

31.8.–3.9.2015
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Docker Dir einen

Virtualisierung mit Docker

Simon Sudler

Redheads Ltd.



Docker Dir einen!

Virtualisierung mit Docker

Inhalt

- Warum Docker?
- Docker vs. Virtuelle Maschine
- Architektur
- Zombie-Reaping-Problem
- Schnellstart
- Eigenes Basis-Image erstellen
- Netzwerkanbindung
- Beispiel mit Gerrit, Jenkins und Build-Nodes

Warum Docker?

- Was ist das Problem?
 - Komplexe Anwendungen → viele Abhängigkeiten
 - Abhängigkeiten von unterschiedlicher Versionen gleicher Bibliotheken
 - Bibliotheken nicht identisch in unterschiedlichen Distributionen
 - Anwendungen benötigen spezielle Konfiguration verwendeter Systemdienste
 - Inkompatible Konfiguration der Systemdienste für verschiedene Anwendungen
- Lösung: Eigene virtuelle Umgebung für jede einzelne Anwendung

Warum Docker?

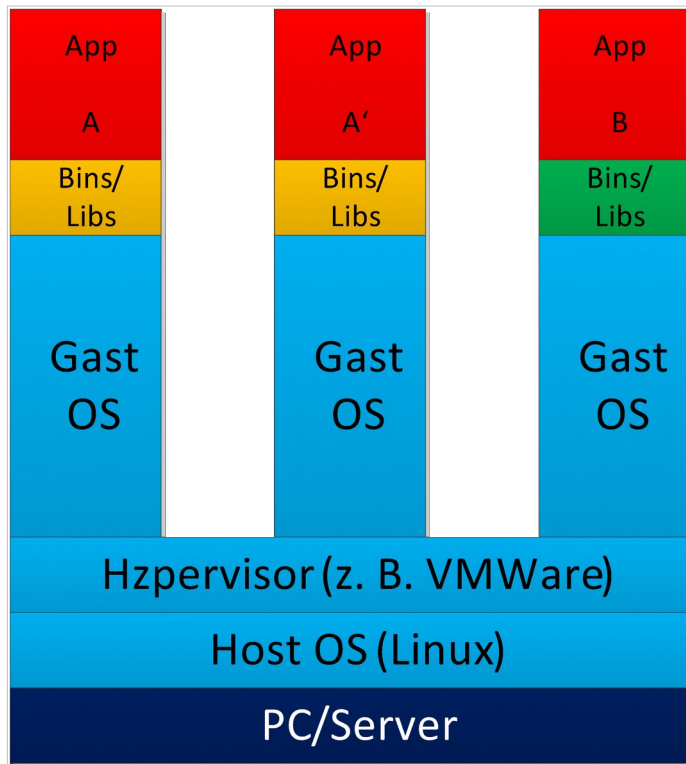
- Anforderungen an die virtuelle Umgebung:
 - Reproduzierbar: einfach, schnell, vollautomatisch
 - Sicher: isoliert vom Host-System
 - Erweiterbar: neue Systemdienste, neue Programme
 - Aktualisierbar: einfacher Update-Mechanismus
 - Austauschbar: einfacher Wechsel zwischen Versionen
 - Übertragbar: Lauffähig auf unterschiedlichen Host-Systemen
 - Effizient: minimaler Ressourcenverbrauch, kleine Images
 - ...

Docker vs. Virtuelle Maschine

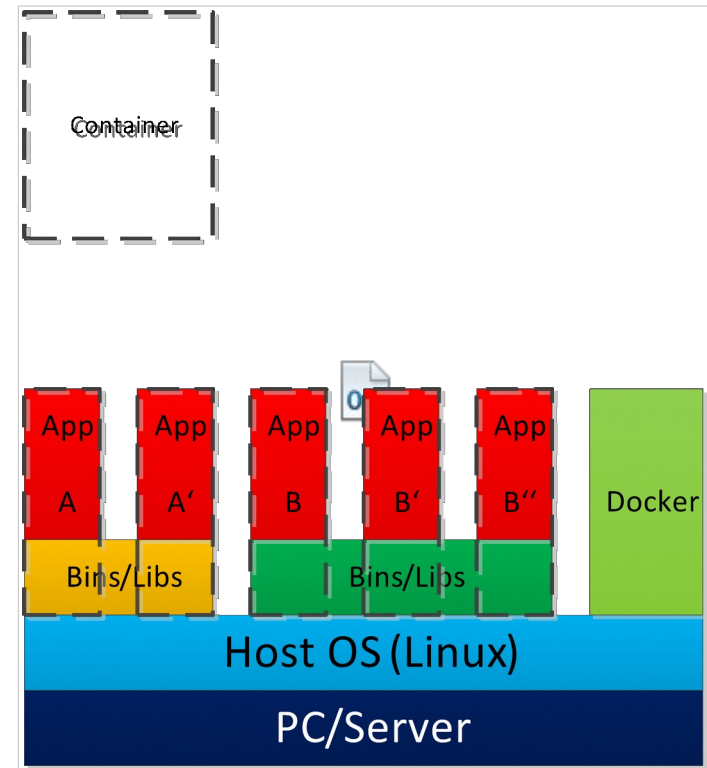
- Virtuelle Maschine
 - Virtualisiert ganzes System
 - Benötigt eigenes Betriebssystem
 - VM-Images können nicht zwischen virtuellen Maschinen geteilt werden
- Docker
 - Virtualisiert nur Applikationen
 - Nutzt Host-Betriebssystem
 - Binärdateien und Bibliotheken können zwischen Container geteilt werden

