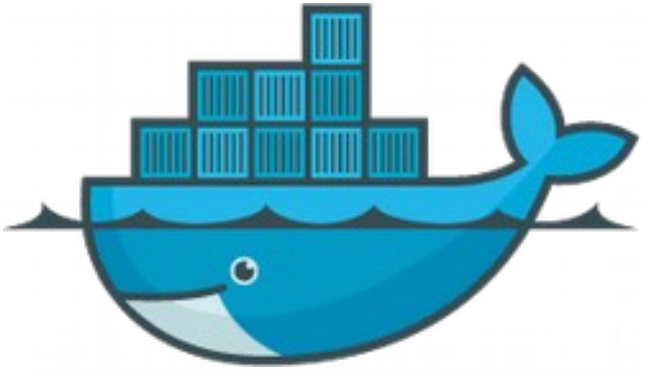
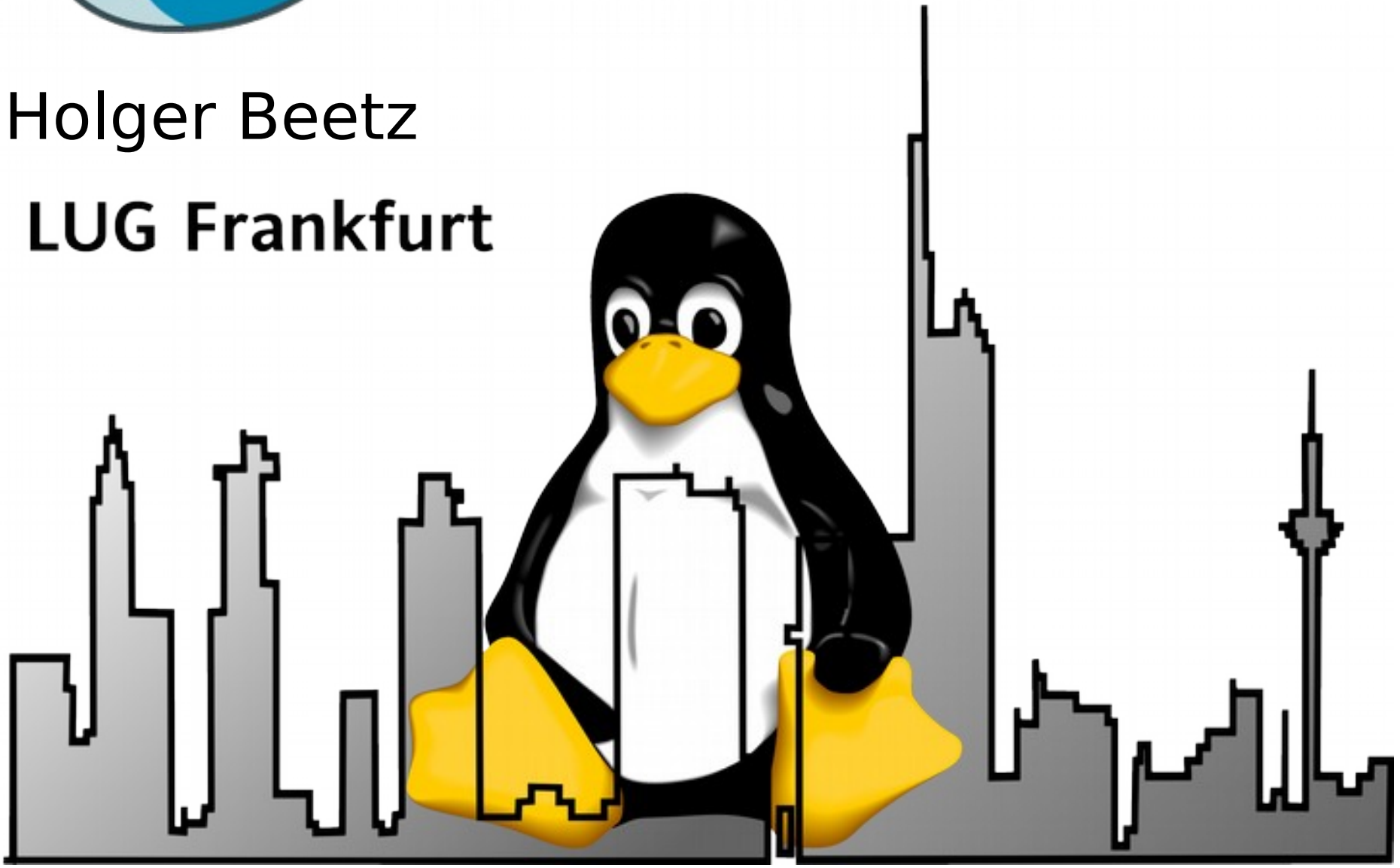
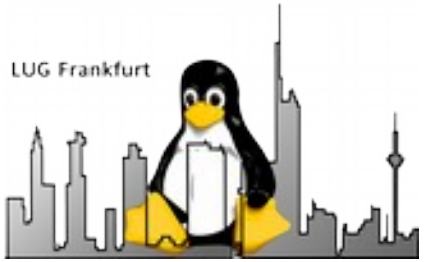


