EPJ Web of Conferences **214**, 07005 (2019) https://doi.org/10.1051/epjconf/201921407005
*CHEP 2018*

# Building and using containers at HPC centres for the ATLAS experiment

*Douglas* Benjamin[1], *Taylor* Childers[2], *David* Lesny[3], *Danila* Oleynik[4], *Sergey* Panitkin[5], *Vakho* Tsulaia[6], *Wei* Yang[7,*], *Xin* Zhao[5], on behalf of the ATLAS Collaboration

[1]Duke University, Durham, NC, USA
[2]Argonne National Lab, Lemont, IL, USA
[3]University of Illinois, Urbana-Champaign, Champaign, IL, USA
[4]University of Texas, Arlington, Arlington, TX, USA
[5]Brookhaven National Lab, Upton, NY, USA
[6]Lawrence Berkeley National Lab, Berkeley, CA, USA
[7]SLAC National Accelerator Lab, Menlo Park, CA, USA

**Abstract.** The HPC environment presents several challenges to the ATLAS experiment in running their automated computational workflows smoothly and efficiently, in particular regarding issues such as software distribution and I/O load. A vital component of the LHC Computing Grid, CVMFS, is not always available in HPC environments. ATLAS computing has experimented with all-inclusive containers, and later developed an environment to produce such containers for both Shifter and Singularity. The all-inclusive containers include most of the recent ATLAS software releases, database releases, and other tools extracted from CVMFS. This helped ATLAS to distribute software automatically to HPC centres with an environment identical to those in CVMFS. It also significantly reduced the metadata I/O load to HPC shared file systems. The production operation at NERSC has proved that by using this type of containers, we can transparently fit into the previously developed ATLAS operation methods, and at the same time scale up to run many more jobs.

## 1 Motivation

The Grid computing model developed by the World Wide LHC Computing Grid (WLCG) provided most of the computing resource for the LHC Run 1 and Run 2. This model consists of many Grid sites, from small to large, at the participating universities and laboratories [1]. Each provides computing clusters and storage resources. While quite successful, the expected growth in CPU requirements for LHC Runs 3 and 4 will outpace the resources that will be available at Grid sites [2]. It is clear that the ATLAS experiment [3] needs to explore non-Grid opportunistic resources in large quantity in order to satisfy the need of LHC Run 3 and Run 4.

Supercomputers such as Cori [4] and Edison [5] at NERSC in the United States, Theta [6] at ALCF, Titan [7] at OLCF, Piz Daint [8] at CSCS in Switzerland and MaroNostrum [9] at BSC in Spain are usually much larger systems. They present significant opportunistic

* Corresponding author: yangw@slac.stanford.edu

resources that are available to the LHC computing. However, they serve a large number of users in many science disciplines, and it is not possible to ask the HPC centres to tailor their environment for the need of ATLAS computing. Early ATLAS effort at HPC centres showed several challenges in running ATLAS production workflows there. This paper will discuss two of those challenges: making ATLAS software available on HPCs and reducing metadata I/O on HPC shared file systems.

### 1.1 Making ATLAS software available on HPCs

The LHC Computing Grid has been very successful in using CVMFS [10] to distribute LHC experiments' software environment to their Grid sites. However, the CVMFS client on each computing node requires a FUSE kernel model to be loaded. For various reasons, many HPC centres are reluctant to deploy CVMFS. An alternative is to install the ATLAS software environment in a HPC's shared POSIX file system. This installation is a non-trivial task that has to be performed every time a new ATLAS software release comes out. It also uses a large number of precious i-nodes of the shared file systems. The installed environment is often different from those in CVMFS due to various constraints such as hardwired path starting with /cvmfs in software and configuration files, making it hard to debug problems.

Since most of these HPC centres now support container technologies one way or the other, it is possible to put the ATLAS software environment in a container. This will distribute ATLAS software environment consistently, without high labor cost.

