

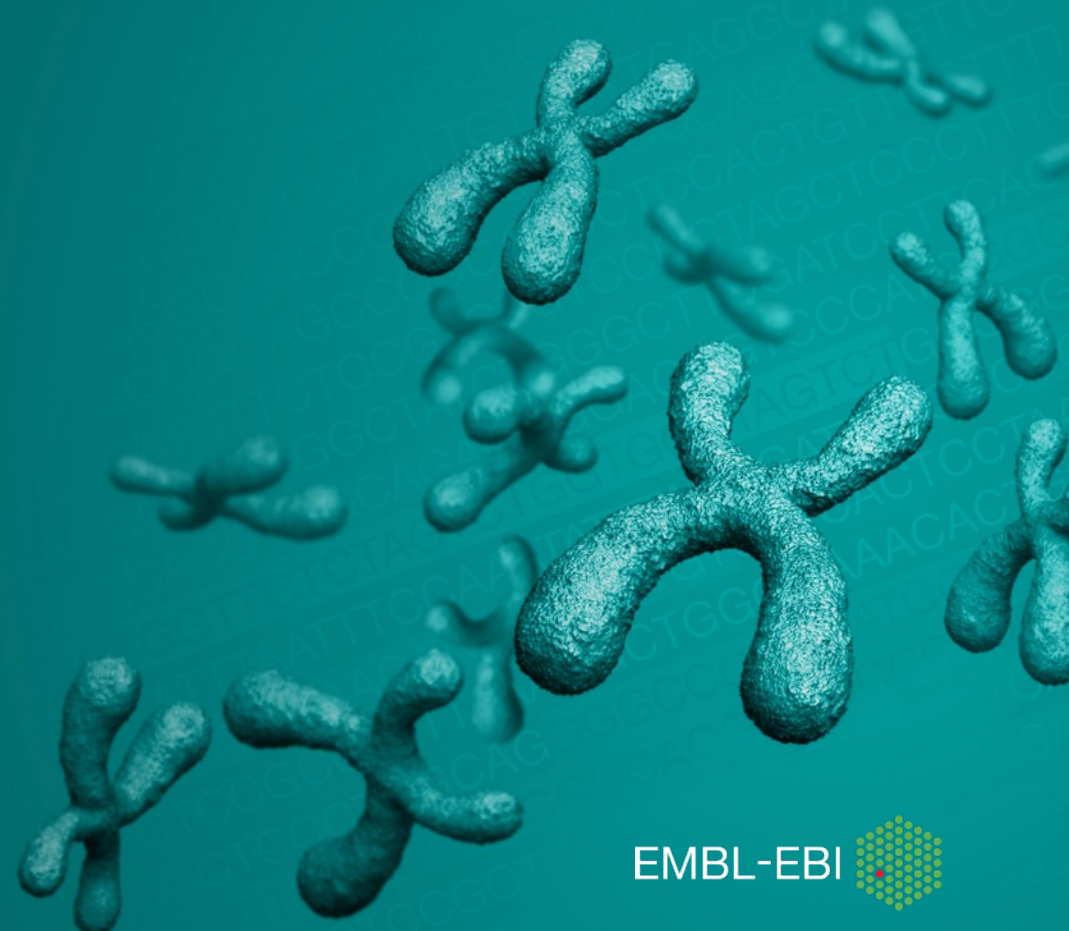
Creating Containers with Docker

Tony Wildish

Cloud Bioinformatics Lead Architect

EMBL-EBI

wildish@ebi.ac.uk



Docker: What is it?

- Docker is a ‘container technology’
 - Linux-specific
 - Can’t run Mac OSX or Windows *in* docker containers, but
 - Can run docker containers *on* Mac OSX or Windows
 - Shrink-wrap your software, run it on ~any Linux platform
- *Not* a virtual machine
 - Similar, but more lightweight
 - Smaller, faster to start, easier to maintain and manage
 - Lighter on system resources, much more scalable!

