



# Jenkins - Pipelines

**Mohamed Salah MEDDEB**



# Jenkins Pipeline

---

- Jenkins Pipeline est une **suite de plugins** qui prend en charge l'implémentation et l'intégration de pipelines de livraison continue dans Jenkins.
- Chaque modification apportée à votre logiciel (validée dans le contrôle de code source) passe par un processus complexe avant d'être publiée.
- Ce processus implique la construction du logiciel de manière fiable et reproductible, ainsi que la progression du logiciel construit (appelé «construction») à travers plusieurs étapes de test et de déploiement.
- Pipeline fournit un ensemble extensible d'outils pour modéliser des pipelines de livraison simples à complexes «sous forme de code» via la syntaxe du langage **DSL (Domain-Specific Language)** de Pipeline.

# Avantages du pipeline Jenkins

---

- **Code**: les pipelines sont implémentés dans le code et généralement archivés dans le contrôle de code source, ce qui donne aux équipes la possibilité de modifier, de réviser et d'itérer sur leur pipeline de livraison.
- **Durable**: les pipelines peuvent survivre aux redémarrages planifiés et non planifiés du contrôleur Jenkins.
- **Pausable**: les pipelines peuvent éventuellement s'arrêter et attendre une entrée humaine ou une approbation avant de poursuivre l'exécution du pipeline.
- **Polyvalence**: les pipelines prennent en charge les exigences complexes des CD du monde réel, y compris la capacité de **fork / join**, de **boucler** et d'effectuer des travaux **en parallèle**.
- **Extensible**: le plugin Pipeline prend en charge les extensions personnalisées de son DSL et plusieurs options d'intégration avec d'autres plugins.

# Jenkins Pipeline

---

- La définition d'un pipeline Jenkins est écrite dans un fichier texte (appelé **Jenkinsfile**),
- C'est le fondement du "**Pipeline-as-code**"; traiter le pipeline de CD comme une partie de l'application à contrôler et à réviser comme tout autre code.
- Bien que la syntaxe de définition d'un Pipeline, soit dans l'interface utilisateur Web ou avec un Jenkinsfile soit la même, il est généralement considéré comme une meilleure pratique de définir le Pipeline dans un Jenkinsfile et de l'archiver dans le contrôle de code source.

# Les différents types de pipeline Jenkins

---

- Le pipeline Jenkins est écrit sur la base de deux syntaxes, à savoir:

Le pipeline déclarative	Le pipeline scripté
<ul style="list-style-type: none"><li>○ est une fonctionnalité relativement nouvelle qui prend en charge le pipeline en tant que concept de code.</li><li>○ rend le code du pipeline plus facile à lire et à écrire.</li><li>○ Ce code est écrit dans un <b>Jenkinsfile</b> qui peut être archivé dans un système de <b>gestion de contrôle de source</b> tel que Git.</li><li>○ défini dans un bloc étiqueté «<b>pipeline</b>»</li></ul>	<ul style="list-style-type: none"><li>○ est une manière traditionnelle d'écrire le code.</li><li>○ Dans ce pipeline, le Jenkinsfile est <b>écrit sur l'instance de l'interface utilisateur Jenkins</b>.</li><li>○ Bien que ces deux pipelines soient basés sur le groovy DSL, le pipeline scripté utilise des syntaxes groovy plus strictes car il s'agissait du premier pipeline à être construit sur des bases groovy.</li><li>○ défini dans un bloc étiqueté «<b>node</b>»</li></ul>

- Le pipeline déclaratif a été introduit pour offrir une syntaxe Groovy plus simple et plus d'options.

# Concepts de pipeline

## Pipeline

- Il s'agit d'un bloc défini par l'utilisateur qui contient tous les processus tels que la construction, le test, le déploiement, etc.
- C'est une collection de toutes les étapes d'un Jenkinsfile.
- Toutes les étapes et étapes sont définies dans ce bloc.
- C'est le bloc clé pour une syntaxe de pipeline déclarative.

```
pipeline {  
  
}
```

```
// Declarative //  
pipeline {  
    agent any  
    stages {  
        stage('Build') {  
            steps {  
                sh 'make'  
            }  
        }  
        stage('Test'){  
            steps {  
                sh 'make check'  
                junit 'reports/**/*.xml'  
            }  
        }  
        stage('Deploy') {  
            steps {  
                sh 'make publish'  
            }  
        }  
    }  
}
```

