The vacuum platform

A McNab

1 School of Physics and Astronomy, University of Manchester, UK
E-mail: andrew.mcnab@cern.ch

Abstract. This paper describes GridPP’s Vacuum Platform for managing virtual machines (VMs), which has been used to run production workloads for WLCG and other HEP experiments. The platform provides a uniform interface between VMs and the sites they run at, whether the site is organised as an Infrastructure-as-a-Service cloud system such as OpenStack, or an Infrastructure-as-a-Client system such as Vac. The paper describes our experience in using this platform, in developing and operating VM lifecycle managers Vac and Vcycle, and in interacting with VMs provided by LHCb, ATLAS, ALICE, CMS, and the GridPP DIRAC service to run production workloads.

1. Introduction

A previous CHEP paper[1] in 2014 presented the Vacuum model, with Vac as its first implementation, which can be summarised as:

*The Vacuum model can be defined as a scenario in which virtual machines are created and contextualized for experiments by the resource provider. The contextualization procedures are supplied in advance by the experiments and launch clients within the virtual machines to obtain work from the experiments’ central queue of tasks.*

Here “resource provider” is understood to be the entity which decides to provide VM instances to the experiments at a given time. Usually the site will also be the resource provider, but this function may be delegated to an external entity. For example, by giving it an account on the site’s Infrastructure-as-a-service cloud system.

The Vac[1] and Vcycle[2] VM lifecycle management systems were developed in tandem with the corresponding LHCb virtual machine definition[3] and so the systems were naturally compatible. However with the separate development of CMS vacuum VMs[4] and the HTCondor Vacuum system[5], it became clear that a proper specification of the interfaces between the VMs and the VM lifecycle managers were needed. This was further underlined by the developments to converge the VMs used for ATLAS[6] on Vac and Cloud platforms.

We refer to this collection of interfaces as “the Vacuum Platform” and published initials drafts in January 2016 with a detailed specification given by HEP Software Foundation technical note HSF-TN-2016-04[7], “Vacuum Platform”, published in October 2016.

2. Design

The overarching design aim is that a single VM definition for each experiment will be able to run at all sites, irrespective of the underlying technology. For example, Vac provides VM
lifecycle management on autonomous VM factories; and Vcycle supports VM management on OpenStack[8], Amazon EC2[9] (and compatible implementations), OCCI[10], and MS Azure[11] via per-API plugins.

The Vacuum Platform achieves this universality by presenting uniform APIs between Virtual Machines and the Virtual Machine Lifecycle Managers which control them. The basis of the platform are the $JOBOUTPUTS, VacQuery, and VacUserData interfaces plus requirements for using Machine/Job Features, APEL[12], and GOCDB[13]. The specification also describes how experiments should publish the boot images and contextualization of their VMs.

The design of the platform has been driven by operational experience gained by GridPP and the experiments in running jobs in VM-based systems. We are especially interested in ways of reducing the effort required to run a site and the lightweight site concept. In particular, to reduce the amount of manual effort required to install, configure, and maintain sites.

Where possible, the specification encourages the use of OpenStack APIs for booting and metadata discovery, either on native OpenStack or by compatible implementations such as Vac. For other cases, alternatives are described which can also be supported for maximum portability of VMs.

The platform is designed to integrate with the existing WLCG/EGI infrastructure, including at mixed sites using conventional grid middleware and VM-based approaches at the same time. Accounting information is reported using APEL and it is encouraged that resources will be registered in GOCDB as with other CE types. Limits on available CPU time, number of processors etc are published to the VMs and jobs using the WLCG Machine/Job Features (MJF) mechanism[14].

Some components of the API may also be of wider interest. These include the VacMon specification for communicating VM status between co-operating VMs at a Vac site, which is also being used by GridPP to publish monitoring messages to a central site; and the $JOBOUTPUTS extension to Machine/Job Features which allows VMs (or batch jobs) to communicate log files to sites.

3. Interfaces
The technical note describes the use of the following interfaces as part of the Vacuum Platform, to allow sites to run VMs provided by the LHC experiments and others, using compatible VM lifecycle managers.

3.1. Machine/Job Features mechanism
The MJF mechanism is used by the resource provider to communicate VM lifetime information to the VM. Additional information, such as the memory and disk limits may be made available following the Machine/Job Features specification. These limits have traditionally been supplied to running jobs by batch systems using native APIs. MJF has been developed by WLCG to provide a common API for this, and the Vacuum Platform uses MJF as its native API for communicating with jobs.

