# Working with Databases

## Overview

In this lab you'll see how to run MongoDB in a container, and optionally how to persist its data in an external data storage area such as a Docker volume.

## Source folders

* (There are no source code files in this lab)

## Roadmap

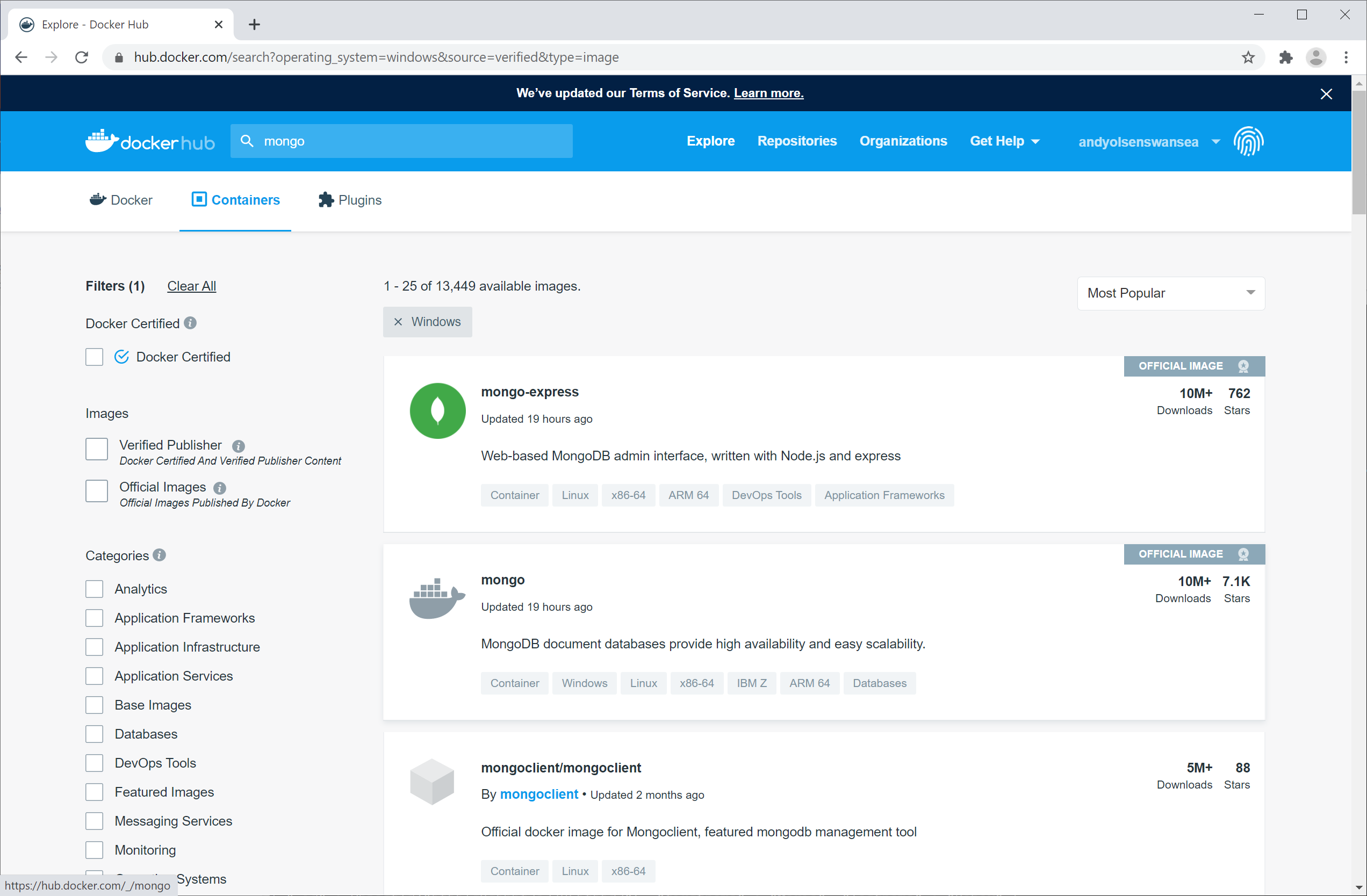
There are 6 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Running MongoDB in a container
2. Verifying the MongoDB container is running
3. Exec'ing into the MongoDB container
4. Opening a MongoDB shell and running commands to manipulate data
5. Observing the transient nature of containers
6. (If time permits): Mapping the MongoDB data directory to a volume

