



# Agenda

- [Pruning](#)
- [Cleaning Logs](#)
- [Network Address Pools](#)
- [Netshoot](#)
- [Layers](#)
- [Buildkit](#)
- [Local Volume Driver](#)
- [Fixing Permissions](#)



# Tips and Tricks From A Docker Captain



Brandon Mitchell  
Twitter: @sudo\_bmitch  
GitHub: sudo-bmitch

```
$ whoami
```

- Solutions Architect @ BoxBoat
- Docker Captain
- Frequenter of StackOverflow



Who is a Developer?



# Ops 101 - Full Harddrive

# Prune

```
$ docker system prune
```

WARNING! This will remove:

- all stopped containers
- all networks not used by at least one container
- all dangling images
- all build cache



# Prune

```
$ docker system prune
```

WARNING! This will remove:

- all stopped containers
- all networks not used by at least one container
- all dangling images
- all build cache

What this doesn't clean by default:

- Running containers (and their logs)
- Tagged images
- Volumes



# Prune - YOLO

```
$ docker run -d --restart=unless-stopped --name cleanup \  
  -v /var/run/docker.sock:/var/run/docker.sock \  
  docker /bin/sh -c \  
  "while true; do docker system prune -f; sleep 1h; done"
```

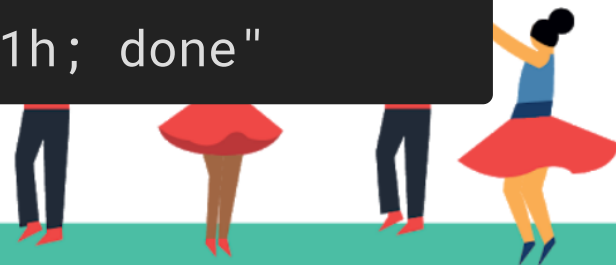




# Prune - YOLO

```
$ docker run -d --restart=unless-stopped --name cleanup \  
  -v /var/run/docker.sock:/var/run/docker.sock \  
  docker /bin/sh -c \  
  "while true; do docker system prune -f; sleep 1h; done"
```

```
$ docker service create --mode global --name cleanup \  
  --mount type=bind,src=/var/run/docker.sock,\  
  dst=/var/run/docker.sock \  
  docker /bin/sh -c \  
  "while true; do docker system prune -f; sleep 1h; done"
```



# Clean Your Logs

- Docker logs to per container json files by default, without any limits
- Rotating yourself could break that json formatting
- Luckily "without any limits" is just the default... we can change that



# Clean Your Logs

```
$ docker container run \  
  --log-opt max-size=10m --log-opt max-file=3 \  
  ...
```



# Clean Your Logs

```
$ docker container run \  
  --log-opt max-size=10m --log-opt max-file=3 \  
  ...
```

```
$ cat docker-compose.yml  
version: '3.7'  
services:  
  app:  
    image: your_app  
    logging:  
      options:  
        max-size: "10m"  
        max-file: "3"
```



