

Software containers

Or, the definitive solution for escaping the dependency hell and ensuring scientific reproducibility.

Stefano Alberto Russo - INAF / University of Trieste

Introduction

Software containers are lightweight, standalone, executable packages of software that include everything needed to run an application: code, runtime, system tools, system libraries and settings.

They bring basically the same benefits of virtual machines but require a fraction of their resources. For this reason, software containers are the perfect solution for providing reproducible test fast and flexible working environments.

Containers are a standard in the IT industry since many years, and are gaining traction in the scientific community as well.

This seminar will introduce the basics of software containerization using the Docker engine, how to avoid the common pitfalls when using them, and how to containerize scientific codes in order to make them both reproducible and easily shareable.

Why should you listen to me?

An hybrid profile:

- BSc in Computer Science
- MSc in Computational Physics

Started at CERN, as research fellow working on data analysis & Big Data

Then, 5 years in startups.

- Core team member of an IoT energy metering and analytics startup,
- Joined Entrepreneur First, Europe's best deep tech startup accelerator
- ..and launched my own one :)

Now back into research:

- INAF and UniTS, working on resource-intensive data analysis
- adjunct prof. of computer science at University of Trieste
- plus, experienced consultant for a number of private companies

Pointers



stefano.russo@gmail.com



<https://sarusso.github.io>



https://twitter.com/_sarusso

The deal

- 1) I will use Docker as reference, but the concepts are 100% engine-agnostic.
- 2) Always interrupt if you have question, doubts, something not clear, curiosities.
Let's try to keep it interactive!
- 3) Over the talk, think about a concrete use case close to your work.
We can discuss a few at the end.



Outline

- The dependency hell problem
 - Meet Mike
 - Solutions spectrum
- Containers for the win
 - Meet ████████
 - Main concepts
 - Docker
- Containers hands-on
 - Using containers
 - Building your first container
 - Sharing containers

Outline

- The dependency hell problem
 - Meet Mike
 - Solutions spectrum
- Containers for the win
 - Meet ██████
 - Main concepts
 - Docker
- Containers hands-on
 - Using containers
 - Building your first container
 - Sharing containers

The dependency hell problem

→ *Meet Mike*

Mike wants to install a new software.

Mike cannot find a precompiled version that works with his OS and/or libraries.

Mike ask/Google for help and get some basic instructions - like “compile it”.

Mike starts downloading all the development environment, and soon realizes that he needs to upgrade (or downgrade!) some parts of his main Operating Systems.

During this process, something goes wrong.

Mikes spends an afternoon fixing his own OS, and all the next day in trying to compile the software. Which at the end turns out not to do what he wanted.

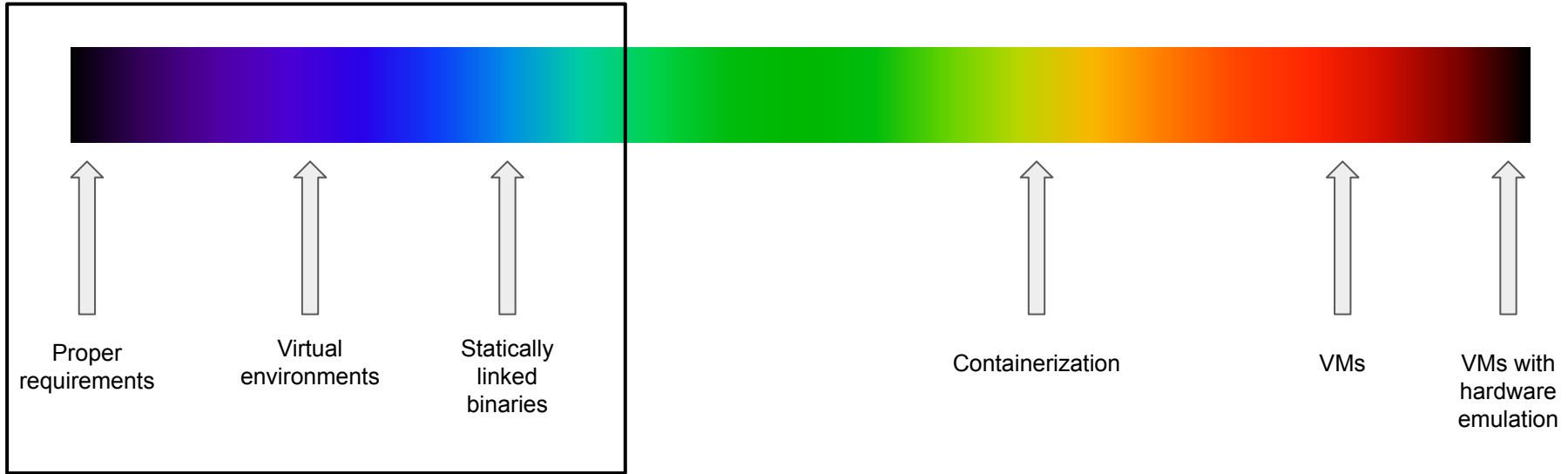
The dependency hell problem

→ *Solutions spectrum*



The dependency hell problem

→ *Solutions spectrum*



The dependency hell problem

→ *Proper requirements*

- Carefully keep track of what libraries/OS features are used in development and report them on the documentation, for each release.
- Prone to human error → we stop here.

Next!

The dependency hell problem

→ *Virtual environments*

- Work in a reproducible environment where libraries are the same for developers and for users. Each release has a virtual environment definition.
- Requires the user to set up and activate its own environment, and works only with some libraries (i.e. Python),
- Not a comprehensive solution and prone to human error → we stop here.

Next!

The dependency hell problem

→ *Statically linked binaries*

- Works only for compiled or compilable languages → we stop here.

Next!

The dependency hell problem

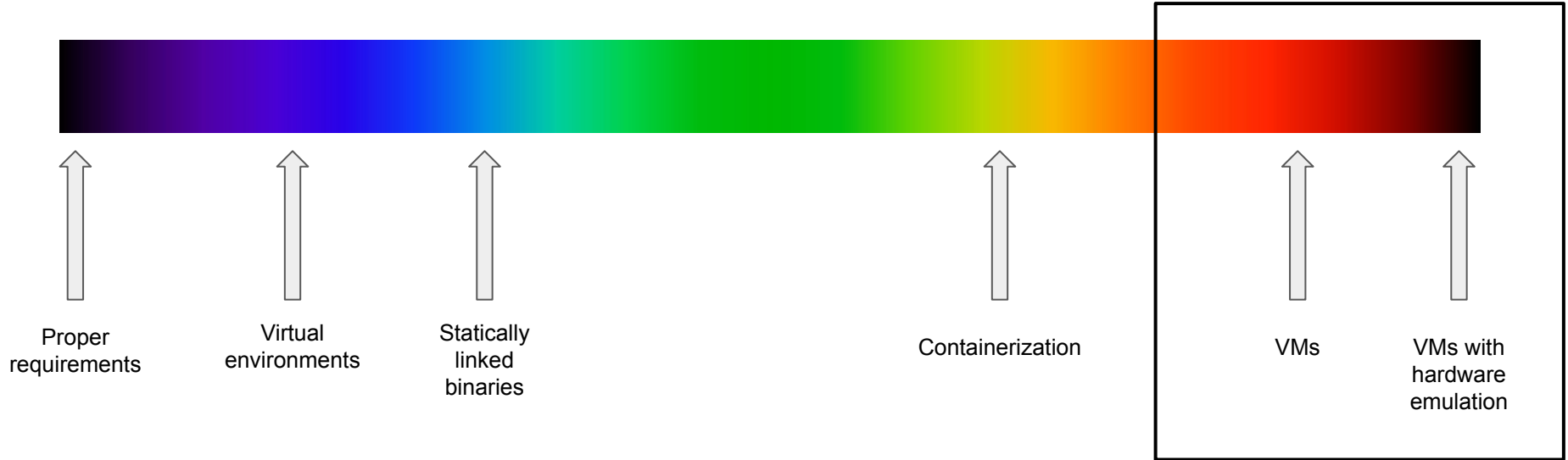
→ *Statically linked binaries*

- Works only for compiled or compilable languages → we stop here.

Next!

The dependency hell problem

→ *Solutions spectrum*



The dependency hell problem

→ *Virtual machines with hardware emulation*

- Well... a bit of over-engineering.  we stop here.

Next!

