

Automagische Server-Installationen – Baue dir die helfende Zauberhand mit Ansible

von Thomas Bahn, 21. Juni 2021

DominoCamp'21 online



Thomas Bahn

- computerbegeistert seit 33 Jahren
- Mathematik & Informatik studiert
- Geschäftsführer bei IT-Dienstleister

Agenda

- Warum? Welches Problem löst Ansible?
- Was? die wichtigsten Grundlagen, Begriffe und Konzepte
- Wo? Herunterladen und Installieren
- Wie? von einfach bis komplex: Ansible an Beispielen erklärt
Auswahl der Ziele, direkter Aufruf, Playbooks, Variablen, Vorgaben,
Module, Rollen, ...

Warum?

Welches Problem
löst Ansible?



Welches Problem löst Ansible?

- Grundinstallation für neue Server
 - Konfigurationen, z. B. Firewall, Routing, IPv6
 - Programme & Werkzeuge
 - (administrative) Benutzer
- zusätzliche Installation für bestimmte Server, z. B. DMZ, Domino, ...
- spezielle Installation für jeden einzelnen Server
- nicht nur Server, sondern auch aktive Netzwerkkomponenten, Drucker, ...

Welches Problem löst Ansible?

- Installation & Konfiguration soll geändert werden
 - neue Standards
 - neue Programme/Werkzeuge
- etwas funktioniert nicht, weil irgendwo etwas geändert wurde
 - schnell zurück zum Standard oder
 - automatische Neuinstallation des Servers (und Rücksicherung der Daten)

Welches Problem löst Ansible?

- Dokumentation des Sollzustands
- Serverkonfigurationen verwalten und flexibel ausspielen
- Versionsverwaltung
- modular pflegen (Wiederverwendung!)

Wie löst Ansible das Problem?

- Beschreibung des Sollzustands
- bei Ausführung wird der Sollzustand hergestellt
- Änderungen werden nur gemacht, wenn Soll $\neq$ Ist
- ggf. parallel auf vielen Zielsystemen

Voraussetzungen

- „Control node“ aka Ansible Controller:
 - Red Hat, CentOS, Debian, Ubuntu, macOS, jedes BSD-System, Solaris, ...
 - Python und Ansible installiert
- „Managed node“ aka Zielsysteme:
 - *nix und Windows
 - Zugang per SSH (bzw. Windows Remote Management = WinRM)
 - sonst nichts auf dem Zielsystem notwendig („Agentless“)



Was?

Die wichtigsten
Grundlagen, Begriffe
und Konzepte

Die wichtigsten Grundlagen, Begriffe und Konzepte

- „Control Node“ und „Managed Nodes“
- Inventory
- Task
- Modules
- Playbook
- Roles
- Variables
- Jinja2 Templates

„Control Node“

- aka Ansible Controller
- dort ist Ansible installiert
- dort liegen alle Konfigurationen für die Zielsysteme
- dort wird Ansible ausgeführt
- verbindet sich dann mit den Zielsystemen
- typischerweise ein zentraler Server für die komplette Infrastruktur

„Managed Nodes“

- die Zielsysteme
- **kein** Ansible installiert (auch keine Agenten o.ä.)
- müssen vom Ansible Controller erreicht werden können (SSH, WinRM)

Inventory

- Liste der (möglichen) Zielsysteme
 - Name, Adresse, Zugangsdaten
- Servergruppen (auch geschachtelt)
- Einstellungen für einzelne Server oder Gruppen

Task

- auszuführende Aufgabe, z. B.
 - Daten vom Zielsystem auslesen
 - Datei kopieren
 - Vorlage kopieren und Platzhalter ersetzen
 - Zeile/Textblock in Datei hinzufügen oder ersetzen
 - Programm installieren
 - Programm ausführen
 - ...

