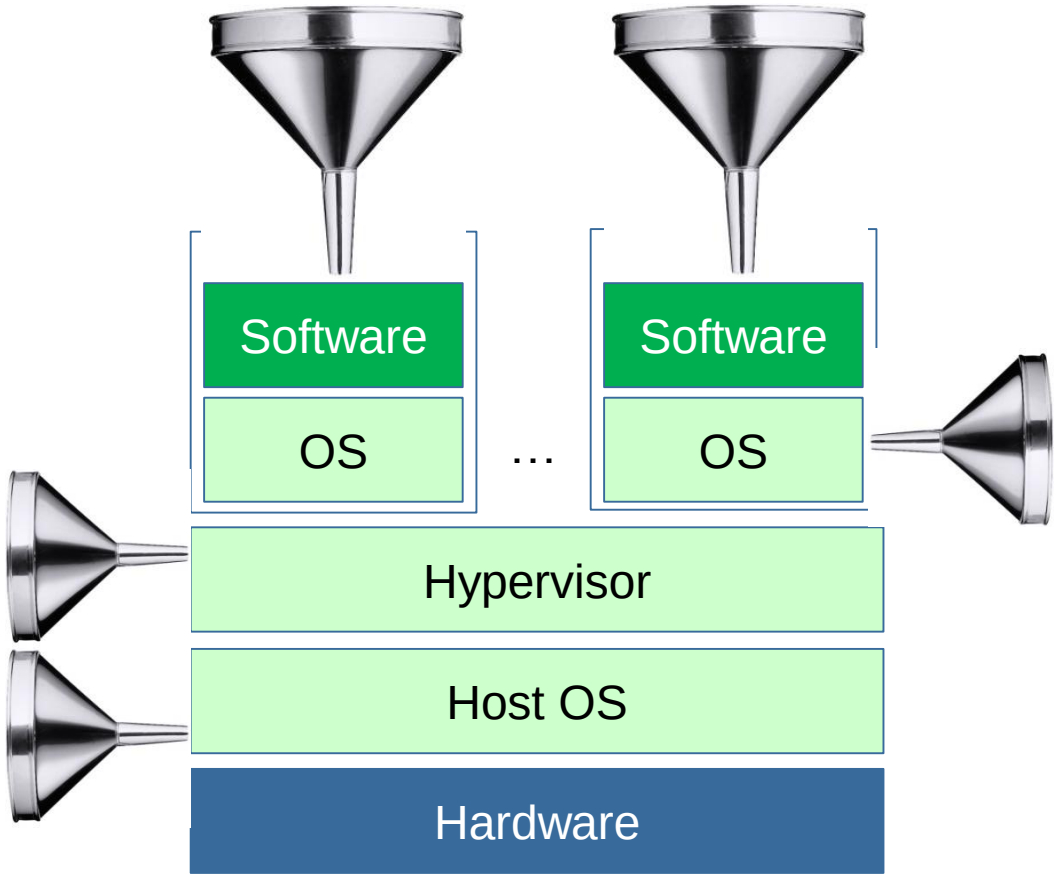
QAWARE
SOFTWARE ENGINEERING

Cloud Computing

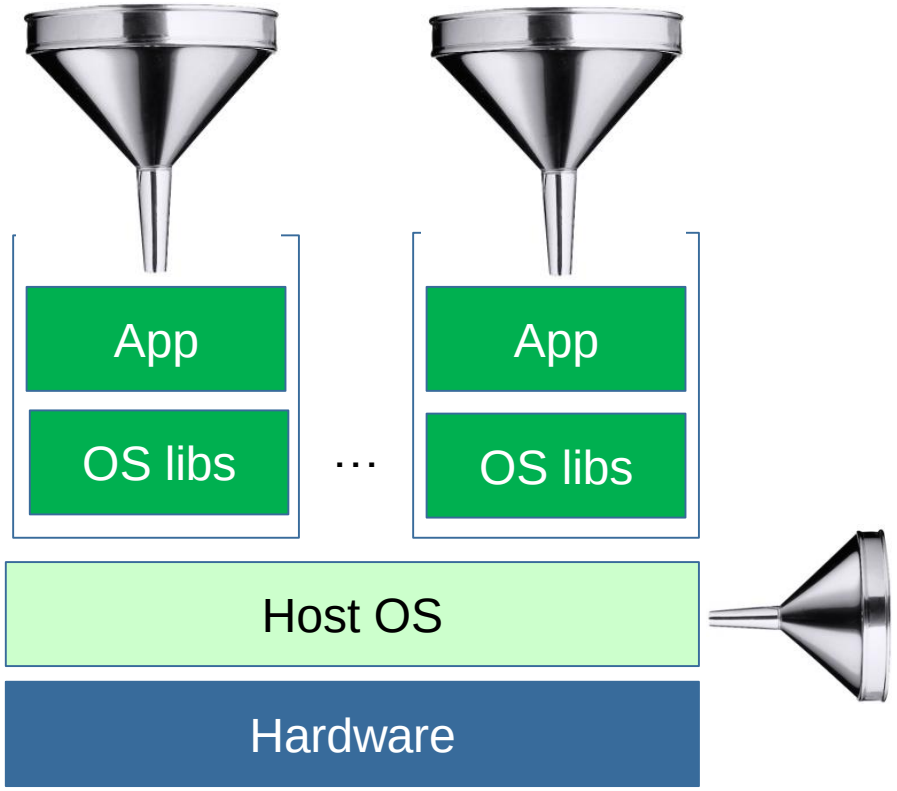
Kapitel 4: Provisionierung

Dr. Josef Adersberger

Wie kommt Software in die Boxen? Provisionierung!



Hardware-Virtualisierung



