Mesos in a WLCG Tier-1 Grid Site

Andrew Lahiff

International Symposium on Grids and Clouds 2016, Taiwan

Overview

- Introduction
- Mesos & Marathon
- How we're dealing with dynamic environments
  - Service discovery
  - Monitoring
- Migrating grid services
  - Examples of benefits
- Moving into production

Introduction

- RAL is a Tier-1 for all 4 LHC experiments
  - This year will provide over 146K HS06 CPU, 13 PB disk, 35 PB tape
  - Also support many non-LHC experiments, becoming increasingly important
- Technology used
  - Storage: CASTOR, Ceph (not yet in production)
  - Compute: HTCondor, OpenNebula
  - Services: HyperV

Long-running services

- At RAL and other WLCG sites, services are
  - Running on manually instantiated VMs (or bare metal)
  - Configured using configuration management systems
- There are limitations with this model, including
  - manual intervention is required to restore services in the event of problems (service crashes, hypervisor crashes, ...)
  - resource utilization not optimal
    - many VMs likely have unused resources, no way to make use of them
  - very static environment
    - no way to scale capacity to respond to changes in demand quickly enough to be useful

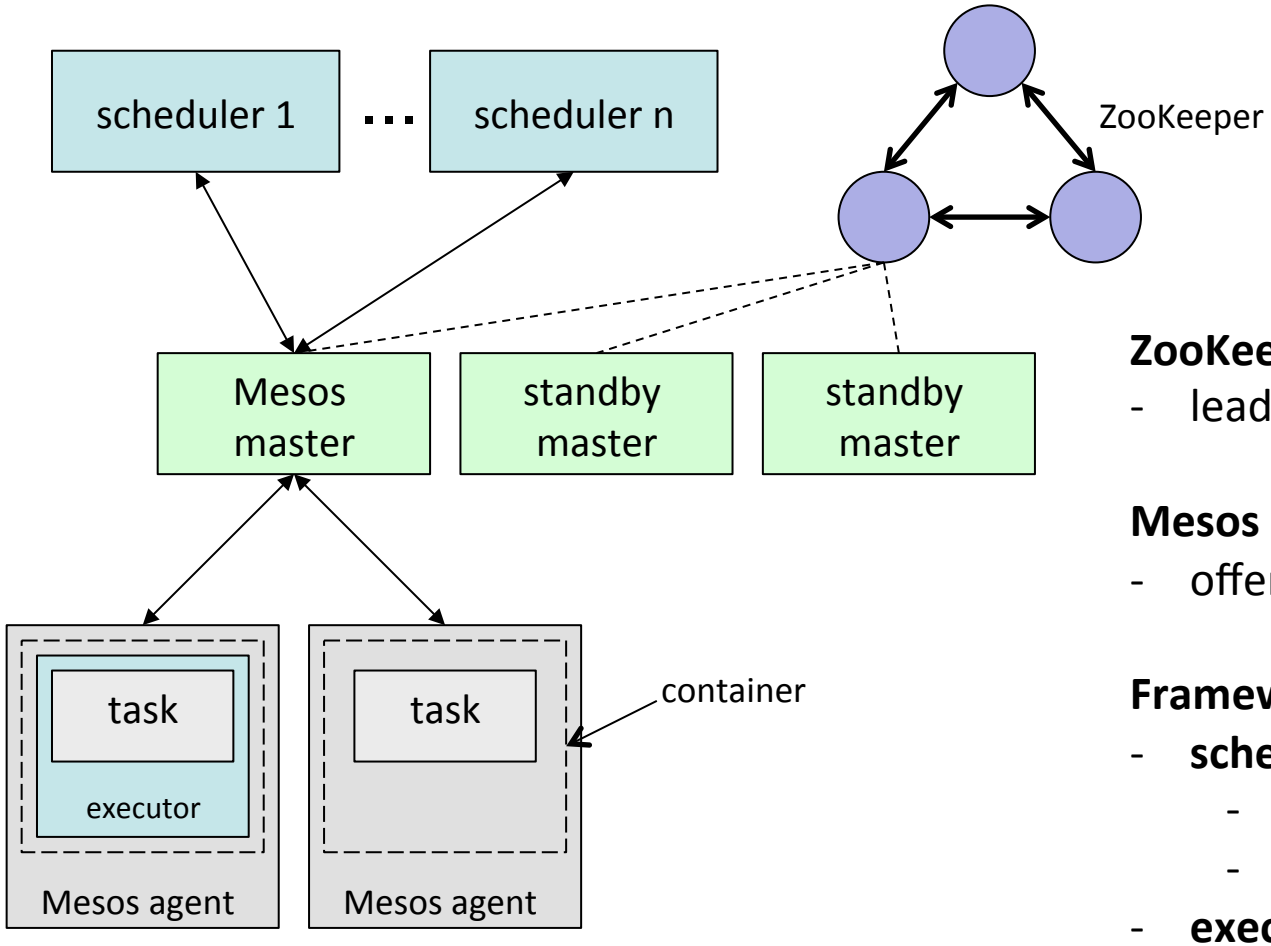
Avoiding these problems

- 2 fundamental changes are required
  - **Manage applications using a scheduler**
    - if an application dies, it will be restarted
    - if a machine dies, applications running on it will be re-scheduled elsewhere
    - automated staged-rollouts
    - auto scaling
    - application-specific scheduling
  - **Run applications in containers**
    - removes the dependency between applications & hosts
    - enables applications to be quickly started anywhere
    - allows for isolation between different applications

Apache Mesos

- What it does
  - enables a large group of machines to appear as a single pool of resources
  - abstracts away the whole concept of individual machines
    - lets users manage applications, not machines
