INTRODUCTION TO DOCKER

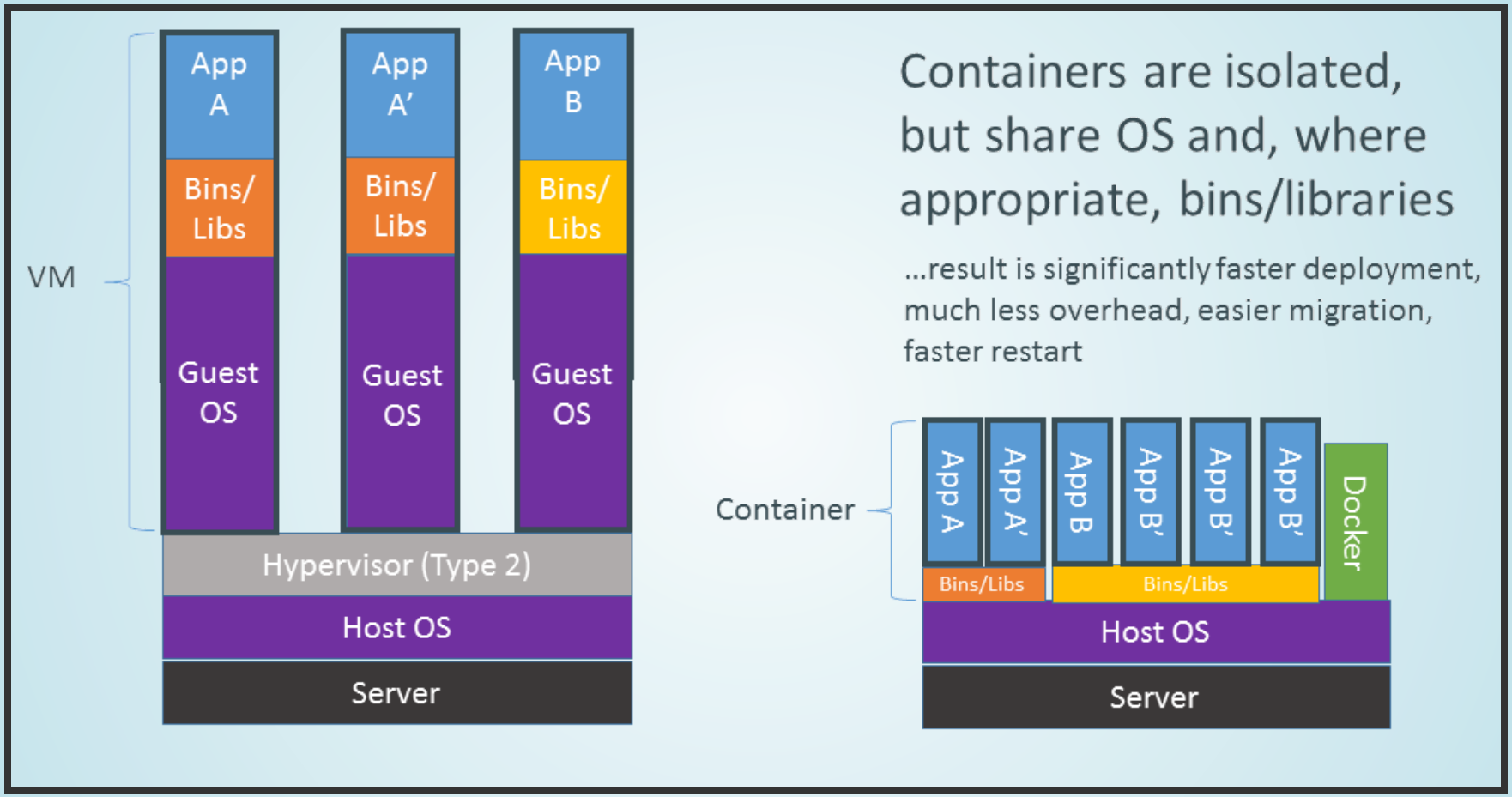
JOHANNES 'FISH' ZIEMKE

- twitter/github: @discordianfish
- Building Docker's infrastructure for central services
- docker.com
- hub.docker.com
- Former SoundCloud System and Infrastructure engineer
- Scaled infrastructure by 10x in 2 years

DOCKER?

