Science & Technology Facilities Council

Further Adventures in Container Orchestration at RAL

Andrew Lahiff, Ian Collier

STFC, Rutherford Appleton Laboratory

HEPiX Fall 2016 Workshop, LBNL

Overview

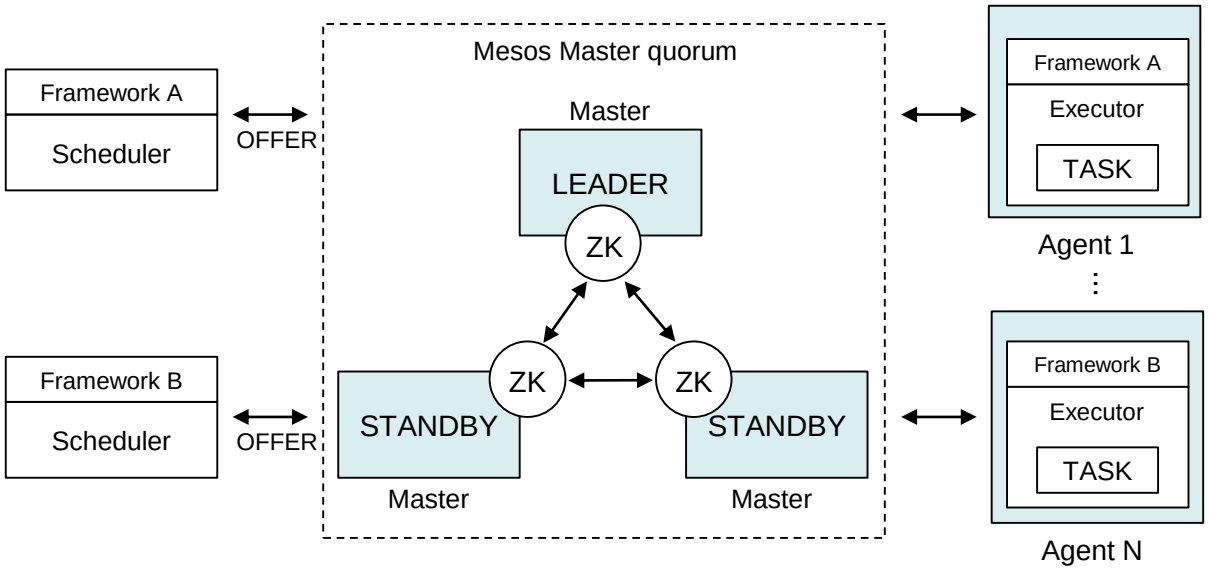
- Introduction
- Mesos
  - Creating images
  - Private docker registry
  - Grid Worker nodes
  - Running production services
- Commercial clouds
- Summary

Introduction

- Investigating ways to manage existing services & potentially provide more services with less effort
- Container orchestration has the potential to provide 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

Introduction

- Using Apache Mesos
  - Marathon framework for managing long-running services
  - Consul for service discovery
  - cAdvisor, InfluxDB, Grafana for metrics
  - Filebeat, Logstash, Elasticsearch & Kibana for logging
- More information in previous HEPiX meetings



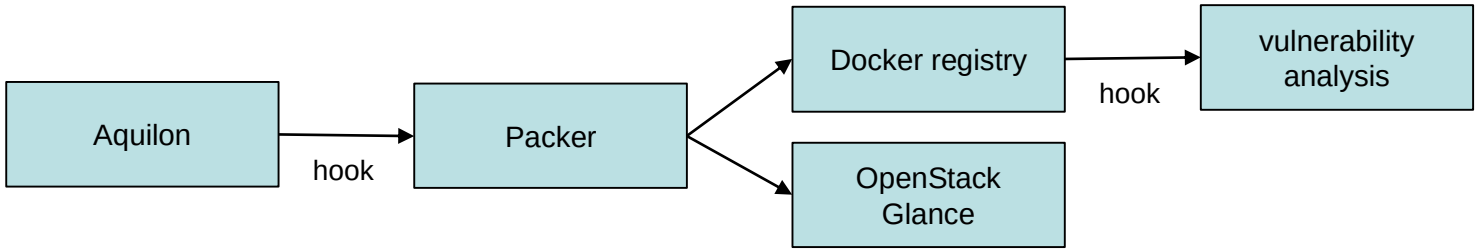
Mesos agents provide resources to the Mesos master