# Concepts de pipeline

---

## Node

- Un ou plusieurs blocs de node effectuent le travail de base dans tout le Pipeline
- Planifie les étapes contenues dans le bloc à exécuter en ajoutant un élément à la file d'attente Jenkins. Dès qu'un exécuteur est libre sur un nœud, les étapes s'exécutent.
- Crée un espace de travail où le travail peut être effectué sur les fichiers extraits du contrôle de code source.

```
node {  
  
}
```

```
// Script //  
node {  
    stage('Build') {  
        sh 'make'  
    }  
    stage('Test') {  
        sh 'make check'  
        junit 'reports/**/*.xml'  
    }  
    stage('Deploy') {  
        sh 'make publish'  
    }  
}
```

# Concepts de pipeline

---

## Agent

- Un agent est une directive qui peut exécuter plusieurs builds avec une seule instance de Jenkins.
- Cette fonctionnalité permet de distribuer la charge de travail à différents agents et d'exécuter plusieurs projets dans une seule instance Jenkins.
- Il demande à Jenkins d' allouer un exécuteur pour les builds.
- Un seul agent peut être spécifié pour un pipeline entier ou des agents spécifiques peuvent être attribués pour exécuter chaque étape d'un pipeline.

Any	None	Label	Docker
Exécute le pipeline/l'étape sur n'importe quel agent disponible.	Pas d'agent global pour l'ensemble du pipeline, chaque étape doit spécifier son propre agent.	Exécute le pipeline/l'étape sur l'agent étiqueté.	Ce paramètre utilise le conteneur docker comme environnement d'exécution pour le pipeline ou une étape spécifique.



# Concepts de pipeline

---

## Stages

- Ce bloc contient tous les travaux à effectuer.
- Le travail est spécifié sous forme d'étapes.
- Il peut y avoir plus d'une étape dans cette directive.
- Chaque étape exécute une tâche spécifique.

```
pipeline {  
    agent any  
    stages {  
        stage('Build') {  
            ...  
        }  
        stage('Test') {  
            ...  
        }  
        stage('QA') {  
            ...  
        }  
        stage('Deploy') {  
            ...  
        }  
        stage('Monitor') {  
            ...  
        }  
    }  
}
```

