Docker - Didapro 7 - DidaSTIC

Adrian Kosmaczewski

2018-02-06

Bonjour, et bienvenus a cet atelier a propos de Docker chez Didapro.

Je m'appelle Adrian Kosmaczewski, et je travaille comme développeur d'applications mobiles et aussi comme enseignant de programmation dans le privé. J'ai aussi publié 5 livres sur le sujet de la programmation mobile et je parle regulierement dans des conferences partout dans le monde.

J'ai prevu une session de questions et reponses a la fin de cette session, donc je vous prie de garder vos questions pour la fin. Cette presentation et tout le materiel que je vous montre sera disponible gratuitement sur Github, pour que vous puissiez suivre cette presentation sans devoir prendre des notes.

Le probleme

Bien plus important, depuis 2003 je donne regulierement des cours et des ateliers a propos de la programmation; j'ai enseigne Java, C#, JavaScript, Objective-C, Swift, Kotlin. Depuis toujours il y a un probleme qui revient chaque fois que je commence un nouveau cours: faire en sorte que tous les etudiantes et etudiants aient la meme configuration de systeme d'exploitation, langages de programmation et librairies pour suivre ce que je fais sur l'ecran de mon portable.

Generalement mes cours sont au format "workshop" ou les 10 ou 12 etudiant(e)s que je recoit suivent en direct les manipulations que je fais sur l'ecran, et ensuite je m'assure que le code "compile et tourne" sur chacune des machines. Ensuite je donne les elements de theorie qui correspondent, et je repete cette boucle quelques fois pendant deux ou trois jours.

Il est donc tres important pour moi que tout le monde puisse suivre, et cela veut dire imperativement que tout le monde ait les memes outils installes de la meme facon sur leurs portables. Pendant des années j'ai cherche a resoudre ce probleme de plusieurs facons, en utilisant des outils comme:

- Des instructions precises via e-mail quelques jours auparavant de la session; le probleme de cette approche c'est que les gens ne lisent pas toujours ce qui y est ecrit. Donc, il faut mettre de cote un peu de temps, le "lundi matin" du cours, pour etre sur que tout le monde a le setup de base pour travailler, et que tout le monde pourra y participer.

- Telecharger une machine virtuelle, par exemple pour VirtualBox; le probleme avec cette approche c'est la question des licenses, car on ne peut pas donner comme ca une copie de macOS ou Windows; aussi il y a le probleme de la taille des fichiers, qui peut être conséquente, et aussi le fait de mettre à jour ces machines virtuelles, qui est un peu difficile.

Aussi, il faut dire que le monde du developpement logiciel d'aujourd'hui est vraiment complexe; par exemple, pour enseigner le developpement web avec un stack comme "MEAN" (Mongo, Express.js, Angular et Node) sur des portables Mac (ce qui n'est pas deraisonnable ces jours-ci), il faut installer, dans l'ordre:

1. Homebrew
2. Node & npm
3. Mongo
4. Express
5. Angular
6. Des lib de testing, comme Jasmine, Karma ou Mocha
7. Un editeur de texte, par exemple WebStorm ou Visual Studio Code
8. Un terminal comme iTerm2 pour y etre plus confortable, avec d'autres outils comme htop, tmux, etc

Et on peut y ajouter des couches! Par example si ensuite vous voulez montrer comment faire des applications mobiles sur le stack MEAN vous devriez y ajouter su ionic, et on pourrait encore continuer.

Le probleme, je suis sur, ne vous est pas etrange, et dans mon cas je n'ai que 12 etudiant(e)s! Je ne voudrais pas m'imaginer ce que ca donne avec une classe de 30 personnes.

Idealement, il me fallait une plateforme avec les caracteristiques suivantes:

- Legere a installer et distribuer a mes eleves
- Rapide a demarrer et eteindre
- Libre de droits
- Cross-platform Windows, Linux et Mac
- Facile a mettre a jour pour les prochaines sessions de cours
- Eventuellement, facile a passer a mes collegues enseignant(e)s aussi si besoin etait.

