate requirements and recommendations*.
2. Generate the TLS certificates and configure Cloudera Manager. For more information, see *Manually configuring TLS for Cloudera Manager*.

## Procedure

1. From Cloudera Manager, click Cluster NiFi .



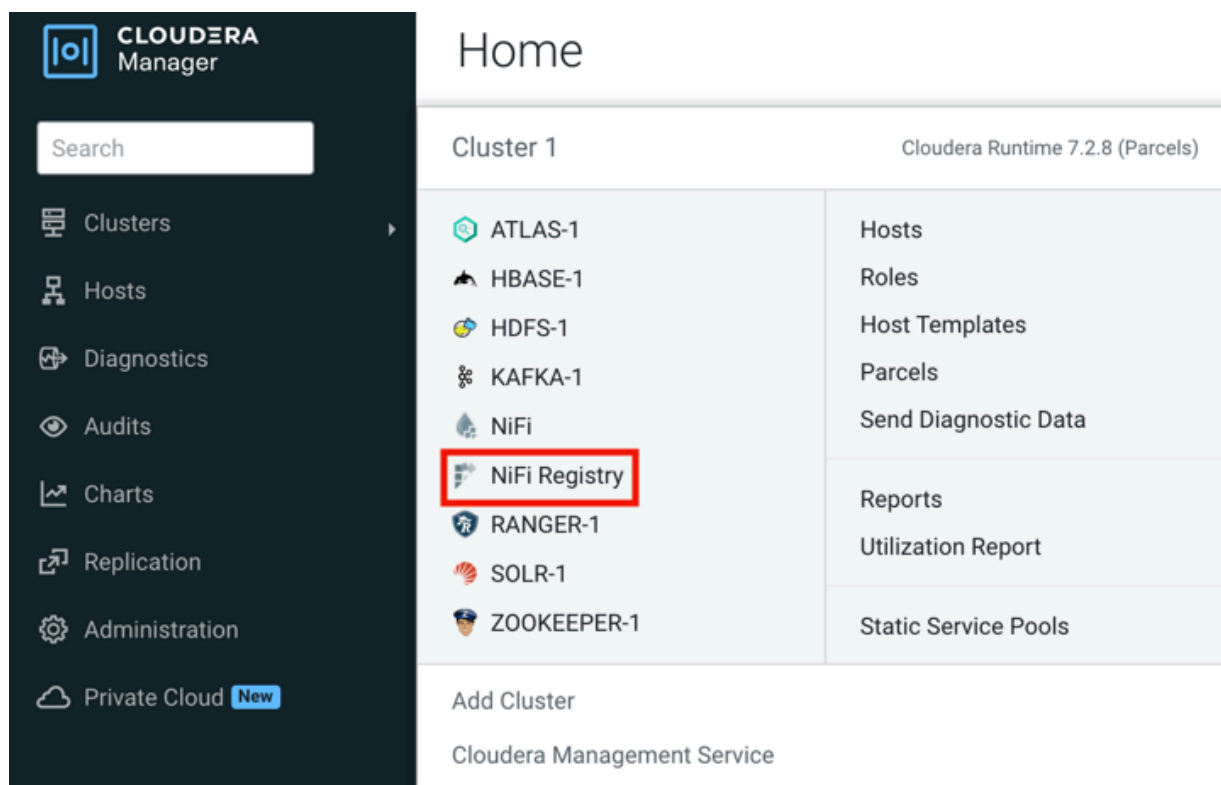
2. Click the **Configuration** tab.



3. Enter ssl in the Search field.  
The TLS/SSL Security properties for NiFi appear.
4. Edit the TLS/SSL Security properties.
5. Click Save Changes.
6. Restart the NiFi service.

- Click **Cluster NiFi Registry** and repeat these steps to configure the TLS/SSL Security properties for NiFi Registry.

If a property is not exposed in Cloudera Manager, use a safety valve to override the associated value.



### Related Information

[TLS/SSL certificate requirements and recommendations](#)

[Manually Configuring TLS Encryption for Cloudera Manager](#)

## NiFi TLS/SSL properties

To enable and configure TLS manually for NiFi, edit the security properties according to the cluster configuration.

The following table lists the TLS/SSL security properties for NiFi:

Property	Description
NiFi Node TLS/SSL Server JKS Keystore File Location nifi.security.keystore	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when NiFi Node is acting as a TLS/SSL server. The keystore must be in JKS format.
NiFi Node TLS/SSL Server JKS Keystore Type Password nifi.security.keystoreType	The type of the NiFi Node JKS keystore. It must be PKCS12 or JKS or BCFKS. JKS is the preferred type, BCFKS and PKCS12 files are loaded with BouncyCastle provider.
NiFi Node TLS/SSL Server JKS Keystore File Password nifi.security.keystorePasswd	The password for the NiFi Node JKS keystore file.
NiFi Node TLS/SSL Server JKS Keystore Key Password nifi.security.keyPasswd	The password that protects the private key contained in the JKS keystore used when NiFi Node is acting as a TLS/SSL server.
NiFi Node TLS/SSL Client Trust Store File nifi.security.truststore	The location on disk of the trust store, in JKS format, used to confirm the authenticity of TLS/SSL servers that NiFi Node might connect to. This is used when NiFi Node is the client in a TLS/SSL connection. This trust store must contain the certificate(s) used to sign the service(s) connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.

Property	Description
NiFi Node TLS/SSL Client Trust Store Type <code>nifi.security.truststoreType</code>	The type of the NiFi Node TLS/SSL Certificate Trust Store. It must be PKCS12 or JKS or BCFKS. JKS is the preferred type, BCFKS and PKCS12 files are loaded with BouncyCastle provider.
NiFi Node TLS/SSL Client Trust Store Password <code>nifi.security.truststorePasswd</code>	The password for the NiFi Node TLS/SSL Certificate Trust Store File. This password is not required to access the trust store, the field can be left blank. This password provides optional integrity checking of the file. The contents of trust stores are certificates, and certificates are public information.



**Note:** Make sure to fill in all properties or NiFi will not start.

## NiFi Registry TLS/SSL properties

To enable and configure TLS manually for NiFi Registry, edit the security properties according to the cluster configuration.

The following table lists the TLS/SSL security properties for NiFi Registry:

Property	Description
NiFi Registry TLS/SSL Server JKS Keystore File Location <code>nifi.registry.security.keystore</code>	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when NiFi Registry is acting as a TLS/SSL server. The keystore must be in JKS format.
NiFi Registry TLS/SSL Server JKS Keystore Type Password <code>nifi.registry.security.keystoreType</code>	The type of the NiFi Registry JKS keystore. It must be PKCS12 or JKS or BCFKS. JKS is the preferred type, BCFKS and PKCS12 files are loaded with BouncyCastle provider.
NiFi Registry TLS/SSL Server JKS Keystore File Password <code>nifi.registry.security.keystorePasswd</code>	The password for the NiFi Registry JKS keystore file.
NiFi Registry TLS/SSL Server JKS Keystore Key Password <code>nifi.registry.security.keyPasswd</code>	The password that protects the private key contained in the JKS keystore used when NiFi Registry is acting as a TLS/SSL server.
NiFi Registry TLS/SSL Client Trust Store File <code>nifi.registry.security.truststore</code>	The location on disk of the trust store, in JKS format, used to confirm the authenticity of TLS/SSL servers that NiFi Registry might connect to. This is used when NiFi Registry is the client in a TLS/SSL connection. This trust store must contain the certificate(s) used to sign the service(s) connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
NiFi Registry TLS/SSL Client Trust Store Type <code>nifi.registry.security.truststoreType</code>	The type of the NiFi Registry TLS/SSL Certificate Trust Store. It must be PKCS12 or JKS or BCFKS. JKS is the preferred type, BCFKS and PKCS12 files are loaded with BouncyCastle provider.
NiFi Registry TLS/SSL Client Trust Store Password <code>nifi.registry.security.truststorePasswd</code>	The password for the NiFi Registry TLS/SSL Certificate Trust Store File. This password is not required to access the trust store; this field can be left blank. This password provides optional integrity checking of the file. The contents of trust stores are certificates, and certificates are public information.
NiFi Registry TLS/SSL Client Authentication <code>nifi.registry.security.needClientAuth</code>	This specifies that connecting clients must authenticate with a client cert. The default value is true. Setting the property to false will specify that connecting clients may optionally authenticate with a client cert, but may also login with a username and password against a configured identity provider.



**Note:** Make sure to fill in all properties or NiFi Registry will not start.

## Authentication

TLS/SSL must be enabled before NiFi can support any form of user authentication. The primary mechanisms of authenticating to NiFi and NiFi Registry in a Cloudera Base on premises cluster are Kerberos and LDAP.

NiFi also supports user authentication through a Security Assertion Markup Language (SAML) and OpenID Connect provider.

### Kerberos authentication

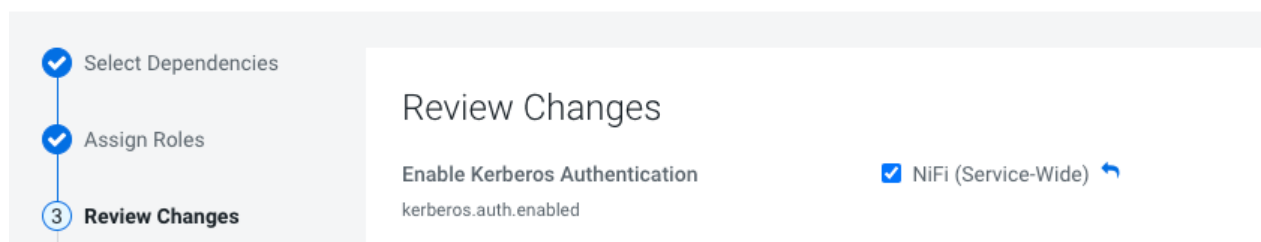
Authenticate your cluster by enabling Kerberos.



**Important:** You must enable TLS/SSL for NiFi to support authentication.

When you add NiFi or NiFi Registry to a kerberized environment, Cloudera Manager provides the Enable Kerberos Authentication option for the NiFi and NiFi Registry services.

#### Add NiFi Service to Cluster 1



The Enable Kerberos Authentication option is the UI label for the `kerberos.auth.enabled` parameter.

By default, the option is checked (parameter set to true) when you add NiFi or NiFi Registry and selecting a dependent service that is kerberized.

The parameter enables the Kerberos Login Identity Provider which allows access to the NiFi/NiFi Registry UI using a Kerberos principal and password.

When the option is enabled, NiFi and NiFi Registry use Kerberos to interact with external systems such as Ranger and Atlas. If the option is not enabled in a kerberized environment, NiFi and NiFi Registry fail to authenticate to external systems.

Alternatively, when Kerberos is enabled you may also authenticate to the KDC from the command line and then configure your browser to forward your credentials to authenticate through SPNEGO.

### Customizing Kerberos principal

The Kerberos principal for NiFi and NiFi Registry is configured by default to use the same service principal as the default process user. However, you can change the default setting by providing a custom principal in Cloudera Manager.

#### About this task



**Important:** configures services to use the default Kerberos principal names and default system users. 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.

## Procedure

1. Go to your cluster in Cloudera Manager.
2. Select NiFi or NiFi Registry from the list of services.
3. Select the **Configuration** tab.
4. Search for the Kerberos Principal by entering kerberos in the search field.
5. Enter a custom name in the Kerberos Principal field.
6. Click Save changes.
7. Click Action Restart next to the NiFi or NiFi Registry service name to restart the service.

## LDAP authentication


After you install NiFi or NiFi Registry, you can enable LDAP authentication.



**Important:** You must enable TLS/SSL for NiFi to support authentication.


In a kerberized environment, enabling the LDAP Login Identity Provider takes precedence over the Kerberos Login Identity Provider.

Set the following required LDAP parameters for NiFi:

LDAP Parameters for NiFi	Sample Value
Enable TLS/SSL for NiFi Node	Checked
LDAP Enabled	Checked
Login Identity Provider: Default LDAP Provider Class	org.apache.nifi.ldap.LdapProvider
Initial Admin Identity	admin
Login Identity Provider ID	ldap-provider
LDAP Authentication Strategy	SIMPLE, LDAPS, or START_TLS  <b>Note:</b> If you select LDAPS or START_TLS, set the LDAP TLS properties for the keystore and truststore.
LDAP Manager DN	uid=admin,ou=people,dc=hadoop,dc=apache,dc=org
LDAP Manager Password	admin-password
LDAP URL	ldap://<ldap-hostname>:33389
LDAP User Search Base	ou=people,dc=hadoop,dc=apache,dc=org
Login Identity Provider: Default LDAP User Search Filter	uid={0}
Login Identity Provider: Default LDAP Identity Strategy	USE_USERNAME
Login Identity Provider: Default LDAP TLS - Keystore	/<path to>/keystore.jks
Login Identity Provider: Default LDAP TLS - Keystore Password	Default LDAP TLS - Keystore Password
Login Identity Provider: Default LDAP TLS - Keystore Type	JKS or PKCS12
Login Identity Provider: Default LDAP TLS - Truststore	/<path to>/truststore.jks
Login Identity Provider: Default LDAP TLS - Truststore Password	Default LDAP TLS - Truststore Password
Login Identity Provider: Default LDAP TLS - Truststore Type	JKS or PKCS12
TLS - Client Auth	Client authentication policy when connecting to LDAP using LDAPS or START_TLS. Possible values are REQUIRED, WANT, and NONE.

