Docker Intro

Karel de Grote Hogeschool

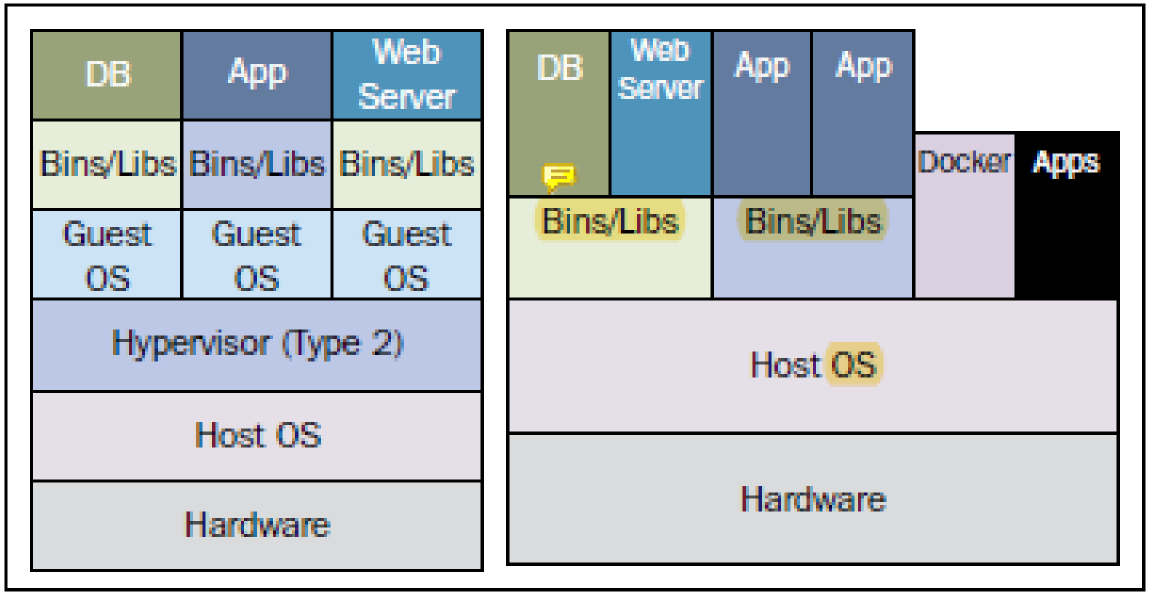
11-2021

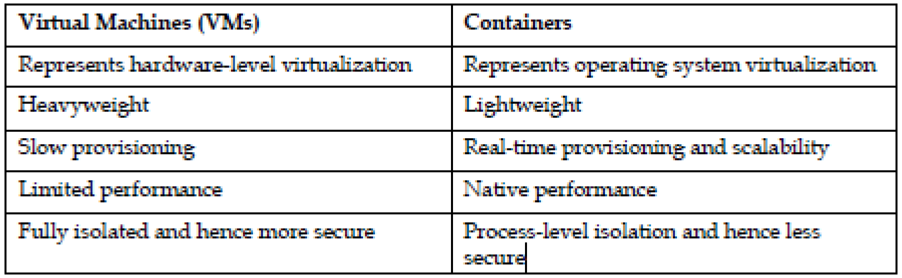
Introduction

- What is docker?
- Terminology

What is Docker?

- Not virtualization but containerization:





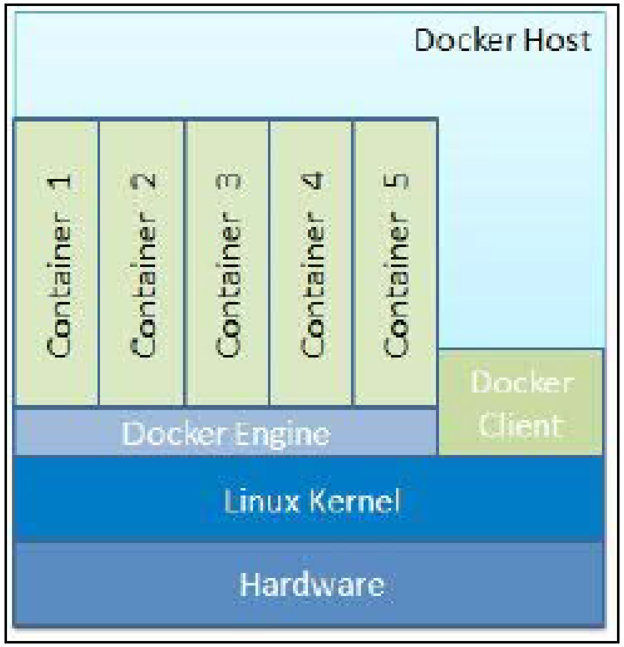
Virtual Machines (VMs)	Containers
Represents hardware-level virtualization	Represents operating system virtualization
Heavyweight	Lightweight
Slow provisioning	Real-time provisioning and scalability
Limited performance	Native performance
Fully isolated and hence more secure	Process-level isolation and hence less secure

What is Docker?

- Container Engine
  - LXC, FreeBSD jail, OpenVZ, AIX WPARs, Solaris Containers, …
- Open source
- Written in GO
- “Software bucket” containing everything to run software independently.
  - Process runs isolated on the OS of the host

What is Docker?

- Process runs on host OS. (Shared OS Kernel)
- Process runs isolated
  - Uses features of Linux kernel for isolation:
    - Namespaces
    - Control groups
  - Can share OS parts between containers
- Uses a Union file system



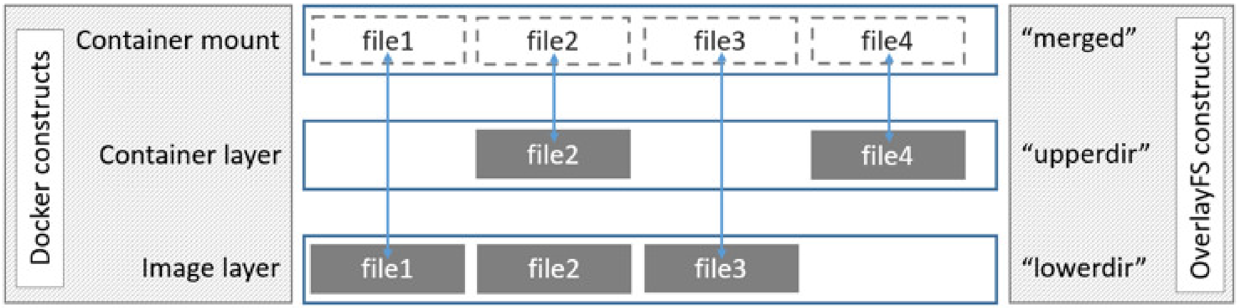
What is Docker?

“Even when you don’t run containers, you are in a container”

By Jérôme Petazzoni (2015)

What is Docker?

- Uses a Union file system (part of main linux kernel since version 3.18)
  - 3+1 layers/directories:
    - lower (base = R), upper (diff = W) & merged (overlay = visibility & user interaction)
    - Work (workdir) = intermediate layer= used to prepare files as they are switched between the layers (file copy = atomic action) = internal to FS
      - Side-note: lower can also be an merged (overlay)
  - Copy-on write filesystem
  - Used in “image-building”
  - Types:
    - AUFS
    - Overlay2
    - BTRFS



Terminology

- Images - The file system and configuration of our application which are used to create containers.
  - Read-only
  - Instructions for container creation
  - “Layered” (based on other images)
- Containers - Running instances of Docker images — containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS.
  - Read/Write
  - Runnable

Terminology

- Docker Solution:
  - Docker Engine
    - Runs the containers
  - Docker Hub
    - Contains the docker images
    - Registry & Repository

Terminology

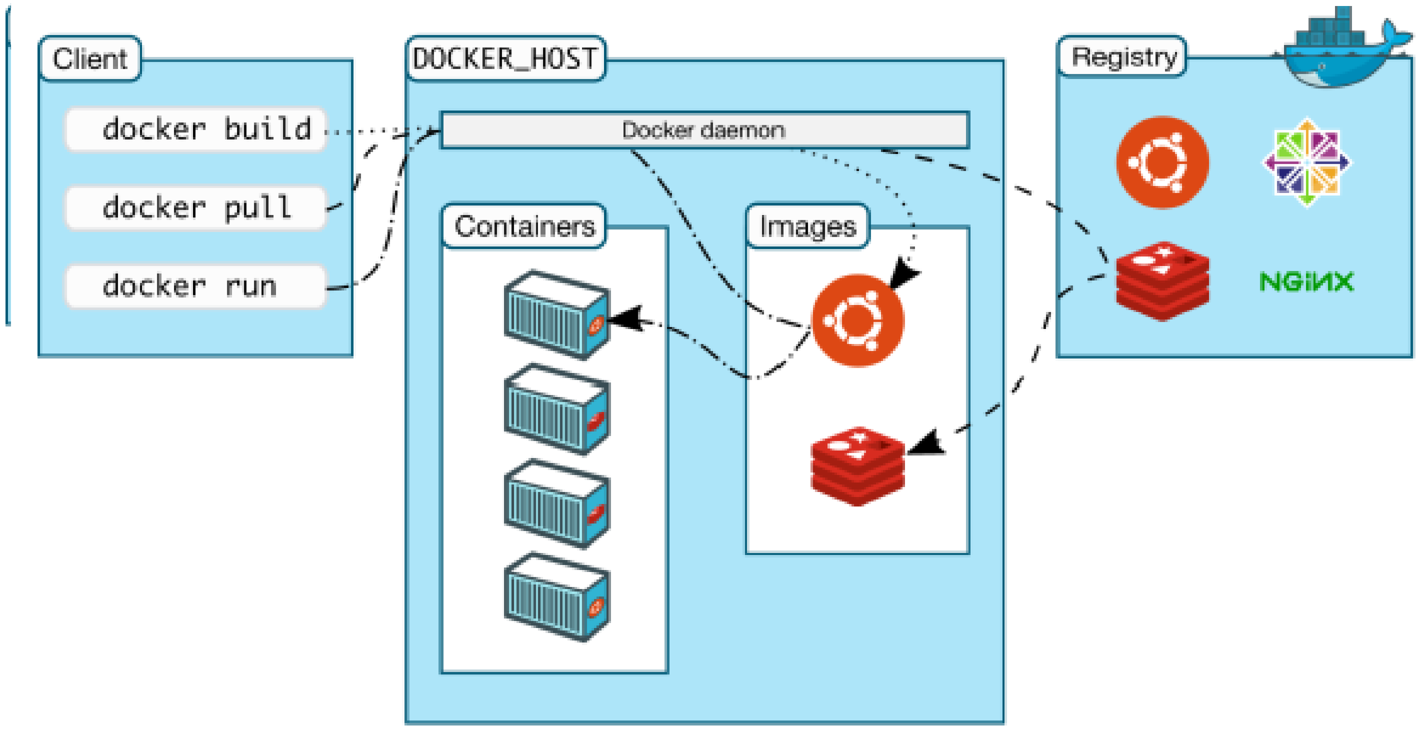
Docker Image
Basis van een Docker container. Bevat volledige applicatie met nodige executables en bibliotheken (cfr class)

Docker Container
Hierin draait de applicatie of service (cfr instance)

Docker Engine
Maakt of verdeelt Docker containers lokaal of in de cloud

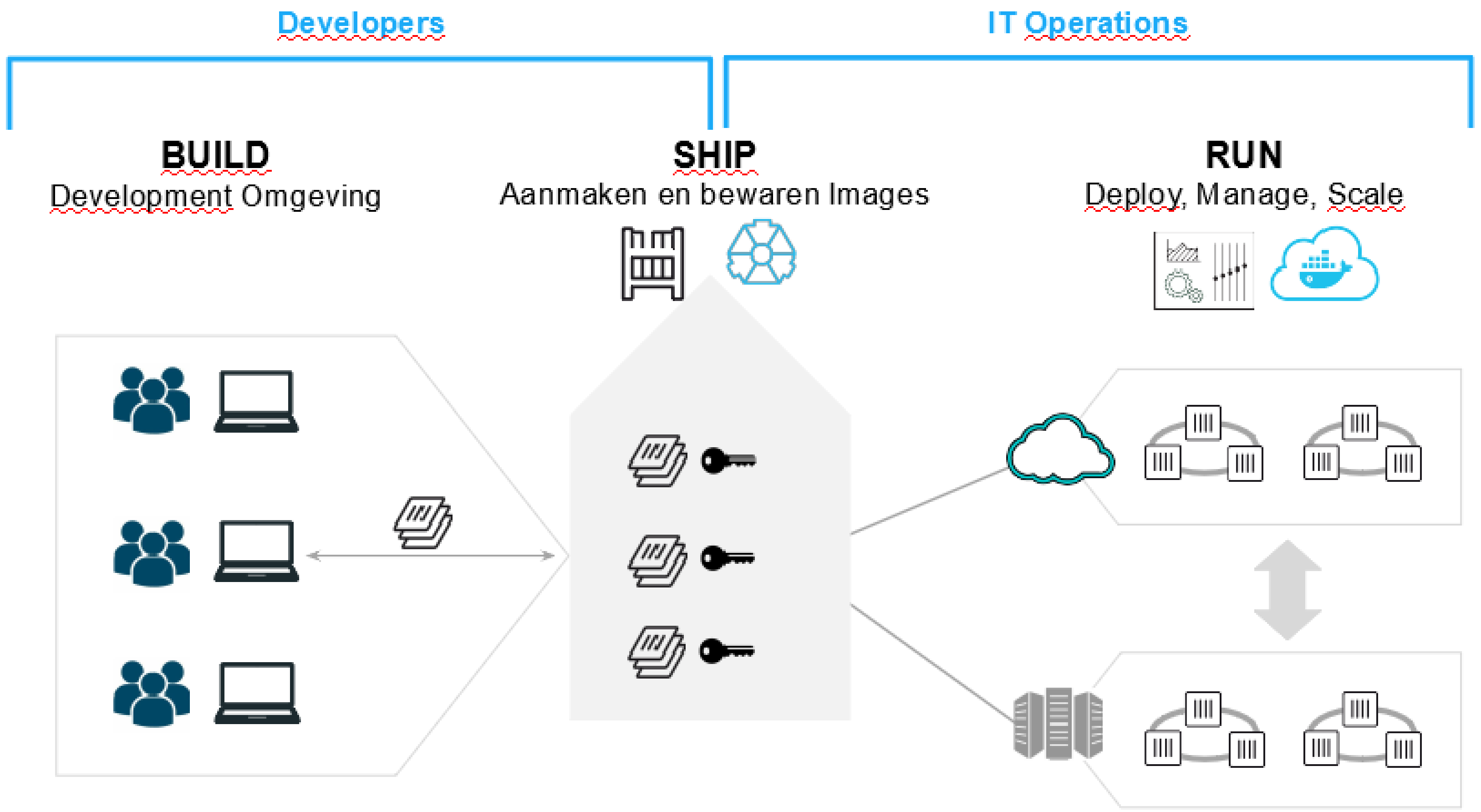
Registry Service
Docker Hub (Publiek) of Docker Trusted Registry (Privaat)
Cloud of server gebaseerde opslag/distributie van images