Une solution: Docker

Depuis peu j'utilise Docker pour resoudre ce probleme; Docker est une technologie relativement recente, tres a la mode (et avec raison!) dans le monde des startups.

Pour resumer ce qu'est Docker, c'est un systeme de machines virtuelles legeres, qui se base sur certaines proprietes et librairies du noyau Linux. Les machines virtuelles Docker ne sont pas completement isolees de leur hote, et peuvent partager des resources comme le kernel ou des librairies. Pour cette raison,

elles demarrent presque instantanément et n'ont besoin que d'une fraction des resources requises par une machine virtuelle "traditionnelle" comme VirtualBox ou VMWare.

En plus, Docker et 100% gratuit et libre, et tourne aussi sur Mac et Windows. La compagnie derriere Docker fait de l'argent avec du support pour entreprises et des produits "pro" dont je ne connais pas les caracteristiques, ne les ayant jamais utilises dans mon travail.

Images et Containers

Pour comprendre Docker, il faut connaitre la terminologie qui y est associée: **les Images, la Registry et les Containers.**

Les machines virtuelles Docker sont crées et distribuées dans la forme de *Images*. Une image est la representation en disque d'une machine virtuelle Docker, de la meme facon qu'une image de disque "ISO" represente les contenus d'un CD de musique ou un DVD.

Pour creer une image, il faut utiliser un fichier au format particulier appele un `Dockerfile`. On va apprendre plus sur ce fichier dans quelques instants. Il faut ensuite faire tourner une commande Docker sur un terminal, `docker build` en y passant le chemin d'acces d'un Dockerfile.

Lorsque l'on cree une image, elle est ajoutee au *Registry* local. C'est le deuxieme terme important de Docker. Une Registry c'est un repository d'images. On peut utiliser une registry pour partager des images avec d'autres utilisateurs. Docker offre gratuitement une image Docker (bien sur!) qui peut etre utilisee pour faire une Registry locale dans votre institution, ecole ou college. Avec une registry, vous pouvez reutiliser des images et en creer des nouvelles qui se basent sur celles deja existantes.

Lorsque l'on demarre une image Docker, on obtient un *Container*, qui est une entite en cours d'execution de cette image. C'est ici que vous aurez vos propres services, applications, et librairies pretes a l'emploi. On peut lancer plusieurs containers separes a partir de la meme image; en fait, la seule limite pour cela est la quantite de memoire vive disponible.

Differences entre VM et Docker

Pour comprendre la difference entre une VM "classique" (VMWare, Parallels ou VirtualBox) et Docker, les graphiques suivants peuvent aider.

Les principales differences entre les deux peuvent etre resumees ainsi:

- Chaque instance d'une VM doit charger tout un systeme d'exploitation complet, ce qui peut etre relativement lourd avec des systemes avec interface graphique. Les containers Docker, au contraire, ne chargent qu'une

partie du systeme Linux, utilisant le kernel du hôte pour la plupart des taches.
- L'isolation d'une VM et complete; les applications y tournent sans avoir, a priori, aucune connexion avec le hôte, hormis des connexions reseau et des drivers video ou peripherique. Les containers Docker, eux, ne sont pas completement isoles, et peuvent avoir un acces plus large a la machine hote.
- Le temps de demarrage d'une VM est generalement long; il faut compter quelques minutes parfois. Un container Docker ne prends que quelques millisecondes a demarrer, et les applications peuvent y tourner plus vite.
- Les containers Docker ne peuvent pas "emuler" d'autres architectures logicielles; c'est a dire que l'on ne peut pas avoir des containers Docker qui tournent du code x64 sur un hôte avec une CPU de type ARM.

Creation d'images

Comme mentionné precedemment, pour créer une image Docker, on va devoir creer un fichier appele `Dockerfile`. Ces fichiers comportent des instructions tres simples a lire et a comprendre, qui vont faire des changements precis, couche par couche, a une machine de base, qui peut ne pas etre disponible sur le hote en question.

