HowToContribute

Overview

1. Pick an issue to work on. If you don't have a specific itch to scratch, some possibilities are marked with the low-hanging fruit label in JIRA. 2. Read the relevant parts of Architecture Internals; other resources include the following videos: DataStax Deep Dives into Apache Cassandra internals, Planet Cassandra videos such as this Summit 2013 talk, and Cassandra user group videos like this one from Austin Cassandra. 3. Check if someone else has already created an ticket for the change you have in mind in the issue tracker. 4. If not, create a ticket describing the change you're proposing in the issue tracker. 5. When you're ready to start working on the ticket, if it has not yet been assigned, assign the ticket to yourself in JIRA. This is done using the "Assign" button at the top of the ticket. 6. Clone the latest version of the source code:
    - `git clone http://git-wip-us.apache.org/repos/asf/cassandra.git cassandra`
      You'll want to checkout out the branch corresponding to the lowest version in which you want to introduce the change. For example,
    - `git checkout cassandra-3.0`
      From there, create a branch for your changes. Many contributors name their branches based on ticket number and Cassandra version.
    - `git checkout -b 12345-3.0`
      7. Modify the source to include the improvement/bugfix
    - Verify that you follow Cassandra's CodeStyle.
    - Verify that your change works by adding a unit test.
    - Make sure all tests pass using the commands below. If you suspect a test failure is unrelated to your change, it may be useful to check the test's status by searching the issue tracker or looking at CI results for the relevant upstream version.
    - For testing multi-node behavior, CCM, a tool to easily create local clusters, is useful.
    - Consider going through the Review Checklist for your code. This will help you to understand how others will consider your change for inclusion.
      8. When you're happy with the result, create a patch:
    - `git add <any new or modified file>`
    - `git commit -m '<message>'`
    - `git format-patch`
    - `mv <patch-file> <ticket-branchname.txt> (e.g. 12345-trunk.txt, 12345-3.0.txt)`
      Alternatively, many contributors prefer to make their branch available on GitHub. In this case, fork the Cassandra repository on GitHub and push your branch:
    - `git push --set-upstream origin 12345-3.0`
      9. To make life easier for your reviewer/committer, you may want to make sure your patch applies cleanly to later branches and create additional patches/branches for later Cassandra versions to which your original patch does not apply cleanly. That said, this is not critical, and you will receive feedback on your patch regardless.
      10. Attach the newly generated patch to the ticket/add a link to your branch and click "Submit Patch" at the top of the ticket. This will move the ticket into "Patch Available" status, indicating that your submission is ready for review.
      11. Wait for other developers or committers to review it and hopefully +1 the ticket (see How To Review). If your change does not receive a +1, do not be discouraged. If possible, the reviewer will give suggestions to improve your patch or explain why it is not suitable.
      12. If the reviewer has given feedback to improve the patch, make the necessary changes and move the ticket into "Patch Available" once again.
      13. Once the review process is complete, you will receive a +1. Wait for a committer to commit it.

Testing and Coverage

There are two major sets of tests for Cassandra: unit tests and dtests. The unit tests are part of the Cassandra repository; dtests are functional tests that are available at https://github.com/riptano/cassandra-dtest.

Running the Unit Tests

Run `ant test` from the top-level directory of your Cassandra checkout to run all unit tests. To run a specific test class, run `ant -Dtest.name=<ClassName>`. In this case, ClassName should not be fully qualified. For example, it might be `StorageProxyTest`. To run a specific test method, run `ant testsome -Dtest.name=<ClassName> -Dtest.methods=<comma-separated list of method names>`. When using the `testsome` command, ClassName should be a fully qualified name (like `org.apache.cassandra.service.StorageProxyTest`).

You can also run tests in parallel: `ant test -Dtest.runners=4`.

Running the dtests

The dtests use CCM to test a local cluster. If the following instructions don't seem to work, you may find more current instructions in the dtest repository.

1. Install CCM. You can do this with pip by running `pip install ccm`.
2. Install nosetests. With pip, this is `pip install nose`.
3. Install Cassandra Python driver. This can be installed using `pip install git+git://github.com/datastax/python-driver@cassandra-test`. This installs a dedicated test branch driver for new features. If you're working primarily on older versions, `pip install cassandra-test` may be sufficient.
4. Clone the dtest repository:
    - `git clone https://github.com/riptano/cassandra-dtest.git cassandra-dtest`.
5. Set `$CASSANDRA_DIR` to the location of your cassandra checkout. For example: `export CASSANDRA_DIR=/home/joe/cassandra`. Make sure you've already built Cassandra in this directory. You can build Cassandra by running `ant`.
6. Run all tests by running `nosetests` from the dtest checkout. You can run a specific module like so: `nosetests cql_tests.py`. You can run a specific test method like this: `nosetests cql_tests.py:TestCQL.counters_test`.
    - If you encounter any failures, you can confirm whether or not they exist in upstream branches by checking to see if the failing tests or test classes are tagged with the `known_failure` decorator. This decorator is documented inline in the dtests. If a test that is known to

fail passes, or a test that is not known to fail succeeds, you should check the linked JIRA ticket to see if you've introduced any detrimental changes to that branch.

Running the code coverage task

1. Run a basic coverage report of unit tests using `ant codecoverage`.
2. Alternatively, run any test task with `ant jacoco-run -Dtaskname=some_test_taskname`. Run more test tasks in this fashion to push more coverage data onto the report in progress. Then manually build the report with `ant jacoco-report` (the `codecoverage` task shown above does this automatically).
3. View the report at `build/jacoco/index.html`.
4. When done, clean up JaCoCo data so it doesn't confuse your next coverage report: `ant jacoco-cleanup`.

Continuous integration

Jenkins runs the Cassandra tests continuously: http://cassci.datastax.com/. For frequent contributors, this Jenkins is set up to build branches from their GitHub repositories. It is likely that your reviewer will use this Jenkins instance to run tests for your patch.

IDE

Most Cassandra developers use an IDE. Instructions to use IntelliJ IDEA or Eclipse for Cassandra development are available.

- RunningCassandraInIDEA
- RunningCassandraInEclipse

Committing

Got commit access? Outstanding! Here are the conventions we follow.

Commit messages take the form of

```
<explanation>

patch by <author>; reviewed by <committer> for CASSANDRA-<ticket>
```

When committing to multiple branches, start with the most-stable and merge forwards. For instance, if you had a fix to apply to 1.1, 1.2, and trunk, you would first commit to 1.1, and push changes. Then, switch to your 1.2 branch by doing

```
    git checkout cassandra-1.2
```

- and run

```
    git merge cassandra-1.1
```

If there are conflicts, resolve them and commit, followed by a push. Finally, switch to trunk by doing

```
    git checkout trunk
```

and run

```
    git merge cassandra-1.2
```

again resolve conflicts if the exist, and commit and push.

See http://www.youtube.com/watch?v=AJ-CpGsCpM0 for an in-depth explanation of why fixes should be merged forwards from more-stable branches, rather than backported from trunk.

This workflow also makes it so git knows what commits have been made to earlier branches but not to trunk: if you forget to merge a fix immediately, the next time someone goes to merge from the branch, git will incorporate the forgotten ones too.

Bundled Drivers

A copy of the Python driver is included for use in `cqlsh`. For instructions on how to package the bundled driver for the Cassandra project, see the instructions here.

https://c.statcounter.com/9397521/0/fe557aad/1/|stats