

Le CI/CD avec Jenkins

Sommaire :

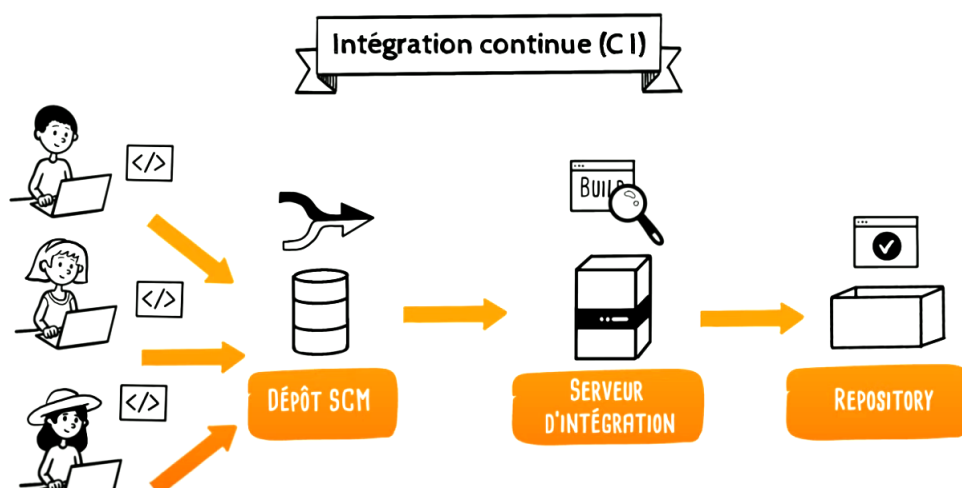
Définition de CI/CD	3
Jenkins	4
Prérequis :	4
Installation :	5
Installer Jenkins sur Mac	5
Installer Jenkins sur Linux (Ubuntu)	5
Installer Jenkins sur Windows	6
Configuration minimale	9
Utilisation de Jenkins	12
1 - Créer un simple job	12
2 - Créer une pipeline	15
3 - Créer une pipeline avec un fichier Jenkinsfile	17
CAS concret : mise en place d'une pipeline qui est déclenché suite à un changement sur le dépôt GitHub	20
Comment créer des tests unitaires avec Cucumber.io	23
Qu'est-ce qu'un test unitaire ?	23
Cucumber :	23
Installer Cucumber dans un projet :	24
Installer Cucumer sur Node.js :	24
Création du 1er scénario	26
5 - Référence	28

1. Définition de CI/CD

Le **CI/CD** est un ensemble de méthodes et de pratiques qui permet **d'automatiser et d'accélérer le déploiement** d'une application. Un processus de déploiement est long et passe par différentes étapes avant une mise en production :

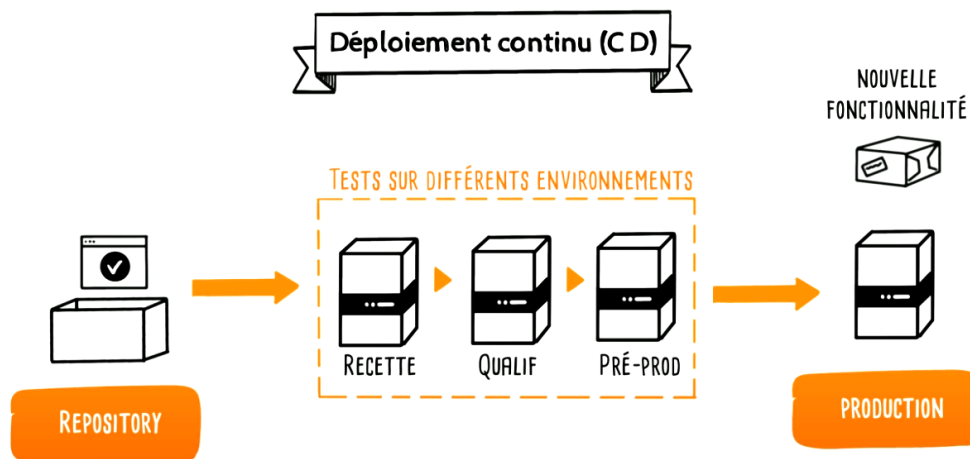
- Développement
- Test et intégration
- Recette
- Mise en production

La méthode **CI** (intégration continue) **cherche à automatiser les opérations de développements** et les tests d'intégration. À chaque modification du code, des **tests automatisés** sont effectués pour assurer le déploiement de l'application. Il est possible de créer ces scripts de tests pour qu'ils s'exécutent à chaque commit sur un système de contrôle de version (GitHub, GitLab...).



La méthode **CD** (déploiement continu ou livraison continue) cherche à **automatiser les opérations de recette et mise en production**. Une fois l'application validée sur l'environnement de test, le déploiement de l'environnement se fera automatiquement dans le cas d'un déploiement continu.

Dans le cas de la livraison continue le processus sera le même mais nécessitera une **intervention manuelle** afin d'effectuer le déploiement.



Avantages	Inconvénients
<ul style="list-style-type: none"> • Supprimer les tâches humaines récurrentes et chronophages • Automatise et accélère le processus de déploiement • Automatise et accélère le processus d'intégration • Marge d'erreur limitée grâce aux tests automatisés du serveur d'intégration (Jenkins) • Alerte les développeurs si des tests unitaires ont échoué • Favorise le time to market* • Limite les coûts de production 	<ul style="list-style-type: none"> • Trop automatiser les tâches réduit la vigilance humaine • Tout le monde n'apprécie pas le changement continu • Conflit entre les différents microservices

2. Jenkins

Il s'agit d'outil d'intégration continue de type CI (intégration continue), Jenkins est une application open source développée en Java, dédié aux DevOps*. À chaque modification du code Jenkins se charge automatiquement de recompiler l'application pour la création d'un nouveau build.

2.1. Prérequis :

Techno	Prérequis minimum
Espace disque	1 Go (10 Go pour Docker)
RAM	256 Mo
OS	Windows, Linux, MAC OS, Docker

Java	version 8 ou 11
------	-----------------

2.2. Installation :

- **Installer Jenkins sur Mac**

Étape 1 : Installer le gestionnaire de package Homebrew

Installez Homebrew à partir du lien suivant : <https://brew.sh/>.

Pour vérifier que l'installation a bien été effectuée, utilisez la commande suivante.

```
brew --version
```

Étape 2 : Installer le paquet Jenkins

Écrivez les lignes de commande suivantes dans le terminal.

```
brew install jenkins-lts
```

Maintenant, lançons notre service Jenkins, en utilisant la commande suivante.

```
brew services start jenkins-lts  
brew services stop jenkins-lts  
brew services restart jenkins-lts
```

Après avoir démarré le service, nous pouvons aller à l'URL de Jenkins, qui est localhost:8080.

- **Installer Jenkins sur Linux (Ubuntu)**

Étape 1 : Ajouter le dépôt Jenkins

Téléchargez et ajoutez la clé de registre.

