Orchestrating Docker containers at scale

Maciej Lasyk

12 Sesja Linuksowa

Wrocław 2015-04-18

Join Fedora Infrastructure!

- learn Ansible
- learn Docker with Fedora Dockerfiles

http://fedoraproject.org/en/join-fedora

Quick survey

How many of you...

- Knows what Docker is?
- Played with Docker?
- Runs it on production?

Quick survey

How many of you...

- Knows what Docker is?
- Played with Docker?
- Runs it on production?

Quick survey

How many of you...

- Knows what Docker is?
- Played with Docker?
- Runs it on production?

Quick survey

How many of you...

- Knows what Docker is?
- Played with Docker?
- Runs it on production?

Why use Docker?

With Docker we can solve many problems

→ “it works on my machine”

→ reducing build & deploy time

→ Infrastructure configuration spaghetti – automation!

→ Libs dependency hell

→ Cost control and granularity

Why use Docker?

With Docker we can solve many problems

→ “it works on my machine”

→ reducing build & deploy time

→ Infrastructure configuration spaghetti – automation!

→ Libs dependency hell

→ Cost control and granularity

Why use Docker?

With Docker we can solve many problems

→ “it works on my machine”

→ reducing build & deploy time

→ Infrastructure configuration spaghetti – automation!

→ Libs dependency hell

→ Cost control and granularity

Why use Docker?

With Docker we can solve many problems

→ “it works on my machine”

→ reducing build & deploy time

→ Infrastructure configuration spaghetti – automation!

→ Libs dependency hell

→ Cost control and granularity

Why use Docker?

With Docker we can solve many problems

→ “it works on my machine”

→ reducing build & deploy time

→ Infrastructure configuration spaghetti – automation!

→ Libs dependency hell

→ Cost control and granularity

Why use Docker?

With Docker we can solve many problems

→ "it works on my machine"

→ reducing build & deploy time

→ Infrastructure configuration spaghetti – automation!

→ Libs dependency hell

→ Cost control and granularity

Docker – what is it?

“automates the deployment of any
application as a lightweight, portable,
self-sufficient container
that will run virtually anywhere”

Java’s promise: Write Once. Run Anywhere.

Java’s promise: Write Once. Run Anywhere.

Even on Windows now!

https://blog.docker.com/2014/10/docker-microsoft-partner-distributed-applications/

Is Docker is lightweight?

Is Docker is lightweight?

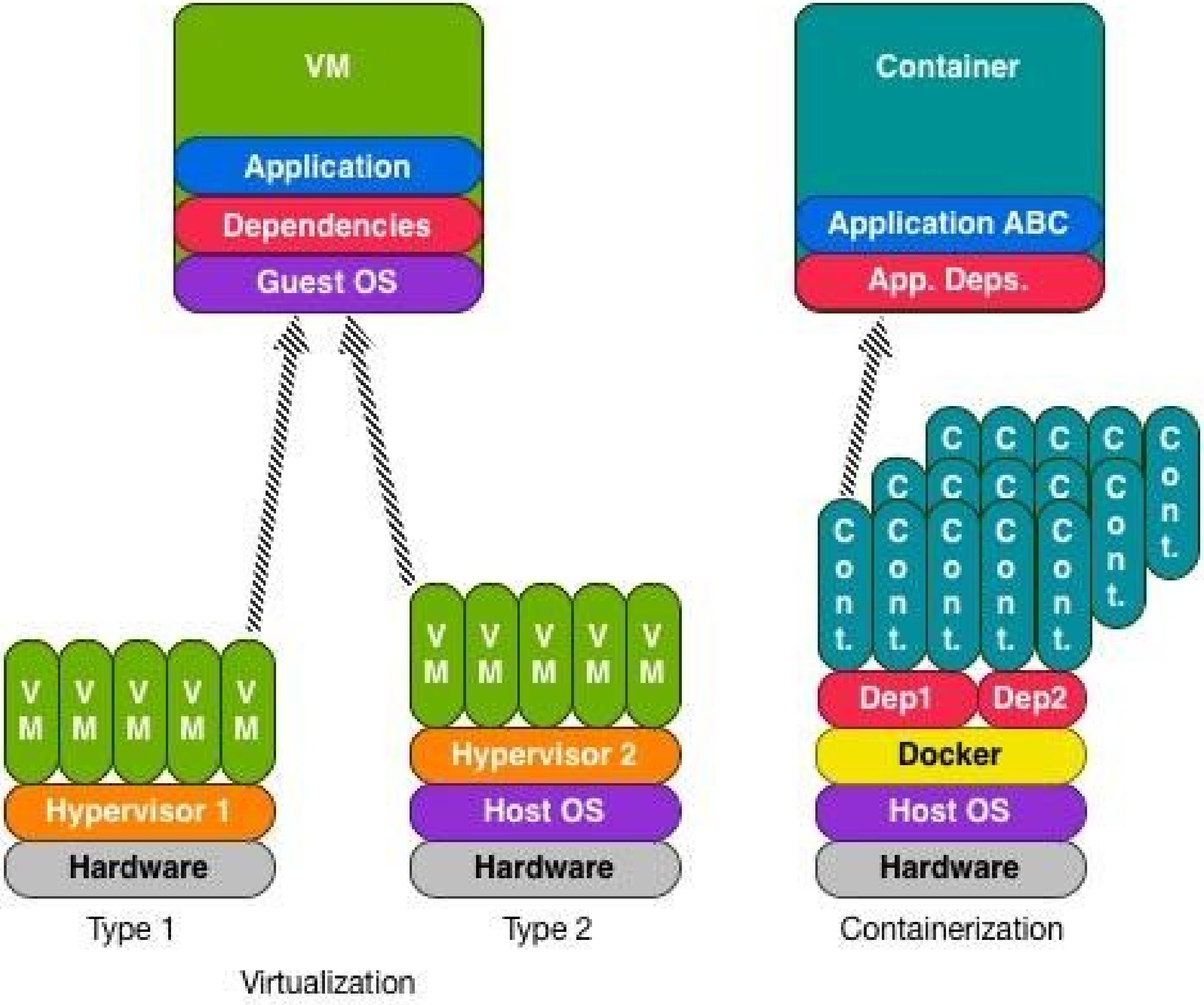
```
===========================================================
 Package              Arch              Version                    Repository          Size
===========================================================
Installing:
 docker-io            x86_64             1.3.0-1.fc20                  updates            4.3 M
```

Is Docker is lightweight?

```
==========================================================
 Package              Arch            Version               Repository        Size
==========================================================
Installing:
 docker-io            x86_64           1.3.0-1.fc20           updates          4.3 M
```

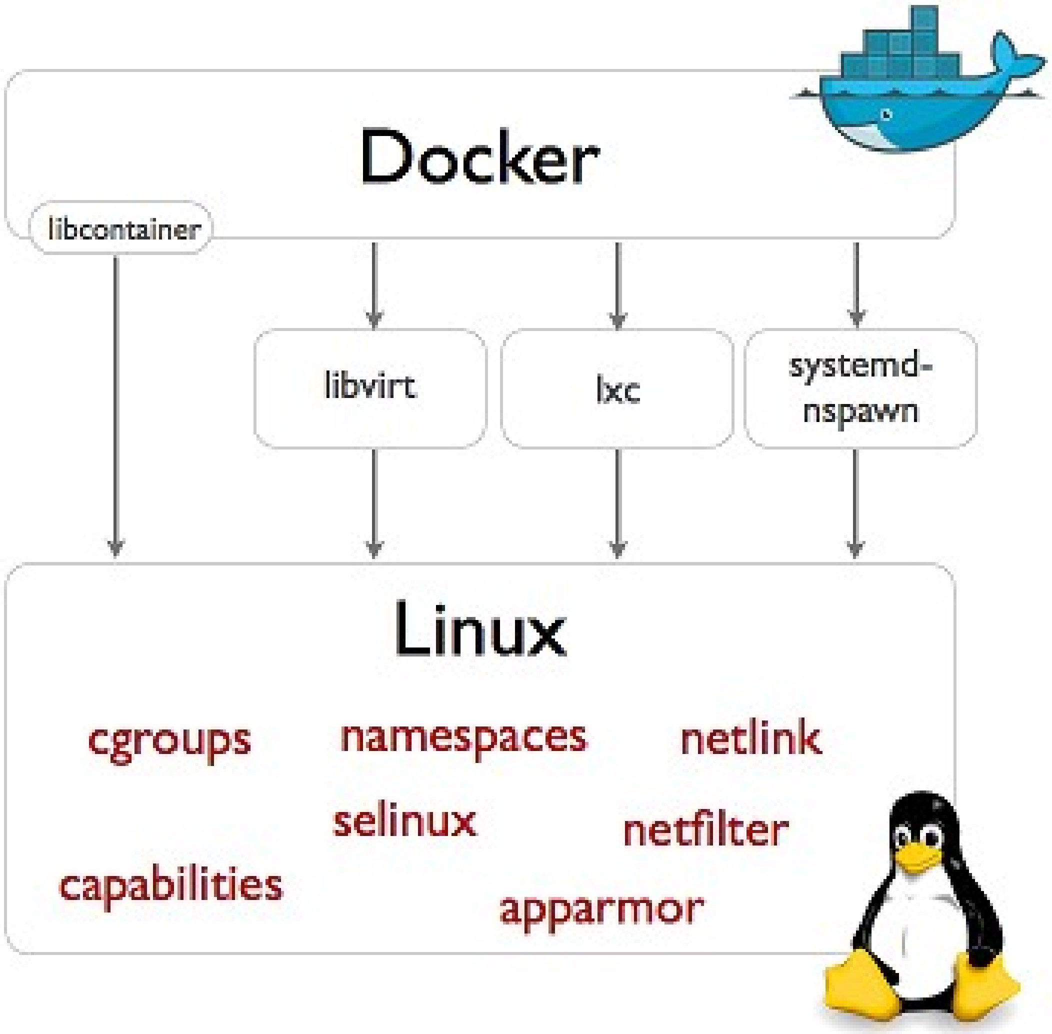
```
==========================================================
 Package              Arch            Version               Repository        Size
==========================================================
Installing:
 docker-io            x86_64           1.5.0-2.fc21           updates          26 M
```

VMs vs. Containers



http://sattia.blogspot.com/2014/05/docker-lightweight-linux-containers-for.html

Docker – how it works?



http://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/

Docker – how it works?

→ LXC & libcontainer

→ control groups

→ kernel namespaces

→ layered filesystem

  → no more AUFS (perf sucks)

  → devmapper thin provisioning & loopback mounts

  → OverlayFS!

Docker – how it works?

→ LXC & libcontainer

→ control groups

→ kernel namespaces

→ layered filesystem

  → no more AUFS (perf sucks)

  → devmapper thin provisioning & loopback mounts

  → OverlayFS!

Docker – how it works?

- → LXC & libcontainer
- → control groups
- → kernel namespaces
- → layered filesystem
  - → no more AUFS (perf sucks)
  - → devmapper thin provisioning & loopback mounts
  - → OverlayFS!

Docker – how it works?

- → LXC & libcontainer
- → control groups
- → kernel namespaces
- → layered filesystem
  - → no more AUFS (perf sucks)
  - → devmapper thin provisioning & loopback mounts
  - → OverlayFS!

Docker – how it works?

- → LXC & libcontainer
- → control groups
- → kernel namespaces
- → layered filesystem
  - → no more AUFS (perf sucks)
  - → devmapper thin provisioning & loopback mounts
  - → OverlayFS!

Docker – how it works?

- → LXC & libcontainer
- → control groups
- → kernel namespaces
- → layered filesystem
  - → no more AUFS (perf sucks)
  - → devmapper thin provisioning & loopback mounts
  - → OverlayFS!

Docker – how it works?

- → LXC & libcontainer
- → control groups
- → kernel namespaces
- → layered filesystem
  - → no more AUFS (perf sucks)
  - → devmapper thin provisioning & loopback mounts
  - → OverlayFS!

control groups (cgroups)

Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behavior

control groups (cgroups)

- → grouping processes
- → allocating resources to particular groups
  - → memory
  - → network
  - → CPU
  - → storage bandwidth (I/O throttling)
  - → device whitelisting

control groups (cgroups)

- → grouping processes
- → allocating resources to particular groups
  - → memory
  - → network
  - → CPU
  - → storage bandwidth (I/O throttling)
  - → device whitelisting

control groups (cgroups)

→ grouping processes

→ allocating resources to particular groups

  → memory

  → network

  → CPU

  → storage bandwidth (I/O throttling)

  → device whitelisting

control groups (cgroups)

- → grouping processes
- → allocating resources to particular groups
  - → memory
  - → network
  - → CPU
  - → storage bandwidth (I/O throttling)
  - → device whitelisting

control groups (cgroups)

- → grouping processes
- → allocating resources to particular groups
  - → memory
  - → network
  - → CPU
  - → storage bandwidth (I/O throttling)
  - → device whitelisting

control groups (cgroups)

→ grouping processes

→ allocating resources to particular groups

  → memory

  → network

  → CPU

  → storage bandwidth (I/O throttling)

  → device whitelisting

control groups (cgroups)

→ grouping processes

→ allocating resources to particular groups

  → memory

  → network

  → CPU

  → storage bandwidth (I/O throttling)

  → device whitelisting

control groups (cgroups)

little demo?

Kernel Namespaces

Providing a unique views of the system for processes.

→ PID – PIDs isolation

→ NET – network isolation (via virt-ifaces; demo)

→ IPC – won't use this

→ MNT – chroot like; deals w/mountpoints

→ UTS – deals w/hostname

Kernel Namespaces

Providing a unique views of the system for processes.

→ PID – PIDs isolation

→ NET – network isolation (via virt-ifaces; demo)

→ IPC – won't use this

→ MNT – chroot like; deals w/mountpoints

→ UTS – deals w/hostname

Kernel Namespaces

Providing a unique views of the system for processes.

→ PID – PIDs isolation

→ NET – network isolation (via virt-ifaces; demo)

→ IPC – won't use this

→ MNT – chroot like; deals w/mountpoints

→ UTS – deals w/hostname

Kernel Namespaces

Providing a unique views of the system for processes.

