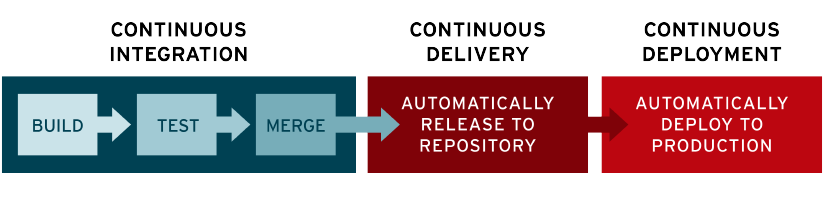
**Approche CI/CD**

# Schéma global



# Vision détaillée

### Objectif :

* Modification régulière de code et ajout de nouvelles fonctionnalités
* Vérification du bon fonctionnement des nouvelles briques de codes
* Intégration des nouvelles briques de codes entre elles et avec le code précédent sur un environnement de test dédié
* Déploiement en production et génération automatique des fichiers nécessaire au déploiement global de l’application

### Pourquoi :

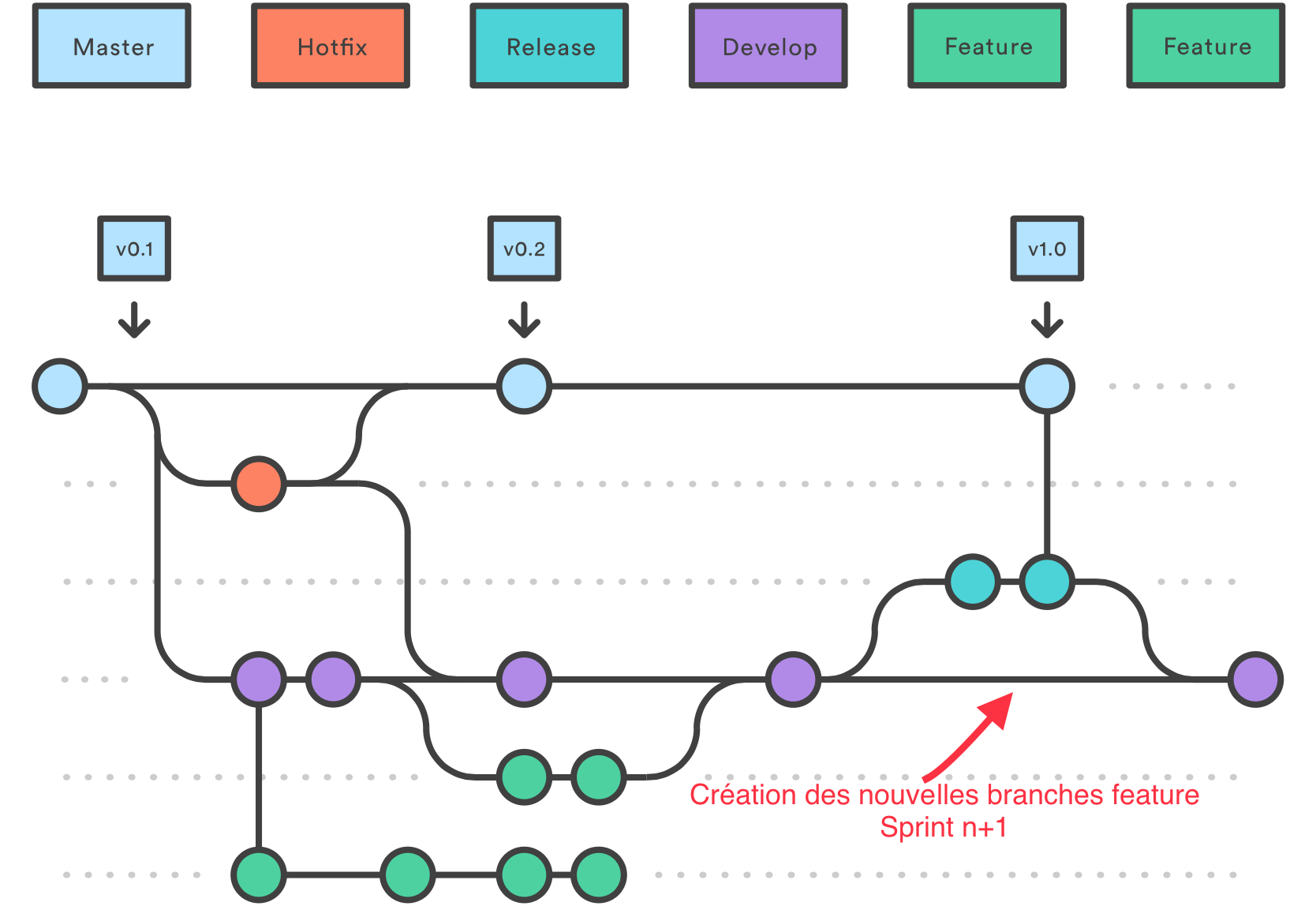
L’approche CI/CD permet de travailler plus efficacement à plusieurs sur une même application. En effet, la régularité des commits et des push permet d’éviter les dysfonctionnement générés par deux personnes travaillant sur le même code pendant trop longtemps. (Différents écarts dans des fonctions modifiées qui ne font plus la même chose mais qui sont toutes deux utiles sur les deux versions du code).

