WELCOME TO DAY 2

MINING THE DOCKER FILE SYSTEM

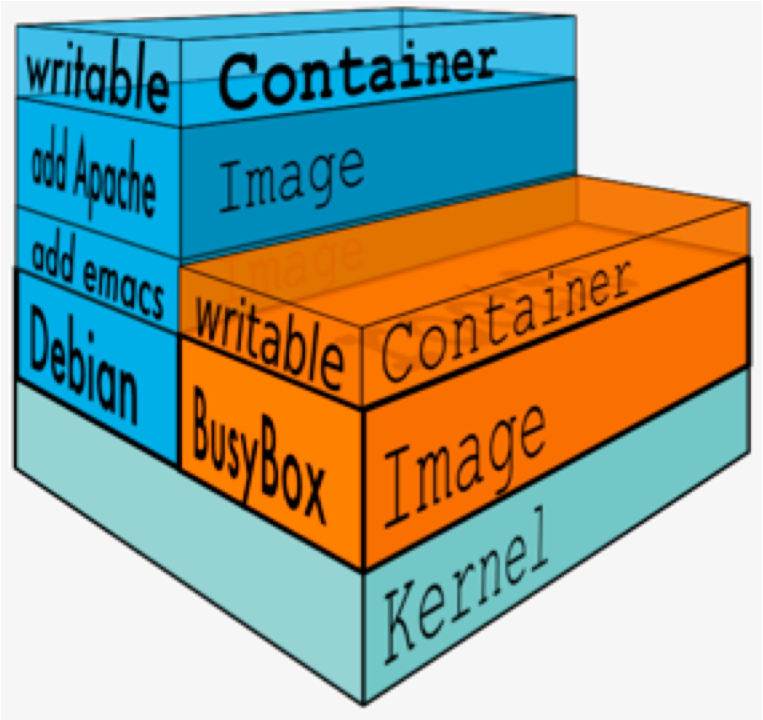
module 04

Section 4.1 | remember this!

nebulaworks

docker's first technical drawing

- visualize the layers of a docker container image
- distinguish how a container layers on the image
- depict possible actions at each layer



Images

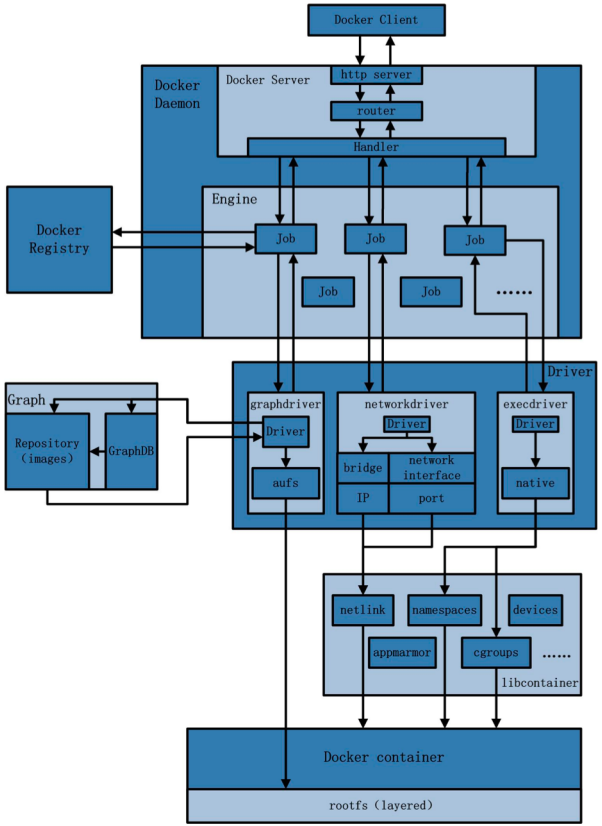
Images are read-only, how do we change it?

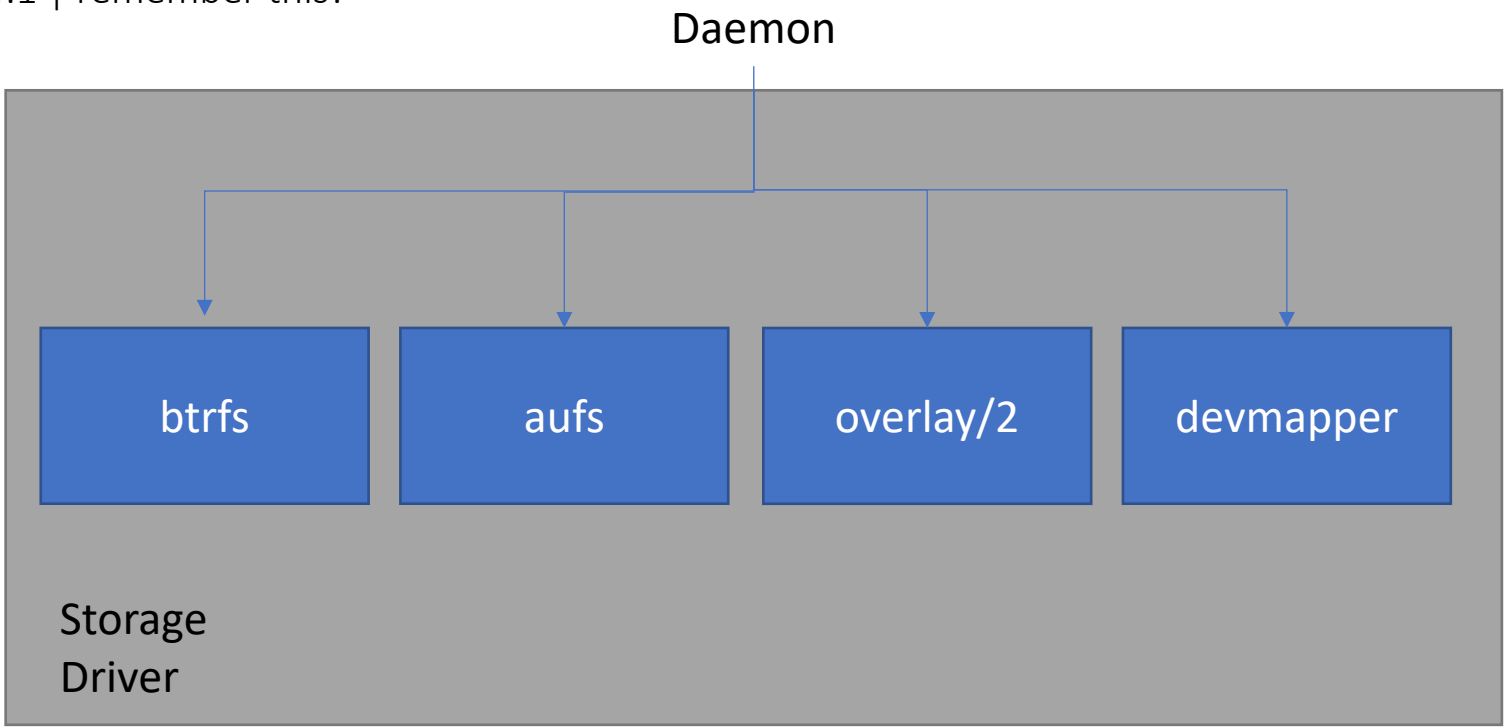
- We create a new container from that image.
- Then we make changes to that container.
- When we are satisfied with those changes, we transform them into a new layer.
- A new image is created by stacking the new layer on top of the old image.

the graph driver

- manages how containers and container images are stored
- independent from volume mount storage
- default == `/var/lib/docker/`

Section 4.1 | remember this!