Docker vs. Virtuelle Maschine

- Virtuelle Maschine

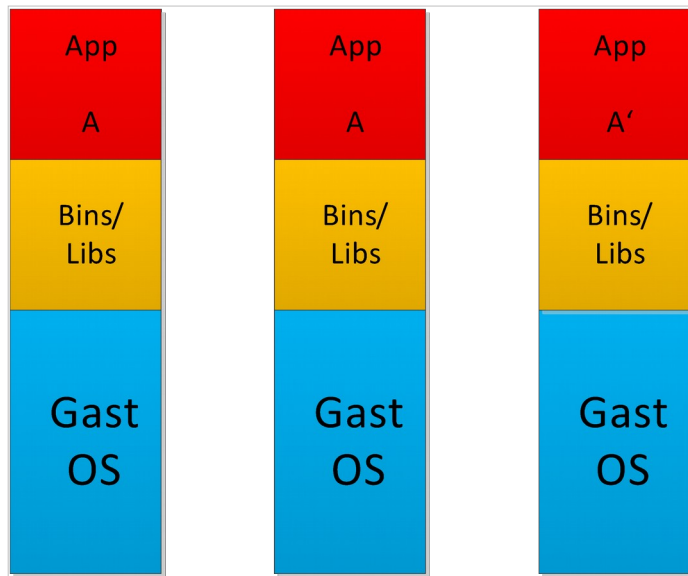


- Docker

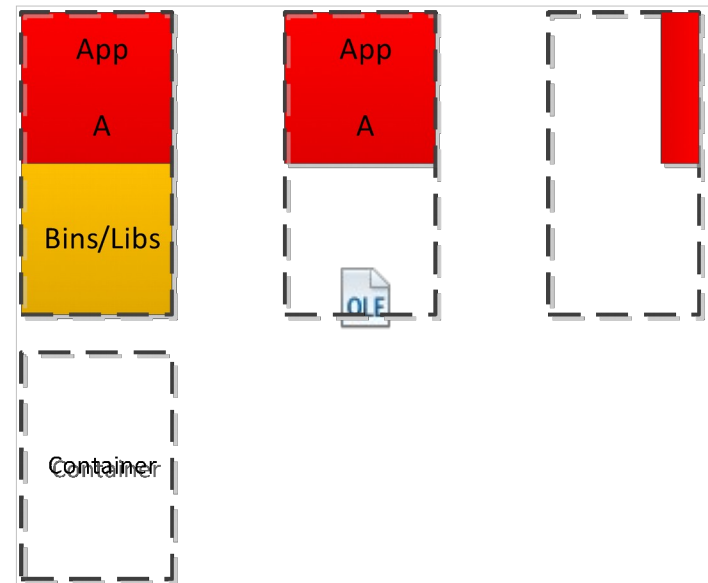


Docker vs. Virtuelle Maschine

- Virtuelle Maschinen
 - Jede Änderung, Kopie oder neue Version erfordert eigenes VM-Image



- Docker
 - Kopien benötigen kein OS und keine Bins/Libs
 - Änderungen werden differenziell gespeichert



Architektur

- Docker-Daemon (Server)
 - Läuft auf Host-System
 - Verwaltet Container
 - Anwender kommuniziert nicht direkt mit dem Server
- Docker-Client
 - Kommandozeilenprogramm „docker“
 - Kommuniziert mit Docker-Deamon
- Kommunikation zwischen Server und Client über Sockets und RESTful API