La solution apportée par l’approche CI/CD est de permettre aux développeurs de commit régulièrement sur des nouvelles branches, chaque fonctionnalités développées, qui sont ensuite mergées sur une branche de développement global où toutes les briques sont réunies et testées via des tests d’intégrations et unitaires, permettant en cas de dysfonctionnement de retrouver la source du problème rapidement et efficacement.

La branche de développement est déployée dans un environnement de test dockerisé afin de pouvoir vérifier le bon fonctionnement de l’application.

Enfin, le déploiement de l’application en production est automatisé une fois que tous les voyants sont au vert. Le but étant de soulager l’équipe de développement de tâches rébarbatives et où l’erreur est facile et rapide.

Intégration continue



#### Explications :

Un développeur travaillant sur une nouvelle fonctionnalité (Feature)

-> Création d’une nouvelle branche

**CONVENTION DE NOMMAGE :** Feature/”Nom de la feature”

Exemple : Feature/Login

La feature semble complétée

-> Merge request sur Develop

-> Merge automatisée, lancement des tests unitaires et d’intégration

**KO** -> Revue du code sur la branche Feature et corrections

**OK** -> Passage à la suite

Le sprint est terminé ET la date de release de la nouvelle version approche

-> Merge automatique sur de Dévelop sur Release, nouvelle batterie de tests

-> Utilisation de l’environnement de test dockerisé

**KO** -> Revue du code et corrections

**OK** -> Passage à la suite

Déploiement automatique de la nouvelle version sur la branche Master

-> Merge de la branche Release sur la branche Master

**CONVENTION DE NOMMAGE COMMIT:** ”Version X.X”

Exemple : Version 1.1

-> Déploiement d’un nouvel APK et mise à jour sur le playstore

Dans un premier temps, les Hotfixs ne seront pas pris en compte. Ce document est amené à évoluer avec le temps.

Choix technique  
**Nous** avons exploré trois outils d'intégration continue.

Jenkins :

* -  open source
* -  compatible avec Window, Linux, Mac Os
* -  configuration possible depuis l’interface graphique

Gitlab ci/cd

* -  open source avec gitlab ce (community edition)
* -  abonnement pour des fonctionnalités supplémentaires
* -  un outil d’auto devops
* -  possibilité d’utiliser la fonctionnalité multiple-runners

Github Actions

* -  open source
* -  possibilité d’utiliser la fonctionnalité multiple-runners
* -  interface graphique intuitive.
* -  des pipelines préconfigurés pour tous les environnements
* -  prise en main facile

Nous avons choisi github Actions car :

* -  github est le choix d’Epitech pour héberger les projets académiques
* -  prise en main facile de la partie CI/CD
* -  il dispose déjà de pipelines préconfigurés pour les technos de notre projet
* -  c’est un nouvel outil puissant que nous voulons explorer

