Automation and Testing for Simplified Software Deployment

André Sailer[1],* and *Marko* Petrič[1],**

[1]CERN, CH-1211 Geneva 23, Switzerland

Abstract. Creating software releases is one of the more tedious occupations in the life of a software developer. For this purpose we have tried to automate as many of the repetitive tasks involved as possible from getting the commits to running the software. For this simplification we rely in large parts on free collaborative services available around GitHub: issue tracking, code review (GitHub), continuous integration (Travis-CI), static code analysis (coverity). The dependencies and compilers used in the continuous integration are obtained by mounting CVMFS into a docker container. This enables running any desired compiler version (e.g., gcc 6.2, llvm 3.9) or tool (e.g, clang-format, pylint). To create tags for the software package the powerful GitHub API is used. A script was developed that first collates the release notes from the description of each pull request, commits the release notes file, and finally makes a tag. This moves the burden of writing release notesfrom the package maintainer to the individual developer. The deployment of software releases to CVMFS is handled via GitLab-CI. When a tag is made the software is built and automatically deployed. In this paper we will describe the software infrastructure used for the iLCSoft and iLCDirac projects, which are used by CLICdp and the ILC detector collaborations, and give examples of automation which might be useful for others.

1 Introduction

Sophisticated simulation and reconstruction software is needed to address the detector and physics issues for the future Compact Linear Collider (CLIC), a high-energy electron–positron accelerator [1]. Our software suite is developed in the linear collider community and consists of: detector geometry description using DD4hep [2–4] which is based on Geant [5, 6] and Root [7]; the iLCSoft framework for reconstruction and analysis based on Marlin [8]; the event data model and persistency format LCIO [9, 10]; and the distributed workload and data management tool iLCDirac [11] based on the DIRAC grid middleware [12]. The simulation and reconstruction software is used to perform detector optimisation [13] and physics studies [14, 15].

To preserve our agile and pragmatic approach to software development without sacrificing maintainability and code quality, we rely on a number of tools to automate necessary but tedious tasks as much as possible. Code review is an irreplaceable ingredient of code quality

*e-mail: andre.philippe.sailer@cern.ch
**e-mail: marko.petric@cern.ch

assurance, but many aspects can be automated before someone has to take a look at new developments or improvements. Unit-testing, formatting and static analysis can be performed automatically and thus remove some of the burden from the code reviewer. Further automation is provided for the creation of release notes and the creation and deployment of software releases. In the following sections we describe some of the tools that we have adopted into our workflow that have significantly improved our process of software development. First we describe tools related to the GitHub environment, which are most straightforward and free to use for anyone working with open source software. Then we describe tools related to the GitLab instance available at CERN, and finally some plug-ins available for Jenkins.

2 Continuous Integration with GitHub and Integrated Tools

The commonly developed and maintained software is hosted on GitHub for ease of distribution and developer permission management. Continuous integration (CI) can be easily set up for any public project on GitHub with various services integrated with that platform. We have added the use of Travis-CI to the projects we maintain or contribute to: the packages that make up iLCSoft and DD4hep, which are based on C++, and `DIRAC` [12] which is based on Python. The Travis-CI can run the configured tasks for each *push*, *pull request*, or as a weekly *cron job*.

2.1 CI for Projects without Dependencies

The simplest use case occurs for projects without, or only with easily available, dependencies. An example is the Python-based `DIRAC` software. Listing 1 shows part of the configuration to execute the CI tasks. In lines 2–4 the requirements are installed with the standard `pip` program. This is sufficient to run the unit-tests discovered and executed by `py.test` (line 9). In lines 11–12 the documentation is compiled. If warnings occur during the compilation of the documentations the build is marked as failed, which, for example, ensures that all files are referred to inside the documentation[1]. Line 14 contains a command to run the `pylint` static code analysis, which is configured to look only for errors, for example the use of undefined variables or functions.

Lines 15–19 verify that the formatting of the modified lines in a pull request conform to the `DIRAC` conventions. A strict enforcement of coding style was only recently added, therefore large parts of the code base do not yet conform to the style conventions. By only verifying modified lines of the source code we can enforce the conventions for new files without having to format the entire code base, which would affect the information of the revision history for almost every line of code.

