



# Ansible — Orchestrierung, Konfiguration und Administration

Carsten Feuls  
<carsten.feuls@creativ.de>

30. November 2018

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

---

## Windows

- ▶ Vortrag
- ▶ Ansible (Dokumentation) / virtuelle Umgebung
- ▶ Fragen und Feedback

- ▶ distributions- und herstellerunabhängig
- ▶ mehr als 19 Jahre im Geschäft
- ▶ international aufgestellt
- ▶ etwa 45 Mitarbeiter in Deutschland
- ▶ breites Spektrum an OS-Software
- ▶ Schwerpunkte *Betrieb(OSSC)* (Debian/Ubuntu, RHEL/CentOS, etc.), *Datenbanken(PSCC)* (PostgreSQL), *Mail-Systeme(SC)*

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance



# Übersicht III



---

Windows



 Python

 2012

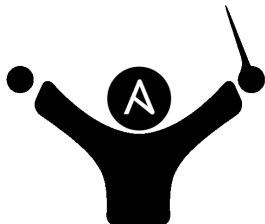
 Wachsende Anwendergemeinde

 Linux/Unix

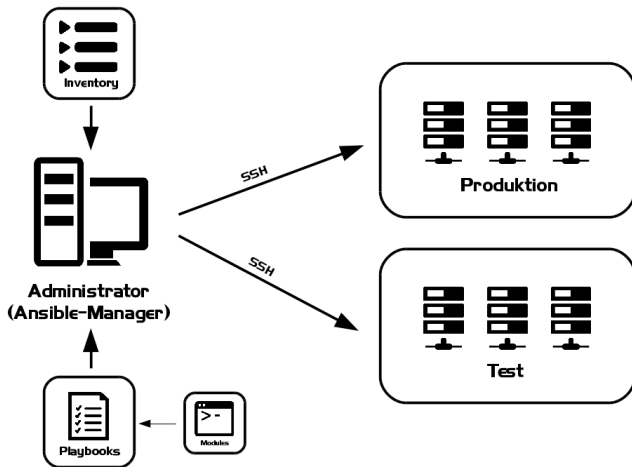
- ▶ <http://www.ansible.com/blog/2013/12/08/the-origins-of-ansible>
- ▶ Der Name Ansible kommt aus Enders Game von Orson Scott Card
- ▶ Ansible Inc., die Firma hinter Ansible, gehört seit Oktober 2015 zu Red Hat
- ▶ Ansible ist unter der GPL v3 lizenziert

## Ansible

- ▶ Orchestrierung
- ▶ Konfigurations-Management
- ▶ Administration: Ausführung  
Ad-hoc-Kommandos,  
Software-Deployment



- ▶ Deklarativ und idempotent: Wie soll das **Ergebnis** aussehen?
- ▶ Verwendet **YAML** zur Beschreibung von Systemen
- ▶ Benötigt **keine Agenten**: Kommunikation i.d.R. über SSH
- ▶ Zusätzlich wird Python auf den zu konfigurierenden Maschinen benötigt
- ▶ Entwickler betonen stets: Wir folgen dem **KISS**-Prinzip



- ▶ Vor Ansible wurde entweder vieles händisch gemacht.
- ▶ Eine Automatisierung gab es höchstens durch Shell-Skripte.
- ▶ Mehrere Hosts ließen sich nur durch PSSH oder Ähnlichem konfigurieren.
- ▶ Konfigurationsfehler waren keine Ausnahme sondern eher die Regel.
- ▶ Gleiche Konfiguration auf mehreren Servern war kaum möglich.

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks



Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

---

Windows

## Suse Installation

```
zypper ar https://download.opensuse.org/repositories ↵  
    /systemsmanagement/openSUSE_Leap_42.3/ ↵  
    systemsmanagement  
zypper install ansible ansible-lint
```

Ansible benötigt nicht unbedingt einen ssh-key, aber es vereinfacht die allgemeine Verwendung deutlich.

## SSH-Setup

```
ssh-keygen -t ecdsa  
ssh-copy-id <hostname>
```

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

---

Windows

## Inventory

- ▶ Einfache Textdatei
- ▶ Ein Host pro Zeile
- ▶ Möglichkeiten zur Zusammenfassung durch Patterns
- ▶ Es können Gruppen definiert werden („INI-Style“ oder „YAML-Style“)
- ▶ Es können Host- und Gruppen-spezifische Variablen definiert werden





# Beispiel 1



```
[client]
```

```
192.168.56.220
```

```
192.168.56.221
```

```
192.168.56.222
```

```
192.168.56.223
```

```
[web]
```

```
192.168.56.230
```

```
192.168.56.231
```

```
[database]
```

```
db0.beispiel.de
```

```
db1.beispiel.de
```

# Beispiel 2

```
[mynet]
192.168.56.[01:99]

[database]
db[0:1].meine-datenbanken.de

[cloud]
db-[a:f].meine-cloud.de

[merged-hosts:children]
database
cloud
```

# Beispiel 3: Variablen

```
[web]
web0.beispiel.de ansible_user=admin ansible_pass= ↵
secret

[database]
db[0:1].beispiel.de

[database:vars]
var0="foo"
var1="bar"
```

# Beispiel 4: Gruppen



```
router.beispiel.de
```

```
[web]
```

```
web0.beispiel.de ansible_user=admin ansible_pass=secret
```

```
[database]
```

```
db[0:1].beispiel.de
```

```
[web-env:children]
```

```
web
```

```
database
```

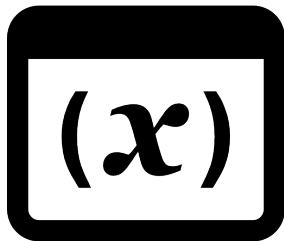
# Beispiel 5




```
[vagrant_boxes]
ansible ansible_host=10.123.123.101
managed-ubuntu ansible_host=10.123.123.102
managed-centos ansible_host=10.123.123.103
```

## Variablen in Dateien aufteilen


- ▶ Variablen-Definitionen können, losgelöst von den Host-Listen, in verschiedene Dateien aufgeteilt werden
- ▶ Dateien liegen i.d.R. relativ zur Inventory-Datei
- ▶ Dateien im YAML-Format



