

# Vacuum Platform

A. McNab<sup>1</sup>

<sup>1</sup>*University of Manchester*

## Abstract

This technical note describes components of the Vacuum Platform developed by GridPP for managing VMs, including the \$JOBOUTPUTS, VacQuery, and VacUserData interfaces.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Environment</b>	<b>2</b>
<b>3</b>	<b>Machine/Job Features</b>	<b>2</b>
<b>4</b>	<b>\$JOBOUTPUTS</b>	<b>3</b>
4.1	Shutdown Messages . . . . .	3
<b>5</b>	<b>Image URLs</b>	<b>4</b>
<b>6</b>	<b>VacUserData templates</b>	<b>4</b>
<b>7</b>	<b>VacQuery</b>	<b>6</b>
7.1	Factory messages . . . . .	6
7.1.1	factory_query . . . . .	6
7.1.2	factory_status . . . . .	6
7.2	Machine messages . . . . .	8
7.2.1	machines_query . . . . .	8
7.2.2	machine_status . . . . .	8
7.3	Machinetype messages . . . . .	9
7.3.1	machinetypes_query . . . . .	9
7.3.2	machinetype_status . . . . .	9
7.4	VacMon services . . . . .	10
<b>8</b>	<b>APEL</b>	<b>10</b>
<b>9</b>	<b>GOCDB</b>	<b>11</b>
<b>10</b>	<b>Summary</b>	<b>11</b>

## 1 Introduction

This technical note describes components of GridPP’s Vacuum Platform for managing virtual machines (VMs) to run jobs for WLCG and other HEP experiments.

The \$JOBOUTPUTS, VacQuery, and VacUserData interfaces are described, which have been developed for managing the VM lifecycle. These are used by two GridPP software systems, Vac and Vcycle, which can be described as VM lifecycle managers (VMLMs).

This note is written in terms of VMs, but the interfaces have been designed to be generalise to other forms of logical machine in the future, such as Docker containers and unikernels.

The term “resource provider” is used to refer to the entity which is able to take the decision about creating each VM. That is, the decision about whether resources will be provided or not. Typically, this is owner of an OpenStack or other cloud tenancy managed by Vcycle or the manager of Vac VM factories. The term is not used here to refer to higher or lower layers of resource provision in terms of legal owners of services and hardware, funding agencies, operators of the site infrastructure etc.

A location at which VMs can be created managed by one or more VMLMs which are cooperating to achieve a set of target shares is referred to as a “space”. This is equivalent to a Compute Element (CE) at a Grid site, and spaces must be given CE-style DNS names in DNS space available to the resource provider. However, it is not necessary to register the space name in the corresponding DNS zone. For Vac, a space is a set of VM factory machines which are communicating via VacQuery and may be said to be neighbours. For Vcycle, a space corresponds to an OpenStack or similar tenancy or project, with a specified endpoint to contact and identity tokens to use.

Each space is occupied by VMs which are instances of one or more “machinetype” that the VMLM is able to create. Each machinetype corresponds to a specific combination of VM boot image and contextualization.

## 2 Environment

Where possible, an approximation of OpenStack’s environment for VMs, which is derived from EC2, should be provided. Any contextualization user\_data file required and a metadata service should be provided via a “Magic IP” HTTP service at 169.254.169.254 from the point of view of the VM. Monolithic VM images which do not use a user\_data file require a metadata service to be able to discover the URLs of the \$MACHINEFEATURES, \$JOBFEATURES, and \$JOBOUTPUTS locations.

As not all IaaS cloud systems provide metadata services, VMs and VMLMs should also implement the VacUserData substitutions described in section 6. These include substitutions giving the URLs of the \$MACHINEFEATURES, \$JOBFEATURES, and \$JOBOUTPUTS locations.

VMLMs must ensure they support VMs which use Cloud Init contextualization.

VMs must not block access to the address 169.254.169.253, which is used by Amazon for DNS and by Vac for the default route, the local DNS, and for the Machine/Job Features HTTP service.

## 3 Machine/Job Features

The Machine/Job Features (MJF) mechanism described in [1] allows resource providers to communicate information to batch jobs and virtual machines, including the number of processors they are allocated and how long they may run for. The MJF terminology is derived from batch job environments, and job equates to virtual machine when applied to virtualized environments such as the Vacuum Platform.

Resource providers using the Vacuum Platform must make the MJF `$MACHINEFEATURES` and `$JOBFEATURES` locations available over HTTP(S) rather than through a shared filesystem, and should publish the URLs of these locations in OpenStack/EC2 `machinefeatures` and `jobfeatures` metadata tags and using the `VacUserData` substitutions in the `user_data` files supplied to VMs.

