Java. Cloud. Leadership.

Container-less Development or Immutable Containers?

Mark Little, JBoss CTO, Red Hat VP

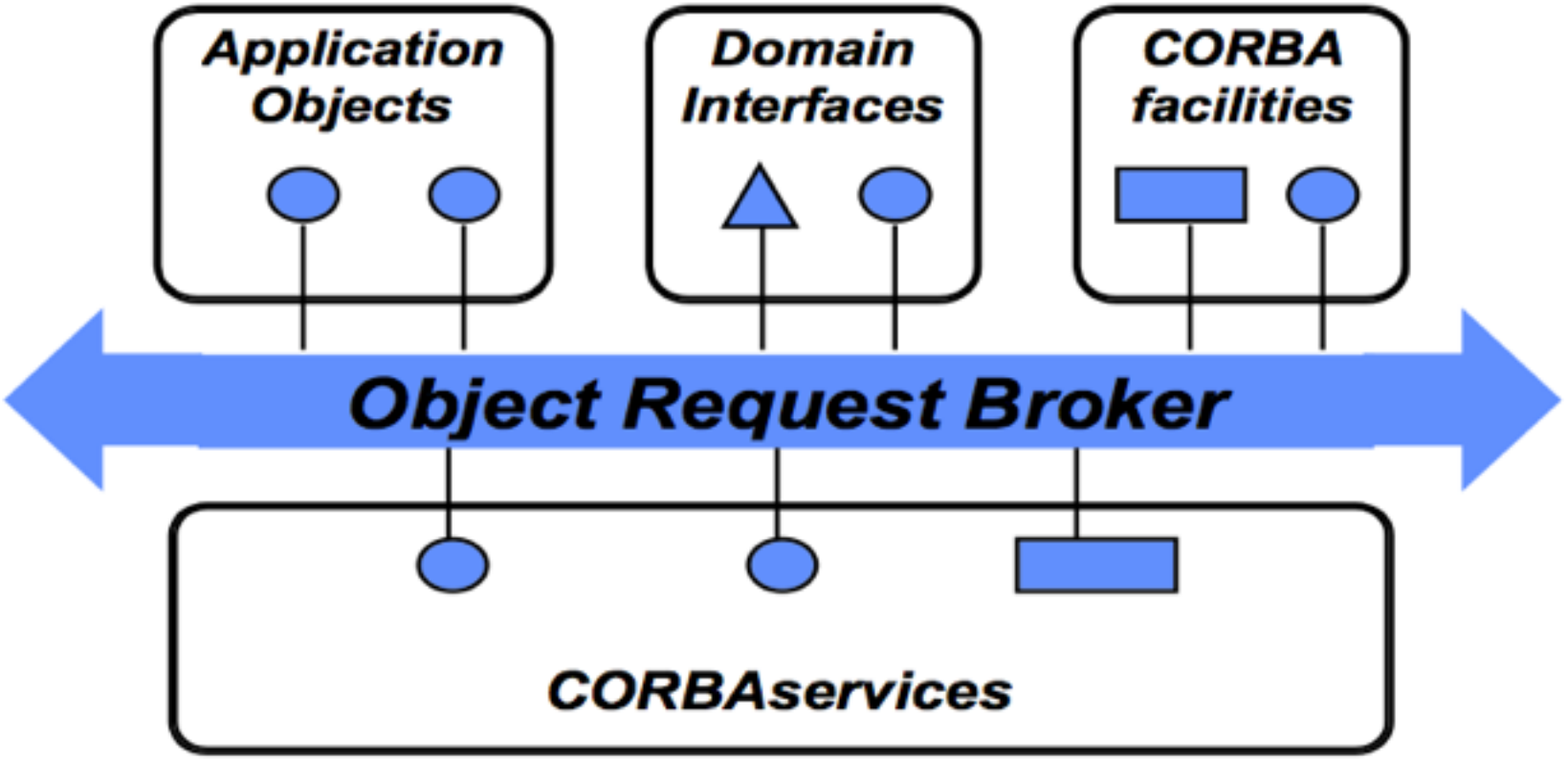
Overview

- Application containers have dominated Java middleware for almost 2 decades
  - Simplifying complex requirements
- Linux containers and immutable infrastructure forcing a change in mindset
- Combined with fat jars and microservices
- Do application containers have a role?