Mesos master offers resources to frameworks

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

Creating images

- Container images are the basic starting point for applications
- Currently creating images “by hand” from Dockerfiles, then manually uploading to a private Docker registry
- Work in progress on leveraging HashiCorp Packer
  - build both VM & container images from Aquilon, our configuration management system
  - automatically
    - upload images to a private Docker registry
    - carry out vulnerability analyses of images (e.g. CoreOS Clair)
    - potentially also deploy to a test environment

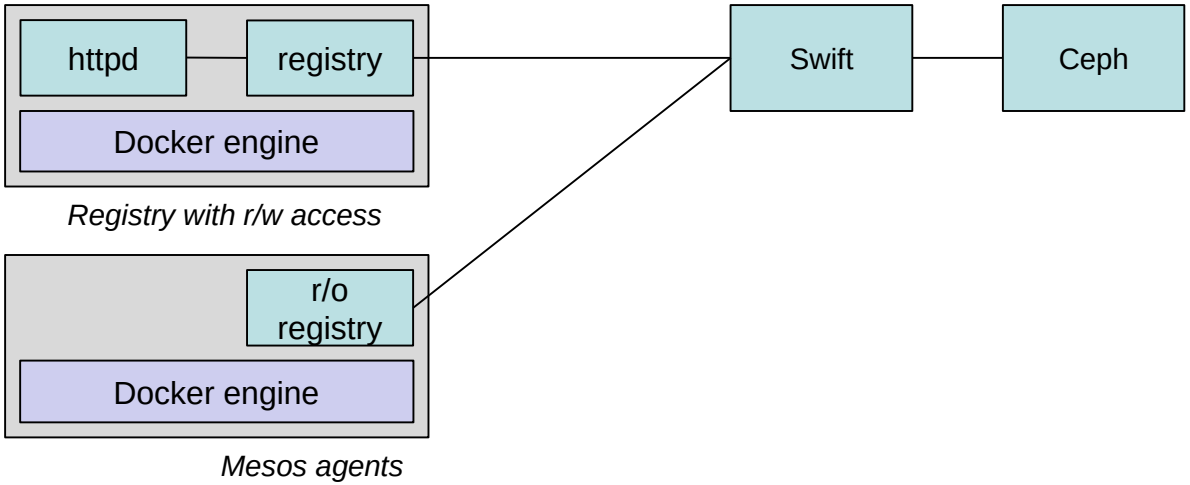


Private Docker registry

- First attempt
  - single VM running Docker registry container
  - storage backend: volume bind-mounted from the VM
- Security
  - httpd providing SSL + simple authn/authz
  - investigating authorization servers for more advanced features (e.g. LDAP, groups, ...)
    - docker_auth
    - SUSE Portus
- Problems with this simple setup
  - it's a single point of failure
  - it's a network bottleneck

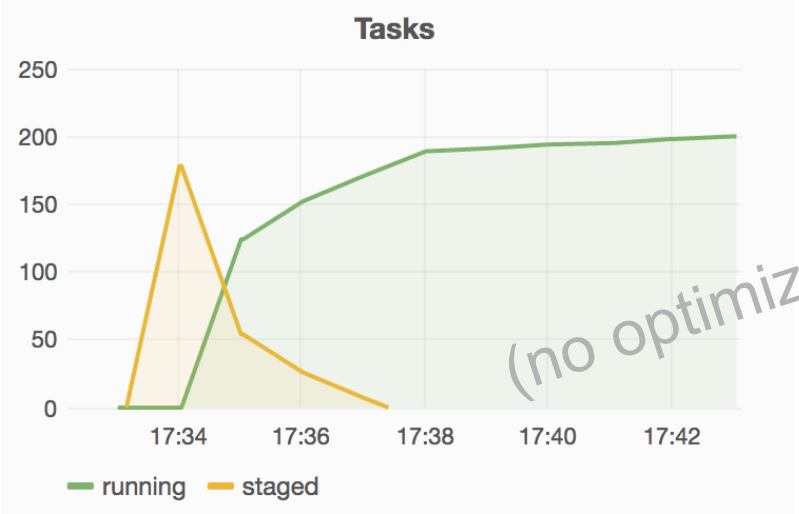
Private Docker registry

- Alternative: use Ceph as the storage backend with Swift gateway
  - Central registry with read/write access
  - Read-only registries on every Mesos agent
    - it's very lightweight
    - when images are pulled the network traffic comes directly from the Swift gateway to the appropriate Mesos agent

