

Container Universe

Greg Thain

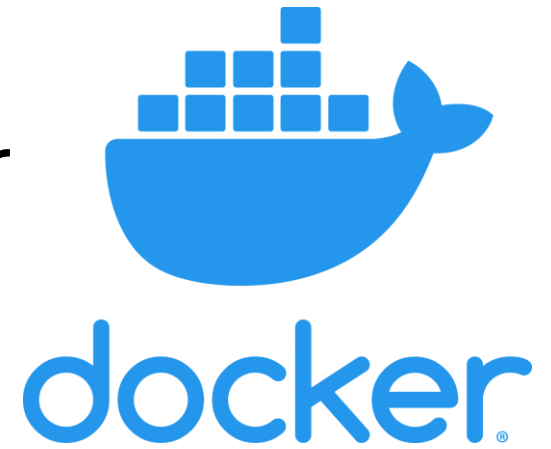
Center for High Throughput Computing
University of Wisconsin - Madison

Overview

- Quick definition of container world
- Existing ways
 - Docker Universe
 - Singularity support
- Differences between docker and singularity
- Solution: Container universe
- Future work



In the beginning, there was Docker

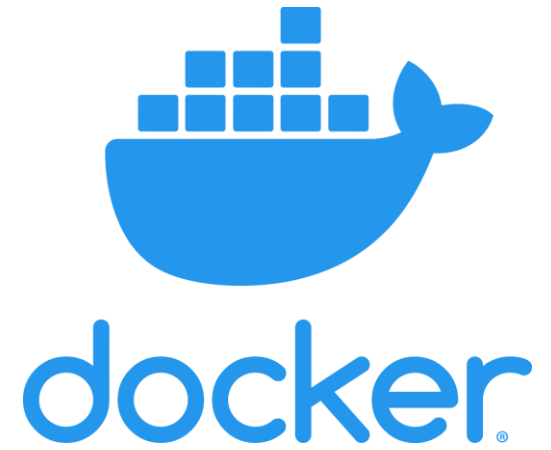


Docker runs processes ...

1. In a chroot like virtualized filesystem
 1. (with escapes to the host filesystem)
2. Where the filesystem is fetched from a remote hub, then locally cached, in a "stacked image" format
3. In cgroups that protect memory and CPU usage
4. Where the process runs as uid 0 by default
 1. Using Linux capabilities to restrict most damage
5. Using the host's Linux kernel



A Mixed bag for HTC:



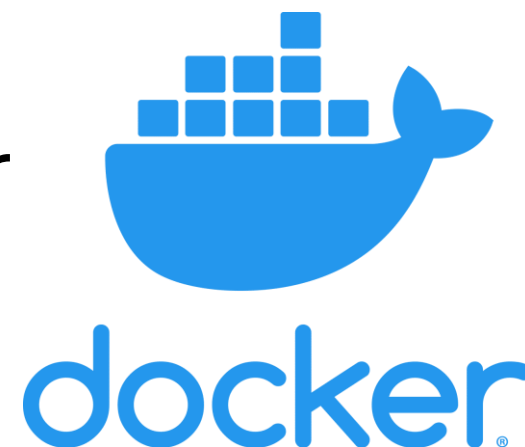
Docker runs processes ...

1. **In a chroot like virtualized filesystem**
 1. (with escapes to the host filesystem)
2. Where the filesystem is fetched from a remote hub, then locally cached, in a "stacked image" format
3. **In cgroups that protect memory and CPU usage**
4. Where the process runs as uid 0 by default
 1. Using Linux capabilities to restrict most damage

These we like!



In the beginning, there was Docker



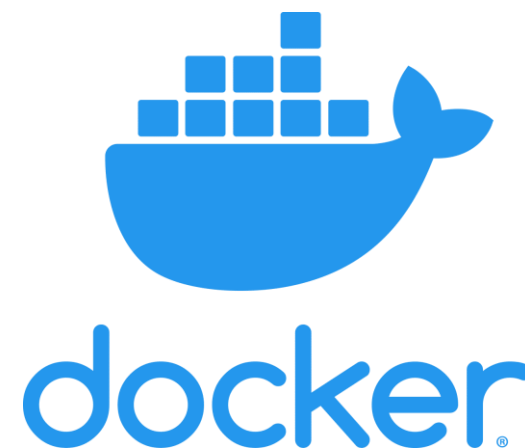
Docker runs processes ...

