ANSIBLE

ANSIBLE BEST PRACTICES: ROLES & MODULES

Tim Appnel, Senior Product Manager
GitHub: tima
Twitter: appnelgroup

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.

ROLES + MODULES

Use the right tool for the job

ROLES

- Self-contained portable units of Ansible automation
- Expressed in YAML and bundled with associated assets
- Decoupled from assumptions made by plays

MODULES

- Small “programs” that perform actions on remote hosts or on their behalf
- Expressed as code
  - i.e. Python, PowerShell
- Called by an Ansible task
- Modules do all of the heavy lifting in Ansible

ROLES + MODULES

Use the right tool for the job

ROLES

- Reuse and collaboration of common automation workflows & configurations
- Provide full life-cycle management of a service, microservice or container
- “De-facto” enforcement of standards and policies

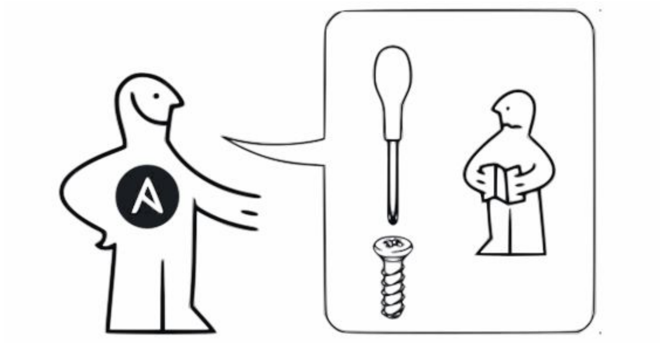
MODULES

- Sophisticated interactions and logic of a unit of work usually with a command line tool or APIs
- Abstract complexity away from users to make powerful automation simple

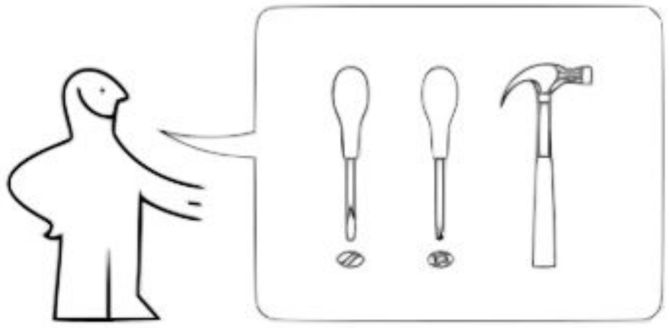
ROLES + MODULES

Use the right tool for the job

ROLES



MODULES



BEST PRACTICES: ROLES

ROLES ARE ANSIBLE CONTENT

The same best practices for your plays still apply

- Use native YAML syntax
- Version control your Ansible content
- Use command modules sparingly
- Always seek out a module first
- “Name” your plays, blocks and tasks
- Use human meaningful names with variables, hosts, etc.
- Clean up your debugging messages

ROLE DESIGN

Keep the purpose and function of a role self-contained and focused to do one thing well

- Think about the full life-cycle of a service, microservice or container — not a whole stack or environment
- Keep provisioning separate from configuration and app deployment

- Roles are not classes or object or libraries – those are programming constructs
- Keep roles loosely-coupled — limit hard dependencies on other roles or external variables

EXHIBIT A

```yaml
# blackbox_role_playbook.yml
---
- hosts: all
  roles:
    - umbrella_corp_stack
```

EXHIBIT B

```yaml
# componentized_roles_playbook.yml
---
- hosts: localhost
  roles:
    - azure_provisioner
- hosts: all
  roles:
    - system_security
- hosts: webservers
  roles:
    - python_common
    - python_django
    - nginx_uwsgi
    - racoon_app
- hosts: databases
  roles:
    - pgsql-replication
```

ROLE DESIGN

Maximize your role design for portability and reuse

- Use `ansible-galaxy` to install your roles
- Use a roles files (i.e. `requirements.yml`) to manifest your project roles
- When using a shared role always declare a specific version such as a tag or commit

```yaml
# requirements.yml
---
- src: nginxinc.nginx
  version: 0.8.0
- src: samdoran.pgsql-replication
  version: b5013e6
- src: geerlingguy.firewall
  version: 2.4.0
```

ROLE USABILITY

Roles should run with as few, if any, parameter variables as possible

- Practice convention over configuration
- Provide sane defaults
- Use variable parameters to modify default behaviour
- Easier to develop, test and use quickly & securely
- A role should always be more than a single task file

ANSIBLE

EXHIBIT A

```yaml
# defaults_no_playbook.yml
---
- hosts: webservers
  roles:
    - role: apache_simple
      apache_http_port: 80
      apache_doc_root: /var/www/html
      apache_user: apache
      apache_group: apache
    - role: apache_simple
      apache_http_port: 8080
      apache_doc_root: /www/example.com
      apache_user: apache
      apache_group: apache
```

EXHIBIT B

```yaml
# defaults_yes_playbook.yml
---
- hosts: webservers
  roles:
    - role: apache_simple
    - role: apache_simple
      apache_http_port: 8080
      apache_doc_root: /www/example.com
```

```yaml
# default/main.yml
---
apache_http_port: 80
apache_doc_root: /var/www/html
apache_user: apache
apache_group: apache
```

redhat.

