Lab 01- Docker

Premiers conteneurs Alpine Linux

source: https://training.play-with-docker.com/ops-s1-hello/

Dans cet atelier, vous allez explorer les bases du fonctionnement des conteneurs, ainsi que la manière dont le moteur Docker exécute et isole les conteneurs les uns des autres.

Concepts :

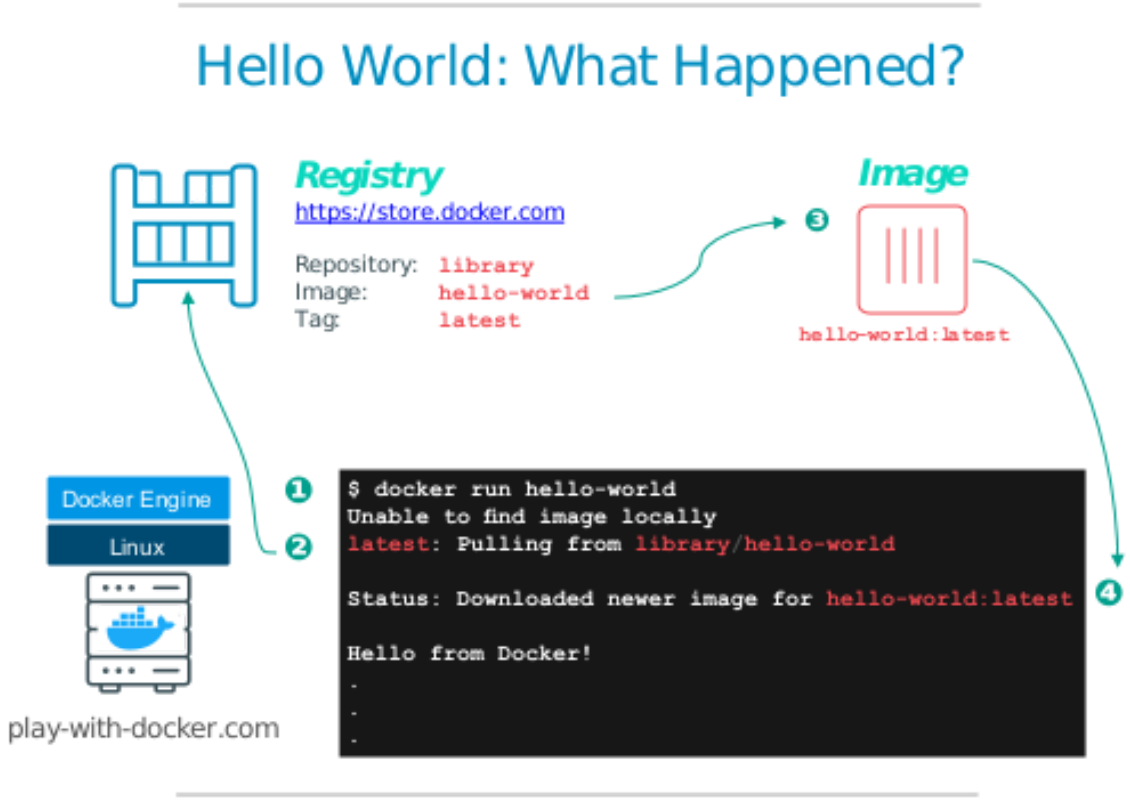
- Moteur Docker (Docker Engine)
- Conteneurs et images
- Registres d'images et Docker Hub
- Isolation des conteneurs

1.0 Exécuter votre premier conteneur

Pour exécuter votre premier conteneur Docker, tapez :

```
docker container run hello-world
```

La sortie du conteneur *hello-world* vous en dit un peu plus sur ce qui vient de se passer. Essentiellement, le moteur Docker exécuté dans votre terminal a essayé de trouver une **image** nommée hello-world. Puisque vous venez de commencer, aucune image n'est stockée localement (`Unable to find image...`), le moteur Docker accède donc à son **registre Docker** par défaut, qui est Docker Hub , pour rechercher une image nommée « hello-world ». Il y trouve l'image, l'extrait, puis l'exécute dans un conteneur. Et la seule fonction de hello-world est d'afficher le texte que vous voyez dans votre terminal, ensuite le conteneur se ferme.



Questions:

1. Pensez vous que cela revient à exécuter une machine virtuelle?
2. Un conteneur est une abstraction ?
    a. *matérielle*
    b. *d'application*
3. Est-il possible d'utiliser à la fois des machines virtuelles et des conteneurs dans un même environnement ? Comment ?

1.1 Images Docker

Dans la suite de cet atelier, vous allez exécuter un conteneur Alpine Linux . Alpine est une distribution Linux légère, elle est donc rapide à extraire et à exécuter, ce qui en fait un point de départ populaire pour de nombreuses autres images.

