DS-I Africa
Data Science for Health Discovery and Innovation in Africa

Introduction to Containers

Denis Yuen
Ontario Institute for Cancer Research - Dockstore

eLwazi
OPEN DATA SCIENCE PLATFORM

Learning Objectives

- Introduction to containerization
- Docker basics
- Docker security best practices
- Q&A
- Containerization practical session

Introduction to containerization

What is Docker?

Docker - platform for building, running, sharing applications

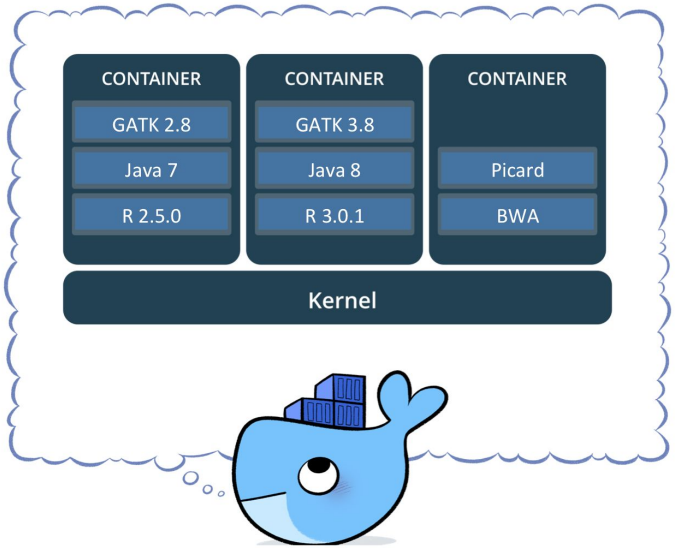
Runs software in containers, portable packages that include software and all its dependencies

Why do we need containers?

Code breaks when we use
different computing environments or
When dependencies update!

Docker is only one container type!

Widely supported commercially
but some HPC users may wish to explore alternatives



Docker Concept Recap: Image, Container Registry

Image:

Packaged code + dependencies.

- Portable software
- Runs quickly and reliably

Official Image

- Regularly updated and scanned for vulnerabilities
- Docker Official Images

Container:

A *running* image

- Packaged up, isolated software environment
- Each one takes a small amount of space and resources

Registry:

Cloud repositories to store images privately or publicly:

- Docker Hub
- Quay.io
- AWS ECR
- Google Artifact Registry (GAR)

***The terms container and image are often used interchangeably, but there is a distinction.*

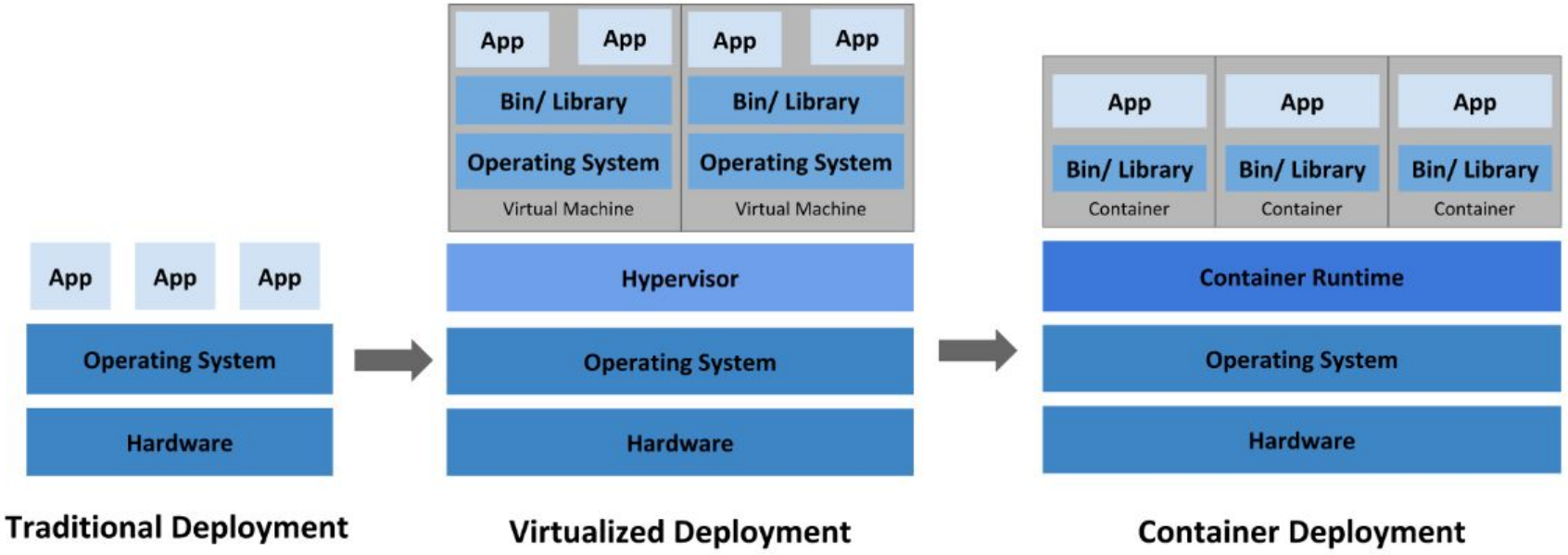
Containerization use cases

- DevOps
  - Testing and Quality Assurance testing of applications
  - Deployment of software systems (e.g. Dockstore!)
- Microservices
  - Components of an application/tool is built and managed separately
  - Upgrading a specific application or service does not necessarily impact the overall application
- Content delivery
  - Relatively easy and quick to deploy web content
- Immutable infrastructure
  - infrastructure that does not change
  - When upgrading a component replace instead of upgrade
- HPC
  - Allows for efficiency, reproducibility and scalability of computational workloads

What specific kinds of problems are solved by containers?

- Installation problems
  - Software was built on a different OS
  - Install documentation is unclear or out of date
- Dependency problems
  - Software requires different version than what is available on machine (ex. Java, Python)
  - Multiple programs have shared dependencies, but conflicting different versions of those dependencies

Evolution of Container Architecture



Gerrit Botha - https://docs.google.com/presentation/d/1yDp5nruen5dDZwpkwJu6Sq6TdqlBtZTcmbetsJthBoM/edit#slide=id.gae7c8e1e2c_0_3

Types of Container Platforms

Docker

Pros

- Docker is the most commonly used container platform
- It's user-friendly and relatively easy to learn for use on single-user environments
- It has a vast repository of pre-built container images
- Huge user base for support
- Can easily be converted to Singularity allowing one image to function across two container platforms

Cons

- Originally designed for single host environments
- Does not align well for HPC use
- Not great for security, especially in multi-user environments > needs to run as root
- Images are managed via the docker engine > requires some tweaking on HPC
- Introduces network complexities, especially in multi-node HPC

Types of Container Platforms

Singularity

Pros

- Specifically designed for HPC cluster environments
- Does not require root access > run based on user's privileges
- Smaller ecosystem than docker containers
- Has native support for interconnects like InfiniBand and is suitable for MPI-based parallel computing workloads
- Simplified configuration > does not require complex network configurations
- Singularity images are stored as normal files on the file system > easier to manage

Cons

- Widely adopted for HPC but not well adopted for end-user environments > limiting community support
- Primarily designed for HPC environments and does not align well for end-user environments
- Does not port well into Docker