Modules

- abstrahiert die Aufgabe
- definiert, wie eine Aufgabe ausgeführt wird (Code)
- Nutzung unabhängig vom Betriebssystem (außer Windows)
- viele Module mitgeliefert
- noch viel mehr öffentlich verfügbar (Stichwort: Ansible Galaxy)

Collections (= Modul-Sammlungen) in der Dokumentation

amazon.aws	cisco.nxos	community.okd	fortinet.fortios	netbox.netbox
ansible.builtin	cisco.ucs	community.postgresql	frr.frr	ngine_io.cloudstack
ansible.netcommon	cloudscale_ch.cloud	community.proxysql	gluster.gluster	ngine_io.exoscale
ansible.posix	community.aws	community.rabbitmq	google.cloud	ngine_io.vultr
ansible.utils	community.azure	community.routeros	hetzner.hcloud	openstack.cloud
ansible.windows	community.crypto	community.skydive	hpe.nimble	openvswitch.openvswitch
arista.eos	community.digitalocean	community.sops	ibm.qradar	ovirt.ovirt
awx.awx	community.docker	community.vmware	infinidat.infinibox	purestorage.flasharray
azure.azcollection	community.fortios	community.windows	inspur.sm	purestorage.flashblade
check_point.mgmt	community.general	community.zabbix	junipernetworks.junos	sensu.sensu_go
chocolatey.chocolatey	community.google	containers.podman	kubernetes.core	servicenow.servicenow
cisco.aci	community.grafana	cyberark.pas	mellanox.onyx	splunk.es
cisco.asa	community.hrobot	dellemc.enterprise_sonic	netapp.aws	t_systems_mms.icinga_director
cisco.intersight	community.kubernetes	dellemc.openmanage	netapp.azure	theforeman.foreman
cisco.ios	community.kubevirt	dellemc.os10	netapp.cloudmanager	vyos.vyos
cisco.iosxr	community.libvirt	dellemc.os6	netapp.elementsw	wti.remote
cisco.meraki	community.mongodb	dellemc.os9	netapp.ontap	
cisco.mso	community.mysql	f5networks.f5_modules	netapp.um_info	
cisco.nso	community.network	fortinet.fortimanager	netapp_eseries.santricity	

https://docs.ansible.com/ansible/latest/collections/index_module.html

Module alleine in ansible.builtin

add_host	dpkg_selections	import_role	pip	subversion
apt	expect	import_tasks	raw	systemd
apt_key	fail	include	reboot	sysvinit
apt_repository	fetch	include_role	replace	tempfile
assemble	file	include_tasks	rpm_key	template
assert	find	include_vars	script	unarchive
async_status	gather_facts	iptables	service	uri
blockinfile	get_url	known_hosts	service_facts	user
command	getent	lineinfile	set_fact	validate_argument_spec
copy	git	meta	set_stats	wait_for
cron	group	package	setup	wait_for_connection
debconf	group_by	package_facts	shell	yum
debug	hostname	pause	slurp	yum_repository
dnf	import_playbook	ping	stat	

- Liste von Tasks (Aufgaben)
- Festlegung der betroffenen Zielsysteme

Roles

- machen Playbooks leichter verwaltbar
- Playbooks werden mit zugehörigen Dateien zusammen gespeichert
- Einheit zur Wiederverwendung

Variables

- Konfiguration flexibel anpassen je Server oder Gruppe
 - Servername, IP-Adresse, Gateway, bestimmte Tasks (nicht) ausführen, ...
- werden im Inventory und in Rollen gespeichert oder beim Aufruf mitgegeben
- klare Regeln für Priorisierung

- Daten flexibel „massieren“ und umformen
- z. B.
 - Strings formatieren
 - Pfad aus Dateinamen extrahieren
 - erstes Element einer Liste zurückgeben
 - Liste sortieren

Wo?

Herunterladen und
Installieren



Herunterladen und Installieren

- am bequemsten: Paketmanager der Distribution
- aktueller: mit pip
- noch aktueller: devel-Branch aus GitHub
- Ich bin bequem...

