# Leveraging Lessons from the Cloud

## Strategies every system can benefit from

Jayson Raymond, Accelerant Mobile Corp.

# Motivation for the Cloud

*Need a massive internet presence that doesn't cost you monthly?*

**Amazon turned a cost center to a $6B/yr profit center**

# Key Principles

Economies of Scale

Applying Principles of RAID to the Data Center

- “Redundant Array of commodity hardware”
- A.k.a. Horizontal Scaling in contrast to throwing fancier hardware at the problem (vertical scaling)

Elasticity - on demand horizontal scaling out and back in

# Cloud Concepts

**Deployments:** Private, Hybrid, Community, Public

**Workloads:** Static, Periodic, Once-in-a-lifetime, Unpredictable, Continuously Changing

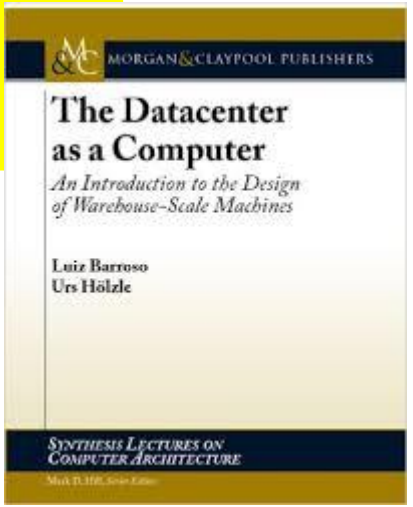
**Service Provided:** Infrastructure, Platform, Software, or ‘Metal’ as a Service

**Isolation:** Virtualization vs. Containerization

# Common Architecture Concerns

- Scalability
- Availability
- Resilience
- Cost-aware
- Secure
- Manageability
- Vendor lock-in awareness
- Time-to-Market

But addressed on a larger scale



# Patterns






OPEN DATA CENTER ALLIANCE℠ BEST PRACTICES:
ARCHITECTING CLOUD-AWARE APPLICATIONS REV. 1.0

# Scalability

- Request Queueing
- Queue-based Workflow
- Request Collapsing
- Stateless Services
- Caching (especially HTTP)
- Microservices
- Data Grids

# Scalability: Microservices

Spread the application across many nodes allows:

- applying more resources to the parts of the app that need it, when it’s needed
- ‘canary’ testing of a new version of a component by deploying it to just a few nodes in the cluster simultaneous with the previous version

Drawback: every component that depends on another deployed to another machine incurs a latency penalty

# Scalability: Data Grids

**Distributing** data across nodes allows horizontal scaling: to increase memory - simply add more commodity hardware.

**Replicating** data across nodes requires vertical scaling: To increase memory requires more costly machines capable of holding more memory.

# Scalability: Data Grids

Look to products like:

- Hazelcast
- Gridgain
- Tibco ActiveSpace
- JBoss Infinispan


But not:

- Ehcache, Redis

# Manageability Patterns

**Managing the scale:** With many servers, coming and going, how do you:

- Find a node with the services you depend on
- Synchronize state across the nodes
- Keep all nodes configured

# Finding Services

Traditionally finding another local service was done using DNS.

**Problem:** In elastic environments, cached DNS results are quickly out of date and too costly to query latest

**Solution:** Local background monitoring of which nodes dependent services are on has an accurate answer when the application needs it

# Finding Services

**Implementations**:

**Google Kubernetes** - via Environment Variables

**Hashimoto Consul** - DNS or HTTP

**Skynet SkyDNS** - DNS (atop Etcd)

**Netflix Eureka** - Java library

# Configuration

**Manageability:** Information must be synchronized across cluster even as it scales elastically

**Availability:** Configuration updates should instantly propagated without restarts

# Configuration

**Implementations**

***Netflix Archaius***

Dynamic updates to app, without shutdown:

- **Netflix Archaius**: JMX interface, Typed

Clustered replicated store with notifications:

- **Apache Zookeeper**: Java-library
- **CoreOS Etcd**: HTTP Service

etcd

# Resiliency Patterns

**Circuit Breaker:** Specify fallback strategies that should be taken once a certain level of errors occur.

**Caching:** Rest-based Microservices can use offline caching in their HTTP client when dependant services are down. Reduces costs by rducing bandwidth required.

