Buildah: Création et gestion de conteneurs sans Docker

Table of Contents

Les conteneurs Linux

Les conteneurs Linux constituent un moyen efficace de développer et de déployer de nouvelles applications. Les technologies de conteneur regroupent et isolent les applications avec l'ensemble de l'environnement d'exécution. Les conteneurs sont ainsi rapidement prêts à l'emploi et plus faciles à porter que les applications conventionnelles, car ils contiennent l'environnement d'application complet.

Les précisions suivantes sont importantes:

- Les conteneurs sont des processus et s'exécutent sur le noyau Linux. Les conteneurs sont donc Linux.
- Le démon Docker n'est que l'un des nombreux outils et bibliothèques de l'espace utilisateur qui communiquent avec le noyau Linux pour créer des conteneurs.

Le démon et la commande docker représentent un environnement d'exécution de conteneur intégré pour la gestion des conteneurs. Avec l'avènement des normes de l'Open Container Initiative (OCI), d'autres environnements d'exécution sont en cours de développement. CRI-O est l'un de ces environnements d'exécution. Il a été créé avec un ensemble d'outils destinés à fournir d'autres moyens de travailler directement avec les conteneurs.

Ces outils peuvent être utilisés avec des conteneurs au format docker ou conformes à OCI. Certaines de ces commandes ont été créées pour être utilisées avec CRI-O, mais d'autres peuvent également interagir avec le démon docker (en remplacement des fonctionnalités de la commande docker) ou pour gérer des conteneurs sans environnement d'exécution actif:

- **podman**: peut exécuter et gérer des conteneurs et des images de conteneurs. Il prend en charge la plupart des fonctionnalités et options de commande que vous retrouver dans la commande docker, les principales différences étant que **podman** n'a pas besoin du service

Docker ni de tout autre runtime de conteneur actif pour que la commande fonctionne. De plus, **podman** stocke ses données dans la même structure de répertoires que celle utilisée par CRI-O, ce qui permettra à **podman** de travailler éventuellement avec des conteneurs activement gérés par CRI-O dans OpenShift.
- **runc**: peut être utilisée pour démarrer des conteneurs au format docker ou OCI.
- **skopeo**: permet d'inspecter des images à partir de registres d'images de conteneurs, d'obtenir des images et des calques d'images et d'utiliser des signatures pour créer et vérifier des image (La version de **skopeo** actuellement livrée avec RHEL et Atomic Host ne fonctionne qu'avec les registres v2 et non les registres v1 que Red Hat utilise actuellement).
- **buildah**: peut être utilisée à la place de docker build pour créer des images de conteneur à partir de Dockerfiles et, finalement, des fichiers dans d'autres formats.

Comprendre Buildah

Utiliser Buildah est différent de la construction d'images avec la commande docker des manières suivantes:

- **No Daemon**: Buildah contourne le démon Docker! Donc, aucune utilisation de conteneur (Docker, CRI-O ou autre) n'est nécessaire pour utiliser Buildah.
- **Image de base ou scratch**: buildah permet non seulement de créer une image basée sur un autre conteneur, mais également de commencer avec une image vide (scratch).
- **Outils de construction externes**: buildah n'inclut pas les outils de construction dans l'image elle-même. En conséquence, Buildah:
  - Réduit la taille des images construites
  - Sécurise l'image en ne disposant pas du logiciel utilisé pour construire le conteneur (comme gcc, make et dnf) dans l'image résultante.
  - Crée des images nécessitant moins de ressources pour les transporter (car elles sont plus petites).

Buildah peut fonctionner sans Docker ni aucun autre environnement d'exécution de conteneur en stockant les données séparément et en incluant des fonctionnalités qui vous permettent non seulement de créer des images, mais également de les exécuter en tant que conteneurs. Par défaut, Buildah stocke les images dans une zone identifiée comme conteneur-storage (/var/lib/conteneurs). Lorsque vous aller valider un conteneur dans une image, vous pouver exporter ce conteneur en tant qu'image Docker locale en indiquant docker-daemon (stocké dans/var/lib/docker).

L'emplacement conteneurs-stockage utilisé par défaut par la commande buildah est identique à celui utilisé par l'environnement d'exécution du conteneur CRI-O pour stocker les copies locales des images. Ainsi, les images extraites d'un registre par CRI-O ou Buildah, ou validées par la commande buildah, doivent être visibles pour les deux.

Installation de Buildah

Sous Centos/redHat

Le paquet buildah est disponible à partir du référentiel, installer le package buildah comme suit:

```
sudo dnf install  buildah
```

Par défaut, les commandes **Buildah** sont exécutées avec les privilèges root, précédés de la commande sudo. Cependant, l'une des fonctionnalités les plus attrayantes de **Buildah** est sa capacité à exécuter des conteneurs en mode rootless. Cela permet aux utilisateurs limités de travailler en toute sécurité avec **Buildah**.

