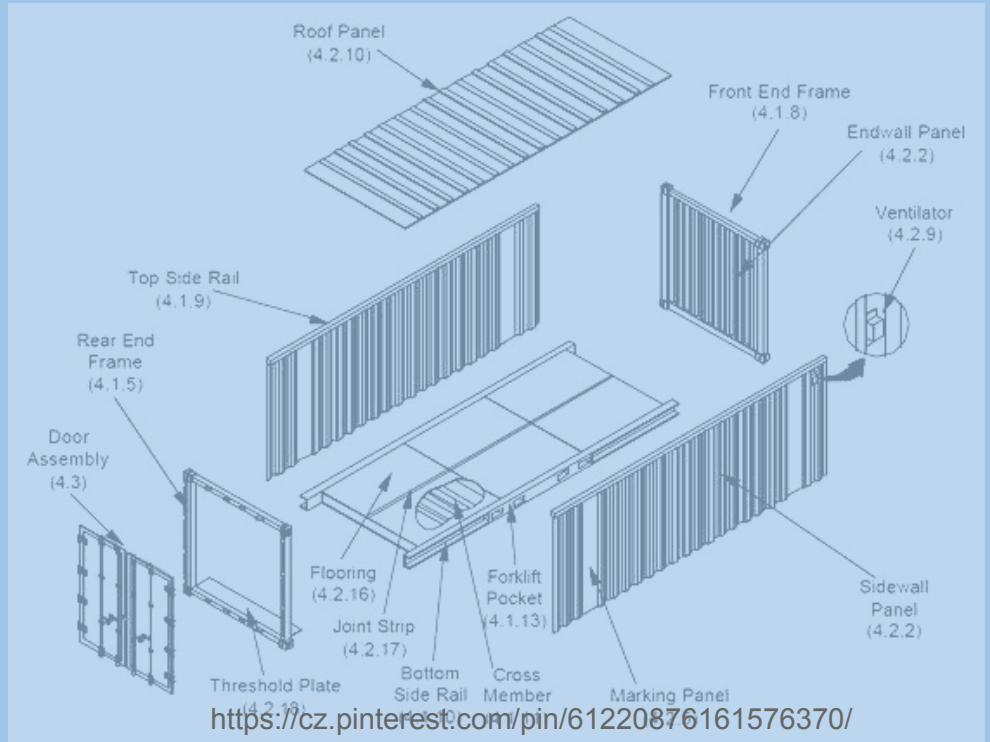
best practices for...

Docker image **building**

Etnetera a.s.
@jhrcz



https://cz.pinterest.com/pin/612208761615763 70/

Documentation as a start point...

docker-inc

- https://docs.docker.com/engine/reference/builder/
- https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices

atomic

- https://github.com/projectatomic/container-best-practices

openshift

- https://docs.openshift.com/enterprise/3.0/creating_images/guidelines.html

.dockerignore

docker build is sending build context (content of the dir) ti building process

sometime it does not make sense to send all the fluf from the repo to d-b

makefile strikes again

```
NAME=php55fpm-devtools
BUILD_IMG=build_$(NAME)
REGISTRY_IMG=mynamespace/$(NAME)
REGISTRY=docker-registry.local:5000

all: build tag push

build:
	sudo docker build --build-arg=http_proxy=$$http_proxy --build-arg=https_proxy=$$https_proxy -t $(BUILD_IMG) .

build-nocache:
	sudo docker build --build-arg=http_proxy=$$http_proxy --build-arg=https_proxy=$$https_proxy --no-cache -t $(BUILD_IMG) .

tag:
	sudo docker tag $(BUILD_IMG) $(REGISTRY)/$(REGISTRY_IMG)

push: tag
	sudo docker push $(REGISTRY)/$(REGISTRY_IMG)
	/srv/docker-BUILD/notify.sh "$(REGISTRY)/$(REGISTRY_IMG)"
```

Ordering layers, squashing

1. logical layer: base
2. logical layer: app
3. logical layer: updates

the point is … during rebuilds, transfer just layers that are smaller

testing is important

reusing caches

composer, npm, gradle, maven… these all have some sort of local caching

mount cache dirs as persistent volumes on app rebuilds … this could make builds faster… i mean FASTER

scripts for complex build instructions

it’s more useful when using build scripts injected into docker build environmen

- build.sh
- Makefile

… what suits your needs

be compatible in build scripts with your envs

env http_proxy

vs.

```
sudo docker build \
    --build-arg=http_proxy=$$http_proxy --build-arg=https_proxy=$$https_proxy \
    -t $(BUILD_IMG) .
```

entrypoint syntax with […] vs. PID 1

ENTRYPOINT ["/entrypoint.sh"]

CMD ["xxx", “yyy”]

… do not be confused by not evaluating variables in this “exec” form

ADD vs COPY

ADD

- knows how to download from web
- knows how to unpack archive

COPY

- just adds files

pkg installation during build vs yum/rpm

```
RUN \
    set -x; PKGS=" \
    centos-release-scl \
    ...
    " && \
    yum -y install $PKGS && \
    rpm -q $PKGS && \
    rm -rf /var/cache/yum
```

entrypoint for developer’s exec, PS1

```
ENV CONTAINER_IMAGE=phpfpm
ENV TZ=Europe/Prague
RUN echo 'export PS1="[\u@\h:\$CONTAINER_IMAGE:\$APP_NAME] \W # "' \
        >> /etc/bashrc



exec sudo docker $cmd \
        --user "$dc_username" -ti \
        "$container" \
        "$entrypoint" $dcmd sh -c "export TERM=xterm;  exec ${@:-bash}"
```

RO user a RW user

not running services in container as root is normal, right?

what about readonly vs readwrite perms to code and data by services?

```
RUN adduser -u 1001 -U -d /app app-rw
USER app
```

on-build instructions

[...]
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
[...]

dead or alive

- in good old days there were only one way to detect state of container … process was running on not running
- HEALTCHECK (since 1.12)

```
HEALTHCHECK --interval=5m --timeout=3s \
  CMD curl -f http://localhost/ || exit 1
```

Q/A