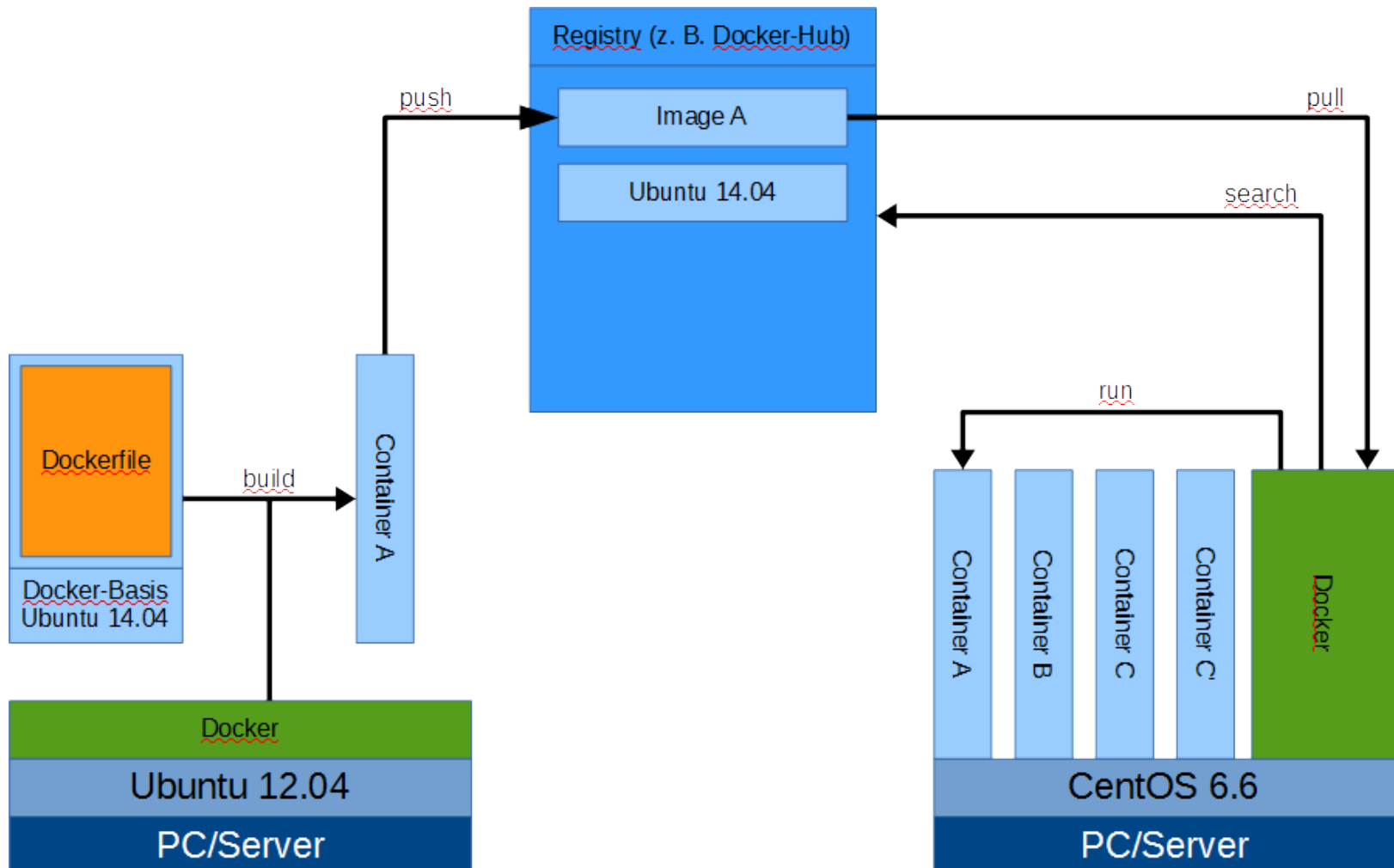
Architektur

- Docker-Images (build component)
 - Schreibgeschütztes Dateisystem
 - Enthält nur Anwendungen, kein Betriebssystem
 - Notwendig für das Erstellen von Docker-Containern
 - Einfach zu erstellen und zu verwalten
 - Einfach über eine Docker-Registry verteilbar
- Docker-Registry (distribution component)
 - Speichert Docker-Images
 - Öffentlich (z. B. Docker-Hub) oder nicht öffentlich

Architektur

- Docker-Container (run component)
 - Vergleichbar mit einem Inhaltsverzeichnis
 - Enthält alle Komponenten, um eine Anwendung auszuführen
 - Wird auf Basis eines Docker-Images erstellt
 - Container können ausgeführt, angehalten, gestoppt und gelöscht werden
 - Isolierte und sichere Anwendungsumgebung (cgroups)

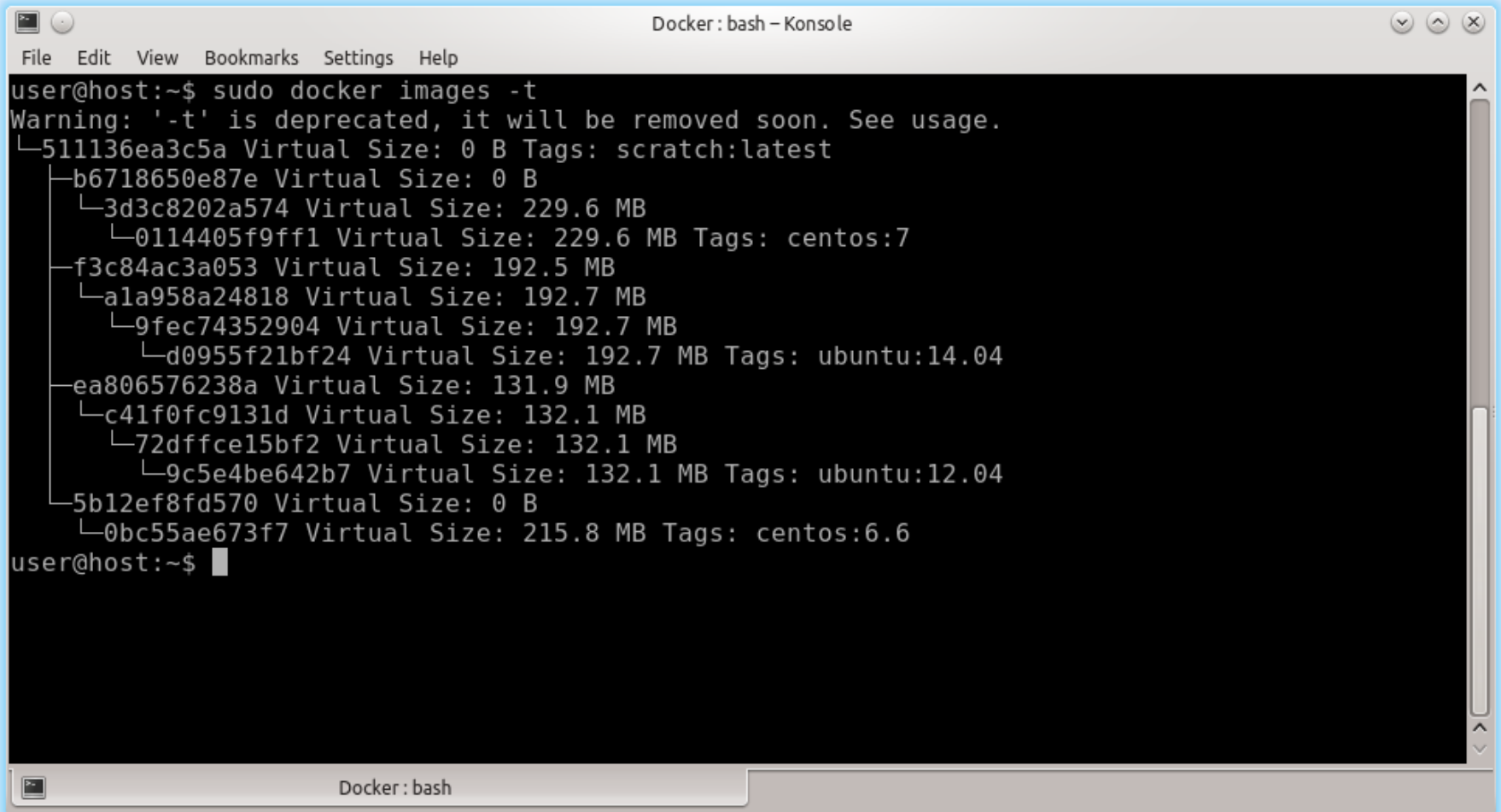
Architektur



Architektur – Docker-Images

- Images bestehen aus Layern
- Layer werden mittels UnionFS zu einem Dateisystem zusammen gefügt
- Änderungen an einem Image erzeugen einen neuen differenziellen Layer
- Update eines Images erfordert nur neuen Layer
- Layer starten von einem leeren Image (leeres Basis-Image (Ubuntu, CentOS, RHEL, ...))
- Basis-Image stammt meist von der Registry „Docker-Hub“

