

L'intégration continue via Jenkins

Nous allons voir dans ce module comment utiliser Jenkins et les outils qui sont fournis avec ce dernier. Mais avant toute chose, nous devons comprendre pourquoi un tel outil a été créé et pourquoi la demande est si forte ces dernières années en entreprise. Dans ce cours, nous allons voir une brève définition de ce qu'est l'intégration continue (et plus précisément le mouvement DevOps), l'installation et la configuration de Jenkins, les principes de bases de Jenkins mais aussi comment intégrer cette solution dans un environnement informatique.

1. L'intégration continue et le mouvement DevOps

Si on se réfère à Wikipédia, voici la définition du mouvement DevOps :

"Le DevOps est un mouvement en ingénierie informatique et une pratique technique visant à l'unification du développement logiciel et de l'administration des infrastructures informatiques, notamment l'administration système".

Dans d'autres termes, il s'agit de pouvoir suivre toutes les étapes d'un projet de création de logiciel que ce soit le développement, l'intégration, les tests, mais aussi de pouvoir automatiser certaines tâches dans ce processus afin de fiabiliser le rendu et d'augmenter la fréquence auxquels les livrables sont déployés. Une des parties de ce processus est l'intégration continue.

L'intégration continue est un ensemble de processus qui permet de vérifier à chaque instant du développement que les modifications faites à un code source ne produisent pas de régression (bug, dysfonctionnement d'une fonctionnalité mise en place ...). Ce concept a été mentionné par Grady Booch. Cela permet aux développeurs d'intervenir plus rapidement sur des soucis d'intégration qui n'aurait été vus qu'en fin de développement, voire au moment du déploiement en production de l'application. L'intégration continue permet aussi l'exécution automatique des différents tests qu'ils soient unitaires ou fonctionnels. On peut aussi avoir un état de l'évolution du développement du logiciel, comme par exemple, le bon respect des normes de codage définies par le TechLead.

L'intégration continue possède plusieurs contraintes :

- Il faut que le code source partagé par le biais d'un système de gestion de versions (GIT, SVN, etc...)
- Les développeurs doivent intégrer le travail réalisé le plus souvent possible.
- Des tests soient développés afin de contrôler la conformité du code source.

L'utilisation d'un outil d'intégration continue est ensuite nécessaire. C'est maintenant qu'intervient Jenkins. Jenkins est l'un des outils les plus représentés en entreprise, du fait de sa fiabilité et sa robustesse. Il existe bien sûr d'autres outils comme TravisCI, Azure DevOps, etc... Nous allons voir plus en détails Jenkins dans le paragraphe qui suit.

2. Jenkins, qu'est-ce c'est ?

Comme nous l'avons expliqué précédemment, Jenkins est un outil d'intégration continue. C'est un logiciel OpenSource et un fork (une branche) de Hudson. Hudson est un logiciel d'intégration continue qui a été créé par Kohsuke Kawaguchi lorsqu'il travaillait pour Sun Microsystems. Ce logiciel a été développé entièrement en Java. À la suite d'un différend avec les responsables d'Oracle qui avait racheté Hudson en 2011, Kohsuke Kawaguchi décide de créer un fork de Hudson et de créer Jenkins. Jenkins est encore maintenant en 2021 et a encore de beaux jours devant lui malgré son interface un peu vieillotte.

Jenkins fonctionne dans un conteneur de servlets comme Apache Tomcat mais on peut aussi l'utiliser en mode autonome avec son propre serveur web.

3. Installation et configuration de Jenkins

POUR LES PARTIES A VENIR, IL EST CONSEILLE DE CREER UNE MACHINE VIRTUELLE (WINDOWS OU LINUX) AFIN DE CONSERVER UNE MACHINE LOCALE PROPRE.

Sous Windows l'installation se déroule grâce à un assistant d'installation, il suffit de suivre.

Pour Linux (et ce que nous utiliserons pour le reste des parties) :

- Installation du JDK Java
 - `sudo apt update`
 - `sudo apt install openjdk-8-jdk`
- Ajout du dépôt Jenkins Debian
 - `wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add --`
 - `sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/' > /etc/apt/sources.list.d/jenkins.list`
- Installation de Jenkins
 - `sudo apt install ca-certificates`
 - `sudo apt update`
 - `sudo apt install jenkins`

Après l'installation, connectez-vous à l'interface web via le port 8080.

Vous devez tomber sur cette page :

172.22.23.114:8080/login?from=%2F

Démarrage


Débloquer Jenkins

Pour être sûr que Jenkins soit configuré de façon sécurisée par un administrateur, un mot de passe a été généré dans le fichier de logs ([où le trouver](#)) ainsi que dans ce fichier sur le serveur :

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Veuillez copier le mot de passe depuis un des 2 endroits et le coller ci-dessous.

Mot de passe administrateur



Pour trouver ce mot de passe, il suffit d'aller dans les dossiers de Jenkins sur votre disque et de trouver le fichier "*initialAdminPassword*" qui se trouve dans le dossier Secrets. Voici la ligne de commande sous Linux :

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Il suffit ensuite de recopier le mot de passe dans la page web de Jenkins.

