How to Contribute

Welcome contributors! We strive to include everyone's contributions. This page provides necessary guidelines on how to contribute effectively towards furthering the development and evolution of Flume. You should also read the guide on setting up Development Environment where you will find details on how to checkout, build and test Flume.

Note: This guide applies to general contributors. If you are a committer, please read the How to Commit as well.

What can be contributed?

There are many ways you can contribute towards the project. A few of these are:

Jump in on discussions: It is possible that someone initiates a thread on the mailing list describing a problem that you have dealt with in the past. You can help the project by chiming in on that thread and guiding that user to overcome or workaround that problem or limitation.

File Bugs: If you notice a problem and are sure it is a bug, then go ahead and file a JIRA. If however, you are not very sure that it is a bug, you should first confirm it by discussing it on the Mailing Lists.

Review Code: If you see that a JIRA ticket has a "Patch Available" status, go ahead and review it. It cannot be stressed enough that **you must be kind in your review** and explain the rationale for your feedback and suggestions. Also note that not all review feedback is accepted - often times it is a compromise between the contributor and reviewer. If you are happy with the change and do not spot any major issues, then +1 it. More information on this is available in the following sections.

Provide Patches: We encourage you to assign the relevant JIRA issue to yourself and supply a patch for it. The patch you provide can be **code**, **documentation**, **build changes**, or any combination of these. More information on this is available in the following sections.

Providing Patches

In order to provide patches, follow these guidelines:

- **Make sure there is a JIRA**:
    1. If you are working on fixing a problem that already has an associated JIRA, then go ahead and assign it to yourself.
    2. If it is already assigned to someone else, check with the current assignee before moving it over to your queue.
    3. If the current assignee has already worked out some part of the fix, suggest that you can take that change over from them and complete the remaining parts.

- **Attach the patches as you go through development**:
    - While small fixes are easily done in a single patch, it is preferable that you create a GitHub pull request and update it as needed. This serves as an early feedback mechanism where interested folks can look it over and suggest changes where necessary. It also ensures that if for some reason you are not able to find the time to complete the change, someone else can take up your initial patches and drive them to completion.

- **Before you submit your patch**:
    1. Your change should be well-formatted and readable. Please use two spaces for indentation (no tabs).
    2. Carefully consider whether you have handled all boundary conditions and have provided sufficiently defensive code where necessary.
    3. Add one or more unit tests, if your change is not covered by existing automated tests.
    4. Insert javadocs and code comments where appropriate.
    5. Update the Flume User Guide (source) if your change affects the Flume config file or any user interface. Include those changes in your patch.
    6. Make sure you update the relevant developer documentation, wiki pages, etc. if your change affects the development environment.

- **Test your changes before submitting a review**:
    - Before you make the JIRA status as "Patch Available", please test your changes thoroughly. Try any new feature or fix out for yourself, and make sure that it works.
    - Make sure that all unit/integration tests are passing, and that the functionality you have worked on is tested through existing or new tests.
    - You can run all the tests by going to the root level of the source tree and typing `mvn clean install`.

- **Submitting your patch for review:**
    1. To submit a patch create a pull request at GitHub. Make sure to reference the Jira issue in the PR title.
    2. Add requested reviewers to the PR when it is ready for a review.

- **Work with reviewers to get your change fleshed out**:
    1. When your change is reviewed, please engage with the reviewer via JIRA or the pull request to get necessary clarifications and work out other details.
    2. The goal is to ensure that the final state of your change is acceptable to the reviewer so that they can +1 it.

Reviewing Code

Flume uses GitHub pull requests for doing code reviews. Feel free to comment on the JIRA requesting the assignee to create a pull request..

Goals for Code Reviews

The net outcome from the review should be the same - which is to ensure the following:

- Bugs/Omissions/Regressions are caught before the change is committed to the source control.
- The change is subjected to keeping the quality of code high so as to make the overall system sustainable. The implementation of the change should be easily readable, documented where necessary, and must favor simplicity of implementation.
- Changes are evaluated from the perspective of a consumer (the reviewer) as opposed to the developer, which often brings out subtleties in the implementation that otherwise go unnoticed.
- The change should be backward compatible and not require extensive work on existing installations in order for it to be consumed. There are exceptions to this in some cases like when work is done on a major release, but otherwise backward compatibility should be upheld at all times. If you are not clear, raise it is as a concern to be clarified during the review.

Code review guidelines

Following are some guidelines on how to do a code review. You may use any other approach instead as long as the above stated goals are met. That said, here is an approach that works fine generally:

- **Understand the problem being solved**: This often requires going through the JIRA comments and/or mailing list threads where the discussion around the problem has happened in the past. Look for key aspects of the problem such as how it has impacted the users and what, if any, is the suggested way to solve it. You may not find enough information regarding the problem in some cases, in which case - feel free to ask for clarification from the developer contributing the change.

- **Think about how you would solve the problem**: There are many ways to solve any code problem, with different ways having different merits. Before proceeding to review the change, think through how you would solve the problem if you were the one implementing the solution. Note the various aspects of the problem that your solution might have. Some such aspects to think about are - impact on backward compatibility, overall usability of the system, any impact on performance etc.

- **Evaluate the proposed change in contrast to your solution**: Unless the change is obvious, it is likely that the implementation of the change you are reviewing is very different from the solution you would go for. Evaluate this change on the various aspects that you evaluated your solution on in the previous step. See how it measures up and give feedback where you think it could be improved.

- **Look for typical pitfalls**: Read through the implementation to see if: it needs to be documented at places where the intention is not clear; if all the boundary conditions are being addressed; if the code is defensive enough; if any bad idioms have leaked in such as double check locking etc. In short, check for things that a developer is likely to miss in their own code which are otherwise obvious to someone trying to read and understand the code.

- **See if the change is complete**: Check if the change is such that it affects the user interface. If it does, then the documentation should likely be updated. What about testing - does it have enough test coverage or not? What about other aspects like license headers, copyright statements etc. How about checkstyle and findbugs - did they generate new warnings? How about compiler warnings?

- **Test the change**: It is very easy to test the change if you have the development environment setup. Run as many tests as you want with the patch. Manually test the change for functionality that you think is not fully covered via the associated tests. If you find a problem, report it.

How to give feedback

Once you have collected your comments/concerns/feedback you need to send it to back to the contributor. In doing so, please be as courteous as possible and ensure the following:

- Your feedback should be clear and actionable. Giving subjective/vague feedback does not add any value or facilitate a constructive dialog.
- Where possible, suggest how your concern can be addressed. For example if your testing revealed that a certain use-case is not satisfied, it is acceptable to state that as is, but it would be even better if you could suggest how the developer can address it. Present your suggestion as a possible solution rather than *the* solution.
- If you do not understand part of the change, or for some reason were not able to review part of the change, state it explicitly so as to encourage other reviewers to jump in and help.

Once you have provided your feedback, wait for the developer to respond. It is possible that the developer may need further clarification on your feedback, in which case you should promptly provide it where necessary. In general, the dialog between the reviewer and developer should lead to finding a reasonable middle ground where key concerns are satisfied and the goals of the review have been met.

If a change has met all your criteria for review, please `+1` the change to indicate that you are happy with it.