

Containers are good for more than serving cat pictures!?

Lightweight HPC containers with Charliecloud

Reid Priedhorsky

Tim Randles

Michael Jennings

*Los Alamos National Laboratory
High Performance Computing Division*

SEA Containers in HPC Symposium
April 4, 2018



1. Scientific software now needs more flexibility than HPC centers offer.
2. Containers are the best way to fix this, but Docker is too hard to deploy at scale in production.
3. A lightweight hybrid approach like Charliecloud is the best balance between functionality, security, and cost.
4. Demonstration



Some people need different software

Standard HPC software stacks are good for a specific purpose

- specifically: MPI-based simulation of physical systems

What if your thing is different?

- non-MPI simulations
- data analytics and machine learning
- epic build process
- cool kids use Ubuntu/Arch/Alpine (not RHEL)

Admins will install software for you.

- BUT only if there's enough demand
- unusual needs go unmet
- are you crackpot or innovative?



BYOS (bring your own software)

- Let users install software of their choice
- ... up to and including a complete Linux distribution
- ... and run this *image* on compute resources they don't own.



Advantages

- software dependencies: numerous, unusual, older, newer, internet ...
- portability of environments: e.g., across dev/test/small/large ...
- consistent environments: validated, standardized, archival ...
- usability

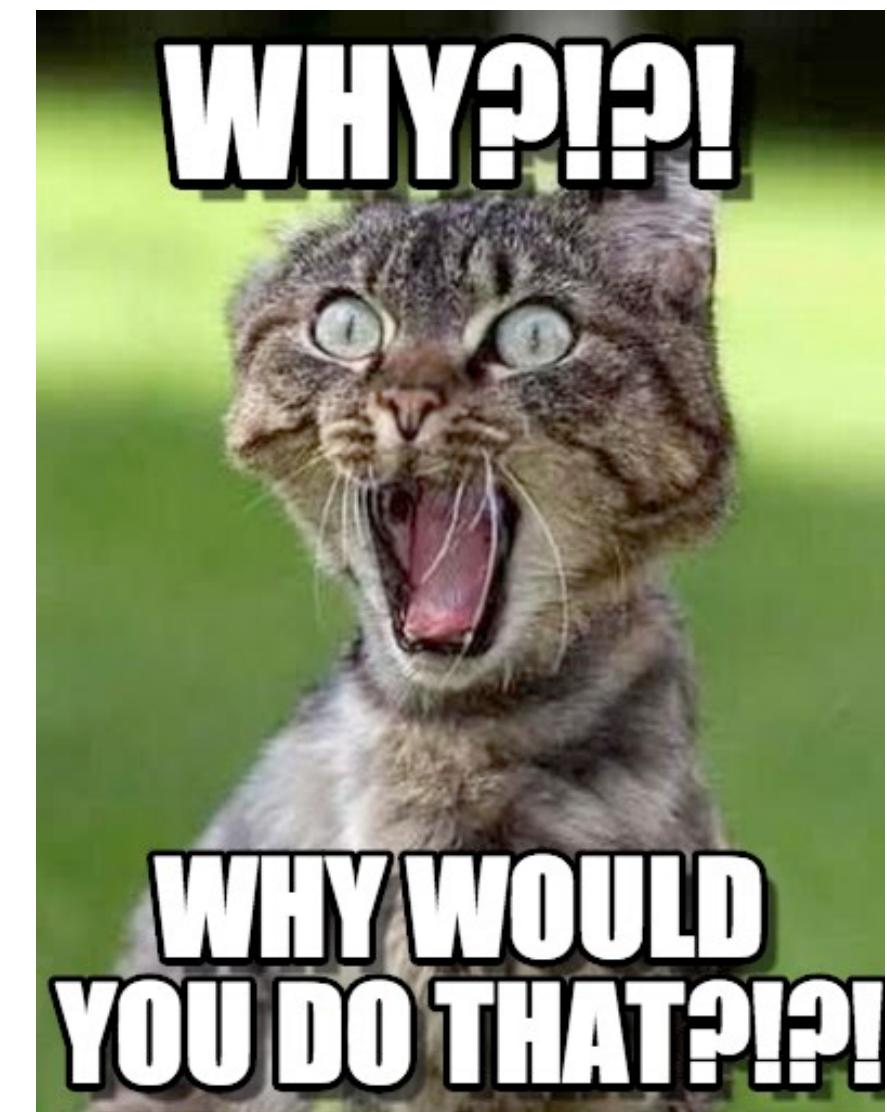
Disadvantages (what we are trying to avoid)

- missing functionality: high-speed network, accelerators, file systems
- performance: many opportunities for overhead
- i.e., you paid a lot of money for that supercomputer, best use it



