Contributing Code Changes

This page documents the various steps required in order to contribute Kafka code changes. This page should be read after Kafka's Contributing page.

There may be bugs or possible improvements to this page, so help us improve it. Credit to the Spark project for tackling the issue of receiving contributions to an Apache project via GitHub pull requests. We have borrowed liberally from their process, tools, and documentation.

When you contribute code, you affirm that the contribution is your original work and that you license the work to the project under the project's open source license. Whether or not you state this explicitly, by submitting any copyrighted material via pull request, email, or other means you agree to license the material under the project's open source license and warrant that you have the legal authority to do so.

Overview

Generally, Kafka uses:

- JIRA to track logical issues, including bugs and improvements
- Kafka Improvement Proposals for planning major changes
- Confluence for documentation
- Github pull requests to manage the review and merge of specific code changes

That is, JIRA and Confluence are used to describe what should be fixed or changed, and high-level approaches, and pull requests describe how to implement that change in the project's source code.

JIRA

1. Find the existing Kafka JIRA ticket that the change pertains to.
    a. Do not create a new JIRA ticket if creating a change to address an existing ticket in JIRA; add to the existing discussion and work instead.
    b. To avoid conflicts, assign the JIRA ticket to yourself if you plan to work on it.
    c. Look for existing pull requests that are linked from the JIRA ticket, to understand if someone is already working on it.
2. If the change is new, then it usually needs a new JIRA ticket. However, trivial changes, where "what should change" is virtually the same as "how it should change" do not require a JIRA ticket. Example: "Fix typos in Foo scaladoc"
3. If required, create a new JIRA ticket (below shows some critical fields to fill-in, a more detailed guidance can be found here):
    a. Provide a descriptive Title. "Update web UI" or "Problem in scheduler" is not sufficient. "Kafka support fails to handle empty queue during shutdown" is good.
    b. Write a detailed Description. For bug reports, this should ideally include a short reproduction of the problem. For new features, it may include a design document (or a Kafka Improvement Proposal if it's a major change).
    c. Set required fields: Type, Priority, and optionally Labels. Please only set Fix Version if you are confident you and committers agree that the issue needs to be fixed by the specified version. Please note that issues considered critical by one organization are not always considered universally critical.
    d. To avoid conflicts, assign the JIRA ticket to yourself if you plan to work on it. Leave it unassigned otherwise.
    e. Do not include a patch file; pull requests are used to propose the actual change.
4. If the change is a large change, consider inviting discussion on the issue at dev@kafka.apache.org first before proceeding to implement the change. Note that changes that modify APIs or that are very visible to users will also require following KIP (Kafka Improvement Proposal) process.

Pull Request

1. Fork the Github repository at http://github.com/apache/kafka if you haven't already
2. Clone your fork, create a new branch, push commits to the branch (review the Kafka Coding Guidelines, if you haven't already).
3. Consider whether documentation or tests need to be added or updated as part of the change, and add them as needed (doc changes should be submitted along with code change in the same PR).
4. Run all tests as described in the project's README.
5. Open a pull request against the trunk branch of apache/kafka. (Only in special cases would the PR be opened against other branches.)
    a. The PR title should usually be of the form `KAFKA-xxxx: Title`, where `KAFKA-xxxx` is the relevant JIRA id and `Title` may be the JIRA's title or a more specific title describing the PR itself. For trivial cases where a JIRA is not required (see JIRA section for more details) `MINOR:` or `HOTFIX:` can be used as the PR title prefix.
    b. If the pull request is still a work in progress, and so is not ready to be merged, but needs to be pushed to Github to facilitate review, then add `[WIP]` after the JIRA id.
    c. Consider identifying committers or other contributors who have worked on the code being changed. The easiest is to simply follow GitHub's automatic suggestions. You can add `@username` in the PR description to ping them immediately.
    d. Please state that the contribution is your original work and that you license the work to the project under the project's open source license.
6. A comment with information about the pull request will be added to the JIRA ticket.
7. Change the status of the JIRA to "Patch Available" if it's ready for review. To do this, click the "Submit Patch" button in JIRA, and then in the resulting dialog, click "Submit Patch".