The complete Travis-CI configuration and the customised commands it runs can be found in the `DIRAC` GitHub repository https://github.com/DIRACGrid/DIRAC in the `.travis.yml` file and the `travis-ci.d` folder. A similar approach to run continuous integration tasks can be adopted by other projects if the installation is easy and fast and no infrastructure needs to be provided.

2.2 CI with CVMFS to Provide Dependencies

For the continuous compilation and testing of the C++ based software, for example DD4hep, we first need to provide its dependencies, which include Geant and Root and for our use

[1]The build in the CI is just for testing, The documentation is automatically deployed with `readthedocs.org` and can be found at https://dirac.readthedocs.org

```
install:
  - pip install --upgrade setuptools
  - pip install --upgrade pip
  - pip install -r requirements.txt
script:
  - export PYTHONPATH=${PWD%/*}
  - ls $PYTHONPATH
  - if [[ "${CHECK}" == "py.test"  ]]; then
        py.test;
    elif [[ "${CHECK}" == "docs"  ]]; then
        sudo apt-get install graphviz; cd docs; SPHINXOPTS=-wsphinxWarnings READTHEDOCS=True make html;
        if [ -s sphinxWarnings ]; then cat sphinxWarnings; echo "Warnings When Creating Doc"; exit 1; fi
    elif [[ "${CHECK}" == "pylint"  ]]; then
        travis_wait 30 .travis.d/runPylint.sh;
    elif [[ "${CHECK}" == "format" ]] && [[ "${TRAVIS_PULL_REQUEST}" != "false" ]]; then
        git remote add GH https://github.com/DIRACGrid/DIRAC.git;
        git fetch --no-tags GH ${TRAVIS_BRANCH};
        git branch -vv;
        git diff -U0 GH/${TRAVIS_BRANCH} | pycodestyle --diff;
    fi
```

Listing 1. Travis-CI Part of the `yml` setup used for the `DIRAC` grid middleware

case also parts of iLCSoft. To access the dependencies, we use the docker service in Travis-CI and also install CVMFS [16]. This is shown in Listing 2. In lines 1–12 the CVMFS packages are downloaded, installed, and configured on the virtual machine and finally the `clicdp.cern.ch` CVMFS repository is mounted. In lines 14–18, the docker container, that runs the compilation and unit tests, is executed.

As we can provide any dependency via CVMFS we can quickly and efficiently compile and test all of our C++ based software packages on GitHub for each pull request.

2.3 Static Code Analysis with Coverity

Another useful tool that is available free to open source projects hosted on GitHub is the `coverity` static code analysis [17]. After registering the project with the service, the analysis can be run via a cron job in the Travis-CI system. Listing 3 shows how the analysis result, which is run parallel to a compilation in Travis-CI, is submitted to the coverity system

```
1  wget https://ecsft.cern.ch/dist/cvmfs/cvmfs-release/cvmfs-release-latest_all.deb
2  sudo dpkg -i cvmfs-release-latest_all.deb
3  sudo apt-get update
4  sudo apt-get install cvmfs cvmfs-config-default
5  rm -f cvmfs-release-latest_all.deb
6  wget https://lcd-data.web.cern.ch/lcd-data/CernVM/default.local
7  sudo mkdir -p /etc/cvmfs
8  sudo mv default.local /etc/cvmfs/default.local
9  sudo /etc/init.d/autofs stop
10 sudo cvmfs_config setup
11 sudo mkdir -p /cvmfs/clicdp.cern.ch
12 sudo mount -t cvmfs clicdp.cern.ch /cvmfs/clicdp.cern.ch
13
14 docker run -it --name CI_container \
15        -v $PKGDIR:/Package \
16        -e COMPILER=$COMPILER \
17        -v /cvmfs/clicdp.cern.ch:/cvmfs/clicdp.cern.ch \
18        -d clicdp/slc6-build /bin/bash
```

