DOCKER

Build, Ship and Run
Any App, Anywhere

Docker - An open platform for distributed applications
for developers and sysadmins.

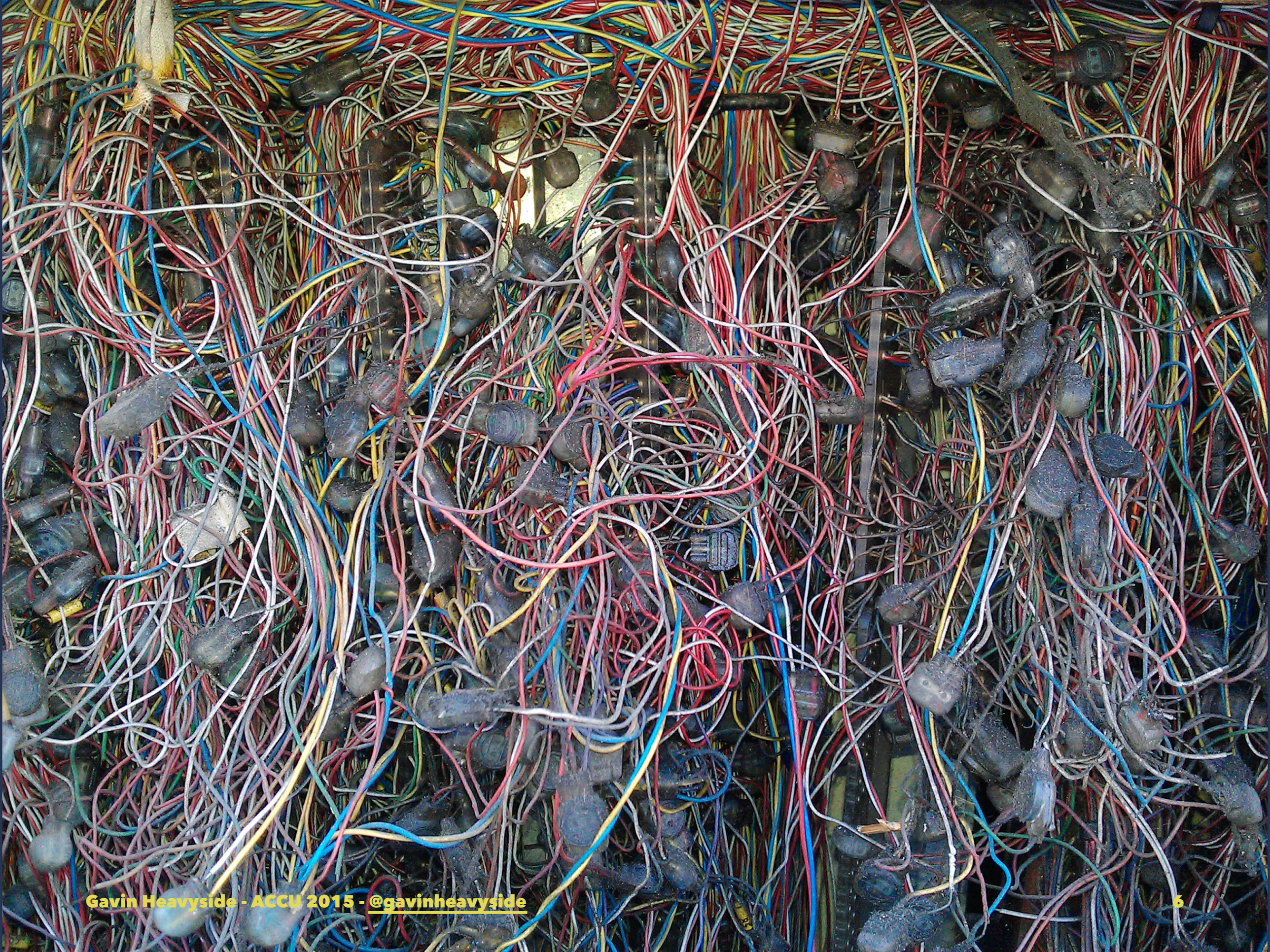
Gavin Heavyside
@gavinheavyside

At the #AWSSummit almost 10 minutes before I heard someone mention docker. And it wasn't me.

Francis Pindar
@radnip

Follow

@gavinheavyside -10mins for me. I didn't even make it to the summit before someone mentioned it on the tube coming in :)



mydrive solutions

Home | The MyDrive Solution | Newsroom | MyDrive Guides | Contact

- End-To-End Telematics
- In-Car Telematics Devices
- One Second Data
- Contextualised Mapping
- Ability Benchmarking
- Road Type & Setting
- Driving Style
- Driver Engagement

End-To End Telematics For Insurance Companies

MyDrive offers an end to end insurance capability. We are agnostic with respect to data collection device, and deliver highly accurate and granular driver profiles to the insurer.

MyDrive has successfully implemented projects for many insurers around the world, we understand the complexity of telematics based policy as a whole, the importance of having access to granular data and the consequent risk calculation presented to drivers.

