Deterministic software development and management

Stefano Alberto Russo

stefano.russo@gmail.com

Talk repository: https://github.com/sarusso/DeterministicSoftware

About me

I holds a MSc in computational physics with a thesis on Big Data at CERN (Geneva, Switzerland) and a BSc in Computer Science with a thesis on High Performance Computing at SISSA (Trieste, Italy).

After university I joined first ATLAS as research affiliate and then CERN as research fellow in the joint industry-research Openlab project.

I have been a core team member of an energy metering and analytics startup, and I then joined Entrepreneur First, Europe’s best deep tech startup accelerator.

As side activities I contracted for EUMETSAT (through eXact Lab) and for Sky UK; I also lectured on Big Data for scientific computing at the Master In High Performance Computing co-organised by SISSA and UN’s International Centre for Theoretical Physics.

A note on the startup world

Tech

Use new technologies for another business, no or light PhDs presence.

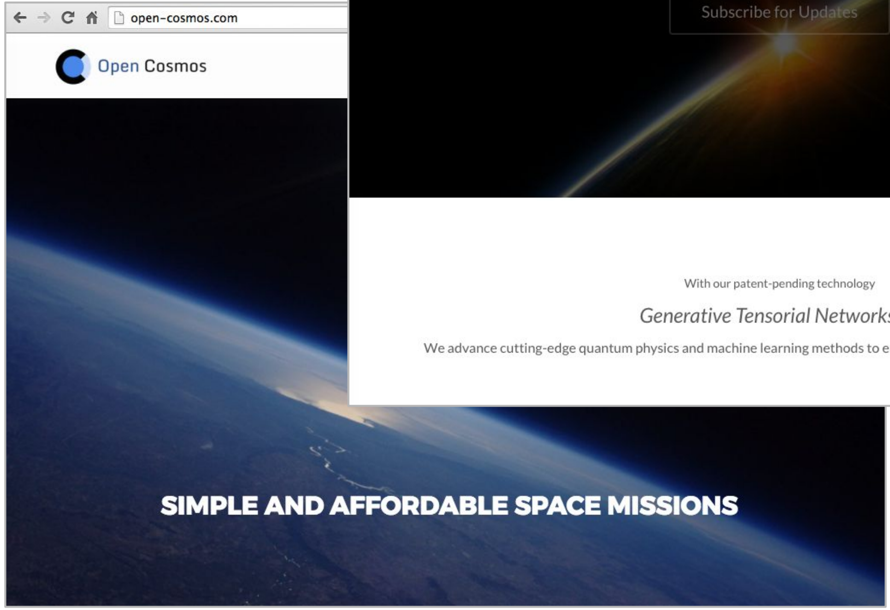
Examples:
- Facebook
- AirBnB
- Amazon (at the beginning)
- Twitter
- Uber
- ..and all the “yet another App”.

Deep Tech

Develop new technologies, extremely strong PhDs presence.

Examples:
- Google (at the beginning)
- Deep Mind
- Tesla
- SpaceX
- Human Longevity
- ..and all the university spin-offs.

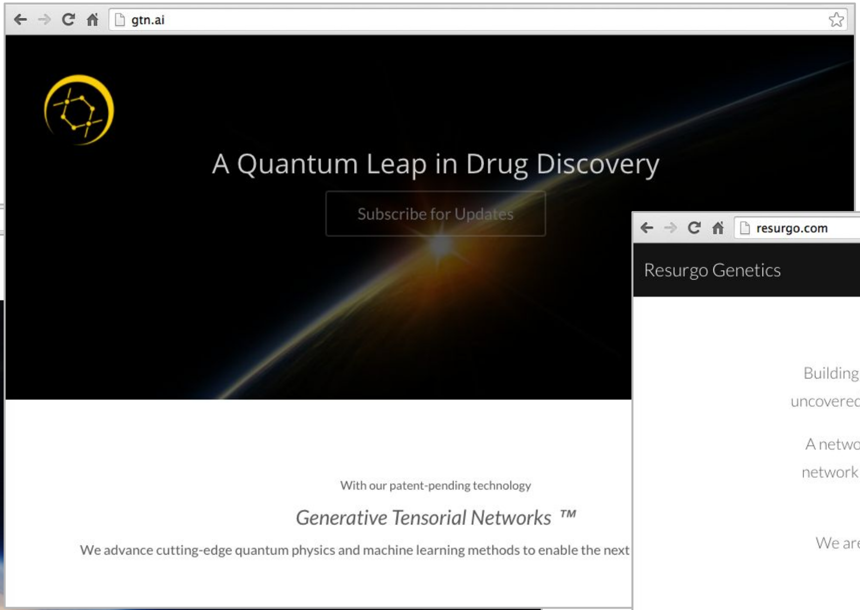
A note on the startup world



open-cosmos.com

Open Cosmos

SIMPLE AND AFFORDABLE SPACE MISSIONS



gtn.ai

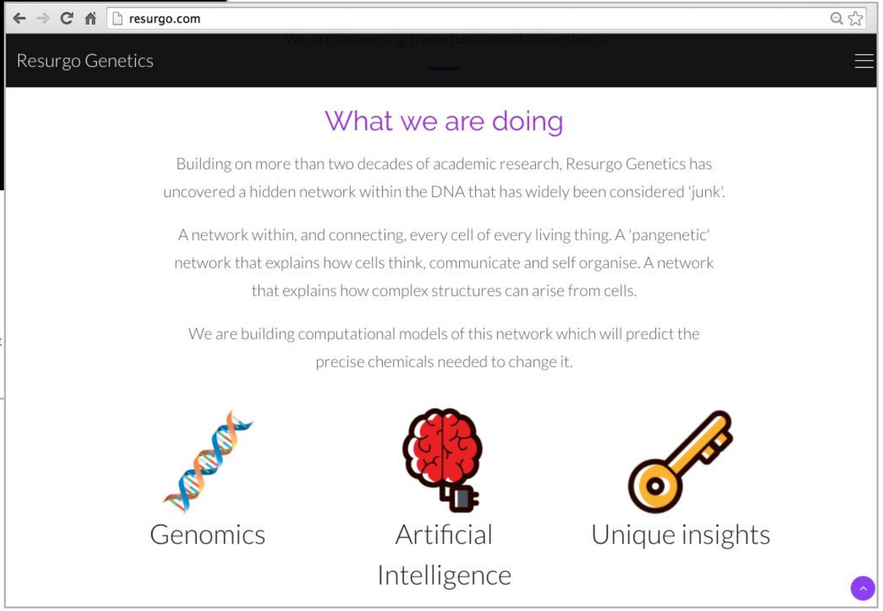
A Quantum Leap in Drug Discovery

Subscribe for Updates

With our patent-pending technology

Generative Tensorial Networks ™

We advance cutting-edge quantum physics and machine learning methods to enable the next



resurgo.com

Resurgo Genetics

What we are doing

Building on more than two decades of academic research, Resurgo Genetics has uncovered a hidden network within the DNA that has widely been considered 'junk'.

A network within, and connecting, every cell of every living thing. A 'pangenetic' network that explains how cells think, communicate and self organise. A network that explains how complex structures can arise from cells.

We are building computational models of this network which will predict the precise chemicals needed to change it.

Genomics

Artificial Intelligence

Unique insights

A note on the startup world

What I will show you in the following it is not only for Web/Mobile Apps.

..it is something very close to you.

..sounds familiar?

Mike wants to install a new software suggested by a colleague.

Mike cannot find a precompiled version.

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 expected.

How we could have avoided all this?

