Advanced Software Engineering (LAB)

Stefano Forti

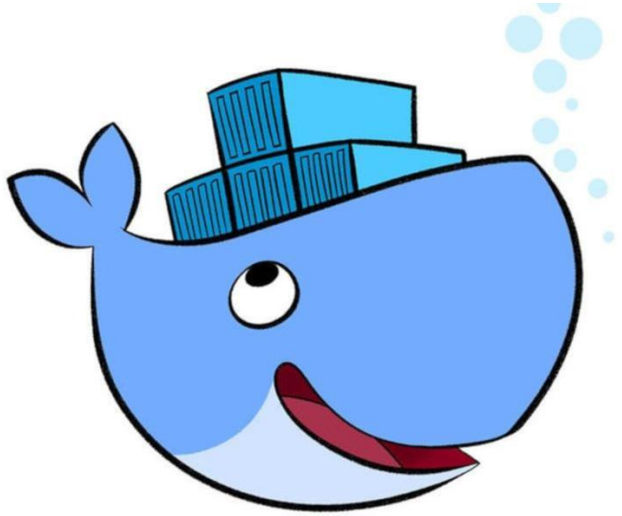
name.surname@di.unipi.it

Department of Computer Science @ University of Pisa

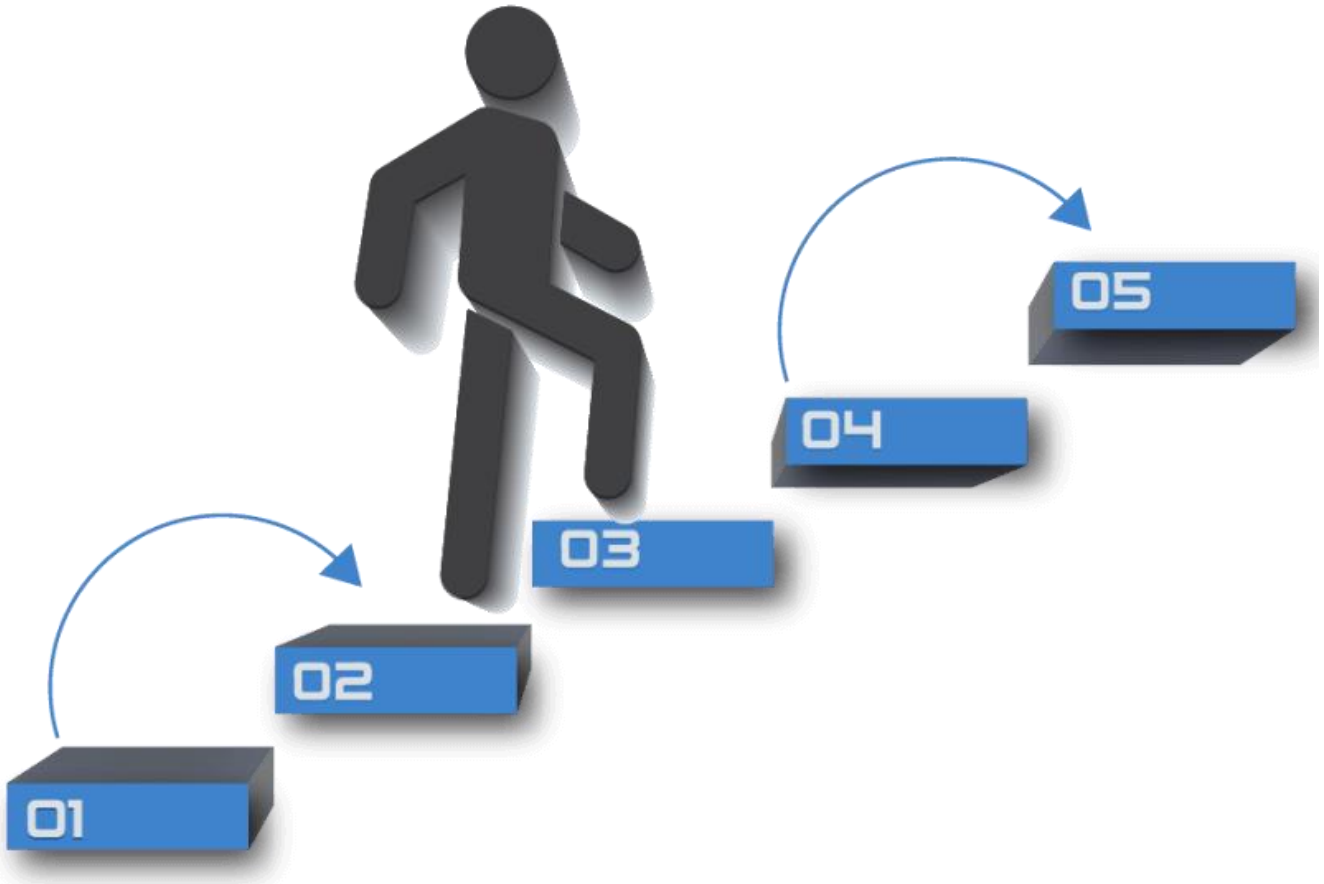

Q&A

What will I do?

- Learn on Docker basic commands.
- Run Docker containers.
- Create Docker images using a `Dockerfile`.
- Use Docker Compose to run multi-container (multi-service) apps.

