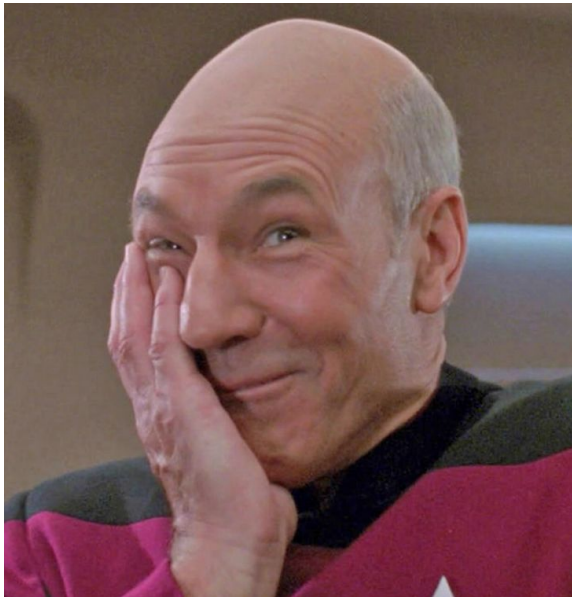
Container Tools

The Next Generation

Presenters

Laine Minor



Joshua Smith

Agenda

1. What are containers?
2. Where do containers run?
3. The Container Lifecycle and its Tools
4. Tangential Tools
5. Recs for Reasonable Defaults

What are containers?

Thanks, Google...

Refine by material

Plastic

Glass

Stainless Steel

Ceramic

A brief history...

A “container” is actually made up of several of the features that are part of the Linux kernel, such as:

- `chroot`
- `cgroups`
- `namespaces`
- `SELinux profiles`

A brief history...

They're also made up of lots of open source projects like:

- `Docker` (the packaging format)
- `Docker` (the container management runtime)

The list of open source projects included *in* a "container," or used to *manage* containers, changes all the time as our collective maturity with them increases.

...that's actually how this talk came to be!

A brief history... (not to scale)

1979

chroot added to Unix - the ability to change the root of the filesystem for a process

2000

jail introduced to BSD, to enforce the concept of chroot for files and resources

Also, **SELinux** created to restrict how groups of processes can interact with other processes, files and devices

2001

Linux-VServer project introduced, a similar jail concept to Linux

2005

Seccomp - limit system calls a task can use (useful later)

2006

cpusets adapted to create “generic process containers” which were renamed control groups, or **cgroups**

2008

user namespaces - processes can have their own users, privileged users in a container

Also, **LXC from IBM** - tooling for containers, but relies on security incomplete Discretionary Access Control

2013

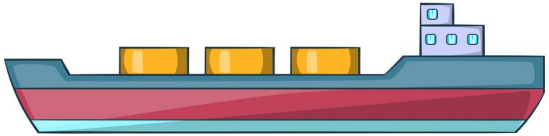
Docker is open sourced

2014

First release of **Kubernetes**

Okay but *why* containers?

"The modern shipping industry only works as well as it does because we have **standardized on a small set of shipping container sizes**.



...Instead of ships that specialize in bringing smartphones from Asia, we can just put them all into containers and know that those will fit on *every container ship*."

- Wtf is a container?

An application, decomposed

For an application to run, it needs…

Hardware

An application, decomposed

For an application to run, it needs…

Operating System
Hardware

An application, decomposed

For an application to run, it needs…

Middleware
Operating System
Hardware

An application, decomposed

For an application to run, it needs…

Dependencies to make the middleware work
Middleware
Operating System
Hardware

An application, decomposed

For an application to run, it needs…

Dependencies to make the application code binary work
Dependencies to make the middleware work
Middleware
Operating System
Hardware

An application, decomposed

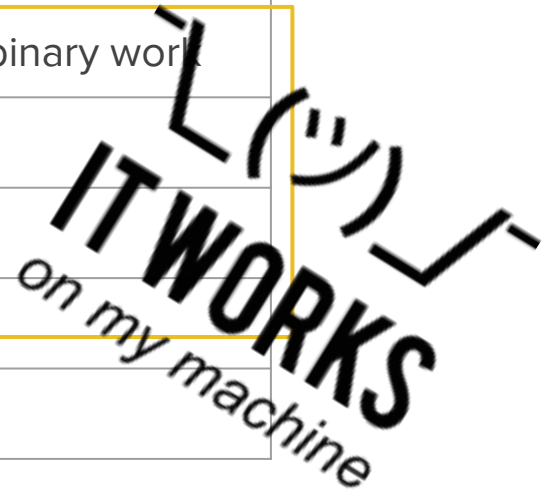
For an application to run, it needs…

Compiled and built application code binary
Dependencies to make the application code binary work
Dependencies to make the middleware work
Middleware
Operating System
Hardware

The Problem…

On a traditional VM or mainframe, this middle area causes…*uhh*, complication.