Design goals

1. Standard, reproducible workflow
2. Work well on existing resources
3. Be very simple



Design goals

- 1. Standard, reproducible workflow
 - in contrast with “tinker ’til it’s ready, then freeze”
 - standard ⇒ reduce training/devel costs, increase skill portability
 - reproducible ⇒ creation of images is easier & more robust
- 2. Work well on existing resources
 - HPC centers are very good at what they do
 - let’s not re-implement and re-optimize
 - resource management: solved (Slurm, Moab, Torque, PBS, etc.)
 - file systems: solved (Lustre, Panasas, GPFS)
 - high-speed interconnect: solved (InfiniBand, OPA)
- 3. Be very simple
 - save costs: development, debugging, security, usability, ...
 - UNIX philosophy: “make each program do one thing well”

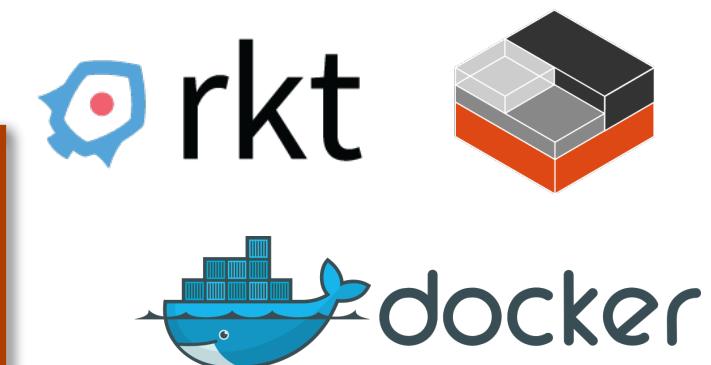


option	definition	UDSS shares with host...			pros	cons
		kernel	core libraries	app libraries		
compile it yourself	download all your dependencies and compile them	yes	yes	mixed	always available; in principle, can do anything	not 1995 anymore; in practice, too hard
virtual machines	program (software) that emulates a computer (hardware)	no	no	no	maximum flexibility and isolation	too heavyweight; HPC is not cloud
containers	isolate UDSS using kernel mechanisms	yes	no	no	easy to manage; good performance; <i>sufficient</i> flexibility and isolation	new



Full-featured

- image building
- image management
 - storage, caching, tagging, signing
- orchestration
- storage management
- runtime setup
 - e.g., default command/script, `inetd`-alike
- stateful containers
- supervisor daemon(s)



Lightweight

- few features
- given an image, run it

`unshare(1)`
`systemd-nspawn` ±
`NsJail` ±
`CCon` ±



Disadvantages ...

1. complexity
2. support burden
3. privileged & trusted operations

Lower-cost deployment



Claim: Lightweight implementations
are a better choice for HPC centers

- most important cloud-like flexibility
- don't compromise existing tools & strengths of HPC centers

1. Linux namespaces

- **mount**: filesystem tree and mounts
- **PID**: process IDs
- **UTS**: host name & domain name
- **network**: all other network stuff
- **IPC**: System V and POSIX
- **user**: UID/GID/capabilities

System calls: `unshare(2)`, `clone(2)`, `setns(2)`

– **privileged**
need root to create, unless you add ...

– **unprivileged**

2. cgroups

- limit resource consumption per process

3. `prctl(PR_SET_NO_NEW_PRIVS)`

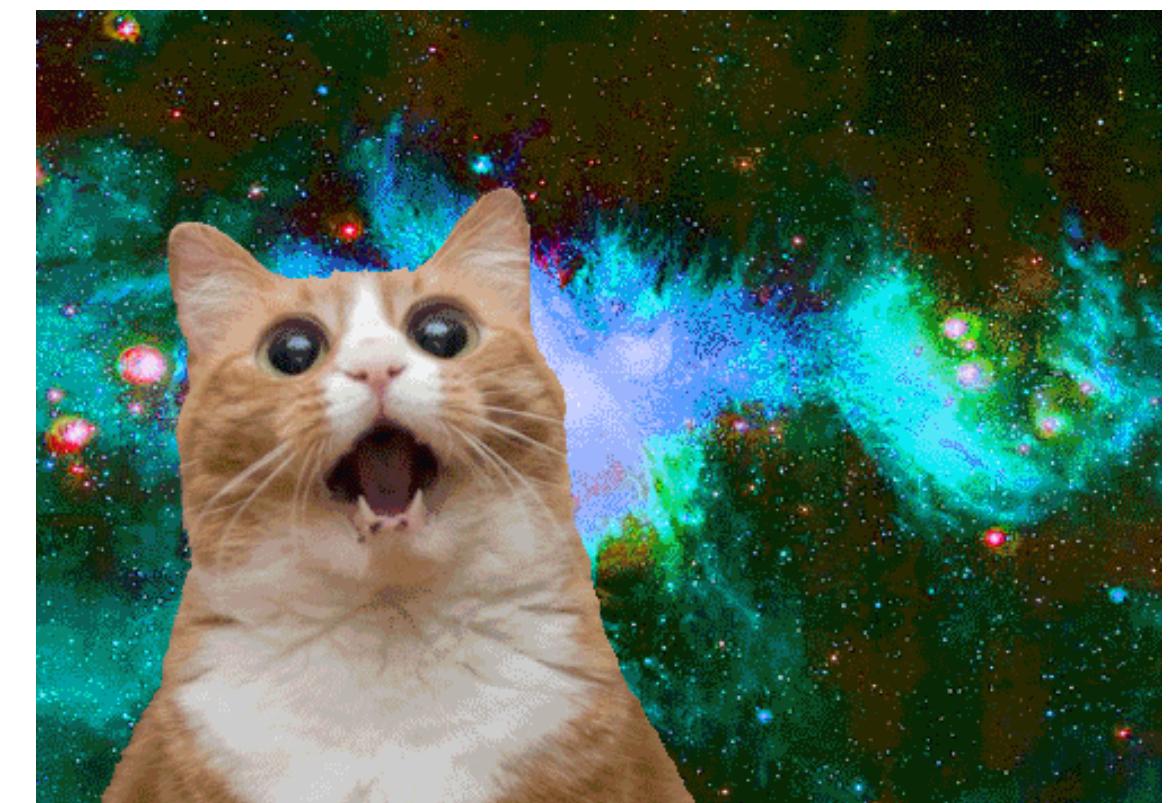
- prevent `execve(2)` from increasing privileges