Problems:

1) Software could not be tried without investing time in setting everything up
2) Setting everything up required modifying the main OS
3) Modifying the main OS generated and inconsistent state

“Legacy” solutions:

- Try the library on your colleague's workstation
- Have a Sysadmin install the software for you (usually big corporations)
- Download a Virtual Machine either with the software already pre-installed or where the software is known to run out of the box. → *this one seems promising*

Downloading a Virtual Machine

- It works and does not touch your main OS
- Allows you to quickly test a given software / library
- Need to download a (big) binary image
- Not suitable for much more than just giving the software a try
- You will not find much software packaged in this way
- You need to trust who created the binary image

$\Rightarrow$ Not that good

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

Containerization: the Virtual Machines alternative

The idea 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 the need of booting an entire (binary) Virtual machine

Different containerization solutions put more or less focus on these points

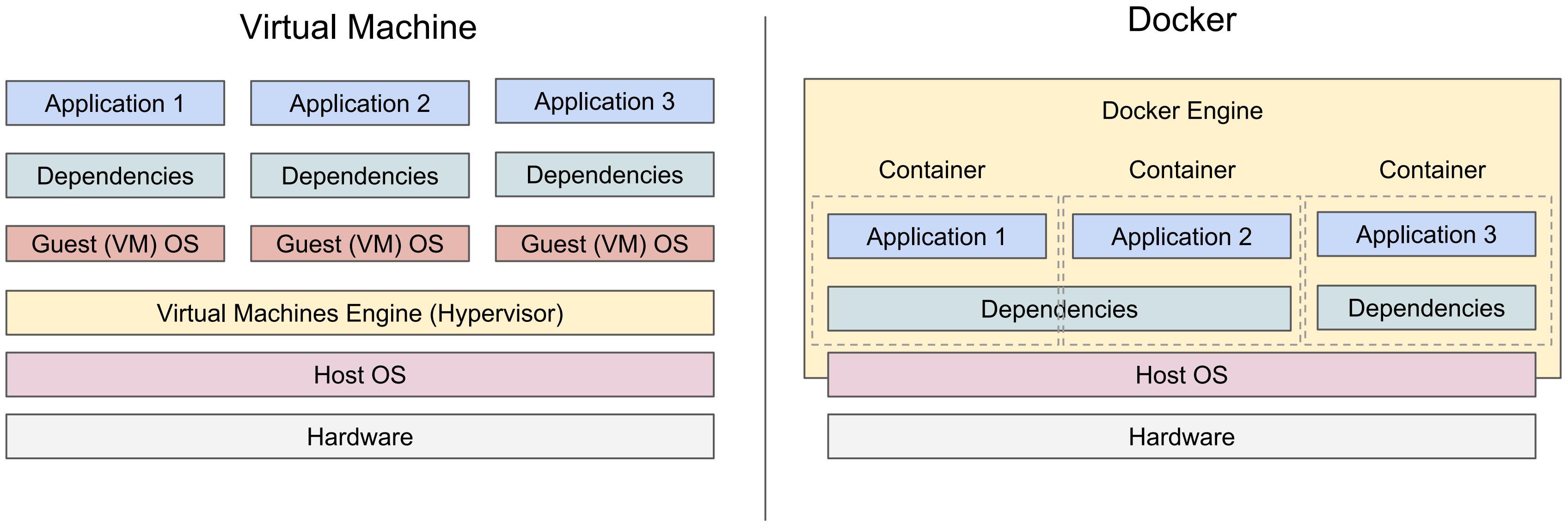
Docker

- Modern containerization solution, open source
- Extremely popular
- Incremental File System
- Plenty of software on Docker Hub
- Always possible to build from scratch, deterministically
- Native on Linux
- Almost native on Macs post-2011 and Windows 10 (through a light VM)

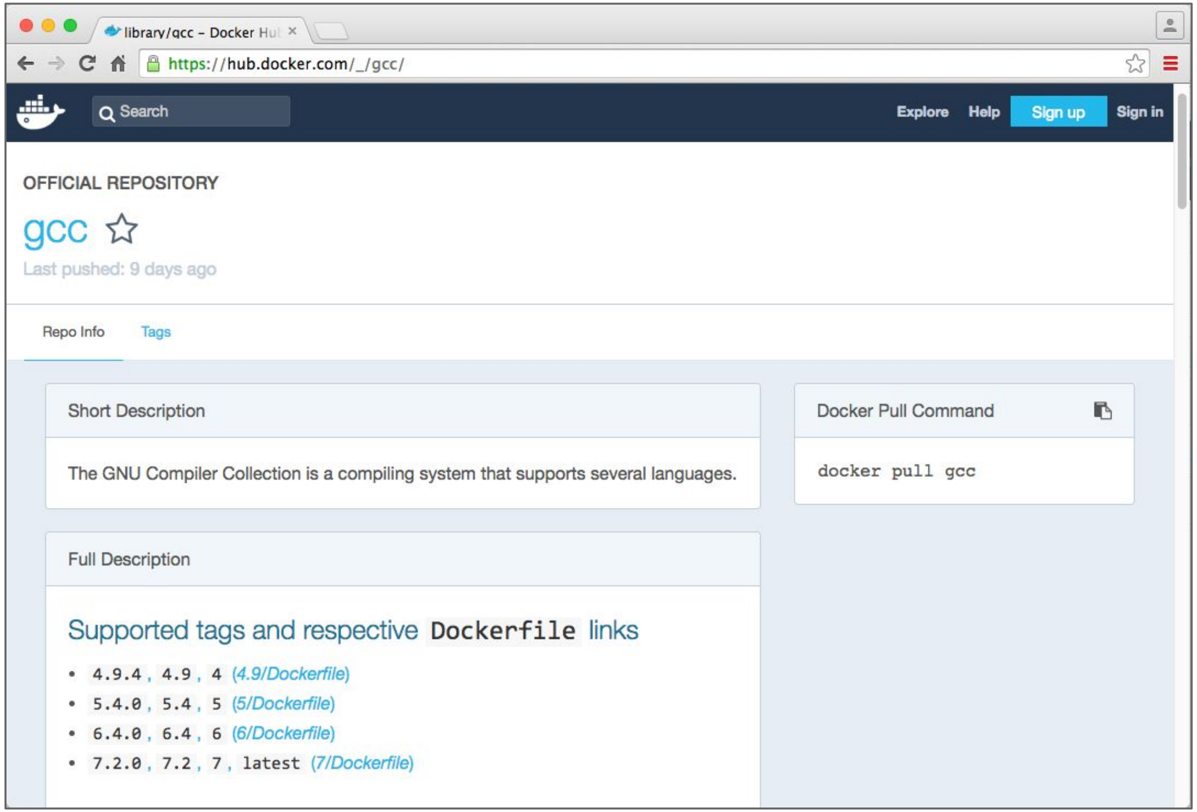
Docker

You *might* think about it as a Virtual Machine in first approximation

→ but remember that they are two completely different things



Gcc on Docker Hub



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

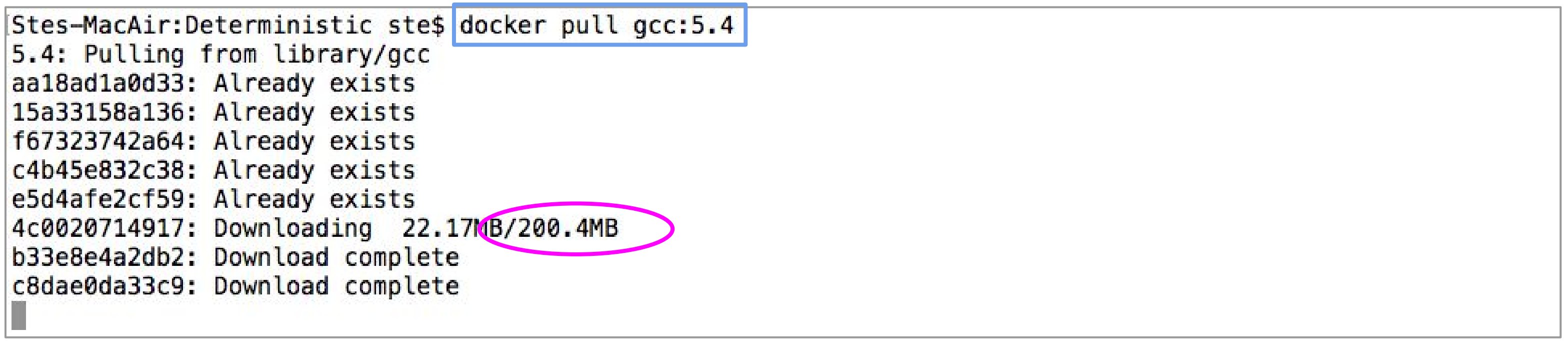
```
docker pull gcc
```

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)

