RX-M

Docker In Practice

Lab – Volumes

Docker makes it possible to mount external directories within a container. Any mount coming from outside the container is known as a volume. Volumes circumvent the unioning file system mechanism improving performance. Volumes also have a lifespan independent of the containers that use them, making them the best choice for durable application state in most cases.

Data volumes provide several useful features for persistent or shared data:

- Volumes are initialized when a container is created. If the container's base image contains data at the specified mount point, that existing data is copied into the new volume upon volume initialization (Note that this does not apply when mounting a host directory)
- Data volumes can be shared and reused among containers
- Changes to a data volume are made directly
- Changes to a data volume will not be included when you update an image
- Data volumes persist even if the container itself is deleted
- Data volumes are designed to persist data, independent of the container's life cycle, Docker therefore never automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container

In this lab you will get a chance to work with data volumes in combination with the Docker registry image. In previous examples our registry containers have saved image data internally, causing it to be deleted when the registry is deleted. By using a data volume we can preserve our registry's images even when the registry container is deleted.

1. Using external Volumes

We can avoid the possible loss of important registry images by mapping the registry containers image storage path to the host system's file system. In production this could be a path hosted by a SAN or RAID subsystem with robust durability specifications.

Create a directory on the host to house the registry server's images (keep in mind that this path can be backed by a locally attached disk, ssd or an NFS, iSCSI or FC mount):

```
user@ubuntu:~$ mkdir images

user@ubuntu:~$ ls -l images
total 0

user@ubuntu:~$ cd images

user@ubuntu:~/images$ pwd
/home/user/images

user@ubuntu:~/images$ cd ..
user@ubuntu:~$
```

Most images that intend to have certain paths mounted via volumes define those volumes in metadata. Display the configured volumes for the registry image:

```
user@ubuntu:~$ docker inspect -f '{{.Config.Volumes}}' registry
map[/var/lib/registry:{}]
```

Now run the registry container mounting the host path (/home/user/images) onto the container volume path (/var/lib/registry):

```
user@ubuntu:~$ docker run -p 5000:5000 --name reg_svr -d -v /home/user/images:/var/lib/registry
registry:2
45c60ed5766eb621553312cfbe8fec5e18589ad01fc83329878d26f82261fb6c
```

To test the server, push an image:

```
user@ubuntu:~$ docker push localhost:5000/bb:test
The push refers to a repository [localhost:5000/bb] (len: 1)
ac6a7980c6c2: Pushed
c00ef186408b: Pushed
test: digest: sha256:e563c38b3882bdf9c3af05f31cf1a22900083e0a9403e6e5dc5efaa51aa11535 size: 2728
```

Notice that the new server has never seen this image before and requested we upload it. Now take a look in the Docker host `~/images` directory:

```
user@ubuntu:~$ ls -l images
total 4
drwxr-xr-x 3 root root 4096 Dec 30 17:42 docker

user@ubuntu:~$ ls -l images/docker/
total 4
drwxr-xr-x 3 root root 4096 Dec 30 17:42 registry

user@ubuntu:~$ ls -l images/docker/registry/
total 4
drwxr-xr-x 4 root root 4096 Dec 30 17:42 v2

user@ubuntu:~$ ls -l images/docker/registry/v2/
total 8
drwxr-xr-x 3 root root 4096 Dec 30 17:42 blobs
drwxr-xr-x 3 root root 4096 Dec 30 17:42 repositories

user@ubuntu:~$ ls -l images/docker/registry/v2/repositories/
total 4
drwxr-xr-x 5 root root 4096 Dec 30 17:42 bb
```

Our important registry data is now secured outside of the container. To simulate a service failure stop and remove the registry server container:

```
user@ubuntu:~$ docker stop reg_svr
reg_svr

user@ubuntu:~$ docker rm reg_svr
reg_svr
```

Examine the images directory on the host:

```
user@ubuntu:~$ ls -l images/docker/registry/v2/repositories/
total 4
drwxr-xr-x 5 root root 4096 Dec 30 17:42 bb
```

Knowing that our data is safe we can simply rerun the *reg_svr* to bring our service back online:

```
user@ubuntu:~$ docker run -p 5000:5000 --name reg_svr -d -v /home/user/images:/var/lib/registry
registry:2
c096be34196f5b08069d73d129127b6899f5013a4c6dfa656666c3329a710616
```

With the server running try to upload the same image you previously uploaded:

```
user@ubuntu:~$ docker push localhost:5000/bb:test
The push refers to a repository [localhost:5000/bb] (len: 1)
ac6a7980c6c2: Image already exists
c00ef186408b: Image already exists
test: digest: sha256:e563c38b3882bdf9c3af05f31cf1a22900083e0a9403e6e5dc5efaa51aa11535 size: 2728
```

Note that the new server recovered the images from the host Volume used by the previous instance, allowing it to inform us that the images we just tried to push were already on the server.

To display the Volume mount points in use by a container you can inspect the Mounts metadata:

```
user@ubuntu:~$ docker inspect -f '{{.Mounts}}' reg_svr
[{ /home/user/images /var/lib/registry   true}]
user@ubuntu:~$ docker inspect reg_svr | grep -A7 Mounts
    "Mounts": [
        {
            "Source": "/home/user/images",
            "Destination": "/var/lib/registry",
            "Mode": "",
            "RW": true
        }
    ],
```

Here we can see that the /home/user/images directory on the host appears as the "/var/lib/registry directory inside the container.

Congratulations! You have completed the Volume lab!!