*Intégration continue d’application avec Jenkins*

|  |  |
| --- | --- |
| Propriétés | Description |
| **Intitulé long** | Intégration continue d’application avec Jenkins |
| **Formation concernée** | BTS Services Informatiques aux Organisations |
| **Matière** | Bloc 2 SLAM – Conception et développement d’applications |
| **Présentation** | Cette séquence permet de pratiquer l’intégration continue d’application à travers l’outil Jenkins. Les labs proposés permettent de créer des builds et de les lancer manuellement ou de manière automatisée en exploitant des webhooks. |
| **Compétences** | Bloc 2 SLAM – Intégrer en continu les versions d’une solution applicative. |
| **Transversalité** |  |
| **Prérequis** | Commandes de base Linux  Notions de scripts Bash  Recommandé mais facultatif  Activité 1  *Installation et prise en main de DOCKER sur Ubuntu*  Acrivité 2  *Gestion de versions d’application avec Git/Github* |
| **Outils** | Docker, Git, Jenkins |
| **Mots-clés** | CI/CD, intégration continue, Docker, Jenkins, Webhook Jenkins |
| **Durée** | 3h |
| **Niveau de difficulté** | Intermédiaire |
| **Auteur(es)** | Zakari BERREMILI avec la relecture de Amal HECKER et Maelle TAURAND |
| **Version** | V7.0 |
| **Date de publication** | 25/09/2023 |
| **Contenu du package** | Machine virtuelle Lubuntu  Ce labo fait partie de l’atelier CI/CD avec Jenkins |

Intégration continue avec JENKINS

Environnement technique

Docker et Jenkins doivent être installés au préalable sur la première machine virtuelle. Vous pouvez également utiliser la machine virtuelle prête à l’emploi.

Identifiant administrateur : rootsio  
Mot de passe : Sio1234\*

Présentation rapide de Jenkins

Jenkins est un logiciel open source qui permet de réaliser l’intégration continue des modifications fréquentes effectuées dans le code d’une application afin de les tester et de détecter rapidement les problèmes éventuels que ces changements pourraient engendrer. Grâce à Jenkins, les tests de nouveaux builds sont automatisés et effectués à chaque détection de modifications dans le code, ce qui accélère le processus d’intégration continue.

Objectifs

A chaque modification de code d'une application dans le gestionnaire de configuration, Jenkins se charge automatiquement de la recompiler et de la tester. Cette séquence a pour objectif de mettre en place un processus d’intégration continue lors du développement d’une application grâce à Jenkins. Elle se compose de plusieurs labs permettant :

* d’installer et de configurer Jenkins ;
* de créer des builds et les lancer de manière automatisée ;
* de créer et de configurer des pipelines dans Jenkins ;
* d’écrire un JenkinsFile ;
* de créer des déclencheurs pour automatiser le lancement de jobs après une mise à jour d’une application ;
* de créer des scripts Jenkins pour répondre à des besoins spécifiques.

Prérequis

Cette séquence nécessite la connaissance de l’outil de conteneurisation Docker (voir séquence 1).

Elle utilise la machine virtuelle Ubuntu fournie dans le package CI/CD.

Lab 1 : Tutoriel d’installation et de configuration de Jenkins

Scénario**Etape 1 : Téléchargement et exécution de l'image Jenkins Docker**

**Etape 2 : Configuration de Jenkins**

Etape 1 : Téléchargement et exécution de l'image Docker Jenkins

Dans cette étape, vous allez télécharger l'image Docker Jenkins. Vous allez ensuite démarrer une instance de l'image et vérifier que le serveur Jenkins est en cours d'exécution.

Tâche n°1 : Télécharger l’image Docker Jenkins

Une image Jenkins Docker est stockée sur le hub Docker à l’adresse suivante : <https://hub.docker.com/r/jenkins/jenkins> (valide au moment de la rédaction de ce laboratoire) ce site spécifie que vous devez utilisez la commande docker pull jenkins/jenkins:lts-jdk11 pour télécharger le conteneur Jenkins.

**Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement**

Tâche n°2 : Démarrer le conteneur (Docker) Jenkins

Cette commande démarrera le conteneur Jenkins Docker, puis autorisera l'exécution des commandes Docker à l'intérieur de votre serveur Jenkins.

**docker run --rm \**

**-p 8080:8080 \**

**-v jenkins-data:/var/jenkins\_home \**

**-v $(which docker):/usr/bin/docker \**

**-v /var/run/docker.sock:/var/run/docker.sock \**

**-v "$HOME":/home --name jenkins\_server jenkins/jenkins:lts-jdk11**

Les options utilisées dans cette commande **docker run** sont les suivantes :

* **--rm** : cette option supprime automatiquement le conteneur Docker lorsque vous arrêtez de l'exécuter.
* **-u** : cette option spécifie l'utilisateur. Vous voulez que ce conteneur Docker s'exécute en tant que root afin que toutes les commandes Docker entrées saisies dans le serveur Jenkins soient autorisées.
* **-p** : cette option spécifie le port sur lequel le serveur Jenkins écoutera localement les requêtes.
* **-v** :ces options lient les volumes de montage nécessaires pour Jenkins et Docker. Le premier -v spécifie où les données Jenkins seront stockées. Le second -v spécifie où obtenir Docker afin que vous puissiez exécuter Docker dans le conteneur Docker qui exécute le serveur Jenkins. Le troisième -v spécifie la variable PATH pour le répertoire home.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Tâche n°4 : Copiez le mot de passe admin qui s'affiche dans le résultat de la commande, comme surligné sur la page précédente. Il sera nécessaire pour configurer le serveur Jenkins à l’étape 2.

**Remarque** : N'entrez aucune commande dans cette fenêtre de serveur. Si vous arrêtez accidentellement le serveur Jenkins, vous devrez entrer à nouveau la commande docker run (Tâche 3 ci-dessus).

Si vous perdez le mot de passe, ou que vous avez oublié de le copier à l’étape 1, ou qu'il ne s'affiche pas comme indiqué ci-dessus, ou encore que vous devez redémarrer le serveur Jenkins, vous pouvez toujours le récupérer en accédant à la ligne de commande du conteneur Jenkins Docker comme indiqué ci-dessous dans une deuxième fenêtre de terminal

**$ docker exec -it jenkins\_server /bin/bash**

root@b86a63302959:/# **cat /var/jenkins\_home/secrets/initialAdminPassword**

3bbf5d586d4d4c6b807f9a9a072ad188

root@b86a63302959:/# **exit**

Votre identifiant de conteneur (b86a63302959 ci-dessus) et votre mot de passe seront différents.  
L’identifiant de connexion est « admin ».

Étudiez les niveaux d'abstraction actuellement en cours d'exécution sur votre ordinateur.

Le diagrammeci-dessous montre les niveaux d'abstraction dans cette implémentation Docker Inside-Docker (dind). Ce niveau de complexité n'est pas inhabituel dans les réseaux et les infrastructures cloud actuelles.

Etape 2 : Configuration de Jenkins

Dans cette étape, vous allez terminer la configuration initiale du serveur Jenkins.

Tâche n°1 : Ouvrir un onglet du navigateur web et

accéder à http://localhost:8080/ . Se connecter à l'aide de votre mot de passe administrateur précédemment copié.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Tâche n°2 : Installez les plugins Jenkins recommandés.

Cliquez sur **Installer les plugins suggérés** et attendre que Jenkins télécharge et installe les plugins.

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

Remarque : Dans la fenêtre du terminal depuis laquelle le serveur a été lancé, il est possible de suivre l’installation des plugins à travers les messages de journal affichés au fur et à mesure que les plugins s’installent. Assurez-vous que vous ne fermez pas cette fenêtre de terminal. Si vous avez besoin de lancer des commandes en ligne, ouvr une autre fenêtre de terminal.

Tâche n°3 : Ignorer la création d'un nouvel utilisateur administrateur.

Une fois l'installation terminée, la fenêtre **Créer le 1er utilisateur Administrateur** s'affiche.

Pour l'instant, cliquez sur **Skip and continue as admin** en bas.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Tâche n°4 : Ignorer la création d'une nouvelle d'instance.

Dans la fenêtre **Configuration de l’Instance**, ne modifiez rien. Cliquez sur **Sauver et continuer** en bas.

Tâche n°5 : Commencer à utiliser Jenkins.

Dans la fenêtre suivante, cliquez sur **Commencez à utiliser Jenkins**. Vous devriez maintenant être sur le tableau de bord principal avec un **Bienvenue Sur Jenkins !** message.

