ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV037

ALBA CONTROLS SYSTEM SOFTWARE STACK UPGRADE

G. Cuni, F. Becheri, S. Blanch-Torné, C. Falcon-Torres, C. Pascual-Izarra,
Z. Reszela, S. Rubio-Manrique, ALBA-CELLS, Barcelona, Spain

Abstract

ALBA, a 3rd Generation Synchroton Light Source located near Barcelona in Spain, is in operation since 2012. During the last 10 years, the updates of ALBA's Control System were severely limited in order to prevent disruptions of production equipment, at the cost of having to deal with hardware and software obsolescence, elevating the effort of maintenance and enhancements. The construction of the second phase new beamlines accelerated the renewal of the software stack. In order to limit the number of supported platforms we also gradually upgraded the already operational subsystems. We are in the process of switching to the Debian OS, upgrading to the Tango 9 Control System framework including the Tango Archiving System to HDB++, migrating our code to Python 3, and migrating our GUIs to PyQt5 and PyQtGraph, etc. In order to ensure the project quality and to facilitate future upgrades, we try to automate testing, packaging, and configuration management with CI/CD pipelines using, among others, the following tools: pytest, Docker, GitLab-CI and Salt. In this paper, we present our strategy in this project, the current status of different upgrades and we share the lessons learnt.

ALBA CONTROLS SYSTEM DEVELOPMENT AND EARLY OPERATION

When building ALBA controls system, the main goal from the software (and hardware) point of view was to use standard tools and be as homogeneous as possible in order to ease development, training, and troubleshooting. Beamlines and accelerators have the same software structure and use the same applications wherever it is possible, for example, vacuum or motion control.

Controls Software Stack Overview

ALBA controls system uses Tango as middleware, a distributed control system framework based on CORBA [1]. It is characterized by a client-server architecture, its object-oriented design, and the use of the database as a broker and name service. The ALBA Controls Group chose Python as the main programming language and strongly invested in developing and supporting PyTango, a Python binding to C++ Tango library [2].

The vast majority of GUIs at ALBA are developed using Taurus [3], a library for building desktop applications in PyQt. Taurus was initially connecting only to Tango models but over years its architecture evolved towards a highly modular and data source agnostic solution broadly used in numerous scientific installations.

Apart from Taurus, other generic and transversal services, initially implemented as Tango device servers interfaced from GUIs, gradually evolved into projects used not only at ALBA beamlines and accelerators but also at many other institutes of the Tango community. These are, among others:

- Sardana, a scientific SCADA suite [4], which consists of Taurus-based widgets for experiment control and IPython based CLI called Spock on the client-side, and a powerful sequencer called MacroServer and Device Pool for interfacing with the hardware on the server-side.
- Panic, an IEC62682 compliant Alarm Handling suite (Alarm Handling Panic GUI and PyAlarm) [5] capable of messaging and automated execution of control system actions.
- Generic tools and device servers (Tango import/export scrips, calculation device servers, vacuum controllers, diagnostics tools) [6]

Apart from the generic services and user interfaces, every sub-system has its specific applications, e.g. MX-CuBE control application for macromolecular crystallography experiments used at BL13, TXM control application for tomography experiments used at BL09, or the accelerator timing system controls stack (Linux drivers, Tango device servers and GUIs).

The ALBA Controls Group used to manage all the software under maintenance with the “bliss” system. The bliss system, developed by the ESRF, is an rpm-based packaging and Software Configuration Management (SCM) tool. It comprises two applications: the blissbuilder and the blissinstaller, both offering intuitive to “non-packaging experts” graphical way of defining and creating packages, and installing them, at the same time being limited in terms of automatic package creation and deployment.

The different pieces of software run on diskless compact PCI (only for the accelerators) and industrial PCs, distributed in the service area or experimental hall with direct access to the hardware devices. The boot servers, archiving, Tango databases, CCD data acquisition, and various other services run on VMs centralized in the computing room. Most of the controls hosts run a standard Linux distribution which at the beginning was openSUSE but there are also some Windows hosts, mainly workstations for data analysis.

ALBA CONTROLS SYSTEM SOFTWARE OBSOLESCENCE

A control system for a scientific facility such as ALBA is not a static system: new hardware needs to be suppor-

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV037

ted, scientific software introduces requirements for up-to-date libraries and other dependencies, etc. Similarly, the different components of the system are heavily interconnected, which makes that, very often, the upgrade of a component requires similar upgrades in other related components.