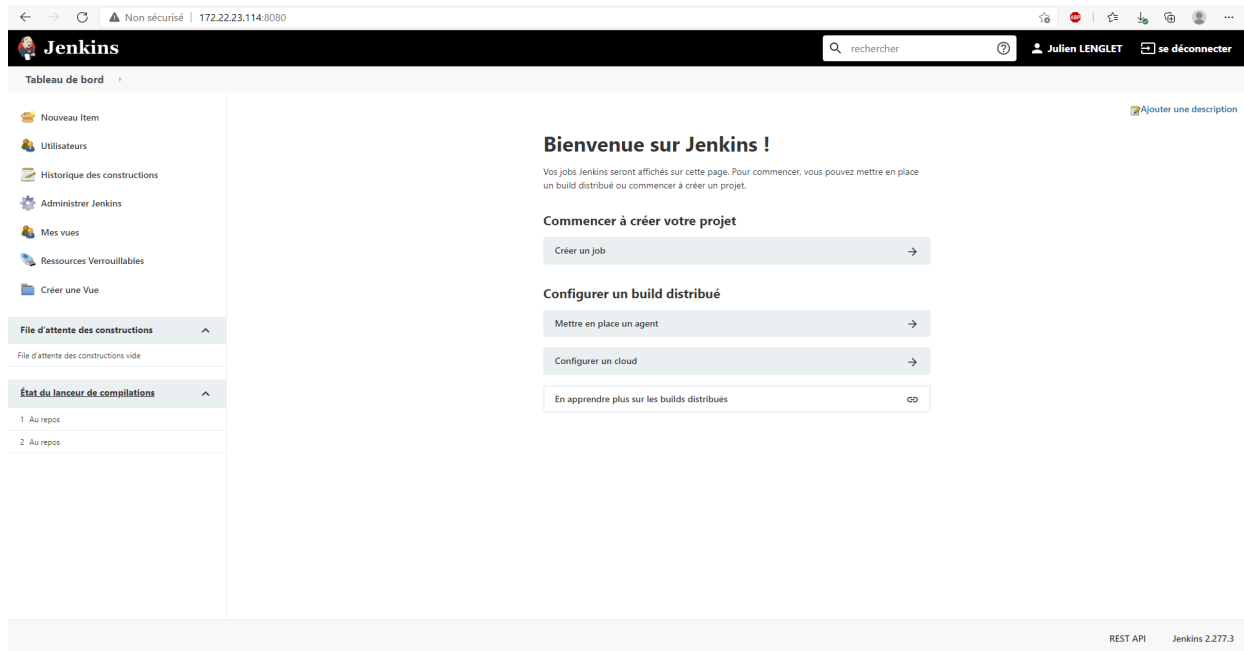
Une nouvelle page doit apparaître avec la possibilité d'installer des plugins pour Jenkins. Il est conseillé et fortement recommandé d'installer les plugins suggérés. Cela permet de gagner du temps dans les futures configurations de Jenkins. Il se peut que l'installation des plugins soit longue à cause des téléchargements. Voici la liste des plugins installés par défaut :

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ Timestampers	Workspace Cleanup	Ant	Gradle
Pipeline	GitHub Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline: Stage View
Git	SSH Build Agents	Matrix Authorization Strategy	PAM Authentication
LDAP	Email Extension	Mailer	

Il est temps maintenant de créer le premier utilisateur de notre outil. Ne perdez pas les informations que vous allez renseigner ici, le premier utilisateur est aussi l'administrateur de votre Jenkins. Cliquer sur "Sauver et continuer". Enfin vérifiez bien ensuite l'URL qui est indiquée sur la page suivante pour qu'elle pointe bien sur le port 8080.

Jenkins est maintenant configuré et prêt à l'emploi.

4. Vue d'ensemble de Jenkins



Sur la gauche de l'écran, nous avons les fonctionnalités principales de Jenkins.

- **Nouveau Item** : C'est ici que nous allons créer des projets (des jobs) Jenkins.
- **Utilisateurs** : On y configure les utilisateurs qui auront accès à votre Jenkins
- **Historique des constructions** : Nous aurons ici la liste des constructions (build) qui ont été effectués à travers des jobs Jenkins.
- **Administrer Jenkins** : Cette partie permet de configurer de nouveaux plugins mais aussi les chemins d'accès vers les SDK et autres qui nous seront utiles.
- **Mes vues** : Cela permet de personnaliser notre tableau de bord. Cette fonction est reliée avec **Créer une vue**.

La partie **Ressources Verrouillables** ne sera pas abordée dans ce document.

Au centre, nous avons pour l'instant une page d'accueil pour la création de nouveaux jobs Jenkins. Mais c'est ici que vous trouverez votre tableau de bord avec diverses informations concernant les jobs (Etat du build, heure du dernier build etc...)