Distributed systems archeology

- 1960's-1990's client-server dominates
  - Typically single-threaded
  - Multi-threading in languages rare
- Core services/capabilities begin to emerge
  - Transactions, messaging, storage, …
- Message passing gives way to RPC (1970)
  - Leverage a well known pattern: local procedure calls

Typical 80’s/early 90’s architecture



And then …

- Late 90's/2000's
  - New generation of chips, e.g., M68030, SPARC, Xeon, Itanium
  - Multi-core, hyper-threads
- RAM sizes "explode"; access times too
  - 64 Meg in Sun 3/80, 512 Meg Pentium 3
- Network speeds improve more slowly

Rise of the application container

- Prior to application containers developers managed a lot
  - CORBA precursors in ORB and Object Adapter
- Improvements encourage co-location of capabilities/services
  - Improve performance & memory footprint
  - Threading becomes the norm
- Not just a Java concept either
  - CORBA Component Model anyone?

Benefits of Java application containers (servers)

- Application containers - taken for granted
  - Thread pooling
  - Connection pooling
  - Transaction management
  - Servlets
  - Logging
  - Inter-service dependencies
- Frameworks can help manage some of this too

Application server backlash

- Not as simple to develop the “easy stuff”
  - Classpath hell?
- “Containers are pure evil!”
- Application servers viewed as bloated
  - Not everyone wants all enterprise services
- OSGi and others evolve to address dynamic updates
- Java EE profiles were an improvement

“Java EE is too bloated”

- Differentiate the standard from implementations!
- It is possible to be lightweight and enterprise ready

The Open Source Java application server *reignited*

Designed for flexibility.
Amped with electrifying speed.
Launch your Java EE applications in a flash!

Lightning Fast... *start-up / deployment / configuration*

WildFly

Java EE stripped down

- Many developers are happy with Java EE
  - Robust and mature components; well understood
  - Scalable, standards compliant, integrates well
- Not everyone wants to use all of Java EE
  - Stripping down is common
- Ditch the container to use services “raw”?

For instance WildFly Swarm

- Allows Java EE components to become independently deployable services
  - Applications deploy with only the components needed
  - Just enough Application Server (JeAS)
- Self-contained services
  - Build applications as fat jars (Java circa 1996)

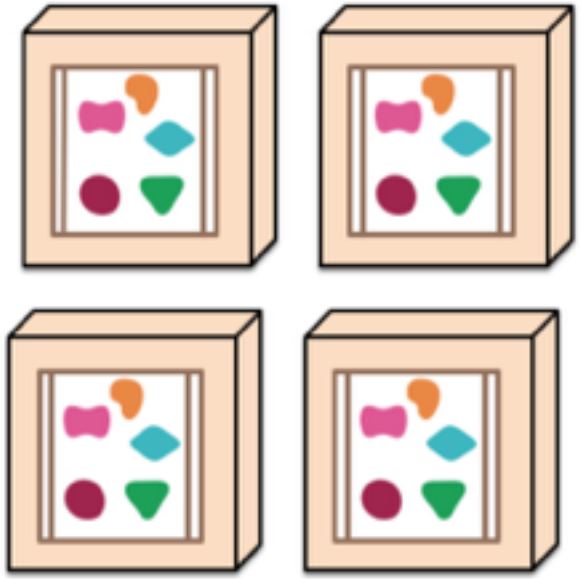
But is it container-less?

- The container is still present
  - WF Swarm just abstracts it further
    - Removes those services you don't need at build time
- Adam Bien …
  - http://www.adam-bien.com/roller/abien/entry/do_you_know_any_container
    - "Without container concepts the business logic and generic infrastructure would be intermingled."
- Is this a non-problem?
  - If application can still boot in under a second
  - And has minimal memory footprint

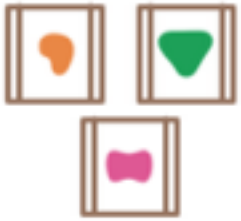
Along come microservices

A monolithic application puts all its functionality into a single process...

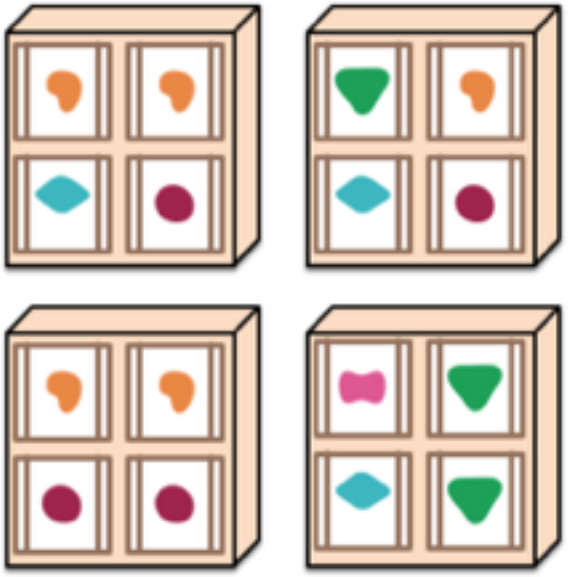
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...

