

---

# Dockerfile Best Practices

Jens Erat

May 12th, 2015

# Outline

- Dockerfile Best Practices
- Building Images

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License.  
To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/>.

# Dockerfile Best Practices

## Sources

- Dockerfile reference  
`https://docs.docker.com/reference/builder/`
- Dockerfile best practices  
`https://docs.docker.com/articles/dockerfile\_best-practices/`
- Guidelines for Creating Official Repositories  
`https://docs.docker.com/articles/dockerfile\_best-practices/#examples-for-official-repositories`
- Docker newsletter

# Goals

- Isolation
- Maintainability
  - Developers
  - Administrators
- Size

# Building Images

## FROM debian:8

---

Distribution	Size
alpine	5,028,556
centos	215,676,104
debian	114,995,306
fedora	241,316,031
opensuse	256,224,454
oraclelinux	198,964,416
ubuntu	188,268,233
ubuntu-debootstrap	87,019,347

---

## FROM debian:8



**debian**

*Figure 1:Debian Swirl*

- Small image
- Stable
- Commonly used
- Minimal set of necessary components
- “Docker-recommended”
- Lots of official PPAs/repositories for Ubuntu!



## FROM debian:8



*Figure 1:Debian Swirl*

- Small image
- Stable
- Commonly used
- Minimal set of necessary components
- “Docker-recommended”
- Lots of official PPAs/repositories for Ubuntu!

## FROM php

- Base images for most important software stacks
- Mostly Debian-based
- Don't reinvent the wheel

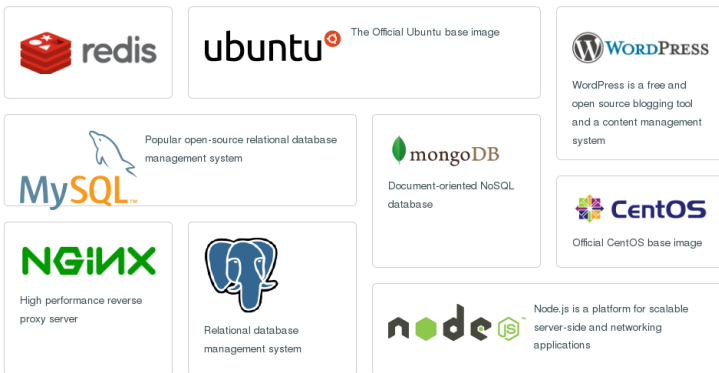


Figure 2: Official base images

## FROM php

- Drop in
  - sources
  - base configuration
  - add documentation (required database links, setup, ...)
  - ready to go!
- Dockerfile directly in your source repository?
  - `.dockerignore` (don't need git history in containers)

# RUN

- Each `RUN` statement adds another layer
  - image grows
  - overhead (container startup/image creation)
- Keep number of `RUN` statements minimal by grouping them
- Clean up after large operations (software installation, ...)

# RUN

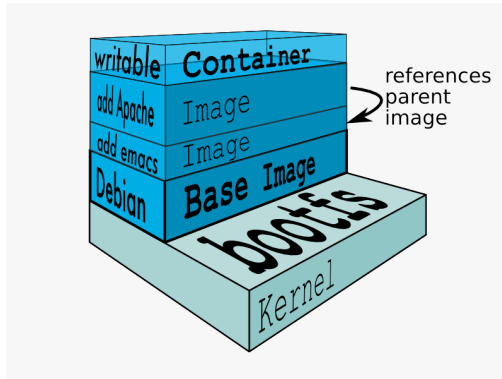


Figure 3: layers of docker container

## RUN apt-get (update | upgrade | install)

### Preparing

- Base images bring `sources.list`, but no packages, thus `RUN apt-get update` before installing packages
- Don't `apt-get upgrade`. instead `docker pull` new image
- Rely on vendor packages if available

## RUN apt-get (update | upgrade | install)

### Install software

- Unattended installation: `apt-get install -y foo`
- Keep things minimal: `--no-install-recommends`
- No useful admin tools: text editors, ping, rsync, ...
  - bloats size & possible attack vector
  - keep image minimal
  - install in container if *really* required

## RUN apt-get (update | upgrade | install)

### Fixed versions

Fix “main product” version to be installed, “cache buster”

```
ENV NGINX_VERSION 1.9.0-1~jessie
RUN apt-get update && \
    apt-get install -y ca-certificates nginx=${NGINX_VERSION}
```

### ~~DEBIAN\_FRONTEND=noninteractive~~

Prevents interactive sessions (database configuration, ...). Now included in base image!