DevOps & Infrastructure As Code

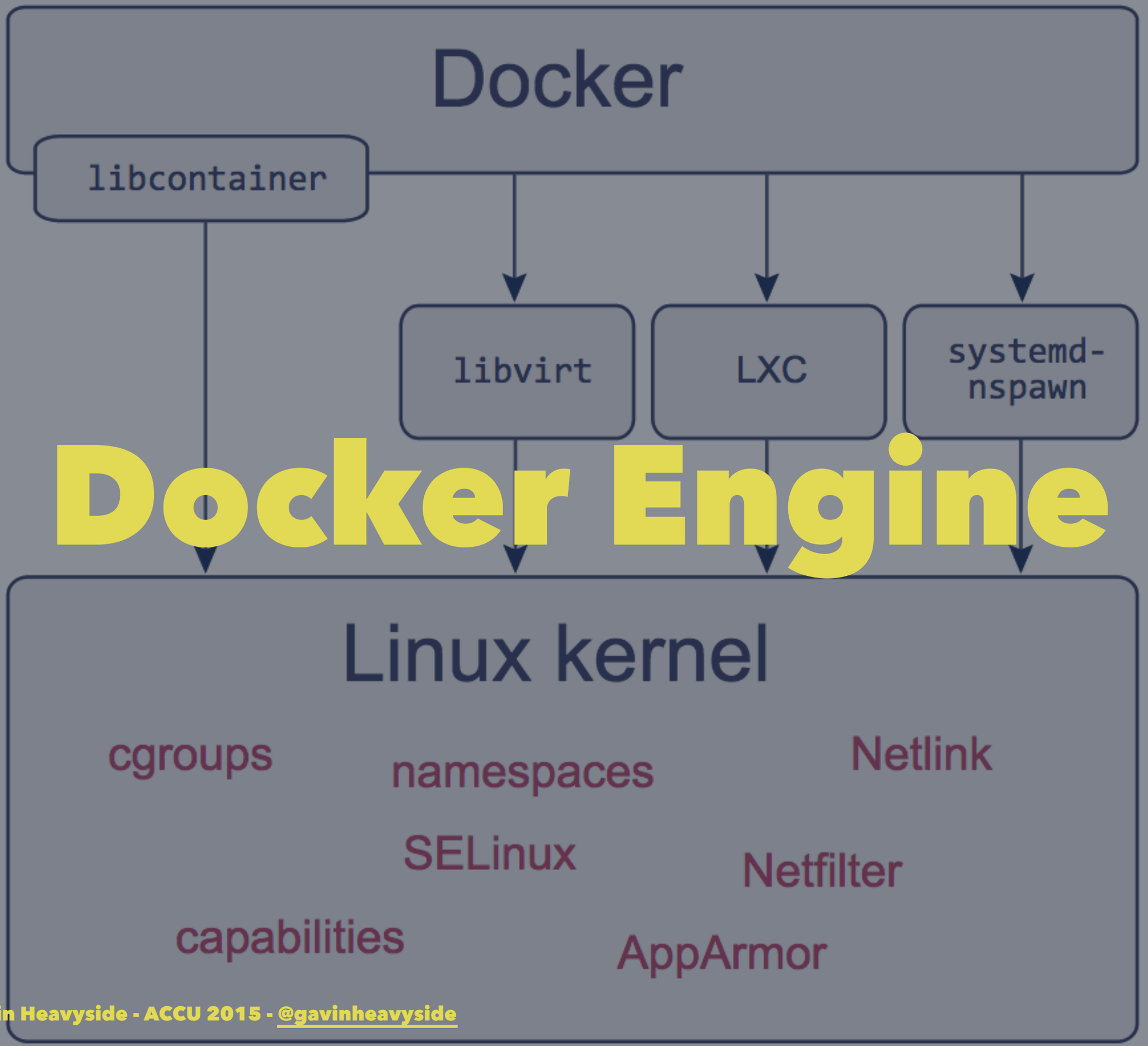
Gavin Heavyside - ACCU 2012 - 27 April 2012
gavin@heavyside.co.uk @gavinheavyside


