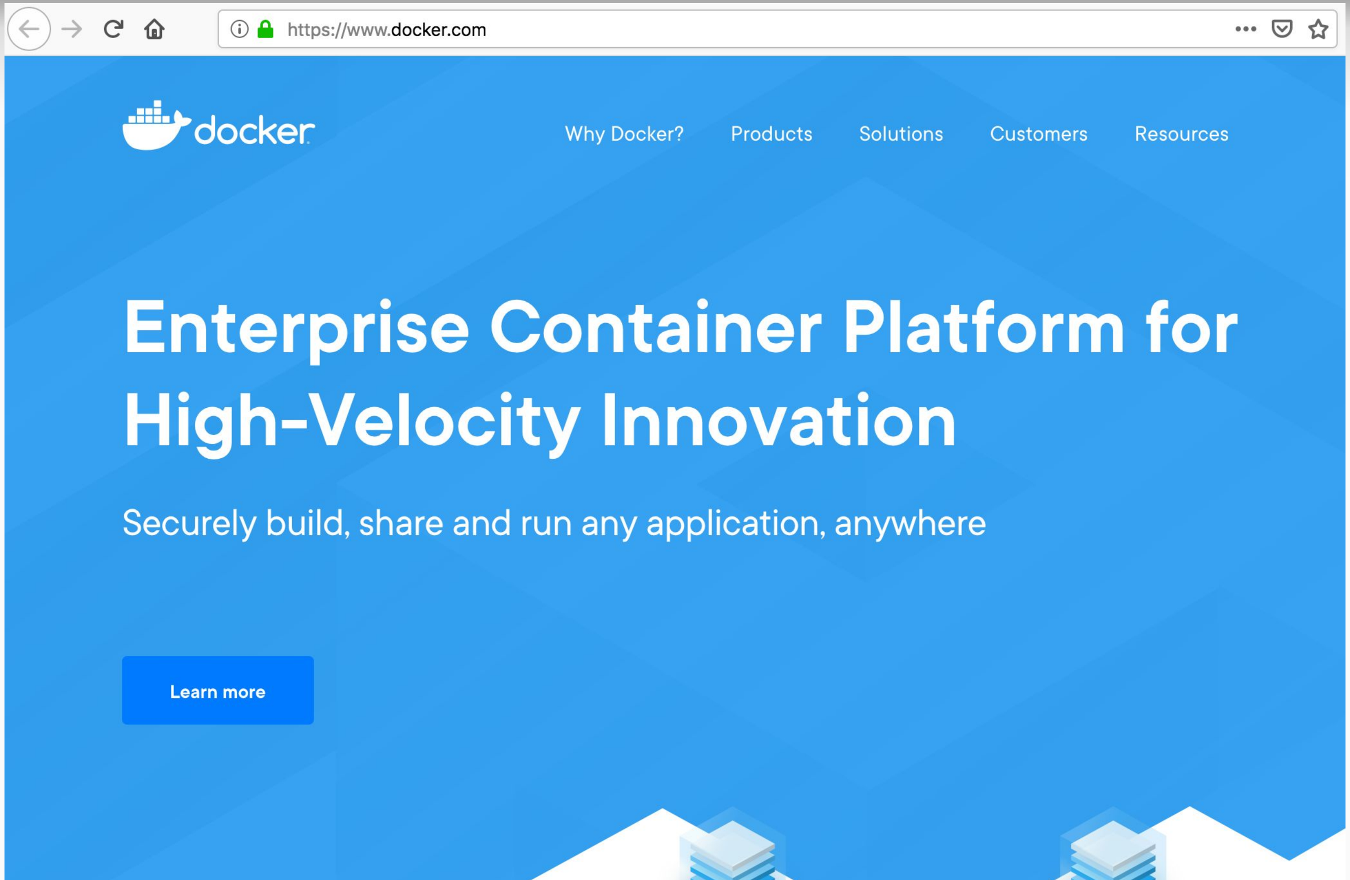
Docker for Research Computing

Dustin Lang / Perimeter Institute / 2019-05-10

Is Docker just a fad?



https://www.docker.com

Enterprise Container Platform for High-Velocity Innovation

Securely build, share and run any application, anywhere

Docker / Containerization basics

- Docker is a company that provides tools (in “Community” and “Enterprise” flavours) for *Containerization*
- Containerization is a set of Linux kernel features for controlling the environment a program runs in
- Lets you *package up* an environment (software libraries + scripts + data) so that it can run on different machines
- Sits on top of the Linux kernel, but you (have to) build the whole filesystem
- Higher-level than a *Virtual Machine* approach
- Lower-level than a *Virtualenv* or *Conda* approach

But what IS a container?

