How to Contribute

Contributing code changes

1. Checkout source code

- Fork the project from Github at https://github.com/apache/ambari if you haven't already
- Clone this fork:

```
# Replace [forked-repository-url] with your git clone url
git clone [forked-repository-url] ambari
```

- Set upstream remote:

```
cd ambari
git remote add upstream https://github.com/apache/ambari.git
```

2. Keep your Fork Up to Date

```
# Fetch from upstream remote
git fetch upstream
# Checkout the branch that needs to sync
git checkout trunk
# Merge with remote
git merge upstream/trunk
```

Repeat these steps for all the branches that needs to be synced with the remote.

3. JIRA

Apache Ambari uses JIRA to track issues including bugs and improvements, and uses Github pull requests to manage code reviews and code merges. Major design changes are discussed in JIRA and implementation changes are discussed in pull requests after a pull request is created.

a. Find an existing Apache JIRA that the change pertains to
   i. Do not create a new JIRA if the change is minor and relates to an existing JIRA; add to the existing discussion and work instead
   ii. Look for existing pull requests that are linked from the JIRA, to understand if someone is already working on the JIRA

b. If the change is new, then create a new JIRA:
   i. Provide a descriptive Title
   ii. 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.
   iii. Fill the required fields:
      1. **Issue Type**. Bug and Task are the most frequently used issue types in Ambari.
      2. **Priority**. Their meaning is roughly:
         a. Blocker: pointless to release without this change as the release would be unusable to a large minority of users
         b. Critical: a large minority of users are missing important functionality without this, and/or a workaround is difficult
         c. Major: a small minority of users are missing important functionality without this, and there is a workaround
         d. Minor: a niche use case is missing some support, but it does not affect usage or is easily worked around
         e. Trivial: a nice-to-have change but unlikely to be any problem in practice otherwise
      3. **Component**. Choose the components that are affected by this change. Choose from Ambari Components
      4. **Affects Version**. For Bugs, assign at least one version that is known to exhibit the problem or need the change
   iv. Do not include a patch file; pull requests are used to propose the actual change.

4. Pull Request

Apache Ambari uses Github pull requests to review and merge changes to the source code. Before creating a pull request, one must have a fork of apache/ambari checked out. Follow instructions in step 1 to create a fork if you haven't already.

a. Commit and Push changes

i. Create a branch **AMBARI-XXXXX-branchName** before starting to make any code changes. Ex: If the Fix Version of the JIRA you are working on is 2.6.2, then create a branch based on branch-2.6

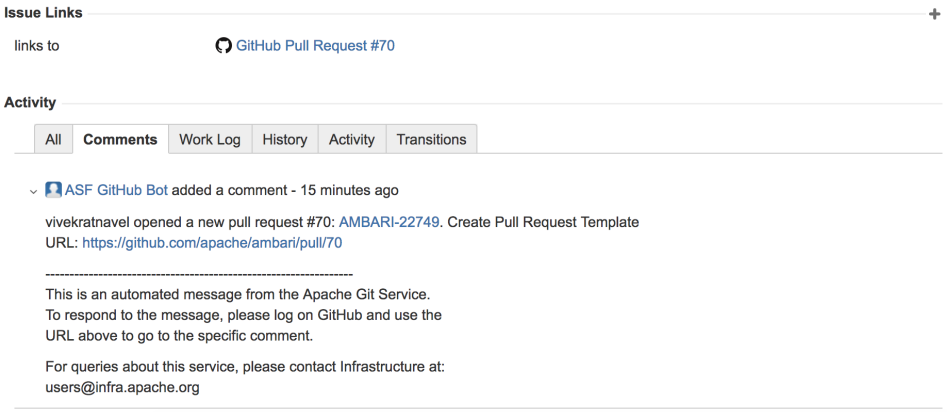
```
git checkout branch-2.6
git pull upstream branch-2.6
git checkout -b AMBARI-XXXXX-branch-2.6
```

ii. Mark the status of the related JIRA as "In Progress" to let others know that you have started working on the JIRA.
iii. Make changes to the code and commit them to the newly created branch.
iv. Run all the tests that are applicable and make sure that all unit tests pass
v. Push your changes. Provide your Github user id and personal access token when asked for user name and *password*

```
git push origin AMBARI-XXXXX-branch-2.6
```

