Technology Comparison

“One of these things is not like the other”

Brad Marshall
brad.marshall@gmail.com

Who am I?

- 20 years+ of being a professional Linux geek
- Worked at small software development company, university, Linux vendor and cloud provider over this time
- Seen lots of technologies come and go

Brief History of Time^WComputing

- Started out as a single computer you ran things on
- Then you clustered the computers to work together
- Then hardware got so fast sometimes you wanted to run more on
- Found things got complicated smooshing different things together, wanted separation
- VMs were born - great for a while but then other problems started popping up
- How to take something from development into production? Containers were born

Containers vs Virtual Machines

- Virtual machines are a emulated machine running on top of a hypervisor
  - Has its own kernel, TCP stack, OS install etc
- Basically a fake version of a real server running
- Examples are KVM, VMWare ESX, HyperV, VirtualBox, Parallels etc

- Containers provide service isolation on the operating system
  - Shares the kernel with the underlying server - so can’t run Windows on a Linux server
  - Easy way of sharing an application configuration and code between people
- Like a mobile application, everything packaged together ready to run
- Examples are LXC/LXD, Docker, Rkt, OpenVZ, FreeBSD Jails, Solaris Zones

How Containers Help

- Very lightweight - not booting a whole OS, so fast to start
- Started out by a developer writing an application on their system, then passing over to a team to deploy
- Issues with not knowing exactly what the developer created it on, what versions of things they used etc
- Containerisation means developers can pass exactly the image they used, and exactly the code they used to deployment team
- Have a common language to talk, not just throw over the wall and walk away

Docker vs LXC

- LXC is an OS level container, you run multiple applications inside it
  - Like a lightweight VM if you can use the same kernel
- Docker is an application level container, with all the dependencies inside it
  - Many apps on Docker Hub to use as a starting point
  - Basically provides templates for existing applications
  - Useful for sharing the exact same experience between dev and prod

Kubernetes

- Platform for managing and operating containerised workloads and services
- Provides options for:
  - Deployment
  - Scaling
  - Load balancing
  - Monitoring
  - Logging
- Developers can build a container on their own computer and then push up to k8s to scale it

Openshift vs Kubernetes

- OpenShift is a commercial version of Kubernetes
- Only runs on RHEL
- OKD is open sourced version, can run anywhere
- Get support from RH at a cost
- Released on a slightly different schedule
- Some features are slightly different, added before Kubernetes had them
    - Routers on OpenShift vs Ingress on Kubernetes

Kubernetes vs Docker-swarm

- Docker Swarm
  - Turns a pool of Docker hosts into a single cluster
  - Exposes only things done by docker
  - Less complex than Kubernetes, so easier to get up to speed
  - Compatible with docker commands so easier transition from current Docker envs
  - Some of the more useful features are in Enterprise edition - so $$$$
- Kubernetes
  - All in one framework that handles HA / scaling / failing nodes etc
  - More complex, so harder to learn
  - More included by default - logging, dashboards, etc
  - Currently seems to be the leader, even Docker is pushing it

Cattle vs Pets

- Might have heard about cattle vs pets
- Are we farmers now?
- Different approach over time to maintaining systems
- TL;DR
  - If a pet goes down its all hands on deck to replace it
  - If a cattle node goes down, simply spawn up another one - even automatically

Pets

Pets vs Cattle - Pets

- Early days you'd install a computer
- Argue over the carefully picked hostname for it
- Hand crafted an installation
- "Lovingly" watched as it grew and evolved over time
- Eventually it would be a tangled mess of interconnecting dependencies
- Treated it like a pet and looked after it - if it disappeared could you even reproduce it?

Cattle

Pets vs Cattle - Cattle

- Containers and better deployment methods means we don’t have to keep things around
- If a server isn’t working right, take it out and replace it
- Easy to do with the right deployment and scaling technology
- Can still keep the misbehaving server for diagnose, but getting it out of production and replace it first
- Allows easy scalability - just add more cattle
- No server is irreplacable

Infrastructure as Code (IaC)

- New paradigm for managing servers
- Configuration is defined in files rather than a running system
- Can be done on bare metal hardware as well as cloud
- Increases the repeatability of deployments and scaling systems
- Usually involves configuration management / orchestration and CI / CD
- Server is not source of truth for config
- Not just config backups
- Push from code rather than pull from server

Deployment vs Provisioning vs Orchestration vs Configuration Management

Deployment is a rollout of an application onto an existing server

Provisioning is getting a server ready to use

Configuration Management is configuring servers in an automated and repeatable fashion

Orchestration is coordinating multiple systems together

Configuration Management

All have Linux servers and Linux/Windows agents

Tool	Arch	Language	Setup	Deployment
Puppet	Multi-master	Puppet DSL and Ruby ERB templates	Master and agent	Pull
Chef	Master-slave	Ruby DSL	Master and agent	Pull
Ansible	Masterless	YAML	Master anywhere, ssh to clients	Push
SaltStack	Multiple masters configured	YAML	Master and minions	Push

Mutable vs Immutable Infrastructure

- Mutable Infrastructure
  - Uses config management or similar to roll out changes to services
  - Can lead to config drift over time - ie, upgrade a package, config can change
  - Perhaps config mgmt doesn’t cover everything on the node

- Immutable Infrastructure
  - No changes made to running systems
  - Use containers / VMs that are replaced by new images

CI / CD

- What is CI/CD?
- Short for Continuous Integration / Continuous Deployment
- Don’t need to do both, but term is together because the techniques are so close
- Continuous Integration
  - All changes to codebase are tested before being accepted in
- Continuous Deployment
  - After changes have been tested they can be rolled out to either testing or production
- Uses a tool like Jenkins, CircleCI, Atlassian Bamboo or TeamCity

Storage - First we played with blocks

- Filesystems made from single disks attached to computers
- Grew to RAID to “cluster” disks together
- Then grew to SANs/NAS where the disks weren’t directly attached to computers
- Then grew to distributed storage (Gluster, DRDB, ZFS, Ceph etc)

Block Storage

- Block storage is like traditional filesystems
- Storage presented to the system using it as a raw device
- Files are split up into raw blocks of data
- Formatted and mounted on the file system
- Like renting space from a storage unit
  - Given a space that is yours
  - You control how things are stored inside the space
  - Usually charged for the entire allocated space
- Strong consistency - read request must give the latest version of data

Storage - Object

- Next step was object storage
- Abstracts away the filesystem details
- Metadata and access control stored on the object
- Unstructured storage of files
- Don’t need to bother with disk failures and making file systems etc
    - Of course someone does, just not you

Object Storage

- Examples are S3, Swift, etc
- Accessed via an API, best for unstructured data
- Like storing your items in a cloak room
  - Your ticket is the API to retrieve it
  - Can be charged on a per item basis
- Unable to edit incrementally, must totally replace the file
- Solves the problems with data growth rate
  - Tried to expand a block base storage beyond hundreds of petabytes?
  - What about doing a filesystem check on it?
- Actual storage in the back end doesn’t matter, can be changed
- Eventually consistent, so can scale well

So What Is The Cloud?

- So what is the cloud?
  - Someone else's computers, obviously
  - Or maybe its your own computers
- API driven virtual machines / containers / functions
- We’re basically providing more and more advanced APIs to cover a lot of uncomfortable and hard things
- Someone still has to do it
- Basically moving the responsibility away from you for the boring stuff
- Lets you focus on what matters to you

Questions?

Any questions?

Contact me at brad.marshall@gmail.com

or @bradleymarshall on Twitter