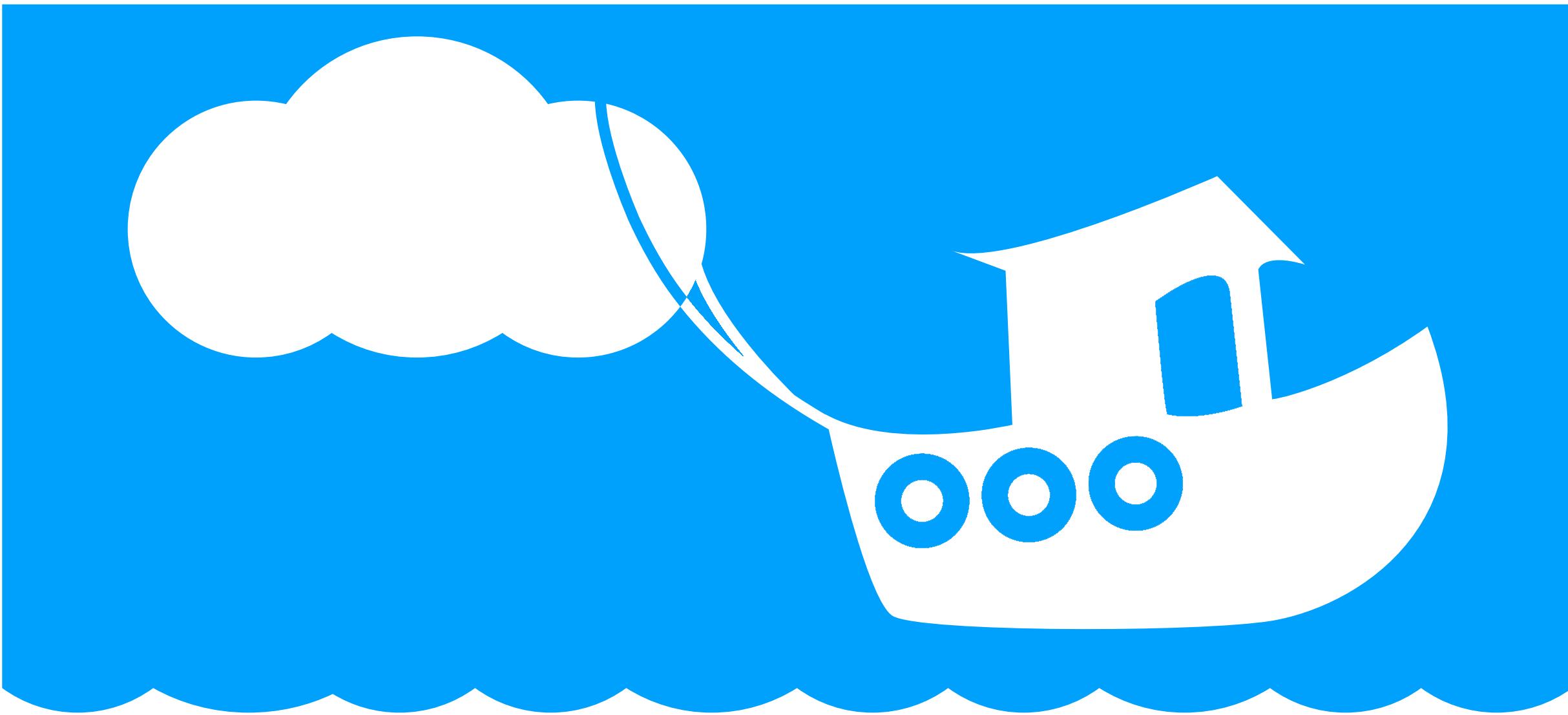
4. `seccomp(2)`

- filter system calls

5. SELinux, AppArmor, etc.

- various features that change what a process may do





charliecloud

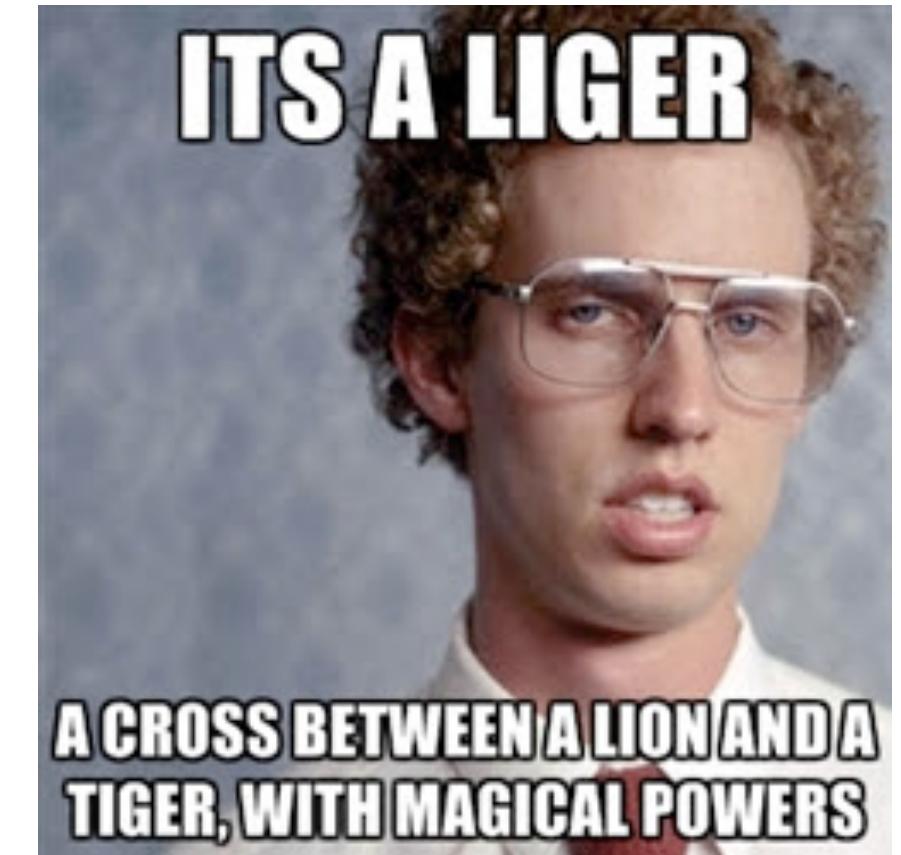
SMALL SOFTWARE DELIVERING BIG FLEXIBILITY TO HPC