Why use Docker?

- Portability:
 - No need to rebuild your application for a new platform!
 - Build a container once, run it anywhere
 - AWS/GCP/...
 - Stable s/w versions across all platforms, no runtime glitches
- Reproducibility:
 - Because your s/w is stable, your pipeline is reproducible
 - Run the exact same binaries again 10 years from now 😊 ☹️

What can you do with it?

- Computational workloads
 - Use applications without having to install them
 - Run your applications anywhere; clouds, HPC centres, laptops
 - **Reproducible pipelines**
- Services
 - Web portals/gateways (R/Shiny, Apache, Jupyter...)
 - Persistent workflow manager interfaces (Airflow etc...)
 - Continuous build systems (Gitlab...)
 - For prototyping or for production running (databases etc)

Docker components

- The 'docker' command-line tool
 - A bit of a kitchen-sink, your one-stop shop for everything docker
- The docker-daemon
 - Works behind the scenes to carry out actions
 - Manages container images, processes
 - Builds containers when requested
 - Runs as root, not a user-space daemon
- Docker.com
 - All things docker: installation, documentation, tutorials
- Dockerhub.com
 - Repository of docker containers. Many other repositories exist

Docker concepts

- **Image**
 - A shrink-wrapped chunk of s/w + its execution environment
- **Image tags**
 - Identify different versions of an image
 - A namespace for separating your images from other peoples
- **Image registry**
 - A place for sharing images with a wider community
 - Dockerhub.com, plus some domain-specific registries
- **Container**
 - A process instantiated from an image
- **Dockerfile**
 - A recipe for building an image: download, compile, configure...
 - Can share either the Dockerfile, or the image, or both

Docker images: layers and caching

- Images use the '**overlay filesystem**' concept
 - Image is built by adding layers to a base
 - Each command in the Dockerfile adds a new layer
 - Each layer is cached independently
 - Layers can be shared between multiple images
 - Change in one layer invalidates all following layers
 - Forces rebuild (similar to 'make' dependencies...)
- Performance considerations
 - Too many layers can impede performance
 - Too few can cause excessive rebuilding
 - Building production-quality images takes care, practice

Building a container: the Dockerfile

- A recipe for building a container
- Start with a base image, add software layer by layer
 - Choosing the base image has a big effect on how large your container will be: go small!
- Add metadata describing the container
 - Always a good idea
- Set the command to run when starting the container, map network ports, set environment variables
 - Not strictly needed for batch applications, useful for services (web apps, databases...)


```
FROM debian:jessie

# LABEL lets you specify metadata, visible with 'docker inspect'
LABEL Maintainer="Tony Wildish, wildish@ebi.ac.uk" Version=1.0

# I can set environment variables
ENV PATH /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

# Commands to prepare the container
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update -y
RUN apt-get upgrade -y
RUN apt-get install --assume-yes apt-utils
RUN apt-get install -y python
RUN apt-get install -y python-pip
RUN apt-get clean all
RUN pip install bottle

# Add local files
ADD hello.py /tmp/

# open a port
EXPOSE 5000

# specify the default command to run
CMD ["python", "/tmp/hello.py"]
```

Name+version

```
FROM debian:jessie
```

```
# LABEL lets you specify metadata, visible with 'docker inspect'
```

Contact info

```
LABEL Maintainer="Tony Wildish, wildish@ebi.ac.uk" Version=1.0
```

```
# I can set environment variables
```

```
ENV PATH /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

```
# Commands to prepare the container
```

```
ENV DEBIAN_FRONTEND=noninteractive
```

```
RUN apt-get update -y
```

```
RUN apt-get upgrade -y
```

```
RUN apt-get install --assume-yes apt-utils
```

```
RUN apt-get install -y python
```

```
RUN apt-get install -y python-pip
```

```
RUN apt-get clean all
```

```
RUN pip install bottle
```

```
# Add local files
```

```
ADD hello.py /tmp/
```

```
# open a port
```

```
EXPOSE 5000
```

```
# specify the default command to run
```

```
CMD ["python", "/tmp/hello.py"]
```

Heavy lifting,
install base
tools before
our code

Name+version

