Configuration Management of Flows

Target release	
Epic	
Document status	DRAFT
Document owner	Joe Witt
Designer	
Developers	
QA	

Goals

- Provide the ability for effective and automated configuration management of dataflows including merges, diffs, and rollbacks.

Background and strategic fit

This capability should aim to address the following gaps that exist today:

1. **Versioned History of a Flow** - Currently users can archive the entire flow, but there is no way to track what has changed at a more granular level, and there is no way to rollback to a previous version of the flow.
2. **Correlation of Provenance Events to a Versioned Flow** - When looking at provenance events it is highly likely that the flow has changed since those events were generated, and there is currently no way to see what the flow looked like at the time those events were generated.
3. **Deployment/SDLC** - Many users operate in a design-and-deploy model using a series of environments (dev, staging, prod). Moving a flow between environments currently requires a significant amount of scripting and manual work involving templates, or possibly copying an entire flow.xml.gz.
4. **Command & Control of MiNiFi** - Currently the deployment model for MiNiFi is to export a template from NiFi and convert the template to a YAML file. Eventually a central command and control capability should be able to deploy a version of a flow to a set of MiNiFi agents.

Overview

If we take the concept of editing code in an IDE, the IDE has a local history that generally has all the local changes made while editing for some time period, allowing you to revert to a point in time in the local changes. When satisfied with your changes, you commit the changes and push them to a remote version control repository, where others can then see your changes.

This would be analogous to working in NiFi and tracking every change in a local flow history that would only be visible within that NiFi instance. A user could then revert to a point in time in the local history (i.e. undo). At some point a portion of the flow would be ready for sharing with others, and the portion of the flow would be pushed to a repository where it could then be consumer by other users/systems. In order to implement this functionality, NiFi could introduce the following concepts:

- **Flow Registries** for storing/managing versioned flows
- **Versioned flows** for explicit saving and retrieving from a given flow registry
- **Local Flow History** for tracking all actions performed in a local NiFi instance and providing the ability to undo, or revert to points in the local history

These capabilities would relate to the original problems in the following ways:

Problem	**Flow Registry/Versioned Flows**	**Local Flow History**
Versioned History of a Flow	Gives the user the ability to create explicit checkpoints with ability to upgrade/rollback to a given checkpoint. Does NOT give “undo” of any action.	Gives the user ability to view every change made to the flow and revert to a given change (“undo”).
Correlation of Provenance to a Version of the Flow	Not used for correlation of provenance events since it is based off explicit user action, and may not represent the state of the flow at time of the event.	Used to correlate a provenance event back to the version of the flow at the time of the event.
Deployment/SDLC	Provides the ability to publish flows to a flow registry which can be imported to another environment, and upgraded to future versions.	Not used for deployment scenarios, only local to a given NiFi instance.
C&C of MiNiFi	Provides a central place (flow registry) for MiNiFi to consume new versions of a flow.	Not used for C&C of MiNiFi, only local to a given NiFi instance.

Local Flow History

Local Flow History refers to a given NiFi instance tracking every change made in the UI and providing users with the ability to view the local history, and revert to points in time in the local history, essentially offering an “undo” capability.

Each incremental change to the flow would be assigned a flow identifier and would be persisted in some way. Periodic checkpoints would compute the representation of the flow at a point in time by applying all incremental updates back to the last checkpoint, or to an empty flow if it is the first checkpoint.

Security Implications

When NiFi is running in secure mode, each incremental change would also capture the user that performed this change. Further design is required to determine the implications of security on functionality involving the Local Flow History.

For example, if user1 and user2 have read/write access to a process group (pg1), but only user1 has read/write to a processor (p1) in that group, and user1 deletes that processor (p1), can user2 ‘undo’ the delete? And once the processor is deleted, how do we even know that user2 didn’t have read/write to it because the policies for that component will be deleted as well?

Provenance

Since the Local Flow History would have every change to the flow, the flow identifier from the Local Flow History could be added as a field to provenance events, allowing a provenance event to compute what the flow looked like during that event.

