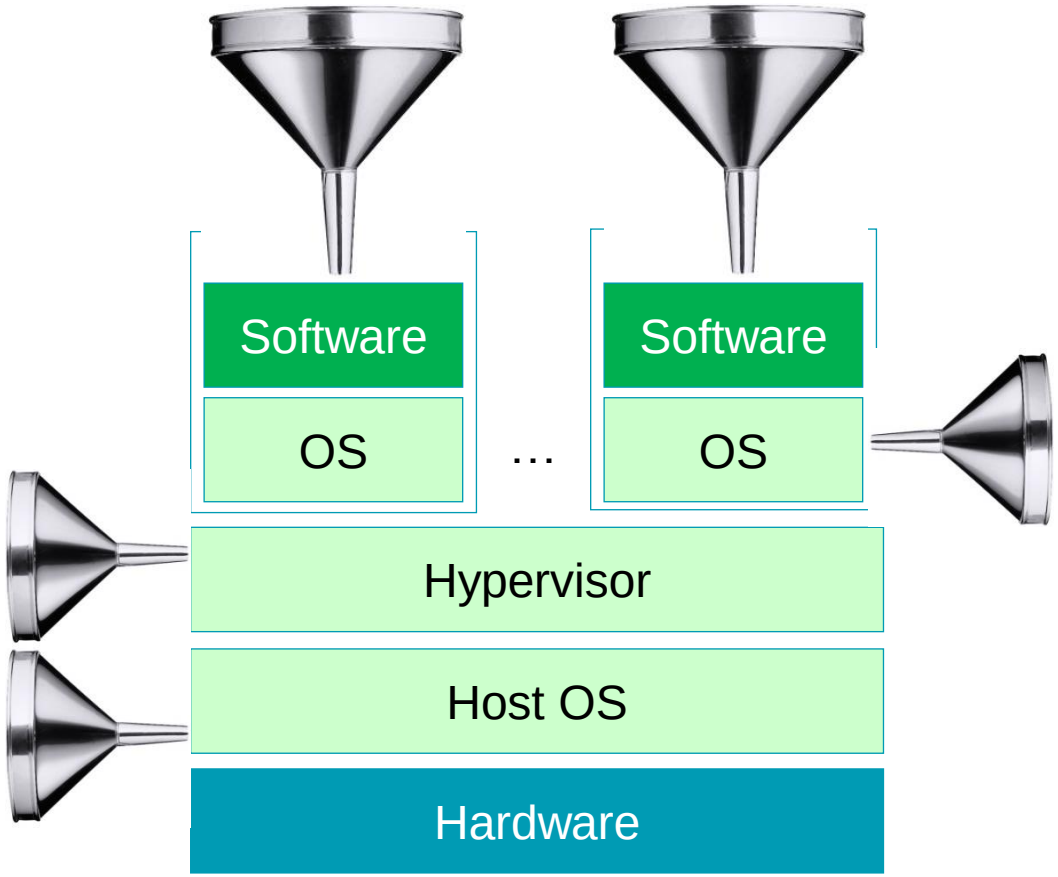
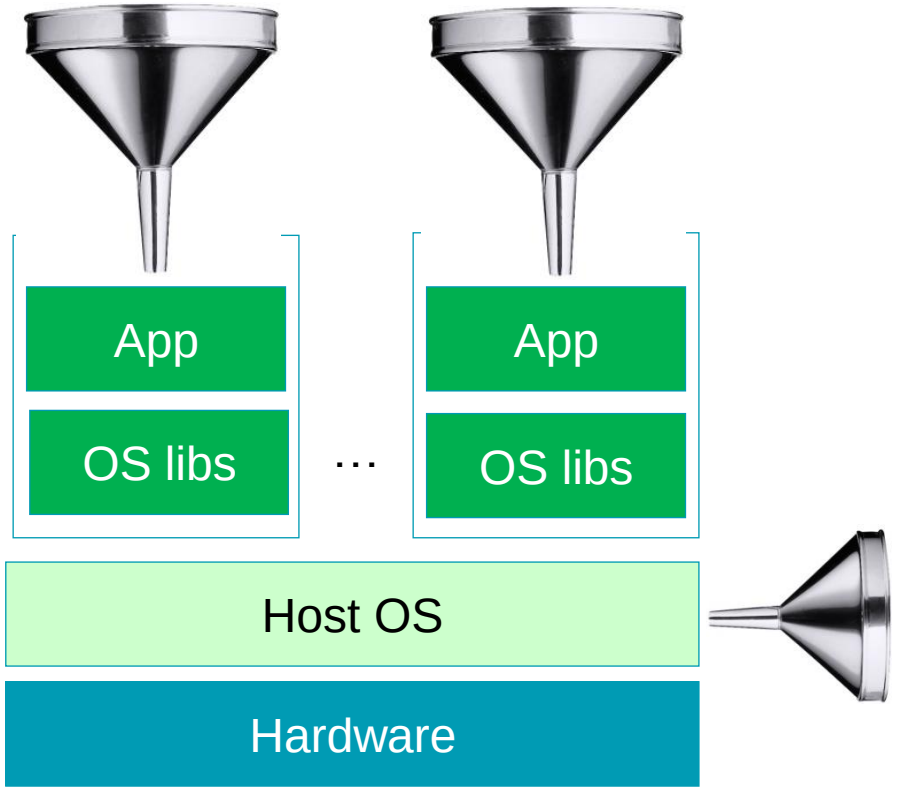
Provisionierung

