Science & Technology Facilities Council

Container Orchestration - Simplifying Use of Public Clouds

Andrew Lahiff, Ian Collier

STFC Rutherford Appleton Laboratory

27th April 2017,

HEPiX Spring 2017 Workshop

Overview

- Introduction
  - Clouds in HEP
- Kubernetes
  - As an abstraction layer
  - On public clouds
  - Running LHC jobs
- Tests with CMS, ATLAS, LHCb
- Summary & future plans

Clouds in HEP

- Clouds have been used in HEP for many years
  - as a flexible infrastructure upon which traditional grid middleware can be run
  - as an alternative to traditional grid middleware
  - as a way of obtaining additional resources quickly
    - potentially very large amounts of resources, e.g. to meet a deadline
- Generally this has always involved different methods for provisioning VMs on demand, which
  - join a HTCondor pool, or
  - directly run an experiment's pilot

Clouds in HEP

- Limitations with this approach
  - each cloud provider has a different API
    - a lot of effort is required to develop software to interact with each API
  - some investigations have made use of cloud-provider specific services (sometimes many services)
    - this is a problem if you need or want to use resources from a different cloud provider
- There is no portability
  - between on-premises resources & public clouds, or
  - between different public clouds
- Is there another way?

Kubernetes

- Kubernetes is an open-source container cluster manager
  - originally developed by Google, donated to the Cloud Native Computing Foundation
  - schedules & deploys containers onto a cluster of machines
    - e.g. ensure that a specified number of instances of an application are running
  - provides service discovery, configuration & secrets, ...
  - provides access to persistent storage
- Terminology used in this talk: pod
  - smallest deployable unit of compute
  - consists of one or more containers that are always co-located, co-scheduled & run in a shared context

Why Kubernetes?

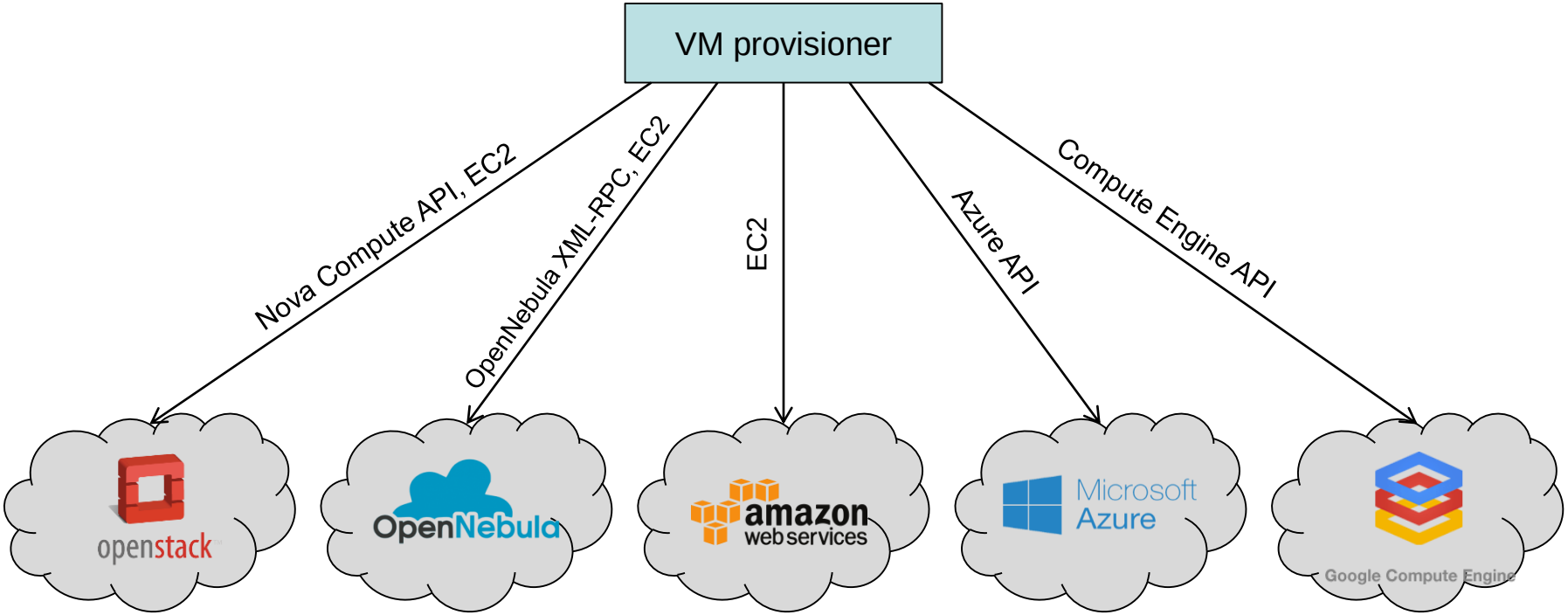
- It can be run anywhere: on premises & multiple public clouds
- Idea is to use Kubernetes as an abstraction layer
  - migrate to containerised applications managed by Kubernetes & use only the Kubernetes API
  - can then run out-of-the-box on any Kubernetes cluster
- Avoid vendor lock-in as much as possible by not using any vendor specific APIs or services
- Can make use of (or contribute to!) developments from a much bigger community than HEP
  - other people using Kubernetes want to run on multiple public clouds or burst to public clouds

