BUILDAH: Utilisation avancée

Table of Contents

Généralement, Buildah offre une grande variété d'options pour la création et la modification de conteneurs. Il existe donc plus d'une douzaine de façons d'utiliser la commande Buildah. Les commandes et activités clés comprennent:

- Créer un conteneur à partir d'une image différente ou à partir de zéro: Créez un nouveau conteneur (Système de fichiers racine du conteneur) basé sur une image de base existante (buildah à partir de <nom_image>) ou à partir de zéro (buildah à partir de zéro).
- Créer un conteneur à partir d'un fichier Docker: Utilisez un fichier Docker pour créer une nouvelle image de conteneur (bourgeon d'accumulation).
- Requêtes de conteneurs ou d'images: Vérifiez les métadonnées associées au conteneur ou à l'image (buildah inspect).
- Monter un conteneur: monter un fichier racine de conteneur ystems sur l'hôte pour ajouter ou modifier du contenu (buildah mount).
- Création d'une nouvelle couche de conteneur: Utilisation du contenu d'un système de fichiers racine de conteneur en tant que couche du système de fichiers pour ajouter du contenu à une nouvelle image (buildah commit).

- Supprimer un conteneur ou une image: Supprimez un conteneur (Buildah rm) ou une image de conteneur (Buildah rmi).

Utilisation basique de Buildah.

Utilisation dans les variables shell

```
container=$(buildah from centos)
echo $container

  centos-working-container-2
```

Exécution des commandes

```
buildah run $container echo "Hello Buildah"

  Hello Buildah

buildah run $container bash
yum install ruby
exit
buildah run $container whereis ruby

  ruby: /usr/bin/ruby /usr/lib64/ruby /usr/share/ruby
```

Copier le contenu dans un conteneur de travail.

```
echo "buildah test" > buildah.txt
buildah copy $container buildah.txt /tmp/buildah.txt

  ce9440fee6e37151fcc872dbf3fca2a77b95e66a627cabed14ba8349e1825607

buildah run $container cat /tmp/buildah.txt

  buildah test
```

Modifier les paramètres du conteneur de travail.

```
buildah config --hostname my-hostname $container
buildah run $container hostname
```

```
  my-hostname
```

Afficher les paramètres actuels

```
buildah inspect $container

  {
    "Type": "buildah 0.0.1",
    "FromImage": "docker.io/library/centos:latest",
    "FromImageID":
"5182e96772bf11f4b912658e265dfe0db8bd314475443b6434ea708784192892",
  .....
  .....
        "config": {
            "Hostname": "my-hostname",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
  .....
  .....
```

Autres options pour [buildah config]

- **–annotation annotation, -a annotation**: ajouter une annotation par ex. annotation = valeur, pour l'image cible (par défaut [])
- **–arch architecture**: définit architecture de l'image cible
- **–author information**: définit image auteur informations de contact
- **–cmd commande**: définit la commande par défaut à exécuter pour les conteneurs basés sur l'image
- **–comment comment**: définit un commentaire dans l'image cible
- **–created-by description**: définit description de la création de l'image
- **–domainname name**: définit un nom de domaine pour les conteneurs en fonction de l'image
- **–entrypoint entry point**: définit le point d'entrée pour les conteneurs en fonction de l'image
- **–env variable d'environnement, -e .....**: ajoute une variable d'environnement à définir lors de l'exécution de conteneurs basés sur image (par défaut [])
- **–history-comment comment**: définit un commentaire pour l'historique de l'image cible
- **–hostname name**: définit un nom d'hôte pour les conteneurs en fonction de l'image
- **–label label, -l label**: ajoute une étiquette de configuration d'image, par exemple. label = valeur
- **–onbuild value** ajoute onbuild commande à exécuter sur des images basées sur cette image. Uniquement pris en charge sur les images au format “docker”
- **–os operating system**: définit le système d'exploitation de l'image cible0
- **–port port, -p port**: ajoute un port à exposer lors de l'exécution de conteneurs basés sur une image (par défaut [])
- **–shell shell**: ajoute un shell à exécuter dans des conteneurs
- **–stop-signal stop signal**: définit stop signal pour les conteneurs en fonction de l'image
- **–user user, -u user**: définit l'utilisateur par défaut à exécuter dans des conteneurs basés sur

une image
- **–volume volume, -v volume**: ajoute le chemin du volume par défaut à créer pour les conteneurs basés sur une image (par défaut [])
- **–workingdir directory**: définit la directory pour les conteneurs basés sur une image