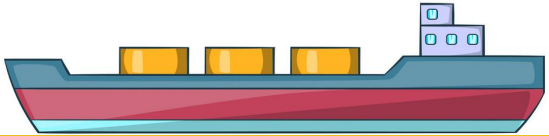
Compiled and built application code binary
Dependencies to make the application code binary work
Dependencies to make the middleware work
Middleware
Operating System
Hardware



The Solution! (or part of it, anyway)

Containers are amazing because they bundle all of *this* into **one standardized deliverable**:

Compiled and built application code binary
Dependencies to make the application code binary work
Dependencies to make the middleware work
Middleware
Operating System
Hardware

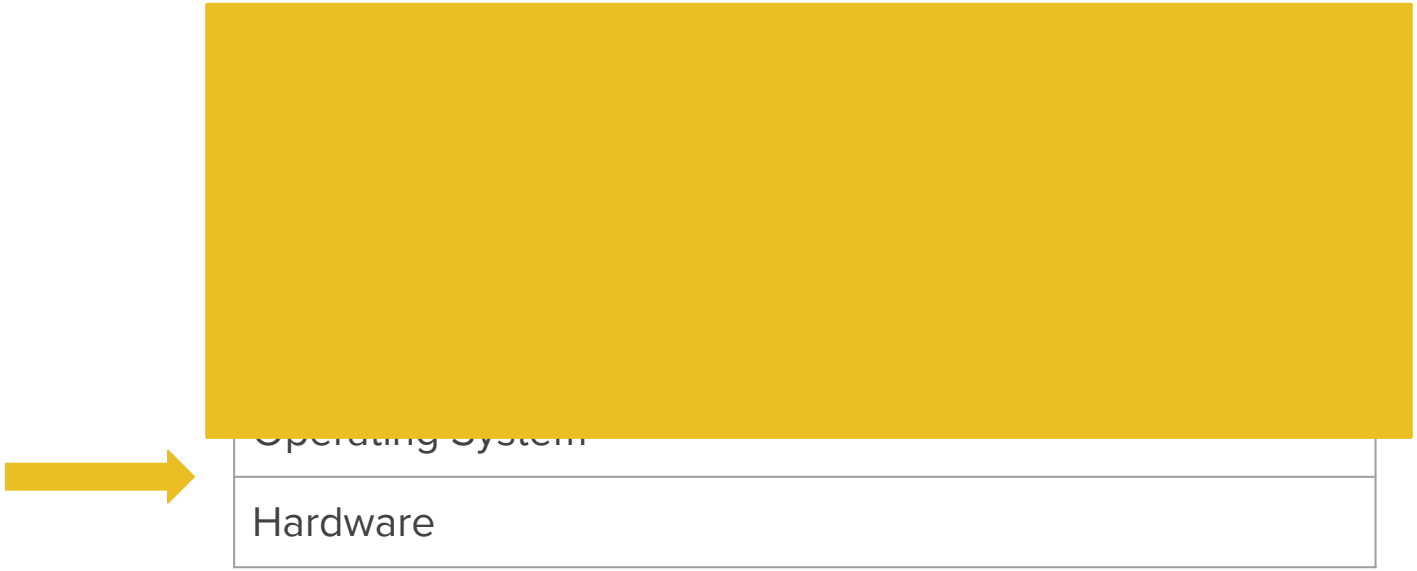


Where do containers run?

Where do containers run?

Compiled and built application code binary
Dependencies to make the application code binary work
Dependencies to make the middleware work
Middleware
Operating System
Hardware

Where do containers run?



Where do containers run?

