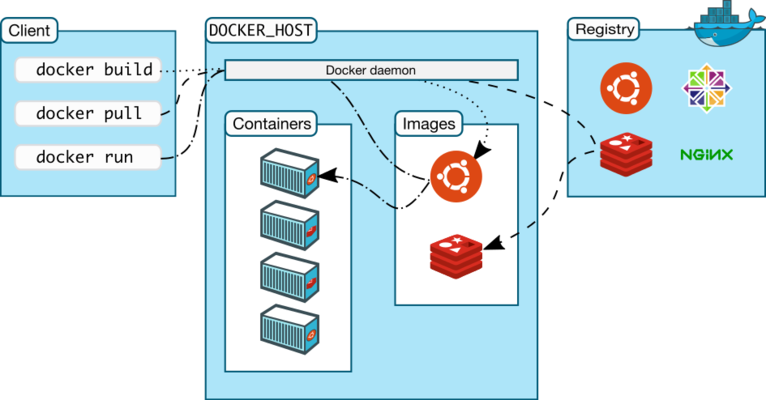
**What is Docker?**

* World before Docker
  + no virtualization
  + hypervisor-based virtualization
  + container-based virtualization

**Architecture**

* Docker uses a client-server architecture

[](https://training-course-material.com/training/File:Architecture.svg)

**Main Docker elements**

* **deamon**: process that runs on a host machine (server)
* **client**: primary Docker interface
* **image**: read-only template (build component)
* **registry**: public or private image repositories (distribution, ship component)
* **container**: created form image, holds everything that is needed for an application to run (run component)

**Benefits of Docker**

* separation of roles and concerns
  + developers focuses on building applications
  + system administrators focuses on deployment
* portability: build in one environment, distributed and run on many others
* faster development, testing, deployment
* scalability: easy to spin up new containers or migrate to more powerful hosts
* better resource utilization: more apps on one host

**The underlying technology**

* **namespaces**
  + **pid** namespace: used for process isolation (Process ID)
  + **net** namespace: used for managing network interfaces
  + **mnt** namespace: used for managing mount-points
  + **ipc** namespace: used for managing access to IPC resources (InterProcess Communication)
  + **uts** namespace: used for isolating kernel and version identifiers (Unix Timesharing System)
* **control groups** (cgroups)
  + used for sharing available hardware resources
  + and setting up limits and constraints
* **union file system** (UnionFS)
  + file system that operate by creating layers
  + many layers are merged and visible as one consistent file system
  + many available file systems: **AUFS**, btrfs, vfs, DeviceMapper
* **container format**
  + two supported container formats: **libcontainer**, LXC

**Getting started**

**Installation of Docker engine and client**

# easiest way to install Docker

$ wget -qO- https://get.docker.com/ | sh

# to use docker as ubuntu user without using sudo (optional)

$ sudo usermod -aG docker ubuntu

# to check the installation

$ docker --version

* [Manual installation steps, more details and installation instructions for other operating systems](https://docs.docker.com/engine/installation/ubuntulinux/)

**Hello world example**

$ docker run hello-world

$ docker images

$ docker ps -a

**Dockerized bash terminal**

$ docker run -it ubuntu

$ docker run -it ubuntu:latest

$ docker run -it ubuntu:14.04 bash

$ docker run -it ubuntu ps -aux

* docker run **-t**: allocate a pseudo-tty
* docker run **-i** (--interactive): keep STDIN open even if not attached
* use **CTRL + p + q** to detach from running container
* use **attach** command to reattach to a detached container

$ docker attach container\_name

* the importance of PID 1
* PID in the container and Docker host:

$ ps -fe | grep $(pidof docker)

* installing packages: mc, vim

**Investigating containers and images**

* docker **inspect** displays low-level information on a container or image

$ docker inspect webapp

$ docker inspect --format='{{.NetworkSettings.IPAddress}}' webapp

$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' webapp

* docker **diff** displays changes on a container's filesystem

$ docker diff webapp

$ docker diff webapp | grep ^A

* docker **history** shows the history of an image (layers)

