**Docker Commands, Help & Tips**

**Show commands & management commands**

$ docker

**Docker version info**

$ docker version

**Show info like number of containers, etc**

$ docker info

**WORKING WITH CONTAINERS**

**Create an run a container in foreground**

$ docker container run -it -p 80:80 nginx

**Create an run a container in background**

$ docker container run -d -p 80:80 nginx

**Shorthand**

$ docker container run -d -p 80:80 nginx

**Naming Containers**

$ docker container run -d -p 80:80 --name nginx-server nginx

**TIP: WHAT RUN DID**

* Looked for image called nginx in image cache
* If not found in cache, it looks to the default image repo on Dockerhub
* Pulled it down (latest version), stored in the image cache
* Started it in a new container
* We specified to take port 80- on the host and forward to port 80 on the container
* We could do "$ docker container run --publish 8000:80 --detach nginx" to use port 8000
* We can specify versions like "nginx:1.09"

**List running containers**

$ docker container ls

OR

$ docker ps

**List all containers (Even if not running)**

$ docker container ls -a

**Stop container**

$ docker container stop [ID]

**Stop all running containers**

$ docker stop $(docker ps -aq)

**Remove container (Can not remove running containers, must stop first)**

$ docker container rm [ID]

**To remove a running container use force(-f)**

$ docker container rm -f [ID]

**Remove multiple containers**

$ docker container rm [ID] [ID] [ID]

**Remove all containers**

$ docker rm $(docker ps -aq)

**Get logs (Use name or ID)**

$ docker container logs [NAME]

**List processes running in container**

$ docker container top [NAME]

**TIP: ABOUT CONTAINERS**

Docker containers are often compared to virtual machines but they are actually just processes running on your host os. In Windows/Mac, Docker runs in a mini-VM so to see the processes youll need to connect directly to that. On Linux however you can run "ps aux" and see the processes directly

**IMAGE COMMANDS**

**List the images we have pulled**

$ docker image ls

**We can also just pull down images**

$ docker pull [IMAGE]

**Remove image**

$ docker image rm [IMAGE]

**Remove all images**

$ docker rmi $(docker images -a -q)

**TIP: ABOUT IMAGES**

* Images are app bianaries and dependencies with meta data about the image data and how to run the image
* Images are no a complete OS. No kernel, kernel modules (drivers)
* Host provides the kernel, big difference between VM

**Some sample container creation**

NGINX:

$ docker container run -d -p 80:80 --name nginx nginx (-p 80:80 is optional as it runs on 80 by default)

APACHE:

$ docker container run -d -p 8080:80 --name apache httpd

MONGODB:

$ docker container run -d -p 27017:27017 --name mongo mongo

MYSQL:

$ docker container run -d -p 3306:3306 --name mysql --env MYSQL\_ROOT\_PASSWORD=123456 mysql

**CONTAINER INFO**

**View info on container**

$ docker container inspect [NAME]

**Specific property (--format)**

$ docker container inspect --format '{{ .NetworkSettings.IPAddress }}' [NAME]

**Performance stats (cpu, mem, network, disk, etc)**

$ docker container stats [NAME]

**ACCESSING CONTAINERS**

**Create new nginx container and bash into**

$ docker container run -it --name [NAME] nginx bash

* i = interactive Keep STDIN open if not attached
* t = tty - Open prompt

**For Git Bash, use "winpty"**

$ winpty docker container run -it --name [NAME] nginx bash

**Run/Create Ubuntu container**

$ docker container run -it --name ubuntu ubuntu

**(no bash because ubuntu uses bash by default)**

**You can also make it so when you exit the container does not stay by using the -rm flag**

$ docker container run --rm -it --name [NAME] ubuntu

**Access an already created container, start with -ai**

$ docker container start -ai ubuntu

**Use exec to edit config, etc**

$ docker container exec -it mysql bash

**Alpine is a very small Linux distro good for docker**

$ docker container run -it alpine sh

(use sh because it does not include bash) (alpine uses apk for its package manager - can install bash if you want)

**NETWORKING**

**"bridge" or "docker0" is the default network**

**Get port**

$ docker container port [NAME]

**List networks**

$ docker network ls

**Inspect network**

$ docker network inspect [NETWORK\_NAME]

("bridge" is default)

**Create network**

$ docker network create [NETWORK\_NAME]

**Create container on network**

$ docker container run -d --name [NAME] --network [NETWORK\_NAME] nginx

**Connect existing container to network**

$ docker network connect [NETWORK\_NAME] [CONTAINER\_NAME]