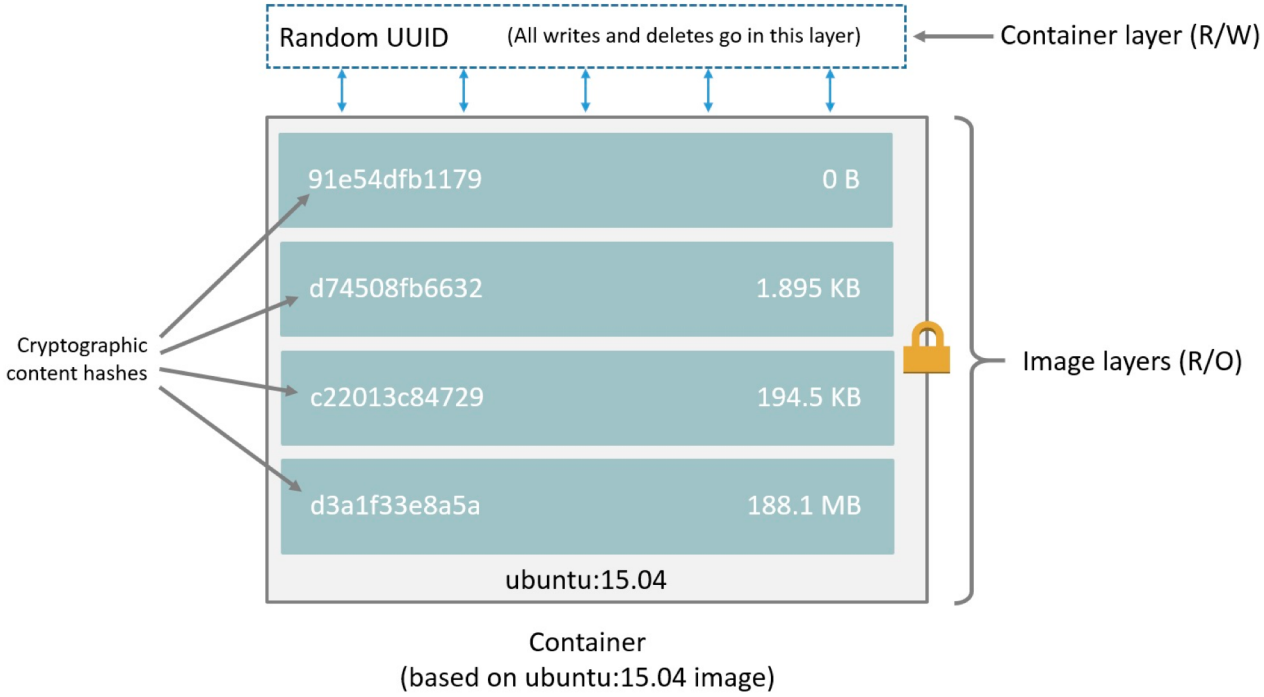



Daemon
btrfs
aufs
overlay/2
devmapper
Storage
Driver

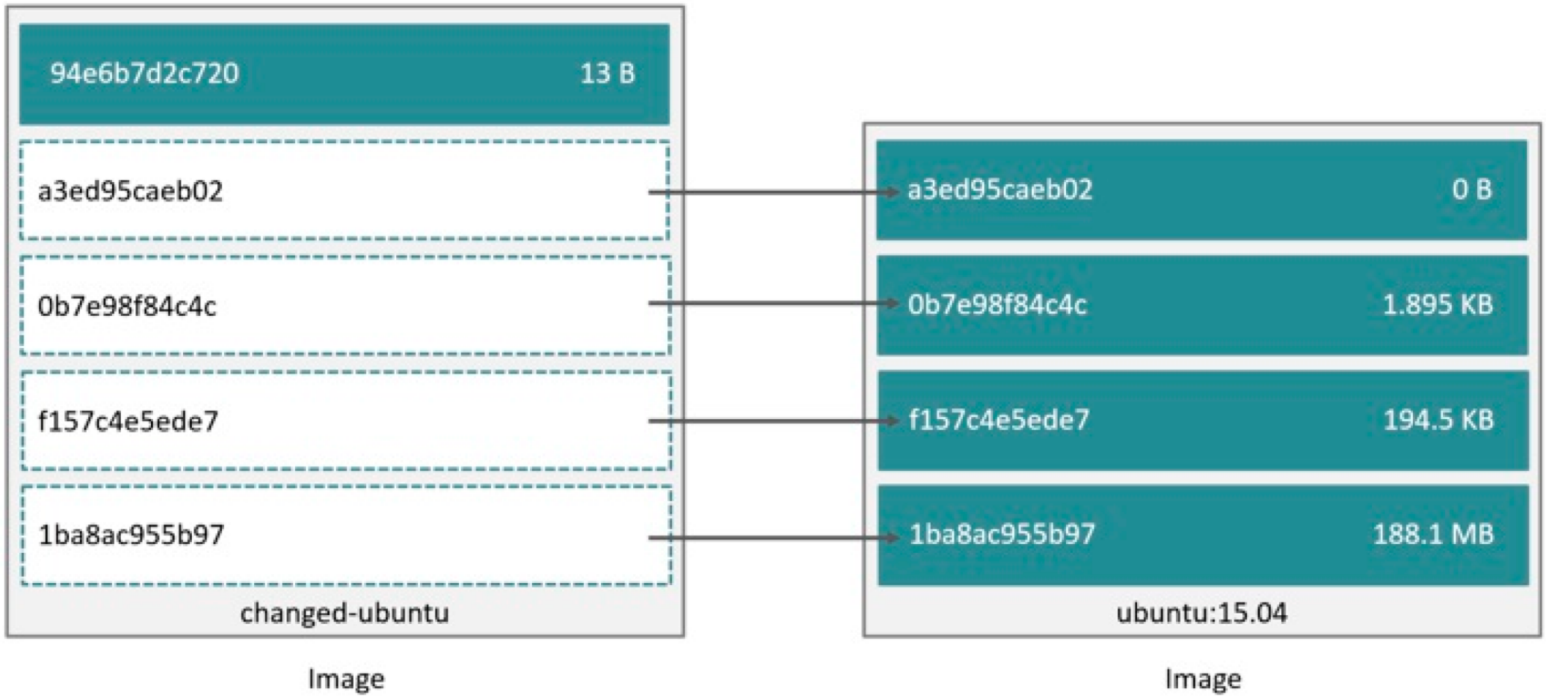
Section 4.2 | images

nebulaworks

content addressable storage

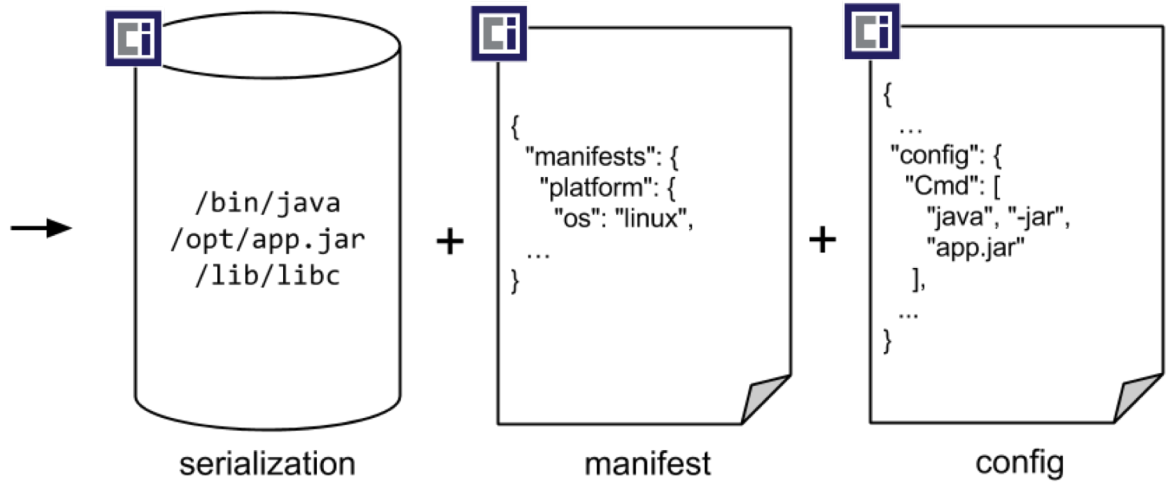


content addressable storage



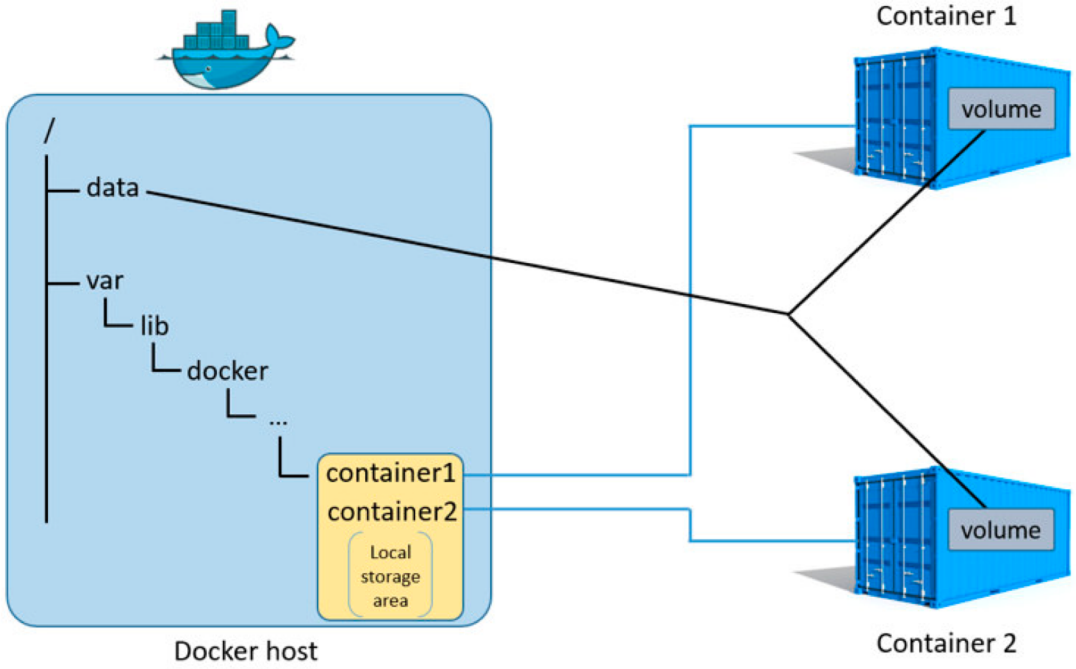
oci image spec

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, World");
  }
}
```

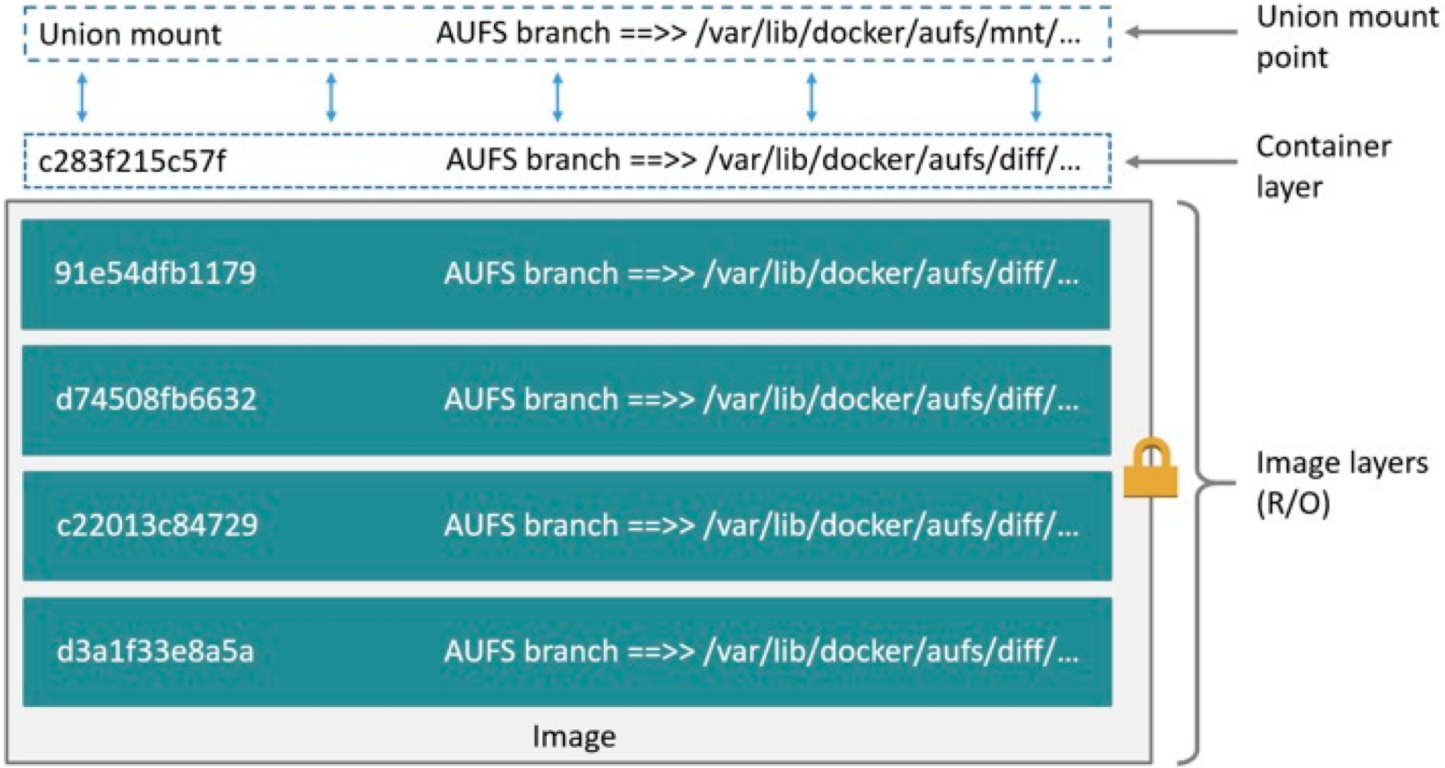


Section 4.3 | storage drivers

graph directory



aufs

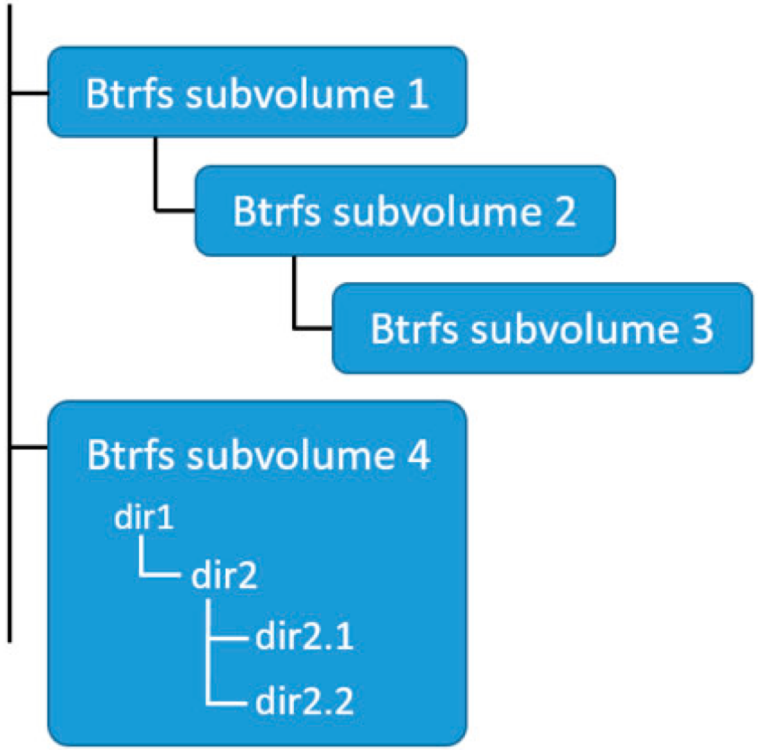

Union mount
AUFS branch ==>> /var/lib/docker/aufs/mnt/...
Union mount point
c283f215c57f
AUFS branch ==>> /var/lib/docker/aufs/diff/...
Container layer
91e54dfb1179
AUFS branch ==>> /var/lib/docker/aufs/diff/...
d74508fb6632
AUFS branch ==>> /var/lib/docker/aufs/diff/...
c22013c84729
AUFS branch ==>> /var/lib/docker/aufs/diff/...
d3a1f33e8a5a
AUFS branch ==>> /var/lib/docker/aufs/diff/...
Image
Image layers (R/O)

aufs

- Container layer read scenarios
  - File does not exist in container
  - File only exists in the container layer
  - Exists in both

- Write Scenarios
  - Writing to file for the first time (copy up)
  - Deleting files (white files, opaque file)
  - Renaming Directories (not supported)

btrfs


Btrfs subvolume 1
Btrfs subvolume 2
Btrfs subvolume 3
Btrfs subvolume 4
dir1
dir2
dir2.1
dir2.2

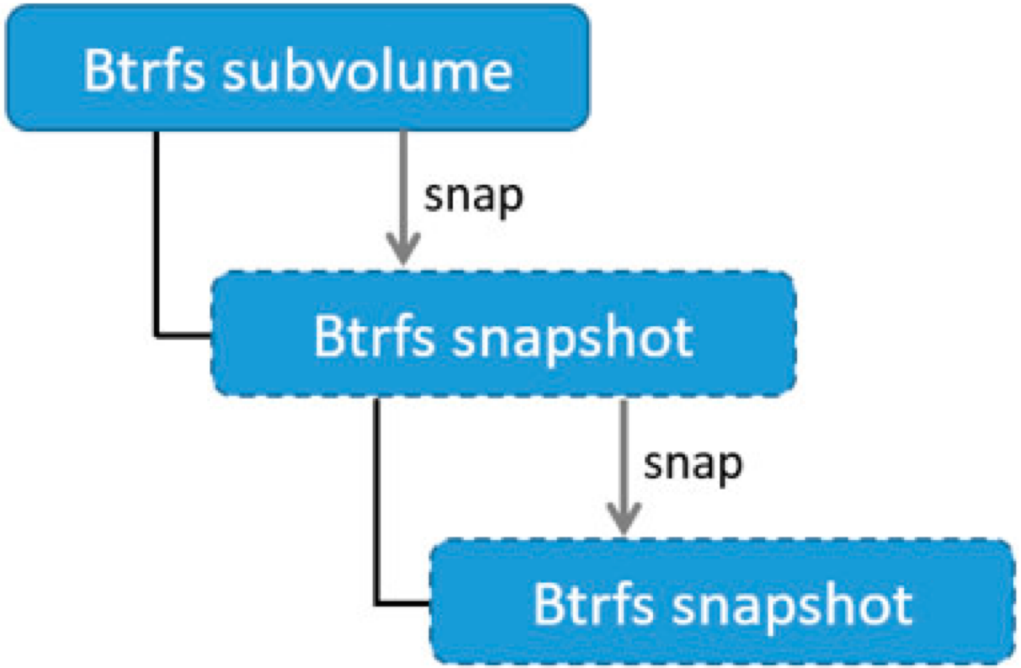
btrfs

- Block level ops, thin provisioning, copy on write
- Supported on Docker CE on Ubuntu, Docker EE/CS on SLES
- Requires dedicated storage, btrfs mount at /var/lib/docker
- Requires a “btrfs” tool for creating mounts
- Configure daemon.json

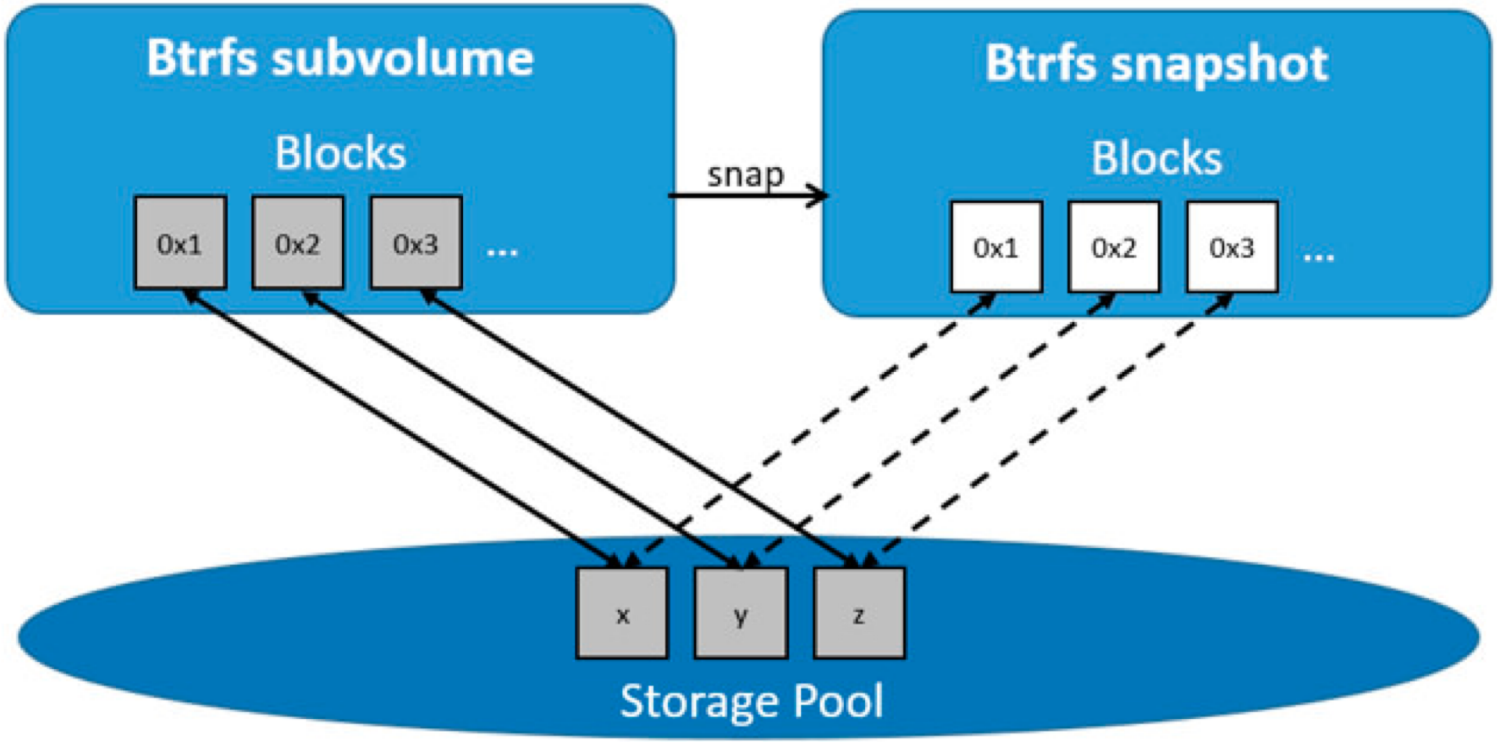
Btrfs (continued…)

- When using balanced mode with BTRFS, 1 GB chunks will be added when storage pressures occur
- The /subvolumes/ dir
  - One directory per image layer
  - Union filesystem built from a layer plus all its parent layers
  - Can be nested and snapshotted if necessary

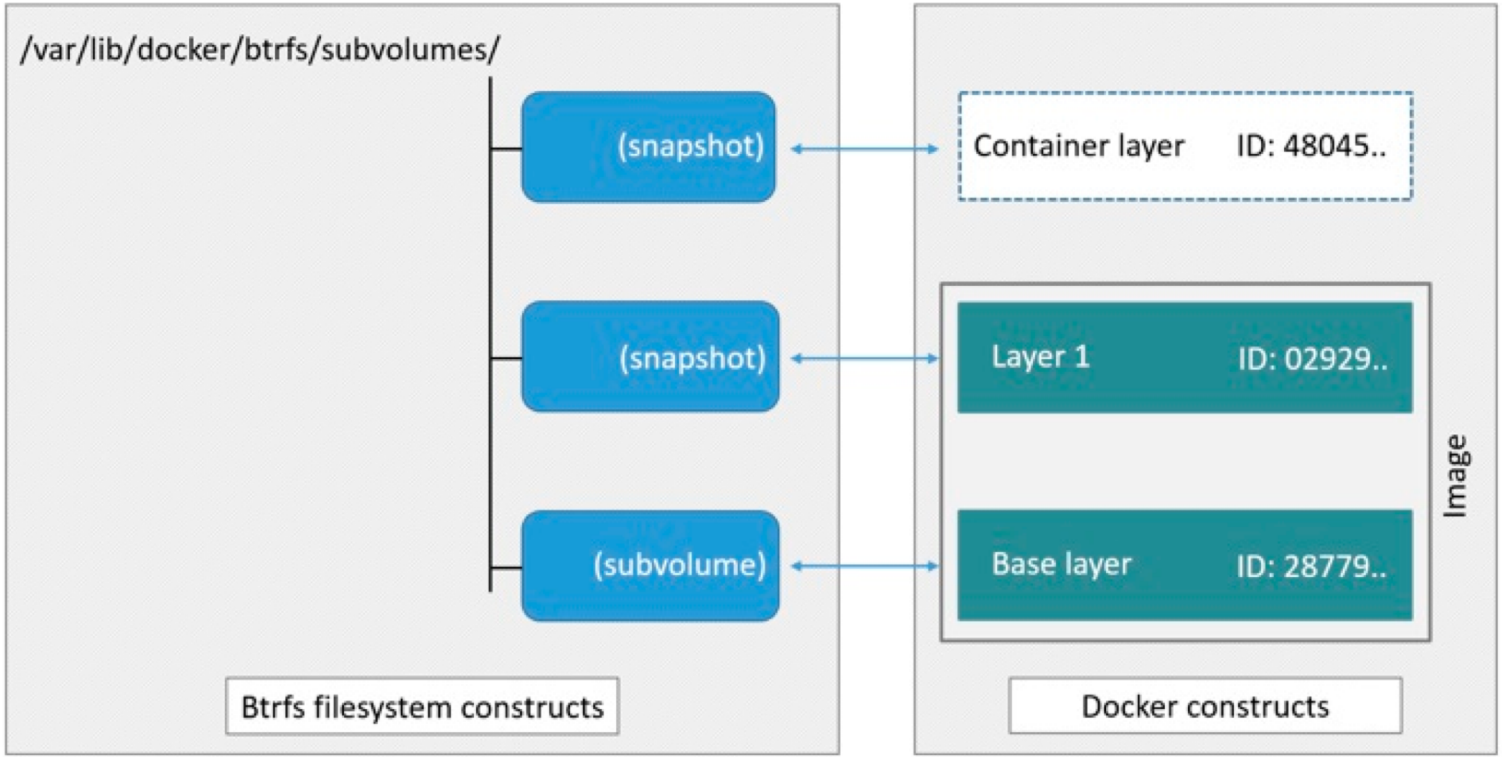
btrfs



btrfs


Btrfs subvolume
Blocks
0x1
0x2
0x3
...
snap
Btrfs snapshot
Blocks
0x1
0x2
0x3
...
x
y
z
Storage Pool

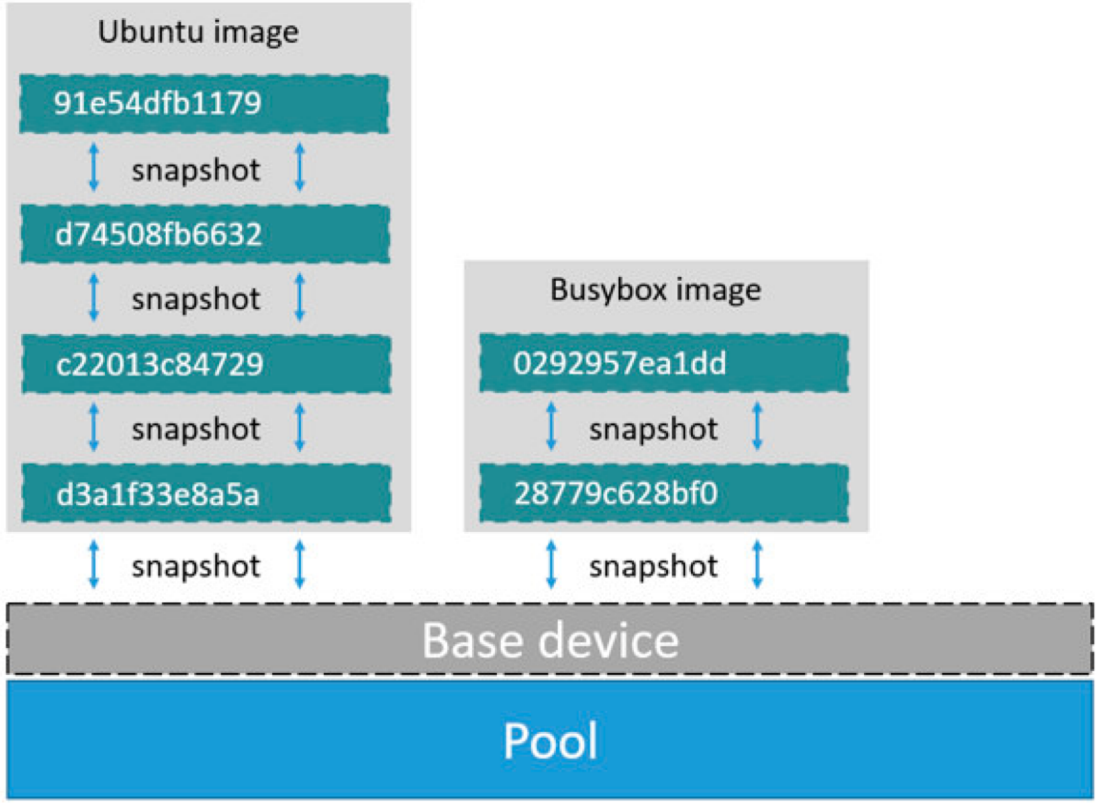
btrfs


/var/lib/docker/btrfs/subvolumes/
(snapshot)
(snapshot)
(subvolume)
Btrfs filesystem constructs
Container layer ID: 48045..
Layer 1 ID: 02929..
Base layer ID: 28779..
Image
Docker constructs

btrfs

- Reads
  - A container is a space-efficient snapshot of an image
  - Metadata blocks in the storage pool point to actual blocks
- Writes
  - Write a new file allocates new block in the containers snapshot
  - Same as writing to a subvolume
- Write to Existing file
  - Updateing an existing file in a container is a COW op
  - The oringal data is read from the layer where the file exists, and only the modified blocks are written to top layer
  - Performing many writes on small files may slow performance

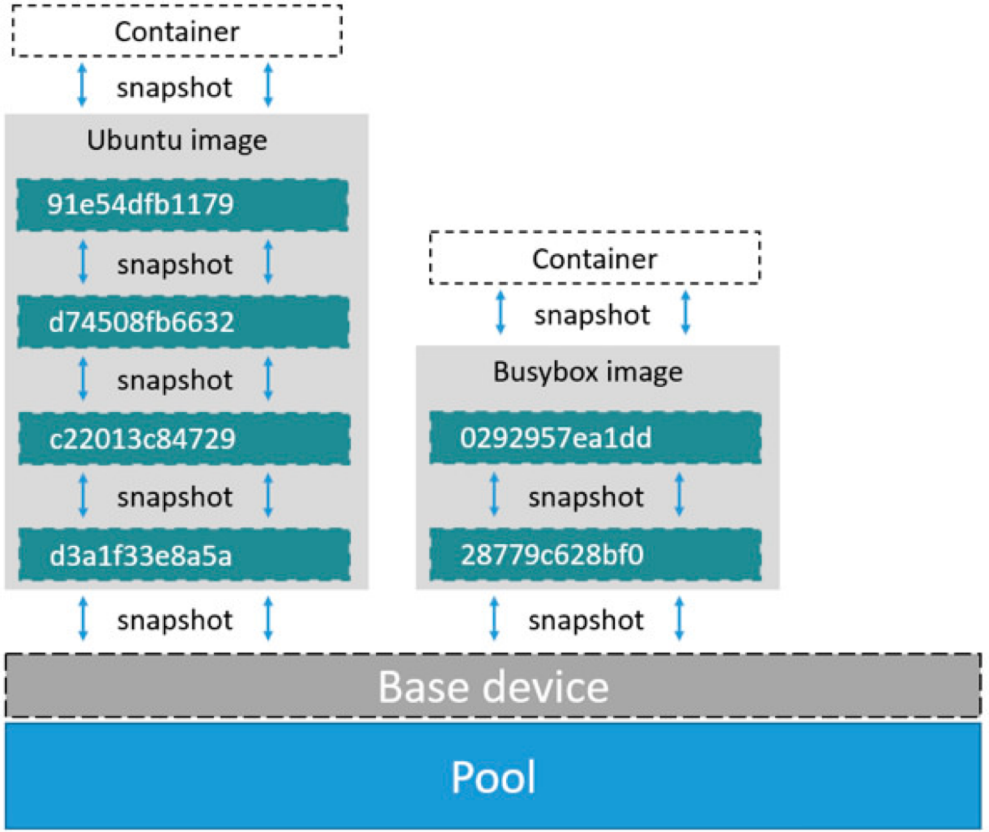
dev mapper


Ubuntu image
91e54dfb1179
snapshot
d74508fb6632
snapshot
c22013c84729
snapshot
d3a1f33e8a5a
snapshot
Busybox image
0292957ea1dd
snapshot
28779c628bf0
snapshot
Base device
Pool

dev mapper

- /var/lib/devicemapper/mnt/ containers metadata about the dev mapper config
- /var/lib/docker/mnt
    - Containers a mountpoint for each image and layer that exists

dev mapper

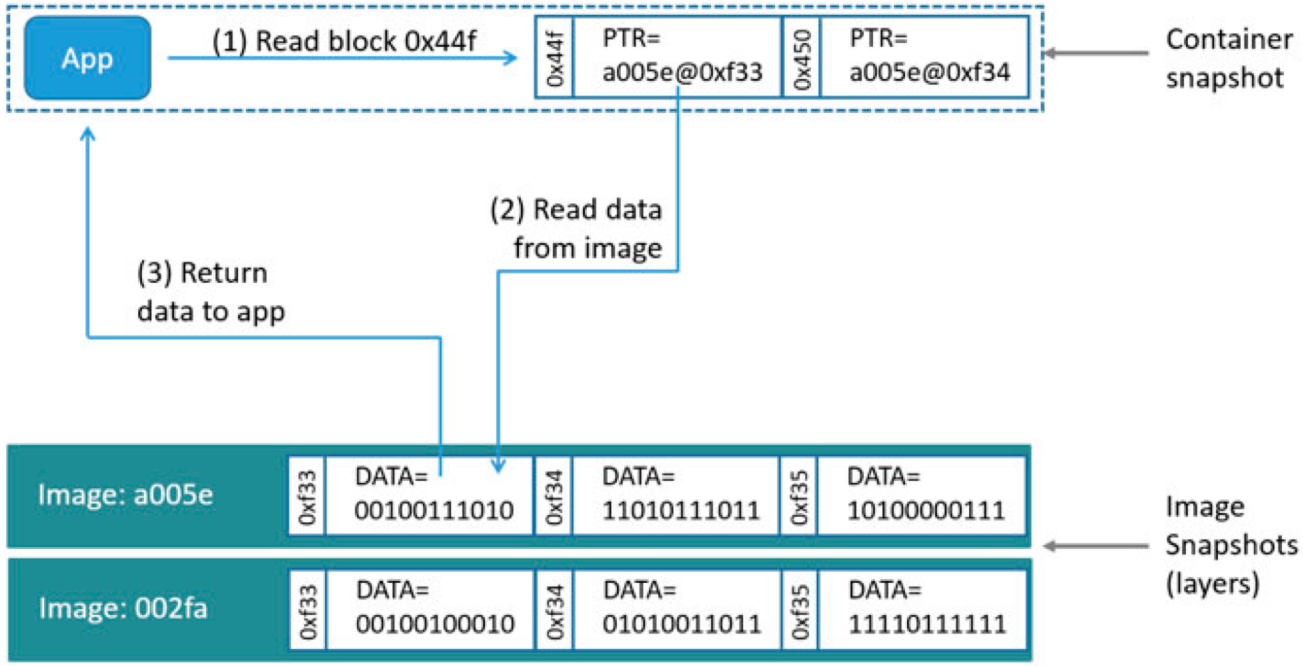

Container
snapshot
Ubuntu image
91e54dfb1179
snapshot
d74508fb6632
snapshot
c22013c84729
snapshot
d3a1f33e8a5a
snapshot
Container
snapshot
Busybox image
0292957ea1dd
snapshot
28779c628bf0
snapshot
Base device
Pool

dev mapper

- Writing File Scenarios
  - Writing a new file uses allocate-on-demand op
  - Each block of the new file is added in the containers writable layer

- Update a file
  - Block of the file is read from nearest layer where it exists, only modified blocks are written to writable layer
- Delete a file
  - Devicemapper will intercepts all reads to this resource and will through directory does not exist

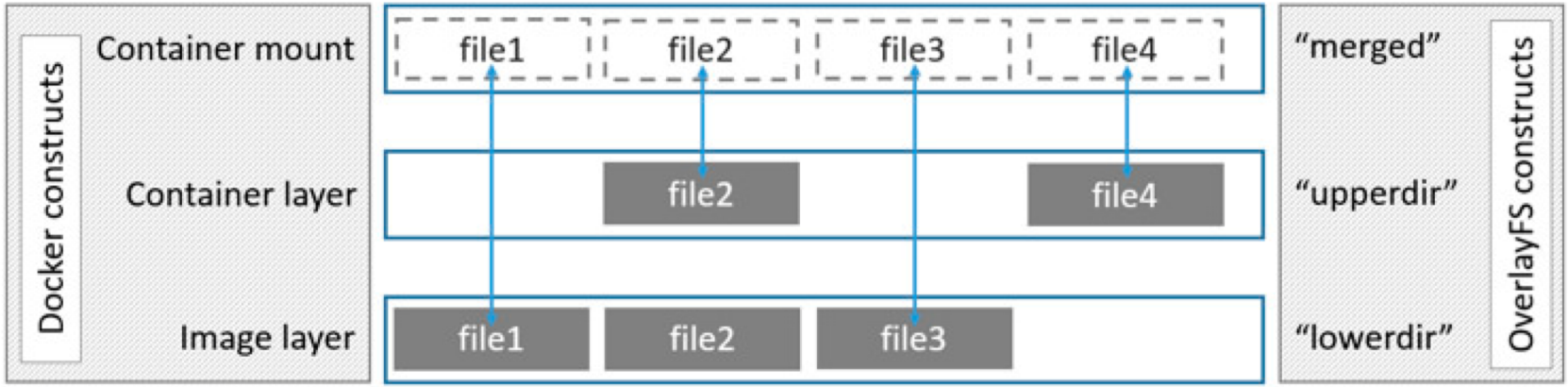
dev mapper


App
(1) Read block 0x44f
0x44f
PTR= a005e@0xf33
0x450
PTR= a005e@0xf34
Container snapshot
(2) Read data from image
(3) Return data to app
Image: a005e
0xf33
DATA= 00100111010
0xf34
DATA= 11010111011
0xf35
DATA= 10100000111
Image: 002fa
0xf33
DATA= 00100100010
0xf34
DATA= 01010011011
0xf35
DATA= 11110111111
Image Snapshots (layers)

Overlay and overlay 2

- Recommend to use overlay2, more efficient
- Overlay 2 supported on Docker EE/CE
- Needs Kernel 4.0 or higher
- Recommended to use a separate backing filesystem form the one used by /var/lib/docker
- Overlay2 has up to 128 lower level layers, while overlay supported 2

overlayfs


Docker constructs
Container mount
Container layer
Image layer
file1
file2
file3
file4
file2
file4
file1
file2
file3
"merged"
"upperdir"
"lowerdir"
OverlayFS constructs

overlay2

- Use overlay2
- /var/lib/docker/overlay2/l
  - Lowercase L
  - Layer identifiers symbolic links

- Lowest layer is called a link, that contains a name of the shortened identifier
  - Contains diff directory as well
- Second lowest layer and all above it have “lower ” directory
  - Lower denotes a parent, and diff contains the layer contents

AUFS	stable	production-ready	good memory use	smooth Docker experience	high write activity	PaaS-type work
Devicemapper (loop)	stable	in mainline kernel	smooth Docker experience	production	performance	lab testing
Devicemapper (direct-lvm)	stable	production-ready	in mainline kernel	smooth Docker experience	PaaS-type work	
Btrfs	in mainline kernel	high write activity	container churn	build pools		
Overlay	stable	good memory use	in mainline kernel	container churn	lab testing	
ZFS native (ZoL)	PaaS-type work					
ZFS FUSE	stable	lab testing	production			

Overlay2

Key

Has attribute	attribute
If good for use case	use case
If bad for use case	use case

Section 4.5 | volume plugins

The problem

- We can store data inside of containers BUT we usually don't want to.
  - Containers are ephemeral
  - Storage drivers aren't very efficient.

How the heck do I persist data?

The problem

- We can store data inside of containers BUT we usually don't want to.
  - Containers are ephemeral
  - Storage drivers aren't very efficient.

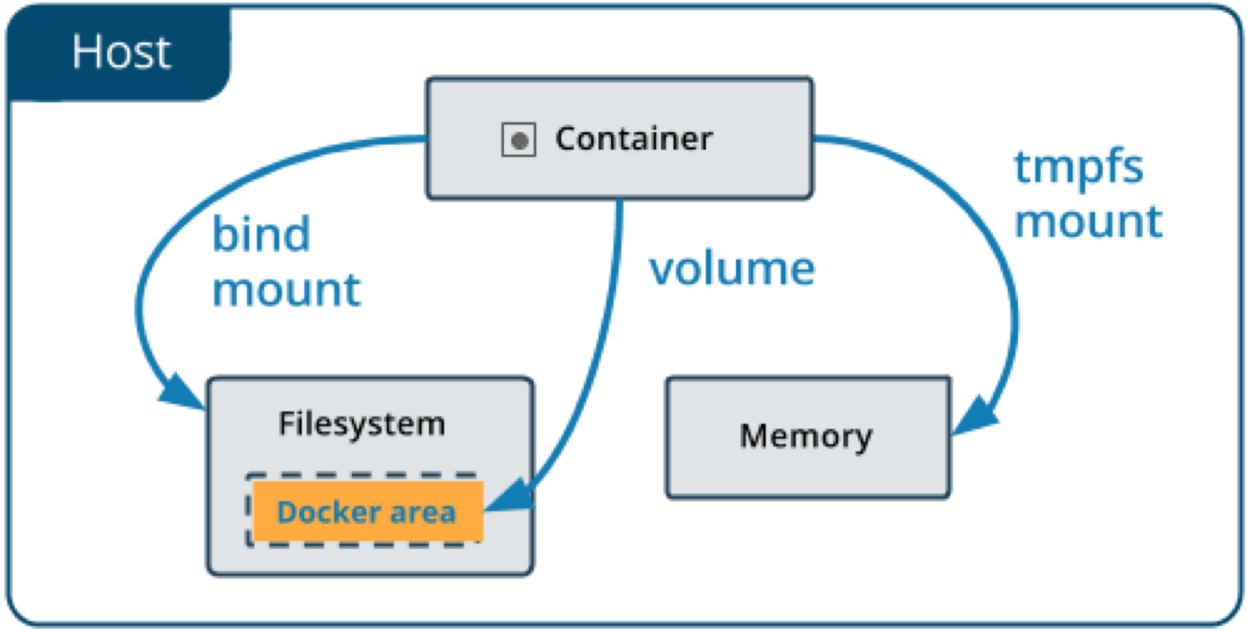
How the heck do I persist data?

What are volumes?

- Volumes are a place to store data that live outside the lifecycle of a container.
- They are mounted into a container
- First class citizens in Docker
- Work with Linux and Windows containers.
- They don’t add to a containers size!
- Can be created and managed (by docker) outside the scope of any container