Why not Mesos?

- DC/OS is available on public clouds
  - DC/OS is the Mesosphere Mesos “distribution”
  - DC/OS contains a subset of Kubernetes functionality
    - but contains over 20 services
    - in Kubernetes this is all within a couple of binaries
- One (big) limitation: security
  - Open-source DC/OS has no security enabled at all
    - communication between Mesos masters, agents & frameworks is all insecure
    - no built-in way to distribute secrets securely
  - Not ideal for running jobs from third-parties (i.e. LHC VOs)
- “Plain” Mesos also not a good choice for our use-case
  - would need to deploy complex infrastructure on each cloud

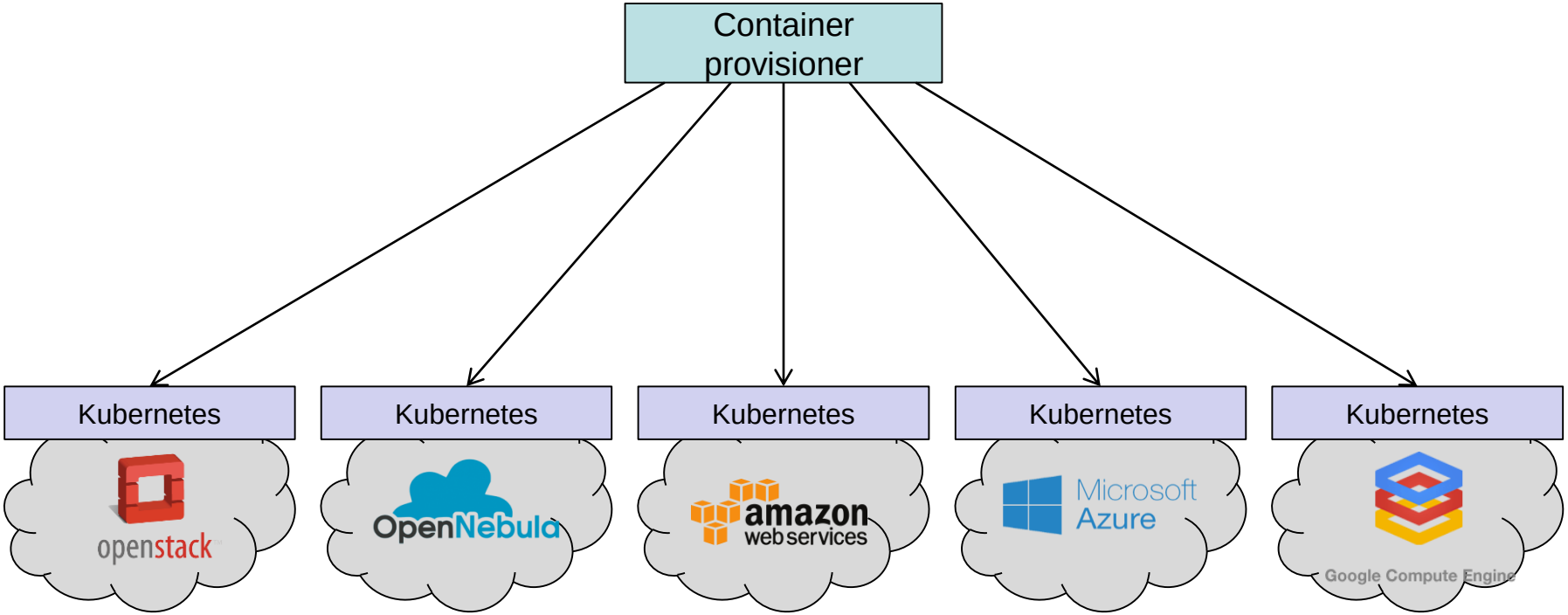
Instead of this...

