bw|HPC

KIT | NHR
National High-Performance Computing Center

Enroot and Pyxis - first experiences at NHR@KIT

Samuel Braun, SCC, KIT

Funding:
Baden-Württemberg
MINISTERIUM FÜR WISSENSCHAFT, FORSCHUNG UND KUNST
Bundesministerium für Bildung und Forschung

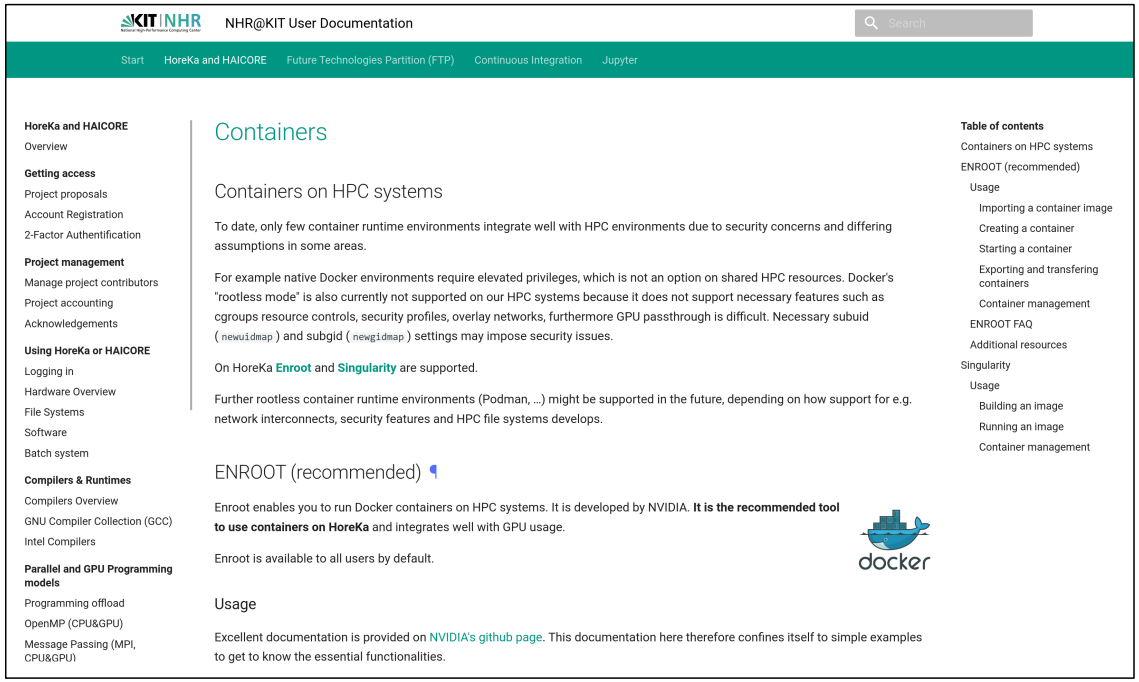
www.bwhpc.de / www.nhr.kit.edu

Outline

- How NVIDIA sees Enroot
- How-To
- What we think of Enroot
- PoC: Bring-your-own-Jupyter
- Open Questions and Summary
- Questions

Enroot Documentation

- Enroot + Pyxis: Documentation
  - Enroot project: https://github.com/NVIDIA/enroot
  - Pyxis project: https://github.com/NVIDIA/pyxis
  - NHR@KIT Wiki: https://www.nhr.kit.edu/userdocs/horeka/containers/



NHR@KIT User Documentation

Search

Start | HoreKa and HAICORE | Future Technologies Partition (FTP) | Continuous Integration | Jupyter

HoreKa and HAICORE
Overview

Getting access
Project proposals
Account Registration
2-Factor Authentification

Project management
Manage project contributors
Project accounting
Acknowledgements

Using HoreKa or HAICORE
Logging in
Hardware Overview
File Systems
Software
Batch system

Compilers & Runtimes
Compilers Overview
GNU Compiler Collection (GCC)
Intel Compilers

Parallel and GPU Programming models
Programming offload
OpenMP (CPU&GPU)
Message Passing (MPI, CPU&GPU)

Containers

Containers on HPC systems

To date, only few container runtime environments integrate well with HPC environments due to security concerns and differing assumptions in some areas.