docker

Holger Beetz

LUG Frankfurt

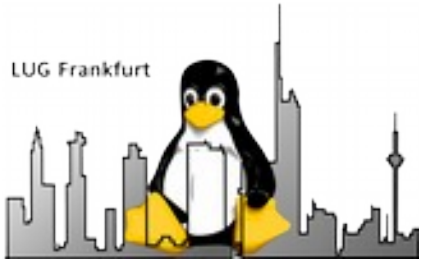




Herkömmlicher Betrieb einer Anwendung

Lieferant (Dev):
- Entwicklung
- Test
- Auslieferung

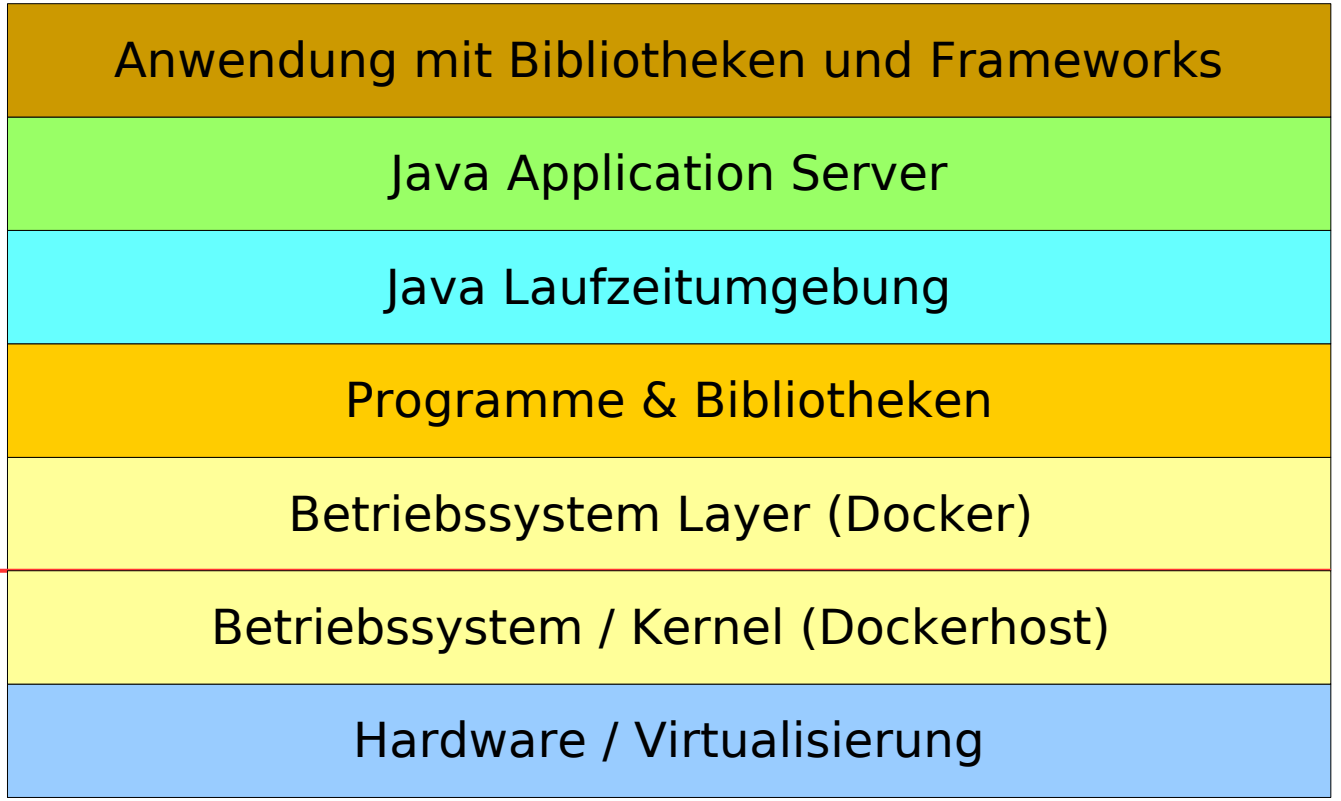
Kunde (Ops):
- Installation
- Betrieb
- Patchmanagement

