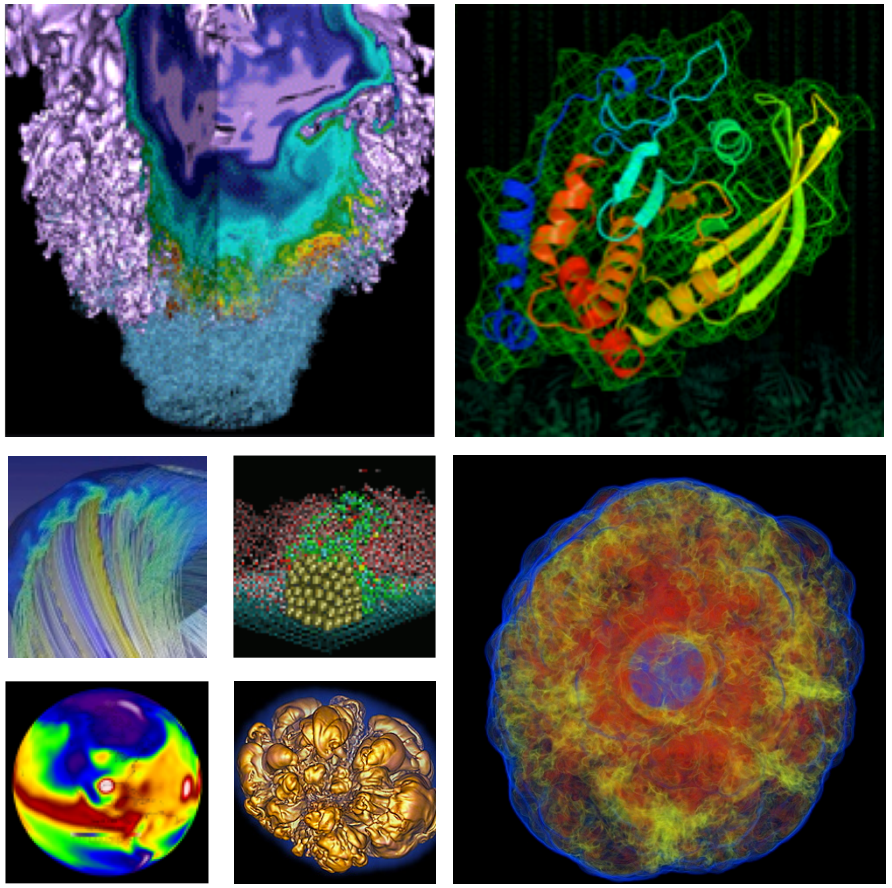
# Containers in HPC




## Shane Canon
## GPUs for Science Days
### Data & Analytics Group, NERSC


July 3, 2019

# Contents

- What are containers and why should I use them?
- Basic Demo
- Containers in HPC with Shifter
- Other HPC Container Runtimes
- Tips and Tricks
- Summary

https://github.com/NERSC/Shifter-Tutorial

(See this repo for tutorials)

# The Struggles

- My software doesn't build on this system…
- I'm missing dependencies…
- I need version 1.3.2 but this system has version 1.0.2..
- I need to re-run the exact same thing 12 months from now…
- I want to run this exact same thing somewhere else…
- I want my collaborators to have the same exact software as me…
- I've heard about these Containers, can I just run that?
- Can I run docker on this HPC system?

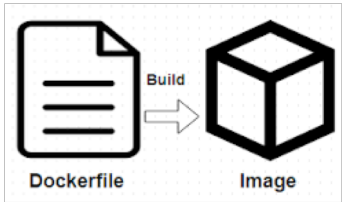
# Solution - Containers

**What are Containers?**

- **Uses a combination of Kernel “cgroups” and “namespaces” to create isolated environments**
- **Long history of containers Solaris Zones (2005), LXC(2008), LMCTFY/Google and then Docker(2013)**
- **Docker provided a complete tool chain to simplify using containers from build to run.**
- **Entire ecosystem has grown around containers especially around orchestration.**

# Docker Basic's

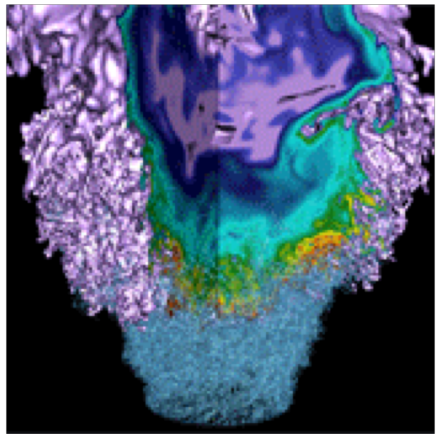

- Build images that captures applications requirements.
- Manually commit or use a recipe file.
- Push an image to DockerHub, a hosted registry, or a private Docker Registry.
- Share Images
- Use Docker Engine to pull images down and execute a container from the image.

