Abstract. The Software Process & Infrastructure (SPI) project provides a build infrastructure for regular integration testing and release of the LCG Applications Area software stack. In the past, regular builds have been provided using a system which has been constantly growing to include more features like server-client communication, long-term build history and a summary web interface using present-day web technologies. However, the ad-hoc style of software development resulted in a setup that is hard to monitor, inflexible and difficult to expand. The new version of the infrastructure is based on the Django Python framework, which allows for a structured and modular design, facilitating later additions. Transparency in the workflows and ease of monitoring has been one of the priorities in the design. Formerly missing functionality like on-demand builds or release triggering will support the transition to a more agile development process.

1. Motivation
The LCG Applications Area software projects share a single development infrastructure; this infrastructure is provided by the Software Process & Infrastructure (SPI) project. A set of basic services and support are provided for the various activities of software development: release building, providing external software and testing frameworks.

SPI provides a nightly build system based on Python for the regular testing of LCG Applications Area projects. It tests the projects on a set of different configurations and platforms to cover experiments’ use cases. Different combinations of the supported operating systems (Mac OS X, Scientific Linux –SLC–), compilers (gcc, llvm, icc) and architectures (32 bit, 64 bit) are built every day and released systematically. The LCG software stack is built, tested and installed every night in more than 30 different combinations.

SPI also provides releases on a regular basis. Depending on the needs of the experiments, the periodicity can be between weeks to months. There are 12 supported platforms for which the releases are built, which were agreed with the experiments. For each release, binaries, source code and documentation are made available on AFS. External packages are also added or updated by request of the experiments. There are more than 70 external software packages which need to be updated for each of the supported platforms.

This nightly build system has been totally independent from the release workflow and the external packages builds. However, these processes are deeply connected, so an integrated infrastructure to centralize all these tasks would accelerate and simplify the whole workflow.

Moreover, the nightly build system itself, which has been suffering from multiple additions and transformations during the last years, was lacking some functionality which would ease the task, i.e. build nodes monitoring, more transparency and communication from the build node within the build process, modular and clean design to facilitate maintenance, on demand builds, etc… Given all these
drawbacks, a new system was proposed based on the group expertise with the previous system. This new system has been designed aiming for an integrated solution providing all the previously lacking features and putting special emphasis on modularity, stability and maintainability.

2. The new system

2.1. Ideas for the new system

All the drawbacks and a set of needed features were the main motivations for developing the new version. The main improvements are:

- On demand builds
- Incremental builds
- Remote clean-up
- AFS management
- Remote doxygen documentation generation
- Remote opengrok updates
- Release database integration
- External packages builds
- Automatic release workflow
- Status of the infrastructure

2.1.1. On demand builds

In the previous system the nightly builds were loaded from an xml file which was reloaded using a cronjob. To add again a job to the queue if it failed meant that the server had to reload again the xml file, so previously the file had to be edited to put only the aimed build. This process made the whole process very inflexible and difficult to operate.

In the new system a request web interface connects with the build server via XMLRPC to manage the queue of jobs and add/edit/remove jobs easily. With this feature, the maintenance of the infrastructure decreases dramatically and improves its flexibility. Now, if we see that a machine is too loaded or that one build froze, we can stop the whole process remotely and then perform a remote cleanup to leave the machine in the initial state and start from scratch.

2.1.2. Incremental builds

Incremental builds were not supported on the previous system, which only allowed full builds from scratch. The new system also allows building incrementally a given build instead of starting every time from a clean checkout. This allows faster feedback to know if any committed piece of code solves a given problem spotted on a previous build. The system keeps track of the location where the failing build is and runs the incremental build on the same area. This is a key feature to improve the feedback frequency and to facilitate agile software development. Also, this could be used from any external software (e.g. svn triggering) using the server API.

2.1.3. Remote clean-up

In order to improve the machine management, the new system supports automatic build cleanup. Until now, a cronjob was responsible of cleaning all the build areas, and when something was failing manual intervention was needed. Now the server is aware of the build nodes status, and when the free space is under a given threshold, the server will trigger the clean-up of older builds. Also, the request interface allows manual triggering of the cleanup procedure without having to log in the machine.