Docker Architecture



Source: https://docs.docker.com/engine/docker-overview/#docker-architecture

“DevOps”


Developers
IT Operations
BUILD
Development Omgeving
SHIP
Aanmaken en bewaren Images
RUN
Deploy, Manage, Scale

Docker Installing

Karel de Grote Hogeschool

11-2021

Installing

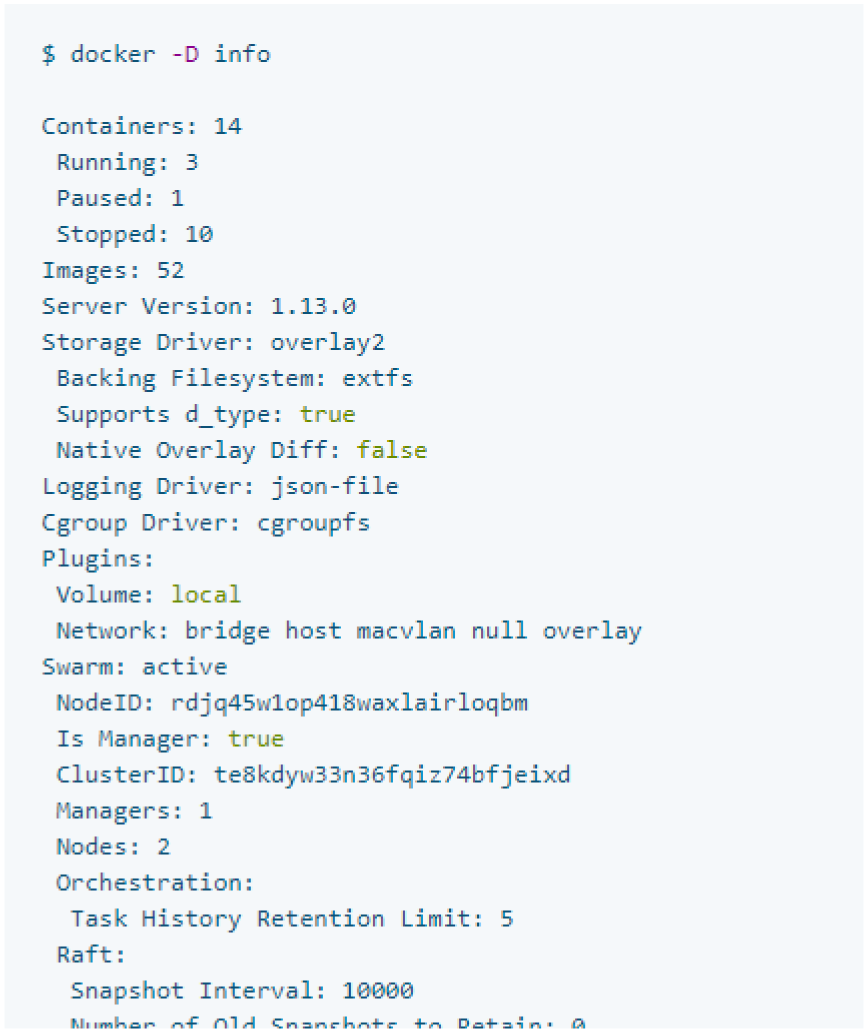
- Types

Installing

- Linux Kernel
  - Ubuntu
    - Package Repository
      - Distributions must package latest version
      - Alternative: manually
    - Docker.io script
      - Automated script from Docker community
- Windows/Mac (not treated)
  - Docker Desktop (no VirtualBox, needs Hyper-V)
  - VMs
    - Boot2Docker (old, depreciated)

Installing

- Check install:
  - Docker version
    - Shows:
    - The client version
    - The client API version
    - The server version
    - The server API version
    - + …
  - Docker –D info



```
$ docker -D info

Containers: 14
 Running: 3
 Paused: 1
 Stopped: 10
Images: 52
Server Version: 1.13.0
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: false
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: active
 NodeID: rdjq45w1op418waxlairloqbm
 Is Manager: true
 ClusterID: te8kdyw33n36fqiz74bfjeixd
 Managers: 1
 Nodes: 2
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
```

Installing

- Running containers as non-root:
  - Add username to “docker” group
    - sudo usermod -aG docker gebruikersnaam
  - Reboot

Important:
Some of the labs still use sudo.
This is NOT best-practice.
You should add your non-root user to the docker group.
And use this user for your implementations.

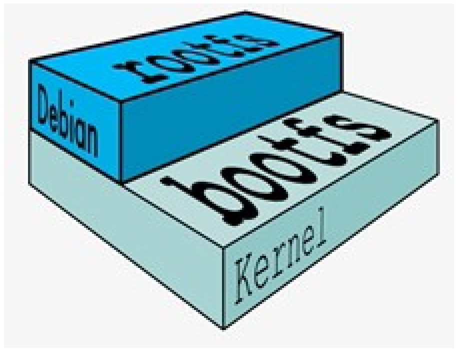
Docker
First Images & Containers

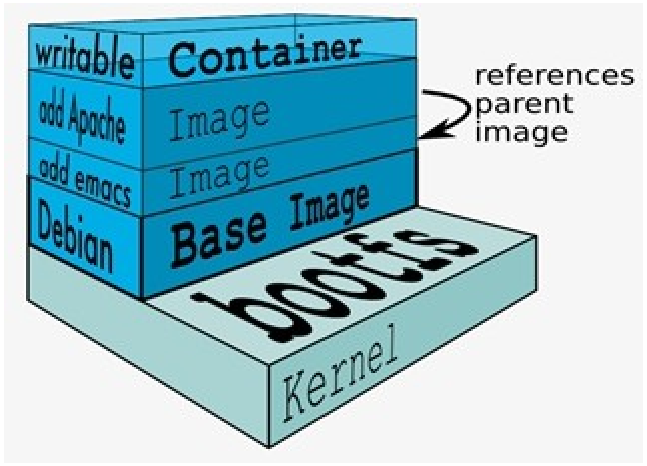
KdG Karel de Grote Hogeschool

/30/2020

First Images & Containers

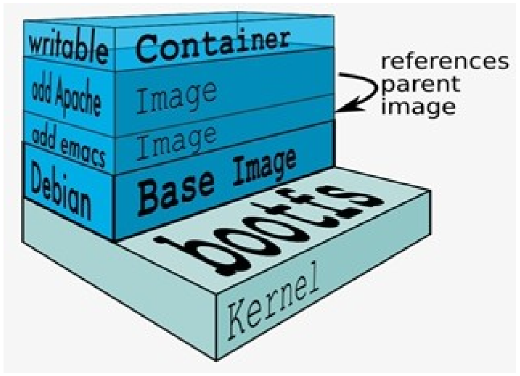
- Image:
  - Basic building blocks for containers
  - "Layered"
    - Base Image (e.g. Debian)
    - Extra Modules = extra images
  - Each image has own ID/name
  - Each image has own version (tag) ("latest")





First Images & Containers

- Container:
  - read-write layer
  - sits on (one or more) read-only images
  - When the container is run, the Docker engine
    - merges all of the required images together.
    - merges the changes from the read-write layer into the container
  - "commit" = merge changes = new layer on top of old layers
  - "start" = pull required image and parent images until base image is reached



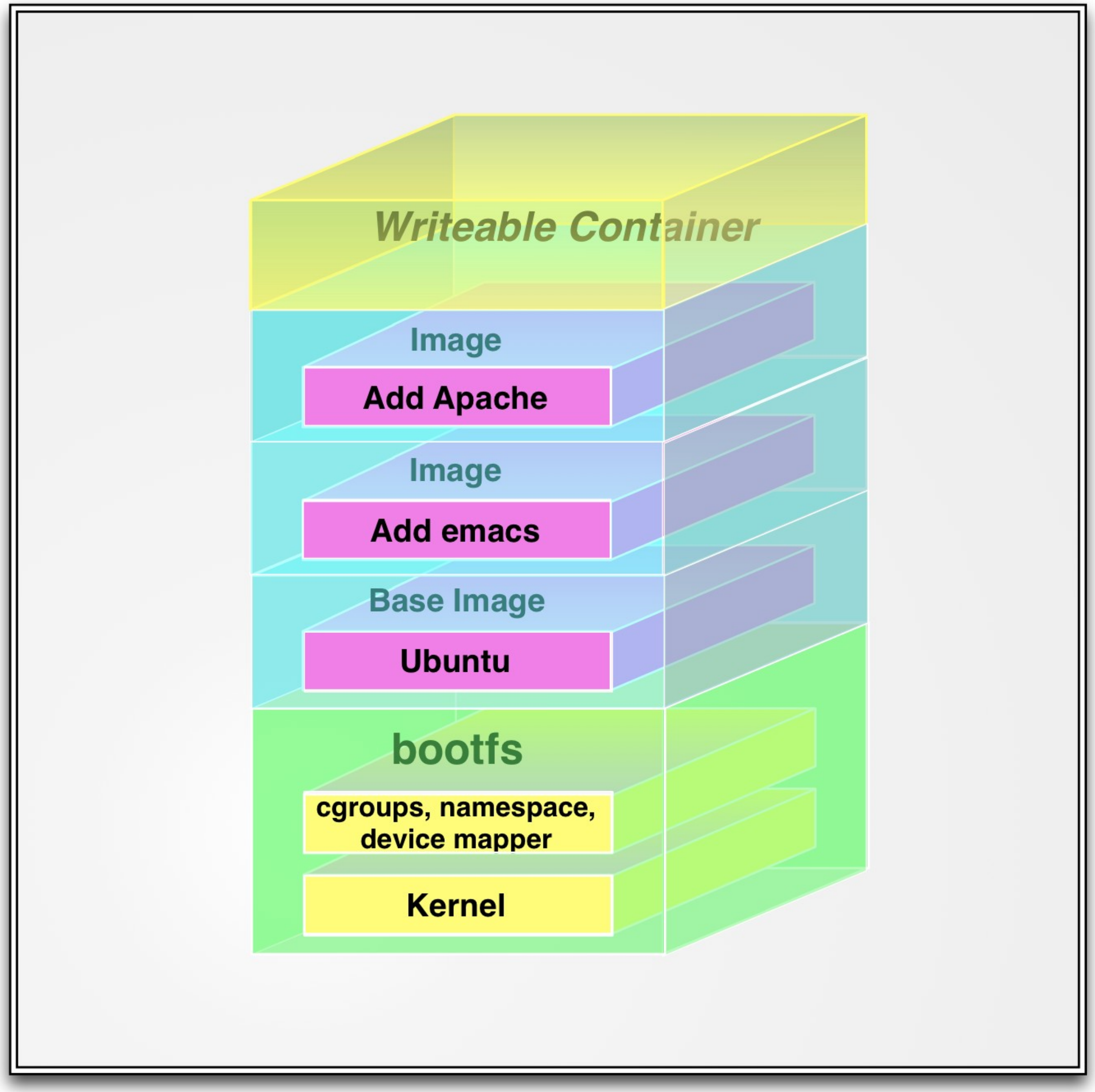
DockerFile

- Note: The docker commit command only commits the differences between the image the container was created from and the current state of the container. This means updates are very lightweight.

Docker Images

- Layered Filesystems
- Base = bootfilesystem = bootfs
  - when a container has booted, it is moved into memory, and the boot filesystem is unmounted to free up the RAM used by the initrd disk image.
- Rootfs
  - In Docker the root filesystem stays in read-only mode
- More read-only filesystems on top of rootfs
  - UNION mount
- All these filesystems are IMAGES
- Layered on top of each other.
  BASE = lowest, image below = PARENT

Docker Images

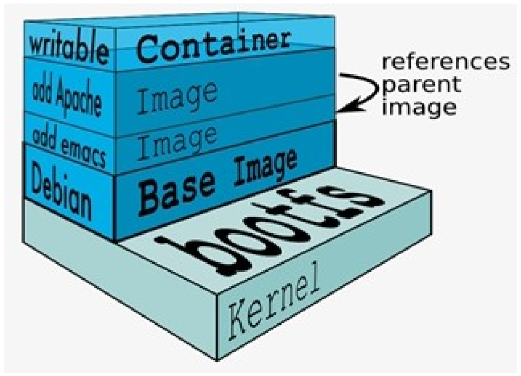


Docker Images

- Top layer
  - Read/write
  - Empty at start
  - File changed?
    - Copy from lower layer into RW layer
    - File is then changed
    - “Copy on Write”

First Images & Containers

- Containers:
  - `docker container ls -a`
    - Running containers
    - ID/Name
  - Isolated

