The ALICE Software Release Validation cluster

This content has been downloaded from IOPscience. Please scroll down to see the full text.
(http://iopscience.iop.org/1742-6596/664/2/022006)

View the table of contents for this issue, or go to the journal homepage for more

Download details:
IP Address: 188.184.3.52
This content was downloaded on 06/01/2016 at 16:06

Please note that terms and conditions apply.
The ALICE Software Release Validation cluster

D Berzano\textsuperscript{1} and M Krzewicki\textsuperscript{2}

\textsuperscript{1} European Organization for Nuclear Research (CERN), Genève, Switzerland
\textsuperscript{2} Frankfurt Institute for Advanced Studies (FIAS), Frankfurt, Germany

E-mail: dario.berzano@cern.ch

Abstract. One of the most important steps of software lifecycle is Quality Assurance: this process comprehends both automatic tests and manual reviews, and all of them must pass successfully before the software is approved for production. Some tests, such as source code static analysis, are executed on a single dedicated service: in High Energy Physics, a full simulation and reconstruction chain on a distributed computing environment, backed with a sample “golden” dataset, is also necessary for the quality sign off. The ALICE experiment uses dedicated and virtualized computing infrastructures for the Release Validation in order not to taint the production environment (i.e. CVMFS and the Grid) with non-validated software and validation jobs: the ALICE Release Validation cluster is a disposable virtual cluster appliance based on CernVM and the Virtual Analysis Facility, capable of deploying on demand, and with a single command, a dedicated virtual HTCondor cluster with an automatically scalable number of virtual workers on any cloud supporting the standard EC2 interface. Input and output data are externally stored on EOS, and a dedicated CVMFS service is used to provide the software to be validated. We will show how the Release Validation Cluster deployment and disposal are completely transparent for the Release Manager, who simply triggers the validation from the ALICE build system’s web interface. CernVM 3, based entirely on CVMFS, permits to boot any snapshot of the operating system in time: we will show how this allows us to certify each ALICE software release for an exact CernVM snapshot, addressing the problem of Long Term Data Preservation by ensuring a consistent environment for software execution and data reprocessing in the future.

1 Overview

Ensuring quality releases of the software framework of LHC experiments is a challenging task, given the continuous evolutions and the diversity of profiles of code authors. The ALICE experiment\cite{1} adopts a validation procedure for its core framework that involves a full reconstruction and calibration from a reference raw dataset and comparison with the expected results, as well as performance benchmarks.

The validation procedure is made of several independent batch jobs, whose results are subsequently merged. Even if their batch nature would make the Grid a suitable running environment, there are however at least two reasons why we prefer to use a dedicated cluster.

First off, we cannot afford the validation procedure to run into site-specific problems, such as misconfigurations or, in general, non-controlled worker node deployments: any of such problems potentially yields false negatives making it more difficult to detect software regressions.

In addition, running the validation procedure on the Grid would imply deploying release candidates on the large scale. ALICE uses CernVM-FS\cite{2} as software deployment technology: any new published release candidate would make the central repository catalog tainted with dozens of rejected...
releases, adding an additional load on the whole system and increasing the chances of a release candidate erroneously used for a user job.

To this point it is clear that a non-chaotic infrastructure, whose configuration is thoroughly verified, should be used for running release validation jobs, the Grid not being a suitable candidate. In order to make a full release validation task entirely reproducible, the running environment must be versionable as well: we have created a portable and self-contained Release Validation Cluster, constituted of CernVM virtual machines and dynamically scalable on any cloud deployment supporting the EC2 interface [3].

In the following paragraphs we will give an overview of the various technologies contributing to the Release Validation Cluster (section 2): in particular, batch jobs run on the HTCondor [4] batch system, and elastiq [5] provides automatic scalability (section 2.1); release candidates are privately distributed using an embedded and isolated CernVM-FS server (section 2.2); the CernVM operating system [6] ensures environment consistency through snapshots (section 2.3). Secure worldwide access to both input and output data is achieved by means of the EOS [14] distributed filesystem (section 2.4).

A brief overview of the validation jobs is also provided in section 3.

The possibility to trivially run the Release Validation Cluster has been integrated with the central ALICE build server web interface (section 4); given its self-contained nature, the cluster can possibly be run manually on any cloud without the need to use the central build service. The release validation procedure has been optimized in order to be feasibly run for daily software releases (section 4.1).