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |  
sudo apt-key add -
```

Ajoutez l'adresse du registre.

```
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >  
/etc/apt/sources.list.d/jenkins.list'
```

Vérifiez les mises à jour pour tenir compte du nouveau registre de Jenkins.

```
sudo apt update
```

Installez Jenkins.

```
sudo apt install jenkins
```

Étape 2 : Démarrer Jenkins

Lancer le service Jenkins

```
sudo systemctl start jenkins
```

Vérifiez l'état du service Jenkins

```
sudo systemctl status jenkins
```

Le résultat devrait être le suivant :

```
> sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
   Drop-In: /etc/systemd/system/jenkins.service.d
            └─override.conf
   Active: active (running) since Wed 2022-02-23 08:23:02 +04; 1h 41min ago
 Main PID: 8016 (java)
    Tasks: 48 (limit: 9318)
   Memory: 2.3G
    CGroup: /system.slice/jenkins.service
            └─8016 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot
```

Après avoir démarré le service, nous pouvons aller à l'URL de Jenkins, qui est localhost:8080.

● Installer Jenkins sur Windows

Étape 1 : Télécharger Jenkins

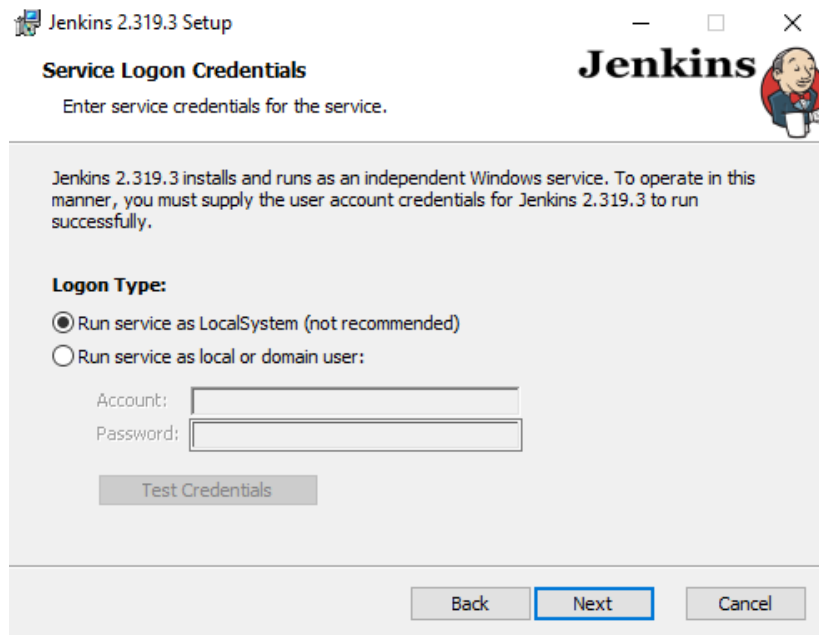
Télécharger Jenkins avec le lien suivant : <https://www.jenkins.io/download/>

Étape 2 : Installation de Jenkins

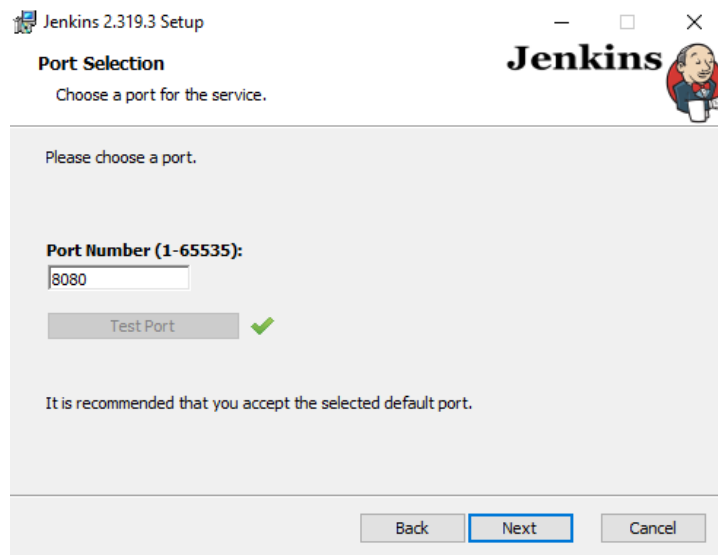
Commençons par lancer l'exécutable Jenkins et cliquer sur 'next'.



Veillez à bien sélectionner "run service as localSystem" car nous n'avons pas de compte.

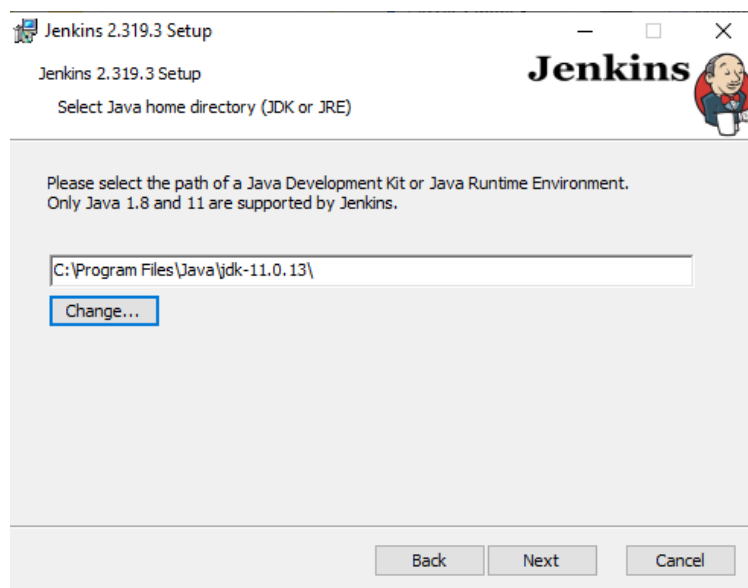


Puis choisissez un port et vérifiez-le :



Enfin, sélectionnez l'emplacement où se trouve Java, celui que Jenkins utilisera pour son bon fonctionnement. N'oubliez pas que Jenkins n'est compatible qu'avec la version 8 ou la version 11 de java !

Note : la version 8 peut causer quelques problèmes, je vous recommande la 11 :)



Il ne vous reste plus qu'à terminer l'installation.

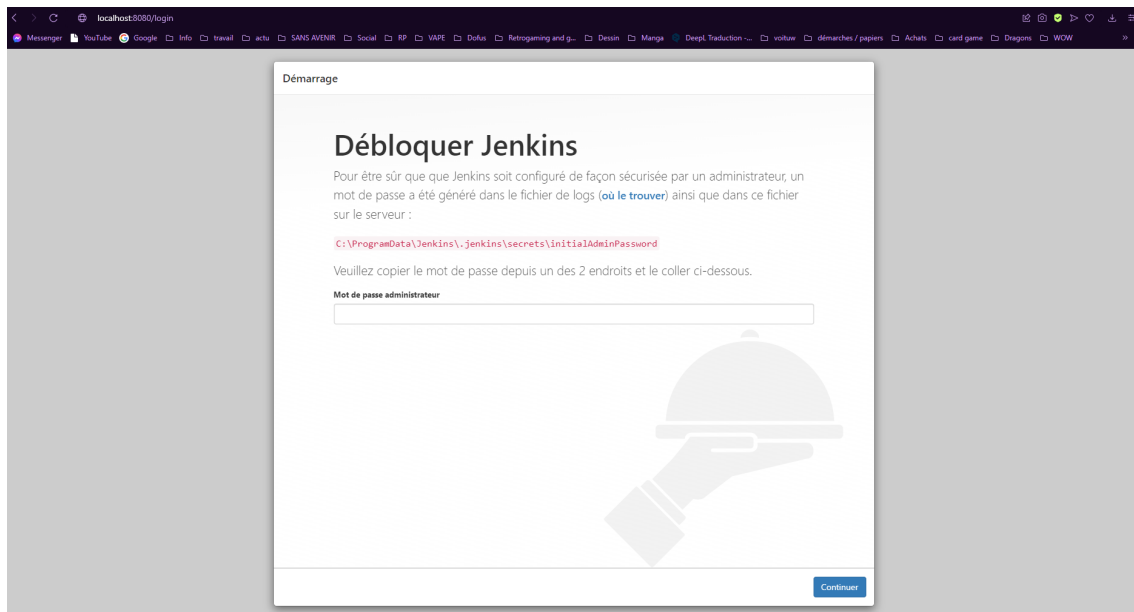
Une fois terminé pour vérifier que tout s'est bien passé, ouvrez votre gestionnaire de tâches et dans la section "services" vérifiez que Jenkins est présent et en cours d'exécution.

Si le service est correctement démarré, nous pouvons nous rendre à l'URL de Jenkins, qui est localhost:8080 par défaut.

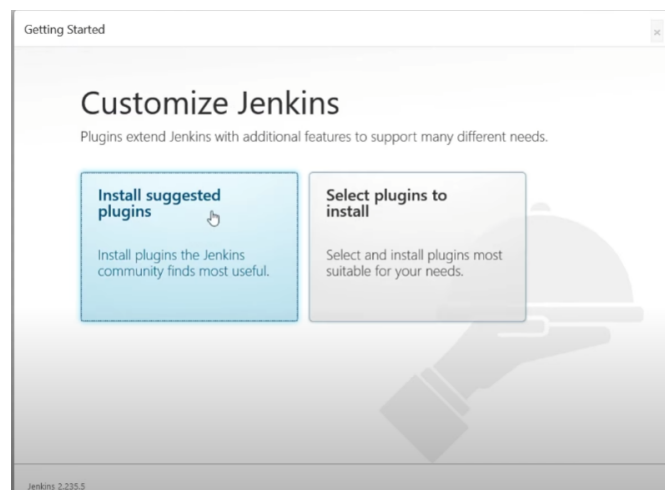
3. Configuration minimale