**Prise en main**

Deux possibilités

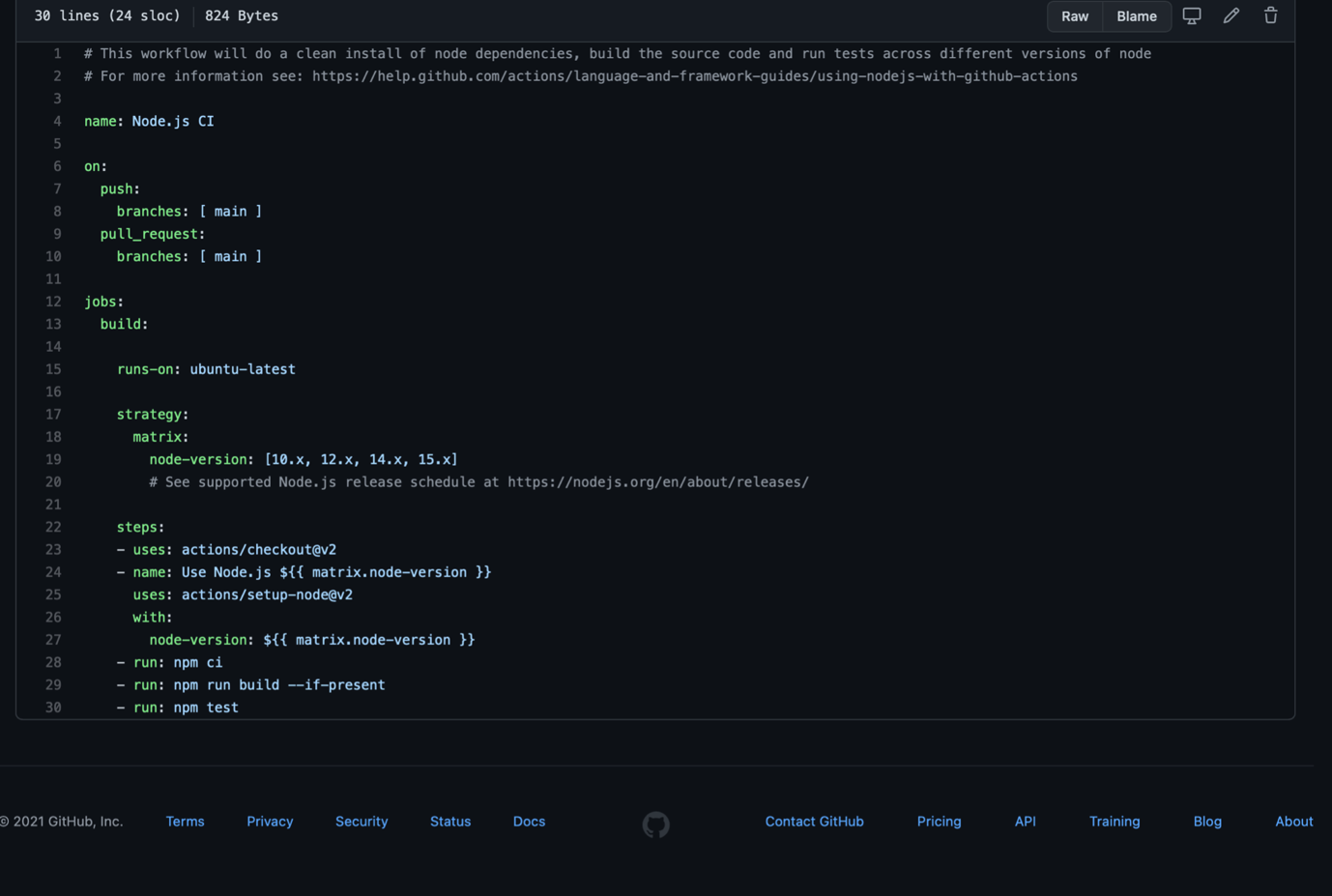
1. Création manuelle

* -  à la base du projet, créer un dossier .github/workflows/
* -  créer un fichier yaml du nom de votre choix
* -  c’est dans ce fichier que vous écrirez la configuration de votre pipeline
* -  Si vous choisissez le commit comme action de déclenchement du