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Lab2 : Jobs et Pipelines Jenkins

Scénario **Etape 1 : Création du dépôt d’application sur Git**

**Etape 2 : Modification de l'exemple d'application et remontée des modifications sur Git**

**Etape 3 : Utilisation de Jenkins pour exécuter une version de votre application**

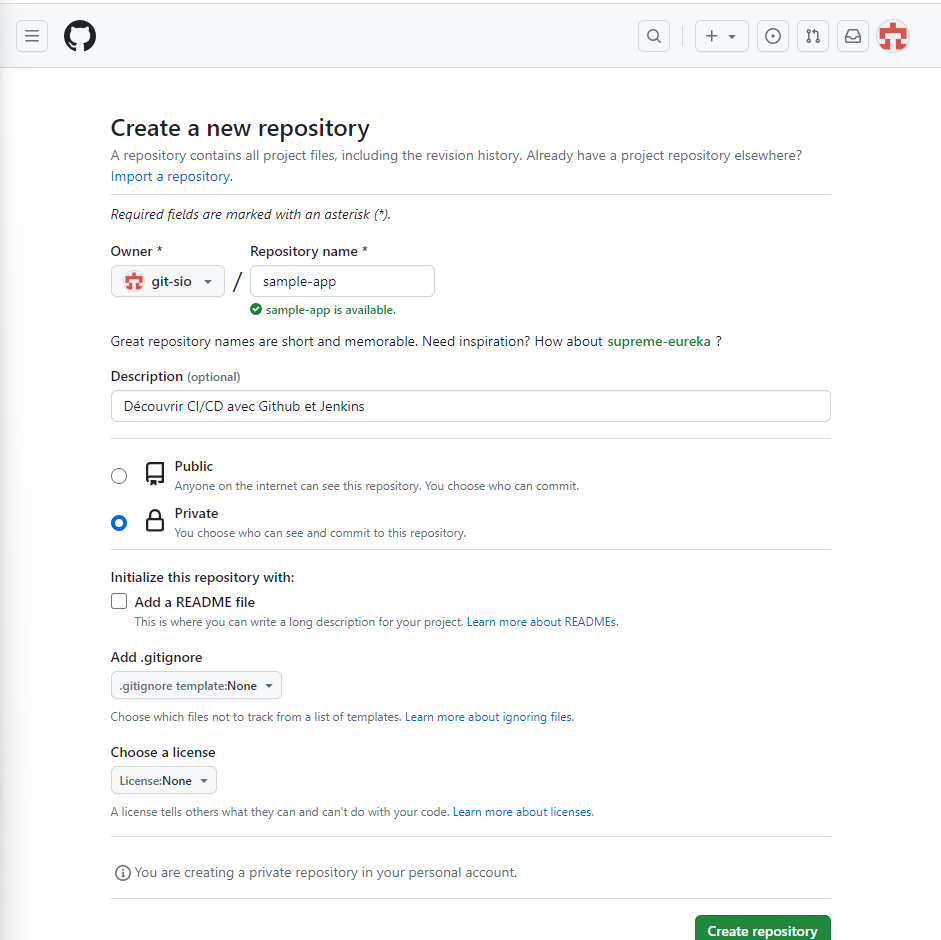
**Etape 4 : Utilisation de Jenkins pour tester une construction**

**Etape 5 : Création d’un pipeline dans Jenkins**

Etape 1 : Création du dépôt d’application sur Git

Dans cette étape, vous allez créer un dépôt GitHub pour valider les fichiers sample-app qui vous sont fournis dans la machine virtuelle. Vous utiliserez un compte GitHub. Si vous ne possédez pas encore de compte, rendez-vous sur github.com.

Tâche n°1 : Depuis votre compte GitHub, créer un nouveau référentiel sample-app.



Tâche n°2 : Configurer vos informations d'identification Git localement dans la machine virtuelle si cela n’a pas été déjà fait (voir séquence 2 : gestion de versions d’applications avec Git/Github) ;

Tâche n°3 : Initialiser le répertoire en tant que dépôt Git.

Vous pouvez utiliser les fichiers d'exemples d'application que vous avez créés précédemment. Toutefois, ces fichiers sont également stockés pour votre commodité dans le répertoire **labs/jenkins/sample-app**.

Tâche n°4 : Pointer vers le dépôt nouvellement créé sur GitHub

* Utiliser la commande **git remote add** pour ajouter une URL Git avec un alias (origin) et pointez vers le dépôt nouvellement créé sur GitHub. En utilisant l'URL du dépôt Git que vous venez créer (tâche1), vous devez seulement remplacer le nom ***github-username*** dans la commande suivante par votre nom d'utilisateur GitHub.

**Remarque** : Votre nom d'utilisateur GitHub est sensible à la casse.

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

**~$ git remote add** **origin** https://github.com/git-sio/sample-app.git



Tâche n°4 : Activer le suivi des fichiers, valider et pousser les fichiers de l’exemple d'application dans le référentiel GitHub.

* Utilisez la commande **git add** pour mettre en scène les fichiers dans le répertoire **jenkins/sample-app**. Utilisez l'argument astérisque (\*) pour mettre en scène **tous** les fichiers dans le répertoire courant.

**~/labs/jenkins/sample-app$ git add \***

* Utilisez la commande git status pour voir les fichiers et répertoires qui sont mis en scène et prêts à être validés dans votre dépôt GitHub.

**~/labs/jenkins/sample-app$ git status**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

* Valider les fichiers dont le suivi a été activé et commencer à suivre les modifications. Ajouter un message de votre choix ou utilisez celui-ci : **"Commit(validation) des fichiers sample-app."**.

**~/labs/jenkins/sample-app$ git commit -m "Commit(validation) des fichiers sample-app."**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* Pousser vos fichiers sample-app locaux dans votre dépôt GitHub.

**~/labs/jenkins/sample-app$ git push origin master**

**Username for 'https://github.com': nom d’utilisateur**

**Password for 'https://AllJohns@github.com': Jeton (Attention !!! A présent GitHb à**

**adopter le fonctionnement par jeton (Token))**

* Pour récupérer un jeton, voir la séquence Git

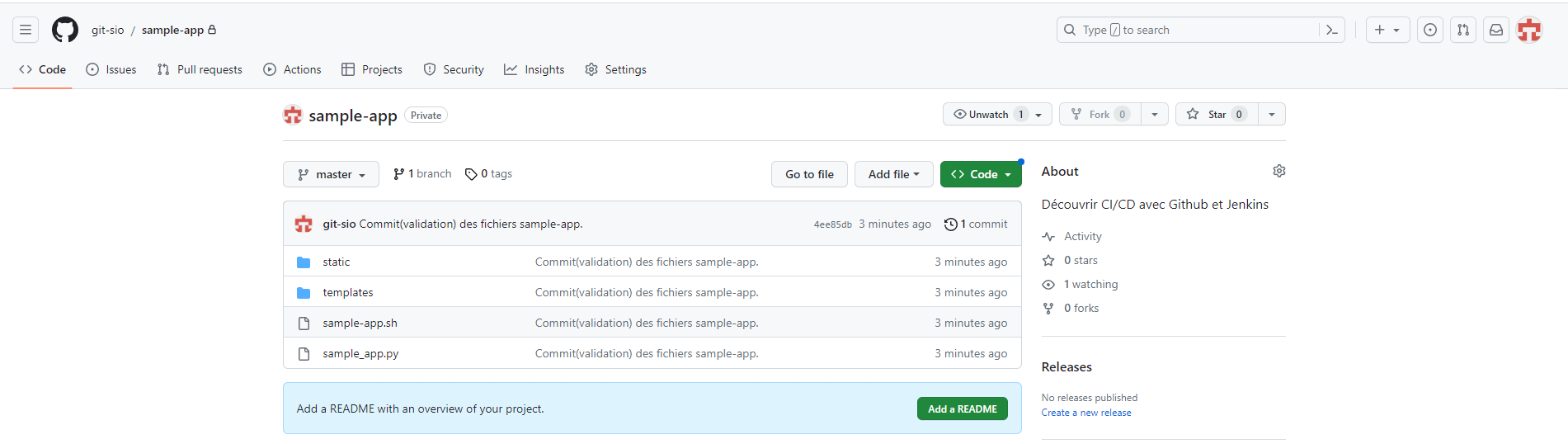
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**Remarque** : Si, au lieu d'une demande pour votre nom d'utilisateur, vous recevez un message : **L'extension 'Git' veut vous connecter en utilisant GitHub, alors vous avez mal configuré soit vos informations d'identification GitHub et/ou lURL GitHub. L'URL doit avoir le nom d'utilisateur correct sensible à la casse et le nom du référentiel que vous avez créé. Pour annuler votre précédente** commande git add, utilisez la commande **git remote rm origin**. Revenez ensuite à la tâche 2 en vous assurant d'entrer les informations d'identification correctes puis en saisissant un URL correcte.

Si, après avoir saisi votre nom d'utilisateur et votre mot de passe, vous obtenez une erreur fatale indiquant que le dépôt n'est pas trouvé, vous avez très probablement soumis une URL incorrecte. Vous devrez inverser votre commande **git add** avec la commande **git remote rm origin**.

* Vérifier que vos fichiers sont bien présents sur GitHub.



Etape 2 : Modification de l'exemple d'application et remontée des modifications sur Git

Précédemment, vous avez installé une image Jenkins Docker qui utilise le port 8080. Rappelez-vous que les fichiers de paramétrage du service web spécifient également le port 8080. Le serveur Flask et le serveur Jenkins ne peuvent pas utiliser simultanément le port 8080.

Dans cette étape, vous allez donc modifier le numéro de port utilisé par les fichiers sample-app, exécuter à nouveau sample-app pour vérifier qu'il fonctionne sur le nouveau port, puis envoyer vos modifications dans votre dépôt GitHub.

Tâche n° 1 : Consulter les fichiers sample\_app.py et sample-app.sh présents dans le répertoire **~/labs/jenkins/sample-app**  . .

