Apache NiFi Registry 3

Apache NiFi Registry System Administrator's Guide

Date of Publish: 2020-12-15

https://docs.cloudera.com/

Contents

System Requirements

NiFi Registry has the following minimum system requirements:

- Requires Java Development Kit (JDK) 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)

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 directories are created:

- flow_storage
- database
- work
- logs
- run

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

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.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.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.

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.

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

User Authentication

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

Note: 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.

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

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.

Set the following in nifi-registry.properties to enable LDAP username/password authentication:

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

Modify identity-providers.xml to enable the ldap-identity-provider. Here is the sample provided in the file:

```
<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>
```

The ldap-identity-provider has the following properties:

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

Property Name	Description
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>).
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.

Kerberos

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

Set the following in nifi-registry.properties to enable Kerberos username/password authentication:

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

Modify identity-providers.xml to enable the kerberos-identity-provider. Here is the sample provided in the file:

```
<provider>
    <identifier>kerberos-identity-provider</identifier>

 <class>org.apache.nifi.registry.web.security.authentication.kerberos.KerberosIdentityPr
class>
    <property name="Default Realm">NIFI.APACHE.ORG</property>
    <property name="Authentication Expiration">12 hours</property>
    <property name="Enable Debug">false</property>
</provider>
```

The kerberos-identity-provider has the following properties:

Property Name	Description
Enable Debug	Enables debug logging output for the SunJaasKerberosClient used internally by the KerberosIdentityProvider. By default, this is set to false.
Default Realm	Default realm to provide when user enters incomplete user principal (i.e. NIFI.APACHE.ORG).
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 *Kerberose Service* to allow single sign-on access via client Kerberos tickets.

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'.

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.

Authorizers.xml Setup

The authorizers.xml file is used to define and configure available authorizers.

StandardManagedAuthorizer

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

Property Name	Description
Access Policy Provider	The identifier for an Access Policy Provider defined above.

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.

UserGroupProvider

FileUserGroupProvider

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

Property Name	Description
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".
Users File	The file where the FileUserGroupProvider stores users and groups. By default, users.xml in the conf directory is chosen.

Note: 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.

LdapUserGroupProvider

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:

Property Name	Description
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. memberUid: user1)
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.

Property Name	Description
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).
Group Membership - Enforce Case Sensitivity	Sets whether group membership decisions are case sensitive. When a user or group is inferred (by not specifying or 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.
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.

Property Name	Description
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.
User Group Name Attribute - Referenced Group Attribute	If blank, the value of the attribute defined in User Group Name Attribute is expected to be the full dn of the group. If not blank, this property will define the attribute of the group LDAP entry that the value of the attribute defined in User Group Name Attribute is referencing (i.e. name). Use of this property requires that Group Search Base is also configured.
Group Search Base	Base DN for searching for groups (i.e. ou=groups,o=nifi). Required to search groups.
Group Object Class	Object class for identifying groups (i.e. groupOfNames). Required if searching groups.
Group Search Scope	Search scope for searching groups (ONE_LEVEL, OBJECT, or SUBTREE). Required if searching groups.
Group Search Filter	Filter for searching for groups against the Group Search Base. Optional.
Group Name Attribute	Attribute to use to extract group name (i.e. cn). Optional. If not set, the entire DN is used.
Group Member Attribute	Attribute to use to define group membership (i.e. member). Optional. If not set group membership will not be calculated through the groups. Will rely on group 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 memberUid 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. memberUid: user1)

Composite Implementations

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:

Property Name	Description
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:

Property Name	Description
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"
Configurable User Group Provider	A configurable user group provider.

AccessPolicyProvider

After you have configured a UserGroupProvider, you must configure an AccessPolicyProvider that will control Access Policies for the identities in the UserGroupProvider.

FileAccessPolicyProvider

The default AccessPolicyProvider is the FileAccessPolicyProvider, however, you can develop additional AccessPolicyProvider as extensions. The FileAccessPolicyProvider has the following properties:

Property Name	Description
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.
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.

