Hortonworks DataFlow

Apache NiFi Registry Administration

(June 6, 2018)

docs.hortonworks.com

Hortonworks DataFlow: Apache NiFi Registry Administration

Hortonworks DataFlow (HDF) is powered by Apache NiFi. A version of this documentation originally appeared on the Apache NiFi website.

HDF is the first integrated platform that solves the real time challenges of collecting and transporting data from a multitude of sources and provides interactive command and control of live flows with full and automated data provenance. HDF is a single combined platform that provides the data acquisition, simple event processing, transport and delivery mechanism designed to accommodate the diverse dataflows generated by a world of connected people, systems and things.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. Hortonworks DataFlow is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain free and open source.

Please visit the Hortonworks page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.

Table of Contents

1. Apache NiFi Registry System Administrator's Guide

1.1. Apache NiFi Registry System Administrator's Guide

- Requires Java 8, newer than 1.8.0_45
- Supported Operating Systems:
  - Linux
  - Unix
  - Mac OS X
- Supported Web Browsers:
  - Google Chrome: Current & (Current - 1)
  - Mozilla FireFox: Current & (Current - 1)
  - Safari: Current & (Current - 1)

1.1.1. How to install and start NiFi Registry

- Linux/Unix/OS X
  - Decompress and untar into desired installation directory
  - Make any desired edits in files found under <installdir>/conf
  - From the <installdir>/bin directory, execute the following commands by typing ./nifi-registry.sh <command>:
    - start: starts NiFi Registry in the background
    - stop: stops NiFi Registry that is running in the background
    - status: provides the current status of NiFi Registry
    - run: runs NiFi Registry in the foreground and waits for a Ctrl-C to initiate shutdown of NiFi Registry
    - install: installs NiFi Registry as a service that can then be controlled via
      - service nifi-registry start
      - service nifi-registry stop

- service nifi-registry status

When NiFi Registry first starts up, the following files and directories are created:

- flow_storage directory
- database directory
- work directory
- logs directory
- run directory

See the System Properties section of this guide for more information about NiFi Registry configuration files.

1.1.2. Security Configuration

NiFi Registry provides several different configuration options for security purposes. The most important properties are those under the "security properties" heading in the *nifi-registry.properties* file. In order to run securely, the following properties must be set:

Property Name	Description
`nifi.registry.security.keystore`	Filename of the Keystore that contains the server's private key.
`nifi.registry.security.keystoreType`	The type of Keystore. Must be either PKCS12 or JKS. JKS is the preferred type, PKCS12 files will be loaded with BouncyCastle provider.
`nifi.registry.security.keystorePasswd`	The password for the Keystore.
`nifi.registry.security.keyPasswd`	The password for the certificate in the Keystore. If not set, the value of `nifi.registry.security.keystorePasswd` will be used.
`nifi.registry.security.truststore`	Filename of the Truststore that will be used to authorize those connecting to NiFi Registry. A secured instance with no Truststore will refuse all incoming connections.
`nifi.registry.security.truststoreType`	The type of the Truststore. Must be either PKCS12 or JKS. JKS is the preferred type, PKCS12 files will be loaded with BouncyCastle provider.
`nifi.registry.security.truststorePasswd`	The password for the Truststore.
`nifi.registry.security.needClientAuth`	**This specifies that connecting clients must authenticate with a client cert. Setting this 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. The default value is true.**

Once the above properties have been configured, we can enable the User Interface to be accessed over HTTPS instead of HTTP. This is accomplished by setting the `nifi.registry.web.https.host` and `nifi.registry.web.https.port` properties. The `nifi.registry.web.https.host` property indicates which hostname the server should run on. If it is desired that the HTTPS interface be

accessible from all network interfaces, a value of `0.0.0.0` should be used for `nifi.registry.web.https.host`.

> It is important when enabling HTTPS that the `nifi.registry.web.http.port` property be unset.

1.1.3. User Authentication

A secured instance of NiFi Registry cannot be accessed anonymously, so a method of user authentication must be configured.

> NiFi Registry does not perform user authentication over HTTP. Using HTTP, all users will have full permissions.

Any secured instance of NiFi Registry supports authentication via client certificates that are trusted by the NiFi Registry's SSL Context Truststore. Alternatively, a secured NiFi Registry can be configured to authenticate users via username/password.

Username/password authentication is performed by an *Identity Provider*. The Identity Provider is a pluggable mechanism for authenticating users via their username/password. Which Identity Provider to use is configured in the *nifi-registry.properties* file. Currently NiFi Registry offers Identity Providers for LDAP and Kerberos.

Identity Providers are configured using two properties in the *nifi-registry.properties* file:

- The `nifi.registry.security.identity.providers.configuration.file` property specifies the configuration file where identity providers are defined. By default, the *identity-providers.xml* file located in the root installation conf directory is selected.
- The `nifi.registry.security.identity.provider` property indicates which of the configured identity providers in the *identity-providers.xml* file to use. By default, this property is not configured meaning that username/password must be explicitly enabled.

> NiFi Registry can only be configured to use one Identity Provider at a given time.