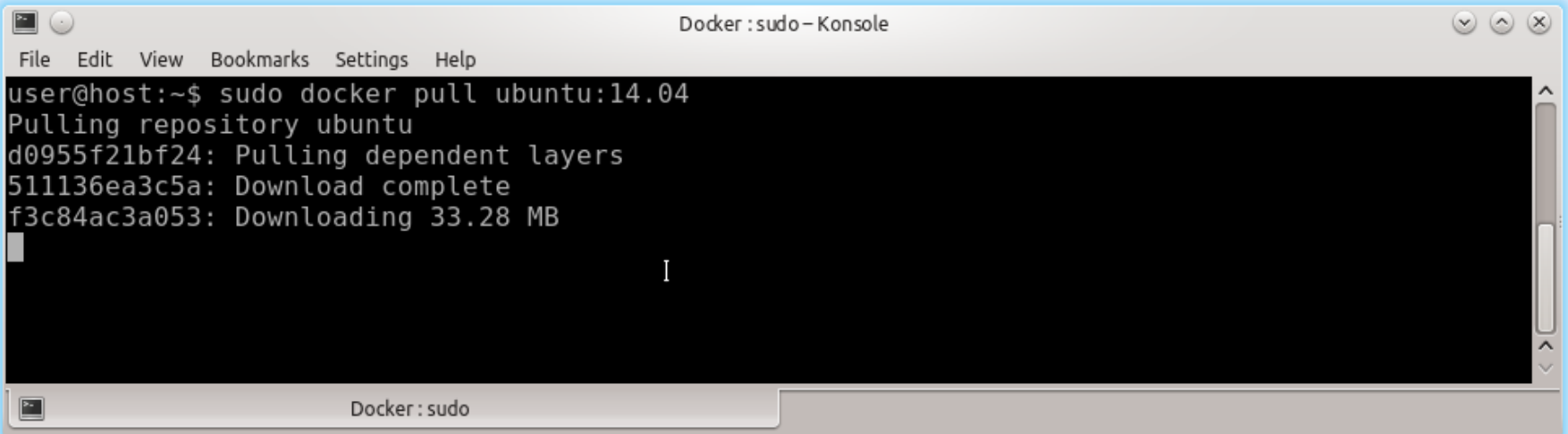
Architektur – Docker-Images



```
Docker: bash - Konsole
File Edit View Bookmarks Settings Help
user@host:~$ sudo docker images -t
Warning: '-t' is deprecated, it will be removed soon. See usage.
├─511136ea3c5a Virtual Size: 0 B Tags: scratch:latest
├─b6718650e87e Virtual Size: 0 B
├─┬3d3c8202a574 Virtual Size: 229.6 MB
├─┬└0114405f9ff1 Virtual Size: 229.6 MB Tags: centos:7
├─f3c84ac3a053 Virtual Size: 192.5 MB
├─┬a1a958a24818 Virtual Size: 192.7 MB
├─┬└9fec74352904 Virtual Size: 192.7 MB
├─┬└└d0955f21bf24 Virtual Size: 192.7 MB Tags: ubuntu:14.04
├─ea806576238a Virtual Size: 131.9 MB
├─┬c41f0fc9131d Virtual Size: 132.1 MB
├─┬└72dffcel5bf2 Virtual Size: 132.1 MB
├─┬└└9c5e4be642b7 Virtual Size: 132.1 MB Tags: ubuntu:12.04
├─5b12ef8fd570 Virtual Size: 0 B
├─└0bc55ae673f7 Virtual Size: 215.8 MB Tags: centos:6.6
user@host:~$
```

Architektur – Docker-Images

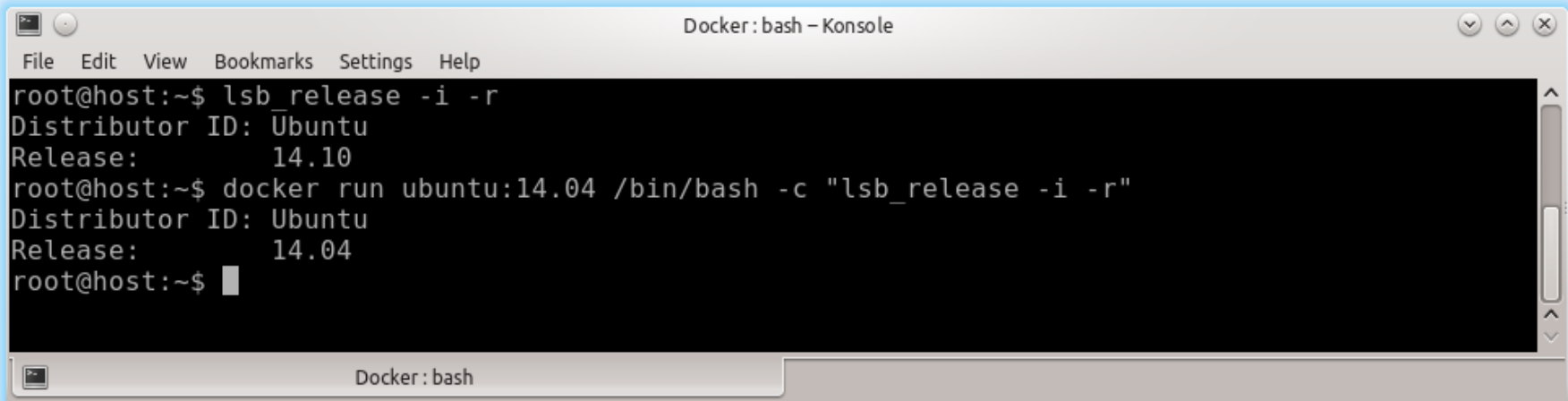
- Registry: Speichert Images
- Images können via Docker push/pull ausgetauscht werden
- Stellt Suchfunktionen für Images bereit



```
Docker : sudo – Konsole
File Edit View Bookmarks Settings Help
user@host:~$ sudo docker pull ubuntu:14.04
Pulling repository ubuntu
d0955f21bf24: Pulling dependent layers
511136ea3c5a: Download complete
f3c84ac3a053: Downloading 33.28 MB
█
I
```

