Docker volume orchestration

<< Back to Docker main

Contents

Áttekintés

Az úgynevezett volume orkesztráció kulcsfontosságú a konténeres cluster építésben. Orchestration alatt azt érjük, hogy a több VM-en futó, sok konténerb?l álló cluster-en az egyes konténerek (amiket "random" telepít fel a swarm valamelyik szabad kapacitású VM-re) hogyan férnek hozzá a futásukhoz szükséges perzisztens tárolóhoz, ahol a konfigurációs állományuk van. Pl. a Portainer-nek hozzá kell férnie egy olyan perzisztens tárolóhoz, ahol az adatbázisát, beállításait tárolni tudja. Ha a Portainer-t swarm service-ként telepítjük fel, akkor nem tudjuk el?re megmondani, hogy a swarm melyik node-ra fogja feltelepíteni. Bárhova is telepíti föl, ott rendelkezésre kell álljon a neki szükséges perzisztens állomány. Két f? csapásirány közül választhatunk:

Mount VM fileshare to container

Valamilyen network file system-et mount-unk minden egyes VM-re, ami részt vesz a cluster-ben. (pl NFS vagy Samba). Aztán a VM-en létrejött NFS vagy Samba megosztásra mount-oljuk a konténer egyik mappáját **bind moun**t-al. Ezzel több baj is van, ez produckicós környezetben csak nagyon kis cluster-en m?köd?képes, és nem ez az ajánlott módszer.

docker volume plugin

Használhatunk úgynevezett **docker volume plugin**-eket, amik lehet?év teszik, hogy on-demand alapon közvetlen a konténerre mount-oljuk föl a network fájlrendszer megfelel? megosztását (Pl Netshare vagy REX-ray) vagy a VM-re mount-olt network megosztást használhassuk volume-ként a konténer számára transzparnes módon (Pl Convoy) . Nagyon sokféle volume plugin elérhet? a docker-hez, amikkel NFS, AWS, Azure és még sok más network fájlrendszert tudunk közvetlen a konténerbe bekötni. A volume plugin használatakor pont úgy kell megadni a volume-ot mint ha az a lokális VM-en lenne. A plugin elfedi el?lünk a network filesystem-hez való csatlakozás komplexitását.

Mikor egy volume-ot megadunk akár a **docker run**, akár a **docker service create** parancsban, akkor igazából mindig megadjuk a volume driver típusát a **volume-driver** paraméterben. Ha külön nem definiáljuk, akkor a **local** driver-t fogja használni, amivel a volume a lokális VM-en fog létrejönni. Ha nem a lokális driver-t akarjuk használni, akkor két dolgunk van.

1. Fel kell telepíteni a kiválasztott volume plugin-t.
2. A service definiálásakor a mount parancsban meg kell adni a **volume-driver** paraméterrel a használni kívánt driver nevét.

Vannak olyan driver-ek, amiket csak egy bizonyos virtualizációs környezetben lehet használni (pl AWS), mert kifejezetten az ottani network fájlrendszert támogatja csak. Ilyen pl. a **REX-Ray** vagy a **Trident**. Ezeket nem tudjuk pl KVM-en használni. Az REX-Ray pl képes maga létrehozni a megosztást az Amazon EC2-ön, ha a swarm service létrehozásakor a megadott megosztás még nem létezett volna.

Vannak olyan driver-ek, amik a szabványos network fájlrendszereket (pl NFS) is támogatják. Ezek bármilyen virtualizációs környezetben használhatóak, lényeg hogy a guest VM támogassa a használatukat (pl hogy NFS esetén telepítve legyen az NFS kliens a guest-en). Ilyen plugin pl a **Netshare** és a **Convoy**.

Lista az elérhet? hivatalosan támogatott plugin-ekr?l: https://docs.docker.com/engine/extend/legacy_plugins/
Bizonyos pluginek benne fent vannak a docker hub-on, így a **docker plugin install** paranccsal automatikusan telepíthet?ek.

Volume plugin vs storage driver

A **docker volume plugin**-eket nem szabad összekeverni a **storage driver** témakörrel (https://docs.docker.com/storage/storagedriver/#copying-makes-containers-efficient). A **storage driver**-ek a konténer tetején található vékony írható réteget kezelik, míg a volume plugin-ek a konténerbe felcsatolt volume-ok kezeléséért felel?sek.