Si nous entrons dans l'URL de Jenkins pour la première fois, nous devons configurer Jenkins pour qu'il soit opérationnel.

Tout commence par la récupération du mot de passe initial fourni par Jenkins à l'adresse indiquée à l'écran (le chemin d'accès avec une police et un fond rouges). Ensuite, on clique sur continuer.



Cliquez ensuite sur "Install suggested plugins", pour installer les plugins suggérés.



Une fois les installations terminées, il faudra créer un premier compte administrateur.

Démarrage

Créer le 1er utilisateur Administrateur

Nom d'utilisateur:

Mot de passe:

Confirmation du mot de passe:

Nom complet:

Adresse courriel:

Jenkins 2.319.3

Continuer en tant qu'Administrateur

Sauver et continuer

Vous pouvez ensuite modifier l'instance si vous le souhaitez, mais vous pouvez aussi la conserver par défaut.

Démarrage

Configuration de l'instance

URL de Jenkins :

http://localhost:8080/

L'URL de Jenkins est utilisée pour fournir l'URL de base pour les liens absolus vers les diverses ressources Jenkins. Cela signifie que cette valeur est nécessaire pour le bon fonctionnement de nombreuses fonctionnalités de Jenkins, notamment les notifications par mail, les mises à jour des statuts des pull requests, et la variable d'environnement BUILD_URL fournie pour les étapes de build.

La valeur par défaut affichée n'est pas encore sauvegardée et est générée à partir de la requête actuelle, lorsque c'est possible. Il est fortement recommandé d'utiliser comme valeur l'URL qui est censée être utilisée par les utilisateurs. Cela évitera des confusions lors du partage ou de la visualisation de liens.

Jenkins 2.319.3

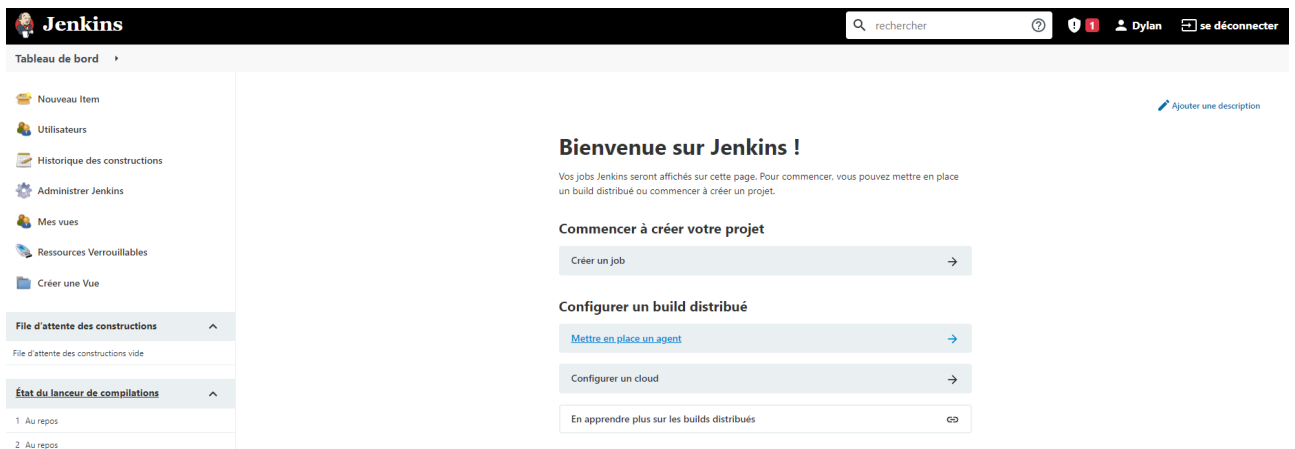
Passer cette étape et terminer

Sauver et terminer

Et voilà, l'installation de Jenkins est terminée.



Cliquez sur " commencer à utiliser Jenkins " et le tableau de bord s'ouvre.



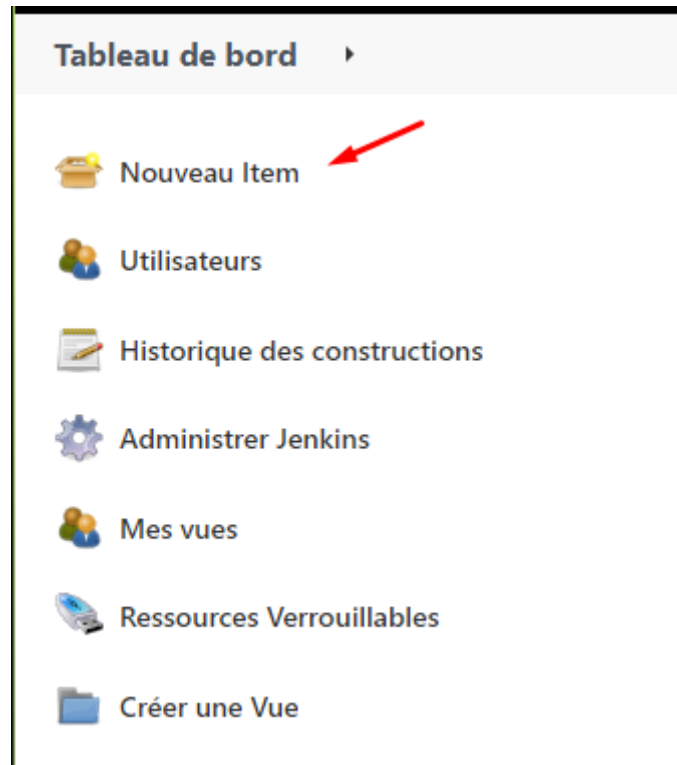
```
C:\Users\sautr>cd C:\Program Files\Jenkins
```

jenkins.exe stop

4. Utilisation de Jenkins

4.1. Créer un simple job

Pour commencer, cliquer sur “**Nouveau item**” :



Ensuite **saisissez un nom**, sélectionnez “**Construire un projet free-style**” et enfin cliquer sur “**OK**”

Saisissez un nom

» Champ obligatoire



Construire un projet free-style

Ceci est la fonction principale de Jenkins qui sert à build (construire) votre projet. Vous pouvez intégrer tous les outils de gestion de version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour tout autre chose qu'un build logiciel.