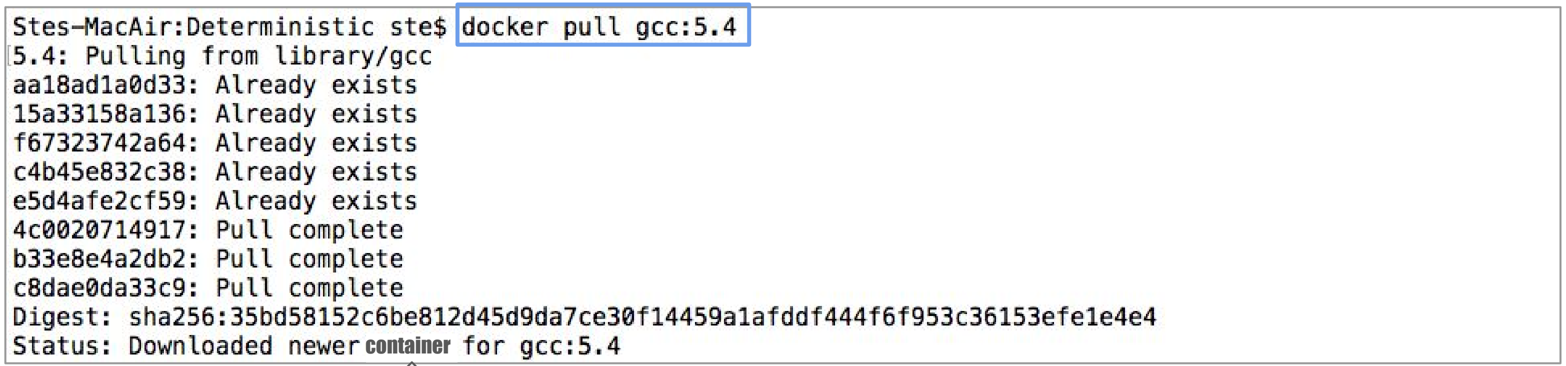
Gcc on Docker Hub



```
Stes-MacAir:Deterministic ste$ docker pull gcc:5.4
5.4: Pulling from library/gcc
aa18ad1a0d33: Already exists
15a33158a136: Already exists
f67323742a64: Already exists
c4b45e832c38: Already exists
e5d4afe2cf59: Already exists
4c0020714917: Downloading  22.17MB/200.4MB
b33e8e4a2db2: Download complete
c8dae0da33c9: Download complete
```

- You are downloading a minimalistic Linux distribution (Debian Jessie, as we will see later) which will serve for “containing” 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.

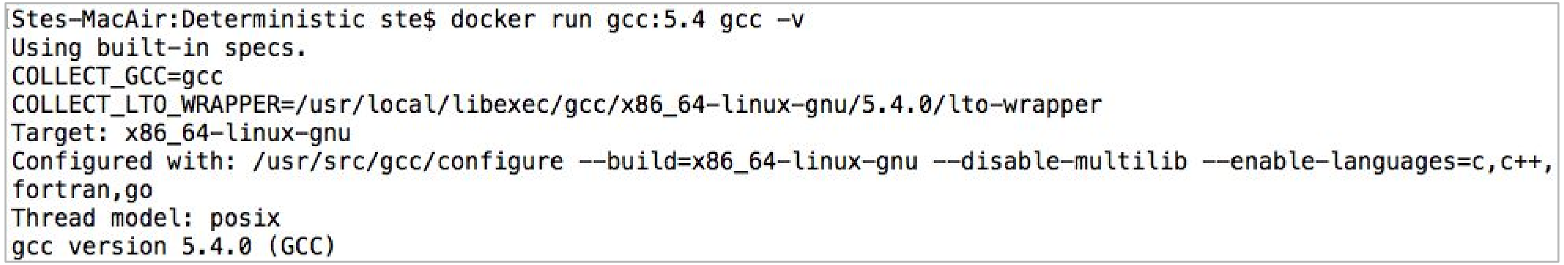
Gcc on Docker Hub



```
Stes-MacAir:Deterministic ste$ docker pull gcc:5.4
5.4: Pulling from library/gcc
aa18ad1a0d33: Already exists
15a33158a136: Already exists
f67323742a64: Already exists
c4b45e832c38: Already exists
e5d4afe2cf59: Already exists
4c0020714917: Pull complete
b33e8e4a2db2: Pull complete
c8dae0da33c9: Pull complete
Digest: sha256:35bd58152c6be812d45d9da7ce30f14459a1afddf444f6f953c36153efe1e4e4
Status: Downloaded newer container for gcc:5.4
```

Poetic licence for the sake of exposition clarity.
The real word will be unveiled in a couple of slides.

Run Gcc (5.4) with Docker



```
Stes-MacAir:Deterministic ste$ 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)
```

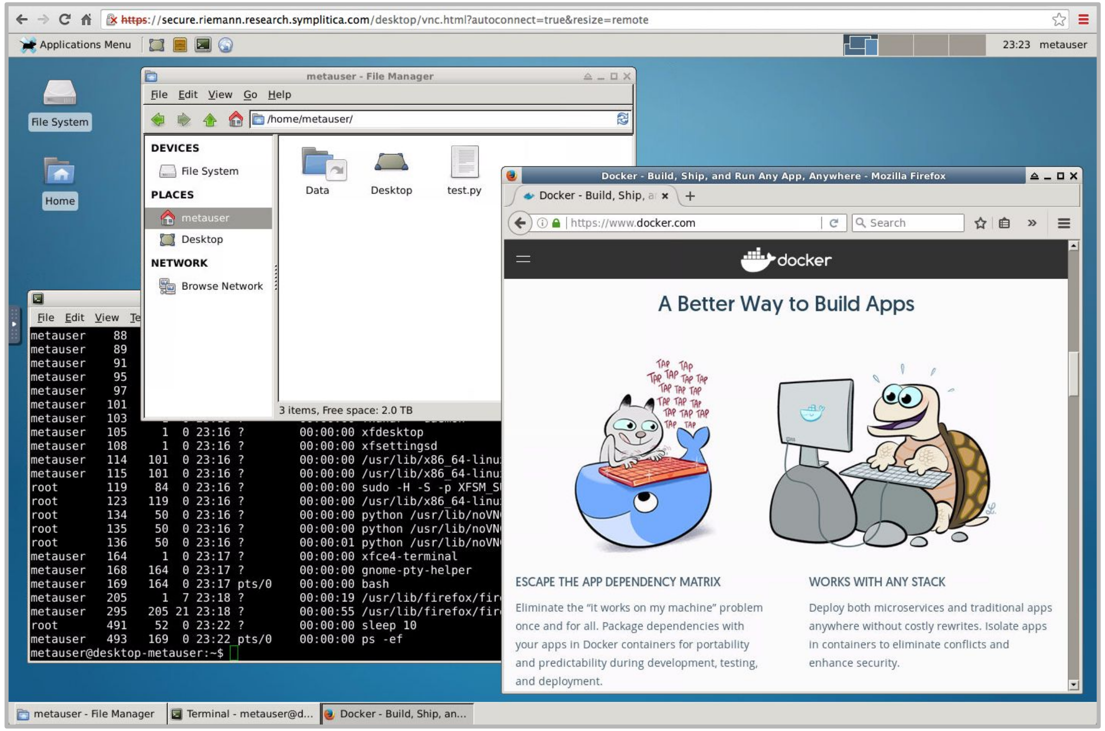
How powerful is Docker?

You can have an entire Virtual Desktop (running in your browser)

See this talk's repository for a working example
→ Directory Examples/VirtualDesktop

P.s. you will just need:

```
docker build . -t virtualdesktop
docker run -p8590:8590 virtualdesktop
```



```
metauser     88
metauser     89
metauser     91
metauser     95
metauser     97
metauser    101
metauser    103
metauser    105     1  0 23:16 ?        00:00:00 xfdesktop
metauser    108     1  0 23:16 ?        00:00:00 xfsettingsd
metauser    114   101  0 23:16 ?        00:00:00 /usr/lib/x86_64-linu
metauser    115   101  0 23:16 ?        00:00:00 /usr/lib/x86_64-linu
root        119    84  0 23:16 ?        00:00:00 sudo -H -S -p XFSM_S
root        123   119  0 23:16 ?        00:00:00 /usr/lib/x86_64-linu
root        134    50  0 23:16 ?        00:00:00 python /usr/lib/noVN
root        135    50  0 23:16 ?        00:00:00 python /usr/lib/noVN
root        136    50  0 23:16 ?        00:00:01 python /usr/lib/noVN
metauser    164     1  0 23:17 ?        00:00:00 xfce4-terminal
metauser    168   164  0 23:17 ?        00:00:00 gnome-pty-helper
metauser    169   164  0 23:17 pts/0    00:00:00 bash
metauser    205     1  7 23:18 ?        00:00:19 /usr/lib/firefox/fir
metauser    295   205 21 23:18 ?        00:00:55 /usr/lib/firefox/fir
root        491    52  0 23:22 ?        00:00:00 sleep 10
metauser    493   169  0 23:22 pts/0    00:00:00 ps -ef
metauser@desktop-metauser:~$
```

A Better Way to Build Apps

ESCAPE THE APP DEPENDENCY MATRIX

Eliminate the "it works on my machine" problem once and for all. Package dependencies with your apps in Docker containers for portability and predictability during development, testing, and deployment.

