AARHUS UNIVERSITET

Microservices and DevOps

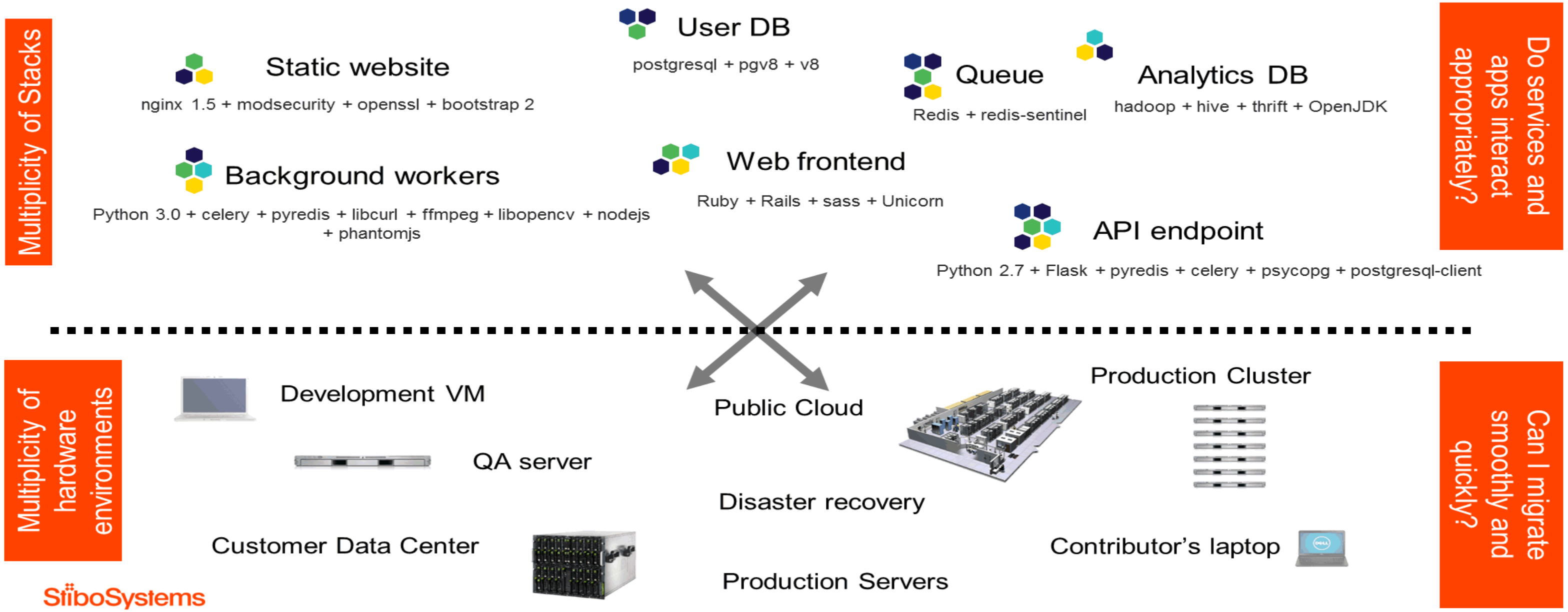
DevOps and Container Technology

Docker

Henrik Bærbak Christensen

The DevOps Problem

- Crossing boundaries, that is, *moving code*



Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center

Public Cloud

Disaster recovery

Production Servers

Production Cluster

Contributor's laptop

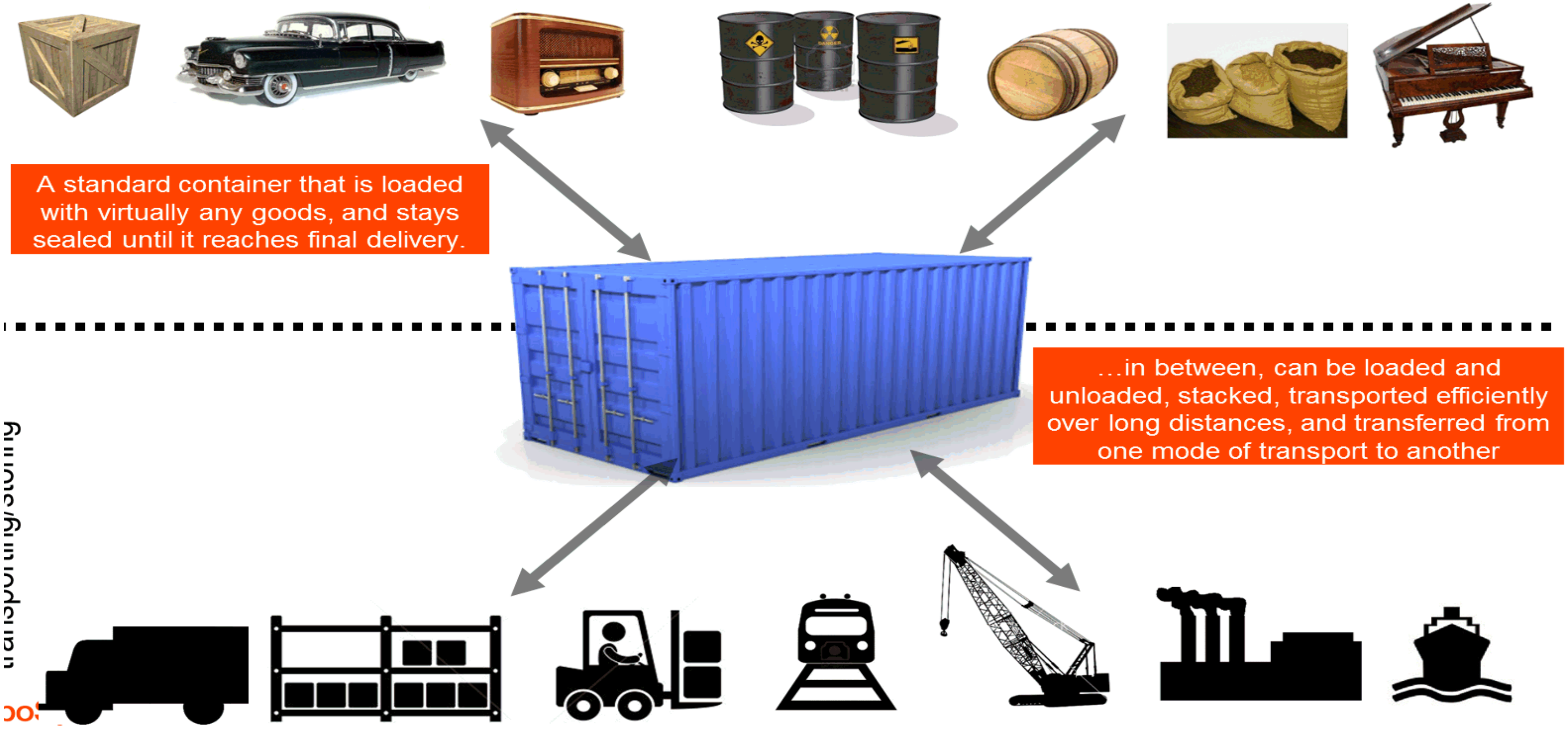
Can I migrate smoothly and quickly?