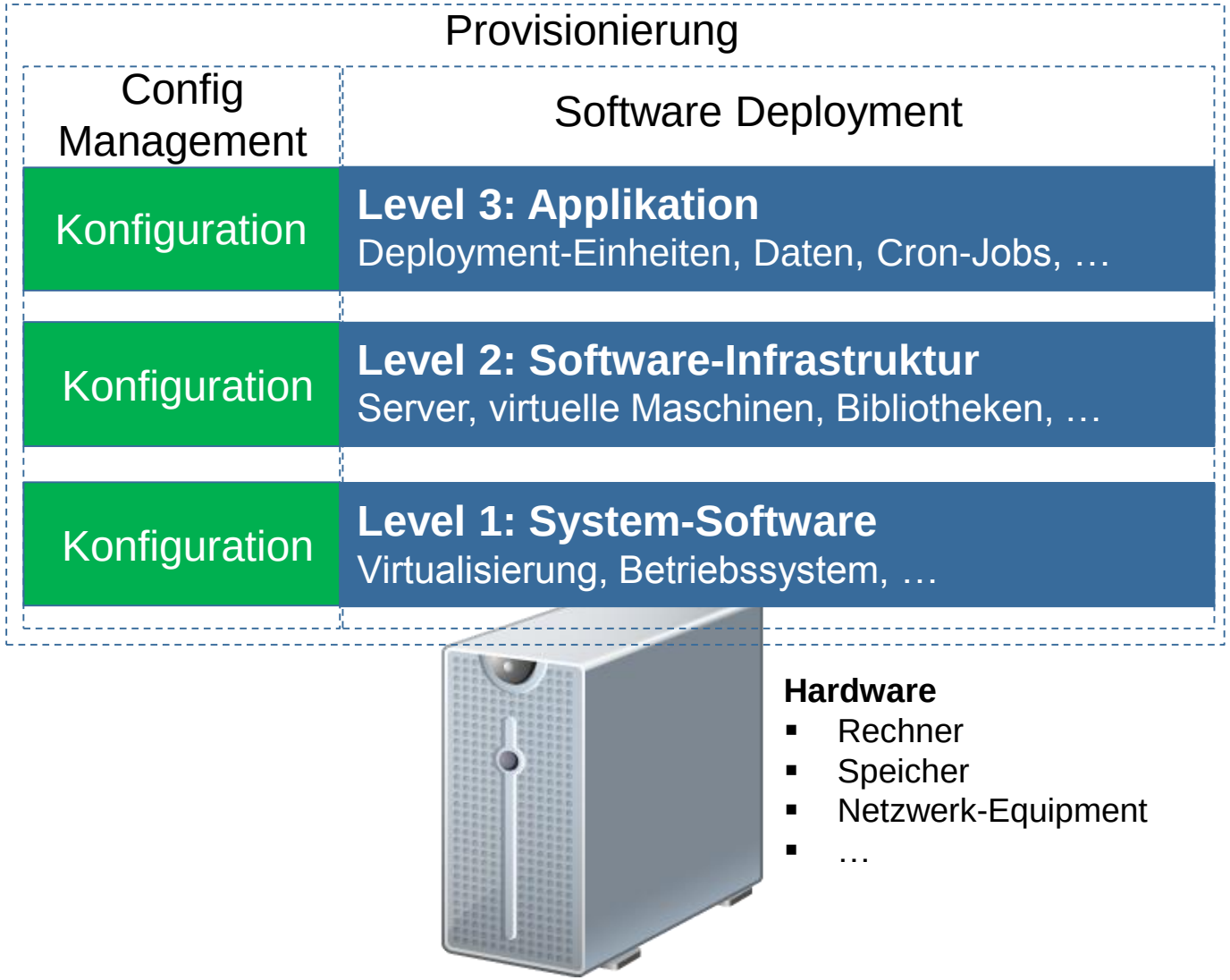
Betriebssystem-Virtualisierung

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

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

Provisierung erfolgt auf drei verschiedenen Ebenen und in vier Stufen.