Tâche n°2 : Apporter les modifications suivantes :

* Dans sample\_app.py, changer l'instance du port 8080 à 5050 comme indiqué ci-dessous.

from flask import Flask

from flask import request

from flask import render\_template

sample = Flask(\_\_name\_\_)

@sample.route("/")

def main():

return render\_template("index.html")

if \_\_name\_\_ == "\_\_main\_\_":

sample.run(host="0.0.0.0", port=5050)

* Dans sample-app.sh, changer les trois instances du port 8080 à 5050 comme indiqué ci-dessous.

#!/bin/bash

mkdir tempdir

mkdir tempdir/templates

mkdir tempdir/static

cp sample\_app.py tempdir/.

cp -r templates/\* tempdir/templates/.

cp -r static/\* tempdir/static/.

echo "FROM python" >> tempdir/Dockerfile

echo "RUN pip install flask" >> tempdir/Dockerfile

echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile

echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile

echo "COPY sample\_app.py /home/myapp/" >> tempdir/Dockerfile

echo "EXPOSE 5050" >> tempdir/Dockerfile

echo "CMD python3 /home/myapp/sample\_app.py" >> tempdir/Dockerfile

cd tempdir

docker build -t sampleapp.

docker run -t -d -p 5050:5050 --name samplerunning sampleapp

docker ps -a

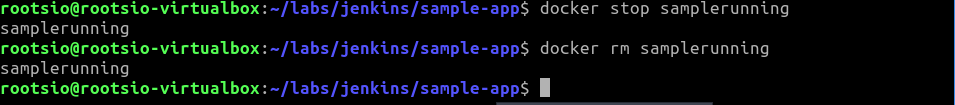
Tâche n°3 : Exécuter le script sample-app.sh dans le shell courant

Lancer et tester l’application en exécutant la commande suivante :

* Entrez la commande bash pour créer votre application à l'aide du nouveau port 5050.
* ~/labs/jenkins/sample-app$ **bash ./sample-app.sh**

****

Tâche n°4 : Après avoir testé le fonctionnement <http://localhost:5050> vous pouvez stopper le conteneur puis le supprimer.



* Tâche n°5 : A l’aide des commandes git appropriées, valider les modifications et pousser (push) celles-ci vers le dépôt en précisant le message suivant : "Changement de port de 8080 à 5050.". (Consulter la fiche savoir Git)

Tâche n°6 : Vérifier que votre dépôt Git est mis à jour sur le site GitHub. Vous devriez voir votre nouveau message (Changement de 8080 à 5050) (l’horodatage de validation a été mis à jour).

BUne image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

Etape 3 : Utilisation de Jenkins pour exécuter une version de votre application

L'unité fondamentale de Jenkins est le Job (également connu sous le nom de projet). Vous pouvez créer des tâches qui effectuent diverses tâches, notamment les suivantes :

* Récupérer du code à partir d'un référentiel de gestion de code source tel que GitHub, GitLab ….
* Créer une application à l'aide d'un script ou d'un outil de construction.
* Créer un package d’une application et l'exécuter sur un serveur

**Dans cette partie, vous allez créer un Job Jenkins simple qui récupère la dernière version de votre sample-app à partir de GitHub et exécute le script de construction. Dans Jenkins, vous pouvez ensuite tester votre application et l'ajouter à un pipeline de développement**

A faire : Exécuter cette commande pour permettre à Jenkins de lancer docker   
$sudo chmod 666 /var/run/docker.sock

Tâche n°1 : Créer une nouvelle tâche.

* Cliquer sur le lien **Créer un job** directement en dessous de la page **Bienvenu sur Jenkins!** message. Vous pouvez également cliquer sur **Nouveau Item** dans le menu de gauche.

Une image contenant texte, logiciel, Page web, Site web

Description générée automatiquement



* Dans le champ **Entrez un nom d'élément**, renseigner le nom **BuildAppJob**.
* Cliquer sur **Projet Freestyle** comme type de tâche. Dans la description, l'abréviation SCM signifie « gestion de la configuration logicielle », qui est une classification des logiciels qui est responsable du suivi et du contrôle des modifications apportées aux logiciels.
* Descendre et validez « ok »

Tâche n°2 : Configurer le Job BuildAppJob

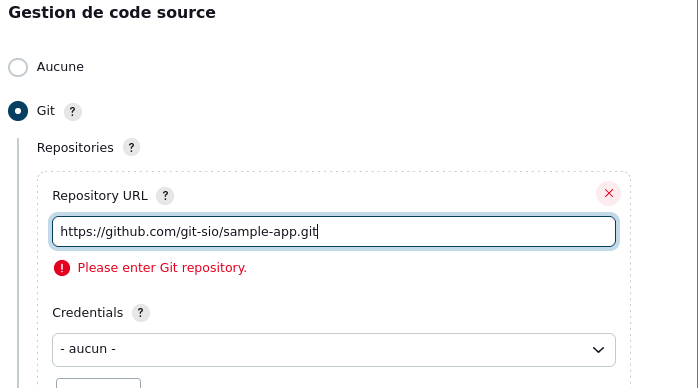
Vous êtes maintenant dans la fenêtre de configuration où vous pouvez entrer des détails sur votre Job. Les onglets situés en haut ne sont que des raccourcis vers les sections ci-dessous. Cliquer sur les onglets pour explorer les options que vous pouvez configurer.

* Cliquer sur l'onglet **General**, ajouter une description pour votre travail. Par exemple, "**Mon premier Job Jenkins.**"

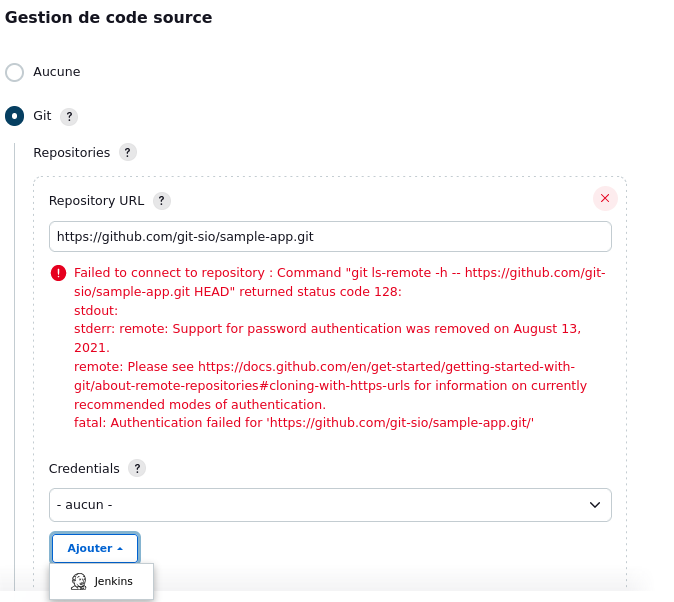
Une image contenant texte, Police, nombre, ligne

Description générée automatiquement

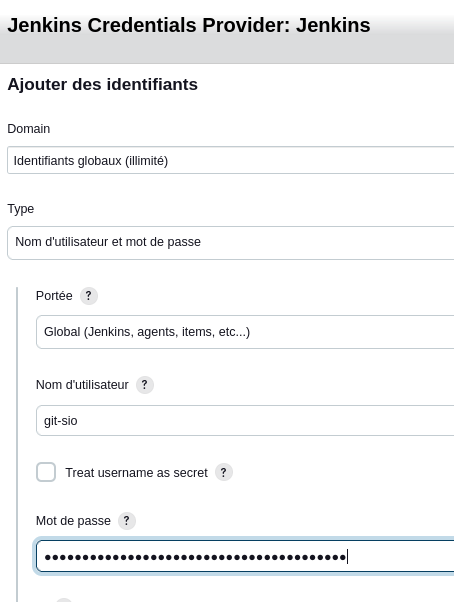
* Cliquer sur l'onglet **Gestion de code Source** et cliquer sur le bouton radio **Git**. Dans le champ URL du référentiel, ajouter votre lien de référentiel GitHub pour l'exemple d'application en prenant soin de saisir votre nom d'utilisateur correctement car sensible à la casse.



* Pour **Credentials** cliquer sur le bouton **Add** et choisir **Jenkins**.



* Dans la boîte de dialogue **Ajouter des identifiants**, renseigner votre nom d'utilisateur et votre mot de passe GitHub, puis cliquer sur **Add**. (Attention !!! Ici le mot de passe est en fait un jeton)



Une image contenant texte, capture d’écran, ligne, Police

Description générée automatiquement

**Remarque** : Vous recevrez un message d'erreur indiquant que la connexion a échoué. Cela est dû au fait que vous n'avez pas encore sélectionné les informations d'identification.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* Dans la liste déroulante des **Credentials** où il est actuellement indiqué **None**, choisir les informations d'identification que vous venez de configurer.
* Après avoir ajouté l'URL et les informations d'identification correctes, Jenkins teste l'accès au référentiel dépôt (ou repository). Vous ne devriez plus avoir de message d'erreur. Si c’est quand même le cas, vérifiez votre URL et vos informations d'identification. Vous devrez les créer à la suite car il n'y a aucun moyen à ce stade de supprimer ceux que vous avez entrés précédemment.
* Une image contenant texte, capture d’écran, Police, nombre

  Description générée automatiquementEn haut de la fenêtre de configuration de cliquer sur l'onglet **Build Steps**.
* Pour la liste déroulante **Ajouter une étape au build**, choisir **Exécuter un script shell.**

.