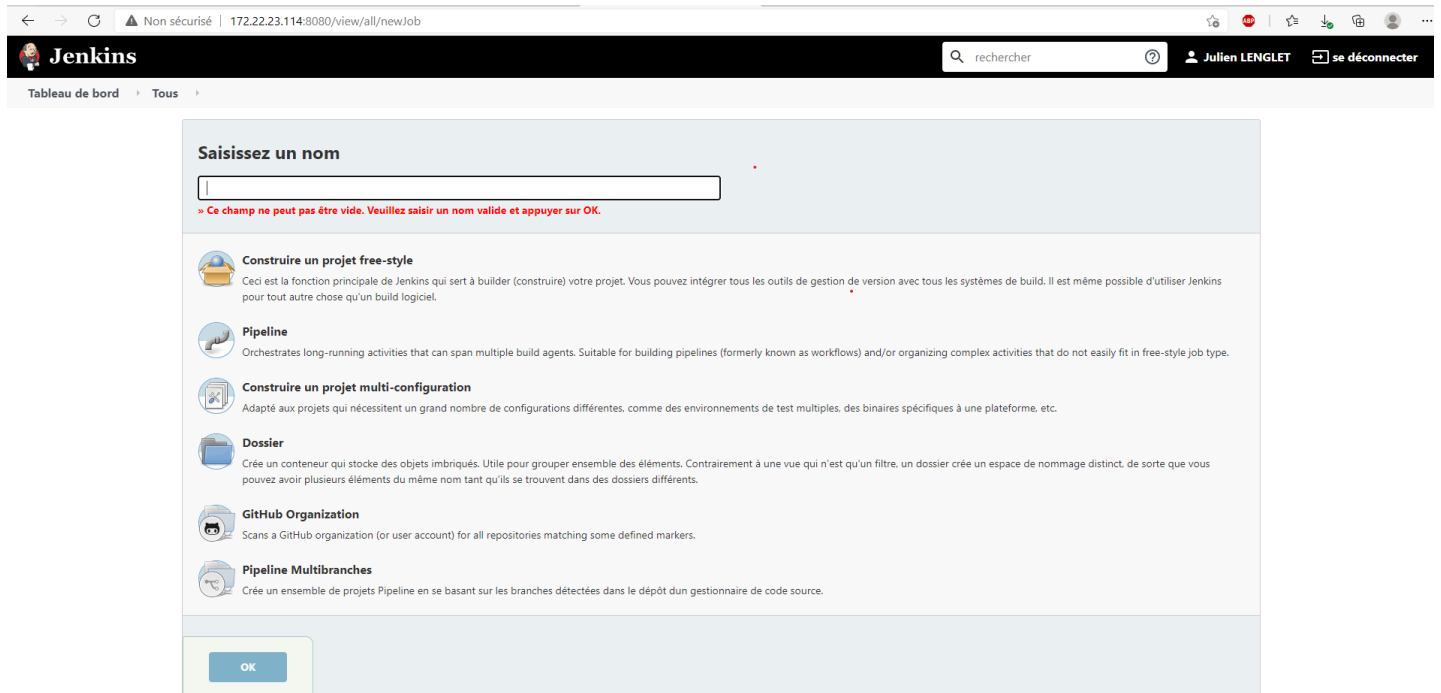
5. Les Jobs Jenkins

Un job Jenkins correspond à un projet. Il contient :

- **Un workspace** : Il s'agit de l'endroit où sont contenus les sources
- **Des tâches** : Il s'agit des scripts qui seront lancés pour effectuer les builds
- **Des builds** : Comme son nom l'indique, nous y trouverons tous les builds qui ont été réalisés via ce job
- **Des logs et un historique** : Pour savoir ce qu'il s'est passé

Il existe différents types de job Jenkins, mais nous allons nous attarder seulement sur deux d'entre eux : Le job **Free-Style** et le job **Pipeline**.

Créons maintenant un nouveau job. Pour cela, nous allons cliquer sur "Nouveau Item". Vous arrivez sur la page suivante :



← → ↻ ⚠ Non sécurisé | 172.22.23.114:8080/view/all/newJob

Jenkins ? Julien LENGLET se déconnecter

Tableau de bord > Tous >

Saisissez un nom

» Ce champ ne peut pas être vide. Veuillez saisir un nom valide et appuyer sur OK.

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
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.

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.

GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

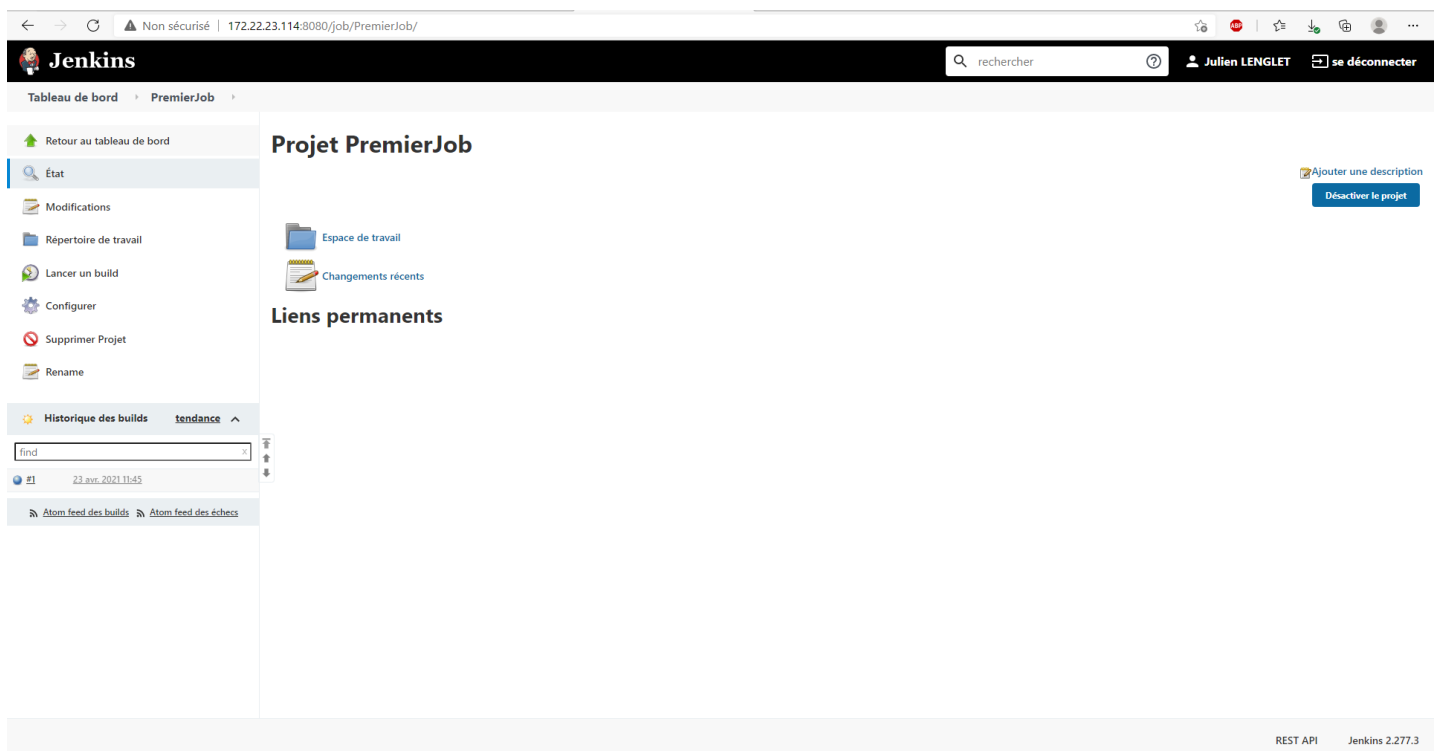
Pipeline Multibranches
Crée un ensemble de projets Pipeline en se basant sur les branches détectées dans le dépôt d'un gestionnaire de code source.