→ PID – PIDs isolation

→ NET – network isolation (via virt-ifaces; demo)

→ IPC – won't use this

→ MNT – chroot like; deals w/mountpoints

→ UTS – deals w/hostname

Kernel Namespaces

Providing a unique views of the system for processes.

→ PID – PIDs isolation

→ NET – network isolation (via virt-ifaces; demo)

→ IPC – won't use this

→ MNT – chroot like; deals w/mountpoints

→ UTS – deals w/hostname

Kernel Namespaces

Providing a unique views of the system for processes.

→ PID – PIDs isolation

→ NET – network isolation (via virt-ifaces; demo)

→ IPC – won't use this

→ MNT – chroot like; deals w/mountpoints

→ UTS – deals w/hostname

Kernel Namespaces

little demo?

OverlayFS

→ hell fast (you'll see)

→ page cache sharing

→ finally in upstream kernel (in rhel from 7.2)

→ finally supported by docker (-s overlay)

→ SELinux not there yet (but will be)

OverlayFS

→ hell fast (you'll see)

→ page cache sharing

→ finally in upstream kernel (in rhel from 7.2)

→ finally supported by docker (-s overlay)

→ SELinux not there yet (but will be)

OverlayFS

→ hell fast (you'll see)

→ page cache sharing

→ finally in upstream kernel (in rhel from 7.2)

→ finally supported by docker (-s overlay)

→ SELinux not there yet (but will be)

OverlayFS

→ hell fast (you'll see)

→ page cache sharing

→ finally in upstream kernel (in rhel from 7.2)

→ finally supported by docker (-s overlay)

→ SELinux not there yet (but will be)

OverlayFS

→ hell fast (you'll see)

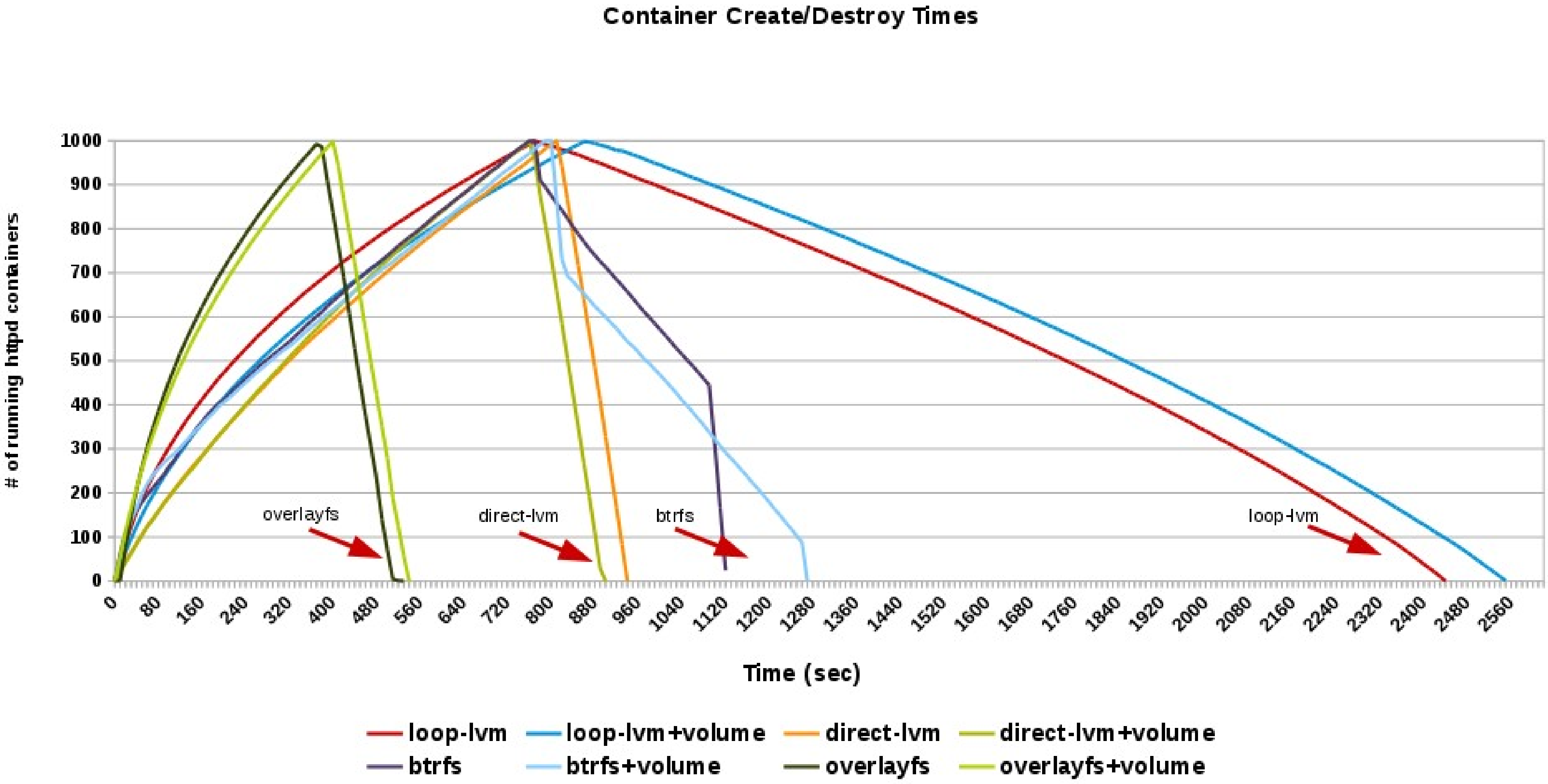
→ page cache sharing

→ finally in upstream kernel (in rhel from 7.2)

→ finally supported by docker (-s overlay)

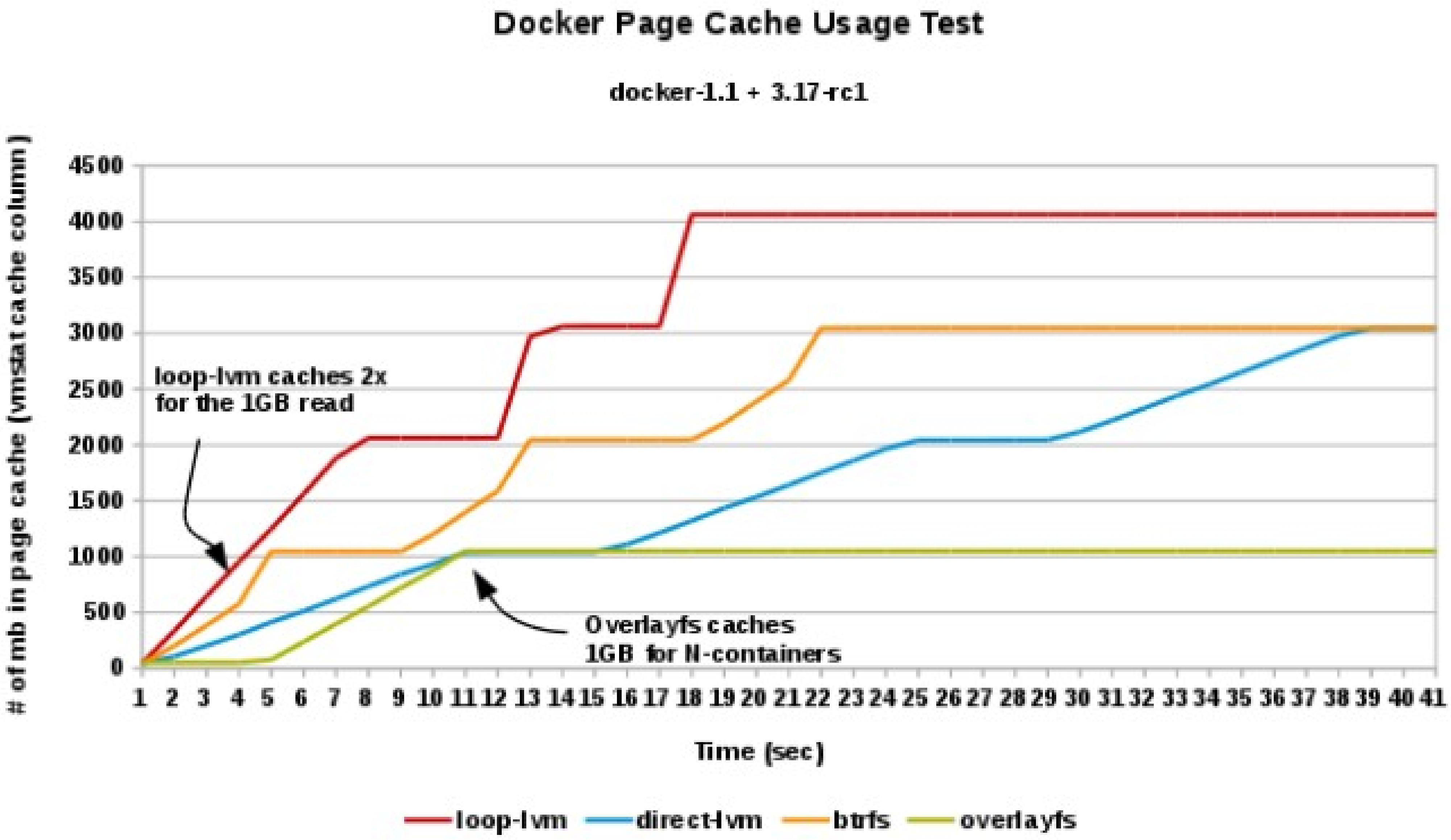
→ SELinux not there yet (but will be)

OverlayFS



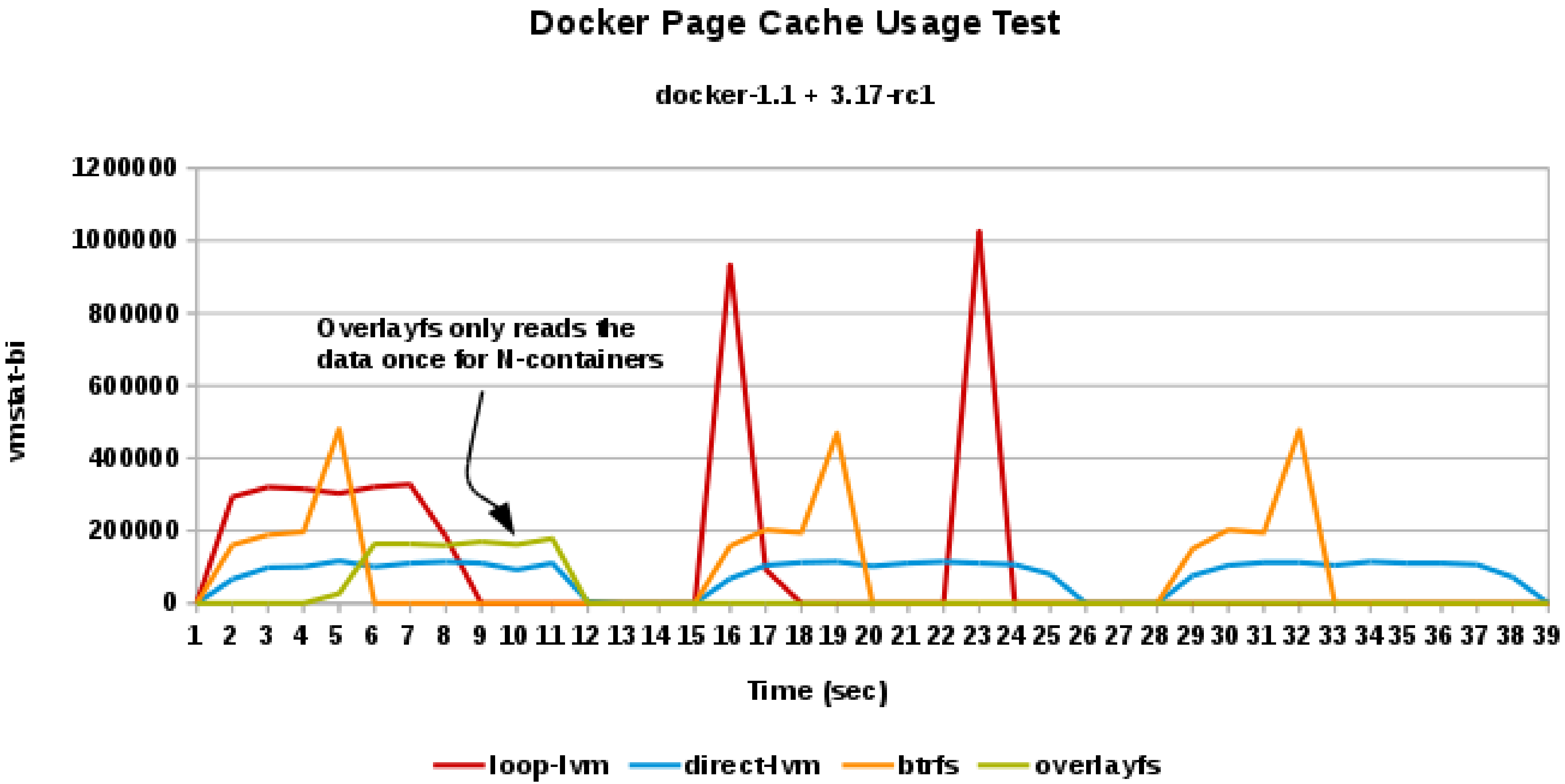
http://developerblog.redhat.com/2014/09/30/overview-storage-scalability-docker/

OverlayFS



http://developerblog.redhat.com/2014/09/30/overview-storage-scalability-docker/

OverlayFS



http://developerblog.redhat.com/2014/09/30/overview-storage-scalability-docker/

OverlayFS

little demo?

