Learning docker 01: Basics

**### Verify docker install**

$ docker run hello-world

**### Run an Ubuntu container in the foreground**

$ docker run –it ubuntu bash

-t : allocate a pseudo—tty

-i : keep STDIN open even if not attached

**### Run an nginx container in background mode and mapping a port**

$ docker run –d –p 8080:80 nginx

-d=False : detached mode

-p=[] : port mapping – publish container’s port or range or ports to host

format: ip:hostPort:containerPort | ip::containerPort

Learning docker 02: Building images

**### Tree**

| Dockerfile

| html/

| ---------index.html

### **Dockerfile**

FROM nginx

COPY html /usr/share/nginx/html

### **Command**

$ docker build –tag my\_http .

$ docker run -d -p 8080:80 --name my\_http\_container my\_http

Learning docker 03: Attaching volumes

Starting point – no volumes

**Dockerfile**

FROM php:7.2-cli

COPY ./src /usr/src/myapp <-- this will be burned into the image / be made immutable

WORKDIR /usr/src/myapp

CMD ["php", "./run.php"]

**run.php**

<?php

echo 'Running in docker';

$ docker build –t run-php .

$ winpty docker run -it --rm --name run-app run-php

\* changing the content of run.php won’t be reflected until the image is re-built

**Attaching volumes**

$ docker run -it --rm --name php-runner -v //c/Users/ronaldg/Desktop/learning/docker/tutorials/willitscale/tut3/src:/usr/src/myapp -w /usr/src/myapp php:7.2-cli php run.php

-w : working dir

$ docker run -it --rm --name php-runner –v //c:/users/ronaldg/Desktop/learning/docker/tutorials/willitscale/tut3/src:/usr/src/myapp -w /usr/src/myapp php:7.2-cli php run.php

\* doesn’t work, figuring out why

Learning docker 04: Compose

**Dockerfile**

FROM php:7.2-apache

**docker-compose.yml**

version: "3"

services:

www:

build: .

ports:

- "8040:80"

volumes:

- ./src:/var/www/html/

networks:

- default

**/src/index.php**

<?php

echo "We're in our composed containers";

Learning docker 05: Customising Docker Images with Builder

* adding customer modules and environment variables

RUN : Execute a bash command

ENV : Define environment variables

RUN pecl install xdebug-2.6.0

* installs the module

RUN docker-php-ext-enable xdebug

* enable the module

RUN pecl install redis-4.0.1

RUN docker-php-ext-enable redis

ENV REDIS\_HOST redis

Learning docker 06: Docker bridge network

Network driver summary

* **User-defined bridge networks** are best when you need multiple containers to communicate on the same Docker host.
* **Host networks** are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.
* **Overlay networks** are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.
* **Macvlan networks** are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.
* **Third-party network plugins** allow you to integrate Docker with specialized network stacks.

Bridge networks:

* User-defined bridges provide better isolation and interoperability between containerized applications.
* User-defined bridges provide automatic DNS resolution between containers**.**
* Containers can be attached and detached from user-defined networks on the fly**.**
* Each user-defined network creates a configurable bridge**.**
* Linked containers on the default bridge network share environment variables**.**

website B

Memcache

Ubuntu

default

MongoDB

Java

MySQL

PHP + Apache

website A

**# Create a network**

$ docker network create mynetwork

$ docker network ls

**# Create container within your network**

$ docker run -d -p 8080:80 --network mynetwork --name myhost2 nginx

$ docker run -it --mynetwork tutum/curl bash # spin up container on network - if no network, couldn’t curl myhost2

$ docker network disconnect mynetwork myhost2 # disconnect

$ docker network connect mynetwork myhost2 # re-connect

Learning docker 07: Advanced networking with PHP & MySQL

Create network consisting of:

* 2 bridge networks
* 1 MySQL Container
* 1 Apache + PHP Container

Networks:

\* Docker automatically assigns gateways to networks

PHP + Apache

Gateway

Frontend

Backend

Gateway

MySQL

Containers (PHP + Apache) can sit across / attached to multiple networks

MySQL

PHP + Apache

Frontend

Backend

Gateway

Gateway

**.env**

MYSQL\_USER=sys\_admin

MYSQL\_PASSWORD=sys\_password

MYSQL\_ROOT\_PASSWORD=root\_password

**docker-compose.yml**

version: "3"

networks:

tut07-frontend:

driver: bridge

ipam:

driver: default

config:

- subnet: 172.10.1.0/24

tut07-backend:

driver: bridge

ipam:

driver: default

config:

- subnet: 172.10.2.0/23

services:

tut07-db:

build: ./db

command: --default-authentication-plugin=mysql\_native\_password

ports:

- 3306:3306

volumes:

- ./src:/var/www/html

networks:

- tut07-backend:

ipv4\_address: 172.10.3.2

env\_file:

- ./development.env

tut07-www:

build: ./www

ports:

- 8080:80

networks:

- tut07-backend:

ipv4\_address: 172.10.2.2

- tut07-frontend:

ipv4\_address: 172.10.1.2

depends\_on:

- tut07-db

env\_file:

- ./development.env

**db/Dockerfile**

FROM mysql:8.0

**web/Dockerfile**

FROM php:7.2-apache

RUN docker-php-ext-install mysqli

RUN docker-php-ext-enable mysqli

RUN docker-php-ext-install xdebug

RUN docker-php-ext-enable xdebug

Frontend

172.10.1.0 - 172.10.1.255

Backend

172.10.2.0 - 172.10.3.255

Gateway

172.10.2.1

MySQL

172.10.3.2

Gateway

172.10.1.1

PHP + Apache

172.10.1.2/172.10.2.2