Docker

An introduction for the VM initiate

Docker

VMs are heavy

- Lots of disk space
- Lots of duplicate OS processes to run
- Crude resource management between guest and master
- Unnecessary battery drain
- Still useful enough to be worth it

Docker containers are different

- Are running process groups, started a single command
- Have their own isolated writable storage
- Are based on common images
- Have isolated networking stacks
- Are very cheap to create and destroy
- Simple mental model - per process VMs

Simple black box view

- Adds capabilities - every app deploys the same way on the same host.
- Composition of containers
- Dependency management
- Automation of deployment
- e.g. Fig, Geard, Decking, Centurion, Octohost, Panamax

Docker is light

- Linux cgroups provides cpu/memory/io isolation and identifier namespaces
- Runs only the processes you tell it to - no services.
- Better resource management than VMs - one kernel handles all processes interactively
- Images stored in layered union filesystems to save space.
- Cheap to create
- Can do everything VMs can, if you want a Linux guest

It all sounds so easy…

- It is, but it’s very new
- There are new ways of doing things to learn
    - start a shell in a running container
    - connect up containers
    - expose containers to the outside world

Getting docker

- Linux
    - Kernel 3.8 or later, 3.10 advised.
- Windows/Mac - boot2docker
    - Runs a small Linux VM that containers run in
- http://docker.com/ - really good documentation

A toy example

Starting a shell in Ubuntu Linux:

```
docker run --rm -it ubuntu bash
```

The ubuntu command

Delete when done

The image - Ubuntu

Interactive, with TTY

More toy examples

Starting a shell in Ubuntu Linux:

```
docker run --rm -it ubuntu bash
```

Or Centos:

```
docker run --rm -it centos bash
```

Starting a database server:

```
docker run -d --name postgres postgres
```

(no command)

Named container

Daemon

Data volumes

Run database server in container with data volume

```
docker run -d --restart=always --name postgres  -v /
var postgres
```

Run database server with host directory as a data volume

```
docker run -d --restart=always --name postgres  -v /
data/postgres:/var postgres
```

Run container with access to data volumes from another

```
docker run -d --restart=always --volumes-
from=postgres postgres-shell bash
```

Linked containers

Run database server in container

```
docker run -d --restart=always --name postgres
postgres
```

Run mediawiki in container, linked to the database

```
docker run -d --restart=always --link
postgres:db synctree/mediawiki
```

Danger Will Robinson!

- Docker's isolation can be disabled with options to `docker run ...`
  - --privileged
  - --cap-add
  - --net=host
  - --lxc-conf
- Review commands before running them.

Building an image

First we need a Dockerfile - this tells docker how to build the image.

This will specify:

- a base image
- the changes to make to the base
- the command that the image will run

Hello, world!

```
FROM ubuntu

MAINTAINER Ronan Klyne <docker@rklyne.net>

ENV NAME world

CMD echo "Hello, ${NAME}!"

$ docker build -t hello-world .
$ docker run --rm -it hello-world
Hello, world!

$ docker run --rm -it -e NAME="geeks" hello-world
Hello, geeks!
```

Bigger things

```
FROM ubuntu

MAINTAINER Ronan Klyne <docker@rklyne.net>

RUN apt-get update -y && \
    apt-get install -y python

CMD python -c "print 'Hello, world!'"


$ docker build -t hello-world .
...
$ docker run --rm -it hello-world
```

A webapp!

```
FROM ubuntu

MAINTAINER Ronan Klyne <docker@rklyne.net>

RUN apt-get update -y && \
    apt-get install -y python

EXPOSE 80

CMD python -c "from wsgiref import simple_server;
simple_server.make_server('0.0.0.0', 80,
simple_server.demo_app).serve_forever()"

$ docker build -t hello-world .
...
$ docker run --rm -it -p 8001:80 hello-world
```

A Django app with a database

```
FROM ubuntu

MAINTAINER Ronan Klyne <docker@rklyne.net>

RUN apt-get update -y && \
    apt-get install -y python

EXPOSE 80

ADD schema-update.sh /schema-update.sh

CMD /schema-update.sh && ./manage.py devserver --
settings=myapp.settings.docker 0.0.0.0:80

$ docker build -t hello-world .
...
$ docker run --rm -it -p 8001:80 --link postgres:db hello-world
```

Working on containers

Run database server in container

```
docker run -d --restart=always --name postgres
postgres
```

Run database shell

```
docker run --rm -it --link postgres:db postgres
psql -h db -U postgres
```

```
docker exec -it postgres psql -U postgres
```

```
docker exec -it postgres bash
```

An exercise for the reader

- docker ps/kill
- docker save/load
- docker stop/start
- docker images
- docker rm/rmi

Tips and tricks

- Multiple processes per container - use supervisord
- Preserving data - use data volumes
- Order your Dockerfiles. Remember the caching order
- Makefile. It's a good way of keeping your commands in order.

Questions?

Ronan Klyne

@ronanklyne

http://github.com/rklyne/docker-examples

docker@rklyne.net