developer.skatelescope.org Documentation

Release 0.1.0-beta

Marco Bartolini

Sep 21, 2023

Home

This repository holds the common job templates used by the pipelines in the SKA Projects.

It is developed to provide common functionality across SKAO CI/CD by calling the make targets residing in ska-cicd-makefile repository.

For any help requests, suggestions, improvements or any issues you may have with the content of this repository please drop us a message on #team-system-support channel at SKAO slack.

CHAPTER 1

Disclaimer

The latest releases of ansible (v7.1.0 and v7.0.0) introduced a bug into ansible-core (v2.14.1) that breaks ansible-lint. In order to fix this the ansible version being installed is going to be pinned to version 6.7.0 and should go back to the latest version once the bug has been patched. Ansible-lint version has also been temporarily pinned to version 6.11.0

CHAPTER 2

Templates

Currently the following artefact types are supported:

- **ansible:** ansible collections
- **helm:** Helm Charts
- **oci-images:** OCI Images
- **python:** Python Code and Artefacts
- **raw:** Raw Artefacts
- **conan:** Conan Artefacts
- **rpm:** RPM Artefacts
- **gpu:** Python Code and Artefacts with GPU support for the testing stage
- **terraform:** Terraform Code

In addition, several use-cases for a typical repository are also supported:

- **docs:** Sphinx documentation support for ReadTheDocs
- **k8s:** Kubernetes in-cluster testing
- **release:** GitLab tag based release and changelog automation
- **finaliser:** GitLab CI Badges and Pipeline workflow rules
- **tmdata:** Upload telescope model data
- **xray:** upload BDD test results to XRAY

2.1 Template Inheritance

There is a hierarchy to the templates which express a certain amount of inheritance, through nested includes:

```
graph LR

  NodeRoot[Template Hierarchy] --> NodeFinaliser[finaliser.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeRelease[release.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeDocs[docs.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeRaw[raw.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeRPM[rpm.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeConan[conan.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeAnsible[ansible.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeHelm[helm-chart.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeK8s[k8s.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeK8sTestRunner[k8s-test-runner.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeDeploy[deploy.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeOCI[oci-image.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodePython[python.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeTerraform[terraform.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeTMData[tmdata.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeXray[xray-publish.gitlab-ci.yml]
  NodeRoot[Template Hierarchy] --> NodeConfigCapture[config-capture.gitlab-ci.yml]

  subgraph "Finaliser/Badge Templates"
  NodeFinaliser --> NodeFinaliserRules[rules.gitlab-ci.yml]
  end

  subgraph "Release Management Templates"
  NodeRelease --> NodeReleaseChangeLogBuild[changelog-build.gitlab-ci.yml]
  end

  subgraph "Docs Templates"
  NodeDocs --> NodeDocsBuild[docs-build.gitlab-ci.yml]
  NodeDocs --> NodeDocsPages[docs-pages.gitlab-ci.yml]
  end

  subgraph "Raw Package Templates"
  NodeRaw --> NodeRawBuild[raw-build.gitlab-ci.yml]
  NodeRaw --> NodeRawPublish[raw-publish.gitlab-ci.yml]
  end

  subgraph "RPM Package Templates"
  NodeRPM --> NodeRPMBuild[rpm-build.gitlab-ci.yml]
  NodeRPM --> NodeRPMPublish[rpm-publish.gitlab-ci.yml]
  end

  subgraph "Conan Package Templates"
  NodeConan --> NodeConanBuild[conan-build.gitlab-ci.yml]
  NodeConan --> NodeConanPublish[conan-publish.gitlab-ci.yml]
  end

  subgraph "Ansible Templates"
  NodeAnsible --> NodeAnsibleLint[ansible-lint.gitlab-ci.yml]
  NodeAnsible --> NodeAnsiblePublish[ansible-publish.gitlab-ci.yml]
  end

  subgraph "Helm Templates"
  NodeHelm --> NodeHelmLint[helm-chart-lint.gitlab-ci.yml]
  NodeHelm --> NodeHelmBuild[helm-chart-build.gitlab-ci.yml]
  NodeHelm --> NodeHelmPublish[helm-chart-publish.gitlab-ci.yml]
```