LDAP Parameters for NiFi	Sample Value
TLS - Protocol	Protocol to use when connecting to LDAP using LDAPS or START_TLS. For example, 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.

Set the following required LDAP parameters for NiFi Registry:

LDAP Parameter for NiFi Registry	Sample Value
Enable TLS/SSL for NiFi Registry	Checked
LDAP Enabled	Checked
Identity Provider: Default LDAP Provider Class	org.apache.nifi.registry.security.ldap.LdapIdentityProvider
Initial Admin Identity	admin
Identity Provider Identifier	ldap-provider
LDAP Authentication Strategy	SIMPLE, LDAPS, or START_TLS  <b>Note:</b> If you select LDAPS or START_TLS, set the LDAP TLS properties for the keystore and truststore.
LDAP Manager DN	uid=admin,ou=people,dc=hadoop,dc=apache,dc=org
LDAP Manager Password	admin-password
LDAP URL	ldap://<ldap-hostname>:33389
LDAP User Search Base	ou=people,dc=hadoop,dc=apache,dc=org
Identity Provider: Default LDAP User Search Filter	uid={0}
Identity Provider: Default LDAP Identity Strategy	USE_USERNAME
Client Authentication Required	Unchecked
Identity Provider: Default LDAP TLS - Keystore	/<path to>/keystore.jks
Identity Provider: Default LDAP TLS - Keystore Password	Default LDAP TLS - Keystore Password
Identity Provider: Default LDAP TLS - Keystore Type	JKS or PKCS12
Identity Provider: Default LDAP TLS - Truststore	/<path to>/truststore.jks
Identity Provider: Default LDAP TLS - Truststore Password	Default LDAP TLS - Truststore Password
Identity Provider: Default LDAP TLS - Truststore Type	JKS or PKCS12
TLS - Client Auth	Client authentication policy when connecting to LDAP using LDAPS or START_TLS. Possible values are REQUIRED, WANT, and NONE.
TLS - Protocol	Protocol to use when connecting to LDAP using LDAPS or START_TLS. For example, 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.



**Note:** If during the initial install of NiFi and NiFi Registry, you did not set Initial Admin Identity to the correct LDAP admin user, then for each service select **Actions** **Reset File-based Authorizer Users and Policies**. This will cause a new users.xml and authorizations.xml file to be generated at start up and archives the previous users.xml and authorizations.xml files.

## SAML authentication

After you install NiFi, you can enable authentication through a Security Assertion Markup Language (SAML) identity provider.



**Important:** You must enable TLS/SSL for NiFi to support authentication.

With SAML authentication, when a user attempts to access NiFi, NiFi redirects the user to the corresponding identity provider to log in. After the user logs into the identity provider, the identity provider sends NiFi a response that contains the user's credentials. With knowledge of the user's identity, NiFi can now authenticate the user.

To enable authentication through a SAML identity provider, set the following SAML related properties in the `nifi.properties` file. Then, restart NiFi for the changes in the `nifi.properties` file to take effect. If NiFi is clustered, configuration files must be the same on all nodes.

Property	Description
<code>nifi.security.user.saml.idp.metadata.url</code>	The URL for obtaining the identity provider's metadata.  The metadata can be retrieved from the identity provider through <code>http://</code> or <code>https://</code> , or a local file can be referenced using <code>file://</code> .
<code>nifi.security.user.saml.sp.entity.id</code>	The entity ID of the service provider (i.e. NiFi).  This value will be used as the Issuer for SAML authentication requests and should be a valid URI. In some cases the service provider entity ID must be registered ahead of time with the identity provider.
<code>nifi.security.user.saml.identity.attribute.name</code>	The name of a SAML assertion attribute containing the user's identity.  This property is optional and if not specified, or if the attribute is not found, then the NameID of the Subject will be used.
<code>nifi.security.user.saml.group.attribute.name</code>	The name of a SAML assertion attribute containing group names the user belongs to.  This property is optional, but if populated the groups will be passed along to the authorization process.
<code>nifi.security.user.saml.metadata.signing.enabled</code>	Enables signing of the generated service provider metadata.
<code>nifi.security.user.saml.request.signing.enabled</code>	Controls the value of <code>AuthnRequestsSigned</code> in the generated service provider metadata from <code>nifi-api/access/saml/metadata</code> .  This indicates that the service provider (i.e. NiFi) should not sign authentication requests sent to the identity provider, but the requests may still need to be signed if the identity provider indicates <code>WantAuthnRequestSigned=true</code> .
<code>nifi.security.user.saml.want.assertions.signed</code>	Controls the value of <code>WantAssertionsSigned</code> in the generated service provider metadata from <code>nifi-api/access/saml/metadata</code> .  This indicates that the identity provider should sign assertions, but some identity providers may provide their own configuration for controlling whether assertions are signed.
<code>nifi.security.user.saml.signature.algorithm</code>	The algorithm to use when signing SAML messages. See the <i>Open SAML Signature Constants</i> for a list of valid values.  If not specified, a default of SHA-256 will be used.
<code>nifi.security.user.saml.signature.digest.algorithm</code>	The digest algorithm to use when signing SAML messages. See the <i>Open SAML Signature Constants</i> for a list of valid values.  If not specified, a default of SHA-256 will be used.
<code>nifi.security.user.saml.message.logging.enabled</code>	Enables logging of SAML messages for debugging purposes.
<code>nifi.security.user.saml.authentication.expiration</code>	The expiration of the NiFi JWT that will be produced from a successful SAML authentication response.

Property	Description
nifi.security.user.saml.single.logout.enabled	Enables SAML SingleLogout which causes a logout from NiFi to logout of the identity provider. By default, a logout of NiFi will only remove the NiFi JWT.
nifi.security.user.saml.http.client.truststore.strategy	The truststore strategy when the IDP metadata URL begins with https. A value of JDK indicates to use the JDK's default truststore. A value of NIFT indicates to use the truststore specified by nifi.security.truststore.
nifi.security.user.saml.http.client.connect.timeout	The connection timeout when communicating with the SAML IDP.
nifi.security.user.saml.http.client.read.timeout	The read timeout when communicating with the SAML IDP.

### Related Information

[Open SAML Signature Constants](#)

## OpenID Connect authentication

After you install NiFi, you can enable authentication through OpenID Connect.



**Important:** You must enable TLS/SSL for NiFi to support authentication.

With OpenID Connect authentication, when a user attempts to access NiFi, NiFi redirects the user to the corresponding identity provider to log in. After the user logs into the identity provider, the identity provider sends NiFi a response that contains the user's credentials. With knowledge of the user's identity, NiFi can now authenticate the user.

To enable authentication through OpenID Connect, set the following OpenID Connect related properties in the `nifi.properties` file. Then, restart NiFi for the changes in the `nifi.properties` file to take effect. If NiFi is clustered, configuration files must be the same on all nodes.

Property	Description
nifi.security.user.oidc.discovery.url	The discovery URL for the desired OpenID Connect provider. See <i>OpenID Connect Discovery 1.0</i> .
nifi.security.user.oidc.connect.timeout	Connect timeout when communicating with the OpenID Connect provider.
nifi.security.user.oidc.read.timeout	Read timeout when communicating with the OpenID Connect provider.
nifi.security.user.oidc.client.id	The client id for NiFi after registration with the OpenID Connect provider.
nifi.security.user.oidc.client.secret	The client secret for NiFi after registration with the OpenID Connect provider.
nifi.security.user.oidc.preferred.jwsalgorithm	The preferred algorithm for validating identity tokens.  If this value is blank, it will default to RS256 which is required to be supported by the OpenID Connect provider according to the specification.  If this value is HS256, HS384, or HS512, NiFi will attempt to validate HMAC protected tokens using the specified client secret.  If this value is none, NiFi will attempt to validate unsecured/plain tokens.  Other values for this algorithm will attempt to parse as an RSA or EC algorithm to be used in conjunction with the JSON Web Key (JWK) provided through the <code>jwtks_uri</code> in the metadata found at the discovery URL.

Property	Description
nifi.security.user.oidc.additional.scopes	Comma separated scopes that are sent to OpenID Connect provider in addition to openid and email.
nifi.security.user.oidc.claim.identifying.user	Claim that identifies the user to be logged in; default is email. May need to be requested through nifi.security.user.oidc.additional.scopes before usage.
nifi.security.user.oidc.fallback.claims.identifying.user	Comma separated possible fallback claims used to identify the user in case nifi.security.user.oidc.claim.identifying.user claim is not present for the login user.

### Related Information

[OpenID Connect Discovery 1.0](#)

## Identity mapping properties

Identity mapping properties can be utilized to normalize user identities. When implemented, identities authenticated by different identity providers (certificates, LDAP, Kerberos) are treated the same internally in NiFi. As a result, duplicate users are avoided and user-specific configurations such as authorizations only need to be setup once per user.

The following examples demonstrate normalizing DN's from certificates and principals from Kerberos:

```
nifi.security.identity.mapping.pattern.dn=^CN=(.*?), OU=(.*?), O=(.*?), L=(.*?), ST=(.*?), C=(.*?)$
nifi.security.identity.mapping.value.dn=$1@$2
nifi.security.identity.mapping.transform.dn=NONE
nifi.security.identity.mapping.pattern.kerb=^(.*)/instance@(.*?)$
nifi.security.identity.mapping.value.kerb=$1@$2
nifi.security.identity.mapping.transform.kerb=NONE
```

The last segment of each property is an identifier used to associate the pattern with the replacement value. When a user makes a request to NiFi, their identity is checked to see if it matches each of those patterns in lexicographical order. For the first one that matches, the replacement specified in the nifi.security.identity.mapping.value.xxxx property is used. So a login with

```
CN=localhost, OU=Apache NiFi, O=Apache, L=Santa Monica, ST=CA,
C=US
```

matches the DN mapping pattern above and the DN mapping value \$1@\$2 is applied. The user is normalized to localhost@Apache NiFi.

In addition to mapping, a transform may be applied. The supported versions are NONE (no transform applied), LOWER (identity lowercased), and UPPER (identity uppercased). If not specified, the default value is NONE.



**Note:** These mappings are also applied to the "Initial Admin Identity", "Cluster Node Identity", and any legacy users in the authorizers.xml file as well as users imported from LDAP.

Group names can also be mapped. The following example will accept the existing group name but will lowercase it. This may be helpful when used in conjunction with an external authorizer.

```
nifi.security.group.mapping.pattern.anygroup=^(.*)$
nifi.security.group.mapping.value.anygroup=$1
nifi.security.group.mapping.transform.anygroup=LOWER
```



**Note:** These mappings are applied to any legacy groups referenced in the authorizers.xml as well as groups imported from LDAP.

# Hardening ZooKeeper Znodes for NiFi security

By default, the ZooKeeper Znodes used by NiFi are not secured, which can expose sensitive data to unauthorized access. This section provides instructions for identifying, validating, and hardening the ZooKeeper Znodes used by NiFi. These steps are essential for securing NiFi's interaction with ZooKeeper in a production environment.

## Preparations

Run the following commands on one of your NiFi nodes.

### Procedure

#### 1. Identify the unhardened Znode.

By default, the Znode used by NiFi in Zookeeper is not secured. To identify the root Znode used by NiFi in your Zookeeper cluster, follow these steps:

##### a) Identify the Znode used by NiFi.

```
grep "nifi.zookeeper.root.node" $(ps -aux | grep -Eo "[***/var/[^=]*nifi\n\\.properties***)" )
```

This command checks the nifi.properties file for the nifi.zookeeper.root.node property and outputs the exact Znode name. For example:

```
[root@ccycloud-4.quasar-nncuew.root.comops.site 215-nifi-NIFI_NODE]# gre  
p "nifi.zookeeper.root.node" nifi.properties  
nifi.zookeeper.root.node=/1
```

Use this Znode name in all subsequent steps to ensure you are modifying the correct configuration.

##### b) Verify the current Znode configuration.

Once you have determined your Znode, you can validate it and its subnodes in Zookeeper, using the following commands. For demonstration, in the syntax Znode is referred to as /BASE. Replace it with your determined Znode name.

#### 1. List the contents of the Znode:

```
[zk: localhost:2181(CONNECTED) 2] ls /BASE  
[leaders]
```

#### 2. Check the access control list (ACL) of the Znode:

```
[zk: localhost:2181(CONNECTED) 3] getAcl /BASE  
'world,'anyone  
: cdrwa
```

#### 3. Check the ACL of subnodes:

```
[zk: localhost:2181(CONNECTED) 4] getAcl /BASE/leaders  
'world,'anyone  
: cdrwa
```

## 2. Prepare the JAAS file.

- a) Create a JAAS configuration file to Kerberize the NiFi ZooKeeper client.

Example configuration:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="/hadoopfs/fs1/working-dir/nifi.keytab"
  storeKey=true
  useTicketCache=false
  principal="nifi/zz-mwies-nifi-nifil.zz-mwies.a465-9q4k.cloudera.site@ZZ-
  MWIES.A465-9Q4K.CLOUDERA.SITE";
};
```

### Keytab path

The keytab file path is typically located where `nifi.working.directory` is configured. For on-premises deployments, the default path is `/var/lib/nifi`, unless otherwise specified.

### Per-node configuration

If you have multiple nodes, remember that the principal name is unique to each node because it includes the hostname. You will need to create a slightly modified JAAS configuration for each node to reflect its specific principal.

### Principal

To determine the exact principal:

1. Log in to the Cloudera Manager UI.
  2. Go to Administration Security Kerberos Credentials .
  3. Locate the principal for NiFi.
- b) Save this file in a location accessible to NiFi, such as `/etc/nifi-jaas/jaas.conf`.
- ## 3. Validate the JAAS file.

To ensure the JAAS configuration file is valid, test it against Zookeeper to see if SASL authentication is successful.

- a) Export the JVM flags.

