ISIS Neutron and Muon Source

Containers – Introduction

RSE L&D 27/02/2024

Dr Franz Lang

Overview

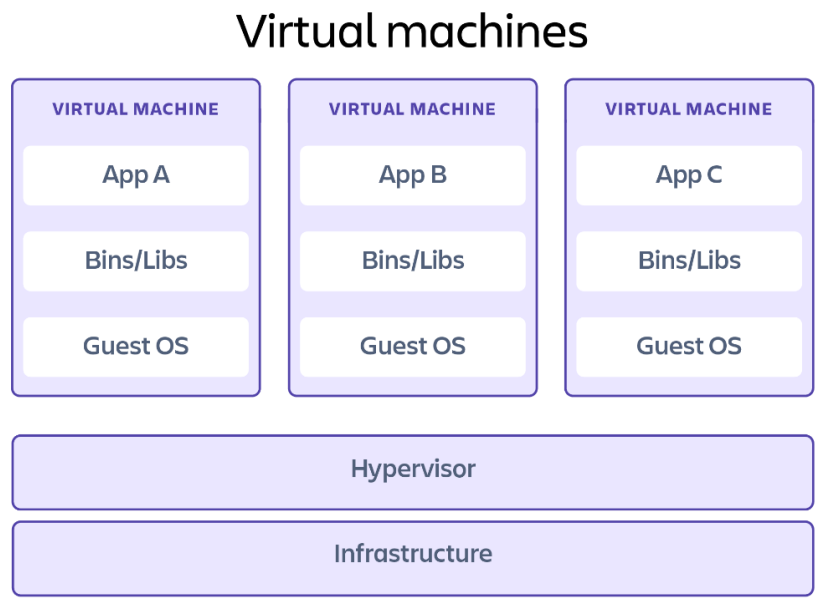
- Big Picture
- VM vs Container
- Architecture
- Workflow
- Docker vs Singularity/Apptainer
- Apptainer definition file
- Examples
- Registry
- Container Orchestration

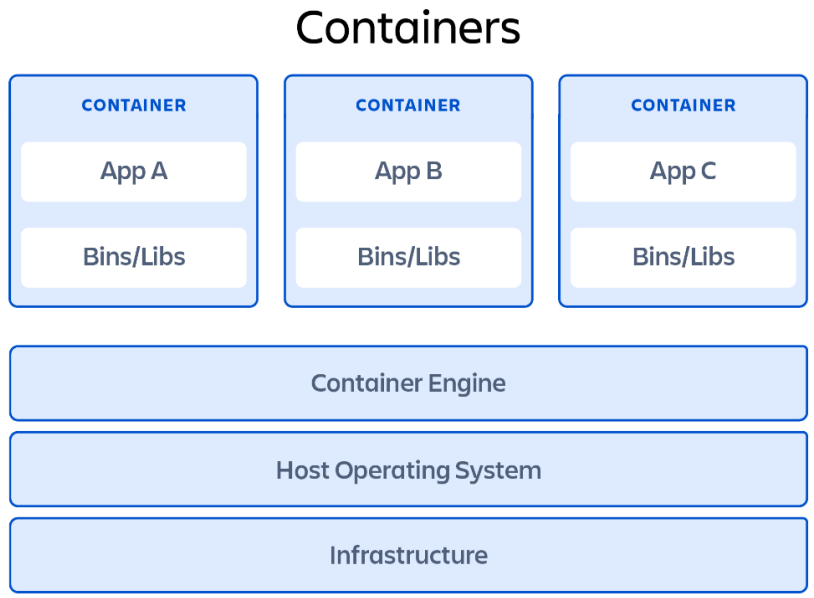
Big Picture

- Interoperability: OS-agnostic (almost)
  - Windows containers only on Windows
  - Linux containers on Windows via WSL
  - kernel function (in-)compatibilities
- Reproducable: standalone package for each app/service
  - contains all dependencies required
  - lightweight: can optimise content to be minimal
- Maintainable: text-file build recipe
- Deployable: hosting via a registry
  - dockerhub, harbour
- Orchestration: usable with clusters
  - kubernetes, docker swarm, podman…
- Security:
  - isolation of applications
  - can apply user privileges and security settings