Linux containers equation

Linux Containers = namespaces + cgroups + storage

Docker – concepts

Images

- → read only
- → act as templates

Dockerfile

- → like a makefile
- → commands order & cache'ing
- → extends the base image
- → results in a new image

Containers: instances running apps

Docker – concepts

Images

→ read only

→ act as templates

Dockerfile

→ like a makefile

→ commands order & cache'ing

→ extends the base image

→ results in a new image

Containers: instances running apps

Docker – concepts

Images

→ read only

→ act as templates

Dockerfile

→ like a makefile

→ commands order & cache'ing

→ extends the base image

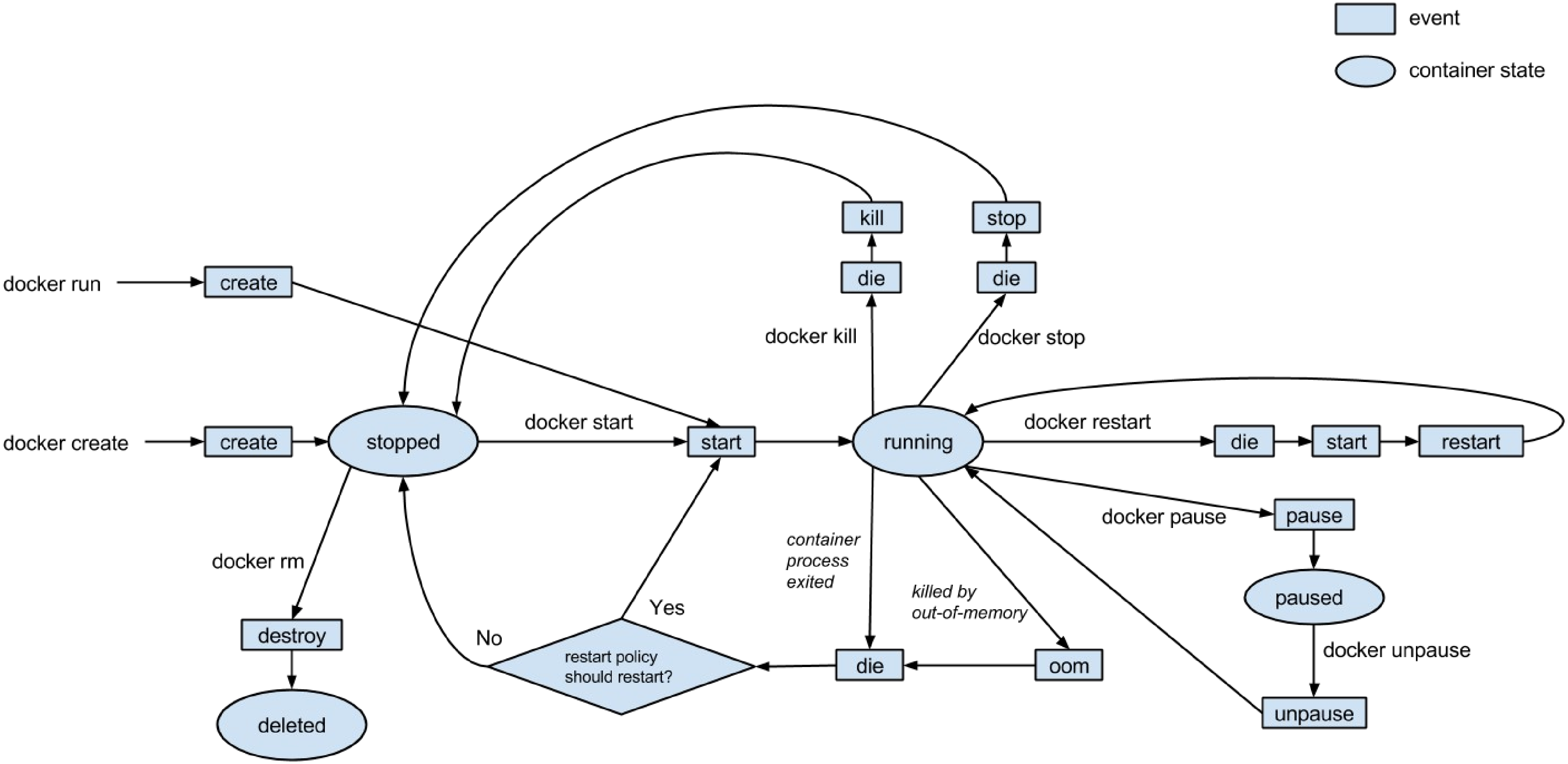
→ results in a new image

Containers: instances running apps

Docker – concepts

dockerfile + base image = docker container

Docker – events



http://gliderlabs.com/blog/2015/04/14/docker-events-explained/

Dockerfile

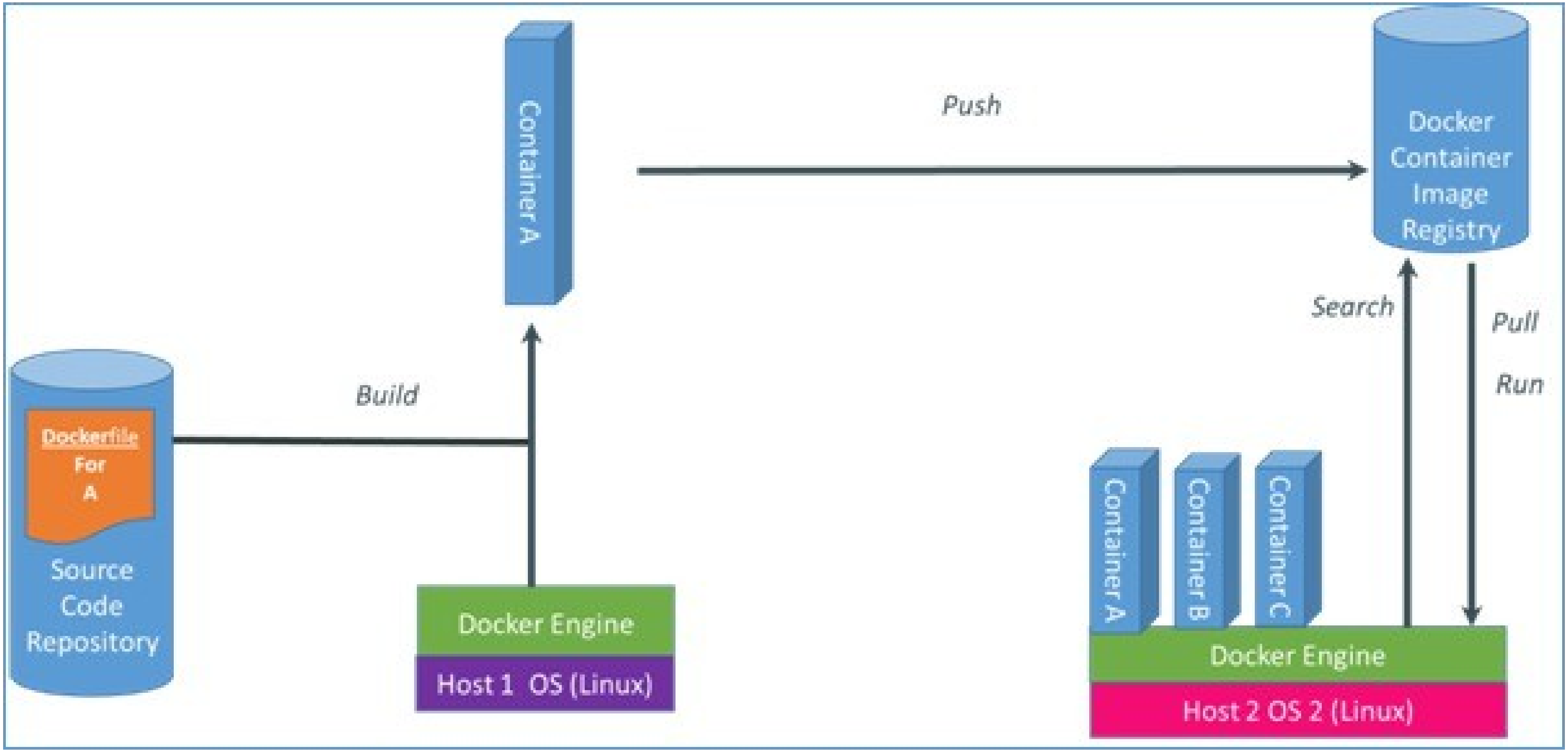
```
FROM fedora
MAINTAINER scollier <scollier@redhat.com>

RUN yum -y update && yum clean all
RUN yum -y install nginx && yum clean all
RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN echo "nginx on Fedora" > /srv/www/index.html

EXPOSE 80

CMD [ "/usr/sbin/nginx" ]
```

Docker – registry



http://osv.io/blog/blog/2014/06/19/containers-hypervisors-part-2/

Docker – registry

→ git like semantics

→ pull, push, commit

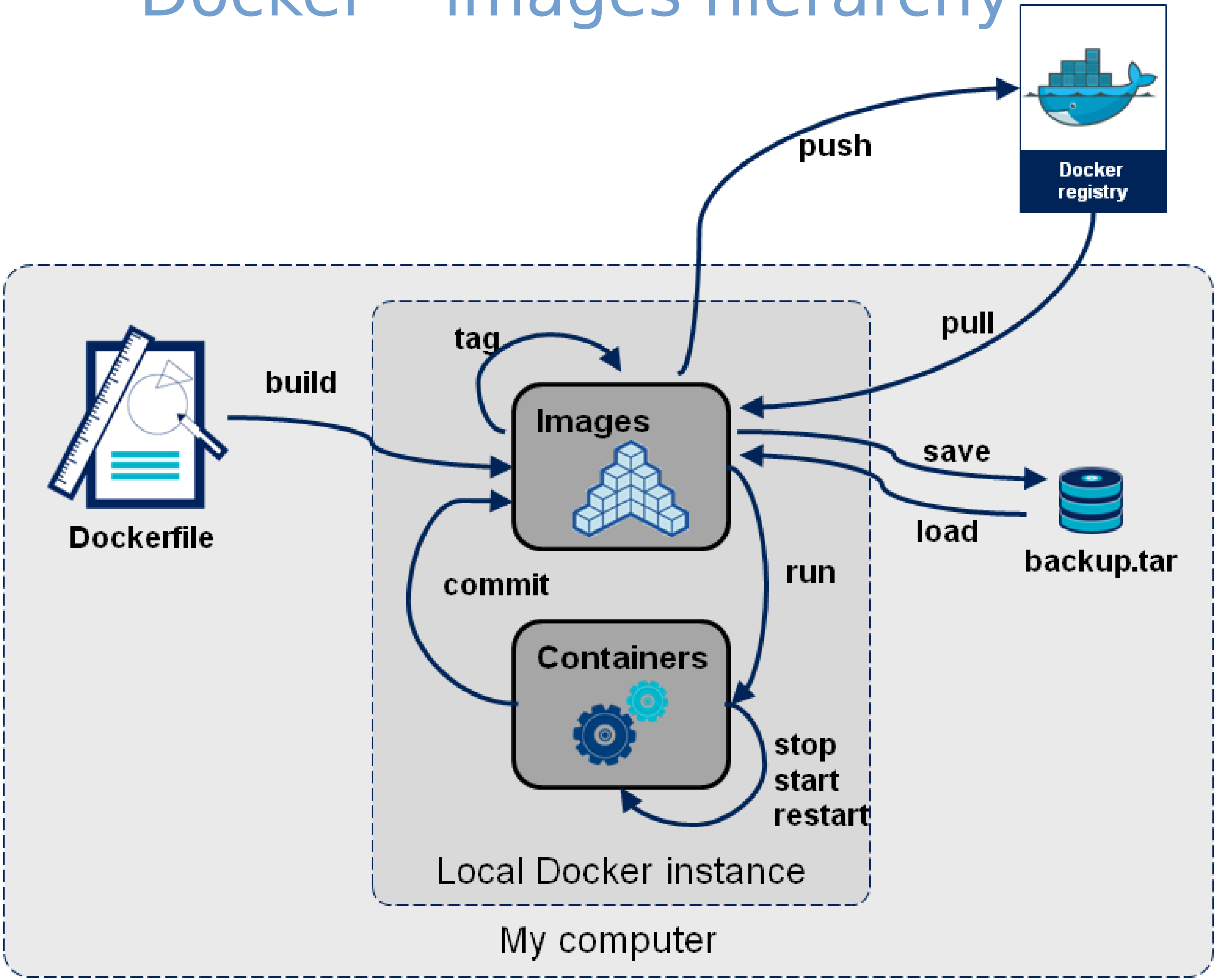
→ private and public registry

→ https://github.com/dotcloud/docker-registry

→ yum install docker-registry

```
$ docker pull
$ docker push
$ docker commit
```

Docker – images hierarchy



http://blog.octo.com/en/docker-registry-first-steps/

Docker – images hierarchy

```
base image
    -> child image
        -> grandchild image
```

Docker – images hierarchy

```
base image
    -> child image
        -> grandchild image
```

Git’s promise: Tiny footprint with lightning fast performance

Docker – security

→ Isolation via kernel namespaces

→ Each container gets own network stack

→ Control groups for resources limiting

→ Additional layer of security: SELinux / AppArmor / GRSEC

f20 policy: https://git.fedorahosted.org/cgit/selinux-policy.git/tree/docker.te?h=f20-contrib