La premiere commande d'un `Dockerfile` est la commande `FROM` qui specifie l'image de base qui va etre utilisee pour construire une nouvelle image. Pour resoudre le probleme de la poule et les oeufs, Docker propose une image appellee `scratch` qui est un noyau Linux minuscule avec le minimum necessaire pour faire tourner une image.

Une image de base tres habituelle est `ubuntu:16.04`, dont le nom décrit bien les contenus. Tres populaire aussi, la distribution Arch Linux est souvent utilisee comme image de base, grace a sa capacite de mettre a jour automatiquement les packages ajoutes, ce qui en fait un choix interessant pour services connectes en permanence.

Apres on y ajoute une commande `LABEL` qui a pour but d'ajouter des metadonnees qui seront utilisees para la Registry pour indexer et chercher des images, et qui permettent a d'autres utilisateurs de comprendre ce que fait votre image.

Ensuite, on utilise des commandes comme `RUN` pour faire tourner des programmes dans l'image; typiquement la premiere commande est toujours celle requise pour mettre a jour le gestionnaire de packages de votre distribution Linux:

RUN apt-get update RUN apt-get install -y build-essential

Ensuite il y a d'autres commandes, comme `COPY` pour pouvoir y ajouter des fichiers arbitraires dans l'image, ou `WORKDIR` pour changer de directoire a l'interieur de l'image.

Tout cela permet de creer des images assez complexes, avec meme des utilisateurs et une grande variete de variables d'environnement pretes a l'emploi.

Demo

On va voir maintenant un petit movie (fait avec Asciinema) qui montre les etapes et les mutations requises pour faire une petite image Docker.

Imaginons que vous devez enseigner la programmation en C. Oui, je sais, c'est pas forcement votre tasse de the mais c'est un exemple tres simple pour vous montrer comment faire.

L'idee ce sera de creer une image Docker avec les outils de base de developpement en C, pour compiler une petite application en C, le desormais classique "Hello World".

On va donc se baser sur Ubuntu 16.04, et on va y mettre a jour le systeme de gestion de packages. Ensuite on y installera les outils de developpement C de base, et finalement on va y creer un folder appele "build". Dans ce folder on y copiera notre code source, et un Makefile pour nous faire la vie plus simple.

Ensuite, on tape la commande pour faire construire notre image:

```
docker build --tag akosma/helloworld .
```

Le point a la fin de la command fait reference au folder courant, qui doit contenir imperativement un `Dockerfile`. Le `tag` qui est passe dans la commande ci-dessus donne un nom a l'image, pour qu'elle soit plus facile a reperer entre tant d'autres.

Une fois qu'on a construit notre image, on la retrouve facilement grace a la commande

```
docker images
```

qui montre la liste des images disponibles dans la registry locale. On y voit notamment deux images, celle que l'on vient de creer et aussi l'image `ubuntu` qui reste copie en locale, de telle facon a ce que les prochaines constructions en base a celle-ci se fassent plus rapidement.

On peut creer un container a partir d'une image de facon tres simple:

```
docker run --tty --interactive akosma/helloworld bash
```

Les parametres `--tty --interactive` apparaissent tres souvent dans la litterature comme simplement `-it`, et donnent l'instruction a Docker de ne pas seulement faire tourner le container en memoire, mais aussi d'attendre nos commandes et de montrer les resultats de facon interactive. La commande qu'on passe a la fin, `bash` indique le terminal de ligne de commande que l'on veut utiliser pour notre session interactive.

On peut aussi passer un parametre `--name` pour donner un nom sympa a notre container.

On peut se promener maintenant a l'interieur de notre container, qui a exactement la configuration que l'on veut: un Ubuntu 16.04, avec les outils de developpement du langage C, et un folder `/build` avec un `Makefile` et notre fichier `hello.c`.

On peut faire un `ls ..` et voir le contenu du systeme de fichiers; il y a tout ce que l'on attends d'une distribution Linux, en ce cas precis, Ubuntu.

