Ansible

Agenda

- Why we need Ansible
- What is Ansible
- Desired states
- How Ansible works
- Ansible Inventory
- Ansible Ad-Hoc
- Ansible Playbook and YAML
- Validate Playbook
- Ansible Roles and reuse the code
- Ansible Galaxy

Why we need Ansible

- Orchestration
- Configuration management
- Provisioning
- Deploy
- Remote API call

What is Ansible

Ansible is **connecting** to the nodes from list named **Ansible Inventory** and sending to them programs called **Ansible Modules**, which are by the nature are **resource model** operating **desired states**

Ansible advantages

Repetable

- desired state
- idempotency

What is desired state?

Most Ansible modules check whether the desired final state has already been achieved, and exit without performing any actions if that state has been achieved, so that repeating the task does not change the final state. Modules that behave this way are often called ‘idempotent.’ Whether you run a playbook once, or multiple times, the outcome should be the same

However, not all modules behave this way

List of Ansible modules

Ansible advantages

Simple

- Text (formatted)
- VCS ready
- No agent needed
- Strong community (Ansible Galaxy)

Flexible

- long list of built-in modules
- able to use remote REST API
- ready to use modules to Cloud API
- can write your own module

How Ansible works

What is needed to run Ansible

- Python
- Ansible
- Ability to deliver modules to hosts from inventory

How Ansible works

What is connections options

```
ansible-doc -t connection -l
buildah      Interact with an existing buildah container
chroot       Interact with local chroot
docker       Run tasks in docker containers
funcd        Use funcd to connect to target
httpapi      Use httpapi to run command on network appliances
iocage       Run tasks in iocage jails
jail         Run tasks in jails
kubectl      Execute tasks in pods running on Kubernetes
libvirt_lxc  Run tasks in lxc containers via libvirt
local        execute on controller
lxc          Run tasks in lxc containers via lxc python library
lxd          Run tasks in lxc containers via lxc CLI
napalm       Provides persistent connection using NAPALM
netconf      Provides a persistent connection using the netconf protocol
network_cli  Use network_cli to run command on network appliances
oc           Execute tasks in pods running on OpenShift
paramiko_ssh Run tasks via python ssh (paramiko)
persistent   Use a persistent unix socket for connection
podman       Interact with an existing podman container
psrp         Run tasks over Microsoft PowerShell Remoting Protocol
qubes        Interact with an existing QubesOS AppVM
saltstack    Allow ansible to piggyback on salt minions
ssh          connect via ssh client binary
vmware_tools Execute tasks inside a VM via VMware Tools
winrm        Run tasks over Microsoft's WinRM
zone         Run tasks in a zone instance
```

Inventory

```
---
[webservers] <--- groups
www1.example.com
www2.example.com

[dbservers]
db0.example.com <--- hosts
db1.example.com
```

ansible-inventory [options] [host|group]

Ansible Ad-Hoc

```
ansible all -m ping -i inventory.ini
```

```
ansible www.example.org -i inventory.ini -a "date"
```

Ansible Playbooks as an automation approach

Ansible Playbooks offer a repeatable, re-usable, simple configuration management and multi-machine deployment system, one that is well suited to deploying complex applications. If you need to execute a task with Ansible more than once, write a playbook and put it under source control

Ansible Playbooks can

- declare configurations
- orchestrate steps on multiple sets of machines, in a defined order
- launch tasks synchronously or asynchronously

By default, Ansible executes each task in order, one at a time, against all machines matched by the host pattern (pre-defined in **Inventory**). Each task executes a module with specific arguments. When a task has executed on all target machines, Ansible moves on to the next task.

This behaviour may be changed by choosing strategies

MVP: Minimum Viable Playbook

- targets
- at least one task to execute

```
---
- name: test play
  hosts: all
  tasks:
    - name: ping
      ansible.builtin.ping:
```

Playbooks are expressed in YAML format

- A dictionary is represented in a simple key: value form

```
users:
  name: kevit
```

- Values can span multiple lines using | or >

```
include_newlines: |
            exactly as you see
            will appear in the key
```

- Ansible uses "{{ var }}" for variables.