# Clean Your Logs

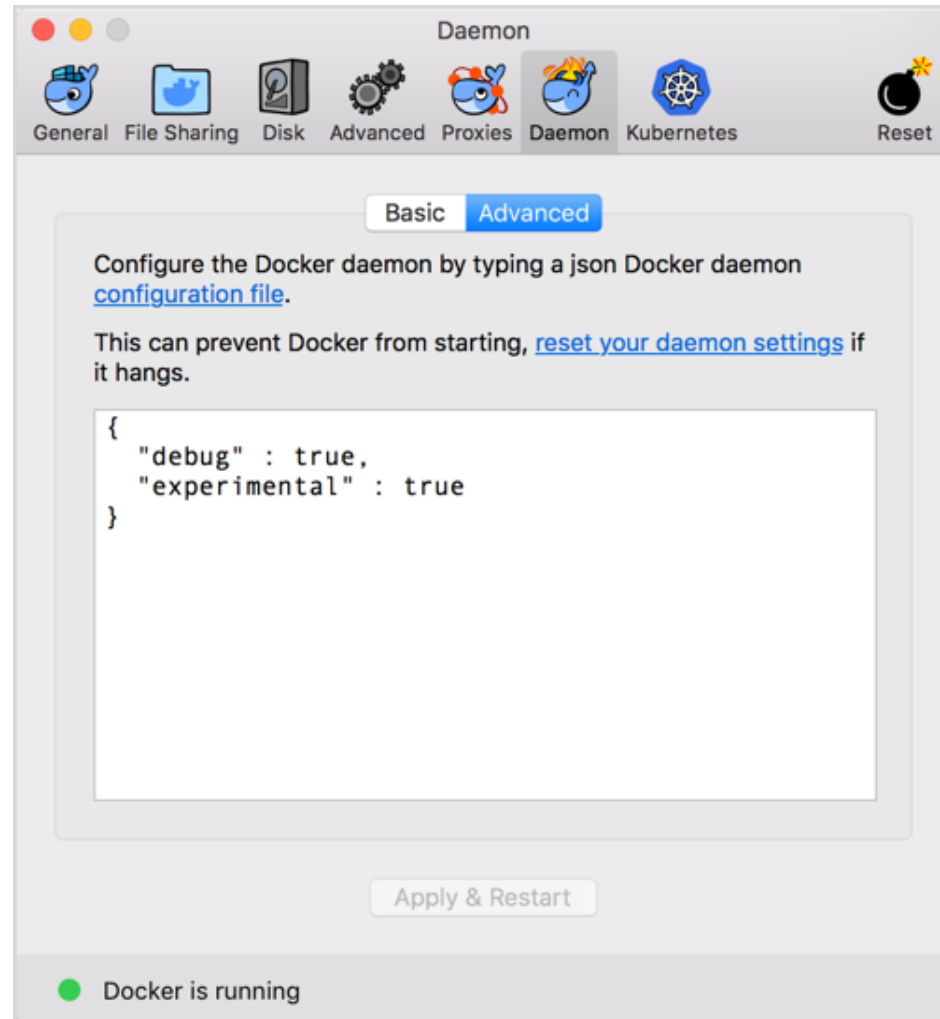
```
version: '3.7'
x-default-opts: &default-opts
  logging:
    options:
      max-size: "10m"
      max-file: "3"
services:
  app_a:
    <<: *default-opts
    image: your_app_a
  app_b:
    <<: *default-opts
    image: your_app_b
```




# Clean Your Logs

```
$ cat /etc/docker/daemon.json
{
  "log-opts": {"max-size": "10m", "max-file": "3"}
}
$ systemctl reload docker
```





 Settings

General

Shared Drives

Advanced

Network

Proxies

Daemon

Kubernetes

Reset

# Daemon

Configure the Docker daemon by typing a json docker daemon [configuration file](#).

☒ Advanced


This can prevent Docker from starting. Use at your own risk!

```
{  
  "registry-mirrors": [],  
  "insecure-registries": [],  
  "debug": true,  
  "experimental": false  
}
```

Docker will restart when applying these settings.

Apply

Docker is running









# Networking

# Subnet Collisions

- Docker networks sometimes conflict with other networks



# Subnet Collisions

- Docker networks sometimes conflict with other networks
- Originally we had the BIP setting

```
$ cat /etc/docker/daemon.json
{
  "bip": "10.15.0.0/24"
}
```



# Subnet Collisions

- Default address pool added in 18.06

```
$ cat /etc/docker/daemon.json
{
  "bip": "10.15.0.0/24",
  "default-address-pools": [
    {"base": "10.20.0.0/16", "size": 24},
    {"base": "10.40.0.0/16", "size": 24}
  ]
}
```



# Subnet Collisions

```
$ docker swarm init --help
...
--default-addr-pool ipNetSlice
--default-addr-pool-mask-length uint32
```



