QAWARE
SOFTWARE ENGINEERING

TH BINGEN
University of Applied Sciences

Cloud Computing

Kapitel 4: Provisionierung

Simon Bäumler

simon.baeumler@qaware.de

TH Bingen, 21.11.2017

Eine kurze Geschichte der Systemadministration.

Ohne Virtualisierung (vor 2000)

- Manuelles Installieren von Betriebssystem auf dedizierter Hardware
- Manuelle Installation von Infrastruktur-Software
- Manuelle / Teilautomatisierte / Automatische Installation der Anwendungssoftware per Installer, Skript, proprietäre Lösungen

Virtualisierung einzelner Maschinen (2000 – heute)

- Manuelles Installieren von virtuellen Maschinen
- Manuelle Installation von Infrastruktur-Software
- Manuelle / Teilautomatisierte / Automatische Installation der Anwendungssoftware per Installer, Skript, proprietäre Lösungen

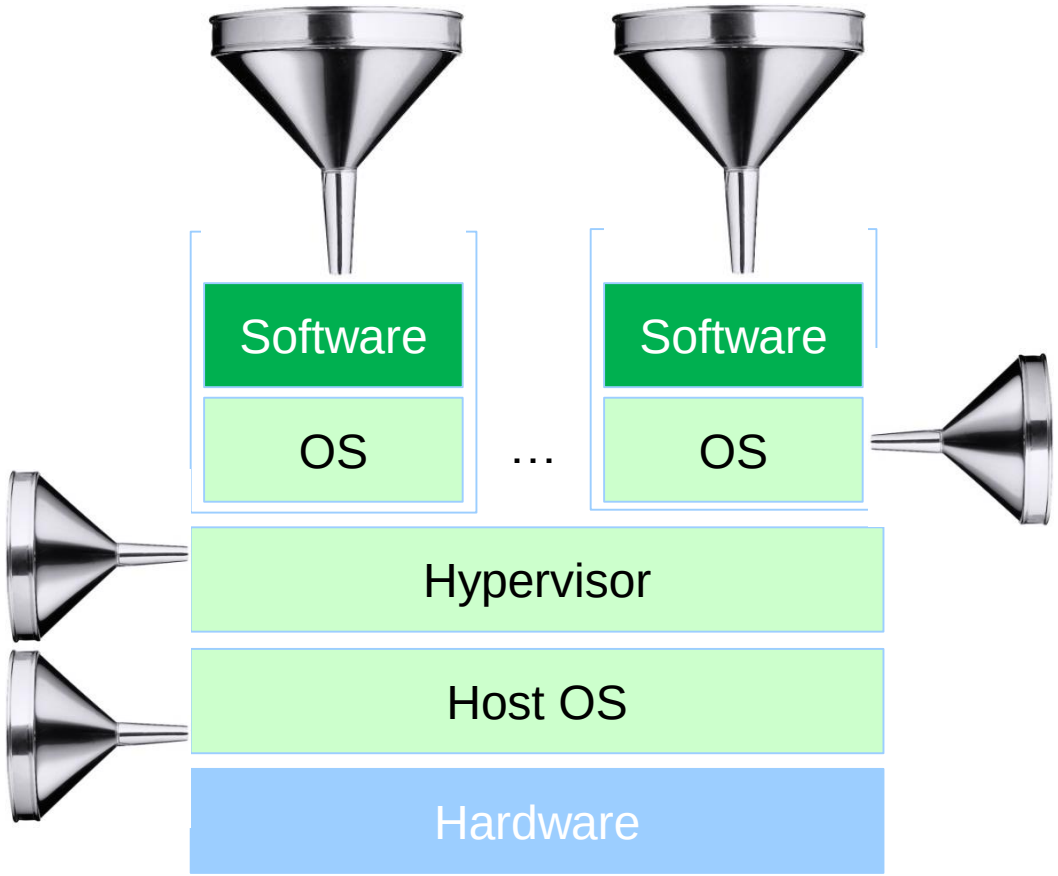
Virtualisierung in der Cloud (seit 2010)

- Automatisches Bereitstellen von Betriebssystem Klonen auf beliebiger Hardware
- Manuelle Installation der Infrastruktur-Software nur 1x im Klone-Master-Image
- Bereitstellen einer definierten Umgebung auf Knopfdruck

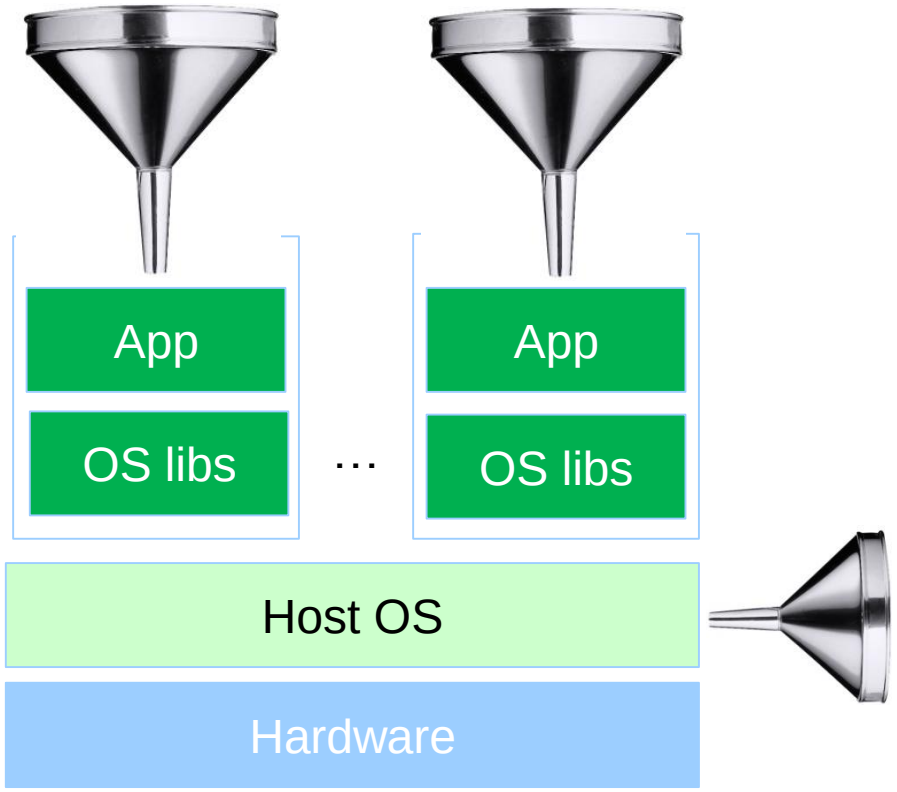
Infrastructure-as-code (2010 – heute)

