ResBaz
RESEARCH BAZAAR

Containers in Research

Hands-on introduction to leveraging containers in research

Luis Gracia, PhD
lgracia
luisico

July 2024

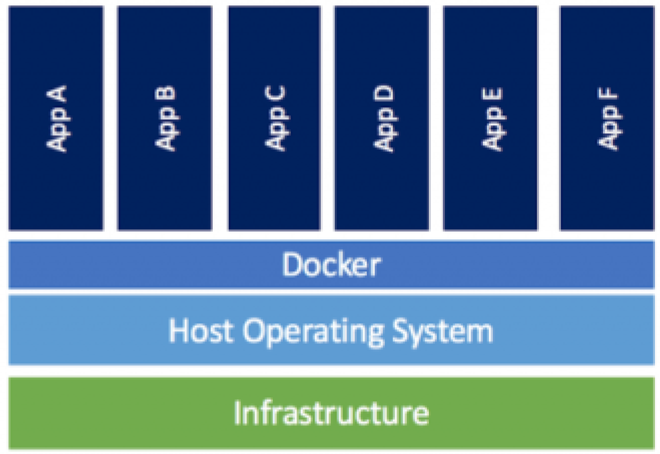
Agenda

- What to expect from this workshop
- Introduction to containers
- Why should we care in scientific research
- How to get started – Hands on
  - Running containers
  - Creating container images
  - Orchestrating containers

What are Containers?

*"A container is a standard unit of software that **packages up code and all its dependencies** so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings."* – docker.com

- Fast startup, low resources
- "Isolated" running context
- Can be further configured at runtime
- Standard (OCI)



credits: docker.com

Packaged as an Image, runs as a Container

Depends on the OS kernel:

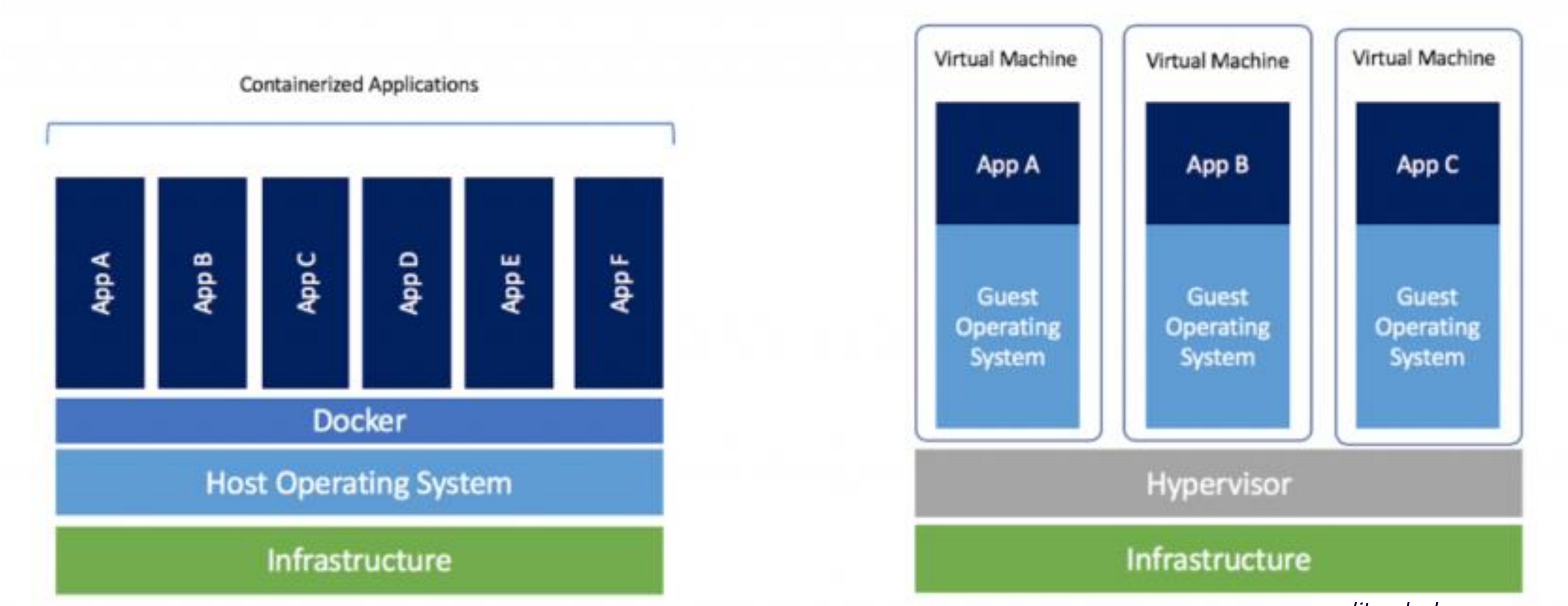
- **Linux**
- Mac
- Windows

Why should we care in research?