# Concepts de pipeline

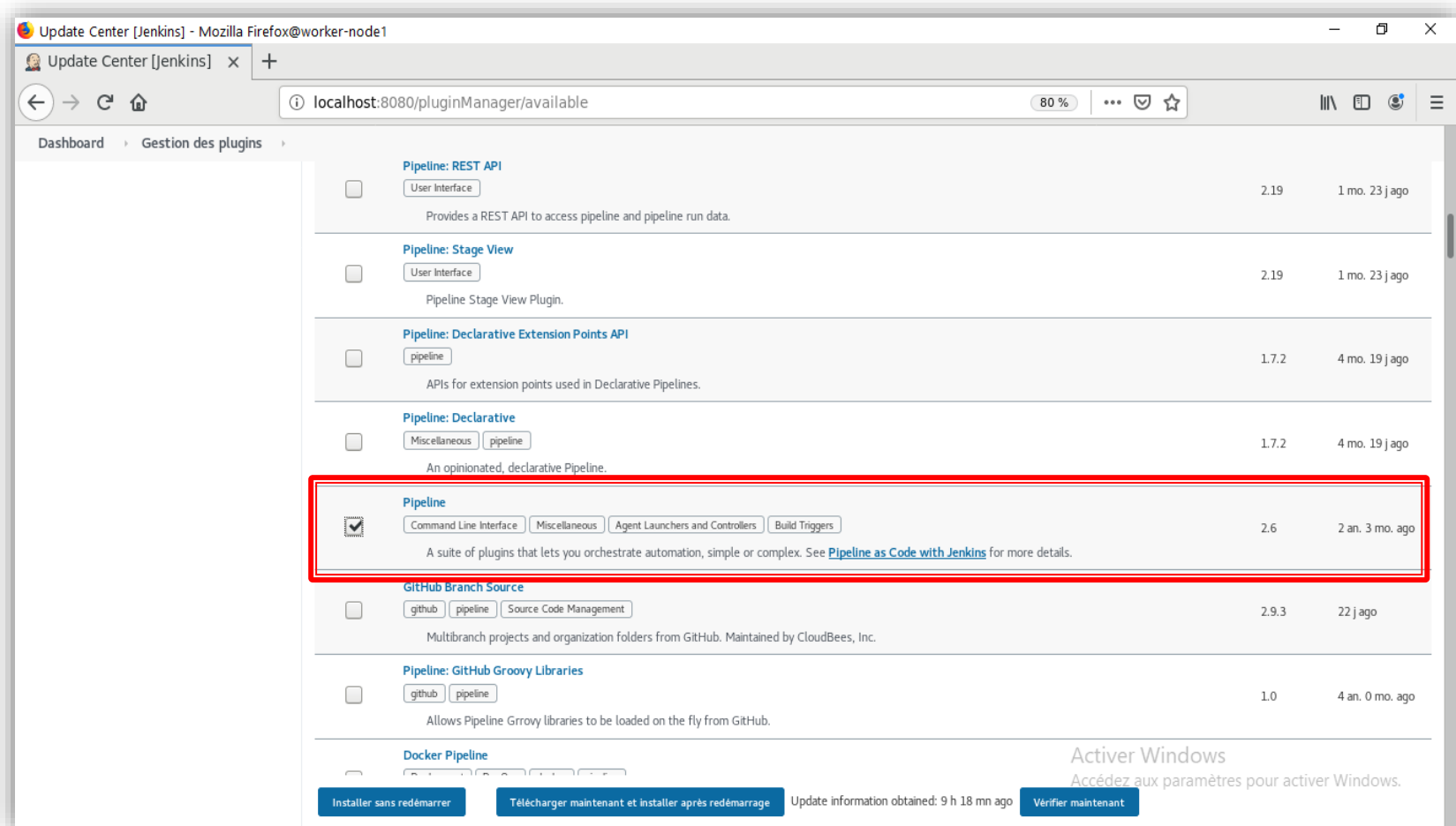
---

- **Steps**
- Une série d'étapes peut être définie dans un bloc d'étape.
- Ces étapes sont exécutées en séquence pour exécuter une étape.
- Il doit y avoir au moins une étape dans une directive d'étapes.

```
pipeline {  
    agent any  
    stages {  
        stage('Build') {  
            steps {  
                echo "Exécuter une étape de construction"  
            }  
        }  
    }  
}
```

# Configuration du Jenkins

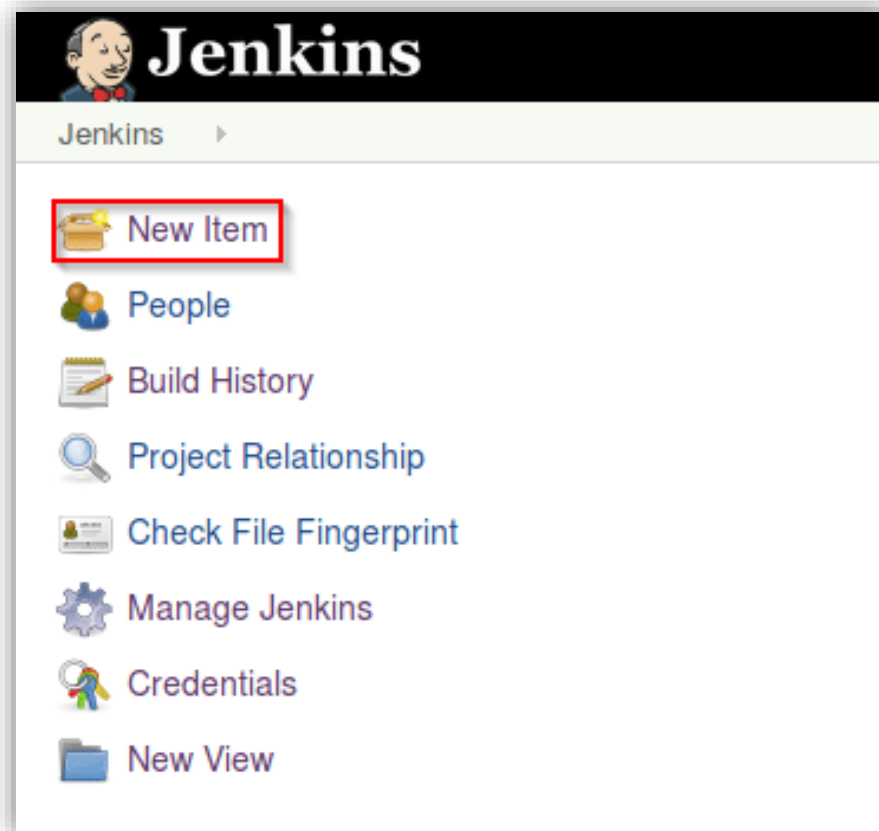
- Il faut installer le plugin « Pipeline » et ses dépendances