### 1.2 Reducing metadata I/O on HPC shared file systems

The ATLAS software environment has many small files. They are python scripts, shared libraries, configuration files, detector database release files, etc. Initially we put them directly in HPC shared file systems. As a result, ATLAS jobs generated an excessive amount of metadata I/O due to file lookup, locking, open and close. We ran into scaling issues when running a large number of ATLAS jobs.

Studies at Lawrence Berkeley National Laboratory demonstrated that by putting the software environment in a container, one can significantly reduce metadata I/O exposed to the HPC shared file system [11], as shown in Figure 1.

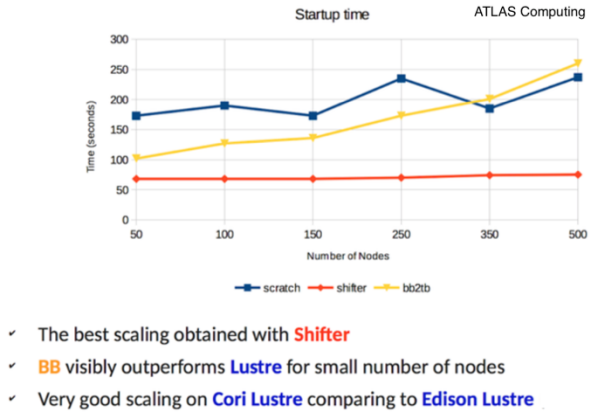


**Fig.** 1 Startup time of ATLAS software when installed in different storage on NERSC Cori. From within Shifter containers [12], from all solid-state Burst Buffers (BB) or Lustre file system.

Doing so is similar to putting software in a file-based filesystem and loop mounting the filesystem on computing nodes. This way, metadata I/O happens on the computing nodes. The HPC shared file systems do not see metadata I/O, and thus only need to serve data blocks, which is what those file systems were tuned to do.

## 2 Building all-inclusive containers for ATLAS

Through automatic deduplication and caching at various levels, CVMFS is very good at hosting large amount of current and legacy software (in ATLAS case, several Terabytes, and tens of millions of files) and delivering only needed software pieces on demand to the client. Over time, a large number of software packages were installed there. Because of those nice features in CVMFS, physicists and software developers so far have not had a need to worry about cleaning up in CVMFS. This presents a challenge when building containers to host ATLAS software.

In theory, we can use any method to put ATLAS software environments in container images. In practice, a complete fresh installation of all software packages is not possible because many of them were installed and configured by their respective experts over a long period, without clear documentation. Also given the tens of millions of files in the ATLAS CVMFS repository, we suspect that each fresh installation will take weeks.

Another approach is to copy the content of the ATLAS CVMFS into the containers. Since most files in the ATLAS software are text files, including identical files in different software versions, CVMFS deduplication was effective in reducing the size and total number of files of the repository. When we build our container, we also need to preserve the feature to ensure reasonable size of our final container images. In addition, we can also filter out unneeded software (for example, software for online trigger) based on our specific need, which is offline computing.

Certain things are difficult to determine when filtering. To fit our needs, we tried to include as many software packages as possible. The resulting container includes most of the software environment and configuration for automated ATLAS workflows, grid tools, recent ATLAS software releases for offline data productions, and the most recent database releases.

The resulting container image can be large, O(100) GB. Note that container image size is not the highest priority to us. This is because, similar to real file systems, its size has nothing to do with its performance. The size of a container is only an issue when transporting and storing them.