1.1.3.1. Lightweight Directory Access Protocol (LDAP)

Below is an example and description of configuring a Identity Provider that integrates with a Directory Server to authenticate users.

```
<provider>
    <identifier>ldap-identity-provider</identifier>
    <class>org.apache.nifi.registry.security.ldap.LdapIdentityProvider</class>
    <property name="Authentication Strategy">START_TLS</property>

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

    <property name="TLS - Keystore"></property>
    <property name="TLS - Keystore Password"></property>
    <property name="TLS - Keystore Type"></property>
```

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

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

    <property name="Url"></property>
    <property name="User Search Base"></property>
    <property name="User Search Filter"></property>

    <property name="Identity Strategy">USE_DN</property>
    <property name="Authentication Expiration">12 hours</property>
</provider>
```

With this configuration, username/password authentication can be enabled by referencing this provider in *nifi-registry.properties*.

```
nifi.registry.security.identity.provider=ldap-identity-provider
```

Property Name	Description
`Authentication Strategy`	How the connection to the LDAP server is authenticated. Possible values are ANONYMOUS, SIMPLE, LDAPS, or START_TLS.
`Manager DN`	The DN of the manager that is used to bind to the LDAP server to search for users.
`Manager Password`	The password of the manager that is used to bind to the LDAP server to search for users.
`TLS - Keystore`	Path to the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
`TLS - Keystore Password`	Password for the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
`TLS - Keystore Type`	Type of the Keystore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
`TLS - Truststore`	Path to the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
`TLS - Truststore Password`	Password for the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
`TLS - Truststore Type`	Type of the Truststore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
`TLS - Client Auth`	Client authentication policy when connecting to LDAP using LDAPS or START_TLS. Possible values are REQUIRED, WANT, NONE.
`TLS - Protocol`	Protocol to use when connecting to LDAP using LDAPS or START_TLS. (i.e. TLS, TLSv1.1, TLSv1.2, etc).
`TLS - Shutdown Gracefully`	Specifies whether the TLS should be shut down gracefully before the target context is closed. Defaults to false.
`Referral Strategy`	Strategy for handling referrals. Possible values are FOLLOW, IGNORE, THROW.
`Connect Timeout`	Duration of connect timeout. (i.e. 10 secs).
`Authentication Expiration`	**The duration of how long the user authentication is valid for. If the user never logs out, they will be required to log back in following this duration.**

Property Name	Description
`Read Timeout`	Duration of read timeout. (i.e. 10 secs).
`Url`	Space-separated list of URLs of the LDAP servers (i.e. ldap://<hostname>:<port>).
`User Search Base`	Base DN for searching for users (i.e. CN=Users,DC=example,DC=com).
`User Search Filter`	Filter for searching for users against the *User Search Base*. (i.e. sAMAccountName={0}). The user specified name is inserted into *{0}*.
`Identity Strategy`	Strategy to identify users. Possible values are USE_DN and USE_USERNAME. The default functionality if this property is missing is USE_DN in order to retain backward compatibility. USE_DN will use the full DN of the user entry if possible. USE_USERNAME will use the username the user logged in with.
`Authentication Expiration`	**The duration of how long the user authentication is valid for. If the user never logs out, they will be required to log back in following this duration.**

1.1.3.2. Kerberos

Below is an example and description of configuring an Identity Provider that integrates with a Kerberos Key Distribution Center (KDC) to authenticate users.

```
<provider>
    <identifier>kerberos-identity-provider</identifier>
    <class>org.apache.nifi.registry.web.security.authentication.kerberos.
KerberosIdentityProvider</class>
    <property name="Default Realm">NIFI.APACHE.ORG</property>
    <property name="Kerberos Config File">/etc/krb5.conf</property>
    <property name="Authentication Expiration">12 hours</property>
</provider>
```

With this configuration, username/password authentication can be enabled by referencing this provider in *nifi-registry.properties*.