## RUN apt-get (update | upgrade | install)

### Clean up

Remove package lists (outdated anyway). temporary files

```
RUN apt-get clean && rm -rf \  
  /var/lib/apt/lists/* \  
  /tmp/* \  
  /var/tmp/*
```

## apt-get example: nginx

```
RUN apt-key adv --keyserver hkp://pgp.mit.edu:80 --recv-keys
    ↪ 573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62
RUN echo "deb http://nginx.org/packages/mainline/debian/
    ↪ jessie nginx" >> /etc/apt/sources.list

ENV NGINX_VERSION 1.9.0-1~jessie

RUN apt-get update && \
    apt-get install -y ca-certificates nginx=${NGINX_VERSION}
    ↪ && \
    rm -rf /var/lib/apt/lists/*
```

## apt-get example: prosody

```
ENV PROSODY_VERSION 0.9.8-1~jessie2
RUN apt-key adv --keyserver pool.sks-keyservers.net --recv-
    ↪ keys 107D65A0A148C237FDF00AB47393D7E674D9DBB5 && \
    echo deb http://packages.prosody.im/debian jessie main
    ↪ >>/etc/apt/sources.list && \
apt-get update && \
apt-get install -y --no-install-recommends \
    lua-dbi-mysql \
    lua-dbi-postgresql \
    lua-dbi-sqlite3 \
    lua-event \
    lua-sec \
    lua-zlib \
    prosody=${PROSODY_VERSION} && \
apt-get clean && rm -rf \
    /var/lib/apt/lists/* \
    /tmp/* \
    /var/tmp/*
```

## Installing from sources, binaries

- Verify signatures/ hashes
- Reference specific versions, not `latest.tgz` (cache buster!)
- Clean up after installing/compiling
  - Uninstall build chain when finished or use language base image

## Build example: redis

<https://github.com/docker-library/redis>

### Fetch and verify

```
ENV REDIS_VERSION 3.0.0
ENV REDIS_DOWNLOAD_URL http://download.redis.io/releases/
  ↪ redis-3.0.0.tar.gz
ENV REDIS_DOWNLOAD_SHA1
  ↪ c75fd32900187a7c9f9d07c412ea3b3315691c65

RUN buildDeps='gcc libc6-dev make'; \
  set -x \
  && apt-get update && apt-get install -y $buildDeps --no-
    ↪ install-recommends \
  && rm -rf /var/lib/apt/lists/* \
  && mkdir -p /usr/src/redis \
  && curl -sSL "$REDIS_DOWNLOAD_URL" -o redis.tar.gz \
  && echo "$REDIS_DOWNLOAD_SHA1 *redis.tar.gz" | sha1sum -c
    ↪ - \
```

## Build example: redis

### Build and install

```
&& tar -xzf redis.tar.gz -C /usr/src/redis --strip-  
    ↪ components=1 \  
&& make -C /usr/src/redis \  
&& make -C /usr/src/redis install \  

```

### Cleanup

```
&& rm redis.tar.gz \  
&& rm -r /usr/src/redis \  
&& apt-get purge -y --auto-remove $buildDeps
```

## ADD vs COPY

- COPY just copies
- ADD can perform some fetch- and unarchive-magic
  - don't use unless you definitely need it (untar)
  - use curl for remote files, ADDED files are in their own layer (cannot be deleted)
- watch for trailing / for directories

## ENV, VOLUME, COPY

- pull together lines (reduce layers)
- don't:

```
ENV foo bar
ENV BATZ 42
EXPOSE 80
EXPOSE 443
EXPOSE 8080
```

- do:

```
ENV foo=bar batz=42
EXPOSE 80 443 8080
```



## ENTRYPOINT vs CMD

- ENTRYPOINT is the binary executed
  - default `/bin/sh -c`
- CMD is passed as argument
  - default empty
  - overridden at container startup, eg.

```
docker run -ti debian bash
```

- Docker runs `$ENTRYPOINT $CMD` at startup

## Startup scripts

- Do you really need one?
- write script that starts daemon
- terminate execution, if any command fails

```
set -eu
```

- exec into main process (keep PID, receive signals)
- Docker recommends `gosu` for switching users
- Really need multiple processes? Use `supervisord`.
  - Consider again if you really need it. Twice.
  - Watch for dumping logfiles to container stdout!
- Don't apply too much magic. KISS!

