Reproducibilité des environnements

Ryan Lahfa

30 juin 2020

Docker, la reproducibilité, pourquoi et pourquoi pas ?

La reproducibilité

La reproducibilité c'est un concept clef en tant que DevOps, car elle vous assure de vous tirer du:

(a) Dependency hell des projets (NPM/Yarn, PyPI/Poetry/Pipfile, Maven, Go, Cargo, la liste ne s'arrête jamais.)
(b) La documentation imprécise, car la documentation devient le "code"
(c) Le code qui pourrit, vous revenez six mois plus tard et une mise à jour casse tout votre code
(d) Les barrières d'adoptions ou de réutilisation: personne ne veut contribuer à un projet compliqué à installer chez soi

Reproducibilité forte

Est ce que votre distribution Linux est reproducible ? Cherchez !

- C'est le cas de Debian (pas intégralement): https://wiki.debian.org/ReproducibleBuilds
- C'est le cas d'Arch Linux (non plus): https://wiki.archlinux.org/index.php/Reproducible_Builds
- C'est le cas de NixOS (à 96,53 % sur l'image minimale à ce jour!): https://r13y.com/
- Et sûrement d'autres.

Qu'appelle-t-on reproducibilité forte ?

Est-ce que si vous changez le système de fichier, vos paquets produisent les mêmes octets? Est-ce que si vous changez les ordres des résultats de commandes comme `ls`, cela reste vrai ? Pour mieux comprendre, https://salsa.debian.org/reproducible-builds/disorderfs

Cette reproducibilité est assurée que lorsque vous mettez les moyens, des cgroups [1], occulter des chemins d'accès classiques, s'assurer qu'après non-déterministe, les paquets restent eux les mêmes.

1. Mécanisme derrière Docker pour enfermer les processus dans un contexte différent.

Reproducibilité faible

La reproducibilité faible c'est plutôt celle qui, à OS fixé, à système de fichier fixé, à "configuration" fixé, produit le même paquet sur votre machine. Elle ne transfère pas de machine à machine car elle dépend de paramètres comme la date, l'heure, le réseau ou des choses qui sont éphèmères mais pas méchant.

Docker est faiblement reproducible puisqu'il enferme une forme d'image qui contient des mappings UIDs "hardcodés", mais voyons encore plus d'exemples.

Dockerfile

```
# DO NOT USE THIS DOCKERFILE AS AN EXAMPLE, IT IS BROKEN
FROM python:3

COPY yourscript.py /

RUN pip install flask

CMD [ "python", "./yourscript.py" ]
```

Ce n'est pas reproducible, puisque la version de Python varie. Celle de Flask aussi.

Mieux

```
FROM python:3.7.3-stretch
COPY requirements.txt /tmp/
RUN pip install -r /tmp/requirements.txt
RUN useradd --create-home appuser
WORKDIR /home/appuser
USER appuser
COPY yourscript.py .
CMD [ "python", "./yourscript.py" ]
```

C'est mieux, mais l'utilisateur est crée avec quelle UID ? Quizz surprise.

Docker

La liste est longue.

- https://github.com/moby/moby/issues/37182
- https://github.com/moby/moby/issues/41136
- https://github.com/moby/moby/issues/41121

Et on peut en rajouter, encore des couches et des couches. Entre les problèmes de noyau, les problèmes de système hôte, le cross-platform, les conteneurs de piètre qualité, le multi-processus dans un conteneur.

Question (réflexion) pour vous: pourquoi utiliser Docker plutôt qu'un gros binaire statiquement lié et une technologie de sandbox comme firejails, alors que c'est plus simple?

Pourquoi Docker c'est si intéressant ?

On dit souvent que Docker est simple. C'est faux.

- Qui sait comment marche Docker ?
- Qui sait quel système de fichier est utilisé et comment celui ci se comporte sur votre machine ?
- Qui sait quelle surface d'attaque vous exposez avec Docker ?
- Qui sait écrire des Dockerfile simples, performants, dans l'esprit de Docker qui ne prennent pas 50GB de votre espace disque ?

Pourquoi Docker c'est pas si intéressant ?

Docker est avant tout une pièce du puzzle dans le travail d'un DevOps, c'est souvent la pièce, mettre son code dans un truc et le donner à une équipe ou l'envoyer sur un cluster.

Mais vous déplacez le problème en faisant cela, au lieu de maintenir des scripts Bash ou des playbooks de déploiement, vous maintenez un déploiement de cluster Docker et des Dockerfile et des ACL et des, et des, et des.

Coût au niveau de l'organisation de Docker ?

Docker a un coût organisationnel.

Vous sacrifiez une vague simplicité apparente **au départ** (i.e. utile pour des startups, qui ont besoin de faire face très vite à du traffic) pour des coûts de gestion (e.g. utiliser EKS, des clusters déjà préfaits des clouds).

En faisant cela, la question est: pour quel travail concret vous êtes payés ? Pour expliquer à l'entreprise qu'il faut payer EKS et vous payer pour écrire des Dockerfile difficilement maintenable et créer de la complexité apparente ?

Bienvenue aux OS magiques: CoreOS, RancherOS

OK, y a des OS qui résolvent tout, c'est vrai. CoreOS, RancherOS, et plein d'autres entreprises se sont lancés dans la RedHat-ification de Docker pour le rendre plus business-friendly.