Listing 2. Setting up CVMFS with docker inside Travis-CI

```
if [[ "${TRAVIS_EVENT_TYPE}" == "cron" && "${COMPILER}" == "gcc"  ]]; then
    export COVERITY_REPO=`echo ${TRAVIS_REPO_SLUG} | sed 's/\//\%2F/g'`
    wget https://scan.coverity.com/download/linux64 \
          --post-data "token=${COVERITY_SCAN_TOKEN}&project=${COVERITY_REPO}"\
          -O Package/coverity_tool.tgz; \
     cd Package; mkdir cov-analysis-linux64; \
     tar -xf coverity_tool.tgz -C cov-analysis-linux64 --strip-components=2;
     curl --form token=${COVERITY_SCAN_TOKEN} --form email=noreply@cern.ch \
          --form file=@${PKGDIR}/build/myproject.tgz --form version="master" \
          --form description="${description}" \
          https://scan.coverity.com/builds?project=${COVERITY_REPO} ;
fi
```

Listing 3. Submitting new build information to the coverity static analysis service in a cron job on Travis-CI

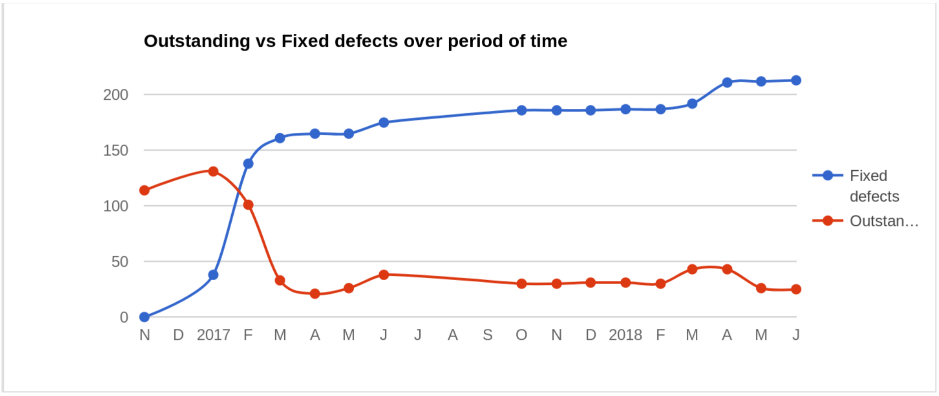

Figure 1. Defect density in DD4hep over time

for analysis. The discovered *defects* can be browsed by persons authorised by the project maintainers.

Figure 1 shows the number of *outstanding* and the cumulative number of *fixed defects* in DD4hep over time. Whenever a new defect is found, an email is sent to the members of the project, and usually significant defects are quickly fixed, as can be seen by the bump in the outstanding defects in March and April of 2018, shown in Figure 1.

3 Continuous Integration with GitLab

The continuous integration tasks running on GitHub integrated services can also be run in a similar fashion on the CERN GitLab instance, however the access cannot be as easily obtained as on GitHub. One advantage of using an in-house deployed tool is that tasks requiring privileges can be executed. For our iLCDirac [11] extension of `DIRAC` we run the same checks as for `DIRAC`, but in addition we can automatically deploy the documentation as a website, and publish the client installation to our CVMFS repository.

3.1 Continuous Deployment to CVMFS with GitLab

To deploy our software releases and required dependencies we are using a `ssh` based `gitlab-runner` to connect to the CVMFS *stratum 0*. Many of our projects are using the same runner, but as we do not allow the execution of jobs in parallel, GitLab-CI is responsible for scheduling the tasks one after the other. In case the same project has more than one active pipeline to publish to CVMFS any redundant pipeline not yet in the deploy stage will be cancelled. This mostly happens for the continuous deployment of the *HEAD* installations of iLCSoft, when multiple packages trigger a new build of the software suite simultaneously.

4 Code Quality Monitoring with JenkinsCI

While the GitLab-CI is easy to configure and use, and allows pass–fail level checks for pull requests, the JenkinsCI software allows one to install a wide variety of plug-ins that can be used to monitor the code quality over time.