Alors que **Docker** permet également d'exécuter des commandes en tant qu'utilisateur limité, le démon **Docker** s'exécute toujours en tant que root. Il s'agit d'un problème de sécurité potentiel avec **Docker**, qui peut permettre à des utilisateurs limités d'exécuter des commandes privilégiées via le démon.

Le mode rootless de **Buildah** résout ce problème car il exécute complètement les conteneurs dans un environnement rootless, sans démon racine. Pour configurer **Buildah** pour une utilisation rootless, installer les outils **slirp4netns** et **fuse-overlayfs** :

```
sudo dnf install slirp4netns fuse-overlayfs
```

Créer rapidement de nouveaux conteneurs avec Buildah

Au lieu de commencer par une image de base, Buildah peut créer un nouveau conteneur sans contenu et juste quelques métadonnées de conteneur, un conteneur appelé scratch. Un script sur GitHub appelé buildah_intro.sh montre comment créer un conteneur Buildah utilisant la compatibilité Podman et Docker. L'exemple prouve également qu'aucun démon Docker n'est nécessaire.

Un autre exemple d'ajout d'un service Web (httpd) à un conteneur et de sa configuration pour son exécution illustre l'approche simple. L'exemple montre comment créer une image afin qu'elle puisse être gérée directement par le service Docker local. Le démon Docker est stocké localement dans /var/lib/docker.

Création du conteneur

Pour commencer créer un conteneur de travail:

```
# buildah from scratch
working-container
```

Ensuite, nous créer un conteneur vide qui peut être monté comme suit:

```
# scratchmnt=$(buildah mount working-container)
# echo $scratchmnt
/var/lib/containers/storage/devicemapper/mnt/cc92011e9a2b077d03a97c0809f1f3e
7fef0f29bdc6ab5e86b85430ec77b2bf6/rootfs
```

Installation des paquets

L'étape suivante consiste à initialiser une base de données RPM dans l'image de travail et à ajouter un package Red Hat Release Package contenant les fichiers requis supplémentaires pour le déploiement RPM:

```
# rpm --root $scratchmnt –initdb
# yum install yum-utils       (if not already installed)
# yumdownloader --destdir=/tmp redhat-release-server
# rpm --root $scratchmnt -ihv /tmp/redhat-release-server*.rpm
```

Ensuite, installer le service httpd dans le répertoire de travail ...

```
# yum install -y --installroot=$scratchmnt httpd
```

... et ajouter du texte à un fichier index.html dans le conteneur pour un test ultérieur:

```
# echo "Votre conteneur httpd à partir de zéro fonctionne." >
$scratchmnt/var/www/html/index.html
```

Les options de configuration de Buildah sont utilisées pour exécuter le démon httpd “par défaut” au démarrage du conteneur:

```
# buildah config --entrypoint "/usr/sbin/httpd -DFOREGROUND" working-
container
# buildah config --port 80/tcp working-container
```

Enregistrer l'image

```
# buildah commit working-container docker-daemon:myhttpd:latest
```

Par défaut, la commande buildah commit ajoute le nom du référentiel docker.io au nom de l'image et la copie dans la zone de stockage du service Docker local (/var/lib/docker). Pour ajouter le référentiel à un registre local tout en désactivant la vérification tls utiliser:

```
# buildah commit --tls-verify=false containerID
docker://localhost:5000/imageId
```

À partir de maintenant, l'ID d'image peut être utilisé comme conteneur avec la commande docker pour exploiter la nouvelle image:

```
# docker images

REPOSITORY          TAG      IMAGE ID       CREATED          SIZE
docker.io/myhttpd   latest   47c0795d7b0e   9 minutes ago   665.6 MB

# docker run -p 8080:80 -d --name httpd-server 47c0795d7b0e
# curl localhost:8080
```

Le conteneur httpd à partir de zéro fonctionne.

conclusion

L'exemple montre à quel point un conteneur peut être créé rapidement et facilement avec Buildah, sans runtime ni démon. Buildah ne nécessitant pas de démon, il est non seulement plus facile de créer des conteneurs, mais cela simplifie également les opérations. Il n'est donc pas nécessaire de déployer une infrastructure spéciale sur l'hôte.

Mais à part la construction et l'exploitation des conteneurs, Buildah a un autre point: c'est un outil de ligne de commande (CLI). Cela facilite également l'intégration des développeurs dans les pipelines de création d'applications existants.