|  |
| --- |
| Tutoriel – Utilisation de Jenkins |
| Application principalement écrite en HTML/Javascript. |
| Le but de ce tutoriel est de vous montrer quelques grandes fonctionnalités disponibles avec Jenkins. Il ne présente en aucun cas un guide détaillé de Jenkins.  |
| RdVa© |



Contenu

[Introduction 2](#_Toc391535543)

[Accéder à Jenkins. 3](#_Toc391535544)

[Création d’un Job 4](#_Toc391535545)

[Exécution des tests unitaires 8](#_Toc391535546)

[Analyse de la qualité du code 11](#_Toc391535547)

[Conclusion 16](#_Toc391535548)

# Introduction

**Jenkins** est un outil open source d'intégration continue, fork de l'outil **Hudson** après les différends entre son auteur, Kohsuke Kawaguchi, et Oracle. Écrit en Java, Jenkins fonctionne dans un conteneur de servlets tel qu’Apache Tomcat, ou en mode autonome avec son propre serveur Web embarqué.

Source : Wikipédia.

Ce tutoriel a pour objectif de montrer les grandes lignes de Jenkins.
Jenkins intègre énormément de modules utilisables selon vos projets (exemple : modules pour Android, iOS, .NET,…).

Pour ce tutoriel nous avons décidé d’utiliser, comme langage principale, le JavaScript mais Jenkins peut être utilisé pour bien d’autre langage grâce à des plugins.

Nous allons commencer par vous présenter comment créer un job, puis comment faire exécuter une série de tests unitaires sur du code JavaScript par Jenkins.
Enfin nous mettrons en place un outil d’analyse du code JavaScript.

# Accéder à Jenkins.

Une version de Jenkins est déjà mise en place sur la machine Kubuntu.

Il suffit simplement de se rendre avec Firefox à l’adresse suivante : <http://localhost:8080/>

Vous arriverez alors sur l’interface d’administration.

Jenkins a déjà été configuré. Nous allons donc voir comment intégrer le projet Calculatrice à Jenkins.



#

# Création d’un Job

Un job est l’équivalent d’un projet. Il va donc contenir un workspace (le répertoire où seront stockés les fichiers sources), des tâches et des builds, que nous paramétrons plus tard.

Sur l’interface d’accueil, on peut ajouter un job en passant par le lien « Nouveau Item » disponible dans le menu de gauche.

Vous arrivez donc sur le formulaire de création d’un job.



Tout d’abord, donnez un nom à votre job.

Différentes options se présentent pour le type de projet. A ses débuts, Jenkins était prévu pour des projets Maven, mais suite à la forte demande des utilisateurs, d’autres projets ont été rendus possibles d’intégrer.
Le premier choix : « Construire un projet free-style » permet d’intégré tout type de projet.
C’est ce type de projet que nous allons utiliser.

Une fois le formulaire rempli, vous pouvez cliquer sur « OK ». Vous passez alors à une deuxième configuration.

Pour le moment, nous laissons toutes les valeurs par défaut, à l’exception de « Description » où vous pouvez écrire à quoi correspond votre job. Vous pouvez ensuite cliquer sur le bouton sauver.



Votre premier job est alors créé et vous vous retrouvez sur l’interface de gestion de votre job.

Pour le moment, aucun fichier n’a été relié à votre projet. SI vous vous rendez dans votre workspace par le lien disponible dans le menu de gauche (« Répertoire de travail »), vous serez confronté à cette erreur :



Comme l’erreur l’indique, il faut en premier lieu lancer un premier build qui va initialiser des métadonnées, (et s’il y’a un gestionnaire de versionning, il va alors récupérer pour la première fois les fichiers sur le dépôt configuré)

Pour lancer un build, rien de plus simple, il suffit de cliquer sur « Lancer un build », disponible dans le menu de gauche.

Une fois le build lancé, vous pouvez voir l’historique des builds dans un encadré en dessous du menu de gauche :



La couleur bleue à gauche de la date du build indique que tout s’est bien passé. La couleur rouge annonce elle que le build a échoué.

Au milieu, vous voyez maintenant que le workspace existe. Il est cependant vide.
Nous allons donc mettre les fichiers dans le workspace.

Ouvrez Dolphin en faisant : ALT + F2, et saisissez : Dolphin.
Naviguez ou saisissez dans l’URL l’adresse suivante : /var/lib/jenkins/jobs/
Vous pourrez y voir le dossier de votre job. Entrez dedans puis dans workspace.
C’est dans ce dernier dossier qu’il va falloir mettre les sources du projet.
Allez copier-coller les sources du projet disponibles dans le répertoire suivant : /home/rdeva/source/
*Si vous rencontrez un problème de droit ne vous permettant pas de copier les fichiers dans le repértoire workspace, ouvrez un terminal et rendez-vous dans le répertoire parent de workspace (/var/lib/jenkins/jobs/Calculatrice dans notre cas), et saisissez cette commande :*sudo chmod 777 workspace/ -R
Le mot de passe est : rdeva4of6
Vous pourrez alors copier les fichiers

Quand vous retournez dans votre Firefox, vous pouvez actualiser la page (un rafraichissement a déjà pu se faire le temps de la copie de votre fichier). Vous verrez alors vos fichiers.