$ docker history ubuntu

$ docker history --no-trunc ubuntu

* docker **logs** fetches the logs of a container

$ docker logs webapp

$ docker logs --tail 15 webapp

$ docker logs -f webapp

$ docker logs -t webapp

$ docker logs --since 1h webapp

* docker **top** displays all running processes of a container

$ docker top webapp

$ docker exec webapp ps aux

**Cleaning up and keeping clean**

$ docker images

$ docker ps -a

* docker run **--rm** automatically removes the container when it exits

$ docker run --rm -it ubuntu bash

* docker run **--name** assigns your own, meaningful name of the container

$ docker run --rm -it --name=test ubuntu bash

* container name must be unique and you can change it whenever you like

$ docker rename test meaningful\_name

* docker **rm** removes a container
* docker **rmi** removes an image

$ docker rm meaningful\_name

# removes all not running containers

$ docker rm $(docker ps -a | grep 'Exited' | awk '{print $1}')

# removes all images without a tag

$ docker rmi $(docker images | grep '^<none>' | awk '{print $3}')

*Exercises*

**Building images**

There are two ways of building images:

* build an image based on existing container
* build an image from a Dockerfile

**Committing changes**

$ docker run -it --name midnight ubuntu

# inside of the container run: apt-get update && apt-get install -y mc

$ docker diff midnight

$ docker commit -p -m "added mc" midnight training:mc

*Exercises*

**Building an image from a Dockerfile**

* Create a new directory and "Dockerfile" text file
* After that run the command below to build an image

$ docker build [options] PATH | URL | -

$ docker build -t training:mc3 .

* build context
  + the **PATH** is a directory on your local filesystem.
  + the **URL** is a the location of a Git repository.
* the build is run by the Docker daemon, not by the Client
  + .dockerignore
* some selected options:
  + **-t** - name and optionally a tag in the repository/name:tag format
  + **-f** or **--file** - name of the Dockerfile (default: PATH/Dockerfile)
  + **--no-cache** - do not use cache when building the image
  + **--pull** - always attempt to pull a newer version of the base image

**FROM**

* sets the base image for subsequent instructions
* **from** must be the first instruction in Dockerfile

FROM <image>

FROM <image>:<tag>

FROM <image>@<digest>

FROM ubuntu:trusty

**COPY and ADD**

* the **COPY** instruction copies files or directories from <src> to the filesystem of the container at the path <dest>
* <src> path must be inside the context of the build
* if <src> is a directory, the entire contents of the directory are copied but the directory itself is not copied
* slash after the directory name is important
* if <dest> doesn’t exist, it is created along with all missing directories in its path
* **COPY** has two forms:
  + second one is required for paths containing whitespace

COPY <src> <src2>... <dest>

COPY ["<src>", "<src2>",... "<dest>"]

* **ADD** can copy local files like **COPY** but has some extra features like local-only tar extraction and remote URL support

COPY file1 file1

COPY fil\* /

COPY dir1/ /dest\_dir/

COPY file1 dir1/ /dest\_dir/

COPY ["file 1", "dir 1", "/dest/dir/"]

*Exercises*

**RUN**

* RUN instruction will execute any commands in a new layer on top of the current image and commit the results
* newly created image will be used for the next step in the Dockerfile
* RUN has 2 forms:
  + **shell form** - the command is run in a shell (/bin/sh -c)
  + use a \ (backslash) to continue a single run instruction onto the next line
  + use a && (double ampersand) to combine many commands in one layer

RUN <command>

RUN apt-get update

RUN apt-get install -y mc

RUN apt-get update && apt-get install -y mc

* + **exec form** makes it possible to avoid shell string munging
  + makes it possible to run commands using a base image that does not contain /bin/sh

RUN ["executable", "param1", "param2"]

RUN ["apt-get", "update"]

RUN ["apt-get", "update", "-y", "mc"]

**Layering RUN instructions and cache**

* one of key concepts of Docker: **commits are cheap**
* containers can be created from any point in an image’s history (docker images -a)
* cache and commands like apt-get update

