



Ansible_

Orchestrierung



Recap

Konfigurationsmanagement

Konfiguration von Systemen - Infrastructure as Code

Verteilung und Orchestrierung von Software

Orchestrierung von Systemen

Zero-Downtime Updates

Ad-hoc Kommandos

Operation am offenen Herzen

Orchestrierung

Aufteilung in Host-Gruppen möglich

Automatisierung nutzen

Konfigurationsunterschied vermeiden

Änderungen deployen, testen und ausrollen?

Orchestrierung - Staging vs. Produktion

Aufteilung der inventory-Dateien

Nutze `-i inventory_stage / -i inventory_prod`

Umgebungsspezifische Variablen?

Produktion

```
[webservers]
10.10.50.12
10.10.50.13

[appservers]
10.10.51.11

[prod:children]
webservers
app servers
```

Staging

```
[webservers]
10.10.50.12

[appservers]
10.10.51.11

[stage:children]
webservers
app servers
```

Orchestrierung - Staging vs. Produktion

Unterschiedliche Aufteilungen möglich

```
Playbook.yml
group_vars/
  stage/
    all/
  prod/
    all/
inventories/
  inventory_prod
  inventory_stage
```

```
Playbook.yml
prod/
  inventory
  group_vars/
    all/
stage/
  inventory
  group_vars/
    all/
```

Updates

Ansible ist für Multi-Tier Deployments Designed

Möglichkeiten für Zero-Downtime Updates implementiert

- Loadbalancer ansprechen

- Monitoring informieren

Batching und Host Delegation möglich

Orchestrierung - Rollout

Batch Size

- name: test play
hosts: webserver
serial: 3
- name: test play
hosts: webserver
serial: "30%"
- name: test play
hosts: webserver
serial:
 - "10%"
 - "20%"
 - "100%"

http://docs.ansible.com/ansible/latest/playbooks_delegation.html#rolling-update-batch-size

Maximale Fehleranteile

```
- hosts: webserver  
  max_fail_percentage: 30  
  serial: 10  
  
  any_errors_fatal: True
```

Delegation

- hosts: webserver
serial: 5
- tasks:
 - name: take out of load balancer pool
command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
 - delegate_to: 127.0.0.1
 - name: actual steps would go here
yum: name=acme-web-stack state=latest
 - name: add back to load balancer pool
command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
 - delegate_to: 127.0.0.1

Einmalige Ausführung

- command: /opt/application/upgrade_db.py
run_once: true
- command: /opt/application/upgrade_db.py
run_once: true
delegate_to: web01.example.org
- command: /opt/application/upgrade_db.py
when: inventory_hostname == webservers[0]

Default: lineare Ausführung

- hosts: all
- strategy: free
- tasks:

Testing

Änderungen vor Rollout zunächst Testen - Fail Fast

Aufteilung in Multistages ist Vorbereitung dazu

Testing ist die Grundlage für sinnvoll Continuous Deployment

Durch Versionierung können Änderungen rückgängig gemacht werden

Orchestrierung - Testing: Richtiges Level

Ansible: Modelliert erwarteten Status

Keine Tests notwendig für

Service gestartet

Package installiert

Nicht checken ob User existiert → User module nutzen

Orchestrierung - Testing: Check Mode

Testen, ob Änderungen notwendig sind

```
ansible-playbook playbook.yml --check
```

Tasks überspringen: Conditionals nutzen

- name: this task will be skipped in check mode
git: repo=ssh://git@github.com/mylogin/hello.git dest=/home/mylogin/hello
when: not ansible_check_mode

Orchestrierung - Testing: Module

Port

tasks:

```
- wait_for: host={{ inventory_hostname }} port=22  
  delegate_to: localhost
```

Host, der den Task ausführen soll

Web Service

tasks:

- action: uri url=http://www.example.com return_content=yes
register: webpage
- fail: msg='service is not happy'
when: "'AWESOME' not in webpage.content"

Skript

tasks:

- script: test_script1
- script: test_script2 --parameter value --parameter2 value

Return-Wert != 0 → Fehler

Assert

tasks:

- shell: /usr/bin/some-command --parameter value
register: cmd_result
- assert:
 - that:
 - "'not ready' not in cmd_result.stderr"
 - "'gizmo enabled' in cmd_result.stdout"

Orchestrierung - Testing: Lifecycle

Für jede Umgebung dieselben Playbooks verwenden

Dev mit zusätzlichen Testing-Playbooks

Staging- ist der Produktionsumgebung identisch

Integration-Tests in Staging (wahrscheinlich kein Ansible - Selenium ...)

Deploy in Produktion

Orchestrierung - Continuous Deployment

Automatisierung der Erstellung lokaler Development VMs

CI System (Jenkins ...) : bei Änderungen Staging Umgebung updaten

Testing Skripte - Pass/Fail testen bei jedem Build

Alles erfolgreich - dasselbe Skript gegen Produktion