Monter le système de fichiers du conteneur de travail.

```
buildah mount $container

  /var/lib/containers/storage/overlay
  /58c70666c54601d8ce6c96c6079ed45ca60af25ff97cadc77cdfe2c937081b25/merged

ll /var/lib/containers/storage/overlay

  /58c70666c54601d8ce6c96c6079ed45ca60af25ff97cadc77cdfe2c937081b25/merged

  total 16
  -rw-r--r--.  1 root root 12005 Aug  5 07:05 anaconda-post.log
  lrwxrwxrwx.  1 root root     7 Aug  5 07:04 bin -> usr/bin
  drwxr-xr-x.  2 root root     6 Aug  5 07:04 dev
  drwxr-xr-x. 47 root root  4096 Aug  5 07:05 etc
  drwxr-xr-x.  2 root root     6 Apr 11 13:59 home
  .....
  .....
```

Démonter

```
buildah umount $container

  cb2fdccb666dcfd76ce526021db07652d87702bb245c711ef28ff5b34ed5588e
```

Création simple de conteneur avec Buildah

Buildah est différent de la création d'images avec Docker de plusieurs manières. Buildah peut créer un conteneur en utilisant une image de conteneur et un fichier Docker, ou il peut être démarré avec une image vide. En outre, le processus peut faire appel à des outils d’emballage externes au lieu de faire appel à un gestionnaire d’emballage dans l’image elle-même. La raison en est qu'on peut utiliser Buildah pour créer en utilisant une image existante à l'intérieur d'un conteneur en cours d'exécution (similaire à Docker) ou on peut créer l'image directement sur le volume de stockage (conteneurs/stockage) sans exécuter le conteneur. Docker nécessite un conteneur en cours d'exécution, pas Buildah.

La différence entre Buildah et la construction d’images avec la commande Docker présente plusieurs avantages:

- La taille de l'image créée est plus petite.
- Amélioration de la sécurité de l'image car le logiciel utilisé pour créer le conteneur (tel que gcc, make et dnf) n'est pas contenu dans l'image.
- Moins de ressources sont nécessaires pour déplacer les images en raison de leur taille réduite.
- On peut créer une image à couche unique qui conviendrait mieux aux environnements de test et de production.
- Meilleure gestion des secrets pour les abonnements ou les informations d'identification pour sécuriser les registres.

Buildah offre généralement un large éventail d'options pour créer et travailler avec des conteneurs; Par exemple, il existe plus d'une douzaine d'options pour utiliser la commande Buildah. Voici quelques unes des principales commandes et actions:

- Création d'un conteneur basé sur une image différente ou à partir de zéro: Créer un nouveau conteneur (système de fichiers racine du conteneur) basé sur une image de base disponible (Buildah à partir de <nom de l'image>) ou à partir de zéro (Buildah à partir de zéro).
- Création d'un conteneur à l'aide d'un fichier Dockerfile: Utiliser un fichier Dockerfile pour créer une nouvelle image de conteneur (bourgeon d'accumulation).
- Demande d'informations sur des conteneurs ou des images: vérifier les métadonnées associées au conteneur ou à l'image (buildah inspect).
- Montage d'un conteneur: monter un système de fichiers racine du conteneur sur l'hôte pour ajouter ou modifier du contenu (montage Buildah).
- Conception d'un nouveau calque de conteneur: Utiliser le contenu d'un système de fichiers racine de conteneur en tant que calque de système de fichiers pour ajouter du contenu à une nouvelle image (buildah commit).
- Suppression d'un conteneur ou d'une image: Supprimer un conteneur (Buildah rm) ou une image de conteneur (Buildah rmi).

Créer un conteneur de travail à partir d'une image.