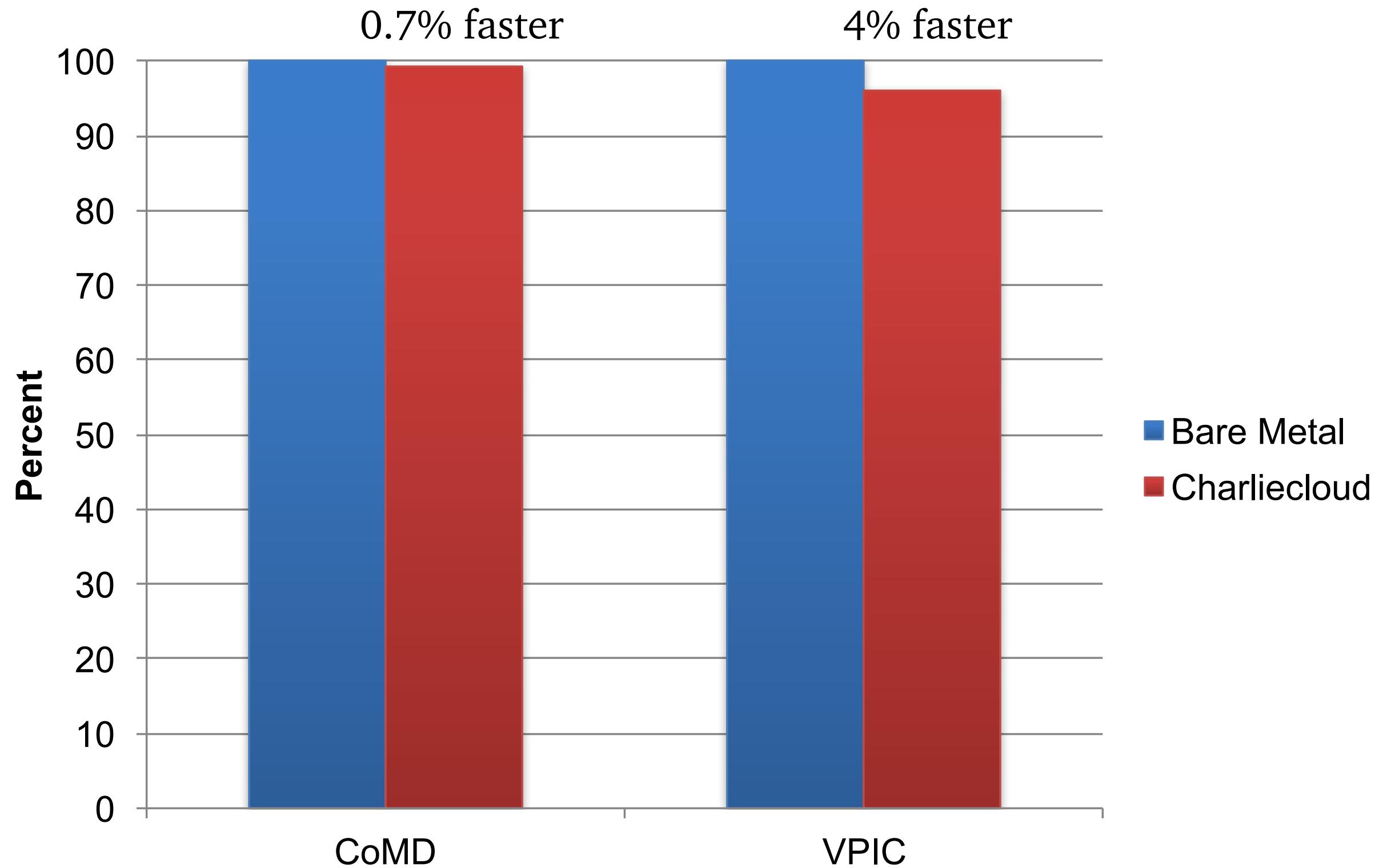
1. Image building & sharing goes in a sandbox

- safe place for users to be root: user workstation or virtual machine
- wrap Docker for image building
 - or anything else that can produce a filesystem tree
debootstrap(8), yum --installroot, etc.

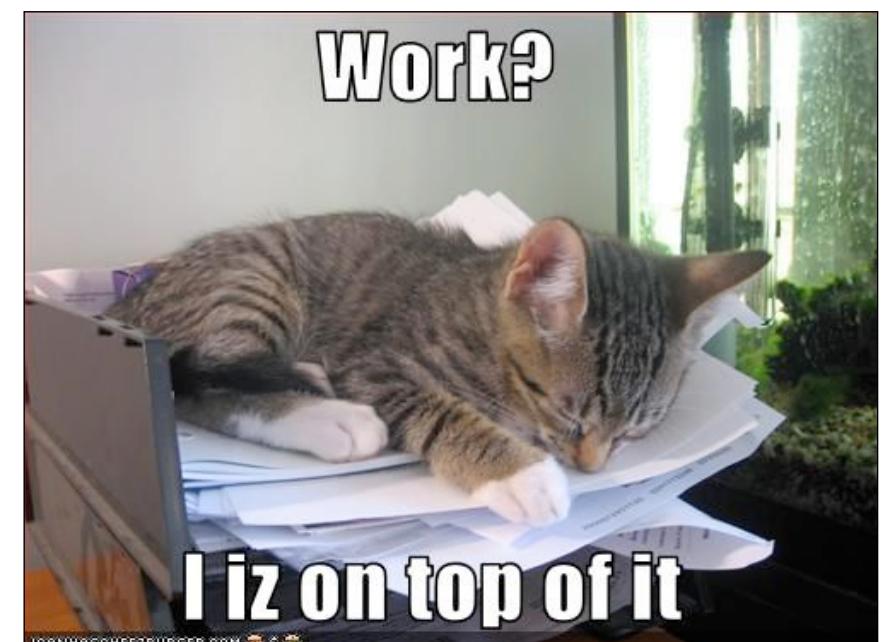
2. Run images with our own unprivileged runtime

- mount & user namespaces only
 - requires new-ish kernel (or setuid test mode)
 - most distros have the right kernel
 - Cray UP04 has it
 - RHEL/CentOS 7 can install via ElRepo
- **it's a user program!!!**
- admins don't need to do (or know) anything





step	where		privileged?
	sandbox	prod.	
1. Build Docker/etc. image	✓		maybe
2. Dump image to tarball	✓		maybe
3. Copy tarball to where you want to run	✓	✓	no
4. Unpack tarball		✓	no
5. Configure your stuff (sometimes)		✓	no
6. Run your commands in container		✓	no



LET'S DO

A DEMO

memegenerator.net

All container implementations expose traditionally privileged operations to unprivileged users.