Pour commencer, exécutez ce qui suit dans votre terminal :

```
docker image pull alpine
```

La commande `pull` récupère l' **image** alpine du **registre Docker** et l'enregistre dans notre système. Dans ce cas, le registre est **Docker Hub** . Vous pouvez modifier le registre.

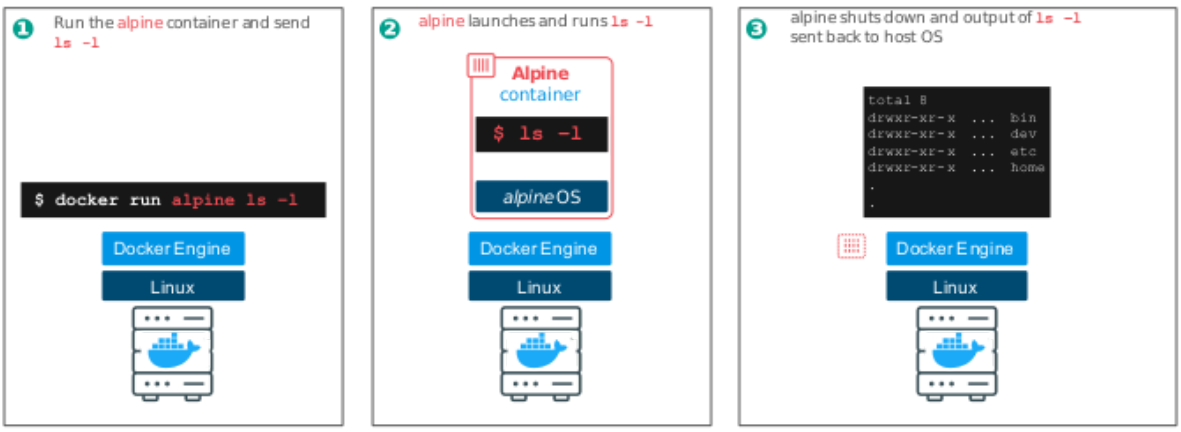
Vous pouvez utiliser la commande `docker image` pour voir la liste de toutes les images de votre système. Exécutez dans votre terminal :

```
docker image ls
```

1.1 Exécution du conteneur Docker

Exécutez maintenant un **conteneur** Docker basé sur cette image. Pour ce faire, vous allez utiliser la commande `docker container run`:

```
docker container run alpine ls -l
```



Lorsque vous appelez `run`, le client Docker trouve l'image (alpine dans ce cas), crée le conteneur puis exécute une commande dans ce conteneur. Lorsque vous exécutez `docker container run alpine`, vous avez fourni une commande (`ls -l`), donc Docker a exécuté cette commande dans le conteneur pour lequel vous avez vu la liste des répertoires. Une fois la commande `ls` terminée, le conteneur s'est arrêté.

Le fait que le conteneur se soit fermé après l'exécution de notre commande est important. Tapez ce qui suit :

```
docker container run alpine echo "hello from alpine"
```

Qu'est ce qui s'est passé ?

Essayez une autre commande:

```
docker container run alpine /bin/sh
```

Qu'est ce qui s'est passé ?

Essayez encore une autre commande::

```
docker container run -it alpine /bin/sh
```

Qu'est ce qui s'est passé ?

Vous avez exécuté chacune de vos commandes ci-dessus dans une instance de conteneur distincte. Nous pouvons voir ces instances en utilisant la commande `docker container ls` qui liste les conteneurs en cours d'exécution :

```
docker container ls
```

Puisqu'aucun conteneur n'est en cours d'exécution, vous voyez une ligne vide. Essayons une variante plus utile `docker container ls -a` :

```
docker container ls -a
```

Que représente le résultat de cette commande ?

Pour en savoir plus sur `run`, utilisez `docker container run --help` pour voir une liste de tous les indicateurs qu'il prend en charge.

1.2 Isolation des conteneurs

Dans les étapes ci-dessus, nous avons exécuté plusieurs commandes via des instances de conteneur à l'aide de `docker container run`. La commande `docker container ls -a` nous a montré qu'il y avait plusieurs conteneurs répertoriés.

Pourquoi y a-t-il autant de conteneurs répertoriés s'ils proviennent tous de l' image *alpine*?

