CONTINUOUS DELIVERY AT SOLEIL

G. Abeillé, A. Buteau, X. Elattaoui, S. Lê, Synchrotron SOLEIL, Gif-sur-Yvette, France
G. Boissinot, Zenika, Paris, France

Abstract

IT Department of Synchrotron SOLEIL [1] is structured along of a team of software developers responsible for the development and maintenance of all software from hardware controls up to supervision applications. With a very heterogonous development environment such as, several software languages, strongly coupled components and an increasing number of releases of the entire software stacks, it has become mandatory to standardize the entire development process through a "Continuous Delivery approach"; making it easy to release and deploy on time at any time. We achieved our objectives by building up a Continuous Delivery solution around two aspects, Deployment Pipeline [2] and DevOps [3].

A deployment pipeline is achievable by extensively automating all stages of the delivery process (the continuous integration of software, the binaries build and the integration tests). Another key point of Continuous Delivery is also a close collaboration between software developers and system administrators, often known as the DevOps movement.

This paper details the feedbacks on how we have adopted this Continuous Delivery approach, modifying our daily development team life and give an overview of the future steps.

SOLEIL CONCERNS&ANALYSIS

Synchrotron SOLEIL is a service center that delivers photons to external scientific users. The 24/7 operation of the Synchrotron SOLEIL facility relies on developed software by the "Control and Acquisition Software" team. This team has to regularly deliver changes requested by the business; it has become a challenge to perform it in the right timeframe. That is why the Continuous Integration and Delivery concepts have been studied. The main objective of these concepts is to automate every single process from the code building up to its operation in production.

Continuous Integration

Continuous Integration is a practice for detecting software defects at the earliest stage possible within the full development cycle to limit them via the automation of every step. Continuous Integration relies on key elements such as:

- *Version Control*: everything must be checked in to a single version control repository: code, tests, build and deployment scripts…
- *Automated Build*: Every project must be built and tested from the command line and must be runnable into the Continuous Integration environment.
- *Team agreement*: The commitment of the whole development team is required. Everyone must frequently check in small changes; and agree that the highest priority task is to fix any change that breaks the application.

Continuous Delivery

Continuous Delivery goes a step further. It assures to deliver a fully working and tested software in small incremental chunks to the production platform. It enables frequent release and deployment. All phases of a software product lifecycle from its coding up to production maintenance have to be automated (Fig. 1).

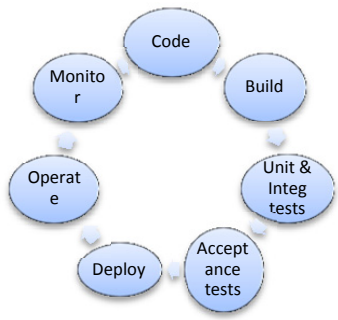

Figure 1: Continuous delivery.

DevOps

Soleil DevOps philosophy strengthens up a collaborative mode between the Development and Operations staff to achieve the same objectives: provide a full set of IT services meeting both business and IT requirements in terms of functionalities, responsiveness, and availability. DevOps is the mandatory journey to obtain a fully operational continuous delivery process.

Expected Benefits

The main expected benefits for SOLEIL of these methods are:

- Reduce all potential error-prone manual actions
- Reduce time to delivery; improve the reproducibility, repeatability and reliability of the delivery process.
- Federate the team around a common work process
- Focus on coding business requirements rather than on tooling
- Improve the synchronisation between change management and operation; reduce the number of incidents

ISBN 978-3-95450-148-9

SOLEIL CONTEXT

Figures

The Soleil Control and Acquisition Software team is composed by:

- 4 full time equivalent Java developers
- 3 full time equivalent C++ developers
- 1 full time equivalent for system administration
- 1 full time equivalent for quality assurance

This team has to manage:

- 380 software modules in Java or C++.
- Production under Linux or Windows
- All source code and scripts are checked-in various version control repositories: 1 local *CVS* server and 3 remote *SVN* servers.
- The dependency management and build tool used for Java and C++ codes is *Maven 2* [4]. All code is continuously built and monitored within a "scheduler" web server called *Jenkins* [5]. It contains approximately 500 jobs.
- All the artifacts produced by Jenkins are stored in an artifact repository called *Nexus* [6]

The current CI platform architecture is the following (Fig.2):

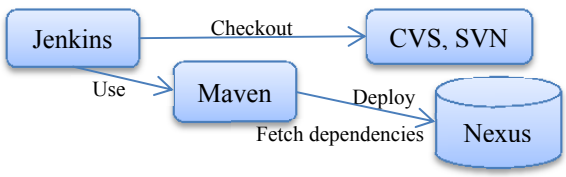

Figure 2: Current CI platform.