- Programmierung der Provisionierung und weiterer Betriebsprozeduren

Provisionierung: Wie kommt Software in die Boxen?



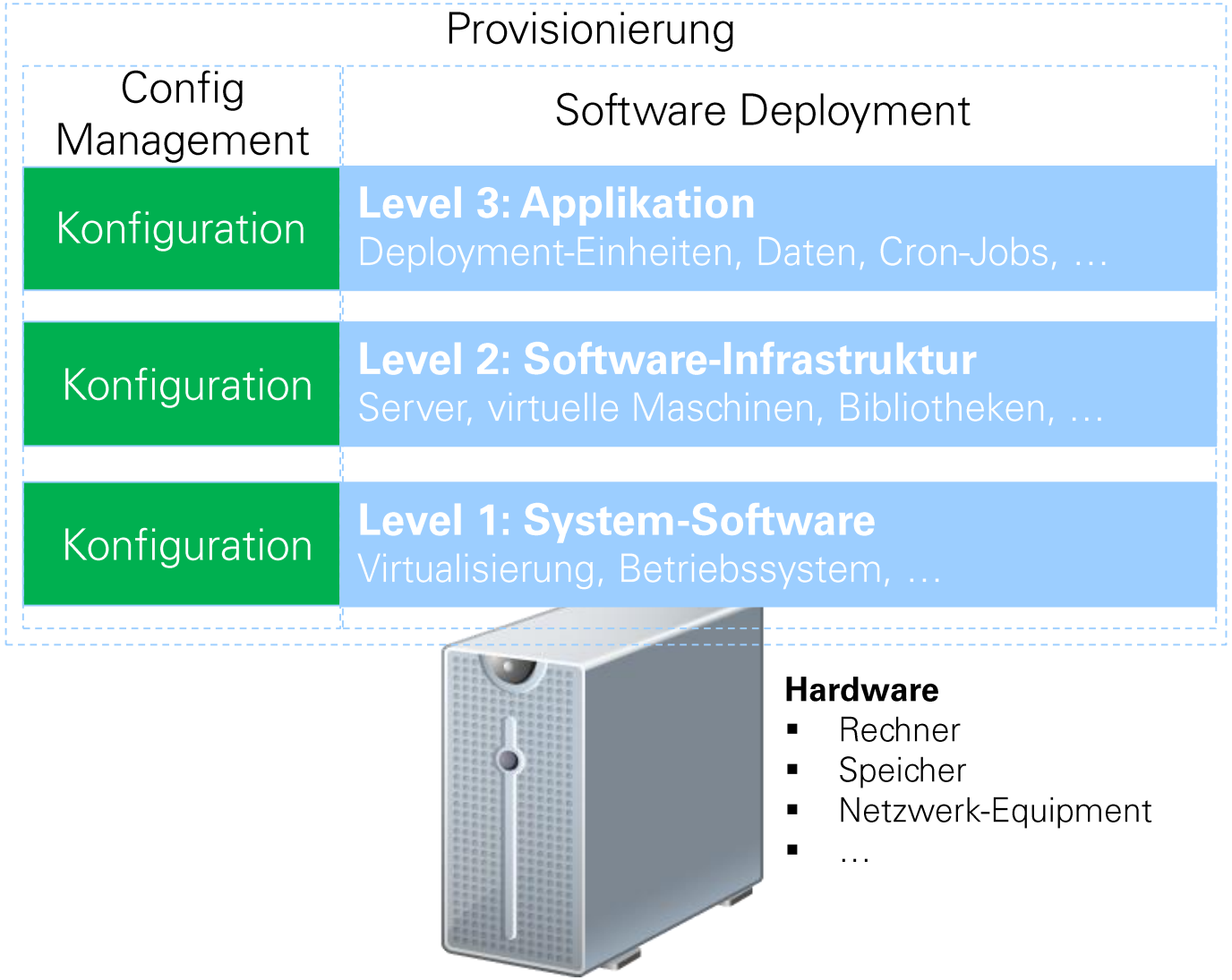
Hardware-Virtualisierung



Betriebssystem-Virtualisierung

Provisionierung ist die Bezeichnung für die automatisierte Bereitstellung von IT-Ressourcen.
http://wirtschaftslexikon.gabler.de/Definition/provisionierung.html

Provisierung erfolgt auf drei verschiedenen Ebenen und in vier Stufen.



Laufende Software!