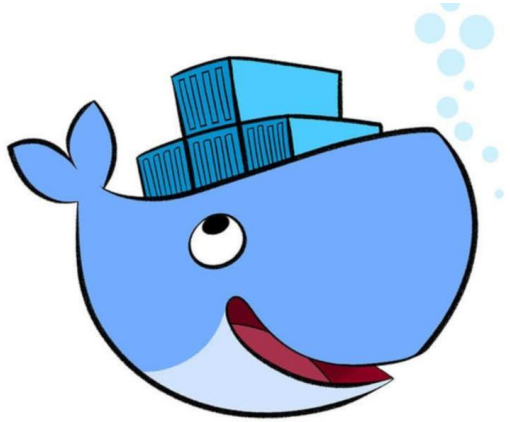


step-by-step



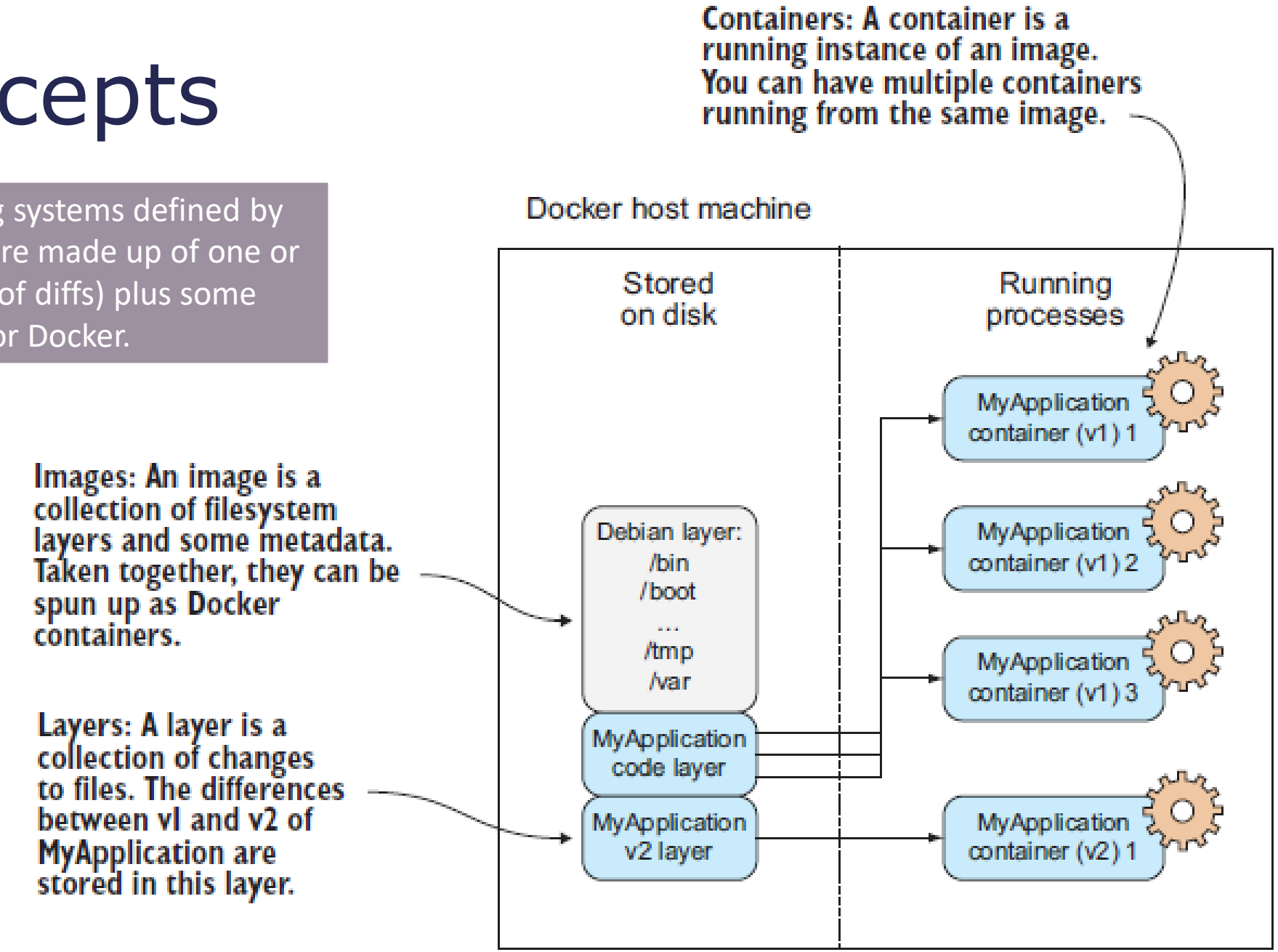
4 more lines on Docker

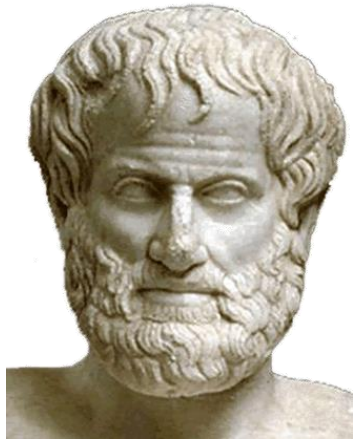
Docker is a platform that allows you to **“build, ship, and run any app, anywhere”**. It has come a long way in an incredibly short time and is now considered a standard way of solving one of the costliest aspects of software: **deployment**.



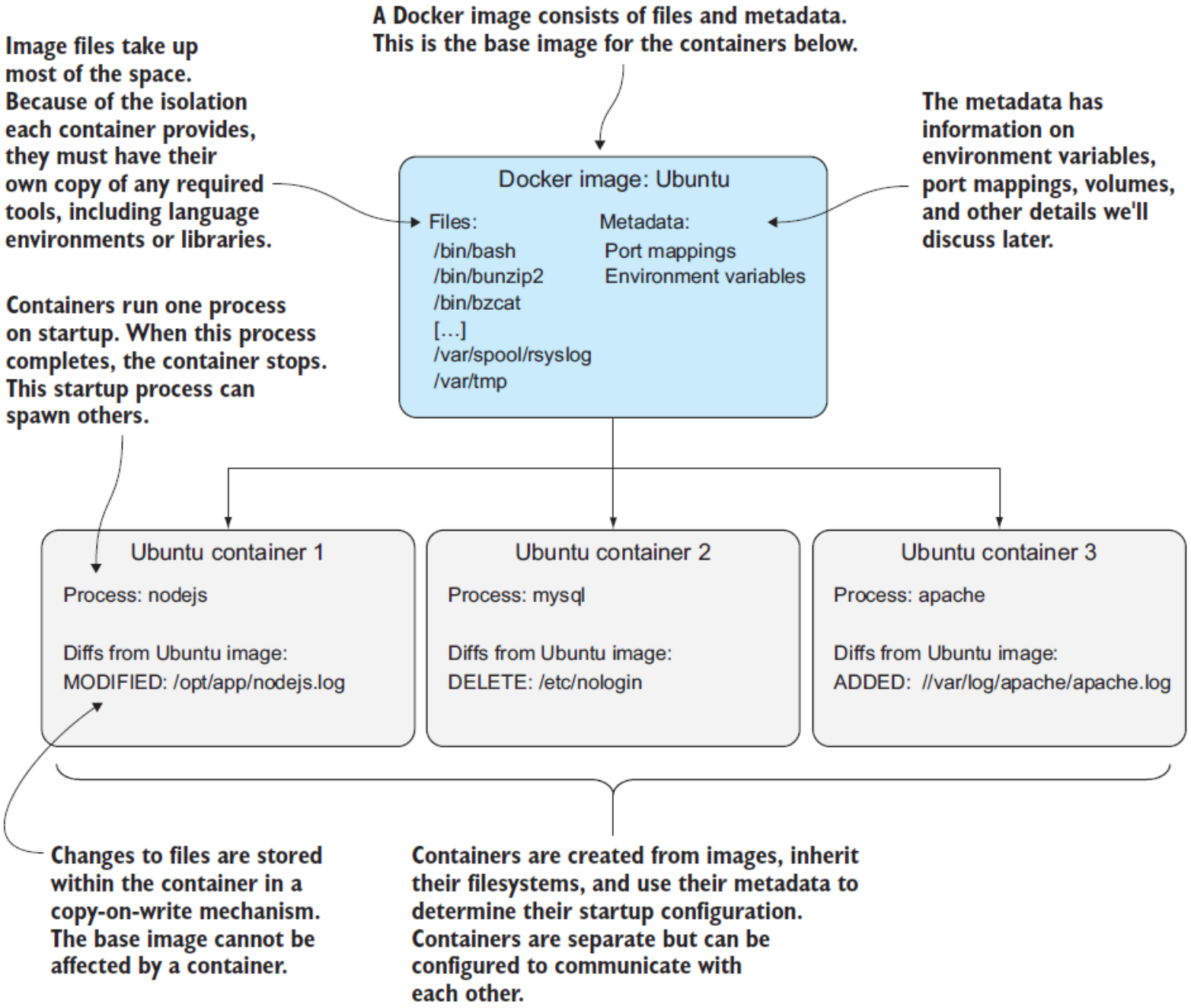
Key concepts

Containers are running systems defined by *images*. These images are made up of one or more *layers* (or sets of diffs) plus some metadata for Docker.