Each member of the team has to manage a very high number of components and their dependencies. Figure 3 is dependency tree example that illustrates the complexity of mastering a module's dependencies:

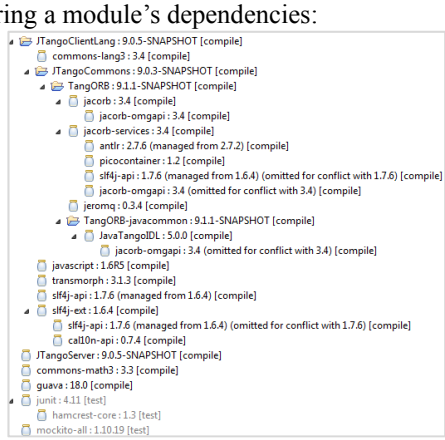

Figure 3: Dependency tree of a SOLEIL Java module.

Only a subset of our projects contains unit and integration tests.

"Zip" packages are assembled with Maven and delivered at every Machine shutdown (~ every 2 months) on 30 production platforms. The deployment process is fully automated with Linux shell scripts. The packages deployment interrupts each production platform for approximately 2 hours. Very basic acceptance tests are run manually directly after deployment during this time-slot.

The production platforms have been standardized almost everywhere:

- All the Linux servers have been just virtualised with the container technology OpenVZ [7]. There are 145 containers in operation.
- 200 industrial PC that controls hardware with windows.
- 130 "exotic" PCs that are imposed by suppliers of detectors, power supplies…

Real Life Situation

With such heterogeneity, it is a daily challenge to maintain such a Continuous Integration platform. As the main issue of the current solution is due on its global complexity, a limited number of admin are able to administer this platform.

The number of failed jobs in Jenkins is still high as some developers are considering Jenkins notification emails just as spam emails.

The current solution suffers also from technical issues, i.e. we are stuck with legacy versions of the tools because the upgrade is far from trivial.

Moreover, some parts of the C++ modules are not even considered in the continuous integration system as some Maven C++ plugins are not fully compliant (i.e. mixing compiler versions or 32/64 platforms).

Our Continuous Delivery process highlights the complexity of our software and infrastructure which has required an audit of the current process. Some aspects have been totally revamped as detailed hereafter.

OUR CONTINUOUS DELIVERY SOLUTION

Some key aspects of the delivery pipeline must be reviewed considering both methodological and technical aspects.

Team Collaboration

A fully operational deployment pipeline relies primarily on the commitment of the entire development team. This aspect has been quite underestimated and it is essential to emphasize it. A background activity is now being driven to convince the team that improving the quality of their deliverables should be the highest priority. A first step is to encourage everyone to discuss and share on technical aspects or on work methodologies either during organized meetings or stand-up-meetings. Furthermore, SOLEIL has the chance that the Control and Acquisition Software team is a mix of both developers and system administrators, but there is still room for improvement regarding the DevOps movement:

ISBN 978-3-95450-148-9

- System administrators need to get closer to the business requirements for efficiently managing the infrastructure.
- Developers need to get closer to the operation so that it becomes a reflex to include key operational indicators and logging information for incident analysis when developing a software service.

The continuous delivery process also introduces a new central role for the quality assurance staff that is that a daily challenge considering the small size of the team and the business pressure for delivering. The quality assurance manager have to take in charge the building of the team cohesion, collaboration, communication to change for adopting the Continuous Delivery paradigm. He must have also a full DevOps profile to manage the whole deployment pipeline from assistance and training of developers to the administration and monitoring of the Continuous Delivery platform. And finally, he should constantly review and ensure that the delivery demands are respected.

Another key success factor is the involvement of the business since it is directly impacted by some aspects of the Continuous Delivery process (i.e. the deployment and the operation of our software deliverables). Exposing our process and our commitment through a service agreement enables the business to be part of the process of change management to validate, accept and plan every delivery.

Change Management

Today, many incidents are still raised due to direct modifications in the production environment. Forbid this bad habits and force a strict change management process are keys to limit incidents in production. Any change has to be tested before it goes live. Only an approved deliverable by both IT and business can be shipped to the production environment.

Simplification

For maintaining the whole Continuous Delivery platform at a reasonable total cost of ownership, the only solution is to reduce the number of parameters like:

- the number of software components;
- the number of target platforms;
- the number of Version Control System.

