NiFi Project and Repository Restructuring

This wiki page is based on a discussion on the dev@nifi mailing list. The discussion thread that prompted this is here:

https://lists.apache.org/thread.html/939a7630a2e32594cd10444e48b7a1321fd9ce51834d911a8c04b6a9@%3Cdev.nifi.apache.org%3E

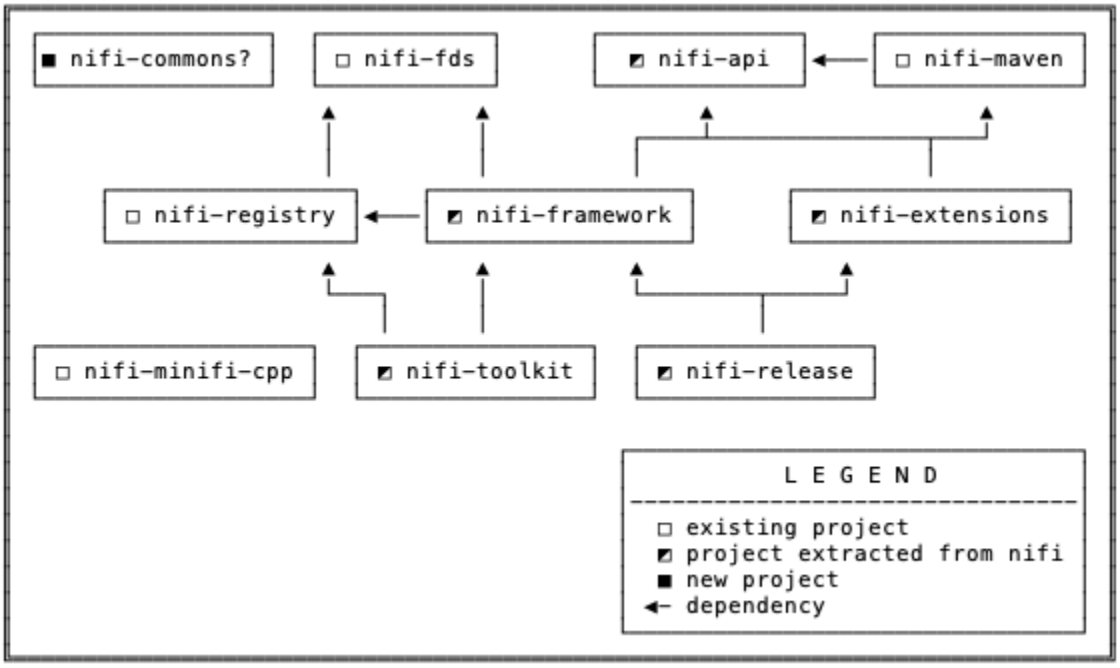
Table of Contents

Goals

- Faster build times for dev/testing/review/release
- Smaller release artifacts
- Maintain or increase the productivity of dev community (i.e., the new project structure should not be too burdensome for existing and future contributors to adopt)
- Facilitate code reuse across projects, such as nifi and nifi-registry, e.g., shared security provider implementations for authentication and authorization.
- Position the NiFi project to more easily support and take advantage of future Java language features, such as modules.
- Better separation of concerns & better defined interfaces / touch points between project components
    - Promote the idea that each module contains its own code and unit tests against public APIs such that it can be developed, tested, and released independently
    - Cross-module integration tests can exist outside of each module, in their own module that tests the integration of two or more modules

Proposed End State

Diagram of Repositories



Description of Repositories

The following repositories are proposed. Each repository would contain a top-level maven project that inherits from the `org.apache:apache` parent pom.

Project / Repo Name	New?	Dependencies	Description
`nifi-api`	new - extracted from nifi		Contains any Java APIs that need to be agreed upon by multiple top-level projects, such as nifi-framework and nifi-extensions
`nifi-framework`	new - extracted from nifi	nifi-api, nifi-maven, nifi-fds, nifi-registry, nifi-standard-libs	Contains the NiFi web server and front end without any unneeded flow extension bundles such as processors, controller services, and reporting tasks.
`nifi-extensions`	new - extracted from nifi	nifi-api, nifi-maven, nifi-standard-libs	Extension bundles for NiFi, such as processors, controller services, and reporting tasks.
`nifi-release`	new - extracted from nifi	nifi-framework, nifi-extensions	Will be the new home for the nifi-assembly that produces convenience binaries such as `nifi-X.Y.Z.zip`, and in the future alternate convenience binaries, for example, `nifi-slim-x.y.z.zip`. Eventually, this repository will also take over the current role of the nifi-minifi project/repository by providing the `nifi-minifi-X.Y.Z.zip` assembly, but this will require moving some additional modules from nifi-minifi into nifi-framework and nifi-extensions.
`nifi-toolkit`	new - extracted from nifi	nifi-framework, nifi-registry, nifi-standard-libs	Contains the nifi-toolkit source code and assembly
`nifi-maven`	existing	nifi-api	Contains the Maven plugin used for packaging NARs
`nifi-fds`	existing		NiFi Flow Design system - reusable front end components used to create consistent front ends, such as the nifi and nifi-registry web UIs
`nifi-registry`	existing	nifi-api, nifi-fds, nifi-standard-libs	A web service for centralized storage and versioning of NiFi flows and extensions
`nifi-minifi-cpp`	existing		A native implementation of libminifi and the MiNiFi agent

