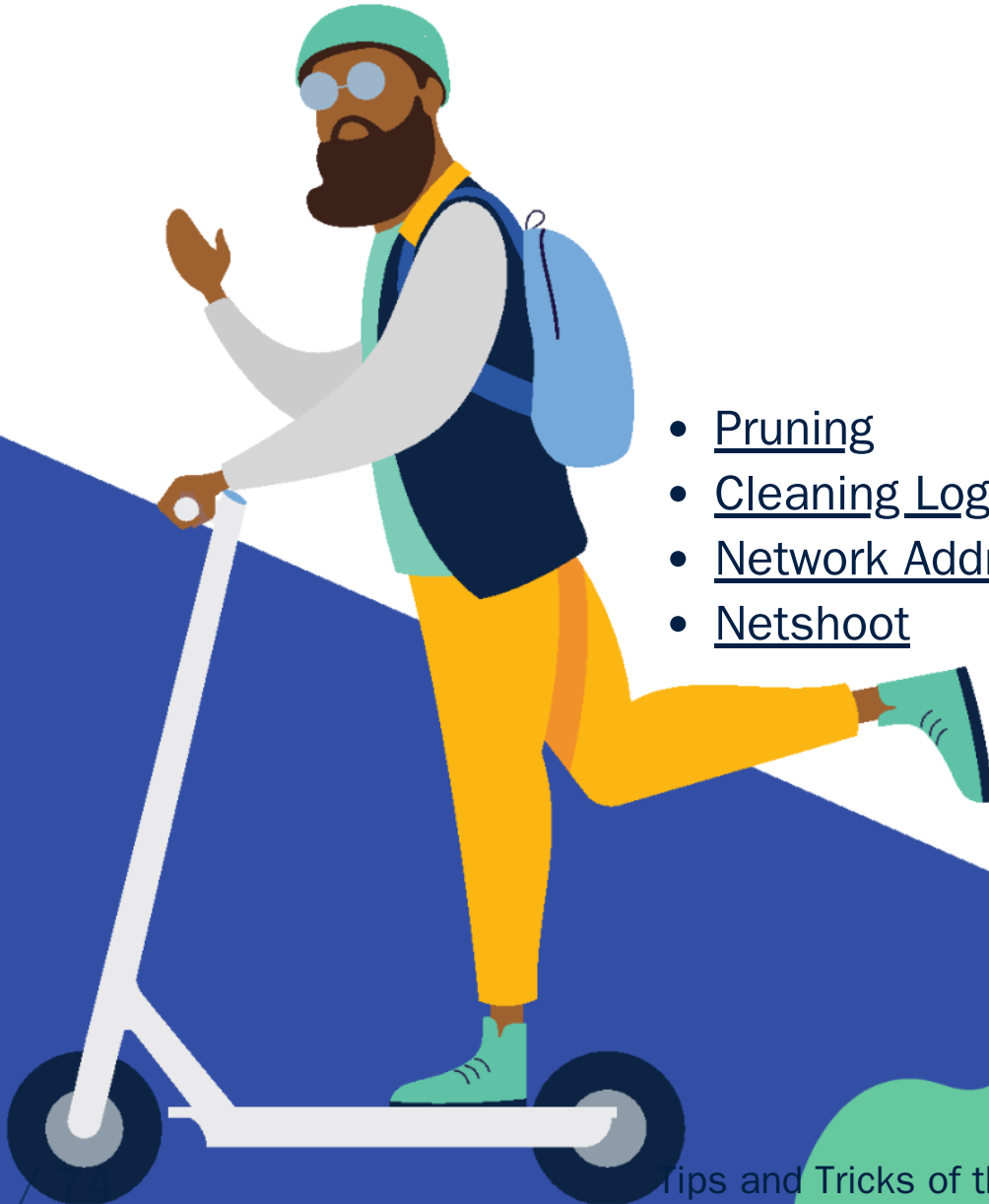


Agenda

- Pruning
- Cleaning Logs
- Network Address Pools
- Netshoot
- Layers
- BuildKit
- Local Volume Driver
- Fixing Permissions



