TP DEVOPS Chaîne de déploiement (Docker, Jenkins, etc.)


(https://creativecommons.org/licenses/by-nc-sa/4.0/)

Alexis Lahouze (Entreprise Sysnove (https://www.sysnove.fr/)) & Sébastien Rey-Coyrehourcq (IGR UMR IDEES (http://umr-idees.fr/user/s%c3%a9bastien-rey-coyrehourcq/))

Support initialement créé pour la journée DEVOPS Rouen 20/04/2018 (http://normandev.cnrs.fr/index.php/manifestations/evenements/10-journee-devops) du réseau métier BAP E normanDEV (http://normandev.cnrs.fr)

Release :

- v0.1 : Document initial, 20/04/2018

[Introduction] Une chaîne de déploiement automatisé pour les développeurs

Le déroulé du TP suit les mésaventures d'un développeur au sein d'une entreprise en pleine reflexion sur son système d'intégration continue et de déploiement.

L'utilisateur d'une entreprise veut développer sa propre version d'un logiciel open-source en **html/php** disponible sur une forge distante (http://gogs.univ-rouen.fr (http://gogs.univ-rouen.fr)). Pour cela il fork le projet et le clone sur sa machine en local
ETAPE 1, mise en place

Comme notre développeur n'est pas particulièrement compétent en administration système, il doit faire souvent faire appel à l'équipe du même nom afin de déployer et tester en ligne les dernières modifications qu'il a faites sur le code de son application.

Fatigués par les problèmes de compatibilité entre applications sur les serveurs de production, et fatigués aussi par les échanges permanents de codes sources entre les équipes, les responsables décident de mettre en place un nouveau système de déploiement basé sur des containers Docker.

Notre développeur est à présent contraint de construire une *image* capable de servir son application, de la transmettre à un registry privé, pour que les administrateurs système puissent déployer la nouvelle application. Le développeur grogne **!#?!** de devoir ainsi penser en amont l'architecture nécessaire au fonctionnement de son programme… Mais très vite il s'aperçoit que cet effort vaut la peine **ETAPE 2, les dockerfiles**

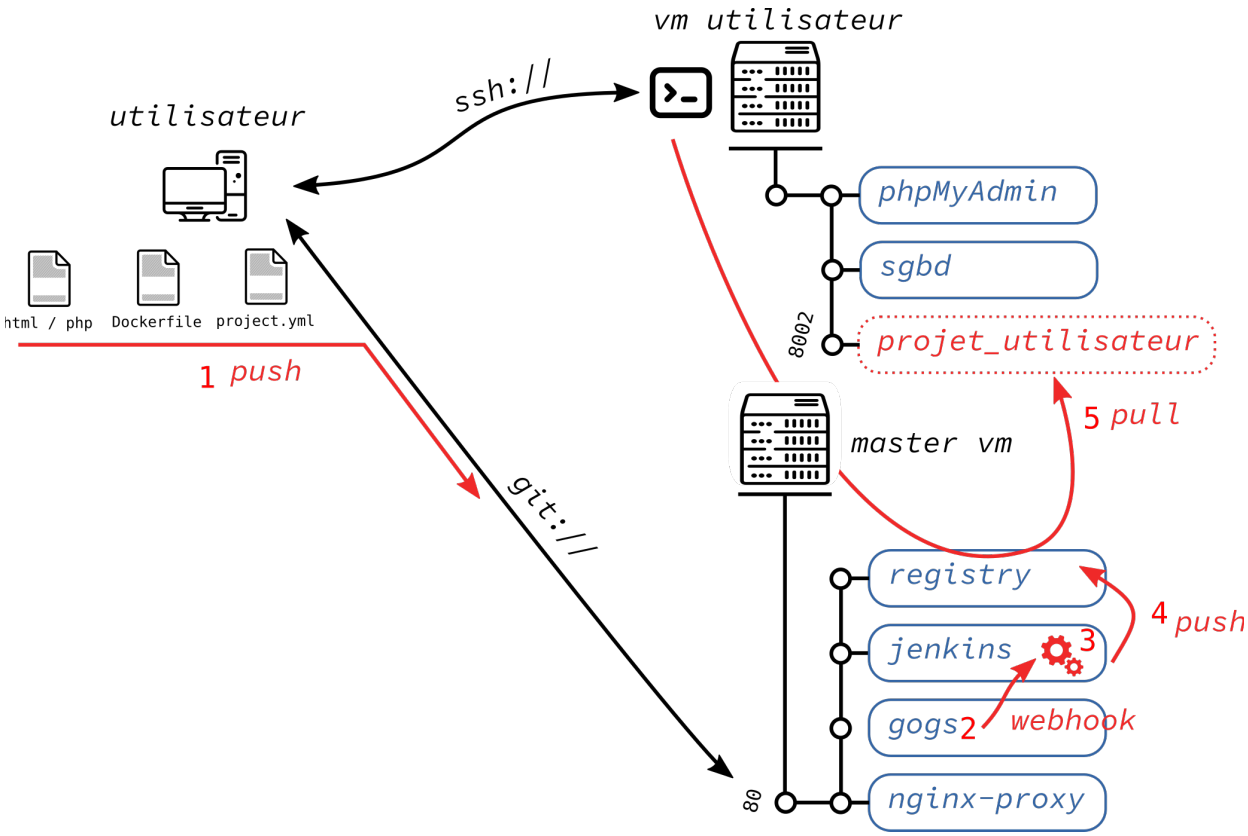
En discutant à table avec un de ses collègues, notre développeur se rend compte qu'il existe un moyen d'automatiser à la fois la construction de l'image mais également son inscription dans la registry privé, et cela à chaque fois qu'il change le code **ETAPE 3, les webhooks et jenkins**

Enfin, notre développeur doit encore faire une étape manuelle en rappatriant la nouvelle image envoyée sur la registry sur la vm utilisateur qui présente le service web. Une façon d'automatiser cette étape est de la déléguer à Jenkins, qui se connecte (en ssh) à chaque fin de build réussi à la VM Utilisateur, et passe les commandes de mise à jour et de déploiement de l'image **ETAPE 4, automatiser le déploiement de l'image via Jenkins et SSH**

Les jours passent, et l'application de notre développeur se complexifie toujours un peu plus, et nécessite plusieurs services (php, base de données, etc.). **ETAPE 5, complexifier le docker-compose**.