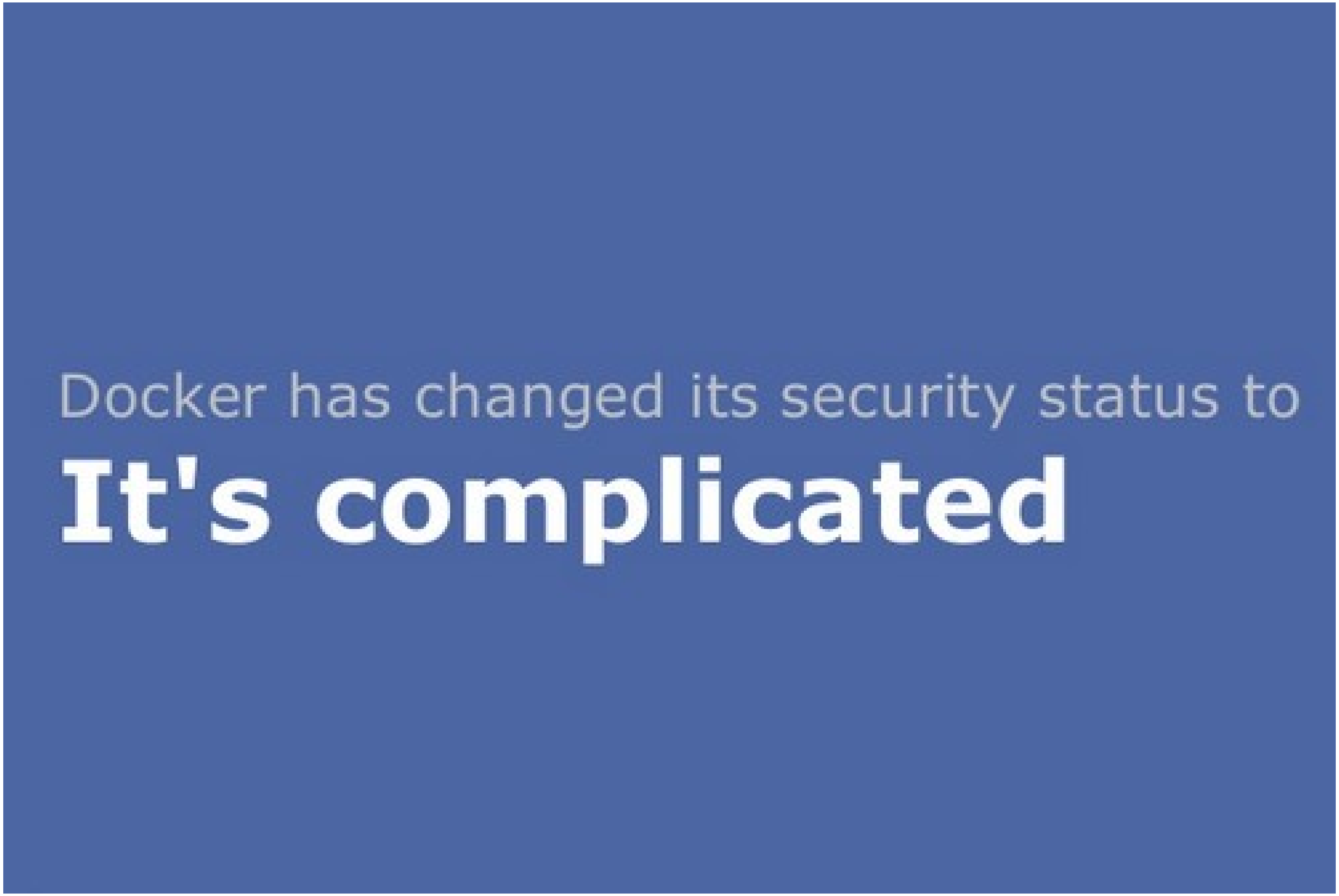
What's there?

```
seinfo -t -x | grep docker
```

```
sesearch -A -s docker_t (and the rest)
```

or just unpack docker.pp with semodule_unpackage

Docker – security



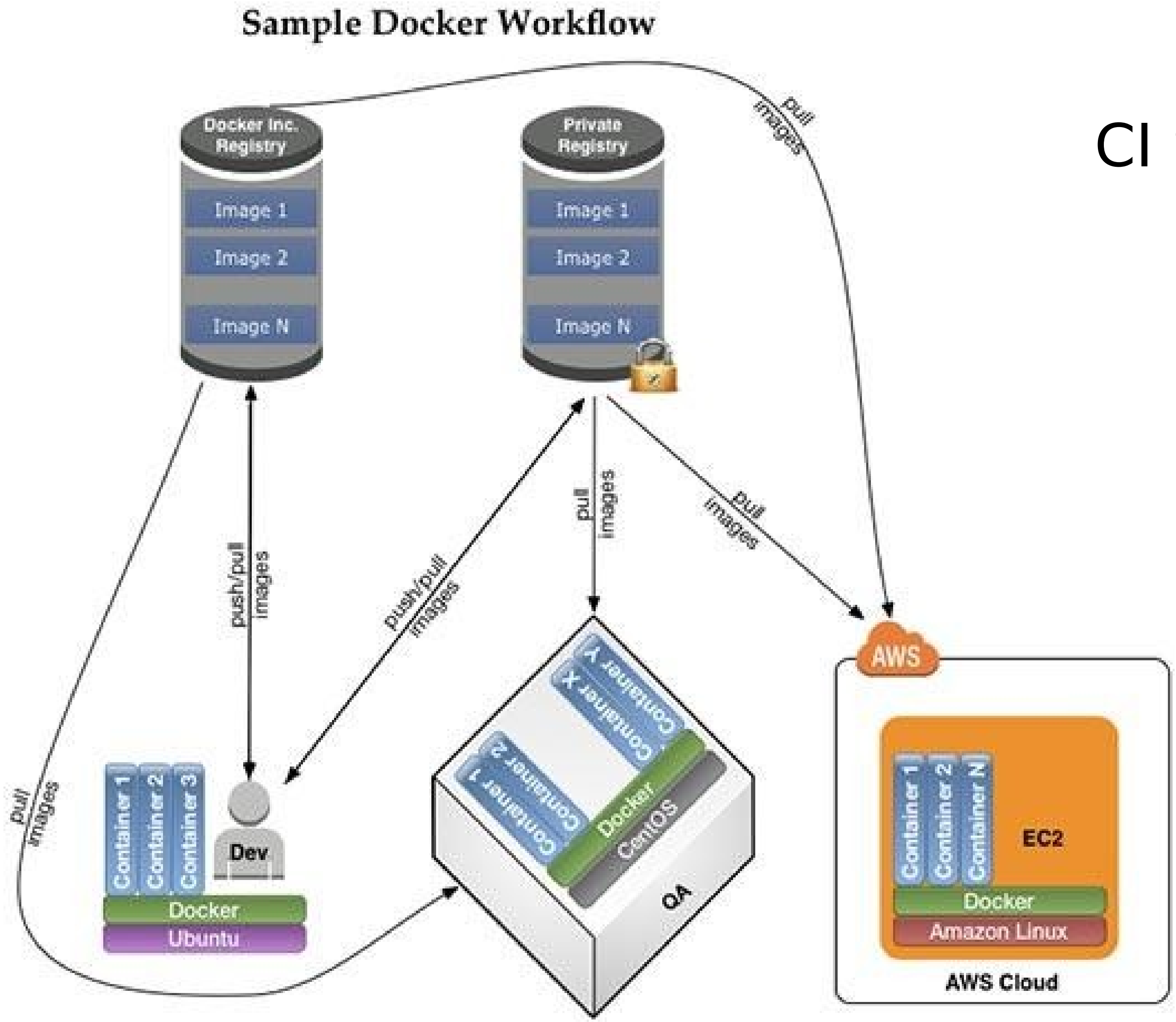
Docker has changed its security status to

It's complicated

http://www.projectatomic.io/blog/2014/08/is-it-safe-a-look-at-docker-and-security-from-linuxcon/

Docker – security

Docker – use cases



CI Stack

http://sattia.blogspot.com/2014/05/docker-lightweight-linux-containers-for.html

Docker – use cases

→ Continuous Integration

→ local dev

  → with Docker it's easy to standardize envs

→ deployment (rolling updates (e.g. w/Ansible))

→ testing

  → unit testing of any commit on dedicated env

  → don't worry about cleaning up after testing

  → paralleled tests across any machines

Docker – use cases

→ version control system for apps

→ microservices

  → Docker embraces granularity

  → Services can be deployed independently and faster

  → paralleled tests across any machines

→ continuous delivery

→ PaaS

Docker – history

→ 2013-01: dotCloud worked on own PaaS (Python based)

→ 2013-03: Docker went public (AUFS, LXC)

→ middle 2013: Red Hat joined, devmapper, SELinux

→ late 2013: removed LXC, rewritten in Go

→ 2014-02: stable 1.0

Docker – history

→ 2013-01: dotCloud worked on own PaaS (Python based)

→ 2013-03: Docker went public (AUFS, LXC)

→ middle 2013: Red Hat joined, devmapper, SELinux

→ late 2013: removed LXC, rewritten in Go

→ 2014-02: stable 1.0

Docker – history

→ 2013-01: dotCloud worked on own PaaS (Python based)

→ 2013-03: Docker went public (AUFS, LXC)

→ middle 2013: Red Hat joined, devmapper, SELinux

→ late 2013: removed LXC, rewritten in Go

→ 2014-02: stable 1.0

Docker – history

→ 2013-01: dotCloud worked on own PaaS (Python based)

→ 2013-03: Docker went public (AUFS, LXC)

→ middle 2013: Red Hat joined, devmapper, SELinux

→ late 2013: removed LXC, rewritten in Go

→ 2014-02: stable 1.0

Docker – history

→ 2013-01: dotCloud worked on own PaaS (Python based)

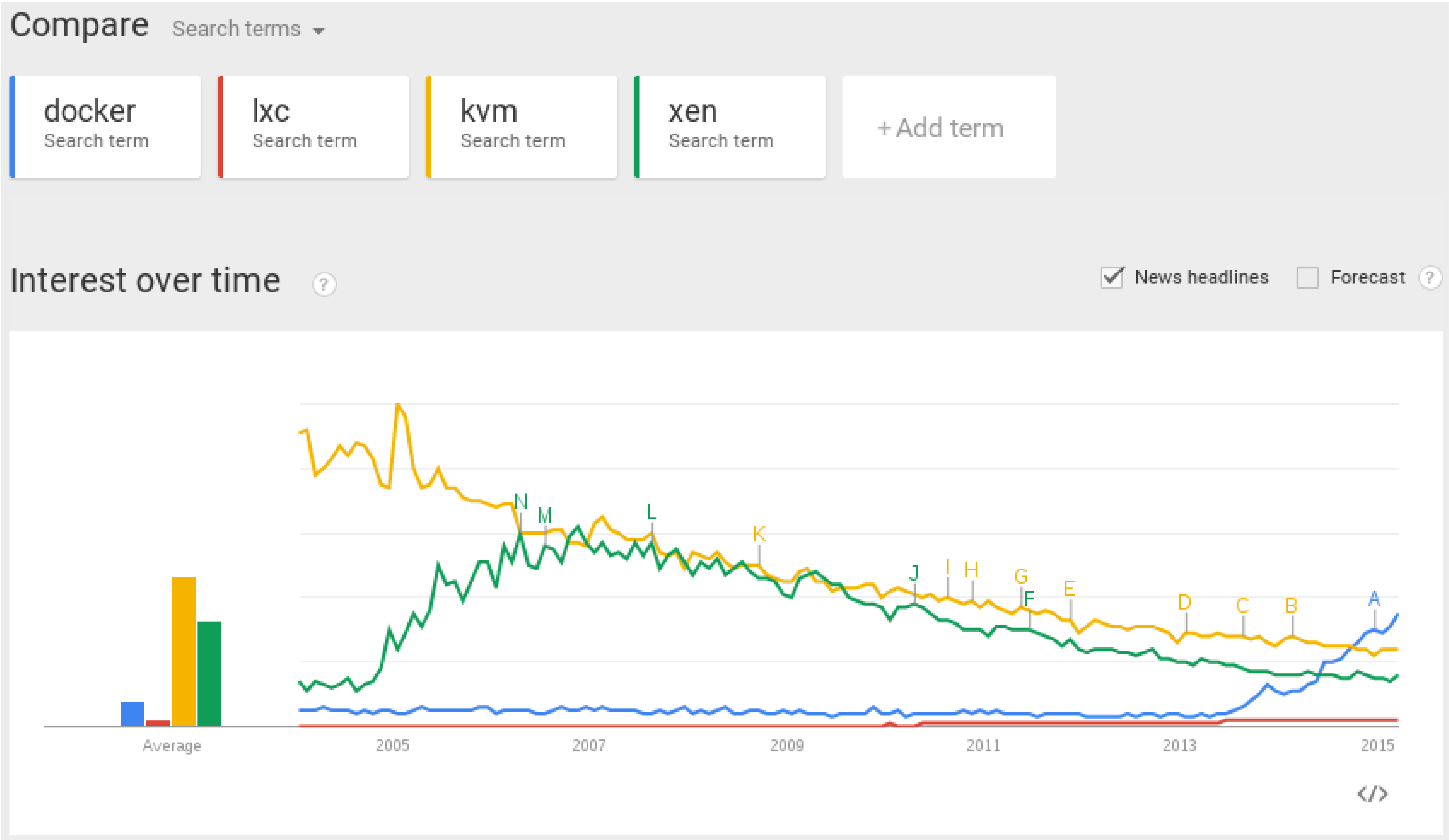
→ 2013-03: Docker went public (AUFS, LXC)

