Hilfe bei der Automatisierung – warum Ansible für DevOps eine gute Wahl ist

Simon Hahn, OPITZ CONSULTING Deutschland GmbH

„DevOp DBA“ (also Developer und Administratoren) sind in der heutigen Zeit in aller Munde, wenn es um neuartige Technologien geht, die in den Systembetrieb eingebunden werden müssen. Zu ihren Aufgaben gehört es, System-Landschaften zu installieren, zu konfigurieren, zu überwachen und, ganz wichtig, zu automatisieren – dem Kunden also eine maßgeschneiderte Software-Lösung schnellstmöglich zur Verfügung zu stellen.

Dieser Artikel gibt einen kurzen Einblick in Ansible, das als Tool zur Automatisierung und Standardisierung von Software-Installationen immer mehr an Bedeutung gewinnt. Einfache Anwendungsbeispiele bis hin zur Skizze einer automatisierten Oracle-Datenbank-Installation zeigen, welche Vorteile automatisierte Vorgänge in der IT haben können.

In welcher Hinsicht spielt ein mächtiges und relativ neues Automatisierungs- und Orchestrierungstool eine wichtige Rolle für den heutigen DBA? Anhand eines Automatisierungstasks lässt sich gut darlegen, wie Ansible die tägliche Arbeit erleichtern kann und wie einfach es ist, mit diesem Tool eine Rolle und ein komplettes Playbook zu schreiben.

Ansible ist ein Konfigurationsmanagementtool, das in Python geschrieben ist. Kenntnisse in Python müssen aber nicht

zwingend vorhanden sein. Ansible verfügt über eine allgemeine, leicht verständliche Syntax sowie einfache Funktionen und kann auf einem Linux- oder Mac-System installiert sein. Oft reicht ein „yum install ansible“, „apt-get install ansible“ oder „brew install ansible“ und es ist sofort einsatzbereit. Dieser Artikel betrachtet Ansible in Version 2.1.0.0. Nähere Infos zur Installation siehe *„http://docs.ansible.com/ansible/intro_installation.html“*. Derzeit kann Ansible nicht direkt auf Windows als Hostsystem installiert sein, aber es kann genutzt werden, um Windows-Systeme zu konfigurieren.

Das Programm ist schlank, übersichtlich, flexibel und, wie schon gesagt, einfach zu lesen *(siehe Abbildung 1)*. In Ansible können Playbooks und Rollen geschrieben werden, die zum Beispiel eine vorher immer wiederkehrende, manuelle Aufgabe automatisieren, oder es werden Software „Deployments“ in unterschiedlichen Abhängigkeiten automatisiert auf unterschiedliche Server ausgerollt.

Zur Erläuterung: Playbooks sind zusammengefasste Anweisungen mehrerer Kommandos, die auf Remote-Systemen ausgeführt werden. Ein Playbook ruft eine oder mehrere Rollen auf. Rollen wiederum sind speziellere Teilbereiche innerhalb eines Playbooks. Playbooks und Rollen bestehen jeweils aus mehreren Tasks, die einzeln angestoßen werden können. Ein Task ist vergleichbar mit einem Ausführungsplan, der sequentiell innerhalb der Rolle abgearbeitet wird.

Ein Tool für alle Fälle

Bei der Anwendung von Ansible ist es fast egal, welches Betriebssystem auf dem Server installiert ist. Dass das Tool aus der Linux-Welt kommt und dort verstärkt unterstützt wird, ist dabei nicht entscheidend. Obwohl die einzelnen Schritte (Tasks) seriell abgearbeitet werden, kann eine logische Parallelisierung erfolgen, zum Beispiel, um Aufgaben auf vielen Servern gleichzeitig durchzuführen. Als Standard benutzt Ansible SSH (Secure Shell), um sich mit dem Zielsystem zu verbinden; sobald der „Public SSH Key“ auf dem Server hinterlegt ist (zu finden im Verzeichnis „~/.ssh/authorized_keys“), kann das Zielsystem von Ansible bearbeitet werden. Eine Installation von Client-Komponenten oder Agenten ist nicht erforderlich. Das Einsatzgebiet reicht von simplen über vielfältige bis hin zu komplexen Anwendungsfällen.

Ansible kann direkt ein Modul ausführen, zum Beispiel „setup“, wobei „–m“ dafür sorgt, dass das Modul „setup“ aufgerufen wird. Eine Liste aller Module steht unter *„http://docs.ansible.com/ansible/list_of_all_modules.html“*. Das Kommando „root@adminbox:~/git/ansible-oracle# ansible servername -m setup --tree /tmp/facts“ zeigt alle Facts auf dem Server, die später verwendet und abgefragt werden können. Der Servername kann vorher auch unter „host_vars“ definiert sein. Jegliche Ansible-Rollen oder -Playbooks sind im „Yaml“-Format erstellt.