There are two options:

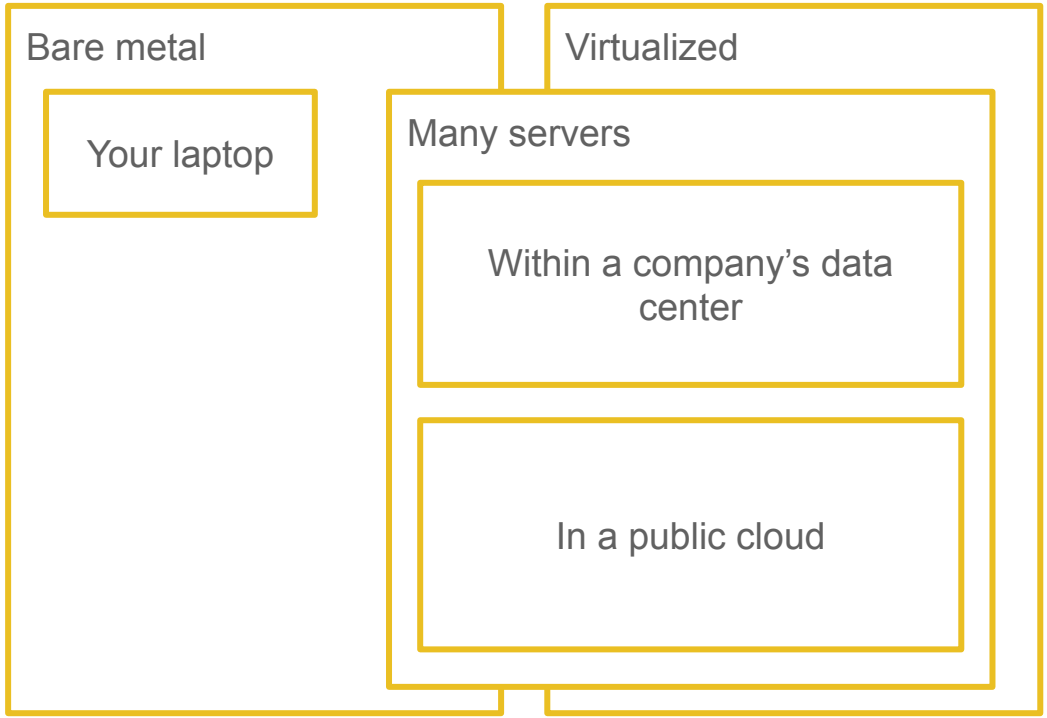
1. Linux
    a. *various…*
    b. *…flavors…*
    c. *…of…*
    d. *…Linux*
2. Windows
    a. *except this is super complicated…*

Where do containers run?

Those two options can further be swizzled to include *virtualized* environments (as opposed to bare metal).

And those virtualized environments can be “on prem” (in a company’s data center(s)) or in one of public clouds.

Where do containers run?