Et enfin, il est possible de solidifier l'infrastructure, en mettant en place des checks de santé (health check), un cluster de nœuds Docker (Swarm), et autres outils notamment de visualisation (Portainer). **ETAPE 6, pour aller plus loin**

Une fois dessinée, cette architecture automatisant la plupart des tâches de construction et de déploiement est alors la suivante :



[Info] Démarrage

Pour éviter tout conflits avec les images du TP de la matinée, il est conseillé de faire le grand ménage sur votre VM :

```
docker stop $(docker ps -q -a)
docker container prune -f
```

Les serveurs :

- ssh formation@registry.univ-rouen.fr (mailto:formation@registry.univ-rouen.fr)
- ssh formation@docker-x.univ-rouen.fr (mailto:formation@docker-x.univ-rouen.fr)

Les comptes sont au format “prenomNom” et le mdp NormanDev2018

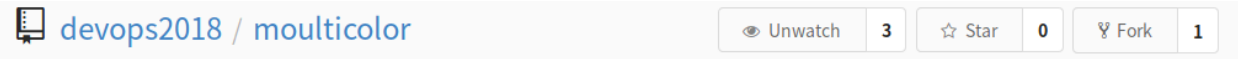
[Etape 1] Mise en place du dépôt local

Accéder à la forge distante gogs.univ-rouen.fr (http://gogs.univ-rouen.fr)

Nous allons d’abord nous identifier sur la webui de la forge logicielle mise en ligne

pour ce TP à l'adresse http://gogs.univ-rouen.fr/ (http://gogs.univ-rouen.fr/)

Pour ne pas partir de rien, nous allons créer un nouveau projet à partir d'un template que nous avons déposé à l'adresse suivante : http://gogs.univ-rouen.fr/devops2018/moulticolor (http://gogs.univ-rouen.fr/devops2018/moulticolor)

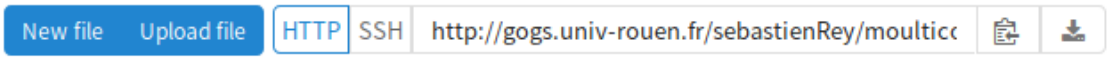


Récupération (Fork & Clone) du projet exemple

Cliquez sur le bouton *fork*, ce qui devrait vous afficher la fenêtre suivante.

Une fois le *fork* créé, vous pouvez également y accéder depuis votre profil ou la homepage.

Nous allons ensuite cloner ce dépôt sur notre machine virtuelle cliente. Pour cela, à la racine de votre home, nous allons créer un répertoire `repository` qui contiendra le clone de ce projet. Vous pouvez copier coller l'url qui apparait sur votre page.



Ce qui donne dans votre terminal la commande suivante, ou vous devez remplacer `prenomNom` par les bonnes informations :

```
git clone http://gogs.univ-rouen.fr/prenomNom/moulticolor.git
```

Avant d'effectuer des opérations sur le dépôt, nous allons renseigner les données d'auteur :

```
1 git config --global user.name "Prénom Nom"
2 git config --global user.email "prenom.nom@univ-rouen.fr"
```

De même, pour ne pas avoir à rentrer à chaque fois notre mot de passe pour envoyer les modifications sur le serveur, nous allons d'abord rentrer ces deux commandes git :

```
1 git config --global credential.helper store
2 git config --global credential.helper 'cache --timeout 14400'
```

> Attention, cette phase de remplacement de texte *prenomNom* ou *prenomnom* revient très souvent dans le code du TP, soyez attentif 😄

Modification du projet exemple

Maintenant nous allons faire des modifications sur ce dépot local, les valider et les renvoyer sur le serveur distant. À la racine du dépot venant d'être créé, nous allons modifier le fichier `README.md` pour y ajouter notre nom et notre prénom.

Pour ce faire, nous utilisons l'éditeur de texte `nano` .

```
cd moulticolor
nano README.md
```

Ajoutez votre nom, prénom, et enregistrez le document en faisant `Ctrl+O` , puis `Ctrl+X` pour quitter.

Normalement, la commande `git status` vous indique que des modifications ont été faites. Pour les valider, nous allons appeler la commande `git commit -a` qui ouvre l'éditeur `nano` .

Entrer le commentaire du *commit*, par exemple "Modification du fichier README, ajout du prénom et du nom du propriétaire". Faites de nouveau un `Ctrl+O`, `Ctrl+X` pour valider le commit.

Mise à jour du projet exemple sur la forge distante

Nous allons maintenant envoyer les modifications sur la forge `gogs.univ-rouen.fr` avec la commande `git push`

Le serveur va vous demander votre login et votre mot de passe, il s'agit de celui de la forge `prenomNom` , puis `NormanDev2018`

Transition

Nous savons à présent comment modifier et mettre à jour le code source sur la forge logicielle distante depuis notre poste en local. Si l'objectif est de déployer cette page web de façon automatique sur un serveur via un container Docker alors la première étape est d'abord de construire une image qui pourra intégrer et présenter ce code. C'est exactement le but du Dockerfile que nous allons mettre en

oeuvre.

[Etape 2] Dockerfile

Notre application web est relativement simple, elle contiendra un fichier `info.php` et un fichier `color.php` que nous allons rajouter dans le dossier `src` du projet `moulticolor`. Pour afficher le contenu des pages en php nous avons besoin d'un serveur web Nginx, de Php 7 avec le module php-FPM (*FastCGI Process Manager*).

