Docker (Swarm) and NFS volumes

Posted by: Anthony Dahanne in DevOps November 22nd, 2017

Investigating how to use shared volumes with Docker (Swarm), I decided to take a look at NFS volum on premises way to share folders.

With Docker, you have 3 different syntaxes to mount NFS volumes :

- simple container (via docker volume create + docker run)
- single service (via docker service create)
- complete stack (docker deploy -f stack.yml)

I actually had some trouble mounting NFS volumes, especially with images that COPY files into decla

So, without further ado…

Mounting a NFS share to a Docker container

Let's start creating a folder under our nfs share (/nfs)

```
#(host) mkdir /nfs/test
#(host) ls -al
drwxr-xr-x 2 nfsnobody nfsnobody 4096 Nov 18 18:37 test
#(host) touch /nfs/test/hello-test
```

Now we have a file named « hello-test » under /nfs/test/

Let's create a new docker volume linked to this nfs share, and then let's spin up a new container that

```
#(host) docker volume create --driver local --opt type=nfs --opt
o=addr=nfs.my.corporate.network,rw --opt device=:/nfs/test test
 test
#(host) docker run --rm -it -v test:/data alpine
#(container) touch /data/hello-from-container
#(container) ls -al /data
-rw-r--r-- 1 nobody nobody 0 Nov 19 02:44 hello-from-container
-rw-r--r-- 1 nobody nobody 0 Nov 19 02:43 hello-test
#(host) ls -al /nfs/test/
-rw-r--r-- 1 nfsnobody nfsnobody 0 Nov 18 18:44 hello-from-containe
-rw-r--r-- 1 nfsnobody nfsnobody 0 Nov 18 18:43 hello-test
```

So far, so good ! (did you notice the host root user mapping to nfsnobody and the container root user this a bit later)
Things unfortunately get weirder when you mount to a defined VOLUME location.
For example, with this Dockerfile :

```
FROM alpine
VOLUME /data
```

```
#(host) docker build -t volume:defined .
#(host) docker run --rm -it -v test:/data alpine
```

```
VOLUME /data
```

```
#(host) docker build -t volume:defined .
#(host) docker run --rm -it -v test:/data alpine
```

Everything works great.
Now if you try to do the same thing with an image that copied something to this volume :

```
FROM alpine
VOLUME /data/
COPY empty.file /data/empty.file
```

```
#(host) docker build -t volume:defined-with-copy .
```

and try to map /data to the nfs volume, this time you'll get :

```
2
#(host) docker run --rm -it -v test:/data volume:defined-with-copy
docker: Error response from daemon: chown /var/lib/docker/volumes/t
permitted.
```

To get out of this issue, you need the nocopy option :

```
#(host) docker run --rm -it -v test:/data:nocopy volume:defined-wit
#(container) ls -al /data
-rw-r--r-- 1 nobody nobody 0 Nov 19 02:44 hello-from-container
-rw-r--r-- 1 nobody nobody 0 Nov 19 02:43 hello-test
```

Notice how the original empty.file from the Docker file was « nocopy »ed
So we got away with our issue, well, except something that was provided by the image (empty.file), is

Let's clean up first the previous experimentation

```
docker volume rm test
test
```

and move on to services.

Mounting a NFS share to a Docker service

This use case is more interesting than the previous one, since you only need to mount once to the se
any container created by this service, on any host part of the swarm. (here the service will just ls /data
fancy in entrypoint overriding from a service creation – the parsed command is … not obvious…)

```
# docker service create --mount 'type=volume,src=test,volume-driver
```

```
# docker service logs -f test
test.1.4wteqs8zsd99@worker01 | hello-from-container
test.1.4wteqs8zsd99@worker01 | hello-test
```

So far so good, but now with the infamous VOLUME with copy :

```
docker service create --mount 'type=volume,src=test,volume-driver=l
```

The service won't create the container ! To see that, use service ps :

```
docker service create --mount 'type=volume,src=test,volume-driver=l
```

