TP 8 : Mise en production

Il y a différentes façons pour installer son projet en production :

- Publication automatique via un système d'intégration continue
- Dockeisation et déploiement sur un serveur ou un système d'orchestration
- Publication sur un serveur dédié/virtuel, ...

Ici nous allons voir la plus simple à mettre en place : Docker

Bonnes pratiques

Dans vos projets, le mieux est d'avoir directement un fichier Dockerfile à la racine de votre projet. Cela permet de savoir directement quelle image correspond au projet en cours.

Pour un projet Node.JS le mieux est de partir sur une image alpine existante dans la dernière version LTS en date (il est conseillé d'utiliser les versions LTS en production).

Si vous ne savez pas ce que sont les images alpines, il s'agit d'un système ultralégé et sécurisé utilisé pour avoir des images légères à déployer en production. Par exemple, une image Node.JS Alpine contre une image Node.JS Ubuntu peut varier de 50Mo à 600Mo.

Voici un exemple d'image de base pour un projet HapiJS :

```
FROM mhart/alpine-node:8

ENV NODE_ENV=production
ENV ENVIRONMENT=production
ENV HAPI_DIR=/app/hapi

EXPOSE 8080

WORKDIR ${HAPI_DIR}

ADD . ${HAPI_DIR}

RUN npm install --only=prod -d && \
    npm cache clear --force

# Start it

CMD ["node", "server.js"]
```

Pour revenir en détail sur l'iage, que fasons nous :

1. On indique la variable `NODE_ENV` comme étant en production. Quand elle est initiée en production, `npm-install` ne fait que ce qui est noté dans `dependencies` et ignore tout de la partie `devDependencies`.
2. On initnie l'environnement de notre projet à `production`. Il est conseillé de bien séparer la notion d'environnement Node.JS par rapport au projet que l'on veut lancer.
3. On indique que le directory contenant le projet sera dans `/app/hapi`
4. On expose le port 8080.
5. On dit de se placer dans le répertoire HAPI_DIR
6. On copie tout le contenu de notre projet dans HAPI_DIR sur l'image Docker.

7. On installe les packages NPM **et** on supprime le cache afin de ne pas avoir de l'espace disque inutile dans l'image finale.
8. On donne la commande de démarrage de l'image. Ici `node server.js`

> Pour que le point 6 fonctionne correctement, il est nécessaire d'avoir un fichier .dockerignore qui dit d'ignorer, à minima, le répertoire `node_modules` et toute la partie git

```
# git
.gitignore
.git/

node_modules
```

Publication

Ensuite, comme chaque image docker vous devrez la builder (et la publier si nécessaire) :

```
docker build -t votrenom/tpnode:untag .

# si nécessaire
docker publish votrenom/tpnode:untag
```

Utilisation

Le plus simple si vous voulez tester cette image et avoir une liaison avec votre MongoDB reste de passer par `docker-compose`.Par exemple, voilà ce que pourrait donner un `docker-compose.yml` avec l'image faite au dessus :

```
version '3'
services:
  mongo:
    image: mongo
  hapi:
    image: votrenom/tpnode:untag
    ports:
      - "8080:8080"
    environment:
      - ENVIRONMENT=production
    links:
      - mongo
```

> J'ai volontairement ommis de mettre toutes les variables. À vous de les rajouter dans votre propre fichier.

Normalement si vous exécutez ce projet, vous devriez avoir votre serveur Hapi et mongoDB qui se lancent et communiquent l'un avec l'autre.

Bien entendu, ce fichier n'est pas propre à aller en production. Il faudrait gérer des volumes pour la partie MongoDB pour que, si on coupe docker, les données ne soient pas perdues.

Conclusion

J'espère que ce module de Node.JS vous a plus, que vous avez découverts d'autres façons de développer et de voir ce que Hapi est capable de faire.

Pour rappel, le devoir est à rendre avant le dimanche 25 Février 20h. Tout commit/push effectué après cette heure sera ignoré. Si jamais vous partagez le lien d'un repo privé, merci de m'en donner les accès.

Pour rappel, envoyez, avec votre nm, le lien vers le repository GIT à l'adresse suivante : benjamin.besse02@gmail.com