ATLAS organises its software into “projects”, built from “packages”. The smallest (analysis) project is composed of ~140 packages, while the full offline project is ~2200 packages strong. Some packages (like Event Data Model definitions) are used in multiple projects at the same time.

The build system needs to be able to:
- Discover the dependencies between packages, building them in the correct order
- Allow a package to be moved from one project to another one
- Allow a patch release to be built on top of an existing release, containing updated versions of just a few packages
- Be robust against build errors. Even with code errors present, the build and RPM creation needs to progress as far as it possibly can.
- Allow the users to test changes to packages on top of a software release/nightly

ATLAS switched to using CMake (https://cmake.org) as its offline software build tool in 2016.

- Replacing CMT (http://www.cmtsite.net)
- Converting the CMT configuration of >2200 packages into CMake configurations, using a custom Python script translating the configuration

Package Configuration

Packages in ATLAS can have multiple roles:
- Hold the source for libraries and executables, building those libraries and executables
- Most packages depend on source code from other packages for their build
- Hold scripts and other files necessary for the runtime environment, installing these into the right place during the build

Each one comes with a single CMakeLists.txt file, describing how that one package should behave.

Packages need to depend on each other, while not knowing where a dependent package is coming from.
- The upstream package could either be in a base project, or in the current one

Performance

Switching to CMake reduced the build time over CMT by ~40%. RPM packages are created as part of the nightly build. Delivering a software release is just copying the RPMs from a nightly server to the release YUM (http://yum.baseurl.org) server.

- Installing a full offline release on CVMFS from RPM using a modified version of YUM (https://gitlab.cern.ch/rauser/avum) takes ~1 hour.

There are things still to do:
- Reduce the number of projects. To avoid the installation downtowm between project builds.
- Parallelise installation and RPM building
- Only build unit test code/executables after the general build step.
- Some further improvements for using Continuous Integration in the builds