The dependency hell problem

→ *Virtual machines*

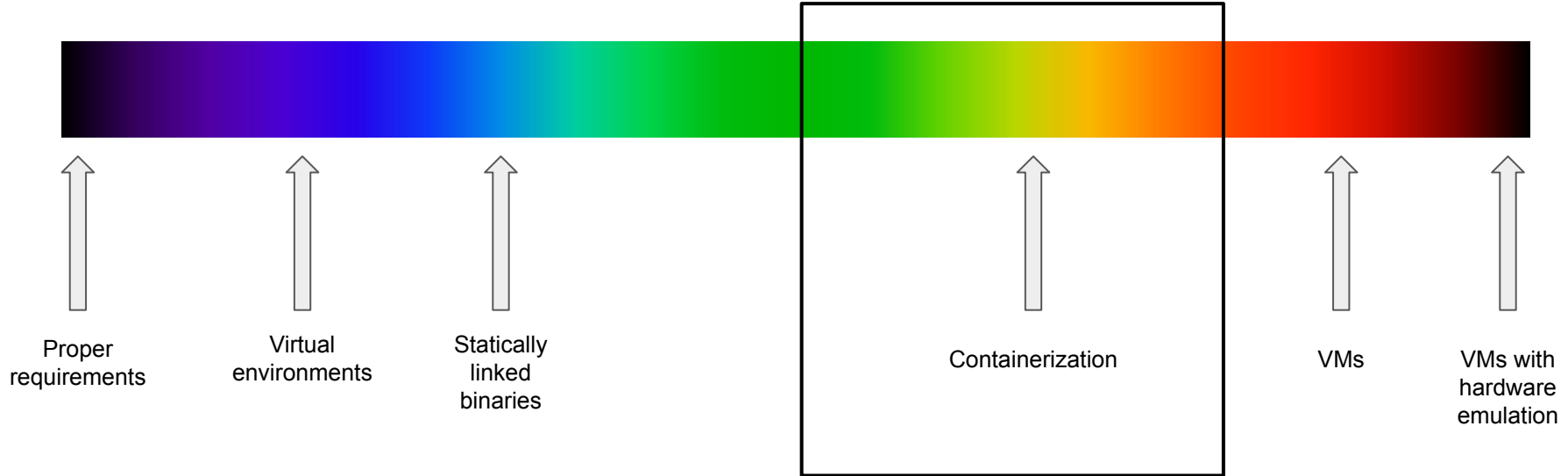
- Works out-of-the box and does not touch the main OS;
- Allows to quickly test a given software / library;
- Need to download a (big) pre-built, *trusted* image (no “source” code);
- Requires pre-allocating dedicated memory at startup, and an entire boot;
- Not suitable for much more than just giving the software a try;
- You will not find much software packaged in this way.

cons > pros → ***we stop here.***

... but we are on the right path. We want this kind of insulation!

The dependency hell problem

→ *Containerization*



The dependency hell problem

→ *Containerization*

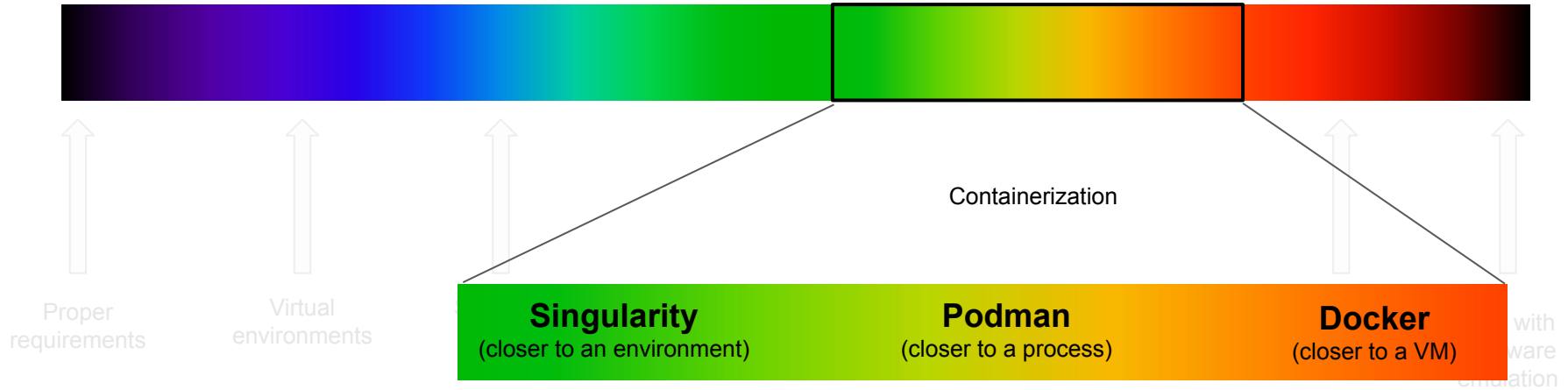
Containers are lightweight, standalone, executable packages of software that include everything needed to run an application:

- code
- runtime
- system tools
- system libraries
- settings etc.

Containers allow to reliably move and distribute software from one computing environment to another, without the burden of VMs.

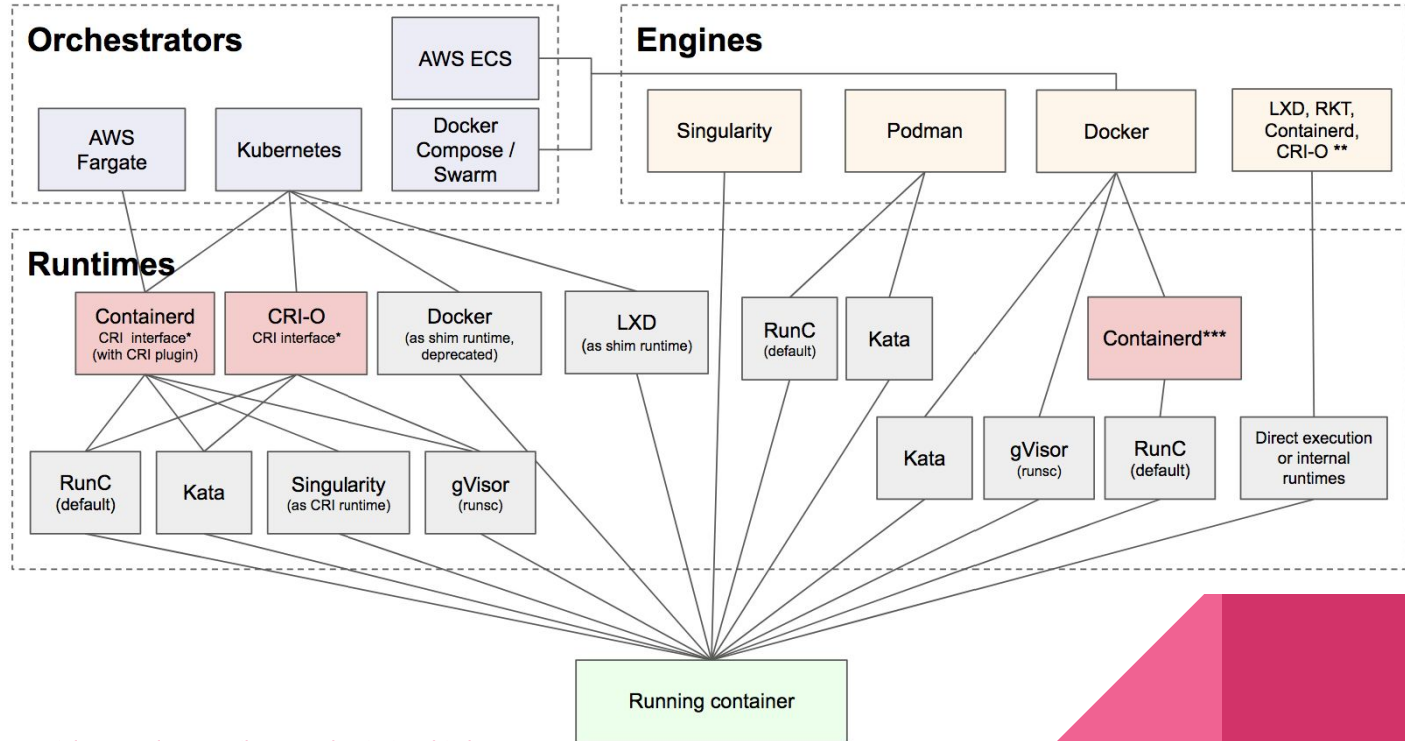
The dependency hell problem

→ *Containerization*



The dependency hell problem

→ Containerization



https://sarusso.github.io/blog_container_engines_runtimes_orchestrators.html

The dependency hell problem

→ *Containerization*

The image shows two overlapping browser windows. The background window displays the Rosetta website's main page, which includes the logo, a navigation menu, and introductory text about the platform. The foreground window shows the 'Software Containers' page, which lists various containerized environments. Each environment card includes a title, a brief description, the Docker image name, a tag selector, and a play button icon.

