Containers

Docker and 12 Factor Apps
Who we are

Luk Burchard
Computer Science Student
Software Engineer @ Loodse

twitter.com/lukburchard
github.com/realfake

Matthias Loibl
Computer Science Student
Software Engineer @ Loodse

twitter.com/metalmatze
github.com/metalmatze
Who are you?

- Developer?
- SysAdmin/Ops?
- Data Science?
- ...

Who are you?

- Who knows about Docker?
- Who knows about Kubernetes?
- Who uses Docker for Development?
- Who uses Docker in Production?

Who tried to use Docker, but couldn't do it?
Deployment

How do you deploy your apps?
Deployment

How do you deploy your apps?

Do you like SSH?
Deployment

How do you deploy your apps?

Do you like SSH?
Do you like SSH on 5 Servers?
Deployment

How do you deploy your apps?

Do you like SSH?
Do you like SSH on 5 Servers?
Do you like SSH on 100 Servers?
The Challenge

- **Static website**: nginx 1.5 + modsecurity + openasl + bootstrap 2
- **Background workers**: Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs
- **User DB**: postgresql + pgv6 + v6
- **Queue**: Redis + redis-sentinel
- **Analytics DB**: hadoop + hive + thrift + OpenJDK
- **Web frontend**: Ruby + Rails + sass + Unicorn
- **API endpoint**: Python 2.7 + Flask + pyredis + celery + psycopg + postgresql client

**Multiplicity of Stacks**

- Development VM
- QA server
- Customer Data Center
- Public Cloud
- Production Cluster
- Disaster recovery
- Contributor’s laptop
- Production Servers

**Can we migrate smoothly and quickly?**

**Do services and apps interact appropriately?**
### The Matrix from Hell

|--------------------|----------------|-----------|--------------------|----------------|--------------|----------------------|------------------|
Cargo Transport Pre-1960

- Multiplicity of Goods
- Multiplicity of methods for transporting/storing
- Do I worry about how goods interact (e.g., coffee beans next to spices)?
- Can I transport quickly and smoothly (e.g., from boat to train to truck)?
<table>
<thead>
<tr>
<th><img src="image" alt="Barrels" /></th>
<th><img src="image" alt="Cargo" /></th>
<th><img src="image" alt="Car" /></th>
<th><img src="image" alt="Barrel" /></th>
<th><img src="image" alt="Piano" /></th>
<th><img src="image" alt="Gold" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Train" /></td>
<td><img src="image" alt="Cargo" /></td>
<td><img src="image" alt="Loader" /></td>
<td><img src="image" alt="Crane" /></td>
<td><img src="image" alt="Ship" /></td>
<td><img src="image" alt="Dock" /></td>
</tr>
</tbody>
</table>
Solution: Shipping Container

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another.

Do I worry about how goods interact (e.g., coffee beans next to spices)?

Can I transport quickly and smoothly (e.g., from boat to train to truck)?
Docker: Container for shipping Software

- Static website
- User DB
- Web frontend
- Queue
- Analytics DB

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container...

...that can be manipulated using standard operations and run consistently on virtually any hardware platform.

- Development VM
- QA server
- Customer Data Center
- Public Cloud
- Production Cluster
- Contributor's laptop

Do services and apps interact appropriately?

Can I migrate smoothly and quickly?
### Eliminate the Matrix from Hell

<table>
<thead>
<tr>
<th></th>
<th>Development VM</th>
<th>QA Server</th>
<th>Single Prod Server</th>
<th>Onsite Cluster</th>
<th>Public Cloud</th>
<th>Contributor’s laptop</th>
<th>Customer Servers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static website</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Web frontend</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Background workers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>User DB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Analytics DB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Queue</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is a Container?
chroot

- chroot = change root
- Extract a filesystem to /mnt
- Change the root to /mnt
  - Uses the same (Linux) Kernel as before

Installing or repairing a Linux System with chroot
cgroups & namespaces

**cgroups**

limit & isolate the resource usage

Example:
Kill process using more than 256MB memory

**namespaces**

isolate and virtualize system resources of a collection of processes

- Mount
- Process ID
- Network
- User ID
- cgroups
LXC (Linux Containers)

- Operating-system-level virtualization
- Run multiple isolated Linux systems on a single Linux kernel
- Combines `cgroups` and `namespaces` to run Linux Containers
Containers vs. VMs

Containers are isolated, but share OS and, where appropriate, bins/libraries.
Docker
What is this Docker?

- Written in Go
- Released on March 13th, 2013
- Client-Server:
  - Docker Engine (daemon)
  - Docker Client, CLI
- Ready for production use
- Used LCX to run Containers
  - Uses cgroup, namespaces and OverlayFS
- Use their own libcontainer implementation
What does Docker provide?