For example native Docker environments require elevated privileges, which is not an option on shared HPC resources. Docker's "rootless mode" is also currently not supported on our HPC systems because it does not support necessary features such as cgroups resource controls, security profiles, overlay networks, furthermore GPU passthrough is difficult. Necessary subuid (`newuidmap`) and subgid (`newgidmap`) settings may impose security issues.

On HoreKa **Enroot** and **Singularity** are supported.

Further rootless container runtime environments (Podman, ...) might be supported in the future, depending on how support for e.g. network interconnects, security features and HPC file systems develops.

ENROOT (recommended) ¶

Enroot enables you to run Docker containers on HPC systems. It is developed by NVIDIA. **It is the recommended tool to use containers on HoreKa** and integrates well with GPU usage.

Enroot is available to all users by default.

docker

Usage

Excellent documentation is provided on NVIDIA's github page. This documentation here therefore confines itself to simple examples to get to know the essential functionalities.

Table of contents

How NVIDIA sees Enroot

NVIDIA's Motivation*

- Containers for HPC – What do we need?
  - High performance
  - Support for Docker images
  - Soft cluster multi-tenancy
  - Exposing NVIDIA GPUs and Mellanox InfiniBand cards inside containers
  - Resources (CPU/Mem/GPUs/HCAs) isolation through cgroups
  - Launching multi-node jobs
  - Development workflow: interactive jobs, installing packages, debugging
  - **No existing container runtime fulfilled all our requirements, so we built our own**

***) Distributed HPC Applications with Unprivileged Containers**,
Felix Abecassis, Jonathan Calmels, Fosdem 2020
https://fosdem.org/.../containers_hpc_unprivileged.pdf

How NVIDIA describes Enroot + Pyxis**

- **Enroot**
  - Traditional container images → unprivileged sandboxes
  - Lightweight, performant container runtime
  - Built-in NVIDIA GPU support
  - No performance impact
- **Pyxis**
  - Plugin, integrates Enroot with Slurm
  - Adds command-line arguments to srun
  - One or more nodes
  - Support for PMIX

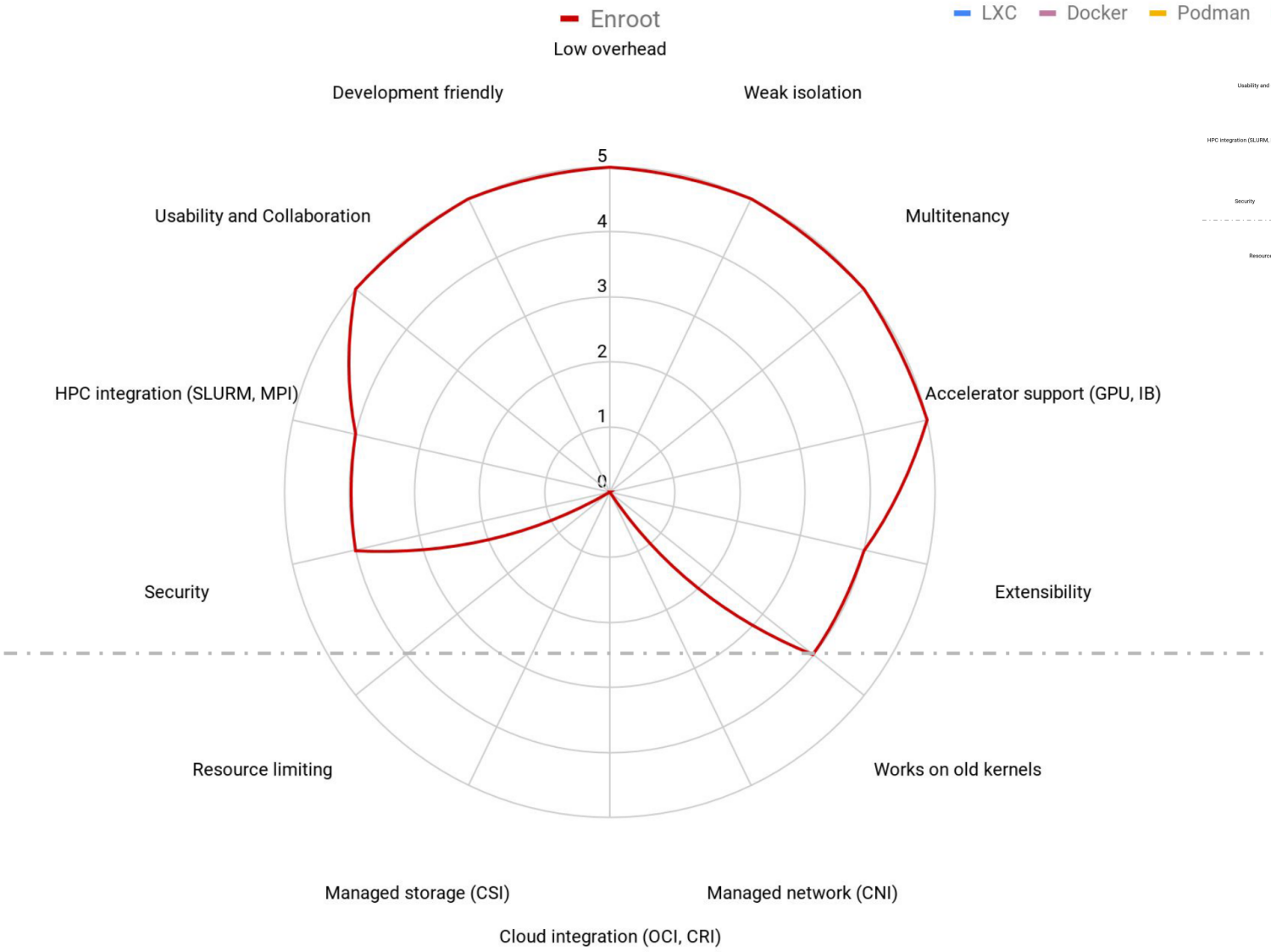
) **"MULTI-NODE BERT User Guide", DU-09659-001_v |September 2019, https://docs.nvidia.com/ngc/pdf/multi-node-bert-user-guide.pdf

