Using Containers in a Standardized HEP Workflow

Tom Eichlersmith

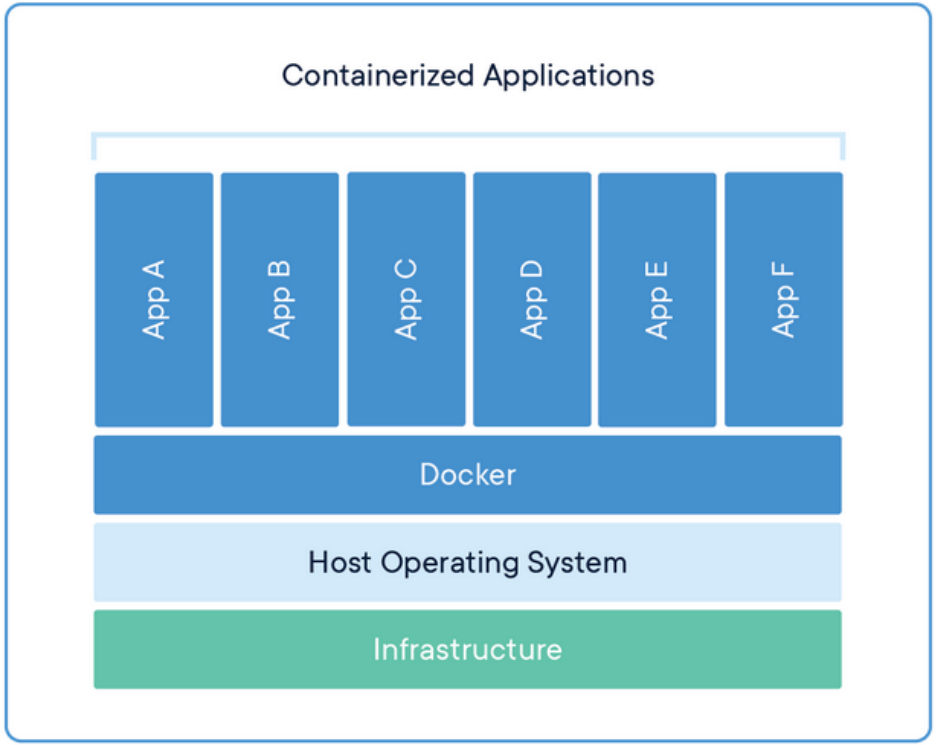
University of Minnesota

eichl008@umn.edu

August 16, 2021

What?

Containers???



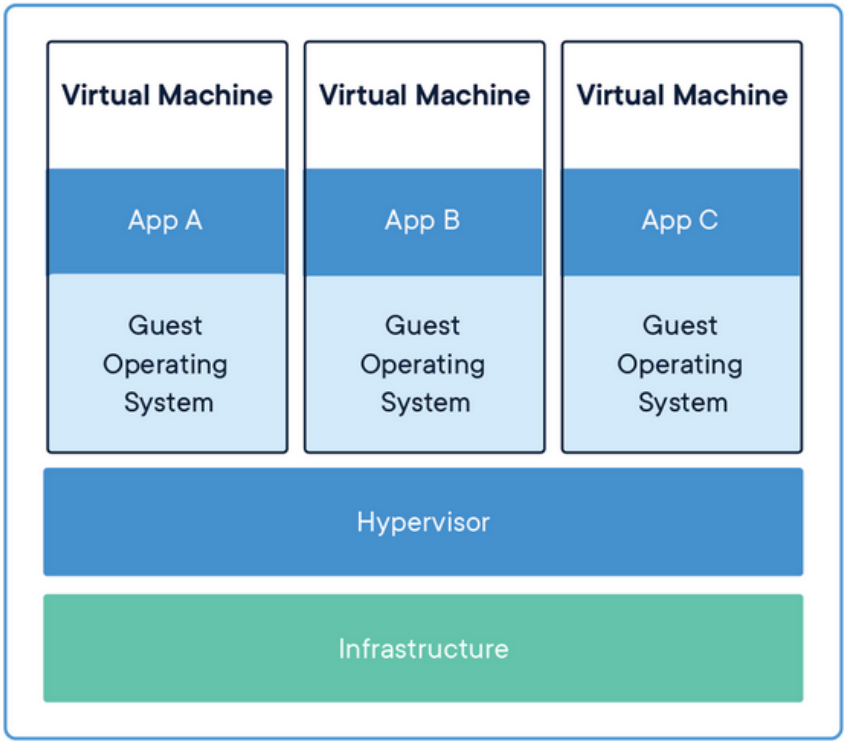


Figure : From ▸ Docker Docs

Why?

<rant>

Steep Learning Curve

Things new high-energy physicists need to learn before they can even start...

- Command-line basics (`cd`, `ls`, ...)
- Remote access to a computer (`ssh`)
- Terminal-based text editor (`vim`, `emacs`, ...)
- How terminal environments work (`source`, sub-processes, ...)
- Building source code from the command line (`cmake`, `make`, ...)
- Building ROOT from source (this is different than any old source code)
- **Both** Python and C++
- Version control (`git`)

This is absolute bonkers.

Containers can help us remove four to five of this implicit pre-requisites for early users.

</rant>

Areas of Improvement

Collaboration

Identical environments across users

Accessibility

Go from nothing to analyzing simulated data in a single day

Useability

Run software on a much broader set of operating systems

Common Arguments

LDMX

Isn't the container slower?

The average start-up time for the LDMX container is $\sim$ 0.3s. This is a constant that does not scale with the length of time the command inside the container takes.
$\Rightarrow$ only slow if consistently running commands faster than 1s

What about parallelization?

▸ Embarrassingly parallel vs ▸ Multiprocessing

HEP can use Embarrassingly parallel style for a vast majority of our workload.

Containers aren't persistent

That's what bind-mounts are for.

How?

Three Part Process

Build container image

- Installations of software you wish to use
- An "entrypoint script" which sources the necessary environment scripts before running the user-supplied command

Wrap container-running commands in shell helper functions

- Helps ensure user launchs container in correct manner
- Download default (or requested) container images
- Bind-mounts necessary directories for container

Test newly built image

- Make sure dependencies were successfully installed
- Check that the shell functions are operating correctly

Proof of Concept

Quick Start

- Install the docker engine
- (on Linux systems) Manage docker as non-root user
- Clone the repo: `git clone --recursive https://github.com/LDMX-Software/ldmx-sw.git`
- Setup the environment (in bash): `source ldmx-sw/scripts/ldmx-env.sh`
- Make a build directory: `cd ldmx-sw; mkdir build; cd build;`
- Configure the build: `ldmx cmake ..`
- Build and Install: `ldmx make install -j2`
- Now you can run any processor in *ldmx-sw* through `ldmx fire myconfig.py`

Figure : ▸ ldmx-sw Quickstart

Even CMSSW is trying to use ▸ singularity

Although they are using it differently (worse in my opinion)

More Detail

LDMX

On GitLab: ▸ tbeichlersmith/hep-env

- Operational image-building using GitLab CI
- Entrypoint script with necessary environment setup
- External environment script with basic functionality
- Container registry with foundation, root, and ldmx containers as examples

In Development

Needs more documentation and debugging, but will be a good leap for getting started