  - allows you to have multiple schedulers sharing the same resources
- Features
  - fault-tolerant
  - used in production at scales of > 10,000 machines
- Consists of 4 major components
  - ZooKeeper, Mesos masters, Mesos agents, Frameworks

Mesos architecture



ZooKeeper
- leader election & coordination

Mesos master
- offers resources to frameworks

Frameworks consist of
- **scheduler**
  - receive resource offers
  - launch tasks
- **executor** (optional)
  - launched by Mesos agents
  - execute tasks

Frameworks

- Marathon
  - distributed “init” for long-running services
  - Placement contraints (nodes, racks, ...)
  - Health checks
  - Rolling restarts & upgrades
  - Handles dependencies between applications
- Other frameworks include
  - Chronos: distributed “cron”
  - Elasticsearch, HDFS, Cassandra, Kafka, Kubernetes, ...
- Multiple frameworks can be run on a single Mesos cluster
  - Mesos allows you to have multiple distributed systems sharing the same resources

Deployment at RAL

- Infrastructure managed by Quattor
  - 5 Mesos masters
    - ZooKeeper, Mesos master, Marathon, Consul server
    - httpd proxies provide SSL & X509 authentication for UIs
  - 16 Mesos agents
    - 16 CPUs, 24 GB RAM each
    - Mesos agent, Docker engine, Consul agent
  - Private Docker registry
- Impressions
  - easy to setup basic infrastructure
  - very stable
  - service discovery, monitoring, logging & security is where effort is required to ensure a production-quality service

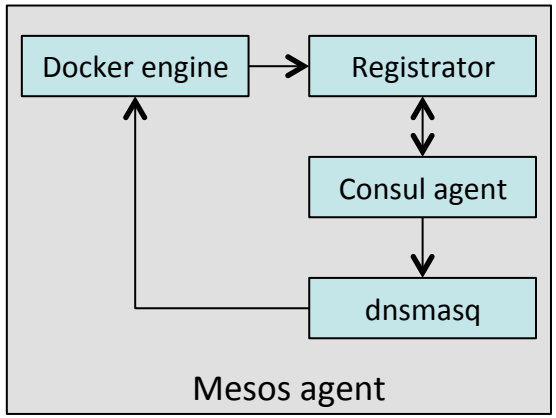
Service discovery

- A significant change: static hostnames in config files no longer make sense
  - hostnames can change at any time
  - number of instances of a particular service could change
- How can services find each other?
  - One solution: Consul, a distributed tool for service discovery
- Service registration
  - Registrator daemon runs on each Mesos agent
    - detects Docker containers starting & stopping
  - registers services with Consul
    - name, hostname, port(s), tags, health checks