The value of `$JOBFEATURES/job_id` should be set to the VM UUID by the VMLM as soon as it is known. For example, with Vac the UUID is chosen by VMLM and its value can be set when the first `$JOBFEATURES` key/values are created. However with Vcycle managing OpenStack, the VM UUID is only available after the VM has been created, and is then recorded by Vcycle in the `$JOBFEATURES` directory it provides.

## 4 `$JOBOUTPUTS`

The `$JOBOUTPUTS` mechanism is an extension to Machine/Job Features by which the URL of a location to which VMs can write status and log files is communicated to the VMs. This value of the `$JOBOUTPUTS` URL should be given in the same way as the `$MACHINEFEATURES` and `$JOBFEATURES` URLs, using a `VacUserData` substitution and an OpenStack/EC2 `joboutputs` metadata key.

Any log file which the VMs wish to make available to resource providers may be written to the `$JOBOUTPUTS` location, for later examination in case of problems. All of these files must have unique names, and are all written to the same level (“directory”) of the URL space on the `$JOBOUTPUTS` HTTP(S) server. This mechanism is also used to provide the `shutdown_message` file described in the next section.

### 4.1 Shutdown Messages

When VMs finish, they should write a `shutdown_message` file to `$JOBOUTPUTS/shutdown_message` containing one line of text without a trailing newline character. This text consists of a three digit shutdown message code in the range 100-999, a space, and then a human-readable description of the message code.

The message code (and not the human-readable description) will be used by the resource provider’s software to determine why the VM finished and whether to create more VMs of this type in the immediate future as slots become available.

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

Table 1: Shutdown codes and messages

The shutdown codes are designed to be extensible by the insertion of intermediate numbers for finer-grained reporting. This is similar to the three digit response codes of internet protocols such as SMTP and HTTP.

## 5 Image URLs

Experiments should provide the HTTPS URL of the image file required to boot their VMs, which VMLMs should use. VMLMs should support both standard CAs and International Grid Trust Federation (IGTF) endorsed CAs when verifying the X.509 certificates used by the relevant HTTPS webserver.

To avoid overloading these web servers, VMLMs must cache images by Last-Modified time, and should use the HTTP If-Modified-Since mechanism when fetching images. If this header is used, then it is acceptable to check the URL for updates each time a VM is created.

Where the VMLM is unable to update the image used to boot the VMs itself, it should attempt to verify that the image being used is current and refuse to create new VMs with an old image. Typically this applies to IaaS cloud systems where users are unable to upload new images, or a manual upload step is required. VMLM authors should consider how resource providers will be made aware of this situation when it arises, but for scalability reasons, the VMLM should not rely on the experiment suffering from VMs failing due to an out of date VM image and then notifying resource providers.

## 6 VacUserData templates

In most cases, a generic image such as CernVM is used which then requires further contextualization as the VM starts using a user\_data file supplied by VMLM. The VMLM must be able to retrieve a template for the user\_data file from an HTTPS URL nominated by the experiment each time a VM is to be created. That is, without any caching. The VMLM must include an appropriate HTTP User-Agent header indicating the VMLM implementation and version when making this request to allow experiments to monitor

which VMLM versions are in use. The VMLM should support both standard CAs and IGTF-endorsed CAs when verifying the X.509 certificates used by the relevant HTTPS webserver.

The VMLM must apply the following pattern based substitutions to the `user_data` template supplied by the experiment. These patterns are all in the form `##user_data.XXX##`.

The following substitutions are performed automatically using data the VMLM holds internally:

```
##user_data_space## Space name
##user_data_machinetype## Name of the machinetype of this VM
##user_data_machine_hostname## Hostname assigned to the VM by the
    VMLM
##user_data_manager_version## A string giving the VMLM version
##user_data_manager_hostname## Hostname of the VMLM
##user_data_manager_machinefeatures_url## $MACHINEFEATURES URL
    (section 3)
##user_data_manager_jobfeatures_url## $JOBFEATURES URL (section 3)
##user_data_manager_joboutputs_url## $JOBOUTPUTS URL (section 4)
```

The VMLM must also provide a mechanism for the resource provider to specify strings or files whose static values will be used in pattern substitutions required by the VM. These patterns take the form `##user_data.option.XXX##` where `XXX` is an arbitrary string consisting of letters, numbers, and underscores.

If the VM requires the address(es) of HTTP proxies to use with CernVM-FS, it must expect this value as the special pattern `##user_data.option.cvmfs.proxy##`. The VM must be able to accept compound CernVM-FS proxy expressions containing semicolon and pipe characters. Typically this will involve placing the substitution pattern in appropriate quotation marks within the `user_data` template.

If the VM requires an X.509 proxy, it must expect that the special pattern `##user_data.option.x509.proxy##` will be replaced by the PEM encoded X.509 certificates and RSA private key which compromise the proxy. VMLMs should provide a mechanism for creating X.509 proxies dynamically for each VM instance from a host or robot certificate owned by the resource provider, with an X.509 proxy lifetime reflecting the maximum VM lifetime.