StiboSystems

Source: Torben Haagh, StiboSystems

AARHUS UNIVERSITET

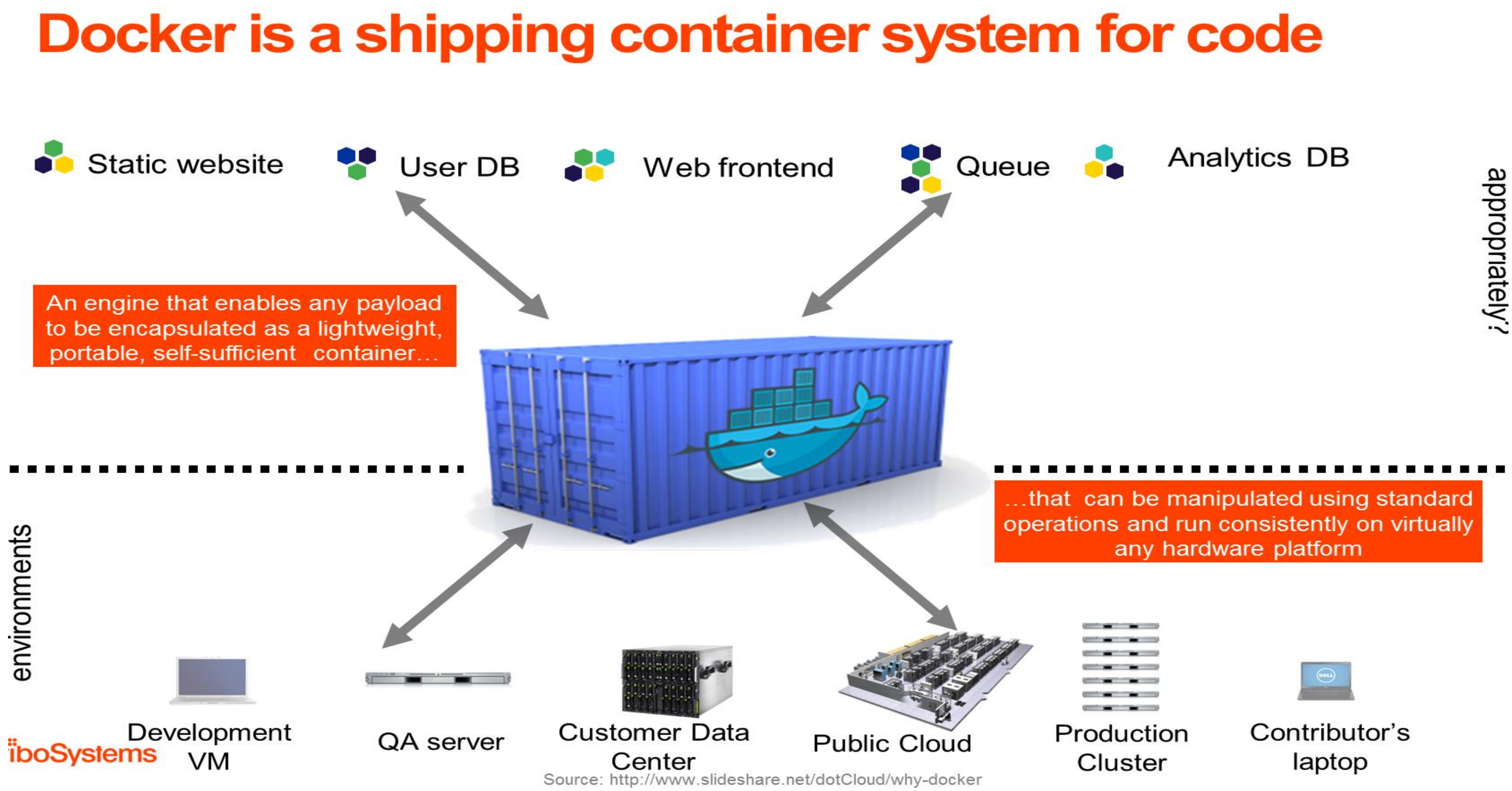
Was Solved in 1960'ies



A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Docker = Container



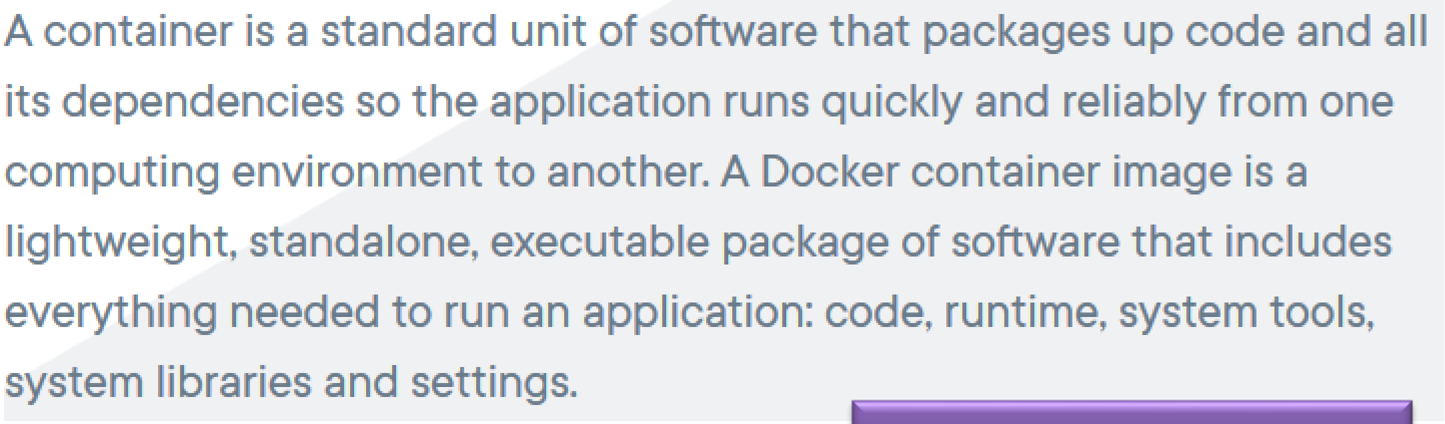
Docker is a shipping container system for code

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…

…that can be manipulated using standard operations and run consistently on virtually any hardware platform

Source: http://www.slideshare.net/dotCloud/why-docker

Definition



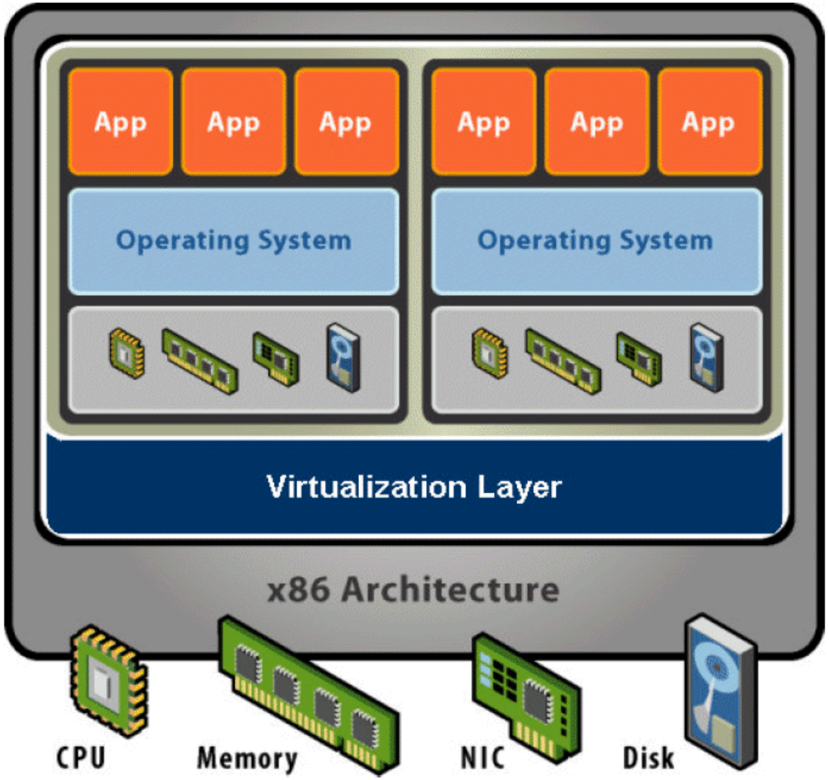
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

(Docker web site, 2019)

That is, not just program/code, but the required execution environment

Containers are Virtual Machines

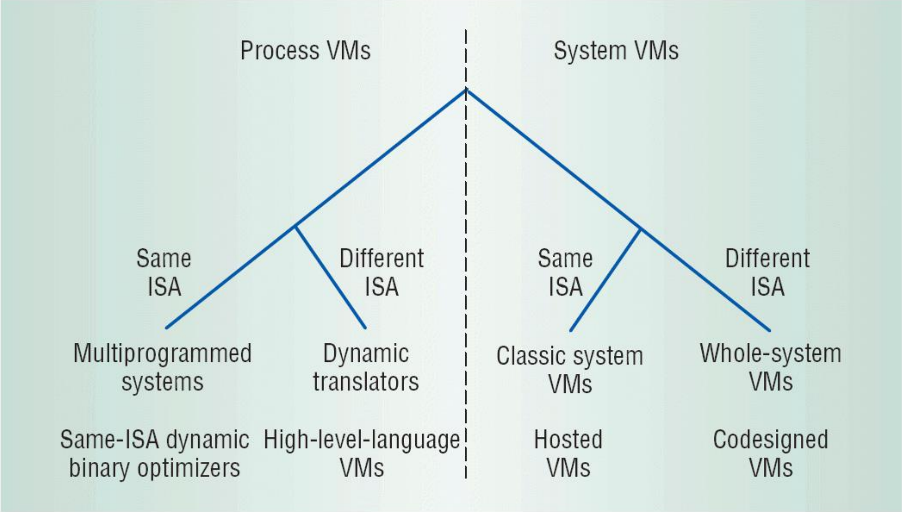
A Virtual Machine



- Hardware Abstraction
  - Virtual processor, memory, devices, etc.
- Virtualization Software
  - Indirection: Decouple hardware and OS
  - Multiplex physical hardware across guest VMs

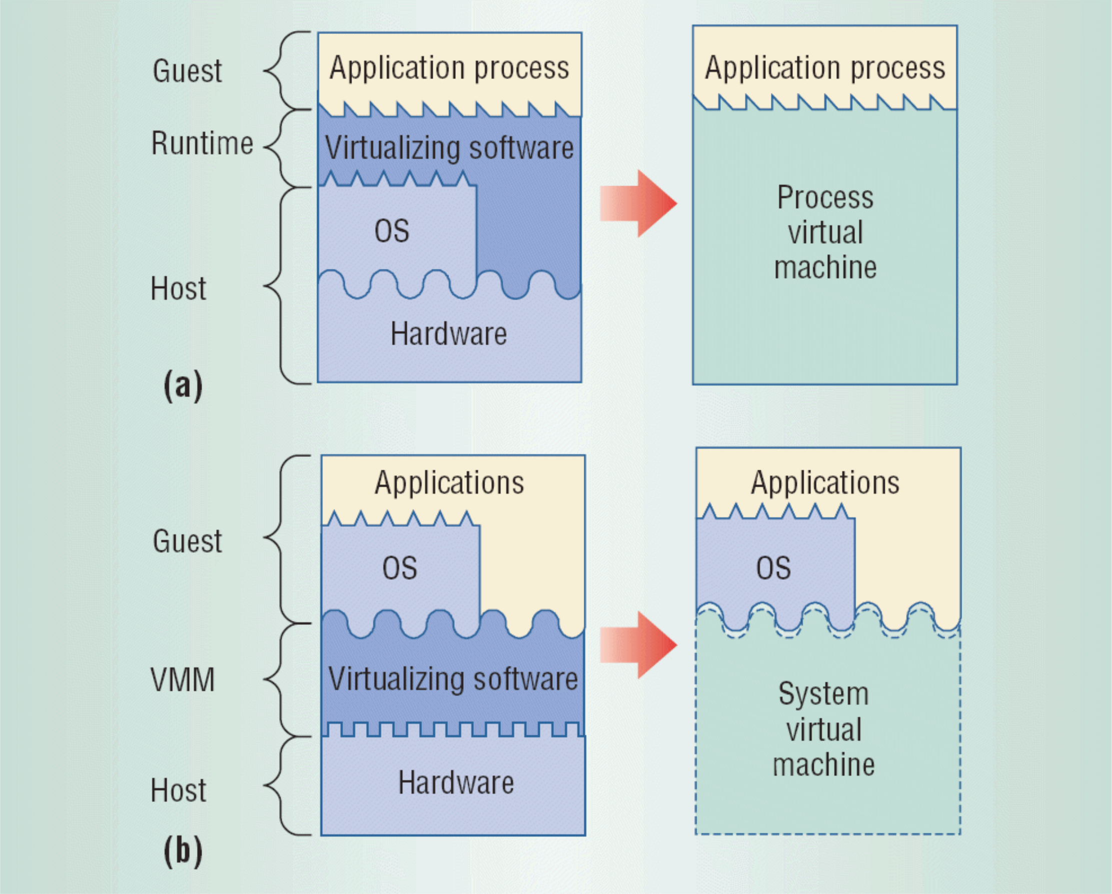
Types of VMs

- Smith & Nair 2005
- Two super classes
  - Process VM
  - System VM



- Both can be sub classed based upon supporting virtualization of *same* or *different* ISA (Instruction Set Architecture).