(continues on next page)

(continued from previous page)

```
end

subgraph "Kubernetes Templates"
NodeK8s --> NodeK8sBuild[k8s-test.gitlab-ci.yml]
NodeK8sTestRunner
end

subgraph "Xray Templates"
NodeXray
end

subgraph "Config Capture Templates"
NodeConfigCapture
end

subgraph "Deployment Templates"
NodeDeploy --> NodeDeployDev[deploy.dev.gitlab-ci.yml]
NodeDeploy --> NodeDeploySharedDev[deploy.shared.dev.gitlab-ci.yml]
NodeDeploy --> NodeDeployIntegration[deploy.integration.gitlab-ci.yml]
NodeDeploy --> NodeDeployStaging[deploy.staging.gitlab-ci.yml]
end

subgraph "OCI Image Templates"
NodeOCI --> NodeOCILint[oci-image-lint.gitlab-ci.yml]
NodeOCI --> NodeOCIBuild[oci-image-build.gitlab-ci.yml]
NodeOCI --> NodeOCIPublish[oci-image-publish.gitlab-ci.yml]
NodeOCI --> NodeOCIScan[oci-image-scan.gitlab-ci.yml]
end

subgraph "Python Templates"
NodePython --> NodePythonLint[python-lint.gitlab-ci.yml]
NodePython --> NodePythonBuild[python-build.gitlab-ci.yml]
NodePython --> NodePythonTest[python-test.gitlab-ci.yml]
NodePython --> NodePythonPublish[python-publish.gitlab-ci.yml]
end

subgraph "Terraform Templates"
NodeTerraform --> NodeTerraformLint[terraform-lint.gitlab-ci.yml]
end

subgraph "Telescope Model Data Templates"
NodeTMData --> NodeTMDataBuild[tmdata-package.gitlab-ci.yml]
NodeTMData --> NodeTMDataPublish[tmdata-publish.gitlab-ci.yml]
end
```

To learn more about how to use the templates, follow *How to Use* section.

To learn more about how the templates are designed and integrated, follow *Design* section.

2.2 Pipeline Rules

The `finaliser.gitlab-ci.yml` file includes top-level workflow rules(`rules.gitlab-ci.yml`) that enables the Merge Request Pipelines and prevents duplication with the Branch Pipelines. Please see below where the different types of pipelines are described. It also enables the use of Merged Result pipelines.

Branch Pipelines: pipelines that target a source branch, i.e. the pipeline is run against what you pushed to the remote

every time.

Merge Request Pipelines: pipelines that are attached to a merge request. They are different because they only run against commits associated with a merge requests

`rules.gitlab-ci.yml` file(`workflow:rules`) defines whether the pipeline should be created or not at all. This is to manage branch and merge request pipelines while preventing duplicate pipelines. For example, without any rules if there is an open MR, there would be two identical pipelines running, one for the MR pipeline and the other for the branch pipeline. The added `workflow:rules` prevents this by enabling/disabling one of the pipeline types. The current workflow is if there are no open merge requests, a branch pipeline is created. If there is an open merge request, then the branch pipelines are disabled and merge request pipelines are enabled. So, there would only be single pipeline in the end.

The above is needed for the merged result pipelines. It enables the pipeline to really see the results of the master branch pipeline as if the changes are merged instead of just seeing the results for the source branch.

To illustrate, It can easily happen that a new commit in a branch works (i.e. the pipeline status is green, and as a result the Merge Request seems safe to merge), while in fact the merge breaks the pipeline in the master branch. By using Merge Result pipelines we can potentially avoid this, by running pipelines on what the master branch would like if our changes were merged. This setting is enabled for all projects in SKAO group. To learn more, please have a look at the GitLab blog post about it.

