DOCKER FOR DATA SCIENCE

Data science academy - HelloFresh

Max Halford

June 2017

WHY USE DOCKER?

SETUP ISSUES

A lot of the time...

- Data science teams have a single server
- Project X needs Python 3 and Project Y needs Python 2
- Local OS is Windows or Mac, production server OS is Ubuntu
- Reproducing local setup in production is a pain
- Some ML software requires a complicated setup which can break your computer

VIRTUAL ENVIRONMENTS SOLVE SOME PROBLEMS

- Each project gets it's own dedicated interpreter
- Dependencies are kept separate
- `virtualenv` for Python, `packrat` for R
- Only applies at a programming language level

CONTAINERIZATION AS A SUPER VIRTUAL ENVIROMENT

- Think of Docker as an older cousin of `virtualenv` and `packrat`
- It's like having a computer inside your computer
- Everything can be kept separate: OS, databases, languages, cronjobs, ...
- Docker can be used for a lot of use cases but it has quite a steep learning curve

DOCKER CONCEPTS

DOCKER CONCEPTS

- **Host**: computer on which Docker is installed
- **Image**: a template/blueprint for creating containers in an idempotent way
- **Container**: virtual computer created from an image and located on a host

Example 1: Dockerizing an R interpreter

DOCKERIZING AN R INTERPRETER (1)

1. Search for images containing the term “R”: `docker search R`
2. Create a container called **`arr`** from the **`r-base`** image and attach the input to the terminal: `docker run -it –name arr r-base`
3. Play around with the R interpreter to make sure it works and then run `quit()` to go back to your host’s terminal

DOCKERIZING AN R INTERPRETER (2)

1. `docker ps` displays the running containers, for the while it should show nothing
2. `docker ps –all` displays all the existing containers, even the non-running ones
3. `docker images` displays the images contained on the host, here there should be a single one
4. Running `docker run -t -i –name arr r-base` should fail because a contained named `arr` already exists

DOCKERIZING AN R INTERPRETER (3)

1. `docker start arr` will run the `arr` container but won't attach your terminal to it
2. You can see it running with `docker ps`
3. You can attach to the R interpreter with `docker attach arr`
4. After exiting with `quit()`, run `docker stop arr` to stop the container
5. Run `docker rm arr` to delete the `arr` container
6. Run `docker rmi r-base` to delete the `r-base` image

Example 2: Dockerizing a Python App

DEPLOYING A PYTHON APP (1)

1. Say you have a Python that runs on your computer when you launch a script (eg. `python run.py`)
2. We want to do more than open an interpreter, we want to send code on the host to the container (or pull it from GitHub while in the container)
3. **Dockerfiles** allow to list a succession of commands describing how to build a container in an idempotent fashion

DEPLOYING A PYTHON APP (2)

1. Clone the Cerebro repository with `git clone https://github.com/hellofresh/data-science-cerebro`
2. Go into the folder with `cd data-science-cerebro`
3. The folder contains a `Dockerfile`, run `docker build -t cerebro .` to build a container with the name `cerebro` (this takes time)
4. Run `docker run cerebro python cli.py` to execute `python cli.py` as if you were in the container

DEPLOYING A PYTHON APP (3), DOCKERFILE (1)

```
FROM jfloff/alpine-python

MAINTAINER Max Halford "mh@hellofresh.com"

VOLUME /data

# Install git, ssh and mariadb-dev
RUN apk add --update git openssh mariadb-dev

# Numpy requirement
RUN ln -s /usr/include/locale.h /usr/include/xlocale.h
```

Deploying a Python app (3), Dockerfile (2)

```
# Python packages
RUN pip install pandas
RUN pip install impyla
RUN pip install click
RUN pip install tinydb
RUN pip install tinydb-serialization

# Copy the code over
ADD . /cerebro
WORKDIR /cerebro

# Set the configuration file
RUN ln -s setup/config_docker.py config.py
```

DEPLOYING A PYTHON APP (4)

- In practice you want to be able to update the Docker container with new code
- If you edit code on the host then running `docker build -t cerebro .` again will only execute `ADD . /cerebro` and the commands that are afterwards in the Dockerfile
- Data (databases, CSV outputs) can but should not be stored in the same container as the application because it would lose the idempotency property
- It's possible to store data in separate containers that can be shared between other containers but that's for another presentation :)

EXAMPLE 3: DOCKERIZING JUPYTERHUB

DOCKERIZING JUPYTERHUB (1)

1. `docker run -d -p 2424:8000 –name jupyterhub jupyterhub/jupyterhub` will run the `jupyterhub/jupyterhub` image in detached mode (basically a daemon) and link the host's port 2424 to the container's port 8000
2. You can now access JupyterHub if you navigate to `http://localhost:4242` in your browser
3. This image isn't perfect, we actually have to install a Python library so that individual notebooks can be spun up; run `sudo docker exec -it jupyterhub bash` to access the container's console
4. Run `pip install notebook` to install the notebook library on the container

DOCKERIZING JUPYTERHUB (1)

1. JupyterHub requires adding users
2. Run `sudo docker exec -it jupyterhub bash` to access the container's console
3. Add user **homer** with `useradd homer`
4. Set Homer's password with `passwd homer`
5. Run `mkdir /home/homer` to create a **/home** folder to store the notebooks produced with JupyterHub
6. Give full access to the **/home/homer** folder with `chmod 777 /home/homer`

USEFUL LINKS

1. R related images: **https://github.com/rocker-org/rocker**
2. Tensorflow image for CPUs and GPUs: **https://hub.docker.com/r/tensorflow/tensorflow/**
3. A good overview of Docker: **https://rominirani.com/docker-tutorial-series-a7e6ff90a023**
4. A Dockerfile for running Python on Alpine Linux: **https://hub.docker.com/r/jfloff/alpine-python** (I can recommend it)

Thanks for listening!