Software release build process and components in ATLAS offline.

Emil Obreshkov, on behalf of the ATLAS Collaboration

DESY, Hamburg and Zeuthen, Notkestr., D-22607 Hamburg, Germany

Email: Emil.Obreshkov@cern.ch

Abstract. ATLAS is one of the largest collaborations in the physical sciences and involves 3000 scientists and engineers from 174 institutions in 38 countries. The geographically dispersed developer community has produced a large amount of software which is organized in 10 projects. In this presentation we discuss how the software is built on a variety of compiler and operating system combinations every night. File level and package level parallelism together with multi-core build servers are used to perform fast builds of the different platforms in several branches. We discuss the different tools involved during the software release build process and also the various mechanisms implemented to provide performance gains and error detection and retry mechanisms in order to counteract network and other instabilities that would otherwise degrade the robustness of the system. The goal is to provide high quality software built as fast as possible ready for final validation and deployment.

1. Introduction

1.1. ATLAS

The ATLAS (A Toroidal LHC Apparatus) [1] project is one of the biggest collaborative efforts ever undertaken in the physical sciences: more than 3,000 scientists and engineers, approximately 174 institutions from 38 countries. The offline software community is a highly dispersed collaboration of physicists, students and computing professionals and there are more than 600 software developers, mainly part time. The ATLAS offline code base has approximately 5 million lines of code organized in around 2000 packages. The main programming languages in use are C++ and Python but also we have lots of code written in Java, Perl and Fortran. The software is built on several platforms using several compilers in order to guarantee the robustness and better validation of the produced code. Several important tools have been developed and put in place to insure robust code base maintenance, correct and rapid bug fixing, software change submission, compilation and validation on a large number of build machines, testing and finally reporting the results to the developers.

2. ATLAS offline software components

2.1. Packages

The smallest management unit inside the ATLAS offline software is the package. A package combines groups of C++ and/or Python classes and possibly scripts. Packages are the primary development and management units and they can have dependencies against classes in other packages. Typically a package defines and implements some simple functionality, which later can be used together with other packages to form more complex functionality and provide a base for development of more complex objects. Each package is defined using a Configuration Management Tool (CMT) [2] and has a “requirements” file, which describes conventions and tools for automating as much as possible the implementation of these conventions. It permits the description of the configuration requirements and automatically deduces from the description the effective set of configuration parameters needed to build and execute the packages. In a given ATLAS offline release there are ~2000 packages and currently there are approximately 200 package changes per week (in all active nightly builds).

ATLAS offline also uses some packages which are supplied by external software providers [3].

2.2. Projects

A package is always maintained in a single project and the organization of projects and packages serves to both define and provide a logical structure for software dependencies. A “Project” is a group of packages that can be built together because there are similar dependencies between the packages inside a given project. They are organized to reflect the order of the ATLAS offline domain specific chain. Examples of projects are AtlasCore, AtlasConditions, AtlasReconstruction, AtlasAnalysis etc. Projects are the primary release coordination units and for each project we have one or sometimes several release coordinators. There are 10 specific offline base projects. Some of the projects are externally supplied and provide external packages. Figure 1 shows the project dependencies for a full release 16.0.1