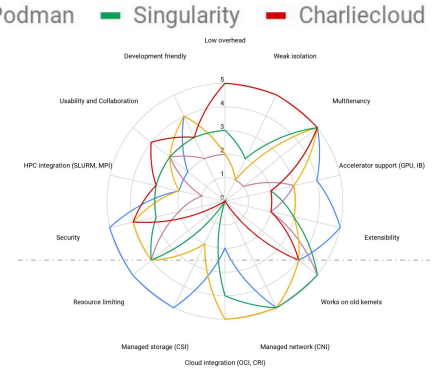
How NVIDIA describes Enroot + Pyxis***

- Fully unprivileged “chroot” (with optional root-remapping)
- Standalone (no daemon, no extra process)
- Simple and easy to use (UNIX philosophy, KISS principle)
- Little isolation, no overhead
- Docker image support (5x pull speedup, shared cache)
- Simple image format (single file + UNIX configs)
- Composable and extensible (system/user configs, lifecycle hooks)
- Advanced features (runfiles, scriptable configs, in-memory containers)

***) **SLURM: Seamless Integration With Unprivileged Containers**
Felix Abecassis, Julie Bernauer, Jonathan Calmels, Louis Capps, Michael Knox, Luke Yeager
Slurm User Group Meeting 2019
https://slurm.schedmd.com/SLUG19/NVIDIA_Containers.pdf

Competence Spider Diagram***





Enroot and Pyxis: How-To

How-To: Use

- **Import a container image**
  - `enroot import docker://alpine`
  - Pulls the latest alpine image from dockerhub (default registry)

    → `alpine.sqsh`

  - `enroot import docker://nvcr.io#nvidia/pytorch:21.04-py3`
  - Pulls the latest pytorch image from NVIDIA's NGC registry

    → `nvidia+pytorch+21.04-py3.sqsh`

  - `enroot import dockerd://myalpine`
  - `enroot import podman://myalpine`
  - Get image from a running Docker daemon / local Podman registry

    → `myalpine.sqsh`

How-To: Use

- **Create a container**
  - `enroot create --name alpine alpine.sqsh`
  - Create a container named `alpine` by unpacking the `.sqsh`-file

