

Security: attack surface: access by containers to root!

- docker daemons get access to root

Contents of containers- malicious code

- just as shipping containers can conceal harmful cargo

Configuring Docker containers to talk over the network

One Ubuntu machine and a CentOS machine:

Ubuntu:

service docker stop (if already running)

docker -H 192.168.56.50:2375 -d &

(puts it daemon mode and listens on the network port instead of local socket)

netstat -tlnp (to see it listed, it is)

Now, if you run docker info, "can't connect to the docket daemon is -docker -d running on this host?"

On the CentOS machine set

export DOCKER_HOST="tcp://192.168.56.50:2375"

Now immediately compare version numbers etc., to see that when you run docker info, docker -v docker version it will be returning the values from the daemon running on the other machine (Ubuntu) not the localhost (CentOS)

Issue "export DOCKER_HOST=" to turn it off, and stop the process on the machine sharing that connection.

Remember that doing something like this is leaving the docker "root" process wide open on the machine allowing the connection

(see "containers and the attack surface")

If you wanted it to be listening on BOTH the network port and the unix socket:

docker -H 192.168.56.50:2375 -H unix:///var/run/docker.sock -d &

----- Container shares SOME files with machine's kernel, not others, PLUS container persistence of data

So fire up a CentOS container on an Ubuntu host

docker run -it centos /bin/bash

Once it is up and running the prompt changes to that of the container's:

root@ff74543a54e7 /#

Run ps -elf - only shows container's stuff (/bin/bash and ps -elf)

cat /etc/hosts - shows 172.16 (reserved) ip for the container, the loopback and the normal IPv6 stuff

ip a -- same thing

uname -a --This shows the Ubuntu kernel- since they are sharing some of the same stuff.

But the FILE cat /etc/redhat-release brings up the CentOS release number and stuff
apt-get upgrade brings 'command not found' but yum check-update works

vim isn't found, but yum install vim puts it on the container
 Make a file put in contents and save in vim wq.
 Exit container,
 docker ps -a to get the name (copy and paste it)
 Check the file system now that it is shut down "ls -al /var/lib/docker/aufs/diff/
 ff74543a54e7 <<tab completion here>>
 (The first part of the UUID is the container number and the right folder)
 Find the file in the /tmp folder in there where it was saved, cat'ing it works, fire up the
 container again and it still works as it was.
 docker start ff74543a54e7
 docker attach ff74543a54e7

---->Analogy time

Containers - shipping containers
 Docker images - shipping manifests
 Docker engine - shipping yard
 Docker engine is really daemon or runtime - but like a shipping yard has components
 like networking, kernel, namespace(s), cgroups, other components; moving containers
 from one to another, but the shipping yards all look the same basically (run on different
 architectures) standardized components, capacities, etc

docker run -it fedora /bin/bash <---will pull image currently tagged as fedora:latest
 docker pull -a fedora <---will pull all of the different versions of Fedora rather than just
 fedora:latest
 docker images fedora <---lists 5 this time, 5 different names too, but sharing only 3
 different image IDs
 by the names- it is explained there are two pair with the same IDs ("20" aka
 heisenbug) "21" the same as "latest" since it IS the latest)
 Always stored locally in /var/lib/docker/<storage_driver> (in var/lib/docker/aufs for
 Ubuntu)

Ctrl + p + q = to exit the container without killing it's instance
 docker ps <---for all running containers
 docker ps -a <---for all containers on the host

Pull from repos (MongoDB, Canonical's Ubuntu, etc.), which are inside of registries
 (Docker Hub):

```
docker-machine ls
NAME          ACTIVE    DRIVER      STATE     URL
SWARM
default      *         virtualbox  Running   tcp://
192.168.99.100:2376
docker-machine kill default <--- kills the Kitematic process
```

Container layers/ images:

Layers aka Images

One image (container) comprised of 3 layered images:

Image3 - updates/patches - built on:

Image2 - nginx - built on:

Image1 - ubuntu - a rootfs - a base image

-- It all comprises one "image" that you base new containers on

Need to change something or tweak a program (image)? Just change the layer (Enter Puppet or Chef)

This single image can be shared amongs multiple containers on the host

Each layer gets it's own UUID, higher layer hides lower layers

Layering accomplished using union mounts, all share one mountpoint.

Conflicts: different copies of /etc/resolv.conf in layer 1 and in 3... 3 (being the top layer) wins out

(the top layer's contents always gain precedence)

Images are read-only- "invisible" RW layer added on top of container while running.

-- (This is where changes are made to commit later to bottom layers - copy-on-write union mount)

Similar "invisible" bootfs underlying layer for boot containers -exists only when container is started and then is disposed of. (containers are started, not booted, since it is the host OS that is booted (in the case of Mac and Windows, the headlessVM. This temporary 'layer0' is used to get everything going for the started instance)

This view comprises a read-only entity which gets a r/w layer on top of it (copy-on-write)

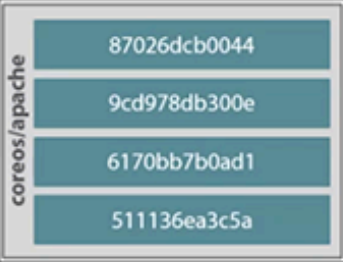
Before the container is started/run, it actually has a base bootfs layer underneath everything

AUFS implements a union mount for Linux file systems (originally stood for AnotherUnionFS)



When this is "flattened" into a single view the higher layers win out so if something is in 3 but also in 0, top layer 3 wins out. Union mounts puts multiple filesystems on top of eachother (one could even be root, another home). Here is how we see the layers in a freshly pulled image of coreos/apache. "docker images --tree" shows the hierarchy:

```
coreos/etcd      latest      649a8540db5e    5 weeks ago
coreos/etcd      v0.4.6     649a8540db5e    5 weeks ago
root@ubuntu1404-02:~# docker pull coreos/apache
Pulling repository coreos/apache
87026dcb0044: Download complete
511136ea3c5a: Download complete
6170bb7b0ad1: Download complete
9cd978db300e: Download complete
Status: Downloaded newer image for coreos/apache:latest
root@ubuntu1404-02:~# docker images --tree
Warning: '--tree' is deprecated, it will be removed soon. See usage.
├── a2d6848456de Virtual Size: 0 B
│   └── 995c2b064ad7 Virtual Size: 12.75 MB
│       ├── d4e20e5984a9 Virtual Size: 20.15 MB
│       └── bfd1f10d9c74 Virtual Size: 20.15 MB
│           └── 649a8540db5e Virtual Size: 20.15 MB Tags: coreos/etcd:latest, coreos/etcd:v0.4.6
├── 511136ea3c5a Virtual Size: 0 B
│   ├── 01bf15a18638 Virtual Size: 192.5 MB
│   ├── 30541f8f3062 Virtual Size: 192.7 MB
│   ├── e1cdf371fbde Virtual Size: 192.7 MB
│   └── 9bd07e480c5b Virtual Size: 192.7 MB Tags: ubuntu:latest
├── 00a0c78eeb6d Virtual Size: 0 B
│   ├── bfe0bb6667e4 Virtual Size: 250.2 MB Tags: fedora:21, fedora:latest
│   ├── 782cf93a8f16 Virtual Size: 0 B
│   ├── 2e613b1df961 Virtual Size: 374.1 MB Tags: fedora:heisenbug, fedora:20
│   ├── 9da4e4dbc6be Virtual Size: 379.5 MB Tags: fedora:rawhide
│   ├── 5b12ef8fd570 Virtual Size: 0 B
│   ├── 34943839435d Virtual Size: 224 MB Tags: centos:latest
│   ├── 6170bb7b0ad1 Virtual Size: 0 B
│   ├── 9cd978db300e Virtual Size: 204.4 MB
│   └── 87026dcb0044 Virtual Size: 294.5 MB Tags: coreos/apache:latest
root@ubuntu1404-02:~#
```