1. In a chroot like virtualized filesystem
 1. (with escapes to the host filesystem)
2. **Where the filesystem is fetched from a remote hub, then locally cached, in a "stacked image" format**
(and we are subject to network blocking if we pull too much...)
3. In cgroups that protect memory and CPU usage
4. **Where the process runs as uid 0 by default**
 1. **Using Linux capabilities to restrict most damage**

These we Don't Like!



Solution: Docker Universe



- Unrestricted Docker allows root access to host
- HTCondor's ***Docker Universe*** limits docker options
 - To a "safe" subset
 - Translated from condor submit language
 - With the usual scratch directory mounted into the container from host
- Allows the user to run an arbitrary container



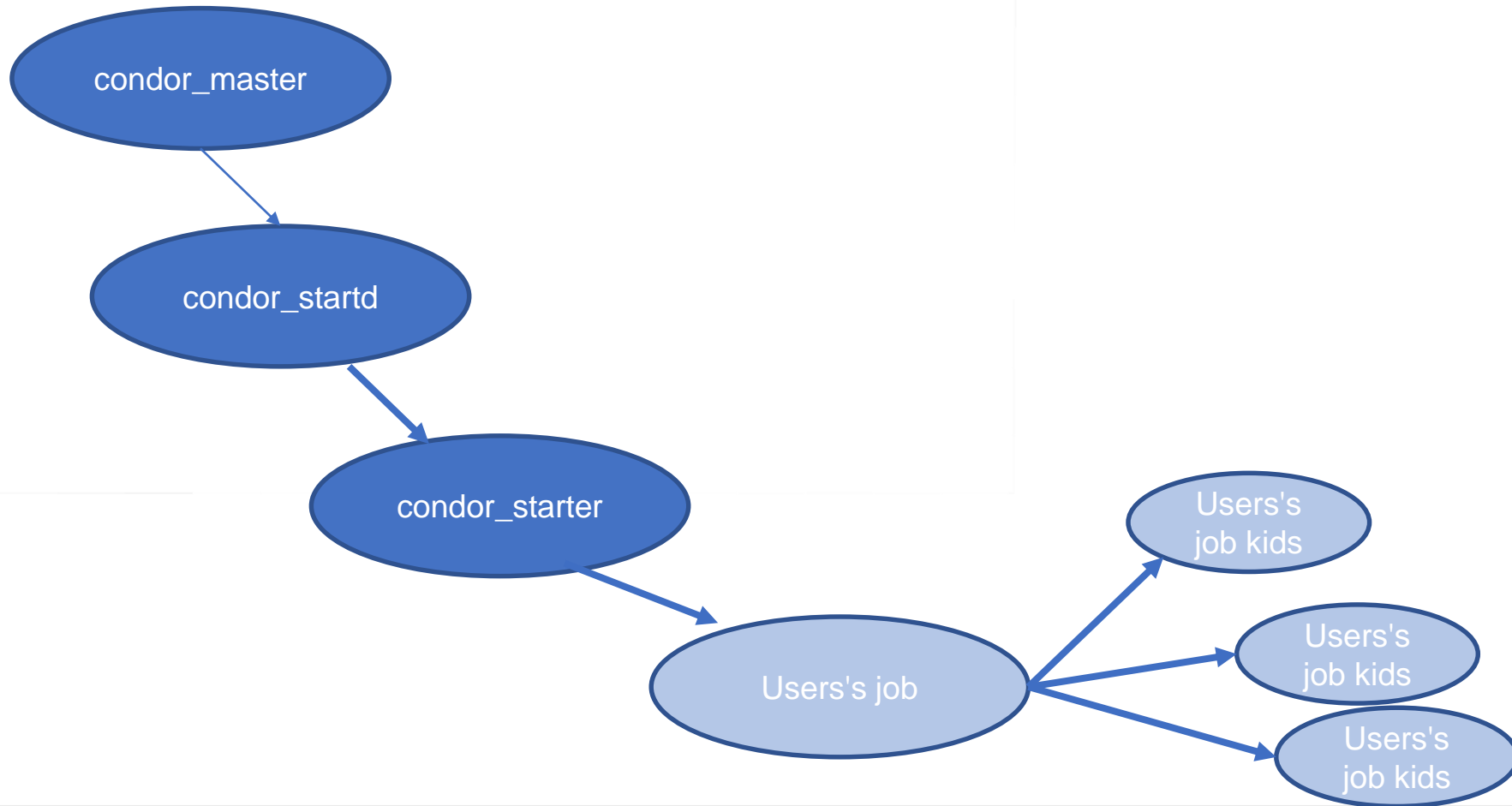
Example submit file

```
universe = docker
docker_image = ubuntu:22
# Note that executable is optional
executable = run_me.sh
arguments = one two three
```

