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
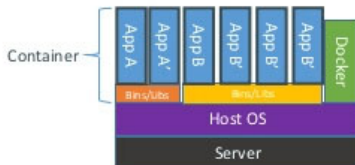
Containers

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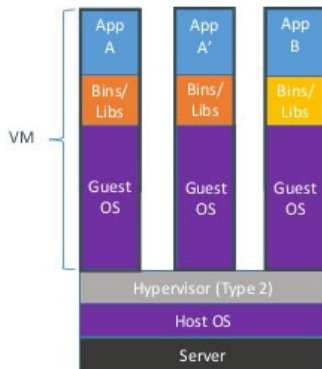
Containers vs VMs

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

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

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

Getting data into a container

- Using env variables:
  $ docker run -e INPUT=IamSECURE -P ubuntu bash

- Using Volumess:
  $ mkdir -p /tmp/volume && echo IamSECURE > /tmp/volume/pass
  $ docker run –i –t –v /tmp/volume:/volume ubuntu bash

Docker images

Layering

- Docker uses a copy-on-write (union) 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

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

```
FROM python:3.6

RUN apt-get update
RUN apt-get install -y build-essential musl-dev
↪    libxml2-dev git
RUN pip3 install -U pymysql
RUN pip3 install -U uwsgi
RUN git clone
↪    git://git.openstack.org/openstack/openstack-health
RUN pip3 install -U ./openstack-health

RUN cp
↪    openstack-health/etc/openstack-health-api.conf
↪    /etc/openstack-health.conf

EXPOSE 80

CMD ["/usr/local/bin/uwsgi", "--http", ":80",
↪    "--wsgi-file",
↪    "/usr/local/bin/openstack-health"]
```

Running your own registry

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

Using a Local Registry

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

Where to get more information

- 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/