Contributing to Mesos: Where to Begin

Joris Van Remoortere
Michael Park
Moris!
Patch for Maintenance Primitives!

Yay! Shepherd?
You...? Review & Commit by EOD?

Uh... ok? Maintenance what? Design doc?
Next Day...
JPark! Patch for Maintenance Primitives!

Cool! No shepherd yet? Biiig Feature! Community Consensus... riiight?
Who should see patch?

1. Find Shepherd.
2. Design Doc.
4. Patches.
But my patches work!?! ..

Let’s find a shepherd on the dev mailing list first :-)
Getting Started
Tools

- IRC (#mesos on freenode)
- Mailing Lists (builds, commits, dev, issues, reviews, user)
- JIRA
- Reviewboard
- Git
Finding a Task

465 Tasks accepted in the Backlog column of [Apache Mesos Agile Board](https://example.com)
Look for tickets marked “newbie”
Participate in the conversations in IRC, mailing lists to gauge interest in tasks
What’s Involved?

Trivial Changes
1. Patches

Small Features / Bug Fixes
1. JIRA Ticket
2. Record of Design Discussion / Decision
3. Patches

Large Features
1. JIRA Ticket
2. Design Document
3. Community Consensus
4. Patches
Who’s Involved?

- Contributor (You!)
- Reviewer
- Shepherd
Reviewer: Who Are They?

Someone who can help you directly review the patches

- Typically, a senior contributor
- There should be a design decision that the patch can be reviewed against
- Assists in areas such as:
  - Adhering to the design decision
  - Adhering to the Mesos coding style
  - Sharing knowledge of existing libraries to simplify the code
Shepherd: Who Are They?

Someone who can commit the patch for you

- A Mesos committer
- Validate failure scenarios
- Ensures project vision is maintained

They’re people too!

Generally, an organization pays them to advance specific components of Mesos
Finding a Reviewer + Shepherd

The list of [Committers and Maintainers](#)

Gauge by asking for a shepherd on the dev mailing list

Build a relationship/trust with a shepherd

- Ping them on IRC or Hangouts
- Introduce yourself at a meetup/conference
- Convince them that your work is aligned with their goals
- Show history of driving tasks to completion and being responsive
Writing Patches
Preparation

An agreed-upon concrete design specification
Style Guides

- **Documentation**
  - C++
  - Doxygen
  - Markdown
Tests

Writing:
- googletest and googlemock
- Documentation of intended behavior
- Demonstrates correctness and reliability

Running:
- make check -j <N>
- make check -j <N> GTEST_FILTER="MyFeatureTest.*"
- GLOG_v=2 ./bin/mesos-tests.sh --verbose

For more information: ./bin/mesos-tests.sh --help
Submitting Patches
Preparation

Outline a clear context, motivation, and expected outcome of the patch
Posting Patches to ReviewBoard

Get familiar with git interactive rebase: `git rebase -i`

Break up the commits into logical chunks: `git add -p`

Post the commits as individual patches: `./support/post-reviews.py`

- Much easier to review
- Much easier to cherry-pick commits
- Separation of concern between behavioral change and code refactor
Addressing Comments

- Don’t take it personally!
- Broken rules are not an excuse to break them further (Broken windows theory)
- On contentious issues, opt for consistency. Keep the scope of the review narrow and file JIRA tickets to track changes proposed at the wider scope.

Remember, conflict is good for progress!
Summary

- Agree upon a concrete design before submitting patches
- Try not to surprise the reviewers
- Build a relationship and trust with reviewers (e.g. writing comprehensive tests)