Using container orchestration to improve service management at the RAL Tier I

Andrew Lahiff, Ian Collier
Rutherford Appleton Laboratory

10th October, CHEP 2016, San Francisco
Overview

- Motivation & goals
- Apache Mesos
- Marathon
- Image creation & storage
- Metrics & logging
- Service discovery
- Example
- Summary & future plans
Motivation

• UK Tier-1 facility at RAL
  – provides resources to all 4 LHC experiments, many non-LHC experiments
  – also provides resources to local facilities, e.g. Diamond Light Source, ISIS Neutron & Muon Source

• Changing landscape
  – LHC resource requirements increasing in size
  – broadening user base of the Tier-1
  – non-LHC experiments & local facilities becoming more important
    • allocation expected to exceed UK contribution to the LHC in coming years
    – staff effort unlikely to increase & in fact will probably decrease

• Important to investigate ways of managing existing services & potentially provide more services with less effort
How do we run services?

- Aquilon (Quattor) for configuration management
- Services run on enterprise virtualization platforms or in some cases bare metal
- Significant amount of manual work involved for both deployment & upgrades, also:
  - manual intervention is required to resolve problems
    - many Nagios tests but at most they result in emails or pager alarms
  - very static environment
    - many manual steps in order to scale horizontally
  - problem with VM or hypervisor results in loss of service
    - recent improvement due to use of shared storage under hypervisors
  - dead machines visible to users due to use of static DNS aliases
    - started to use load balancers to get around this (only FTS3 so far)
- Very little has changed in the past 6 years or more
Goals

• Ideally would like to have an environment where
  – The infrastructure itself is
    • flexible
    • fault-tolerant
    • scalable
  – Services are
    • quickly & easily deployable, easily updated
    • self-healing
    • elastic, auto-scaling
    • multi-tenant

• Looking at what’s happening in the wider world, one way to do this is to
  – migrate to running applications in containers
  – manage the containers using schedulers, not people

• This is a very significant change in the way we manage services
Apache Mesos

- Originated in UC Berkeley in 2011 & became a Top Level Project at Apache in 2013
- Mesos is a cluster manager which
  - enables a large group of machines to appear as a single pool of resources
  - allows you to have multiple schedulers sharing the same resources

Mesos agents provide resources to the Mesos master

Mesos master offers resources to frameworks

Frameworks decide what offers to accept & what to do with them
Marathon

- A Mesos framework for long-running services
  - applications & groups of applications with dependencies
  - health checks

Benefits include

Self–healing (no more 2am pager alarms)

Rolling upgrades

Each colour represents a task (instance of an application)
Deployment at RAL

- **Mesos agents**
  - 164 x 32 cores, 84 x 16 cores (all bare metal)
  - Mesos agent, Docker engine, Consul agent

- **Mesos masters**
  - 5 VMs (4 cores, 8 GB memory each)
    - can lose up to 2 hosts without any problems
  - Mesos master, ZooKeeper, Marathon, Consul server

- **Some experiences**
  - CPU & memory usage on the Mesos masters is low
    - Consul biggest user of CPU
  - Disk i/o important for ZooKeeper
    - 2 of our 3 virtualization platforms don’t quite have fast enough disk storage
Image creation & storage

- Distributed private Docker registry
  - Ceph storage backend via Swift gateway
  - central registry instance providing write access
  - read-only registry on every Mesos agent
    - lightweight
    - avoids having a single point of failure / bottleneck

- Creating images
  - currently creating images “by hand” from Dockerfiles
  - work in progress on leveraging Packer to build VM & container images from our configuration management system (Aquilon)
Metrics & logging

- Traditional monitoring is host-centric & assumes static infrastructure
  - not suitable for a dynamic containerized environment
  - no static hostnames or IP addresses & number of instances can change

- Use metadata to dynamically aggregate metrics & logs
  - e.g. application name rather than hostname

- Metrics
  - Telegraf: collecting infrastructure metrics (input plugins for Mesos, ZooKeeper, ...)
  - cAdvisor: collecting container metrics (resource usage, application metrics)
  - InfluxDB, Grafana

- Logging
  - centralized is logging important: applications can move around
  - Filebeat (stdout/err from containers, Mesos logs, ...)
  - Logstash, Elasticsearch, Kibana
Service discovery

- Static hostnames in configuration files no longer make sense
- Using Consul, a distributed tool for service discovery
  - Containers exposing ports are automatically registered in Consul
- Within the Mesos cluster
  - Services can be accessed in several ways, including DNS, e.g.
    - logstash.service.consul
- External access to services in the Mesos cluster
  - Pairs of load balancers
    - HAProxy (load balancing)
    - Keepalived (HA floating IP addresses)
  - Configuration dynamically updated by Consul
    - It’s possible to setup HAProxy for zero-downtime reloads
Example: worker nodes

- Investigating the ability to run HTCondor worker nodes on Mesos
  - Existing production HTCondor central managers & ARC CEs
  - Running on Mesos
    - worker nodes
    - squids
- Container management
  - Marathon for squids
    - autoscaling based on request rate
  - A custom framework for worker nodes
    - creates worker node containers as needed
    - Why not Marathon? Need to be able to scale down & perform rolling upgrades without killing jobs
- Marathon & the custom framework registered in Mesos as different roles
  - can ensure that worker nodes can’t take over entire cluster
Worker nodes

- CVMFS & condor_startd inside the container
  - host doesn’t need anything at all related to worker nodes installed
  - allows us to run as many worker nodes as required without having to dedicate a set of resources configured as “WLCG worker nodes”

- Each job
  - runs in it’s own CPU & memory cgroups nested in the worker node container
  - has it’s own PID & mount namespace

- Container exits if there has been no work for a specified duration
Worker nodes

- Example of recent tests with real jobs from all 4 LHC experiments

HTCondor startd ClassAds contain details such as Mesos task ID & image name which are automatically added to job ClassAds for traceability.
Auto-scaling

- Example of the number of squid instances changing based on load (request rate)

Spike in request rate triggers creation of additional squid instances

Drop in request rate therefore number of squid instances is reduced

Squid caches warming up
Summary & future plans

• The use of containers & container orchestration seems to have many benefits compared to our existing infrastructure
  – potentially higher availability with less effort & higher resource utilization
• Status of Mesos at RAL
  – used for testing & development; some non-critical internal services; a small fraction of batch jobs
  – not yet an “official” production service
    • it’s a significant change in philosophy
    • the Tier-1 facility at RAL has to meet high SLAs, so moving away from a well-established infrastructure takes time
  – technologies initially looked at as part of the work on Mesos now being used more widely within the RAL Tier-1
    • Keepalived & HAProxy used in production in front of FTS3 for > 6 months
    • Telegraf, InfluxDB & Grafana now monitoring over 900 hosts
Summary & future plans

• Future plans include
  – increased integration with our configuration management system
    • move to using images created by Packer from configuration in Aquilon
  – using Ceph to allow containers to have persistent storage
  – investigate running OpenStack hypervisors in containers
    • will allow us to have cloud & batch sharing the same resources
  – contributions to INDIGO-DataCloud
Questions?