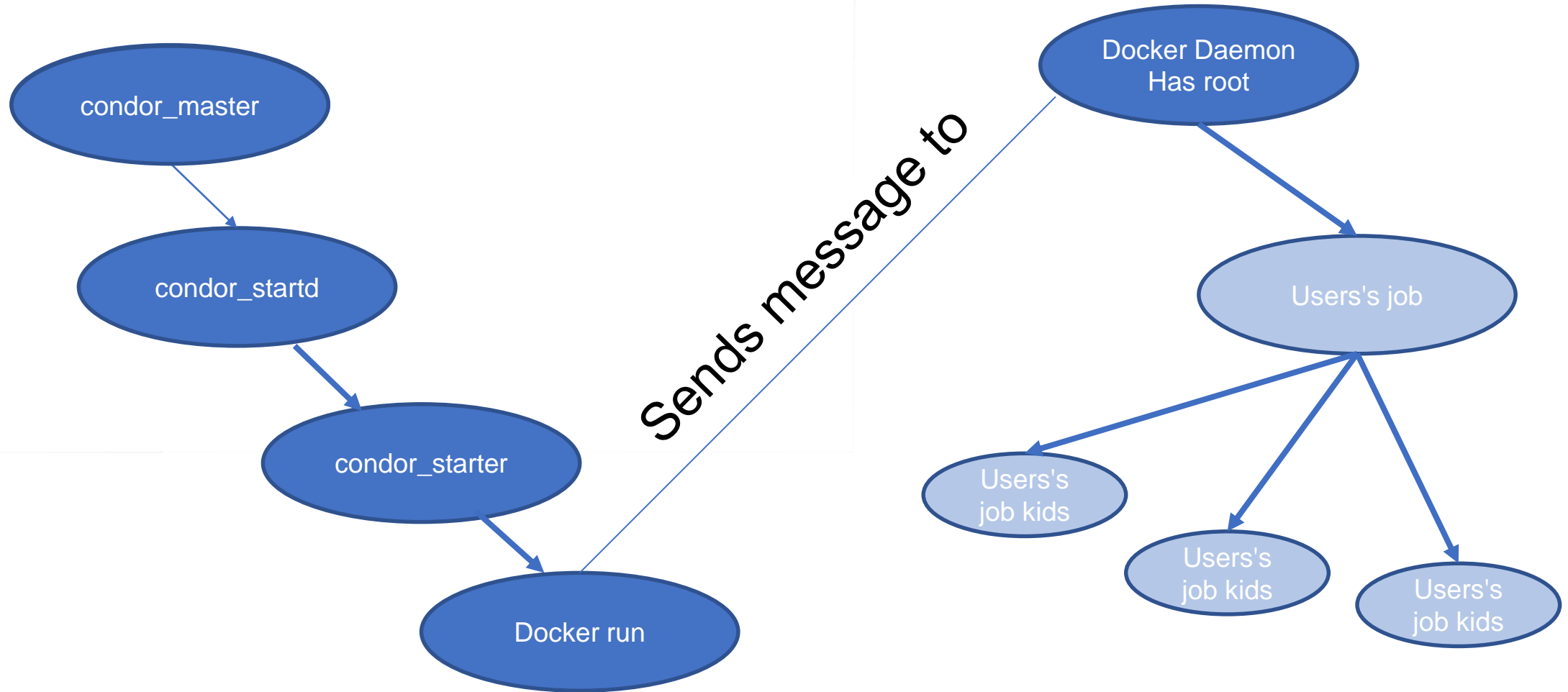
```
Output = output
queue
```



EP Process hierarchy in vanilla universe

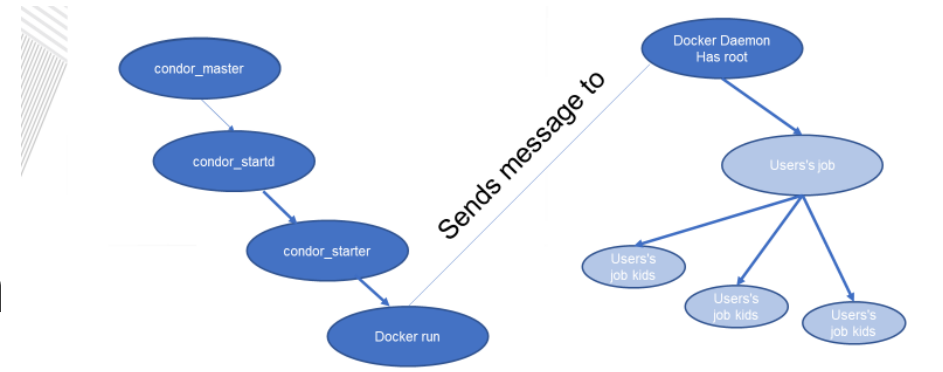


EP Process hierarchy in docker universe



Problems with Docker Uni

- Requires systemd-startd docker daemon
- With root
 - (difficult for glideins)
- User job not in same Unix process tree as starter
 - Ssh_to_job headaches
 - HTCondor doesn't see or control cgroups
 - Resources usage measurement works differently
- We can't use condor file xfer to send image
 - Image must come from local cache



Enter ... Singularity

Mainly added to HTCondor to support glideins

Singularity:

- Needs setuid, but not daemon (changes happening now...)
- Can run from image in a single file
- Transferred by HTCondor
- Doesn't contain with cgroups
- No network isolation



Singularity support in HTCondor is different

Glidein folks wanted the EP Admin to be in charge
not the user

Singularity support in condor is a EP-side KNOB, e.g.

```
SINGULARITY_JOB = true  
SINGULARITY_IMAGE_EXPR = "/path/to/image"
```

Admin can wire expr to allow user to opt it, but site-specific

```
SINGULARITY_IMAGE_EXPR = Target.SingularityImage
```



Example singularity submit file

```
universe = vanilla
+SingularityImage = /path/to/ubuntu_22
executable = run_me.sh
arguments = one two three
```

```
Output = output
queue
```





**SUBMIT +ATTRIBUTES
ARE CONFUSING FOR USERS**



**SO, WE'LL MAKE COMMON
ONES FIRST CLASS, RIGHT?**



Problem

- Most users don't care which container runtime they use
- "Docker" is kind of generic name
- But they do want to have the same submit file for glidein and local



And then things got even worse...

- Singularity project forked into Singularity and Apptainer
- Red Hat reimplemented most of docker as "podman"