Service discovery

- Services can then be accessed in several ways, including:
  - DNS, e.g.

    logstash-fts3.service.consul
  - Consul-template
    - daemon which populates configuration files dynamically
- Docker engine configured to use local dnsmasq for DNS
  - dnsmasq using Consul for .consul domain

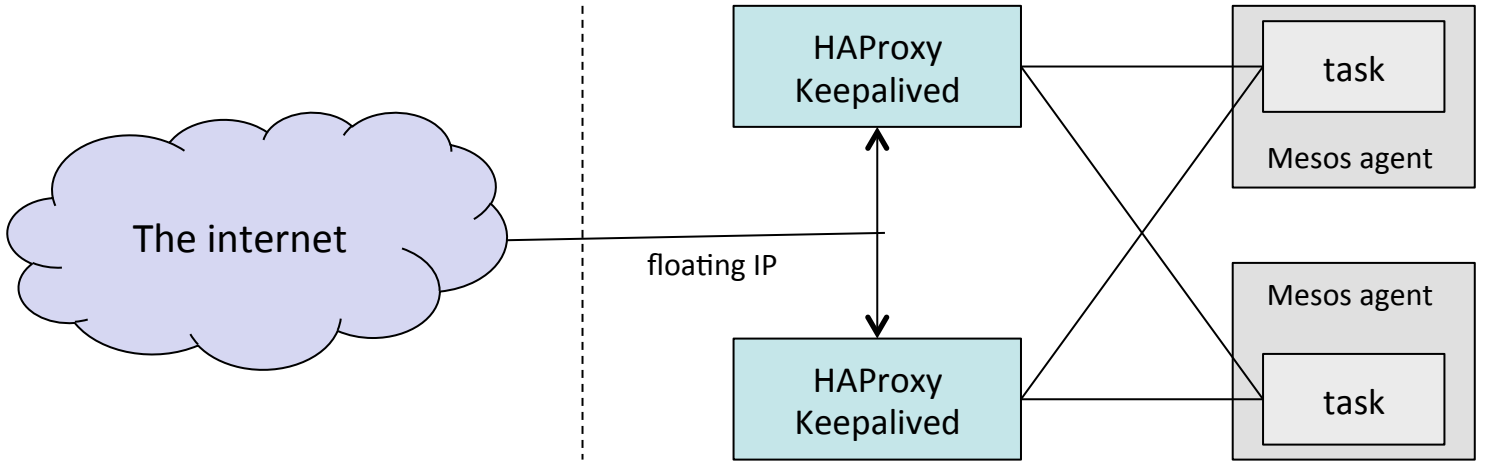


Service discovery

- Can also register health checks with Consul
  - HTTP, TCP, TTL, script
  - Run by each local Consul agent (decentralized)
- Unlike Nagios checks, these can change dynamically
  - checks specified as labels on containers
  - Registrator registers/de-registers checks as containers come & go
- Health checks are a critical part of service discovery
  - it should be considered normal that some instances of applications will fail
  - essential to prevent unhealthy services from being used
    - e.g. DNS lookup will only return healthy services

External access to services

- Using HAProxy as a load balancer in front of services
  - configuration dynamically updated
  - ensures clients can only access healthy instances of services
- Keepalived provides
  - a floating IP for each service
  - highly available load balancing