```
export CLIENT_JVMFLAGS="-Djava.security.auth.login.config=/etc/nifi-jaas
/jaas.conf"
```

- b) Connect to ZooKeeper.

```
zookeeper-client -server $(hostname -f):2181
```

- c) Confirm the output.

```
Connecting to zz-mwies-nifi-nifil.zz-mwies.a465-9q4k.cloudera.site:2181
Welcome to ZooKeeper!
JLine support is enabled
[zk: zz-mwies-nifi-nifil.zz-mwies.a465-9q4k.cloudera.site:2181(CONNEC
TING) 0]
WATCHER::
WatchedEvent state:SyncConnected type:None path:null
WATCHER::
WatchedEvent state:SaslAuthenticated type:None path:null
```

- d) If you see `state:SaslAuthenticated`, the JAAS file has been created correctly. If there are any issues, use the `cat -vet` command to identify and fix malformations in the file.
- e) Distribute the JAAS files to each NiFi node and change the hostname accordingly.

## Reconfiguring NiFi

This section guides you through reconfiguring NiFi for secure communication with ZooKeeper using Kerberos authentication. The process involves updating several NiFi configuration files with advanced settings (safety valves) to ensure proper integration and security.

### About this task

**Important:**

Follow the instructions carefully to avoid misconfiguration. Do not start or restart NiFi unless explicitly instructed to do so, as early restart may disrupt the cluster setup or lead to security vulnerabilities.

### Procedure

1. For each safety valve, add the corresponding XML snippet.

NiFi Node Advanced Configuration Snippet (Safety Valve) for staging/bootstrap.conf.xml:

```
<property><name>java.arg.15</name><value>-Djava.security.auth.login.config=/etc/nifi-jaas/jaas.conf</value></property>
```

NiFi Node Advanced Configuration Snippet (Safety Valve) for staging/nifi.properties.xml:

```
<property><name>nifi.zookeeper.auth.type</name><value>sasl</value></property><property><name>nifi.zookeeper.kerberos.removeHostFromPrincipal</name><value>true</value></property><property><name>nifi.zookeeper.kerberos.removeRealmFromPrincipal</name><value>true</value></property>
```

NiFi Node Advanced Configuration Snippet (Safety Valve) for staging/state-management.xml:

```
<property><name>xml.state-management.cluster-provider.zk-provider.property.Access Control</name><value>CreatorOnly</value><final>true</final></property>
```

2. Save all the changes made to the configuration files.
3. Stop NiFi.

**Important:**

Do not restart NiFi!

## Znode hardening

### Procedure

1. Back up Components Znode.

Before proceeding, in case there is a components Znode, ensure a backup of it is created.

```
/opt/cloudera/parcels/CFM-***[version]***/TOOLKIT/bin/zk-migrator.sh -r -z $(hostname -f):2181/BASE/components -k /etc/nifi-jaas/jaas.conf -f components-backup.json
```

This command saves the content of components Znode into a JSON file.

**2. Recreate Components Znode.**

- a) Delete the existing components Znode in Zookeeper.

```
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 0] deleteall /BASE/components
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 1] ls /BASE
[]
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 2] quit
WATCHER::
WatchedEvent state:Closed type:None path:null
```

- b) Re-import the components Znode from the backup file.

```
[root@zz-mwies-nifi-nifi2 backup]# /opt/cloudera/parcels/CFM-***[version]*** /TOOLKIT/bin/zk-migrator.sh -s -z $(hostname -f):2181/BASE/component s -k /etc/nifi-jaas/jaas.conf -f components-backup.json --ignore-source
```

- c) After recreating the components Znode, verify that it has the correct permissions and content.

```
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 0] ls /BASE
[components]
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 1] getAcl /BASE/components
'sasl,'nifi
: cdrwa
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 2] ls -R /BASE/components
/BASE/components
/BASE/components/0c2f3aaf-0188-1000-ffff-ffffa939f0e7
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 3] getAcl /BASE/components/0c2f3aaf-0188-1000-ffff-ffffa939f0e7
'sasl,'nifi
: cdrwa
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 4]
```

**3. Set the permissions for BASE Znode.**

To harden the BASE Znode, open Zookeeper and set the appropriate ACL using the following command.

```
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 5] setAcl /BASE sasl:nifi:cdrwa,world:anyone:r
[zk: zz-mwies-nifi-nifi2.zz-mwies.a465-9q4k.cloudera.site:2181(CONNECTED) 6] getAcl /BASE
'sasl,'nifi
: cdrwa
'world,'anyone
: r
```

If you have a /BASE/leaders Znode, set its ACL recursively.

Similarly, set the ACL for /BASE/components with the same permissions:

```
setAcl /BASE/components sasl:nifi:cdrwa,world:anyone:r
```

**4. Start NiFi.**

Once all configurations are verified and updated, you can safely start the NiFi service.

# Authorization

Authorization can be managed using Apache Ranger or through the internal file-based authorizer provided by NiFi and NiFi Registry.

## User group providers

This section provides information on user group providers.

### LDAP integration

After Ranger or file-based authorizations are implemented, the authorizations can be configured to integrate with LDAP.

However, after installation, the authorization configuration can be re-configured to setup an LDAPUserGroupProvider.

When you setup an LDAPUserGroupProvider, the FileUserGroupProvider is replaced with the LDAPUserGroupProvider.



**Note:** The following is an important distinction between the FileUserGroupProvider and the LDAPUserGroupProvider:

- When using the FileUserGroupProvider, the composite provider is the CompositeConfigurableUserGroupProvider.
- When using the LDAPUserGroupProvider, the provider is the non-configurable CompositeUserGroupProvider.

### LDAP and Ranger policies

Set up the LDAP and Ranger integration in NiFi and NiFi Registry.

### About this task

Each authorizers.xml file produced in NiFi and NiFi Registry when using LDAP with Ranger policies, contain the following logical configuration:

- CompositeUserGroupProvider
  - LdapUserGroupProvider
  - CMUserGroupProvider
- RangerAuthorizer
  - Configured with CompositeUserGroupProvider



**Note:** Confer with your Active Directory/LDAP team to get the values you would need to set the LDAP User Group Provider properties. For a list of the properties, see *LDAP User Group Provider properties*.

### Procedure

1. From Cloudera Manager, select the NiFi/NiFi Registry Service, and click the **Configuration** tab.
2. Uncheck Authorizers: Enable File User Group Provider to disable the file-user-group-provider.
3. Uncheck Authorizers: Enable Composite Configurable User Group Provider to disable the composite-configurable-user-group-provider.

4. Check Authorizers: Enable Composite User Group Provider to enable composite-user-group-provider.
  - a) Enter ldap-user-group-provider for Authorizers: Composite User Group Provider Property - User Group Provider 1.
  - b) Enter cm-user-group-provider for Authorizers: Composite User Group Provider Property - User Group Provider 2.
5. Check LDAP Enabled to enable ldap-user-group-provider.
6. In the Search field, enter ldap-user-group-provider to see the list of the LDAP User Group Provider properties. For a list of the properties, see *LDAP User Group Provider properties*.
7. Update the LDAP User Group Provider properties.
8. Update Authorizers: Ranger Authorizer Property - User Group Provider to use the composite-user-group-provider instead of the configurable one.
9. Save the changes.
10. Locate the Login Identity Provider ID and verify that it is set to your authentication provider. Either:
  - kerberos-provider
  - or
  - ldap-provider

### Related Information

[LDAP User Group Provider properties](#)

### LDAP and file-based policies

Set up the LDAP and file-based integration in NiFi and NiFi Registry.

### About this task

Each authorizers.xml file produced in NiFi and NiFi Registry when using LDAP with file-based policies, contain the following logical configuration:

- CompositeUserGroupProvider
  - LdapUserGroupProvider
  - CMUserGroupProvider
- FileAccessPolicyProvider
  - Configured with CompositeUserGroupProvider
- StandardManagedAuthorizer
  - Configured with FileAccessPolicyProvider



**Note:** Confer with your Active Directory/LDAP team to get the values you would need to set the LDAP User Group Provider properties. For a list of the properties, see *LDAP User Group Provider properties*.

### Procedure

1. From Cloudera Manager, select the NiFi/NiFi Registry Service, and click the **Configuration** tab.
2. Uncheck Authorizers: Enable File User Group Provider to disable the file-user-group-provider.
3. Uncheck Authorizers: Enable Composite Configurable User Group Provider to disable the composite-configurable-user-group-provider.
4. Check Authorizers: Enable Composite User Group Provider to enable composite-user-group-provider.
  - a) Enter ldap-user-group-provider for Authorizers: Composite User Group Provider Property - User Group Provider 1.
  - b) Enter cm-user-group-provider for Authorizers: Composite User Group Provider Property - User Group Provider 2.
5. Check LDAP Enabled to enable ldap-user-group-provider.

6. In the Search field, enter ldap-user-group-provider to see the list of the LDAP User Group Provider properties.  
For a list of the properties, see *LDAP User Group Provider properties*.
7. Update the LDAP User Group Provider properties.
8. Update Authorizers: Default File Access Policy Property - User Group Provider to use the composite-user-group-provider instead of the configurable one.
9. Save the changes.
10. Locate the Login Identity Provider ID and verify that it is set to your authentication provider.  
Either kerberos-provider or ldap-provider.

## Related Information

### LDAP User Group Provider properties

#### LDAP User Group Provider properties

After you enable authorization through Ranger or file-based policies, set the LDAP User Group Provider properties to enable NiFi/NiFi Registry to sync users and user groups and determine the association between them.

Set the following LDAP User Group Provider properties (ldap-user-group-provider) in the Cloudera Manager **Configuration** tab.

LDAP User Group Provider properties	Description
Authorizers: LDAP Authentication Strategy	How the connection to the LDAP server is authenticated. Possible values are ANONYMOUS, SIMPLE, LDAPS, or START_TLS.
Authorizers: LDAP Manager DN	The DN of the manager that is used to bind to the LDAP server to search for users.
Authorizers: LDAP Manager Password	The password of the manager that is used to bind to the LDAP server to search for users.
Authorizers: LDAP TLS - Keystore	Path to the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
Authorizers: LDAP TLS - Keystore Password	Password for the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
Authorizers: LDAP TLS - Keystore Type	Type of the Keystore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
Authorizers: LDAP TLS - Truststore	Path to the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
Authorizers: LDAP TLS - Truststore Password	Password for the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
Authorizers: LDAP TLS - Truststore Type	Type of the Truststore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
Authorizers: LDAP TLS - Client Auth	Client authentication policy when connecting to LDAP using LDAPS or START_TLS. Possible values are REQUIRED, WANT, NONE.
Authorizers: LDAP TLS - Protocol	Protocol to use when connecting to LDAP using LDAPS or START_TLS. (i.e. TLS, TLSv1.1, TLSv1.2, etc).
Authorizers: LDAP TLS - Shutdown Gracefully	Specifies whether the TLS should be shut down gracefully before the target context is closed. Defaults to false.
Authorizers: LDAP Referral Strategy	Strategy for handling referrals. Possible values are FOLLOW, IGNORE, THROW.
Authorizers: LDAP Connect Timeout	Duration of connect timeout. (i.e. 10 secs).
Authorizers: LDAP Read Timeout	Duration of read timeout. (i.e. 10 secs).
Authorizers: LDAP Url	Space-separated list of URLs of the LDAP servers (i.e. ldap://<host name>:<port>).
Authorizers: LDAP Page Size	Sets the page size when retrieving users and groups. If not specified, no paging is performed.

LDAP User Group Provider properties	Description
Authorizers: LDAP Group Membership - Enforce Case Sensitivity	Sets whether group membership decisions are case sensitive. When a user or group is inferred (by not specifying user or group search base or user identity attribute or group name attribute) case sensitivity is enforced since the value to use for the user identity or group name would be ambiguous. Defaults to false.
Authorizers: LDAP Sync Interval	Duration of time between syncing users and groups. (i.e. 30 mins). Minimum allowable value is 10 secs.
Authorizers: LDAP User Search Base	Base DN for searching for users (i.e. ou=users,o=nifi). Required to search users.
Authorizers: LDAP User Object Class	Object class for identifying users (i.e. person). Required if searching users.
Authorizers: LDAP User Search Scope	Search scope for searching users (ONE_LEVEL, OBJECT, or SUBTREE). Required if searching users.
Authorizers: LDAP User Search Filter	Filter for searching for users against the User Search Base (i.e. (memberof=cn=team1,ou=groups,o=nifi)). Optional.
Authorizers: LDAP User Identity Attribute	Attribute to use to extract user identity (i.e. cn). Optional. If not set, the entire DN is used.
Authorizers: LDAP 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. The value of this property is the name of the attribute in the user ldap entry that associates them with a group. The value of that user attribute could be a dn or group name for instance. What value is expected is configured in the User Group Name Attribute - Referenced Group Attribute.
Authorizers: LDAP 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.
Authorizers: LDAP Group Search Base	Base DN for searching for groups (i.e. ou=groups,o=nifi). Required to search groups.
Authorizers: LDAP Group Object Class	Object class for identifying groups (i.e. groupOfNames). Required if searching groups.
Authorizers: LDAP Group Search Scope	Search scope for searching groups (ONE_LEVEL, OBJECT, or SUBTREE). Required if searching groups.
Authorizers: LDAP Group Search Filter	Filter for searching for groups against the Group Search Base. Optional.
Authorizers: LDAP Group Name Attribute	Attribute to use to extract group name (i.e. cn). Optional. If not set, the entire DN is used.
Authorizers: LDAP 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 membership being defined through User Group Name Attribute if set. The value of this property is the name of the attribute in the group ldap entry that associates them with a user. The value of that group attribute could be a dn or memberId for instance. What value is expected is configured in the Group Member Attribute - Referenced User Attribute. (i.e. member: cn=User 1,ou=users,o=nifi vs. memberId: user1)
Authorizers: LDAP 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 vs. memberId: user1)

## Pairing LDAP with a Composite Group Provider

If you need to combine multiple user/group provider mechanisms into a composite provider, you can do so using the Cloudera Manager safety valves for the `authorizers.xml` file.

This example shows how file-based users/group provider can be paired with an LDAP user group provider using a `CompositeConfigurableUserGroupProvider`.

Property Name	Property Value (Default)
<code>xml.authorizers.userGroupProvider.composite-user-group-provider.property.User Group Provider 1</code>	<code>ldap-user-group-provider</code>
<code>xml.authorizers.userGroupProvider.composite-configurable-user-group-provider.property.User Group Provider 2</code>	<code>ldap-user-group-provider</code>

## Access policies providers

This section provides information on access policies providers.

### Ranger authorization

Leverage Apache Ranger access policies to administer permissions for groups or individual users.

A Ranger access policy for flow management contains one or more access rights to NiFi or NiFi Registry resources in a cluster. You can add users and groups to a predefined policy or you can create a custom policy to add users and groups to.



**Note:** The Ranger predefined policies are only available if you selected Ranger as your authorization option when you deployed Cloudera Flow Management.

### Understanding the Ranger authorization process for Cloudera Flow Management

When you select Ranger during the installation process, Ranger will be used for NiFi and NiFi Registry authorization. A set of predefined access policies at the controller level and component level will be available for you to assign to users.



**Note:** The Ranger predefined policies are only available if you selected Ranger during the installation process.

When Ranger is selected, the NiFi and NiFi Registry CSD scripts perform the following steps:

- Create a new repository/service in Ranger to store policies for the given NiFi or NiFi Registry instance. Each instance appears on the Ranger UI with a unique name in the following format: `<CM cluster name>_nifi` or `<CM cluster name>_nifiregistry`.

Example: `myCFMcluster_nifi`

- Create policies for the following Initial Admin Identity and Initial Admin Groups:
  - For NiFi: `nifi.initial.admin.identity` and `nifi.initial.admin.groups`
  - For NiFi Registry: `nifi.registry.initial.admin.identity` and `nifi.registry.initial.admin.groups`
- Create policies for proxies specified by `nifi.proxy.group` or `nifi.registry.proxy.group`.

Each `authorizers.xml` file produced in NiFi and NiFi Registry when using Ranger, contains the following logical configuration:

- `CompositeConfigurableUserGroupProvider`
  - `FileUserGroupProvider`
  - `CMUserGroupProvider`
- `RangerAuthorizer`
  - Configured with `CompositeConfigurableUserGroupProvider`

The CMUserGroupProvider has the following purposes:

- Obtain the NiFi node identities (and Knox identity if present) from Cloudera Manager.
- Associate the NiFi node identities with a group.

The group associated with the identities is used as the proxy group that is placed in the Ranger policy for the/proxy resource.



**Note:** Group based authorization policies configured in Ranger does not work unless NiFi is also configured with a NiFi user-group-provider that returns the same group along with its associated users. Default user-group-provider at installation is the file-user-group-provider which requires manually adding users and group associations. For more information, see [User group providers](#).



**Note:** The CMUserGroupProvider is only aware of hostnames. The default identity map maps a configured DN to its CN value. Disabling this identity map will cause the CMUserGroupProvider to stop working.

### Ranger-based NiFi policy descriptions

You can review how NiFi policies defined in Ranger align with NiFi's default file-based authorizer accessible through the NiFi user interface. The focus is on both controller-level policies and component-level policies, showing what access is granted to entities (users and servers) associated with them.

In Apache NiFi, policies are used to control access to various aspects of the system. You can define access policies at the controller or the component level. The combination of the two types of policies allows for a flexible access control mechanism.

### Controller-level policies

Controller level policies provide a higher-level governance framework, overseeing global aspects of NiFi configuration and management. They are not tied to any specific component UUID. In Ranger, these policies are outlined as base policies and they show as `/<policy name>`.

Ranger policy (base policy)	NiFi policy	Ranger permission description
/resources <sup>1</sup>	N/A	Allows Ranger to retrieve a list of NiFi policies. The server/user from the keystore Ranger must be granted read privileges to this resource.
/flow <sup>2</sup>	View user interface	Read/View: allows users to open and view the NiFi UI. Ensure that all users are granted read privileges to this policy, otherwise they will not be able to open the NiFi UI. If you run a NiFi cluster and/or access NiFi through a proxy, you need to grant read access to all nodes and any proxies involved. Write/Modify: N/A
/system	View system diagnostics	Read/View: provides access to system diagnostics, essential for users and nodes in the cluster to display system diagnostic stats returned by other nodes. Write/Modify: N/A
/controller	Access controller	Read/View: grants users and/or NiFi cluster nodes access to view: <ul style="list-style-type: none"> <li>• Controller thread pool configuration</li> <li>• Cluster management page</li> <li>• Controller-level reporting tasks</li> <li>• Controller-level controller services</li> </ul> Write/Modify: enables users and/or NiFi cluster nodes to create/modify: <ul style="list-style-type: none"> <li>• Controller thread pool configuration</li> <li>• Cluster management page</li> <li>• Controller-level reporting tasks</li> <li>• Controller-level controller services</li> </ul>

<sup>1</sup> No policies are available until this policy is manually added.

<sup>2</sup> All users must at a minimum be assigned to the /flow policy to be able to view the NiFi UI.

Ranger policy (base policy)	NiFi policy	Ranger permission description
/counters	Access counters	Read/View: enables users to view counters. Write/Modify: enables users to modify counters.
/provenance	Query provenance	Read/View: allows users to run provenance queries or access provenance lineage g Write/Modify: N/A
/restricted-components <sup>3</sup>	Access restricted components	Read/View: N/A Write/Modify: gives granted users ability to add components to the canvas that are as 'restricted'.
/proxy <sup>4</sup>	Proxy user requests	Read/View: allows proxy servers to send request on behalf of other users. Write/Modify: required
/site-to-site	Retrieve site-to-site details	Read/View: allows other NiFi nodes to retrieve site-to-site details about the current