Images vs Containers



Installing Docker (1)

- Remove any older version:

```
sudo apt-get remove docker docker-engine docker.io
```

- If you are using Ubuntu Trusty 14.04, install `linux-image-extra-*`

```
sudo apt-get update
sudo apt-get install \
  linux-image-extra-$(uname -r) \
  linux-image-extra-virtual
```

See also: https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/

Installing Docker (2)

- Update **apt** package index:

```
sudo apt-get update
```

- Install some packages to permit package retrieval over https:

```
sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  software-properties-common
```

- Add Docker's official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
```

See also: https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/

Installing Docker (3)

- Check the fingerprint:

```
$ sudo apt-key fingerprint 0EBFCD88

pub   4096R/0EBFCD88 2017-02-22
      Key fingerprint = 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid                  Docker Release (CE deb) <docker@docker.com>
sub   4096R/F273FCD8 2017-02-22
```

- Install your Docker (see next page if using Ubuntu 18.xy):

amd64:

```
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install docker-ce
```

See also: https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/

Installing Docker (4)

- On latest Ubuntu versions you should install the developers release of Docker, by running:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic test"
sudo apt install docker-ce
```

See also: https://www.thecodecampus.de/blog/install-docker-on-ubuntu-18-10/

Use docker without sudo

- As Docker is a privileged binary, by default we need to prefix commands with **sudo** in order to run.
- To avoid this, it is sufficient to add our user to docker group:

```
sudo groupadd docker
sudo gpasswd –a $USER docker
```

- Logout and login from Linux.
- Then:

```
docker run hello-world
```

Hello, World!

```
ase@ASE-VM: ~
ase@ASE-VM:~$ docker run hello-world

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.
 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://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/

ase@ASE-VM:~$
```

Assessment of Homework 2

YELLOW GROUP A

BUTTER GROUP A

BARLETTA GROUP B

- Objective fullfillment considers all previous runs
- No green (3.*) story implemented
- Coverage 82%

GROUP 1 F

- Commit by contributor external to the group
- Did not implement all red & yellow stories (2.2 not implemented)
- [Not clear whether account deletion always works]
- No green (3.*) story implemented
- Coverage 56%

//YELLOW (6)	
Uccheddu	A
D'Aquino	A
Bongiorno	A
Liut	A
Toloni	A
Boffa	A

//BUTTER (8*)	
Franceschi	A
Baldini	A
Gadler	A
Guglielmo	A
Ottimo	A
Paoletti	A
Tosoni	A

**only 7 delivered*

//BARLETTA (9**)	
Bellomo	B
Frioli	B
Yagublu	B
De Liberali	B
Hajiyev	B
Cincinelli	B
Bruno	B

*** 2 pending evaluations*

GROUP 1 dissolved on Nov16th lab. Gnoiski and Dall'Oro assigned to YELLOW group.

Please recall that the language of our MSs is English.

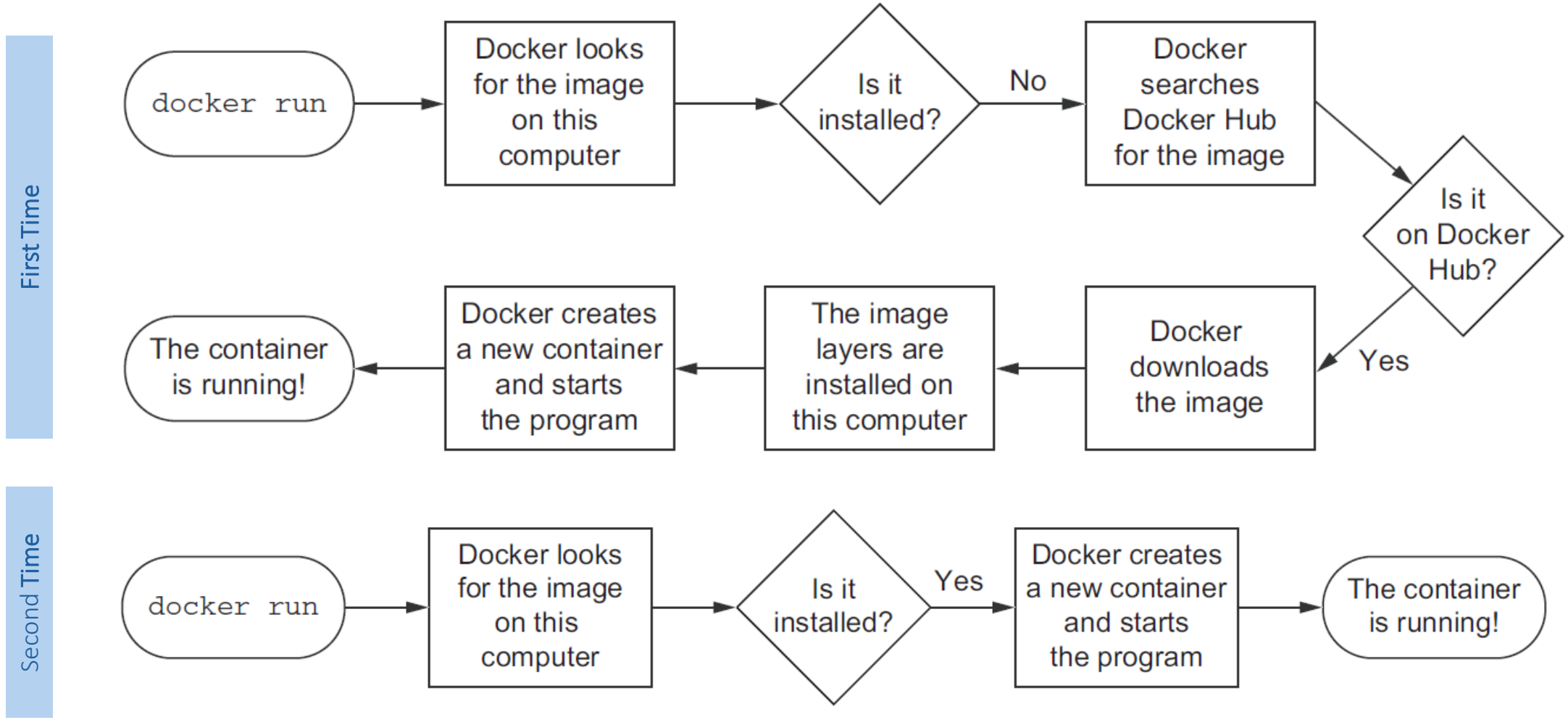
Running your first image

```
docker run debian echo “Hello, World!”
```

- What happened?
  1. We’ve called docker run that launches containers.
  2. The argument debian is the name of the image we want to use (a stripped-down version of Debian Linux distro)
  3. We did not have a local copy, thus Docker downloaded it from the DockerHub.
  4. The image is then turned into a running container.
  5. The specified command **`echo “Hello, World!”`** executes.
  6. The container is shut down.

If you run the same command again, it will be much faster… why?

docker run


First Time
docker run
Docker looks for the image on this computer
Is it installed?
No
Docker searches Docker Hub for the image
Is it on Docker Hub?
Yes
Docker downloads the image
The image layers are installed on this computer
Docker creates a new container and starts the program
The container is running!
Second Time
docker run
Docker looks for the image on this computer
Is it installed?
Yes
Docker creates a new container and starts the program
The container is running!

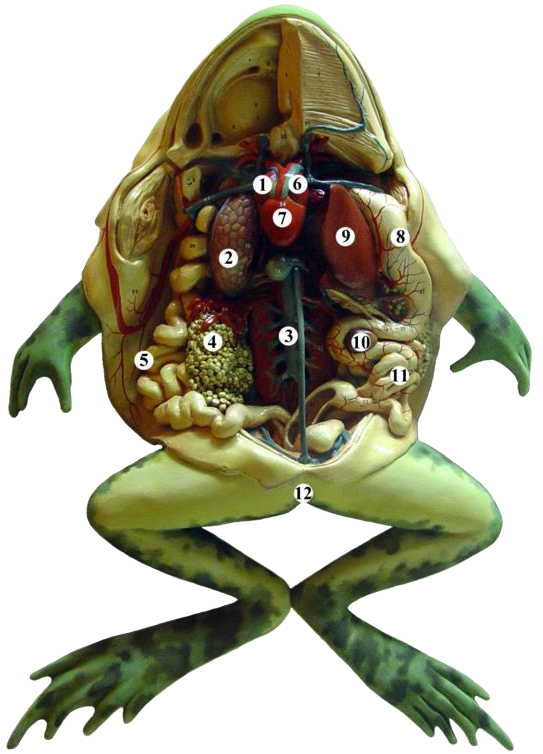
A shell in the container

- To get a shell running inside a container, simply launch:

```
docker run -i -t debian /bin/bash
```

- The flags **`-i`** and **`-t`** tell Docker to create an interactive session with a text terminal attached.
- The command **`/bin/bash`** starts the bash.
- Try it with any command you like!
- When you **`exit`** the shell, the container will stop – *containers only run as long as their main process*.