While the initial stack of technologies and strategic decisions proved successful in the first stage of building the ALBA controls system and first years of commissioning and operation, they inevitably needed to be reviewed and updated after more than a decade of use and in the new stage of operation.

Operating System

Most of the control machines (and also the generic IT servers and even personal desktop machines) installed during the first years of ALBA were running OpenSuSE 11.1, which had already reached its end-of-life in 2011, even before ALBA entered into operation. This imposed many limitations on the support for new software and hardware, due to being tied to core libraries and modules (e.g. libc, Python, numpy, PyQt4,...) that dated back to 2008. As this situation became increasingly problematic, some selected machines, typically those accessed by end-users, were upgraded to newer SuSE versions (mostly OpenSuSE 12.1 , but also up to OpenSuSE 13.1), which alleviated some specific issues but increased the maintenance effort due to the larger number of platforms being supported.

Operating systems security updates became scarce after some years in operation, we had a big dependence on 32 bits systems that were not already supported with a lack of hardware drivers for evolving these systems into more up-to-date platforms. This situation was not sustainable, and by 2015 a task force was constituted to evaluate the different options for a general upgrade of the OS of control system machines and also on the workflows for simplifying future upgrades. As a result, in 2018 ALBA started replacing OpenSuSE with Debian in the control system machines (see more details in the "ALBA Controls System Software Upgrade" section).

Packaging System

From the beginning, it was decided that all the software should be packaged in order to be deployed on a control system machine. This included software that was directly developed by ALBA's control group but also any other software that needed to be installed and which was not available for our OS as a system package from official or third-party repositories. Furthermore, it was decided to separate the management of the software that was installed as part of the control system from that related to the "general operating system". To that effect, and in order to facilitate the packaging tasks, the "bliss" system from the ESRF was selected and custom-adapted to ALBA's own conventions. While the packages produced and managed by bliss were of the same type (RPMs) as those of the system, they were different in that they were unpackaged into non-standard paths and did not follow the same quality rules (e.g. the dependency declaration was managed by bliss and was very limited in comparison with that of the system packages). This decision made more sense in the early times of ALBA, when the standard package build and management tools from the Linux distributions were much less user friendly, but it introduced several limitations that recommended its reconsideration:

- the separation between "general system" and "control system" was often not clear (e.g. in the case of generic libraries such as hdf5 which were also dependencies of control system packages, or the case of custom kernel drivers necessary for hardware support, or system daemons which were hard to manage by bliss).
- it was difficult to have a clear view of the whole stack of software required for a given purpose since its management was splitted into different tools (standard ones for the general system and bliss for the control system)
- the custom nature of the bliss packages prevented us from collaborating with other facilities in the packaging of software (even for software developed in collaboration, such as e.g. Tango)
- while bliss had a very gentle learning curve, it being a custom solution often required using workarounds to several of its limitations, which were not properly documented but passed as "group-lore"
- with the advent of containerization, the use of a custom packaging system made it very difficult to benefit from standard container images
- the maintenance of the bliss packaging system itself was a burden for the control system group

Because of these reasons, it was decided to move towards using the native package system for the host OS and standard package management tools as part of the change to an updated OS.

Python

Most of the software developed and maintained at ALBA is based on Python. Because of our dependency on old OpenSuse 11.1 we were forced to support Python 2.6 this disincentivated the adoption of Python 3 within the ALBA controls team even after 12 years of its release in 2008. As a consequence, in 2021, after Python 2 reached its end-of-life we still have a considerable part of our software depending on Python 2. This is especially problematic because most of the core modules on which we depend had actively stopped supporting Python 2 and current Linux distributions already stopped providing them, making a gradual transition more difficult and forcing us to keep whole systems outdated because of it.

A lot of effort is being put on adapting our software to Python 3, starting in 2019 when Taurus started supporting it (with a common code base that also supported Python2), and following with Sardana, some GUIs, Tango

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV037

Device Servers, etc. mostly associated to the OS update of machines in the Control Room and Beamlines.

Tango

The development of ALBA Control System was initiated using Tango 5 on OpenSuSE 10.2 platform. Prior to starting operation it already evolved to Tango 6 over OpenSuSE 11.1 and later to Tango 7, enhancing the performance of the system and providing what has been our stable platform for 10 years. Despite of the success on the development of the control system, several limitations generated recurrent issues, especially the dependence on outdated Java packages due to old OS and the lack of performance of the OmniNotify event system (notifd), that limited the event throughput causing continuous incidences and blocking the evolution to a full event-based architecture.