- Reproducibility
  - App and dependencies are pre-installed and configured in the image
  - Defined in code (track, version)
  - Use versioning to run old workflows years later
  - Hardware and kernel (OS) changes can affect reproducibility
- Portability and collaboration
  - Compile/Package once, run multiple times
  - Run same image in different systems:
    - laptop, HPC, cloud
    - Development, test, production
  - Share software or pipelines with colleagues
  - Use code from lab mates
  - Other labs can test/review your code/results
  - Run standard software or develop your own

- Isolated and conflict-free environment
  - Avoid "dependency hell" and conflicts between libraries, languages and dependencies
  - Use different versions of python simultaneously
  - Test different versions of a library
  - Run different environments simultaneously (i.e. tests parameters)
- Near native operation
  - Low computing resources (memory, cpu)
  - Fast start
  - Run multiple containers simultaneously
- Alternatives:
  - Self managed
  - Software repositories
    - Distro packages (deb, rpm)
    - Distro-agnostic packages: Flatpack, Snaps, AppImage
    - Version managers: nvm, nodenv, Renv, rbenv
      - Language specific:
      - General: asdf, mise, anyenv
  - VMs

Containers vs VMs



credits: docker.com

Very fast	Startup	Slow
Low mem/cpu	Resource usage	High mem/cpu (reserved)
Small (lots of tricks to reduce it)	Size	Big
Yes, but can get tricky	Isolation	Yes
App and dependencies; runs on same OS	Portability/Maintenance	OS, app and dependencies; runs anywhere
Yes	Versioning	Yes
Easy	Provenance	Difficult
Easy	Orchestration	Difficult

Hands-on Instructions

- How to follow along with Docker
  - Use provided VMs at https://www.container-workshop.cloud.edu.au/guacamole
    - One pre-allocated VM per person, login with your credentials
      - ❖ Username: the complete name you registered with
      - ❖ Password: email address you registered with
    - Web based terminals
    - Instructors can access your terminal to help troubleshooting (pair programming style)
    - No GUI
    - No direct access to VM via SSH or browser
    - Access configuration with `CTRL+SHIFT+ALT` (or `CTRL+SHIFT+OPT` on Macs)
  - Docker installed on your own computer (https://docs.docker.com/get-docker)
  - Play with Docker online
- Content at https://github.com/UoA-eResearch/resbaz-2024-containers
- Need help?
  - We assume you have some Linux shell terminal knowledge
  - Contact us (Luis, Andre, Jason) via Zoom chat
  - Feel free to contact me (https://profiles.auckland.ac.nz/luis-gracia-valen) via email after the session
- Docker alternatives
  - Podman
  - Apptainer (Singularity)
  - LXC

Hands-on: some examples

- Hello world: sum "2 + 3"
  - Python
  - Node.js
  - R
- Interactive R
- Serving websites with Nginx
- Local RStudio via browser

Docker main commands

Reference: https://docs.docker.com/reference/cli/docker

```
docker
  run          # Create and run a new container from an image
  ps           # List containers
  rm           # Remove one or more containers
  kill         # Kill one or more running containers
  exec         # Execute a command in a running container
  pull         # Download an image from a registry
  push         # Upload an image to a registry
  image        # Operations with images
  cp           # Copy files/folders between a container and the local filesystem
  logs         # Fetch the logs of a container
```

Running Containers

```
docker run [options] image [command]
  --rm                                          # Remove the container when it exits
  --interactive -i                              # Interactive session
  --tty         -t                              # Allocate a pseudo-TTY

  --name             name                       # Assign a name to the container
  --detach      -d                              # Run container in background
  --volume      -v   host_dir:container_dir     # Share a host directory inside the container

  --workdir     -w   container_dir              # Starting directory inside the container
  --port        -p   host_port:container_port   # Publish a container's port(s) to the host
  --env         -e   VAR=value                  # Set environment variable inside container
  --env-file         /path/to/file              # Pass environment variables from file

  --user        -u   user                       # Use username in container
  --entrypoint       /path/to/executable        # Overwrite the default entrypoint of the image
```

Hands-on: running containers

- Running, listing and cleaning containers
- Managing container names
- Executing non-default commands
- Running interactive containers
- Running containers in the background
- Exposing container ports to the host machine (or the internet)
- Configuring containers: sharing files and variables
- Configuring commands (“entrypoints”)
- Dealing with permissions when sharing files