Basic Commands



- Let's launch a new container and play with it.
- We can give it a new hostname with the flag **`-h`**:

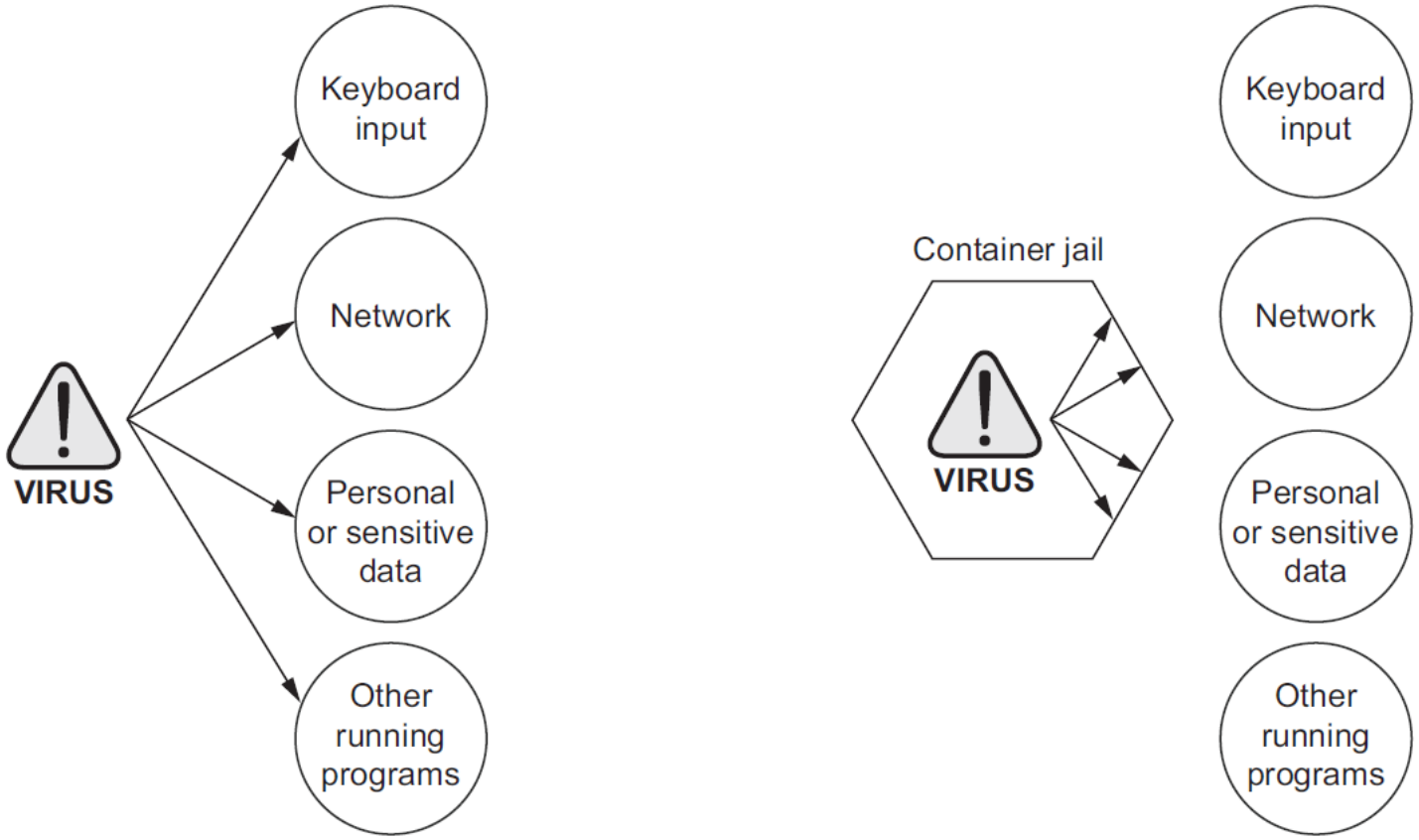
```
docker run –h CONTAINER –i –t debian /bin/bash
```

- Then we break it (don't try this on your host):

```
mv /bin /basket
```

- The container is now pretty useless (why?). Open a new terminal.

Breaking Containers



ps and inspect

- Launch `docker ps [-a]` and you will get some details on all running containers, along with a readable name to identify the container.

```
ase@ASE-VM:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
1071d7eab8a8        debian              "/bin/bash"         45 seconds ago      Up 44 seconds                           ecstatic_wright
```

- `docker inspect` gives some more information about the container which can then be parsed with **`grep`** or **`–format`** followed by Go templates*. E.g.,

```
docker inspect ecstatic_wright | grep IPAddress
```

```
docker inspect --format {{.NetworkSettings.IPAddress}} ecstatic_wright
```

*https://golang.org/pkg/html/template/

diff

- Let’s move onto another command:

```
docker diff ecstatic_wright
```

- Docker uses a **union file system** (UFS) which allows multiple file system to be mounted as a hierarchy and to appear as a single file system.
- The file system of the base image (i.e., debian) is a **read-only layer**, and any changes are made to a read-write layer mounted on top of it.
- The `diff` command lists all files that have been changed in the running container with respect to the base image.

logs

- Let’s move onto another command:

```
docker logs ecstatic_wright
```

- The `logs` command lists everything that happened within a container since its creation.
- We can exit from the shell, stopping also the container.