An initial upgrade project was started for several ALBA beamlines, for which a hybrid Tango 7/Tango 8 system was deployed. Tango 8 introduced ZMQ as a new event system, thus increasing the theoretical performance of the control system and eliminating the need for a permanent daemon dedicated to event dispatching, which was a recurrent cause of memory leaks and event bottlenecks. This hybrid system performed poorly, due to the usage of already obsolete operating systems and the (unknown at the time) incompatibility between different versions of ZMQ. Finally, a full Tango 8 based approach was adopted for those upgraded control systems. Thid required the deployment of multiple Jenkins servers in order to automatically recompile all existing Tango Device Servers for every new release of Tango.

Although Tango 8 was finally discarded as an alternative for the main Accelerators Control System (due to the legacy OS 32 bit dependency), all the work done on CI for the Tango 8 upgrade became the base for the current CI/CD workflow, which allowed a much smoother upgrade of both OS and Tango to the latest versions (Tango 9.3.4 over Debian 9), which solved all the previous performance issues and allowed a successful migration of all control systems at ALBA. The experience during the migration process also triggered the Tango Community to move from short-term releases to Long Term Supported (LTS) releases, a move that benefited the whole community in terms of development stability and larger reusability of software stack throughout the community.

PyQt and PyQwt

Taurus, and as a consequence most GUIs at ALBA, was originally based on PyQt4 [7] for its general widgets and on PyQwt5 [8] for its plot widgets. PyQt4 reached its end of life in 2018 and PyQwt5 was by then already unmaintained (its latest release from 2011) and was never ported to PyQt5 or Python 3.

As with Python, the need to run the control GUIs on an obsolete OS shipping outdated versions of PyQt slowed the porting of our own software. Taurus started supporting PyQt5 in 2019 and switched to it as its default binding in 2020. It also started implementing alternative plotting widgets based on PyQtgraph [9] in 2017 but these only became ready to replace the old PyQwt5 ones in 2020.

After that, some GUIs (mostly the most modern ones which were already based on Taurus >= 4) could be ported with little effort, but many remain unported because they still use Qt APIs that were already deprecated in 2013. These are being gradually migrated as part of the effort to update the OS of machines in the Control Room and Beamlines.

ALBA CONTROLS SYSTEM SOFTWARE UPGRADE

New Operating System Selection

In 2015 a task force involving various members of the controls group in coordination with the IT infrastructure group started evaluating the various candidates to replace the old OpenSuSE 11 and 12 machines. One key decision was to decouple the choice of the OS for control system machines from that for generic IT systems because it became evident that, apart from a common requirement for robustness and long term maintainability, the requirements were not compatible: in the case of the control system we stressed the availability of scientific software and the support of scientific instrumentation and varied hardware platforms, while for generic IT systems the focus was on vendor support and network and filesystem robustness. As a result, OpenSuSE Leap and Debian were shortlisted as the candidates for the control system, while the IT systems moved towards CentOS.

During two years we conducted an in-depth evaluation of OpenSuSE (using versions 13.1 and 42.1) and Debian (with versions 8 and 9). The main aspects being evaluated were:

- life cycle and duration of supported updates
- amount of available software relevant for ALBA (i.e., mostly science-related). In order of preference we considered packages already included in the distribution, then those officially packaged for the distribution by its authors and then those packaged unofficially by some third-party
- usage in other similar facilities, specially within the Tango Collaboration
- relevant hardware platforms supported. In order of priority: amd64, i386 and arm
- quality of the documentation and community support
- quality of native package build and management tools

These aspects were not just evaluated from theoretical research, but also tested in practice by deploying both systems during more than a year to production environments (a relatively simple beamline and a support lab) for which we had to package and deploy a full basic stack of control system software, ranging from hardware kernel drivers to taurus-based GUIs and including also a Tango system with several device servers.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV037

This selection process has been constrained by our needs to evolve our stack of technologies, the feasibility of porting existing devices, drivers, and applications from 32 to 64 bits hardware keeping consistency between the new and legacy control systems as well as the necessity to keep our CI/CD workflows in-house to avoid dependency from external services. To be able to recompile and test all the software stack onto new platforms, multiple Jenkins servers were configured for all our existing OS at ALBA (SuSE 11/12/42 and Debian 8/9 in 32/64 bits), including physical hardware machines to test and evaluate hardware drivers compatibility and performance. It allowed to test all platforms and confirmed the necessity to keep 32 bits support for the next 5 years.

