containerized **d**evelopment **env**ironments

Tom Eichlersmith

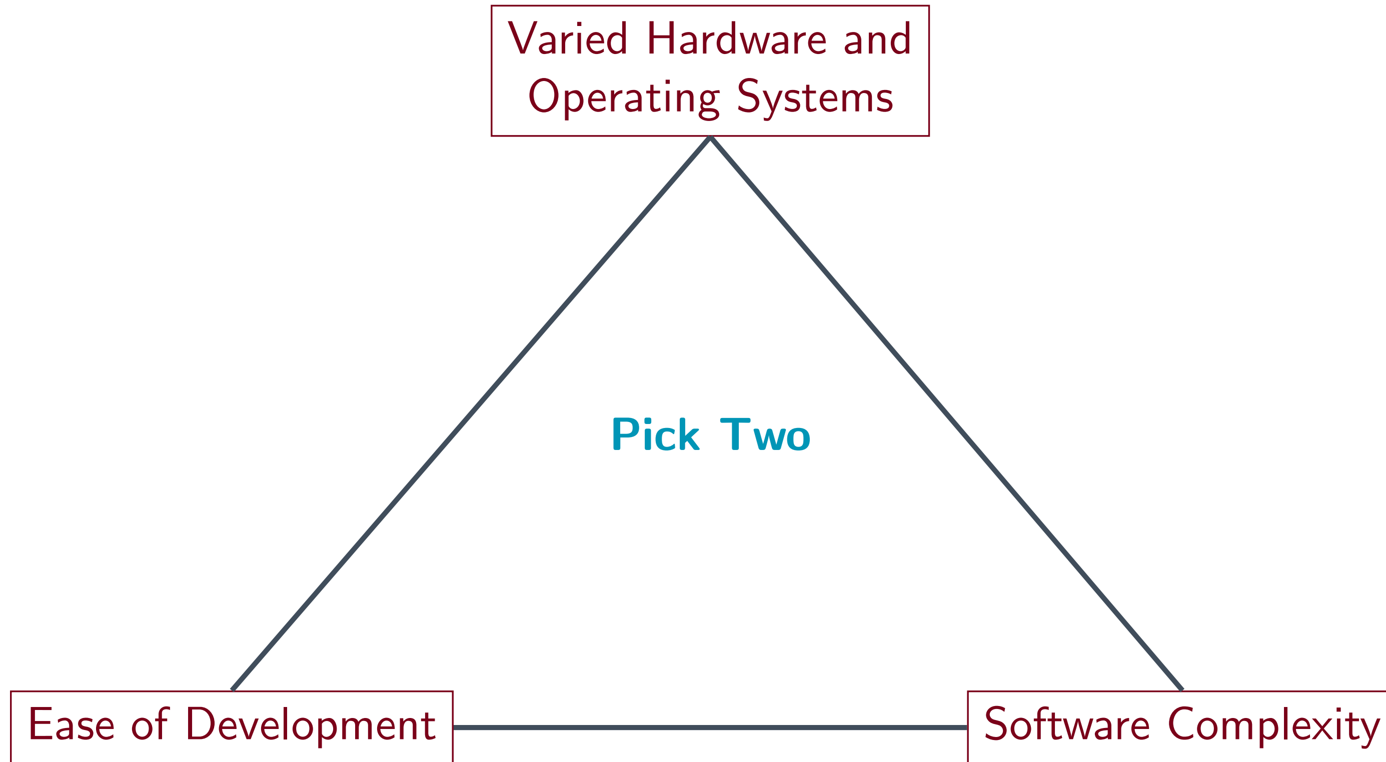
he/him/his

University of Minnesota

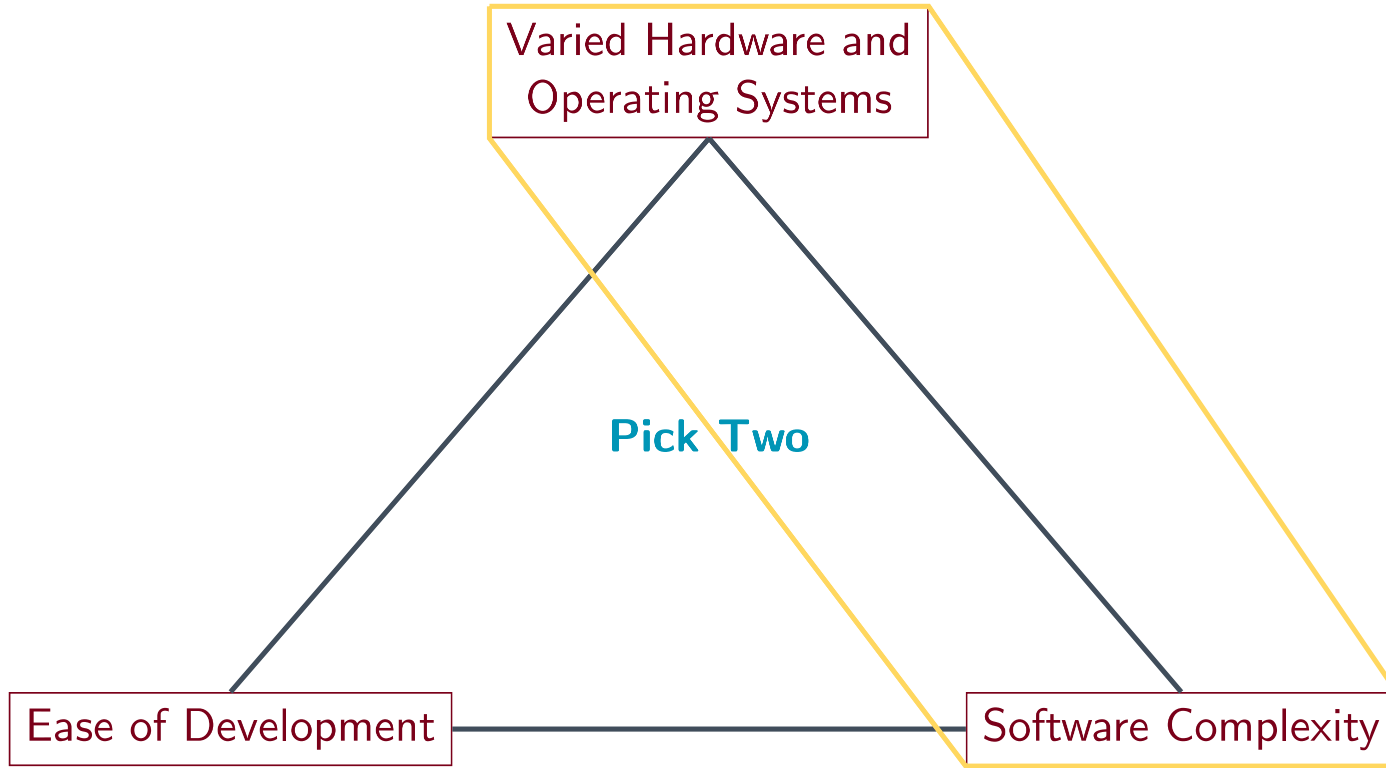
eichl008@umn.edu

September 5, 2023

Problem Statement



Problem Statement



Currently Employed Technologies in HEP

CVMFS

- Able to handle a variety of (Unix) operating systems
- Can efficiently provide pre-built packages
- "Delicate" development environment (must remember to initialize correct packages and sometimes in certain order)
- Must interface with central provider if new package (or different version) is desired

Containers as an Advanced Technique

- Often used for batch computing on clusters or in CI/CD workflows
- Difficulty of configuring them properly often leads to lack of use
- Some wrapper scripts exists (e.g. cmssw-cc*) but are often limited to experiment (e.g. CMS) or container runner (e.g. singularity).

Obtain Ease of Development

via containers

1. Offload building dependencies and configuring environment to experts
2. Put this constructed environment into a container image and distribute
3. Simplify usage by wrapping container runner for normal developer

- Constructing (1) and Persisting (2) $\Rightarrow$ complicated and for experts
- **Distribution** allows sharing the benefits of expert knowledge

Step 3 is Key

A "normal developer" will

- want to avoid writing a 100+ character command
- be told how to access or setup the environment

Available Solutions

I am not the first to recognize these benefits.

All of these do the wrapping and some help alleviate the complexities of construction and persistence.

Runner Separation

A lot of industry uses docker/podman while most academic/HEP HPCs use apptainer/singularity

This is what **denv** addresses.

- distrobox: allows for mutable environment within a container via docker and podman
- VSCode Dev Containers: runs VS Code from within a docker container with pre-installed dependencies
- devpod: provides dev-container-like behavior for a variety of IDEs
- devbox: "similar to a package manager ... except packages it manages are at the [OS] level"
- toolbox: built on top of podman allowing for experimentation
- repro-env: evolves a manifest into a lock file, attached to Arch/Debian package managers

Proof of Concept

ldmx-sw

Past

Standard HEP workflow

- Initialized dependencies from CVMFS