/policies <sup>5</sup>	Access all policies	Read/View: allows users to view existing policies. Write/Modify: allows users to create new policies and modify existing policies.
/tenants <sup>5</sup>	Access users/user groups	Read/View: allows users to view currently authorized users and user groups. Write/Modify: allows users to add, delete, and modify existing users and user group
/parameter-contexts	Access parameter contexts	Read/View: allows users to view and use existing parameter contexts. Write/Modify: allows users to create, modify, and delete parameter contexts.

### Component-level policies

Component level policies offer more granular access control, allowing administrators to regulate actions at the level of individual components within the NiFi data flow. These policies are based on assigned UUIDs, enforcing the access policies for specific components within NiFi, such as processors, input/output ports, or process groups.

Ranger component-based policies	Equivalent NiFi file based authorizer policy: policy	Ranger permissions description
/data-transfer/input-ports/<uuid>	Each NiFi remote input port is assigned a unique <uuid>	Both read and write are required and should be granted to the source NiFi servers sending data to this NiFi through this input port.

<sup>3</sup> See [NiFi Restricted Components Policy Descriptions](#) for more information.

<sup>4</sup> All nodes in your NiFi cluster must be assigned to the /proxy policy.

<sup>5</sup> In the context of Ranger, using this policy is unnecessary and serves no functional purpose.

Ranger component-based policies	Equivalent NiFi file based authorizer policy: policy	Ranger permissions description
/data-transfer/output-ports/<uuid>	Each data through site-to-site NiFi remote output port is assigned a unique <uuid>	Both read and write are required and should be granted to the source NiFi servers pulling data from this NiFi through this output port.
/process-groups/<uuid>	Each NiFi component	Read: allows users to view process group details only.
	Each process group is assigned a unique <uuid>	Write: allows users to start, stop or delete process group. Users are able to added components inside process group and add controller services to process group.
/data/process-groups/<uuid>	Each data NiFi process	Read: allows users to view data was processed by components in this process group and list queues.
	Each process group is assigned a unique <uuid>	Write: allows users to empty queues/purge data from queues within process group.
/policies/process-groups/<uuid> <sup>6</sup>	Each NiFi policies	Read: N/A in Ranger
	Each process group is assigned a unique <uuid>	Write: N/A in Ranger
/processors/<uuid>	Each NiFi component	Read: allows users to view processor configuration only.
	Each processor is assigned a unique <uuid>	Write: allows users to start, stop, configure and delete processor.
/data/processors/<uuid>	Each data NiFi processor	Read: allows users to view data processed by this processor and list queues on this processor's outbound connections.
	Each processor is assigned a unique <uuid>	Write: allows users to empty queues/purge data from this processor's outbound connections.

<sup>6</sup> Not needed when using Ranger.

Ranger component-based policies	Equivalent NiFi file based authorizer policy: policy	Ranger permissions description
/policies/processors/<uuid> <sup>6</sup>	Each policies	Read: N/A in Ranger
	Each processor is assigned a unique <uuid>	Write: N/A in Ranger
/controller-services/<uuid>	Each component	Read: allows users to view controller service configuration.
	Each controller services is assigned a unique <uuid>	Write: allows users to enable, disable, configure and delete controller services.
/provenance-data/<component-type>/<component-UUID>	Each provenance	Read: allows users to view provenance events generated by this component.
	Each component is assigned a unique <uuid>	Write: N/A in Ranger
/operation/<component-type>/<component-UUID>	Each component	Read: N/A in Ranger
	Each NiFi component is assigned a unique <uuid>	Write: allows users to operate components by changing component run status (start/stop/enable/disable), remote port transmission status, or by terminating processor threads.

Each component is assigned a unique UUID, resulting in a distinct policy for each specific component. Component-level authorizations are inherited from the parent process group when no explicit processor or sub-process group component-level policy is defined. Ranger facilitates policy assignment using the '\*' wildcard, providing a versatile approach to policy configuration.

In a NiFi cluster, all nodes must be granted the ability to view and modify component data in order for users to list or empty queues in processor component outbound connections. With Ranger, you can accomplish this by using a wildcard to grant all the NiFi nodes read and write permissions to the /data/\* NiFi resource.

### Predefined Ranger access policies for Apache NiFi

You can review the predefined Ranger policies for NiFi to determine the appropriate policy to assign to a user.

The following table lists the predefined Ranger access policies for NiFi. If you create a custom policy, refer to the Resource Descriptor column in this table to enter the value in the NiFi Resource Identifier field on the **New Policy** page.



#### Important:


Do not rename the default policies as some cluster operations rely on these policy names.

Do not select the Delegate Admin checkbox.



**Note:** The NiFi and Knox nodes have permission to the following Ranger policies:

- Proxies; /proxy
- Root Group Data; /data/process-groups

Ranger Policy	Description	Resource Descriptor
Controller	Allows users to view and modify the controller including Reporting Tasks, Controller Services, Parameter Contexts and Nodes in the Cluster.	/controller
Flow	Allows users to view the NiFi UI.	/flow
Policies	Allows users to view the policies for all components.	/policies
Provenance	Allows users to submit a Provenance Search and request Event Lineage.	/provenance
Proxies	Allows NiFi and Knox hosts to proxy user requests. Does not apply to users or user groups.	/proxy
Restricted Components	<p>Allows users to create/modify restricted components assuming other permissions are sufficient.</p> <p>The restricted components may indicate the specific permissions that are required.</p> <p>Permissions can be granted for specific restrictions or be granted regardless of restrictions. If permission is granted regardless of restrictions, the user can create/modify all restricted components.</p> <p>Some examples of restricted components are ExecuteScript, List/FetchHDFS, and TailFile.</p>	<p>/restricted-components</p> <p>See the <i>NiFi Restricted Components</i> topic for information on the sub-policies.</p>
Root Group Data	<p>Allows users and the nifi group to view and delete data from the root group and down the hierarchy unless there is a more specific policy on a component.</p> <p> <b>Note:</b> The nifi group is a dynamically managed list of Knox and NiFi node identities. The group exists on all Data Hub Flow Management hosts.</p>	/data/process-groups/<uuid>
Root Group Provenance Data	Allows users to view provenance data.	/provenance-data/process-groups/
Root Process Group	<p>Allows users to view and modify the root process group including adding/removing processors to the canvas.</p> <p>This policy is inherited down the hierarchy unless there is a more specific policy on a component.</p>	/process-groups/<uuid>
Tenants	Allows users to view and modify user accounts and user groups.	/tenants

## Related Information

[Apache NiFi restricted components](#)

## Predefined Ranger access policies for Apache NiFi Registry

You can review the predefined Ranger policies for NiFi Registry to determine the appropriate policy to assign to a user.

The following table lists the pre-defined Ranger access policies for NiFi Registry. If you create a custom policy, refer to the Resource Descriptor column in this table to enter the value in the NiFi Registry Resource Identifier field on the **New Policy** page.



**Important:**

Do not rename the default policies as some cluster operations rely on these policy names.

Do not select the Delegate Admin checkbox.



**Note:** The NiFi Registry and Knox nodes have permission to the Proxies (/proxy) Ranger policy.

Ranger Policy	Description	Resource Descriptor
Actuator	Allows users to access the Spring Boot Actuator end-points.	/actuator
Buckets	Allows users to view and modify all buckets.	/buckets
Policies	Allows users to view the policies for all components.	/policies
Proxies	Allows NiFi Registry and Knox hosts to proxy user requests. Does not apply to users or user groups.	/proxy
Swagger	Allows users to access the self-hosted Swagger UI.	/swagger
Tenants	Allows users to view and modify user accounts and user groups.	/tenants

### Predefined component-level policies for Apache NiFi

The component-level granular policies are based on the UUID of each component. For connections, the policies are enforced based upon the processor component that the connection originates from.

Note the following:

- There is a unique policy for every component based on the specific components assigned UUID.
- Component level authorizations are inherited from the parent process group when no specific processor or sub process group component level policy is set.
- Ranger supports the " \* " wildcard when assigning policies.
- In a NiFi cluster, all nodes must be granted the ability to view and modify component data in order for user to list or empty queues in processor component outbound connections. With Ranger this can be accomplished by using the a wildcard to grant all the NiFi nodes read and write to the /data/\* NiFi resource.

Ranger component-level policies	NiFi component-based policy: Component	Equivalent NiFi file-based authorizer policy: Policy	Ranger permissions
/data-transfer/input-ports/<UUID>	Each NiFi remote input port is assigned a unique <UUID>	Send data through site-to-site.	Both read and write is required and should be granted to the source NiFi servers sending data to this NiFi through this input port.
/data-transfer/output-ports/<UUID>	Each NiFi remote output port is assigned a unique <UUID>	Retrieve data through site-to-site.	Both read and write is required and should be granted to the source NiFi servers pulling data from this NiFi via this output port.
/process-groups/<UUID>	Each NiFi process group is assigned a unique <UUID>	View or modify the component.	Read - (allows user to view process group details only) Write - (allows user to start, stop or delete process group. Users are able to added components inside process group and add controller services to process group)

Ranger component-level policies	NiFi component-based policy: Component	Equivalent NiFi file-based authorizer policy: Policy	Ranger permissions
/data/process-groups/<UUID>	Each NiFi process group is assigned a unique <UUID>	View or modify the data.	Read - (allows user to view data was processed by components in this process group and list queues)Write - (allows users to empty queues/purge data from queues within process group)
 <b>Note:</b> Not needed when using Ranger	Each NiFi process group is assigned a unique <UUID>	View or modify the policies.	Read - N/A in RangerWrite - N/A in Ranger
/processors/<UUID>	Each NiFi processor is assigned a unique <UUID>	View or modify the component.	Read - (Allows user to view processor configuration only)Write - (Allows user to start, stop, configure and delete processor)
/data/processors/<UUID>	Each NiFi processor is assigned a unique <UUID>	View or modify the data.	Read - (allows user to view data processed this processor and list queues on this processors outbound connections)Write - (allows users to empty queues/purge data from this processors outbound connections)
 <b>Note:</b> Not needed when using Ranger	Each NiFi processor is assigned a unique <UUID>	View or modify the policies.	Read - N/A in RangerWrite - N/A in Ranger
/controller-services/<UUID>	Each NiFi controller services is assigned a unique <UUID>	View or modify the component.	Read - (Allows user to view controller service configuration only)Write - (Allows user to enable, disable, configure and delete controller services)
/provenance-data/<component-type>/<component-UUID>	Each NiFi component is assigned a unique <UUID>	View provenance.	Read - Allows users to view provenance events generated by this componentWrite - N/A in Ranger
/operation/<component-type>/<component-UUID>	Each NiFi component is assigned a unique <UUID>	Operate the component.	Read - N/A in RangerWrite - Allows users to operate components by changing component run status (start/stop/enable/disable), remote port transmission status, or terminating processor threads

### Apache NiFi restricted components

As the administrator, you should be aware of the capabilities of NiFi restricted components and explicitly enable them for trusted users.

Restricted components are the processors, controller services, or reporting tasks that have the ability to run user-defined code or access/alter localhost filesystem data using the NiFi OS credentials. An authorized NiFi user can use these components to go beyond the intended use of the application, escalate privilege, or expose data about the internals of the NiFi process or the host system. For this reason, you must grant the user or user group the specific permission they require to the specific restricted component.

The restricted-components policy allows you to fine-tune the permission for each component and also makes a distinction between processors that access the local filesystem where NiFi is running and the processors that access a distributed file system like the Hadoop related processors.

The following list describes the available Ranger restricted-components policies you can use to control access to a restricted component:

**/restricted-components/access-keytab**

Allows users to access the keytab for the restricted component.

**/restricted-components/execute-code**

Allows users to run code for the restricted component.

**/restricted-components/export-nifi-details**

Allows users to export NiFi details accessed by the restricted component.

**/restricted-components/read-filesystem**

Allows users to use processors that require read access to the local filesystem.

**/restricted-components/read-distributed-filesystem**

Allows users to use processors that require read access to the distributed filesystem.

**/restricted-components/write-filesystem**

Allows users to use processors that require write access to the local filesystem.

**/restricted-components/write-distributed-filesystem**

Allows users to use processors that require write access to the distributed filesystem.

The following tables list the restricted components that you can set the /restricted-components/<permission level> for.



**Note:** Some components may be found under multiple restricted-component sub policies. In order for a user to utilize that component, you must be grant the user access to every sub policy required by that component.

**Table 1: Access-keytab**

NiFi component	Component type	Access provisions
KeytabCredentialsService	Controller Service	Allows user to define a Keytab and principal that can then be used by other components.

**Table 2: Execute-code**

NiFi component	Component type	Access provisions
ScriptedReportingTask	Reporting Task	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.
ScriptedLookupService	Controller Service	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.
ScriptedReader	Controller Service	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.
ScriptedRecordSetWriter	Controller Service	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.
ExecuteFlumeSink	Processor	Provides operator the ability to run arbitrary Flume configurations assuming all permissions that NiFi has.
ExecuteFlumeSource	Processor	Provides operator the ability to run arbitrary Flume configurations assuming all permissions that NiFi has.
ExecuteGroovyScript	Processor	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.
ExecuteProcess	Processor	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.

NiFi component	Component type	Access provisions
ExecuteScript	Processor	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.
ExecuteStreamCommand	Processor	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.
invokeScriptedProcessor	Processor	Provides operator the ability to run arbitrary code assuming all permissions that NiFi has.

**Table 3: Export-nifi-details**

NiFi component	Component type	Access provisions
SiteToSiteBulletinReportingTask	Reporting Task	Provides operator the ability to send sensitive details contained in bulletin events to any external system.
SiteToSiteProvenanceReportingTask	Reporting Task	Provides operator the ability to send sensitive details contained in Provenance events to any external system.

**Table 4: Read-filesystem**

NiFi component	Component type	Access provisions
FetchFile	Processor	Provides operator the ability to read from any file that NiFi has access to.
GetFile	Processor	Provides operator the ability to read from any file that NiFi has access to.
TailFile	Processor	Provides operator the ability to read from any file that NiFi has access to.