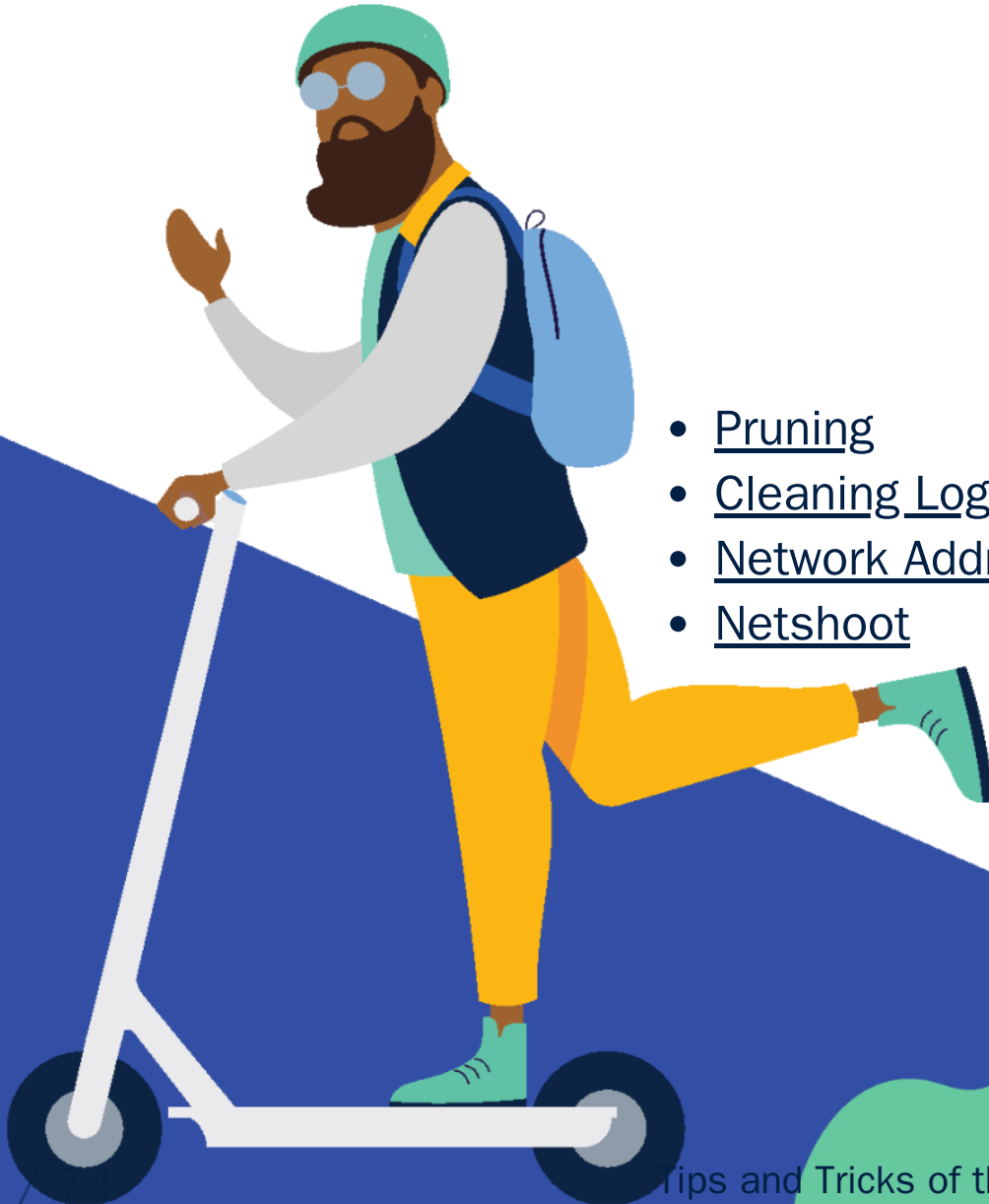
Tips and Tricks Of The Docker Captains



Brandon Mitchell
Twitter: @sudo_bmitch
GitHub: sudo-bmitch

Agenda

- Pruning
- Cleaning Logs
- Network Address Pools
- Netshoot
- Layers
- BuildKit
- Local Volume Driver
- Fixing Permissions



```
$ whoami
```

```
Brandon Mitchell aka bmitch
```

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



Who is a Developer?