- ▶ Annahme 1: Inventory-Datei: `/etc/ansible/hosts`
- ▶ Annahme 2: In dieser Datei wurden der Host `h0.meine-hosts.de`, sowie die Gruppen `database` und `cloud` definiert

 /etc/ansible/host\_vars/h0.meine-hosts.de


```
---  
ansible_become: true  
ansible_become_user: admin  
ansible_become_pass: secret  
ansible_become_method: sudo
```

 /etc/ansible/group\_vars/database

```
---  
ansible_become_user: dbadmin  
ansible_become_pass: secret
```




- ▶ Seit Ansible 1.4 können statt Dateien auch Verzeichnisse mit den entsprechenden Group- und Host-Namen erstellt werden
- ▶ Aus diesen Dateien werden dann Variablen aus **allen** Dateien in dem jeweiligen Verzeichnis eingelesen

 /etc/ansible/group\_vars/cloud/part\_0

---

```
ansible_user: admin
ansible_pass: secret
```

 /etc/ansible/group\_vars/cloud/part\_1

---

```
ansible_become: True
ansible_become_method: sudo
```

- ▶ Gleiche Variablen-Definitionen an unterschiedlichen Orten haben eine Rangfolge
- ▶ Verbindungs-Variablen im Inventory (`ansible_host`, ...) können nicht überschrieben werden
- ▶ Host-Variablen überschreiben Gruppen-Variablen (Definitionen in `host_vars` überschreiben jene in `group_vars`)
- ▶ Untergruppen- überschreiben Gruppen-Variablen (Definitionen in `group_vars/hamburg` überschreiben jene in `group_vars/all`)

- ▶ Ein Inventory kann auch ein Skript sein
- ▶ Interessant bei existierenden CMDB
- ▶ Ausgabe eines JSON-Hash/Dictionary aller Gruppen (Hosts, Variablen) auf stdout
- ▶ Der Parameter `--host <hostname>` gibt ein JSON-Hash/Dictionary aller Variablen des entsprechenden Hosts zurück

```
{
  "databases" : {
    "hosts" : [ "host1.example.de", "host2.example.de" ],
    "vars" : {
      "a" : true
    }
  },
  "webserverns" : [ "host2.example.de", "host3.example.de" ],
  "duesseldorf" : {
    "hosts" : [ "host1.example.de", "host4.example.de", "host5.example.de" ],
    "vars" : {
      "b" : false
    },
    "children": [ "mabuse", "5points" ]
  },
  "mabuse" : [ "host6.example.com" ],
}
```

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

---

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

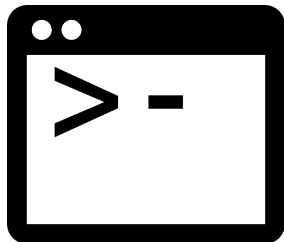
---

Windows



## Modules

- ▶ Modules arbeiten einzelne Aufgaben ab (z.B. Benutzer anlegen, Pakete installieren...)
- ▶ Ausgabe über stdout im JSON-Format
- ▶ Werden von Ansible auf dem Client ausgeführt
- ▶ In Verbindung mit dem Inventory sehr gut zur Ad-Hoc-Administration geeignet



➤ Einzelne Kommandos

📌 Einfache Aufgaben

❗ Sammeln von Informationen



```
ansible {Pattern} -i {Inventory} -m {Module} -a {Parameter}
```

Weitere Parameter:

- ▶ `-vvvv`
- ▶ `--user|-u`
- ▶ `--ask-pass|-k`
- ▶ `--private-key`
- ▶ `--become|-b, --become-method, --become-user`
- ▶ `--ask-become-pass|-K`



```
ansible {Pattern} -i {Inventory} -m {Module} -a {Parameter}
```

- ▶ Hier kann die Inventory-Gruppe oder das Schlüsselwort `all` angegeben werden
- ▶ Desweiteren können auch die „klassischen Mengenoperatoren“ angewandt werden (Vereinigung, Schnitt, Komplement)

## Vereinigung

```
ansible webservers:dbservers ...
```

## Schnitt

```
ansible webservers:&duesseldorf ...
```

## Komplement/Schnitt

```
ansible webservers:!moenchengladbach ...
```



```
ansible all -i demo_hosts -m ping
```

```
ansible | success >> {  
  "changed": false,  
  "ping": "pong"  
}  
  
managed-ubuntu | success >> {  
  "changed": false,  
  "ping": "pong"  
}  
  
managed-centos | success >> {  
  "changed": false,  
  "ping": "pong"  
}
```

## Beispiel 2



```
ansible all -i demo_hosts -m shell -a "ps -eo pcpu,user,args |  
    sort -r -k1 | head -n5" ↵
```



```
ansible all -i demo_hosts -m setup  
ansible all -m setup | grep '"ansible_system"'
```

Ergebnis: Zeigt umfangreiche System-Informationen an (**Facts**).



## Beispiel 4



```
ansible all -i demo_hosts -m copy -a 'src=foo.bar dest=/etc/foo ↵  
.bar mode=640'
```

Ergebnis: Die Datei lokale foo.bar wird auf **alle** Hosts nach /etc/foo.bar kopiert.

- ▶ Herunterfahren von n Maschinen
- ▶ Dateien kopieren
- ▶ Einfaches Software-Deployment
- ▶ Benutzer anlegen/entfernen
- ▶ Pakete installieren

- ▶ [http://docs.ansible.com/modules\\_by\\_category.html](http://docs.ansible.com/modules_by_category.html)
- ▶ GitHub
- ▶ Können in beliebiger Sprache selbst verfasst werden (stdout, JSON)

# Auflisten aller verfügbaren Module



```
ansible-doc -l
```

```
ansible-doc copy
```

- ▶ PostgreSQL
- ▶ MySQL
- ▶ Redis
- ▶ MongoDB
- ▶ ...

- ▶ shell
- ▶ command
- ▶ script
- ▶ raw
- ▶ expect
- ▶ ping
- ▶ wait\_for

- ▶ copy
- ▶ fetch
- ▶ file
- ▶ template
- ▶ assemble
- ▶ lineinfile
- ▶ replace
- ▶ stat
- ▶ slurp
- ▶ ...



- ▶ package
- ▶ zypper
- ▶ pip
- ▶ gem

- ▶ bzt
- ▶ git
- ▶ hg
- ▶ svn

- ▶ user
- ▶ group

- ▶ cron
- ▶ mount
- ▶ sysctl
- ▶ service
- ▶ systemd
- ▶ ...

