Using containers with ATLAS offline software

J Elmsheuser[1], L Heinrich[2], G Stewart[3], M Vogel[4] on behalf of the ATLAS Collaboration

[1] Brookhaven National Laboratory, PO Box 5000, Upton, NY 11973, USA
[2] New York University, 70 Washington Square South, New York, NY 10012, USA
[3] University of Glasgow, University Avenue, Glasgow Metropolitan Area G12 8QQ, UK
[4] University of Wuppertal, Gausstrasse 20, 42119 Wuppertal, Germany

E-mail: marcelo.vogel@cern.ch

Abstract. This paper describes the deployment of ATLAS offline software in containers for software development. For this we are using Docker, which is a lightweight virtualization technology that encapsulates a piece of software inside a complete file system snapshot. The deployment of offline releases via containers removes the strict requirement of compatibility between the runtime environment needed for job execution and the configuration of worker nodes at computing sites. If these two are decoupled from each other, sites can upgrade their nodes whenever and however they see fit. In this work, ATLAS software is distributed in containers either via the CernVM File System (CVMFS) or by means of a full ATLAS offline release installation. In software development, separating the build and runtime environment from the development environment allows users to take advantage of many modern code development tools that may not be available in production runtime setups like SLC6. It also frees developers from depending on resources like lxplus at CERN, allowing the use of an average laptop for ATLAS code development. We include in this document a basic comparison in performance of the two deployment options in two popular host operating systems: Ubuntu and OS X.

ATL-SOFT-PROC-2017-057
16/02/2018

1. Introduction

Docker is a light weight, software based virtualization technology [1], which encapsulates a piece of software inside a complete file system that contains everything that applications need to run: code, runtime, system tools, system libraries, etc. (see Figure 1, and 2). Processes run natively in the host yet isolated in containers, differentiating Docker from virtual machines, which simulate both hardware and software. This isolation guarantees that the software will always run in the same way regardless of the host's configuration, thus providing a uniform runtime environment ideal for distributed computing. The ATLAS Computing Group of the ATLAS Collaboration [2] is actively engaged in the deployment of container technologies for software development.

2. ATLAS software in Docker images

The ATLAS Offline Software Group currently hosts a fairly comprehensive body of documentation with detailed instructions for building, running and managing containers with ATLAS software [3]. At the core of the 'containers' section of the documentation is the so-called Dockerfile, which is a configuration file used by the Docker engine to build images in a completely automatic way. In a nutshell, Dockerfiles consist of a set of sequential instructions with each one adding a new layer to the image (see Figure 3).

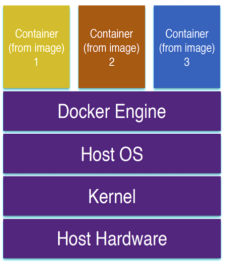

Figure 1. Docker will share the host OS kernel, but isolate different environments or 'containers' with respect to data, code, networking, etc.

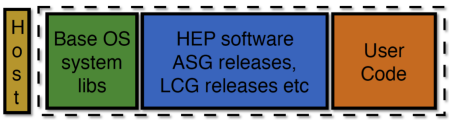

Figure 2. Docker images can be thought of as archived, executable file system snapshots.

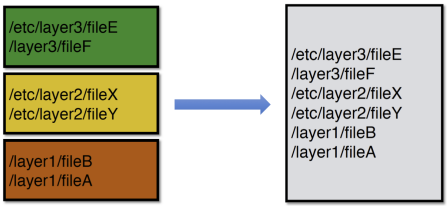

Figure 3. Docker images are built from layers.

In the containers sections of the ATLAS Software Documentation, two workflows are described for deploying ATLAS software in containers running images built via Dockerfiles. In the first option, a light-weight docker image with a basic Scientific Linux OS (see Figure 4) delivers ATLAS software through mounted CVMFS volumes [4]. In the second option, a complete release is installed in a docker image, which only depends on external CVMFS volumes for conditions and geometry data (see Figure 5). The CVMFS volume shown in Figures 4 and 5 is bind mounted to unprivileged containers via the CVMFS Docker Volume plugin supported by CERN IT [5]. CVMFS is installed in the host system where the cache can be shared across containers.

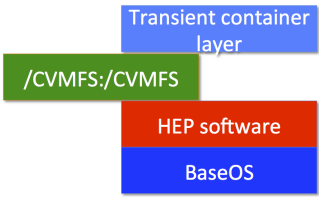

Figure 4. Light image (~2GB). ATLAS software available through CVMFS.

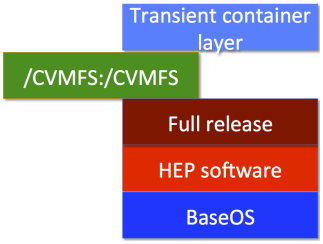

Figure 5. Full release (~23GB). CVMFS is used only for conditions data.