# Configuration du Pipeline

---

**Étape 1** : Connectez-vous à Jenkins et sélectionnez «Nouvel Item» dans le tableau de bord.



# Configuration du Pipeline

---


**Étape 2** : Ensuite, entrez un nom pour votre pipeline et sélectionnez le projet «pipeline». Cliquez sur «ok» pour continuer.

Jenkins > All >


### Enter an item name

PIPELINE DEMO


» Required field



**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

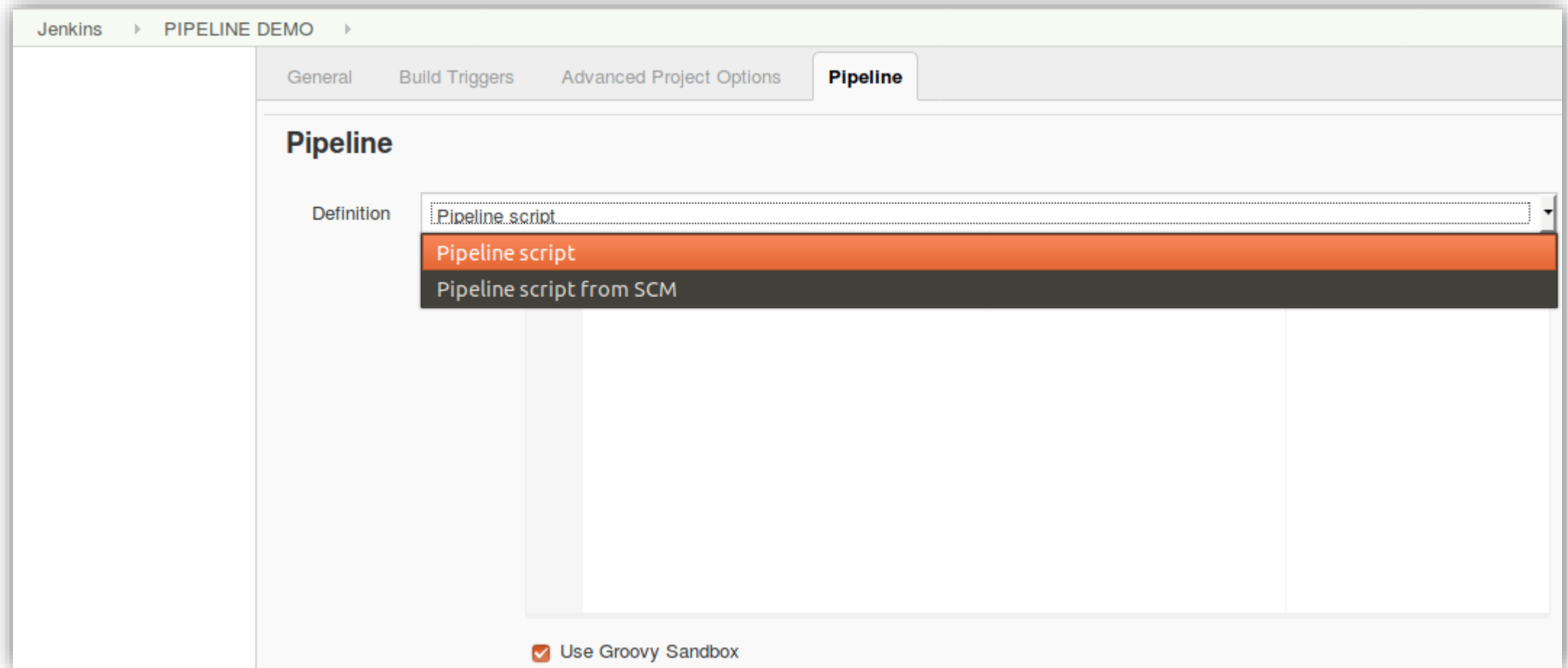


**Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

# Configuration du Pipeline

---

**Étape 3** : Faites défiler vers le bas jusqu'au pipeline et choisissez si vous voulez un pipeline **déclaratif** ou **scripté**.