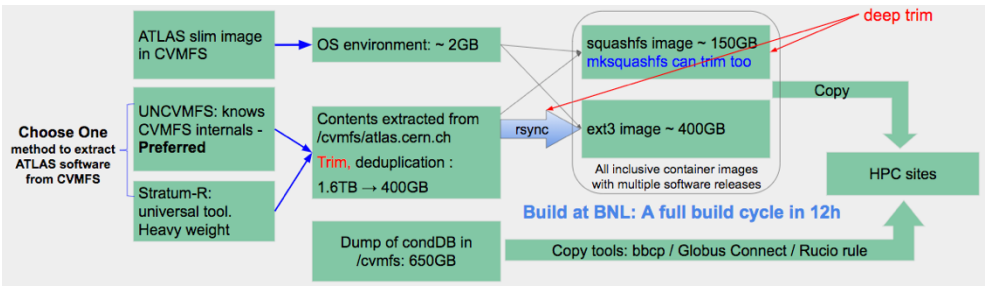


**Fig.** 2. Procedure of building all-inclusive ATLAS container images for HPCs. Note the ATLAS conditions database (condDB) is not part of the all-inclusive container. But we also extract and copy them to the HPC sites (Explanation of UNCVMFS, Stratum-R, etc. will be given below).

In summary, to build containers for ATLAS offline computing, we need to extract CVMFS contents, deduplicate, filter unneeded, and pack into a container image. Figure 2

shows the whole building process workflow (for both read-only, compressed squashfs image and writeable, uncompressed ext3 image). To make our building process usable by others, the building process needs to involve as little Unix root privilege as possible. In our current process, setting up needs a onetime root privilege to create an empty EXT3 filesystem in a file. The subsequent building process does not require such privilege.

### 2.1 Extracting CVMFS contents and deduplication

We have successfully experimented with two methods to extract CVMFS contents (a.k.a "data" in CVMFS): using rsync to extract data out of the CVMFS repository and then using Unix tools such as "rdfind" to deduplicate (also known as Stratum-R), or using UNCVMFS to extract and deduplicate the CVMFS contents at the same time.

Stratum-R is completely independent of the internals of CVMFS and will always work no matter how future CVMFS evolves. It is less efficient because during the extraction process, it does not know that some seemingly independent files scattered in the filesystem are actually the same file. It uses extra time during the deduplication process to find that out. This inefficiency can be mitigated by doing incremental updates.

UNCVMFS [13] is a software tool that understands the internal structure of CVMFS. It extracts data efficiently and deduplicates at the same time. It also can make incremental updates. However, UNCVMFS is a third-party tool. It is currently not supported by the CVMFS development team. Good progress has been made to have the CVMFS developers provide and support a similar tool.

Special attention is needed on the per file hard link limit imposed by file system during the building process - we found that a few files can have an many as 900 k hard links each. This can create problems as we later rsync them into an EXT3 container image, since EXT3 only allows up to 65 k hard links for each file. We chose to use an EXT3 file system to host the extracted CVMFS data so that no file will have an excessive number of hard links to cause problems later in the building process.

The container we produced and used in production used UNCVMFS in the process, though we also tested Stratum-R. They are interchangeable in the building process.

### 2.2 Filtering

The volume and number of files in a CVMFS repository have a direct impact on the ability to build all-inclusive containers. Filtering unneeded files is performed at various stages and filtering at early stage of the building process has a bigger impact to the overall performance. But we only want to filter those files that we have high confidence that they won't be used. UNCVMFS has some limited ability to filtering. When using squashfs tools to create compressed container images or using rsync to create uncompressed (writable) images, we have a chance to use their fine-grained filtering rules to further remove unwanted files.

Roughly speaking, the ATLAS CVMFS repository has three major areas – software environment (including grid tools, compilers, ROOT packages, python and libraries, network or non-network services packages), ATLAS software releases, and other things such as database releases. For the purpose of running simulation jobs on HPCs, we removed anything compiled under the gcc 4.3, 4.7, 4.8 compilers, anything built for the SLC5 platform, i386 platform, old ATLAS software releases (at the moment releases below 21.x), ATLAS software for online computing or for user analysis, and all except the most recent database release.

After filtering, we did find a few missing software packages that we still need to add back, such as yampl libraries compiled under SLC5 platform (which is used by some

ATLAS workflow). During operation, we also use bind mounts to provide up-to-dated X509 CA certificate directory and VOMS directories, as well as ATLAS Grid Information System configurations files.

### 2.3 Packing software into a container

The above extracted and filtered software are combined with the base CentOS 6 software in the ATLAS singularity container in CVMFS. We either use squashfs to put them together to build a compressed container image, or use rsync to copy them into an EXT3 filesystem as uncompressed container image.