Environment	Description	Image	Tag
Basic Desktop	A basic desktop environment. Provides a terminal, a file manager, a web browser and other generic applications.	exact/swc/basicdesktop	v0.3.0
CASADesktop v5.6.1-8	CASA, the Common Astronomy Software Applications package, is the primary data processing software for the Atacama Lar...	exact/swc/casadesktop	v5.6.1-8
Jupyter Data Science Lab	The official Jupyter Lab. The Data Science variant, which includes libraries for data analysis from the Julia, Python, and R...	jupyter/scipy-notebook	lab-3.1.17
Jupyter Lab	The official Jupyter Lab. The Scipy variant, which includes popular packages from the scientific Python ecosystem.	jupyter/scipy-notebook	lab-3.2.2
Jupyter Notebook	A Jupyter Notebook server with basic Python numerical data analysis libraries as Pandas and Numpy.	exact/swc/jupyternotebook	v0.3.0
Lofar_visualization	The LOFAR (Low Frequency Array) data visualization toolkit.	exact/swc/lofar_visualization	latest

Outline

- The dependency hell problem
 - Meet Mike
 - Solutions spectrum
- Containers for the win
 - Meet ████████
 - Main concepts
 - Docker
- Containers hands-on
 - Using containers
 - Building your first container
 - Sharing containers

Containers for the win

→ *Meet Bill*

Bill wants to install a new software.

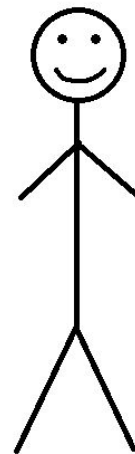
Bill cannot find a precompiled version that works with his OS and/or libraries.

Bill ask/Google for help and finds out that there is a container for it.

Bill pulls the container and runs it.

Bill immediately discovers that theat software is not suitable for his research, and finds a more appropriate one (as a container, of course!)

Bill spends the afternoon writing conclusions on his very important research using his new software while enjoying a hot cup of latte.



Containers for the win

→ *Meet Bill*

Bill wants to install a new software.

Bill cannot find a precompiled version that works with his OS and/or libraries.

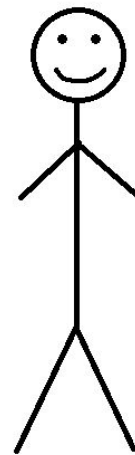
Bill ask/Google for help and finds out that there is a container for it.

Bill pulls the container and runs it.

Bill immediately discovers that that software is not suitable for his research, and finds a more appropriate one (as a container, of course!)

Bill spends the afternoon writing conclusions on his very important research using his new software while enjoying a hot cup of latte.

→ be like Bill.



Containers for the win

→ *Main concepts*

The idea of containers is to insulate a single process from your Operating System, and to:

- Let it live in its own space, including its own network;
- Let it have its own File System with its own libraries;
- Allow to natively access hardware without virtualization;
- Avoid booting an entire Virtual machine and to pre-allocate dedicated memory.

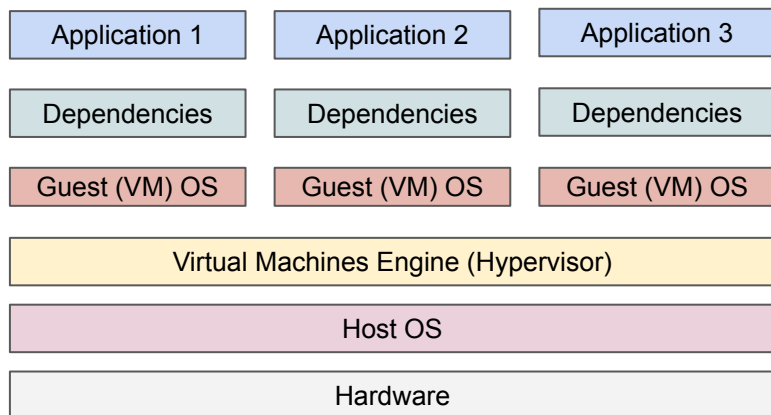
You might think about them as Virtual Machines in first approximation

→ *but keep in mind that they are two completely different beasts.*

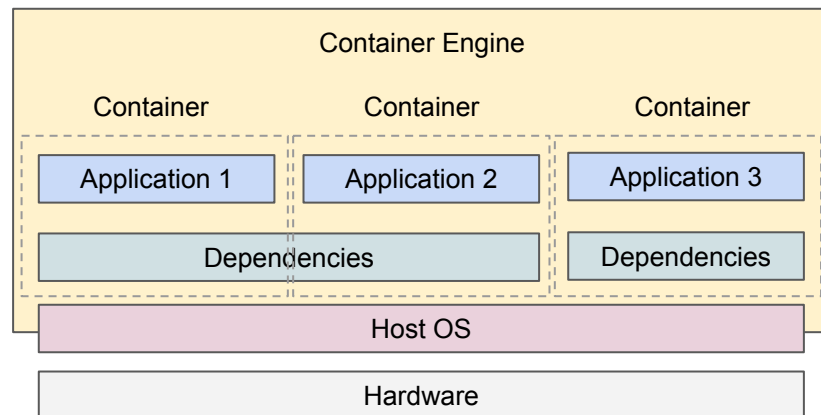
Containers for the win

→ *Main concepts*

Virtual Machines



Software Containers



Containers for the win

→ *Main concepts*

How to run a container?

- 1) Get or build a container image (think about it as a file)
- 2) Run the image: this is your container

Usually:

```
docker run my_container
```

..where “docker” can be replaced with your container engine of choice, e.g Podman.

Note: many engines, if cannot find the image locally, will automatically look online.

Containers for the win

→ *Main concepts*

Example: Docker hello world!

```
$ docker run hello-world
```

Containers for the win

→ *Main concepts*

Example: Docker hello world!

```
$ docker run hello-world

ste@Stes-MacAir:INAF $ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:6d60b42fdd5a0aa8a718b5f2eab139868bb4fa9a03c9fe1a59ed4946317c4318
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Containers for the win

→ *Main concepts*

```
$ docker run -it python:3.8
```

Containers for the win

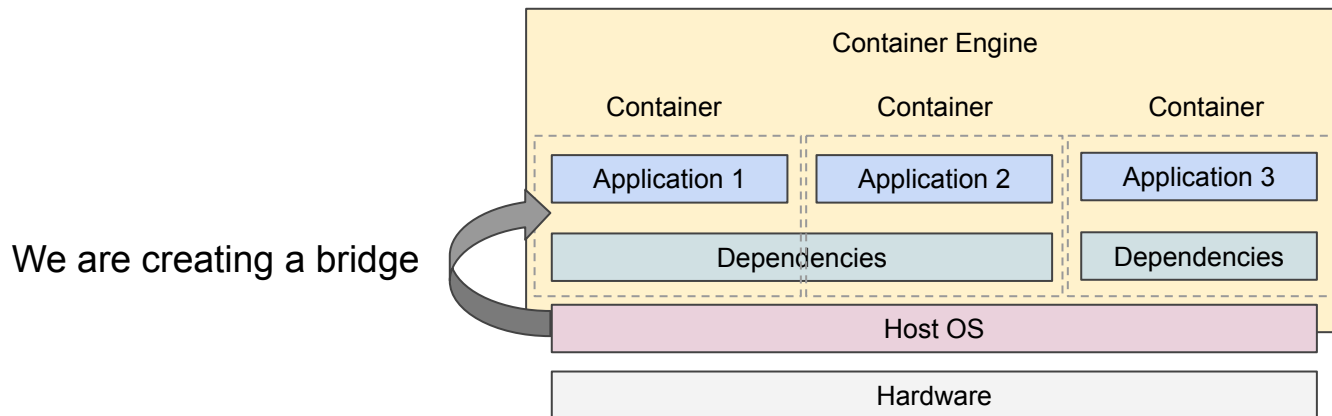
→ *Main concepts*

```
$ docker run --entrypoint /bin/bash -it python:3.8
```


Containers for the win

→ *Main concepts*

How to share files with a container? → **volumes**



Containers for the win

→ *Main concepts*

Example: make your home folder visible from within a container

```
$ docker run -it -v $HOME:/data python:3.8
```

Containers for the win

→ *Main concepts*

Example: make your home folder visible from within a container