  - Creating: unpack squashed container image file
    - `$ENROOT_DATA_PATH/` (default: `$HOME/.local/share/enroot/`)

How-To: Use

- **Start a container**
  - `enroot start --rw nvidia+pytorch+21.04-py3 bash`
  - Start the container in read-write mode (`--rw`) and run bash inside the container

  - `enroot start --root --rw nvidia+pytorch+21.04-py3 bash`
  - Start container with root privileges (`--root`) inside the container

  - ```
    enroot start -m <localDir>:/work -rw \
            nvidia+pytorch+21.04-py3 bash
    ```
  - Start container and mount (`-m`) a local directory to `/work` inside the container.

  - ```
    enroot start -m <localDir>:/work -rw \
            nvidia+pytorch+21.04-py3 jupyter lab
    ```
  - Start container and start the application `jupyter lab`

How-To: Use

```
# Import the CUDA 10.0 image from NVIDIA GPU Cloud
$ enroot import 'docker://$oauthtoken@nvcr.io#nvidia/cuda:10.0-base'

# Create a container out of it
$ enroot create --name cuda nvidia+cuda+10.0-base.sqsh
$ enroot list
cuda

# Compile the nbody sample inside the container
$ enroot start --root --rw cuda sh -c 'apt update && \
        apt install -y cuda-samples-10.0'

$ enroot start --rw cuda sh -c 'cd \
        /usr/local/cuda/samples/5_Simulations/nbody && make -j'

# Run nbody leveraging the X server from the host
$ export ENROOT_MOUNT_HOME=y NVIDIA_DRIVER_CAPABILITIES=all

$ enroot start  --env DISPLAY \
                --env NVIDIA_DRIVER_CAPABILITIES \
                --mount /tmp/.X11-unix:/tmp/.X11-unix cuda \
                /usr/local/cuda/samples/5_Simulations/nbody/nbody
# Remove the container
$ enroot remove cuda
```

How-To: Use

- **Exporting and transferring containers**
  - Export an existing enroot container via
    `enroot export --output myImage.sqsh myImage`

  😎 Create a self extracting bundle from a container image via
    `enroot bundle --output myImage.run myImage.sqsh`
  - Run the self extracting image via `./myImage.run`
    …even if enroot is not installed!

How-To: Use

- Pyxis
  - `srun --container-image=centos grep PRETTY /etc/os-release`
    `PRETTY_NAME="CentOS Linux 8"`
  - `srun --container-image nvcr.io#nvidia/pytorch:20.03-py3 …`
  - `srun --container-image ~/ubuntu.sqsh …`

How-To: Use

- `srun --help`
  - …
  - `--container-image=[USER@][REGISTRY#]IMAGE[:TAG]|PATH`
  - `--container-mounts=SRC:DST[:FLAGS][,SRC:DST...]`
  - `--container-workdir=PATH`
  - `--container-name=NAME`
  - `--container-save=PATH`
  - `--container-mount-home`
  - `--no-container-mount-home`
  - `--container-remap-root`
  - `--no-container-remap-root`
  - `--container-entrypoint`
  - `--no-container-entrypoint`

How-To: Install

Enroot

```
$ arch=$(uname -m)
$ yum install -y epel-release # required on some distributions
$ yum install -y https://github.com/NVIDIA/enroot/releases/download/v3.4.0/enroot-3.4.0-2.el7.${arch}.rpm
$ yum install -y https://github.com/NVIDIA/enroot/releases/download/v3.4.0/enroot+caps-3.4.0-2.el7.${arch}.rpm # optional
```

Pyxis

```
$ make rpm
$ rpm -i x86_64/nvslurm-plugin-pyxis-*-1.el7.x86_64.rpm
$ ln -s /usr/share/pyxis/pyxis.conf /etc/slurm/plugstack.conf.d/pyxis.conf
$ systemctl restart slurmd
```

