CYBERTEC

POSTGRESQL SERVICES & SUPPORT

Use Ansible to shepherd your Elephants

Julian Markwort

pgconf.eu 2023

Introduction

Julian Markwort
Senior Database Consultant

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

- PostgreSQL Consulting
- PostgreSQL Support
- PostgreSQL Remote DBA
- and more ...

Motivation

- I do a lot of semi-automated deployments
  - why not automated? because every customer has different needs…
  - I try to automate many of the tasks using Ansible
- you need help managing all your databases (?)

Topics:

- Part I: Introduction to Ansible, YAML
- Part II: Writing Playbooks
- Part III: Writing Templates
- Part IV: DBA tasks with Ansible
- Part V: Creating and Keeping secrets with Ansible

Part I: Introduction to Ansible, YAML

What is Ansible?

- tool to manage all your servers/machines/containers/…
- works with heterogeneous environments, i.e..:
  - different operating systems and versions
  - different package managers
  - through jump servers…
- executes tasks on targets using *ssh* and *python* only
- tasks described in *YAML*
- lots of modules for additional features

What is YAML ?

- Yet Another Markup Language
- superset of JSON (i.e. JSON is valid YAML but not the other way round):
- good way of making data (variables, objects, arrays) human readable

What is YAML ?

```yaml
hello: world
array:
  - one: foo
  - two: bar
object:
  a: test
  b: 42
```

```json
{
  "hello": "world",
  "array": [
    {
      "one": "foo"
    },
    {
      "two": "bar"
    }
  ],
  "object": {
    "a": "test",
    "b": 42
  }
}
```

create an inventory

- specifying all hosts becomes fairly annoying soon
- Inventories are written in ini or YAML files

`cat inventory`

```
[all:vars]
ansible_user=root

[db_hosts]
demo-server-1
demo-server-2
demo-server-3
```


create an inventory

cat inventory.yaml

```yaml
all:
  vars:
    ansible_user: root
db_hosts:
  hosts:
    demo-server-1:
    demo-server-2:
    demo-server-3:
```


use an inventory

```
ansible db_hosts -i inventory -m shell -a 'uname -r'
```

```
demo-server-3 | CHANGED | rc=0 >>
5.15.0-91-generic
demo-server-2 | CHANGED | rc=0 >>
5.15.0-91-generic
demo-server-1 | CHANGED | rc=0 >>
5.15.0-91-generic
```


a simple playboook

- it becomes annoying to run all modules manually, that's what playbooks are for!

`cat playbook.yaml`

```yaml
- hosts: demo-server-1
  tasks:
    - name: run whoami
      shell:
        cmd: whoami
      register: whoami_result
    - name: show returned value
      debug:
        var: whoami_result.stdout
```


run a playbook

```
ansible-playbook -i inventory playbook.yaml
```

```
PLAY [demo-server-1] ***********************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
Monday 11 December 2023  22:27:20 +0100 (0:00:00.006)       0:00:00.006 *******
ok: [demo-server-1]

TASK [run whoami] **************************************************************************************************
Monday 11 December 2023  22:27:22 +0100 (0:00:01.539)       0:00:01.546 *******
changed: [demo-server-1]

TASK [show returned value] *****************************************************************************************
Monday 11 December 2023  22:27:22 +0100 (0:00:00.468)       0:00:02.015 *******
ok: [demo-server-1] => {
    "whoami_result.stdout": "root"
}

PLAY RECAP *********************************************************************************************************
demo-server-1              : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

Monday 11 December 2023  22:27:22 +0100 (0:00:00.077)       0:00:02.093 *******
===============================================================================
Gathering Facts ---------------------------------------------------------------------------------------------- 1.54s
run whoami --------------------------------------------------------------------------------------------------- 0.47s
show returned value ------------------------------------------------------------------------------------------ 0.08s
```


run a playbook

- “Gathering Facts” takes quite a long time initially
- What are these “Facts” that Ansible gathers?
  - information about the target machines

```
ansible demo-server-1 -i inventory -m setup
```


Ansible Facts - network stuff

```
"ansible_default_ipv4": {
    "address": "192.168.124.66",
    "interface": "enp1s0",
```


Ansible Facts - OS stuff

```
"ansible_distribution": "Ubuntu",
"ansible_distribution_major_version": "22",
"ansible_distribution_release": "jammy",
"ansible_distribution_version": "22.04",
```


Ansible Facts - machine

```
"ansible_memtotal_mb": 991,
"ansible_nodename": "demo-server-1",
"ansible_os_family": "Debian",
"ansible_pkg_mgr": "apt",
"ansible_processor_cores": 1,
```


speed up Fact Gathering

All this fact gathering takes up valuable time! - can be cached!