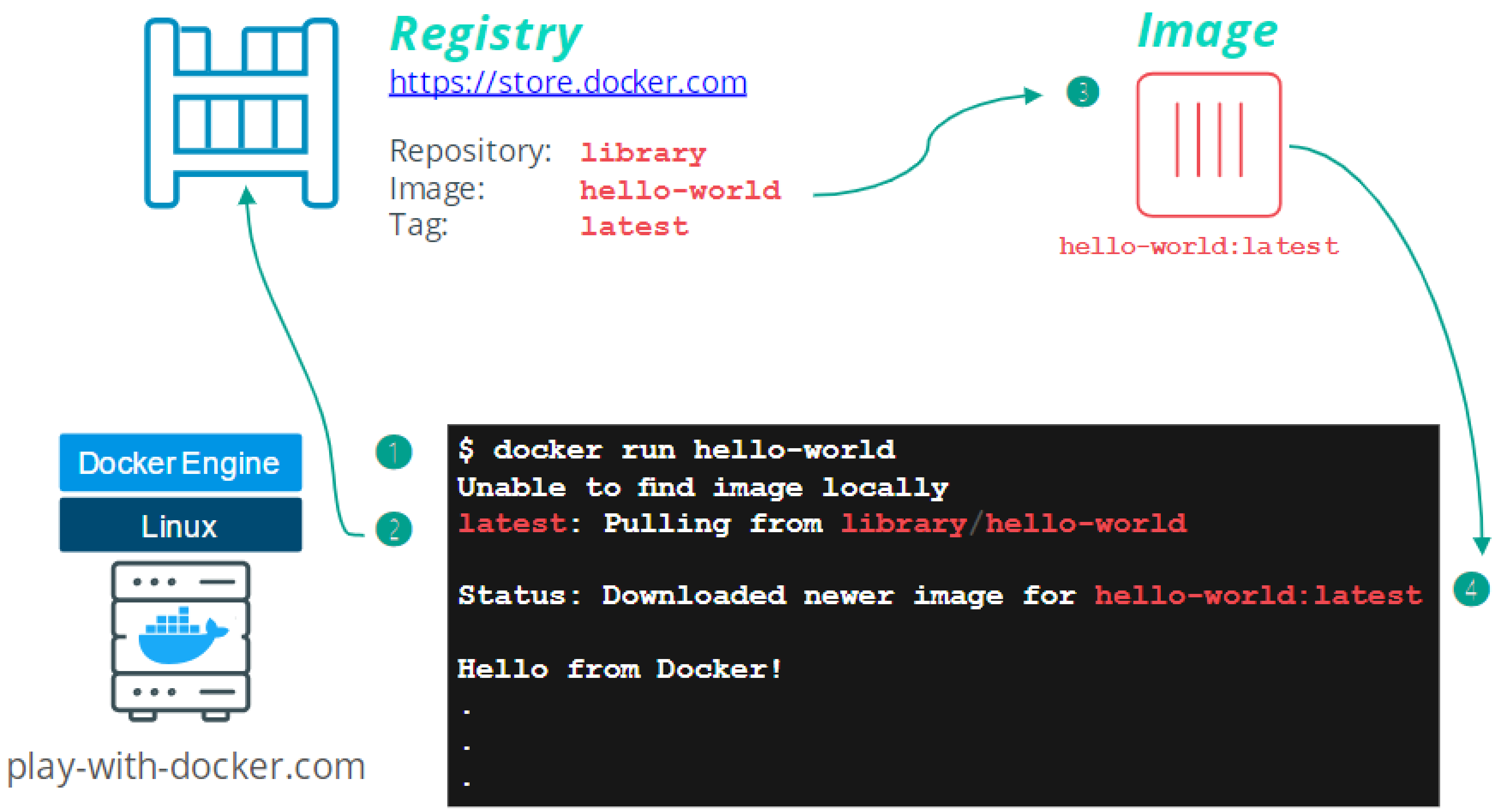


First Images & Containers

- Registry:
  - Registered images
  - Public/Private
  - “push” & “pull”
- Repository
  - Stored images
  - Part of the registry
  - Unique for each user account
    - App = helloworld
    - Username/namespace = itsme
    - Docker repository = itsme/helloworld

First Images & Containers

- Running a container:



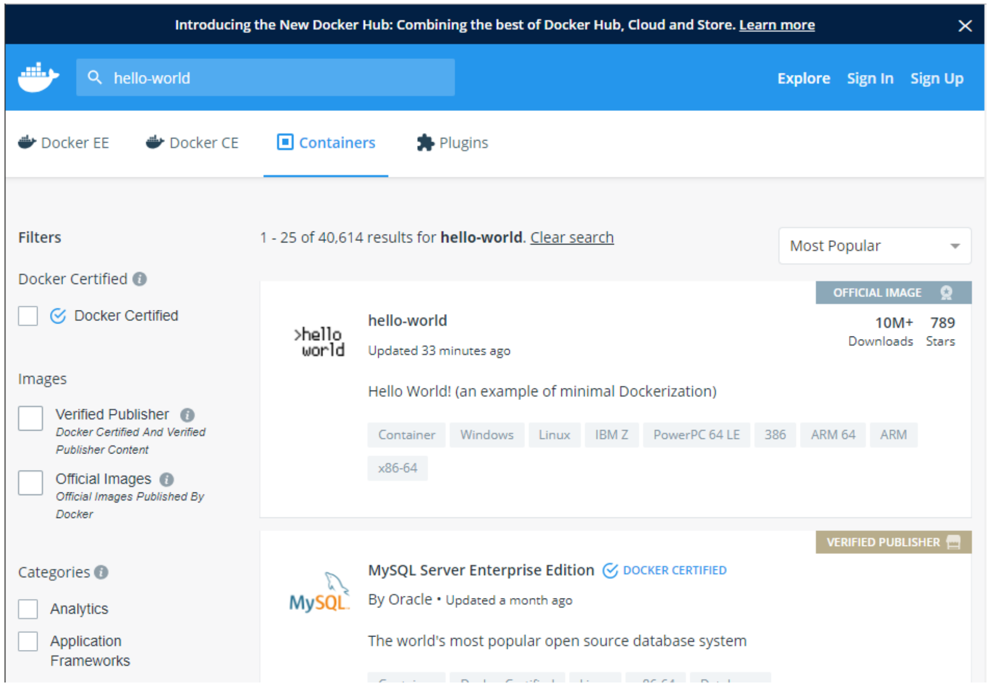
```
$ docker run hello-world
Unable to find image locally
latest: Pulling from library/hello-world

Status: Downloaded newer image for hello-world:latest

Hello from Docker!
.
.
.
```

First Images & Containers

- Docker registry
  - Default "pull" registry
  - https://hub.docker.com
    - Official Images (Published by Docker)
      - Verified
      - Certified
    - Others (Published by "public/companies/…")


Introducing the New Docker Hub: Combining the best of Docker Hub, Cloud and Store. Learn more

hello-world

Explore Sign In Sign Up

Docker EE · Docker CE · Containers · Plugins

Filters

Docker Certified
- Docker Certified

Images
- Verified Publisher — *Docker Certified And Verified Publisher Content*
- Official Images — *Official Images Published By Docker*

Categories
- Analytics
- Application Frameworks

1 - 25 of 40,614 results for **hello-world**. Clear search

Most Popular

OFFICIAL IMAGE
hello-world
Updated 33 minutes ago
10M+ Downloads · 789 Stars
Hello World! (an example of minimal Dockerization)
Container · Windows · Linux · IBM Z · PowerPC 64 LE · 386 · ARM 64 · ARM · x86-64

VERIFIED PUBLISHER
MySQL Server Enterprise Edition · DOCKER CERTIFIED
By Oracle • Updated a month ago
The world's most popular open source database system

Docker DockerFile

Dockerfile

- What?
- Simple Example
- Building & Running
- Commands

Dockerfile

- Filename: Dockerfile
- "Image-building" (literally)
- Automation
- Sequential Instructions
- Each instruction creates a different layer

How to create your custom image

- Download an existing image
  docker pull ubuntu
- Adapt the image with a Dockerfile
  mkdir mijnimage && cd mijnimage
- vi Dockerfile
  FROM ubuntu
  MAINTAINER Jan Celis
  RUN apt-get update && apt-get install -y apache2
- Build the new image
  docker build -t jancelis/ubuntu:apache .

Dockerfile

- Simple example:
  - Base-image selection (**FROM**)
  - Commands (**CMD**)

```
FROM busybox:latest
CMD echo Hello World!!
```

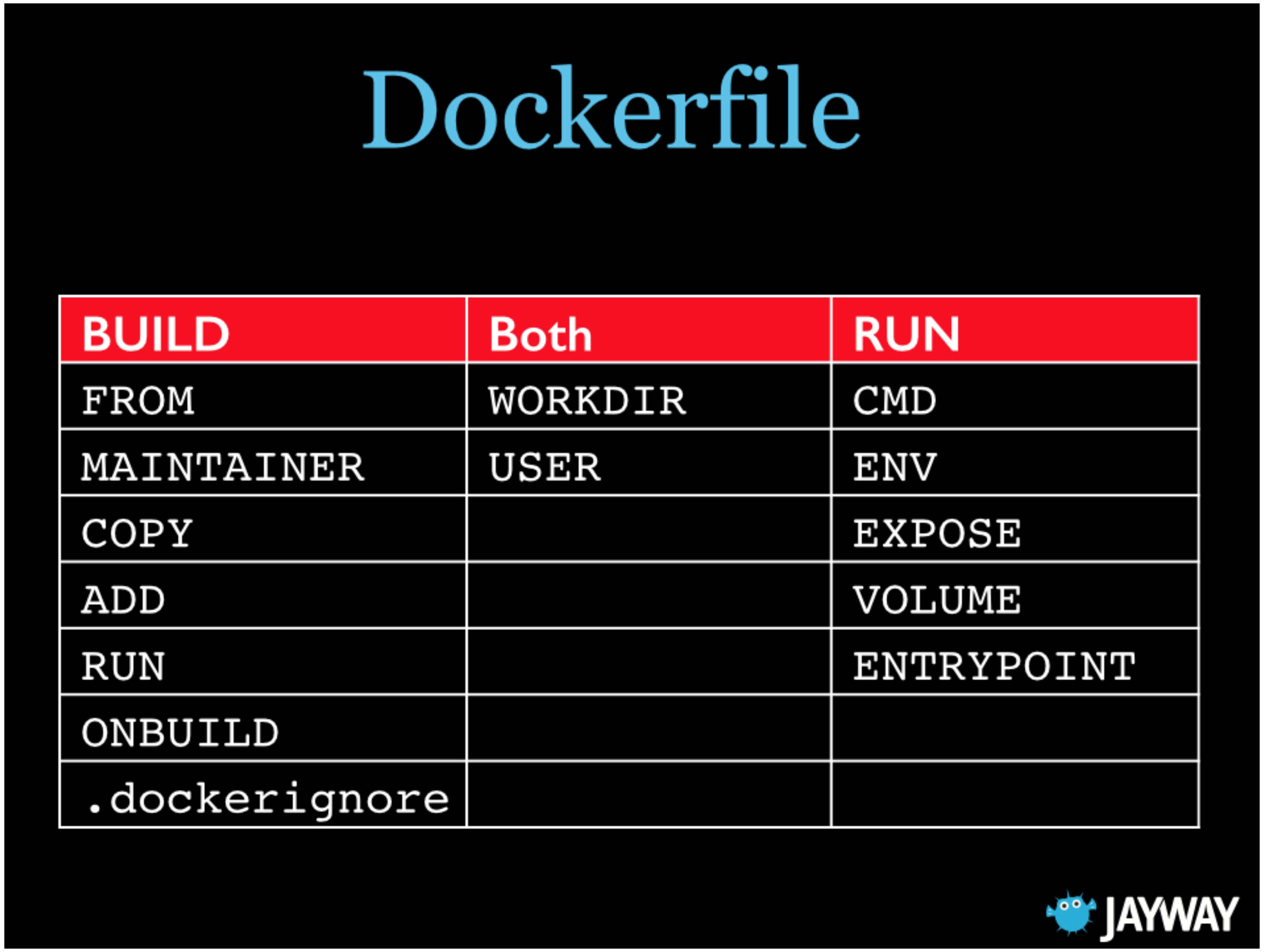
- Syntax:
  - Instructions (INSTRUCTION argument)
  - Comments (# at line-start or within line)
  - Empty lines (structure, are ignored)

2 FASES: Build and run

- Run "docker build":
  - docker build .
  - Remark: The directory . contains the Dockerfile !
- Run "docker run":
  - docker run your-containerID

Dockerfile

Commands and their relation to the 2 build-phases.



Dockerfile

BUILD	Both	RUN
FROM	WORKDIR	CMD
MAINTAINER	USER	ENV
COPY		EXPOSE
ADD		VOLUME
RUN		ENTRYPOINT
ONBUILD		
.dockerignore		

JAYWAY

Dockerfile FROM

- Base-image
- **FROM** <image>[:<tag>]
  - <image>: This is the name of the image which will be used as the base image
  - <tag>: This is the optional tag qualifier for that image. If no tag qualifier has been specified, the tag "latest" is assumed

- Example:
  FROM ubuntu:18.04

Dockerfile MAINTAINER (deprecated)

- Information
- **MAINTAINER** <author's detail>
  - Name
  - E-mail

- Example
  MAINTAINER Jan Celis jan.celis@kdg.be

  - Note: deprecated, use LABEL (see next)

  LABEL
  maintainer="peter.cornelissen@kdg.be"

Dockerfile LABEL

- Add Metadata
- **LABEL** <key>=<value>

- Example

```
LABEL author="Jan Celis <jan.celis@kdg.be>"
LABEL version="1.0"
LABEL description="This image description \
can span multiple lines."
```

Dockerfile RUN

- Runs commands in shell during **build** time
  - Default uses /bin/sh –c
- Shell Form
  - **RUN** <command>
  - Default uses /bin/sh –c
- Exec or JSON array
  - RUN ["<exec>", "<arg-1>", ..., "<arg-n>"]
    - <exec>: This is the executable to run during the build time.
    - <arg-1>, ..., <arg-n>: These are the (zero or more) number of arguments.
- Example:
  - **RUN** apt-get update && apt-get install apache2

Dockerfile CMD

- Runs commands during **launch** time (run of container)
  - Can be overridden by passing another command to the run instruction
  - Default uses /bin/sh –c
- Shell form
  - **CMD** <arg1> <arg2>… (as default parameters to ENTRYPOINT)
  - CMD <command> <arg1> <arg2>…
- Exec or JSON array
  - CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
  - CMD ["<exec>", "<arg-1>", ..., "<arg-n>"]
    - <exec>: This is the executable to run during the launch time.
    - <arg-1>, ..., <arg-n>: Arguments for executable. (No executable, ENTRYPOINT)
- Example
  CMD echo hello world !