*Exercises*

**CMD**

* the main purpose of a **CMD** is to provide defaults for an executing container
* for example */bin/bash* is default CMD for Ubuntu official image
* only the last CMD will take effect
* **CMD** has two forms:
  + **exec form**, this is the preferred form
  + skip executable to define default parameters to ENTRYPOINT
  + does not invoke a command shell, so the variable substitution is not going to happen

CMD ["executable","param1","param2"]

CMD ["ping", "nobleprog.pl", "-c", "3"]

CMD ["param1","param2"]

CMD ["nobleprog.pl", "-c", "3"] # will work if ENTRYPOINT is set to ping

* + **shell form**
  + command is exectuted in */bin/sh -c*

CMD command param1 param2

CMD ping nobleprog.pl -c 3

**ENTRYPOINT**

* allows you to configure a container that will run as an executable
* only the last ENTRYPOINT will take effect
* command line arguments to docker run will be appended after all elements in an exec form ENTRYPOINT, and will override all elements specified using CMD
* use docker run --entrypoint to override image ENTRYPOINT
* **ENTRYPOINT** has two forms:
  + **exec form**, this is the preferred form
  + this form allows to gracefully shut down using docker stop command

ENTRYPOINT ["executable", "param1", "param2"]

ENTRYPOINT ["ping", "-c", "5"]

CMD ["localhost"]

* + **shell form**
  + shell form prevents any CMD or run command line arguments from being used
  + ENTRYPOINT will be started as a subcommand of /bin/sh -c (which does not pass signals)

ENTRYPOINT command param1 param2

ENTRYPOINT ping -c 5 localhost

*Exercises*

**ENV**

* sets the environment variable <key> to the value <value>
* **ENV** has two forms:
  + first form - entire string after the first space will be treated as the <value> (including characters such as spaces and quotes)
  + second form (uses equal sign) - allows for multiple variables to be set at one time (quotes and backslashes can be used)

ENV <key> <value>

ENV <key>=<value> ...

ENV APACHE\_LOCK\_DIR /var/lock/apache2

ENV APACHE\_PID\_FILE="/var/run/apache2/apache2.pid" APACHE\_RUN\_USER="www-data"

ENV APACHE\_RUN\_GROUP="www-data" \

APACHE\_LOG\_DIR="/var/log/apache2/"

**EXPOSE**

* informs Docker that the container listens on the specified network ports at runtime
* it does not make the ports of the container accessible to the host
  + use run **-p** flag to publish a range of ports
  + use run **-P** flag to publish all of the exposed ports

EXPOSE 80, 22

docker run -P image\_name

docker run -p 8080:80 -p 22:22 image\_name

*Exercises*

**USER**

* sets the user name or UID to use when running the image
  + and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile
* try to avoid installing or using sudo since it has unpredictable TTY and signal-forwarding behaviour
  + if you need functionality similar to sudo you should use **gosu** (<https://github.com/tianon/gosu>)

USER mongodb

**VOLUME**

* creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers
* *docker run* command initializes the newly created volume with any data that exists at the specified location within the base image

VOLUME ["/data"]

VOLUME ["/var/log"]

*Exercises*

**Other commands**

* **MAINTAINER** - set the author field of the generated images

MAINTAINER <name>

MAINTAINER "Kamil Baran" <kamil.baran@nobleporg.pl>

* **LABEL** - adds metadata to an image

LABEL <key>=<value> <key>=<value> <key>=<value> ...

LABEL version="1.0" description="Docker Training" company="NobleProg"

* **ARG** - defines a variable that users can pass at build-time to the builder
* **ONBUILD** - adds to the image a trigger instruction to be executed at a later time, when the image is used as the base for another build
* **WORKDIR** - sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile
* **STOPSIGNAL** - sets the system call signal that will be sent to the container to exit.

