QA Consulting

Docker Volumes

TRAINING MATERIALS - MODULE HANDOUT

Contacts

robert.crutchley@qa.com
team.qac.all.trainers@qa.com

www.consulting.qa.com

Contents

Overview

Volumes are another way in Docker to persist data from the container. Unlike Bind Mounts, Volumes are managed by Docker. Bind Mounts rely on a host's directory structure. Volumes are a newer addition to Docker compared Bind Mounts, because of this they have more functionality and benefits. The main benefit about using volumes is that they are just easier to manage. Bind Mounts are great for just plugging one file into one container but when you want to start sharing files or directories across multiple containers then volumes becomes a much more manageable solution.

Creating and Managing Volumes

Before even creating a container you can create and manage volumes in Docker.

Create a Volume

Just provide the name to create a volume in Docker.

```
docker volume create [VOLUME_NAME]
```

```
docker volume create my-volume
```

List the Existing Volumes

We can view a list of all the volumes available on the Docker host.

```
docker volume ls
```

Remove a Volume

Provide the name of the volumes to delete it.

```
docker volume rm [VOLUME_NAME]
```

```
docker volume rm my-volume
```

View more information about a Volume

The inspect command can be used to see more details about a volume in Docker.

```
docker volume inspect [VOLUME_NAME]
```

```
docker volume inspect my-volume
```

Mounting Volumes

Just like with Bind Mounts, you can choose to use either the volume or mount flags. Mounting the volumes into containers is very similar to creating Bind Mounts, except for the source we are pointing to the volume that we want to be mounted not a location on the host file system. Options for mounting volumes as read only are available, just like with Bind Mounts.

When specifying a volume name with any of these commands, if the volumes doesn't exist then Docker will create one for you. This is handy feature as it allows us to skip a step if need be, but make sure to spell your volume name correctly because Docker won't care if you don't.

Using the --volume Flag

When using the volume flag we need provide the name of the volume that is going to be mounted and the location which the volume is going to be mounted in on the container.

```
docker run --volume [VOLUME_NAME]:[LOCATION_ON_CONTAINER]
```

```
docker run --volume my-volume:/usr/share/nginx/html
```

Using the --mount Flag

With the mount flag provide the volume name and the destination on the container to mount the volume.

```
docker run --mount source=[VOLUME_NAME],destination=[LOCATION_ON_CONTAINER]
```

```
docker run --mount source=my-volume,destination=/usr/share/nginx/html
```

Volume Drivers

Whilst we won't be getting hands on with volume drivers here, they are definitely worth knowing about for future reference. When you do a listing for all the existing volumes, you might have noticed a driver column called local. This basically means that the volume is stored on the host machine. You may at some point want to develop a solution where the volume can be stored on a remote host or in a cloud storage solution perhaps. Plugins will allow you run different drivers to attach volumes to remote places like NFS servers or cloud storage.

Tasks

This exercise will get you to create and manage volumes in Docker. You will be able to see that data can be persisted after a container is destroyed and how a single volume can be used across multiple containers at the same time. NGINX will serve as another good tool demonstrate this, we will create a simple webpage for NGINX to serve and store it on the volume. If the NGINX container is stopped and removed, when it is created again the same webpage will be served. If another NGINX server is created with the volume, the same webpage will be accessible from that instance of NGINX as well.

Create a New Volume

We are going to need a volume, whether its created separately now for while it is being mounted as well in the next step is up to you. A suggested name for the volume could be webpage.

Create an NGINX Container

Create an NGINX container with the volume mounted to /usr/share/nginx/html in the container. Make sure that you publish the port (80) so that the NGINX service is available from the host.

Create a Webpage

Let's make a change to the default NGINX home page, so it is our page. You will need to connect to the container that you created and edit the index.html file, replacing the entire contents with our one. For that however we need a text editor such as vim or nano. The latest NGINX Docker image is based on Debian so you can use the apt package manager to install one of these.

```
# update
apt update
# install vim
apt install -y vim
# edit the index.html file
vim /usr/share/nginx/html/index.html
```

/usr/share/nginx/html/index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>NGINX</title>
</head>
<body>
       <h3>index.html file stored in a Docker Volume</h3>
</body>
</html>
```

Recreate the NGINX Container

Once you have updated the index.html, stop and remove the NGINX container and then create it again. When you go to access NGINX with a HTTP request, you will see that your index.html is being served to you still, even though the container we created it in was destroyed.

Start Another NGINX Container

Create another NGINX container using the same volume configurations and publish it to a different port on your host, when you connect to that instance of NGINX you will see your index.html there as well.

Make a Change to the Webpage

Connect to the second NGINX container that you created, install a text editor and make a change to the `/usr/share/nginx/html/index.html` file inside the `<h3>` tag. The changes you make should be reflected on both of the containers when you make a HTTP request to them.

Persist Jenkins Jobs Configurations

Stop and remove all containers, delete the volume that was created.
Try to persist all the jobs and configurations for Jenkins, so that when the container has been stopped and removed, Jenkins can pick up where it left off when you create the container again.
The default Jenkins home folder in the Jenkins Docker image is /var/jenkins_home, so this is the folder that you will want to look at persisting.