- ▶ apache2\_mod\_proxy (nur Debian)
- ▶ apache2\_module (nur Debian)
- ▶ django\_manage
- ▶ ejabberd\_user
- ▶ httpasswd
- ▶ jboss
- ▶ letsencrypt
- ▶ ...

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

# Übersicht III



---

Windows



## YAML

- ▶ Rekursives Akronym: YAML Ain't Markup Language
- ▶ vereinfachte, menschenlesbare Auszeichnungssprache
- ▶ Datenserialisierung
- ▶ Idee: Man kann alles mittels (assoziativer) Felder beschreiben

YAML

- ▶ Bezeichner: Alphanumerisch inkl. Unterstrich ( \_ )
- ▶ Werte: String, Integer, Float, Date/Time, Sequence, Mapping
- ▶ Datentypen werden automatisch erkannt, sonst String

```
key: value
```

- ▶ Sind i.d.R. nicht innerhalb von Anführungszeichen definiert
- ▶ Können aber auch mit einfachen oder doppelten Anführungszeichen definiert werden
- ▶ In doppelten Anführungszeichen können weitere Anführungszeichen *gequotet*, sowie auch *Steuerzeichen* verwendet werden

A string in YAML

'A singled-quoted string in YAML'

"A double-quoted \"string\" in YAML"

"This is a colon: inside"

- Zahlen können in „gängigen“ Ausdrücken formuliert werden

```
12      # integer
014     # octal
0xC     # hexadecimal
13.4    # float
1.2e+34 # exponential number
.inf    # infinity
```

- ▶ Der Nullwert kann mit `null` oder `~` angegeben werden

- Ein Datum und/oder Uhrzeit wird im ISO-8601-Standard angegeben

```
2002-12-14          # simple date  
2015-06-22 11:02:57 # date und time
```

- ▶ Liste von Werten durch vorgestelltes Minus- und Leerzeichen
- ▶ Alternativ auch eingeklammert in [] und durch Kommata getrennt
- ▶ Sehr häufig Liste oder Array genannt

- PHP
- Perl
- Python

[PHP, Perl, Python]

- ▶ Bei Mappings werden die Bezeichner, gefolgt von einem Doppelpunkt, mind. einem Leerzeichen und dem Wert angegeben
- ▶ Sehr häufig auch *Hash*, *Dictionary* oder *assoziative Liste/Array* genannt

```
PHP: 5.4  
Perl: 5.14  
Python: 2.7
```

```
{PHP: 5.4, Perl: 5.14, Python: 2.7}
```



- ▶ Listen können verschachtelt werden, da die Werte selbst wieder Sequences oder Mappings sein können

```
- {name: John Smith, age: 33}  
- name: Mary Smith  
  age: 27
```

```
men: [John Smith, Bill Jones]  
women:  
  - Mary Smith  
  - Susan Williams
```

- ▶ Werden mit einem # eingeleitet

```
# Comment on a line
```

DBs:

```
- name: "erste_datenbank"
  num: "1"
  mount: "/DBA"
  drive: "sdc"
  port: "9410/tcp"
  alias: "sqlauto"
  partitions:
    rootdbs: {id: "1", size: "2G"}
    tmpdbs: {id: "2", size: "2G"}
    logdbs: {id: "3", size: "2G" }
    blobdbs: {id: "5", size: "2G"}
    datadbs: {id: "6", size: "max"}
```

ONCONFIG\_VARS:

## VHOSTS:

```
- name: "www.example.de"
  https: "true"
  enabled: "true"
  root: "/var/www/example/web/"
  default_values: |
    Options -Indexes +FollowSymlinks +MultiViews
    AllowOverride All
    Require all granted
    DirectoryIndex index.php
  dir:
    - name: "example"
      php_version: "7.2"
      data_path: "/var/www/example/web/"
      enabled: "true"
      port: "9000"
```

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

---

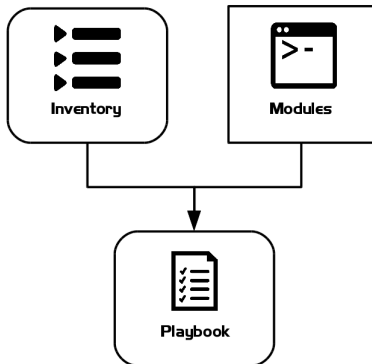
Windows

## Playbooks


- ▶ Playbooks sind deklarative Beschreibungen im YAML-Format
- ▶ Tasks beschreiben Zustände
- ▶ Variablen und Kontrollstrukturen
- ▶ Blöcke
- ▶ Templates und Dateien
- ▶ Verwendung zusammen mit Inventory








- Jedes Playbook besteht aus mindestens einem **Play**

 update-repo.yml

```
---  
- name: update repos  
  hosts: managed-ubuntu  
  tasks:  
    - name: update repos  
      apt: update_cache=yes
```

- ▶ Jeder **Task** löst ein **Modul** aus (parametrisiert)
- ▶ **Reihenfolge** wird eingehalten (von „oben“ nach „unten“)
- ▶ **Parallel** auf allen Nodes ausgeführt

 install-apache2.yml

```
---  
- name: ensure apache available  
  hosts: managed-ubuntu  
  tasks:  
    - name: update repos  
      apt: update_cache=yes  
    - name: install apache2  
      apt: name=apache2 state=present
```

- ▶ Ermöglicht es Tasks vor/nach den eigentlichen Tasks auszuführen.
- ▶ Damit kann z.b. das Monitoring während des Ansible Laufs stumm geschaltet werden.
- ▶ Auch besteht die Möglichkeit für die Wartung eines Webserver diesen aus dem LoadBalancer auszutragen.



```
ansible-playbook -i {Inventory} {Playbook}
```

Bis auf wenige Unterschiede die gleichen Parameter wie `ansible`.

