Docker Best Practice

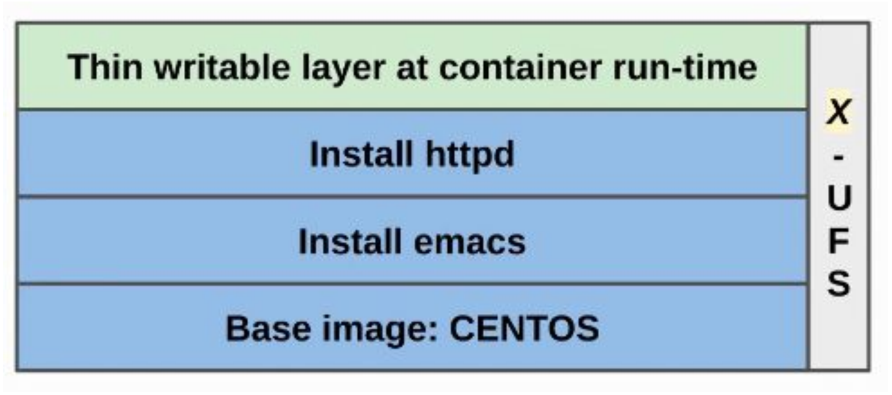
Docker basic level II

Why bother?

- Writing a good Dockerfile will result in docker container images, which are:
    - Smaller
    - Cleaner
    - Secure
- To archive these, we write Dockerfiles which follow the best practice, i.e:
    - They are cleanly written - thus easy to read / debug
    - They are organized in logical sections
    - They have instructions written in logical order
    - … and more (covered next)

Re-cap: Docker container layers

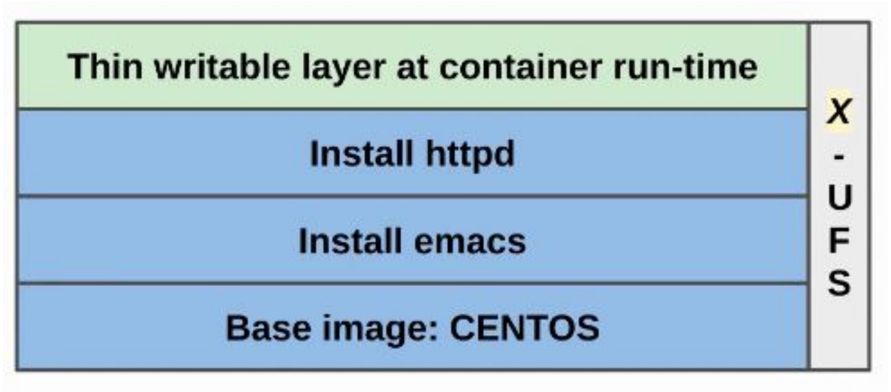
```
$ cat Dockerfile

FROM centos:7
RUN yum -y install emacs
RUN yum -y instal httpd
```



```
$ docker build -t local/centos-apache .
$ docker image ls

local/centos-apache     latest     c8b3036d9c8a     5  seconds  ago          631MB
centos                  latest     5e2539a48d9s     5 minutes  ago           300MB
```

Re-cap: Docker container layers ...

```
$ docker image history local/centos-apache

IMAGE           CREATED           CREATED BY                                          SIZE
c8b3036d9c8a    9 minutes ago     /bin/sh -c yum -y install httpd               631MB
22b134c9419c    19 minutes ago    /bin/sh -c yum -y install emacs                       350MB
5e2539a48d9s    1 months ago      /bin/sh -c #(noh)  CMD [“/bin/bash”]                  300MB
<missing>       1 months ago      /bin/sh -c #(noh)  LABEL org.label-scheme..   0B
<missing>       1 months ago      /bin/sh -c #(noh) ADD  file:14a318104952b.. 203B
```



AUFS - Advanced (multi-layered) Unification FileSystem

Problem:

- You have 3 directories, dir1, dir2 and mountdir.
- You want to somehow mount dir1 and dir2 in mountdir, so you can see (combined) contents of dir1 and dir2 in mountdir.

Older Solutions:

- Use unionfs or aufs

Latest Solution:

- AUFS is not used anymore
- Use OverlayFS

OverlayFS - How it works?

