Stealth Cloud: How not to waste CPU during grid to cloud transitions

To cite this article: D Bauer and S Fayer 2017 J. Phys.: Conf. Ser. 898 082001

View the article online for updates and enhancements.
Stealth Cloud: How not to waste CPU during grid to cloud transitions

D Bauer and S Fayer
HEP Group, Department of Physics, Imperial College London, London, SW7 2AZ, UK
E-mail: lcg-site-admin@imperial.ac.uk

Abstract. UKI-LT2-IC-HEP is a WLCG tier-2 comprising around 4000 job slots and 3.7 PB of storage supporting LHC and non-LHC VOs. When first looking at converting a part of our site’s grid infrastructure into a cloud based system in late 2013 we needed to ensure the continued accessibility of all of our resources during a potentially lengthy transition period. To accomplish this we brought together a number of existing middleware and cloud tools. This proved to be a viable long-term solution to maintain resource usage during extended periods of transition.

1. Introduction
In late 2013 it was decided that cloud based computing systems were stable enough to be used within our tier-2 grid site. The initial motivation for providing a production cloud was partly driven by the big LHC experiments exploring cloud technology and a need to provide a flexible system for future requirements. However, it quickly became obvious that we needed to ensure the continued accessibility of all of our resources during a potentially lengthy transition period. Moving a limited number of nodes to the cloud proved ineffective as users expected a significant number of cloud resources to be available to justify the effort of converting their workflows onto the cloud. However, moving a substantial part of the cluster into the cloud carried an inherent risk, such as the cloud nodes sitting idle while waiting for the VOs to finish their development work and other external factors. Therefore it was important for users to be able to continue submitting jobs via the classic grid interfaces (such as UMD [1], CRAB [2] & PanDA [3]) as well as being able to access the new cloud interfaces (e.g. for using custom images) at the same time.

2. Implementation
2.1. Set-up
The system we designed is based around the glideinWMS [4] software. GlideinWMS is a pilot submission framework created for CMS that uses HTCondor [5]. A local HTCondor pool of user jobs is monitored by the glideinWMS front-end process; if more computing capacity is needed a second pool (the factory) sends virtual machine (VM) start requests to a cloud (EC2) interface to increase the scale of the computing resource available.

The glideinWMS system was combined with an ARC-CE [6] to provide a grid interface. ARC-CE was picked over the other available interfaces as it readily supported HTCondor at the time. A overview of our set-up is shown in figure 1.
IOP Conf. Series: Journal of Physics: Conf. Series 898 (2017) 082001  doi:10.1088/1742-6596/898/8/082001

Figure 1. Diagram of the set-up at UKI-LT2-IC-HEP showing both the grid and cloud components.

2.2. Payload Image and Cloud Configuration

The payload images for the grid are built using a bespoke tool (CloudStamp [7]). The images are built using a standard kickstart-based EL6 (CentOS6) installation which is then customised using a single run of the Puppet [8] configuration tool. Puppet is used to install all of the grid specific software, the glideinWMS bootstrap and to configure the required pool accounts, start-up scripts, etc. Finally the payload images are compressed and uploaded to OpenStack [9]; this process has to be re-run to encapsulate any required software updates (e.g. security patches and bug-fixes to the operating system).

When an instance of the payload image is started, the glideinWMS bootstrap process is launched. This looks at the instance’s user-data provided by the cloud system and decodes it into parameters used to configure an HTCondor client and request a job. If the bootstrap remains idle for more than 10 minutes, the instance shuts itself down; this works both as a black-hole worker node prevention and as a method of providing feedback on whether there was any work available (i.e. a short-lived VM indicates that no work was available).

