









# TESE510 - Industrialisation des Processus de Tests - Support

## Partie 2

 Owner	 Julien Couraud
 School	EPSI Bordeaux
 Promotion	Bachelor 3
 Cursus	DEV FS - IA
 Year	2024-2025

### 1. GitFlow

### 2. Circle CI

2.1 Gestion des membres et des rôles

2.2 Gestion des branches et environnements

2.3 Setup et connexion avec Github

### 3. Langage YAML et configuration

3.1 Le langage YAML

3.2 Le fichier 'config.yml' pour CircleCI

3.3 Mots-clés utiles

### 4. TP - Projet individuel évalué

4.1 Setup du projet Next.js avec CircleCI

4.2 Cahier des charges du projet

4.3 Modalité d'évaluation et rendus attendus

## 1. GitFlow

Gitflow est un modèle de gestion de branches dans Git qui est largement utilisé dans les projets de développement logiciel. Il définit une structure de workflow

pour les branches Git, en divisant le cycle de vie d'un projet en plusieurs phases clés, chacune représentée par une branche Git distincte.

Le modèle Gitflow est basé sur deux branches principales :

- La branche "**master**" ou "**main**": elle contient les versions stables du code, prêt à être déployé en production.
- La branche "**develop**" : elle contient la dernière version du code en cours de développement.

En plus de ces deux branches principales, Gitflow définit également plusieurs branches secondaires, chacune ayant un rôle spécifique dans le workflow de développement. Ces branches sont les suivantes :

- La branche "**feature**" : elle est utilisée pour le développement de nouvelles fonctionnalités. Elle est créée à partir de la branche "develop" et est fusionnée dans "develop" une fois que la fonctionnalité est terminée.
- La branche "**release**" : elle est utilisée pour la préparation d'une nouvelle version du code qui sera déployée en production. Elle est créée à partir de la branche "develop" et est fusionnée dans "master" une fois que la version est prête.
- La branche "**hotfix**" : elle est utilisée pour corriger rapidement les bugs critiques dans la version en production. Elle est créée à partir de la branche "master" et est fusionnée dans "master" et "develop" une fois que le correctif est effectué.

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

## 2. Circle CI

CircleCI est une plateforme d'intégration et de déploiement continu qui automatise la construction, les tests et le déploiement des applications. Elle s'intègre avec les dépôts Git et propose des outils simplifiant le flux de travail des équipes de développement.

La plateforme utilise un fichier de configuration (nommé `.circleci/config.yml`) pour définir les pipelines d'intégration continue. Ce fichier précise les étapes du pipeline, les tâches à exécuter, les dépendances et les conditions d'exécution.

CircleCI permet également le déploiement continu, offrant aux équipes la possibilité de déployer leurs applications automatiquement sur différents environnements une fois les tests validés.

## 2.1 Gestion des membres et des rôles

- **Les membres** : Les utilisateurs peuvent être ajoutés à des projets et des groupes, et leur niveau d'accès peut être défini en fonction de leur rôle. CircleCI offre également une fonctionnalité de gestion des invités pour les utilisateurs externes qui doivent collaborer sur un projet spécifique.
- **Les rôles** : CircleCI propose des rôles prédéfinis pour les membres de l'équipe, tels que Owner, Maintainer, Developer et Guest.

## 2.2 Gestion des branches et environnements

- **Les branches** : CircleCI permet de gérer les branches de code pour chaque projet et est donc synchronisé avec toutes les entités de votre architecture DevOps (outil de versionning en local et à distance).
- **Les environnements** : CircleCI permet de définir des environnements pour chaque projet. Chaque environnement peut être associé à une branche spécifique et peut être utilisé dans le fichier de configuration.

Documentation officielle: <https://circleci.com/docs/>

## 2.3 Setup et connexion avec Github

- Rendez-vous sur le site de CircleCI: <https://circleci.com/>
- Connectez-vous avec votre compte GitHub.
- Utilisez les explications de la plateforme CircleCI pour créer un nouveau projet basé sur un répertoire de votre compte Github, il est possible de générer une connexion ssh avec votre poste de travail, pour cela:
  - Générez une paire de clés (publique et privée) en local sur votre machine.

```
ssh-keygen -t ed25519 -f ~/.ssh/project_key -C email@example.com
```

- Saisissez la clé publique ( `~/.ssh/project_key.pub` ) dans la partie `Settings>Deploy keys` de votre projet Github via l'interface web.