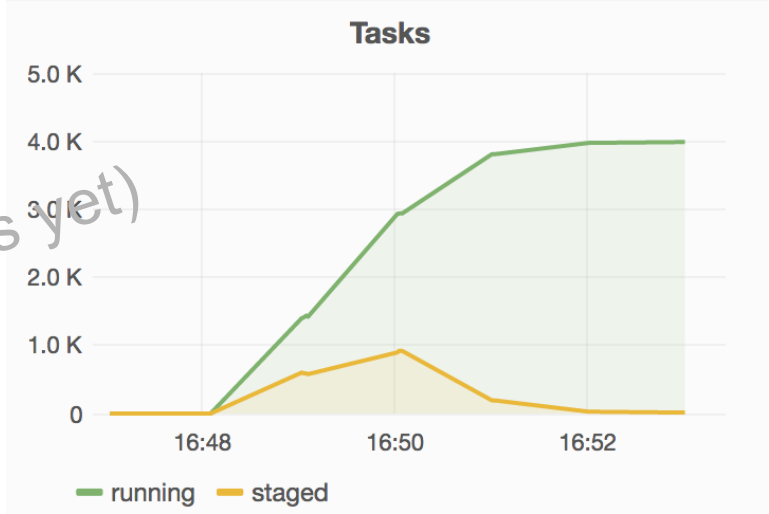


Private Docker registry

- Tried starting 200 instances of a container with 1 GB image size
  - result using a single registry: the registry crashes
- Everything is fine when using a “distributed” registry:



1 GB image x 200 instances
(image pulled for every single instance)



15 MB image x 4000 instances
(image pulled for every single instance)

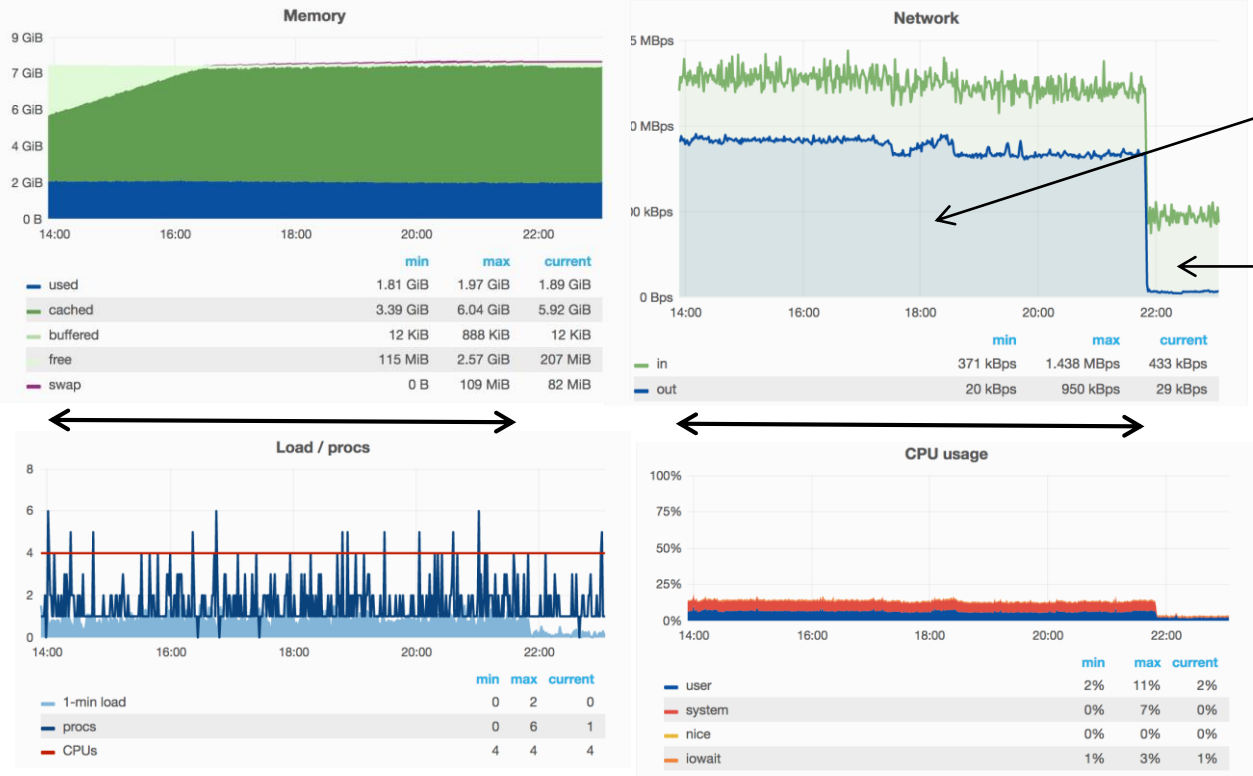
plots use data with 1-minute time resoulution