```
/var/lib/docker/volumes/<volume_name>
```

Volumes



Source : docker documentation

managing volumes with the docker volume object

manages persistent data

- local copy by default
- stored in the graph dir
- accessed by labels
- created with a plugin architecture

The bad

- Docker ships with a default volume driver that supports local, host-based volumes.
- The default volume driver may not be suitable for certain workloads (data-intensive workloads).
- It won't cut it for distributed stateful applications.

Lab 3

nebulaworks

Volume Plugins A closer look

The Basics

- Docker Engine Plugins allow you to extend the capabilities of the Docker Engine.
- Extends the local volume driver
- Allow you to interface with external storage devices / platforms.

Volume Plugin Architecture
The Docker Plugin Specification

- Standardized API interface (9 endpoints)
- Plugin packaging (build as an image)
- Plugin storage (hub/store)
- Plugin Management (installation, configuration, deletion)
    - https://docs.docker.com/engine/extend/plugin_api/ - plugin-lifecycle

managing volumes

docker volume command (under the hood)

- `volume create`
  - `/VolumeDriver.Get` (checks to see if it exists)
  - `/volumeDriver.Create`
- `volume inspect`
  - `VolumeDriver.Get`
- `volume ls`
  - `/VolumeDriver.List`
- `volume rm`
  - `/VolumeDriver.Remove`

Standardized API Interface

- /Plugin.Activate
- /VolumeDriver.Create
- /VolumeDriver.Remove
- /VolumeDriver.Mount
- /VolumeDriver.Path
- /VolumeDriver.Unmount
- /VolumeDriver.Get
- /VolumeDriver.List
- /VolumeDriver.Capabilities

/VolumeDriver.Create

- Request body: {“Name”: “volume_name”, “opts”:{}}
- Response : {“Err”:””}
- Notes : This should only create the volume (no mounting)

- This endpoint gives our plugin the ability to create a volume
- Docker CLI - > Docker Daemon -> Plugin API.
- The plugin must be able to respond with the required response.

Info on the plugin api :

- https://docs.docker.com/engine/extend/plugins_volume/ - command-line-changes

Managing a Plugin Lifecycle

first class citizens : `docker plugin <cmds>`

```
plugin create
plugin inspect
plugin ls
plugin rm
```

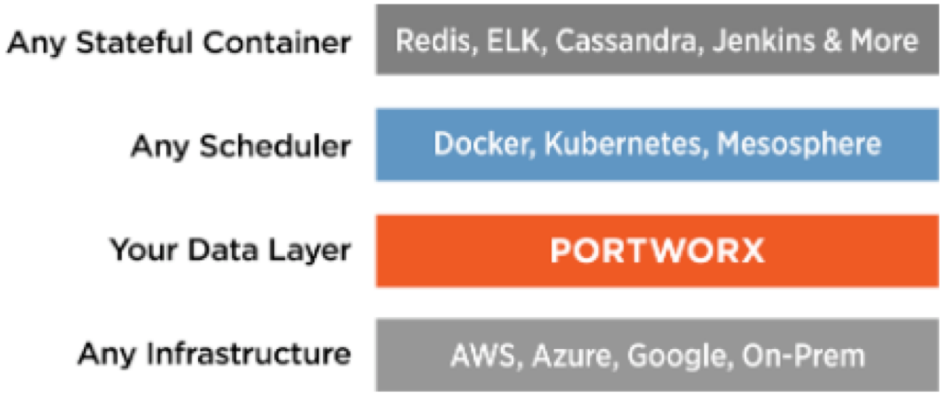
- Established in 2014 by Goutham Rao and friends
- Container data services platform
- Software-defined storage

portworx

portworx

Providing a “Cloud Agnostic” Storage Solution

- Portworx can run on any infrastructure.
- It will aggregate the host storage resources into storage classes and allow them to be consumed as docker volumes by application containers.



portworx

MESOS

portworx

Use Cases

- Databases : compatible with a lot of dbs
- CI/CD : Testing prod env & the golden image
- Content Management : Horizontal scale of CMS sites
- Docker storage : Persistent Container storage across a swarm
- Kubernetes storage

portworx

Deploying portworx with Docker

Prerequisites:

- You must have a storage device that you can allocate to portworx
  - Non-root storage devices.

```
lsbk
```

- You must configure Docker to allow shared mounts propagations.

```
$ docker run -it -v /mnt:/mnt:shared busybox sh -c /bin/date
$ sudo mount –make-shared
```

- A minimum version of Docker 1.10 must be installed.
- runC must be installed

```
$ sudo apt-get install runC
```

Deploying portworx with Docker

Install PX OCI bundle

- Portworx gives us a Docker based installation utility that helps deplpy the PX OCI bundle. You can install the bundle by running the following Docker container:
- Curl for the latest version of portworx

```
$ latest_stable=$(curl -fsSL
'http://install.portworx.com:8080?type=dock&stork=false' | awk '/image: /
{print $2}')
```

- Download the OCI bits

```
$ sudo docker run --entrypoint /runc-entry-point.sh \ --rm -i --
privileged=true \ -v /opt/pwx:/opt/pwx -v /etc/pwx:/etc/pwx \
$latest_stable
```

Deploying portworx with Docker

Configure PX under runC

- Once the PX OCI bundle is installed, you can run the following command from the bundle to configure system to start PX runC.

```
$ sudo /opt/pwx/bin/px-runc install -c MY_CLUSTER_ID \ -k
etcd://myetc.company.com:2379 \ -s /dev/xvdb
```

Deploying portworx with Docker

Starting PX runC

- Reload systemd configurations and enable and start the Portworx service

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable portworx
$ sudo systemctl start portworx
```

- Now that portworx is installed we can create a stateful application with Docker Swarm!

Deploying a Jenkins Service with the portworx driver for Docker

Create the Volume

```
$ docker volume create -d pxd --name jenkins_vol --opt size=4 \
  --opt block_size=64 --opt repl=3 --opt fs=ext4 --opt shared=true
```

Create the Service

```
$ docker service create --name jenkins \
 --replicas 3 \
 --publish 8082:8080 \
 --publish 50000:50000 \
 -e JENKINS_OPTS="--prefix=/jenkins" \ --reserve-memory 300m \
 --mount "type=volume,volume-driver=pxd,source=jenkins_vol,target=/var/jenkins_home”\
```

volume plugin – single platform

plugin	description
Blockbridge Volume plugin	A volume plugin that provides access to an extensible set of container-based persistent storage options. It supports single and multi-host Docker environments with features that include tenant isolation, automated provisioning, encryption, secure deletion, snapshots and QoS.
Contiv Volume Plugin	An open source volume plugin that provides multi-tenant, persistent, distributed storage with intent based consumption using ceph underneath.

plugin	description
Flocker plugin (last source code commit 1 year)	A volume plugin that provides multi-host portable volumes for Docker, allowing you to run databases and other stateful containers and move them across a cluster of machines.
gce-docker plugin (last source code commit 2 years)	A volume plugin able to attach, format and mount Google Compute persistent-disks.
GlusterFS plugin (last source code commit 2 years)	A volume plugin that provides multi-host volumes management for Docker using GlusterFS.
IPFS Volume Plugin (last source code commit 1 year)	An open source volume plugin that allows using an ipfs filesystem as a volume.

volume plugin

plugin	description
Convoy plugin (Last source code update 1 year)	A volume plugin for a variety of storage back-ends including device mapper and NFS. It's a simple standalone executable written in Go and provides the framework to support vendor-specific extensions such as snapshots, backups and restore.
Horcrux Volume Plugin	A volume plugin that allows on-demand, version controlled access to your data. Horcrux is an open-source plugin, written in Go, and supports SCP, Minio and Amazon S3.
Netshare plugin	A volume plugin that provides volume management for NFS[34], AWS EFS and CIFS file systems.
Quobyte Volume Plugin	A volume plugin that connects Docker to Quobyte's data center file system, a general-purpose scalable and fault-tolerant storage platform.

volume plugin – multi-platform

plugin	description
OpenStorage Plugin	It is a clustered implementation of the Open Storage specification which relies on the docker runtime. It allows you to run stateful services in Linux Containers in a multi-host environment. It plugs into CSI and Docker volumes in order to provide storage to a container. It also plugs into Kubernetes and Mesosphere.
REX-Ray plugin	A volume plugin that provides a vendor agnostic storage orchestration engine. The primary aim is to provide persistent storage for Mesos, Docker, and Kubernetes.

volume plugin

plugin	description
Local Persist Plugin (Last source code commit 1 year)	A volume plugin that extends the default **local**driver's functionality by allowing you specify a mountpoint anywhere on the host, which enables the files to *always persist*, even if the volume is removed via **docker volume rm**.

