Ansible Advanced

Ansible ecosystem

- Ansible Collections
- Testing roles and playbooks
- Linting your Code
- Ansible-cmdb
- Ansible CI/CD

Useful Ansible features

- Selecting variables, files, or templates based on facts
- Ansible-vault
- Delegation + Run Once
- Error-Handling
- Common errors
- Working with data

Changing Ansible execution

- Plugins
- Filters

Ansible Collections

What are they and why do they exist?

- Maintenance burden
- Slow releases

Status quo

- Ansible-core
  - Ansible engine
  - Core Plugins
- Ansible Community Collection
  - All existing plugins
  - collections

Ansible Collections - Structure

```
collection/
├── docs/
├── galaxy.yml
├── plugins/
│   ├── modules/
│   └── inventory/
├── README.md
├── roles/
│   ├── role1/
├── playbooks/
│   ├── files/
│   ├── vars/
│   ├── templates/
│   └── tasks/
├── tests/
```

Installation

```
# Install a collection from Ansible Galaxy
ansible-galaxy collection install t_systems_mms.icinga_director

# Install a collection in a repository using the latest commit on the branch 'devel'
ansible-galaxy collection install git+https://github.com/organization/repo_name.git,devel
```

Installation - continued

with requirements.yml:

```
collections:
  - name: https://github.com/organization/repo_name.git
    type: git
    version: devel
```

Usage

ansible.cfg

```
[defaults]

collections_path = /path/to/ansible/collections/
```

Usage - continued

```
- name: create all hostgroups according to the ansible host groups
  hosts: localhost
  tasks:
    - name: create all hostgroups according to the ansible host groups
      t_systems_mms.icinga_director.icinga_host:
        state: present
        object_name: "foohost"
```

Usage - continued

```
- name: create all hostgroups according to the ansible host groups
  hosts: localhost
  collections:
    - t_systems_mms.icinga_director
  tasks:
    - name: create all hostgroups according to the ansible host groups
      icinga_hostgroup:
        state: present
        object_name: "foohost"
```

Converting your role to a collection

- create collection path
- update ansible.cfg collections path
- create a new collection
- update playbooks that use the content

Collections of AOC

- icinga director
- icinga
- mongodb atlas
- lets encrypt
- hardening

Testing Roles And Playbooks With Molecule

Molecule aids in the development and testing of Ansible roles.

https://github.com/ansible/molecule

Installation

- Install molecule

```
bash$> pip install --user molecule[docker]
```

- Create a new role with Molecule

```
bash$> molecule init role my.molecule_test -d docker
```

- Add Molecule to an existing role

```
bash$> molecule init scenario -d docker
```

Installation with Ansible

```
- name: molecule
  hosts: localhost
  pre_tasks:
    - name: install docker-py
      yum:
        name:
          - python-docker-py.noarch
          - python-requests
        state: present
  vars:
    pip_install_packages:
      - name: molecule[docker]
  roles:
    - geerlingguy.pip
    - geerlingguy.docker
  post_tasks:
    - name: add user to docker group
      user:
        name: "{{ username }}"
        shell: /bin/bash
        groups: docker
        append: yes
```

Configuration

```
dependency:
  name: galaxy
  options:
    role-file: molecule/os_hardening/requirements.yml
driver:
  name: docker
lint: |
  yamllint roles/os_hardening/ molecule/os_hardening/ .github/workflows/os_hardening.yml
  ansible-lint roles/os_hardening/
platforms:
  - name: instance
    image: "rndmh3ro/docker-${MOLECULE_DISTRO}-ansible:latest"
    command: ${MOLECULE_DOCKER_COMMAND:-/lib/systemd/systemd}
    env:
      container: docker
      http_proxy: ${http_proxy}
      https_proxy: ${https_proxy}
      no_proxy: ${no_proxy}
provisioner:
  name: ansible
verifier:
  name: ansible
```

Usage

```
bash$> molecule test     # test everything
bash$> molecule lint     # linting

bash$> molecule converge # create host and run ansible
bash$> molecule login    # login into host to debug
```

Linting your Code

Ansible playbooks, roles, and collections should read like documentation, be production ready, unambiguous, and provide consistent results.

