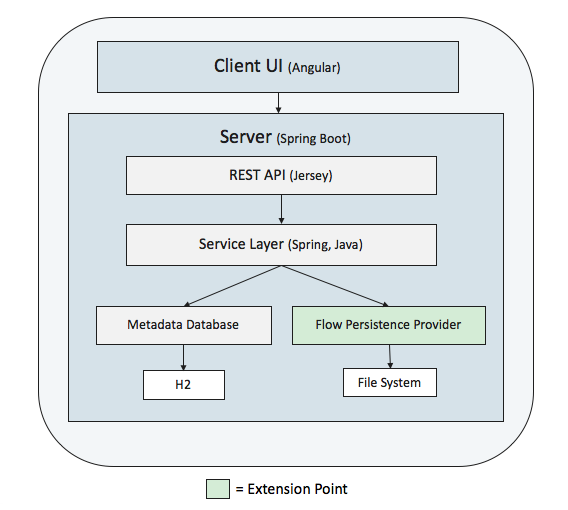
**Apache NiFi Registry 0.2.0**

The Apache NiFi community recently completed the second release of NiFi Registry (0.2.0). This post will introduce some of the new features, including git-based flow persistence, a more configurable metadata database, and event hooks.

**Background**

Before jumping into the new features, lets review how NiFi Registry works behind the scenes.

The user interface is a single page webapp built with Angular, and communicates with the server via the NiFi Registry REST API. Behind the REST API is a service layer where the primary business logic is implemented, and the service layer interacts with the metadata database and flow persistence provider.



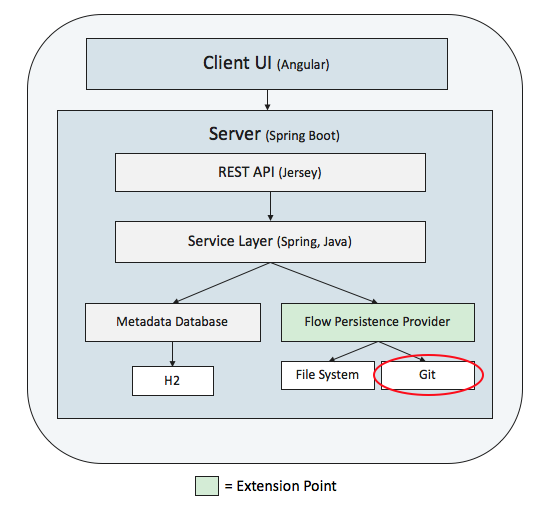
The *metadata database* stores information about buckets and versioned items, such as identifiers, names, descriptions, and commit comments, as well as which items belong to which bucket. The initial release utilized an embedded H2 database that was primarily hidden from end users, except for configuring the directory location.

The *flow persistence provider* stores the content of each versioned flow and is considered a public extension point. This means custom implementations can be provided by implementing the FlowPersistenceProvider interface. The initial release provided a file-system implementation of flow persistence provider which used the local file-system for persistence.

The overall idea is that the metadata database contains information across all types of versioned items (which may eventually be more than just flows), and each type of item may have its own persistence mechanism for the content of the item.

**Git FlowPersistenceProvider**

The 0.2.0 release provides a new git-based implementation of the FlowPersistenceProvider utilizing the JGit library. This means the content of versioned flows can now be stored in a git repository.



The git provider can be configured in providers.xml via the following configuration:

<flowPersistenceProvider>

<class>

org.apache.nifi.registry.provider.flow.git.GitFlowPersistenceProvider

</class>

<property name="Flow Storage Directory">./flow\_storage</property>

<property name="Remote To Push"></property>

<property name="Remote Access User"></property>

<property name="Remote Access Password"></property>

</flowPersistenceProvider>

The *“Flow Storage Directory”* property specifies a local directory that is expected to already be a git repository. This could be done by creating a new directory and running “git init”, or from cloning an existing remote repository.

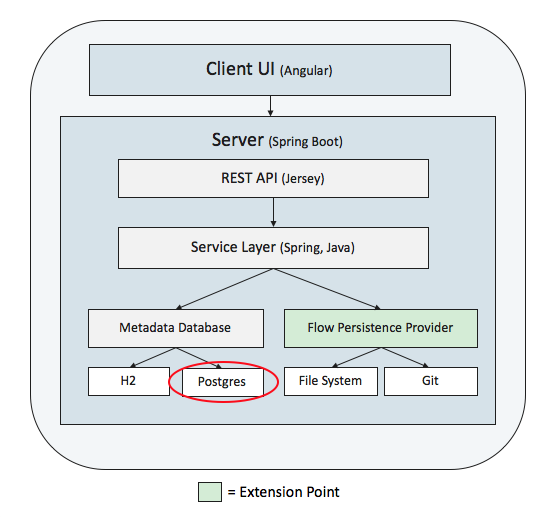
The *“Remote To Push”* property specifies the name of remote to automatically push to. This property is optional and if not specified then commits will remain in the local repository, unless a push is performed manually.