Since a shared filesystem is not typically available, MJF is used in its HTTP(S) mode with the $MACHINEFEATURES and $JOBFEATURES variables containing the base URL rather than a base filesystem path. Since virtualized environments may provide secure private networks, it may be acceptable to use plain HTTP rather than HTTPS.

3.2. $JOBOUTPUTS
The $JOBOUTPUTS mechanism is an extension to Machine/Job Features defined in the Vacuum Platform technical note. It allows the VM to communicate to the resource provider. This may be used for sharing logs but its principal use is to return the shutdown_message file when the VM finishes to explain why the VM stopped. The VM lifecycle manager can use this
information to decide whether to create more VMs of that type. As with MJF, the HTTP(S) mode is used. Authentication is either based on the use of a trustworthy private network, or the use of an X.509 credential within the VM of which the VM lifecycle manager is aware (and will typically have supplied to the VM.)

3.3. Image URLs
Experiments are encouraged to publish the HTTPS URL of the boot image used by their VMs, which the VM lifecycle managers can automatically download and cache when new versions are available. This removes the need for site administrators to manually fetch new versions, and gives the experiments operations team direct control of which version is in production across the sites.

3.4. VacUserData templates
Cloud systems such as Amazon EC2, OpenStack, and Google Cloud Platform[15] support providing a “user-data” file to a virtual machine to contextualize it. Vac emulates OpenStack’s method of providing the file via HTTP.

For their part, the boot images support user-data by fetching the file from a known location and extracting configuration directives or shell scripts from it. Cloud Init[16] is the most common procedure for this, implemented by CernVM[17] and many other VM boot images, and consisting of a text file which can contain multiple configuration files and scripts.

The Vacuum Platform specification does not specify which contextualization procedure VMs must use, but does require that VM lifecycle managers support user-data files in general, including the Cloud Init format, and that the VM lifecycle manager obtains its copy of the user-data file from an HTTPS URL nominated by the experiment.

Additionally, a series of textual pattern substitutions will be applied, consisting of standard substitutions such as the hostname and CernVM-FS[18] HTTP proxy the VM should use, and any substitutions required by the experiment. As such the user-data file really a template from which the user-data file given to a particular instance of the experiment’s VM is created.

One substitution of particular importance allows the insertion of an X.509 Proxy[19] generated by the VM lifecycle manager using an appropriate X.509 certificate and key it possesses. This proxy can then be used by the VM to authenticate to the central services run by the experiment and obtain jobs to run, and this procedure gives the experiment the final say in whether VMs created by a particular resource provider are permitted to do this.

3.5. VacQuery messages
The VacQuery protocol specifies queries and status messages which can be sent over UDP as short JSON[21] documents.

The principal use of the VacQuery protocol is to allow Vac factories to gather information from their neighbours about what VMs are running for what experiments. This is done using the machinetypes_query and machinetype_status UDP messages. Factory and machine message pairs are also supported which can be used for automated or manual monitoring of sites. The protocol has been designed to keep JSON messages and IP headers below the ethernet MTU of 1500 bytes to avoid fragmentation on local networks. Where the number of response messages is not known when the query is issued (for example, when requesting a status message for every VM running on particular VM factory), then each message includes a count of how many responses to expect. This allows the querying agent to resend queries in case of UDP message loss until at least one response is received and then until all the expected replies are received.

The factory messages factory_query and factory_status are intended for monitoring the state of the VM factories themselves, including generic Linux health metrics such as free disk and
CPU load. As well as manual queries by administrators using the provided command-line tool, these messages may also be used for automated site monitoring and alarms.

3.6. VacMon
Components of the VacQuery protocol have been reused to produce Ganglia-style[20] monitoring of individual sites or groups of sites. This “VacMon” service periodically receives factory_status and machinetype_status messages from Vac and Vcycle daemons sent on UDP port 8884. As VacQuery messages are sent as JSON documents, they may be conveniently recorded in document-orientated databases such as ElasticSearch.

A prototype implementation[22] of the VacMon system has been produced and is used for daily monitoring of participating sites. Figure 1 shows one of the many charts it produces, covering nine sites.

![Running processors by site](image)

**Figure 1.** VacMon monitoring of nine Vac and Vcycle sites

3.7. APEL
The specification requires that VM lifecycle managers should support reporting of usage to the central APEL service with messages of the type “APEL-individual-job-message”. These are the records used for conventional grid sites, rather than those developed for cloud resources.
3.8. GOCDB
Vacuum Platform resources should be registered in the GOCDB entries for the site, using appropriate service types. To facilitate this, the service types `uk.ac.gridpp.vac` and `uk.ac.gridpp.vcycle` have been created for Vac and Vcycle spaces and registered with the OGF GLUE working group[23].