Types of Container Platforms

Podman

Pros

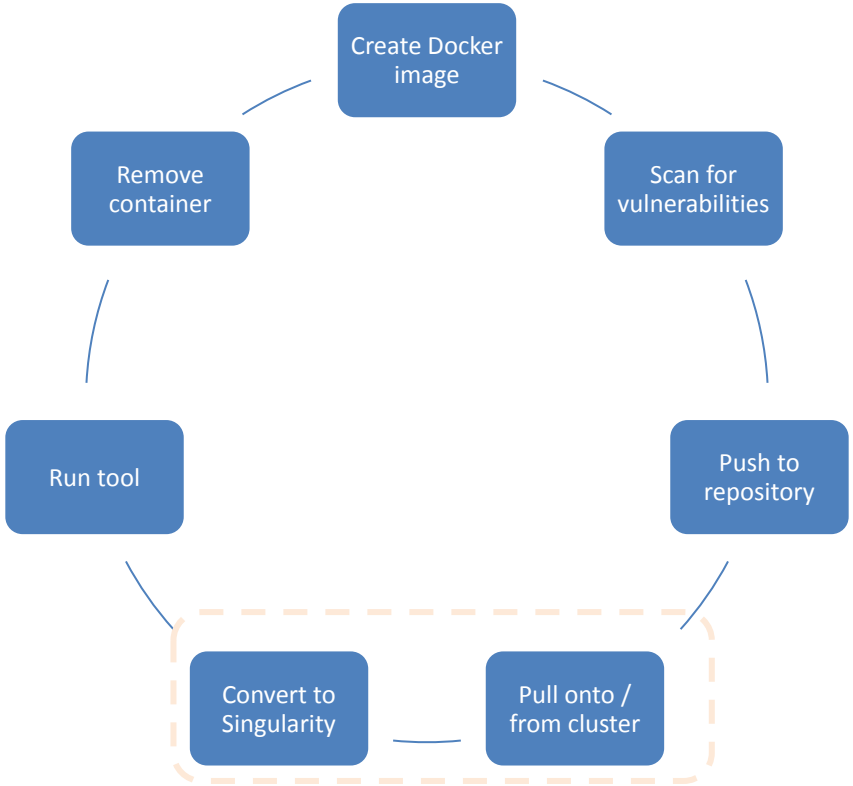
- Provides all the benefits of Docker but is more secure
- Provides features for managing single containers and pods (groups of containers)

Cons

- While Podman is similar to docker, it still requires a bit of a learning curve
- Can be challenging to maintaining the networking complexities in HPC environments > similar to Docker
- Podman community and ecosystem not as extensive as Docker

Docker basics

Docker HPC lifecycle



Docker Basics : Exploring the CLI

- Live demo
  - Run ubuntu in an ad hoc container
    - Demonstrate how environment is transient
  - Show images
  - Show stopped containers
  - Build a container and tag it
  - Enter container
  - Clean up individual images and containers
  - Clean-up system

Docker Basics : Create Dockerfile/Tool

A simple text file with instructions to build an image:

- YAML syntax
- Start from a base image
- Metadata
- Install software and dependencies
- Set up scripts
- Other environment prep
- Define commands to run when container starts

Docker Basics : Exploring the CLI

- Demo
  - Layers and cache busting
    - Each layer is saved individually so sometimes you'll want to be careful
    - e.g. download a large zip, extract what you need, and delete the rest in one command
    - Line 48 in dockstore Dockerfile

Docker Basics : Exploring the CLI

Command	Description
docker --version	See what version of Docker is installed on your system.
docker info	Displays system-wide information about Docker.
docker --help	Get help and information about Docker commands.
docker images	Lists all available Docker images on your system. Useful for getting the size of a specific container.
docker ps	Lists all running containers.
docker ps -a	Lists all running and stopped containers. Similar to "docker images" but provides more information.
docker <function> ls	Supplement "function" with "images", "network", etc. for information about the function

Docker Basics : Exploring the CLI

Command	Description
docker build -t <image-name:tag> <path-to-Dockerfile-directory>	Build a docker image from a Dockerfile.
docker push <image-name:tag>	Upload a Docker image to an online repo, e.g. Docker Hub, Github, etc.
docker logs <container-name or ID>	View the logs generated by the specific container. Useful for troubleshooting. Add the "-f" option to view logs in real-time.
docker exec -it <container-name or ID> <command>	Run the specified command inside a running container.
docker rm <container-name or ID> docker rmi <image-name or ID>	Removes the specified container. Add "-f" to force the removal of a running container.
docker stop <container-name or ID>	Stops the specified running container

Docker Basics : Exploring the CLI

Command	Description
docker start <container-name or ID>	Starts the specified container.
docker restart <container-name or ID>	Restarts the specified container.
docker pull <image-name>	Download a Docker image for a registry. When you don't provide a repo and login credentials, the image is pulled from docker hub.
docker run <options> <image-name>	Create and start a new container from an image/Dockerfile. -d :runs the container in detached mode -it :starts and interactive session with the container --name <container-name> :assigns a custom name to the container -p <host-port>:<container-port> :maps ports between the host system and the container

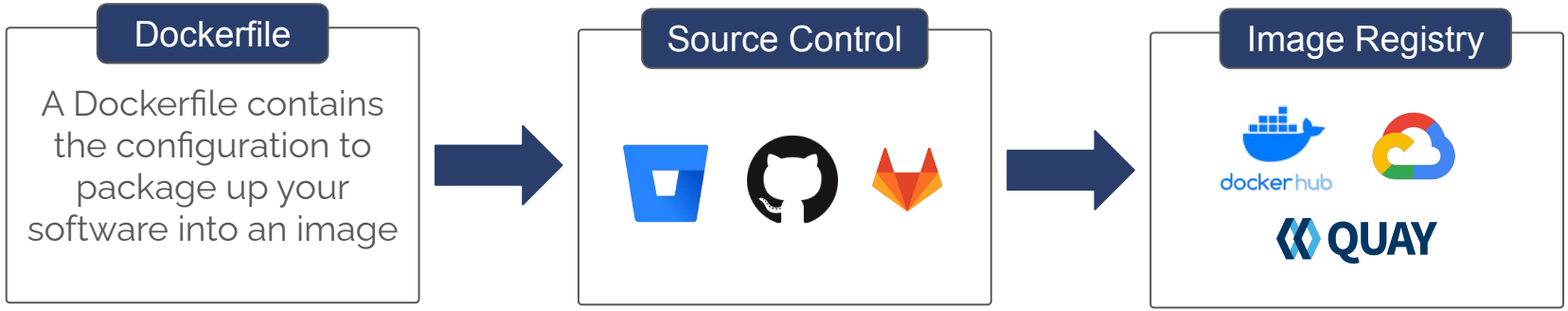
Docker Basics : Create Dockerfile/Tool

- Typically, OS based on a watered down base image that you pull down from an online repository
- Docker images can be created on various operating systems
  - Ubuntu – bigger image
  - Debian – bigger image
  - Alpine – smaller image but tricky with fewer tools built-in
- many ways of installing a tool into a Dockerfile
  - via the local apt repository
  - dpkg
  - build from source
  - via maven. pip, nodule modules, etc.

Best Practices (Practical Session)

- Only add what you need (keep containers light),
  - One way to do this is to use multiple containers in your workflows
  - Recommend not including large reference data within containers
    - Instead provide that data at runtime through data binding or via platform
  - Consider multi-stage builds to keep your images small
    - e.g. one stage can build executables or artifacts used in a second stage
      - e.g. build Java jars in an environment with Maven and then run them in a simpler environment
- Version your containers for re-use later (use --tag when building)

Sharing your Dockerfiles and Images



Dockstore recommends storing your Dockerfile in an external repository (Bitbucket, GitHub, GitLab) and then registering your source controlled Dockerfile to an image registry (Docker Hub, Quay.io, Google Container Registry, etc).