Beispiel: Control Node unter CentOS 7

- mit dem Paketmanager:

```
yum install epel-release
```

```
yum install ansible
```

- Abhängigkeiten, wie Python, werden mit installiert
- SSH-Zugang einrichten (nicht zwingend)

```
ansible --version
```

```
ansible 2.9.21
```

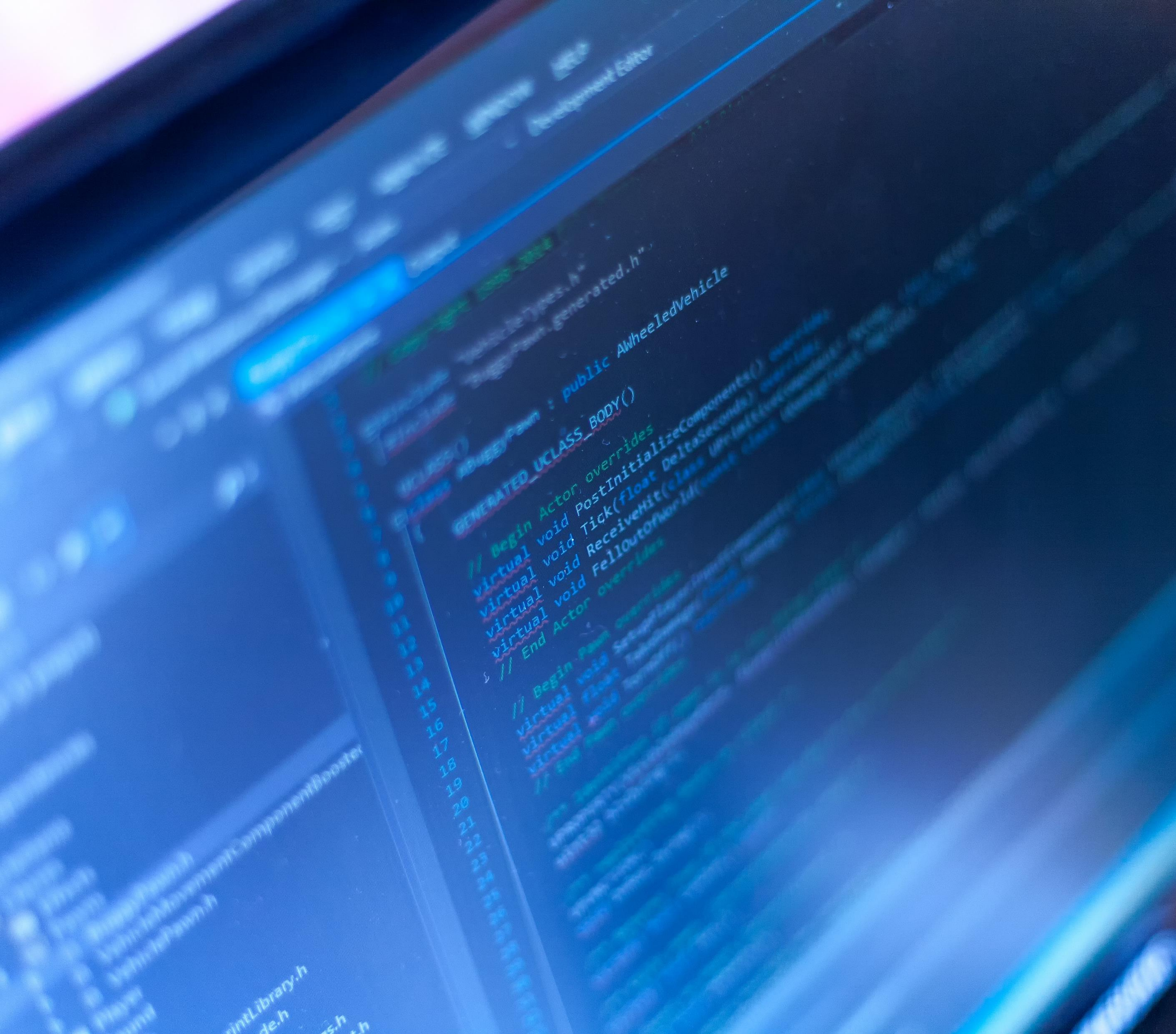
...

Beispiel: Managed Node unter CentOS 7

- **kein** Ansible installieren
- SSH-Zugang einrichten (hier zwingend)
- SSH-Zugang vom Ansible Controller aus testen
- fertig!

