Wie kann Ansible-Automatisierung das Leben von DBAs verbessern?

Nicolas Penot, dbi services

Ich liebe IT-Automatisierung. Meiner Ansicht nach ermöglicht die Automatisierung die Entwicklung von IT-Umgebungen. Automatisieren Sie Ihre sich wiederholenden Aufgaben, und Sie werden Zeit sparen, um an wichtigen Aufgaben arbeiten zu können: an Aufgaben, die den Wert Ihres IT-Ökosystems exponentiell steigern werden. Es macht außerdem Ihren Arbeitsalltag interessanter: Lernen, Programmieren, Testen, Automatisieren und etwas Neues lernen.

Welches Automatisierungswerkzeug passt also zu Ihrem Bedarf? Es gab eine Zeit, in der Bash-Skripte mein Lieblingswerkzeug für diese Aufgabe waren. Einfach zu programmieren und einfach zu implementieren. Bash ist auch in fast alle Umgebungen integriert. Damit ist es breit verfügbar und portabel. Für mich funktioniert es immer noch gut. Wenn jedoch verschiedene Leute an einer gemeinsamen Infrastruktur arbeiten, wird es nützlich, einige Standards zu haben. Eine vorhersagbare Codierungsstruktur, die leicht zu lesen und zu pflegen ist. Nachdem ich Ansible entdeckt habe, muss ich zugeben, dass es ein ernstzunehmendes Tool für die IT-Automatisierung ist.

Dieser Artikel stellt Ansible mit einigen Beispielen rund um GoldenGate vor. Wie können wir eine Automatisierung für ein Oracle-Produkt wie beispielsweise GoldenGate programmieren?

Anwendungsfall mit GoldenGate

In diesem Beispiel verfüge ich über drei Server. Zwei Server hosten eine Oracle-Datenbank und die GoldenGate-Binärdateien. Der dritte Server dient als Ansible-Host, von dem aus ich meine Ansible-Aufgaben ausführen werde *(siehe Abbildung 1)*:

Ansible

Aus meiner Sicht sind dies die drei Punkte, um Ansible und seinen Nutzen schnell zu verstehen:

- **Kein Agent:** Ansible erfordert keine Installation von Agenten auf Ihren Zielservern. Stattdessen verbindet es sich

über das gesicherte SSH-Protokoll, um seine Aufgaben auszuführen.

- **Fakten:** Fakten sind ein Satz von Variablen, den Ansible zur Laufzeit auf den Zielrechnern sammelt. Diese Variablen können dann in Ihren Skripten verwendet werden. Fakten sind fast alle Informationen, die Sie von einem Zielhost benötigen, wie beispielsweise IPs, NICs, Geräte usw. Sie können sogar Ihre Fakten hinzufügen, wie etwa die Liste der Oracle-Instanzen, die mit Ihrem Oracle Home laufen *(siehe Abbildung 2)*.

Zwei Stufen:

- Sie können das sogenannte Ad-hoc-Befehlszeilenwerkzeug verwenden. Mit diesem Tool können Sie mit einer Befehlszeile Aktionen wie zum Beispiel das Erstellen von Betriebssystembenutzern auf mehreren Servern ausführen.
- Dann können Sie eine Reihe von Operationen skripten. Zu diesem Zweck werden Sie ein Playbook verwenden. Ein Playbook ist eine Datei, die Ihre Abfolge von Operationen im YAML-Format enthält.

Installation

Sie kennen jetzt die Grundlagen und sind bereit, sie selbst zu testen. Zuerst müssen Sie vermutlich Ansible installieren. Zu dem Zeitpunkt, an dem ich diesen Artikel schreibe, enthält die Ansible-Dokumentation die folgenden Anweisungen *(siehe Listing 1)*:

Nun müssen Sie über eine passwortlose SSH-Konnektivität vom Ansible-Host zu den Oracle-Servern verfügen. Das nachstehende Snippet wird auf dem Ansible-Host ausgeführt *(siehe Listing 2)*. Der erste Befehl erstellt einen RSA-Authentifizierungsschlüssel. Die nächsten drei Befehle installieren den öffentlichen Schlüssel in der Datei „remote authorized_keys".

