Docker and HTCondor

Greg Thain

HTCondor Week 2016

Start with the Basics…

HTCondor is designed to:

- Allow a machine “owner” to loan it out
- The machine must be protected from job

Ancient History: Chroot

HTCondor used to chroot every job:

1. No job could touch the file system
2. Private files in host machine stayed private

Chroot: more trouble than value

Increasingly difficult to work:

- Shared libraries
- /dev
- /sys
- /etc
- /var/run pipes for syslog, etc.

How to create root filesystem?

Easier now with yum, apt get, etc., but still hard:

We gave up!

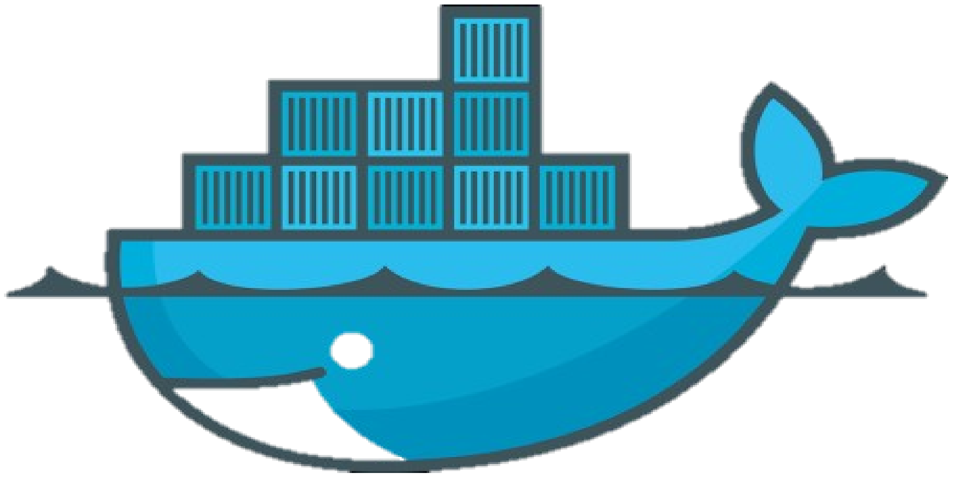
HTCondor no longer chroots all jobs

But you can optionally do so.

Very few site sites do…

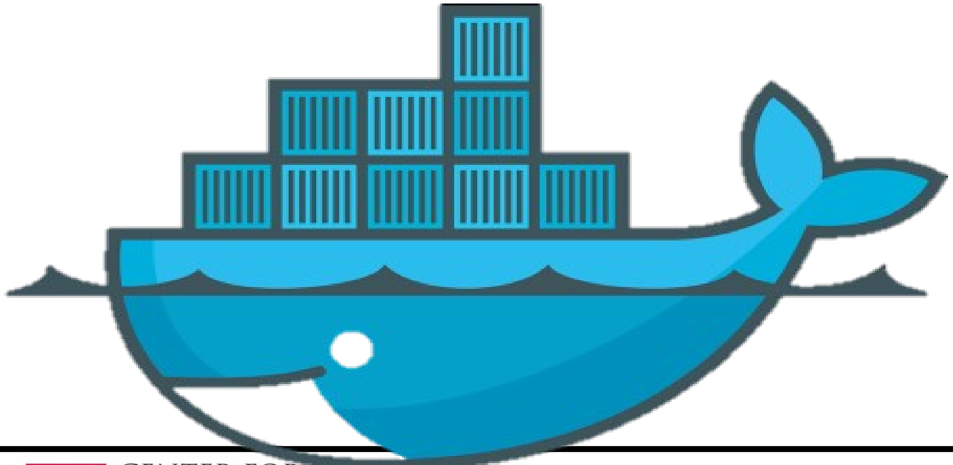
NAMED_CHROOT = /foo

Enter Docker!