Voici un fichier `Dockerfile`, basé sur la distribution linux Alpine (plus légère !), qui permet de construire l'image contenant à la fois et l'environnement adéquat pour servir le code sur le web :

```
FROM gliderlabs/alpine:edge
LABEL maintainer="prenom.nom@univ-rouen.fr"

RUN echo http://nl.alpinelinux.org/alpine/edge/testing >> /etc/apk/repo

RUN addgroup -g 1000 -S www-data \
 && adduser -u 1000 -D -S -G www-data www-data

RUN apk upgrade --update-cache --available
RUN apk add --update && apk add --no-cache -f bash nginx ca-certificate

COPY ./conf/supervisord.conf /etc/supervisord.conf
COPY ./conf/nginx.conf /etc/nginx/nginx.conf
COPY src/ /var/www/html/

RUN mkdir -p /run/nginx && mkdir /tmp/nginx
RUN chown -R www-data:www-data /var/lib/nginx && chown -R www-data:www-

RUN sed -i s/'user = nobody'/'user = www-data'/g /etc/php7/php-fpm.d/ww
RUN sed -i s/'group = nobody'/'group = www-data'/g /etc/php7/php-fpm.d/

EXPOSE 80

CMD ["/usr/bin/supervisord", "-n" , "-c", "/etc/supervisord.conf"]
```

`supervisord` (https://docs.docker.com/config/containers/multi-service_container/) est un outil qui permet de lancer et gérer plusieurs processus (qui eux même peuvent lancer plusieurs processus …) de façon sécurisée/contrôlée dans une seule image, comme un `init` à l'intérieur du conteneur.

Notes aux développeurs Pensez à jeter un oeil au DockerFiles multi-stage (https://docs.docker.com/develop/develop-images/multistage-build/). Cette fonctionalité permet d'enchaîner plusieurs étapes de build dans un `Dockerfile` en ne gardant que le dernier container construit. Par exemple, un premier stage compile les sources d'un logiciel, puis le deuxième stage copie le binaire de ce logiciel du premier stage avant d'être publié. Le container final sera beaucoup plus léger car il n'aura pas toutes les dépendances/paquets liés à la compilation !

Vous pouvez jeter un oeil aux fichiers de configs pour nginx, php et supervisor dans le répertoire `./conf/` du projet.

Vous pouvez ensuite ajouter un fichier `src/index.php` contenant un simple appel à phpinfo :

```
<?php
phpinfo();
?>
```

Vous pouvez ajouter le fichier `src/color.php` avec votre nom, prénom, et vos couleurs préférées (format RGB) :

```
<?php
header("Content-type: image/png");

// Création d'une image de 400x200 pixels
$im = imagecreatetruecolor(400, 200);
$white = imagecolorallocate($im, 255, 255, 255);
$black = imagecolorallocate($im, 0, 0, 0);
$yourcolor = imagecolorallocate($im, 128, 128, 128);
$yourcolor2 = imagecolorallocate($im, 60, 80, 57);

// Le texte
$text = 'DEVOPS 2018';
$text2 = 'SEBASTIEN REY';

// La police
$font = 'consolas.ttf';

// Dessine un double rectangle
imagefilledrectangle($im, 0, 0, 400, 200, $yourcolor);
imagefilledrectangle($im, 10, 10, 390, 190, $yourcolor2);

// Ajout du texte
imagettftext($im, 20, 0, 50, 50, $white, $font, $text);
imagettftext($im, 12, 0, 50, 80, $white, $font, $text2);

// Sauvegarde l'image
imagepng($im);
imagedestroy($im);
?>
```

Vous pouvez ensuite build l’image et la lancer sur votre daemon docker local :

```
docker build -t moulticolor:latest .
```

Puis lancer l’image :

```
docker run -d -p 80:80 --name moulticolor moulticolor:latest
```

Normalement si tout se passe bien vous devriez voir :

- `http://docker-x.univ-rouen.fr/index.php` affiche les informations php du `<?php phpinfo()?>`

- `http://docker-x.univ-rouen.fr/color.php` affiche une image généré dynamiquement par php à partir des instructions dans `color.php`

Pour la suite, nous allons ajouter tous les fichiers et les pousser dans notre *fork*.

Nous pourrions continuer à complexifier notre image, en ajoutant tout un tas de services dans l'image… Toutefois ce n'est pas l'**esprit Docker**, et souvent il est recommandé de découpler au maximum les services. Cela reste à discuter au cas par cas. En effet il peut être tentant d'avoir un container nginx, un container php, et un volume partagé entre les deux containers contenant le code html/php à servir. Cette solution a pourtant des désavantages, et peut limiter l'application dans sa scalabilité.

Pour éviter les problèmes de conflit pour la suite, nous allons supprimer le conteneur précédemment créé :

```
1 docker rm --force moulticolor
```

[Etape 3] Webhooks et Jenkins

Les webhooks (https://en.wikipedia.org/wiki/Webhook) permettent de déclencher une action en fonction d'un événement.

Dans notre cas, l'événement déclencheur correspond à un *commit* sur la forge (http://forge.univ-rouen.fr), et l'action qui en découle est le rebuild de l'image par le service de build continu, dans notre cas Jenkins (http://jenkins.univ-rouen.fr)

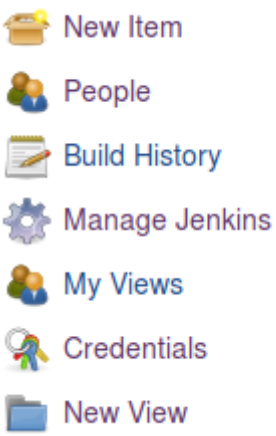
L'action se fait donc en deux étapes, sur lesquels nous revenons plus en détail ensuite :

- le webhook côté forge envoit un signal à Jenkins sur une url dédié,
- Jenkins clone le projet indiqué pour le Job, lit le `Jenkinsfile` et build le `Dockerfile` correspondant

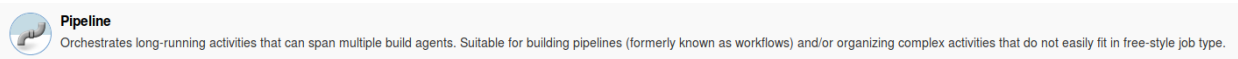
Créer un job sur Jenkins

Jenkins est un serveur d'intégration continue. C'est un planificateur de tâches amélioré. Nous l'avons installé à l'adresse http://jenkins.univ-rouen.fr (http://jenkins.univ-rouen.fr).

