Environnement de travail collaboratif

ESGI - Semaine du 04/11/19

Christophe Villeger
Vincent Monjaret
Lucas Moisan

Gitlab CI

Continuous Integration

- Ensemble de pratiques consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.
- Le principal but de cette pratique est de détecter les problèmes d'intégration au plus tôt lors du développement.
- De plus, elle permet d'automatiser l'exécution des suites de tests et de voir l'évolution du développement du logiciel.

GitLab Runner

- GitLab Runner est le projet open source utilisé pour exécuter vos jobs et renvoyer les résultats à GitLab.
- Il est utilisé avec GitLab CI, le service d'intégration continue open-source inclus avec GitLab qui coordonne les jobs.

Installation

Le manifeste

Fichier de configuration à la racine du projet

Par défaut : **.gitlab-ci.yml**

Si besoin de le customiser :

➢ Settings > CI/CD > General pipelines > Custom CI config path

job

Nombre illimité de job

Ne dois pas faire partie des mots clé reservé

Dans la description d'un job seul **script** est obligatoire

```
job:1
  script: /bin/echo "Hello CI"

job:2
  script: /bin/echo "Hello from another job"
```

script

le coeur du job car c’est ici que vous indiquerez les actions à effectuer.

Il peut appeler un ou plusieurs script(s) de votre projet, voire exécuter une ou plusieurs ligne(s) de commande.

```yaml
job:script:
  script: ./bin/script/my-script.sh

job:scripts:
  script:
    - pwd
    - ./bin/script/my-script-2.sh
```

before-script / after-script

Exécuter des actions avant et après votre script principal.

Utilise pour:

- ➢ Installer des packages avant un script
- ➢ Nettoyer un repertoire après le script

```yaml
before_script:
  - echo 'start jobs'

after_script:
  - echo 'end jobs'

job:no_overwrite:
  script:
    - echo 'script'

job:overwrite:all:
  before_script:
    - echo 'overwrite'
  script:
    - echo 'script'
  after_script:
    - echo 'overwrite'
```

image

Permet de définir une image docker par défaut ou pour un job

Cette image doit être publié sur un registry

```yaml
image: alpine

job:node:
    image: node
    script: yarn install

job:alpine:
    script: echo $PWD
```

stage

Grouper des jobs en étapes

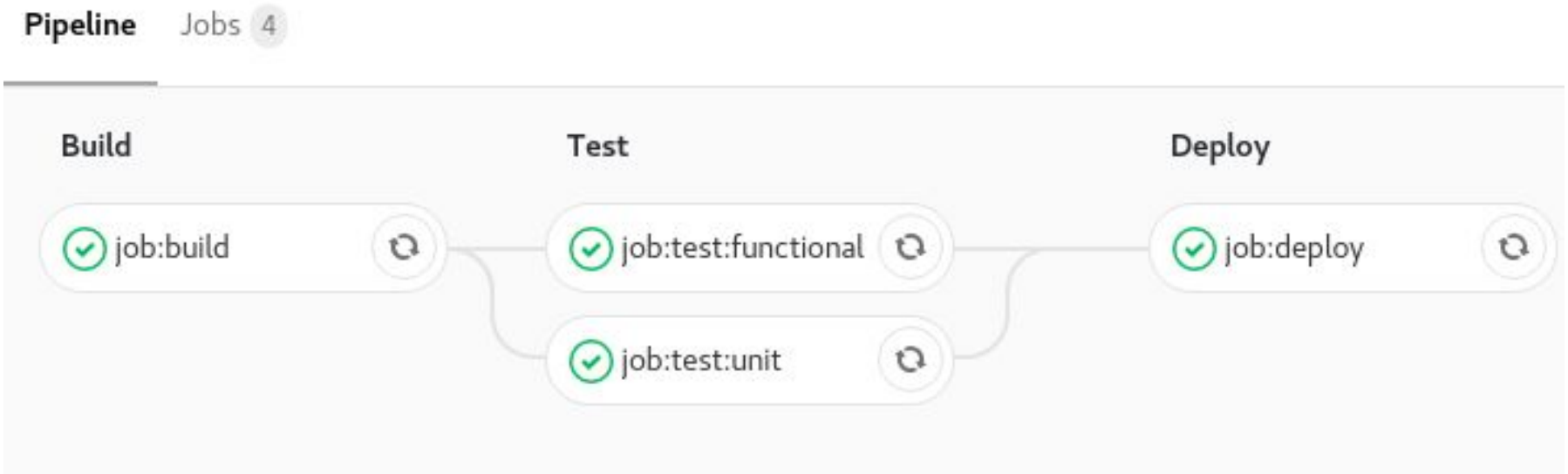
```yaml
stages:
  - build
  - test
  - deploy

job:build:
  stage: build
  script: make build

job:test:unit:
  stage: test
  script: make test-unit

job:test:functional:
  stage: test
  script: make test-functional

job:deploy:
  stage: deploy
  script: make deploy
```


Pipeline
Jobs 4
Build
Test
Deploy
job:build
job:test:functional
job:test:unit
job:deploy

ONLY & EXCEPT

Ces deux directives permettent de mettre en place des contraintes sur l'exécution d'une tâche.