Hardening of Container Infrastructure

Module 05

Section 5.1 | Hardening the Docker host

First: Determine how "hard" you need to be, and this will help determine which guidelines you should follow.

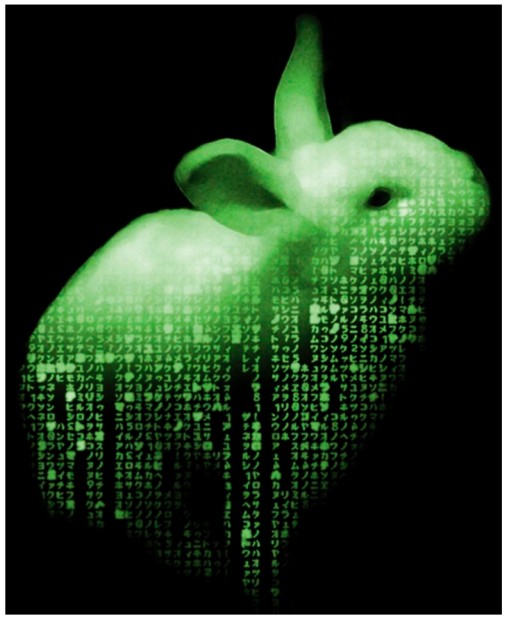
- Is this a hobby/trial/learning machine? Lab machine? Business-critical? Military-grade?
- What policies exist: corporate/organizational policy may have specific recommendations or requirements

the CIS (Center for Internet Security) benchmark reference

docker's security bible

- Full CIS reference doc here.


Follow the white rabbit. . .



is security a function of the >

>>>host

or

>>> process

Tip: Put Docker filesystem into its own partition

A recommended practice is to create a separate partition for /var/lib/docker, and mount this (in /etc/fstab), so that Docker doesn't wedge the entire system, should things grow out of control suddenly (or just slowly grow, unobserved, over time.)

Building Secure Containers

- Verify authenticity of the build
- Assure public images are signed and have valid checksums
- Use HTTPS for remote file uploads
- Pin to specific files
- Always use version images
  - E.g ubuntu:14.04, vs Ubuntu

Lightweight Images

- Favor lightweight images Alpine is a light weight distro
- Build minimum images to run apps
- Do not run SSH in a container

Docker Security Scanning

- Provides binary level scanning of Docker image
- Presents a security profile of the image
- Push image to registry -> Docker cloud performs scanning
- Uses CVE Scanning validation service that is updated regularly
- Available in Docker Cloud, Docker EE
- Each vulnerability is associated with a severity
- Use official repos, they've been using it for a while now

Tip: Make sure you are using a modern kernel version.

- v3.10 is the bare minimum required version for Docker to function, but v4.6.1 is current, as of June 1, 2016.
- "Reasonably fresh" is generally better than stale, when it comes to security.
- (Though "too fresh" can lead to new bugs, so it's good to find the sweet spot where code is reasonably well tested, but not getting old and creaky.)

Tip: Remove all non-essential services from the Docker host.

- This would include other (non-Docker-related) services:
  - web services
  - database
  - user accounts & services
  - file shares
  - etc.

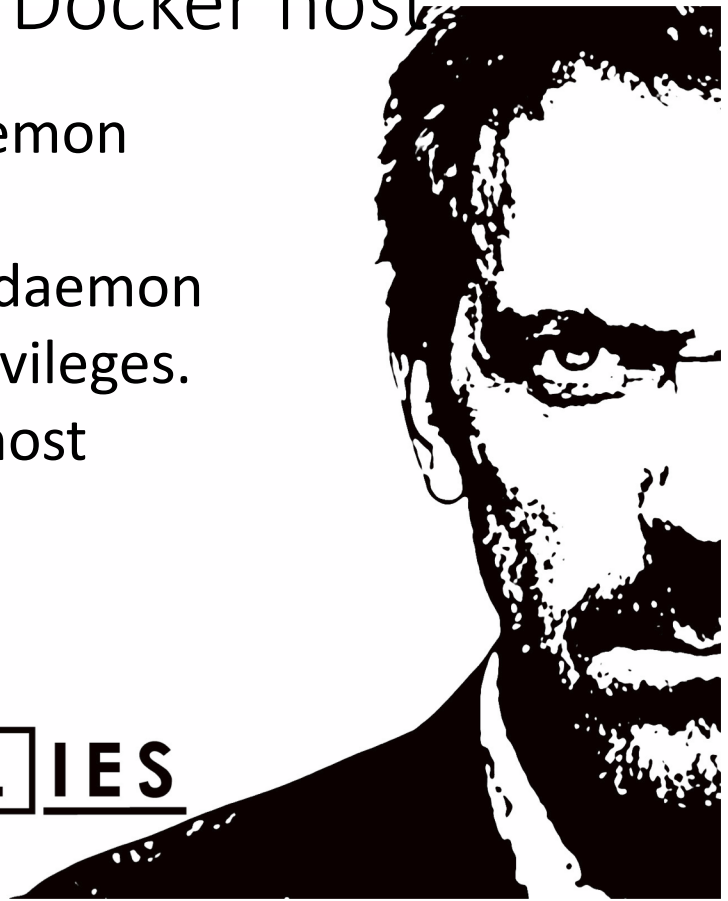
Tip: Keep Docker up to date!

- Docker ecosystem is evolving VERY fast
  - months == years
  - major == minor
- Audit "`docker version`"
- Infrastructure as "Code" anyone

Section 5.1 | Hardening the Docker host

Tip: Limit access to the Docker daemon

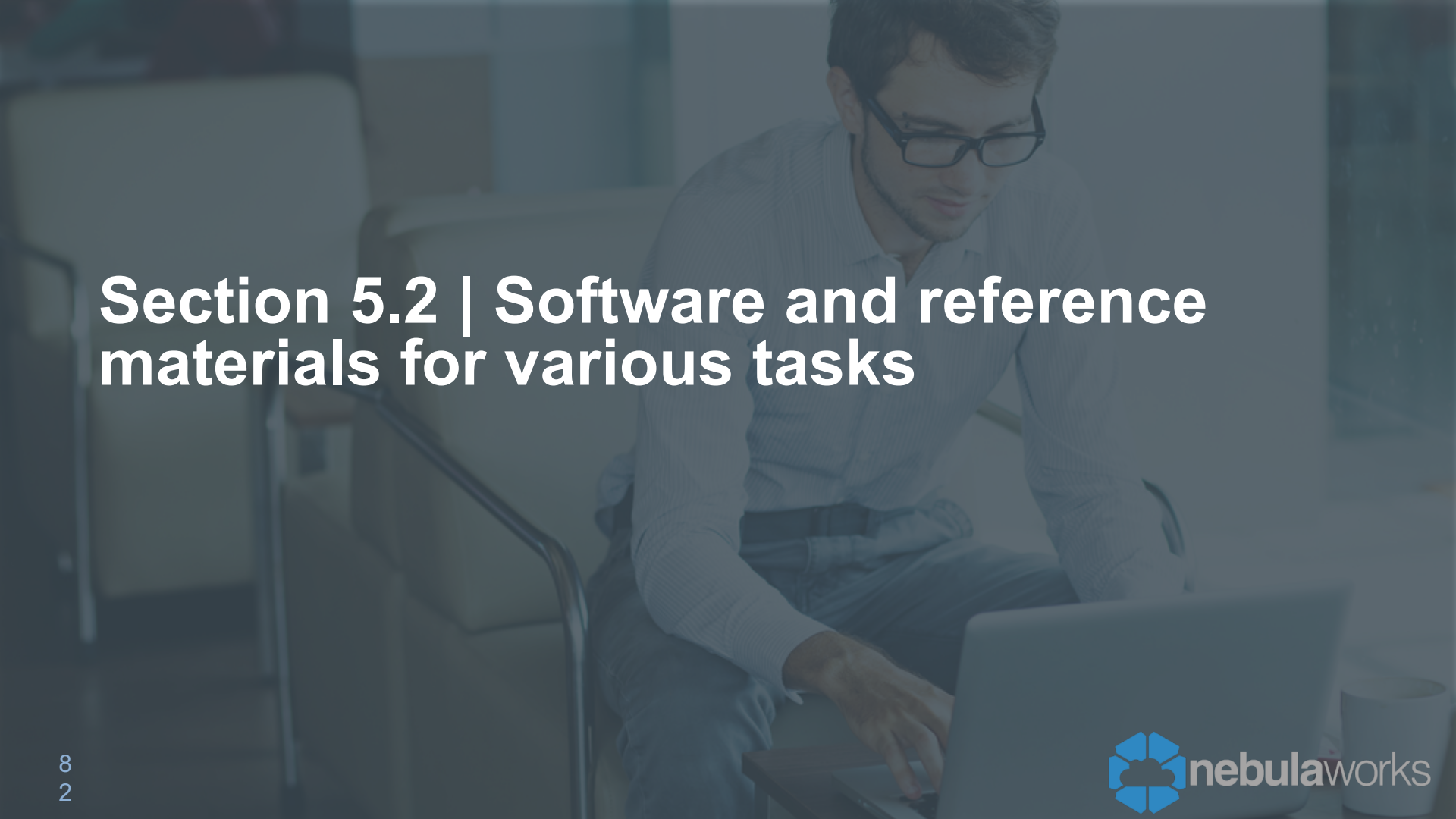
- Remember that the Docker daemon runs with elevated (root) privileges.
- Consider vulnerabilities on host



Tip: Use auditd and auditctl to monitor the activity and usage of the Docker daemon, and related files.

File to watch >>

- /var/lib/docker
- /etc/docker
- /etc/default/docker
- /etc/docker/daemon.json
- /usr/bin/docker-containerd
- /usr/bin/docker-runc
- docker.service
- docker.socket



Section 5.2 | Software and reference materials for various tasks

Linux kernel capabilities

For fine-grained control, rather than simplistic root/non-root permissions.

- http://man7.org/linux/man-pages/man7/capabilities.7.html
- The "Other kernel security features" section discusses hardening options, and a few useful tool sets, profiles, etc.

Seccomp - secure computing mode

- used to sandbox processes in the Linux kernel, and isolate them from making any system calls except for a very few approved ones.
- Leverages a whitelist that will deny system calls
- Distribution must support Seccomp
- https://en.wikipedia.org/wiki/Seccomp
- https://docs.docker.com/engine/security/seccomp/

- AppArmor - A method of restricting and controlling application capabilities on an individual basis, using profiles, and with a "learning mode" to help study usage patterns and generate profiles.
- Security profiles automatically added to containers, not daemon
- Will use default but can be overriden

```
$ docker run --rm -it --security-opt apparmor=docker-default hello-world
```

- This is often touted as a more "friendly-to-implement" alternative to SELinux.
- https://en.wikipedia.org/wiki/AppArmor
- https://docs.docker.com/engine/security/apparmor/

SELinux - Security-Enhanced Linux

- NSA and DoD developed guidelines for hardening systems.
- Thorough, though sometimes difficult to administer.
- https://en.wikipedia.org/wiki/Security-Enhanced_Linux

TLS - Transport Layer Security

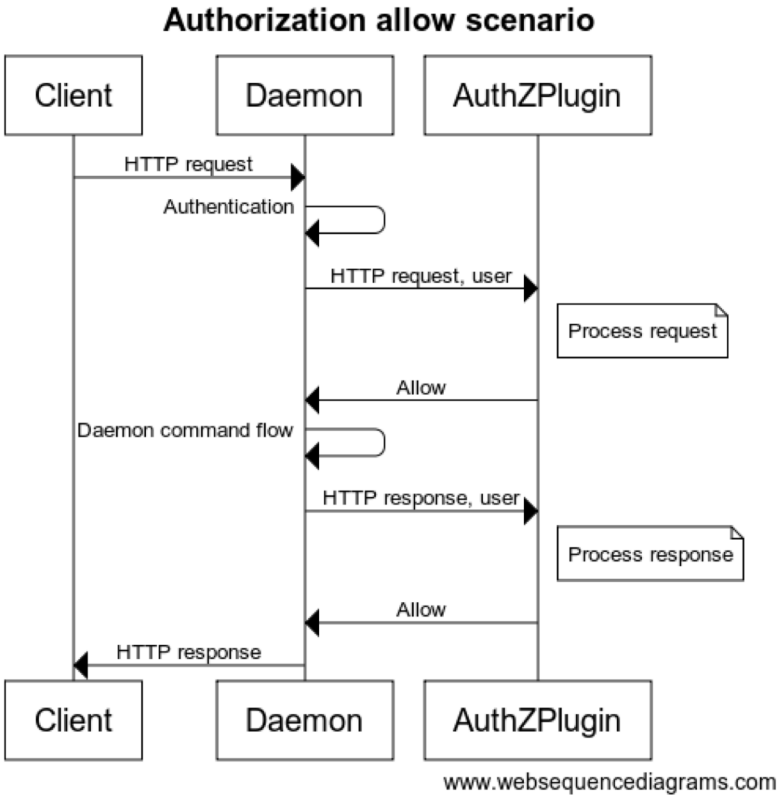
- Most commonly seen with `https://` URLs.
- Docker has no inherent built-in security.
- TLS is the recommended system for securing communications with encryption and authentication.
- https://docs.docker.com/engine/security/https/

TLS - Transport Layer Security (cont'd)

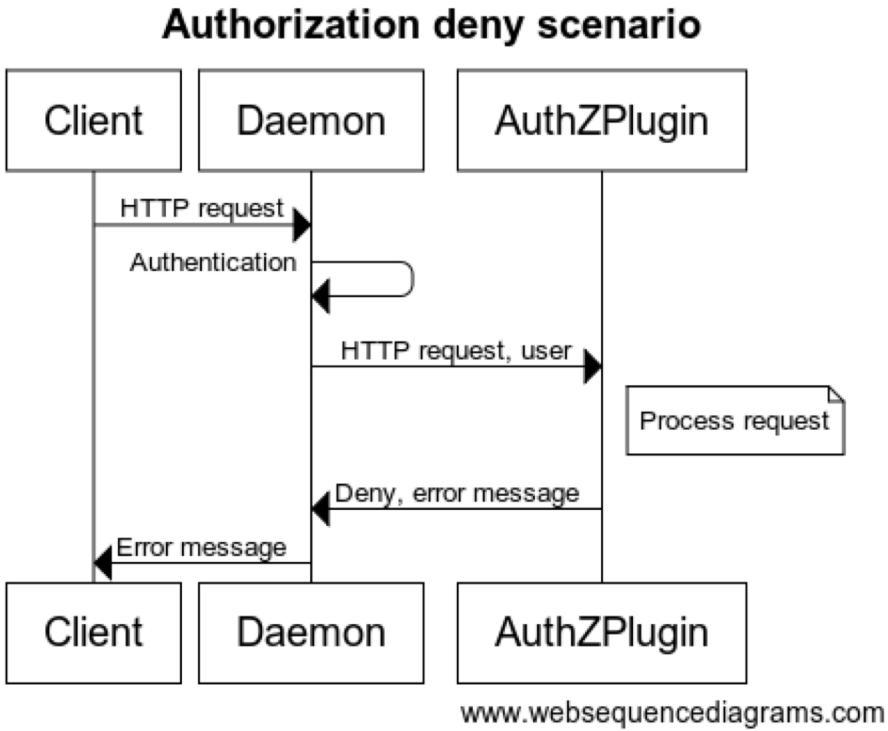
- Don't use self-signed certificates on anything other than a testing machine.
  - No one will trust it, and it will raise lots of warnings/alerts.
  - If your organization is larger than a startup, you probably have someone on staff who deals with certificates and security. Talk to an IT or Security person to track them down, and they can usually provide you with proper certificates.

Docker authorization & authentication plugin

Allow >>>

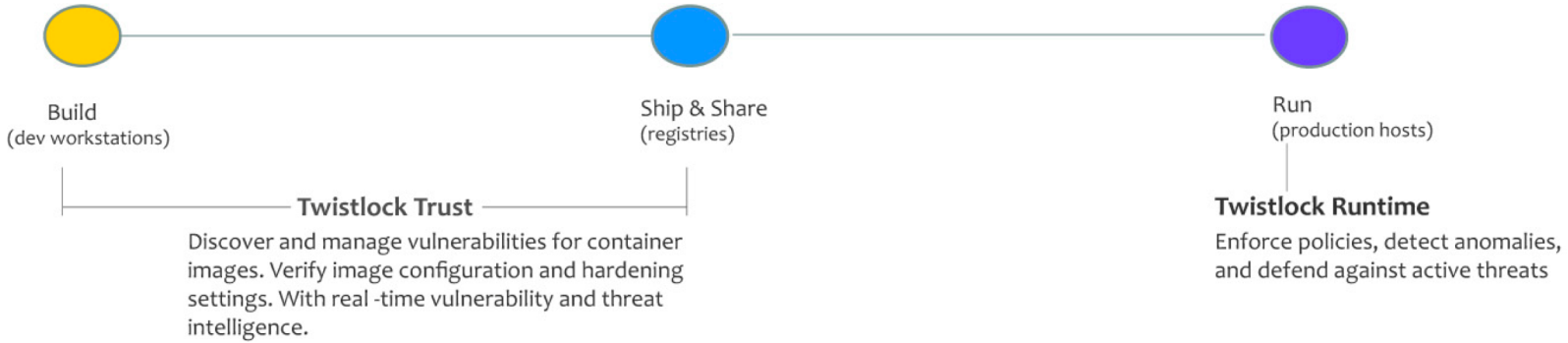


Docker authorization & authentication plugin



Twistlock
Casbin AuthZ
HB Plugin

- Design: agentless, full stack, run anywhere, cradle to scale



Vault by Hashicorp

- Provide single source of secrets for humans and machines
- Data is encrypted at rest as well as in transit
- Manage secret lifecycle
- Revoke credentials in case of attack
- Traceability through audit logs
    - https://www.hashicorp.com

Section 5.3 | Docker daemon configuration

Tuning the docker engine

- Restrict network traffic between containers
- Set logging levels
- Setting storage drivers
- Configure centralized and remote logging
- Do not use insecure registries
- Allow docker to manage/update iptables
  - (It's probably much better at it than you are. ;-))
- Use TLS authentication

Isolating containers with a user namespace

- Prevent privilege-escalation attacks from inside a container by configuring your application to run as an unprivileged user
- If your process must run as a root user, you can re-map this user to a less-privileged user on the docker host!
- Added in v1.10 to address root privilege issues:
  - supported directly as an engine option.
  - allows for the root user in a container to be mapped to a non uid-0 user outside the container
  - all container root users are mapped to the same outside user

How do we map user IDs from the host to the container?