# Configuration du Pipeline

---

**Étape 4 (1/2)** : Si vous voulez un **pipeline scripté**, choisissez «script pipeline» et commencez à taper votre code.

The screenshot shows the Jenkins configuration page for a project named 'PIPELINE DEMO'. The 'Pipeline' tab is selected, and the 'Definition' dropdown is set to 'Pipeline script'. Below this, there is a text area for the script with a line number '1' and the placeholder text '// TYPE YOUR CODE HERE'. To the right of the text area is a button labeled 'try sample Pipeline...'. At the bottom, there is a checkbox labeled 'Use Groovy Sandbox' which is checked.

Jenkins > PIPELINE DEMO >

General Build Triggers Advanced Project Options **Pipeline**

**Pipeline**

Definition Pipeline script

Script 1 // TYPE YOUR CODE HERE try sample Pipeline...

☒ Use Groovy Sandbox

# Configuration du Pipeline

**Étape 4 (2/2)** : Si vous voulez un **pipeline déclaratif**, sélectionnez «script de pipeline de SCM» et choisissez votre SCM.

The screenshot shows the Jenkins configuration interface for a pipeline. The breadcrumb navigation at the top indicates 'Jenkins > PIPELINE DEMO >'. Below this, there are tabs for 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline', with 'Pipeline' being the active tab. The main section is titled 'Pipeline' and contains the following fields:

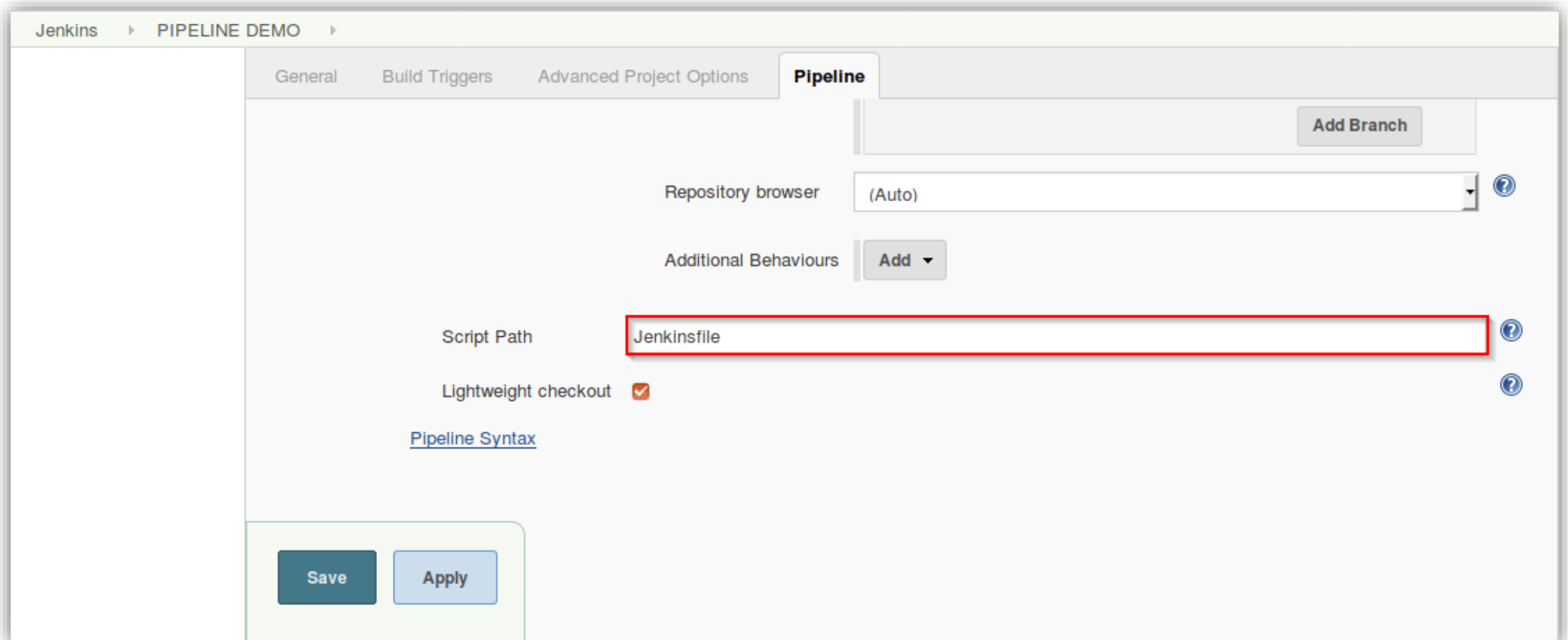
- Definition:** A dropdown menu set to 'Pipeline script from SCM'.
- SCM:** A dropdown menu set to 'Git'.
- Repositories:** A section containing:
  - Repository URL:** A text field with the value 'https://github.com/Zulaikha12/git-test.git'.
  - Credentials:** A dropdown menu set to '- none -' and an 'Add' button.
  - Buttons:** 'Advanced...' and 'Add Repository'.