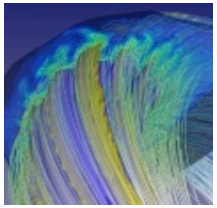


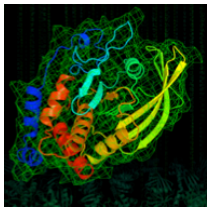

# Containers and Science

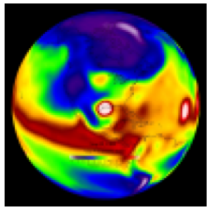
- **Productivity**
  - Pick the OS that works best for your app and use the system package manager to install dependencies.
- **Reusability and Collaboration**
  - Share images across a project to avoid rebuilds and avoid mistakes
- **Reproducibility**
  - Everything you need to redo a scientific analysis can be in the image (apps, libraries, environment setup, scripts)
- **Portability**
  - Can easily run on different resources (of the same architecture)

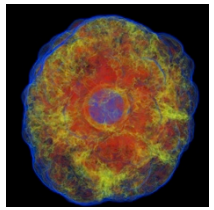
# Containers in Action - Demo

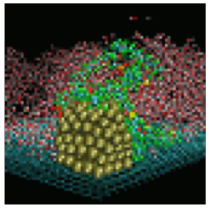




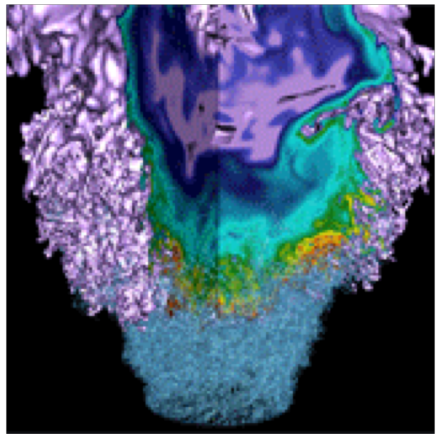


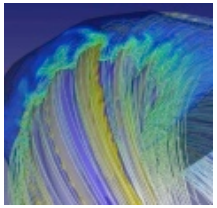


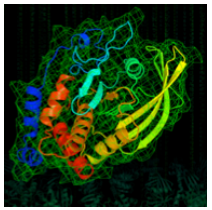


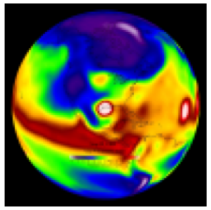


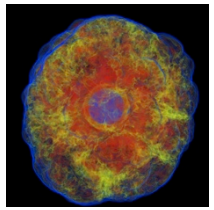
# HPC Container Runtimes

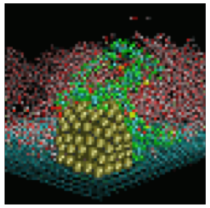






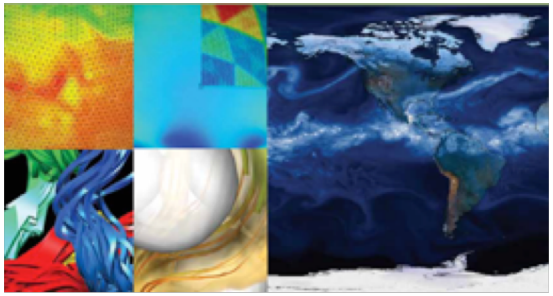


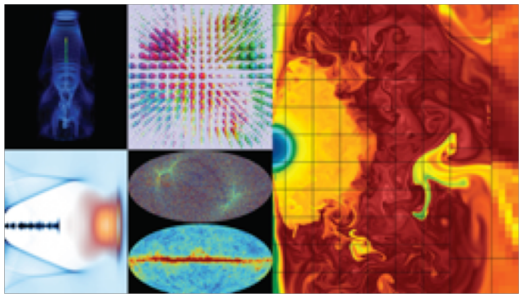


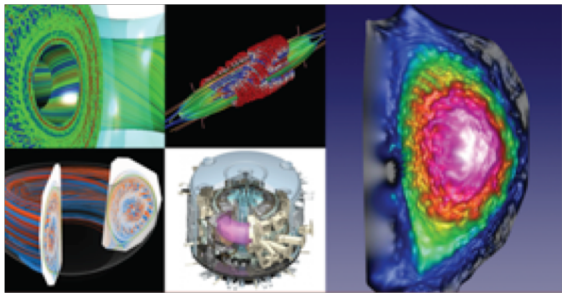


# Why Containers at NERSC

- NERSC deploys advanced HPC and data systems for the broad Office of Science community
- Approximately 6000 users and 750 projects