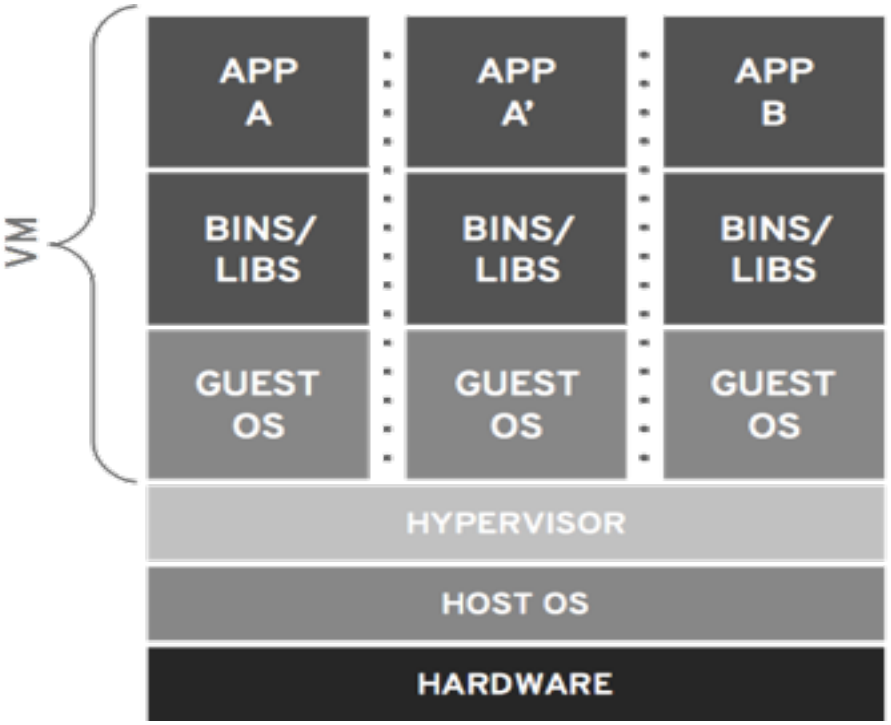


... and scales by distributing these services across servers, replicating as needed.

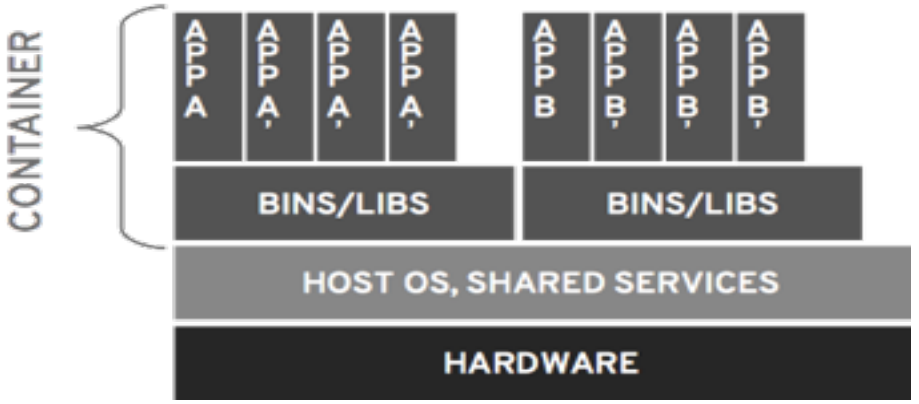


Followed closely by Linux Containers

VIRTUALIZATION



CONTAINERS



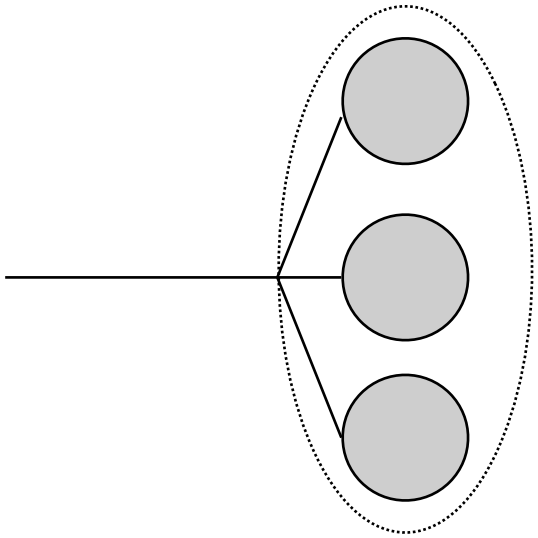
Kubernetes

- Open source project from Google
- The de facto standard for cluster management for Linux containers
- Packages Orchestration, service discovery, load balancing – all behind a simple REST API
- Backing from Google, IBM, Red Hat, Microsoft, Rackspace, Cloudbees etc.

