LU10c - Provisioning mit Ansible (idempotent konfigurieren)

Ziel: Du kannst **Provisioning** verstehen und anwenden: Du konfigurierst VMs automatisiert mit **Ansible** (idempotent), und du kannst Ansible-Provisioning sinnvoll von **Shell-Skripten** und **Dockerfiles** abgrenzen.

Übersicht

Thema	Kernaussage
Provisioning	„Maschine nach dem Boot automatisch einrichten“ (Pakete, Konfig, Services, Users, Files).
Shell Provisioner	Schnell, aber oft „fragil“ (nicht idempotent, schwer wartbar).
Ansible	Konfiguration als Code: deklarativ, **idempotent**, gut strukturierbar (Rollen/Tasks).
Idempotenz	„Mehrfach ausführen → gleicher Zustand, ohne Chaos“.
Handler	„Nur bei Änderung Service neu starten/reload“.
Playbook	Sammlung von Tasks, die auf Hosts angewendet werden.

Warum Provisioning?

Ohne Provisioning passiert das hier:

- Jede:r installiert manuell Pakete
- Jede:r vergisst etwas
- Jede:r hat andere Versionen/Einstellungen
- Debugging wird unfair („bei mir geht’s“)

Mit Provisioning:

- Einmal definieren → immer reproduzierbar
- Gleiche Ausgangslage für alle
- Ideal für Multi-Node Labs

Merke: Vagrant erstellt VMs. Provisioning macht daraus „fertige Nodes“ für dein verteiltes System.

Shell vs. Ansible (kurzer Vergleich)

Kriterium	Shell-Skript	Ansible
Lesbarkeit	ok, aber schnell unübersichtlich	sehr gut (Tasks/Module)
Wiederholbarkeit	oft problematisch	**idempotent**
Fehlerhandling	selber bauen	eingebaut (Return-Codes, Changed-Status)
Strukturierung	eher „Script-Spaghetti“	Rollen, Variablen, Templates
Lerntransfer	Script-Skills	DevOps/IaC-Transfer sehr hoch

Kernkonzepte in Ansible

Inventory (Hosts-Liste)

Welche Maschinen gibt es und wie heissen sie?

Beispiel (klassisch):

```
[db]
192.168.56.11

[api]
192.168.56.12

[proxy]
192.168.56.13
```

Playbook

Ein YAML-Dokument, das beschreibt, was auf welchen Hosts passieren soll.

Task

Ein einzelner Schritt, z.B. „nginx installieren“.

Module

„Bausteine“ für typische Dinge: `apt`, `copy`, `template`, `service`, `user`, `lineinfile`…

Handler

Wird nur ausgelöst, wenn sich etwas ändert (z.B. Config geändert → nginx reload).

Idempotenz ist das Killer-Feature: Ein Playbook kann man „einfach nochmal laufen lassen“, ohne alles kaputt zu machen.

Ansible in Vagrant einbinden

Es gibt zwei gängige Wege:

Variante A: ansible (Host-basiert)

- Ansible läuft auf deinem **Host**
- Vagrant übergibt die VM-Infos an Ansible
- Vorteil: schnell, simpel (wenn Host Ansible hat)
- Nachteil: Host muss Ansible installiert haben (Windows teils mühsam)

Variante B: ansible_local (Guest-basiert)

- Ansible wird in der **VM** installiert und dort ausgeführt
- Vorteil: host-unabhängiger, oft „klassen-tauglicher“
- Nachteil: Playbook-Lauf ist minimal langsamer

In Klassen mit gemischten Betriebssystemen ist **ansible_local** oft robuster.

