Open Source Development Course

Continuous integration and deployment (CI/CD)

Vojtěch Trefný
`vtrefny@redhat.com`

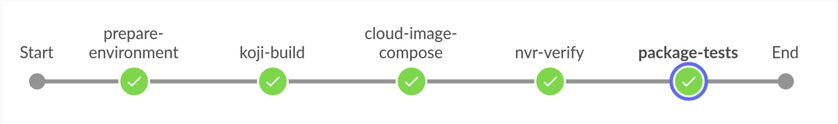
23. 3. 2022

twitter.com/vojtechtrefny
github.com/vojtechtrefny
gitlab.com/vtrefny

Pipeline

CI/CD Pipeline

- Steps that need to be performed to test and deliver a new version of the software.
- Defines what needs to be done: when, how and in what order.
- Steps can vary for every project.
- Multiple pipelines or steps can run in parallel.



CI/CD Pipeline

1. Testing environment

Preparation of the environment to run the tests: deploying containers, starting VMs...

2. Static Analysis

Finding defects by analyzing the code without running it.

3. Code style

Checking for violations of the language or project style guides.

4. Build

Building the project from source.

5. Tests

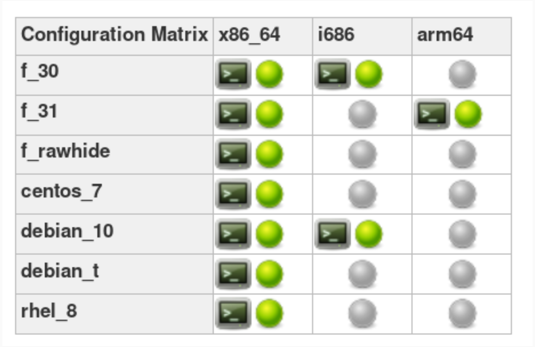
Running project test suite or test suites.

6. Packaging and Deployment

Building source archives, packages or container images.

Testing Environment

Testing Environment



Configuration Matrix	x86_64	i686	arm64
f_30	🖥 🟢	🖥 🟢	⚪
f_31	🖥 🟢	⚪	🖥 🟢
f_rawhide	🖥 🟢	⚪	⚪
centos_7	🖥 🟢	⚪	⚪
debian_10	🖥 🟢	🖥 🟢	⚪
debian_t	🖥 🟢	⚪	⚪
rhel_8	🖥 🟢	⚪	⚪

1. Preparation of VMs/containers to run the tests

We might want to run tests in different environments on multiple different distributions or architectures.

2. Installation of the test dependencies

Test dependencies are usually not covered by the project dependencies.

3. Getting the code

Clone the PR or get the latest code from the master branch.

Static Analysis

Static Analysis

- Tools that can identify potential bugs by analyzing the code without running it.
- Can detect problems not covered by the test suite – corner cases, error paths etc.
    - Coverity (C/C++, Java, Python, Go...)[1]
    - Cppcheck (C/C++)[2]
    - Pylint (Python)[3]
    - RuboCop (Ruby)[4]

