Atelier EPSI-B3DEV: Industrialisation des tests

Infos pratiques

Module: Cours EPSI B3 Dev - 2021/2022
Format: 10H
Enseignant/Formateur: Julien Couraud
Contact: julien.couraud@reseau-cd.net

Rappel Meteor

Documentation et installation

https://docs.meteor.com/install.html#installation

Sources du projet

- Lien vers le tutoriel Meteor React

https://react-tutorial.meteor.com/simple-todos/

- Lien vers les sources du projet à utiliser dans ce module (Step12)

https://github.com/meteor/react-tutorial/tree/master/src/simple-todos/step12

Commandes de base

- Installer les dépendances relatives au projet et définies dans le fichier `package.json` (à exécuter une fois lors de la récupération des sources du projet, puis à chaque ajout d'une librairie externe npm au sein du projet).

```
meteor npm install
```

- Lancer le projet sur le port par défaut (localhost:3000)

```
meteor
```

Commande liée au tests unitaires

- Exécuter les tests unitaires de l'application via la librairie externe `mocha`

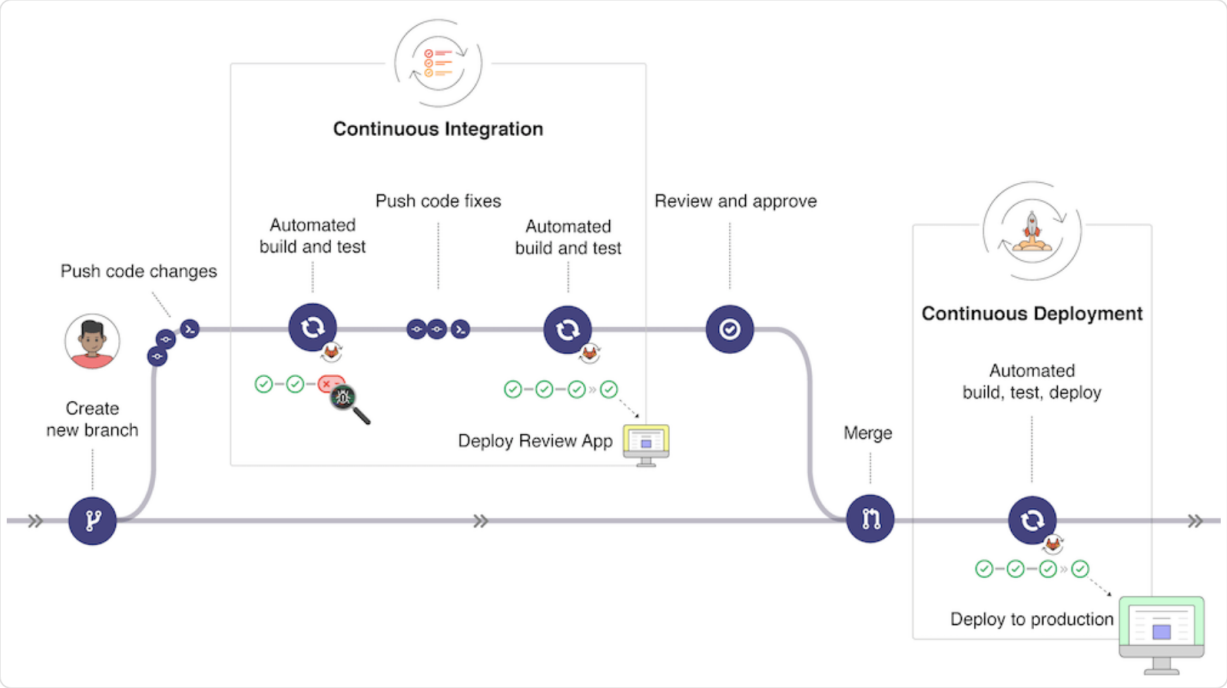
```
meteor test --once --driver-package meteortesting:mocha
```