- Growing number of users around Analyzing Experimental and Observational Data, "Big Data" Analytics, and Machine Learning
- Shift towards converged systems that support traditional modeling and simulation workloads plus new models





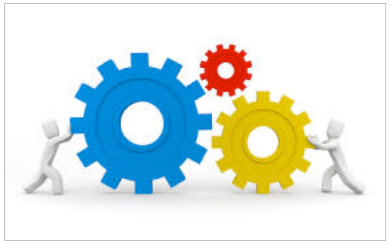


# Why not just run Docker

- **Security: Docker currently uses an all or nothing security model. Users would effectively have system privileges**

```
> docker run -it -v /:/mnt --rm busybox
```

- **System Architecture: Docker assumes local disk**
- **Integration: Docker doesn't play nice with batch systems.**
- **System Requirements: Docker typically requires very modern kernel**
- **Complexity: Running real Docker would add new layers of complexity**



# Solution: Shifter


- **Design Goals:**
  - User independence: Require no administrator assistance to launch an application inside an image
  - Shared resource availability (e.g., file systems and network interfaces)
  - Leverages or integrates with public image repos (i.e. DockerHub)
  - Seamless user experience
  - Robust and secure implementation
- **Hosted at GitHub:**
  - https://github.com/nersc/shifter

# Usage

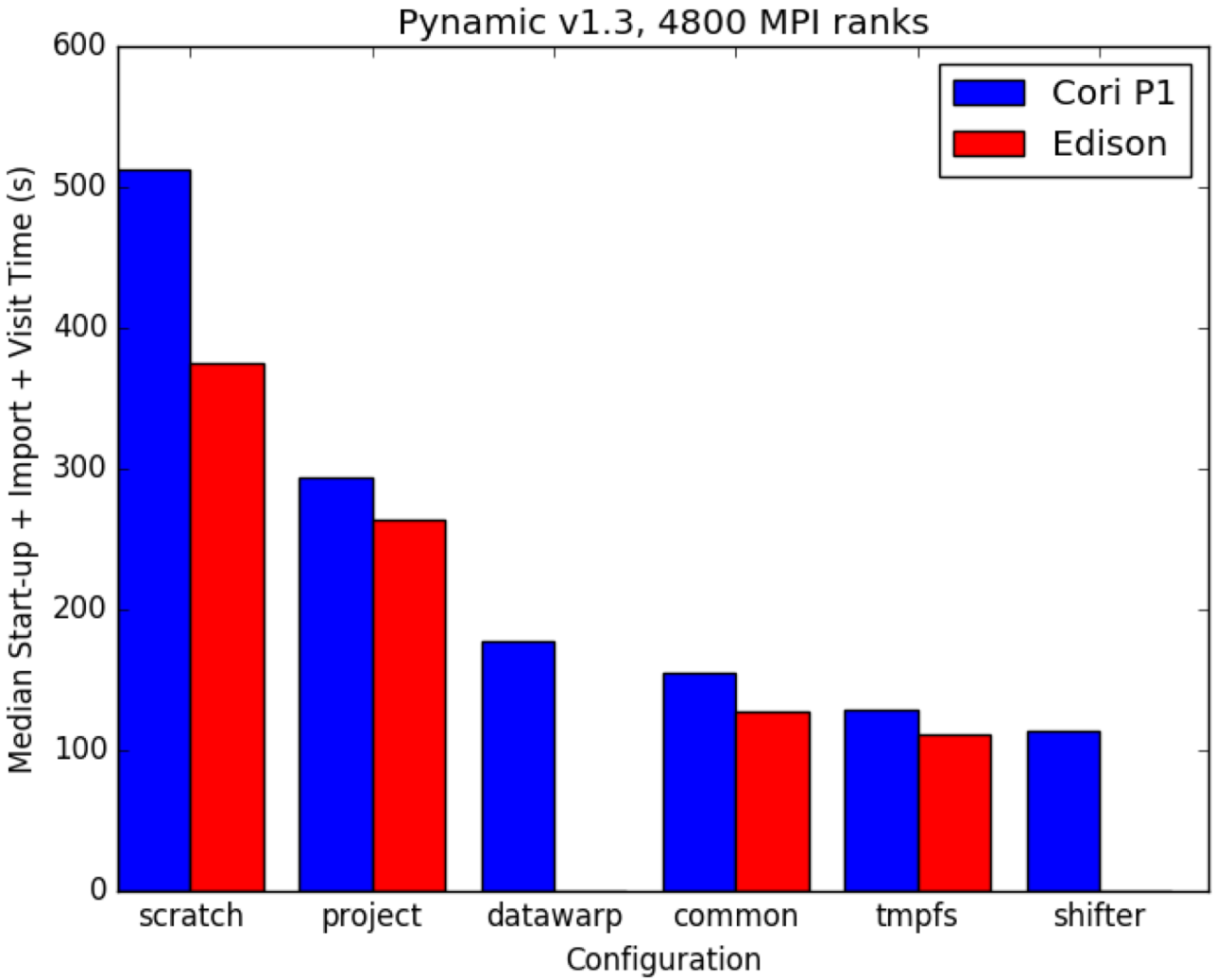
- **Use shifterimg pull to pull images from a registry**
  - Only do this once or after an update

```
> shifterimg pull ubuntu:14.04
```