Goals
2007 PETER FUCHS


ship it

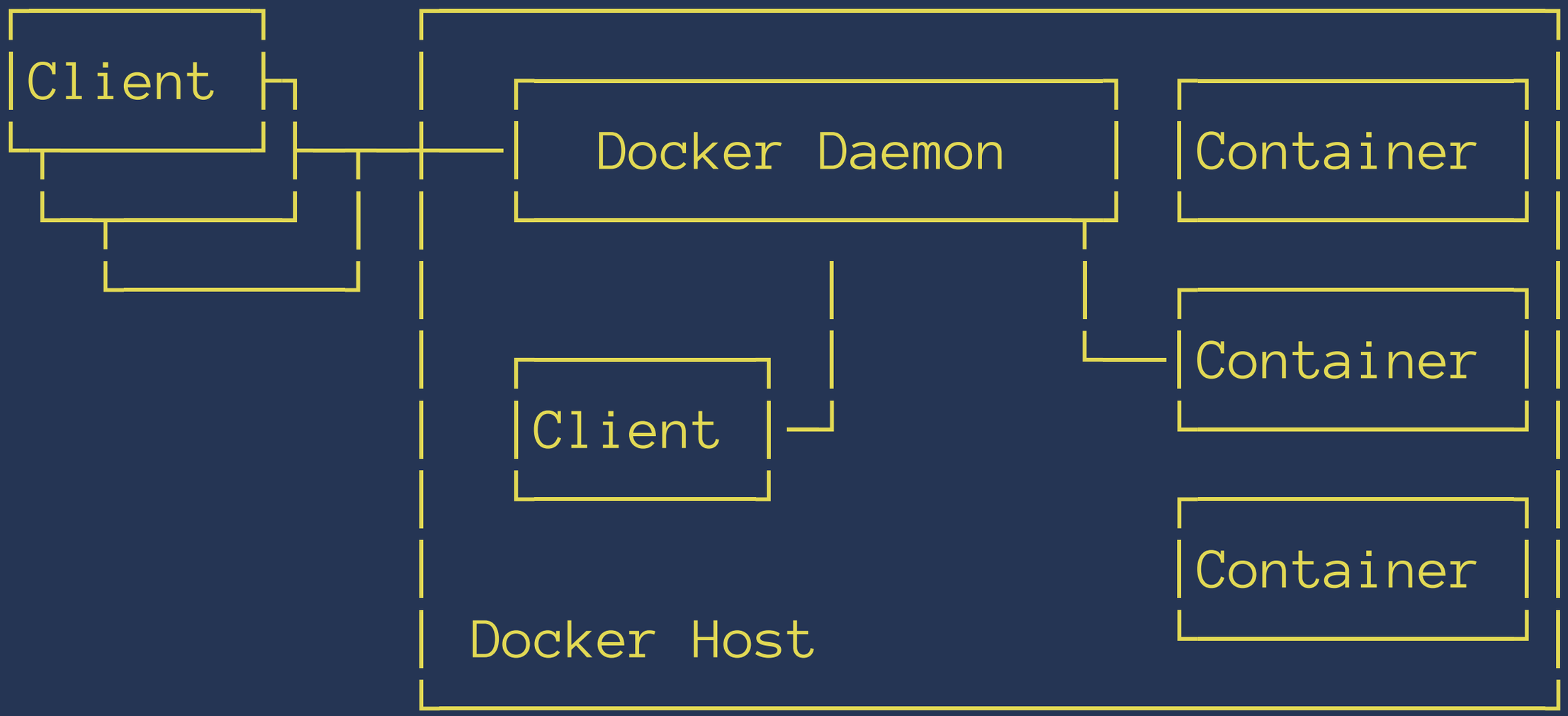
Docker Components

- Engine
- Hub
- Compose
- Swarm
- Machine



Docker Engine

Docker Client-Server



Docker Client

attach **build** commit cp create diff events
exec export history **images** import info
inspect kill load login logout **logs** port
pause **ps pull push** rename restart **rm** rmi
run save search start stats stop **tag** top
unpause version wait

Running a Docker Container

```
docker run -i -t ubuntu /bin/bash
```

Running a Docker Container

```
docker run -i -t ubuntu /bin/bash
```

- Pulls the ubuntu image

Running a Docker Container

```
docker run -i -t ubuntu /bin/bash
```

- Creates a new container

Running a Docker Container

```
docker run -i -t ubuntu /bin/bash
```

- Allocates a filesystem and mounts a R/W layer

Running a Docker Container

```
docker run -i -t ubuntu /bin/bash
```

- Allocates a network / bridge interface

Running a Docker Container

```
docker run -i -t ubuntu /bin/bash
```

- Sets up an IP address

Running a Docker Container

```
docker run -i -t ubuntu /bin/bash
```

- Executes your process

Running a Docker Container

```
docker run -i -t ubuntu /bin/bash
```

- Captures and provides application output

How it works (Short Version)

- Written in Go
- Takes advantange of Linux kernel features
    - Namespaces
    - Control Groups (cgroups)
- Union File System
- libcontainer

Namespaces

- Separation of groups of processes
  - Can't 'see' resources in other groups
- PID namespace, network, mount, IPC, and more

Namespaces

- Docker creates a set of namespaces for each container.
- Isolation layer
  - each aspect of a container runs in own namespace
  - does not have access outside it
- some used by Docker: pid, net, ipc, mnt, uts

Control Groups (cgroups)

- limit, account, and isolate resources used by a collection of processes
- CPU, memory, disk I/O, network, etc.
- The basis of many container projects
  - Docker, LXC, lmctfy, Mesos, and more

cgroups

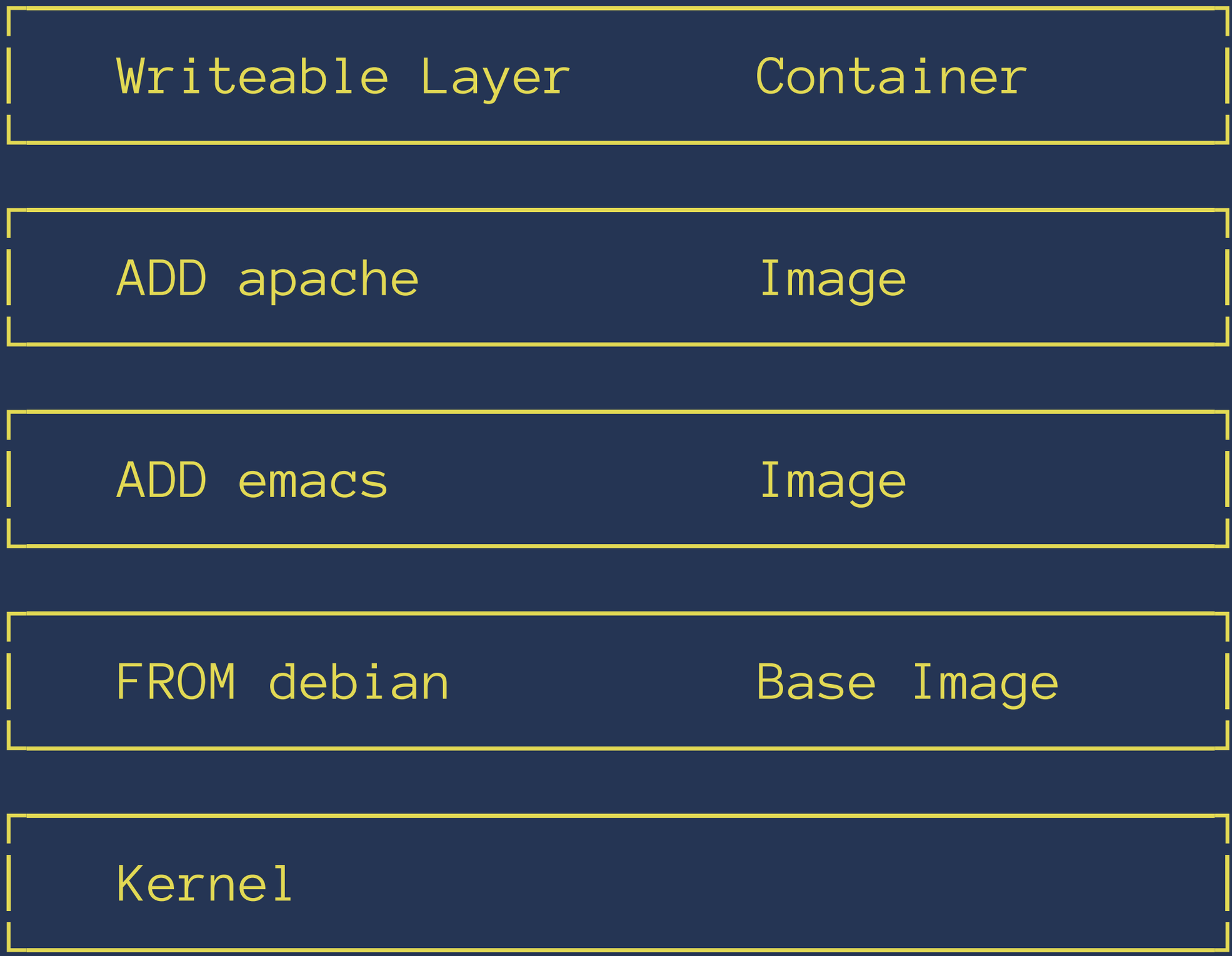
- allow Docker to share available hardware resources to containers
- set up limits and constraints, if required