Pipeline

Organise des activités de longue durée qui peuvent s'étendre sur plusieurs agents de construction. Adapté pour la création des pipelines (anciennement connues comme workflows) et/ou pour organiser des activités complexes qui ne s'adaptent pas facilement à des tâches de type libre.



Construire un projet multi-configuration

Adapté aux projets qui nécessitent un grand nombre de configurations différentes, comme des environnements de test multiples, des binaires spécifiques à une plateforme, etc.



Dossier

Crée un conteneur qui stocke des objets imbriqués. Utile pour grouper ensemble des éléments. Contrairement à une vue qui n'est qu'un filtre, un dossier crée un espace de nommage distinct, de sorte que vous pouvez avoir plusieurs éléments du même nom tant qu'ils se trouvent dans des dossiers différents.



Organization Folder

Creates a set of multibranch project subfolders by scanning for repositories.

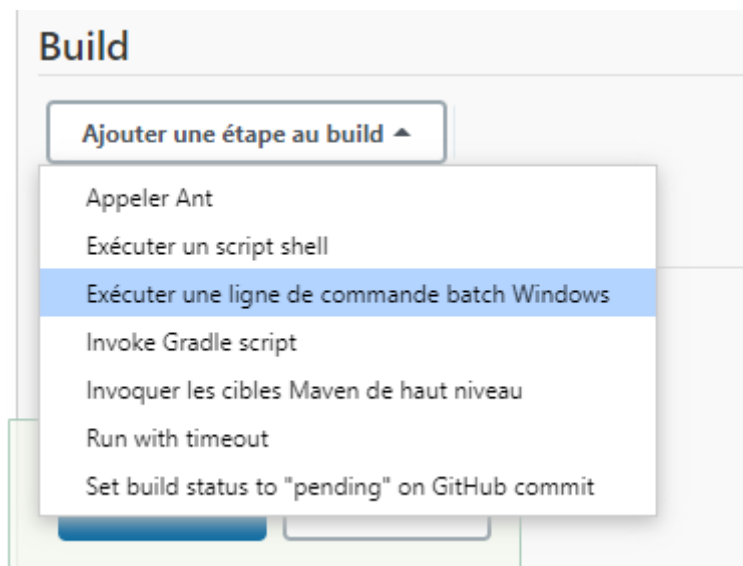


Pipeline Multibranches

Crée un ensemble de projets Pipeline en se basant sur les branches détectées dans le dépôt d'un gestionnaire de code source.

OK

Ensuite dans build, sélectionnez “**Exécuter une ligne de commande batch Windows**”



Ensuite veuillez remplir le champ par le script hello world puis cliquer sur “**sauver**” tout en bas.

Build

Exécuter une ligne de commande batch Windows

Commande

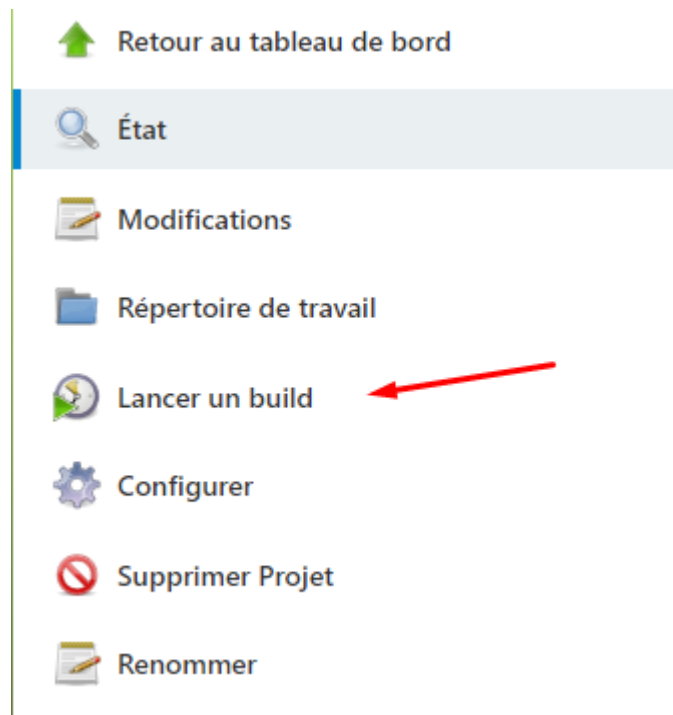
```
echo "hello world"
```

[Voir la liste des variables d'environnement disponibles](#)

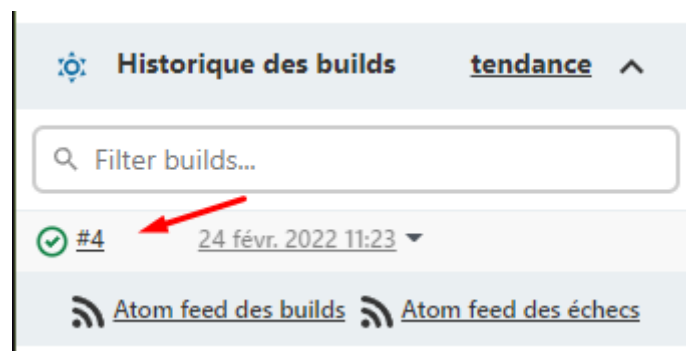
Avancé...

Ajouter une étape au build ▼

C'est fait, pour tester ce job il faut cliquer sur “**Lancer un build**” :




Le build va apparaître maintenant dans l'historique :




On pourra cliquer dessus pour avoir toutes les informations du build.


Et pour vérifier que notre script à bien fonctionner on cliquera sur “**Sortie de la console**”.


 Retour au projet


 État

 Modifications

 Sortie de la console

 Voir en texte brut

 Informations de la construction

 Supprimer le build "#4"

Sortie de la console

Démarré par l'utilisateur **Dylan**

Running as SYSTEM

Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\MonPremierProjet
[MonPremierProjet] \$ cmd /c call C:\WINDOWS\TEMP\jenkins4261153700982801489.bat

