Mise en production

Nous regardons dans cette partie deux façon de mettre notre programme en production.

La première consiste à utiliser un service d'hébergement spécialisée dans les applications web : *Netlify*. Ce service nous permet de déployer directement notre application à partir de son *build*.

Nous voyons ensuite, comment conteneuriser notre application, de façon à pouvoir la déployer dans des environnements plus génériques qui conviendraient à d'autres types d'application, typiquement un cluster *kubernetes*.

Mise en production avec Netlify

Rendez vous sur le site de *Netlify* (https://www.netlify.com) et créez vous un compte.

Ceci fait, revenez dans votre projet, et dans un terminal installez les outils de Netlify avec la commande :

`npm install netlify-cli --save-dev`

Puis obtenez un token d'accès à Netlify avec la commande `node_modules/netlify-cli/bin/run.js  login`

Enfin réalisez manuellement un déploiement de votre application avec la commande :

`node_modules/netlify-cli/bin/run.js  deploy --prod`

Cette commande va vous proposer plusieurs options. Il faudra choisir de créer un nouveau projet, et lui donner un nom qui n'existe pas déjà chez Netlify.

A la fin votre application est accessible à l'adresse https://nomduprojet.netlify.app/

Par la suite, quand vous voudrez automatiser le déploiement dans le pipeline Jenkins, vous utiliserez la commande :

`node_modules/netlify-cli/bin/run.js  deploy --prod --site nomduprojet.netlify.app`

pour préciser quel site vous voulez déployer.

Ajout du déploiement au pipeline Jenkins

Nous souhaitons que le déploiement que nous avons effectué manuellement soit automatiquement réalisé en cas de push sur la branche principale du projet. Par contre, nous ne voulons pas qu'un nouveau déploiement soit fait quand on pousse sur la branche `dev`.

Pour distinguer les deux branches dans le `Jenkinsfile`, vous pouvez utiliser l'instruction `when` comme ceci par exemple :