Setting Resource Limits

```
docker run -m 256m --cpu-shares 512 yourapp
```

Union File Systems

- Layer files and dirs
- Can be from different file systems
- Present as a single filesystem
- Can have RO and RW layers


Writeable Layer
Container
ADD apache
Image
ADD emacs
Image
FROM debian
Base Image
Kernel

Union File Systems

- UnionFS
- aufs
- btrfs
- and more...

libcontainer

- https://github.com/docker/libcontainer
- Default supported container format
- Creates containers with namespaces, cgroups, capabilities, and filesystem access controls
- Manages lifecycle of the container

Other container technologies

- Solaris Zones
- lmctfy
- rkt
- LXC
- BSD Jails

Building a Container

- Write a Dockerfile
- build the image with `docker build`
- run it with `docker run`
- Share by pushing to a registry

The Dockerfile

- Plain text file
- Series of directives
    - add files
    - expose ports
    - execute commands
    - configure runtime

The Dockerfile

```
FROM busybox

RUN mkdir -p /usr/local/bin

COPY ./hello /usr/local/bin/hello

CMD ["/usr/local/bin/hello"]
```

FROM

```
FROM ubuntu:14.04
```

- Base image (& tag) to start building from

MAINTAINER

```
MAINTAINER Peter V "venkman@1984.com"
```

- Who ya gonna call?

RUN

```
RUN apt-get update && apt-get -y upgrade
```

- Execute command in a new layer and commit
- defaults to using `/bin/sh`
- `RUN ["/bin/bash", "-c", "uptime"]`

CMD

```
CMD ["executable","param1","param2"]
```

- Default command to execute

EXPOSE

```
RUN apt-get install nginx
EXPOSE 80
```

- Ports for the continer to listen on
- Used for interconnecting linked containers
- Doesn't automatically map to the host

ENV

```
ENV FOO=bar
```

- Set an environment variable in the container

ADD / COPY

```
COPY /my/src /opt/container/src
```

- Copy content to the container filesystem

USER

```
USER nginx
```

- Set the UID for the image and any following directives

WORKDIR

```
WORKDIR /path/to/workdir
```

- set the working dir for the image and any following directives

ONBUILD

`ONBUILD bin/rake db:assets:precompile`

- Trigger instruction to run when image is used as a base for another build
- Only for direct child of this image
- Runs after `FROM` directive in child build

22 lines (12 sloc) | 0.593 kb

```
1  FROM java:8-jre
2
3  RUN apt-key adv --keyserver pool.sks-keyservers.net --recv-keys 46095ACC8548582C1A2699A9D27D6
4
5  ENV ELASTICSEARCH_VERSION 1.4.4
6
7  RUN echo "deb http://packages.elasticsearch.org/elasticsearch/${ELASTICSEARCH_VERSION%.*}/deb
8
9  RUN apt-get update \
10         && apt-get install elasticsearch=$ELASTICSEARCH_VERSION \
11         && rm -rf /var/lib/apt/lists/*
12
13 ENV PATH /usr/share/elasticsearch/bin:$PATH
14 COPY config /usr/share/elasticsearch/config
15
16 VOLUME /usr/share/elasticsearch/data
17
18 EXPOSE 9200 9300
19
20 CMD ["elasticsearch"]
21
```

Dockerfile Tips

- Choose your base image wisely
- Do the expensive work first
- Take advantage of caching and layering
- Use `.dockerignore`

docker What is Docker? Use Cases Try It! Browse Install & Docs Log In Sign Up

Repositories (125)

Users (0)

Organizations (1)

library

Show: All Sort by: Relevance Results: Size

ubuntu 3 hours ago
Official Ubuntu base image
1590 3529215

node an hour ago
Node.js is a JavaScript-based platform for server-side and networking applications.
654 560977

debian 2 hours ago
(Semi) Official Debian base image.
417 265777

postgres an hour ago
The PostgreSQL object-relational database system provides reliability and data integrity.
658 1013860

mysql 12 hours ago
MySQL is a widely used, open-source relational database management system (RDBMS).
617 3911902

Pulling Images From a Registry

```
docker pull elasticsearch

docker pull private.globocorp.com/elasticsearch
```

Tags

```
docker pull nginx:latest
docker pull nginx:1.7.11
            _____ ______
              |     |
            repo    tag
```

Running your own registry

- registry (Docker < 1.6)
- distribution (Docker 1.6+)
- dogestry

Storage

- Transient
- Local
- Persistent (portable)
- Probably the hardest thing to get right at the moment

Volumes

VOLUME directive

- Indicates the container wants to use external storagee

`--volumes-from`

- mount VOLUME paths from container A in container B

Persistent Storage

```
docker run -v /local/path:/container/
path elasticsearch
```

- local path on filesystem is mounted in container
- persists after the container exits
- Portability across machines in a cluster is still a hard problem

Linking Containers

```
docker run -d -p 80:80 --name app1 app1:latest
```

```
docker run --link app1:app1 app2:latest
```

- The code running in the app2 container can now talk to app1 on port 80, using the URI `http://app1:80`
- Not limited to HTTP!