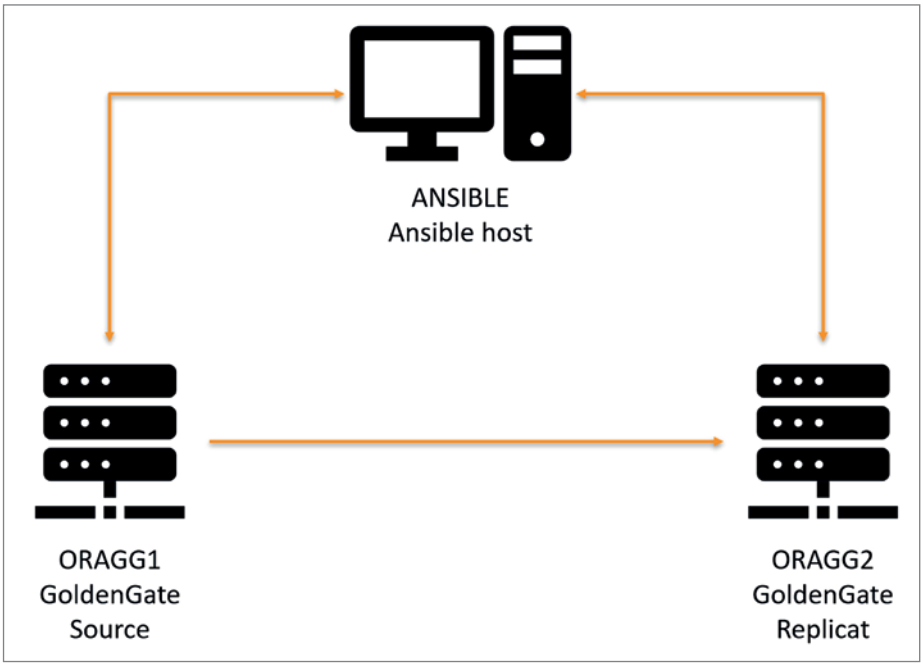

Abbildung 1 (Quelle: Nicolas Penot)