There are some conditions:

- The MR has to be ready, i.e. no Draft or WIP tags in the MR title
- There shouldn't be any merge conflicts If the MR is not mergeable (above conditions), then what we have is the old pipelines that are targeting the source branch as usual. However, these are now labelled as detached by GitLab (I think poor choice of words) to differentiate them from merged result pipelines. Fear not! They are still targeting your branch so you are not actually in a detached state from your branch.

CHAPTER 3

How to Use

First, add the ska-cicd-makefile repository as a submodule to your project.

```
git submodule add https://gitlab.com/ska-telescope/sdi/ska-cicd-makefile.git .make
```

Make sure that you add and commit the `.gitmodules` file and `.make` directory:

```
git add .gitmodules .make
git commit -s
```

Now include the `*.mk` files in your `Makefile`:

```
include .make/*.mk
```

Or, you can include specific ones like below. Note that you have to include the ones you are going to use in your pipeline.

```
# include makefile targets for Kubernetes management
-include .make/k8s.mk

## The following should be standard includes
# include core makefile targets for release management
-include .make/base.mk

# include your own private variables for custom deployment configuration
-include PrivateRules.mak
```

To ensure that your GitLab CI pipeline automatically clones submodules, add the following to `.gitlab-ci.yml`:

```
variables:
  GIT_SUBMODULE_STRATEGY: recursive
```

In most scenarios, include the appropriate main template files in your repositories `gitlab-ci.yml` file. To include python, oci, kubernetes and documentation support:

```
include:
    # Python
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/python.gitlab-ci.yml'
    # Terraform
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/terraform.gitlab-ci.yml'
    # Docs pages
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/docs-pages.gitlab-ci.yml'
    # k8s steps
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/k8s.gitlab-ci.yml'
    # configuration capture
   - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/config-capture.gitlab-ci.yml'
    # OCI Images
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/oci-image.gitlab-ci.yml'
    # Raw
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/raw.gitlab-ci.yml'
    # RPM
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/rpm.gitlab-ci.yml'
    # Conan
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/conan.gitlab-ci.yml'
    # Tag Based GitLab Release Management
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/release.gitlab-ci.yml'
    # .post step finalisers eg: badges
  - project: 'ska-telescope/templates-repository'
    file: 'gitlab-ci/includes/finaliser.gitlab-ci.yml'
```

Note, `finaliser.gitlab-ci.yml` is required by every repository as it includes functionality for ci-badges and workflow rules which is explained in *Templates* section.

If you don't require every job for an artefact in your project, you could also just include the necessary part:

```
include:
  # We don't need python build and publish
  # as we are not releasing a python artefact
  - project: 'skatelescope/templates-repository'
    file: 'gitlab-ci/includes/python-lint.gitlab-ci.yml'
  - project: 'skatelescope/templates-repository'
    file: 'gitlab-ci/includes/python-test.gitlab-ci.yml'
```

If you want to get detailed help on what each job is going to do, run `make long-help` to get additional documentation on the actual make targets.

3.1 Customisation

All of the jobs should be configured by their respective make-targets and variables and hooks provided. Follow ska-cicd-makefile to learn more.

Example; If you want to include additional `pytest` flags, you can define `PYTHON_VARS_AFTER_PYTEST` variable in your main Makefile as below:

```
PYTHON_VARS_AFTER_PYTEST = -m 'not post_deployment' --forked \
      --disable-pytest-warnings
```

Note that although you can achieve the same effect for the pipelines by providing variables in your `gitlab-ci.yml` file as below, **it is not recommended!** as it makes the local development and pipeline behaviour diverge from each other.

```
# NOT RECOMMENDED!
variables:
 - PYTHON_VARS_AFTER_PYTEST: "-m 'not post_deployment' --forked \
      --disable-pytest-warnings"
```

CHAPTER 4

Design

For every artefact type in SKAO, there should be a template that addresses some of the common steps in DevSecOps Lifecycle:

- Lint
- Build
- Test
- Package
- Release
- Deploy

Each artefact type has its own template file in `<artefact-type>.gitlab-ci.yml` which includes avaiable steps. Each step have its own `<artefact-type>-<step>.gitlab-ci.yml` file to describe when and how it should run.