- Each cloud has a different API



...we have this

- A single API for accessing multiple resources
- On-premises resources & public clouds used in exactly the same way



Kubernetes on public clouds

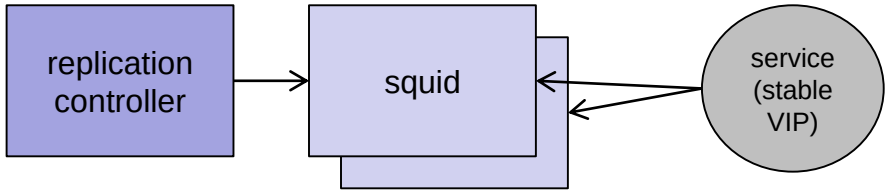
- Availability on the major cloud providers:

	Native support	Node auto-scaling
Google	✔	✔
Azure	✔	✗
AWS	✗	✗

- There are a variety of CLI tools available
  - automate deployment of Kubernetes
  - enable *production-quality* Kubernetes clusters on AWS & other clouds to be deployed quickly & easily
    - some include node auto-scaling
- Also third-party companies providing Kubernetes-as-a-Service on multiple public clouds

Running LHC jobs

- In principle this is straightforward, the main requirements are
  - squids for CVMFS & Frontier
  - auto-scaling pool of pilots/worker nodes with CVMFS
  - credentials for joining the HTCondor pool or pilot system
- Squids
  - use a Replication Controller
    - ensures a specified number of squid instances are running
  - use a Service to provide access, e.g. via http://squid:3128
    - DNS alias for a virtual IP which points at the squids

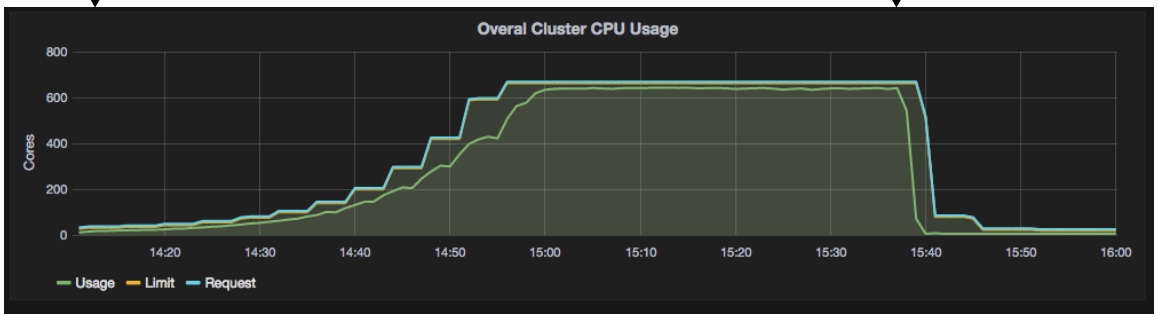


Pool of workers

- Need to have a pool of worker pods which
  - scale up if there's work
  - scale down when there's not as much work
- Initially tried using existing Kubernetes functionality only
  - replication controller + horizontal pod auto-scaler
    - scales the number of pods to maintain a target CPU load

CMS Monte-Carlo production jobs submitted
- number of pods increases

Jobs killed
- number of pods quickly decreases



Test on Azure with just under 700 cores

Pool of workers

- For CPU intensive activities this works reasonably well, but there are issues
  - Auto-scaling based on CPU usage is of course not perfect
    - CPU usage can vary during the life of a pilot (or job)
  - Kubernetes won't necessarily kill 'idle' pods when scaling down
    - quite likely that running jobs will be killed
    - this is a problem, e.g. ATLAS "lost heartbeat" errors
- Alternative approach
  - wrote a custom controller pod which creates the worker pods
    - scales up when necessary using the "vacuum" model idea
    - downscaling occurs only by letting worker pods exit