## Startup scripts

- Do you really need one?
- write script that starts daemon
- terminate execution, if any command fails

```
set -eu
```

- `exec` into main process (keep PID, receive signals)
- Docker recommends `gosu` for switching users
- Really need multiple processes? Use `supervisord`.
  - Consider again if you really need it. Twice.
  - Watch for dumping logfiles to container stdout!
- Don't apply too much magic. KISS!

## Startup scripts

- Do you really need one?
- write script that starts daemon
- terminate execution, if any command fails

```
set -eu
```

- `exec` into main process (keep PID, receive signals)
- Docker recommends `gosu` for switching users
- Really need multiple processes? Use `supervisord`.
  - Consider again if you really need it. Twice.
  - Watch for dumping logfiles to container stdout!
- Don't apply too much magic. KISS!

## Startup scripts

- Do you really need one?
- write script that starts daemon
- terminate execution, if any command fails

```
set -eu
```

- `exec` into main process (keep PID, receive signals)
- Docker recommends `gosu` for switching users
- Really need multiple processes? Use `supervisord`.
  - Consider again if you really need it. Twice.
  - Watch for dumping logfiles to container stdout!
- Don't apply too much magic. KISS!

## Persisting data

- `VOLUMES` get persistent directories mounted onto
  - empty at container creation
  - can be overridden by `--volume` and `--volumes-from`
- don't install database systems in container
  - all relevant DBMS available as images
  - link against them
  - including memcached et al.
  - lets administrator decide where to put it

## Persisting data

- build images to be ephemeral
  - no state in containers
  - no data in containers
- 
- throw away container, create new one
  - document for administrator what to persist!

## Logs

- no syslogd, no logrotate inside containers
- Docker expectations: log to stdout/stderr
- Docker daemon takes care of output
  - have a look at debug-flags
- single process, no need for several log files per container
  
- Docker 1.6 can log to host syslogd



## Logs

What if application strictly writes to a file?<sup>1</sup>

- `tail -F` monitors for file to be created
- `tail --pid` makes tail terminate if process is terminated

```
#!/usr/bin/env bash
set -eu

rm -rf /var/log/my-application.log
tail --pid $$ -F /var/log/my-application.log &

exec /path/to/my-application
```

---

<sup>1</sup>As proposed in <http://serverfault.com/a/599209/98727>

## Users

- Don't run daemons as root<sup>2</sup>
- Fix user ID (reduces permission issues)

```
adduser --system --home /srv --disabled-password --  
↳ disabled-login --uid 1984 basex
```

- Set user in Dockerfile using `USER basex`
- Consider unSUIDing binaries to reduce possible attack vector

```
RUN for i in `find / -perm +6000 -type f`; do chmod a-s  
↳ $i; done
```

---

<sup>2</sup>unless you can argue why

## Debugging help

- Don't clean up during development (makes installing `vim` easier)
- `docker exec -ti [container] /bin/bash`
- Start with individual `RUN` lines, merge later (keep expensive download-operations in cache)
- Check permissions. Again.
- Ubuntu-based systems: stuck in app-armor?
- Check host syslog for denied operations inside containers (missing privileges?)
- UDP ports must be exposed separately

## Docker registry

- Let Docker build, do not upload images
- Hooks to base image
- Directly use official base images

## Documentation

- Don't just dump a Dockerfile!
- Describe what's...
  1. inside the image
  2. needed for setup, especially
    - database container links
    - what folders to persist
    - what's listening on which port
    - configuration hickups (logfiles, non-daemonized execution, ...)
    - reverse proxy configuration hints
    - cronjobs (`docker exec`)
  3. required action during upgrades (database maintenance?)
- Explain general hickups
- For public images/Dockerfiles: License of product and Dockerfile
- YAML file for `docker-compose`