What should keep this safe?

- Novel security boundary that inevitably has bugs?
- Well-understood user-land approaches still based on setuid-root?
- Let the kernel handle it?

You already trust the Linux kernel.

- *Charliecloud leaves security to the kernel.*
- Kernel holds the most access control information.
- Lots of ongoing general hardening work in the kernel.
- Huge kernel community means vulnerabilities have a harder time hiding and are addressed quickly when exposed.



Why Docker? (160k lines of code)

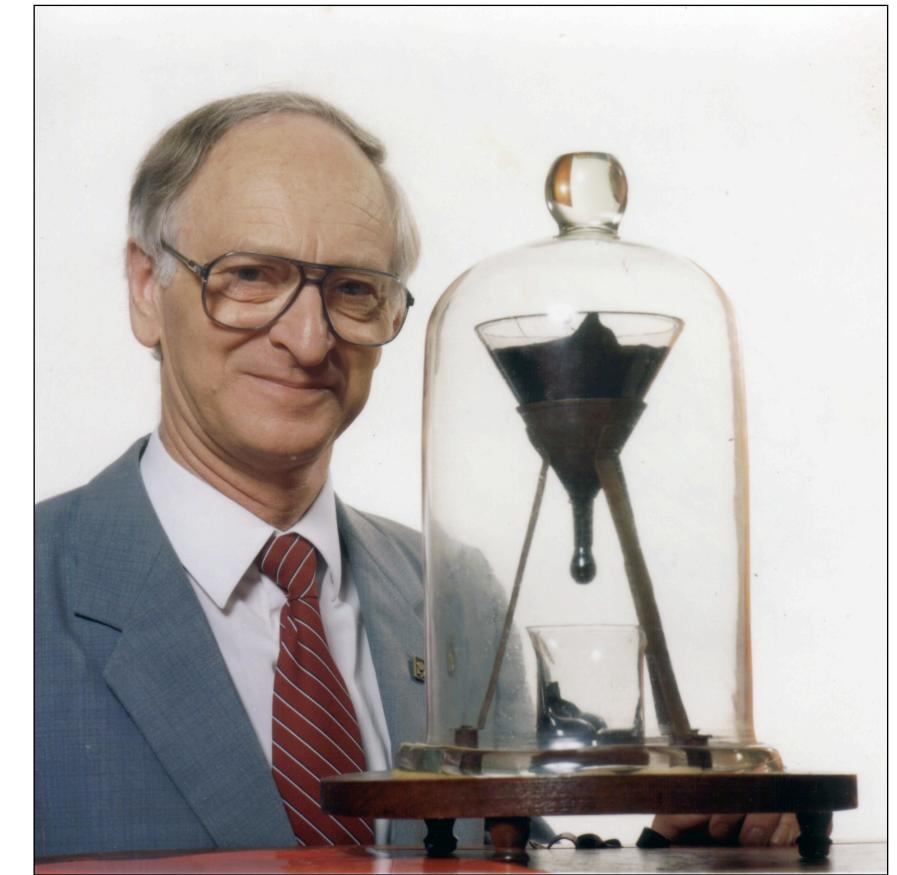
- It's the industry standard, but it's a big system with constraints that make deployment at scale on production HPC resources tricky.
 - e.g., standard authentication is root-or-nothing
 - e.g., HPC nodes typically don't have local storage; where does the Docker cache go?
 - e.g., what happens when 10,000 nodes start pulling layers via HTTPS?

Why Singularity? (15k LOC)

- Aims to re-invent Docker for HPC
 - ... but with a much smaller community.

Why Shifter? (19k LOC)

- Great user experience (“run job X in Docker container Y ”).
- Significant admin cost to set this up and support it.



John Mainstone / University of Queensland

Why Charliecloud? (1k LOC)

- User simplicity + admin simplicity + no new security boundary

1. Available now (version 0.2.3; 0.2.4 in preparation)
 - newer kernel needed (roughly 4.4+), or setuid tolerance
 - works on cloud VMs too
2. Installed now on several LANL clusters
3. Becoming available in Linux distributions
 - Debian, Gentoo
 - openSUSE Build Service
 - submitted to OpenHPC
 - ...
4. Instructions for pre-installed VirtualBox image
 - no root needed
 - Mac, Windows, Linux, Solaris



100
Continue

Containers are good for more than serving cat pictures!?