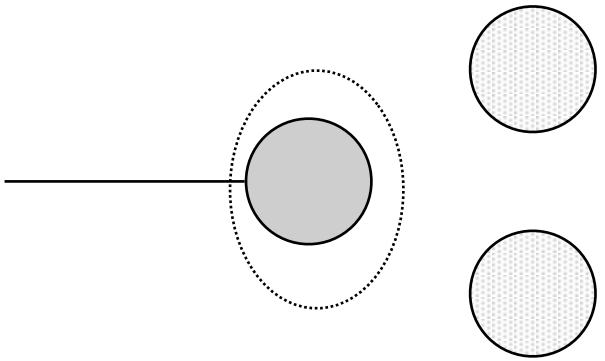
Kubernetes and Containers

- Kubernetes assumes immutability of images
  - Any changes to a running image are lost
  - Changes can still be made but volatile
- State must be stored off image
  - Shared (persistent) volumes
  - Non-Container services etc.
- Persistent changes require new container image
  - Or could be handled with persistent volumes

Immutable (stateless?)

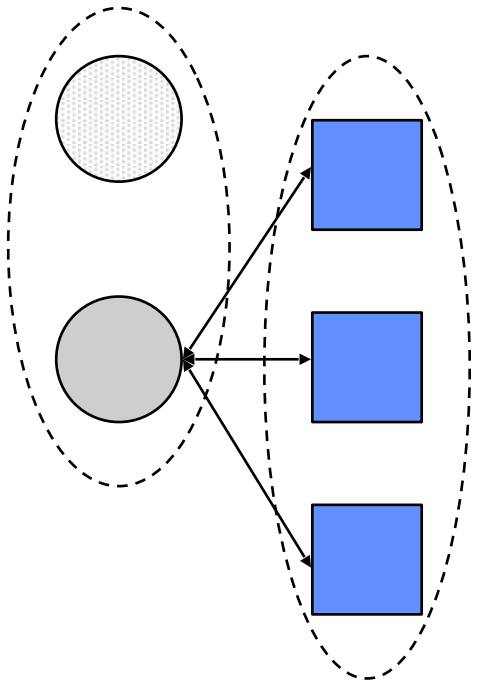


Active

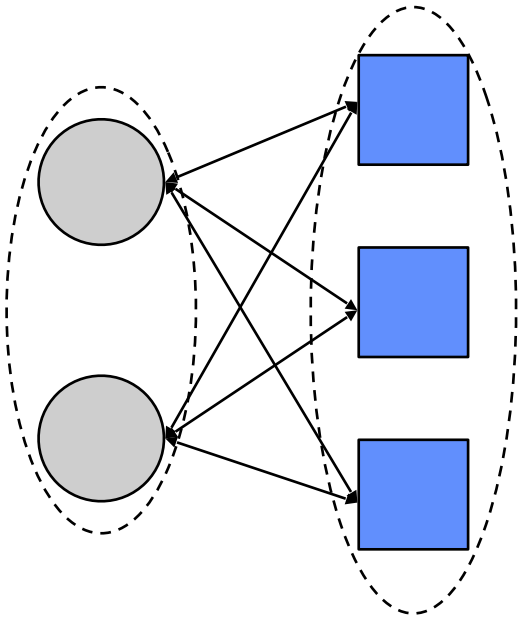


Single-copy passive

Mutable (stateful?)