* Dans le champ **Command**, entrer la commande que vous utilisez pour exécuter le script build pour sample-app.sh.  **bash ./sample-app.sh**

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* Cliquer sur le bouton **Sauver** (Enregistrer). Vous êtes retourné au tableau de bord Jenkins avec le **BuildAppJob** sélectionné.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementUne image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Tâche n°3 : Dire à Jenkins de construire l’application

* Sur le côté gauche, cliquer sur **Lancer un build** pour démarrer le travail. Jenkins va télécharger votre dépôt Git et exécuter la command build **bash ./sample-app.sh**. Votre construction devrait réussir parce que vous n'avez rien changé dans le code.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Tâche n°4 : Accéder aux détails de la construction (le build)

* Sur la gauche, dans la section **Historique des builds**, cliquer sur votre numéro de build qui devrait être le **#1** sauf si vous avez créé l'application plusieurs fois.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Tâche n°5 : Afficher la sortie de la console.

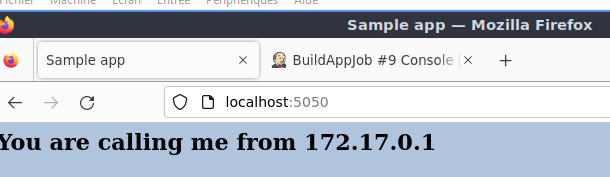
* Sur la gauche, cliquer sur **Sortie de la console**. Vous devriez voir un résultat similaire à ce qui suit. Noter les messages de succès en bas ainsi que la sortie de la commande **docker ps -a**. Deux conteneurs docker sont en cours d'exécution : un pour votre application d'échantillonnage fonctionnant sur le port local 5050 et un pour Jenkins sur le port local 8080.

Une image contenant texte, capture d’écran, lettre, document

Description générée automatiquement

Tâche n°6 : Ouvrir un autre onglet du navigateur Web et vérifier que l'application est en cours d'exécution.

* Saisir l'adresse locale, **localhost: 5050**. Vous devriez voir le contenu de votre index.html affiché en couleur de fond bleu acier clair avec **You are calling me from 172.17.0.1** affiché en taille H1.

****

* Vous pouvez stopper et supprimer le conteneur.

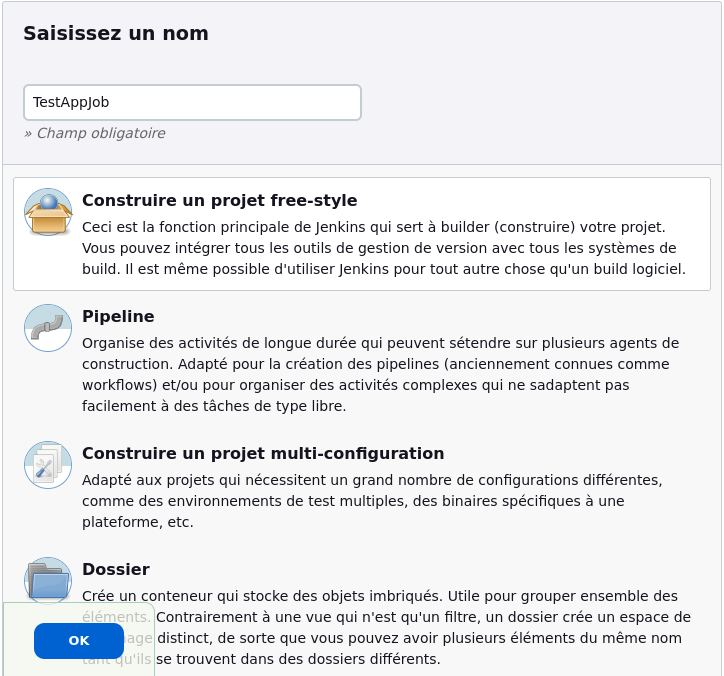
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Etape 4 : Utilisation de Jenkins pour tester une construction

Dans cette étape, vous allez créer une deuxième tâche qui teste la construction pour vous assurer qu'elle fonctionne correctement.

Tâche n°1 : Créer un nouveau Job pour tester sample-app.



* Retourner à l'onglet du navigateur Web Jenkins et cliquer sur le lien **Jenkins** dans le coin supérieur gauche pour revenir au tableau de bord principal.
* Cliquer sur le lien **Nouveau Item** pour créer un nouvel emploi.
* Dans le champ Entrez un nom d'élément, renseigner le nom **TestAppJob**.
* Cliquer sur **Free-style** comme type de tâche.
* Faire défiler jusqu'en bas et cliquez sur **OK**.

Tâche n°2 : Configurer le job Jenkins : « TestAppJob ».

* Ajouter une description pour votre travail. Par exemple, "Mon premier test Jenkins."
* Laisser Source Code Management définie sur **Aucune**.

Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

* Cliquer sur l'onglet **Ce qui déclenche le build** et cocher la case **Construire après le build sur d’autre projets**. Pour les **Projects à surveiller**, renseigner le nom **BuildAppJob**.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Tâche n°3 : Écrire le script de test qui doit s'exécuter après une version stable de BuildAppJob.

* Cliquer sur l'onglet **Build Steps**.
* Cliquer sur **Add build step** et choisir **Exécuter un script shell**.
* Entrer le script suivant. La commande **if** doit être sur une seule ligne, y compris le ; **then**. Cette commande (**grep[[1]](#footnote-1)**) va parcourir la sortie retournée par la commande cURL pour voir si **You are calling me from 172.17.0.1** est retourné. Si c'est vrai, le script sort avec un code de 0, ce qui signifie qu'il n'y a pas d'erreur dans la construction de **BuildAppJob**. Si elle est fausse, le script sort avec un code de 1, ce qui signifie que le **BuildAppJob a échoué**.
* (ATTENTION ICI LE TEXTE EST EN ANGLAIS, POSSIBILITE D’ERREUR, VOUS POUVEZ SIMPLIFIER (grep « 172.17.0.1 »)

|  |
| --- |
| if curl http://172.17.0.1:5050/ | grep "You are calling me from 172.17.0.1"; then  exit 0  else  exit 1  fi |

* Clique sur **Save**  et retourne au tableau de bord (en haut à gauche)

Une image contenant texte, ligne, logiciel, nombre

Description générée automatiquement

Tâche n°4 : Dire à Jenkins d’exécuter à nouveau le job BuildAppJob.

* Actualiser la page Web avec le bouton d'actualisation de votre navigateur.
* Vous devriez maintenant voir vos deux tâches répertoriées dans un tableau plus bas. Sur le job **BuildAppJob**, cliquer sur le bouton build à l'extrême droite (une horloge avec une flèche).

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Tâche n°5 : Vérifier que les deux jobs sont terminés.

* Si tout se passe bien, vous devriez voir l'horodatage pour la mise à jour de la colonne **Last Success** pour **BuildAppJob** et **TestAppJob**. Cela signifie que votre code pour les deux tâches s'est exécuté sans erreur. Mais vous pouvez également le vérifier par vous-même.

**Remarque** : Si les horodatages ne sont pas mis à jour, assurez-vous que l'actualisation automatique est activée ou réactualisez manuellement.

Une image contenant texte, Police, nombre, logiciel

Description générée automatiquement

* Cliquer sur le lien pour **TestAppJob**. Cliquer sur le lien correspondant à votre dernière version, puis cliquer sur **Sortie de la console**. Vous devriez voir un résultat similaire à ce qui suit :

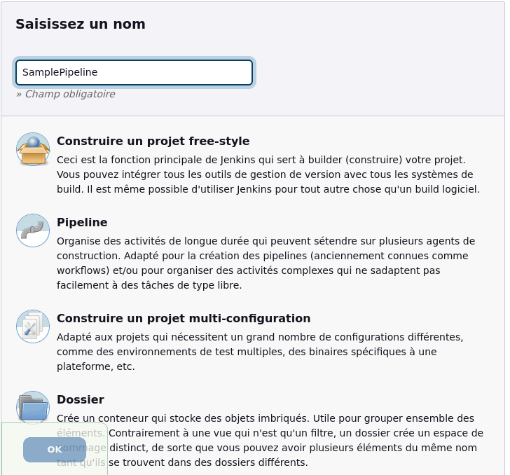
Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Etape 5 : Création d’un pipeline dans Jenkins

Bien que vous puissiez actuellement exécuter vos deux jobs en cliquant simplement sur le bouton Build Now pour le BuildAppJob, les projets de développement de logiciels sont généralement beaucoup plus complexes. Ces projets peuvent bénéficier grandement de l'automatisation des builds pour l'intégration continue des changements de code et la création continue de builds de développement prêts à être déployés. C'est l'essence même du CI/CD.

Un pipeline peut être automatisé pour s'exécuter sur la base d'une variété de déclencheurs, y compris périodiquement, sur la base d'un sondage GitHub pour les changements, ou à partir d'un script exécuté à distance. Cependant, dans cette partie, vous allez scripter un pipeline dans Jenkins pour exécuter vos deux applications chaque fois que vous cliquez sur le bouton Build Now du pipeline.

Tâche n°1 : Créer un Pipeline

* Cliquer sur le lien **Jenkins** en haut à gauche, puis sur **Nouveau Item**
* Dans le champ **Saisissez un nom**, taper **SamplePipeline**.
* Sélectionner **Pipeline** comme type de tâche.
* Faire défiler jusqu'en bas et cliquer sur **OK**.

