Containerization

What are Containers?

Why Containers?

Containers != VMs

Containers != Microservices

Just a little bit of How

Many of the images in this presentation taken from Docker's public blog,

What are Containers?

The Cutty Sark, Greenwich UK

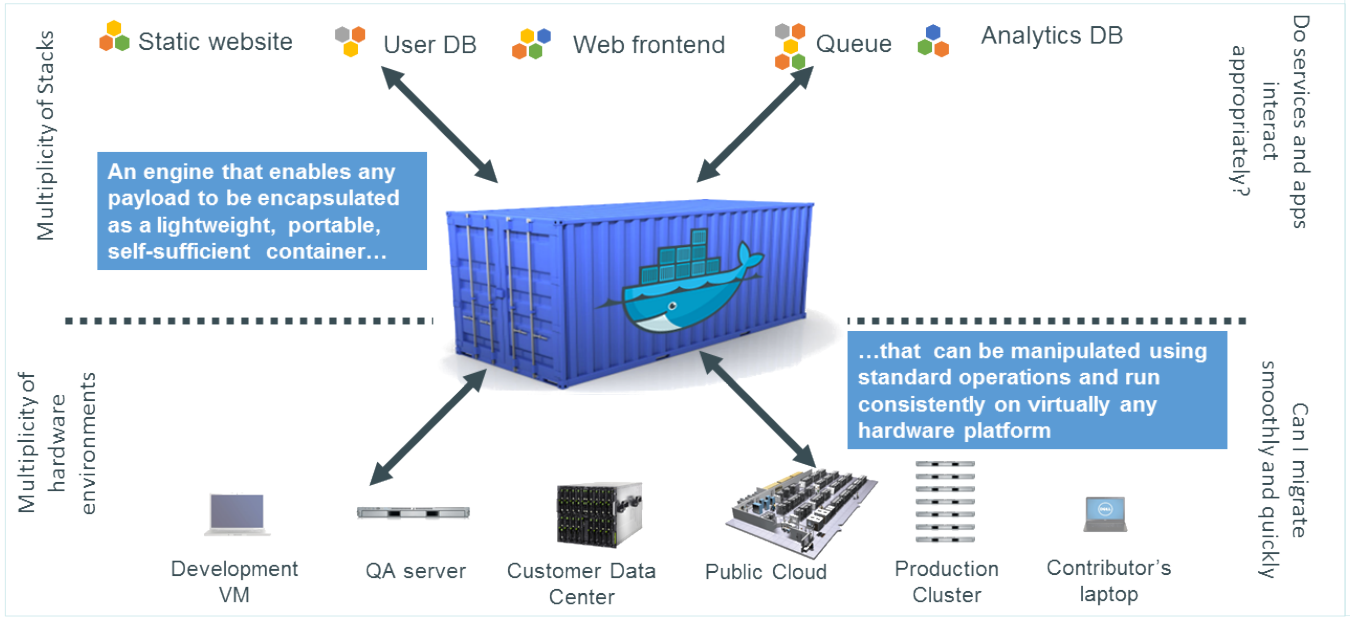
Until the 1950s, international shipping at sea was done by packing wooden crates into the lower cargo holds of large sailing vessels. These were packed and unpacked by for-rent longshoremen, hundreds at a time. It was slow and expensive and difficult to schedule

In 1956American entrepreneur Macolm McLean invented the shipping container, a large rectangular, transportable stackable crate, resulting in a 300% speed increase in load/ unloading and a 97% reduction in the cost of cargo handling [1]

[1] According to https://kkcontainer.com/shipping-containers-globalize-world/

What are Containers?

Shipping containers can be loaded with anything, and be stacked for storage or transport via multiple modalities (trucks, rail, ships, etc.)



A software container provides a bundling mechanism that can serve as a build target, dev env, deploy env, etc. It can be hosted by any number of physical computing envs

What are Containers?