b. Create Pull Request

i. Navigate to your fork in Github and create a pull request. The pull request needs to be opened against the branch you want the patch to land.
   1. The pull request title should be of the form **[AMBARI-xxxx] Title**, where AMBARI-xxxx is the relevant JIRA number
   2. 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 **AMBARI-XXXX**
   3. Consider identifying committers or other contributors who have worked on the code being changed. Find the file(s) in Github and click “Blame” to see a line-by-line annotation of who changed the code last. You can add @username in the PR description or as a comment to request review from a developer.
      Note: Contributors do not have access to edit or add reviewers in the "Reviewers" widget. Contributors can only @mention to get the attention of committers.
   4. The related JIRA will automatically have a link to the PR as shown below. Mark the status of JIRA as "Patch Available" manually.

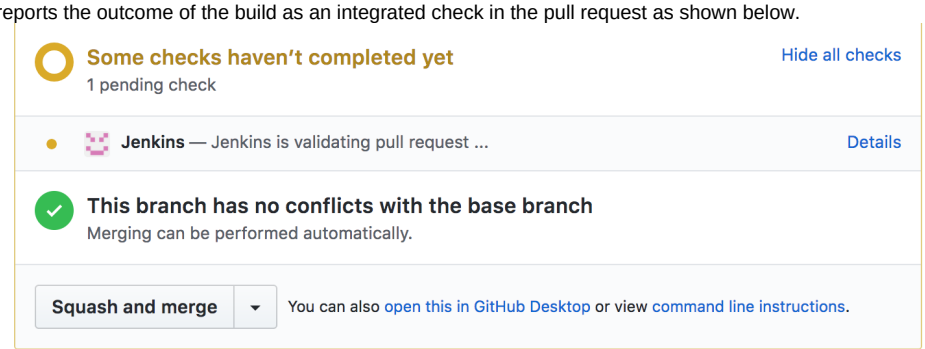


c. Jenkins Job

i. A Jenkins Job is configured to be triggered everytime a new pull request is created. The job is configured to perform the following tasks:

   1. Validate the merge
   2. Build Ambari
   3. Run unit tests

ii. It reports the outcome of the build as an integrated check in the pull request as shown below.



iii. It is the responsibility of the contributor of the pull request to make sure that the build passes. Pull requests should not be merged if the Jenkins job fails to validate the merge.
iv. To re-trigger the build job, just comment "retest this please" in the PR. Visit this page to check the latest build jobs.

d. Repeat the above steps for patches that needs to land in multiple branches. Ex: If a patch needs to be committed to branches branch-2.6 and trunk, then you need to create two branches and open two pull requests by following the above steps.

Review Process

Ambari uses Github for code reviews. All committers are required to follow the instructions in this page and link their github accounts with gitbox to gain Merge access to apache/ambari in github.

To try out the changes locally, you can checkout the pull request locally by following the instructions in this guide.

1. Other reviewers, including committers can try out the changes locally and either approve or give their comments as suggestions on the pull request by submitting a review on the pull request. More help can be found here.
2. If more changes are required, reviewers are encouraged to leave their comments on the lines of code that require changes. The author of the pull request can then update the code and push another commit to the same branch to update the pull request and notify the committers.
3. The pull request can be merged if atleast one committer has approved it or commented "LGTM" which means "Looks Good To Me" and the jenkins job validated the merge successfully. If you comment LGTM you will be expected to help with bugs or follow-up issues on the patch. (Remember committers cannot review their own patch. If a committer opens a PR, they should make sure that another committer reviews it.)
4. Sometimes, other changes might be merged which conflict with the pull request's changes. The PR can't be merged until the conflict is resolved. This can be resolved by running **git fetch upstream** followed by **git rebase upstream/[branch-name]** and resolving the conflicts by hand, then pushing the result to your branch.
5. If a PR is merged, promptly close the PR and resolve the JIRA as "Fixed".

Apache Ambari Committers

Please read more on Apache Committers at: http://www.apache.org/dev/committers.html

In general a contributor that makes sustained, welcome contributions to the project may be invited to become a committer, though the exact timing of such invitations depends on many factors. Sustained contributions over 6 months is a welcome sign of contributor showing interest in the project. A contributor receptive to feedback and following development guidelines stated above is a good sign for being a committer to the project. We have seen contributors contributing 20-30 patches become committers but again this is very subjective and can vary on the patches submitted to the project. Ultimately it is the Ambari PMC that suggests and votes for committers in the project.