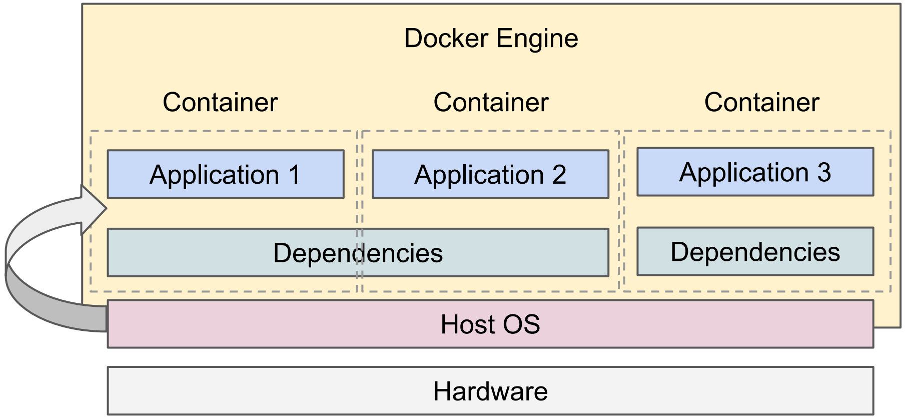
WORKS WITH ANY STACK

Deploy both microservices and traditional apps anywhere without costly rewrites. Isolate apps in containers to eliminate conflicts and enhance security.

Pass stuff to a Docker container: the volumes

The container is by definition insulated from your main (host) Operating System

- But you can make some folders visible from the containers as volumes (think about an usb pendrive)
- Just append "`-v your_os_folder:path_inside_the_container`" to docker command



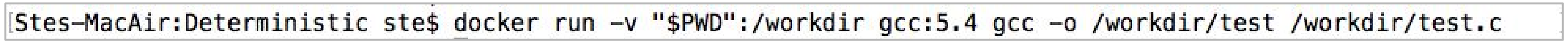
We are creating a bridge

Compile your code with Gcc (5.4)

Our test.c code:

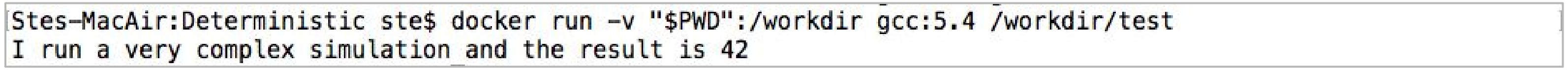
```c
#include<stdio.h>

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

Compile using volumes:



```
Stes-MacAir:Deterministic ste$ docker run -v "$PWD":/workdir gcc:5.4 gcc -o /workdir/test /workdir/test.c
```

Run using volumes:



```
Stes-MacAir:Deterministic ste$ docker run -v "$PWD":/workdir gcc:5.4 /workdir/test
I run a very complex simulation and the result is 42
```

Enter in the Gcc (5.4) container

Execute a (bash) shell in the container

```
$ docker run -t -i gcc:5.4 bash
root@28bdca47c6e6:/#
```

List running containers (on another shell

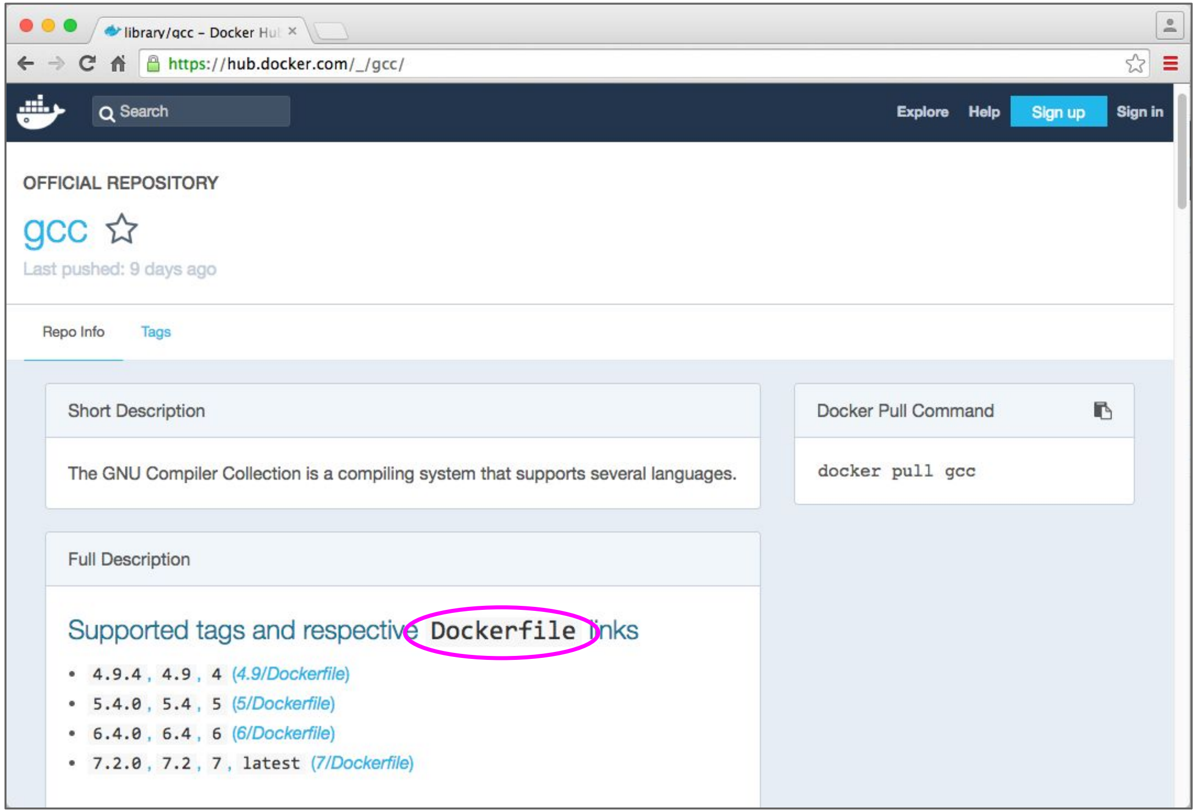
```
$ docker ps
CONTAINER ID    IMAGE    COMMAND    CREATED          STATUS        PORTS         NAMES
28bdca47c6e6   gcc:5.4  "bash"    3 seconds ago  Up 1 second               friendly_goodall
```

Exit the shell, and therefore the container

```
root@28bdca47c6e6:/# exit
$
```

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

The Dockerfile



OFFICIAL REPOSITORY

gcc

Last pushed: 9 days ago

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)

The Dockerfile

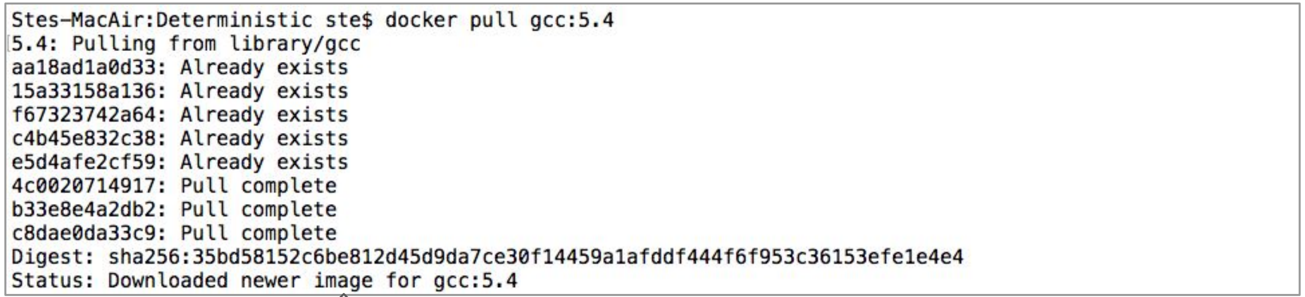
- The *Dockerfile* is what defines a Docker Container. Think about it as its source code.
- When you build it, it generates a *Docker Image*. When you run a Docker Image, this “becomes” a *Docker Container*.