OK

Nous allons saisir tout d'abord un nom pour notre job par exemple *PremierJob*. Puis on va sélectionner "Construire un projet free-style". Puis appuyez sur OK.

Nous arrivons sur une page de configuration de notre job. On peut y trouver plusieurs onglets en haut de la page qui correspondent chacun à une section dans l'onglet Général (ce sont juste des raccourcis). Si vous parcourez un peu les onglets, vous pouvez voir que nous avons énormément d'options à notre disposition. Pour l'instant, nous n'allons rien configurer et laisser tel quel. Appuyez sur le bouton "Sauver" en bas à gauche pour terminer la création du job. On reviendra configurer plus tard.

On a maintenant créé un projet (job) *PremierJob*. Nous avons maintenant quelques possibilités sur notre gauche. Nous allons lancer un build.



← → ↻ ⚠ Non sécurisé | 172.22.23.114:8080/job/PremierJob/

Jenkins ? Julien LENGLET se déconnecter

Tableau de bord > PremierJob >

Retour au tableau de bord

État

Modifications

Répertoire de travail

Lancer un build

Configurer

Supprimer Projet

Rename

Historique des builds **tendance** ^

find

#1 23 avr. 2021 11:45

Atom feed des builds Atom feed des échecs

Projet PremierJob

Ajouter une description

Désactiver le projet

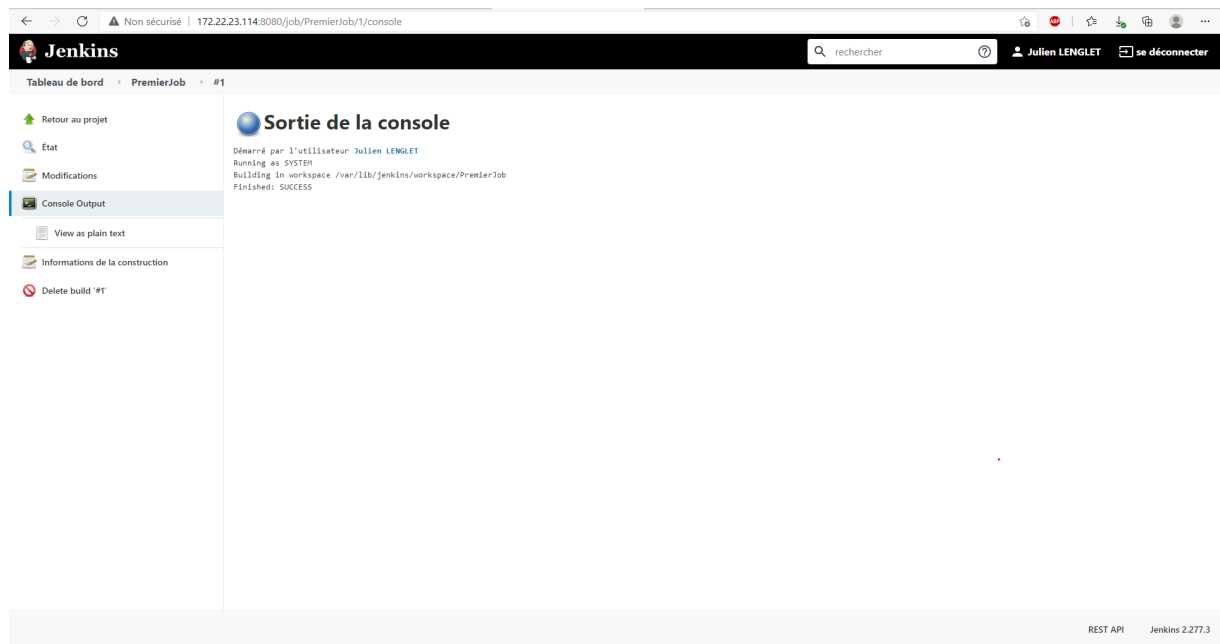
Espace de travail

Changements récents

Liens permanents

Vous pouvez apercevoir qu'un build a bien été ajouté dans l'historique des builds. Regardons plus en détails maintenant le contenu de ce build. On peut voir sur la droite la durée d'exécution du build. Allons maintenant dans la partie "Console Output" qui se trouve sur la gauche.