**Disconnect container from network**

$ docker network disconnect [NETWORK\_NAME] [CONTAINER\_NAME]

**Detach network from container**

$ docker network disconnect

**IMAGE TAGGING & PUSHING TO DOCKERHUB**

**tags are labels that point ot an image ID**

$ docker image ls

Youll see that each image has a tag

**Retag existing image**

$ docker image tag nginx btraversy/nginx

**Upload to dockerhub**

$ docker image push bradtraversy/nginx

**If denied, do**

$ docker login

**Add tag to new image**

$ docker image tag bradtraversy/nginx bradtraversy/nginx:testing

**DOCKERFILE PARTS**

* FROM - The os used. Common is alpine, debian, ubuntu
* ENV - Environment variables
* RUN - Run commands/shell scripts, etc
* EXPOSE - Ports to expose
* CMD - Final command run when you launch a new container from image
* WORKDIR - Sets working directory (also could use 'RUN cd /some/path')
* COPY # Copies files from host to container

**Build image from dockerfile (reponame can be whatever)**

**From the same directory as Dockerfile**

$ docker image build -t [REPONAME] .

**TIP: CACHE & ORDER**

* If you re-run the build, it will be quick because everythging is cached.
* If you change one line and re-run, that line and everything after will not be cached
* Keep things that change the most toward the bottom of the Dockerfile

**EXTENDING DOCKERFILE**

**Custom Dockerfile for html paqge with nginx**

FROM nginx:latest # Extends nginx so everything included in that image is included here

WORKDIR /usr/share/nginx/html

COPY index.html index.html

**Build image from Dockerfile**

$ docker image build -t nginx-website

**Running it**

$ docker container run -p 80:80 --rm nginx-website

**Tag and push to Dockerhub**

$ docker image tag nginx-website:latest btraversy/nginx-website:latest

$ docker image push bradtraversy/nginx-website

**VOLUMES**

**Volume - Makes special location outside of container UFS. Used for databases**

**Bind Mount -Link container path to host path**

**Check volumes**

$ docker volume ls

**Cleanup unused volumes**

$ docker volume prune

**Pull down mysql image to test**

$ docker pull mysql

**Inspect and see volume**

$ docker image inspect mysql

**Run container**

$ docker container run -d --name mysql -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=True mysql

**Inspect and see volume in container**

$ docker container inspect mysql

**TIP: Mounts**

* You will also see the volume under mounts
* Container gets its own uniqe location on the host to store that data
* Source: xxx is where it lives on the host

**Check volumes**

$ docker volume ls

**There is no way to tell volumes apart for instance with 2 mysql containers, so we used named volumes**

**Named volumes (Add -v command)(the name here is mysql-db which could be anything)**

$ docker container run -d --name mysql -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=True -v mysql-db:/var/lib/mysql mysql

**Inspect new named volume**

docker volume inspect mysql-db

**BIND MOUNTS**

* Can not use in Dockerfile, specified at run time (uses -v as well)
* ... run -v /Users/brad/stuff:/path/container (mac/linux)
* ... run -v //c/Users/brad/stuff:/path/container (windows)

**TIP: Instead of typing out local path, for working directory use $(pwd):/path/container - On windows may not work unless you are in your users folder**

**Run and be able to edit index.html file (local dir should have the Dockerfile and the index.html)**

$ docker container run -p 80:80 -v $(pwd):/usr/share/nginx/html nginx

**Go into the container and check**

$ docker container exec -it nginx bash

$ cd /usr/share/nginx/html

$ ls -al

**You could create a file in the container and it will exiost on the host as well**

$ touch test.txt

**DOCKER COMPOSE**

* Configure relationships between containers
* Save our docker container run settings in easy to read file
* 2 Parts: YAML File (docker.compose.yml) + CLI tool (docker-compose)

**1. docker.compose.yml - Describes solutions for**

* containers
* networks
* volumes

**2. docker-compose CLI - used for local dev/test automation with YAML files**

**Sample compose file (From Bret Fishers course)**

version: '2'

# same as

# docker run -p 80:4000 -v $(pwd):/site bretfisher/jekyll-serve

services:

jekyll:

image: bretfisher/jekyll-serve

volumes:

- .:/site

ports:

- '80:4000'

**To run**

docker-compose up

**You can run in background with**

docker-compose up -d

**To cleanup**

docker-compose down