A konténerek fels? vékony írható rétege nem perzisztens, az csak arra való, hogy a konténer futása közben módosított image fájlokat tárolja vagy az esetlegesen létrehozott új fájlokat, amik benne sem voltak az image-ban. Mikor a konténer hozzá akar férni vagy módosítani akar egy fájlt, akkor a **storage driver** megpróbálja azt megkeresni a konténert felépít? (nem írható) image rétegekben fentr?l lefelé. Azt hogy ezt pontosan milyen algoritmussal csinálja, és hogy milyen hatásfokkal az a storage driver-t?l függ.

Ez a vékony írható réteg csak "gyenge" írási terhelésre van kitalálva, semmi képen sem arra, hogy egy adatbázis kezel? ott tárolja az adatbázis fájlokat. Er?s írási terhelésre kifejezetten a docker volume-okat kell használni tipikusan valamelyik volume plugin-al együtt.

Elérhet? storage diver-ek: **aufs, devicemapper, overlay2, vfs**
Az alapértelmezett az overlay2.

NFS mount on the VM

Az itt leírtaknak még semmi köze a docker volume plugin-ekhez. Itt annyit mutatok be, hogy lehet minden egyes VM-re felcsatolni ugyanazt az NFS megosztást. Ez egy jó kiindulási alap. A guest VM-ekre felcsatolt NFS megosztásokat kétféle képen használhatják a konténerek:

- Tovább mount-oljuk a konténer belsejébe b**ind mount**-al a megosztást (csak tesztelési célra, kis méret? cluster-ekre)
- A megfelel? docker volume diver plugin-el volume-okat hozunk létre az NFS megosztáson a konténer számára transzparens módon. (Convoy plugin)

A következ?kben KVM/qemu VM-en futó boot2docker Lunux-ra fogunk felcsatolni egy NFS megosztást. A példában a Portainer service számára fogunk létrehozni egy megosztást, ahol a /data mappáját perzisztens módon tudja tárolni, amiben a beállítások és felhasználó adatok vannak.

NFS szerver telepítése

Els?ként a host gépen (vagy bárhol máshol, amit elérnek majd a VM-ek) hozzunk létre egy NFS megosztást.

```
# dnf install -y nfs-utils
# systemctl enable nfs
# systemctl restart nfs
```

A host gép t?zfalán az alábbi portokat kell kinyitni:

- **2049** - udp + tcp
- **111** - udp + tcp

Ha a GUI t?zfal szerkeszt?ben csak a Permanent beállítások közé vesszük fel, akkor újra kell indítani a t?zfalat vagy a gépet. Ha a Runtime-ba is felvesszük, akkor azonnal érvényre jut. (A 'No route to host' tipikusan t?zfal hibára vall)

A NFS szerver konfigurációs fájlja a **/etc/exports** . Ebbe egy sort kell csak belerakjunk:

```
/home/adam/docker/portainer/data/ 192.168.42.0/255.255.255.0(rw,sync,no_root_squash,no_subtree_check)
```

Vagyis, hogy a /home/adam/docker/portainer/data/ mappának a megosztását megengedjük a 192.168.42. -ás alhálózaton

Töltsük újra a NFS fájlt:

```
# exportfs -ra
```

Majd nézzük meg, a showmount paranccsal hogy a szerverünk valóban kiajánlja e a fenit mappát:

```
# showmount -e 192.168.42.1
Export list for 192.168.42.1:
/home/adam/Projects/DockerCourse/portainer/data 192.168.42.0/255.255.255.0
```

NFS kliens beállítása

Az összes node-on mount-olni kell az el?bb létrehozott NFS megosztást, amiben a Portainer a perzisztens adatait fogja tárolni. Ehhez be kell lépni SSH-val egyenként az összes node-kora és ott futtatni kell a megfelel? **mount** parancsot.