8. The project uses Apache Jenkins for continuous testing on Linux AMD64 and ARM64 build nodes. A CI job will be started automatically for a new pull request, and rerun each time a new commit is pushed. If you need to trigger a new run for any reason you can either tag a committer to trigger a new build or just push a new commit.
9. Once ready, the PR `checks` box will be updated with the test results along with links to the full results on Jenkins (we have separate builds in order to test multiple Java and Scala versions).
10. Investigate and fix failures caused by the pull the request
    a. Fixes can simply be pushed to the same branch from which you opened your pull request.
    b. Please address feedback via additional commits instead of amending existing commits. This makes it easier for the reviewers to know what has changed since the last review. All commits will be squashed into a single one by the committer via GitHub's squash button or by a script as part of the merge process.
    c. Jenkins will automatically re-test when new commits are pushed.
    d. Despite our efforts, Kafka may have flaky tests at any given point, which may cause a build to fail. You need to ping committers to trigger a new build. If the failure is unrelated to your pull request and you have been able to run the tests locally successfully, please mention it in the pull request.
11. In addition to unit and integration tests, we also have a set of system tests that run on a nightly basis. For large, impactful or risky changes, it is preferable to run the system tests before merging the pull request. Currently that can be done via a manually triggered job in a Jenkins instance provided by Confluent (ask for access in the pull request).

The Review Process

- Other reviewers, including committers, may comment on the changes and suggest modifications. Changes can be added by simply pushing more commits to the same branch.
- Please add a comment and "@" the reviewer in the PR if you have addressed reviewers' comments. Even though GitHub sends notifications when new commits are pushed, it is helpful to know that the PR is ready for review once again.
- Patches can be applied locally following the comments on the JIRA ticket, for example: git pull https://github.com/[contributor-name]/kafka KAFKA-xxxx.
- Lively, polite, rapid technical debate is encouraged from everyone in the community. The outcome may be a rejection of the entire change.
- Reviewers can indicate that a change looks suitable for merging by approving it via GitHub's review interface. This indicates the strongest level of technical sign-off on a patch and it means: "I've looked at this thoroughly and take as much ownership as if I wrote the patch myself". If you approve a pull request, you will be expected to help with bugs or follow-up issues on the patch. Consistent, judicious use of pull request approvals is a great way to gain credibility as a reviewer with the broader community. Kafka reviewers will typically include the acronym LGTM in their approval comment. This was the convention used to approve pull requests before the "approve" feature was introduced by GitHub.
- Sometimes, other changes will be merged which conflict with your pull request's changes. The PR can't be merged until the conflict is resolved. This can be resolved with "git fetch origin" followed by "git merge origin/trunk" and resolving the conflicts by hand, then pushing the result to your branch.
- Try to be responsive to the discussion rather than let days pass between replies.

Closing Your Pull Request / JIRA

- If a change is accepted, it will be merged and the pull request will automatically be closed, along with the associated JIRA if any
    - Note that in the rare case you are asked to open a pull request against a branch besides trunk, that you will actually have to close the pull request manually
    - The JIRA will be Assigned to the primary contributor to the change as a way of giving credit. If the JIRA isn't closed and/or Assigned promptly, comment on the JIRA.
- If your pull request is ultimately rejected, please close it.
- If a pull request has gotten little or no attention, consider improving the description or the change itself and ping likely reviewers again after a few days. Consider proposing a change that's easier to include, like a smaller and/or less invasive change.
- If a pull request is closed because it is deemed not the right approach to resolve a JIRA, then leave the JIRA open. However if the review makes it clear that the issue identified in the JIRA is not going to be resolved by any pull request (not a problem, won't fix) then also resolve the JIRA.