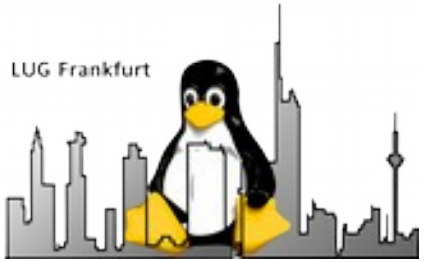


Betrieb einer Anwendung im „Container“

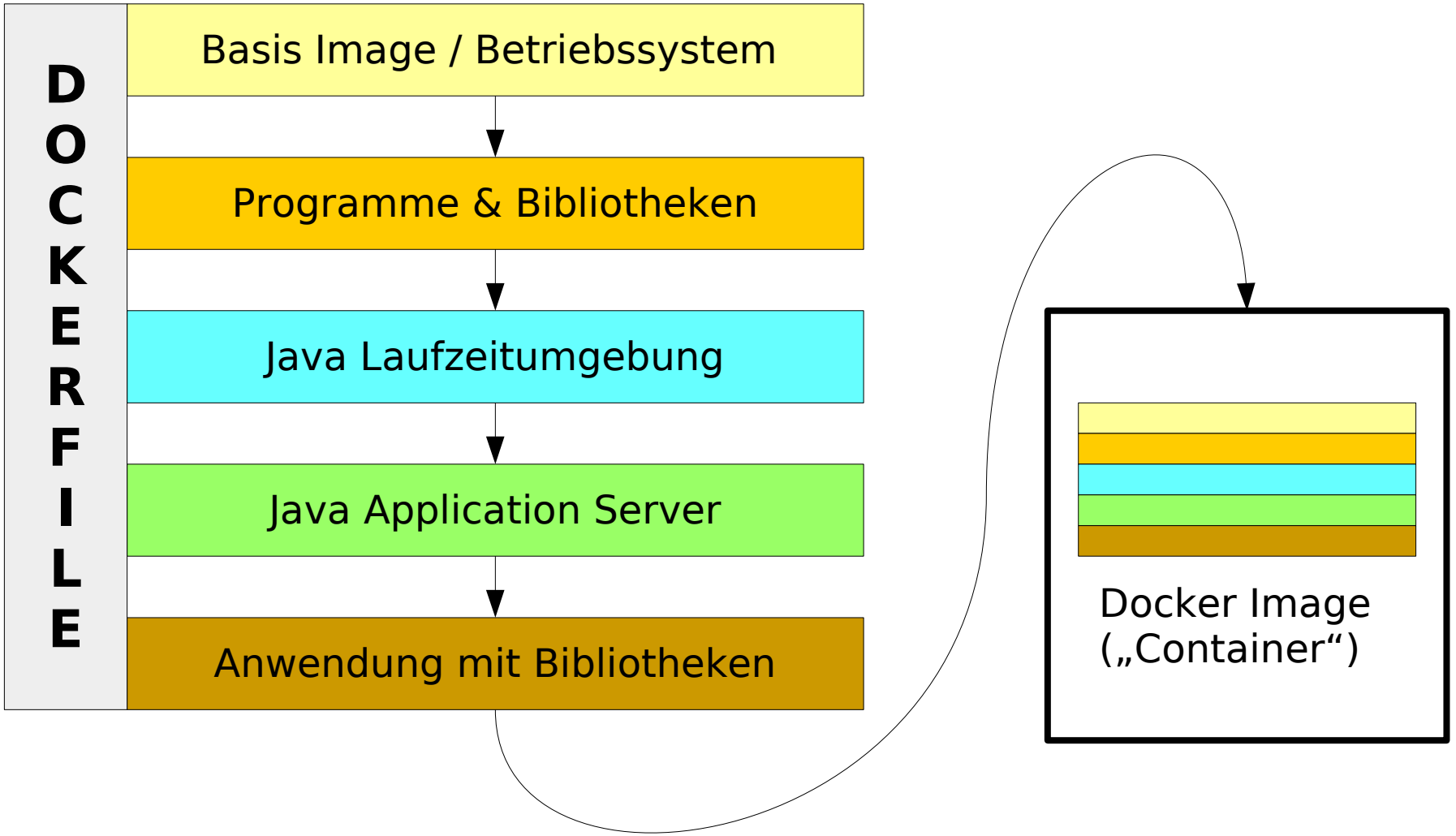
Lieferant (DevOps):
- Entwicklung
- Test
- Auslieferung
- Installation
- Betrieb
- Patchmanagement