2 Key technologies

The ALICE Release Validation cluster is a specific application of the CernVM-based Elastic Clusters [7], this technology being in turn an evolution of the Virtual Analysis Facility [8][9].

CernVM Elastic Clusters were made in order to address the problem of exploiting cloud resources temporarily for running an isolated task that cannot run on a single virtual machine, but needs larger batch-like resources instead. A CernVM Elastic Cluster provides cloud tenants with an all-in-one environment where batch jobs can be submitted to a preconfigured HTCondor instance. The tenant only takes care of instantiating a single CernVM virtual machine, being the head node of the virtual cluster, and she can start submitting jobs right away: when properly configured, the head node itself instantiates virtual worker nodes, and disposes of them too when they are no longer needed.

HTCondor jobs running on a CernVM Elastic Cluster do not need to be specially crafted: any HTCondor job using the “vanilla universe” can be executed on those virtual machines. Moreover, once the head node has been instantiated, Elastic Cluster users can submit jobs by being totally unaware of the underlying cloud infrastructure, as virtual machines are created and destroyed transparently.

CernVM Elastic Clusters are self-contained, meaning that no external dependency is needed, either on the user’s laptop or on the public or private cloud deployment running the virtual machines: the only basic requirements are the ability to run the CernVM image (this includes accessing CernVM-FS either directly or through a proxy) and access to the EC2 interface of the cloud of your choice. No knowledge of configuring a HTCondor cluster are required: Elastic Clusters come preconfigured, and once their task is done they can be wiped away by simply deleting its virtual machines without leaving traces behind, optimizing the way virtual resources are exploited.

ALICE is also using Elastic Clusters with success for the interactive use case in its PROOF-based Virtual Analysis Facilities [8][9], and even for running Grid jobs opportunistically on top of the new High-Level Trigger cluster [10], showing that CernVM Elastic Clusters are a simple and effective technology that can be easily adapted to different use cases.

2.1 HTCondor and elastiq

The batch system that comes preconfigured with the Release Validation Cluster is HTCondor [4]. Among its several features, we are particularly interested in HTCondor’s ability to deal with dynamic resources: notably, when a new HTCondor worker node is created, it self-registers to its configured
head node, without modifying the configuration manually and centrally, as it is the case with other batch systems (TORQUE for example [11]). Removal of workers is dealt as smoothly.

HTCondor jobs queue is monitored via elastiq [5]. This Python daemon, running on the head node only, periodically controls if HTCondor has some jobs waiting to be executed for too long (the exact amount of time is configurable): if this is the case, it requests new virtual worker nodes through the EC2 API. The elastiq daemon also checks for idle HTCondor workers: if a worker has not been running jobs for a configurable amount of time, elastiq requests its termination. Both processes are summarized in Figure 1.

**Figure 1.** The elastiq daemon monitors the HTCondor queue for waiting jobs and reacts by starting new virtual workers. Workers are also monitored and terminated if they have been idle for some time.

It is also possible to configure a minimum number of virtual workers always running: elastiq will always make sure that at least some of workers are always there, by launching new virtual machines in case they are too few. It is also possible to configure a maximum quota.

This daemon effectively implements a simple virtual machines orchestration mechanism that works on any cloud supporting the EC2 interface: by just instantiating the cluster’s head node, the rest of the cluster will be instantiated automatically and only when needed. The latter is particularly useful on commercial clouds where billing is performed on the effective number of resources used.

It is worth mentioning that elastiq features a powerful error checking and recover mechanism: virtual machines ending in error are detected and replaced, and even virtual machines that appear to have started successfully but did not join the cluster are deleted and replaced. If not enough resources are available on the cloud, elastiq will retry until the requested virtual machines are executed. Error detection also detects running virtual machines becoming faulty at some point.

As we will see in section 3, the release validation procedure is constituted of several stages, each one of them ending with a merge stage. As each merging occurs on a single job and it can take some time to complete, the number of running virtual machines might oscillate during the full validation procedure, depending on the configured idle time. On a cloud configuration that reacts quickly when new virtual machines are requested, it is convenient to keep the idle time low in order to improve resources utilization: unused virtual machines are simply terminated when unneeded.

The whole cluster runs using the same CernVM image snapshot for all virtual machines (section 2.3). After running the validation procedure we might want to dispose of the cluster in order to replace it with the most recent CernVM snapshot for the next validation test: the elastiq daemon facilitates cluster cleanup by leaving by default a single virtual machine running (the head node) when the procedure is finished.