Il est possible de créer une image au format OCI (Open Container Initiative) ou une image au format Docker sans le démon de service Docker.

```
buildah from centos

  Getting image source signatures
  Copying blob
sha256:256b176beaff7815db2a93ee2071621ae88f451bb1e198ca73010ed5bba79b65
   71.23 MiB / 71.23 MiB  4s
  Copying config
sha256:5182e96772bf11f4b912658e265dfe0db8bd314475443b6434ea708784192892
   2.13 KiB / 2.13 KiB  0s
  Writing manifest to image destination
  Storing signatures
  centos-working-container
```

On peut voir le conteneur ainsi créé

```
buildah containers

  CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME
CONTAINER NAME
  c28ce3a30750     *     5182e96772bf docker.io/library/centos:latest
centos-working-container
```

ainsi que l'image

```
buildah images

  IMAGE ID             IMAGE NAME                                CREATED AT
SIZE
  5182e96772bf         docker.io/library/centos:latest       Aug 7, 2018
04:21      208 MB
```

Pour créer un conteneur vide, spécifier [scratch]

```
buildah from scratch

  working-container [root@dlp ~]# buildah containers

  CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME
CONTAINER NAME
  c28ce3a30750     *     5182e96772bf docker.io/library/centos:latest
centos-working-container
  df3709dc7552     *                  scratch
working-container
```

[scratch] est vide, donc ne figure pas dans la liste des images

```
buildah images

  IMAGE ID             IMAGE NAME                                 CREATED AT
SIZE
  5182e96772bf         docker.io/library/centos:latest        Aug 7, 2018
04:21      208 MB
```

Créer une image de conteneur à partir du conteneur de travail.

Créer Le contenair de travail

```
buildah commit $container my-centos:latest
```

```
  Getting image source signatures
  Skipping fetch of repeat blob
  sha256:1d31b5806ba40b5f67bde96f18a181668348934a44c9253b420d5f04cfb4e37a
  Copying blob
sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1
   32 B / 32 B  0s
  Copying config
sha256:4b5dfca9b6165627a7328eef35db5216f28c5c0504e05c2bb8d3e98a1d2382a3
   1.18 KiB / 1.18 KiB  0s
  Writing manifest to image destination
  Storing signatures
  4b5dfca9b6165627a7328eef35db5216f28c5c0504e05c2bb8d3e98a1d2382a3
```

```
buildah images

IMAGE ID             IMAGE NAME                                CREATED AT
SIZE
5182e96772bf         docker.io/library/centos:latest           Aug 7, 2018
04:21      208 MB
4b5dfca9b616         localhost/my-centos:latest                Aug 28, 2018
19:13     208 MB
```

Pousser une image de conteneur à l'emplacement spécifié.

```
buildah images

  IMAGE ID             IMAGE NAME                                CREATED AT
SIZE
  5182e96772bf         docker.io/library/centos:latest           Aug 7, 2018
04:21      208 MB
  4b5dfca9b616         localhost/my-centos:latest                Aug 28, 2018
17:13     208 MB
```

push [localhost/my-centos] image to [www.srv.world:5000] registry

```
buildah push localhost/my-centos www.srv.world:5000/my-centos

  Getting image source signatures
  Copying blob
sha256:1d31b5806ba40b5f67bde96f18a181668348934a44c9253b420d5f04cfb4e37a
   198.64 MiB / 198.64 MiB  19s
  Copying blob
sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
   1.00 KiB / 1.00 KiB  0s
  Copying config
sha256:4b5dfca9b6165627a7328eef35db5216f28c5c0504e05c2bb8d3e98a1d2382a3
   1.18 KiB / 1.18 KiB  0s
  Writing manifest to image destination
```

```
  Copying config
sha256:4b5dfca9b6165627a7328eef35db5216f28c5c0504e05c2bb8d3e98a1d2382a3
   0 B / 1.18 KiB  0s
  Writing manifest to image destination
  Storing signatures
```

il est possible de tirer sur d'autres nœuds Docker

```
docker pull www.srv.world:5000/my-centos
docker images

  REPOSITORY                     TAG         IMAGE ID         CREATED
SIZE
  www.srv.world:5000/my-centos   latest      4b5dfca9b616     40 hours ago
200 MB
  docker.io/centos               latest      5182e96772bf     3 weeks ago
200 MB
  docker.io/registry             2           b2b03e9146e1     7 weeks ago
33.3 MB

newcontainer=$(buildah from scratch)
buildah containers

  CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME
CONTAINER NAME
  73939f4effb5     *     5182e96772bf docker.io/library/centos:latest
centos-working-container
  5e9fe96b5c0f     *                  scratch
working-container

mount [scratch] container
scratchmnt=$(buildah mount $newcontainer)
echo $scratchmnt

  /var/lib/containers/storage/overlay
  /274b88752542b38dce02c332f31032c6cef86cf67f7738d5fb8a8e47cf5a868f/merged
```