Sharing Images demo

- Manual image uploading demo
  - Build, tag
  - Push to docker hub
- Stretch goal for practical session following
  - Upload to Docker Hub or GitHub packages automatically from GitHub
    - **Publishing Docker images - GitHub Docs**
    - Demo example at https://github.com/dockstore/dockstore-tool-bamstats but you should probably create your own
      - Show GitHub action template (change username)

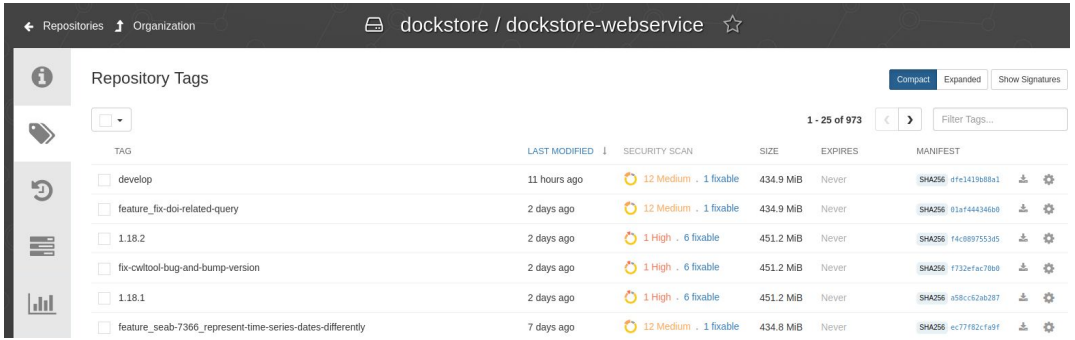
Docker security best practices

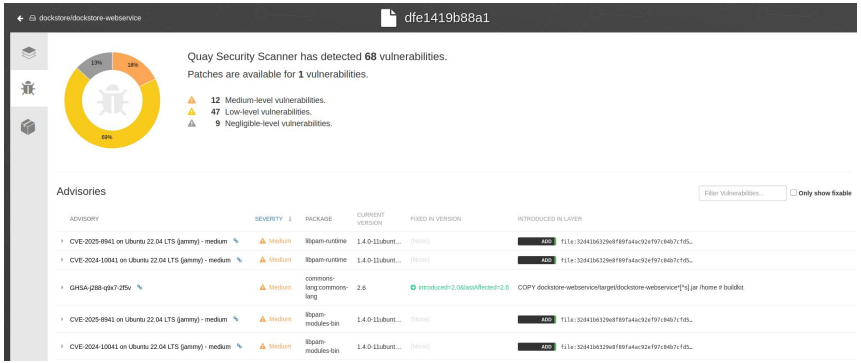
Docker Security Best Practices

- Use small images
- Use only trusted images
- Docker image scans for vulnerabilities
- Don't mount unnecessary volumes
- Be mindful of the content of your docker file when uploading to public repositories : secrets, private keys, passwords, etc

Container Vulnerability Scanners

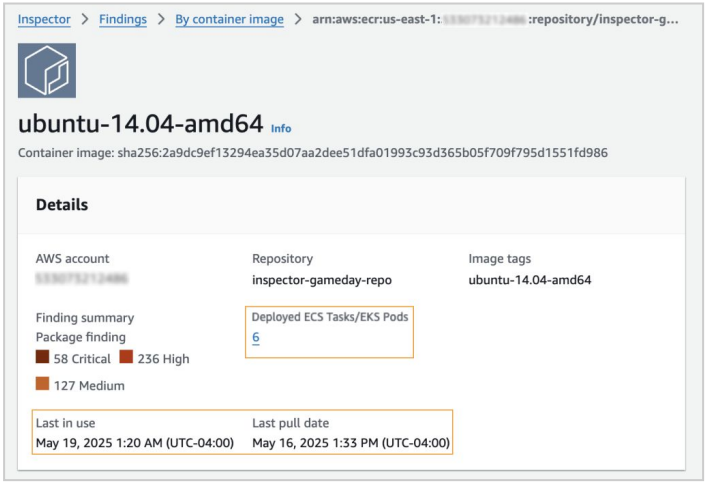
- Quay.io
  - Free for public use
  - Security scanner breaks down issues by layer/command
  - e.g. Dockstore





Container Vulnerability Scanners

- Another handy choice if you're working in a private environment (how Dockstore is hosted)
  - AWS ECR (Elastic Container Registry) scans for vulnerabilities and maps them to running containers
  - more info



Pre-made Bioinformatics containers

- Don't always start from scratch (Reusable afterall)
- Consider pre-baked containers from sources such as
  - BioContainers
  - DNAstack https://github.com/DNAstack/bioinformatics-public-docker-images
  - Containers referenced in nf-core workflows
  - Or even from other workflows on Dockstore and WorkflowHub

Key Takeaway for the week

- Develop in Docker
- Convert to Singularity for HPC environments
- Always use trusted base images and tools
- Always scan images for vulnerabilities and malware before introducing it to your production systems
- Consider building and uploading your Docker images automatically

Q&A

Containerization practical session

Hands-on Exercise

- Create/Pull a docker image: Ubuntu or Alpine base
- Install a Bioinformatics tool: apt, snap, from source, etc.
- Push it to docker hub or quay.io repository
- Log into Ilifu > pull the image from docker hub and convert it to singularity
- Test that your tool work
- Automatically build and upload your containers from GitHub using GitHub actions

Extra slides

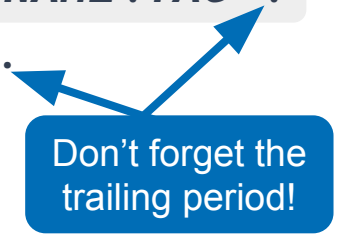
Exercise 4: Downloading, building, and pushing your Docker Image

1. **Download** the example Dockerfile into a **new folder** on your local machine (ex. my_project)
2. Open a **terminal window** and navigate to the **new folder**
3. Run the command

   ```
   docker build -t DOCKERHUB_ACCOUNT_NAME/REPO_NAME:TAG .
   ex. docker build -t ekiernan/docker-101:v1 .
   ```

   Mac 2021 and later:

   ```
   docker build --platform linux/amd64 -t
   ```

4. Run the command

   ```
   docker push DOCKERHUB_ACCOUNT_NAME/REPO_NAME:TAG
   ex. docker push ekiernan/docker-101:v1
   ```

Exercise 5: Clean up your images

1. If you just want to test something quick, run container using a --rm parameter to remove container after exiting
2. If you want to keep your container, you can clean up after your done:
    - Check containers on computer: `**docker ps -a**`
    - Check images: `**docker image ls**`
    - Remove all images and containers: `**docker system prune -a**`
    - Remove volumes: `**docker system prune –volume -a**`

WARP repository has best practices for building dockers:

- Reduce image size
  - Start with small images (like Alpine)
  - Minimal run steps
    - Might want more than one if you're pulling a giant piece of software
    - Layers cache

```
FROM alpine:3.9

RUN set -eux; \
        apk add
--no-cache \
            curl \
             bash \
```

WARP Docker Best Practices Quick-start

WARP repository has best practices for building dockers:

- Make dockers publicly accessible
- Use semantic tagging
  - major.minor.patch

```
us.gcr.io/broad-gotc-prod/samtools:<image-version>-<samtools-version>-<unix-timestamp>
```

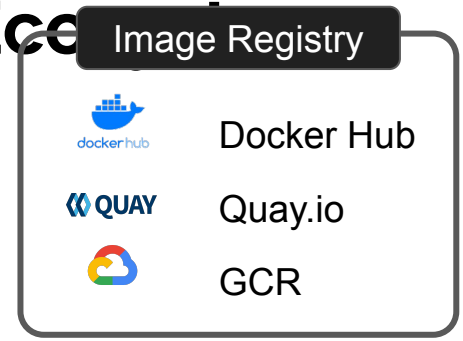
```
us.gcr.io/broad-gotc-prod/samtools:1.0.0-1.11-1624651616
```

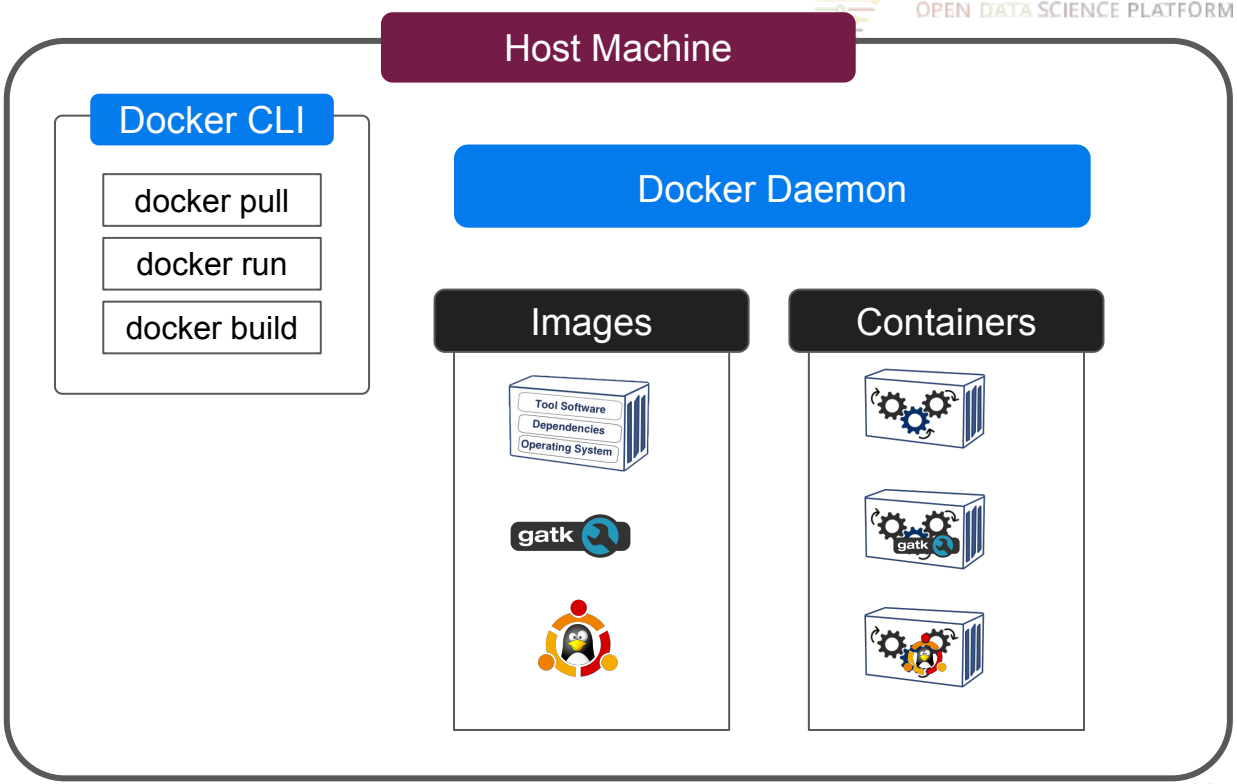
WARP Docker Best Practices Quick-start

WARP repository has best practices for building dockers:

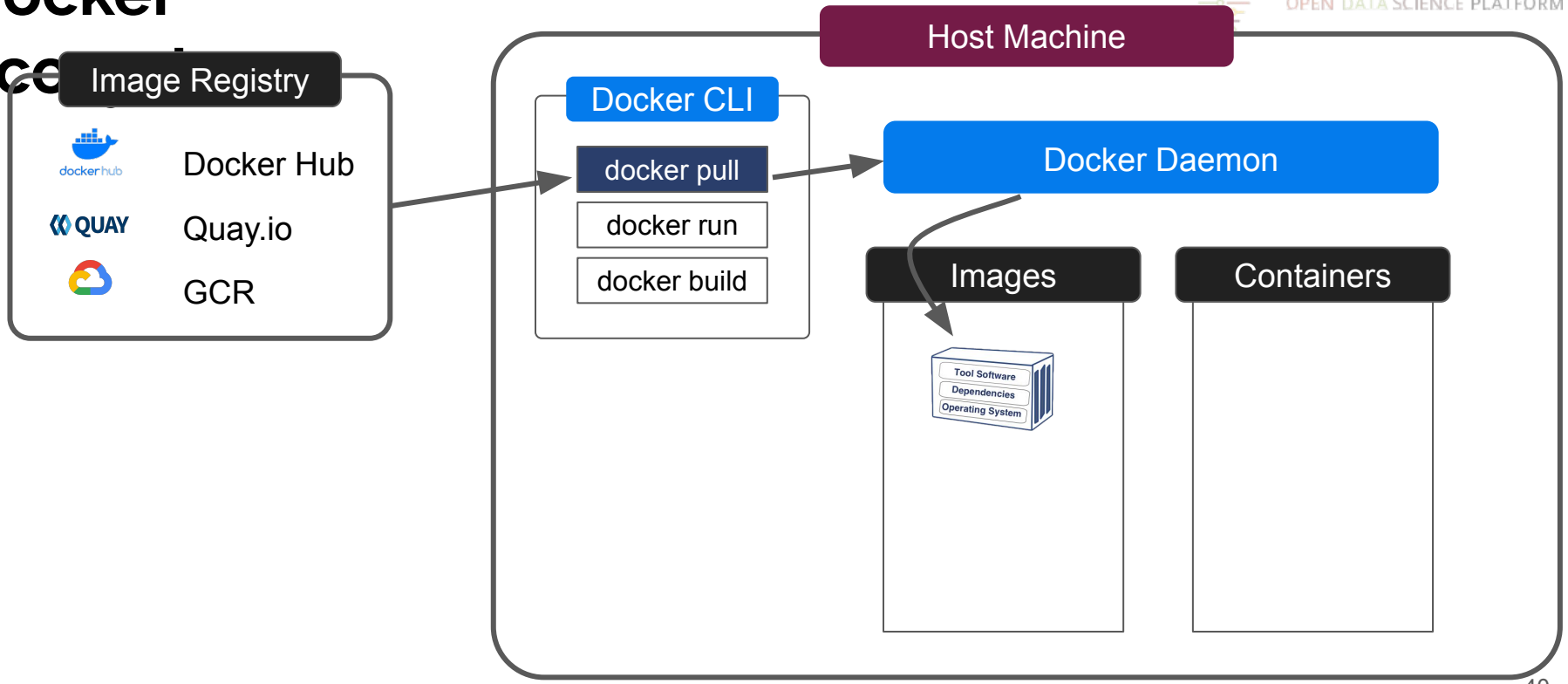
- Follow formatting guidelines for easier reading:
  - ARGS, ENV, LABEL in that order
  - Always add versions of tools in the LABEL
  - Minimal RUN steps
  - Alphabetize package install
  - Clean up package index cache
  - Use ; instead of && for line continuation
  - Logically separate steps within RUN
  - Four spaces per tab indent
  - Short comments to describe each step
  - tini is always default entrypoint