→ middle 2013: Red Hat joined, devmapper, SELinux

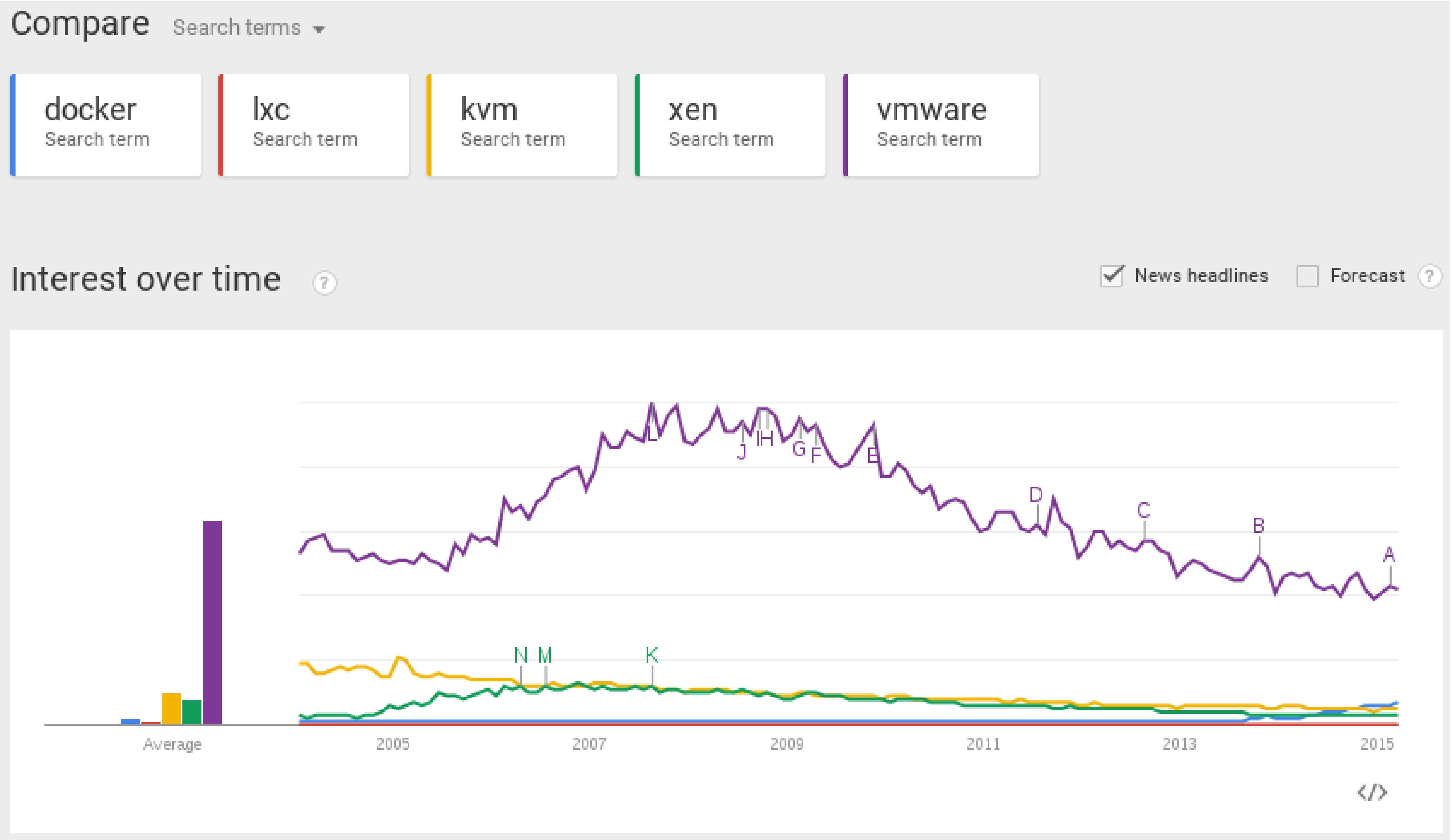
→ late 2013: removed LXC, rewritten in Go

→ 2014-02: stable 1.0

Docker – popularity



Docker – popularity



Compare Search terms

docker
Search term

lxc
Search term

kvm
Search term

xen
Search term

vmware
Search term

Interest over time

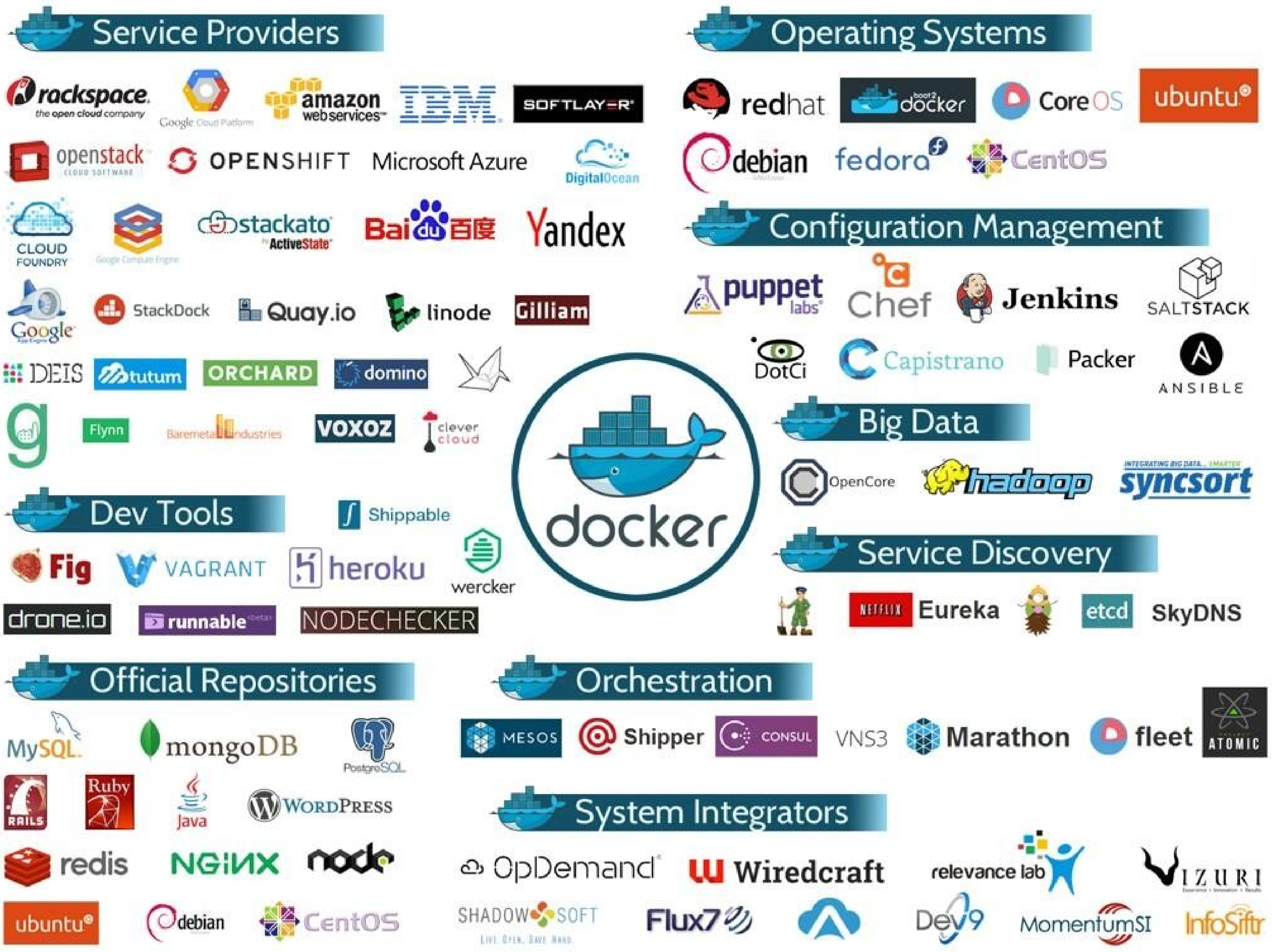
☑ News headlines □ Forecast

Orchestration at scale w/Docker

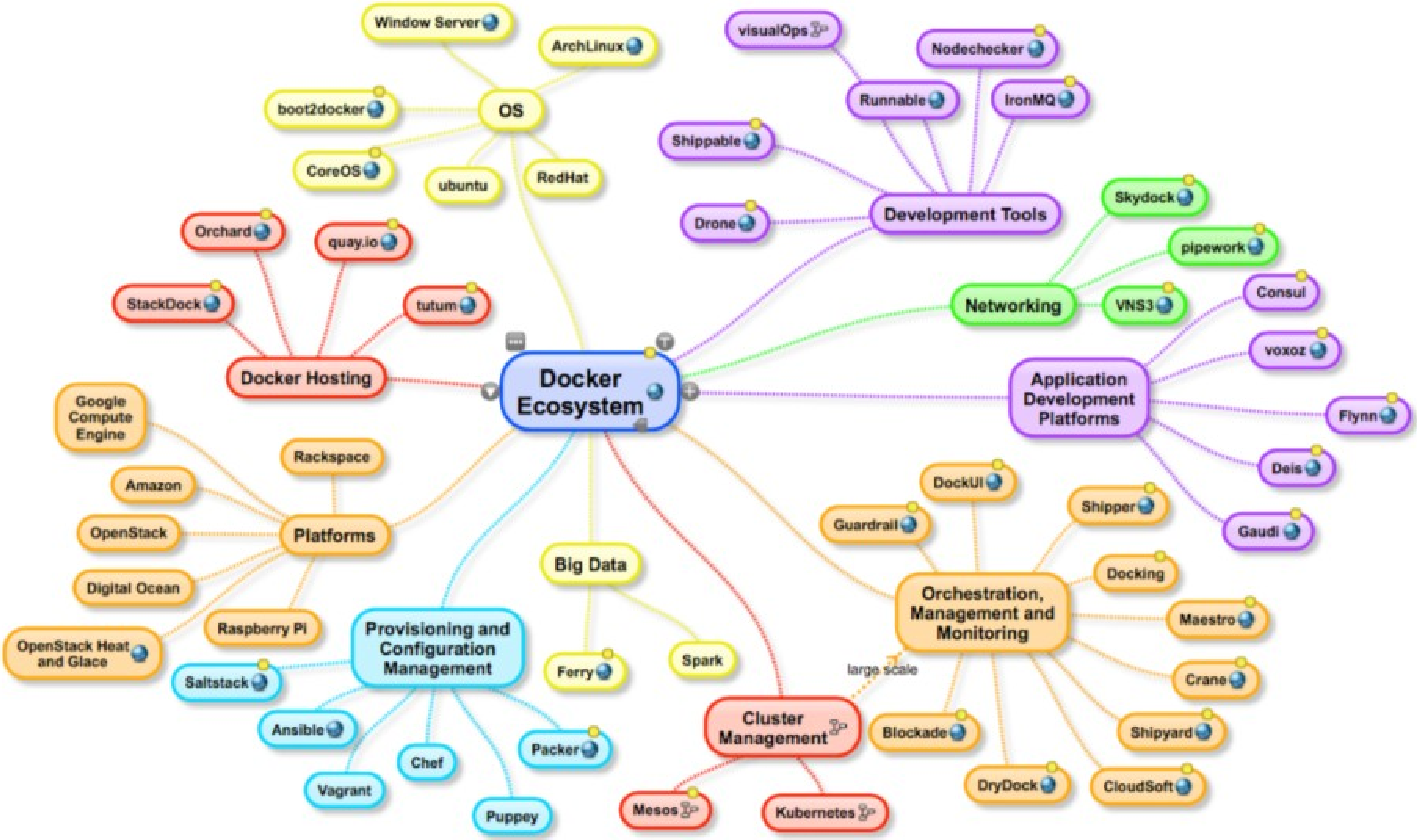
Orchestration at scale w/Docker

This might be a little problem

Orchestration at scale w/Docker

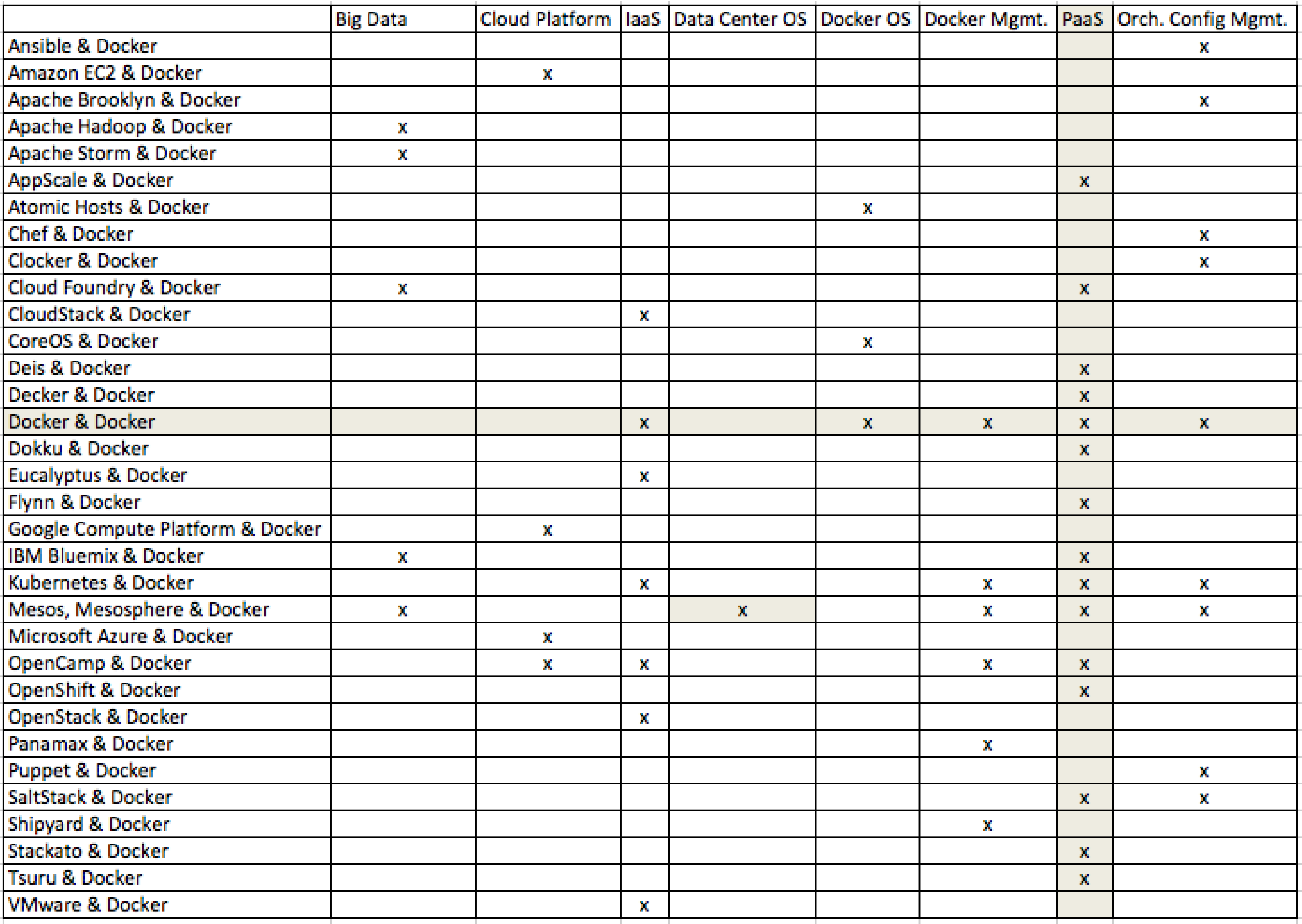


Orchestration at scale w/Docker



- Docker Ecosystem
  - OS
    - Window Server
    - ArchLinux
    - boot2docker
    - CoreOS
    - ubuntu
    - RedHat
  - Development Tools
    - visualOps
    - Nodechecker
    - Runnable
    - IronMQ
    - Shippable
    - Drone
  - Networking
    - Skydock
    - pipework
    - VNS3
  - Application Development Platforms
    - Consul
    - voxoz
    - Flynn
    - Deis
    - Gaudi
  - Docker Hosting
    - Orchard
    - quay.io
    - StackDock
    - tutum
  - Platforms
    - Google Compute Engine
    - Rackspace
    - Amazon
    - OpenStack
    - Digital Ocean
    - Raspberry Pi
    - OpenStack Heat and Glace
  - Provisioning and Configuration Management
    - Saltstack
    - Ansible
    - Chef
    - Packer
    - Vagrant
    - Puppey
  - Big Data
    - Ferry
    - Spark
  - Cluster Management
    - Mesos
    - Kubernetes
  - Orchestration, Management and Monitoring
    - DockUI
    - Guardrail
    - Shipper
    - Docking
    - Maestro
    - Crane
    - Shipyard
    - CloudSoft
    - DryDock
    - Blockade

