Containerization of CMS applications with Docker

G. Eulisse, T. Boccali, E. Mazzoni, D. Bonacorsi

March 20th, 2015
Outline

Part 1: Docker and the CMS software

Part 2: Docker experience at a CMS Tier-2
Part 1
Docker and the CMS software
Open platform to build, ship and run distributed applications

✦ A container-based virtualisation framework that uses Linux containers (LXC)

Ease of packaging and deploying applications

✦ Lightweight containers, use of UnionFS
✦ Image storing and sharing via Docker Index / Hub

Large set of use-cases and adoptions in the commercial sector

✦ Spotify - Continuous delivery, service testing and deployment

✦ Baidu - Platform-as-a-Service (PaaS), flexibility for many framework and apps

✦ eBay - Easy application deployment, continuous integration process

✦ just examples… many more, and new ones appearing at a high rate …
A growing community
Why exploring Docker for CMS?

You can do virtualization in several ways, depending on invasiveness:

✦ chroot or cgroups
✦ containers
✦ virtual machines

With VMs:

✦ high overhead in both CPU and memory
✦ time needed for creating the images, booting up and for snapshotting

Docker provides CPU, memory and filesystem isolation. Runs different processes on the same kernel, but with completely different runtime

✦ e.g. you can run SLC5 on SLC6 or on Ubuntu

Interesting in many ways.
Docker Hub

**hub.docker.com**

- Management of lifecycle of distributed apps with cloud services for building and sharing containers and automating workflows
- Browse, search, push, pull repositories
  - 100 official images, ~45k public images
- You can retrieve (/upload) pre-build images via `docker pull (/push)`
- It provides a repository for pre-build images. Can be retrieved via `docker pull`
- Commands in a given image can be executed via `docker run`
  - they only share the kernel with the host OS (slc6, in the case below):

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

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

```bash
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
```

Images are layered, no need to re-download previously downloaded layers

✦ Each of the above statements is a layer

Few CMS examples (credits: G.Eulisse) here:

Setting up Docker on your OS

Docker comes pre-packaged on most modern distributions
✦ including slc6

\[
\text{sudo yum install docker-io} \\
\text{sudo service docker start}
\]

Test set-up (used for everything shown in next slides)
✦ 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

For Mac / Windows users:
✦ boot2docker.io: provides a VirtualBox environment and wrappers which make it look like a native setup
  - lightweight Linux distribution based on Tiny Core Linux made specifically to run Docker containers
  - It runs completely from RAM, weighs ~27MB and boots in few seconds
See next

github.com/cms-sw/cms-docker
“ParFullCMS” is a parametrized G4-based geometric description of the CMS detector

✧ Done for benchmarking. E.g. used to do tests on ARM

It can be installed from the CMS apt repository

✧ see https://github.com/cms-sw/cms-docker/tree/master/parfullcms

Running as easy as:

```
docker run --env EVENTS=2400 --volume=$PWD:/data -it cmssw/parfullcms
```

You can configure the nb evts and the # threads to use, and you run G4 simulation:

✧ see next
$ docker run -e EVENTS=2400  -v $PWD:/data -it cmssw/parfullcms
(…)

Geant4 version Name: geant4-10-00-patch-03 [MT] (31-October-2014)
<< in Multi-threaded mode >>

Copyright : Geant4 Collaboration
Reference : NIM A 506 (2003), 250-303
WWW : http://cern.ch/geant4

### Number of threads is forced to 4 by Environment variable G4FORCENUMBEROFTHREADS.

(…)

G4WT0 > ### Run 0 starts on worker thread 1.
G4WT0 > Run terminated.
G4WT2 > Run terminated.
G4WT3 > Run terminated.
G4WT1 > Run terminated.

Run Summary
Number of events processed : 2400
User=4224.1s Real=1077.34s Sys=0.53s

Final random number = 0.609626

Still to be understood, but at least it stands as a confirmation that with docker it is not slower!
Another example

With Docker, you can ship CMSSW as a whole to the host

✦ i.e. not downloading it via CVMFS, just ship it all from the outside!
✦ interesting to exploit it to run a set of CMS specific tests which are usually run for CMS releases validation

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

If e.g. you have a new machine, and want to know how fast it is for CMS workflows:

✦ you put the ~10 GB on a USB key and run a real CMSSW workflow and measure
✦ 10 GB is for 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

Done this with a CMS reconstruction workflow of a ttbar sample, bare metal vs Docker

✦ **times are very similar, basically indistinguishable!**
Benchmarks

Originally, we had SPEC and SPECfp..