The VM must not assume that any other grid, HEP middleware, or scripts are running as part of the VMLM and able to provide dynamic values for pattern substitutions. For example, it must not require that resource providers provide proxies with VOMS attributes to the VM. If this is needed, the VM should use the proxy provided to obtain the VOMS credentials itself, using software managed by the experiment within the VM.

## 7 VacQuery

The VacQuery protocol specifies queries and status messages which can be sent over UDP as short JSON 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 machinetypes. 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 Vac-based sites.

VacQuery queries sent to Vac daemons take the form of JSON documents in packets directed to the unused UDP port 995.<sup>1</sup> Responses are sent to the UDP port from which the query was sent. 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.

All dates/times in VacQuery messages are expressed as Unix seconds. That is, the integer number of seconds since 00:00:00 1st Jan 1970.

### 7.1 Factory messages

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

#### 7.1.1 `factory_query`

The `factory_query` message is sent to a factory to request a `factory_status` message in response.

**message\_type** “factory\_query”  
**vac\_version** Name and software version of Vac  
**vacquery\_version** Name and version of the VacQuery protocol  
**space** Vac space name  
**cookie** Freely chosen by the sender

#### 7.1.2 `factory_status`

`factory_status` messages are returned in response to `factory_query` messages directed to a factory. They may also be generated spontaneously and sent to a VacMon service as described in section 7.4.

The format and units of the disk and memory values are aligned with the values returned by the relevant system calls and the `/proc` interface.

---

<sup>1</sup>In Roman numerals, V=5 and M=1000. 995 could be written as VM = 1000 - 5, although this violates conventions invented in modern times.

**message\_type** “factory\_status”

**vac\_version** Name and software version of Vac

**vacquery\_version** Name and version of the VacQuery protocol

**cookie** Matching the value supplied by the recipient

**space** Vac space name

**factory** FQDN of the factory

**time\_sent** Time in Unix seconds

**site** Name of the site registered in the GOCDB, or the Vac space name if the site is not registered

**running\_cpus** Number of processors assigned to running VMs

**running\_machines** Number of running VMs

**running\_hs06** Total HS06 of running VMs

**max\_cpus** Maximum number of (logical) processors available to VMs

**max\_machines** Maximum possible number of VMs

**max\_hs06** Maximum HS06 available to all VMs

**boot\_time** The time when the factory booted up in Unix seconds

**factory\_heartbeat\_time** Time of the last heartbeat created by the VM factory agent in Unix seconds

**responder\_heartbeat\_time** Time of the last heartbeat created by the VacQuery responder service in Unix seconds

**mjf\_heartbeat\_time** Time of the last heartbeat created by the HTTP Machine/Job Features service in Unix seconds

**metadata\_heartbeat\_time** Time of the last heartbeat created by the HTTP Metadata service in Unix seconds

**vac\_disk\_avail\_kb** Free space available in Vac’s workspace, in units of 1024 bytes

**root\_disk\_avail\_kb** Free space available on the root partition, in units of 1024 bytes

**vac\_disk\_avail\_inodes** Free inodes available in Vac’s workspace

**root\_disk\_avail\_inodes** Free inodes available on the root partition

**load\_average** The 15 minute load average on the factory

**kernel\_version** The kernel version of the factory

**os\_issue** A string identifying the operating system (typically the first line of /etc/issue)

**swap\_used\_kb** Swap space in use on the factory, in units of 1024 bytes

**swap\_free\_kb** Free swap space, in units of 1024 bytes

**mem\_used\_kb** Physical memory in use on the factory, in units of 1024 bytes

**mem\_total\_kb** Free physical memory, in units of 1024 bytes



## 7.2 Machine messages

The `machines_query` (plural) and `machine_status` (singular) messages can be used to create views of the VMs running within a Vac space, similar to the views from the `top` command of running processes on a single host.

### 7.2.1 `machines_query`

The `machines_query` message is sent to a factory to request a `machine_status` message for each of its VM slots.

**message\_type** “`machines_query`”  
**vac\_version** Name and software version of Vac  
**vacquery\_version** Name and version of the VacQuery protocol  
**space** Vac space name  
**cookie** Freely chosen by the sender

### 7.2.2 `machine_status`

`machine_status` messages are returned in response to `machines_query` messages directed to a factory.