- **Use shifter command to run a container with an image**

```
> shifter --image=ubuntu:14.04 bash
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.5 LTS
Release:  14.04
Codename:     trusty
```

# Shifter accelerates Python Apps



# Shifter and MPI

- **In Image**
  - Add required libraries directly into image.
  - Users would have to maintain libraries and rebuild images after an upgrade.
- **Managed Base Image (Golden Images)**
  - User builds off of a managed image that has required libraries.
  - Images are built or provided as part of a system upgrade.
  - Constrained OS choices and a rebuild is still required.
- **Volume Mounting**
  - Applications built using ABI compatibility.
  - Appropriate libraries are volume mounted at run time.
  - No rebuild required, but may not work for all cases.

# Running an MPI Job – Building Image

```
FROM nersc/mpi-ubuntu:14.04

ADD . /app
RUN cd /app && \
    mpicc –o hello helloworld.c
```

Dockerfile

```
> docker build –t scanon/hello .
> docker push scanon/hello
```

# Running an MPI Job – Submit and run

```
#!/bin/sh
#SBATCH --image=scanon/hello
srun -np 10 shifter /app/hello
```

submit.sl

```
> sbatch submit.sl
```

# GPU Example

```
###############################################################################
#   Build stage 1
###############################################################################
#
ARG CUDA_VERSION=10.0
ARG UBUNTU_RELEASE=18.04
FROM nvidia/cuda:${CUDA_VERSION}-devel-ubuntu${UBUNTU_RELEASE} as builder
…
COPY ./docker/optimized/requirements.txt /tmp/build/
# install tomopy dependencies
RUN conda install -n tomopy -c conda-forge -c jrmadsen --file requirements.txt
RUN source activate tomopy && \
    echo ${PWD} && ls -la --color=auto && \
    python setup.py install -- -DSKIP_GIT_UPDATE=ON
…
```

# GPU Example - Continued

```
#   Build stage #2 -- compress to single layer
FROM scratch
COPY --from=builder / /
ENV HOME /root
….

COPY ./docker/runtime-entrypoint.sh /runtime-entrypoint.sh
```

# How does Shifter differ from Docker?

**Most Noticeable**

- **Image read-only on the Computational Platform**
- **User runs as the user in the container – not root**
- **Image modified at container construction time (e.g. additional mounts)**

