Journal of Physics: Conference Series **664** (2015) 022031 doi:10.1088/1742-6596/664/2/022031

Managing virtual machines with Vac and Vcycle

A. McNab[1], P. Love[2], and E. MacMahon[3]

[1] School of Physics and Astronomy, University of Manchester, UK
[2] Department of Physics, Lancaster University, UK
[3] Department of Physics, University of Oxford, UK

E-mail: andrew.mcnab@cern.ch

Abstract. We compare the Vac and Vcycle virtual machine lifecycle managers and our experiences in providing production job execution services for ATLAS, CMS, LHCb, and the GridPP VO at sites in the UK, France and at CERN. In both the Vac and Vcycle systems, the virtual machines are created outside of the experiment's job submission and pilot framework. In the case of Vac, a daemon runs on each physical host which manages a pool of virtual machines on that host, and a peer-to-peer UDP protocol is used to achieve the desired target shares between experiments across the site. In the case of Vcycle, a daemon manages a pool of virtual machines on an Infrastructure-as-a-Service cloud system such as OpenStack, and has within itself enough information to create the types of virtual machines to achieve the desired target shares. Both systems allow unused shares for one experiment to temporarily taken up by other experiements with work to be done. The virtual machine lifecycle is managed with a minimum of information, gathered from the virtual machine creation mechanism (such as libvirt or OpenStack) and using the proposed Machine/Job Features API from WLCG. We demonstrate that the same virtual machine designs can be used to run production jobs on Vac and Vcycle/OpenStack sites for ATLAS, CMS, LHCb, and GridPP, and that these technologies allow sites to be operated in a reliable and robust way.

1. Introduction

A previous paper[1] in 2014 presented the Vacuum model and Vac as its first implementation:

> *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.*

In this paper we describe further developments to Vac, and the introduction of Vcycle, which implements the vacuum model for Infrastructure-as-a-Service (IaaS) cloud systems such as OpenStack[2].

2. Pilot VM model

Both Vac and Vcycle implementations are designed to manage VMs that have a well-defined lifecycle, in particular, that the VM itself shuts itself down if it has no work to do. Beyond this there are no requirements on the behaviour of the VMs but they have the option of providing information about why they finished or confirmation that they are still running correctly.

The Pilot VM model implicitly assumes that a series of identical VMs can be created for the experiment and that each of them will discover what portion of work they should carry out, or stop if none is avaiable. The model was developed by analogy with pilot jobs at conventional grid sites fetching payload jobs from a central task queue. However, Vac and Vcycle can equally well manage VMs with a different internal model, such as the client of an experiment's event server, a local batch queue worker node, or even a parallel analysis worker for a system like PROOF[3].

The outcome of the VM can be reported as a "shutdown message". The message consists of a three digit code followed by human readable text, in a similar way to status messages in internet protocols such as HTTP[4] and SMTP[5]. The values are listed in Table 1. This scheme provides room to insert more numbers for finer-grained information in the future.

Table 1. Shutdown codes and messages

100 Shutdown as requested by the VM's host/hypervisor
200 Intended work completed ok
300 No more work available from task queue
400 Site/host/VM is currently banned/disabled from receiving more work
500 Problem detected with environment/VM provided by the site
600 Grid-wide problem with job agent or application within VM
700 Transient problem with job agent or application within VM

Vac and Vcycle treat messages in the range 300 to 699 as aborts which are more significant than transient problems. If the VM does not supply a shutdown message, then aborts are identified by comparing the duration of the VM to the parameter fizzle_seconds. If a class of VM, referred to in the Vac and Vcycle configurations as a "vmtype", has a VM instance which aborts, then creation of more instances of that vmtype is blocked for the duration of its parameter backoff_seconds. This backoff algorithm is applied to all of the VM slots which are managed together as a "space", corresponding to a Compute Element and batch system in a conventional grid site.