```
nifi.registry.security.user.identity.provider=kerberos-identity-provider
```

Property Name	Description
`Default Realm`	Default realm to provide when user enters incomplete user principal (i.e. NIFI.APACHE.ORG).
`Kerberos Config File`	Absolute path to Kerberos client configuration file.
`Authentication Expiration`	**The duration for which the user authentication is valid. If the user never logs out, they will be required to log back in following this duration.**

See also Kerberos Service to allow single sign-on access via client Kerberos tickets.

1.1.4. Authorization

After you have configured NiFi Registry to run securely and with an authentication mechanism, you must configure who has access to the system and their level of access. This is done by defining policies that give users and groups permissions to perform a particular action. These policies are defined in an *authorizer*.

1.1.4.1. Authorizer Configuration

An *authorizer* manages known users and their access policies. Authorizers are configured using two properties in the *nifi-registry.properties* file:

- The `nifi.registry.security.authorizers.configuration.file` property specifies the configuration file where authorizers are defined. By default, the *authorizers.xml* file located in the root installation conf directory is selected.
- The `nifi.registry.security.authorizer` property indicates which of the configured authorizers in the *authorizers.xml* file to use.

1.1.4.2. Authorizers.xml Setup

The *authorizers.xml* file is used to define and configure available authorizers. The default authorizer is the StandardManagedAuthorizer. The managed authorizer is comprised of a UserGroupProvider and a AccessPolicyProvider. The users, group, and access policies will be loaded and optionally configured through these providers. The managed authorizer will make all access decisions based on these provided users, groups, and access policies.

During startup there is a check to ensure that there are no two users/groups with the same identity/name. This check is executed regardless of the configured implementation. This is necessary because this is how users/groups are identified and authorized during access decisions.

The default UserGroupProvider is the FileUserGroupProvider, however, you can develop additional UserGroupProviders as extensions. The FileUserGroupProvider has the following properties:

- Users File - The file where the FileUserGroupProvider stores users and groups. By default, *users.xml* in the *conf* directory is chosen.
- Initial User Identity - The identity of a user or system to seed an empty Users File. Multiple Initial User Identity properties can be specified, but the name of each property must be unique, for example: "Initial User Identity A", "Initial User Identity B", "Initial User Identity C" or "Initial User Identity 1", "Initial User Identity 2", "Initial User Identity 3"

> Initial User Identities are only created if the specified Users File is missing or empty during NiFi Registry startup. Changes to the configured Initial Users Identities will not take effect if the Users File is populated.

Another option for the UserGroupProvider is the LdapUserGroupProvider. By default, this option is commented out but can be configured in lieu of the FileUserGroupProvider. This will sync users and groups from a directory server and will present them in NiFi Registry UI in read only form. The LdapUserGroupProvider has the following properties:

- Authentication Strategy - How the connection to the LDAP server is authenticated. Possible values are ANONYMOUS, SIMPLE, LDAPS, or START_TLS
- Manager DN - The DN of the manager that is used to bind to the LDAP server to search for users.

- Manager Password - The password of the manager that is used to bind to the LDAP server to search for users.
- TLS - Keystore - Path to the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
- TLS - Keystore Password - Password for the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
- TLS - Keystore Type - Type of the Keystore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
- TLS - Truststore - Path to the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
- TLS - Truststore Password - Password for the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
- TLS - Truststore Type - Type of the Truststore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
- TLS - Client Auth - Client authentication policy when connecting to LDAP using LDAPS or START_TLS. Possible values are REQUIRED, WANT, NONE.
- TLS - Protocol - Protocol to use when connecting to LDAP using LDAPS or START_TLS. (i.e. TLS, TLSv1.1, TLSv1.2, etc).
- TLS - Shutdown Gracefully - Specifies whether the TLS should be shut down gracefully before the target context is closed. Defaults to false.
- Referral Strategy - Strategy for handling referrals. Possible values are FOLLOW, IGNORE, THROW.
- Connect Timeout - Duration of connect timeout. (i.e. 10 secs).
- Read Timeout - Duration of read timeout. (i.e. 10 secs).
- Url - Space-separated list of URLs of the LDAP servers (i.e. ldap://<hostname>:<port>).
- Page Size - Sets the page size when retrieving users and groups. If not specified, no paging is performed.
- Sync Interval - Duration of time between syncing users and groups. (i.e. 30 mins).
- User Search Base - Base DN for searching for users (i.e. ou=users,o=nifi). Required to search users.
- User Object Class - Object class for identifying users (i.e. person). Required if searching users.
- User Search Scope - Search scope for searching users (ONE_LEVEL, OBJECT, or SUBTREE). Required if searching users.

- User Search Filter - Filter for searching for users against the *User Search Base* (i.e. (memberof=cn=team1,ou=groups,o=nifi)). Optional.
- User Identity Attribute - Attribute to use to extract user identity (i.e. cn). Optional. If not set, the entire DN is used.
- User Group Name Attribute - Attribute to use to define group membership (i.e. memberof). Optional. If not set group membership will not be calculated through the users. Will rely on group membership being defined through Group Member Attribute if set.
- Group Search Base - Base DN for searching for groups (i.e. ou=groups,o=nifi). Required to search groups.
- Group Object Class - Object class for identifying groups (i.e. groupOfNames). Required if searching groups.
- Group Search Scope - Search scope for searching groups (ONE_LEVEL, OBJECT, or SUBTREE). Required if searching groups.
- Group Search Filter - Filter for searching for groups against the *Group Search Base*. Optional.
- Group Name Attribute - Attribute to use to extract group name (i.e. cn). Optional. If not set, the entire DN is used.
- Group Member Attribute - Group Member Attribute - Attribute to use to define group membership (i.e. member). Optional. If not set group membership will not be calculated through the groups. Will rely on group member being defined through User Group Name Attribute if set.

Another option for the UserGroupProvider are composite implementations. This means that multiple sources/implementations can be configured and composed. For instance, an admin can configure users/groups to be loaded from a file and a directory server. There are two composite implementations, one that supports multiple UserGroupProviders and one that supports multiple UserGroupProviders and a single configurable UserGroupProvider.

The CompositeUserGroupProvider will provide support for retrieving users and groups from multiple sources. The CompositeUserGroupProvider has the following properties:

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

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

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

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

After you have configured a UserGroupProvider, you must configure an AccessPolicyProvider that will control Access Policies for the identities in the UserGroupProvider. The default AccessPolicyProvider is the FileAccessPolicyProvider, however, you can develop additional AccessPolicyProvider as extensions. The FileAccessPolicyProvider has the following properties:

- User Group Provider - The identifier for an User Group Provider defined above that will be used to access users and groups for use in the managed access policies.
- Authorizations File - The file where the FileAccessPolicyProvider will store policies. By default, *authorizations.xml* in the *conf* directory is chosen.
- Initial Admin Identity - The identity of an initial admin user that will be granted access to the UI and given the ability to create additional users, groups, and policies. For example, a certificate DN, LDAP identity, or Kerberos principal.
- NiFi Identity - The identity of a NiFi instance/node that will be accessing this registry. Each NiFi Identity will be granted permission to proxy user requests, as well as read any bucket to perform synchronization status checks.

> The identities configured in the Initial Admin Identity and NiFi Identity properties must be available in the configured User Group Provider. Initial Admin Identity and NiFi Identity properties are only read by NiFi Registry when the Authorizations File is missing or empty on startup in order to seed the initial Authorizations File. Changes to the configured Initial Admin Identity and NiFi Identities will not take effect if the Authorizations File is populated.

The default Authorizer is the StandardManagedAuthorizer, however, you can develop additional Authorizers as extensions. The StandardManagedAuthorizer has the following properties:

- Access Policy Provider - The identifier for an Access Policy Provider defined above.

1.1.4.2.1. Initial Admin Identity (New NiFi Registry Instance)

If you are setting up a secured NiFi Registry instance for the first time, you must manually designate an "Initial Admin Identity" in the *authorizers.xml* file. This initial admin user is granted access to the UI and given the ability to create additional users, groups, and policies. The value of this property could be a certificate DN , LDAP identity (DN or username), or a Kerberos principal. If you are the NiFi Registry administrator, add yourself as the "Initial Admin Identity".

Here is an example LDAP entry using the name John Smith:

```
<authorizers>

    <userGroupProvider>
```

```
        <identifier>file-user-group-provider</identifier>
        <class>org.apache.nifi.registry.security.authorization.file.
FileUserGroupProvider</class>
        <property name="Users File">./conf/users.xml</property>
        <property name="Legacy Authorized Users File"></property>
        <property name="Initial User Identity 1">cn=John Smith,ou=people,dc=
example,dc=com</property>
    </userGroupProvider>

    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.registry.security.authorization.file.
FileAccessPolicyProvider</class>
        <property name="User Group Provider">file-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">cn=John Smith,ou=people,dc=
example,dc=com</property
        <property name="NiFi Identity 1"></property>
    </accessPolicyProvider>

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

Here is an example Kerberos entry using the name John Smith and realm `NIFI.APACHE.ORG`:

```
<authorizers>

    <userGroupProvider>
        <identifier>file-user-group-provider</identifier>
        <class>org.apache.nifi.registry.security.authorization.file.
FileUserGroupProvider</class>
        <property name="Users File">./conf/users.xml</property>
        <property name="Initial User Identity 1">johnsmith@NIFI.APACHE.ORG</
property>
    </userGroupProvider>

    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.registry.security.authorization.file.
FileAccessPolicyProvider</class>
        <property name="User Group Provider">file-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">johnsmith@NIFI.APACHE.ORG</
property>
        <property name="NiFi Identity 1"></property>
    </accessPolicyProvider>

    <authorizer>
        <identifier>managed-authorizer</identifier>
```

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

After you have edited and saved the *authorizers.xml* file, restart NiFi Registry. The *users.xml* and *authorizations.xml* files will be created, and the "Initial Admin Identity" user and administrative policies are added during start up. Once NiFi Registry starts, the "Initial Admin Identity" user is able to access the UI and begin managing users, groups, and policies.

If initial NiFi identities are not provided, they can be added through the UI at a later time by first creating a user for the given NiFi identity, and then giving that user Proxy permissions, and permission to Buckets/READ in order to read all buckets.

Here is an example loading users and groups from LDAP. Group membership will be driven through the member attribute of each group. Authorization will still use file based access policies.

Given the following LDAP entries exist:

```
dn: cn=User 1,ou=users,o=nifi
objectClass: organizationalPerson
objectClass: person
objectClass: inetOrgPerson
objectClass: top
cn: User 1
sn: User1
uid: user1

dn: cn=User 2,ou=users,o=nifi
objectClass: organizationalPerson
objectClass: person
objectClass: inetOrgPerson
objectClass: top
cn: User 2
sn: User2
uid: user2

dn: cn=users,ou=groups,o=nifi
objectClass: groupOfNames
objectClass: top
cn: users
member: cn=User 1,ou=users,o=nifi
member: cn=User 2,ou=users,o=nifi
```

An Authorizer using an LdapUserGroupProvider would be configured as:

```
<authorizers>
    <userGroupProvider>
        <identifier>ldap-user-group-provider</identifier>
        <class>org.apache.nifi.registry.security.ldap.tenants.
LdapUserGroupProvider</class>
        <property name="Authentication Strategy">ANONYMOUS</property>
```

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

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

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

        <property name="Url">ldap://localhost:10389</property>
        <property name="Page Size"></property>
        <property name="Sync Interval">30 mins</property>

        <property name="User Search Base">ou=users,o=nifi</property>
        <property name="User Object Class">person</property>
        <property name="User Search Scope">ONE_LEVEL</property>
        <property name="User Search Filter"></property>
        <property name="User Identity Attribute">cn</property>
        <property name="User Group Name Attribute"></property>

        <property name="Group Search Base">ou=groups,o=nifi</property>
        <property name="Group Object Class">groupOfNames</property>
        <property name="Group Search Scope">ONE_LEVEL</property>
        <property name="Group Search Filter"></property>
        <property name="Group Name Attribute">cn</property>
        <property name="Group Member Attribute">member</property>
    </userGroupProvider>

    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.registry.security.authorization.file.
FileAccessPolicyProvider</class>
        <property name="User Group Provider">ldap-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">User 1</property>
        <property name="NiFi Identity 1"></property>
    </accessPolicyProvider>

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

The *Initial Admin Identity* value would have loaded from the cn of the User 1 entry based on the *User Identity Attribute* value.