**Automated Docker builds**

* <https://github.com/KamilBaran/nobleprog_docker_training>
* <https://hub.docker.com/r/kamilbaran/nobleprog_docker_training/>

**gosu helper tool**

* Simple Go-based setuid+setgid+setgroups+exec
* <https://github.com/tianon/gosu>

RUN apt-get update \

&& apt-get install -y --no-install-recommends ca-certificates curl numactl \

&& rm -rf /var/lib/apt/lists/\* \

&& gpg --keyserver ha.pool.sks-keyservers.net --recv-keys B42F6819007F00F88E364FD4036A9C25BF357DD4 \

&& curl -o /usr/local/bin/gosu -SL "https://github.com/tianon/gosu/releases/download/1.6/gosu-$(dpkg --print-architecture)" \

&& curl -o /usr/local/bin/gosu.asc -SL "https://github.com/tianon/gosu/releases/download/1.6/gosu-$(dpkg --print-architecture).asc" \

&& gpg --verify /usr/local/bin/gosu.asc \

&& rm /usr/local/bin/gosu.asc \

&& chmod +x /usr/local/bin/gosu

#!/bin/bash

# exit immediately if a command exits with a non-zero status.

set -e

# if the first character in arguments is -

if [ "${1:0:1}" = '-' ]; then

# set all arguments to mongod process

set -- mongod "$@"

fi

# if the cmd is mongod

if [ "$1" = 'mongod' ]; then

# disable numa

numa='numactl --interleave=all'

if $numa true &> /dev/null; then

set -- $numa "$@"

fi

# run mongod as mongodb user

exec gosu mongodb "$@"

fi

# run any other than mongod process

exec "$@"

**Running more than one process in a container**

* <http://docs.docker.com/engine/admin/using_supervisord/>
* <http://supervisord.org>

FROM ubuntu:trusty

RUN apt-get update \

&& apt-get install -y openssh-server supervisor \

&& apt-get clean \

&& rm -rf /var/lib/apt/lists/\* \

&& mkdir -p /var/run/sshd /var/log/supervisor

COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf

EXPOSE 22

CMD ["/usr/bin/supervisord"]

[supervisord]

nodaemon=true

[program:ping1]

command=/bin/ping nobleprog.pl

[program:ping2]

command=/bin/ping nobleprog.co.uk

[program:ping3]

command=/bin/ping nobleprog.cn

[program:sshd]

command=/usr/sbin/sshd -D

*Exercises*

**General guidelines and recommendations**

* Containers should be ephemeral
* Use a .dockerignore file
* Avoid installing unnecessary packages
* Run only one process per container
* Minimize the number of layers
* Sort multi-line arguments
* Build cache

*Exercises*

**Storage and data persistence**

**Within the container**

* data is only visible inside the container
* data is not persisted outside of the container
* data is lost if the container is removed

**Directly on Docker host**

$ docker run -v /host/dir:/container/dir:rw ...

$ docker run -v /home/ubuntu/docker/training\_httpd1/html:/var/www/html/ -d --name www --net host training:httpd1

$ docker run -v $PWD/html:/var/www/html/ -d --name www --net host training:httpd1

$ docker run -v $PWD/html:/var/www/html/:ro -d --name www --net host training:httpd1

* data is visible inside the container, Docker host and can be shared between containers
* data is persisted outside of the container even if the container is removed
* this provides near bare metal performance
* host directory can be an existing NFS share, formatted block device or anything that can be mounted on Docker host

**Outside Docker’s UnionFS**

$ docker run -itd -v /data --name data1 ubuntu

$ docker inspect data1

$ docker exec data1 touch /data/file1

$ docker exec data1 ls -l /data/

$ docker run -itd --volumes-from data1 --name data2 ubuntu

$ docker inspect data2

$ docker rm -fv data1 data2

$ docker volume create --name kb\_volume

$ docker run --rm -it -v kb\_volume:/data ubuntu touch /data/kb

$ docker run --rm -it -v kb\_volume:/data ubuntu ls -l /data

$ docker volume rm kb\_volume

$ docker volume ls

* data is visible inside the container and can be shared between containers
* data is persisted outside of the container even if the container is removed
  + use docker run **rm -v** to remove a container with its volumes (unless the other container uses them)
* this provides near bare metal performance
* it solves the problem with privileges (users and groups with different IDs on host and in the container)