Ansible-lint should be considered a trusted advisor, helping ansible content creators write and package high-quality Ansible content. While not all rules may be applicable in all situations, they should be followed whenever possible.

```
> ansible-lint init_master.yml
WARNING  Overriding detected file kind 'yaml' with 'playbook' for given positional argument: init_master.yml
WARNING  Listing 129 violation(s) that are fatal
yaml[truthy]: Truthy value should be one of [false, true]
init_master.yml:5

name[casing]: All names should start with an uppercase letter. (warning)
init_master.yml:15 Task/Handler: save username of AWS user

yaml[indentation]: Wrong indentation: expected 8 but found 9
init_master.yml:17

 count tag                       profile    rule associated tags
     2 deprecated-command-syntax basic      command-shell, deprecations
     1 key-order[task]           basic      formatting, experimental (warning)
     1 jinja[spacing]            basic      formatting (warning)
```

Linting Rules

- https://ansible-lint.readthedocs.io/rules/

Ansible-cmdb

“ Generate host overview from ansible fact gathering output ”

https://github.com/fboender/ansible-cmdb

Installation

```
bash$> sudo pip install ansible-cmdb
```

Usage

First, generate Ansible output for your hosts:

```
bash$> mkdir out
bash$> ansible -m setup --tree out/ all
```

Next, call ansible-cmdb on the resulting out/ directory to generate the CMDB overview page:

```
bash$> ansible-cmdb out/ > overview.html
```

Ansible CI/CD with Gitlab

(Why not) Ansible AWX

What is Ansible AWX?

> AWX provides a web-based user interface, REST API, and task engine built on top of Ansible.

> It is the upstream project for Tower, a commercial derivative of AWX.

What is Ansible AWX for me?

- unstable (with no useful changelog)
- Complex
- no more features than Jenkins
- **Use Tower if you have the money or use Gitlab**

Ansible AWX in Production?

> There are definitely people running AWX in production. There are so many little edge cases to solve as time goes on though. It's definitely possible (just like most things are in tech). But, I do not recommend it.
>
> - Chris Short, former Ansible Principal Product Marketing Manager

Jenkins?

- Installation and demo is cumbersome
- Using Gitlab to show CI/CD is easier

Others?

- Semaphore - Modern UI for Ansible
- Polemarch - Ansible based service for IT infrastructure management

Gitlab

- Continous Integration
- Continous Deployment

Gitlab CI

- Lint your Ansible-Code with this one easy step!

ansible-lint

```
ansible:
  stage: linting
  image: pipelinecomponents/ansible-lint:latest
  script:
    - chmod g-w,o-w .  # https://gitlab.com/gitlab-org/gitlab-runner/-/issues/1736
    - ansible-lint -f rich --force-color
  only:
    - pushes
```

Gitlab CD

.gitlab-ci.yml

```
ansible-playbooks:
  stage: ansible-playbooks
  variables:
    ANSIBLE_HOST_KEY_CHECKING: "false"
  image: rndmh3ro/docker-centos7-ansible:latest
  script:
    - ansible-playbook site.yml -i user0-instances.txt -v --diff --become
    - rm -f .vault
  only:
    refs:
      - schedules
    variables:
      - $run_all_playbooks
```

Gitlab CD

CI-playbook.yml

```
- import_playbook: "resolv.yml"
- import_playbook: "repositories.yml"
- import_playbook: "hardening.yml"
- import_playbook: "hosts_file.yml"
- import_playbook: "packages.yml"
- import_playbook: "disable_firewalld.yml"
```

Useful Ansible Features

Selecting variables, files, or templates based on facts

```
- name: install apache
  package:
    name: "{{ apache_package_name }}"
    state: "{{ apache_package_state }}"
    update_cache: "{{ (ansible_pkg_mgr == 'apt') | ternary('yes', omit) }}"
    cache_valid_time: "{{ (ansible_pkg_mgr == 'apt') | ternary(cache_timeout, omit) }}"
```

Selecting variables, files, or templates based on facts

```
- name: set OS dependent variables
  include_vars: '{{ item }}'
  with_first_found:
    - '{{ ansible_facts.distribution }}_{{ ansible_facts.distribution_major_version }}.yml'
    - '{{ ansible_facts.distribution }}.yml'
    - '{{ ansible_facts.os_family }}_{{ ansible_facts.distribution_major_version }}.yml'
    - '{{ ansible_facts.os_family }}.yml'
```