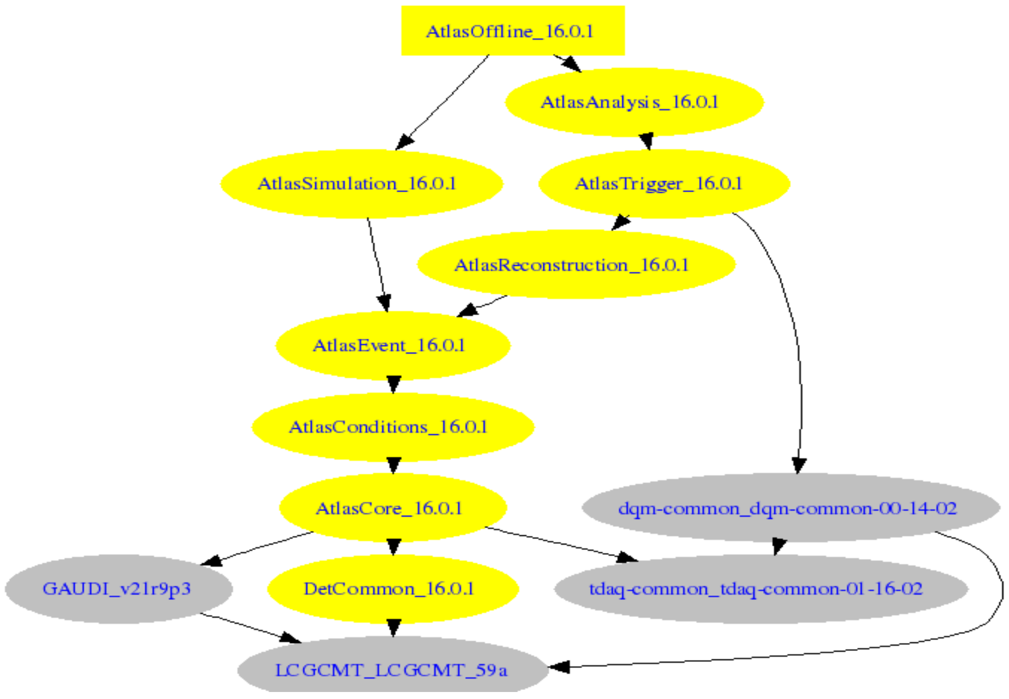


Figure 1 Project dependencies for the ATLAS offline software

2.3. Platforms
A platform is a combination of operating system (Linux or Mac OS X), compiler and compiler flags or modes. We build our software in 32bit and 64bit mode, optimized (faster execution) or debug mode (suitable for debugging purposes), using different compilers including gcc3.4, gcc4.3, icc11 and our future plan is to start platform specific builds with the llvm compiler. So for a given platform build we choose the specific compiler, operating system and the build mode and compile all projects.

2.4. Branches
A branch is a self consistent snapshot of the software, typically targeting a particular production deployment (eg. reprocessing of data). Each branch is typically built for several platforms. This approach supports production and development at the same time. Currently 32bit platform builds are used for production activities, while 64bit platform builds are under development.

3. Code and configuration management

3.1. Subversion (SVN)
ATLAS uses SVN [5] as the repository that holds code submitted by the developers. All ATLAS offline projects share the same repository but the software build infrastructure can also incorporate software from other SVN repositories. We use SVN tags provided by the developers to track stable package versions. We have developed specific ATLAS hooks for SVN which provide a mechanism to control access (read and write) to specific sections inside the repository for specific users.

We support and control access to 3 SVN repositories

- Atlasoff: offline repository holding all software packages for the projects
- Atlasusr: repository structured as plain list of user directories, dedicated to particular users
- Atlasgrp: group repository, which is structured by working groups and also by institutes.

3.2. Configuration Management Tool (CMT)
The Configuration Management Tool [2] is the primary build and configuration tool that uses package-oriented principles. It manages dependencies between packages and projects, which project a package belongs to, name and version, etc. Every package must specify its configuration in a text file called “requirements”, which is easy to read and modify. CMT identifies the elements of the build configuration: projects, packages, applications, libraries and actions. It establishes common patterns used by all packages and projects: where to install software, how to build shared libraries and applications on different platforms by using appropriate sets of compiler flags and linker options, etc. However the definition of the specific patterns or actions is done only in desired packages. CMT manages the build process to ensure the correct build sequence, so that packages that depend on other packages are compiled in the correct order.

4. Numbered and nightly releases
In order to satisfy the software development requirements of the large developer community as well as providing well validated versions of the software for production deployment a complex strategy of software releases has been developed. The characteristics of the different nightly releases are described below.

4.1. Nightly releases
These are daily builds of the software retained for several days (usually a week) before being overwritten. They are coupled with several validation frameworks, described later, and a package approval mechanism involving domain specific experts to minimize instabilities during the software build and test process. These nightly releases are the primary candidates for becoming full stable releases from which an installation kit is made which can be distributed for production

4.2. Migration releases
These are nightly releases for developer communities which allow migrations that would otherwise be disruptive. They are standard nightly releases with a few specific package overrides. They allow several developer communities to work on changes having impact on a large part of the software without disrupting the main nightly releases and hence keep them more stable for the rest of the developers. Such migrations are later merged into the primary nightly releases.

4.3. Numbered releases
These are stable snapshots of software used for production and analysis. They are deployed after high statistics validation described in section 8. We make bugfix or development numbered releases depending whether we have to fix problems found during production or we have to test new functionality and/or new external software changes.

4.4. Patch releases
These releases are rather small compared to the previous ones and provide overrides to fix small problems discovered after deployment of numbered releases. The physics analysis patch releases which are dedicated to physics groups to capture specific analysis software on top of a patch release, also fall in this category.

5. Tag Collector and software package approvals
The Tag Collector [8] is a web based tool to assign packages to projects, package versions to releases and describe the project dependencies. This tool is part of the software validation and build processes. Newly submitted packages go through a validation and approval procedure before being accepted into a release branch. The approval is performed by a set of domain-specific release coordinators. The tool sends automatically generated emails for any action performed by the developer or the release coordinator. This improves the reaction speed between the developers and coordinators. Also the Tag Collector has the concept of bundles whereby a group of package tags can be requested, validated and then accepted or rejected in a group. For this purpose all the packages are combined under the same bundle name and the manipulation of a bundle is as easy as manipulation of a single tag request.