Monitoring

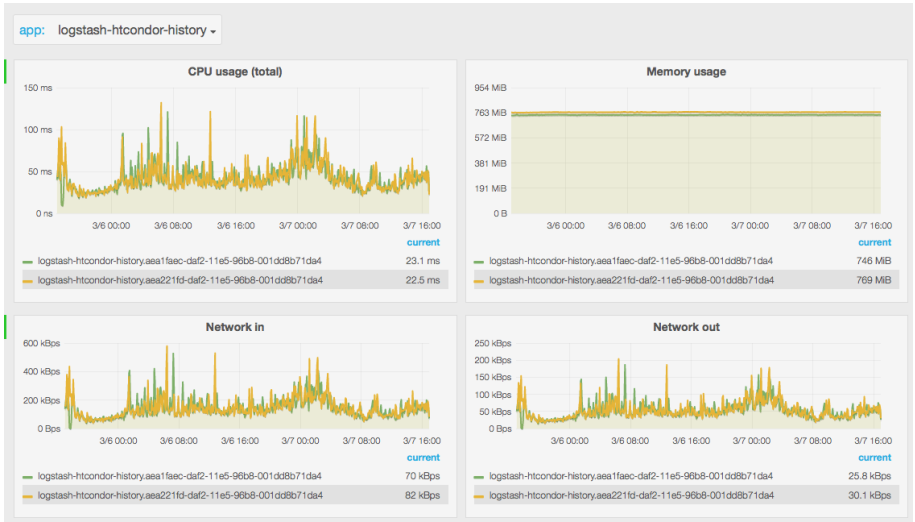
- The container world is different to traditional infrastructure
  - hosts not relevant
  - containers can move from one host to another
- Traditional monitoring for both metrics & alerts
  - static
    - can't handle an application moving from one host to another
  - host-centric
    - e.g. a myproxy-server process must be running on host X
- Need a different approach
  - Metrics
    - need to be able to track & aggreggrate multiple containers
  - Query-based alerts
    - e.g. number of running squids > N

Metrics

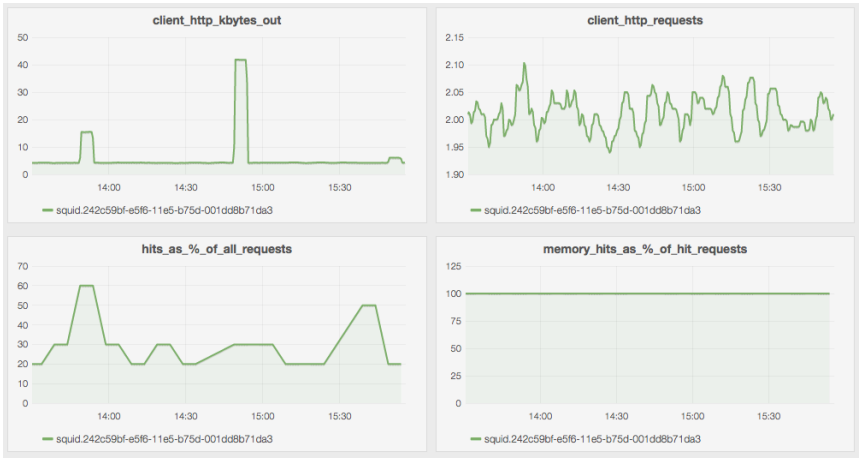
- cAdvisor
  - daemon running on each Mesos agent
  - collects container resource usage metrics, including
    - CPU, load, memory, network, disk
  - collects application metrics
    - metrics exposed on a port on each container
    - no site specific metrics configuration required inside containers
  - metrics tagged by useful information from Mesos
    - e.g. application name, task id
  - data stored in InfluxDB, Grafana used for visualization

Metrics

- Can easily view all instances of a particular application
  - irrespective of how many instances there are or what hosts they're running on
  - note that it wouldn't be possible to do this with Ganglia



Resource usage metrics



Application metrics (example: squid)