For python;

```
python.gitlab-ci.yml
├─ python-lint.gitlab-ci.yml
├─ python-build.gitlab-ci.yml
├─ python-test.gitlab-ci.yml
├─ python-publish.gitlab-ci.yml
```

You can check the contents of the `python.gitlab-ci.yml` file and investigate the actual step template files under gitlab-ci/includes directory.

For each job, `script` part should just compose of calling the make targets associated with the artefact and step. `before-script` part provides checking if the target exists and provides documentation.

Providing that the underlying implementation is the same as makefile targets allows repositories and developers to be flexible in their own implementation and aligns both the development and pipeline environment. *You can read more on how to customise the make targets in makefile repository documentation*

`rules` part describes when the job should be included in the pipeline to eliminate unnecessary/invalid jobs in the pipeline. To push artefacts to GitLab or CAR from the publish stage jobs, the git tags are used as identifier.

4.1 General Layout

For any artefact type:

- the main `<artefact>.gitlab-ci.yml` file has **only** the consisting include jobs.
- Each lifecycle step in a pipeline is named with the correct prefix and should reside in its own `<artefact-type>-<step>.gitlab-ci.yml` file.
- Each step template file includes the necessary jobs starting with the step name `<artefact-type>-<step>...`. Multiple jobs are provided to handle different use-cases. *i.e. to handle python modern/legacy build system, to manage development and production publishing of artefacts etc.*
- Each job performs any preliminary actions in `before_script` such as logins, config settings for the environment.
- Each job just calls the respective `make targets` and passes any variables that needs to be used in `script` part. The script itself should not have any logic to not deviate from the local development workflow.

If we follow the python example as described above. The main `python.gitlab-ci.yml` file only includes other jobs:

```
# unbrella include for all Python life cycle stages
include:
  # Linting stage
  - local: gitlab-ci/includes/python-lint.gitlab-ci.yml
  # Build stage
  - local: gitlab-ci/includes/python-build.gitlab-ci.yml
  # Test stage
  - local: gitlab-ci/includes/python-test.gitlab-ci.yml
  # Publish stage
  - local: gitlab-ci/includes/python-publish.gitlab-ci.yml
```

and the `python-publish.gitlab-ci.yml` describes the actual python build stage jobs. As you can see the necessary variables are passed to the `make python-publish` target and `rules` is used to separate tag pipelines from development pipelines to publish to central artefact repository or gitlab registry, respectively.

```
# Python publish stage template
python-publish-to-car:
  stage: publish
  tags:
    - k8srunner
  image: $SKA_K8S_TOOLS_BUILD_DEPLOY
  before_script:
    - '[ -f .make/python.mk ] || (echo "File python.mk not included in Makefile; exit␣
↪1")'
    - 'make help | grep python-publish'
  script:
    - make PYTHON_PUBLISH_USERNAME=${CAR_PYPI_USERNAME} PYTHON_PUBLISH_PASSWORD=${CAR_
↪PYPI_PASSWORD} PYTHON_PUBLISH_URL=${CAR_PYPI_REPOSITORY_URL} python-publish
  rules:
    - if: '$CI_COMMIT_TAG'
      exists:
        - pyproject.toml
```

(continues on next page)

(continued from previous page)

```
        - setup.py

python-publish-to-gitlab:
  stage: publish
  tags:
    - k8srunner
  image: $SKA_K8S_TOOLS_BUILD_DEPLOY
  allow_failure: true
  before_script:
    - '[ -f .make/python.mk ] || (echo "File python.mk not included in Makefile; exit␣
↪1")'
    - 'make help | grep python-publish'
  script:
    - make PYTHON_PUBLISH_USERNAME=gitlab-ci-token PYTHON_PUBLISH_PASSWORD=${CI_JOB_
↪TOKEN} PYTHON_PUBLISH_URL=https://gitlab.com/api/v4/projects/${CI_PROJECT_ID}/
↪packages/pypi python-publish
  rules:
    - exists:
        - pyproject.toml
        - setup.py
```