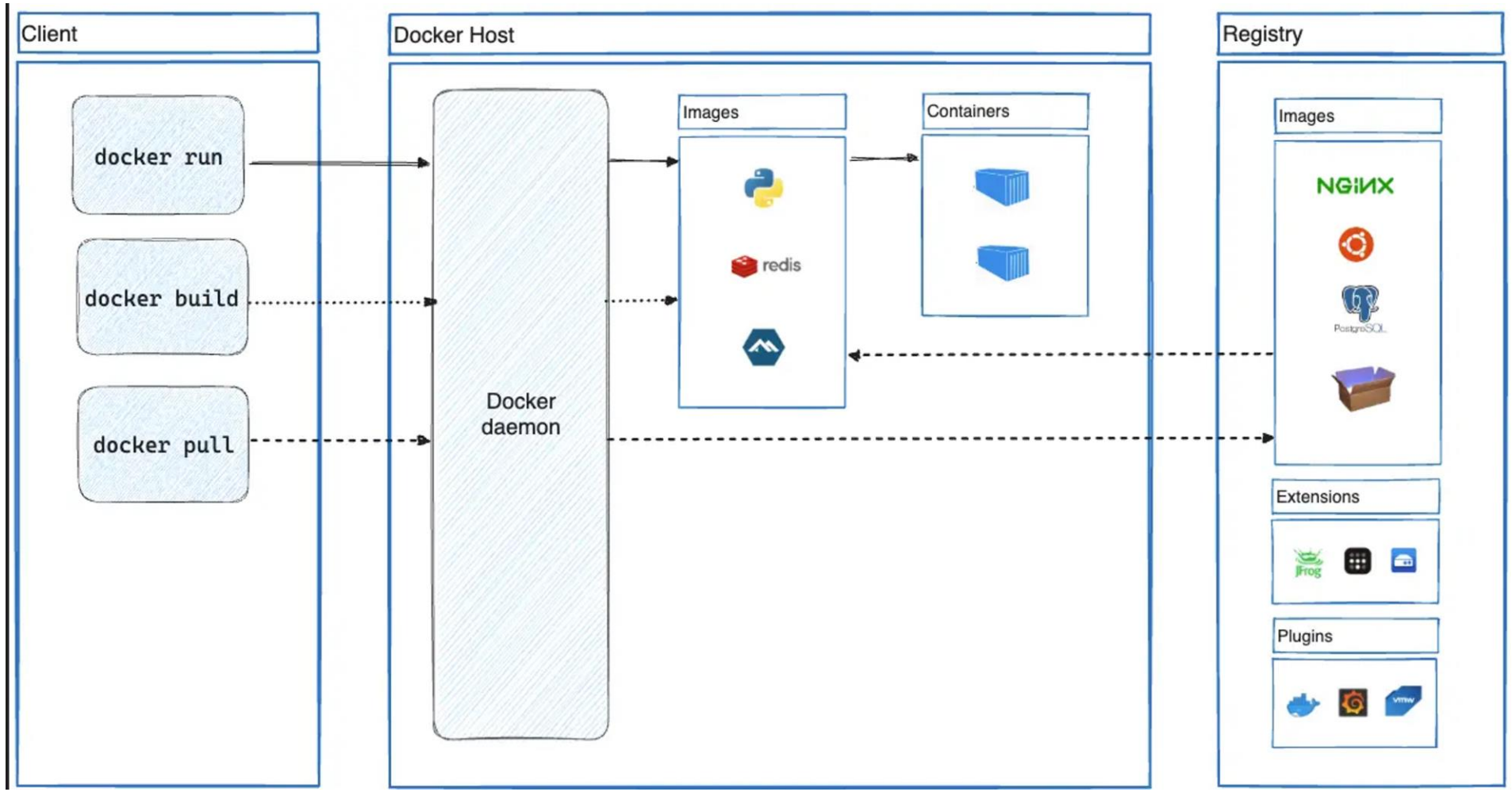
VM vs Container

- VM
  - abstraction of physical hardware: 1 server to many servers
  - hypervisors allows multiple VMs on single machine
  - each VM contains full OS, binaries, libraries, application
  - typically larger and slower to boot
- Container
  - abstraction at application layer
  - multiple containers can run on same machine as isolated processes
  - share host OS kernel
  - typically smaller and fast to launch
  - made up of layers





