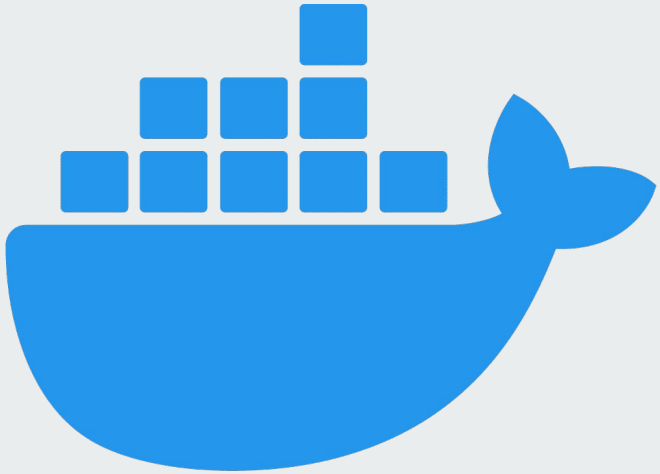
Basil Presentation: Docker

Randy Harnarinesingh



- *What is Docker?*
- *Benefits and Use cases*
- *Examples of:*
  - *Docker CLI*
  - *docker-compose*
  - *Dockerfile*
- *Benefits of microservices architecture*
- *Some useful Docker commands*

What is Docker?

- **Docker** is an OS-emulation product that executes software (containers) based on some predefined set of instructions (images).
- Images are essentially environment snapshots that contain pre-installed software and preset configurations.
- Some of the more popular images includes:
  - Alpine
  - Busybox
  - Nginx
  - Ubuntu
  - Python
  - PostGreSQL
  - Redis

Some advantages

- Docker stands apart from **VMware** in that VMware emulates **system hardware** whereas Docker emulates the **Operating System**.
- For this reason, Docker has fewer overheads than VMware and is a lightweight alternative.
- Other benefits include:
  - Runs in an isolated environment
  - Modular
    - Can build new images on top of other images
  - Easy to use
    - Once you get past the initial learning curve
  - Repeatable
    - Code runs the same way regardless of which host machine is used

Hobbyist case for Docker

- Experiment with tech without having to install dedicated software suite on PC
  - Particular useful if you want to test out a web server config on local or a database
- Create test environment for model training or other software execution and share with colleagues
- Wide selection of docker images on Docker Hub means you can test code against multiple OS/python environments quite quickly

Live demo time!

Example 1: Wordpress using Docker CLI

- ➔ Docker CLI is one of the ways you can launch Docker containers
- ➔ With docker installed, run the following command on CLI:

```
docker run --name my-wordpress -p 8080:80 -d wordpress
```

- ➔ Command explanation:
  - ◆ `--name`: Assigns container the name “some-wordpress”
  - ◆ `-p`: Maps container port 80 (HTTP) to host port 8080
  - ◆ `-d`: Runs in headless mode

Example 2: Wordpress using docker-compose

- ➔ docker-compose provides a way of running containers with preset configurations in a .yml file
- ➔ Simplifies execution compared to CLI since you don’t need to remember command options
- ➔ Provides an easier way to share containerised environment with others.

```yaml
version: '3.1'

services:

  wordpress:
    image: wordpress
    restart: "no"
    ports:
      - 80:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - ./wordpress:/var/www/html

  db:
    image: mysql:5.7
    restart: "no"
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    volumes:
      - ./db:/var/lib/mysql
```

Example 3: Creating a custom image

- Custom images can be built based on other existing images
- Context example:
  - Jupyter env for dev work
  - Port forward for Streamlit App

```
FROM jupyter/datascience-notebook
ENV GRANT_SUDO=yes
ENV JUPYTER_ENABLE_LAB=yes
USER root
RUN sudo apt-get update && sudo apt-get -y upgrade  # update system
COPY ./jupyter_notebook_config.py /home/jovyan/.jupyter/
COPY ./okta-aws /home/jovyan/.okta-aws
COPY ./requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
RUN rm requirements.txt
RUN rm -r /home/jovyan/work
```