Here is an example composite implementation loading users and groups from LDAP and a local file. Group membership will be driven through the member attribute of each group. The users from LDAP will be read only while the users loaded from the file will be configurable in UI.

```
<authorizers>

    <userGroupProvider>
        <identifier>file-user-group-provider</identifier>
        <class>org.apache.nifi.registry.security.authorization.file.
FileUserGroupProvider</class>
        <property name="Users File">./conf/users.xml</property>
        <property name="Initial User Identity 1">cn=nifi-node1,ou=servers,dc=
example,dc=com</property>
        <property name="Initial User Identity 2">cn=nifi-node2,ou=servers,dc=
example,dc=com</property>
    </userGroupProvider>

    <userGroupProvider>
        <identifier>ldap-user-group-provider</identifier>
        <class>org.apache.nifi.registry.security.ldap.tenants.
LdapUserGroupProvider</class>
        <property name="Authentication Strategy">ANONYMOUS</property>

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

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

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

        <property name="Url">ldap://localhost:10389</property>
        <property name="Page Size"></property>
        <property name="Sync Interval">30 mins</property>

        <property name="User Search Base">ou=users,o=nifi</property>
        <property name="User Object Class">person</property>
        <property name="User Search Scope">ONE_LEVEL</property>
        <property name="User Search Filter"></property>
        <property name="User Identity Attribute">cn</property>
        <property name="User Group Name Attribute"></property>

        <property name="Group Search Base">ou=groups,o=nifi</property>
        <property name="Group Object Class">groupOfNames</property>
        <property name="Group Search Scope">ONE_LEVEL</property>
        <property name="Group Search Filter"></property>
        <property name="Group Name Attribute">cn</property>
        <property name="Group Member Attribute">member</property>
    </userGroupProvider>
```

```
    <userGroupProvider>
        <identifier>composite-user-group-provider</identifier>
        <class>org.apache.nifi.registry.security.authorization.
CompositeUserGroupProvider</class>
        <property name="User Group Provider 1">file-user-group-provider</
property>
        <property name="User Group Provider 2">ldap-user-group-provider</
property>
    </userGroupProvider>

    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.registry.security.authorization.file.
FileAccessPolicyProvider</class>
        <property name="User Group Provider">composite-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">User 1/property>
        <property name="NiFi Identity 1">cn=nifi-node1,ou=servers,dc=example,
dc=com</property>
        <property name="NiFi Identity 2">cn=nifi-node2,ou=servers,dc=example,
dc=com</property>
    </accessPolicyProvider>

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

In this example, the users and groups are loaded from LDAP but the servers are managed in a local file. The *Initial Admin Identity* value came from an attribute in a LDAP entry based on the *User Identity Attribute*. The *NiFi Identity* values are established in the local file using the *Initial User Identity* properties.

1.1.5. Encrypted Passwords in Configuration Files

In order to facilitate the secure setup of NiFi Registry, you can use the `encrypt-config` command line utility to encrypt raw configuration values that NiFi Registry decrypts in memory on startup. This extensible protection scheme transparently allows NiFi Registry to use raw values in operation, while protecting them at rest. In the future, hardware security modules (HSM) and external secure storage mechanisms will be integrated, but for now, an AES encryption provider is the default implementation.

If no administrator action is taken, the configuration values remain unencrypted.

> The `encrypt-config` tool for NiFi Registry is implemented as an additional mode to the existing tool in the `nifi-toolkit`. The following sections assume you have downloaded the binary for the nifi-toolkit.

1.1.5.1. Encrypt-Config Tool

The `encrypt-config` command line tool can be used to encrypt NiFi Registry configuration by invoking the tool with the following command:

```
./bin/encrypt-config nifi-registry [options]
```

- `-h,--help` Show usage information (this message)
- `-v,--verbose` Enables verbose mode (off by default)
- `-p,--password <password>` Protect the files using a password-derived key. If an argument is not provided to this flag, interactive mode will be triggered to prompt the user to enter the password.
- `-k,--key <keyhex>` Protect the files using a raw hexadecimal key. If an argument is not provided to this flag, interactive mode will be triggered to prompt the user to enter the key.
- `--oldPassword <password>` If the input files are already protected using a password-derived key, this specifies the old password so that the files can be unprotected before re-protecting.
- `--oldKey <keyhex>` If the input files are already protected using a key, this specifies the raw hexadecimal key so that the files can be unprotected before re-protecting.
- `-b,--bootstrapConf <file>` The bootstrap.conf file containing no master key or an existing master key. If a new password/key is specified and no output bootstrap.conf file is specified, then this file will be overwritten to persist the new master key.
- `-B,--outputBootstrapConf <file>` The destination bootstrap.conf file to persist master key. If specified, the input bootstrap.conf will not be modified.
- `-r,--nifiRegistryProperties <file>` The nifi-registry.properties file containing unprotected config values, overwritten if no output file specified.
- `-R,--outputNifiRegistryProperties <file>` The destination nifi-registry.properties file containing protected config values.
- `-a,--authorizersXml <file>` The authorizers.xml file containing unprotected config values, overwritten if no output file specified.
- `-A,--outputAuthorizersXml <file>` The destination authorizers.xml file containing protected config values.
- `-i,--identityProvidersXml <file>` The identity-providers.xml file containing unprotected config values, overwritten if no output file specified.
- `-I,--outputIdentityProvidersXml <file>` The destination identity-providers.xml file containing protected config values.

As an example of how the tool works, assuming that you have installed the tool on a machine supporting 256-bit encryption and with the following existing values in the *nifi-registry.properties* file:

```
# security properties #
nifi.registry.security.keystore=/path/to/keystore.jks
nifi.registry.security.keystoreType=JKS
nifi.registry.security.keystorePasswd=thisIsABadKeystorePassword
nifi.registry.security.keyPasswd=thisIsABadKeyPassword
nifi.registry.security.truststore=
nifi.registry.security.truststoreType=
nifi.registry.security.truststorePasswd=
```

Enter the following arguments when using the tool:

```
./bin/encrypt-config.sh nifi-registry \
-b bootstrap.conf \
-k 0123456789ABCDEFFEDCBA98765432100123456789ABCDEFFEDCBA9876543210 \
-r nifi-registry.properties
```

As a result, the *nifi-registry.properties* file is overwritten with protected properties and sibling encryption identifiers (`aes/gcm/256`, the currently supported algorithm):

```
# security properties #
nifi.registry.security.keystore=/path/to/keystore.jks
nifi.registry.security.keystoreType=JKS
nifi.registry.security.keystorePasswd=oBjT92hIGRElIGOh||MZ6uYuWNBrOA6usq/
Jt3DaD2e4otNirZDytac/w/KFe0HOkrJR03vcbo
nifi.registry.security.keystorePasswd.protected=aes/gcm/256
nifi.registry.security.keyPasswd=ac/BaE35SL/esLiJ||
+ULRvRLYdIDA2VqpE0eQXDEMjaLBMG2kbKOdOwBk/hGebDKlVg==
nifi.registry.security.keyPasswd.protected=aes/gcm/256
nifi.registry.security.truststore=
nifi.registry.security.truststoreType=
nifi.registry.security.truststorePasswd=
```

When applied to *identity-providers.xml* or *authorizers.xml*, the property elements are updated with an `encryption` attribute. For example:

```
<!-- LDAP Provider -->
<provider>
   <identifier>ldap-provider</identifier>
   <class>org.apache.nifi.registry.security.ldap.LdapProvider</class>
   <property name="Authentication Strategy">START_TLS</property>
   <property name="Manager DN">someuser</property>
   <property name="Manager Password" encryption="aes/gcm/
128">q4r7WIgN0MaxdAKM||SGgdCTPGSFEcuH4RraMYEdeyVbOx93abdWTVSWvh1w+klA</
property>
   <property name="TLS - Keystore">/path/to/keystore.jks</property>
   <property name="TLS - Keystore Password" encryption="aes/gcm/128">Uah59TWX
+Ru5GY5p||B44RT/LJtC08QWA5ehQf01JxIpf0qSJUzug25UwkF5a50g</property>
   <property name="TLS - Keystore Type">JKS</property>
   ...
</provider>
```

Additionally, the *bootstrap.conf* file is updated with the encryption key as follows:

```
# Master key in hexadecimal format for encrypted sensitive configuration
 values
nifi.registry.bootstrap.sensitive.key=
0123456789ABCDEFFEDCBA98765432100123456789ABCDEFFEDCBA9876543210
```

Sensitive configuration values are encrypted by the tool by default, however you can encrypt any additional properties, if desired. To encrypt

additional properties, specify them as comma-separated values in the `nifi.registry.sensitive.props.additional.keys` property.

If the *nifi-registry.properties* file already has valid protected values and you wish to protect additional values using the same master key already present in your *bootstrap.conf*, then run the tool without specifying a new key:

```
# bootstrap.conf already contains master key property
# nifi-registy.properties has been updated for nifi.registry.sensitive.props.
additional.keys=...

./bin/encrypt-config.sh nifi-registry -b bootstrap.conf -r nifi-registry.
properties
```

1.1.5.2. Sensitive Property Key Migration

In order to change the key used to encrypt the sensitive values, provide the new key or password using the `-k` or `-p` flags as usual, and provide the existing key or password using `--old-key` or `--old-password` respectively. This will allow the toolkit to decrypt the existing values and re-encrypt them, and update *bootstrap.conf* with the new key. Only one of the key or password needs to be specified for each phase (old vs. new), and any combination is sufficient:

- old key # new key
- old key # new password
- old password # new key
- old password # new password

1.1.6. Bootstrap Properties

The *bootstrap.conf* file in the *conf* directory allows users to configure settings for how NiFi Registry should be started. This includes parameters, such as the size of the Java Heap, what Java command to run, and Java System Properties.

Here, we will address the different properties that are made available in the file. Any changes to this file will take effect only after NiFi Registry has been stopped and restarted.

Property	Description
java	Specifies the fully qualified java command to run. By default, it is simply `java` but could be changed to an absolute path or a reference an environment variable, such as `$JAVA_HOME/bin/java`
run.as	The username to run NiFi Registry as. For instance, if NiFi Registry should be run as the *nifi_registry* user, setting this value to *nifi_registry* will cause the NiFi Registry Process to be run as the *nifi_registry* user. This property is ignored on Windows. For Linux, the specified user may require sudo permissions.
lib.dir	The *lib* directory to use for NiFi Registry. By default, this is set to `./lib`
conf.dir	The *conf* directory to use for NiFi Registry. By default, this is set to `./conf`
graceful.shutdown.seconds	When NiFi Registry is instructed to shutdown, the Bootstrap will wait this number of seconds for the process

	to shutdown cleanly. At this amount of time, if the service is still running, the Bootstrap will "kill" the process, or terminate it abruptly. By default, this is set to `20`.
java.arg.N	Any number of JVM arguments can be passed to the NiFi Registry JVM when the process is started. These arguments are defined by adding properties to *bootstrap.conf* that begin with `java.arg.`. The rest of the property name is not relevant, other than to different property names, and will be ignored. The default includes properties for minimum and maximum Java Heap size, the garbage collector to use, etc.

1.1.7. Proxy Configuration

When running Apache NiFi Registry behind a proxy there are a couple of key items to be aware of during deployment.

- NiFi Registry is comprised of a number of web applications (web UI, web API, documentation), so the mapping needs to be configured for the **root path**. That way all context paths are passed through accordingly.
- If NiFi Registry is running securely, any proxy needs to be authorized to proxy user requests. These can be configured in the NiFi Registry UI through the Users administration section, by selecting *Proxy* for the given user. Once these permissions are in place, proxies can begin proxying user requests. The end user identity must be relayed in a HTTP header. For example, if the end user sent a request to the proxy, the proxy must authenticate the user. Following this the proxy can send the request to NiFi Registry. In this request an HTTP header should be added as follows.

