Workshop

Docker

Basics

Rainer Stropek

software architects gmbh

Web http://www.timecockpit.com
Mail rainer@timecockpit.com
Twitter @rstropek

time cockpit
Saves the day.

Your Host

Rainer Stropek

Developer, Entrepreneur
Azure MVP, MS Regional Director
IT-Visions

Contact

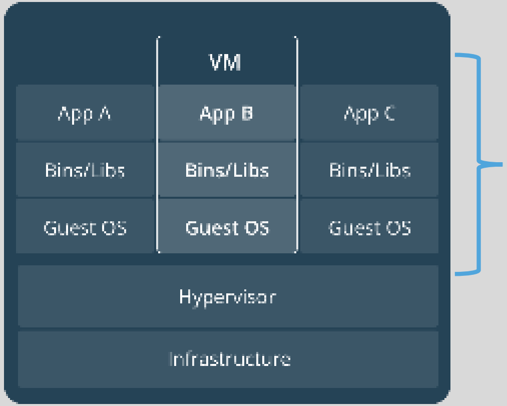
software architects gmbh
rainer@timecockpit.com
Twitter: @rstropek

What is Docker?

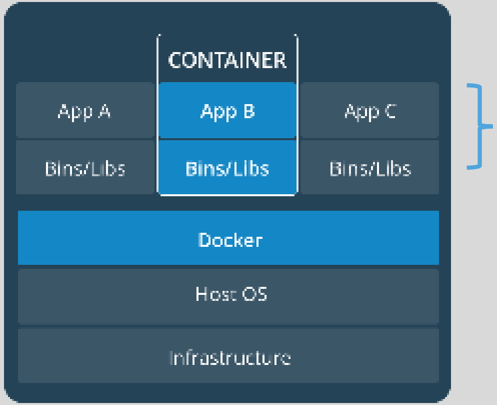
Introduction

What is Docker?

Virtual machines vs. Docker



Virtual Machines



Docker Container

- Each VM runs its own guest operating system
- Container reuse the host operating system
  - Container run in user space
- Not a total replacement of classical hypervisors or config management tools

Image Source:
https://docs.docker.com/get-started/#containers-and-virtual-machines

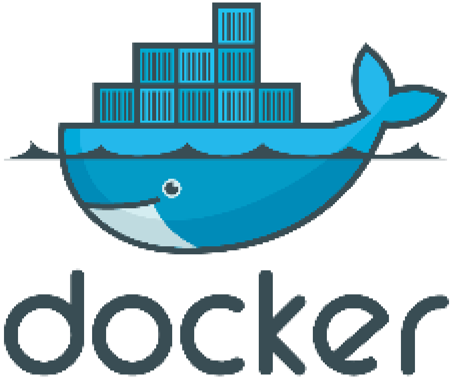
What’s Docker?

Container virtualization

Container run in user space and use kernel of host
Has been existing in Linux for quite a while
Docker builds on Linux Containers (LXC) and makes it easy to use and consume

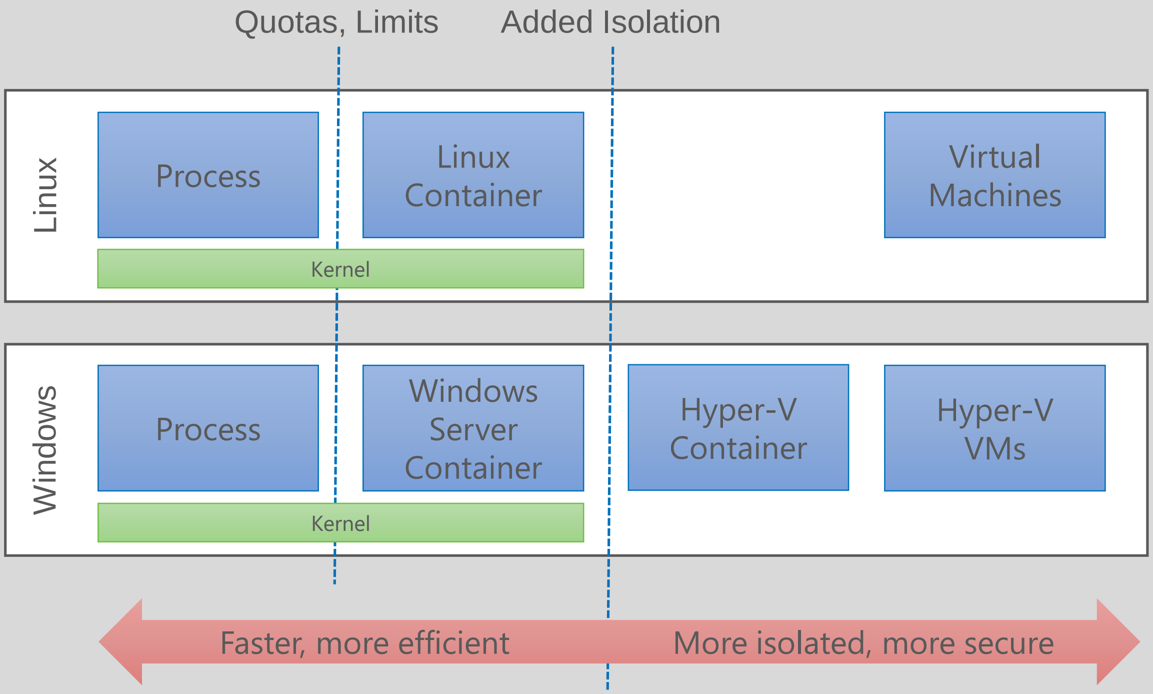
Advantages?