- ▶ `--check|-C`
- ▶ `--extra-vars|-e`



```
ansible-playbook -i {Inventory} {Playbook}
```



```
ansible-playbook -i hosts demo.yml
```

```
PLAY [all] *****

GATHERING FACTS *****
ok: [managed-ubuntu]
ok: [managed-centos]

TASK: [root msg] *****
changed: [managed-ubuntu]
changed: [managed-centos]

PLAY RECAP *****
managed-centos      : ok=2    changed=1    unreachable=0    failed=0
managed-ubuntu      : ok=2    changed=1    unreachable=0    failed=0
```

- ▶ Variablen können in Playbooks der YAML-Syntax folgend definiert werden
- ▶ Variablen sind stets global

```
- hosts: all
  vars:
    # settings for ansible to connect to the vagrant machine ←
    ansible_ssh_user: "admin"
    ansible_ssh_private_key_file: ~/.ssh/ ←
      insecure_private_key
    my_user: user0
    users: [user1, user2]
    customers:
      - {name: John Doe, id: 1}
      - {name: Jane Doe, id: 2}
```



## Playbook

```
- hosts: all
vars:
  Hallo: Welt
pre_tasks:
  - name: disable nagios alerts for this host webserver service
    nagios: 'action=disable_alerts host={{ inventory_hostname }} services=webserver'
tasks:
  - name: Geben Welt aus
    debug:
      msg: "{{ Hallo }}"
    notify: Gebe Hallo aus
roles:
  ....
post_tasks:
  - name: wait for webserver to come up
    wait_for: 'host={{ inventory_hostname }} port=80 state=started timeout=80'

handlers:
  ....
```

- ▶ Variablen werden mittels Jinja2 referenziert
- ▶ Syntax: Variablenname von `{{` und `}}` umschlossen

```
{{ my_user }}  
{{ users[0] }}  
{{ customers[1].name }}
```

- ▶ Die Verwendung einer undefinierten Variable lässt den Task scheitern
- ▶ Ein Standardwert kann mittels eines sogenannten *Jinja2-Filter* gesetzt werden

```
{{ some_variable | default(8) }}
```

```
inventory_hostname
ansible_hostname
hostvars['<hostname>']['<wert>']
group_names -> Liste aller Gruppen in welcher der Host ist ←
groups -> Liste aller Hosts und Gruppen im Inventory
ansible_play_hosts -> Alle Hosts des aktuellen Plays
ansible_play_batch -> Alle Hosts des aktuellen Batch
ansible_check_mode -> Check Modus ist aktiv
```

/etc/hosts

```
{% for group_name in group_names %}  
{% for host in groups[group_name] %}  
{{hostvars[host]['ansible_host']}} {{hostvars[host][' ←  
    inventory_hostname']}}  
{% endfor %}  
{% endfor %}
```

- Es existiert eine Vielzahl von Jinja2-Filtern

```
{{ list1 | min }}  
{{ [3, 4, 2] | max }}  
{{ [3, 4, 2] | length }}  
{{ list2 | unique }}  
{{ list3 | union(list0) }}  
{{ ubuntu_version | version_compare('12.04', '>=') }}  
{{ mynumber | log }}  
{{ list4 | join(" ") }}  
{{ myaddr | ipaddr }}  
{{ mypassword | password_hash('sha512') }}  
{{ path | basename }}  
{{ "hello" | upper }}
```

### ansible linux -i hosts -m setup

```
managed-ubuntu | success >> {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "10.0.2.15",
      "10.123.123.101"
    ],
    ...
    "ansible_distribution": "Ubuntu",
    "ansible_distribution_release": "trusty",
    "ansible_distribution_version": "14.04",
    "ansible_os_family": "Debian",
    ...
    "ansible_devices": {
      "sda": {
        "model": "VMware Virtual S",
        ...
      }
    }
  }
  "changed": false
}
```

- ▶ Facts sind verschachtelte Datenstrukturen umfangreicher Systeminformationen
- ▶ Referenzierbar mittels Jinja2

```
{{ ansible_devices.sda.model }}
```

```
{{ ansible_all_ipv4_addresses[0] }}
```




- ▶ `group_by` ermöglicht es Gruppen Anhand von Facts zu definieren
- ▶ Diese Gruppen haben die gleichen Berechtigungen wie die im Inventory.

## `group_by`

```
- group_by:  
  key: machine_{{ ansible_machine }}
```

- ▶ Mittels des Schlüsselwortes `register` kann die Ausgabe eines Tasks in eine Variable umgeleitet werden
- ▶ Die so definierte Variable ist eine Liste, welche die beiden Variablen `stdout_lines` und `rc` enthält

 `demo-play.yml`

```
- name: demo play
  hosts: all
  tasks:
    - name: content of motd
      shell: cat /etc/motd
      register: motd_contents
    - command: /bin/false
      register: foo_result
      ignore_errors: True
```

```
{{ motd_contents.stdout_lines[0] }}
{{ foo_result.rc }}
```

- ▶ Parameter `-vvvv` für `ansible-playbook`
- ▶ Mit dem Schlüsselwort `debug` können pro Task Ausgaben erzeugt werden
- ▶ Sinnvoll in Verbindung mit der `when`-Direktive

```
- debug:
  msg="System {{ inventory_hostname }} has uuid ←
    {{ ansible_product_uuid }}"
- name: display all variables/facts known for a ←
  host
  debug:
    var=hostvars[inventory_hostname]
```

```
- debug:
  msg="System {{ inventory_hostname }} has ←
      gateway {{ ansible_default_ipv4.gateway }}"
  when: ansible_default_ipv4.gateway is defined
- debug:
  msg="System {{ inventory_hostname }} has ←
      gateway {{ ansible_default_ipv4.gateway }}"
  verbosity: 1
  when: ansible_default_ipv4.gateway is defined
```

Die Ausführung von Tasks kann mit dem Schlüsselwort `when` konditioniert werden:

```
- name: install curl
  apt: name=curl
  when: ansible_pkg_mgr == "apt"
```

Hinweis: `ansible_pkg_mgr` ist ein Fact.