Passive Replication



Active Replication

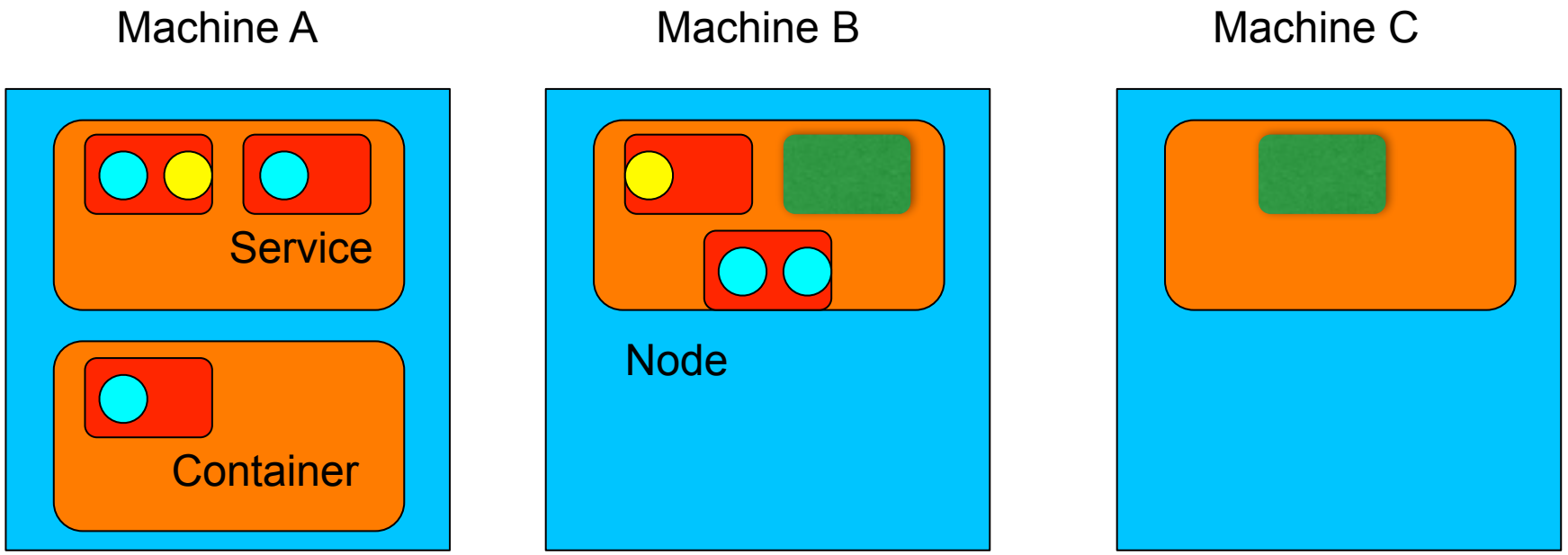
Immutability simplifies architectures

- Dynamic reconfiguration?
  - New container images can be created quickly (?)
- Rip out some application container code
  - Update needed? Create new image and redeploy!
  - What about connection pools, thread pools, ..?
- Strip down the application container
  - Just Enough Container?

Enterprise capabilities

- However, the need for transactions, reliable messaging etc. doesn't go away
  - Applications still need them
- Application containers breaking into pieces
  - Independently deployable (Linux container) services
- Available to different language clients using REST/HTTP and other protocols
  - Still A LOT slower than IPC!

Services, Linux containers and JVMs



But …

- Java developers are used to quick develop and redeploy cycles
- Red Hat Summit keynote demo 10000 Linux containers in seconds
  - But typical dev-to-deploy cycles are getting shorter and shorter
  - Memory footprint impact
- Immutability makes sense closer to the final solution

Distributed Linux containers

- Linux containers form distributed systems
  - Fault tolerance?
  - Management?
  - Performance?
  - Availability?
  - Reliability?
- Approaches such as Kubernetes can help
  - Google run containers at scale!
- But distribution WILL have an impact

Application containers and monoliths

- Using an application container doesn't mean you're building a monolith!
- Application containers can help separation of concerns
  - Do you really want to deal with thread pools?
  - Do you really want to deal with dependency injection?
- Likewise using a fat jar doesn't mean you're using microservices!
- Understanding your architecture is key!

Balls of mud made of services

"**If you're building a monolithic system and it's turning into a big ball of mud, perhaps you should consider whether you're taking enough care of your software architecture.** Do you really understand what the core structural abstractions are in your software? Are their interfaces and responsibilities clear too? **If not, why do you think moving to a microservices architecture will help?** Sure, the physical separation of services will force you to not take some shortcuts, but you can achieve the same separation between components in a monolith." http://www.infoq.com/news/2014/08/microservices_ballmud

Conclusions

- Linux containers driving a new approach to development and deployment
  - Immutability must be an architectural consideration
- Application containers being stripped back
  - Enterprise services still needed!
  - Not going away quickly though
- Don’t lose sight of good software engineering principles and practices
  - Start with the architecture not software!