Selecting variables, files, or templates based on facts

```
# RedHat and Debian system are too different in regards of
# package management, so keep a task file for each
- name: Perform package system tasks
  include_tasks: "{{ ansible_pkg_mgr }}_packages.yml"
```

```
tasks/yum_packages.yml
tasks/apt_packages.yml
```

Selecting variables, files, or templates based on facts

```
- import_tasks: package-apt.yml
  when: ansible_pkg_mgr == 'apt'
- import_tasks: package-yum.yml
  when: ansible_pkg_mgr == 'yum'
- import_tasks: package-dnf.yml
  when: ansible_pkg_mgr == 'dnf'
- import_tasks: package-brew.yml
  when: ansible_os_family == 'Darwin'
- import_tasks: package-zypper.ymlec2_vpc_route_table_facts
  when: ansible_pkg_mgr == 'zypper'
```

Ansible-vault

Store credentials in code and use them in Ansible.

- Encrypt whole files or single strings in files
- store them with different passwords
- rekey encrypted files

Overview

	Encrypted variables	Encrypted files
How much is encrypted?	Variables within a plaintext file	The entire file
When is it decrypted?	On demand, only when needed	Whenever loaded or referenced
What can be encrypted?	Only variables	Any structured data file

create and view individual variables

```
# before
root_password: hunter2

# after
root_password: !vault |
          $ANSIBLE_VAULT;1.2;AES256;
          376365613666366434643763363034666130626335732363230656665353338383366462366662
          656535306330306530383132353965613865386335323062 0a65363864363933313330633133636 5
          623737376233376161303861373734613065353835383731623162633861653761316236313234 34
          386636386236333562 0a376466656164383032633338306162326639643635663936623939666238
          3161
```

create and view individual variables

```
# encrypt a password
ansible-vault encrypt_string <password_source> '<string_to_encrypt>' --name '<string_name_of_variable>'

# view the password
ansible localhost -m ansible.builtin.debug -a var="new_user_password" -e "@vars.yml"-
```

Encrypt, create, view files

- Encrypt vars-files, tasks-files, handlers-files, binary files

```
# create a new encrypted file
ansible-vault create foo.yml

# encrypt an existing file
ansible-vault encrypt foo.yml bar.yml baz.yml

# view encrypted files
ansible-vault view foo.yml bar.yml baz.yml

# edit encrypted files
ansible-vault edit foo.yml

# change the password of encrypted files
ansible-vault rekey foo.yml bar.yml baz.yml
```

Using encrypted variables and files

```
ansible-playbook --ask-vault-pass site.yml
ansible-playbook --vault-password-file /path/to/my/vault-password-file site.yml
```

Delegation + Run Once

- **Delegation**: perform a task on one host with reference to other hosts
- **Run Once**: Only run a task one time for a batch of hosts

```
- hosts: web
  tasks:
    - name: disable nagios monitoring
      command: /bin/disable_monitoring.sh
      delegate_to: monitoring_server
      run_once: true

    - name: actual steps would go here
      yum:
        name: acme-web-stack
        state: latest

    - name: enable nagios monitoring
      command: /bin/enable_monitoring.sh
      delegate_to: monitoring_server
      run_once: true
```

Error-Handling

- ignore_errors
- failed_when
- changed_when

ignore_errors

Useful for:

- deletion of things that would otherwise throw an error (users, kernel modules)

```
bash$> userdel non_existant
userdel: User "non_existant" does not exist.
bash$> echo $?
6
```

- to check if something is installed on remote system
  - `ps axu | grep java`

failed_when

Useful in:

- error-msg in logs
- output from command

Example #1

```
- name: Fail task when the command error output prints FAILED
  command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"
```

Example #2

```
- name: check if there are failed login attempts
  shell: cat /var/log/secure
  register: user_accts
  failed_when: user_accts.stdout.find('failure') != -1

- name: Wait until string "completed" is in the logfile
  wait_for:
    path: /var/log/tomcat/catalina.out
    search_regex: completed
```

Example #3

```
- name: Making sure the Physical Memory more than 2gb
  shell: "cat /proc/meminfo|grep -i memtotal|awk '{print $2/1024/1024}'"
  register: memory
  failed_when: "memory.stdout|float < 2"
```

changed_when

Useful for:

- making checks idempotent

```
    - name: Get user accounts | os-09
      command: "awk -F: '{print $1}' /etc/passwd"
      changed_when: False
      check_mode: False
      register: users
```

