# Containerization: Best Practices & Advanced Topics

Peter Z. Vaillancourt
Computational Scientist
Center for Advanced Computing (CAC)
Cornell University
XCRI Engineer, XSEDE

XSEDE
Extreme Science and Engineering Discovery Environment

Containerization: Best Practices & Advanced Topics

# Outline

- Lifecyle of a Container
  - Development vs. Production
  - Reducing Container Sizes
  - Data Management
- Deploying a Container
  - Registries
  - In the Cloud
  - On a Shared Resource
- Security
- Next Steps
  - Reproducible Containers
  - Container Orchestration

## Lifecycle of a Container

# Containers for Development vs. Production

1. **Production:** Containers as a software distribution method
   - Portability of a consistent environment for users
   - Easily distributed
   - Highly accessible
   - Pre-packaged software containers often require customization

Lifecycle of a Container

# Containers for Development vs. Production

1. **Production:** Containers as a software distribution method
   - Portability of a consistent environment for users
   - Easily distributed
   - Highly accessible
   - Pre-packaged software containers often require customization
2. **Development:** Containers as a development environment
   - Builds a consistent environment early, including dependencies
   - Useful for teams of developers/researchers
   - Larger if including dev tools
   - Often requires cleanup for production

Lifecycle of a Container

# Containers for Development vs. Production

## Development

## Production

## Lifecycle of a Container

# Containers for Development vs. Production

### Development

- Contains dependencies, code, environment variables, etc.
- No real size limit: text editors, VNC, data visualization, etc.
- Code is changed and updated
- Runs can be varied and versatile to initiate

### Production

## Lifecycle of a Container

# Containers for Development vs. Production

### Development

- Contains dependencies, code, environment variables, etc.
- No real size limit: text editors, VNC, data visualization, etc.
- Code is changed and updated
- Runs can be varied and versatile to initiate

### Production

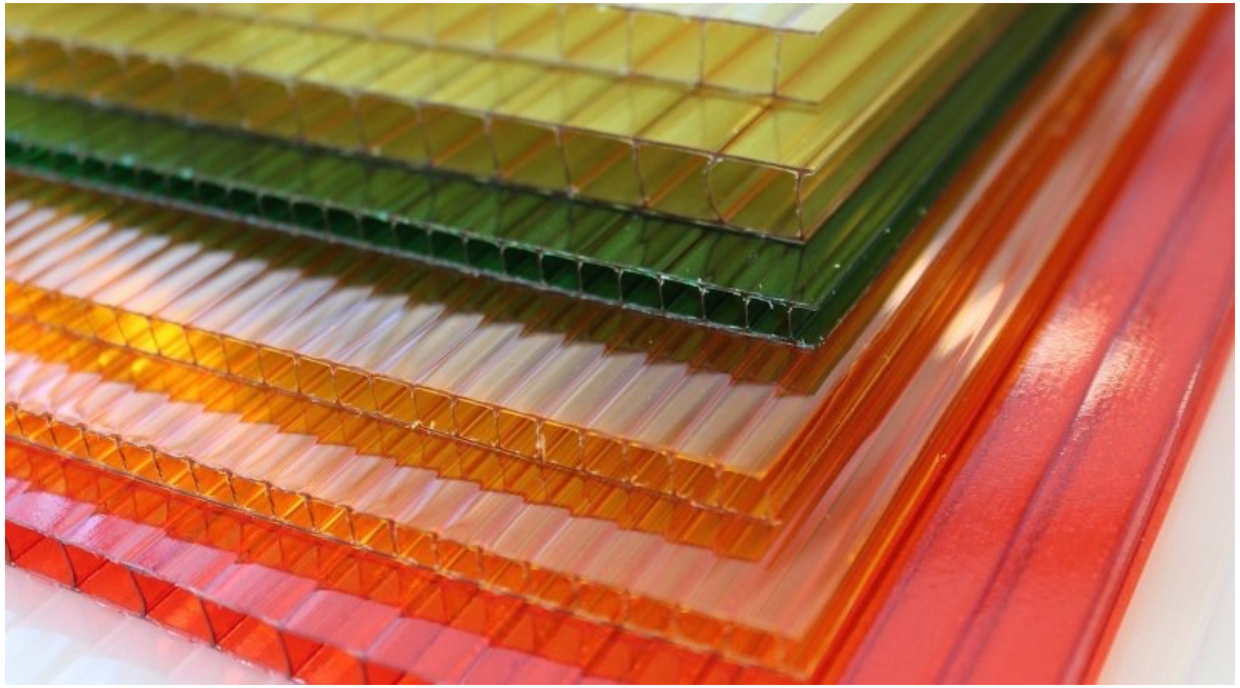
- Contains dependencies, code, environment variables, etc.
- Should be as lightweight as possible: no need for nice aesthetic features
- Code is static
- Requires a run script or easy commands

## Lifecycle of a Container

# Reducing Container Sizes

- Docker Layers
  - Base image

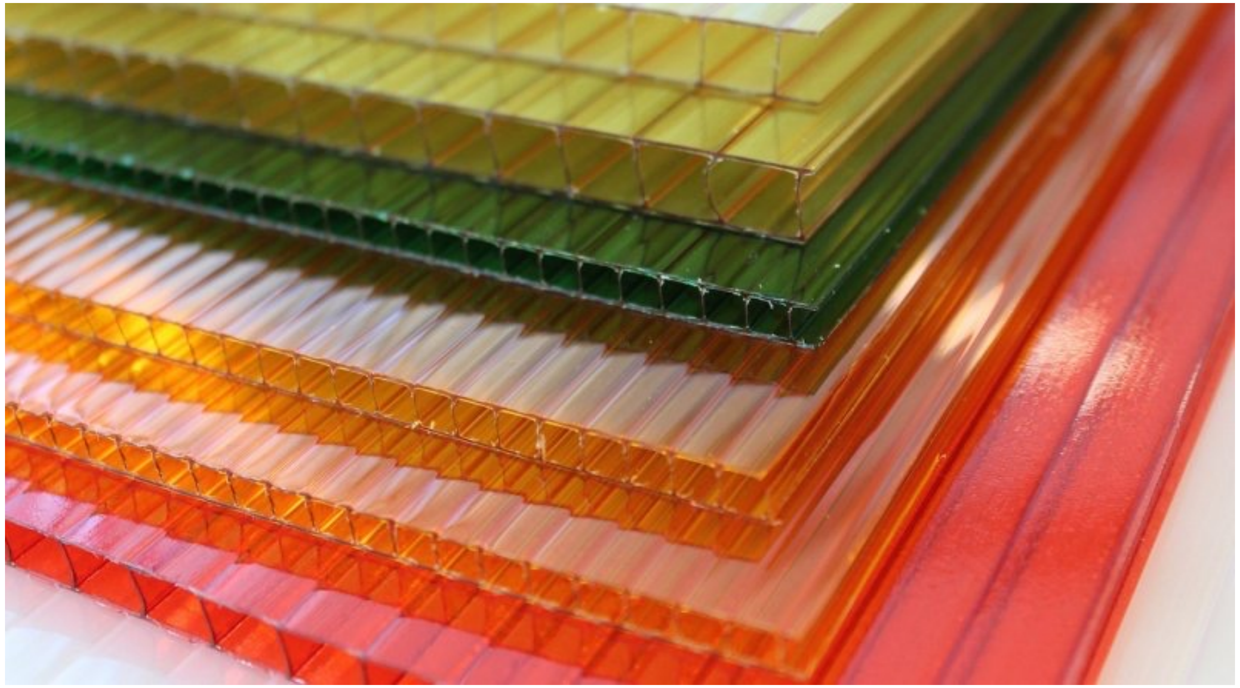
| | |
|---|---|
| CentOS | 215MB |
| Debian | 114MB |
| Ubuntu | 73.9MB |
| Alpine | 5.57MB |



# Lifecycle of a Container

## Reducing Container Sizes

- Docker Layers
  - Base image
    - CentOS  215MB
    - Debian   114MB
    - Ubuntu   73.9MB
    - Alpine    5.57MB
  - Certain commands add layers: RUN, ADD, COPY
  - 1 instruction = 1 layer
  - Other commands create temporary layers
  - Also see the Docker docs



## Lifecycle of a Container

# Reducing Container Sizes

- Combining multiple commands
  - Pip commands can use a requirements file

Example requirements.txt:

```
alembic
fitsio==0.9.11
requests_oauthlib
marshmallow
ephem
scikit-sparse
corner
numexpr
astropy
runipy
…
```

## Lifecycle of a Container

# Reducing Container Sizes

- Combining multiple commands
  - Pip commands can use a requirements file
  - If using several RUN commands in a row, it’s an opportunity to combine:

```
RUN wget -q https://bitbucket.org/psrsoft/tempo2/get/master.tar.gz && \
    tar zxf master.tar.gz && \
    cd psrsoft-tempo2-* && \
    ./bootstrap && \
    CPPFLAGS="-I/opt/pulsar/include" LDFLAGS="-L/opt/pulsar/lib" ./configure -- prefix=/opt/pulsar --with-calceph=/opt/pulsar && \
    make && make install && make plugins && make plugins-install && \
    mkdir -p /opt/pulsar/share/tempo2 && \
    cp -Rp T2runtime/* /opt/pulsar/share/tempo2/. && \
    cd .. && rm -rf psrsoft-tempo2-* master.tar.gz
```

# Reducing Container Sizes

- Combining multiple commands
  - Pip commands can use a requirements file
  - If using several RUN commands in a row, it’s an opportunity to combine

- Use multi-stage builds
  - Leverages docker build cache
  - Can be used to remove some contents after build for security

## Lifecycle of a Container

# Reducing Container Sizes

- Combining multiple commands
  - Pip commands can use a requirements file
  - If using several RUN commands in a row, it’s an opportunity to combine

- Use multi-stage builds
  - Leverages docker build cache

- Don’t install what you don’t need

- Multiple decoupled containers (microservices)

## Lifecycle of a Container

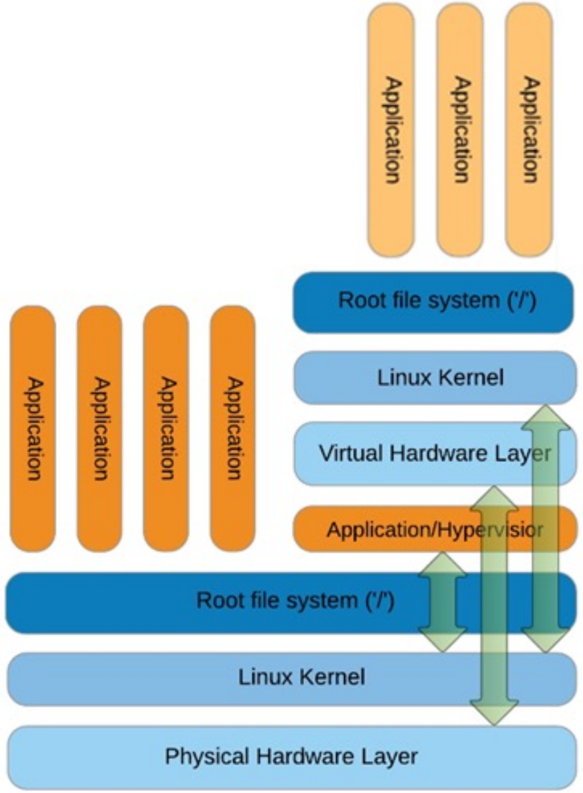
# Reducing Container Sizes

- Using a prebuilt image from a registry?
  - Decide whether to build your own
  - Look at the layers to see what is excess
  - Build files on GitHub can reveal unnecessary additions

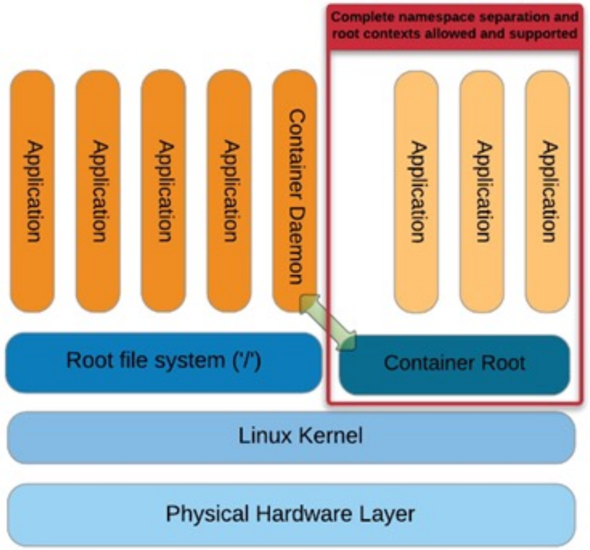
- Building your own is the best way to ensure you have only what you need

# Lifecycle of a Container

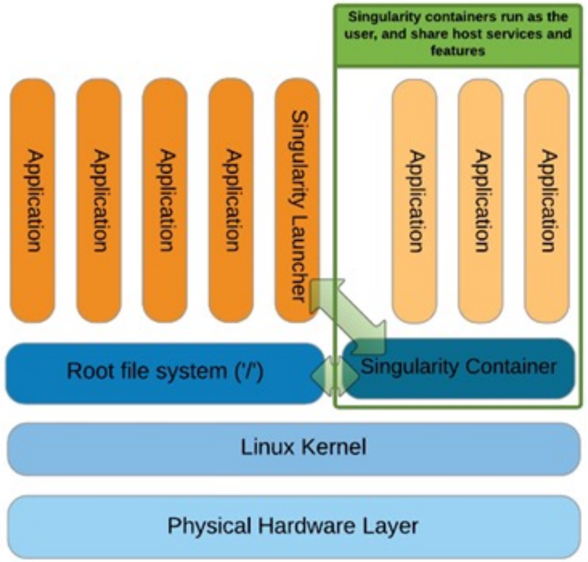
## Data Management



General VM
eg ESXi


Complete namespace separation and root contexts allowed and supported

General Container
eg Docker


Singularity containers run as the user, and share host services and features

HPC Container
Singularity

# Data Management

- Manage data in Docker: https://docs.docker.com/storage/
  - Volumes https://docs.docker.com/storage/volumes/
  - Bind mounts https://docs.docker.com/storage/bind-mounts/
  - tmpfs mounts https://docs.docker.com/storage/tmpfs/

- Manage data in Singularity
  - Access to the root filesystem: https://sylabs.io/guides/3.5/user-guide/quick_start.html#working-with-files
  - Bind Paths and Mounts: https://sylabs.io/guides/3.5/user-guide/bind_paths_and_mounts.html
  - Persistent Overlays: https://sylabs.io/guides/3.5/user-guide/persistent_overlays.html

# Deploying a Container

## Uploading Containers

- Containers on public registries can be a great way to
  - Share scientific software portably
  - Help others reproduce your research
  - Create a community around a software/workflow
  - Set up automated builds (connect your GitHub repo)

- Common Public Registries
  - DockerHub
  - Sylabs Cloud Hosting (11GB limit)

# Best Practices for Uploading Containers

- Don’t upload
  - Private data – very important for research
  - Private or licensed software

- Do include
  - Software licenses
  - Documentation
  - Software and dependencies
  - Runscripts for production

# Deploying a Container

## In the Cloud

- Will it work in the cloud?
  - Moving from HPC adds complexity
    - MPI
    - May require container orchestration
  - Data management

- Use Docker
  - Public cloud providers offer managed services
  - Container Orchestration options
  - Ease of use

Vaillancourt and Wineholt et al. 2020 PEARC20

# Deploying a Container

## In the Cloud

- What is the right cloud?
  - Private clouds using OpenStack have various options
  - Public clouds have vendor-specific options galore
  - Multi-cloud or generalized tools also available
- Security is a consideration

Vaillancourt and Wineholt et al. 2020 PEARC20

# Deploying a Container
# On HPC Resources

- Containers can simplify getting started
  - No need to install to your home directory
  - No need to pester sysadmins to install your software

- Using Singularity on XSEDE
  - It’s available and secure
  - Automatic mounts for easy data access
  - Multinode orchestration via Slurm

- Job scripts and bind mounts may vary on different systems

# Deploying a Container
# On HPC Resources

- Singularity with MPI
  - MPI **major version** in the container *must* match the host
    https://sylabs.io/guides/3.5/user-guide/mpi.html
  - Singlenode will work if MPI is run inside the container
  - Multinode is more dependent on system
    - Choice of MPI library may be limited by hardware
    - Scheduler integrations, such as Slurm

## Security

# Root Access

- Sensitive systems cannot allow root access

- XSEDE systems use Singularity

- Another option is Docker Rootless Mode
  - Docker Docs on Rootless Mode
  - DockerCon 2020 Talk on Rootless Mode
  - Recently became a fully supported feature

- Setup a user or users for shared Docker containers
  (same as shared system)
  https://docs.docker.com/develop/develop-images/dockerfile_best-practices

## Security

# Cloud VMs

- Implement security at a Virtual Machine (VM) level
  - Firewall
  - Security Groups
  - Limit ssh access

- For public images
  - Pay attention to what they contain
  - Look for the Dockerfile
  - GitHub repo

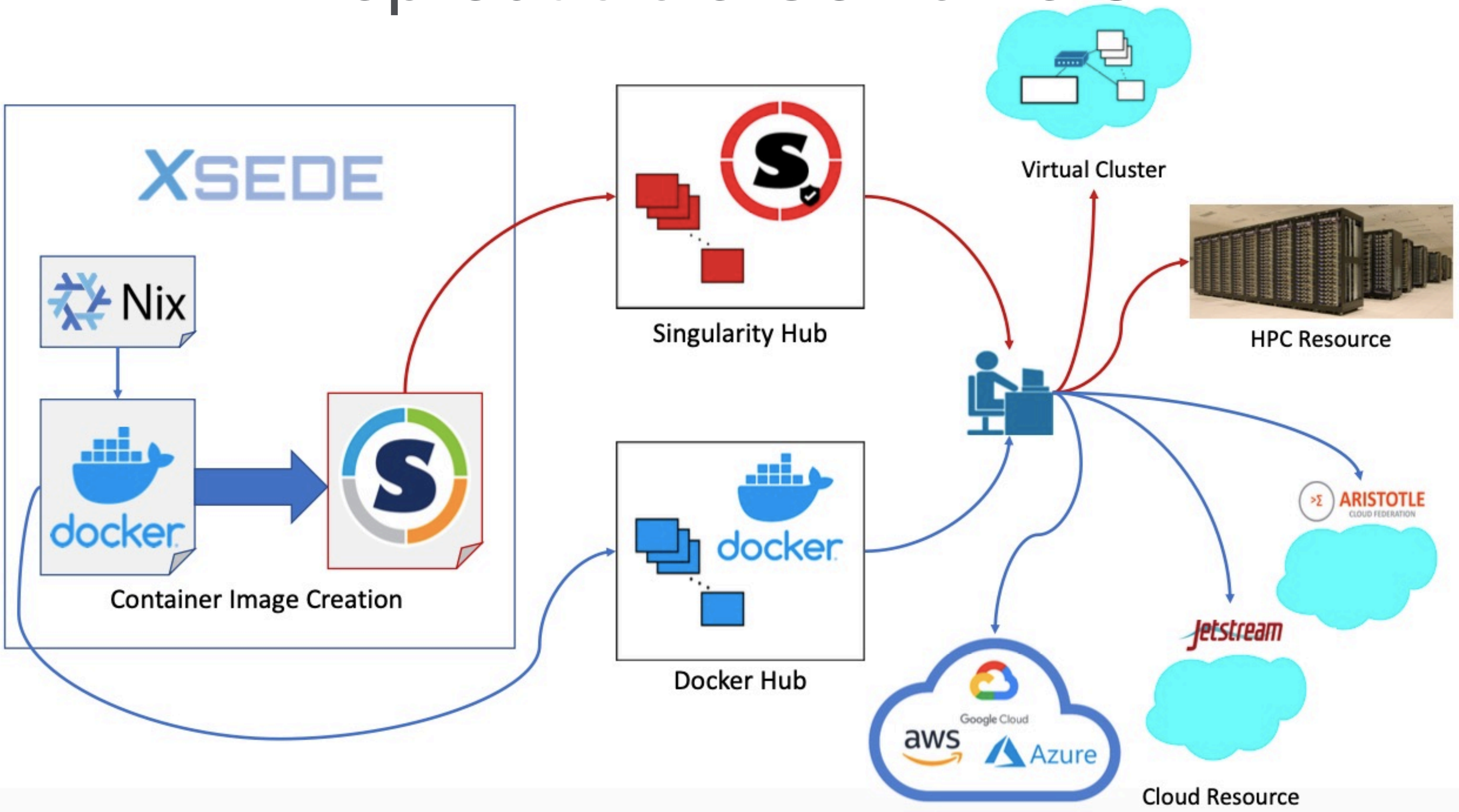
# Other Security Features

- Docker vulnerability scanning
  https://docs.docker.com/engine/scan
  https://github.com/docker/scan-cli-plugin

- Singularity Security Options
  - Linux Capabilities
  - Encrypted Containers
    https://sylabs.io/guides/3.5/user-guide/security_options.html#security-options

## Next Steps

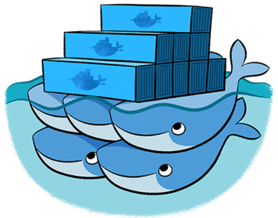
# Reproducible Containers



Vaillancourt and Coulter et al. 2020 Self-Scaling Clusters and Reproducible Containers to Enable Scientific Computing

## Next Steps

# Container Orchestration



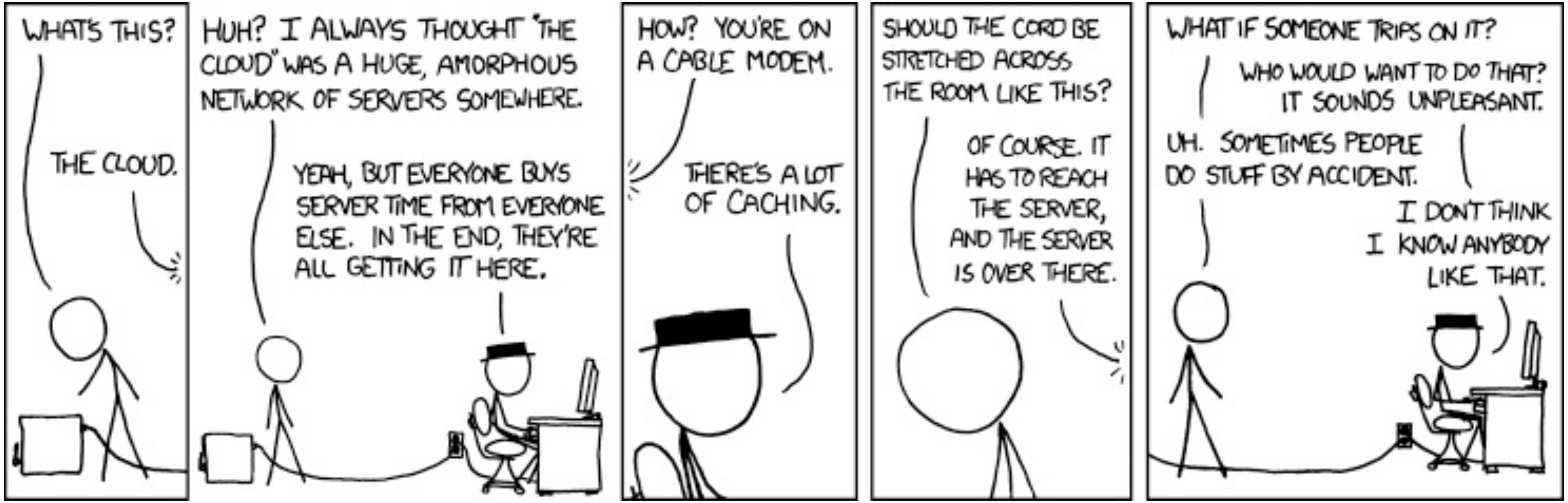
kubernetes

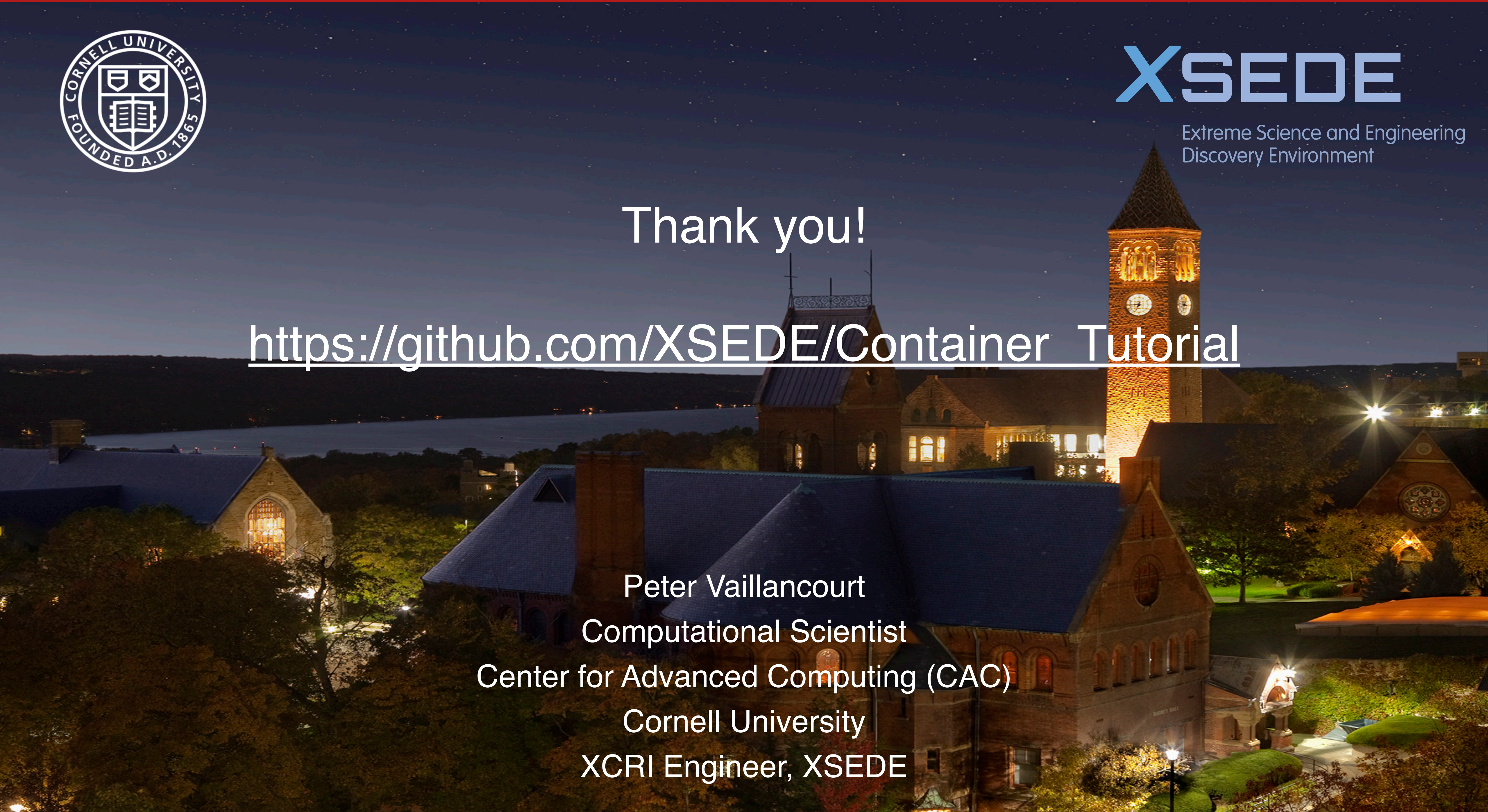
HashiCorp
Terraform

+ ANSIBLE

Vaillancourt and Wineholt et al. 2020 PEARC20

# Questions?


WHAT'S THIS?
THE CLOUD.
HUH? I ALWAYS THOUGHT "THE CLOUD" WAS A HUGE, AMORPHOUS NETWORK OF SERVERS SOMEWHERE.
YEAH, BUT EVERYONE BUYS SERVER TIME FROM EVERYONE ELSE. IN THE END, THEY'RE ALL GETTING IT HERE.
HOW? YOU'RE ON A CABLE MODEM.
THERE'S A LOT OF CACHING.
SHOULD THE CORD BE STRETCHED ACROSS THE ROOM LIKE THIS?
OF COURSE. IT HAS TO REACH THE SERVER, AND THE SERVER IS OVER THERE.
WHAT IF SOMEONE TRIPS ON IT?
WHO WOULD WANT TO DO THAT? IT SOUNDS UNPLEASANT.
UH. SOMETIMES PEOPLE DO STUFF BY ACCIDENT.
I DON'T THINK I KNOW ANYBODY LIKE THAT.



XSEDE

Extreme Science and Engineering
Discovery Environment

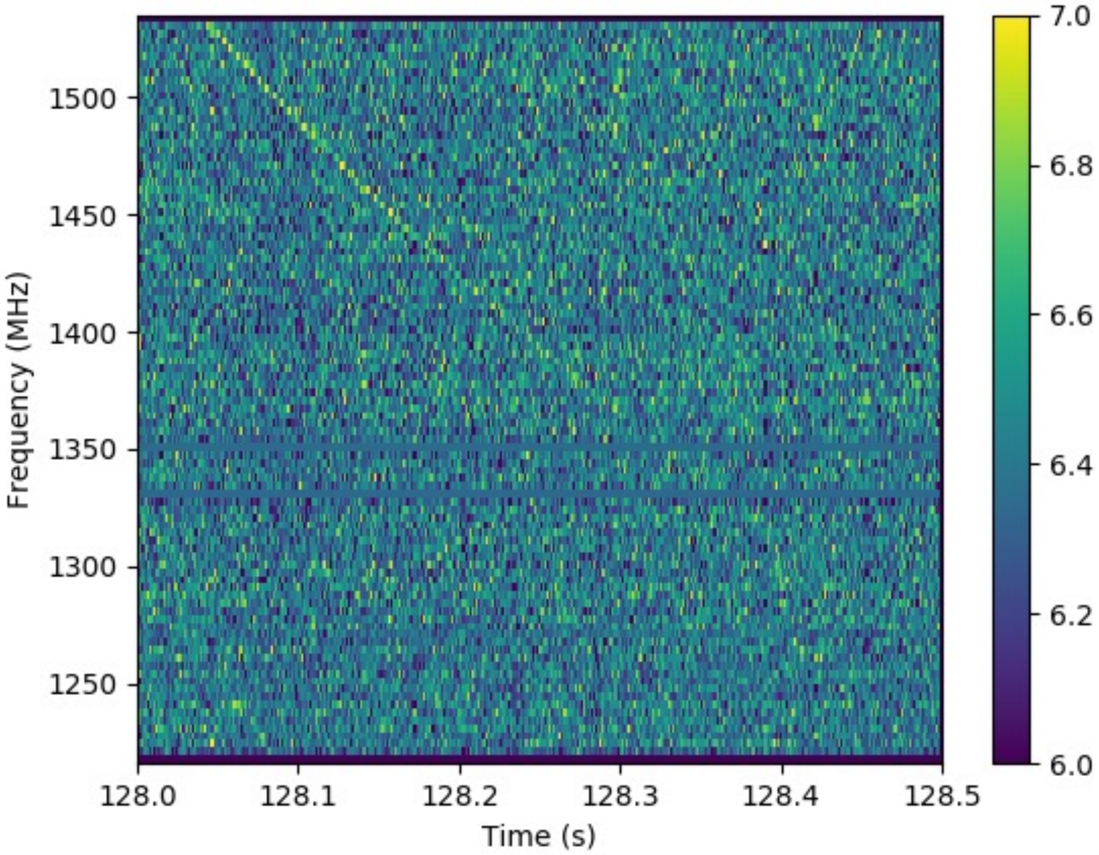
# Thank you!

https://github.com/XSEDE/Container_Tutorial

Peter Vaillancourt
Computational Scientist
Center for Advanced Computing (CAC)
Cornell University
XCRI Engineer, XSEDE

# Lifecycle of a Container

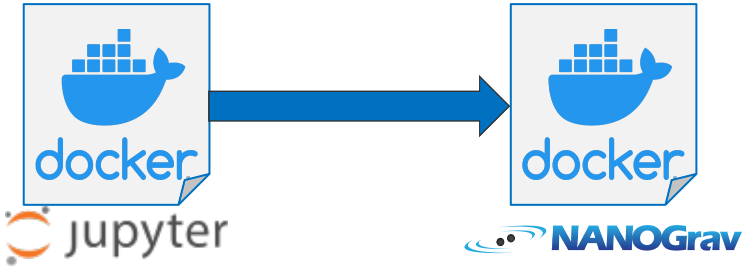
## Example Container: Radio Astronomy



## Lifecycle of a Container

# Example Container: Radio Astronomy

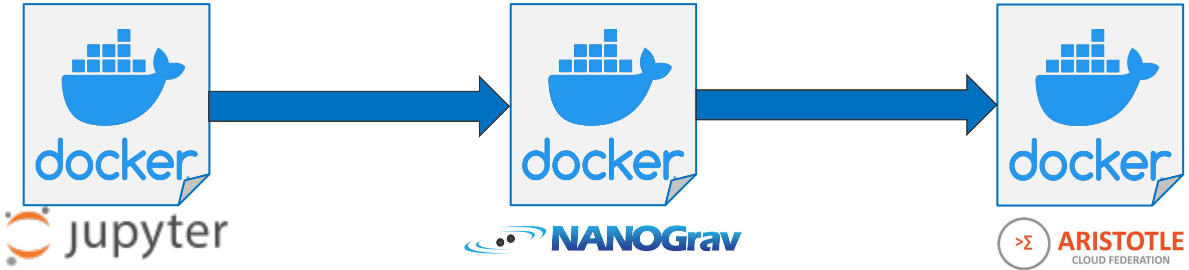
- Started with a NANOGrav container: nanograv/nanopulsar
  - Based on jupyter/datascience-notebooks (includes Python, R, and more)
  - Wide variety of Radio Astronomy software and tools



## Lifecycle of a Container

# Example Container: Radio Astronomy

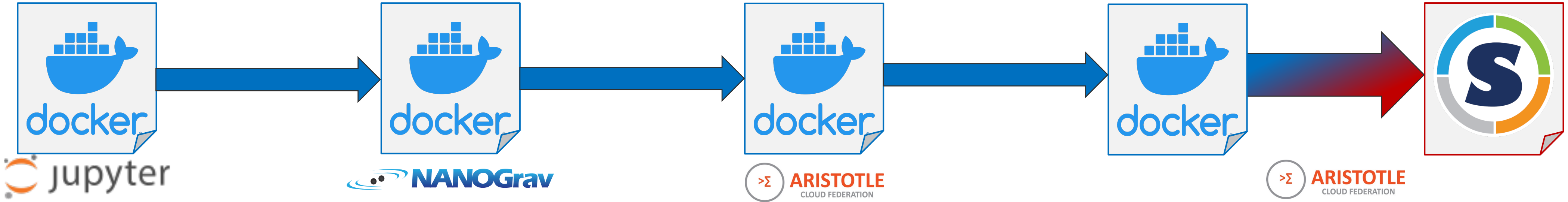
- Started with a NANOGrav container: nanograv/nanopulsar
  - Based on jupyter/datascience-notebooks (includes Python, R, and more)
  - Wide variety of Radio Astronomy software and tools
- After 1 year, needed to be updated: federatedcloud/nanopulsar
- Used for development with additions: federatedcloud/modulation_index
  - ~11GB for just dependencies



## Lifecycle of a Container

# Example Container: Radio Astronomy

- Started with a NANOGrav container: nanograv/nanopulsar
  - Based on jupyter/datascience-notebooks (includes Python, R, and more)
  - Wide variety of Radio Astronomy software and tools
- After 1 year, needed to be updated: federatedcloud/nanopulsar
- Used for development with additions: federatedcloud/modulation_index
  - ~11GB for just dependencies
- Created a minimal container for production runs
  - ~3GB for just dependencies
  - Docker version: federatedcloud/docker-PRESTO
  - Singularity version: federatedcloud/singularity-PRESTO


docker
jupyter
docker
NANOGrav
docker
ARISTOTLE
CLOUD FEDERATION
docker
ARISTOTLE
CLOUD FEDERATION