LENGLET Julien – EPSI ARRAS – 2020-2021



On peut alors voir que notre build a été un succès après avoir lancé le contenu de notre workspace qui se trouve au chemin de ce dernier. Nous allons maintenant ajouter des choses dans notre workspace. Pour cela, nous allons créer un petit script shell qui va être exécuté par notre job Jenkins. Pour ajouter quelque chose dans notre workspace, il faut aller dans le dossier de notre job qui se trouve au chemin suivant :

../jenkins/workspace/PremierJob

Puis nous allons créer un nouveau fichier `test.sh` qui contiendra juste l'affichage d'un HelloWorld. (N'oubliez pas de mettre les droits nécessaires à ce fichier pour qu'il soit utilisable).

```

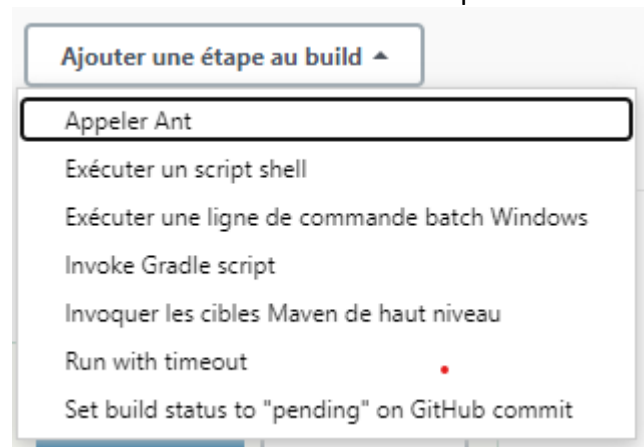
vmjenkins1@vmjenkins1-Virtual-Machine: /var/lib/jenkins/workspace/PremierJob
File Edit View Search Terminal Help
GNU nano 2.9.3 test.sh Modified

#!/bin/bash
echo 'Hello World'

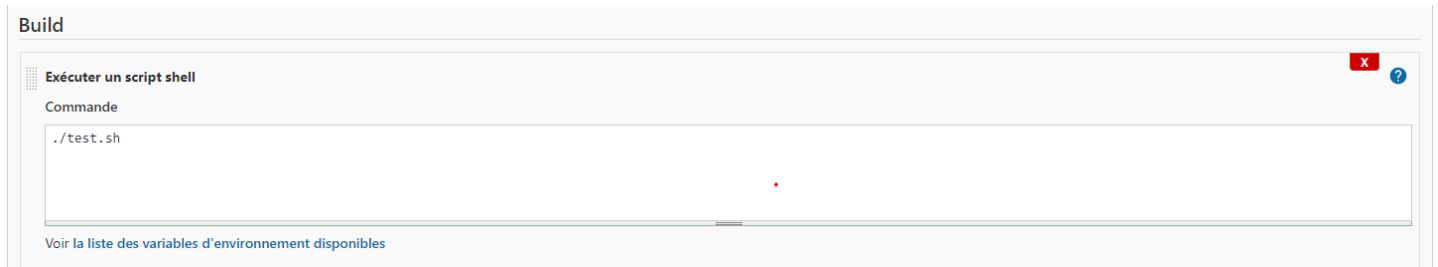
File Name to Write: test.sh
^G Get Help      M-D DOS Format   M-A Append      M-B Backup File
^C Cancel        M-M Mac Format   M-P Prepend     ^T To Files

```

Maintenant, nous allons configurer notre job Jenkins pour qu'il puisse exécuter notre script. Retournons sur la page de notre job, et cliquez sur "Configurer".
 Pour ajouter notre script shell au build, nous allons nous placer dans la partie "Build" en cliquant sur l'onglet qui porte le même nom. Ici, vous trouverez une liste déroulante portant le libellé "Ajouter un étape au build".



Nous avons plusieurs choix pour l'étape que nous voulons ajouter à notre build. Celle qui nous intéresse ici est "Exécuter un script shell" vu qu'il s'agit d'un fichier avec l'extension *sh*. Sous Windows, on utilisera "Exécuter une ligne de commande batch Windows" qui nous permettra d'exécuter des fichiers avec l'extension *bat*.
 Ensuite il suffit juste de rajouter la commande que vous souhaitez exécuter. Il n'y aura pas besoin de définir le chemin du workspace car Jenkins connaît déjà l'emplacement et "pointe" directement dans ce dossier. Après l'ajout de la ligne de commande, appuyez sur "Sauver".



Relançons notre build via "Lancer un build". Un deuxième build apparaît dans notre historique. Ouvrez ce build puis allez dans la section "Console Output". Vous verrez que notre Hello World est bien présent dans le résultat de notre build.