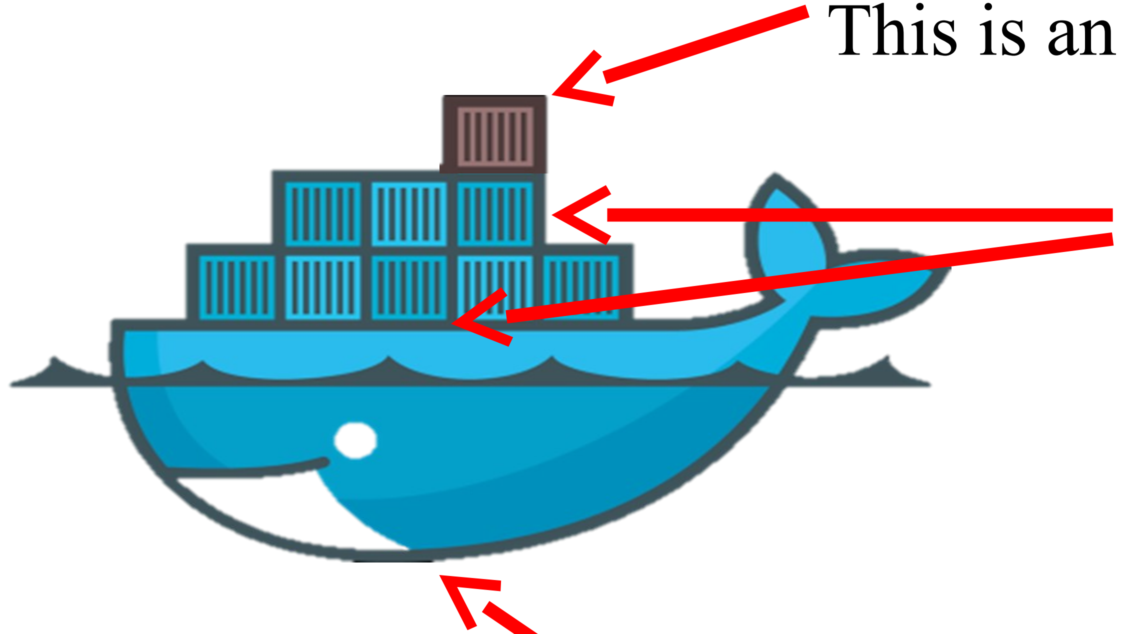
This is Docker

Docker manages Linux containers.
Containers give Linux processes a private:



- Root file system
- Process space
- NATed network
- UID space

Examples

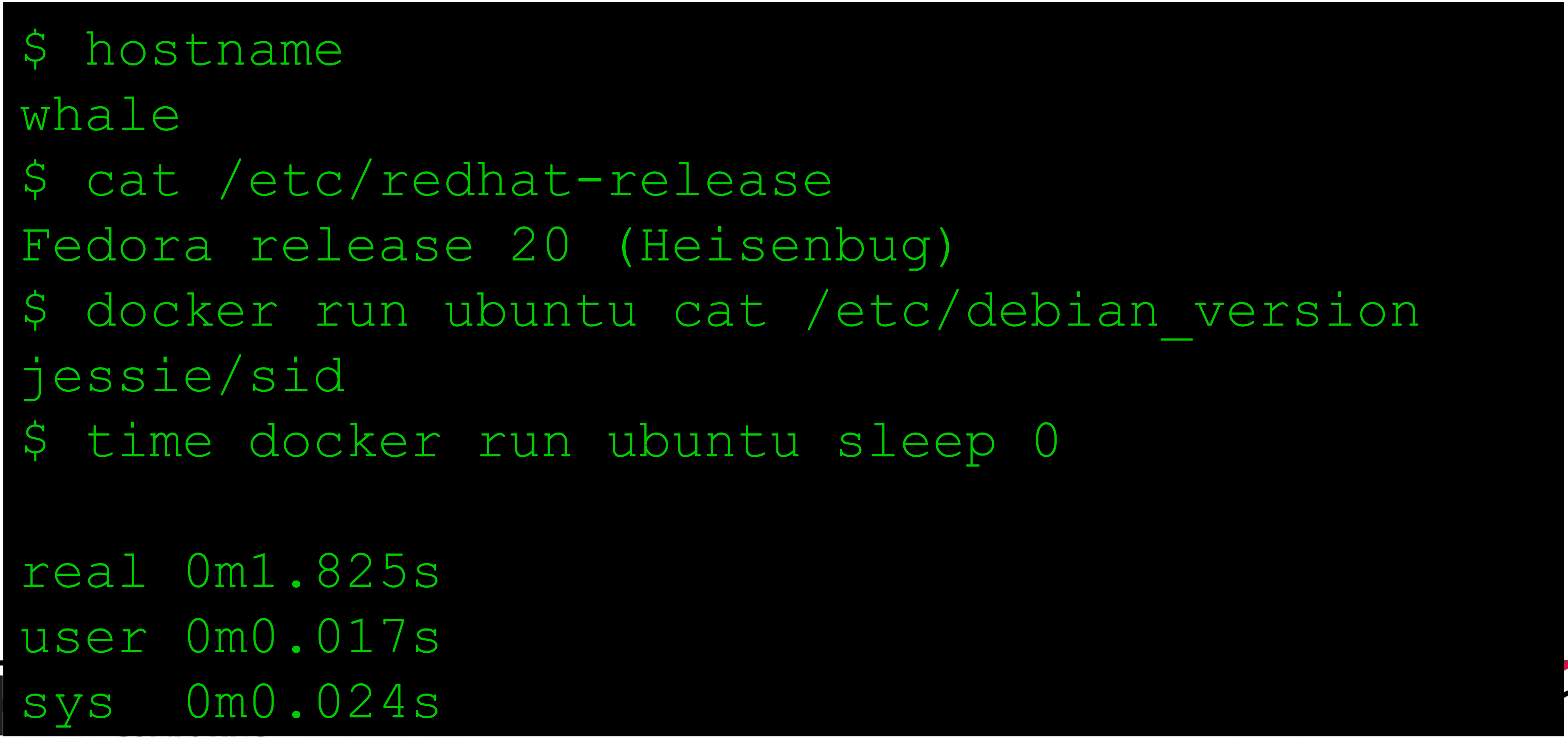


This is an "ubuntu" container

Processes in other containers on this machine can NOT see what's going on in this "ubuntu" container

This is my host OS, running Fedora

At the Command Line



```
$ hostname
whale
$ cat /etc/redhat-release
Fedora release 20 (Heisenbug)
$ docker run ubuntu cat /etc/debian_version
jessie/sid
$ time docker run ubuntu sleep 0

real 0m1.825s
user 0m0.017s
sys  0m0.024s
```

More CLI detail

```
$ docker run ubuntu cat /etc/debian_version
```

"cat" is the Unix process, from the image we will run (followed by the arguments)

"ubuntu" is the base filesystem for the container an "image"

"run" command runs a process in a container

All docker commands are bound into the "docker" executable

Images

Images provide the user level filesystem

- Doesn’t contain the linux kernel
- Or device drivers
- Or swap space

Very small: ubuntu: 200Mb.

Images are READ ONLY

Where images come from

Docker, inc provides a public-access **hub**

Contains **10,000+** publically usable images behind a CDN

What's local?

```
 $ docker images
```

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
new_ubu             latest              b67902967df7        8 weeks ago         192.7 MB
<none>              <none>              dd58b0ec6b9a        8 weeks ago         192.7 MB
<none>              <none>              1d19dc9e2e4f        8 weeks ago         192.7 MB
rocker/rstudio      latest              14fad19147b6        8 weeks ago         787 MB
ubuntu              latest              d0955f21bf24        8 weeks ago         192.7 MB
busybox             latest              4986bf8c1536        4 months ago        2.433 MB
```

How to get

```
 $ docker search image-name
 $ docker pull image-name
```

Wait!
I don’t want my images public!

Easy to make your own images (from tar files)

The docker hub is open source

Straightforward to start your own

How is it distributed?

Under the hood of images

Images are composed of layers

Images can share base layers:

ubuntu	: 200 Mb
ubuntu + R	: 250 Mb
ubuntu + matlab	: 250 Mb

All three: 300 Mb.

Container vs. Image

Image is like Unix program on disk
- read only, static

Container is like Unix process

Docker run starts a container from an image

Container states: like a condor job:
- Running
- Stopped

Containers

```
$ docker ps

CONTAINER ID IMAGE         COMMAND     NAMES
b71fff77e7b9  ubuntu:latest /bin/sleep  owly_tannenba
```

shows running containers

```
$ docker ps -a
CONTAINER ID IMAGE         COMMAND     NAMES
b71fff77e7b9  ubuntu:latest /bin/sleep  owly_tannenba
7eff0a4dd0b4  debian:jessie /bin/sleep  owly_tannenba
```

Operations on Containers

```
$ docker ps -a

$ docker run …

$ docker stop containerId

$ docker restart containerId

$ docker rm containerId
```

Where is my output?