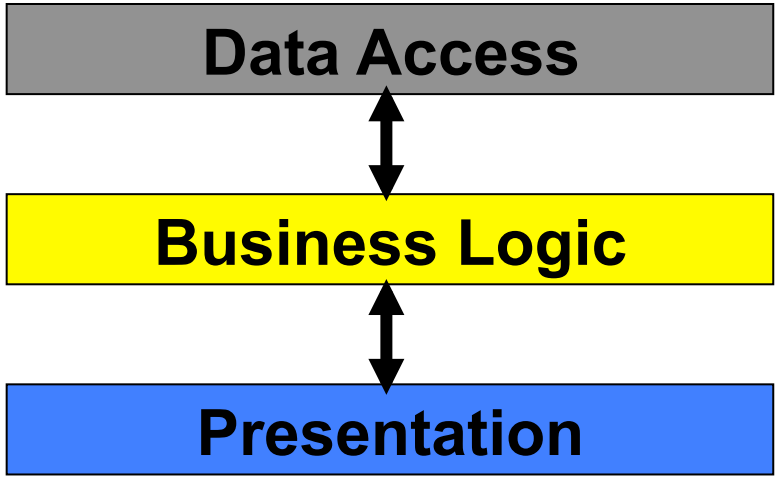
According to Docker, the company that fostered a new marketplace for container technology, a container is

> *"… a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings."*

OK, let's parse this:

- *Lightweight* – that word again. It is usually used relative to something else, and implies speed, minimal resource requirements, and minimal dependencies
- *Stand-alone* – means independence, or a more commonly used term is *isolation*. This means the container can operate with dependencies on outside software (besides the container platform)
- *Executable package of a piece of software* – the "piece" brings in *size* and *decomposition (modularity)*, while *executable package* menas it runs (it is a program) and is *assembled* in some way.

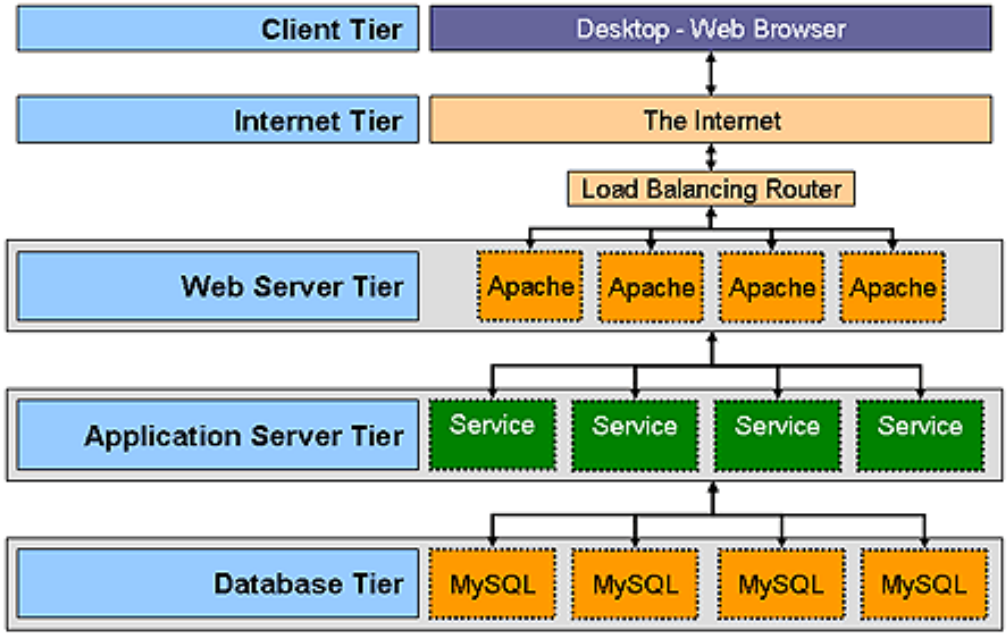
Why Containers?



Recall in our last discussion, we said “horizontally partitioned” systems driven by traditional Separation of Concerns (SoC) tainted our software architecture.

Well, it tainted deployment architectures too! Runtime responsibilities factored over all shared resources.

- Deployment complexity
- Update complexity
- Monitoring/logging complexity
- Capacity planning complexity

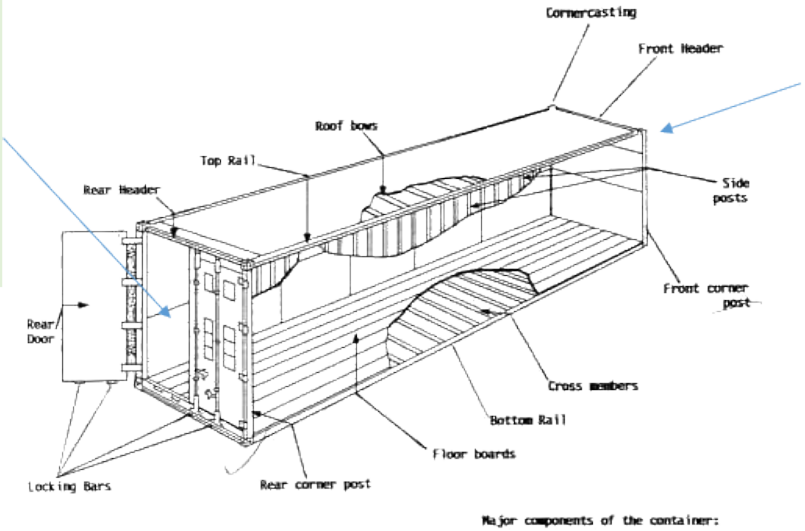


www.scottburkett.com

Why Containers?