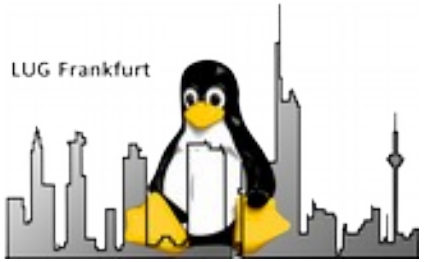
Kunde (Ops):
- Installation
- Betrieb
- Patchmanagement



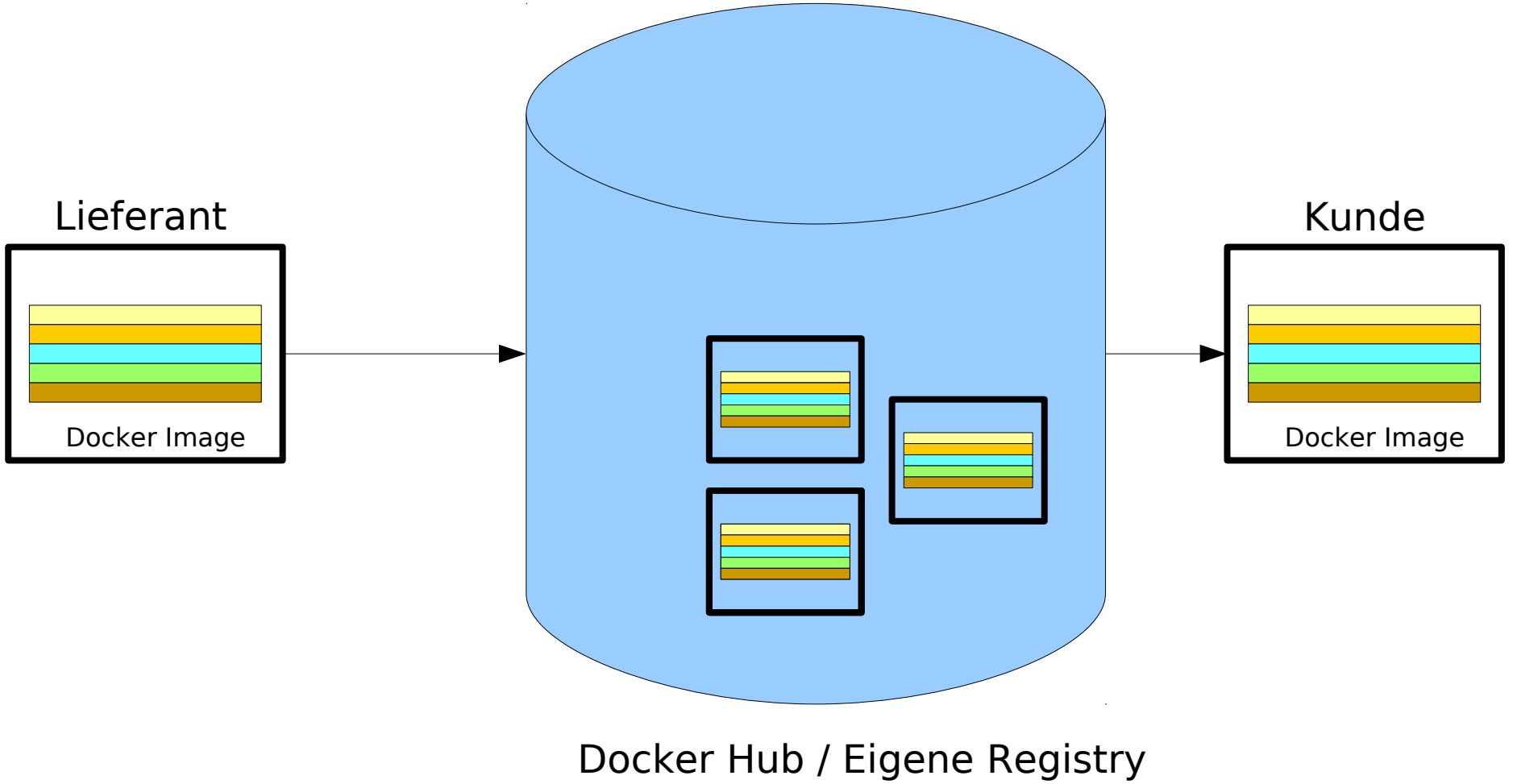


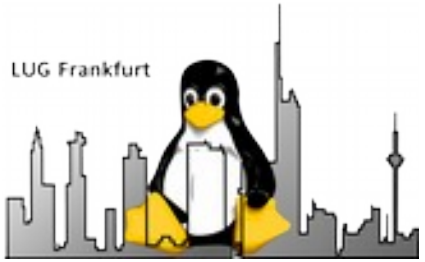
Erstellung eines „Containers“ mit einer Anwendung





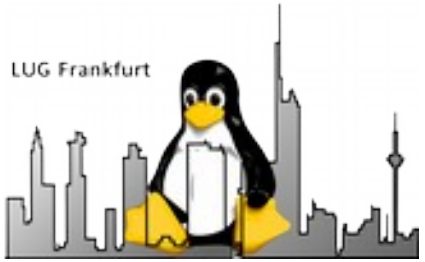
Auslieferung einer Anwendung mittels Docker





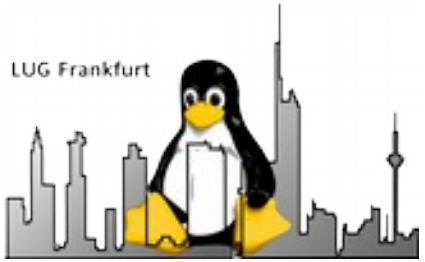
Was macht Docker aus ?

- Keine Vollvirtualisierung wie Xen, KVM, VMware, behive und viele Andere mehr.
- Benötigt keine Hardwarevirtualisierung in der CPU
  → super für stromsparende Plattformen wie ARM
  → effiziente Ausnutzung der Hardware.
- Versionskontrolliertes Speichern von Images zur Containererstellung.
- Kapselung der Anwendung und deren Abhängigkeiten.
- Breite Unterstützung & Integration durch namhafte Hersteller wie Microsoft, VMware oder redhat.



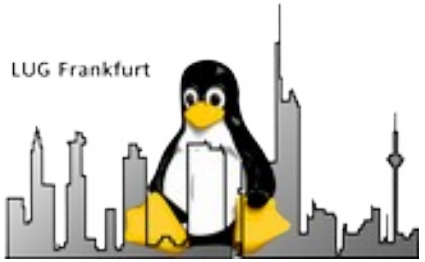
Docker im Linuxkontext

- Cgroups: Kontrolle über verwendete Ressourcen (CPU, RAM, Netzwerk).
- Namespaces: Eigene Devices für jeden Container wie veth.
- SELinux (optional): Abschottung der Container untereinander und zum Host.
- Transparente Netzwerkkommunikation nach außen durch NAT, eigenes DHCP und interne Bridge.
- Zugriff auf das native Dateisystem des Hosts.