Disk Usage

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"
```



Container Logs

Demo

Showing json logs

Demo

Showing "max-size" and "max-file" options

Demo

Showing "local" logging driver

Clean Your Logs

```
$ cat docker-compose.yml
version: '3.7'
services:
  app:
    image: sudobmitch/loggen
    command: [ "150", "180" ]
    logging:
      options:
        max-size: "10m"
        max-file: "3"
```

Clean Your Logs

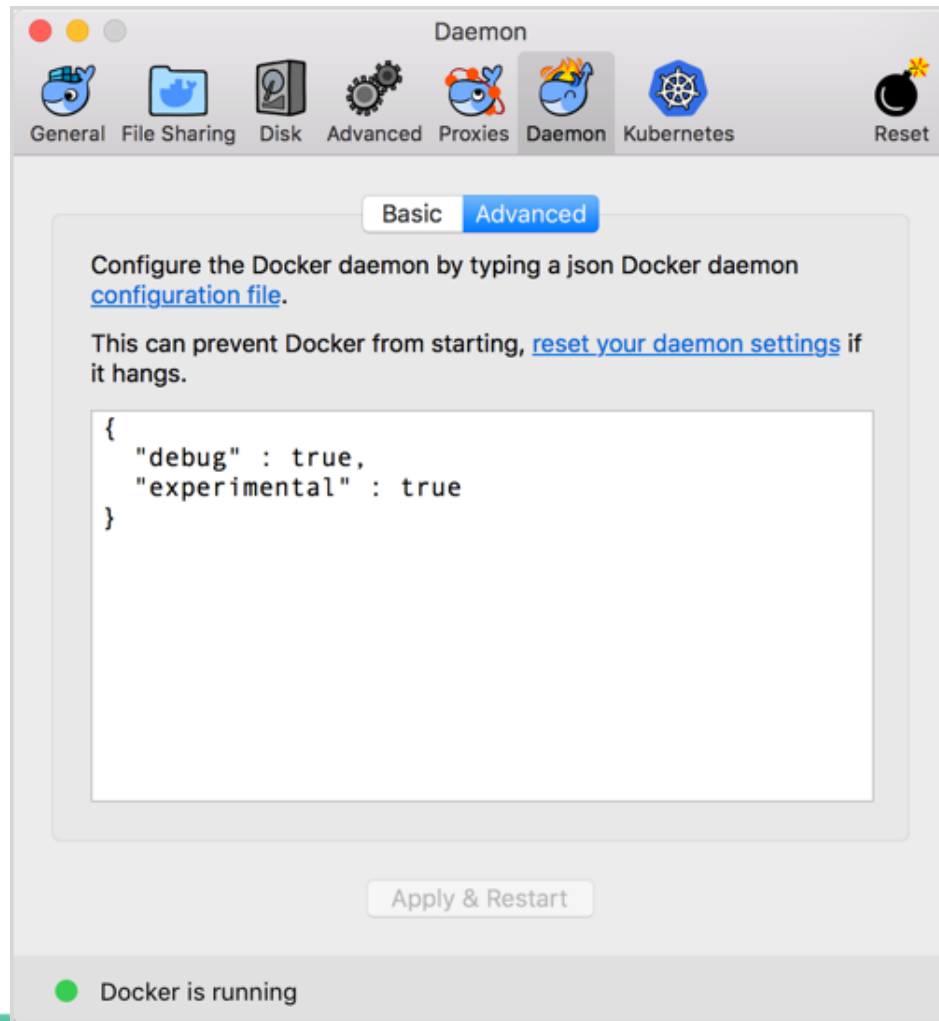
```
version: '3.7'
x-defaults:
  service: &default-svc
  image: sudobmitch/loggen
  logging: { options: { max-size: "10m", max-file: "3" } }
services:
  cat:
    <<: *default-svc
    command: [ "300", "120" ]
    environment: { pet: "cat" }
  turtle:
    <<: *default-svc
    labels: { name: "gordon", levels: "all the way down" }
```


Clean Your Logs

- Best option to prevent container logs from filling disk space

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

$ systemctl reload docker
```