4 **Application Provisioning**

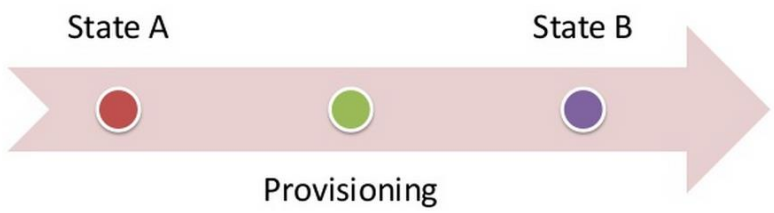
3 **Server Provisioning**
Bereitstellung der notwendigen Software-Infrastruktur für die Applikation.

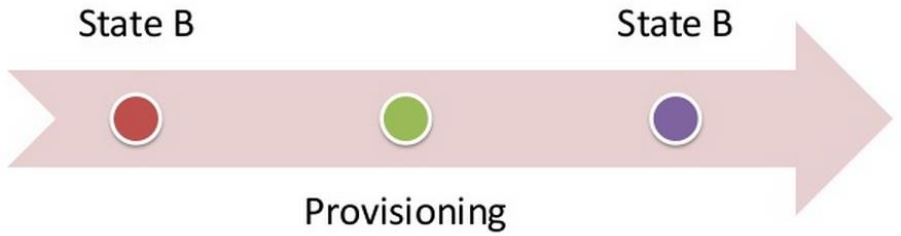
2 **Bootstrapping**
Bereitstellung der Betriebsumgebung für die Software-Infrastruktur.

1 **Bare Metal Provisioning**
Initialisierung einer physikalischen Hardware für den Betrieb.

Konzeptionelle Überlegungen zur Provisionierung.

Systemzustand := Gesamtheit der Software, Daten und Konfigurationen auf einem System.
Provisionierung := Überführung von einem System in seinem aktuellen Zustand auf einen Ziel-Zustand.





Was ein Provisionierungsmechanismus leisten muss:

1. Ausgangszustand feststellen
2. Vorbedingungen prüfen
3. Zustandsverändernde Aktionen ermitteln
4. Zustandsverändernde Aktionen durchführen
5. Nachbedingungen prüfen und ggf. Zustand zurücksetzen

Zusicherungen

Idempotenz: Die Fähigkeit eine Aktion durchzuführen und sie das selbe Ergebnis erzeugt, egal ob sie einmal oder mehrfach ausgeführt wird.

Konsistenz: Nach Ausführung der Aktionen herrscht ein konsistenter Systemzustand. Egal ob einzelne, mehrere oder alle Aktionen gescheitert sind.

Die neue Leichtigkeit des Seins.

Old Style

1. Ausgangszustand feststellen
2. Vorbedingungen prüfen
3. Zustandsverändernde Aktionen ermitteln
4. Zustandsverändernde Aktionen durchführen
5. Nachbedingungen prüfen und ggF. Zustand zurücksetzen

New Style

„Immutable Infrastructure / Phoenix Systems“

1. ~~Ausgangszustand feststellen~~
2. ~~Vorbedingungen prüfen~~
3. ~~Zustandsverändernde Aktionen ermitteln~~
4. Zustandsverändernde Aktionen durchführen
5. Nachbedingungen prüfen ~~und ggF. Zustand zurücksetzen~~

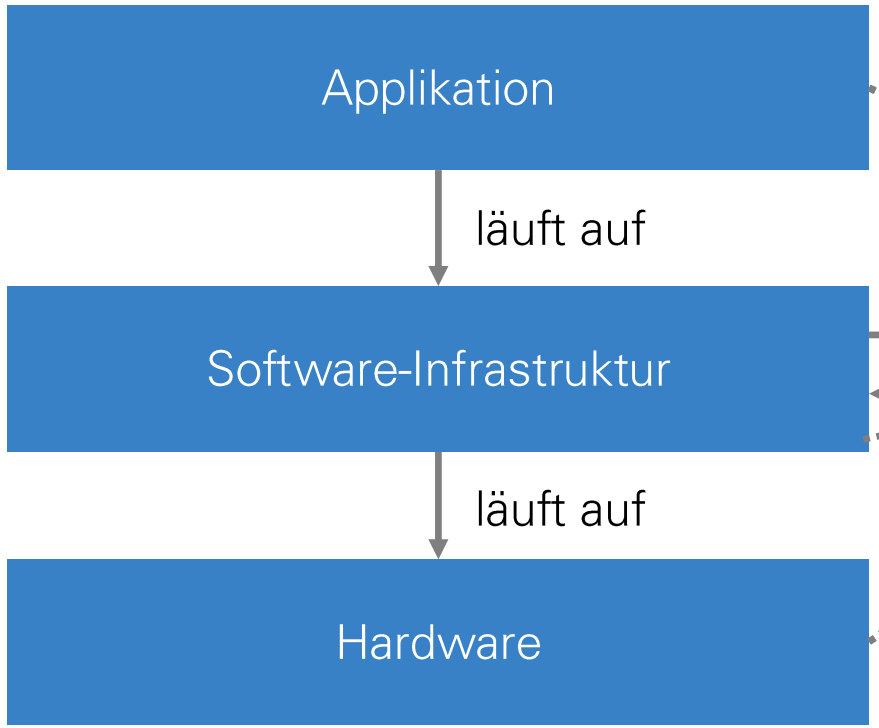
Eine Übersicht gängiger Provisionierungswerkzeuge.