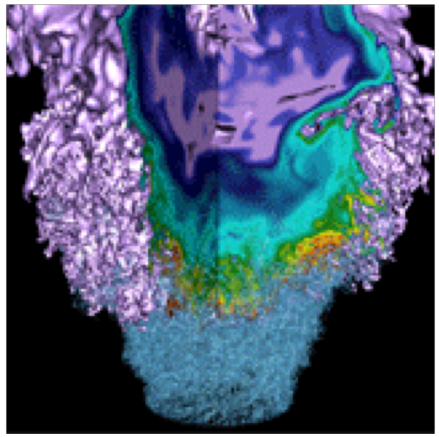
**Less Noticeable:**

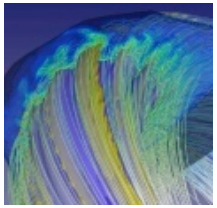
- **Shifter only uses mount namespaces, not network or process namespaces**
- **Shifter does not use cgroups directly (integrated with the Workload Manager)**
- **Shifter uses individual compressed filesystem files to store images, not the Docker graph (slows down iterative updates)**
- **Shifter starts some additional services (e.g. sshd in container space)**

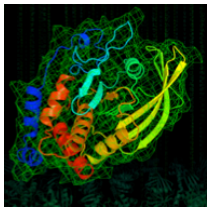
# Other HPC Container Solutions

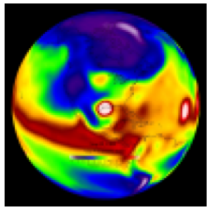
- **Singularity**
  - Available at many DOE Centers
  - Very popular
  - Easy Installation
  - Runtime similar to Shifter
  - Native Image format in addition to Docker
  - Commercial company (Sylabs) now developing it
- **CharlieCloud**
  - Very light-weight
  - Developed and deployed at LANL
  - No special privileges required (so users can install it themselves)
  - Separate tools to unpack Docker images

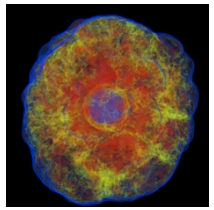
# Other Tips and Tricks

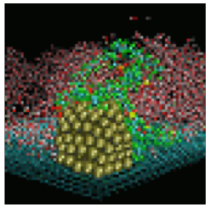












# Volume Mounts

- Volume Mounts provide a way to map external paths into container paths.
- This allows paths in the container to be abstracted so it can be portable across different systems.
- All runtimes support volume mounts but the syntax may vary.
- Basic syntax is:

```
-volume <external path>:<container path>
```

# Using Volume Mounts

```
canon@cori06:~> ls $SCRATCH/myjob
config   data.in

canon@cori06:~> shifter --image=ubuntu --volume=$SCRATCH/myjob:/data bash
~$ ls /data/
config   data.in
```

# PerNode Write Cache (Shifter)

- PerNodeWrite extends the volume concept to create temporary writeable space that aren't shared across nodes.
- These spaces are ephemeral (removed on exit)
- These are node local and the size can be adjusted
- Performs like a local disk but is more flexible
- Basic syntax is

```
--volume <external path>:<container path>:perNodeCache=size=XXG
```

# Using Volume Mounts

```
canon@cori06:~> shifter --image=ubuntu \
      --volume=$SCRATCH:/scratch:perNodeCache=size=100G /bin/bash
~$ df -h /scratch/
Filesystem      Size  Used Avail Use% Mounted on
/dev/loop4      100G   33M  100G   1% /scratch
~$ dd if=/dev/zero bs=1k count=10M of=/scratch/output
10485760+0 records in
10485760+0 records out
10737418240 bytes (11 GB, 10 GiB) copied, 22.2795 s, 482 MB/s
~$ ls -lh /scratch/output
-rw-r--r-- 1 canon canon 10G Nov  9 23:38 /scratch/output
~$ exit

canon@cori06:~> shifter --image=ubuntu \
     --volume=$SCRATCH:/scratch:perNodeCache=size=100G /bin/bash
~$ ls -l /scratch
total 0
```

# Dockerfile Best Practices

**Bad:**

```
RUN wget http://hostname.com/mycode.tgz
RUN tar xzf mycode.tgz
RUN cd mycode ; make; make install
RUN rm -rf mycode.tgz mycode
```

**Good:**

```
RUN wget http://hostname.com/mycode.tgz && \
 tar xzf mycode.tgz && \
 cd mycode && make && make install && \
 rm -rf mycode.tgz mycode
```

# Dockerfile Best Practices

**Bad:**

```
RUN wget http://hostname.com/mycode.tgz ; \
 tar xzf mycode.tgz ; \
 cd mycode ; make ; make install ; \
 rm -rf mycode.tgz mycode
```