Dockerfile ENTRYPOINT

- Runs commands/app during **launch** time (run of container)
  - Default uses /bin/sh –c
  - Can be used to change the default /bin/sh –c
- Shell Form
  - **ENTRYPOINT** <command>
- Exec or JSON array
  - ENTRYPOINT ["<exec>", "<arg-1>", ..., "<arg-n>"]
    - <exec>: This is the executable to run during the launch time.
    - <arg-1>, ..., <arg-n>: Arguments for executable.
- Example
  ENTRYPOINT /bin/echo
  CMD "hello world"

Dockerfile - ENTRYPOINT

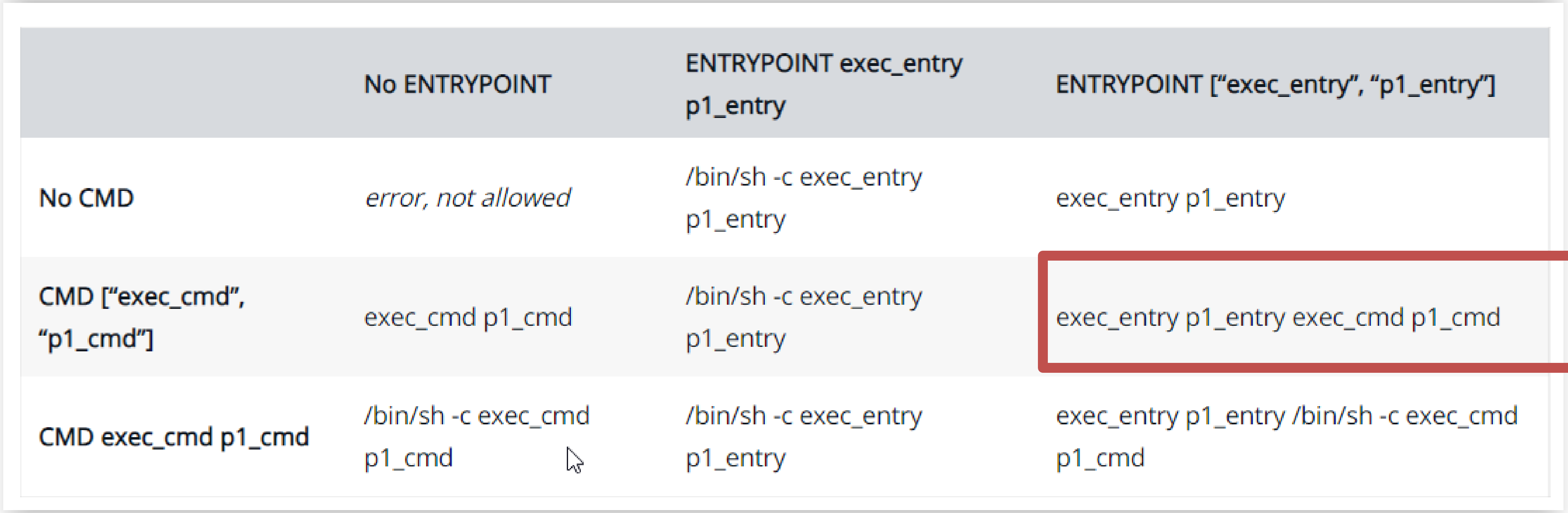
- Runs commands/app during launch time = execution of container
- End of launched app <=>end of container
- Run command arguments will be passed as extra arguments
- Override can be done via --entrypoint option during run

Dockerfile – RUN/CMD/ENTRYPOINT

- CMD/ENTRYPOINT
  - Syntactically, you can have more than one of these instructions in a Dockerfile.
  - However, the build system will ignore all the instructions except the last one.
- RUN
  - Every RUN builds a new layer. So command-chaining is useful.

Dockerfile CMD/ENTRYPOINT

- Override of ENTRYPOINT can be done via --entrypoint option during run
- Syntactically, you can have more than one of the CMD/ENTRYPOINT instructions in a Dockerfile.



	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT [“exec_entry”, “p1_entry”]
No CMD	*error, not allowed*	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD [“exec_cmd”, “p1_cmd”]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Dockerfile COPY

- **COPY** <src> ... <dst>
- COPY ["<src>",... "<dest>"]
  - <src>: This is the source directory, the file or the directory from where the docker build subcommand was invoked
  - ...: This indicates that multiple source files can either be specified directly or be specified by wildcards
  - <dst>: This is the destination path for the new image. It must end with a slash /
  - Use absolute paths if possible

- Example
  COPY ./website/*.htm /var/www/

Dockerfile ADD

- **ADD** <src> ... <dst>
- ADD ["<src>",... "<dest>"]
  - <src>: This is the source directory, the file or the directory from where the docker build subcommand was invoked. Source can be a **TAR** or remote **URL** !
  - ...: This indicates that multiple source files can either be specified directly or be specified by wildcards
  - <dst>: This is the destination path for the new image. It must end with a slash /
  - Use absolute paths if possible
- Example
  ADD ./website/website.tar /var/www/

Dockerfile ENV

- Sets environment variable in the new image
- **ENV** <key>=<value>
  - <key>: This is the environment variable
  - <value>: This is the value that is to be set for the environment variable

- This is used to set image options
  - Example (Official mysql image)
    ENV MYSQL_ROOT_PASSWORD=“Secret007”

Note: Old syntax: ENV <key> <value>

Old syntax can still be present in examples/documentation.
Change where/when necessary.

Dockerfile VOLUME

- Creates a mountpoint
  - Used as an external mount to native system
- Shell form
  - **VOLUME** <mountpoint>
- Exec or JSON array (all values must be within doublequotes ("")):
  - VOLUME ["<mountpoint>"]

- Example
  VOLUME /var/log
  VOLUME ["/var/log"]

Dockerfile EXPOSE

- "Opens" container ports (Documentation)
- **EXPOSE** <port>[/<proto>] [<port>[/<proto>]...]
  - <port>: This is the network port that has to be exposed to the outside world.
  - <proto>: Optional field provided for a specific transport protocol, such as TCP and UDP. Default TCP is assumed
- Example
- **EXPOSE** 8080

Dockerfile WORKDIR

- Changes the pwd (present or current working directory)
  - Default = /
  - Absolute or relative to previous
- **WORKDIR** <dirpath>
- Example

```
RUN mkdir -p /scripts
RUN echo 'echo helloworld' > /scripts/hello.sh
WORKDIR /scripts
```

Dockerfile USER

- Sets startup user in the new image
  - Default = root

- **USER** <user>[:<group>]
- **USER** <UID>[:<GID>]
  - <UID>: This is a numerical user ID
  - <GID>: This is a valid groupID

- Example:

```
USER webadmin:webgroup
USER 1008:1200
```

Dockerfile USER Example

- User has to be created
  - FROM ubuntu
  - RUN groupadd user1
  - RUN useradd -r -u 1001 -g user1 user1
  - **USER** user1
  - CMD echo Hello World from user $(whoami) !

Dockerfile ONBUILD

- The ONBUILD instruction registers a build instruction to an image and this is triggered when another image is built by using this image as its **base image**.
- **ONBUILD** <INSTRUCTION>
  - <INSTRUCTION> is another Dockerfile build instruction, which will be triggered later.
- Example: Build python runtime and then continue
  ONBUILD ADD . /app/src
  ONBUILD RUN /usr/bin/python-build --dir /app/src

Dockerfile Optimize

- **.dockerignore**
- The .dockerignore is a newline-separated TEXT file, wherein you can provide the files and the directories which are to be excluded from the build process.
- The exclusion list in the file can have both the fully specified file or directory name and the wild cards.
- Example

```
cat .dockerignore
#comment
/temp*
PASSWORDFILE
```

Dockerfile Cache

- Successful steps are "cached"
- If Dockerfile is changed, the build can start from the last successful step.
- If you don’t want this: **--no-cache** flag with build command
  - For example you can use this to force an apt-get update at every build
  - Example next slide

Dockerfile ENV REFRESHED_AT

- If needed, make sure the cache is “hit” early in the build process, for example by **changing a date** in your DockerFile
- Example:
- FROM ubuntu:18.04
- MAINTAINER Jan Celis "jan.celis@kdg.be"
- ENV **REFRESHED_AT** 2020-02-29
- RUN apt-get -qq update
- This makes sure that the update command is executed.

References

- Dockerfile Best Practice
  - https://docs.docker.com/develop/develop-images/instructions/
- Full Reference:
  - https://docs.docker.com/engine/reference/builder

Docker Publishing

Karel de Grote Hogeschool

11-2022

Dockerfile

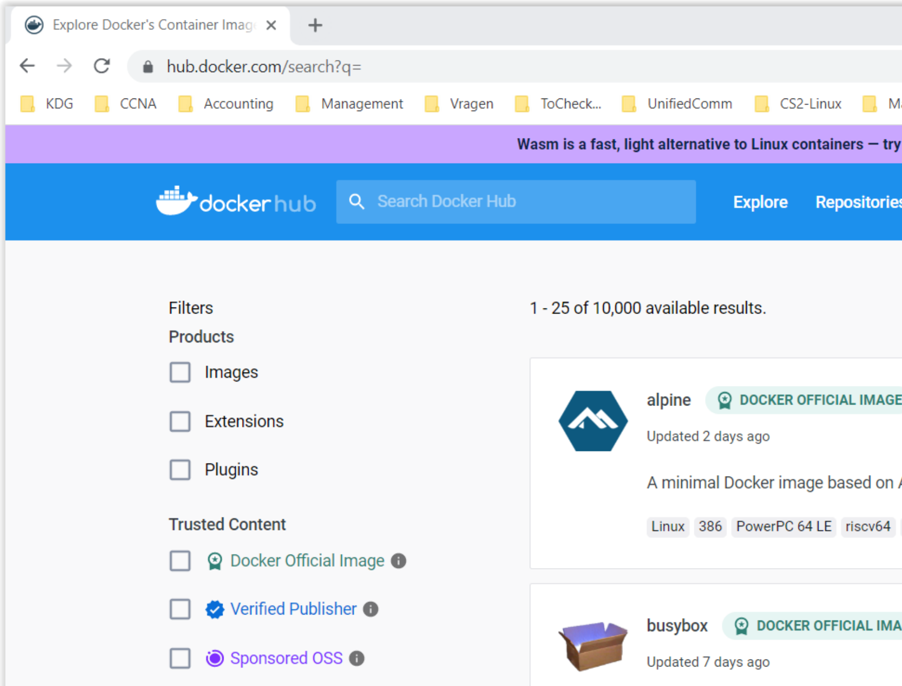
- Remember…
- Overview
- Publishing
- Automated Builds

See before: First Images & Containers

- Locally → see previous slides/labs
- Docker Hub → used in previous labs
- Own Registry

See before: First Images & Containers

- Docker registry
  - Default "pull" registry
  - https://hub.docker.com
    - Official Images (Published by Docker) = "trusted content"
      - Verified
      - Certified
    - Others (Published by "public/companies/…" = Publisher Images + certified)
      - Verified
      - Sponsored



Explore Docker's Container Imag ×

hub.docker.com/search?q=

KDG CCNA Accounting Management Vragen ToCheck... UnifiedComm CS2-Linux M

Wasm is a fast, light alternative to Linux containers – try

docker hub Search Docker Hub Explore Repositories

Filters

Products

- Images
- Extensions
- Plugins

Trusted Content

- Docker Official Image
- Verified Publisher
- Sponsored OSS

1 - 25 of 10,000 available results.

alpine DOCKER OFFICIAL IMAGE
Updated 2 days ago
A minimal Docker image based on
Linux 386 PowerPC 64 LE riscv64

busybox DOCKER OFFICIAL IMA
Updated 7 days ago

Overview

1. Private Docker Infrastructure
   - Setup on local infrastructure
   - Open source: https://github.com/docker/distribution

2. Public Docker Infrastructure
   - Public registry/repositories
     - Free (note: 1 private possible)
     - Docker Personal
   - Private registry/repositories
     - Paid service
       - Docker Pro (unlimited private repositories)
       - Docker Team
       - Docker Business

Public

- Public = “Docker Hub”
- Steps:
  - Sign up for docker account
  - Verify e-mail
  - Login to Docker Hub

Publishing

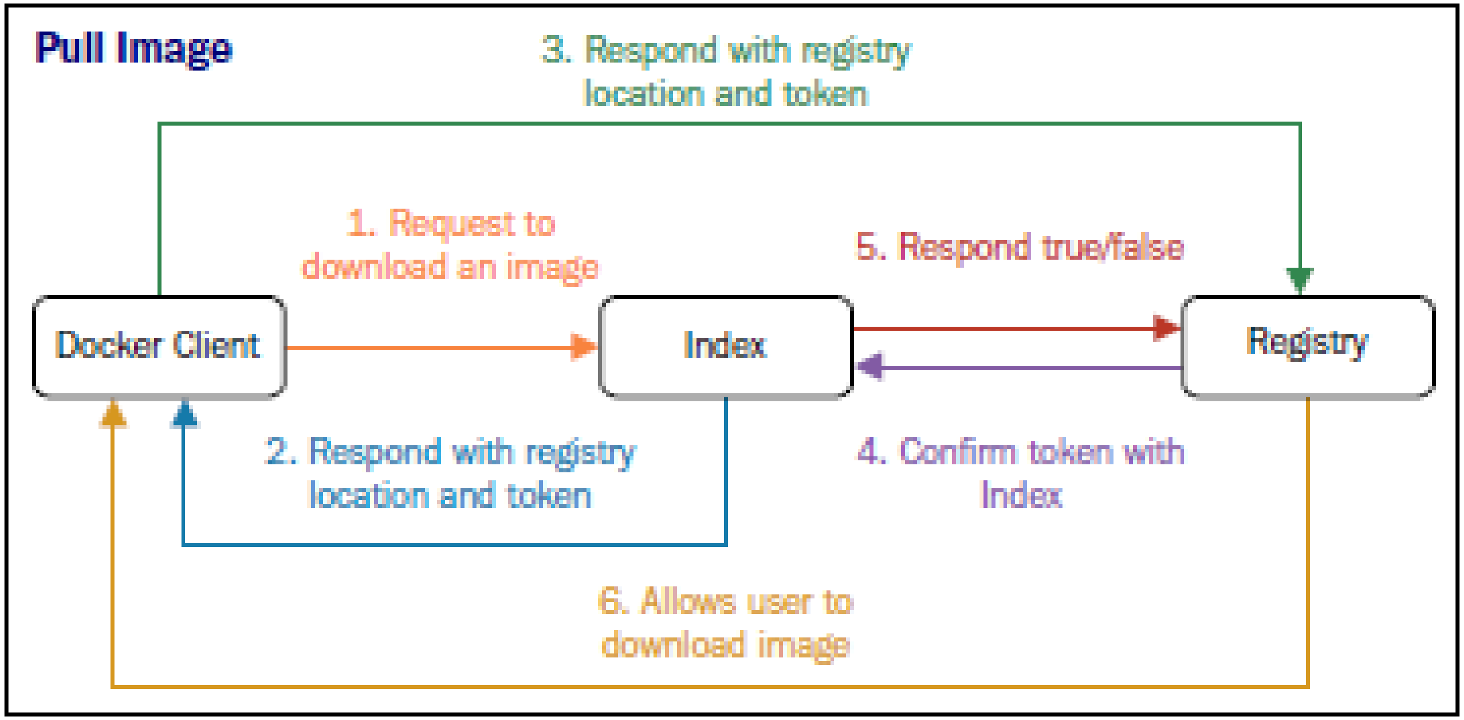
- Docker Hub
  - WebAccess
  - Console Access (CLI)

Publishing

- Docker Hub
  - Searching for images
    - $ docker search centos
  - Getting images = "PULL"
    - Build (see before…) → put Docker username in image or "tag" your image with your username.
      - docker build -t <your_username>/my-first-repo .
    - Pull
      - $ docker pull centos
    - Run
  - Putting images = "PUSH"
    - Commit
    - Push
      - docker push <your_username>/my-first-repo
      - docker push ip_local_server:port_local_server/<your_username>/my-first-repo
  - Verify image availability
    - Web GUI or Console (search)

Publishing

- Docker Hub
  - Searching for images
    - $ docker search centos
  - Getting images = "PULL"


Pull Image
3. Respond with registry location and token
1. Request to download an image
5. Respond true/false
Docker Client
Index
Registry
2. Respond with registry location and token
4. Confirm token with Index
6. Allows user to download image

Automated Builds

- GitHub
- Automated builds
  - are supported on both private and public repositories of GitHub and Bitbucket.
- The Docker Hub Registry:
  - keeps all the automated build images.
  - is based on open source and can be accessed from https://github.com/docker/distribution
- Webhooks:
  - Trigger actions after a successful push to a repository to integrate Docker Hub with other services.