Installer des packages dans [scratch] container

```
yum -y install bash coreutils --releasever=7 --installroot=$scratchmnt
```

unmount

```
buildah umount $newcontainer

  5e9fe96b5c0f1fc13e0d5bcd45e81df17285cce90a21688bef0691e9a755d152
```

run container

```
buildah run $newcontainer bash

bash-4.2#
bash-4.2# ls

bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr

bash-4.2# exit
```

faire un commit d'une image

```
buildah commit $newcontainer centos-minimum:latest

  Getting image source signatures
  Copying blob
sha256:3a143b7ad55d76d0113ac74c10e3c76a73f8394f1cd92a6f868ff1483080da0e
   83.06 MiB / 83.06 MiB  4s
  Copying config
sha256:8b03f6f93c4c95614984f54e58529ea9341d02736eb42b69f4c4bd94b76f3ed1
   279 B / 279 B  0s
  Writing manifest to image destination
  Storing signatures
  8b03f6f93c4c95614984f54e58529ea9341d02736eb42b69f4c4bd94b76f3ed1

buildah images

  IMAGE ID          IMAGE NAME                                CREATED AT
SIZE
  5182e96772bf      docker.io/library/centos:latest           Aug 7, 2018 04:21
208 MB
  8b03f6f93c4c      localhost/centos-minimum:latest           Aug 30, 2018
11:02     257 MB
```

Créer des conteneurs scratch

Au lieu de commencer par une image de base, on peut créer un nouveau conteneur qui ne contient aucun contenu et uniquement une petite quantité de métadonnées de conteneur (conteneur scratch. Voici quelques problèmes à prendre en compte lorsqu'on choisit de créer une image à partir d'un conteneur de travail avec la commande buildah:

- Avec un conteneur scratch, on peut simplement copier des fichiers exécutables ne dépendant pas de l'image de travail et définir quelques paramètres de configuration pour faire fonctionner un conteneur minimal.
- Pour utiliser des outils tels que les packages yum ou rpm pour renseigner le conteneur de travail, il faudra au moins initialiser une base de données RPM dans le conteneur et ajouter un package de publication.
- Si on ajoute de nombreux packages RPM, il faut envisager d'utiliser les images de base rhel ou

rhel-minimal au lieu d'une image scratch. La documentation, les modules linguistiques et d'autres composants de ces images de base ont été supprimés, ce qui peut finalement réduire la taille de l'image.

L'exemple suivant illustre comment créer une image afin qu'elle puisse être gérée directement par le service Docker local (démon Docker stocké localement dans /var/lib/docker).

À partir de la création d'un conteneur de travail:

```
buildah from scratch

  working-container
```

Créer un conteneur vide pouvant être monté comme suit:

```
scratchmnt=$(buildah mount working-container)
echo $scratchmnt /var/lib/containers/storage/devicemapper

/mnt/cc92011e9a2b077d03a97c0809f1f3e7fef0f29bdc6ab5e86b85430ec77b2bf6/rootfs
```

Initialiser une base de données RPM dans l'image de travail et ajouter un package de version Red Hat contenant les fichiers requis supplémentaires pour utiliser le 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
```

Installation du service httpd dans le répertoire scratch:

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

Ajouter du texte à un fichier index.html dans le conteneur pour les tests suivants

```
echo "Your httpd container from scratch worked." >
$scratchmnt/var/www/html/index.html
```

Les options ‘buildah config’ sont utilisées pour exécuter le démon httpd par défaut lorsque le conteneur est démarré:

```
buildah config --entrypoint "/usr/sbin/httpd -DFOREGROUND" working-container
buildah config --port 80/tcp working-container
buildah commit working-container docker-daemon:myhttpd:latest
```

Sur la base d’un paramètre par défaut, la commande ‘buildah commit’ ajoute le nom du référentiel

docker.io au nom de l'image et copie l'image dans la zone de stockage du service Docker local (/var/lib/docker). On peut ensuite utiliser l'ID image pour utiliser la nouvelle image en tant que conteneur à l'aide de la commande 'docker':