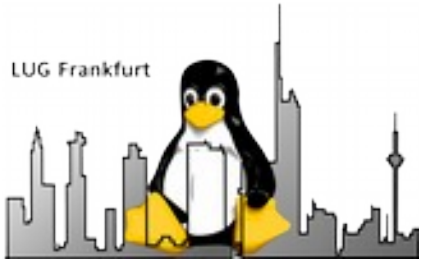
Docker unter der Haube: Networking

- Jeder Container hat eine interne IP
- Jeder Container kann mit allen anderen Container und dem Host kommunizieren.
- Ports werden bei Bedarf nach außen freigegeben.
- Anfragen vom Container nach draußen erfolgen über NAT → Somit ist eine transparente Kommunikation möglich.
- An der Bridge docker0 hängt ein eigener DHCP-Server.



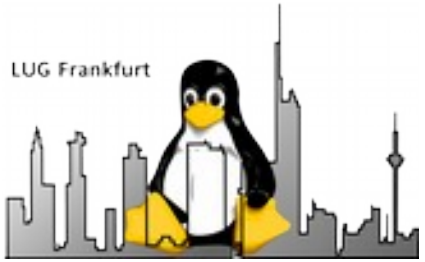
Docker unter der Haube: Filesystem

- Docker arbeitet mit einem Layered Filesystem.
- Images basieren häufig auf einem Basisimage einer bekannten Linux Distro.
- Jedes ADD in einem Dockerfile bewirkt ein neues Layer im Image.
- Schreiboperationen im Container über viele Layer kosten viel Zeit → Einsatz von Volumes



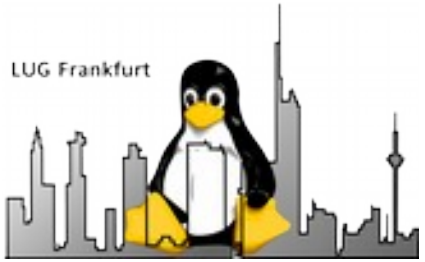
Der Docker-Slang

- Image: Datei, welche die Anwendung und die dazugehörenden Binaries & Libs enthalten.
- Container: Gestartete Anwendung, welche auf einem Image basiert.
- Registry: Repository in dem versionsverwaltete Images liegen.



Der Docker-Slang (ff)

- Docker Hub: Öffentliche Registry
- Docker Host: Linux-Server mit installiertem Docker, auf dem Container gestartet werden.
- Docker Engine: Service, welcher die Images & gestarteten Container auf einem Docker Host verwaltet.



Dockerfile Beispiel

FROM ubuntu:trusty

MAINTAINER Tux <tux@northpole.com>

RUN sudo apt-get update && sudo apt-get upgrade -y

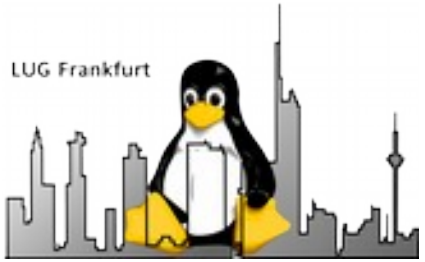
RUN sudo apt-get install -y apache2

*RUN sudo apt-get clean && sudo rm -rf /var/lib/apt/lists/**

EXPOSE 80 EXPOSE 443

ADD directory1 /var/www/directory1

ENTRYPOINT ["/usr/sbin/apache2", "-D", "FOREGROUND"]



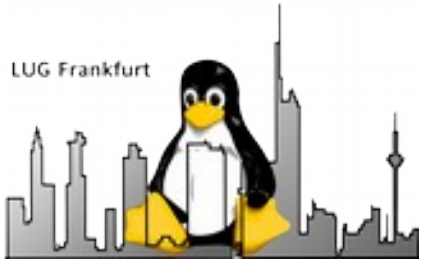
Bauen eines Images

docker build -f ***mein_dockerfile*** *-t* ***ubuntu_apache2*** *myPath*

-f = Pfad und Dateiname zum Dockerfile welches den Aufbau des Images beschreibt.