**Good:**

```
RUN wget http://hostname.com/mycode.tgz && \
 tar xzf mycode.tgz && \
 cd mycode && make && make install && \
 rm -rf mycode.tgz mycode
```

# Dockerfile Best Practices

**Bad:**

```
ADD . /src

RUN apt-get update -y && atp-get install gcc

RUN cd /src && make && make install
```

**Good:**

```
RUN apt-get update -y && apt-get install gcc

ADD . /src

RUN cd /src && make && make install
```

# Multi-Stage Builds

- Added in Docker 17.05
- Allows a build to progress through stages
- Files can be copied from a stage to later stages
- Useful for splitting images between build and run-time to keep image sizes small
- Can be used to make public images that make use of commercial compilers

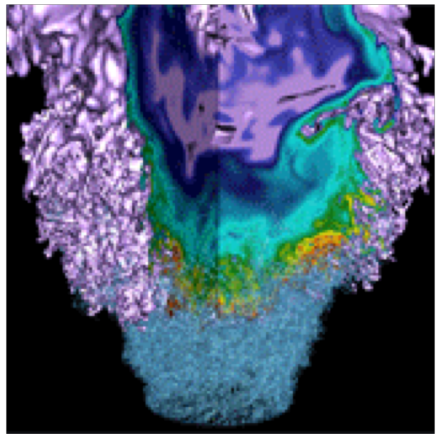
# Dockerfile – Multistage build

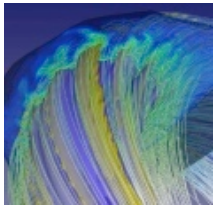
```
FROM centos:7 as build
RUN yum -y install gcc make
ADD code.c /src/code.c
RUN gcc -o /src/mycode /src/code.c

FROM centos:7
COPY --from=build /src/mycode /usr/bin/mycode
```

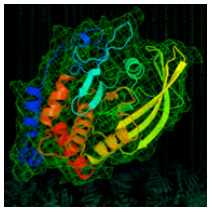
# Other Considerations

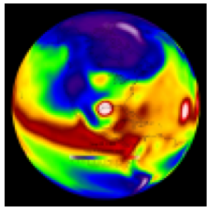
- Avoid very large images (> ~5 GB)
- Keep data in $SCRATCH and volume mount into the container if data is large
- Use volume mounts for rapid prototyping and testing, then add that into the image after code stabalizes