Lightweight HPC containers with Charliecloud

Reid Priedhorsky, Tim Randles, Michael Jennings

{reidpr, trandles, mej}@lanl.gov

:login: article (USENIX magazine)

- “Linux containers for fun and profit in HPC”
- <https://usenix.org/publications/login/fall2017/priedhorsky>

Supercomputing 2017

- “Charliecloud: Unprivileged containers for UDSS in HPC”
- <https://dl.acm.org/citation.cfm?id=3126925>

Documentation

- included detailed tutorials
- <https://hpc.github.io/charliecloud>

Source code

- <https://github.com/hpc/charliecloud>



photo: Reid Priedhorsky

1. Standard, reproducible workflow

- build with Docker, the industry standard for reproducible builds
- ... or anything else you want to script up

2. Work well on existing resources

- ch-run: minimal but sufficient isolation (mount & user namespaces)
- performance unchanged, direct access to everything
- scales using standard HPC tools

3. Be very simple

- 5 shell scripts, 2 C programs, 1000 lines of code

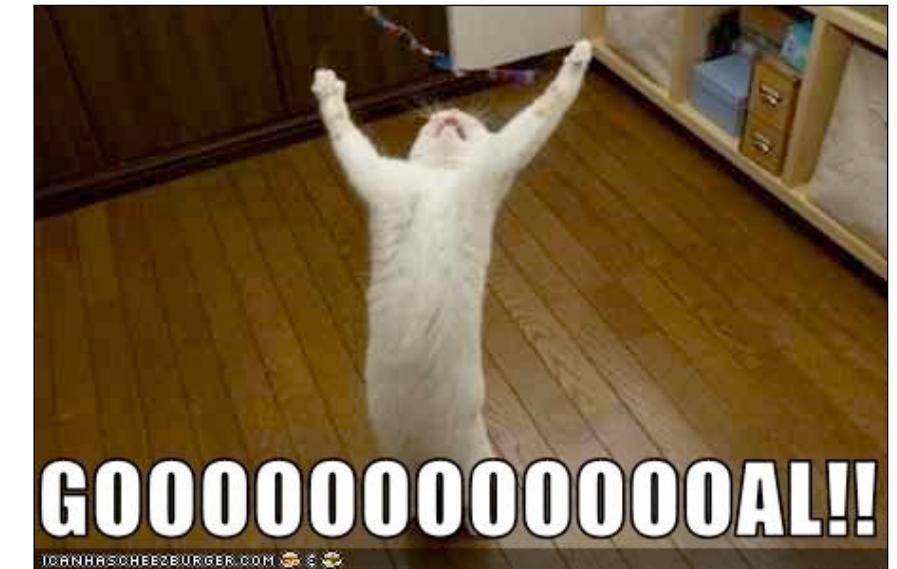
- for comparison ...

NsJail: 4,000 lines

Singularity: 14,500

Shifter: 19,000

Docker: 160,000



demo backup slides

```
snbx$ cd ~/charliecloud/examples/other/spark
```

privileged

```
snbx$ ls
```

```
Dockerfile slurm.sh test.bats
```

```
snbx$ ch-build -t spark ~/charliecloud
```

```
[sudo] password for reidpr:
```

```
Sending build context to Docker daemon 19.39MB
```

```
Step 1/10 : FROM debian:stretch
```

```
---> 2b98c9851a37
```

```
[...]
```

```
Step 10/10 : RUN mv /spark/conf /mnt/0 && ln -s /mnt/0 /  
spark/conf
```

```
---> Using cache
```

```
---> f5f17aa3a634
```

```
Successfully built f5f17aa3a634
```

```
Successfully tagged spark:latest
```



```
snedb$ ch-docker2tar spark /var/tmp                               privileged
289M /var/tmp/spark.tar.gz

snedb$ tar tvf /var/tmp/spark.tar.gz | head
-rwxr-xr-x 0/0          0 2018-03-26 14:31 .dockerenv
drwxr-xr-x 0/0          0 2018-03-15 11:56 bin/
-rwxr-xr-x 0/0 1099016 2017-05-15 13:45 bin/bash
-rwxr-xr-x 0/0 35448 2017-01-29 11:30 bin/bunzip2
hrwxr-xr-x 0/0          0 2017-01-29 11:30 bin/bzcat link to bin/bunzip2
lrwxrwxrwx 0/0          0 2017-01-29 11:30 bin/bzcmp -> bzdiff
-rwxr-xr-x 0/0          2140 2017-01-29 11:30 bin/bzdiff
lrwxrwxrwx 0/0          0 2017-01-29 11:30 bin/bzegrep -> bzgrep
-rwxr-xr-x 0/0          4877 2017-01-29 11:30 bin/bzexe
lrwxrwxrwx 0/0          0 2017-01-29 11:30 bin/bzfgrep -> bzgrep
```