Konditionierungen können aufgereiht werden:

```
- name: install curl
  apt: name=curl
  when: ansible_pkg_mgr == "apt"
- name: install curl
  yum: name=curl
  when: ansible_pkg_mgr == "yum"
```

Mehrere Bedingungen für die Konditionierung können mit Hilfe der Schlüsselworte `and` und `or` formuliert werden:

```
- name: install curl on Red Hat 6, derivatives, and later ↵  
  yum: name=curl  
  when: ansible_os_family == "RedHat" and ↵  
        ansible_lsb.major_release|int >= 6
```

Eine bedingte Ausführung eines Tasks anhand eines Rückgabewertes kann ebenfalls erreicht werden:

```
- command: /bin/check  
  register: foo_result  
  ignore_errors: true  
- command: /bin/execute  
  when: foo_result|success  
- command: /bin/execute_else  
  when: foo_result|failed
```

```
when: foo_result.rc == 0  
when: foo_result.rc != 0
```



Das Ausführungsergebnis einzelner Aufgaben lässt sich beeinflussen und bietet sich gerade bei Nutzung des 'command'-Modules an:

```
- command: cat /etc/passwd  
  changed_when: false
```

Auch der Fehlschlag selbst ist änderbar:

```
- command: cat /etc/passwd  
  register: passwd  
  changed_when: false  
  failed_when: 'admin' in passwd.stdout
```

„Schleifen“ helfen Beschreibungen zu verkürzen:

```
- name: add several users
  user: name="{{ item }}" state=present groups=wheel
  with_items:
    - anna
    - paul
```

Äquivalent:

```
- name: add user anna
  user: name=anna state=present groups=wheel
- name: add user paul
  user: name=paul state=present groups=wheel
```

„Schleifen“ helfen Beschreibungen zu verkürzen:

```
- name: add several users
  user: name="{{ item }}" state=present groups=wheel
  loop:
    - anna
    - paul
```

Äquivalent:

```
- name: add user anna
  user: name=anna state=present groups=wheel
- name: add user paul
  user: name=paul state=present groups=wheel
```

„Schleifen“ über einer Sequence von Mappings:

```
- name: add several users
  user:
    name="{{ item.name }}"
    state=present
    groups="{{ item.groups }}"
  with_items:
    - { name: 'paul', groups: 'wheel' }
    - { name: 'anna', groups: ['admins', 'wheel'] }
```

„Schleifen“ über einer Sequence von Mappings:

```
- name: add several users
  user:
    name="{{ item.name }}"
    state=present
    groups="{{ item.groups }}"
  loop:
    - { name: 'paul', groups: 'wheel' }
    - { name: 'anna', groups: ['admins', 'wheel'] }
```

„Schleifen“ können auch verschachtelt werden:

```
- vars:
  users: ['paul', 'anna']
- name: give users access to multiple databases
mysql_user: name="{{ item[0] }}" priv="{{ item[1] }} ←
             }}".*:ALL append_privs=yes password=foo
with_nested:
  - "{{ users }}"
  - [ 'kundendb', 'angestelltentdb', 'lieferantendb' ]
```

„Schleifen“ können auch verschachtelt werden:

```
- vars:
    users: ['paul', 'anna']
- name: give users access to multiple databases
  mysql_user: name="{{ item[0] }}" priv="{{ item[1] }} ←
               """.*:ALL append_privs=yes password=foo
  loop: "{{ users | product(['kundendb', 'angestellten ←
                           ', 'lieferantendb' ]) | list }}"
```

„Schleifen“ sind ein mächtiges Werkzeug.

- ▶ Fileglobs: `with_fileglob`
- ▶ Sequenzen: `with_sequence`(alt)
- ▶ [http://docs.ansible.com/playbooks\\_loops.html](http://docs.ansible.com/playbooks_loops.html)



- ▶ Nicht direkt möglich
- ▶ Nur durch Umwege Realisierbar
- ▶ Hebt die Parallelität von den Tasks auf.

### „Achtung“ Kaputtes Beispiel.

```
- vars:
  users:['paul', 'anna']
- name: ssh-key einrichten
  authorized_key: user="{{ item }}" , key="{{ item }}"
  with_fileglob: "/playbooks/files/{{ item }}/*"
  loop: "{{ users }}"
```

playbook: configure\_user.yml

```
- name: Gruppe hinzufügen
  group: name="{{ single_user }}", state=present
- name: Benutzer hinzufügen
  user: name="{{ single_user }}", group="{{ single_user }}", state=present ←
- name: ssh-key einrichten
  authorized_key: user="{{ single_user }}", key="{{ item }}" ←
  with_fileglob: "/playbooks/files/{{ single_user }}/*"
```

playbook: main.yml

```
- vars:
  users:['paul', 'anna']
- include: configure_user.yml
  loop: "{{ users }}"
  loop_control:
    loop_var: single_user
```

„Blöcke“ erlauben das Zusammenfassen von Aufgaben.

```
- block:
  - name: install apache2 on CentOS
    yum: name=httpd state=latest
  - name: copy static content
    copy: src=index.html dest=/var/www/html/index.html mode ←
          =0755
when: ansible_pkg_mgr == 'yum'

- block:
  - name: install apache2 on SUSE
    zypper: name=apache2 state=latest
  - name: copy static content
    copy: src=index.html dest=/srv/www/html/index.html mode ←
          =0755
when: ansible_pkg_mgr == 'zypper'
```

- ▶ Werden mit **eindeutigem Namen** wie Tasks definiert
- ▶ Werden mittels `notify` innerhalb einer Task angegeben und am Ende der Task-Liste des Plays ausgelöst
- ▶ Jeder Handler wird dabei nur genau einmal ausgelöst

```
tasks:
  - name: configuration file
    copy: src=httpd.conf dest=/etc/apache2/httpd.conf
    notify:
      - restart apache
handlers:
  - name: restart apache
    service: name=apache state=restarted
```

- ▶ Das Schlüsselwort `serial` definiert wie viele Durchläufe jeweils **gleichzeitig** (batch) ablaufen sollen
- ▶ Kann sowohl für einzelne Tasks als auch den ganzen Play definiert werden

```
- name: update  
  hosts: all  
  serial: 5
```

- ▶ Tasks können an andere Hosts delegiert werden
- ▶ Ideal um Wartungsfenster zu öffnen
- ▶ Man darf diese funktion auf keinen Fall unterschätzen

```
- name: schedule downtime
  nagios:
    action=downtime
    host="{{ ansible_hostname }}"
    minutes=15
    service=all
    cmdfile=/var/lib/icinga/rw/icinga.cmd
  delegate_to:
    monitor-server.beispiel.de
```




## Firewalling (Bsp: ferm)

```
- name: Firewall
  template:
    src: templates/firewall.conf.j2
    dest: /etc/ferm.d/apache.conf
    delegate_to: '{{ nfs_master }}'
```

