Intro to Docker

Nathan White

#fococoders

Today's plan

- What is Containerization?
- Architecting Applications for Containers
- Docker Overview
- Building a Docker Image
- Docker Basics
- What is Container Orchestration?
- Current Orchestration Options
- Next Steps
- Q & A

What is a Container?

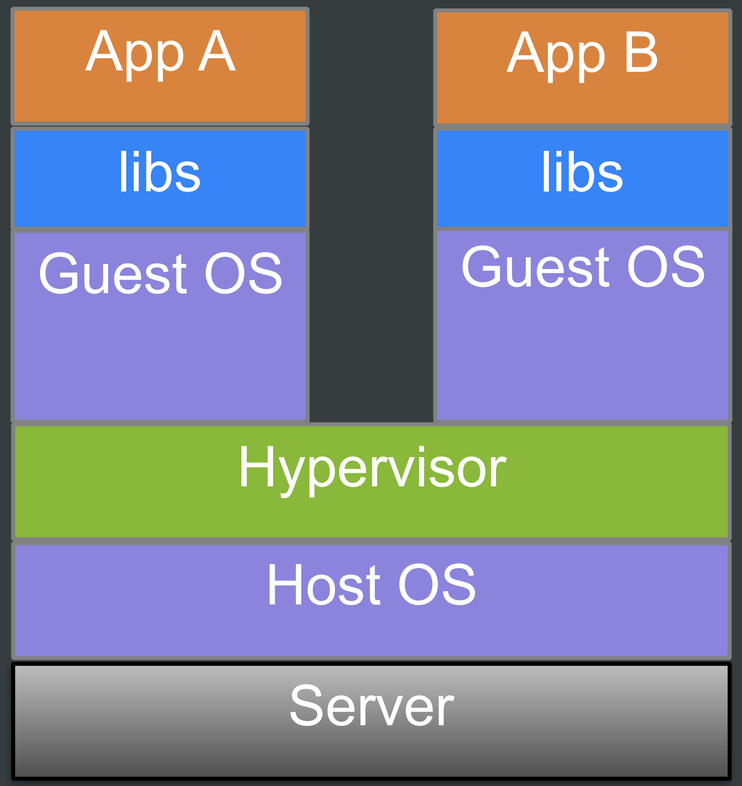
Containers

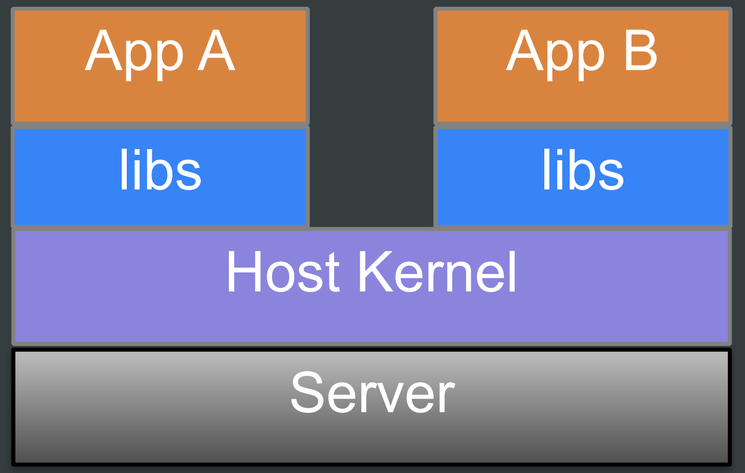
- Operating system level virtualization
- run multiple isolated Linux systems (containers) on a single kernel
- The Linux kernel provides **cgroups**
- cgroups - a feature that limits, accounts for and isolates resource usage (CPU, Memory, disk I/O, network, etc)
- Also known as **LXC** - Linux Containers

Why Containers

Application Containers

- Images: layered alternative to virtual machine disks
- Container: single process or application started from image
- Containerized application runs directly on host kernel





Virtual Machine vs Container

Container advantages

- Lower overhead compared to VMs
- Boot in milliseconds, not minutes
- “Run once, run anywhere”
- Mirrored production and development environments
- Scaling not tightly coupled to cloud-provider

Container use cases

- Microservice or distributed architectures
- A/B testing
- Capturing services and infrastructure as code

Containers = Happy Ops

- Immutability
- Explicit Dependencies
- Fast boot & restart
- Scale at process level

Manifestation of Infrastructure as Code

- Dialogue between Ops and Devs
- Pristine reference environments
- No configuration drift
- Kube/Compose etc. are to services what package.json is to node_modules

Architecting for Containers (Microservices)

Advantages

- Strong Module Boundaries
- Independent Deployment
- Technology Diversity
- Permissionless innovation - the ability of others to create new things on top of the communication constructs
- Disrupt Trust - effective model for evolving orgs that scale far beyond the limits of personal contact
- Accelerate Deprecations

