Docker

What Is Docker?

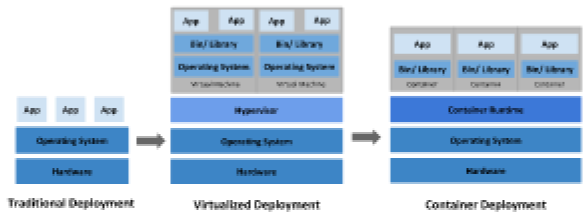
> Docker containerises a program so it will run on any environment by bundling an application and all of its dependencies into a container. It even includes the operating system (OS).

- Standardised development environment across many workstations
- Consistent deployment of applications
- Full reproducibility (e.g. for research)

> A container can be considered a virtual machine (VM); however, whilst a VM virtualises the hardware (the available RAM), a container only virtualises the OS. Note that Docker does not simply make a copy of the OS you want to work with; rather, it provides the necessary tools for working with a specific OS.

> If an application runs in the latest ubuntu version, Docker will not install the latest version each time the application is run. It will simply obtain the tools necessary to run that version without installing an entire OS.

> Furthermore, although the tools need to be installed, Docker installs them only once (unless they are uninstalled) and accesses them the next time the same application is run. To do so, Docker relies on a powerful tool named Docker images, which are the templates for running containers.



> Conventionally, a regular application is quite lightweight during deployment. This is because only the code required to run the application is deployed; however, we are not usually aware of the OS on which the application will be run. To deploy the entire VM, for example, we must install and launch the entire OS every time the application is run. As a solution, containers offer both advantages of the lightweight deployment without needing to install the OS for each instance of the software and the ability to run the application consistently in multiple environments.

- Containers are lighter than VMs.
- They are **immutable**, meaning that their contents remain constant.
- A container can easily be created or destroyed depending on the user's needs.
- All the dependencies of an application can be packed in a container, including the OS.
- Containers are reproducible since they run consistently, regardless of the host OS.
- **Micro-services architecture** is supported. An app can be divided into multiple separate containers that communicate with each other:
  - Only the required image is changed.
  - There is no need to deploy everything anew.
  - The components are easily switchable.

Docker Image

> A Docker *image* is a blueprint that indicates all the steps required to run the container that holds an application. Initially, when the image is `built`, the required tools are downloaded and installed. Subsequently, Docker accesses the tools when the application is run again.

> To ensure efficiency, Docker shares these tools between images; thus, if another image is created where these tools are used again, Docker will not download or install them because it is aware of their location.

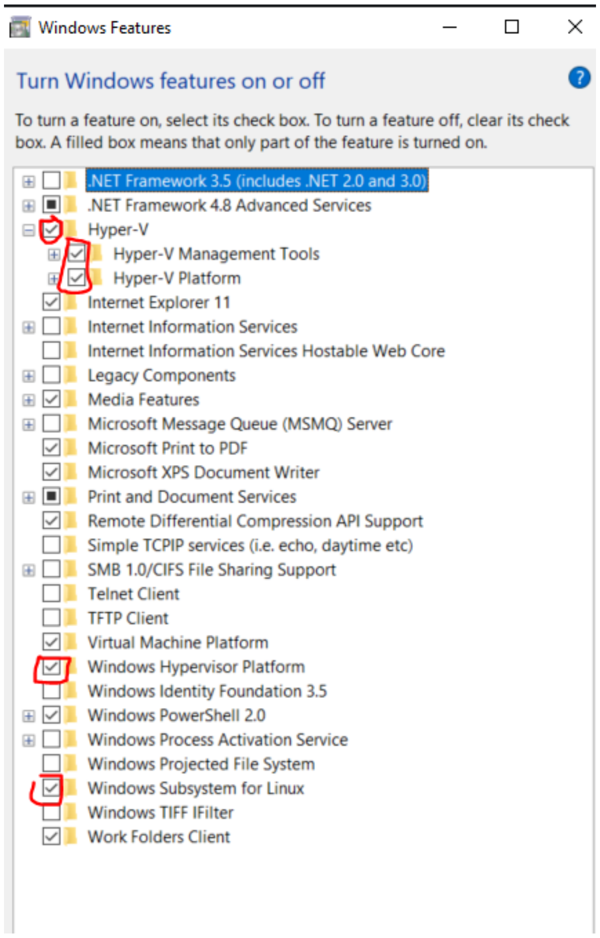
> Docker enables us to package code, apps, etc. with all the necessary dependencies in a self-contained environment called a `Docker image`. Afterward, we can create instances of these images called `Docker containers`

Set Up

> When creating a Docker image, your computer must create containers that can allocate a `memory slot` for running certain tools (even though they are not VMs) that your OS might not support.