Docker Ecosystem

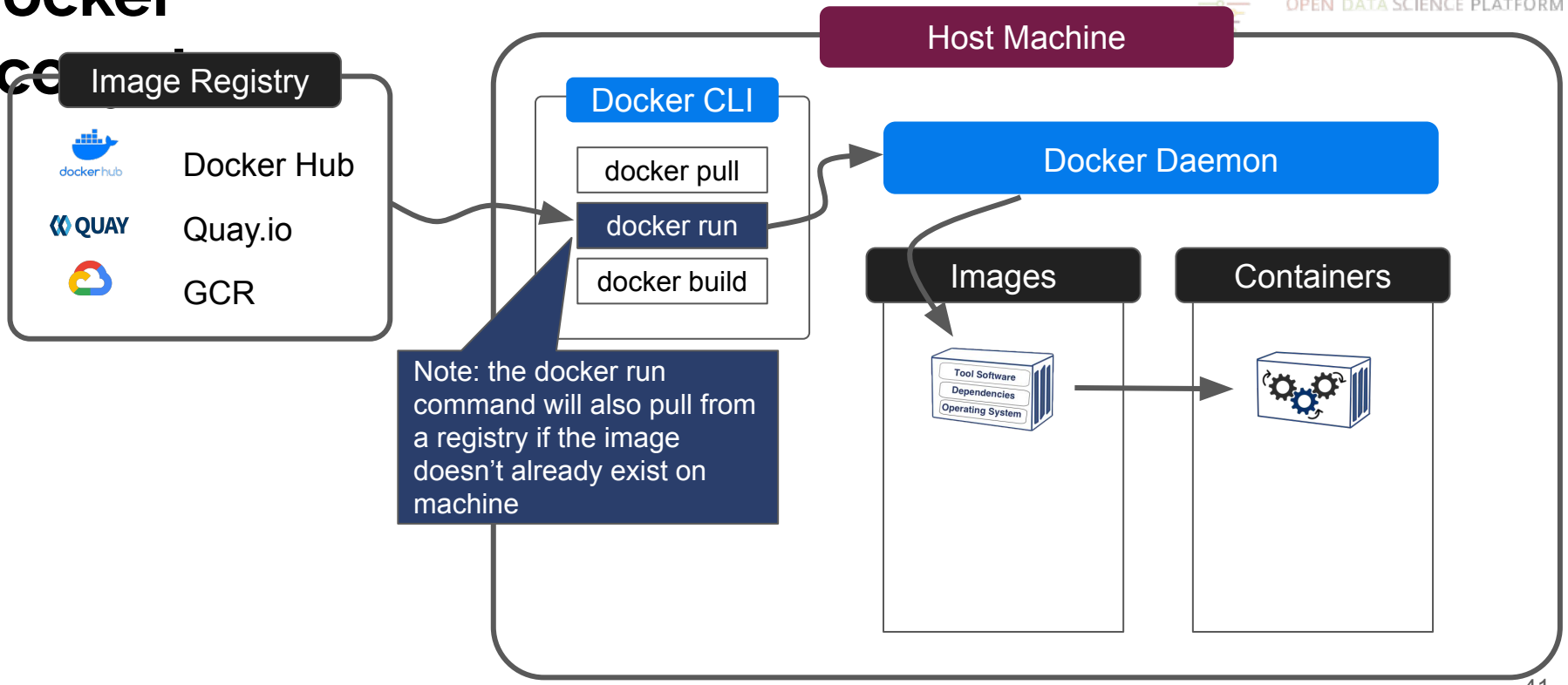




Docker Ecosystem



Docker Ecosystem



Docker Client (CLI)

A command-line utility for:

- downloading and building Docker images
- running Docker containers
- managing both images and containers (think disk space, cleanup)

```
docker [sub-command] [-flag options] [arguments]
```

Basic Docker Sub Commands:

Docker has a whole library of commands, here are some basic examples:

Display system-wide information about your installation of docker:

```
docker info [OPTIONS]
```

Managing docker images:

```
docker image [COMMAND]
```

Managing docker containers:

```
docker container [COMMAND]
```

Running docker containers:

```
docker run [-flags] [registry name]/[path to image repository]:[tag] [arguments]
```

How are containers commonly used?

Run and done

1. Execute docker run command
2. Actions executed in container (stuff happens)
3. Container stops running after completion

How are containers commonly used?

Your main method of containers!!

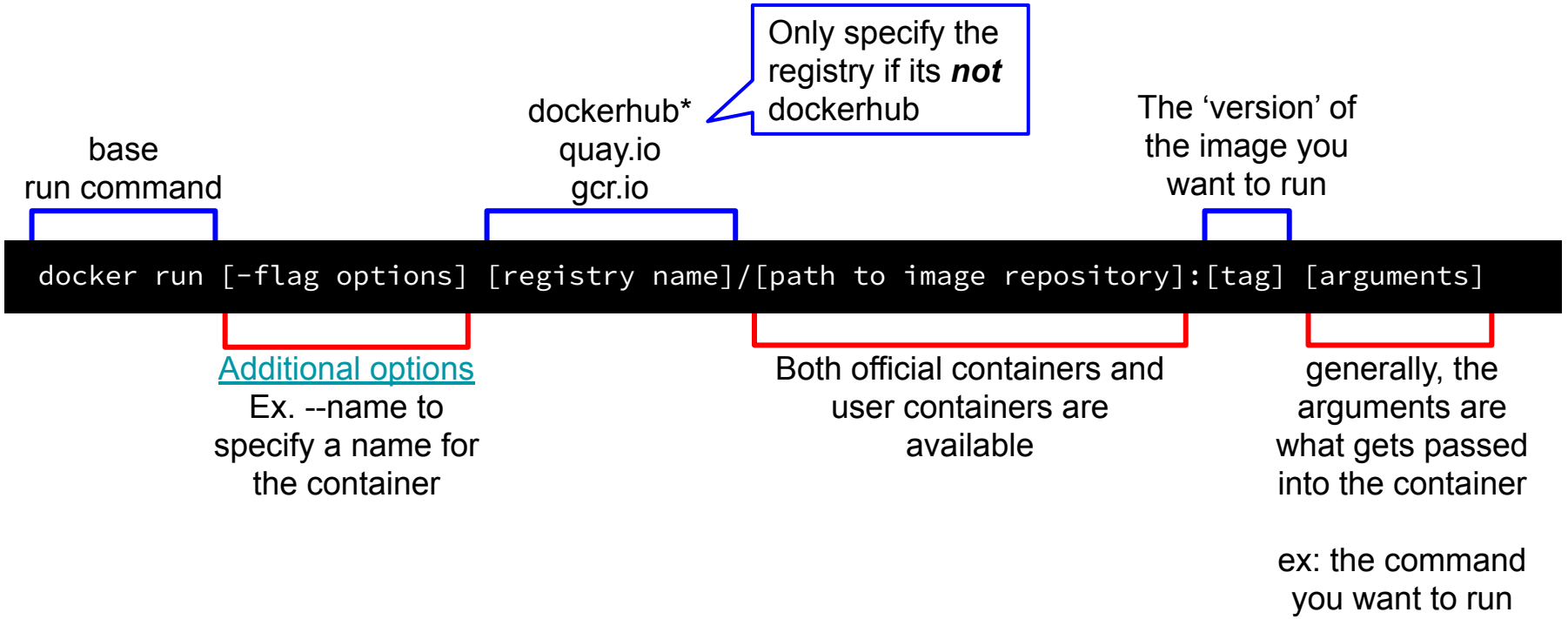
Run and done

1. Execute docker run command
2. Actions executed in container (stuff happens)
3. Container stops running after completion

Run continuously

1. Execute docker run command
2. Container starts and runs in the background *continuously*
3. Other processes can interact with the container (stuff happens)
4. ***Container keeps running unless it is stopped***

How do I ‘run’ a container?



