ANSIBLE BEST PRACTICE

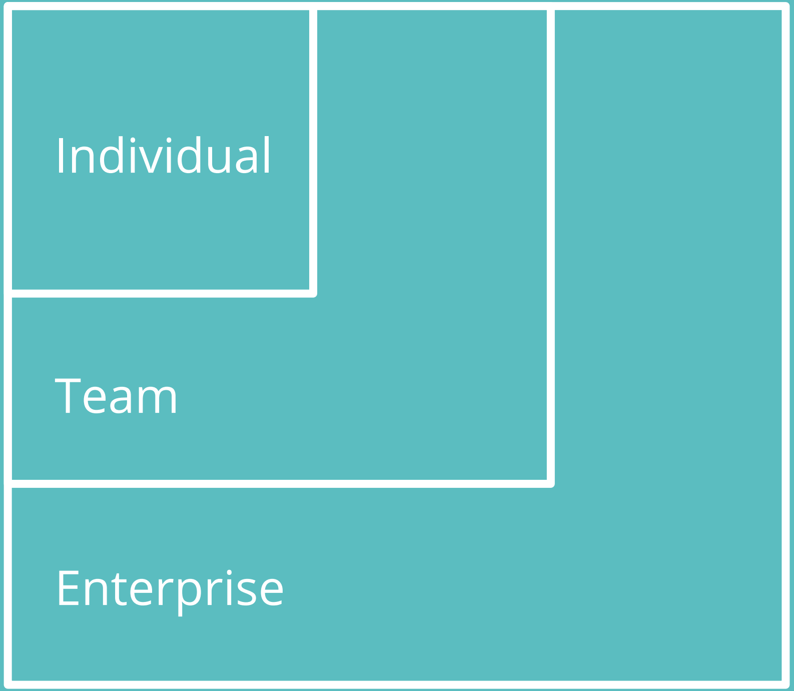
ANDREAS NEEB
SOLUTION ARCHITECT
RED HAT

aneeb@redhat.com
http://github.com/andyneeb/presentations

For God sakes...aren't best practices in 2019 dead?

Successful Automation

A Powerful Team



An enterprise-wide automation strategy must benefit individuals first.

START THE GROUNDSWELL

Starting with the BIG picture is not the best path to enlightenment

Start the revolution from your desk

Solving smaller problems in repeatable fashion is easier to unify

Look for quick wins, current gaps

Make easy but noticeable progress

Map out orchestration, workflows etc

Part 1: Principles

#1: 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.**

#2: OPTIMIZE FOR READABILITY

If done properly, it can be the documentation of your workflow automation.

#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.

Part 2: Practices

#1: Simplicity

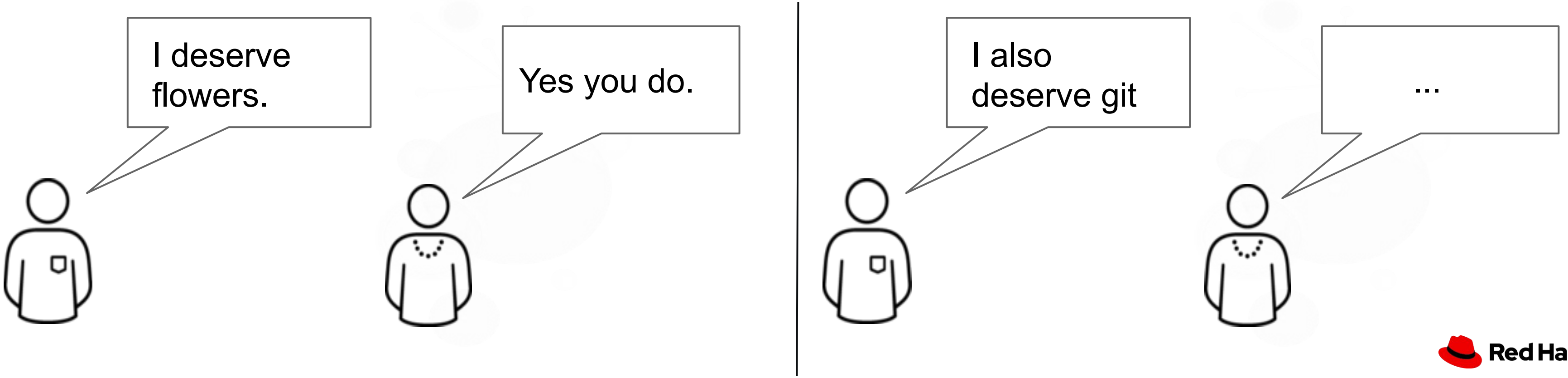
KISS

- Keep plays and playbooks focused. Multiple simple ones are better than having a huge single playbook full of conditionals
- Once a playbook gets long or you’re repeating tasks, use roles
- Follow Linux principle of do one thing, and one thing well

#2: Treat you Ansible content like code

Step 1: Version control your Ansible content

- Start as simple as possible and iterate
  - Start with a basic playbook and static inventory
  - Refactor and modularize later
- Start with one Git repository - but when it grows, use multiple!