- OverlayFS layers two directories on a single Linux host and presents them as a single directory.
- These directories are called **layers** and the unification process is referred to as a **union mount.**
- OverlayFS refers to the lower directory as **lowerdir** and the upper directory a **upperdir**.
- The unified view is exposed though its own directory called merged.
- The overlay2 driver natively supports up to 128 lower OverlayFS layers, and provides better performance.

OverlayFS - Keep packages to the minimum

- When creating container images, start with smallest possible base image based on your requirements
- Install only enough packages which are needed to run your application. I.e.
  - Don’t install an entire operating system inside it
  - Don’t download the entire internet inside it
- For debugging purposes, attach “tools” containers to your application containers when necessary

OverlayFS - Use unix’s philosophy - De-couple applications

- A container should do only one-thin, and do it well
  - For example: Don’t combine Apache and MySQL in one container: they should be two separate containers
- Individual containers serving/providing individual services are easier to: manage, scale,troubleshoot, update, upgrade, etc.

Best Practice: Minimize number of layers in an image

- Remember, only the instructions RUN, COPY, ADD create layers
- Consolidate multiple RUN commands into one, or have lesser RUN commands
- Organize files in a better way before COPY-ing into the container image.
- Use COPY instead of ADD, as COPY is more transparent, and has predictable behavior
- Don’t use ADD to fetch packages from remote URLs. Use curl or wget instead.

Best Practice: Minimize number of layers in an image

```
$ cat Dockerfile.long
FROM debian
RUN    apt-get update
RUN    apt-get install -y vim-common
RUN    apt-get install -y curl
RUN    apt-get install -y netcat
RUN    apt-get install -y tcpdump
RUN    apt-get install -y dnsutils
RUN    apt-get install -y net-tools
```

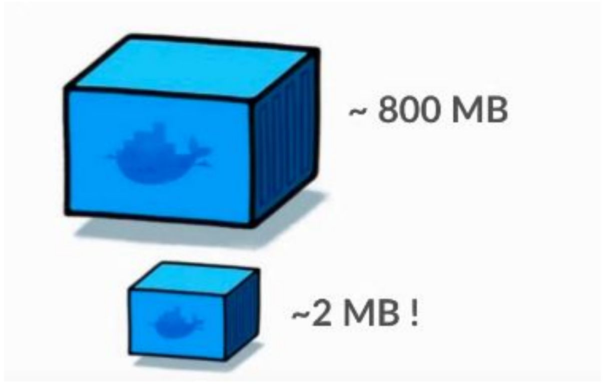
```
$ cat Dockerfile.compact
FROM debian
RUN   apt-get update \
  &&  apt-get -y install vim-common \
          curl netcat tcpdump dnsutils \
      net-tools
```

```
$ docker build -f Dockerfile.long -t local/simple:long .  --no-cache
$ docker build -f Dockerfile.compact -t local/simple:compact  . --no-cache

$ docker image ls
REPOSITORY        TAG          IMAGE ID               CREATED             SIZE
local/simple      compact      372v7c6789sb8     1 minutes ago       229MB
local/simple      simple       876252g2u752b         2 minutes ago        239MB
```

Best practice: Use multi-stage builds

- **Only copy the artifacts you need into the final image.**

```
$ cat Dockerfile
FROM       golang:1.14 as builder
WORKDIR /tmp
COPY       myapp.go   /tmp
RUN        go build myapp.go

FROM       scratch as myapp
COPY       --from=builder  /tmp/myapp  /myapp
CMD        ["/myapp"]
```



```
$ docker build -t local/myapp .  --no-cache

$ docker image ls | grep golang
golang            1.14            12a265b042a           9 days ago              809MB

$ docker image ls | grep myapp
local/myapp       latest       fbd7826bs09c          20 seconds ago         2.07MB
```

Best practice: sort multi-line arguments

- For better readability, prevent double entries and debugging sort all command arguments alphabetically.

```
$ cat Dockerfile
FROM        centos:7
RUN         yum -y install \
            vim-common \
            curl  \
            socat \
            traceroute \
            net-tools \
            bind-utils \
            tcpdump
```

```
$ cat Dockerfile
FROM        centos:7
RUN         yum -y install \
            bind-utils \
            curl \
            net-tools \
            socat \
            tcpdump
            traceroute \
            vim-common
```