Mesos at larger scales

- Until recently have only had a small cluster (256 cores)
- How are things at larger scales?
  - Now have 164 x 32 cores, 84 x 16 cores (all bare metal)
  - No problems found as a result of having a larger cluster
- Load on Mesos masters
  - With just some relatively-static long-running services resource usage is low
  - When large numbers of containers are being created regularly there is more load visible *(see next slide)*
- ZooKeeper
  - Known to require fast disk
  - Have noticed that on 2 of our 3 Hyper-V virtualization clusters disk i/o not fast enough (warnings about fsyncs taking too long)

Mesos at larger scales

- Resource usage of leading Mesos master under higher load
  - running containers which live for a random time < 60s, around 2000 simultaneously
  - over 2 million containers created & destroyed over a few hours



Large numbers of containers being continually created

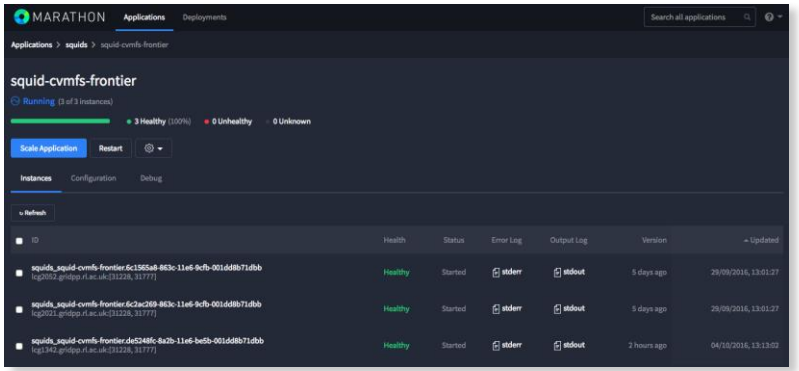
Long-running services only

Generic compute resources

- Currently have separate cloud & batch resources
  - however for ~ 1.5 years our batch system has made opportunistic use of free resources in our private cloud
    - worker nodes running on virtual machines
  - but no way for the cloud to make use of idle batch resources
- Investigating whether we can have a generic set of machines which can be used for
  - worker nodes
  - OpenStack hypervisors
  - potentially other compute activities (e.g. Spark)
  - running services
- Can we move away from the idea of resources partitioned into dedicated silos for different uses?