Process / System VM

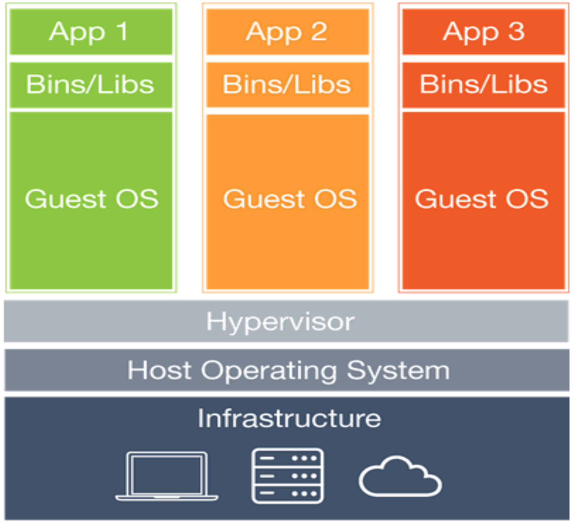


Ex: Java VM

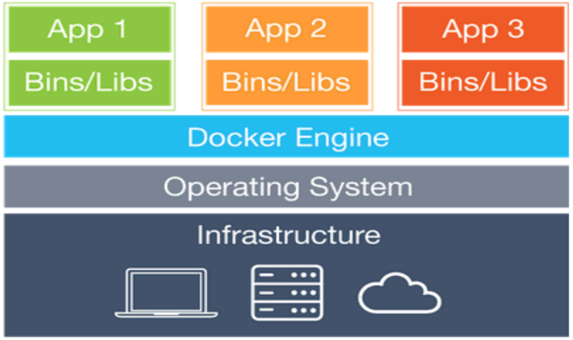
Ex: VMWare

Moving the boundary

- Mission: To make virtualization *lightweight*



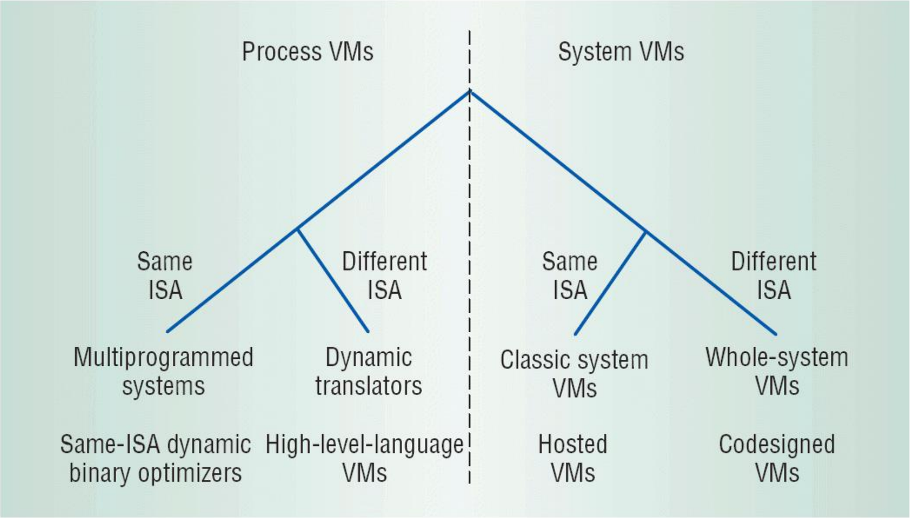
Traditional VMs



Docker

Classifying Docker

- *A Same-ISA Process VM*



- Docker containers provide a ***virtual Linux OS***
- Docker containers typically ***execute a single process***
- Ex: Run a RabbitMQ broker, an apache web server, …

AARHUS UNIVERSITET

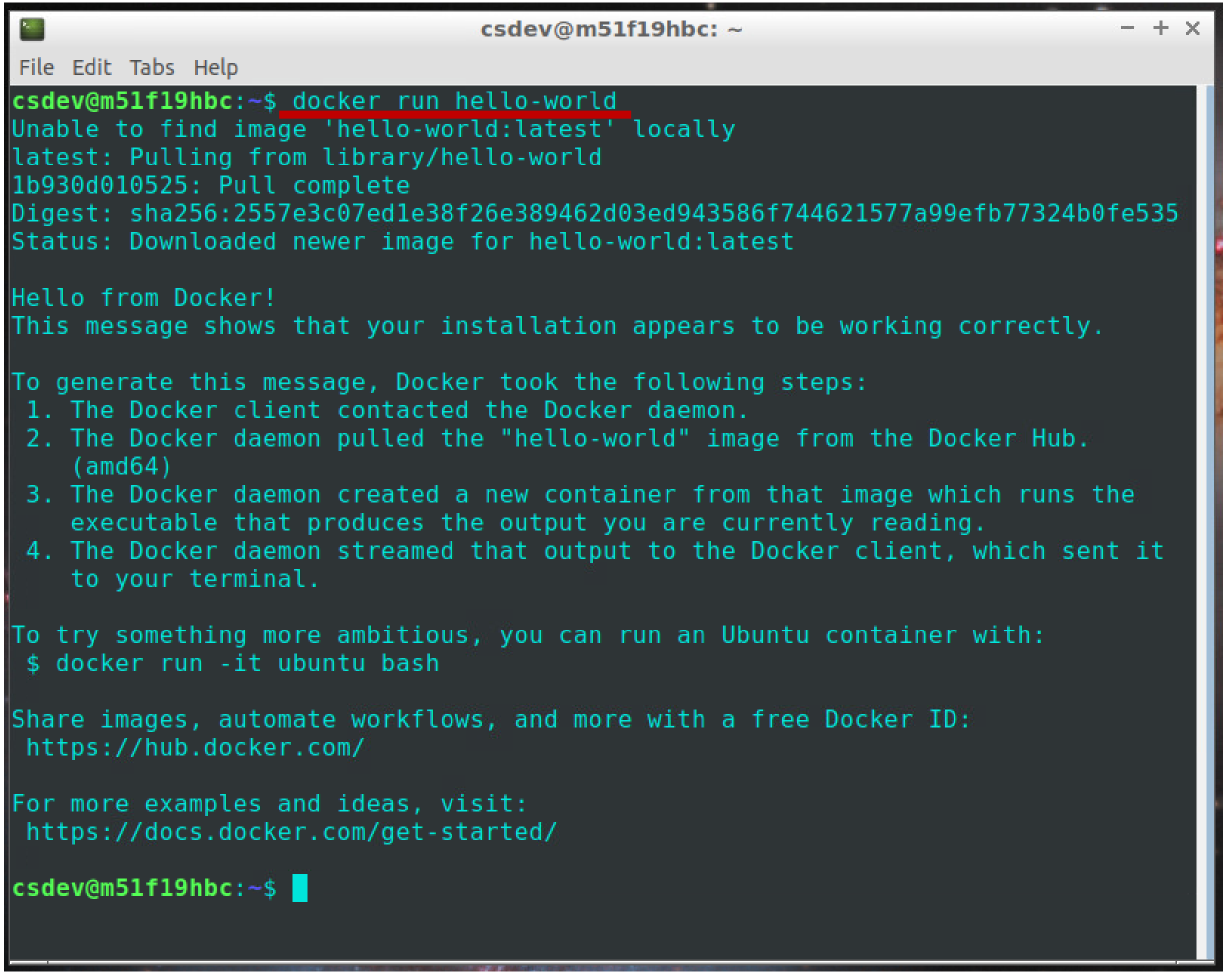
Docker Engine

The VMM of Docker

Images and Containers

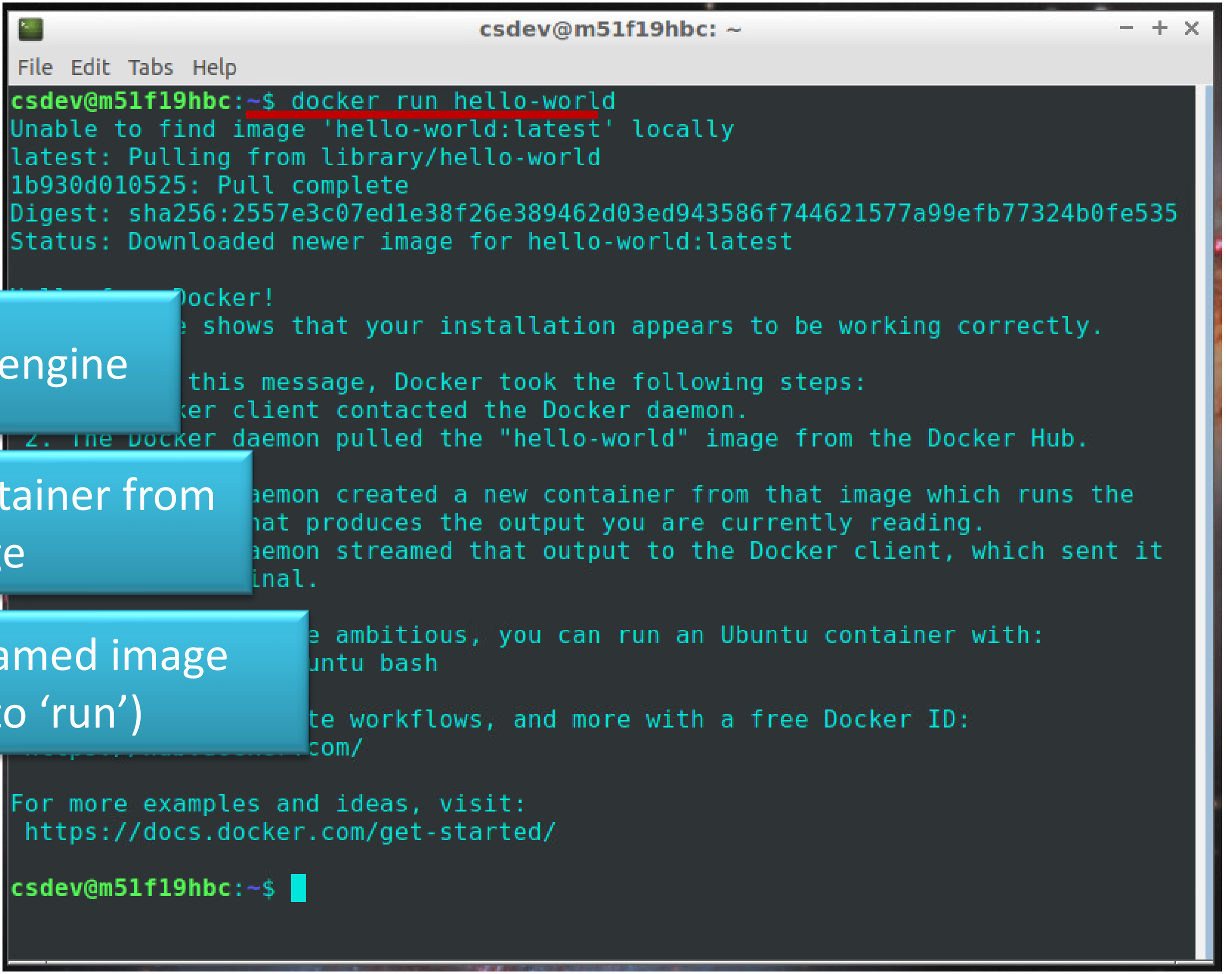
- Core concepts in Docker
  - **Image** The encapsulation of a VM
    - I.e. the physical file that contains the VM – **deployment unit**
    - Similar to a Java Jar file, DLL, .exe, war file, etc.
  - **Container** The executing instance of an image
    - Similar to an executing Java system, running the main() from the Jar file
  - **Docker Engine** The VMM program on Linux (Windows)
    - That handles images and executes containers

