ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV050

DevOps AND CI/CD FOR WinCC OPEN ARCHITECTURE APPLICATIONS AND FRAMEWORKS

R. P. I. Silvola, CERN, Geneva, Switzerland
L. Sargsyan, A. Alikhanyan National Laboratory (former YerPhI), Yerevan, Armenia

Abstract

This paper presents the Continuous Integration and Continuous Deployment (CI/CD) tool chain for WinCC Open Architecture applications and frameworks developed at CERN, enabling a DevOps oriented approach of working. By identifying common patterns and time consuming procedures, and by agreeing on standard repository structures, naming conventions and tooling, we have gained a turnkey solution which automates the compilation of binaries and generation of documentation, thus guaranteeing they are up to date and match the source code in the repository. The pipelines generate deployment-ready software releases, which pass through both static code analysis and unit tests before automatically being deployed to short and long-term repositories.

The tool chain leverages industry standard technologies, such as GitLab, Docker and Nexus. The technologies chosen for the tool chain are well understood and have a long, solid track record, reducing the effort in maintenance and potential long term risk. The setup has reduced the expert time needed for testing and releases, while improving the release quality.

INTRODUCTION

The CERN Industrial Control Systems Group of the Beams Department (CERN BE/ICS) [1] provides support and software solutions to the SIMATIC WinCC Open Architecture (WinCC OA) [2] community for setting up SCADA applications. This includes general infrastructure (cryogenics, electricity, radiation protection, etc) as well as controls for the experiments, and associated institutes. The group provides a CERN specific distribution of WinCC OA, repackaging the software released by ETM Professional Control, a Siemens AG subsidiary. In addition, the group develops and maintains WinCC OA applications, as well as frameworks for building such applications, and for connecting them to the CERN IT infrastructure.

The WinCC OA software catalogue supported by the group spans hundreds of applications and two frameworks composed of a large set of components, totalling up to millions of lines of source code written in multiple languages, including C++, PL/SQL, CTRL – which is a WinCC OA proprietary scripting language, and others. Many of these projects are required to run on both Linux and Windows, increasing their complexity.

While the code itself has always been kept in a central repository, the procedures for building a release, including compilation, packaging and deployment to our package repositories, were left to the developer in charge of each individual project. This approach meant that each component was built in a different way, and releases were only possible to be done when the expert was present. Much of this was standardized by the introduction of ARES [3], yet the approach taken was in a way upside down, requiring many manual steps for releases, leading to automated commits to the repository. This required developers to change contexts each time a release was to be prepared, be it for validation or for production, and the automated commits resulted in unnecessary pollution of the commit log, making it hard to read.

While ARES greatly simplified and automated releases, there was a disconnect between development and releases. The tools used for the release infrastructure were based on technologies poorly understood by framework developers, which lead to recurrent delays while debugging issues as the ARES expert intervention was required. The tooling, while advanced, was designed with Java projects in mind, which differ radically from WinCC OA and Frameworks development.

To alleviate these issues, a deep look was taken into the development processes, and based on it, a new set of tooling was designed and put in place. The resulting DevOps processes and CI/CD infrastructure are described in the next sections.

WORKFLOW

Deciding on a workflow (see Fig. 1) for development with git, the industry "standard" git flow was adopted, later morphing closer to the GitHub flow [4]. For any given project the master branch is considered stable, yet not necessarily production quality. There is no development branch, simply bug fix and feature branches, all of which tend to be short-lived. The frequent merges promote targeted and granular development, reducing the occurrence and probability of complicated and time-consuming merge conflicts.

To start development for a bug fix or a feature, the developer manually creates a ticket in the ticketing system. The name of that ticket is then used as the branch name. On that branch each commit produces a release that can be passed on for validation by users. Once a merge request is created, unit tests are automatically run, and each subsequent push will trigger a new set of tests to be executed. After a review and with the tests successful, the branch is merged into master, resulting in a snapshot build. A tag produces a release that is automatically deployed into an array of repositories. All the steps of the process can be performed either from the command line interface, or from a single browser tab.