`cat ansible.cfg`

```
[defaults]
gathering = smart
fact_caching = jsonfile
fact_caching_connection = .ansible_cache/
# timeout in seconds
fact_caching_timeout = 7200
```


Task Failure

- Ansible will halt execution of further tasks for a host that failed a task
  - other hosts can still continue with the remaining tasks
- behaviour can be changed for every task:
  - `ignore_errors` - ignore errors if not critical
  - `failed_when` - what constitutes a failure
  - `changed_when` - what consistutes a “changed” state

Part II: Writing Playbooks

Playbook concepts

- playbooks are like cooking recipes
- combine multiple modules in the right order
- use results from previous modules to change behaviour
- use variables to change behaviour

define your own variables

- add variables directly to inventory

```
[db_hosts:vars]
pg_port=5432
pg_version=16
pg_base_dir=/pgsql
```


Playbook to install PostgreSQL

cat install_postgresql.yaml

```
- hosts: db_hosts
  tasks:
  - name: install postgresql-common
    apt:
      name: postgresql-common

  - name: run script to install postgresql.org repos
    shell: "/usr/share/postgresql-common/pgdg/apt.postgresql.org.sh -y"
```


Playbook to install PostgreSQL

```
- name: disable creation of main postgresql cluster
  lineinfile:
    path: /etc/postgresql-common/createcluster.conf
    regexp: "^#create_main_cluster = true"
    line: "create_main_cluster = false"
```


Playbook to install PostgreSQL

```
- name: install postgresql
  apt:
    name:
    - "postgresql-client-{{pg_version}}"
    - "postgresql-{{pg_version}}"
    update_cache: yes
```


More Variables

```
cat vars.yaml
```

```
pg_bin_dir: "/usr/lib/postgresql/{{pg_version}}/bin"
pg_data_dir: "{{pg_base_dir}}/{{pg_version}}/data"
```


Include Variables

cat initdb.yaml

```
- hosts: db_hosts
  tasks:
  - include_vars: vars.yaml

  - name: make sure postgres base dir exists
    file:
      name: "{{pg_base_dir}}"
      state: directory
      owner: postgres
      group: postgres
      mode: 0700
```


Conditional Tasks

```
- name: Check if PostgreSQL database is initialized.
  stat:
    path: "{{pg_data_dir}}"
  register: pg_data_dir_stat

- name: Run initdb
  when: not pg_data_dir_stat.stat.exists
  become: true
  become_user: postgres
  command: >-
    {{pg_bin_dir}}/initdb --pgdata {{pg_data_dir}}
      {% if use_data_checksums is defined %}
      --data-checksums
      {% endif %}
```


Part III: Writing Templates

Jinja Templating

- Jinja is nice for templating words, sentences, whole documents
- We've already seen some Jinja magic in the playbooks
  - everything surrounded by `{{}}` is treated as a Jinja block

Jinja offers:

- boolean expression evaluation
- (nested) loops
- filters:
  - make sure a variable is not undefined

```
possibly_undefined | default('defined')
```

  - list to CSV

```
list_of_things | join(',')
```

  - set operations

```
set_a | difference (set_b)
```

  - and many more

Template a Config File

```
cat template/postgresql.conf.j2
```

```
# cluster
listen_addresses = '0.0.0.0'
port = '{{pg_port}}'

# memory
shared_buffers: "{{(ansible_memtotal_mb*0.25) | int}}MB"

# Parallel queries:
max_worker_processes: "{{ansible_processor_cores}}"
```


pg_hba - goal

- connections can be established when
  - they match a database,
  - they match a user,
  - and they match a host

pg_hba - helper vars

```
- databases:
  - dbname: db1
    users:
    - uname: db1_user
      hosts:
      - app1_host1.local
      - app1_host2.local
    - uname: db1_admin
      hosts:
      - 10.100.200.123/32
  - dbname: all
    users:
    - uname: pgwatch2
      hosts:
      - 10.100.200.224/31
```

pg_hba - template

cat template/pg_hba.conf.j2

```
local  all  postgres  peer

{% for database in databases %}
{% for user in database.users %}
{% for host in user.hosts %}
host  {{database.dbname}}  {{user.uname}}  {{host}}  scram-sha-256
{% endfor %}
{% endfor %}
{% endfor %}
```


pg_hba - result

```
local all postgres peer

host  db1  db1_user  app1_host1.local  scram-sha-256
host  db1  db1_user  app1_host2.local  scram-sha-256
host  db1  db1_admin  10.100.200.123/32  scram-sha-256
host  all  pgwatch2  10.100.200.224/31  scram-sha-256
```


Part IV: DBA tasks with Ansible

Ansible modules for PostgreSQL

There are several modules in Ansible purely for interacting with PostgreSQL:

- `postgresql_query` : Run PostgreSQL queries
- `postgresql_db` : Add or remove PostgreSQL databases
- `postgresql_user` : Add or remove a user/role
- `postgresql_privs` : Grant or revoke privileges database objects
- and many more ...

requirements

- all postgresql_* modules require psycopg2 on the target host
- syntax is often similar (`login_host`, `login_db`, `login_user`, `login_password`...)