```
C:\ProgramData\Jenkins\.jenkins\workspace\MonPremierProjet>echo "hello world"  
"hello world"
```

```
C:\ProgramData\Jenkins\.jenkins\workspace\MonPremierProjet>exit 0  
Finished: SUCCESS
```


Bravo, vous avez créé votre premier job hello world !! :)


2 -Créer une pipeline

Nous allons créer un job de type "pipeline". Il s'agit d'un regroupement de test qui s'exécute les uns à la suite des autres.

Saisissez un nom

» Champ obligatoire

**Construire un projet free-style**
Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet. Vous pouvez intégrer tous les outils de gestion de version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour tout autre chose qu'un build logiciel.

**Pipeline**
Organise des activités de longue durée qui peuvent s'étendre sur plusieurs agents de construction. Adapté pour la création des pipelines (anciennement connues comme workflows) et/ou pour organiser des activités complexes qui ne s'adaptent pas facilement à des tâches de type libre.

Copier collez le script de base de la doc Jenkins puis **"sauver"** :

<https://www.jenkins.io/doc/book/pipeline/jenkinsfile/> .

General
Build Triggers
Advanced Project Options
Pipeline

Pipeline

Definition

Pipeline script

Script ?

```

1 pipeline {
2   agent any
3
4   stages {
5     stage('Build') {
6       steps {
7         echo 'Building..'
8       }
9     }
10    stage('Test') {
11      steps {
12        echo 'Testing..'
13      }
14    }
15    stage('Deploy') {
16      steps {
17        echo 'Deploying....'
18      }
19    }
20  }
21 }

```


try sample Pipeline...


☒ Use Groovy Sandbox ?

Sauver


Apply

Cliquer sur “Lancer un build”. Pour voir les logs du serveur cliquer sur le statut du job puis sur “Console Output”.


Historique des builds
tendance ^


#1

24 févr. 2022 10:31


[Atom feed des builds](#)


[Atom feed des échecs](#)


[Back to Project](#)

[Status](#)

[Changes](#)

[Console Output](#)

[View as plain text](#)


Sortie de la console

Démarré par l'utilisateur **Océane**

```

[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\jenkins\workspace\Job-pipeline
[Pipeline] {

```


3 - Crée une pipeline avec un fichier Jenkinsfile

Nous allons maintenant voir comment mettre en place une pipeline avec un Jenkinsfile.

Comme nous l'avons vu précédemment, un pipeline Jenkins peut également être déclaré dans **le tableau de bord**. Selon la documentation officielle, il est préférable de déclarer un pipeline avec un **Jenkinsfile**.

L'avantage : versioning possible avec un dépôt (ex: Github)

Étape 1 : création du repository

Nous devons créer un repository sur un dépôt tel que GitHub ou GitLab. Celui-ci doit contenir un fichier nommé "**Jenkinsfile**", il est recommandé de le mettre à la **racine** (non obligatoire).

Nous possédons deux façons de déclarer des pipelines Jenkins. L'exemple ci-dessous correspond à une "**Declarative Pipeline**" et le suivant correspond à une "**Scripted Pipeline**".

```
pipeline {
  agent any

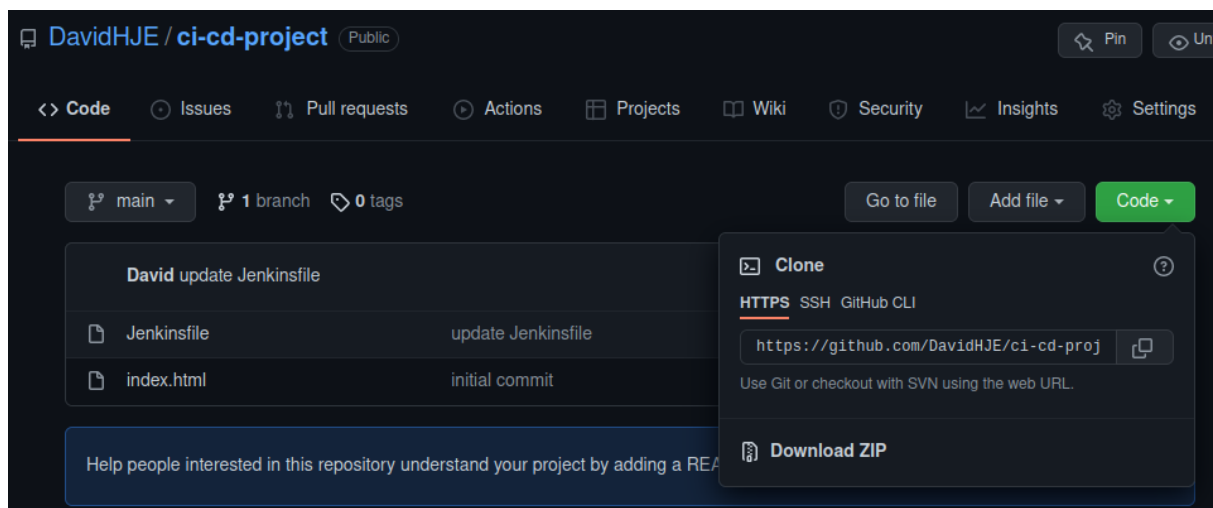
  stages {
    stage('Build') {
      steps {
        echo 'Building..'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing..'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying....'
      }
    }
  }
}
```