[1] `https://scan.coverity.com`

[2] `http://cppcheck.sourceforge.net/`

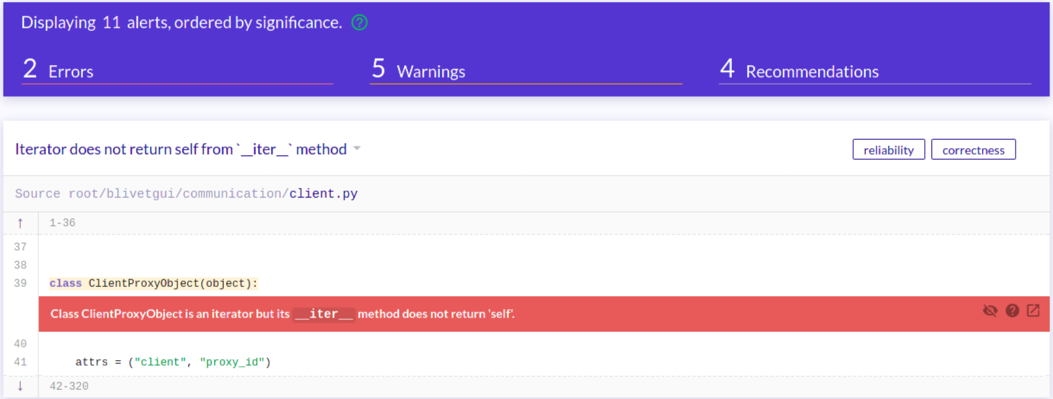
[3] `https://www.pylint.org`

[4] `https://docs.rubocop.org`

Coverity

```
Error: USE_AFTER_FREE (CWE-825):
libblockdev-2.13/src/plugins/lvm-dbus.c:1163: freed_arg: "g_free"
frees "output".
libblockdev-2.13/src/plugins/lvm-dbus.c:1165: pass_freed_arg: Passing freed
pointer "output" as an argument to "g_set_error".
# 1163|       g_free (output);
# 1164|       if (ret == 0) {
# 1165|->          g_set_error (error, BD_LVM_ERROR, BD_LVM_ERROR_PARSE,
# 1166|                       "Failed to parse number from output: '%s'",
# 1167|                       output);
```

LGTM



Displaying 11 alerts, ordered by significance.

2 Errors 5 Warnings 4 Recommendations

Iterator does not return self from `__iter__` method

reliability correctness

Source root/blivetgui/communication/client.py

```
1-36


class ClientProxyObject(object):
```

Class ClientProxyObject is an iterator but its `__iter__` method does not return 'self'.

```

    attrs = ("client", "proxy_id")
42-320
```

https://lgtm.com/projects/g/storaged-project/blivet-gui/

Runtime Analysis

- Tools that can identify bugs during runtime.
- Needs the code to actually run, either through manual testing or when running the test suite.
    - Valgrind – memory management and threading bugs[5]
    - ASan – *AddressSanitizer* – memory management bugs (buffer overflow, dangling pointers...). Part of the LLVM Analyzers project, integrated into `gcc` and `clang`[6]

[5] https://valgrind.org/

[6] https://github.com/google/sanitizers/wiki/AddressSanitizer

Code Style

Code style and style guides

- Coding conventions – naming, code lay-out, comment style. . .
- Language specific (PEP 8[7]), project specific (Linux kernel coding style[8]) or library/toolkit specific (GTK coding style[9]).
- Automatic checks using specific tools (`pycodestyle`) or (partially) by the static analysis tools.

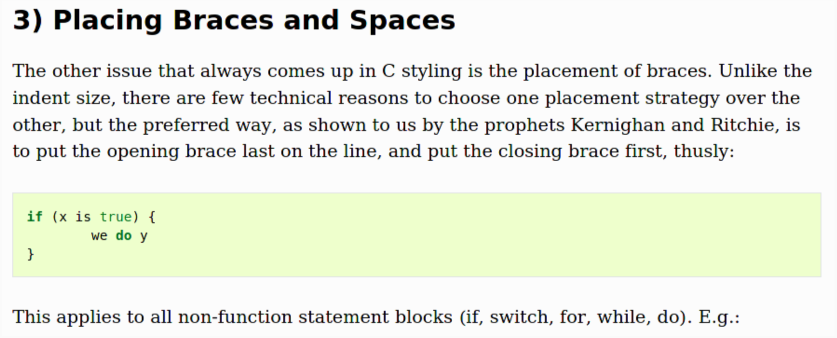
7 `https://www.python.org/dev/peps/pep-0008/`

8 `https://www.kernel.org/doc/html/v5.11/process/coding-style.html`

9 `https://developer.gnome.org/programming-guidelines/stable/c-coding-style.html.en`

Linux kernel coding style

https://www.kernel.org/doc/html/v5.11/process/coding-style.html



3) Placing Braces and Spaces

The other issue that always comes up in C styling is the placement of braces. Unlike the indent size, there are few technical reasons to choose one placement strategy over the other, but the preferred way, as shown to us by the prophets Kernighan and Ritchie, is to put the opening brace last on the line, and put the closing brace first, thusly:

```
if (x is true) {
        we do y
}
```