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/managing-deploy-keys#set-up-deploy-keys>

- Saisissez la clé privée ( `~/.ssh/project_key` ) dans le formulaire de création de projet CircleCI.
- Finalisez la liaison entre vos différentes entités DevOps en sélectionnant le projet et en validant le formulaire de création de projet CircleCI.

! Vous pouvez utiliser la génération d'un fichier de configuration automatique suggérée par l'outil, mais celui-ci risque d'aboutir à un échec dans le pipeline. Notre application n'est pas prête encore pour générer un build correcte sur la branche main !

Il sera possible de créer votre fichier de configuration

`.circleci/config.yml` sur votre projet par la suite et tester pour la première fois l'exécution d'un pipeline déclenché par un changement d'état sur votre outil de versionning.

## 3. Langage YAML et configuration

### 3.1 Le langage YAML



YAML (*Yet Another Markup Language*, puis rebaptisé *YAML Ain't Markup Language*) est un langage de sérialisation de données, conçu pour être lisible par les humains et facile à manipuler. Il est souvent utilisé pour la configuration d'applications, l'échange de données entre services et le stockage d'informations structurées.

Ce langage a été créé en **2001** par *Clark Evans*, avec la contribution de *Brian Ingerson* et *Oren Ben-Kiki*. L'objectif initial était de proposer une alternative plus simple et plus lisible à **XML** et **JSON** pour la configuration de processus automatisés et industrialisés.

Principes fondamentaux :

- **Lisibilité humaine** : Syntaxe claire et proche du langage naturel.
- **Simplicité** : Moins verbeux que XML, plus intuitif que JSON.
- **Structure hiérarchique** : Basé sur l'indentation, comme Python.
- **Prise en charge des types de données** : Nombres, chaînes, booléens, listes, dictionnaires, etc.
- **Extensibilité** : Possibilité d'ajouter des références et des ancres pour éviter la répétition de données.

Objectif :

- **Fichiers de configuration** (*ex: Kubernetes, Ansible, Docker Compose, CI/CD pipelines*).
- **Échange de données entre applications.**
- **Sérialisation d'objets dans divers langages de programmation.**

Concurrents et alternatives:

- **JSON** : Plus répandu, utilisé dans les APIs web, mais moins lisible pour les fichiers de configuration.
- **XML** : Plus puissant pour les documents complexes, mais très verbeux.
- **TOML** : Similaire à YAML, plus strict et structuré, souvent utilisé pour la configuration (*ex: fichiers de config Rust*).
- **INI** : Ancien format de configuration, plus limité que YAML.

Inconvénients et faiblesses

- **Indentation obligatoire** : Source fréquente d'erreurs (comme en Python).
- **Ambiguïtés de syntaxe** : Certains formats peuvent être interprétés différemment selon les implémentations.
- **Performances** : Moins optimisé que JSON pour certaines applications à grande échelle.

## 3.2 Le fichier 'config.yml' pour CircleCI

- **workflows** : CircleCI utilise des workflows pour organiser et exécuter des jobs en fonction de certaines conditions. Chaque workflow est déclenché par des événements spécifiques, tels que le push sur une branche ou l'ouverture d'une pull request.
- **jobs** : Les jobs représentent des tâches spécifiques à exécuter dans le cadre d'un workflow. Chaque job peut contenir des étapes spécifiques, telles que la construction d'un conteneur Docker, l'exécution de tests unitaires ou le déploiement sur un environnement en particulier. Les jobs peuvent également spécifier des variables d'environnement, des commandes à exécuter et des conditions pour l'exécution de la tâche.
- **orbs** : Les orbs sont des paquets de configuration réutilisables qui simplifient la définition des jobs et des workflows. Ils peuvent être partagés entre différents projets et facilitent la maintenance des configurations CI/CD.

```
# .circleci/config.yml
version: 2.1
orbs:
  node: circleci/node@5
jobs:
  # First job declared is for building things..
  build:
    executor: node/default
    steps:
      - checkout
      - run: echo "Building things..."
  # Second job declared is for launching unit tests with Jest
  unit-tests:
    executor: node/default
    environment:
      JEST_JUNIT_OUTPUT_DIR: ./test-results/
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
```

```

      command: npm install jest-junit
    - run:
      name: Run tests
      command: npm run test:ci
    - store_test_results:
      path: ./test-results/
workflows:
  version: 2
  build_and_test:
    jobs:
      - build
      - unit-tests:
        requires:
          - build

```

Dans cet exemple, il y a deux jobs : `build` et `unit-tests`. Le job `build` construit l'application, et le job `unit-tests` exécute les tests après la construction. Le workflow `build_and_test` organise l'exécution de ces jobs, en spécifiant que `unit-tests` dépend de `build` via le mot clé `requires`.