Beispiel-Projektstruktur

```
projekt/
├─ Vagrantfile
└─ provision/
   ├─ playbook.yml
   ├─ group_vars/
   │  ├─ all.yml
   └─ templates/
      └─ nginx.conf.j2
```

Beispiel: Vagrantfile mit ansible_local

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/jammy64"
```

```
  nodes = {
    "db"    => "192.168.56.11",
    "api"   => "192.168.56.12",
    "proxy" => "192.168.56.13"
  }

  nodes.each do |name, ip|
    config.vm.define name do |node|
      node.vm.hostname = name
      node.vm.network "private_network", ip: ip
    end
  end

  # Ansible läuft in der VM (ansible_local)
  config.vm.provision "ansible_local" do |ansible|
    ansible.playbook = "provision/playbook.yml"
    ansible.install = true
  end
end
```

Beispiel: Playbook (Basis + Rollen light)

Dieses Playbook zeigt typische Patterns:

- Pakete installieren
- /etc/hosts pflegen (damit Namen auflösbar sind)
- nginx als Proxy konfigurieren
- Services starten/aktivieren

```
# provision/playbook.yml
- name: Multi-Node Setup fuer verteilte Systeme
  hosts: all
  become: true

  vars:
    cluster_hosts:
      - { name: "db",    ip: "192.168.56.11" }
      - { name: "api",   ip: "192.168.56.12" }
      - { name: "proxy", ip: "192.168.56.13" }

  tasks:
    - name: Pakete installieren (Basis)
      ansible.builtin.apt:
        name:
          - curl
          - ca-certificates
          - net-tools
```

```
          - python3
          - python3-pip
        update_cache: true
        state: present

    - name: /etc/hosts fuer Cluster-Namen pflegen
      ansible.builtin.lineinfile:
        path: /etc/hosts
        line: "{{ item.ip }} {{ item.name }}"
        state: present
      loop: "{{ cluster_hosts }}"

- name: Proxy Node konfigurieren
  hosts: proxy
  become: true

  tasks:
    - name: nginx installieren
      ansible.builtin.apt:
        name: nginx
        update_cache: true
        state: present

    - name: nginx config deployen
      ansible.builtin.template:
        src: templates/nginx.conf.j2
        dest: /etc/nginx/sites-available/default
      notify: reload nginx

    - name: nginx aktivieren und starten
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true

  handlers:
    - name: reload nginx
      ansible.builtin.service:
        name: nginx
        state: reloaded
```

Beispiel: Nginx Template (Reverse Proxy)

```
# provision/templates/nginx.conf.j2
server {
  listen 80;

  location / {
    proxy_pass http://api:8000;
```

```
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
  }
}
```

Was hier wichtig ist: Der Proxy spricht **api:8000** (Name, nicht IP). Das funktioniert, weil wir via Ansible `/etc/hosts` gesetzt haben. (Später kann man das durch „echte“ Service Discovery ersetzen.)

Brücke zu Dockerfile / Docker-Compose

Konzept	Docker-Welt	VM/Ansible-Welt
Artefakt	Image	VM/Box + Provisioning
„Build“	Dockerfile → Image	(optional) Packer → Image / oder Vagrant + Provisioning
Konfiguration	ENV, COPY, RUN	apt, template, service, systemd
Reproduzierbarkeit	sehr hoch	hoch (wenn idempotent)
Runtime	Container	VM

Wenn du Ansible sauber machst, lernen die Lernenden „Konfiguration als Code“ – das ist in der Praxis extrem gefragt, egal ob VM, Cloud oder Kubernetes.

Debugging & Qualität

Playbook testen

```
# in der VM (ansible_local) typischerweise:
sudo ansible-playbook /vagrant/provision/playbook.yml

# mehr Details:
sudo ansible-playbook /vagrant/provision/playbook.yml -vv
```

Häufige Fehler

- falscher Pfad zu Templates
- Service heisst anders (z.B. nginx vs apache2)
- Ports stimmen nicht (Proxy zeigt auf falschen Port)
- Rechte fehlen → `become: true` vergessen

Profi-Pattern (optional)

- **check mode** (trocken laufen lassen): `–check`
- **syntax check**: `–syntax-check`
- Variablen pro Umgebung (`group_vars`, `host_vars`)
- Rollen (`roles/`) für saubere Wiederverwendung

M319-LU06, M319-C1E