Tailoring your app for Docker

- Docker works best when containers have a single responsibility
- not *necessarily* a single process
- Some design choices can make you life easier in production

The 12-Factor App

- http://12factor.net
- Codebase
- **Dependencies**
- **Config**
- Backing Services
- Build;Release;Run
- Processes

The 12-Factor App

- **Port Binding**
- Concurrency
- Disposability
- **Dev/Prod Parity**
- **Logs**
- Admin Processes

12 Factor - Dependencies

- http://12factor.net/dependencies
- Explicitly declare and isolate dependencies
- No implicit deps "leak in"
- Full and explicit dependency spec is applied in all envs, dev and prod

12 Factor - Config

- http://12factor.net/config
- Store config in the environment
- Config is *everything* that can change between deploys; dev, test, and production

12 Factor - Port Binding

- http://12factor.net/port-binding
- App should be entirely self-contained
- Expose services via port binding
- Not just for HTTP
- Remember health check endpoints

12 Factor - Dev/Prod Parity

- http://12factor.net/dev-prod-parity
- Keep development, staging, and production as similar as possible
- Fewer moving parts means fewer people, skills, less time to push to production

12 Factor - Logs

- http://12factor.net/logs
- Treat logs as event streams
- Log to stdout
- Collect, rotate, and centralise logs outside the app

Computation Containers

$$P\{Q\}R$$

- A program *Q*, with preconditions *P*, will produce output *R*
- *P* and *Q* can change when we move between environments
- Docker containers can form a complete statement of the runtime environment *P*, and the program to run *Q*

Toolchain in a container

```
$ docker run --rm -v `pwd`:/src \
  -w /src golang:1.4 go build hello.go
```

Toolchain in a container

```
$ docker run --rm -v `pwd`:/src \
  -w /src golang:1.4 go build hello.go
```

BUT - I'm on OS X and my boot2docker host is running Linux

Toolchain in a container

```
$ docker run --rm -v `pwd`:/src \
  -w /src golang:1.4 go build
$ ./hello
exec format error: hello
$ file hello
hello: ELF 64-bit LSB executable, ...
```

Toolchain in a container

```
$ docker run --rm -v src:/src \
  -e "GOOS=darwin" \
  -w /src golang:1.4-cross \
  go build
$ file hello
hello: Mach-O 64-bit executable x86_64
$ ./hello
Hello, World
```

Choosing a base image

- Enough foundation, but not too much
- Security and hardening, provenance
- Reuseability
- Compatibility

The PID 1 Reaping Problem

- Unix processes are modelled like a tree
- PID 1 is the top-most process
- Typically this is an init process


CAUTION!
ZOMBIES!
AHEAD!!!

What to do?

- Nothing
- Specify a different init
  - runit
  - supervisord
  - phusion/baseimage-docker
  - other init process

Minimalist Host OS

Features of the New Minimal OSes

- Small and lightweight
  - Specialised, not general purpose
  - Quick to install and boot
  - Smaller surface area to harden and defend
  - Applications deployed as containers

Features of the New Minimal OSes

- Read-only system files
- Transactional platform updates
- Backup, rollback
- Delta patches
- Signatures and fingerprints

Examples of Minimalist OSes

- Snappy Ubuntu Core
- Project Atomic
- CoreOS
  - Docker compatible, pushing own containers
- RancherOS

CoreOS

- Etcd
- Rkt
- Fleet
- Flannel

WEDNESDAY, APRIL 8, 2015

Microsoft Unveils New Container Technologies for the Next Generation Cloud

Docker on Windows

MIKE NEIL
General Manager, Windows Server

In today's cloud-first world, businesses increasingly rely on applications to fuel innovation and productivity. As the cloud evolves, containers are emerging as an attractive way for developers to quickly and efficiently build and deploy these applications at the speed of business. Offering developers and IT professionals the ability to deploy applications from a workstation to a server in mere seconds, containers are taking application development to a whole new level.

C:\windows\system32\cmd.exe

```
C:\Users\ahmetb>docker
Usage: docker [OPTIONS] COMMAND [arg...]

A self-sufficient runtime for linux containers.

Options:
  --api-enable-cors=false                      Enable CORS headers in the remote API
  -D, --debug=false                            Enable debug mode
  -d, --daemon=false                           Enable daemon mode
  -G, --group="docker"                         Group to assign the unix socket specified by -H when run
ning in daemon mode
                                                 use '' (the empty string) to disable setting of a grou
p
  -H, --host=[]                                The socket(s) to bind to in daemon mode or connect to in
 client mode, specified using one or more tcp://host:port, unix:///path/to/socket, fd://* or fd://socketfd.
  --tls=false                                  Use TLS; implied by --tls-verify flags
  --tlscacert="C:\\Users\\ahmetb\\.docker\\ca.pem"  Trust only remotes providing a certificate signed by the
 CA given here
  --tlscert="C:\\Users\\ahmetb\\.docker\\cert.pem"  Path to TLS certificate file
  --tlskey="C:\\Users\\ahmetb\\.docker\\key.pem"    Path to TLS key file
  --tlsverify=false                            Use TLS and verify the remote (daemon: verify client, cl
ient: verify daemon)
  -v, --version=false                          Print version information and quit

Commands:
    attach    Attach to a running container
    build     Build an image from a Dockerfile
    commit    Create a new image from a container's changes
    cp        Copy files/folders from a container's filesystem to the host path
    create    Create a new container
    diff      Inspect changes on a container's filesystem
    events    Get real time events from the server
    exec      Run a command in a running container
    export    Stream the contents of a container as a tar archive
    history   Show the history of an image
    images    List images
    import    Create a new filesystem image from the contents of a tarball
    info      Display system-wide information
    inspect   Return low-level information on a container
```