# Resiliency Patterns

**Netflix Hystrix:** is the classic implementation of the Circuit Breaker pattern.


**Apache HTTP Components:** The HTTP client can cache. If backed with a distributed store, the cache can be shared amongst all nodes.

# Virtualization vs. Containers

**Docker vs. KVM:** (source: IBM Presentation on YouTube)

- 9.5x more efficient cpu at steady state
- 6x less disk usage
- 2x higher file io random r/w

# Virtualization vs. Containers

*Provisioning Time Comparison (source: Linux Journal)*

| | ***Time*** | ***State*** |
|---|---|---|
| Real Machines: | 8-24 hours | Stateful |
| Virtual Machines: | 5-10 minutes | Stateful |
| Containers w/ | | |
| Copy-on-Write: | 5-15 seconds | Stateless |
| Overhead: < 2% | | |

# Application Container


**Docker**: Started it

**appc**: The open spec - driven by CoreOS

**rkt**: CoreOS's first appc implementation

**Kurma**: Apcera's implementation of **appc**

# Kubernetes

Manages containerized applications across multiple hosts providing application deployment, maintenance & scaling.

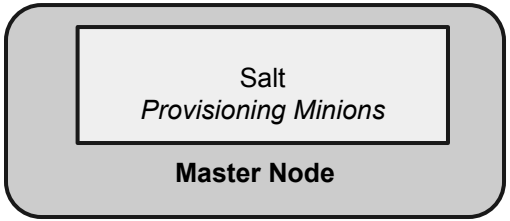
A product of “a decade and a half of experience at Google running production workloads at scale, combined with best-of-breed ideas and practices from the community.”

Kubernetes is:

- **open source**
- **portable**: public, private, hybrid, multi cloud
- **self-healing**: auto-placement, auto-restart, auto-replication

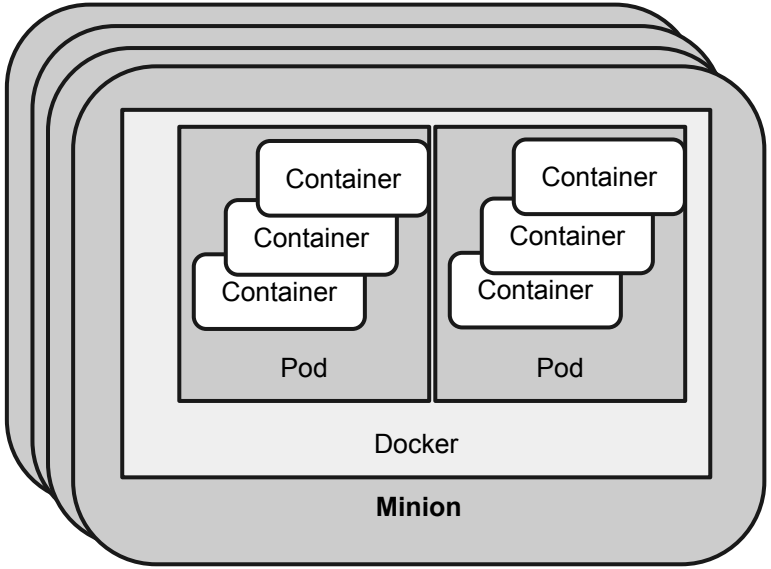

Provides deployment affinity (“pods”), ensuring apps are deployed on the same node solving the Microservice added latency problem.