```
X-ProxiedEntitiesChain: <end-user-identity>
```

If the proxy is configured to send to another proxy, the request to NiFi Registry from the second proxy should contain a header as follows.

```
X-ProxiedEntitiesChain: <end-user-identity><proxy-1-identity>
```

An example Apache proxy configuration that sets the required properties may look like the following. Complete proxy configuration is outside of the scope of this document. Please refer the documentation of the proxy for guidance for your deployment environment and use case.

```
...
<Location "/my-nifi">
    ...
        SSLEngine On
        SSLCertificateFile /path/to/proxy/certificate.crt
        SSLCertificateKeyFile /path/to/proxy/key.key
        SSLCACertificateFile /path/to/ca/certificate.crt
        SSLVerifyClient require
        RequestHeader add X-ProxyScheme "https"
        RequestHeader add X-ProxyHost "proxy-host"
        RequestHeader add X-ProxyPort "443"
        RequestHeader add X-ProxyContextPath "/my-nifi-registry"
        RequestHeader add X-ProxiedEntitiesChain "<%{SSL_CLIENT_S_DN}>"
        ProxyPass https://nifi-registry-host:8443
        ProxyPassReverse https://nifi-registry-host:8443
        ...
</Location>
```

```
...
```

1.1.8. Kerberos Service

NiFi Registry can be configured to use Kerberos SPNEGO (or "Kerberos Service") for authentication. In this scenario, users will hit the REST endpoint `/access/token/kerberos` and the server will respond with a `401` status code and the challenge response header `WWW-Authenticate: Negotiate`. This communicates to the browser to use the GSS-API and load the user's Kerberos ticket and provide it as a Base64-encoded header value in the subsequent request. It will be of the form `Authorization: Negotiate YII...`. NiFi Registry will attempt to validate this ticket with the KDC. If it is successful, the user's *principal* will be returned as the identity, and the flow will follow login/credential authentication, in that a JWT will be issued in the response to prevent the unnecessary overhead of Kerberos authentication on every subsequent request. If the ticket cannot be validated, it will return with the appropriate error response code. The user will then be able to provide their Kerberos credentials to the login form if the `KerberosIdentityProvider` has been configured. See Kerberos identity provider for more details.

NiFi Registry will only respond to Kerberos SPNEGO negotiation over an HTTPS connection, as unsecured requests are never authenticated.

See Kerberos Properties for complete documentation.

1.1.8.1. Notes

- Kerberos is case-sensitive in many places and the error messages (or lack thereof) may not be sufficiently explanatory. Check the case sensitivity of the service principal in your configuration files. The convention is `HTTP/fully.qualified.domain@REALM`.
- Browsers have varying levels of restriction when dealing with SPNEGO negotiations. Some will provide the local Kerberos ticket to any domain that requests it, while others whitelist the trusted domains. See Spring Security Kerberos - Reference Documentation: Appendix E. Configure browsers for SPNEGO Negotiation for common browsers.
- Some browsers (legacy IE) do not support recent encryption algorithms such as AES, and are restricted to legacy algorithms (DES). This should be noted when generating keytabs.
- The KDC must be configured and a service principal defined for NiFi and a keytab exported. Comprehensive instructions for Kerberos server configuration and administration are beyond the scope of this document (see MIT Kerberos Admin Guide), but an example is below:
- Kerberos tickets may use AES encryption with keys up to 256-bits in length, and therefore unlimited strength encryption policies may be required for the Java Runtime Environment (JRE) used for NiFi Registry when Kerberos SPNEGO is configured.

Adding a service principal for a server at `nifi.nifi.apache.org` and exporting the keytab from the KDC:

```
root@kdc:/etc/krb5kdc# kadmin.local
Authenticating as principal admin/admin@NIFI.APACHE.ORG with password.
kadmin.local:  listprincs
K/M@NIFI.APACHE.ORG
```

