Ansible Best Practices

How to write, how to execute, and how to use in real life
How to use
Treat your Ansible content like code

- Version control your Ansible content
- Iterate
  - Start with a basic playbook and static inventory
  - Refactor and modularize later
Do it with style

- Create a style guide for consistency:
  - Tagging
  - Whitespace
  - Naming of Tasks, Plays, Variables, and Roles
  - Directory Layouts
- Enforce the style
- Nice example: openshift-ansible Style Guide
  example: https://goo.gl/JfWBCW
CODE MUST BE ORGANIZED

USE GIT!
Do it with style

site.yml  # master playbook, calling others
webservers.yml  # playbook for webserver tier
deployonce.yml  # separate playbook for single-shot tasks
inventories/
    production/
        hosts
        group_vars/
        host_vars/
    london/
roles/
    requirements.yml  # includes roles from some other place
common/
webtier/
    # inventory file for production servers
    # different stages via inventory
    # additional, alternative grouping if useful
    # base line, company wide configuration
Start with one Git repository - but when it grows, use multiple!

At the beginning: put everything in one Git repository

In the long term:
- One Git repository per role
- Dedicated repositories for completely separated teams / tasks

SO, WHAT DO WE HAVE?
Give inventory nodes human-meaningful names rather than IPs or DNS hostnames.

- 10.1.2.75  db1 ansible_host=10.1.2.75
- 10.1.5.45  db2 ansible_host=10.1.5.45
- 10.1.4.5   db3 ansible_host=10.1.4.5
- 10.1.0.40  db4 ansible_host=10.1.0.40
- w14301.acme.com  web1 ansible_host=w14301.acme.com
- w17802.acme.com  web2 ansible_host=w17802.acme.com
- w19203.acme.com  web3 ansible_host=w19203.acme.com
- w19304.acme.com  web4 ansible_host=w19203.acme.com
Group hosts for easier inventory selection and less conditional tasks -- the more the better.

<table>
<thead>
<tr>
<th>Group</th>
<th>DB</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>[db]</td>
<td>db1</td>
<td>web1</td>
</tr>
<tr>
<td>[web]</td>
<td>db3</td>
<td>web3</td>
</tr>
<tr>
<td>[east]</td>
<td>db1</td>
<td>web1</td>
</tr>
<tr>
<td>[west]</td>
<td>db2</td>
<td>web2</td>
</tr>
<tr>
<td>[dev]</td>
<td>db1</td>
<td>web1</td>
</tr>
<tr>
<td>[testing]</td>
<td>db3</td>
<td>web3</td>
</tr>
<tr>
<td>[prod]</td>
<td>db2</td>
<td>web2</td>
</tr>
<tr>
<td></td>
<td>db4</td>
<td>web4</td>
</tr>
</tbody>
</table>
Use dynamic sources where possible. Either as a single source of truth - or let Ansible unify multiple sources.

- Stay in sync automatically
- Reduce human error
- No lag when changes occur
- Let others manage the inventory
VARIABLES
JUST WORDS, RIGHT?
Proper variable names can make plays more readable and avoid variable name conflicts

<table>
<thead>
<tr>
<th>a: 25</th>
<th>apache_max_keepalive: 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>data: ab</td>
<td>apache_port: 80</td>
</tr>
<tr>
<td>data2: abc</td>
<td>tomcat_port: 8080</td>
</tr>
<tr>
<td>id: 123</td>
<td></td>
</tr>
</tbody>
</table>
Avoid collisions and confusion by adding the role name to a variable as a prefix.

apache_max_keepalive: 25
apache_port: 80
tomcat_port: 8080
PLACE VARIABLES APPROPRIATELY

Know where your variables are

- Find the appropriate place for your variables based on what, where and when they are set or modified
- Separate logic (tasks) from variables and reduce repetitive patterns
- Do not use every possibility to store variables - settle to a defined scheme and as few places as possible
MAKE YOUR PLAYBOOK READABLE
- name: install telegraf
  yum: name=telegraf-{{ telegraf_version }} state=present update_cache=yes
  notify: restart telegraf

- name: start telegraf
  service: name=telegraf state=started
Better, but no

- **name: install telegraf**
  yum: >
    name=telegraf-{{ telegraf_version }}
    state=present
    update_cache=yes
    enablerepo=telegraf
    notify: restart telegraf

- **name: start telegraf**
  service: name=telegraf state=started
Yes!

- name: install telegraf
  yum:
   name: “telegraf-{{ telegraf_version }}”
   state: present
   update_cache: yes
   enablerepo: telegraf
   notify: restart telegraf

- name: start telegraf
  service:
   name: telegraf
   state: started
PLAY [web]

********************************

TASK [setup]

****************************************

ok: [web1]

TASK [yum]

****************************************

ok: [web1]

TASK [service]

****************************************

ok: [web1]
Exhibit B

- hosts: web
  name: installs and starts apache

  tasks:
  - name: install apache packages
    yum:
      name: httpd
      state: latest
  - name: starts apache service
    service:
      name: httpd
      state: started
      enabled: yes

PLAY [install and starts apache]
*******************************

TASK [setup]
*******************************
ok: [web1]

TASK [install apache packages]
*******************************
ok: [web1]

TASK [starts apache service]
*******************************
ok: [web1]
Blocks can help in organizing code, but also enable rollbacks or output data for critical changes.

```yaml
- block:
  copy:
    src: critical.conf
    dest: /etc/critical/crit.conf
  service:
    name: critical
    state: restarted
  rescue:
    command: shutdown -h now
```
EXECUTING THE ANSIBLE COMMANDS

How to execute
Ansible provides multiple switches for command line interaction and troubleshooting.

