Containerizing All the Things: Using Containers for Research

Robert Kalescky
HPC Research Scientist
Adjunct Professor of Data Science

Research Technology Services
Office of Information Technology
Southern Methodist University

Outline

Research Support

Containers

Docker

Apptainer/Singularity

Spack

Kubernetes

Research Support

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 Technology Services team provides research computing support, consultations, and collaborations.

General Support

- Provides research computing tools, support, and training to all faculty, staff, and students using research computing resources
- help@smu.edu with [HPC] as part of the subject line
- Documentation and account requests at `https://southernmethodistuniversity.github.io/hpc_docs/`

Containers


LSCU 107737 9
JP 4300
MAX.GW 67200LB 30480KG
TARE 8510LB 3860KG
MAX.CW 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

Common Issues with Software Stacks

- 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?

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

Possible 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).
- Entire ecosystem has grown around containers including open standards and governance.

Possible Solution: Containers

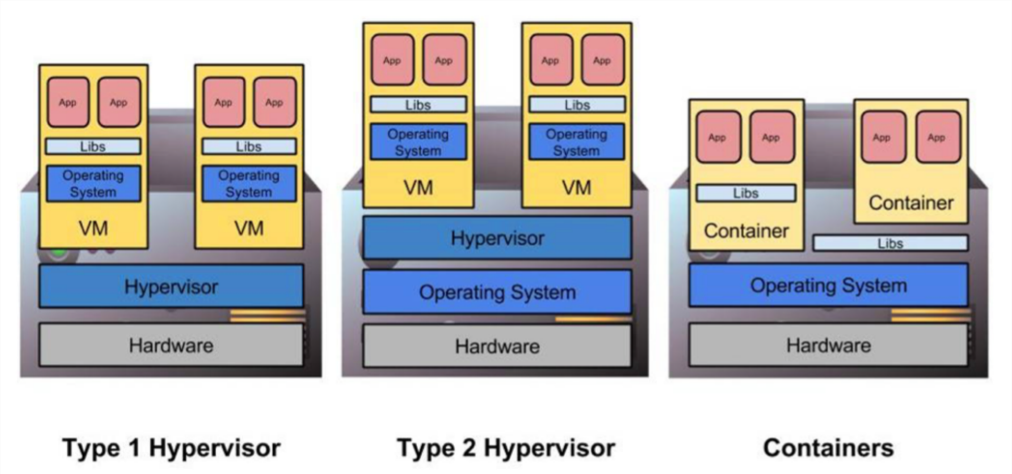
- A lightweight collection of executable software that encapsulates everything needed to run an application
  - Minus the OS kernel
  - Based on Linux only
- Processes and all user-level software is isolated
- Creates a portable* software ecosystem
- Think `chroot` on steroids
- Docker is the most common tool today
  - Available on all major platforms
  - Widely used in industry
  - Integrated container registry via Dockerhub

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

Hypervisors and Containers

- Type 1 hypervisors insert layer below host OS
- Type 2 hypervisors work as or within the host OS
- Containers do not abstract hardware, instead provide “enhanced chroot” to create isolated environment using a common kernel
- Location of abstraction can have impact on performance
- All enable custom software stacks on existing hardware


App
App
Libs
Operating System
VM
App
App
Libs
Operating System
VM
Hypervisor
Hardware
App
App
Libs
Operating System
VM
App
App
Libs
Operating System
VM
Hypervisor
Operating System
Hardware
App
App
Libs
Container
App
App
Container
Libs
Operating System
Hardware
Type 1 Hypervisor
Type 2 Hypervisor
Containers

Container Benefits

Performant Containers can perform at near native performance.

Flexible Install (almost) any software you need.

Reproducible Define complex software environments that are verifiable.

Compatible Built on open standards that works on all major Linux distributions.

Portable Build once and run (almost) anywhere.

Container Limitations

Hardware Containers are limited to the same CPU architecture (x86_64, ARM, Power, etc.) and binary formats

Software 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 Paths can be different when viewed from inside or outside of a container

Nomenclature

Image A read-only template that defines how to create a container

Container An instantiation of an image, a running instance

Container Runtime Tool or service to execute and manage containers

Registry A service that is used to store and distribute images

Docker and HPC

- We don’t allow direct Docker use on SMU HPC systems
- 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

Apptainer/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 Apptainer/Singularity
- Building and running software that require newer systems and libraries
- Running commercial software binaries that have specific requirements

Docker

- Is a “recipe” for how to construct an image.
- Starts `FROM` a defined base image.
- Several basic commands (`ADD`, `COPY`, `RUN`, etc.) can be applied to mutate the image to the desired state.
- Metadata labels can also be added to provide information about the image.
- In addition to the file system changes, the Dockerfile can also control settings like the environment, the starting directory, and default commands.

Basic Dockerfile

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND noninteractive

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

RUN pip3 install\
 jupyterlab

ENTRYPOINT ["python3"]

```

Docker Image Registries

- There are several public and private sources for Docker images.
- Images can be used as the base image for custom images.
- Already optimized images can help with reproducible and efficient development workflows.

Docker Image Registries

Docker	`https://hub.docker.com`
GitHub Packges	`https://github.com/features/packages`
Quay.io	`https://quay.io`
NVIDIA	`https://catalog.ngc.nvidia.com`
Intel	`https://hub.docker.com/u/intel`
AMD	`https://hub.docker.com/u/amdih`

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

pull_and_check() {
  name=${1}
  tag=`echo ${name} | cut -d':' -f2`
  docker pull ${name}
  docker image ls | egrep "REPOSITORY|${tag}"
  docker run --rm -it ${name} bash
}

images[0]="ubuntu:jammy"
images[1]="nvcr.io/nvidia/nvhpc:22.3-devel-cuda_multi-ubuntu20.04"
images[2]="nvcr.io/nvidia/nvhpc:20.7-runtime-cuda10.1-centos7"

for image in ${images[@]}; do
  pull_and_check ${image}
done

```

- Images are CPU-architecture specific
- Docker supports multi-architecture builds
  - Platforms: amd64, arm32v5, arm32v6, arm32v7, arm64v8, i386, ppc64le, and s390x
  - `docker build --platform` with single platform
  - `docker buildx --platform` with list of platforms
- Builds on non-native platforms will be slower as it is running via emulation

Docker Multi-Architecture Builds

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

# Create builder to build images
docker buildx create --name builder --use

# Build images for x86_64 and ARM64
docker buildx build --no-cache --platform\
 linux/amd64,linux/arm64 -t rkalescky/python3:latest\
 -f python3.dockerfile --push .

# Inspect the built images
docker buildx imagetools inspect rkalescky/python3:latest
```

Multi-Stage Builds with Docker

- Images with build tools can be very large.
- Use the needed image for building.
- Use the smallest image for running.
- Define both the build and execution in a single Dockerfile.

Basic Multi-Stage Dockerfile

```
FROM nvcr.io/nvidia/nvhpc:22.3-devel-cuda_multi-ubuntu20.04 as builder
WORKDIR /build
COPY hello_world.cpp ./
RUN nvc++ -Bstatic -o hello_world hello_world.cpp

FROM alpine:3.15.4
WORKDIR /opt/hello/bin
COPY --from=builder /build/hello_world ./
ENTRYPOINT ["/opt/hello/bin/hello_world"]
```

Docker Multi-Architecture Builds

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

# Build image using multi-stage Dockerfile
docker build -t hello:20.04 -f hello_world.dockerfile .

# Run the image
docker run hello:20.04

# Note the size difference
docker image ls | egrep "hello|22.3-devel"

```

Apptainer/Singularity

Building Apptainer/Singularity Images

- Singularity has it's own image definition language.
  - Requires (re)writing the definition file.
  - Requires root or "fakeroot", which is not widely available on HPC systems.
  - Can be done on a Linux system with Singularity installed and them copying the image.
  - Not generally recommended as there would be two definition files to maintain, presumably Docker and also Singularity.
- Pull from Docker registries.
  - Requires pushing and pulling of Docker images.
- Build from Docker archives.
  - Requires exporting, copying, and conversion of Docker images.

Pulling Docker Containers

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

# Build images for both x86_64 and ARM64
. ./docker_buildx.sh

# Consume the image on M3 via Singularity
ssh m3 'bash -l -c "module load apptainer\
 && apptainer run docker://rkalescky/python3 -c \"import numpy as np;
↪  print(np.pi)\""'
```

Pulling Docker Containers

```
# Pull Docker image to a Singularity image
ssh m3 'bash -l -c "module load apptainer\
 && apptainer pull -F python3_3.9.13-slim.sif docker://python:3.9.13-slim\
 && ls -lh ./python3_3.9.13-slim.sif\
 && apptainer exec ./python3_3.9.13-slim.sif python3 -c \"import sys;
 ↪ print(sys.version)\""'

# Singularity mount points
ssh m3 'bash -l -c "module load apptainer\
 && echo \$APPTAINER_BIND"'

```

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 the HPC system. If you used Docker, you will need to convert the image
- Run your Singularity containers

Converting from Docker Archives

```
# Export, upload, convert, and run on M3 via Singularity
docker save python3:20.04 | ssh m3 'bash -l -c "n=python3_20_04\
 && cat > ~/$n.tar\
 && module load apptainer\
 && apptainer build -F $n.sif docker-archive:$HOME/$n.tar\
 && apptainer shell $n.sif'
```

Full Example Workflow

- Build application, separating build and deployment containers
- Script container image build and make Lmod module file for ease of use
- Convert Docker image to Apptainer/Singularity image using the docker2singularity tool

Full Example Workflow

```
# docker build -t smuresearch/molden:latest .
# docker run -it smuresearch/molden:latest

FROM ubuntu:22.04 AS build

# Tarball available at https://www.theochem.ru.nl/molden/howtoget.html
ENV MOLDEN_VERSION=6.9
ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update &&\
 apt-get install -y\
 build-essential\
 gfortran\
 libx11-dev\
 libglu1-mesa-dev\
 freeglut3-dev\
 xutils-dev\
 vim
```

Full Example Workflow

```
COPY molden${MOLDEN_VERSION}.tar.gz /

# `makefile` Edits
# 1. -fallow-argument-mismatch to allow for type mismatches
# 2. Add quotes due to above flag.
# 3. Don't do desktop integration steps
RUN tar -xf molden${MOLDEN_VERSION}.tar.gz &&\
 cd molden${MOLDEN_VERSION} &&\
 sed -i '/FC = gfortran/{s/$/ -fallow-argument-mismatch/}' makefile &&\
 sed -i 's/FC=\${FC}/FC="\${FC}"/g' makefile &&\
 sed -i 's/\$(EXTENZ\{0,1\})//g' makefile &&\
 cat makefile &&\
 make -j all
```

Full Example Workflow

```
FROM ubuntu:22.04

ENV MOLDEN_VERSION=6.9
ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update &&\
 apt-get install -y\
 libgfortran5\
 libx11-6\
 libglu1-mesa\
 freeglut3 &&\
 apt-get clean &&\
 rm -rf /var/lib/apt/lists/*

COPY --from=build /molden${MOLDEN_VERSION}/bin /usr/local/bin/
```

Full Example Workflow

```zsh
#!/usr/bin/env zsh

# Project-specific variables
name="molden"
version="6.9"

# Stop on failure
set -e -o pipefail

# Set log file
exec > >(tee build.log) 2>&1
```

Full Example Workflow

```bash
# Set platform
case $1 in
native)
    platform=$(uname -m)
    ;;
*)
    platform="amd64"
    ;;
esac
```

Full Example Workflow

```bash
# Singularity image name
img="${name}_${version}_${platform}_$(date "+%Y_%m_%d_%H_%M_%S").sif"

# Build container with Docker
docker build --no-cache --progress=plain\
 --platform linux/${platform} -t ${name}:${version} .
```

Full Example Workflow

```
# Convert Docker image to Singularity image
docker run -v /var/run/docker.sock:/var/run/docker.sock\
 -v $PWD:/output --privileged -t --rm singularityware/docker2singularity\
 -n ${img} ${name}:${version}
mv ${img%.sif}.simg ${img}
```

Full Example Workflow

```
# Change Singularity image permissions
if [[ $(uname -s) == "Linux" ]]; then
  sudo chown $USER:$USER $img
fi

# Update module file with new Singularity image name
sed -i'' -e "s/^local img_name.*/local img_name      = \'${img}\'/g"\
 ${name}.lua
```

Full Example Workflow

```
load("apptainer")

local img_name      = 'molden_6.9_amd64_2024_04_02_14_31_08.sif'
local img_directory = '/hpc/m3/containers/'
local img_path      = pathJoin(img_directory, img_name)
```

Full Example Workflow

```lua
function build_command(cmd)
  local cmd_beginning = 'apptainer exec '
  local cmd_ending    = img_path .. ' ' .. cmd
  local sh_ending     = ' "$@"'
  local csh_ending    = ' $*'
  local sh_cmd        = cmd_beginning .. cmd_ending .. sh_ending
  local csh_cmd       = cmd_beginning .. cmd_ending .. csh_ending
  set_shell_function(cmd, sh_cmd, csh_cmd)
end
```

Full Example Workflow

```lua
local executables = {
  'ambfor',
  'ambmd',
  'gmolden',
  'gmolden.exe',
  'molden',
  'surf'
}

for _, executable in ipairs(executables) do
  build_command(executable)
end
```

Spack

Spack Containerize

- Build images defined by Spack environments.
- Spack-based build optimizations are preserved.
- Intermediate Dockerfile uses multi-stage builds
- Currently does not work for multi-architecture builds.

Define a Spack Environment

```yaml
spack:
  specs:
  - gromacs+mpi
  - mpich
  container:
    format: docker
    images:
      os: "ubuntu:20.04"
      spack: develop
```

Build the Image from the Environment

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

# Define the Spack environment
cat spack.yaml

# Build container definition file
spack containerize > Dockerfile

# Build the container image
docker build -t gromacs:latest .

```

Kubernetes

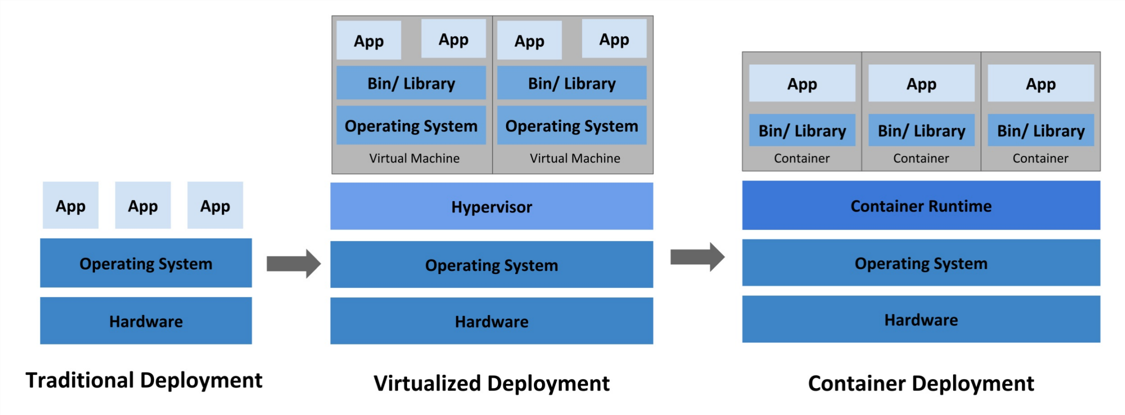
What is Kubernetes

- Platform for managing containerized workloads and services
  - Portable
  - Extensible
  - Open Source
  - Originally developed by Google for managing service deployment
- Kubernetes (K8s)
  - Kubernetes, from Greek, means helmsman or pilot
  - “K8s” comes from the 8 characters between the “k” and the “s”

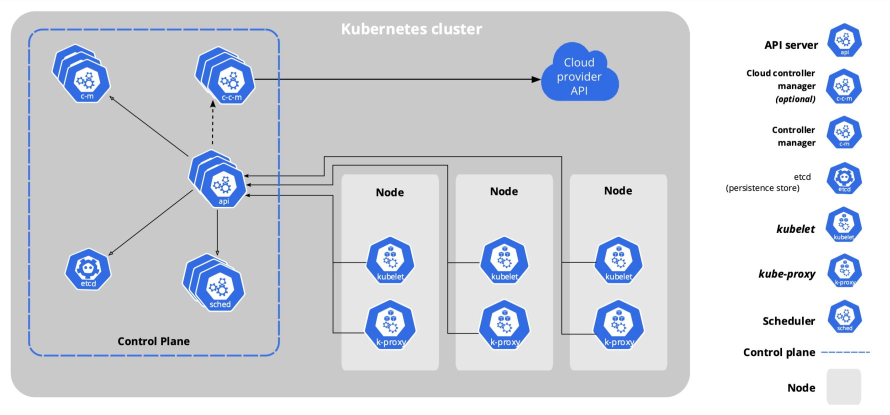
Benefits of Kubernetes

- Persistence
- Load balancing
- Self-healing
- Automated rollouts and rollbacks
- Resource optimization
- Infrastructure as code

Application Deployment Scenarios



K8s Cluster Components



Kubernetes Distributions

- kubeadm
- Minikube
  - Local K8s clusters on macOS, Linux, and Windows
  - Commonly used for developing applications
- K3s
- GKE (Google Kubernetes Engine)
- AKS (Microsoft Azure Kubernetes Services)
- EKS (Amazon Elastic Kubernetes Service)
- RKE (SUSE Rancher Kubernetes Engine)
- Ubuntu
  - Charmed Kubernetes
  - MicoK8s

Pods

- Job definition
  - One or more containers
  - Specification on how to run them
- Shared resources
  - Compute
  - Storage
  - Networking
  - Context
    - Location and scheduling

Types of Workloads

Deployments Manage changes to running Pods as specified rate

ReplicaSet Specify the number of available and identical Pods for high-availability

StatefulSets Deployment of Pods with a specific ordering, e.g. Pods with unique network IDs

DaemonSets Every node gets a Pod, which dynamically changes as the cluster size changes

Jobs Batch scheduling of workloads (similar to Slurm Arrays, but with redundancy)

Cronjobs Scheduled workloads

Pod Security

- Specify the Pod level of isolation
- Pod Security Standards
  - Privileged (allows for privilege escalation)
  - Baseline (prevents privilege escalation)
  - Restricted (hardened security)
    - Security-critical applications
    - Low-trust users
- Levels of enforcement
  - Enforce
  - Audit (log annotation)
  - Warn (user-facing warning)

Help?

Need help or have questions?

rkalescky@smu.edu

jlagrone@smu.edu

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