```
$ docker run -it -v $HOME:/data python:3.8

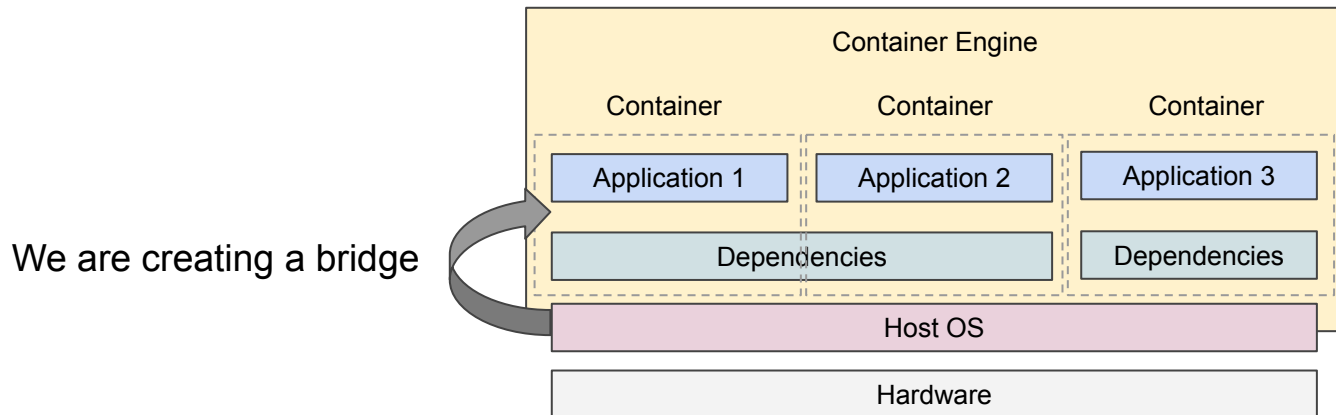
Python 3.8.12 (default, Dec 21 2021, 10:45:09)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.listdir('/data')
['Applications', 'Desktop', 'Documents', 'Downloads', 'Dropbox', 'iCloud',
'Library', 'Movies', 'Music', 'Pictures', 'Public']
```

Containers for the win

→ *Main concepts*

How to access servers* in a containers? → **port mapping**

*e.g. a Jupyter Notebook server



Containers for the win

→ *Main concepts*

Example: two Jupyter notebooks running with two Tensorflow versions

```
$ docker run -p 9001:8888 jupyter/tensorflow-notebook:tensorflow-2.4.1
```

```
$ docker run -p 9002:8888 jupyter/tensorflow-notebook:tensorflow-2.4.3
```

Containers for the win

→ *Main concepts*

Performance aspects

Testing a package without a container, on OSX

```
$ python3 -m unittest discover
```

```
.....  
-----  
Ran 90 tests in 41.405s
```

Testing a package inside a container on OSX: it is even faster!

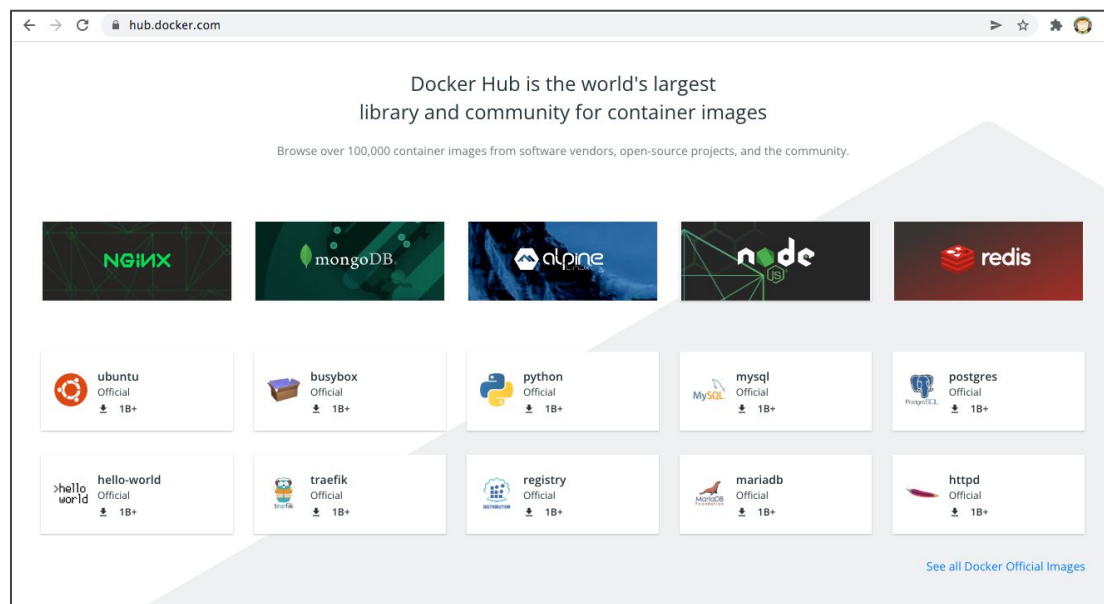
```
$ python3 -m unittest discover
```

```
.....  
-----  
Ran 90 tests in 34.108s
```

Containers for the win

→ *Main concepts*

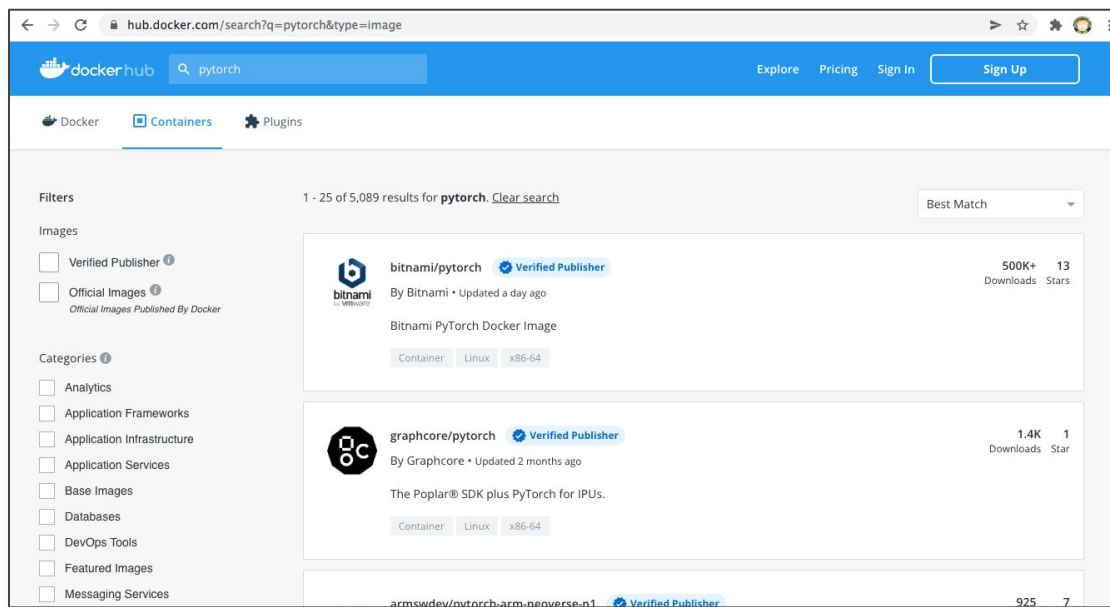
How to share containers? → **registries** (the GitHub for software containers)



Containers for the win

→ *Main concepts*

How to share containers? → **registries** (the GitHub for software containers)



Containers for the win

→ *Docker*

- Modern containerization solution, open source + freemium
- Extremely popular, the “de facto” containerization standard
- Incremental File System
- Plenty of software on Docker Hub
- Native on Linux
- Almost native on Macs post-2011 and Windows 10 (through a light VM)
 - Issues with new Apple M1 (ARM) chips!

Containers for the win

→ *Docker*

- Relies on a system daemon to manage containers
- Running containers are seen as (micro)services
- Containers have an IP address by default
- Extensive support for networking between containers
- Requires a *privileged* user (do not expect to do a “docker run” on clusters)
- Loads of orchestrators (docker-compose, kubernetes..)

Containers for the win

→ *Docker*

Common commands
<i>docker build</i> : Build a container
<i>docker pull</i> : Pull a container (from a registry)
<i>docker run</i> : Run a container (and execute the default command, or a custom one)
<i>docker ps</i> : List running containers
<i>docker exec</i> : Run a command in a running container
<i>docker stop</i> : Stop a running container
<i>docker rm</i> : Remove a container

Containers for the win

→ *Docker*

Isolation (to keep in mind)
Filesystem at runtime: completely isolated by default, use volumes to bind folders
Network: isolated within the Docker engine, use --net-host to use the host network
Environment at runtime: from scratch

Outline

- The dependency hell problem
 - Meet Mike
 - Solutions spectrum
- Containers for the win
 - Meet Bill
 - Main concepts
 - Docker
- Containers hands-on
 - Using containers
 - Building your first container
 - Sharing containers

Containers hands-on

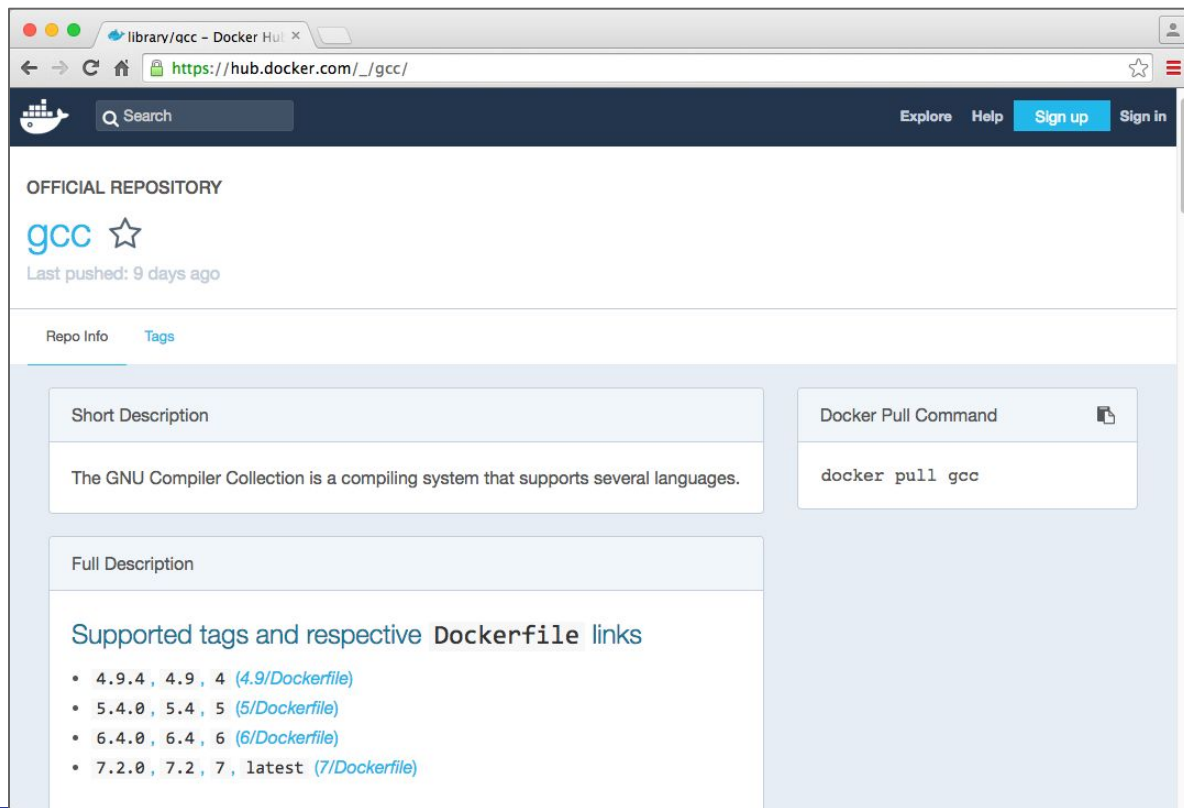
→ *Start*

- If you want to follow, ensure you can run the hello world