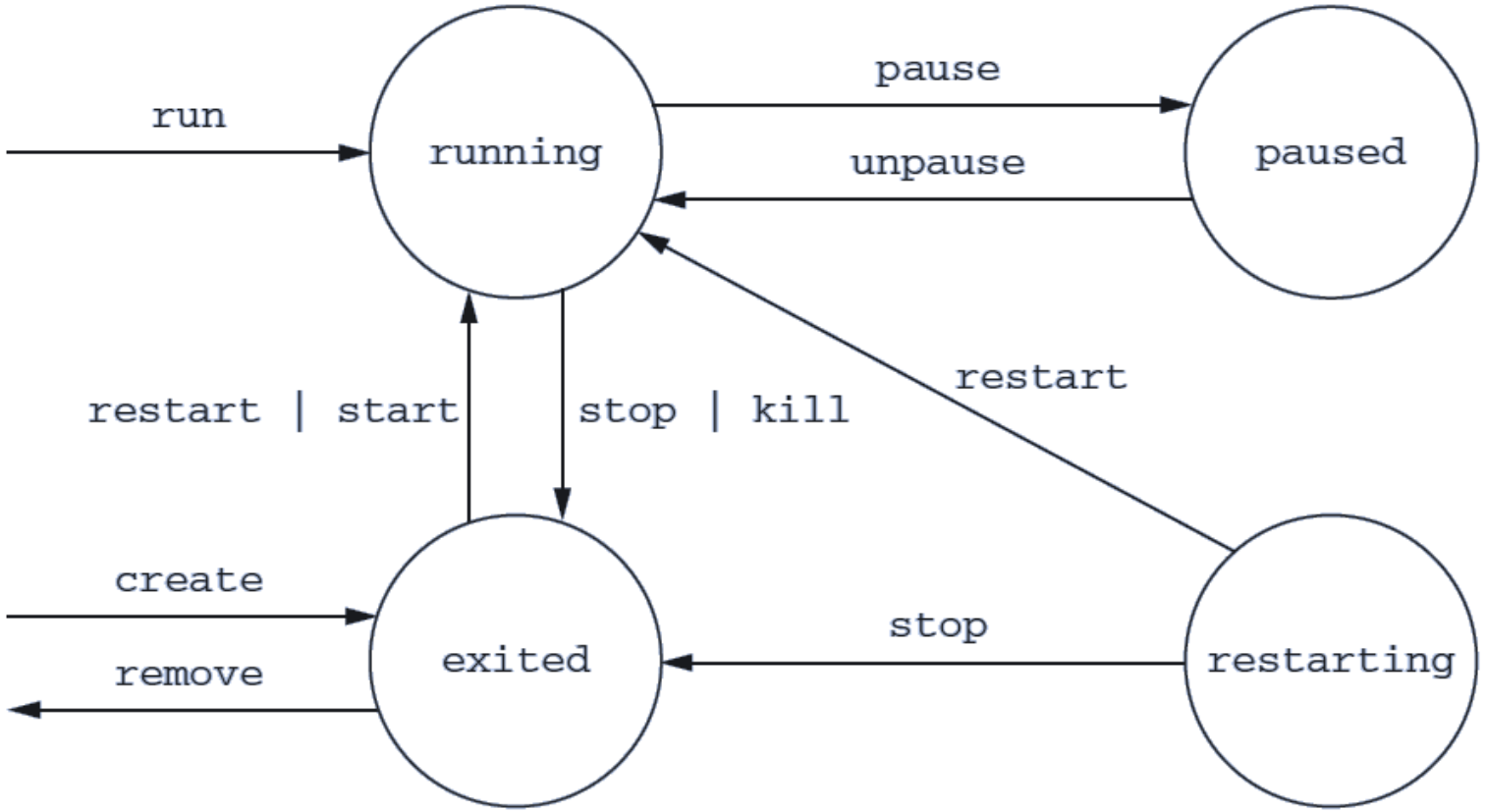
```
docker ps -a
```

```
ase@ASE-VM:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
1071d7eab8a8        debian              "/bin/bash"         About an hour ago   Up About an hour                        ecstatic_wright
```

- To remove an exited container, use:

```
docker rm ecstatic_wright
```

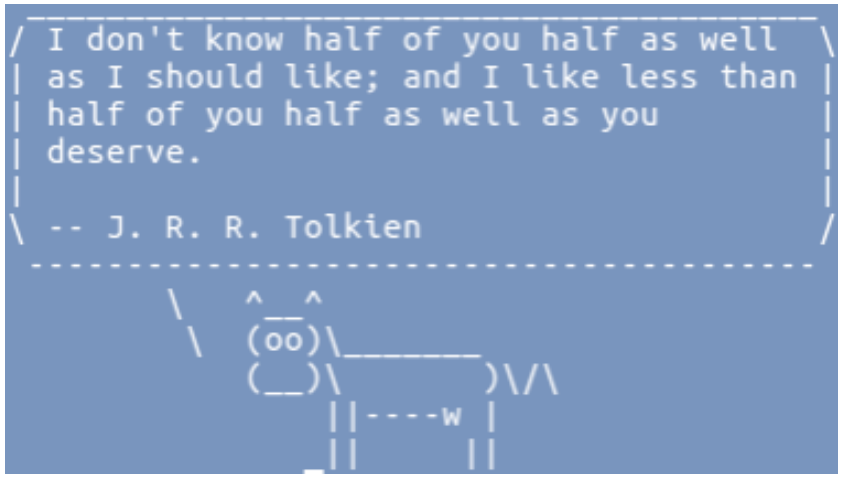
States for a container



Cowsay

- Let's create a *useful* container that we may want to keep as an image.
- We are going to *dockerise* a **Cowsay** application.

```
docker run –it --name cowsay --hostname cowsay debian bash
```




```
apt-get update
```

```
apt-get install –y cowsay fortune
```

```
/usr/games/fortune | /usr/games/cowsay
```

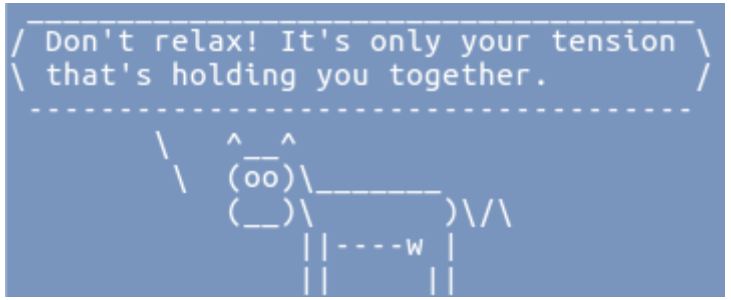
commit

- Let’s keep this container by turning it into an image.

```
docker commit cowsay test/cowsayimage
```

- The returned value is the unique id for our image.
- We can run the image by simply running:

```
docker run test/cowsayimage /usr/games/cowsay “Moo”
```

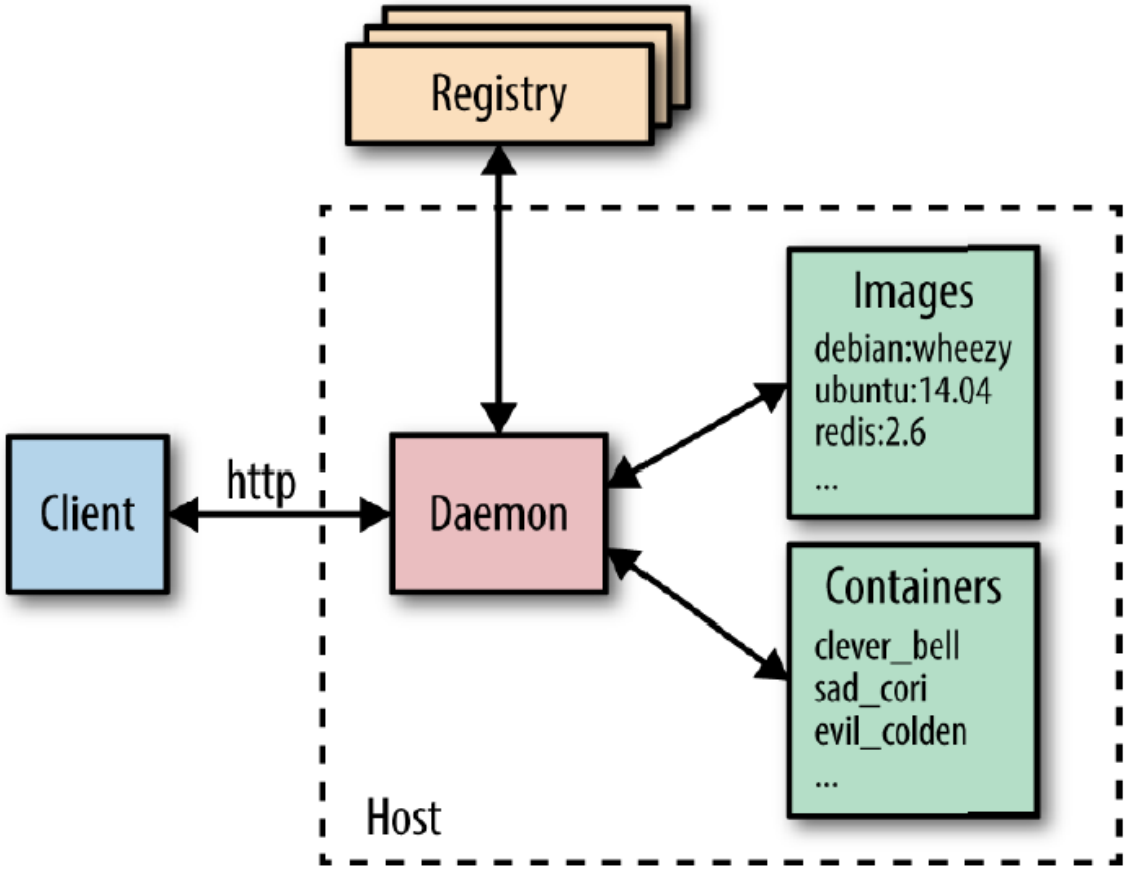


Registries, Repositories, Images, Tags

- **Registry** → A service that hosts and distributes images (e.g., DockerHub).
- **Repository** → A collection of related images (usually providing different versions of the same application or service).
- **Tag** → An alphanumeric identifier attached to images within a repository (e.g., `default`, `stable`, `14.04`).
- To pull an image from the Docker Hub, it suffices to run:

```
docker pull <repository>:<tag>
```