`nifi-standard-libs`	new - introduced later		Contains shared code / implementations, such as security provider implementations for authentication and authorization. Note that introducing a nifi-standard-libs project/repository is a long-term goal as part of the project structure roadmap and not part of the initial restructuring. The initial decomposition of nifi into new projects will not include any code changes, and therefore extracting shared code out of nifi-framework and nifi-registry into nifi-standard-libraries will happen in a future phase.

The following repositories are proposed to be archived. That is, frozen from farther changes once the new repositories are ready.

- `nifi`: will have been broken up into nifi-api, nifi-framework, nifi-extensions, nifi-release
    - Alternatively: this repo could be reduced to a single pom module containing the `org.apache.nifi:nifi` pom that inherits from `org.apache:apache` and can be a parent pom for other projects.
- `nifi-minifi`: will become a new assembly of nifi-framework and nifi-extensions that lives in the nifi-release repo

Benefits of Multiple Repositories

Multiple git repositories are not required in order to have multiple Maven projects. An alternative is a single git source code repository with multiple projects in sub directories. This was considered. Pros and cons of both approaches were also discussed on the discussion thread related to this proposal. This section of the proposal serves as a summary of that discussion and the thinking behind why this proposal recommends multiple repositories at this time.

Advantages of single repository:

- Easier to make cross-project changes, especially for new comers. Also for some, it will be easier to review changes that impact multiple projects if they are in a single PR.
- Git history for combined/entire project in one place
- Single focal point for entire community to collaborate on and follow
- Less infra to manage (though potentially mitigated now that access to repos is managed by ASF LDAP groups)

Advantages of multiple repositories:

- Easier for Release Mangers as they are performing source releases of an entire repository.
- Quicker to build an entire repository. Easier to setup reliable, fast CI builds. In a multi-repo setup, builds will never span multiple projects as commits are, by definition, isolated to a single repo. Also avoids having to "build all project" or setting up CI tools such as Travis or Jenkins to scope builds based on contents of commits.
- Separate repos forces pull requests to stay scoped to a given component, making for overall smaller PRs. Easier to look at a repo and see what work/contributions are still open.
- Easier to understand the history for a single project or module in the git history, e.g., "show me all changes to this project since the last tag / release"
- Lower learning curve for making changes to a single project, as the overall code base is smaller.

At this time (though it is still being discussed), the multiple repository approach seems to be merited, especially based on it being the optimized approach for managing source code releases.

To mitigate the downsides of multiple repositories, the following is recommended:

- Additional documentation and guidance be prepared for developers and contributors, especially getting started guides that target new comers. For example, document the project and module structure to help newcomers navigate repositories.
- Setup a CI job somewhere that publishes a SNAPSHOT build of every project's main branch, and allow development to depend on SNAPSHOT versions of dependencies. This will enable work to continue without releasing stable versions of dependencies. The CI infrastructure and where to host SNAPSHOT artifacts has not been identified. It is possible that GitHub Actions could be setup to run the CI jobs to publish SNAPSHOT artifacts to the Apache Maven repository managed by ASF infra.

A Note About Jira

This proposal only impacts source code projects and repositories. At this time, no change is being suggested for our Jira project structure or issue tracking system. It is recommended that even under different source code projects and repositories, issues are still tracked in the existing Jira project (i.e., NIFI) and differentiated as to which source code project/repo/artifact they impact via fields on the issue, such as component, label, fix version, etc.