```
$ docker run hello-world
```

Containers hands-on

→ *Gcc on Docker Hub*



The screenshot shows the Docker Hub repository page for the 'gcc' image. The browser address bar shows 'https://hub.docker.com/_/gcc/'. The page features a search bar, navigation links for 'Explore', 'Help', 'Sign up', and 'Sign in'. The repository is identified as the 'OFFICIAL REPOSITORY' for 'gcc', with a star icon and a note that it was 'Last pushed: 9 days ago'. Below this, there are tabs for 'Repo Info' and 'Tags'. The main content area is divided into two columns. The left column contains a 'Short Description' box with the text 'The GNU Compiler Collection is a compiling system that supports several languages.' and a 'Full Description' box with the heading 'Supported tags and respective Dockerfile links' and a bulleted list of tags: '4.9.4, 4.9, 4 (4.9/Dockerfile)', '5.4.0, 5.4, 5 (5/Dockerfile)', '6.4.0, 6.4, 6 (6/Dockerfile)', and '7.2.0, 7.2, 7, latest (7/Dockerfile)'. The right column contains a 'Docker Pull Command' box with the command 'docker pull gcc' and a copy icon.

library/gcc - Docker Hub x

← → ↻ 📄 https://hub.docker.com/_/gcc/ ☆ ☰

🚢 Search Explore Help Sign up Sign in

OFFICIAL REPOSITORY

gcc ☆

Last pushed: 9 days ago

Repo Info Tags

Short Description

The GNU Compiler Collection is a compiling system that supports several languages.

Docker Pull Command 📄

```
docker pull gcc
```

Full Description

Supported tags and respective Dockerfile links

- 4.9.4, 4.9, 4 ([4.9/Dockerfile](#))
- 5.4.0, 5.4, 5 ([5/Dockerfile](#))
- 6.4.0, 6.4, 6 ([6/Dockerfile](#))
- 7.2.0, 7.2, 7, latest ([7/Dockerfile](#))

Containers hands-on

→ Gcc on Docker Hub (downloading)

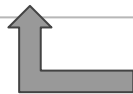
```
$ docker pull gcc:5.4
5.4: Pulling from library/gcc
aa18ad1a0d33: Extracting [=====>] 33.98 MB/52.6 MB
15a33158a136: Download complete
f67323742a64: Download complete
c4b45e832c38: Downloading [=====>] 51.59 MB/134.7 MB
e5d4afe2cf59: Download complete
4c0020714917: Downloading [=====>] 30.59 MB/200.4 MB
b33e8e4a2db2: Download complete
c8dae0da33c9: Waiting
```

- You are downloading a minimalistic Linux distribution (Debian Jessie, as we will see later) on which has been installed gcc (version 5.4).
- Thanks to Docker's incremental file system, another container based on Debian Jessie *will not* require to download/store it again.

Containers hands-on

→ *Gcc on Docker Hub (downloaded)*

```
$ docker pull gcc:5.4
5.4: Pulling from library/gcc
aa18ad1a0d33: Pull complete
15a33158a136: Pull complete
f67323742a64: Pull complete
c4b45e832c38: Pull complete
e5d4afe2cf59: Pull complete
4c0020714917: Pull complete
b33e8e4a2db2: Pull complete
c8dae0da33c9: Pull complete
Digest: sha256:e6ef7f0295b9d915f8521de360e30803bf8561cfb9cea8e320aa66761be8ec42
Status: Downloaded newer image for gcc:5.4
```



Terminology reminder:

- image: a “file” from which you can run a container
- container: an “entity” run from an image

Containers hands-on

→ *Run Gcc (5.4)*

```
$ docker run gcc:5.4 gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/local/libexec/gcc/x86_64-linux-gnu/5.4.0/lto-wrapper
Target: x86_64-linux-gnu
Configured with: /usr/src/gcc/configure --build=x86_64-linux-gnu --disable-multilib
--enable-languages=c,c++,fortran,go
Thread model: posix
gcc version 5.4.0 (GCC)
$
```

Containers hands-on

→ *Compile your code with Gcc (5.4)*

Our test.c code:

```
#include<stdio.h>

int main()
{
    printf("I run a very complex simulation and the result is 42\n");
}
```

Containers hands-on

→ *Compile your code with Gcc (5.4)*

```
$ docker run -v$PWD:/data gcc:5.4 gcc -o /data/test.bin --verbose /data/test.c
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/local/libexec/gcc/x86_64-linux-gnu/5.4.0/lto-wrapper
Target: x86_64-linux-gnu
Configured with: /usr/src/gcc/configure --build=x86_64-linux-gnu --disable-multilib
--enable-languages=c,c++,fortran,go
Thread model: posix
gcc version 5.4.0 (GCC)
COLLECT_GCC_OPTIONS='-o' '/data/Test/test.bin' '-v' '-mtune=generic' '-march=x86-64
[...]
```

Containers hands-on

→ *Run your code compiled with Gcc (5.4)*

On your computer → no!