Ensuite on tape `make` et on voit, avec `ls` que l'on a cree un executable a partir de notre fichier `hello.c`; on peut faire tourner ce fichier comme n'importe quel autre programme.

Je peux maintenant me "detacher" de ce container, en tapant la combinaison de touches `CTRL + P` et `CTRL + Q` a la suite. Je peux voir maintenant les containers qui tournent:

```
docker container list
```

Cette commande est la meme que `docker ps`. On y voit un "nom" a la fin de l'output, qui est un nom aleatoire que Docker donne a chaque container. Il y a aussi un ID alphanumerique plus complexe, lui aussi aleatoire, qui est le container ID officiel.

On peut "rentrer" dans notre container en tapant la commande

```
docker exec -it angry_kilby bash
```

Et au lieu de `angry_kilby` je peux utiliser le container ID, of course. Je vois que mon fichier executable `hello` est toujours la.

Maintenant je tape `exit` et cela a pour effet de quitter `bash`, mais puisque `bash` et le programme principal de ce container et c'est un container interactif, alors le container aussi il disparait. Un `docker ps` nous montre qu'il est parti maintenant. Par contre, notre image est toujours la.

Si on veut arreter un container sans y rentrer, on peut faire

```
docker stop angry_bird
```

et cela aura l'effet d'arreter le container. On peut y donner un timeout a ce processus (par defaut c'est 10 secondes), apres quoi le container est `kill`ed.

Bien sur vous pouvez `docker kill` si besoin est, mais ce n'est pas vraiment recommande.

Tres souvent on a besoin de partager des document ou des fichiers entre le container et le hôte. Pour cela, rien de plus simple que d'utiler la commande:

```
docker run -it --volume $PWD:/build akosma/helloworld bash
```

Cette fois-ci on utilise un parametre `--volume` qui prend un parametre en deux parties: la premiere partie `$PWD` fait reference a folder courant dans le hôte, et la deuxieme fait reference a un folder dans le container. Maintenant si on execute `make` dans notre container, et qu'on sort du container, on a un fichier executable Linux `hello` dans le folder courant, mais bien entendu il n'est pas executable car je suis sur un Mac. Le container vraiment est un Linux qui tourne n'importe quel logiciel Linux!

Maintenant on va creer un nouveau container base sur notre image "helloworld". Pour cela on a besoin d'un nouveau `Dockerfile` qui va nous permettre de faire un peu de developpement web en PHP. Pour cela, on va ajouter quelques commandes et reconstruire notre image, en ajoutant la derniere mouture de PHP. A la fin de notre `Dockerfile` on ajoute une command `EXPOSE` pour documenter le fait que l'on exposera le port 4000 au monde exterieur.

Par défaut, les containers peuvent se parler entre eux, mais n'exposent pas leur ports au monde exterieur.

```
docker run -it --detach --publish 127.0.0.1:8000:4000 akosma/php
php -S 0.0.0.0:4000
```

Le parametre `--detach` (`-d`) specifie que l'on fait un container "detached" qui commence a tourner sans interactivite. Aussi on specifie le parametre `--publish` (`-p`) pour mapper le port 8000 du hôte au port 4000 du container. Finalement, on lance le serveur web incorporé dans PHP a l'aide de la commande `php -S 0.0.0.0:4000` qui le lance de telle façon qu'il ecoute des requetes depuis n'importe quelle interface. Ajoutez-y une commande `--volume` et la base de données MySQL dans le mix, et vous avez un environnement d'etudes LAMP pret a l'usage!

Outils

Bien sûr, utiliser la ligne de commande n'est pas du gout de chacun. En pensant à cela, Docker a fourni un outil appele Kitematic, compatible avec Windows, Mac et Linux, qui peut etre utilise pour gerer facilement les images et containers Docker.