## Exercise 1: Running MongoDB in a container

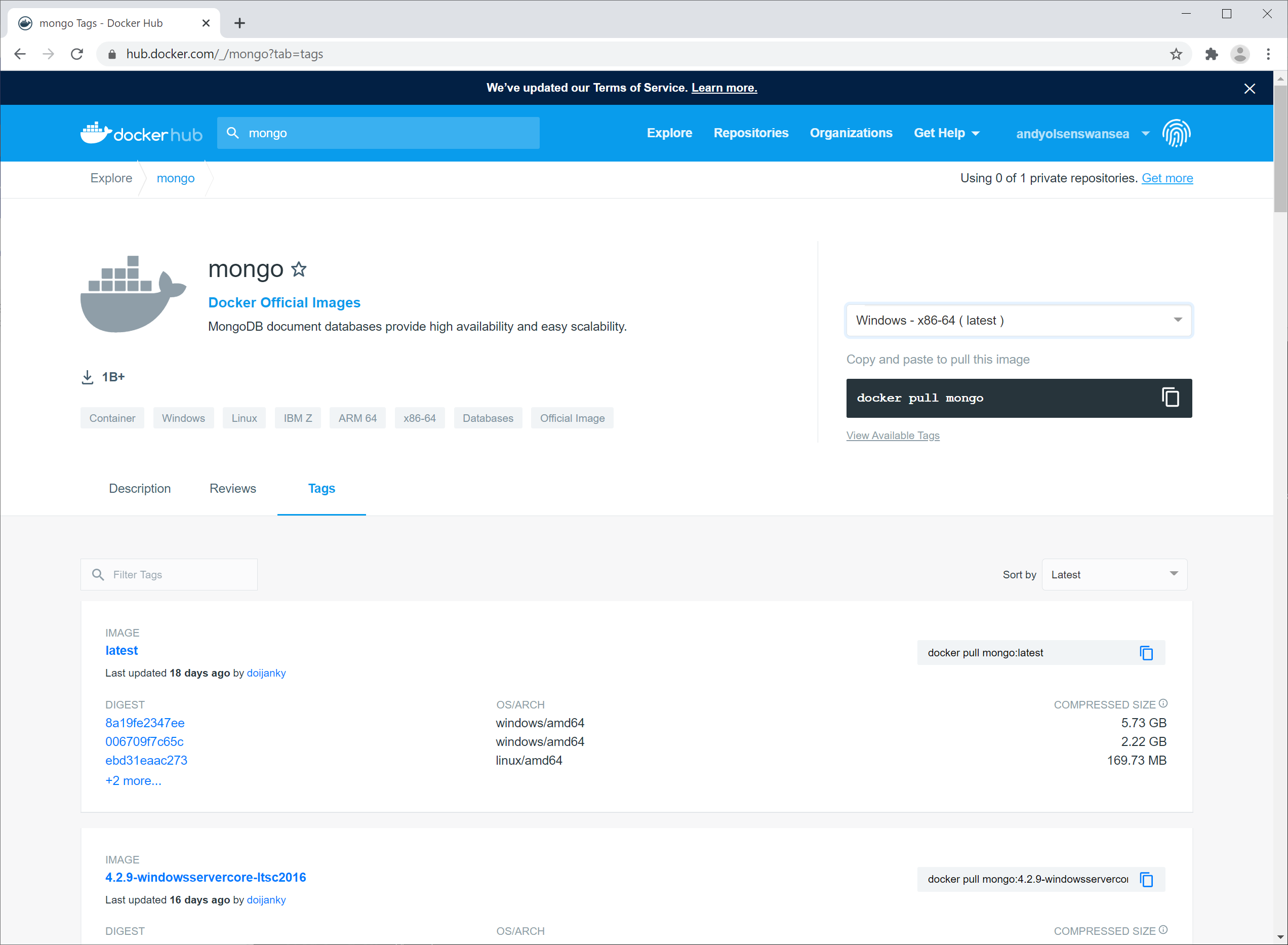
One of the great benefits of containerization is that you can quickly spin-up containers running database engines, web servers, message queue brokers, etc. For example, you can easily start and stop a MongoDB container whenever you like - there's no need to install MongoDB on your actual host computer.