```
$ Test/test.bin  
-bash: Test/test.bin: cannot execute binary file
```

Inside the container → yes!

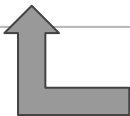
```
$ docker run -v$PWD:/data gcc:5.4 /data/test.bin  
ste@Stes-MacAir:Examples (master) $  
I just ran a very complex simulation and the result is 42
```

Containers hands-on

→ *Enter in the Gcc (5.4) container*

Execute a (bash) shell in the container

```
$ docker run -it gcc:5.4 bash  
root@b9c1414bab3d:/#
```



You are root!

List the root directories

```
root@b9c1414bab3d:/# ls  
bin boot dev etc home lib lib64 media mnt opt  
proc root run sbin srv sys tmp usr var
```

Containers hands-on

→ *Enter in the Gcc (5.4) container*

List running processes

```
root@b9c1414bab3d:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  1  13:54 pts/0        00:00:00 bash
root           8        1  0  13:54 pts/0        00:00:00 ps -ef
```

Get the container IP address

```
root@b9c1414bab3d:/# ip addr show dev eth0
[...]
inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
[...]
```

Containers hands-on

→ *Enter in the Gcc (5.4) container*

List running Docker containers (on another shell of your computer)

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
b9c1414bab3d  gcc:5.4  "bash"   3 seconds ago  Up 1 second           friendly_goodall
```

Exit the shell, and therefore the container

```
root@b9c1414bab3d:/# exit
$
```

When you exit a container, you lose every change to the container File System

Containers hands-on

→ *The Dockerfile*

library/gcc - Docker Hub x

← → ↻ 🏠 https://hub.docker.com/_/gcc/ ☆ ☰

🚢 Search Explore Help Sign up Sign in

OFFICIAL REPOSITORY

gcc ☆

Last pushed: 9 days ago

Repo Info Tags

Short Description

The GNU Compiler Collection is a compiling system that supports several languages.

Docker Pull Command

```
docker pull gcc
```

Full Description

Supported tags and respective Dockerfile links

- 4.9.4, 4.9, 4 ([4.9/Dockerfile](#))
- 5.4.0, 5.4, 5 ([5/Dockerfile](#))
- 6.4.0, 6.4, 6 ([6/Dockerfile](#))
- 7.2.0, 7.2, 7, latest ([7/Dockerfile](#))

Containers hands-on

→ *The Dockerfile*

- The *Dockerfile* is what defines a Docker Container. Think about it as its source code.
- When you build it, it generates a *Docker Image*. When you run a Docker Image, this “becomes” a *Docker Container*, as mentioned before.

```
FROM <base image>

RUN <a setup command>

COPY <source file/folder on your OS> <dest file/folder in the container>

RUN <another setup command>
```

Containers hands-on

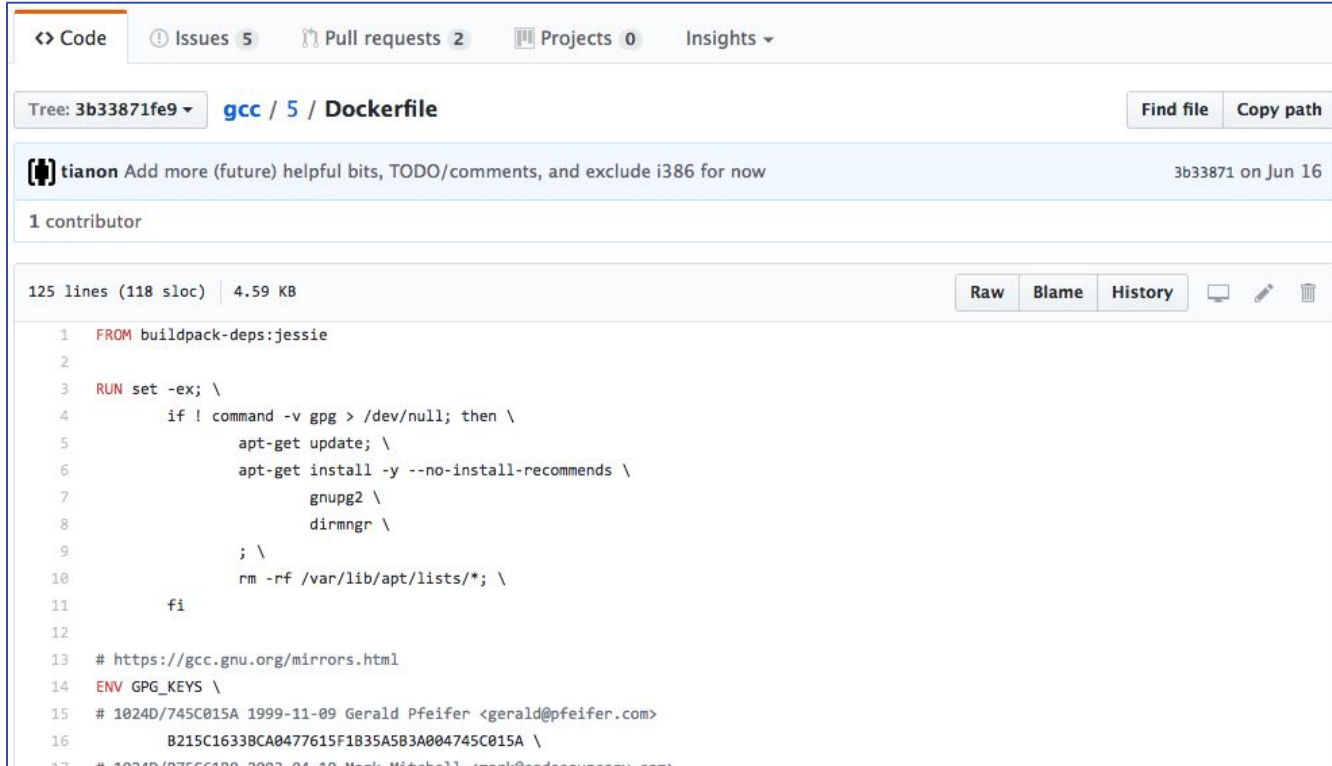
→ *On what is the Gcc (5.4) container built upon?*

There is NO black magic in Docker.

Now that we know that its source code is in the Dockerfile, we can see on what the Gcc (5.4) image is *built from*.

Containers hands-on

→ *On what is the Gcc (5.4) container built upon?*



Code Issues 5 Pull requests 2 Projects 0 Insights

Tree: 3b33871fe9 gcc / 5 / Dockerfile Find file Copy path

tianon Add more (future) helpful bits, TODO/comments, and exclude i386 for now 3b33871 on Jun 16

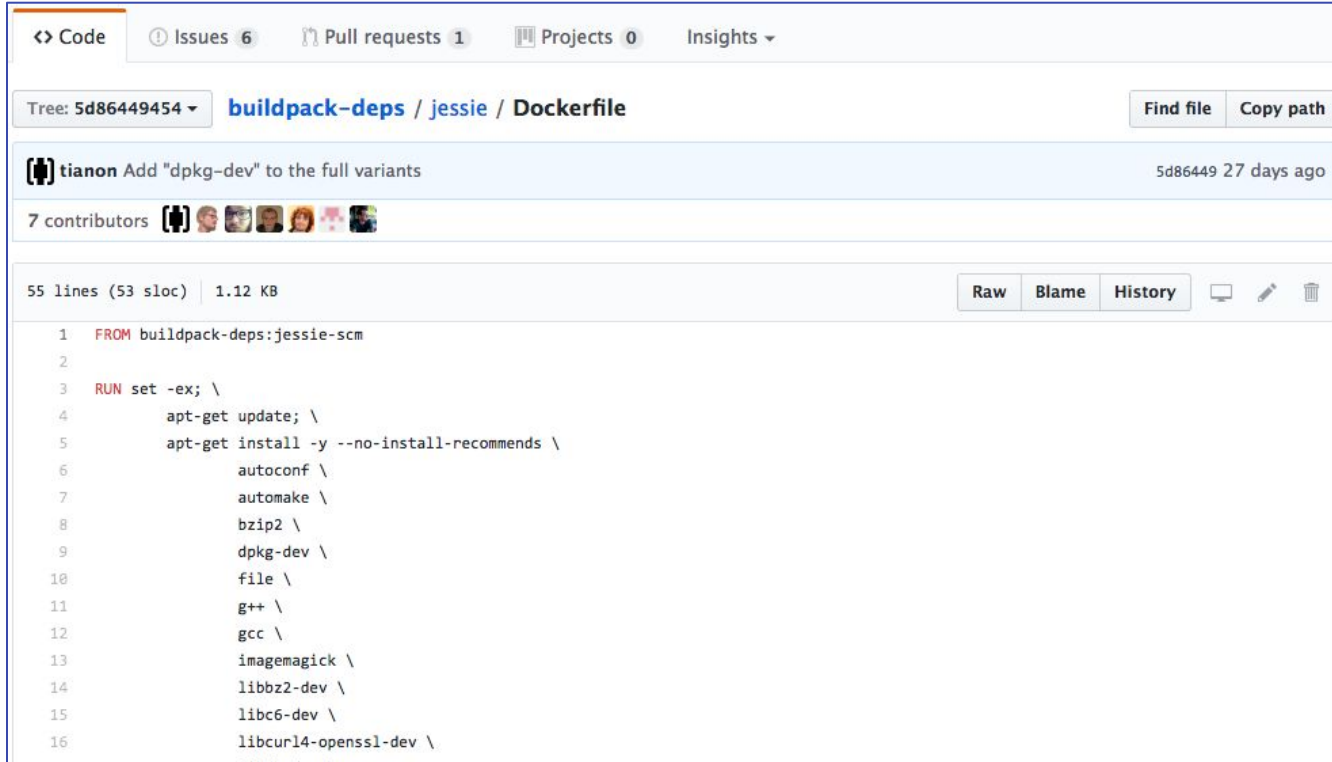
1 contributor

125 lines (118 sloc) 4.59 KB Raw Blame History

```
1 FROM buildpack-deps:jessie
2
3 RUN set -ex; \
4     if ! command -v gpg > /dev/null; then \
5         apt-get update; \
6         apt-get install -y --no-install-recommends \
7             gnupg2 \
8             dirmngr \
9         ; \
10        rm -rf /var/lib/apt/lists/*; \
11    fi
12
13 # https://gcc.gnu.org/mirrors.html
14 ENV GPG_KEYS \
15 # 1024D745C015A 1999-11-09 Gerald Pfeifer <gerald@pfeifer.com>
16 # B215C1633BCA0477615F1B35A5B3A004745C015A \
17 # 1024D875C6188 2003-04-10 Mark Mitchell <mark@codesourcery.com>
```