Note
KVM driver által telepített **boot2docker** Linux-okon csak a **/var/lib/boot2docker** mappa tartalma boot perzisztens, minden máshol történt módosítást ki fog dobni újraindításkor

Kézi mount

A boot2docker-ben az NFS mount-nak speciális szerkezete van, leírás itt: https://gist.github.com/leeor/e70419fd7f656ca4bff3
Mivel a boot2docker-ben csak a **/var/lib/boot2docker/** tartalma boot perzisztens, ezen a mappán belül hozzuk létre a portainer számára azt a mappát ahova az NFS megosztást fel fogjuk csatolni.

```
sudo mkdir /var/lib/boot2docker/portainerdata
sudo /usr/local/etc/init.d/nfs-client start
sudo mount 192.168.42.1:/home/adam/docker/portainer/data/ /var/lib/boot2docker/portainerdata -o rw,async,noatime,rsize=32768,wsize=32768,prot
```

A második parancs elindítja az NFS klienst a mount el?tt. (normál esetben a mount így nézne ki: *mount -t nfs source target*)

Ha ez m?ködik, akkor az utolsó két sort írjuk be a **/var/lib/boot2docker/profile** fájlba, ami a docker user-hez tartozó profile-ja, hogy boot perzisztensé tegyük a mount-ot. Fontos, hogy a profile fájlt csak akkor tudjuk írni, ha el?tte 777 jogot adunk rá még akkor is ha sudo-t használunk. Ki érti ezt?!

Automatizált mount script-el

mountNFS.sh

```
#!/bin/bash

# Usage: ./mountNFS.sh machine-name

machineName=$1

PORTA_HOST_DIR='/home/adam/docker/portainer/data'
PORTA_GUEST_DIR='/var/lib/boot2docker/portainerdata'
PORTA_MOUNT_OPT='-o rw,async,noatime,rsize=32768,wsize=32768,proto=tcp'

#Mount NFS for this session
docker-machine ssh $machineName sudo mkdir /var/lib/boot2docker/portainerdata
docker-machine ssh $machineName sudo /usr/local/etc/init.d/nfs-client start
docker-machine ssh $machineName sudo mount 192.168.42.1:$PORTA_DATA_DIR $PORTA_GUEST_DIR $PORTA_MOUNT_OPT

#Make the NFS mount boot persistent
docker-machine ssh $machineName sudo chmod 777 /var/lib/boot2docker/profile
docker-machine ssh $machineName "echo 'sudo /usr/local/etc/init.d/nfs-client start'"
docker-machine ssh $machineName "echo 'sudo mount 192.168.42.1:$PORTA_DATA_DIR $PORTA_GUEST_DIR $PORTA_MOUNT_OPT' >> /var/lib/boot2docker/pro
```

Portainer telepítése

A portainer-t swarm service-ként fogjuk telepíteni. Ki fogjuk kötni hogy csak manager node-ra telepítheti a swarm, tehát biztos, hogy az el?bb létrehozott mg0, mg1 és mg2 valamelyikén fog landolni. Fontos, hogy megadjuk, hogy csak egy példány jöhessen bel?le létre. Két fontos mount-ot kell beállítani:

1. **/data**: ezt a guest VM /var/lib/boot2docker/portainerdata mappájába fogjuk felcsatolni, ahova az NFS megosztás mount-olva van.
2. **/var/run/docker.sock**: ezt a guest VM docker socket-jére kell rákötni, hogy a portainer hozzáférjen a docker/swarm adatokhoz és tudja is módosítani azokat.

```
eval $(docker-machine env mg0)

docker service create \
--name portainer \
--publish 9000:9000 \
--replicas=1 \
--constraint 'node.role == manager' \
--mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
--mount type=bind,src=/var/lib/boot2docker/portainerdata,dst=/data \
portainer/portainer -H unix:///var/run/docker.sock
```

Note
A **/var/lib/boot2docker/portainerdata** VM mappába mount-oltuk a távol NFS megosztás **/home/adam/docker/portainer/data** mappáját, és erre mount-oltuk rá a swarm service-ben a portainer **/data** mappáját

Netshare

http://netshare.containx.io/

A Netshare docker plugin-el háromféle network fájlrendszert mount-olhatunk közvetlen a docker konténerekbe. Ezek:

- NFS v3/4
- AWS EFS
- CIFS

A Netshare a legegyszer?bb az itt bemutatott megoldások közül. Nem kell telepíteni, nem rendelkezik parancssoros interfésszel sem.