- the remapping itself is handled by two files: `/etc/subgid` and `/etc/subuid`.
- Example of an entry in `/etc/subuid`:
  `user1:241082:65536`
- on the host, especially when you have not explicitly created non-root users for use in the containers.
- Example issue: RedHat has a known issue where automatic mapping of sub-UIDs and sub-GIDs doesn't happen. Manual mapping may be required for now on those systems.

Audit the user namespace:

- `ps -p $(docker inspect --format='{{ .State.Pid }}' <CONTAINER ID>) -o pid,user`
- The above command would find the PID of the container and then would list the host user associated with the container process. If the container process is running as root, then this recommendation is non-compliant.
- Alternatively, you can run the below command to ensure that the docker daemon is running with --userns-remap flag.
- `ps -ef | grep docker`

Remediation:

- Please consult Docker documentation for various ways in which this can be configured depending upon your requirements. Your steps might also vary based on platform - For example, on Red Hat, sub-UIDs and sub-GIDs mapping creation does not work automatically. You might have to create your own mapping.

- General repair steps >>>

Step 1: Ensure that the files ```/etc/subuid``` and ```/etc/subgid``` exist.

- `touch /etc/subuid /etc/subgid`

Step 2: Start the docker daemon with ```--userns-remap``` flag

- `docker daemon --userns-remap=default &`

Section 5.3 | Docker daemon configuration

Impact:

- User namespace remapping makes quite a few Docker features incompatible and also currently breaks a few functionalities. Check out the Docker documentation and referenced links for details.

Default Value:

- By default, user namespace is not remapped.

(Above section on user namespaces from CIS Docker 1.11.0 benchmark, section 2.8)

And there's also this, which goes into more detail about kernel capabilities, and advises not relying completely on just user namespaces.

- https://medium.com/@ewindisch/linux-user-namespaces-might-not-be-secure-enough-a-k-a-subverting-posix-capabilities-f1c4ae19cad#.cwyx9fp0l

And this video:
Netdev 1.1 - Namespaces and CGroups, the basis of Linux containers

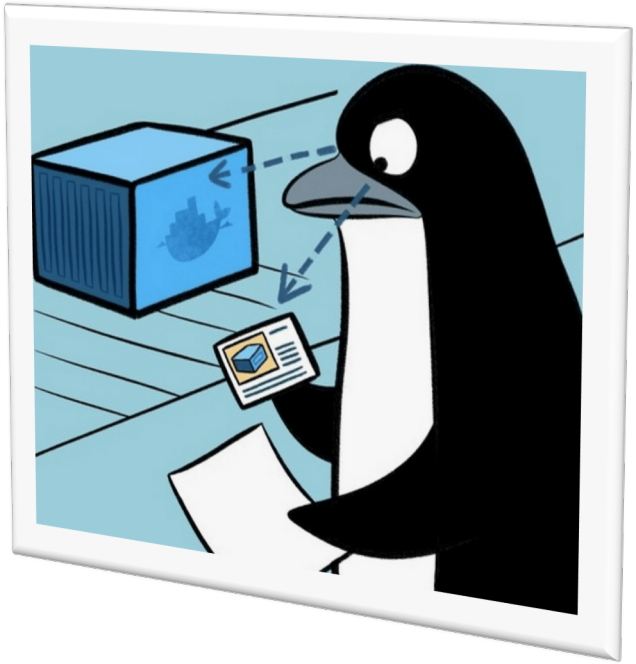
- https://www.youtube.com/watch?v=zMJD8PJKoYQ (47.5 minutes)

Section 5.4 | Trusted content/trusted images

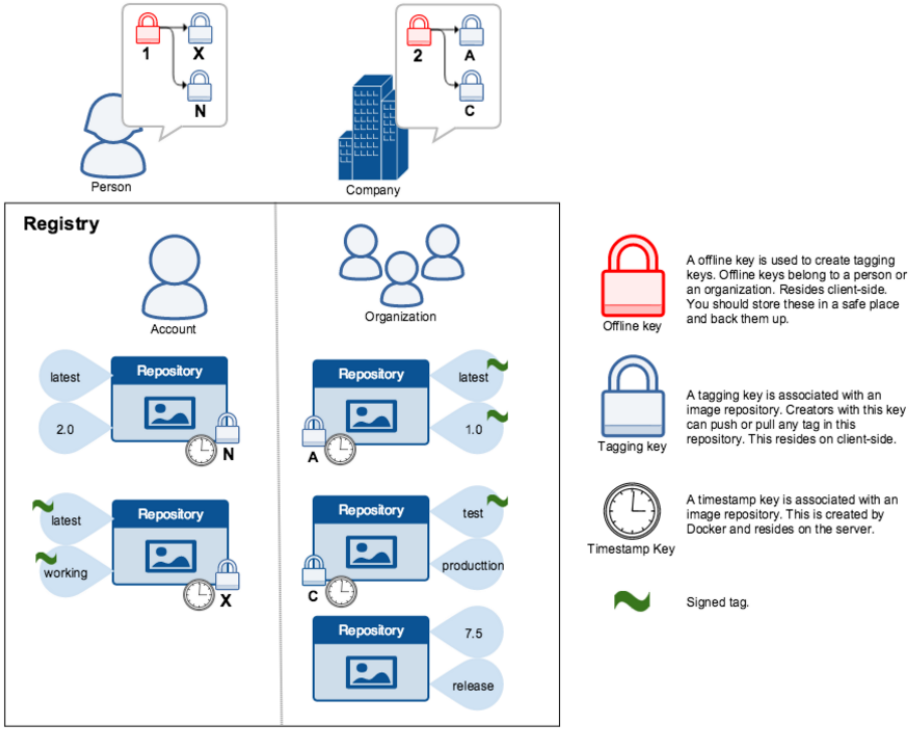
How do you know that an image is safe?

Sure, there's lots of stuff out there, but can it all be trusted?
Is it maintained with the latest patches?

(Not always, but you shouldn't blindly trust, either…)



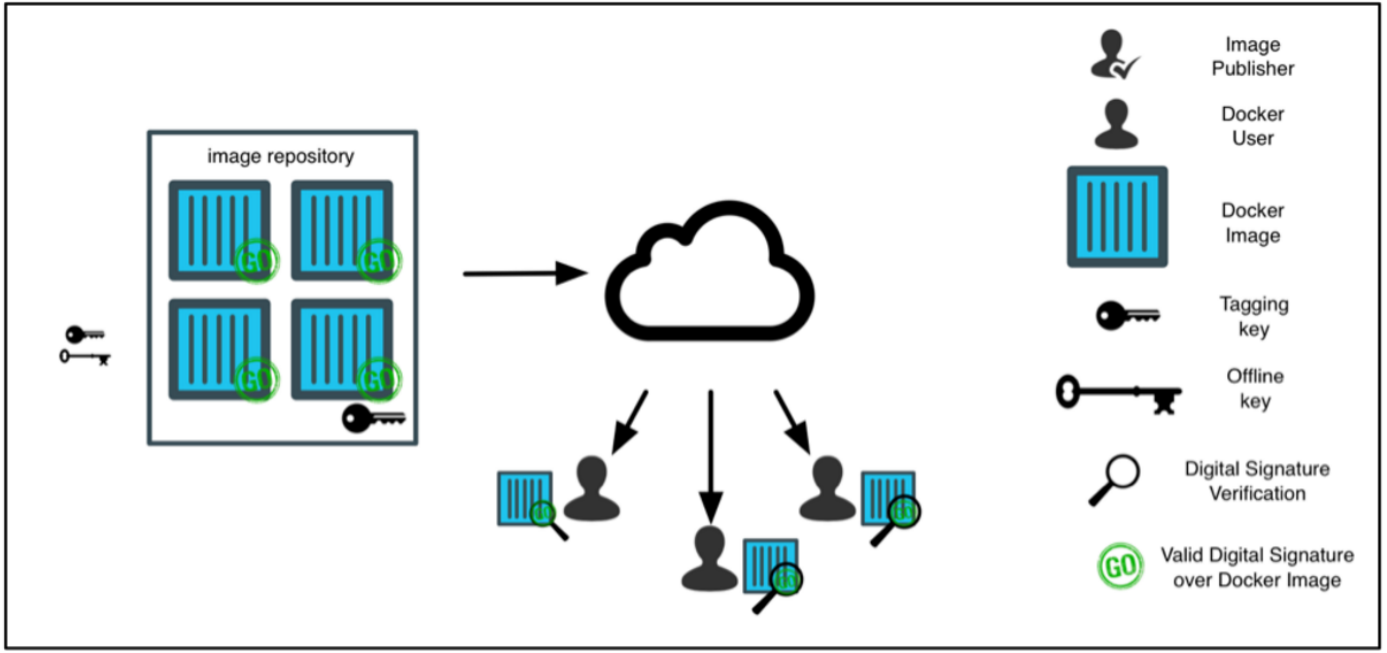
How images get signed



How images get signed

- Once Docker Content Trust has been enabled, only signed images can be used
- If you try to pull an image or run a container from an image that is not signed, it will fail
- The first time you pull an image, trust is established to the repository with the offline key
- All subsequent interactions with that image require a valid signature verification from that same publisher
- Once trust is established, TUF will ensure integrity and freshness on the content, via the use of a timestamp key
- Docker uses and manages the timestamp key

How images get pulled



Docker Content Trust (Image signing and verification)

- https://blog.docker.com/2015/08/content-trust-docker-1-8/

The Update Framework (TUF)

Docker Notary project:

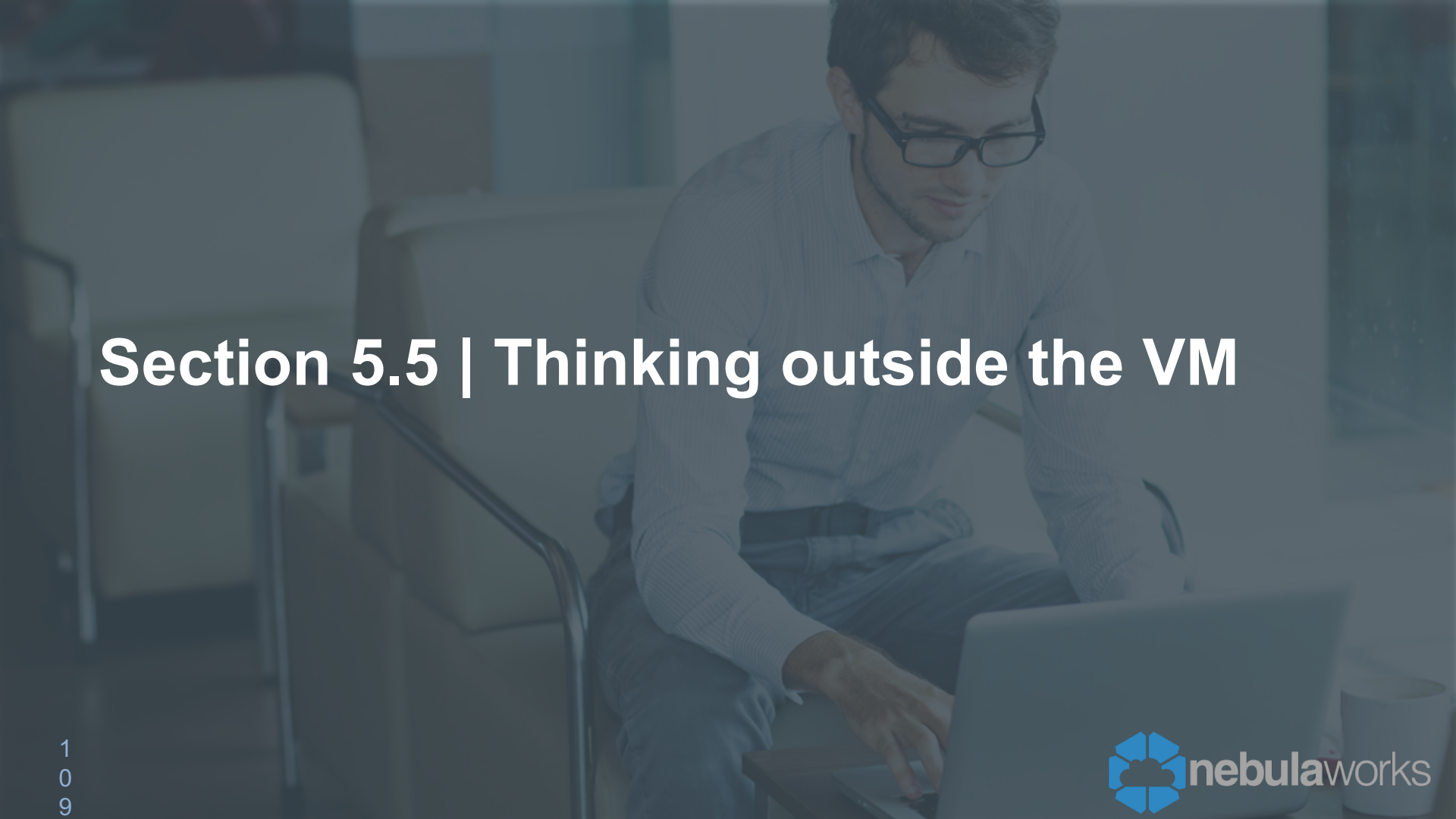
- https://github.com/docker/notary

Docker Security Scanning safeguards the container content lifecycle

- https://blog.docker.com/category/security-2/

There is also the Banyan study finding 30-40% of images on the Docker Hub having known security vulnerabilities.

- Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities
- http://www.banyanops.com/blog/analyzing-docker-hub/



Section 5.5 | Thinking outside the VM

Section 5.5 | Thinking outside the VM

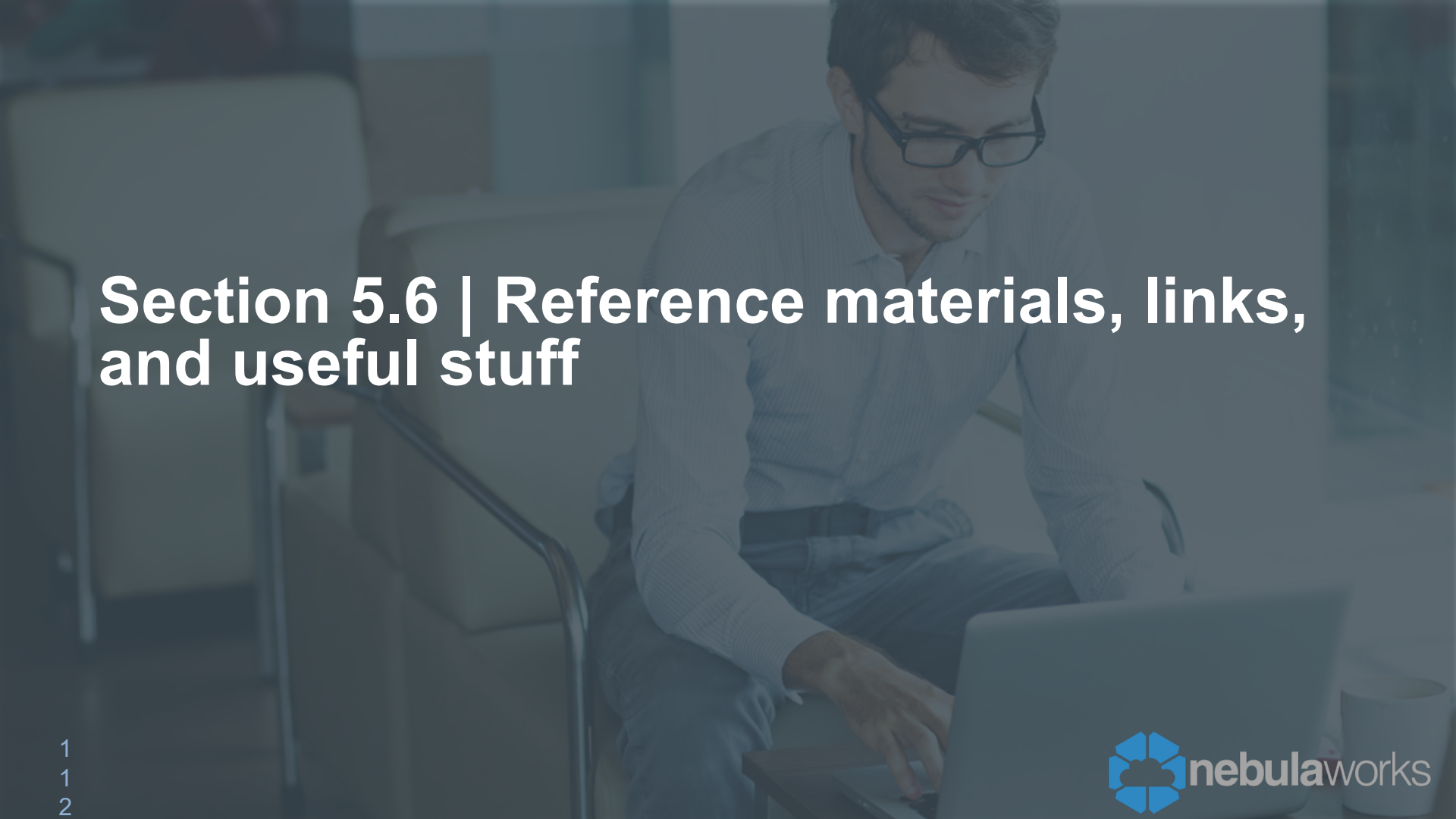
…to unlearn what you have learned is the most necessary kind of learning.
-- Antisthenes

Section 5.5 | Thinking outside the VM

Don’t run sshd in a container

- think in terms of loosely coupled microservices.
- See advice from [Jerome Petazzoni]
- Alternatives to ssh : nsenter or docker exec

NEBULAWORKS



Section 5.6 | Reference materials, links, and useful stuff

Docker security references

- https://docs.docker.com/engine/security/security/
- https://blog.docker.com/2015/05/understanding-docker-security-and-best-practices/
- https://www.docker.com/sites/default/files/WP_Intro%20to%20container%20security_03.20.2015%20%281%29.pdf

CIS Docker Benchmark documentation:

- https://benchmarks.cisecurity.org/tools2/docker/CIS_Docker_1.11.0_Benchmark_v1.0.0.pdf

Docker Security Cheat-Sheet

- http://container-solutions.com/docker-security-cheat-sheet/

Trust models in Docker. (There's a LOT on this page.)

- https://docs.docker.com/engine/security/trust/content_trust/

Docker Security, Hardened Containers and a Layered Strategy

- http://blog.flux7.com/docker-security-hardened-containers-and-a-layered-strategy
- A review of some of the security features introduced in v1.10

"On the Security of Containers" (2014):

- https://medium.com/@ewindisch/on-the-security-of-containers-2c60ffe25a9e#.hhuvxhmb6

Five security concerns when using Docker

- https://www.oreilly.com/ideas/five-security-concerns-when-using-docker
- This is an excerpt from (and has a link to) a larger paper, which is a security-related report/chapter from one of O'Reilly's Docker book authors. (It can be downloaded from their site if you give them your contact info.)

Docker Security Resources & Best Practices

- https://www.twistlock.com/docker-security-resources/

Understanding and Hardening Linux Containers

- https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2016/april/understanding-and-hardening-linux-containers/
- 122 page PDF version available

Linux-Audit: Docker Security: Best Practices for your Vessel and Containers

- http://linux-audit.com/docker-security-best-practices-for-your-vessel-and-containers/
- There's quite a bit of good information here. Worth mining.