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

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

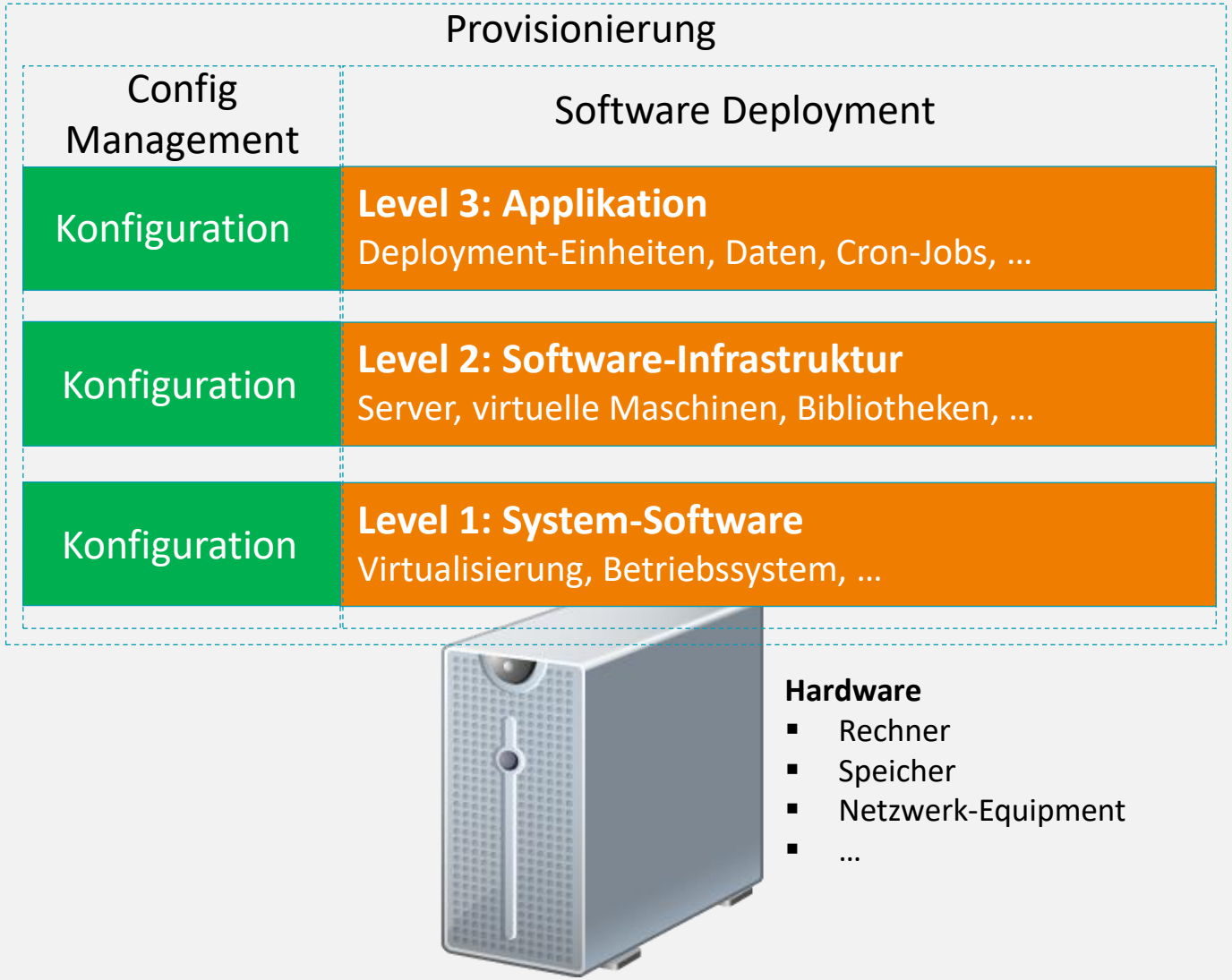
Virtualisierung in der Cloud (seit 2010)

- Automatisches Bereitstellen von vorgefertigten virtuellen Maschinen und Containern
- Manuelle Installation der Infrastruktur-Software nur 1x im Clone-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**

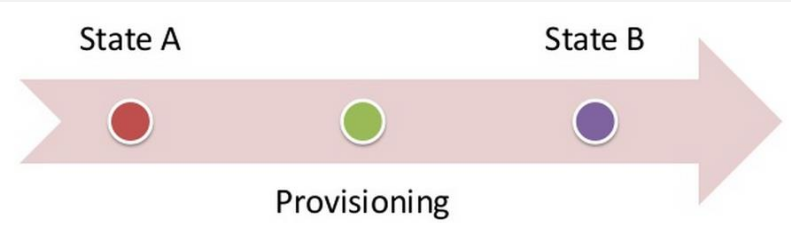
Server Provisioning
3 Bereitstellung der notwendigen Software-Infrastruktur für die Applikation.

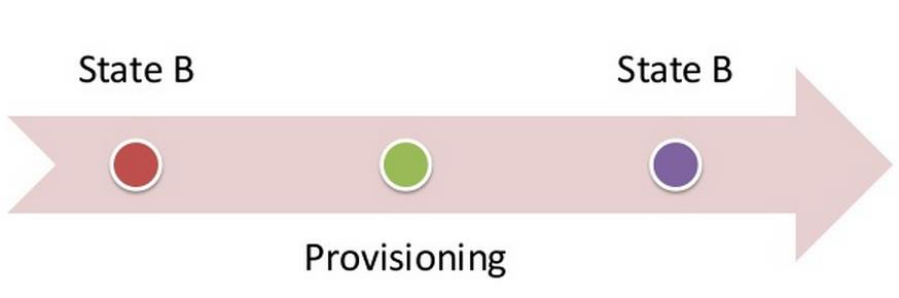
Bootstrapping
2 Bereitstellung der Betriebsumgebung für die Software-Infrastruktur.

Bare Metal Provisioning
1 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

Beliebiger Zustand

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

Ziel-Zustand

New Style

„Immutable Infrastructure / Phoenix Systems“

Basis-Zustand

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

Ziel-Zustand

Immutable Infrastructure

An *immutable infrastructure* is another infrastructure paradigm in which servers are **never modified** after they're deployed. If something needs to be updated, fixed, or modified in any way, **new servers built from a common image with the appropriate changes** are provisioned to replace the old ones. After they're validated, they're put into use and **the old ones are decommissioned**.

