Putting your users in a Box

John (TJ) Knoeller
Center for High Throughput Computing

Outline

- Why put jobs in a box?
- Boxes that work everywhere
- Shiny new Linux-only boxes
- Older Linux-only boxes

3 Protections

1) Protect the machine from the job.

2) Protect the job from the machine.

3) Protect one job from another.

The ideal box

- Allows nesting
- Need not require root
- Can't be broken out of
- Portable to all OSes
- Allows full management:
  - Creation // Destruction
  - Monitoring
  - Limiting

Resources a job can (ab)use

- CPU
- Memory
- Disk
- Network.
- Signals

The Big Hammer

- Some people see this problem, and say
- “I know, we’ll use a Virtual Machine”

Problems with VMs

- Might need hypervisor installed
  - The right hypervisor (the right Version…)
- Need to keep full OS image maintained
- Difficult to debug
- Hard to see into
- Hard to federate
- Just too heavyweight

Containers, not VMs

- One way glass
  - Opaque from the inside not from the outside
  - Work with Best feature of HTCondor ever
    - Which is …. ?
- Linux containers (LXC) applicable here
  - But rootly powers required

How far can we get without root?

Boxing without root

- HTCondor Preempt expression

  PREEMPT =

  TARGET.MemoryUsage > threshold

  ProportionalSetSizeKb > threshold

- setrlimit call

  USER_JOB_WRAPPER

  STARTER_RLIMIT_AS

CPU AFFINITY

ASSIGN_CPU_AFFINITY = true

- Pros
  - Works with dynamic slots
  - Works even if not root
  - Any Linux/Windows version
- Cons
  - Glide-ins don’t know which CPU to use
  - Doesn’t allow the job to soak up idle cycles