unprivileged

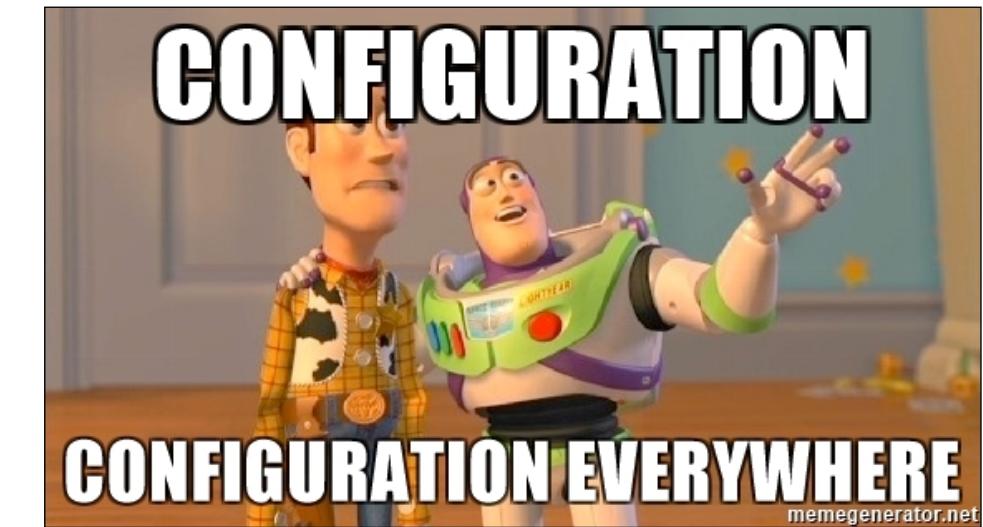
```
snbx$ scp /var/tmp/spark.tar.gz tfta:/users/reidpr  
spark.tar.gz                                100% 289MB 96.2MB/s 00:03  
snbx$ ssh wc-fe  
wc-fe$ ls -lh spark.tar.gz  
-rw-r----- 1 reidpr reidpr 289M Mar 26 14:37 spark.tar.gz
```



```
wc-fe$ salloc -N32                                         unprivileged
wc003$ module load openmpi
wc003$ module load charliecloud
wc003$ srun ch-tar2dir ~/spark.tar.gz /var/tmp
creating new image /var/tmp/spark [32 times]
/var/tmp/spark unpacked ok [32 times]
wc003$ ls /var/tmp/spark
WEIRD_AL_YANKOVIC  boot  etc   lib    media   opt    root   sbin   srv    tmp    var
bin                 dev    home  lib64  mnt    proc   run    spark  sys    usr
wc003$ du -sh /var/tmp/spark
500M /var/tmp/mpihello
wc003$ ls -R /var/tmp/mpihello | wc -l
14784
```



```
wc003$ MASTER_IP=$( ip -o -f inet addr show dev $DEV \
| sed -r 's/^.*inet ([0-9.]+).*/\1/' )
wc003$ MASTER_URL=spark://$MASTER_IP:7077
wc003$ mkdir -p sparkconf && chmod 700 sparkconf
wc003$ cat <<EOF > sparkconf/spark-env.sh
SPARK_LOCAL_DIRS=/tmp/spark
SPARK_LOG_DIR=/tmp/spark/log
SPARK_WORKER_DIR=/tmp/spark
SPARK_LOCAL_IP=127.0.0.1
SPARK_MASTER_HOST=$MASTER_IP
EOF
wc003$ MYSECRET=$(cat /dev/urandom | tr -dc 'a-z' | head -c 48)
wc003$ cat <<EOF > sparkconf/spark-defaults.sh
spark.authenticate true
spark.authenticate.secret $MYSECRET
EOF
wc003$ chmod 600 sparkconf/spark-defaults.sh
```



```
wc003$ ch-run -b ~/sparkconf /var/tmp/spark -- \
  /spark/sbin/start-master.sh
```

```
wc003$ tail -1 /tmp/spark/log/*master*.out
```

```
18/03/26 20:53:59 INFO Master: I have been elected leader! New state: ALIVE
```

```
wc003$ mpirun -pernode \
  ch-run -b ~/sparkconf /var/tmp/spark -- \
  /spark/sbin/start-slave.sh $MASTER_URL &
```

```
wc003$ fgrep worker /tmp/spark/log/*master*.out | wc
 32      416     2976
```



```
wc003$ ch-run -b ~/sparkconf /var/tmp/spark -- \
    /spark/bin/pyspark --master $MASTER_URL
>>> import operator
>>> import random
>>>
>>> def sample(p):
...     (x, y) = (random.random(), random.random())
...     return 1 if x*x + y*y < 1 else 0
...
>>> SAMPLE_CT = int(2e8)
>>> ct = sc.parallelize(xrange(0, SAMPLE_CT)) \
...     .map(sample) \
...     .reduce(operator.add)
>>> 4.0*ct/SAMPLE_CT
3.14225776
```