Sortie de la console

```
Démarré par l'utilisateur Julien LENGLET
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/PremierJob
[PremierJob] $ /bin/sh -xe /tmp/jenkins2781129430306508008.sh
+ ./test.sh
Hello World
Finished: SUCCESS
```

Si on retourne sur notre tableau de bord, on peut voir l'état de notre job Jenkins. Une pastille bleue correspond au succès du dernier build que nous avons exécuté. Si le build rencontre un problème, la pastille passerait alors à la couleur rouge.

Nous avons lancé notre premier job Jenkins qui contient un petit script shell. Maintenant, il faut relier du code source, c'est la partie qui nous intéresse.

6. Utiliser Jenkins avec GitHub

La partie suivante demandera l'utilisation d'un compte GitHub ainsi que la création d'un dépôt avec un projet. De plus, la machine qui aura votre serveur Jenkins doit avoir Git d'installer. (Python doit être aussi installé pour notre exemple).

Tout d'abord, nous devons créer un dépôt sur GitHub pour notre projet. Pour cet exemple n'hésitez pas à le mettre en public. Maintenant, nous allons créer un dépôt GIT en local afin de le relier à notre distant. On y ajoutera un fichier *deuxiemejob.py* qui sera exécuté par le build de Jenkins.

Maintenant, votre GitHub doit ressembler à ça si vous avez bien tout fait correctement :

[Code](#)
[Issues](#)
[Pull requests](#)
[Actions](#)
[Projects](#)
[Wiki](#)
[Security](#)
[Insights](#)
[Settings](#)

master
1 branch
0 tags
Go to file
Add file
Code

EPSIArrasSensei
 Commit deuxiemejob.py
 6c8b557 1 hour ago
1 commit

deuxiemejob.py
 Commit deuxiemejob.py
1 hour ago

Help people interested in this repository understand your project by adding a README.
 [Add a README](#)

Il faut maintenant configurer votre serveur Jenkins avec le dépôt GitHub distant afin qu'il puisse récupérer le code source et ensuite exécuter ou compiler le code. Pour cela, nous allons créer un nouveau job Jenkins que nous allons appeler DeuxiemeJob. Puis lors de l'étape de configuration, nous irons dans l'onglet Gestion de code source et dans cette section , sélectionnez "Git".

Tableau de bord
 >
 DeuxiemeJob

General
 Gestion de code source
 Ce qui déclenche le build
 Environnements de Build
 Build
 Actions à la suite du build

☐ Aucune
 ☒ Git

Repositories

Repository URL

Please enter Git repository.

Credentials

- aucun -
 Ajouter

Avancé...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

*/master

Add Branch

Navigateur de la base de code

(Auto)

Sauver
 Apply

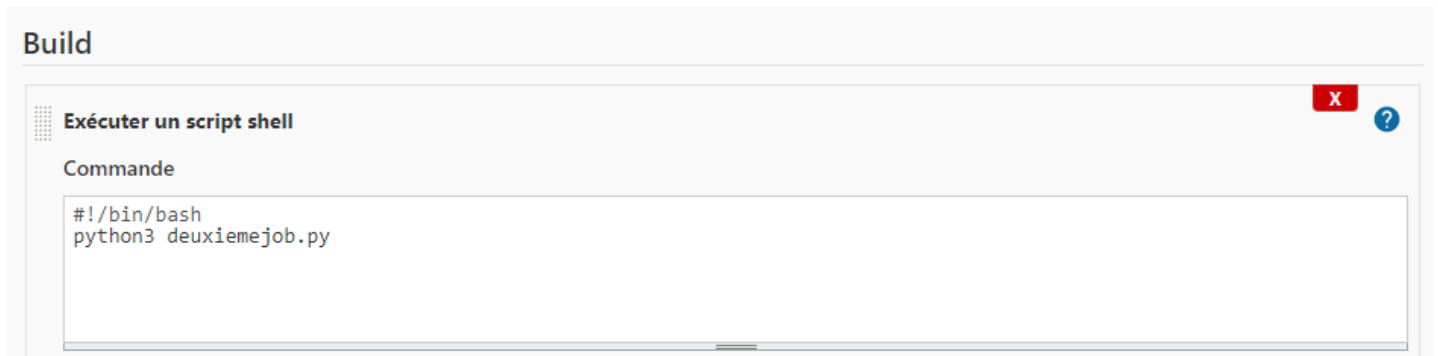
Nous devons ici renseigner le lien de notre dépôt distant sur GitHub (on peut le retrouver sous le bouton "Code" sur la page principale de notre dépôt Github) dans le champ "Repository URL". Si tout se passe bien, aucun message d'erreur ne doit s'afficher.

Note : La partie "Credentials" est utile lorsque vous configurez votre dépôt distant en privé. Il permet de mettre des accès (login / mot de passe) qui ont des droits sur votre dépôt distant. La plupart du temps, on crée des identifiants qui représente notre serveur Jenkins. Dans notre exemple, le projet étant en public, il n'y a pas besoin de configurer tout cela.

Après la configuration de votre dépôt Github, il nous reste une petite chose à faire : demander à Jenkins exécuter le fichier Python lors de l'exécution du build.

Pour cela, nous allons ajouter une étape à notre build qui sera un script shell. Tout à l'heure, nous avions un script shell qui se trouvait dans le workspace directement. Or ici, nous n'avons pas créé de fichier .sh.