```
FROM <base image>

RUN <a setup command>

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

RUN <another setup command>
```

On what is the Gcc (5.4) container based on?

There is NO black magic in Docker.

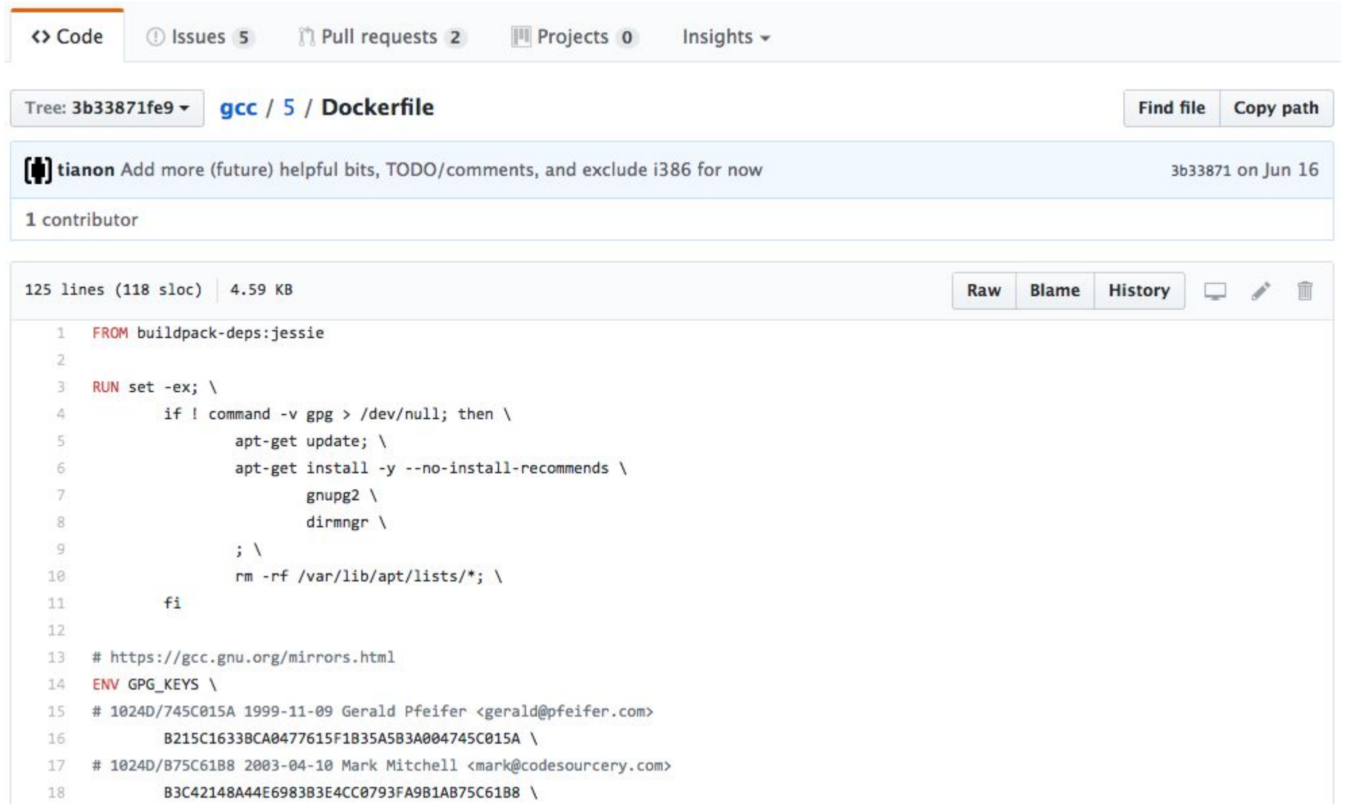
Now that we know that its source code is in the Dockerfile, we can see on what the Gcc (5.4) *image* is based on.



```
Stes-MacAir:Deterministic ste$ docker pull gcc:5.4
5.4: Pulling from library/gcc
aa18ad1a0d33: Already exists
15a33158a136: Already exists
f67323742a64: Already exists
c4b45e832c38: Already exists
e5d4afe2cf59: Already exists
4c0020714917: Pull complete
b33e8e4a2db2: Pull complete
c8dae0da33c9: Pull complete
Digest: sha256:35bd58152c6be812d45d9da7ce30f14459a1afddf444f6f953c36153efe1e4e4
Status: Downloaded newer image for gcc:5.4
```

Here the word was indeed image, not container.

On what is the Gcc (5.4) container based on?



gcc / 5 / Dockerfile

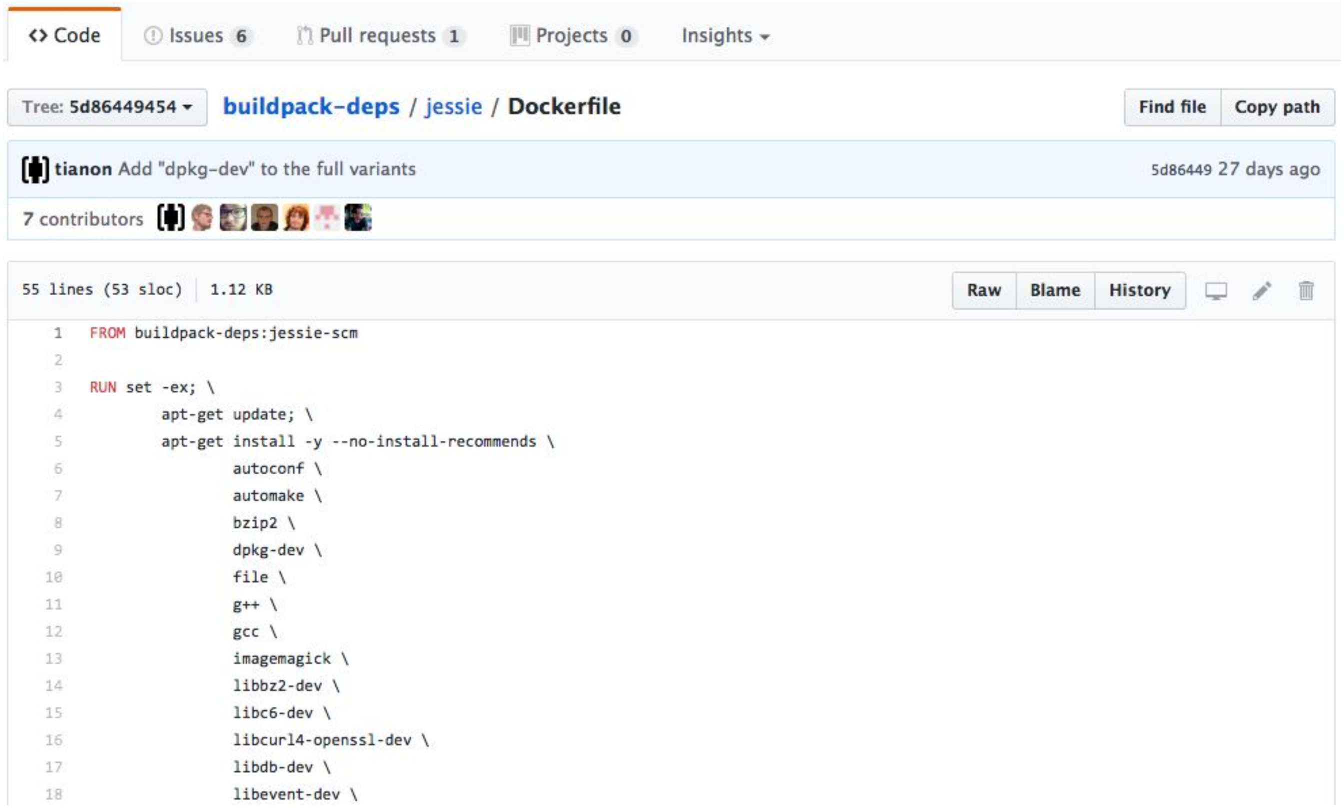
tianon Add more (future) helpful bits, TODO/comments, and exclude i386 for now 3b33871 on Jun 16

1 contributor

125 lines (118 sloc) | 4.59 KB

