ANSIBLE BEST PRACTICES: THE ESSENTIALS

Timothy Appnel
Senior Product Manager, Ansible

GitHub: tima
Twitter: appnelgroup
COMPLEXITY KILLS PRODUCTIVITY

That's not just a marketing slogan. We really mean it and believe that. We strive to reduce complexity in how we've designed Ansible tools and encourage you to do the same. **Strive for simplification in what you automate.**
OPTIMIZE FOR READABILITY

If done properly, it can be the documentation of your workflow automation.
Principal 3
THINK DECLARATIVELY

Ansible is a desired state engine by design. If you're trying to "write code" in your plays and roles, you're setting yourself up for failure. Our YAML-based playbooks were never meant to be for programming.
Treat your Ansible content like code

- Version control your Ansible content
- Start as simple as possible and iterate
  - Start with a basic playbook and static inventory
  - Refactor and modularize later
Do It with Style

- Create a style guide for developers
- Consistency in:
  - Tagging
  - Whitespace
  - Naming of Tasks, Plays, Variables, and Roles
  - Directory Layouts
- Enforce the style
basic-project
├── inventory
│   ├── group_vars
│   │   └── web.yml
│   ├── host_vars
│   │   └── db1.yml
│   └── hosts
└── site.yml
PROJECT LAYOUTS: ORGANIZATIONAL ROLES

```
myapp
├── roles
│   ├── myapp
│   │   ├── tasks
│   │   │   └── main.yml
│   │   └── ...
│   └── nginx
│       └── ...
└── proxy
    └── ...
```

site.yml
myapp
├── config.yml
├── provision.yml
├── roles
│   └── requirements.yml
└── site.yml
## INVENTORY

Give inventory nodes human-meaningful

### EXHIBIT A

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Ansible Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.2.75</td>
<td>ansible_host=10.1.2.75</td>
</tr>
<tr>
<td>10.1.5.45</td>
<td>ansible_host=10.1.5.45</td>
</tr>
<tr>
<td>10.1.4.5</td>
<td>ansible_host=10.1.4.5</td>
</tr>
<tr>
<td>10.1.0.40</td>
<td>ansible_host=10.1.0.40</td>
</tr>
<tr>
<td>w14301.example.com</td>
<td>ansible_host=w14301.example.com</td>
</tr>
<tr>
<td>w17802.example.com</td>
<td>ansible_host=w17802.example.com</td>
</tr>
<tr>
<td>w19203.example.com</td>
<td>ansible_host=w19203.example.com</td>
</tr>
<tr>
<td>w19304.example.com</td>
<td>ansible_host=w19304.example.com</td>
</tr>
</tbody>
</table>

### EXHIBIT B
Group hosts for easier inventory selection and less conditional tasks -- the more groups the better.

<table>
<thead>
<tr>
<th>WHAT</th>
<th>WHERE</th>
<th>WHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>[db]</td>
<td>[east]</td>
<td>[dev]</td>
</tr>
<tr>
<td>db[1:4]</td>
<td>db1</td>
<td>db1</td>
</tr>
<tr>
<td></td>
<td>web1</td>
<td>web1</td>
</tr>
<tr>
<td>[web]</td>
<td>[west]</td>
<td>[test]</td>
</tr>
<tr>
<td>web[1:4]</td>
<td>db3</td>
<td>db3</td>
</tr>
<tr>
<td></td>
<td>web3</td>
<td>web3</td>
</tr>
<tr>
<td></td>
<td>[west]</td>
<td>[prod]</td>
</tr>
<tr>
<td></td>
<td>db2</td>
<td>db2</td>
</tr>
<tr>
<td></td>
<td>web2</td>
<td>web2</td>
</tr>
<tr>
<td></td>
<td>db4</td>
<td>db4</td>
</tr>
<tr>
<td></td>
<td>web4</td>
<td>web4</td>
</tr>
<tr>
<td>db1 = db, east, dev</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Use a single source of truth if you have it -- even if you have multiple sources, Ansible can unify them.

- Stay in sync automatically
- Reduce human error
Proper variable naming can make plays more readable and avoid variable name conflicts

- Use descriptive, unique human-meaningful variable names
- Prefix role variables with its “owner” such as a role name or package

```yaml
apache_max_keepalive: 25
apache_port: 80
tomcat_port: 8080
```
Make the most of variables

- Find the appropriate place for your variables based on what, where and when they are set or modified
- Separate logic (tasks) from variables to reduce repetitive patterns and provided added flexibility.
- name: Clone student lesson app for a user
  host: nodes
  tasks:
    - name: Create ssh dir
      file:
        state: directory
        path: /home/{{ username }}/ssh
    - name: Set Deployment Key
      copy:
        src: files/deploy_key
        dest: /home/{{ username }}/ssh/id_rsa
    - name: Clone repo
      git:
        accept_hostkey: yes
        clone: yes
        dest: /home/{{ username }}/exampleapp
        key_file: /home/{{ username }}/ssh/id_rsa
        repo: git@github.com:example/apprepo.git