Since our image's metadata are stored in `/var/lib/docker/aufs/layers` (in the aufs directory on an Ubuntu box) we can go and interrogate those files too, layer-by-layer. Each one simply contains the layers underneath. Again, the UUID begins with the ImageID:

```
root@ubuntu1404-02:~# ls -l /var/lib/docker/aufs/layers/
total 96
-rw-r--r-- 1 root root 65 Dec 19 10:57 00a0c78eeb6d81442efcd1d7c02e8b141745e3a06f1ee3458e1bae628e0067d3
-rw-r--r-- 1 root root 195 Dec 26 18:19 87026dcb00443eb7f1725b1c9f4fb8210027a19364103854a1e5f606b95019ff
-rw-r--r-- 1 root root 195 Dec 15 09:34 ff7f984c22e7b75e63ee68addc15886d40f19a69656a10e2592c2a4e52881704
root@ubuntu1404-02:~#
root@ubuntu1404-02:~# cat /var/lib/docker/aufs/layers/87026dcb00443eb7f1725b1c9f4fb8210027a19364103854a1e5f606b95019ff
9cd978db300e27386baa9dd791bf6dc818f13e52235b26e95703361ec3c94dc6
6170bb7b0ad1003a827e4dc5253ba49f6719599eac485db51eaafd507c13c311
511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158
root@ubuntu1404-02:~# cat /var/lib/docker/aufs/layers/9cd978db300e27386baa9dd791bf6dc818f13e52235b26e95703361ec3c94dc6
6170bb7b0ad1003a827e4dc5253ba49f6719599eac485db51eaafd507c13c311
511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158
root@ubuntu1404-02:~# cat /var/lib/docker/aufs/layers/6170bb7b0ad1003a827e4dc5253ba49f6719599eac485db51eaafd507c13c311
511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158
root@ubuntu1404-02:~# cat /var/lib/docker/aufs/layers/511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158
22c158
root@ubuntu1404-02:~#
```

Everything in this `/aufs/layers` directory is just metadata. The actual contents are located in `/var/lib/docker/aufs/diff/`
`ls -al /var/lib/docker/aufs/diff/87dbc3869385....`
`/dev`
`/etc`
`/lib`

/tmp
/usr
/var

These are the items making up that layer... another had all of the core files needed for base Ubuntu

----- Copying containers around

`docker run ubuntu /bin/bash -c "echo 'my short process output' > /tmp/test-output"`

- Makes new container, based on local ubuntu image already present, runs the bash command.
- In detached mode, without "-it", the process runs, outputs it's stuff, and the process ends so the container exits
- It did make a change saving the contents to /tmp inside (we changed *it's state*)
- The container lists with "docker ps -a" so we can grab the container ID (37f637d73a).

`docker commit 37f637d73a new_containerName`

- This spits out a new container, that we can see when running "docker images"
- Running "docker image -tree" shows predicable hierarchy, and "docker history new_containerName" will even show some command history back through all the layers.

`docker save -o /tmp/new_containerName.tar new_containerName`

- This exports the image. You dont have to specify tar since this is default

```
root@ubuntu1404-02:~# docker history fridge
IMAGE                CREATED              CREATED BY          SIZE
f405b65fb1f4        1 minute ago       /bin/bash -c echo 'cool content' > /tmp/cool-    13 B
9bd07e480c5b        3 weeks ago       /bin/sh -c #(nop) CMD [/bin/bash]              0 B
e1cdf371fbde        3 weeks ago       /bin/sh -c sed -i 's/^#\s*\((deb."universe\)$/    1.895 kB
30541f8f3062        3 weeks ago       /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic   194.8 kB
01bf15a18638        3 weeks ago       /bin/sh -c #(nop) ADD file:0491afac3601ebc9a8    192.5 MB
511136ea3c5a        18 months ago
root@ubuntu1404-02:~# docker save -o /tmp/fridge.tar fridge
```

Copy this image to another machine. Here is what the contents of the tar archive look like. This output is truncated, but for each layer there is a directory named after it's UUID, containing the items VERSION, json (for the layer's metadata), and layer.tar (the actual layer data). There is also a file "repositories".

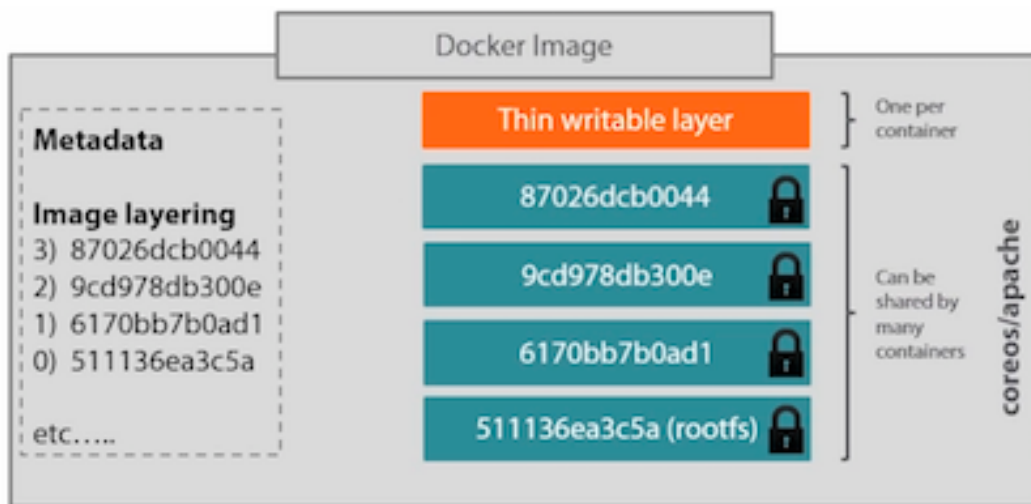
```
[root@localhost ~]# tar -tf /tmp/fridge.tar
01bf15a18638145eb44f0363ece1845b0f0dcf24adc03700ca519ea3d5fe6b8e/
01bf15a18638145eb44f0363ece1845b0f0dcf24adc03700ca519ea3d5fe6b8e/VERSION
01bf15a18638145eb44f0363ece1845b0f0dcf24adc03700ca519ea3d5fe6b8e/json
01bf15a18638145eb44f0363ece1845b0f0dcf24adc03700ca519ea3d5fe6b8e/layer.tar
f405b65fb1f4dacee1dc68674706c2eb91242cc9821b1b6802a009084a1524b7/
f405b65fb1f4dacee1dc68674706c2eb91242cc9821b1b6802a009084a1524b7/VERSION
f405b65fb1f4dacee1dc68674706c2eb91242cc9821b1b6802a009084a1524b7/json
f405b65fb1f4dacee1dc68674706c2eb91242cc9821b1b6802a009084a1524b7/layer.tar
repositories
```