This applies to all non-function statement blocks (if, switch, for, while, do). E.g.:

Python and PEP 8

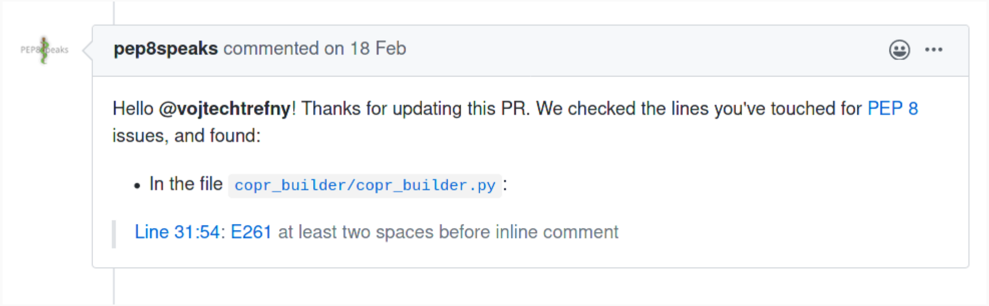
- Automatic code style checking tools exist for the Python PEP 8 style code.
- `pycodestyle`[10](formerly `pep8`) is used for checking/enforcing PEP 8 in many Python applications.
- `black`[11]can be used to automatically format Python code in a PEP 8 compliant way.
- Static analysis tools like `pylint` or `pyflakes` also check for some PEP 8 style violations.

[8] `https://github.com/PyCQA/pycodestyle`

[9] `https://github.com/psf/black`

Python and PEP 8

```
$ pycodestyle-3 blivetgui/blivetgui.py
blivetgui/blivetgui.py:23:80: E501 line too long (80 > 79 characters)
blivetgui/blivetgui.py:30:1: E402 module level import not at top of file
```


pep8speaks commented on 18 Feb

Hello @vojtechtrefny! Thanks for updating this PR. We checked the lines you've touched for PEP 8 issues, and found:

- In the file `copr_builder/copr_builder.py`:

> Line 31:54: E261 at least two spaces before inline comment

Documentation style

- Documentation might be checked in the same way code is.
- Similar style documents and tools for checking documentations exist (for example PEP 257[12]and pydocstyle[13]for Python).
- In some cases wrong or missing documentation (docstrings in the code) can lead to a broken build or missing features.

[10] https://www.python.org/dev/peps/pep-0257/
[11] http://www.pydocstyle.org

Build

Build

- Building the project, a preparation to run the test suite.
- Depends on language – mostly no-op for interpreted languages, more complicated for compiled ones.
- Build in the CI environment can detect issues with dependencies.
- Builds on different architectures can help detect issues related to endianness or data types sizes.

Tests

Tests

- Running tests that are part of the project.
- New tests should be part of every change to the codebase.
  - New features require new unit and integration tests.
  - Bug fixes should come with a regression test.
- For some project (like libraries) running test suites of their users might be an option.

Coverage

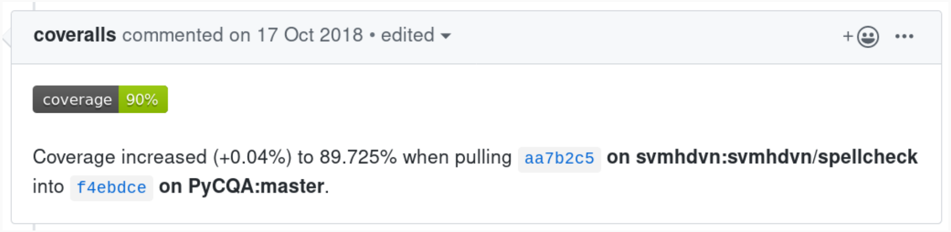
- Code coverage (or Test coverage) represents how much of the code is covered by the test suite.
- Usually percentual value that shows how many lines of the code were “visited” by the test.
- Generally a check that all functions and branches are covered by the suite.
- Used as a measure of the test suite “quality”.