Note: 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.

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".

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.

Note: 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 both Proxy permissions and permission to Buckets/READ in order to read all buckets.

Some common use cases are described below.

File-based (LDAP Authentication)

Here is an example certificate DN 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>
```

File-based (Kerberos Authentication)

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>
```

LDAP-based Users/Groups Referencing User DN

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="Group Membership - Enforce Case Sensitivity">false</
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="User Group Name Attribute - Referenced Group
 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>
        <property name="Group Member Attribute - Referenced User
 Attribute"></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.

Composite - File and LDAP-based Users/Groups

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="Group Membership - Enforce Case Sensitivity">false</
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="User Group Name Attribute - Referenced Group
 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>
        <property name="Group Member Attribute - Referenced User
 Attribute"></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.

Access Policies

You can manage the ability for users and groups to view or modify NiFi Registry resources using 'access policies'. Access policies can be created to control access to buckets, as well as to grant special privileges to users for managing a NiFi Registry instance.

Bucket Policies

Bucket policies govern the following bucket level authorizations:

Policy	Privilege	Resource Descriptor
Read Bucket	Allows users to read items in the bucket	resource="/buckets/<bucket-UUID>" action="R"
Write Bucket	Allows users to write items to the bucket	resource="/buckets/<bucket-UUID>" action="W"
Delete Bucket	Allows users to delete the bucket	resource="/buckets/<bucket-UUID>" action="D"

Special Privilege Policies

Special privilege policies govern the following system level authorizations:

Policy	Privilege	Resource Descriptor
Can Manage Buckets (Read)	Allows users to read from all buckets	resource="/buckets" action="R"
Can Manage Buckets (Write)	Allows users to write to all buckets	resource="/buckets" action="W"
Can Manage Buckets (Delete)	Allows users to delete all buckets	resource="/buckets" action="D"
Can Manage Users (Read)	Allows users to view users	resource="/tenants" action="R"
Can Manage Users (Write)	Allows users to create and modify users	resource="/tenants" action="W"
Can Manage Users (Delete)	Allows users to delete users	resource="/tenants" action="D"
Can Manage Policies (Read)	Allows users to view policies	resource="/policies" action="R"
Can Manage Policies (Write)	Allows users to create and modify policies	resource="/policies" action="W"
Can Manage Policies (Delete)	Allows users to delete policies	resource="/policies" action="D"
Can Proxy Requests (Read)	Allows users to proxy read requests (GET)	resource="/proxy" action="R"
Can Proxy Requests (Write)	Allows users to proxy write requests (POST, PUT, PATCH)	resource="/proxy" action="W"
Can Proxy Requests (Delete)	Allows users to proxy delete requests (DELETE)	resource="/proxy" action="D"
View Swagger	Allows users to access the self-hosted Swagger UI	resource="/swagger" action="R"
View Actuator	Allows users to access the Spring Boot Actuator end-points	resource="/actuator" action="R"

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.

Note: 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.

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 --nifiRegistry [options]
```

You can use the following command line options with the encrypt-config tool:

- -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, assume 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=0123456789ABCDEFFEDCBA98765432100123456789ABCDEFFE
```

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 --nifiRegistry -b bootstrap.conf -r nifi-
registry.properties
```

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

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.

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 to the documentation of the proxy for guidance with 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>
...
```

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* 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.

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 https://docs.spring.io/autorepo/docs/spring-security-kerberos/1.0.2.BUILD-SNAPSHOT/reference/htmlsingle/#browserspnegoconfig 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 https://web.mit.edu/kerberos/krb5-current/doc/admin/index.html), 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 Jave 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:~#
```

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.

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

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.

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.

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.file. Setting this property will trigger NiFi Registry to support username/password authentication.

Identity Mapping Properties

These 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 Registry. 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 DNs from certificates and principals from Kerberos:

```
nifi.registry.security.identity.mapping.pattern.dn=^CN=(.*?), OU=(.*?),
 O=(.*?), L=(.*?), ST=(.*?), C=(.*?)$
nifi.registry.security.identity.mapping.value.dn=$1@$2
nifi.registry.security.identity.mapping.transform.dn=NONE
nifi.registry.security.identity.mapping.pattern.kerb=^(.*?)/instance@(.*?)$
nifi.registry.security.identity.mapping.value.kerb=$1@$2
nifi.registry.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 Registry, 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.registry.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" in the authorizers.xml file, as well as users imported from LDAP (See Authorizers.xml Setup).