Import into Docker with "docker load -i /tmp/new_containerName.tar"

Once loaded, it will show in the "docker images" listing.

Images are build-time constructs and containers are runtime constructs
Images are used to launch containers, so containers are runtime instances of images

Thin writable layer is where state is stored- a "state-layer"



- Usually in Linux, the rootfs is read-only at boot time, and then remounted as read-write afterwards.
- In a container the rootfs is on the bottom layer and is ALWAYS read-only
- Since the top-most layer info has precedence over any differing information, and the lower layers are locked, this precedence in union-mounting is what give the IMPRESSION of making changes to the underlying OS inside the rootfs layer.
- The new info goes into the read-write top state-layer overriding pre-existing stuff in the bottom ones.
- The bottom layers are essentially sandboxed runtime execution environments- but, the data in the top layer overlay can make it appear or behave otherwise

Need to change something or tweak a program (image)? Just change the layer (Enter Puppet or Chef)

One process thread per container "term"

(Usually) there is one process per container, and when it exits, the container stops

Example: (run -d is run detached)

`docker run -d ubuntu /bin/bash -c "ping 8.8.8.8 -c 30"`

`docker ps` <----lists/shows it running

`docker top 455dd3425263` (get number from ps) <---- shows processes inside container

`docker ps` <---- drops from list of running containers when done

`docker ps -a` <----but this lists that it ran and when it stopped

`docker run --cpu-shares` (how many CPU "shares" it gets - 1024 is 100%, 256 is 25%)

`docker run memory=1g`

`stop start restart`

`docker stop (containerID)` - sends SIGTERM to container's pid1 (NOT init but the main process of the container)

`docker kill -s SIGNAL`

`docker restart ID`

`docker history`

`docker ps -l` (last run)

At the cmd line of a docker container, you can switch to host CLI (detach) with `ctrl-p-q`

`docker rm containerID` -- can't remove running containers, but not unsurprisingly -f can.

`docker logs containerID -f` (or --follow) will show container's log and the -f option is like doing a tail on the log

Normally, it is one process per container, (meaning one concern, lean and simple functionality) but you can run multiple processes in one container

See phusion images in repo for multiprocess containers

phusion/baseimage has a process `/usr/bin/python3 -u /sbin/my_init -- bash -l`

Remember that "docker ps" PIDs do not match PIDs inside of the running container

```
root@ubuntu1404-02:~# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
8b2da74bd40a   phusion/baseimage:0.9.15           "/sbin/my_init -- ba    41 seconds ago Up 40 seconds
TS            NAMES
8b2da74bd40a   grave_turing

root@ubuntu1404-02:~#
root@ubuntu1404-02:~# docker top 8b2da74bd40a
UID            TIME          PID           PPID          C              STIME         TTY
root           00:00:00      2033          768           0              21:50         pts/1
root           00:00:00      2146          2033          0              21:50         pts/1
root           00:00:00      2147          2033          0              21:50         pts/1
root           00:00:00      2148          2146          0              21:50         ?
root           00:00:00      2149          2146          0              21:50         ?
root           00:00:00      2150          2146          0              21:50         ?
root           00:00:00      2151          2150          0              21:50         ?
root           00:00:00      2152          2149          0              21:50         ?
root           00:00:00      2153          2148          0              21:50         ?
root           00:00:00      2148          2148          0              21:50         ?
root@ubuntu1404-02:~#
```

docker inspect 455dd3425263 <----detailed info on running container or image ID including IP address and state

config.json and hostconfig.json hold many of the values you will see in docker inspect:

```
root@ubuntu1404-02:~# ls -l /var/lib/docker/containers/8b2da74bd40aeec791d9e96eeefb78640c38483f3956e699ef025c027be45b79c/
total 24
-rw-r----- 1 root root 3802 Dec 30 21:52 8b2da74bd40aeec791d9e96eeefb78640c38483f3956e699ef025c027be45b79c-json.lo
8
-rw-r--r-- 1 root root 1637 Dec 30 21:50 config.json
-rw-r--r-- 1 root root 334 Dec 30 21:50 hostconfig.json
-rw-r--r-- 1 root root 13 Dec 30 21:50 hostname
-rw-r--r-- 1 root root 174 Dec 30 21:50 hosts
-rw-r--r-- 1 root root 181 Dec 30 21:50 resolv.conf
root@ubuntu1404-02:~# cat /var/lib/docker/containers/8b2da74bd40aeec791d9e96eeefb78640c38483f3956e699ef025c027be45b79c/config.json
{"State":{"Running":true,"Paused":false,"Restarting":false,"OOMKilled":false,"Pid":2033,"ExitCode":0,"Error":"","StartedAt":"2014-12-30T21:50:31.607395916Z","FinishedAt":"0001-01-01T00:00:00Z"},"ID":"8b2da74bd40aeec791d9e96eeefb78640c38483f3956e699ef025c027be45b79c","Created":"2014-12-30T21:50:31.315452835Z","Path":"/sbin/my_init","Args":["--","bash","-l"],"Config":{"Hostname":"8b2da74bd40a","Domainname":"","User":"","Memory":0,"MemorySwap":0,"CpuShares":0,"Cpuset":"","AttachStdin":true,"AttachStdout":true,"AttachStderr":true,"PortSpecs":null,"ExposedPorts":null,"Tty":true,"OpenStdin":true,"StdinOnce":true,"Env":["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin","HOME=/root"],"Cmd":["/sbin/my_init","--","bash","-l"],"Image":"phusion/baseimage","Volumes":null,"WorkingDir":"","Entrypoint":null,"NetworkDisabled":false,"MacAddress":"","OnBuild":null,"Image":"cf39b476aeec4d2bd097945a14a147dc52e16bd88511ed931357a5cd6f6590de","NetworkSettings":{"IPAddress":"172.17.0.5","IPPrefixLen":16,"MacAddress":"02:42:ac:11:00:05","Gateway":"172.17.42.1","Bridge":"docker0","PortMapping":null,"Ports":{}},"ResolvConfPath":"/var/lib/docker/containers/8b2da74bd40aeec791d9e96eeefb78640c38483f3956e699ef025c027be45b79c/resolv.conf","HostnamePath":"/var/lib/docker/containers/8b2da74bd40aeec791d9e96eeefb78640c38483f3956e699ef025c027be45b79c/hostname","HostsPath":"/var/lib/docker/containers/8b2da74bd40aeec791d9e96eeefb78640c38483f3956e699ef025c027be45b79c/hosts","Name":"/grave_turing","Driver":"aufs","ExecDriver":"native-0.2","MountLabel":"","ProcessLabel":"","AppArmorProfile":"","RestartCount":0,"Volumes":{},"VolumesRW":{}}root@ubuntu1404-02:~#
```