Example



```
csdev@m51f19hbc:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

csdev@m51f19hbc:~$
```

Example



```
csdev@m51f19hbc:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
Status: Downloaded newer image for hello-world:latest

           Docker!
             e shows that your installation appears to be working correctly.

              this message, Docker took the following steps:
             ker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

                   aemon created a new container from that image which runs the
                   hat produces the output you are currently reading.
                   aemon streamed that output to the Docker client, which sent it
                   inal.

                        e ambitious, you can run an Ubuntu container with:
                        untu bash

                        te workflows, and more with a free Docker ID:
                        com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

csdev@m51f19hbc:~$
```

‘docker’ = invoke docker engine

‘run’ = instantiate container from named image

‘hello-world’ = named image (parameter to ‘run’)

Host and Guest

- Now, you have two ”computers in one” – which is which?
- **Host:** The physical machine, providing CPU, RAM, etc.
- **Guest:** The virtual machine, handled by the VMM
- Of course, hosts often run multiple guests…
  - And your laptop (**host)** runs M101 (**guest**) that is actually the **host** of the Docker **guest** !

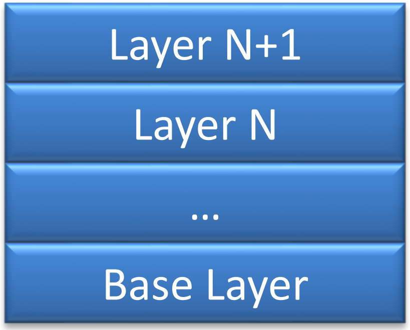
Linux LXC technologies

- Container
  - Application running on a *slice/view* of a (shared) OS
- Linux LXC technology extended
  - *namespaces (isolation)* provides an isolated share of OS resources
  - *cgroups (configuration)* provides resource management of OS resources (RAM, cpu, …)

Docker Image

Images

- Onion file system: *Copy-on-Write*
  - Every operation basically creates a new *file layer*
    - *Changing 'hans.txt' in layer N creates a (modified) copy of 'hans.txt' in layer N+1*



- **Base images** = 'prebaked file system'
  - All layers up-till N forms an Image
- I.e. *henrikbaerbak/cloudarch:e16.1*
  - Ubuntu 16.04 LTS server base image
  - Java, Ant, Ivy, Git, …   are all layered on top

Building Images

- How do you build a traditional server?
  - Unbox the machine, power up, install Linux, install application suite and libraries, execute server software

- Lifecycle - *classic*
  - container = instantiate(image1)
    - Docker run …
  - modify container
    - Install software, change files, add stuff, …
  - commit container → image2
    - Docker commit

Power up

Install your app

'Freeze' the machine

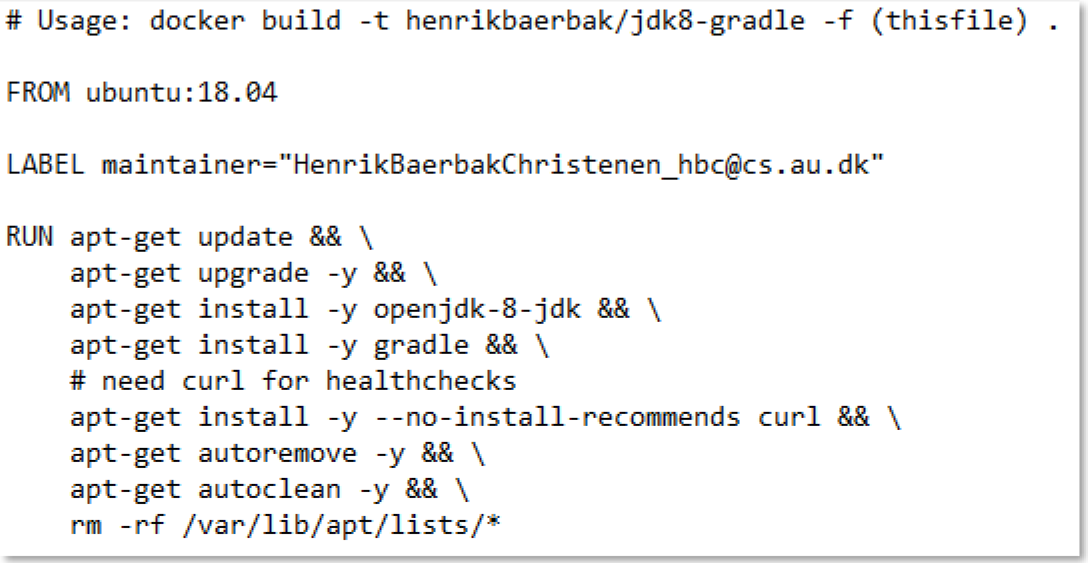
Dockerfiles

Infrastructure-as-code

Building Images

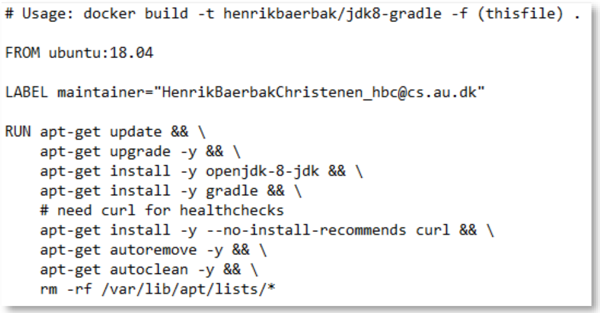
- Lifecycle – *infrastructure-as-code*
  - You automate the install script: Dockerfile

- Example: *henrikbaerbak/jdk8-gradle*



```
# Usage: docker build -t henrikbaerbak/jdk8-gradle -f (thisfile) .

FROM ubuntu:18.04

LABEL maintainer="HenrikBaerbakChristenen_hbc@cs.au.dk"

RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get install -y openjdk-8-jdk && \
    apt-get install -y gradle && \
    # need curl for healthchecks
    apt-get install -y --no-install-recommends curl && \
    apt-get autoremove -y && \
    apt-get autoclean -y && \
    rm -rf /var/lib/apt/lists/*
```

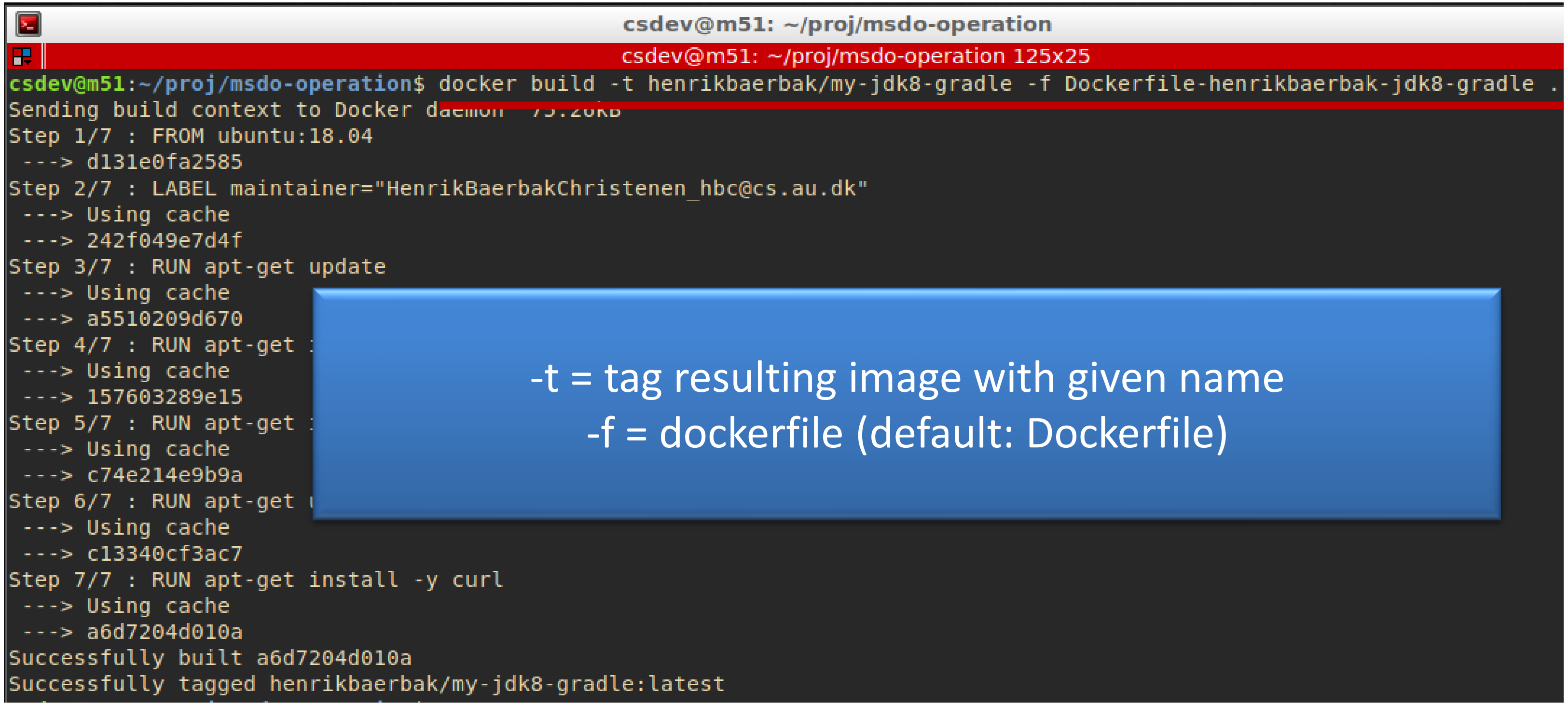
Anatomy



```
# Usage: docker build -t henrikbaerbak/jdk8-gradle -f (thisfile) .

FROM ubuntu:18.04

LABEL maintainer="HenrikBaerbakChristenen_hbc@cs.au.dk"

RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get install -y openjdk-8-jdk && \
    apt-get install -y gradle && \
    # need curl for healthchecks
    apt-get install -y --no-install-recommends curl && \
    apt-get autoremove -y && \
    apt-get autoclean -y && \
    rm -rf /var/lib/apt/lists/*
```