Architektur – Docker-Images

- Container werden von einem Docker Image abgeleitet
- Container erzeugen einen beschreibbaren Layer auf den schreibgeschützten Basis-Images
- Startet das im Container angegebene oder beim Starten übergebene Programm



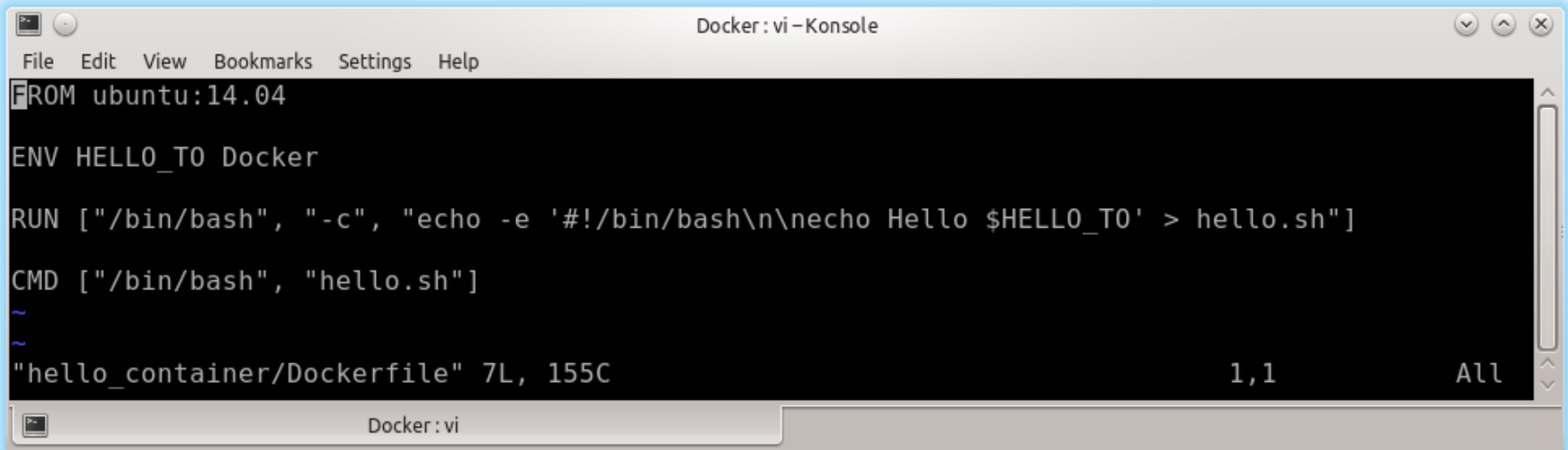
The screenshot shows a terminal window titled "Docker: bash - Konsole". The terminal displays the following commands and output:

```
root@host:~$ lsb_release -i -r
Distributor ID: Ubuntu
Release: 14.10
root@host:~$ docker run ubuntu:14.04 /bin/bash -c "lsb_release -i -r"
Distributor ID: Ubuntu
Release: 14.04
root@host:~$
```

The terminal window has a menu bar with "File", "Edit", "View", "Bookmarks", "Settings", and "Help". The bottom status bar shows "Docker: bash".

Architektur – Docker-Images

- Dockerfile: Instruktionsfolge zum Erstellen eines Images
 - Run: startet ein Programm innerhalb des Containers
 - Add: fügt dem Image Dateien oder Verzeichnisse hinzu
 - Env: erzeugt Umgebungsvariablen innerhalb des Images
 - Cmd: Anwendung, die beim Starten des Containers ausgeführt werden soll

