Introduction to Docker

Matthew Treinish
Developer Advocate - IBM
mtreinish@kortar.org
`mtreinish on Freenode`
https://github.com/mtreinish/intro-to-docker

July 9th, 2018

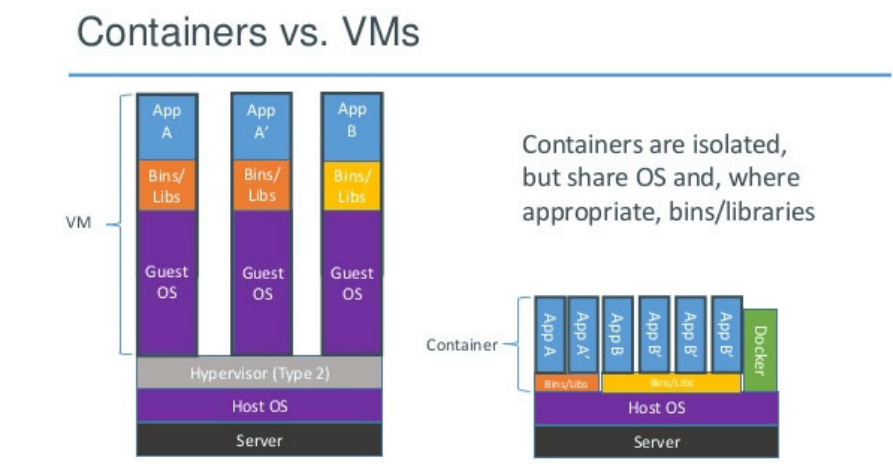
What is Docker?

- Tooling and platform to manage containers
- Manages the lifecycle of containers
- Simplified interface on top of existing technologies for ease of use
- Works on Linux, Mac OS X, and Windows

Containers

- A group of processes run in isolation
  - Similar to VMs but managed at the process level
  - Run on a shared kernel
- Each container has its own namespaces
  - **PID** process IDs
  - **USER** user and group IDs
  - **UTS** hostname and NIS domain name
  - **NS** mount points
  - **NET** network devices, stacks, ports, etc.
  - **IPC** inter-process communications, message queues
  - **cgroups** controls limits and monitoring of resources

Containers vs VMs



Why use containers?

- Most of the same reasons as VMs (like isolation)
- Faster startup time, just the time to:
  - Create new directory
  - Setup the container's filesystem
  - Setup network, mounts, etc
  - Start the process
- Better Resource utilization
- Reusability and distributability

Installing Docker

- On Linux - Distro Packages
- From Docker:
  - Mac: https://docs.docker.com/docker-for-mac/install/
  - Windows: https://docs.docker.com/docker-for-windows/install/
  - Ubuntu: https://docs.docker.com/v17.09/engine/installation/linux/docker-ce/ubuntu/

First container

$ docker run ubuntu echo Hello World

What Happened

- Docker created a directory with an Ubuntu filesystem (image)
- Docker created a new set of namespaces
- Ran a new process: echo Hello World
- Using those namespaces to isolate it from other processes
- Using that new directory as the root of the filesystem (chroot)
- Notice as a user I never installed Ubuntu
- Run it again, notice how quickly it ran

ssh-ing into a container

$ docker run -ti ubuntu bash

What Happened

- Now the process is *bash* instead of *echo*
- But its still just a process
- Look around, mess around, its isolated

Getting data into a container

- Using env variables:
  $ docker run -e INPUT=IamSECURE -P ubuntu bash
- Using Volumes:
  $ mkdir -p /tmp/volume && echo IamSECURE > /tmp/volume/pass
  $ docker run –i –t –v /tmp/volume:/volume ubuntu bash

Look under the covers

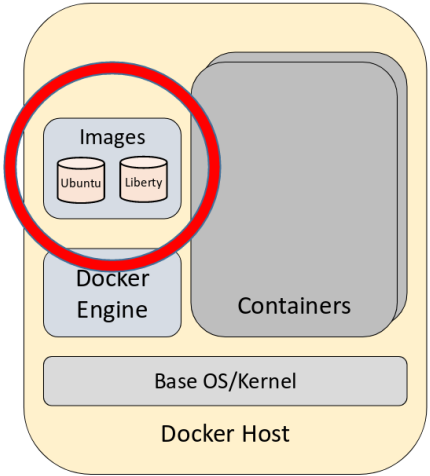
$ docker run ubuntu ps -ef

Things to notice with these examples

- Each container only sees its own processes
- Running as root
- Running as PID 1

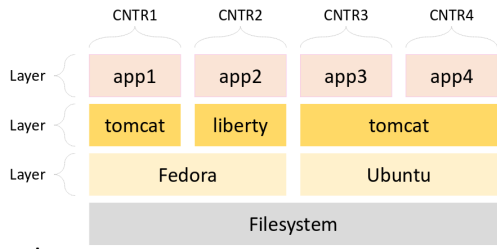
Docker images

- Tarball of container's filesystem and metadata
- Makes sharing and distribution of containers easy
- Applications are packaged as images



Layering

- Docker uses a copy-on-write filesystem
- New files (or modifications) are only visible to current/above layers
- Layers allow for reuse
- Images are tarballs of layers



Dockerhub

https://hub.docker.com

- Public registry of Docker Images
- Hosted by Docker Inc.
- Free for public images
- By default docker engines will look in DockerHub for images
- Browser interface for searching, descriptions of images

Pick an application

Look at dockerhub and find a container for that application.