Containers hands-on

→ *On what is the Gcc (5.4) container built upon?*

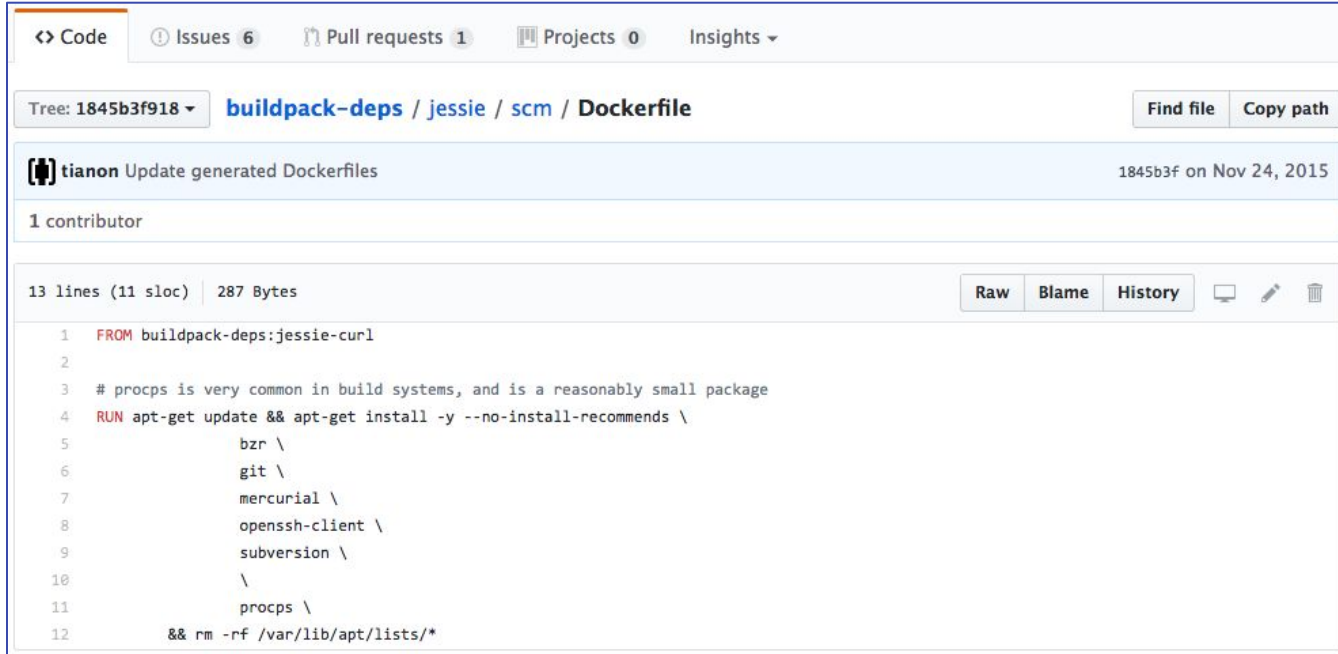


The screenshot shows a GitHub repository for 'buildpack-deps / jessie / Dockerfile'. A commit by 'tianon' is highlighted, titled 'Add "dpkg-dev" to the full variants'. The Dockerfile content is as follows:

```
1 FROM buildpack-deps:jessie-scm
2
3 RUN set -ex; \
4     apt-get update; \
5     apt-get install -y --no-install-recommends \
6         autoconf \
7         automake \
8         bzip2 \
9         dpkg-dev \
10        file \
11        g++ \
12        gcc \
13        imagemagick \
14        libbz2-dev \
15        libc6-dev \
16        libcurl4-openssl-dev \
```

Containers hands-on

→ *On what is the Gcc (5.4) container built upon?*



The screenshot shows a GitHub repository page for a Dockerfile. The repository is named 'buildpack-deps' and is located in the 'jessie' directory. The file is named 'Dockerfile'. The commit is by 'tianon' and is dated 'Nov 24, 2015'. The Dockerfile content is as follows:

```
1 FROM buildpack-deps:jessie-curl
2
3 # procps is very common in build systems, and is a reasonably small package
4 RUN apt-get update && apt-get install -y --no-install-recommends \
5     bzip \
6     git \
7     mercurial \
8     openssh-client \
9     subversion \
10    \
11    procps \
12    && rm -rf /var/lib/apt/lists/*
```

Containers hands-on

→ *On what is the Gcc (5.4) container built upon?*

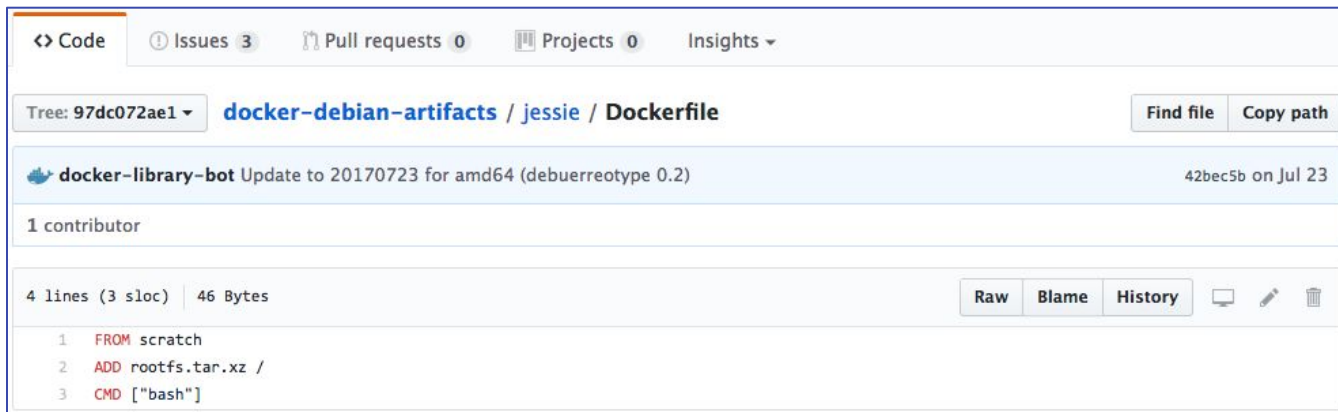


The screenshot shows a GitHub repository interface for the file `buildpack-deps / jessie / curl / Dockerfile`. The commit is by `yosifkit` with the message "Ensure gpg exists in curl variant" and a date of "9f60e19 on Jul 6". The file is 18 lines long and 349 bytes. The code is as follows:

```
1 FROM debian:jessie
2
3 RUN apt-get update && apt-get install -y --no-install-recommends \
4     ca-certificates \
5     curl \
6     wget \
7     && rm -rf /var/lib/apt/lists/*
8
9 RUN set -ex; \
10     if ! command -v gpg > /dev/null; then \
11         apt-get update; \
12         apt-get install -y --no-install-recommends \
13             gnupg2 \
14             dirmngr \
15         ; \
16     rm -rf /var/lib/apt/lists*; \
```

Containers hands-on

→ *On what is the Gcc (5.4) container built upon?*



The screenshot shows a GitHub repository page for `docker-debian-artifacts` in the `jessie` directory, specifically the `Dockerfile`. The repository is owned by `docker-library-bot` and has 3 issues, 0 pull requests, and 0 projects. The commit hash is `42bec5b` and it was pushed on July 23. The Dockerfile content is as follows:

```
1 FROM scratch
2 ADD rootfs.tar.xz /
3 CMD ["bash"]
```