Credentials

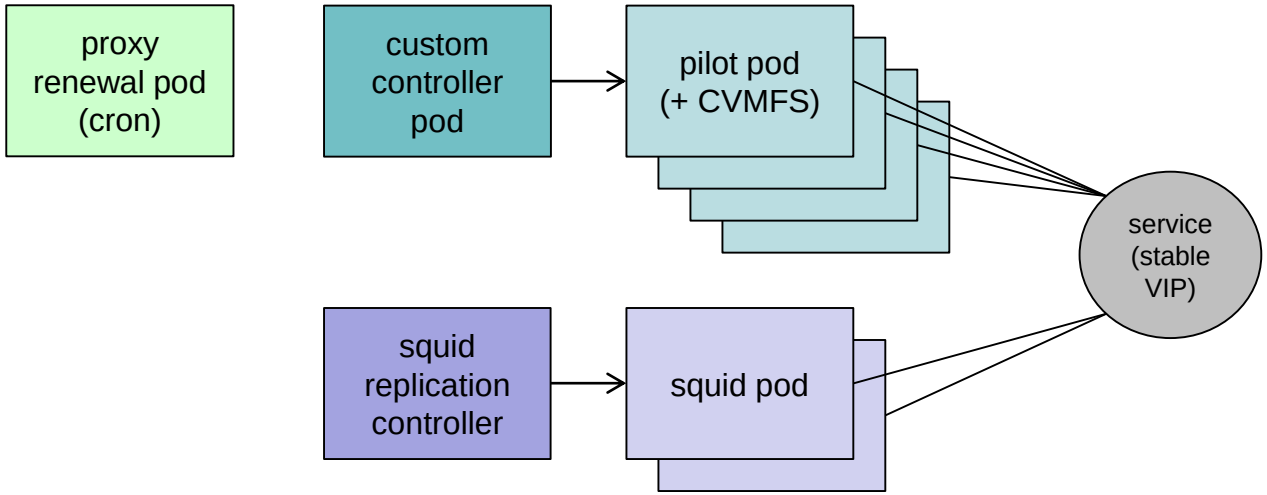
- Need to provide a X509 proxy to authenticate to the experiment's pilot framework
  - store a host certificate & key as a Kubernetes Secret
  - cron which runs a container which
    - creates a short-lived proxy from the host certificate
    - updates a Secret with this proxy
- Containers running jobs
  - proxy made available as a file
  - new containers will have the updated proxy
  - the proxy is updated in already running containers

What about CVMFS?

- It’s complicated
  - Currently Kubernetes only allows containers to have private mount namespaces
    - CVMFS in one container is invisible to others on the same host
  - Installing CVMFS directly on the agents is not an option
    - this would defeat the whole purpose of using Kubernetes as an abstraction layer
- Therefore
  - For proof-of-concept testing we’re using privileged containers both providing CVMFS & running jobs
  - Once Kubernetes supports non-private volume mounts we will be able to separate CVMFS & jobs

Running LHC jobs on Kubernetes

- Summary of what we're deploying on Kubernetes
  - it's quite straightforward



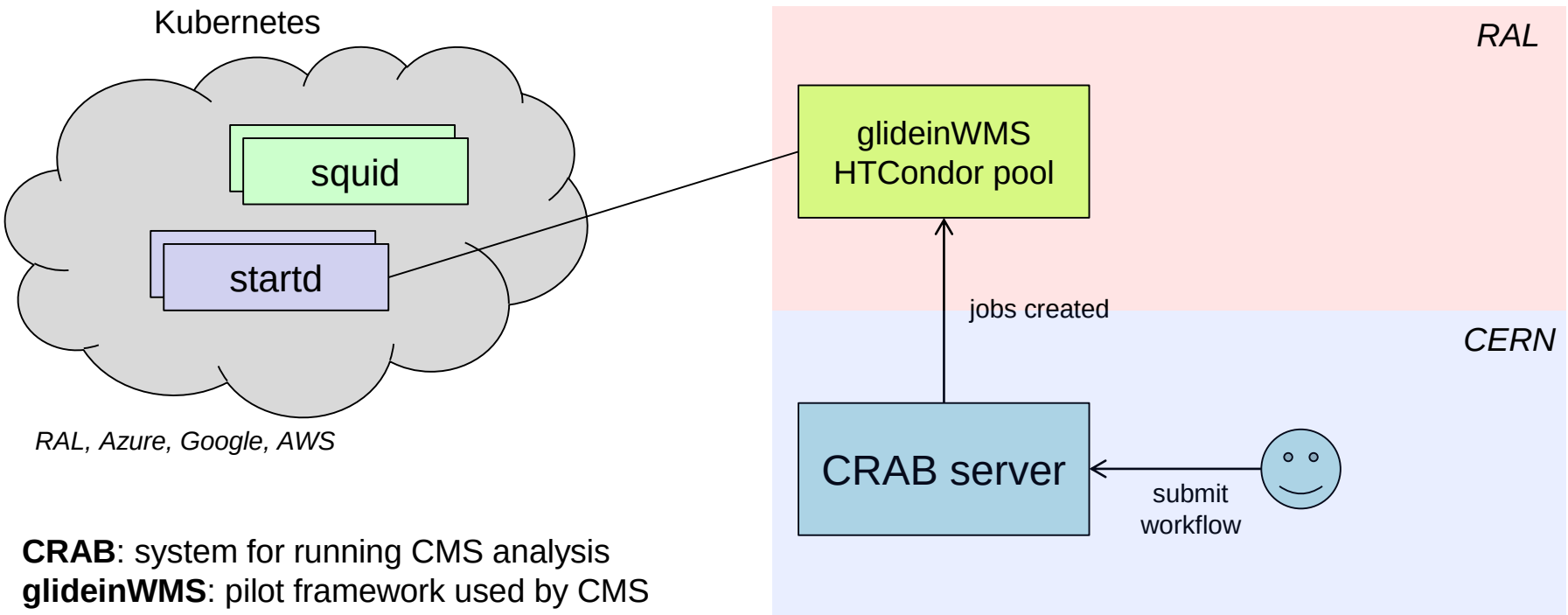

```
$ kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
atlaspilot-0jjx5                  1/1       Running   0          17m
atlaspilot-5p7z5                  1/1       Running   0          8m
squid-2127217573-9sst3            1/1       Running   0          6d
squid-2127217573-bvvn6            1/1       Running   0          17d
vacuum-atlas-2108054901-t85rk     2/2       Running   0          23s
```

