Using Docker in Production

Brian Prodoehl
CTO, Connectify
bprodoehl@connectify.me
@BrianProdoehl
Slides at https://github.com/bprodoehl/DevOpsMeetup

What do we do?

We make consumer-oriented networking software that makes complex things easy.

It should be easy to share the Internet from your PC to all your devices.

Connectify Hotspot

Turns your PC into a Wi-Fi Hotspot

- >65 Million installs
- >500 Million hotspots

It should be easy to use all of your Internet connections.

Cafe Wi-Fi + tethered 4G? Why not?

Speedify

Software-as-a-Service for your Mac and PC that speeds up your Internet by combining all of your connections through our servers

Speedify Infrastructure

Proximity to our customers is vital, so by the numbers:

- 24 data centers
- 12 countries
- 8 hosting providers

Technologies Involved

Speedify's backend is a combination of C++ and Node.js, running entirely in Docker containers.

What is Docker?

What is Docker?

Open source project to bring containers to the Linux mainstream

Docker Engine - runtime and packaging tool

Docker Hub - cloud service for storing and sharing applications

What is a container?

- A container is a bundle of software with its dependencies, that runs isolated from other things on the host
- Many of the benefits of virtualization, but runs on the host's kernel, so it is fast and lean

What is a container?

A container can be as simple as just holding files used by other containers...

or as complex as a full LAMP stack (and beyond)

Basics: Boot2Docker

Docker is still Linux-only

Boot2Docker packages up VirtualBox, a lightweight Linux VM, and a simple CLI to bring Docker to OS X and Windows

Basics: the Dockerfile

A Dockerfile is used to define a container

```
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y nginx
RUN echo "daemon off;" >> /etc/nginx/nginx.conf
EXPOSE 80
CMD /usr/sbin/nginx
```

Basics: building a container

The Docker Engine reads a Dockerfile and creates a container

```
$ docker build -t example1 .
Sending build context to Docker daemon  2.56 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:14.04
 ---> 96864a7d2df3
Step 1 : RUN apt-get update && apt-get install -y nginx
...
```

Basics: layers

A container is comprised of multiple filesystem layers

- rebuilding or downloading after changing a Dockerfile can leverage cached layers
- different (but similar) containers can share the same base layers

Basics: exposing ports

Define which ports should be exposed in the Dockerfile

At runtime, specify manual mapping of host port -> container port, or let Docker do it dynamically

Why did we choose Docker?

Why Docker?

Docker provides a consistent interface on top of very different hosting providers

Why Docker?

Just need Ubuntu and SSH, with Docker installed on top

Spin up, tear down and deploy new application servers without relying on hosting provider API - no provider lock-in!

Why Docker?

Easily decentralize, and push as much of our backend out to the edge as we like

- Very easy to deploy different server processes to hosts wherever we need them
- Mix-and-match processes on any given host

Why Docker?

Docker enables us to upgrade a stateful C++ daemon without interrupting our customers.

10 Lessons Learned

Lesson #1

Use a "lightweight virtual machine" container model **where it makes sense**

Lesson #1 - Lightweight VMs

Single process per container can lead to needless complexity with no gain

There are many ways to run multiple processes in a container. Some are better than others.

Lesson #1 - Lightweight VMs

Example: nginx + syslog + cron

- Approach #1: single process per container
- Approach #2: script to kick off all three
- Approach #3: init daemon

Lesson #1 - Lightweight VMs

Approach #1: single process per container

- One container runs nginx
- Run rsyslogd in another container, or on the host
- Run cron in another container

Lesson #1 - Lightweight VMs

Problems with Approach #1

- Pretty high complexity
- You're assuming things about the host environment
- You need to export volumes so syslog and cron can do their job

Lesson #1 - Lightweight VMs

Approach #2: script to kick off all three

- Container's command is a shell script

```
/usr/sbin/cron &
/usr/sbin/rsyslogd &
/usr/sbin/nginx
```

Lesson #1 - Lightweight VMs

Problems with Approach #2

- What if a process crashes?
- It's easy to outgrow this

Lesson #1 - Lightweight VMs

Approach #3: use an init daemon

- Container's command is an init daemon (runit, supervisord, whatever you like)
- You tell the init daemon what services to run

Lesson #1 - Lightweight VMs

Advantages of an init daemon

- Processes can be restarted if they crash
- Modular, lends itself to cookbook-style code sharing

Lesson #1 - Lightweight VMs

baseimage-docker from Phusion is a great starting point for lightweight-VM style containers

Lesson #2 - Service Discovery

A good service discovery system is a must!

Lesson #2 - Service Discovery

It is easy to lose track of what is running where.

Manual port mappings can get unwieldy, and dynamic port mappings can be difficult to sort out.

Lesson #2 - Service Discovery

You need something to let your services advertise themselves as online.

Stable solutions available for this now:

- etcd
- Consul
- discoverd

Lesson #3

Configuration management is a must!

Lesson #3 - Configuration

Docker is not a replacement for Chef, Puppet, or Ansible. They work well together!

Lesson #3 - Configuration

You can pass in configuration numerous ways:

- etcd, Consul, and friends
- at container creation time
- at run time, by passing in environment variables

Lesson #4 - Bugs!

Docker can still be buggy

- Life is easier if you run Ubuntu on your hosts
- Team is very responsive
- Likely to see more bugs as a developer

Lesson #5 - Container Lifecycle

You have to manage the container lifecycle

- Finally Docker 1.2 added ability to automatically restart containers

Lesson #6 - Monitoring

Good continuous monitoring is a must!

Lesson #7 - Avoid SSH

Avoid SSH servers in your containers.

Lesson #7 - Avoid SSH

If you need to access your container's command line, use nsenter or docker-enter from the host.

Lesson #8 - Use Docker Hub

Running your own private registry can be a **serious undertaking**.

Lesson #8 - Use Docker Hub

Private repositories are available on the Docker Hub, and are pretty reasonable.

Lesson #9 - Teach your Devs!

Get your developers using Docker!

Lesson #9 - Teach your Devs!

It's easy to set your devs up with an entire staging setup.

Step 1: Install Docker or Boot2Docker

Step 2: Run this script

Lesson #10 - Keep on the Lookout

The Docker ecosystem is always changing, so keep on the lookout for great projects

Lesson #10 - Keep on the Lookout

Projects are maturing, and if you are starting out, these are well worth a look.

- Kubernetes
- CoreOS
- Flynn
- Shipyard

We’re hiring!

jobs@connectify.me

Thank You!

Brian Prodoehl
CTO, Connectify
bprodoehl@connectify.me
@BrianProdoehl
Slides at https://github.com/bprodoehl/DevOpsMeetup