Common Errors

Example #1

Code

```
- hosts: localhost
  tasks:
  - name: Simple A record (IPV4 address) lookup
    debug:
      msg: "{{ lookup('dig', '52.59.192.245/PTR')}}
```

Error

```
ERROR! Syntax Error while loading YAML.
  found unexpected end of stream

The error appears to have been in 'test.yml': line 12, column 1, but may
be elsewhere in the file depending on the exact syntax problem.

(specified line no longer in file, maybe it changed?)
```

Reason:

Missing quotes at the end

Example #2

Code

```
- hosts: localhost
  tasks:
  - name: Simple A record (IPV4 address) lookup
     debug:
      msg: "{{ lookup('dig', '52.59.192.245/PTR')}}"
```

Error

```
ERROR! Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to have been in 'lookup_dig.yml': line 4, column 11, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

  - name: Simple A record (IPV4 address) lookup
     debug:
          ^ here
```

Reason

Wrong indentation

Example #3

Code:

```
- hosts: localhost
  tasks:
  - name: Simple A record (IPV4 address) lookup
    debug:
    msg: "{{ lookup('dig', '52.59.192.245/PTR')}}"
```

Error

```
ERROR! 'msg' is not a valid attribute for a Task

The error appears to have been in 'lookup_dig.yml': line 3, column 5, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

  tasks:
  - name: Simple A record (IPV4 address) lookup
    ^ here

This error can be suppressed as a warning using the "invalid_task_attribute_failed" configuration
```

Reason

Wrong indentation

Working with data

- https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html#managing-data-types
- https://docs.ansible.com/ansible/latest/user_guide/complex_data_manipulation.html

Basic data structures in Ansible

Key-Value-Pairs (scalars)

```
user: test
```

Lists

```
domains:
  - example.com
  - example.de
```

Dictionaries (or hashes)

```
users_dict:
  test1:
    name: user1
    fullname: "Test No. 1"
  test2:
    name: user2
    fullname: "Test No. 2"
```

Nested data structures

```
list_of_users_dicts:
  - name: user1
    fullname: "Test No. 1"
  - name: user2
    fullname: "Test No. 2"
```

Important

- Loops require lists, not dicts, docs

Templating

Ansible embeds the Jinja2 templating engine that can be used to dynamically:

- Set and modify play variables
- Conditional logic
- Generate files such as configurations from variables

Jinja-docs: https://jinja.palletsprojects.com/en/3.1.x/templates/

General Syntax

- `{{ ... }}` for variables
- `{% ... %}` for conditional statements
- `{# ... #}` for comments (describe the task)

for-loops with gathered facts

```
{% for host in groups['all'] %}

{{ hostvars[host].ansible_facts.default_ipv4.address }}
{{ hostvars[host].ansible_facts.fqdn }}
{{ hostvars[host].ansible_facts.hostname }}

{% endfor %}
```

for-loops with list variables

```
fruits:
{% for item in fruits %}
{{ item }}
{% endfor %}
```

for-loops with list variables

```
- hosts: all
  vars:
    fruits:
      - name: apple
        taste: very good
      - name: banana
        taste: sweet
      - name: clementine
        taste: sour
  tasks:
    - template:
        src: httpd.conf.j2
        dest: httpd.conf
```

for-loops with list variables

```
{% for item in fruits %}

name: {{ item.name }}
taste: {{ item.taste }}

{% endfor %}
```

for-loops with registered variables

```
- hosts: localhost
  tasks:
    - name: register a variable
      command: hostname
      register: output

    - template:
        src: httpd.conf.j2
        dest: "httpd.conf"
```

for-loops with registered variables

```
{% for item in output.stdout_lines %}
{{ item }}
{% endfor %}
```

for-loops with dictionaries

```
- hosts: all
  vars:
    users:
      test1:
        name: user1
        fullname: "Test No. 1"
      test2:
        name: user2
        fullname: "Test No. 2"
```

for-loops with dictionaries

```
{% for item in users.values() %}
username: {{ item.name }}
full name: {{ item.fullname }}
{% endfor %}
```

if-else clauses

```
{% if users is defined %}

{% for item in users.values() %}
username: {{ item.name }}
full name: {{ item.fullname }}
{% endfor %}

{% else %}
no users configured!

{% endif %}
```

Changing Ansible Execution

Plugins

Plugins are pieces of code that augment Ansible's core functionality.