Best practice: “Debian” & derivatives - Avoid RUN apt-get upgrade and dist-upgrade

- Many of these “essential” packages from the parent images cannot upgrade inside an unprivileged container.
- If a package contained in the parent image is out-of-date, contact its maintainers.
- Use apt-get install -y <package> to update any particular package automatically.

```
$ cat Dockerfile
FROM        debian
RUN         apt-get update \
      &&    apt-get upgrade \
      &&   apt-get dist-upgrade \
      &&   apt-get -y install curl \
      &&   dns-utils \
            net-tools \
            netcat \
```

```
$ cat Dockerfile
FROM        debian
RUN         apt-get update \
      &&    apt-get -y install curl \
            dns-utils \
            netcat \
            net-tools \
```

Best practice: “debian & derivatives - Always have “apt-get update” and “apt-get install” in one RUN command

- Using apt-get update in a separate RUN statement causes caching issues
- If apt-get update is a separate command, it pulls in apt package index, and docker stores it as a layer.
- For every package in need of installation, apt-get consults this package index. It may be “now”, or “later in time”.
- When “later in time” comes, the package index - which is cached layer - is old, which can cause an older version of the package to be installed!

```
$ cat Dockerfile
FROM      debian
RUN       apt-get update
RUN       apt-get install -y curl
```

```
$ cat Dockerfile
RUN       apt-get update
RUN       apt-get install -y curl nginx
```

```
$ cat Dockerfile
RUN   apt-get update  \
   &&  apt-get install -y curl nginx
```

Best practice : Use CMD, Entrypoint correctly

- CMD should almost always be used in the form of CMD [“executable”,”param1”,”param2”...]
  - E.g. if the image is for a service, such as Apache, CMD should be **[“apache2”,”-DFOREGROUND”]**
- Entrypoint should only be used to “prepare” the container, not run some command as default.
  - ...unless that is the sole purpose of that container, such as a container used as an executable.

```
$ cat Dockerfile
FROM httpd:2.4
…
CMD [“apache2”, “-DFOREGROUND”]
```

```
$ cat Dockerfile
…
COPY entrypoint.sh /
ENTRYPOINT [“/entrypoint.sh”]
CMD [“apache2”, ”-DFOREGROUND”]
```

```
$ cat Dockerfile
#!/bin/bash
Echo “This is a web  host : $(hostname)” >> /var/www/html/index.html
Chmod +r
```

Best practice: be careful in using pipe(|) in RUN commands

- Docker executes these commands the `/bin/sh -c` interpreter, which only evaluates the exit code of the last operation in the pipe to determine success.
  - There is no guarantee if the RUN command shown below will always work as expected.

```
$ cat Dockerfile
FROM busybox
. . .
RUN wget -O  - https://somerandom.site  | wc -l > /number
. . .
```

- If you want the command to fail due to an error at any stage in the pipe, prepend ‘set -o pipefail &&’ to the RUN command.

```
$ cat  docker-entrypoint.sh
FROM busybox
.   .   .
RUN  set -o  pipefail  && wget  -O - https://somerandom.site  | w c -l  |  /number
```

Best practice: Avoid running as root

- Running a container as a root does not automatically mean that there is a security problem.
- However, if a service can run as a normal user, then it is best to set a non-root user in the USER directive to ensure that the container will not run as root.
- Avoiding installing or using sudo, as it has unpredictable TTY and signal-forwarding behaviour that can cause problems.
- If you absolutely need functionality similar to sudo, such as initializing the daemon as root but running it as non-root, consider using “gosu”
- To reduce layers and complexity, avoid switching USER back and forth frequently in your Dockerfile.

Best practice: use WORKDIR properly

- For clarity and reliability, you should always use absolute path for your WORDIR
- You should use WORKDIR instead of proliferating instructions like: `RUN cd … &&  do-something`

```
$ cat Dockerfile
FROM   httpd:2.4
. . .
COPY web-content.tar /var/www/html
RUN   cd /var/www.html
RUN   untar  web-content.tar
RUN   cd ../../..
. . .
```

```
$ cat Dockerfile
FROM  httpd:2.4
. . .
WORKDIR /var/www/html
COPY        web-content.tar /var/www/html
RUN         untar web-content.tar
. . .
```