The benefits of an immutable infrastructure include **more consistency and reliability** in your infrastructure and a **simpler, more predictable deployment process**. It mitigates or entirely **prevents** issues that are common in mutable infrastructures, like **configuration drift and snowflake servers**. However, using it efficiently often includes comprehensive deployment automation, fast server provisioning in a cloud computing environment, and solutions for handling stateful or ephemeral data like logs.

Quelle: https://www.digitalocean.com/community/tutorials/what-is-immutable-infrastructure

Eine Übersicht gängiger Provisionierungswerkzeuge.

Imperativ

Shell Scripting

Shell Abstraktion

Deskriptiv

Zustandsautomaten

Dockerfiles und Docker Compose / Docker Stack

Provisionierung mit DockerFile und Docker Compose

Deployment-Ebenen

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

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

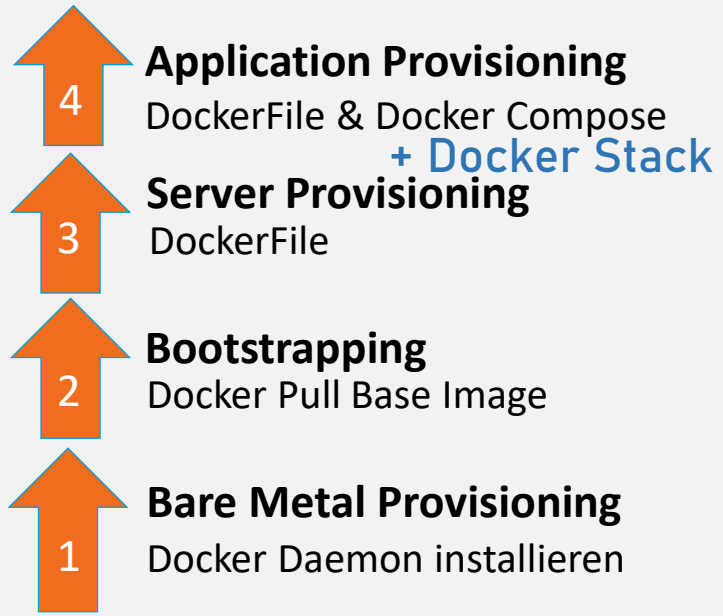
Level 1: System-Software
Virtualisierung, Betriebssystem, …

Docker-Image-Kette

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

Server Image
(z.B. NGINX)

Base Image
(z.B. Ubuntu)



Nutzung von Docker Compose für Multi-Container Apps.

docker-compose.yml

```
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

```
$ docker-compose up -d --build

$ docker-compose stop

$ docker-compose rm -s -f
```

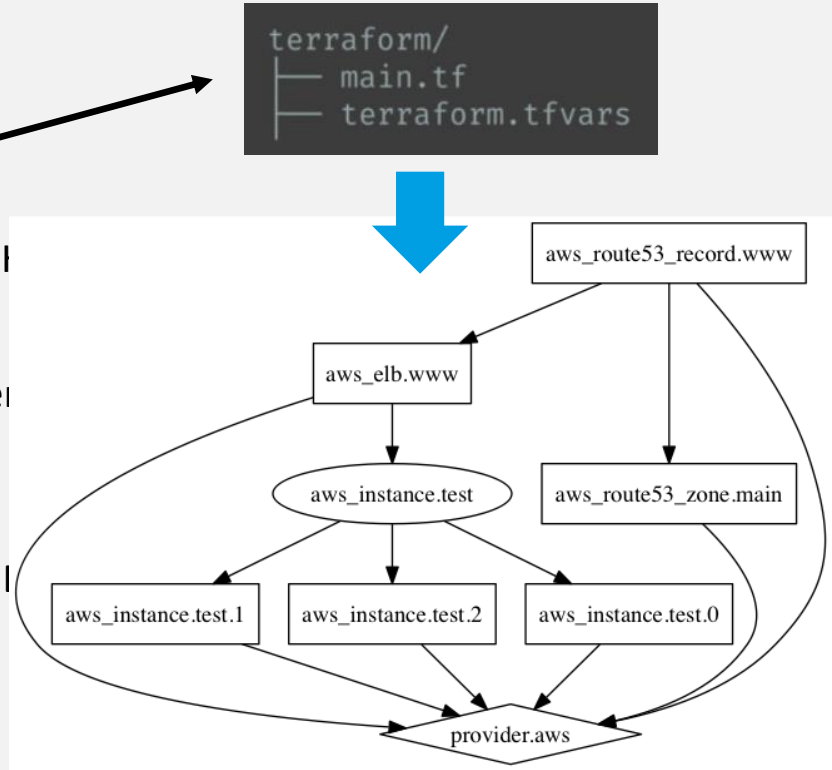
HashiCorp

Terraform

Write, Plan, and Create Infrastructure as Code

Terraform

- Entwickelt von HashiCorp
- Open Source, in Go geschrieben
- Kommandozeilenwerkzeug
- Gedacht um eine Cloud-Infrastruktur zu provisionieren (VMs, VPN, Loadbalancer, Cloud-Storage, etc)
- Nicht zur Installation der Software auf den einzelnen VMs
- Direkte Anbindung vieler Cloud Provider (AWS, Azure, OTC, …)
- Deklarative Programmierung
  - **Write**: Beschreibung Zielzustand über eine domänenspezifische Sprache I (HashiCorp Configuration Language)
  - **Plan** (`terraform plan`): Ist-Zustand ermitteln. Notwendige Änderunge planen (entsprechend Abhängigkeiten geordnet und parallelisiert, Unterbrechungen möglichst minimal)
  - **Apply** (`terraform apply`): Idempotente Herstellung des Zielzustands. Zustand (*.tfstate* Datei) wird dabei lokal oder in einem Remote Store (S3, HTTP, …) gespeichert

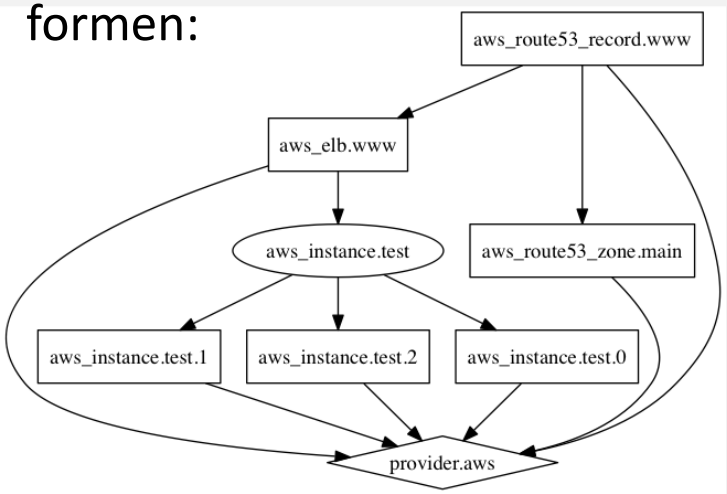


Die Kern-Entitäten eines Terraform-Skripts

Ressource: Provisionierte Komponente der Infrastruktur