With root comes…

- PID namespaces
- Named Chroots
- Cgroups
- Mount under scratch

  and/or

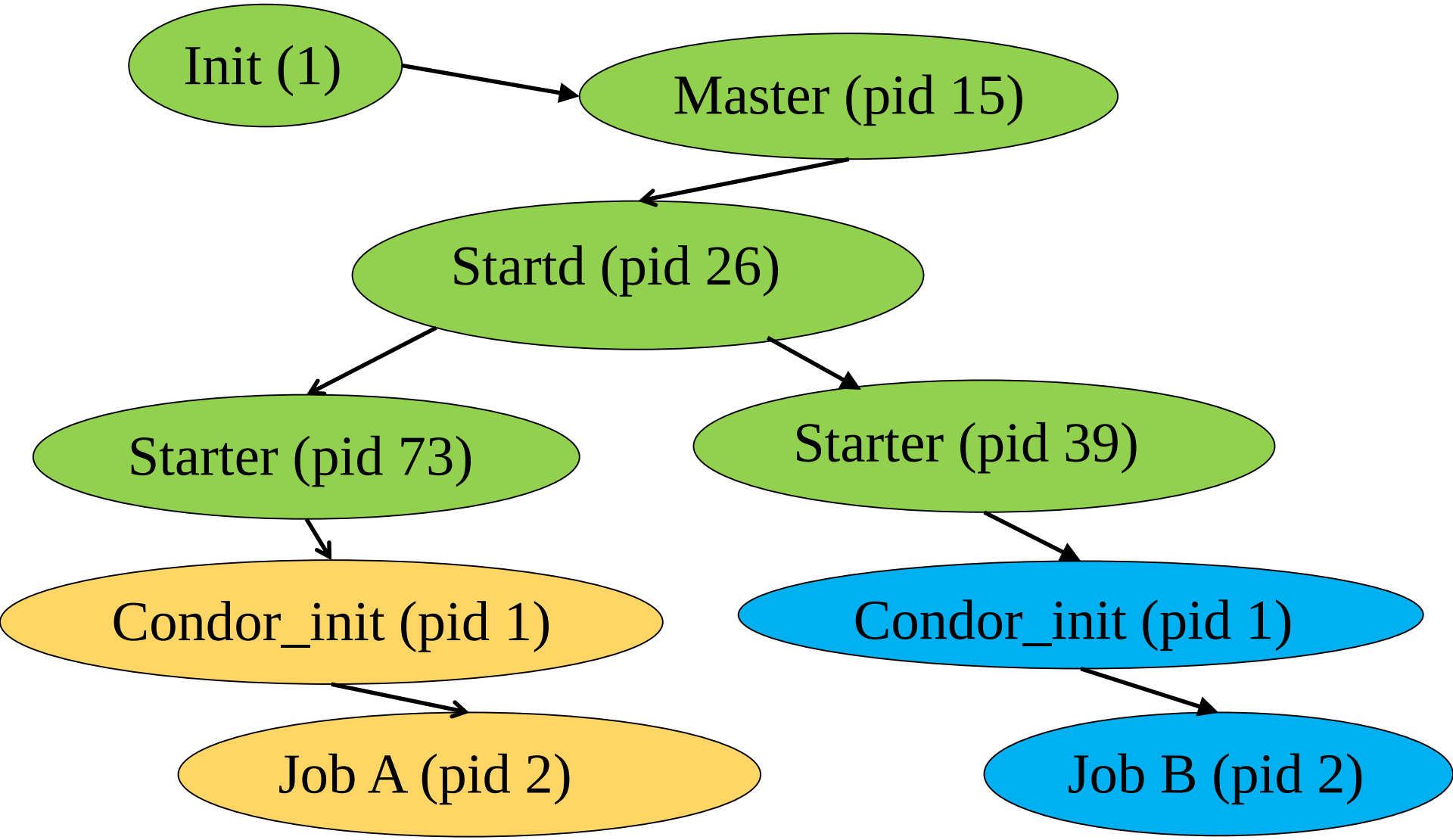
- Docker *

  * Not root exactly, but might as well be…

PID namespaces

- You can't kill what you can't see
- Requirements:
  - HTCondor 8.0+
  - RHEL 6 or later
  - `USE_PID_NAMESPACES = true`
    - (off by default)
  - Must be root

PID Namespaces



Named Chroots

- Isolate the filesystem from the job

- Startd advertises available chroots

```
NAMED_CHROOT = /foo/R1,/foo/R2
```

- Job picks one:

```
+RequestedChroot = "/foo/R1"
```

- Make sure path is secure!

Ancient History: Chroot

- HTCondor used to chroot every job:
  - No job could touch the file system
  - Private files in host machine stayed private

But…

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!