Resources used for testing

- On-premises resources & public clouds
  - RAL
    - Kubernetes cluster on bare metal
  - Microsoft Azure
    - initially used a CLI tool from Weaveworks, then changed to Azure Container Service
  - Google
    - Google Container Engine
  - Amazon
    - Used StackPointCloud to provision Kubernetes clusters

Initial tests with CMS analysis jobs

- Successfully ran CRAB3 analysis jobs with a test glideinWMS instance at RAL
  - Performance (time per event, CPU efficiency, wall time, job success rate) was similar across the different resources

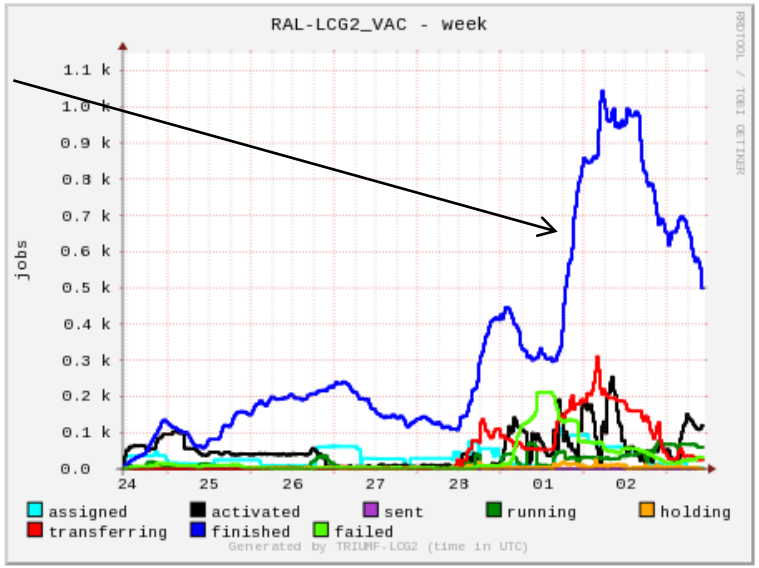


CRAB: system for running CMS analysis
glideinWMS: pilot framework used by CMS

ATLAS & LHCb

- Have run real ATLAS & LHCb jobs on Kubernetes at RAL
  - at a small scale
- Using containers which
  - run a startd which joins an existing ATLAS HTCondor pool
  - run the LHCb DIRAC agent