- State your *base image*
- Identify yourself
- Install software
  - WORKDIR set working directory in container
  - COPY copy <src> to <dest> (from host to cnt.)
  - RUN run command *at build time*
- Configure
  - EXPOSE expose port to the outside
- Execute
  - (ENTRYPOINT) state default script to run
  - CMD run command *at run time (container process)*

Building Image



csdev@m51: ~/proj/msdo-operation
csdev@m51: ~/proj/msdo-operation 125x25

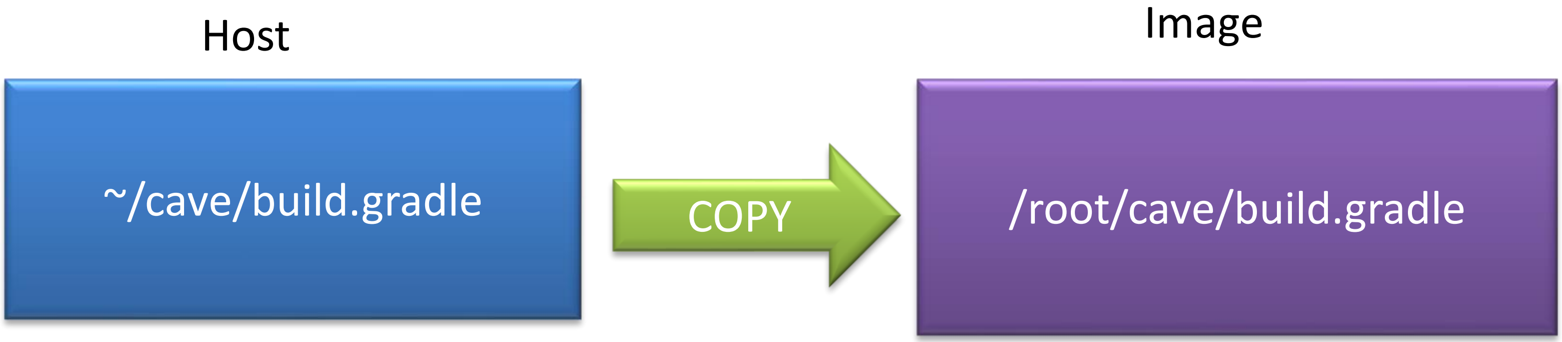
```
csdev@m51:~/proj/msdo-operation$ docker build -t henrikbaerbak/my-jdk8-gradle -f Dockerfile-henrikbaerbak-jdk8-gradle .
Sending build context to Docker daemon  73.26kB
Step 1/7 : FROM ubuntu:18.04
 ---> d131e0fa2585
Step 2/7 : LABEL maintainer="HenrikBaerbakChristenen_hbc@cs.au.dk"
 ---> Using cache
 ---> 242f049e7d4f
Step 3/7 : RUN apt-get update
 ---> Using cache
 ---> a5510209d670
Step 4/7 : RUN apt-get i
 ---> Using cache
 ---> 157603289e15
Step 5/7 : RUN apt-get i
 ---> Using cache
 ---> c74e214e9b9a
Step 6/7 : RUN apt-get u
 ---> Using cache
 ---> c13340cf3ac7
Step 7/7 : RUN apt-get install -y curl
 ---> Using cache
 ---> a6d7204d010a
Successfully built a6d7204d010a
Successfully tagged henrikbaerbak/my-jdk8-gradle:latest
```

-t = tag resulting image with given name

-f = dockerfile (default: Dockerfile)

Building Image

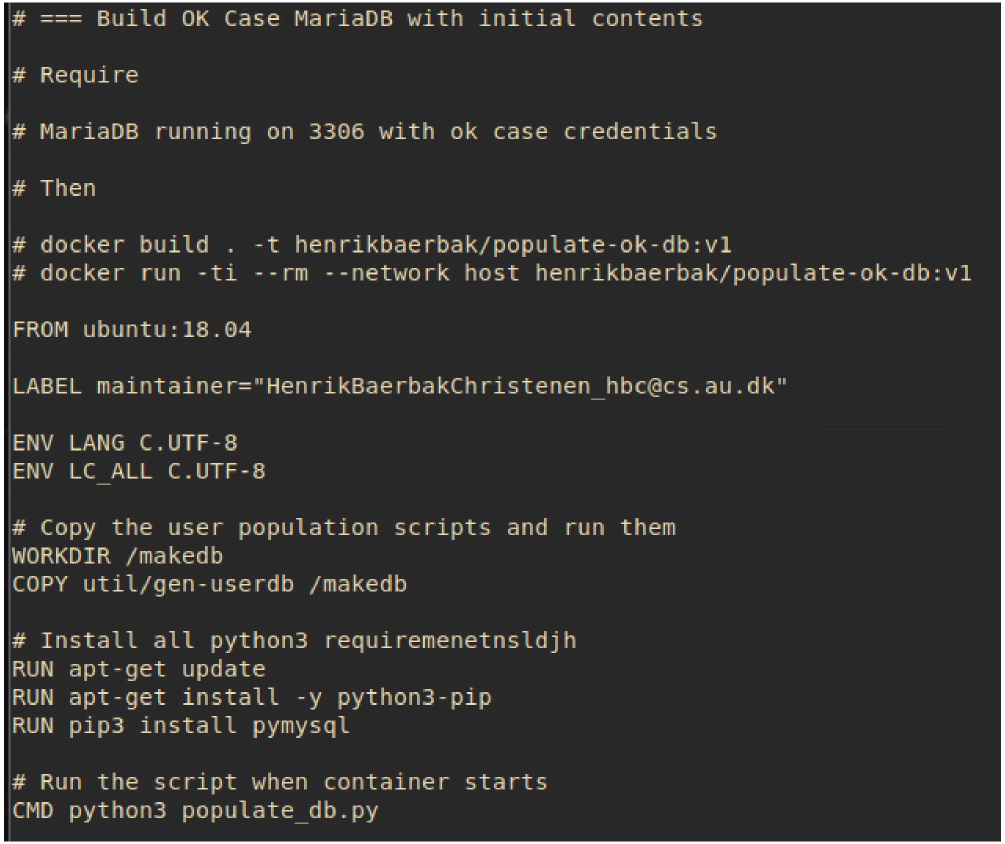
- When ‘build’ is executed the *host* and the *container* co-exists and you typically copy files from host to image



- Dockerfile is *typically in the project root folder!*
  - Version controlled along with project !!!
  - Modifiability QA: Group related things together – Cohesion

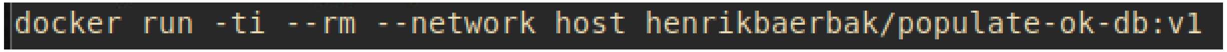
Example

- Populate a DB
  - Populate_db.py
    - Runs against DB on *localhost:3306*
- *Exercise:*
  - *What happens?*



```
# === Build OK Case MariaDB with initial contents

# Require

# MariaDB running on 3306 with ok case credentials

# Then

# docker build . -t henrikbaerbak/populate-ok-db:v1
# docker run -ti --rm --network host henrikbaerbak/populate-ok-db:v1

FROM ubuntu:18.04

LABEL maintainer="HenrikBaerbakChristenen_hbc@cs.au.dk"

ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8

# Copy the user population scripts and run them
WORKDIR /makedb
COPY util/gen-userdb /makedb

# Install all python3 requiremenetnsldjh
RUN apt-get update
RUN apt-get install -y python3-pip
RUN pip3 install pymysql

# Run the script when container starts
CMD python3 populate_db.py
```

- To run



```
docker run -ti --rm --network host henrikbaerbak/populate-ok-db:v1
```

CMD vrs RUN

- CMD = Execute
- RUN = Execute
- What is the difference???
- **Big!**
  - RUN Execute at **build time**
    - That is, during the ‘docker build .’ phase
  - CMD Execute at **container run time**
    - That is, when the ‘docker run …’ is executed

No No's

- Much to my **dislike**, Docker has opted for using a .dockerignore file
  - Just like you have a .gitignore file
    - Wildcard specs of files **not** to add to git staging area

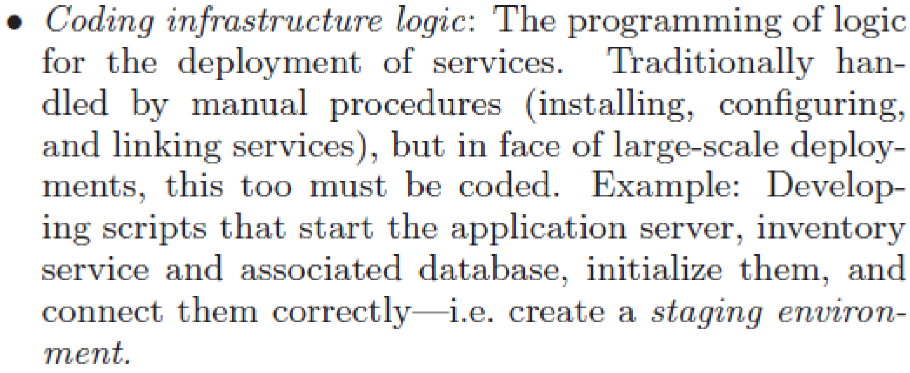
- **But!** A Dockerfile is the infrastructure-code that *explicitly* states what goes into an image!

  - *"Ups, by the way, I regret copying *.BAK files into the image, please remove them again"*

Do not use .dockerignore !!! Except…

Infrastructure-as-code

- DevOps is about speed and agility in going from Dev to Ops

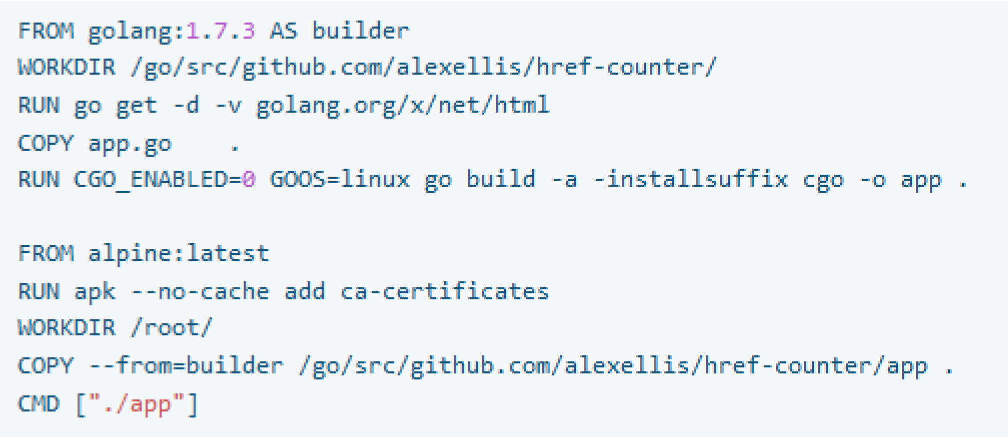


- *Coding infrastructure logic*: The programming of logic for the deployment of services. Traditionally handled by manual procedures (installing, configuring, and linking services), but in face of large-scale deployments, this too must be coded. Example: Developing scripts that start the application server, inventory service and associated database, initialize them, and connect them correctly—i.e. create a *staging environment*.