Támogat több docker cluster eszközt is: Mesos/Marathon és Docker Swarm

Installálás a boot2docker VM-en

A netshare -nek egy futtatható bináris állománya van. Nincs más dolgunk, mint hogy ezt letöltsük a boot2docker VM-ekbe, és ott elindítsuk. Ekkor létre fog hozni egy volume plugin-t a docker-ben. Fontos, hogy privilegizált felhasználóként futtassuk, hogy hozzá férjen a docker plugin mappájához. A **boot2docker** mindent módosítást kitöröl újraindításkor, kivéve a **/var/lib/boot2docker** mappa tartalmát. Csak az marad meg, amit ide írnunk. Tehát a netshare-t is ide kell kicsomagolni, és innen kell futtatni.

```
docker@mg0:~$ cd /var/lib/boot2docker
docker@mg0:~$ sudo wget https://github.com/ContainX/docker-volume-netshare/releases/download/v0.35/docker-volume-netshare_0.35_linux_amd64.ta

docker@mg0:~$ sudo tar -xvzf docker-volume-netshare_0.35_linux_amd64.tar.gz
docker@mg0:~$ sudo mv docker-volume-netshare_0.35_linux_amd64 netshare
docker@mg0:~$ cd netshare/
```

A **docker-volume-netshare** futtatásakor meg kell adni az els? argumentumban, hogy az adott **netshare** példány milyen megosztást támogasson az adott VM-en. Ez lehet **NFS**, **AWS EFS** vagy **CIFS**. Minden egyes megosztási fajtára indíthatunk egy netshare példányt. A **boot2docker** csak az **nfs3** protokollt támogatja, ezért fontos, hogy a **netshare** indításakor megadjuk az nfs verziót, különben az nfs4-et fogja elindítani, ami boot2docker alatt nem m?ködik.

```
$ sudo ./docker-volume-netshare nfs -v 3
INFO[0000] == docker-volume-netshare :: Version: 0.35 - Built: 2018-01-27T22:43:03-08:00 ==
INFO[0000] Starting NFS Version 3 :: options: ''
```

Induláskor beírja magát ide: **/run/docker/plugins**, ezért nem kell külön docker plugin-ként telepíteni.

```
# docker-machine ssh mg0 sudo ls /run/docker/plugins
nfs.sock
```

Docker standalone

```
# docker run -i -t --name ubuntu --volume-driver=nfs -v 192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/portainer/data/:/data ub
root@5e82828c7a37:/#

...
INFO[0060] Mounting NFS volume 192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/portainer/data/ on /var/lib/docker-volumes/netsh
...



root@5e82828c7a37:/# ls /data
adam



# docker rm -f ubuntu

...
INFO[0388] Unmounting volume name 192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/portainer/data/ from /var/lib/docker-volumes/n
...
```

Volume létrehozása

```
# docker volume create -d nfs --name portainerdata -o share=192.168.42.1:/home/adam/Projects/DockerCourse/persistentstore/portainer/data
portainerdata

# docker volume ls
DRIVER              VOLUME NAME
nfs                 portainerdata
```

Majd mount-oljuk föl az új nevesített volume-ot:

```
# docker run -i -t --name ubuntu --volume-driver=nfs -v portainerdata:/data ubuntu /bin/bash
root@5e82828c7a37:/#
```

Swarm service

```
docker service create \
--name portainer \
--publish 9000:9000 \
--replicas=1 \
--constraint 'node.role == manager' \
--mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
--mount type=volume,volume-driver=nfs,src=192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/portainer/data/,dst=/data \
portainer/portainer -H unix:///var/run/docker.sock

# docker volume ls
DRIVER              VOLUME NAME
nfs                 192.168.42.1/home/adam/Projects/DockerCourse/persistentstore/portainer/data/
```

Warning
Valami olyat vettem észre, hogy ha swarm service-ként is fel van csatolva egy mappa, akkor azt nem lehet egy standalone swarm konténerbe is felcsatolni: **failed: Device or resource busy**

Viszont még egy VM-en belül is, akár több konténerhez is fel lehet ugyan azt csatolni

Futtatás a háttérben

```
sudo chmod 777 /var/lib/boot2docker/profile
sudo echo 'cd /var/lib/boot2docker/netshare' >> /var/lib/boot2docker/profile
sudo echo 'sudo nohup ./docker-volume-netshare nfs -v 3 &' >> /var/lib/boot2docker/profile
```

A logokat a **/var/log/boot2docker.log** fájlba fogja írni, nem a **nohup.out**-ba.