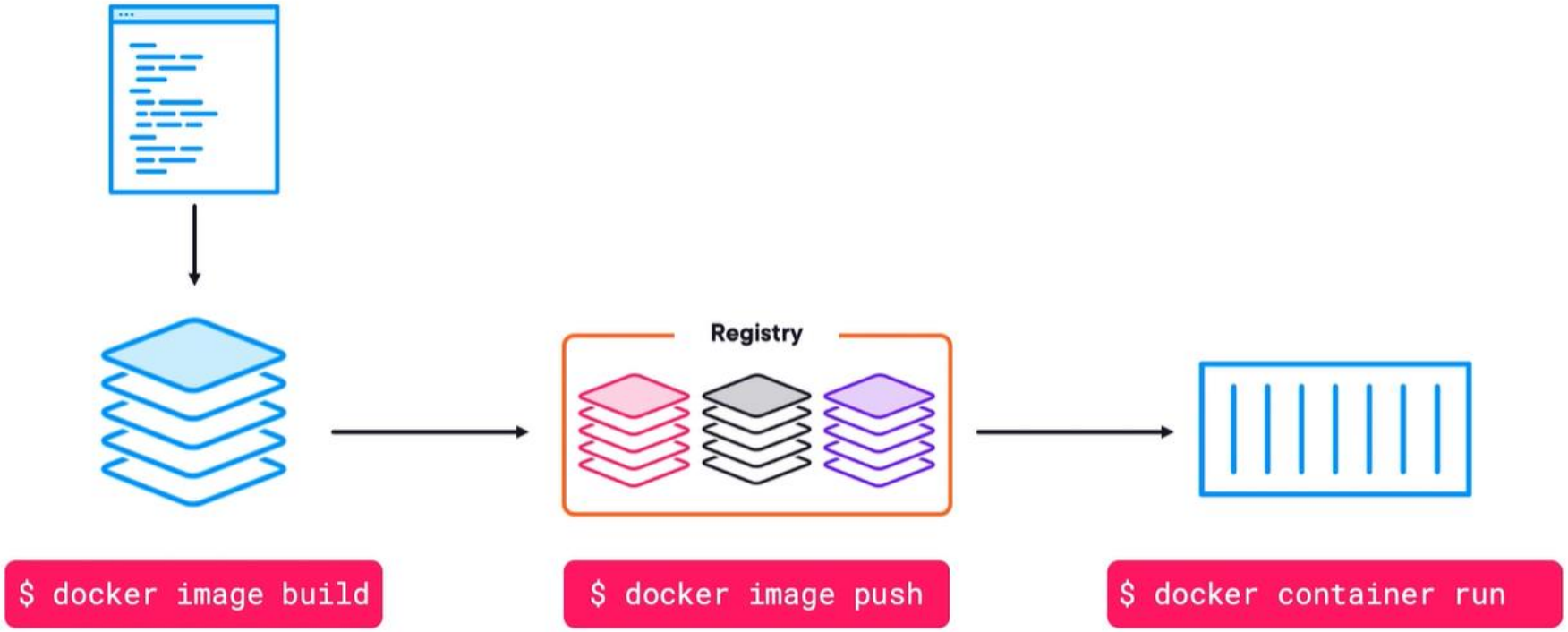
Architecture



Graphic from https://docs.docker.com/get-started/overview/

Workflow

- create starting build recipe
- create image in sandbox mode
- develop/debug image in sandbox mode
- update build recipe to final version
- create final image
- push image to registry
- test deployment of image via the registry



Docker vs Singularity/Apptainer

Docker

+ established orchestration options
  + docker-compose, Kubernetes, podman, docker swarm
+ existing registry: Dockerhub
+ Windows containers possible

- requires root access (hard to use on HPC)
- Windows: Linux containers via WSL only
- commercial

Singularity/Apptainer

+ open source
+ no root required (usable on HPC)
+ can use all Docker containers

- Windows: Linux containers via WSL only
- registry creation
- no easy orchestration

Singularity project split in 2021 into two forks: Singularity CE (https://sylabs.io/) and Apptainer (https://apptainer.org/). The original Singularity project is no longer being developed.

Apptainer definition file - Header

- definition (or `def`) file divided into two parts: Header and Sections
- blueprint for building a container
- mandatory: `Bootstrap` and `From`
- Bootstrap
  - agent used for creating base OS
  - depends on the container/OS you want to use
- From
  - determines OS to use
- additional options for multi-stage builds or specific URLs

```
Bootstrap: docker
From: ubuntu:{{ VERSION }}
Stage: build
```

