Future Approach to tier-0 extension

B Jones¹, G McCance¹, C Cordeiro¹, D Giordano¹, S Traylen¹ and D Moreno García¹

¹ CERN IT Department

Email: ben.dylan.jones@cern.ch

Abstract. The current tier-0 processing at CERN is done on two managed sites, the CERN computer centre and the Wigner computer centre. With the proliferation of public cloud resources at increasingly competitive prices, we have been investigating how to transparently increase our compute capacity to include these providers. The approach taken has been to integrate these resources using our existing deployment and computer management tools and to provide them in a way that exposes them to users as part of the same site. The paper will describe the architecture, the toolset and the current production experiences of this model.

1. Introduction
The constant requirement for increased compute capacity, concomitant with LHC upgrades and stability, have led to various investigations in how to take advantage of the growth of public cloud resources. CERN adopted [1] a virtualization and cloud computing paradigm in the last few years, and one expected benefit was that this would enable the use of a more standard toolkit. Using commonly used tools should mean they can be leveraged to take advantage of developments in the wider industry.

Initial investigations [2] into cloud providers focused on integration into specific WLCG workflows, with the VMs provided directly to experiments. In 2016 there were two cloud deployments, firstly with IBM SoftLayer [3] and then with T-Systems [4]. A goal for these deployments was to discover the extent to which resources could be provided seamlessly. Rather than just providing cloud capacity, could we use the same toolset, the same entry points, the same staff and procedures to present the capacity as an extension of the Tier-0? Clearly there would be a requirement to be able to identify the resources as a discrete component of the overall capacity, in terms of monitoring and accounting, but also for job submission. As far as possible though, the intention was to be able to integrate capacity as just another resource in the overall pool available for workload submitted to the Tier-0.

2. HTCondor job submission
The standard way that CERN, in its function as a WLCG [5] site, provides compute resources is via a batch system. This ensures that resource usage is driven up with the use of a scheduler able to ensure different groups are allocated their fair share of capacity over time. The product used for the future batch service at CERN is HTCondor [6]. It is important to consider some of the features of HTCondor, and how they can be leveraged to seamlessly manage resources in disparate locations.
2.1. Symmetric job matching
HTCondor relies on a system of symmetric matchmaking in order to match jobs to available resources. Jobs submitted to the system can specify their requirements, or as we shall see, can be decorated to have requirements added automatically. The compute resources themselves can specify their capabilities and requirements. On both the submit side and the execute side, this is achieved through the use of ClassAds [7]. A user job is submitted to an HTCondor Schedd, which a central collector polls to build a list of pending jobs. On the execute side, the worker nodes submit their StartD ClassAds to the collector. The central negotiator then matches the ClassAds of pending jobs to those of available resources, and then reserves a slot in which the job can run.

2.2. HTCondor-CE and the Job Router
There were two slightly contradictory requirements in our cloud activities. It was desirable to both have a common submission framework, treating the extra resources as an extension of the pool, and also to be able to identify the cloud resources and specify the jobs that were able (and willing) to run on them. This was achieved using ClassAds, the HTCondor-CE [8] and the Job Router [9]. The HTCondor-CE is effectively two Schedd processes. Jobs are submitted to the Grid Schedd of the CE, and they are then routed to Batch Schedd using the Job Router. The Batch Schedd is then the entry point to the site batch system. This route on the CE allows for the decoration of the job ClassAd in order to determine which resources it is matched with in the batch pool.

The “external cloud” resources had a custom ClassAd that specified them as such, namely ‘Xbatch = True’. They also specified a requirement, via a “Start Expression” that the submitting job also is specifying it is able to run on the resources, namely ‘(WantExternalCloud =?= True)’. The jobs themselves have “WantExternalCloud” set to “True” and the requirement that ‘(XBatch =?= True)’ set by the job router. This ensures that both sides of the job match have a requirement for an attribute they are mutually expressing.

Our original expectation was that it would be fairly simple for jobs to be submitted with a ClassAd of ‘WantExternalCloud = True’ but in fact the experiments participating in the cloud exercise all achieved this final job requirement in a slightly different way. CMS indeed added an entry point to their job factories which added the ClassAd. ATLAS used APF to add ‘+remote_queue=”external_cloud”’, a slightly different method to decorate. LHCb uses CEs to delineate sites, and therefore in order for monitoring to be effective, required a separate CE for the cloud resources. In this implementation, rather than have a CE routing according to the job’s ClassAd, a dedicated CE decorated all submitted jobs with the correct ClassAd and requirements for jobs to run on the external cloud resources. ALICE, whilst able to submit jobs with the additional ClassAds, also effectively used a separate CE from normal job submissions, in order for the cloud jobs to be more easily monitored. Therefore, whilst the goal of having a single entry point was not entirely met, we showed that this would be easily achievable. During the project, we had to ensure that the activity itself was observable as a distinct pool.