Nous allons créer un nouveau **JOB** en cliquant sur le bouton **New Item** en haut à gauche de l'accueil.

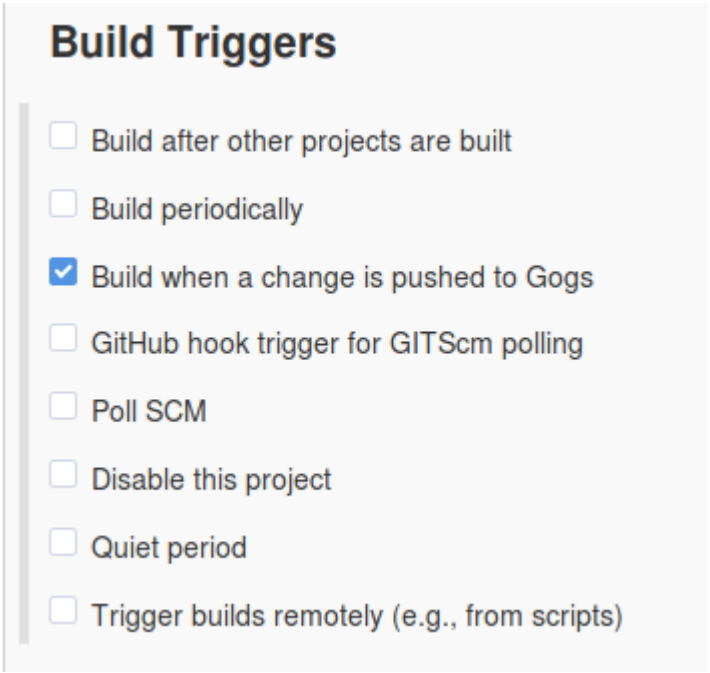


Nous allons adopter le type de **JOB PIPELINE** Celui-ci permet de décrire des déploiements complexes sous la forme d'un code **Groovy** stocké dans fichier nommé **Jenkinsfile** à la racine du projet.

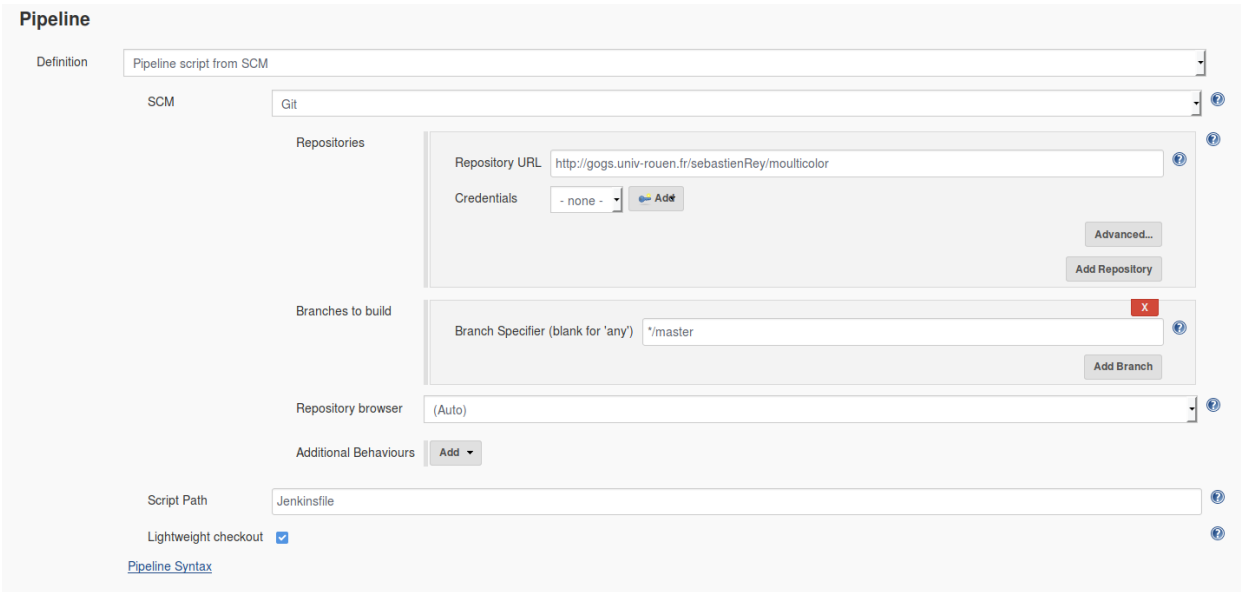


Comme il y a autant de personnes ici que de projets à gérer, nous allons adopter la dénomination suivante pour le nom du **JOB** : *prenomNom-moulticolor*

Pour que Jenkins clone le projet lorsqu'il reçoit le signal en provenance du webhook, il faut lui indiquer quel comportement suivre.



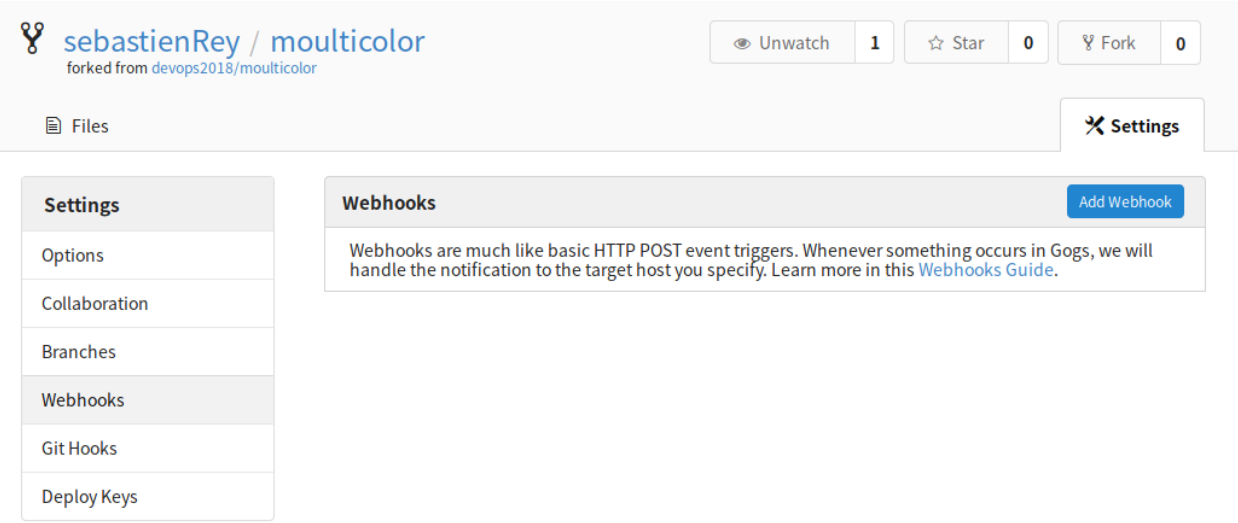
Au niveau du Pipeline, la définition est *Pipeline script from SCM*, avec SCM *Git* et le repository URL : http://gogs.univ-rouen.fr/prenomNom/moulticolor (http://gogs.univ-rouen.fr/prenomNom/moulticolor)