Coverage

```
Name    Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------------------
a.py      487      9    178     11    97%   206, 268, 377->376, 393->392, 418,
                                            448, 452, 460, 660, 729, 746,
                                            889->891, 891->894
b.py      220      8     74      8    95%   81, 173, 193->197, 279, 340, 342, 346
c.py       19      9      8      2    44%   35-36, 50->48, 53, 60-70
d.py        5      0      6      1    91%   31->exit
e.py       46      0      4      0   100%
...
-----------------------------------------------------------------------
TOTAL    3600   1477   1381    100    56%
```

Coverage

- Automated coverage tests might be part of the CI.
- Decrease in coverage can be viewed as a reason to reject contribution to the project.



coveralls commented on 17 Oct 2018 • edited

Coverage increased (+0.04%) to 89.725% when pulling `aa7b2c5` **on svmhdvn:svmhdvn/spellcheck** into `f4ebdce` **on PyCQA:master**.

Delivery and Deployment

Packaging and publishing

- **Delivery** – releasing new changes quickly and regularly (daily, weekly...).
- **Deployment** – delivery with automated push to production, without human interaction.

- Usually after merging the changes, not for the PRs.
- Building packages, container images, ISO images. . .
- Built packages can be used for further testing (manually by the Quality Assurance or in another CI infrastructure) or directly pushed to production or included in testing/nightly builds of the project.

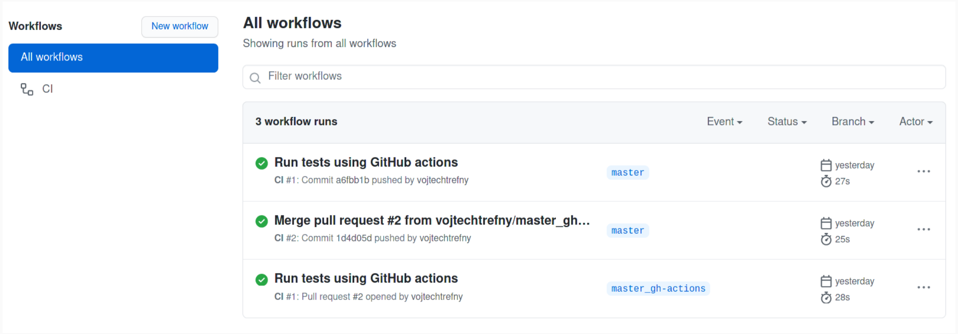
CI Tools

Demo

GitHub Actions

- Automation *framework* integrated into GitHub.
- Does not cover only CI but also CD (publishing packages on various services and deploying on many public clouds) and project and issue management.
- Free for all public repositories, limited and paid options for private projects.
- https://github.com/features/actions

GitHub Actions



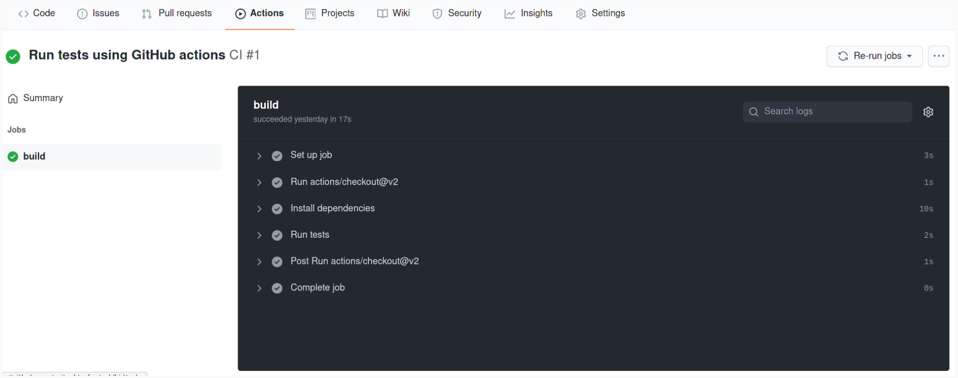
All workflows

Showing runs from all workflows

3 workflow runs

✓ **Run tests using GitHub actions** CI #1: Commit a6fbb1b pushed by vojtechtrefny	master	yesterday 27s	...
✓ **Merge pull request #2 from vojtechtrefny/master_gh...** CI #2: Commit 1d4d05d pushed by vojtechtrefny	master	yesterday 25s	...
✓ **Run tests using GitHub actions** CI #1: Pull request #2 opened by vojtechtrefny	master_gh-actions	yesterday 28s	...