NOTE: In order to use GitHub, do the following:

* Create a new GitHub repo and clone it locally
* Set “Flow Storage Directory” to the directory where the repo was cloned
* Go to GitHub’s “Developer Settings” for your account
* Create a new “Personal Access Token” for your NiFi Registry instance
* Set “Remote Access User” to your GitHub username
* Set “Remote Access Password” to the access token

**External Metadata Database**

The 0.2.0 release provides new configuration that allows the metadata database to utilize an external database. Currently Postgres is the only supported database besides H2, although others may work with additional testing.



The 0.1.0 release originally had two properties in nifi-registry.properties related to the H2 database:

nifi.registry.db.directory=

nifi.registry.db.url.append=

In order to make the database more configurable, the 0.2.0 release adds the following properties:

nifi.registry.db.url=

nifi.registry.db.driver.class=

nifi.registry.db.driver.directory=

nifi.registry.db.username=

nifi.registry.db.password=

If you are starting NiFi Registry for the first time, then the properties will default to use H2 and you don’t need to do anything unless you want to change the configuration to use a Postgres database.

If you are upgrading an existing NiFi Registry, it will determine if the old nifi.registry.db.directory property is populated and determine if the directory contains an existing H2 database. If an existing database is found, and if the target database is empty, then data will automatically be migrated from the existing database to the new database.

NOTE: This essentially offers a one-time migration from the existing H2 database, to a new H2 database, or a new Postgres database.

For specifics about each property please refer to the [NiFi Registry Administration Guide](https://nifi.apache.org/docs/nifi-registry-docs/index.html).

**Event Hooks**

An event hook is a new extension point that allows custom code to be triggered when application events occur.

In order to implement an event hook, the following Java interface must be implemented:

public interface EventHookProvider extends Provider {

void handle(Event event) throws EventHookException;

}

The Event object will contain an EventType, along with a list of field/value pairs that are specific to the event. At the time of writing this, the possible event types are the following:

CREATE\_BUCKET

CREATE\_FLOW

CREATE\_FLOW\_VERSION

UPDATE\_BUCKET

UPDATE\_FLOW

DELETE\_BUCKET

DELETE\_FLOW

The list of event types and fields can be found in the code in the [EventType class](https://github.com/apache/nifi-registry/blob/master/nifi-registry-provider-api/src/main/java/org/apache/nifi/registry/hook/EventType.java).

The 0.2.0 release comes with two provided event hooks, *LoggingEventHookProvider* and *ScriptedEventHookProvider*.

**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.

To enable this hook, simply uncomment the configuration in providers.xml:

<eventHookProvider>

<class>

org.apache.nifi.registry.provider.hook.LoggingEventHookProvider

</class>

</eventHookProvider>

After creating a bucket and starting version control on a process group, the event log should show something like the following:

2018-06-19 11:30:46,174 ## CREATE\_BUCKET [BUCKET\_ID=5d81dc5e-79e1-4387-8022-79e505f5e3a0, USER=anonymous]

2018-06-19 11:32:16,503 ## CREATE\_FLOW [BUCKET\_ID=5d81dc5e-79e1-4387-8022-79e505f5e3a0, FLOW\_ID=a89bf6b7-41f9-4a96-86d4-0aeb3c3c25be, USER=anonymous]

2018-06-19 11:32:16,610 ## CREATE\_FLOW\_VERSION [BUCKET\_ID=5d81dc5e-79e1-4387-8022-79e505f5e3a0, FLOW\_ID=a89bf6b7-41f9-4a96-86d4-0aeb3c3c25be, VERSION=1, USER=anonymous, COMMENT=v1]

**ScriptEventHookProvider**

The *ScriptedEventHookProvider* allows a custom script to be executed for each event. This can be used to handle many situations without having to formally implement an event hook in Java.

The configuration for this event hook looks like the following:

<eventHookProvider>

<class>

org.apache.nifi.registry.provider.hook.ScriptEventHookProvider

</class>

<property name="Script Path"></property>

<property name="Working Directory"></property>

</eventHookProvider>

The *“Script Path”* property is the 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. As an example, lets say we created a script called logging-hook.sh with the following content:

#!/bin/bash

echo $@ >> /tmp/nifi-registry-event.log

For each event, this would echo the arguments to the script and append them to /tmp/nifi-registry-event.log. After creating a bucket and starting version control on a process group, the log should show something like the following:

CREATE\_BUCKET feeb0fbe-5d7e-4363-b58b-142fa80775e1 anonymous

CREATE\_FLOW feeb0fbe-5d7e-4363-b58b-142fa80775e1 1a0b614c-3d0f-471a-b6b1-645e6091596d anonymous

CREATE\_FLOW\_VERSION feeb0fbe-5d7e-4363-b58b-142fa80775e1 1a0b614c-3d0f-471a-b6b1-645e6091596d 1 anonymous v1

**Summary**

The new features in the 0.2.0 release provide additional options for the metadata database and flow persistence provider, allowing data to be pushed off the server where the application is running, and on to external locations. In addition, event hooks provide an easy way to initiate custom workflows based on various events.