- Docker 1.0 release 2h ago
- Open Source, writte in Go
- developed publicly; GitHub, Mailinglists
- Builds, packs and ships applications as lightweight containers
- Build once, run (almost) everywhere
- Linux 3.8+, OS X via transparent VM wrapper

CONTAINER VS VM



WHO CARES?

SOME DO

- Docker 0.1 release Spring 2013
- by dotCloud (now Docker Inc)
- Rewrite of code that powers dotCloud PaaS
- Since release
- >15000 Images on central registry
- >6000 Dockerfiles on GitHub
- >2000 external pull requests/contributions

WELL, OKAY..

but why should **I** care?

TEST THINGS?

- Start always with a clean slate
- Spawn up complete test infrastructures in seconds
- Run your tests against various versions of libraries and services

MAKE THINGS EASIER, HELP OR TEACH PEOPLE?

- Provide students with software environment
- Distribute complex setups as self-contained container for
- Bioinformatics, Information sciences
- Your favorite blog
- `docker run -p 8080:8080 -e URL=http://example.com fish/ghost`
- Empower people to run their own services

BUILDING INFRASTRUCTURES?

90S INFRASTRUCTURES

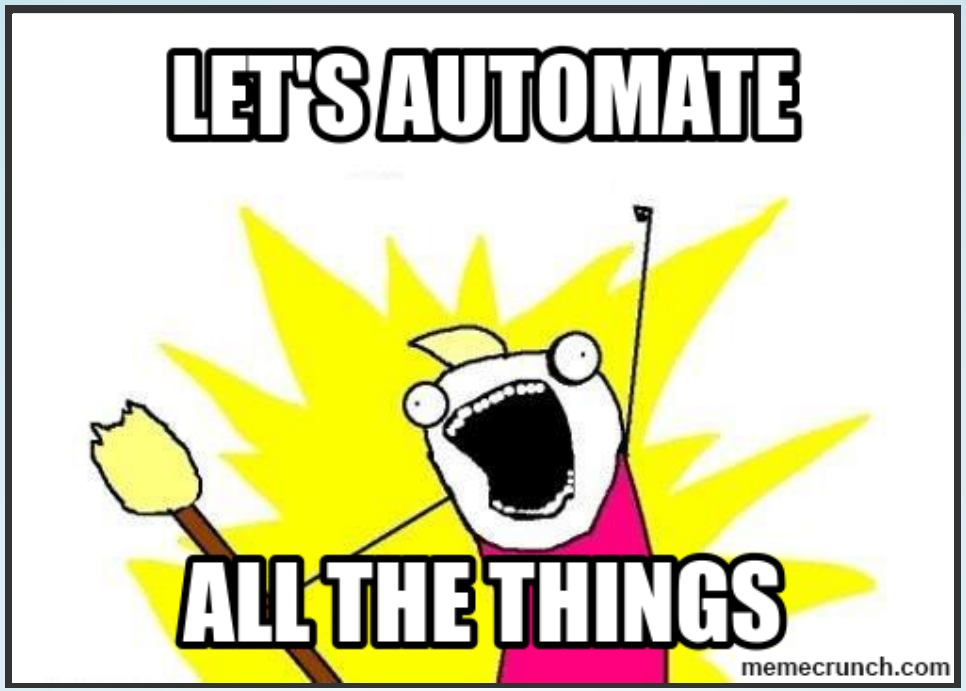
- Propritary, homogeneous and often vertically scaled stack
- Release cycles of months
- Huge companies targeting rather small audience/vertical businesses
- Huge IT/Ops departments