Einschub: Sicherheit

- Root-Benutzer nutzen?
- besser zusätzliche(r) Benutzer mit sudo-Rechten
- Best Practice: jeder Admin hat eigenen Benutzer mit eigenem Passwort
- SSH-Zugang nur per Zertifikat (per Passwort verbieten)
- Ansible Controller massiv härten („Bastion Host“)



Was?

von einfach bis
komplex: Ansible an
Beispielen erklärt

Inventory erstellen

- Welche Server gibt es?
- Wie kann der Ansible-Controller sie erreichen (IP-Adresse)?
- Welcher Benutzer, Passwort, Schlüsseldatei?
- Welche speziellen Einstellungen sollen noch gelten
- Welche Gruppen von Servern gibt es?
 - einheitliche Einstellungen pro Gruppe
 - Ansible-Aufrufe pro Gruppe

Inventory erstellen

- YAML oder ini-Datei: (fast) gleich mächtig
- Globale Datei vs. Übergabe per Kommandozeilenparameter
- im Weiteren: host.ini-Datei, die beim Aufruf übergeben wird

einfache Host.ini

```
server-080 ansible_host=192.168.4.17 ↵  
ansible_connection=ssh ansible_user=ansible
```

Host.ini mit Gruppe

```
server-080 ansible_host=192.168.4.17
```

```
[centos7]
```

```
server-080
```

```
[centos7:vars]
```

```
ansible_connection=ssh
```

```
ansible_user=ansible
```

Host.ini mit geschachtelten Gruppen

```
server-080 ansible_host=192.168.4.17  
[centos7]  
server-080  
[linux:children]  
centos7  
[linux:vars]  
ansible_connection=ssh  
ansible_user=ansible
```

Host.ini mit zwei Gruppenhierarchien

server-080 ...	[ki-lan]	[hh:children]
	[ki-dmz]	hh-lan
[centos7]	server-080	hh-dmz
server-080	[hh-lan]	[hh:vars]
[centos8]	[hh-dmz]	email_host=192.168.16.xxx
[debian10]		[lan:children]
[linux:children]	[ki:children]	ki-lan
centos7	ki-lan	hh-lan
centos8	ki-dmz	[dmz:children]
debian10	[ki:vars]	ki-dmz
[linux:vars]	email_host=192.168.0.xxx	hh-dmz
...		

- ansible-Kommando -i Inventory -m Modul Ziel
- 1. Beispiel

```
ansible -i host.ini -m ping server-080
```

```
server-078 | SUCCESS => {
```

```
    "ansible_facts": {
```

```
        "discovered_interpreter_python": "/usr/bin/python3.6"
```

```
    },
```

```
    "changed": false,
```

```
    "ping": "pong"
```

```
}
```

- 2. Beispiel

```
ansible -i host.ini -m setup server-080
```

- setup liefert ganz viele Fakten zum Ziel, formiert als JSON
- Hardware, Betriebssystem, Netzwerk, ...
- ausführlichere Ausgaben mit `-v ... -vvvv`, gilt auch für `ansible-playbook` u.a.

Ad-hoc-Ausführung

```
server-080 | SUCCESS => {  
  "ansible_facts": {  
    "ansible_default_ipv4": {  
      "address": "192.168.4.xxx",  
      "alias": "ens192",  
      "broadcast": "192.168.4.255",  
      "gateway": "192.168.4.xxx",  
      ...  
    },  
    ...  
  },  
  "ansible_distribution": "CentOS",  
  "ansible_distribution_major_version": "7",  
  "ansible_distribution_version": "7.9",  
  ...  
  "ansible_fqdn": "server-080",  
  "ansible_hostname": "server-080",  
  ...  
  "ansible_memfree_mb": 983,  
  "ansible_memtotal_mb": 1837,  
  ...  
  "ansible_nodename": "server-080",  
  "ansible_os_family": "RedHat",  
  ...  
  "ansible_processor_cores": 2,  
  "ansible_processor_count": 2,  
  ...  
}
```