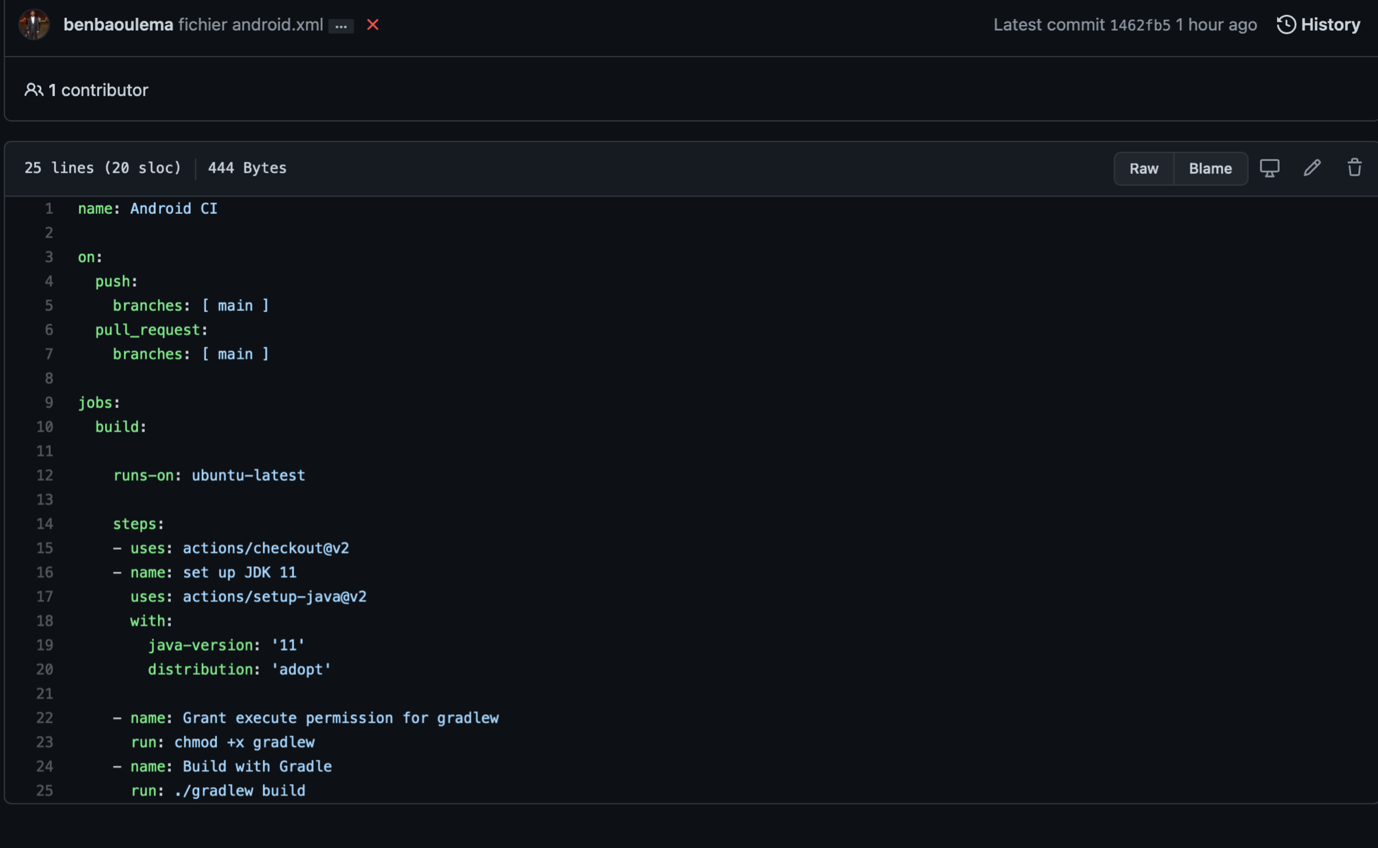
pipeline, alors à chaque nouveau commit sur la branche définie, les jobs sont lancés.

2. Choix d’un pipeline prédéfini

* -  créer le projet sur github
* -  aller dans l’onglet *Actions*
* -  choisir la techno de votre projet
* -  lancer enregistrer les changements
* -  après le commit la pipeline est lancé



Configuration du CI de notre api en nodejs



Configuration du CI de notre application Android

IMPLEMENTATION DU WORKFLOW

* 1. Build de l’application Android

Lorsqu’une branche reçoit une demande de merge, l’application est buildé, testé avant de valider manuellement la demande

* 1. Automatisation des pull request

Chaque nouveau commit declanche une merge request vers la branche supérieure. La demande doit être validé manuellement par un mainteneur

* 1. Suppression automatique d’une branche après la validation d’une merge request

Pour chaque nouvelle fonctionnalité de l’application, une branche feature est créée pour permettre au développeur de travailler tranquillement en local.

Lorsque le développement de la fonctionnalité, le développeur fait une commit sur la branche feature, cela declanche une merge request et lorsque la demande de fusion est validée, la branche develop est supprimée automatiquement

CREATION D’UNE NOUVELLE BRANCHE feature3

Une image contenant texte

Description générée automatiquement

Une image contenant texte, capture d’écran, moniteur, écran

Description générée automatiquement

LA NOUVELLE APPARAIT

COMMIT SUR LA BRANCHE feature3

Une image contenant texte, capture d’écran, écran

Description générée automatiquement

UNE ACTION EST DECLANCHEE AVEC DEUX JOBS

Une image contenant texte, moniteur, capture d’écran, noir

Description générée automatiquement

NOUVELLE DE DEMANDE DE FUSION

Une image contenant texte, capture d’écran, moniteur, écran

Description générée automatiquement

LA BRANCHE feature3 EST SUPPRIMEE

Une image contenant texte, capture d’écran, moniteur, intérieur

Description générée automatiquement