CENTER FOR
HIGH THROUGHPUT
COMPUTING

HTCondor

Cgroups, containers and HTCondor, oh my

Center for High Throughput
Computing

Outline

- Why put contain jobs?

- Ersatz HTCondor containment
- Docker containers
- Singularity containers

3 Protections

1) Protect the machine from the job.

2) Protect the job from the machine.

3) Protect one job from another.

The ideal container

- 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

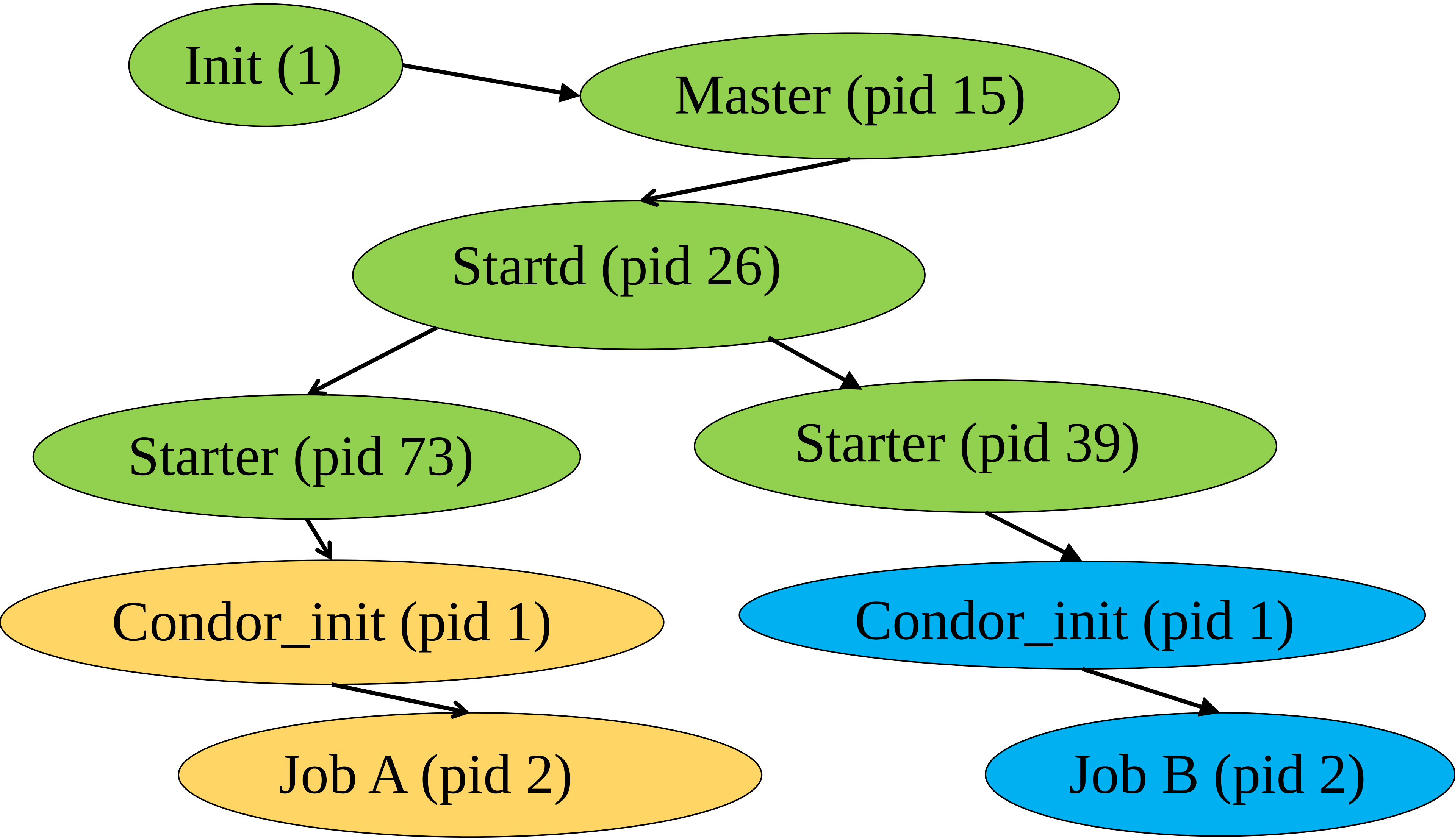
- CPU
- Memory
- Disk
- Network
- Signals
- L1-2-3 cache