# Subnet Collisions

```
$ docker swarm init --help
...
--default-addr-pool ipNetSlice
--default-addr-pool-mask-length uint32
```

```
$ docker swarm init \
--default-addr-pool 10.20.0.0/16 \
--default-addr-pool 10.40.0.0/16 \
--default-addr-pool-mask-length 24
```



# Network Debugging

- Networks in docker come in a few flavors: bridge, overlay, host, none
- You can also configure the network namespace to be another container



# Network Debugging

- Networks in docker come in a few flavors: bridge, overlay, host, none
- You can also configure the network namespace to be another container

```
$ docker run --name web-app -p 9080:80 -d nginx
```

```
$ docker run -it --rm --net container:web-app \
    nicolaka/netshoot ss -lnt
```

State	Recv-Q	Send-Q	Local	Address:Port	Peer	Address:Port
LISTEN	0	128		*:80	*	*





# Network Debugging

```
$ docker run -it --rm --net container:web-app \
  nicolaka/netshoot tcpdump -n port 80
tcpdump: verbose output suppressed, use -v or -vv for full
protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size
262144 bytes
```



# Network Debugging

```
$ docker run -it --rm --net container:web-app \
  nicolaka/netshoot tcpdump -n port 80
tcpdump: verbose output suppressed, use -v or -vv for full
protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size
262144 bytes
```

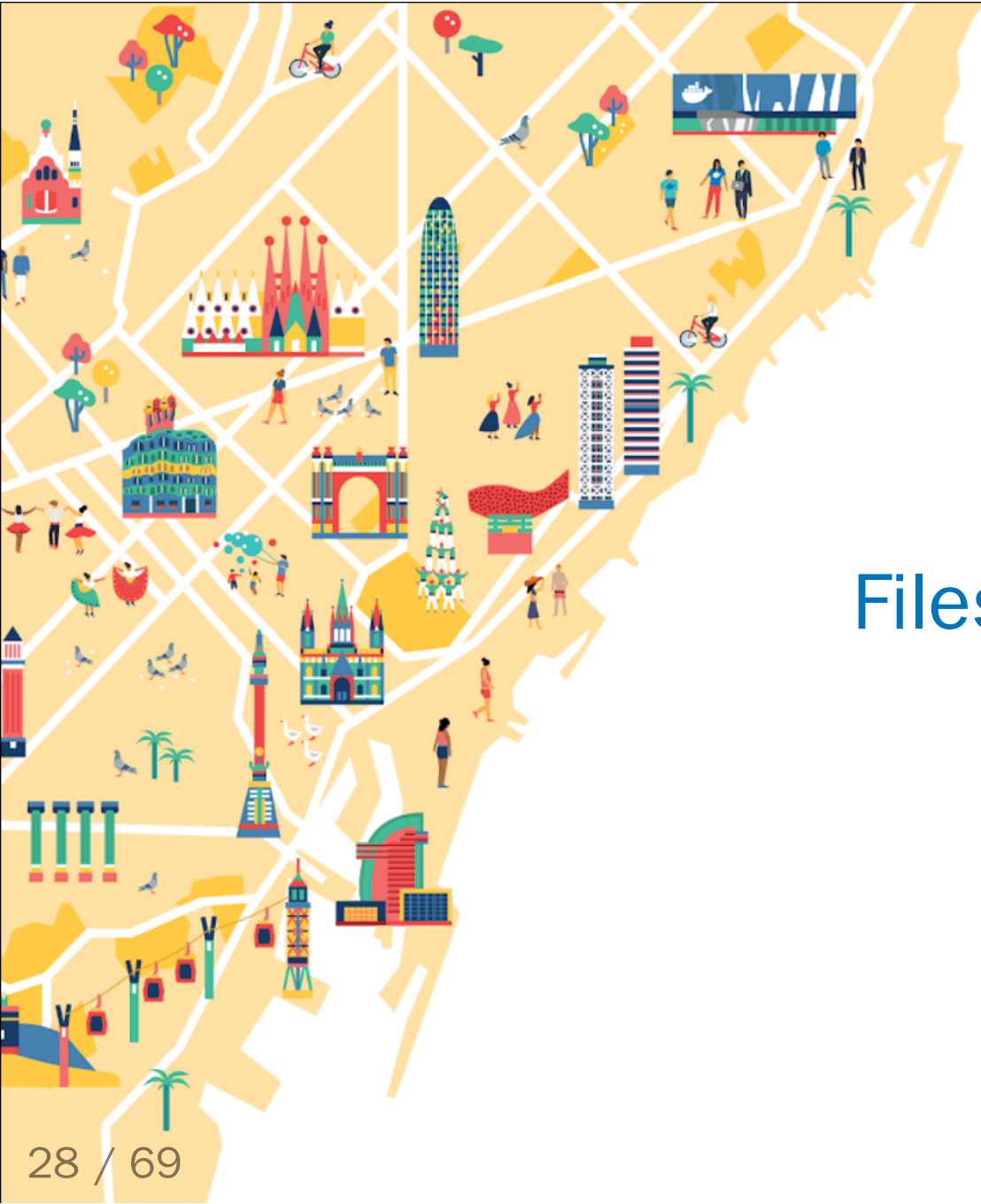
```
$ curl localhost:9080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```



# Network Debugging