Docker Client

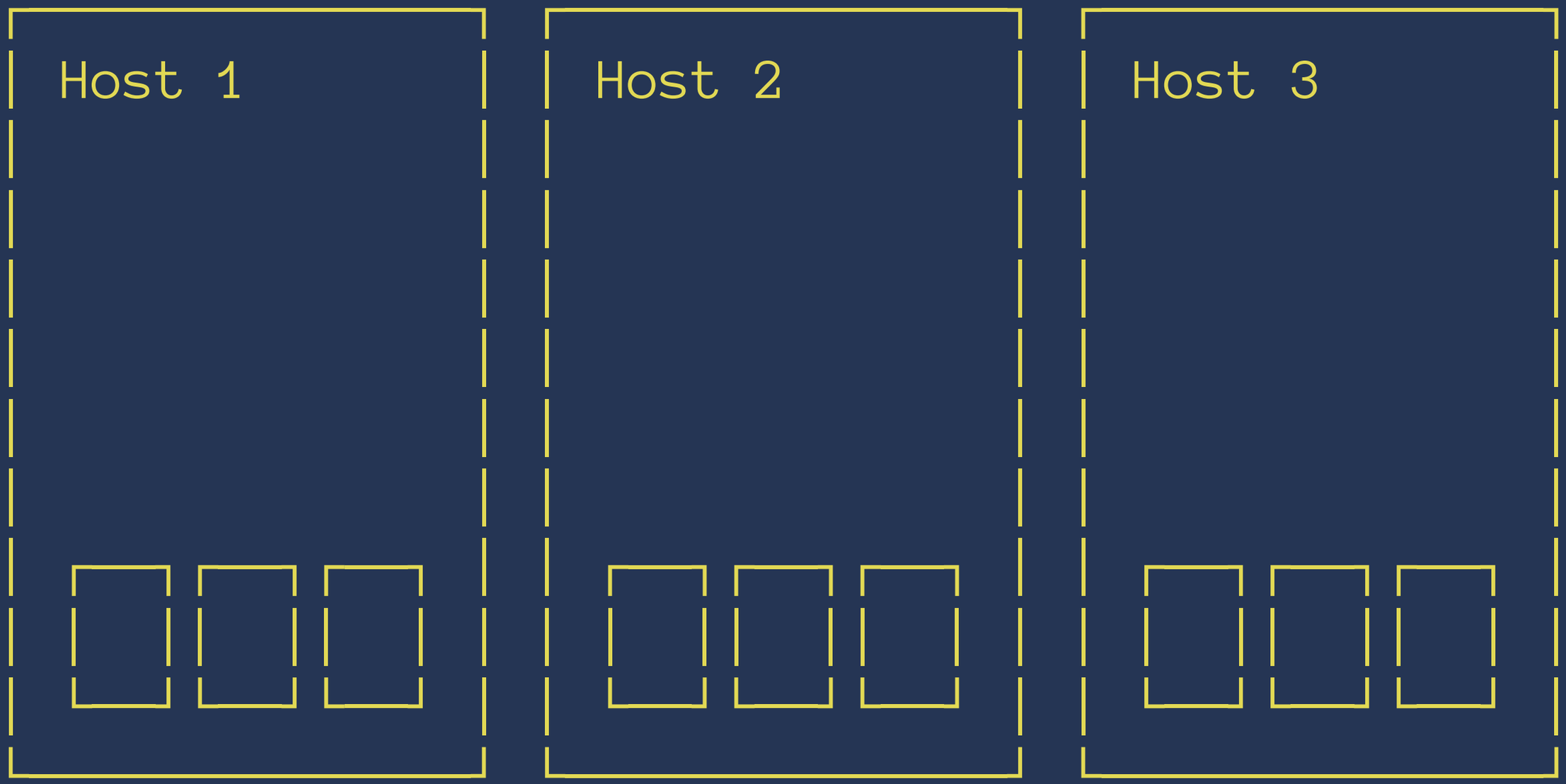
Windows Links

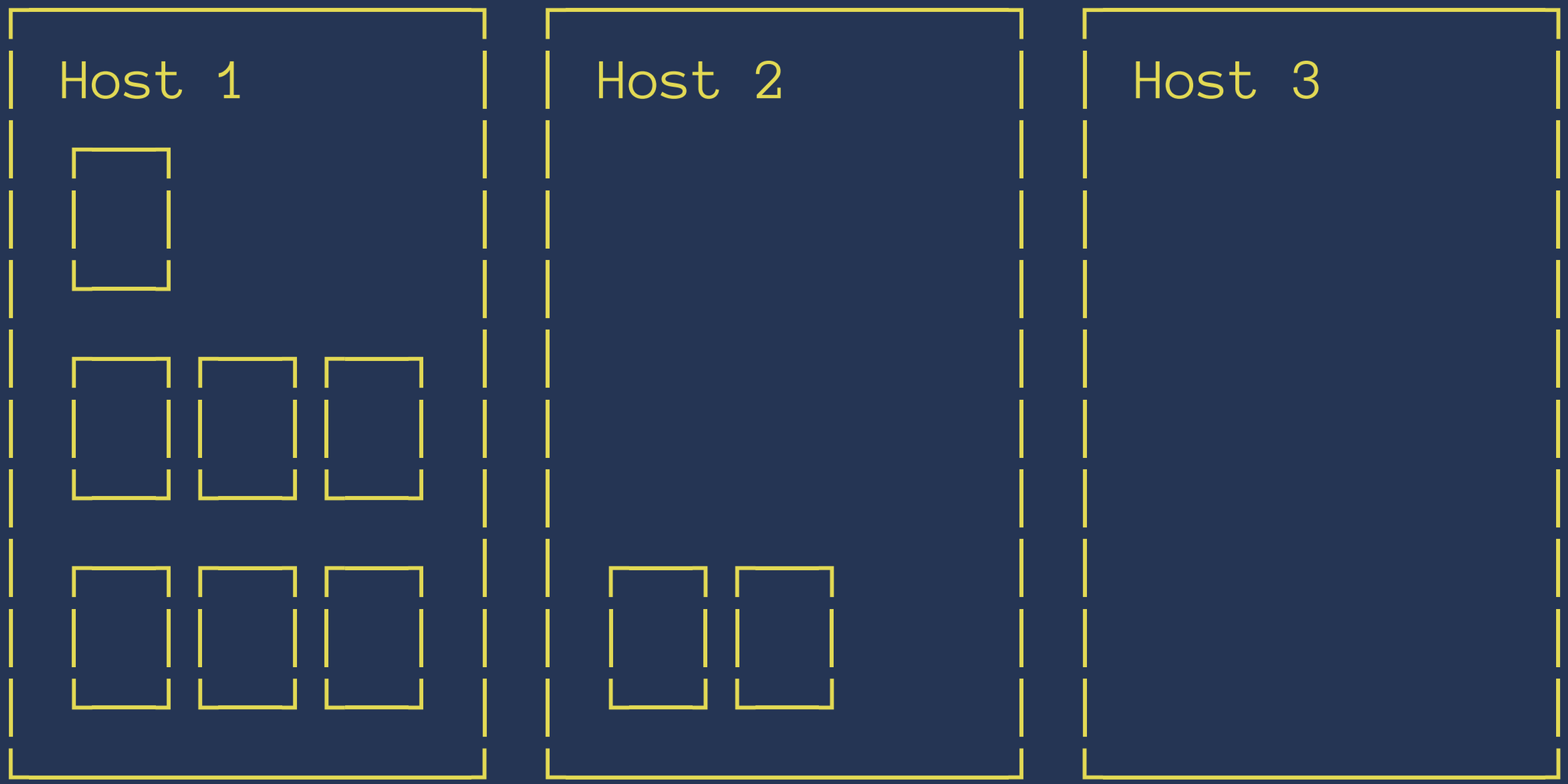
- http://azure.microsoft.com/blog/tag/docker/
- http://azure.microsoft.com/blog/2015/04/08/microsoft-unveils-new-container-technologies-for-the-next-generation-cloud/
- http://azure.microsoft.com/blog/2015/04/16/docker-client-for-windows-is-now-available/