Laufende Software!

4 **Application Provisioning**

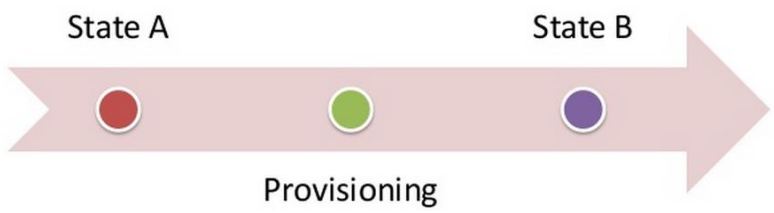
3 **Server Provisioning**
Bereitstellung der Laufzeitumgebung für die Applikation.

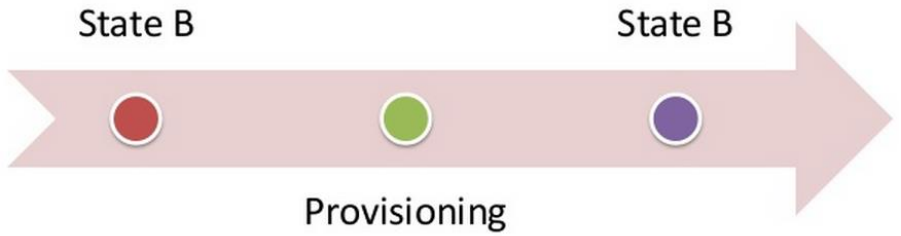
2 **Bootstrapping**
Bereitstellung der Betriebsumgebung für die Software-Infrastruktur.

1 **Bare Metal Provisioning**
Intialisierung 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“

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

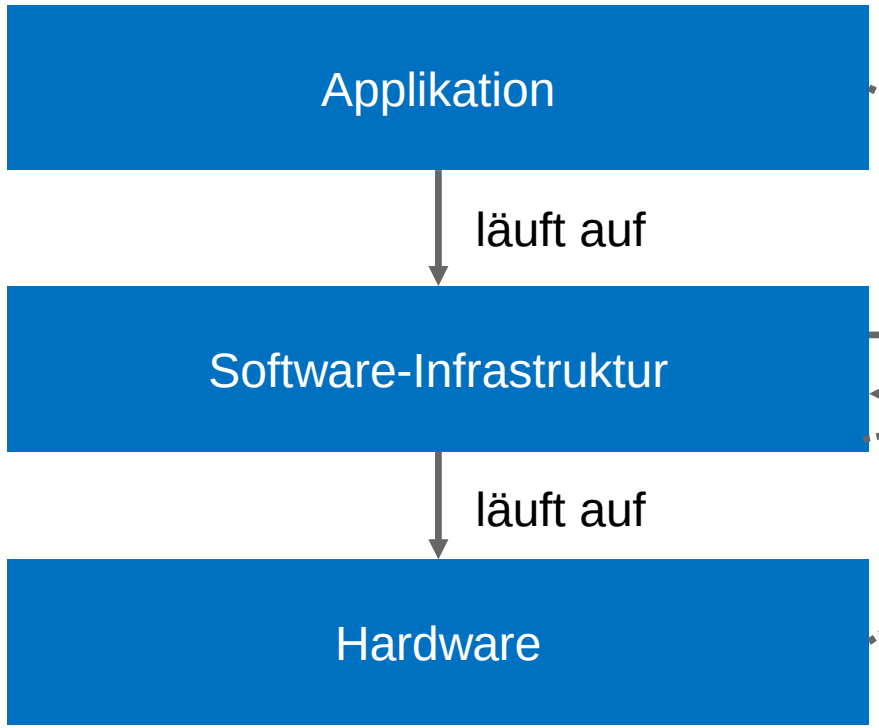
Eine Übersicht gängiger Provisionierungswerkzeuge.