While our goal is to remove all compiler warnings – and once this is achieved for one of our packages it will be enforced with `-Werror` in CI compilations – the *Warnings* plug-in for Jenkins provides an overview of the existing warnings, and allows one to browse the source code where the warnings occur. Similar functionality is provided by the *clang scan-build plug-in*, which allows one to monitor the reports provided by the clang *scan-build* tool. The final plug-in we make use of is the *Valgrind* plug-in for the *valgrind mem-check* [18] tool to find memory leaks or errors. These three plug-ins also offer a graphical representation. Figure 2 shows the trend of the warnings (left), bugs (middle), and issues affecting memory (right). The valgrind plug-in allows one in addition to set thresholds of acceptable number of lost bytes. If these thresholds are exceeded, the build is marked as *unstable* or *failed*. The nightly running and analysis of the valgrind output greatly reduces the burden on the release manager and decreases the danger of releasing faulty software. As can be seen in the right graph of Figure 2, the spike in *definitely lost* memory was identified and fixed in the following day.

5 Leveraging APIs

Both GitHub and GitLab offer programmable access to their functionality. We use the APIs to access the pull requests and their content or comments, and to create new releases.

5.1 Release Notes

Release notes should contain the new developments for new versions of the software. To spread the burden of writing release notes and for simplification we moved the drafting of release notes to the comments accompanying each pull request. Each pull request should contain a brief statement explaining its new features or bugfixes. The release notes are written between BeginReleasenotes EndreleaseNotes and parsed by a script accessing the information via the APIs. All pull requests that occurred between the last tag on a given branch are selected and parsed. Depending on the preferences of the maintainers the release notes can be sorted, for example by pull requests or sub-system. An example of a script using the GitHub API can be found in the `DIRAC` repository[2] and a script using the GitLab API in the iLCDirac repository[3].

This approach has the advantage of directly asking contributors to add release notes to their pull requests. It also avoids *merge conflicts*, that would be almost guaranteed if every contributor were to modify a release notes file by themselves.

The output from the collation of the release notes can be further formatted, manually committed into a release notes file, or used in the meta data of tags made in GitHub or GitLab. It is also possible to automate the manual steps, as described in the next section.

[2]https://github.com/DIRACGrid/DIRAC/blob/integration/docs/Tools/GetReleaseNotes.py
[3]https://gitlab.cern.ch/CLICdp/iLCDirac/ILCDIRAC/blob/Rel-v29r0/docs/GetReleaseNotes.py

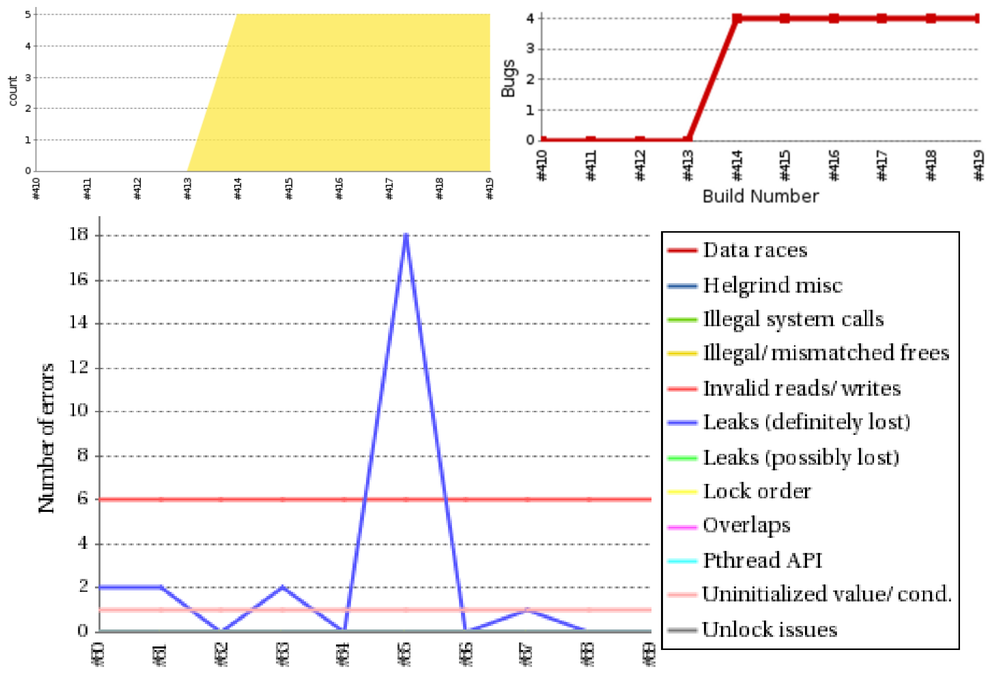

Figure 2. Trend graphs from different plug-ins for Jenkins: compiler warnings, clang scan-build, valgrind mem-check