Standard benchmarking was found not really adequate for HEP
- a machine benchmarked as 2x was not twice as fast for our app, quite RAM-intensive, heavy on caches, etc

HepSpec06 was designed in 2006 to scale as typical LHC experiments applications

Now, not completely true anymore…
- especially in moving to 64-bit

Docker (see previous slide’s example), may be the tool to offer a simple, portable way to run a benchmark.
- This could be given to hardware vendors to run realistic (and a large set, and always up-to-date!) benchmarks of CMS workflows.
Part 2

“Dockerizing” a site
The CMS Pisa Tier-2

Pisa is one of the biggest Italian computing centers
It is a Tier-2 in WLCG, supporting CMS
But CMS is a minority part of the resources, which include

✦ to a smaller extent, also LHCb and ATLAS are supported
✦ >20 other VOs supported
✦ a National Theoretical Physics computing center
✦ a 2000+ cores fluidodynamic cluster, used in industry

Resources: ~8k cores, >2 PB

✦ WN hosts are highly heterogeneous
  - Some 1 Gbe, some 10 Gbe, some Infiniband
✦ OS requests also heterogeneous
  - Some still SL5, even SL4 up to some months ago; some prefer very recent releases (OpenSuSe, generally)

Only common points are:

✦ GPFS is used to serve data for all the use-cases supported on-site
✦ AFS is used for user areas
✦ LSF for resource access
Provision the correct environment

Problem:
The main issue was understanding how to provision the correct environment to all these diverse resources. Perhaps VMs? (e.g. OpenStack)

✦ older machines are low in RAM..
✦ Infiniband seems to loose performance in a completely virtualised env..

Solution (so far):
A very light virtualisation via chroot

✦ every host machine (with the very latest OpenSUSE) has a tar file from which it uncompressed an SL6 (or SLx) and then chroot to this and it works
✦ more in next slide
Current solution: chroot

OpenSUSE (as recent as possible) is used. Pisa mounts natively all the file-systems (CVMFS/GPFS/AFS) to the host and they are passed via bind to the chroot env.

Every machine has pre-installed as many tar files as the possible environments are:

- which can be un-tarred and “started” on demand via a set of scripts, basically deciding which pool the machine lives in.
- It needs to manage a set of scripts to select which machine starts and when / where to get the tar files, etc.

It works, in production since 3 years.

But..

- Docker seems to be a viable solution, too.
- Same overhead (virtually 0) but should be easier to manage.
Two score points for Docker in Pisa

Main points in favour of Docker adoption in Pisa:

1. 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)

2. start 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
```
The final approach

A Tier-2 on-demand approach was attractive
✦ but site performances are most important at the moment

E.g. 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 in this way:
✦ avoids running docker privileged
✦ avoids multiple copies of caches (CVMFS/AFS) if >1 container is running

LSF in the container
✦ Automatically connects the machine to the proper LSF queue
✦ start command:

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
So far so good!

Pisa moved ~10% of the WNs to Docker so far

- Over last 1 week or so
- No user actually realised the difference.
- Rest of the site will follow soon!

At this point, the idea would be to move all the services used by CMS via Docker, e.g.:

- **Squid**: just a Linux standard machine
- **CEs**: Pisa moved one and it’s working
- **UIs**: easy, similar to WNs
- **PhEDEx** (transfer agents on a UI): did not try but it is just a “special” UI
- **xrootd** redirector (data federations): moved from chroot and already running now with Docker
Conclusions

Docker offers interesting opportunities to CMS.

✦ can ship entire CMSSW distributions and do benchmarking easily
✦ offers plenty of desired features and flexibility at a site level
✦ all this at 0 cost (it seems!) in terms of performances (as from the tests done so farm, at least)

The transition in itself is *very* easy, indeed matter of days

✦ 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...

A neat tool, and CMS will continue testing it (and possibly integrating it) in its activities.