Imperativ **Deskriptiv**

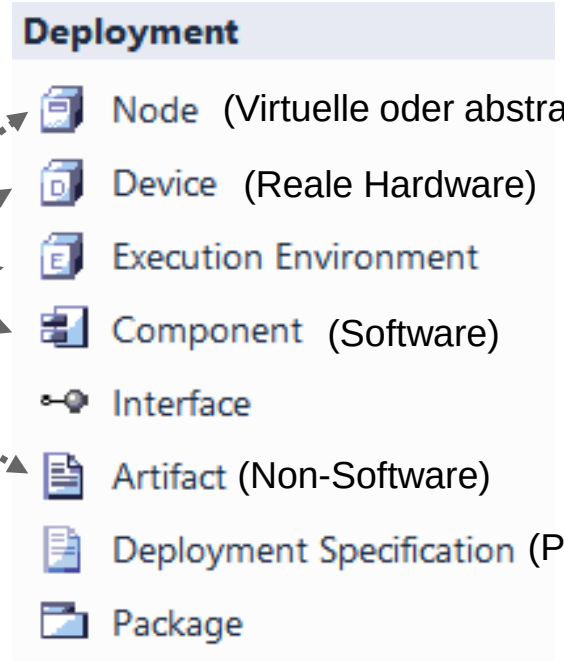
Shell Scripting — Shell Abstraktion — Zustandsautomaten