ATTENTION, pour que le signal envoyé par GOGS soit compris par Jenkins, le plugin suivant doit être installé sur Jenkins https://plugins.jenkins.io/gogs-webhook (https://plugins.jenkins.io/gogs-webhook). Pour cette formation le plugin est déjà installé.

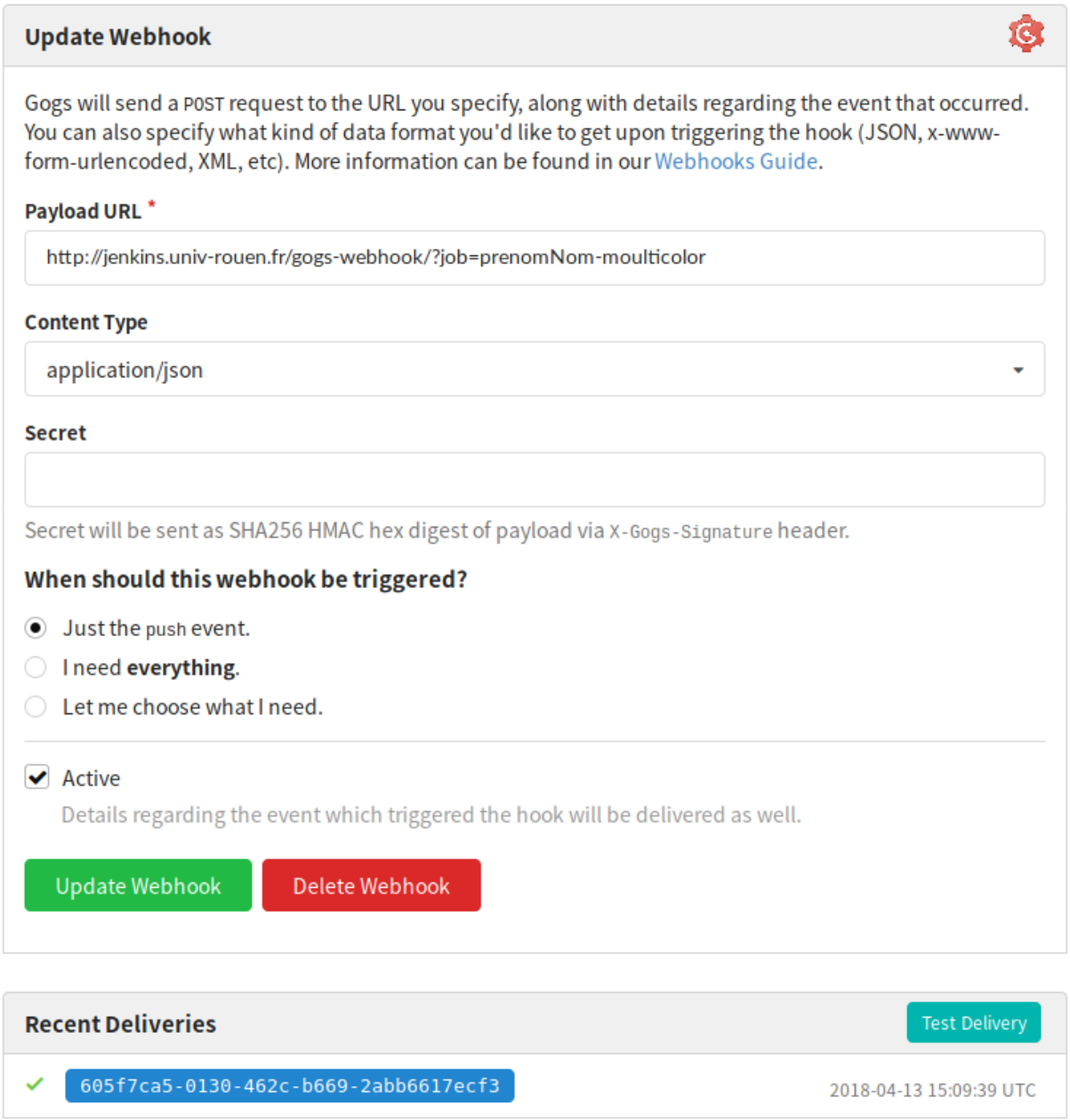
Ajouter le WebHook sur la forge Gogs

Du côté de la forge, dans les settings de votre *fork*, nous allons créer un nouveau *webhook* de type **GOGS** en cliquant sur le bouton *Add Webhook*



Lorsqu'un *commit* déclenchera ce webhook, un contenu *HTTP POST* sera émis vers une URL pointant vers le job **prenomNom-moulticolor** que vous avez déclaré dans l'étape précédente :

```
http://jenkins.univ-rouen.fr/gogs-webhook/?job=prenomNom-moulticolor
```



Ajouter un fichier Jenkinsfile au projet

Le fichier *Jenkinsfile* contient le code Groovy (voir la syntaxe complète (https://jenkins.io/doc/book/pipeline/syntax/)) qui décrit l'ensemble de la chaîne de traitement, ou Pipeline (https://jenkins.io/doc/book/pipeline/), que l'on veut mettre en oeuvre dans notre intégration continue.