Shifter and recent versions of Singularity support container image in squashfs format. The squash tools (mksquashfs) provide rsync-like fine-grained inclusion/exclusion/filtering capabilities. The building process is CPU intensive and is much quicker than building EXT3 based containers. Since the image is compressed, it is also much smaller (100 - 200 GB), However it is read only and thus cannot be customized by sites – site customization needs to happen before building the images.

Older Singularity containers only support EXT2/EXT3 format. To build EXT format containers, we first use root privilege to create a large enough EXT3 filesystem in a file, and copy the CentOS 6 base software into the filesystem. This becomes the first Singularity container image, in which we run rsync inside to sync ATLAS software into the container itself. Given that ATLAS has tens of millions of files, the first such rsync process take days to complete. But later, incremental rsyncs are much shorter - on the order of several hours. The resulting container images / EXT filesystems consist of 400 - 500 GB used space, plus some empty spaces. We keep this container for future rsync operations. In the meantime, the final container image product is a copy of this container image with most of the empty space removed.

Though creating squashfs images is much quicker, easier and saves space, we keep the ability to rsync and create uncompressed images because this sub-process (the blue "rsync" in Figure 2) can be repeated in the building pipeline several times to create different container images for different use cases. We cannot do the same with a compressed image.

We tested the first few all-inclusive container images at the SLAC National Accelerator Laboratory. There we identified problems and fixed them until we were able to run ATLAS simulation workflows and event service workflows. Except for known significant layout changes in the ATLAS CVMFS repository, future incremental builds do not always need this kind of validation.

### 2.4 Container building environment

We build the all-inclusive ATLAS container at the Brookhaven National Laboratory. The machine we use has a decent amount of CPU cores (16), memory (80 GB) and the ATLAS CVMFS repository.

GPFS space was provided to host the extracted CVMFS data, as well as intermediate and final container images. To speed up operation on large number of small files during CVMFS data extraction and deduplication, as well as squashfs and rsync, we use the same technology we proposed to use on the HPCs – create a large EXT3 file system in a GPFS file and loop mount it (to host the extracted CVMFS data). This is part of the build environment setup but not part of the regular building process. This setup is optional and require root privileges.

In our current building process, once the UNCVMFS (or Stratum-R) incremental syncing is completed, we fork the process and start building the squashfs image and the

EXT3 image at the same time because building squashfs image is CPU intensive, while building EXT images is IO intensive. A squashfs image is usually produced within 8 to 10 hours, whereas an EXT3 image is usually finished in 12 to 16 hours.

## 3 Use cases

The all-inclusive container images have been used at several HPC centres, including Cori 2 (#9 in TOP500 HPC, spring 2018) and Edison at National Energy Research Supercomputer Center (NERSC) in the US, Theta (#21) at Argonne Leadership Computing Facility (ALCF) in the US, as well as MaroNostrum (#22) at the Barcelona Supercomputer Center in Spain.

An ATLAS team working on the Titan (#7) at the Oakridge Leadership Computing Facility (OLCF) found that an ATLAS job loads many shared libraries and python scripts at startup time, and that loading these small files causes thousands of data and metadata operations against the main shared POSIX file system. Moreover, the number of metadata operations are amplified by a long LD_LIBRARY_PATH. By using containers similar to the all-inclusive container above, they were able to reduce the metadata I/O to near zero, as shown in Figure 3.