Note
Ha újra indul a VM (és vele együtt a netshare) akkor az összes megosztás elveszik, mivel a netshare nem perzisztálja a rajta létrehozott megosztásokat. Ez a swarm service-eket kicsit sem zavarja, mert ha a swarm service nem indul el, akkor indít bel?le egy új példányt a swarm, ami létre fogja hozni megint a megosztást, és újra fel fogja csatolni a távoli mappát. Ugyanez nem elmondható a standalone konténerekr?l, amiket a **docker run**-al hoztunk létre. Ezekb?l nem fog új példány indulni, viszont nem fognak tudni elindulni, mivel a megosztás már nem létezik a netshare dirver-ben. Ezeket újra létre kell hozni (lehet hogy a **--restart always** segíthet ezen)

Automatikus telepítés

installNetshare.sh

```
#!/bin/bash


# Usage: ./installNetshare.sh machine-name

machineName=$1

#Download Netshare binary into the users home directory (/home/docker)
docker-machine ssh $machineName "sudo wget https://github.com/ContainX/docker-volume-netshare/releases/download/v0.35/docker-volume-netshare_

#Unpack Netshare
docker-machine ssh $machineName "sudo tar -xvzf docker-volume-netshare_0.35_linux_amd64.tar.gz"
docker-machine ssh $machineName "sudo mv docker-volume-netshare_0.35_linux_amd64 /var/lib/boot2docker/"
docker-machine ssh $machineName "sudo chmod 777 /var/lib/boot2docker/docker-volume-netshare_0.35_linux_amd64/"

#Start NFS during system start up
docker-machine ssh $machineName "sudo chmod 777 /var/lib/boot2docker/profile"
docker-machine ssh $machineName "sudo echo 'cd /var/lib/boot2docker/docker-volume-netshare_0.35_linux_amd64' >> /var/lib/boot2docker/profile"
docker-machine ssh $machineName "sudo echo 'sudo nohup ./docker-volume-netshare nfs -v 3 &' >> /var/lib/boot2docker/profile"

#Restart the sytem for the modifications to take effect
virsh shutdown $machineName
sleep 20s
virsh start $machineName
```

Note
Fontos, hogy **shutdown**-al állítsuk le a VM-et, ne **destroy**-al (force off), mert akkor nem fogja tudni a boot2docker elmenteni a változtatásokat, és a netshare installációnk el fog veszni.

Sajnos a **docker-machine ssh machine command** paranccsal nem lehet valamiért elindítani a **netshare**-t. Tehát ez a formula nem m?ködik:

```
docker-machine ssh mg0 sudo nohup nohup ./docker-volume-netshare nfs -v 3 &
```

Ugyan nem dob hibát, de nem is csinál semmit. Így muszáj újraindítani a VM-et, hogy a /**profile** fájlban lév? **netshare** indítás lefusson. Lehet hogy valahogy újra lehetne tölteni a profile-t, de itt a **boot2docker**-ban semmi nem megy könnyen.

Ezen felül azt is észrevettem, hogy ha a **docker-machine restart** paranccsal indítom újra a VM-et néha elveszik minden módosítás, hiba írtam azokat a perzisztens mappába (/var/lib/boot2docker). Ezrét inkább a **virsh**-val indítom újra.

Érdemes lenne monitorozni, hogy elindult e már a VM miel?tt tovább megyünk. Vagy hogy tényleg leállt e miel?tt elindítjuk.

installNetshare.sh futtatása az összes node-on.

```
for machine in $(docker-machine ls -q); do
        ./installNetshare.sh $machine
done
```

Ellen?rizzük, hogy minden node-on elindult e a plugin:

```
# docker-machine ssh mg0 ps -ef | grep netshare
 1793 root     ./docker-volume-netshare nfs -v 3
 1833 root     ./docker-volume-netshare nfs -v 3
```

Convoy

https://github.com/rancher/convoy

A Convoy 4 féle volume kezelést támogat:

- Device Mapper
- NFS
- Amazon EBS
- Digital Ocean

Az NFS kezelése eltér a Netshare plugin-nál látottaktól, ugyanis nem közvetlen a konténerbe csatolja fel megosztást. A guest VM-en létre fel kell csatolni egy NFS megosztást, ahol a Convoy a volume-okat fogja tárolni a konténer számára transzparens módon.

REX-Ray

https://rexray.readthedocs.io/en/latest/

Trident