# Containers and Spack

John LaGrone
HPC Applications Scientist

February 23, 2022

Research and Data Sciences Services
Office of Information Technology
Center for Research Computing
Southern Methodist University

# Outline

# Research Support

# Research and Data Science Services

| Domain | Name | Email |
|---|---|---|
| Data Science | Dr. Eric Godat | egodat@smu.edu |
| High-Performance Computing | Dr. Robert Kalescky | rkalescky@smu.edu |
| | Dr. John LaGrone | jlagrone@smu.edu |
| Machine Learning & Artificial Intelligence | Dr. Tue Vu | tuev@smu.edu |
| Custom Devices (IOT, wearables, etc.) | Guillermo Vasquez | guillermov@smu.edu |

**Table 1:** The OIT Research and Data Science Services team provides research computing support, consultations, and collaborations.

- Maintains our primary shared resource for research computing, ManeFrame II (M2), in collaboration with OIT
- Provides research computing tools, support, and training to all faculty, staff, and students using research computing resources
- `www.smu.edu/crc` has documentation and news
- help@smu.edu or rkalescky@smu.edu or jlagrone@smu.edu for help
- Request an account at `www.smu.edu/crc`

# Spring 2022 CRC HPC Workshop Series

| Date | Time | Workshop |
|---|---|---|
| February 2 | 2-4 | ManeFrame II (M2) Introduction |
| February 9 | 2-4 | Workflows in R |
| February 15 | 3-5 | Finding and Preparing Text Data Sets for Mining |
| Febryary 16 | 1-4 | Machine Learning with Python Part 1 |
| February 17 | 12-1 | AI for the Non-Expert |
| February 18 | 12-1 | Introduction to GitHub |
| February 23 | 2-4 | Containers and Spack |
| March 2 | 2-4 | ManeFrame II (M2) Introduction |
| March 3 | 1-4 | Data Science Workflow with R |
| March 8 | 3-6 | Introduction to Python for Text Mining |
| March 9 | 1-4 | Machine Learning with Python Part 2 |
| March 22 | 3-6 | Getting Support for Text Mining |
| March 23 | 2-4 | Shared Memory Parallelism |
| March 30 | 1-4 | Deep Learning with Python Part 1 |
| April 6 | 2-4 | ManeFrame II (M2) Introduction |
| April 13 | 2-4 | Accelerator Libraries and APIs |
| April 20 | 1-4 | Deep Learning with Python Part 2 |
| April 27 | 2-4 | MPI/NCCL/SHMem |
| May 4 | 2-4 | ManeFrame II (M2) Introduction |

- Workshops will be held each Wednesday from 2:00 to 4:00 PM.
- Sessions will typically be recorded and posted along with session materials.
- Register on the Library Workshop Calendar https://libcal.smu.edu/calendar/libraryworkshops.

# Containers


LSCU 107737 9
JP 4300
MAXGW 67200LB 30480KG
TARE 8510LB 3860KG
MAXCW 58690LB 26620KG
CU.CAP 2389CU.FT 67.7CU.M

# Before Containerization

- Goods had to be loaded and unloaded individually
- Inefficient - it was not uncommon to spend more time loading and loading goods than transporting them
- Insecure - goods had be handled by many people, increasing the chance for loss and theft
- Inaccessible - Long distance shipping only available to the wealthy

# After Containerization

- Standardized - containers are all the same size and weight allowances
- Efficient - containers are easy to load and unload and transfer to other modes of transportation
- Secure - goods may be secured in containers from source to final destination
- Available - cost effective to ship goods across the world

# What about computing?

- It's common to run on multiple systems with different requirements
- We would like to avoid installing the same sets of software again and again
- We would like other people to run our software without our help
- We would like to preserve a known configuration that our software works in

# Containers Overview

- Containers offer the ability to run fully customized software stacks, e.g. based on different Linux distributions and versions
- Containers are not virtual machines, where an entire hardware platform is virtualized, rather containers share a common kernel and access to physical hardware resources
- Docker is a popular container platform in development and server environments
- Singularity is a popular container platform in HPC environments
    - Docker is not directly support on ManeFrame II
    - Singularity can consume and run Docker containers