> To grant access to these memory slots, your computer must start an engine that creates the containers within your computer. Different OSs have different approaches for carrying out this task. For example, for Windows and iOS, Docker Desktop must be installed, which will handle the creation of the corresponding engine that will create the containers. Conversely, Linux distributions are

> considerably more flexible, and Docker can be run simply by installing the engine that will create the containers.


Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ☐ .NET Framework 3.5 (includes .NET 2.0 and 3.0)
- ■ .NET Framework 4.8 Advanced Services
- ☑ Hyper-V
  - ☑ Hyper-V Management Tools
  - ☑ Hyper-V Platform
- ☑ Internet Explorer 11
- ☐ Internet Information Services
- ☐ Internet Information Services Hostable Web Core
- ☐ Legacy Components
- ☑ Media Features
- ☐ Microsoft Message Queue (MSMQ) Server
- ☑ Microsoft Print to PDF
- ☑ Microsoft XPS Document Writer
- ■ Print and Document Services
- ☑ Remote Differential Compression API Support
- ☐ Simple TCPIP services (i.e. echo, daytime etc)
- ☐ SMB 1.0/CIFS File Sharing Support
- ☐ Telnet Client
- ☐ TFTP Client
- ☑ Virtual Machine Platform
- ☑ Windows Hypervisor Platform
- ☐ Windows Identity Foundation 3.5
- ☑ Windows PowerShell 2.0
- ☐ Windows Process Activation Service
- ☐ Windows Projected File System
- ☑ Windows Subsystem for Linux
- ☐ Windows TIFF IFilter
- ☑ Work Folders Client

1. https://docs.docker.com/desktop/
2. Download for your OS
3. To ensure that the installation was successful, run `docker --version` in the CLI (This applies to all OSs).
4. In the Windows search bar, type 'Turn Windows Features On or Off' and subsequently **enable** the following
5. The latest version of WSL might also be required

Docker Hub

> Docker Hub is a repository for storing Docker images created by users globally.

> Bear in mind that you will utilise base images to create Docker images, and these base images will be stored in Docker Hub.

1. https://hub.docker.com/

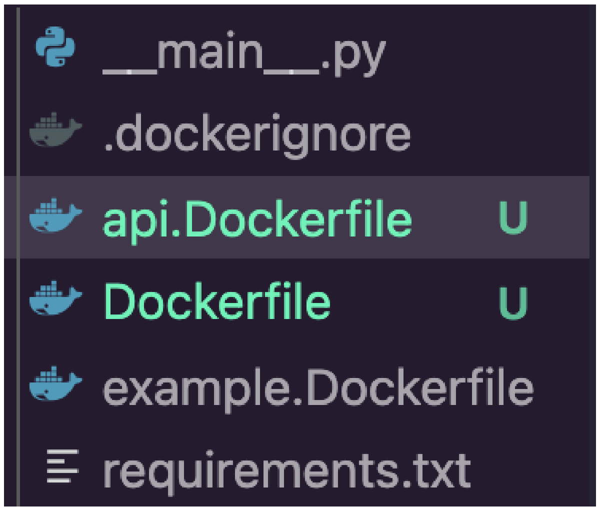
2. Make an account

Docker Images + Dockerfile

Docker images are the instructions required to create an instance of a Docker container.

Thus, Docker images are essentially a set of steps followed by the Docker engine to create the environment for running an application. These steps are declared in a file named `Dockerfile`. Docker searches for this special file whenever an image is to be built.

`Dockerfiles` do not contain any extensions. The name of the file is literally `Dockerfile`; however, an extension may be used. For example, if the Dockerfile specifies the steps for creating an image for an API image, it can be called `api.Dockerfile`.


```
__main__.py
.dockerignore
api.Dockerfile        U
Dockerfile            U
example.Dockerfile
requirements.txt
```

When a Dockerfile is created in VSCode, it will automatically be recognised as a Dockerfile, as indicated by the characteristic whale icon.

Once the Dockerfile is created, you can start containerising your application. However, you need to specify the commands you want Docker to run.