Settings

- General
- Shared Drives
- Advanced
- Network
- Proxies
- Daemon**
- Kubernetes
- Reset

● Docker is running

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



Networking

Subnet Collisions

- Docker networks sometimes conflict with other networks

Subnet Collisions

- Docker networks sometimes conflict with other networks
- BIP, bridge network named "bridge"

```
$ cat /etc/docker/daemon.json
{
  "bip": "10.15.0.1/24"
}
```

Subnet Collisions

- Default address poll added in 18.06

```
$ cat /etc/docker/daemon.json
{
  "bip": "10.15.0.1/24",
  "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

- Debugging networks from the host doesn't see inside the container namespace
- Debugging inside the container means installing tools inside that container

Network Debugging

- Debugging networks from the host doesn't see inside the container namespace
- Debugging inside the container means installing tools inside that container
- Sidecars aren't just for Kubernetes

Demo

Showing "netshoot"

Network Debugging

```
$ docker run --name web -p 9999:80 -d nginx
```

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

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



Layered Filesystem

Demo

Showing layers in "docker inspect"

Demo

Showing "docker image history"

Understanding Layers

```
$ docker image build --rm=false --no-cache .  
$ docker container diff ...
```

Demo

Showing "docker container diff"

Understanding Layers

- Delete temporary file in the same step where they are created
- Small changes to big files are big changes
- Merge your **RUN** commands together

From Bad ...

```
FROM golang:1.11
RUN adduser --disabled-password --gecos appuser appuser
WORKDIR /src
COPY . /src/
RUN go build -o app .
WORKDIR /
RUN cp /src/app /app
RUN chown appuser /app
RUN chmod 755 /app
RUN rm -r /src
USER appuser
CMD /app
```

... to Okay

```
FROM golang:1.11
RUN adduser --disabled-password --gecos appuser appuser
COPY . /src/
RUN cd /src \
  && go build -o app . \
  && cd / \
  && cp /src/app /app \
  && chown appuser /app \
  && chmod 755 /app \
  && rm -r /go/pkg /root/.cache/go-build /src
USER appuser
CMD /app
```

Multi-stage Builds

- Everything we learned about making efficient images is now wrong
- Build stage splits RUN lines to maximize caching
- Only the released stage needs to be layer efficient

```
FROM golang:1.11-alpine as build
RUN apk add --no-cache git ca-certificates
RUN adduser -D appuser
WORKDIR /src
COPY . /src/
RUN CGO_ENABLED=0 go build -o app .

FROM scratch as release
COPY --from=build /etc/passwd /etc/group /etc/
COPY --from=build /src/app /app
USER appuser
CMD [ "/app" ]

FROM alpine as dev
COPY --from=build /src/app /app
CMD [ "/app" ]

FROM release
```

Demo

Showing multi-stage results

"Hold my beer."

--BuildKit

BuildKit Features For Everyone

- GA in Docker 18.09
- Context only pulls needed files
- Multi-stage builds use a dependency graph
- Cache from a remote registry
- Pruning has options for cache age and size to keep

BuildKit Cache Pruning

```
$ docker builder prune --keep-storage=1GB --filter until=72h
```

BuildKit Cache Pruning

```
$ docker builder prune --keep-storage=1GB --filter until=72h
```

```
$ cat /etc/docker/daemon.json
{
  "builder": {
    "gc": {
      "enabled": true,
      "policy": [
        {"keepStorage": "512MB", "filter": ["unused-for=168h"]},
        {"keepStorage": "30GB", "all": true}
      ]
    }
  }
}
```

BuildKit Experimental Features

- Frontend parser can be changed
- Bind Mounts, from build context or another image
- Cache Mounts, similar to a named volume
- Tmpfs Mounts
- Build Secrets, file never written to image filesystem
- SSH Agent, private Git repos