The ATLAS software documentation provides detailed information on how to install Docker on a personal Mac and Ubuntu machine. The deployment option (light vs. full-release) depends of course on the particular use case at hand. The light image is best suited for raw data reconstruction, which relies heavily on CVMFS for conditions and calibrations; while the stand-alone image is best fitted for environments with no connectivity to the Internet and for analysis and simulation jobs. External mounts guarantee small image sizes, but make it difficult to add a user code layer. On the other hand stand-alone images are large, but provide the best

encapsulation of software. Given their individual advantages, they both represent the current strategy for delivering ATLAS offline and/or analysis software on more modern OS platforms than Scientific Linux 6. In fact, there is currently an effort underway to produce the light Docker image as part of the nightlies production system.

Dockerfiles corresponding to the two build options shown in Figures 4 and 5 can be obtained from the ATLAS Gitlab docker repository using the 'git clone' command:

```
git clone https://:@gitlab.cern.ch:8443/atlas-sit/docker.git
```

As mentioned before, the Docker engine is instructed through the Dockerfile to execute instructions in a command-line fashion. The same could be done manually, of course, as when installing packages directly on a Linux box; however, a Dockerfile facilitates image building by providing a clean and standardized way of installing and configuring software on top of more basic images and/or prebuilt layers. The light image option differs little from the current strategy for delivering experiment-specific software to sites connected to the WLCG, that is, through mounted CVMFS volumes. The most important difference is that the whole runtime comes packaged in a container, which can in principle run on any minimally configured host. On the other hand, in the case of the stand-alone option, the ultimate goal is to provide complete independence from the network and network file systems. In this case the Dockerfile executes the commands necessary to build and configure an entire release from RPMs, much like the releases served through CVMFS are built. We currently provide examples of both types of images in a public registry at Docker Hub [6]. To pull the images, execute:

```
docker pull atlas/atlas_external_cvmfs # for the light image
docker pull atlas/athena:<RELEASE NUMBER> # for the full-release image
```

whether the light or full-release option is used, the resulting image can be used for software development by first cloning the Git Athena repository inside the container, and then follow the standard development workflow described in the ATLAS Software Documentation [7]. This tutorial describes the current framework to install packages, which is based on the Git version control system and the CMake build system.

3. Performance of Containers Running ATLAS Software

We document here a basic comparison in performance of the two deployment options discussed in section 2 in two platforms frequently used by developers: Ubuntu and OS X. The metrics measured are the wall-time and the CPU-time during a raw data reconstruction and a full simulation job. For each type of transformation the job is executed using both the light image, where ATLAS software is available through CVMFS, and the full-release image, where CVMFS is only used for conditions and geometry. As a reference, the jobs are also run on bare metal on a SLC6 test-bed machine.The general technical specifications of the machines used in the tests are listed in Table 1.

The reconstruction test job ran on 25 raw data events. Figures 6 and 7 show the performance of containers relative to bare metal when using the light image and the full-release image, respectively. Table 2 summarizes the results of the reconstruction test. The full-simulation test job ran on 5 events. Figures 8 and 9 show the performance of containers relative to bare metal, and Table 3 summarizes the results of the full-simulation test. All measurements were taken with a 'warm' CVMFS cache.

Table 1. Technical specifications of testing systems.

Test Machine	Description
`Mac laptop`	MacBook Pro, i7 2.2 GHz, macOS Sierra 10.12.6 Docker for Mac 17.06.0 configured for 2 cores and 4GB CVMFS 2.3.3, FUSE 3.5.4
`Linux desktop`	Intel Core 2 DUO CPU E7500, 2.93GHz, Memory 3.6GB Ubuntu 14.04, Docker 17.05.0, CVMFS 2.3.5
`Linux test-bed`	Intel Xeon CPU E5645 2.40GHz, Memory 2.4GB SLC6 6.8, bare metal, CVMFS 2.4.1

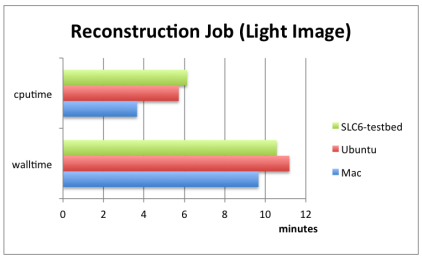

Figure 6. Light image.

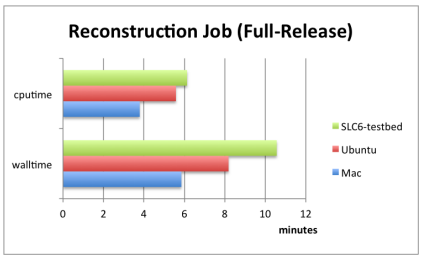

Figure 7. Full release image.