Il s'agit d'un concept de sécurité critique dans le monde des conteneurs Docker ! Même si chaque commande utilisait la même ***image*** `docker container run` alpine , chaque exécution était un ***conteneur*** distinct et isolé . Chaque conteneur possède un système de fichiers distinct et s'exécute dans un espace de noms différent ; par défaut, un conteneur n'a aucun moyen d'interagir avec d'autres conteneurs, même ceux de la même image. Essayons un autre exercice pour en savoir plus sur l'isolation, tapez :

```
docker container run -it alpine /bin/ash
```

Pour `/bin/ash` il s'agit d'un autre type de shell disponible dans l'image alpine. Une fois le conteneur lancé et que vous êtes à l'invite de commande du conteneur, tapez les commandes suivantes :`echo "hello world" > hello.txt`

```
ls
```

La première commande `echo` crée un fichier appelé « hello.txt » contenant les mots « hello world ». La deuxième commande vous donne une liste de répertoires des fichiers et devrait afficher votre fichier « hello.txt » nouvellement créé. Tapez maintenant `exit` pour quitter ce conteneur.

Pour montrer comment fonctionne l'isolation, exécutez la commande suivante :

```
docker container run alpine ls
```

C'est la même commande `ls` que nous avons utilisée dans le shell interactif du conteneur, mais cette fois, avez-vous remarqué que votre fichier « hello.txt » est manquant ? C'est l'isolation, votre commande s'est exécutée dans une nouvelle *instance* distincte , même si elle est basée sur la même *image* . La 2ème instance n'a aucun moyen d'interagir avec la 1ère instance car le Docker Engine les maintient séparées et nous n'avons configuré aucun paramètre supplémentaire qui permettrait à ces deux instances d'interagir.

Dans leur travail quotidien, les utilisateurs de Docker profitent de cette fonctionnalité non seulement pour des raisons de sécurité, mais aussi pour tester les effets des modifications apportées aux applications. L'isolation permet aux utilisateurs de créer rapidement des copies de test distinctes et isolées d'une application ou d'un service et de les exécuter côte à côte sans

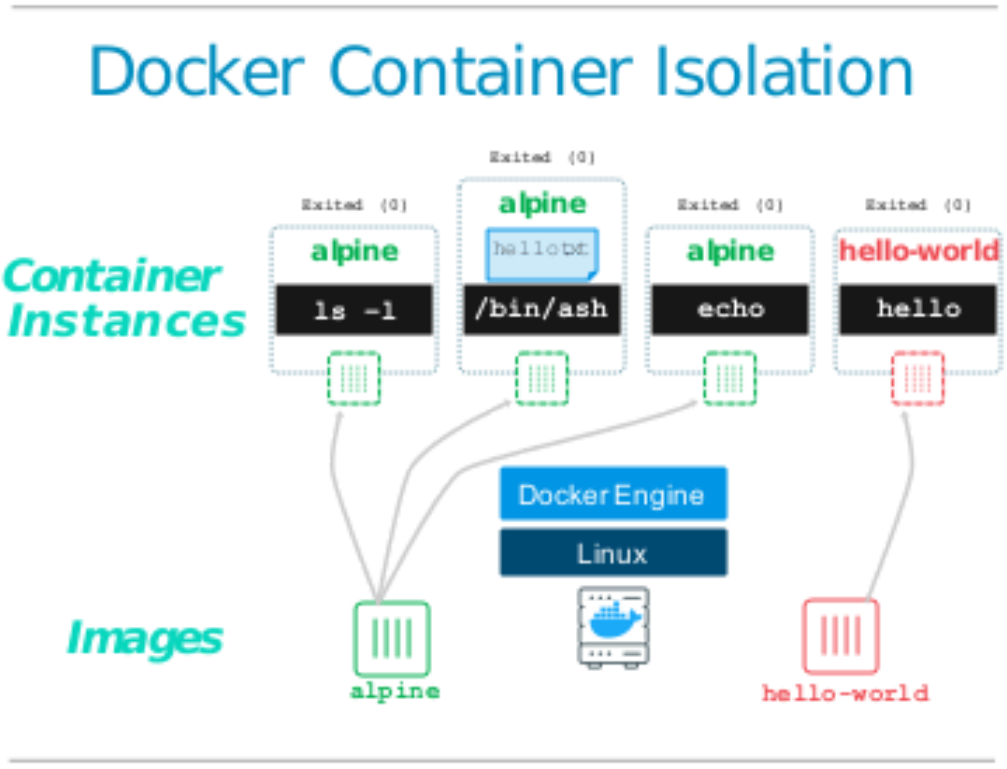
interférer les unes avec les autres. En fait, il existe tout un cycle de vie au cours duquel les utilisateurs prennent leurs modifications et les transfèrent en production en utilisant ce concept de base et les fonctionnalités intégrées de Docker Enterprise.