Both distributions showed that they were fit for the task, but Debian performed a bit better in most of the aspects being evaluated and had better support to all our hardware platforms, to the point that it was enough to overcome the main advantage of OpenSuSE which was that it involved less change with respect to the old system. As a consequence, Debian was chosen and, in 2018 the migration started with the upgrade of the Control Room machines to Debian 9, and followed by the machines dedicated to Sardana in the beamlines (at the moment of writing all but two). Once this is done, other systems will be gradually updated, leaving only some legacy systems out of the upgrade.

Packaging

As mentioned in the "ALBA Controls System Software Obsolescence" section, an important conclusion for the OS update was to also switch from our custom packaging system (bliss) to using standard system tools for both building and managing the control system packages.

As part of the new OS evaluation, we evaluated the "OpenSuse Build Service" [10] and the Debian packaging toolchain. Of both systems, we valued that the packaging could be managed with a workflow involving a version control system: a domain-specific one inspired in CVS in the case of OpenSuSE or git in the case of Debian [11].

Once Debian was chosen as our new OS for controls, we produced a step-by-step guide, a set of examples, and a Docker image with the whole Debian packaging toolchain in order to facilitate the transition from bliss to the standard Debian packaging workflow. We provided training to the whole group of controls developers, including hand-on sessions in which we collectively packaged the whole software stack required for the upgrading the Control Room machines.

On a first stage, the packaging workflow was done manually using the mentioned Docker image but, with the introduction of "salsa" service by Debian [12], we extended the Debian CI pipeline to automate the process [13]. Currently, the whole packaging process is done by a Gitlab CI pipeline triggered by pushing a version tag to the packaging repository and resulting in the package being automatically built for various distributions, tested and uploaded to a staging package repository. User intervention is required only for adjusting the package configuration during the creation of the first version of the package, or when the package configuration needs to be modified (e.g., for updating a dependency). These interventions can be entirely done using the web interface of Gitlab, eliminating the need of locally installing the packaging toolchain. The same toolchain is also capable of promoting the package to a production repository if the user confirms that it is ready after testing it using the staging repository.

The packages produced using this pipeline need to pass the same automated quality assurance tests used for the official Debian packages, resulting in much better quality packages than those that were created with bliss.

GUIs for Accelerators Operation

The upgrade of Graphical User Interfaces (GUI) for operation has been a critical procedure, as it has been performed gradually without interrupting the operation of the accelerator. A gradual replacement of Control Room consoles has been performed over a 2 years period, in which all legacy OpenSuSE machines have been replaced by Debian, upgrading from Taurus 3 to Taurus 4 and Tango 9 in the process.

Paradoxically, the performance increase provided by Tango 9 caused several issues on old applications, due to the higher event throughput on those applications not properly designed to manage high update rates. Replacement of old applications by newer versions was performed on various stages, following a strict schedule in which users were informed and allowed to test each application prior to its deployment in production. Several applications required changes in its management of attribute updates. Having all applications built on top of a common framework (Taurus) helped to implement the solutions required on Taurus itself, allowing the upgrade of multiple applications at once and allowing to solve issues once-for-all as they appeared providing flexibility at the application level to choose between polling or event-based refresh for each attribute.

In addition to the Tango upgrade, several other technologies were upgraded. Qt4 and Qwt libraries were replaced by Qt5 and PyQtGraph, and the database backend for both Tango database and Archiving System was upgraded from MySQL 5.0 to MariaDB 10.0 the newest event-based Tango Archiving System [14]. The adoption of Taurus 4 on Qt5, which was profoundly refactored, required to modify existing applications to use new Qt style signals, and the migration to PyQtGraph of archiving visualization tools is still an ongoing project, for which new Python 3 database extractors have been developed capable of online decimatiob of the incoming data from the high-performance event system.

The usage of Taurus framework for developing all custom applications required for day-to-day operation allowed to do a transparent upgrade of those applications developed using only generic Taurus widgets[3][15], allowing to upgrade dozens of applications without only minor modifications, mostly on unit visualization

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV037

(managed by Pint on Taurus 4) and attribute values formatting.

Tango Device Servers Upgrade

Upgrading device servers to new technologies have become one of the most difficult processes, due to the dependency on old hardware and the lack of drivers for newer operating systems. Industrial PC's devoted to hardware testing have been deployed on several ALBA laboratories for testing every upgraded device server prior to its deployment in production. Drivers have been updated whenever open source code was available, and several projects have been started to migrate DAQ acquisition cards to newer platforms.