```

node {
  stage('Build') {
    echo 'Building....'
  }
  stage('Test') {
    echo 'Testing....'
  }
  stage('Deploy') {
    echo 'Deploying....'
  }
}
}

```

Maintenant nous devons récupérer le **lien vers le dépôt git** pour pouvoir déclarer notre pipeline dans Jenkins.



Retour sur le dashboard Jenkins allez dans **“nouveau item”** et puis sélectionnés **“pipeline”**.

Nous cochons **“GitHub projet”**, il permet d'ajouter un **onglet dans la vue** du pipeline, ainsi on peut facilement être redirigé vers le projet.

☐ Do not allow the pipeline to resume if the controller restarts

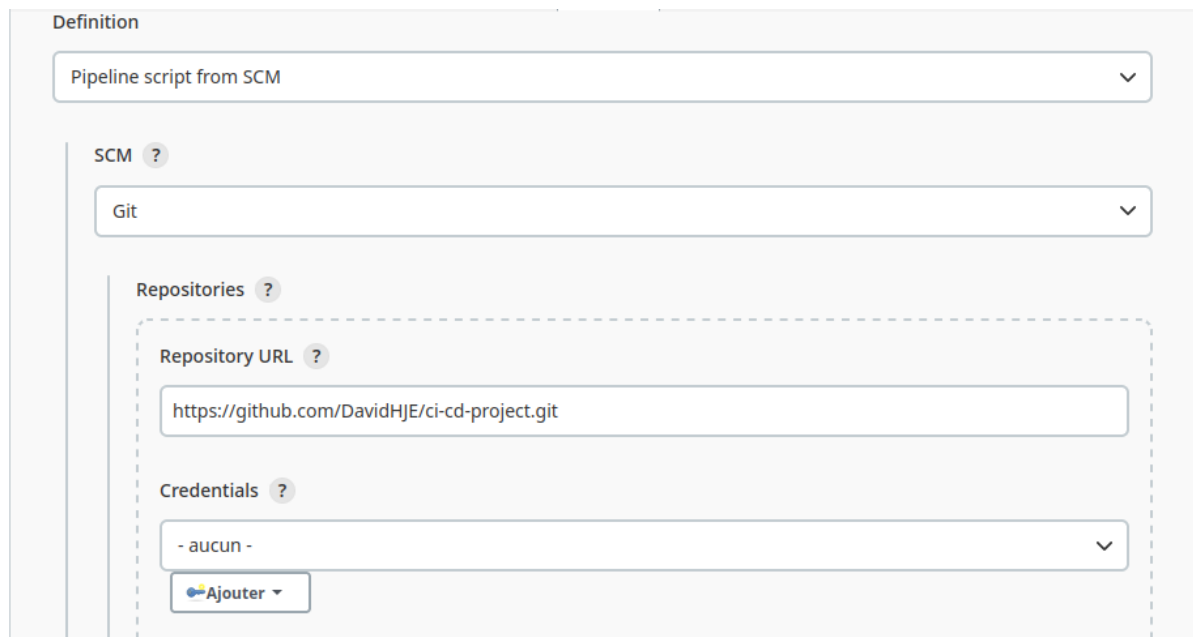
☒ **GitHub project**

Project url ?

☐ Pipeline speed/durability override ?

Maintenant, il suffit d'indiquer à Jenkins où trouver le fichier de configuration du pipeline. Pour cela on sélectionne **"Pipeline script from SCM"**, pour la déclaration de la pipeline, Ensuite **"Git"** > **Repositories URL** > **lien du dépôt git**.

Note : L'authentification pour git est obligatoire uniquement si le dépôt est privé ou que Jenkins doit faire des modifications sur le projet.



Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

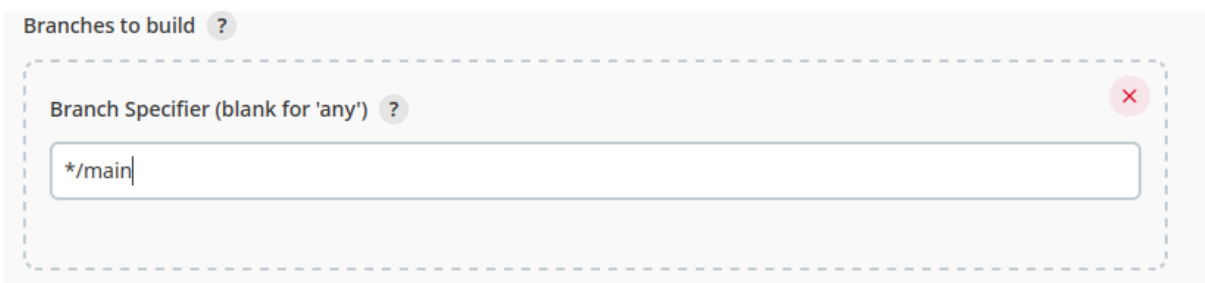
https://github.com/DavidHJE/ci-cd-project.git

Credentials ?

- aucun -

Ajouter

Nous finissons par **choisir la branche** où le jenkinsfile se trouve.

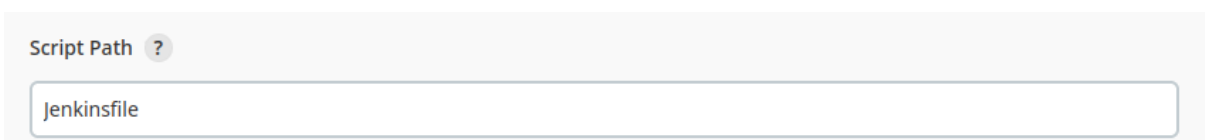


Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Ainsi que son emplacement sur la branche choisie. Dans l'exemple ci-dessous à la racine du projet.



Script Path ?

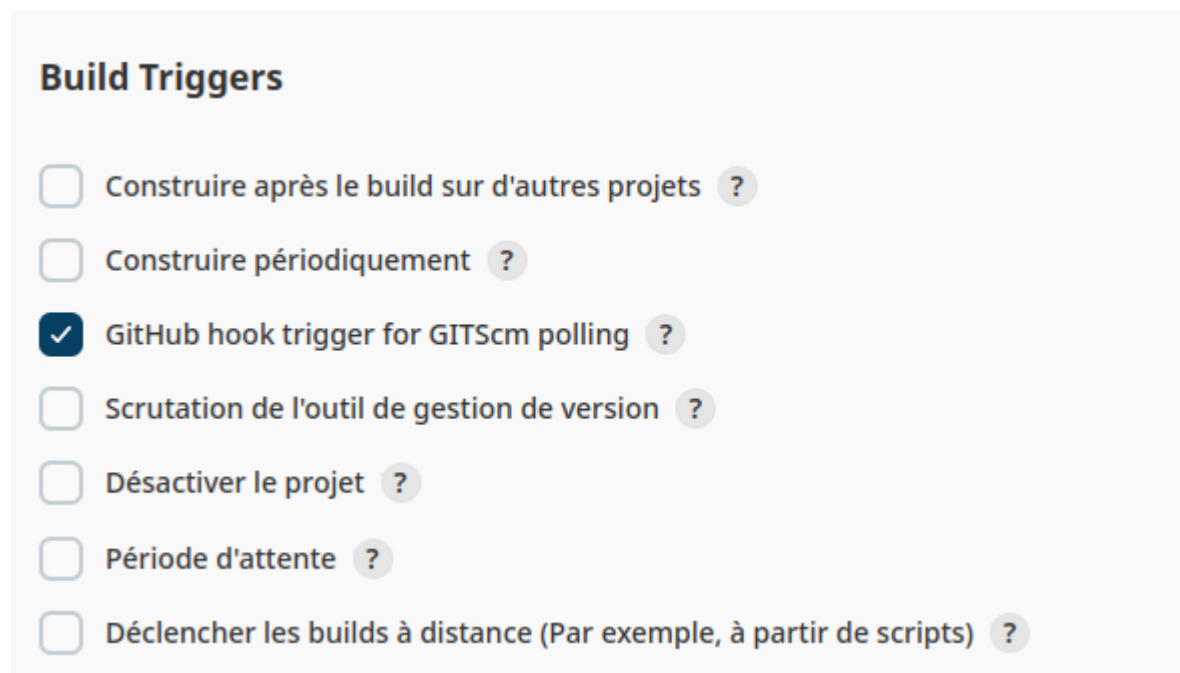
jenkinsfile

5. CAS concret : mise en place d'une pipeline qui est déclenché suite à un changement sur le dépôt GitHub

Étape 1 : Mise en place du projet.

Pour cet exemple, nous allons mettre sur GitHub un projet avec un fichier jenkinsfile et créer une pipeline (**voir l'étape 4.3**).

Lors de la création du pipeline dans Jenkins, nous devons cocher le "**GitHub hook trigger for GITScm polling**". Le pipeline ne se déclenche que lorsqu'il reçoit des informations du GitHub.



Build Triggers

- ☐ Construire après le build sur d'autres projets ?
- ☐ Construire périodiquement ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Scrutation de l'outil de gestion de version ?
- ☐ Désactiver le projet ?
- ☐ Période d'attente ?
- ☐ Déclencher les builds à distance (Par exemple, à partir de scripts) ?

Étape 2 : Rendre Jenkins public

Pour pouvoir utilisé un **webhook**, Jenkins doit être accessible de l'extérieur, c'est-à-dire avoir une **adresse IP public**. Dans notre cas vu que Jenkins est installé en local, nous allons utiliser **ngrok** pour rendre Jenkins public.

Note : Avoir une URL publique permet au collaborateur d'avoir accès au dashboard de Jenkins

Installer ngrok sur notre machine à l'adresse suivante <https://ngrok.com/download>.
Après son installation, il suffit d'utiliser cette commande pour créer le tunnel sécurisé.