There are Docker images available for almost every server, platform, or library you care to mention. Your starting point for finding images is Docker Hub at <https://hub.docker.com/>. Go to Docker Hub now and search for *mongo*.



The image you want is mongo, so click this entry in the list.

You'll now see a page like this, showing all the versions of the mongo image (each version is called a "tag"). To use the correct terminology, we say mongo is a "repository" storing all the different "tags" of the *mongo* image. There are lots of tags available, e.g. mongo:latest:



When you want to download an image from a Docker registry such as Docker Hub, you can either specify the latest tag (i.e. mongo:latest), a precise tag (e.g. mongo:4.2.9) or just the repository name itself (i.e. mongo). In the latter case, you'll get the latest version.

You can run a Docker container directly based on an image in Docker Hub, if that suits your purposes (i.e. you don't have to write a Dockerfile if you're happy to just use an off-the-shelf image). With this in mind, go into 'Play with Docker' and enter the following command:

docker run -d -p 27017:27017 --name mongodb mongo:4.2.9

The first time you run this command, Docker will download the mongo image (tag 4.2.9 here) from Docker Hub. Docker will then start a container based on this image, as follows:

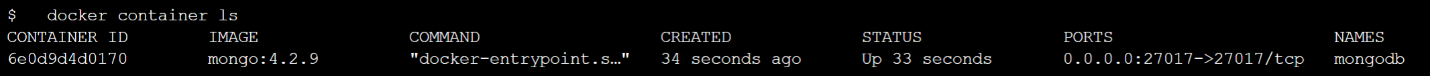
* The --name option specifies the name of the container (mongodb here).
* The -d option causes the container to run in detached mode.
* The -p option maps port 27017 on the host computer to port 27017 within the container.

**Exercise 2:** **Verifying the MongoDB container is running**

Run the following command to list Docker containers:

docker container ls

All being well, you should see your mongodb container is running smoothly:



**Exercise 3:** **Exec'ing into the MongoDB container**

Run the following command to exec into the MongoDB container in interactive mode, and to open a Linux Bash shell inside the container:

docker exec -it mongodb bash

You're now inside the MongoDB container. You can run Linux commands such as the following, to poke around the file system within the MongoDB container:

pwd

ls

ls data

ls data/db

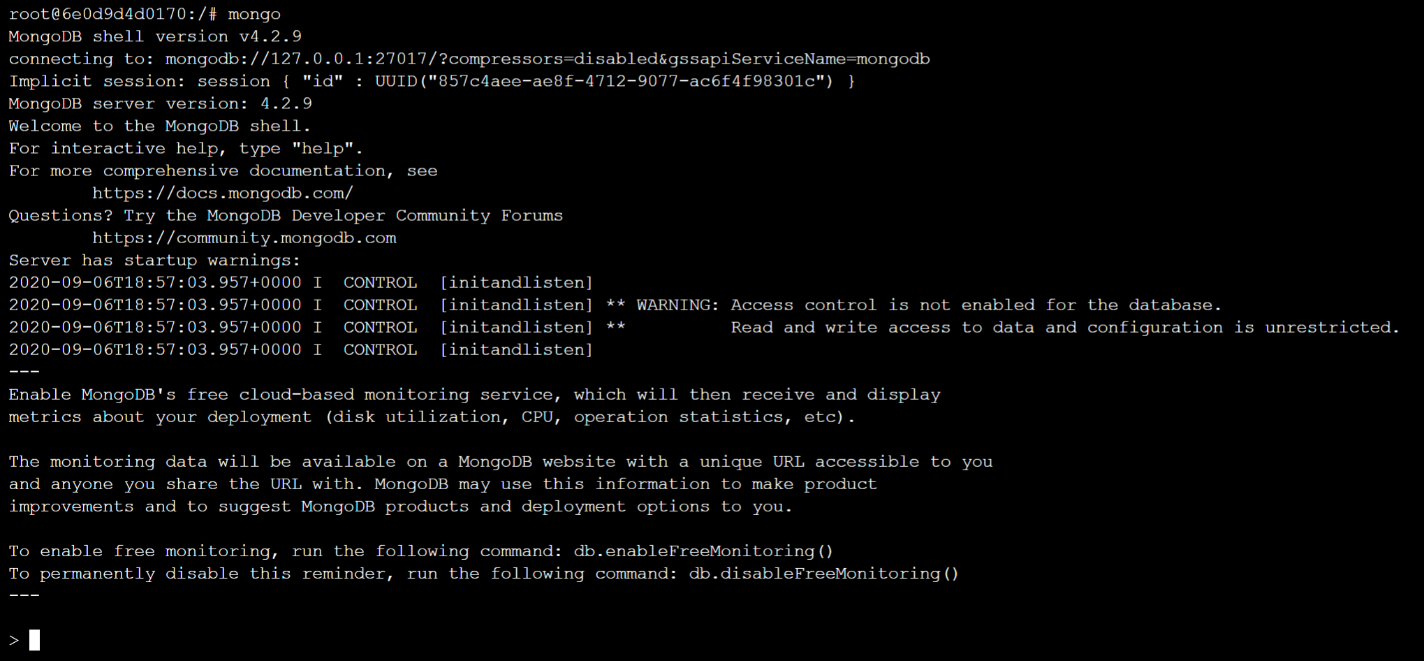
The /data/db directory is where MongoDB stores its data.