# Container Benefits

- **Performance**: containers can perform at near native performance
- **Flexibility**: install (almost) any software you need
- **Reproducibility**: create complex software environments that are easy to manage and are verifiable.
- **Compatibility**: built on open standards that works on all major Linux distributions
- **Portability**: Build once and run (almost) anywhere

# Container Limitations

- **Architecture dependent**: Containers are (currently) limited to the same CPU architecture (x86_64, ARM, etc.) and binary formats
- **Portability**: Requires glibc and kernel compatibility between host and container. Other kernel level APIs may also need to be compatible (e.g. CUDA/GPU drivers, network drivers, etc.)
- **Filesystem Isolation**: paths are (usually) different when viewed from inside or outside of a container

# Docker and HPC

- We don’t allow direct Docker use on M2
- Docker’s security model is designed to support users “trusted” users running “trusted” containers (e.g. users who can escalate to root access)
- Docker is not designed to support scripted / batch based workflows
- Docker is not designed to support parallel applications

# Singularity Features

- Containers are a single image file
- No root owned daemon processes
- User inside containers are the same as users outside the container (no contextual changes)
- Supports shared, multi-user environments
- Supports HPC hardware such as GPUs and Infiniband networks
- Supports HPC applications like MPI

# Common Use Cases

- Converting Docker containers to Singularity
- Building and running software that require newer systems and libraries
- Running commercial software binaries that have specific requirements

# Singularity Workflow

- Build your Singularity containers on a local system you have root or sudo access. Alternatively build a Docker container
- Transfer your container to M2 or other HPC system. If you used Docker, you will need to convert the image
- Run your Singularity containers

# Singularity Options for Mac and Windows

- Install VirtualBox on your computer
- Create an Ubuntu or Red Hat based virtual machine
- Install Singularity on the virtual machine. (Note: install the same version the HPC system uses.)

# MPI and GPU Accelerated Containers

- Generally you should try to use the same MPI distribution and Infiniband drivers inside and outside the container
- You should use the same GPU drivers and libraries inside and outside the container
- You can sometimes mount system GPU settings with the –*nv* option
- **Ask us for help!** Configuration settings are often subtle and may not be obvious from the user environments

# Singularity Commands

singularity [options] <subcommand>[subcommand options] ...

The main subcommands you should know are

- **build**: Build your container from a definition file, download an existing container, or convert a container from one format to another (Docker to Singularity)
- **shell**: Spawn an interactive shell session inside your container
- **exec**: Run an arbitrary command inside your container

# Singularity Build Script Example

```
Bootstrap: docker                       # Base container source
From: ubuntu:18.04                      # Base container version

%post                                   # Changes to container
export DEBIAN_FRONTEND=noninteractive
apt-get update
apt-get -y install\
 python3-pip\
 python3-numpy\
 python3-pandas
pip3 install\
 jupyterlab

%runscript                              # Default container command
python3

```

**Listing 1:** Example Singularity build file that uses Ubuntu 18.04 and Python3 with package installation via `apt` and `pip`.

# Building Singularity Container Images

```
#!/usr/bin/env sh

module purge
module load singularity/3.5.3

srun -p development -x k001 -c 1 --mem=1G\
 --pty singularity build\
 --fakeroot python3.sif python3.singularity

```

**Listing 2:** Steps to build a Singularity container image. Note that building Singularity container images from build scripts on M2 requires permission.

# Docker Build Script Example

```
FROM ubuntu:18.04

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update &&\
 apt-get -y install\
 python3-pip\
 python3-numpy\
 python3-pandas

RUN pip3 install\
 jupyterlab

ENTRYPOINT ["python3"]
```

**Listing 3:** Example Dockerfile that uses Ubuntu 18.04 and Python3 with package installation via `apt` and `pip`.

# Building and Converting Docker Container Images

```sh
#!/usr/bin/env sh

docker build -t python3:18.04 -f python3.dockerfile .
docker save -o python3_18_04.tar python3:18.04
scp python3_18_04.tar m2.smu.edu:~/workshops/examples/

