The Linux & Open Source Company

Building CI/CD Pipelines for your Ansible code

ATIX AG

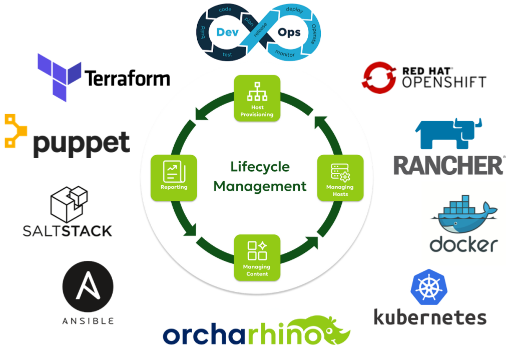
Hello 👋

- Ottavia Balducci
- IT consultant at ATIX AG

My focus:

- Ansible
- AWX/AAP
- orcharhino/Foreman

ATIX AG



Open-Source & Automation:

- Consulting
- Engineering
- Support
- Trainings

Agenda

Ansible

- Gentle learning curve
- Many YAML files
- Maybe some Python code
- Multiple locations possible to store information

git

- Version control
- Collaboration
- CLI, GitLab, GitTea, GitHub, Bitbucket, …

CI/CD pipelines

- Execute jobs on git events
- Usually on a runner
- For tests, linting, releases, deployments, …

CI/CD pipelines with GitLab

.gitlab-ci.yml

```yaml
---
stages:
  - build
variables:
  REGISTRY: ${CI_REGISTRY}/${CI_PROJECT_PATH}
  VERSION: "3.13.1-alpine3.21"
build_image:
  stage: build
  image: ${CI_REGISTRY}/docker:latest
  script:
    - set -eu
    - printf '%s\n' "${CI_REGISTRY_PASSWORD}" | docker login -u "${CI_REGISTRY_USER}" --password-stdin "${CI_REGISTRY}"
    - docker build --pull --no-cache --build-arg VERSION="${VERSION}" --file Containerfile -t "${REGISTRY}:${VERSION}" .
    - docker push "${REGISTRY}:${VERSION}"
...
```

git hooks

- git hooks are great, but CI/CD tools have
  - much more workflow options
  - nice logging
  - dedicated runners
  - and much more!
- pre-commit hooks can be useful to keep your remote clean!

ansible-lint

- Static Analysis
- Checks Ansible code for best practices and possible caveats
- Enforce standards

```
# install
pip install ansible-lint

# use
ansible-lint <directory or task/playbook file>
```

What about `yamllint`?

- Generic linter for YAML
- Used by default by ansible-lint
- Checks syntax validity, indentation, trailing spaces and much more

ansible-lint configuration

Project configuration (.ansible-lint):

```yaml
---
profile: min
exclude_paths:
  - test.yaml
extra_vars:
  foo: bar
  multiline_string_variable: |
    line1
    line2
  complex_variable: ":{;\t$()"
```

Disable rules on a per line basis:

```yaml
---
- ansible.builtin.debug:
    msg: "This line is a debug message" # noqa unnamed-task
```

Example .gitlab-ci.yml

```
---
stages:
  - lint
ansible:
  stage: lint
  image: ${CI_REGISTRY}/ansible-lint:main
  before_script:
    - ansible --version
    - |
      for EXT in "yml" "yaml"
      do
      [ -f requirements.$EXT ] && ansible-galaxy collection install -r requirements.$EXT
      done
  script:
    - ansible-lint --version
    - ansible-lint -vvvvv
...
```

Ansible Collections

- Modular and reusable
- Easily distributed to different users/hosts
- Clearer context
- Easier version control
- Better maintenance

```
atix/
└── cfgmgmtcamp/
    ├── roles/
    ├── playbooks/
    ├── plugins/
    │   ├── modules/
    │   ├── inventory/
    │   ├── filter/
    │   └── ...
    ├── galaxy.yml
    ├── README.md
    ├── CHANGELOG.rst
    └── ...
```

galaxy.yml

```yaml
---
authors:
  - ...
description: Kubernetes Collection for Ansible.
documentation: ""
homepage: ""
issues: https://github.com/ansible-collections/kubernetes.core/issues
license_file: LICENSE
namespace: kubernetes
name: core
readme: README.md
repository: https://github.com/ansible-collections/kubernetes.core
tags:
  - kubernetes
  - k8s
  - cloud
  - infrastructure
  - openshift
  - okd
  - cluster
version: 5.0.0
build_ignore:
  - .DS_Store
  - "*.tar.gz"
```

ansible-galaxy

- Initialize a collection

```
ansible-galaxy collection init atix.cfgmgmtcamp
```

- Build a collection

```
ansible-galaxy collection build
```

- Publish a collection

```
ansible-galaxy collection publish
```

- Install a collection

```
ansible-galaxy collection install community.general
ansible-galaxy collection install -r requirements.yml
```

Where can collections be stored?