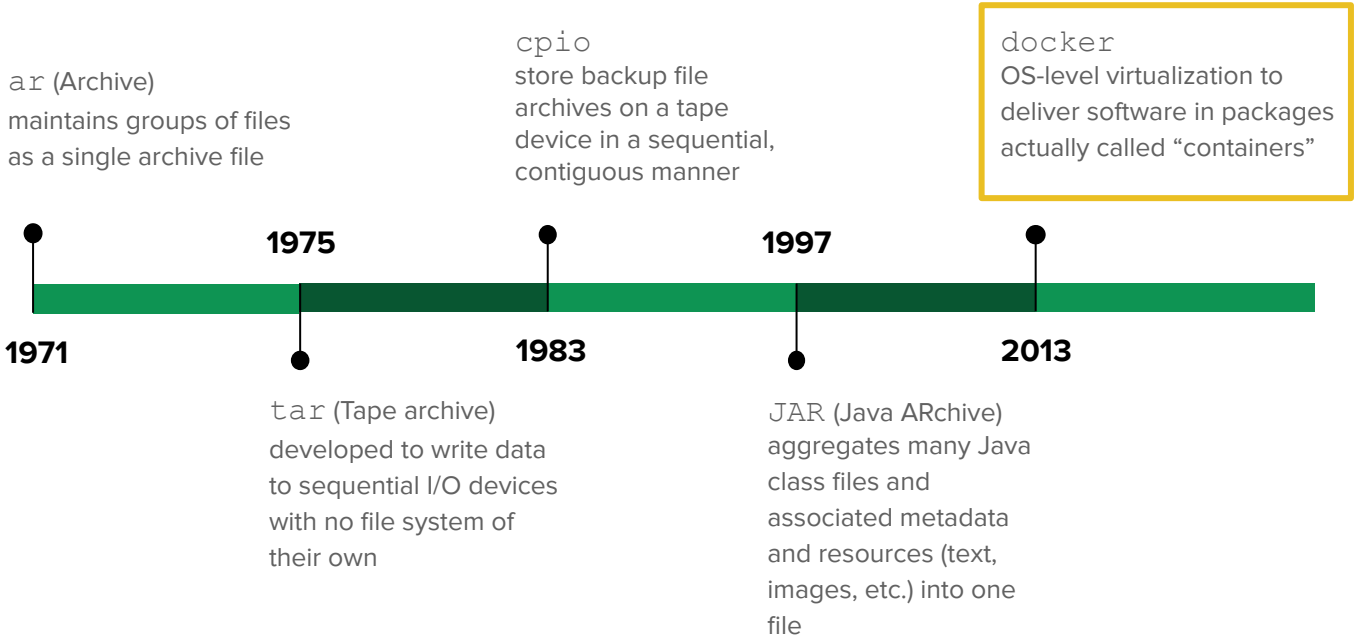
The Container Lifecycle and its Tools

Another brief history...

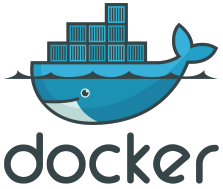
The purpose of the container lifecycle is to deliver... well, *something* in the format of a container.

Linux archives/containers/etc have been delivered in a number of ways over the years.

Another brief history... (still not to scale)



Why Docker?



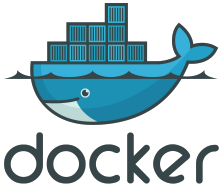
Standard, consistent format, consistent runtime, open source.

Brought together several pieces needed to create and manipulate containers into one complete package:

- Pull and push images to/from an image registry
- Make copies of images in local container storage and add layers to those containers
- Commit containers and remove local container images from the host repository

Article: Podman and Buildah for Docker users

Why Docker?



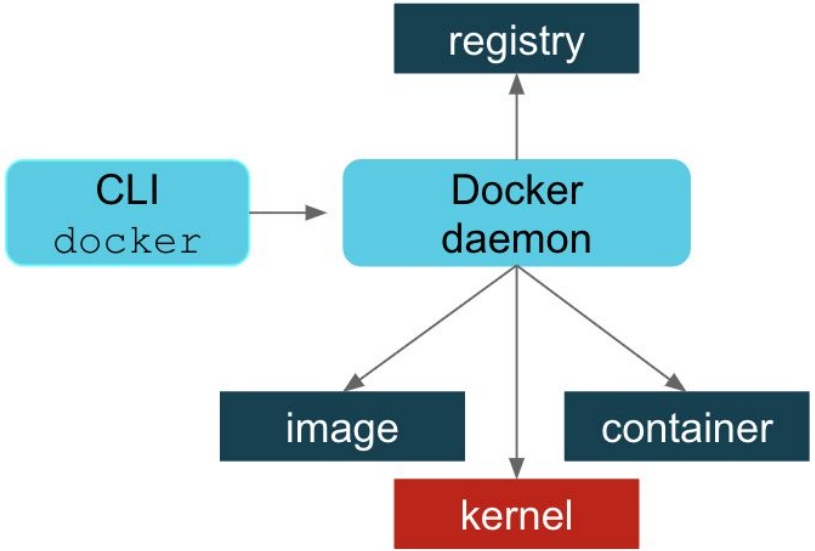
Standard, consistent format, consistent runtime, open source.

Brought together several pieces needed to create and manipulate containers into one complete package:

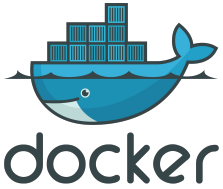
- Ask the kernel to run a container with the right namespace and cgroup
- Manage resources

Article: Podman and Buildah for Docker users

How Docker Works

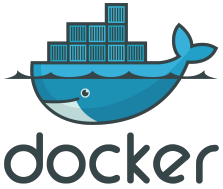


Why *Not* Docker (Technical Reasons)



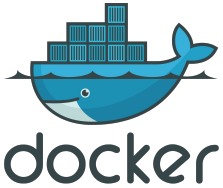
1. Complicated multi-level builds
   - required root access for all stages
   - had to be done in sequence, on only one host
     - one process, one point of failure
       - this parent process owned all child processes (running containers)
       - if a failure occurred, there were orphaned processes
   - required a Daemon

Why *Not* Docker (Technical Reasons)



2. Security concerns
   - root access means increased security risk
   - insecure registry - provenance and security cycles

Why *Not* Docker (Business Reasons)



Corporately - struggled to make money

This led to some wavering regarding their stance on keeping their tools open source.