Table 2. Reconstruction test job (times in minutes).

System	Light image		Full release	
	wall-time (σ)	CPU-time (σ)	wall-time (σ)	CPU-time (σ)
Mac	9.66 (0.23)	3.66 (0.05)	5.85 (0.50)	3.79 (0.03)
Ubuntu	11.19 (0.20)	5.72 (0.01)	8.18 (0.35)	5.58 (0.01)
SLC6-testbed	10.56 (0.56)	6.13 (0.08)	10.56 (0.56)	6.13 (0.08)

σ is the sample standard deviation

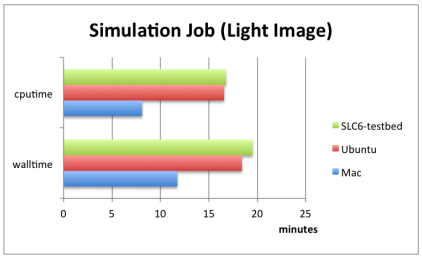

Figure 8. Light image.

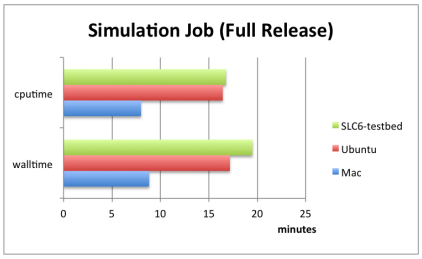

Figure 9. Full release image.

4. Consistency requirement of bind mounts in macOS

The current implementation of bind mounts in macOS provides a way of relaxing a default requirement of perfect consistency between the views of the mounted filesystem as seen from the host and from the container. The following flags can be added to the bind mount Docker arguments:

Table 3. Simulation test job (times in minutes).

System	Light image		Full release	
	wall-time (σ)	CPU-time (σ)	wall-time (σ)	CPU-time (σ)
Mac	11.75 (0.37)	8.12 (0.17)	8.84 (1.14)	8.00 (0.11)
Ubuntu	18.42 (0.28)	16.58 (0.03)	17.18 (0.24)	16.44 (0.03)
SLC6-testbed	19.51 (2.00)	16.77 (0.31)	19.51 (2.00)	16.77 (0.31)

σ is the sample standard deviation

- cached: delays are allowed before updates in the host appear in the container
- delegated: delays are allowed before updates in the container appear in the host

The same metrics used in section 3, wall-time and CPU-time, were compared for the three options specifying levels of consistency: consistent, cached and delegated in a reconstruction test. The degradation in performance seen when consistency is required (see Figure 10) is practically removed when the full-release image is used, regardless of the level of consistency (see Figure 11).

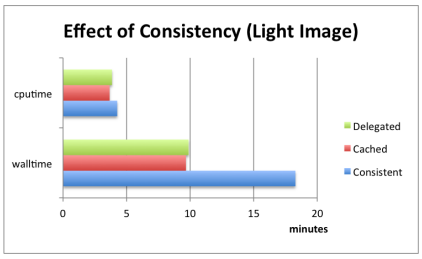

Figure 10. Light image. Relaxing consistency improves performance.

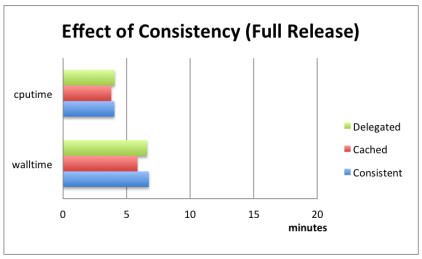

Figure 11. Full release image. Minimal use of cvmfs.

5. Summary and outlook

The preliminary tests carried out in this study showed that jobs running in containers with ATLAS software performed as well as the same jobs running on bare metal. Although the immediate objective here was to provide a platform for ATLAS software development, the long term goal is to exploit the advantages of containers in physics analyses and in production workloads such as event generation, simulation, reconstruction and physics derivations. The unique feature of containers of packaging and isolating software inside an entire file system represents an efficient implementation of a uniform runtime environment ideal for distributed computing.

References

[1] Docker: https://www.docker.com
[2] ATLAS Collaboration 2008 *J. Inst. 3* S08003.
[3] How to use Docker containers in ATLAS: https://atlassoftwaredocs.web.cern.ch/athena/intro-docker
[4] CernVM File System: http://cernvm.cern.ch/portal/filesystem
[5] CVMFS Docker Volume plugin: https://gitlab.cern.ch/cloud-infrastructure/docker-volume-cvmfs
[6] ATLAS Docker Hub image repository: https://hub.docker.com/u/atlas/dashboard
[7] ATLAS Software Git workflow: https://atlassoftwaredocs.web.cern.ch/gittutorial