GitHub Actions



Run tests using GitHub actions CI #1

build

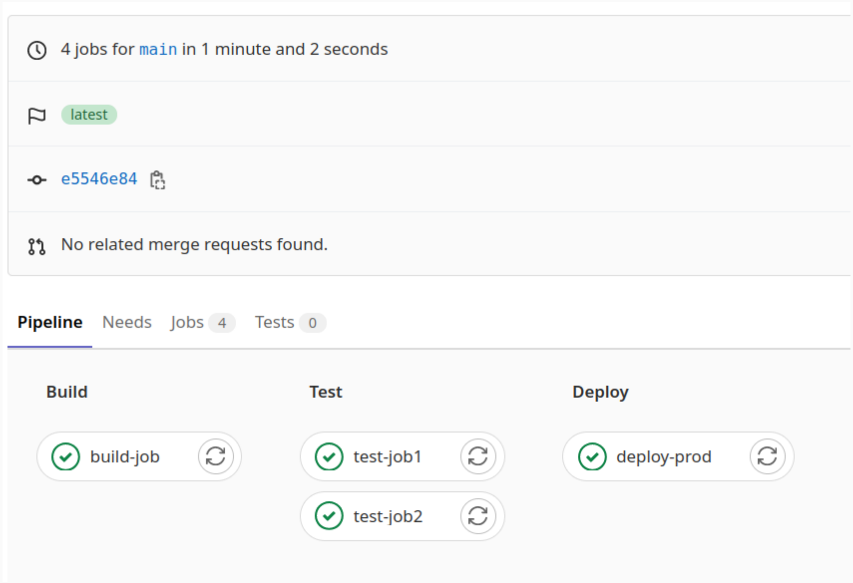
build
succeeded yesterday in 17s

```
Set up job                      3s
Run actions/checkout@v2         1s
Install dependencies           10s
Run tests                       2s
Post Run actions/checkout@v2    1s
Complete job                    0s
```

GitLab CI

- CI/CD automation integrated into GitLab.
- Configuration is done with a YAML file in the repository.
- Pipleines/tests can run either on infastructure provided by GitLab or on custom runners.
- `https://docs.gitlab.com/ee/ci/`

GitLab CI



4 jobs for main in 1 minute and 2 seconds

latest

e5546e84

No related merge requests found.

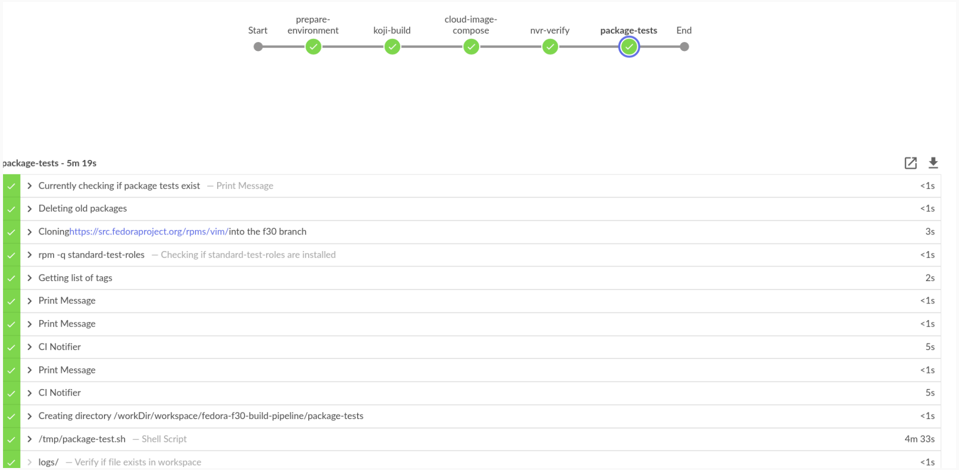
Jenkins

- Automation system, not a “true” CI/CD tool.
- Can automatically run given tasks on a node or set of nodes.
- Tasks can be started on time basis or triggered by an external event (like a new commit or PR on GitHub).
- https://jenkins.io/