Néanmoins, il est possible de directement coder dans l'espace qui nous est alloué par Jenkins.
Ajoutez le code en vous basant sur le screenshot suivant :



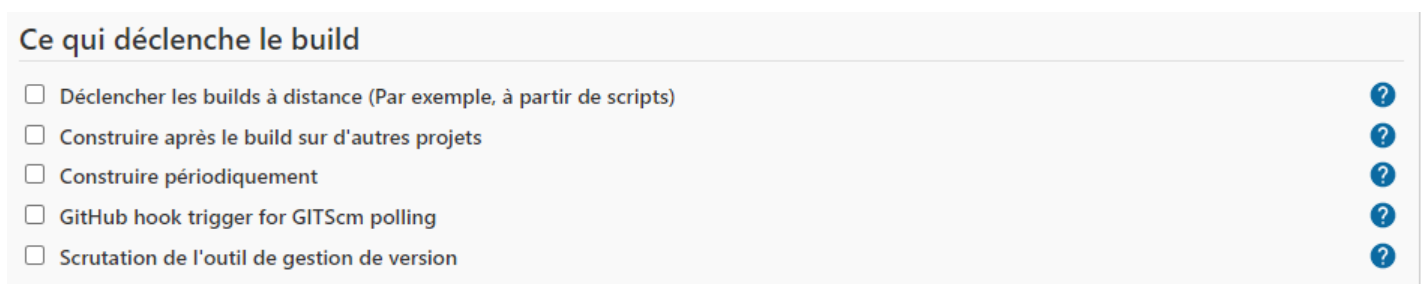
Sauvegardez ensuite le tout ; la configuration de notre job est terminée. Maintenant il suffit de vérifier si tout se passe bien. Lancez un build et regardez si le build passe dans un premier temps, puis dans un deuxième temps, si la pastille est bleue, regardez dans "Console Output" si le code que vous aviez mis dans votre fichier Python a bien été exécuté (Dans notre exemple, notre fichier Python contient du code pour afficher la date du système sur lequel Jenkins est installé).

Sortie de la console

```
Démarré par l'utilisateur Julien LENGLET
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/DeuxiemeJob
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/DeuxiemeJob/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/EPSIArrasSensei/TestJenkins.git # timeout=10
Fetching upstream changes from https://github.com/EPSIArrasSensei/TestJenkins.git
> git --version # timeout=10
> git --version # 'git version 2.17.1'
> git fetch --tags --progress -- https://github.com/EPSIArrasSensei/TestJenkins.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 6c8b5576b5fd7ef647ff43a888dfdfb35df007e (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 6c8b5576b5fd7ef647ff43a888dfdfb35df007e # timeout=10
Commit message: "Commit deuxiemejob.py"
First time build. Skipping changelog.
[DeuxiemeJob] $ /bin/bash /tmp/jenkins4514546214804064753.sh
2021-04-23 15:36:30.265784
Finished: SUCCESS
```

7. L'automatisation des builds

Pour l'instant, nous avons lancer manuellement chaque build pour les différents jobs que nous avons créés. L'intérêt de l'intégration continue est de ne pas à avoir à faire cela et que tout soit automatisé. Pour cela, nous allons retourner dans la configuration de notre job *DeuxiemeJob* et nous allons faire en sorte qu'un build soit lancé toutes les cinq minutes.
Plaçons-nous dans l'onglet "Ce qui déclenche le build" et regardons ce que nous avons :



Nous avons plusieurs options qui s'offrent à nous. Celle qui nous intéresse ici et de pouvoir "construire périodiquement". Lorsque vous cochez cette case, une fenêtre de texte apparaît avec le libellé Planning. C'est ici que nous allons pouvoir indiquer quand un build doit être déclencher. Pour cela, Jenkins utilise une expression CRON. Il s'agit d'une chaîne de caractères qui se compose de cinq à six champs qui sont séparés d'espaces. Cette chaîne représente une durée et est construite selon le schéma suivant :

Field	Required	Allowed values	Allowed special characters	Remarks
Minutes	Yes	0–59	* , -	
Hours	Yes	0–23	* , -	
Day of month	Yes	1–31	* , - ? L W	? L W only in some implementations
Month	Yes	1–12 or JAN–DEC	* , -	
Day of week	Yes	0–6 or SUN–SAT	* , - ? L #	? L # only in some implementations
Year	No	1970–2099	* , -	This field is not supported in standard/default implementations.

Source : [cron - Wikipedia](https://en.wikipedia.org/wiki/Cron)

Les champs que vous ne souhaitez pas renseigner seront indiqué par le symbole *, la récurrence sera indiqué par le symbole /. Nous souhaitons nous faire en sorte que notre build soit exécuté toutes les cinq minutes, l'expression sera la suivante : `* / 5 * * * *`

Ajoutons cela dans notre fenêtre de Planning :

☒ Construire périodiquement

Planning

`* / 5 * * * *`

⚠ Etaler la charge de façon régulière en utilisant `'H / 5 * * * *'` plutôt que `'* / 5 * * * *'`

Aurait été lancé à vendredi 23 avril 2021 16 h 00 CEST; prochaine exécution à vendredi 23 avril 2021 16 h 05 CEST.

Sauvegardez et attendez 5 minutes pour que le build se lance automatiquement.

#3	23 avr. 2021 16:10
#2	23 avr. 2021 16:05
#1	23 avr. 2021 15:36

On peut voir ici que les builds ont bien été espacés de cinq minutes à chaque fois après la mise en place du planning.