TODAY'S INFRASTRUCTURES

- Interconnected services scaled horizontal in heterogenious environment
- Lots of spinning wheels
- Several deploys per day
- Small but fast growing startups targeting Millions of users

CHALLENGES

- Managing such infrastructures is *incredible* hard
- Millions of knobs and switches, Billions of possible combinations
- High complexity make it hard to reason about
- Nobody can completely understand it



Go away or I will replace you with a very small shell script

STATE CONVERGENCE

- Describe what you want and how to get there
- Change state somewhere, encode what might affected
- CFEngine, Puppet, Chef, Ansible, Salt
- Manage everything!
- Doesn't *solve* anything

MANAGING COMPLEXITY

- Similar problems:
- Software complexity
- modules, classes, plugins
- Human communication
- named concepts like cat, nerd or car
- Shipping goods
- intermodal containers
- Solution: Abstraction!

WHERE CAN DOCKER HELP?

- Containers = abstracted application, including dependencies and configuration
- The *container* is the same where ever it runs
- Lightweight; can be deployed/rolled back fast and easily
- Isolation makes sure one container isn't affecting others
- Clear separation of concerns

DEVELOPER: WHY I LOVE DOCKER

Because I just have to care about **my** container:

- my libraries
- my package manager
- my code

I own the container and I don't care where it's running.

OPS: WHY I LOVE DOCKER

Because I just have to care about running containers:

- provide systems with the Docker
- resources planning & monitoring
- orchestration, remote access

I own the platform.

WHAT IS DOCKER EXACTLY DOING?

RUNNING COMMANDS ON:

- **immutable, shippable, layered images**
- with **copy-on-write storage** on top
- in **isolated environment**
- via **RESTish API**

SHIPPABLE IMAGES

- immutable
- multiple layers
- defined by **Dockerfile**, built by **builder**
- pushed to/pulled from Docker **registry**

DOCKERFILE/BUILDER

- Dockerfile: Simple text file with instructions:

```
 FROM ubuntu
MAINTAINER Johannes 'fish' Ziemke <fish@docker.com>
RUN apt-get -yq update
RUN apt-get -yq install nginx
ENTRYPOINT [ "/usr/sbin/nginx" ]
CMD [ "-g", "daemon off" ]
```

