Using Volumes

In this lesson, you will learn how to use volumes to prevent data loss.

When a container writes files, it writes them *inside* of the container. Which means that when the container dies (the host machine restarts, the container is moved from one node to another in a cluster, it simply fails, etc.) all of that data is lost. It also means that if you run the same container several times in a load-balancing scenario, each container will have its own data, which may result in inconsistent user experience.

A rule of thumb for the sake of simplicity is to ensure that containers are stateless, for instance, storing their data in an external database (relational like an SQL Server or document-based like MongoDB) or distributed cache (like Redis). However, sometimes you want to store files in a place where they are persisted; this is done using volumes.

Using a volume, you map a directory inside the container to a persistent storage. Persistent storages are managed through drivers, and they depend on the actual Docker host. They may be an Azure File Storage on Azure or Amazon S3 on AWS. With Docker Desktop, you can map volumes to actual directories on the host system; this is done using the *-v* switch on the *docker run* command.

Suppose you run a MySQL database with no volume:

```
docker run -d mysql:5.7
```

Any data stored in that database will be lost when the container is stopped or restarted. In order to avoid data loss, you can use a volume mount:

```
docker run -v /your/dir:/var/lib/mysql -d mysql:5.7
```

It will ensure that any data written to the */var/lib/mysql* directory inside the container is actually written to the */your/dir* directory on the host system. This ensures that the data is not lost when the container is restarted.

In the next lesson, you will be introduced to registries.