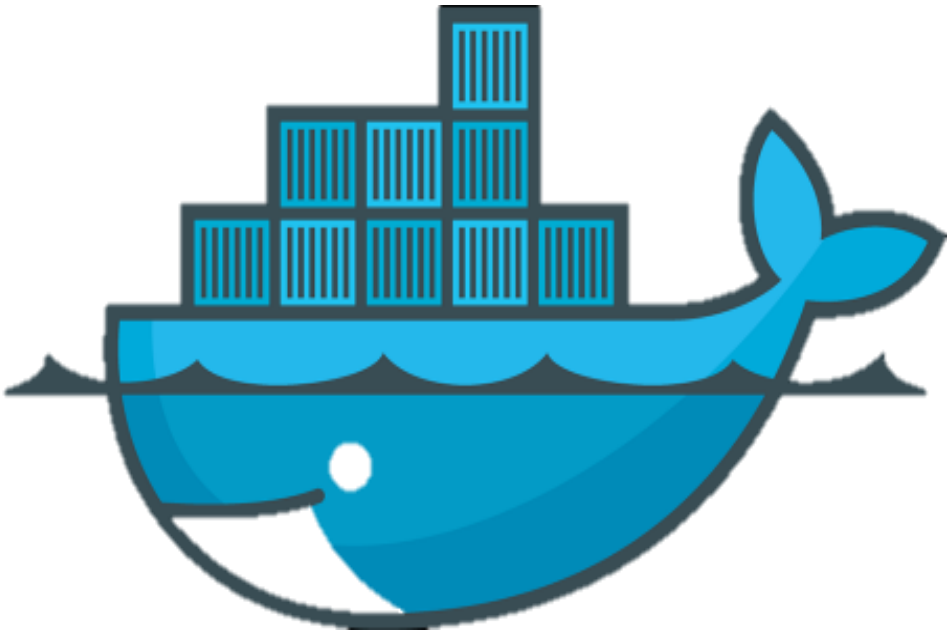
HTCondor no longer chroots all jobs

But you can optionally do so.

Very few site sites do…

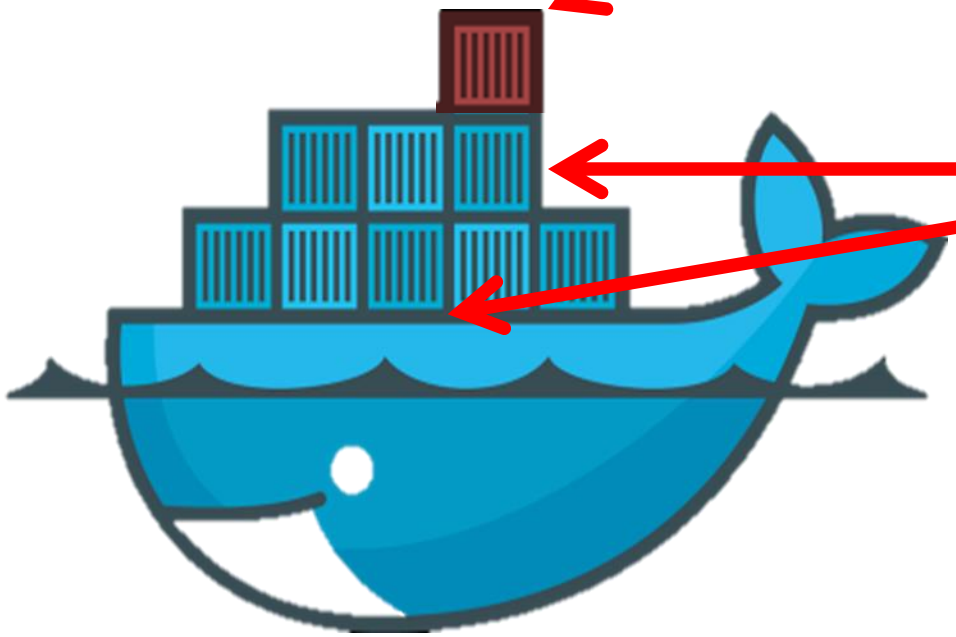
Enter Docker

Docker manages Linux containers.
And gives 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

Docker 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

Docker 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

At the Command Line

```
$ 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
```

Isn't this a Virtual Machine?

- Containers share Linux kernel with host
- Host can "ps" into container
  - One-way mirror, not black box
- Docker containers do not run system daemons
  - CUPS, email, cron, init, fsck, (think about security!)
- Docker images much smaller than VM ones
  - Just a set of files, not a disk image
- Docker provides namespace for images
- Much more likely to be universally available