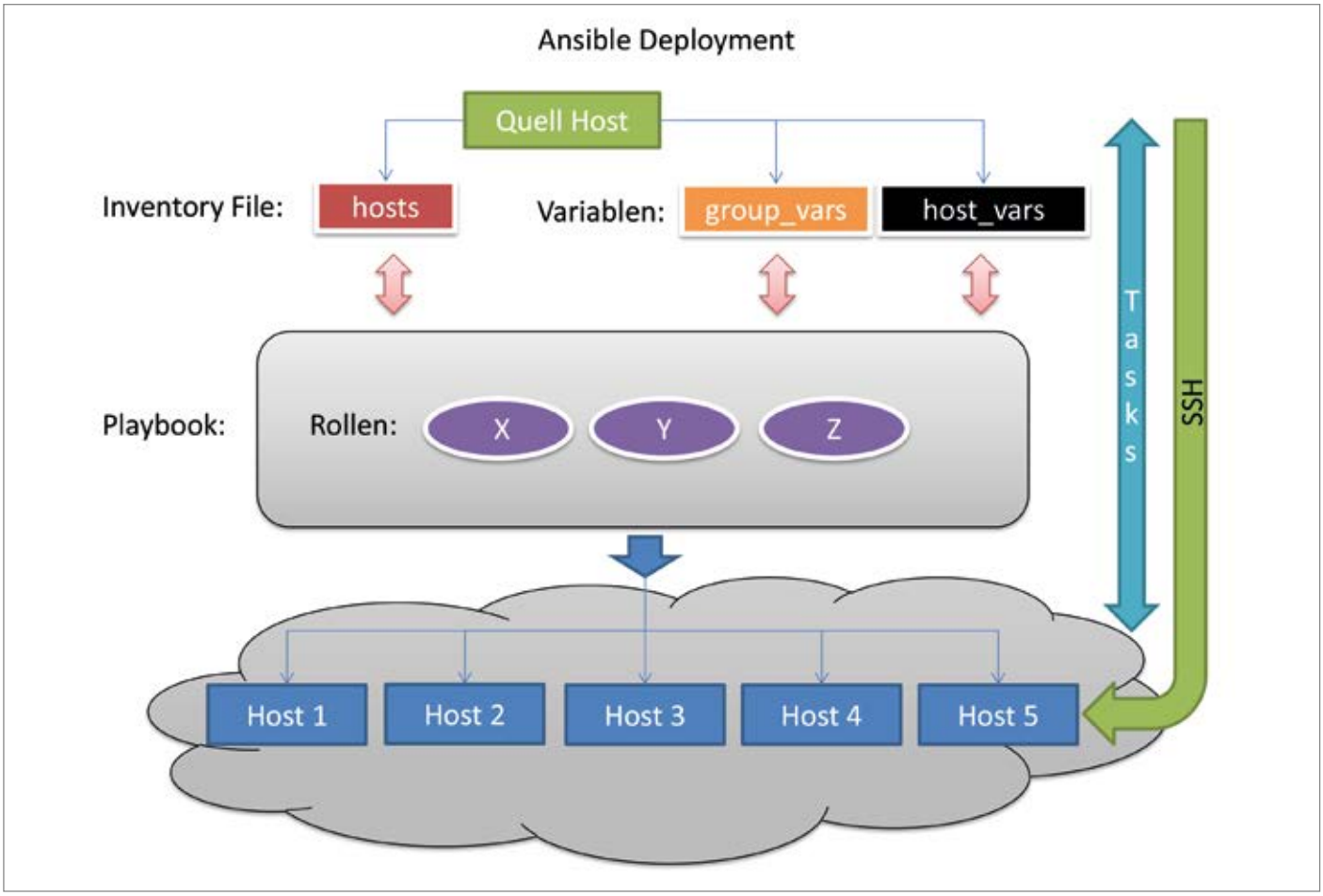

Abbildung 1: Ansible im Überblick

Um sicherzustellen, dass der Server überhaupt per SSH erreichbar ist, führt man ein Skript auf der Kommandozeile aus *(siehe Listing 1)*. Der Server antwortet mit „pong“. *Listing 2* zeigt ein simples Kommando auf allen Servern, die bekannt sind. Der erste Server ist nicht erreichbar, der zweite gibt die Uptime des Servers zurück. So lassen sich komfortabel Server überprüfen und warten.