### 2.2 A private CernVM-FS server

In order not to taint the official CernVM-FS ALICE repository with release candidates, each Release Validation Cluster has an embedded CernVM-FS server accessible only from the cluster nodes.

Every time a new release candidate is built by the ALICE build server, a tarball containing the binaries is exposed on a HTTP server. When running a new release validation task, the tarball URL is pushed to the release validation cluster head node.
A small application [12] receives the URL: it downloads the software tarball along with all the dependent packages. Everything is unpacked on the head node under a certain directory: this directory is published via CernVM-FS using the `cvmfs_server publish` command.

On complex setups, CernVM-FS allows for a tiered architecture [13]: software is published on a single server called “stratum 0”, and it is asynchronously replicated on nodes members of the “stratum 1”. Clients do not normally access stratum 1 servers directly but caching proxy servers are used, leveraging HTTP as transport protocol. This layered architecture introduces an approximate delay of 2 hours before newly published ALICE software is accessible from all client nodes in the world.

The Release Validation Cluster is a very small setup compared to the Grid, therefore it does not need such a complex architecture. The head node is a CernVM-FS stratum 0, and client nodes access it directly, without any intermediate caching proxy server.

A disposable CernVM-FS server, configured on the fly, has proven to be as easy to configure as a NFS share, with a number of advantages that turn out useful for our use case. First of all, CernVM-FS features aggressive caching on the client side, as it targets serving read-only content that never changes once published. In addition, using CernVM-FS as software distribution platform better simulates the environment a Grid job would encounter, and helps us finding potential pitfalls of software distribution before they occur on the large scale: it is possible, for instance, to test how new CernVM-FS client versions perform before they are adopted by Grid sites.

CernVM-FS implements a form of authentication based on server-side data signing using a private key: clients need the corresponding public key to authenticate received data. The CernVM-FS client package is distributed with a predefined set of keys covering the most common CernVM-FS servers used by LHC experiments.

Our CernVM-FS server is private, and its keys are generated on the fly during boot time: thanks to elastiq (section 2.1), we can inject the generated public key to the clients before they are booted by appending it to their contextualization manifest.

The publishing process and key distribution is represented in Figure 2.
2.3 CernVM base image
CernVM 3 is used as base operating system [6]. This operating system is binary compatible with SLC 6, and its most important feature is that its base image is less than 20 MB large, as the root filesystem is mounted from CernVM-FS and therefore single files are downloaded on demand.

Every new CernVM update implies a new snapshot in its CernVM-FS repository; by default, CernVM picks the latest version. From the release validation perspective, the CernVM snapshot is chosen explicitly when deploying the head, and cluster workers are configured with the same snapshot, in order to ensure environment consistency at cluster level. If the release validation procedure is successful, the tested release candidate will be certified for working on a certain CernVM snapshot.

CernVM snapshots are archived and will be kept forever: no snapshot will ever be deleted and will always be accessible in the future. The fact that ALICE software versions are now optionally coupled to a CernVM snapshot has a particular interest from the Long Term Data Preservation perspective: the process of rerunning the same software within the same environment in the future is greatly simplified.

2.4 EOS
The reference input dataset for the ALICE release validation is almost 2 TB large: for this reason it cannot be distributed along with the virtual machine images.

Data is made accessible using the EOS storage technology developed at CERN [14][15]. EOS can be accessed using the standard xrootd protocol. The Release Validation Cluster uses the EOS FUSE client to mount the EOS namespace and use it as if it were part of the local filesystem. EOS is mounted in read-write mode, as it is both used for accessing the input dataset and writing output results.

The approximate size of the output results ranges between 80 GB to 250 GB per validation run depending on the event size of the selected events from the reference dataset.

As output files are written on EOS, the release validation cluster can be safely destroyed when the validation is over without losing any important data. A web server, also mounting EOS locally through FUSE, has been configured to access all release validation results from anywhere in the world using only a web browser (section 3).

3 Validation workflow
The release validation procedure uses an input dataset of particularly crafted raw data for calibration called “filtered raw”. The procedure runs reconstruction algorithms twice (the first time without calibration, the second time with the calibration produced during the first step) in order to get a more precise calibration database as final output.