Docker Volumes

Karel de Grote Hogeschool

Volumes

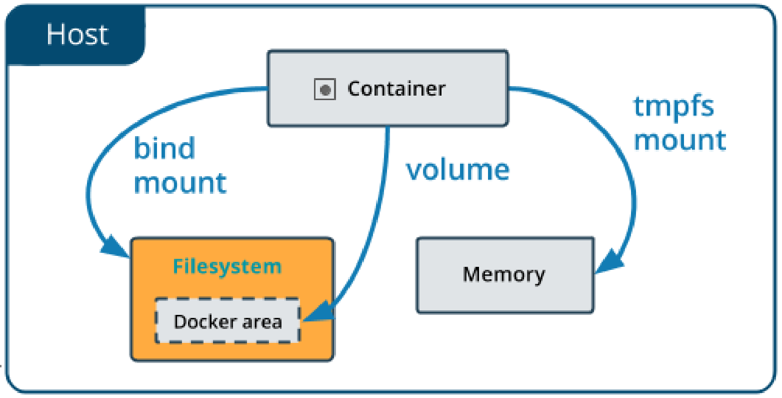
- Data & Persistence
- Volumes
- Share with Host (Bind Mount)
- Data Volume Container = deprecated
- Pitfalls
- Extra: mount
- Dockerfile

Volumes: Data & persistence

- Containers are temporary in nature:
  - Exist as long as application lives
  - Upgrades, malfunctions, changes …. → container is deleted
  - “Persistent data” is not preserved
    - Part of the “Union File System”
- Need to preserve data files:
  - Databases
  - Logs
  - …
    - Part of the Docker Host’s filesystem

Volumes: Data & persistence

- Docker has 3 ways to persist data from containers:
- Volume:
  - Specific part of the host filesystem (/var/lib/docker/volumes)
  - Can't be reached by non-docker processes
- Bind Mount
  - Anywhere on host system.
  - So can be reached by non-docker processes
- Tmpfs Mount
  - Memory only
  - No write to fs

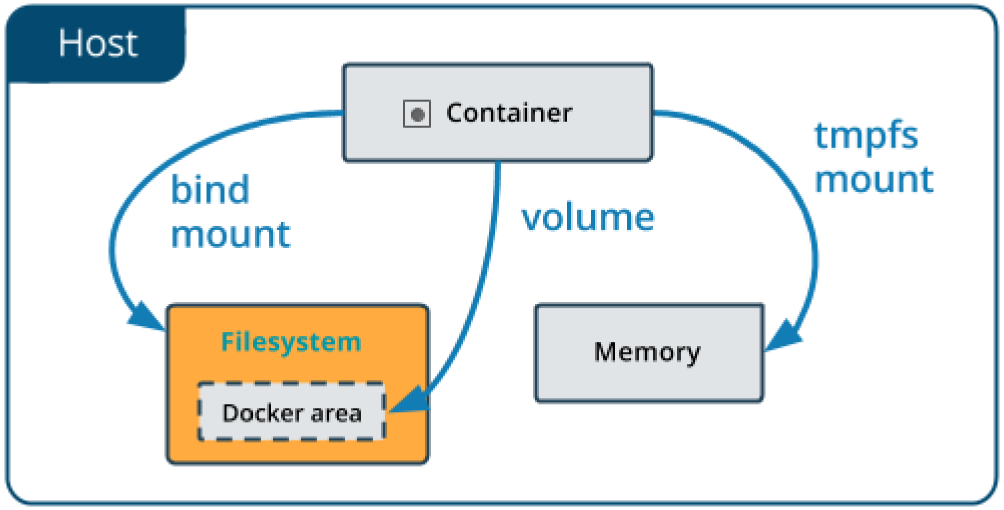


Volumes: Data & persistence

- Volume:
  - Share data between running containers
  - No knowledge needed about host-filestructure (/var/lib/docker/volumes)
  - Store remote (host, cloud, …) instead of locally
  - Explicit need for backup/restore/migration
- Bind Mount
  - Share config/source code between host & container
- Tmpfs Mount
  - Don’t persist = security / performance

Volumes: Volumes

- Use Volumes over bind/tmpfs:
  - Easier to back up or migrate.
  - Manage volumes using Docker CLI commands or the Docker API.
  - Work on both Linux and Windows containers.
  - More safely shared among multiple containers.
  - Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
  - Can have their content pre-populated by a container.



Volume

- Practical (creation)
  - Explicit
    - “docker volume create [OPTIONS] [VOLUME]”
  - Docker Run:
    - docker run -v [VOLUME_NAME:]CONTAINER_PATH IMAGE_NAME
- Part of “Docker Host” filesystem
  - "/var/lib/docker/vfs/dir/737e0355c5d81c96a99d41d1b9f54 0c2a212000661633ceea46f2c298a45f128”
  - "/var/lib/docker/volumes”
- Not part of UFS
  - Host directory gets mounted:
  - dev/disk/by-uuid/721cedbd-57b1-4bbd-9488-ec3930862cf5 on /MountPointDemo type ext3 (rw,noatime,nobarrier,errors=remount-ro,data=ordered)

Volume

- Practical (deletion)
  - Run:
    - Run “docker volume rm [OPTIONS] VOLUME [VOLUME...]”
    - https://docs.docker.com/edge/engine/reference/commandline/volume_rm/

Data Volume

- Docker Volume commando
  - Subcommands:
    - Create (make)
    - Inspect (show info)
    - Ls (list all)
    - Rm (remove)

Share with Host (Bind Mount)

Note: -v = --volume

- Exposing a specific host directory (host FS)
  - docker run -v <host path>:<container mount path> image
  - docker run -v <host path>:<container mount path>:<read write mode> image
    - Mode = ro or rw
- Extra:
  - If host path does not exist, it will be created.
- Warning:
  - Dependent on host's directory structure
  - Data directories can/will leave a (big) footprint
  - Manually remove!
- Practical Examples:
  - Share log files and place them on host's filesystem
  - Share websites between different container webservers
  - …

Pitfalls

- Directories on host are not automatically removed.
  - Solution: explicitly use rm –v in run
- Issue: volumes created during auto-generated containers:
  - Problem: no idea which volumes are created
  - Solutions
    - Use docker rm –v → proactive
    - Keep record of created volumes → reactive
    - Docker inspect = check data volume associated with image → reactive
    - Remove ALL volumes → super reactive ☺
      - docker volume rm $(docker volume ls –a)

Volume Extra: mount

Extra run option that can be used:

--mount:

- key-value pairs <key>=<value>
- separated by commas
- Keys:
  - type: The type of the mount (bind, volume, or tmpfs)
  - source/src: name of the volume. For anonymous volumes, this field is omitted.
  - destination/dst/target: path where the file or directory is mounted in the container.
  - readonly: option, if present, causes the bind mount to be mounted into the container as read-only.
  - volume-opt option: can be specified more than once, takes a key-value pair consisting of the option name and its value.

Volume: Dockerfile

Remember?

VOLUME command in Dockerfile:

- e.g. VOLUME /mymountpoint
- Creates a mount-point (directory) within the container
- Can be linked to when starting the container with –v or –volume option
  - Link can be docker volume
  - Link can be host directory (bind mount)
- If not linked, the directory points to a directory within the docker filesystem.
  - /var/lib/docker/volumes/8c2339b16d43663ad597fc1eab8cd65f2c5ae44f6e7269028354d40619d1183f/_data

Volume: DEMO

- Aanmaken volume
- Gebruiken van directory in WebServer
- Gebruiken in 1 container
- Gebruiken in 2 containers
- Verwijderen = lukt niet?
- Kijken aan welke container deze gelinkt is?

Docker Networking

Karel de Grote Hogeschool

Networking

- Networking
- IP
- Ports
- Ports & Dockerfile

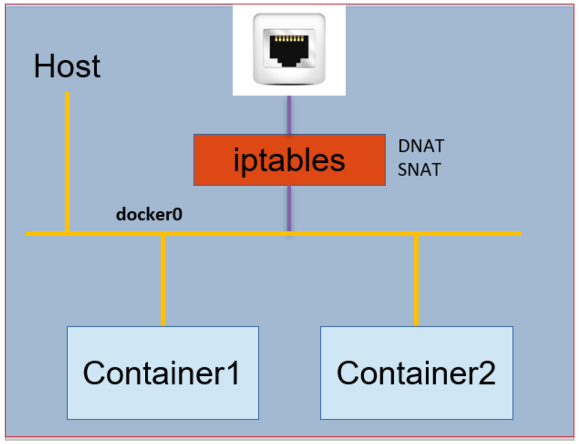
Networking

- Knowledge
  - Running containers (attach/detach)
  - Interactive containers (open console; run -i)
  - Detached/Daemonised containers (services; run –d)
- Upto now:
  - “Implementation view”
  - No “client/server” connections
    - Networking needed

Networking

- Layer3:
  - IP address
  - Port
  - = Socket

Networking - IP



- Connectivity
  - On Docker host
    - Virtual interface: docker0
    - Selects private range & assigns IP address
    - Private IP range: 172.17.0.0 to 172.17.255.255
    - Command to see networks: “docker network ls”
  - All containers automatically get IP address
    - Bridged
    - Range within selected private range: Private IP range: 172.17.0.0 to 172.17.255.255
    - Command to see networks: “docker network inspect bridge”
- Customisation:
  - Docker command
  - /etc/docker/ → daemon.json

Daemon.json

```
{
  "bip": "192.168.1.5/24",
  "fixed-cidr": "192.168.1.5/25",
  "fixed-cidr-v6": "2001:db8::/64",
  "mtu": 1500,
  "default-gateway": "10.20.1.1",
  "default-gateway-v6": "2001:db8:abcd::89",
  "dns": ["10.20.1.2","10.20.1.3"]
}
```

Retrieval - IP

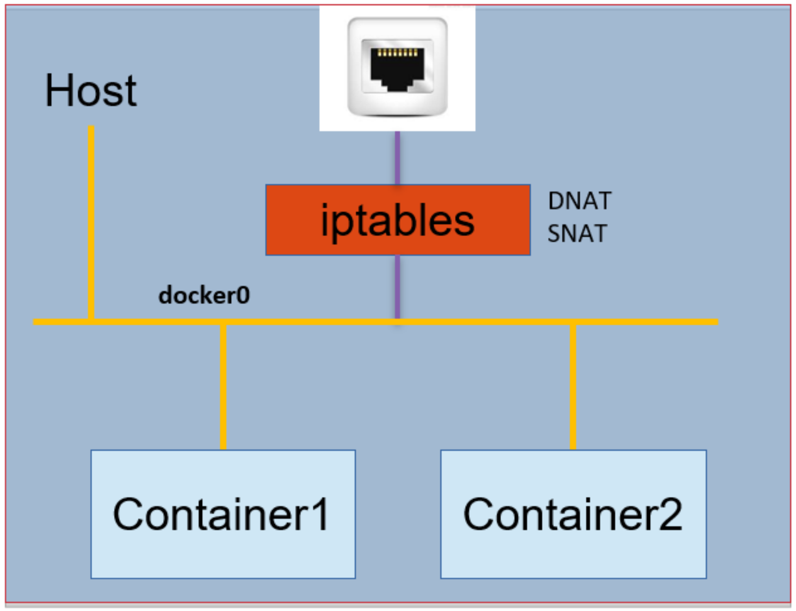
- Interactive container
  - Easy: ifconfig, ip addr show
- Detached container
  - No shell…
  - Docker inspect
    - “NetworkSettings”
      - Lots of information (see next slide)
    - Simply retrieve ip-address
      - $ sudo docker inspect --format='{{.NetworkSettings.IPAddress}}' 4b0b567b6019

Retrieval - IP

```
"NetworkSettings": {
"Bridge": "docker0",
"Gateway": "172.17.42.1",
"IPAddress": "172.17.0.12",
"IPPrefixLen": 16,
"PortMapping": null,
"Ports": {}
},
```

- Bridge: This is the bridge interface to which the container is bound
- Gateway: This is the gateway address of the container, which is the address of the bridge interface as well
- IPAddress: This is the IP address assigned to the container
- IPPrefixLen: This is the IP prefix length, another way of representing the subnet mask
- PortMapping: This is the port mapping field, which is now being deprecated, and its value is always null
- Ports: This is the ports field that will enumerate all the port binds.

