Containers Walkthrough

Yee-Ting Li, April 2020

ML@SLAC, SLAC

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

(SLAC Shared) Scientific Data Facility

- **Greenfield** deployment of hardware, software and policies for Scientific Computing at SLAC
  - Initially led by LCLS, CryoEM and ML initiatives
- Governed by newly proposed **SDF Steering Committee** members whom are ‘**Owners’** in the **SDF**
- New **Storage**
  - Most ‘legacy’ storage will be migrated to new disks
    - Manual process, will need to work with you to determine best way to move data (archive data)
  - Baseline ‘Free’ storage - to be determined.

-

(SLAC Shared) Scientific Data Facility

- New Hardware
  - All **GPUs** migrated from LSF to Slurm
  - 88 x **AMD Rome** Dual 64-core ordered (~12k cores)
  - 200GB/sec “HDR” Infiniband network
- **‘Baseline’** paid out of indirect costs and investments
  - Every SLAC (computer) account will have
    - Some allotment of compute CPU and GPU ‘for free’
    - Some allotment of storage ‘for free’
  - Balancing act of resources!

SDF Compute Usage

- Priority Access to Compute Resources
  - **‘Owners’** can contribute compute hardware and/or FTE to help support planning, operations and development
  - Unused (compute) cycles from ‘Owners’ hardware will be made available to all Users → ‘Shared’
  - Owners will have **pre-emptive access** to their own hardware contribution
  - Minimum contribution: 1 server
- Extra ‘Shared’ (compute) cycles bourne from lab direct investment
  - E.g. ocio-gpuXX machines
- Hardware spec to be reviewed and governed annually by the **SDF Steering Committee**

SDF Storage

- Designed and built upon DDN Exascaler Platform
- High speed controllers (~70GB/sec each)
- Current deployment
  - 2 sets of controllers, plan to expand to more.
  - 6+PB of spinning disk storage (2PB LCLS, 2PB CryoEM, 2PB Baseline)
- Calls for more storage will be pooled each period (per quarter?)
  - Minimum pool size ~500TB, approx 25k/5 years (TBC)
  - Possible chargebacks for % fair usage of shared hardware and service
- Migration
  - Plan to be de-facto storage for SLAC
  - Will need to work with everyone to existing data into SDF
  - Limited legacy cross mounts will be provided

Common Challenges

- Spend a long time installing applications and dependencies on each new cluster
  - Where do install? Do i have an existing script for installs?
- Sometimes certain versions of libraries not available
  - python3?! boost, tensorflow v1 code etc.
  - `module load` only helps a little - often maintained by cluster admins, may not have specific versions available (cuda?), conflicts between modules etc.
  - virtualenv/conda - complementary to containers
- Typical to install into group/home directories
  - End up hard coding paths - what if filesystem isn't available where you need it?
- Difficult to determine versions etc. (unless CVMFS tree structure like)
  - How to version control the application? Does new library create different results?
  - How to use different versions of the application?

Container Benefits

- Consistent Environment
    - Self documenting
    - Shareable! Fork, pull etc.
    - Repeatable, reproducible
- Run anywhere
    - No need to compile for each cluster
- (Much) smaller and more efficient than Virtual Machines
- Faster than rebuilding everywhere with conda, pip etc.
    - But can utilize standard scripts and methods to create container
- Don’t like using yum? Use apt instead, even if Host OS is CentOS.

Container Challenges

SLAC

- Numerous different technologies
  - Docker, singularity, shifter, charliecloud, podman, enroot…
- Most solutions require sudo/root to build images
- GPU support can be… interesting
- Primarily a Linux only technology
  - Most cross platform solutions are VM based (exception WSL)
- Mostly aimed at Enterprise use cases (running webservers) rather than HPC (massively parallel workloads? Multi-user systems?)

Docker

- The 'de facto' container technology
- Dockerhub allows free storage and searching of containers
  - Create account at https://dockerhub.com
  - Chances are someone's already ported what you want
- Uses a local cache to store images, so each computer needs a duplicate copy
- Needs sudo/root to build and run (some 'rootless' features in latest version)
- Running docker containers typically have full root

Singularity

- Container technology aimed at HPC applications
- Uses local ‘.sif’ file per container that can be copied and moved like any other file
- Can convert from a docker image to singularity file
- Supported at most academic and laboratory institutions
  - Major exceptions: NERSC
- Containers can be pull’d and ran without sudo/root
  - Great for multi-user environments like our clusters
- Running singularity container has same uid/gids as on host OS (inside == outside)
- Simple integration with GPUs, MPI, X etc.
- Some (persistent) issues on AFS systems.

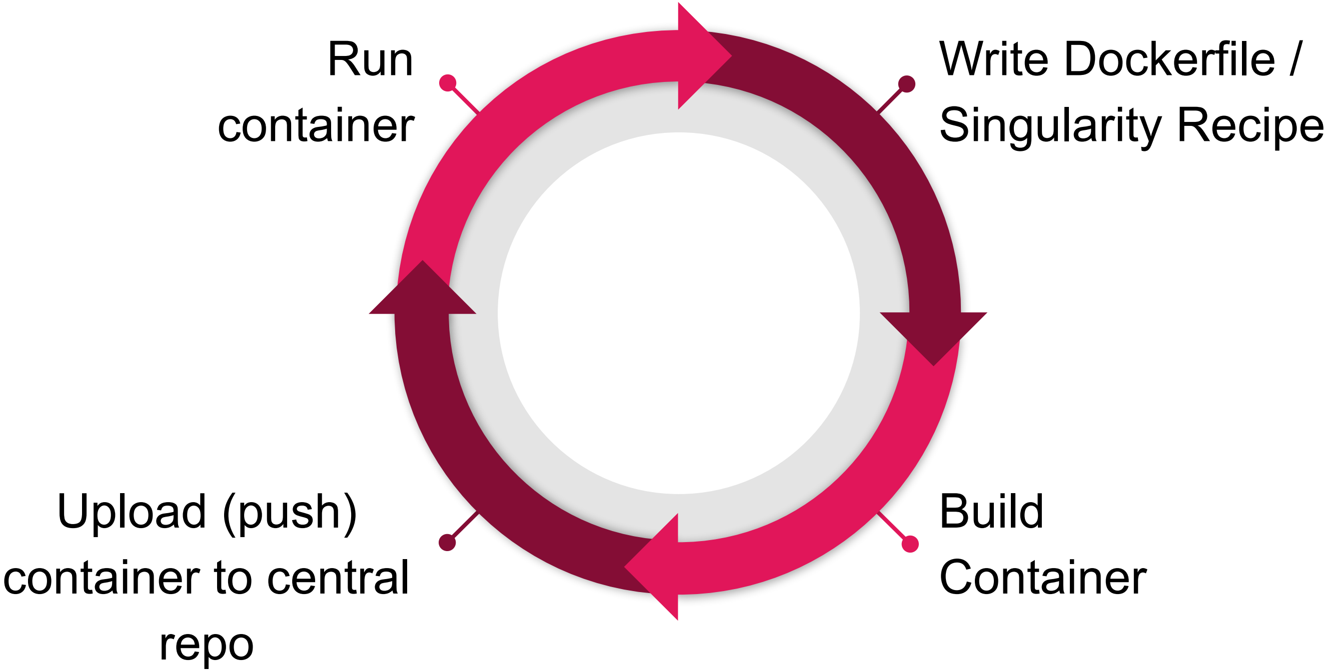
Why use Docker + Singularity?

- Use Docker for building
  - Simple language, well documented, plenty of examples
  - Support for cached ‘layers’ to allow for quick changes
  - Many different container technologies allow conversion from Docker images directly
    - Ie use the same image to run at NERSC
- Use singularity for running
  - No need for sudo/root
  - Can share applications as a single file

Demo Outline

- Create a simple python based application
- Build a Docker container from in
- Push container to Dockerhub
- Convert the Docker container to a Singularity container
- Deal with bind-mounts
- Run the singularity container
- Note about GPUs

Container Lifecycle



Not covering singularity image create - not self documenting

Setup

	Mac	Linux
Install Docker	https://docs.docker.com/docker-for-mac/	https://docs.docker.com/engine/install/
Install Singularity	https://sylabs.io/singularity-desktop-macos/ or Vagrant (see later)	https://sylabs.io/guides/3.0/user-guide/installation.html

Vagrant Install

- Need sudo! :(
- We can’t give sudo on HPC nodes
- Only viable solution currently is for you to use your own laptop/desktop
- Use a VM (not necessary on Linux)
  - Control it Vagrant

```
# on mac
brew cask install virtualbox
brew cask install vagrant
```

https://www.vagrantup.com/downloads.html

```
# on ubuntu
sudo apt install virtualbox
dpkg install vagrant_2.2.7_x86_64.deb

# centos
sudo curl
http://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo > /etc/yum.repo.d/virtualbox.repo
yum update && yum install VirtualBox-5.1
yum install vagrant_2.2.7_linux_amd64.zip
```

Start a Vagrant VM

```
# on host OS, start a new VM
mkdir centos7
vagrant init centos/7  # vagrant init sylabs/singularity-3.5-centos-7-64
vagrant up

# enter VM
vagrant ssh
ls /vagrant # directory from host OS shared here

# install docker and singularity
sudo -s
yum install -y epel-release
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install -y singularity
yum install -y docker-ce && systemctl start docker

# exit out of VM and delete it
^d^d
vagrant snapshot save clean-install # vagrant destroy
```

Lets create a pytorch container!

- Find useful/official/collaborator's image (or build your own from scratch):
  - https://hub.docker.com/search?q=pytorch&type=image
  - Numerous versions: look at TAGs to determine differences
- Pull pytorch image from Dockerhub and run using singularity:
  - ```
    singularity pull \
        docker://pytorch/pytorch:1.5-cuda10.1-cudnn7-runtime
    ```
  - Creates new container image at `pytorch_1.5-cuda10.1-cudnn7-runtime.sif`
    ```

Now what?

- A container, **in singularity**, is a single file
- You need singularity installed on the host that you intend to run it on
- All of your programs are **inside** the container
- So to run the containerized application, you need to
  - ```
    singularity exec \
        pytorch_1.5-cuda10.1-cudnn7-runtime.sif \
        python [args…]
    ```
- Can also bring up a shell inside the container with
  - ```
    singularity shell \
        pytorch_1.5-cuda10.1-cudnn7-runtime.sif
    ```

Where are my files?

- By default, only the working directory is ‘mounted’ into the container
- Need to add ‘bind mounts’ to singularity command
- **`singularity shell \`**

  `pytorch_1.5-cuda10.1-cudnn7-runtime.sif`

  ```
  Singularity> ls /gpfs
  ls: cannot access '/gpfs': No such file or directory
  ```

- **`singularity shell -B /gpfs \`**

  `pytorch_1.5-cuda10.1-cudnn7-runtime.sif`

  ```
  Singularity> ls /gpfs
  automountdir  slac
  ```

Run pytorch container with GPU

```
$ singularity exec pytorch_1.5-cuda10.1-cudnn7-runtime.sif python
Python 3.7.7 (default, Mar 23 2020, 22:36:06)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print('Using device:', torch.device('cuda' if torch.cuda.is_available() else 'cpu'))
Using device: cpu
```

```
$ singularity exec --nv pytorch_1.5-cuda10.1-cudnn7-runtime.sif python
Python 3.7.7 (default, Mar 23 2020, 22:36:06)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print('Using device:', torch.device('cuda' if torch.cuda.is_available() else 'cpu'))
Using device: cuda
```

But I need uproot!

- … or any other library or module or binary
- You could create a singularity ‘sandbox’ or --writeable and add it all manually, but…
  - You would be left with an non-reproducible container image
  - You’ll have to manually find ways to version control etc.
- Contribute back!
  - Build new docker image with changes
  - Push new container into dockerhub (with tags and metadata)
  - Pull image back as singularity image to run
- Why not use Singularity directly? (ie write Singularity recipe)

Create a Dockerfile with additions

- $ **cat Dockerfile**

  ```
  FROM pytorch/pytorch:1.5-cuda10.1-cudnn7-runtime
  RUN pip install uproot
  ```

- Build it with docker
  - $ sudo **docker build . -t slaclab/pytorch-test:1.0**
- Push it to dockerhub (need account + docker login)
  - $ **docker push slaclab/pytorch-test:1.0**
- Pull new image with singularity
  - $ **singularity pull docker://slaclab/pytorch-test:1.0**
- Run
  - $ **singularity exec pytorch-test_latest.sif python**

    ```
    ...
    >>> import uproot
    >>>
    ```

Limitations

- Building can take a while...
  - But ultimately worth it as a means to
    - Document installation methods (Dockerfile) - git commit!
    - Have multiple versions as things evolve
    - Build once, run anywhere
  - Pushing and Pulling images network limited
    - Can minimise size through intermediate images etc.
- Compiling GPU code requires GPUs
  - requires GPU node + sudo + nvidia-docker

sudo?

- Use TravisCI (or other) to auto build and push docker images
- podman - still early days, little gpu support

Useful Docker images

- Base Images
  - https://hub.docker.com/r/continuumio/miniconda/
  - https://hub.docker.com/_/python
  - https://hub.docker.com/r/tensorflow/tensorflow
  - https://hub.docker.com/r/pytorch/pytorch
  - https://hub.docker.com/r/nvidia/cuda
  - https://hub.docker.com/_/centos
  - https://hub.docker.com/_/ubuntu
- Ubuntu images generally easier, more up-to-date
- Try to avoid alpine for scientific (musl instead of glibc)

Writing Dockerfiles

- Each RUN, COPY, RUN command creates a new ‘layer’
  - Keeps a snapshot of all changes caused (eg on FS) by that command
  - More layers means a larger container
  - Building containers etc. will
    - Leave behind gcc, make, etc
    - yum/apt cache files and other temp files
  - Ensure you delete at the end of the RUN command
  - Can use multi-stage builds to COPY files from one image to another
- Copy sensitive files:
  - Use BuildKit:
    - Add to Dockerfile

      ```
      # syntax=docker/dockerfile:experimental
      RUN --mount=type=secret,id=license_id
      ```

    - ```
      sudo DOCKER_BUILDKIT=1 docker build \
      --secret id=license_id,src=./license_id.txt .
      ```

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
      ```

Writing Dockerfiles

- Each RUN, COPY, RUN command creates a new ‘layer’
  - Keeps a snapshot of all changes caused (eg on FS) by that command
  - More layers means a larger container
  - Building containers etc. will
    - Leave behind gcc, make, etc
    - yum/apt cache files and other temp files
  - Ensure you delete at the end of the RUN command
  - Can use multi-stage builds to COPY files from one image to another
- Copy sensitive files:
  - Use BuildKit:
    - Add to Dockerfile

      ```
      # syntax=docker/dockerfile:experimental
      RUN --mount=type=secret,id=license_id
      ```

    - ```
      sudo DOCKER_BUILDKIT=1 docker build \
      --secret id=license_id,src=./license_id.txt .
      ```

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
      ```

Let’s submit a job

- Slurm will replace LSF on SDF
- All GPUs in slurm
- Use a GPFS space (for now) for your working directory
- AFS (Home) not readily available (no tokens)

Slurm - example submission script

```
#!/bin/bash
#SBATCH --account=shared --partition=shared
#
#SBATCH --job-name=test
#SBATCH --output=output-%j.txt --error=output-%j.txt
#
#SBATCH --nodes=1 --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=1g
#SBATCH --time=10:00
#
#SBATCH --gpus=1

# cd /gpfs/slac/cryo/fs1/u/ytl/containers/
singularity exec --nv -B /gpfs pytorch-test_1.0.sif \
     python test.py
```

Whilst we're testing...:

```
$ ssh slacgpu.slac.stanford.edu
$ module load slurm
```

Submit with

```
$ sbatch <script_path>
```

Slurm - common commands

`sbatch`	Submit batch job
`squeue`	Look at the job list
`scancel <job id>`	Cancel job id
`scontrol show job`	Show detailed job information
`sstat`	Show per time step job usage
`scontrol update job`	Update job details
`srun`	Connect to an allocated job

Slurm - requesting GPUs

		Available	GPU Memory	FP32 (TFLOPS)	FP64 (TFLOPS)
`--gpus=1`	Any available	~325			
`--gpus=geforce_gtx_1080_ti:1`	Geforce 1080ti	~100	11GB	~11.3	~0.3
`--gpus=geforce_rtx_2080_ti:1`	Geforce 2080ti	~200	11GB	~13.4	~0.4
`--gpus=v100:1`	Tesla v100	~24	32GB	~15.7	~7.8

Summary

- SDF will be our new platform for Scientific Compute and Storage
  - Feedback and suggestions please!
- Containers are a great way of packaging programs
  - Shareable and repeatable (spend less time getting things working)
  - Build once, run anywhere (version control!)
  - Small additional prefix to run
- Still a little early...
  - sudo required
  - Network limited (some containers can be 10GB+)
  - GPU support (especially during building) under development
- Submit to SLURM