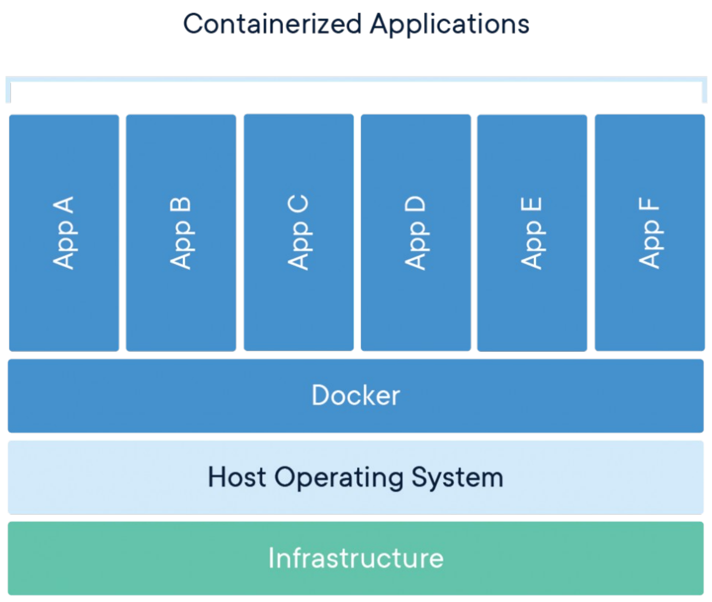
Docker

containerizing applications

What is docker?

- “Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines.”

  -- source: WikiPedia



source: docker.com

How does it work?

- Dockerfile

$$\Big\downarrow \text{build}$$

- Image

$$\Big\downarrow \text{run}$$

- Container

How does it work?

$$\begin{tikzcd}
\bullet\ \text{Dockerfile} \arrow[d, "\text{build}"] & \\
\bullet\ \text{Image} \arrow[r, "\text{push}"] \arrow[d, "\text{run}"] & \text{Registry (“publish image”)} \\
\bullet\ \text{Container} &
\end{tikzcd}$$

How does it work?

- Registry (“published image”)

  $\Big\downarrow$ pull

- Image

  $\Big\downarrow$ run

- Container

Using docker

- Using existing images

```
docker run <image[:version]>
```

```
docker run hello-world
```

- Output

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:49a1c8800c94df04e9658809b006fd8a686cab8028d33cfba2cc049724254202
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
...
```

Using docker

- Listing images

```
docker image ls
```

```
REPOSITORY      TAG             IMAGE ID          CREATED             SIZE
hello-world     latest          bf756fb1ae65      6 months ago        13.3kB
```

- Listing containers

`docker ps` (only running ones)
`docker ps -a` (all)

```
CONTAINER ID  IMAGE       COMMAND  CREATED    STATUS                    PORTS  NAMES
4e3eeed5cb9a  hello-world "/hello" 4 min ago  Exited (0) 4 min ago             sleepy_haibt
```

Using docker

- Stop containers

```
docker stop $(docker ps -a -q)
```

- Remove containers

```
docker rm $(docker ps -a -q)
```

- Prune everything

```
docker system prune -a
```

Dockerfile

```
# Use the official image as a parent image
FROM node:current-slim
# Set the working directory
WORKDIR /usr/src/app
# Copy the file from your host to your current location
COPY package.json .
# Run the command inside your image filesystem
RUN npm install
# Add metadata to image to describe which port the container is listening on
EXPOSE 8080
# Run the specified command within the container
CMD [ "npm", "start" ]
# Copy the rest of your app's code from your host to your image filesystem
COPY . .
```

source: Sample Dockerfile

Reference

Mapping

- Mapping directories
  - volumes
    - `-v/--volume /local/dir:/container/dir:ro`
  - bind volumes
    - `--mount type=bind source=/local/dir,target=/cont/dir,readonly`
  - “volume” combines everything in one expression, “mount” uses key-value pairs (more verbose)
- Mapping ports
  - container networking
    - -p [HOST_IP:]HOST_PORT:CONT_PORT[/PROTOCOL]
    - -p 8080:80
    - -p 192.168.1.100:8080:80/tcp -p 192.168.1.100:8080:80/udp

nginx example

- Images available at

  https://hub.docker.com/_/nginx/
- HTML files in `/local/dir/html`
- Run as daemon (“`-d`”) on port 8080 on host
- command-line

```
docker run \
  -v /local/dir/html:/usr/share/nginx/html \
  -p 8080:80 -d nginx[:latest]
```

Interactive containers

- Containers can be run interactively using `-it` (`--interactive –tty`)
- Can be confusing what container you are in
- use custom `/etc/bash.bashrc`
- generate them with docker-banner-gen

Interactive containers

- Example

```
 _____ _____    ___  _     _ ____       _   
|_   _|  ___|  / _ \| |__ (_)  _ \  ___| |_ 
  | | | |_    | | | | '_ \| | | | |/ _ \ __|
  | | |  _|   | |_| | |_) | | |_| |  __/ |_ 
  |_| |_|      \___/|_.__// |____/ \___|\__|
                        |__/                

1.14.0_2019-08-31

WARNING: You are running this container as root, which can cause new files in
mounted volumes to be created as the root user on your host machine.

To avoid this, run the container by specifying your user's userid:

$ docker run -u $(id -u):$(id -g) args...

tf-objdet / >
```

Pitfalls

- docker hashes the commands
  - layer doesn’t get regenerated if hash hasn’t changed – “`pip install NAME`” won’t trigger a rebuild without version number change!
- temporary files from a command need to get removed as part of that command to reduce size of layer
  - apt/pip require cleaning

```
RUN apt-get update && \
    apt-get install -y --no-install-recommends vim &&\
    pip install Cython && \
    rm -Rf /root/.cache/pip && \
    rm -rf /var/lib/apt/lists/*
```

Pitfalls

- containers run (usually) as root, files generated in volumes will have root/root as user/group
  - run as different user: `-u UID:GID`
  - as current user: `-u $(id -u):$(id -g)`
- "`docker login`" (for private registries) stores passwords unencrypted in base64
  - credentials stores