Scripting für Playbook und Rollen

In der „Yaml“-Syntax muss auf den korrekten Abstand in den definierten „Yaml“-Dateien, nämlich zwei Leerzeichen, besonders geachtet werden. Andernfalls kommt es zu Syntaxfehlern, wenn das Playbook aufgerufen wird. Am besten werden diverse Plug-ins wie „ansible-linter“ etc. für den entsprechenden Editor installiert, um die Syntax auf Fehler zu überprüfen. Als Editor eignen sich Atom oder Sublime, obwohl es auch andere gute Alternativen gibt, wie unter anderem Eclipse *(siehe „https://atom.io“, „https://www.sublimetext.com/3“ und „https://eclipse.org“)*.

Über „– hosts:“ wird die dbserver-Gruppe aus dem Inventory File aufgeru-

```
root@adminbox:~/git/ansible-oracle# ansible <servername> -m ping
<servername> | SUCCESS => {
   "changed": false,
   "ping": "pong"
}
```

Listing 1

```
root@adminbox:~/git/ansible-oracle# ansible all -a „uptime“
<servername_test>| UNREACHABLE! => {
   "changed": false,
   "msg": "SSH Error: data could not be sent to the remote host. Make
sure this host can be reached over ssh",
   "unreachable": true
}
<servername> | SUCCESS | rc=0 >>
 15:12:29 up 49 min, 1 user, load average: 0,00, 0,01, 0,05
```

Listing 2

```
root@adminbox:~/git/ansible-oracle# cat ~/git/ansible-oracle/roles/oracle_sqlscripts/tasks/main.yaml

---
   # Copy all Sqlscripts to the server

   # create directory /home/oracle/CREATE
   - name: Oracle Directory /home/oracle/CREATE
       file: path={{ ora_home }}/CREATE state=directory owner={{ ora_user }} group={{ oinstall_group }}
mode=0770
       tags: create_directory_home_oracle_CREATE

   # create directory /home/oracle/CREATE/SQL
   - name: Oracle Directory /home/oracle/CREATE/SQL
       file: path={{ ora_home }}/CREATE/SQL state=directory owner={{ ora_user }} group={{ oinstall_group }}
mode=0770
       tags: create_directory_home_oracle_CREATE_SQL

   # create directory /home/oracle/sqlscripts
   - name: Oracle Directory /home/oracle/sqlscripts
       file: path={{ ora_home }}/sqlscripts state=directory owner={{ ora_user }} group={{ oinstall_group }}
mode=0770
       tags: create_directory_home_oracle_sqlscripts

   # copy files
   - name: copy SQL Scripts files
       copy: src={{ item }} dest={{ ora_bin_dir }} owner=oracle group=dba mode=640
       with_fileglob:
           - ../files/*.sql
           - ../files/*.sh
       tags: sqlscripts

# copy template scripts
   - name: copy SQL CREATE Scripts
       template: src={{ item }} dest={{ ora_home }}/CREATE/SQL owner=oracle group=dba mode=640
       with_fileglob:
           - ../templates/*.sql.j2
       tags: sqlscripts
```

Listing 3

fen, das zuvor definiert wurde. In dieser Gruppe befinden sich die Systeme, die zugeordnet sind. Unter „roles:“ wird die Rolle „oracle_sqlscripts“ aufgerufen, die in der Ansible-Struktur schon angelegt worden ist, die im Verzeichnis „~/git/ansible-oracle/roles/oracle_sqlscripts/tasks/main.yaml“ aber noch mit Inhalt befüllt werden muss. Hier liegt der eigentliche Code, der ausgeführt werden soll, wenn die Rolle aufgerufen wird. *Listing 3* zeigt die Rolle „oracle_sqlscripts“.

File-Management automatisiert steuern

In der gezeigten „oracle_sqlscripts“-Rolle sind drei Verzeichnisse erstellt. Alle Dateien, die im Verzeichnis „files“ mit „*.sql“ und „*.sh“ enden, werden über eine For-Schleife mit der Funktion „with_fileglob“ auf den entsprechenden Server in das Verzeichnis „/home/oracle/sqlscripts“ kopiert. Die Benutzerrechte sind dort definiert.

Der Unterschied im zweiten „copy template scripts“-Skript liegt im Modul „template“, das die Dateien aus dem Verzeichnis „templates“ auf den Server kopiert. Der Unterschied zu „Copy“ ist, dass die Dateien zuvor interpretiert werden.

Beim Template können Variablen innerhalb der Skripte mit Jinja2 definiert sein, die von Ansible interpretiert werden. Das „Tag“ dient dazu, den Aufruf auf der Kommandozeile einzeln nachzusteuern. So wird aus der Rolle „oracle_sqlscripts“ nur der einzelne Task mit dem definierten „Tag“ ausgeführt *(siehe Listing 4)*. Variablen werden innerhalb der „Yaml“-Files mit „{{ Variablenname }}“ aufgerufen. Diese werden dann vorher in den „host_vars“ oder „group_vars“ eingebunden beziehungsweise definiert oder direkt in „vars“ für die entsprechende Rolle gesetzt. *Listing 5* zeigt ein Beispiel für ein File im Verzeichnis „host_vars“.