Alerting

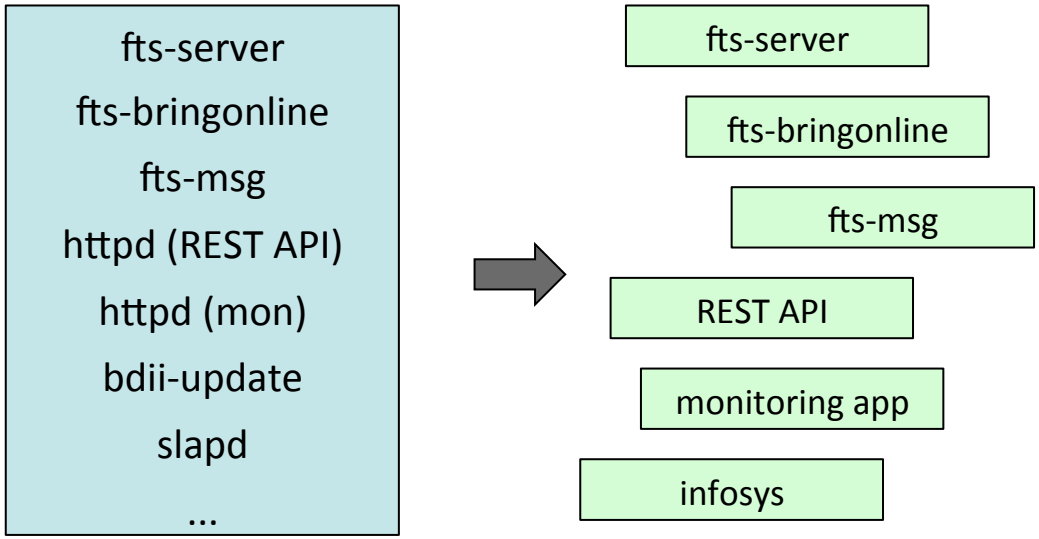
- Things will always fail
  - e.g. no need for a pager alarm if 1 of 8 FTS3 servers dies
  - the important question: is the service degraded (or about to be) beyond a critical point?
- Mesos infrastracture
  - Masters have standard Nagios tests
  - Agents like worker nodes in a batch system
    - only care if number of healthy agents drops below a threshold
- Services running on Mesos
  - Nagios tests on the load balancers
    - number of healthy instances of backend servers
    - callout if below a minimum threshold

Migrating grid services

- Grid middleware – typically consist of multiple sub services running in single VMs
  - problems with one can affect others in the same VM
  - frequently have multiple instances of all services, even if not needed
- Not easy to split services into multiple VMs
  - very inefficient use of resources
  - too complicated for most sites to configure this
- However
  - can split into multiple containers
    - each container has a single purpose
    - container orchestration combined with dynamic service discovery makes this both possible & straightforward

Migrating grid services

- Example: FTS3 server
  - typical example of grid middleware
  - large number of daemons & crons running in a single VM
- Split into multiple containers, e.g.



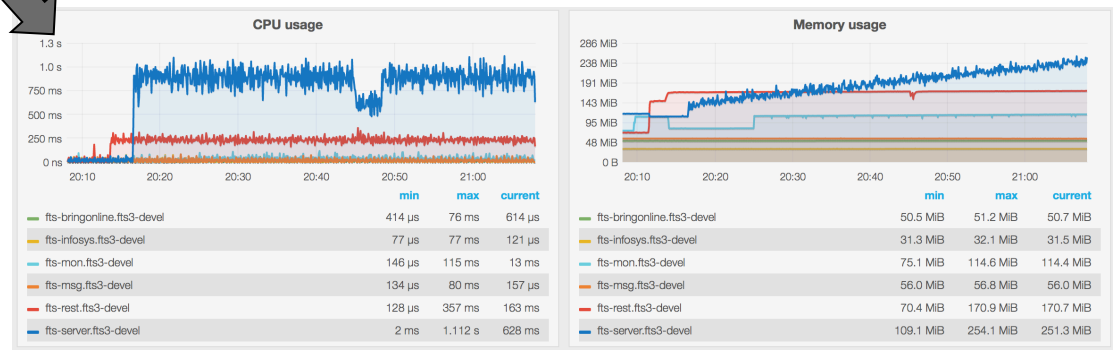
large VM *multiple (small) containers*

Migrating grid services

- At first glance may seem more complicated, but
  - greater visibility into what each sub service is doing
  - can scale only the components that need to be scaled, instead of everything

