DATASTAX®
DataStax Monday Learning

DataStax Monday Learning

Upgrade yourself, unlock new skills

- Every Week
- Best Instructors
- Most Important Topics
- From Engineers to Engineers
- Absolutely Free

DATASTAX®
DataStax Monday Learning

Docker Containers

From Basics to Best Practices

5-weeks Learning Path: 28.09-23.10.2020

Speakers:

- Aleks Volochnev
- Developer Advocates of DataStax

Schedule:

- **Week I** **28.09.2020** Docker Fundamentals I
- **Week II** **05.10.2020** Docker Fundamentals II
- **Week III** **12.10.2020** Application Development with Docker
- **Week IV** **19.10.2020** Best Practices
- **Week V** **26.10.2020** **Introduction to Kubernetes** + Final Assignment

DATASTAX®
DataStax Monday Learning

Docker Containers

From Basics to Best Practices

- >1.300 registrations
- Over 6K views on Youtube
- Over 2,300 HOURS overall watch time

Thank you!

DATASTAX®
DataStax Monday Learning

Docker Containers
From Basics to Best Practices

Week IV

Docker Best Practices

3 Questions to know you better

Best Practices

Best Practices Goals

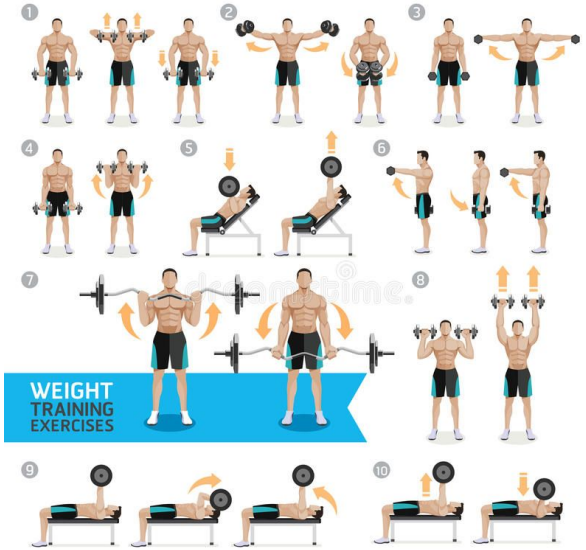
- Maintainability
- Security
- Error Prevention

Size does Matter

Help your Image to be Fit

Lightweight images are much easier to operate.

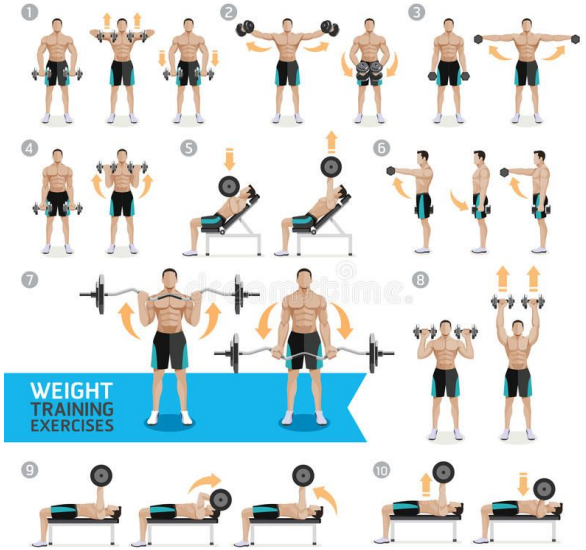
- Use minimal possible base image *-slim? -alpine? scratch?*
- Use multistage build to exclude build-time dependencies*



Help your Image to be Fit

Lightweight images are much easier to operate.