Exercise #1a: Running containers

```
docker run [-flag options] [registry name]/[path to image repository]:[tag] [arguments]
```

Exercises:

Use the whalesay container from Docker Hub to print a welcome message

```
docker run docker/whalesay cowsay "fill me in"
```

Exploring containers interactively

```
docker run [-flag options] [registry name]/[path to image repository]:[tag] [arguments]
```

-i -t

- **The ‘-it’ flags drop you *inside* the container**
- (-i) Keeps STDIN open for interactive use and (-t) allocates a terminal
- You can then interact with the container’s terminal like a normal command line

Example:

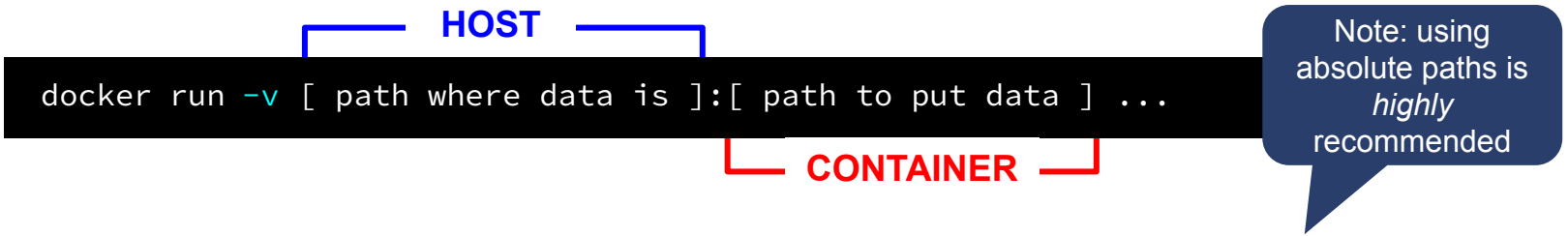
Enter the samtools container and confirm that samtools is installed

```
docker run -it quay.io/ldcabansay/samtools:latest
```

Sharing data between host and container

Bind mounts (-v) (aka two-way data binding)

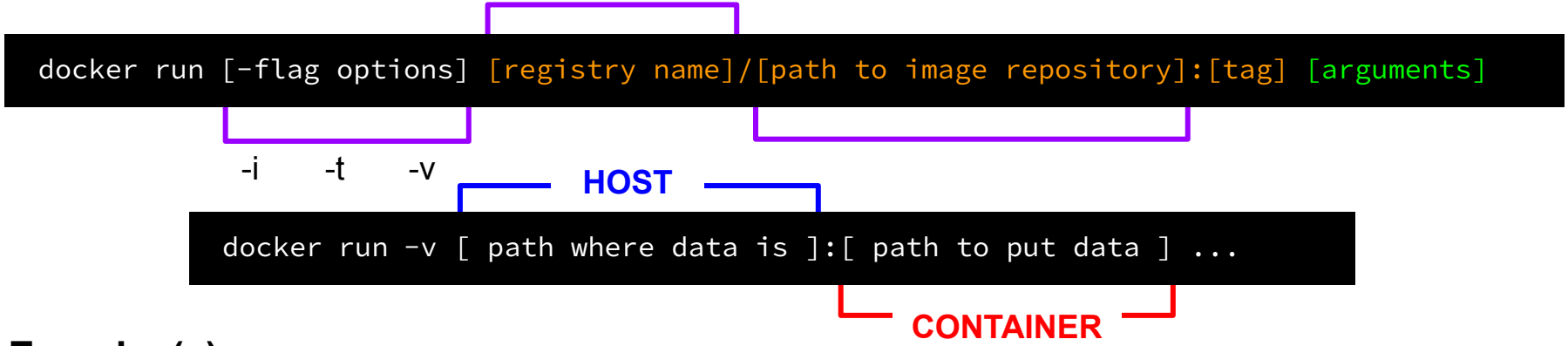
- Map a host directory to a directory in a container
- Any files added to either directory is available in the other



Output stored in the container directory `/tmp/data` will also be available on the host at `/usr/data`

```
docker run -v /usr/data:/tmp/data ...
```

Exercise #1b: exploring containers



Exercise(s):

Enter the samtools container, but this time bring in some data!

```
docker run -it -v /root/bcc2020-training/data:/data quay.io/ldcabansay/samtools:latest
```

Convert a sam file to a bam file using the samtools container.

```
docker run -v /root/bcc2020-training/data:/data quay.io/ldcabansay/samtools:latest
samtools view -S -b /data/mini.sam -o /data/mini.bam
```

Data binding: In-depth (Extra Reading)

- Useful tips and tricks to know about databinding
- Advanced features and caveats to keep in mind
- Read more here: https://docs.docker.com/storage/

Dockerfiles: Custom Images

- Sometimes an existing image isn't available for the software we want to use or an existing image may be lacking something we require
- **Dockerfiles** are used to create our own custom images
  - Starts from a base image
  - Contains a series of steps that set up our environment
- We can then also share these custom images via an image registry

Primer: How is software installed and used?

Package managers

- Managed collection of software with automated install, upgrades, and removal

Executable Files or Binaries

(ex: *.jar, *.c, grep, tar, diff, md5sum)

- software that has already been built or compiled into an executable file.

Building or running from source files

- Compiling from source build an executable (requires compiler & dependencies)
- For languages that don't need compiling (python), necessary runtime dependencies required in environment

Author of dockerfile programmatically details the software installation and any other steps for environment setup