**Table 5: Read-distributed-filesystem**

NiFi component	Component type	Access provisions
FetchHDFS	Processor	Provides operator the ability to retrieve any file that NiFi has access to in HDFS or the local filesystem.
FetchParquet	Processor	Provides operator the ability to retrieve any file that NiFi has access to in HDFS or the local filesystem.
GetHDFS	Processor	Provides operator the ability to retrieve any file that NiFi has access to in HDFS or the local filesystem.
GetHDFSSequenceFile	Processor	Provides operator the ability to retrieve any file that NiFi has access to in HDFS or the local filesystem.
MoveHDFS	Processor	Provides operator the ability to retrieve any file that NiFi has access to in HDFS or the local filesystem.

**Table 6: Write-filesystem**

NiFi component	Component type	Access provisions
FetchFile	Processor	Provides operator the ability to delete any file that NiFi has access to.
GetFile	Processor	Provides operator the ability to delete any file that NiFi has access to.

NiFi component	Component type	Access provisions
PutFile	Processor	Provides operator the ability to write to any file that NiFi has access to.

**Table 7: Write-distributed-filesystem**

NiFi component	Component type	Access provisions
DeleteHDFS	Processor	Provides operator the ability to delete any file that NiFi has access to in HDFS or the local filesystem.
GetHDFS	Processor	Provides operator the ability to delete any file that NiFi has access to in HDFS or the local filesystem.
GetHDFSSequenceFile	Processor	Provides operator the ability to delete any file that NiFi has access to in HDFS or the local filesystem.
MoveHDFS	Processor	Provides operator the ability to delete any file that NiFi has access to in HDFS or the local filesystem.
PutHDFS	Processor	Provides operator the ability to delete any file that NiFi has access to in HDFS or the local filesystem.
PutParquet	Processor	Provides operator the ability to write any file that NiFi has access to in HDFS or the local filesystem.

**Adding user to a pre-defined Ranger access policy**

When a user attempts to view or modify a NiFi or NiFi Registry resource, the system checks whether this user has privileges to perform that action. These privileges are determined by the Ranger access policies that a user is associated with.

**About this task**

You can determine what the user can command, control, and observe in a NiFi dataflow or in NiFi Registry and accordingly add the user or a group of users to the appropriate pre-defined Ranger access policies.

Each pre-defined Ranger access policy confers specific rights to NiFi or NiFi Registry resources.

For more information, see:

- *Pre-defined Ranger access policies for NiFi resources*
- *Pre-defined Ranger access policies for NiFi Registry resources*

**Before you begin**

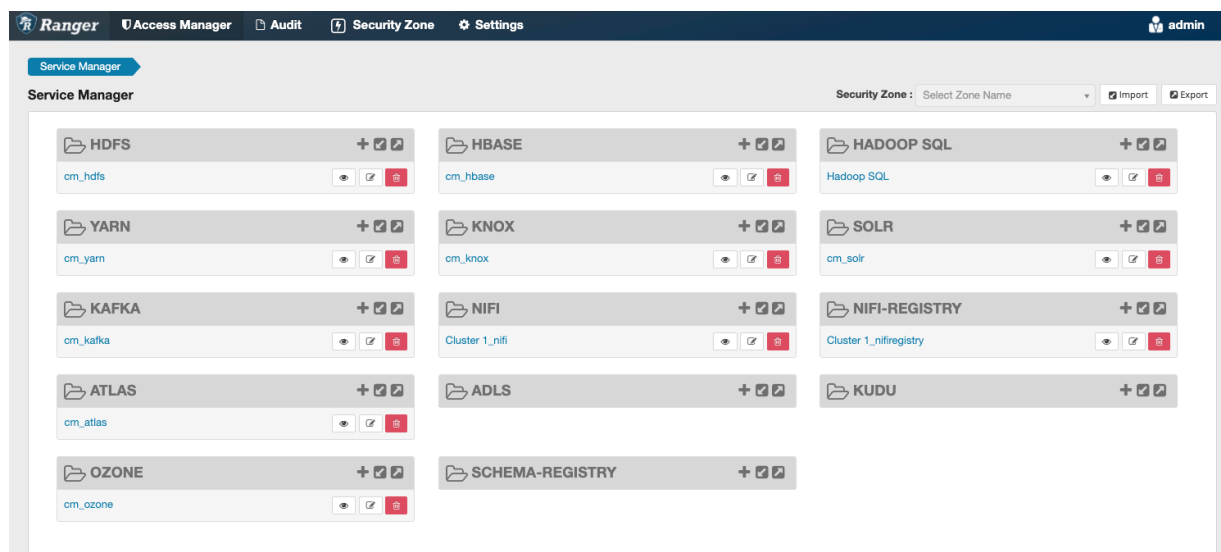
Ensure that you meet the following prerequisites:

- You have installed NiFi and NiFi Registry.
- You have determined the permission level for each user.

## Procedure

1. From the Base cluster, select Ranger from the list of services. Click Ranger Admin Web UI and log into Ranger.

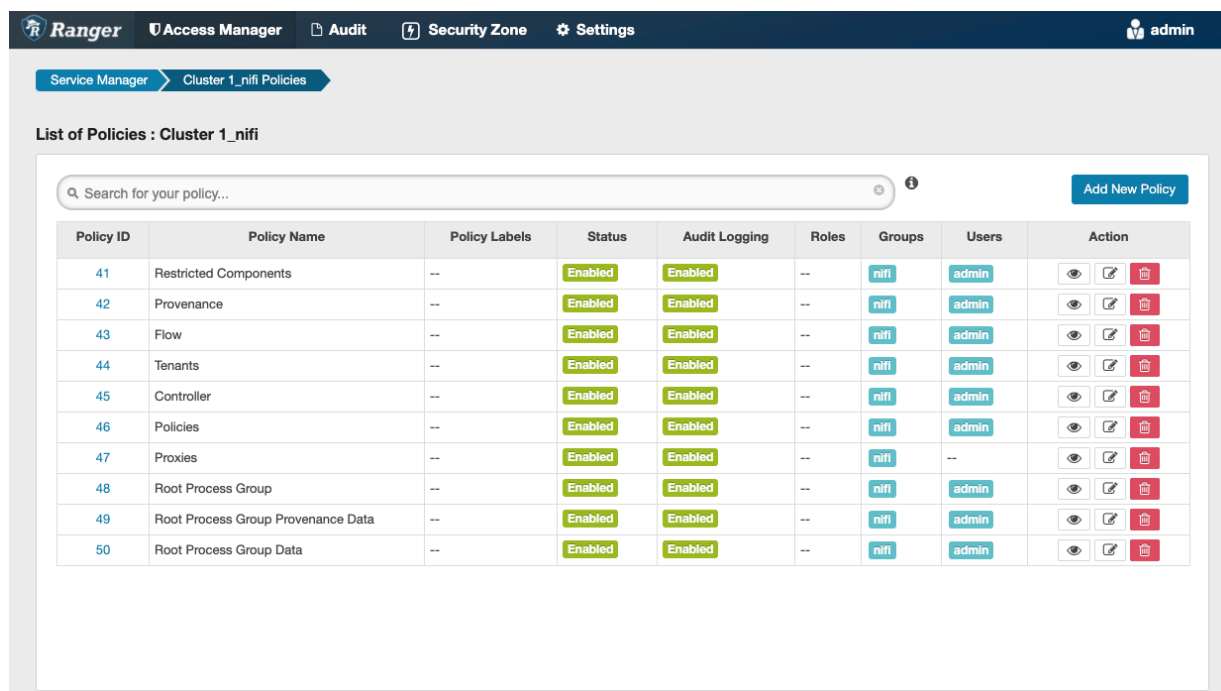
The **Ranger Service Manager** page displays.



Each cluster in the environment is listed under its respective service. For example, the NiFi clusters in the environment are listed under NiFi.

2. Select a cluster from either the NiFi or NiFi Registry section.

The **List of Policies** page appears.



- Click the ID for a policy.

The **Edit Policy** page appears.

**Edit Policy**

**Policy Details :**

Policy Type: **Access** ⓘ ⌵ Add Validity Period

Policy ID: **43**

Policy Name \*: Flow ⓘ enabled normal

Policy Label: Policy Label

NiFi Resource Identifier \*: /flow

Description:

Audit Logging: **YES**

**Allow Conditions :** hide

Select Role	Select Group	Select User	Permissions	Delegate Admin	
Select Roles	/ nifi	/ admin	Read	<input type="checkbox"/>	X

+

Deny All Other Accesses : **False**

**Save** **Cancel** **Delete**

- In the Allow Conditions section, add the user or the user group to the Select User field.
- Click Save.

## Results

The user now has the NiFi and NiFi Registry rights according to the policies you added the user or user group to. These rights are inherited down the hierarchy unless there is a more specific policy on a component.

## Creating a custom Ranger access policy

A user might need access to specific NiFi or NiFi Registry resources such as a process group or bucket. If the user cannot access the component through an inherited Ranger access policy, then you must create a custom Ranger access policy for the specific component and add the user to this policy. If all the users in a group require the same access, you can add the user group to the Ranger access policy.

## About this task

Each custom Ranger access policy provides access to a specific component.

First determine which NiFi or NiFi Registry components a user needs access to, then you can create a new policy for each component and add the user or user group to the new policy. When you create a new policy, you must specify the ID of the component that the user requires access to.

**Note:**

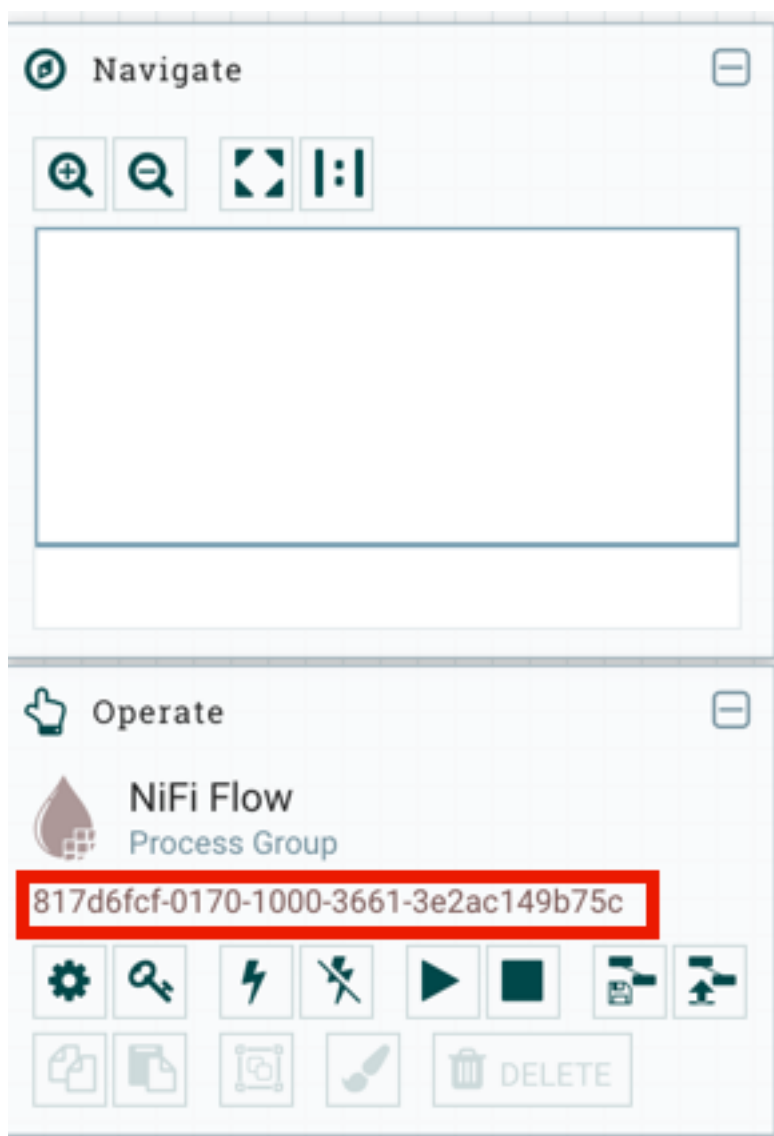
If a user requires permission to view or modify data for a specific component, you must create a custom data access policy and add the user and the nifi group to that policy.

The nifi group is a dynamically-managed group that exists on all Flow Management hosts and contains the identities of NiFi and Knox nodes. When you add the nifi group to the data policy for a specific component, you authorize the nodes to access data on behalf of the user.

**Procedure**

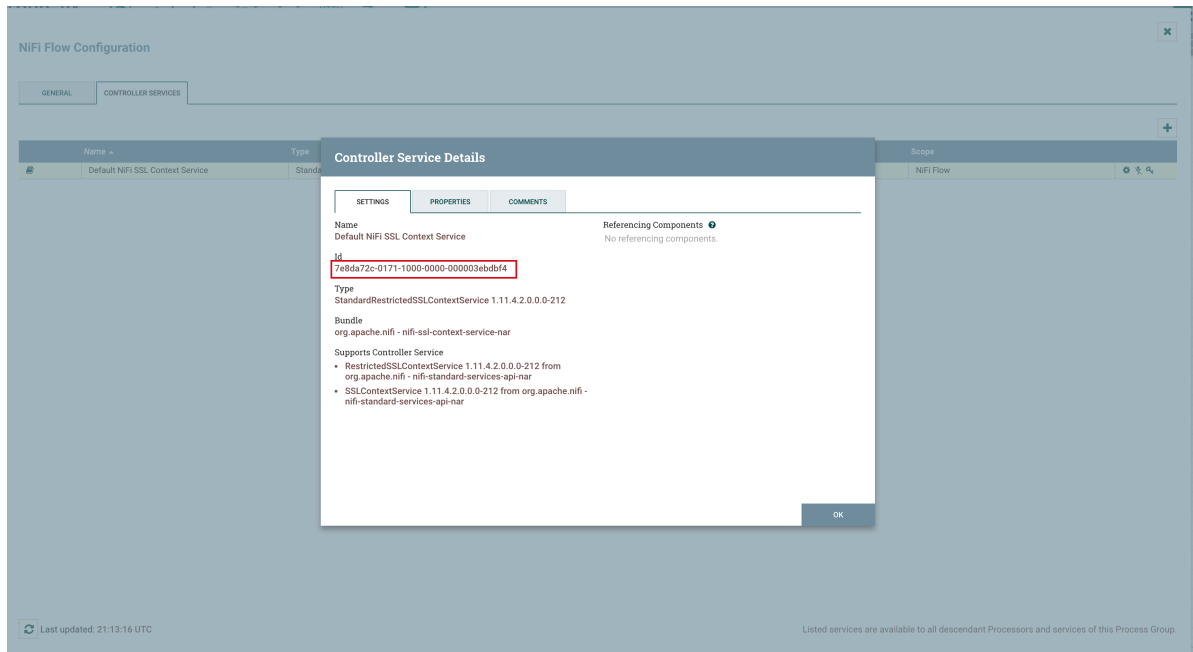
1. From the NiFi canvas, copy the ID of the process group, SSL Context Service, or controller service for reporting tasks that the user needs access to.
2. To locate the ID for a process group:
  - a) Click the process group.

The ID appears in the **Operate** pane.

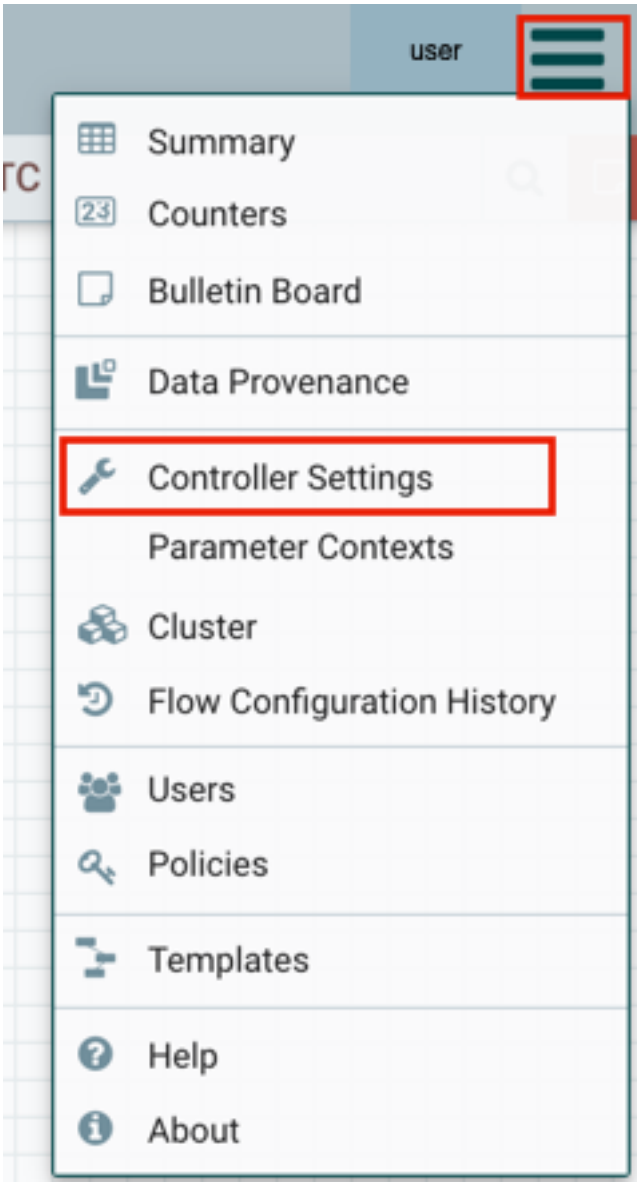


- b) Copy the ID.

3. To locate the ID of the SSL Context Service:
  - a) Click the settings icon on the process group.  
The **NiFi Flow Configuration** appears.
  - b) Click the **Controller Services** tab.
  - c) Click the **Settings** icon for the Default NiFi SSL Context Service.  
The **Controller Service Details** window appears.
  - d) From the **Settings** tab, copy the ID from the Id field.

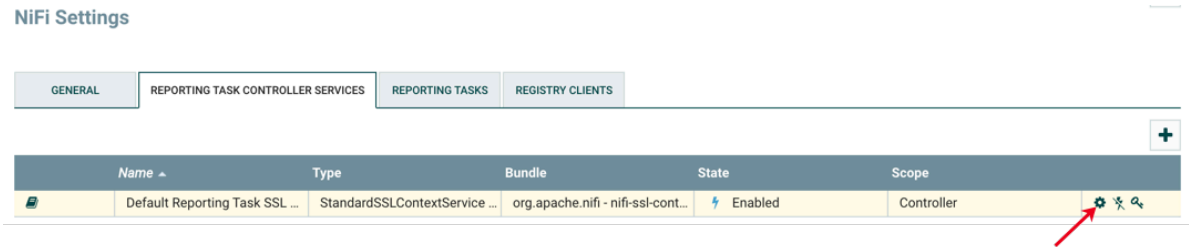


- 4. To locate the ID of a controller service for reporting tasks:
  - a) Click the process group.
  - b) Click the menu on the top right of the UI and select Controller Settings.



The **NiFi Settings** page appears.

- c) Click the **Reporting Tasks Controller Services** tab.
- d) Click the Settings icon for the controller service.



The **Controller Service Details** page appears.

- e) From the **Settings** tab, copy the ID from the Id field.

## Controller Service Details

SETTINGS

PROPERTIES

COMMENTS

Name

Default Reporting Task SSL Context Service

Referencing Components ?

No referencing components.

Id

05ad168c-0171-1000-ffff-ffffe39561d8

Type

StandardSSLContextService 1.11.3.2.0.0-195

Bundle

org.apache.nifi - nifi-ssl-context-service-nar

Supports Controller Service

- SSLContextService 1.11.3.2.0.0-195 from org.apache.nifi - nifi-standard-services-api-nar

OK

5. Go back to the **Ranger List of Policies** page.

6. Click Add New Policy.

The screenshot shows the Ranger Access Manager interface. The top navigation bar includes 'Ranger', 'Access Manager', 'Audit', 'Security Zone', and 'Settings'. Below this, the 'Service Manager' tab is active, showing 'docs\_flowm\_nifi Policies'. The main section is titled 'List of Policies : docs\_flowm\_nifi'. A search bar is present with the text 'Search for your policy...'. To the right of the search bar, the 'Add New Policy' button is highlighted with a red box. Below the search bar is a table listing various policies.

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
52	all - nifi-resource	--	Enabled	Enabled	--	_c_ranger_admins_a44480d	rangerlookup	[Eye] [Edit] [Delete]
53	Restricted Components	--	Enabled	Enabled	--	_c_nifi_admins_a44480d	--	[Eye] [Edit] [Delete]
54	Tenants	--	Enabled	Enabled	--	_c_nifi_admins_a44480d	--	[Eye] [Edit] [Delete]
55	Controller	--	Enabled	Enabled	--	_c_nifi_admins_a44480d	--	[Eye] [Edit] [Delete]
56	Flow	--	Enabled	Enabled	--	_c_nifi_admins_a44480d	--	[Eye] [Edit] [Delete]
57	Policies	--	Enabled	Enabled	--	_c_nifi_admins_a44480d	--	[Eye] [Edit] [Delete]
58	Proxies	--	Enabled	Enabled	--	nifi	--	[Eye] [Edit] [Delete]
66	Root Process Group	--	Enabled	Enabled	--	_c_nifi_admins_a44480d	--	[Eye] [Edit] [Delete]
67	Root Group Data	--	Enabled	Enabled	--	nifi _c_nifi_admins_a44480d	--	[Eye] [Edit] [Delete]

The **Create Policy** page appears.

7. Enter a unique name for the policy.

8. Optionally, enter a keyword in the Policy Label field to aid in searching for a policy.

9. Enter the resource descriptor and the resource ID in the NiFi Resource Identifier or NiFi Registry Resource Identifier field in the following format: <resource descriptor>/<resource ID>

To determine a NiFi resource descriptor, see *Pre-defined Ranger access policies for Apache NiFi*.

To determine a NiFi Registry resource descriptor, see *Pre-defined Ranger access policies for Apache NiFi Registry*.

10. Enter a description.

**11. Add a user or a group.**

**Note:** If a user requires permission to view or modify the data for a specific component, you must create a data policy with `/data/<component-type>/<component-UUID>` as the resource identifier. Then add the user and the nifi group to the policy to authorize the NiFi and Knox nodes to access data on behalf of the user.

**12. Set the permission level for the user or group.****13. Click Add.****Results**

The user or group of users can now access the component specified in the custom policy.

**Authorization example**

You can review this example to understand how you can enable a flow-management user to perform specific tasks like setting up version control for a flow, by assigning the appropriate Ranger policies.

User A must be able to do the following tasks:

- Access the NiFi UI.
- Export a flow.
- View data queued in connections.
- View data flowing through.
- Use a NiFi SSLContextService to connect to SSL-enabled systems.
- Set up version control for a flow.

Complete the following steps to enable User A to perform the required tasks:

**1. Add User A to the predefined Ranger access policy for NiFi, Flow. Set the permissions to Read.**

The Flow policy gives the user the right to view the NiFi UI.

**2. Create a Ranger access policy for NiFi with:**

- Resource descriptor: `/data/process-groups/<ID of process-group>`
- Permission: Read and Write

Add User A to this custom policy. The policy gives the user the right to export the data, view the data that is queued and flowing through the connections.

**3. Create a Ranger access policy for NiFi with:**

- Resource descriptor: `/controller-service/<ID of SSL Context Service>`
- Permission: Read

Add User A to this custom policy. The policy gives the user the right to use the specified SSLContextService in their flows to connect to SSL-enabled systems.

**4. Create a Ranger access policy for NiFi Registry with:**

- Resource descriptor: `/buckets/<ID of bucket>`
- Permission: Read, Write, and Delete

Add User A to this custom policy. The policy gives the user the right to set up version control for a flow.

**Enabling access to Knox and NiFi**

When NiFi is set up behind Knox, you need to define a Ranger policy that allows users to access NiFi through Knox.

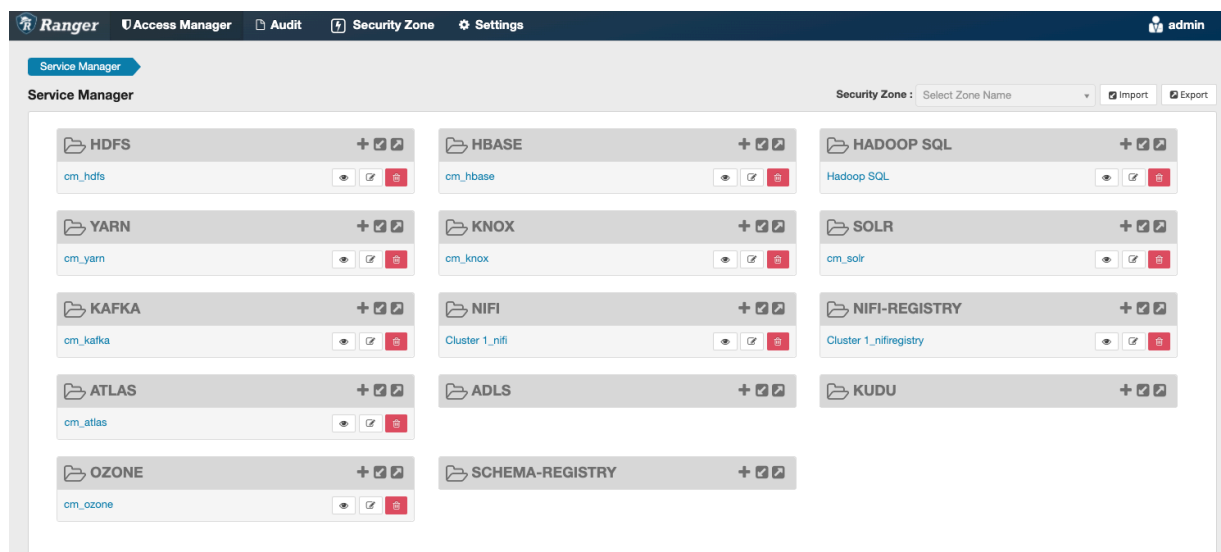
**Before you begin**

You are a system administrator.

## Procedure

1. From the Base cluster, select Ranger from the list of services. Click Ranger Admin Web UI and log into Ranger.

The Ranger **Service Manager** page displays.



Each cluster in the environment is listed under its respective service.

2. From the Knox folder, select the cluster that you need to provide access to.

The **List of Policies** page appears.

3. Click Add New Policy.

The **Create Policy** page appears.

4. Enter a unique name for the policy.
5. Enter a keyword in the Policy Label field to aid in searching for a policy.
6. Enter the Knox topology in the Knox Topology field.
7. Enter nifi in the Knox Service field.
8. Optionally, enter a description.
9. Add a user or a group.



**Note:** If a user requires permission to view or modify the data for a specific component, you must create a data policy with `/data/<component-type>/<component-UUID>` as the resource identifier. Then add the user and the nifi group to the policy to authorize the NiFi and Knox nodes to access data on behalf of the user.

## 10. Set the Permissions field to Allow.

The screenshot shows the Ranger Access Manager interface for creating a new policy. The policy is named 'NiFi Policy' and is of type 'Access'. It is currently 'Enabled'. The Knox Topology is set to 'default' and the Knox Service is set to 'nifi'. The permissions are set to 'Allow'. The audit logging is set to 'Yes'. The policy conditions are currently empty. At the bottom, there is a table with columns: Select Role, Select Group, Select User, Policy Conditions, Permissions, Delegate Admin, and a red 'X' button. The 'Permissions' column shows 'Allow'.

## 11. Click Add to save the new policy.

### Results

The user or group of users can now access NiFi through Knox based on what you defined in the Knox Topology field.

## File-based authorization

When Ranger is not selected as a dependency during installation, NiFi or NiFi Registry's internal file-based authorizer will be used for authorization.

When Ranger is not selected, the NiFi and NiFi Registry CSD scripts will perform the following steps:

- By default, during start-up, NiFi and NiFi Registry will create the following files in `/var/lib/nifi` and `/var/lib/nifi/registry`:
  - `users.xml`
  - `authorizations.xml`

These files will include the users and policies for the Initial Admin Identity, Initial Admin Groups, and proxy group.

- Create policies for the following Initial Admin Identity and Initial Admin Groups:
  - For NiFi: `nifi.initial.admin.identity` and `nifi.initial.admin.groups`
  - For NiFi Registry: `nifi.registry.initial.admin.identity` and `nifi.registry.initial.admin.groups`
- Create policies for proxies specified by `nifi.proxy.group` or `nifi.registry.proxy.group`.

Each authorizers.xml file produced in NiFi and NiFi Registry when using file-based authorization contains the following logical configuration:

- CompositeConfigurableUserGroupProvider
  - FileUserGroupProvider
  - CMUserGroupProvider
- FileAccessPolicyProvider
  - Configured with the CompositeConfigurableUserGroupProvider
- StandardManagedAuthorizer
  - Configured with FileAccessPolicyProvider

## Migrating file-based authorization to Ranger

Both NiFi and NiFi Registry services have the option to convert existing file-based provider policies to Ranger provider policies.

### Migrating NiFi file-based authorization to Ranger

You can convert existing file-based provider NiFi policies to Ranger provider policies.

#### Before you begin

The following steps assume that the Ranger service is installed in the Cloudera Base on premises cluster.

#### Procedure

1. Create any users and groups from the NiFi users.xml that do not already exist in Ranger.
2. Select Ranger as a dependency from NiFi configuration.
3. Restart NiFi.
4. Select Migrate File-based Authorizations to Ranger from the Actions drop-down and confirm the action.
5. After a successful migration, verify that the policies are available in the NiFi Ranger service.

### Migrating NiFi Registry file-based authorization to Ranger

You can convert existing file-based provider NiFi Registry policies to Ranger provider policies.

#### Before you begin

The following steps assume that the Ranger service is installed in the Cloudera Base on premises cluster.

#### Procedure

1. Create any users and groups from the NiFi Registry users.xml that do not already exist in Ranger.
2. Select Ranger as a dependency from NiFi Registry configuration.
3. Restart NiFi Registry.
4. Select Migrate File-based Authorizations to Ranger from the Actions drop-down. Confirm the action.
5. After a successful migration, verify that the policies are available in the NiFi Registry Ranger service.

## Environment variables

This section provides information on environment variables.

### Kerberos credentials

Learn how to provide the Kerberos credentials by defining the Keytab Credentials Controller Service.

In most processors there are two ways to provide the Kerberos credentials: either via properties directly available in processor's configuration (this is the legacy way) or via the definition of a Keytab Credentials Controller Service. The controller service is the recommended way in multi tenant environments where access to keytab configuration should be managed independently between different teams.

An environment variable is available to manage which option is used. In order to prevent the use of the old free-form keytab properties that were left around for backwards compatibility, it is possible to configure an environment variable in nifi-env.sh:

```
export NIFI_ALLOW_EXPLICIT_KEYTAB=true
```

Setting this value to false will produce a validation error in any component where the free-form keytab property is entered, which means the component cannot be started unless it uses a Keytab Controller service.

This environment variable set to false in combination with the `/restricted-components/access-keytab` policy is the recommended way to have the finest grained control over keytabs.

## Local file system access

Learn how to prevent access to the local file system.

The Hadoop processors such as the HDFS and Hive processors, (processors where some core-site, hdfs-site, XML configuration files are required) could theoretically be used, with very specifically tailored configurations, to access the local file system where NiFi is running.

In order to prevent access to the local file system, set the following environment variable in the `nifi-env.sh` file:

```
export NIFI_HDFS_DENY_LOCAL_FILE_SYSTEM_ACCESS=false
```

By default, this variable is set to false. Setting this value to true forces the Hadoop processors to evaluate the file system being accessed during scheduling and deny access in case it tries to access the local file system.

## Network

This section provides network information for NiFi and NiFi Registry.

### Default ports for NiFi and NiFi Registry

Reference for the NiFi and NiFi Registry default ports.

#### NiFi

The following table lists the default ports used by NiFi and the corresponding property in the `nifi.properties` file. You can change these values as required.



#### Note:

- The default values are set by Cloudera Manager.
- If you install NiFi-only binaries not managed by Cloudera Manager, the defaults will be different and can be found in the `nifi.properties` file.

Function	Property	Default Value set by Cloudera Manager
HTTP Port	<code>nifi.web.http.port</code>	8080
HTTPS Port	<code>nifi.web.https.port</code>	8443
Remote Input Socket Port	<code>nifi.remote.input.socket.port</code>	none
Cluster Node Protocol Port	<code>nifi.cluster.node.protocol.port</code>	9088
Cluster Node Load Balancing Port	<code>nifi.cluster.node.load.balance.port</code>	6342
Web HTTP Forwarding Port	<code>nifi.web.http.port.forwarding</code>	none

#### NiFi Registry

The following table lists the default ports used by NiFi Registry and the corresponding property in the `nifi-registry.properties` file. You can change these values as required.

**Note:**

If you install NiFi-only binaries not managed by Cloudera Manager, then:

- When enabling HTTPS, unset the `nifi.registry.web.http.port` property.
- The default values will be different and can be found in the `nifi-registry.properties` file.

Function	Property	Default Value set by Cloudera Manager
HTTP Port	<code>nifi.registry.web.http.port</code>	18080
HTTPS Port	<code>nifi.registry.web.https.port</code>	18433

## FIPS 140-2 compliance

Federal Information Processing Standards (FIPS) are publicly announced standards developed by the National Institute of Standards and Technology for use in computer systems by non-military American government agencies and government contractors. You can configure Cloudera Base on premises to use FIPS-compliant cryptography.

To install and configure a Cloudera cluster that is FIPS-compliant, see *Installing and configuring Cloudera with FIPS*. In combination with AutoTLS, the cluster will use BouncyCastle FIPS Keystore (BCFKS) across all the components.

Note the following points about FIPS compliance in Cloudera Flow Management:

- Cloudera Flow Management is compatible with a FIPS 140-2 compliant environment.
- Cloudera Flow Management can run on an OS with FIPS turned on and can use FIPS-compliant crypto libraries.
- By default, the KeyStore and TrustStore are in Java KeyStore (JKS) format. This format is not FIPS compliant.
- By default, NiFi dataflows are not FIPS compliant. You must specifically design a dataflow to be FIPS compliant.
- You can encrypt NiFi sensitive properties, such as the password for a database connection pool service, with a secret key generated by the FIPS 140-2 approved PBKDF2 algorithm. For information on how to do this, see *Encrypting NiFi sensitive properties with FIPS 140-2 approved algorithm*.

For the National Institute of Standards and Technology publication, see *FIPS 140-2 Security Requirements for Cryptographic Modules*.

**Note:**

The Bouncy Castle TLS library `bctls-safelogic.jar` includes an implementation of the TLS protocol that takes precedence over the standard Java implementation when configuring the `BouncyCastleJsseProvider` as a provider in `java.security`. The default configuration of the BCTLS library does not enable GCM-based ciphers, which results in TLS server components attempting to negotiate weak cipher suites based on AES-CBC. Modern web browsers such as Google Chrome and Mozilla Firefox disable weak cipher suites, resulting in cipher mismatch errors when attempting to connect to a FIPS-enabled deployment of Cloudera Flow Management.

Setting the following Java System property enables support for GCM-based ciphers using the Bouncy Castle TLS library: `org.bouncycastle.jsse.fips.allowGCMCiphers=true`

This setting must be specified in the `bootstrap.conf` configuration using the following setting: `java.arg.all owgcm=-Dorg.bouncycastle.jsse.fips.allowGCMCiphers=true`

Enabling GCM-based ciphers allows clients to negotiate modern TLS cipher suites, avoiding connection issues related to weak algorithms.

### Related Information

[FIPS 140-2 Security Requirements for Cryptographic Modules](#)

[Installing and configuring Cloudera with FIPS](#)

## Encrypting NiFi sensitive properties with FIPS 140-2 approved algorithm

You can encrypt NiFi sensitive properties, such as the password for a database connection pool service, with a secret key generated by the FIPS 140-2 approved PBKDF2 algorithm.

### About this task

The PBKDF2 algorithm uses 160,000 hashing iterations with the SHA-512 digest function. The generated secret key is then used to encrypt properties with AES Galois/Counter Mode (GCM), which provides both encryption and integrity protection.

To generate secret keys using the PBKDF2 algorithm, you must specify the algorithm in the `nifi.sensitive.props.algorithm` field and specify a password in the `nifi.sensitive.props.key` field.

### Before you begin

See [Installing and configuring Cloudera with FIPS](#).

### Procedure

1. Open the `nifi.properties` file.
2. Set the `nifi.sensitive.props.algorithm` property to one of the following PBKDF2 options:
  - `NIFI_PBKDF2_AES_GCM_128` to specify a 128-bit key length
  - `NIFI_PBKDF2_AES_GCM_256` to specify a 256-bit key length
3. Set the `nifi.sensitive.props.key` property with a password that is at least 12 characters long.  
The encryption key is derived from this password.
4. Save the `nifi.properties` file.
5. If you are installing Cloudera Flow Management, start NiFi. If you are upgrading to a newer Cloudera Flow Management version, see the *Upgrade and migration paths* documentation.

### Related Information

[Upgrade and migration paths](#)

## Deploying Cloudera Flow Management on FIPS-enabled clusters

FIPS 140-2 compliance is mandatory for many government and regulated industry environments. Cloudera Flow Management is compatible with FIPS-enabled operating systems and cryptographic libraries but requires specific configuration changes to ensure successful deployment and runtime operation.

### About this task

This guide provides the required steps and configurations to successfully deploy Cloudera Flow Management on clusters running in FIPS 140-2 mode.

### Before you begin

- Cloudera Base on premises installed with FIPS mode enabled
- Java 11 (FIPS-compliant build)
- Cloudera Flow Management 2.x (for example: 2.1.7)
- AutoTLS configured (recommended)
- Access to required cryptographic JARs from SafeLogic/Bouncy Castle

## Procedure

1. Install the required cryptographic JARs by copying the following files into the Cloudera Flow Management parcel's toolkit directory.