Fast (boot time), small, and agile (e.g. Docker in Docker)
Portable
Immutable



Disadvantages?

Linux on Linux and Windows on Windows, no mix
Security (less isolated)



Strengths and Limits

- Windows Server vs. Hyper-V Containers
  - Managed almost identically (Docker and PowerShell)
  - Difference: Isolation level
  - More details in [Microsoft Docs]

Source: Mark Fussel (Microsoft), Azure Service Fabric - Build always-on, hyper-scalable, microservice-based cloud applications

Docker's Technical Components

Linux container format (`libcontainer`)

Isolation layers

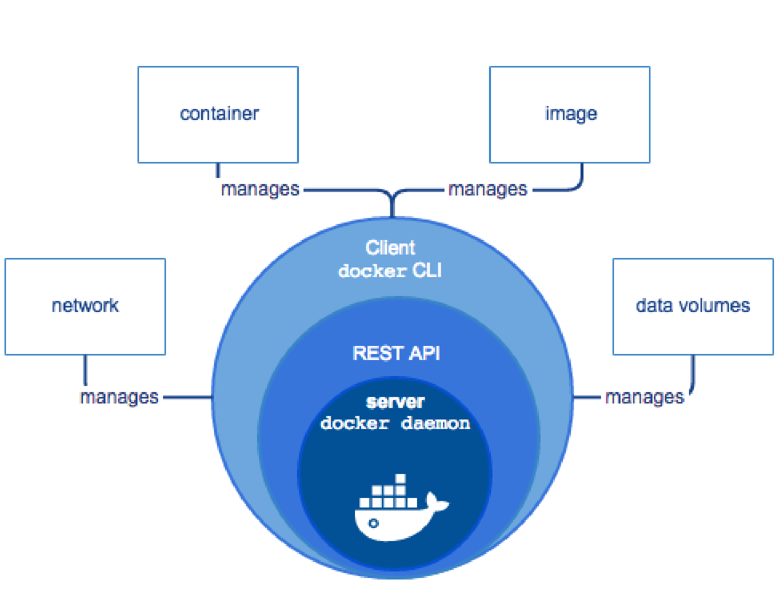
Filesystem – each container has its own filesystem (layered, copy-on-write)
Processes – each container has its own process environment
Network – separate virtual network interfaces
Resources – individually allocated CPUs, memory

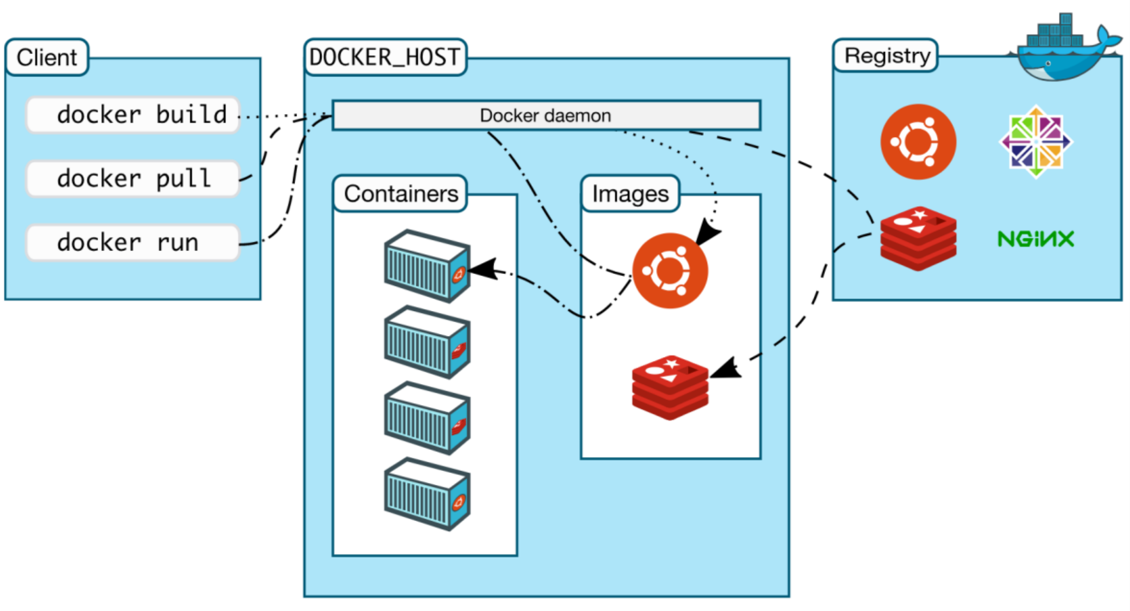
Logging

STDOUT, STDERR, STDIN are logged for analysis purposes

Interactive shell

Pseudo-tty attached to STDIN





What’s Docker?

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

What's Docker?

Docker Daemon

Server

Command line tool, REST services

Docker client can manage remote Docker daemon

Container packaging format

Dockerfiles for image creation from source code

Version management for images

Images can be based on images

Docker Hub: Platform to exchange images and Dockerfiles

Publishing on Docker Hub is not in scope of this talk

What to Use Docker For?