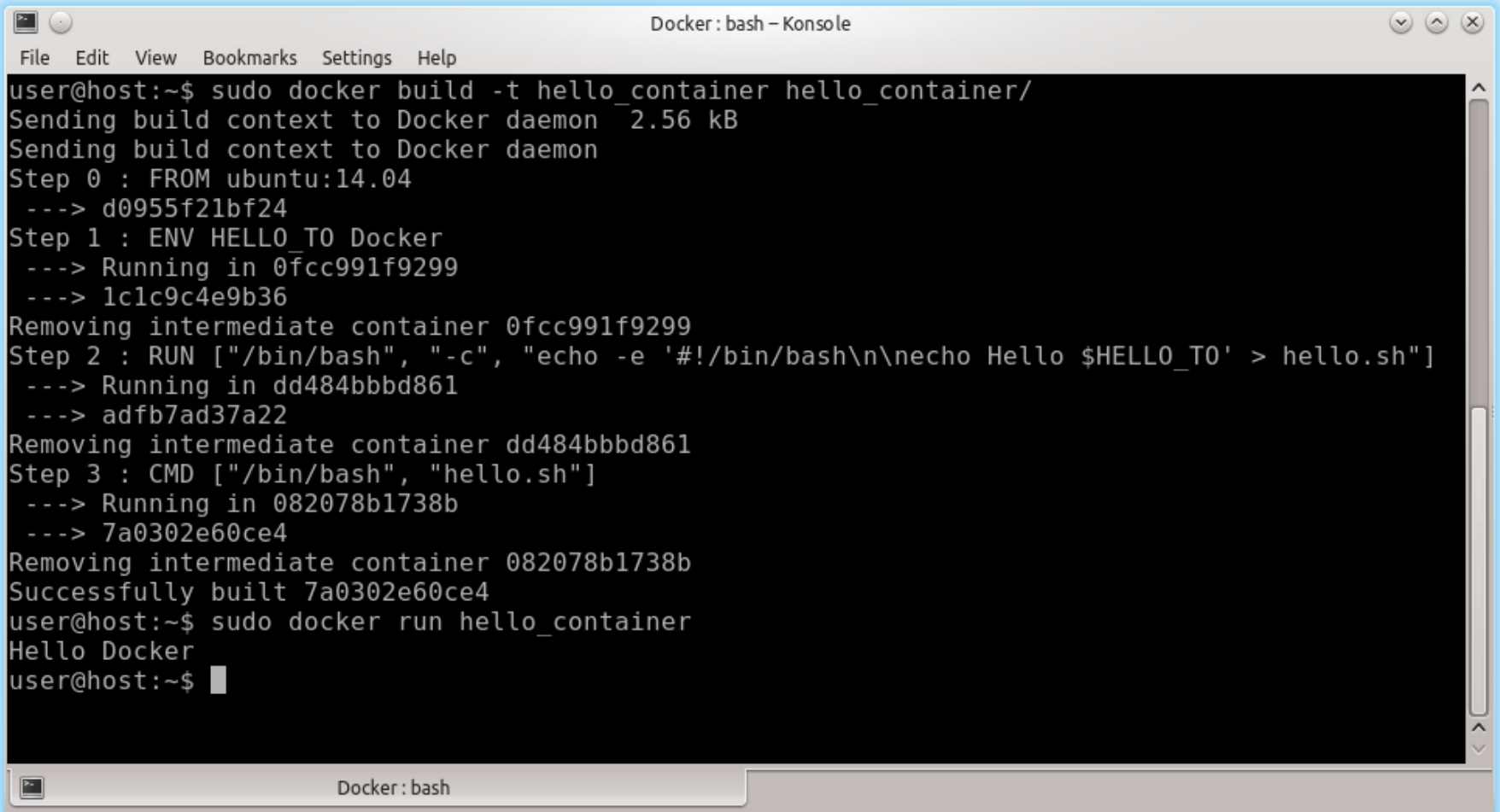


The screenshot shows a terminal window titled "Docker: vi - Konsole". The terminal content is a Dockerfile with the following instructions:

```
FROM ubuntu:14.04
ENV HELLO_TO Docker
RUN ["/bin/bash", "-c", "echo -e '#!/bin/bash\n\nnecho Hello $HELLO_TO' > hello.sh"]
CMD ["/bin/bash", "hello.sh"]
```

At the bottom of the terminal, the status bar shows the file path `"hello_container/Dockerfile"`, the cursor position `7L, 155C`, and the search results `1,1` and `All`.

Architektur – Docker-Images



```
Docker: bash – Konsole
File Edit View Bookmarks Settings Help
user@host:~$ sudo docker build -t hello_container hello_container/
Sending build context to Docker daemon 2.56 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:14.04
--> d0955f21bf24
Step 1 : ENV HELLO_TO Docker
--> Running in 0fcc991f9299
--> 1c1c9c4e9b36
Removing intermediate container 0fcc991f9299
Step 2 : RUN ["/bin/bash", "-c", "echo -e '#!/bin/bash\n\nnecho Hello $HELLO_TO' > hello.sh"]
--> Running in dd484bbbd861
--> adfb7ad37a22
Removing intermediate container dd484bbbd861
Step 3 : CMD ["/bin/bash", "hello.sh"]
--> Running in 082078b1738b
--> 7a0302e60ce4
Removing intermediate container 082078b1738b
Successfully built 7a0302e60ce4
user@host:~$ sudo docker run hello_container
Hello Docker
user@host:~$
```

Zombie-Reaping-Problem

- PID 1 Zombie-Reaping-Problem
 - Unix-Prozesse sind in einer Baumstruktur organisiert
 - Jeder Prozess hat einen Elternprozess, außer init(1)
 - Terminiert ein Prozess, wird er zum Zombie, bis der Elternprozess den Grund für das Terminieren erfragt (syscall waitpid(), AKA reaping)
 - Prozesse, die keine Elternprozess mehr haben, werden von init(1) übernommen → entfernt Zombies

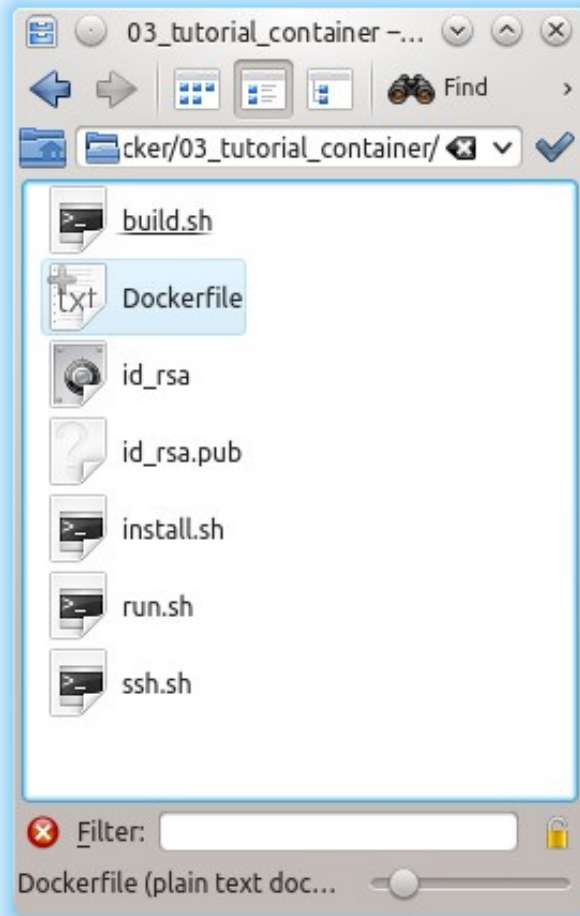
Zombie-Reaping-Problem

- Innerhalb eines Containers gibt es eine PID 1, aber keinen Init-Prozess!
- Docker-Anwendung wird mit PID 1 gestartet
- Bash „tötet“ adoptierte Kindprozesse (wenn sie zu Zombies werden):
 - DayZ: CMD ["/path-to-your-app"]
 - Besser: CMD ["/bin/bash", "-c", "set -e && /path-to-your-app"]