- *Image*: a tar file that becomes the root filesystem, plus some instructions

- *Container*: just a *process* running on your computer, but run in a special environment:
    - has its own root filesystem
    - (can't see the "host" computer's filesystem unless you let it)
    - can see the CPU, Linux kernel, and network
    - thinks it has *root* power, but only *inside* the container

How can Docker be useful for research?

- Set up software without as much excruciating pain
- Share your setup with collaborators
- Move your analysis from laptop to compute cluster
- Archive your setup for your own later use
- Publish your setup for reproducibility

When Docker is helpful

Helpful especially when:

- you're using all open-source libraries
- you have a deep dependency stack
- some of your dependencies are fussy to set up, or
- the libraries you use are available only as packages for some distribution (ubuntu, centos, debian, …)

Less helpful / more tricky:

- you use proprietary / licensed software (eg, mathematica, idl, intel compiler/libraries)
- you need graphics/interaction
- you depend on specific CPU features

I don't really know, but apparently there is good support for:

- GPUs / TensorFlow

Why is the software industry keen on Docker?

Containers:

- are *isolated* from each other. You can run many containers on one computer and they can't see each other (unless you say they can)
- *encapsulate* or *package up* all the dependencies of a project -- you can update a library inside one container without breaking other containers
- make it easier to run on different computers (local or cloud)
- simplify automated testing & deployment
- help with security -- a container can get hacked without compromising anything else

… and Docker caught on as maybe the easiest way to deploy containers

Demo (1)

```
> docker run -it ubuntu:18.04
root@deaa60bc7eeb:/#
```

← Create and run a new container starting from the image called "ubuntu:18.04". This doesn't exist on my local computer, so will be retrieved from "Docker Hub", at https://hub.docker.com/_/ubuntu

← The container gets a new "hash" name ("deaa60bc7eeb"). Inside the container, we are "root".

```
root@deaa60bc7eeb:/home# ls /home
root@deaa60bc7eeb:/home#
```

← Inside the container, we can't see the host's filesystem: the /home/ directory is empty. If you run "top", you can't see any processes running on the host either.

```
root@deaa60bc7eeb:/home# apt update
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
...
root@deaa60bc7eeb:/home# apt install python3 python3-pip
…
root@deaa60bc7eeb:/# exit
exit
> docker commit d7b035254dc8 mysetup
sha256:ec77d190b5270c8bffe3ec46bad8099f9a35e649c5d9192195c9d4fc368d6474
```

← You can now start installing Ubuntu packages, setting up the environment your code needs

← When you're done, "exit" the container and you can save your state to a new *image* with the "docker commit" command

Demo (2)

```
> docker run -it mysetup
root@d699e85d8963:/#
```

← You can create a new container from the image you just "commit"ed. It will get a new random hash name.

```
root@d699e85d8963:/# touch /home/my-file.txt
root@d699e85d8963:/# exit
exit

> docker run -it mysetup
root@180d274dd38f:/# ls /home
root@180d274dd38f:/# exit
exit
```

← Containers (unlike *images*) are ephemeral -- here, we create a file and exit the container. If we then run a new container from the same image, that new file does not exist.

It is possible, however, to re-"start" that old container and "attach" a shell to it -- and then we can see that the file exists inside that container. If we want to save our state, we need to "commit" it with a new name.

```
> docker start d699e85d8963
d699e85d8963
> docker attach d699e85d8963
root@d699e85d8963:/# ls /home
my-file.txt
```

Using a Dockerfile

- Instead of using *docker commit* to save the state of a container, standard practice is to *script it up* using a *Dockerfile*. See here for the syntax: https://docs.docker.com/engine/reference/builder/

```
FROM ubuntu:18.04
RUN apt update
RUN apt install -y python3 python3-pip
RUN pip3 install emcee
RUN mkdir /myproject
COPY amazing.py /myproject
```

- Create a new directory for your project, save this text to a file named “Dockerfile”, and use “docker build .” to run your Dockerfile recipe.

```
> docker build . -t demo
….
> docker run -it demo
```

Code development workflow

While developing your software, a reasonable approach is to add all your dependencies to the container, but *mount* your project directory in the container. You then use your usual editor and development tools, but when you’re running your code, you run it within the container.

```
> mkdir ~/myproject
> docker run --mount type=bind,src=$HOME/myproject,dst=/myproject -it demo
root@e7ed7f95998e:/# ls /myproject/
root@e7ed7f95998e:/#

# Now edit ~/myproject/script.py on your computer as usual

root@e7ed7f95998e:/# python3 /myproject/script.py
Hello world!
```

When you’re ready to ship...