## Postgresql pg\_hba.conf

```
- name: Konfiguriere Datenbank Zugriff
  lineinfile:
    path: /var/lib/postgres/data/pg_hba.conf
    line: "host test test {{ ansible_host }}/32 trust"
    regexp: "host test test {{ ansible_host }}/32 trust"
    state: present
    delegate_to: "{{ database_server }}"
```

- Tasks können mittels des Schlüsselwortes **include** auch aus anderen Dateien inkludiert werden

 tasks/foo.yml

```
- name: placeholder foo  
  command: /bin/foo  
- name: placeholder bar  
  command: /bin/bar
```

```
tasks:  
  - include: tasks/foo.yml
```

- ▶ include (Deprecated)
- ▶ import\_playbook
- ▶ import\_role
- ▶ import\_tasks
- ▶ include\_role
- ▶ include\_tasks
- ▶ include\_vars

- ▶ Mit dem Schlüsselwort `tags` können einzelne Tasks markiert werden
- ▶ Beim Aufruf von `ansible-playbook` kann mittels des Parameters `--tags` die Ausführung auf entsprechend markierte Tasks eingeschränkt werden
- ▶ Definierte Tags können mit der Option `--list-tags` abgerufen werden.

```
- name: install apache
  apt: name=apache2 state=present
  tags:
    - web
    - testenv
    - install
```

- ▶ Einen Tag mit dem Namen `always` wird immer ausgeführt egal welcher Tag gerade läuft
- ▶ Wenn der Tag dagegen `never` lautet wird dieser Task nur ausgeführt wenn er explizit durch einen Tag aufgerufen wird.

- ▶ Tasks können als unterschiedliche Benutzer auf dem Zielhost laufen
- ▶ Dazu eignen sich `become` und `become_user`

```
- name: message for admin  
  copy: src=msg dest=~ /msg mode=640  
  become: true  
  become_user: admin
```

- Es können mittels `environment` Umgebungsvariablen **pro Task** gesetzt werden

```
- apt: name=cobbler state=present
  environment:
    http_proxy: "http://proxy.example.com:8080"
```



- ▶ Mit `wait_for` kann auf bestimmte „Zustände“ gewartet werden
- ▶ Nützlich dem initialen Deployment von Diensten

```
- name: wait for webserver to come up  
  wait_for: host="{{ inventory_hostname }}" port=80 ←  
            state=started timeout=80
```

- ▶ Mit `wait_for_connection` kann man auf eine SSH Verbindung warten
- ▶ Nützlich um auf den Erfolgreichen Bootvorgang eines Hosts zu warten

## Fehlschlag Provuzieren

```
- fail:  
  msg: "The system may not be provisioned according to ←  
       the CMDB status."  
  when: cmdb_status != "to-be-staged"
```

- Nützlich um durch eine Bedingung einen Automatischen Abbruch zu erwirken.

- ▶ Statt bestimmte Tasks an *localhost* zu delegieren sollte das Schlüsselwort `local_action` verwendet werden
- ▶ Task wird auf dem Ansible-Host ausgeführt

```
- name: take out of load balancer pool
  command: /usr/bin/take_out_of_pool {{ inventory_hostname }} ↵
  delegate_to: 127.0.0.1
```

```
- name: take out of load balancer pool
  local_action: command /usr/bin/take_out_of_pool ↵
  {{ inventory_hostname }}
```

- ▶ Normalerweise laufen die Tasks solange bis Zugriff auf alle Hosts fehlschlägt
- ▶ Kann mit dem Schlüsselwort `max_fail_percentage` beeinflusst werden
- ▶ Mit `any_errors_fatal: true` weisen wir Ansible an mit einem sofortigen Abbruch im Fehlerfall auszulösen.

```
- hosts: webserver  
  max_fail_percentage: 30  
  serial: 10  
- hosts: datenbanken  
  any_errors_fatal: true
```

- ▶ Mit dem Ausdruck `gather_facts: false` kann verhindert werden, dass für den jeweiligen Task eine Fact-Datenstruktur aufgebaut wird

```
- hosts: webserver  
  gather_facts: no
```

- ▶ Mit dem Schlüsselwort `vars_prompt` können Aufforderungen zur Benutzereingabe ausgelöst werden
- ▶ Variablen werden **vor** den Tasks abgefragt

```
vars_prompt:
- name: passphrase
  prompt: "Passphrase"
  private: yes
tasks:
- name: create temp file
  command: mktemp /tmp/encdisk.XXXXXXX
  register: tmp
- name: write passphrase to temp file
  template: src=templates/encrypt-pass.j2 dest={{tmp.stdout}}
- name: open encrypted disk
  shell: cat {{tmp.stdout}} | cryptsetup luksOpen {{dev}} {{name}}
- name: shred temp file
  command: shred -z -u {{tmp.stdout}}
```

```
# Requires ansible 1.8+
- name: 'YUM - fire and forget task'
  yum: name=docker-io state=installed
  async: 1000
  poll: 0
  register: yum_sleeper

- name: 'YUM - check on fire and forget task'
  async_status: jid={{ yum_sleeper.ansible_job_id }}
  register: job_result
  until: job_result.finished
  retries: 30
```



`run_once`

- ▶ Ein Task wird nur auf einem Hosts des Batches ausgeführt.
- ▶ Ermöglicht das die Konfiguration eines Clusters nur einmal ausgeführt wird.

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

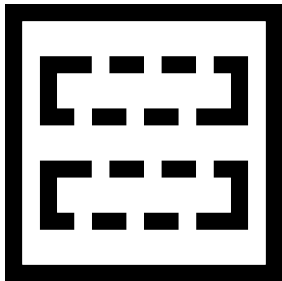
Performance

---

## Windows

## Templates

- ▶ Ersetzt Datei-Inhalte und kopiert diese auf den Zielhost
- ▶ Template-Engine **jinja2**
- ▶ `http://jinja.pocoo.org/docs/dev/`
- ▶ Schlüsselwort `template`
- ▶ Es können automatisierte Backups von bestehenden, zu überschreibenden Dateien angelegt werden



- ▶ Zwei Arten von Trennsymbolen: {% ... %} und {{ ... }}
- ▶ Erstere führen Schleifen oder Konditionierungen aus
- ▶ Letztere referenziert die Variable

```
{{ variable }}
```

```
{% for user in users %}  
  {{ user.name }} {{ user.password }}  
{% endfor %}
```


```
{% if user.name is defined %}  
  {{ user.name }} {{ user.password }}  
{% endif %}
```