## Exercise 4: Opening a MongoDB shell and running commands to manipulate data

In the previous exercise you exec'd into the MongoDB container and had a poke around its filesystem. Now enter the following command, to start a MongoDB shell within the container:

mongo

This opens a MongoDB shell, connected to the MongoDB database server running inside the container. You should see a message such as the following:



Enter the following command to show the databases inside MongoDB:

show dbs

Now run the following command to create a new database named mydb inside MongoDB:

use mydb

Now run the following command to insert two documents (i.e. records) into a collection named people (this collection doesn't exist yet, so MongoDB will create it automatically):

db.people.save({ firstname: "Donald", lastname: "Duck" })

db.people.save({ firstname: "Mickey", lastname: "Mouse" })

Now run the following command to find all people whose first name is Donald:

db.people.find({ firstname: "Donald" })

Then type exit to leave the MongoDB shell back to the Bash shell, and then type exit again to leave the Bash shell and return back to main command line.

## Exercise 5: Observing the transient nature of containers

In the previous exercise you did the following within the MongoDB container:

* Created a database named mydb
* Implicitly created a collection named people in the mydb database
* Inserted two documents into the people collection

As we discussed earlier, this data resides in the /data/db directory in the MongoDB container. This data is transient - as soon as the container is shut down, you lose the data. To prove this point, do the following…

First, forcibly stop and remove your MongoDB container as follows:

docker container rm -f mongodb

Now start the MongoDB container again as follows:

docker run -d -p 27017:27017 --name mongodb mongo:4.2.9

Now exec into the MongoDB container as follows:

docker exec -it mongodb bash

Once you're in the Bash shell, run the following command to open a MongoDB shell:

mongo

Once you're in the MongoDB shell, type the following command to get a list of databases:

show dbs

Notice that the MongoDB container doesn't have a database named mydb. You created this database in Exercise 4 above, but the data resided inside the container's filesystem. As soon as the container stops, it's *Goodnight Vienna* for the data inside it.

In order to achieve proper persistence (always a good characteristic in a database), one solution is to map the data directory to an external and permanent location such as a Docker volume. You'll do this in the next exercise, if you have time.

## Exercise 6 (If time permits): Mapping the MongoDB data directory to a volume

In this exercise, you'll create a Docker volume and map your MongoDB container's data directory to the volume. Thus whenever you stop or restart a MongoDB container, its data will be stored in the persistent volume rather than in the ephemeral filesystem of the container itself.

Here's a high-level summary of the steps you need to perform. Consult the materials in the chapter notes if you need a reminder of how to do anything:

* First, forcibly stop and remove your current MongoDB container.
* Create a Docker volume, e.g. named mongovol.
* Start a new instance of your MongoDB container, and map its /data/db directory to your volume.

Now do the following, to insert some data into the MongoDB container:

* Exec into your MongoDB container, and then open a MongoDB shell.
* Create a new database, create a collection, and insert some documents. All being well, this data will actually be stored persistently in the volume.
* Exit the MongoDB shell, and then exit the Bash shell.

And now the acid test - was the data persisted properly? Let's see:

* Forcibly stop and remove your current MongoDB container.
* Start a new instance of your MongoDB container, and again map its /data/db directory to your volume.
* Exec into your MongoDB container, and then open a MongoDB shell.
* Run various MongoDB shell commands to verify the data changes you made above have been persisted successfully.