An Initial Evaluation of Docker at the RACF

Chris Hollowell <hollowec@bnl.gov>
RHIC/ATLAS Computing Facility
Brookhaven National Laboratory

What is Docker?

Docker is an opensource software project that helps automate, and simplify the deployment, management, and instantiation of applications in OS containers

- Uses containerization features built into the modern Linux kernel (2.6.26+)
  - Cgroups
  - Mount namespaces
  - PID namespaces
  - Network namespaces
- Red Hat only officially supports docker on RHEL7+
- Available for Windows 7.1/8 – runs Docker daemon in a Linux VM
  - Windows Server 2016 support announced using Hyper-V containers
- Oracle recently announced they will be integrating Docker into Solaris, utilizing Zones

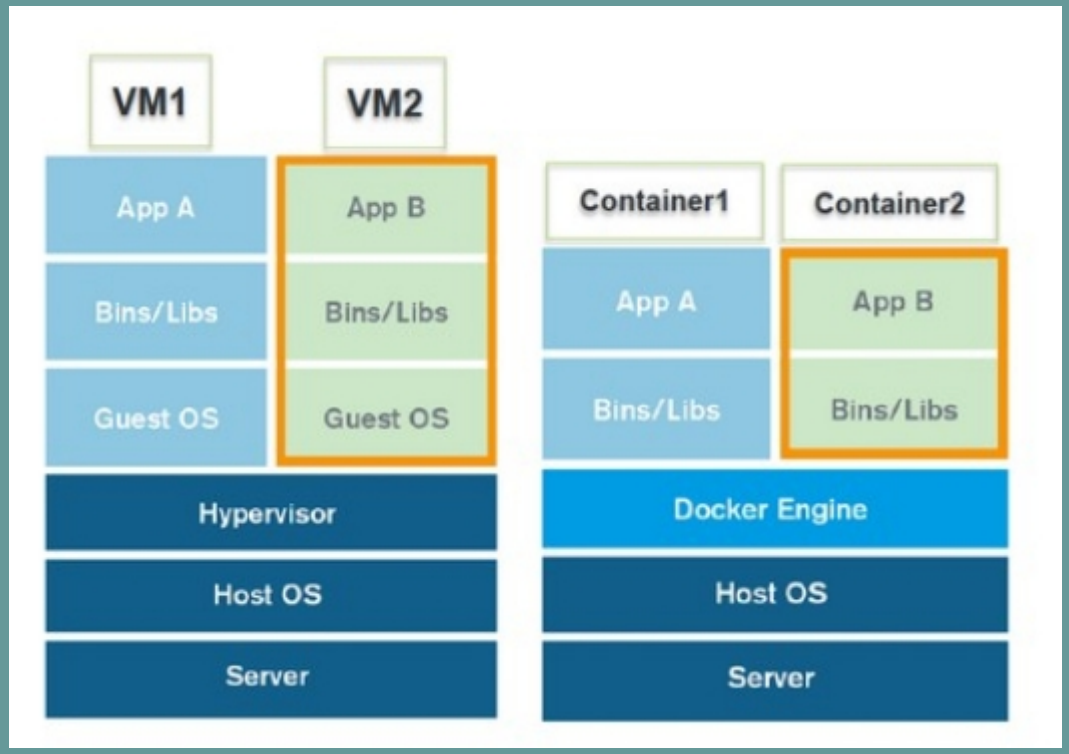
Initial release in March 2013, and has quickly become popular

Currently the basis for Amazon's EC2 Container Service

Containers vs Virtualization

- Virtualization: Hardware virtualization
  - Hypervisor presents fully abstracted guest instances of the host hardware platform (x86_64, etc.)
- Containers: OS-level virtualization
  - OS kernel presents multiple guest instances of itself
  - Guest OS userland may be different than the host's, assuming ABI compatibility with the running kernel
    - Linux kernel notoriously strict about maintaining ABI compatibility
    - For instance, allows one to run:
      - An SL6 container on SL7
      - A Debian 7 container on SL7
      - etc.
  - Can reduce overhead associated with starting/stopping VMs
  - Eliminates performance penalties associated with virtualizing hardware

Containers vs Virtualization (Cont.)



Layers of Hardware vs OS Virtualization

More than 99.9% of the software run on our servers at RACF is developed for Linux, but often for a specific distribution and/or release

- In this situation, full hardware/platform virtualization is overkill
- A perfect problem for containers to solve
- Likely the case at other HEP/NP labs?

Containers vs Virtualization (Cont.)