Il existe d'autres possibilités pour pouvoir automatiser des builds comme le fait de capter une modification sur le dépôt GitHub distant et build dès l'instant où une modification a été effectuée. Cependant, ici, nous utilisons une méthode locale et il faudrait une adresse ip publique afin de pouvoir communiquer avec notre machine contenant Jenkins.

8. Les pipelines

Les pipelines sont des solutions d'automatisation proposés par Jenkins qui permettent de créer des scripts plus ou

moins complexes pour réaliser des opérations d'intégration continue. Ils peuvent être créés grâce à un langage de script qui est le Groovy ou de manière déclarative qui est une approche plus simple et plus claire pour la réalisation de pipelines, c'est cette manière de faire qui va nous intéresser dans ce document.


Les pipelines sont aussi une autre manière de créer des jobs via Jenkins. En effet, ils permettent d'avoir de la complexité dans l'exécution de build, de déploiement etc...

Nous allons voir maintenant comment créer notre premier pipeline simple. Pour cela, nous allons ajouter un nouvel item depuis notre tableau de bord Jenkins que nous nommerons "TestPipeline" en choisissant bien sûr Pipeline lors de la création de l'item.


Saisissez un nom

TestPipeline


» 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


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.


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.


GitHub Organization

Scans a GitHub organization (or user account) for all repositories matching some defined markers.


Pipeline Multibranches

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

L'interface suivante ressemble beaucoup à celle qu'on a pu voir avec les jobs mais certains onglets manquent à l'appel. Nous n'allons pas nous attarder sur ces onglets, et nous allons directement aller dans la section Pipeline. Et nous arrivons sur une fenêtre intitulée "Pipeline Script". C'est ici que nous allons définir notre script (de façon "Déclarative"). On appelle aussi le fichier qui va être généré par cette section, un *JenkinsFile*. Nous allons voir un exemple que nous est fourni par Jenkins dans la liste déroulante qui se trouve sur la droite. Pour cela, choisissez "Hello World" et voici ce qu'il devrait y avoir à l'écran.

Pipeline

Definition

Pipeline script

Script

```

1 pipeline {
2   agent any
3
4   stages {
5     stage('Hello') {
6       steps {
7         echo 'Hello World'
8       }
9     }
10  }
11 }
12

```

?

Hello World

pipeline

Un *JenkinsFile* qui est défini de manière déclarative commence toujours par le mot-clé "pipeline". Toutes les instructions suivantes sont contenues dans deux accolades qui contiendra tout le script un peu à la manière d'une fonction main en langage C. Ce block est obligatoire.

```
1 pipeline {
2
3 }
```

agent

Ce mot-clé permet de définir où le job Jenkins doit être lancé. Il peut être défini au niveau du pipeline ou au niveau des stages (on verra la notion de stage ensuite). Il est nécessaire de définir un agent. Un agent peut être défini de plusieurs manières, ici on laissera la ligne "agent any". Le mot "any" permet au job d'utiliser n'importe quel agent pour être lancé. Notez qu'on peut forcer quel agent nous utilisons et nous pouvons aussi utiliser Docker à cet endroit.

```
1 pipeline {
2     agent any
3 }
```

stages

Le block "stages" constitue les différents blocks "stage" qui vont être exécutés. Il faut au moins un block "stage" dans le block "stages"

```
1 pipeline {
2     agent any
3     stages {
4         ...
5     }
6 }
```

stage

Ce sont ces blocks qui contiennent l'exécution de chaque étape. Un block "stage" est obligatoirement défini dans un block "stages" et il doit y avoir au moins un block. Il est aussi obligatoire de nommer chaque block "stage" car ce nom sera affiché dans une fenêtre du tableau de bord de notre pipeline. Ici par exemple, on ajoute un block "stage" qui sera nommé "Etape Build".

steps

Le block "steps" est obligatoire dans un block stage. Selon le système d'exploitation de la machine Agent, on pourra utiliser ici des lignes de commandes shell, bat, etc... Par exemple, ici on va faire un echo d'une chaîne de caractères.

```
1 pipeline {
2     agent any
3     stages {
4         stage('Etape Build') {
5             steps {
6                 echo 'Build en cours'
7             }
8         }
9     }
10 }
```

Maintenant exécutons notre pipeline pour voir le résultat. On peut voir qu'une fenêtre est apparue, celle du "Stage View". Cette fenêtre permet de voir le temps d'exécution de chaque "stage" que nous avons configuré dans notre *JenkinsFile*. En passant la souris au-dessus d'un stage, on peut avoir des logs qui nous permettent de connaître ce qui a été exécuté. Cette fenêtre permet d'identifier très rapidement si un souci a été rencontré et surtout de savoir à quel moment de notre pipeline.

On peut avoir plus d'informations en allant dans le "Console Output" de notre build. On peut apercevoir que

chaque ligne de notre *JenkinsFile* s'y trouve dans une couleur grise légère. On voit aussi que notre echo a bien été réalisé.



Sortie de la console

```
Démarré par l'utilisateur Julien LENGLET
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/TestPipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Etape Build)
[Pipeline] echo
Build en cours
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

L'avantage d'un pipeline est la configuration totale de notre processus. Nous n'avons pas vu ici toutes les configurations possibles, mais sachez qu'il est possible de configurer le chemin d'un sdk, et autres.