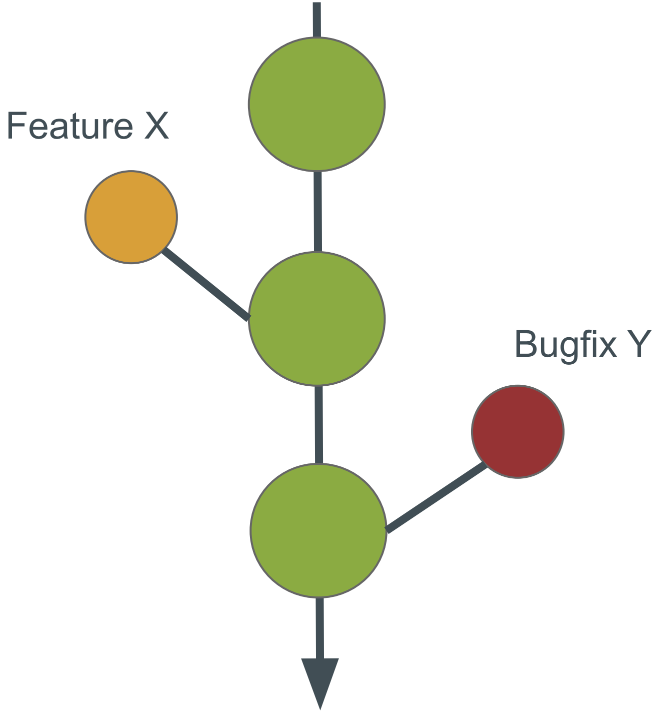


#2: Treat you Ansible content like code

ANSIBLE

Example: GitHub workflow

1. **Does not** require GitHub, the workflow model is just called that
2. **A** very simple workflow
3. **Master** branch is always possible to release
4. **Branches** are where you develop and test new features and bugfixes.
5. **Yes,** I wrote test. If you do not test your Ansible code you cannot keep the master branch releasable and this all fails.



Red Hat

#2: Treat you Ansible content like code

ANSIBLE

Step 2 - N: CI / CD

- Verify correct syntax (--syntax-check)
- Verify style for bad practices (ansible-lint)
- Run your playbook or role and ensure it completes without failures, run again, check idempotency
- Test, test, test (automated!)

Red Hat

#3: Inventory

Give inventory nodes human-meaningful

EXHIBIT A

```
10.1.2.75
10.1.5.45
10.1.4.5
10.1.0.40

w14301.example.com
w17802.example.com
w19203.example.com
w19304.example.com
```

EXHIBIT B

```
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.example.com
web2 ansible_host=w17802.example.com
web3 ansible_host=w19203.example.com
web4 ansible_host=w19203.example.com
```

#3: Inventory

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

WHAT

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

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

```
db1 = db, east, dev
```

WHERE

```
[east]
db1
web1
db3
web3

[west]
db2
web2
db4
web4
```

WHEN

```
[dev]
db1
web1

[test]
db3
web3

[prod]
db2
web2
db4
web4
```

#3: Inventory

Use dynamic sources where possible. Either a single source of truth or let Ansible unify them.

- Stay in sync automatically
- Reduce human error
- No lag when changes occur
- Let others manage the inventory

PUBLIC / PRIVATE CLOUD
CMDB

#4: Use Native YAML Syntax

No!

```
- name: install telegraf
  yum: name=telegraf-{{ telegraf_version }} state=present update_cache=yes disab
  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
```

#4: Use Native YAML Syntax

Yes!

```yaml
- 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
```

#5: Smoke Tests

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
```

#6: Modularity

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
```

#7: Idempotency

$f(f(x)) = f(x)$

- 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

Still using command a lot? Develop your own modules

#8: Variables

The world is flat - 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

```
apache_max_keepalive: 25
apache_port: 80
tomcat_port: 8080
```

- Do not use every possibility to store variables - settle to a defined scheme and as few places as possible
- Document in /defaults

#9: Task Names

No!

```
- 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]
```

#9: Task Names

Yes!

```yaml
- hosts: web
  name: install and start apache
  tasks:
    - name: install apache packages
      yum:
        name: httpd
        state: latest

    - name: start apache service
      service:
        name: httpd
        state: started
        enabled: yes
```

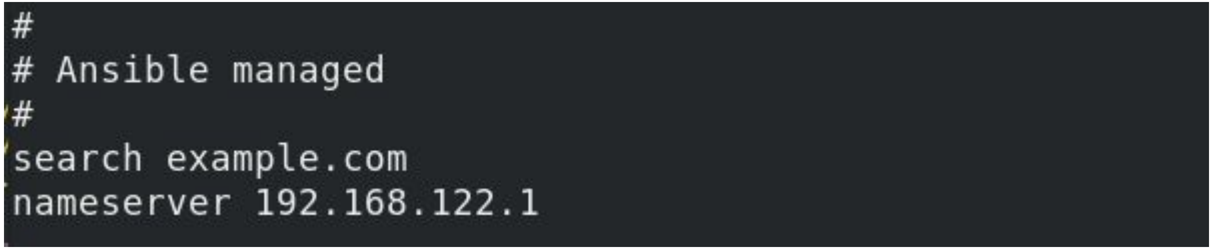
```
PLAY [install and start apache]
********************************

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

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

TASK [start apache service]
********************************
ok: [web1]
```

#10: Templates

Careful when mixing manual and automated configuration (Or even different automation frameworks…)

- Label template output files as being generated by Ansible

```
{{ ansible_managed | comment }}
```



```
#
# Ansible managed
#
search example.com
nameserver 192.168.122.1
```

#11: Roles

Keep in mind

- 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

#11: Roles

Tricks and tips

- 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

Thank you

Complexity kills productivity
Optimize for readability
Think declaratively