The service won't create the container ! To see that, use service ps :

```
# docker service ps test --no-trunc
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS
i3 test.1 volume:defined-with-copy worker01 Ready Rejected 2 seconc
/volumes/test/_data: operation not permitted"
```

So adding a volume-nocopy=true should solve the issue :

```
# docker service create --mount 'type=volume,src=test,volume-driver
```

Since, per the documentation :

« By default, if you attach an empty volume to a container, and files or directories already existed at th

```
dst
```

), the Engine copies those files and directories into the volume, allowing the host to access them. Set

```
volume-nocopy
```

to disables copying files from the container's filesystem to the volume and mount the empty volume. A

and indeed, the service is now started :

```
# docker service logs -f test
test.1.on5ahp585oqy@worker01 | hello-from-container
test.1.on5ahp585oqy@worker01 | hello-test
```

now, to have this service interact with others, let's create a stack

Mounting a NFS share in a Docker Stack

To create a stack, we'll create a docker-compose.yml file this time :

```
version: '3.4'

networks:
  terracotta-net:
    driver: overlay

volumes:
  test:
    driver: local
    driver_opts:
      type: nfs
      o: addr=nfs.my.corporate.network,rw
      device: ":/nfs/test"
services:
  producer:
    image : volume:defined-with-copy
    command:  |
      /bin/sh -c "
        while true;
          do touch /data/hello-from-producer;
```

```
    image : volume:defined-with-copy
    command:  |
      /bin/sh -c "
        while true;
          do touch /data/hello-from-producer;
          echo '/data/hello-from-producer written';
          sleep 5;
          rm /data/hello-from-producer;
          echo '/data/hello-from-producer deleted';
          sleep 5;
        done
      "
    volumes:
      - type: volume
        source: test
        target: /data
        volume:
          nocopy: true
  consumer:
    image : volume:defined-with-copy
    command :  |
      /bin/sh -c "
        while  true; do ls -al /data/;  sleep  1;  done
      "
    volumes:
      - type: volume
        source: test
        target: /data
        volume:
          nocopy: true
```

So a producer will write a file to the nfs share, and a consumer will ls -al the folder where the file is su and see how well that goes :

```
# docker stack deploy test --compose-file docker-compose.yml
Creating network test_default
Creating service test_producer
Creating service test_consumer
# docker service logs -f test_producer
test_producer.1@worker01 | /data/hello-from-producer written
test_producer.1@worker01 | /data/hello-from-producer deleted
[...]
# docker service logs -f test_consumer
test_consumer.1@worker02 | drwxr-xr-x 2 nobody nobody 4096 Nov 20 (
test_consumer.1@worker02 | drwxr-xr-x 19 root root 218 Nov 20 04:3
test_consumer.1@worker02 | total 4
test_consumer.1@worker02 | drwxr-xr-x 2 nobody nobody 4096 Nov 20 (
test_consumer.1@worker02 | drwxr-xr-x 19 root root 218 Nov 20 04:4(
test_consumer.1@worker02 | -rw-r--r-- 1 nobody nobody 0 Nov 20 04:4
test_consumer.1@worker02 | total 4
[...]
```

So that worked pretty well : you'll notice the nocopy syntax that is this time :

```
volumes:
  - type: volume
    source: test
    target: /data
    volume:
      nocopy: true
```

If you make the **mistake** of putting it like this :


```
volumes:
```

If you make the **mistake** of putting it like this :

```
volumes:
  - type: volume
    source: test
    target: /data:nocopy
```

you'll probably end up with :

```
worker01 Shutdown Failed 19 seconds ago "starting container failed:
```

since there is no folder named /nfs/test:nocopy !

Was that so terrible ?

Well, the different syntaxes are a bit confusing, and the error messages sometimes are…

Oh, I almost forgot, about the users ownerships : if you run an image that creates a new user and use
write the file to the NFS share : well, according to your NFS share setup, it could be that the user crea
whole different user than nfsnobody !
In that case make sure all your images create this user the same way on Docker nodes that are confi
nicely with the NFS share !