GitFlow

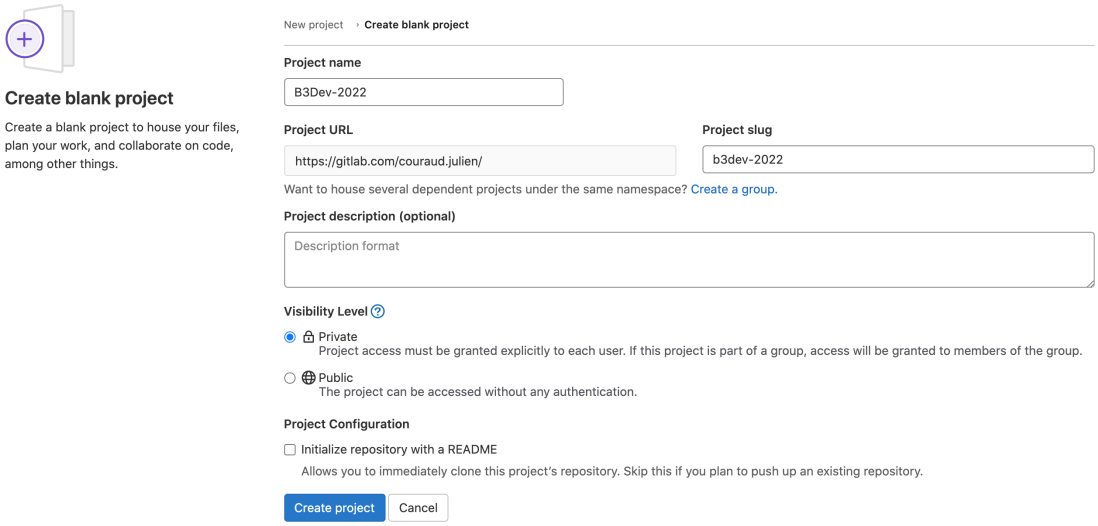
https://www.atlassian.com/fr/git/tutorials/comparing-workflows/gitflow-workflow

Fonctionnalités de GitLab

"Continuous Integration" et "Continuous Deployment"

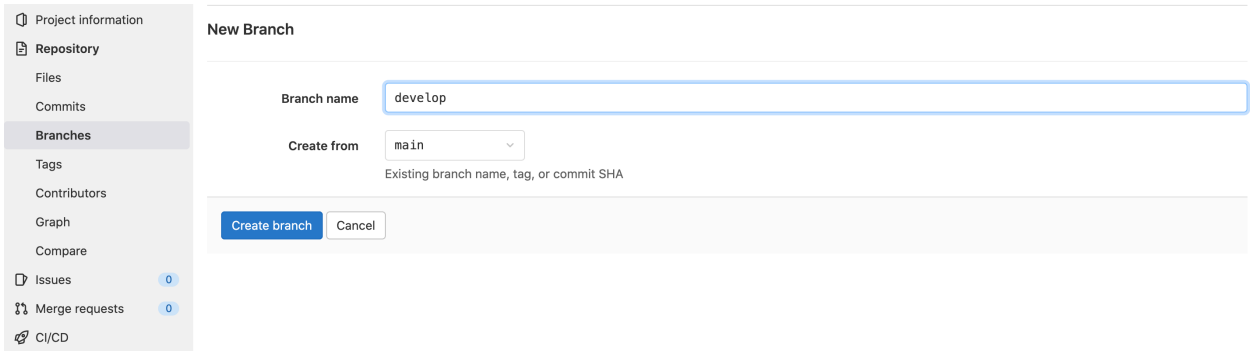


Initialisation du "Repository"