HTCondor’s containment

PID namespaces

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

PID Namespaces



Init (1)

Master (pid 15)

Startd (pid 26)

Starter (pid 73)

Starter (pid 39)

Condor_init (pid 1)

Condor_init (pid 1)

Job A (pid 2)

Job B (pid 2)

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

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,
  - Groups are *per controller*
    - E.g. /sys/fs/cgroup/memory/my_group
    - /sys/fs/cgroup/cpu/my_group
  - Interesting contents of virtual groups:
    - /sys/fs/cgroup/memory/my_group/tasks
  - Condor default is
    - /sys/fs/cgroup/<controller>/htcondor
  - Compare with systemd's slices

Cgroup controllers

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

This is the slide where someone asks about the blkio controller

Enabling cgroups

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

  - And… condor_master takes care of the rest

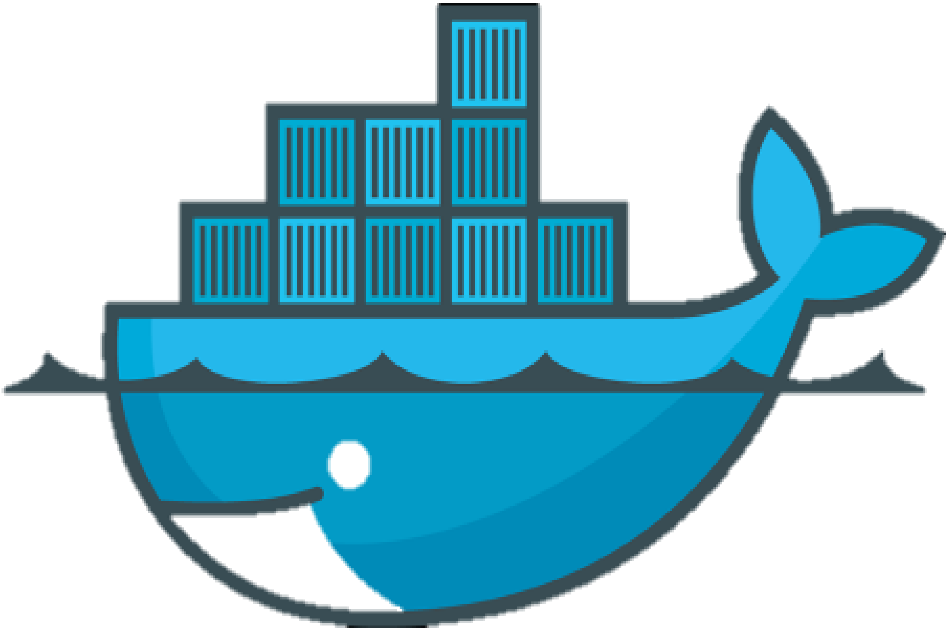
Cgroups with HTCondor

- Starter puts each job into own cgroup
  - Named exec_dir + job id
- Procd monitors
  - Procd freezes and kills atomically
- CPUS attr * 100 > cpu.shares
- MEMORY attr into memory controller
- CGROUP_MEMORY_LIMIT_POLICY
  - Hard or soft
  - Job goes on hold with specific message

Cgroups seem fiddly, why not let something else do it?