- Run in the same environment
- Run in a lightweight environment
- Run in a sandboxed environment
- Pull images with all its dependencies
OCI (Open Container Initiative)

- Standard for container formats and runtimes
  - Standardizes how images are unpacked on the filesystem
  - Standardizes how containers are run from images
- Under auspices of the Linux Foundation
- docker, rkt and others now run the same specification
- runc is an OCI implementation
Install Docker

- Docker on Linux, ask your package manager
- Docker for Mac
- Docker for Windows

Run $ docker version

Use our GCP Codes
Docker Group on Linux

# Add the Docker group
$ sudo groupadd docker

# Add yourself to the group
$ sudo gpasswd -a $USER docker

# Restart the Docker daemon
$ sudo systemctl restart docker

$ docker ps # run docker without sudo
# Excerpt of most important docker commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>docker build</code></td>
<td>Build an image from a Dockerfile</td>
</tr>
<tr>
<td><code>docker exec</code></td>
<td>Run a command in a running container</td>
</tr>
<tr>
<td><code>docker inspect</code></td>
<td>Return low-level information on Docker objects</td>
</tr>
<tr>
<td><code>docker kill</code></td>
<td>Kill one or more running containers</td>
</tr>
<tr>
<td><code>docker logs</code></td>
<td>Fetch the logs of a container</td>
</tr>
<tr>
<td><code>docker pull</code></td>
<td>Pull an image or a repository from a registry</td>
</tr>
<tr>
<td><code>docker push</code></td>
<td>Push an image or a repository to a registry</td>
</tr>
<tr>
<td><code>docker rm</code></td>
<td>Remove one or more images</td>
</tr>
<tr>
<td><code>docker run</code></td>
<td>Run a command in a new container</td>
</tr>
<tr>
<td><code>docker stop</code></td>
<td>Stop one or more running containers</td>
</tr>
<tr>
<td><code>docker tag</code></td>
<td>Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE</td>
</tr>
</tbody>
</table>
OverlayFS:
each layer ‘overlays’
the lower layer

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>91e54dfb1179</td>
<td>0 B</td>
</tr>
<tr>
<td>d74508fb6632</td>
<td>1.895 KB</td>
</tr>
<tr>
<td>c22013c84729</td>
<td>194.5 KB</td>
</tr>
<tr>
<td>d3a1f33e8a5a</td>
<td>188.1 MB</td>
</tr>
</tbody>
</table>

Image
- CMD `"/bin/bash"`
- `mkdir -p /run/systemd && echo ...`
- `sed -i 's/^\s*deb.*universe\..`
- `ADD file:280a445783f309c..`
The Container
(a running program)

The Image
(a blueprint for a container)
Container Registries

- **hub.docker.com**
  - Docker's official Registry
- **quay.io**
  - Public Registry by CoreOS
- **cloud.google.com/container-registry**
  - Shorter: `gcr.io/google_containers/pause-amd64`
  - Often used in combination with Kubernetes
- Host your own private Registry
Container Registry Commands

Use docker CLI to authenticate

$ docker login
$ docker logout

# Login to a private registry
$ docker login registry.example.com
Container Architecture

Client
- docker build
- docker pull
- docker run

DOCKER_HOST
- Docker daemon

Containers

Images

Registry
- NGINX
Run a Container

$ docker run alpine echo 'hello world'
$ docker ps

What did just happen?

- Pulled alpine image from the registry
- Created a new container
- Allocated a filesystem and mounts a read-write layer
- Allocated a network/bridge interface
- Sets up an IP address
- Executes a process that you specify (/bin/bash)
- Captures and provides application output
Run a long-lived Container

$ docker run --name hw alpine /bin/sh -c "while true; do echo hello world; sleep 1; done"
$ docker ps
$ docker logs (-f) hw

Ctrl+C the container
$ docker ps
$ docker ps -a
Run nginx in a Container

# Ports
$ docker run --rm -p 8080:80 nginx
$ docker run --rm -p 8080:80 nginx:1.13
$ docker run -d --name nginx -p 8080:80 nginx

# Volumes
$ docker run --rm -p 8080:80 -v /tmp/nginx:/usr/share/nginx/html:ro nginx
Dockerfile

- Build steps to create an image
- Invoke with "\$docker build ."
- Output is and image
- Cache image layers

FROM alpine:latest
ADD hostsrc /containerdest
WORKDIR /pwdofcontainerstart
CMD ./main
Docker: "don't"s

- Don't store data in containers
  - All data will be lost
- Don't create large images
  - Use alpine
- Don't use only the latest tag
  - How would you rollback?
- Don't run more than one process in a single container
12-Factor Apps

https://12factor.net
12 Factors

1. Codebase
   Use something like **git**

2. Dependencies
   Use **dep, pip, gem, npm** etc...

3. Configuration
   Use **EnvVars**, not config files

4. Backing services
   Independent of depended services
   Example: DB, MySQL or RDS

5. Build, release, run
   Build a immutable release, use CI/CD

6. Processes
   Apps are just a stateless process
   *(Containers ;-)*)
12 Factors

7. Port binding
   Expose Apps via Ports
   Example: HTTP:80, Postgres:5432

8. Concurrency
   Keep horizontal scaling in mind

9. Disposability
   Fast start time, terminate on SIGTERM
   Container send SIGTERM ;-)  

10. Dev/prod parity
    Deploy often, DevOps, run same containers in dev

11. Logs
    Streams, not files. Write to stdout

12. Admin processes
    Run admin tasks as one-off processes
    Example: Run script to migrate DB
I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes
12 Factor - Implications

- Portability
- Deployability
- Scalability
- Immutability
Let’s begin!

Shall we?

unzip cotbat.zip
github.com/realfake/cotlaader
github.com/realfake/cotbat
You can host your own!
It's just a docker container with BasicAuth

# Run a registry locally
$ docker run -d -p 5000:5000 --name registry registry:2

# Use your images
$ docker tag project:1.2.3 registry.example.com/project:1.2.3
$ docker push registry.example.com/project:1.2.3