- Used custom-builds of Geant4 and ROOT on each cluster
- Required new users to learn SSH and terminal skills in order to contribute

Present

- Development environment version controlled as a container image
- All software dependencies boiled down to a container runner
- Batch running done in environments identical to where software is developed

Proof of Concept

The "Quick Start" is the **full set** of instructions.

`ldmx-env.sh` wraps `docker` or `singularity` so that everyone can use `ldmx` in the same way.

All variability in setup is handled by installation of container runner.

Quick Start

- Install the docker engine
- (on Linux systems) Manage docker as non-root user
- Clone the repo: `git clone --recursive git@github.com:LDMX-Software/ldmx-sw.git`
  - **Note**: You need to setup an SSH-key with your GitHub account on the computer you are using.
- Setup the environment (in bash): `source ldmx-sw/scripts/ldmx-env.sh`
  - **Note**: If you are working with ldmx-sw at SLAC's SDF, you will need to set the `TMPDIR` environment variable so that program running the container has more than ~5GB of space to write intermediate files. The default temporary space (`/tmp`) is often full of other files already. A decent replacement is `TMPDIR=/scratch/$USER` which gives the program plenty of room for the files it needs to manipulate.
- 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`

denv

Originated from desire to generalize ldmx-env.sh
Supports docker, podman[1], apptainer, and singularity.

POSIX Compliant

Usable in any POSIX-compliant shell.

Interactivity

Open a shell or run a command inside the denv

Workspace as Home

"Re-map" code workspace to be the home directory within the environment so natural *PATH variables and *rc files can be used.

Text File Config

Configuration stored in a plain text file and can be kept within version control along with code being developed within the denv.

[1] not perfectly, slightly different in-container environment, see denv#9

Going Forward

Development Tasks

- Maintain support for these runners as they are developed
- Potentially add other runners if demand exists
- Improve documentation especially in how to get started (often new users would need help constructing a first version of the environment)

Where HSF Comes In

- Expand to broader user and developer base
- A longer-term support infrastructure

Questions