```
resource "aws_instance" "web" {
  ami           = "ami-408c7f28"
  instance_type = "t1.micro"
}
```

… haben Abhängigkeiten zueinander, die einen DAG formen:



Provider: Integration der zu provisionierenden Infrastruktur

Alicloud	Archive	AWS
Azure	Bitbucket	CenturyLinkCloud
Chef	Circonus	Cloudflare
CloudScale.ch	CloudStack	Cobbler
Consul	Datadog	DigitalOcean
DNS	DNSMadeEasy	DNSimple
Docker	Dyn	External
Fastly	GitHub	Gitlab

```
provider "aws" {
  access_key = "${var.aws_access_key}"
  secret_key = "${var.aws_secret_key}"
  region     = "us-east-1"
}
```

OpsGenie	Oracle Public Cloud	Oracle Cloud Platform
OVH	Packet	PagerDuty
Palo Alto Networks	PostgreSQL	PowerDNS
ProfitBricks	RabbitMQ	Rancher
Random	Rundeck	Scaleway
SoftLayer	StatusCake	Spotinst
Template	Terraform	Terraform Enterprise
TLS	Triton	UltraDNS
Vault	VMware vCloud Director	VMware NSX-T

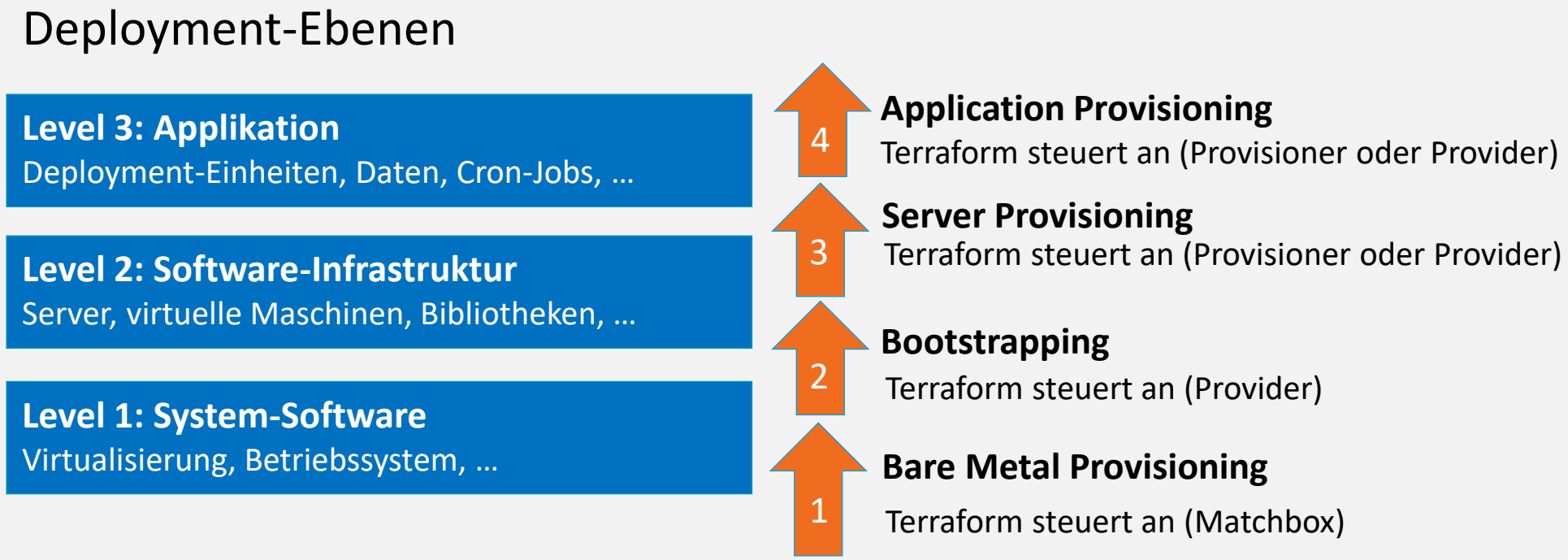
Provisioner: Ausführung von Änderungen auf Ressourcen.