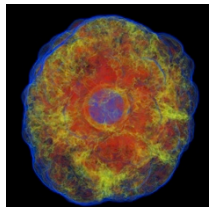
# Use Case Example and Summary

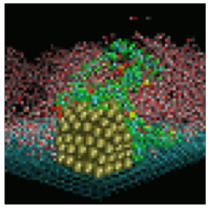






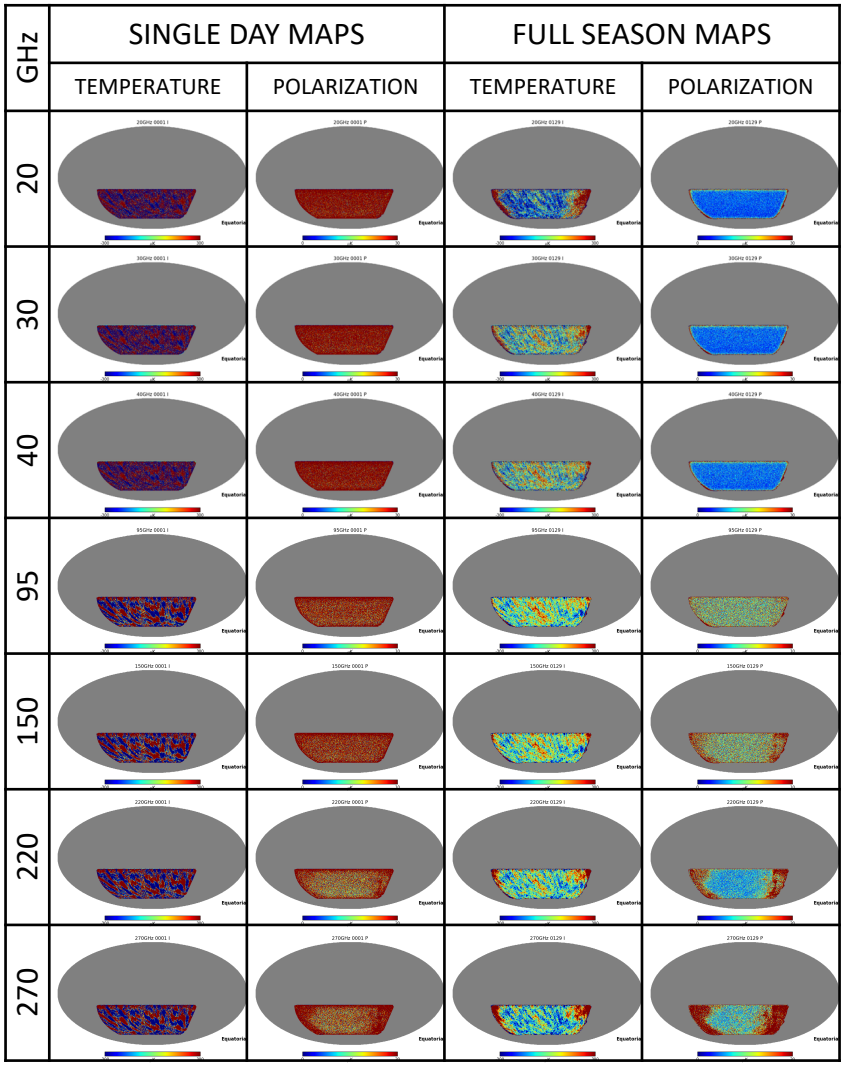






# Measuring the Composition of the Universe

- **CMB – S4**
  - Ambitious collection of telescopes to measure the remnants of the Big Bang with unprecedented precision
- **Simulated 50,000 instances of telescope using 600,000 cores on Cori KNL nodes.**
- **Why Shifter?**
  - Python wrapped code needs to start at scale

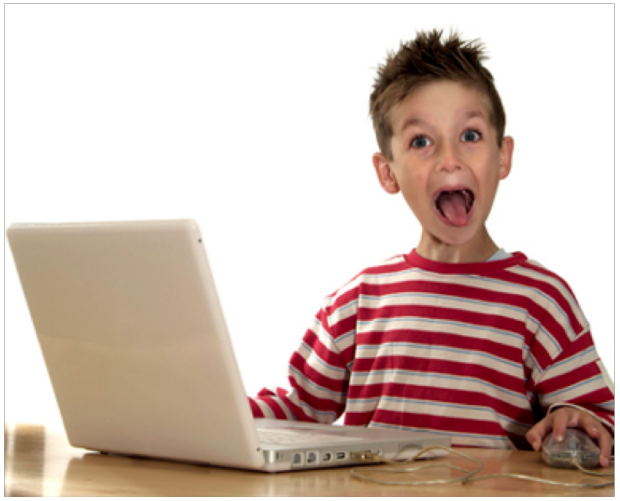



**Courtesy of Ted Kisner**

# Summary

**Containers are great**

- ✓ **Productivity – Get exactly what you need for your application**
- ✓ **Portable – Run the same software on different resources (assuming architectural compatibility)**
- ✓ **Sharable – Collaborators can run the same code as you with less chance of problems**
- ✓ **Reproducible – Run the same image later**
- ✓ **Performant – Can actually speed up applications in some cases**



# Resources

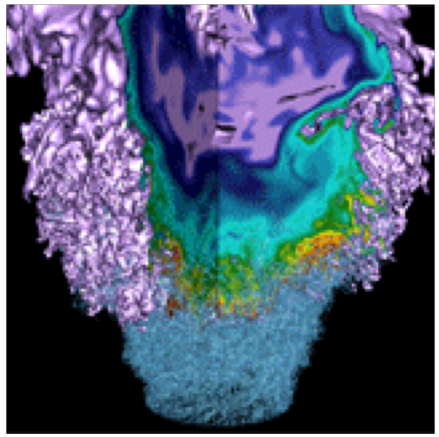
- **Hand on exercises: https://github.com/NERSC/Shifter-Tutorial (look at the IDEAS Branch)**
- **Repo includes previous tutorials and previous slides.**

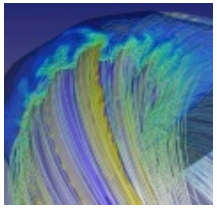
# Questions…

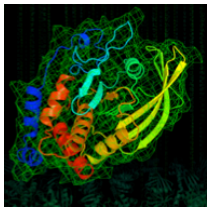
Shane Canon:  scanon at lbl.gov

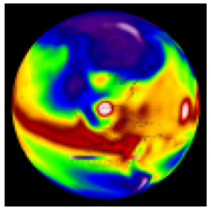
This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under Contract No. **DE-AC02-05CH11231**.

