devoteam

Ansible Entwickler Plattform

Plus Powershell Unit-Tests

Creative tech for Better Change

Viktor Dück

Komponenten

Basierend auf CI/CD Setup in Azure DevOps

Tooling Container Image

- Wird automatisch gebaut und in interne Container-Registry veröffentlicht
- Gleiche Ansible Version wie in AAP/AWX
- Powershell Core + Pester
- Enthält alle nötigen Skripte für Tests und Publish von Collections
- Wird von Pipelines verwendet
- Kann von Entwicklern verwendet werden um lokal zu testen

Terraform

- Custom Module für Ansible Collections
- Erstellt Collection Git Repo anhand eines Template Repo
- Definiert Branch Policies und Collection Ownership (Required Reviewers)
- Aktualisiert Pipeline Definitionen um neue Collection automatisch zu testen
- Neue Collections brauchen nur einen Eintrag in Liste

Collection Repo

- Enthält Ansible Collection Boilerplate Ordnerstruktur
- main-Branch nur durch PR beschreibbar und damit (theoretisch) immer releasefähig
- PR muss Build Validation erfolgreich durchlaufen, bevor Merge möglich wird
- Collection Owner müssen Approval geben, bevor Merge möglich wird

Build Validation

- Wird durchlaufen, sobald PR auf main-Branch geöffnet wird
- Test-Suite:
  a. Sanity Tests
  b. Unit Tests
  c. Integration Tests
  d. Pester Tests
- Kommentiert PR mit Testergebnissen

Testergebnisse

Ansible Collection Build Service Nov 10, 2023

sanity_test_ansible-doc

✅ Pass

▼ Show Results

```
Running sanity test "ansible-doc"
```

Ansible Collection Build Service Nov 10, 2023

integration_tests

⚠ Warning

▼ Show Results

```
No integration tests defined. Skipping
```

Ansible Collection Build Service • Aug 23, 2023

sanity_test_pslint

❌ Fail

▼ Show Results

```
Running sanity test "pslint"
ERROR: Found 2 pslint issue(s) which need to be resolved:
ERROR: plugins/modules/MyModule.ps1:56:1: PSUseConsistentIndentation: Indentation not consistent
ERROR: plugins/modules/MyModule.ps1:65:20: PSShouldProcess: 'Invoke-MyAction' has the ShouldProcess attribute but does not call ShouldProcess/ShouldConti
See documentation for help: https://docs.ansible.com/ansible-core/2.14/dev_guide/testing/sanity/pslint.html
FATAL: The 1 sanity test(s) listed below (out of 1) failed. See error output above for details.
pslint
```

Release Validation

- Läuft zusätzlich falls PR **[RELEASE]** im Titel enthält
- Prüft:
  a. Collection Build ist erfolgreich
  b. Artefakt kann in PAH/Galaxy importiert werden (galaxy_importer)
  c. Collection Version ist noch nicht veröffentlicht (Hack)

Veröffentlichte Versionen prüfen

```python
from contextlib import contextmanager
import sys
import os
from ansible.cli.galaxy import GalaxyCLI


@contextmanager
def suppress_output():
    with open(os.devnull, 'w') as devnull:
        old_stdout = sys.stdout
        old_stderr = sys.stderr
        sys.stdout = devnull
        sys.stderr = devnull
        try:
            yield
        finally:
            sys.stdout = old_stdout
            sys.stderr = old_stderr

```

```python
galaxy_args = [
    'ansible-galaxy', 'collection', 'list', 'invalid.invalid',
    '-s', os.environ['API_URL'],
    '--token', os.environ['API_TOKEN'],
    '-p', '.']

cli = GalaxyCLI(galaxy_args)
with suppress_output():
    cli.run()

versions = cli.api.get_collection_versions(namespace=os.environ['COLLECTION_NAMESPACE'],
                                           name=os.environ['COLLECTION_NAME'])

print(",".join(versions))
```

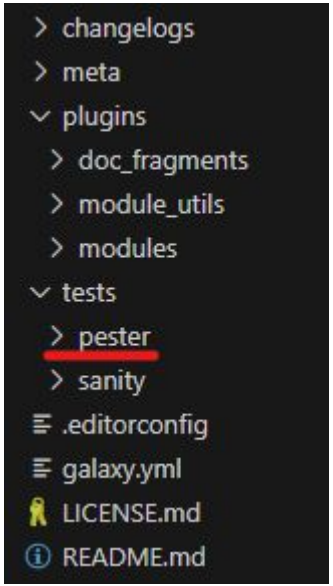
Publish

- Läuft, sobald Merge mit **[RELEASE]** im Titel auf main-Branch auftaucht
- Durchläuft nochmal Release Validation
- Veröffentlicht Collection Artefakt nach PAH/Galaxy

Windows Module und Unit-Tests

Ansible + Pester

Ansible Windows Module testbar machen



Problem:

- Ansible unterstützt nativ keine Unit-Tests für Windows Module
- Folgt man der Dokumentation, sind Windows Module auch nicht mit Pester testbar
- Pester lädt zu testenden Code via dot-sourcing und würde Windows Module (`*.ps1`) damit einfach ausführen

Lösung:

- Ansible Dokumentation (teilweise) ignorieren
- Windows Modul-Code nicht als einfaches Skript schreiben, sondern in eine (main-)Funktion verlagern
- Pester Unit-Tests in eigenen Ordner verlagern
- main-Funktion nur ausführen, falls die Datei nicht von Pester aufgerufen wird

Neue Windows Modulstruktur

```
5
6  > $spec = @{ ···
19   }
20
     12 references
21 > function Invoke-AnsibleModule { ···
63   }
64
65   # don't actually run the module for Pester tests
66   if ($MyInvocation.PSScriptRoot -like "*tests$([IO.Path]::DirectorySeparatorChar)pester*") {
67       exit 0
68   }
69
70   Invoke-AnsibleModule
```

Ergebnis

Entwickler schreiben nur noch Code

Vereinfachter Workflow

Von der Idee zur fertigen Collection

Entwickler
Automatisierung

1. **Neue Collection in Terraform definieren**
2. **Collection Repo erstellen**
3. **Collection Repo konfigurieren**
4. **Code in Feature-Branch schreiben**
5. **PR erstellen**
6. **Code testen**
7. **Code integrieren (PR Merge)**
8. **Collection veröffentlichen**

Vorteile

Garantiertes Minimum Qualität

Sanity Tests **müssen** für Merge erfolgreich sein.

Unit- und Integration-Tests **können** vom Entwickler geschrieben werden, müssen dann aber auch erfolgreich laufen.

Weniger “Mental Load”

Automatisierung nimmt fehleranfällige Arbeit ab.

Standardisierte Tests können nicht “vergessen” werden.

Beschleunigte Entwicklung

Direktes Test-Feedback erlaubt es Änderungen sicher durchzuführen.

Interne Kollaboration fördern

Jeder kann PRs öffnen, Owner entscheiden über Approval, Testergebnisse beschleunigen Entscheidungen.

Veröffentlichte Collections allgemein verwendbar.

Und ihr so?

devoteam