### 3.3 Mots-clés utiles

Nous avons vu comment ordonner des jobs à la suite, maintenant nous devons nous assurer que certains jobs s'exécute conditionnellement dans notre workflow.

Prenons l'exemple des jobs de déploiement. Nous ne souhaitons pas déployer notre application à chaque modification sur tous les environnements disponibles. Nous devons donc filtrer l'exécution de nos jobs de déploiement en fonction de la branche sur laquelle une modification est poussée. Pour cela nous utiliserons les mots-clés `filters`, `branches` et `only` comme dans l'exemple ci-dessous.

```

# .circleci/config.yml

# ...

jobs:

# ...

```

```
deploy_dev:
  executor: node/default
  steps:
    - checkout
    - run: echo "Deploying to development environment..."
```

```
deploy_prod:
  executor: node/default
  steps:
    - checkout
    - run: echo "Deploying to production environment..."
```

```
workflows:
  version: 2
```

```
build-test-deploy:
```

```
  jobs:
```

```
    - build
```

```
    - unit_tests:
```

```
      requires:
```

```
        - build
```

```
    - deploy_dev:
```

```
      requires:
```

```
        - unit_tests
```

```
      filters:
```

```
        branches:
```

```
          only:
```

```
            - develop
```

```
    - deploy_prod:
```

```
      requires:
```

```
        - unit_tests
```

```
      filters:
```



```
branches:
  only:
    - main
```

Il existe d'autres mots-clés disponibles pour conditionner qui ont chacun leurs particularités. La documentation CircleCI permet d'approfondir les spécificités de l'outil.

- **when** permet de définir des conditions pour l'exécution des jobs. Il peut prendre des valeurs telles que "on\_success", "on\_failure", "on\_hold", etc. Par exemple, voici comment on peut utiliser "when" pour exécuter un job uniquement en cas de succès :

```
jobs:
  deploy:
    executor: node/default
    steps:
      - checkout
      - run: echo "Deploying..."
    when: on_success
```

Dans cet exemple, le job "deploy" ne sera exécuté que si les jobs précédents dans le workflow ont réussi.

Documentation: <https://circleci.com/docs/2.0/configuration-reference/#the-when-attribute>

- **context** permet de référencer des contextes de sécurité, qui stockent des variables d'environnement sensibles, telles que des clés d'API ou des informations d'identification.

```
jobs:
  deploy:
    executor: node/default
    steps:
      - checkout
      - run: echo "Deploying..."
    environment:
      CONTEXT: my-credentials-context
```

Dans cet exemple, le job "deploy" peut accéder aux variables d'environnement stockées dans le contexte de sécurité "my-credentials-context".

Documentation: <https://circleci.com/docs/2.0/configuration-reference/#context>

## 4. TP - Projet individuel évalué

### 4.1 Setup du projet Next.js avec CircleCI



Vous l'aurez compris, dans le workflow GitFlow, tout part d'une branche "develop", qui sera ensuite naturellement mergée dans des branches relatives aux environnements distants nécessaires au bon déroulement des développements du produit.

- Creation d'une branche "develop":

```
git checkout -b develop
```

```
git branch -a
```

- Si vous utilisez un outil comme Github Desktop ou même Git au sein de votre IDE, peut être aurez-vous besoin de modifier les droits d'accès à votre projet par votre outil de versionning.

```
sudo chown -R "${USER:-$(id -un)}" .
```

- Nous avons besoin sur la branche "develop" d'initialiser notre fichier de configuration en YAML. Pour cela, comme indiqué dans la procédure sur CircleCI, nous aurons besoin d'un fichier "config.yml" situé dans un dossier ".circleci" à la racine de l'arborescence de l'application. Voici un exemple de fichier de configuration qui nous permettra d'exécuter dans un premier temps nos tests existants dans un pipeline CircleCI:

```

version: 2.1
orbs:
  node: circleci/node@5
jobs:
  test-node:
    # Install node dependencies and run tests
    executor: node/default
    environment:
      JEST_JUNIT_OUTPUT_DIR: ./test-results/
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          command: npm install jest-junit
      - run:
          name: Run tests
          command: npm run test:ci
      - store_test_results:
          path: ./test-results/
workflows:
  build-and-test:
    jobs:
      - test-node

```

- Enfin voici un exemple de comment rapatrier une branche vers une autre (en l'occurrence ici "develop" vers "main")

```

git checkout master
git merge development
git push -u origin master

```

- Vous observerez, sur CircleCI, qu'à chaque changement d'état sur une branche, un pipeline est instancié et les instructions du fichier de configuration sont exécutés automatiquement. Vous avez donc à votre disposition un socle solide d'intégration continue

## 4.2 Cahier des charges du projet

Vous êtes un leader technique, au sein d'une ESN, et vous vous apprêtez à démarrer un projet d'application Web suite à une réponse favorable à un appel d'offre.