- Dockerfiles are one **big** piece of this puzzle: *Installing the software on a server, is coded in a programming language, is under version control with your source !*

MultiStage DockerFiles

- From Engine 17.05+
- Idea
  - Build in steps
    - Step1: compile and assemble deployment unit (‘jar’ in Java)
    - Step2: produce container with *just the jar and ‘execute’ CMD*



```
FROM golang:1.7.3 AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go    .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
CMD ["./app"]
```

Example Exercise

- From the mandatory project

- Step1:
  - Install the full SkyCave system
  - Finally RUN gradle to produce a ‘fatJar’ with full system in a single deployment unit
- Step2:
  - Copy the fatJar from step1 *and only that!*
  - CMD to start skycave daemon *using the fatJar*

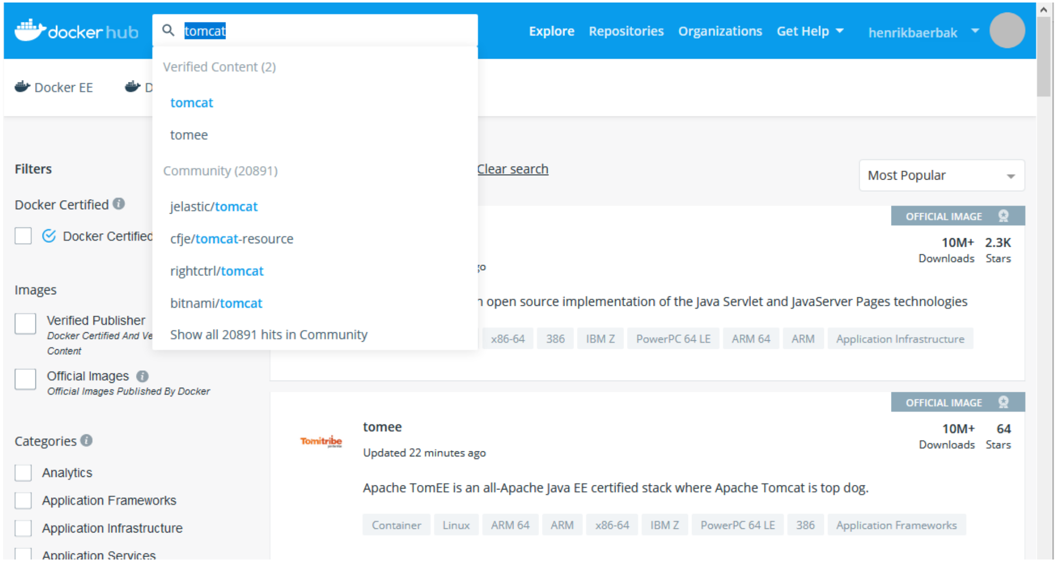
AARHUS UNIVERSITET

Docker Hub

Sharing Images

Docker Hub

- ‘Bootstrapping’ – where do I get the base layer from?
- Docker Hub is a public repository of a lot of (base) images



n open source implementation of the Java Servlet and JavaServer Pages technologies

tomee

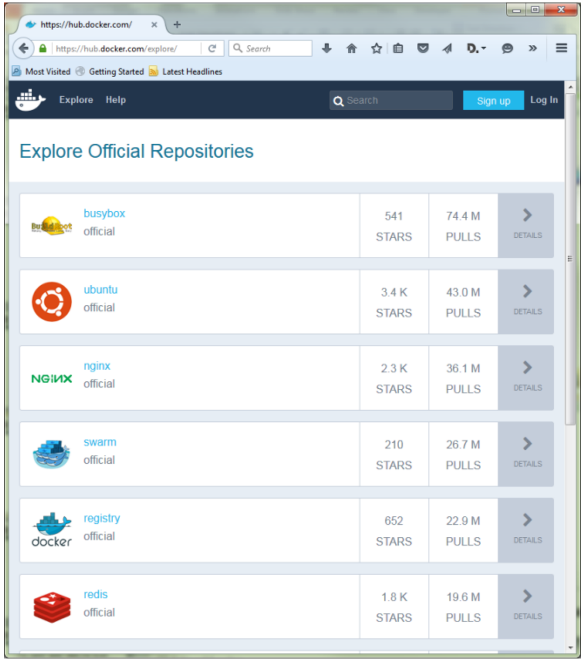
Updated 22 minutes ago

Apache TomEE is an all-Apache Java EE certified stack where Apache Tomcat is top dog.

Docker Hub

- The GitHub / Maven Repo movement
  - ‘push’ your commits to a cloud base storage service
    - Mvnrepository / github / bitbucket …
  - ‘pull/clone’ from there

- Docker Hub
  - Register as a user (free)
  - Push your image to docker hub
  - Done!



Explore Official Repositories

busybox official	541 STARS	74.4 M PULLS	DETAILS
ubuntu official	3.4 K STARS	43.0 M PULLS	DETAILS
nginx official	2.3 K STARS	36.1 M PULLS	DETAILS
swarm official	210 STARS	26.7 M PULLS	DETAILS
registry official	652 STARS	22.9 M PULLS	DETAILS
redis official	1.8 K STARS	19.6 M PULLS	DETAILS

Image Naming

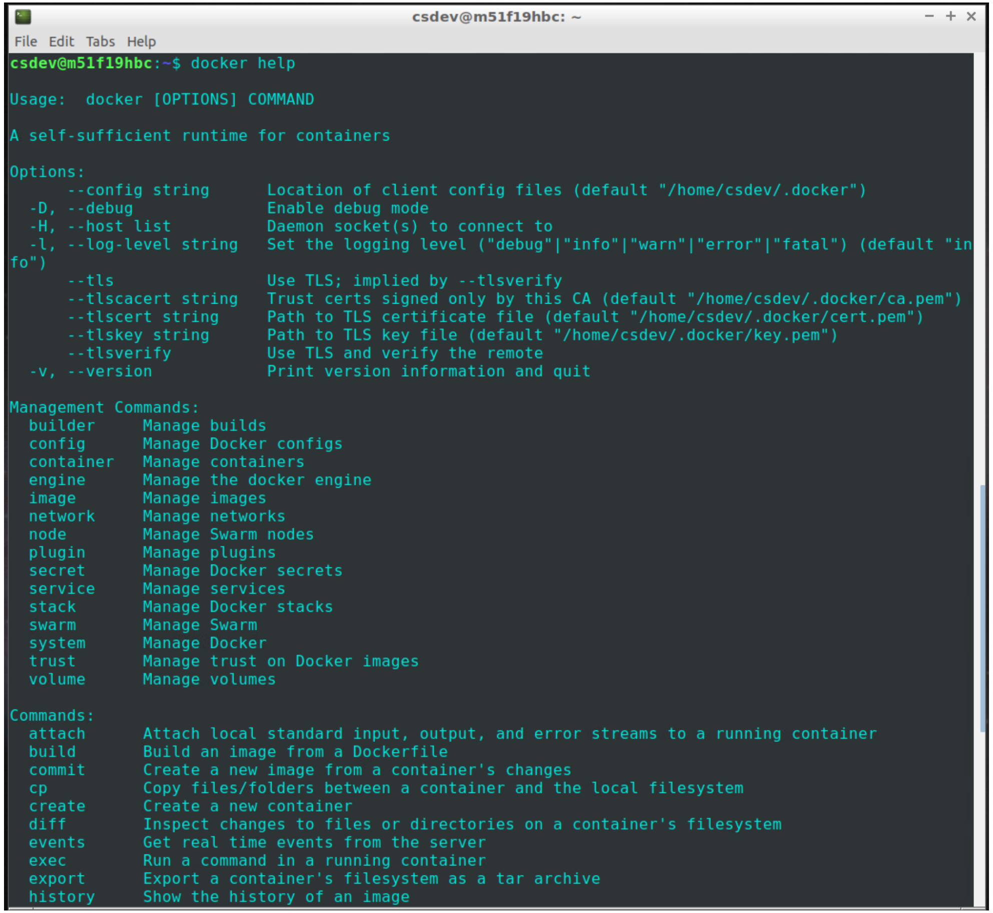
- Images can be called any name
  - Foobar, tmp, fiskesovs
- However, if you have to push them to the hub, they have to follow the convention:
  - ***username/repository:tag***
- Only *one* docker hub repository can be *private*
  - I have called mine for ‘private’, and then use tags to differentiate different teaching images from each other...