Networking - IP



- Host ←→ container: OK
- Container ←→ container: OK
- Container → Internet: OK
- Outside world client → Container Service: NOK
  - Private address space
  - No ports exposed

- So public address space with ports needed.
  - Iptables functionality (linux)
  - Exposed ports (docker run)

Networking - Ports

- Docker run –p
- <containerPort>
  - Autogenerated = system choses host port
- <hostPort>:<containerPort>
- <ip>:<hostPort>:<containerPort>
- <ip>::<containerPort>
  - Ip=host ip

Ports are unique → can’t spin up containers using same port

Note: extra networking options:

--net: select chosen network e.g. –-net=bridge

--ip: set ip address container e.g. –-ip=“172.17.17.3”

--mac: set mac address interface e.g. --mac="02:42:ac:11:00:02“

--dns: set dns server e.g. --dns=“8.8.8.8”

Networking - Ports

Under the hood…

Docker run –p 80:80 apache2

- Iptables does dynamic NAT (DNAT)
  - All tcp source addresses 0.0.0.0/0
  - Can route to all destination addresses 0.0.0.0/0
  - And local port 80 is forwarded to container port 80

Networking - Ports

Retrieve ports:

- sudo docker ps
- docker inspect (with container ID)
  - "ExposedPorts"
  - "PortBindings"
  - "NetworkSettings"
    - Note: format through filter command
- docker port (with container ID)

Networking – Ports & Dockerfile

- 3rd party images will/must define port to use
- Build: through Dockerfile
- EXPOSE command
  - Explicitly define the port that the image will expose when the container is ran.
- Docker run –P
  - No other arguments possible
  - Automatically use the exposed port on host
  - Auto-assignment of host-port is used

Docker Swarm

KdG Karel de Grote Hogeschool

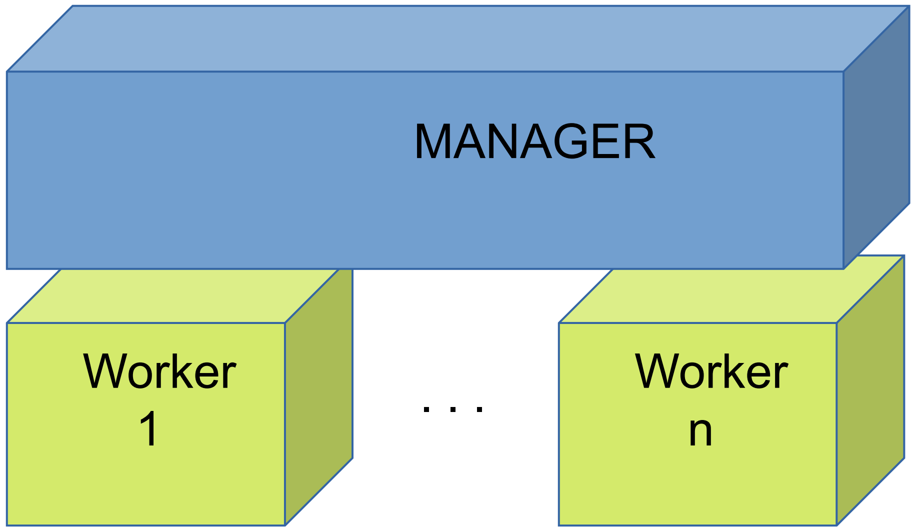
/17/2021

Swarm

- Werking
- Swarm met virtuele machines
- Swarm met docker in docker

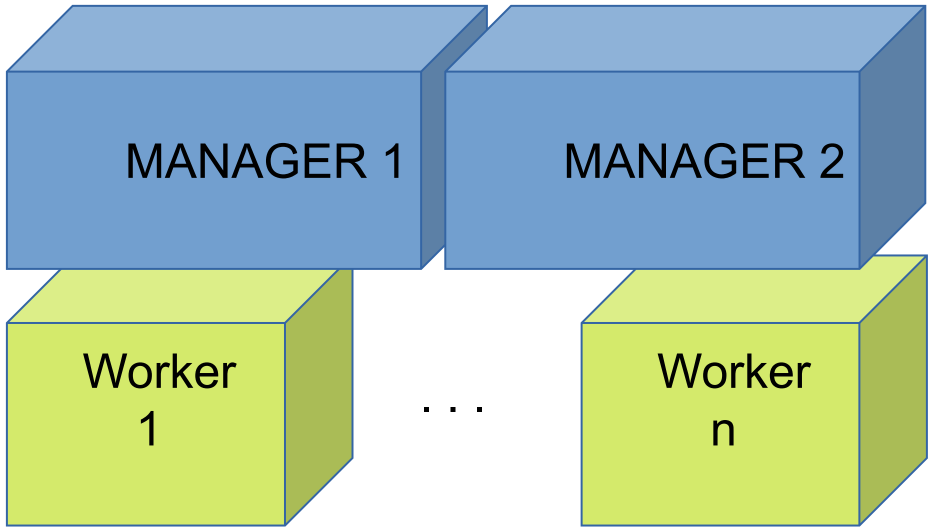
Werking Swarm

- Minstens 1 Manager (Leader genoemd)
  - Een Manager kan ook mee helpen als worker
- Worker(s)



Werking Swarm

- De Manager verdeelt taken (services) tussen de workers, en start indien nodig nieuwe workers op

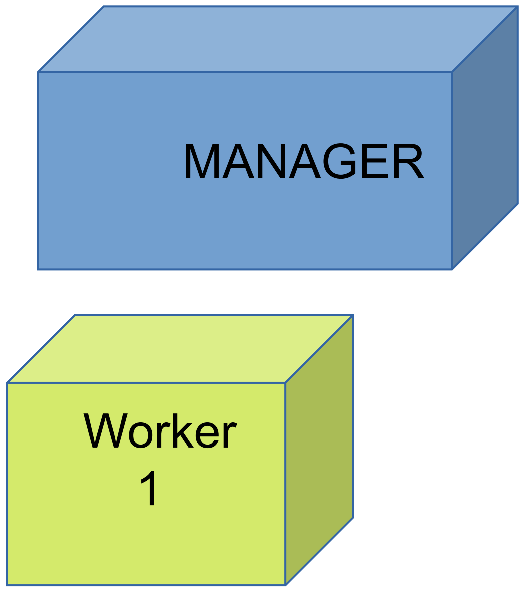


Opstarten swarm manager

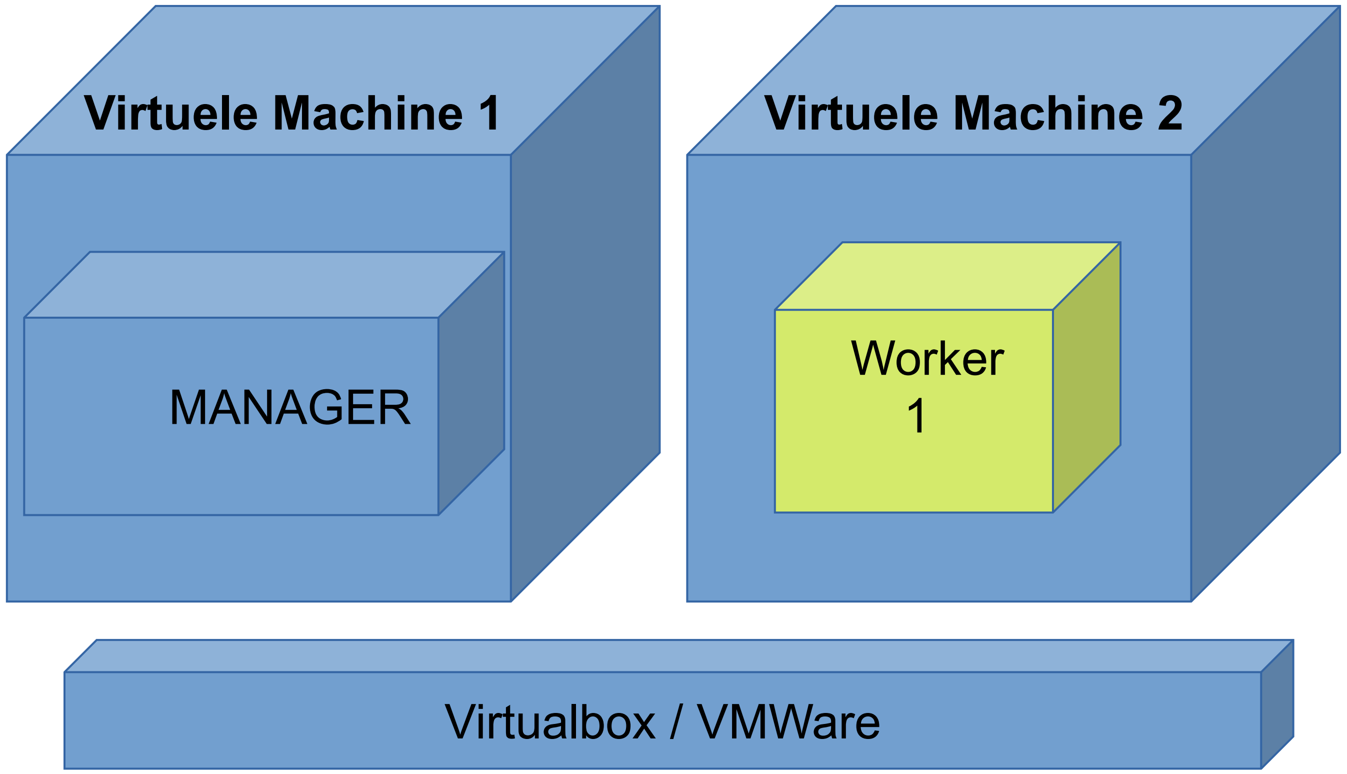
- Commando:
- **docker swarm init**
- De computer/container wordt manager
- Maakt een token waarmee workers kunnen worden toegevoegd aan de swarm

Toevoegen worker

- Commando:
- **docker swarm join --token <token>**

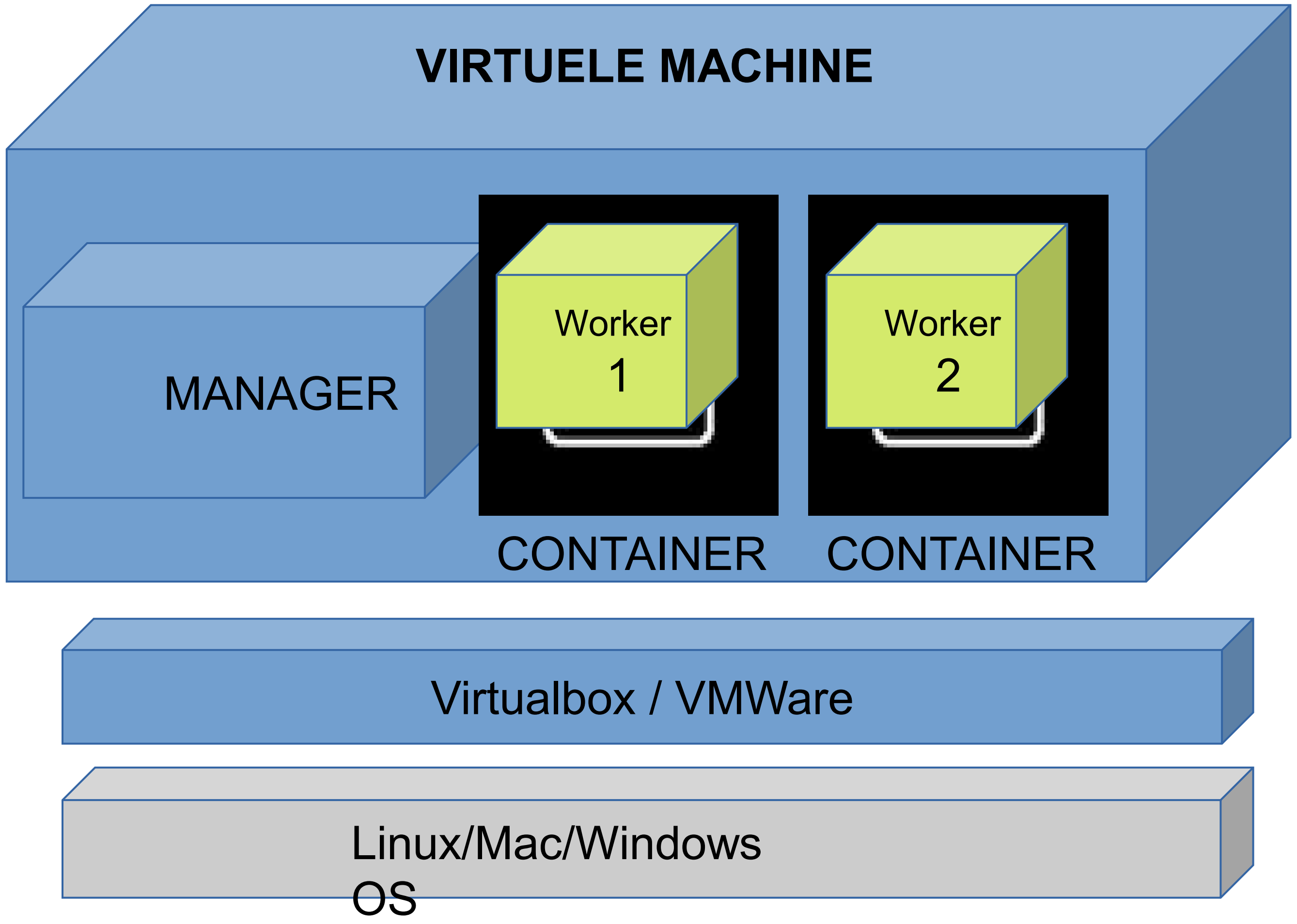


Swarm met virtuele systemen

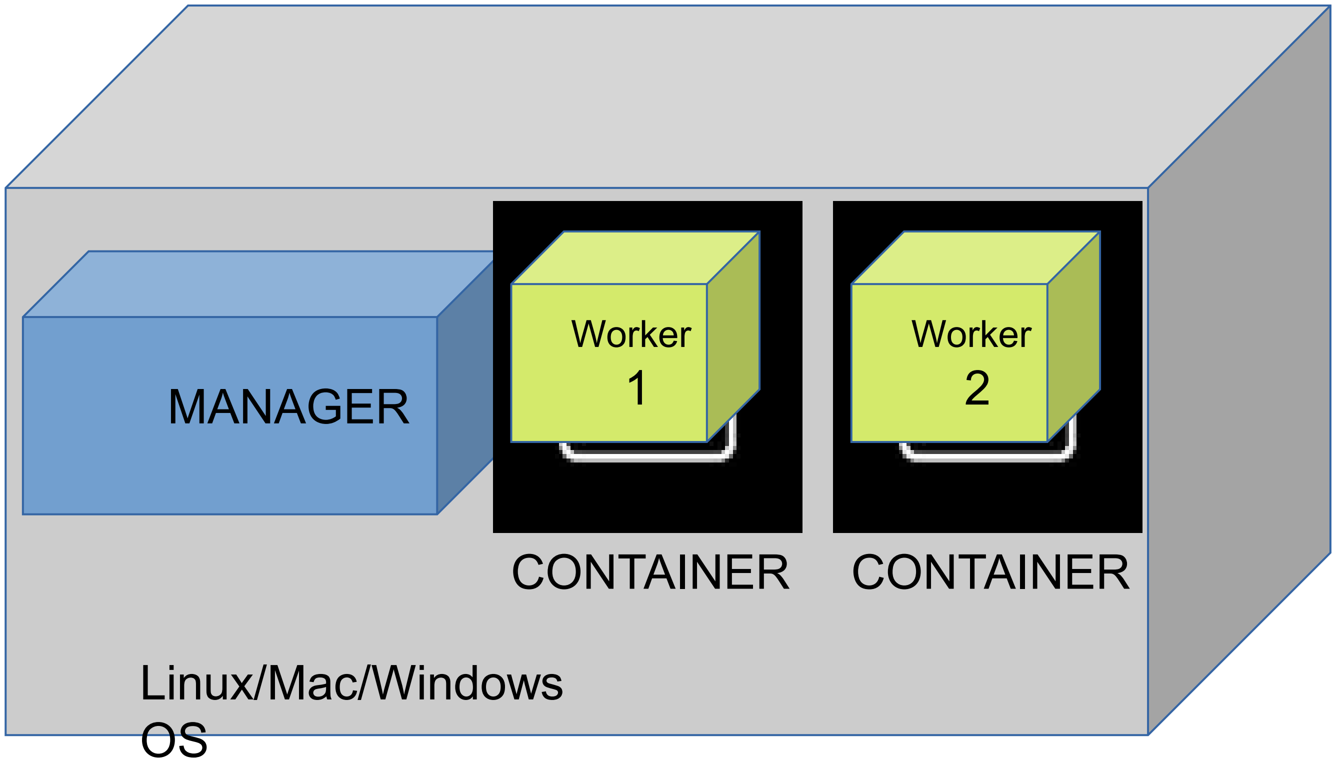


Het tooltje docker-machine kan automatisch nieuwe VM's aanmaken met managers/containers