- Lookup
- Strategy
- Caching
- Callback
- Connection

Lookup Plugins

Lookup plugins allow Ansible to access data from outside sources.

dig

- The dig lookup runs queries against DNS servers to retrieve DNS records for a specific name
- Useful for:
  - Templates where you can't/don't want to use names
  - Get name for IP-address

Examples:

```
- name: Simple A record (IPV4 address) lookup
  debug:
    msg: "{{ lookup('dig', 'google.de')}}"


- debug:
   msg: "Reverse DNS for 192.0.2.5: {{ lookup('dig', '192.0.2.5/PTR') }}"
```

pipe - read output from a command

- Useful for:
  - Get variables from commands

Example #1

```
- name: download githubs ssh-key
  known_hosts:
    path: ~/.ssh/known_hosts
    name: github.com
    key: "{{ lookup('pipe', 'ssh-keyscan -t rsa github.com') }}" #### Example #2

- hosts: localhost
  tasks:
  - name: tell the host about our servers it might want to ssh to
    known_hosts:
      path: ~/.ssh/known_hosts
      name: yourhost
      key: "{{ lookup('pipe', 'ssh-keyscan -t rsa yourhost') }}" #### Example #3
```

- check if server has virtual IP

```
    VIP=`/sbin/ip addr | grep {{ lookup('pipe', 'getent hosts VIP | cut -d " " -f1') }} | wc -l`
```

More lookups

- url - return contents from URL
- file - read file contents
- fileglob - list files matching a pattern
- password - retrieve or generate a random password, stored in a file
- env – read the value of environment variables

Strategy Plugins

Strategy plugins control the flow of play execution by handling task and host scheduling.

Types of Strategy Plugins

- linear (default)
- free
- debug
- host_pinned

Linear Strategy

Task execution is in lockstep per host batch up to the fork limit

Useful for:

- every scenario

Free Strategy

Task execution is as fast as possible per host in batch as defined by serial (default all). Ansible will not wait for other hosts.

Useful for:

- Playbook-runs on independent hosts (e.g. deployment)

Callback Plugins

By default, callback plugins control most of the output you see when running the command line programs

Changing Output

Notable Callback Plugins:

- debug
- dense
- minimal
- yaml
- actrionable

Putting Output somewhere

Notable Callback Plugins:

- mail
- logstash
- splunk

Filters

Filters are used for transforming data inside a template expression.

Default-Filter

If a variable is not defined, the value in brackets will be used, rather than an error being raised.

```
- name: Make sure the users are present
  user:
    name: "{{ name }}"
    state: "{{ state | default('present') }}"
```

Default with omit

Skip parameters of a module.

```
- name: Make sure the users are present
  user:
    name: "{{ item.name }}"
    group: "{{ item.group|default(omit) }}"
    groups: "{{ item.groups | join(',') if item.groups is defined else omit }}"  # can be passed as list now
    append: "{{ item.append|default(omit) }}"
    password: "{{ item.pass|default(omit) }}"
    comment: "{{ item.comment|default(omit) }}"
    shell: "{{ item.shell|default(omit) }}"
    uid: "{{ item.uid|default(omit) }}"
    home: "{{ item.home|default(omit) }}"
    system: "{{ item.system|default(omit) }}"
    state: present
  with_items: "{{ users }}"
```

Comment-Filter

Decorate the text with a chosen comment style

```
{{ ansible_managed | comment }}
```

-->

```
# ansible managed
```

- More comment-styles here.

Indent-Filter

Change indentation of variable.

Variables:

```
ssh_server_match_group:
  - group: 'root'
    rules: 'AllowTcpForwarding yes' Template:

{% for item in ssh_server_match_group %}
Match Group {{ item.group }}
    {{ item.rules | indent(4,true) }}
{% endfor %}
{% endif %}
```


Outcome:

Advanced Example

Variables:

```
ssh_ciphers:
  - chacha20-poly1305@openssh.com
  - aes256-gcm@openssh.com
  - aes128-gcm@openssh.com
```

Template:

```
{{ "Ciphers "+ ssh_ciphers | sort | join(',') if ssh_ciphers else "Ciphers"|comment }}
```

Advanced Example

Outcome:

```
Ciphers aes128-ctr,aes128-gcm@openssh.com,aes192-ctr,aes256-ctr,aes256-gcm@openssh.com,chacha20-poly1305@openssh.com
```

Author and License

- Author: Sebastian Gumprich - freelance@gumpri.ch