Imperativ **Deskriptiv**

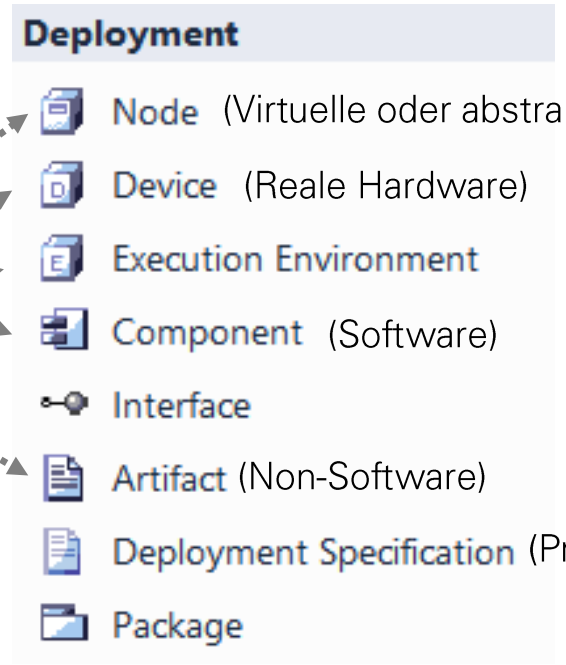
Shell Scripting Shell Abstraktion Zustandsautomaten

Bei virtualisierten und provisionierten Umgebungen helfen Modelle, um den Überblick zu behalten.

