

Simply Ansible: Best Practices

| Writing Ansible scripts that don't completely suck

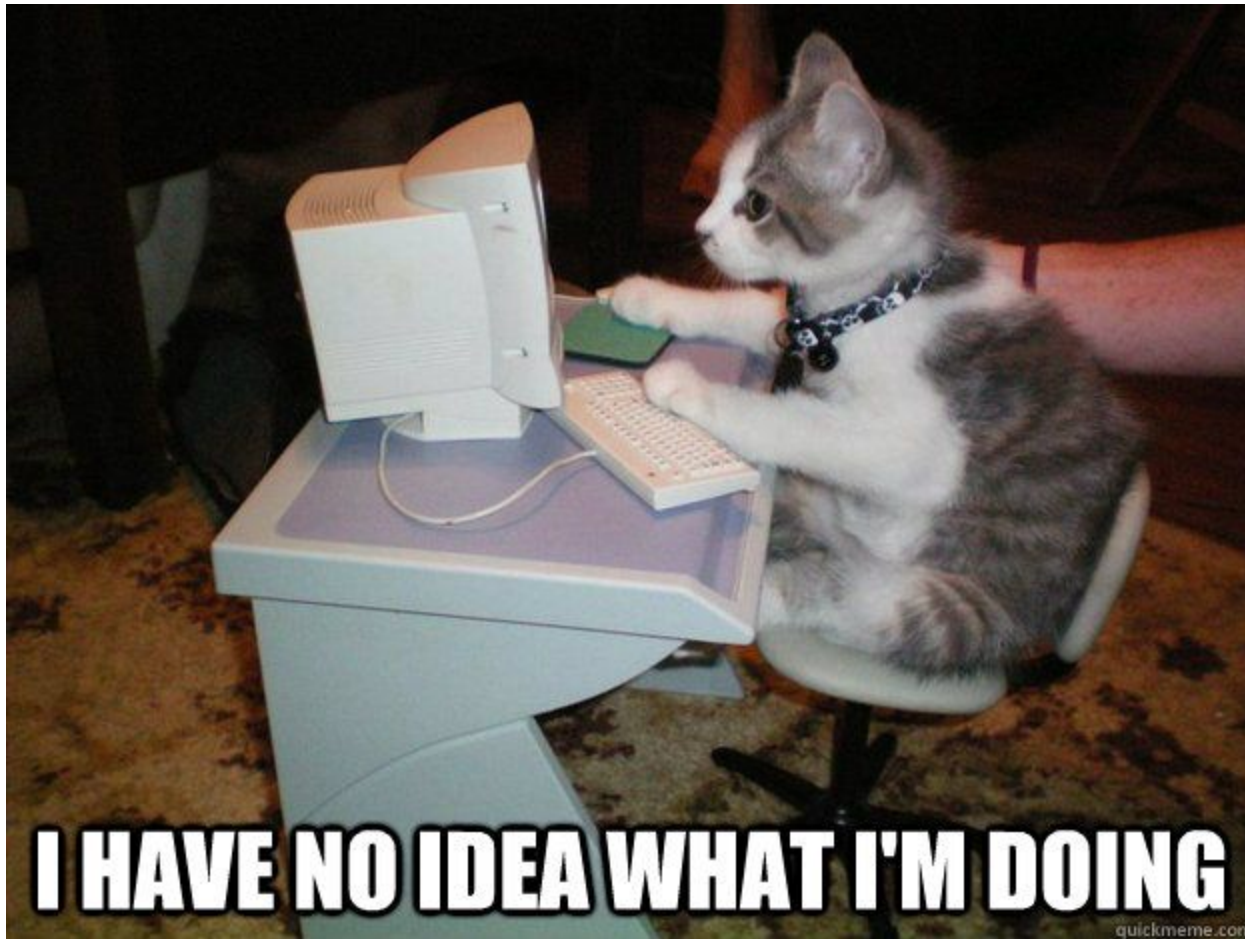
David M. Lee, II

[@leedm777](#) GitHub

[@leedm777](#) Twitter

leedm777@yahoo.com

Fair warning: I might be wrong



Nothing better than an example

I have a [cookiecutter](#) template which sets up a good starting project demonstrating these practices.

```
$ pip install cookiecutter
```

```
$ cookiecutter gh:building5/simply-ansible
```

PROTIPS



Put tasks in roles, not the playbook

While you can list tasks directly in a play, you're usually better off putting the task in a role.

```
# ./site.yml
- name: Some play
  hosts: some-hosts
  pre_tasks:           # meh
    - name: some pre_task
  roles:
    - some-role
  tasks:               # really?
    - name: some task
  post_tasks:          # PLEASE STOP!
    - name: some post_task
```

Really, keep tasks in roles

Besides, content from Ansible Galaxy can only be brought in as a role, so trying to sort that into existing `{pre,post}_tasks` is a pain.

```
- name: Some play
  hosts: some-hosts
  pre_tasks:
    # but what if I wanted to call a galaxy role
    # before this?
    - name: some pre_task
  post_tasks:
    - name: some post_task
    # or maybe a role after this?
```

Keep roles small and single purpose

If the playbook is your 'program', then roles are your 'functions'.