docker kill sends a sigkill instead of sigterm (docker kill -s # also works)

docker restart also works

-----Entering a command prompt in ANY container

In most containers, you need bash, and it isn't going to be PID1 to easily get to, or running shh either.

nsenter - allows to enter namespaces and helps get to a shell if not obviously available (Assume for a second 455dd3425263 is the container's short ID)

Use "docker inspect 455dd3425263 | grep -i pid" to get the actual PID of the running container (1924, e.g.)

nsenter -m -u -n -p -i -t 1924

-m mount namespace, -u uts, -n network, process, and IPC namespaces, of -t the target ContainerPID

This will give you a bash prompt, you can say 'exit' and it won't shut the container down, since it isn't a PID1 inside the container! You are just returned safely to your host machine's bash prompt.

There is an easier way though:

docker-enter 455dd3425263

And yet another way is what's probably recommended:

docker -exec -it 455dd3425263 /bin/bash

This can execute any other command as well, (not just bash)

A possible 4th option - Say you don't have nsenter installed?

Copy it from an existing container to /usr/local/bin

docker run -v /usr/local/bin:/target jpetazzo/nsenter

-v to bind mount a volume

base the container on jpetazzo/nsenter (from hub)

-----Dockerfiles

#ubuntu based hello world container

FROM ubuntu:15.04 #base image is needed

MAINTAINER bill@microsoft.com

RUN apt-get update # every run instruction adds a new layer to the image...

creates container, this is run, stops container,

committed to a new image layer

RUN apt-get install -y nginx # creates container based on the last image layer we created,

this is run, stops, commits it's new layer

CMD ["echo", "hello world"]

-- ok so save this named as "**Dockerfile**" in an empty directory

ANY items with it would be included in the build

docker build -t helloworld:0.1 .

-- filename:version.number . ("." meaning "this directory" and "filename" needs to be lowercase)

or docker build -t="apache-img" .

During the build you will see:

```

root@ubuntu1404-02:/pluralsight# docker build -t helloworld:0.1 .
Sending build context to Docker daemon 2.048 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:15.04
--> b12dbb6f7084
Step 1 : MAINTAINER npn@hotmail.com
--> Running in fa6a239afdf8
--> f85bd9d5b925
Removing intermediate container fa6a239afdf8
Step 2 : RUN apt-get update
--> Running in 5a1d42d3cb95
Ign http://archive.ubuntu.com vivid InRelease
Ign http://archive.ubuntu.com vivid-updates InRelease
Ign http://archive.ubuntu.com vivid-security InRelease
Get:1 http://archive.ubuntu.com vivid Release.gpg [933 B]
Get:2 http://archive.ubuntu.com vivid-updates Release.gpg [933 B]
Get:3 http://archive.ubuntu.com vivid-security Release.gpg [933 B]
Get:4 http://archive.ubuntu.com vivid Release [215 kB]
Get:22 http://archive.ubuntu.com vivid-security/main amd64 Packages [40 B]
Get:23 http://archive.ubuntu.com vivid-security/restricted amd64 Packages [40 B]
Get:24 http://archive.ubuntu.com vivid-security/universe amd64 Packages [40 B]
Fetched 20.5 MB in 11s (1788 kB/s)
Reading package lists...
--> dcb3509fde03
Removing intermediate container 5a1d42d3cb95
Step 3 : CMD echo Hello World
--> Running in ccab73aeff95
--> 1b140dcbb438
Removing intermediate container ccab73aeff95
Successfully built 1b140dcbb438

```

Note: the Docker daemon does the build- not the client - which means this could be done over the network if you wanted. The 2.048K "build context" is the contents of the directory (just our Dockerfile)

At each step, you see

"Running in <container#>" and then afterward "Removing intermediate container <container#>"

What this means is the Docker daemon uses a temporary "intermediate" container to build

the step into, then it commits the build, reports the new layer/image build number, and discards that

temporary container used for the build. It then makes another on the next step

It basically repeats: 1) make a workspace, 2) build on it, 3) keep completed work and remove the workspace - and it does this for every layer. It even made a temp container just to put the MAINTAINER directive into!

At the end, "Successfully built d3425263" - this is our final output ContainerID

After the build, "docker images --tree" and "docker history" excerpts show the same layer/image IDs

The 4th image (b12dbb6f7084) is the original Ubuntu image in the FROM directive

```
root@ubuntu1404-02:/pluralsight# docker images --tree s/etcd:latest, coreos/etcd:v0.
└─511136ea3c5a Virtual Size: 0 B
   └─e6c9ea3c5464 Virtual Size: 116.9 MB
      └─ef70b517e969 Virtual Size: 117.2 MB
         └─8680e7835433 Virtual Size: 117.2 MB
            └─b12dbb6f7084 Virtual Size: 117.2 MB Tags: ubuntu:15.04
               └─f85bd9d5b925 Virtual Size: 117.2 MB
                  └─dcb3509fde03 Virtual Size: 137.6 MB
                     └─1b140dccb438 Virtual Size: 137.6 MB Tags: helloworld:0.1
                        └─fe95bf7d5f50 Virtual Size: 188.1 MB
```

1b140dccb438
dcb3509fde03
f85bd9d5b925

```
root@ubuntu1404-02:/pluralsight# docker history 1b140dccb438
IMAGE          CREATED        CREATED BY          SIZE
1b140dccb438   6 minutes ago /bin/sh -c #(nop)  CMD [echo Hello World] 0 B
dcb3509fde03   6 minutes ago /bin/sh -c apt-get update                  20.46 MB
f85bd9d5b925   6 minutes ago /bin/sh -c #(nop)  MAINTAINER              0 B
b12dbb6f7084   44 hours ago  /bin/sh -c #(nop)  CMD [/bin/bash]         0 B
8680e7835433   44 hours ago  /bin/sh -c sed -i 's/^#\s*\s*(deb.*universe\)$/' 1.879 kB
ef70b517e969   44 hours ago  /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic 215 kB
e6c9ea3c5464   44 hours ago  /bin/sh -c #(nop)  ADD file:6a5711b9f3102a767f 116.9 MB
511136ea3c5a   18 months ago
```

Docker Hub:

Looking at Ubuntu Dockerfile (pasted into addendum of this doc)

FROM scratch - for the base image

ADD ubuntu-image-blahblahblah.tar.gz / <----- copy contents to "/"

RUN echo '#!/bin/sh /

&& more script code /

&& etc script code /

and some other RUN directives. One run line containing many instructions is better than a dozen which would make a dozen layers. You can also insert scripts into run directives!