```
FROM buildpack-deps:jessie

RUN set -ex; \
	if ! command -v gpg > /dev/null; then \
		apt-get update; \
		apt-get install -y --no-install-recommends \
			gnupg2 \
			dirmngr \
		; \
		rm -rf /var/lib/apt/lists/*; \
	fi

# https://gcc.gnu.org/mirrors.html
ENV GPG_KEYS \
# 1024D/745C015A 1999-11-09 Gerald Pfeifer <gerald@pfeifer.com>
	B215C1633BCA0477615F1B35A5B3A004745C015A \
# 1024D/B75C61B8 2003-04-10 Mark Mitchell <mark@codesourcery.com>
	B3C42148A44E6983B3E4CC0793FA9B1AB75C61B8 \
```

On what is the Gcc (5.4) container based on?



Tree: 5d86449454 **buildpack-deps / jessie / Dockerfile**

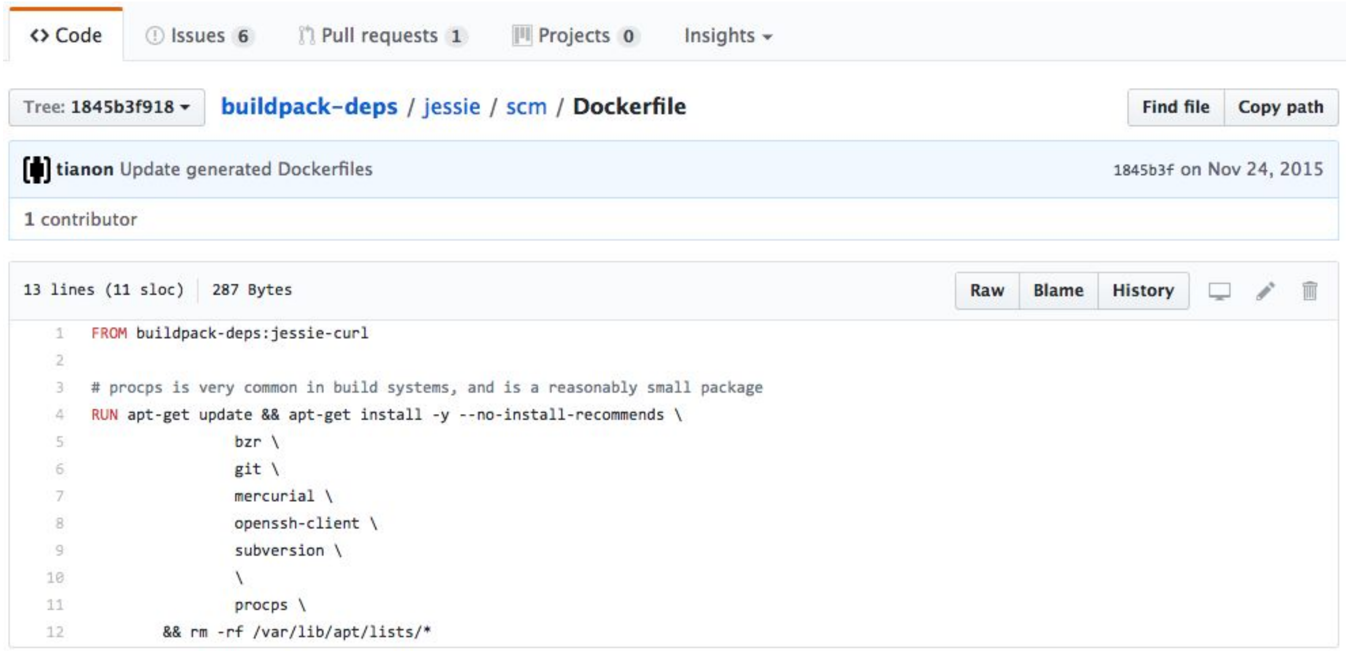
tianon Add "dpkg-dev" to the full variants 5d86449 27 days ago

7 contributors

55 lines (53 sloc) | 1.12 KB

```
FROM buildpack-deps:jessie-scm

RUN set -ex; \
	apt-get update; \
	apt-get install -y --no-install-recommends \
		autoconf \
		automake \
		bzip2 \
		dpkg-dev \
		file \
		g++ \
		gcc \
		imagemagick \
		libbz2-dev \
		libc6-dev \
		libcurl4-openssl-dev \
		libdb-dev \
		libevent-dev \
```

On what is the Gcc (5.4) container based on?



buildpack-deps / jessie / scm / Dockerfile

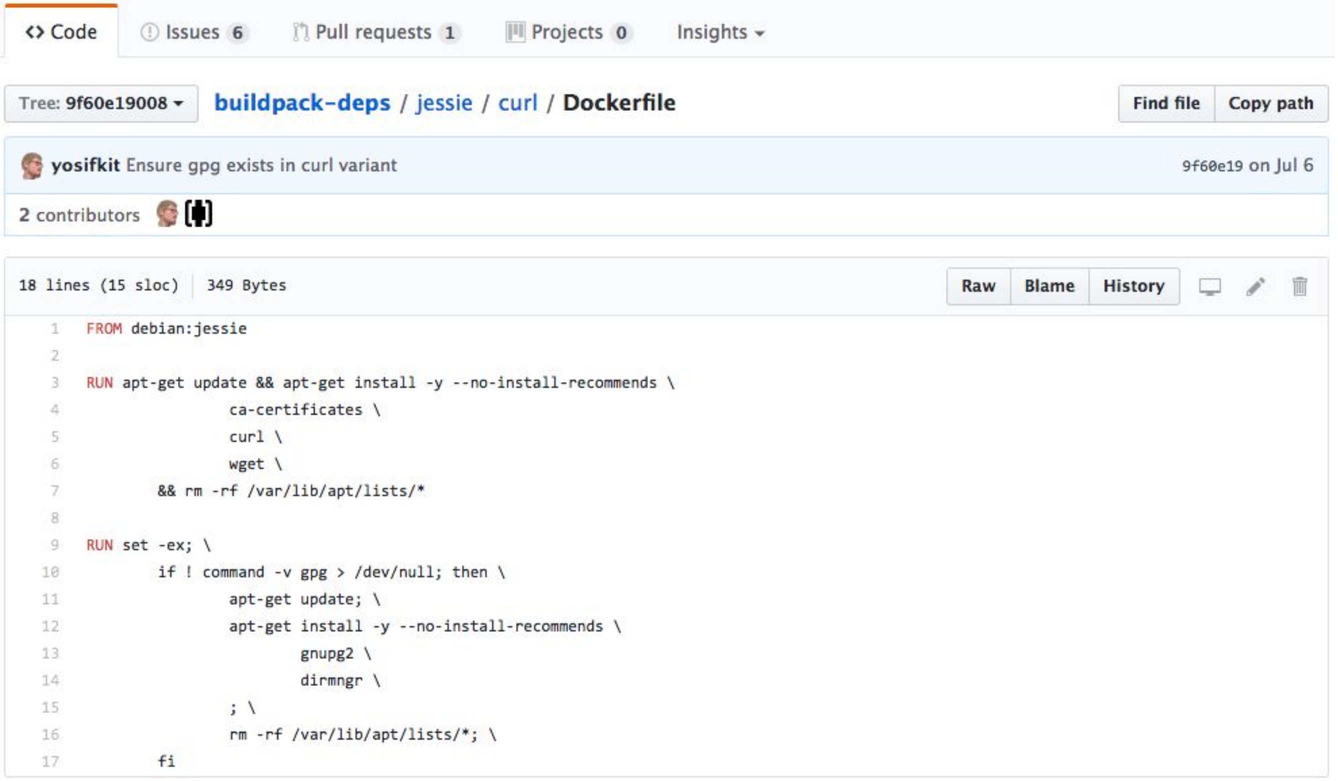
tianon Update generated Dockerfiles   1845b3f on Nov 24, 2015

1 contributor

13 lines (11 sloc) | 287 Bytes