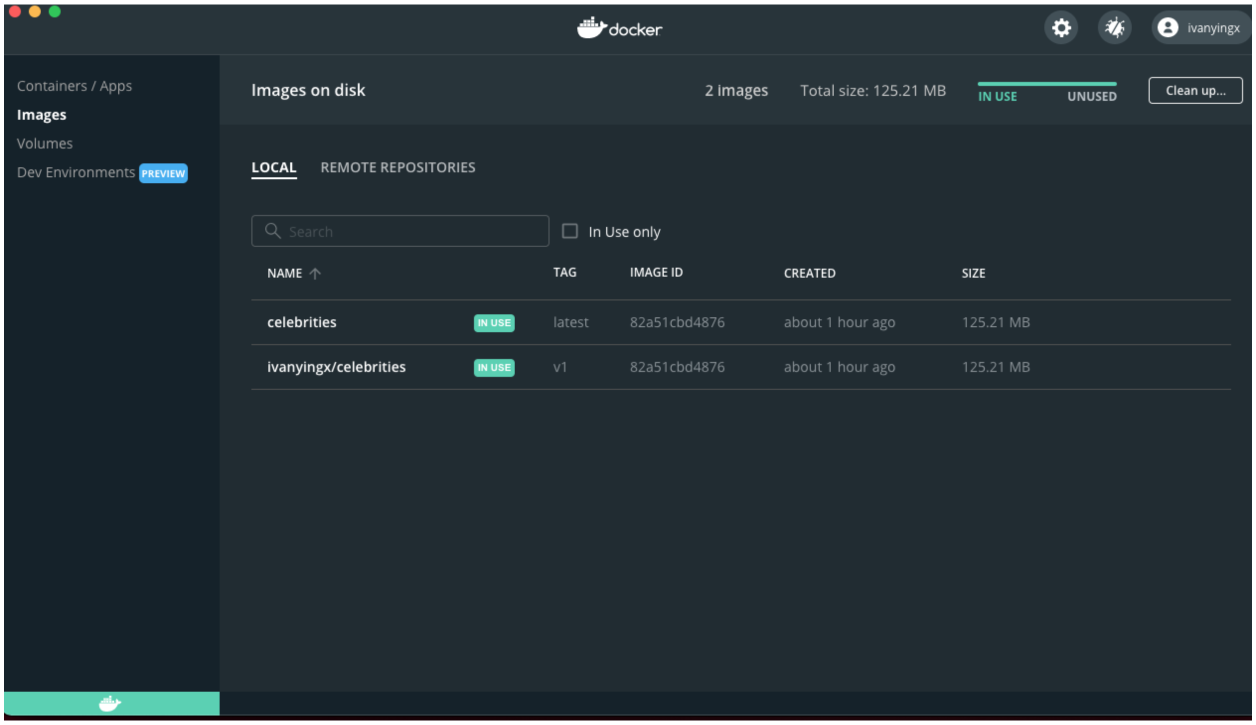
Dockerfiles will contain instructions, such as `FROM`, `RUN`, `CMD`, `COPY`, ... The uppercase words that start each line in the Dockerfile are called **instructions** and are basically commands, followed by arguments (similar to the case in the terminal), which the docker build command knows how to execute. Docker build runs each of these instructions in turn to create the docker image.

```
FROM python:3.8-slim-buster
COPY . .
RUN pip install -r requirements.txt
CMD ["python", "scraper/celebrity_scraper.py"]
```

1. `cd` to the folder
2. Make a Dockerfile
3. Write the file
4. Conventionally, Docker images are built from a pre-built image Docker that can be found on Docker Hub. The pre-built image usually contains some dependencies. A common practice is to use an image with Python installed. You can download and run the pre-built image using the `FROM` clause
5. Install or copy anything of our choosing into the docker container. Remember that the directory you add is relative to the location of the Dockerfile.
6. This will copy everything in the Dockerfile directory (`requirements.txt` and the `scraper` folder) into the container.
7. Understanding this step is extremely important. When an image is built, the relevant files are copied into the container, which is analogous to copying them into a different and separate computer. In other words, it is almost as if there is a separate mini-computer containing the scraper, with Python installed.
8. The first `.` argument following the `COPY` instruction is the location of the assets *on your machine* that you wish to copy. The second `.` argument following the `COPY` instruction is the location where the assets will be copied to *on the Docker container*.
9. As the final step before running the scraper, your python packages must be installed, e.g. `beautifulsoup` and `requests`. Fortunately, the requirements file was also copied into the image; thus, the packages can be installed directly using the `RUN` command, which runs the bash command that it follows.
10. Now, we run the python script. Note that the `RUN` clause is unsuitable here because `RUN` is executed when the image is built. We need a command that is executed when the image is run, and that clause is `CMD`.
11. The `CMD` clause can be declared in many ways. In this case, we employ square brackets, and the first item is the executable (`python`), while the rest are the parameters (files).
12. Build the image. In the CLI, change the directory to `celebrity_example`. Thereafter, use the `build` command from Docker `docker build [OPTIONS] [Dockerfile path]`