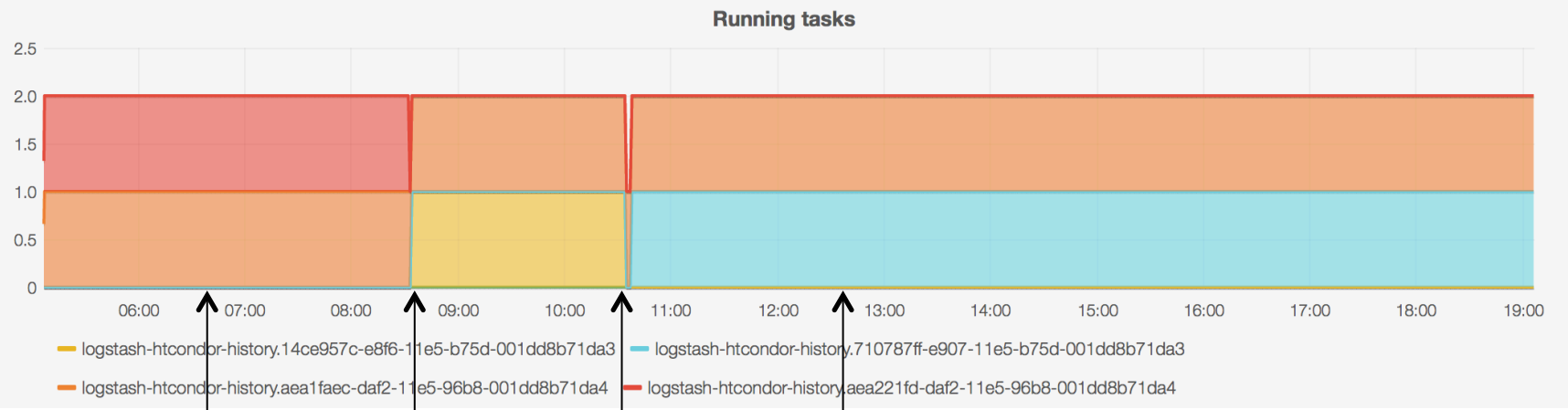
```
top - 21:13:39 up 22 days,  8:41,  1 user,  load average: 0.02, 0.01, 0.00
Tasks: 241 total,   1 running, 240 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.1%us,  0.8%sy,  0.0%ni, 99.1%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8052088k total,  3403416k used,  4648672k free,   657620k buffers
Swap:  2097148k total,        0k used,  2097148k free,  1640172k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4315 fts3      20   0 2523m  27m 8672 S  3.0  0.4 616:34.87 fts_server
 4714 fts3      20   0 1343m  11m 3996 S  0.3  0.2 112:41.57 fts_bringonline
36305 root      20   0  658m  19m 4532 S  0.3  0.3  15:55.63 filebeat
38417 fts3      20   0  710m  18m  10m S  0.3  0.2   0:00.39 fts_url_copy
39256 fts3      20   0  709m  16m  10m S  0.3  0.2   0:00.33 fts_url_copy
39356 root      20   0 15160 1420  988 R  0.3  0.0   0:00.01 top
63558 apache    20   0  396m  12m 2044 S  0.3  0.2   0:00.18 httpd.worker
    1 root      20   0 23472 1592 1300 S  0.0  0.0   0:03.11 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthre
```



Fault tolerance

- One benefit of running services on Mesos is improved availability due to fault tolerance
  - example: an application with 2 running instances



Another Mesos master shutdown – no effect

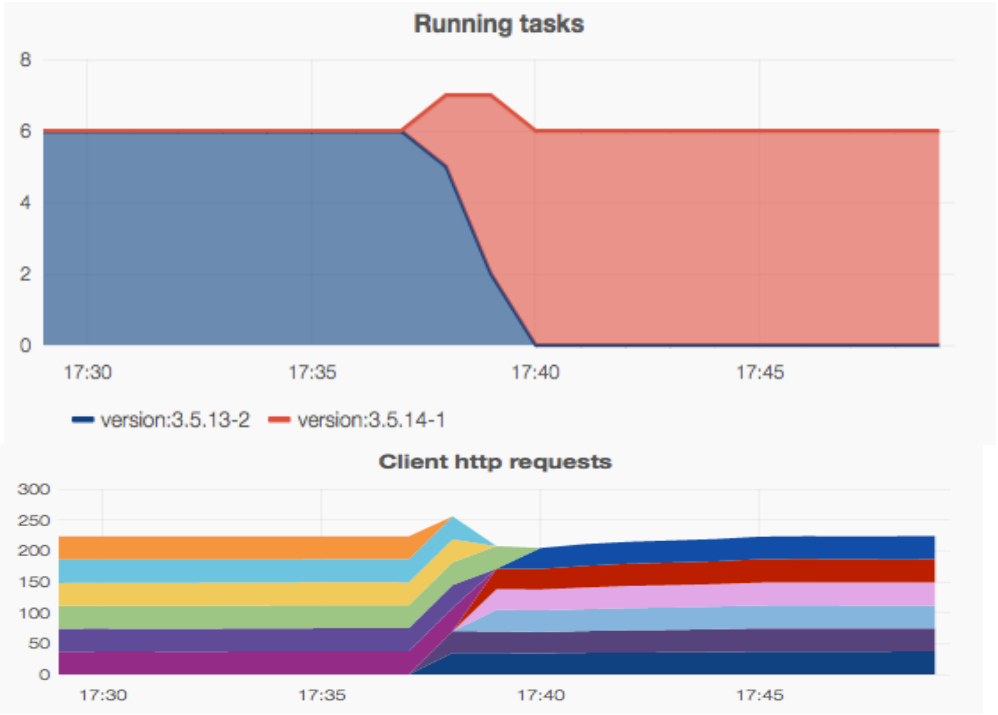
Mesos agent shutdown – task restarted automatically on another host