Docker Architecture



Create an Image

- There are 4 ways we can create a Docker image:
  1. Docker **commands** (aka “a manina”)
  2. **`Dockerfile`** (build from a known base image, and specify build with a limited set of simple commands).
  3. **`Dockerfile`** & **configuration management** tool
  4. **Scratch image** and import a set of files (from an empty image, import a TAR file with the required files.)

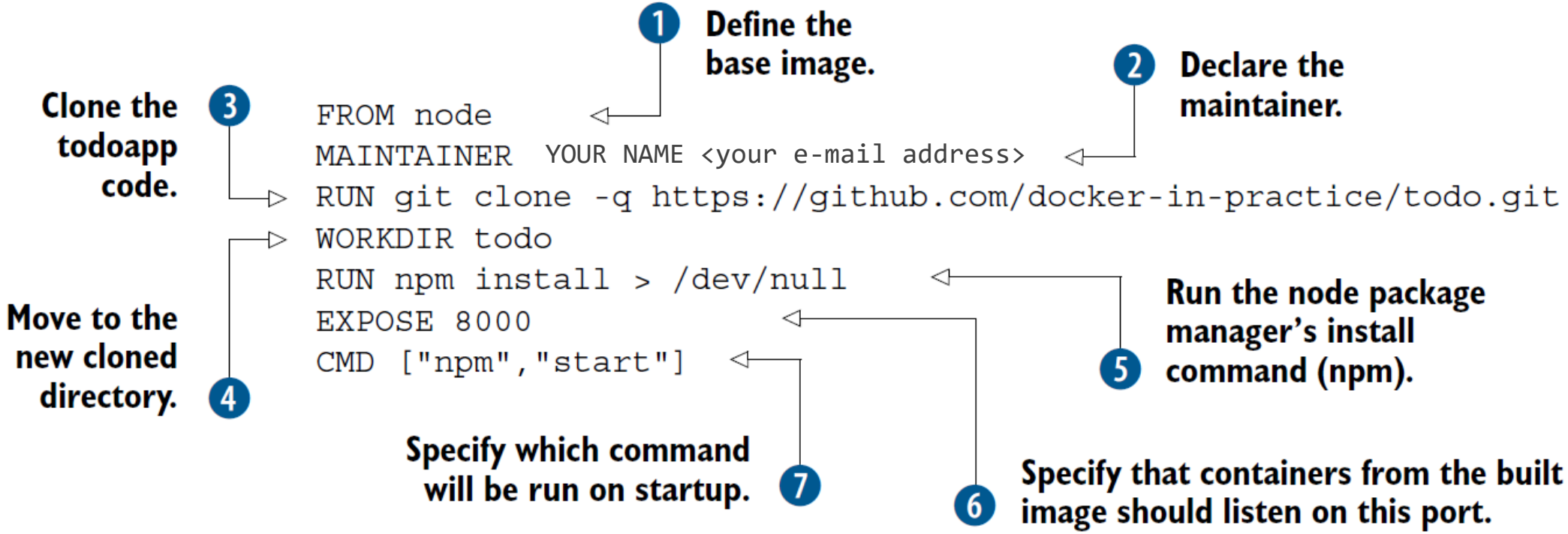
- Let’s start with a **real app**!

https://github.com/docker-in-practice/todo.git

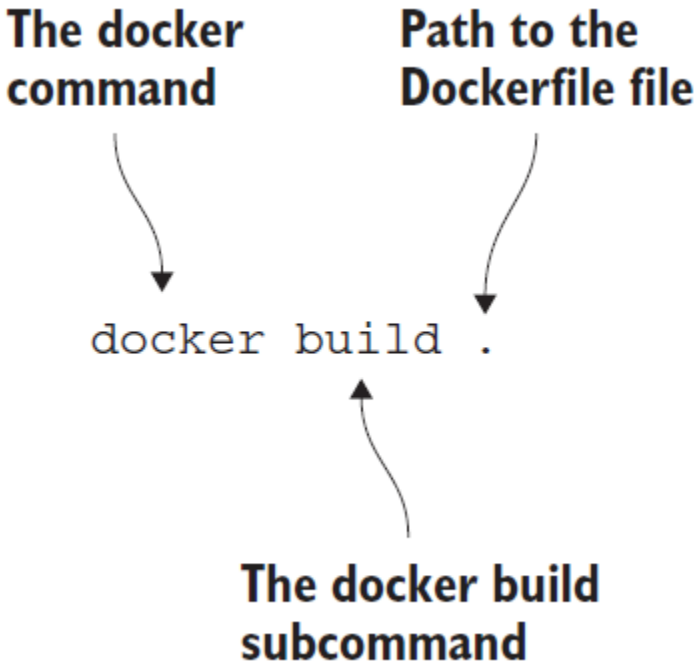
A to-do app is one that helps you keep track of things you want to get done. The app we’ll build will store and display short strings of information that can be marked as done, presented in a simple web interface.

Your first `Dockerfile`

- A `Dockerfile` is a text-file that contains a set of steps that can be used to create a Docker image.