- Use COPY defensively, copy only what you need. Use .dockerignore.
- Install and Clean-up in one step*



Help your Image to be Fit

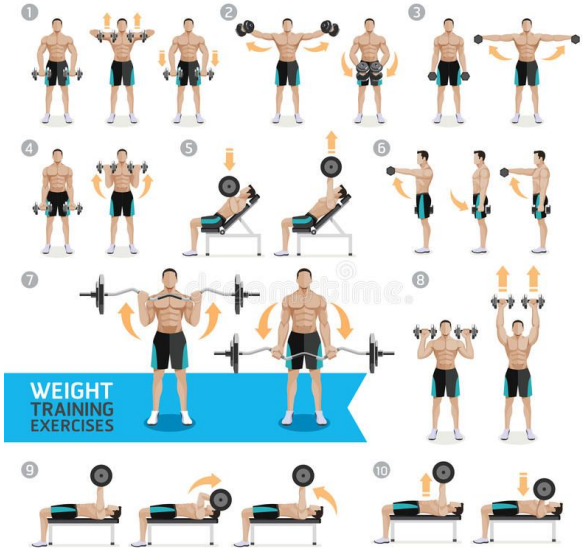
Lightweight images are much easier to operate.

- Watch number of layers, in most cases less is better *
  (RUN, COPY, ADD)
- *RUN apt install* needs version (caching, stability)

Help your Image to be Fit

Utilise cache!

- Copy as late as possible, as specific as possible
- Put less-frequently changed lines higher, so they don't break cache.



Redundant Layers

Every RUN, COPY and ADD will create a new layer. First example creates two layers while second only one.

```
FROM debian:9

RUN apt-get update
RUN apt-get install -y nginx
```

```
FROM debian:9

RUN apt-get update && \
    apt-get install -y nginx
```

Redundant Files

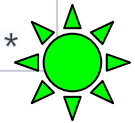
Remove all build-time dependencies!

```
FROM debian:9

RUN apt-get update && \
    apt-get install -y \
    [buildpackage]
RUN [build my app]
RUN apt-get autoremove --purge \
    -y [buildpackage] && \
    apt-get -y clean && \
    rm -rf /var/lib/apt/lists/*
```

```
FROM debian:9

RUN apt-get update && \
    apt-get install -y \
    [buildpackage] && \
    [build my app] && \
    apt-get autoremove --purge \
    -y [buildpackage] && \
    apt-get -y clean && \
    rm -rf /var/lib/apt/lists/*
```



Multistage Builds

Multistage Builds are great, especially for statically-linked compiled Applications.

```
FROM golang:1.10 as builder

WORKDIR /tmp/go
COPY hello.go ./
RUN CGO_ENABLED=0 go build -a -ldflags '-s' -o hello

FROM scratch
CMD [ "/hello" ]
COPY --from=builder /tmp/go/hello /hello
```

Labels

Labels are important! Label at least:

- Repository
- Maintainer / Team
- Build Number / Link
- Build Date
- Git Commit Hash
- Etc.

Stay Traceable, you aren't James Bond!

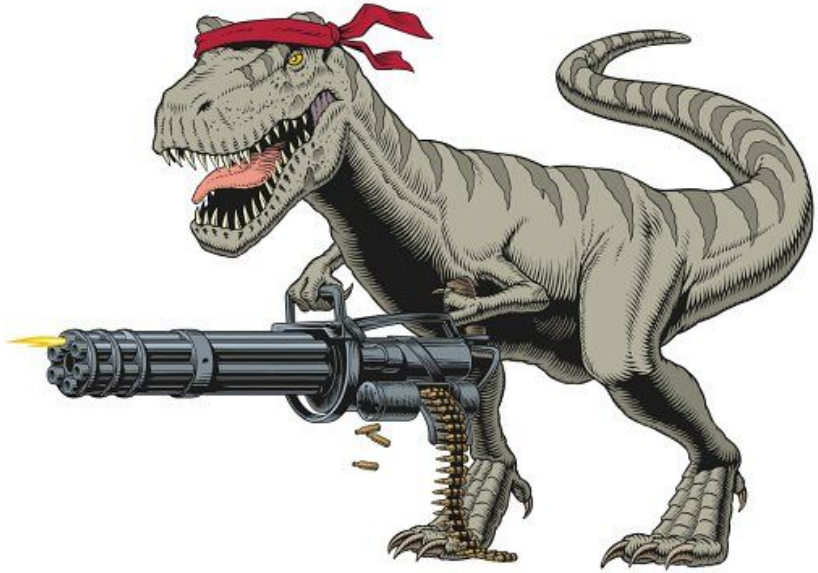
PID 1 Problem

PID 1 Problem

In Docker, the first process has additional responsibilities:

- System signals forwarding
- Pass back exit codes
- Reaps zombies

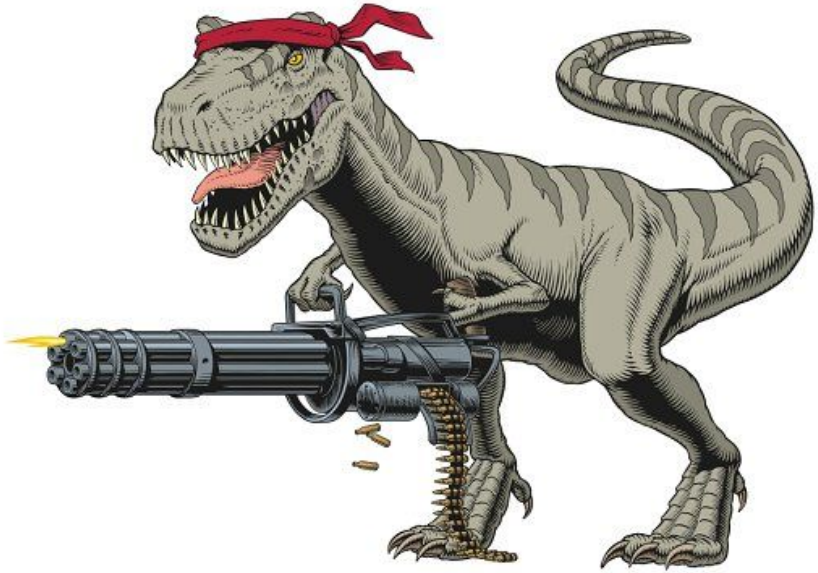
Most container entrypoint processes (like python, java) can't fulfill the PID 1 responsibilities properly. Bash too!



PID 1 Process reaps Zombies

PID 1 Problem

To address that, we have to install... nothing. Since docker version 1.13 ***tini*** is a built-in docker feature to answer all the PID 1 concerns.



PID 1 Process reaps Zombies

Multi-Process?

Multi-Process Containers?

As a general rule, try to run single-process containers. This helps keeps system less complex and usually easier to scale.

ONE CONTAINER
ONE RESPONSIBILITY
ONE PROCESS

Multi-Process Containers?

But in some cases, focus on a service is better than focus on a process. Good example will be a Nginx+PHP-FPM combination.

Notice, I still mean SMALL SERVICES.

ONE CONTAINER
ONE RESPONSIBILITY
~~**ONE PROCESS**~~
ONE SERVICE

Multi-Process Containers

That's important to not simply run processes in a container, it's more complex. You need a manager.

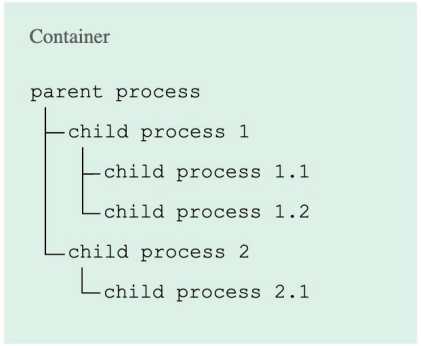
Remember PID 1 Problem? It's the same but now worse.

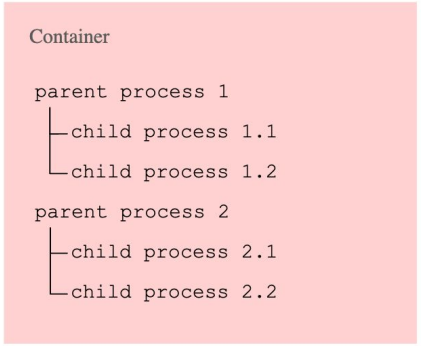
ONE CONTAINER
ONE RESPONSIBILITY
~~**ONE PROCESS**~~
ONE SERVICE

Multi-Process Containers

That's important to not simply run processes in a container, it's more complex. You need a manager.