large scale



	Big Data	Cloud Platform	IaaS	Data Center OS	Docker OS	Docker Mgmt.	PaaS	Orch. Config Mgmt.
Ansible & Docker								x
Amazon EC2 & Docker		x						
Apache Brooklyn & Docker								x
Apache Hadoop & Docker	x							
Apache Storm & Docker	x							
AppScale & Docker							x	
Atomic Hosts & Docker					x			
Chef & Docker								x
Clocker & Docker								x
Cloud Foundry & Docker	x						x	
CloudStack & Docker			x					
CoreOS & Docker					x			
Deis & Docker							x	
Decker & Docker							x	
Docker & Docker			x		x	x	x	x
Dokku & Docker							x	
Eucalyptus & Docker			x					
Flynn & Docker							x	
Google Compute Platform & Docker		x						
IBM Bluemix & Docker	x						x	
Kubernetes & Docker			x			x	x	x
Mesos, Mesosphere & Docker	x			x		x	x	x
Microsoft Azure & Docker		x						
OpenCamp & Docker		x	x			x	x	
OpenShift & Docker							x	
OpenStack & Docker			x					
Panamax & Docker						x		
Puppet & Docker								x
SaltStack & Docker							x	x
Shipyard & Docker						x		
Stackato & Docker							x	
Tsuru & Docker							x	
VMware & Docker			x					

http://www.cloudssky.com/en/blog/Docker-Is-Not-Enough

To Paas or not to PaaS?

To Paas or not to PaaS?

→ you think PaaS will solve your problems?

→ it will rather clone them :)

→ installing PaaS might be easy

→ operating PaaS will be tough

→ so is operating your apps tough enough to move?

→ private PaaS or not?

→ is your app ready for PaaS?

→ Rainbow and Unicorn Piss:

http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

To Paas or not to PaaS?

→ you think PaaS will solve your problems?

→ it will rather clone them :)

→ installing PaaS might be easy

→ operating PaaS will be tough

→ so is operating your apps tough enough to move?

→ private PaaS or not?

→ is your app ready for PaaS?

→ Rainbow and Unicorn Piss:

http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

To Paas or not to PaaS?

→ you think PaaS will solve your problems?

→ it will rather clone them :)

→ installing PaaS might be easy

→ operating PaaS will be tough

→ so is operating your apps tough enough to move?

→ private PaaS or not?

→ is your app ready for PaaS?

→ Rainbow and Unicorn Piss:

http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

To Paas or not to PaaS?

→ you think PaaS will solve your problems?

→ it will rather clone them :)

→ installing PaaS might be easy

→ operating PaaS will be tough

→ so is operating your apps tough enough to move?

→ private PaaS or not?

→ is your app ready for PaaS?

→ Rainbow and Unicorn Piss:

http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

To Paas or not to PaaS?

→ you think PaaS will solve your problems?

→ it will rather clone them :)

→ installing PaaS might be easy

→ operating PaaS will be tough

→ so is operating your apps tough enough to move?

→ private PaaS or not?

→ is your app ready for PaaS?

→ Rainbow and Unicorn Piss:

http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

To Paas or not to PaaS?

- → you think PaaS will solve your problems?
- → it will rather clone them :)
- → installing PaaS might be easy
- → operating PaaS will be tough
- → so is operating your apps tough enough to move?
- → private PaaS or not?
- → is your app ready for PaaS?
- → Rainbow and Unicorn Piss:

http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

To Paas or not to PaaS?

→ you think PaaS will solve your problems?

→ it will rather clone them :)

→ installing PaaS might be easy

→ operating PaaS will be tough

→ so is operating your apps tough enough to move?

→ private PaaS or not?

→ is your app ready for PaaS?

→ Rainbow and Unicorn Piss:

http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

To Paas or not to PaaS?

→ you think PaaS will solve your problems?

→ it will rather clone them :)

→ installing PaaS might be easy

→ operating PaaS will be tough

→ so is operating your apps tough enough to move?

→ private PaaS or not?

→ is your app ready for PaaS?

→ Rainbow and Unicorn Piss:

http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

To Paas or not to PaaS?

→ flynn.io

- Open source PaaS (Go)
- Uses Docker to manage containers
- Ops should be a product team, not consultants
- under development
- Git push deployment
- https://flynn.io

→ dokku

- The smallest PaaS implementation you've ever seen
- Docker powered mini-Heroku
- Less than 100 lines of Bash
- Git push deployment
- https://github.com/progrium/dokku

→ deis.io

- Open source PaaS (Python)
- Git push deployment
- On top of CoreOS w/ Docker
- http://deis.io/

Don't forget maestro-ng!
https://github.com/signalfuse/maestro-ng

To Paas or not to PaaS?

→ flynn.io

- Open source PaaS (Go)
- Uses Docker to manage containers
- Ops should be a product team, not consultants
- under development
- Git push deployment
- https://flynn.io

→ dokku

- The smallest PaaS implementation you've ever seen
- Docker powered mini-Heroku
- Less than 100 lines of Bash
- Git push deployment
- https://github.com/progrium/dokku

→ deis.io

- Open source PaaS (Python)
- Git push deployment
- On top of CoreOS w/ Docker
- http://deis.io/

Don't forget maestro-ng!
https://github.com/signalfuse/maestro-ng

To Paas or not to PaaS?

→ flynn.io

- Open source PaaS (Go)
- Uses Docker to manage containers
- Ops should be a product team, not consultants
- under development
- Git push deployment
- https://flynn.io

→ dokku

- The smallest PaaS implementation you've ever seen
- Docker powered mini-Heroku
- Less than 100 lines of Bash
- Git push deployment
- https://github.com/progrium/dokku

→ deis.io

- Open source PaaS (Python)
- Git push deployment
- On top of CoreOS w/ Docker
- http://deis.io/

Don't forget maestro-ng!
https://github.com/signalfuse/maestro-ng

To Paas or not to PaaS?

→ flynn.io

- Open source PaaS (Go)
- Uses Docker to manage containers
- Ops should be a product team, not consultants
- under development
- Git push deployment
- https://flynn.io

→ dokku

- The smallest PaaS implementation you've ever seen
- Docker powered mini-Heroku
- Less than 100 lines of Bash
- Git push deployment
- https://github.com/progrium/dokku

→ deis.io

- Open source PaaS (Python)
- Git push deployment
- On top of CoreOS w/ Docker
- http://deis.io/

Don't forget maestro-ng!
https://github.com/signalfuse/maestro-ng

To Paas or not to PaaS?

Remember, that PaaS might fail; plan & test for disaster !

Docker & CLI

```
$ docker run -t -i fedora /bin/bash
> yum -y update
> yum -y install nginx
$ docker commit 73fa45674fd fedora-nginx
$ docker run -d -p 80:80 fedora-nginx
```

Docker & CLI

Or via Dockerfile:

```
---
FROM fedora
MAINTAINER scollier <scollier@redhat.com>
RUN yum -y update
RUN yum -y install nginx
EXPOSE 80
CMD [ "/usr/sbin/nginx" ]
---
```

```
$ docker build -t fedora –-rm .
$ docker run --name=nginx fedora
```

FIG?

- http://www.fig.sh?

Actually: Docker-Compose

- http://docs.docker.com/compose/
- for single host env

Docker-Compose

- http://docs.docker.com/compose/
- for single host env

```
FROM python:2.7
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/
```

Docker-Compose

```
db:
  image: postgres
web:
  build: .
  command: python manage.py runserver 0.0.0.0:8000
  volumes:
    - /srv/app/code:/code
  ports:
    - "8000:8000"
  links:
    - db
```

Docker-Compose

```
$ alias fig='docker-compose'

$ fig run web django-admin.py startproject figexample .
$ fig up
```

Management during runtime?

```
$ fig run web python manage.py syncdb
```

Docker & Ansible

Ansible + Docker

&

Docker + Ansible

Docker & Ansible

Ansible docker core module:
http://docs.ansible.com/docker_module.html

```
- hosts: web
  sudo: yes
  tasks:
  - name: run tomcat servers
    docker: >
           image=centos
           command="service tomcat6 start"
           ports=8080
           count=5
           memory_limit=128MB
           link=mysql
           expose=8080
           registry=...
           volume=...
```

Docker & Ansible

Building image with Ansible:

```
FROM ansible/centos7-ansible:stable
ADD ansible /srv/example
WORKDIR /srv/example
RUN ansible-playbook web.yml -c local
EXPOSE 80
CMD ["/usr/sbin/nginx"]
```

Docker & Ansible

Building image with Ansible:

```
FROM ansible/centos7-ansible:stable
ADD ansible /srv/example
WORKDIR /srv/example
RUN ansible-playbook web.yml -c local
EXPOSE 80
CMD ["/usr/sbin/nginx"]
```

ansible/web.yml:

```
- name: Install webserver
  hosts: localhost
  tasks:
    - yum: pkg=nginx state=latest
    - shell: echo "ansible" > /usr/share/nginx/html/index.html
```

Docker & Ansible

Yet another demo?

CoreOS

→ Designed for massive server deployments

→ Support Docker container out of the box

→ It's a Chrome OS fork

→ Consists of couple of components:

  → SystemD – not just a init system ;)

  → Fleet – cluster level manager & scheduler

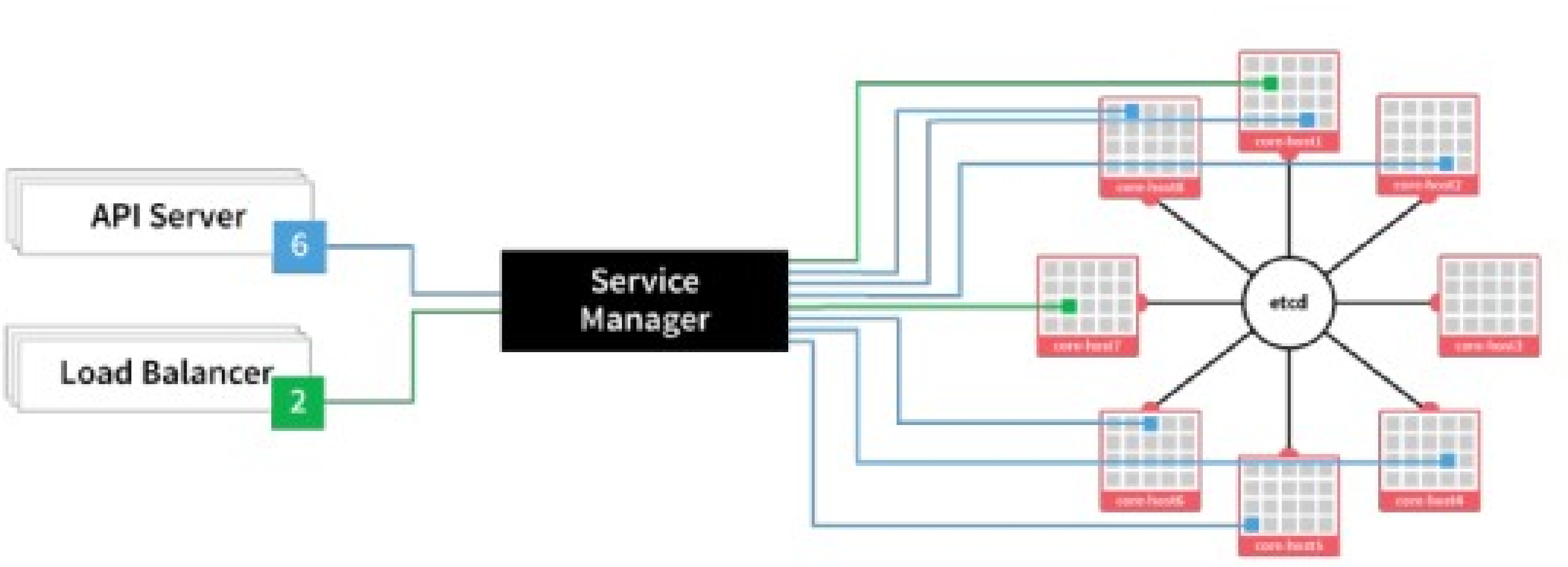
  → etcd – light & distributed key / value store