- For example, `pip` , `firewall` , `nodejs` , `timezone` , `auton`
- Not things like `common` , `general` , `misc` , ...
- Keep them small; most will fit on the screen

Bending the rule: debug-tools

I usually have a rule with a bunch of one-line installers for things that are generally useful for debugging. But if you can't install it with a single `apt`, `pip`, `npm`, `gem`, etc., then it gets its own role.

```
# ./roles/debug-tools/tasks/main.yml
- when: ansible_distribution == 'Ubuntu'
  name: apt-get install <cool-stuff>
  apt: name={{ item }}
  with_items:
    - curl
    - htop
    - iotop
    - jq
  # ...
```


Avoid repeating roles...

Here both `foo` and `bar` need `zygon`. Since plays are sequential, `zygon` will be installed in sequence. Can really slow down the playbook.

```
- hosts: foo
  roles:
    - zygon # Bad!
    - foo

- hosts: bar
  roles:
    - zygon # Bad!
    - bar
```

i.e., consolidate roles to get parallelism

Since the `zygon` role is in a single play, it will execute in parallel on both `foo` and `bar`. You can't always do this, but when you can, it helps tremendously.

```
- hosts: foo,bar # both foo and bar groups
  roles:
    - zygon      # installs in parallel
                  # across foo & bar nodes

- hosts: foo
  roles:
    - foo

- hosts: bar
  roles:
    - bar
```

Tag each role with the role's name

By keeping tag names consistent with role names, it's easier to keep track of what tags you have.

```
# ./roles/zygon/tasks/main.yml
---
- tags: zygon
  block: # Ansible 2.x syntax
    - name: First zygon invasion
      # ...
```

Since roles are small and single purpose, you're usually debugging individual roles.

```
$ ansible-playbook site.yml --tags zygon
```

Role tagging in Ansible 1.9

Before Ansible 2.x, the best way to tag an entire role was to put the tag in the playbook. Either way is fine.

```
roles:  
- { role: zygon, tags: zygon } # Ansible 1.9 syntax
```

Don't bother with meta

Meta declares dependencies between roles, which seems nice.

But...

- Dependencies are executed in every play they are referenced from
 - Which means they probably get executed multiple times
- Plays/roles execute in order, so you can just put the dependency earlier in the list

```
# ./role/rani/meta/main.yml
dependencies: [{ role: apt-get-update }] # Bad!
            # can cause apt-get-update to run
            # multiple times!
```

```
# ./site.yml
- roles:
  - apt-get-update # Much Better!
  - rani
```

Prefer defaults to vars

Roles give you two places to define variables. They only differ in precedence, and defaults (lowest precedence of any variable) is the least confusing.

```
roles/  
  some-role/  
    defaults/  
      main.yml # Good play for role's variables  
    vars/  
      main.yml # Confusing place for role's variables
```

Don't put tons of tags on a task

If you find yourself putting lots of tags on a task, you're probably doing it wrong. A tag should be there for a reason.

```
tags:
```

- provision *# okay, but every task in the
playbook is for provisioning*
- bamboo *# and everything is for bamboo*
- bamboo-agent
what are you doing?
- bamboo-agent-ubuntu
wait...
- bamboo-agent-ubuntu-precise
MAKE IT STOP
- bamboo-agent-ubuntu-precise-provision
*# *sigh**

Put variable definitions in the right spot

```
site.yml          # in the occasional vars block
                  # for parameterized roles

group_vars/
  all/
    main.yml      # default values
    ...           # can break into multiple files
  staging/        # Have a group for each environment
    main.yml      # override defaults for env
    vault.yml     # safe place for secrets
roles/
  kandy-man/
    defaults/
      main.yml    # specific vars for the role
                  # version numbers, SHA hashes, etc.
```


Vaults: too many vaults will kill you

- We tried having single-var per vault
 - PRO: Avoids merge conflicts w/ vaults
 - CON: No one follows the rules
 - Vars end up in random vaults
 - Or var was renamed and vault wasn't
 - Grep is either powerless, or really slow
- Use [ansible-vault-tools](#) and as few vaults as possible
 - Single vault per `group_vars`
 - `ansible-vault-merge` - magical vault merging
 - `ansible-vault-textconv` - magical vault diffing
 - And `git grep --textconv` can search in vaults!