Task died – restarted automatically (not necessarily on the same host)

1 Mesos master shutdown – no effect

Rolling upgrades

- Example of an automated rolling upgrade
  - not normal to upgrade running containers: you replace them
  - old instances killed only when new ones become healthy



Requests directed to new instances as they become healthy
- *the upgrade is transparent*

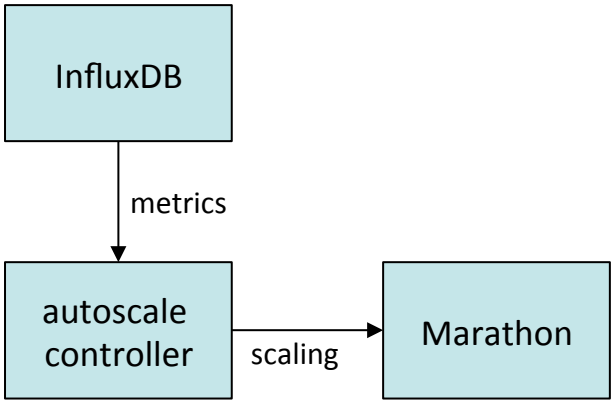
Each colour corresponds to a different container

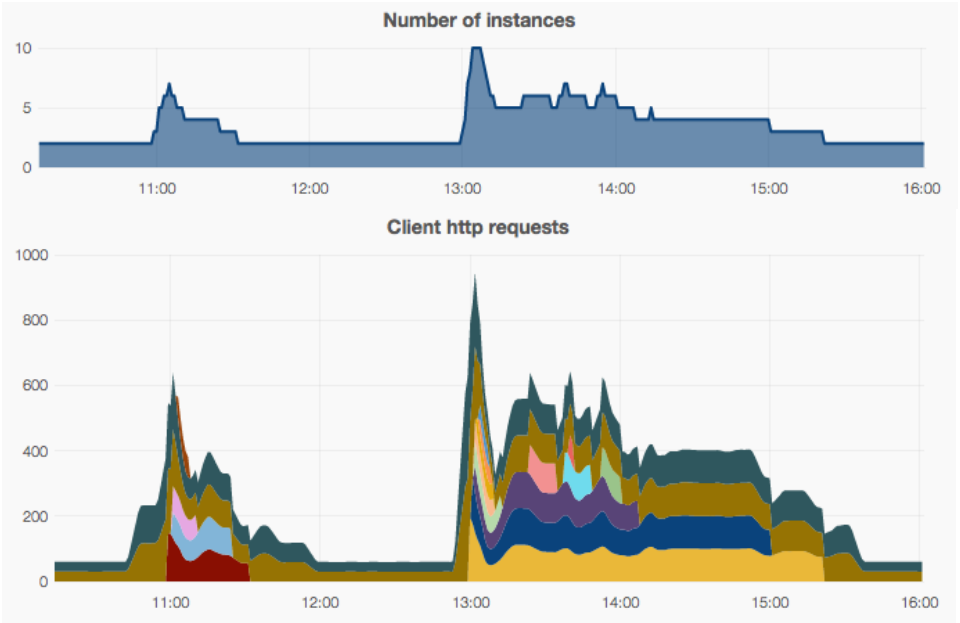
Auto scaling

- Automatically scale capacity according to demand
  - VMs take minutes to start
    - this can be too long
  - Containers take seconds to start
    - can more quickly respond to spikes in demand
- Scaling based on metrics collected by cAdvisor
  - Could use resource usage, e.g. CPU, memory, network
  - And/or application metrics, e.g. request rate