```
$ docker run -it --rm --net container:web-app \
  nicolaka/netshoot tcpdump -n port 80
14:08:58.878822 IP 172.17.0.1.55194 > 172.17.0.2.80: Flags [S], ...
14:08:58.878848 IP 172.17.0.2.80 > 172.17.0.1.55194: Flags [S.], ...
14:08:58.878872 IP 172.17.0.1.55194 > 172.17.0.2.80: Flags [.], ...
14:08:58.879089 IP 172.17.0.1.55194 > 172.17.0.2.80: Flags [P.], ...
14:08:58.879110 IP 172.17.0.2.80 > 172.17.0.1.55194: Flags [.], ...
14:08:58.879208 IP 172.17.0.2.80 > 172.17.0.1.55194: Flags [P.], ...
14:08:58.879238 IP 172.17.0.1.55194 > 172.17.0.2.80: Flags [.], ...
14:08:58.879267 IP 172.17.0.2.80 > 172.17.0.1.55194: Flags [P.], ...
14:08:58.879285 IP 172.17.0.1.55194 > 172.17.0.2.80: Flags [.], ...
14:08:58.879695 IP 172.17.0.1.55194 > 172.17.0.2.80: Flags [F.], ...
14:08:58.879739 IP 172.17.0.2.80 > 172.17.0.1.55194: Flags [F.], ...
14:08:58.879776 IP 172.17.0.1.55194 > 172.17.0.2.80: Flags [.], ...
```





# Filesystems and Volumes

# Understanding Layers

```
$ docker image inspect localhost:5000/jenkins-docker:latest \
  --format '{{json .RootFS.Layers}}' | jq .
[
  "sha256:b28ef0b6fef80faa25436bec0a1375214d9a23a91e9b75975bb...",
  ...,
  "sha256:08794ff8753b0fbca869a7ece2dff463cdb7cffd5d7ce792ec0...",
  "sha256:37986c5c5dff18257b9a12a19801828a80aea036992b34d35a3...",
  "sha256:34bb0412a3f6c0f3684e05fcd0a301dc999510511c3206d8cd3...",
  "sha256:696245ae585527c34e2cbc0d01d333aa104693e12e0b79cf193...",
  "sha256:91b63ceb91a75edb481c1ef8b005f9a55aa39d57ac6cc6ef490...",
  "sha256:afddea070d31e748730901215d11b546f4f212114e38e685465...",
  "sha256:0c05256b3bb44190557669126bf69897c7faf7628ff1ed2e2d4...",
  "sha256:0c05256b3bb44190557669126bf69897c7faf7628ff1ed2e2d4..."
]
```



# Understanding Layers