Make dev/test/prod-cycle more productive

Developers build containers, not apps
Containerize build-, test- and CI-tools

Segregation of duties

Dev cares for app running in container, ops cares for managing containers

Microservices

Isolate services
Consistency across stages (dev/test/prod)

Test even complex environments locally

Containers are lightweight → run on rather small dev boxes

Docker Tools

Introduction

Docker and Microsoft

Docker Desktop

Docker environment for Windows and Mac

Container virtualization in Windows

Windows Containers Quick Start

Use Azure to play with Docker

Existing VM image (*Docker on Ubuntu server*) in Azure marketplace

Use Docker container to run Azure tools (e.g. https://hub.docker.com/r/microsoft/azure-cli/)

Azure Container Instance

Azure Container Registry

Azure Kubernetes Service

Visual Studio DevOps Tooling

Docker Extension for Visual Studio Code

https://marketplace.visualstudio.com/items?itemName=PeterJausovec.vscode-docker

Container Tools in Visual Studio

https://docs.microsoft.com/en-us/visualstudio/containers/overview?view=vs-2019

Docker Cluster Solutions

Docker Swarm Mode

https://docs.docker.com/engine/swarm/

Native clustering for Docker, turns a pool of Docker hosts into a single, virtual Docker host

Kubernetes

https://kubernetes.io/

Azure Kubernetes Service (AKS)

https://azure.microsoft.com/en-us/services/kubernetes-service/

Managed Kubernetes

Access Docker Remotely

Default: Docker runs on non-networked Unix socket

`unix:///var/run/docker.sock`

TCP socket can be enabled (see Docker docs) → Docker Remote Web API

Docker available on the network → enable TLS

Docker docs

Remote Docker

```
// Connect to Docker client in Azure
// (see also https://github.com/rstropek/DockerVS2015Intro)

// Set environment variable (secure by default)
export DOCKER_HOST=tcp://dockertraining
  .northeurope.cloudapp.azure.com:2376 DOCKER_TLS_VERIFY=1
docker info
docker ps
```

Container

Working with containers

Containers

Launched from images

Layered, copy-on-write
Will be covered in details later

Contain one or more processes

Can be short-lived

Sometimes even to run jus a single command

Shared via registries

Docker Hub (private and public repositories)
Run your own private registry (`registry` image on Docker Hub)

Docker CLI

Documentation

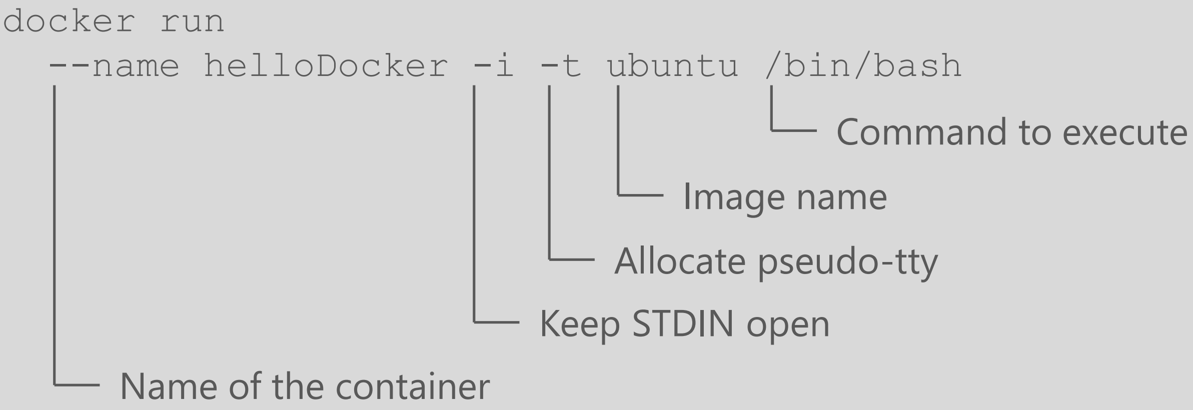
http://docs.docker.com/reference/commandline/cli

Important Commands for Containers

`docker run` – Run a command in a new container
`docker ps` – List containers
`docker start/stop` – Restarts/stops a container
`docker rm` – Removes container(s)
`docker attach` – Attach to running container
`docker top` – Display processes running in container
`docker exec` – Run a command in a container
`docker container prune` – Remove all stopped containers

Docker CLI

Starting Containers



```
docker run
  --name helloDocker -i -t ubuntu /bin/bash
```

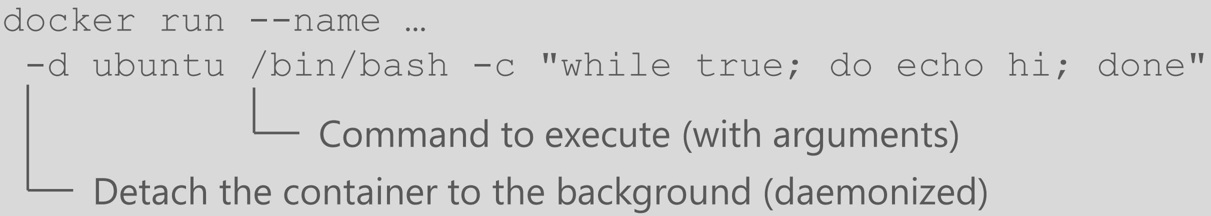
Command to execute

Image name

Allocate pseudo-tty

Keep STDIN open

Name of the container



```
docker run --name …
 -d ubuntu /bin/bash -c "while true; do echo hi; done"
```

Command to execute (with arguments)

Detach the container to the background (daemonized)

Interactive container

Daemonized container

Running in the background

`--rm` removes container when it exits

Demo

Interactive Container

```
# Check if docker is running
docker info

# Start interactive container
docker run -it ubuntu /bin/bash
  echo Hello > hello.txt
  exit

# List containers
docker ps
docker ps –a
docker ps --no-trunc -aq

# Restart container
docker start …

# Attach to container
docker attach …

# Remove container
docker rm …
# Remove all containers
docker rm `docker ps --no-trunc -aq`
docker container prune
```