2.1.4. AFS management

In order to automatize as much as possible the whole process, AFS management must be available also in the request interface, as it is itself an important part of the manual steps for release and external
packages building. Now from the request interface we will be able to create volumes, replicas, increase volume sizes, etc.

2.1.5. Remote doxygen documentation generation
This step is triggered now from the request interface, either manually or by starting a release. This is a big improvement from generating it manually from a prompt, reducing the maintenance and time for releasing. This step is part of the release workflow.

2.1.6. Remote opengrok updates
Same as happened with the doxygen documentation, our instance of the opengrok source code browser had to be updated manually, increasing the time for a release preparation. Now is available to be triggered from the request interface, and will be automatically triggered when a release is started. Our opengrok instance is running on the central IT CERN web services. Same as the doxygen documentation generation, this step is a part of the release workflow.

2.1.7. Release database integration
We provide a webpage as a frontend to our releases database, legsoft.cern.ch. This database was updated manually every time a release was built. In the new system, this database has been merged into the new system, and has been improved including more information about dependencies and keeping results of the builds and information about the nodes the release was built in. Of course, no manual intervention is needed to update the release web frontend as the release automatic build will set the information as public once the release process has finished successfully.

2.1.8. External packages builds
Until now, when a new external package was to be built, all the process was done manually for each of the different platforms independently. Of course, this task is a much known workflow which could be automatized easily adapting the nightly scripts to provide the specific steps which external package building needs. For this, we need the new AFS management support as we create new AFS areas for each new version and we replicate them after the binaries installation has been done.

2.1.9. Automatic release workflow
Similar as it was happening with the external packages, the release workflow had a big part of manual intervention. The nightly builds were already being used for the build itself, and once the build of the wanted platform with the given tags was successful, all what followed was manually done. The AFS management capabilities are again a key functionality to automatize the workflow. Doxygen documentation generation and opengrok update, as explained before, are also part of the release process, and will be automatically triggered when the release is started.

2.1.10. Status of the infrastructure
One of the most helpful and important new features is the real-time monitoring of the different nodes, where the administrators are able to see the load and availability of the nodes in build cluster. Until now if a machine was down there was not feedback available for the mangers until they checked manually by themselves. Knowing the status of the infrastructure is very important to act fast accordingly when something goes wrong in the build process.

2.2. Technologies used
The scripts are written in python. To allow a modular and structured design, we decided to move towards a framework. We took this chance to reformat all existing code and to make it easier to maintain and to expand in the future. We chose django [2] as it is a very well-known framework with a very clean design, good readability, large community and extensive documentation. So the central
build server will run a django instance and it will represent the model part of a model-view-controller design.

Over this instance a listener will act as the controller of the system. This XML-RPC server will wait for requests from the request interface webpage or the build machines, and modify the model accordingly.

The releases webpage and the nightly build results webpage will also run their own instance of django. They are read-only clients of the DB and their functionality is not linked with the build nodes, so we can run them separately.

For the new web interfaces, up-to-date technologies will be used, like HTML 5 and AJAX \cite{1} communication. Those were already being used in the results webpage, which will not suffer any major change apart of adapting it to django in the backend.

The build nodes use CMT \cite{3} to build the different projects. CMT is a configuration management tool which gives us the possibility to control the build and test environment with the required versions of the external dependencies. Also, the plugin system, used to define the builders for the different projects, makes possible to define custom builders to use different tools, e.g. CMake, GNU Makefiles, etc.

2.3. The central build server

The build server is the central part of the infrastructure. It acts as the manager of the builds and it will take care of storing the results when the nodes finish. Also, it will be affected by the requests done via the web interface. It consists of three different processes which run in parallel:

- Pinger process: This process will monitor the build nodes to know which ones are alive and responsive. That way, the system knows which nodes are available to build which kind of architectures. The server will keep in the DB the history of machines which have ever connected and when started will do an initial ping to each of them to know if any are already available to build.

