Container Primer

Giuliano Taffoni

Hands on

To start!

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

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

Hands on: gcc

library/gcc - Docker Hu

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)

Get GCC image

```
$ 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.

Get gcc Image

```
$ 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
```

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

Run gcc

```
$ 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)
$
```

Prepare a test code…

```
#include<stdio.h>
int main() {
     printf("I run a very complex simulation
and the result is 42\n");
}
```

Compile the code

```
$ 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
[...]
$
```

Run your code…

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
```

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
```

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
    [...]
```

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

The Dockerfile

library/gcc - Docker Hub

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)

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>
```

Dockerfile examples

- On what is the Gcc (5.4) container built upon?
- Explore the leafs

Tree: 3b33871fe9 ▾ gcc / 5 / Dockerfile Find file Copy p

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

1 contributor

125 lines (118 sloc) | 4.59 KB Raw Blame History

```
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 \
```

Dockerfile examples

- leaf 2

Tree: 5d86449454 ▾ **buildpack-deps** / jessie / **Dockerfile** Find file Copy

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

7 contributors

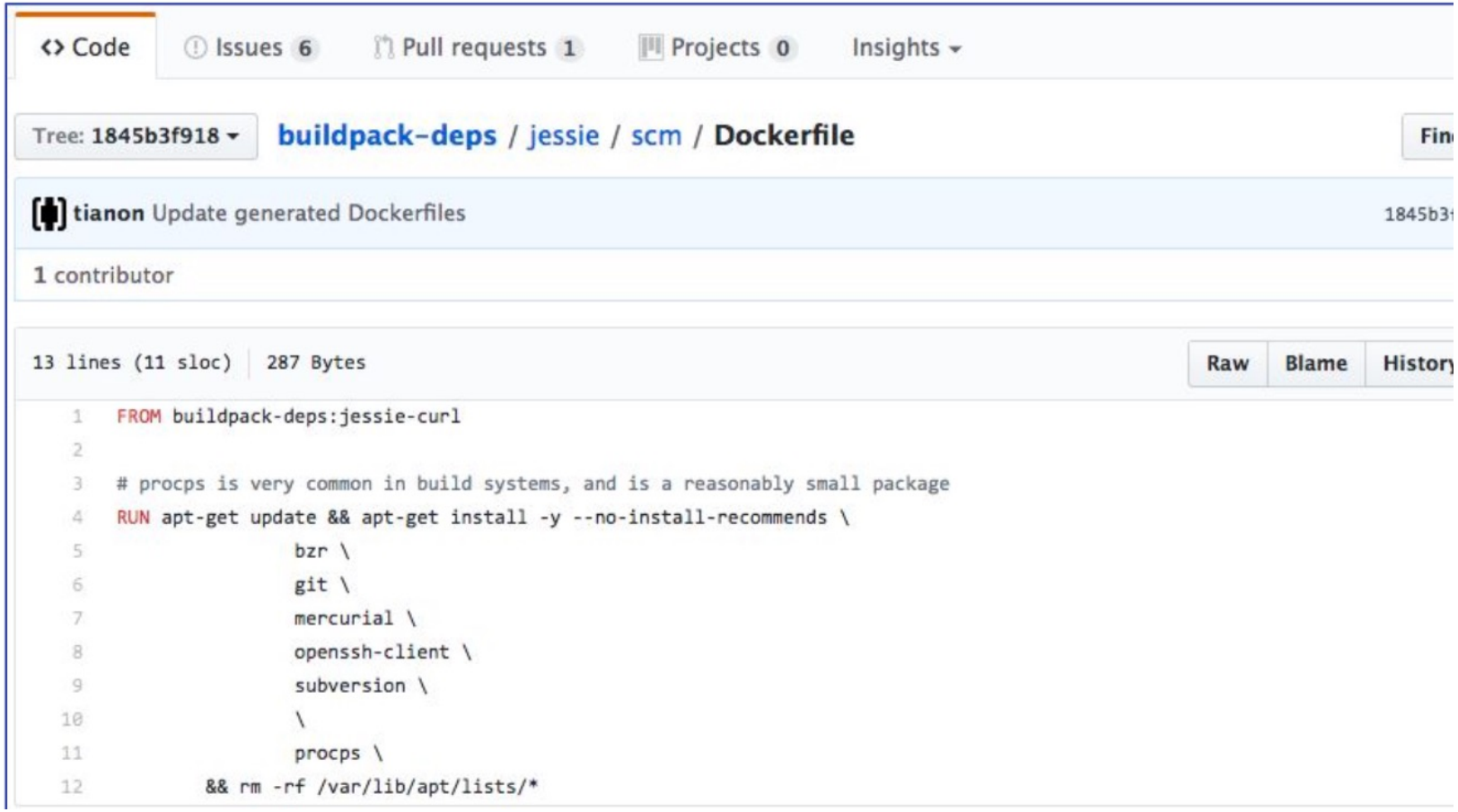
55 lines (53 sloc) | 1.12 KB Raw Blame History