Die in „group_vars“ bestehende „dbserver.yaml“ wird ähnlich aufgebaut. Grundsätzlich gilt: Host-Variablen haben eine höhere Priorität als Group-Variablen und überschreiben diese. Um die SQL-Skripte über die Rolle „oracle_sqlscripts“ auszurollen, wird nun der Befehl „root@adminbox:~/git/ansible-oracle# ansible-playbook oracle_sqlscripts.yml“ ausgeführt. Dabei wird der Servername weggelassen, weil dieser direkt über die „Yaml“-Datei „oracle_sqlscripts.yml“ definiert ist.

Die Inventory-Datei „hosts“ fehlt ebenfalls, weil sie in der allgemeinen „ansible.cfg“ definiert worden ist, die aber auch unter Angabe „-i <InventoryFile>“ definiert aufgerufen werden kann. Es können also auch unterschiedliche Inventory Files angelegt werden. *Listing 6* zeigt ein Beispiel für „ansible.cfg“. Weitere Einstellungen sind in der offiziellen Ansible-Dokumentation zu finden.

Komfortables Log-Management

Sobald das Modul auf dem Server ausgerollt wurde, werden diverse Tasks abgear-

```
root@adminbox:~/git/ansible-oracle# ansible -m ping singledb1
singledb1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
root@adminbox:~/git/ansible-oracle# ansible -m ping singledb2
 [WARNING]: provided hosts list is empty, only localhost is available

root@adminbox:~/git/ansible-oracle# ansible -m ping singledb3
singledb3 | UNREACHABLE! => {
    "changed": false,
    "msg": "SSH Error: data could not be sent to the remote host. Make sure this host can be reached over ssh",
    "unreachable": true
}
```

Abbildung 2: Das Ergebnis-Log

beitet. Die Ergebnisse landen in einem Ergebnis-Log direkt in der Shell. Grün gleich „ok", Rot gleich „failed", Gelb gleich „changed" *(siehe Abbildung 2)*. Wenn ein „ansible.log"-File in der „ansible.cfg" definiert wurde, kann jederzeit im Logfile nachgeschaut werden, wie der „ansible run" gelaufen ist. Durch das Ausrollen der Ansible-Rolle werden die Python-Pakete direkt per SSH im Temp-Ordner „/tmp" auf dem Server abgelegt und im Anschluss von Python entpackt und interpretiert. Eine detaillierte Ausgabe wird mit „ansible-playbook oracle.yml --tags oracle_sqlscripts -vvvv" erreicht. Dieses Beispiel zeigt einen großen Vorteil von Ansible sehr deutlich: Auf allen zuvor festgelegten Servern wird die gleiche Struktur der zuvor erstellten Rolle „oracle_sqlscripts" abgebildet, die Skripte werden standardisiert in den vorgesehenen Verzeichnissen auf den Servern zur Verfügung gestellt.

```
root@adminbox:~/git/ansible-oracle# ansible-playbook -i ~/git/
ansible-oracle/hosts oracle_sqlscripts --tags sqlscripts
```

Listing 4

```
root@adminbox:~/git/ansible-oracle/host_vars# cat servername.yaml
---
# spezifische Einstellungen/Variablen pro Host
ora_home: /home/oracle
ora_bin_dir: /home/oracle/sqlscripts
ora_user: oracle
oinstall_group: oinstall
.
.
```

Listing 5

```
root@adminbox:~/git/ansible-oracle# cat ansible.cfg
[defaults]
log_path = ./ansible.log
hostfile = ~/git/ansible-oracle/hosts
nocows = 1
host_key_checking = False

[ssh_connection]
pipelining = True
control_path = /tmp/ansible-ssh-%%h-%%p-%%r
scp_if_ssh = True
```

Listing 6

Fazit

Vorteile von Ansible sind die enorme Flexibilität, die automatische Standardisierung und ein einfacher Versionswechsel zu einer neueren Ansible-Version. Zeitintensives Migrieren entfällt dadurch. Die sehr gute Lesbarkeit und die Definition von sequentiellen Tasks helfen dem Anwender, sich auch in fremde, komplexe Ansible-Projekte einzuarbeiten.

Weitere Eindrücke über sehr interessante Ansible-Projekte für die unterschiedlichsten Anwendungsfälle finden sich direkt auf *„https://github.com"*, seien es automatisierte Installationen von MySQL, Postgres, Docker oder Betriebssystemen.

Wichtige Links für Oracle-DBAs

- *https://github.com/oravirt/ansible-oracle*
- *https://fritshoogland.wordpress.com/2016/08/02/a-total-unattended-install-of-linux-and-the-oracle-database*

Simon Hahn
simon.hahn@opitz-consulting.de