- Listener process: It will keep open an XML-RPC server waiting for any request or information. It will receive requests from the request interface and update the list of jobs and their status. It will be responsible also for receiving the build results from the build nodes and the status of the running builds.

- Dispatcher process: It’s the process responsible to manage the queue of jobs and dispatch them to the build nodes accordingly to their load and specifications to match the desired build architectures. It reloads the queue every certain time to look for changes on the queue made from the request interface (which are indeed done by the listener process).

The server also has AFS management capabilities, so creating AFS volumes and replicas is directly accessible from the server API. It provides as well APIs to update opengrok source browser with any given version of a package and to generate its doxygen documentation.

2.4. The build nodes

The build nodes also run an XML-RPC server to listen requests from the central build server. They are much more communicative with the server and they will send to it the current step for each of the builds they are currently doing. The communication protocol has also been improved so the server is aware of any change on the cluster. There is also a failover protocol to restart the communication if the network is down in any of the nodes.

The python wrappers to execute the command line instructions for each of the packages, called builders, will be now python plugins, so the nodes can import different builders to build and test any package if the builder is provided. This improves the portability and adaptability of the system depending on the needs of the software to test.

2.5. The release database
The release database (available at lcgsoft.cern.ch) is a webpage frontend to consult the releases which are available. It hasn’t suffered major changes in functionality. We have adapted the backend to connect to the django database instead of its own. The scripts previously used to fill the database are now not needed, as now the request interface updates automatically the django database during the release process. This is a good example of how merging the different tools it has allowed us to simplify the whole workflow and get rid of many redundant pieces of software.

2.6. The nightly builds results

This webpage (available at http://lcgapp.cern.ch/spi/cgi-bin/nightly.py), which shows the results of the building and the testing of the projects, hasn’t suffered either any major change. Only a change to support the incremental builds, which didn’t exist in the previous system, to be able to relate a build with its predecessor. It is based on AJAX and displays the results according to the requirements of each build; i.e. a profiling or a coverage build displays a different kind of information with specific data which is not shown in normal builds. Of course, it also shares the database with the rest of the system.

The interface allows advanced filtering and ordering of the platforms, storing the preferences on cookies to allow keeping a customized view for each of the users. Also, when multiple dates are selected the view can be organized grouped by projects or by dates, making it easier to compare results between different builds.

2.7. The request interface

This new feature gives us control on whole cluster. It presents the status of the cluster to give feedback to the administrators and control over the running builds. It allows managing directly the builds and actions like stop, restart and cleanup the build area are accessible from the web interface remotely instead of having to log in manually in the machine and perform all these actions in place or having to configure cronjobs to do it.
Thanks to the API that the new server provides, we can actually build releases and external packages directly from the request interface (or extend the API from another client, increasing the number of possibilities), and the server can create the AFS volumes and replicas for them automatically and set the correct ACL.

With the request interface we have also access to the queue of jobs and manage it accordingly to our needs. Adding a new platform on-the-fly is also possible without having to restart the server as it was done with the previous system. All these features on the jobs queue provide us a much better and dynamic management of the whole infrastructure.

Also, the release and external package installation workflows are defined in the model, so creating a new release or installing a new external package is as easy as triggering the different steps in the web.

3. Summary
The new system is more transparent, autonomous and proactive. We have drastically simplified the administration procedures of the build system, making it easier to use. All the new features provided by the request interface were previously done manually on a console, and the feedback from the build nodes was inexistent. The new system will improve the productivity of the group resources and will give more, faster and better feedback to developers.

Furthermore, merging the release database, the nightly build results and the jobs queue into a single system has allowed to structure all together into a modular, expandable and organized tool which can be easily extended following the API that it provides, when before all this set of features were spread in different script without a common meet point.

Finally, we’ve chosen django because of its large and active community and its nice programming guidelines. The readability of the whole system has improved very much, making it easy to maintain and implement future feature additions. Thanks to the use of a framework and the restructuration which has been made, the code base has been reduced in more than a 60 per cent compared with the previous system, so its maintainability has been very much improved.

References
[1] http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications
[2] http://www.djangoproject.com/
[3] http://www.cmtssite.org/