Tâche n°2 : Configurer SamplePipeline

* En haut, cliquer sur les onglets et examiner z chaque section de la page de configuration. Noter qu'il existe un certain nombre de façons différentes de déclencher une construction. Pour le travail **SamplePipeline**, la construction sera à déclencher manuellement.
* Dans la section **Pipeline script**, ajouter le script suivant :

node {

stage('Preparation') {

catchError(buildResult: 'SUCCESS') {

sh 'docker stop samplerunning'

sh 'docker rm samplerunning'

}

}

stage ('Build') {

build 'BuildAppJob'

}

stage('Results') {

build 'TestAppJob'

}

}

Ce script effectue les opérations décrites à la page suivante :

Une image contenant texte, capture d’écran, affichage, nombre

Description générée automatiquement

Ce script effectue les opérations suivantes :

1. Il crée une construction à un seul nœud par opposition à une configuration distribuée ou multi-nœuds. Les configurations distribuées ou multi-nœuds sont pour des pipelines plus grands que celui que vous construisez dans ce laboratoire et sont au-delà de la portée de ce cours.
2. Dans l'étape de **préparation**, **SamplePipeline** s'assure d'abord que toutes les instances précédentes du conteneur docker **BuildAppJob** sont arrêtées et supprimées. Mais s'il n'y a pas encore de conteneur en cours d'exécution, vous obtiendrez une erreur. Par conséquent, vous utilisez la fonction catchError pour attraper toutes les erreurs et renvoyer une valeur "SUCCESS". Cela permettra au pipeline de passer à l'étape suivante.
3. Dans l'étape **Build**, **SamplePipeline** va construire votre **BuildAppJob**.
4. Dans l'étape **Results**, **SamplePipeline** va construire votre **TestAppJob**.
5. Cliquer sur **Save,** vous êtes retournerez au tableau de bord Jenkins de la tâche **SamplePipeline**.

Tâche n°3 : Exécuter SamplePipeline

* Sur la gauche, cliquer sur **Lancer un Build** pour exécuter le job **SamplePipeline**. Si vous avez codé votre script Pipeline sans erreur, la vue des étapes devrait afficher trois cases vertes avec le nombre de secondes nécessaires à la construction de chaque étape. Sinon, cliquer sur Configure à gauche pour revenir à la configuration de **SamplePipeline** et vérifier votre script Pipeline.

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Tâche n°4 : Vérifier la sortie SamplePipeline

* Cliquer sur le lien de construction le plus récent, puis cliquer sur **Console Output**. Vous devriez voir un résultat similaire à ce qui suit :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Etape 6 : Pour aller plus loin, déposer une JenkinsFile dans GIT

Une évolution souhaitable serait également de stocker le script contenant le pipeline directement dans Git. C’est très utile et il suffit de créer un fichier JenkinsFile et de le déposer sur Git pour que Jenkins le charge lors du pipeline.

Défi : Créer et utiliser JenkinsFile

Pour vous aider : <https://youtu.be/GNT-zTQ-4aA> ou [Jenkins - tutoriaux français - Xavki](https://xavki.blog/jenkins-tutoriaux-francais/) vidéo 17.

Lab3 : Automatisation du Job lors d’une mise à jour (commit)

Scénario**Etape 1 : Paramétrage d’un conflit de port empêchant Flask de fonctionner**

**Etape 2 : Pousser de la modification et constater l’erreur**

**Etape 3 : Préparation de l’automatisation Webhook (rappel http) en gérant le problème d’adresse privée**

**Etape 4 : Configuration du Webhook (rappel http) sur Jenkins et Git**

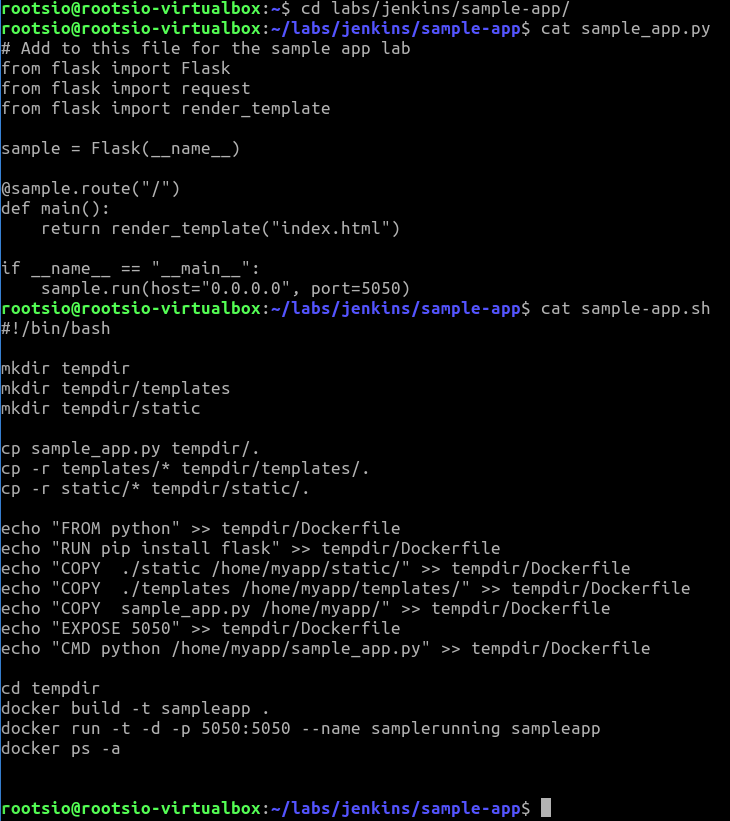
**Etape 5 : Pousser de la modification pour constater l’automatisation**

**Etape 1 :** **Paramétrage d’ un conflit de port empêchant Flask de fonctionner**

Dans cette étape, vous allez simuler une erreur en paramétrant votre serveur web Flask pour fonctionner sur un port déjà utilisé et créer ainsi un conflit. Vous constaterez l’erreur sur Jenkins.

Tâche n°1 : Editer les fichiers d'exemples d'application sample\_app.py et sample-app.sh présents dans le répertoire ~/labs/jenkins/sample-app .

* S’assurer que vous êtes toujours dans le répertoire **~/labs/jenkins/sample-app** car il s'agit des fichiers associés à votre dépôt GitHub. Pour créer l’erreur ouvrir **sample\_app.py** et **sample-app.sh** pour l'édition



Tâche n°2 : Vérifier la suppression du conteneur samplerunning , à l’aide de la commande docker appropriée. Juste avant, s’assurer que conteneur samplerunning a bien été supprimé

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Tâche n°3 : Modifier vos fichiers

* Dans sample\_app.py, changer l'instance du port 5050 à 8080 comme indiqué ci-dessous.

from flask import Flask

from flask import request

from flask import render\_template

sample = Flask(\_\_name\_\_)

@sample.route("/")

def main():

return render\_template("index.html")

if \_\_name\_\_ == "\_\_main\_\_":

sample.run(host="0.0.0.0", port=8080)

* Dans sample-app.sh, changer les trois instances du port 5050 à 8080 comme indiqué ci-dessous.

#!/bin/bash

mkdir tempdir

mkdir tempdir/templates

mkdir tempdir/static

cp sample\_app.py tempdir/.

cp -r templates/\* tempdir/templates/.

cp -r static/\* tempdir/static/.

echo "FROM python" >> tempdir/Dockerfile

echo "RUN pip install flask" >> tempdir/Dockerfile

echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile

echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile

echo "COPY sample\_app.py /home/myapp/" >> tempdir/Dockerfile

echo "EXPOSE 8080" >> tempdir/Dockerfile

echo "CMD python3 /home/myapp/sample\_app.py" >> tempdir/Dockerfile

cd tempdir

docker build -t sampleapp.

docker run -t -d -p 8080:8080 --name samplerunning sampleapp

docker ps -a

Tâche n°4 : En utilisant les commandes git appropriées, pousser (push) vos modifications vers GitHub

* Entrer les commandes suivantes :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**~/labs/jenkins/sample-app$ git add \***

**~/labs/jenkins/sample-app$ git status**

* Valider le commit.

~/labs/jenkins/sample-app$ git commit -m "Changement de port de 5050 à 8080." 