Ansible variables

Ansible uses variables to manage differences between systems

You can define these variables

- in your playbooks
- in your inventory
- in roles
- at the command line
- at runtime via register:

Typical playbook directory structure

```
cat inventory.ini
---
[webservers] <--- groups
www1.example.com
www2.example.com

[dbservers]
db0.example.com <--- hosts
db1.example.com
```

```
group_vars/
   webservers.yml
   dbservers.yml
host_vars/
   db0.example.com.yml
   db1.example.com.yml
inventory.ini
playbook.yml
```

Ansible variables: precedence

```
command line values (for example, -u my_user, these are not variables)
role defaults (defined in role/defaults/main.yml)
inventory file or script group vars
inventory group_vars/all
playbook group_vars/all
inventory group_vars/*
playbook group_vars/*
inventory file or script host vars
inventory host_vars/*
playbook host_vars/*
host facts / cached set_facts
play vars
play vars_prompt
play vars_files
role vars (defined in role/vars/main.yml)
block vars (only for tasks in block)
task vars (only for the task)
include_vars
set_facts / registered vars
role (and include_role) params
include params
extra vars (for example, -e "user=my_user")(always win precedence)
```

foo_port is a good name for variable but foo-port, 5foo or foo.port is bad (python!)

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

Filling out variables and transform them

Lookups

```
vars:
  motd_value: "{{ lookup('file', '/etc/motd') }}"
tasks:
  - debug:
      msg: "motd value is {{ motd_value }}"
```

Filters

```
{{ 'secretpassword' | password_hash('blowfish', '1234567890123456789012', ident='2b') }}
```

How to write a good Playbooks?

- Use VCS
- Remember about Idempotency
- Imperative! ()
- Fail properly (fail: or assert:)

```yaml
- name: "Unknown error"
  fail:
    msg: "Something happened"
  when: result.stdout == "Error"
```

Name tasks right way

```
- name: "nginx: 005 - create directories"

ansible-playbook -i inventory.ini playbook.yml --start-at-task "nginx: 005 - create directories"
```

Use different verbosity levels

```
- debug:
  msg: "always"
- debug:
  msg: "only at -vv"
  verbosity: 2
```

Validate

- yamllint helps to check yaml syntax
- The ansible-playbook command offers several options for verification
  --check, --diff, --list-hosts, --list-tasks, and --syntax-check.
- ansible-lint helps to check ansible-specific issues
- ansible-playbook -i inventory.ini playbook.yml --step
- internal debugger ANSIBLE_STRATEGY=debug

https://github.com/ansible/test-playbooks

Ansible Roles Reuse a code right way

- Ansible Roles helps to reuse your code
- Roles let you automatically load related vars, files, tasks, handlers as a part of playbook
- After you group your content in roles, you can easily reuse them and share them with other users.

Typical role skeleton

```
ansible-galaxy init role_name
role_name/
    README.md
    defaults/
        main.yml
    files/
    handlers/
        main.yml
    meta/
        main.yml
    templates/
    vars/
        main.yml
```

Typical role skeleton: meaning

```
tasks/main.yml - the main list of tasks that the role executes.
handlers/main.yml - handlers, which may be used within or outside this role.
defaults/main.yml - default variables for the role
vars/main.yml - other variables for the role
files/ - files that the role deploys.
templates/ - templates that the role deploys.
meta/main.yml - metadata for the role, including role dependencies.
```

How to call the role

```yaml
---
- hosts: webservers
  roles:
    - webservers
```

```yaml
- hosts: webservers
  tasks:
    - name: Include the example role
      include_role:
        name: webservers
```

Using Ansible Galaxy

```
ansible-galaxy install -r requirements.yml:

cat requirements.yml
# Role on galaxy
- your.rolename
# Public role on github
- name: role-public
  src: https://github.com/user/role-public.git
```