```
FROM buildpack-deps:jessie-curl

# procps is very common in build systems, and is a reasonably small package
RUN apt-get update && apt-get install -y --no-install-recommends \
		bzr \
		git \
		mercurial \
		openssh-client \
		subversion \
		\
		procps \
	&& rm -rf /var/lib/apt/lists/*
```

On what is the Gcc (5.4) container based on?



buildpack-deps / jessie / curl / Dockerfile

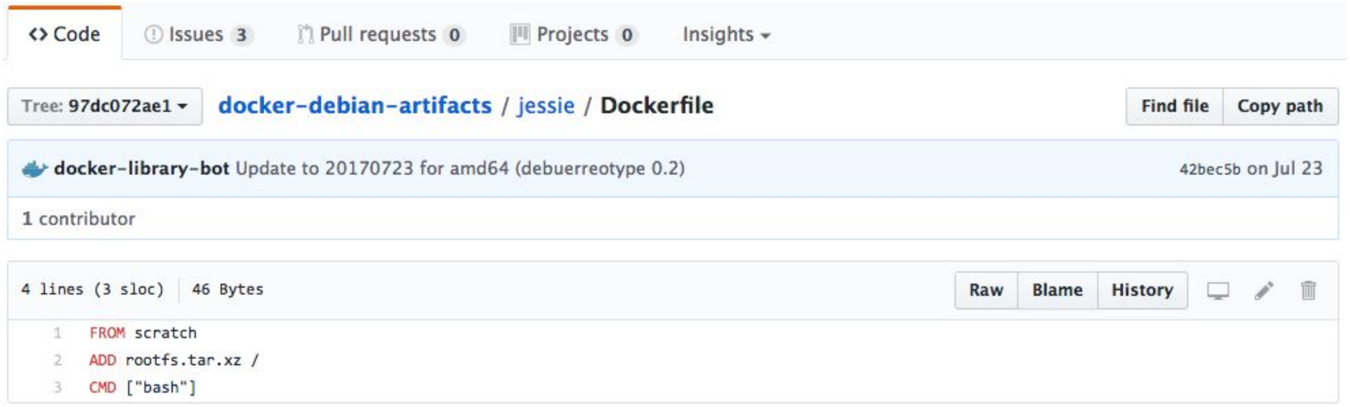
yosifkit Ensure gpg exists in curl variant 9f60e19 on Jul 6

2 contributors

18 lines (15 sloc) 349 Bytes

```
FROM debian:jessie

RUN apt-get update && apt-get install -y --no-install-recommends \
		ca-certificates \
		curl \
		wget \
	&& rm -rf /var/lib/apt/lists/*

RUN set -ex; \
	if ! command -v gpg > /dev/null; then \
		apt-get update; \
		apt-get install -y --no-install-recommends \
			gnupg2 \
			dirmngr \
		; \
		rm -rf /var/lib/apt/lists/*; \
	fi
```

On what is the Gcc (5.4) container based on?



docker-debian-artifacts / jessie / Dockerfile

docker-library-bot Update to 20170723 for amd64 (debuerreotype 0.2) 42bec5b on Jul 23

1 contributor

4 lines (3 sloc) | 46 Bytes

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

Your first Dockerfile

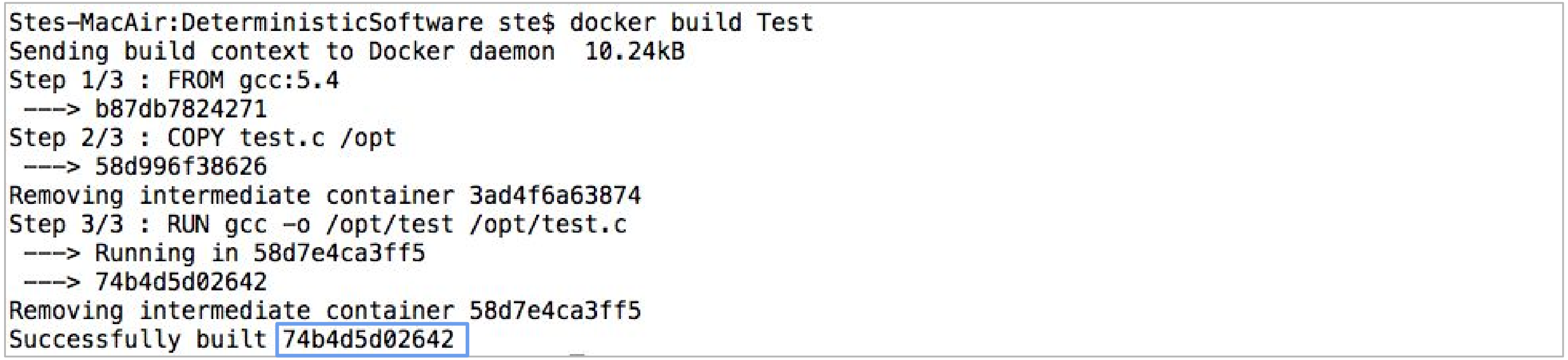
We will now include and compile your code directly from a Dockerfile

```
FROM gcc:5.4

# Add the test code
COPY test.c /opt

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

Your first Dockerfile

Let’s now build it.

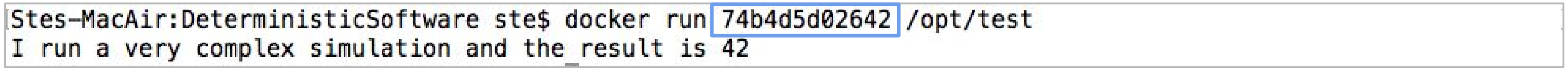
Assuming you have the“test.c” and the Dockerfile files in a folder named “Test”:



```
Stes-MacAir:DeterministicSoftware ste$ docker build Test
Sending build context to Docker daemon  10.24kB
Step 1/3 : FROM gcc:5.4
 ---> b87db7824271
Step 2/3 : COPY test.c /opt
 ---> 58d996f38626
Removing intermediate container 3ad4f6a63874
Step 3/3 : RUN gcc -o /opt/test /opt/test.c
 ---> Running in 58d7e4ca3ff5
 ---> 74b4d5d02642
Removing intermediate container 58d7e4ca3ff5
Successfully built 74b4d5d02642
```

Your first Dockerfile

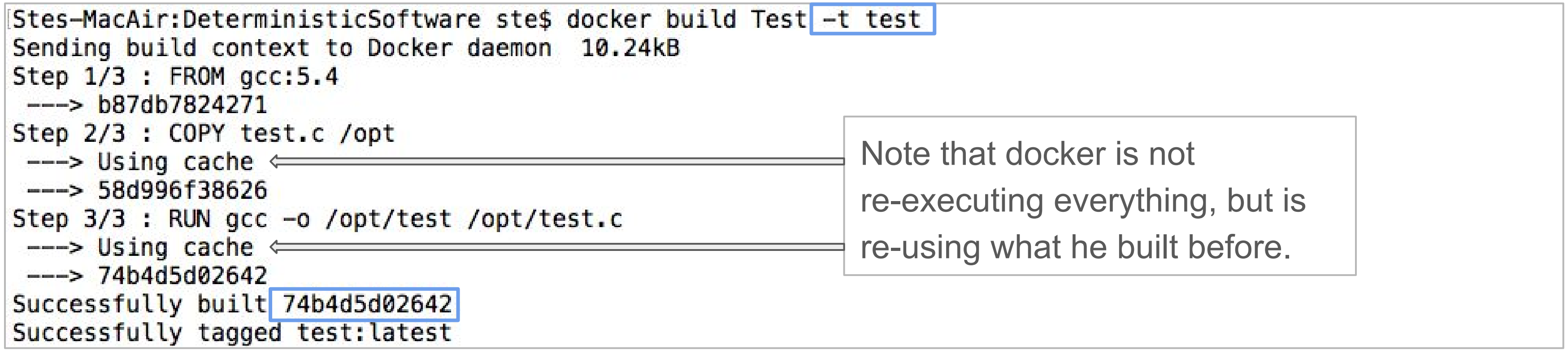
..and we can run it:



```
Stes-MacAir:DeterministicSoftware ste$ docker run 74b4d5d02642 /opt/test
I run a very complex simulation and the result is 42
```

Your first Dockerfile

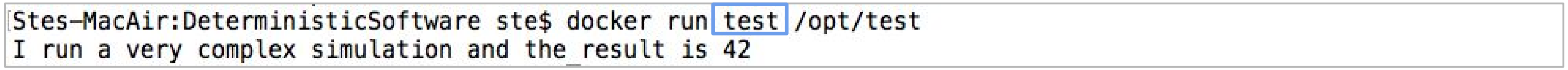
We can also tag the image when we build it, so we can run it more easily:



```
Stes-MacAir:DeterministicSoftware ste$ docker build Test -t test
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
 ---> 58d996f38626
Step 3/3 : RUN gcc -o /opt/test /opt/test.c
 ---> Using cache
 ---> 74b4d5d02642
Successfully built 74b4d5d02642
Successfully tagged test:latest
```

Note that docker is not re-executing everything, but is re-using what he built before.