Shutdown messages are communicated via the machineoutputs extension to the WLCG Machine/Job Features mechanism described below. This mechanism can also be used to communicate a heartbeat signal to the VM lifecycle manager by updating the modification time of a nominated heartbeat file in the Vac or Vcycle configuration. It is possible to nominate a log file which is updated frequently in the normal operation of the VM, or to update a special-purpose heartbeat file.

3. Vac motivation and implementation

Vac was developed in parallel with the Vacuum model[1] itself and the LHCb Pilot VM architecture described recently[6]. Vac creates VMs on behalf of the resource provider directly on the hypervisor machine or "factory" on which it runs, using the libvirt toolkit[7]. The Vac daemon acts as an autonomous agent, taking decisions about which VMs to create and possibly stop based on their observed behaviour, information gathered from other Vac factory machines within the same space, and the configuration files created by the resource provider. This design aims for maximum robustness, as individual factories can continue to create VMs even if their peers fail or the mechanism by which configurations are updated fails.

The Vac configuration file scheme has been designed with manual and automated modes in mind, such as Puppet[8]. Files ending in .conf in /etc/vac.d are read in alphanumeric

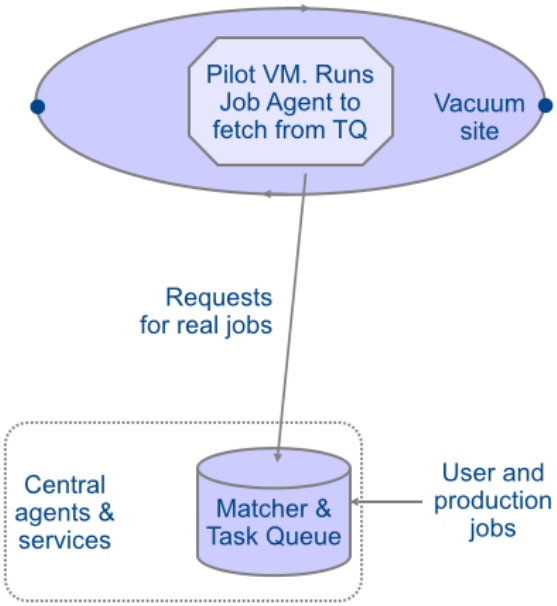

Figure 1. The Vacuum model as implemented by Vac

order followed by the file /etc/vac.conf . All these files are merged with more recent values overriding any present in earlier files. This allows fragments of configuration file to be installed and assembled by Vac at the start of each cycle to enable per site, per space, and per machine configuration file fragments to be combined. This design aims to avoid the need to build a complete monolithic configuration file for Vac, with a preprocessor or a system like Puppet's Hiera.

The Vac daemon gathers information and takes actions in cycles which occur approximately once per minute. At the start of each cycle, the configuration files are reread, the running VMs are examined using the libvirt API, and if new VMs are to be created, other Vac factories are interrogated using the VacQuery UDP protocol. Some state information about the VMs, such as their vmtype, cannot be discovered from libvirt and is saved as files, from where it is rebuilt at the start of each cycle. This design removes any need to restart Vac daemons if configuration files are updated, and ensures that Vac recovers form failures at the level of the VMs or the virtualization layer on the factory. The Vac software can typically be updated between compatible versions without disrupting running VMs, and this is extremely convenient when apply bug or security fixes.

To handle VacQuery UDP messages, the Vac daemon spawns a Vac Responder process from the main Vac Factory process. The protocol involves the factories sending queries to UDP port 995 in the form of JSON[9] structures. Replies are made by the responder to the sender on its chosen port again in the form of a JSON structure, with information about which VMs are running for which vmtypes, and when each vmtype last experienced a VM abort and why. Each factory is given a list of all the factories in the space as part of its configuration and factories which fail to respond are assumed to have failed. Packet loss or delays may result in less accurate decisions in sharing capacity across the space, but since the bulk of the decision making is done within each factory using information obtained directly from the VMs and libvirt,

the possibilities for these problems to have a significant effect on the operation of running VMs are minimised.