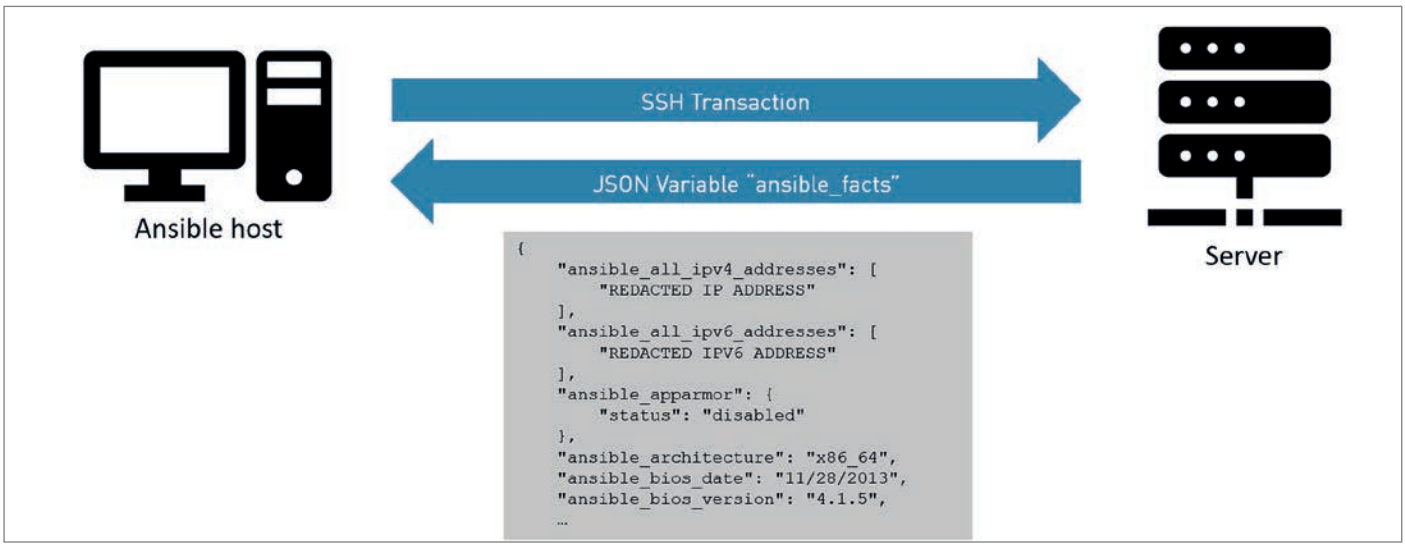

Abbildung 2 (Quelle: Nicolas Penot)

```
# CentOS / RHEL
$ yum install ansible
      #OR
# Ubuntu
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt-get install ansible
```

Listing 1

```
$ ssh-keygen -t rsa -C nicolas.penot@dbi-services.com
$ ssh-copy-id root@oragg1
$ ssh-copy-id root@oragg2
$ ssh-copy-id root@ansible
```

Listing 2

```
$ cat /etc/ansible/hosts    ## default location for the inventory file
[db_prod]
oragg[1:2]
[source]
oragg1
[replicat]
oragg2
[db_prod:children]
source
replicat
[db_prod:vars]
oracle_user=oracle
inventory_location=/u01/app/oraInventory
[jumphost]
ansible
```

Listing 3

```
$ ansible-inventory --graph
@all:
  |--@db_prod:
  |   |--@replicat:
  |   |   |--oragg2
  |   |--@source:
  |   |   |--oragg1
  |   |--oragg1
  |   |--oragg2
  |--@jumphost:
  |   |--ansible
  |--@ungrouped:
```

Listing 4

```
$ ansible db_prod --module-name command --args "hostname -f"
oragg2 | CHANGED | rc=0 >>
oragg2

oragg1 | CHANGED | rc=0 >>
oragg1
```

Listing 5

- copy - Copies files to remote locations
- cpanm - Manages Perl library dependencies.
- cpm_user - Get various status and parameters from WTI OOB and PDU devices
- cron - Manage cron.d and crontab entries

Abbildung 3 (Quelle: Nicolas Penot)

- azure - create or terminate a virtual machine in azure (D)

Abbildung 4 (Quelle: Nicolas Penot)

Ansible-Inventar

Das Inventar ist eine von Ansible verwendete Datei, um den Gruppennamen aufzulösen. Das ist ähnlich wie bei einer DNS, die aus einem Hostnamen eine IP-Adresse erhalten würde. Ansible verwendet das Inventar, um aus einer Gruppe eine Liste der Hosts zu erhalten. Dies liegt daran, dass Ansible die Hostgruppe als Parameter verwendet, um Aktionen auszuführen, statt die Hostnamen zu verwenden. Hier ist ein Beispiel *(siehe Listing 3)*.

Sie können den Befehl „ansible-inventory" verwenden, um zu sehen, was Ansible derzeit in seinem Inventar hat *(siehe Listing 4)*.

Es gibt zwei Gruppen: „db_prod" und „jumphost". Die Gruppe „db_prod" enthält zwei Untergruppen: eine mit den Servern, die als Extrahierung für GoldenGate verwendet werden, und eine mit den Hosts, die für die Replikation genutzt werden.

Das Ansible-Ad-hoc-Befehlszeilenwerkzeug

Der Ad-hoc-Befehl lautet „ansible". Im folgenden Beispiel verwende ich das Ansible-Modul „command", um auf jedem Server der Gruppe „db_prod" den Befehl „hostname -f" auszuführen *(siehe Listing 5)*.

Dann gibt Ansible einen Bericht über die Ausführung aus, einschließlich des Ergebnisses für den „hostname"-Befehl. Mit diesem einfachen Befehl können Sie sich bereits vorstellen, was Sie einfach und schnell tun können.

Jetzt lassen Sie vielleicht Ihre Fantasie spielen und Sie möchten eine komplexere Automatisierung erstellen. Gut, lassen Sie uns die Playbooks kennenlernen.