Docker run two step

`docker run` **`ubuntu`** `cat /etc/deb..`

Every image that docker runs must be local

`Docker run` implies `docker pull`

run can fail if image doesn't exist

or can't be downloaded

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
```

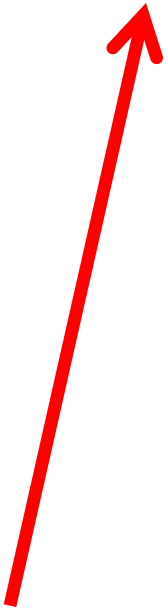
```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
new_ubu             latest              b67902967df7        8 weeks ago         192.7 MB
<none>              <none>              dd58b0ec6b9a        8 weeks ago         192.7 MB
<none>              <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
```

Full Image name

`hub.demo.org:8080/user/`**`image`**`:ver`



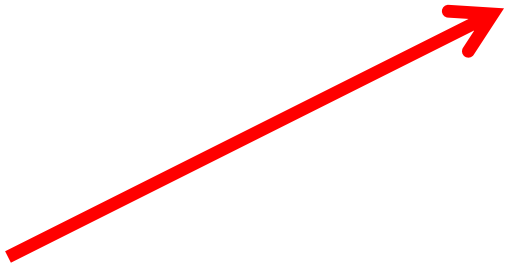


Image name and version: (default: “latest”)

Name of the hub (default docker-io)

Run your own docker hub

- Docker hub is an image!

```
$ docker run docker/docker-registry
```

(and a bunch of setup – google for details)

Any production site will want to run own hub

Or put a caching proxy in front of the public one

HTCondor docker universe

Need condor 8.4+

Need docker (maybe from EPEL)

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

Docker is moving fast: docker 1.8+, ideally

Condor needs to be in the docker group!

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

What? No Knobs?

- condor_starter detects docker by default

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

- If docker is in a non-standard place

  DOCKER = /usr/bin/docker

“Docker” Universe jobs

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

A docker Universe Job Is a Vanilla 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
docker_image =deb7_and_HEP_stack
# executable = /bin/my_executable
```

- Image is the name of the docker image on the execute machine. Docker will pull it
- Executable is from submit machine or image NEVER FROM execute machine!
- Executable is optional

(Images can name a default command)

Docker Universe and File transfer

```
universe = docker
transfer_input_files = <files>
When_to_transfer_output = ON_EXIT
```

- HTCondor volume mounts the scratch dir
  And sets the cwd of job to scratch dir
- RequestDisk applies to scratch dir, not container
- Changes to container are NOT transferred back
- Container destroyed after job exits

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

RequestDisk applies to the scratch dir only

10 Gb limit rest of container

Docker universe “just works”

- Install docker
- Give HTCondor access to docker
- Profit!

But can you effectively box jobs without it?

Control Groups aka “cgroups”

- Two basic kernel abstractions:

  1) nested groups of processes

  2) “controllers” which limit resources

Control Cgroup setup

- Implemented as filesystem
  - Mounted on /sys/fs/cgroup, or /cgroup or …
- User-space tools in flux
  - Systemd
  - Cgservice
- /proc/self/cgroup
- Docker manages cgroups for containers

Cgroup controllers

- Cpu
  - Allows fractional cpu limits
- Memory
  - Need to limit swap also or else…
- Freezer
  - Suspend / Kill groups of processes

Enabling cgroups

- Requires:
  - RHEL6, RHEL7 even better
  - HTCondor 8.0+
  - Rootly condor

  - BASE_CGROUP=htcondor

  - And… cgroup fs mounted…

Cgroups with HTCondor

- Starter puts each job into own cgroup
  - Named exec_dir + job id
- Procd monitors
  - Procd freezes and kills atomically
- MEMORY attr into memory controller
- CGROUP_MEMORY_LIMIT_POLICY
  - Hard or soft
  - Job goes on hold with specific message

MOUNT_UNDER_SCRATCH

- Or, “Shared subtrees”
- Goal: protect /tmp from shared jobs
- Requires
  - Condor 8.0+
  - RHEL 5
  - HTCondor must be running as root
  - MOUNT_UNDER_SCRATCH = /tmp,/var/tmp

MOUNT_UNDER_SCRATCH

```
MOUNT_UNDER_SCRATCH=/tmp,/var/tmp
```

Each job sees private /tmp, /var/tmp

Downsides:

No sharing of files in /tmp

Questions?