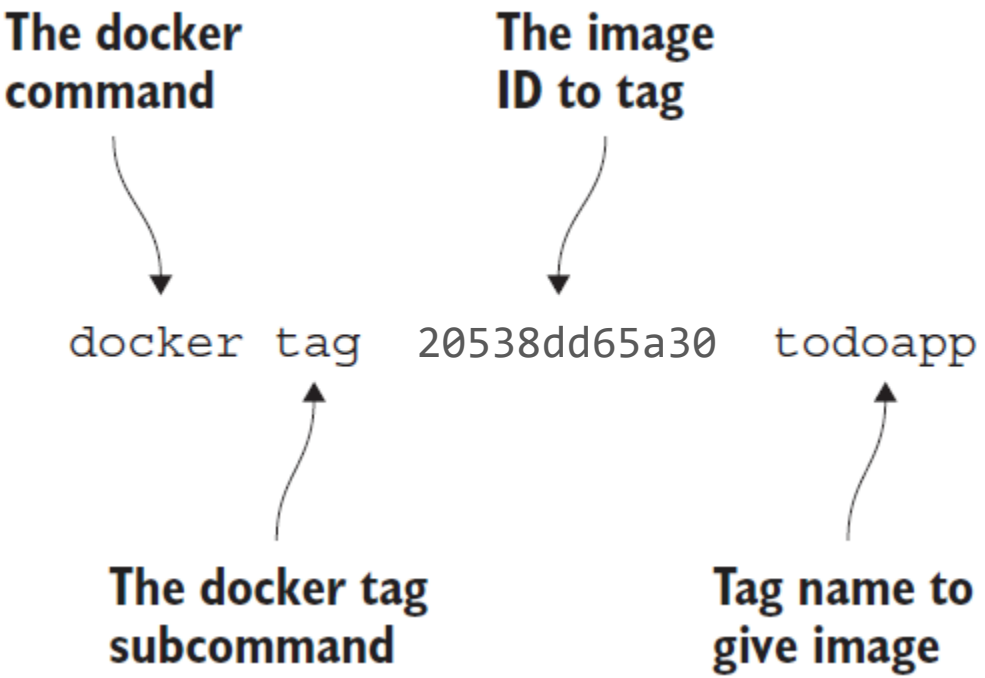


As easy as `docker build .`




```
Sending build context to Docker daemon  2.048kB
Step 1/6 : FROM node
 ---> cf20b9ab2cbc
Step 2/6 : RUN git clone -q https://github.com/docker-in-practice/todo.git
 ---> Using cache
 ---> 468430cabedc
Step 3/6 : WORKDIR todo
 ---> Using cache
 ---> b24ef8ca1e56
Step 4/6 : RUN npm install > /dev/null
 ---> Using cache
 ---> 3148d4ac8832
Step 5/6 : EXPOSE 8080
 ---> Using cache
 ---> 5981c7e43840
Step 6/6 : CMD ["npm", "start"]
 ---> Using cache
 ---> 20538dd65a30
Successfully built 20538dd65a30
```

Tag an image


The docker command
The image ID to tag
docker tag 20538dd65a30 todoapp
The docker tag subcommand
Tag name to give image

Run it!

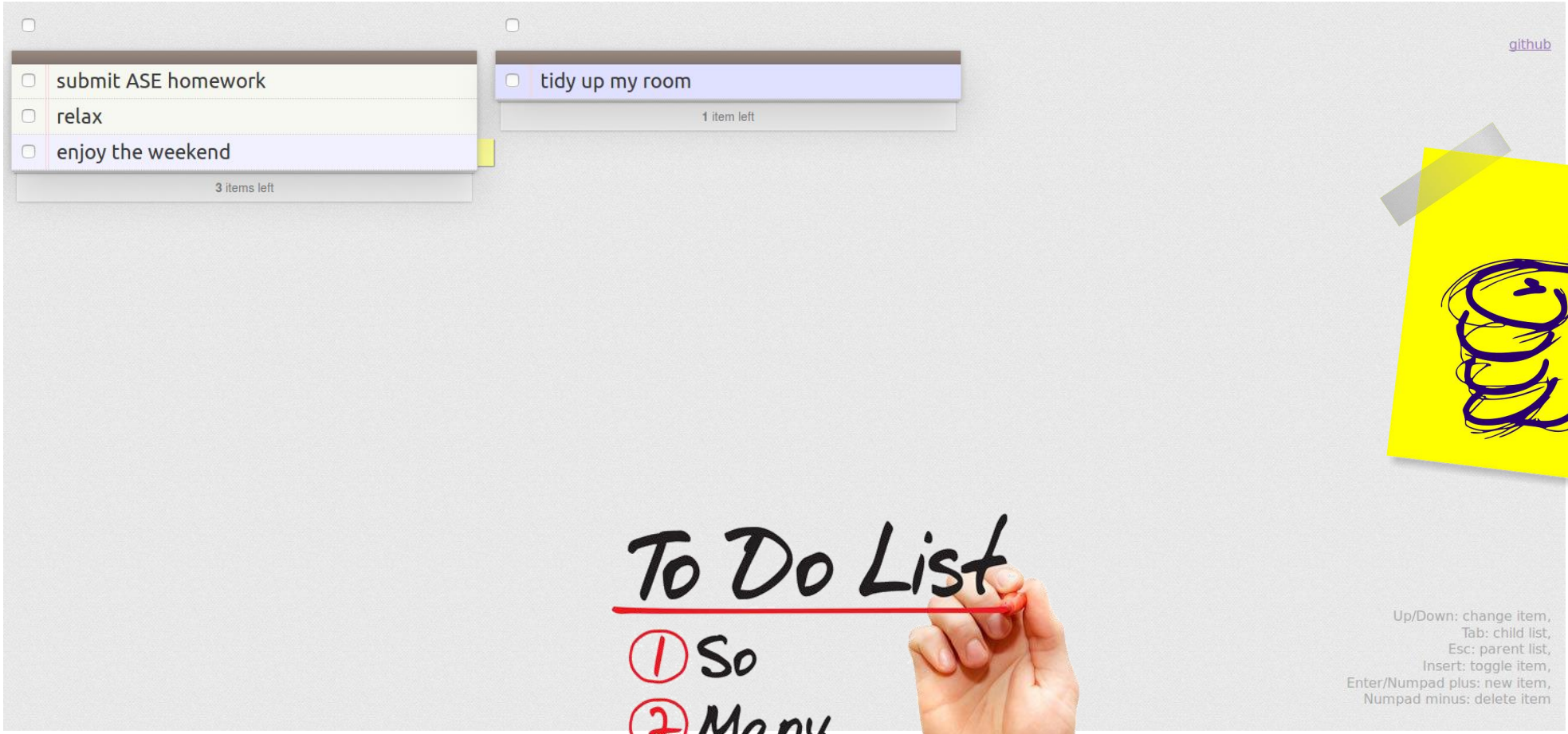
```
docker run -p 8000:8000 --name example1 todoapp
npm install
npm info it worked if it ends with ok
npm info using npm@2.14.4
npm info using node@v4.1.1
npm info prestart todomvc-swarm@0.0.1

> todomvc-swarm@0.0.1 prestart /todo
> make all
```

The output of the container's starting process is sent to the terminal.

The docker run subcommand starts the container, -p maps the container's port 8000 to the port 8000 on the host machine, --name gives the container a unique name, and the last argument is the image.

(1)

To Do App is Up



Key Docker Commands

Command	Purpose
`docker build`	Build a Docker image.
`docker run`	Run a Docker image as a container.
`docker commit`	Commit a Docker container as an image.
`docker tag`	Tag a Docker image.

pull

- If you search the Docker Hub* for a popular application/service (e.g., Java, PostgreSQL) you will find 100's of results.
- You can pull any image as seen before. Let's pull the Redis (key-value store) official image:

```
docker pull redis
```

- And run it in a background (`-d`) container:

```
docker run --name myredis –d redis
```

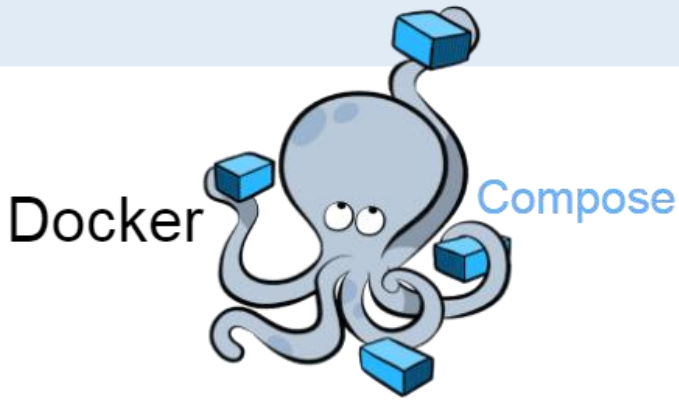
*https://hub.docker.com

Docker Compose

- Compose is a tool for **defining and running multi-container** Docker **applications**.
- Then, with a **single command**, you create and start all the services from your configuration.
- Install by running:

```
curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname -s)-$(uname -m)"\
 -o /usr/local/bin/docker-compose
```

```
chmod +x /usr/local/bin/docker-compose
```



Overview

Using Compose is basically a three-step process:

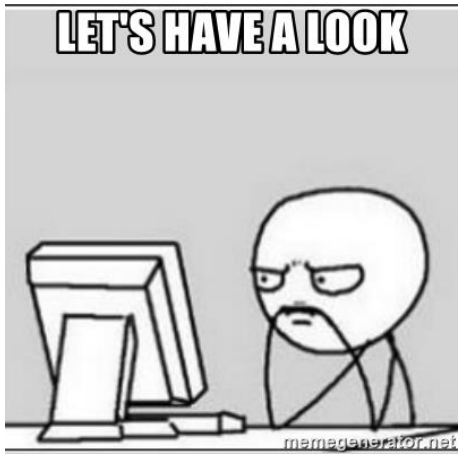
1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Run `docker-compose up` and Compose starts and runs your entire app.