# Exécution des tests unitaires

#

Maintenant que les fichiers sont bien en place dans notre job, on va pouvoir ajouter des tâches qui vont nous permettre d’effectuer différentes opérations sur nos fichiers sources.
On commence par s’intéresser aux tests unitaires.
Tout d’abord, une explication sur les tests unitaires. Rendez-vous avec Dolphin dans le workspace et lancer le fichier index.html avec Firefox. Vous voyez que tous les tests passent à l’exception d’un.
Le problème est qu’ici, nous avons une page HTML, or Jenkins veut un XML format JUnit pour pouvoir analyser les résultats des tests.
Nous allons donc utiliser phantomjs qui permet d’exécuter un fichier JavaScript sans passer par un navigateur. Il existe un fichier JavaScript tout prêt qui permet de générer un rapport JUnit XML, en lui donnant une url.
Concrètement, la commande permettant de générer les XML des tests est la suivante :

phantomjs run-jasmine-xml-reporter.js <http://localhost:8080/job/Calculatrice/ws/index.html>

run-jasmine-xml-reporter.js est le script qui va générer l’xml via l’adresse donné en paramètre. Le fichier est déjà dans votre workspace.

Attention, l’adresse est différente si vous n’avez pas appelé le job comme nous.

Cette commande-là, nous allons la mettre dans une tâche qui sera appelé lors du build.

Pour cela, dans l’administration de Jenkins, dans votre job, allez dans « Configurer » dans le menu de gauche.
Dans la partie build, faites : Ajouter une étape au build => Exécuter un script Shell.
On va mettre la commande, comme ci-dessous :



Cette commande, lors du build va donc générer des XML. Ils seront à la racine du workspace, et suivront la convention de nommage pour les fichiers : TEST-\*.xml

Nous allons maintenant lancer une autre tâche mais cette fois-ci à la fin du build :
Pour cela, cliquer sur Ajouter une action après le build => Publier le rapport des résultats des tests JUnit.
On va lui donner un paramètre qui est le nom des fichiers à analyser. L’étoile correspond à n’importe quelle suite de caractères.



Sauvegarde votre changement via le bouton sauver.
Maintenant, effectuer un nouveau build.
Vous devriez voir le build être en rouge. Cliquer dessus.
Voici le résultat que vous avez :



On voit bien qu’un test est en échec. Vous pouvez cliquer dessus (suite5.Add/Sous/Muiltip/Div) et vous aurez alors le résultat de votre test, et pourquoi il a échoué.
Pour résoudre l’erreur, dans le workspace, ouvrez js/CalculatriceJsSPec.js/ et chercher : expect(4).toEqual(calculer("1+1\*2/1")); et remplacer le 4 par 3

Vous pouvez faire un nouveau build et cette fois-ci, la pastille sera en bleue.
En cliquant sur Résultats des tests, vous pourrez voir les détails. Naviguez en cliquant sur root puis suite1 ou suite2 ou suite3….

# Analyse de la qualité du code

#

Maintenant, intéressons-nous à l’analyse de qualité du code source.
Nous avons déjà installé le plugin JsLint pour Jenkins.

Retournez dans la configuration de votre job et ajouter une étape au build et choisissez JsLint

Le premier paramètre correspond à vos fichiers js que vous voulez scanner. Ici, on veut scanner tous les fichiers JavaScript du dossier js, d’où le chemin : js/\*.js

Le deuxième paramètre est le(s) chemin(s) des fichiers que l’on veut exclure. Ici, on exclut les fichiers js de jasmine.

Le troisième est le nom d’un fichier XML qui sera créé et qui contiendra les résultats de l’analyse.



Cette étape ne fait que créer l’XML. Pour l’exploiter, un autre plugin a été installé : CHECKSTYLE. Il permet d’analyser tout type d’XML au bon format. On peut alors créer une tâche (action) après le build. Choisissez : « Publier les résultats de l’analyse Checkstyle » dans « Actions à la suite du build », toujours dans la configuration. Mettez le fichier XML dans le champ que vous avez paramétré au-dessus dans l’autre tâche.



En cliquant sur avancée, vous pouvez définir une valeur qui permet de dire à Jenkins d’échouer le build au-delà d’un certain nombre d’erreur. Allons le paramétrer à 20 :



Vous pouvez maintenant sauver la configuration et lancer un build.

Regarder le résultat :



Cliquer sur XX (95 ici) warnings

Vous obtenez des détails :



Vous pouvez maintenant cliquer sur les fichiers en bas, exemple calculatriceJs.js et vous obtenez un détail précis pour le fichier :



Maintenant, pour avoir l’analyse précise, cliquer sur les liens en bas. Exemple calculatriceJs.js :18 :
NOTE : l’infobulle se déclenche sur le survol de la ligne orange.



Maintenant, nous vous invitons à résoudre le problème pour que le build marche. La meilleure solution serait d’exclure deux fichiers car ils sont dépendants d’autres, ce qui rend l’analyse peu efficace.

# Conclusion

On a pu donc voir comment on peut gérer les régressions de codes avec des plugins comme JSLint pour la qualité du code JavaScript.
Des plugins d’analyse de code sont aussi présents dans d’autres langages comme PHPCPD pour détecter du code dupliqué,…

Vous pourrez trouver une liste exhaustive de plugins à installer sous Jenkins [ici](https://wiki.jenkins-ci.org/display/JENKINS/Plugins).

Pour résumer ; pour mettre un projet sous Jenkins on **crée un job** et on **effectue un premier build** pour créer le workspace (répertoire des fichiers sources).
Ensuite on **configure le build** ou **on crée des tâches** (des actions) à effectuer lors du build. Par exemple, dans le projet Calculatrice nous avons effectué une tâche pour générer un rapport des tests unitaires puis une autre tâche pour générer un rapport sur l’analyse du code JavaScript.
Après cette modification, à chaque build ces tâches seront effectuées.
Si le **build réussit**, toutes les tâches ont été acceptées, **sinon** cela signifie qu’une tâche a échoué et on peut voir d’où vient cette erreur facilement.