EXHIBIT A

● Embedded parameter values and repetitive home directory value pattern in multiple places
● Works but could be more clearer and setup to be more flexible and maintainable
- name: Clone student lesson app for a user
  host: nodes
  vars:
    user_home_dir: /home/{{ username }}
    user_ssh_dir: "{{ user_home_dir }}/\.ssh"
    deploy_key: "{{ user_ssh_dir }}/id_rsa"
    app_dir: "{{ user_home_dir }}/exampleapp"
  tasks:
    - name: Create ssh dir
      file:
        state: directory
        path: "{{ user_ssh_dir }}"
    - name: Set Deployment Key
      copy:
        src: files/deploy_key
        dest: "{{ deploy_key }}"
    - name: Clone repo
      git:
        dest: "{{ app_dir }}"
        key_file: "{{ deploy_key }}"
        repo: git@github.com:example/exampleapp.git
        accept_hostkey: yes
        clone: yes

**EXHIBIT B**

- Parameters values are set thru values away from the task and can be overridden.
- Human meaningful variables “document” what’s getting plugged into a task parameter
- More easily refactored into a role
Use native YAML syntax to maximize the readability of your plays

- Vertical reading is easier
- Supports complex parameter values
- Works better with editor syntax highlighting in editors
NO!

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

- name: configure telegraf
  template: src=telegraf.conf.j2 dest=/etc/telegraf/telegraf.conf

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

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

- name: configure telegraf
  template: src=telegraf.conf.j2 dest=/etc/telegraf/telegraf.conf

- name: start telegraf
  service: name=telegraf state=started enabled=yes
Yes!

- name: install telegraf
  yum:
    name: telegraf-{{ telegraf_version }}
    state: present
    update_cache: yes
    disable_gpg_check: yes
    enablingrepo: telegraf
  notify: restart telegraf

- name: configure telegraf
  template:
    src: telegraf.conf.j2
    dest: /etc/telegraf/telegraf.conf
  notify: restart telegraf

- name: start telegraf
  service:
    name: telegraf
    state: started
    enabled: yes
Names improve readability and user feedback

- Give all your playbooks, tasks and blocks brief, reasonably unique and human-meaningful names
- hosts: web
  tasks:
  - yum:
    name: httpd
    state: latest
  - service:
    name: httpd
    state: started
    enabled: yes

PLAY [web]
******************************************************************************

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

TASK [yum]
******************************************************************************
ok: [web1]

TASK [service]
******************************************************************************
ok: [web1]
PLAY [install and start apache]
*****************************************************

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

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

TASK [start apache service]
****************************
ok: [web1]
Focus avoids complexity

- Keep plays and playbooks focused. Multiple simple ones are better than having a huge single playbook full of conditionals
- Follow Linux principle of do one thing, and one thing well
Clean up your debugging tasks

- Make them optional with the verbosity parameter so they're only displayed when they are wanted.

  - debug:
    msg: "This always displays"
  
  - debug:
    msg: "This only displays with ansible-playbook -vv+"
    verbosity: 2
Don’t just start services -- use smoke tests

```yaml
- 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
```
Use command modules sparingly

- Use the run `command` modules like `shell` and `command` as a last resort
- The `command` module is generally safer
- The `shell` module should only be used for I/O redirect
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
Still using command modules a lot?

```yaml
- hosts: all
  vars:
    cert_store: /etc/mycerts
    cert_name: my cert
  tasks:
  - name: check cert
    shell: certify --list --name={{ cert_name }} --cert_store={{ cert_store }} | grep "{{ cert_name }}"
    register: output

  - name: create cert
    command: certify --create --user=chris --name={{ cert_name }} --cert_store={{ cert_store }}
    when: output.stdout.find(cert_name) != -1
    register: output

  - name: sign cert
    command: certify --sign --name={{ cert_name }} --cert_store={{ cert_store }}
    when: output.stdout.find("created") != -1
```
Develop your own module

- hosts: all
  vars:
    cert_store: /etc/mycerts
    cert_name: my cert
  tasks:
    - name: create and sign cert
      certify:
        state: present
        sign: yes
        user: chris
        name: "{{ cert_name }}"
        cert_store: "{{ cert_store }}"

- Understandable by non-technical people
- CRUD (Create, read, update and delete)
Separate provisioning from deployment and configuration tasks

```
acme_corp/
├── configure.yml
├── provision.yml
└── site.yml

$ cat site.yml
---
- import_playbook: provision.yml
- import_playbook: configure.yml
```
Jinja2 is powerful but you needn't use all of it

- Templates should be simple:
  - Variable substitution
  - Conditionals
  - Simple control structures/iterations
  - Design your templates for your use case, not the world's

- Things to avoid:
  - Anything that can be done directly in Ansible
  - Managing variables in a template
  - Extensive and intricate conditionals
  - Conditional logic based on embedded hostnames
  - Complex nested iterations
Careful when mixing manual and automated configuration

- Label template output files as being generated by Ansible

```
{{ ansible_managed | comment }}
```
● Like playbooks -- keep roles purpose and function focused
● Use a roles/ subdirectory for roles developed for organizational clarity in a single project
● Follow the Ansible Galaxy pattern for roles that are to be shared beyond a single project
● Limit role dependencies
● Use `ansible-galaxy init` to start your roles...
● ...then remove unneeded directories and stub files
● Use `ansible-galaxy` to install your roles -- even private ones
● Use a roles files (i.e. `requirements.yml`) to manifest any external roles your project is using
● Always peg a role to a specific version such as a tag or commit
Command line tools have their limitations

- Coordination across a distributed teams & organization...
- Controlling access to credentials...
- Track, audit and report automation and management activity...
- Provide self-service or delegation...
- Integrate automation with enterprise systems...
Thank you

Complexity Kills Productivity
Optimize For Readability
Think Declaratively