6. Parallel and distributed builds
These are techniques required in order to build the huge code base of ATLAS offline software in a timely manner. Several techniques are used together:

6.1. Platform level parallelism
Builds for each platform are performed in parallel and results merged together in the cases where we have several platforms for a given branch. There is a dedicated machine per platform build, which requires that we have a farm of dedicated build machines. The merging itself is performed by ATLAS Nightly Control System (NICOS) - described in section 7.

6.2. Project level parallelism
We build several projects at the same time in the case where there are no mutual dependencies between the projects. This level of parallelism is also implemented inside NICOS.

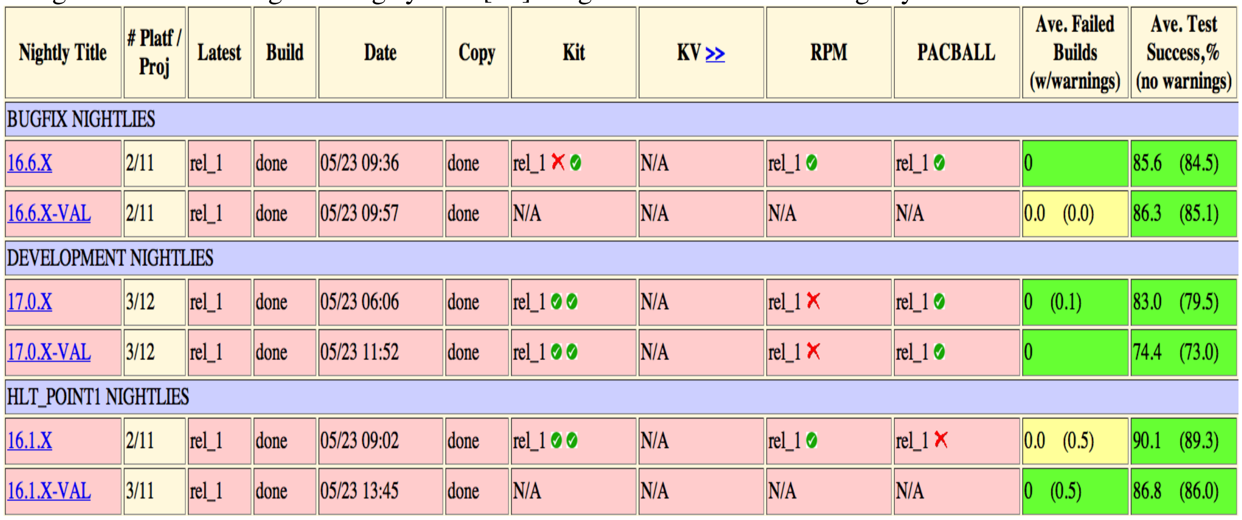
6.3. Package level parallelism
Packages with no cross dependencies are built in parallel. This way we take advantage of multi-core chips. This parallelism is implemented through a python coded script “tbroadcast” [2] which is defined on top of CMT.

6.4. File level parallelism

Here we use parallel make (make –j<n>) together with a distributed external compilation application “distcc” [6].

7. Nightly Control System (NICOS)

The Nightly Control System [7] uses CMT to build and test the different branches and platforms of the ATLAS offline software every night. NICOS connects to the Tag Collector and retrieves the package tags for a given release, requests the specific packages from SVN, checks out the source code and starts the build for the specific platform on a dedicated machine. Full release build is done in about 10 hours. Once the build is complete NICOS performs compilation error/warnings analysis and in case of problems it sends emails to the relevant developers and release coordinators. At the same time it starts defined validation tests and monitors them for completion. Upon completion it analyses the test results and in case of problems also sends an email to the developers concerned. NICOS also publishes the results on a web page as soon as they are available. Currently there are 55 nightly builds in a 24 hours period. NICOS is constantly being improved to handle the large number of builds and tests and to provide the results as rapidly as possible. Nightly built stability is improved by usage of local disk and automatic recovery from failures. By using local disk for the build and test we avoid AFS problems and reduce the AFS load. One platform is used as master for the copy to AFS and the other platforms only merge the binaries. NICOS incorporates error detection and retry mechanisms to ensure that the builds are correctly performed. It detects errors on checkout from SVN and failures to connect to the Tag Collector and will wait and retry. It can also detect AFS or network failures and retry compilations. ATLAS offline has a set of shifters that check the results every day and report problems using the Savannah bug tracking system [10]. Figure 2 shows a set of nightly builds and test results.