But the path towards simplification is tricky as it implies a deep reconsideration of our practices. Moreover, the cost of simplification can be huge.

Build lifecycle and Dependency Management

A software module metadata is split in various locations:

- Inside the build descriptor on the Version Control System
- Inside the project job configuration in Jenkins

To manage the large set of components, it is mandatory to provide a central service that manages the information and lifecycle of a project, for example:

- Version Control System information
- Authors: e-mail addresses
- Status: OPEN, CLOSED

To afford developers a simple way for managing their project dependencies, the only solution for SOLEIL is to enforce a “LATEST” strategy meaning that all retrieved dependencies of a component are automatically set to the latest version. It has been made possible by using some advance features of Maven 2. But the current Maven solution suffers from issues:

- The update to the latest versions is done by a polling mechanism and it contains many caches along the build process. The latest version is not always guaranteed.
- This latest strategy relies on a deprecated Maven key word “RELEASE”.

Consequently, we cannot upgrade to the latest build tools (Maven 3 [4] or Gradle 2 [8]) because of this “LATEST strategy”. Since there is no off-the-shelf solution, the only way is to design a totally custom one.
The lifecycle of our components is imposed by Maven with only 2 statuses (SNAPSHOT and RELEASE). The review of the solution has led us defining new statuses and promotion in between. The new defined statuses are:

- *BUILD*: project under build
- *INTEGRATION*: project ready for continuous integration
- *RELEASE*: project build, integrated and tagged. Ready for packaging.

On the other hand, The Maven solution for C++ does not fulfil all the requirements. So the best solution is to move to a native tool for C++ build like for example Scons [9] or CMake [10].

User Acceptance Test (UAT) Platform

Acceptance testing ensures that the user needs, requirements and business process are met.

At SOLEIL, the only actual testing is done manually directly into production just after deployment by the IT and the business teams (when related to Machine equipment). The deployment operation is quite long and risky, with an iterative patching and re patching process until disappearance of incidents, regardless of user satisfaction.

It is obvious that an UAT platform that mirrors the production should be set. Then the continuous integration server will automatically deploy our fresh packages with the same deployment process as the production’s one. On this platform, we should at least run manual tests before establishing more and more automated tests.

Considering that 20% of our software controls hardware, with some of them unique (vacuum pumps, power supplies…), we have to develop a simulation strategy.

Deployment

Currently, only a system administrator deploys zip packages for linux and windows. A better solution is to

ISBN 978-3-95450-148-9

use the OS native packaging system such as rpm for our RedHat distributions so that libraries conflicts can be detected at the deployment time instead of runtime.

Moreover, all of our binaries are deployed on a central mounting point with Linux shell scripts. A better solution is to deploy only the necessary binaries directly of the right target with tools that allow easy large-scale deployment whilst adding consistence and compliance checks. So we need to include tools like Puppet [11], Ansible [12] or CFEngine [13]…for our deployment phase.

Monitoring

Enabling efficient and easy ways to monitor the operation of our deliveries is also one of the key success factors.

For the monitoring of the infrastructure we use Nagios [14]. Our system administrator is currently thinly tuning it to provide key performance indicators, so that every alarm will be considered as an incident that must be taken into account immediately.

Our software applications do not provide at this stage enough audit logging or monitoring indicators. We must standardize our application logging system for archiving all of them. Developers should also integrate runtime application indicators that can then be included in a monitoring system and thus raise alarms.

Our New Architecture

To address the whole technical points previously addressed, a new solution has been designed. Here is the overview of the architecture (Fig. 4):

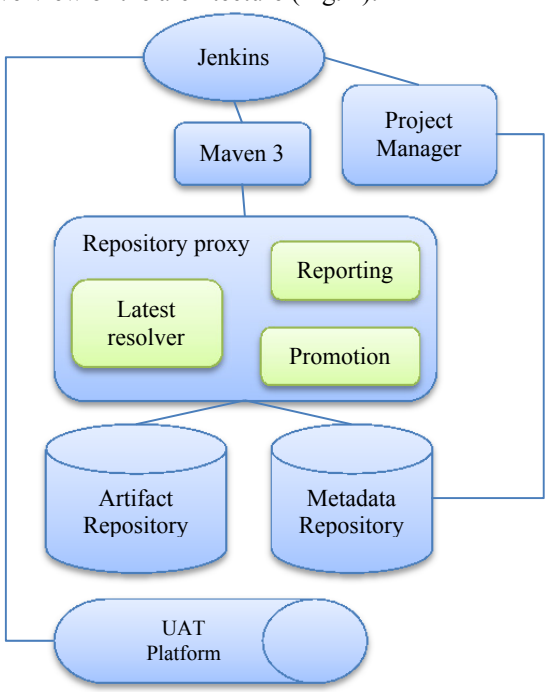

Figure 4: The new Continuous Delivery platform.