Hardening Docker Containers: Disable SUID Programs

- https://blog.tutum.co/2015/02/03/hardening-containers-disable-suid-programs/

Docker: Best Security Practices

- https://elastic-security.com/2016/04/11/docker-best-security-practices/

Docker Bench for Securty project

- https://github.com/docker/docker-bench-security

Docker crocker-blocker aims at stopping Docker shockers (Oldy but Goody)

- http://www.theregister.co.uk/2015/05/08/hardening_guide_a_docker_shocker_blocker/

- Humorous title, from The Register (UK), and an introduction to some of the CIS security benchmarks.
- This article is about a year old, and refers to an older version (1.6) of the security docs, but there are links to the new version for v1.11 listed elsewhere in these slides.:

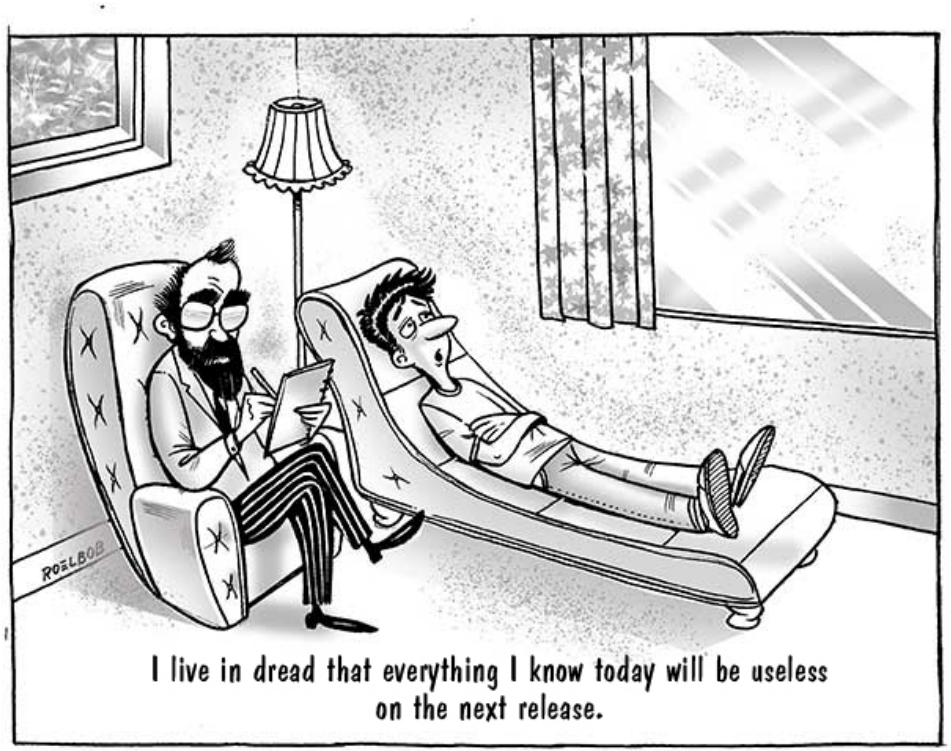
RE-IMAGINING INFRASTRUCTURE

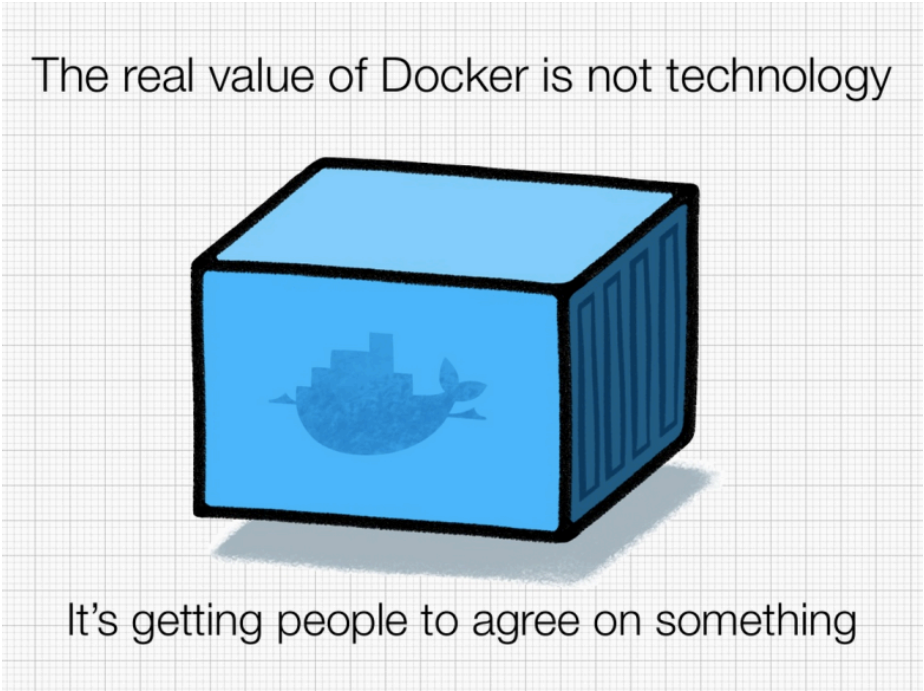
module 06

Section 6.1 | embracing a new artifact

Docker isn’t about containers, it's about artifacts.

The benefits of embracing the “container” artifact are realized when you treat them as the universal means for delivering application services.





artifact types

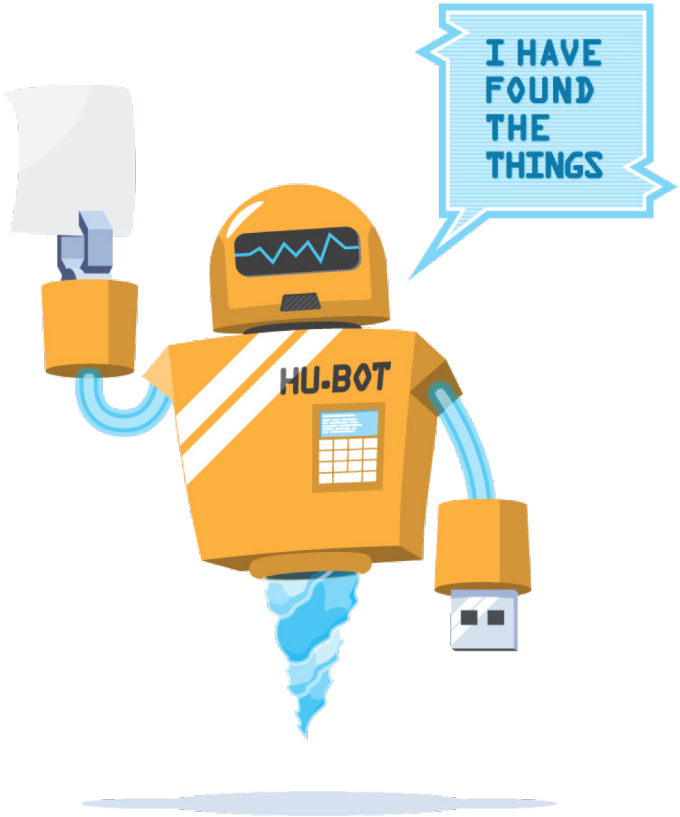
- development
- debug
- test
- production

development artifact characteristics

- complete dev environment
- able to consume mounted source
- self assembling support services
- tied to build or branch

debug artifact characteristics

- full debug tool sets
- embedded tests
- auto attaches to feedback systems
- ulimts should conform to production environment



test container characteristics

- service plumbing
- minimal debug tools
- test frameworks
- feedback hooks

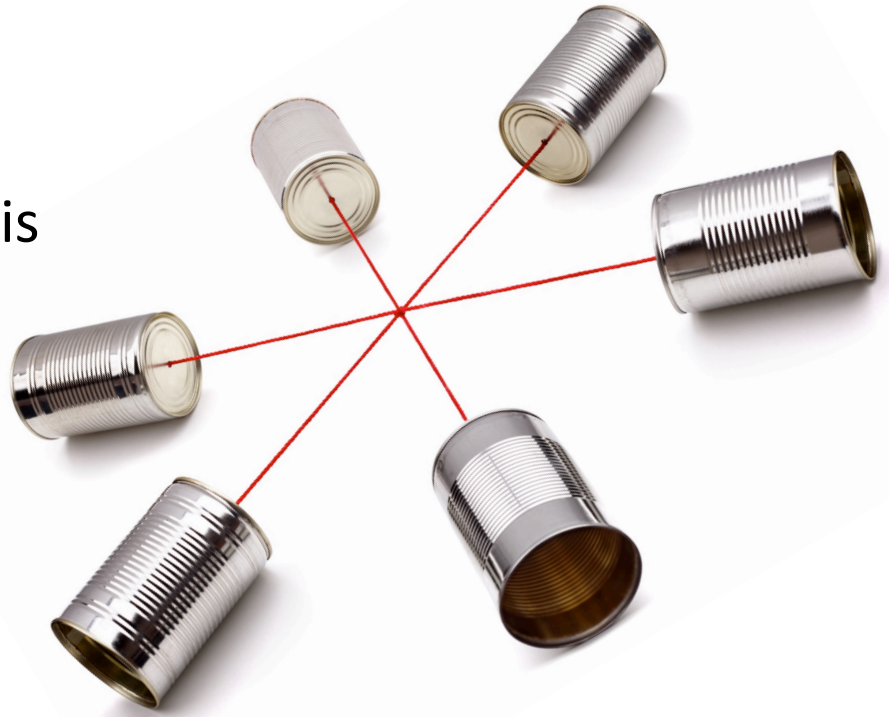
production artifact characteristics

- lean
- composable
- auto configuring
- minimal attack surface

Effective artifacts comply with Conway's Law

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

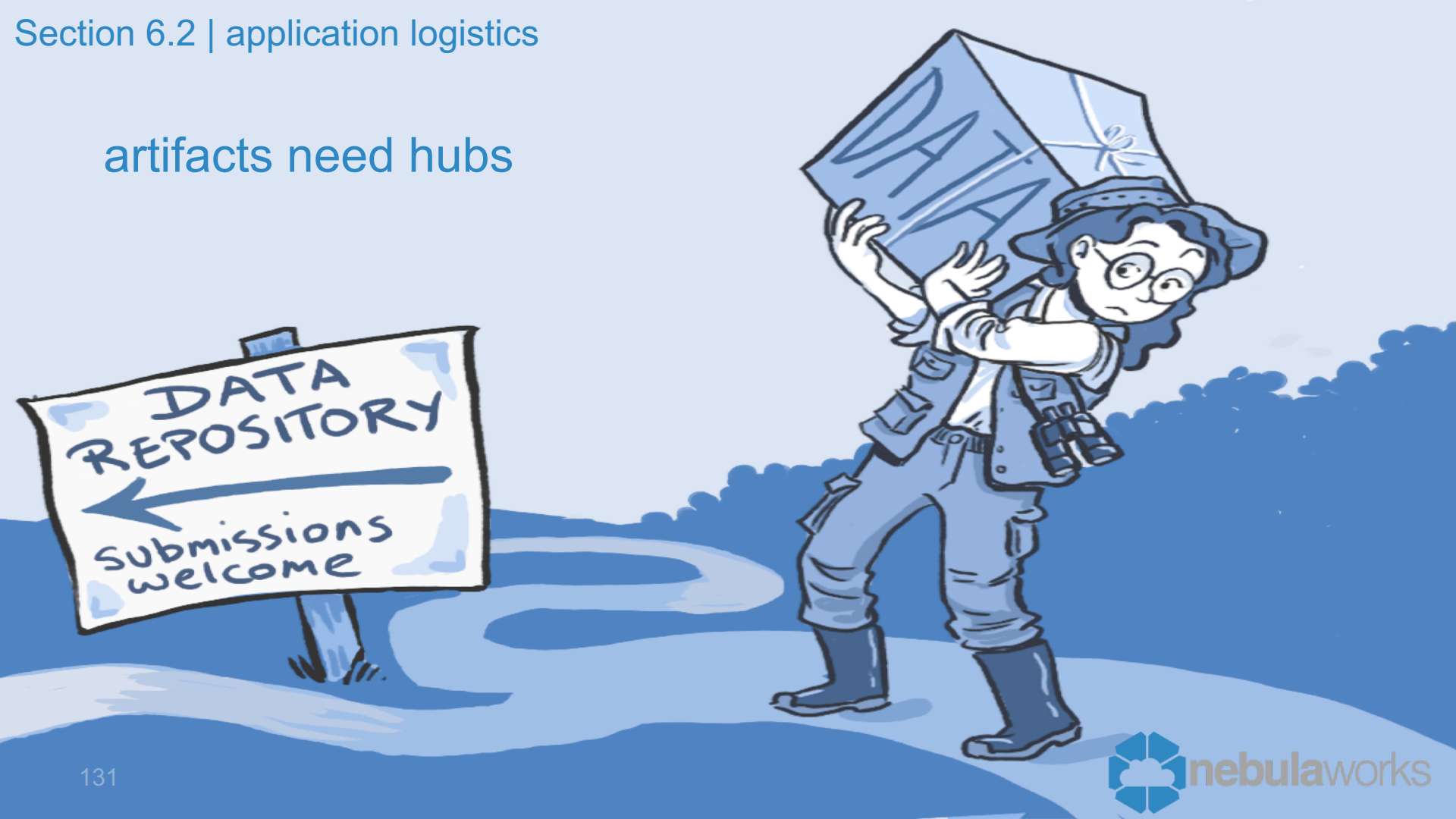
-- Melvyn Conway, 1967



Section 6.2 | application logistics

applying logistics as a metaphor for IT servicing of application products. this encourages designing an automated JIT ("just in time") process for manufacturing and delivery applications to production end points.

artifacts need hubs



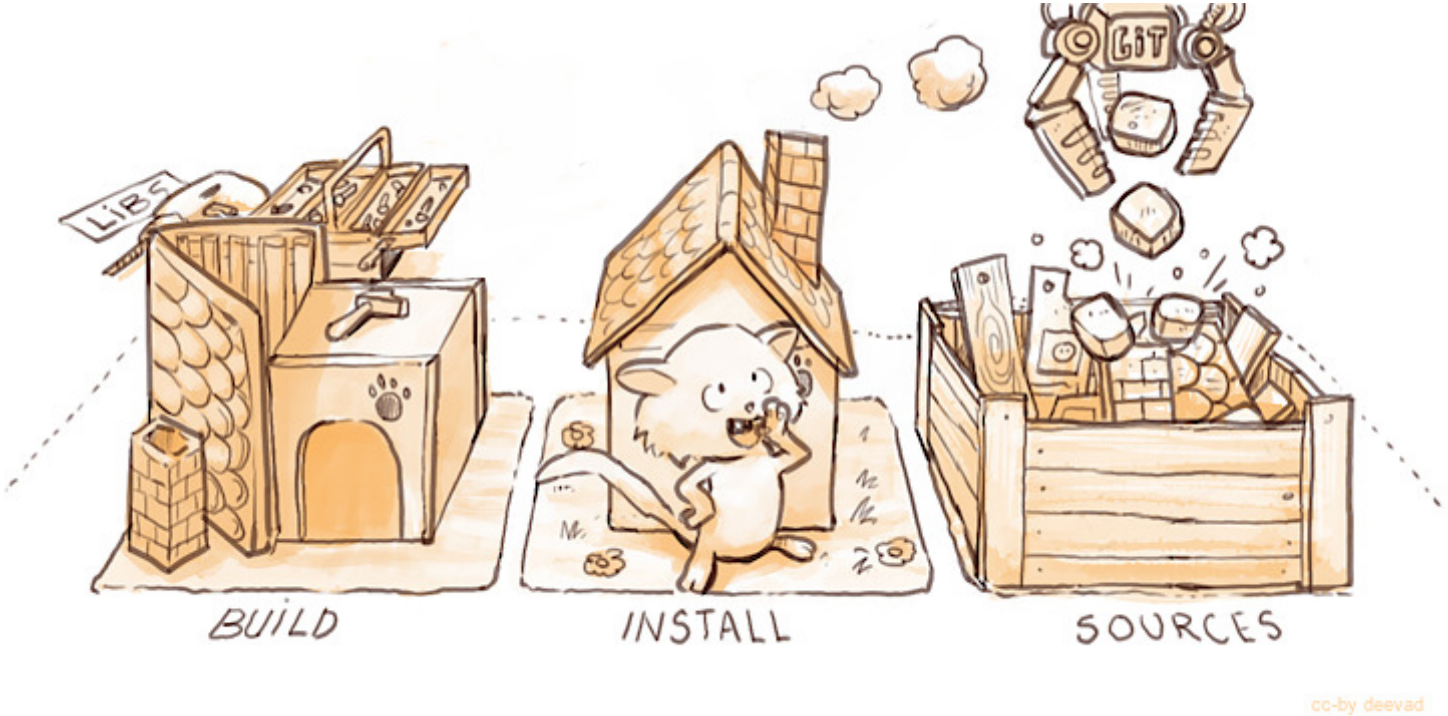
hubs provide >>>

- version control
- garbage collection
- replication facilities

hub types >>>

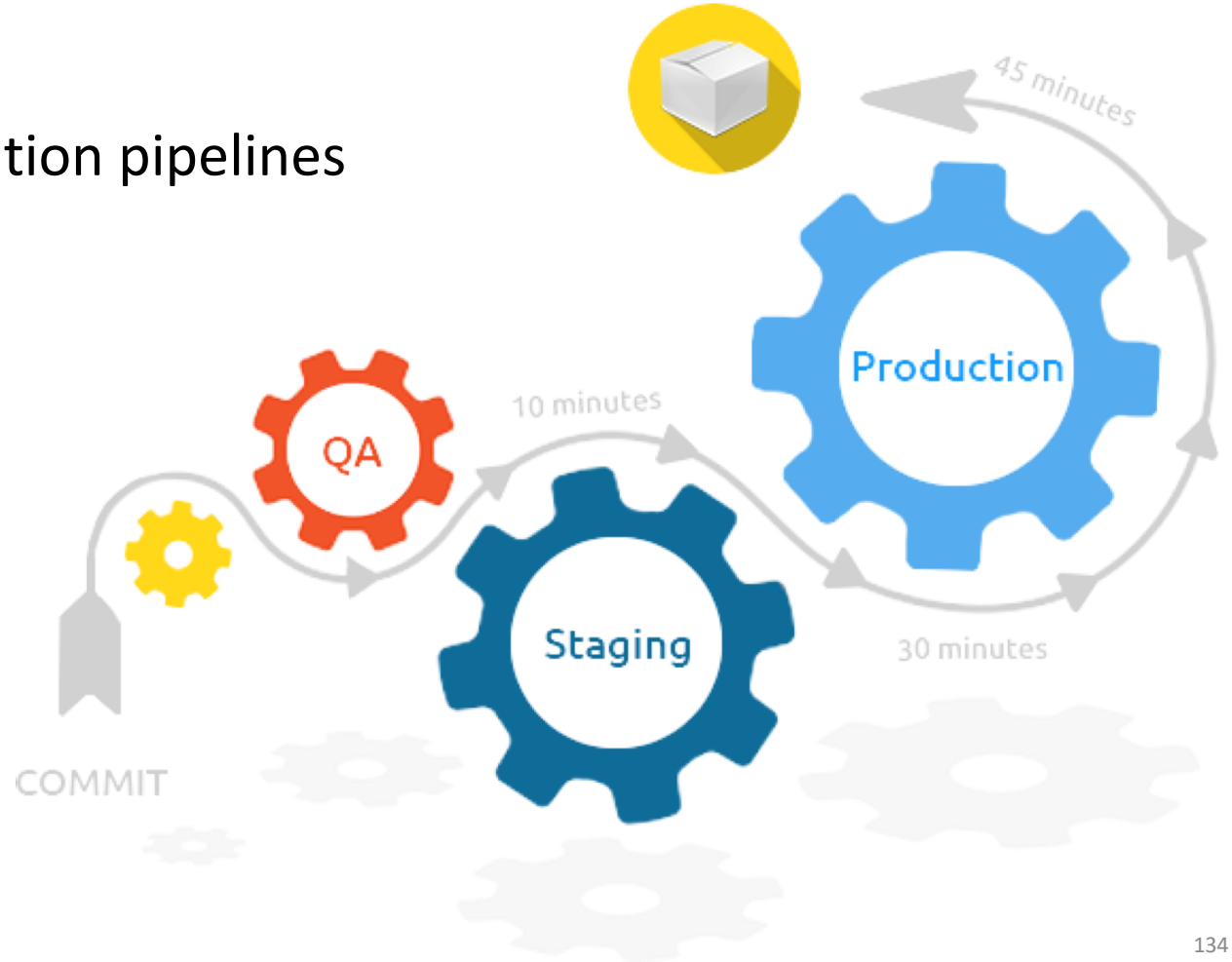
- source files
- container images
- automation tools