**Etape 2 :** **Pousser la modification et constater l’erreur**

* Pousser (push) vos modifications

**~/labs/jenkins/sample-app$ git push origin master**

**Username for 'https://github.com': nom d’utilisateur**

**Password for 'https://AllJohns@github.com': Jeton (Attention !!! A présent GitHub à a**

**adopter adopté le fonctionnement par jeton (Token))**

REMARQUE : Si vous régénérez un jeton, le précédent deviendra obsolète. Il vous faudra donc mettre à jour vos paramètres d’identification à GitHub dans Jenkins :

(Dashboard/BuilAppJob/configure/Source Code Management/Credentials)

Une image contenant texte, capture d’écran, Police

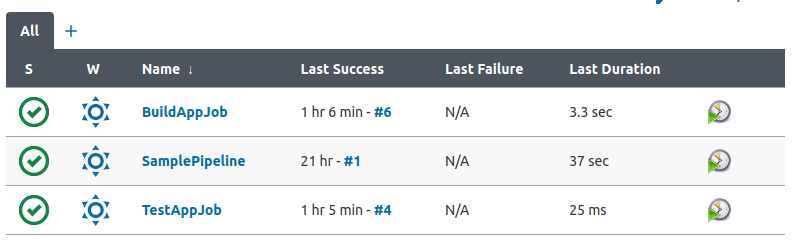
Description générée automatiquement

* Adapter TestAppJob

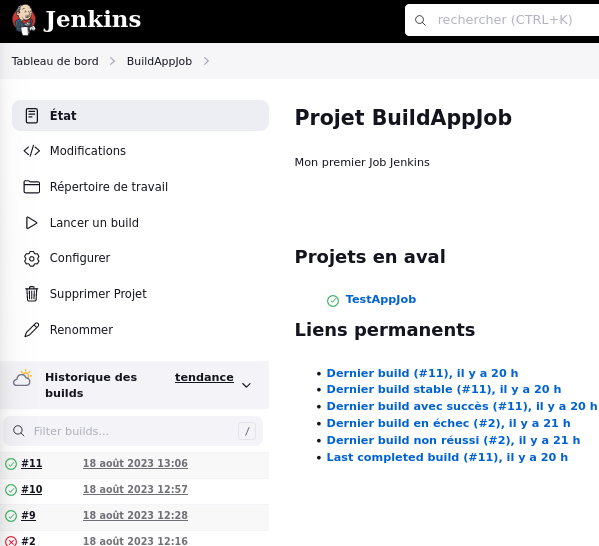
Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

* Se rendre dans jenkins sur le job BuildAppJob



* Lancer le job en cliquant sur Lancer un Build



* Constater que le test a très logiquement échoué

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* Remettre TestAppJob en l’état initial

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

**Etape 3 : Préparation de er l’automatisation Webhook (rappel http) en gérant le problème d’adresse privée**

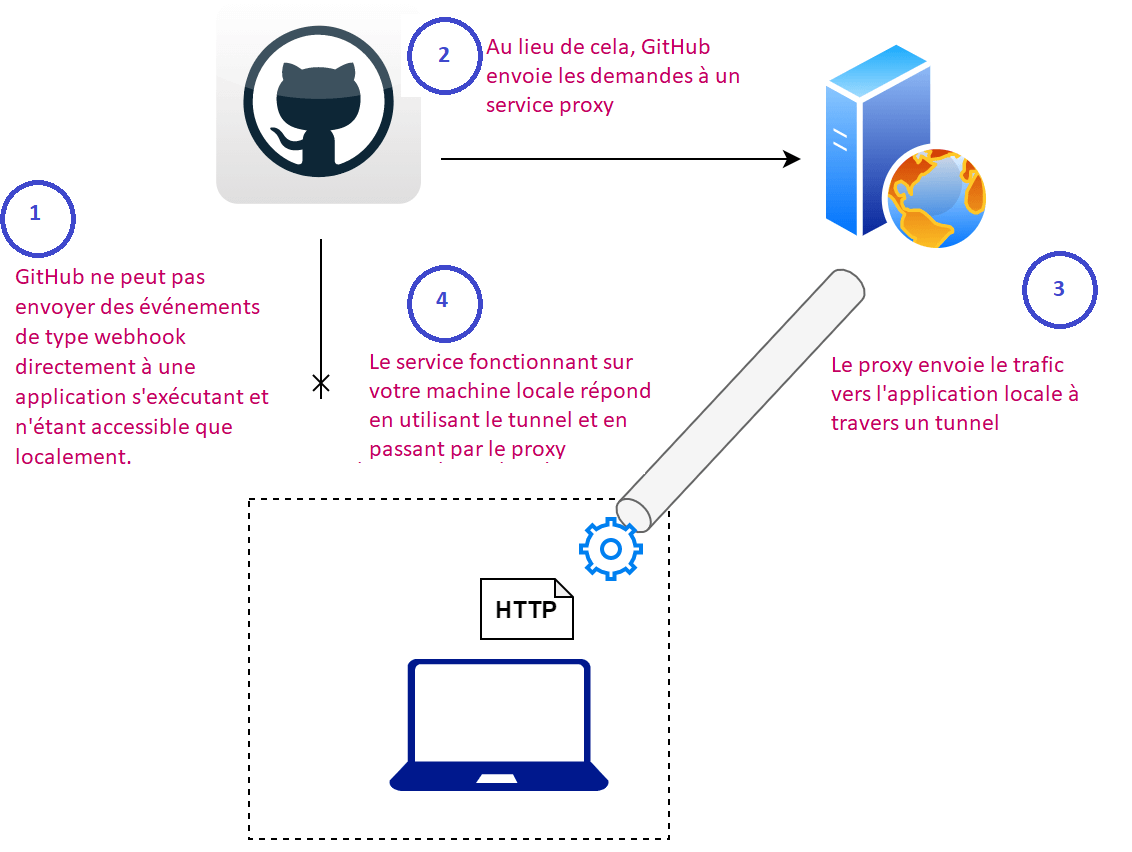
La suite logique est de corriger cette erreur et d’automatiser le fonctionnement pour qu’à chaque mise à jour de GitHub, se lancent les jobs dont le test.

Cela permettra également de travailler en équipe et à distance afin que chaque modification d’un concepteur soit validée par un test automatisé. Nous verrons plus loin la fusion de branche.

L’outil est ici un WebHook ou rappel http : généralement déclenchés par un événement, comme l'envoi de code vers un dépôt ou un commentaire publié sur un blog. Lorsque cet événement se produit, le site source envoie une requête HTTP à l'URL configurée pour le webhook. Ici la cible est Jenkins qui pourra être paramétré pour que ce rappel déclenche une action (ici le build du job BuildAppJob qui lance à sa suite le job TestAppjob).

Problème  ! : Notre Jenkins n’est pas accessible depuis le web et utilise l’adresse privée localhost (127.0.0.1) et l’url locale <http://localhost:8080>

Solution  ! : Nous utiliserons <https://localhost.run/>, un service qui peut être utilisé pour créer un point de terminaison publique qui effectue un tunnel vers un site web s’exécutant localement.



Tâche n°1 : Générer une paire de clefs SSH.

Les clés SSH seront utilisées pour sécuriser la communication entre votre ordinateur local et le serveur distant. Vous pouvez utiliser le client SSH intégré à Ubuntu pour générer une nouvelle paire de clés.

* Générer les clefs
  + Valider le chemin par défaut pour le stockage des clefs
  + Ecraser éventuellement des clefs préexistantes.
  + Utiliser une phrase simple de 3 à 4 lettres

~$ ssh-keygen -t rsa -b 2048

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Tâche n°2 : Créer le tunnel

* Nous utiliserons localhost.run pour créer le tunnel HTTP GitHub. Dans notre cas, notre application Jenkins écoute sur le port 8080, d’où la commande suivante pour tunneliser le trafic HTTP public du port 80 vers localhost sur le port 8080.

~$ ssh -R 80:localhost:8080 localhost.run



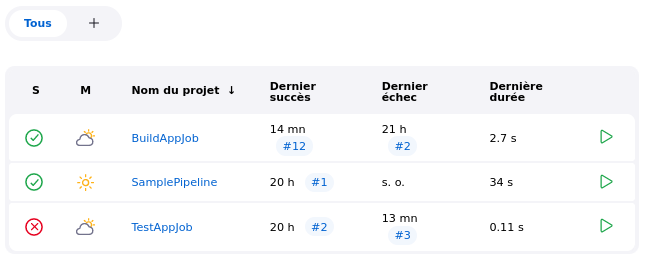
* Le lien à utiliser est surligné en bas (Le vôtre sera différent)  
  <https://e4cce95dcc9c3e>.lhrtunnel.link
* Attention de ne pas utiliser le raccourci Ctrl+C pour le copier, vous quitteriez le processus.

**Etape 4 : Configuration du Webhook (rappel http) sur Jenkins et Git**

Dans Jenkins, nous allons informer l’application qu’un build de BuildAppJob doit être réalisé à chaque modification du dépôt sur GitHub. TestAppJob, quant lui, est paramétré pour se déclencher à la suite de chaque Build de BuildAppJob.