```
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 \
```

Dockerfile examples

- leaf 3



Dockerfile examples

- leaf 4

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

2 contributors

18 lines (15 sloc) | 349 Bytes

Raw Blame History

```
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/*; \
```

Dockerfile examples

- The root

Dockerfile example: compile a code

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
```

Build the image

- 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 c839379f1fbe
Using built-in specs.
COLLECT_GCC=gcc
[...]
Removing intermediate container c839379f1fbe
 ---> 2f0c6f89fdc0
Successfully built 2f0c6f89fdc0
Successfully tagged testcontainer:latest
```

…and run it…

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

…and share it (old way):

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

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

Your first container: tag it!

```
$ docker tag testcontainer gtaff/testcontainer
$ docker push gtaff/testcontainer
The push refers to repository [docker.io/gtaff/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
```

Versioning: hash and tags

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*

Versioning: hash and tags

- A **hash** is the result of applying an hash function
- A **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 a hash.

Versioning: hash and tags

```
$ 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
```

Versioning: hash and tags

- 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.

Versioning: hash and tags

library/gcc - Docker Hub

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)

Where do I save my 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.

Where do I save my Dockerfiles?

- 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**

Running a container for a scientific app.

The problem:

- I have two set of data (in this case, dummy data) to fit with an MCMC.
- Data are random sample from a line

  `y = mx + q.`

  find m, q is the goal;
- 1.txt is an observation, 2.txt is another observation

Running a container for a scientific app.

The material:

- Dockerfile
- 1.txt, 2.txt some data to analize
- Mcmc.py - a Python based emcee script used for fitting the data. The environment variable RUN is used by the container to select the file to analize
- Requirements.txt - a list of package that will be installed using pip.

- Output - Corner plot (1..png, 2.png) that contains the results of the analyses.

Running a container for a scientific app.

HOWTO:

```
$ docker build -t mcmc .
```

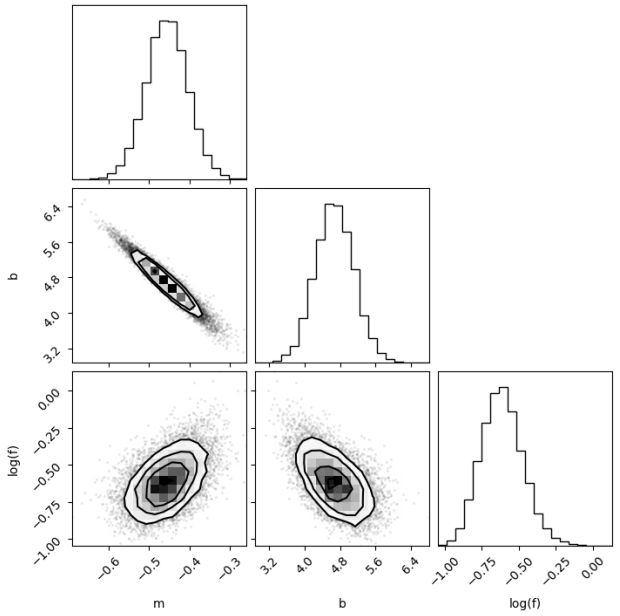
then

```
$ docker run -v $PWD/data:/app --env RUN=1 mcmc
```

or

```
$ docker run -v $PWD/data:/app --env RUN=1 mcmc

$ docker run -v $PWD/data:/app --env RUN=2 mcmc
```



Docker: service example

```
$ docker build  . -t myapp

$ docker run --rm -p 9002:80 myapp

$ docker ps
```

```
# Use an official Python runtime as a parent image
FROM python:latest

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "main.py"]
```

Docker: service example (2)

```
# Use an official Python runtime as a parent image
FROM python:latest

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "main.py"]
```

```
$ docker build  . -t myapp

$ docker run --rm -p 9002:80 -v$PWD:/data myapp

$ docker ps
```

Docker-compose basic commands

```
$ docker compose version

$ docker compose up

$ docker compose down
```

Yaml example

```yaml
version: '3'
services:
  web:
    build: .
    ports:
     - "5000:5000"
  redis:
    image: "redis:alpine"
```

Running a container for a scientific app.

HOWTO:

```
$ docker compose up
Attaching to mcmc-1, mcmc-2
mcmc-2  | Maximum likelihood estimates:
mcmc-2  | m = 0.282
mcmc-2  | b = 5.064
mcmc-2  | f = 0.535
mcmc-1  | Maximum likelihood estimates:
mcmc-1  | m = 0.282
mcmc-1  | b = 5.064
mcmc-1  | f = 0.535
100%|##################################| 5000/5000 [00:02<00:00, 2299.26it/s]
100%|##################################| 5000/5000 [00:02<00:00, 2314.66it/s]
mcmc-2 exited with code 0
mcmc-1 exited with code 0
```


Questions?