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:~/websvr$ cd

user@ubuntu:~$ mkdir images

user@ubuntu:~$ ls images/
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 image inspect -f '{{json .Config.Volumes}}' registry:2 | jq
```

```
.

{
  "/var/lib/registry": {}
}

user@ubuntu:~$
```

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

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

1987e6fcca06257fc532976c5b2ba42f2b64958847ac207a954854edd07f49c0

user@ubuntu:~$
```

To test the server, push an image:

```
user@ubuntu:~$ docker image push localhost:5000/devenv:latest

The push refers to a repository [localhost:5000/devenv]
fd5ec982e7c8: Pushed
bd00cdbae641: Pushed
af43131c4039: Pushed
9bd4c7af882a: Pushed
04ab82f865cf: Pushed
c29b5eadf94a: Pushed
latest: digest:
sha256:3083b0d5d114200b6a17cdcb9d5a301086440a3b1aff420457c639a54ecedcef size:
1571

user@ubuntu:~$
```

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:~$ sudo apt-get install tree

user@ubuntu:~$ tree -d images/

images/
└── docker
    └── registry
        └── v2
            ├── blobs
            │   └── sha256
            │       ├── 0e
            │       │   └──
0e4fa347e95dcc0e86c4721597369ee2cfe58b07e080eb5b976ea208f0835360
```

```
            │       ├── 2f
            │       │   └──
2f0af262ade799215323703723e717e526613c910f96df3408d8bf63b414980b
            │       ├── 30
            │       │   └──
3083b0d5d114200b6a17cdcb9d5a301086440a3b1aff420457c639a54ecedcef
            │       ├── 39
            │       │   └──
393440c87ab49d14b617199b95fffbfd9bb89166eed8ea4464c2ffd15500fe84
            │       ├── 4b
            │       │   └──
4b7dcc92c29519d4e4f8d32848902733e99b87de1794b4953a8a81bbde4dafa3
            │       ├── 95
            │       │   └──
958046a70c9d6a8160f8f4fb8f81fff3cdb85bceb6206b5842a8f988493425f7
            │       ├── a9
            │       │   └──
a994fca0b2a328c2e14dc93b4ae43b033fcfb726a73d22e38f27b5b9ab2f6f71
            │       └── d2
            │           └──
d24bc6f195c3546ffad21f479176fecee0b8a6770b852e35d6c45186608de767
            └── repositories
                └── devenv
                    ├── _layers
                    │   └── sha256
                    │       ├──
0e4fa347e95dcc0e86c4721597369ee2cfe58b07e080eb5b976ea208f0835360
                    │       ├──
2f0af262ade799215323703723e717e526613c910f96df3408d8bf63b414980b
                    │       ├──
393440c87ab49d14b617199b95fffbfd9bb89166eed8ea4464c2ffd15500fe84
                    │       ├──
4b7dcc92c29519d4e4f8d32848902733e99b87de1794b4953a8a81bbde4dafa3
                    │       ├──
958046a70c9d6a8160f8f4fb8f81fff3cdb85bceb6206b5842a8f988493425f7
                    │       ├──
a994fca0b2a328c2e14dc93b4ae43b033fcfb726a73d22e38f27b5b9ab2f6f71
                    │       └──
d24bc6f195c3546ffad21f479176fecee0b8a6770b852e35d6c45186608de767
                    ├── _manifests
                    │   ├── revisions
                    │   │   └── sha256
                    │   │       └──
3083b0d5d114200b6a17cdcb9d5a301086440a3b1aff420457c639a54ecedcef
                    │   └── tags
                    │       └── latest
                    │           ├── current
                    │           └── index
                    │               └── sha256
                    │                   └──
3083b0d5d114200b6a17cdcb9d5a301086440a3b1aff420457c639a54ecedcef
                    └── _uploads

43 directories

user@ubuntu:~$
```

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 container stop reg_svr

reg_svr

user@ubuntu:~$ docker container rm reg_svr

reg_svr

user@ubuntu:~$
```

Examine the images directory on the host:

```
user@ubuntu:~$ ls -l images/docker/registry/v2/

total 8
drwxr-xr-x 3 root root 4096 Mar  7 22:33 blobs
drwxr-xr-x 3 root root 4096 Mar  7 22:33 repositories

user@ubuntu:~$ ls -l images/docker/registry/v2/repositories/
total 4
drwxr-xr-x 5 root root 4096 Mar  7 22:33 devenv

user@ubuntu:~$
```

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

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

c518926776b4746c8e45e9078759892e3eac5fc85a931b15192b9fac128c682d

user@ubuntu:~$
```

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

```
user@ubuntu:~$ docker image push localhost:5000/devenv:latest

The push refers to a repository [localhost:5000/devenv]
fd5ec982e7c8: Layer already exists
bd00cdbae641: Layer already exists
af43131c4039: Layer already exists
9bd4c7af882a: Layer already exists
04ab82f865cf: Layer already exists
c29b5eadf94a: Layer already exists
latest: digest:
```

```
sha256:3083b0d5d114200b6a17cdcb9d5a301086440a3b1aff420457c639a54ecedcef size:
1571

user@ubuntu:~$
```

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 container inspect -f '{{json .Mounts}}' reg_svr | jq

[
  {
    "Type": "bind",
    "Source": "/home/user/images",
    "Destination": "/var/lib/registry",
    "Mode": "",
    "RW": true,
    "Propagation": ""
  }
]

user@ubuntu:~$
```

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

```
user@ubuntu:~$ docker container exec reg_svr ls
/var/lib/registry/docker/registry/v2/repositories

devenv

user@ubuntu:~$
```

Congratulations! You have completed the volume lab!