Demo

Daemonized Container

```
# Start demonized container and get logs
docker run -d ubuntu /bin/bash \
  -c "while true; do echo hello world; sleep 1; done"

# Get the logs (-f for continuous monitoring)
docker logs …

# Check the processes in docker container
docker top …

# Open interactive shell in running container
docker exec -it … /bin/bash

# Inspect the details of a running container
docker inspect …


# WINDOWS
docker run –it mcr.microsoft.com/windows/servercore cmd

docker build –t myweb .
docker run
```

Docker Events

Docker reports real time events from the server

`docker events`

Usages

Admin and monitoring purposes

Triggering auto-configurations (e.g. load balancer configuration with Interlock and Nginx)

Networking

Docker Networking

Networks

By default, three networks

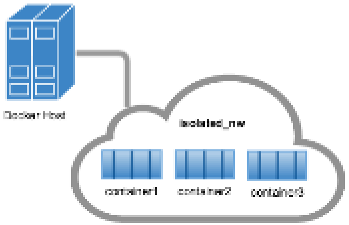
none, *host*, *bridge* (default)

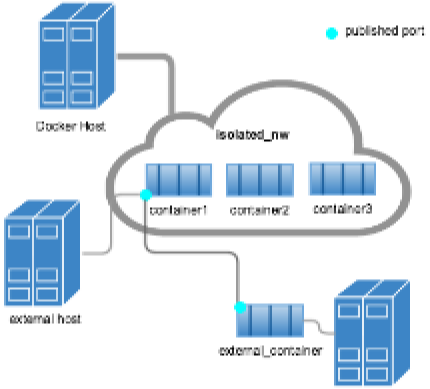
Additional networks can be created

Bridge network = single host

Overlay network (advanced topic, see Docker docs) can include multiple hosts

Network isolation





For details see https://docs.docker.com/engine/userguide/networking/dockernetworks/

Networks

```
# List all networks
docker network ls

# Inspect network details
docker network inspect bridge

# Disconnect a container from network
docker network disconnect bridge mycontainer
```

Container name

Network name

```
# Connect a container to a network
docker network connect mynetwork mycontainer

# Create own network
docker network create -d bridge mynetwork
```

Network name

Driver name

```
# Start container in a specific network
docker run -it --net=mynetwork ubuntu
```

For details about network security, see Docker docs

DNS

Docker daemon contains embedded DNS server

```
# Start nginx web server on a custom network
docker run -d --net mynetwork --name web nginx
```

Container name in DNS

```
# Start Ubuntu client in same network
docker run -it --net mynetwork --name client ubuntu

  # Ping web server
  ping web

  # Install curl and access web server
  apt-get install curl
  curl web

# Start Ubuntu container and link it using alias
docker run -it --net mynetwork --link=server3:nginx ubuntu
```

Container-specific link

For details see
https://docs.docker.com/engine/userguide/networking/configure-dns/

Binding container ports to host

```
docker run -d --net bridge -p 8080:80 nginx
```

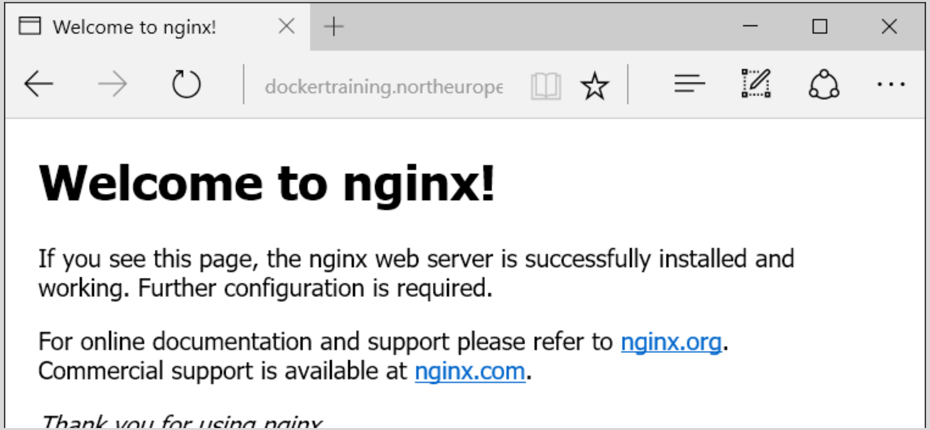
Host port

Container port

```
# Start nginx web server on host network
docker run -d --net host nginx
```

Assign container to *host* network

```
# Nginx is now available on the public internet:
```