Tâche n°1 : Paramétrer BuildAppJob

* Se rendre sur BuilAppJob





* Puis sur Configurer
* Paramétrer le déclencheur sur BuildAppJob en cliquant sur Ce qui déclenche le build puis en sélectionnant « GitHub hook trigger for GITScm polling »

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* Sauvegarder en cliquant sur le bouton « Save » en bas de page « Save »

Tâche n°2 : Paramétrer GitHub

* Se rendre sur votre dépôt
* Une image contenant texte, Police, capture d’écran, ligne

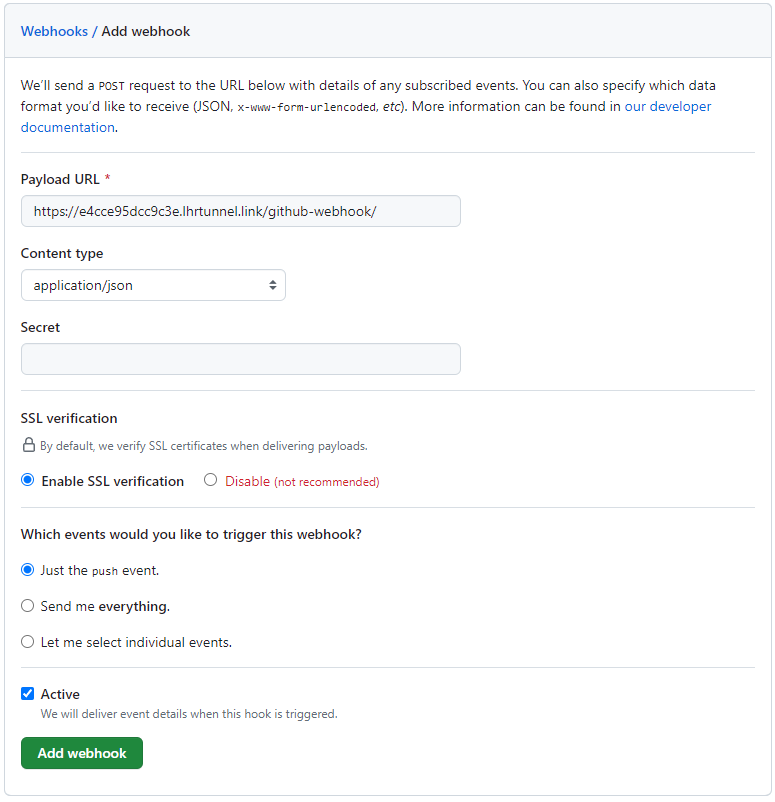
  Description générée automatiquementSélectionner Settings Une image contenant texte, capture d’écran, nombre, Police

  Description générée automatiquement
* Sélectionner Webhooks
* Sélectionner Add WebHook

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

* Dans Payload Url, saisir l’URL généré précédemment suivi de **/github-webhook/**
* Dans Content Type, choisir JSON
* Laisser Secret vide
* Laisser active la vérification SSL
* Laisser le déclenchement sur l’évènement push.
* Valider en cliquant sur AddWebhook
* (Capture d’écran à la page suivante)



* Vous devriez être informé que tout s’est bien passé. Si ce n’est pas le cas, reprendre les étapes une par une.

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

**Etape 5 : remontée de Pousser la modification pour constater l’automatisation**

Pour vérifier le bon fonctionnement du rappel http (Webhook), nous allons paramétrer à nouveau le conteneur pour que Flask fonctionne sur le port 5050 qui est non utilisé. Il n’y aura donc plus de conflit dû à l’utilisation conjointe simultanée du port 8080. TestAppJob devrait réussir.

Puis nous allons pousser dans GitHub nos modifications pour constater que les Jobs dans Jenkins se sont lancéseronter automatiquement lors du push vers GitHub.

Tâche n°1 :

Ouvrir les fichiers d'exemple d'application.

S’assurer que vous êtes toujours dans le répertoire ~/labs/jenkins/sample-app car il s'agit des fichiers associés à votre dépôt GitHub. Ouvrir sample\_app.py et sample-app.sh pour l'édition.

Tâche n°2 : Vérifier S’assurer (dans une nouvelle fenêtre) que conteneur samplerunning a bien été supprimé

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Tâche n°3 : Modifier vos fichiers

* Dans sample\_app.py, changer l'instance du port 8080 à 5050 comme indiqué ci-dessous.

from flask import Flask

from flask import request

from flask import render\_template

sample = Flask(\_\_name\_\_)

@sample.route("/")

def main():

return render\_template("index.html")

if \_\_name\_\_ == "\_\_main\_\_":

sample.run(host="0.0.0.0", port=5050)

* Dans sample-app.sh, changer les trois instances du port 8080 à 5050 comme indiqué ci-dessous.

#!/bin/bash

mkdir tempdir

mkdir tempdir/templates

mkdir tempdir/static

cp sample\_app.py tempdir/.

cp -r templates/\* tempdir/templates/.

cp -r static/\* tempdir/static/.

echo "FROM python" >> tempdir/Dockerfile

echo "RUN pip install flask" >> tempdir/Dockerfile

echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile

echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile

echo "COPY sample\_app.py /home/myapp/" >> tempdir/Dockerfile

echo "EXPOSE 5050" >> tempdir/Dockerfile

echo "CMD python3 /home/myapp/sample\_app.py" >> tempdir/Dockerfile

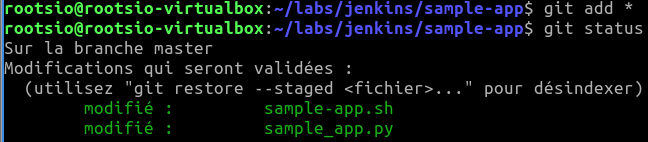
cd tempdir

docker build -t sampleapp.

docker run -t -d -p 5050:5050 --name samplerunning sampleapp

docker ps -a

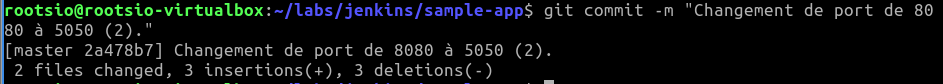
Tâche n°4 : En utilisant les commandes git appropriées, pousser (push) vos modifications vers GitHub

* Entrer les commandes suivantes :

**~/labs/jenkins/sample-app$ git add \***

**~/labs/jenkins/sample-app$ git status**

* Valider le commit.

~/labs/jenkins/sample-app$ git commit -m "Changement de port de 8080 à 5050 (2)." 

* Pousser (push) vos modifications

**~/labs/jenkins/sample-app$ git push origin master**

**Username for 'https://github.com': nom d’utilisateur**

**Password for 'https://AllJohns@github.com': Jeton (Attention !!! A présent GitHub à a**

**adopter adopté le fonctionnement par jeton (Token))**

REMARQUE : Si vous régénérez un jeton, le précédent deviendra obsolète. Il vous faudra donc mettre à jour vos paramètres d’identification à GitHub dans Jenkins :

(Dashboard/BuilAppJob/configure/Source Code Management/Credentials)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

A ce stade, toute modification du dépôt Git entrainera, grâce au rappel http ou WebHook, le déclenchement de BuildAppJob puis de TestAppJob qui se lance après chaque exécution du Job BuildAppJob.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Lab4 : Fusion et création de versions du dépôt Git

Scénario **Etape 1 : Création d’un script qui modifie les fichiers et lance un commit et un push**

**Etape 2 : Vérification de l’exécution du script utilisant Flask**

**Etape 3 : Paramétrage de Jenkins**

**Etape 1 : Création d’un script qui modifie les fichiers et lance un commit et un push**

Tâche n°1 : Analyser et commenter le script ci-dessous.

*Remarque : ghp\_BBepFqb9goht4jDrnVDZ1lsIMpobWT1ye30H est le jeton utilisé sur GitHub.*

|  |
| --- |
| git checkout BrancheDeTest  git config --global user.name "XXXXXXX"  git config --global user.email "XXXXXXXX@XXXXXXX.fr"  ls -l  echo "body {background: pink;}" > ./static/style.css  git add \*  git status    git commit -m "pink1"  git remote add origin3 https://ghp\_BBepFqb9goht4jDrnVDZ1lsIMpobWT1ye30H@github.com/ZakariB/sample-app.git  git push --set-upstream origin3 BrancheDeTest  bash ./sample-app.sh  sleep 10  if curl 172.17.0.1:5050 | grep "172.17.0.1"; then  echo "ok"        git checkout master  git merge BrancheDeTest  exit 0  else  exit 1  fi |

Tâche n°1 : Vérifier que le conteneur samplerunning est bien en cours d’exécution.

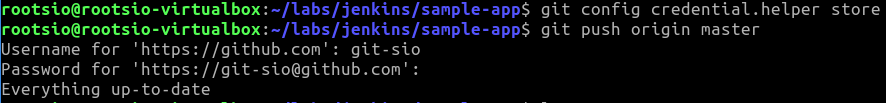
Tâche n°2 : Afficher la page localhost :5050

Une image contenant texte, Police, logiciel, capture d’écran

Description générée automatiquement

Tâche n°3 : Arrêter puis supprimer le conteneur samplerunning et créer une nouvelle branche my-new-branch

Tâche n°4 :Utiliser cette petite astuce pour ne pas avoir à recopier toujours le jeton.



**Etape 3 : Paramétrage de Jenkins**

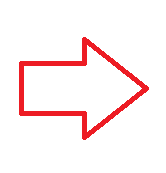
Tâche n°1 : Configurer à nouveau BuilAppJob

Une image contenant texte, Police, nombre, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

****Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Tâche n°2 : Veiller à utiliser le bon jeton puis cliquer sur Avancé…

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Tâche n°3 : Le nommer MonDepot

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Tâche n°4 : Modifier le script à exécuter

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Tâche n°5 : Définir les actions à la suite du build : choisir GitPublisher

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

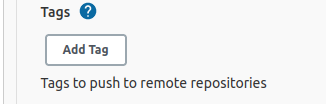
Tâche n°6 : Configurer pour publier uniquement s’il y a un build avec succès.