Container Images

- A container is an instance of an image
- When a container stops, changes are lost (i.e. pip install)
- Naming convention: `registry/org/name:tag`
  - `registry`:
    - Docker Hub: https://hub.docker.com (default)
    - Quay: https://quay.io/search
    - Git platform: GitHub, GitLab
    - Cloud: AWS, Google Cloud, Azure
    - Private
  - `org/name`:
    - Usually called "repository" or image name
    - Only `name` is mandatory
  - `tag`:
    - Default is "latest"
    - Arbitrary text, no special meaning

Examples:

- ubuntu
- ubuntu:24.04
- ubuntu/mysql
- quay.io/quay/ubuntu
- localhost:5000/testing/ubuntu:20240709

Images main commands

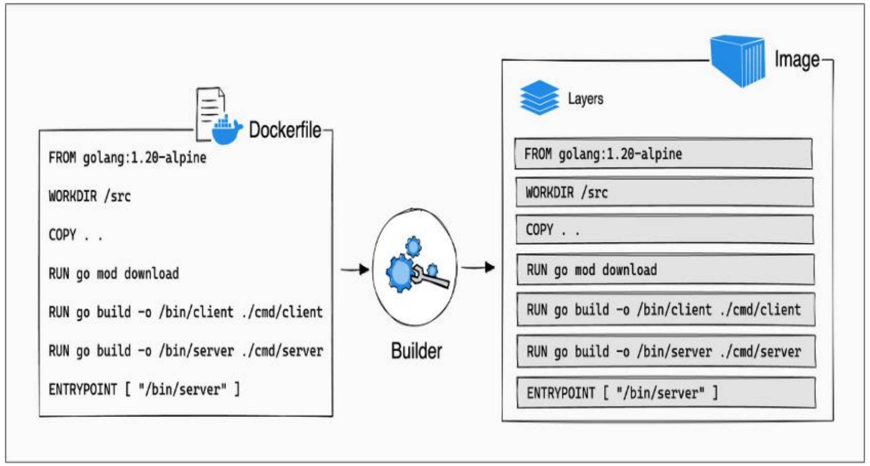
```
docker image
  ls        # List images (or `docker images`)
  pull      # Download an image from a registry
  push      # Upload an image to a registry
  rm        # Remove an image
  prune     # Remove unused images
  build     # Build an image from a Dockerfile (`docker build`)
  inspect   # Inspect properties of an image (`docker inspect`)
  history   # Show the build history of an image
  save      # Export and image as a tar archive
  load      # Import a tarball as an image
```

Pulling (downloading) Images

- `docker pull image`
  - Image is downloaded layer by layer, existing layers being reused
  - `docker run` will pull the image if not present locally, but it will not check if there is a newer version in the registry
  - `docker pull` will download newer versions of an image already present locally
- Inspect: `docker inspect image`
- Keep an eye on disk space: `docker system df`

Building Container Images

- Dockerfile: define how the image is built using code:
  - Selecting a base image to start with
  - Adding, removing and changing files
  - Running commands
  - Defining ports, volumes, variables, commands, etc…
  - Reference: https://docs.docker.com/reference/dockerfile
- Layers:
  - New layer every time the filesystem is changed (i.e. adding files, running a command that changes files)
  - Layers are downloaded independently and composed on top of each other to create the final image
  - Shared among images, reducing disk space
- Metadata:
  - Labels (authorship, copyright, versions)
  - Ports
  - Volumes



```
FROM golang:1.20-alpine
WORKDIR /src
COPY . .
RUN go mod download
RUN go build -o /bin/client ./cmd/client
RUN go build -o /bin/server ./cmd/server
ENTRYPOINT [ "/bin/server" ]
```

```
FROM golang:1.20-alpine
WORKDIR /src
COPY . .
RUN go mod download
RUN go build -o /bin/client ./cmd/client
RUN go build -o /bin/server ./cmd/server
ENTRYPOINT [ "/bin/server" ]
```

credits: docker.com

```
docker build [options] path
  --tag        -t   Tag name           # Image name
  --file       -f   Dockerfile         # Path to Dockerfile
  --progress        auto|plain|tty     # Verbosity
  --build-arg       ARG=value          # Pass build argument
  --no-cache
```

Hands-on: Images

- Finding images in Docker Hub
- Pulling images locally
- Creating images from a Dockerfile
  - Develop and share own software (*Campsites*)
  - Package open source software (*Fastutils*)
  - Compile open source software (*Open Babel*)
- Sharing images
  - Push to registry (need an account in Docker Hub or another registry)
  - Save/Load