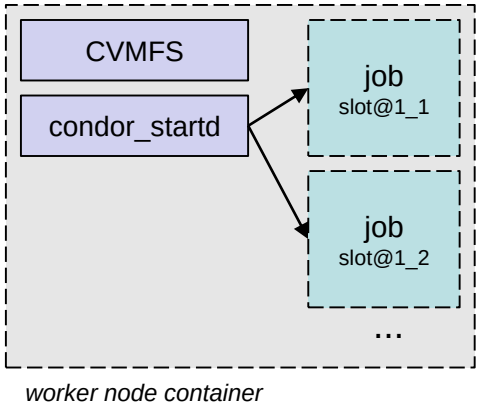
Grid Worker nodes

- Investigating running 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



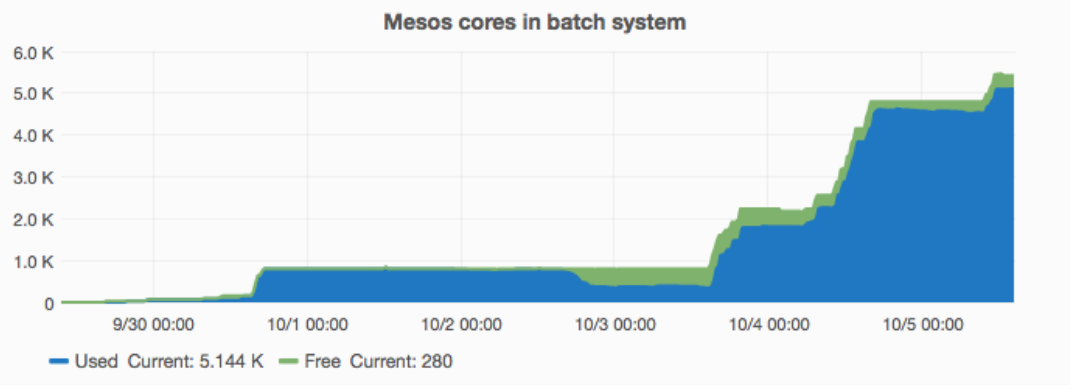
Grid Worker nodes

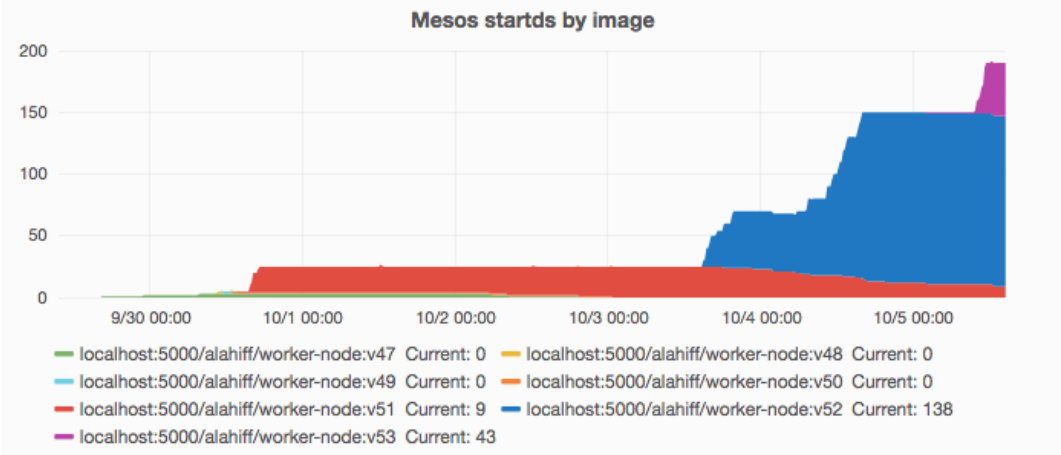
- CVMFS & condor_startd inside the container
  - host doesn't need anything at all related to worker nodes installed
  - allows us to run as many worker nodes as required without having to dedicate a set of resources configured as "WLCG worker nodes"
- Each job
  - runs in it's own CPU & memory cgroups 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



Grid Worker nodes

Example of recent tests running jobs from all 4 LHC VOs





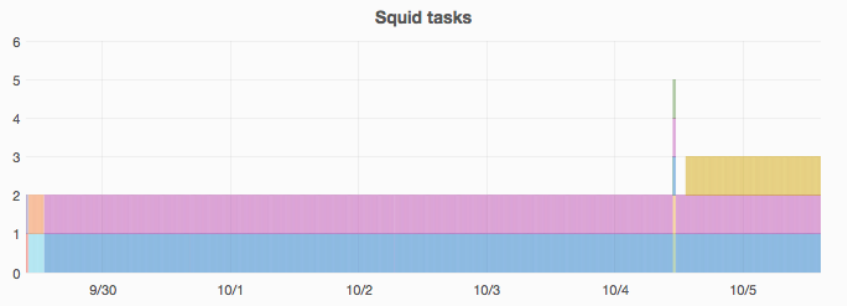
For traceability
- *information from Mesos made available in startd ClassAds (task ID, image name, ...)*
- *also added to job ClassAds*

Therefore for every HTCondor job we can identify e.g.
- *host it ran on*
- *the Mesos task ID*
- *container ID*
- *image used*

and can easily find the HTCondor & glexec logs (even if the container is no longer running)

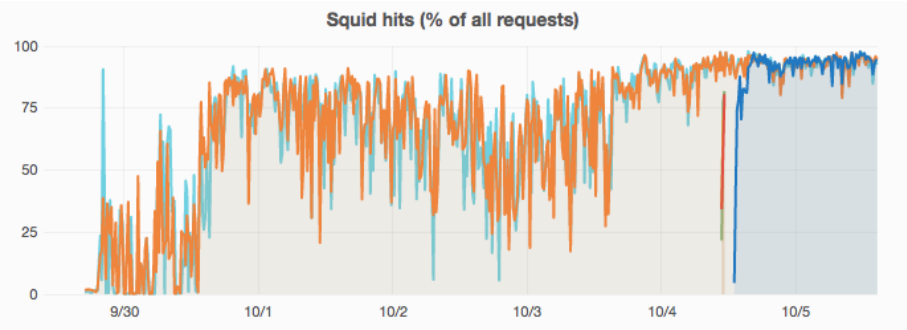
Grid Worker nodes

Squids running on Mesos for CVMFS (all VOs), Frontier (CMS)