This system has been running in production at the UKI-LT2-IC-HEP grid site for a number of years and has been used with the following cloud configurations:

- OpenStack Icehouse & Juno: Using the in-built EC2 interface and the Glance backend for Gluster [10]. This originally used Nova networking, but was switched to using Neutron as part of the Juno upgrade.
- Kilo: The EC2 interface was no-longer bundled with the main OpenStack release and had to be installed separately and patched for VPC (Virtual Private Cloud) support. The CEPH [11] backend for Glance was used and the networking was based on Neutron.
• Mitaka: A short-term instance of Mitaka was installed for testing both the compatibility of Mitaka with the glideinWMS system as well as validating some pre-production hardware. This configuration was picked to be as simple as possible, using an unpatched external EC2 interface, Nova networking and the local/POSIX Glance backend.

2.3. Operations
Unlike a traditional batch system, there is no work queue in OpenStack; an instance request either starts a new VM immediately or fails with an out of resources error. To work around this we partition the cloud into two tenants, one for the traditional grid work and a second to run the cloud specific workloads. The quota of the grid tenant must be dynamically adjusted to use up any spare resources. If this is too large then the cloud work will never run, but if it is left too small then a large fraction of the resources will be left idle. A custom made python script manages the available tenant quota by leaving a small window (e.g. 10) of slots idle for the cloud workload to use. If all of the idle slots have been consumed then the grid quota will be decreased; if the available number of slots is greater than the fixed window size then the cloud is idle and the grid quota will be increased.

Figure 2 shows a typical example of the quota being adjusted. The system starts by running a full complement of grid jobs; when a batch of cloud jobs arrive there is an initial step as the window gets filled. After a while, as the grid jobs end, the quota is moved until the required number of slots has been reallocated to the cloud work. At the end the slots are reclaimed for grid usage and the number of running grid jobs returns to its maximum value.

![Figure 2. Cloud vs Grid System Usage. The effects of the quota adjustment can be clearly seen.](image)

A significant amount of effort was spent on setting up and maintaining an up-to-date OpenStack service. OpenStack was still undergoing major development, meaning that there was
no meaningful way to upgrade the service. Any change in version usually required a complete reinstall, with the associated downtime and re-configuration issues.

We also observed some interoperational issues between the glideinWMS and OpenStack. The most prominent one was a bug that occasionally caused a key-pair used by the glideinWMS/condor EC2 backend to track each request to remain on OpenStack even after the request was complete. Over time these key-pairs would accumulate, eventually depleting the tenant’s quota and preventing any further instances from being started. To work around this, we manually emptied the installed key-pairs during regular maintenance windows when no other VMs were running.

3. Conclusions and Outlook
We have been running grid and cloud jobs concurrently, sharing the same hardware, for nearly three years. Our set-up has proved to be stable and reliable, yet to support more recent features such as multi-core job support and longer lived VMs would require further work. In the meantime a number of alternative technologies which achieve a similar goal have become available (e.g. vcycle [12] & HTCondorCE [13]). We are currently evaluating their interoperability with our cloud resources.

References
[1] EGI Software Repository http://repository.egi.eu/
[2] Spiga, D et al 2007 The CMS remote analysis builder (CRAB) High Performance Computing – HiPC 2007: 14th International Conference, Goa, India, December 18-21, 2007. Proceedings
[3] Maeno, T et al 2008 PanDA: Distributed production and distributed analysis system for ATLAS Journal for Physics 119 part 6
[4] glideinWMS http://glideinwms.fnal.gov/doc.prd/index.html
[5] HTCondor https://research.cs.wisc.edu/htcondor/
[6] NorduGrid - Advanced Resource Connector http://www.nordugrid.org/arc/
[7] GitHub - sfayer/cloudstamp https://github.com/sfayer/cloudstamp
[8] Puppet https://puppet.com/
[9] OpenStack https://www.openstack.org/
[10] Gluster https://www.gluster.org/
[11] Weil, S et al 2006 Ceph: A scalable, high-performance distributed file system Proceedings of the 7th Symposium on Operating Systems Design and Implementation
[12] Vcycle: VM lifecycle management https://www.gridpp.ac.uk/vcycle/
[13] HTCondorCE https://twiki.grid.iu.edu/bin/view/Documentation/Release3/HTCondorCEOverview