Options avancées des pipelines

F. Lassabe

IUT Nord-Franche-Comté

Introduction

- CI/CD dans des projets complexes
  - Sous-projets
  - Dépendances
  - Production de fichiers
  - etc.
- Syntaxe YAML de gitlab CI/CD
- La plupart dans les jobs

allow_failure

- Autorise un job à être validé même s'il échoue
- N'interrompt pas le pipeline
- Uniquement sur un job non essentiel
- En fonction de vos critères de qualité
  - Documentation incomplète
  - Mise en forme incorrecte
  - Certaines suites de tests (non bloquantes)
- Variante `exit_codes` : uniquement quand un script échoue avec un code autorisé

artifacts

- Produit des fichiers utilisables hors du job
- Plusieurs mot-clés
  - path : chemins à partir de la racine du job
  - exclude : exclut des fichiers des chemins
  - `expire_in` : date limite de conservation
  - access : spécifie le droit d'accès (all, developer, maintainer, none)
  - when : spécifie quand générer les artéfacts (always, on_success, on_failure)

dependencies

- Lister des jobs dont on récupère les artéfacts
- Par exemple, si on compile pour Linux, Windows et MacOS, les tests dépendent aussi de l'environnement

```
build osx:
  stage: build
  script: make build:osx
  artifacts:
    paths:
      - binaries/

build linux:
  stage: build
  script: make build:linux
  artifacts:
    paths:
      - binaries/

test osx:
  stage: test
  script: make test:osx
  dependencies:
    - build osx

test linux:
  stage: test
  script: make test:linux
  dependencies:
    - build linux
```

extends

- Réutilise des sections de configuration
- Exemple :

```
.tests:
  stage: test
  image: ruby:3.0

rspec:
  extends: .tests
  script: rake rspec

rubocop:
  extends: .tests
  script: bundle exec rubocop
```

inherit

- Permet d'hériter les propriétés d'autres sections
- Totalement ou partiellement
- Valeur par défaut true (tout)

```
default:
  retry: 2
  image: ruby:3.0
  interruptible: true

job1:
  script: echo "This job does not inherit any default keywords."
  inherit:
    default: false

job2:
  script: echo "This job inherits only both listed default keywords. \
                It does not inherit 'interruptible'."
  inherit:
    default:
      - retry
```

inherit:variables

- Fonctionne également sur des variables
- Exemple :

```
variables:
  VARIABLE1: "This is default variable 1"
  VARIABLE2: "This is default variable 2"

job1:
  script: echo "This job does not inherit any default variables."
  inherit:
    variables: false

job2:
  script: echo "This job inherits only the  listed default variable. \
                It does not inherit 'VARIABLE3'."
  inherit:
    variables:
      - VARIABLE1
```

Interruptible

- Permet d'autoriser l'annulation d'un job en cours en cas de nouveau commit
- Dépend de workflow:auto_cancel:on_new_commit

```yaml
workflow:
  auto_cancel:
    on_new_commit: conservative # default

step-1:
  stage: stage1
  script:
    - echo "Can be canceled."
  interruptible: true

step-2:
  stage: stage2
  script:
    - echo "Can not be canceled."

step-3:
  stage: stage3
  script:
    - echo "Because step-2 can not be canceled, \
            this step can never be canceled, even \
            though it's set as interruptible."
  interruptible: true
```

```yaml
workflow:
  auto_cancel:
    on_new_commit: interruptible

step-1:
  stage: stage1
  script:
    - echo "Can be canceled."
  interruptible: true

step-2:
  stage: stage2
  script:
    - echo "Can not be canceled."

step-3:
  stage: stage3
  script:
    - echo "Can be canceled."
  interruptible: true
```

Pages

- Permet la publication de fichiers sur Gitlab Pages
- Par exemple, des contenus HTML dans un dossier public
- Exemple :

```
create-pages:
  stage: deploy
  script:
    - mv *.html public/
    - mv *.css public/
    - mv *.js public/
  pages: true  # specifies that this is a Pages \
         job and publishes the default public directory
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

Parallel

- Permet de paralléliser plusieurs instances d'un job
- Prend un nombre de jobs entre 1 et 200
- Pour chaque job, 2 variables sont définies :
  - `CI_NODE_INDEX`
  - `CI_NODE_TOTAL`
- Permet d'appliquer des paramètres différents
- Il faut assez de runners, sinon les jobs sont mis en attente

Release

- Utilisé pour créer une *release* du projet
- Plusieurs sous-clés pour définir la release :
  - `tag_name`
  - `tag_message` (optionnel)
  - `name` (optionnel)
  - `description`
  - `ref` (optionnel)
  - `milestones` (optionnel)
  - `released_at` (optionnel)
  - `assets:links` (optionnel)
- Il doit exister un script dans le job (même si juste un *echo*)

Retry

- Permet de re-essayer un job qui échoue jusqu'à 2 fois
- 3 propriétés
  - `max` : nombre d'essais supplémentaires
  - `when` : condition pour réessayer (parmi une liste)
  - `exit_codes` : liste de valeurs de sortie qui autorisent des tentatives supplémentaires

```
test_advanced:
  script:
    - echo "Run a script that results in exit code 137."
    - exit 137
  retry:
    max: 2
    when: runner_system_failure
    exit_codes: 137
```

Rules

- Pour inclure/exclure des jobs du pipeline
- La première règle validée est appliquée
- Règles basées sur notamment :
  - if
  - when
  - changes
  - exists
- Note : quand `when: never` dans un `if`, permet d'exclure le job sur condition

Run

- Permet d'exécuter une série d'étapes (`steps`)
- Chaque étape est un script ou une étape définie par ailleurs

```
job:
  run:
    - name: 'hello_steps'
      script: 'echo "hello from step1"'
    - name: 'bye_steps'
      step: gitlab.com/gitlab-org/ci-cd/runner-tools/echo-s
      inputs:
        echo: 'bye steps!'
      env:
        var1: 'value 1'
```

Trigger

- Fait d'un job un job déclencheur pour soit
  - un pipeline multi-projets
  - un pipeline enfant
- Quand le job déclenché définit des `spec:inputs`, on peut lui transmettre :

```yaml
trigger-multi-project:
  - project: 'my-group/my-project'
    inputs:
      website: "My website"

trigger-child-pipeline:
  trigger:
    include: path/to/child-pipeline.gitlab-ci.yml
```

When

- Configurer les conditions de lancement d'un job :

```yaml
stages:
  - build
  - cleanup_build
  - deploy
  - cleanup

build_job:
  stage: build
  script:
    - make build

cleanup_build_job:
  stage: cleanup_build
  script:
    - cleanup build when failed
  when: on_failure
```

```yaml
deploy_job:
  stage: deploy
  script:
    - make deploy
  when: manual
  environment: production

cleanup_job:
  stage: cleanup
  script:
    - cleanup after jobs
  when: always
```