Container Lifecycle Management

Docker 101

We need to…

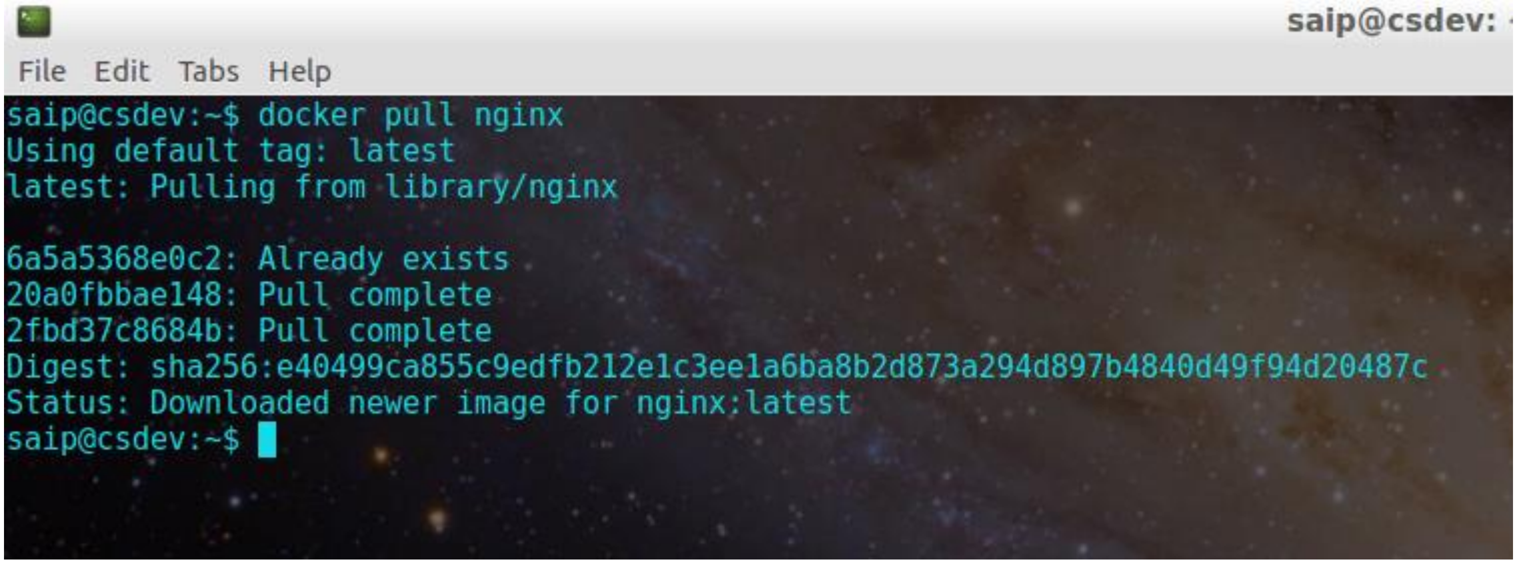
- Build images
- Execute containers
- Monitor containers
- Kill containers



```
csdev@m51f19hbc:~$ docker help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/home/csdev/.docker")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "in
fo")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/home/csdev/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/home/csdev/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/home/csdev/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
  engine      Manage the docker engine
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
```

Images

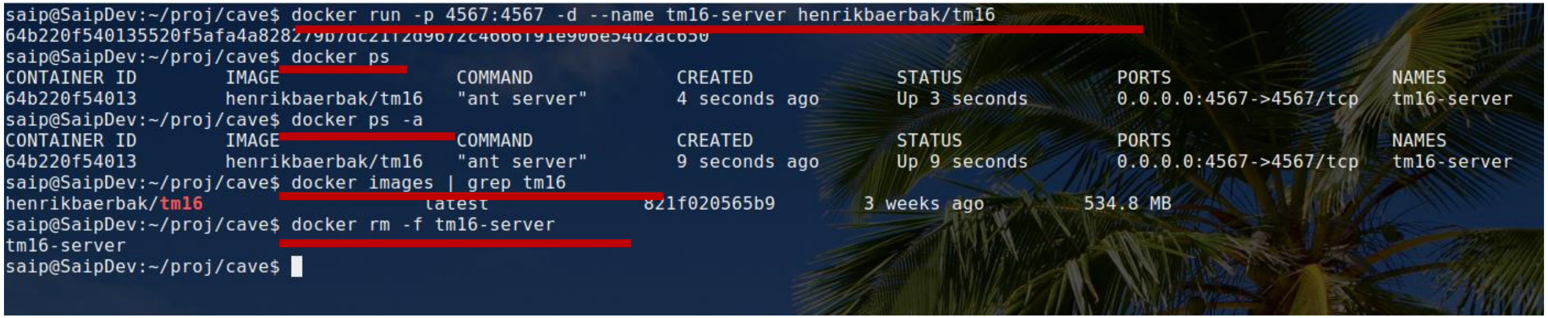
- ‘pull’ = pull a named image from Hub
- ‘push’ = push a named image to Hub



```
saip@csdev:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx

6a5a5368e0c2: Already exists
20a0fbbae148: Pull complete
2fbd37c8684b: Pull complete
Digest: sha256:e40499ca855c9edfb212e1c3ee1a6ba8b2d873a294d897b4840d49f94d20487c
Status: Downloaded newer image for nginx:latest
saip@csdev:~$
```

AARHUS UNIVERSITET

Containers



```
saip@SaipDev:~/proj/cave$ docker run -p 4567:4567 -d --name tm16-server henrikbaerbak/tm16
64b220f540135520f5afa4a828279b7dc21f2d9672c4666f91e906e54d2ac650
saip@SaipDev:~/proj/cave$ docker ps
CONTAINER ID        IMAGE                COMMAND             CREATED             STATUS              PORTS                    NAMES
64b220f54013        henrikbaerbak/tm16   "ant server"        4 seconds ago       Up 3 seconds        0.0.0.0:4567->4567/tcp   tm16-server
saip@SaipDev:~/proj/cave$ docker ps -a
CONTAINER ID        IMAGE                COMMAND             CREATED             STATUS              PORTS                    NAMES
64b220f54013        henrikbaerbak/tm16   "ant server"        9 seconds ago       Up 9 seconds        0.0.0.0:4567->4567/tcp   tm16-server
saip@SaipDev:~/proj/cave$ docker images | grep tm16
henrikbaerbak/tm16                    latest              821f020565b9        3 weeks ago         534.8 MB
saip@SaipDev:~/proj/cave$ docker rm -f tm16-server
tm16-server
saip@SaipDev:~/proj/cave$ █
```

- run = (pull image), and start container (-d = in background)
- **ps** = see all running containers (-a = all, also the dead ones)
- images = see all images
- rm = remove container (-f =force, even if currently running)

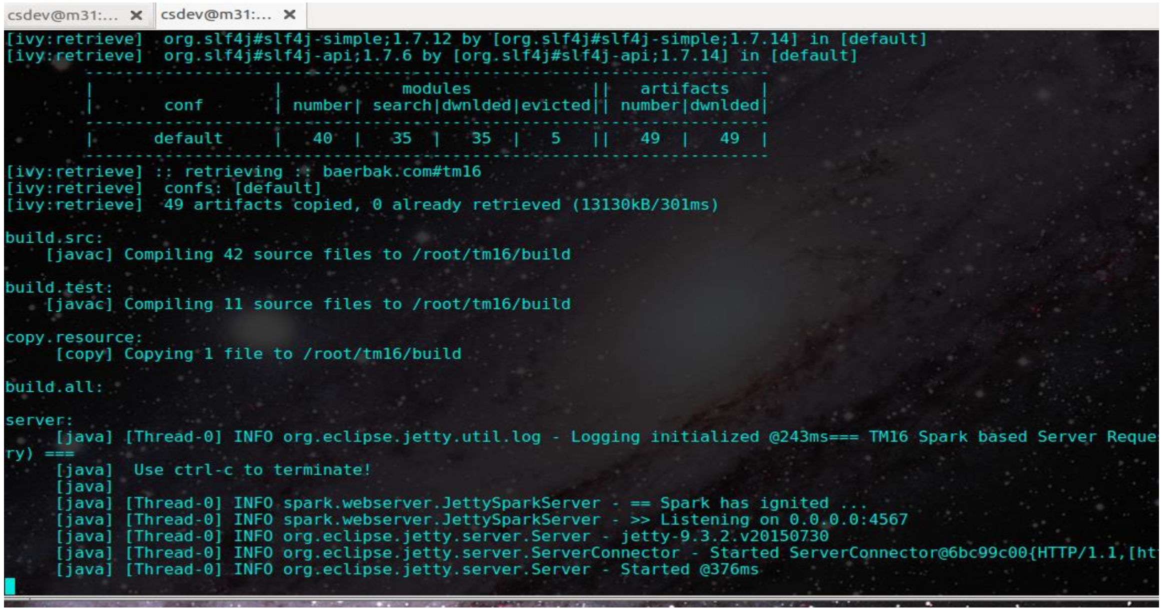
OldSchool: docker ps =
NewSchool: docker container ls

Docker run

- Docker run (image) (command parameters)
  - Acceptable parameters defined by the image!
- Zillion parameters, the most important are
  - docker run –d : -d = daemon mode / server
  - -ti: terminal interactive, so you can interact!
  - --name (myname): give it a name
  - -p (host):(container): **port-mapping**
  - --network=(network): **define network to use**
    - See later…

Containers

- ‘logs’ = See the log (shell output) of running container
- ‘logs –f’ = Tails the log (keeps running)

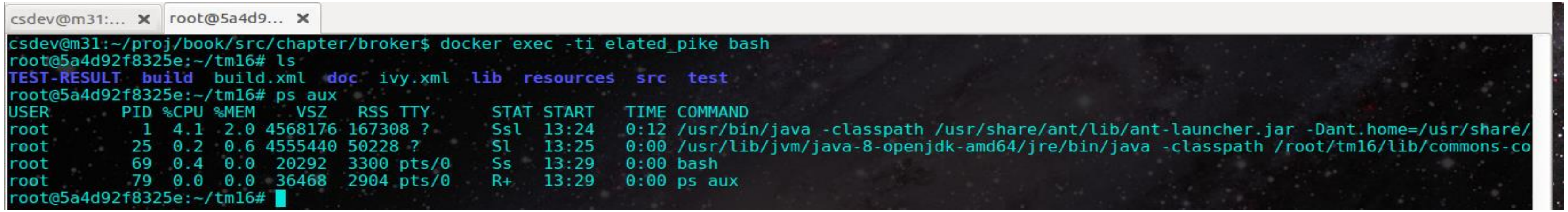


```
[ivy:retrieve]  org.slf4j#slf4j-simple;1.7.12 by [org.slf4j#slf4j-simple;1.7.14] in [default]
[ivy:retrieve]  org.slf4j#slf4j-api;1.7.6 by [org.slf4j#slf4j-api;1.7.14] in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   40  |   35  |   35  |   5   ||   49  |   49  |
        ---------------------------------------------------------------------
[ivy:retrieve] :: retrieving :: baerbak.com#tm16
[ivy:retrieve]  confs: [default]
[ivy:retrieve]  49 artifacts copied, 0 already retrieved (13130kB/301ms)

build.src:
    [javac] Compiling 42 source files to /root/tm16/build

build.test:
    [javac] Compiling 11 source files to /root/tm16/build

copy.resource:
     [copy] Copying 1 file to /root/tm16/build

build.all:

server:
     [java] [Thread-0] INFO org.eclipse.jetty.util.log - Logging initialized @243ms=== TM16 Spark based Server Reque
ry) ===
     [java]  Use ctrl-c to terminate!
     [java]
     [java] [Thread-0] INFO spark.webserver.JettySparkServer - == Spark has ignited ...
     [java] [Thread-0] INFO spark.webserver.JettySparkServer - >> Listening on 0.0.0.0:4567
     [java] [Thread-0] INFO org.eclipse.jetty.server.Server - jetty-9.3.2.v20150730
     [java] [Thread-0] INFO org.eclipse.jetty.server.ServerConnector - Started ServerConnector@6bc99c00{HTTP/1.1,[ht
     [java] [Thread-0] INFO org.eclipse.jetty.server.Server - Started @376ms
```

Containers

- A container is very opaque ☹
  - What the heck is going on inside it ???
  - Where are the files located???