Playbooks

- Mehrere Tasks auf bestimmten Zielen ausführen
- Beispiele:
 - Grundinstallation
 - Härtung für DMZ
 - Vorbereitung für Domino-Installation

Einschub: YAML

- YAML Ain't Markup Language
- „Sprache“ für Konfigurationsdateien
- einfach zu lesen
- arbeitet mit Einrückungen (z. B. Argumente für Kommando)
- Listen mit Einrückung und Bindestrichen
- Assoziative Listen (aka Objekte) mit Einrückung und „Name: Wert“-Paaren
- Es gibt auch Inline-Notationen für beides.
- # für Kommentare
- **nur Leerzeichen, keine Tabulatoren!!!**

Einfaches Playbook

```
---  
- name: Grundinstallation  
  hosts:  
    - server-080 (oder centos7 oder all)  
  tasks:  
    - name: Aktualisiere alle installierten Pakete (CentOS 7)  
      yum:  
        name: '*'  
        state: latest
```

Playbook ausführen

- Ausführung mit

```
ansible-playbook -i host.ini einfaches-playbook.yml
```

- sehr ausführliche Ausgaben mit:

```
ansible-playbook -i host.ini einfaches-playbook.yml -vvvv
```

- Syntaxprüfung mit:

```
ansible-playbook -i host.ini einfaches-playbook.yml --check-syntax
```

- Schritt-für-Schritt-Ausführung (Einzelbestätigungen) mit:

```
ansible-playbook -i host.ini einfaches-playbook.yml --step
```

Debug-Ausgaben

- z. B. Variablen oder Fakten ausgeben, Fortschritt nachvollziehen, Debuggen
 - name: Variablen ausgeben
- ```
debug:
 msg:
 - "server_ip: {{server_ip}}"
 - "server_name: {{server_name}}"
 - "ansible_distribution: {{ansible_distribution}}"
 - "ansible_distribution_version: {{ansible_distribution_version}}"
```

# Komplexeres Playbook

```

- name: Grundinstallation
 hosts:
 - centos7
 become: yes
 tasks:
 - name: Setze den hostname
 when: not skip_hostname
 hostname:
 name: "{{server_name}}"
 - name: Aktualisiere alle installierten Pakete (CentOS 7)
 yum:
 name: '*'
 state: latest
```

## Alle Programme aktualisieren

- in CentOS 7 mit yum, in CentOS 8 mit dnf, in Debian mit apt oder allgemein mit package (weniger Optionen)
- name: Aktualisiere alle installierten Pakete (CentOS 7)

**yum:**

**name:** '\*'

**state:** latest

## Programm installieren und deinstallieren

- Installieren:
  - name: Fail2Ban - fail2ban installieren (CentOS 7)  
yum:  
    name: fail2ban  
    state: **present**
- Deinstallieren:
  - name: smartmontools deinstallieren (CentOS 7)  
yum:  
    name: smartmontools  
    state: **absent**

## Verzeichnisse erstellen

- Erstelle ein Verzeichnis:
    - name: "Erstelle Verzeichnisse"
- file:**
- path: "/opt/assono/tls"
  - state: directory**
  - recurse: yes**
  - owner: chatbot
  - group: chatbot
  - mode: '0755'

## Tasks für mehrere Listenelemente ausführen (with\_items)

- statt eines gleich viele Verzeichnisse erstellen mit `with_items`:
  - name: "Erstelle mehrere Verzeichnisse"
    - file:
      - path: "{{item}}"
      - state: directory
      - ...
    - with\_items:
      - "{{target\_path\_backend\_files}}"
      - "{{target\_path\_backend\_config\_files}}"
      - "{{target\_path\_backend\_update\_files}}"
      - ...