![Figure 3. The release validation workflow. Filtered raw files are reconstructed twice, with the second reconstruction using calibration data calculated during the first iteration. Output results are an accurate calibration database and several Event Summary Data files containing reconstructed data. Information trees for Quality Assurance are also produced. ESDs and QA trees are the output of parallel jobs, whereas calibration databases are produced during merge.](image-url)
Filtered raw data are the input for the first step of parallel batch jobs, called *calibration pass 0* or cpass0. Filtered raw data is reconstructed producing several Event Summary Data (ESD) files. A merging phase is run at the end of cpass0 to produce a single calibration database and a summary report.

The second step is called *calibration pass 1* or cpass1: filtered raw data are processed one more time by taking the calibration database produced during cpass0 into account. Several batch jobs run in parallel, each one of them producing reconstructed data in the form of ESD files like cpass0. In addition, each job also outputs two summary trees containing information used for Quality Assurance purposes—a simple generic one and a more complex one reserved to detector experts. The merge phase that follows, running on a single job, produces a second calibration database, more accurate than the first iteration, along with a summary report.

The full release validation workflow schema is depicted in Figure 3.

Each reconstruction job and both big merging jobs also produce HTML data as reports, with plots extracted from ROOT files: such data is published on EOS and served via a web server (section 2.4), making its quick examination possible without even opening the ROOT files. An example of the output data is presented in Figure 4.

![Figure 4](image)

**Figure 4.** Validation results are accessible from the web from a central interface. Relevant data is rendered to plots visible from a browser without the need to open ROOT.

### 3.1 Makeflow

As we have seen, and as simplified in Figure 3, the workflow is constituted by several batch jobs running in parallel and a merge step collecting results from the single jobs. This procedure is repeated twice, one for cpass0 and one for cpass1.

Output of parallel jobs is the input for merge jobs, and some of the output produced by cpass0 is used during cpass1: in order to handle these input-output dependencies we have used Makeflow [16], integrated with CernVM.

Makeflow is used by writing a single manifest where you specify exactly what is produced and what is needed by each job, using a syntax heavily inspired from GNU Makefiles, but much simpler.

The whole task is managed by Makeflow: once run, it will create an internal dependency graph and it will launch and monitor batch jobs accordingly. For instance, the merge step is launched only when the last parallel job has finished. Makeflow can also relaunch failed jobs for a certain number of times before failing.

If the dependencies are correctly specified and some output files were already produced, rerunning Makeflow will not go through all the steps, but it will only execute the ones needed to produce the missing output. This is the same that *make* does: when changing some source files, only outdated targets are rebuilt.

### 4 Running the Release Validation

A Release Validation Cluster is meant to be disposable: the cluster, including the head node, lives for the time needed to run the release validation procedure.
The cluster has been designed in order to be extremely portable: for instance, experts in the ALICE collaboration can run it on the cloud infrastructure of their choice, in order to test their software before publishing their commits. Input and output data will always be on the same EOS location (section 2.4), and accessible from the same central web interface.

A more simplified procedure has been integrated with the ALICE build server in order to make it even easier to run the procedure for the ALICE librarians. The interface is incredibly simple: using the standard ALICE web interface for queuing a new release build, the librarian can click an optional checkbox for running the release validation right after the build is complete, as visible in Figure 5.

Behind the scenes, the whole validation cluster, including the head node, is spawned on the OpenStack-based CERN Agile Infrastructure [17], and the validation procedure is executed. When the validation is finished, the cluster is destroyed and an email is sent to the reviewers. With the simplified procedure, neither the librarians nor the reviewers must have any knowledge of the virtualization technologies supporting the procedure, as the release validation is literally available with one click.

4.1 Time to results

The whole release validation procedure can be run in less than 24 hours on 50 virtual machines on the CERN Agile Infrastructure. The virtual machines are of flavor m1.large, meaning in this case 4 virtual CPUs and 8 GB RAM.

The release cycle of ALICE core software ranges from two weeks to one month, making a 24 hours run perfectly acceptable for our use case.

5 Conclusions

With this work we have addressed the two biggest problems of running quality controls on the data produced by ALICE software: an isolated and consistent sandbox where to run release validation tests, and an easy to use interface allowing librarians to effectively use it, and reviewers to examine results quickly.

The complexity of the ALICE release validation workflow has always been discouraging, with only a restricted number of specialists effectively capable of running it. This work wrapped the existing validation procedure within a self-contained package that makes it easily accessible, opening the possibility for daily tests during Run 2.

References


[16] http://ccl.cse.nd.edu/software/makeflow/