  → Docker – the only packaging method in CoreOS

CoreOS

CoreOS

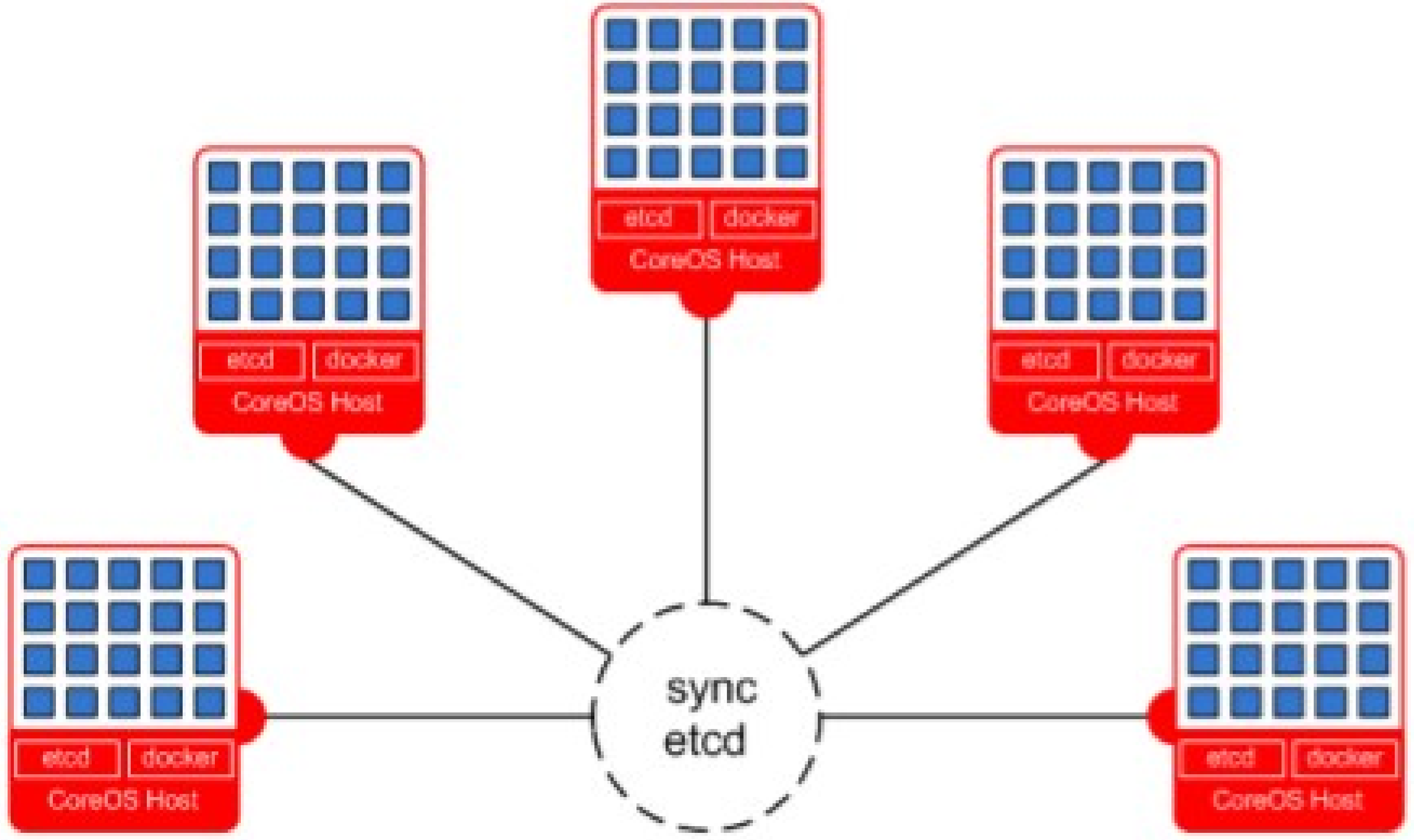
Fleet – cluster level manager & scheduler



https://coreos.com/using-coreos/clustering/

CoreOS

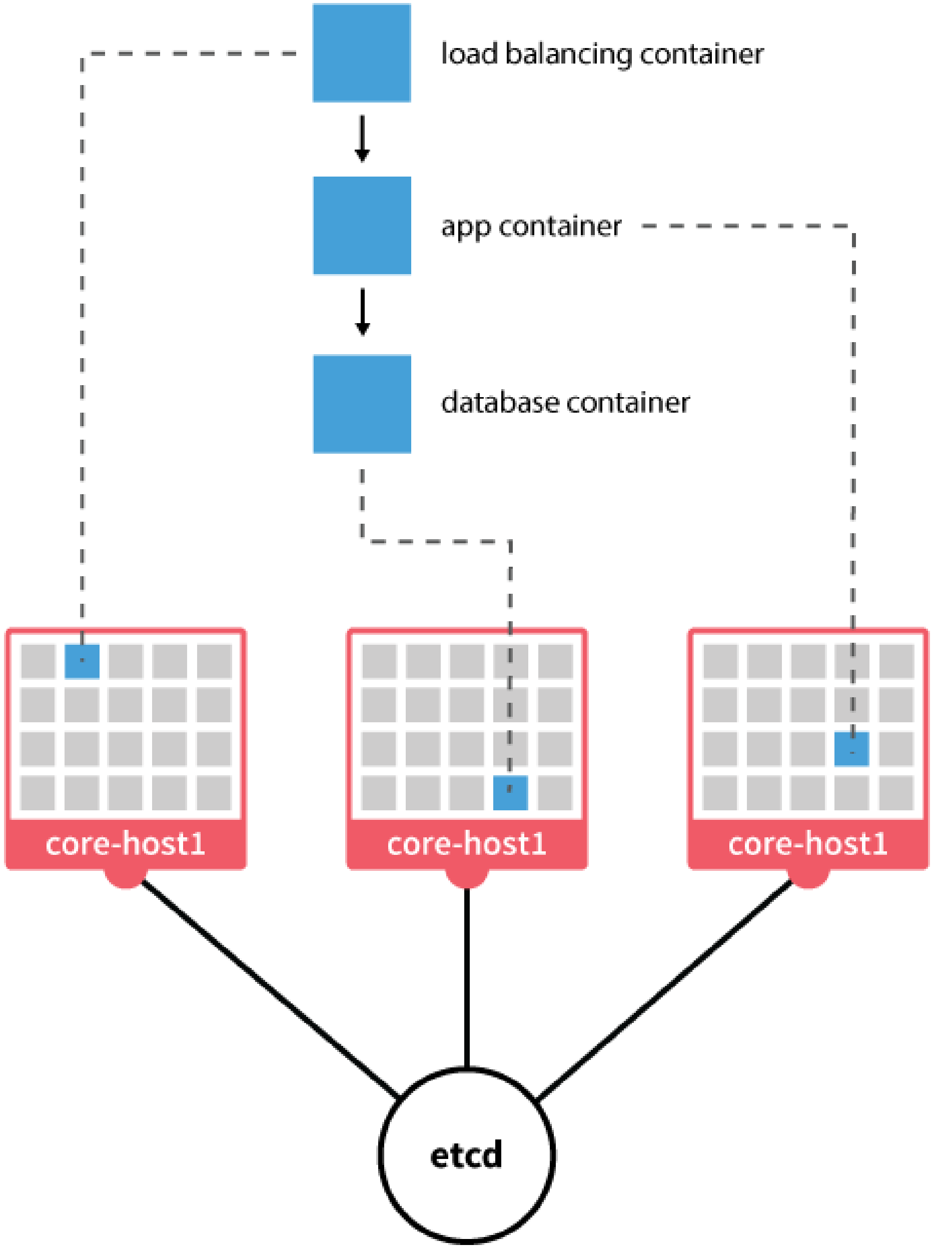
etcd – light & distributed key / value store
(used for configuration store)



https://coreos.com/docs/#cluster-management

CoreOS

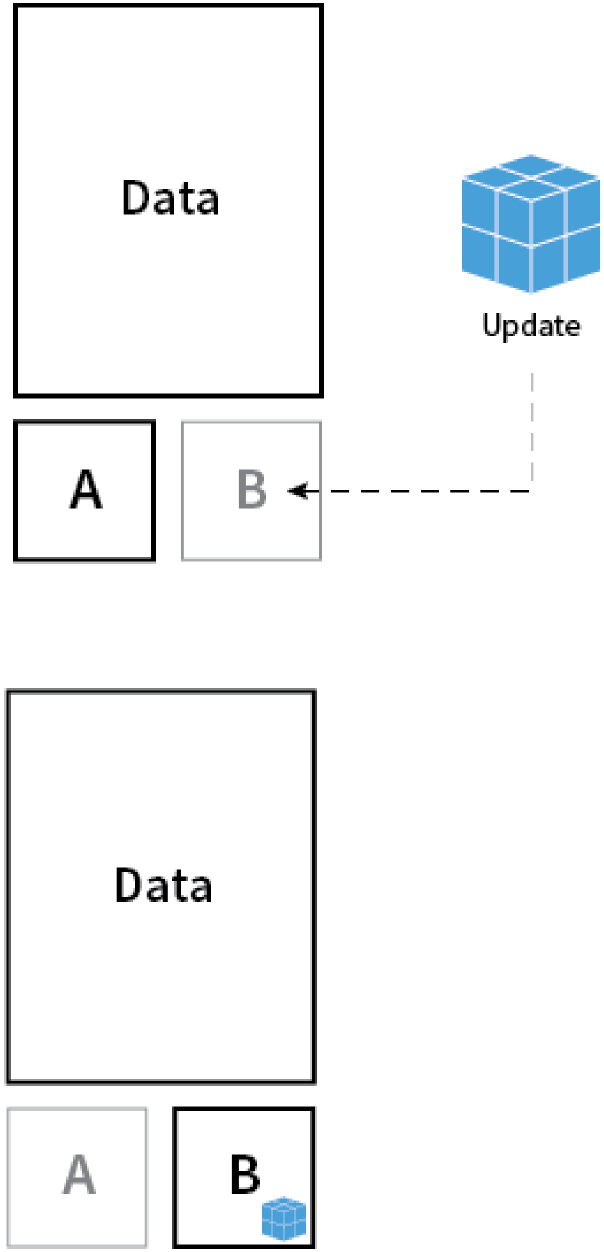
Docker – the only packaging method in CoreOS



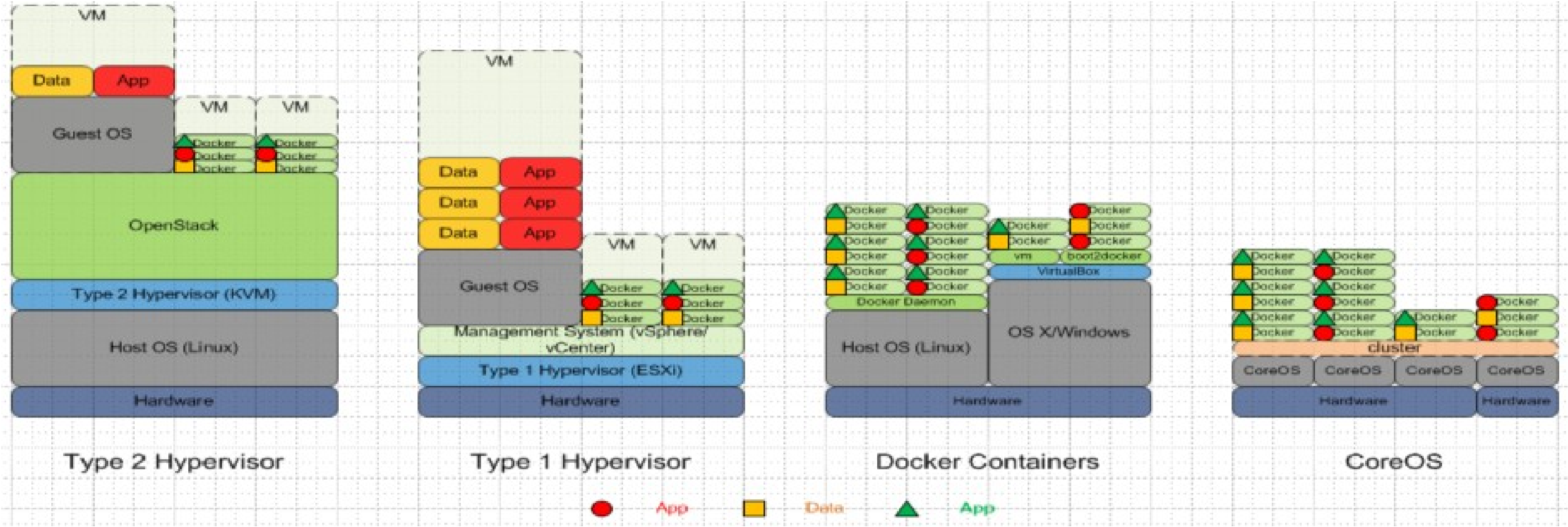
CoreOS

Full Control Over Updates:

- active–passive (dual) partitioning
- update all-packages-at-once
- safe rollback
- no more partially upgraded state
- reboot to get upgraded



CoreOS


VM
Data
App
Guest OS
VM
VM
Docker
OpenStack
Type 2 Hypervisor (KVM)
Host OS (Linux)
Hardware
Type 2 Hypervisor
VM
Data
App
Guest OS
Management System (vSphere/ vCenter)
Type 1 Hypervisor (ESXi)
Hardware
Type 1 Hypervisor
Docker
Docker Daemon
Host OS (Linux)
vm
boot2docker
VirtualBox
OS X/Windows
Hardware
Docker Containers
cluster
CoreOS
Hardware
CoreOS
App
Data
App

CoreOS

Cluster management with Fleet & SystemD

```
[Unit]
Description=My Service
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill hello
ExecStartPre=-/usr/bin/docker rm hello
ExecStartPre=/usr/bin/docker pull busybox
ExecStart=/usr/bin/docker run --name hello busybox /bin/sh -c
                         "while true; do echo Hello World; sleep 1; done"
ExecStop=/usr/bin/docker stop hello
```