Avant le démarrage des développements, vous êtes chargés de préparer l'automatisation des processus de build, des différents types de tests et des déploiements continus.

Vous héritez d'un répertoire sur l'outil de CI/CD CircleCI ainsi qu'un cahier des charges à respecter. Votre tâche sera donc dans un premier temps de configurer cet outil afin qu'il puisse structurer les automatisations liées aux développements et à la méthodologie de gestion de projet Agile.



Vous devrez mettre en place une configuration pertinente avec CircleCI pour faire vivre l'application qui sera basée sur le framework Next.js. Votre tâche principale est donc de configurer l'outil et de compléter le fichier de configuration `.circleci/config.yml` selon les objectifs et contraintes définis.

- **Branchage GitFlow et mise en place des environnements**

L'objectif est de déployer l'application sur trois environnements distincts :

- Un environnement "development" correspondant à la branche "develop".
- Un environnement "integration" correspondant à la branche "integration".
- Un environnement "production" correspondant à la branche "main".

Le workflow de branches doit respecter le pattern GitFlow, et vous êtes chargés de créer les branches principales ainsi que de gérer les branches temporaires "feature/xxx" et "hotfix/xxx" ( `/^feature.*/` et `/^hotfix.*/` ) directement dans le fichier `.circleci/config.yml` .

- **Les différents jobs au sein du stage "build" :**

- Installation des dépendances
- Analyse de code

- Cleaning et Packaging
- **Les différents jobs au sein du stage "tests" :**
  - Tests unitaires
  - Tests d'intégration
  - Tests de régression
  - Tests de performance
  - Tests de sécurité
  - Tests de compatibilité
  - Tests d'accessibilité
- **Les différents jobs au sein du stage "deploy" :**
  - Préparation de l'environnement de déploiement
  - Déploiement de l'application
  - Tests de vérification
  - Tests de validation fonctionnelle
  - Tests de charge
  - Déploiement
  - Surveillance et suivi
- **Contraintes et cas particuliers**
  - Appliquez votre compréhension acquise en cours pour définir des filtres d'exécution de certains jobs dans des cas particuliers. Par exemple, il peut être judicieux de ne pas exécuter les jobs de déploiement lorsqu'une modification est apportée sur les branches "feature/xxx" et "hotfix/xxx". De même, certains jobs de tests peuvent être évités à certains moments de la phase de développement.
  - Evidemment nous n'aurons pas le temps dans le cadre de ce module d'implémenter de réels stacks de builds, de tests et de déploiements. Votre travail est uniquement préparatoire pour les équipes de

développement et donc vous simulerez les process à l'intérieur des différents jobs proposés à l'aide de la commande "echo"

### 4.3 Modalité d'évaluation et rendus attendus

Votre travail sera évalué en fonction de la manière dont vous appliquez les connaissances théoriques vues en cours à la configuration du fichier

`.circleci/config.yml`. Voici les critères d'évaluation :

- Respect du pattern GitFlow dans la gestion des branches.
- Création correcte des environnements dans CircleCI.
- Définition cohérente du `workflow` pour différencier les grandes étapes du processus CI/CD.
- Définition pertinente des `jobs` pour préciser les scripts à exécuter à chaque étape.
- Utilisation efficace des keywords `filters` et/ou `when` pour définir des conditions pour l'exécution des jobs.
- Utilisation appropriée du keyword `requires` pour définir des ordres d'exécution de jobs.
- Commentaires pertinents dans le fichier pour expliquer le flux et les décisions prises.



Il est attendu de vous pour l'évaluation un dossier de type documentation technique contenant et expliquant au minimum:

- la gestion des branches
- la description du workflow de travail des équipes de développements pendant la réalisation du produit en agilité et la description technique des jobs que vous avez préparés.
- des captures d'écrans des pipelines exécutés avec succès sur les différentes branches préparées.
- un lien vers un répertoire Github **public** contenant un README professionnel et les sources de votre projet et notamment le fichier de configuration CircleCI correctement structuré et commenté.