

Vacuum Platform

A. McNab¹

¹*University of Manchester*

Abstract

This technical note describes components of GridPP's Vacuum Platform for managing VMs, including the `$JOBOUTPUTS`, `VacQuery`, and `VacUserData` interfaces.

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 generalised 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 tenancy managed by `Vcycle` or the manager of `Vac` VM factories. Beyond this may be many layers of resource provision in terms of legal owners of services and hardware and even funding agencies.

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 at a Grid site, and spaces must be given 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 tenancy or project, with a specified endpoint to contact and identity tokens to use.

2 Environment

Where possible, VMLMs should provide an approximation of OpenStack’s environment for VMs, which is derived from EC2. 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 standard `VacUserData` substitutions described in section 6. These include substitutions giving the URLs of the `$MACHINEFEATURES`, `$JOBFEATURES`, and `$JOBOUTPUTS` locations.

VMLMs must also support VMs which use Cloud Init contextualization.

3 Machine/Job Features

The Machine/Job Features (MJF) mechanism described in [?] allows resource providers to communicate information to batch jobs and virtual machines, including the number of

38 processors they are allocated and how long they may run for. The MJF terminology is
39 derived from batch job environments, and job equates to virtual machine when applied to
40 virtualized environments such as the Vacuum Platform.

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

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

50 4 `$JOBOUTPUTS`

51 The `$JOBOUTPUTS` mechanism is an extension to Machine/Job Features by which the URL
52 of a location to which VMs can write status and log files is communicated to the VMs. This
53 value of the `$JOBOUTPUTS` URL should be given in the same way as the `$MACHINEFEATURES`
54 and `$JOBFEATURES` URLs.

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

60 4.1 Shutdown Messages

61 When VMs finish, they should write a `shutdown_message` file to
62 `$JOBOUTPUTS/shutdown_message` containing one line of text without a trailing
63 newline character. This text consists of a three digit shutdown message code in the range
64 100-999, a space, and then a human-readable description of the message code.

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

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

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

71 5 Image URLs

72 Experiments should provide the HTTPS URL of the image file required to boot their
73 VMs, which VMLMs should use. VMLMs should support both standard CAs and IGTF
74 endorsed-CAs when verifying the X.509 certificates used by the relevant HTTPS webserver.

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

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

86 6 VacUserData templates

87 In most cases, a generic image such as CernVM is used which then requires further
88 contextualization as the VM starts using a user_data file supplied by VMLM. The VMLM
89 must be able to retrieve a template for the user_data file from an HTTPS URL nominated
90 by the experiment each time a VM is to be created. That is, without any caching. The
91 VMLM must include an appropriate HTTP User-Agent header indicating the VMLM
92 implementation and version when making this request to allow experiments to monitor
93 which VMLMs are in use. The VMLM should support both standard CAs and IGTF
94 endorsed-CAs when verifying the X.509 certificates used by the relevant HTTPS webserver.

95 The VMLM must apply the following pattern based substitutions to the user_data tem-
96 plate supplied by the experiment. These patterns are all in the form **##user_data.XXX##**.

97 The following substitutions are performed automatically using data the VMLM holds

98 internally:

```
99     ##user_data_space## Space name
100     ##user_data_machinetype## Name of the machinetype of this VM
101     ##user_data_machine_hostname## Hostname assigned to the VM by the
102         VMLM
103     ##user_data_manager_version## A string giving the VMLM version
104     ##user_data_manager_hostname## Hostname of the VMLM
105     ##user_data_manager_machinefeatures_url## $MACHINEFEATURES URL
106         (section 3)
107     ##user_data_manager_jobfeatures_url## $JOBFEATURES URL (section 3)
108     ##user_data_manager_joboutputs_url## $JOBOUTPUTS URL (section 4)
```

109 The VMLM must also provide a mechanism for the resource provider which uses the
110 VMLM to specify strings or files which will be used in substitutions required by the VM.
111 These patterns take the form `##user_data_option.XXX##` where XXX is an arbitrary
112 string consisting of letters, numbers, and underscores.

113 If the VM requires an X.509 proxy, it must expect that the special pattern
114 `##user_data_option_x509_proxy##` will be replaced by the PEM encoded X.509 cer-
115 tificates and RSA private key which comprise the proxy. VMLM's should provide a
116 mechanism for creating X.509 proxies dynamically for each VM instance from a host or
117 robot certificate owned by the resource provider, with an X.509 proxy lifetime reflecting
118 the maximum VM lifetime.