**message\_type** “`machine_status`”  
**vac\_version** Name and software version of Vac  
**vacquery\_version** Name and version of the VacQuery protocol  
**cookie** Matching the value supplied by the recipient  
**space** Vac space name  
**factory** FQDN of the factory  
**num\_machines** Number of `machine_status` messages to expect from this factory  
**time\_sent** Time in Unix seconds  
**machine** Hostname of the VM slot  
**state** State of the current or most recent VM in this slot, as a string  
**uuid** Lowlevel UUID, as used by `libvirtd`  
**created\_time** Unix time of the VM’s creation  
**started\_time** Unix time the VM entered the running state  
**heartbeat\_time** Unix time when the VM was last observed to be running (this is not the same as any heartbeat generated within the VM)  
**cpu\_seconds** CPU seconds used by the VM  
**cpu\_percentage** Recent CPU percentage use. May be over 100% for multi-processor VMs

**hs06** Total HEPSPEC06 for the processors assigned to this VM  
**machinetype** Name of the machinetype  
**shutdown\_message** Any shutdown message given by the last VM to run in this slot  
**shutdown\_time** Unix time of the shutdown\_message

## 7.3 Machinetype messages

The machinetypes\_query (plural) and machinetype\_status (singular) messages are used by factories to gather information from neighbours within the same Vac space about what they are running, and outcomes of recently started VMs which have finished.

### 7.3.1 machinetypes\_query

The machinetypes\_query message is sent to a factory to request a machinetype\_status message for each of the machinetypes it supports.

**message\_type** “machinetypes\_query”  
**vac\_version** Name and software version of Vac  
**vacquery\_version** Name and version of the VacQuery protocol  
**space** Vac space name  
**cookie** Freely chosen by the sender

### 7.3.2 machinetype\_status

machinetype\_status messages are returned in response to machinetypes\_query messages directed to a factory. They may also be generated spontaneously and sent to a VacMon service as described in section 7.4.

**message\_type** “machinetype\_status”  
**vac\_version** Name and software version of Vac  
**vacquery\_version** Name and version of the VacQuery protocol  
**cookie** Matching the value supplied by the recipient  
**space** Vac space name  
**factory** FQDN of the factory  
**num\_machinetypes** Number of machinetype\_status messages to expect from this factory  
**time\_sent** Time in Unix seconds  
**machinetype** Name of the machinetype  
**running\_hs06** Total HEPSPEC06 of all the processors allocated to running VMs for this machinetype on this factory

**running\_machines** Number of running VMs for this machinetype on this factory

**running\_cpus** Number of CPUs allocated to running VMs for this machinetype on this factory

**num\_before\_fizzle** Number of running VMs which have not yet reached fizzle\_seconds

**shutdown\_message** Shutdown message given by the most recently created VM for this machinetype on this factory which has finished

**shutdown\_time** Unix time of the shutdown\_message

**shutdown\_machine** Name of the VM slot associated with the shutdown\_message

## 7.4 VacMon services

VacMon services receive factory\_status and machinetype\_status messages from Vac daemons on UDP port 8884. These may be used for Ganglia-style monitoring of individual sites or groups of sites. As VacQuery messages are sent as JSON documents, they may be conveniently recorded in data stores such as ElasticSearch.

## 8 APEL

VMLMs 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.

VLMs must include the following in the messages:

**FQAN** VOMS FQAN specified by the experiment when configuring the space

**SubmitHost** Must be of the form [space name] + "/" + [vmlm] + "-" + [VMLM host name], where “vmlm” is a lowercase name for the VMLM software such as “vac”

**LocalJobId** VM UUID

**LocalUserId** VMLM hostname

**Queue** Name of the machinetype

**GlobalUserName** The space name converted to an X.500 DN with DC components. For example, vac01.example.com would become /DC=vac01/DC=example/DC=com

**InfrastructureDescription** Such as APEL-VAC or APEL-VCYCLE, with APEL and then an uppercase name for the VMLM software.

**Processors** The number of virtual CPUs assigned to the VM.

APEL Sync records must also be sent, and these can conveniently be generated by each VMLM instance from an archive of the individual job messages, as the SubmitHost is unique to the VMLM instance in both cases.

In addition to grid-style APEL records generated by the VMLM, an underlying cloud infrastructure may also be instrumented to submit cloud-style APEL usage records to the central APEL service. This is especially likely at sites participating in the EGI Federated Cloud. In this case, resource providers must ensure that double counting is avoided by disabling reporting from the VMLM to the central APEL service.

## 9 GOCDB

Spaces should be registered in the GOCDB entries for the site, using appropriate service types. The service types `uk.ac.gridpp.vac` and `uk.ac.gridpp.vcycle` have been created for Vac and Vcycle spaces.

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

## 10 Summary

This note has described the various interfaces between virtual machines and virtual machine lifecycle managers required by the Vacuum Platform. Three new interfaces (`$JOBOUTPUTS`, `VacQuery`, and `VacUserData`) have been introduced, and requirements set out for the use of Machine/Job Features, APEL, and GOCDB with the platform. Procedures for how experiments providing VMs should present their VM boot images and contextualization are also explained.

## References

- [1] M. Alef et al, HSF-TN-2016-02 “Machine/Job Features Specification” (HEP Software Foundation)