Registry - hub.docker.com - Recall that registries like hub.docker.com contain repositories

Repos might be

Fedora (image-x, image-y, image-z); Ubuntu (image-x, image-y, image-z); MongoDB (image-x, image-y, image-z)

If you start an account on Docker Hub, it will typically have yourname namespace/reponame

Give a description and make repo public or private. Free accounts only get one private repo

Pushing Docker images up

You must tag the image first.

- docker tag (imageid) myusername/repo-name:1.version

- docker tag d3425263 dingo12/helloworld:1.0

do "docker images" again and it will show up with both old and new tagged entry

docker build -t to add tag at build

To publish

- docker push dingo12/helloworld:1.0

Please login prior to push: (email, username, pass)

Only new image layers get pushed, so you will see the list of lower layers (containing the core os and stuff):

5111336ea3c54a: image already pushed, skipping

(...until...)

Pushing tag for rev [d3425263] on https://cdn-registry1-docker.io/v1/repositories/dingo12/helloworld/tags/1.0

Cleaning things- to delete an image you first have to delete any containers linking to it
docker images (shows our images)

docker ps -a (show containers)

docker rm (containerID1) (containerID2) (containerID3) <----remove containers

docker rmi (imageID1) (imageID2) (imageID3) <----remove images

- As of this video, no way to delete a repo from Docker Hub on command line- use GUI

Private registries

Registry v1 = Python, v2 Golang which brings v2 image format, JSON registry API, and is more secure

docker run -d -p 5000:5000 registry

The "-p" option exposes the port on the container and the host, port 5000 and specifies a container based on the image "registry"

When you do a docker push, it will be referencing the repo with the
dns.name.given:5000/repo-name

This becomes a permanent part of the naming context of any repo that is pushed to the registry, so when those get moved around they will be looking for that DNS info

Example tagged the image to move to private registry:

- docker tag 7868e987f987a7 myrepo.home.zone:5000/priv-test

Then tells push to move images tagged for this repo

- docker push myrepo.home.zone:5000/priv-test

It is noted here that this line was placed in /etc/default/docker (the config file) in place of showing us SSL setup

DOCKER_OPTS="--insecure-registry myrepo.home.zone:5000"

(That's on Ubuntu)

(At this point, example switched to another machine that is so far unaware of the private repo, and this sets it up)

When you need Docker to pull from the specific repo when issuing the run command:

- docker run -d myrepo.home.zone:5000/priv-test

In the systemd unit file for the docker service (This is on CentOS) this was added to get around not provisioning SSL options. After you do this you'll need to restart Docker to force reading the config file.

In /usr/lib/systemd/system/docker.service

ExecStart=/usr/bin/docker -d \$OPTIONS \$DOCKER_STORAGE_OPTIONS --insecure-

registry myrepo.home.zone:5000

2014 - DHE - Docker Hub Enterprise- alt to using Docker Hub or setting up fledgling registries

Docker build cache - when you build something, if you build something with the same build steps, those steps can just be received from the build cache so the images/layers wont have to be downloaded all over again

Make a webserver dockerfile

```
mkdir web; cd web; vim Dockerfile
```

```
--  
FROM ubuntu:15.04  
RUN apt-get update  
RUN apt-get install -y apache2  
RUN apt-get install -y apache2-utils  
RUN apt-get install -y vim  
RUN apt-get clean  
EXPOSE 80  
CMD ["apache2ctl", "-D", "FOREGROUND"]  
--  
docker build -t="webserver" .  
docker images  
docker run -d -p 80:80 webserver
```

That was 7 layers! Optimize that- this one is 3 images and 20MB smaller:

```
---  
FROM ubuntu:15.04  
RUN apt-get update && apt-get install -y apache2 apache2-utils vim \  
    && apt-get clean \  
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*  
EXPOSE 80  
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

The CMD directive

RUN in dockerfiles is done at build-time, generally to install/remove apps

CMD directives happen at runtime of the container's launch

- you only get one per Dockerfile
- are the equivalent of "docker run" items
- **when "docker run" is used at the command line it overrides CMD lines in the container**

Two forms:

Shell form:

- As if the command was preceded by "/bin/sh -c"

- you get variable expansion, etc. just like a shell
Example `echo "hi there!"` or `echo $var5`

Exec form use a JSON array style `[command, "arg1", "arg2" ...]`

- Containers don't need a shell, and it avoids "string munging"
 - No shell features like variable expansion or even use of special characters like `&&`, `||`
- Despite that, this is the preferred way of doing things.

Example: `["apache2ctl", "-D", "FOREGROUND"]`

ENTRYPOINT preferred method

- specifies the program that the container will run, PERIOD.
- can't be overridden at runtime with "docker run"
- instead, anything following a "docker run" command are taken as arguments to the ENTRYPOINT
- you can use "docker run --entrypoint" to escape this behavior [???

ENTRYPOINT can't be overrun by normal commands like `docker run`
Any command at runtime is an argument to ENTRYPOINT

So

FROM `ubuntu:15.04`

RUN `apt-get ...`

ENTRYPOINT `["echo"]`

Build and run

`docker build -t "hw2"`

`docker run hw2 hellooooo there`

`>hellooooo there`

This is also illustrated in these Dockerfile lines (even though this isn't efficient in a Dockerfile)

ENTRYPOINT `["apachectl"]`

CMD `["-D", "FOREGROUND"]`

ENTRYPOINT (default command to execute at runtime)

`--entrypoint=""`: Overwrite the default entrypoint set by the image

The ENTRYPOINT of an image is similar to a COMMAND because it specifies what executable to run when the container starts, but it is (purposely) more difficult to override. The ENTRYPOINT gives a container its default nature or behavior, so that when you set an ENTRYPOINT you can run the container as if it were that binary, complete with default options, and you can pass in more options via the COMMAND. But, sometimes an operator may want to run something else inside the container, so you can override the default ENTRYPOINT at runtime by using a string to specify the new ENTRYPOINT. Here is an example of how to run a shell in a container that has

been set up to automatically run something else (like /usr/bin/redis-server):