```
resource "aws_instance" "web" {
  # ...

  provisioner "local-exec" {
    command = "echo ${self.private_ip} > file.txt"
  }
}
```

Beispiel

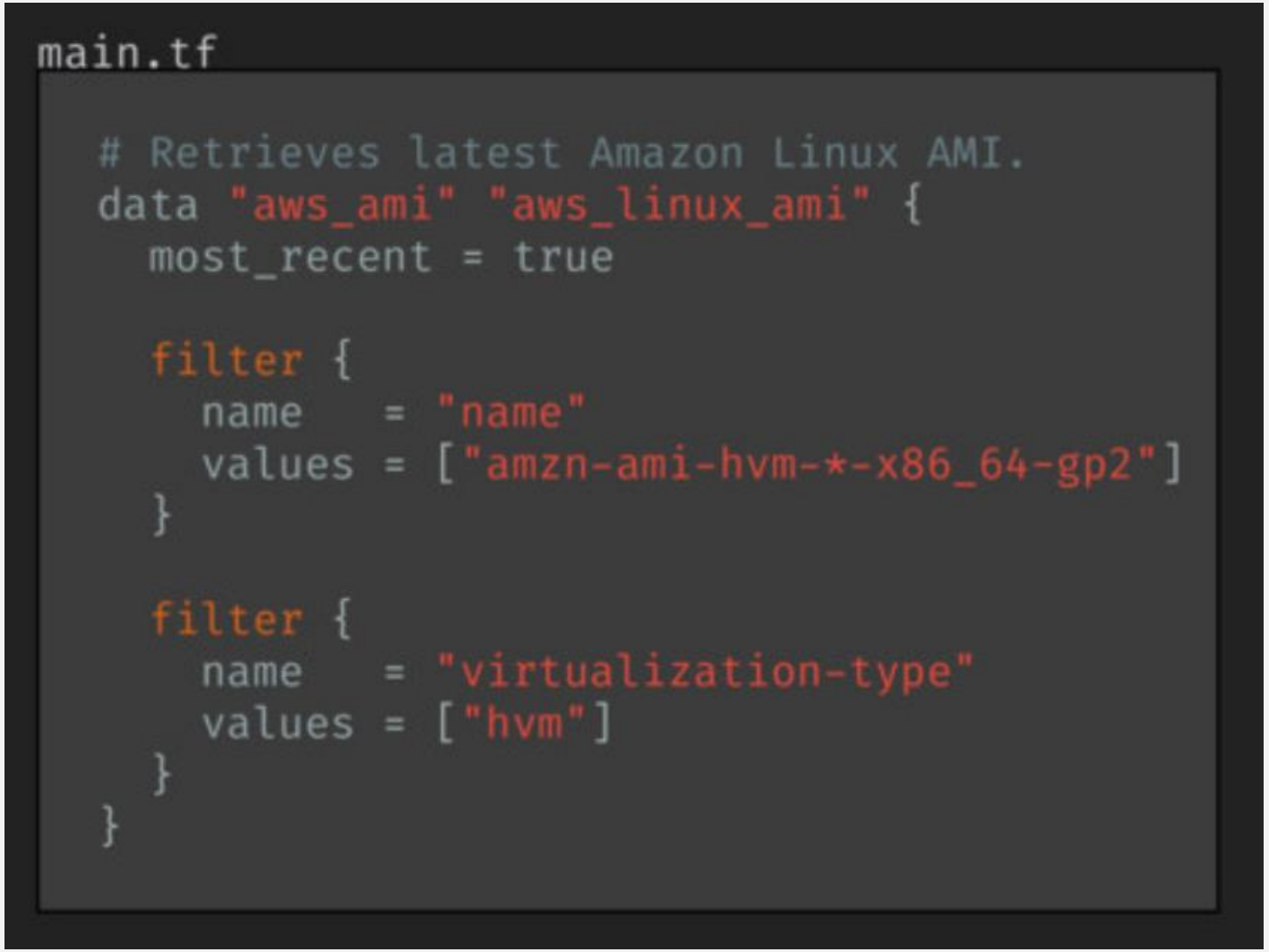

```
# New resource for the S3 bucket our application will use.
resource "aws_s3_bucket" "example" {
  # NOTE: S3 bucket names must be unique across _all_ AWS accounts, so
  # this name must be changed before applying this example to avoid naming
  # conflicts.
  bucket = "terraform-getting-started-guide"
  acl    = "private"
}

# Change the aws_instance we declared earlier to now include "depends_on"
resource "aws_instance" "example" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"