Schnellstart

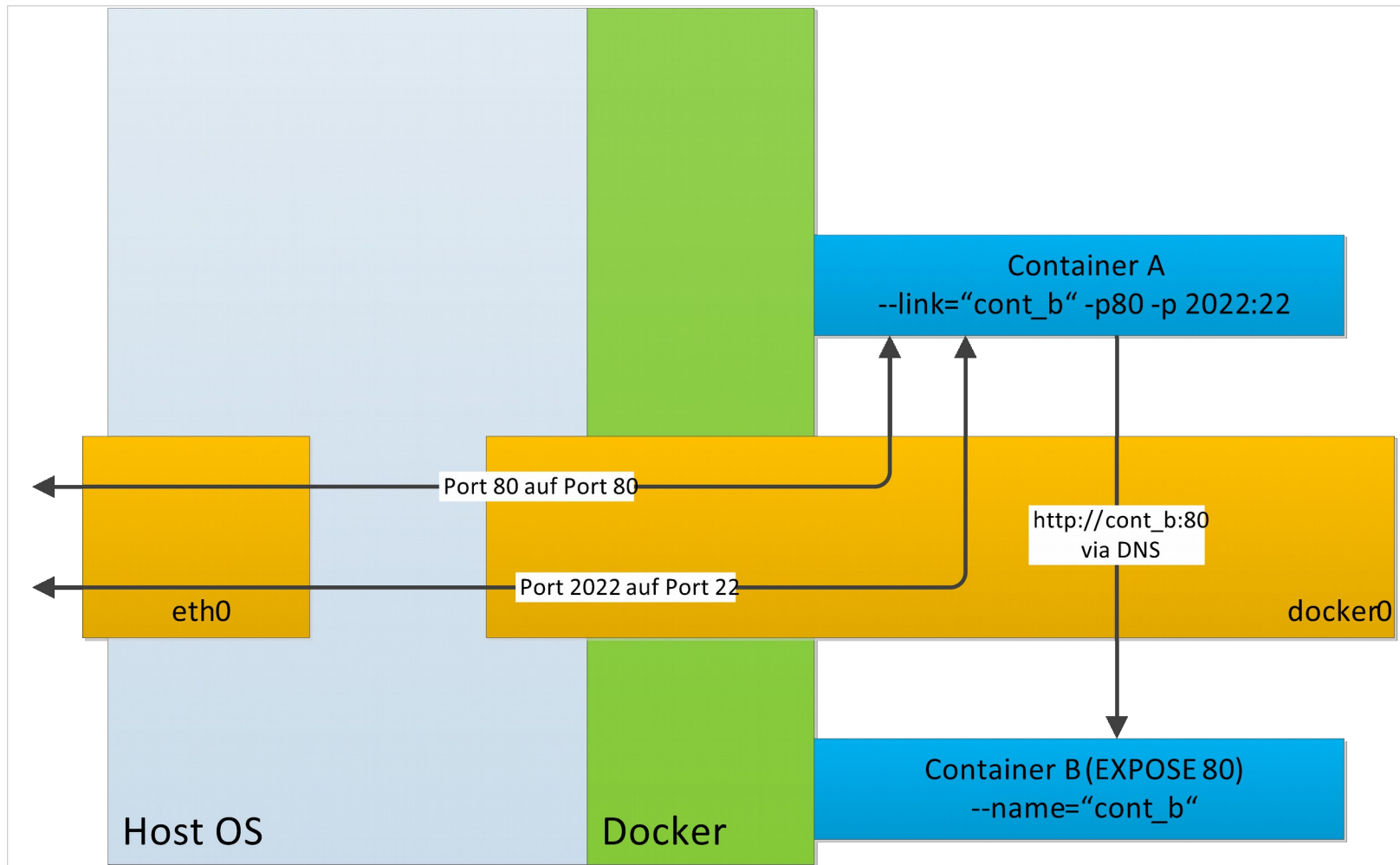
- Docker-Basis-Befehle des CLI (nicht vollständig)
 - pull: holt Image aus der Registry
 - build: erstellt ein neues Image
 - run: startet einen neuen Container
 - ps: zeigt die laufenden Container
 - stop/start: Container stoppen oder starten
 - attach: zu einem laufenden Container verbinden
 - rm: Container löschen
 - rmi: Image löschen
 - images: Images anzeigen