In the meantime, most of the hardware-based control systems are still kept on legacy control system, while only ethernet-based hardware (PLCs [16], cameras, scopes) have been migrated to newer platforms using virtualized hosts. We are in the process of migrating all serial-line based device servers using serial-to-ethernet adapters, thus eliminating the need for Industrial PC's for most vacuum systems, and a new 32 bit Debian 9 image is under development to replace most of our legacy control systems. We expect to migrate two thirds of our legacy control system to Debian during the next year, but assuming that still many hardware dedicated IOCs will have to be kept on legacy versions waiting for a future hardware replacement.

For those devices already running on Debian 9 and all our purely software-based devices (database extractors, data processors, calculation and simulation devices, alarm systems) we started a project to migrate them to Python 3. The project started with the migration of the common-shared libraries (fandango [17], panic [5], pytangoarchiving) and the adoption of the new Python HL Tango API for every new development, which provides fully Python 3 compatible code.

Sardana Python 3

Sardana is used by various European institutes, mainly synchrotrons, and at ALBA it is used at the beamlines, accelerator, and auxiliary laboratories. Due to the software stack obsolescence at ALBA, the Sardana codebase was required to maintain compatibility with Python 2.6, Tango 7, PyQt4, etc. for a long time. Even if the Sardana community was well aware of the Python 2 EoL the Sardana codebase migration to Python 3 was always being delayed in favor of other developments. This changed when the ESRF announced that after its accelerator upgrade program they would like to use Sardana with Python 3.

Sardana is a final application, not a library, and it followed a different migration approach than the Taurus project. Sardana codebase was migrated to Python 3 without supporting Python 2 at the same time. Sardana users highly demand new feature requests and bug fixes that are continuously being released. So, it was in the interest of both users and developers that the setups were upgraded to use the new Python 3 compatible version ASAP. The upgrade process required the end users to migrate their Sardana plugins to Python 3 as well.

In the case of ALBA, the Sardana upgrade to Python 3 required upgrading the OS to Debian 9, Tango to version 9, and Taurus to version 4. The project scope was limited to upgrade only the strictly necessary part of the controls system to provide to the users the Sardana service running with Python 3. As a consequence, the coexistence of different software versions in different sub-systems was assumed. A transversal workforce of nine controls engineers was formed to work on the software migration to Python 3, packaging, testing, and commissioning.

The upgrade process consisted of two phases, first, the Tango database service was upgraded to Tango 9. In this process, the TANGO_HOST configuration was changed to use a DNS network alias, in order to facilitate eventual rollbacks and future upgrades. Here, two problems appeared: the Tango 7 event system started to suffer memory leaks in certain conditions and the Tango Java event system stopped working. In the second phase, the Sardana Tango servers were moved to dedicated VMs and a new workstation was prepare to run the Sardana client applications. Here, appeared problems with the Tango 9 causing deadlocks and hangs of the experiment procedures.

The upgrade process required from the users a thorough testing process, nevertheless, rollbacks were not avoided. In order to advance in the upgrade, immediate workarounds were applied and issues were escalated to the Tango community, which was always very helpful in debugging and solving problems. In the current state, all, but two ALBA first phase beamlines were successfully migrated and all the second phase beamlines were built using Sardana Python 3. Other institutes in the Sardana community are also very well advanced in the upgrade process.

Following Debian updates

The selection of Debian as a new OS for the control system was never considered as a one-time upgrade effort but was more a commitment to try to follow future Debian releases according to the needs and capabilities of the ALBA Controls Group. New Debian releases happen every two years, are supported by three years by the Debian organization, and later receives an additional two years of Long Term Support. During the three years of support, the release is updated once in a while with security fixes and fixes for important bugs. Those updates are called "point releases".

All the control system Debian hosts at ALBA, unless explicitly excluded, are upgraded to point releases during short shutdowns (several times per year). This upgrades only the non-control-system packages and the whole process is highly automated using the Software Configuration Management tool.

Due to the still not fully automated packaging creation for the current and future Debian releases, the upgrade to Debian 10 could not start as soon as it was available. Since the packaging infrastructure is now ready, soon we will start a project of upgrading the Sardana service to use De-

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV037

bian 10, which will open new possibilities in the project development thanks to the updated dependency stack. This will be a minor effort thanks to the automatic packaging and isolation of the Sardana Tango device servers on a dedicated VM.