Vous pouvez dire qu'une tâche s'exécutera uniquement sur l'événement d'un push sur master ou s'exécutera sur chaque push d'une branche sauf master.

```
job:only:master:
  script: make deploy
  only:
    - master

job:except:master:
  script: make test
  except:
    - master
```

ONLY & EXCEPT

Voici les possibilités :

- **branches** : déclenche le job quand un un push est effectué sur la branche spécifiée.
- **tags** : déclenche le job quand un tag est créé.
- **api** : déclenche le job quand une deuxième pipeline le demande grâce à API pipeline.
- **pipelines** : déclenche le job grâce à une autre pipeline, utile pour les multiprojets grâce à l'API et le token CI_JOB_TOKEN.
- **external** : déclenche le job grâce à un service de CI/CD autre que GitLab.
- **pushes** : déclenche le job quand un push est effectué par un utilisateur.
- **schedules** : déclenche le job par rapport à une planification à paramétrer dans l'interface web.
- **triggers** : déclenche le job par rapport à un jeton de déclenchement.
- **web** : déclenche le job par rapport au bouton Run pipeline dans l'interface utilisateur.

when

Comme pour les directives only et except, la directive **when** est une contrainte sur l'exécution de la tâche. Il y a quatre modes possibles :

- **on_success** : le job sera exécuté uniquement si tous les jobs du stage précédent sont passés
- **on_failure** : le job sera exécuté uniquement si un job est en échec
- **always** : le job s'exécutera quoi qu'il se passe (même en cas d'échec)
- **manual** : le job s'exécutera uniquement par une action manuelle

allow_failure

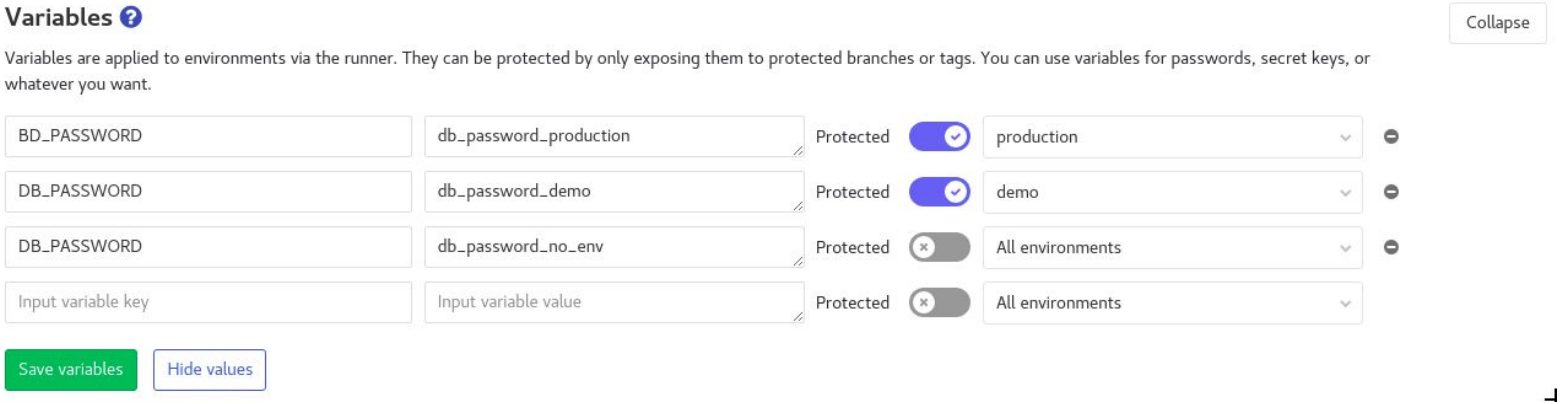
Cette directive permet d’accepter qu’un job échoue sans faire échouer la pipeline

```
job:clean:
  script:
    - make clean
  when: always
  allow_failure: true
```

Variables

Cette déclaration permet de définir des variables pour tous les jobs

Settings > CI/CD > Variables




```
build:
  script: echo ${SYMFONY_ENV}
```

Resources

.gitlab-ci.yml Reference

TP

- Créer votre fichier .gitlab-ci.yml
- Créer un job pour lancer les tests
  - Regrouper dans un stage nommé test
- Créer un nouveau job pour déployer (lié au stage deploy)
  - Uniquement sur la branche master
  - Déclenché uniquement si les test fonctionne bien
  - Installer la gem dpl avant l'exécution du script
  - Utiliser dpl pour déployer sur Heroku (attention aux secrets)