ANSIBLE

ANSIBLE BEST PRACTICES: THE ESSENTIALS

Asaf Hirshberg
Solution Architect,
Red Hat

redhat

THE ANSIBLE WAY

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.

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
- Iterate
  - Start with a basic Playbook and static inventory
  - Refactor and modularize later

Do it with style

- Create a style guide for your Playbook developers
- Consistency in:
  - Tagging
  - Whitespace
  - Naming of Tasks, Plays, Variables, and Roles
  - Directory Layouts
- Enforce the style

PROJECT LAYOUTS: BASIC


```
basic-project/
├── config.yml
├── inventory
│   ├── group_vars
│   ├── host_vars
│   └── hosts
├── provision.yml
└── site.yml
```

PROJECT LAYOUTS: ORGANIZATIONAL ROLES


```
myapp/
├── config.yml
├── provision.yml
├── roles
│   ├── myapp
│   │   ├── tasks
│   │   │   └── main.yml
│   │   └── etc.etc
│   ├── nginx
│   │   └── etc.etc
│   └── proxy
│       └── etc.etc
└── site.yml
```

PROJECT LAYOUTS: SHARED ROLES


```
myapp/
├── config.yml
├── provision.yml
├── roles
│   └── requirements.yml
└── setup.yml
```

Give inventory nodes human-meaningful names rather than IPs or DNS hostnames

```
10.1.2.75
10.1.5.45
10.1.4.5
10.1.0.40

w14301.acme.com
w17802.acme.com
w19203.acme.com
w19304.acme.com
```



```
db1 ansible_host=10.1.2.75
db2 ansible_host=10.1.5.45
db3 ansible_host=10.1.4.5
db4 ansible_host=10.1.0.40

web1 ansible_host=w14301.acme.com
web2 ansible_host=w17802.acme.com
web3 ansible_host=w19203.acme.com
web4 ansible_host=w19203.acme.com
```

Group hosts for easier inventory selection and less conditional tasks -- the more the better

```
[db]
db[1:4]

[web]
web[1:4]
```

```
[east]
db1
web1
db3
web3

[west]
db2
web2
db4
web4
```

```
[dev]
db1
web1

[testing]
db3
web3

[prod]
db2
web2
db4
web4
```

Use a single source of truth if you have it -- even if you multiple sources Ansible can unify them

- Stay in sync automatically
- Reduce human error

Proper variable names can make plays more readable and avoid variable name conflicts

- Use descriptive, unique human-meaningful variable names
- Prefix role variables with role name

```
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 and reduce repetitive patterns

SEPARATE LOGIC FROM VARIABLES


```
- 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 }}/lightbulb
        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

SEPARATE LOGIC FROM VARIABLES


```
- name: Clone student lesson app for a user
  host: nodes
  vars:
    user_home: /home/{{ username }}
    user_ssh: "{{ user_home }}/.ssh"
    deploy_key: "{{ user_ssh }}/id_rsa"
    app_dest: "{{ user_home }}/exampleapp"
  tasks:
    - name: Create ssh dir
      file:
        state: directory
        path: "{{ user_ssh }}"

    - name: Set Deployment Key
      copy:
        src: files/deploy_key
        dest: "{{ deploy_key }}"

    - name: Clone repo
      git:
        dest: "{{ app_dest }}"
        key_file: "{{ deploy_key }}"
        repo: git@github.com:example/exampleapp.git
        accept_hostkey: yes
        clone: yes
```

EXHIBIT B

- Parameter and home directory values are set thru values away from the task
- Human meaningful variables “document” what’s getting plugged into a parameter
- More easily refactored into a role

USE SYNTAX HIGHLIGHTING

- Vim
  - pearofducks/ansible-vim
  - Glench/Vim-Jinja2-Syntax
- Sublime
- Atom
- Emacs
- PyCharm

site.yml

```
---
- hosts: web
  name: This is a play within a playbook
  become: yes
  vars:
    httpd_packages:
      - httpd
      - mod_wsgi
    apache_test_message: This is a test message
    apache_max_keep_alive_requests: 115

  tasks:
    - name: install libselinux-python
      yum:
        name: libselinux-python
        state: present
      tags: package

    - name: install httpd packages
      yum:
        name: "{{ item }}"
        state: latest
      with_items: "{{ httpd_packages }}"
      notify: restart apache service
      tags: package
```

Maximize the readability of your plays

- Use native YAML syntax
  - Vertical reading is easier
  - Supports complex parameter values
  - Works better with editor syntax highlighting

USE NATIVE YAML SYNTAX

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
       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
```

USE NATIVE YAML SYNTAX

Yes!

```
- 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
  notify: restart telegraf

- name: start telegraf
  service:
    name: telegraf
    state: started
    enabled: yes
```

Names improve readability and user feedback

- Give all your Playbooks and tasks brief, reasonably unique and human-meaningful names

EXHIBIT A

```
- 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]
```

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]
```

Focus avoids complexity

- Keep plays and Playbooks focused. Multiple simple ones are better than having a huge single playbook full of conditionals

Separate provisioning from deployment and configuration tasks

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

$ cat site.yml
---
- include: provision.yml
- include: configure.yml
```

Clean up your debugging tasks

- Remove your debug tasks in production or make them optional with the verbosity param in v2.1

```
- debug:
    msg: "This always displays"

- debug:
    msg: "This only displays with ansible-playbook -vv+"
    verbosity: 2
```

Use run commands sparingly

- Use the run command modules like `shell` and `command` as a last resort
- Use the `command` module unless you really need the pipelining that `shell` permits -- but be careful

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 run commands a lot?

```
- 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 }}"
```

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
```

Jinja2 is powerful but you needn't use all of it

- Templates should be simple:
  - Variable substitution
  - Conditionals
  - Simple control structures/iterations
  - Design for your use case, not the world's
- Things to avoid:
  - Managing variables in a template
  - Extensive and intricate conditionals
  - Conditional logic based on hostnames
  - Complex nested iterations

Jinja2 is powerful but you needn't use all of it

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

```
{{ 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 organization...
- Controlling access to credentials...
- Track, audit and report Ansible usage...
- Provide self-service or delegation...
- Integrate Ansible with enterprise systems...

THANK YOU!!!

AND JOIN US
AT THE
TECH-LAB
ON THE

16.7.17