The "*Project Manager*" provides the service to create, update and close a project. This service will automatically configure the Continuous Integration server (Jenkins).

The "*Repository proxy*" is a service on top a collection of binary software artifacts and metadata. It provides some custom functions such as:

- *Latest Resolver*: it retrieves the latest version of a component.
- *Promotion*: it promotes a component ensuring the pre-defined lifecycle. For example, releasing a component will create a tag on the Version Control System and deploy the binary into the artifact repository.
- *Reporting*: provide audit reports like for example the list of promoted artifacts between two dates.

The whole solution is based on the Java technology. All services are provided as REST (REpresentational State Transfer) Web Services following the Microservices architecture [15]. The artifact repository in implemented with a MongoDB [16] document oriented NoSQL database. For the UAT automated tests, the technology choice is to be made.

Plan of Action

Putting in application the whole continuously delivery process is an enormous challenge given SOLEIL complexity and the limited resources allocated for this project. Hence, an iterative strategy has to be defined.

Firstly, the cases of Java and C++ have to be addressed separately. So a new continuous integration server will be set-up first for Java modules. The C++ case will be addressed later. Secondly, the continuous integration platform will be upgraded before moving on other aspects such as the UAT platform, the deployment and monitoring processes.

The new architecture is currently under development. The "project manager" service is fully operational and the "repository proxy" service is still under coding.

The SOLEIL infrastructure virtualization has opened up new possibilities. It is now faster and easier to set up and change the environment for development, continuous delivery, and acceptance platforms. So, the set-up and qualification of new solution platform is in progress.

The toggle to the new Continuous Integration solution for the Java part is straightforward for Java developers as the build tool is still the same. The only induced change should be in the management of the component lifecycle; the developers have to be trained on how to promote their components.

For the C++ part, the solution will depend on the workload. So the a-minima solution will keep Maven while the "deluxe" will use a native C++ build tool.

Moreover, at every new step or change, a continuous communication and training of the team must be driven to remind the IT and business teams why and how it will be applied. It is now obvious at SOLEIL that the methodologies and tool set up must follow the maturity of our team and not the other way round.

ISBN 978-3-95450-148-9

CONCLUSION

Continuous Delivery is a key activity for conducting the quality of. IT services. But, the cost for putting the process smoothly in operation is always underestimated.

The Continuous Delivery approach is holistic and challenges us on every aspect of our daily practices. It enforces to manage the total cost of ownership of a service from its inception up to its operation. A deployment pipeline automation goal is to reduce the time and cost of a delivery, so each special case can add more complexity and cost to it. Therefore there is no other solution than to standardizing and simplifying all our practices. In this context, SOLEIL is currently initiating the implementation of some ITIL practices which is a framework of best practices for managing IT services [17].

After a 10 years' experience of operation, SOLEIL still need to emphasize a close coordination between developers and system administrators following DevOps philosophy.

Continuous Delivery relies on strong convictions. You have to be intimately convinced and motivated to be able to convince the whole company including technical staff; managers; and even business users.

In conclusion, SOLEIL can finally see the light at the end of the deployment pipeline but putting in application the Continuous Delivery paradigm is really a long-winded task worthwhile the effort to obtain the requested quality level of service.

REFERENCES

[1] http://www.synchrotron-soleil.fr
[2] http://martinfowler.com/bliki/DeploymentPipeline.html
[3] https://sdarchitect.wordpress.com/2012/07/24/understanding-devops-part-1-defining-devops
[4] https://maven.apache.org
[5] https://jenkins-ci.org
[6] http://www.sonatype.org/nexus
[7] https://openvz.org
[8] http://gradle.org
[9] http://www.scons.org
[10] https://cmake.org
[11] https://puppetlabs.com
[12] http://www.ansible.com
[13] http://cfengine.com
[14] https://www.nagios.org
[15] https://www.mongodb.org
[16] http://martinfowler.com/articles/microservices.html
[17] Improving SOLEIL Computing Operation with a Service-Oriented Approach, A.Buteau, ICALECPS 2015, Melbourne, MOPGF150

ISBN 978-3-95450-148-9