artifacts need integrated application pipelines

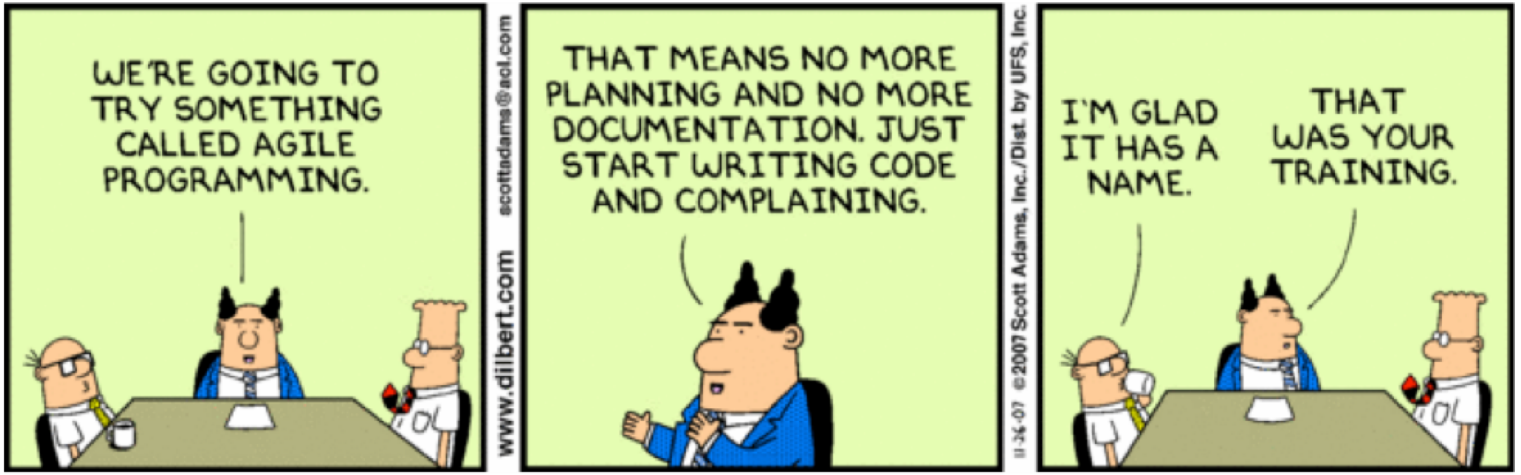


integrated application pipelines

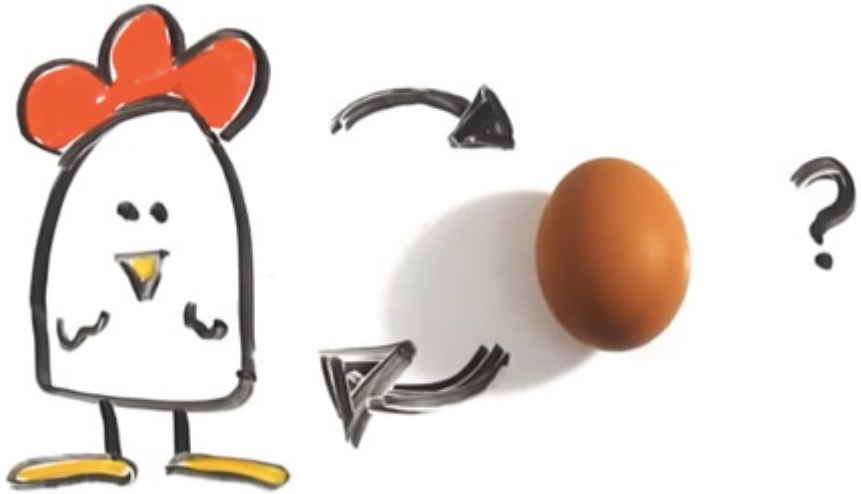
- automated
- closed
- scalable

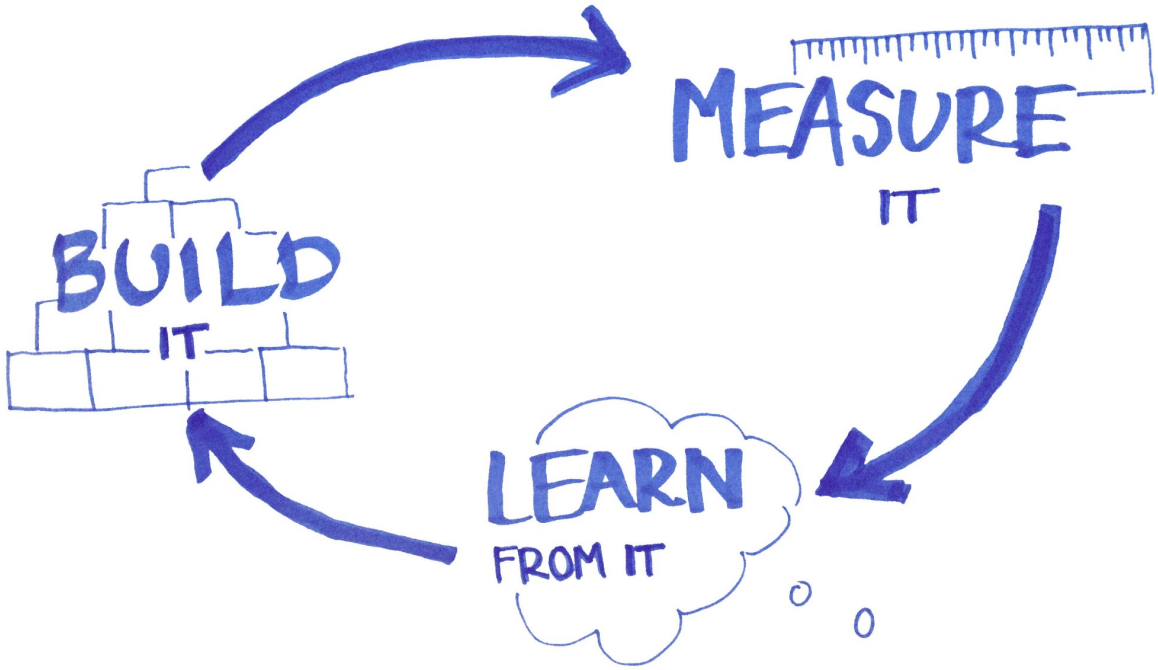


how does agile development fit in



Section 6.3 | containerization process

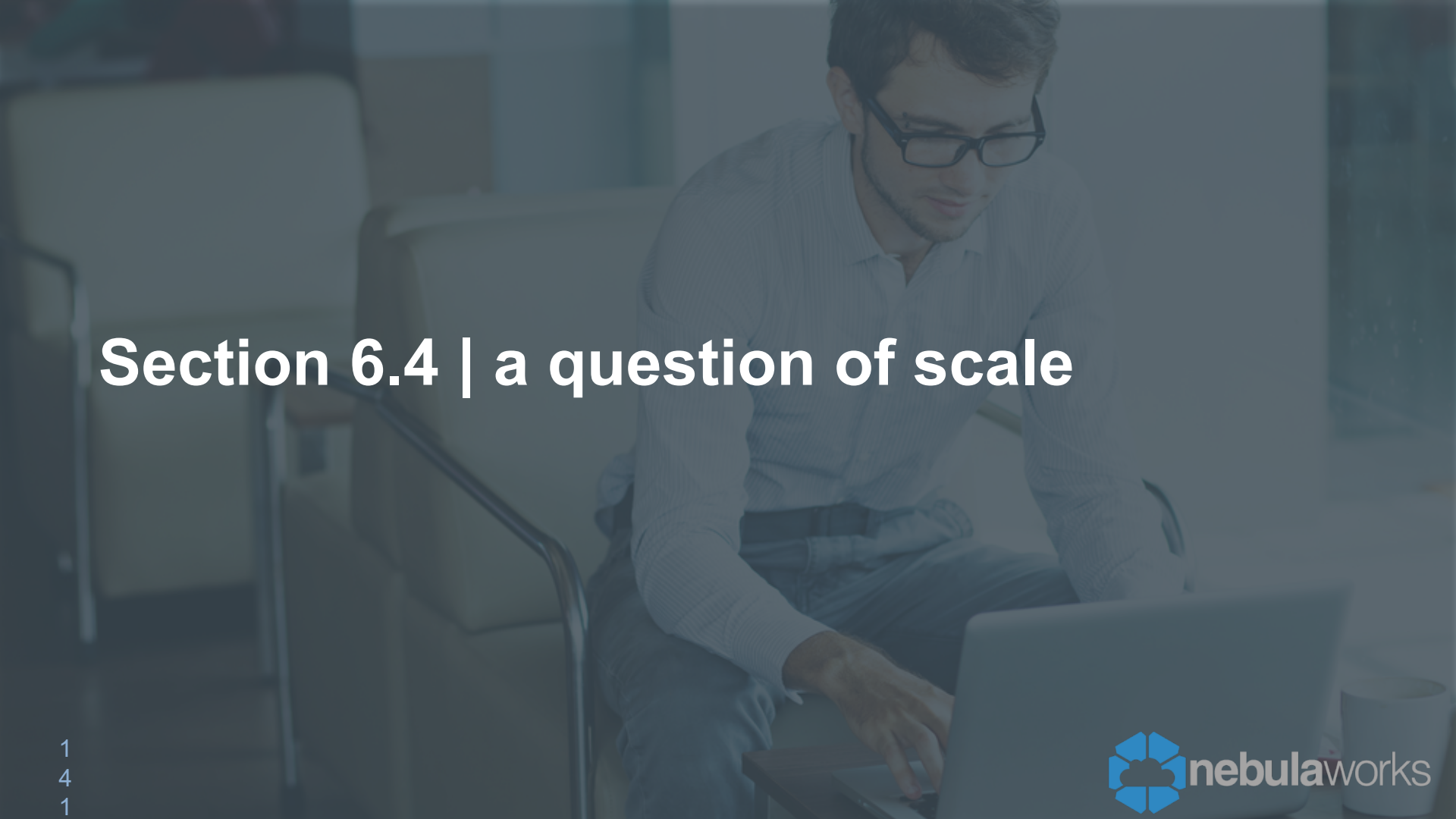



MEASURE
IT
BUILD
IT
LEARN
FROM IT

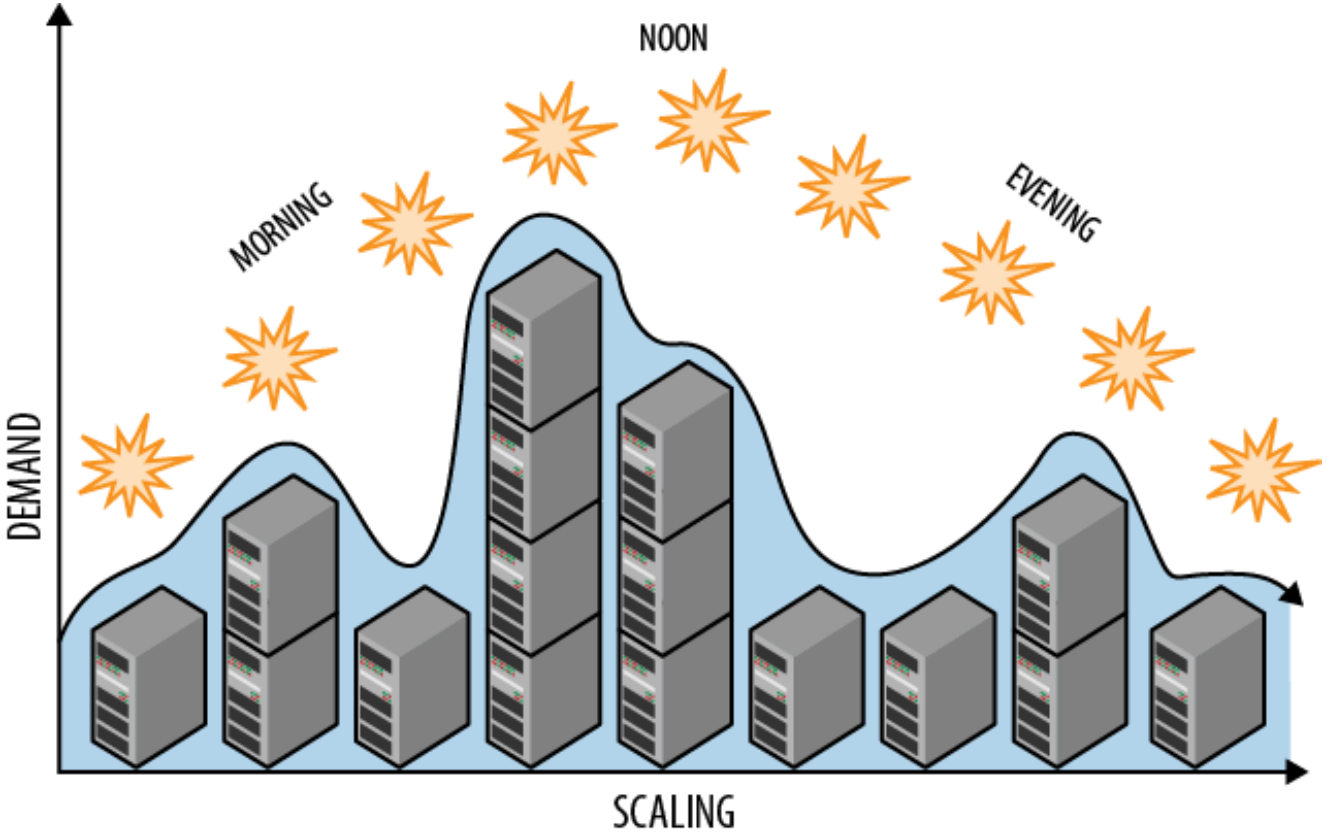
restructuring services

- networking
- logging
- process management
- discovery
- persistence

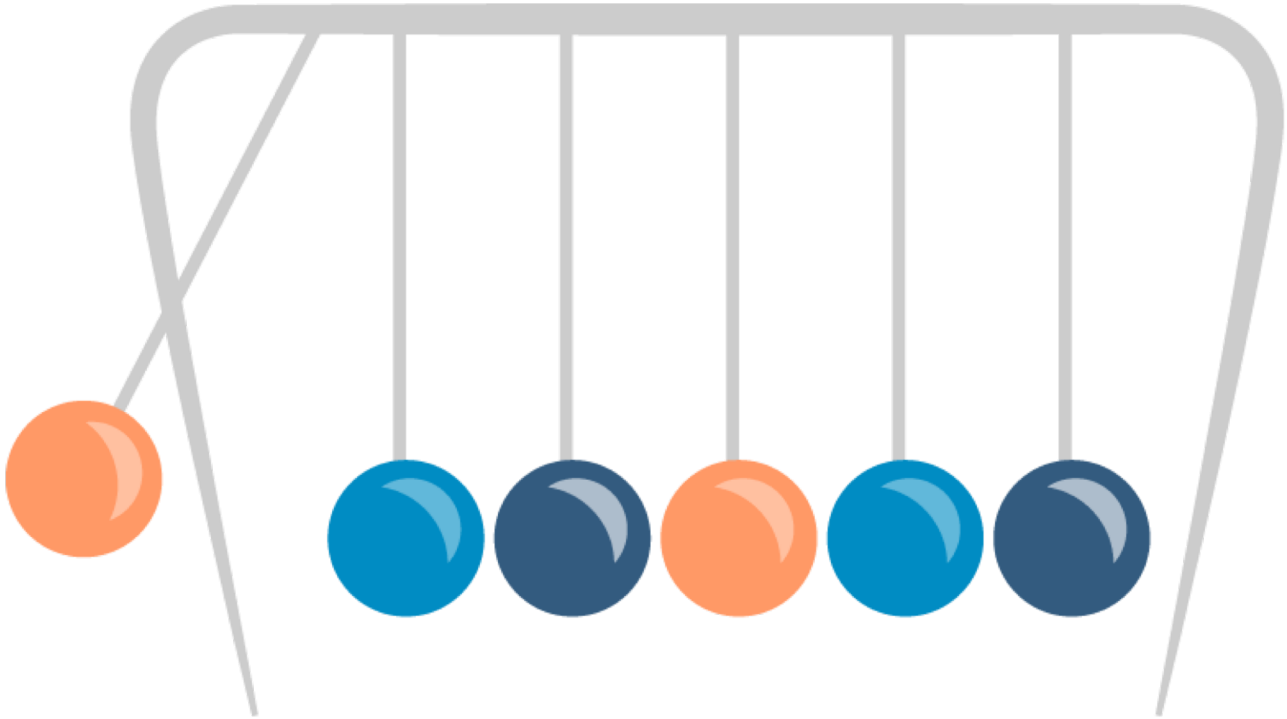
containers must be everywhere

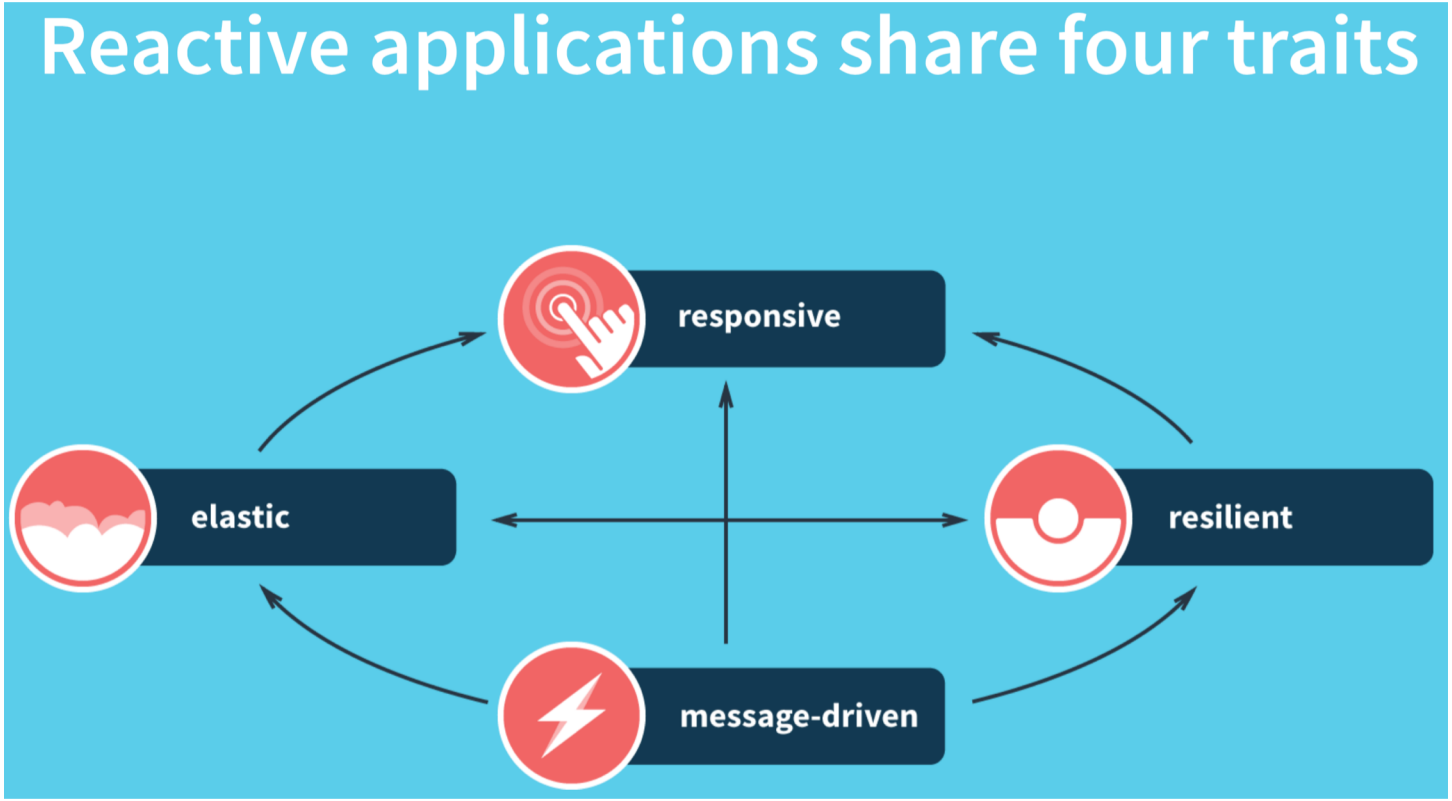


Section 6.4 | a question of scale


NOON
MORNING
EVENING
DEMAND
SCALING

REACTIVE




Reactive applications share four traits
responsive
elastic
resilient
message-driven