```
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

 Your httpd container from scratch worked.
```

L'exemple ci-dessus montre à quel point il est rapide et pratique de créer un conteneur à l'aide de Buildah. De plus, il n'est pas nécessaire de lancer un conteneur ou démon. Le fait que Buildah ne nécessite pas de démon facilite non seulement la construction d'un conteneur, mais simplifie également les opérations puisque l'hôte sur lequel il est déployé ne nécessite pas d'infrastructure spéciale.

En plus de la construction et de l'exploitation de conteneurs, Buildah offre un avantage supplémentaire: c'est un outil en ligne de commande. Cela signifie que les développeurs peuvent l'intégrer avec beaucoup plus de facilité dans les pipelines existants pour la création d'applications.

Créer de petits conteneurs avec Buildah

Buildah la création d'images OCI (Open Container Initiative) et permet en particulier de personnaliser la taille de l'image créée.

A titre de comparaison on créé une image docker

```
docker pull centos:7
docker images

  REPOSITORY           TAG              IMAGE ID                CREATED
SIZE
  docker.io/centos          7                  2d194b392dd1        2 weeks
ago
```

On note que la taille de l'image de Docker est de 195 Mo. Ensuite on crée une image minimale (scratch) à l'aide de Buildah, avec uniquement les packages coreutils et bash ajoutés à l'image, à l'aide du script suivant:

```
cat  ./buildah-base.sh
#!/bin/bash

set -x
```

```
# build a minimal image
newcontainer=$(buildah from scratch)
scratchmnt=$(buildah mount $newcontainer)

# install the packages
yum install --installroot $scratchmnt bash coreutils --releasever 7 --setopt
install_weak_deps=false -y
yum clean all -y --installroot $scratchmnt --releasever 7

sudo buildah config --cmd /bin/bash $newcontainer

# set some config info
buildah config --label name=centos-base $newcontainer

# commit the image
buildah unmount $newcontainer
buildah commit $newcontainer centos-base
```

Lorsqu'on utilise ce script pour créer une image :

```
sudo ./buildah-base.sh

sudo buildah images

  IMAGE ID               IMAGE NAME
CREATED AT            SIZE
  8379315d3e3e     docker.io/library/centos-base:latest           Mar 25,
2018 17:08   212.1 MB
```

L'image est plus grande de 17 Mo, car python et yum n'étaient pas installés dans l'image Buildah, alors qu'ils l'étaient dans l'image Docker. On peut constater que la documentation et les archives de localisation prennent un peu plus de 100 Mo d'espace dans l'image Buildah.

Dans le script buildah-bash.sh précédent on va forcer les paramètres régionaux dans le programme d'installation yum:

```
#!/bin/bash

set -x

# build a minimal image
newcontainer=$(buildah from scratch)
scratchmnt=$(buildah mount $newcontainer)

# install the packages
yum install --installroot $scratchmnt bash coreutils --releasever 7 --
setopt=install_weak_deps=false --setopt=tsflags=nodocs --
setopt=override_install_langs=en_US.utf8 -y
yum clean all -y --installroot $scratchmnt --releasever 7
```

```
sudo buildah config --cmd /bin/bash $newcontainer

# set some config info
buildah config --label name=centos-base $newcontainer

# commit the image
buildah unmount $newcontainer
buildah commit $newcontainer centos-base
```

Lorsqu'on exécuté ce nouveau script, la taille de l’image a été réduite à 92 Mo. Elle a perdu 120 Mo de la taille originale de l’image Buildah et se rapproche de la taille attendue. Cependant, on peut aller plus loin en supprimant des packages de paramètres régionaux individuels pour économiser encore plus d'espace.

Maniputlation des conteneurs

Modification d'un conteneur pour créer une nouvelle image avec Buildah

Il existe plusieurs façons de modifier un conteneur existant à l'aide de la commande buildah et de valider ces modifications dans une nouvelle image de conteneur:

- Monter un conteneur et y copier des fichiers
- Utiliser Buildah Copy et Buildah Config pour modifier un conteneur

Une fois qu'on a modifié le conteneur, utiliser buildah commit pour valider les modifications apportées à une nouvelle image.

Utilisation de Buildah Mount pour modifier un conteneur

Après avoir obtenu une image avec Buildah, on peut utiliser cette image comme base d’une nouvelle image. Le texte suivant montre comment créer une nouvelle image en montant un conteneur de travail, en ajoutant des fichiers à ce conteneur, puis en validant les modifications apportées à une nouvelle image.

Taper ce qui suit pour afficher le conteneur de travail qu'on a utilisé précédemment:

```
buildah containers

  CONTAINER ID BUILDER IMAGE ID     IMAGE NAME  CONTAINER NAME

  dc8f21af4a47   *     1456eedf8101 registry.access.redhat.com/rhel7/rhel-
minimal:latest
               rhel-minimal-working-container
  6d1ffccb557d   *     ab230ac5aba3 docker.io/library/myecho:latest
```

```
                myecho-working-container
```

Monter l'image du conteneur et définir le point de montage sur une variable ($mymount) pour faciliter la gestion:

```
mymount=$(buildah mount myecho-working-container)
echo $mymount

  /var/lib/containers/storage/devicemapper
/mnt/176c273fe28c23e5319805a2c48559305a57a706cc7ae7bec7da4cd79edd3c02/rootfs
```

Ajouter du contenu au script créé précédemment dans le conteneur monté:

```
echo 'echo "We even modified it."' >>
$mymount/usr/local/bin/myechoal/bin/myecho
```

Pour valider le contenu que l'on a ajouté pour créer une nouvelle image (nommée myecho), taper ce qui suit:

```
buildah commit myecho-working-container containers-storage:myecho2
```

Pour vérifier que la nouvelle image inclut les modifications, créer un conteneur de travail et l'exécuter:

```
buildah images
  IMAGE ID     IMAGE NAME     CREATED AT          SIZE
  a7e06d3cd0e2 docker.io/library/myecho2:latest
                            Oct 12, 2017 15:15  3.144 KB
buildah from docker.io/library/myecho2:latest
  myecho2-working-container
buildah run myecho2-working-container
  This container works!
  We even modified it.
```

On peut voir que la nouvelle commande echo ajoutée au script affiche le texte supplémentaire.

Lorsqu'on a terminé, on peut démonter le conteneur:

```
buildah umount myecho-working-container
```

Utiliser buildah copy et buildah config pour modifier un conteneur

Avec Buildah Copy, on peut copier des fichiers dans un conteneur sans le monter au préalable. Voici un exemple, en utilisant le conteneur myecho-working-container créé (et démonté) dans la section précédente, pour copier un nouveau script dans le conteneur et modifier la configuration du

conteneur pour exécuter ce script par défaut.

Créer un script appelé newecho et rendez-le exécutable:

```
cat newecho
echo "I changed this container"
chmod 755 newecho
```

Créer un nouveau conteneur de travail:

```
buildah from myecho:latest
  myecho-working-container-2
```

Copier newecho dans/usr/local/bin à l'intérieur du conteneur:

```
buildah copy myecho-working-container-2 newecho /usr/local/bin
```

Changer la configuration pour utiliser le script newecho comme nouveau point d'entrée:

```
buildah config myecho-working-container-2 --entrypoint "/bin/sh -c
/usr/local/bin/newecho"
```

Exécuter le nouveau conteneur, ce qui devrait entraîner l'exécution de la commande newecho:

```
buildah run myecho-working-container-2
  I changed this container
```

Si le conteneur se comporte comme prévu, on peut alors le valider pour une nouvelle image (mynewecho):

```
buildah commit myecho-working-container-2 containers-storage:mynewecho
```

Suppression d'images ou de conteneurs avec Buildah

Lorsqu'on a terminé avec des conteneurs ou des images spécifiques, on peut les supprimer avec buildah rm ou buildah rmi, respectivement. Voici quelques exemples.

Pour supprimer le conteneur créé dans la section précédente, taper ce qui suit pour voir le conteneur monté, le démonter et le supprimer:

```
buildah containers
  CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME
CONTAINER NAME
```

```
  05387e29ab93     *     c37e14066ac7 docker.io/library/myecho:latest
myecho-working-container
buildah mount
  05387e29ab93
/var/lib/containers/storage/devicemapper/mnt/9274181773a.../rootfs
buildah umount 05387e29ab93
buildah rm 05387e29ab93
  05387e29ab93151cf52e9c85c573f3e8ab64af1592b1ff9315db8a10a77d7c22
```

Pour supprimer l'image créée précédemment, taper les éléments suivants:

```
buildah rmi docker.io/library/myecho:latest
  untagged: docker.io/library/myecho:latest
  ab230ac5aba3b5a0a7c3d2c5e0793280c1a1b4d2457a75a01b70a4b7a9ed415a
```