  # Tells Terraform that this EC2 instance must be created only after the
  # S3 bucket has been created.
  depends_on = ["aws_s3_bucket.example"]
}
```

Provisionierung mit Terraform

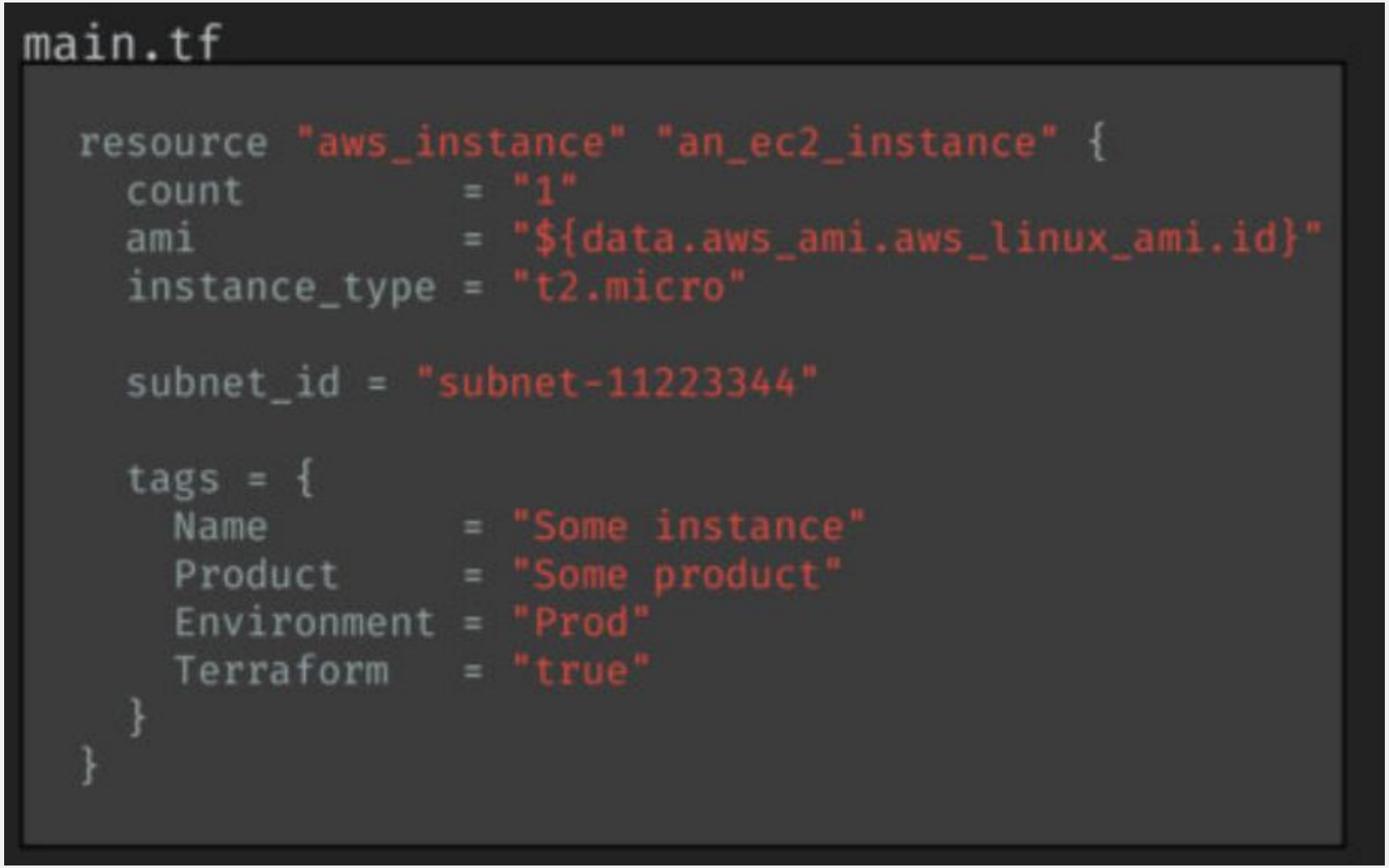


Beispiel: Provider

Beispiel: Data



Beispiel: Resources



QA|WARE

Beispiel: Output

QA|WARE

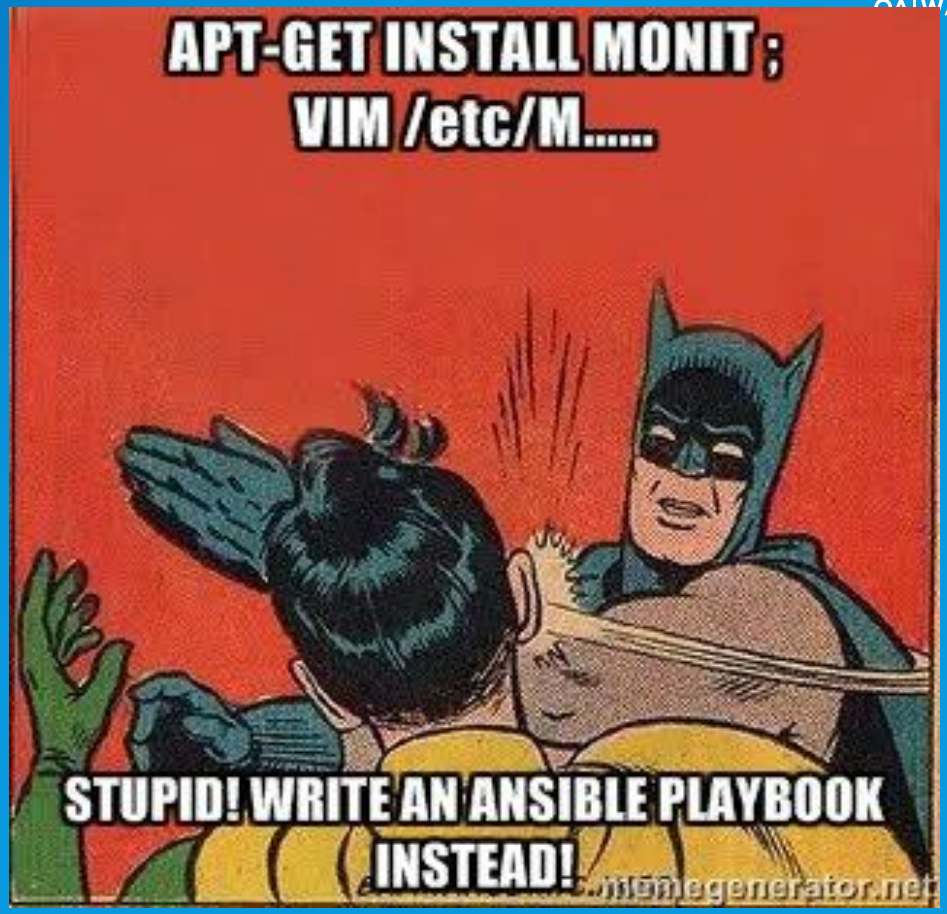
main.tf