Par chance, le plugin *docker-pipeline* (https://plugins.jenkins.io/docker-workflow) qui intègre la gestion de Docker dans les pipelines Jenkins est déjà installé.

Il y'a deux façons d'écrire des pipelines en Groovy pour Jenkins, le format *déclaratif*

ou *scripté*. Nous retiendronts ici la version scriptée, car elle permet d'effectuer plus d'opérations.

```
node {
    checkout scm

    def customImage = docker.build("prenomnom/moulticolor:${env.BUILD_I

    docker.withRegistry('http://registry.univ-rouen.fr', '17137c80-9c11
        customImage.push()
    }
}
```

Le numéro suivant `17137c80-9c11-4933-8d19-86f35480c93a` est un hash généré par Jenkins (via le menu > *credentials*, ajout d'un nouveau credentials) qui permet de ne pas avoir à utiliser le login/mdp d'authentification de la forge en clair dans nos scripts.

Une fois le fichier `Jenkinsfile` créé, il faut le commiter et le pousser.

Ouvrez Jenkins, et normalement vous devriez voir apparaitre votre job. Cliquez dessus, puis sur le dernier numéro **#xx** de *build*, puis la section *ConsoleOutput* pour voir le log du job.

Si tout s'est bien passé, votre image est à présent stockée sur la registry privée :

- Vous devriez voir votre image apparaitre dans la *collection* de la registry à l'URL suivante : http://registry.univ-rouen.fr/v2/_catalog (http://registry.univ-rouen.fr/v2/_catalog).
- Vous pouvez également consulter les tags relatifs à votre image `moulticolor` à l'url suivante : http://registry.univ-rouen.fr/v2/prenomnom/moulticolor/tags/list (http://registry.univ-rouen.fr/v2/prenomnom/moulticolor/tags/list)

Pour aller encore plus loin, quelques pointeurs vers la documentation :

- La documentation des Pipeline : https://jenkins.io/doc/book/pipeline/ (https://jenkins.io/doc/book/pipeline/)
- La documentation de Docker dans le Pipeline, en version script et déclaratifs : https://jenkins.io/doc/book/pipeline/docker/ (https://jenkins.io/doc/book/pipeline/docker/)

- L'api de la Registry v2 pour Docker : https://docs.docker.com/registry/spec/api/#scope (https://docs.docker.com/registry/spec/api/#scope)

Note aux sysadmin, comment accéder à une forge en http:

Pour se connecter sur une registry privée qui ne serait pas en https, il est nécessaire de modifier la configuration du daemon docker faisant tourner les jobs Jenkins.

Ajouter l'url de la registry privée dans `/etc/docker/daemon.json` en créant le fichier si il n'existe pas déjà :

```
{
  "insecure-registries" : ["registry.univ-rouen.fr"]
}
```

Relancer le serveur avec la commande :

```
sudo service docker restart
```

Note aux sysadmin, configurer le plugin Docker pour s'authentifier à une registry privé:

Pour s'authentifier sur une registry privé avec la commande `docker.withRegistry(...)` , nous avons du faire face à un bug Jenkins https://issues.jenkins-ci.org/browse/JENKINS-44143 (). Nous avons résolu celui-ci en suivant ce commentaire (https://issues.jenkins-ci.org/browse/JENKINS-44143?focusedCommentId=325215&page=com.atlassian.jira.plugin.system.issuetabpanels%3Acomment-tabpanel#comment-325215). En se connectant à l'image Jenkins-Docker avec la commande `docker exec -it /bin/bash jenkins` nous avons ajouté un fichier `config.json` dans `/var/jenkins_home/.docker/` avec le code json suivant :

```
{
"auths": {}
}
```

[Etape 4] Déploiement automatiques des images via SSH Jenkins

Configurer la connection ssh de Jenkins vers docker-x

Afin de déployer automatiquement la nouvelle version de l'image, nous allons utiliser une connection SSH. Pour celà, nous avons préinstallé le plugin Publish Over SSH (https://wiki.jenkins.io/display/JENKINS/Publish+Over+SSH+Plugin) sur Jenkins.

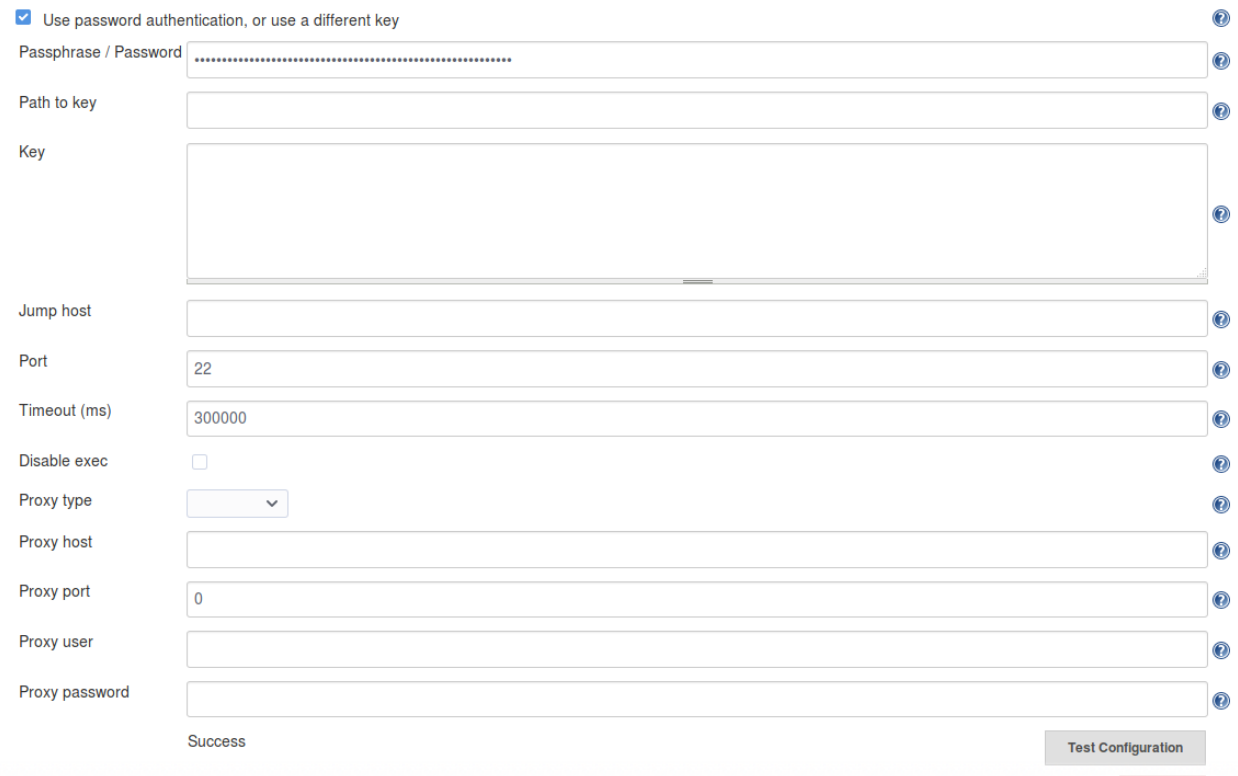
> Remplacer *docker-x* par le nom du serveur qui vous a été attribué