Debian 11 was released in August 2021 and upgrading to it instead of Debian 10 has been discussed and is still an option, but the decision is strongly conditioned by the fact that Debian 11 does not provide a full Python 2 stack nor PyQt4 and therefore all software running on it needs to be ported to Python 3 and PyQt5 (or alternatively, run on virtual environments or containers).

APPLIED PRACTICES AND TOOLS

Each author should submit the PDF file and all source files (text and figures) to enable the paper to be reconstructed if there are processing difficulties.

Testing

Testing plays a very important role in the control system upgrade process, and testing strategy must be tailored to the nature and development status of each of the projects.

The SEP5 [18] introduced the first conventions for developing automatic tests with unittest module (part of the Python standard library) for Sardana and Taurus on the unit and integration level. Setting up Continuous Integration (CI) jobs for running those tests provided valuable feedback and let us avoid many new bugs and regressions before the upgrades in the production setups. Since the testing coverage is still not satisfactory manual tests are performed before every major release. Recently we decided to switch from unittest to PyTest [19] for developing automatic tests because this second one provides a better programming interface and many useful features available out-of-the-box e.g. fixtures, parameterization of tests, code patching, temporary resources context managers.

Developing automatic tests for the Tango control system requires setting up necessary resources for the needs of the tests. Our initial approach was to set up a disposable container running a Tango database with prepared instances of the Tango devices and starting and stopping Tango device servers for different classes of tests. Recently, in some tests, we started employing a different approach and use the PyTango (Multi)DeviceTestContext which allows better test control and isolation.

Automatic tests for the controls software can either interface with the real hardware or with simulators [20]. Currently, we employ this second strategy, for example, in the case of Sardana all the possible controller types e.g. motors or experimental channels are available in a simulation mode and are used during the tests. Similarly, in the case of the PyIcePAP, a Python library for interfacing the IcePAP motion control, simulated hardware is used during the tests. Furthermore, in the near future, we plan to employ automatic tests with the real hardware equipment located in our computing laboratory into our CI and in addition run nightly stress tests to discover non-easily reproducible bugs and performance degradations.

Automation and Reproducibility

The advantages of using Continuous Integration for the testing of both Taurus and Sardana have already been described in the previous section. Our experience in these cases has been unequivocally positive, because the use of CI not only improved the overall quality of the code but also enabled a much more agile collaboration as a consequence of the reduced risk of regressions. We will certainly promote the use of CI in other developments.

The use of Continuous Integration for software packaging in ALBA has also been discussed above. It is important to note that our current pipeline enables going one step further towards using a Continuous Delivery approach. Also, the fact that the packages are now artifacts of a fully reproducible pipeline whose logs are always available for inspection, has also facilitated the collaboration and support in the packaging process.

Additionally to the use of CI/CD for the packaging of our internal developments, we are also using it for publishing some of our software to PyPI, to Conda-Forge and other Anaconda Cloud channels, and even to salsa for inclusion in the official Debian distribution. This greatly improves the visibility and usability of our code outside ALBA.

Finally, it is also interesting to mention another area in which we found automation to be very helpful: the generation of documentation for our projects. In the case of Taurus and Sardana, we originally started by generating the documentation of the Taurus and Sardana projects using sphinx, and soon we automated it thanks to the ReadTheDocs service which we found excellent. However, we eventually decided to move the documentation build to the same CI system that we used for the code (first, Travis CI and now Gitlab CI [21]) in order to unify the CI. Apart from these projects, we also use Gitlab CI for autogenerating documentation of internal software and deploy it automatically.

Configuration

The ALBA Controls Group uses Salt [17] to configure and manage software on remote nodes. We are moving to the central Salt Master the following tasks: the installation of Debian packages, the clones of Git repositories, the installations with Pip, and the installation and configuration of the Conda environments. We define a catalog of ALBA Services e.g.: Tango, Sardana, Taurus, Icepapcms, Archiving, etc. The application of the Salt recipes allows us to automate the installation and configuration of the same Services in parallel in different machines and in a reproducible way. Each remote node is configured through Salt Grains that specify which Services should be installed in that node.

We are still in the process to fully adapt the Salt solution to ALBA controls system and we have some open points, like the best use of version control of the services,

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV037

the proper workflow for testing a new version, or the use of a graphical user interface.

