PROCEEDINGS OF SCIENCE

Containerization of CMS Applications with Docker

Giulio Eulisse
CERN
E-mail: Giulio.Eulisse@cern.ch

Tommaso Boccali
INFN-Pisa
E-mail: Tommaso.Boccali@pi.infn.it

Enrico Mazzoni
INFN-Pisa
E-mail: enrico.mazzoni@pi.infn.it

Daniele Bonacorsi[∗]
University of Bologna and INFN-Bologna, Italy
E-mail: daniele.bonacorsi@unibo.it

Clouds and virtualisation offer typical answers to the needs of large-scale computing centres to satisfy diverse sets of user communities in terms of architecture, OS, etc. On the other hand, solutions like Docker seems to emerge as a way to rely on Linux kernel capabilities to package only the applications and the development environment needed by the users, thus solving several resource management issues related to cloud-like solutions. In this paper, an exploratory work done based on the workflows of the CMS experiment at the LHC accelerator at CERN is presented. Progresses towards the deployment of a full data center via exploiting containerised software stacks for CMS will be reported and discussed.

International Symposium on Grids and Clouds (ISGC) 2015,
15 -20 March 2015
Academia Sinica, Taipei, Taiwan

[∗]Speaker.

1. Introduction

Docker is an open platform to build, ship and run distributed applications. It is a container-based virtualisation framework that uses Linux containers (LXC), offering ease of packaging and deploying applications, a system based on lightweight containers, the use of UnionFS, as week as image storing and sharing via Docker Index/Hub. Recently, the commercial sector is showing a large set of use-cases and adoptions. Some examples are Spotify, that uses Docker for continuous delivery, service testing and deployment; Baidu, that uses Docker as a Platform-as-a-Service (PaaS), profiting of its flexibility for many framework and applications; eBay, that uses Docker for its easy application deployment, and for continuous integration process; and many more. Recently, Docker has reached 120k lines of code, about 10k commits and roughy 600 core contributors, of which about 95% work outside Docker.

2. Docker and CMS

Virtualization can be achieved in several ways, depending on the desired (or acceptable) level of invasiveness. A first option is chroot or cgroups, a second options is the use of containers, and a third option is the use of virtual machines. In the latter case, virtual machines imply a high overhead in both CPU and memory, plus a non negligible time needed for creating the images, booting up and for snapshotting. Docker provides CPU, memory and filesystem isolation, and can run different processes on the same kernel but with completely different runtime (e.g. one can run SLC5 on SLC6 or on Ubuntu).

Another interesting aspect is the availability of a service like Docker Hub (hub.docker.com). It offers the management of the lifecycle of distributed applications with cloud services for building and sharing containers and automating workflows. It allows to browse, search, push, pull repositories (about 100 official images, 45k public images), with easy mechanisms to retrieve (/upload) pre-build images via `docker pull (/push)` commands. Commands in a given image can be executed via `docker run`: they only share the kernel with the host OS (slc6, in the example case below):

```
$ docker run -it centos:centos7 cat /etc/redhat-release
CentOS Linux release 7.0.1406 (Core)
```

The images can be built using so called “Dockerfiles”, e.g.:

```
FROM cmssw/slc6-vanilla
    RUN yum -y update && yum -y install rubygems ruby-devel \
                                  gcc ruby193
    RUN echo "gem: --no-ri --no-rdoc" > ~/.gemrc && \
        gem install puppet && \
        gem install librarian-puppet -v 1.0.9
    CMD /bin/bash
```

All images are layered, there is no need to re-download previously downloaded layers (e.g. each of the above statements is a layer). Several CMS examples are documented [1].

Additionally, it is relatively straightforward to set-up Docker on one's own operating system of choice. Docker comes pre-packaged on most modern distributions, including slc6:

```
sudo yum install docker-io
sudo service docker start
```

The test set-up (used for most tests reported in this paper) consists of 2 socket Intel(R) Xeon(R) CPU E5-2630L 0 2.00GHz (12 real cores, 24 with HT); 2 GHz, 16 MB Cache, Rotating disks; Real HW, no hypervisor, Docker running directly on top of slc6. On the other hand, for Mac/Windows users an interesting option is boot2docker.io: it provides a VirtualBox environment and wrappers which make it look like a native setup. A lightweight Linux distribution based on Tiny Core Linux made specifically to run Docker containers is used; it runs completely from RAM, weighs 27MB and boots in few seconds.

3. Examples of CMS applications

As stated in the previous section, several CMS examples are documented [1]. One of these is particularly interesting and will be discussed briefly. "ParFullCMS" [2] is a parametrised Geant4-based geometric description of the CMS detector. It is mainly used for benchmarking (e.g. it is being used to do realistic tests of CMS software on ARM). It can be installed from the CMS apt repository, and running it is as easy as:

```
docker run -e EVENTS=2400 -v $PWD:/data -it cmssw/parfullcms
```

One can easily configure the number of events and the number of threads to use, and can run Geant4 simulation. Several tests done with ParFullCMS (3 runs, 2400 evts, 24 threads) with Docker and on bare metal show that the time performances are comparable, the confirming that no overhead is observed when using Docker.

