Red Stack
eArticle

How Ansible automation can improve DBA's life?

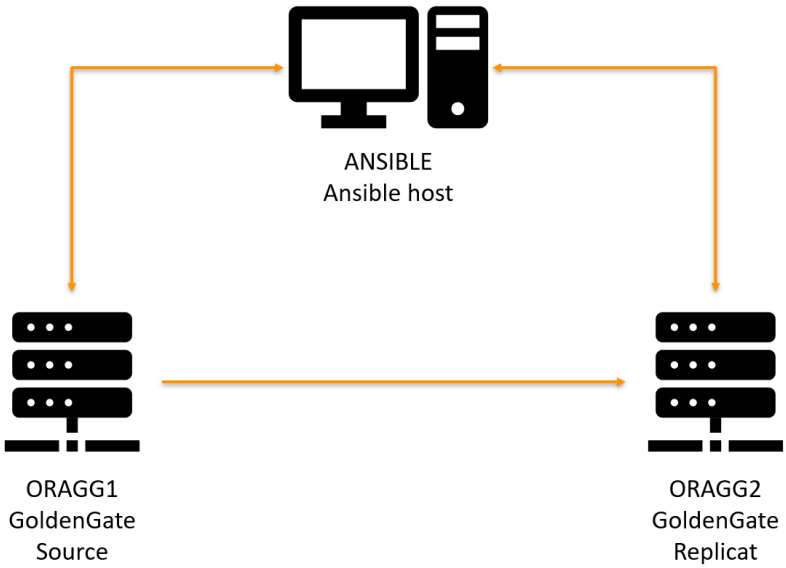
Nicolas Penot, dbi services

Introduction

I like IT automation. I believe automation enables the evolution of IT environments. Indeed, automate your repetitive tasks, and you save time to work on jobs that matter, on tasks that increase the value of your IT ecosystem exponentially. It also makes your day-to-day workload more interesting: learn, code, test, automate and learn something new.
Nevertheless, what automation tool fits your need? It was a time bash scripts were my favourite tool for that job. Easy to code, easy to implement. Bash is also embedded in almost all environments which makes it broadly available and portable. It still works nicely for me. However, when various folks work around a common infrastructure, that becomes interesting to have some standards. A predictable codding structure easy to read and easy to maintain. After having discovered Ansible, I have to say that it's a serious tool for IT automation.
This article presents Ansible with examples around GoldenGate. How can we code automation for Oracle product like GoldenGate?

Use case with GoldenGate

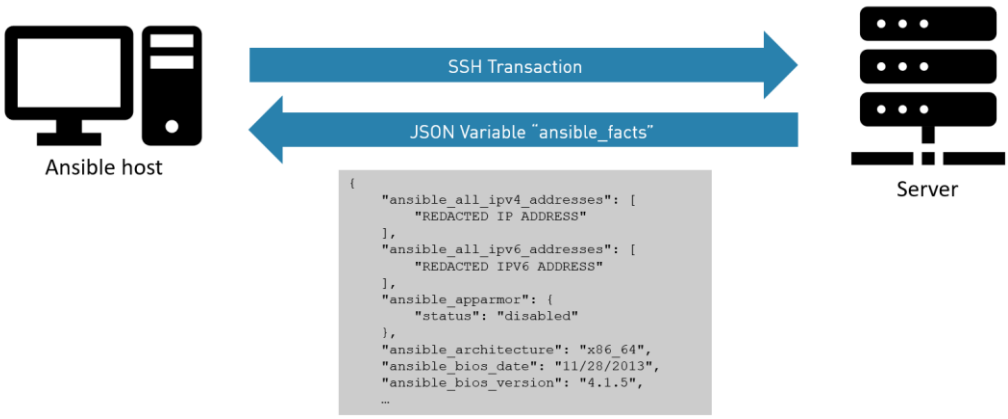
In this demonstration, I have three servers. Two servers host an Oracle database plus the GoldenGate binaries. The third one will serve as Ansible host from which I'll execute my Ansible tasks:



Ansible

From my point of view, these are the 3 points to understand Ansible and its value quickly:

- **Agentless:** Ansible doesn't require agents to be installed on your target servers. Instead, it connects through the secured SSH protocol to execute its tasks.
- **Facts:** Facts are a set of variables that Ansible will collect on target hosts at run time. Those variables are then usable within your scripts. Facts are almost all information you may need from a target host like the IPs, NICs, Devices, etc. You can even add your fact like the list of Oracle instance running with their Oracle Home for example.



- **Two levels**:
  - You can use the so-called ad-hoc command line tool. This tool will permit you with one command line to execute actions, like creating OS users, on multiple servers.
  - Then you may want to script a set of operations. For that purpose, you'll use Playbook. A playbook is a file contention your sequence of operations in a YAML format.