119 7 VacQuery

120 The principal use of the VacQuery protocol is to allow Vac factories to gather information
121 from their neighbours about what VMs are running for what machinetypes. This is done using
122 the `machinetypes_query` and `machinetype_status` UDP messages. Factory and machine
123 message pairs are also supported which can be used for automated or manual monitoring
124 of Vac-based sites.

125 VacQuery queries sent to Vac daemons take the form of JSON documents in packets
126 directed to the unused UDP port 995¹. The protocol has been designed to keep JSON
127 message and IP header below the ethernet MTU of 1500 bytes to avoid fragmentation on
128 local networks. Responses are sent to the UDP port from which the query was sent.

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

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

131 7.1 Factory messages

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

136 7.1.1 `factory_query`

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

139 **message_type** “factory_query”
140 **vac_version** Software version of Vac
141 **vacquery_version** Version of the VacQuery protocol
142 **space** Vac space name
143 **cookie** Freely chosen by the sender

144 7.1.2 `factory_status`

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

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

150 **message_type** “factory_status”
151 **vac_version** Software version of Vac
152 **vacquery_version** Version of the VacQuery protocol
153 **cookie** Matching the value supplied by the recipient
154 **space** Vac space name
155 **factory** FQDN of the factory
156 **time_sent** Time in Unix seconds
157 **site** Name of the site registered in the GOCDB, or the Vac space name if the
158 site is not registered
159 **total_cpus** Number of (logical) processors available to VMs
160 **running_cpus** Number of processors assigned to running VMs
161 **running_machines** Number of running VMs
162 **total_machines** Maximum possible number of VMs
163 **total_hs06** Total HS06 available to VMs
164 **boot_time** The time when the factory booted up in Unix seconds

165 **factory_heartbeat_time** Time of the last heartbeat created by the VM
 166 factory agent in Unix seconds

167 **responder_heartbeat_time** Time of the last heartbeat created by the Vac-
 168 Query responder service in Unix seconds

169 **mjf_heartbeat_time** Time of the last heartbeat created by the HTTP Ma-
 170 chine/Job Features service in Unix seconds

171 **metadata_heartbeat_time** Time of the last heartbeat created by the HTTP
 172 Metadata service in Unix seconds

173 **vac_disk_avail_kb** Free space available in Vac’s workspace, in units of 1024
 174 bytes

175 **root_disk_avail_kb** Free space available on the root partition, in units of
 176 1024 bytes

177 **vac_disk_avail_inodes** Free inodes available in Vac’s workspace

178 **root_disk_avail_inodes** Free inodes available on the root partition

179 **load_average** The 15 minute load average on the factory

180 **kernel_version** The kernel version of the factory

181 **os_issue** A string identifying the operating system (typically the first line of
 182 /etc/issue)

183 **swap_used_kb** Swap space in use on the factory, in units of 1024 bytes

184 **swap_free_kb** Free swap space, in units of 1024 bytes

185 **mem_used_kb** Physical memory in use on the factory, in units of 1024 bytes

186 **mem_total_kb** Free physical memory, in units of 1024 bytes

187 7.2 Machine messages

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

191 7.2.1 **machines_query**

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

194 **message_type** “machines_query”

195 **vac_version** Software version of Vac

196 **vacquery_version** Version of the VacQuery protocol

197 **space** Vac space name

198 **cookie** Freely chosen by the sender

199 7.2.2 machine_status

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

202 **message_type** “factory_status”
203 **vac_version** Software version of Vac
204 **vacquery_version** Version of the VacQuery protocol
205 **cookie** Matching the value supplied by the recipient
206 **space** Vac space name
207 **factory** FQDN of the factory
208 **num_machines** Number of machine_status messages to expect from this
209 factory
210 **time_sent** Time in Unix seconds
211 **machine** Hostname of the VM slot
212 **state** State of the current or most recent VM in this slot, as a string
213 **uuid** Lowlevel UUID, as used by libvirt
214 **created_time** Unix time of the VM’s creation
215 **started_time** Unix time the VM entered the running state
216 **heartbeat_time** Unix time when the VM was last observed to be running
217 (this is not the same as any heartbeat generated within the VM)
218 **cpu_seconds** CPU seconds used by the VM
219 **cpu_percentage** Recent CPU percentage use. May be over 100% for multit-
220 processor VMs
221 **hs06** Total HEPSPEC06 for the number of processors assigned to this VM
222 **machinetype** Name of the machinetype
223 **shutdown_message** