Introduction to Singularity: A Container Platform for Scientific and High-Performance Computing

Marty Kandes, Ph.D.

Computational & Data Science Research Specialist
High-Performance Computing User Services Group
San Diego Supercomputer Center
University of California, San Diego

SDSC Summer Institute 2021
Friday, August 6th, 2021
8:30AM - 9:00AM PDT

Today's Session

- What are containers?
- Why are containers useful?
- What is Singularity?
- How is Singularity different?
- Should you consider using Singularity?
- Q&A

What is Containerization?

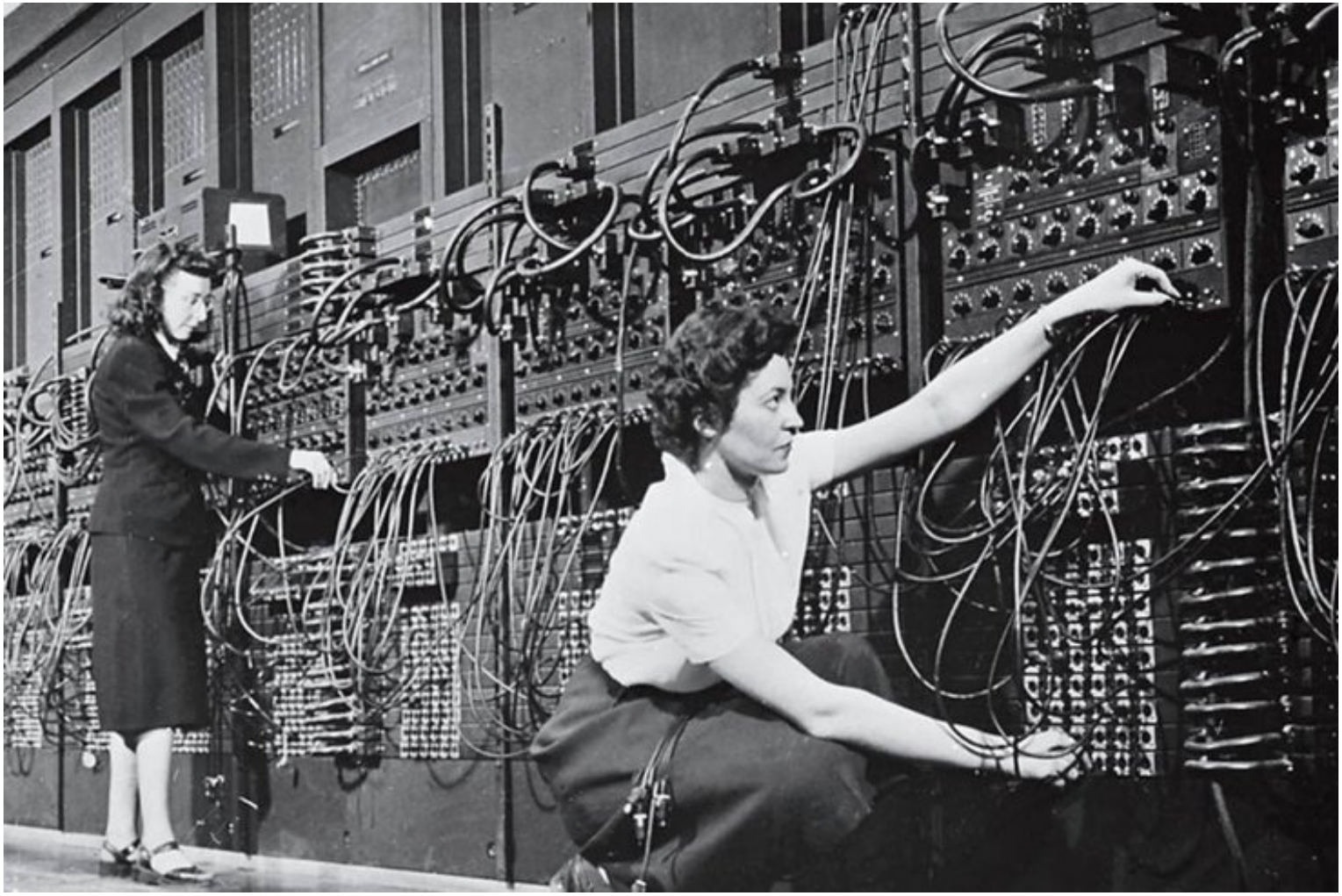
Before Containerization (Bulk Break Cargo)

- Bespoke - goods must be loaded and unloaded individually, many times by hand
- Inefficient - more time spent loading and unloading goods than transporting them
- Insecure - goods must be stored and handled by intermediaries during transport; potential loss and/or theft of goods
- Local - only luxury and specialty goods shipped long-distance

After Containerization (Intermodal Freight Transport)

- Standardized - containers of known dimensions and permissible weight tolerances
- Efficient - portable containers allow fast loading and unloading from multiple modes of transportation
- Secure - goods remained stored within the same container during transport
- Global - cost effective to ship almost any good from anywhere in the world

Evolution of Software Deployment



1945: Electronic Numerical Integrator and Computer (ENIAC)

Evolution of Software Deployment

1964: IBM System/360

Punch card

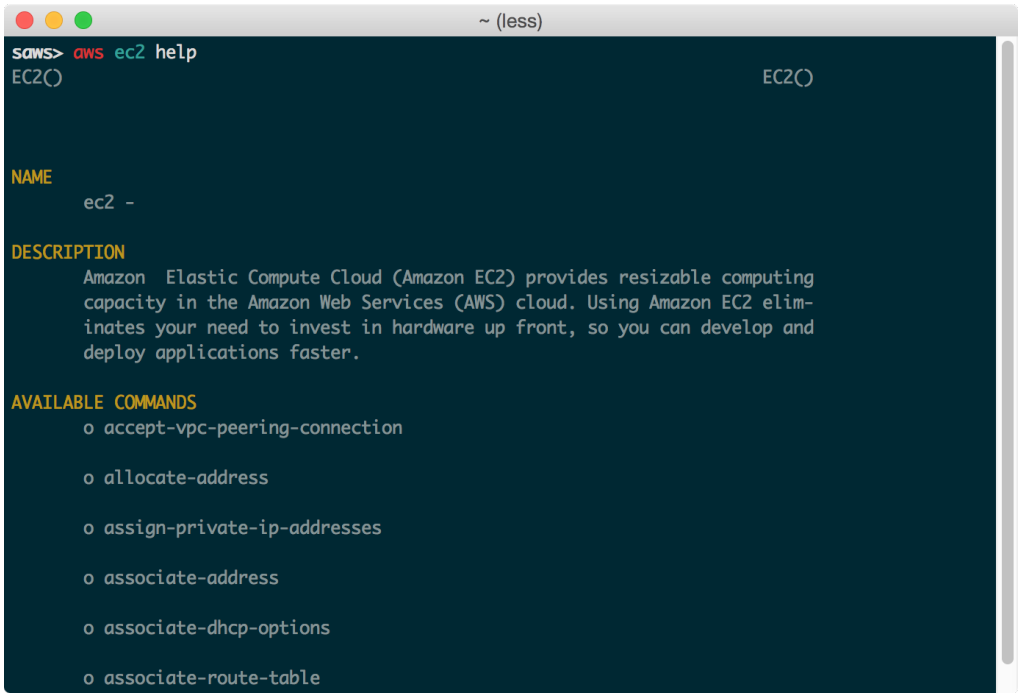
Evolution of Software Deployment

1998: Desktop PC

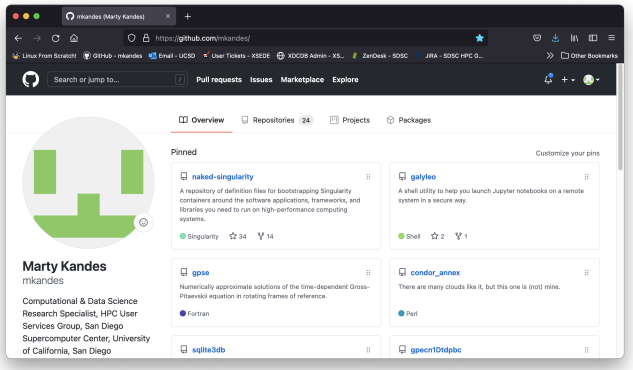
Floppy disk

CD-ROM

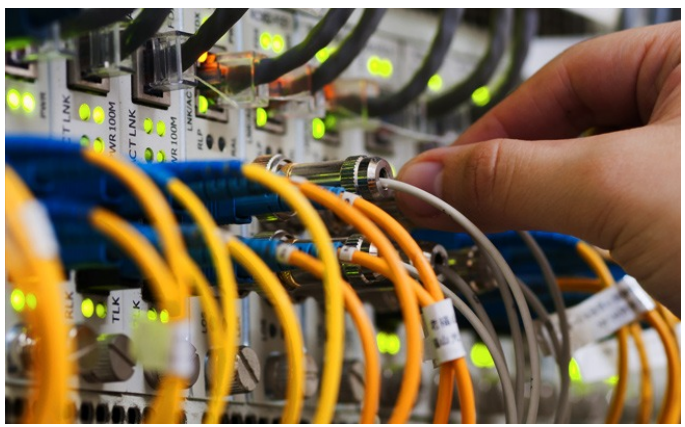
Evolution of Software Deployment



Today: Amazon Web Services

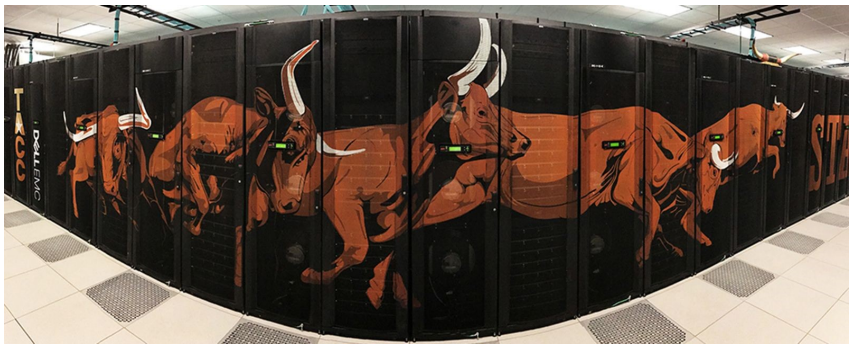


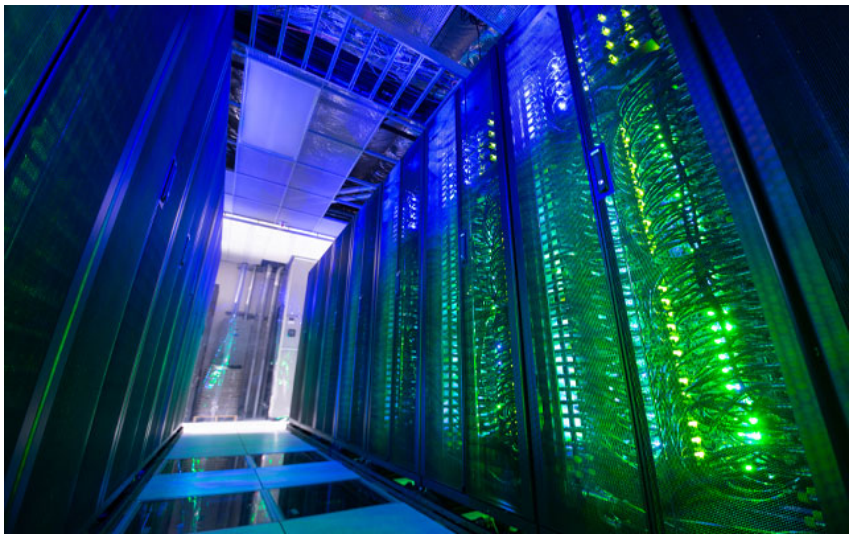
GitHub



Fiber-optics

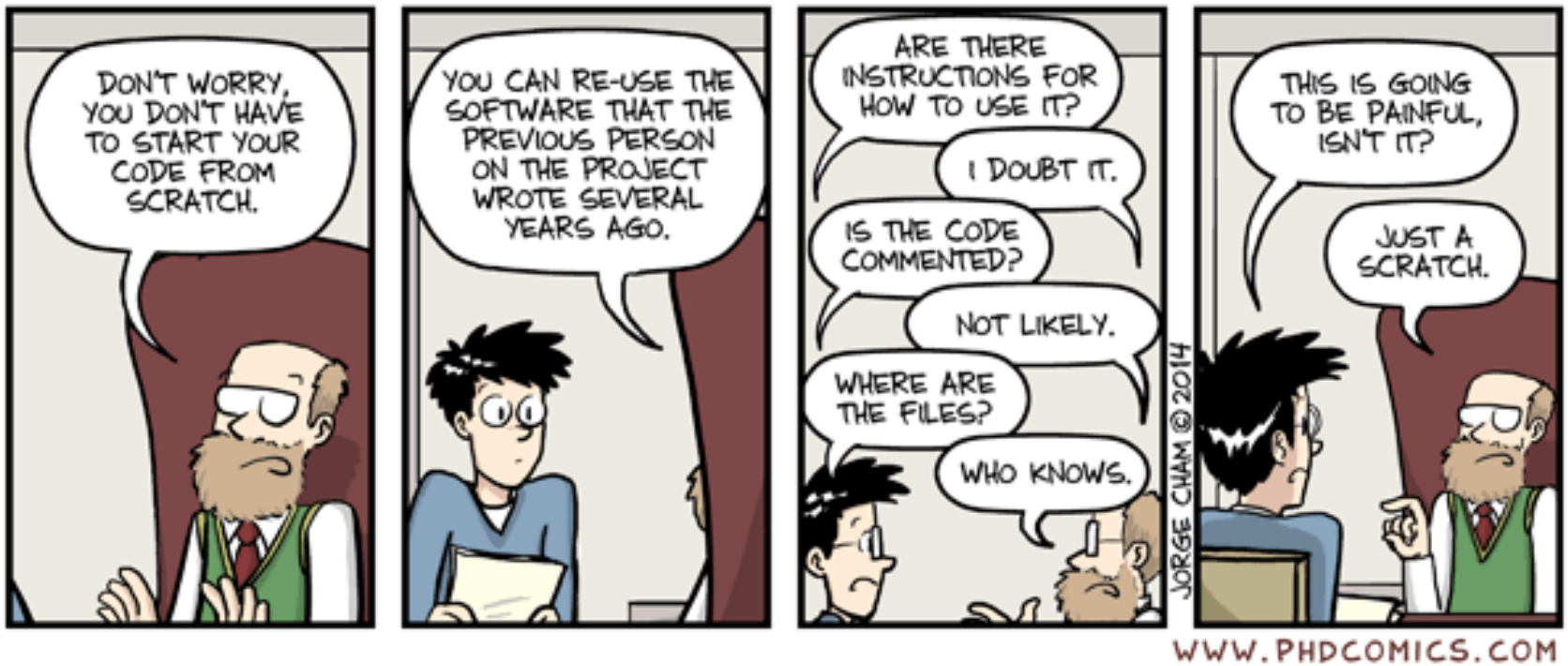
What about deploying my software to HPC systems?



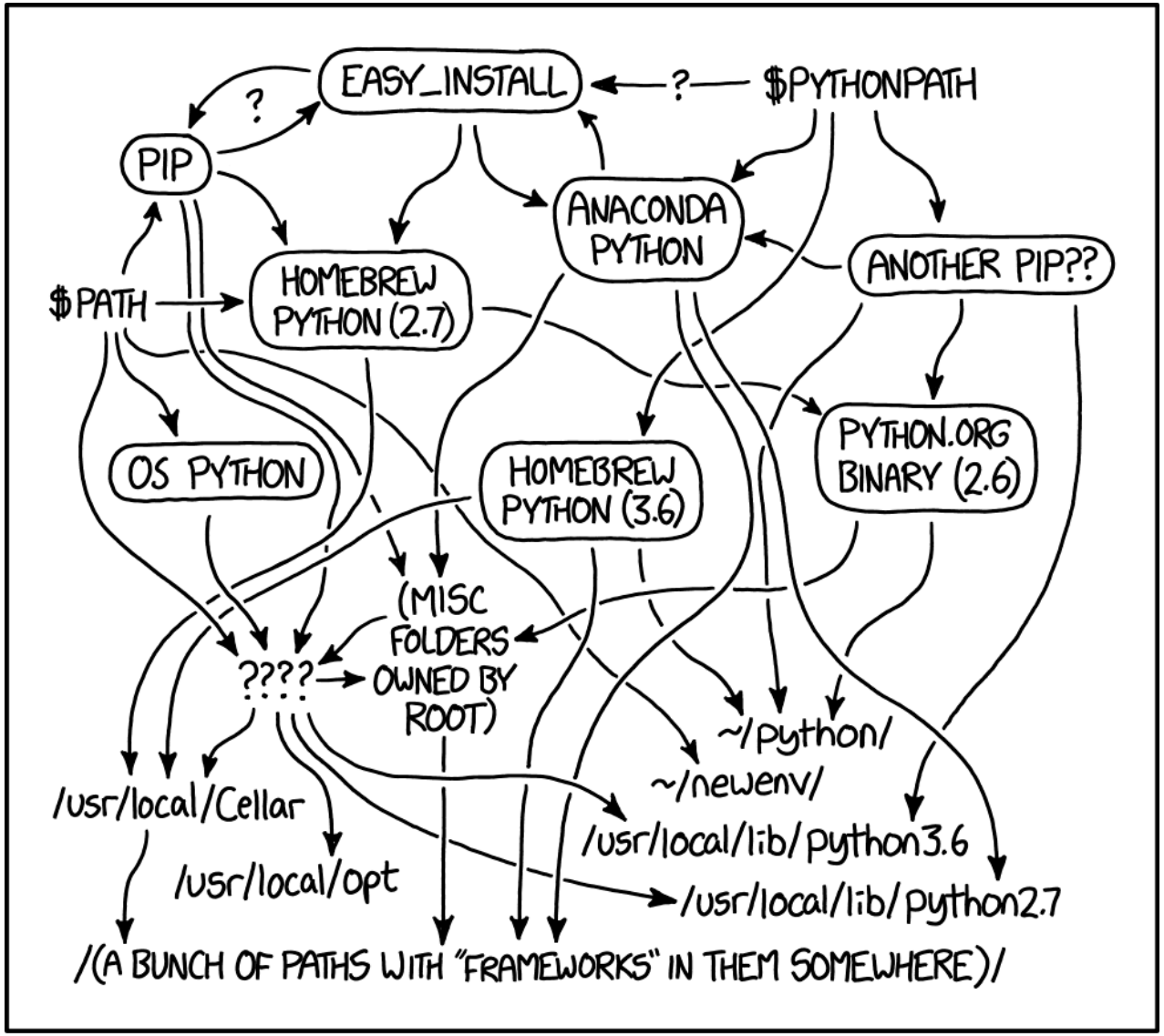


Managing your HPC software environment

How It Started

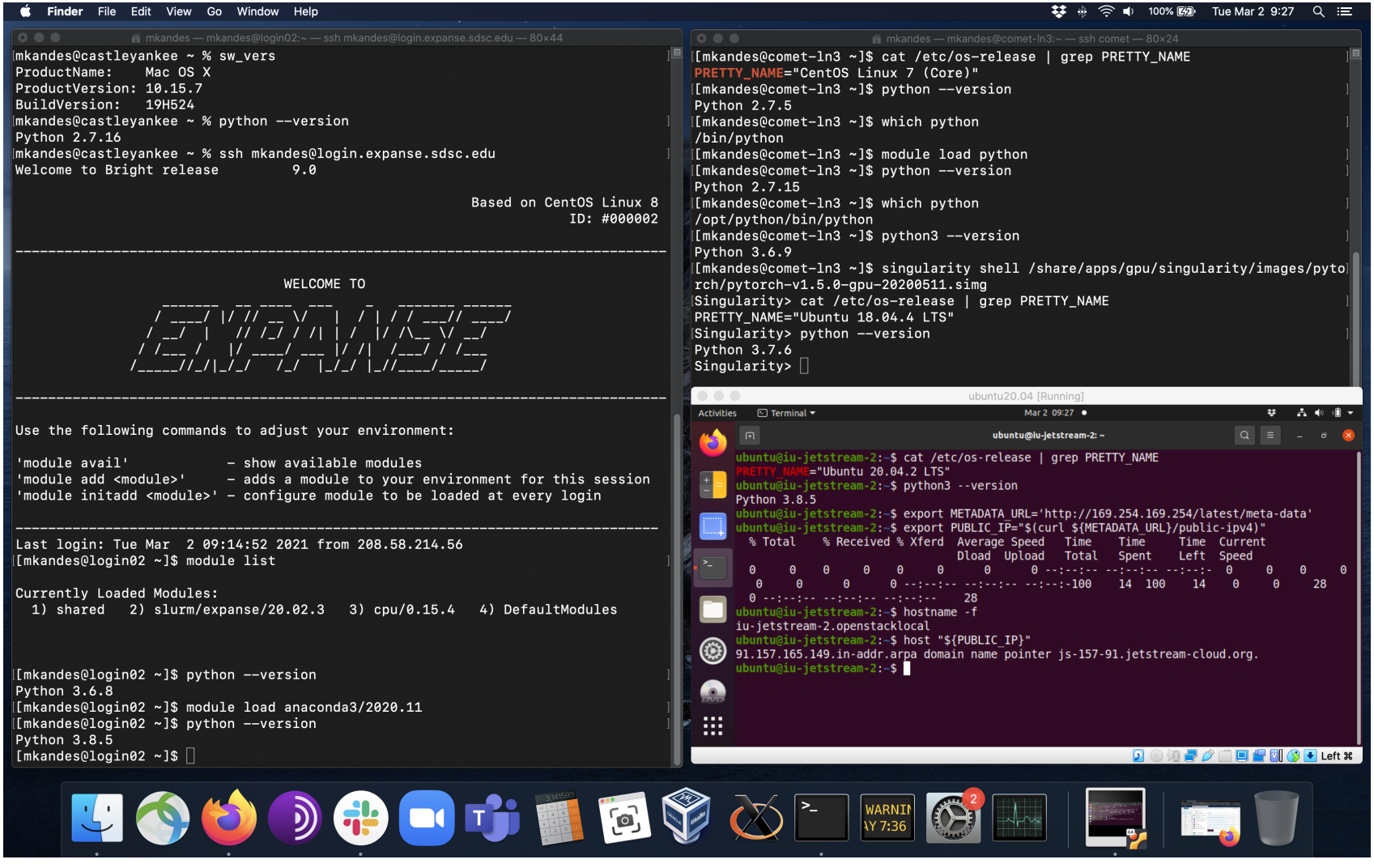


How It's Going



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

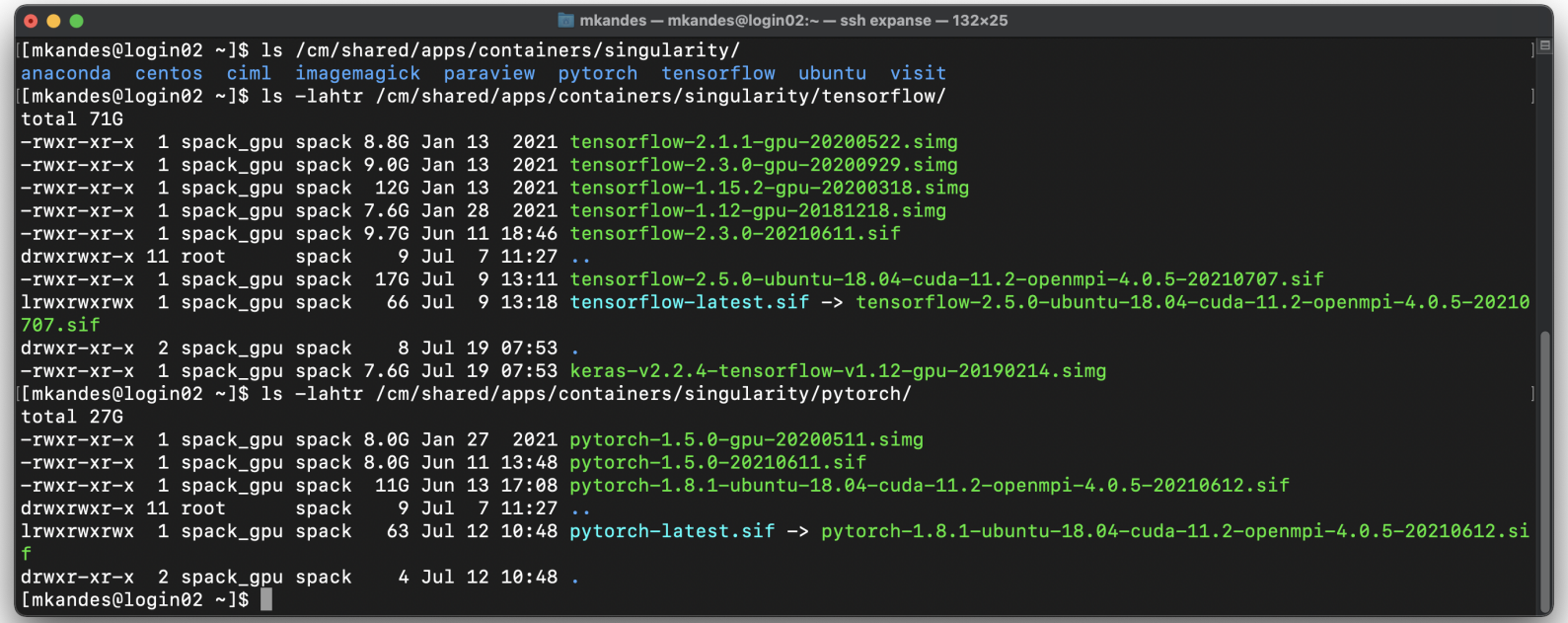
A Python Hodgepodge



What is a (Software) Container?

A (software) container is an abstraction for a set of technologies that aim to solve the the problem of how to get software to run reliably when moved from one computing environment to another.

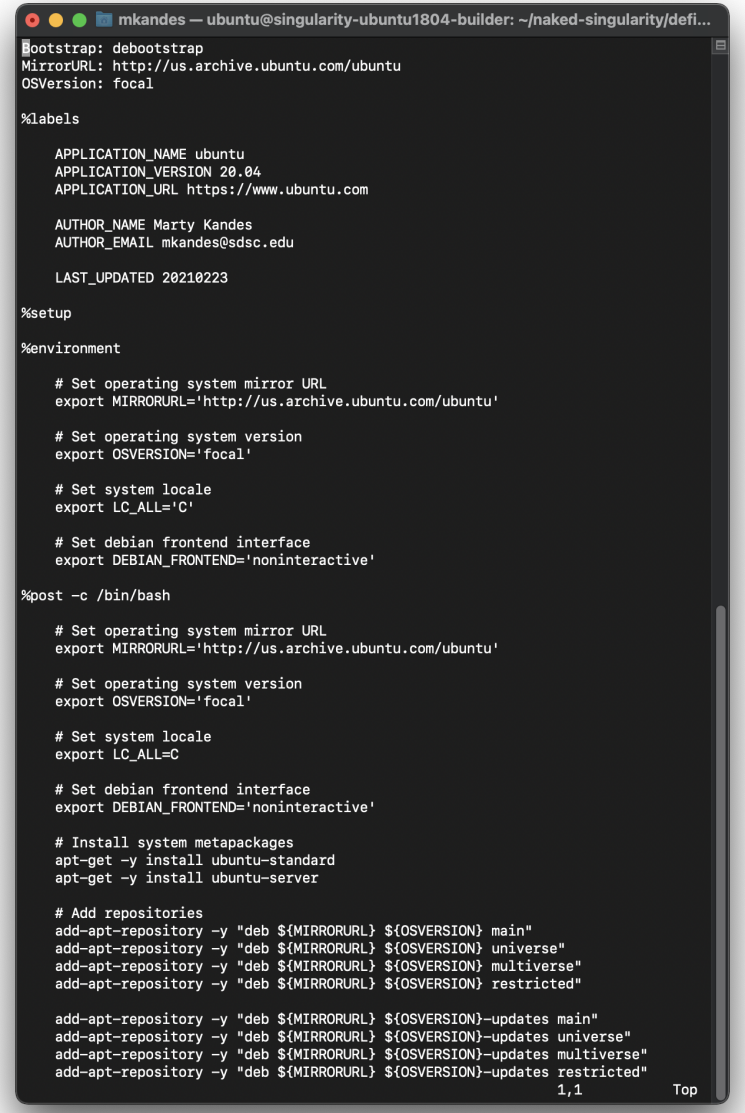
Container Image



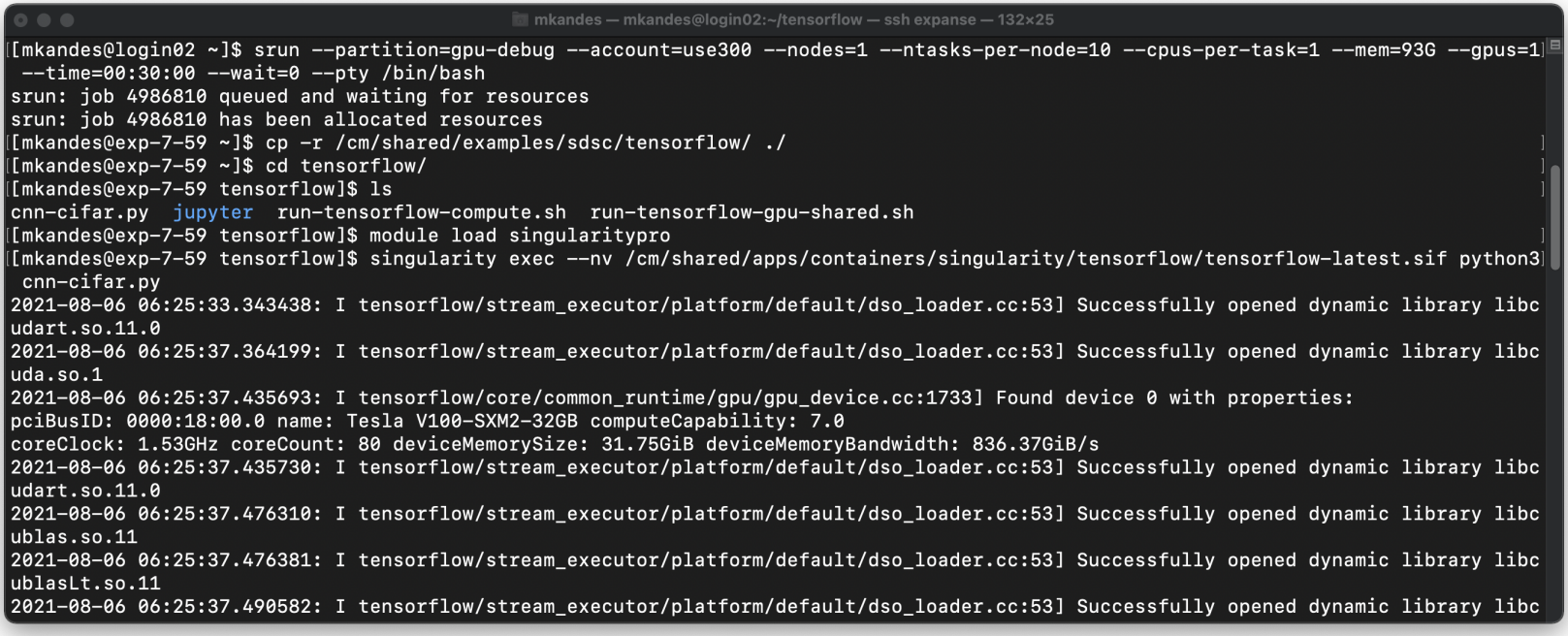
A container image is simply a file (or collection of files) saved on disk that stores everything you need to run a target application or applications: code, runtime, system tools, libraries, etc.

Container Definition File

A container definition file is simply a text file that contains all of the commands required to assemble a container image — i.e., it is a manifest of all software to be installed within the container, environment variables to be set, files to be added, directories to be mounted, container metadata, etc.

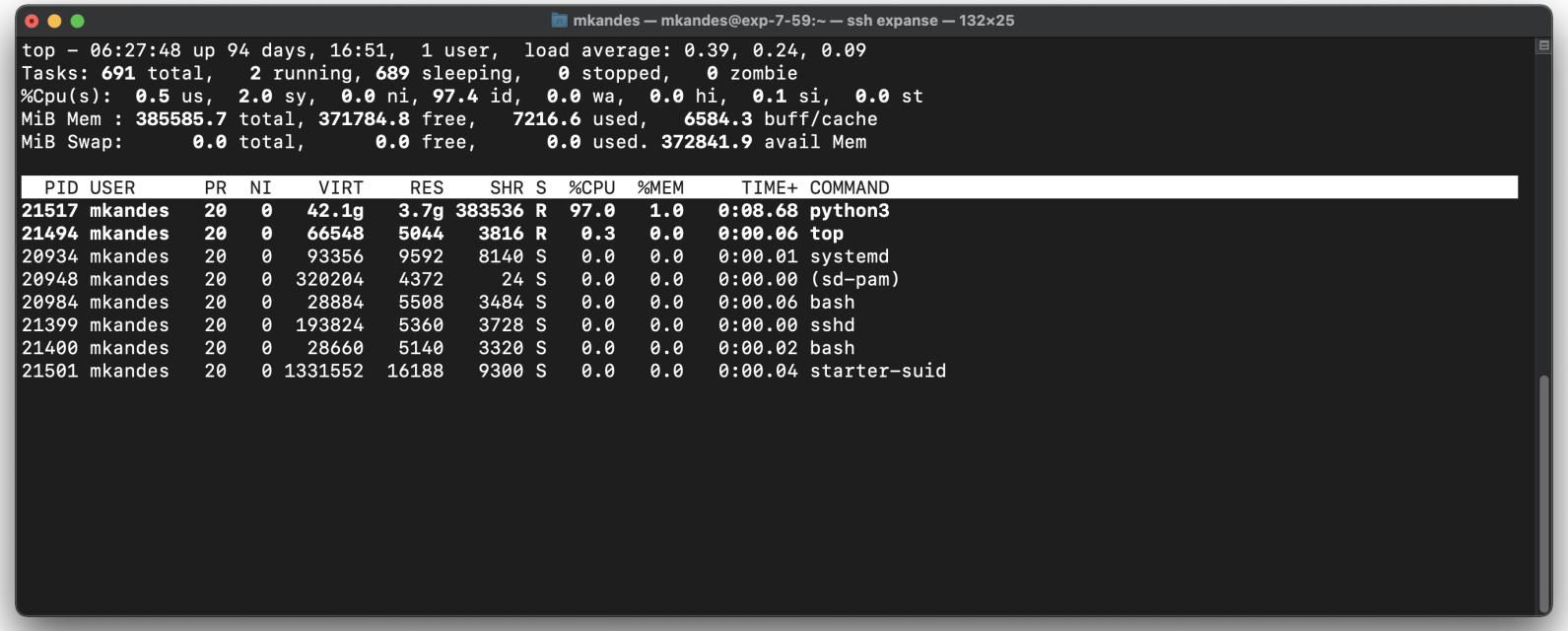


Container Process



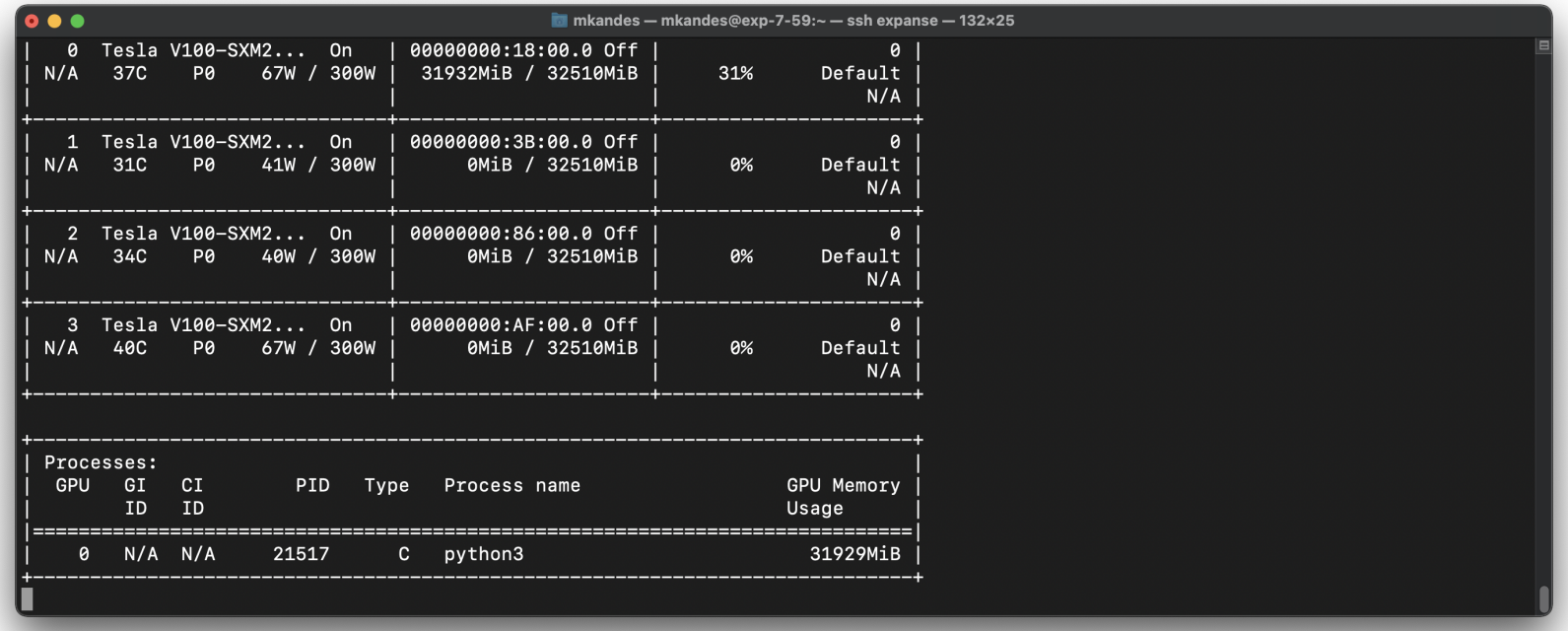
A container process is simply a standard (Linux) process running on top of the underlying host's operating system and kernel, but whose software environment is defined by the contents of the container image.

Container Process



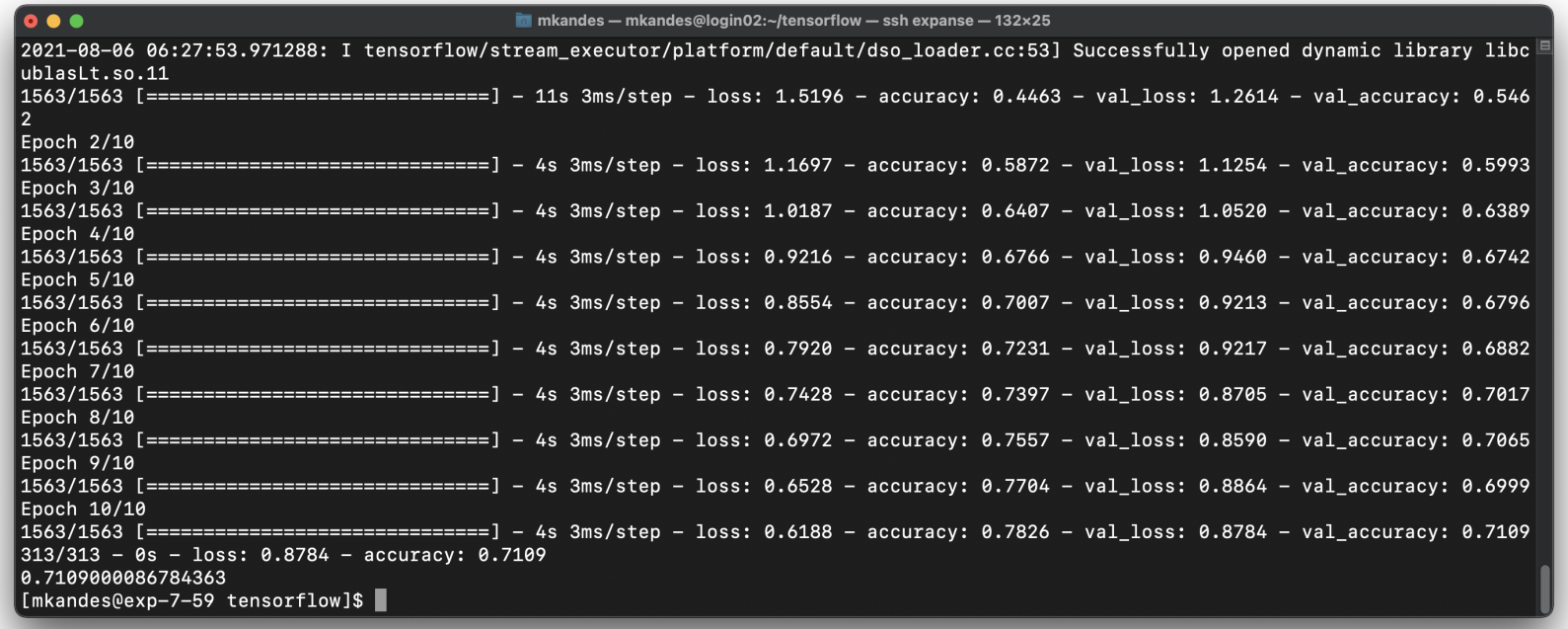
A container process is simply a standard (Linux) process running on top of the underlying host's operating system and kernel, but whose software environment is defined by the contents of the container image.

Container Process



A container process is simply a standard (Linux) process running on top of the underlying host's operating system and kernel, but whose software environment is defined by the contents of the container image.

Container Process



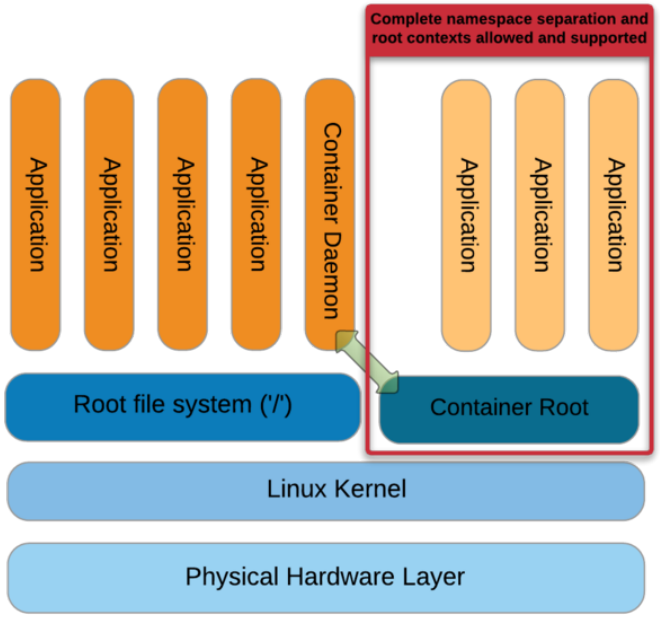
A container process is simply a standard (Linux) process running on top of the underlying host's operating system and kernel, but whose software environment is defined by the contents of the container image.

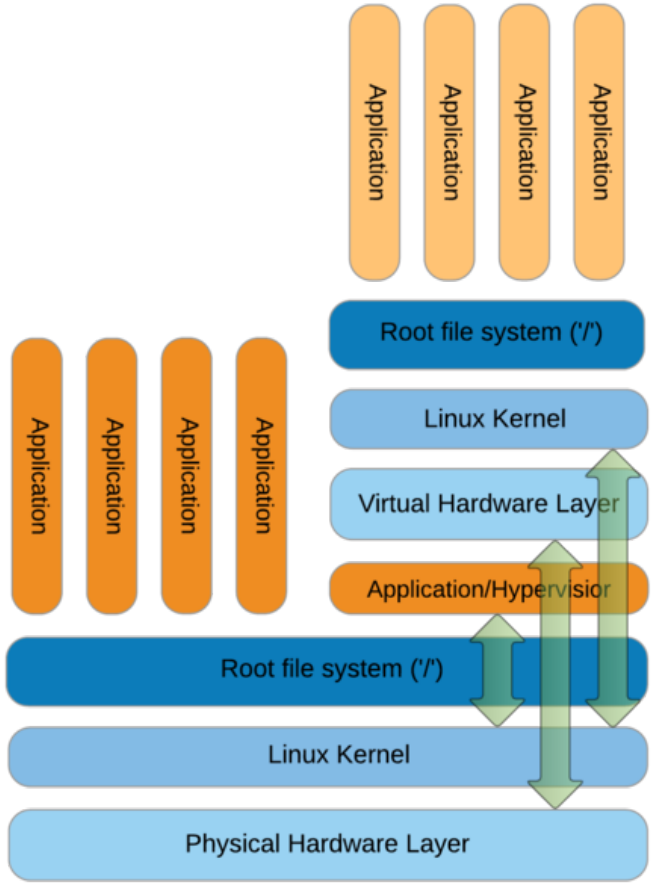
Container : Supercomputer :: Construct : Matrix

“ ... it’s our loading program.”

“ We can load anything ... anything we need.”

Containers vs. Virtual Machines





Container-based applications have *direct access* to the host kernel and hardware and, thus, are able to achieve similar performance to native applications. In contrast, VM-based applications only have *indirect access* via the guest OS and hypervisor, which creates a significant performance overhead.

Advantages of Containers

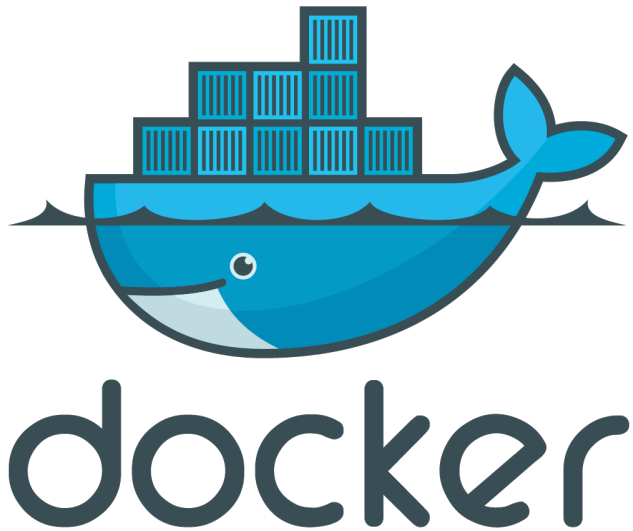
- **Performance**: Near-native application performance
- **Freedom**: Bring your own software environment
- **Reproducibility**: Package complex software applications into easy to manage, verifiable software units
- **Compatibility**: Built on open standards available in all major Linux distributions
- **Portability**: Build once, run (almost) anywhere

Limitations of Containers

- **Architecture-dependent**: Always limited by CPU architecture (x86_64, ARM) and binary format (ELF)
- **Portability**: Requires glibc and kernel compatibility between host and container; also requires any other kernel-user space API compatibility (e.g., OFED/IB, NVIDIA/GPUs)
- **Filesystem isolation**: filesystem paths are (mostly) different when viewed inside and outside container

Docker

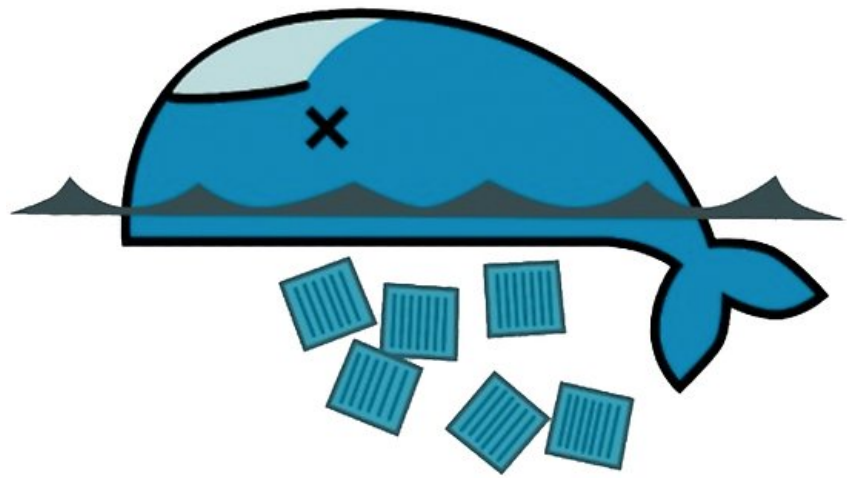
- Most common container platform in use today
- Provides tools and utilities to create, maintain, distribute, and run containers images
- Designed to accommodate network-centric services (web servers, databases, etc)
- Easy to install, well-documentated, and large, well-developed user community and container ecosystem (DockerHub)



`https://www.docker.com`

Docker on HPC Systems

- HPC systems are shared resources
- Docker's security model is designed to support trusted users running trusted containers; e.g., users can escalate to root
- Docker not designed to support batch-based workflows
- Docker not designed to support tightly-coupled, highly distributed parallel applications (MPI).



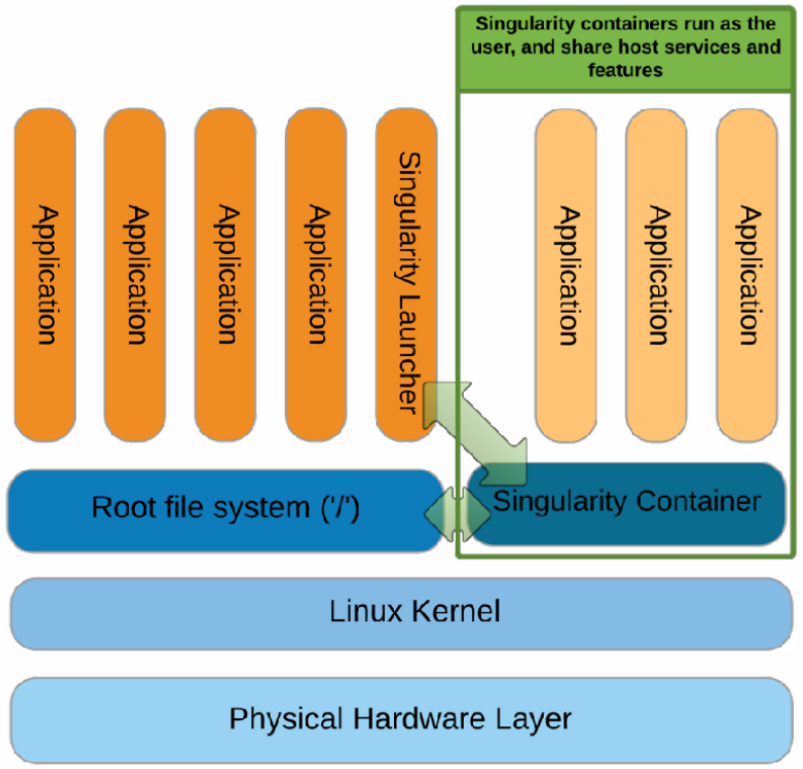
Singularity: A Container Platform for HPC

- Reproducible, portable, sharable, and distributable containers
- No trust security model: untrusted users running untrusted containers
- Support HPC hardware and scientific applications

`https://github.com/hpcng/singularity`
`https://www.sylabs.io`

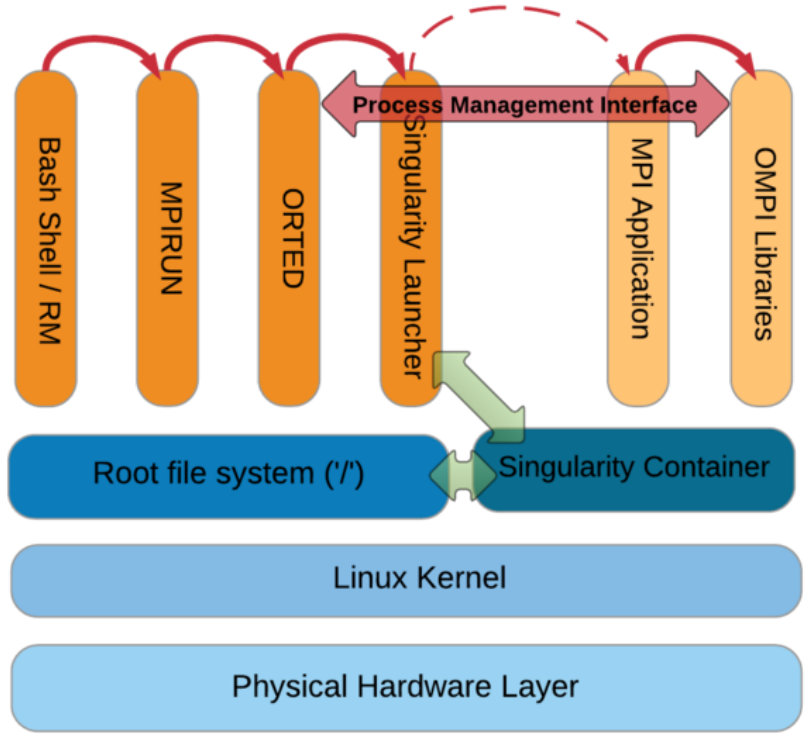
Features of Singularity

- Each container is a single image file
- No root owned daemon processes
- No user contextual changes or root escalation allowed; user inside container is always the same user who started the container
- Supports shared/multi-tenant resource environments



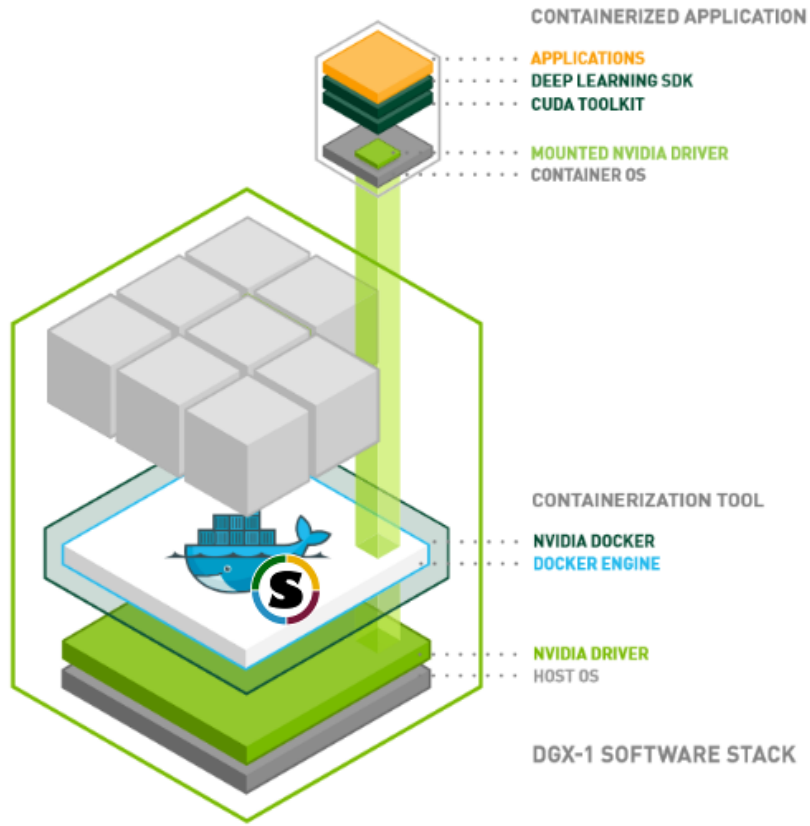
MPI-based Singularity Containers

- Use same Message Passing Interface (MPI) distribution and version within container as would be used outside the container.
- If using Infiniband (IB), install same OFED drivers and libraries inside the container as used on underlying HPC hardware.



GPU-accelerated Singularity Containers

- GPU-accelerated containers also require an interface for accessing GPU drivers and libraries on the underlying host system.
- Traditionally, you would install the same driver and libraries within container that match distribution and version of them available on the host system.
- Today, Singularity actually allows you to bind mount the GPU driver and its supporting libraries at runtime with the `--nv` option.

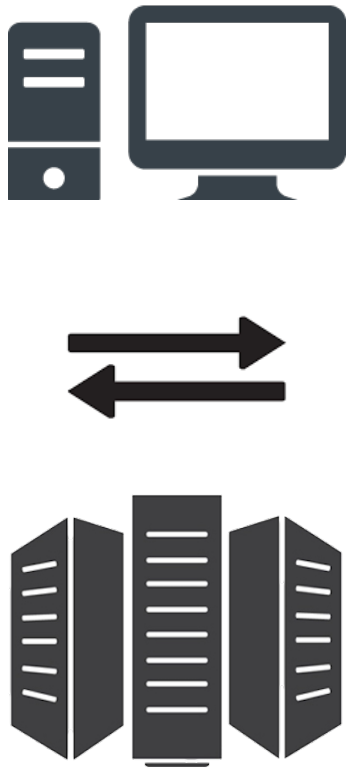


Most Common Singularity Use Cases

- Building and running applications that require newer system libraries than are available on host system
- Running commercial applications binaries that have specific OS requirements not met by host system
- Converting Docker containers to Singularity containers

The Singularity Workflow

1. **Build** your Singularity containers
2. **Transfer** your Singularity containers to the HPC system(s) where you want to run them
3. **Run** your Singularity containers on those HPC system(s)



Running Singularity on Mac OS X or Windows

1. Install VirtualBox on your personal computer:
2. Create either an Ubuntu or Fedora-based virtual machine, where you will build and test your Singularity containers
3. Install Singularity* on that virtual machine:

* Recommendation: Install the same version of Singularity used on the HPC system where you plan to run your containers. If you plan to run on multiple HPC systems, then install the lowest version number you expect to use.

naked-singularity

- A repository of definition files for building Singularity containers around the software applications, frameworks, and libraries you need to run on high-performance computing systems.

- Aim of the project is to:
  1. Version control the Singularity containers we're building, maintaining, and deploying for you;
  2. Make it easy for you to see what is installed within these Singularity containers; and
  3. Make available to you the same base definition files we use to build our Singularity containers, which can serve as a starting point for your own custom Singularity containers.

- `https://github.com/mkandes/naked-singularity`

Singularity: A Summary

1. You can now install (almost) any software you like on your favorite HPC system without having to make a special request to the system's administrators or user support staff.

2. In many cases, your software is now completely portable between the different HPC systems you want to run on.

3. And finally, you now have discrete software units (containers) that you can use to help maintain science reproducibility over the lifetime of a project, independent of how the software environment on any given HPC system changes over time.

Additional References

- Singularity User Guide:
  `https://sylabs.io/guides/latest/user-guide/`
- Example Singularity Definition Files:
  `https://github.com/mkandes/naked-singularity`
- Talks @ 1st Singularity User Group Meeting:
  `https://sylabs.io/videos`