➡ easy

Build dependencies:
`git gcc make libcap libtool automake`

Runtime dependencies:
`jq squashfs-tools parallel`

optional:
`fuse-overlayfs libnvidia-container-tools pigz squashfuse`

What we think of Enroot

Experiences

- Our experiences so far
  - No corruption of file permissions (à la udocker*)
  - No special permissions needed (à la Singularity)
  - Works on Intel, AMD, Fujitsu A64FX
  - GPU support: NVIDIA, AMD (sic!)
  - Easy installation, lightweight
- Our customers' experiences
  - MLPerf team switched to Singularity :(
  - Very few tickets (good or bad?)
- What we DIDN'T try extensively yet:
  - Multi-node behavior
  - Performance measurements

*) https://github.com/indigo-dc/udocker

Support

- We needed sbatch functionality for JupyterHub

Support for sbatch #55

Closed **dr-br** opened this issue on 4 Aug · 9 comments

🎉

flx42 commented on 7 Aug Member

@dr-br could you try the following branch? https://github.com/NVIDIA/pyxis/tree/2021-08-06/sbatch-salloc-support
It enables all the pyxis arguments for salloc and sbatch.

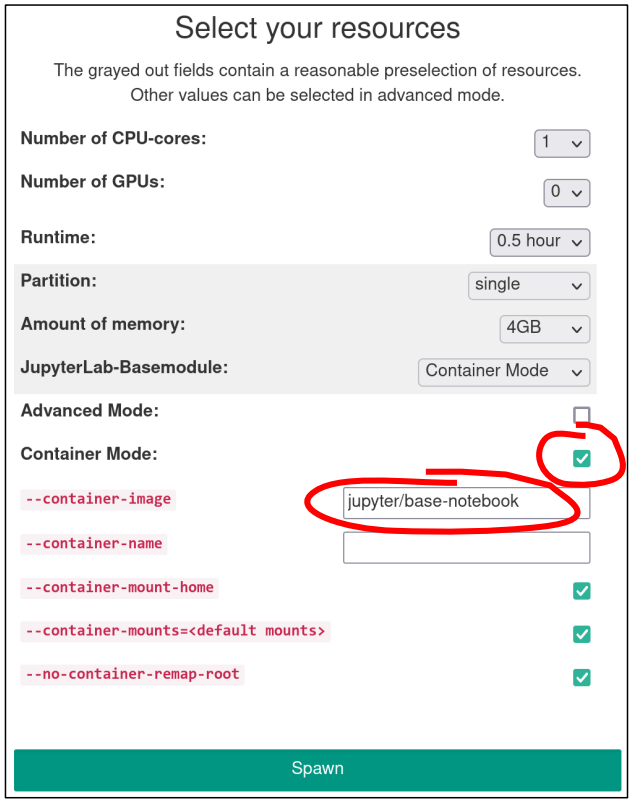
Only single node, unfortunately

PoC: Bring-your-own-Jupyter

PoC: Bring-your-own-Jupyter

- Connect containerized Jupyter with JupyterHub@HoreKa

- Docker images from any registry
- For complicated/intrusive software stacks
- Optimized software stacks
  - Intel, e.g.
    `intel/intel-optimized-tensorflow`
  - Nvidia, e.g.
    `nvcr.io#nvidia/tensorflow:21.10-tf2-py3`
  - AMD, e.g.
    `rocm/tensorflow:rocm4.3.1-tf2.6-dev`



PoC: Bring-your-own-Jupyter

- **{JupyterHub + batchspawner} + {Pyxis + Enroot}**

options_form.html

```
// Containerimage
var containerimageValue = document.getElementById("containerimage").value;
if (containerimageValue != "") {
    document.getElementById("containerimagestring").value = "#SBATCH --container-image " + containerimageValue;
} else {
    document.getElementById("containerimagestring").value = "";
}
```

jupyterhub_config.py

```
{containermodedeactivate[0]}#SBATCH --get-user-env=L
{containermodedeactivate[0]}#SBATCH --export={keepvars}


##SBATCH --no-container-remap-root
{enrootremaprootstring[0]}

##SBATCH --container-mount-home
{enrootmounthomestring[0]}

##SBATCH --container-mounts=/etc/slurm/task_prolog.ucc:/etc/slurm/task_prolog.ucc,/scratch:/scratch,/var/spool/slurmd:/var/spool/slurmd
{enrootdefaultmountstring[0]}

##SBATCH --container-image jupyter/base-notebook
{containerimagestring[0]}

##SBATCH --container-name jupyter+base-notebook
{containernamestring[0]}

#SBATCH --container-workdir={homedir}
```