```
FROM debian:jessie
```

'heavy' base image: 123 MB

```
# LABEL lets you specify metadata, visible with 'docker inspect'
```

Contact info

```
LABEL Maintainer="Tony Wildish, wildish@ebi.ac.uk" Version=1.0
```

```
# I can set environment variables
```

```
ENV PATH /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

```
# Commands to prepare the container
```

```
ENV DEBIAN_FRONTEND=noninteractive
```

Heavy lifting,
install base
tools before
our code

```
RUN apt-get update -y
```

Blind update – to what???
Container != VM

```
RUN apt-get upgrade -y
```

```
RUN apt-get install --assume-yes apt-utils
```

```
RUN apt-get install -y python
```

```
RUN apt-get install -y python-pip
```

```
RUN apt-get clean all
```

```
RUN pip install flask
```

Lots of RUN commands
means lots of layers,
not ideal for the cache

```
# Add local files
```

```
ADD hello.py /tmp/
```

```
# open a port
```

```
EXPOSE 5000
```

Final image
size: 360 MB

```
# specify the default command to run
```

```
CMD ["python", "/tmp/hello.py"]
```

FROM alpine:3.5

Base image only 5 MB

LABEL Maintainer="Tony Wildish, wildish@ebi.ac.uk" Version=1.0

ENV PATH /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

RUN apk add --no-cache --update-cache --update python && \
apk add --no-cache --update py2-pip && \
pip install flask

Command chaining with &&, reduces #layers

Install only what we want

ADD hello.py /tmp/

EXPOSE 5000

CMD ["python", "/tmp/hello.py"]

Final image size: 53.2MB

Other Docker directives

- WORKDIR
 - Set the working directory inside the container
- CMD & ENTRYPOINT
 - Very similar. If you get stuck with CMD, look at ENTRYPOINT
- ARG
 - Pass information through the build chain (see exercises)
- USER
 - Specify the user to run as inside the container (see exercises)

Building containers

- Build your container with 'docker build'
 - **docker build -t *user/package:version* -f Dockerfile \$dir**
 - Tag (-t) not obligatory, but *very* good idea
- Build 'context'
 - Everything in \$dir is sent to the build as the 'context'
 - Use '**.dockerignore**' file to exclude files/directories
 - Can greatly speed build times – don't send your entire home directory!
- Upload your container to Dockerhub (hub.docker.com)
 - **docker push *user/package:version***