Flow Registries

The Apache NiFi community would define a REST API for managing versioned flows. A flow registry is any system that provides an implementation of this REST API.

The Apache NiFi community would provide a standard implementation of the REST API. This implementation would be provided as a standalone application with a pluggable back-end for the persistence of versioned flows. The default implementation would persist versioned flows to the filesystem, but additional persistence implementations may be developed by the community. In addition, any system may implement the same REST API without leveraging the application provided by the NiFi community.

A NiFi instance would allow the configuration of zero or more flow registries by entering the URL of the registry. When one or more flow registries are configured, NiFi would enable functionality to interact with versioned flows, described further in the section below.

A goal of this design would be to allow flow registries to be defined through the NiFi UI, and avoid having to manually edit configuration files resulting in a restart of the application.

Versioned Flows

A flow would be placed under version control at the process group level. Being “under version control” means the process group is linked to a versioned flow in a flow registry.

By using process groups as the unit of version control, the entire flow could be versioned by placing the root group under version control, or individual process groups could be version controlled for more granular versioning.

Each versioned flow would consist of the following:

- UUID of the process group it was created from
- Version identifier indicating the version of the given flow
- Parent Version indicating the version that came before this version (blank for v1)
- Label providing a user friendly name
- Tags providing additional metadata
- Description providing a user friendly description
- Representation of all the components within the versioned flow at the time of creation/save/save-as
- Hash of the components within the versioned flow
- The creation date of the version

Once under version control, the versioned flow would be “saved” to create a new version, or “saved-as” to create a new versioned flow starting with the components of the original versioned flow, but no longer related to the original versioned flow.

As an example, if a process group with an id of “pg1” was placed under version control as “Bryan’s Flow” we would start with the following:

Process Group	**Label**	**Description**	**Version**	**Parent**
pg1	Bryan’s Flow	NA	v1	

After modifying and saving the versioned flow three times we would end up with the following:

Process Group	**Label**	**Description**	**Version**	**Parent**
pg1	Bryan’s Flow	NA	v1	
pg1	Bryan’s Flow	NA	v2	v1
pg1	Bryan’s Flow	NA	v3	v2
pg1	Bryan’s Flow	NA	v4	v3

If the versioned flow is then saved-as “Matt’s Flow” we would end up with the following:

Process Group	**Label**	**Description**	**Version**	**Parent**
pg1	Bryan’s Flow	NA	v1	
pg1	Bryan’s Flow	NA	v2	v1
pg1	Bryan’s Flow	NA	v3	v2
pg1	Bryan’s Flow	NA	v4	v3
pg1	*Matt’s Flow*	*NA*	*v1*	

“Matt’s Flow” would be a new versioned flow for “pg1” that started with the state of “Bryan’s Flow” at v3, but is now disconnected from “Bryan’s Flow”.

In order to successfully save a new version, the parent version of the new version would need to match the latest version with in the given label at the time of saving. If an attempt was made to save a version where the parent version was no longer the latest version, then this would be considered a conflict and the client would need to take action to resolve this conflict.

If we took the previous example where v1 of Matt’s flow exists, and two save operations were initiated that each have a parent of v1, one of the saves would be successful and create v2, and the other save would be considered a conflict because it’s parent of v1 would no longer be the latest version of “Matt’s Flow”.

Versioned Flows User Interaction

The end-user of Apache NiFi could interact with versioned flows in the following:

Action	Description
Define Flow Registries	The user could define flow registries by entering a URL of the REST API for a flow registry, along with a label.
Start Version Control	The user could start version control for a process group after it has been created. As part of initiating version control, the registry will be selected, and information such as the label, description, and tags will be entered.
Attach Version Control	The user could attach an existing versioned flow to an existing process group. As part of attaching version control, the flow registry will be selected.
Stop Version Control	The user could stop version control of a process group which disassociates the process group from the registry it came from.
Save /Snapshot	The user could save a new version through a right-click menu on the process group, available once under version control.
Save-As	The user could save-as a new versioned flow through a right-click menu on the process group, available once under version control.
Instantiate/Import	The user could instantiate a versioned flow by creating a new process group and selecting to import as a versioned flow from a given provider.
Upgrade In-Place	The user could upgrade a process group in-place to a new version by selecting an upgrade option from a right-click menu, and selecting the version to upgrade to. The new version would be compared to the existing version to determine if upgrading is possible, and if not the user will be notified of a conflict.