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

Note: These mappings are applied to groups imported from LDAP.

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. For further details on persistence providers, refer *Persistence Providers*.

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.

Alias Properties

These properties pertain to the support for URL aliasing. For further details, refer to URL Aliasing.

Property	Description
nifi.registry.registry.alias.configuration.file	This is the location of the file where URL aliases are configured. The default value is ./conf/registry-aliases.xml.

Database Properties

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

The 0.1.0 release leveraged an embedded H2 database that was configured via the following properties:

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.

The 0.2.0 release introduced a more flexible approach which allows leveraging an external database. This new approach is configured via the following properties:

Property	Description
nifi.registry.db.url	The full JDBC connection string. The default value will specify a new H2 database in the same location as the previous one. For example, jdbc:h2:./database/nifi-registry-primary;.
nifi.registry.db.driver.class	The class name of the JDBC driver. The default value is org.h2.Driver.
nifi.registry.db.driver.directory	An optional directory containing one or more JARs to add to the classpath. If not specified, it is assumed that the driver JAR is already on the classpath by copying it to the lib directory. The H2 driver is bundled with Registry so it is not necessary to do anything for the default case.
nifi.registry.db.username	The username for the database. The default value is nifireg.
nifi.registry.db.password	The password for the database. The default value is nifireg.
nifi.registry.db.maxConnections	The max number of connections for the connection pool. The default value is 5.
nifi.registry.db.sql.debug	Whether or not enable debug logging for SQL statements. The default value is false.

Note: When upgrading from 0.1.0 to a future version, if nifi.registry.db.directory remains populated, the application will attempt to migrate the data from the original database to the new database specified with the new properties. This will only happen the first time the application starts with the new database properties.

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

Note: 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.

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.

Metadata Database

The metadata database maintains the knowledge of which buckets exist, which versioned items belong to which buckets, as well as the version history for each item.

Currently, NiFi Registry supports using H2, Postgres 9.x, and MySQL (5.6, 5.7, 8.0) for the relational database engine.

Note: NiFi Registry 0.1.0 only supports H2.

H2

H2 is an embedded database that is pre-configured in the default nifi-registry.properties file. The contents of the H2 database are stored in a file on the local filesystem.

For NiFi Registry 0.1.0, the location of the H2 database is specified by the property:

nifi.registry.db.directory=./database

For NiFi Registry 0.2.0 and forward, the location of the H2 database is specified as part of the JDBC URL property:

nifi.registry.db.url=jdbc:h2:./database/nifi-registry-primary;

Postgres

Postgres provides the option to use an externally located database that also supports high availability.

The following steps are required to use Postgres:

1. Download the Postgres JDBC driver and place it somewhere accessible to NiFi Registry

```
/path/to/drivers/postgresql-42.2.2.jar
```

2. Create a database inside Postgres

```
createdb nifireg
```

3. Create a database user and grant privileges

```
psql nifireg
CREATE USER nifireg WITH PASSWORD 'changeme';
GRANT ALL PRIVILEGES ON DATABASE nifireg to nifireg;
\q
```

4. Configure the database properties in nifi-registry.properties

```
nifi.registry.db.url=jdbc:postgresql://<POSTGRES-HOSTNAME>/nifireg
nifi.registry.db.driver.class=org.postgresql.Driver
nifi.registry.db.driver.directory=/path/to/drivers
nifi.registry.db.username=nifireg
nifi.registry.db.password=changeme
```

MySQL

MySQL also provides the option to use an externally located database that also supports high availability.