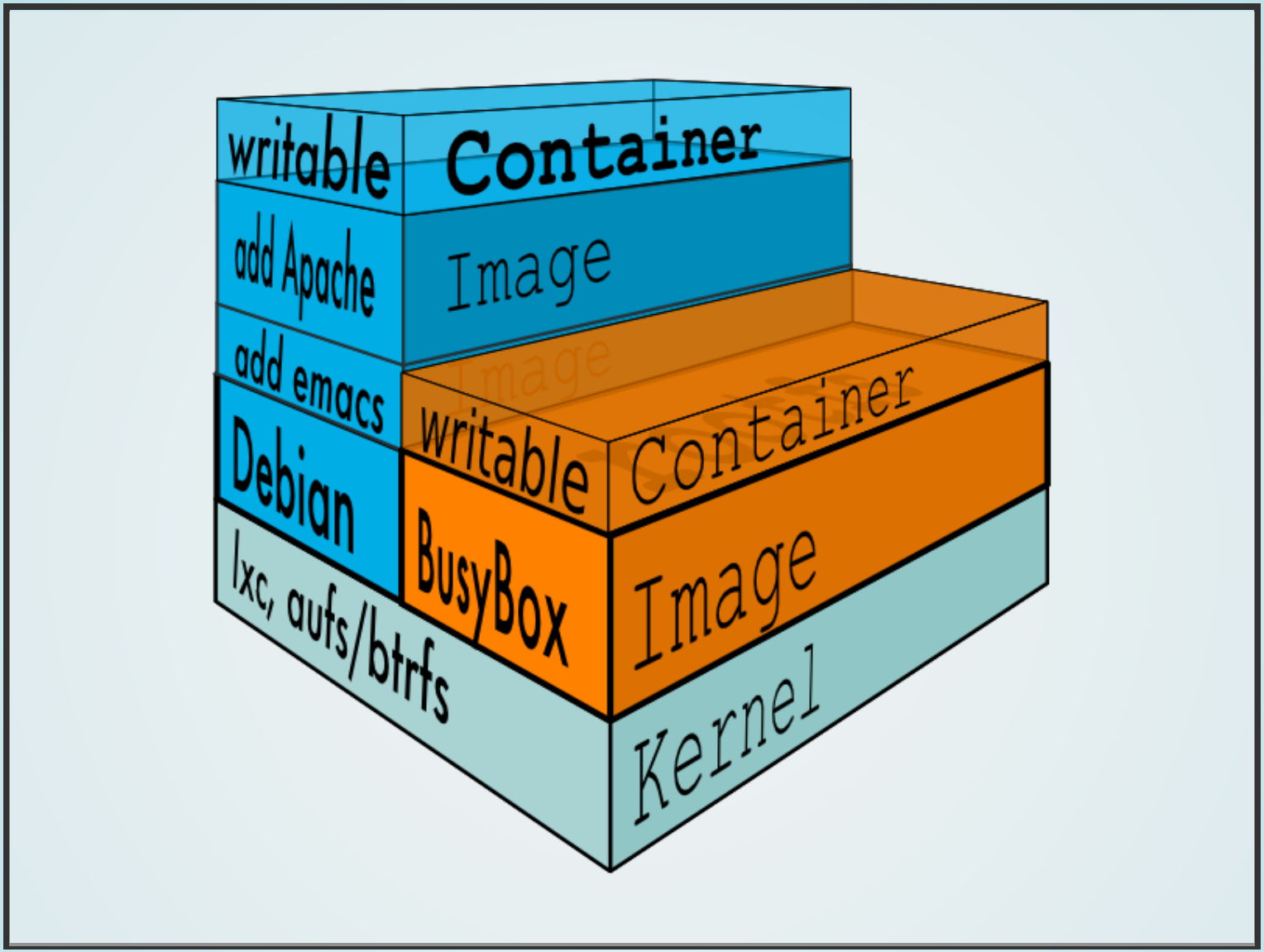
- `docker build` creates image from Dockerfile
- Each instruction creates new layer
- If instruction hasn't changed, uses cached layer
- `docker push` uploads image layers to **registry**

DOCKER REGISTRY

- hosting/delivery of images
- open source project
- supports various storage backends
- hosted platform: hub.docker.com

COPY-ON-WRITE STORAGE

- Provides writable layer on top of (read-only) images
- Persists all changes done by running container
- Pluggable, supported drivers:
- aufs
- btrfs
- devicemapper


writable
Container
add Apache
Image
add emacs
Debian
writable
Container
BusyBox
Image
lxc, aufs/btrfs
Kernel

ISOLATED EXECUTION

- Plugable, supported: lxc, native
- Using kernel features:
- namespaces
- Isolation by scoping
- Available: pid, mnt, net, uts, ipc, user
- cgroups (control groups)
- limit, account and isolate resources
- CPU, memory, I/O and general devices
- Future: solaris zones, BSD jails, full blown virtualization

DOCKER API

- RESTish API, defaults to UNIX socket
- Optional TLS client and server authentication
- *The system API*
- No need for any other remote access
- Ready for building your infrastructure deployment/automation on top

MISSING PIECES

- Docker can be used for building, sharing and deploying you services
- Docker does not (yet) address service discovery or dynamic scheduling
- Projects to close the gap
- OpenStack
- CoreOS
- Mesos + Marathon + Mesos-Docker
- flynn.io

DEMO TIME!

THANKS!

QUESTIONS?

- Johannes 'fish' Ziemke fish@docker.com/fish@freigeist.org
- GitHub/Twitter/Facebook: @discordianfish