A whistle-stop tour of Docker

What is Docker?

Docker is containerisation software that allows us to package up software and apps with all of their dependencies in a lighweight manner that allows us to share containers in a scalable and easy manner.

Docker commands

Get a list of all the docker images:

```
(base) NarwhalMacBookPro:docker_experiments abh$ docker image ls
REPOSITORY                  TAG                 IMAGE ID       CREATED         SIZE
getting-started             latest              f4db23aa0846   3 hours ago     64.7MB
custom-drupal               latest              0208a940b37e   6 hours ago     483MB
andreper/spark-worker       3.0.0               801335c4586a   2 days ago      1.24GB
andreper/jupyterlab         3.0.0-spark-3.0.0   150c9a796d9a   2 days ago      2.09GB
andreper/spark-master       3.0.0               db12bb655d9c   2 days ago      1.24GB
rabbitmq                    3-management        3fc359184bad   2 days ago      187MB
postgres                    9.6-alpine          67ff6cfc14c1   11 days ago     37.2MB
mongo                       latest              b8264a857eba   12 days ago     449MB
cassandra                   3.11.10             87779a1dd1f7   3 weeks ago     402MB
jupyter/pyspark-notebook    latest              cf1ced85fc59   4 weeks ago     3.5GB
ubuntu                      latest              f63181f19b2f   5 weeks ago     72.9MB
continuumio/miniconda3      latest              b4adc22212f1   11 months ago   429MB
hello-world                 latest              bf756fb1ae65   14 months ago   13.3kB
mongo                       3.4                 ab4287b7a939   19 months ago   428MB
node                        6-alpine            dfc29bfa7d41   22 months ago   56.1MB
drupal                      8.2                 68b9d32702ee   3 years ago     447MB
(base) NarwhalMacBookPro:docker_experiments abh$
```

`docker image ls`

Start a container running the `hello-world` image

```
(base) NarwhalMacBookPro:docker_experiments abh$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:7e02330c713f93b1d3e4c5003350d0dbe215ca269dd1d84a4abc577908344b30
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

(base) NarwhalMacBookPro:docker_experiments abh$
```

`docker run hello-world`

Try and remove the docker image we just downloaded

`docker image rm hello-world`

This fails and we get an error message like:

`Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container 6d51de6adb50 is using`

This is because there is a "stopped" container based on this image. Even though we launched a container and it immediately stopped there is still a copy of that specific container.

To show a list of running containers we run:

`docker container ls`

But nothing is running, we already know our container stopped and so we need to use the `-a` flag to see **ALL** containers!

`docker container ls -a`

```
(base) NarwhalMacBookPro:docker_experiments abh$ docker container ls -a
CONTAINER ID   IMAGE         COMMAND    CREATED         STATUS                     PORTS     NAMES
afa581a552e4   hello-world   "/hello"   5 minutes ago   Exited (0) 5 minutes ago             gracious_rubin
(base) NarwhalMacBookPro:docker_experiments abh$
```

Since we still have the container there we can also inspect the logs; this would work if the container was running live as well as stopped. There are specific flags or ways if you need to actively monitor logs but we won't worry about that now. We can inspect the logs using

`docker logs <name or container id>`

For our hello-world container (called gracious_rubin in this instance) we see a repeat of the original text that was pushed to us when we ran the programme. By default Docker logs everything that goes to *stdout*.

```
(base) NarwhalMacBookPro:docker_experiments abh$ docker container ls -a
CONTAINER ID   IMAGE         COMMAND    CREATED         STATUS                     PORTS     NAMES
afa581a552e4   hello-world   "/hello"   5 minutes ago   Exited (0) 5 minutes ago             gracious_rubin
(base) NarwhalMacBookPro:docker_experiments abh$ docker logs gracious_rubin

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

(base) NarwhalMacBookPro:docker_experiments abh$
```

Tidying up after ourselves. It's easy to fill your system with experiments so make sure to clean up when you don't need containers/images anymore.

To remove our container we can use either the name or container id:

`docker container rm gracious_rubin`

if the container was still running and we want to force removal we would need the `-f` flag

`docker container rm gracious_rubin`

Once we don't have any containers using an image anymore we can then remove that image using:

`docker image rm hello-world`

A practical example with MongoDB

We will start a mongodb version 3.4 server running and then write and read data to it from Python.

`docker run -d --publish 27017:27017 --name myMongoDB mongo:3.4`

This will run Mongo in a detached mode in the background and will open port 27017 to our computer.

Testing our database out using Python

From Python we can connect to our database and add records and delete records (see the notebook:).

The code we used is summarised below

```
import pandas as pd
from pymongo import MongoClient

# This establishes a connection to Mongo

client = MongoClient('localhost', 27017)

# Connect to a database called "movie_db",
# if it doesn't exist it will create a new database
db = client["movie_db"]

# Connect to a collection called "fave_movies",
# if it doesn't exist it will create a new collection
collection = db["fave_movies"]

# Make some fake data
sample_movie1 = {
    "Title": "Star Wars",
    "Director": "George Lucas",
    "Year": 1977
}

sample_movie2 = {
    "Title": "Tenet",
    "Director": "Christopher Nolan",
    "Year": 2020
}

all_data = [sample_movie1, sample_movie2]

# Bulk insert our list of movies
result = collection.insert_many(all_data)

# Query a random record (its not really random, it is the 1st record)
collection.find_one()

# Extract all the records and put them in a DataFrame
pd.DataFrame(collection.find())
# Show there are 2 records in our collection so far
collection.estimated_document_count()

# Add a new movie with additional fields
sample_movie3 = {
    "Title": "Batman",
    "Director": "Christopher Nolan",
    "Actor": "Christian Bale",
    "Year": 2020,
    "BoxOffice": 10000000
}

result = collection.insert_one(sample_movie3)

# Show the number of records has increased
collection.estimated_document_count()

# If we store the data in a DataFrame we get blank fields
pd.DataFrame(collection.find())

# If we store the data in a list we only get the original fields
for record in list(collection.find()):
    print(record)
```

Shutting everything down!

Once we have finished we need to: 1. Stop the container - `docker container stop myMongoDB` 2. Remove the container - `docker container rm myMongoDB` 3. (Optional) Remove the image if we don't think we need it again - `docker image rm mongo:3.4`