Die Bestandteile einer Ausführungssicht auf Systeme

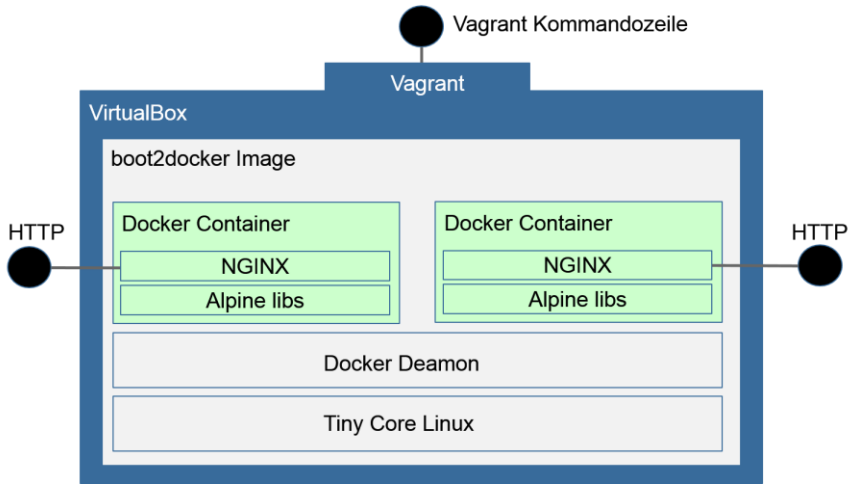


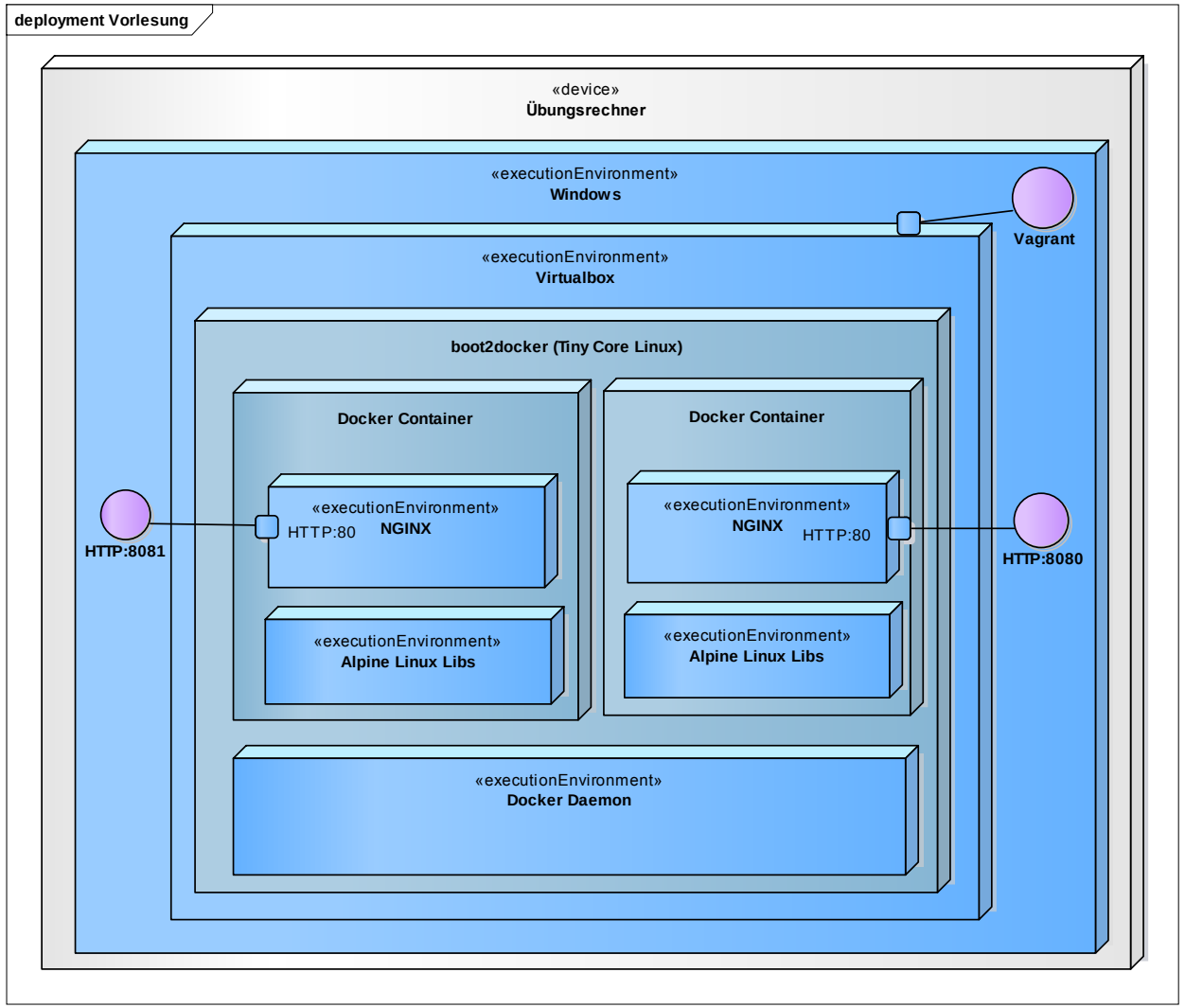
Elemente UML Deployment Modell



Neben der UML gibt es auch spezielle Modellierungssprachen für Cloud-Infrastruktur wie z.B. TOSCA

Vagrant Beispiel als UML Deployment Modell.





Dockerfiles

Provisionierung mit Docker

Deployment-Ebenen

Level 3: Applikation
Deployment-Einheiten, Daten, Cron-Jobs, …

Level 2: Software-Infrastruktur
Server, virtuelle Maschinen, Bibliotheken, …

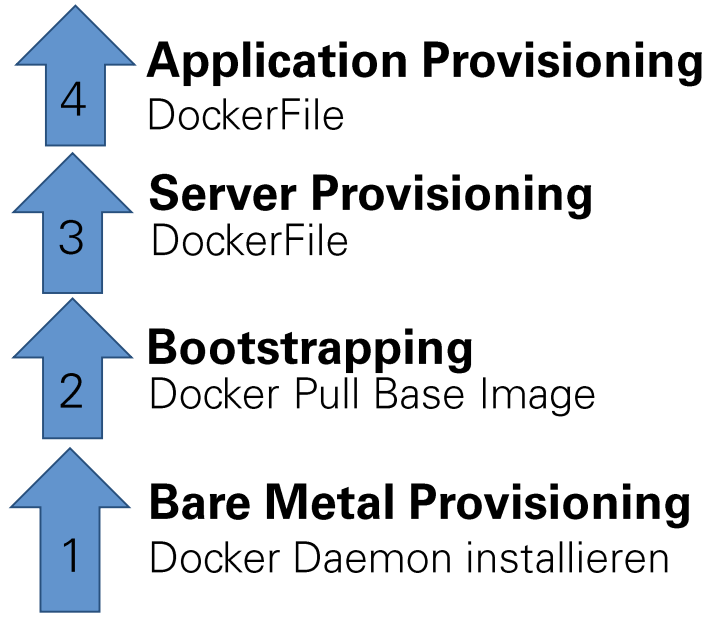
Level 1: System-Software
Virtualisierung, Betriebssystem, …

Docker-Image-Kette

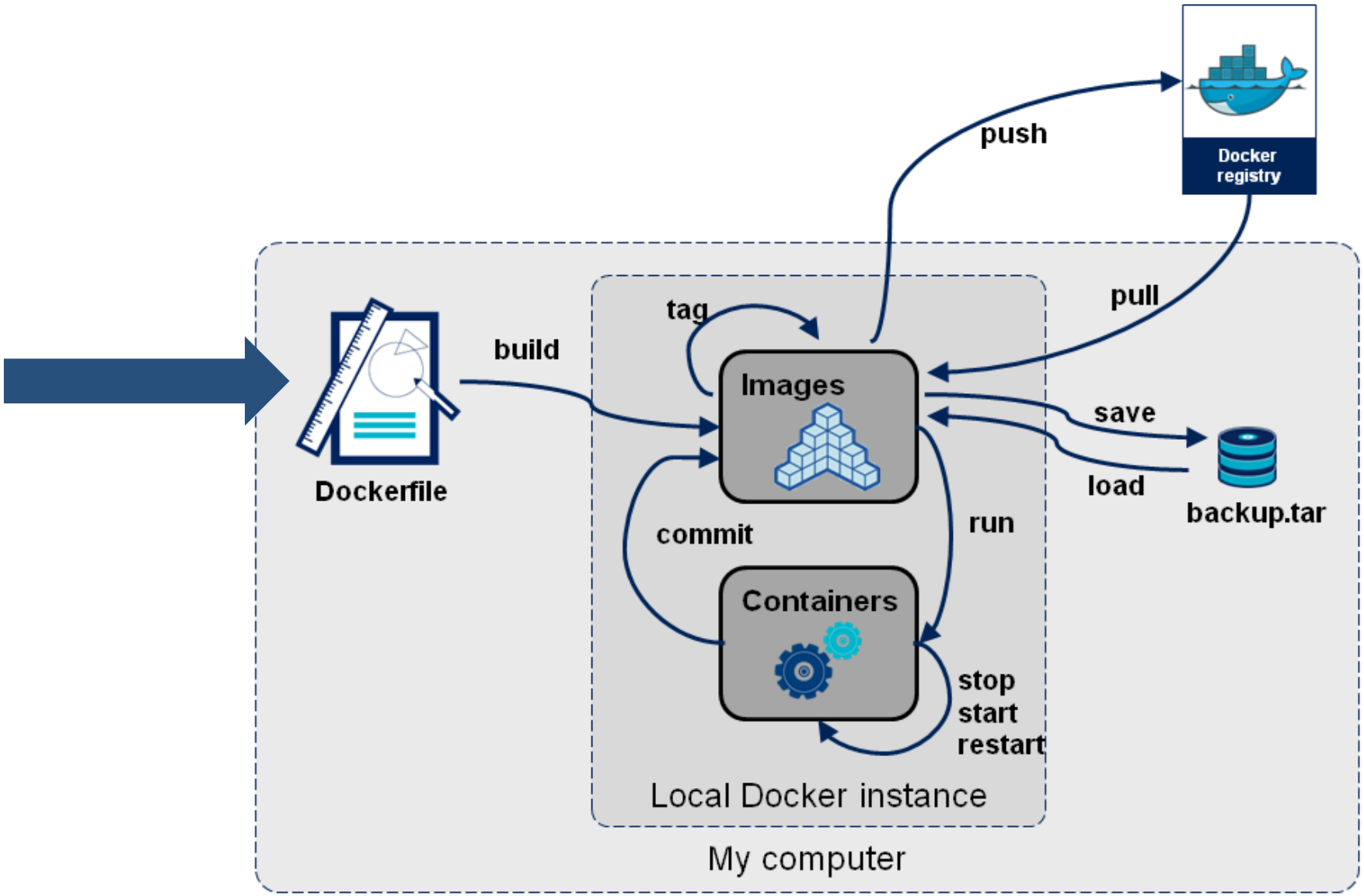
Applikations-Image
(z.B. www.qaware.de)