One particularly problematic point is how to manage rollbacks: the Salt recipes express the changes to be done in the target machine, which is not exactly equivalent to specify the state of the system. It is easy to conceive a situation in which reverting a salt change leaves the system in a state which is not identical to that previous to the original application. For the moment, the use of "snapshots" in the case of Virtual Machines seems the best workaround.

Control Room Scheduled Updates

The upgrade of existing applications at ALBA Control Room required a careful approach, as it needed to guarantee maximum availability and stability for the operation of ALBA Accelerators. It required careful testing of every application, including hardware interaction, before applying any update to the main operator consoles. The need for hardware tests also forced us to integrate the testing procedures with the Accelerators Operation calendar, thus constraining the CI/CD workflow and defining our Control Room Update Flow states:

- Testing prior to deployment: Developers perform tests of GUI applications and device servers under development (from git or staging repositories) on a dedicated machine, only during machine maintenance periods (8-9 dedicated weeks per year).
- Testing on production: Packages are deployed on a machine available to machine operators that can be used to validate the latest releases (staging) of each software package prior to its propagation to the Control Room operators consoles. Applications are kept on this stage for at least a machine-run period (4 to 5 weeks) prior to their promotion to the production environment (only after validation by users).
- Production: Finally, validated applications available in the production repository are deployed into the 10 operator consoles in the Control Room for its use by all operators. Prior to the update, all previous stable versions of packages are deployed into an "old_stable" machine, available as a backup in case a bug appears on operation despite all the previous testing.

This workflow implies a delay of 5-6 weeks before a development reaches its production status, a conservative approach in order to guarantee the maximum stability of the system. In the case hot-fixes are required, a special mechanism to perform updates during machine days is enabled (once per week), which requires a notification to affected users via an Accelerator Interventions ticketing system and an update of the configuration system (Salt recipes), which is managed by a git repository thus providing a register of changes.

Containerization

Up to now our use of (Docker) containers has been mainly limited to the context of software development as, e.g.:

- providing the environments for the CI jobs
- providing a pre-configured a clean environment for manual testing or packaging of software
- developing or debugging in a reproducible environment

However, we are also considering the use of containers for deploying the control system in ALBA. In particular, we are currently evaluating how to deploy our typical beamline stack of applications as microservices in a Kubernetes cluster. Some of the expected potential advantages of this approach are:

- it could facilitate upgrades, thanks to the increased isolation of each component
- rollbacks would be much easier since the whole system can be re-deployed from scratch from a version-controlled configuration
- it would simplify the debugging thanks to the possibility of running clones of a given deployment

TOWARDS ALBA II

The recently approved ALBA II project will consist of an upgrade of our installation to the 4th generation class of synchrotron light source and is planned to happen in 2028. The new accelerator will provide lower emittance and higher brilliance beam to the beamlines and new challenges to the control system. The ALBA Computing Division started preparing for the future requirements by analyzing the lessons learnt from the ALBA construction and operation as well as identifying and overviewing cutting-edge solutions at similar facilities. In the following years, it is planned to start exploratory projects to acquire the necessary expertise, at least, in the following mainstream technologies which if applied could ensure long-term maintainability of the ALBA II control system. First, isolation of the controls software stack from the host OS, by use of containers or isolated environments e.g. conda. Second, use of web technologies for operators GUIs, which in comparison to desktop applications are cross-platform compatible and more manageable (isolation is done on the server placing minimal requirements on the end-user workstation). We also look forward to the upcoming Tango 10 version release, which code will be refactored/rewritten in order to make it immune to the obsolescence of libraries and technologies e.g. CORBA.

In parallel, some more specific projects for the ALBA II control system have already been started. Here we would like to mention a joint effort between BESSY, DESY, and ALBA in the development of the third harmonic cavity with the DLLRF at 1.5GHz. From the side of controls, accelerators provides us with interface information to the FPGA they program. We have developed a Python binding using cython to access this FPGA and the rest of the cards of the MicroTCA used for

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV037

this system. This binding is used to simplify their development as well as to prepare an Agent in the DCS to represent this card. The interface of registers provided by accelerators is described in the csv file and this file is used as a source for an auto-generation tool capable to build not only the Tango device server but also the Taurus GUI, based on conda environments using Python 3.9. The current development is installed in an autonomous rack to be moved to even another facility to be used. In the installation, there is a pair of containers with the Tango frontend and the MariaDB backend, and the Tango control and the Taurus GUI work on their own conda environments.