```
$ docker image inspect jenkins/jenkins:lts \
  --format '{{json .RootFS.Layers}}' | jq .
[
  "sha256:b28ef0b6fef80faa25436bec0a1375214d9a23a91e9b75975bb...",
  ...,
  "sha256:08794ff8753b0fbca869a7ece2dff463cdb7cffd5d7ce792ec0...",
  "sha256:37986c5c5dff18257b9a12a19801828a80aea036992b34d35a3...",
  "sha256:34bb0412a3f6c0f3684e05fcd0a301dc999510511c3206d8cd3..."
]
```



# Understanding Layers

```
$ docker image history localhost:5000/jenkins-docker:latest
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
6ca185e69f2e	292 years ago	LABEL org.label-schema	0B	buildkit
<missing>	292 years ago	HEALTHCHECK &{["CMD-SH	0B	buildkit
<missing>	292 years ago	ENTRYPOINT ["/entrypoi	0B	buildkit
<missing>	3 weeks ago	COPY entrypoint.sh /en	1.08kB	buildkit
<missing>	3 weeks ago	RUN  2 GOSU_VERSION=1.	203MB	buildkit
<missing>	3 weeks ago	RUN /bin/sh -c apt-get	83.6MB	buildkit
<missing>	292 years ago	USER root	0B	buildkit
<missing>	6 weeks ago	/bin/sh -c #(nop) COPY	6.11kB	
<missing>	6 weeks ago	/bin/sh -c #(nop) USER	0B	
<missing>	6 weeks ago	/bin/sh -c #(nop) EXPO	0B	
<missing>	7 weeks ago	/bin/sh -c apt-get upd	2.21MB	
<missing>	7 weeks ago	/bin/sh -c #(nop) ADD	101MB	



# Understanding Layers

```
$ DOCKER_BUILDKIT=0 docker build --no-cache --rm=false .  
Sending build context to Docker daemon 146.4kB  
...  
Step 5/17 : RUN apt-get update && DEBIAN_FRONTEND=noninteracti...  
---> Running in 1fc215ebb603  
...  
---> d6dff86e8b89  
Step 9/17 : RUN curl -fsSL https://download.docker.com/linux/de...  
---> Running in a7a3a942a617  
...  
---> a241c22525d8  
...  
Successfully built b01e4c46a2bf
```





# Understanding Layers

```
$ docker container diff 1fc215ebb603
```

```
C /etc
```

```
A /etc/python3.5
```

```
A /etc/python3.5/sitecustomize.py
```

```
...
```

```
C /usr/bin
```

```
A /usr/bin/pygettext3
```

```
A /usr/bin/help2tags
```

```
A /usr/bin/python3
```

```
A /usr/bin/rvim
```

```
A /usr/bin/view
```

```
A /usr/bin/python3.5
```

```
...
```



# Understanding Layers

- If you create a temporary file in a step, delete it in that same step
- Look for unexpected changes that trigger a copy-on-write, e.g. timestamps
- Do your dirty work in early stages of a multi-stage build



# Merge RUN Commands

```
RUN apt-get update  
RUN apt-get install -y curl  
RUN rm -rf /var/lib/apt/lists/*
```



# Merge RUN Commands

```
RUN apt-get update  
RUN apt-get install -y curl  
RUN rm -rf /var/lib/apt/lists/*
```

```
RUN apt-get update \  
&& apt-get install -y curl \  
&& rm -rf /var/lib/apt/lists/*
```



# Use Multi-Stage

```
FROM openjdk:jdk as build
RUN apt-get update \
  && apt-get install -y maven
COPY code /code
RUN mvn build

FROM openjdk:jre as final
COPY --from build /code/app.jar /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```



"Hold my beer."

--BuildKit

# BuildKit is Awesome

```
# syntax = tonistiigi/dockerfile:runmount20180607
FROM openjdk:jdk as build
RUN apt-get update \
  && apt-get install -y maven
RUN --mount=type=bind,target=/code,source=code \
  --mount=type=cache,target=/root/.m2 \
  mvn build

FROM openjdk:jre as final
COPY --from build /output/app.jar /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```