**Creating groups and users with custom ID**

$ groupadd -r -g 27017 mongodb

$ useradd -r -u 27017 -g mongodb mongodb

**Backup and restore data from volumes**

# backup data from one container

$ docker run -itd -v /data --name data1 ubuntu

$ docker exec data1 touch /data/file1

$ docker exec data1 chown www-data:www-data /data/file1

$ docker run --rm --volumes-from data1 ubuntu ls -l /data

$ docker run --rm --volumes-from data1 -v $PWD:/backup ubuntu tar -cvpf /backup/backup.tar /data

$ docker rm -fv data1

# restore data into brand new container

$ docker run -itd -v /data --name data1 ubuntu

$ docker run --rm --volumes-from data1 -v $PWD:/backup ubuntu tar -xvpf /backup/backup.tar

$ docker run --rm --volumes-from data1 -v $PWD:/backup ubuntu bash -c "cd /data && tar -xvf /backup/backup.tar --strip 1"

$ docker run --rm --volumes-from data1 ubuntu ls -l /data

**Outside Docker Host**

**Networking**

$ docker network --help

$ docker network ls

**Docker host network**

* the **host network** adds a container on the hosts network stack
* the network configuration inside the container is identical to the host

$ docker run --name db1 -d --net host training:mongod

$ docker run --name www -d --net host training:httpd1

$ docker inspect www

$ docker network inspect host

**Without network interface**

* the **none network** adds a container to a container-specific network stack
* use *docker exec* command to connect to the container

$ docker run --name networkless --net none -it --rm ubuntu bash

**Default network (bridge)**

$ docker run --name db2 -d --volumes-from db1 training:mongod

$ docker run --name www -d training:httpd1

$ docker run --name www -d -P training:httpd1

$ docker run --name www -d -p 80:80 training:httpd1

$ docker run --name www -d -p 127.0.0.1:88:80 training:httpd1

$ docker run --name www -d -p 80:80 --link db2 training:httpd1

$ docker run --name www -d -p 80:80 --link db2:db training:httpd1

$ docker inspect www

$ docker network inspect host

* this is a default network for all containers
* containers are able to communicate with each other using IP addresses
* Docker does not support automatic service discovery on the default bridge network
* to communicate by using names in this network, you must connect the containers via the legacy docker **run --link** option

**Custom user networks (bridge)**

$ docker network create --driver bridge --subnet 10.1.2.0/24 net1

$ docker network create --subnet 10.1.2.0/24 --gateway=10.1.2.1 net1

$ docker run -d --net net1 --name mongo --net-alias db training:mongod

$ docker run -d --net net1 --name apache --net-alias www --hostname httpd -p 80:80 --env=db\_host=db training:httpd

$ docker network ls

$ docker network inspect net1

$ docker network disconnect net1 www

$ docker network connect net1 www

**Docker Compose**

**Installation**

$ sudo -i

$ curl -L https://github.com/docker/compose/releases/download/1.14.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

$ chmod +x /usr/local/bin/docker-compose

$ curl -L https://raw.githubusercontent.com/docker/compose/$(docker-compose version --short)/contrib/completion/bash/docker-compose > /etc/bash\_completion.d/docker-compose

$ exit

$ docker-compose --version

* more about the installation: <https://docs.docker.com/compose/install/>

**Example**

* create an empty directory called dc-app
* copy content form httpd (Dockerfile and html directory) and mongodb (Dockerfile) directories created previously
* create a docker-compose.yml file containing below configuration

version: '2'

services:

httpd:

build: ./httpd

image: training:httpd

network\_mode: host

mongodb:

build: ./mongodb

image: training:mongodb

network\_mode: host

* in order to build all images at once and start entire application run below commands

$ docker-compose build

$ docker-compose up -d

* more information about options available in docker-compose files: <https://docs.docker.com/compose/compose-file/>