CONCLUSIONS

Decisions from the ALBA control system development, commissioning and early operation were successful in terms of the selected technologies e.g.: Tango, Python, PyQt, Tango, etc. From the perspective of time, we believe that it is crucial to keep the OS updated. In our case, the lack of update in the OS conditioned many other updates and, in the long term, generated more efforts in workarounds than the effort of keeping it up to date. Also, it generated a huge effort when it had to be finally updated. The big turnover in the team composition when transitioning from the development into the operation phase was probably decisive in postponing this effort.

Keeping the controls system up-to-date is a collective responsibility of the whole team. Decisions at different levels, from day-to-day to strategic, should be taken considering the long-term maintainability of the control system. Last but not least, it is highly recommended to continuously follow and explore emerging technologies in order to propose improvements, feel self-confident and determined in proposing and conducting upgrade projects.

ACKNOWLEDGMENTS

All the work presented in this paper is an effort of the ALBA Control Seccion members. We would like to mention here: Jordi Andreu, Manuel Broseta [on leave], Tiago Coutinho [on leave], Roberto Homs, Gabriel Jover [on leave], Jairo Moldes, Daniel Roldan [on leave], and Marc Rosanes [on leave]. The upgrade projects would not be possible without support from other sections of the Computing Division, mainly IT Systems Section, here we would like to acknowledge Sergi Puso for his support in multiple fields and Marc Rodriguez [on leave] for sharing his expertise with us about Salt, and the MIS Section, here we would like to acknowledge Daniel Sanchez for his support in the GitLab administration. The ALBA Accelerators and Beamlines Divisions, as users of the control system, played an important role in the controls system upgrade projects by performing acceptance tests and provided valuable feedback. We would like to recognize the help provided by the Tango Controls Community members (a list of whom would be too large to fit into this paper). Finally, we would like to acknowledge the help from the Debian Science Team, and especially Frederic Picca in guiding us in the process of adopting the Debian packaging policies.

REFERENCES

[1] Tango, https://tango-controls.org
[2] PyTango, https://pytango.readthedocs.io
[3] C. Pascual-Izarra *et al.*, "Effortless Creation of Control & Data Acquisition Graphical User Interfaces with Taurus", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 1138-1142. doi:10.18429/JACoW-ICALEPCS2015-THHC3003
[4] T. M. Coutinho *et al.*, "Sardana: The Software for Building SCADAS in Scientific Environments", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper WEAAUST01, pp. 607-609.
[5] S. Rubio-Manrique, G. Cuni, D. Fernandez-Carreiras, and G. Scalamera, "PANIC and the Evolution of Tango Alarm Handlers", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 170-175. doi:10.18429/JACoW-ICALEPCS2017-TUBPL03
[6] S. Rubio-Manrique, T. Coutinho, R. Suñé, and E. T. Taurel, "Dynamic Attributes and Other Functional Flexibilities of PyTango", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THP079, pp. 824-826.
[7] PyQt, https://www.riverbankcomputing.com/software/pyqt/
[8] PyQwt5, http://pyqwt.sourceforge.net/
[9] PyQtGraph, https://www.pyqtgraph.org/
[10] OBS, https://build.opensuse.org/
[11] gbp, https://wiki.debian.org/PackagingWithGit
[12] salsa, https://salsa.debian.org
[13] C. Pascual-Izarra et al., "Collaborative and Automated Packaging", 32nd Tango Meeting, Prague, https://indico.eli-beams.eu/event/310/contributions/738/attachments/547/700/CollabPkg.pdf
[14] L. Pivetta *et al.*, "New Developments for the HDB++ TANGO Archiving System", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 801-805. doi:10.18429/JACoW-ICALEPCS2017-TUPHA166
[15] S. Rubio et al., , "Unifying All TANGO Control Services in a Customizable Graphical User Interface", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 1052-1055. doi:10.18429/JACoW-ICALEPCS2015-WEPGF148
[16] PyPLC, https://gitlab.com/tango-controls/PyPLC
[17] Fandango, https://gitlab.com/tango-controls/fandango
[18] SEP5, https://github.com/sardana-org/sardana/blob/develop/doc/source/sep/SEP5.md
[19] PyTest, pytest.org
[20] S. Rubio-Manrique *et al.*, "Reproduce Anything, Anywhere: A Generic Simulation Suite for Tango Control Systems", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 280-284. doi:10.18429/JACoW-ICALEPCS2017-TUDPL01
[21] A. Götz et al., , "Migration of Tango Controls Source Code Repositories", presented at the ICALEPCS'21, Shanghai, China, Oct. 2021, paper MOPV034, this conference.