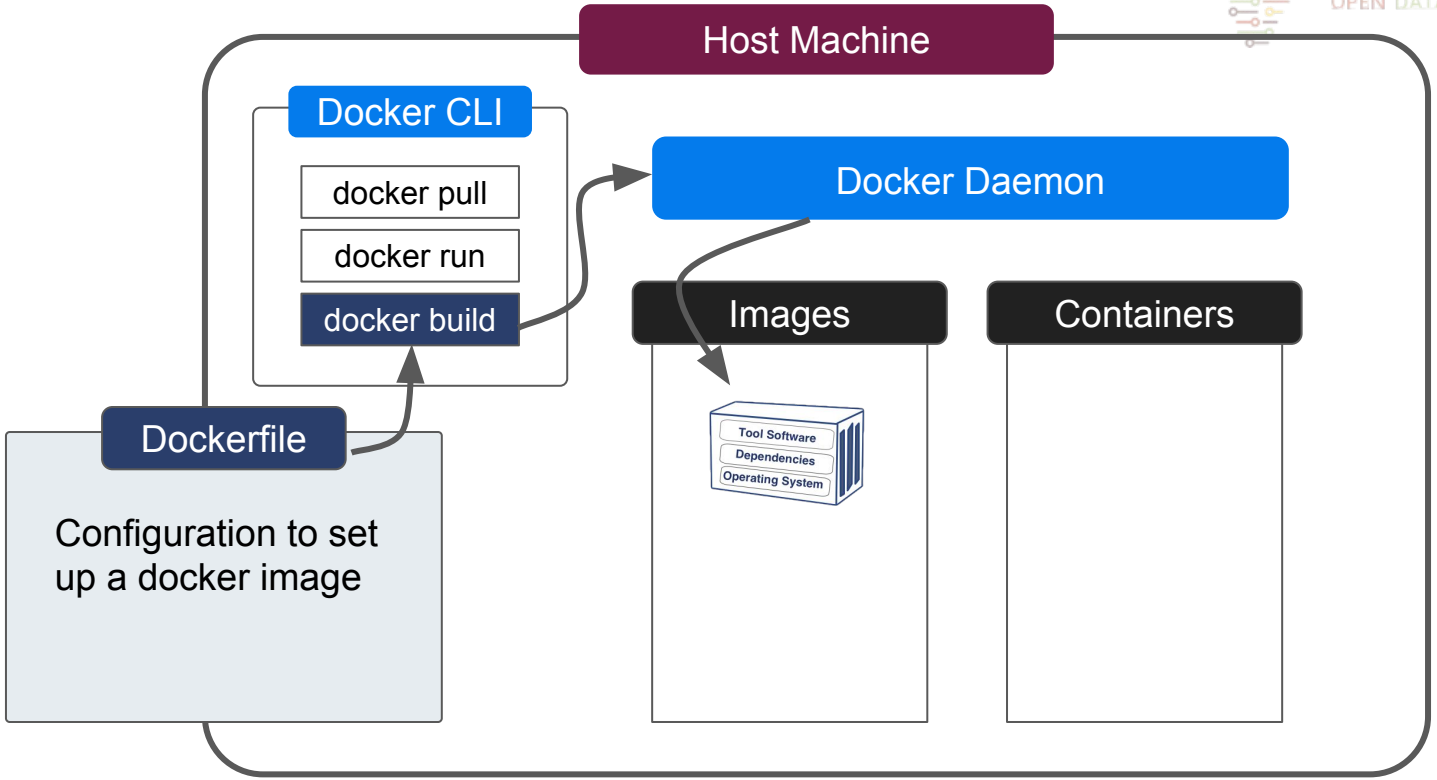
Image built from the dockerfile can then be used for 'off-the-shelf' software usage by others.

Dockerfiles Overview:

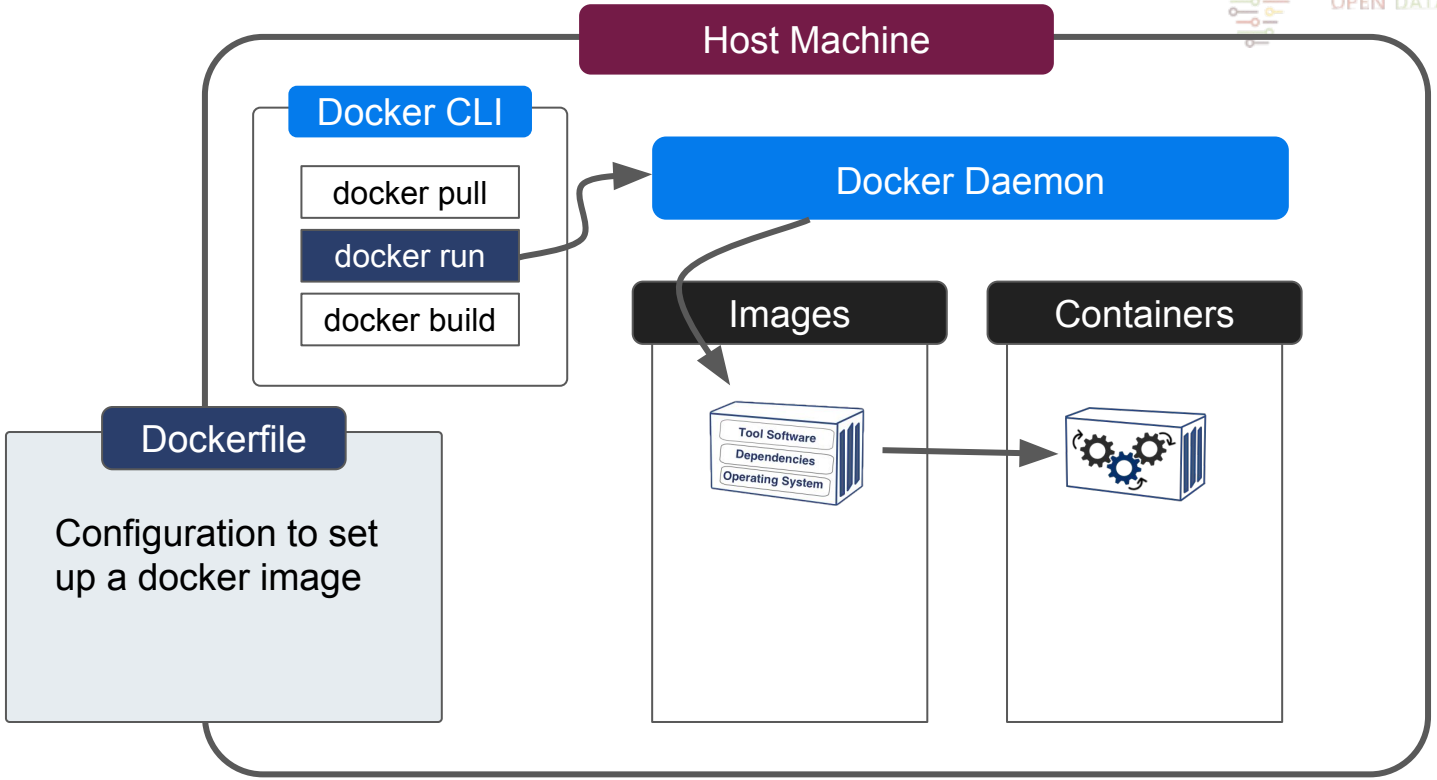
A simple text file with instructions to build an image:

- YAML syntax
- Start from a base image
- Metadata
- Install software and dependencies
- Set up scripts
- Other environment prep
- Define commands to run when container starts

Dockerfiles - local

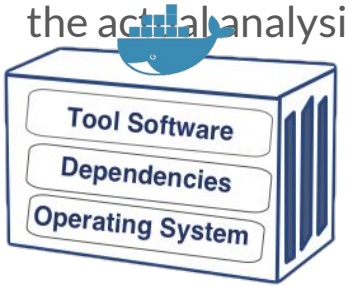


Dockerfiles - local



Example: BWA (via package manager)

- Images of containers are built from Dockerfiles
- Dockerfiles describe the packaged up environment:
  - operating system or base image to build upon
  - dependencies needed for the software
  - the actual analysis



Dockerfile

```
1  ##########################################################
2  # Dockerfile to build a sample container for bwa
3  ##########################################################
4
5  # Start with a base image
6  FROM ubuntu:18.04
7
8  # Add file author/maintainer and contact info (optional)
9  MAINTAINER Louise Cabansay <lcabansa@ucsc.edu>
10
11 #set user you want to install packages as
12 USER root
13
14 #update package manager & install dependencies (if any)
15 RUN apt update
16
17 # install analysis software from package manager
18 RUN apt install -y bwa
19
20
```

Basic Commands: docker build

```
docker build [-flag options] [build context]
```