# Kubernetes Components

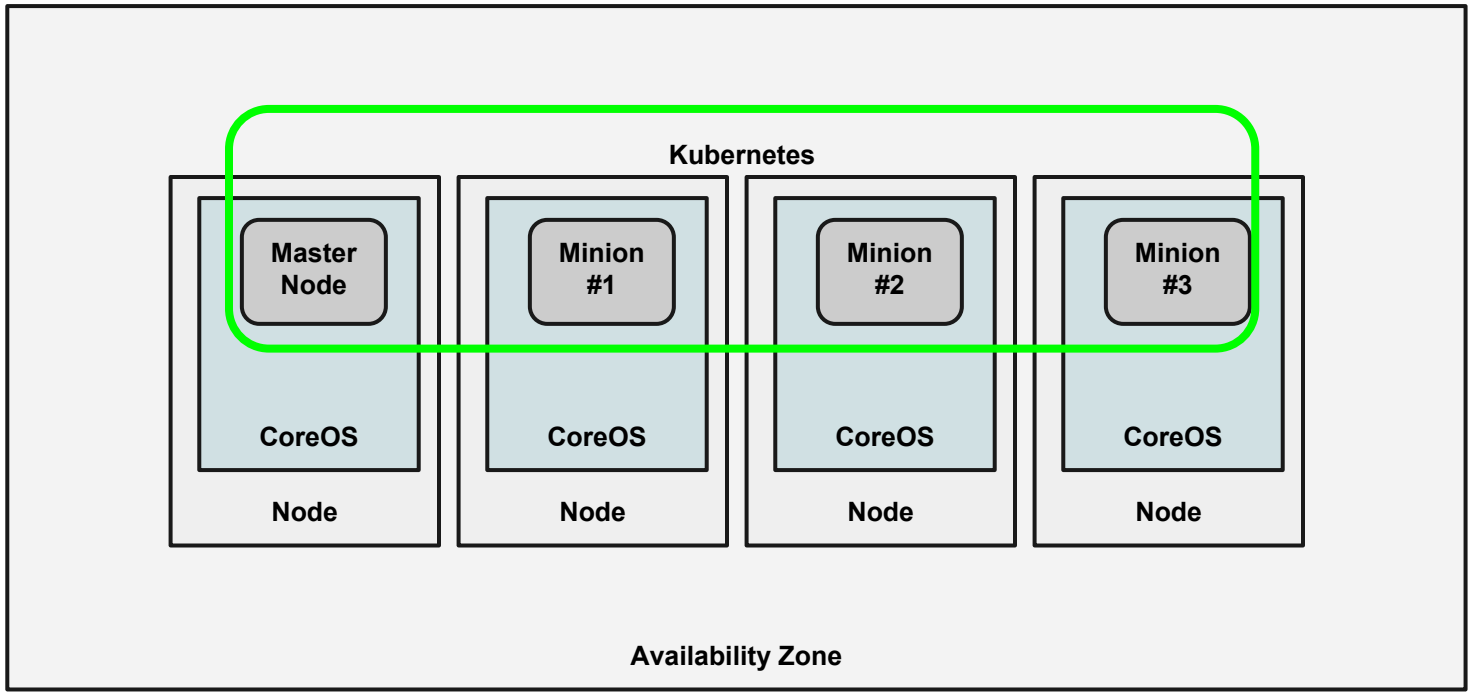



**Replication Controller**
*Ensures requested number of Pods are running, created from Pod template including health monitoring*