ATIX

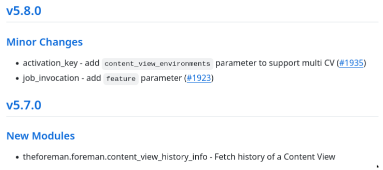
- Directly in git
- Public Galaxy server
- Galaxy NG
- Private Automation Hub

Red Hat
Ansible Automation
Platform

ANSIBLE
GALAXY

CHANGELOG.rst

- Clear project history
- Easier maintenance and support
- Better user trust
- Smoother collaboration



v5.8.0

Minor Changes

- activation_key - add `content_view_environments` parameter to support multi CV (#1935)
- job_invocation - add `feature` parameter (#1923)

v5.7.0

New Modules

- theforeman.foreman.content_view_history_info - Fetch history of a Content View

antsibull-changelog

- Each commit must have a changelog fragment
- Generate `CHANGELOG.rst` from fragments
  - `antsibull-changelog release`
- Automatic recognition of plugins/modules

```
├── CHANGELOG.rst
├── changelogs
│   ├── config.yaml
│   └── fragments
│       ├── 123-some-module.yaml
│       └── 124-doc.yaml
├── galaxy.yml
├── playbooks
│   └── ...
├── plugins
│   └── ...
└── ...
```

Fragments syntax

```yaml
deprecated_features:
  - >-
    docker_container - the ``trust_image_content`` option will be removed.
    It has always been ignored by the module.
  - >-
    docker_stack - the return values ``err`` and ``out`` have been deprecated.
    Use ``stdout`` and ``stderr`` from now on instead.

breaking_changes:
  - >-
    "docker_container - no longer passes information on non-anonymous volumes
    or binds as ``Volumes`` to the Docker daemon. This increases compatibility
    with the ``docker`` CLI program. Note that if you specify ``volumes: strict``
    in ``comparisons``, this could cause existing containers created with
    docker_container from Ansible 2.9 or earlier to restart."
```

Validate syntax with: `antsibull-changelog lint`

CHANGELOG check in CI/CD

```yaml
---
stages:
  - lint

antsibull-changelog-lint:
  stage: lint
  image: ${CI_REGISTRY}/ansible:latest
  before_script:
    - python3 -m pip install antsibull-changelog
    - |
      [ -f "requirements.yml" ] \
      && ansible-galaxy install -r requirements.yml
  script:
    - antsibull-changelog lint
```

Execution Environments

- Container images for executing Ansible
- Created with `ansible-builder`
- Customizable with:
  - own Ansible collections
  - own Python packages
  - own certificates
  - ...
- use with `ansible-navigator`, AWX, AAP

execution-environment.yml

```yaml
---
version: 3

dependencies:
  galaxy: requirements.yml

images:
  base_image:
    name: quay.io/ansible/awx-ee:24.6.1

additional_build_steps:
  prepend_final: |
    RUN whoami
    RUN cat /etc/os-release
  append_final:
    - RUN echo This is a post-install command!
    - RUN ls -la /etc
```

ansible-builder

```
ansible-builder build --container-runtime docker -t <registry>/<image-name>:<tag>
```

molecule

- Create test infrastructure
- Run Ansible code against it
- Verify results
- Test idempotence
- Destroy test infrastructure

`molecule` - Usage

- Initialize scenario:
  `molecule init scenario my_scenario`
- Create infrastructure:
  `molecule create`
- Run the test: `molecule converge`
- Ensure idempotency:
  `molecule idempotence`
- Verify results: `molecule verify`
- Destroy infrastructure:
  `molecule destroy`

```
molecule/
└── my_scenario/
    ├── converge.yml
    ├── create.yml
    ├── destroy.yml
    ├── molecule.yml
    └── verify.yml
```

CI/CD example

```yaml
---
stages:
  - test

molecule_test:
  stage: test
  image: ${CI_REGISTRY}/docker:latest
  script:
    - apk update
    - apk upgrade
    - >-
      apk add --no-cache python3 python3-dev
      py3-pip gcc git curl build-base
      autoconf automake py3-cryptography linux-headers
      musl-dev libffi-dev openssl-dev openssh
    - python3 -m venv .molecule
    - source .molecule/bin/activate
    - python3 -m pip install ansible  molecule-plugins[docker]
    - molecule test
```

Thank you for your attention!

Q&A