Versioned Flows User Scenarios

These scenarios assume there is an instance of the flow registry installed, and that a given NiFi instance has been configured to know about this flow registry.

1) Versioning of Flow for Rollback

1. User creates a process-group and builds a flow within the group
2. User starts version control for the given process-group, which includes:
    a. Selecting the flow registry for version control
    b. Entering a unique name and metadata for the versioned flow
3. Version 1 of the versioned flow is pushed to the flow registry
4. User makes changes to the flow and chooses to Save
5. Version 2 of the versioned flow is pushed to the flow registry
6. User decides version 2 is incorrect and selects to rollback to Version 1
7. The process group is now at Version 1

2) Deployment of Process-Group with In-Place Upgrade

1. Dev user creates a new process-group and builds a flow within the group
2. Dev user starts version control for the given process-group, which includes:
    a. Selecting the flow registry for version control
    b. Entering a unique name and metadata for the versioned flow
3. Version 1 of the versioned flow is pushed to the flow registry
4. Prod user creates a new process-group and chooses to import from versioned flow, which includes:
    a. Selecting the flow registry to import from
    b. Selecting Version 1 of the flow
5. Prod now contains the same process-group as Dev
6. Dev user makes changes to the process-group on Dev and chooses to Save
7. Version 2 of the versioned flow is pushed to the flow registry
8. Prod user selects to upgrade the Prod process-group to version 2
9. Version 2 is checked for compatibility with Prod version
10. If compatiblity check passes the upgrade is performed, otherwise the reason for conflict is presented to the Prod user

3) Deploy of Process-Group with Side-By-Side Upgrade

Starting from Step 7 of Scenario #2 (previous scenario):

1. Prod user wants to test Version 2 of the flow and creates a new process-group, choosing to import from Version 2
2. Prod now has two process-groups for the same versioned flow

    a. The first process-group is at Version 1
    b. The second process-group is at Version 2
3. Prod user tests Version 2
    a. If it functions as expected the second process-group can become the production group and the first process-group can be deleted, or the second process-group can be deleted and the first process-group can be safely upgraded
    b. If it doesn't function as expected, the second process-group can be deleted, and the first process-group is still at Version 1

4) Deployment of Root Group as the Root Group

1. Dev user builds a flow on the root canvas
2. Dev user starts version control for the root group, which includes:
    a. Selecting the flow registry for version control
    b. Entering a unique name and metadata for the versioned flow
3. Version 1 of the versioned flow is pushed to the flow registry
4. Prod user attaches the root group of Prod to version control, which includes:
    a. Selecting the flow registry to attach from
    b. Selecting Version 1 of the flow
5. Prod now contains the same root group as Dev
6. Proceeds the same from Step 6 of Scenario #2

5) Deployment of Root Group as a Sub-Group

1. Dev user builds a flow on the root canvas
2. Dev user starts version control for the root group, which includes:
    a. Selecting the flow registry for version control of the group
    b. Entering a unique name and metadata for the versioned flow
3. Version 1 of the versioned flow is pushed to the flow registry
4. Prod user creates a new process-group and chooses to import from versioned flow, which includes:
    a. Selecting the flow registry to import from
    b. Selecting Version 1 of the flow
5. Prod now contains the root group from Dev as a sub-group on Prod

NOTE: When importing a root group as a sub-group, or importing a sub-group to a root group, a conversion of ports would be performed. For example, a port on the root canvas is a remote port and would be converted to a local port when that group is imported as a sub-group. A port in a sub-group is a local port and would be converted to a remote port when imported to a root group.