## file ist sehr vielseitig

- (leere) Datei erstellen:
  - **file:**
    - path: "..."
    - state: touch**
- Datei oder Verzeichnis löschen:
  - **file:**
    - path: "..."
    - state: absent**
- Status abfragen:
  - **file:**
    - path: "..."
    - state: file**
- Symbolischen Link erstellen:
  - **file:**
    - src: "..."
    - dest: "..."
    - state: link**
- Berechtigungen ändern:
  - **file:**
    - path: "..."
    - owner: "notes"**
    - group: "notes"**
    - mode: "644"**

## Zeile in Datei schreiben

- Datei wird erstellt, falls sie noch nicht existiert

- name: Swappiness reduzieren

**lineinfile:**

**path:** /etc/sysctl.d/50-swappiness.conf

**line:** "vm.swappiness = 1"

**create:** yes

**owner:** root

**group:** root

**mode:** '0644'

## Textblock in Datei hinzufügen

- Für manche Änderungen muss ein Dienst oder der Rechner neu gestartet werden:
- name: Maximale Anzahl offener Dateien erhöhen

**blockinfile:**

path: /etc/security/limits.conf

**block:** |

\* soft nofile 65535

\* hard nofile 65535

**insertbefore:** '^# End of file'

- vom Ansible-Controller zum Ziel

- name: IPv6 deaktivieren

**copy:**

**src:** files/etc/sysctl.d/10-no-ipv6.conf

**dest:** /etc/sysctl.d/10-no-ipv6.conf

**owner:** root

**group:** root

**mode:** '0644'

- mit `remote_src` innerhalb des Zielsystems kopieren
- für die andere Richtung gibt es `fetch`

## Vorlagen verwenden

- Kopieren und Platzhalter ersetzen
- name: Fail2Ban - Konfiguration erstellen (CentOS 7)

### **template:**

**src:** jail.local.j2

**dest:** /etc/jail.local

**owner:** root

**group:** root

**mode:** '0644'

### **notify:**

- Fail2Ban - fail2ban neustarten

## Vorlagen verwenden

- Vorlagendatei `jail.local.j2`

```
Destination email address used solely for the interpolations in
jail.{conf,local,d/*} configuration files.
```

```
destemail = {{email_recipient}}
```

```
Sender email address used solely for some actions
```

```
sender = {{server_name}}@{{email_domain}}
```

## Suchen und Ersetzen in Dateien

- häufig sucht man eine Zeile, um eine Option zu setzen
- name: Logwatch - Output setzen (CentOS 7)

**lineinfile:**

**path:** /usr/share/logwatch/default.conf/logwatch.conf

**regexp:** '^Output'

**line:** "Output = mail"

**backup:** yes

## Neustart nach Änderungen

- Für manche Änderungen muss ein Dienst oder der Rechner neu gestartet werden:

- name: SELinux deaktivieren (CentOS 7)

lineinfile:

path: /etc/selinux/config

regexp: '^SELINUX='

line: "SELINUX=disabled"

backup: yes

notify:

- Rechner neustarten

## Handler erstellen und registrieren

- Definition des Handlers:
  - name: Rechner neustarten  
reboot:
  - name: Fail2Ban - fail2ban neustarten  
service:
    - name: fail2ban
    - state: restarted

## Programm ausführen

- Beispiel: Import einer Collection in MongoDB

- name: importiere Collection in MongoDB

```
command: /usr/bin/mongoimport --port 27017 --db {{mongodb_chatbot_dbname}} ↵
 --authenticationDatabase admin -u {{mongodb_admin_user}} ↵
 -p "{{mongodb_admin_password}}" --collection convomap --drop ↵
 --file "/tmp/db-export/convomap.json"
```

## Tasks für mehrere Dateien ausführen (with\_fileglob)

- Beispiel: Import mehrerer Collections in MongoDB
    - name: Durchlaufe alle Collections und importiere sie
      - command: `/usr/bin/mongoimport --port 27017 --db ... ↵`
        - `--collection {{item | basename | splitext | first}}` ↵
        - `--file {{item}}`
- with\_fileglob:**
- `"/tmp/db-export/*.json"`

## Dateien entpacken

- unarchive kann mit verschiedenen Archiven umgehen:
- name: Entpacke die Datenbanksicherung

### **unarchive:**

```
src: '{{ (database_exports.files | sort(attribute="path") | ↵
 last).path }}'
dest: '/tmp/db-export'
remote_src: yes
```