```
# syntax=docker/dockerfile:experimental

FROM golang:1.11-alpine as build
RUN apk add --no-cache git ca-certificates tzdata
RUN adduser -D appuser
WORKDIR /src
COPY . /src/
RUN --mount=type=cache,id=gomod,target=/go/pkg/mod/cache \
    --mount=type=cache,id=goroot,target=/root/.cache/go-build \
    CGO_ENABLED=0 go build -o app .
USER appuser
CMD ./app
```

Demo

Showing BuildKit

Enable BuildKit

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


Enable BuildKit

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

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



Volumes

Local Volume Driver

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/on/server \  
  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-vol:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /home/user/host-dir
services:
  app:
    volumes:
      - "bind-vol:/container-dir"
      - "./code:/code"
...
```


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

That's nice, but I just use:

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

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

Dockerfile for Java

```
# syntax=docker/dockerfile:experimental
FROM openjdk:jdk as build
RUN apt-get update \
  && apt-get install -y maven \
  && useradd -m app
COPY code /code
RUN --mount=target=/home/app/.m2,type=cache \
  mvn build
CMD ["java", "-jar", "/code/app.jar"]
USER app

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

Developer Compose File

```
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 /code/app.jar'"
    volumes:
      - m2:/home/app/.m2
      - ./code:/code
```

Problem with the Developer Workflow

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

Problem with the Developer Workflow

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

- UID for `app` inside the container doesn't match our UID on the host

Problem with the Developer Workflow

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

- UID for `app` inside the container doesn't match our UID on the host
- Unless you're on MacOS or VirtualBox

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 bad 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 bad 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`

Another solution:

- "Use a shell script" - Some Ops Guy

Disclaimer

The following slide may not be suitable for all audiences

Fixing UID/GID: fix-perms

```
# update the uid
if [ -n "$opt_u" ]; then
    OLD_UID=$(getent passwd "${opt_u}" | cut -f3 -d:)
    NEW_UID=$(stat -c "%u" "$1")
    if [ "$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: Dockerfile

```
# syntax=docker/dockerfile:experimental
FROM openjdk:jdk as build
COPY --from=sudobmitch/base:scratch / /
RUN apt-get update \
    && apt-get install -y maven \
    && useradd -m app
COPY code /code
RUN --mount=target=/home/app/.m2,type=cache \
    mvn build
COPY entrypoint.sh /usr/bin/
ENTRYPOINT ["/usr/bin/entrypoint.sh"]
CMD ["java", "-jar", "/code/app.jar"]
USER app
```

Fixing UID/GID: entrypoint.sh

```
#!/bin/sh
if [ "$(id -u)" = "0" ]; then
    # running on a developer laptop as root
    fix-perms -r -u app -g app /code
    exec gosu app "$@"
else
    # running in production as a user
    exec "$@"
fi
```

Fixing UID/GID: Developer Compose File

```
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 /code/app.jar'"
    user: "0:0"
    volumes:
      - m2:/home/app/.m2
      - ./code:/code
```

Fixing UID/GID: Production Compose File

```
version: '3.7'  
services:  
  app:  
    image: registry:5000/app/app:${build_num}
```


Fixing UID/GID: Recap

Developers:

- Mount code as from the host
- Container starts entrypoint as root
- Entrypoint changes uid of `app` user to match uid of `/code`
- Entrypoint switches from root to app
- Pid 1 is the app with a uid matching the host
- Reads and writes to `/code` happen as the developers uid

Production:

- Runs without root or a volume
- Entrypoint skips `fix-perms` and `gosu`

Thank You

- Rate this session in the DockerCon App
- github.com/sudo-bmitch/presentations
- github.com/sudo-bmitch/docker-base



Brandon Mitchell
Twitter: @sudo_bmitch
GitHub: sudo-bmitch