Eigenes Basis-Image erstellen

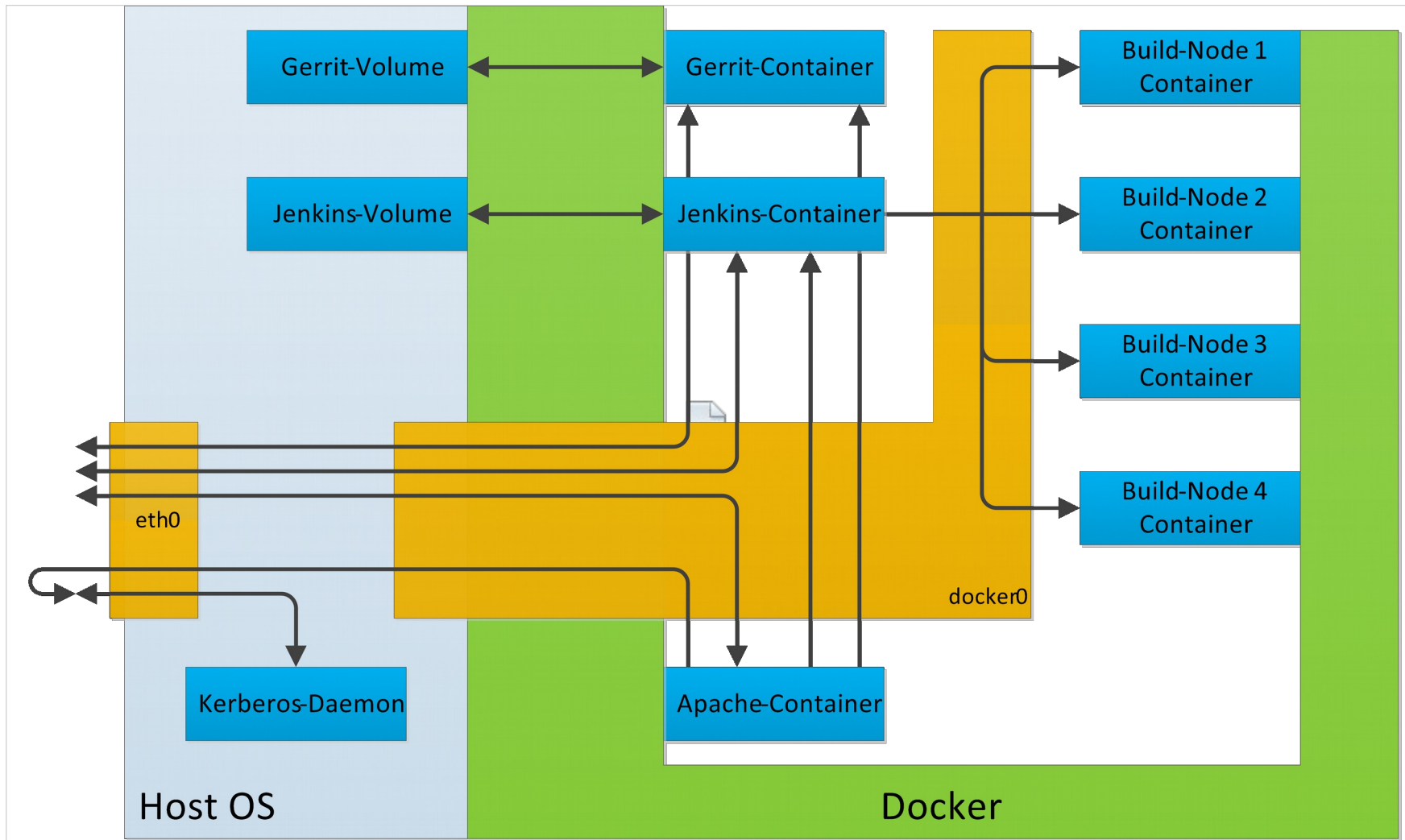


- Minimales Dockerfile
 - Kopiert install.sh und den öffentlichen SSH-Key
 - Install-Skript modifiziert Paketquelle, installiert und konfiguriert SSH-Server und Supervisord
- Build.sh, startet Image build
- Run.sh startet Container
- Ssh.sh ssh in den Container

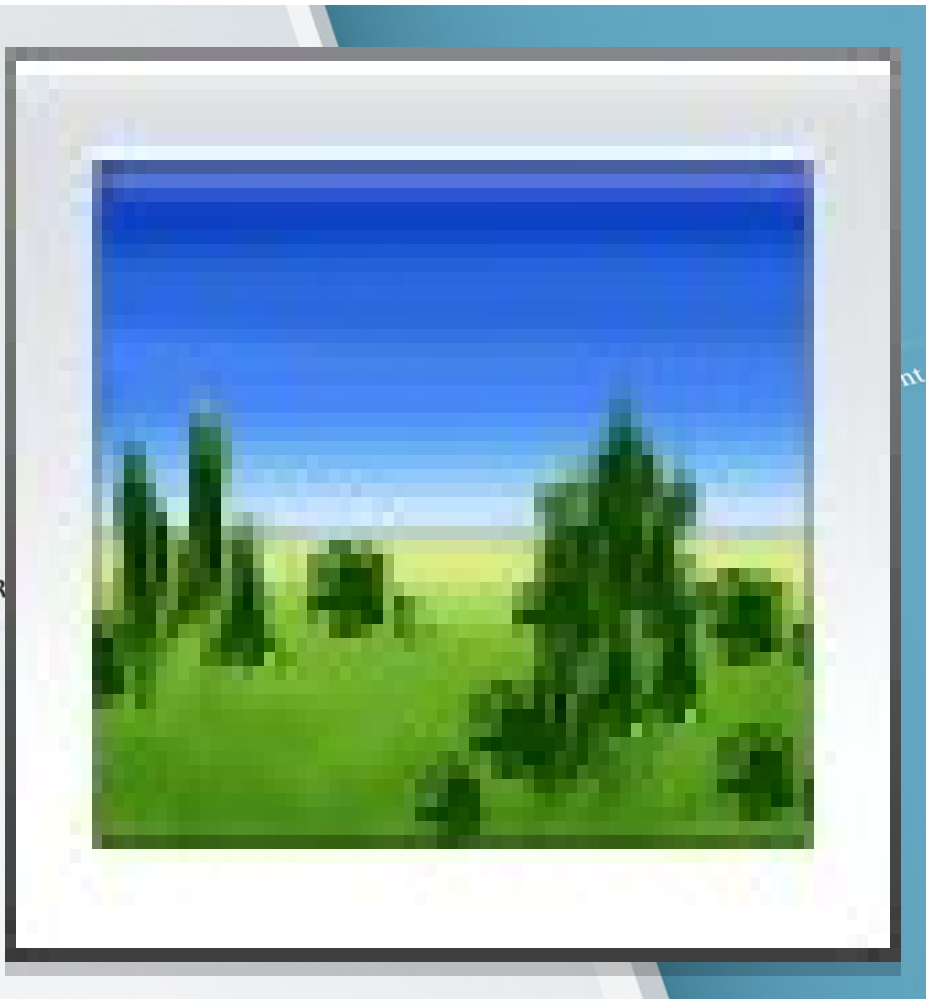
Netzwerkanbindung



Beispiel mit Gerrit, Jenkins und Build-Nodes



Vielen Dank für Ihre Aufmerksamkeit!



Simon Sudler

PD TI ATS TM4 1

Gleiwitzer Str. 555

90475 Nürnberg

Telefon: +49 (911) 895-4135

E-Mail:

simon.sudler@siemens.com

[siemens.com/answers](https://www.siemens.com/answers)