Fedora CI

- Complex CI system with the task to deliver an "Always Ready Operating System".
- Packages are tested after every change and *gated* if the CI pipeline fails.
- The goal is to prevent breaking the distribution. CI will stop the broken package before it can affect the distribution.

Fedora CI



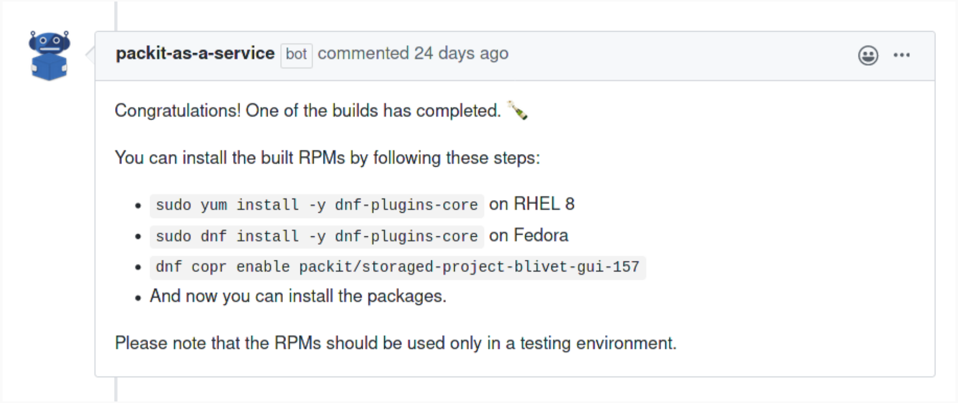
package-tests - 5m 19s

```
✓ > Currently checking if package tests exist  — Print Message                          <1s
✓ > Deleting old packages                                                               <1s
✓ > Cloninghttps://src.fedoraproject.org/rpms/vim/into the f30 branch                    3s
✓ > rpm -q standard-test-roles  — Checking if standard-test-roles are installed         <1s
✓ > Getting list of tags                                                                 2s
✓ > Print Message                                                                       <1s
✓ > Print Message                                                                       <1s
✓ > CI Notifier                                                                          5s
✓ > Print Message                                                                       <1s
✓ > CI Notifier                                                                          5s
✓ > Creating directory /workDir/workspace/fedora-f30-build-pipeline/package-tests       <1s
✓ > /tmp/package-test.sh  — Shell Script                                             4m 33s
✓ > logs/  — Verify if file exists in workspace                                         <1s
```

Packit

- Tool for integrating upstream projects to Fedora.
- RPM packages are automatically built on every pull request.
- New releases can be automatically built and pushed to Fedora.

Packit



Congratulations! One of the builds has completed. 🍾

You can install the built RPMs by following these steps:

- `sudo yum install -y dnf-plugins-core` on RHEL 8
- `sudo dnf install -y dnf-plugins-core` on Fedora
- `dnf copr enable packit/storaged-project-blivet-gui-157`
- And now you can install the packages.

Please note that the RPMs should be used only in a testing environment.

Travis CI

- Used to be the most popular CI service for open source products.
- Can be integrated into your projects on GitHub and GitLab.
- Configured using `.travis.yml` file in the project
- Unfortunately Travis drastically limited free plans for open source projects in 2020[14].
- https://travis-ci.com

[12] https://blog.travis-ci.com/2020-11-02-travis-ci-new-billing

Questions

Questions

Thank you for your attention.

https://github.com/crocs-muni/open-source-development-course