



# Tutorial 1: Installation



- Mac: <https://store.docker.com/editions/community/docker-ce-desktop-mac>
- Windows 10 Pro: <https://store.docker.com/editions/community/docker-ce-desktop-windows>
- Windows (other): <https://www.docker.com/products/docker-toolbox>
  - May need to enable virtualisation in BIOS (google how to do this for your machine)
- Linux: <https://docs.docker.com/engine/installation/>
  - Go to “Server” and find your OS.
  - Select “CE” (community edition) if it gives you two choices
  - May also be in your package manager (yum/apt)
- **`docker pull ubuntu`**



# Tutorial 2: Running



- `docker run --rm -it ubuntu bash`
- Run some commands and explore the system, create some files, download some packages, etc.
- In another tab: `docker ps`
- `exit` when complete
- `docker images`



# Tutorial 2: Running



- `docker run --rm -it ubuntu bash`
- `--rm` means the container is deleted after the command stops
  - Data you create in the container is deleted with `rm`.
  - Otherwise have to delete manually later
  - Alternative: use a single long-running container
- `-i` and `-t` make the container properly interactive

# ★ Tutorial 3: Dockerfiles ★

- Add these line to end of your Dockerfile:

```
RUN pip3 install jupyter ipython numpy
```

- Build again:

```
docker build -t my-image .
```

- Run a container:

```
docker run -it --rm my-jupyter ipython
```

# ★ Tutorial 3: Dockerfiles ★

- Lines executed sequentially
- Each line creates a new layer in file system
- -t identifies image with tag



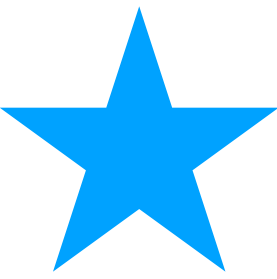
# Tutorial 4: Lifetimes



- `docker run -it ubuntu bash`
- Create some large files and then exit  
e.g. `cat > my.txt`
- `docker ps -as`
- Container is stopped but not removed:
- `docker start -i container_number`
- Can only re-run the same command you started with
- Task: Figure out how to use the `docker rm` command to remove your container



# Tutorial 4: Lifetimes



- While other container is running, new terminal:
- **`docker exec -it container_number bash`**
- Run a new command in a running container



# Tutorial 5: Data



- `docker run -it --rm my-jupyterter bash`

- `ipython`


```
import numpy as np
x=np.random.randn(1000)
np.savetxt("data.txt", x)
```

- Task: figure out how to use the `docker cp` command to get data.txt out




# Tutorial 5: Data Bind mounts

- Make a new directory you want to share:
- `mkdir ./data`
- `docker run -it --rm -v $PWD/data:/data my-jupyter bash`
- `/data` now mounted inside image
- Can use this in clever ways!



# Tutorial 5: Data

## Bind mounts



- `docker run -it --rm -v $PWD/data:/data my-jupyter bash`
- Full path is always required for both elements.
- Can have multiple `-v` commands for more mounts



# Tutorial 6: Jupyter



- `docker run --rm -it -p 8888 my-jupyter  
jupyter notebook --port=8888 --ip=0.0.0.0  
--allow-root`
- Open the URL in your browser - need to change 0.0.0.0 to 127.0.0.1
- Task: Combine these commands with the data mounting we looked at earlier to save notebook and results to host disk.



# Tutorial 6: Jupyter



- `docker run --rm -it -p 8888 my-jupyter`  
*jupyter notebook --port=8888 --ip=0.0.0.0*  
*--allow-root*
- Green italics = command run inside container
  - Could have just put “bash” then typed all this

# Tutorial 7: More Dockerfile Directives

- CMD
- ENV
- COPY
- USER
- Task: Investigate these directives in the Dockerfile documentation
- Task: Use the CMD directive to make the running your notebook easier

# ★ Tutorial 8: Docker Hub ★

- Make an account at [hub.docker.com](https://hub.docker.com)
- `docker build -t username/my-jupyter .`
- `docker push username/my-jupyter`
- Pull your neighbour's image



# Tutorial 9: Exploring Images



- Save your image to disk:  
`docker save -o my-jupyter.tar my-jupyter`
- Extract the tar file:  
`tar -xf my-jupyter.tar`
- Have a look around
- These layers are read-only but can be built on
  - Union File System