Installation

You now know the basics and are ready to test it by yourself. First, you'll probably need to install Ansible. At the time I write this article the Ansible documentation provides the following instructions:

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

Now, you need to have passwordless SSH connectivity from the Ansible host to the Oracle servers. The snippet below is executed on the Ansible host. The first command creates an RSA authentication key. The next three commands install the public key in the remote authorized_keys file.

```
$ 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
```

Ansible inventory

The inventory is a file used by Ansible to resolve group name. Like a DNS would get an IP from a hostname. Ansible uses the inventory to get a list of hosts from a group. This is because Ansible will use host group as a parameter to execute actions instead of using the hostnames. Here is an example:

```
$ 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
```

You can use “ansible-inventory” command to see what’s Ansible currently has in its inventory:

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

There are two groups, “db_prod” and “jumphost”. The “db_prod” group includes two subgroups one including the servers used as extraction for GoldenGate and one including the hosts used for replication.

Ansible ad-Hoc command line tool

The ad-hoc command is “ansible”. In the example below, I use the Ansible module “command” to execute the command “hostname -f” on each server from the group “db_prod”.

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

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

Then Ansible output a report of the execution including the return of the hostname command. With this simple command, you already imagine what you can do easily and quickly.

Maybe now, your imagination is growing, and you'd like to create more complex automation. Ok, let's discover the Playbooks.

Ansible module

As seen above, I used the "--module-name" option. Ansible works with modules. We can compare modules as plugins. A module gives capacity to Ansible to perform actions like copying a file, creating an OS user, or even create an Amazon EC2 instance. Here is some example of the more than 2000 modules which currently exist:

- 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
- azure – create or terminate a virtual machine in azure (D)
- ec2 – create, terminate, start or stop an instance in ec2
- ec2_ami – create or destroy an image in ec2

For instance, here is the Ansible command I can use to create a new user called "nicolas" on all servers from the group "db_prod":

```
$ 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
}
```

Playbooks

Now that you can run ad-hoc commands, you may be interested in combining multiple modules to perform more elaborate actions. Instead of putting all commands sequentially into a bash script, Ansible provides the concept of playbook. A playbook is a YAML formatted file in which you list the modules sequentially to be executed on a group of hosts. This is an example:

```
$ 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_shiphome.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 -waitforcompletion
INSTALL_OPTION=ORA12c SOFTWARE_LOCATION={{ goldengate_home }}
START_MANAGER=false INVENTORY_LOCATION={{ inventory_location }}
UNIX_GROUP_NAME=oinstall"
    become: yes
    become_user: oracle
```

This playbook includes four modules. The first module “file” is used to create a home for our GoldenGate installation. The second module “unarchive”, unzip the GoldenGate installation files into the directory "/u01/app/software/goldengate". The third one, “fact” create a variable I can use in the next module of the current playbook. And the last one, “command”, which we already know, execute the so-called Oracle runInstaller in command line.
All those modules from that playbook will be executed against the “db_prod” group. This can be seen in the third line of the YAML file in the “hosts” attribute. The hosts attributes describe a group of hosts from the inventory, not a hostname list of your servers.
You can then execute the playbook with the “ansible-playbook” command like:

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

Variables

In the previous snippet, we can also see variables in the YAML file. The variable are wrapped between double curly brackets such as “{{ My_varibale }}”. Those variables are substituted according to the current environment while running. Therefore, the same playbook can be reused for any server and values which differ between servers like IPs for instance, can be converted in variables which change according to their run time environment. It also permits to modify a variable instead of all playbooks when you change a global variable like, for instance, the default Oracle home.

The model used for variables is Jinja2. Jinja2 is a library for Python used by Ansible as a templating language. It also permits more complex structure like conditional substitutions or loop substitutions (More to discover on their website: http://jinja.pocoo.org/docs/2.10/).

Roles

Imagine you've created a playbook which creates a home directory for Oracle, creates the OS users and groups as required by your standards plus configures the OS kernel parameters and installs all Oracle packages as are necessary for Oracle software. Maybe you want to reuse or make reusable that code?
To do so, Ansible has the concept of role. A role is a set of modules you've developed which can be reused in multiple playbooks. We can compare them as a function in a programming language.
For instance, I have a playbook below which only call roles to perform the GoldenGate initial load:

```
$ 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
```

This playbook includes three roles. Two of them are executed on the group of hosts called "source" and the last one is executed in the group of hosts called "replicat". As said, roles are a set of modules written in a YAML format. For instance, this is the YAML file of the role "GoldenGate" with the option "add_replication":

```
$ 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"
```

This role is composed of 4 Ansible modules: “file”, “template”, “template” and “shell”.
To use that playbook, I also need to add additional parameters to indicate the database on which I’ll install the replication:

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

And here is the whole output for our use case:

```
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]
```

```
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]
```

Conclusion

In a nutshell: Ansible uses **modules** to execute tasks **thought SSH** on **groups** of hosts defined into an **inventory.**

As we've seen in this article, in few steps Ansible strengthen your productivity and make your infrastructure proper:

1. Install Ansible on a central host
2. Create an inventory of your servers
3. Convert repetitive tasks to playbooks

I hope this article helps you to understand Ansible efficiently. However, do not hesitate to contact us should you need more details.