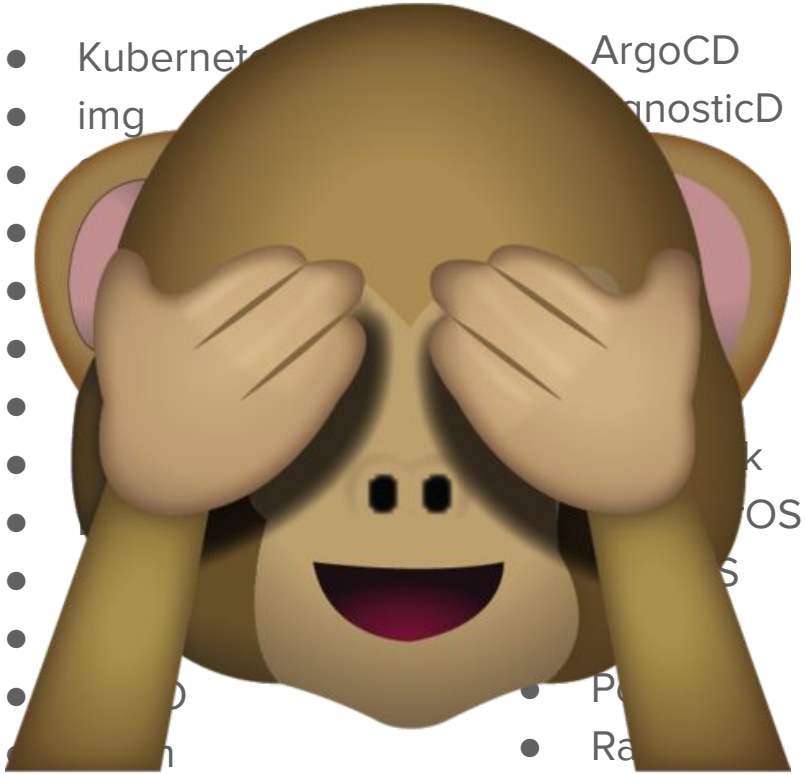
https://www.zdnet.com/article/docker-is-in-deep-trouble/
https://opensource.stackexchange.com/questions/5436/is-docker-still-free-and-open-source

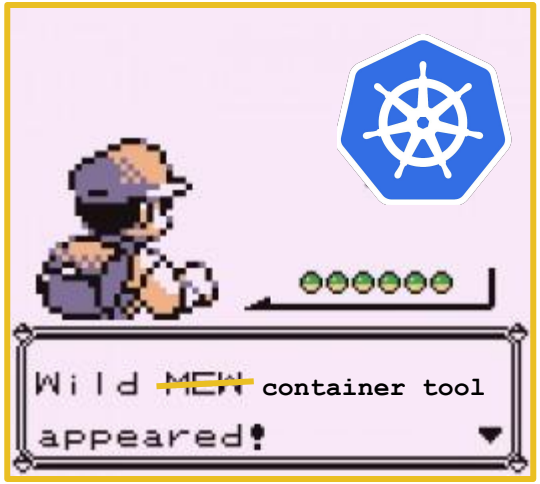
So... New Tools Needed!

- Easier to use
- Reflective of the container lifecycle and increasing adoption
  - building containers at scale
  - distributing containers at scale
  - *running* containers at scale

So... New Tools Needed!

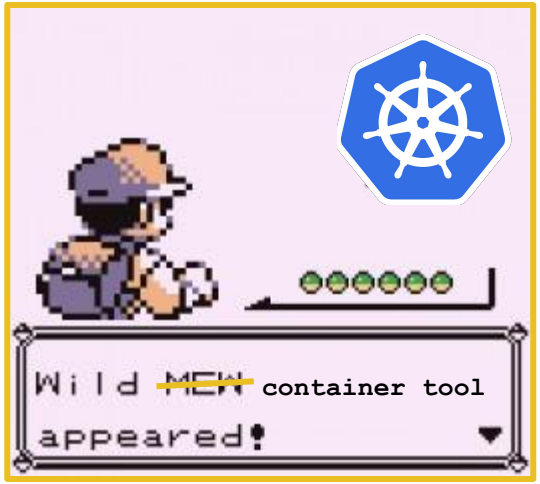
- Kubernet
- img
- ArgoCD
- nosticD
- k
- OS
- S
- D
- n
- P
- Ra





So... New Tools Needed!