Apptainer definition file - Sections

- different sections prefaced by `%` symbol to add different content/commands/features during build process
- none mandatory
- possible section names:
  - arguments, setup, **files**, **app***, **post**, test, **environment**, startscript, **runscript**, labels, **help**
- order of sections within definition file irrelevant
- order of execution is well-defined: order of section names above

```
Bootstrap: docker
From: ubuntu:{{ VERSION }}
Stage: build

%arguments
    VERSION=22.04

%setup
    touch /file1
    touch ${APPTAINER_ROOTFS}/file2

%files
    /file1
    /file1 /opt

%environment
    export LISTEN_PORT=54321
    export LC_ALL=C

%post
    apt-get update && apt-get install -y netcat
    NOW=`date`
    echo "export NOW=\"${NOW}\"" >> $APPTAINER_ENVIRONMENT

%runscript
    echo "Container was created $NOW"
    echo "Arguments received: $*"
    exec echo "$@"

%startscript
    nc -lp $LISTEN_PORT

%test
    grep -q NAME=\"Ubuntu\" /etc/os-release
    if [ $? -eq 0 ]; then
        echo "Container base is Ubuntu as expected."
    else
        echo "Container base is not Ubuntu."
        exit 1
    fi

%labels
    Author alice
    Version v0.0.1

%help
    This is a demo container used to illustrate a def file that uses all
    supported sections.
```

Examples

- login into IDAaaS and create a workspace
  - small CPU + RAM sufficient
    - e.g. Muon (EMU, MUSR, HIFI,…) or Excitations Powder (ALF, LET, MAPS, …)
- open a terminal and git clone repo for session
  - git clone https://github.com/franzlang/container_workshop.git
- slides and examples (with solutions) contained in repo

Example 1 – hello world

- make sure you can run Apptainer by downloading and running Docker's `hello-world` container
  - for example via apptainer run docker://hello-world
- learn to:
  - specify a OS
  - build your own minimal image
  - make your image perform a basic task
  - run your minimal image in a container
- %help
  - metadata for helpful information about the container
- %runscript
  - commands executed when container is run
  - $*: options passed to container at runtime
  - $@: passing the options to a command via a quoted array

```
$ apptainer build alpine.sif docker://alpine
```

```
$ apptainer build lolcow.sif lolcow.def
```

```
apptainer run [run options...] <container> [args...]
```

```
%help
    This is a demo container used to illustrate a def file that uses all
    supported sections.
```

```
%runscript
    echo "Container was created $NOW"
    echo "Arguments received: $*"
    exec echo "$@"
```

Example 2 – basic shell commands

- learn to:
  - use the sandbox feature to build/develop/debug a container interactively
  - open a shell inside a container to run commands inside the container
  - use %post to run shell commands when building the image
  - use %environment to specify environment variable(s)
- %post
  - commands run during image creation
- %environment
  - setting environment variables within the container

```
apptainer shell [shell options...] <container>
```

```
%post
    apt-get update && apt-get install -y netcat
```

```
%environment
    export LISTEN_PORT=54321
    export LC_ALL=C
```

Example 3 – files

- learn to:
  - use the %files section to add a file from the host system to the container image
- %files
  - copying files from a stage during the build to another stage
    - e.g.: compile stage creates binary, which is copied to final stage
  - copying files from host machine into the image

```
%files [from <stage>]
    <source> [<destination>]
    ...
```

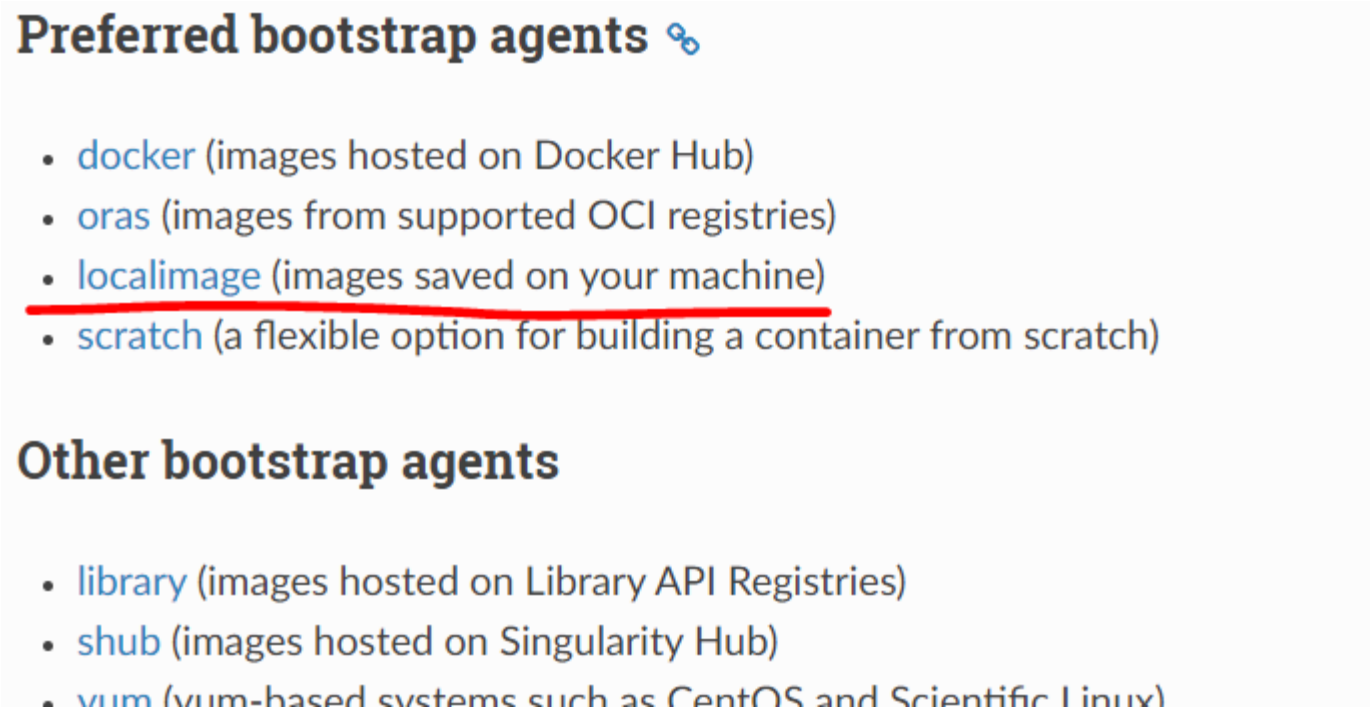
```
%files
    /file1
    /file1 /opt
```