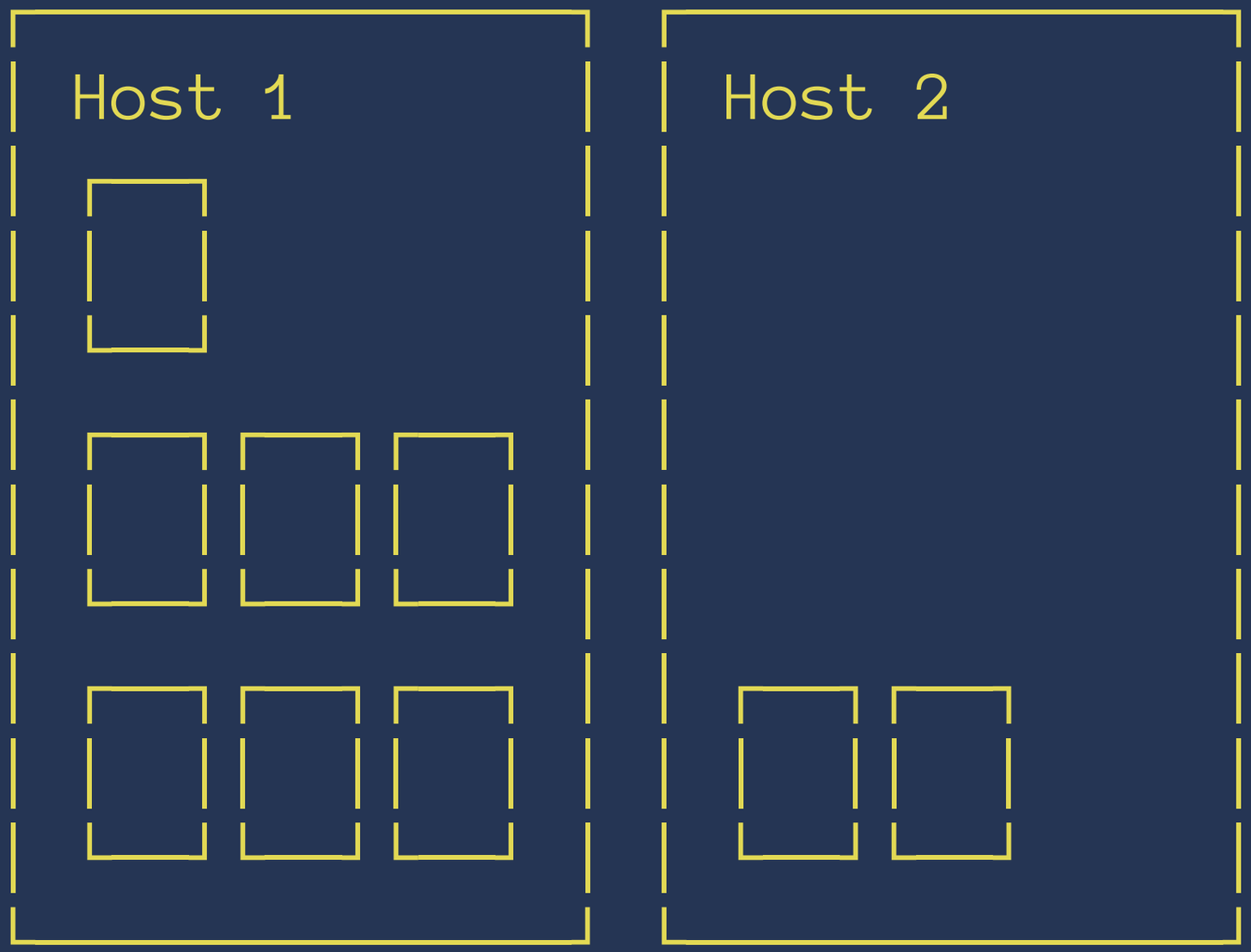
Cluster Management

Cattle, Not Pets

- Not snowflakes, either
- Care about the service, not the server
- Easier said than done