successful jobs

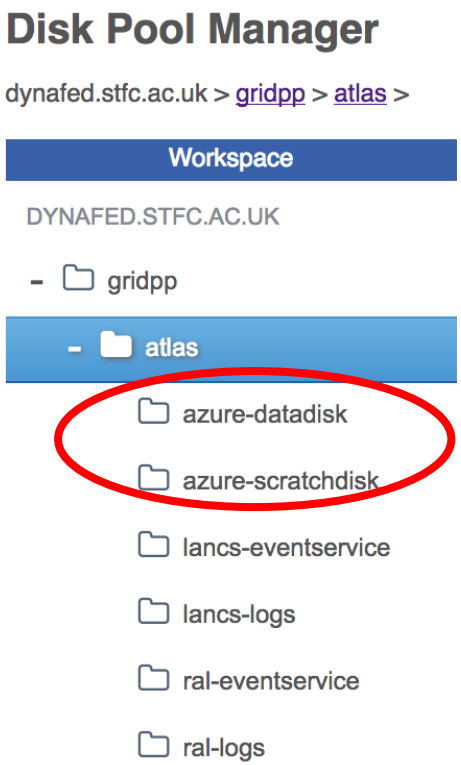


ATLAS

Cluster resource limits & usage

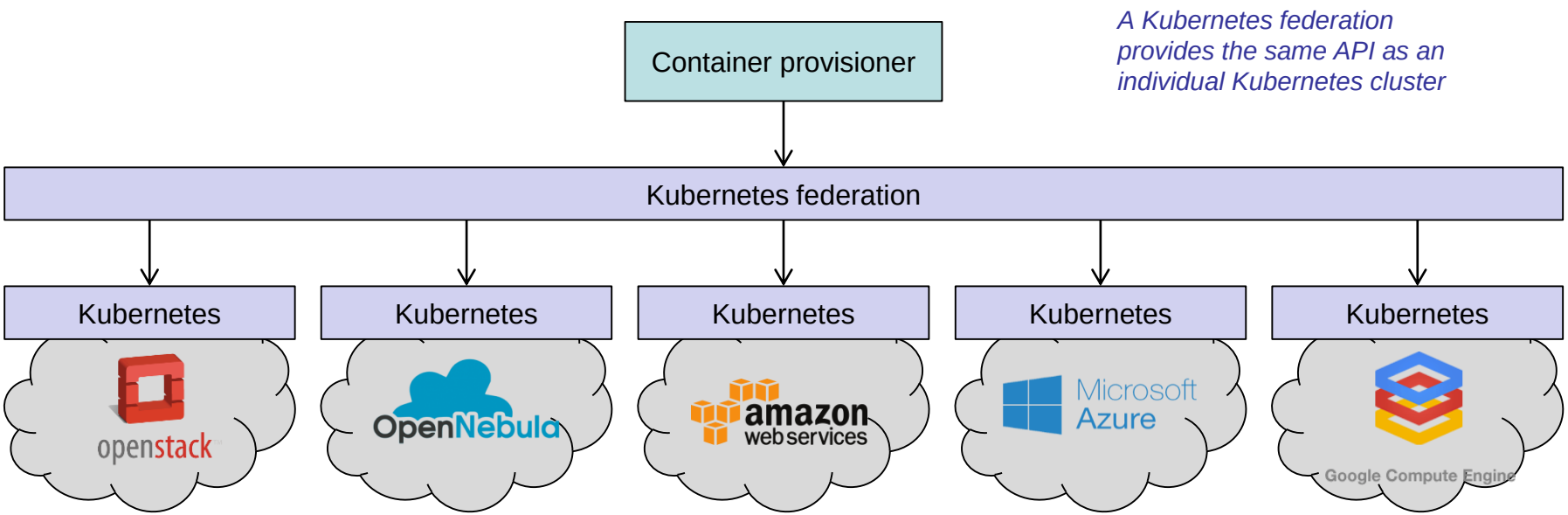
ATLAS status & plans

- Planning to do larger scale tests with ATLAS on Azure soon
  - A RAL Azure site has been configured in AGIS & PANDA
    - have successfully run test jobs
  - would like to test up to ~5000 running jobs
- Step 1
  - use RAL storage (Echo) via GridFTP
- Step 2
  - use Azure Blob storage via RAL Dynafed
    - can use gfal commands to access Azure Blob storage



Federations

- Provides the standard Kubernetes API, but applies across multiple Kubernetes clusters
- New, but eventually should make it easy to overflow from on-premise resources to public clouds



Summary

- Kubernetes is a promising way of running LHC workloads
  - enables portability between on-premises resources & public clouds
    - no longer need to do one thing for grid sites, different things for clouds
  - have successfully run ATLAS, CMS & LHCb jobs
- Looking forward to larger scale tests with real ATLAS jobs

Thankyou

Questions?

Acknowledgements

- Microsoft (Azure Research Award)
- Google
- Amazon
- Alastair Dewhurst, Peter Love, Paul Nilsson (ATLAS)
- Andrew McNab (LHCb)