**Docker Machine**

**Installation**

* <https://docs.docker.com/machine/install-machine/>

$ sudo -i

$ curl -L https://github.com/docker/machine/releases/download/v0.12.0/docker-machine-`uname -s`-`uname -m` > /usr/local/bin/docker-machine

$ chmod +x /usr/local/bin/docker-machine

$ curl -L https://raw.githubusercontent.com/docker/machine/master/contrib/completion/bash/docker-machine.bash > /etc/bash\_completion.d/docker-machine.bash

$ exit

$ docker-machine --version

**Amazon Web Services driver**

docker-machine create \

--driver amazonec2 \

--amazonec2-vpc-id vpc-6f559a0a \

--amazonec2-access-key AKI...EQA \

--amazonec2-secret-key A6Z...AKM \

--amazonec2-region eu-west-1 \

--amazonec2-zone b \

--amazonec2-instance-type t2.micro \

--amazonec2-ami ami-47a23a30 \

--amazonec2-root-size 8 \

--amazonec2-security-group myGroup \

aws-machine-1

**Generic driver**

Make sure you can connect, from your Docker Machine host (client) to *server-s1* (Docker host) using public key authentication

# on docker-machine-host generate keys, and transfer public key to docker-host

$ ssh-keygen

$ ssh-copy-id ubuntu@server-s1

# check the connection

$ ssh ubuntu@server-s1

You can change *server-s1* network configuration from DHCP into static IP (edit /etc/network/interfaces).

auto eth0

iface eth0 inet static

address 10.0.2.10

netmask 255.255.255.0

gateway 10.0.2.1

dns-nameservers 8.8.8.8

Make sure you are using passwordless sudo on *server-s1*

$ sudo chmod 640 /etc/sudoers

# replace: "%sudo ALL=(ALL:ALL) ALL" with: "%sudo ALL=(ALL) NOPASSWD:ALL"

$ sudo sed -i "s/%sudo\tALL=(ALL:ALL) ALL/%sudo ALL=(ALL) NOPASSWD:ALL/g" /etc/sudoers

$ sudo chmod 440 /etc/sudoers

Add new docker-host (server-s1) to Docker Machine

$ docker-machine create \

--driver generic \

--generic-ip-address server-s1 \

--generic-ssh-user ubuntu \

server-s1

$ docker-machine ls

**Docker Swarm mode**

* Docker Swarm is a native clustering tool that turns a group of Docker engines into a single, virtual Docker Engine.
  + When you run Docker Engine outside of swarm mode, you execute container commands.
  + When you run the Engine in swarm mode, you orchestrate services.
* Key futures
  + Cluster management integrated with Docker Engine
  + Declarative service model (scaling, desired state reconciliation, rolling updates)
  + Multi-host networking
* Requirements - 3 networked VM's
  + first one with docker-machine installed, all commands should be executed in terminal on this member
  + two others (server-s1, server-s2) with docker engine
* Create cluster manager

$ eval $(docker-machine env server-s1)

$ docker swarm init --advertise-addr 10.0.2.10

* Check cluster state and list of nodes

$ docker info

$ docker node ls

* Add another node to the cluster

$ docker swarm join-token worker

$ eval $(docker-machine env server-s2)

$ docker swarm join --token SWMTKN-1-3nt...sfe 10.0.2.10:2377

* Create, inspect and scale service called *training*

$ eval $(docker-machine env server-s1)

$ docker service ls

$ docker service create --replicas 1 --name training alpine ping nobleprog.pl

$ docker service inspect --pretty training

$ docker service ps training

$ docker ps

$ docker service scale training=5

* Rolling update of a service (from alpine:latest to alpine:3.4)

$ docker service update --update-delay 7s training

$ docker service update --image alpine:3.4 training

$ docker service inspect --pretty training

$ docker service ps training

* Ingress network and routing mesh

$ docker service create --name web-app --publish 80:80 --replicas 1 nginx

$ docker service ps web-app

$ curl 10.0.2.10

$ curl 10.0.2.20

* Removing a service and a node

$ docker service rm training

$ docker node rm server-s2