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

Die Bestandteile einer Ausführungssicht auf Systeme

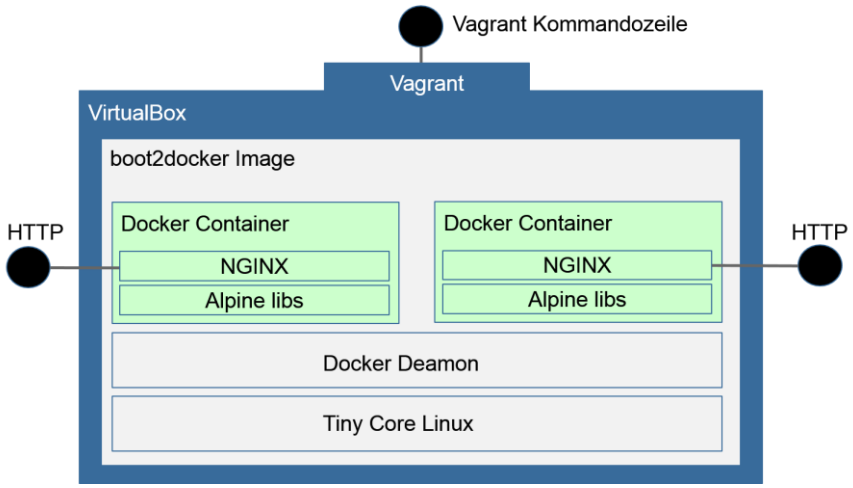


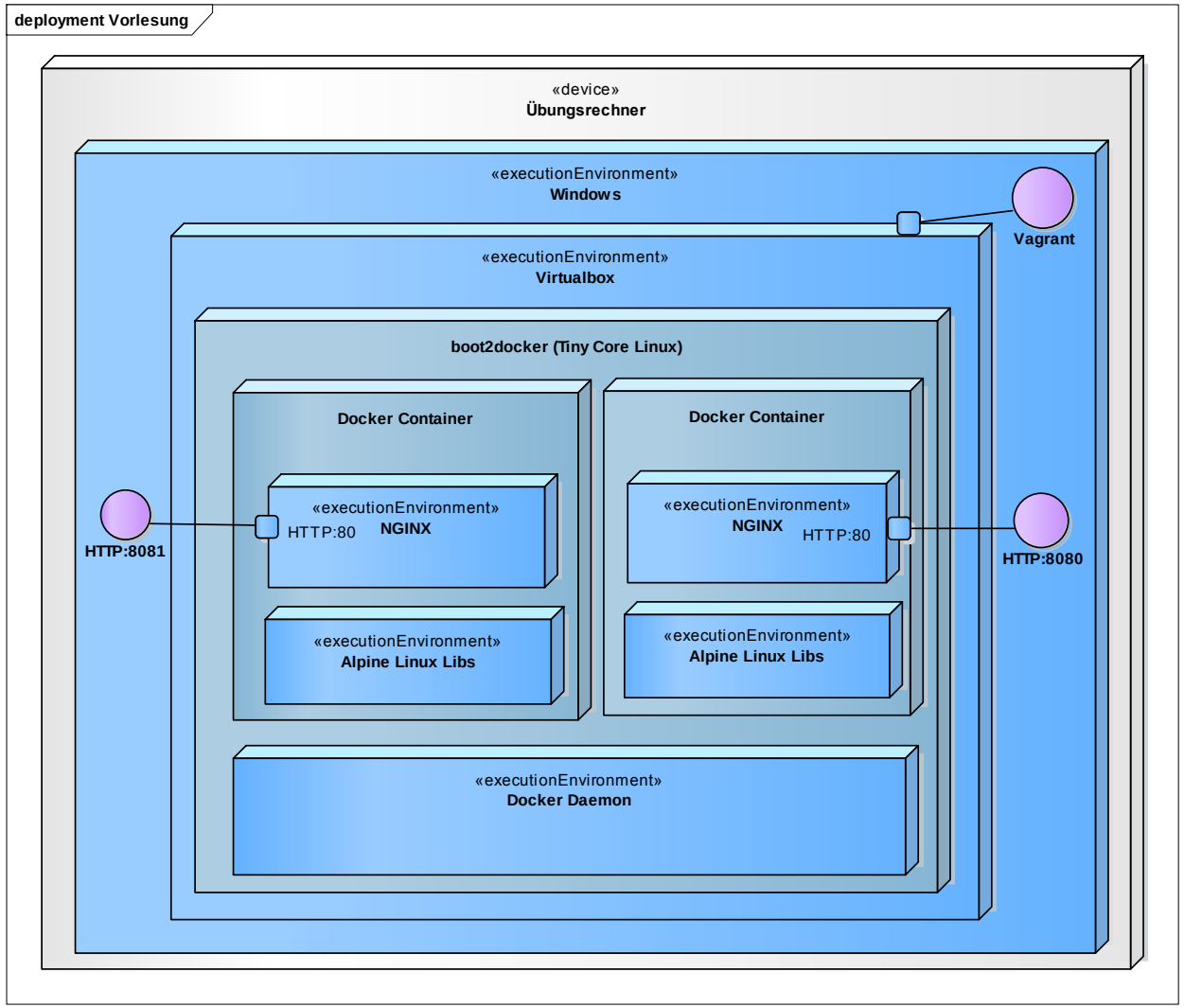
Elemente UML Deployment Modell



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

Unser Übungsbeispiel vom letzten Mal als UML Deployment Modell.





Docker Files

Provisionierung mit Docker

Deployment-Ebenen

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

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

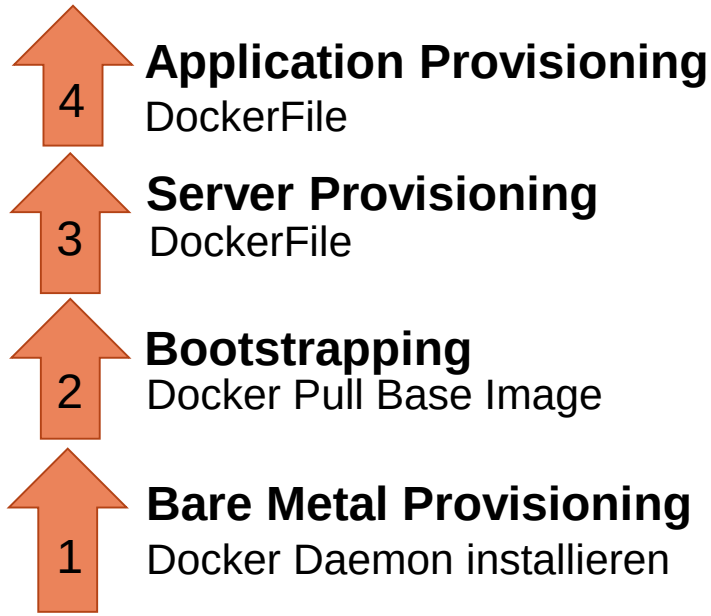
Level 1: System-Software
Virtualisierung, Betriebssystem, …