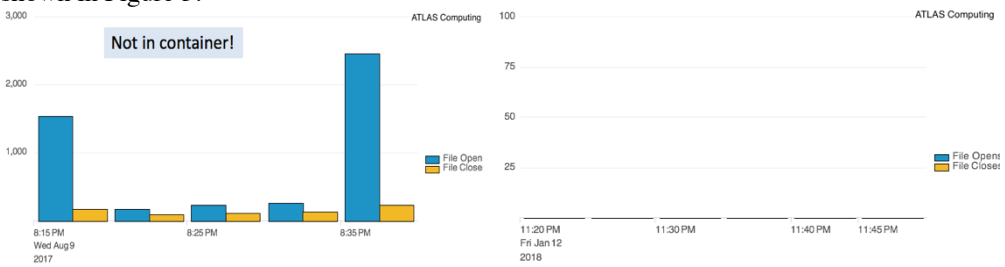


**Fig**. 3. Numbers of file opens and closes of an ATLAS job against Titan's shared file system, Left: Container is not used. Right: Container is used. Shared file system does not see metadata operations.

All-inclusive containers have been used on Theta HPC at ALCF. Jobs running there, the ATLAS Yoda Jobs [14], use MPI to coordinate operations amount tasks. In order to get higher batch scheduling priority, large MPI jobs (using hundreds of nodes) is preferred. The scalability of these large MPI jobs are sensitive to the IO contention. Figure 4 and 5 show that the all-inclusive container enables linear scaling of concurrent running those jobs.

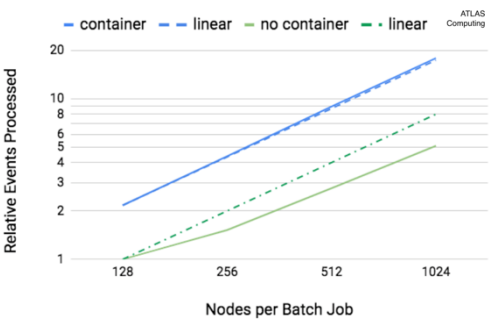


**Fig.** 4. Scaling with the number of nodes for ATLAS Yoda jobs [14], compared to a 128-node Yoda job that does not use containers. "Linear" means the expected linear scaling.

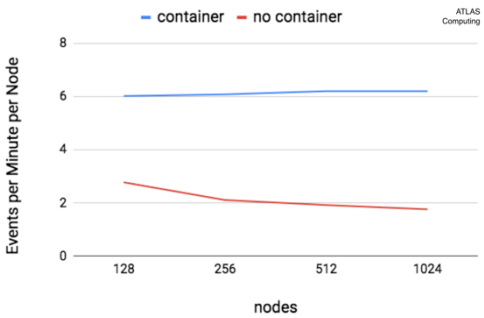


**Fig.** 5. Scaling of simulated events per minute per node along with MPI job size (number of nodes). All-inclusive containers enabled ATLAS (MPI) Yoda jobs to maintain their performance.

At NERSC, we started by installing the ATLAS software environment and releases in the main Lustre shared file system. We successfully ramped up the ATLAS production operation to 1000 nodes on Cori 2 (each node has 68 KNL cores, and can run two threads (136 logical cores) without slowing down). Beyond 1000 nodes, we quickly run into IO contention issues.

After we switched to use all-inclusive containers at NERSC, we were able to ramp up from 1000 nodes, to 2000 nodes and eventually to 3000 nodes (as shown in Figure 6) without slowing down by the IO contention (with 3000 nodes, we were using 1/3 of Cori 2 and the chance that we could use more nodes than that is slim).