Running containers

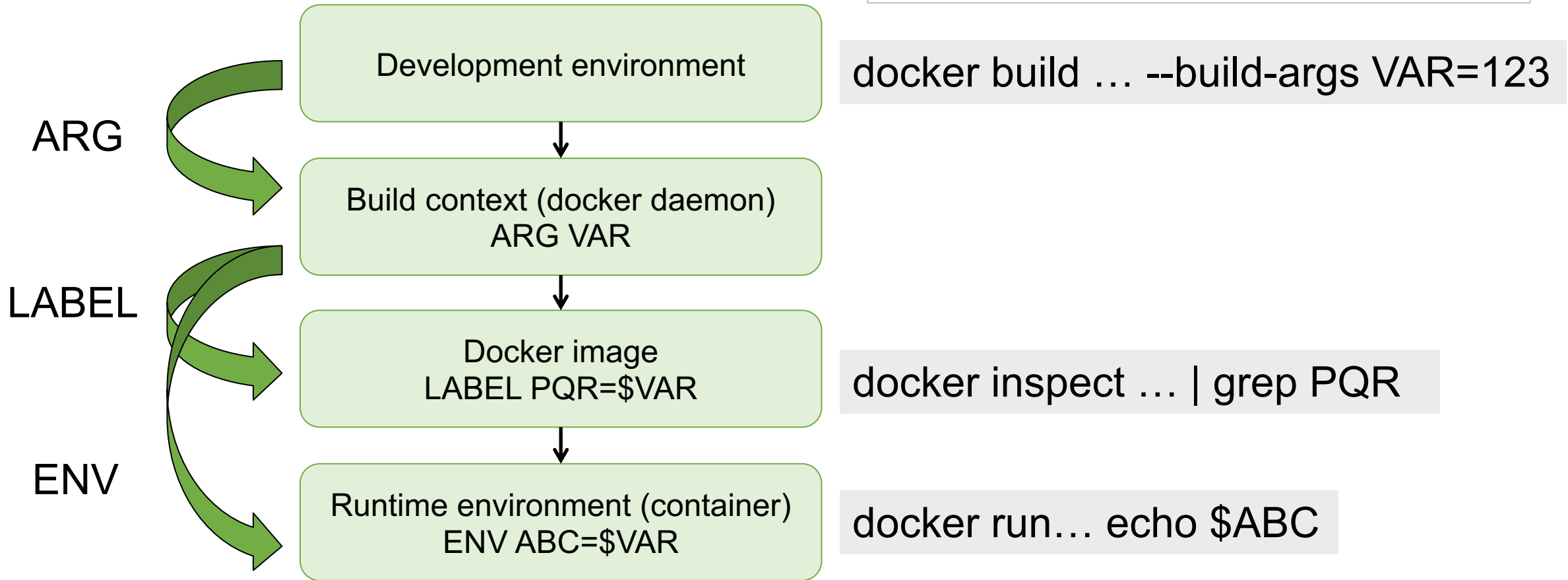
- Run a container with a default command
 - **docker run -i -t ubuntu**
 - Gives you a shell prompt, 'exit' or CTRL-D to quit
 - -i -t -> use for interactive containers
- Run a container, specify the command explicitly
 - **docker run alpine:3.5 /bin/ls --l**
- Set an environment variable
 - **docker run -e PATH=/bin:/usr/bin alpine:3.5 ls**

Docker environments

- **Development** environment
 - The environment in which you issue the 'docker build' command
- **Build** environment
 - The Docker daemon, which executes the build for you
- Docker **image**
 - The shrink-wrapped software, with its baked-in environment
- Docker **container**
 - The running container, with a runtime environment derived from the image

Using metadata in containers

How metadata goes from the command-line to the build environment, to the image, and to the running container



See <https://docs.docker.com/engine/reference/builder/#arg> for more

Getting data in/out of containers

- Map external directories into a container
 - **docker run --volume /external/path:/internal/path**
- E.g: list your current directory, the docker way!
 - **docker run --volume `pwd`:/mnt alpine:3.5 /bin/ls -l /mnt**
- Can map multiple volumes
 - Don't nest them!

Finding pre-built containers

- Q: What's the best way to build a container?
 - A: Don't! Find one that's been built already!

> docker search spades

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
nucleotides/spades		3		[OK]
achubaty/r-spades-devel	Provides a testing environment for buildin...	0		[OK]
biodckrdev/spades	Tools (written in C using htplib) for mani...	0		[OK]
ycogne/spades	spades tools	0		[OK]
bioboxes/spades	St. Petersburg genome assembler	0		[OK]
unlhcc/spades		0		
[...]				

- Q: How do you know which one to pick?
 - A: trial and error ☹️
 - Look for official builds, #stars.
 - depends on the details of how the container was built

Finding pre-built containers

- Alternative sources
 - Dockerhub.com
 - Same as 'docker search', but can get information about the build, instructions for use etc
 - Google: “dockerfile NCBI blast”
- Ask the authors of your favorite package if they have a container already
 - But check it before using, they may not be experts!
 - `docker images | grep <image> # check size`
 - `docker history --no-trunc <image> # see how it was built`
 - Check their github repository!