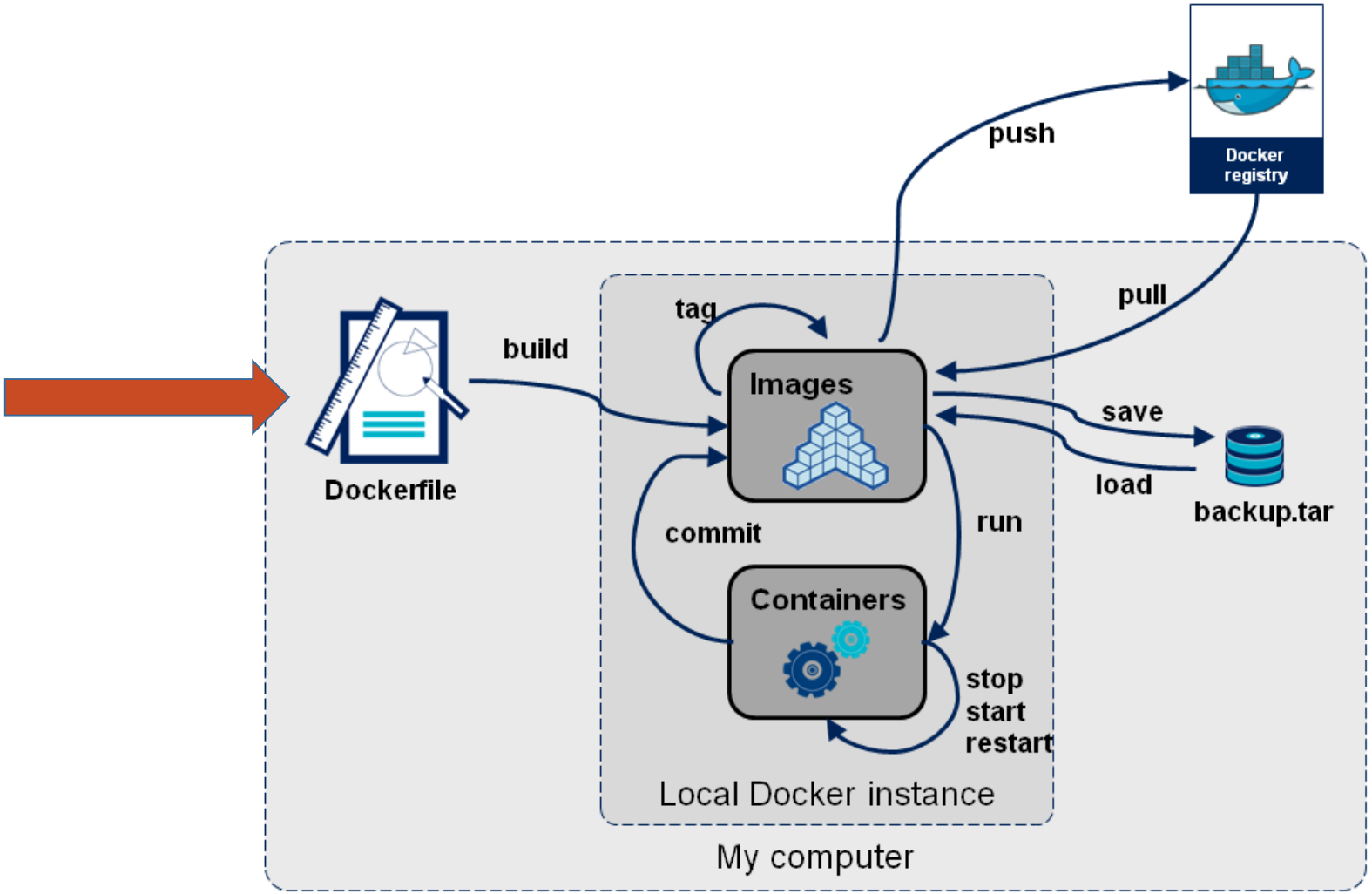
Docker-Image-Kette

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

Server Image
(z.B. NGINX)

Base Image
(z.B. Ubuntu)




Docker registry
push
pull
tag
build
Images
save
load
backup.tar
Dockerfile
commit
run
Containers
stop
start
restart
Local Docker instance
My computer

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>`

Die Syntax

```
FROM ubuntu
MAINTAINER Josef Adersberger, jad@qaware.de
RUN apt-get update
RUN apt-get --assume-yes install nginx
ADD https://provisoning.server.de/nginx.conf /etc/nginx/nginx.conf
ENTRYPOINT nginx
EXPOSE 80
```

Element	Bedeutung
FROM <image-name>	Das Basis-Image setzen
MAINTAINER <autor>	Den Autor definieren (Name + Kontakt)
RUN <kommando>	Ein Shell-Kommando ausführen und das Ergebnis committen
ADD <src> <dest>	Eine Datei in den Container kopieren. <src> kann eine URL sein. Ist <src> eine TAR-Datei, dann wird sie automatisch entpackt.
VOLUME <container-dir> <host-dir>	Stellt ein Verzeichnis im Host im Container zur Verfügung.
ENV <key> <value>	Eine Umgebungsvariable setzen
ENTRYPOINT <kommando>	Das Kommando, das ausgeführt werden soll, sobald der Container startet
WORKDIR <dir>	Setzt für alle Kommandos das Arbeitsverzeichnis
EXPOSE <port>	Port freigeben (Container lauscht auf dem Port)
USER <name>	Den Benutzer im Container setzen

