Dockerizé un serveur web simple

Enoncé

1. Développez un serveur HTTP qui expose le endpoint */ping* sur le port 80 et répond par PONG

2. Créez le fichier Dockerfile qui servira à construire l'image de l'application. Ce fichier devra décrire les actions suivantes

- image de base: alpine:3.8
- installation du runtime du langage choisi
- installation des dépendances de l'application
- copie du code applicatif
- exposition du port d'écoute de l'application
- spécification de la commande à exécuter pour lancer le serveur

3. Construire l'image en la taguant *pong:v1.0*

4. Lancez un container basé sur cette image en publiant le port 80 sur le port 8080 de la machine hôte

5. Tester l'application

Correction

1. Dans cette exemple nous avons choisi Node.js

Le code suivant est le contenu du fichier *pong.js* (code du serveur web)

```
var express = require('express');
var app = express();
app.get('/ping', function(req, res) {
    console.log("received");
    res.setHeader('Content-Type', 'text/plain');
    res.end("PONG");
});
app.listen(80);
```

Le fichier *package.json* contient les dépenances de l'application (dans le cas présent, il s'agit de la librairie *expressjs* utilisée pour la réalisation d'application web dans le monde NodeJs.

```
{
  "name": "pong",
  "version": "0.0.1",
  "main": "pong.js",
  "scripts": {
    "start": "node pong.js"
  },
   "dependencies": {  "express": "^4.14.0" }
}
```

2. Une version du Dockerfile pouvant être utilisé pour créer une image de l'application

```
FROM node:10.15-alpine
COPY . /app/
RUN cd /app && npm install
WORKDIR /app
EXPOSE 80
CMD ["npm", "start"]
```

Note: il y a toujours plusieurs approches pour définir le fichier *Dockerfile* d'une application. On aurait par exemple pu partir d'une image de base comme *ubuntu* ou *alpine*, et installer le runtime *nodejs* comme dans l'exemple ci-dessous.

```
FROM alpine:3.8
RUN apk update && apk add nodejs
COPY . /app
RUN cd /app && npm install
WORKDIR /app
EXPOSE 80
CMD ["npm", "start"]
```

3. La commande suivante permet de construire l'image à partir du *Dockerfile* précédent

```
$ docker image build -t pong:1.0 .
Sending build context to Docker daemon  7.168kB
Step 1/6 : FROM node:10.15-alpine
 ---> 288d2f688643
Step 2/6 : COPY . /app/
 ---> dbc76081fd62
Step 3/6 : RUN cd /app && npm install
```

```
 ---> Running in 82048b4eee68
Removing intermediate container 82048b4eee68
 ---> 68a5ed5f0bbd
Step 4/6 : WORKDIR /app
 ---> Running in 126a6062e5bc
Removing intermediate container 126a6062e5bc
 ---> db01c43f76dc
Step 5/6 : EXPOSE 80
 ---> Running in 01ffbd9641f7
Removing intermediate container 01ffbd9641f7
 ---> b1ffff2cf6dc
Step 6/6 : CMD ["npm", "start"]
 ---> Running in db0bec23d302
Removing intermediate container db0bec23d302
 ---> 17f2f66a6a20
Successfully built 17f2f66a6a20
Successfully tagged pong:1.0
```

4. La commande suivante permet de lancer un container basé sur l'image *pong:1.0* et le rend accessible depuis le port 8080 de la machine hôte.

```
$ docker container run -d -p 8080:80 pong:1.0
a42b30cf27490d73d795103c6b28285dd01785933b82ee6404121c7cd743cb5b
```

Note: assurez-vous que le port n'est pas déjà pris par un autre container ou une autre application. Si c'est le cas, utilisez par exemple le port 8081 dans la commande ci-dessus.

5. Afin de tester le serveur ping, il suffit d'envoyer une requête GET sur le endpoint /ping et vérifier que l'on a bien PONG en retour

```
$ curl localhost:8080/ping
PONG
```