## Sind dir die Pipes (|) aufgefallen?

- In den letzten beiden Tasks kamen | vor:

```
--collection {{item | basename | splitext | first}}
```

```
{{ (database_exports.files | sort(attribute="path") | last).path }}
```

- Das sind **Jinja2-Templates**
- ein bisschen wie @Formelsprache

## Ergebnisse zwischenspeichern mit register

- Noch einmal einen Schritt zurück: Wie finden wir die aktuellen Datensicherung?
- name: Finde die Liste der Datenbanksicherungen  
find: paths="{{source\_path\_database\_exports}}"  
file\_type=file  
**register:** database\_exports
- Das Ergebnis des find wird in database\_exports (JSON-Struktur) gespeichert

## Ergebnisse zwischenspeichern mit register

- Mal anschauen:

- debug:

```
msg: 'Aktuellste Sicherung: {{ (database_exports.files | ↵
 sort(attribute="path") | last).path ↵
 if database_exports.files|count > 0 ↵
 else "Es gibt keine passende Datenbanksicherung" }}'
```

- debug:

```
var: database_exports
```

## Bedingte Ausführung

- Tasks nur unter bestimmten Bedingungen ausführen mit **when**
- Beispiel (**skip\_hostname** wird im Inventory gesetzt):

- name: Setze den hostname

**when: not skip\_hostname**

hostname:

name: "{{server\_name}}"

## Blöcke von Tasks erstellen

- Man kann mehrere Tasks zu einem Block zusammenfassen:
  - name: Aktuellste Datenbanksicherung importieren, wenn es eine gibt
- block:**
  - name: Verzeichnis erstellen ...
  - name: Entpacke die Datenbanksicherung ...
  - name: Durchlaufe alle Collections und importiere sie ...
  - name: Verzeichnis mit entpackten Dateien wieder löschen ...
- when: database\_exports.files|count > 0
- häufiger Anwendungsfall ist die Kombination mit when

## Playbooks auf mehrere Dateien aufteilen

- für mehr Übersicht kann man mehrere Dateien erstellen und einfügen:
  - name: Basis-Installation (CentOS 7)  
  **include\_tasks:** centos7.yml  
  when: ansible\_distribution == "CentOS" and ansible\_distribution\_major\_version == "7"
  - name: Basis-Installation (CentOS 8)  
  **include\_tasks:** centos8.yml  
  when: ansible\_distribution == "CentOS" and ansible\_distribution\_major\_version == "8"
  - name: Basis-Installation (Oracle Linux 8)  
  **include\_tasks:** ol8.yml  
  when: ansible\_distribution == "OracleLinux" and ansible\_distribution\_major\_version == "8"
  - name: Basis-Installation (Debian)  
  **include\_tasks:** debian.yml  
  when: ansible\_distribution == "Debian"

# Rollen

- für Wiederverwendung und leichtere Wartbarkeit Playbooks kapseln
- Dafür gibt es Rollen, die eine bestimmte Verzeichnisstruktur vorgeben.
- Einbinden in "Start-Playbook" über roles:

---

```
- name: Grundinstallation
 hosts: linux
 become: yes
 roles:
 - base-install
```

## Rolle generieren

- Grundgerüst für eine Rolle generieren:

```
ansible-galaxy role init --init-path roles/ base-install
```

```
└─ roles
 └─ base-install
 ├── defaults
 │ └─ main.yml
 ├── files
 ├── handlers
 │ └─ main.yml
 ├── meta
 │ └─ main.yml
 ├── README.md
 ├── tasks
 │ └─ main.yml
 ├── templates
 ├── tests
 │ ├── inventory
 │ └─ test.yml
 └─ vars
 └─ main.yml
```