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

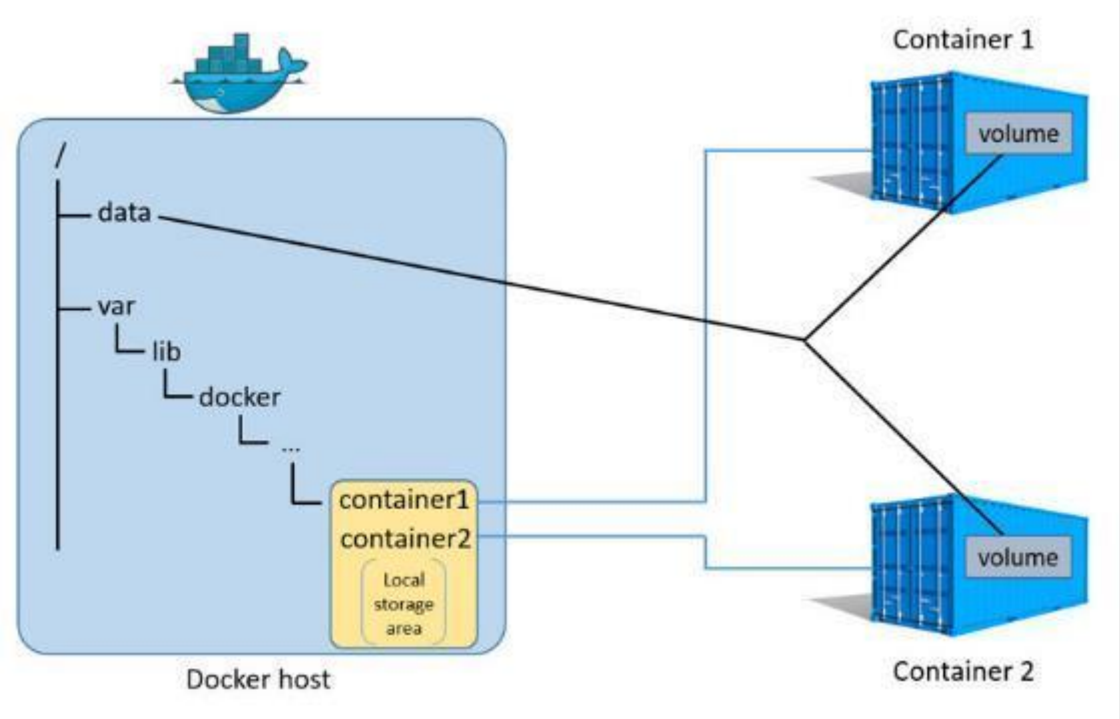
Thank you for using nginx.

Port mapping

EXPOSE in Dockerfiles

See Docker docs

Use *host* network



Data Volumes

Directory or file in the Docker host’s filesystem that is mounted directly into a container

Details see [Docker docs]

Source: https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/

Mount Host

```
# Run postgres in a new container
docker run --name mydb -e POSTGRES_PASSWORD=P@ssw0rd \
  -d postgres
docker inspect mydb

# Run client and execute some SQL
docker run -it --rm postgres /bin/bash
  psql -h 172.17.0.2 -p 5432 -U postgres

  # Execute some SQL (e.g. create and fill a table)
  CREATE TABLE Test (ID INT PRIMARY KEY);
  INSERT INTO Test VALUES (1);
  SELECT * FROM Test;
  \q

# Delete container --> data is gone
docker rm –f mydb
```

Mount Host

```
# Create data directory on host
mkdir dbdata

# Repeat the same example but this time with volume mapping
docker run --name mydb -e POSTGRES_PASSWORD=P@ssw0rd! \
  --mount
'type=bind,src=/home/rainer/dbdata,dst=/var/lib/postgresql/data
' -d postgres
```

Bind Mount

Data Volume Container

```
# Create volume
docker volume create dbstore
docker volume inspect dbstore

# Create postgres container and mount data volume container
docker run --name mydb -e POSTGRES_PASSWORD=P@ssw0rd \
  -e PGDATA=/dbdata \
  --mount 'type=volume,src=dbstore,dst=/dbdata' \
  -d postgres

# Run client and execute some SQL (see previous example)
# Remove postgres container, recreate it --> data still there

# Start container to backup data
mkdir backup
docker run --rm \
  --mount 'type=volume,src=dbstore,dst=/dbdata' \
  --mount 'type=bind,src=/home/rainer/backup,dst=/backup' \
  ubuntu tar cvf /backup/backup.tar /dbdata
ls –la backup/
```

Docker Volumes on Azure Files

Azure Files-Driver for Docker Volumes available

https://github.com/Azure/azurefile-dockervolumedriver

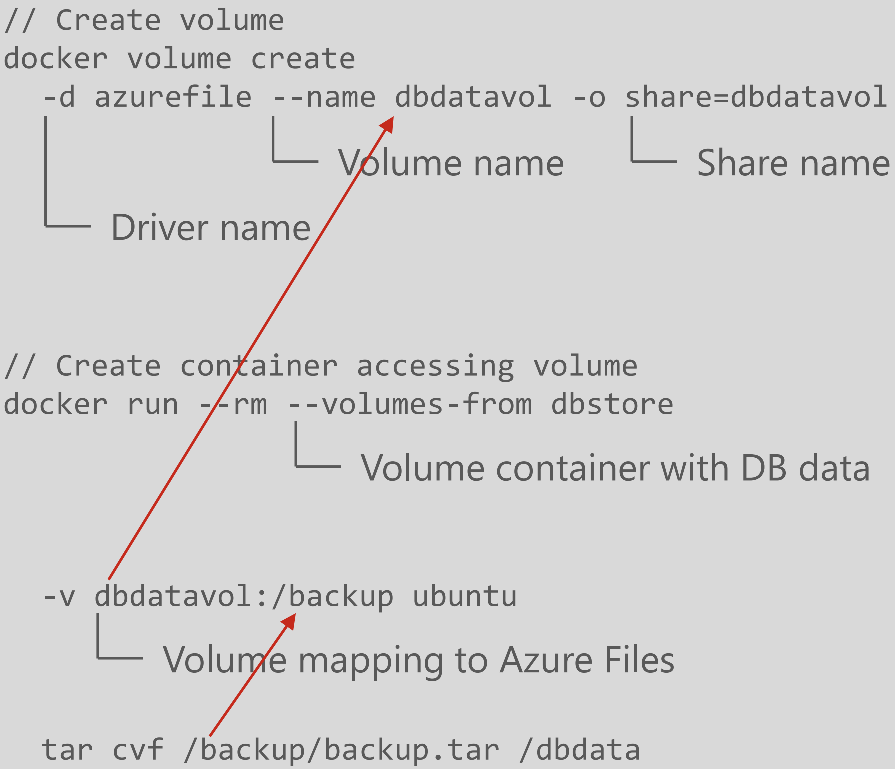
Store persistent data outside of Docker Containers