```
resource "aws_instance" "an_ec2_instance" {
  count         = "1"
  ami           = "${data.aws_ami.aws_linux_ami.id}"
  instance_type = "t2.micro"
}

output "instance_ids" {
  value = "${aws_instance.ec2.id}"
}
```

Beispiel: Struktur

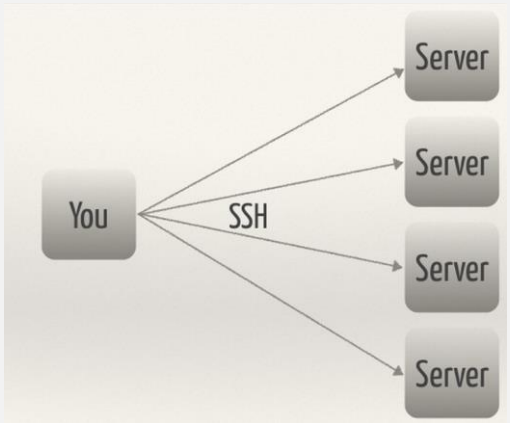
QA|WARE

Ansible

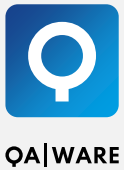

APT-GET INSTALL MONIT ;
VIM /etc/M.....
STUPID! WRITE AN ANSIBLE PLAYBOOK
INSTEAD!
memegenerator.net

Ansible

- Open-Source-Provisionierungswerkzeug von Red Hat
- 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
- Variante ansible-container zur Provisionierung von Containern
- 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)



Provisionierung mit Ansible



Deployment-Ebenen

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

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

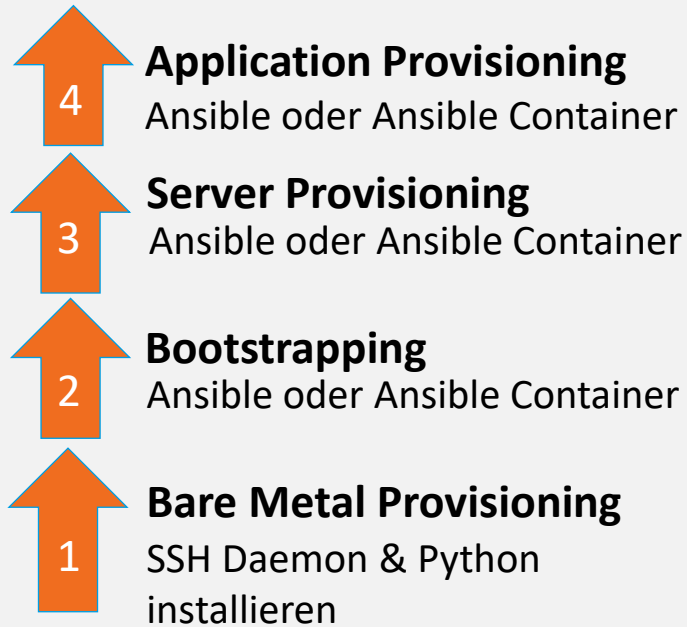
Level 1: System-Software
Virtualisierung, Betriebssystem, …

Docker-Image- oder VM-Kette

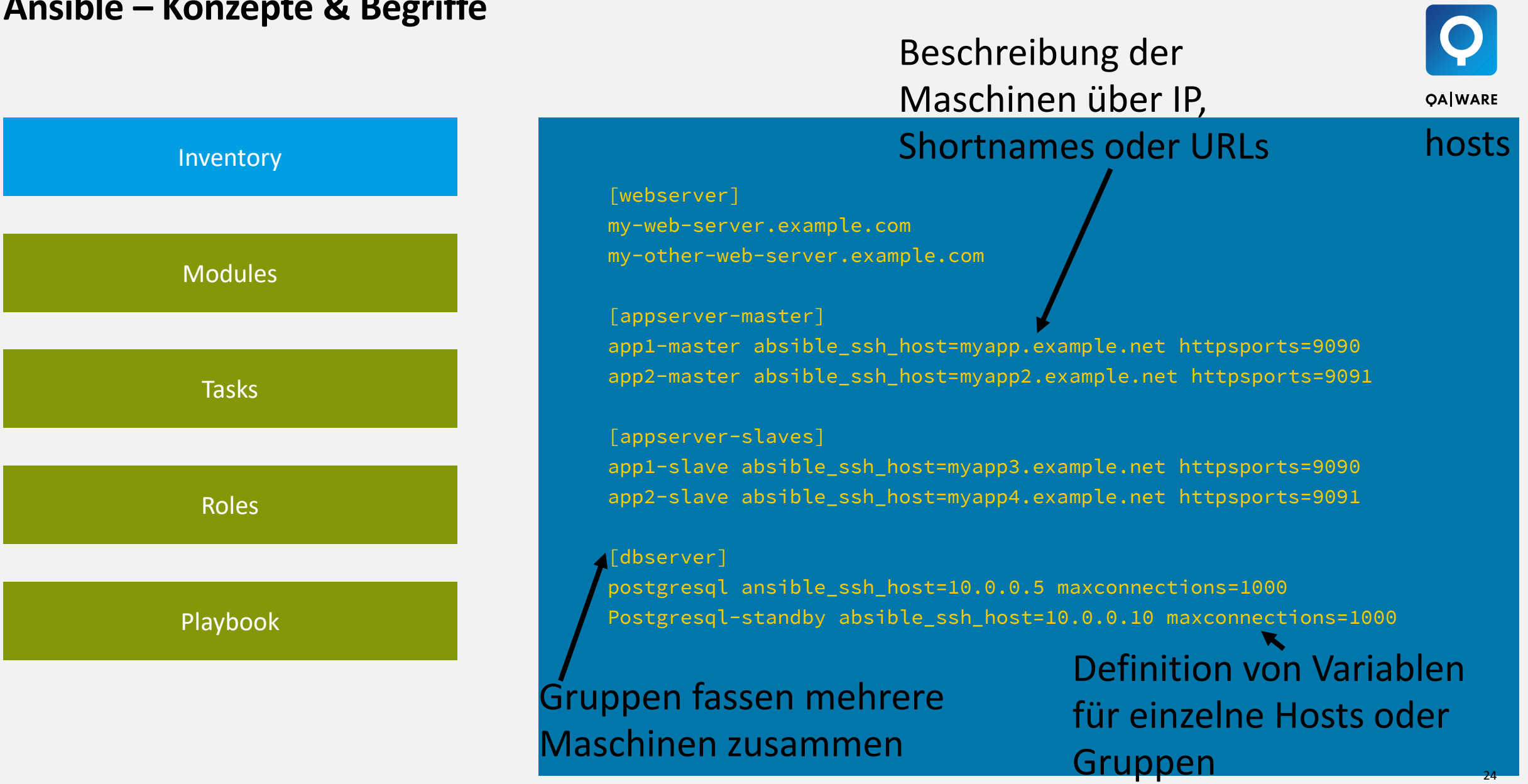
Applikations-Image
(z.B. www.qaware.de)

Server Image
(z.B. NGINX)

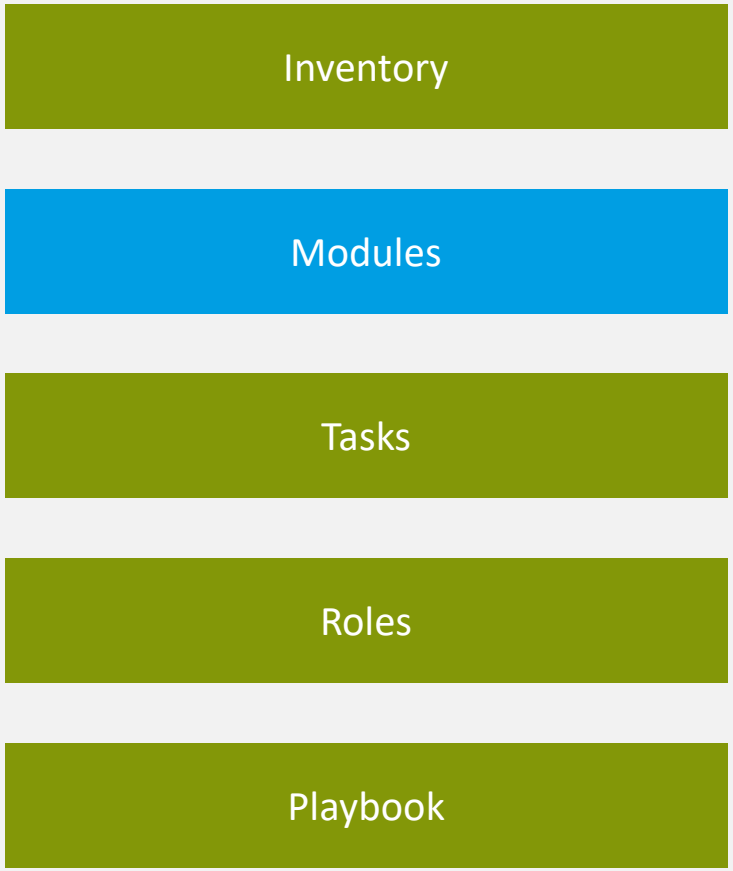
Base Image
(z.B. Ubuntu)



Ansible – Konzepte & Begriffe



Ansible – Konzepte & Begriffe



- Modules erlauben die Interaktion über Ansible
- Man kann
  - selbst Modules schreiben
  - offizielle Ansible Modules nutzen (Core), diese sind schon Teil von Ansible
  - Modules aus der Community nutzen (Extras)
- Beispiele:
  - **File handling**: file, copy, template
  - **Remote execution**: command, shell
  - **Package management**: apt, yum

Ansible – Konzepte & Begriffe

Inventory

Modules

Tasks

Roles

Playbook

- Jeder Task beschreibt eine Provisionierungs-Aktion
- Beispiel: Installieren von Paketen über apt
- Dabei ruft der Task ein Modul auf, das die aktuelle Aufgabe umsetzt.

- Ausführung über Ad Hoc Commands