# BuildKit is Awesome

```
$ export DOCKER_BUILDKIT=1  
$ docker build -t your_image .
```





# BuildKit is Awesome

```
$ export DOCKER_BUILDKIT=1  
$ docker build -t your_image .
```

```
$ cat /etc/docker/daemon.json  
{ "features": {"buildkit": true} }
```



# BuildKit is Awesome

```
# syntax = docker/dockerfile:experimental
FROM python:3
RUN pip install awscli
RUN --mount=type=secret,id=aws,target=/root/.aws/credentials \
    aws s3 cp s3://... ..
```

```
$ docker build --secret id=aws,src=$HOME/.aws/credentials \
    -t s3-app .
```



# BuildKit is Awesome

```
FROM openjdk:jdk as build
COPY src /src
RUN mvn build
CMD java -jar /app-a.jar

FROM build as test
RUN mvn test

FROM build as dev
CMD /bin/bash

FROM openjdk:jre as release
COPY --from=build /app.jar /
CMD java -jar /app.jar
```



# BuildKit is Awesome

```
FROM openjdk:jdk as build
COPY src /src
RUN mvn build
CMD java -jar /app-a.jar
```

```
FROM build as test
RUN mvn test
```

```
FROM build as dev
CMD /bin/bash
```

```
FROM openjdk:jre as release
COPY --from=build /app.jar /
CMD java -jar /app.jar
```



# BuildKit is Awesome

- Only runs the build steps needed for target stage
- Dockerfile parser can be updated without updating docker engine
- Build context ignores files you do not ADD or COPY
- The build context is cached, similar to rsync
- Uses remote registries efficiently for layer caching





# Volumes

# Local Volume Driver

the docs   Guides   Product manuals   Glossary   Reference   Samples

Another example that uses `btrfs` :

```
$ docker volume create --driver local \  
  --opt type=btrfs \  
  --opt device=/dev/sda2 \  
  foo
```

Another example that uses `nfs` to mount the `/path/to/dir` in `rw` mode from `192.168.1.1` :

```
$ docker volume create --driver local \  
  --opt type=nfs \  
  --opt o=addr=192.168.1.1,rw \  
  --opt device=:/path/to/dir \  
  foo
```



# NFS Mounts

```
$ docker volume create \  
  --driver local \  
  --opt type=nfs \  
  --opt o=nfsvers=4,addr=nfs.example.com,rw \  
  --opt device=:/path/to/dir \  
  foo
```





# NFS Mounts

```
$ docker container run -it --rm \  
--mount \  
    type=volume,\  
    dst=/container/path,\  
    volume-driver=local,\  
    volume-opt=type=nfs,\  
    \"volume-opt=o=nfsvers=4,addr=nfs.example.com\",\  
    volume-opt=device=:/host/path \  
foo
```



# NFS Mounts

```
$ docker service create \  
  --mount \  
    type=volume,\  
    dst=/container/path,\  
    src=foo-nfs-data,\  
    volume-driver=local,\  
    volume-opt=type=nfs,\  
    \"volume-opt=o=nfsvers=4,addr=nfs.example.com\",\  
    volume-opt=device=:/host/path \  
foo
```



# NFS Mounts

```
version: '3.7'
volumes:
  nfs-data:
    driver: local
    driver_opts:
      type: nfs
      o: nfsvers=4,addr=nfs.example.com,rw
      device: ":/path/to/dir"
services:
  app:
    volumes:
      - nfs-data:/data
...
```



# Other Filesystem Mounts

```
version: '3.7'
volumes:
  ext-data:
    driver: local
    driver_opts:
      type: ext4
      o: ro
      device: "/dev/sdb1"
services:
  app:
    volumes:
      - ext-data:/data
...
```



# Overlay Filesystem as a Volume

```
version: '3.7'
volumes:
  overlay-data:
    driver: local
    driver_opts:
      type: overlay
      device: overlay
      o: lowerdir=${PWD}/data2:${PWD}/data1, \
        upperdir=${PWD}/upper,workdir=${PWD}/workdir
services:
  app:
    volumes:
      - overlay-data:/data
  ...
```