A good practice is to develop your code in a Git(hub) repository.

When you are ready to share your code with a collaborator or run it on a cluster, modify your Dockerfile (which you should also keep in the repository) to check out your code from Github:

```
FROM ubuntu:18.04
RUN apt -y update && apt install -y --no-install-recommends \
        python3 python3-pip python3-setuptools python3-wheel git
RUN pip3 install numpy==1.15 emcee
MKDIR /myproject
WORKDIR /myproject
RUN git clone https://github.com/dstndstn/docker-demo.git .
COPY my-data-file.txt /myproject
CMD ["python3", "amazing.py", "my-data-file.txt"]
```

When you’re ready to ship… (2)

Now build your container, tagging it with your Docker Hub username, a slash, and the new Docker Hub repository name, then “push” it to Docker Hub.

```
> docker build -t dstndstn/docker-demo .
…
Successfully tagged dstndstn/docker-demo:latest

> docker push dstndstn/docker-demo
The push refers to repository [docker.io/dstndstn/docker-demo]
38dc68462c75: Pushing [====>                                              ]  6.657MB/73.22MB
…
```

When you’re ready to ship… (3)

Aside: if you update your code in Github and want to update the container, you can force “docker build” to re-clone your repository by adding a “&& echo 1” to the end of the RUN line, and increment the number to force a re-clone:

```
…
RUN git clone https://github.com/dstndstn/docker-demo.git . && echo 2
…
```

Moving from laptop to supercomputer

- I have my Docker container set up, but I need more compute power!
- In theory: *just docker run it on the supercomputer!*
- In practice: *lol nope,* security issues prevent us from allowing that
- On Symmetry: *singularity* can run Docker containers

```
$ module load singularity
$ singularity build myproject.sif docker://dstndstn/docker-demo:latest
$ singularity run --hostname container myproject.sif bash
dlang@container:~$ cd /myproject
dlang@container:/myproject$ python3 amazing.py my-data-file.txt
Hello World!
Hello Esteemed Colleagues!
Hello Everyone!
dlang@container:~$ exit
```

- (and inside the container can see your home directory, /gpfs, etc)

Moving from laptop to supercomputer (2)

- Slurm script, if you have a single- or multi-process script:

```
#! /bin/bash

#SBATCH -p debugq
#SBATCH -t 5:00
#SBATCH -N 1

echo "Hello from $(hostname)"
module load singularity
singularity run myproject.sif python3 /myproject/amazing.py --cores 10 run1 &
singularity run myproject.sif python3 /myproject/amazing.py --cores 10 run2 &
singularity run myproject.sif python3 /myproject/amazing.py --cores 10 run3 &
wait
echo "All done!"
```

Moving from laptop to supercomputer (3)

- Slurm script using MPI: in this Github repo I have a Dockerfile and example script that builds MPI in the container. https://github.com/dstndstn/docker-demo.git

```
#! /bin/bash

#SBATCH -p debugq
#SBATCH -t 5:00
#SBATCH -N 2

echo "Hello from $(hostname)"
module load mpich
# Turns out, we can’t run singularity directly
# because we need to load a module...
mpirun -n 80 ./run-singularity.sh
echo “All done”
```

run-singularity.sh:

```
#! /bin/bash
module load singularity
singularity run myproject.sif \
        python3 /demo/sampler.py --mpi
```

Docker and reproducibility

Three+ ways using Docker helps reproducibility:

- Other researchers can immediately re-run your code using Docker
- In ten years when Docker is long gone (maybe), the Dockerfile still has simple human-readable instructions for setting up the environment
- It's possible to *export* a Docker container to a set of tarballs (so if you really needed to, you could figure out exactly which files existed in your container's filesystem)
- The *best practices* push you toward better reproducibility practices (use tagged code, script up your analyses, think about reproducibility from the beginning)