Risks

- Dependency hell
- Distribution - harder to program
- Eventual Consistency
- Operational Complexity

Best Practices

- Create a Separate Data Store for each service
- Keep code at a similar level of maturity
- Do a separate build for each service
- Treat servers as stateless
- Deploy in containers

Warnings

- Different services do coordinated deployments
- A change in one service has consequences or requires a change in another service
- Services share a persistence store
- Engineers need intimate knowledge of the designs and schema of other services
- You have compliance controls that apply uniformly to all services
- Your infrastructure isn’t programmable
- You can’t do one-click deployments and rollbacks

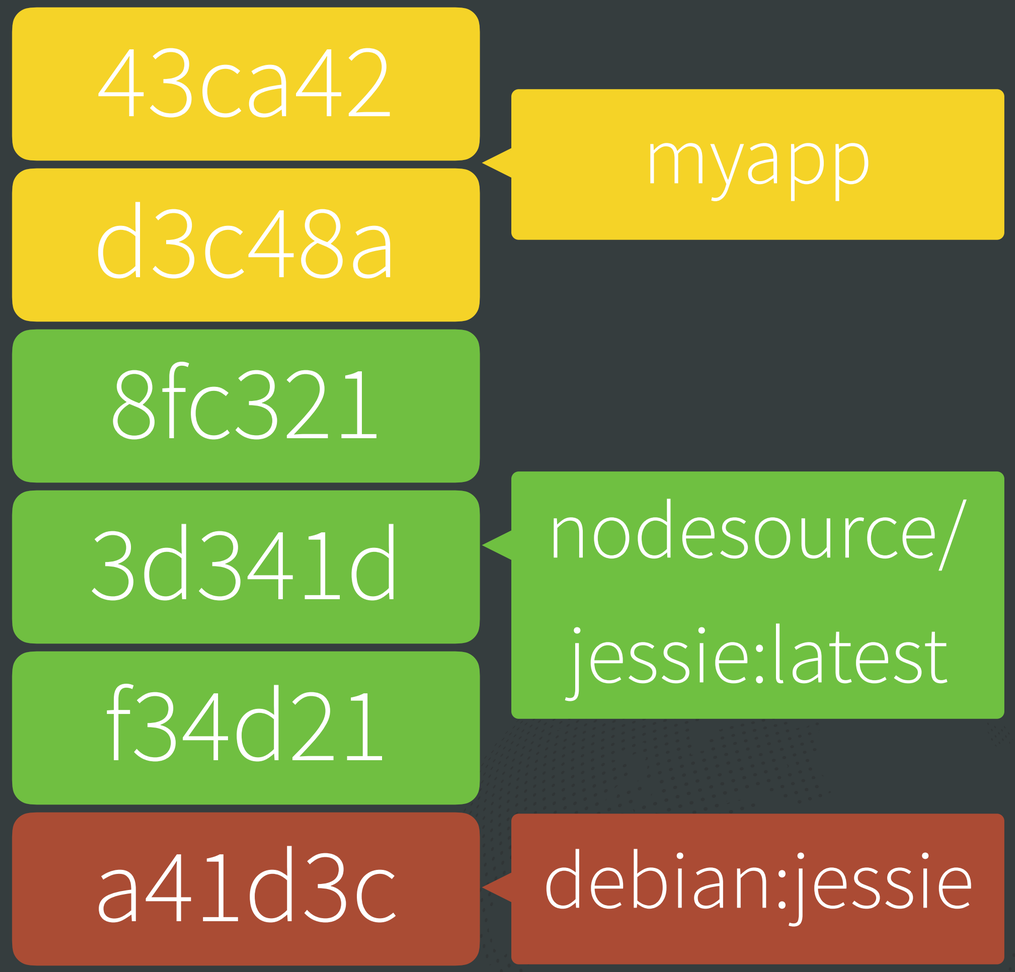
Docker Overview

Overview

- LXC (Linux Containers) and other Linux magic
- A layered filesystem for shipping images
- Docker containers are easily “linked” to other containers
- Images are built using a Dockerfile

Layers and Caching

- Docker commands create a new layers
- More layers equals larger container sizes
- Docker caches layers if no changes are made
- Similar to git style deltas



Layers and Caching

Docker Hub

- https://hub.docker.com
- Docker Hub is to Docker what GitHub is to git
- Create a login (if you don’t have one already)
- Link Docker cli tool to hub account `docker login`

Dockerizing Applications

```
1   FROM nodesource/node:4
2
3   RUN mkdir -p /home/nodejs/app
4   WORKDIR /home/nodejs/app
5
6   COPY . /home/nodejs/app
7   RUN npm install --production
8
9   CMD ["node", "index.js"]
10
```

