Automation Good Practices written by you
An open source project of the Red Hat Automation CoP
Brant, Moritz, Eric, Vinny
Eric Lavarde
Principal Architect
EMEA Automation Practice
Automation CoP Manager

Moritz Schönwetter
Architect
Consulting Team Germany
AutoCoP manager, too
Agenda
Automation Good Practices

Introduction
Examples
Call to action
Introduction

What is AGP?
Automation Good Practices (AGP) is a public Git repository of good practices for automation (mostly Ansible related, of course). Everybody can read and participate there.

Each good practice is made of:

- Title
- Explanation
- Rationale
- Example
History

Automation Good Practices (AGP)

First commit
February 2021, created by the Automation CoP

Based on
https://github.com/oasis-roles/meta_standards

Purpose
Provide consistent (and good) code and common structures for consultants, product teams, customers and partners

⇒ AGP is basis for linting of “validated content”

Ultimate Goal
Make a community book out of it

Source:
https://github.com/redhat-cop/automation-good-practices/commit/0abad47b5ff7a10e1fba171254de3c3cc5b81cda9f
https://openclipart.org/detail/259278/herodotus
https://ansible-lint.readthedocs.io/profiles/#production
Introduction

Maintenance process

https://github.com/redhat-cop/automation-good-practices/

Anybody can participate
Read the contribution guidelines, fork the repo, offer a pull request

Review process
Community based, offline review

Approval
during one of the bi-weekly Red Hat-internal Automation CoP meetings
(each 2nd Wednesday)

Merge & Build
merged by one of the repo maintainers and automatically published

Source:
https://red.hat/autocop-chat
https://red.hat/autocop-meet
We have good practices for...

- Structures
- Roles
- Collections
- Playbooks

- Inventories
- Variables
- Plugins
- Coding in general
Few examples of good practices

The recommendations we love the most (just like, our opinion, man)
Defining which structure to use for which purpose

- Define which use case to use roles, playbooks, potentially workflows, and how to split the code you write.

Because a house made of bricks of various sizes is difficult to build.

<table>
<thead>
<tr>
<th>Landscape</th>
<th>Type</th>
<th>Function</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. the business app landscape</td>
<td>e.g. 1 internet facing web server</td>
<td>e.g. a web server function</td>
<td>e.g. nginx server</td>
</tr>
<tr>
<td>= 1 controller</td>
<td>= 1 Playbook resp. Job Template</td>
<td>= 1 Ansible role</td>
<td>= 1 task-file in a function role or 1 dependent role</td>
</tr>
<tr>
<td>Workloads or Playbook of Playbooks</td>
<td>Each server has 1 and only 1 type</td>
<td>A matter of re-usability</td>
<td>A matter of readability</td>
</tr>
</tbody>
</table>

- Each server has 1 and only 1 type.
- A matter of re-usability.
- A matter of readability.
Avoid names like `package` in favor of a name like `foo_package`.

Prefix internal variables with two underscores: `__foo_variable`.

Use `snake_case_naming` for variable names.
Identify your Single Source(s) of Truth and use it/them in your inventory

- Git repos
- Satellite
- Cloud inventory
- Network Infrastructure Service
Using Single Source(s) of Truth and your inventory

Some general rules (exceptions do exist)

- Filter at the source (i.e. in the API-request), not in the inventory
- Use caching
- Treat the inventory in Controller as disposable (i.e. don’t define variables there)
Examples

- Define your inventory as a structured directory instead of a single file.
- Easier to maintain and grow at scale.
- Cleanly separate lists of hosts & groups from the variable definitions at the group and host level, also including dynamic plugins or even scripts.
- Group and host variable files named like the role they concern (capital_letter, etc...).

Examples:

```
[all]
host1.example.com
host2.example.com
host3.example.com

[alphas]
host1.example.com

[betas]
host2.example.com

[greek_letters:children]
alphas
betas

[hebrew_letters:children]
alephs

(YAML is also fine here)
```
Split long expressions into multiple lines

>, +, -, and | are your friends

```
- name: set a very long variable
  set_fact:
    meaningless_variable: >-
      Ut ac neque sit amet turpis ullamcorper auctor.
      Cras placerat dolor non ipsum posuere malesuada at ac ipsum.
      Duis a neque fermentum nulla imperdiet blandit.
```

...your weird friends

```
YAML
example: |+7

----- Several lines of text,
----- with some "quotes" of various 'types',
----- and also a blank line:
-----
----- and some text within
----- extra indentation
----- on the next line,
----- plus another line at the end.
----
-----

Source:
https://yaml-multiline.info
https://redhat-cop.github.io/automation-good-practices/#_yaml_and_jinja2_syntax
And don’t forget the Zen of Python Ansible

By Guru Tim Appnel
Call to action

There is a reason why we're here!
What can you do?

A lot...

Read

It should be the beginning, shouldn’t it?

https://redhat-cop.github.io/automation-good-practices/

Apply & Share

With colleagues, customers and partners; the more people use and know about it, the better

Improve

Ask questions, create issues, address existing issues (we’ve got plenty of them), improve language, clarify, add new practices, create Pull Requests!

Source:
https://redhat-cop.github.io/automation-good-practices/
https://github.com/redhat-cop/automation-good-practices/issues
https://github.com/redhat-cop/automation-good-practices/pulls
Thank you

Red Hat is the world’s leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.