Tout est inclus dedans et ça marche du tonnerre.

Adieu aux OS magiques

Or, RancherOS est mort:
https://github.com/rancher/os/issues/3000

Or, CoreOS est mort (février 2020): https://www.reddit.com/r/devops/comments/ezpbvj/coreos_end_of_life/ (premier commentaire, lisez le.)

Or, rkt est mort aussi, je ne mets pas de lien, vous trouverez facilement.

Quelle solutions réalistes nous reste-t-il ? Peu.

Bonjour à Kubernetes

Bienvenue à Kubernetes, le monde de Google. Une solution d'orchestration avec 100 000 pièces mouvantes, des versions qui cassent des anciennes fonctionnalités, des problèmes de sécurité, des bugs, des difficultés.

Mais, c'est magique ! Vous envoyez des conteneurs, ça scale tout seul, ça utilise le DNS tout seul, c'est vraiment une expérience très agréable.

Après...

- Qui sait déployer Ceph ? Même le CERN a des gens spécialisés pour le faire.
- Qui sait lire les logs dans un Kubernetes ? Ah oui, mettre en place un mécanisme de collection de logs...
- Qui sait débugger et attacher un bon vieux gdb sur Kubernetes ?
- C'est quoi votre plan si tout votre cluster Kubernetes crash ? Vous faites quoi ?
- Les backups ?

Tout le monde expliquera que Kubernetes est difficile, et n'est pas nécessairement fait pour tout le monde, pourtant tout le monde en veut et tout le monde en a, le problème c'est le coût. À qui profite une telle complexité ? Bien évidemment, les clouds publiques, les ISP, les (re)vendeurs et même moi qui fait ce cours...

Bonjour mini-Kubernetes

OK, il y a des Kubernetes tout petit avec l'agilité de Kube sans la souffrance de Kube, découvrez: https://k3s.io/

Intéressant, mais, à nouveau, c'est spécifique et de niche.
Cependant, c'est bon pour démarrer et se déplacer vers une nouvelle chose après.

Coucou Dokku

Mieux, dans 99 % des cas, une boîte veut se déplacer vite avec une équipe jeune en DevOps, alors dans ce cas là, on peut faire du bien avec Docker en utilisant http://dokku.viewdocs.io/dokku/

Essayez le, c'est simple, et c'est comme Heroku à la maison !

Enfin, où va-t-on ?

Pourquoi je critique autant Docker ?

- https://news.ycombinator.com/item?id=19763413
- https://news.ycombinator.com/item?id=19255603
- https://lemire.me/blog/2020/06/19/computational-overhead-due-to-docker-under-macos/
- https://news.ycombinator.com/item?id=22158176
- https://news.ycombinator.com/item?id=21321601
- https://news.ycombinator.com/item?id=20542915
- https://news.ycombinator.com/item?id=20499397

Conclusion temporaire

Bref, la liste est sans fin. Mais Docker n'est pas que le mal, c'est un outil intéressant, qui a sa place, mais qu'il faut utiliser avec parcimonie à nouveau. Ne succombez pas à la hype sinon vous serez à la fin de la Gartner hype, avec une compétence ultra niche qui sera inutilisable pour votre prochain poste. Et ça, c'est pas drôle, et c'est l'histoire qui le raconte.

Nix et Docker

En attendant, vous pouvez tout à fait fabriquer des conteneurs Docker depuis Nix, dans le TP que nous allons faire avec `3_docker`.

- Un exemple bidon
- Un exemple de Redis
- Comparez avec l'image du Docker Hub :) (pas de triche, pas d'Alpine sinon vous devez m'expliquer comment une build Alpine marche et quelles sont les risques des Docker alpine et pourquoi il ne faut pas les utiliser :D)

Nix et Docker

```
> du -hs /nix/store/sd21my0zyaz74f9f8mnf4cj8n7bs9xga-docker
17M /nix/store/sd21my0zyaz74f9f8mnf4cj8n7bs9xga-docker-imag
```

Bon, clairement, y a pas photo, j'ai désinstallé aucun paquet, vous remarquerez, Nix **calcule** exactement ce qu'il faut.

On peut aller plus loin: https://grahamc.com/blog/nix-and-layered-docker-images et optimiser les caches de Docker avec Nix "intelligemment" (avec de la théorie des graphes très élémentaire).

Quizz surprise: combien de layers maximum Docker peut avoir ? Et à votre avis, Nix ?

NixOS et Docker

En attendant, comment utilise-t-on NixOS et Docker de façon intéressante ?

Vous pouvez gérer vos Docker depuis NixOS aussi, en important `3_docker/nixos-exemple.nix` dans votre `configuration.nix`.

Exercice : Mettre BBB dans un fichier NixOS pour Docker: https://github.com/bigbluebutton/docker

Enfin, Docker vraiment ?

À Google, on s'amuse à allier le meilleur des deux mondes: https://github.com/google/nixery

Plus sérieusement, ne faites pas confiance aveuglément à Docker, ça reste un outil, qui a sa place et son mérite dans des situations précises, ne foncez pas sur un Docker, ne cédez pas à la hype. N'importe qui peut écrire un Dockerfile et faire du Docker, tout le monde ne prend pas nécessairement le temps de réfléchir à la surface, au coût, et à l'intérêt ou non de s'engager sur une stack dockerisée, ne faites pas cette erreur.