CoreOS

Cluster management with Fleet & SystemD

```
$ fleetctl load hello.service
Unit hello.service loaded on 8145ebb7.../10.10.1.3
$ fleetctl start hello.service
Unit hello.service launched on 8145ebb7.../10.10.1.3

$ fleetctl list-machines
MACHINE                                 IP          METADATA
148a18ff-6e95-4cd8-92da-c9de9bb90d5a    10.10.1.1   -
491586a6-508f-4583-a71d-bfc4d146e996    10.10.1.2   -
c9de9451-6a6f-1d80-b7e6-46e996bfc4d1    10.10.1.3   -

$ fleetctl list-units
UNIT            MACHINE                 ACTIVE    SUB
hello.service   c9de9451.../10.10.1.3   active    running
```

CoreOS

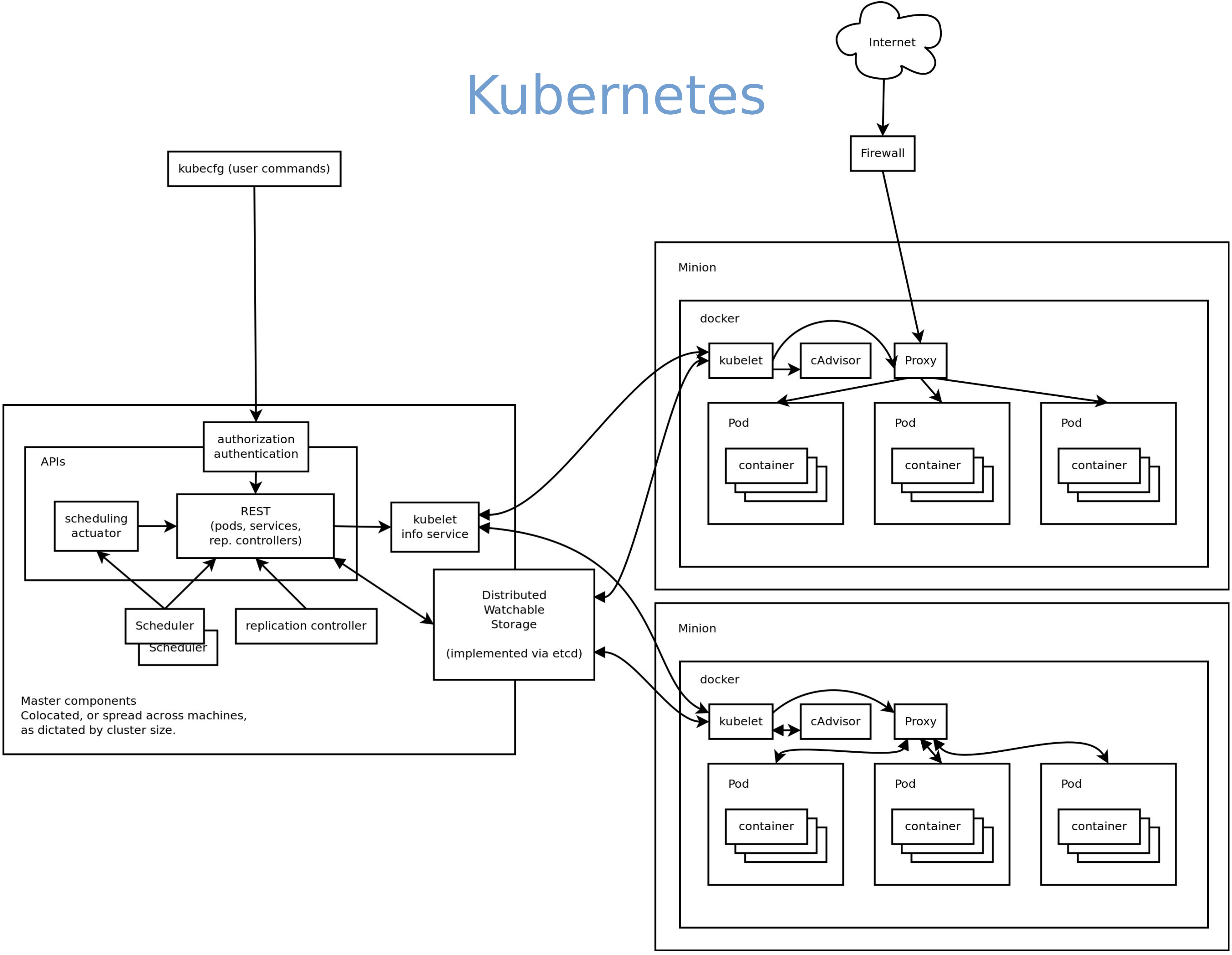
- scheduler
- HA
- dependencies

https://coreos.com/docs/launching-containers/launching/launching-containers-fleet/

Kubernetes

- https://github.com/GoogleCloudPlatform/kubernetes
- Advanced cluster manager (more than 1k hosts is a fit)
- Architecture:
    - master
    - minion
    - pod
    - replication controller
    - label

Kubernetes



Master components
Colocated, or spread across machines,
as dictated by cluster size.

Kubernetes

SmartStack

- automated service discovery and registration framework
- ideal for SOA architectures
- ideal for continuous integration & delivery
- solves “works on my machine” problem

SmartStack

- automated service discovery and registration framework
- ideal for SOA architectures
- ideal for continuous integration & delivery
- solves “works on my machine” problem

haproxy + nerve + synapse + zookeper = smartstack

SmartStack

Synapse

→ discovery service (via zookeeper or etcd)

→ installed on every node

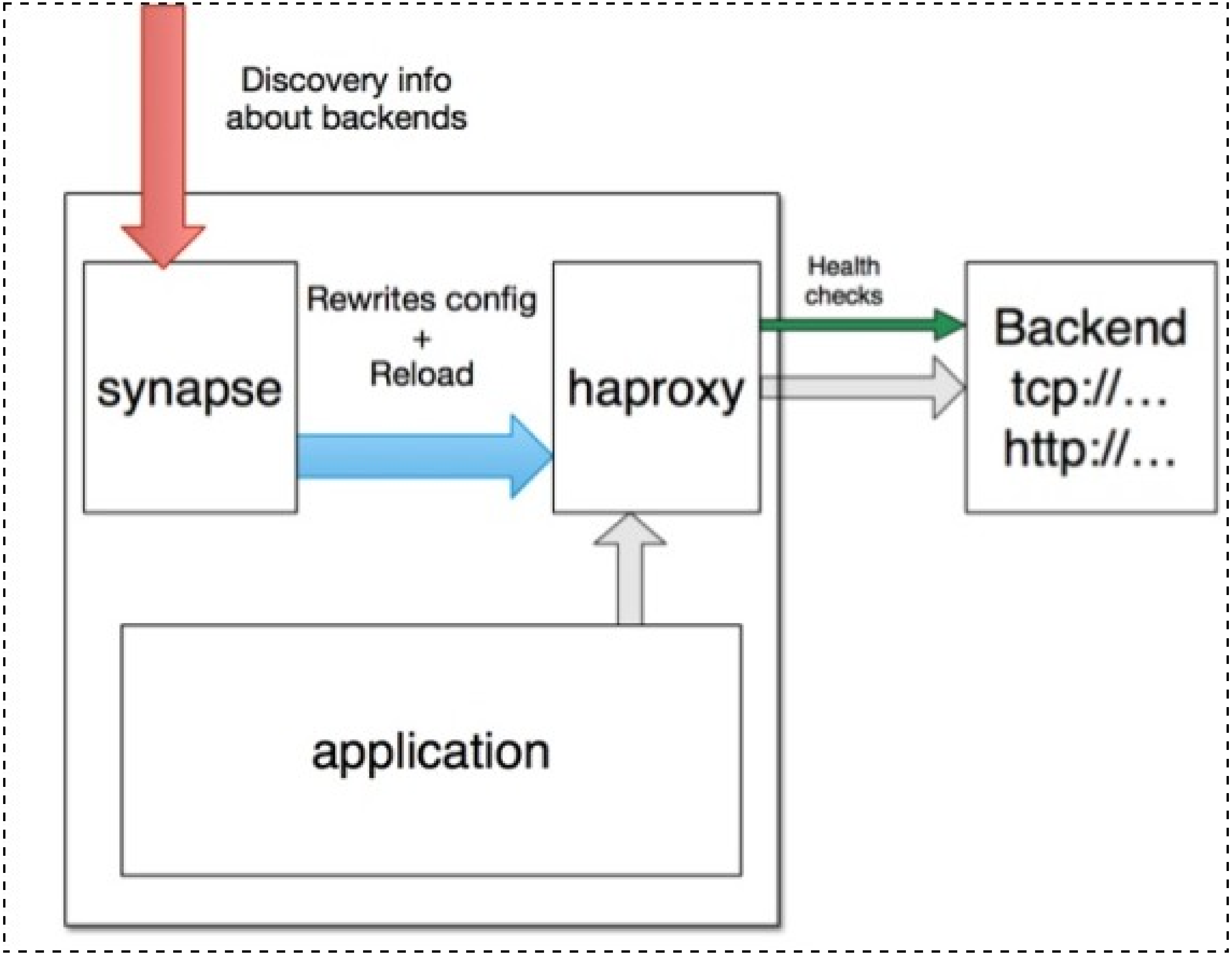
→ writes haproxy configuration

→ application doesn't have to be aware of this

→ works same on bare / VM / docker

→ https://github.com/airbnb/nerve

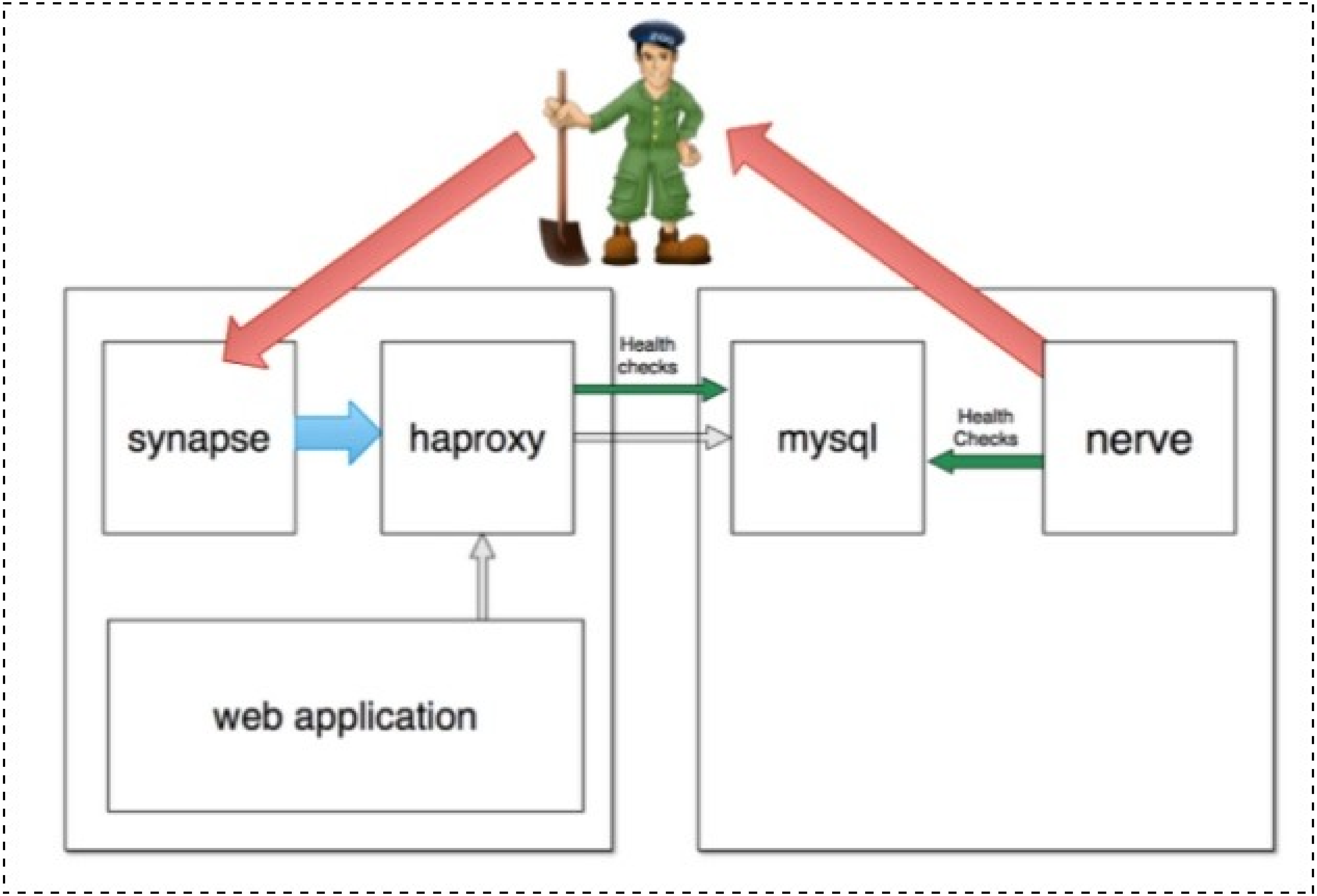
SmartStack


Discovery info
about backends
synapse
Rewrites config
+
Reload
haproxy
Health
checks
Backend
tcp://...
http://...
application

SmartStack

Nerve
→ health checks (pluggable)
→ register service info to zookeper (or etcd)
→ https://github.com/airbnb/synapse

SmartStack


Health
checks
synapse
haproxy
mysql
Health
Checks
nerve
web application

SmartStack

SmartStack

Smartstack + Docker = <3

SmartStack

Smartstack + Docker = <3

but also remember about Consul
(come to #dockerkrk 2 meetup!)

Summary

Wanna learn Docker?

http://dockerbook.com/

Summary

Freenode #docker
#KrkDocker meetups (http://www.meetup.com/Docker-Krakow-Poland/)
https://github.com/docker/docker

sources?

→ docker.io documentation

→ dockerbook.com

→ slideshare!

→ zounds of blogposts (urls provided)

→ and some experience ;)

Thank you :)

Orchestrating Docker containers at scale

Maciej Lasyk

12 Sesja Linuksowa

2015-04-18, Wrocław

http://maciej.lasyk.info

maciej@lasyk.info

@docent-net