Docker containers/hosts can be moved, recreated, etc. without loosing data

High availability, replication for data

Multiple containers/hosts can access the same volume

Azure Files Volume



```
// Create volume
docker volume create
  -d azurefile --name dbdatavol -o share=dbdatavol
```

Volume name

Share name

Driver name

```
// Create container accessing volume
docker run --rm --volumes-from dbstore
```

Volume container with DB data

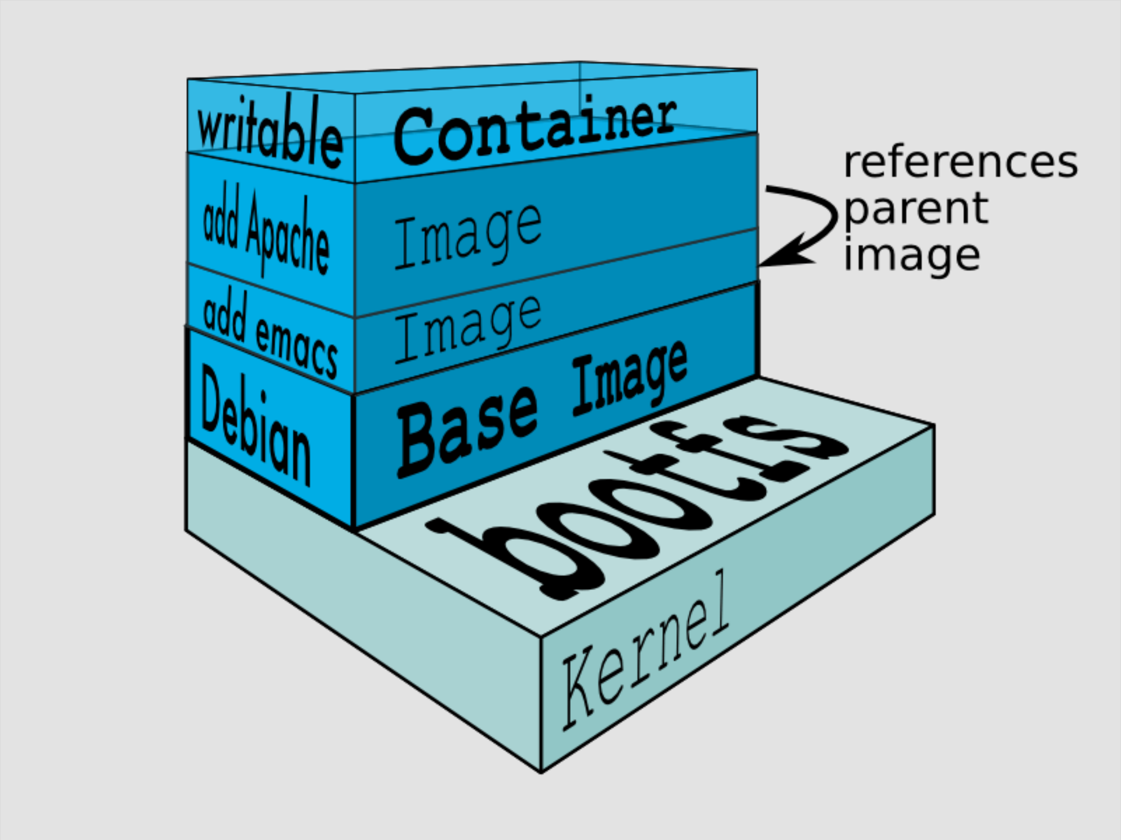
```
  -v dbdatavol:/backup ubuntu
```

Volume mapping to Azure Files

```
  tar cvf /backup/backup.tar /dbdata
```

Images

Working with images



File System Layers

Rootfs stays read-only

Union-mount file system over the read-only file system

- Multiple file systems stacked on top of each other

Only top-most file system is writable

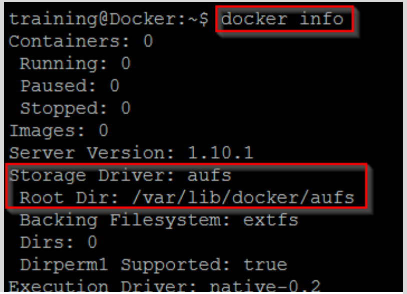
- Copy-on-write

Images

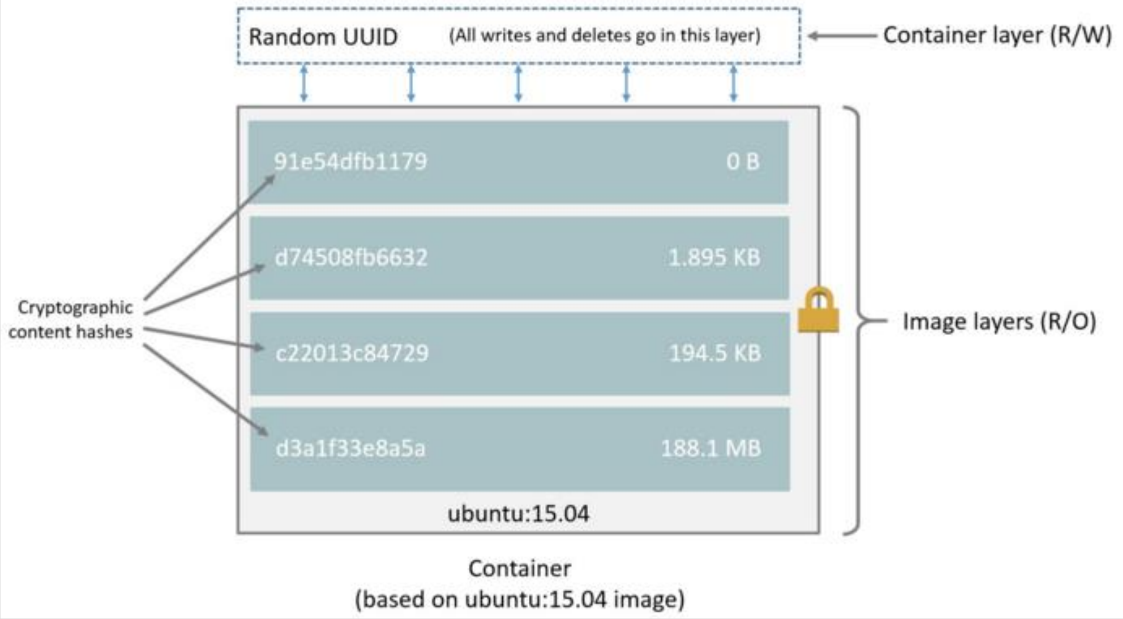
```
# Pull image from docker hub
docker pull ubuntu

# Look for image directories on disk
ls /var/lib/docker/aufs/layers
```

└── Docker data directory



```
training@Docker:~$ docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.10.1
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 0
 Dirperm1 Supported: true
Execution Driver: native-0.2
```



More about storage drivers
see [Docker docs](https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/)

Source:
https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/

Docker CLI

Important Commands for Images

`docker images` – List all images