```
cp -a /path/to/ccj/jars/bctls.jar \
    /path/to/ccj/jars/ccj-3.0.2.1.jar \
    /opt/cloudera/parcels/CFM-[*VERSION*]/TOOLKIT/lib/
```

Replace `[*VERSION*]` with the appropriate Cloudera Flow Management version string.

2. Configure GCM Cipher support and NiFi Bootstrap settings.

Modern web browsers like Chrome or Firefox reject weak TLS cipher suites. By default, Bouncy Castle's FIPS TLS library does not enable GCM ciphers, which are required for compatibility with secure browsers. This configuration enables modern GCM-based TLS ciphers through Bouncy Castle, avoiding connection issues caused by legacy AES-CBC suites, especially in FIPS environments.

To enable GCM cipher support and configure bootstrap settings:

- a) Navigate to Cloudera Manager NiFi Configuration Advanced NiFi Node Advanced Configuration Snippet (Safety Valve) for `staging/bootstrap.conf.xml`.
- b) Add the following properties in XML view.

```
<property>
  <name>java.arg.modulepath</name>
  <value>--module-path=/tmp/jars</value>
</property>
<property>
  <name>java.arg.allowgcm</name>
  <value>-Dorg.bouncycastle.jsse.fips.allowGCMCiphers=true</value>
</property>
<property>
  <name>java.arg.truststoretype</name>
  <value>-Djavax.net.ssl.trustStoreType=bcfks</value>
</property>
<property>
  <name>java.arg.truststorepath</name>
  <value>-Djavax.net.ssl.trustStore=/var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_truststore.jks</value>
</property>
<property>
  <name>java.arg.truststorepassword</name>
  <value>-Djavax.net.ssl.trustStorePassword=REPLACE_ME</value>
</property>
```

- c) If using AutoTLS Use Case 3, you can retrieve the truststore password.

```
sudo -u postgres psql
\c scm
SELECT * FROM CONFIGS WHERE attr LIKE 'truststore_password';
```

3. Encrypt NiFi sensitive properties with FIPS algorithm, by editing the `nifi.properties` file.

```
nifi.sensitive.props.algorithm=NIFI_PBKDF2_AES_GCM_256
nifi.sensitive.props.key=your_secure_password_here
```

- Use either `NIFI_PBKDF2_AES_GCM_128` or `256`.
- The password must be at least 12 characters.

4. Restart Cloudera SCM agent to start and validate the Cloudera Flow Management deployment.

```
sudo systemctl stop cloudera-scm-supervisord.service
```

```
sudo systemctl restart cloudera-scm-agent
```

5. Start the NiFi service through Cloudera Manager.
6. Confirm that NiFi stays running and that no cipher mismatch or keyStore errors are present in the logs.

## Integrations

This section provides information on integrating NiFi and NiFi Registry with other components.

### Integrating NiFi and Atlas

You can integrate NiFi with Apache Atlas to take advantage of robust dataset and application lineage support.

#### Manually integrating with Atlas when Auto-TLS is not enabled

If Cloudera Flow Management or the Cloudera Base on premises cluster does not have Auto-TLS enabled and you want to Atlas, then you must manually integrate with Atlas by creating the `ReportLineageToAtlas` reporting task.

##### About this task

Perform this task if:

- Cloudera Flow Management does not have TLS enabled; AND
- The Cloudera Base on premises cluster does not have auto-TLS enabled; AND
- You do not want to enable auto-TLS; AND
- You want Atlas as part of Cloudera Flow Management on your Cloudera Base on premises deployment.

##### Procedure

1. From the Global Menu located in NiFi's upper right corner, select Controller Services and click the Reporting Tasks tab.
2. Click the Add (+) icon to launch the Add Reporting Task dialog.
3. Select `ReportLineageToAtlas` and click Add.

4. Click the Edit icon to launch the Configure Reporting Task dialog. The following properties are required:

- Atlas URLs – a comma-separated list of Atlas Server URLs. Once you have started reporting, you cannot modify an existing Reporting Task to add a new Atlas Server. When you need to add a new Atlas Server, you must create a new reporting task.
- Atlas Configuration Directory - This specifies where the atlas-applications.properties is created.

The directory must:

- Be located and accessible/writable by the user running the NiFi process.
- Be available on each NiFi node.
- Pre-exist. It will not be created by the reporting task.
- Not be in the /tmp directory.
- Create Atlas Configuration File – Set to True. When set to True, the atlas-application-properties file and the Atlas Configuration Directory are automatically created when the Reporting Task starts.
- Lineage Strategy – Specifies the level of granularity for your NiFi dataflow reporting to Atlas. Once you have started reporting, you should not switch between simple and complete lineage reporting strategies.
- Provenance Record Start Position – Specifies where in the Provenance Events stream the Reporting Task should start.
- Provenance Record Batch Size – Specifies how many records you want to send in a single batch
- NiFi URL for Atlas – Specifies the NiFi cluster URL.
- Atlas Authentication Method – Specifies how to authenticate the Reporting Task to the Atlas Server. Basic authentication is the default.
- Kafka Security Protocol – Specifies the protocol used to communicate with Kafka brokers to send Atlas hook notification messages. This value should match Kafka's `security.protocol` property value.

## Manually integrating with Atlas when Auto-TLS is enabled

You must perform some manual steps to integrate with Atlas when auto-TLS is enabled on your Cloudera Base on premises cluster.

### About this task

You must perform these steps if:

- You want CFM to integrate with Atlas; AND
- The Cloudera Base on premises cluster has auto-TLS enabled

### Procedure

1. Select the Atlas integration checkbox.
2. Restart NiFi.
3. Click Create required NiFi object in the Cloudera Manager Actions menu.

## Integrating NiFi and NiFi Registry with Knox

Integrate NiFi and NiFi Registry with Knox to securely access NiFi and NiFi Registry nodes.

Apache Knox Gateway (Knox) provides the following benefits:

- Centralized access to all services in the cluster.
- Authentication with single sign-on.
- Service-level authorization to the cluster.
- Does not expose the service endpoints such as URLs, ports, IP addresses.

When you integrate NiFi and NiFi Registry with Knox, you can use the Knox URL as a single entry point to securely access all NiFi nodes and switch nodes if one fails.

For information more information on Knox, see *Apache Knox Overview*.

For information on how to select Knox during the NiFi and NiFi Registry installation, see *Cloudera Flow Management Deployment*.

### Related Information

[Apache Knox Overview](#)

[Cloudera Flow Management Deployment](#)

## Customizing properties in Cloudera Manager

You can customize NiFi and NiFi Registry beyond what the customization page in Cloudera Manager allows. To make any changes, use the dot notation to represent the actual schema for a given property file.

### About this task

The following steps show how to enhance or overwrite xml based properties in Cloudera Manager using dot notation.

### Procedure

Use the following structure:

```
xml.<properties-type>.<entity>.<identifier>.class
xml.<properties-type>.<entity>.<identifier>.property.<property-value>
```

Where:

- <properties-type> for NiFi can be `authorizers` and `loginIdentityProviders`
- <properties-type> for NiFi Registry can be `authorizers` and `identityProviders`.

The following property key/value example creates a user group provider entry into the `authorizers` file for NiFi:

```
Name: xml.authorizers.userGroupProvider.file-user-group-provider.class
Value: org.apache.nifi.authorization.FileUserGroupProvider

Name: xml.authorizers.userGroupProvider.file-user-group-provider.property
.Initial User Identity 2
Value: CN=localhost, OU=NIFI
```

This translates to the following entry in the generated `authorizers.xml` file:

```
<authorizers>
...
  <userGroupProvider>
    <identifier>file-user-group-provider</identifier>
    <class>org.apache.nifi.authorization.FileUserGroupProvider</class>
    <property name="Initial User Identity 2">CN=localhost, OU=NIFI</prop
erty>
  </userGroupProvider>
...
  ...
</authorizers>
```

Properties names that have spaces are supported and do not need to be escaped.

### Example

For an example, see *Pairing LDAP with a Composite Group Provider*.

**Related Information**[Pairing LDAP with a Composite Group Provider](#)