> A la fin de cette manipulation il doit y avoir autant de serveur SSH déclaré que de machines virtuelles docker-x, chacun la sienne !

Pour déclarer un serveur SSH dans l'interface d'administration de Jenkins :

- Aller dans la section *Configurer le système*
- puis section *Publish over SSH*
- Cliquer sur *Ajouter* juste en dessous de la zone de clé
- Saisir *Name* : **docker-x**
- Saisir *Hostname* : **docker-x.univ-rouen.fr** (http://docker-x.univ-rouen.fr)
- Saisir *Username* : **formation**

- Cliquer sur *Advanced…*
- Cliquer sur *Use password authentication, or use a different key*
- Saisir *Password* : **NormanDev2018**
- Cliquer sur *Test configuration*, ça devrait afficher *Success*



Construire un docker-compose pour déployer notre image sur docker-x

Créer un `docker-compose.yml` à la racine de votre projet *prenomNom-moulticolor*

Dans celui-ci, très simple pour le moment, nous indiquons l'image à déployer lors de l'appel de la commande `docker-compose up -d` .

```
version: '2'

services:
    moulticolor:
        image: registry.univ-rouen.fr/prenomnom/moulticolor:${BUILD_ID}
        restart: always
        ports:
          - "80:80"
```

Envoyer des commandes SSH

Ensuite, ajouter le code suivant a la suite du code existant, avant la dernière accolade dans votre `Jenkinsfile` :

Attention à modifier le `docker-x` avec votre nombre ci dessous!

```
sshPublisher(publishers: [
    sshPublisherDesc(configName: 'docker-x', transfers:[
        sshTransfer(execCommand: '''
            docker login -u formation -p NormanDev2018 http://regis
            BUILD_ID=${BUILD_ID} docker-compose up -d
        ''', sourceFiles: 'docker-compose.yml')
    ])
])
```

N'oubliez pas de faire un *commit* et un *push* sur la forge pour observer les changements 😄

Une fois le push effectué, vous pouvez aller voir le job Jenkins, et notamment sa sortie console, et aussi lancer la commande `docker ps` sur votre instance `docker-x` pour vérifier que le conteneur `formation_moulticolor_1` est bien lancé.

Vous pouvez aussi accéder à color.php sur la même URL que précédemment : http://docker-x.univ-rouen.fr/color.php (http://docker-x.univ-rouen.fr/color.php).

Vous pouvez aussi effectuer une nouvelle itération en changeant les couleurs dans `src/color.php` , commitant et poussant, et voir les modifications en rechargeant la page précédente. N'hésitez pas à jeter un oeil au log de votre pipeline Jenkins !

[Etape 5] Complexifier le docker-compose

Architecture multiservice

Un des plus grands intérêts de Docker et de déployer rapidement une infrastructure complète pour un outil.

Nous pouvons, en une seule commande, déployer une application, avec sa base de données.

Modification du docker-compose.yml

Tout d'abord, rajoutons la base de données et phpmyadmin dans le docker-compose :

```
version: '2'

services:
    moulticolor:
        image: registry.univ-rouen.fr/prenomnom/moulticolor:${BUILD_ID}
        restart: always
        ports:
          - "80:80"
        depends_on:
          - db

    db:
        image: mysql:5.7
        restart: always
        volumes:
          - mysql:/var/lib/mysql
        environment:
          MYSQL_DATABASE: moulticolor
          MYSQL_USER: moulticolor
          MYSQL_PASSWORD: moulticolor
          MYSQL_ROOT_PASSWORD: chuuuut!

    phpmyadmin:
        image: phpmyadmin/phpmyadmin
        restart: always
        environment:
          MYSQL_ROOT_PASSWORD: chuuuut!
        ports:
          - "443:80"
        links:
          - db

volumes:
    mysql:
```

Notez l'ajout de `links` dans le service `moulticolor`, qui permettra de donner accès à la base de données depuis le conteneur du service `moulticolor`, avec la résolution du hostname `db`.

Initilisation de la base de données

Après avoir validé et poussé les modifications sur le dépôt git, et attendu quelques secondes (n'hésitez pas à jeter un oeil au log de build sur Jenkins !), PhpMyAdmin devrait être accessible sur http://docker-x.univ-rouen.fr:443 (http://docker-x.univ-rouen.fr:443).

> Les ports ouverts pour la formation étant limités aux 22, 80 et 443, nous faisons tourner phpmyadmin sur le port 443 en HTTP.