`docker search` – Search for image on Docker Hub

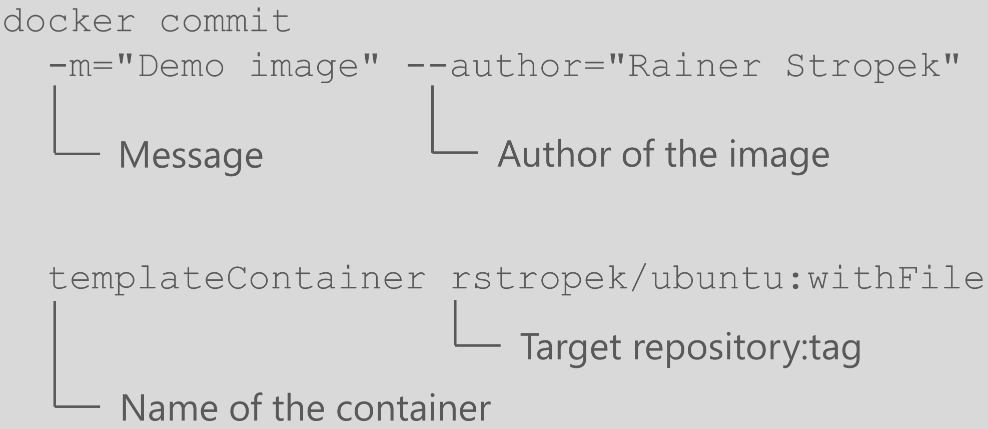
`docker pull` – Pulls an image from the registry (Docker Hub)

`docker commit` – Create image from container

`docker inspect` – Get low-level information on container or image

Docker CLI

Building Images from Containers



```
docker commit
  -m="Demo image" --author="Rainer Stropek"
  templateContainer rstropek/ubuntu:withFile
```

Message

Author of the image

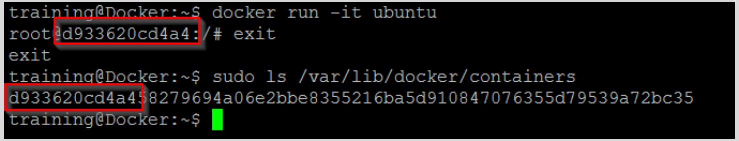
Target repository:tag

Name of the container

Demo

Create Image

```
# Start interactive container
docker run -it ubuntu /bin/bash
  echo "Hello Docker" > helloWorld.txt
  exit
```



```
training@Docker:~$ docker run -it ubuntu
root@d933620cd4a4:/# exit
exit
training@Docker:~$ sudo ls /var/lib/docker/containers
d933620cd4a458279694a06e2bbe8355216ba5d910847076355d79539a72bc35
training@Docker:~$
```

```
# Build image from container
docker commit … rainer:withFile

# Remove container
docker rm -f …

# Create new container from new image
docker run -it rainer:withFile /bin/bash
# View history of image
Docker history rainer:withFile

# Remove image
docker rmi rainer:withfile
```

Checklist for Dockerfiles

Select approriate base image

Prefer existing (official) base images

Use multistage builds

Use image with SDK just for building

Use specialized runtime images for running containers

Consolidate *RUN* statements

```
RUN apt-get -y update && apt-get install -y python
```

Use tags to describe purpose of images

E.g. dev, prod, version, base (e.g. *alpine*)