The following steps are required to use MySQL:

1. Download the MySQL JDBC driver and place it somewhere accessible to NiFi Registry

```
/path/to/drivers/mysql-connector-java-8.0.16.jar
```

2. Create a database inside MySQL (enter mysql shell using mysql -u root -p

```
CREATE DATABASE nifi_registry;
```

3. Create a database user and grant privileges (for remote users, use nifireg'@'<IP-ADDRESS>, or nifireg'@'% for any remote host)

```
GRANT ALL PRIVILEGES ON nifi_registry.* TO 'nifireg'@'localhost'
 IDENTIFIED BY 'changeme';
```

4. Configure the database properties in nifi-registry.properties

```
nifi.registry.db.url=jdbc:mysql://<MYSQL-HOSTNAME>/nifi_registry
nifi.registry.db.driver.class=com.mysql.cj.jdbc.Driver
nifi.registry.db.driver.directory=/path/to/drivers
nifi.registry.db.username=nifireg
nifi.registry.db.password=changeme
```

Schema Differences & Limitations

Due to differences across database implementations, there are two versions of the schema for NiFi Registry's metadata database. The original version supports H2 and Postgres, and a second versions supports MySQL.

MySQL has limitations on the maximum size of text columns that are part of an index, or unique key. This means the maximum length of some columns is significantly less when using MySQL vs. H2/Postgres.

Note: If choosing to use MySQL it is important to understand these limitations and accept them.

The following tables summarizes the schema differences in column lengths:

Table.Column	H2/Postgres	MySQL
BUCKET.NAME	1000	767
FLOW_SNAPSHOT.CREATED_BY	4096	767
SIGNING_KEY.TENANT_IDENTITY	4096	767
BUNDLE.GROUP_ID	500	200
BUNDLE.ARTIFACT_ID	500	200
BUNDLE_VERSION.CREATED_BY	4096	767
BUNDLE_VERSION.BUILT_BY	4096	767
BUNDLE_VERSION_DEPENDENCY.GROUP_ID	500	200
BUNDLE_VERSION_DEPENDENCY.ARTIFACT_ID	500	200
EXTENSION_PROVIDED_SERVICE_API.CLASS_NAME	500	200
EXTENSION_PROVIDED_SERVICE_API.GROUP_ID	500	200
EXTENSION_PROVIDED_SERVICE_API.ARTIFACT_ID	500	200

Persistence Providers

NiFi Registry uses a pluggable persistence provider to store the content of each versioned item. Each type of versioned item, such as a versioned flow or extension bundle, has its own persistence provider.

Each persistence provider has its own configuration parameters, which can be configured in an XML file specified in *Providers Properties*.

Flow Persistence Providers

The flow persistence provider stores the content of the flows saved to the registry.

The XML configuration file looks like below. It has a flowPersistenceProvider element in which qualified class name of a persistence provider implementation and its configuration properties are defined. See following sections for available configurations for each provider.

Example flow persistence provider in providers.xml

```
        <flowPersistenceProvider>
    <class>persistence-provider-qualified-class-name</class>
    <property name="property-1">property-value-1</property>
    <property name="property-2">property-value-2</property>
    <property name="property-n">property-value-n</property>
</flowPersistenceProvider>
```

FileSystemFlowPersistenceProvider

FileSystemFlowPersistenceProvider simply stores serialized Flow contents into {bucket-id}/{flow-id}/{version} directories.

Example of persisted files:

```
Flow Storage Directory/
### {bucket-id}/
#   ### {flow-id}/
#       ### {version}/{version}.snapshot
### d1beba88-32e9-45d1-bfe9-057cc41f7ce8/
    ### 219cf539-427f-43be-9294-0644fb07ca63/
        ### 1/1.snapshot
        ### 2/2.snapshot
```

Qualified class name: org.apache.nifi.registry.provider.flow.FileSystemFlowPersistenceProvider

Property	Description
Flow Storage Directory	REQUIRED: File system path for a directory where flow contents files are persisted to. If the directory does not exist when NiFi Registry starts, it will be created. If the directory exists, it must be readable and writable from NiFi Registry.