```
admin/admin@NIFI.APACHE.ORG
...
kadmin.local:  addprinc -randkey HTTP/nifi.nifi.apache.org
WARNING: no policy specified for HTTP/nifi.nifi.apache.org@NIFI.APACHE.ORG;
 defaulting to no policy
Principal "HTTP/nifi.nifi.apache.org@NIFI.APACHE.ORG" created.
kadmin.local:  ktadd -k /http-nifi.keytab HTTP/nifi.nifi.apache.org
Entry for principal HTTP/nifi.nifi.apache.org with kvno 2, encryption type
 des3-cbc-sha1 added to keytab WRFILE:/http-nifi.keytab.
Entry for principal HTTP/nifi.nifi.apache.org with kvno 2, encryption type
 des-cbc-crc added to keytab WRFILE:/http-nifi.keytab.
kadmin.local:  listprincs
HTTP/nifi.nifi.apache.org@NIFI.APACHE.ORG
K/M@NIFI.APACHE.ORG
admin/admin@NIFI.APACHE.ORG
...
kadmin.local: q
root@kdc:~# ll /http*
-rw------- 1 root root 162 Mar 14 21:43 /http-nifi.keytab
root@kdc:~#
```

1.1.9. System Properties

The *nifi-registry.properties* file in the *conf* directory is the main configuration file for controlling how NiFi Registry runs. This section provides an overview of the properties in this file and includes some notes on how to configure it in a way that will make upgrading easier. **After making changes to this file, restart NiFi Registry in order for the changes to take effect.**

> Values for periods of time and data sizes must include the unit of measure, for example "10 secs" or "10 MB", not simply "10".

1.1.9.1. Web Properties

These properties pertain to the web-based User Interface.

Property	Description
nifi.registry.web.war.directory	This is the location of the web war directory. The default value is `./lib`.
nifi.registry.web.http.host	The HTTP host. It is blank by default.
nifi.registry.web.http.port	The HTTP port. The default value is `18080`.
nifi.registry.web.https.host	The HTTPS host. It is blank by default.
nifi.registry.web.https.port	The HTTPS port. It is blank by default. When configuring NiFi Registry to run securely, this port should be configured.
nifi.registry.web.jetty.working.directory	The location of the Jetty working directory. The default value is `./work/jetty`.
nifi.registry.web.jetty.threads	The number of Jetty threads. The default value is `200`.

1.1.9.2. Security Properties

These properties pertain to various security features in NiFi Registry. Many of these properties are covered in more detail in the Security Configuration section of this Administrator's Guide.

Property	Description
nifi.registry.security.keystore	The full path and name of the keystore. It is blank by default.
nifi.registry.security.keystoreType	The keystore type. It is blank by default.
nifi.registry.security.keystorePasswd	The keystore password. It is blank by default.
nifi.registry.security.keyPasswd	The key password. It is blank by default.
nifi.registry.security.truststore	The full path and name of the truststore. It is blank by default.
nifi.registry.security.truststoreType	The truststore type. It is blank by default.
nifi.registry.security.truststorePasswd	The truststore password. It is blank by default.
nifi.registry.security.needClientAuth	This specifies that connecting clients must authenticate with a client cert. Setting this 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. The default value is true.
nifi.registry.security.authorizers.configuration.file	This is the location of the file that specifies how authorizers are defined. The default value is `./conf/authorizers.xml`.
nifi.registry.security.authorizer	Specifies which of the configured Authorizers in the authorizers.xml file to use. By default, it is set to `managed-authorizer`.
nifi.registry.security.identity.providers.configuration.file	This is the location of the file that specifies how username/password authentication is performed. This file is only considered if `nifi.registry.security.identity.provider` is configured with a provider identifier. The default value is ./conf/identity-providers.xml.
nifi.registry.security.identity.provider	This indicates what type of identity provider to use. The default value is blank, can be set to the identifier from a provider in the file specified in `nifi.registry.security.identity.providers.configuration.fi` Setting this property will trigger NiFi Registry to support username/password authentication.

1.1.9.3. Providers Properties

These properties pertain to flow persistence providers. NiFi Registry uses a pluggable flow persistence provider to store the content of the flows saved to the registry. NiFi Registry provides the `FileSystemFlowPersistenceProvider`.

Property	Description
nifi.registry.providers.configuration.file	This is the location of the file where flow persistence providers are configured. The default value is `./conf/providers.xml`.

1.1.9.4. Database Properties

These properties define the settings for the Registry database, which keeps track of metadata about buckets and all items stored in buckets.

Property	Description
nifi.registry.db.directory	The location of the Registry database directory. The default value is `./database`.
nifi.registry.db.url.append	This property specifies additional arguments to add to the connection string for the Registry database. The default

	value should be used and should not be changed. It is: `;LOCK_TIMEOUT=25000;WRITE_DELAY=0;AUTO_SERVER=FALSE`.

1.1.9.5. Extension Directories

Each property beginning with "nifi.registry.extension.dir." will be treated as location for an extension, and a class loader will be created for each location, with the system class loader as the parent.

Property	Description
nifi.registry.extension.dir.1	The full path on the filesystem to the location of the JARs for the given extension
	Multiple extension directories can be specified by using the **nifi.registry.extension.dir.** prefix with unique suffixes and separate paths as values. For example, to provide an additional extension directory, a user could also specify additional properties with keys of: `nifi.registry.extension.dir.2=/path/to/extension2` Providing 2 total locations, including `nifi.registry.extension.dir.1`.

1.1.9.6. Kerberos Properties

Property	Description
nifi.registry.kerberos.krb5.file	The location of the krb5 file, if used. It is blank by default. At this time, only a single krb5 file is allowed to be specified per NiFi instance, so this property is configured here to support SPNEGO and service principals rather than in individual Processors. If necessary the krb5 file can support multiple realms. Example: `/etc/krb5.conf`
nifi.registry.kerberos.spnego.principal	The name of the NiFi Registry Kerberos SPNEGO principal, if used. It is blank by default. Note that this property is used to authenticate NiFi Registry users. Example: `HTTP/nifi.registry.example.com` or `HTTP/nifi.registry.example.com@EXAMPLE.COM`
nifi.registry.kerberos.spnego.keytab.location	The file path of the NiFi Registry Kerberos SPNEGO keytab, if used. It is blank by default. Note that this property is used to authenticate NiFi Registry users. Example: `/etc/http-nifi-registry.keytab`
nifi.registry.kerberos.spengo.authentication.expiration	The expiration duration of a successful Kerberos user authentication, if used. The default value is `12 hours`.