- vvvv
- --step
- --check
- --diff
- --start-at-task
Ansible has switches to show you what will be done

Use the power of included options:
--list-tasks
--list-tags
--list-hosts
--syntax-check
If there is a need to launch something without an inventory - just do it!

- For single tasks - note the comma:
  ```
  ansible all -i neon.qxyz.de, -m service -a "name=redhat state=present"
  ```

- For playbooks - again, note the comma:
  ```
  ansible-playbook -i neon.qxyz.de, site.yml
  ```
Don’t just start services -- use smoke tests

- name: check for proper response
  uri:
    url: http://localhost/myapp
    return_content: yes
  register: result
  until: "Hello World" in result.content
  retries: 10
  delay: 1
Try to avoid the command module - always seek out a module first

- name: add user
  command: useradd appuser

- name: install apache
  command: yum install httpd

- name: start apache
  shell: |
    service httpd start && chkconfig httpd on

- name: add user
  user:
    name: appuser
    state: present

- name: install apache
  yum:
    name: httpd
    state: latest

- name: start apache
  service:
    name: httpd
    state: started
    enabled: yes
If managed files are not marked, they might be overwritten accidentally

- Label template output files as being generated by Ansible
- Use the ansible_managed** variable with the comment filter

```python
{{ ansible_managed | comment }}
```
Roles enable you to encapsulate your operations.

- Like playbooks -- keep roles purpose and function focused
- Store roles each in a dedicated Git repository
- Include roles via `roles/requirements.yml` file, import via `ansible-galaxy` tool
- Limit role dependencies
Get roles from Galaxy, but be careful and adopt them to your needs

- Galaxy provides thousands of roles
- Quality varies drastically
- Take them with a grain of salt
- Pick trusted or well known authors
ACCESS RIGHTS
Root access is harder to track than sudo - use sudo wherever possible

- Ansible can be run as root only
- But login and security reasons often request non-root access
- Use become method - so Ansible scripts are executed via sudo (sudo is easy to track)
- Best: create an Ansible only user
- Don’t try to limit sudo rights to certain commands - Ansible does not work that way!
DEBUG YOUR PROBLEM
Check logging on target machine

ansible-node sshd[2395]: pam_unix(sshd:session): session opened for user liquidat by (uid=0)
ansible-node ansible-yum[2399]: Invoked with name=['httpd']
   list=None install_repoquery=True conf_file=None
disable_gpg_check=False state=absent disablerepo=None
update_cache=False enablerepo=None exclude=None
How to keep the code executed on the target machine

Look into the logging of your target machine

$ ANSIBLE_KEEP_REMOTE_FILES=1 ansible target-node -m yum -a "name=httpd state=absent"

Execute with:

$ /bin/sh -c 'sudo -u $SUDO_USER /bin/sh -c "/usr/bin/python /home/liquidat/.ansible/tmp/..."
Debugging tasks can clutter the output, apply some housekeeping

- name: Output debug message
  debug:
    msg: "This always displays"

- name: Output debug message
  debug:
    msg: "This only displays with ansible-playbook -vv+"
    verbosity: 2
How to use in real life
Simple: Use Tower.

- Tower was developed with Ansible in mind
- Extends the limits of Ansible to meet enterprise needs: Scalability, API, RBAC, audits, etc.
Tower has inbuilt help

- Tower provides in-program help via questionmark bubbles
- Can include examples or links to further docs
BRANCHES, ANYONE?
Tower can import a repository multiple times with different branches

- Use feature or staging branches in your Git
- Import them all separately, address them separately
- Useful for testing of new features but also to move changes through stages
MANY, MANY ROLES
Tower automatically imports Roles during Project update

- Do not copy roles into your playbook repository, just create a `roles/requirements.yml`
- Tower will automatically import the roles during Project installation
- Mix roles from various sources
- Fix version in `roles/requirements.yml` to have auditable environment!
WHAT ARE WE TALKING TO?
Use dynamic & smart inventories

- Combine multiple inventory types
- Let Tower take care of syncing and caching
- Use smart inventories to group nodes
DOING GOOD JOBS
Tower job templates provide multiple options - use them wisely

- Keep jobs simple, focussed - as playbooks or roles
- Add labels to them to better filter
- For idempotent jobs, create “check” templates as well - and let them run over night
- Combine with notifications - and get feedback when a “check” failed
Multiple playbooks can be combined into one workflow

- Simple jobs, complex workflows
- React to problems via workflow
- Combine playbooks of different teams, different repositories
- Re-sync inventories during the play
DO ASK PROPER QUESTIONS
Use surveys to get variable values

- Use good, meaningful variable names
- Provide a default choice
- Multiple choice > free text
- If answer not required - do you really need it at all?
A POWERFUL TEAM
Tower provides tenants, teams, and users - use them for separation

- Provide automation to others without exposing credentials
- Let others only see what they really need
- Use personal view instead of full Tower interface
ONE KEY TO RULE THEM ALL ...
Tower credentials should only be used by Tower - not by others

- Set up a separate user and password/key for Tower
- That way, automation can easily be identified on target machines
- The key/password can be ridiculously complicated and secure
- Store key/password in a safe for emergencies
Tower can send notifications if a job succeeds, fails or always - as mail, IRC, web hook, and so on

- Let Tower notify you and your team if something breaks
- Send mails/web-hooks automatically to a ticket systems and monitoring if there is a serious problem
LOGS, ANYONE?
Send all logs from Tower to central logging

- Splunk, Loggly, ELK, REST
- Send results from Ansible runs - but also from Tower changes
ALWAYS KEEP THE LIGHTS ON
Tower can be easily set up HA - and for restricted networks, deploy isolated nodes

- Make Tower HA - it is easy! (Well, except the DB part maybe....)
- For distant or restricted networks, use isolated nodes
Thank you

Red Hat is the world’s leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.