Socket Connections

the easiest connection option is to use unix sockets

```
- name: run SELECT 1 through socket
  become: yes
  become_user: postgres
  postgresql_query:
    login_unix_socket: "/var/run/postgresql/"
    login_port: "{{pg_port}}"
    query: SELECT 1
```


Network Connections

if you cannot cannot connect locally

```
- name: run SELECT 1 through network connection
  local_action:
    module: postgresql_query
    query: SELECT 1
    login_host: "{{ansible_hostname}}"
    login_port: "{{postgres_port}}"
    login_password: "{{postgres_password}}"
```

- `local_action` delegates task execution to Ansible host
- you need to consider transport security

role management

- for each database defined in the vars file
  - create a database in the cluster
  - add users: _owner and _user
  - grant appropriate privileges

Loop over all databases

cat postgresql_privileges.yaml

```
- hosts: db_hosts
  become: yes
  become_user: postgres
  tasks:
    - name: include vars
      include_vars: vars.yaml

    - name: main block
      include_tasks: tasks/postgresql_privileges_db.yaml
      when: db.dbname != 'all'
      loop: "{{databases}}"
      loop_control:
        loop_var: "db"
```


create database

cat tasks/postgresql_privileges_db.yaml

```yaml
- name: create db_owner role with LOGIN
  postgresql_user:
    db: postgres
    name: "{{db.dbname}}_owner"
    role_attr_flags: LOGIN
    password: "{{lookup('vars', db.dbname + '_owner_password')}}"

- name: create database with OWNER db_owner
  postgresql_db:
    name: "{{db.dbname}}"
    owner: "{{db.dbname}}_owner"
```


privileges

```
- name: REVOKE CREATE ON SCHEMA public FROM PUBLIC;
  postgresql_privs:
    db: "{{db.dbname}}"
    state: absent
    priv: CREATE
    type: schema
    obj: public
    role: PUBLIC
```


roles

```
- name: CREATE ROLE db_user LOGIN
  postgresql_user:
    db: "{{db.dbname}}"
    name: "{{db.dbname}}_user"
    role_attr_flags: LOGIN
    password: "{{lookup('vars', db.dbname + '_user_password')}}"
```


default privileges

```
- name: ALTER DEFAULT PRIVILEGES FOR ROLE db_owner GRANT ... to db_user
  postgresql_privs:
    db: "{{db.dbname}}"
    type: default_privs
    target_roles: "{{db.dbname}}_owner"
    privs: "{{item.privs}}"
    objs: "{{item.objs}}"
    role: "{{db.dbname}}_user"
  loop:
  - {privs: "SELECT,INSERT,UPDATE,DELETE", objs: TABLES}
  - {privs: USAGE, objs: SEQUENCES}
  - {privs: EXECUTE, objs: FUNCTIONS}
```


Part V: Keeping secrets with Ansible

ansible-vault

- can be used to encrypt vars
- encrypted vars can be used as usual in playbooks and jinja templates
- encryption key can be given to ansible via a password file or prompt

store a secret in vault

```
ansible-vault encrypt_string \
  --vault-password-file vault.key \
  --name db1_user_password \
  $(openssl rand -base64 24) \
  >> vars/cluster_one.yaml
```


vault contents

cat vars/cluster_one.yaml

```
db1_user_password: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          66643636373036323465326630613366666164343136383738623661646565343831616238626539
          36366332316231666633373037323164613934663731343 10a343963616665613766346663386162
          63633262626166366261353130623334373066653031323464666166656664346131373066336237
          37643439396139663 60a333366616331386535623264333635343861613438316131633735663864
          30343665386539356562353336646239306263643035366263333361666164346665353761633162
          35626332393539353764303937383333331633137373666613936
```


Conclusion

Conclusion

- Ansible is very powerful
- the learning curve is not very steep
  - wrapping your head around the YAML and Jinja Syntax takes a bit of time
- the Ansible docs are a great resource for finding modules, understanding their functionality, and for learning new patterns like loops and filters

Conclusion

- challenge is not using the modules, but using them effectively
- the prerequisite to doing something effectively in Ansible is having good input data in a useable format
- you cannot write a single playbook that can be used for every scenario
  - just like you cannot write one recipe to cover every meal
- therefore it's often fruitless to piggyback onto someone else's playbooks
- learn how to write playbooks yourself!

Thank you