An Alternative: Single project in a mono-repo

There are downsides to multiple projects, regardless of single or multiple repository. The main downside is development. If one project depends on another, and changes need to be made in the dependency, then the dependency must be updated, build, and installed as a SNAPSHOT in order for work to continue. This can be avoided by structuring the project in the other extreme, so that every component is part of one maven project. In order for this approach to be considered, someone would need to do exploratory work to understand how we could significantly speed up builds or facilitate partial builds of sub-modules automatically based on which files have been changed in a commit to the mono-repo.

Proposed Approach

Phase 1: nifi

Decompose nifi project/repo into:

- nifi-api
- nifi-framework
- nifi-extensions
- nifi-release
- nifi-toolkit

Assumptions

1. The project version for new projects being extracted from the existing nifi repository/project will match the current version that is set on the nifi project at the time of restructuring.
2. Every Maven module that currently exists in the nifi repository/project will be migrated to exactly one destination repository/project. By "exactly one", we mean that we do not anticipate the contents of any existing module will need to be split across multiple destinations.
3. No maven artifact coordinates will change for existing modules as part of the restructuring. That is, an existing artifact's `group:artifact:version` should not change even if it is in a new location, meaning modules that have dependencies on moved modules should not need to have their dependencies updates.

Steps

1. Create a mapping of existing modules in the nifi repository/project into where they will live in the new structure
2. Use utilities such as git filter-branch or git-subtree to move existing modules while retaining as much revision history as possible
3. Update the pom files for migrated modules to contain their new parents, etc.
4. Each new repository/project will get a new assembly module that just packages that artifact so that releases can be done from that project repository
5. Write new assemblies in the nifi-release project/repository that produces something that closely matches the existing assembly
6. Get all builds and tests working, although this should be minimal as no maven module coordinates are changing.

Process

1. One or more community volunteers will manually attempt the above steps, taking detailed notes of commands they use and changes that they need to make in order to get everything working, such as modifications to pom files.
2. From those notes, a script will be made that streamlines (ideally, fully automated) the restructuring the nifi repository into new repositories. This will enable us to repeatedly perform the restructuring from any given revision of the main branch.
3. The restructuring automation script(s) and instructions for use will be distributed on the Apache NiFi dev@ mailing list. The NiFi dev community will be asked to verify the procedure and resulting output in a manner similar to a RC verification vote. The final output assembly of the restructured process should match the output assembly of the current nifi-assembly module from the 1.x line on the main branch today.
4. The automation process will go through as many reviews as needed until it passes a vote by the NiFi PMC.
5. A specified date and time will be chosen for the restructuring. Any large branches from main should be merged before that date to avoid burdensome rebasing that will be necessary for any changes not merged before the restructuring.
6. On the agreed upon date and time, with the assistance of ASF Infra, the existing nifi repository will be frozen from modification. The restructuring script will be run and the resulting repositories will be added as top level Github repositories, again, with the help of ASF Infra.
7. Unmerged pull requests to the nifi repo that were under active development will need to be recreated as PRs to the new repositories. This responsibility will fall to the author of each PR. PRs will stay open to the old nifi repository for a time as a historical record, but only those that are recreated against the new repositories by their authors will be considered to be merged.
8. Contributions continue as normal, but from forks of the new repositories.
9. The first release after the restructuring will be a release from all repositories/projects in the order dictated by the dependency graph shown above. After that, releases will only need to be performed as needed from each repository/project.
    - NOTE: There should be a community discussion on how versioning and releasing of the new, smaller projects should be handled. That is, should we always release everything so that versions stay aligned, even if there are not changes to a particular component such as nifi-api, which changes very infrequently, or should we follow semantic versioning for each new project and allow them to advance and be released independently of each other and only as needed.

Phase 2: nifi-standard-libs

Introduce nifi-standard-libs as a home for shared, top-level, reusable libraries of components across the new collection of projects

Steps

1. A new nifi-standard-libs project and repository is introduced.
2. Similar code across nifi-framework and nifi-registry, and possibly other projects such as nifi-toolkit and nifi-extensions, is rewritten as generic, reusable libraries in sub-modules of nifi-standard-libs, e.g., nifi-standard-libs-security could contain shared authentication and authorization APIs and provider implementations.
3. Code from nifi-framework, nif-registry, etc. is removed and replaced with library implementations from nifi-standard-libs modules.

Phase 3: nifi-minifi

Migrate code modules from nifi-minifi into nifi-framework, nifi-extensions, and (possibly) nifi-standard-libs. Migrate the nifi-minifi assembly into nifi-release.

Additional Potential Phases

Once the initial restructuring and decomposition of the nifi project/repo is complete, there may be opportunities for farther steps such as breaking nifi-extensions into smaller projects, or dropping support for some legacy extensions.