CR TP2

Lors de la création du premier dockerfile, des problèmes sont survenus sur les droits de l’utilisateur. D’où le fait d’ajouter, lors de la commande COPY, le flag —chown=node:node

# 4.

-p permet de faire une redirection de port : en faisant –p 8123:8000, en dehors du docker, on se connecte au port 8000 et on est redirigé vers le port 8123 du docker, qui sera écouté par le serveur.

-m permet de préciser la mémoire qu’on accorde au docker

—cpus permet de préciser le nombre de cpu qu’on accorde au docker

Faire varier leur valeur aura forcément un impact sur les performances du docker mais pas sur ce que api/v1/sysinfo renverra car sysinfo récupère les infos du noyau qui est entièrement partagé entre le pc et les dockers.

On peut maintenant s’interroger sur la méthode de virtualisation qui permet à docker de partager le noyau avec le PC en ayant finalement des ressources limitées :

<https://blog.lizzie.io/linux-containers-in-500-loc.html> explique quels mécanismes sont utilisés pour la virtualisation de Docker en C.

* Namespaces qui permet de regrouper les objets du noyau. Par exemple avec le namespace des pid : tous les pid concernant le docker sont regroupés sous un seul namespace. Finalement dans le Docker, le premier process est pid=1 alors que pour le PC, cela va être un tout autre pid.
* Capabilities qui donne les limites de ce que le process 0 peut faire
* Cgroups qui permet de limiter les ressources cpu mémoire etc... Par notre docker
* Setrlimit qui fait un peu plus de choses que cgroups mais qui est plus vieux.

Alors on comprend que le docker a l’illusion d’être un système à part entière alors qu’il occupe simplement les ressources du système mère comme n’importe quel process. Il n’y a pas d’émulation d’un système (contrairement aux machines virtuelles).

# 5.

En regardant avec dive, on se rend compte que l’image que l’on a créé est lourde : 206 MB, et dive nous dit en plus que au moins 14 MB ne servent pas.

# 6.

On peut se rendre compte que finalement, une fois l’application construite on n’a pas besoin de npm et devDependencies. DevDependencies qui ne sert que pour le développement, et npm ne sert plus non plus car en prod, il ne sert qu’à lancer npm run start, qui est finalement la commande node dist/index.js.

Alors, dans la première image, on construit l’image lourde dans laquelle on va exécuter npm run build, et récupérer le dossier dist et node\_modules.

Une fois construite, cette image ne fait plus que 66.8 MB, 3 fois plus légère que la précédente.

Les deux conséquences que cela peut avoir : l’image est plus simple à déployer et elle occupera moins de ressources disque (donc possiblement moins cher).

# 8.

Docker pull