```
ansible -m <module>
-a <arguments> <server>
```

Ansible – Konzepte & Begriffe

- Inventory
- Modules
- Tasks
- Roles
- Playbook

- Rollen bündeln Tasks zu einer Aufgabe, z.B. Webserver oder SSHD

```
# roles/webserver/tasks/main.yml
- name:
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'
- import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'

# roles/ webserver/tasks/redhat.yml
- yum:
    name: "httpd"
    state: present

# roles/ webserver/tasks/debian.yml
- apt:
    name: "apache2"
    state: present
```

--- - hosts: webservers roles: - common - webservers

Ansible – Konzepte & Begriffe

Inventory

Modules

Tasks

Roles

Playbook

- Playbooks als Basis für Config Management & Orchestrierung
- Legen fest, welche Roles oder Tasks wann auf welcher Maschine ausgeführt werden
- Erlauben synchrone und asynchrone Ausführung

```
- hosts: webserver
  roles:
    - common
    - webserver
[...]
```

Ansible – Konzepte & Begriffe

Inventory

Modules

Tasks

Roles

Playbook

- Alternativ können tasks auch direkt im Playbook verwendet werden:

```
- hosts: webserver
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
[...]
```

Die wichtigsten zu erstellenden Dateien bei einer Provisionierung mit Ansible.

QA|WARE

Playbook (YAML-Syntax)
Provisionierungs-Skript.

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

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

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> -i <inventory-file>`
    `-m <modul> -a „<argumente>“ –f <parallelism>`
  - Beispiele:
    - `ansible all –m ping`
    - `ansible all –a „/bin/echo hello“`
    - `ansible web –m apt –a „name=nginx state=installed“`
    - `ansible web –m service –a „name=nginx state=started“`
    - `ansible all -a "/sbin/reboot" -f 10`
- Playbooks ausführen
  - `ansible-playbook <playbook.yaml>`