Auto scaling

- Example: horizontal scaling of squid proxies
  - number of containers scales up & down depending on load





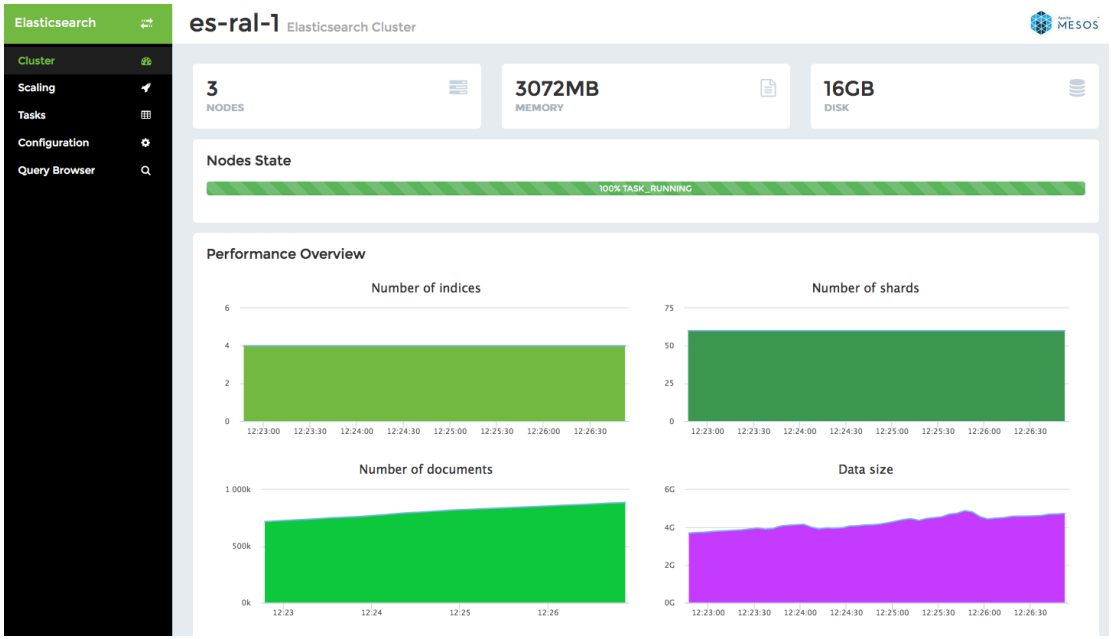
Each colour corresponds to a different container

Target # requests per container very low for this example

Number of running instances restricted to be between 2 & 10

Managed services

- Example: Elasticsearch framework
  - framework designed to manage an Elasticsearch cluster
  - launch a multi-node Elasticsearch cluster in seconds
  - automatically detects & replaces failed Elasticsearch nodes



Moving into production

- **Everything** is dynamic
  - still (relatively) new technology, things are changing rapidly
    - what is difficult to do today maybe simple tomorrow
- Big change in philosophy for service owners
  - what were previously critical VMs become more like jobs running in a batch system on expendable worker nodes
  - people used to managing their services manually
- Security
  - How to securely & automatically provide host certificates to containers?
  - How to tell if a running container has a vulnerability, e.g. nss? It may only exist for a few minutes (or less)

Summary

- Investigated a new way of running services
  - containers managed by a scheduler
  - many benefits compared to traditional static VMs
    - potentially higher availability with less effort & higher resource utilization
  - work is currently underway to make Mesos a production service at RAL
- Technologies looked as part of Mesos work starting to be used more generally within the RAL Tier-1, e.g.
  - load balancers entering production this week for FTS3
    - with Nagios alerts similar to what would be used for services running on Mesos
  - InfluxDB & Grafana for metrics monitoring