GitFlowPersistenceProvider

GitFlowPersistenceProvider stores flow contents under a Git directory.

In contrast to FileSystemFlowPersistenceProvider, this provider uses human friendly Bucket and Flow names so that those files can be accessed by external tools. However, it is NOT supported to modify stored files outside of NiFi Registry. Persisted files are only read when NiFi Registry starts up.

Buckets are represented as directories and Flow contents are stored as files in a Bucket directory they belong to. Flow snapshot histories are managed as Git commits, meaning only the latest version of Buckets and Flows exist in the Git directory. Old versions are retrieved from Git commit histories.

Example persisted files

```
Flow Storage Directory/
### .git/
### Bucket_A/
#   ### bucket.yml
#   ### Flow_1.snapshot
#   ### Flow_2.snapshot
### Bucket_B/
    ### bucket.yml
    ### Flow_4.snapshot
```

Each Bucket directory contains a YAML file named bucket.yml. The file manages links from NiFi Registry Bucket and Flow IDs to actual directory and file names. When NiFi Registry starts, this provider reads through Git commit histories and lookup these bucket.yml files to restore Buckets and Flows for each snapshot version.

Example bucket.yml

```
        layoutVer: 1
bucketId: d1beba88-32e9-45d1-bfe9-057cc41f7ce8
flows:
  219cf539-427f-43be-9294-0644fb07ca63: {ver: 7, file: Flow_1.snapshot}
  22cccb6c-3011-4493-a996-611f8f112969: {ver: 3, file: Flow_2.snapshot}
```

Qualified class name: org.apache.nifi.registry.provider.flow.git.GitFlowPersistenceProvider

Property	Description
Flow Storage Directory	REQUIRED: File system path for a directory where flow contents files are persisted to. The directory must exist when NiFi registry starts. Also must be initialized as a Git directory.
Remote To Push	When a new flow snapshot is created, this persistence provider updates files in the specified Git directory, then creates a commit to the local repository. If Remote To Push is defined, it also pushes to the specified remote repository (e.g. origin). To define more detailed remote spec such as branch names, use Refspec (see https://git-scm.com/book/en/v2/Git-Internals-The-Refspec).
Remote Access User	This username is used to make push requests to the remote repository when Remote To Push is enabled, and the remote repository is accessed by HTTP protocol. If SSH is used, user authentication is done with SSH keys.
Remote Access Password	The password for the Remote Access User.

Initialize Git directory

In order to use GitFlowPersistenceRepository, you need to prepare a Git directory on the local file system. You can do so by initializing a directory with git init command, or clone an existing Git project from a remote Git repository by git clone command.

- git init command https://git-scm.com/docs/git-init
- git clone command https://git-scm.com/docs/git-clone

Git user configuration

This persistence provider uses preconfigured Git user name and user email address when it creates Git commits. NiFi Registry user name is added to commit messages.

Example commit

```
commit 774d4bd125f2b1200f0a5ee1f1e9fedc6a415e83
Author: git-user <git-user@example.com>
Date:   Tue May 8 14:30:31 2018 +0900

    Commit message.

    By NiFi Registry user: nifi-registry-user-1
```

You can configure Git user name and email address by git config command.

- git config command https://git-scm.com/docs/git-config

Git user authentication

By default, this persistence repository only create commits to local repository. No user authentication is needed to do so. However, if 'Commit To Push' is enabled, user authentication to the remote Git repository is required.

If the remote repository is accessed by HTTP, then username and password for authentication can be configured in the providers XML configuration file.

When SSH is used, SSH keys are used to identify a Git user. In order to pick the right key to a remote server, the SSH configuration file ${USER_HOME}/.ssh/config is used. The SSH configuration file can contain multiple Host entries to specify a key file to login to a remote Git server. The Host must match with the target remote Git server hostname.

example SSH config file

```
Host git.example.com
  HostName git.example.com
  IdentityFile ~/.ssh/id_rsa

Host github.com
  HostName github.com
  IdentityFile ~/.ssh/key-for-github

Host bitbucket.org
  HostName bitbucket.org
  IdentityFile ~/.ssh/key-for-bitbucket
```

DatabaseFlowPersistenceProvider

DatabaseFlowPersistenceProvider stores flow contents in a database table.