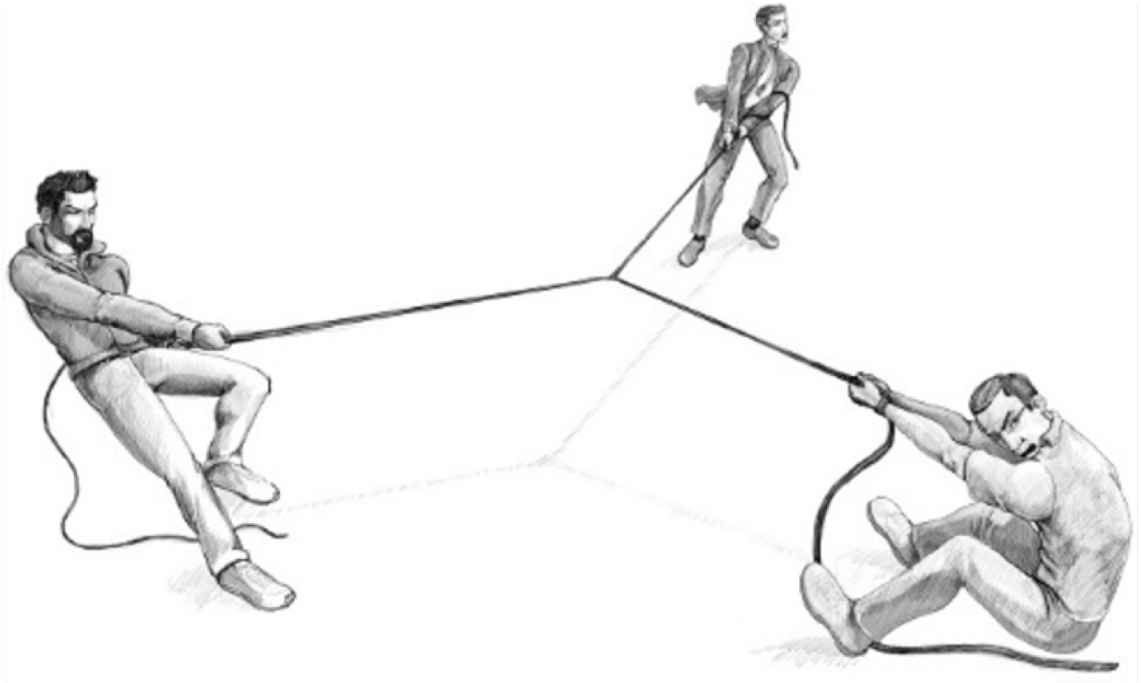
responsive
- timely
- consistent

resilient
- highly available
- ensured by replication

message driven
- asynchronous message-passing
- loosely coupled

elastic
- auto-scaling
- demand driven

CAP theorem



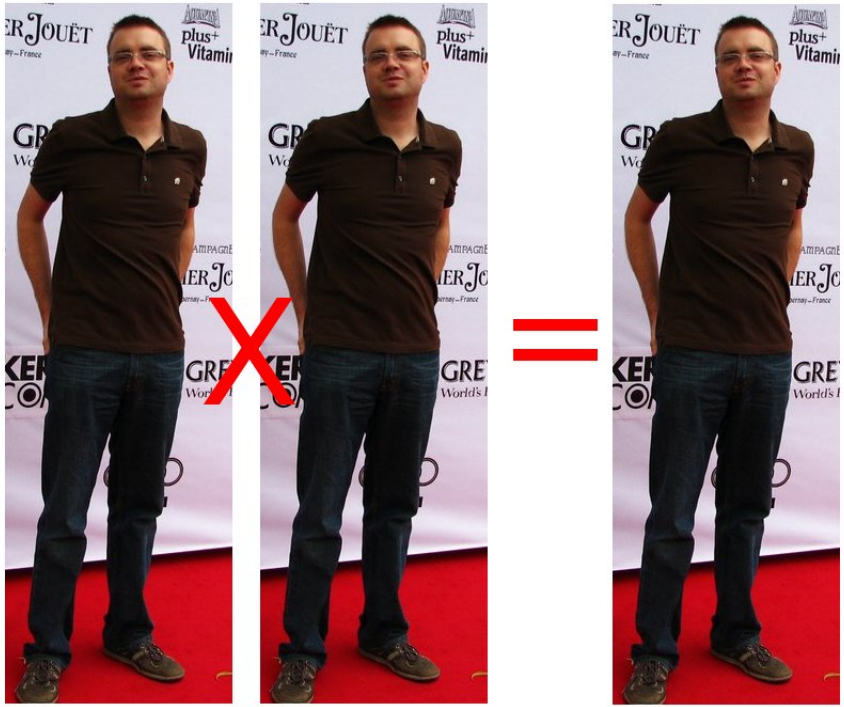
CAP theorem explained

Consistency

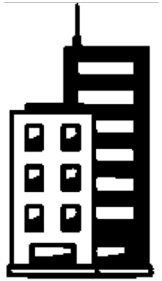
Availability

Partition Tolerance

idempotent
&
immutable



Rebuild vs Upgrade

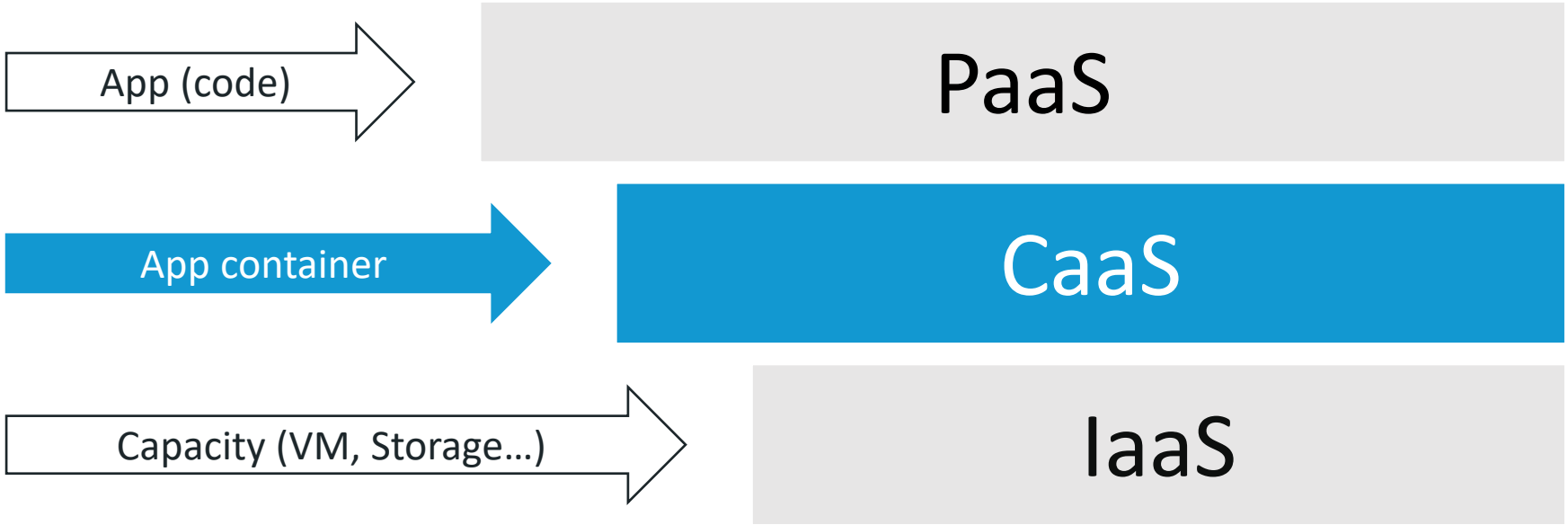


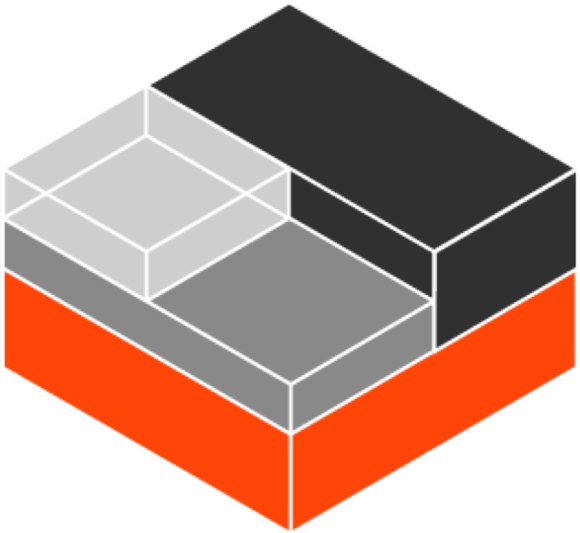
Full stack
Versioning
& idempotency

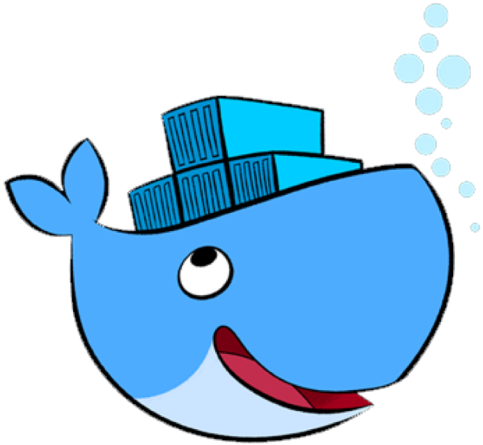
Patch
vs
immutable

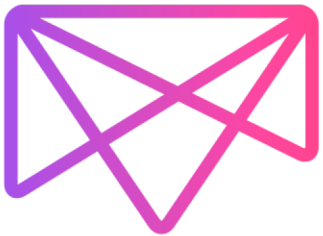
VS

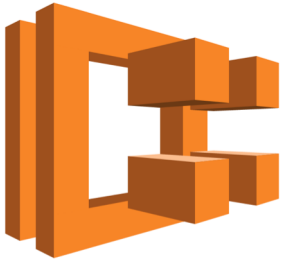
Section 6.5 | platform characteristics

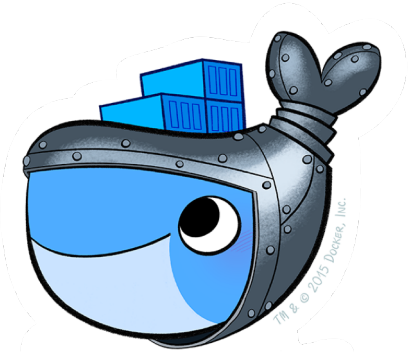

App (code)
PaaS
App container
CaaS
Capacity (VM, Storage…)
IaaS

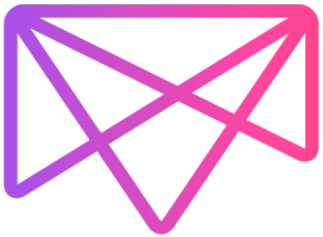


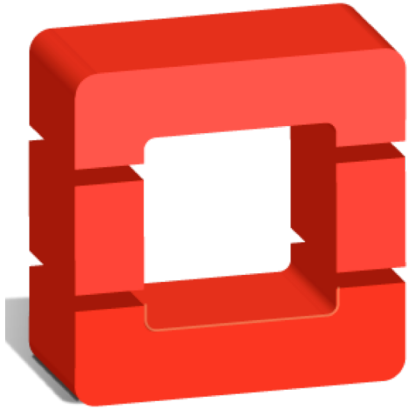







TM & © 2013 Docker, Inc.





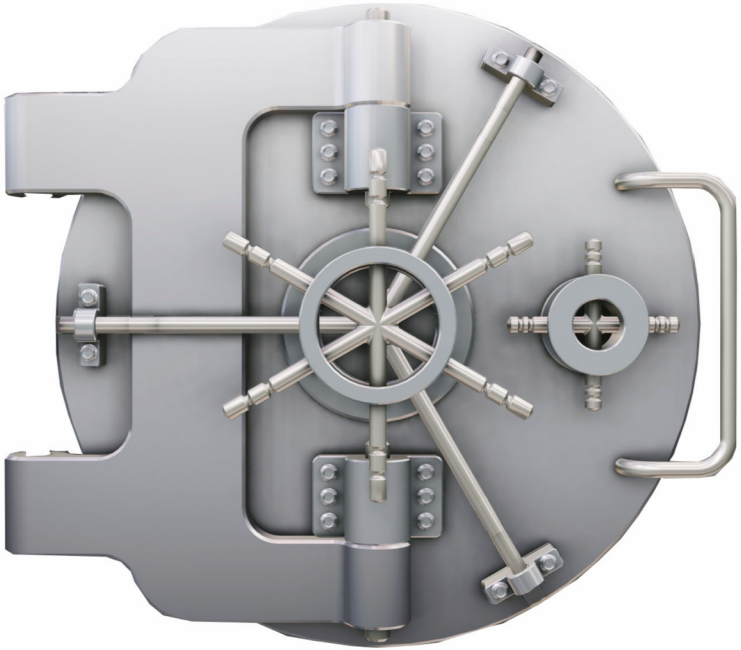

ZOO

A quick demo of gossip protocol in action

- Details https://www.serf.io/docs/internals/gossip.html
- Convergence simulator: https://www.serf.io/docs/internals/simulator.html

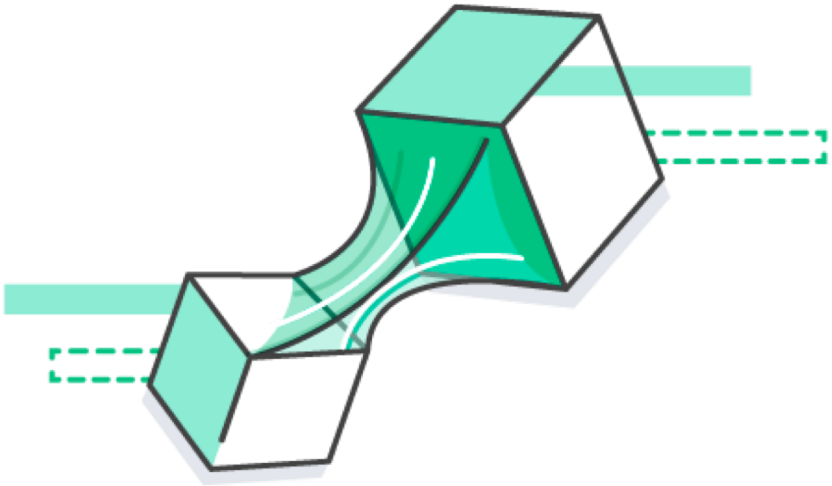
secrets manager

- secure storage
- key rolling (lease)
- detailed audit trail
- secret revocation
- dynamic secrets

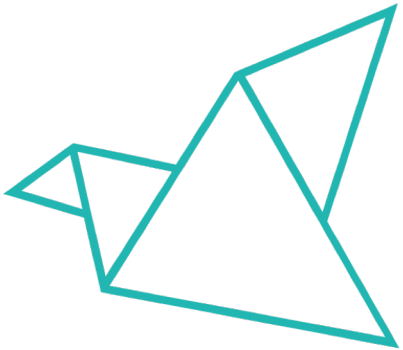


elastic storage

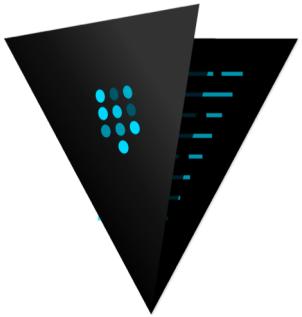
- persistent
- independent
- highly available

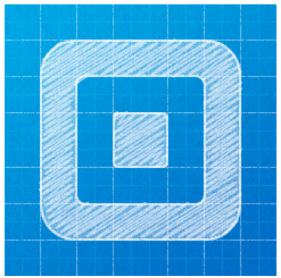



portworx

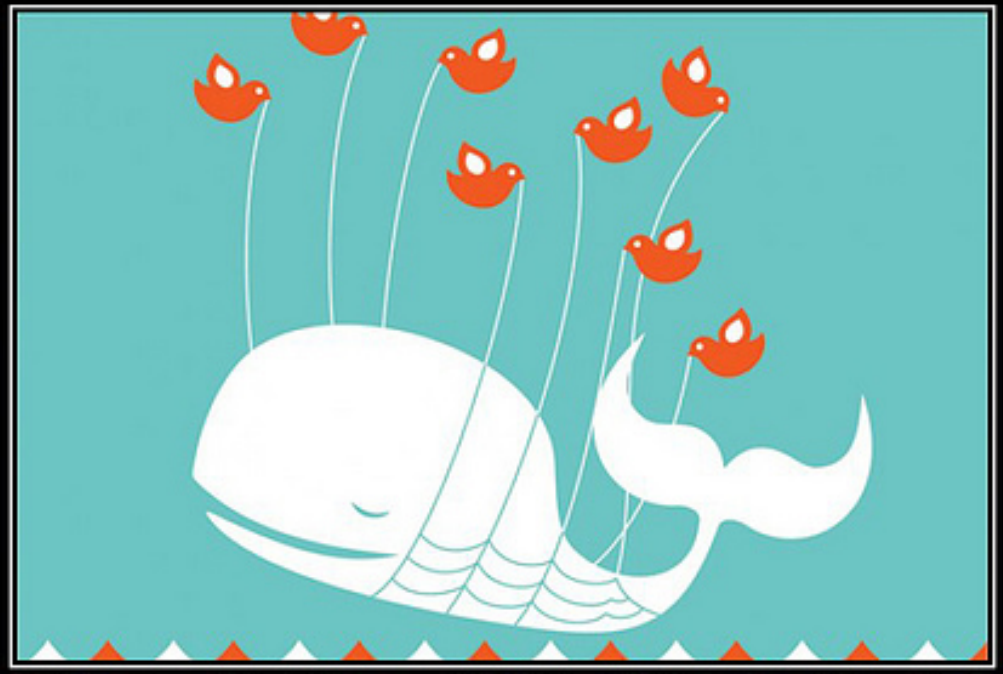


OPTIONS





- Vault by Hashicorp
- Keywhiz by Square
- Amazon KSM (Key Storage Management)
- Competent by Lyft



FAIL WHALE

Twitter: Failure is an option. At least once a day, or whenever you need it.

End of day 2