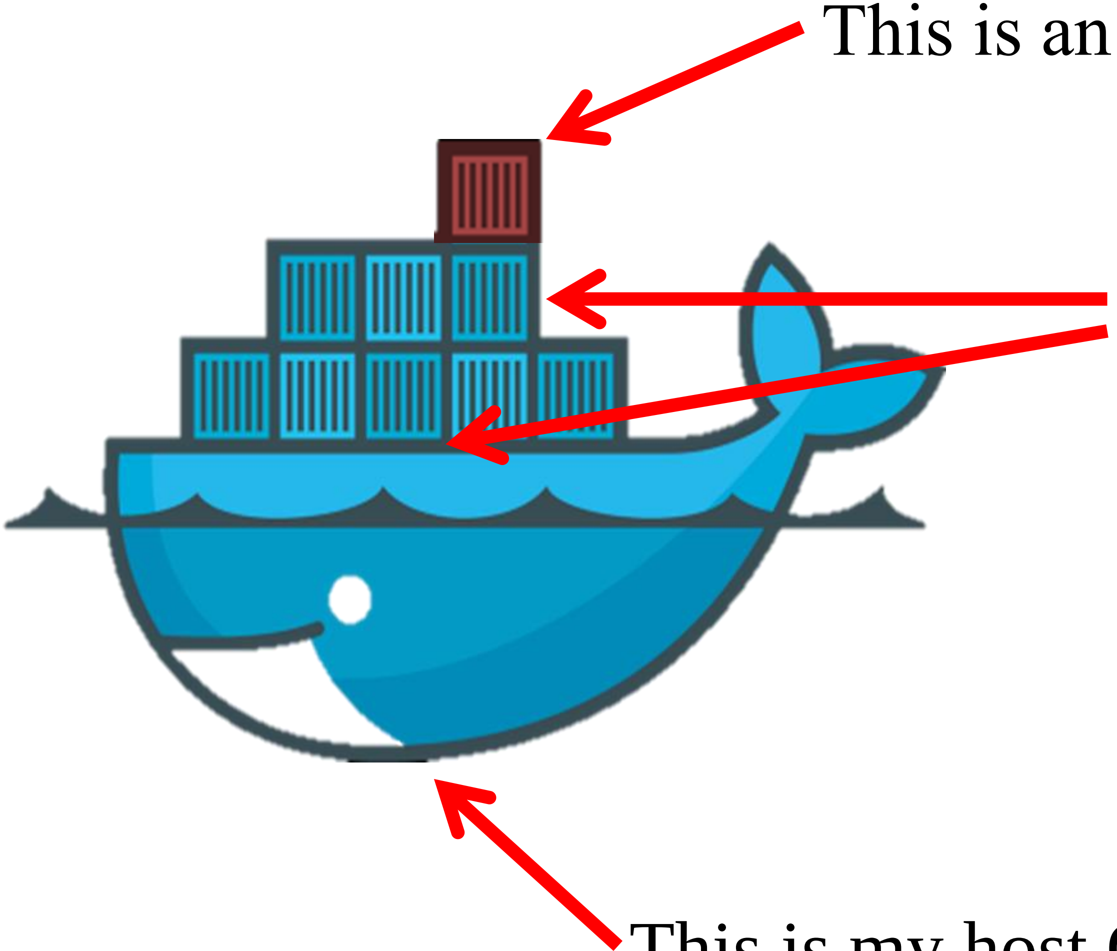
Enter Docker

Docker manages Linux containers via cgroups.
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

HTCondor docker universe

Need condor 8.4+

Need docker (maybe from EPEL)

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

Condor needs to be in the docker group!

```
$ useradd -G docker condor
```

Docker be running:

```
$ 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
```

We had to have some knobs

- DOCKER_DROP_ALL_CAPABILITIES
  - Evaluated with job and machine
  - Defaults to true
  - If false, removes –drop-all-cap from docker run
- DOCKER_VOLUMES = CVMFS, SCR
- DOCKER_VOLUME_DIR_CVMFS = /cvmfs
- DOCKER_MOUNT_VOLUMES = CVMFS

“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

Enter Singularity

- Singularity like light Docker:
  - No daemon
  - Setuid wrapper binary
  - Can work without hub
  - Can work with setuid (soon)

Enabling Singularity for all jobs

- SINGULARITY = /usr/bin/singularity
- SINGULARITY_JOB = true
- SINGULARITY_IMAGE_EXPR =
  “/full/path/to/image”

…for some jobs

```
SINGULARITY_JOB = \
!isUndefined(TARGET.SingularityImage)

SINGULARITY_IMAGE_EXPR = \
    TARGET.SingularityImage
```

Singularity vs Docker

- Designed not as user focused, rather admin
- Jobs may not know when in singularity
- Startd focused

Questions?

Thank you!