$ docker run nginx

What Happened

- Pulled the nginx:latest image form dockerhub
  - Dockerhub entry: https://hub.docker.com/_/nginx/
  - Layered ontop of debian:stretch-slim image: https://hub.docker.com/_/debian/
- Run that container
- No configuration, just a base nginx

Tie it all together

```
$ docker run -P -v examples/nginx/content:/usr/share/nginx/html:ro nginx
```

The Dockerfile

Reference Guide: https://docs.docker.com/engine/reference/builder/

- Input script to build images
- Important instructions:
  - **FROM** - Set base image either another Dockerfile or from a registry
  - **RUN** - Run a command inside a new layer
  - **COPY** - Copy files or directories into the filesystem of the container
  - **CMD** - Set a default command for executing a container
  - **EXPOSE** - Specify a port the container listens on

Example Dockerfile

```
1  FROM python:3.6
2
3  RUN apt-get update
4  RUN apt-get install -y build-essential musl-dev libxml2-dev git
5  RUN pip3 install -U pymysql
6  RUN pip3 install -U uwsgi
7  RUN git clone git://git.openstack.org/openstack/openstack-health
8  RUN pip3 install -U ./openstack-health
9
10 RUN cp openstack-health/etc/openstack-health-api.conf
   ↪  /etc/openstack-health.conf
11
12 EXPOSE 80
13
14 CMD ["/usr/local/bin/uwsgi", "--http", ":80", "--wsgi-file",
   ↪  "/usr/local/bin/openstack-health"]
```

Building a Service image

```
1  FROM alpine:3.7
2
3  LABEL Description="Eclipse Mosquitto MQTT Broker"
4
5  RUN apk --no-cache add mosquitto=1.4.15-r0
6  RUN mkdir -p /mosquitto/config /mosquitto/data /mosquitto/log
7  RUN cp /etc/mosquitto/mosquitto.conf /mosquitto/config
8
9  COPY mosquitto.conf /mosquitto/config/mosquitto.conf
10 COPY mqtt_acl.conf /mosquitto/config/mqtt_acl.conf
11
12 RUN chown -R mosquitto:mosquitto /mosquitto
13
14 COPY run.sh /root/run.sh
15
16 EXPOSE 1883
17 EXPOSE 80
18
19 CMD ["/root/run.sh"]
```

- **$ docker build -t mymosquitto examples/mosquitto**
- **$ docker run -e MQTT_PASS=SecurePASS -P mymosquitto**

Back to nginx

Build an image with the same content as our custom nginx before:

```
1 FROM nginx
2
3 COPY content /usr/share/nginx/html
```

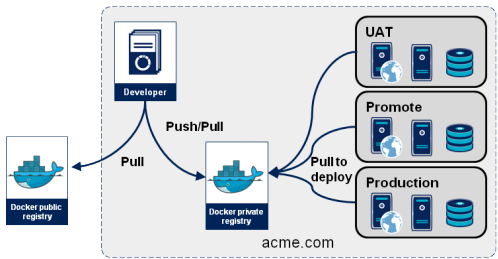
- **$ docker build -t mycustomhtml examples/nginx**
- **$ docker run -P mycustomhtml**

What happened

- Created a new image built on top of the nginx image from Docker Hub
- It copies the content directory into the nginx shared content directory
- Then run that image to get

Local Image Registry

- You can run a local registry to share images
- Basically your own local Docker hub
- Let's you control:
  - Where your images are being stored
  - The image creation and distribution pipeline



Running your own registry

```
$ docker run -d -p 5000:5000 name registry registry
```

What happened

- Pulled the registry container from dockerhub
- Launched the container as a daemon (in the background)
- Maps localhost port 5000 to port 5000 on the container

Using a Local Registry

```
$ docker pull debian
$ docker image tag debian localhost:5000/myspecialimage
$ docker push localhost:5000/myspecialimage
$ docker pull localhost:5000/myspecialimage
```

What happened

- Pulled the latest debian image from dockerhub
- Tagged that image off the local registry
- Push that tagged image to the local registry
- Pull that image from the registry to the local machine

Conclusion

- Docker provides a simple to use interface for containers
- Containers provide fast and lightweight isolation for applications
- Docker enables building packages that are easily deployable
- Large ecosystem of applications built for Docker
- Image layering makes it simple to build off of existing images

Where to get more information

- These Slides: https://github.com/mtreinish/intro-to-docker
- Docker tutorial: https://github.com/docker/labs/tree/master/beginner
- Docker documentation: https://docs.docker.com/
- Best practice for Dockerfiles: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/