Containers for UNIX have been around for a long time
- chroot(2) filesystem level virtualization has been available since 1982
  - CHOS from NERSC available since at least 2004
- FreeBSD jails introduced in the 4.0 release in 2000
- Virtuozzo available for Linux since 2000, and OpenVZ since 2005
  - These require custom kernels

Why is there a more recent focus on containers for Linux?
- With the introduction of Linux kernel namespaces and Cgroups, container support is included in vanilla/unmodified distributions
- Tools like LXC and Docker simplify creating and managing containers

LXC vs Docker

Both LXC and Docker make use of Linux kernel namespaces for containerization

LXC is focussed on instantiating long running containers, with the full userland init system executed, potentially providing multiple services

- Similar to typical hardware VM instantiation models – boot a fully usable system

Docker's strength is running single applications in a container

- If the application exits, so does the container
- Docker philosophy is that one should provision a container as minimally as possible
  - Install the smallest number of required OS packages and services to support the application being used in the container
  - HTC batch jobs should fit well into this design

Red Hat is focussing on Docker

- Deprecated libvirt-lxc support in RHEL 7.1

Docker Basics

Download pre-built image from Docker Hub:

```
docker pull IMAGE (i.e. docker pull centos)
```

Execute a process, or an interactive shell in a container:

```
docker run -t IMAGE PROGRAM (i.e. docker run -i -t centos /bin/sh)
```

List container images:

```
docker images
```

Commit changes to a container:

```
docker commit CID NEWIMAGE
```

List containers:

```
docker ps -a
```

Restart a container:

```
docker restart CID
```

Stop a container:

```
docker kill CID
```

Delete a container:

```
docker rm CID
```

Start your own local registry server (alternative to Docker Hub):

```
docker run -d -p 5000:5000 --restart=always —-name registry registry:2
```

Point push/pull commands, etc. to your local registry, i.e:

```
docker pull localhost:5000/centos
```

Kernel-Userland ABI Issue Encountered

Encountered when running 32-bit software in an SL6 Docker container on an SL7 host

Using XFS for the host OS filesystems (default option)

- XFS supports 64-bit inode numbers
  - This (“inode64”) is the default XFS SL7 (and for kernels >3.7) mount option

Experienced strange issues when running 32-bit software calling stat()

- For instance 32-bit SPEC CPU2006 reported missing files

After signifcant debugging/strace-ing, determined “missing” files had inode numbers greater than 2^{32}

32-bit SL6 glibc stat() not compatible with 64-bit inode numbers (type ino_t)

Using “inode32” mount option solved the problem

Need to be aware that while user-facing ABI compatibility between Linux kernels is important to the kernel developers, esoteric issues like this can still occur

Docker Security Issues?

Docker daemon runs as the root user

By default, uses UNIX sockets so only local access is possible

Any user who can run containers via the daemon (by default, users in the ”docker” group) can map host directories inside a guest container, and thus have root access in those directories, i.e.:

```
docker run -v /etc:/host/etc sl6 /bin/sh
```

The above maps /etc on the system to /host/etc in the container

Docker doesn't currently (release 1.8.x) natively support user namespaces, which could potentially be used to fix this problem

- root in the container would map to a different user in the host OS
- This feature is expected at some point in the future

Benchmarks

Interested in measuring the performance of processing jobs in docker containers versus bare metal and VMs

Tests performed using Docker 1.6.2

Evaluation hardware:

- Dell PowerEdge R720xd
- 2 Intel Xeon E5-2660v2 CPUs @2.20 GHz
- 80 GB 1866 MHz DDR3 RAM
- PERC H730 Controller
- 12 2TB drives in a hardware RAID5