An example

- Create a directory **`composetest`**.

```
mkdir composetest
cd composetest
```

- Download the **`app.py`** file from Moodle.
- The app uses Flask and maintains a hit counter in Redis (have a look!)



Example Dockerfile

- Build an image starting with the Python 3.4 image.
- Add the current directory `.` into the path `/code` in the image.
- Set the working directory to `/code`.
- Install the Python dependencies.
- Set the default command for the container to python `app.py`.

```
FROM python:3.4-alpine
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

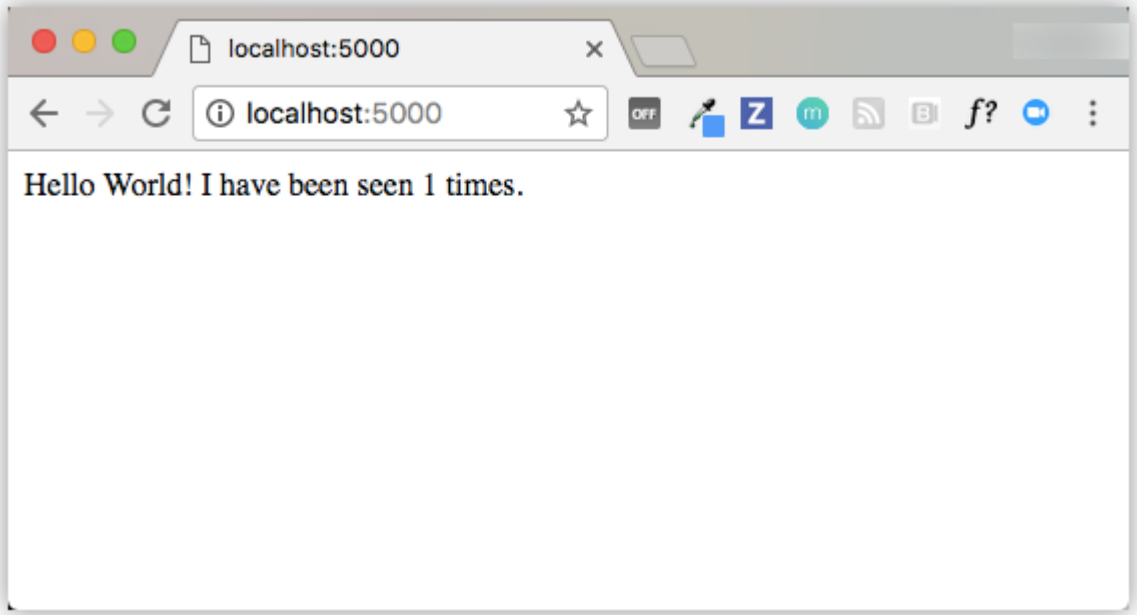
docker-compose.yml

- Define two services, **web** and **redis**.
- **web** uses an image that's built from the Dockerfile in the current directory.
- **web** forwards the exposed port 5000 on the container to port 5000 on the host machine.
- The **redis** service uses a public Redis image from the Docker Hub registry.

```
version: '3'
services:
  web:
    build: .
    ports:
     - "5000:5000"
  redis:
    image: "redis:alpine"
```

Start it!

```
docker-compose build
docker-compose up
```



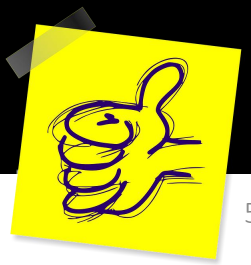
Try to `ps` containers…

docker ps -a

```
COMMAND                  CREATED             STATUS                      PORTS                    NAMES
"python app.py"          27 minutes ago      Up 27 minutes               0.0.0.0:5000->5000/tcp   compose[illegible]_web_1_4a7a49f571b7
"python app.py"          28 minutes ago      Exited (0) 27 minutes ago                            sharp[illegible]
"/bin/sh -c 'sudo pi…"   13 hours ago        Exited (127) 13 hours ago                            quirky[illegible]
"python app.py"          13 hours ago        Exited (1) 13 hours ago                              friend[illegible]
"python app.py"          13 hours ago        Exited (1) 13 hours ago                              loving[illegible]
"python app.py"          13 hours ago        Exited (1) 13 hours ago                              suspi[illegible]
"/bin/sh -c 'apt-get…"   13 hours ago        Exited (127) 13 hours ago                            graci[illegible]
"docker-entrypoint.s…"   13 hours ago        Up 27 minutes               6379/tcp                 compo[illegible]17d1d193
```

docker-compose down

```
stefano@stefano-pc:~/Desktop/DockerLab/composetest$ docker-compose down
Stopping composetest_web_1_4a7a49f571b7   ... done
Stopping composetest_redis_1_e18217d1d193 ... done
Removing composetest_web_1_4a7a49f571b7   ... done
Removing composetest_redis_1_e18217d1d193 ... done
Removing network composetest_default
```



Add a bind mount?

- Edit `docker-compose.yml` in your project directory to add a bind mount for the web service.
- The new `volumes` key mounts the project directory (current directory) on the host to `/code` inside the container, **allowing you to modify the code on the fly, without having to rebuild the image**.

```
version: '3'
services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
  redis:
    image: "redis:alpine"
```

https://docs.docker.com/storage/bind-mounts/

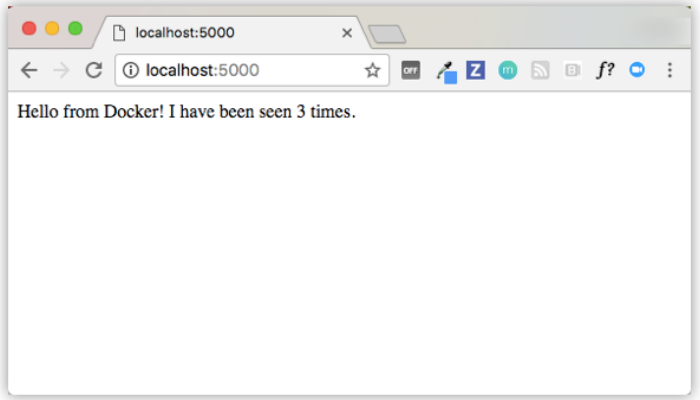
Try it!

```
docker-compose build
docker-compose up
```

- You can now make changes to its code and see the changes instantly, without having to rebuild the image. Change **`app.py`**:

```
return 'Hello from Docker! I have been seen {} times.\n'.format(count)
```

- Refresh your browser.



Other commands

- Detached mode (i.e., background) `docker-compose up -d`
- Run one-off commands for a service `docker-compose run web env`
- When starting with `-d`, stop with `docker-compose stop`
- Remove containers entirely with `docker-compose down --volumes`

What’s next?

- Try the Cats&Dogs tutorial from the Moodle.
- Try the Wordpress tutorial from the Moodle.
- Try to Dockerise your services and compose them (Docker docs are awesome if you need some help!)