For a developer this is a natural workflow, while GitLab [5] allows for less technically minded people to create releases as well, after a short training session.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV050

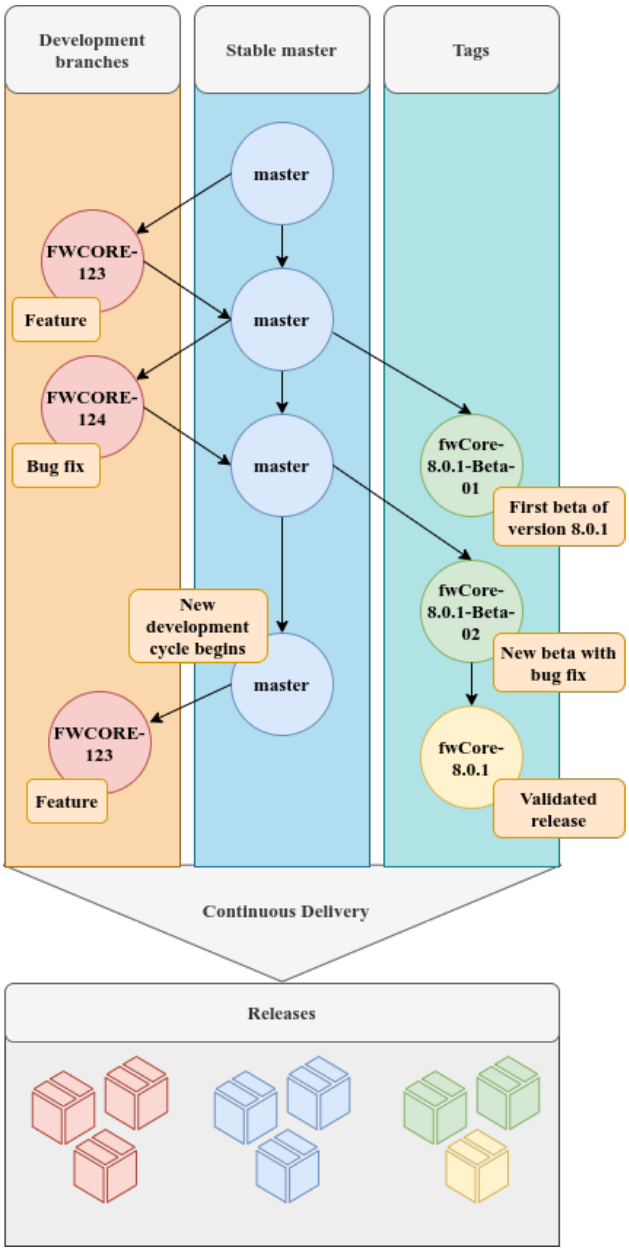

Figure 1: Example of the workflow.

RELEASE GENERATION

RELEASE GENERATION The automation of the processes is achieved by use of GitLab Pipelines, executing jobs on private hosts with Docker [6] executors, which run custom Linux images loaded with WinCC OA and a set of custom tooling. For certain quality assurance tasks other images are used to leverage newer tools. These images themselves are produced also with GitLab on the very same runners, as explained below.

CERN WinCC Open Architecture

The CERN WinCC OA Service repackages WinCC OA for Windows and Linux, in order to add in certain CERN-wide customizations – including in house license fulfilment and Oracle Database integration. This repackaged distribution is provided to the CERN WinCC OA community. As a future part of critical infrastructure, each release must be thoroughly validated to meet the highest QA standards. For this, the software package is installed on a clean production-like system, and specific common operations are executed, including creating applications and running processes. The validation has been completely automated for Linux with the use of containers, and as a final product of each automated release pipeline, a set of development and test container images are released. These are then used to compile executables developed at CERN, build components, and test further integrations.

CERN WinCC OA Framework Components

The compilation and packaging process of WinCC OA Framework components often requires careful setting up of environments, making it close to impossible without the component expert present. This problem was furthermore difficult to tackle since in the past the binaries were all stored in the version control system.

As a part of the migration to git and GitLab, all binaries were stripped from the repositories and their compilation was shifted to the release pipeline, utilizing the standard development container image. Any libraries and other build time dependencies thus must be pulled in automatically during the build, removing dependency on a single developer's environment, or their knowledge on how to recreate it.