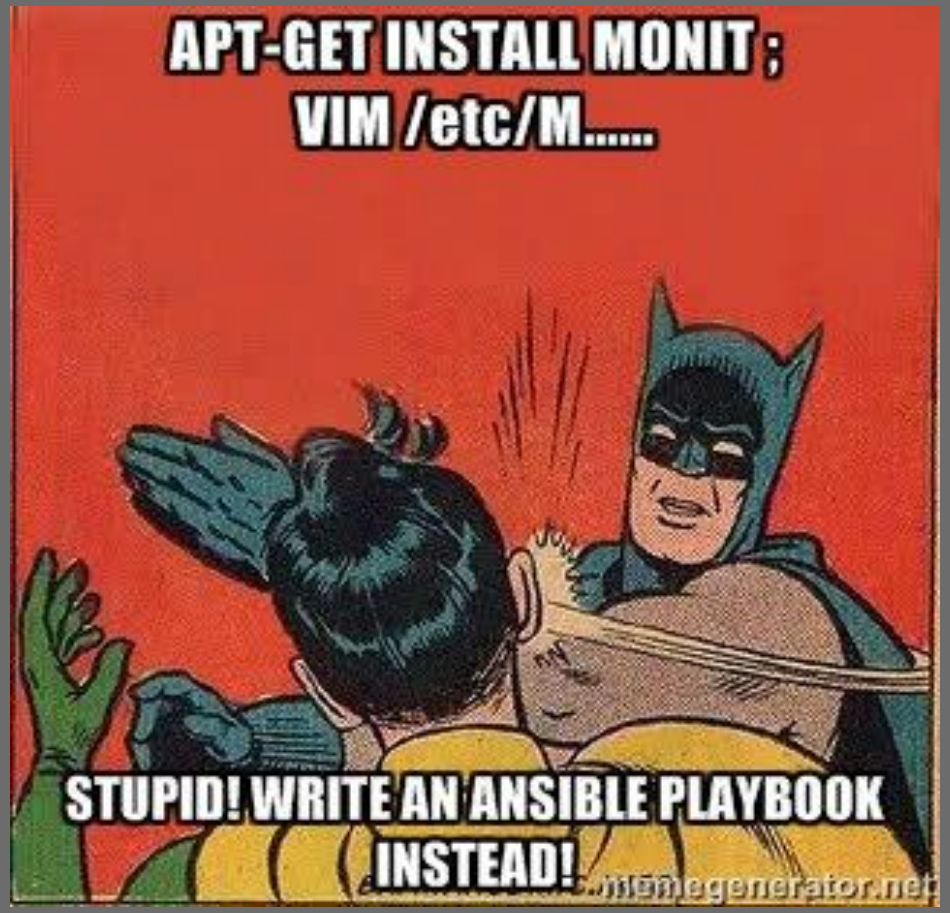
Siehe: http://docs.docker.com/engine/reference/builder/

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


```
gem install serverspec
serverspec-init
```