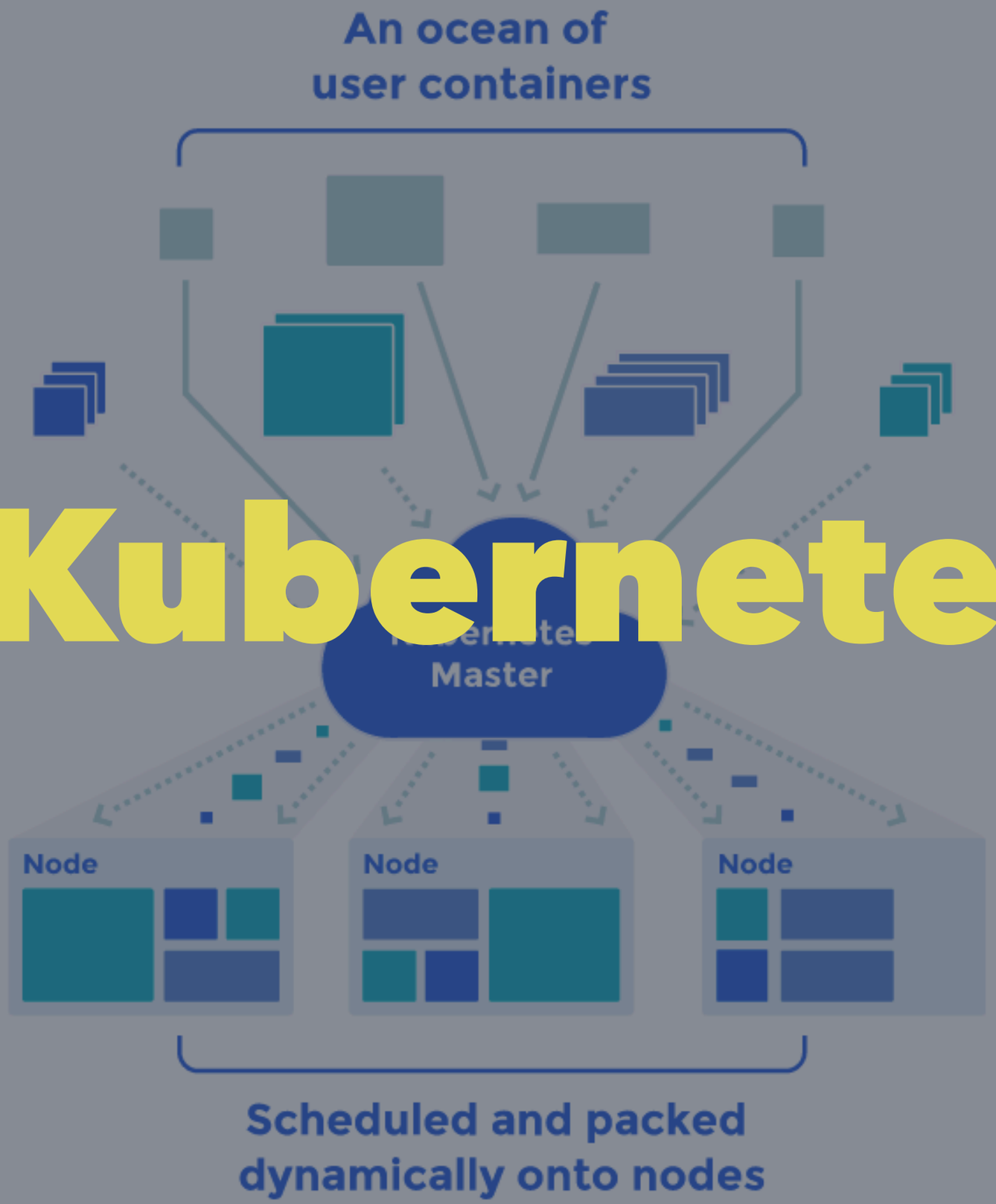

Host 1
Host 2
Host 3


Host 1
Host 2
Host 3


Host 1
Host 2

Cluster Management

- Kubernetes
- Docker Swarm
- CoreOS Fleet
- AWS ECS
- Google Container Service
- More


An ocean of
user containers
Master
Node
Node
Node
Scheduled and packed
dynamically onto nodes

Kubernetes

Kubernetes

- Abstract at the *service* level, not container
- Compose services from containers
- Dependencies
  - CPU, RAM, placement
  - Container start order
- services, load balancing

Hosted Kubernetes

- Google Container Engine (Alpha)
  - Hosted K8 on Google Cloud Platform
- Tectonic (Beta)
  - by CoreOS

AWS EC2 Container Service

- Hosted Docker orchestration on EC2 (GA)
- Multi-container dependencies
- Placement and scheduling
  - one-off
  - service
  - pluggable (e.g. Mesos)

Service Discovery

- How do your services talk to each other?
- How do they find each other in a dynamically allocated cluster?
- Docker container linking only works within a host (so far)

Service Discovery

- Message buses (e.g. rabbitMQ)
- DNS
- Service Discovery Tools
- Load balancing and health checking

Service Discovery Tools

- DNS
- SmartStack (nerve, synapse)
- Etcd (and SkyDNS)
- Consul
- More

Consul

- https://consul.io
- K/V, DNS interfaces, ACLs
- Services, health checks, load balancing
- serf gossip protocol, raft consensus algorithm
- distributed, highly available

Registrator

- https://github.com/gliderlabs/registrator
- Container watches Docker engine events, dynamically registers services with backends
- Etcd, Consul, SkyDNS support
- Automatically publish addresses and ports of services across your infrastructure

Logs

- Easier if containers log to stdout, saved on the host
- Can mount log dir as a volume in container if needed
- Consider running e.g. logstash on the host, archiving and centralising logs
- New syslog support in Docker 1.6

Monitoring

- Some dedicated tools appearing, hosted and open source
- Still an area with catching up to do
- Traditional tools can monitor the health of apps via exposed ports and endpoints

Wrapping Up

Image Credits

- minimalist room: https://www.flickr.com/photos/colinsite/14089317769
- cluster: https://www.flickr.com/photos/skiwalker79/3306092836
- wrapping: https://www.flickr.com/photos/georigami/14253603878
- zombies: https://www.flickr.com/photos/reana/3238910501

Image Credits

- goals: https://www.flickr.com/photos/peterfuchs/1239399915
- complexity: https://www.flickr.com/photos/bitterjug/7670055210
- volume: http://en.wikipedia.org/wiki/Up_to_eleven
- containers: https://www.flickr.com/photos/cseeman/11102312383