Another interesting example is that one can ship the entire CMS software (CMSSW) as a whole to the host i.e. not downloading it via CVMFS, but just ship it all from the outside. This may become interesting to be exploited in case one wants to run a set of CMS specific tests which are usually run for CMS releases validation, e.g.:

```
docker run -e WORKFLOW=25.0 -it cmssw/cmssw:CMSSW_7_3_0
```

Suppose e.g. that one has a brand new machine, and wants to know how fast it is for CMS work-flows. It is sufficient to put (roughly) about 10 GB on a USB key and run a real CMSSW workflow and measure (10 GB correspond to a release plus the necessary input files to be shipped - any limit in the shipped size that Docker does not support out of the box can be overcome by starting it with particular options). This has been tested with a CMS reconstruction workflow (ttbar sample), and it was observed that bare metal vs Docker times are very similar, basically indistinguishable.

This opens the door to interesting application in terms of benchmarking. After SPEC and SPECfp, it was soon realised that standard benchmarking was not really adequate for HEP, as a machine benchmarked as 2x was not twice as fast for LHC typical applications (which are quite RAM-intensive, heavy on caches, etc). HepSpec06 was hence designed in 2006 to scale as typical

LHC experiments applications. Today, this is not completely true anymore, especially in moving to 64-bit. Docker (see previous example), might hence be the tool to offer a simple, portable way to run a benchmark. And this could be given to hardware vendors to run realistic (and a large set, and always up-to-date) benchmarks of CMS workflows.

4. "Dockerizing" a computing site

The CMS Pisa Tier-2 center is one of the biggest Italian computing centers. It is a Tier-2 in WLCG, supporting CMS, but CMS is actually a minority part of the resources: these nclude, to a smaller extent, also LHCb and ATLAS, 20 more Virtual Organizations, a National Theoretical Physics computing center, a 2000 (and more) cores fluidodynamic cluster (used in industry). In terms of resources it consists of 8k cores, >2 PB, highly heterogeneous WN hosts (some are 1 Gbe, some 10 Gbe, some Infiniband). Also the OS requests are heterogeneous (some still SL5, even SL4 up to some months ago; some prefer very recent releases, generally OpenSuSe). The only common points between such a diverse user-community are: *i)* GPFS is used to serve data for all the use-cases supported on-site; *ii)* AFS is used for user areas; *iii)* LSF is used for resource access.

The main issue in Pisa recently was to understand how to provision the correct environment to all these diverse resources. Virtual Machines (e.g. OpenStack) are an option but Pisa also has older machines which are low in RAM. Infiniband seems to loose performance in a completely virtualised environment, also. The solution identified so far is a very light virtualisation via chroot. Every host machine (with the very latest OpenSUSE) has a tar file from which it uncompress an SL6 (or SLx) and then chroot to this and it just works. Pisa mounts natively all the file-systems (CVMFS/GPFS/AFS) to the host and they are passed via bind to the chroot environment. Every machine has pre-installed as many tar files as the possible environments are: these can be un-tarred and "started" on demand via a set of scripts, basically deciding on the flight which pool the machine lives in. This solution works and it is in production for Pisa since 3 years.

On the other hand, Docker seems to be a viable solution, too. The same overhead (virtually 0) was expected, but it should be easier to manage. Basically, Two are the main points in favour of Docker adoption in Pisa: *i)* no more tar files and a complicate management; adopt a system that has a container management a-la git; everything is nicely logged, you can branch, you can pull, you can push, etc. Also, a local registry is set-up (to avoid exposure of sensitive information). *ii)* it starts from a running system based on tar; if you just start from a tar file (chroot-based set-up), Docker provides a single command to build the container:

```
sudo tar -C raring -c . | sudo docker import - raring
```

A Tier-2 on-demand approach was attractive, but site performances for a site in production are most important at the moment. For example, Pisa still decided to mount CVMFS/GPFS/AFS on the host and pass this via "-v". This is not something you can ask e.g. in opportunistic sites, but this *i)* avoids running docker privileged, and *ii)* avoids multiple copies of caches (CVMFS/AFS) if more than one container is running. LSF is also in the container, so it automatically connects the machine to the proper LSF queue, with a start command as follows:

```
docker run -v /cvmfs:/cvmfs -v /afs:/afs -v /gpfs/ddn:/gpfs/ddn
-v /chrootlfs/home:/home/grid -d
-t localregistry.pi.infn.it:5000/enricomazzoni/testwn:0.3
/etc/sysconfig/docker-pi/start
```

Pisa moved 10% of the WNs to Docker so far, all work done in 7-10 days. No user actually realised the difference. The rest of the centre will migrate as well in the next few weeks. At this point, the idea would be to move all the services used by CMS via Docker, e.g.: *i)* Squid: just a Linux standard machine; *ii)* CEs: Pisa moved one and itâĂŹs working; *iii)* UIs: easy, similar to WNs; *iv)* PhEDEx (transfer agents on a UI), it should be as easy as a UI; *v)* an xrootd redirector (data federations): Pisa moved it from chroot and it is already running now with Docker.

5. Conclusions

In summary, Docker offers interesting opportunities to CMS. Itcan ship entire CMSSW distributions and do benchmarking easily. It offers plenty of desired features and flexibility at a site level, and all this at no cost in terms of performances (as from the tests done so farm, at least).

The transition in itself is very easy, indeed matter of days, admittedly maybe also because Pisa was already at chroot point - but other works in CMS show that starting from bare metal is as easy, and Docker provides most of the specific scripts a site may need for image processing, download history, starting, stopping.

In a nutshell, it s a neat tool, and CMS will continue testing it (and possibly integrating it) in its activities.

References

[1] CMS examples with Docker: `http://github.com/cms-sw/cms-docker`

[2] ParFullCMS,
`https://github.com/cms-sw/cms-docker/tree/master/parfullcms`