This provider leverages the same database used for the metadata database, so there is no configuration to provide since the connection details will come from the database properties in nifi-registry.properties.

The database table is named FLOW_PERSISTENCE_PROVIDER and has the following schema:

Column	Description
BUCKET_ID	The identifier of the bucket where the flow is located.
FLOW_ID	The identifier of the flow.
VERSION	The version of the flow.
FLOW_CONTENT	The serialized bytes of the flow content stored as a BLOB.

Switching from other Flow Persistence Provider

In order to switch the Flow Persistence Provider, it is necessary to reset NiFi Registry. For example, to switch from FileSystemFlowPersistenceProvider to GitFlowPersistenceProvider, follow these steps:

1. Stop version control on all ProcessGroups in NiFi
2. Stop NiFi Registry
3. Move the H2 DB (specified as nifi.registry.db.directory in nifi-registry.properties) and Flow Storage Directory for FileSystemFlowPersistenceProvider directories somewhere for back up
4. Configure GitFlowPersistenceProvider provider in providers.xml
5. Start NiFi Registry
6. Recreate any buckets
7. Start version control on all ProcessGroups again

Data model version of serialized Flow snapshots

Serialized Flow snapshots saved by these persistence providers have versions, so that the data format and schema can evolve over time. Data model version update is done automatically by NiFi Registry when it reads and stores each Flow content.

Here is the data model version histories:

Data model version	Since NiFi Registry	Description
2	0.2	JSON formatted text file. The root object contains header and Flow content object.
1	0.1	Binary format having header bytes at the beginning followed by Flow content represented as XML.

Bundle Persistence Providers

The bundle persistence provider stores the content of extension bundles saved to the registry.

The XML configuration file looks like below. It has a extensionBundlePersistenceProvider element in which the qualified class name of a persistence provider implementation and its configuration properties are defined. See following sections for available configurations for each provider.

Example extension bundle persistence provider in providers.xml

```
        <extensionBundlePersistenceProvider>
    <class>persistence-provider-qualified-class-name</class>
    <property name="property-1">property-value-1</property>
    <property name="property-2">property-value-2</property>
    <property name="property-n">property-value-n</property>
</extensionBundlePersistenceProvider>
```

FileSystemBundlePersistenceProvider

The FileSystemBundlePersistenceProvider stores the content of extension bundles on the local file system. The bundles are organized in directories according to bucket id, group, artifact, and version.

Example of persisted extension bundles:

```
Extension Bundle Storage Directory/
### {bucket-id}/
   ### {group-id}/
       ### {artifact-id}
           ### {version}/{artifact-id}-{version}.{extension}
### d1beba88-32e9-45d1-bfe9-057cc41f7ce8/
    ### org.apache.nifi
        ### nifi-example-nar
            ### 1.0.0/nifi-example-nar-1.0.0.nar
            ### 2.0.0/nifi-example-nar-2.0.0.nar
```

Configuration

Qualified class name: org.apache.nifi.registry.provider.extension.FileSystemBundlePersistenceProvider

Property	Description

Extension Bundle Storage Directory	REQUIRED: File system path for a directory where extension bundle contents files are persisted to. If the directory does not exist when NiFi Registry starts, it will be created. If the directory exists, it must be readable and writable from NiFi Registry.

S3BundlePersistenceProvider

The S3BundlePersistenceProvider stores the content of extension bundles in a AWS S3 bucket. The bucket is expected to already exist and be accessible to the credentials provided to the persistence providcer.

Note: This provider must be added to the classpath by specifying a custom extension directory in nifi-registry.properties, such as nifi.registry.extension.dir.aws=./ext/aws/lib, where ./ext/aws/ contains the contents of the extracted nifi-registry-aws-assembly-<version>-bin.zip.