```
ngrok http <PORT_JENKINS>
```

Le résultat montre que ngrok à fournir un URL:

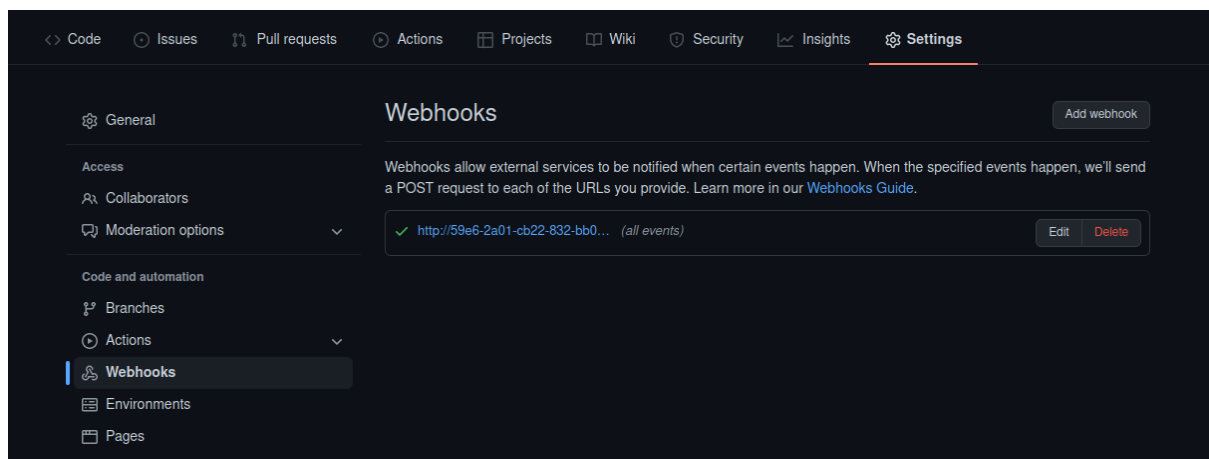
```
ngrok by @inconshreveable

Session Status      online
Account             David Huet (Plan: Free)
Version             2.3.40
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://e7b7-2a01-cb22-832-bb00-ba57-f5ff-e14a-d72.ngrok.io -> http://localhost:9090
Forwarding           https://e7b7-2a01-cb22-832-bb00-ba57-f5ff-e14a-d72.ngrok.io -> http://localhost:9090

Connections          ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00   0.00   0.00   0.00
```

Étape 3 : mise en place du webhook sur Github.

Diriger vous vers votre dépôt github du projet, puis on part dans **Settings > webhooks > add webhook**.



Nous devons copier l'URL public de ngrok est le placer dans le **payload URL**, on ajoute à la fin de l'URL, l'URI **'/github-webhook/'**.

Content type : 'application/json'.

Pour finir, dans la partie événement nous devons choisir quel événement on souhaite que GitHub nous notifie.

Webhooks / Manage webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *

`http://59e6-2a01-cb22-832-bb00-44d5-cd18-381a-2df5.ngrok.io/gitl`

Content type

application/json

Secret

Which events would you like to trigger this webhook?

- ☐ Just the push event.
- ☒ Send me **everything**.
- ☐ Let me select individual events.

☒ Active

We will deliver event details when this hook is triggered.

Update webhook

Delete webhook

Après l'enregistrement, nous avons maintenant un pipeline Jenkins qui se déclenche à chaque changement sur le dépôt GitHub.

Comment créer des tests unitaires avec Cucumber.io

Qu'est-ce qu'un test unitaire ?

Un test unitaire est une procédure qui permet de vérifier le bon fonctionnement d'une partie de code lors du développement d'une application.

Le test unitaire doit s'effectuer indépendamment du reste du programme pour que l'on s'assure que celui-ci réponde aux spécifications fonctionnelles. Celui-ci va exécuter une série de tests qui valide ou non le bon fonctionnement du programme.

Voici le déroulement basic des tests :



- Écrire ou mettez à jour le code.
- Écrire ou mettez à jour des tests pour différents cas pour votre code.
- Exécutez les tests (soit manuellement, soit à l'aide d'un lanceur de tests).
- Voir les résultats des tests. S'il y a des erreurs, corrigez-les et répétez les étapes.

Cucumber :

Cucumber lie et exécuter un fichier qui valide que le logiciel fait ce que ces spécifications disent. Les spécifications se composent de plusieurs exemples, ou scénarios.

Par exemple :