It is important to note that the VacQuery UDP mechanism is only used when deciding what type of VM to create, and so relatively few of these queries are needed over the course of a day. On a factory running 24 VMs each lasting 24 hours, a set of VacQuery requests would be sent only once an hour on average. In a space with 360 such factories (and so 8640 VMs), a responder service would only have have to deal with a UDP packet every 10 seconds. Since the Vac spaces are defined by configuration rather than by association of worker nodes with a headnode or gateway machine, then the number of factories communicating with each other in each space can readily be adjusted.

Through libvirt and its dnsmasq daemon, Vac provides DHCP and DNS services to the VMs which are created on a private NAT network within each factory in the IP address range 169.254.169.0 - 169.254.169.253. On this network, the factory appears at 169.254.169.254 which is the so-called Magic IP in IaaS frameworks such as OpenStack at which an HTTP metadata service is often provided. It is possible for the resource provider to provide IaaS-like services at this IP address, and it is envisaged this mode will be supported by Vac itself in the future.

Vac supplies the WLCG Machine/Job Features[10] (MJF) directories to the VMs using an internal NFS server on each factory, and currently requires that VM contextualizations which wish to use MJF check for the availibility of these NFS directories. In addition, Vac provides a machineoutputs directory on the same NFS server which the VM can write log files, heartbeat files, and shutdown messages to. The contents of this directory are automatically saved on the factory for debugging purposes when each VM finishes.

Vac also configures a Virtual Network Console (VNC) for each VM on the local 127.0.0.1 address, and this allows the resource provider to run a VNC client on the factory and see the console messages.

Finally, Vac can provide the VM with access to a fast logical partition for use with virtio[11] paravirtual disk interfaces, typically mounted at /scratch within the VM. Vac creates and manages these logical partitions within a nominated disk volume group.

4. Vcycle motivation and implementation

Following the successful deployment of Vac for production LHCb and ATLAS workloads, the Vacuum model was applied to virtual machines managed by IaaS systems such as OpenStack and the APIs they expose, rather than libvirt on the host itself. The resulting daemon, Vcycle, shares much of the design, configuration, and code used by Vac and uses the same Pilot VM API with a few necessary modifications for the IaaS environment.

As with Vac, no communication is needed between the experiment and the Vcycle instance and all decisions are based on the observed behaviour of the Pilot VMs which in turn handle all communication with the experiment's workload management system. Figure 2 shows the resulting architecture, where the Vcycle daemon can be placed within the site, within the central infrastructure of the experiment, or at a third party site which has access to the IaaS service. Since Vcycle is experiment-neutral and the relative shares of different experiments are given in the Vcycle configuration, this flexibility allows the resources to be managed in a way which reflects funding and policy decisions associated with those resources. Furthermore, each Vcycle instance can managed VMs in one or more spaces associated with tenancies or accounts at more than one remote IaaS service.

For example, a High Energy Physics laboratory might run Vcycle to manage some of the capacity of its own cloud infrastructure for the set of experiments it supports; a university HEP group might run Vcycle to manage capacity it has been allocated on a cloud facility operated with the university's IT services department; an experiment might run several Vcycle instances to manage resources at national cloud services which have no associated HEP staff; and a