ROLE USABILITY

Use variables in your roles appropriately

- `defaults/` are easy to override and most commonly used to modify behavior
  - i.e. port number or default user

- `vars/` are used by the role and not likely to be changed
  - i.e. a list of packages, lookup table of machine images by region

```yaml
# default/main.yml
---
apache_http_port: 80
apache_doc_root: /var/www/html
apache_user: apache
apache_group: apache
```

```yaml
# vars/main.yml
---
apache_packages:
  redhat:
    - httpd
    - mod_wsgi
  debian:
    - apache2
    - libapache2-mod-wsgi
```

ROLE USABILITY

Automate the testing of your roles

Use molecule, a testing framework designed to aid in the development and testing of Ansible Roles.

Initially developed by the community, led by John Dewey of Cisco, and adopted by Red Hat as an official Ansible project.

https://github.com/ansible/molecule

ROLE USABILITY

Automate the testing of your roles

Use ansible-lint, a command-line static analysis tool that checks playbooks and roles for identifying behaviour that could be improved.

Initially developed by Will Thames and recently adopted by Red Hat as an official Ansible project.

https://github.com/ansible/ansible-lint

HINT: ansible-lint can be run as part of your Molecule test runs.

ROLE READABILITY

Still using command modules a lot?

```
- name: check cert
  shell: certify --list --name={{ cert_name }} --cert_store={{ cert_store }} | grep "{{ cert_name }}"
  register: check_output

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

- name: sign cert
  command: certify --sign  --name={{ cert_name }} --cert_store={{ cert_store }}
  when: create_output.stdout.find("created") != -1
```

ROLE READABILITY

Develop your own module

```
- name: create and sign cert
  certify:
    state: present
    sign: yes
    user: chris
    name: "{{ cert_name }}"
    cert_store: "{{ cert_store }}"
```

BEST PRACTICES: MODULES

MODULE DESIGN

Good modules are user-centric

- Modules are how Ansible balances simple and powerful
- They implement common automation tasks for a user
- They make easy tasks easier and complicated tasks possible
- They abstract users from having to know the details to get things done
- They are **not** one-to-one mapping of an API or command line tool interface
    - This is why you should not auto-generate your modules
- They are **not** monolithic “does everything” modules that are hard to understand and complicated to use correctly

MODULE IMPLEMENTATION

Making the powerful simple starts with the implementation

- No side-effects with multiple runs
- Err on the side of safety
  - i.e. use a temporary file and `atomic_move` when writing to a file
- Fail fast —— immediately detect and report failure conditions
- Support check mode
- Minimal use of dependencies

Modules should provide a predictable user interface

- Think desired state, think declaratively
- Avoid action or command parameters
- Keep parameters focused and narrowly defined — refrain from parameters that take complex data structures
- Parameter names should be in lowercase and use underscores:

```
update_cache   # YES
UpdateCache    # NO
updateCache    # NO
```

MODULES INTERFACE

Modules should provide a predictable user interface

- Normalize common parameter names with other modules such as:
  - name
  - state
  - dest
  - src
  - path
  - username
  - password

For your consideration...

- kubernetes
  - monolithic and requires expert knowledge of k8s
- ansible-kubernetes-modules
  - fine grained API mapping that is autogenerated
- k8s
  - better implementation but complex parameters abound and expert knowledge still required
- k8s_scale
  - more focused on a specific task – more of this please

MODULE RESPONSES

Provide informative and consistent responses

- Be consistent in what you return
- Make response data reusable by a play or role
- Return only relevant output – no logs files please
- Accurately report changed status
- Support diff mode if applicable – and return the diff conditionally

MODULE EXCEPTION HANDLING

Handle errors gracefully and predictably

- Apply defensive programming
- Fail fast – validate upfront and use the built-in argument spec function
- Fail predictably and informatively when errors happen
- Avoid catch all exceptions

Don't reinvent the wheel

- Make use of `module_utils/` -- they're your friends
  - `basic.py`
  - `api.py`
  - `facts/`
  - `urls.py`
  - `six.py`
  - noteworthy others: `ec2.py`, `docker.py`, `database.py`, `mysql.py`, `powershell.ps1` – and many many more!

Documentation is a requirement

- Examples should include the most common and real world usage
- Examples should be in native YAML syntax
- Return responses must be included and described
- Document your dependencies in the requirements section

MODULE TESTING

Test before you commit and push your code

- Utilize the testing tools in `ansible/hacking/`
  - `test-module`
  - `ansible-test sanity <MODULE_NAME>`
- Create roles and playbooks and to test and verify all your documented examples
  - `molecule`
- Test locally – not with the CI/CD system

MODULES IN THE WILD

(More) For your consideration...

- sysctl
  - a master class in writing a “best practice” module
- ping
  - the hello world of Ansible
- cron
  - module implementing an interface to a command line tool
- get_url
  - module implementing an interface to a python library

redhat.

MODULE & ACTION PLUGINS

Sometimes modules need something more

- Local controller execution of a module entirely possible or required...
- Special setup requirements before the module is dispatched to the host...
- Need to supplement a module with the services of another core module such as copy and a role won't cut it...

An Action Plugin executes on the controller and perform logic before dispatching a module

Thanks

MORE RESOURCES

Ansible Developers Guide

http://docs.ansible.com/ansible/devel/dev_guide/index.html