Ansible-Modul

Wie oben gesehen, habe ich die Option „--module-name" verwendet. Ansible funktioniert mit Modulen. Wir können Module mit Plug-ins vergleichen. Mit einem Modul kann Ansible Aktionen wie das Kopieren einer Datei, das Erstellen eines OS-Benutzers oder sogar das Erstellen einer Amazon-EC2-Instanz durchführen. Dies sind Beispiele für die mehr als 2000 Module, die es derzeit gibt *(siehe Abbildungen 3,4 und 5)*.

Hier ist zum Beispiel der Ansible-Befehl, mit dem ich auf allen Servern der Gruppe „db_prod“ einen neuen Benutzer namens „nicolas“ anlegen kann *(siehe Listing 6)*.

Module kombinieren, um komplexere Aktionen auszuführen: Anstatt alle Befehle nacheinander in ein Bash-Skript einfügen zu müssen, bietet Ansible das Konzept des Playbooks. Ein Playbook ist eine Datei im YAML-Format, in der Sie die Module auflisten, die auf einer Gruppe von Hosts nacheinander ausgeführt werden sollen. Dies ist ein Beispiel *(siehe Listing 7)*.

Dieses Playbook enthält vier Module. Das erste Modul „file“ wird verwendet, um ein Zuhause für unsere GoldenGate-Installation zu schaffen. Mit dem zweiten Modul „unarchive“ entpackt GoldenGate die Installationsdateien in das Verzeichnis „/u01/app/software/goldengate“. Das dritte Modul „fact“ erstellt eine Variable, die ich im nächsten Modul des aktuellen Playbooks verwenden kann. Und das letzte Modul – „command“ –, das wir bereits kennen, führt den sogenannten „Oracle runInstaller“ in der Kommandozeile aus.

Alle diese Module aus diesem Playbook werden in der Gruppe „db_prod“ ausgeführt. Dies ist in der dritten Zeile der YAML-Datei im Attribut „hosts“ zu sehen. Die „hosts“-Attribute beschreiben eine Gruppe von Hosts des Inventars und keine Hostnamenliste Ihrer Server.

Sie können das Playbook dann mit dem Befehl „ansible-playbook“ wie folgt ausführen *(siehe Listing 8)*.

Variablen

Im vorigen Snippet können wir auch Variablen in der YAML-Datei sehen. Die Variablen werden in doppelte geschweifte Klammern wie beispielsweise „{{ My_variable }}“ gesetzt. Diese Variablen werden während des Ausführens entsprechend der aktuellen Umgebung ersetzt. Daher kann das gleiche Playbook für jeden Server wiederverwendet werden. Die Werte, die sich zwischen den Servern unterscheiden – wie etwa die IP-Adressen –, können in Variablen umgewandelt werden, die sich entsprechend der Laufzeitumgebung ändern. Das erlaubt auch,

Abbildung 5 (Quelle: Nicolas Penot)

```
$ ansible db_prod --module-name user --args "name=nicolas group=dba"
oragg2 | CHANGED => {
   "changed": true,
   "comment": "",
   "create_home": true,
   "group": 54322,
   "home": "/home/nicolas",
   "name": "nicolas",
   "shell": "/bin/bash",
   "state": "present",
   "system": false,
   "uid": 54322
}
oragg1 | CHANGED => {
   "changed": true,
   "comment": "",
   "create_home": true,
   "group": 54322,
   "home": "/home/nicolas",
   "name": "nicolas",
   "shell": "/bin/bash",
   "state": "present",
   "system": false,
   "uid": 54322
}
```

Listing 6

```
$ cat p_goldengate_installation.yml
- name: "Play 1: GoldenGate installation"
   gather_facts: yes
   hosts: db_prod
   tasks:

   - name: "Creates GoldenGate home"
       file:
           path: "{{ goldengate_home }}"
           state: directory
           owner: oracle
           group: oinstall

   - name: "Unzip GoldenGate binaries"
       unarchive:
           src: "/u01/app/software/goldengate/fbo_ggs_Linux_x64_shipho-
me.zip"
           dest: "/u01/app/software/goldengate"
           owner: oracle
           group: oinstall

   - set_fact:
           installer_dir: "/u01/app/software/goldengate/fbo_ggs_Linux_
x64_shiphome/Disk1"

   - name: "Install GoldenGate"
       command: "{{ installer_dir }}/runInstaller -silent -waitforcom-
pletion INSTALL_OPTION=ORA12c SOFTWARE_LOCATION={{ goldengate_home }}
START_MANAGER=false INVENTORY_LOCATION={{ inventory_location }} UNIX_
GROUP_NAME=oinstall"
       become: yes
       become_user: oracle
```

