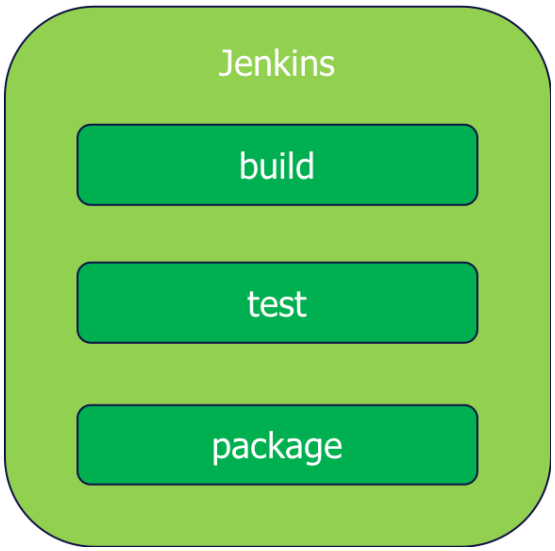
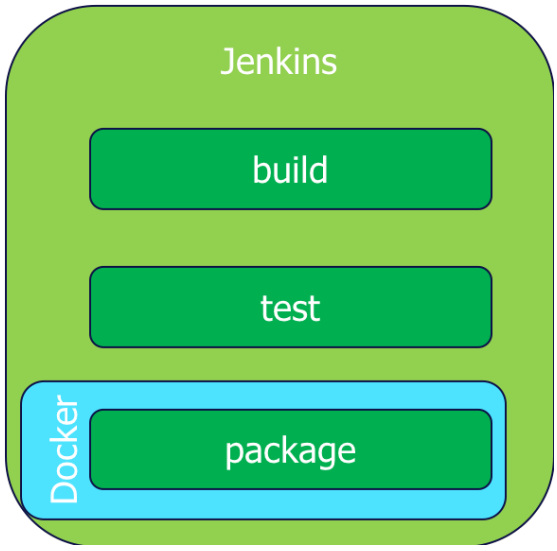
Lesson 5. Docker and development pipeline

Pipeline options

"Try docker"

From	To
	

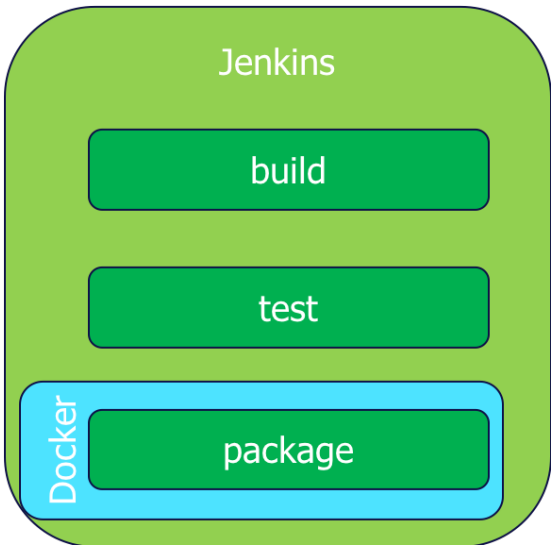
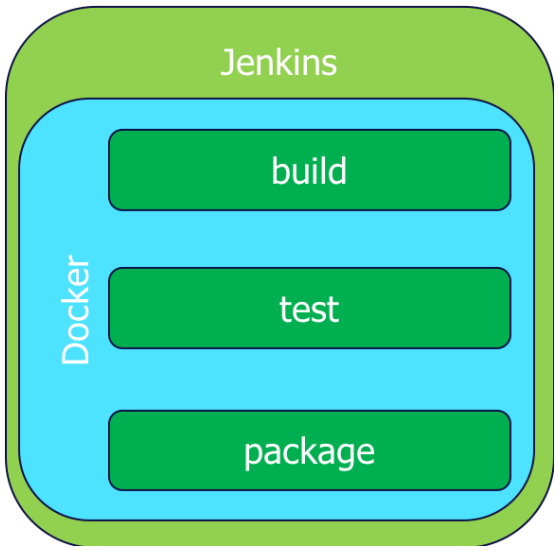
Pros:

- Application portability

Cons:

- Unreproducible environment (dev, test etc.)
- Configuration hell (envs, settings etc.)

"Fat image"

From	To
	

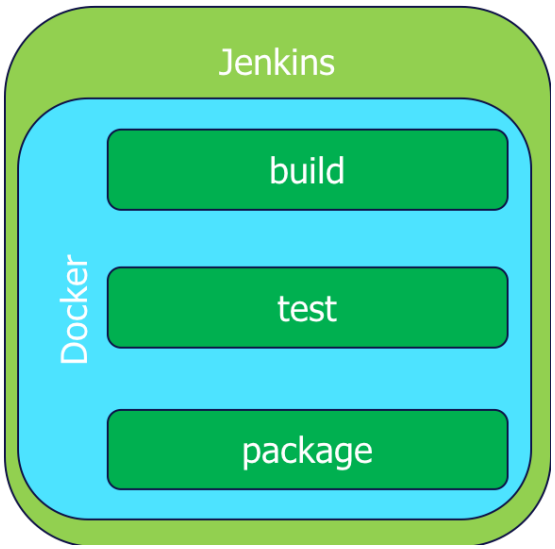
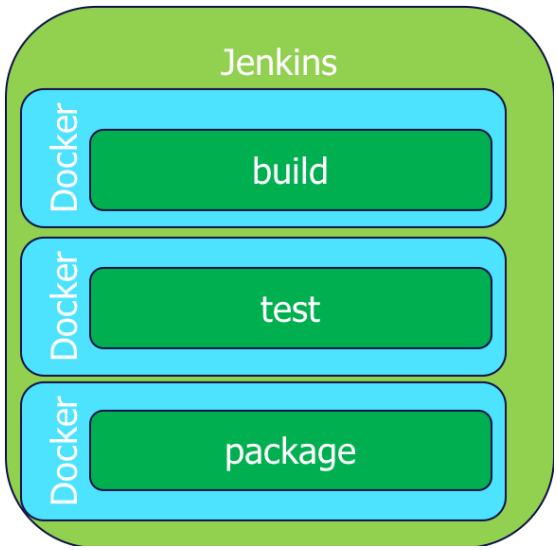
Pros:

- Repeatable environment
- Less configuration logic (once described in Dockerfile)
- Predictable and reproducible tests

Cons:

- Huge image size (testing tools, required packages, test scripts, and perhaps even test data)
- Unneeded dependencies in production (also configurations, logs, runtime test deps)

"Elegant Docker"

From	To
	

Pros:

- Repeatable environment
- Less configuration logic
- Image single responsibility (app and tests are separated as particular images)

Cons:

- More management of images and containers

Pipeline implementations

Multistage builds

Multistage builds introduce stages which allow reusing other images for a building current one.

```
FROM alpine as data
WORKDIR /src
RUN echo "This file is generated while another image is built." >> data.txt
RUN echo "The file was copied to this image using \"Multistage builds\" feature." >> data.txt
RUN echo "" >> data.txt
RUN echo "Inspect the layers of the base image:" >> data.txt
RUN echo "    docker history busybox" >> data.txt
RUN echo "Inspect the layers of current image:" >> data.txt
RUN echo "    docker history magic" >> data.txt
RUN echo "" >> data.txt
RUN echo "As you may see, there are no layers from \"alpine\" image. Run:" >> data.txt
RUN echo "    docker history alpine" >> data.txt

FROM busybox
COPY --from=data /src/data.txt message
#COPY --from=0 /src/data.txt message
CMD cat message
```

```
docker build -t magic .
```

> Read more on https://docs.docker.com/engine/userguide/eng-image/multistage-build

A container as a worker

A worker container allows execution of your pipeline (or its part) on existing sources.

Sample command:

```
docker run -t --rm --volume $PWD:/code --workdir /code extsoft/elegant-git-ci:1 ./run-tests
```

https://github.com/bees-hive/elegant-git uses container for the execution of unit tests. Check out https://travis-ci.org/bees-hive/elegant-git/jobs/404216705#L359 to see real job's execution.

Pipeline best practices

Don’t store data in containers - use volumes

```
# other instructions
VOLUME /data
# other instructions
```

```
docker run --volume $PWD:/data my-image
```

Use your own base images

Reasons:

- security!!! - you control it
- speed up a pipeline

```
# Dockerfile.base
FROM java:8-jdk
LABEL mainteiner="Dmytro Serdiuk <dmytro.serdiuk@gmai.com>" \
      description="This image is used to ...."
# Update packages and install dependencies
RUN apt-get update && apt-get upgrade -y && apt-get dist-upgrade -y
```

```
docker build --tag my-base:1.0.0 -f Dockerfile.base .
```

```
# Dockerfile
FROM my-base:1.0.0
# other instructions
```

Use `USER`

By default docker containers run as root. A docker container running as root has full control of the host system.

Your image should use the USER instruction to specify a non-root user for containers to run as.

```
RUN groupadd -r swuser -g 433 && \
    useradd -u 431 -r -g swuser -d <homedir> -s /sbin/nologin -c "Docker image user" swuser && \
    chown -R swuser:swuser <homedir>
USER swuser
# other instructions
USER root
# root instructions
USER swuser
# other instructions
```

Always `exec` in wrapper scripts

While you use `exec`, the application becomes `PID 1` and it can receive Linux signals.

```
# entry.sh
#!/bin/sh
echo "Do some stuffs..."
exec python app.py
```

```
# other instructions
ENTRYPOINT ["./entry.sh"]
```

Take care about "zombie" processes

The following will care about all "zombie" processes

```
docker run --init ....
```

or

```
# other instructions
ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
ENTRYPOINT ["/tini", "--"]

# Run your program under Tini
CMD ["/your/program", "-and", "-its", "arguments"]
```

> Read more on https://github.com/krallin/tini.

Static analysis of `Dockerfile`s

```
docker run --rm -i hadolint/hadolint < Dockerfile
# or
docker run --rm -i hadolint/hadolint hadolint --ignore DL3009 - < Dockerfile
# or
cat Dockerfile | docker run --rm -i hadolint/hadolint hadolint --ignore DL3009 -
```

> Read more on https://github.com/hadolint/hadolint

Homeworks

Please send the results of homeworks as an email.
Please use the following template:

- *Subject:* `[Docker] Homework 5`
- *To:* `trainer's email`
- *Body:* `your homework as a plain text - NO ATTACHMENTS!!!`

Homework 5.1 (mandatory)

The `home-work-5.zip` contains a project where is a *"Fat image"* approach implemented. You need to update `Dockerfile` with multistage builds and other best practices to provide an "Elegant Docker" solution.

Please send updated `Dockerfile` for review as well as a command to run your image (use `jat:h5` suffix for the image name).

Homework 5.2 (optional)

The `home-work-5.zip` contains a project where is a "Fat image" approach implemented. You need to use *"A container as a worker"* approach
to reproduce pipeline described in the `Dockerfile`.

Please send for review a sequence of the commands which solve the task.