Server Image
(z.B. NGINX)

Base Image
(z.B. Ubuntu)



Provisionierung von Images mit dem Dockerfile.



Provisionierung von Images mit dem Dockerfile

Ein Dockerfile erzeugt auf Basis eines anderen Images ein neues Images. Dabei werden die folgenden Aktionen automatisiert:

- Konfiguration des Images und der daraus resultierenden Container
- Ausführung von Provisionierungs-Aktionen

Ein Dockerfile ist somit eine Image-Repräsentation alternativ zu einem physischen Image

(Bauanteilung vs. Bauteil).

- Wiederholbarkeit beim Bau von Containern
- Automatisierte Erzeugung von Images ohne diese verteilen zu müssen
- Flexibilität bei der Konfiguration und bei den benutzten Software-Versionen
- Einfache Syntax und damit einfach einsetzbar

Befehl: `docker build -t <ziel_image_name> <Dockerfile>`

Das Dockerfile wird zum Bau des Image verwendet

```
FROM centos:7.4.1708

RUN yum install -y epel-release && \
    yum install -y wget nginx && \
    yum install -y php php-mysql php-fpm && \

    sed -i -e "s/;\?cgi.fix_pathinfo\s*=\s*1/cgi.fix_pathinfo = 0/g" /etc/php.ini && \
    sed -i -e "s/daemonize = no/daemonize = yes/g" /etc/php-fpm.conf && \

    sed -i -e "s/;\?listen.owner\s*=\s*nobody/listen.owner = nobody/g" /etc/php-fpm.d/www.conf && \
    sed -i -e "s/;\?listen.group\s*=\s*nobody/listen.group = nobody/g" /etc/php-fpm.d/www.conf && \

    sed -i -e "s/user = apache/user = nginx/g" /etc/php-fpm.d/www.conf && \
    sed -i -e "s/group = apache/group = nginx/g" /etc/php-fpm.d/www.conf

COPY docker/php.conf /etc/nginx/default.d/

# COPY docker/index.html /usr/share/nginx/html/
# COPY docker/info.php /usr/share/nginx/html/

EXPOSE 80
ENTRYPOINT php-fpm && nginx -g 'daemon off;'
```

Dockerfile Commands

Element	Meaning
FROM <image-name>	Sets to base image (where the new image is derived from)
MAINTAINER <author>	Document author
RUN <command>	Execute a shell command and commit the result as a new image layer (!)
ADD <src> <dest>	Copy a file into the containers. <src> can also be an URL. If <src> refers to a TAR-file, then this file automatically gets un-tared.
VOLUME <container-dir> <host-dir>	Mounts a host directory into the container.
ENV <key> <value>	Sets an environment variable. This environment variable can be overwritten at container start with the –e command line parameter of `docker run`.
ENTRYPOINT <command>	The process to be started at container startup
CMD <command>	Parameters to the entrypoint process if no parameters are passed with `docker run`
WORKDIR <dir>	Sets the working dir for all following commands
EXPOSE <port>	Informs Docker that a container listens on a specific port and this port should be exposed to other containers
USER <name>	Sets the user for all container commands

http://docs.docker.com/engine/reference/builder

Dockerfile Best Practices

A Docker build must be repeatable.