Checklist for Dockerfiles

Never store data in container‘s writable layer

Use volumes instead

Use automated CI/CD

E.g. Azure DevOps, GitHub integration

Dockerfiles

Creating images from source

Dockerfiles

```
FROM nginx:alpine
LABEL maintainer=rainer@timecockpit.com
RUN apt-get -y update
```

Execute command in new layer on top of the image and commit the result

```
COPY app/ /usr/share/nginx/html/
```

Copy files to the filesystem of the container

```
docker build -t staticweb .
```

Dockerfile location

Tag for the image

Documentation

- https://docs.docker.com/reference/builder/
- https://registry.hub.docker.com/_/nginx/

Dockerfiles

Multistep Build

```
FROM node as build
ENV approot /app
COPY ./app ${approot}
WORKDIR ${approot}
RUN rm -rf ./dist && rm -rf ./node_modules && npm install &&
npm run build

FROM nginx:alpine
COPY --from=build /app/dist/ /usr/share/nginx/html/
```

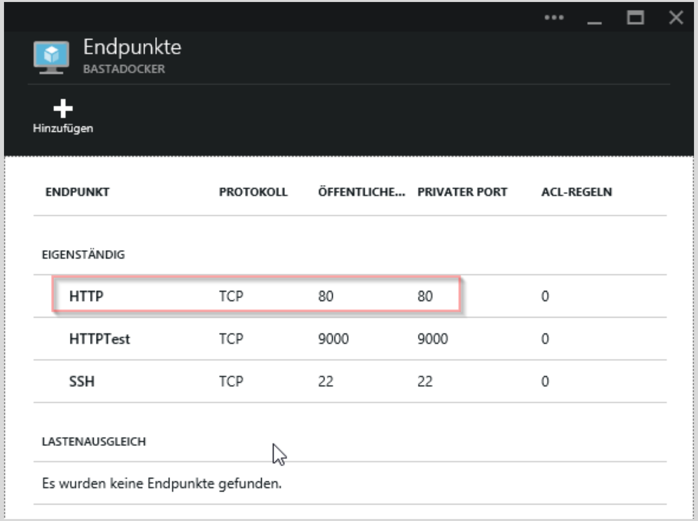
Docker CLI

Exposing ports

```
docker run --name staticwebcontainer \
  -d -p 80:80 staticweb
```

Expose port 80

Run daemonized



ENDPUNKT	PROTOKOLL	ÖFFENTLICHE...	PRIVATER PORT	ACL-REGELN
EIGENSTÄNDIG				
HTTP	TCP	80	80	0
HTTPTest	TCP	9000	9000	0
SSH	TCP	22	22	0
LASTENAUSGLEICH				
Es wurden keine Endpunkte gefunden.				

Demo

Dockerfile

```
# Get sample code from GitHub
git clone https://github.com/rstropek/DockerVS2015Intro.git

# Build website
cd dockerDemos/01-staticWeb/app
npm install
npm run build
cd ..

# Build image from Dockerfile
docker build -t staticweb .
docker run -d -p 80:80 staticweb

# Change website content and rebuild container

# Run a second container, run a third container (linked)
docker run -i -t --link <cont1>:sweb1 --link <cont2>:sweb2
ubuntu /bin/bash
  apt-get install curl
  curl http://sweb1
```

Sample files see
https://github.com/rstropek/DockerVS2015Intro/tree/master/dockerDemos/01-staticWeb

Demo

Automated build

```
# Run webpack inside a docker container
docker run -t --rm -v C:\...\dockerDemos\01-staticWeb\app:/app
-w /app node npm run build


# Run webpack inside a docker container (watch mode)
docker run -t --rm -v C:\...\dockerDemos\01-staticWeb\app:/app
-w /app node bash -c "npm run watch -- --watch-poll 1000"


# Run nginx webserver inside container
docker run --rm -t -p 8081:80 -v C:\...\dockerDemos\01-
staticWeb\app\dist:/usr/share/nginx/html/ nginx:alpine
```

Demo

Run Grunt (build) in Container

```
# Run grunt inside a docker container
docker run --rm
```

Remove the container when it exists

```
  -v ~/dockerDemos/01-staticWeb/app/dist:/app
```

Mount host volume (`host:container`)

```
  node
```

Use existing image

```
  npm run build
```

Run `webpack`

Docker Compose

Tool for running multi-container applications

Demo

```
printer:
  build: .
```

Build local Dockerfile

```
  links:
   - dependent-service
```

Link to other containers (e.g. Redis, MongoDB)

```
dependent-service:
  image: dependent-service
```

Run service container depends on based on an existing image

For more info visit https://docs.docker.com/compose/

Demo

Automated build

```
# Build dependent service
# directory: ~/DockerVS2015Intro/dockerDemos/02-compose/dependentService
npm install
docker build –t dependent-service .

# Run container using dependent service
# directory: ~/DockerVS2015Intro/dockerDemos/02-compose
npm install
docker-compose run printer
```

Sample files see
https://github.com/rstropek/DockerVS2015Intro/tree/master/dockerDemos/02-compose

ASP.NET in Docker

Running ASP.NET in Docker (*Docker-and-dot-net.pptx*)

Application Scenarios

Running continuous integration in containers

Rebuild complex runtime environment on my laptop

Identical environment for dev, test, and prod

Cost reduction in the cloud

High density hosting (e.g. multiple versions)

Split software into multiple, independent services

Micro-services, see Manfred's session tomorrow

Workshop

Q&A

Thank you for attending!

Rainer Stropek

software architects gmbh

Web	http://www.timecockpit.com
Mail	rainer@timecockpit.com
Twitter	@rstropek

time cockpit

Saves the day.