2.3. Job submission across a firewall
With both SoftLayer and T-Systems, the cloud worker nodes were accessible via a public IPv4 address. In the case of T-Systems, this was via a one-to-one SNAT mapping, rather than the more simple public address assigned to the SoftLayer nodes. From the perspective of the HTCondor system, all that is strictly necessary is that the nodes have outbound connectivity to the Collector, and the Schedds (in this case the CEs). There are though some changes required to the usual way that the various components communicate if there is no firewall to traverse.
Figure 1: When submit and execute side have no firewalls to traverse, after a pslot has been assigned for a job to run, the Schedd connects to the Startd directly to send the jobs.

Figure 2: Where there is a firewall to traverse, when the Central Manager reserves a slot for the job, it uses the CCB protocol to instruct the Startd to call back to the Schedd in order to receive the job. Using the CCB, the only requirement is inbound connectivity from the Startd to HTCondor services.

The use of the CCB does have implications for the resources of the Central Manager. The Collector will need to use additional file descriptors, tuned with COLLECTOR_MAX_FILE_DESCRIPTORS [10].

3. Provisioning HTCondor worker nodes in clouds

3.1. Cloud diversity requirements
In the CERN, the private cloud is an OpenStack [11] installation. In terms of public cloud, it is not the case that we can just read Gartner’s magic quadrant [12] and conclude that there are only two, or perhaps three cloud providers and their APIs to worry about. However, both due to public tender processes, and a fragmented nascent European cloud industry, it’s likely that a wide selection of cloud providers may need to be supported. This was certainly the case for the two providers that formed the
basis of this study, and it remains the case for the ongoing Helix Nebula Science Cloud [13]. There was therefore an extra incentive to ensure that any selected tools were as platform agnostic as possible.

3.2. Provisioning stack

3.3. The stack of tools to provision, configure, manage and monitor external cloud machines is, as far as is possible, the same stack as used internally. There are though some differences, as can be seen below in Figure 3. There are tools in the provisioning layer which were adopted due to the challenges of managing cloud resources. Not all cloud providers have the same tools available for personalisation, which made some flexibility necessary. Finally, the use of some orchestration tools presented too much of an overhead to use externally.

![Machine deployment stack](image)

Figure 3: Machine deployment stack

3.4. Provisioning and personalisation
Given that Puppet [14] is used to configure the machines, the role of provisioning and personalisation is to get a VM to the point that they are running with the ability to pull their Puppet catalogue. Internally, a number of tools have grown up around the OpenStack Nova API. They are specific to the OpenStack installation at CERN, and in any case were obviously not immediately usable for the SoftLayer use case, which has a distinct API and is not OpenStack based. Additionally, SoftLayer doesn’t support cloud-init [15], but instead uses Post Provisioning scripts. At the start of the project with T-Systems, whereas cloud-init was used internally for actions such as storing ssh keys on the remote virtual machines, passing user data necessary to configure the machines was not possible. The role of a personalisation script, whether via a Post Provisioning API, or executed in a more ad-hoc way, is relatively limited. It was necessary to install the packages required for Puppet to run, a configuration file for Puppet, the extension of disks for RHEL6 machines built from images, and in the
case of T-Systems, ensuring that the hostname of the machines are set correctly. The role of the provisioning layer is more complex, as it has to manage the creation of machines using different cloud APIs. Even in the case of T-Systems, which is OpenStack based, the creation of virtual machines for use in the CERN batch pool was not the same as the internal OpenStack cloud. Some of these differences were due to different choices of products within the OpenStack suite. For instance, T-Systems used the Neutron API for network operations, whereas the CERN cloud had until recently used nova-network. Other differences were due to the way that the clouds were configured, such as the use of floating-ip APIs in order to provision public IPv4 addresses to resources. There were differences in the versions of tools required to connect to the implementations of the OpenStack APIs at T-Systems as opposed to CERN, to the extent that there was no mix of library versions that would allow the management of both internal and external cloud resources. It should be noted that the number of API calls needed to build a machine can often be considerable. For T-Systems, it is necessary to create networks, subnets, virtual router interfaces, ports, storage volumes, floating IPs as well as the machines themselves.

![Figure 4: Cloud provisioning tools](image)

3.5. Terraform experience
Terraform was selected as the tool to provision the virtual machines. APIs and SDKs, whilst crucial components of public cloud offerings, necessitate a considerable amount of development. Cloud providers often provide CLI tools to provision machines, but these tend to be geared towards building
limited number of machines interactively. In the case of OpenStack CLI tools, separate tools are required for each different API touch point, with – for example – the ID of a freshly created subnet being required for the creation of a network port. Put simply, they do not scale. Cloud Formation [16] and Heat [17] are attempts to solve some of the scale problem, but suffer from being implementation specific. HTCondor has its own tool to burst pools to clouds, in condor_annex. This has a slightly different goal, in that it provides a short-term expansion of capacity in the cloud, rather than the extension of flat capacity. It is also focused on first AWS and now GCE, rather than having to support a more disparate collection of providers.