Building the binaries from source just prior to packaging ensures they match the source code in the repository, and that no local changes make it to any component or framework release. Each release is tagged with the git revision hash, and the source code is publicly available.

After compilation, the release pipeline (see Fig. 2) passes on to the build stage, where the tooling generates documentation and manuals, updates the component XML specification used for installing them, and packages it all into a zip file. Passing into the test stage, several QA jobs are run to find issues earlier in the development cycle, producing a stable and well tested set of components for deployment.

Kobe

Kobe is the tool built to automate framework and component releases. It was originally envisioned as a leaner version of ARES, allowing for continuous builds either on GitLab or locally on the developer's system. It is built on common, historically stable and commonly used GNU/Linux tools and bases its logic on information available in the repository and in the pipeline. It has since its conception grown to do much more, including generation of Online Qt documentation integrated into WinCC OA, as well as rich PDF manuals, using LaTeX templates, and Markdown.

Kobe produces fully framework compatible components with a guaranteed unique version number for each branch, tag and revision. Releases follow a specific format with the component or framework name, the version number and an optional beta or release candidate suffix. The lack of either

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV050

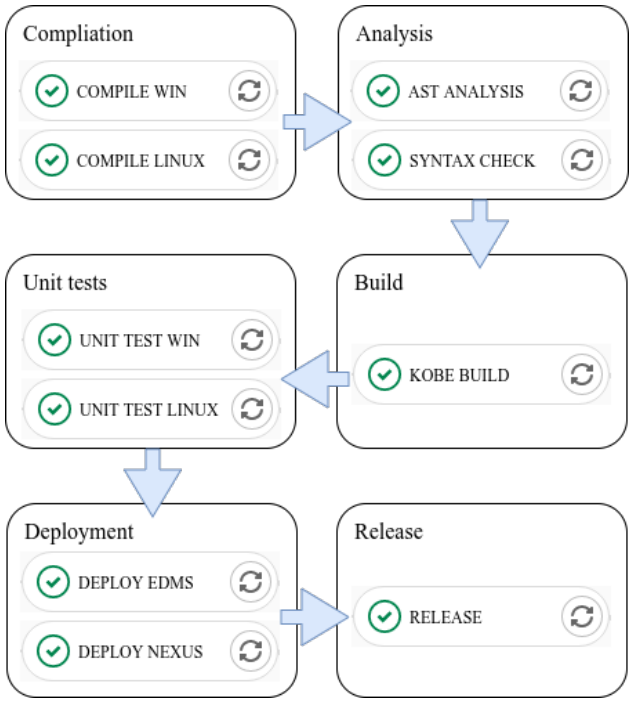

Figure 2: Example of a release pipeline.

suffix indicates a final release of that version, leading to the packages of components from these pipelines getting labelled accordingly, and to be released in to the appropriate repositories – after having passed through automated tests. These unique versions and suffixes allow dependable identification of deployed versions, thus reducing the ambiguity what has been deployed.

CERN WinCC OA Frameworks

Frameworks are built in very much the same way, simply as a collection of components. The collections are specified with a configuration file and Kobe automatically pulls the the components from the appropriate repositories. This could be from a GitLab job, a Nexus [7] repository or from EDMS [8].

During active development, the latest snapshot builds are included into the framework snapshot distributions, which are triggered at the end of each component snapshot pipeline. To ensure the possibility to generate new versions of the full framework at any time in the future, the versions included for framework beta and final releases are locked and pulled from mid- or long-term repositories.

QUALITY ASSURANCE

Quality assurance is an integral part of the CI/CD pipeline, and includes both static analysis as well as unit tests. For C++, abstract syntax trees are analysed against thousands of carefully selected rules from publicly available rule sets, including MISRA [9], HIC++ [10] and Fuchsia [11], among others, and a report is produced and made available in a centralized reporting tool. This allows us to identify hidden bugs and vulnerabilities, as well as code duplication and smells, all with historical trends. As abstract syntax trees are only possible with compiled languages, a different approach is taken for CTRL by analysing it with WinCC OA's own syntax checker. The results are transformed into standards compliant reports, visible with details directly in the GitLab Pipeline UI. In the future it is planned to have these also integrated into centralized reporting tool.