- Variablen sinnvoll vorbelegen
- in `roles/{rolename}/defaults/main.yml`

---

# defaults file for base-install

skip\_hostname: false

skip\_hosts\_file: false

prepend\_hostnames:

append\_hostnames:

skip\_exim4: false

- Vorgaben überschreiben in
  - `roles/{rolename}/variables/main.yml`
  - Gruppenvariables im Inventory  
`[linux:vars]`  
`email_host=192.168.0.###`
  - Definition des Zielsystems  
`server-### ... email_host=127.0.0.1`
  - im Aufruf von `ansible-playbook`  
`ansible-playbook ... -e "email_host=192.168.16.###"`  
`ansible-playbook ... -e "@variablen.yml"`

## Benutzer erstellen

- name: Erstelle den Benutzer ansible

**user:**

name: ansible

shell: /bin/bash

## Öffentlichen Schlüssel zu `authorized_keys` hinzufügen

- name: füge öffentlichen Schlüssel zu `authorized_keys` hinzu

**authorized\_key:**

user: ansible

state: present

key: "{{lookup('file', '/home/ansible/.ssh/authorized\_keys')}}"

## Cron Job erstellen

- nach jedem Neustart:
    - name: Starte das Skript launcher-Skript nach jedem Neustart
- cron:**
- ```
name: "start launcher script at reboot"
special_time: reboot
job: "{{launcher_script}}"
```
- andere "spezielle Zeiten": annually, daily, hourly, monthly, weekly, yearly
 - sonst mit month, weekday, day, hour, minute selbst definieren

Firewall-Regeln hinzufügen (CentOS)

- Task
 - name: http freischalten
 - firewalld:**
 - service: http
 - permanent: yes
 - state: enabled
 - notify:**
 - firewalld neustarten
- zugehöriger Handler:
 - name: firewalld neustarten
 - systemd:**
 - name: firewalld
 - state: restarted

- Benutzer `ansible` erstellen und `sshd` sicherer einstellen:

```
ansible-playbook -i host.ini create-ansible-user.yml
```

- Grundinstallation mit vielen Konfigurationseinstellungen, Paketinstallationen usw.

```
ansible-playbook -i host.ini grundinstallation.yml
```

Variante: Lokale Installation

- Variante für unseren Chatbot-Server
- Ziel: Installation auf Kundenrechner
- Rahmenbedingung: keine zentrale Control Node
- Schritte:
 - manuelle Installation von `ansible`
 - manuelles Kopieren der Ansible-Scripte (für verschiedene Distributionen vorbereitet)
 - Ausführung eines Playbooks

Variante: Lokale Installation

- Playbook eingestellt auf localhost (geht immer, auch ohne Inventory):

- name: Installation grundlegender Pakete und des Chatbot-Servers
hosts: **localhost**
roles:
 - base-install
 - chatbot-server

- für Geheimnisse wie z. B.
 - Passwörter
 - Lizenzschlüssel
 - Dateien
- Informationen werden sicher verschlüsselt gespeichert.

Windows als Zielsystem

- ähnlich, aber etwas anders
- WinRM statt SSH
- für viele Module gibt es vergleichbare Windows-Pendants, z. B.
 - `win_command`
 - `win_copy`
 - `win_file`
 - `win_user`
 - ...

Quellen und Dokumentationen

- Ansible bei Redhat: www.ansible.com
- Ansible bei GitHub: github.com/ansible/ansible
- Ansible Community Documentation: docs.ansible.com/ansible_community.html
- Installing Ansible: docs.ansible.com/ansible/latest/installation_guide/intro_installation.html
- Windows Guides: docs.ansible.com/ansible/latest/user_guide/windows.html
- Templating (Jinja2): docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html
- Ansible Galaxy: galaxy.ansible.com und am allerwichtigsten
- **Index of all Modules: docs.ansible.com/ansible/latest/collections/index_module.html**

Kurse und Bücher

- <https://www.udemy.com/course/ansible-grundlagen/>
- www.rheinwerk-verlag.de/ansible-das-praxisbuch-fuer-administratoren-und-devops-teams/
- www.ansible.com/resources/ebooks
- www.ansible.com/resources/ebooks/mastering-ansible
- www.ansible.com/resources/ebooks/extending-ansible
- www.packtpub.com/catalogsearch/result/?q=ansible
- www.digitalocean.com/community/books/how-to-manage-remote-servers-with-ansible-ebook