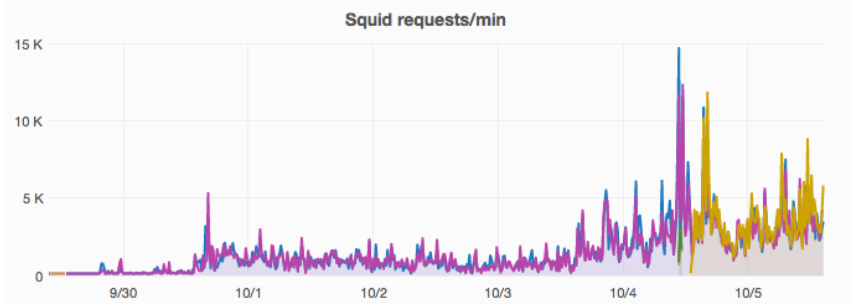


Number of squids; each colour corresponds to a unique task

- *can click a button to create a new squid and/or use auto-scaling*
- *adding a new squid with our traditional infrastructure involves surprising amount of manual work*



Application metrics
-exposed by each container via http
-collected by cAdvisor
-stored in InfluxDB



New squids automatically used by CVMFS & Frontier as they are created without any config files being updated or submitting tickets to request DNS changes

Other benefits

Container orchestration facilitates increased automation & higher service quality – partly because it *requires* automated solutions in areas where we have relied on (got away with) manual effort:

- Monitoring
  - aggregate metrics dynamically using metadata
  - historically we have used hardwired lists of hosts
- Logging
  - More dynamic central logging (e.g. ELK) becomes (almost) essential
- Health checks
  - need functional tests for each application
  - historically many of our grid services have copious Nagios checks on hosts but less emphasis on proper functional tests
- Secrets
  - need to properly store & distribute secrets securely
  - historically we have managed distributing secrets by hand

Running production services

- How can our current production services benefit from this approach to service management?
  - Issues are “cultural”, not technical
  - A significant change in philosophy
  - Hard to approach using our change management process

My service always runs on the same machine & it has a sticker!

My service is being managed by software & is running somewhere in here...

Running production services

- Have to meet high SLAs
  - Any move away from a tried, tested and trusted approach viewed with understandable skepticism
  - Tier 1 evolution until now – virtualisation, config management – make it easier to do the same thing better
  - Here the approach is radically different
- Our configuration management system optimised for ‘static’ hosts – working on better support for:
  - creating container images
  - configuration in Marathon
- Team not yet familiar with how to architect their services in ways suitable for container orchestration
  - e.g. used to every host being a ‘pet’

Running production services

Need places to try things out

- INDIGO DataCloud
  - STFC has funded effort for pilot deployments
  - Software is released as Docker containers
  - The INDIGO DataCloud PaaS itself makes use of Mesos and Marathon
- Will deploy pilot services at RAL using Mesos
  - Gives us operational experience running externally-visible services in a production setting
  - example: APEL accounting service
- Build on that experience
  - Consider running new services in containers before migrating existing production services

INDIGO - DataCloud

Commercial clouds

Related work, an activity part of the RCUK Cloud Working Group

- Most HEP activity on commercial clouds has involved
  - cloud provider specific APIs
    - Nova, EC2, Azure, GCE, ...
  - and/or cloud provider specific services
- Alternative approach
  - use Kubernetes as a way of providing portability between on-premise resources & **multiple** commercial clouds
  - use a single (open-source) API to run your work on multiple commercial clouds
- Have been using Google & Azure, soon AWS
  - have successfully run CMS jobs on Google & Azure

Summary & future plans

- The use of containers & container orchestration has many benefits compared to our existing approach
  - potentially higher availability with less effort & higher resource utilization – all essential to meet our strategic goals
- Future plans include
  - increased integration with our configuration management system
    - images created by Packer from configuration in Aquilon
  - use 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 & similar projects
    - running pilot services on Mesos

Questions?