```
Scenario: Breaker guesses a word
  Given the Maker has chosen a word
  When the Breaker makes a guess
  Then the Maker is asked to score
```

Chaque scénario est une liste d'étapes à suivre par Cucumber. Il vérifie que le programme est conforme aux spécifications techniques et génère un rapport indiquant  succès ou  échec pour chaque scénario.

Pour que Cucumber comprenne les scénarios, ils doivent suivre certaines règles de syntaxe de base, appelées Gherkin .

Installer Cucumber dans un projet :

Cucumber peut être installé sur différent environnement :

<https://cucumber.io/docs/guides/10-minute-tutorial/>

Pour notre exemple nous allons installer Cucumber sur un environnement **Node.js**

Installer Cucumer sur Node.js :

```
npm install --save-dev @cucumber/cucumber
```

Dans le fichier **package.json** ajouter le script suivant

```
"scripts": {"test": "cucumber-js"},
```

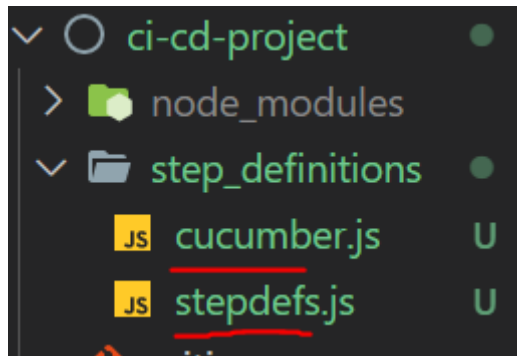
À la racine du projet vous devez crée un dossier appeler **step_definitions**

Dans le dossier, il faudra crée un fichier **cucumber.js** qui contient le script suivant

```
module.exports = {
  default: `--format-options '{"snippetInterface": "synchronous"}'`
}
```


Ensuite, il faudra créer un fichier `stepdefs.js` qui contient le script suivant

```
const assert = require('assert');  
const { Given, When, Then } = require('@cucumber/cucumber');
```



Normalement votre repo doit avoir cette structure

Nous allons tester notre script il suffit d'exécuter la commande suivante:

```
# Run via NPM
```

```
npm test
```

```
# Run standalone
```

```
npx cucumber-js
```

```
D:\documents\ci-cd-project>npm test
```

```
> ci-cd-project@1.0.0 test  
> cucumber-js
```

```
0 scenarios
```

```
0 steps
```

```
0m00.002s (executing steps: 0m00.000s)
```

Share your Cucumber Report with your team at <https://reports.cucumber.io>

Command line option: `--publish`

Environment variable: `CUCUMBER_PUBLISH_ENABLED=true`

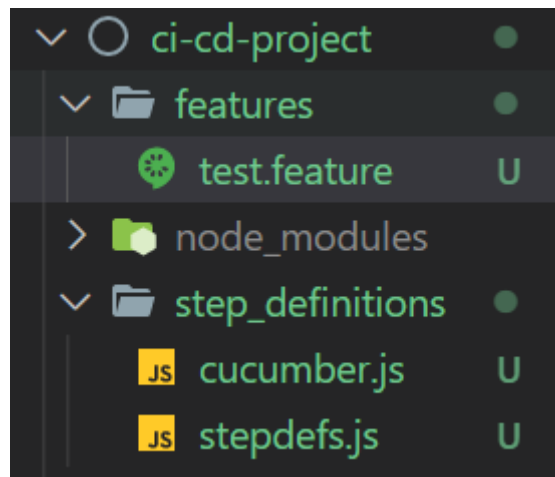
More information at <https://cucumber.io/docs/cucumber/environment-variables/>

To disable this message, add this to your `./cucumber.js`:

```
module.exports = { default: '--publish-quiet' }
```

Création du 1er scénario

Toujours à la racine du dossier, vous devez créer un dossier appeler **features**
Tous les fichiers qui contiennent les scénarios des tests doivent être dans ce dossier
avec extensions du fichier **features/file.feature**



Chaque scénario doit commencer par le mot **Feature** puis le nom du scénario.

La 2ème ligne n'est qu'une description du test Cucumber ne l'exécute pas.

La 3ème ligne, est un [scénario](#) , qui est un exemple concret illustrant comment le logiciel doit se comporter.

Les autres lignes sont les [étapes](#) du scénario.

- **Given** : il s'exécute en 1er avant d'exécuter le test
- **When** : action qui a besoin d'être exécutée
- **And** : autres actions suite du When
- **Then** : résultat attendu

Feature : test

Il s'agit d'une description du test

Scenario: Sunday isn't Friday

Given today is Sunday

When I ask whether it's Friday yet

Then I should be told "Nope"

Lancer le test avec `npm test` voici le résultat

```
1 scenario (1 undefined)
3 steps (3 undefined)
0m00.004s (executing steps: 0m00.000s)
```

Cucumber nous dit que nous avons 1 undefined dans le scénario et 3 undefined dans nos étapes. Il suggère également quelques extraits de code que nous allons copier et utiliser pour définir les étapes.

Vous devez copier les fonctions indiquées dans votre console (voir screenshot) Puis l'ajouter dans le fichier `stepdefs.js`

```
1) Scenario: Sunday isn't Friday # features\test.feature:4
  ? Given today is Sunday
    Undefined. Implement with the following snippet:

    Given('today is Sunday', function () {
      // Write code here that turns the phrase above into concrete actions
      return 'pending';
    });

  ? When I ask whether it's Friday yet
    Undefined. Implement with the following snippet:

    When('I ask whether it\'s Friday yet', function () {
      // Write code here that turns the phrase above into concrete actions
      return 'pending';
    });

  ? Then I should be told "Nope"
    Undefined. Implement with the following snippet:

    Then('I should be told {string}', function (string) {
      // Write code here that turns the phrase above into concrete actions
      return 'pending';
    });
```

5 - Référence

- time to market : Retour client, feedback
- DevOp : Un développeur qui s'occupe du développement d'une app et le déploiement de l'app
- <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>
- <https://www.arcadsoftware.fr/solutions-fr/ci-cd-integration-continue-deploiement-continu/>
- <https://aws.amazon.com/fr/devops/continuous-delivery/#:~:text=La%20livraison%20continue%20est%20une,dans%20un%20environnement%20de%20production.>
- <https://docs.gitlab.com/ee/ci/introduction>
- <https://www.youtube.com/watch?v=ws1qGuFMYlc>
- https://www.splunk.com/fr_fr/data-insider/what-is-ci-cd-pipeline.html
- <https://www.redhat.com/fr/topics/devops/what-is-ci-cd>
- <https://www.lemagit.fr/conseil/Les-avantages-et-les-inconvenients-des-pipelines-CI-CD>
- Tuto Jenkins installation mac
<https://coralogix.com/blog/how-to-install-and-configure-jenkins-on-the-mac-os/>
- <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-20-04-fr>
- Tuto Cucumber java
<https://www.youtube.com/watch?v=b3edllf7oNQ&t=55s>
- <https://www.youtube.com/watch?v=4e9vhX7ZuCW&t=461s>
-