Flow Registry Implementation

The following section describes details of how the flow registry could be implemented.

Project Structure

The source code for the standard flow registry could live in a new git repository called nifi-registry. This module would produce a binary artifact for the nifi-registry application which would initially include the nifi-flow-registry. The nifi assembly could bundle a released version of the nifi-registry to make it easy for users to obtain the nifi-registry.

Deploying the flow registry as a separate application would offer the following benefits:

- The flow registry would be easier to modify as it will not impact the core code paths in Apache NiFi.
- The flow registry would not be tied to the NiFi clustering model, thus simplifying the flow registry REST API.
- The flow registry could be deployed in various configurations without requiring a NiFi instance.

Extensibility

The standard flow registry would provide a pluggable interface for the persistence of versioned flows, referred to as the FlowPersistenceProvider (FPP). A default FPP would be provided that leverages the local filesystem for storage, but anyone could implement their own FPP to leverage a specific repository /database for persistence.

Security

The standard flow registry would provide the ability to run in secure and unsecure modes. The initial implementation may focus on an un-secure mode eventually introducing a security model in the flow registry application, allowing fine-grained access control to a versioned flow.

Versioned Flow Association in NiFi

When version control is started for a process group, NiFi would generate a flow snippet representing all of the components of that group, similar to a template. The components in that snippet would be assigned unique identifiers, different from their identifiers in the original components. These unique identifiers would then be associated with the original components to track which versioned component they originated from.

In the following example, let’s assume we have created a process group called “Bryan’s Flow” containing one processor. A simplified representation of this in the flow.xml.gz would look something like this:

```
<processGroup>

 <id>pg-1</id>

 <name>Bryan's Flow</name>

 <processor>

   <id>p-1</id>

   <name>UpdateAttribute</name>

   <class>org.apache.nifi.SomeProcessor</class>

 </processor>

</processGroup>
```

After starting version control for this process group, a snapshot of the process group would be saved to the flow registry with something like the following:

```
<processGroup>
 <id>pg-1-A</id>
 <name>Bryan's Flow</name>
 <processor>
   <id>p-1-A</id>
   <name>UpdateAttribute</name>
   <class>org.apache.nifi.SomeProcessor</class>
 </processor>
</processGroup>
```

The original process group would be updated to track the registry being used for version control, as well as the ids of the version controlled components:

```
<processGroup>
 <id>pg-1</id>
 <versionId>pg-1-A</versionId>
 <name>Bryan's Flow</name>
 <flowRegistry>http://localhost:7080/flow-registry</flowRegistry>
 <processor>
```

```
    <id>p-1</id>

    <versionId>p-1-A</versionId>

    <name>UpdateAttribute</name>

    <class>org.apache.nifi.SomeProcessor</class>

  </processor>

</processGroup>
```

The component ids in the snapshots must be consistent across all snapshots for a given versioned flow, meaning once the first snapshot is taken, the ids should remain the same in all subsequent snapshots.

The versionIds would be used during upgrades to tie a component on the canvas to the originating component in the snapshot of the versioned flow.

When starting a new process group and importing from a versioned flow, unique identifiers would be generated for the components, and the versionIds would be set to track the components from the versioned flow. This would allow for multiple instances of the same versioned flow to be on the same canvas.

Versioned Flow Sensitive Properties

Sensitive properties must be considered when storing versioned flows in a flow registry. Initially all sensitive property values could be scrubbed prior to saving to a flow registry, similar to how templates currently work.
In the future, versioned flows may be able to leverage future capabilities provided by the variable registry to provide a richer user-experience when parameterizing property values.

Assumptions

Requirements

#	Title	User Story	Importance	Notes
1	Provide a shared SDLC approach and management to both NiFi and MiNiFi environments	Users should be able to have common infrastructure to support the management of both NiFi and MiNiFi dataflows		
2				

User interaction and design

Questions

Below is a list of questions to be addressed as a result of this requirements document:

Question	Outcome

Not Doing