 - Without a daemon, so we kind of like this..
- Others wrote other container systems (Charlie cloud, etc.)
- So what to do?



XKCD 127

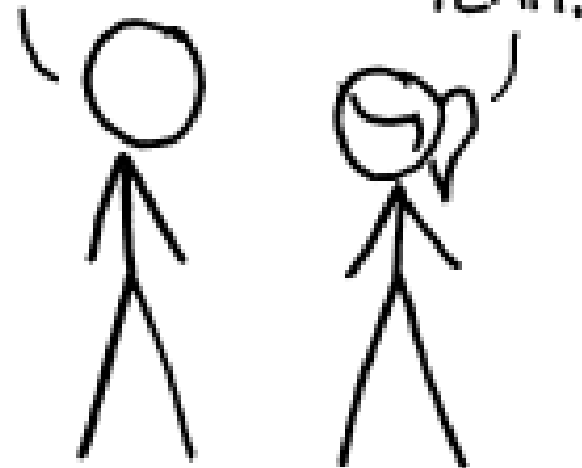


HOW STANDARDS PROLIFERATE:

(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.



Enter ... container universe

- The startd detects if singularity or docker work
 - By running test jobs
- And advertises attributes about those runtimes
 - HasSIF
 - HasDocker
 - HasSandbox



Users just ask for a container and image

```
universe = container
container_image = /path/to/ubuntu_22
executable = run_me.sh
arguments = one two three
```

```
Output = output
queue
```



And matchmaking does the rest

- With new startd-side job transforms
 - That can mutate a vanilla job into a container job
- Note docker universe and old singularity support will work
 - For foreseeable future



Assume container is on EP

```
universe = container
container_image = /path/to/ubuntu_22.sif
Should_transfer_image = false
executable = run_me.sh
arguments = one two three
```

```
Output = output
queue
```



But under the hood, same mechanisms

- Just like saying Universe = docker when docker matches
- Or SingularityJob = true when singularity matches



A container universe job is still a job...

- Has a job log
- Works with DAGMan
- Condor_ssh_to_job (mostly) works
- Condor_tail works
- User level checkpointing works..



Future work

- Add more container runtimes
 - Podman
 - Unprivileged docker
- If job could run on multiple runtimes, who chooses?
- Add automatic checkpoint / restore
 - Creates ~~problems~~ opportunities for checkpoint storage and migration



Thank you and questions

Thank you – Questions?

This work is supported by the NSF under Cooperative Agreement OAC-2030508. Any options, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