```
$ docker run -it --entrypoint /bin/bash example/redis
```

or two examples of how to pass more parameters to that ENTRYPOINT:

```
$ docker run -it --entrypoint /bin/bash example/redis -c ls -l
```

```
$ docker run -it --entrypoint /usr/bin/redis-cli example/redis --help
```

You can reset a containers entrypoint by passing an empty string, for example:

```
$ docker run -it --entrypoint="" mysql bash
```

Note: Passing --entrypoint will clear out any default command set on the image (i.e. any CMD instruction in the Dockerfile used to build it).

ENV - set environmental variables
ENV var1=Hello var2=Dolly
Each use of ENV outputs another layer, this line would output one

Volumes

VOLUME /volume_name <--- this in a Dockerfile does not quite work like host-mounts like below

```
docker run -it -v /test-vol --name=my_volume ubuntu:15.04 /bin/bash
```

- This starts bash as it normally would, but makes a mountpoint /test-vol pointing to my_volume both created when the container is started
- Just like Linux mountpoints, if there is a directory in the container named the same, it shows the mounted volume contents in it's place
- You will see this at the bottom of the output of "docker inspect my_volume"
- As reported, the host machine will have the contents in /var/lib/docker/vfs/dir/ as listed below

- After created, you can also do docker run -it -volumes-from=my_volume ubuntu:15.04 /bin/bash

- It doesn't have to running - this mounts it: docker run -v /data:/data
- To delete volume within the container, you have to run "docker rm -v <container>"
- Without -v the volume doesn't get deleted (recall it is in "/var/lib/docker/vfs", not "/var/lib/docker/aufs/diff")

```
"Volumes": {
  "/test-vol": "/var/lib/docker/vfs/dir/c8d3a36662e5a5c40174ca5e041ed304916e1d3b3fdaffaced38b2c87b6b89df"
},
"VolumesRW": {
  "/test-vol": true
}
```

Docker Networking

The docker0 Bridge

Install the bridge-utils package for brctl

```
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 56:84:7a:fe:97:99 brd ff:ff:ff:ff:ff:ff
    inet 172.17.42.1/16 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::5484:7aff:fe97:9799/64 scope link
        valid_lft forever preferred_lft forever
```

vethx on the Docker host looks like eht0 in the container namespace

In "docker inspect" output:

```
"NetworkSettings": {
  "Bridge": "docker0",
  "Gateway": "172.17.42.1",
  "IPAddress": "172.17.0.15",
  "IPPrefixLen": 16,
  "MacAddress": "02:42:ac:11:00:0f",
  "PortMapping": null,
  "Ports": {}
},
```

Recall that the directory with container contents has resolve.conf - this is just a copy of the host's /etc/resolv.conf

The hosts file will simply have the containers local addresses.

Some items can be set on the fly: "docker run --dns=8.8.8.8 --name=myinstance myimage"

In Dockerfile, EXPOSE directive to specify ports to have open. With EXPOSE 80 you might do this:

docker run -d -p 5001:80 --name=myinstance myimage

Using -P (capital) automatically opens and assigns Dockerfile's ports outside without specifying any of them

You can see open ports with "docker ps" but it's more effective to use "docker port instanceName"

```
root@ubuntu1404-02:~/web# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
7e654f1bf609       apache-img:latest  "apache2ctl -D FOREG  4 seconds ago      Up 3 seconds       0.0.0.0:5001->0.0.0.0:80/tcp
01->80/tcp          web1

root@ubuntu1404-02:~/web# docker port web1
80/tcp -> 0.0.0.0:5001
```

How Docker chooses subnet ranges for bridge:

```

func CreateBridgeIface(ifaceName string) error {
    addrs := []string{
        // Here we don't follow the convention of using the 1st IP of the range for the gateway.
        // This is to use the same gateway IPs as the /24 ranges, which predate the /16 ranges.
        // In theory this shouldn't matter - in practice there's bound to be a few scripts relying
        // on the internal addressing or other stupid things like that.
        // The shouldn't, but hey, let's not break them unless we really have to.
        "172.17.42.1/16", // Don't use 172.16.0.0/16, it conflicts with EC2 DNS 172.16.0.23
        "10.0.42.1/16",   // Don't even try using the entire /8, that's too intrusive
        "10.1.42.1/16",
        "10.42.42.1/16",
        "172.16.42.1/24",
        "172.16.43.1/24",
        "172.16.44.1/24",
        "10.0.42.1/24",
        "10.0.43.1/24",
        "192.168.42.1/24",
        "192.168.43.1/24",
        "192.168.44.1/24",
    }
}

```

What if the Docker Bridge picks a subnet or address that's in use and there is a conflict?

On the host, bring docker0 down (ip link del docker0) stop the Docker service, and edit /etc/default/docker

DOCKER_OPTS=--bip=150.150.151/24 [to set the bridge IP (bip)]

Restart the Docker service, and the new settings take effect.

Linking Containers

You can link network interfaces between two containers directly with no ports are exposed outside of both of them.

- Below, one image used runs Apache, and has designated port 80 to be exposed in the Dockerfile, but is not mapped to any outside port, so not actually exposed to any network.
- The instances below are appropriately named for source (src) and receiver (rcvr)
- The "docker run" for rcvr is given a directive "--link=src:ali-src" It makes an alias named "ali-src" pointing at the src instance.

```

root@ubuntu1404-02:~/web# docker run --name=src -d img
a073d44bf842ca5fea5cfe1c48abc0a97c88e5f13bc0b6736a976e998f6cd725
root@ubuntu1404-02:~/web# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
a073d44bf842        img:latest         "apache2ctl -D FOREG   7 seconds ago       Up 5 seconds       80/tcp
src
root@ubuntu1404-02:~/web# docker run --name=rcvr --link=src:ali-src -it ubuntu:15.04 /bin/bash
root@1c8b2b26e9fa:/# root@ubuntu1404-02:~/web#
root@ubuntu1404-02:~/web# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
1c8b2b26e9fa        ubuntu:15.04       "/bin/bash"             21 seconds ago     Up 21 seconds
rcvr
a073d44bf842        img:latest         "apache2ctl -D FOREG   About a minute ago  Up About a minute  80/tcp
src

```

In the truncated output of the commands below, we can see that when the link is set up,

the link on the on the reciever instance is apparent, but nothing shows on the source, since we didn't inform it of anything outside it's container instance:

```
root@ubuntu1404-02:~/web# docker inspect rcvr
{
  "Devices": [],
  "Dns": null,
  "DnsSearch": null,
  "ExtraHosts": null,
  "IpcMode": "",
  "Links": [
    "/src:/rcvr/ali-src"
  ],
  "LxcConf": [],
  "NetworkMode": "bridge",
  "Name": "/rcvr",
  "NetworkSettings": {
    "Bridge": "docker0",
    "Gateway": "172.17.0.29",
    "IPAddress": "172.17.0.29",
    "IPPrefixLen": 16,
    "MacAddress": "02:42:ac:11:00:1d",
    "PortMapping": null,
    "Ports": {}
  }
}

root@ubuntu1404-02:~/web# docker inspect src | grep Links
"Links": null,
```

Finally, in the reciever instance, we can see that there are environmental variables and entries set up for the link in the /etc/hosts file:

```
root@ubuntu1404-02:~/web# docker attach rcvr
root@1c8b2b26e9fa:/# env | grep ALI
ALI_SRC_PORT_80_TCP_ADDR=172.17.0.28
ALI_SRC_PORT_80_TCP_PROTO=tcp
ALI_SRC_PORT_80_TCP_PORT=80
ALI_SRC_PORT=tcp://172.17.0.28:80
ALI_SRC_NAME=/rcvr/ali-src
ALI_SRC_PORT_80_TCP=tcp://172.17.0.28:80
root@1c8b2b26e9fa:/# cat /etc/hosts
172.17.0.29    1c8b2b26e9fa
127.0.0.1     localhost
::1          localhost ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters
172.17.0.28    ali-src
```

Using this same method, the recipient can automatically configure it's apps to use these values to communicate with the source. No ports are exposed outside of these containers, and using multiple sources or multiple recipients are possible.

IPTables and ICC (inter-container communication)

By default IPTables is set up to allow communication between any/all containers.

ICC (inter-container communication and IPTables are set to true

Setting [DOCKER_OPTS= --icc=false] and restarting the Docker service will put an entry to drop packets from docker0 to/from docker0 to/from anywhere (viewable on host- iptables -Lv)

iptables=true/false determines if Docker is permitted to make any modifications to IPTables rules

If you did this, restarted the Docker service, the IPTables drop rule from setting ICC=false would still be in effect (even though set back to true) since Docker has been explicitly instructed it is forbidden from touching IPTables to set it.

Docker Daemon Logging

Recall that the syslog levels are 0= emergency, 1=alert, 2=critical, 3= error, 4=warning, 5=notice, 6=info, 7=debug

The Docker daemon uses (in the same order) fatal, error, info and debug

`docker -d -l debug &`

`docker -d >> <file> 2>$1`

Set default log level in `/etc/default/docker` (for init and Upstart. Not systemd)

`DOCKER_OPTS="--log-level=fatal"`

For a particular container, "`docker logs -f instance_name>`" will follow the log in stdout

One idea is to mount a container volume where the logs get saved

- Writing dockerfiles: good way to do it is run base container and perform items you will put in Dockerfile to see if there are unseen dependencies that have to be met (requiring install directives). It could even uncover things like making sure the right packagename is given to the installer.

- If you ran a docker build and there was an error, "docker images" will usually list the intermediate images used in the build, so you can go back to the "last good" image to troubleshoot

Sidenote: Note below the # of images, containers reported. 5 images comprise 1 primary image, 0 containers

```
root@ubuntu1404-02:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu               15.04              b12dbb6f7084       7 days ago         117.2 MB
root@ubuntu1404-02:~# docker info
Containers: 0
Images: 5
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
  Dirs: 5
Execution Driver: native-0.2
Kernel Version: 3.13.0-32-generic
Operating System: Ubuntu 14.04.1 LTS
CPUs: 1
Total Memory: 490 MiB
Name: ubuntu1404-02
ID: 3KTC:XUJS:UN5N:2B0D:24MD:BRHT:6HKD:SLDH:SKRQ:I225:QN43:4ESF
WARNING: No swap limit support
root@ubuntu1404-02:~# docker images --tree
Warning: '--tree' is deprecated, it will be removed soon. See usage.
├─511136ea3c5a Virtual Size: 0 B
│   └─e6c9ea3c5464 Virtual Size: 116.9 MB
│       └─ef70b517e969 Virtual Size: 117.2 MB
│           └─8680e7835433 Virtual Size: 117.2 MB
│               └─b12dbb6f7084 Virtual Size: 117.2 MB Tags: ubuntu:15.04
```

To get information on a container use its ID or instance name:

```
$ docker inspect d2cc496561d6
[{"Id":
"d2cc496561d6d520cbc0236b4ba88c362c446a7619992123f11c809cded25b4
7",
"Created": "2015-06-08T16:18:02.505155285Z",
"Path": "bash",
"Args": [],
"State": {
  "Running": false,
  "Paused": false,
  "Restarting": false,
  "OOMKilled": false,
  "Dead": false,
  "Pid": 0,
  "ExitCode": 0,
  "Error": "",
  "StartedAt": "2015-06-08T16:18:03.643865954Z",
  "FinishedAt": "2015-06-08T16:57:06.448552862Z"
},
"Image":
"ded7cd95e059788f2586a51c275a4f151653779d6a7f4dad77c2bd34601d94e
4",
"NetworkSettings": {
  "Bridge": "",
  "SandboxID":
"6b4851d1903e16dd6a567bd526553a86664361f31036eaaa2f8454d6f4611f6
f",
  "HairpinMode": false,
  "LinkLocalIPv6Address": "",
  "LinkLocalIPv6PrefixLen": 0,
  "Ports": {},
  "SandboxKey": "/var/run/docker/netns/6b4851d1903e",
  "SecondaryIPAddresses": null,
  "SecondaryIPv6Addresses": null,
  "EndpointID":
"7587b82f0dada3656fda26588aee72630c6fab1536d36e394b2bfbcf898c971
d",
  "Gateway": "172.17.0.1",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "IPAddress": "172.17.0.2",
  "IPPrefixLen": 16,
```



```

    "IPv6Gateway": "",
    "MacAddress": "02:42:ac:12:00:02",
    "Networks": {
        "bridge": {
            "NetworkID":
"7ea29fc1412292a2d7bba362f9253545fecdfa8ce9a6e37dd10ba8bee712981
2",
            "EndpointID":
"7587b82f0dada3656fda26588aee72630c6fab1536d36e394b2bfbcf898c971
d",
            "Gateway": "172.17.0.1",
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "02:42:ac:12:00:02"
        }
    }
},
"ResolvConfPath": "/var/lib/docker/containers/
d2cc496561d6d520cbc0236b4ba88c362c446a7619992123f11c809cded25b47
/resolv.conf",
"HostnamePath": "/var/lib/docker/containers/
d2cc496561d6d520cbc0236b4ba88c362c446a7619992123f11c809cded25b47
/hostname",
"HostsPath": "/var/lib/docker/containers/
d2cc496561d6d520cbc0236b4ba88c362c446a7619992123f11c809cded25b47
/hosts",
"LogPath": "/var/lib/docker/containers/
d2cc496561d6d520cbc0236b4ba88c362c446a7619992123f11c809cded25b47
/
d2cc496561d6d520cbc0236b4ba88c362c446a7619992123f11c809cded25b47
-json.log",
"Name": "/adoring_wozniak",
"RestartCount": 0,
"Driver": "devicemapper",
"MountLabel": "",
"ProcessLabel": "",
"Mounts": [
    {
        "Source": "/data",
        "Destination": "/data",
        "Mode": "ro,Z",
        "RW": false
    }
]
"Propagation": ""

```

