Continuous Integration

Mark C. Miller
Lawrence Livermore National Laboratory

Software Productivity Track, ATPESC 2020

See slide 2 for license details
License, Citation and Acknowledgements

License and Citation

• This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).
• The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Mark C. Miller, Katherine M. Riley, and James M. Willenbring, Software Productivity Track, in Argonne Training Program for Extreme Scale Computing (ATPESC), August 2020, online. DOI: 10.6084/m9.figshare.12719834
• Individual modules may be cited as Speaker, Module Title, in Software Productivity Track...

Acknowledgements

• Additional contributors include: Patricia Grubel, Rinku Gupta, Mike Heroux, Alicia Klinvex, Jared O’Neal, David Rogers, Deborah Stevens
• This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
• This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
• This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
• This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344. IM Release #LLNL-PRES-813357
• This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.
What is Continuous Integration (CI) Testing

• Testing
  – Focused, critical functionality (infrastructure), fast, independent, orthogonal, complete, …
  – Existing test suites often require re-design/refactoring for CI

• Integration
  – Changes across key branches merged & tested to ensure the “whole” still works
  – Develop, develop, develop, merge, merge, merge, test, test, test…NO!
  – Develop, merge, test, develop, merge, test, develop, merge, test…YES!

• Continuous
  – Changes tested every commit and/or pull-request (like auto-correct)

• CI generally implies a lot of automation
Automated Testing vs. Continuous Integration (CI) Testing

• **Automated Testing**: Software that automatically performs tests and reliably detects and reports anomalous behaviors/outcomes.
  – Examples: Auto-test, CTest/CDash, nightly testing, `make check`
  – Potential issues: change attribution, timeliness of results, multiple branches of development

• **Continuous Integration (CI)**: automated testing performed at high frequency and fine granularity aimed at *preventing* code changes from breaking key branches of development (e.g. *main*)
  – Example: Disabled/enabled “Merge Pull Request” button on GitHub
  – Potential issues: extreme automation, test granularity, coverage, 3rd-party services/resources
Examples...

Automated Nightly Testing Dashboard
Lives “next to” your development work

CI Testing
Lives embedded in your development work
CI Testing is one part of the “Shift Left” movement in DevOps
## What can make CI Difficult

### Common situations

- **Just getting started**
  - Many technologies/choices; often in the "cloud"
  - Solution: start small, simple, build up

- **Developing suitable tests**
  - Many project’s existing tests not suitable for CI
  - Solution: Simplify/refactor and/or sub-setting test suite

- **Ensuring sufficient coverage**
  - Some changes to code never get tested
  - Solution: tools to measure it, enforce always increasing

### Advanced situations

- **Defining failure for many configurations**
  - Bit-for-bit (exact) match vs. fuzzy match
  - Solution: absolute/relative tolerances → AI/ML

- **Numerous 3rd party libraries (TPLs)**
  - Compiling takes too long
  - Solution: cache pre-built TPLs, containers

- **Performance testing**
  - Avoid time-, space-, scaling-performance degradation
  - Solution: Perf. instrumentation and *scheduled* testing
CI Resources (Where do jobs run?)

• Free Cloud Resources (many free on GitHub, BitBucket, GitLab, etc.)
  – Travis-CI, Circle-CI, AppVeyor, Azure Pipelines,…
  – All launch a VM (Linux variants, Windows and OSX)
    • Constrained in time/size, config. (e.g. GPU type/count)
    • Not always suitable for large, HPC projects due to need for longer than usual time to run

• Site-local Resources
  – Examples: Bamboo @ LLNL, Jenkins @ ANL, Travis+CDash @ NERSC, etc.
  – ECP Program: GitLab-CI @ ANL, LANL, LLNL, NERSC, ORNL, SNL

• Create your own by setting up resources/services
Examples…

Your code repository

Your CI Resources

github.com

travis-ci.com

bitbucket.llnl.gov

gitlab.nersc.gov
ECP CI Resources

• ECP investing in GitLab for complex-wide CI

• Non-GitLab projects *mirror* into GitLab

• Complex-wide Federation via OSTI
  – Many hurdles still to overcome
  – Manual federation possible…but non-trivial

• Documentation and on-boarding help
  – [https://ecp-ci.gitlab.io](https://ecp-ci.gitlab.io)
  – email me, [miller86@llnl.gov](mailto:miller86@llnl.gov) for on-boarding contacts
Getting started with CI

• What *configuration* is most important?
  – Examples: gcc, icc, xlc? MPI-2 or MPI-3? Python 2, 3 or 2 & 3?

• What *functionality* is most important?
  – Examples: vanilla numerical kernels? OpenMP kernels? GPU kernels? All of these?

• Good candidates…
  – A “hello world” example for your project
  – Once you’ve got the basics working, its easy to build up from there
Add more commits by pushing to the markmiller86-patch-3 branch on markmiller86/hello-numerical-world.

Some checks were not successful
1 failing and 3 successful checks

- codecov/patch — 0.00% of diff hit (target 51.60%)
- Travis CI - Branch  Successful in 20s — Build Passed
- Travis CI - Pull Request  Successful in 21s — Build Passed
- codecov/project — 72.43% (+20.83%) compared to 1307815

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.
### Getting started with CI:

#### Setting up CI

#### Example `.travis.yml` file
(also doing coverage analysis)

<table>
<thead>
<tr>
<th>Service</th>
<th>Interface</th>
<th><code>.travis.yml</code> in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travis</td>
<td>repo YAML file [&amp; repo scripts]</td>
<td>root of repo</td>
</tr>
<tr>
<td>GitLab</td>
<td>Web page configurator + repo YAML file [&amp; repo scripts]</td>
<td>root of repo</td>
</tr>
<tr>
<td>Bamboo</td>
<td>Web page configurator + repo scripts</td>
<td></td>
</tr>
</tbody>
</table>
Getting started with CI:

Example .travis.yml file (also doing coverage analysis)

Keywords defined by service provider’s YAML docs

Specify environment

Commands to run test
travis-ci.com

codecov.io
After Hours Hands-on Lesson – YouTube Video

- Follow QR code to GitHub repository
  - You can do this exercise entirely in your browser on GitHub
- Fork the repo
- Create .travis.yml using
- Submit Pull Request (PR)
- Increase coverage
  - Change ‘check’ to “check_all”
- Update the PR and observe coverage change
- Extra credit…fail PR if coverage drops
  - Hint: read codecov.io docs