Basic Dockerfile

```
1  FROM nodesource/node:4
2
3  RUN groupadd -r nodejs \
4    && useradd -m -r -g nodejs nodejs
5
6  USER nodejs
7
8  RUN mkdir -p /home/nodejs/app
9  WORKDIR /home/nodejs/app
```

Use non-root user

Dockerizing Applications

```
 8  RUN mkdir -p /home/nodejs/app
 9  WORKDIR /home/nodejs/app
10
11  COPY package.json /home/nodejs/app/package.json
12  RUN npm install --production
13  COPY . /home/nodejs/app
14
15  CMD ["node", "index.js"]
16
```

Cache node_modules

Dockerizing Applications

```
13  COPY . /home/nodejs/app
14
15  ENV NODE_ENV production
16
17  CMD ["node", "index.js"]
18
```

Setup Environment

```
1  # !/bin/sh
2  docker tag helloworld:latest yourorg/helloworld:$SHA1
3  docker tag helloworld:latest yourorg/helloworld:$BRANCH_NAME
4  docker tag helloworld:latest yourorg/build_$BUILD_NUM
5  docker push yourorg/helloworld:$SHA1
6  docker push yourorg/helloworld:$BRANCH_NAME
7  docker push yourorg/build_$BUILD_NUM
8
```

Use proper tags

Dockerizing Applications

```
1  FROM nodesource/node:4
2
3  ADD https://github.com/Yelp/dumb-init/releases/download/v1.1.1/dumb-init_1.1.1_amd64 /usr/local/bin/dumb-init
4  RUN chmod +x /usr/local/bin/dumb-init
5
6  # ...
7
8  CMD ["dumb-init", "node", "index.js"]
9
```

dumb-init for PID 1

https://github.com/Yelp/dumb-init

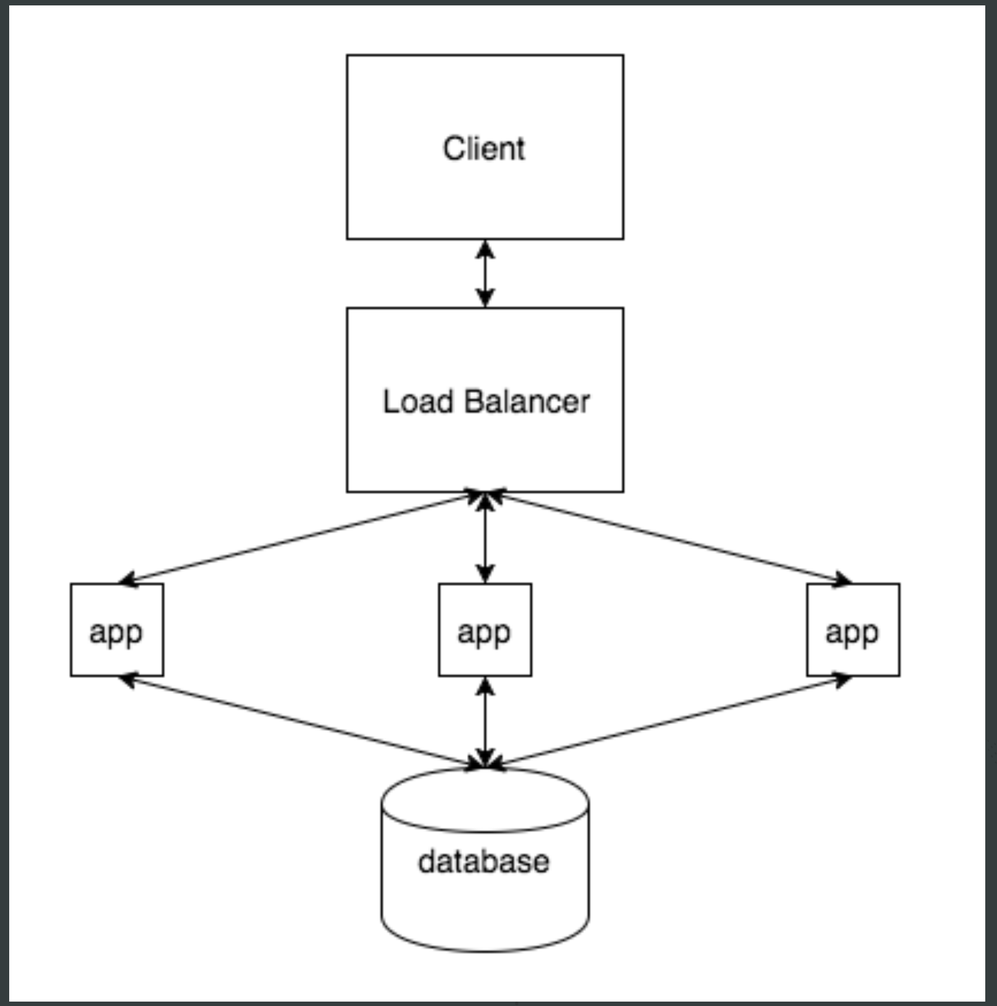
What should go into your container?