Listing 7

```
$ ansible-playbook p_goldengate_installation.yml
```

Listing 8

```
$ cat p_goldengate_initial_load.yml
- name: "Play 1: Prepare Initial load on source"
  gather_facts: yes
  hosts: source
  tasks:

  - name: "Add extraction"
      include_role:
          name: goldengate
      vars:
          option: add_extraction

  - name: "Export schema {{ e_schema }} to {{ target_oracle_sid }}"
      include_role:
          name: goldengate
      vars:
          option: export_import_schema

- name: "Play 2: Configure replication on target"
  gather_facts: yes
  hosts: replicat
  tasks:

  - name: "Add replication to {{ target_oracle_sid }}"
      include_role:
          name: goldengate
      vars:
          option: add_replication
```

Listing 9

```
$ cat roles/goldengate/tasks/t_add_replication.yml
  - name: "Creates GG data directory"
      file:
          path: /u11/app/goldengate/data/{{ oracle_sid }}
          state: directory
          owner: oracle
          group: oinstall

  - name: "Copy repl{{ schema }} parameter file"
      template:
          src: "replicat.prm.j2"
          dest: "{{ goldengate_home }}/dirprm/repl{{ schema }}.prm"
          owner: oracle
          group: oinstall

  - name: "Copy GoldenGate script file"
      template:
          src: "add_replicat.j2"
          dest: "/tmp/add_replicat"
          owner: oracle
          group: oinstall

  - name: "Execute ggsci command"
      shell: "{{ goldengate_home }}/ggsci paramfile /tmp/add_replicat"
          become: true
          become_user: oracle
          environment:
              ORACLE_HOME: "{{ oracle_home }}"
              ORACLE_SID: "{{ oracle_sid }}"
              LD_LIBRARY_PATH: "{{ oracle_home }}/lib"
```

Listing 10

eine Variable anstelle aller Playbooks zu ändern, wenn Sie eine globale Variable ändern, wie beispielsweise das standardmäßige Oracle-Home.

Das für die Variablen verwendete Modell ist Jinja2. Jinja2 ist eine Bibliothek für Python, die von Ansible als Templating-Sprache genutzt wird. Das ermöglicht auch komplexere Strukturen wie bedingte Ersetzungen oder Schleifenersetzung *(mehr zu diesem Thema auf der Website http://jinja.pocoo.org/docs/2.10/)*.

Rollen

Stellen Sie sich vor, dass Sie ein Playbook erstellt hätten, das ein Home-Verzeichnis für Oracle sowie die OS-Benutzer und -Gruppen gemäß Ihren Standards erstellt, die OS-Kernelparameter konfiguriert und alle für Oracle-Software erforderlichen Oracle-Pakete installiert. Möchten Sie diesen Code vielleicht wiederverwenden oder wiederverwendbar machen?

Hierzu benutzt Ansible das Konzept der Rolle. Eine Rolle ist eine Reihe von Modulen, die Sie entwickelt haben und die in mehreren Playbooks wiederverwendet werden können. Wir können sie mit einer Funktion in einer Programmiersprache vergleichen.

Als Beispiel habe ich nachstehend ein Playbook, in dem nur Rollen aufgerufen werden, um GoldenGate erstmalig zu laden *(siehe Listing 9)*.

Dieses Playbook enthält drei Rollen. Zwei dieser Rollen werden in der Hostgruppe namens „source“ und die letzte Rolle in der Hostgruppe namens „replicat“ ausgeführt. Wie bereits gesagt: Rollen sind eine Reihe von Modulen, die im YAML-Format geschrieben sind. Dies ist beispielsweise die YAML-Datei der Rolle „GoldenGate“ mit der Option „add_replication“ *(siehe Listing 10)*.

Diese Rolle besteht aus 4 Ansible-Modulen: „file“, „template“, „template“ und „shell“.