A red error message is displayed below the Repository URL field: 'Please enter Git repository.' The 'Definition' and 'SCM' fields are highlighted with red rectangles in the original image.



# Configuration du Pipeline

**Étape 5** : Dans le chemin du script se trouve le nom du fichier Jenkins qui sera accessible depuis votre SCM pour s'exécuter. Cliquez enfin sur «Appliquer» et «Enregistrer». Vous avez créé avec succès votre premier pipeline Jenkins.



The screenshot shows the Jenkins configuration interface for a project named 'PIPELINE DEMO'. The 'Pipeline' tab is selected, displaying options for repository browser, additional behaviors, script path, and checkout type. The 'Script Path' field is highlighted with a red border and contains the text 'Jenkinsfile'. The 'Lightweight checkout' checkbox is checked. At the bottom, there are 'Save' and 'Apply' buttons.

Jenkins > PIPELINE DEMO

General Build Triggers Advanced Project Options **Pipeline**

Add Branch

Repository browser (Auto) ?

Additional Behaviours Add ▾

Script Path **Jenkinsfile** ?

Lightweight checkout ☒ ?

[Pipeline Syntax](#)

Save Apply

# Pipeline scripté (exemple)

```
properties([
  parameters([
    gitParameter(branch: '',
      branchFilter: 'origin/(.*)',
      defaultValue: 'master',
      description: '',
      name: 'BRANCH',
      quickFilterEnabled: false,
      selectedValue: 'NONE',
      sortMode: 'NONE',
      tagFilter: '*',
      type: 'PT_BRANCH')
  ])
])

def SERVER_ID="artifactory"
```

```
node {
  stage("Checkout") {
    git branch: "${params.BRANCH}",
      url: 'https://github.com/devops/blog-pipelines.git'
  }
  stage("Build") {
    try {
      withMaven(maven: "Maven363") {
        sh "mvn package"
      }
    } catch (error) {
      currentBuild.result='UNSTABLE'
    }
  }
  stage("Publish artifact") {
    def server = Artifactory.server "${SERVER_ID}"
    def uploadSpec = """{
      "files": [
        { "pattern": "target/blog-pipelines*.jar",
          "target": "libs-snapshot-local/com/devops/pipelines/"
        }
      ]
    }"""
    server.upload(uploadSpec)
  }
}
```

# Pipeline Déclaratif (exemple)

```
properties([
  parameters([
    gitParameter(branch: '',
      branchFilter: 'origin/(.*)',
      defaultValue: 'master',
      description: '',
      name: 'BRANCH',
      quickFilterEnabled: false,
      selectedValue: 'NONE',
      sortMode: 'NONE',
      tagFilter: '*',
      type: 'PT_BRANCH')
  ])
])
```

```
pipeline {
  agent any
  environment { SERVER_ID = 'artifactory' }
  stages {
    stage("Checkout") {
      steps {
        git branch: "${params.BRANCH}",
          url: 'https://github.com/devops/blog-pipelines.git'
      }
    }
    stage("Build") {
      steps {
        warnError("Les tests unitaires ont échoué") {
          withMaven(maven: "Maven363") {
            sh "mvn package"
          }
        }
      }
    }
    stage("Publish artifact") {
      steps {
        rtUpload (
          serverId: "$SERVER_ID",
          spec: '''{
            "files": [
              {
                "pattern": "target/blog-pipelines*.jar",
                "target": "libs-snapshot-local/com/devops/pipelines/"
              }
            ]
          }'''
        )
      }
    }
  }
}
```