Terraform [18] is a mature and widely used provisioning tool, which supports a number of different cloud implementations. It is an imperfect judge of the popularity of a project, but it currently has over 7k stars on GitHub [19], as well as a book [20]. There are “providers” written for a number of different cloud providers, including support for SoftLayer and OpenStack. It also has a “provisioner” feature, which enabled us to work around the lack of support for user data in the early days of T-Systems. Provisioners were also used for the touch points with CERN infrastructure, such as registering the nodes in Foreman [21] in order for them to be registered in the Puppet infrastructure.

3.6. Puppet and Foreman

3.6.1. Separation of infrastructure

It is presumed that cloud machines should have a different security profile to on-premise machines. Certainly some updates, such as to PuppetDB come from Puppet clients, and therefore having separate infrastructure to an extent seems prudent. Therefore we deployed separate masters and separate PuppetDB instances for the external machines. Foreman, being a read only datasource, we kept as a single installation, which meant we had a single inventory for the whole pool.

3.6.2. DNS Subdomains

Machines were added to Foreman with DNS subdomains, *.softlayer.cern.ch and *.tsy.cern.ch respectively. They were added as such at provisioning time by a terraform local-exec provisioner. This allowed us to alter configuration based on the fqdn of the machine. In puppet a $::datacentre fact was constructed from the subdomain, and a $::is_external boolean via a datacentre lookup.

3.6.3. Secure deployment of certificates

Both puppet and our HTCondor installation rely on X509 certificates. Puppet uses the certificates for communication with the Puppet Master. We use X509, or specifically GSI, to authenticate HTCondor registered machines – both the worker nodes and the infrastructure roles. Internally we use grid certificates for both. For the cloud machines, honouring the principle of separation of infrastructure, we used Puppet’s own CA. This required ensuring that the HTCondor Central Manager and CEs trusted the Puppet CA that we deployed. It also meant that we could use the Puppet CA autosign facility to register machines. It was necessary just to find a method to sign off machines that would have acceptable security.

Deploying secrets to cloud machines is sometimes achieved using node metadata. This is however a completely insecure method, since metadata can be accessed by any process on a node. This is particularly problematic for batch systems. In order to have the certificate signoff request (CSR) signed off, we relied on adding a challengePassword field to the request. The password is constructed from a secret known to the signoff process, the fqdn of the target, and the datetime. When the signoff request is received, the process ensures that it has the secret, that the fqdn matches, and that the request was received in a short period from the datetime. The signoff process then connects to the cloud API implied by the dns subdomain, and ensures a device of that name exists. The secret used by the worker node to add the attribute to the CSR was deployed using a terraform provider. With the two cloud activities we performed, this meant that it was necessary to have inbound connectivity.
4. Conclusions

With the help of the Job Router and the CCB, it is relatively simple to add a HTCondor pool, machines which located in disparate locations, including outside firewalls. The challenges for the project were to be found in provisioning of resources, and having to deal with different cloud APIs with different levels of maturity and readiness. The ability to use standard industry tools means the cost of supporting heterogeneity is reduced. Even taking the example of OpenStack, where there is a lot of experience within CERN, using external tools was helpful. The use of Neutron APIs, floating IPs and some volume operations had never been necessary in the internal cloud. These features are indeed not present in some public, but less widely used, tools such as libcloud. These features were supported in terraform. Using the right tools meant that costly development effort was avoided. Clearly stability of APIs from cloud providers is also important to avoid investing too much effort.

In terms of usage by experiments, having an additional CE, rather than relying purely on ClassAd decoration, was clearly useful. In the same way, having configuration attributes in Puppet that could also be fed into HTCondor for the datacentre was useful. Both in terms of being able to make configuration decisions, and also to provide monitoring and accounting of capacity provided in the cloud.

References

[1] Bell T et al, Review of CERN Data Centre Infrastructure, 2012 J. Phys.: Conf Ser. 396 042002
[2] Giordano D et al, CERN Computing in Commercial Clouds, to appear in J. Phys.: Conf. Ser., 2016
[3] http://www.softlayer.com
[4] https://www.t-systems.com
[5] http://wlcg.web.cern.ch
[6] http://research.cs.wisc.edu/htcondor/description.html
[7] http://research.cs.wisc.edu/htcondor/manual/v8.5/4_1HTCondor_s_ClassAd.html Accessed: 2017-02-19
[8] https://wlcg-ops.web.cern.ch/htcondor-ce
[9] http://research.cs.wisc.edu/htcondor/manual/v8.5/5_4HTCondor_Job.html
[10] http://research.cs.wisc.edu/htcondor/manual/v8.5/3_9Networking_includes.html#SECTION004940000000000000
[11] https://www.openstack.org
[12] https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519 Accessed: 2017-02-19
[13] http://www.hnscloud.eu/events/hnscloud-pcp-tender-results-announcement-ceremony-design-phase-kick-2-november-2016-lyon
[14] https://docs.puppet.com/puppet/
[15] https://cloud-init.io
[16] https://aws.amazon.com/cloudformation/
[17] https://wiki.openstack.org/wiki/Heat
[18] https://www.terraform.io
[19] https://github.com/hashicorp/terraform Accessed: 2017-02-19
[20] https://terraformbook.com
[21] https://theforeman.org