- Dan the Developer
  - Worries about what's "inside" the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same

SoC in container-land



Major components of the container:

- Oscar the Ops Guy
  - Worries about what's "outside" the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

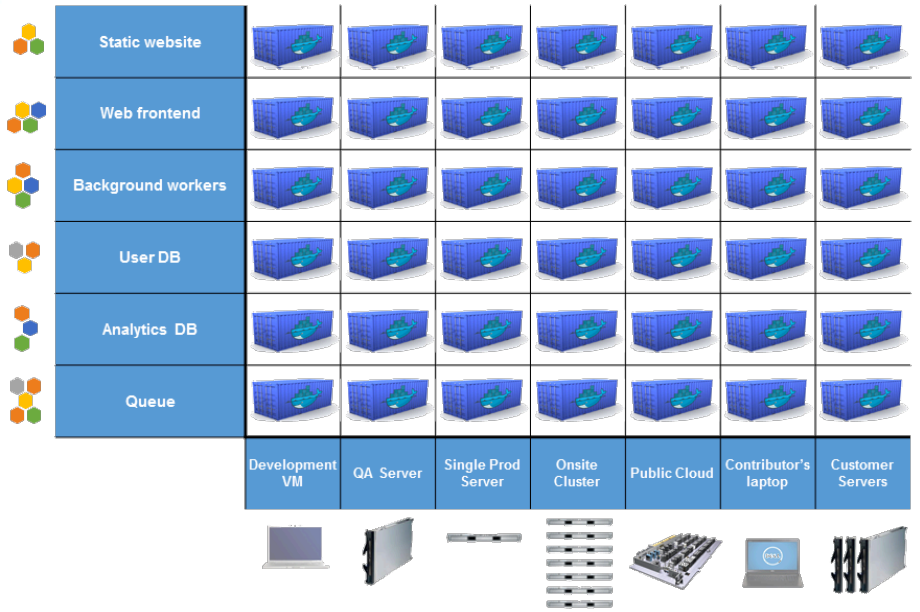
Dr. Gary's take: *(stay tuned for the coming DevOps discussion)*

- For a long long time web prganizations struggled with 1) the role of the engineer in production, and 2) the skills ops personnel need
  - Not many SEs wanted to work in ops
  - Not many ops really knew how to troubleshoot distributed apps
- This different slant on SoC gives a pathway for engineers to support all the way through production, and a consistent role for ops to support all kinds of containers coming their way

Why Containers?

The Matrix of Hell

On Docker's blog they refer to this quagmire of deploying across tiers and across staging environments as the "Matrix of Hell"



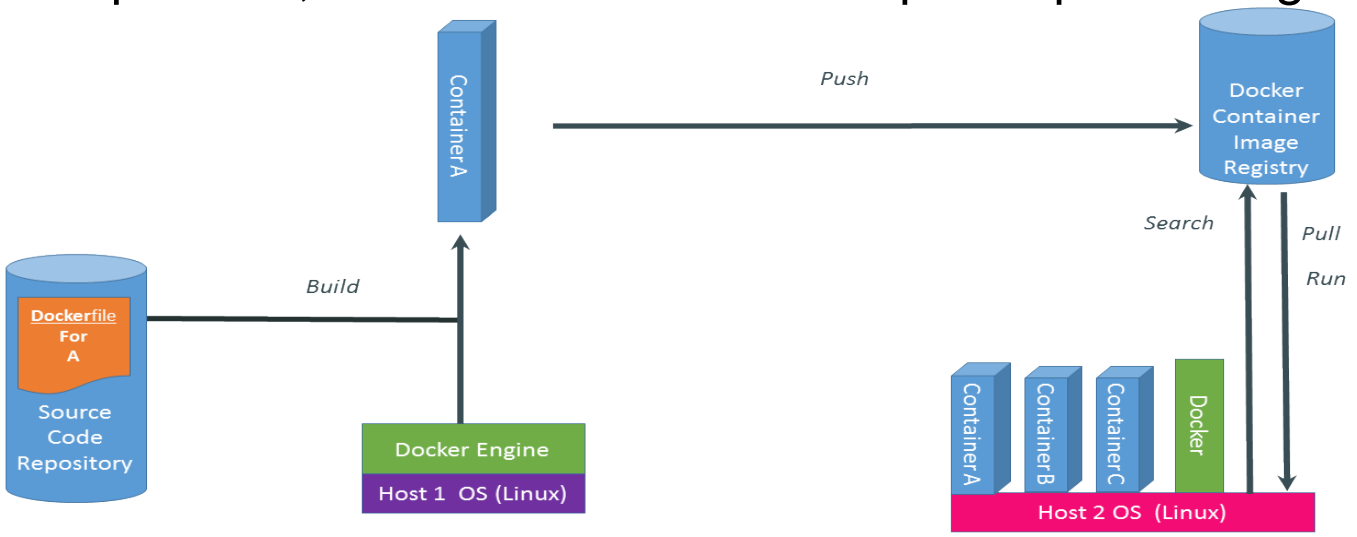
Revisiting what we said up front, the container becomes the target for all envs