The key of an extension bundle in the S3 bucket will be the following:

```
/{registry-bucket-id}/{group-id}/{artifact-id}/{version}/{artifact-id}-
{version}.{extension}
```

If an optional Key Prefix is specified, then that prefix will be applied to the beginning of the above key.

Configuration

Qualified class name: org.apache.nifi.registry.aws.S3BundlePersistenceProvider

Property	Description
Region	REQUIRED: The name of the S3 region where the bucket exists.
Bucket Name	REQUIRED: The name of an existing bucket to store extension bundles.
Key Prefix	An optional prefix that if specified will be added to the beginning of all S3 keys.
Credentials Provider	REQUIRED: Indicates how credentials will be provided, must be a value of DEFAULT_CHAIN or STATIC. DEFAULT_CHAIN will consider in order: Java system properties, environment variables, credential profiles (~/.aws/credentials). STATIC requires that Access Key and Secret Access Key be specified directly in this file.
Access Key	The access key to use when using STATIC credentials provider.
Secret Access Key	The secret access key to use when using STATIC credentials provider.
Endpoint URL	An optional URL that overrides the default AWS S3 endpoint URL. Set this when using an AWS S3 API compatible service hosted at a different URL.

Event Hooks

Event hooks are an integration point that allows for custom code to to be triggered when NiFi Registry application events occur.

Event Name	**Description**
REGISTRY_START	Invoked once the NiFi Registry application has been successfully started. This is only invoked after a complete and successful start.
CREATE_BUCKET	A new registry bucket is created.

Event Name	Description
CREATE_FLOW	A new flow is created in a specified bucket. Only triggered on first time creation of a flow with a given name.
CREATE_FLOW_VERSION	A new version for a flow has been saved in the registry.
UPDATE_BUCKET	A bucket has been updated.
UPDATE_FLOW	A flow that exist in a bucket has been updated.
DELETE_BUCKET	An existing bucket in the registry is deleted.
DELETE_FLOW	An existing flow in the registry is deleted.

Shared Event Hook Properties

There are certain properties that are shared amongst all of the NiFi Registry provided Event Hook implementations. Those properties and their purpose are listed below.

Property Name	Description
Whitelisted Event Type	Event types the hook provider configured with this property should respond to. If this property is left blank or not provided, all events will fire for the configured hook provider. Multiple Whitelisted Event Type can be specified and often are. For example, <property name="Whitelisted Event Type 1">CREATE_FLOW</property> and <property name="Whitelisted Event Type 2">UPDATE_FLOW</property> would invoke the configured hook provider for the CREATE_FLOW and UPDATE_FLOW event types.

ScriptEventHookProvider

The ScriptEventHookProvider invokes a shell script that has been written by a user and placed on a file system that is accessible by the NiFi Registry instance that the provider is configured for.

```
<eventHookProvider>
    <class>org.apache.nifi.registry.provider.hook.ScriptEventHookProvider</
class>
    <property name="Script Path"></property>
    <property name="Working Directory"></property>
    <!-- optional -->
        <property name="Whitelisted Event Type 1">CREATE_FLOW</property>
        <property name="Whitelisted Event Type 2">UPDATE_FLOW</property>
</eventHookProvider>
```

Property Name	Description
Working Directory	Working directory from where the commands will be executed.
Script Path	Full path to a script that will executed for each event. The arguments to the script will be the event fields in the order they are specified for the given event type.

LoggingEventHookProvider

The LoggingEventHookProvider logs a string representation of each event using an SLF4J logger. The logger can be configured via NiFi Registry's logback.xml, which by default contains an appender that writes to a log file named nifi-registry-event.log in the logs directory.

```
<eventHookProvider>
    <class>
      org.apache.nifi.registry.provider.hook.LoggingEventHookProvider
    </class>
</eventHookProvider>
```

URL Aliasing

A versioned item may contain the URL of a registry instance embedded in the content of the item. For example, flows with nested versioning contain the URL of the registry where the nested versioned flow is located. If the location of the registry instances changes, then the content is no longer accurate.