-t -f

(-t) : Builds and creates a tag **v1.0** for a **bwa** image (if in dockerfile directory)

```
docker build -t bwa:v1.0 .
```

(-f) : Build a specific Dockerfile by providing path to file (relative to build context)

```
docker build -t bwa:v1.0 -f dockerfiles/bwa/Dockerfile .
```

View built Docker images

```
docker image ls
```

Exercise #2a: Writing your first Dockerfile: tabix

Dockerfiles describe the packaged up environment:

- operating system or base image to build upon:
  - ubuntu:18.04
- Install dependencies needed: N/A
- Install actual analysis software:
  - tabix

Dockerfile

```
# Start with a base image
FROM { base image name }

# Add file author/maintainer and contact info (optional)
MAINTAINER {your name} <youremail@research.edu>

# set user you want to install packages as
USER root

# update package manager & install dependencies (if any)
RUN apt update

# install analysis software from package manager
RUN apt install -y { software package name }
```

Build an image from Dockerfile:

```
docker image build -t { name } -f { path to dockerfile } .
```

Exercise #2a: Writing your first Dockerfile (solution)

Dockerfiles describe the packaged up environment:

- operating system or base image to build upon:
  - ubuntu:18.04
- Install dependencies needed: N/A
- Install actual analysis software:
  - tabix

Dockerfile

```
# Start with a base image
FROM ubuntu:18.04

# Add file author/maintainer and contact info (optional)
MAINTAINER {your name} <youremail@research.edu>

# set user you want to install packages as
USER root

# update package manager & install dependencies (if any)
RUN apt update

# install analysis software from package manager
RUN apt install -y tabix
```

Build an image from Dockerfile:

```
docker image build -t tabix -f ~/bcc2020-training/docker-training/exercise2/Dockerfile .
```

Exercise #2b: Try out your new container!

```
docker container run [-flag options] [registry name]/[path to image repository]:[tag] [args]
```

Exercises:

1. Verify that your image was built (get the image ID to use in part 2)

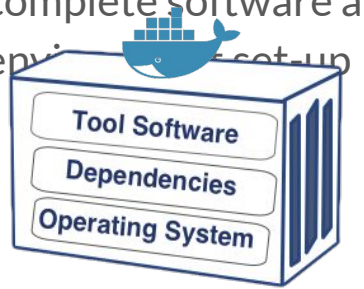
```
docker image ls
```

2. Use your local image to view the tabix command help

```
docker run [image id] tabix
```

Ex: bamstats (executable)

- Dockerfiles describe the packaged up environment:
  - operating system or base image to build upon
  - dependencies needed for the software
  - the actual analysis software
    - Here it's an already compiled executable
  - scripts or commands to complete software and environment set-up

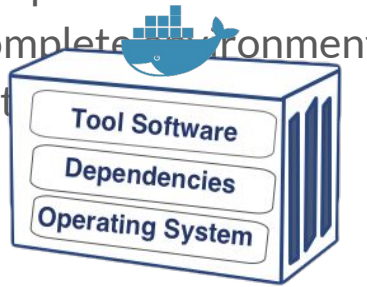


Dockerfile

```
# Start with a base image
FROM ubuntu:14.04

# Add file author/maintainer and contact info (optional)
MAINTAINER Brian OConnor <briandoconnor@gmail.com>

# install software dependencies
USER root
RUN apt-get -m update && apt-get install -y wget unzip \
    openjdk-7-jre zip


# manual software installation from source
# get the tool and install it in /usr/local/bin
RUN wget -q http://downloads.sourceforge.net/project
/bamstats/BAMStats-1.25.zip

# commands/scripts to finish software setup
RUN unzip BAMStats-1.25.zip && \
    rm BAMStats-1.25.zip && \
    mv BAMStats-1.25 /opt/

COPY bin/bamstats /usr/local/bin/
RUN chmod a+x /usr/local/bin/bamstats

# switch back to the ubuntu user so this tool (and the
files written) are not owned by root
RUN groupadd -r -g 1000 ubuntu && useradd -r -g ubuntu -u
1000 -m ubuntu
USER ubuntu

# command /bin/bash is executed when container starts
CMD ["/bin/bash"]
```

Example: samtools (compile from source files)

- Images of containers are built from Dockerfiles
- Dockerfiles describe the packaged up environment:
  - operating system or base image to build upon
  - dependencies needed for the software
  - the actual analysis software
  - Scripts or commands to complete environment set



Dockerfile

```
# Start with a base image
FROM ubuntu:18.04

# Add file author/maintainer and contact info (optional)
MAINTAINER Louise Cabansay <lcabansa@ucsc.edu>

# install software dependencies
RUN apt update && apt -y upgrade && apt install -y \
    wget build-essential libncurses5-dev zlib1g-dev \
    libbz2-dev liblzma-dev libcurl3-dev \

WORKDIR /usr/src

# get the software source files
RUN wget https://github.com/samtools/samtools/releases/
download/1.10/samtools-1.10.tar.bz2


# installation commands to compile source files
tar xjf samtools-1.10.tar.bz2 && \
    rm samtools-1.10.tar.bz2 && \
    cd samtools-1.10 && \
    ./configure --prefix $(pwd) && \
    make

# add newly built executables to path
ENV PATH="/usr/src/samtools-1.10:${PATH}"

```

Container Vulnerability Scanners

- Docker Scan or Docker Scout >> https://docs.docker.com/scout/
  - Free for personal and education
  - Unlimited local image scans and up to 3 repositories
  - Part of Docker Desktop
- Dagda >> https://github.com/eliasgranderubio/dagda
  - Downloads the known CVE's (Common Vulnerabilities and Exposures) to a local database
  - Scans for container, application and system vulnerabilities
  - Uses ClamAV for scanning trojans, viruses and malware
- Trivy >> https://github.com/aquasecurity/trivy
  - Scans for vulnerabilities based on whitelists from the Clair server
- Grype >> https://github.com/anchore/grype
  - Scans multiple image formats including docker and singularity
- Docker Bench for Security >> https://github.com/docker/docker-bench-security

Docker Basics : Installation

Installing Docker on Ubuntu

Docker Desktop

- Remove all previous versions of Docker
- Download .deb package >> https://docs.docker.com/desktop/install/ubuntu/

sudo apt update
sudo apt install ./docker-desktop-<version>-<arch>.deb

- Accept terms of use
- Launch Docker Desktop

sudo systemctl --user start docker-desktop

- Installs:
  - Docker Desktop GUI
  - Docker engine
  - Docker-compose
  - Docker CLI
  - Docker Scout

Docker Basics : Installation

Docker engine and Docker-compose

- Remove all previous versions of Docker
- Update repository and install dependencies
-

```
sudo apt update
sudo apt install software-properties-common curl apt-transport-https ca-certificates –y
```

- Add Docker's GPG signing key to the system

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

- Add the official Docker repository to your system

```
secho "deb [signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update
```

- Install Docker from the official repository

```
sudo apt install docker-ce docker-ce-cli containerd.io uidmap –y
```

- Verify Docker is running

```
sudo systemctl status docker
```

- Test Docker

```
docker run hello-world
```

Installs:

- Docker engine
- Docker-compose
- Docker CLI

Future work

Further Learning

- LinkedIn Learning for UCT staff
- Building custom containers
- Docker volumes
- Docker networking
- Managing containers
  - Desktop applications
  - CLI
  - Disk space if not kept in check
- Port mapping
- Pushing/Pulling from other repositories: GitHub or GitLab
- Docker-compose
- Docker swarm