Security

- Running as root
 - Containers run as root by default, which is a major security risk. E.g, I cannot normally list the **/etc/sudoers** file:
 - **> cat /etc/sudoers**
cat: /etc/sudoers: Permission denied
 - But using a docker container, I can!:
 - **> docker run --volume /etc:/mnt alpine:3.5 cat /mnt/sudoers | head -3**
sudoers file.

This file MUST be edited with the 'visudo' command as root.
- Solution: run as non-root user

Run as non-root user

```
FROM alpine:3.5
```

```
RUN apk update && apk add shadow && \  
    groupadd muggles && \  
    useradd -ms /bin/sh -G muggles dudley
```

```
USER dudley:muggles
```

- **> docker build -f Dockerfile.user -t user .**
- **> docker run user id**
uid=1000(dudley) gid=1000(muggles)
- **> docker run --volume /etc:/mnt user cat /mnt/sudoers | head -2**
cat: can't open '/mnt/sudoers': Permission denied

Running as non-root user

- You can't prevent the user from running as root if they launch the container themselves
 - > **docker run -u root my-container ...**
- Best practices for services:
 - Make sure you don't give your container-user sudo permissions by mistake
 - Create an unprivileged user in Dockerfile, in an unprivileged group
 - If you're mounting a filesystem, take user/group from the files you want there
 - Change to that user before running the application
 - **Test that it works as expected** – don't assume it will, verify it

Singularity?

- Singularity (<https://sylabs.io/docs/>) is a Docker-compatible tool for building and running containers
 - Think of it as Docker minus the dangerous bits
 - No running as root, runs only as user who creates it
 - No exposing ports, not useful for running services
 - Therefore very friendly to HPC centres, as well as the cloud
 - Your cluster admin won't let you run Docker, but will let you run Singularity
- If you're running a workflow on an HPC farm, take a look at Singularity

What goes into an image?

- What goes into a image, what doesn't?
 - No hard and fast rules, here are some guidelines
- Include...
 - Anything 'compiled', i.e. anything with system dependencies
 - Anything that needs 'installing' to run, that has portability issues
 - i.e. if you can't install it on another machine without effort, put it in a container
- Exclude...
 - Simple bash/Perl/Python scripts => install from git etc
 - Need Python/Perl modules? Include them in the container
 - Anything static: big reference DBs etc
 - Anything you could install by just copying to the filesystem

Dockerizing a pipeline:

- Q: how many containers for a pipeline with 25 steps?
- A: That depends on what your pipeline does
 - ☹ Not good: putting the whole pipeline in a single container
 - Maintenance overhead, can't optimize workflow
 - Remember, a container is not a VM!
 - 😊 Better: one container per (related set of) executable(s)
 - Your pipeline then invokes one container after another
 - You can re-use containers built by other people
 - E.g. One container for blast, including blastp, blastn, blastx, tblastn, tblastx is reasonable
 - But do you use all of them? Or only one? Pick what you need!
 - Do you need the other binaries or files that come with it?
 - Blast+2.6.0: 26 MB/binary for those, but 980 MB total installation.

Best practices

- Security
 - Don't run services as root, create & use an unprivileged user for that purpose
- Document your containers
 - Use **LABEL** to add metadata
 - Tag your images: don't use 'latest' by default
- Keep your containers small
 - Start from small image, add only what you need - avoid VM-think!
 - Use one container for one function/functionality
- Optimize your builds
 - Put stable build-commands at the top of your Dockerfile
 - Combine layers where possible ('&&' chaining)
 - Check for bloat: (size of your code)/(size of image)
- Share your containers
 - Put image in dockerhub, Dockerfiles in git, tell us, tell your colleagues...

Summary

- Docker containers allow great portability
 - Because there's nothing to port anymore!
- Building *good* docker containers requires care
 - Not difficult, well worth taking the effort
 - “will I get the same result in one year from now?”
- Security: pay attention if you're running services in your containers
- We can help!

Exercises

- Go to <http://bit.ly/resops-2020>
- Click on '**Docker Practical**'
- Follow the exercises, in any order you like