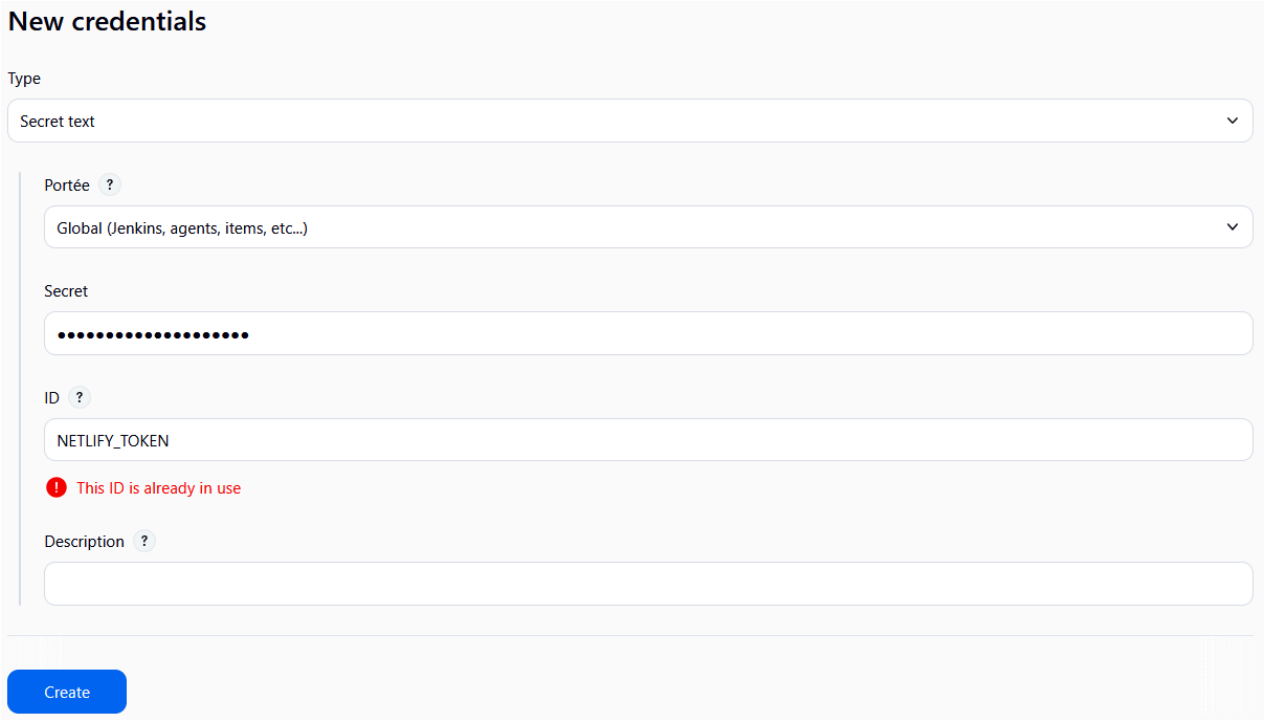
```
when { branch 'master' }
```

Si vous placez cette section juste avant la section `steps`, les étapes ne seront exécutées que pour le build qui concerne la branche principale.

Ensuite, nous devons donner le droit à Jenkins de déployer notre projet sur Netlify. Pour cela :

1. Obtenez un *token* d'accès sur le site de Netlify à l'adresse : https://app.netlify.com/user/applications/personal
2. Enregistrez ce *token* dans les *credentials* de Jenkins à cette adresse : http://localhost:8080/manage/credentials/store/system/domain/_/newCredentials

   Donnez lui un type (Secret Text) et un identifiant (NETLIFY_TOKEN) et coller dans le champ Secret le *token* Netlify obtenu précédemment.



3. Vous utiliserez ce *credential* dans le pipeline dans une section environnement (placée juste avant la section `when`) comme ceci :

```
environment {
   NETLIFY_AUTH_TOKEN = credentials('NETLIFY_TOKEN')
}
```

Avec toutes ces informations, à vous d'ajouter à la fin de votre pipeline, l'étape qui mettra automatiquement en déploiement la dernière version du code de votre projet poussé sur la branche principale.

Vous pouvez également à cette occasion, ajouter un numéro de version que vous ferez dans un premier temps évoluer manuellement à chaque nouvelle fusion dans la branche `master`. Un bon emplacement pour ce numéro est le titre de votre application que vous trouverez dans le fichier `index.html` à la racine de votre projet. Vous pouvez par exemple ajouter le numéro de version dans la balise `<title>` :

```
<title>chessgame v1.1</title>
```

Conteneurisation de l'application

Dans cette partie, nous allons conteneuriser l'application. Cette mise au format *docker* nous permettra de la déployer dans un cadre plus général, et en particulier dans un cluster *kubernetes*.

Bien entendu, c'est le *pipeline* Jenkins qui va réaliser cette contenarisation.

Pour conteneuriser l'application, il faut spécifier la configuration du conteneur. Cela se fait via un fichier `Dockerfile` placé à la racine du projet.

Pour ceux qui n'ont jamais vu *docker*, voici le contenu du `Dockerfile` qui correspond à une application programmée avec *vue.js* :

```
# build stage
FROM node:lts-alpine as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm i
COPY . .
RUN npm run build

# Production stage
FROM nginx:stable-alpine as production-stage
COPY --from=build-stage /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Avant de réaliser la construction du conteneur dans le *pipelin*, vous pouvez tester que le `Dockerfile` est valide en vous mettant à la racin de votre projet et en tapant la commande `docker build -t monapp .` (n'oubliez pas le point à la fin de la commande).

Vous pouvez ensuite exécuter l'application avec la commande `docker run --rm  -p 8181:80 monapp` et connecter votre navigateur sur l'URL `http://localhost:8181` pour voir votre application qui s'exécute.

Pour arréter l'application, repérer l'identifiant du conteneur qui s'éxécute avec la commande `docker ps` puis envoyer un signal de stop au processus repéré avec la commande `docker stop identifiant`

Avant d'automatiser cette construction dans le *pipeline*, il faut aussi identifier un dépôt de conteneur où le déposer. Le plus simple est sans doute de la placer dans le dépot lié à votre projet sur GitHub, GitLab ou autre.

Dans tous les cas, il faut obtenir un *token* avec la permission d'écrire dans le dépôt (*write:packages* pour Github, *write_registry* pour Gitlab). Enregistrez ce *token* dans les *credentials* de Jenkins en l'appelant par exemple `CI_REGISTRY_PASSWORD`.

Une fois ce token enregistré, vous pourrez réaliser un *push* du conteneur vers le dépôt avec la commande *docker push*.

Au final, voici l'étape à insérer dans votre pipeline pour que le conteneur soit construit et poussé vers son dépôt en cas de changement sur la branche principale :