Registration allows new Vacuum Platform resources to be discovered more easily by experiments, and permits the declaration of downtimes for these services.

4. Experience
The participating sites have been able to run jobs for the LHC experiments and others with much simpler software than requirement to run a traditional WLCG or EGI grid site. The simplicity of the Vacuum Platform and its pull model means suitable VM lifecycle managers can be implemented in only a few thousand lines of code. (This is the case for both Vac and Vcycle.) A small codebase makes it easier for system managers to identify problems they are seeing and test and propose fixes themselves, as has been observed. We have received very positive feedback about the lightweight and reliable characteristics of the platform and its implementations, including in a paper presented at CHEP 2015[24].

The simplicity of the platform has also made it straightforward to apply to additional cloud platforms and VM supporting additional experiments.

Operationally, experiment members have noted how desirable it is to be able to use the same VM definition across all underlying cloud platforms and sites. This leads to a cycle of testing new features of the VMs at sites where debugging is convenient, and then rolling out the new versions across all sites using the single HTTPS publication location for the VacUserData templates. It should be pointed out that is further simplified by the excellent cross-platform support provided by CernVM.

5. Further work
During 2016 features were implemented in Vac which go beyond the Vacuum Platform and were released after its publication.

These features include “Vacuum Pipes” where by a single file created by the experiment and retrieved by HTTPS can specify the location of the boot image and user-data file to use, and parameters such as suitable limits on VM lifetime. This system typically allows the addition of an experiment to a site’s configuration by merely stating the Vacuum Pipe URL and setting the desired target share of the resources.

Additional fields have also been added to the VacQuery messages, allowing the monitoring of additional hardware attributes such as temperature. These additions will be specified in a future version of the Vacuum Platform specification.
6. Conclusion
The paper has presented the Vacuum Platform and the HEP Software Foundation technical note which describes it. This platform, implemented by the Vac and Vcycle virtual machine lifecycle managers, is currently providing a uniform environment for VMs for ATLAS, ALICE, LHCb, and the GridPP DIRAC service, running jobs in production at nine research and commercial sites using various underlying virtualization technologies.

References
[1] McNab A et al 2014 J. Phys.: Conf. Ser. 513 032065
[2] Love P et al 2015 J. Phys.: Conf. Ser. 664 022031
[3] Stagni F et al 2015 J. Phys.: Conf. Ser. 664 022030
[4] Andronis A et al 2015 J. Phys.: Conf. Ser. 664 022012
[5] Lahiff A 2015, “Implementation of the vacuum model using HTCondor”, presented at the CHEP 2015 conference. https://indico.cern.ch/event/304944/contributions/1672427/
[6] Taylor R et al, “Consolidation of Cloud Computing in ATLAS”, presented at the CHEP 2016 conference.
[7] McNab A, HSF-TN-2016-04 “The Vacuum Platform” (HEP Software Foundation)
[8] http://www.openstack.org/
[9] https://aws.amazon.com/ec2/
[10] Nyren R et al, GFD-P-R.183 “The OCCI Core Specification” (Open Grid Forum)
[11] https://azure.microsoft.com/
[12] Jiang M et al. 2011 “An APEL Tool Based CPU Usage Accounting Infrastructure for Large Scale Computing Grids” In: Lin S., Yen E. (eds) “Data Driven e-Science”. Springer, New York, NY
[13] Mathieu G et al 2010 J. Phys.: Conf. Ser. 219 062021
[14] Alef M et al, HSF-TN-2016-02 “Machine/Job Features Specification” (HEP Software Foundation)
[15] https://cloud.google.com/
[16] https://cloudinit.readthedocs.io/
[17] Blomer J et al. 2014 J. Phys.: Conf. Ser. 513 032007
[18] Blomer J et al. 2012 J. Phys.: Conf. Ser. 396 052013
[19] Tuecke S et al, RFC3820 “Internet X.509 Public Key Infrastructure Proxy Certificate Profile” (Internet Engineering Task Force)
[20] Sacerdoti F et al. “Wide Area Cluster Monitoring with Ganglia,” IEEE International Conference on Cluster Computing, Hong Kong
[21] Bray T, RFC7159 “The JavaScript Object Notation (JSON) Data Interchange Format” (Internet Engineering Task Force)
[22] http://vacmon.gridpp.ac.uk/
[23] http://redmine.ogf.org/projects/glue-wg/
[24] Jones S H, Fay R, Bland J, “Migrating to virtual technology with Vac”, presented at the CHEP 2016 conference.