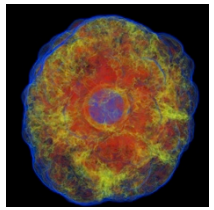
# Reference

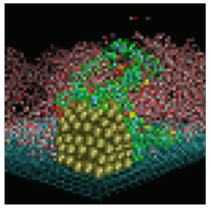






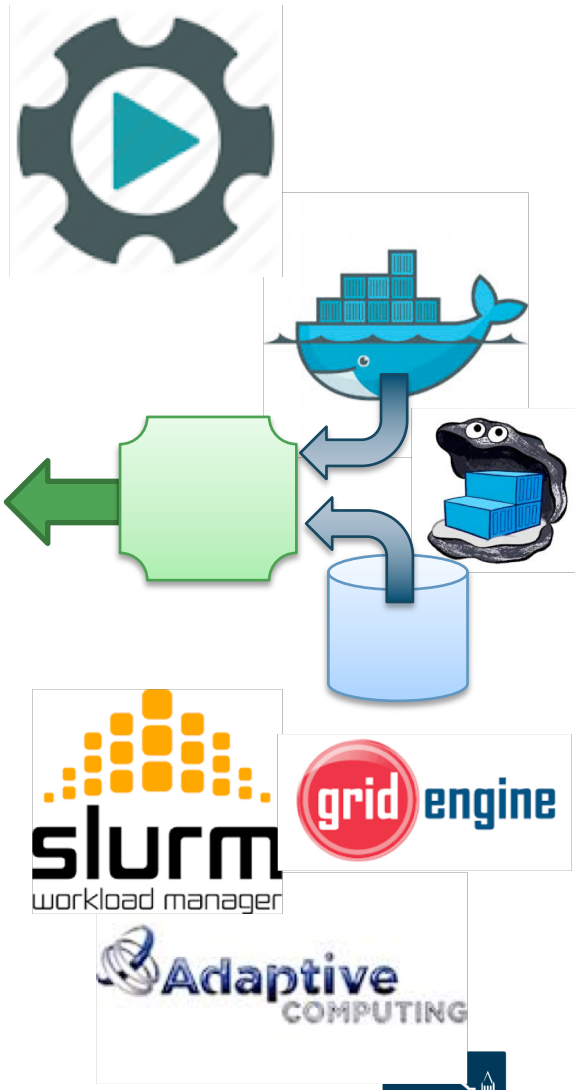






# Shifter Components

- **Shifter Image Gateway**
  - Imports and converts images from DockerHub and Private Registries
- **Shifter Runtime**
  - Instantiates images securely on compute resources
- **Work Load Manager Integration**
  - Integrates Shifter with WLM



# Singularity Recipe File Example

```
Bootstrap: docker
From: ubuntu

%help
Example Singularity Image

%files
    script.sh /script.sh

%labels
    Maintainer I. M. Maintainer
    Version v1.0

%environment
    FOO=bar
    export FOO

%post
    apt-get update -y
    apt-get install -y curl
    echo 'export BAR=blah' >> $SINGULARITY_ENVIRONMENT
```

Singularity

```
> singularity build myimage.simg Singularity
```

# Singularity Execution Examples


```
$ singularity pull --name myimage.simg \
    docker://ubuntu:latest

$ singularity shell myimage.simg
Singularity myimage.simg:~>

$ singularity run myimage.simg
Hello World

$ singularity shell docker://ubuntu:latest
Singularity ubuntu:~>
```

# Charliecloud Execution Examples

```
laptop$ cd /usr/local/src/charliecloud/examples/serial/hello
laptop$ ch-build -t hello .
Sending build context to Docker daemon 5.632kB
[...]
Successfully built 1136de7d4c0a

laptop$ ch-docker2tar hello /var/tmp 114MiB 0:00:03
[==============================================] 103%
 -rw-r----- 1 reidpr reidpr 49M Nov 21 14:05
/var/tmp/hello.tar.gz
```

```
hpc$ ch-tar2dir /var/tmp/hello.tar.gz /var/tmp
creating new image /var/tmp/hello
/var/tmp/hello unpacked ok

hpc$ ch-run /var/tmp/hello -- echo "I'm in a container"
I'm in a container
```