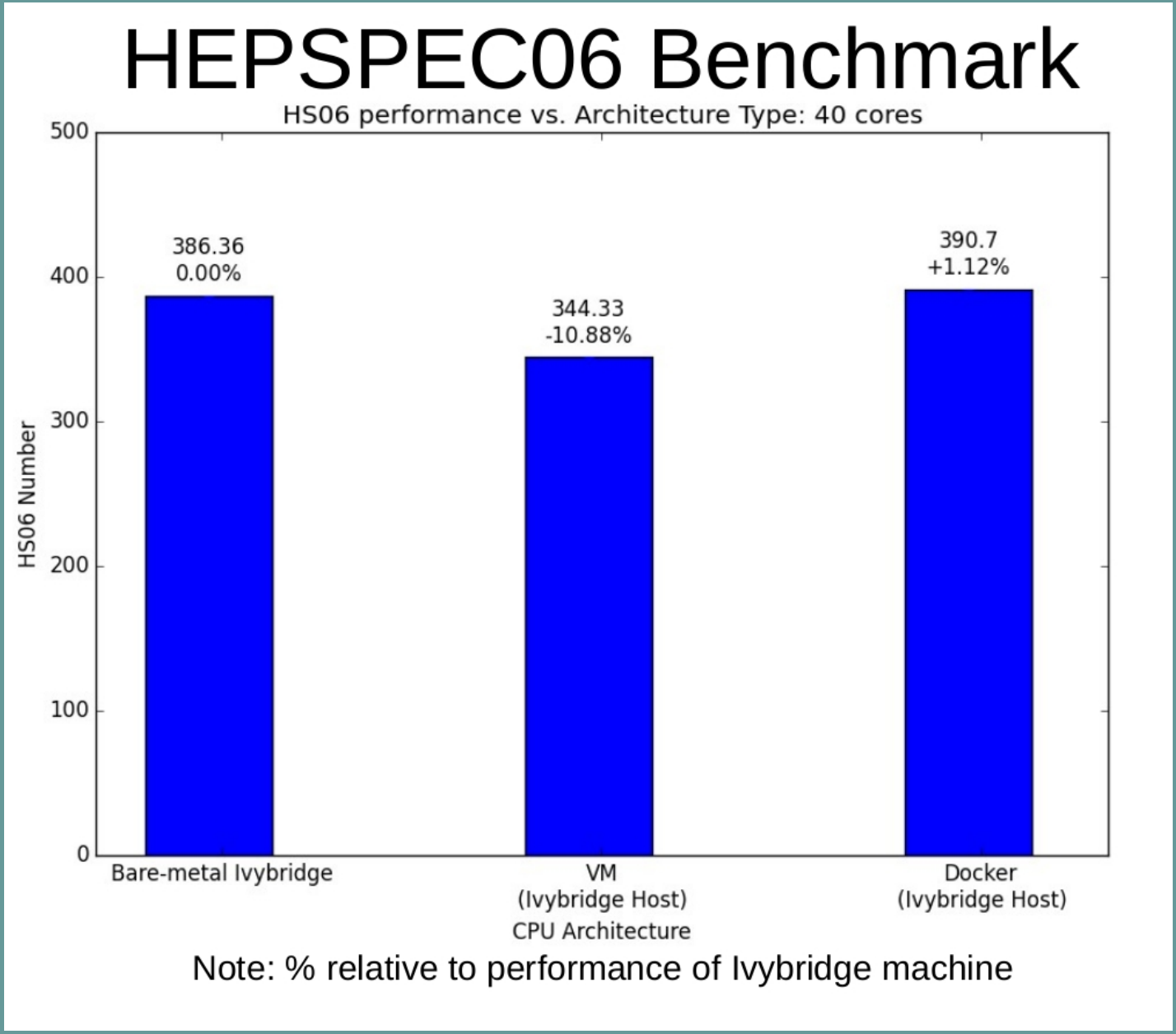
HEPSPEC06

- Standard HEP/NP CPU benchmarking suite
- Based on SPEC CPU2006
- Measures performance of entire system:
  - One instance of the benchmark is run per (logical) core