- ▶ Kommentare in Templates: `{# ... #}`
- ▶ Layouting: `{%- ... -%}`


```
{# Das ist ein Kommentar #}
```

```
{%- for user in users %}  
  {{ user.name }} {{ user.password }}  
{% endfor -%}
```

Beispiel: /etc/hosts

 templates/hosts.j2

```
{% for server in groups.webservers %}  
  {{ hostvars[server].ansible_eth0.ipv4.address }} {{ server }}  
{% endfor %}
```

 main.yml

```
...  
- template: src=/etc/ansible/templates/hosts.j2 dest=/etc/hosts owner=root group=root mode ←  
  =0644 backup=yes  
...
```



## templates/pg\_hba.conf.j2

```
# TYPE DATABASE USER ADDRESS METHOD

# Default:
{% for connection in postgresql_pg_hba_default %}
{{connection.type}} {{connection.database}} {{connection.user}}
{{connection.address}}
{{connection.method}}
{% endfor %}

# Password hosts
{% for host in postgresql_pg_hba_passwd_hosts %}
host all all {{host}} password
{% endfor %}

# Trusted hosts
{% for host in postgresql_pg_hba_trust_hosts %}
host all all {{host}} trust
{% endfor %}

# User custom
{% for connection in postgresql_pg_hba_custom %}
{{connection.type}} {{connection.database}} {{connection.user}} {{connection.address}}
{{connection.method}}
{% endfor %}
```

## templates/vhosts.conf.j2

```
{% for vhost in apache_vhosts %}
<VirtualHost *:80>
  ServerName {{ vhost.servername }}
  DocumentRoot {{ vhost.documentroot }}
  {% if vhost.serveradmin is defined %}
    ServerAdmin {{ vhost.serveradmin }}
  {% endif %}
  <Directory "{{ vhost.documentroot }}">
    AllowOverride All
    Options -Indexes FollowSymLinks
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
{% endfor %}
```

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

---

Templates

Vault

Roles

Lookup

Strategies

Callbacks

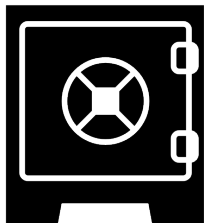
Performance

---

Windows

## Ansible Vault

- ▶ Verschlüsselung auf „Datei-Basis“
- ▶ Jede Datei kann verschlüsselt werden
- ▶ „Transparent“ für Ansible
- ▶ Seit Ansible 1.5
- ▶ Symmetrische AES-Verschlüsselung mittels Shared-Key aus Passwort



- ▶ Vault-Datei erstellen und editieren



```
ansible-vault create foo.yml
```



```
ansible-vault edit foo.yml
```

- ▶ Verschlüsseln, entschlüsseln und neu verschlüsseln



```
ansible-vault encrypt foo.yml
```



```
ansible-vault decrypt foo.yml
```



```
ansible-vault rekey foo.yml
```



- ▶ Parameter `--ask-vault-pass` fragt am Prompt nach dem Passwort



```
ansible-playbook --ask-vault-pass main.yml
```

- ▶ Passwort kann auch in einer einzelnen Text-Datei abgelegt werden
- ▶ Seit Ansible 1.7 kann auch ein (ausführbares) Skript aufgerufen werden, welches das Passwort in die Standard-Ausgabe (stdout) schreibt



```
ansible-playbook main.yml --vault-password-file ~/.vault_pass. ↵  
txt
```



```
ansible-playbook main.yml --vault-password-file ~/scripts/ ↵  
vault_pass
```

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

---

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

---

Windows

## Roles

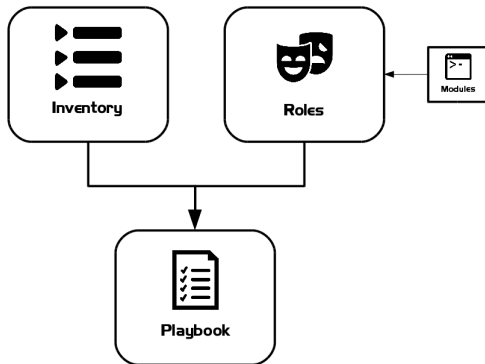
- ▶ Roles fassen Tasks / Dateien / Templates / Variablen... zusammen
- ▶ Roles bestehen aus einem Verzeichnisbaum
- ▶ Roles haben einen eindeutigen Namen
- ▶ Hostgroups können verschiedene Rollen gleichzeitig spielen
- ▶ Roles können Abhängigkeiten zu anderen Rollen haben



 Wiederverwendbar

 Organisation

 Teilen mit der Community






# Wie sieht eine Role aus?

- ▶ Unterverzeichnis roles
- ▶ Weitere Unterverzeichnisse

```
roles/  
  webserver/  
    defaults/main.yml  
    files/...  
    templates/...  
    tasks/main.yml  
    handlers/main.yml  
    vars/main.yml  
    meta/main.yml  
    tests/inventory,test.yml
```


# Wie können Roles zugewiesen werden?

- ▶ Unterverzeichnis roles
- ▶ Weitere Unterverzeichnisse

 provision.yml

```
- hosts: backend
  roles:
    - common
    - { role: webserver, hostname: 's0.meine-seite.de',    ↵
      dir: '/var/www/htdocs/s0' }
    - { role: webserver, hostname: 's1.meine-seite.de',    ↵
      dir: '/var/www/htdocs/s1' }
```

- ▶ „Zentraler Anlaufpunkt“: `roles/role/tasks/main.yml`
- ▶ Dort können Templates / Dateien ... ohne Pfadangabe genutzt werden, sofern sie im entsprechenden Verzeichnis liegen

 `roles/webserver/tasks/main.yml`


```
- name: install apache
  apt: name=apache2-mpm-prefork state=present
- name: delete default Apache vhost
  file: path=/etc/apache2/sites-enabled/000-default  ←
       state=absent
  notify: restart Apache
- name: generate vhosts
  template:
    src=vhost.conf
    dest=/etc/apache2/sites-enabled/{{ item.hostname }}.  ←
```

- ▶ Standard-Variablen-Definition können in `defaults/main.yml` gesetzt werden
- ▶ Haben die niedrigste Priorität in der Rangfolge

1. Extra-Variablen (-e in der Kommandozeile) haben die höchste Priorität
2. Verbindungs-Variablen im Inventory
3. Variablen in Playbooks, Rollen-Variablen u.s.w.
4. Variablen im Inventory
5. Facts
6. Role-Defaults haben die niedrigste Priorität


- Eine in vars Definierte Variable ist wenn Sie in einer Abhängigen Rolle Definiert ist auch in der Aufrufende Rolle verfügbar.