URL aliasing can be used to dynamically handle this situation so that URLs are never written to the stored content, and can be re-written with the correct value when being retrieved by a client.

The aliases are configured in an XML file which can be specified in *Alias Properties* .

If a flow is saved to registry with two child process groups, each under version control, the incoming flow would contain something like the following:

```
"processGroups" : [ {
      ...
      "versionedFlowCoordinates" : {
        "bucketId" : "ca20e058-f6e7-404c-aee0-e30833e792c7",
        "flowId" : "178a6657-e1a7-4cce-8f83-4e615e38f57a",
        "registryUrl" : "http://registry1.nifi.apache.org:18080",
        "version" : 1
      },
      {
      ...
      "versionedFlowCoordinates" : {
        "bucketId" : "ca20e058-f6e7-404c-aee0-e30833e792c7",
        "flowId" : "985cb44b-3aec-32be-860f-d2a0f2c72aac",
        "registryUrl" : "http://registry2.nifi.apache.org:18080",
        "version" : 1
      }
]
```

With the example aliases configuration above, the URLs would be written to the flow persistence provider as the following:

```
"processGroups" : [ {
      ...
      "versionedFlowCoordinates" : {
        "bucketId" : "ca20e058-f6e7-404c-aee0-e30833e792c7",
        "flowId" : "178a6657-e1a7-4cce-8f83-4e615e38f57a",
        "registryUrl" : "NIFI_REGISTRY_1",
        "version" : 1
      },
      {
      ...
      "versionedFlowCoordinates" : {
        "bucketId" : "ca20e058-f6e7-404c-aee0-e30833e792c7",
        "flowId" : "985cb44b-3aec-32be-860f-d2a0f2c72aac",
        "registryUrl" : "NIFI_REGISTRY_2",
```

```
        "version" : 1
      }
]
```

When this flow is retrieved from any API call, the internal values would be rewritten to the external values.

Backup & Recovery

In order to prevent data loss it is important to consider backup and recovery options. The data that needs to be considered is the following:

- Metadata Database
- Persistence providers
- Configuration files

Metadata Database

If using H2, the database file should be backed up periodically to an external location. In order to ensure a proper backup, NiFi Registry should be stopped to ensure no write operations are occurring while copying the file.

If using Postgres, backups may be taken on the Postgres database, or Postgres may be configured for high availability such that there is a failover or backup instance.

If starting a brand new NiFi Registry instance, the metadata database can be automatically rebuilt from the information in the GitFlowPersistenceProvider. This is a one-time operation during the first start of the application, and is not meant to keep the DB in sync with external changes made in Git. This feature only applies to flows and would not be able to restore information about extension bundles.

Persistence Providers

Each persistence provider may have its own option for backup & recovery.

Flow Persistence

If using the FileSystemFlowPersistenceProvider, the directory where flows are stored should be backed up periodically to an external location. In order to ensure a proper backup, NiFi Registry should be stopped to ensure no flows are being written to disk. If using H2 for metadata, H2 should be backed up at the same time to ensure consistency between the flows on disk and the contents in H2.

If using the GitFlowPersistenceProvider, the ability to automatically push to a remote may be configured. This provides an automatic backup of the data in the remote repo.

Bundle Persistence

If using the FileSystemBundlePersistenceProvider, the directory where bundles are stored should be backed up periodically to an external location. In order to ensure a proper backup, NiFi Registry should be stopped to ensure no bundles are being written to disk. If using H2 for metadata, H2 should be backed up at the same time to ensure consistency between the bundles on disk and the contents in H2.

If using the S3BundlePersistenceProvider, data will be stored remotely and automatically replicated.

Configuration Files

If using NiFi Registry's policy based authorization, the users, groups, and policies are stored in files on disk named users.xml and authorizations.xml. These files should be periodically backed up to an external location. In order to ensure a proper backup, NiFi Registry should be stopped to ensure no authorization data is being written to disk.

If using Ranger, then all authorization information is stored externally and there is nothing to back up.