PoC: Bring-your-own-Jupyter

`jupyterhub_config.py` → JupyterHub integration

```
c.SlurmSpawner.batch_script += '''
{containermodedeactivate[0]}module add {basemodule[0]}

{containermodeactivate[0]}export PYTHONUSERBASE=$HOME/.container_{containernameifvoid[0]}_packages
{containermodeactivate[0]}export PATH=$PYTHONUSERBASE/bin:$PATH
{containermodeactivate[0]}python3 -m pip install jupyterhub jupyterlab batchspawner
```

Open Questions & Summary

Open Questions & Summary

Open Questions

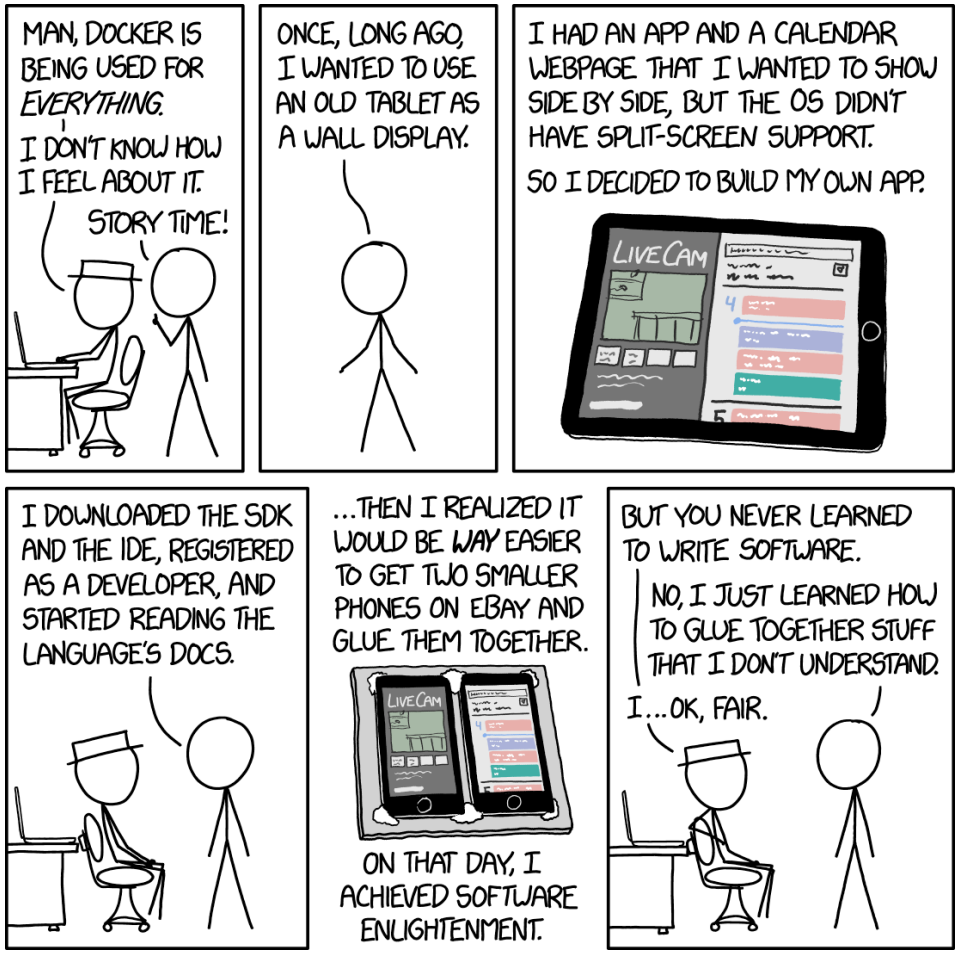
- How to build Containers?
- sbatch + multi node?
- Visibility → Acceptance
  - Singularity/Apptainer → Linux Foundation
- Future proof? Community?

Summary

- Slim, small footprint
- GPU support OOTB

- Possibly not ***THE ultimate***™ solution
- **To keeping an eye on!**

Thank you for your attention!
Questions?



https://xkcd.com/1988