```

**Listing 4:** Building the Docker container off of M2, exporting the container image, and uploading to M2.

```sh
#!/usr/bin/env sh

module purge
module load singularity/3.5.3.lua

srun -p development -x k001 -c 1 --mem=1G --pty singularity build python3_18_04.sif docker-archive://python3_18_04.tar

```

**Listing 5:** Converting the uploaded Docker container image to a Singularity container image.

# Spack

# What is Spack?

- Spack Is a package management tool
- Spack is designed for complex and multi-user environments
- Spack is non-destructive. Installing new versions of packages will not break existing installs
- Spack helps ensure reproducible software installs

```sh
#!/usr/bin/env sh

module purge
module load python/3

cd $WORK
git clone -c feature.manyFiles=true https://github.com/spack/spack.git

# activate spack
source $WORK/spack/share/spack/setup-env.sh

# For tcsh/csh
# source spack/share/spack/setup-env.csh

```

Listing 6: Example of installing Spack.

*NOTE: we have Spack installed as a module on M2, this is mostly to make it easy for users to load software that we have installed using Spack. If you try to install software with the Spack module, you may encounter permission issues, so it is advisable to install

It's a good idea to start with one or more of the compilers already installed on the system.

To add one of these compilers:

```
#!/usr/bin/env sh

module purge

# load a compiler
module load gcc-9.2

# tell spack to find new compilers
spack compiler find
```

**Listing 7:** Example of adding a compiler to Spack.

If you are adding an Intel compiler, you will need to modify the Spack compiler settings to load the compiler module for both Intel and GCC. This can be done with: (hint: this defaults to VIM for editing, you can change that with *export EDITOR=nano*)

*spack config edit compilers*

The result should look like:

```

intel@2021.1:
        paths:
                cc =
                ↪  /hpc/applications/intel/oneapi/compiler/2021.1.1/linux/bin/intel64/ic
                cxx =
                ↪  /hpc/applications/intel/oneapi/compiler/2021.1.1/linux/bin/intel64/ic
                f77 =
                ↪  /hpc/applications/intel/oneapi/compiler/2021.1.1/linux/bin/intel64/if
                fc =
```

# Using Spack

- **spack list <package >** List all of the packages Spack can install and optionally search for the specified package.
- **spack info <package >** List information about the specified package including options and versions available.
- **spack install <package >** Install the specified package. It’s a good idea to add the “–reuse” flag to avoid installing duplicate dependencies.
- **spack find <package >** List all of the packages Spack has installed and optionally search for the specified package.
- **spack load <package >** Load the specified package.*

* Packages should also now be available using the module system on M2.

# Using Spack

- **spack install –reuse package@2.1.3** Install version 2.1.3 of the requested package
- **spack install –reuse package@2.1.3 %gcc@11.2.0** Install version 2.1.3 of the requested package using gcc version 11.2.0 as the compiler (note: you need this compiler installed already)
- **spack install –reuse package@2.1.3 %gcc@11.2.0 +option1 +option2** Install version 2.1.3 of the requested package using gcc version 11.2.0 as the compiler with option 1 and option 2 enabled

*NOTE: if you frequently switch compilers, be mindful that "–reuse" may not use packages with a consistent compiler. This is a new feature that is continuing to improve.

# Spack Enviroments

- You can use Spack environments to create isolated software stacks
- Packages installed in environments are not available outside that environment
- Good for projects because you can see exactly which versions of libraries you are using
- Can result in lots of duplicate installs

# Spack Enviroments

- **spack env create myenv** Create a new environment called “myenv”
- **spack env list** List available environments
- **spack env activate myenv** Activate the environment called “myenv”
- **spack env status** See which environments are active
- **spack concretize** “lock” the versions of packages installed in the current environment so everything stays consistent. You can also use the output to build the same environment on a different system.
- **despacktivate** Deactivae the current environment

# Spack Problems

- Spack is relatively new and improving at a fast pace. Check their github for new features and fixes
- Sometimes Spack installs things out of order (or packages have circular dependencies). Try installing packages where installs fail one at a time.
- Ask us for help!

# Help?

Need help or have questions?

rkalescky@smu.edu

jlagrone@smu.edu

help@smu.edu (include HPC in the subject line)