OpenACS/EuroTcl 2022

GitLab CI pipelines for OpenACS development

Héctor Romojaro Gómez
Learn@WU Systemmanager
Learn@WU

- One of the world’s **most intensively used** E-learning platforms in higher education
- Based on OpenACS + NaviServer
- Started in 2002, designed for scalability

Some numbers:
- Up to 15 Mio hits and 3.3 Mio page impressions/day from registered users
- Up to 2500 concurrent users, over 250 views/sec
- Average response time on views less than 0.05 sec
- More than 120,000 learning resources have been developed since 2002
- Single instance
Back in 2016...

- No Upstream merging
- Package upgrades have been done occasionally
- Divergence increased over time
- It made more complex to integrate from the community
  - Bug Fixes
  - New features
  - Security fixes
  - Performance improvements
- Tests run manually
Numbers

Divergence size as of March 2016

- `git diff ...
- Core packages
  - Files changed: 5514
  - Insertions: 100222
  - Deletions: 272529
- Non Core packages
  - Files changed: 4320
  - Insertions: 103791
  - Deletions: 196018
- Ignoring catalog files, ajaxhelper, white spaces and local packages.

![Bar chart showing files, insertions, and deletions for ACS core and non-core packages.](chart.png)
The Challenge

- Reduce divergency to the **minimum necessary**.
- Integrate OpenACS upstream code **efficiently**
  - Use Github’s OpenACS repository as a *remote* for our Git repository
  - Import new upstream commits quickly and easily
  - Merge OpenACS code with ours keeping the history of both
- Increase Software Quality
  - Decrease duplicity and redundancy
  - Trigger automated tests automatically
The Solution

- Use two Git remotes (*origin* for local code and *github* for upstream)
- Split *non-core* packages into separate repositories, matching upstream’s Github structure
- Initial merge on common ancestor
  - Find the upstream branch more similar to downstream
- Subsequent merges are easy and fast
- *myrepos* to manage multiple repositories easily
- *GitLab* local instance for CI pipelines
  - Trigger automated tests on every new commit in the integration branch, including upstream merges.
Integrated git structure

- 1 repository for `acs-core`
- "n" repositories for non-core packages
  - n = 66
- 2 remotes per repository
  - Github
  - Origin
- All branches available from local copy
  - oacs-x-y from github
  - integration from origin
- oacs-x-y is merged into integration daily
- All local development is in integration
- All history, local and upstream, is preserved
Directory structure

- **Super-repository** stores the myrepos and common Gitlab CI config.

- **acs-core** repository contains OpenACS core and local packages with no upstream counterpart.

- **non-core** repositories, one per non-core package.
GitLab

- Web-based DevOps lifecycle tool
- Git repository manager
- Integrated Web IDE
- Free software (Gitlab CE), with an open-core development model
- Private repositories, groups, forks, permissions, stats...
- On premises
- Store CI/CD config (gitlab-ci.yml) in super-repository
- Integrated CI/CD pipelines
  - Every change in the integration branch triggers the pipeline.
  - GitLab runners execute the jobs in docker containers with a running NaviServer
  - Jobs are run in stages
Job internals

- Run by GitLab runners on docker containers with a running NaviServer
- Access to a shared PG instance, where the DB pool is
- ci-* scripts on /www/ accessed via cURL by GitLab
- Results output using ns_write
- Info retrieved via grep from server logs
  - Success of the job
  - egrep -wq 'ci-tests:....'
- Artifacts
  - error.log
  - rebuild_db
Try to avoid DB rebuild

Gitlab runners in docker containers execute jobs in stages:

1. Build (DB and source tree setup)
2. Test coverage
3. Package upgrade
4. Package safe tests (production_safe)
5. Package message catalog import
6. Global package upgrade
7. Global safe tests (production_safe)
8. Global message catalog import
9. Package unsafe tests
10. Global unsafe tests
11. Cleanup (DB rebuild if tainted and source tree removal)
Stages: Build

- Source tree retrieval using myrepos
- Database reservation
  - Thinned out production database (~270G)
  - Picked from a pool of databases
    - *gitlab-pipeline-free-* renamed to *gitlab-pipeline-$PIPELINE_ID*
  - Manage concurrency with *db-mutex* runner
    - Only one runner is tagged db-mutex
    - *db-mutex* is the only runner allowed to reserve/regenerate DB
Stages: Coverage

- Public procs covered by automated tests
- Fail if coverage decreases (enforces policy)
- New on 5.10
  - `aa::coverage::*`
  - `/test/admin/proc-coverage`
  - Global and per package
Stages: Upgrade

- Upgrades a single or all possible OpenACS packages
- Fail if dependency error or unsuccessful upgrade
- Taints DB if upgrade is performed, triggering a rebuild from the template on the Cleanup stage
- APM api
  - `apm_package_*`
  - `apm_scan_packages`
Stages: Message catalog import

- Imports the message catalog files of a single or all possible packages.
- Detects conflicts.
- Detects changes on message keys (add, delete, update).
- Fail if changes or conflicts are detected without a package upgrade (enforces policy)
- **Taints** DB if changes are performed, triggering a rebuild from the template on the Cleanup stage.
- **acs-lang api**
  - `lang::catalog::import`
  - `lang::message::conflict_count`
Stages: Tests

- Runs tests from *acs-automated-testing* on a single or all possible packages
- Fail if any test fails
- *Taints* DB if non *production_safe* tests are run
- Run as test user
  - *acs::test::user::*
- AA api
  - *aa_runseries*
  - *nsv_get aa_test cases*
Stages: Cleanup

- Deletes the source tree
- Restores or rebuilds DB
  - If DB is not *tainted*, just put it back in the pool
    - Rename `gitlab-pipeline-$PIPELINE_ID` to `gitlab-pipeline-free-*`
  - If DB is *tainted*, recreate it from the template DB
  - Manage concurrency with the `db-mutex` runner
...and the present!

- All work is committed to the integration branch
- Latest upstream code (5.10 branch) is merged into the integration branch daily
- Every push to the integration branch triggers the pipeline
- Internal releases every two weeks
Thanks for watching!

Some unnecessary links

- LEARN: https://learn.wu.ac.at/
- OpenACS: https://openacs.org
- OpenACS on Github: https://github.com/openacs
- myrepos: https://myrepos.branchable.com/
- GitLab: https://about.gitlab.com/