Containers hands-on

→ *Your first Dockerfile*

We will now include and compile your test code directly from a Dockerfile

```
FROM gcc:5.4

# Add the test code
COPY test.c /opt

# Compile the test code
RUN gcc -v -o /opt/test.bin /opt/test.c
```

Containers hands-on

→ *Your first container*

Let's now build it. Place the Dockerfile and the "test.c" in a folder named "Test", then:

```
$ docker build Test -t testcontainer
Sending build context to Docker daemon 10.24kB
Step 1/3 : FROM gcc:5.4
---> b87db7824271
Step 2/3 : COPY test.c /opt
---> f5478f7830ee
Step 3/3 : RUN gcc -v -o /opt/test.bin /opt/test.c
---> Running in c839379f1fbc
Using built-in specs.
COLLECT_GCC=gcc
[...]
Removing intermediate container c839379f1fbc
---> 2f0c6f89fdc0
Successfully built 2f0c6f89fdc0
Successfully tagged testcontainer:latest
```

Containers hands-on

→ *Your first container*

..and run it:

```
$ docker run testcontainer /opt/test.bin  
I just ran a very complex simulation and the result is 42
```

Containers hands-on

→ *Your first container*

..and share it (old school):

```
$ docker save testcontainer > testcontainer.tar
```

```
$ docker load < testcontainer.tar
```

Containers hands-on

→ *Your first container*

..and share it (Docker Hub):

```
$ docker tag testcontainer sarusso/testcontainer
$ docker push sarusso/testcontainer
The push refers to repository [docker.io/sarusso/testcontainer]
4e139ce93449: Pushed
8e5d12c6cc1e: Pushed
531d0aa62df3: Mounted from library/gcc
2ac9aba62fc1: Mounted from library/gcc
4e778218c153: Mounted from library/gcc
8f816dba9ff6: Mounted from library/gcc
7381522c58b0: Mounted from library/gcc
ecd70829ec3d: Mounted from library/gcc
d70ce8b0dad6: Mounted from library/gcc
18f9b4e2e1bc: Mounted from library/gcc
latest: digest: sha256:21563d1b6645af4cf73f01cc471b5f1a8bb902f7f1903bac4b9b878433eecf5e size: 2421
```

Containers hands-on

→ *Versioning: hashes, tags, etc.*

If we rebuild the testcontainer, the caching jumps in. It takes few seconds.

```
$ docker build Test -t testcontainer
Sending build context to Docker daemon 10.24kB
Step 1/3 : FROM gcc:5.4
---> b87db7824271
Step 2/3 : COPY test.c /opt
---> Using cache
---> f5478f7830ee
Step 3/3 : RUN gcc -v -o /opt/test.bin /opt/test.c
---> Using cache
---> 2f0c6f89fdc0
Successfully built 2f0c6f89fdc0
Successfully tagged testcontainer:latest
```

..this is possible thanks to *version hashes*

Containers hands-on

→ *Versioning: hashes, tags, etc.*

- An *hash* is the result of applying an hash function
- An hash function takes some input and generates a fixed-size output, like:

47e0b9046c241cc4653b876c2a8ab01341c00754

- A good hash function allows to virtually never get the same hash from different inputs.
- In both Git and Docker the input is your code, and and hash represents a unique (saved) state. Or, a particular point in your codebase “history”.
- Then, it happens that hashes can be linked together, forming hierarchies.
- A *tag* is a friendly name for an hash.

Containers hands-on

→ *Versioning: hashes, tags, etc.*

Let's have a look at the hashes for the first and second build

```
$ docker build Test -t testcontainer
Sending build context to Docker daemon 10.24kB
Step 1/3 : FROM gcc:5.4
---> b87db7824271
Step 2/3 : COPY test.c /opt
---> f5478f7830ee
Step 3/3 : RUN gcc -v -o /opt/test.bin /opt/test.c
---> Running in c839379f1fbe
Using built-in specs.
COLLECT_GCC=gcc
[...]
Removing intermediate container c839379f1fbe
---> 2f0c6f89fdc0
Successfully built 2f0c6f89fdc0
Successfully tagged testcontainer:latest
```

```
$ docker build Test -t testcontainer
Sending build context to Docker daemon 10.24kB
Step 1/3 : FROM gcc:5.4
---> b87db7824271
Step 2/3 : COPY test.c /opt
---> Using cache
---> f5478f7830ee
Step 3/3 : RUN gcc -v -o /opt/test.bin /opt/test.c
---> Using cache
---> 2f0c6f89fdc0
Successfully built 2f0c6f89fdc0
Successfully tagged testcontainer:latest
```


Containers hands-on

→ *Versioning: hashes, tags, etc.*

- Both Git and Docker implement versioning with hashes, which are fully deterministic, unlike version (incremental) numbers.
- In the Docker ecosystem everything is versioned
- For practical use, also the short hashes are allowed (and commonly used), which are the first 7 characters for Git (i.e. “47e0b90”) and the first 12 for Docker.
- If by chance two hashes in the system starts with the same short hash, you will be required to enter one more character or the full hash.

Containers hands-on

→ Versioning: hashes, tags, etc.

library/gcc - Docker Hub

https://hub.docker.com/_/gcc/

OFFICIAL REPOSITORY

gcc ☆

Last pushed: 9 days ago

Repo Info Tags

Short Description

The GNU Compiler Collection is a compiling system that supports several languages.

Docker Pull Command

docker pull gcc

Full Description

Supported tags and respective Dockerfile links

- 4.9.4, 4.9, 4 (4.9/Dockerfile)
- 5.4.0, 5.4, 5 (5/Dockerfile)
- 6.4.0, 6.4, 6 (6/Dockerfile)
- 7.2.0, 7.2, 7, latest (7/Dockerfile)

```
$ docker build Test -t testcontainer
Sending build context to Docker daemon 10.24kB
Step 1/3 : FROM gcc:5.4
----> b87db7824271
Step 2/3 : COPY test.c /opt
----> f5478f7830ee
Step 3/3 : RUN gcc -v -o /opt/test.bin /opt/test.c
----> Running in c839379f1fbe
Using built-in specs.
COLLECT_GCC=gcc
[...]
Removing intermediate container c839379f1fbe
----> 2f0c6f89fdc0
Successfully built 2f0c6f89fdc0
Successfully tagged testcontainer:latest
```

p.s. the tag “5.4” for the gcc Docker container is actually saying that the tag is “gcc:54”. Sorry for this! :(

The hash for the tag “gcc:5.4” tag is “b87db7824271”

Containers hands-on

→ *Where do you save your Dockerfiles?*

Containers hands-on

→ *Where do you save your Dockerfiles?*

..on a versioning system.

Containers hands-on

→ *Where do you save your Dockerfiles?*

..on a versioning system.

There is no other alternative.

Do not work without versioning.

Seriously, don't.

→ Use Dropbox or Google Drive if you think that more professional versioning tools, like **Git**, are an overkill.

Containers hands-on

→ *The importance of versioning with Docker*

Docker allows to have everything up and running, including dependencies etc. with a single command.

This command trigger a build with a given set of dependencies (the ones you wrote to install in the Dockerfile)

Over time, you will probably make changes in your Dockerfiles and in your code.

If you use a versioning system, you can jump back in time to a particular version/hash, build it, and it will run exactly as it was running at that time

For managing multiple container versions simultaneously, you can use tags

Containers hands-on

→ *The importance of versioning with Docker*

Docker allows to have everything up and running, including dependencies etc. with a single command.

This command trigger a build with a given set of dependencies (the ones you wrote to install in the Dockerfile)

Over time, you will probably make changes in your Dockerfiles and in your code.

If you use a versioning system, you can jump back in time to a particular **version/hash**, build it, and it will run exactly as it was running at that time

For managing multiple container versions simultaneously, you can use **tags**

Containers hands-on

→ *Recap*

- 1) With Docker, your code will build and run in the exact same way, on every operating system, virtually forever.
- 2) If you want to give the code that generates the magic “42” answer to someone, they will just need two commands* to have everything up and running:

```
docker build or pull  
docker run
```

**plus some arguments*

Containers hands-on

→ *Recap*

- A versioning systems protects you first of all from yourself
- Using Docker with a versioning system allows to reach full reproducibility, starting from a repository name and a short hash for a point in time/version.
- Using them even for small personal/research projects helps a lot
- If someone gives you a code without version control *or* that requires dependencies:
 - First, put it under version control;
 - Second, create a Dockerfile with all the commands and dependencies you will need to set it up (which you will need anyway, by the way).
- ..and no, tomorrow you will not remember what you did.
No one does. :)

Thanks!

→ *Questions?*



stefano.russo@gmail.com



<https://sarusso.github.io>



https://twitter.com/_sarusso