- A build at a later time must produce an identical image.
- Keep care with versions
  - All files for the image are stored in the repository of the Dockerfile
  - No LATEST tag, use explicit versions instead
  - Always define a version when installing software

```
RUN apt-get update && apt-get install -y ruby1.9.1
```

Concatenate associated commands in the `RUN` command

- Every RUN command produces a Layer
- Less Layers are better for buildung and contributing images
- Concatenate commands with \

Installation of several software packages

```
RUN apt-get update && apt-get install -y wget \
        git-core=1:1.9.1-1 \
        subversion=1.8.8-1ubuntu3.2 \
        ruby=1:1.9.3.4 && \
        apt-get clean
```

Remove temporary files

- Remove all temporary files of the build process to produce small Docker Images
- Use the clean command
- Don‘t use the clean command in a separate RUN command (it is not possible to clean a different Layer)

Installation of a Linux Package with YUM

```
RUN yum -y install mypackage1 && \
        yum -y install mypackage2 && \
        yum clean all -y
```

Publish important ports with **EXPOSE**

- EXPOSE makes a port accessible for the host system or other containers
- Exposed Ports
  - are shown by the docker ps command
  - are executed in the image meta data by the docker inspect command
  - will be connected automatically by linked containers

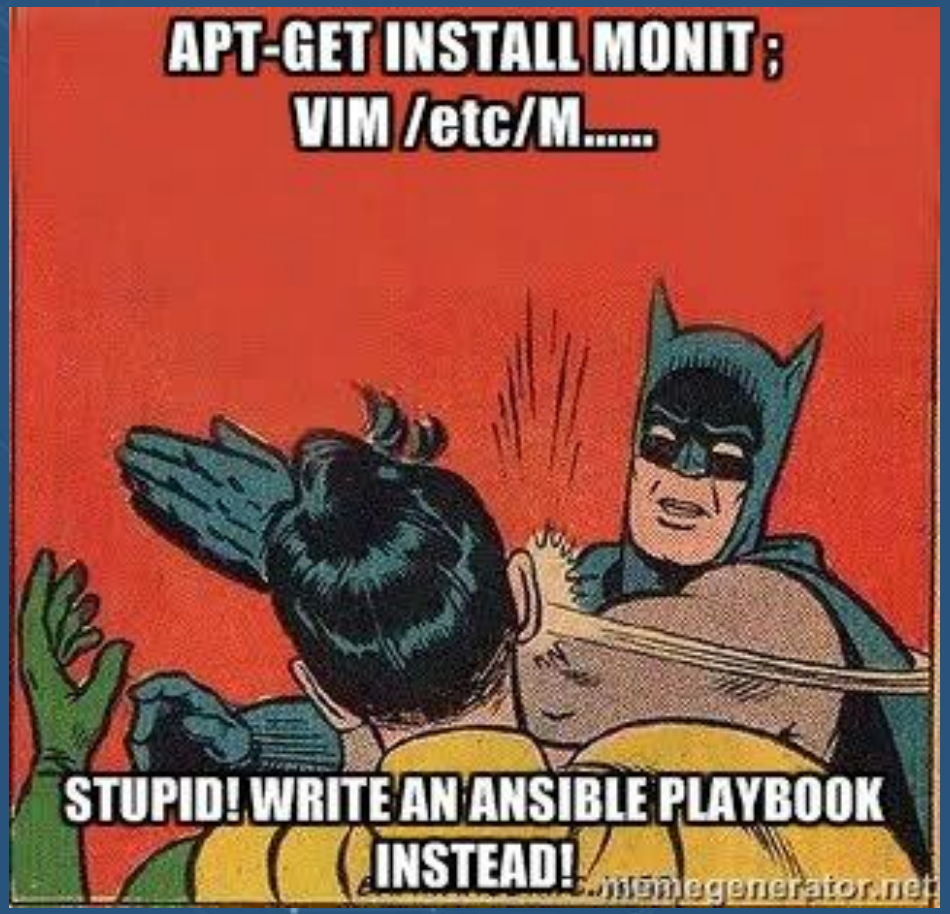
```
EXPOSE 12340
```

Define Environment Variables

- Visible in Dockerfile
- Can be used during Build and Excecution
- Can be overwritten at the start of a container

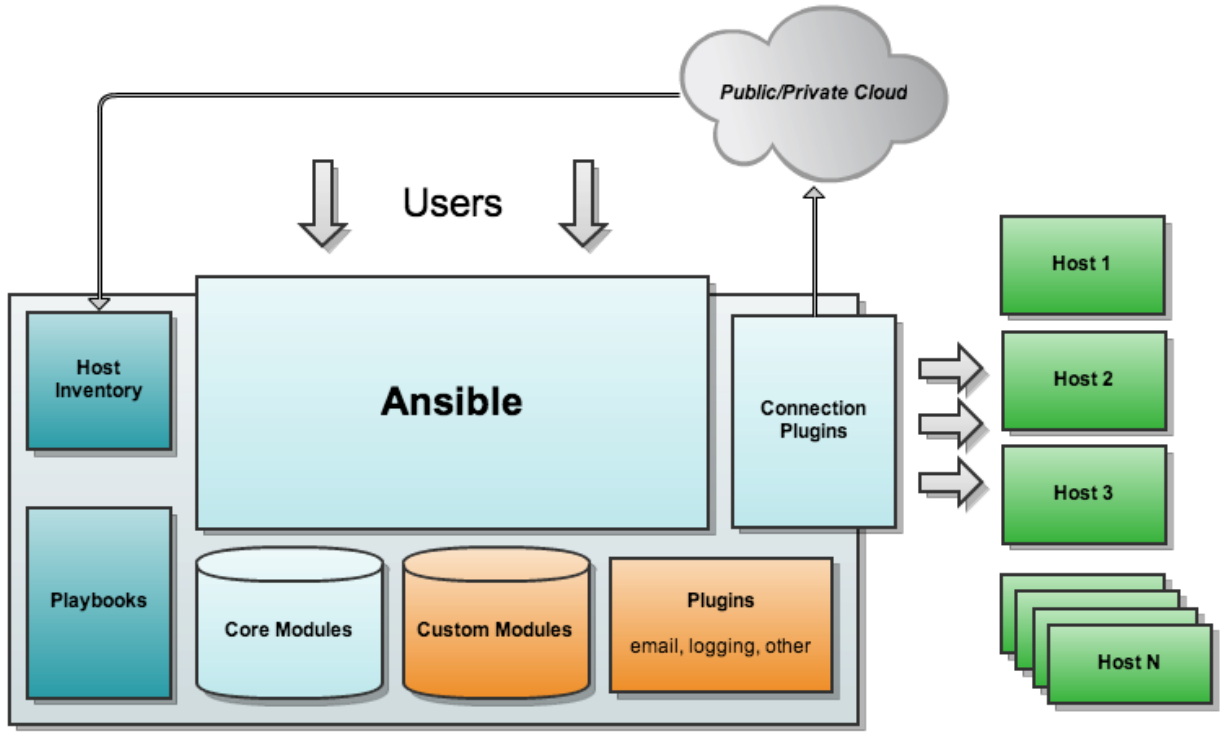
```
ENV JAVA_HOME /opt/java-oracle/jdk1.8.0_92
ENV MAVEN_HOME /usr/share/maven
```