-t = Tag / Bezeichnung mit dem das Dockerfile später in der Registry erkennbar ist.

myPath = Gegenwärtiges Arbeitsverzeichnis auf das sich zum Beispiel Dateioperationen wie ADD beziehen, wenn lokale Dateien in das Images integriert werden sollen.



Speichern eines Image

Nach dem Bau wird das Image in den Docker Hub oder eine private Registry „gepushed“.

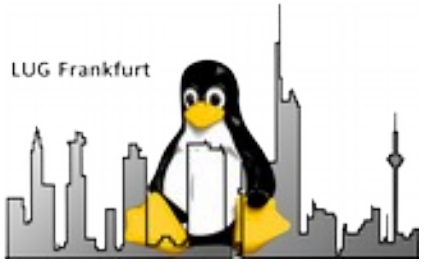
1. Schritt: Taggen des Images in der Registry
*docker tag **ubuntu_apache2** tux-registry.local.northpole/**ubuntu_apache2***

Der Server mit dem FQDN tux-registry.local.northpole ist die lokale Registry.

2. Schritt: Pushen des Images in die Registry
*docker push tux-registry.local.northpole/**ubuntu_apache2***

Eine Registry wird immer in dieser Notation angesprochen:
[REGISTRY_HOSTNAME/FQDN:REGISTRY_PORT]/IMAGENAME

Fehlt der Port, wird einfach vom Standardport 5000 ausgegangen.



Starten eines Containers

Vor dem ersten Start eines Containers wird das Image aus dem Docker Hub oder eine private Registry „gepulled“.

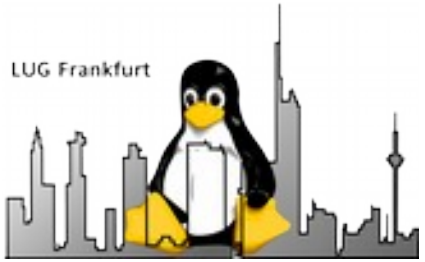
1. Schritt: Pullen des Images in der Registry
*docker pull tux-registry.local.northpole/**ubuntu_apache2***

2. Schritt: Starten eines Containers auf Basis des Images
*docker run -d **ubuntu_apache2** -p 8443:443 –name TUX_APACHE_1*

Der erste Schritt ist optional, da sonst Docker das Image vor dem Start aus der Registry / dem Dockerhub holt. Allerdings dauert der Start dann länger.

-d = Container wird als Dämon gestartet.

-p = < Externer Port>:<Interner Port> sorgt dafür, dass der https-Port des Containers vom Apache von aussen über den Hostnamen / FQDN des Dockerhosts unter Port 8443 erreichbar ist.



Arbeiten mit Containern

Anhalten des Containers:
docker stop TUX_APACHE_1

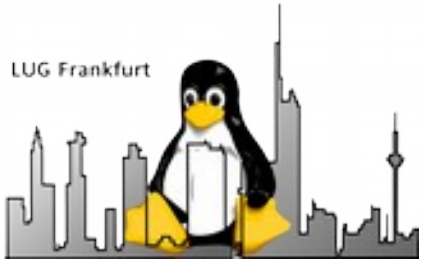
(Wieder)Starten des Containers:
docker start TUX_APACHE_1

Löschen des Containers:
docker rm TUX_APACHE_1

Anzeigen der Eigenschaften des Containers:
docker inspect TUX_APACHE_1

Speichern des Containers in ein lokales Tar:
docker save TUX_APACHE_1 > tux_apache_1.tar

Laden eines Containers aus einem lokalen Tar:
docker load < tux_apache_1.tar



Docker → zum Weiterlesen

- Docker Compose → Multi-tier container:
  https://docs.docker.com/compose/overview/
- Docker Swarm → Docker Hosts im Cluster:
  https://docs.docker.com/engine/swarm/
- Kubernetes → Enterprise Orchestration:
  http://kubernetes.io/
- OpenStack Schnittstelle für Docker:
  https://wiki.openstack.org/wiki/Docker

Danke fürs Mitnehmen und Tschüß