- We "build" the container, not the app/service package
- We deploy it everywhere
  - Take that Java!

Just a little bit of How

How does Docker work?

- We don't care too much, but it helps to have a sense of the architecture and the terms/processes that go with it:
  - A *dockerfile* is basically a script for instantiation in a *container*
  - The *docker engine* running on a host OS instantiates containers
    - A container has a lifecycle, it may startup for one task or as interactive
  - A docker *image* may be pushed to a registry for reuse by others
  - A container may be updated, which docker supports by getting only diffs between apps, not requiring a rebuild or redeploy of the image
- You interact with Docker via a client, like the CLI. The engine runs in a background process, and communicates with public/private registries



Containers != Virtual Machines

Isn't a container just a "lightweight" virtual machine?

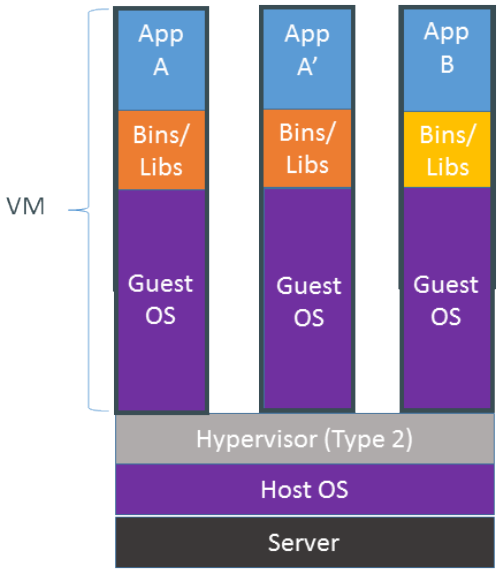
- No, several important differences between containers and VMs

Container

- Shared resources
- No hypervisor licensing
- Host OS (which you optimize)
- Fast instantiation
- Incremental app upgrades

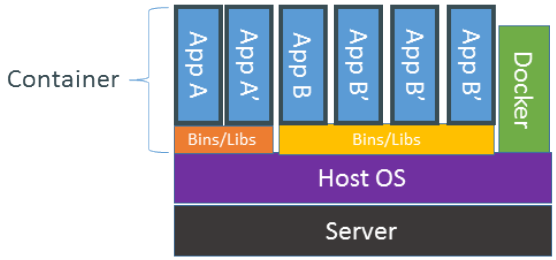
Virtual Machine

- Isolated resources
- Hypervisor licensing
- Guest OS on top of hypervisor
- Slow instantiation
- Rebuild entire image



Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



Containers != Microservices

Microservices are an Architecture, Containers a technology

If you are using Containers are you doing Microservices?

- No! Containers will happily support your monolith, either in its entirety or a horizontal partition (layer) at a time
  - And there are some separate use cases where that makes sense!

But Microservices and Containers go together like so many of the good things in life

- Fast instantiation matters for elasticity and small behaviors
- Lightweight use of resources matters for scale
- Isolation provides transport flexibility and the ability to use the same container from dev through prod.
- Increasing tool sophistication is advancing automation

  (…stay tuned for the Continuous Deployment discussion)

…and there are some advanced capabilities with containers (swarming, composing, etc.) that we won't get to here...