# Named Bind Mount

```
version: '3.7'
volumes:
  bind-test:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /home/user/test
services:
  app:
    volumes:
      - "bind-test:/test"
...
```



That's nice, but I just use:  
`$(pwd)/code:/code`

That's nice, but I just use:

~~\$(pwd)/code:/code~~

"\$(pwd)/code:/code"



# Fixing UID/GID

```
FROM openjdk:jdk as build
RUN useradd -m app
USER app
COPY code /home/app/code
RUN --mount=target=/home/app/.m2,type=cache \
    mvn build
CMD ["java", "-jar", "/home/app/app.jar"]
```



# Fixing UID/GID

```
version: '3.7'
volumes:
  m2:
services:
  app:
    build:
      context: .
      target: build
    image: registry:5000/app/app:dev
    command: "/bin/sh -c 'mvn build && java -jar /home/app/app.jar'"
    volumes:
      - ./code:/home/app/code
      - m2:/home/app/.m2
```



# Fixing UID/GID

```
Error accessing /home/app/code: permission denied
```



# Fixing UID/GID

```
Error accessing /home/app/code: permission denied
```

- `app` inside the container doesn't match `$USER` on the host



# Fixing UID/GID

Possible solutions:

- Run everything as root
- Change permissions to 777
- Adjust each developers uid/gid to match image
- Adjust image uid/gid to match developers
- Change the container uid/gid from `run` or `compose`



# Fixing UID/GID

Possible solutions:

- Run everything as root
- Change permissions to 777
- Adjust each developers uid/gid to match image
- Adjust image uid/gid to match developers
- Change the container uid/gid from `run` or `compose`
- "... or we could use a shell script" - Some Ops Guy



# Disclaimer

The following slide may not be suitable for all audiences

# Fixing UID/GID

```
# update the uid
if [ -n "$opt_u" ]; then
    OLD_UID=$(getent passwd "${opt_u}" | cut -f3 -d:)
    NEW_UID=$(ls -nd "$1" | awk '{print $3}')
    if [ "$OLD_UID" != "$NEW_UID" ]; then
        echo "Changing UID of $opt_u from $OLD_UID to $NEW_UID"
        usermod -u "$NEW_UID" -o "$opt_u"
        if [ -n "$opt_r" ]; then
            find / -xdev -user "$OLD_UID" -exec chown -h "$opt_u" {} \;
        fi
    fi
fi
```





# Fixing UID/GID

```
FROM openjdk:jdk as build
COPY --from=sudobmitch/base:scratch / /
COPY entrypoint.sh /usr/bin/
ENTRYPOINT ["/usr/bin/entrypointd.sh"]
RUN useradd -m app
USER app
COPY code /home/app/code
RUN --mount=target=/home/app/.m2,type=cache \
    mvn build
CMD ["java", "-jar", "/home/app/app.jar"]
```



# Fixing UID/GID

```
#!/bin/sh
if [ "$(id -u)" = "0" ]; then
    fix-perms -r -u app -g app /code
    exec gosu app "$@"
else
    exec "$@"
fi
```



# Fixing UID/GID

```
version: '3.7'
volumes:
  m2:
services:
  app:
    build:
      context: .
      target: build
    image: registry:5000/app/app:dev
    command: "/bin/sh -c 'mvn build && java -jar /home/app/app.jar'"
    user: "0.0"
    volumes:
      - ./code:/home/app/code
      - m2:/home/app/.m2
```



# Fixing UID/GID

- Developers can run the same image and compose file on multiple systems
- App runs with the developers individual uid/gid
- Changes to `./code` are owned by the developer
- Same image in prod can run without ever needing root





# Thank You

[github.com/sudo-bmitch/presentations](https://github.com/sudo-bmitch/presentations)  
[github.com/sudo-bmitch/docker-base](https://github.com/sudo-bmitch/docker-base)



Brandon Mitchell  
Twitter: @sudo\_bmitch  
GitHub: sudo-bmitch