Now, as with the Gcc container, we don’t need a strange reference for the image:



```
Stes-MacAir:DeterministicSoftware ste$ docker run test /opt/test
I run a very complex simulation and the result is 42
```

Recap on Docker

This is very nice, but what is the real bargain?

1) With Docker, this stuff will build and run in the exact same way, on every operating system, virtually forever.

2) If you want to give the code that generate the magic “42” answer to someone, they will just need two commands* to have everything up and running:

```
docker build
docker run
```

**plus some arguments*

Where do you save your code and Dockerfile?

Where do you save your code and Dockerfile?

..on a versioning system.

Where do you save your code and Dockerfile?

..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.

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

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**

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:

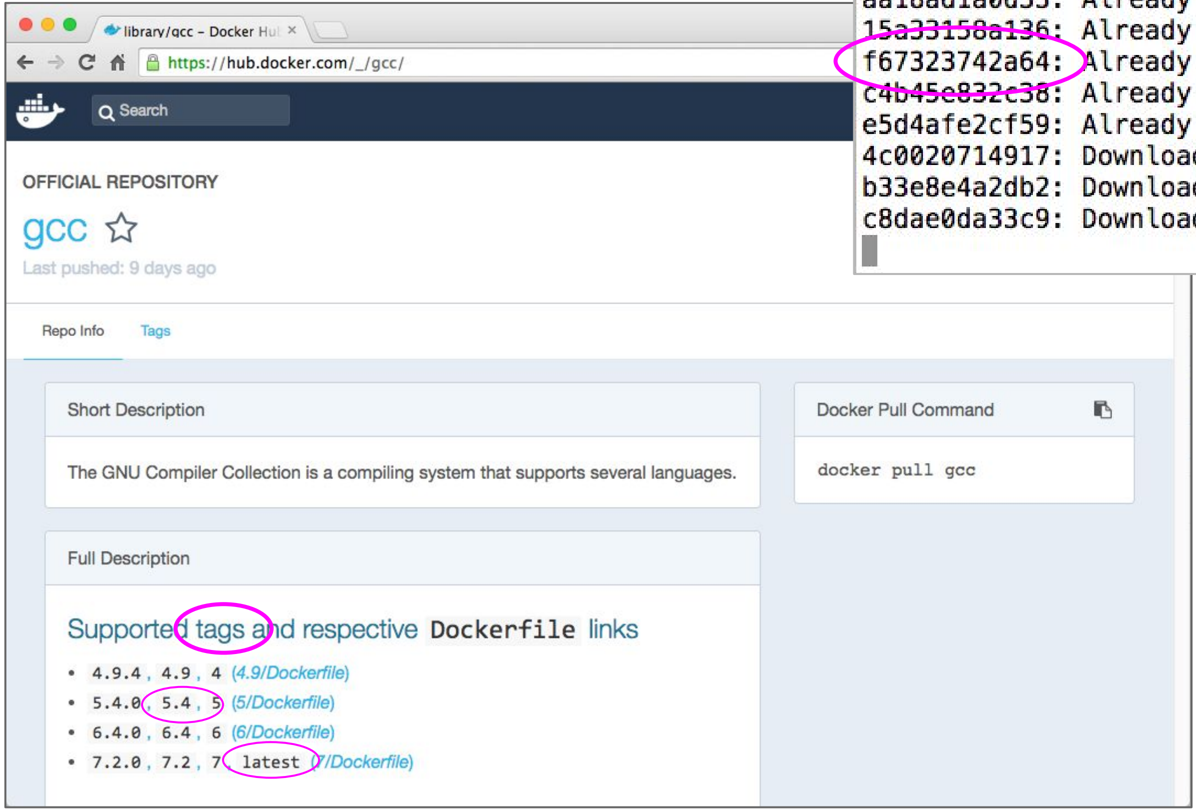
```
47e0b9046c241cc4653b876c2a8ab01341c00754
```

- A good hash function allows to virtually never have 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.

Versioning: hashes, tags, etc.

- Being relatively new projects, 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.

One step back



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.

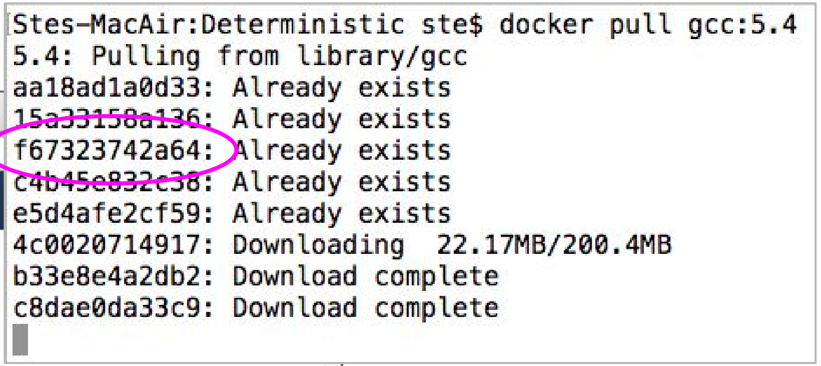
Docker Pull Command

```
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)



```
Stes-MacAir:Deterministic ste$ docker pull gcc:5.4
5.4: Pulling from library/gcc
aa18ad1a0d33: Already exists
15a33158a136: Already exists
f67323742a64: Already exists
c4b45e832c38: Already exists
e5d4afe2cf59: Already exists
4c0020714917: Downloading  22.17MB/200.4MB
b33e8e4a2db2: Download complete
c8dae0da33c9: Download complete
```

Git in one slide (for personal use)

Open and distributed versioning system, standard for all modern development.

1) Create a repo on GitHub, BitBucket, GitLab or similar and note down the repo URL
2) In your workspace:

```
# git clone <your repo https or ssh URL>
```

3) Now enter the new folder which has been created and create a file named “test.py”, then:

```
# git add test.py && git commit test.py -m “My first commit”
```

4) We saved the changes *locally*, and we have an hash for this saved state that we can see with:

```
# git log
```

5) We now push the saved state to the server

```
# git push
```

Git survival in another slide (still for personal use)

1) You can revert to your code at any point in time by using:

```
# git pull <hash or short hash>
```

2) You will not be allowed to revert to a point in time if you have modified files under Git version control. If this happens first save your modified files somewhere else, then undo your changes with:

```
# git checkout -- your_modified_file1
# git checkout -- your_modified_file2
```

You can now pull as stated at point 1).

3) If you get a conflict, until you resolve it you will not be able to use Git anymore. In this case the best shot for you is to start fresh by cloning again the repository. Do not trash the folder for “old” one, just rename it so you can try to see what conflicted and re-apply the changes if necessary.

Disclaimer: these are quick and dirty tricks, you should study how to use Git properly!

Other best practices

Explicit variables names and extensive comments

→ Ok for indexes, no var1, a, myvar, count etc. Better “this_var_holds_this_stuff”

Use an Integrated Development Environment (IDE)

Implement basic, end to-end testing (unit testing as second step)

→ testing is an entire profession, but just a simple script which run your code against a reference input/output help to spot bugs in due course.

Use verbose logging with context information

→ No “error”. Instead, “Error of type X when doing Y with this_var_holds_this_stuff=Z”

Use high level when possible (Python)

Take home messages

- 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.
- Use them even for small personal/research projects, and 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. No one does.

Hope it helps :)

Questions?

Extra slides

What is running inside a container?

```
root@28bdca47c6e6:/#root@28bdca47c6e6:/# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 17:44 pts/0    00:00:00 bash
root         5     1  0 18:32 pts/0    00:00:00 ps -ef
```