Um dieses Playbook zu verwenden, muss ich zusätzliche Parameter hinzufügen, um die Datenbank anzugeben, auf der ich die Replikation installieren werde *(siehe Listing 11)*.

Dies ist das vollständige Ergebnis für unseren Anwendungsfall *(siehe Abbildung 6 und Abbildung 7)*.

```
$ ansible-playbook playbooks/p_goldengate_initial_load.yml \
-e "e_target_hostname=oragg2 e_target_oracle_sid=DB2 e_schema=soe"
```

Listing 11

```
PLAY [Play 1: Prepare Initial load on source] ************************************

TASK [Gathering Facts] ***********************************************************
ok: [oragg1]

TASK [Add extraction] ************************************************************

TASK [goldengate : include_tasks] ************************************************
included: /home/nico/ansible/roles/goldengate/tasks/./t_add_extraction.yml for oragg1

TASK [goldengate : Set facts] ****************************************************
ok: [oragg1]

TASK [goldengate : Creates GG data directory] ************************************
ok: [oragg1]

TASK [goldengate : Add supplemental log data] ************************************
changed: [oragg1]

TASK [goldengate : Copy extrsoe parameter file] **********************************
ok: [oragg1]

TASK [goldengate : Copy pumpsoe parameter file] **********************************
ok: [oragg1]

TASK [goldengate : Copy GoldenGate script file] **********************************
ok: [oragg1]

TASK [goldengate : Execute ggsci command] ****************************************
changed: [oragg1]

TASK [Export schema soe to DB2] **************************************************

TASK [goldengate : include_tasks] ************************************************
included: /home/nico/ansible/roles/goldengate/tasks/./t_export_import_schema.yml for orag

TASK [goldengate : Set facts] ****************************************************
ok: [oragg1]

TASK [goldengate : Get current SCN on source] ************************************
changed: [oragg1]
```

Abbildung 6 (Quelle: Nicolas Penot)

```
TASK [goldengate : Export schema soe] ********************************************
changed: [oragg1]

TASK [goldengate : Copy dumpfile soe.dmp to oragg2] ******************************
changed: [oragg1]

TASK [goldengate : Purge dumpfile /u01/app/oracle/admin/DB1/dpdump/soe.dmp] ******
changed: [oragg1]

TASK [goldengate : Check if schema soe exists] ***********************************
changed: [oragg1 -> oragg2]

TASK [goldengate : Create schema soe if not exists] ******************************
changed: [oragg1 -> oragg2]

TASK [goldengate : Check if tablespace for soe exists] ***************************
changed: [oragg1 -> oragg2]

TASK [goldengate : Create tablespace soe if not exists] **************************
changed: [oragg1 -> oragg2]

TASK [goldengate : Import schema soe] ********************************************
changed: [oragg1 -> oragg2]

TASK [goldengate : Purge dumpfile /u01/app/oracle/admin/DB2/dpdump/soe.dmp] ******
changed: [oragg1 -> oragg2]

PLAY [Play 2: Configure replication on target] ***********************************

TASK [Gathering Facts] ***********************************************************
ok: [oragg2]

TASK [Add replication to {{ target_oracle_sid }}] ********************************

TASK [goldengate : include_tasks] ************************************************
included: /home/nico/ansible/roles/goldengate/tasks/./t_add_replication.yml for oragg2

TASK [goldengate : Set facts] ****************************************************
ok: [oragg2]
```

Abbildung 7 (Quelle: Nicolas Penot)

Fazit

Auf einen Blick: Ansible verwendet Module, um Aufgaben über SSH auf Gruppen von Hosts auszuführen, die in einem Inventar definiert sind.

Wie wir in diesem Artikel gesehen haben, wird Ansible in wenigen Schritten Ihre Produktivität verbessern und Ihre Infrastruktur anpassen:

1. Ansible auf einem zentralen Host installieren
2. Erstellen eines Inventars Ihrer Server
3. Konvertieren sich wiederholender Aufgaben in Playbooks

Ich hoffe, dass Sie diesen Artikel nützlich fanden, um Ansible gut nachvollziehbar zu machen. Zögern Sie bitte nicht, uns zu kontaktieren, falls Sie weitere Informationen wünschen.

Nicolas Penot
penot@dbi-services.com