Dans PhpMyAdmin, nous allons créer la table `color` :

```
CREATE TABLE color (
    name VARCHAR(20) NOT NULL PRIMARY KEY,
    red TINYINT UNSIGNED NOT NULL DEFAULT 0,
    green TINYINT UNSIGNED NOT NULL DEFAULT 0,
    blue TINYINT UNSIGNED NOT NULL DEFAULT 0
);
```

Et mettre quelques données dedans :

```
INSERT INTO color (
    name, red, green, blue
) VALUES
    ('yourcolor', 128, 128, 128),
    ('yourcolor2', 60, 80, 57)
;
```

Adaptation de src/color.php pour utiliser les couleurs de la base de données

Enfin, modifions le `src/color.php` pour aller chercher les couleurs dans la base de données :

```php
<?php
header("Content-type: image/png");

// Création d'une image de 400x200 pixels
$im = imagecreatetruecolor(400, 200);
$white = imagecolorallocate($im, 255, 255, 255);
$black = imagecolorallocate($im, 0, 0, 0);
$yourcolor = imagecolorallocate($im, 128, 128, 128);
$yourcolor2 = imagecolorallocate($im, 60, 80, 57);

// Connexion
$conn = new mysqli("db", "moulticolor", "moulticolor", "moulticolor");

if($conn->connect_error) {
    die("Impossible de se connecter : " . $conn->connect_error);
}

// Récupération des couleurs
$sql = "SELECT name, red, green, blue FROM color;";

$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        if($row["name"] == "yourcolor") {
            $yourcolor = imagecolorallocate($im, $row["red"], $row["gre
        } elseif($row["name"] == "yourcolor2") {
            $yourcolor2 = imagecolorallocate($im, $row["red"], $row["gr
        }
    }
}

// Fermeture de la connexion
$conn->close();

// The text to draw
$text = 'DEVOPS 2018';
$text2 = 'SEBASTIEN REY';

// Replace path by your own font path
$font = 'consolas.ttf';

// Dessine un double rectangle
imagefilledrectangle($im, 0, 0, 400, 200, $yourcolor);
imagefilledrectangle($im, 10, 10, 390, 190, $yourcolor2);
```

```
// Add the text
imagettftext($im, 20, 0, 50, 50, $white, $font, $text);
imagettftext($im, 12, 0, 50, 80, $white, $font, $text2);

// Sauvegarde l'image
imagepng($im);
imagedestroy($im);
?>
```

Suite à un *commit* et un *push*, vous devriez afficher la page `color.php` : http://docker-x.univ-rouen.fr/color.php (http://docker-x.univ-rouen.fr/color.php)

Interactions entre la base de données et l'application

Dans le PhpMyAdmin, vous pouvez modifier les deux couleur et rafraîchir la page pour voir les changements :

```
UPDATE color SET red=100, green=100, blue=50 where name = 'yourcolor2';
```

Conteneurs

Sur votre instance docker dédiée, vous pouvez voir que des nouveaux conteneurs `formation_db_1` , et `formation_phpmyadmin_1` sont apparus :

```
1 docker ps
```

Un volume local nommé `formation_mysql` a aussi été créé :

```
1 docker volume ls
```

Vous pouvez aussi regarder les configurations des différents conteneurs :

```
1 docker inspect formation_db_1
```

[Etape 6] Pour aller plus loin

Health check

Dans docker-compose, nous pouvons définir un “health check”, une commande dont le résultat indique l’état du conteneur, et permet de le faire redémarrer automatiquement en cas de problème : https://docs.docker.com/compose/compose-file/compose-file-v2/#healthcheck (https://docs.docker.com/compose/compose-file/compose-file-v2/#healthcheck)

Scalabilité

Il est possible de déployer plusieurs conteneurs d’un même service pour avoir de la haute disponibilité et de la répartition de charge (notamment dans un cluster Swarm, voir plus loin) : https://docs.docker.com/compose/compose-file/compose-file-v2/#scale (https://docs.docker.com/compose/compose-file/compose-file-v2/#scale), et notamment faciliter les rolling upgrades.

Clustering

Docker permet d’interconnecter plusieurs nœuds avec la technologie Swarm : https://docs.docker.com/engine/swarm/ (https://docs.docker.com/engine/swarm/)

Cette technologie utilise notamment un système de réseau virtuel, interconnecté avec des tunnels GRE, pour donner accès aux conteneurs au traver de tout le cluster.

Docker Swarm est compatible avec docker-compose.yml avec la commande `docker stack` : https://docs.docker.com/get-started/part5/ (https://docs.docker.com/get-started/part5/). Attention cependant à la version de docker-compose, qui ne sera pas la même.

Il existe aussi une interface de visualisation et d’administration : Portainer (https://portainer.io/ (https://portainer.io/)). Pour l’avoir, il suffit de le rajouter dans docker-compose (attention au port qui doit être libre):

```
portainer:
  image: portainer/portainer
  restart: always
  port:
    - "8080:9000"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
```

Orchestration

Il existe des outils d'orchestration comme Kubernetes qui permettent de gérer un cluster complet avec une interface web, et une gestion de l'authentification. Vous pouvez en savoir plus sur https://kubernetes.io/ (https://kubernetes.io/).

Images utiles

- **alpine** : distribution GNU/Linux légère, utilisée comme base de beaucoup d'images
- **portainer/portainer** : Interface Web de gestion d'un cluster Docker Swarm
- **mysql** : SGBD MySQL
- **phpmyadmin/phpmyadmin** : PhpMyAdmin
- **postgresql** : SGBD PostgreSQL
- **jwilder/docker-gen** : permet de générer des fichiers à partir de métadonnées de conteneurs Docker (notamment vhost Nginx)
- **jrcs/docker-letsencrypt-nginx-proxy-companion** : permet de gérer la génération et le renouvellement de certificats Let's Encrypt
- **jwilder/nginx-proxy** : Serveur Web NGinx, en cherchant un peu, vous pouvez trouver facilement des exemples de docker-compose qui fonctionnent avec les deux précédentes pour avoir une gestion automatique d'un Reverse Proxy, avec HTTPS
- **python** : si vous faites du dev Python
- **openjdk** : si vous préférez le Java
- **phusion/passenger-docker** : si vous vous êtes perdus dans Ruby
- **gogs/gogs** : forge "Gogs"
- **getintodevops/jenkins-withdocker** : Chaîne d'intégration continue Jenkins, avec docker embarqué

- **registry** : Registre Docker autohébergé