Hands-on: DOC Campsites

Dockerfile

```
FROM python:3.12.4-alpine3.20
WORKDIR /app
ADD requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
ADD . .
CMD ["python", "tally.py"]
```

requirements.txt

```
pandas==2.2.2
```

data.csv

```
Region,Number of powered sites,Number of unpowered sites
Central North Island,0,12
Fiordland,0,5
```

tally.py

```
data = pd.read_csv('data.csv', header = 0, usecols =
                   ['Region',
                    'Number of powered sites',
                    'Number of unpowered sites'
                    ])
print("Campsites by region")
print("-------------------")
campsites_by_region = data.groupby('Region').Region.count()
print(campsites_by_region)
print("")
print("Site types per region")
print("---------------------")
site_types_per_region = data.groupby('Region').sum()
print(site_types_per_region)
```

Hands-on: Sharing Images

- Push to registry
- Create account in Docker Hub
  1. Login into your account from your machine
  2. Tag image to include your `org` (your account name)
  3. Push (upload) your image

```
docker login
docker tag campites:1.0.0 org/campsites:1.0.0
docker push org/campsites:1.0.0
```

- Save / Load (manual process)

```
docker image save campsites:1.0.0 -o campsites-v1.0.0.tar
```

↓ Transfer (ssh, rsync, shared drive, etc)

```
docker image load -I campsites-v1.0.0.tar
```

Hands-on: Fastutils

Dockerfile

```
FROM debian:bookworm-slim

RUN apt-get update \

    && apt-get install -y git make g++ zlib1g-dev \

    && rm -rf /var/lib/apt/lists/*

RUN git clone https://github.com/haghshenas/fastutils.git /tmp/fastutils \

    && cd /tmp/fastutils \

    && make \

    && cp fastutils /usr/local/bin \

    && rm -rf /tmp/fastutils

WORKDIR /app

CMD ["/usr/local/bin/fastutils"]
```

Dockerfile.alpine

```
FROM alpine:3.20 AS builder

RUN apk --no-cache add \

    g++ \

    git \

    make \

    zlib-dev \

    zlib-static

RUN git clone https://github.com/haghshenas/fastutils.git /tmp/fastutils \

    && cd /tmp/fastutils \

    && env LDFLAGS=-static make

FROM alpine:3.20

COPY --from=builder /tmp/fastutils /usr/local/bin/

WORKDIR /app

CMD ["/usr/local/bin/fastutils"]
```

Hands-on: Open Babel

Dockerfile

```
FROM alpine:3.20
ARG OPENBABEL_REF=master
RUN apk --no-cache add \
    git \
    g++ cmake make perl \
    boost-dev cairo-dev eigen-dev libxml2-dev zlib-dev
RUN git clone https://github.com/openbabel/openbabel.git /tmp/openbabel \
    && cd /tmp/openbabel \
    && git checkout ${OPENBABEL_REF} \
    && env CXXFLAGS="-Wno-deprecated-declarations" cmake . \
    && make -j2 \
    && make install \
    && rm -rf /tmp/openbabel
ENTRYPOINT ["/usr/local/bin/obabel"]
CMD [""]
```

Orchestrating Containers

Docker Compose:

- Declarative instructions to run/orchestrate multiple images with custom configuration
- Reference: https://docs.docker.com/compose/compose-file/
- Per directory

docker-compose.yml

```yaml
services:
  app:
    image: best-app:latest
    environment:
      VAR1: value1
    ports:
      - '8000:80'
    depends_on:
      - database
  database:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: secure
```

```
docker compose
  up          # Create and start services
  down        # Stop and remove services
  ps          # List containers
  exec        # Execute command in container
  pull        # Pull (download) images
  logs        # View output from containers
  build       # Build images
```

Hands-on: RStudio

Setup RStudio with docker compose and pre-installed packages

docker-compose.yml

```
services:
  rstudio:
    image: rocker/verse:4.4
    restart: unless-stopped
    environment:
      PASSWORD: rstudio
    ports:
      - '8787:8787'
    volumes:
      - .:/home/rstudio
```

docker-compose.yml

```
services:
  rstudio:
    image: my-rstudio
    build: .
    restart: unless-stopped
    environment:
      PASSWORD:
    ports:
      - '8787:8787'
    volumes:
      - .:/home/rstudio
```

+

Dockerfile

```
FROM rocker/verse:4.4
RUN install2.r --error --skipinstalled \
    janitor \
  && rm -rf /tmp/downloaded_packages
```