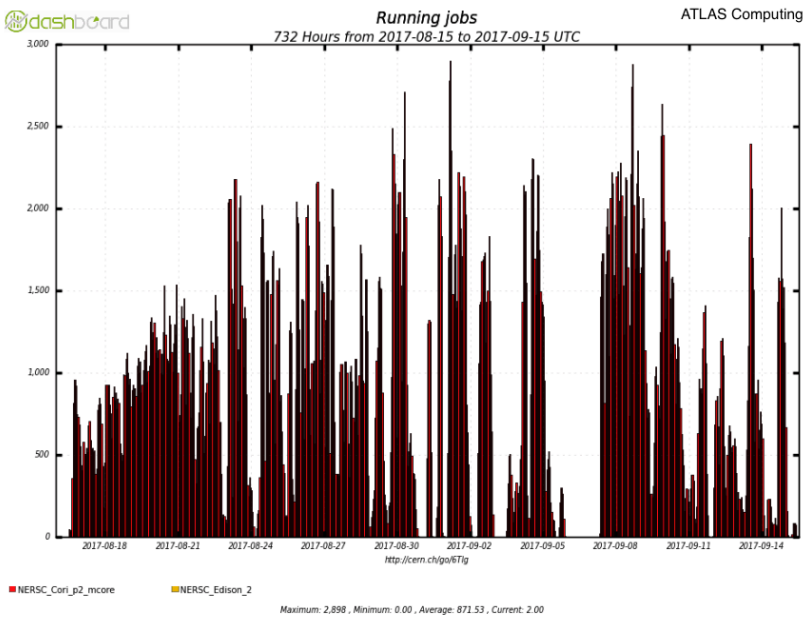


**Fig.** 6. Number of NERSC Cori 2 nodes used by ATLAS jobs running inside the all-inclusive containers during a period of 4 weeks.

## 4 Next steps

The success of CVMFS entices people to dump more software (including different versions of the same software) there without having to worry about how the software are distributed. The challenge is to build all-inclusive container images and yet be able to control their sizes, and be able to build new images in reasonable amount of time. Since we started to build this kind of container about a year ago, its size increased from a little over 100 GB to near 200 GB for compressed images, and from 450GB to more than 600 GB for uncompressed images.

So far we depend on filtering to decide what to keep in the container images. However, for ATLAS software release there is another possible way of doing things. ATLAS software releases follow a predefined installation procedure, in order to support users who want to install them on their laptop and desktop. It is possible for our container building process to initially exclude all ATLAS software releases, and then install them using those pre-defined procedures. This may take longer time as this procedure cannot install the ATLAS software release packages incrementally. But it also open up the possibility to install a single release, a few releases, releases compiled by a specific version of GCC, or compiled for a specific target HPC.

## 5 Conclusion

ATLAS built all-inclusive containers and used them to successfully address the issues of software distribution and metadata I/O reduction on HPCs where CVMFS is not widely available. We have overcome challenges to efficiently extract ATLAS software from CVMFS, to update containers in short period, and to make their sizes small enough to be transportable. As ATLAS puts more software in CVMFS, the method we use to build this kind of container needs to be improved, and we have identified a few potential modifications for our future work.

## 6 Acknowledgement

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

## Reference:

[1] http://wlcg.web.cern.ch/
[2] L. Sexton-Kennedy, *HEP Science Goals for the Next Decade*, https://indico.cern.ch/event/658060/contributions/2844782/attachments/1622746/2582912/ScienceGoalsWLCG-HSFworkshop2018.pdf
[3] The ATLAS Collaboration, *JINST* **3** S08003 (2008)
[4] Cori: https://www.nersc.gov/users/computational-systems/cori/
[5] Edison: http://www.nersc.gov/users/computational-systems/edison/
[6] Theta: https://www.alcf.anl.gov/theta
[7] Titan: https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/
[8] Piz Daint: https://www.cscs.ch/computers/piz-daint/
[9] MaroNostrum: https://www.bsc.es/marenostrum/marenostrum
[10] J. Blomer et al, *CernVM-FS: delivering scientific software to globally distributed computing resources,* Proceedings of the first international workshop on Network-aware data management
[11] L. Gerhardt, W. Bhimji, S. Canon, M. Fasel, D. Jacobsen, M. Mustafa, J. Porter, V. Tsulaia, *Journal of Physics: Conference Series* **898** 082021 (2017)
[12] Using Shifter at NERSC, https://docs.nersc.gov/programming/shifter/how-to-use/
[13] UNCVMFS github: https://github.com/ic-hep/uncvmfs
[14] P. Calafiura, K. De, W. Guan, T. Maeno, P. Nilsson, D. Oleynik, S. Panitkin, V. Tsulaia, P. V. Gemmeren, T. Wenaus on behalf of the ATLAS Collaboration, *Journal of Physics: Conference Series* **664** 092025 (2015)