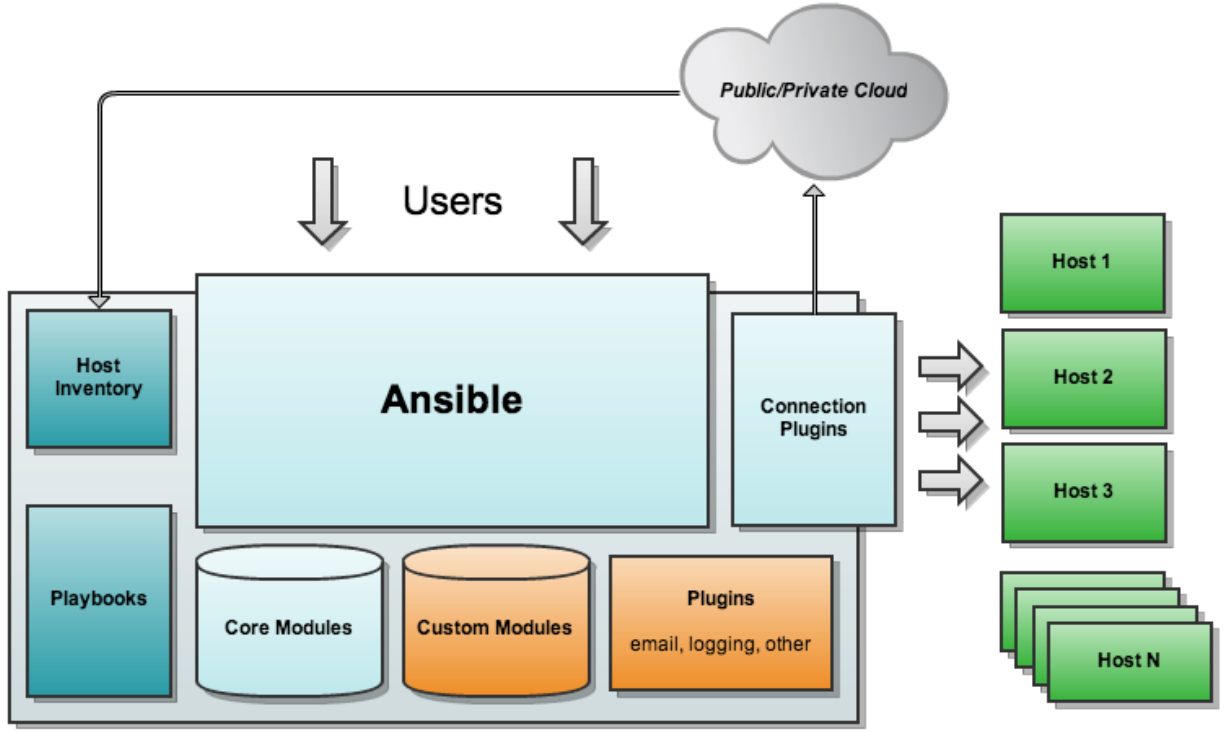
```
require 'spec_helper'

describe "nginx" do
        describe "check running" do
                describe process('nginx') do
                        it { should be_running }
                end
                describe port(80) do
                        it { should be_listening }
                end
        end
        describe file('/etc/logrotate.d/nginx') do
                it { should be_file }
                it { should contain "rotate 14" }
        end
end
```

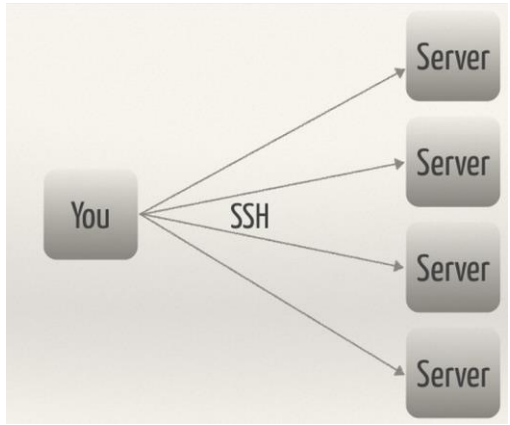
Ausblick: Ansible



Ansible

- Open-Source-Provisionierungswerkzeug
- Kommerzielles Unternehmen steht hinter Ansible
- Ausgelegt auf die Provisionierung großer heterogener IT-Landschaften
- Entwickelt in der Sprache Python
- Pull-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)



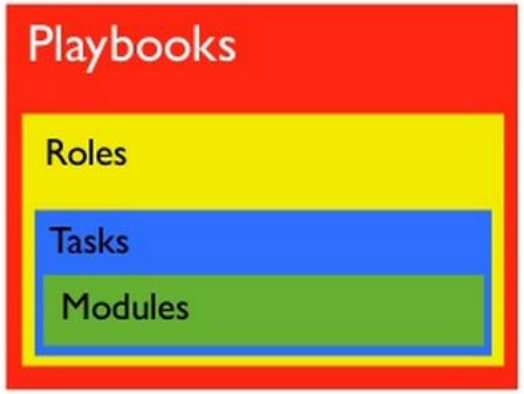


Die wichtigsten zu erstellenden Dateien bei einer Provisionierung mit Ansible.

Playbook (YAML-Syntax)
Provisionierungs-Skript.

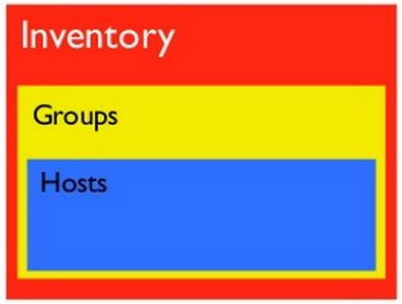
```
- hosts: all
  tasks:
  - yum: pkg=httpd state=installed
```

- 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 gesteuert über die Kommandozeile

- 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 aufrufen.

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