Example 4 – multi-stage build

- learn to:
  - use multiple build stages using the `Stage` part of the Header
  - use files from one build stage, e.g. compiled binaries, in another build stage
- advantage: can deploy compiled binaries without deploying build dependencies

Example 5 – multi-image build

- learn to:
  - use a custom-built image as the base image for a follow-on image
  - use a local image in the build process via the `From` part of the Header
- advantage: can separate complicated images into parts
  - e.g.: parts that rarely change (compiler, simulation engine, …) and parts that change regularly



Preferred bootstrap agents

- docker (images hosted on Docker Hub)
- oras (images from supported OCI registries)
- localimage (images saved on your machine)
- scratch (a flexible option for building a container from scratch)

Other bootstrap agents

- library (images hosted on Library API Registries)
- shub (images hosted on Singularity Hub)
- yum (yum-based systems such as CentOS and Scientific Linux)

Example 6 – multi-app image

- learn to:
  - allow multiple applications to be callable from the same container
  - forward input to the application inside the container
- advantage: can run multiple binaries/apps from the same container

```
%apprun foo
    exec echo "RUNNING FOO"

%appstart foo
    exec echo "STARTING FOO"

%applabels foo
    BESTAPP FOO

%appinstall foo
    touch foo.exec

%appenv foo
    SOFTWARE=foo
    export SOFTWARE

%apphelp foo
    This is the help for foo.
```

```
% apptainer run --app foo my_container.sif
RUNNING FOO
```

Example 7 – mounting volume

- learn to:
  - mount a local volume of the host machine within the container
- advantage: have a shared filespace between the container and the host

```
$ ls /data
bar  foo

$ apptainer exec --bind /data:/mnt my_container.sif ls /mnt
bar  foo
```

Example 8 – GUI/X11

- example to illustrate how a container can launch a graphical element

Example 9 – Jupyter

- learn to:
  - build a minimal python container to run a Jupyter notebook server
  - run a Jupyter notebook server in the container
  - access the server via the host's browser
  - run a notebook on the host system using the containerised notebook server

Real-life examples

- included a number of examples used to deploy applications on IDAaaS
- containers will take too long to build
  - or will not build if I had to sanitise the definition file
- look for the useful Apptainer/container feature that the example tries to illustrate

Registry

- upload images to a cloud/server to allow easier deployment
- STFC cloud has a Harbor registry
  - https://harbor.stfc.ac.uk/
  - login via IRIS IAM and fedID+password
- IDAaaS has dedicated repository
  - can gain push access to specific project
  - allows easy deployment on IDAaaS
- can hook registry into github actions for easy CI/CD
- size limit: 10GB images (Harbor)

Orchestration

- multi-container applications/services
- docker-compose
  - tool to automate command-line options for Docker
  - YAML style
  - mounting/binding of directories
  - networking
  - configuration settings
  - secrets
  - shell commands to be run when starting container
- Kubernetes
  - target: large clusters and services
  - typically: microservice architecture (each service a container)
  - automatic load-balancing, self-healing, scalability, …

Wrap-up

- Questions?
- Feedback?
- time left to try out something yourselves?