- ▶ Roles können Abhängigkeiten zu anderen Roles haben
- ▶ Abhängigkeiten werden in `roles/role/meta/main.yml` definiert
- ▶ Abhängigkeiten werden immer aufgelöst, d.h. in Abhängigkeit stehende Roles werden a priori ausgeführt

 `roles/common/meta/main.yml`


```
---
dependencies:
  - { role: common, some_parameter: 3 }
  - { role: apache, port: 80 }
  - { role: postgres, dbname: customers,      ←
      other_parameter: 12 }
```

- ▶ Abhängigkeiten werden immer nur einmal aufgelöst
- ▶ Falls eine weitere in Abhängigkeit stehende Rolle diesselbe Abhängigkeit definiert, so wird diese nur **einmal** ausgeführt
- ▶ Dieses Verhalten kann mit dem Schlüsselwort `allow_duplicates` beeinflusst werden


 `roles/common/meta/main.yml`

```
---
allow_duplicates: true
dependencies:
  - { role: common, some_parameter: 3 }
  - { role: apache, port: 80 }
  - { role: postgres, dbname: customers,      ←
      other_parameter: 12 }
```



 roles/bike/meta/main.yml

```
---  
dependencies:  
  - { role: wheel, n: 1 }  
  - { role: wheel, n: 2 }
```

 roles/wheel/meta/main.yml

```
---  
allow_duplicates: yes  
dependencies:  
  - { role: tire }  
  - { role: brake }
```

- ▶ Eine in vars Definierte Variable ist wenn Sie in einer Abhängigen Rolle Definiert ist auch in der Aufrufende Rolle verfügbar.
- ▶ Dies ermöglicht es uns diese Variablen auch als Schalter in der aufrufenden Rolle zu verwenden.

## ansible-galaxy

```
ansible-galaxy init <NAME>  
ansible-galaxy search <NAME>  
ansible-galaxy install -p . <NAME>
```

- ▶ Alle in der Rolle verwendete Variablen sollten sinnvoll in den defaults definiert sein.
- ▶ Die main.yml in den Tasks sollte wenn es sinnvoll ist aufgeteilt werden.
- ▶ Handler sollte ein sinnvolles Namensschema haben.

```
<rolle>/tasks/
```

```
main.yml
- include_tasks: install.yml
- include_tasks: config.yml
- include_tasks: firewall.yml
install.yml
config.yml
firewall.yml
```

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

---

Windows



Lookup plugins dienen dazu externe Datenquellen anzusteuern.

```
ansible-doc -t lookup -l
```

config	Lookup current Ansible configuration values
file	read file contents
fileglob	list files matching a pattern
filetree	recursively match all files in a directory tree
hashi_vault	retrieve secrets from HashiCorp's vault
inventory_hostnames	list of inventory hosts matching a host pattern
lastpass	fetch data from lastpass
passwordstore	manage passwords with passwordstore.org's pass utility
template	retrieve contents of file after templating with Jinja2

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

---

## Windows

Mittels den Strategie Option lässt sich das Verhalten ändern wie Ansible Tasks behandelt.

```
- hosts: all
  strategy: linear
  tasks:
    ...
```

```
ansible-doc -t strategy -l
```

	Executes tasks
debug	in interactive debug session.
free	on each host independently
host_pinned	on each host without interruption.
linear	in a linear fashion (Default)

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance



---

Windows

- ▶ Callback-Plugins kontrollieren, wie Ansible auf Ereignisse reagiert
- ▶ die Ausgabe von Ansible selbst ist ein Callback-Plugin
- ▶ es existieren unterschiedliche Plugins (Foreman, HipChat, Mail, ...)
- ▶ `https://github.com/ansible/ansible/tree/devel/lib/ansible/plugins/callback`
- ▶ durch Eigententwicklungen können direkt CMDBs mit aktuellen Informationen beliefert werden

- ▶ Aktivieren über `callback_plugins` Verzeichnis
- ▶ Konfiguration in der Ansible-Konfigurationsdatei
- ▶ Standard-Plugins müssen über die Whitelist freigeschaltet werden

```
ansible-doc -t callback -l
```

<code>debug</code>	formatted stdout/stderr display
<code>default</code>	default Ansible screen output
<code>jabber</code>	post task events to a jabber server
<code>unixy</code>	condensed Ansible output
<code>yaml</code>	yaml-ized Ansible screen output
<code>stderr</code>	Splits output, sending failed tasks to stderr
<code>timer</code>	Adds time to play stats

```
callback_whitelist = timer
```

```
Playbook run took 0 days, 0 hours, 0 minutes, 3 seconds
```

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

Callbacks

Performance

---

Windows

[https://docs.ansible.com/ansible/latest/reference\\_appendices/config.html](https://docs.ansible.com/ansible/latest/reference_appendices/config.html)

```
[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s -o PreferredAuthentications=publickey
pipelining = true

[defaults]
# Default: forks = 5
forks = 20
# Default: internal_poll_interval = 0.001
internal_poll_interval = 0.0001
```



`https://docs.ansible.com/ansible/latest/user\_guide/  
playbooks\_intro.html#ansible-pull`

Rund um Ansible

Ansible unter Suse

Inventory

Modules

YAML

Playbooks

Templates

Vault

Roles

Lookup

Strategies

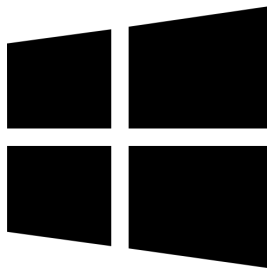
Callbacks

Performance

## Windows

## Windows

- ▶ Managementsystem mit Linux
- ▶ WinRM statt SSH
- ▶ Eigene Module
- ▶ Accounts, Gruppen
- ▶ Dienste, MSI-Pakete
- ▶ setzt Konfiguration von Windows voraus (Powershell, WinRM aktivieren)



Vielen Dank für Ihre  
Aufmerksamkeit.