À l'heure actuelle, la question évidente est « Comment puis-je revenir au conteneur contenant mon fichier « hello.txt » ? »

Exécutez à nouveau le

```
docker container ls -a
```

Graphiquement, voici ce qui s'est passé sur notre Docker Engine :



Le conteneur dans lequel nous avons créé le fichier « hello.txt » est le même que celui dans lequel nous avons utilisé la commande shell `/bin/ash`, que nous pouvons voir répertorié dans la colonne « COMMAND ». Le numéro *d'ID du conteneur* de la première colonne identifie de manière unique cette instance de conteneur particulière. L'ID du conteneur pourrait être par exemple `3030c9c91e12`. Nous pouvons utiliser une commande légèrement différente pour indiquer à Docker d'exécuter cette instance de conteneur spécifique. Essayez de taper :

```
docker container start <container ID>
```

- **Conseil :** au lieu d'utiliser l'ID complet du conteneur, vous pouvez utiliser uniquement les premiers caractères, à condition qu'ils soient suffisants pour identifier un conteneur de

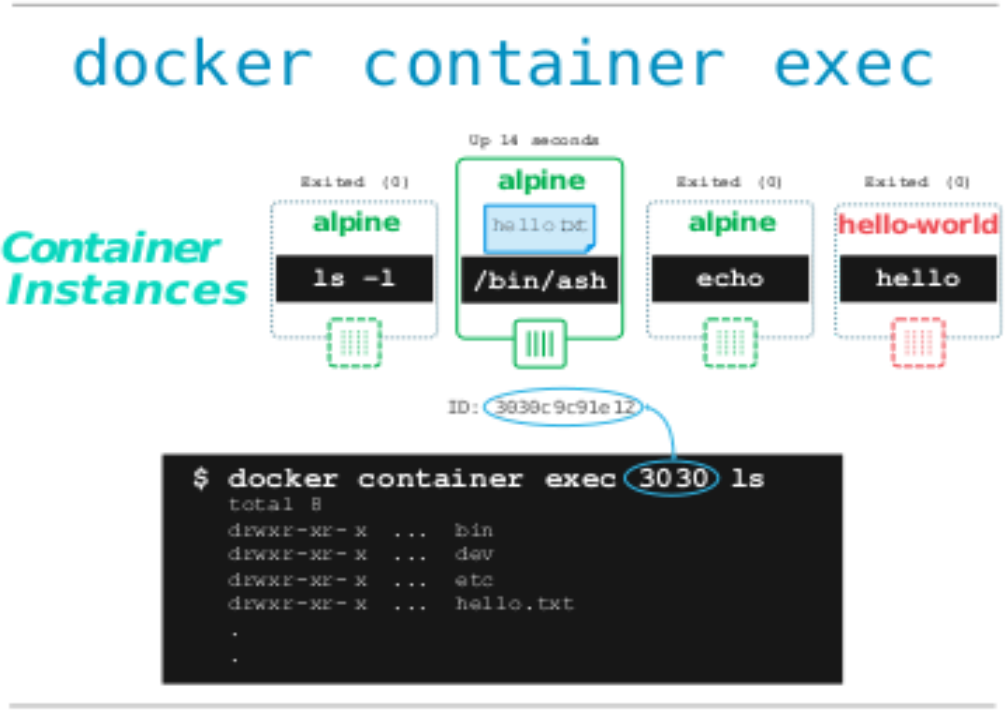
manière unique. Nous pourrions donc simplement utiliser « 3030 » pour identifier l'instance de conteneur dans l'exemple ci-dessus, puisqu'aucun autre conteneur de cette liste ne commence par ces caractères.

Utilisez maintenant à nouveau la commande `docker container ls` pour répertorier les conteneurs en cours d'exécution.

Notez cette fois que notre instance de conteneur est toujours en cours d'exécution. Nous avons utilisé le shell ash cette fois, donc plutôt que de simplement quitter la façon dont /bin/sh l'a fait plus tôt, ash attend une commande. Nous pouvons envoyer une commande au conteneur à exécuter en utilisant la commande `exec`, comme suit :

```
docker container exec <container ID> ls
```

Cette fois, nous obtenons une liste de répertoires et elle affiche notre fichier « hello.txt » car nous avons utilisé l'instance de conteneur dans laquelle nous avons créé ce fichier.



- *Démon Docker* - Le service d'arrière-plan exécuté sur l'hôte qui gère la création, l'exécution et la distribution des conteneurs Docker.
- *Client Docker* - L'outil de ligne de commande qui permet à l'utilisateur d'interagir avec le démon Docker.
- *Docker Hub* - Store est, entre autres, un registre d'images Docker. Vous pouvez considérer le registre comme un répertoire de toutes les images Docker disponibles. Vous l'utiliserez plus tard dans ce didacticiel.