Nightly Title	# Platf / Proj	Latest	Build	Date	Copy	Kit	KV >>	RPM	PACBALL	Ave. Failed Builds (w/warnings)	Ave. Test Success,% (no warnings)
BUGFIX NIGHTLIES											
16.6.X	2/11	rel_1	done	05/23 09:36	done	rel_1 ✗ ✓	N/A	rel_1 ✓	rel_1 ✓	0	85.6 (84.5)
16.6.X-VAL	2/11	rel_1	done	05/23 09:57	done	N/A	N/A	N/A	N/A	0.0 (0.0)	86.3 (85.1)
DEVELOPMENT NIGHTLIES											
17.0.X	3/12	rel_1	done	05/23 06:06	done	rel_1 ✓ ✓	N/A	rel_1 ✗	rel_1 ✓	0 (0.1)	83.0 (79.5)
17.0.X-VAL	3/12	rel_1	done	05/23 11:52	done	rel_1 ✓ ✓	N/A	rel_1 ✗	rel_1 ✓	0	74.4 (73.0)
HLT_POINT1 NIGHTLIES											
16.1.X	2/11	rel_1	done	05/23 09:02	done	rel_1 ✓ ✓	N/A	rel_1 ✓	rel_1 ✗	0.0 (0.5)	90.1 (89.3)
16.1.X-VAL	3/11	rel_1	done	05/23 13:45	done	N/A	N/A	N/A	N/A	0 (0.5)	86.8 (86.0)

Figure 2 Display of NICOS showing set of nightly builds and test results.

8. Test and validation frameworks

ATLAS uses several tools to validate the build and installation of the software produced.

8.1. ATLAS Testing Nightly (ATN). ATN system [8] is integrated inside NICOS and runs around 300 tests on a given platform. This includes unit tests and functionality tests where the functionality of simulation, reconstruction, trigger or analysis is tested at the scale of a few events.

8.2. Run Time Tester (RTT). RTT [9] executes developer defined functionality tests at the scale of a few hundred events. It is the most broad spectrum validation tool for the ATLAS offline software. The developers define their tests using XML files in the packages to be tested. Each RTT test is a three step procedure. It starts with creation of a run directory and preparation of the input data files, then the tests are run meeting the specific requirements described in the XML files and then the last step is to perform any RTT or developer defined actions or additional tests for post processing. RTT provides manual/developer driven test running or fully automated runs every night. The RTT system is written in Python and uses a dedicated cluster of approximately 112 machines. The system runs more than 3200 jobs every day and it requires a considerable amount of AFS space ~6TB. RTT is highly scalable – if more tests are needed/scheduled then the number of test machines and dedicated space can easily be increased. The results of all tests are also published on a dedicated web service.

8.3. Full Chain Test (FCT). FCT checks the entire ATLAS offline software chain – event generation, simulation, digitization, reconstruction and analysis. It is a production test at the scale of thousands of events and is run on a dedicated instance of RTT using simulated data.

8.4. Tier0 Chain Test (TCT). TCT is another production test at the scale of tens of thousands of events. It is also run on dedicated instance of RTT using real data. This test, runs on nightly releases.

8.5. Big Chain Test (BCT). This is a production test at the scale of millions of events. It runs on the GRID or central CERN processing facility (Tier0) using real data and it runs on each release.

8.6. SampleA . Production test at the scale of hundreds of thousands of events, running on the GRID using simulated data. Similarly to the BCT, SampleA runs on numbered releases and together they form the final validation stage prior to production deployment.

9. Conclusion

A robust release building and validation infrastructure is essential for large and complex high energy physics experiments with widely distributed developer community. ATLAS provides tools and infrastructure which are easily accessible to our developers and easy to use. We try to keep pace with the increasing number of developers and releases, which requires increasing the level of validation and performance of the tools involved. Performing the builds and tests every night accelerates software development and increases its quality. The combination of nightly and numbered releases helps the support of development and production activities. Our ability to rapidly fix bugs and provide patches for problems has proven to be quite efficient for the validation and production efforts. However that requires dedicated hardware and manpower resources to keep the infrastructure up and running.

References

[1] http://www.atlas.ch/fact-sheets-1-view.html

[2] http://www.cmtsite.org/
Arnault C 2001 Experiencing CMT in software production of large and complex projects Proceeding of CHEP 2001 Beijing China

[3] http://twiki.cern.ch/twiki/bin/view/SPI/LcgProjectSW#LCG_AA_projects

[4] http://atlastagcollector.in2p3.fr
Albrand S, Collot J, Fulachier J and Lambert F 2004 The tag collector – a tool for ATLAS Code Release Management. Proceedings of CHEP 2004 Interlaken Switzerland

[5] http://subversion.tigris.org

[6] http://code.google.com/p/distcc

[7] http://savannah.cern.ch

[8] Undrus A 2004 Proc.Int.Conf. on Computing in High Energy and Nuclear Physics CHEP'04 (Interlaken, Switzerland) CERN 2005-002 pp 521-3

[9] Ciba K, Richards A, Sherwood P and Simmons B, 2009 The ATLAS RunTimeTester software, Proceedings of CHEP2009, Prague Czech Republic