Aussi, plusieurs editeurs comme Visual Studio Code vous permettent de voir et d'interagir avec vos images et containers directement depuis leur interface, pour pouvoir demarrer ou stopper des containers et en demarrer de nouveaux a partir d'images.

Limitations

Comme tout outil, Docker n'est pas parfait, et a quelques limitations dont il faut tenir compte:

- Un hôte ne peut tourner que des containers de la même architecture; pas de containers PPC ou ARM sur un hôte x64! Ceci est la plus grande

difference entre les containers et les machines virtuelles “classiques”.
- En revanche, et ceci pour plusieurs raisons, les containers Linux tournent sur tous les hôtes; tant Docker pour Mac que pour Windows ont des “couches logicielles de compatibilite” qui permettent de les faire tourner sans probleme.
- Aussi, il faut savoir que les containers Microsoft ne tournent que sur Windows (of course!). Il est ainsi possible de faire tourner des applications ASP.NET sur des containers Docker qui heritent de l’image `windowsservercore`.

Scenarios

Cette demo vous a montre comment faire quelques images, et comment faire quelques containers a partir de ces images. Maintenant on verra d’autres usages possibles de Docker, qui pourraient vous donner des idees.

- Faire tourner Moodle facilement a l’interieur d’un container, sans devoir passer par un serveur ou un departement de IT.
- Enseigner des langages de programmation esoteriques ou historiques, comme LISP, Prolog, COBOL, Pascal ou Fortran, de facon simple et interactive.
- Faire tourner des vieilles versions de logiciels pour les demontrer sans devoir modifier la configuration de la machine hôte.
- Enseigner le developpement web de facon facile, que se soit LAMP ou MEAN, avec les bases de données et les langages de programmation deja installes et prets a l’usage.
- Enseigner Docker! C’est une technologie tres en vogue, tres prisee par l’industrie, qui est devenue un standard en tres peu de temps. Habituer vos eleves a utiliser Docker, ne serait-ce qu’a en connaitre les principes de base, c’est deja un grand pas en avant pour leur insertion laborale.

Recommendations

Ayant vu quelques uns des avantages de Docker, on peut passer maintenant a quelques recommandations pour vos images en milieu educatif:

- Utilisez toujours Linux pour vos images, dans la mesure du possible, car elles sont les plus portables de toutes.
- Creez un “arbre” d’images, avec une image de base que vous pouvez mettre a jour regulierement.
- Automatisez la creation et la mise a jour des images avec un script, car il est facile d’oublier les commandes Docker apres un temps.
- Si vos images sont interactives, je vous recommande d’utiliser une image plus complete que celle de Ubuntu 16.04, comme par exemple la Phusion Base Image qui est plus complete et mieux configuree pour l’usage interactif.

- Si vous avez besoin de partager des applications interactives, ajoutez un serveur de VNC dans votre image et connectez-vous a celle-ci avec un viewer VNC, disponible nativement sur le Mac.
- Il y a une enorme quantite d'images officielles gratuites disponibles sur Github et aussi quelques unes gratuites mais commerciales (aussi payantes) sur le Docker Store.
- Livres: je vous recommande The Docker Book (independant) et le Docker Cookbook de chez O'Reilly.
- Docker propose lui meme de la documentation de grande qualite, et des videos gratuites pour apprendre plus sur cette technologie.

Conclusion

Docker est une technologie nouvelle, ouverte et gratuite, qui est deja devenue un standard de l'industrie. Il est possible de faire tourner des containers sur Windows Azure (le cloud de Microsoft) ou sur Amazon, simplement et avec un clic d'une souris. Plusieurs entreprises suisses deja ont adopte ou sont a point d'adopter Docker pour simplifier leur infrastructure, donc il y a un interet pedagogique et commercial pour exposer les eleves a cette nouvelle technologie, pour qu'ils soient au courant et en fasse la preuve.

Elle peut etre vraiment utile pour standardiser le setup des composants necessaires pour un cours, tout en facilitant la mise a jour et le partage de ces images avec de collegues ou avec la societe toute entiere.