```
    stage('docker') {
             agent any
             when { branch 'master' }
             environment {
                 CI_REGISTRY = 'ghcr.io' // a remplacer par 'registry.gitlab.com' si gitlab
                 CI_REGISTRY_USER = 'nicolassinger' // à remplacer par votre login chez gitlab ou github
                 CI_REGISTRY_IMAGE = "$CI_REGISTRY" + '/' + "$CI_REGISTRY_USER" + '/chess'
                 CI_REGISTRY_PASSWORD = credentials('CI_REGISTRY_PASSWORD')
             }
             steps {
                 sh 'docker build -t --network=host $CI_REGISTRY_IMAGE .'
                 sh 'docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY'
                 sh 'docker push $CI_REGISTRY_IMAGE'
             }
         }
```

Exécutez le *pipeline* et vérifiez que le conteneur a bien été poussé dans son dépôt.

Si c'est bien le cas, votre application est exécutable n'importe où dans le monde à partir du moment où docker est installé avec une commande ressemblant à `docker run -p 8181:80 ghcr.io/nicolassinger/chess:latest`

Nous nous arrêtons là pour ce qui est de ce cours, mais sachez que conteneur est déployable de façon automatisé dans un cluster *kubernetes* ou sur d'autres plateformes de type conteneurs (*docker swarm*, *openshift*, etc.).

Poursuite du développement

En vous plaçant sur la branche GIT de développement, ajoutez au jeu d'échec le respect des règles de déplacement des pièces. Le plus simple pour cela est d'utiliser la librairie javascript `chess.js` (https://github.com/jhlywa/chess.js) qui implémente déjà toutes ces règles.

Voici un prompt qui devrait vous permettre de le faire :

> Actuellement le jeu d'échec permet de déplacer les pièces librement sur l'échiquier. Est-ce que tu peux utiliser la library javascript "chess.js" pour contraindre les déplacements afin qu'ils respectent les règles du jeu d'échec ?

En principe, si vous utilisez l'IA en mode agent, elle devrait implémenter les fonctionnalités demandées en s'aidant de vos tests pour vérifier que le jeu continue de bien fonctionner.

Effectuez un test visuel local et s'il passe, changez la version de votre application et poussez la nouvelle version de la branche *dev* sous git.

Exécutez ensuite le *pipeline* Jenkins de la branche *dev* et vérifiez que tout se passe bien.

Si tout est ok, fusionnez avec GIT la branche principale avec la branche dev, pushez la branche principale, et exécutez le *pipeline* de la branche principale pour mettre la nouvelle version en production.

Conclusion

Le développement devrait se poursuivre ainsi de façon itérative. En principe, vous ne devriez pas avoir besoin de lancer manuellement vos pipeline sur les branches principales et *dev*. En effet quand Jenkins est installé pour pouvoir être contacté publiquement, ces lancements se font de façon automatique en cas de nouveau *push* de code sur ces branches.

On voit que *Jenkins*, par son *pipeline* d'automatisation, permet de fluidifier les tests et les déploiements itératifs de votre application avec pour point de départ le code mis sous GIT. Voilà qui conclu ce cours d'introduction à Jenkins.