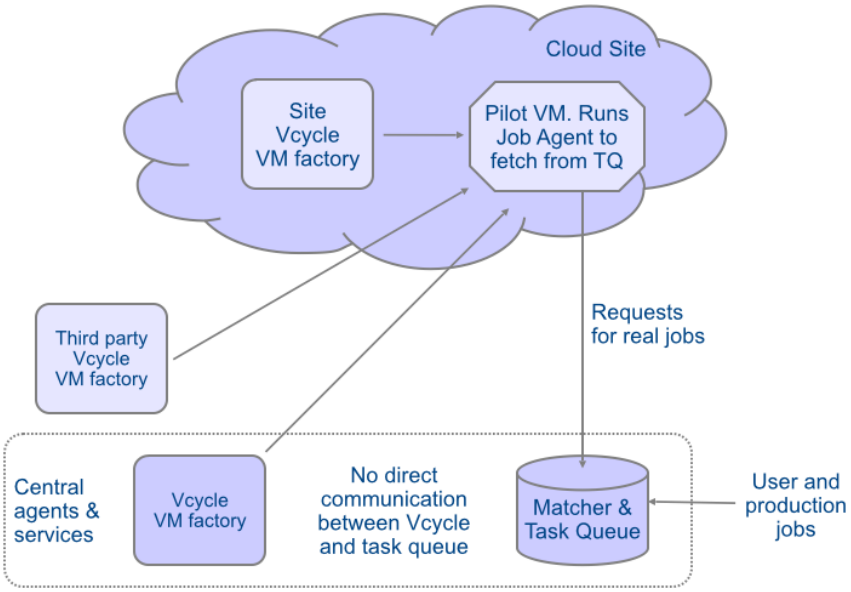

Figure 2. The Vacuum model as implemented by Vcycle

national infrastructure might run a central Vcycle instance to manage cloud resources at sites without requiring them to run HEP-specific services themselves.

As with Vac, Vcycle provides VMs with the Machine/Job Features[10] directories and a writeable machineoutputs directory for log files, heartbeat files and shutdown messages. In Vcycle's case, these directories are provided by HTTPS from an Apache[12] server on the same machine as the Vcycle daemon. The URLs of these directories are declared using the machinefeatures, jobfeatures, and machineoutputs metadata keys avaiable to the VMs.

Vcycle has been designed to be extensible with Python plugins, with OpenStack support provided by a plugin supplied with Vcycle. This plugin uses the OpenStack REST API directly without an intermediate library, and this implementation was the result of a rewrite of Vcycle after first using the Python Nova library. We observed that the REST and Nova Library APIs were of similar complexity and direct use of the REST API removed a dependency. This is largely due to the clean way in which the OpenStack developers have designed their REST API.

Other plugins are planned for integration into the Vcycle distribution, including an OCCI[13] plugin which has been contributed by Luis Villazon Esteban at CERN.

5. user_data templates

To create a virtual machine for an experiment, both Vac and Vcycle start the VM using the boot disk image and then supply the VM with a user_data contextualization file consisting of options and a short shell script. These files can either be loaded from a nominated filesystem location or from an HTTP(S) URL. Typically, the file is supplied by the experiment from a webserver they control, in the form of a template text file. However, any form of user_data file can be used, provided it is compatible with the boot image chosen by the experiment.

Vac and Vcycle apply a series of default substitutions to a user_data template file giving the space name, the vmtype, the hostname of the VM, the Vac or Vcycle version and its hostname.

For example, any instance of the text "##user_data_space##" is replaced by the space name.

In addition, the resource provider can follow instructions from the experiment to define additional custom substitutions, all prefixed with "##user_data_option_" or "##user_data_file_" to be replaced with a string contained in the configuration file or the entire contents of the nominated file respectively. The substitutions "##user_data_proxy_cert##" and "##user_data_proxy_key##" cause the files they reference to be read as an X.509 certificate and key from which a GSI legacy or RFC proxy[14] is produced and inserted into the final user_data file.