- Your application and it’s essential dependencies
- dumb-init
- Nothing else!

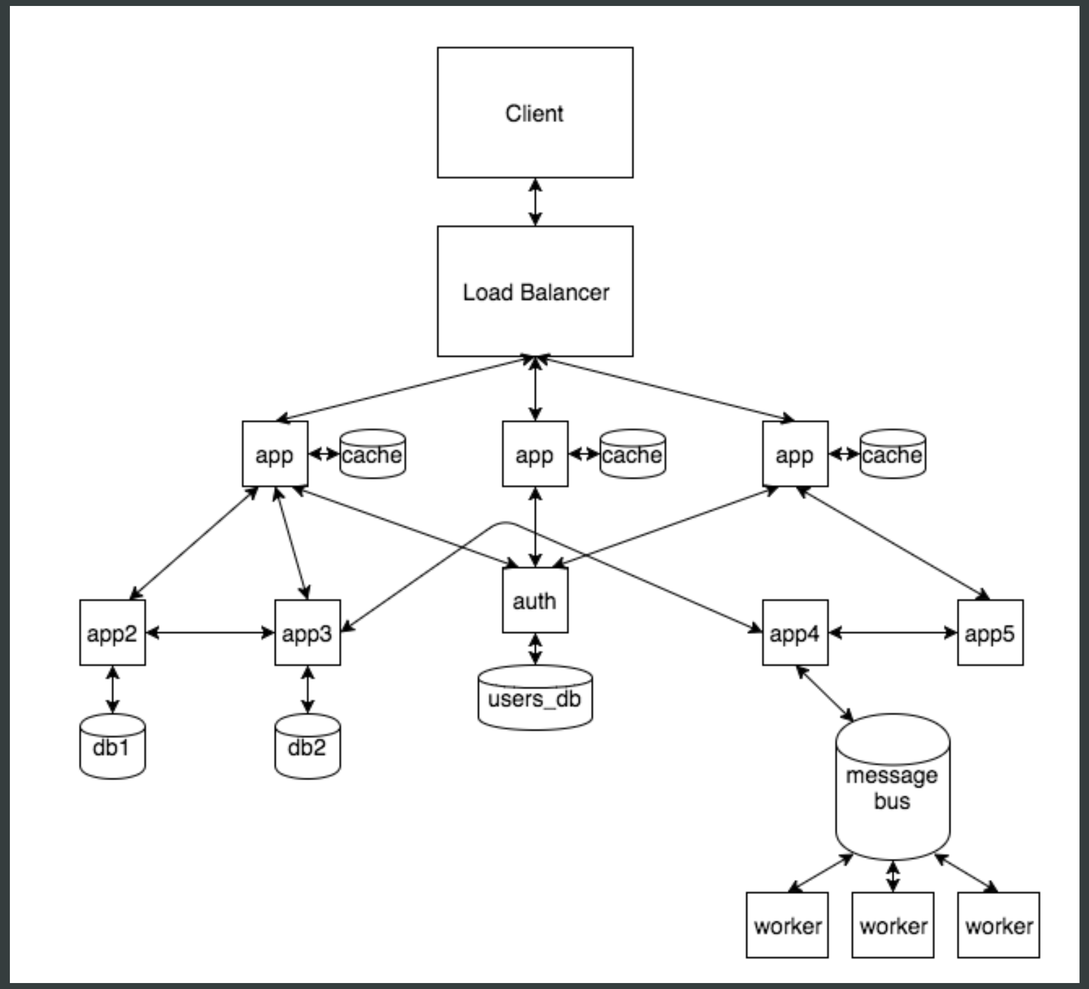
If your architecture looks like this:

Congrats! You're basically done.

Manage Docker applications with Ansible or ECS/ Docker Swarm



Dockerizing Applications



If it looks like this…

We need some orchestration help

Container Orchestration

- Schedule containers to physical machines
- Restart containers if they stop
- Provide private container network
- Scale up and down
- Service discovery

Quick recap

- Docker ideal distribution mechanism for Node.js applications
- Leverage best-practices when building Docker images
- Kubernetes provides efficient orchestration, management, and scaling of containerized Node.js Services

Q/A

#fococoders

Stay in touch!

Nathan White

nw@nwhite.net

@_nw_

#fococoders