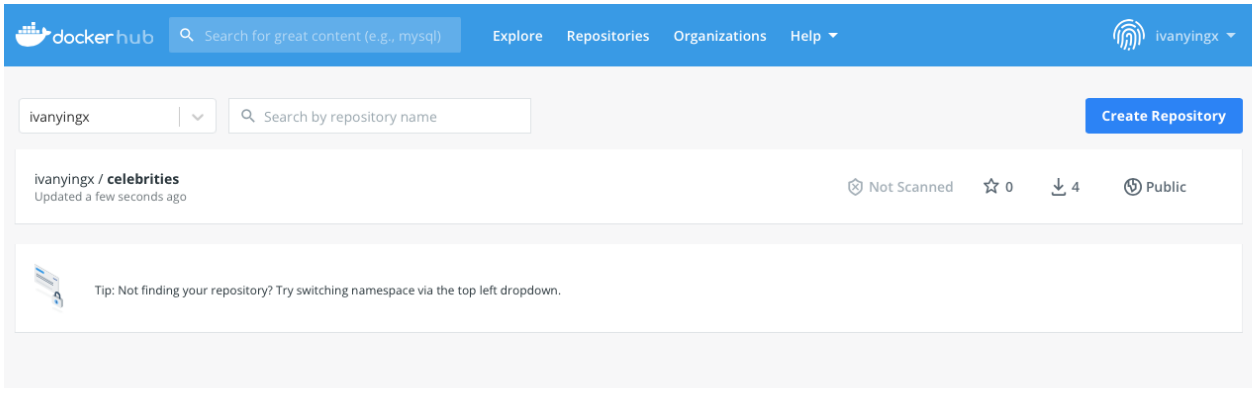
13. The typical naming convention for Docker images is `image_name:version`. Typically we specify the version as `latest` rather than manually writing out the semantic versioning label.
14. Since we are in the same directory as the Dockerfile, the Dockerfile path is simply a dot (`.`).
15. Run the following command in your command line to build the container: `docker build -t celebrities:latest .`
16. Now, we verify if the image was successfully created by running the following command: `docker images`
17. Run it `docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]`
18. Run the image `docker run celebrities` This will throw an error because the script expects an input. However, at present, this is impossible because the image is running in a non-interactive mode. As a solution, we must add the options, `-i` and `-t`: `-i` will keep the STDIN open, and `-t` will make the process interactive.
19. Run `docker run -it celebrities` The image can be used everywhere, regardless of the OS and dependencies installed.
20. Additionally, you can distribute it globally using Docker Hub. To do this, login to the Docker Hub account by running `docker login` and enter your credentials.
21. The images you push to Docker hub need to have a specific name `<username>/<image_name>:<tag>` create a copy of the existing image with a new name using the docker tag command. The Image_id can be observed when `docker image` is run `docker tag 82a51cbd4876 ivanyingx/celebrities:v1` 'ivanyingx' is the username, which you should replace with yours, and 82a51cbd4876 is the Image_id.
22. Confirm that the image has been properly built by running docker images once more. You can also confirm it in Docker Desktop
23. Finally, we push the image to Docker Hub. Pushing an image to Docker hub is similar to pushing a repository to GitHub. Simply use docker push `docker push ivanyingx/celebrities:v1`
24. To verify that your image has been uploaded, go to your Docker hub account

```
docker build [OPTIONS] [Dockerfile path]
docker build -t celebrities:latest

docker images # show our current images on this machine

docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
docker run celebrities
docker run -it celebrities

docker login

<username>/<image_name>:<tag>
docker tag <Image_Id> <New name>
docker tag 82a51cbd4876 ivanyingx/celebrities:v1

docker push ivanyingx/celebrities:v1
```



Images on disk   2 images   Total size: 125.21 MB

NAME		TAG	IMAGE ID	CREATED	SIZE
celebrities	IN USE	latest	82a51cbd4876	about 1 hour ago	125.21 MB
ivanyingx/celebrities	IN USE	v1	82a51cbd4876	about 1 hour ago	125.21 MB



ivanyingx / **celebrities**
Updated a few seconds ago

Not Scanned 0 4 Public

Tip: Not finding your repository? Try switching namespace via the top left dropdown.

If any user wants to run your container, they can do so directly on their local machine. In this case, you can run *ivanyingx's* image as follows:

```
docker pull ivanyingx/celebrities # to download the image
docker run ivanyingx/celebrities # to run the image
docker run ivanyingx/celebrities # runs both
```