This example shows the entire ATLAS configuration used in production on the Vac-based site at the University of Manchester:

```
[vmtype atlasprod]
vm_model = cernvm3
root_image = https://www.gridpp.ac.uk/vac/atlas/cernvm3.iso
root_public_key = /root/.ssh/id_rsa.pub
backoff_seconds = 600
fizzle_seconds = 600
heartbeat_file = vm-heartbeat
heartbeat_seconds = 600
max_wallclock_seconds = 172800
log_machineoutputs = True
accounting_fqan = /atlas/Role=NULL/Capability=NULL
user_data = https://www.gridpp.ac.uk/vac/atlas/user_data
user_data_option_queue = UKI-NORTHGRID-MAN-HEP_VAC
user_data_option_cvmfs_proxy = http://squid-cache.tier2.hep.manchester.ac.uk:3128
user_data_file_hostcert = hostcert.pem
user_data_file_hostkey = hostkey.pem
user_data_option_default_se = bohr3226.tier2.hep.manchester.ac.uk
```

Some of the settings, such as the URL of the Squid caching proxy used by CernVM-FS[15], are provided by the site. The URLs of the root_image and user_data file are provided centrally for all sites.

6. Boot image handling

The root_image setting given to Vac or Vcycle is the filename or URL of a boot image which is supplied to the VM as it starts. When using uCernVM[16] boot images which are around 20MB in size, the image can be conveniently supplied to sites from the same webserver the experiment uses to provide the user_date templates. As the update time of the file is checked each time a VM is created using the HTTP If-Modified-Since header, the experiment can update the image centrally and have it picked up immediately for all new VMs by all Vac and Vcycle instances.

For both implementations the boot images are cached on local disk, with modification time set to that of the remote URL to avoid race conditions due to system clock problems. Vac can pass the cached file directly to libvirt when creating VMs. The Vcycle OpenStack plugin manages uploading new images to OpenStack's image management services.

7. Target shares

Both Vac and Vcycle apply target shares specified by the resource provider in their configuration. Shares are applied at the level of vmtypes within the space. When a VM slot needs to be filled, the target shares are compared with the number of starting or running VMs and a VM is created for the vmtype which is most below its share, taking the backoff algorithm into account. For Vac, the VacQuery UDP protocol is used to gather information about what is happening across

the space. For Vcycle, a complete picture of what is happening is obtained from the IaaS infrastructure such as OpenStack.

Figure 3 shows the target shares mechanism in action on the Manchester Vac site, with varying demand for VMs from ATLAS and LHCb over a one-week period.

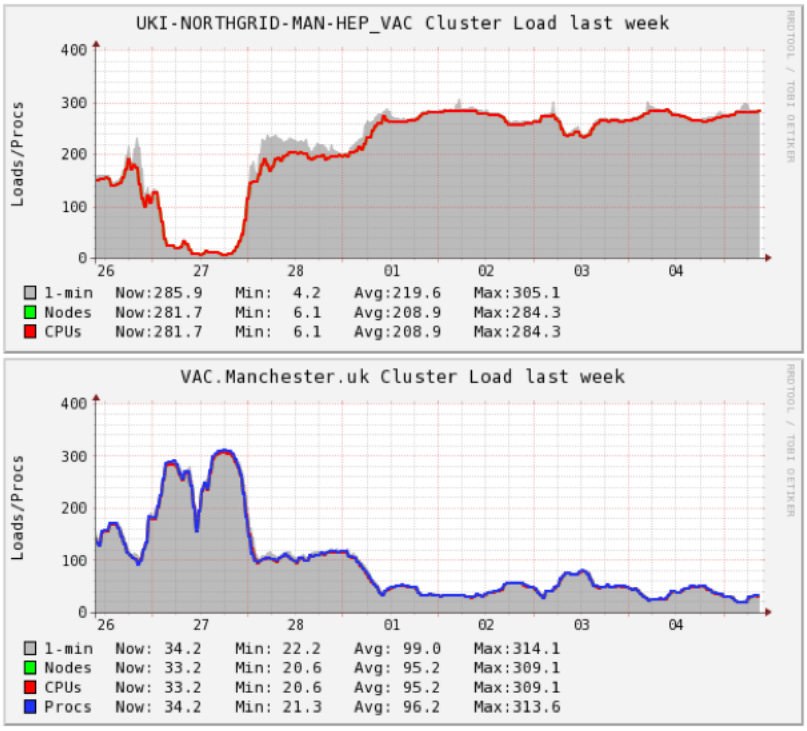

Figure 3. ATLAS (top) and LHCb (bottom) VMs at the Manchester Vac site, showing the target shares mechanism dealing with varying demand. The period around day 28 is with demand from both experiments and the target share ratio of 2:1 being achieved.