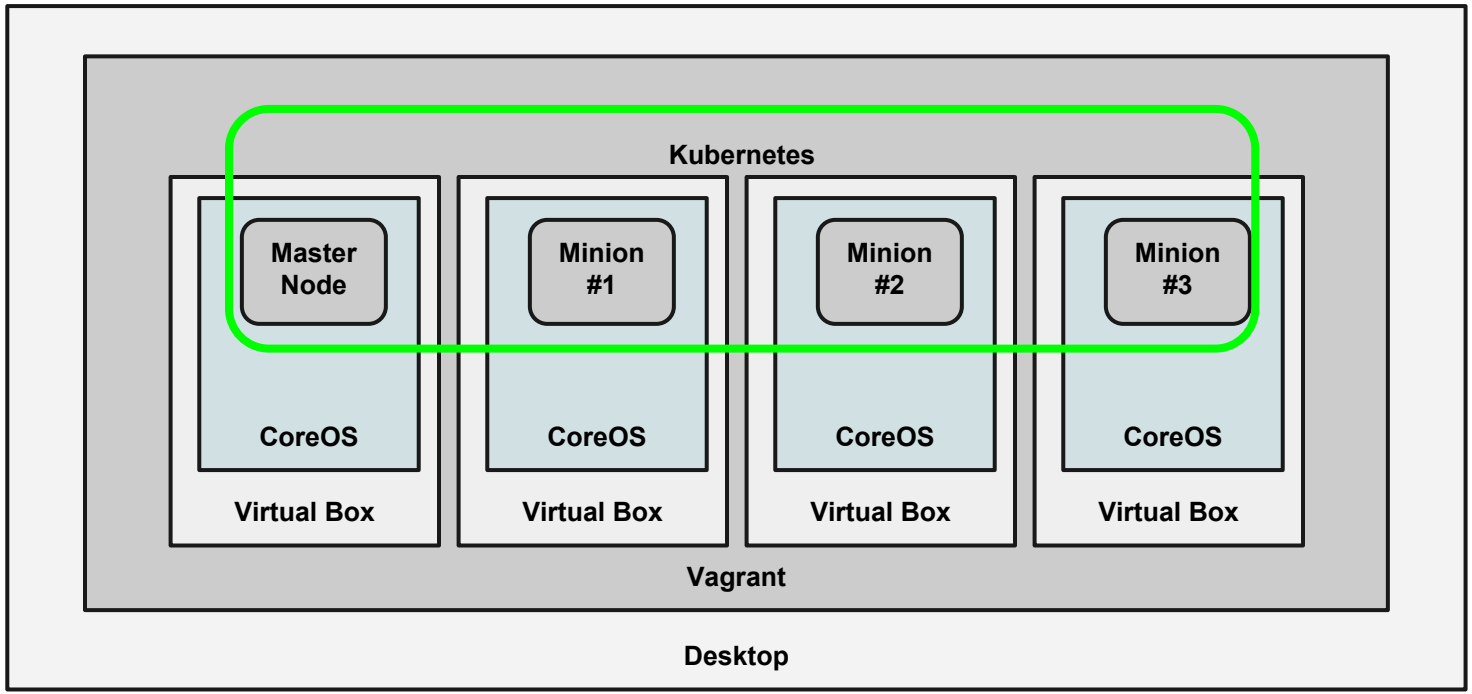
**Service**
*Provides Load Balancing, Service discovery*



# A Kubernetes Cluster



# Developers Cluster



# Container ‘Hypervisor’ OS

Just enough OS to support an application container and keep itself upto date.

- CoreOS
- RedHat Atomic

# CoreOS

An ‘evergreen’ Linux distribution with very current kernel, auto updates with latest patches, and just enough OS to securely run the Application Container.


- Stripped down fork of Chrome OS (itself based on Gentoo) w/ read only system files
- Small: 114MB allowing for quick PXE boot
- Includes Systemd, Docker, Etcd, Cloudinit, Fleet

# Things to Ponder

- Increased usages of Containers
- Does VMWare buy Docker or CoreOS?
- As the sun sets on Moores Law, will we see the rise of the Redundant Array of ARM SoCs as found in our phones?

# Take away

These cloud tools and libraries can make your system better, even if you're not in the cloud.

Your system can be evolved to leverage them, it doesn't require a 'forklift upgrade'.

# References

**Patterns:**

- Microsoft: Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications
- Open Data Center Alliance: Architecting Cloud Aware Applications
- Cloud Computing Patterns: Fundamentals to Design Build and Manage Cloud Applications

**General:**

- Google: The Datacenter as a Computer

**Tools:**

- Kubernetes
- CoreOS
- Hazelcast
- Apache Camel