Remember PID 1 Problem? It's the same but now worse.





Multi-Process Init

- Run multiple child processes, but do not restart them
- Exit as soon as a child process terminates
- Fulfill PID 1 responsibilities

- ~~Supervisord~~
- ~~Runit~~
- ~~Monit~~
- ~~Tini~~
- S6

S6

So you want to do multi-process containers? Ensure you understand:

- PID 1 Responsibilities
- INIT Process Responsibilities

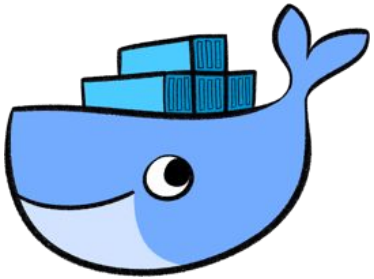
At least read https://github.com/just-containers/s6-overlay first!

- ~~Supervisord~~
- ~~Runit~~
- ~~Monit~~
- ~~Tini~~
- S6

Base Images

Image Rules are simple

- Use official Images
- Be careful with the :latest image, prefer tags
- Custom Base Image?
  - Rebuild regularly!
  - Rebuild automatically!
- Use Semantic Versioning (like in Git)
- Understand mutable and immutable Tags
  - *:12.3.4*
  - *:12.3*
  - *:12*
  - *:latest*



LOGS

Treat Logs Properly

“Treat logs as event streams”

The rule is simple: all the logs go to */dev/stdout*, error logs got to */dev/stderr*.

It’s not a best practice but more a law!

Set the Limits

Set the Limits

- Non-Root User
- Limit Memory and Swap
- Use :ro read-only volumes/mounts
- Bash strict mode `set -euo pipefail`
- Avoid --privileged, use --cap-add instead

Healthcheck

Healthcheck

```
1  FROM node
2
3  COPY server.js /
4
5  EXPOSE 8080 8081
6
7  HEALTHCHECK --interval=5s --timeout=10s --retries=3 CMD curl -sS 127.0.0.1:8080 || exit 1
8
9  CMD [ "node", "/server.js" ]
```

Great feature. Available since 2016 but still not in so wide use! :(

Healthcheck with docker-compose

```
version: '3.1'
services:
   web:
      image: docker-flask
      ports:
         - '5000:5000'
      healthcheck:
         test: curl --fail -s http://localhost:5000/ || exit 1
         interval: 1m30s
         timeout: 10s
         retries: 3
```

Linter

Dockerfile Linter

As with source code, it's a great strategy to lint Dockerfiles if you want to stay safe.

Such linters as will help!

- hadolint
- dockerlinter

https://hadolint.github.io/hadolint/

```
   Always tag the version of an image explicitly
1  FROM debian
   node_verion is referenced but not assigned (did you mean 'node_version'?).
   Delete the apt-get lists after installing something
   Avoid additional packages by specifying `--no-install-recommends`
2  RUN export node_version="0.10" \
3  && apt-get update && apt-get -y install nodejs="$node_verion"
4  COPY package.json usr/src/app
   Use WORKDIR to switch to a directory
   Pin versions in npm. Instead of `npm install <package>` use `npm install <package>@<version>`
5  RUN cd /usr/src/app \
6  && npm install node-static
7
   Valid UNIX ports range from 0 to 65535
8  EXPOSE 80000
9  CMD ["npm", "start"]
```

LIVE QUIZ!

Week IV Assignment

Week IV Assignment

- Optimize one of the dockerfiles of your choice. You can pick one of the suggested or find something on your own. Try to make it smaller, exclude redundant dependencies while keeping dockerfile self-sufficient. Think not only about size but caching and build time as well. For multi-process containers, you may need docker-compose to separate them.
  - Java
  - NodeJS
  - Python

- Implement automated build for a project from p.I or last week using Travis CI, Github Actions or another Build Server of your choice. Build Pipeline should include:
  - docker build
  - testing (at least smoke tests)
  - docker push to hub.docker.com

Resources:

- https://github.com/datastaxdevs/docker-learning-path
- https://discord.gg/va4vnsm

Thank You!

You are awesome!