```

    }
  ],
  "AppArmorProfile": "",
  "ExecIDs": null,
  "HostConfig": {
    "Binds": null,
    "ContainerIDFile": "",
    "Memory": 0,
    "MemorySwap": 0,
    "CpuShares": 0,
    "CpuPeriod": 0,
    "CpusetCpus": "",
    "CpusetMems": "",
    "CpuQuota": 0,
    "BlkioWeight": 0,
    "OomKillDisable": false,
    "Privileged": false,
    "PortBindings": {},
    "Links": null,
    "PublishAllPorts": false,
    "Dns": null,
    "DnsSearch": null,
    "DnsOptions": null,
    "ExtraHosts": null,
    "VolumesFrom": null,
    "Devices": [],
    "NetworkMode": "bridge",
    "IpcMode": "",
    "PidMode": "",
    "UTSMode": "",
    "CapAdd": null,
    "CapDrop": null,
    "RestartPolicy": {
      "Name": "no",
      "MaximumRetryCount": 0
    },
    "SecurityOpt": null,
    "ReadonlyRootfs": false,
    "Ulimits": null,
    "LogConfig": {
      "Type": "json-file",
      "Config": {}
    },
    "CgroupParent": ""
  },
  "GraphDriver": {
    "Name": "devicemapper",

```

```

        "Data": {
            "DeviceId": "5",
            "DeviceName": "docker-253:1-2763198-
d2cc496561d6d520cbc0236b4ba88c362c446a7619992123f11c809cded25b47
",
            "DeviceSize": "171798691840"
        }
    },
    "Config": {
        "Hostname": "d2cc496561d6",
        "Domainname": "",
        "User": "",
        "AttachStdin": true,
        "AttachStdout": true,
        "AttachStderr": true,
        "ExposedPorts": null,
        "Tty": true,
        "OpenStdin": true,
        "StdinOnce": true,
        "Env": null,
        "Cmd": [
            "bash"
        ],
        "Image": "fedora",
        "Volumes": null,
        "VolumeDriver": "",
        "WorkingDir": "",
        "Entrypoint": null,
        "NetworkDisabled": false,
        "MacAddress": "",
        "OnBuild": null,
        "Labels": {},
        "Memory": 0,
        "MemorySwap": 0,
        "CpuShares": 0,
        "Cpuset": "",
        "StopSignal": "SIGTERM"
    }
}
]

```

----- Ubuntu Dockerfile for Zesty

```

FROM scratch
ADD ubuntu-zesty-core-cloudimg-amd64-root.tar.gz /

```

```

# a few minor docker-specific tweaks
# see https://github.com/docker/docker/blob/
9a9fc01af8fb5d98b8eec0740716226fadb3735c/contrib/mkimage/
debootstrap
RUN set -xe \
\
# https://github.com/docker/docker/blob/
9a9fc01af8fb5d98b8eec0740716226fadb3735c/contrib/mkimage/
debootstrap#L40-L48
    && echo '#!/bin/sh' > /usr/sbin/policy-rc.d \
    && echo 'exit 101' >> /usr/sbin/policy-rc.d \
    && chmod +x /usr/sbin/policy-rc.d \
\
# https://github.com/docker/docker/blob/
9a9fc01af8fb5d98b8eec0740716226fadb3735c/contrib/mkimage/
debootstrap#L54-L56
    && dpkg-divert --local --rename --add /sbin/initctl \
    && cp -a /usr/sbin/policy-rc.d /sbin/initctl \
    && sed -i 's/^exit.*/exit 0/' /sbin/initctl \
\
# https://github.com/docker/docker/blob/
9a9fc01af8fb5d98b8eec0740716226fadb3735c/contrib/mkimage/
debootstrap#L71-L78
    && echo 'force-unsafe-io' > /etc/dpkg/dpkg.cfg.d/docker-
apt-speedup \
\
# https://github.com/docker/docker/blob/
9a9fc01af8fb5d98b8eec0740716226fadb3735c/contrib/mkimage/
debootstrap#L85-L105
    && echo 'DPkg::Post-Invoke { "rm -f /var/cache/apt/
archives/*.deb /var/cache/apt/archives/partial/*.deb /var/cache/
apt/*.bin || true"; };' > /etc/apt/apt.conf.d/docker-clean \
    && echo 'APT::Update::Post-Invoke { "rm -f /var/cache/apt/
archives/*.deb /var/cache/apt/archives/partial/*.deb /var/cache/
apt/*.bin || true"; };' >> /etc/apt/apt.conf.d/docker-clean \
    && echo 'Dir::Cache::pkgcache ""; Dir::Cache::srcpkgcache
"";' >> /etc/apt/apt.conf.d/docker-clean \
\
# https://github.com/docker/docker/blob/
9a9fc01af8fb5d98b8eec0740716226fadb3735c/contrib/mkimage/
debootstrap#L109-L115
    && echo 'Acquire::Languages "none";' > /etc/apt/apt.conf.d/
docker-no-languages \
\
# https://github.com/docker/docker/blob/
9a9fc01af8fb5d98b8eec0740716226fadb3735c/contrib/mkimage/

```

```

debootstrap#L118-L130
    && echo 'Acquire::GzipIndexes "true";
Acquire::CompressionTypes::Order:: "gz";' > /etc/apt/apt.conf.d/
docker-gzip-indexes \
\
# https://github.com/docker/docker/blob/
9a9fc01af8fb5d98b8eec0740716226fadb3735c/contrib/mkimage/
debootstrap#L134-L151
    && echo 'Apt::AutoRemove::SuggestsImportant "false";' > /
etc/apt/apt.conf.d/docker-autoremove-suggests

# delete all the apt list files since they're big and get stale
quickly
RUN rm -rf /var/lib/apt/lists/*
# this forces "apt-get update" in dependent images, which is
also good

# enable the universe
RUN sed -i 's/^#\s*\s*(deb.*universe\)$/\1/g' /etc/apt/
sources.list

# make systemd-detect-virt return "docker"
# See: https://github.com/systemd/systemd/blob/
aa0c34279ee40bce2f9681b496922dedbadfca19/src/basic/virt.c#L434
RUN mkdir -p /run/systemd && echo 'docker' > /run/systemd/
container

# overwrite this with 'CMD []' in a dependent Dockerfile
CMD ["/bin/bash"]

```