Unit tests implemented according to specifications drawn up together with the component responsible and users' feedback are included in weekly containerized component tests, daily full stack integration tests on physical machines as well as triggered full framework tests of the JCOP framework [12] and the Unified Industrial Control System and Continuous Process Control framework (UNICOS-CPC) [13]. A CERN custom interface to WinCC OA and a framework unit test component are used to automate the tests. A clean application is instantiated in a container, with the set of components to be tested. The test results generated by the unit test component are likewise in standards compliant format and thus visualized directly on GitLab Pipeline UI. They are also fed into the centralized reporting tool, improving their visibility.

DEPLOYMENT

Once packaged and tested, components and frameworks are deployed to repositories. Each repository serves its own purpose, and depending on the type of release, it is routed to a different one, completely automatically.

GitLab

For active development, GitLab job artifacts act as the short-term repository. With GitLab API [14] they can be pulled programmatically either to a project, or to build a framework. Based on Amazon S3 [14] compatible object storage, the releases are quickly available, and are stored generally for a week. The artefacts of the latest job are stored indefinitely, as are the artefacts of tags – i.e. betas, release candidates and final releases.

Nexus

All betas, release candidates and final releases are automatically deployed to Sonatype Nexus repositories. We do not take advantage of staging repositories, as that purpose is served by GitLab's object storage. Nexus repository is a mid-term repository and is available in parts of the CERN network, but not outside of it, nor is it available in the experiment networks. Thus, it is mainly used for validation and pre-production tests.

EDMS

Final releases are deployed to EDMS, our most long term document repository (more than 30 years). It is also the most accessible one, available in all CERN networks – experiments included – and outside of CERN. The ARES EDMS client is used for the deployments to EDMS, as it greatly simplifies the tasks needed to follow EDMS release workflows. Improvements to the tool have been made to allow

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV050

better diagnostics of deployments and the document statuses, increasing its robustness.

NFS

According to previously mentioned requirements we will now provide details about hardware used and specifications.

INFRASTRUCTURE

GitLab, provided as a service by CERN IT Department, is used as the backbone of the CI/CD infrastructure as illustrated on Fig. 3. The integrated container registry is used to store container images, and the UI is used to configure all of it. CERN GitLab Service also provides public runners, which are sufficient for most jobs. WinCC OA development and testing however have very specific requirements, necessitating the use of private runners.

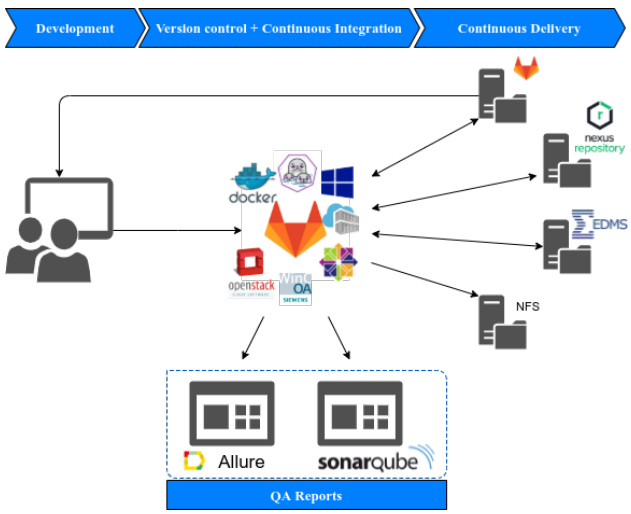

Figure 3: CI/CD infrastructure.

For lightweight jobs, OpenStack [16] Linux and Windows virtual machines are used. Setting up private runners on them guarantees availability for our frequent jobs as well as optimizes the pulling of containers – which are fairly large when loaded with WinCC OA. This has also allowed exploratory work towards alternative container based custom executors which will be necessary for the future operating systems.