```
$ docker diff containerId
$ sudo docker diff 7bbb
C /dev
A /dev/kmsg
C /etc
D /foo
$ docker cp containerId:/path /host
```

Works on running or stopped containers

Or, use “volumes”

```
$ docker run –v /host:/container …
```

Volume is a directory that isn’t mapped

Output to volume goes directly to host

Fast: just a local mount

Why should you care?

- Reproducibilty
  - How many .so’s in /usr/lib64 do you use?
  - Will a RHEL 6 app run on RHEL 9 in five years?
- Packaging
  - Image is a great to package large software stacks
- Ease of inspection and management

- Imagine an OSG with container support!

Docker and HTCondor

New “docker universe”

- (not actually new universe id)

Installation of Docker universe

Need docker (maybe from EPEL)

```
$ yum install docker-io
```

Condor needs to be in the docker group!

```
$ useradd -G docker condor
$ service docker start
```

What? No Knobs?

Default install should require no condor knobs!

But we have them anyway:

```
DOCKER = /usr/bin/docker
```

Condor startd detects docker

```
$ condor_status -l | grep -i docker
HasDocker = true
DockerVersion = "Docker version 1.5.0,
build a8a31ef/1.5.0"

$ condor_status -const HasDocker
```

Check StarterLog for error messages

Docker Universe

```
universe = docker
executable = /bin/my_executable
arguments = arg1
docker_image = deb7_and_HEP_stack
transfer_input_files = some_input
output = out
error = err
log = log
queue
```

Docker Universe Job
Is still a job

› Docker containers have the job-nature
  - condor_submit
  - condor_rm
  - condor_hold
  - Write entries to the ~~user log~~ event log
  - condor_dagman works with them
  - Policy expressions work.
  - Matchmaking works
  - User prio / job prio / group quotas all work
  - Stdin, stdout, stderr work
  - Etc. etc. etc.*

Docker Universe

```
universe = docker
executable = /bin/my_executable
```

Executable comes either from submit machine or image

NEVER FROM execute machine!

Docker Universe

```
universe = docker
# executable = /bin/my_executable
```

Executable can even be omitted!

trivia: true for what other universe?

(Images can name a default command)

Docker Universe

```
universe = docker
executable = ./my_executable
input_files = my_executable
```

If executable is transferred,
Executable copied from submit machine
  (useful for scripts)

Docker Universe

```
universe = docker
executable = /bin/my_executable
docker_image =deb7_and_HEP_stack
```

Image is the name of the docker image stored on execute machine. Condor will fetch it if needed.

Docker Universe

```
universe = docker
transfer_input_files= some_input
```

HTCondor can transfer input files from submit machine into container

(same with output in reverse)

Condor's use of Docker

Condor volume mounts the scratch dir

Condor sets the cwd of job to the scratch dir

  Can't see NFS mounted filesystems!

Condor runs the job with the usual uid rules.

Sets container name to

  HTCJob_$(CLUSTER) _$(PROC)_slotName

Scratch dir == Volume

Means normal file xfer rules apply

- transfer in, transfer out
- subdirectory rule holds
- condor_tail works

RequestDisk applies to scratch dir, not container

Any changes to the container are not xfered

Container is removed on job exit

Docker Resource limiting

```
RequestCpus = 4
RequestMemory = 1024M
RequestDisk = Somewhat ignored…
```

- RequestCpus translated into cgroup shares
- RequestMemory enforced
  - If exceeded, job gets OOM killed
  - job goes on hold

Why is my job on hold?

Docker couldn't find image name:

```
$ condor_q -hold

-- Submitter: localhost : <127.0.0.1:49411?addrs=127.0.0.1:49411
> : localhost
 ID       OWNER             HELD_SINCE  HOLD_REASON

 286.0    gthain             5/10 10:13 Error from slot1@localhost:
Cannot start container: invalid image name: debain
```

Exceeded memory limit?

Just like vanilla job with cgroups

Custom Volume Mounts

- Admin-specified
  - DOCKER_VOLUMES = A, B
  - DOCKER_VOLUME_DIR_A = /path1
  - DOCKER_VOLUME_DIR_B = /path2:ro
  - DOCKER_MOUNT_VOLUMES = A, B
- HasDockerVolumesA = true

Summary

Docker universe runs containers like jobs

Could be game-changing

Very interested in user feedback