- Kubernetes
- img
- S2I
- Buildah
- Ansible Container
- Knative
- Bazel
- kaniko
- Buildkit
- umoci & orca
- OCI
- CRI-O
- Helm
- ArgoCD
- AgnosticD
- Moby
- k3s
- containerd
- Istio
- Mesos
- Twistlock
- RancherOS
- CoreOS
- rkt
- Podman
- Rancher



So… New Tools Needed!

Some of these tools have emerged as the leaders in their respective categories, but…the landscape continues to evolve.

So…what problems are we solving?

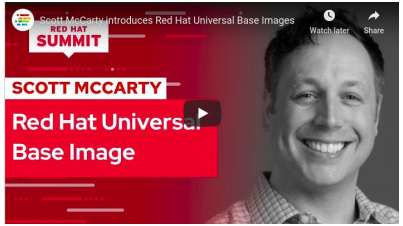
1. Develop
2. Build (Create *or* Update)
3. Store/Archive/Version
4. Deploy
5. Run
6. Move

The Container Lifecycle

1. Develop
2. Build (Create *or* Update)
3. Store/Archive/Version
4. Deploy
5. Run
6. Move

Foundational Tools

- OCI: specification for how to standard-ly run containers
- UBI: free, OCI-compliant base OS images plus frequently used runtimes



Foundational Tools

- CoreOS: minimal Linux distribution, focused on only what's needed for containers
- Kubernetes: Container orchestration and management

Develop

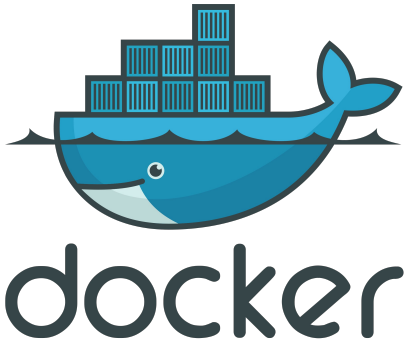
- Eclipse Che: Kubernetes-native IDE
- VSCode + Marketplace

What happened with Docker?

- Docker: build, run, and move

- Podman: “daemonless container engine for developing, managing, and **running** OCI containers”
- Buildah: “efficiently and quickly **building** OCI-compliant images and containers”
- Skopeo: **moves**, inspects, signs images

What happened with Docker?



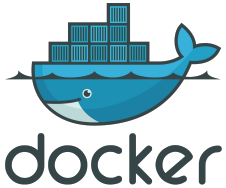
buildah

podman

skopeo

Also...

Build



CRI-O: lightweight container runtime

S2I: produces ready-to-run images via self-assembling builder images

Knative: builds and manages Kubernetes-based serverless workloads

Store/Archive/Version

Skopeo (inspect)

Quay - container registry

Deploy

- Docker
- Podman
- Buildah
- Kubernetes
- Knative
- OpenShift/OKD

Run

- Kubernetes
- Knative
- OpenShift/OKD

Move


Skopeo

Docker


Podman


Buildah

Tangential Tools

Other things containers need…

7. Lifecycle Automation
8. Security

Lifecycle Automation

UBI

Jenkins (mostly build and deployment)

Ansible (mostly infrastructure setup)

Automation is important for consistent, repeatable, auditable builds and deployments. If you were in our DevSecOps workshop on Tuesday, you already heard these words!

Security Comes in Layers

Compiled and built application code binary
Dependencies to make the application code binary work
Dependencies to make the middleware work
Middleware
Operating System
Hardware

Security (Open Source)

UBI

SonarQube (code scanning)

Clair (container dependency scanning)

Demo!

Demo!

https://github.com/lainie-ftw/demos/blob/master/container-tools-tng/container-tools-demo.sh (if you want to try it yourself!)

What did we miss??

What We Missed

Alpine

More questions?

Thank you!

Josh:
josh@soul-repairs.com
@architect_josh

Laine:
laine@soul-repairs.com
@lainie_ftw

soul-repairs.com