Swarm met docker in docker in VM



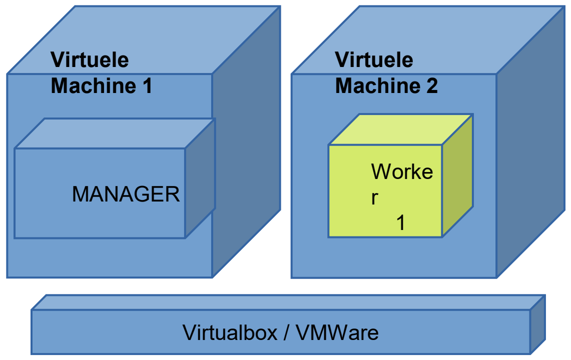
Swarm met docker in docker Native



Docker in docker images krijgen het achtervoegsel dind:
Bv image docker:18.09 met docker in docker heet:
  image docker:18.09-dind

Netwerk bij Swarm

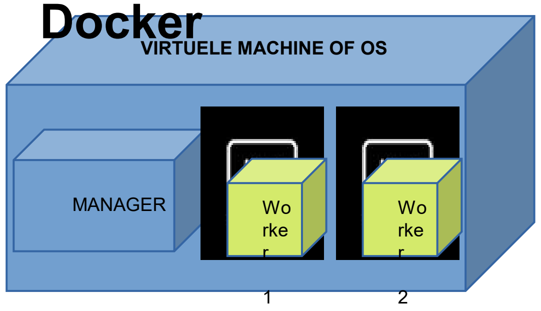
VIRTUELE MACHINES



Host-Only Adapter: Bv VM1: 192.168.56.1

VM2: 192.168.56.101

Docker in Docker



bridge: Bv Manager 172.17.0.1

Worker: 172.17.0.2

Services

- New concept since Docker 1.12
- Uses swarms + “long-running” containers.
- “SCALE-OUT” principle
- Command:
  - On manager:
    - docker service create --name SERVICENAME IMAGE COMMAND
- Service automatically gets distributed over the swarm (nodes/manager).
  - On manager:
    - docker service update --replicas 7 SERVICENAME
    - Containers get distributed over the cluster
- “SCALE-DOWN”
  - docker service update --replicas 4 SERVICENAME

Stack

- The services defined in docker-compose.yml = Stack
- Needs a local running docker registry:
  - $ docker service create --name registry --publish published=5000,target=5000 registry:2
- Create a directory which contains (from apache2 example):
  - Dockerfile
  - Docker-compose.yml
- In the directory:
  - Test: docker-compose up –d
  - List: docker-compose ps
  - Stop: docker-compose down
  - Push stack to registry: docker-compose push
- Deploy stack to swarm:
  - docker stack deploy --with-registry-auth --compose-file docker-compose.yml stackdemo

Docker Orchestration

Karel de Grote Hogeschool

Orchestration

- Linking containers
- Orchestrating containers
- Docker-compose

Linking Containers

LEGACY INFORMATION

- Co-operating containers
  - Source/recipient relationship
  - Security = “tunnel”
- Docker solution
  - Docker run
    - --link <container>:<alias>
      - Container = name of source container
      - Alias = exposed name that can be linked to by recipient

Linking Containers

LEGACY INFORMATION

- Linked container receives environment variables
  - NAME
    - Shows hierarchy
    - <ALIAS>_NAME
    - SRC_NAME=/rec/src.
  - ENV
    - Source environment variables: run –e; ENV in Dockerfile
    - <ALIAS>_ENV_<VAR_NAME>
    - SRC_ENV_SAMPLE.

Linking Containers

LEGACY INFORMATION

- Linked container receives environment variables
  - NAME
  - ENV
  - PORT
    - Source connectivity details: run –p; EXPOSE in Dockerfile
    - <ALIAS>_Port
      - URL of lowest port number of source container
    - <ALIAS>_PORT_<port>_<protocol>
      - <ALIAS>_PORT_<port>_<protocol>_ADDR: This form carries the IP address part of the URL (For example: SRC_PORT_8080_TCP_ADDR=172.17.0.2)
      - <ALIAS>_PORT_<port>_<protocol>_PORT: This form carries the port part of the URL (For example: SRC_PORT_8080_TCP_PORT=8080)
      - ALIAS>_PORT_<port>_<protocol>_PROTO: This form carries the protocol part of the URL (For example: SRC_PORT_8080_TCP_PROTO=tcp)

Linking Containers

LEGACY INFORMATION

- Docker updates host file
  - /etc/hosts
  - source IP address & alias

Orchestration

- Services
  - Process-driven
  - Composed containers
  - Specific composing sequence
- DevOps
  - Developers
  - System Administrators
  - Operations
- Micro-service Architecture
  - Decomposition of service in discrete components
  - Modularity
  - Loose & light coupling

Orchestration

- Docker’s own tool/framework:
  - Docker-compose
    - “fig” (bought by Docker)
    - Purpose: define your application's components (containers, configuration, links, volumes, …) in a single file and spin everything up with a single command
    - “Swarm” to scale & distribute over more than 1 server = “Orchestration”
- Other orchestration Tools
  - Helios
  - Flocker
  - **Kubernetes**

Docker Compose

- Install
  - Github:
    - https://github.com/docker/compose
  - Phython package
    - Uses “pip” installer
    - https://github.com/docker/compose

WAARSCHUWING:
April 26, 2022 marks the GA of Docker Compose V2. Starting today, Compose V2 is the standard across all documentation, and Compose V2 will become the developer default on Docker Desktop. However, you can continue aliasing `docker-compose` to `docker compose` and opt-out of V2 via the Docker Desktop UI — or by entering the `docker-compose disable-v2` command.

Extra: install via apt-get van docker-compose-plugin

Note:
Best to use the latest version of Docker Compose (search on GITHUB for latest stable 1.X.X number)

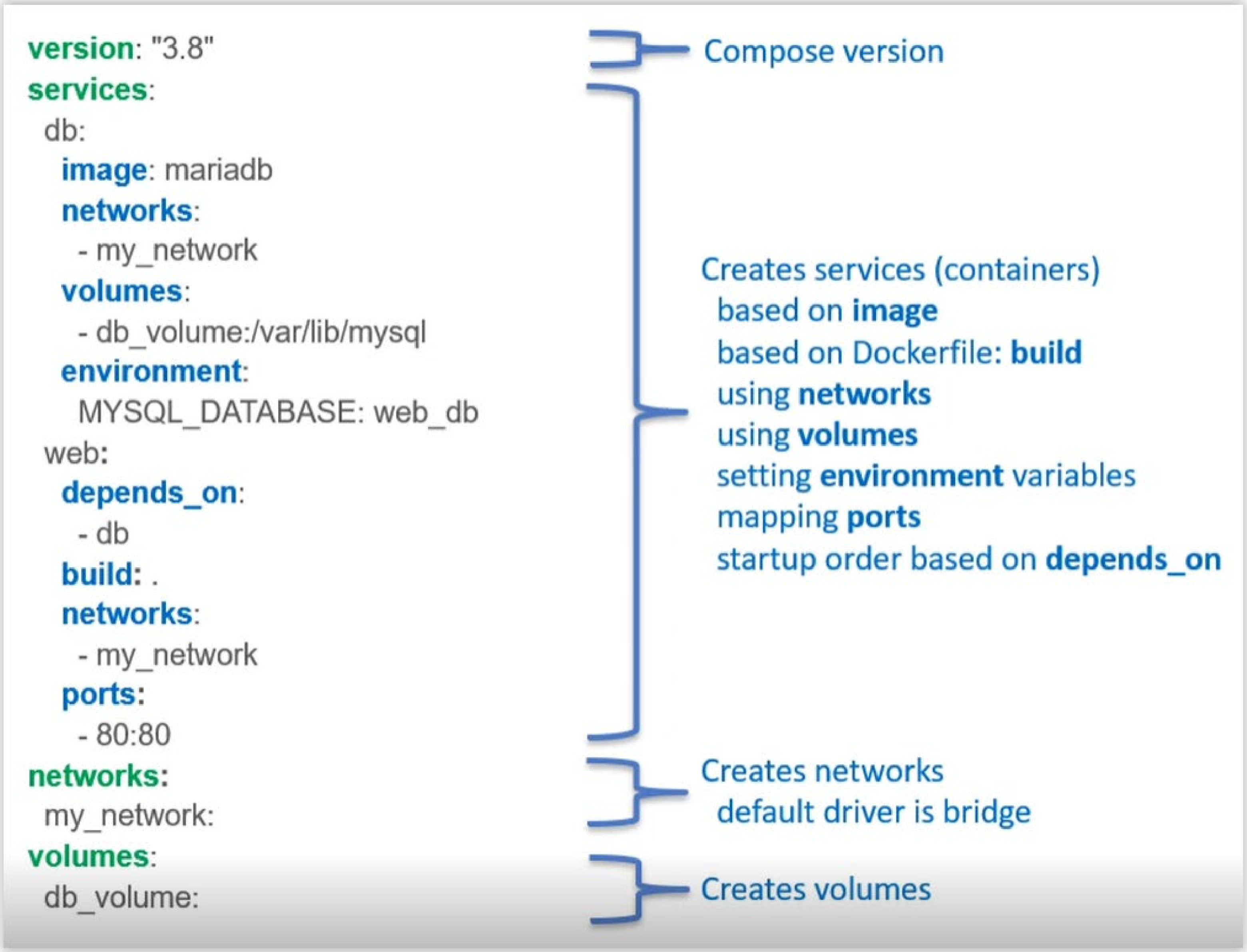
```
sudo curl -L "https://github.com/docker/compose/releases/download/1.X.X/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

In the labs: replace the version number with the latest version you find/found.

Docker Compose: docker-compose.yml

- Docker-compose.yml or .yaml file:
  - Hearth of the orchestration
  - YAML = YAML ain’t markup language (https://yaml.org/)
  - Human-friendly data serialization format
    - <service>:
      - <key>: <value>
      - <key>:
        - - <value>
        - - <value>
  - … different versions with different syntax depending on docker engine release.
  - Start = docker-compose up;
  - Stop = docker-compose down
  - https://docs.docker.com/compose/compose-file/

Docker Compose: docker-compose.yml