Dockerfile example

Example 3: Deploying using docker-compose

- Main points:
  - Port forwarding for jupyterlab and streamlit app
  - Uses 2 persistent volumes to store both noteworks and apps

```
version: "2"
services:
  jupyter:
    build:
      context: .
      dockerfile: ./Dockerfile
    ports:
      - "9000:8888"
      - "9001:8501"
    volumes:
      - ./notebooks:/home/jovyan/notebooks:rw
      - ./streamlit:/home/jovyan/streamlit:rw
    environment:
      - GRANT_SUDO=yes
```

Example 4: phpmyadmin + mysql

- Main points:
  - Latest version of mysql db
  - phpmyadmin provides a web interface to interact with mysql databases
  - Very useful to practice SQL on local machine

```yaml
version : '3'

services:
  mysql:
    image: mysql:latest
    container_name: dev_mysql
    environment:
      MYSQL_USER: user
      MYSQL_PASSWORD: user
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: default_schema
    restart: 'no'
    volumes:
      - ./db:/var/lib/mysql

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: dev_pma
    links:
      - mysql
    environment:
      PMA_HOST: mysql
      PMA_PORT: 3306
      PMA_ARBITRARY: 1
    ports:
      - 8080:80
    restart: 'no'
```

Example 5: Hardware passthrough example

- Context example:
  - GPU passthrough to container environment
  - Allows for model training with GPU support

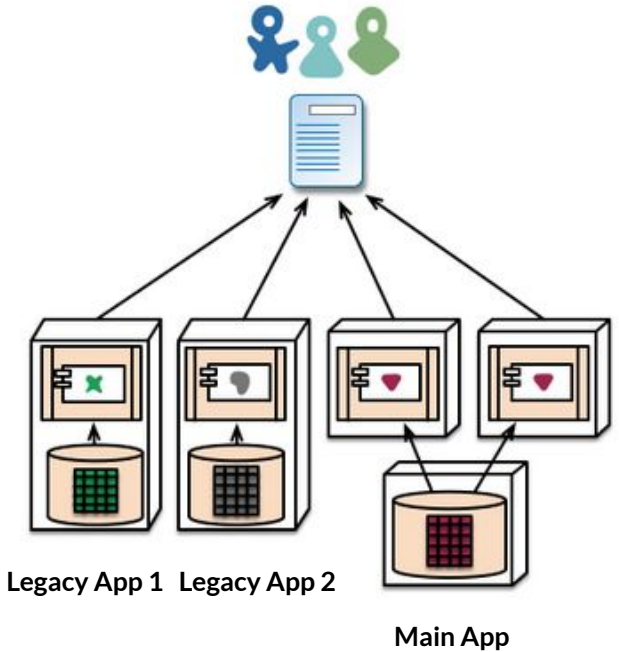
Example 5: Hardware passthrough example

- Main elements:
  - Jupyter env for dev work
  - Device config to allow for GPU passthrough
  - Single volume for persistent storage

```
services:
  test:
    build:
      context: .
      dockerfile: ./Dockerfile
    deploy:
      resources:
        reservations:
          devices:
          - capabilities: [gpu]
    ports:
      - 9000:8888
    devices:
      - /dev/nvidia0:/dev/nvidia0
      - /dev/nvidiactl:/dev/nvidiactl
      - /dev/nvidia-modeset:/dev/nvidia-modeset
      - /dev/nvidia-uvm:/dev/nvidia-uvm
      - /dev/nvidia-uvm-tools:/dev/nvidia-uvm-tools
    volumes:
      - ./data:/home/jovyan/work
```

Microservices example

- Example use case for docker:
  - Company uses legacy apps that require different OS/env setup
  - The different OS/env setups conflict so all can't be installed on a single machine
  - Can run each app in it's own self-contained environment built to requirements
  - Apps can communicate with each other across the Docker Network.



Some useful commands

- Test Docker installation: `docker run hello-world`
- Delete all Docker Images: `docker rmi -f $(docker images -a -q)`
- Get help on specific Docker commands: `docker command_name --help`
- List all Docker images on machine: `docker images`
- List all running Docker containers: `docker ps`
- Get information about specific image: `docker inspect image_name`
- Execute code inside a running container: `docker exec -it cont_id /bin/bash`
- Run docker-compose in headless mode: `docker-compose up -d`
- Stop running Docker container (gracefully): `docker stop image_id`
- Stop running Docker container (forcefully): `docker kill image_id`