- ‘exec -ti (container) bash’
  - Is: execute ‘bash’ interactive TTY in a *running* container



```
csdev@m31:~/proj/book/src/chapter/broker$ docker exec -ti elated_pike bash
root@5a4d92f8325e:~/tm16# ls
TEST-RESULT  build  build.xml  doc  ivy.xml  lib  resources  src  test
root@5a4d92f8325e:~/tm16# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  4.1  2.0 4568176 167308 ?      Ssl  13:24   0:12 /usr/bin/java -classpath /usr/share/ant/lib/ant-launcher.jar -Dant.home=/usr/share/
root        25  0.2  0.6 4555440 50228 ?       Sl   13:25   0:00 /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java -classpath /root/tm16/lib/commons-co
root        69  0.4  0.0  20292  3300 pts/0    Ss   13:29   0:00 bash
root        79  0.0  0.0  36468  2904 pts/0    R+   13:29   0:00 ps aux
root@5a4d92f8325e:~/tm16# 
```

AARHUS UNIVERSITET

Docker Networking

Microservices communicate, right?

Networking

- Distributed systems rely on networking!
- By default, network is an **isolated resource** in Docker!
  - Ten Apache web servers, all listening on port 80, on the same machine!

- Two core technologies
  - Port forwarding
    - For ‘exposing’ container services to the outside/host
  - Docker network drivers
    - For ‘binding’ container services together securely

Port Forwarding

- docker run -p 7777:6745 …
  - Bind container port 6745 in container to host’s external port 7777
  - So if you connect to ‘localhost:7777’ you will actually communicate with port 6745 of the service running in the docker container

- Make docker services act like they are deployed on host

Network drivers

- Any machine has several *network interfaces*
  - Linux: ‘lo’ = Local Loopback, ‘ens32’ = Ethernet, …
- Docker will create new networks and attach containers to them
  - By default they are not shared among containers
- **docker run --*network=container:daemon* (image) cmd**
  - This container will now reuse the network of container named ‘daemon’, i.e. they can communicate!
- Other options are

– --network=host	reuse host’s network
– --network=my-network	use named network

Annoying Note

- *This note has cost me a fair share of gray hair!*
- ***What about firewalls?***
  - On my DigitalOcean machines I want a firewall up!
  - Not my field of expertise, so ‘ufw’ = easy beginners linux firewall
- *But Docker –p circumvents ufw and ufw does not know!*
  - *Docker calls ‘iptables’ directly*
- So – Docker –p punch a hole in the firewall that you cannot deny using ufw ☹

AARHUS UNIVERSITET

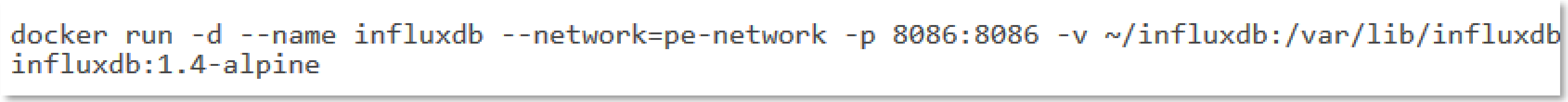
Docker Volumes

Persistence

- A container is self-contained
  - All ‘disks’ are virtual disks within the container
  - Practical for single-file deployments ☺ ☺ ☺
  - But... *Not so practical* for persistent data !!!
- **Docker Volumes**
  - Is the solution to this issue – volumes are stored ***on host***
  - Switch **‘- v hostfolder:containerfolder’** means ”mount hostfolder so it appears on path containerfolder’ in the container!

Exercise

- What does the following do?



```
docker run -d --name influxdb --network=pe-network -p 8086:8086 -v ~/influxdb:/var/lib/influxdb
influxdb:1.4-alpine
```

- One caveat…
  - You have to know in which folder the service stores its persistent data
    - MongoDB: /data/db
    - MariaDB: /var/lib/mysql
    - Etc. ect.

Named Volumes

- Actually ‘-v’ and mounted volumes is *so yesterday…*

- Instead of mounting on the host, you can let Docker organize it using *named volumes*
  - Required when deploying on swarms!

  - So – we will return to that later…

Summary

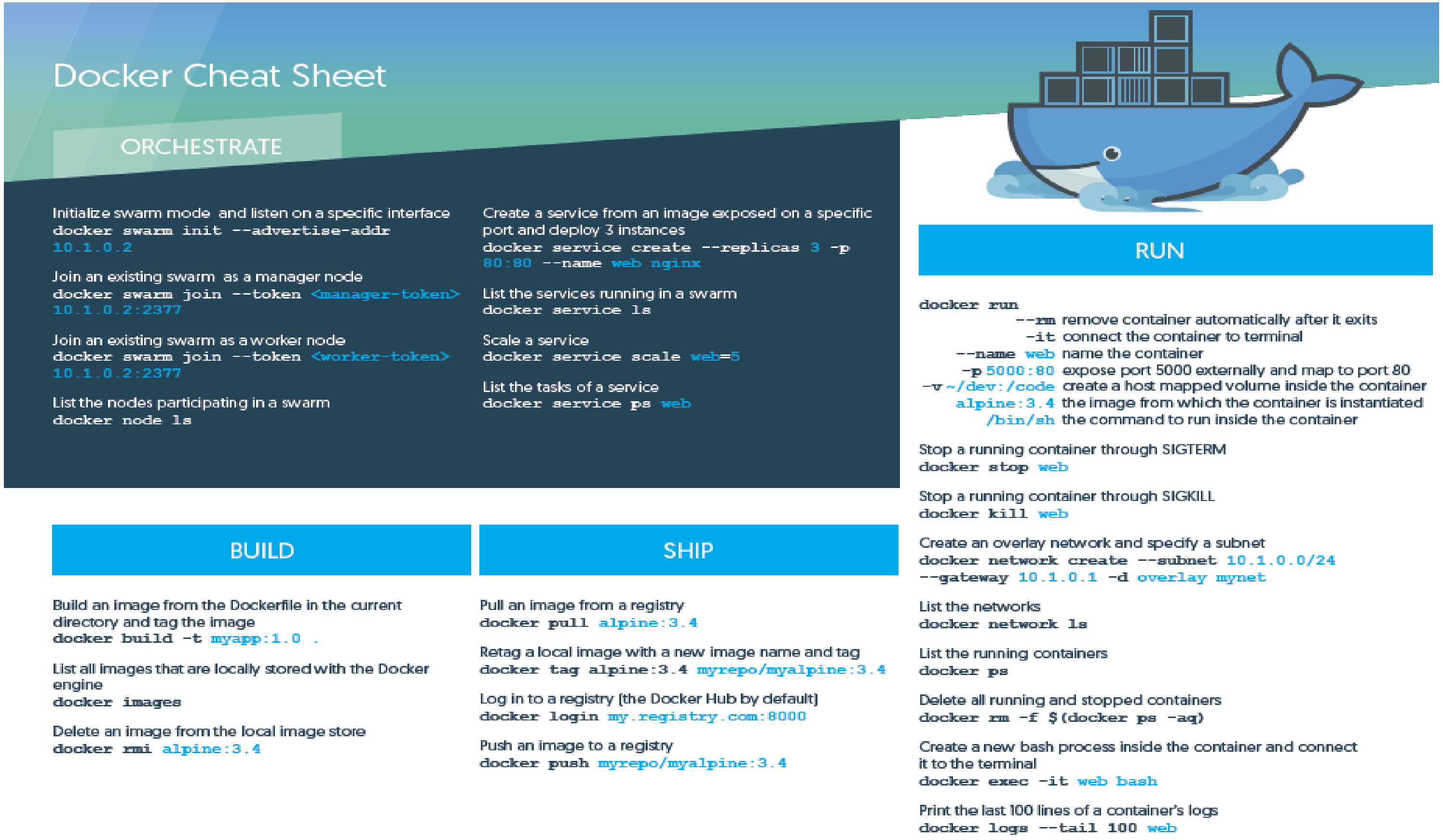
- Docker is a *light-weight process VM technology based upon Linux*

- It will form the backbone throughout this course ☺

- The learning curve is a bit steep – zillions of commands and parameters ☹

AARHUS UNIVERSITET

Find it on the Web ☺



Docker Cheat Sheet

ORCHESTRATE

Initialize swarm mode and listen on a specific interface
```
docker swarm init --advertise-addr
10.1.0.2
```

Join an existing swarm as a manager node
```
docker swarm join --token <manager-token>
10.1.0.2:2377
```

Join an existing swarm as a worker node
```
docker swarm join --token <worker-token>
10.1.0.2:2377
```

List the nodes participating in a swarm
```
docker node ls
```

Create a service from an image exposed on a specific port and deploy 3 instances
```
docker service create --replicas 3 -p
80:80 --name web nginx
```

List the services running in a swarm
```
docker service ls
```

Scale a service
```
docker service scale web=5
```

List the tasks of a service
```
docker service ps web
```

BUILD

Build an image from the Dockerfile in the current directory and tag the image
```
docker build -t myapp:1.0 .
```

List all images that are locally stored with the Docker engine
```
docker images
```

Delete an image from the local image store
```
docker rmi alpine:3.4
```

SHIP

Pull an image from a registry
```
docker pull alpine:3.4
```

Retag a local image with a new image name and tag
```
docker tag alpine:3.4 myrepo/myalpine:3.4
```

Log in to a registry (the Docker Hub by default)
```
docker login my.registry.com:8000
```

Push an image to a registry
```
docker push myrepo/myalpine:3.4
```

RUN

```
docker run
            --rm remove container automatically after it exits
             -it connect the container to terminal
      --name web name the container
      -p 5000:80 expose port 5000 externally and map to port 80
 -v ~/dev:/code create a host mapped volume inside the container
      alpine:3.4 the image from which the container is instantiated
         /bin/sh the command to run inside the container
```

Stop a running container through SIGTERM
```
docker stop web
```

Stop a running container through SIGKILL
```
docker kill web
```

Create an overlay network and specify a subnet
```
docker network create --subnet 10.1.0.0/24
--gateway 10.1.0.1 -d overlay mynet
```

List the networks
```
docker network ls
```

List the running containers
```
docker ps
```

Delete all running and stopped containers
```
docker rm -f $(docker ps -aq)
```

Create a new bash process inside the container and connect it to the terminal
```
docker exec -it web bash
```

Print the last 100 lines of a container's logs
```
docker logs --tail 100 web
```