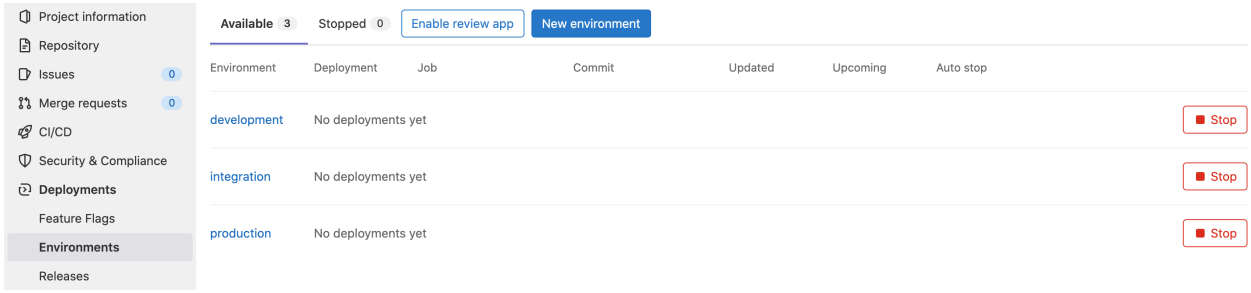
Création des branches

Dans la sidebar GitLab: Repository > Branches



Création des environnements

Dans la sidebar GitLab: Deployments > Environments



Le fichier "gitlab-ci.yml"

```
tests-job:
  stage: test
  script:
    - echo "--------------- METEOR TESTS ---------------"
    - echo "*** Current Repo ***"
    - pwd
    - echo "*** Install Meteor ***"
    - curl https://install.meteor.com/ | sh
    - echo "*** Meteor Commands***"
    - meteor npm install --allow-superuser
    - meteor test --once --driver-package meteortesting:mocha --allow-superuser
```

Multiples "Jobs" au sein d'un même "Stage"

```
before_script:
  - echo "*** Meteor Install ***"
  - curl https://install.meteor.com/ | sh
  - meteor npm install --allow-superuser
  - echo "*** Meteor Install End ***"

unit-tests-job:
  stage: test
  script:
    - echo "*** Unit Tests Starting ***"
    - meteor test --once --driver-package meteortesting:mocha --allow-superuser
    - echo "*** Unit Tests End ***"

system-tests-job:
  stage: test
  script:
    - echo "*** System Tests Starting ***"
    - echo "*** Here some system tests ***"
    - echo "*** System Tests End ***"
```

Utilisation de plusieurs "Stage" pour définir des ordres d'exécution

```
stages:
  - build
  - test
  - deploy

before_script:
  - echo "*** Meteor Install ***"
  - curl https://install.meteor.com/ | sh
  - meteor npm install --allow-superuser
  - echo "*** Meteor Install End ***"

repo-cleaner-job:
  stage: build
  script:
    - echo "*** Cleaning Repository Starting ***"
    - echo "*** Here some repository and file management and cleaning ***"
    - echo "*** Cleaning Repository End ***"

unit-tests-job:
  stage: test
  script:
    - echo "*** Unit Tests Starting ***"
    - meteor test --once --driver-package meteortesting:mocha --allow-superuser
    - echo "*** Unit Tests End ***"

system-tests-job:
  stage: test
  script:
    - echo "*** System Tests Starting ***"
    - echo "*** Here some system tests ***"
    - echo "*** System Tests End ***"

deploy-job:
  stage: deploy
  script:
    - echo "*** Deploy Starting ***"
    - echo "*** Deploy in progress ***"
    - echo "*** Deploy End ***"
```

Exemple Only/Rules/Environments

- Le mot clé `only` permet de lister les branches sur lesquelles un job en particulier sera exécuté. Par exemple, on peut ne pas avoir envie (ou besoin) d'exécuter les tests unitaires lors du tag sur la branche 'main' (synonyme d'environnement de production) car on peut considérer que la version de l'application est stable (sinon on ne merge pas sur 'main' !).

```
unit-tests-job:
  stage: test
  script:
    - echo "*** Unit Tests Starting ***"
    - meteor test --once --driver-package meteortesting:mocha --allow-superuser
    - echo "*** Unit Tests End ***"
  only:
    - /^develop.*/
    - /^release.*/
    - /^hotfix.*/
```

- Le mot clé `rules` permet d'ajouter des règles combinées avec, par exemple, les mots clés `if` et `when`. Dans l'exemple ci-dessous, lorsque le job est exécuté manuellement lors d'une "merge request", alors le job n'interrompra jamais l'execution du pipeline (`allow_failure: true`).

```
job:
  script: echo "Hello, Rules!"
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
      when: manual
      allow_failure: true
```