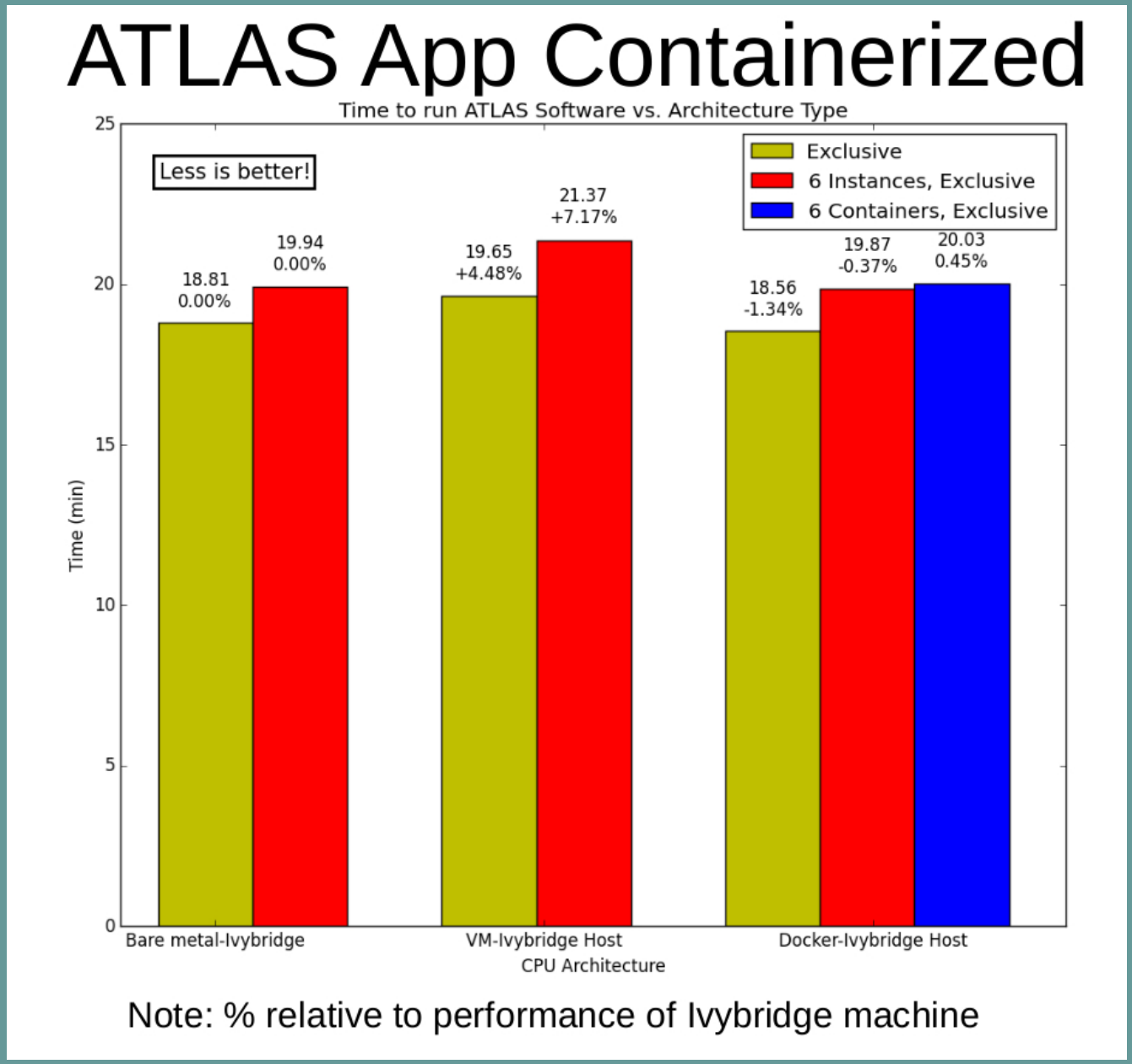
Timed ATLAS software KitValidation execution

- Event generation, Geant4 simulation, digitization,reconstruction
- Ran single instance, and multiple instances

HEPSPEC06


HEPSPEC06 Benchmark
HS06 performance vs. Architecture Type: 40 cores
HS06 Number
500
400
300
200
100
0
386.36
0.00%
344.33
-10.88%
390.7
+1.12%
Bare-metal Ivybridge
VM
(Ivybridge Host)
Docker
(Ivybridge Host)
CPU Architecture
Note: % relative to performance of Ivybridge machine

ATLAS Software (KitValidation)



Docker and HTCondor

Docker universe available in HTCondor 8.3.6+

Ran some basic tests: functioned well

```
####################################
# Example Docker Universe JDF
#
universe         = docker
docker_image     = sl6
executable       = /bin/cat
arguments        = /etc/redhat-release

output           = test.out$(Process)
error            = test.err$(Process)
log              = test.log$(Process)

queue
```

HasDocker classad

Other than execute directory, couldn't arbitrarily mount host or NFS filesystems into the container

Users processes do not enter the container as root

Conclusions

Docker containers are currently a compelling alternative to hardware VMs for software and facilities which have standardized on the use of Linux

- Improved performance
  - Removes penalties associated with needlessly virtualizing hardware/devices
  - Can reduce spin-up/down times
- Simplified management
- Easy for developers to package dockerized applications
  - Need to know less about OS and system administration than with full VMs

Cloud and batch system software beginning to support Docker containers

- Amazon EC2 Containers
- OpenStack Docker Driver
- Condor Docker Universe

Some small hurdles to clear before Docker can be fully adopted

- Some security concerns
- Lack of multiple host mounts such as NFS in the Condor docker universe

Acknowledgments

Thanks to Joe Zuhusky, a student intern at RACF this summer, for his help in conducting this study, and for providing a number of the graphs/graphics used in this presentation.

Thanks to Costin Caramarcu, Tejas Rao, William Strecker-Kellogg, Tony Wong, and Alexandr Zaytsev for their contributions to this project.