5.2 Creation of New Software Releases

For the complex releases of iLCSoft which contains more than 30 separate packages, we create a *tagger* script[4], that uses the GitHub API to first create release notes for each package, directly commits the release notes into the project and then makes a tag with an incremented version. If no modifications have been done in the package with respect to a prior version no new tag is created. While the specific implementation makes use of conventions for iLCSoft the general idea can be adapted to other projects.

6 Conclusions

The combination of continuous integration and continuous deployment has immensely sped up the round trip time from changes in our detector simulation or improvements in the reconstruction algorithms to large scale validation samples. The automation of the testing, validation, and deployment increased the reliability of the release procedure and of the release software product considerably, while at the same time increasing the productivity of the librarians.

Disclaimer

All product and company names are trademarks or registered trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them.

[4]https://github.com/iLCSoft/iLCInstall/blob/master/scripts/ilcsofttagger.py

References

[1] P. Burrows et al., eds., *Updated baseline for a staged Compact Linear Collider* (CERN, 2016), `http://dx.doi.org/10.5170/CERN-2016-004`

[2] M. Frank, F. Gaede, M. Petric, A. Sailer, *DD4hep* (2018), `https://doi.org/10.5281/zenodo.592244`

[3] M. Petric, M. Frank, F. Gaede, S. Lu, N. Nikiforou, A. Sailer, J. Phys. Conf. Ser. **898**, 042015 (2016), `https://doi.org/10.1088/1742-6596/898/4/042017`

[4] M. Frank, F. Gaede, C. Grefe, P. Mato, J. Phys. Conf. Ser. **513**, 022010 (2013), `https://doi.org/10.1088/1742-6596/513/2/022010`

[5] S. Agostinelli et al., Nucl. Inst. & Meth. **A506**, 250 (2003), `https://doi.org/10.1016/S0168-9002(03)01368-8`

[6] J. Allison et al., IEEE T. Nucl. Sci. **53**, 270 (2006), `https://doi.org/10.1109/MASS.1995.528223`

[7] F. Rademakers et al., *root* (2018), `https://doi.org/10.5281/zenodo.848818`

[8] F. Gaede, Nucl. Inst. & Meth. **A559**, 177 (2006), `http://dx.doi.org/10.1016/j.nima.2005.11.138`

[9] F. Gaede, T. Behnke, R. Cassell, N. Graf, T. Johnson, H. Vogt, *LCIO persistency and data model for LC simulation and reconstruction*, in *CHEP 2004* (Interlaken, Switzerland, 2004)

[10] F. Gaede, T. Behnke, N. Graf, T. Johnson, *LCIO — A persistency framework for linear collider simulation studies*, in *CHEP 2003* (La Jolla, California, 2003)

[11] C. Grefe, S. Poss, A. Sailer, A. Tsaregorodtsev, J. Phys. Conf. Ser. **513**, 032077 (2013), CLICdp-Conf-2013-003, `https://doi.org/10.1088/1742-6596/513/3/032077`

[12] F. Stagni et al., *DIRAC* (2018), `https://doi.org/10.5281/zenodo.1451646`

[13] N. Alipour Tehrani et al., *CLICdet: The post-CDR CLIC detector model* (2017), CLICdp-Note-2017-001, `http://cds.cern.ch/record/2254048`

[14] CLICdp Collaboration, Eur. Phys. J. C **77**, 475 (2017), `https://doi.org/10.1140/epjc/s10052-017-4968-5`

[15] CLICdp Collaboration (CLICdp), *Top-Quark Physics at the CLIC Electron-Positron Linear Collider* (2018), `https://arxiv.org/abs/1807.02441`

[16] J. Blomer et al., *The CernVM file system* (2018), `https://doi.org/10.5281/zenodo.1010441`

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

[18] J. Seward, N. Nethercote, *Using Valgrind to detect undefined value errors with bit-precision*, in *Proceedings of the USENIX'05 Annual Technical Conference* (Anaheim, California, USA, 2005)