8. APEL accounting

As each VM finishes, both Vac and Vcycle record accounting information in the format of APEL SSM[17] messages in the directories apel-outgoing and apel-archive. APEL's ssmsend tool can then be used to send the copy created in apel-outgoing to the central APEL service, with the copy being deleted on successful receipt of the message. At the time of writing, this system has been in production at the Manchester and Oxford Vac sites for several months.

9. Deployment at sites

Vac and Vcycle have been deployed and used to run production workloads at the sites listed in Table 2. The Manchester (320 VMs) and CERN (540 VMs) sites are the largest deployments managed by Vac and Vcycle respectively.

The OpenStack-based capacity at CERN and CC-IN2P3 is managed by a Vcycle instance running on one of the LHCb central "VO Box" machines at CERN. The OpenStack capacity at Imperial College, London is managed by the GridPP Vcycle instance at the University of Manchester.

Table 2. LHCb use of VM-enabled sites

CERN	Vcycle & OpenStack
CC-IN2P3	Vcycle & OpenStack
Imperial College	Vcycle & OpenStack
Birmingham	Vac
Lancaster	Vac
Manchester	Vac
Oxford	Vac
University College	Vac

10. Vacuum model implementations

The Vacuum model has also been implemented by the HTCondor Vacuum system[18], in which HTCondor is used to produce VMs and similar target share and backoff procedures are used. The same Pilot VMs with the same user_data templates can be used for all three implementations.

11. Conclusion

We have presented a description of the Vac and Vcycle virtual machine lifecycle managers, which are providing VMs for ATLAS, CMS, LHCb, and the GridPP DIRAC service, in which production jobs are being run at 8 sites. We have explained that these VMs have been used in production for the past year.

References

[1] A. McNab et al 2014 *J. Phys.: Conf. Ser.* **513** 032065
[2] http://www.openstack.org/
[3] M. Ballintijn et al 2003 "The PROOF Distributed Parallel Analysis Framework based on ROOT" *CHEP 2003 Proceedings* arXiv:physics/0306110
[4] T. Berners-Lee et al, RFC2616 "Hypertext Transfer Protocol - HTTP/1.1" (Internet Engineering Task Force) Section 10
[5] J. Postel, RFC 821 "Simple Mail Transfer Protocol" (Internet Engineering Task Force) Section 4.2
[6] A.McNab et al 2015, "LHCB experience with running jobs in virtual machines", presented at the CHEP 2015 conference
[7] http://libvirt.org/
[8] Puppet from PuppetLabs, http://puppetlabs.com/
[9] T. Bray, RFC7159 "The JavaScript Object Notation (JSON) Data Interchange Format" (Internet Engineering Task Force)
[10] https://twiki.cern.ch/twiki/bin/view/LCG/WMTEGEnvironmentVariables
[11] The Virtio toolkit, http://www.linux-kvm.org/page/Virtio
[12] The Apache Webserver Project, http://httpd.apache.org/
[13] R. Nyren et al, GFD-P-R.183 "The OCCI Core Specification" (Open Grid Forum)
[14] S. Tuecke et al, RFC3820 "Internet X.509 Public Key Infrastructure Proxy Certificate Profile" (Internet Engineering Task Force)
[15] J. Blomer et al. 2012 *J. Phys.: Conf. Ser.* **396** 052013
[16] J. Blomer et al. 2014 *J. Phys.: Conf. Ser.* **513** 032007
[17] APEL Secure Stomp Messenger, https://wiki.egi.eu/wiki/APEL/SSM
[18] A. Lahiff 2015, "Implementation of the vacuum model using HTCondor", presented at the CHEP 2015 conference