!!!ENCRYPT YOUR VAULT PASSWORD!!!

The `ansible-vault-tools` project also provides `gpg-vault-password-file`, which can use your PGP key to encrypt your vault password.

- You do [have a PGP key](#), right?
- And it's [encrypted with a passphrase](#)?
- And you use [gpg-agent](#) so you don't go insane typing the passphrase all the time?

Put common settings in `ansible.cfg`

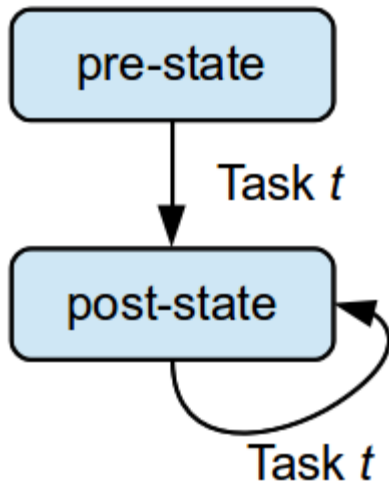
Defaults should be what's best for the humans that run the playbooks by hand the most (probably devs).

```
# ./ansible.cfg
[defaults]
# Why is the default not smart?
gathering = smart
# Default for most common params devs would use
inventory = ./inventory/vagrant.ini
# Consistent configuration for vault passwords
vault_password_file = $HOME/.ansible/some_project_vault

[privilege_escalation]
# Nearly everything requires sudo, so default to it
become = True

[ssh_connection]
# Speeds things up, but requires disabling
# requiretty in /etc/sudoers
pipelining = True
```

Make sure roles are idempotent



The concept that change commands should only be applied when they need to be applied, and that it is better to describe the desired state of a system than the process of how to get to that state.

-- [Ansible Glossary](#)

Handlers are not idempotent!

...but often necessary. Just be aware of this when a playbook fails.

```
# ./site.yml
- # ...
  roles: [ master, cyberman ]

# ./roles/master/tasks/main.yml
- cp: src=missy.conf dest=/etc/master/
  notify: restart master
  # ^^ schedules restart of master if file is copied

# ./roles/cyberman/tasks/main.yml
- fail: msg="***HEAD EXPLODES***"
  # ^^ causes play to fail before restart master
  # even if you fix and re-run, file already copied
  # so it won't notify on re-run
```

Use a wrapper script so everyone runs the playbook consistently

- Setup `virtualenv` so everyone runs the same version
- Add some common/useful options (`-v` , `--diff`)
- Add domain specific options, useful for your team
 - `--env` to specify environment
 - `--service` to deploy a specific service
 - `--` to break for `ansible-playbook` options

```
$ ./deploy.sh --env staging -- --tags cyberman --check
+ exec ansible-playbook -v --diff \
+   --inventory-file ./inventory/staging.ini \
+   --tags cyberman \
+   --check \
+   -- site.yml
Using /Users/dlee/prj/doctor/ansible.cfg as config file

PLAY [pre-roles] *****
```

Don't make a role portable until you have to

There is no such thing as portable code. Only code that's been ported.

Avoid supporting multiple platforms as much as you can. It's better to standardize on a distro/version.

That said,

- You usually end up with one weird thing that needs to be on another distro
- Or you'll have a transition time as you upgrade major versions
- Don't port everything; only port what you need; one thing at a time.

Logic goes in roles; not in playbooks

Ansible allows you to split logic between playbooks and roles, but that just makes it harder to figure out what's going on.

```
# ./site.yml
- # ...
  roles:
    - role: epel
      when: ansible_distribution == 'CentOS' # Bad!

# ./roles/epel/tasks/main.yml
- name: yum install epel-...
  when: ansible_distribution == 'CentOS' # Good!
```


Test your playbooks

- A `make test` target is always useful
- `ansible-playbook --syntax-check` finds syntax errors
- `ansible-lint` finds common errors

```
test: virtualenv galaxy
    @$(MAKE) check-env # errors if we're
                        # not in virtualenv
    ansible-playbook site.yml --syntax-check
    ansible-lint --exclude build/ site.yml
.PHONY: test
```

Follow good programming principles

- Use descriptive names
- Keep playbooks in source control
- Peer review changes before going live
- Use continuous deployment/continuous delivery/ChatOps
- Testing!

Questions



Thanks!



David M. Lee, II

[@leedm777](#) GitHub

[@leedm777](#) Twitter

leedm777@yahoo.com