For heavier tasks bare metal servers are necessary. Repackaging of WinCC OA distributions for Windows and Linux are I/O heavy processes that take over an hour on virtual machines, yet happen in just minutes on dedicated hosts. Our full stack integration tests benefits as well from their dedicated bare metal runner, as the tests require a full production-like system with connected devices, including PLCs, to run tests on multiple levels, from low-level front-ends to middleware and SCADA integration.

Exposing a bare metal system with a shell runner to a common GitLab instance is a compromise in security, and great care is taken to strictly restrict their access.

CONCLUSION

Approximately twenty-five thousand framework snap-shot releases, triggered by a matching twenty-five thousand component snapshot releases, and hundreds of thousands of more development releases have been produced since we started deploying Kobe build infrastructure in the late summer of 2018; not counting hundreds of betas, release candidates and production releases of component and frameworks. All of these releases have passed automatic tests in containers running the latest releases of WinCC OA, which themselves were generated and tested automatically by our CI/CD tooling.

This is an increase of two orders of magnitude from the old automatic release service, while reducing the effort for the developers to close to zero, saving thousands of hours of work only in releases, not to mention the increase in quality of the software. What used to be an error prone and complicated procedure involving multiple different web applications, now takes all of writing ‘git tag‘ and pushing the tag to GitLab. For snapshot and development builds, pushing the sources to GitLab is all there is to it, making it a natural part of development.

The developers thus benefit from a more natural, development-oriented workflow, while the users gain more frequent releases, shorter lead times and simply a better tested end product, all with publicly available reports and up to date documentation.

The quality of support has also improved as we can guarantee version uniqueness, as well as provide instant test releases for any bug fixes, to be validated in production-like environments, while being clearly tagged as development releases. Trusted users can even create merge requests, which will immediately provide them with a usable release of the component.

The added control and improved quality assurance of the strict and replicable release flow also improve the operational safety and security on the organizational level, while further security tooling can now be easily integrated.

With conventions and templates, we have arrived at a highly standardized and unified system. This is not to say everything is the same, and that there are no exceptions.

Most compilations are configured with qmake, on Windows and Linux, yet some developers have preferred to stay with pure make or CMake builds. This knowledge of build system is however abstracted away by scripts and templated pipelines, and while debugging a failing compilation might require expert knowledge, most development can be done by junior developers with limited experience.

Further customizations are also made available through environment variables, allowing for custom repository structures and non-standard naming conventions, among other things. While we approach near complete unification of our conventions in the group, as the tool is being adopted by experiments and application groups around CERN, legacy structures and conventions need to be supported, and as with any software, further improvements will al-ways be needed.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV050

The success of the project hinged on the continuous involvement of the entire team. Their contributions ensured the fit for our needs, and what was seen was a true integration of new and existing tooling and processes. The team's involvement also ensured the project's adoption, and its ever-continuing improvement.

REFERENCES

[1] CERN BE/ICS, https://be-dep-ics.web.cern.ch/

[2] SIMATIC WinCC Open Architecture, https://winccoa.com/

[3] I. Prieto Barreiro and F. Varela, "ARES: Automatic Release Service", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 503-507. doi:10.18429/JACoW-ICALEPCS2017-TUPHA049

[4] GitHub Flow, https://guides.github.com/introduction/flow/

[5] GitLab, https://about.gitlab.com/

[6] Docker, https://www.docker.com/

[7] Sonatype Nexus, http://www.sonatype.org/nexus/

[8] Engineering & Equipment Data Management Service, https://edms.cern.ch/

[9] MISRA, https://www.misra.org.uk/misra-c-plus-plus/

[10] HIC++, https://www.perforce.com/blog/qac/high-integrity-cpp-hicpp

[11] Fuchsia, https://fuchsia.dev/fuchsia-src/development/languages/c-cpp/cpp-style

[12] JCOP Framework, https://jcop.web.cern.ch/

[13] UNICOS Framework https://unicos.web.cern.ch/

[14] GitLab API, https://docs.gitlab.com/ee/api/

[15] Amazon S3, https://docs.aws.amazon.com/s3/

[16] OpenStack, https://www.openstack.org/