Une image contenant texte, capture d’écran, Police, blanc

Description générée automatiquement

Tâche n°7 : Ajouter une étiquette

* Cliquer sur Add Tag



* Utiliser $BUILD-ID qui est une variable d’environnement au nom significatif.
* Ajouter un message « Nouveau Build Jenkins »
* Sélectionner la création et la mise à jour de l’étiquette
* Le target est ici le nom précédemment utilisé « MonDepot » (il est normal qu’il ne soit pas reconnu avant la sauvegarde).
* Remarquer au passage la possibilité de créer de nouvelles branches.
* Sauvegarder

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Tâche n°8 : Constater l’absence de tag au niveau du dépôt

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Tâche n°9  : relancer BuildAppJob

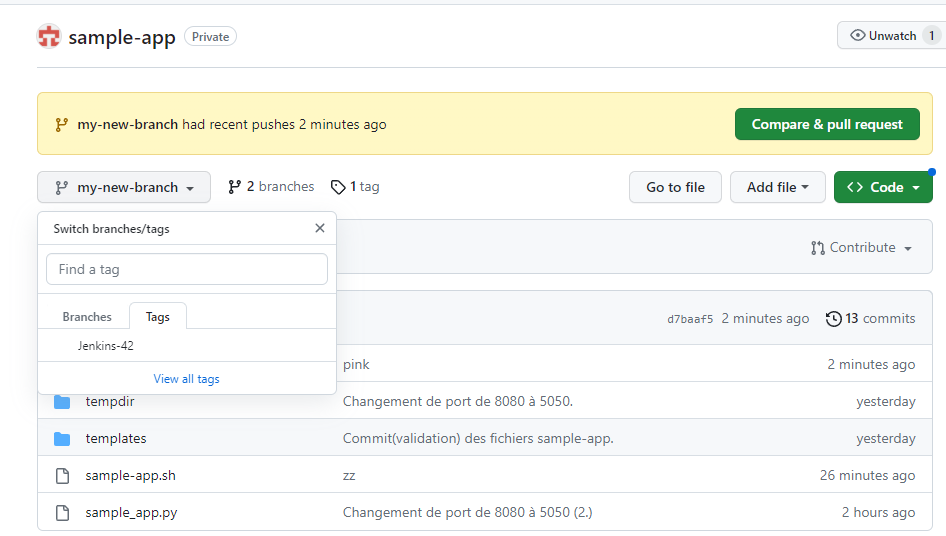
Une image contenant texte, Police, nombre, ligne

Description générée automatiquement

Tâche n°10 : Vérifier que le job BuildAppJob a réussi à vérifier sur GitHub que la nouvelle version a bien été poussée

Une image contenant texte, Police, ligne, nombre

Description générée automatiquement



Tâche n°11 : Tester l’application : la page est bien rose !

Une image contenant texte, Police, logiciel, capture d’écran

Description générée automatiquement

Etape 6 : Pour aller plus loin, utilisation de la fusion automatique

La possibilité d’écrire des scripts dans Jenkins rend le logiciel très puissant. En effet, les scripts s’adaptent très précisément au besoin et peuvent tout réaliser comme une fusion de branche (merge branch). Toutefois, Jenkins automatise un grand nombre d’actions comme l’automatisation d’une fusion après un build.

Défi : A vous de la mettre en œuvre :

Pour vous aider : [Fusion automatique des branches git à l’aide de la | de travail Jenkins par Abhishek Khandelwal | Douleur moyenne (medium.com)](https://medium.com/@abhishekkhandelwal47/auto-merge-git-branches-using-jenkins-job-f5f616f32523)

# Sortie de la console

Started by user [admin](http://localhost:8080/user/admin)

Running as SYSTEM

Building in workspace /var/jenkins\_home/workspace/BuildAppJob

The recommended git tool is: NONE

using credential df354575-09e2-40ea-94b1-2effc3f8c075

> git rev-parse --resolve-git-dir /var/jenkins\_home/workspace/BuildAppJob/.git # timeout=10

Fetching changes from the remote Git repository

> git config remote.MonDepot.url <https://github.com/git-sio/sample-app.git> # timeout=10

Fetching upstream changes from <https://github.com/git-sio/sample-app.git>

> git --version # timeout=10

> git --version # 'git version 2.30.2'

using GIT\_ASKPASS to set credentials

> git fetch --tags --force --progress -- <https://github.com/git-sio/sample-app.git> +refs/heads/\*:refs/remotes/MonDepot/\* # timeout=10

> git rev-parse refs/remotes/MonDepot/master^{commit} # timeout=10

Checking out Revision acd733193b05c8ce9e584bbb71cd6c5e420e70a9 (refs/remotes/MonDepot/master)

> git config core.sparsecheckout # timeout=10

> git checkout -f acd733193b05c8ce9e584bbb71cd6c5e420e70a9 # timeout=10

Commit message: "zz"

> git rev-list --no-walk acd733193b05c8ce9e584bbb71cd6c5e420e70a9 # timeout=10

[BuildAppJob] $ /bin/sh -xe /tmp/jenkins18032767626058742751.sh

+ git checkout my-new-branch

Previous HEAD position was acd7331 zz

Switched to branch 'my-new-branch'

Your branch is up to date with 'origin/my-new-branch'.

+ echo body {background: pink;}

+ git add sample-app.sh sample\_app.py static tempdir templates

+ git status

On branch my-new-branch

Your branch is up to date with 'origin/my-new-branch'.

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: static/style.css

+ git commit -m pink

[my-new-branch cefd8da] pink

1 file changed, 1 insertion(+), 1 deletion(-)

+ git push -u origin my-new-branch

To <https://github.com/git-sio/sample-app.git>

d7baaf5..cefd8da my-new-branch -> my-new-branch

Branch 'my-new-branch' set up to track remote branch 'my-new-branch' from 'origin'.

+ bash ./sample-app.sh

mkdir: cannot create directory ‘tempdir’: File exists

mkdir: cannot create directory ‘tempdir/templates’: File exists

mkdir: cannot create directory ‘tempdir/static’: File exists

DEPRECATED: The legacy builder is deprecated and will be removed in a future release.

Install the buildx component to build images with BuildKit:

<https://docs.docker.com/go/buildx/>

Sending build context to Docker daemon 6.144kB

Step 1/14 : FROM python

---> cd9c1d09c087

Step 2/14 : RUN pip install flask

---> Using cache

---> 998dd2c73585

Step 3/14 : COPY ./static /home/myapp/static/

---> 71ced901f826

Step 4/14 : COPY ./templates /home/myapp/templates/

---> 6428577fd26d

Step 5/14 : COPY sample\_app.py /home/myapp/

---> 6020f40d3c35

Step 6/14 : EXPOSE 5050

---> Running in 90d7f826ce46

Removing intermediate container 90d7f826ce46

---> 58178ee3b4a2

Step 7/14 : CMD python /home/myapp/sample\_app.py

---> Running in 2d139b4b5e01

Removing intermediate container 2d139b4b5e01

---> 9330dcc75943

Step 8/14 : FROM python

---> cd9c1d09c087

Step 9/14 : RUN pip install flask

---> Using cache

---> 998dd2c73585

Step 10/14 : COPY ./static /home/myapp/static/

---> Using cache

---> 71ced901f826

Step 11/14 : COPY ./templates /home/myapp/templates/

---> Using cache

---> 6428577fd26d

Step 12/14 : COPY sample\_app.py /home/myapp/

---> Using cache

---> 6020f40d3c35

Step 13/14 : EXPOSE 5050

---> Using cache

---> 58178ee3b4a2

Step 14/14 : CMD python /home/myapp/sample\_app.py

---> Using cache

---> 9330dcc75943

Successfully built 9330dcc75943

Successfully tagged sampleapp:latest

docker: Error response from daemon: Conflict. The container name "/samplerunning" is already in use by container "28d79902b6ce9428671af2eb8a6016652beb2124eb8443dd936b29fe85233d8d". You have to remove (or rename) that container to be able to reuse that name.

See 'docker run --help'.

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

28d79902b6ce b587a8567724 "/bin/sh -c 'python …" 14 minutes ago Up 14 minutes 0.0.0.0:5050->5050/tcp, :::5050->5050/tcp samplerunning

19de9d5c947a jenkins/jenkins:lts-jdk11 "/usr/bin/tini -- /u…" 5 days ago Up 5 days 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 50000/tcp jenkins\_server

+ sleep 10

+ + grepcurl You are calling me from 172.17.0.1

172.17.0.1:5050

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0

100 178 100 178 0 0 44500 0 --:--:-- --:--:-- --:--:-- 59333

<h1>You are calling me from 172.17.0.1</h1>

+ echo ok

ok

+ exit 0

The recommended git tool is: NONE

using credential df354575-09e2-40ea-94b1-2effc3f8c075

> git tag -l Jenkins-43 # timeout=10

> git tag -a -f -m Nouveau Build Jenkins Jenkins-43 # timeout=10

Pushing tag Jenkins-43 to repo MonDepot

> git --version # timeout=10

> git --version # 'git version 2.30.2'

using GIT\_ASKPASS to set credentials

> git push <https://github.com/git-sio/sample-app.git> Jenkins-43 # timeout=10

Triggering a new build of [TestAppJob](http://localhost:8080/job/TestAppJob/)

Finished: SUCCESS

1. **Grep** est un acronyme qui signifie Global Regular Expression Print. C'est un outil en ligne de commande Linux/Unix utilisé pour rechercher une chaîne de caractères dans un fichier spécifié. Le modèle de recherche de texte est appelé une expression régulière (regex). [↑](#footnote-ref-1)