Voici la liste des mots clés possibles avec `rules` : https://docs.gitlab.com/ee/ci/jobs/job_control.html#common-if-clauses-for-rules

- Le mot clé `environment` permet de classer les jobs par environnements et permettra de retrouver une liste de ces jobs dans la partie Deployments > Environments de GitLab. Cela est particulièrement pratique pour les jobs de déploiement, ainsi on peut avoir une trace directe de la liste de tous les déploiements par environnement. On l'utilisera par exemple notamment avec le mot clé `only` pour avoir les infos relatives à un seul environnements.

```
deploy-development-job:
  stage: deploy
  script:
    - echo "*** Deploy Starting ***"
    - echo "*** Deploy in progress ***"
    - echo "*** Deploy End ***"
  only:
    - /^develop.*/
  environment: development

deploy-production-job:
  stage: deploy
  script:
    - echo "*** Deploy Starting ***"
    - echo "*** Deploy in progress ***"
    - echo "*** Deploy End ***"
  only:
    - main
  environment: production
```

Projet en binôme

Contexte

Vous êtes un(e) architecte technique travaillant dans une société de service en Informatique. Dans le cadre d'un projet web, vous allez être ammené à mettre en place des outils d'automatisation et d'industrialisation des processus de tests et de déploiements.

Les pré-requis techniques du projet sont les suivants:

- Utilisation de la méthodologie GitFlow pour la structure de l'arborescence Git du projet.
- Utilisation de l'outil gratuit et open-source GitLab CI au sein de la solution GitLab.
- Utilisation de 4 environnements distants disponibles et accessible pour des déploiements.

GitFlow

Il y a donc 4 branches principales qui correspondent aux 4 environnements distants:

- `master` : environnement de production;
- `develop` : environnement de développement;
- `release` : environnement de test et d'intégration;
- `hotfix` : environnement de maintenance des versions en production;

Le workflow est le suivant :

- Une branche de développement (develop) est créée à partir de master.
- Une branche de version (release) est créée à partir de develop.
- Les branches de fonctionnalité (feature) sont créées à partir de develop.
- Lorsqu'une branche de fonctionnalité (feature) est terminée, elle est mergée dans la branche de développement (develop).
- Lorsque la branche de version (release) est terminée, elle est mergée dans les branches de développement (develop) et principale (master).
- Si un problème est identifié dans la branche principale (master), une branche de maintenance (hotfix) est créée à partir de master.
- Une fois la branche de maintenance (hotfix) terminée, elle est mergée dans les branches de développement (develop) et principale (master).

Resultats Attendus

En utilisant la documentation mise à disposition et les informations vues en cours, mettez à jour le code du fichier `gitlab-ci.yml` pour préparer une automatisation et une industralisation cohérente avec le pattern GitFlow.

- Définissez des `stages` pour différencier les grandes étapes (build / test / deploy / ..)
- Définissez des `jobs` pour préciser les scripts à exécuter pour chaque segmentation.
- Utilisez le keyword `only` pour limiter les actions à des branches en particulier.
- Utilisez le keyword `rules` pour illustrer des cas particuliers.
- Utilisez le keyword `environment` pour définir des environnements configurés.
- Utilisez tout autre fonctionnalité qui vous semblera pertinente.

Mise à part l'exécution réelle des tests unitaires de l'application, les processus de build, tests et déploiement seront simulés à l'aide d'instructions explicites avec `echo`, comme dans les exemples de ce support.

Il est attendu de vous un lien vers un projet GitLab public. Ce projet sera évalué sur les critères suivants:

- Cohérence des branches, dépendances et ordre d'exécution des processus mis en place.
- Fichier de configuration `gitlab-ci.yml` épuré et lisible.
- Utilisation des notions théoriques issues du bloc de compétences lié aux tests automatisés.
- Présence et historique d'exécution de jobs illustrant votre travail (au moins un pipeline Succès par branche/environnement).