```
version: "3.8"
services:
  db:
    image: mariadb
    networks:
      - my_network
    volumes:
      - db_volume:/var/lib/mysql
    environment:
      MYSQL_DATABASE: web_db
  web:
    depends_on:
      - db
    build: .
    networks:
      - my_network
    ports:
      - 80:80
networks:
  my_network:
volumes:
  db_volume:
```

Compose version

Creates services (containers)
- based on **image**
- based on Dockerfile**: build**
- using **networks**
- using **volumes**
- setting **environment** variables
- mapping **ports**
- startup order based on **depends_on**

Creates networks
- default driver is bridge

Creates volumes

Docker Compose: docker-compose.yml

- Short explanation of content:
  - service: name of the service = image or build key
    - image: This is the tag or image ID
    - build: This is the path to a directory containing a Dockerfile command: This key overrides the default command
  - …

Note: see reference docker-compose.

Docker Compose: versions

Compose file versions

Version 1 is legacy and shouldn't be used.

(If you see a Compose file without version and services, it's a legacy v1 file.)

Version 2 added support for networks and volumes.

Version 3 added support for deployment options (scaling, rolling updates, etc).

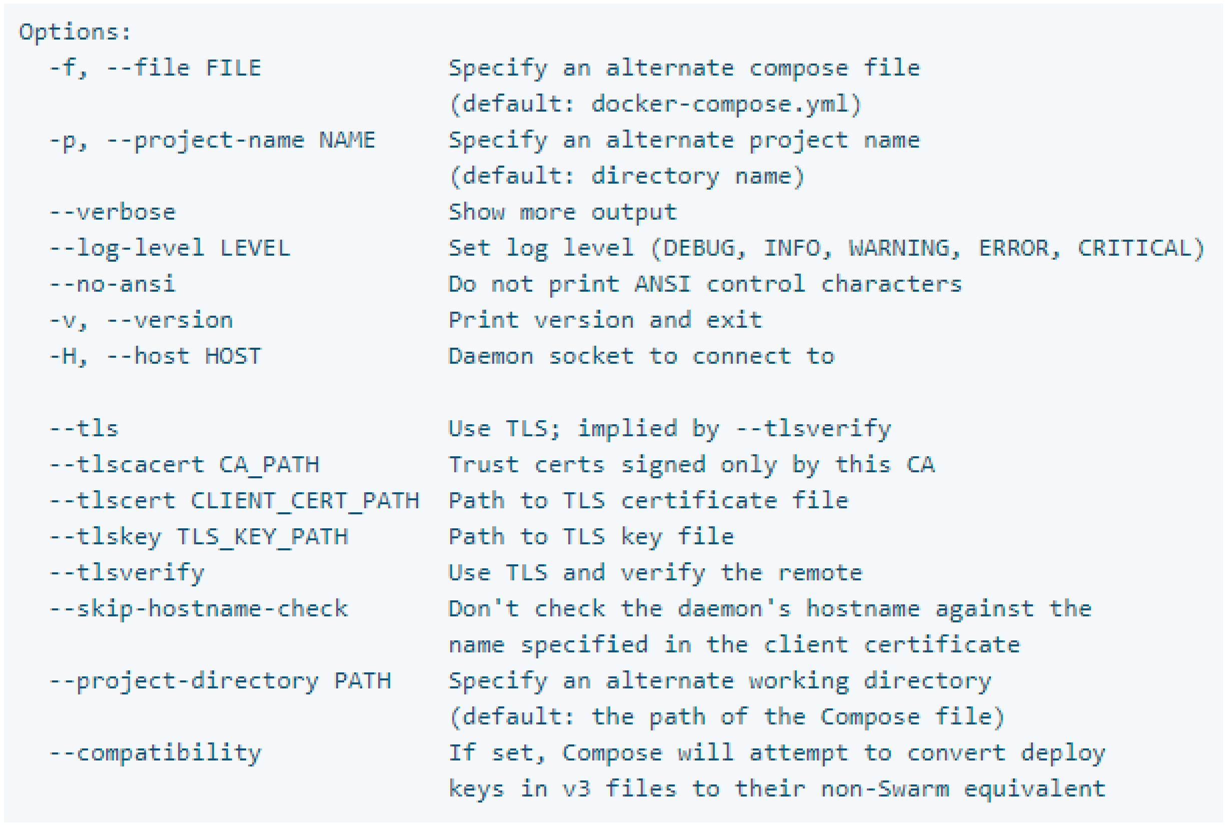
The Docker documentation has excellent information about the Compose file format if you need to know more about versions:

https://docs.docker.com/compose/compose-file/

https://docs.docker.com/compose/compose-file/compose-versioning/

Docker Compose: running

- docker-compose [<options>] <command> [<args>...]
- Options:

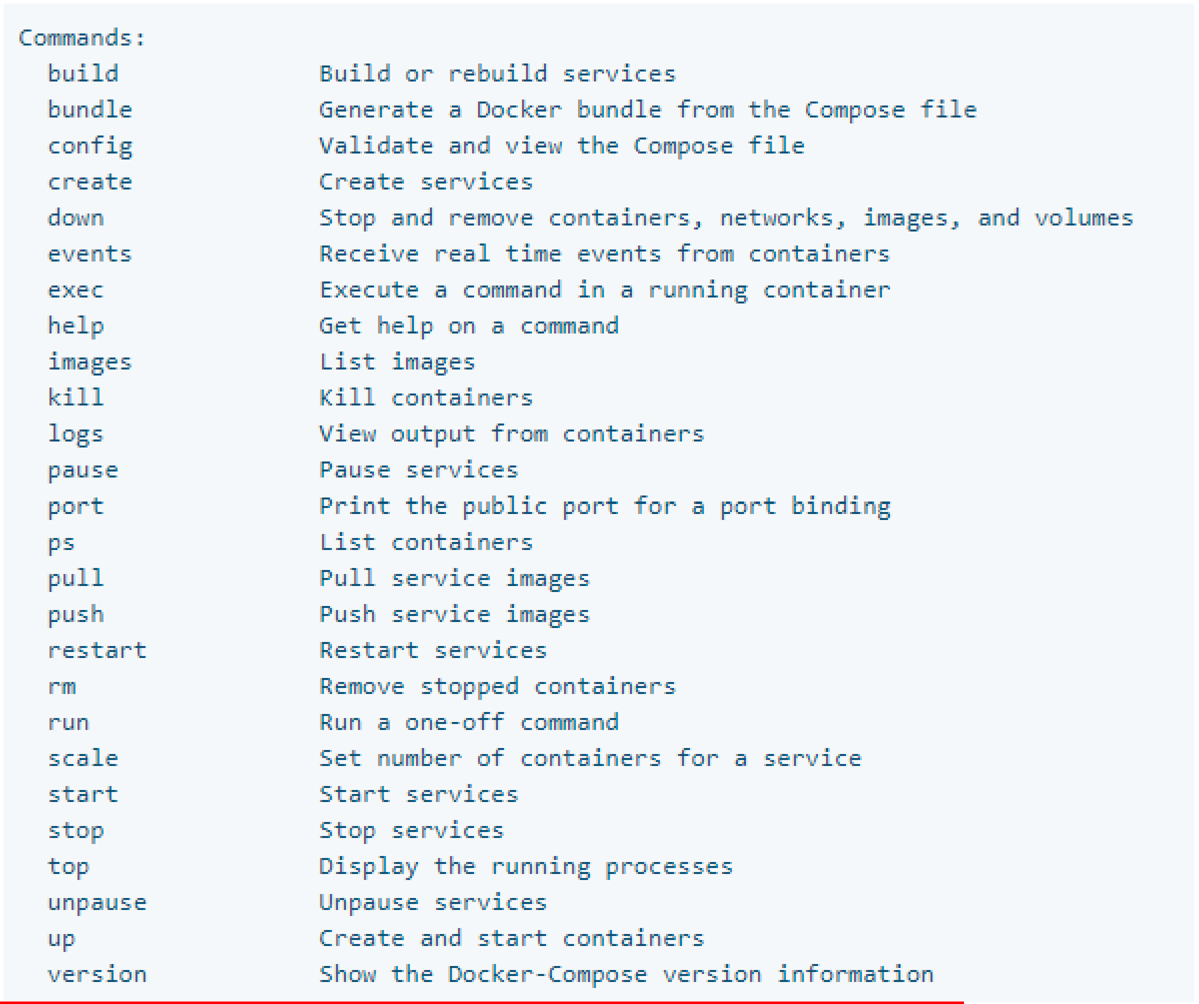


```
Options:
  -f, --file FILE             Specify an alternate compose file
                              (default: docker-compose.yml)
  -p, --project-name NAME     Specify an alternate project name
                              (default: directory name)
  --verbose                   Show more output
  --log-level LEVEL           Set log level (DEBUG, INFO, WARNING, ERROR, CRITICAL)
  --no-ansi                   Do not print ANSI control characters
  -v, --version               Print version and exit
  -H, --host HOST             Daemon socket to connect to

  --tls                       Use TLS; implied by --tlsverify
  --tlscacert CA_PATH         Trust certs signed only by this CA
  --tlscert CLIENT_CERT_PATH  Path to TLS certificate file
  --tlskey TLS_KEY_PATH       Path to TLS key file
  --tlsverify                 Use TLS and verify the remote
  --skip-hostname-check       Don't check the daemon's hostname against the
                              name specified in the client certificate
  --project-directory PATH    Specify an alternate working directory
                              (default: the path of the Compose file)
  --compatibility             If set, Compose will attempt to convert deploy
                              keys in v3 files to their non-Swarm equivalent
```

Docker Compose: running

- docker-compose [<options>] <command> [<args>...]
- Commands:



```
Commands:
  build              Build or rebuild services
  bundle             Generate a Docker bundle from the Compose file
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove containers, networks, images, and volumes
  events             Receive real time events from containers
  exec               Execute a command in a running container
  help               Get help on a command
  images             List images
  kill               Kill containers
  logs               View output from containers
  pause              Pause services
  port               Print the public port for a port binding
  ps                 List containers
  pull               Pull service images
  push               Push service images
  restart            Restart services
  rm                 Remove stopped containers
  run                Run a one-off command
  scale              Set number of containers for a service
  start              Start services
  stop               Stop services
  top                Display the running processes
  unpause            Unpause services
  up                 Create and start containers
  version            Show the Docker-Compose version information
```

Warning: “docker compose rm” = removes containers, not volumes! (use –v to remove anonymous volumes)

Docker Compose: Example

- Dockerfile

```
FROM ubuntu
RUN apt-get update && apt-get install -y apache2
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D",
"FOREGROUND"]
```

Docker Compose: Example

Opletten: INSPRINGEN !!!
En spaties na : en -

- Compose-file = docker-compose.yml

```yaml
version: '3'
services:
  web:
    build: .
    ports:
      - "8080:80"
    volumes:
      - .:/var/www/html
```

Docker Compose: Example (Extra)

```
version: "3"
services:
  app:
    # replace username/repo:tag with your name and image details
    image: hifzak/testing:part2
    deploy:
      replicas: 10
      resources:
        limits:
          cpus: "0.5"
          memory: 4M
      restart_policy:
        condition: on-failure
```

Let op:
Deploy werkt alleen in “swarm” met docker stack deploy.

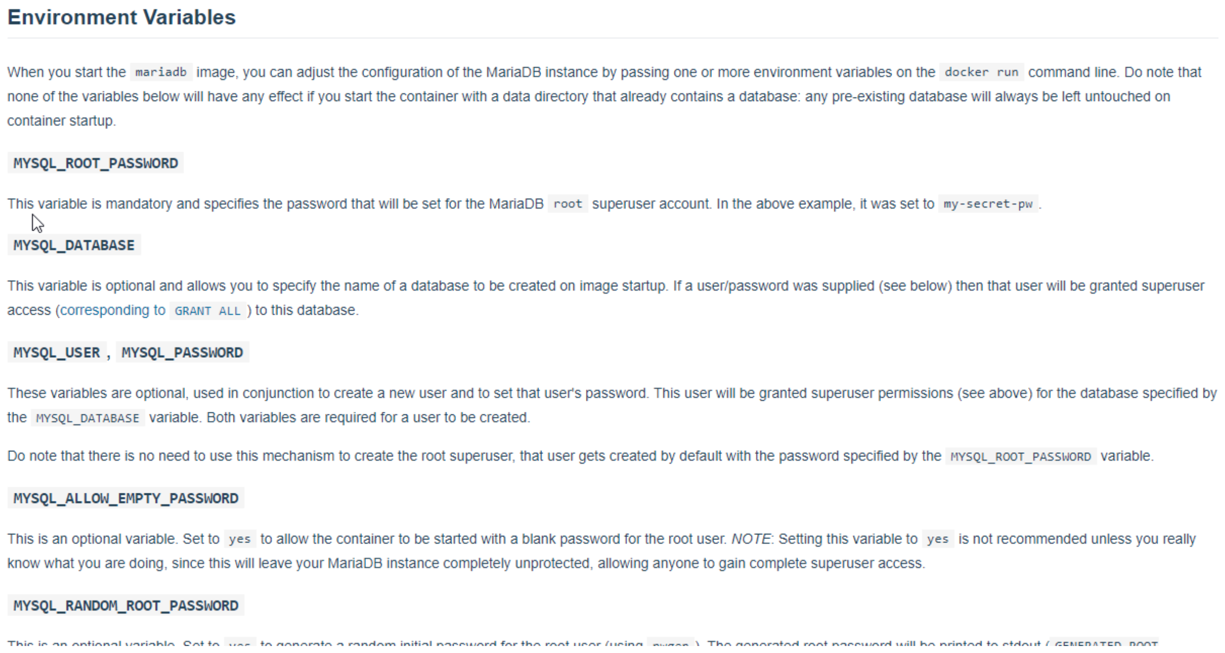
Let op: replicas = deprecated, use scale…

Docker Compose: Example

- Give commands:
  - Start: docker-compose up
  - Stop: docker-compose down

Docker Compose: Example

- Extra:
  - Some images use environment variables
  - Information in image-description on docker hub
    - https://hub.docker.com/search?q=&type=image
    - Example: https://hub.docker.com/_/mariadb



Environment Variables

When you start the `mariadb` image, you can adjust the configuration of the MariaDB instance by passing one or more environment variables on the `docker run` command line. Do note that none of the variables below will have any effect if you start the container with a data directory that already contains a database: any pre-existing database will always be left untouched on container startup.

`MYSQL_ROOT_PASSWORD`

This variable is mandatory and specifies the password that will be set for the MariaDB `root` superuser account. In the above example, it was set to `my-secret-pw` .

`MYSQL_DATABASE`

This variable is optional and allows you to specify the name of a database to be created on image startup. If a user/password was supplied (see below) then that user will be granted superuser access (corresponding to `GRANT ALL`) to this database.

`MYSQL_USER` , `MYSQL_PASSWORD`

These variables are optional, used in conjunction to create a new user and to set that user's password. This user will be granted superuser permissions (see above) for the database specified by the `MYSQL_DATABASE` variable. Both variables are required for a user to be created.

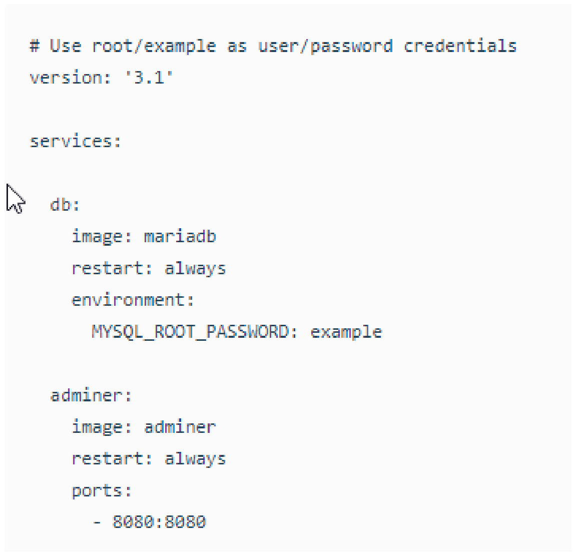
Do note that there is no need to use this mechanism to create the root superuser, that user gets created by default with the password specified by the `MYSQL_ROOT_PASSWORD` variable.

`MYSQL_ALLOW_EMPTY_PASSWORD`

This is an optional variable. Set to `yes` to allow the container to be started with a blank password for the root user. *NOTE*: Setting this variable to `yes` is not recommended unless you really know what you are doing, since this will leave your MariaDB instance completely unprotected, allowing anyone to gain complete superuser access.

`MYSQL_RANDOM_ROOT_PASSWORD`

This is an optional variable. Set to `yes` to generate a random initial password for the root user (using `pwgen`). The generated root password will be printed to stdout (`GENERATED ROOT`



```
# Use root/example as user/password credentials
version: '3.1'

services:

  db:
    image: mariadb
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: example

  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
```