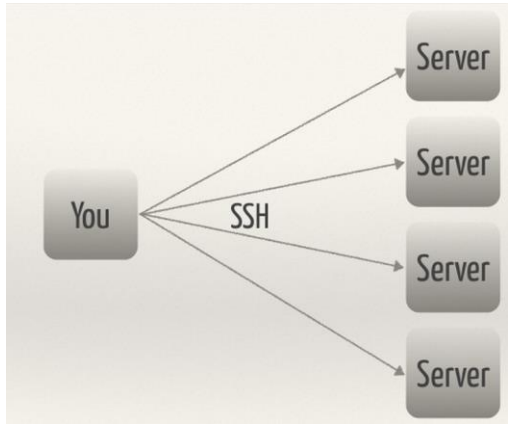
Ansible


APT-GET INSTALL MONIT ;
VIM /etc/M.....
STUPID! WRITE AN ANSIBLE PLAYBOOK
INSTEAD!
memegenerator.net

Ansible

- Open-Source-Provisionierungswerkzeug
- Kommerzielles Unternehmen steht hinter Ansible
- Ausgelegt auf die Provisionierung großer heterogener IT-Landschaften
- Entwickelt in der Sprache Python
- Push-Prinzip: Benötigt im Vergleich zu anderen Lösung weder einen Agenten auf den Ziel-Rechnern (SSH & Python reicht) noch einen zentralen Provisionierungs-Server
- Ist einfach zu erlernen im Vergleich zu anderen Lösungen. Deklarativer Stil.
- Umfangreiche Bibliothek vorgefertigter Provisionierungs-Aktionen inkl. Community-Funktion (https://galaxy.ansible.com) und Beispielen (https://github.com/ansible/ansible-examples)





Die wichtigsten zu erstellenden Dateien bei einer Provisionierung mit Ansible.

Playbook (YAML-Syntax)
Provisionierungs-Skript.

```
- hosts: all
  tasks:
  - yum: pkg=httpd state=installed
```

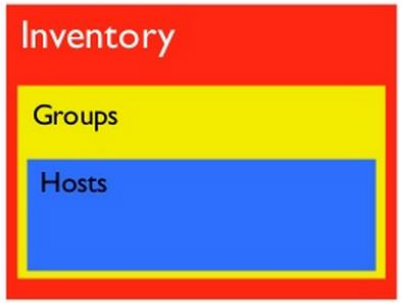
- Task = Beschreibung einer Provisionierungs-Aktion
- Role = Ausführung von Tasks auf Hosts oder Host-Gruppen
- Modul = Implementierung einer Provisionierungs-Aktion

Host Inventory
hosts

```
[mongo_master]
168.197.1.14

[mongo_slaves]
168.197.1.15
168.197.1.16
168.197.1.17

[www]
168.197.1.2
```



Ansible Konfiguration
ansible.cfg

```
[defaults]
host_key_checking = False
hostfile          = /ansible/hosts
private_key_file  = /ansible/id_rsa
```

Es stehen in Ansible viele vorgefertigte Module zur Verfügung.

Module Index

- All Modules
- Cloud Modules
- Commands Modules
- Database Modules
- Files Modules
- Inventory Modules
- Messaging Modules
- Monitoring Modules
- Network Modules
- Notification Modules
- Packaging Modules
- Source Control Modules
- System Modules
- Utilities Modules
- Web Infrastructure Modules
- Windows Modules

http://docs.ansible.com/modules_by_category.html
http://docs.ansible.com/list_of_all_modules.html

Die Provisionierung wird über die Kommandozeile gesteuert.

- Ad-hoc Kommandos
  - ansible <host gruppe> -m <modul> -a „<parameter>“
  - Beispiele:
    - ansible all –m ping
    - ansible all –a „/bin/echo di“
    - ansible web –m apt –a „name=nginx state=installed“
    - ansible web –m service –a „name=nginx state=started“
- Playbooks ausführen
  - ansible-playbook <playbook>

Provisionierung von Vagrant Boxen mit Ansible:

```
config.vm.provision "ansible" do |ansible|
  ansible.playbook = "playbook.yml"
  ansible.sudo = true
 end
```

```
vagrant provision
```

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service: name=httpd state=started
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Zur Ansible Provisionierung wird ein Ansible Client auf dem Host Rechner benötigt. Dieser steht aktuell für Windows nicht zur Verfügung.

Trick bei einem Windows Host Rechner:
Ansible direkt in Vagrant Box installieren und aufrufen,
Oder per Ansible Local Provisioner

```
config.vm.provision "shell" do |sh|
    sh.path = "windows.sh"
    sh.args = "playbook.yml inventory"
  end
```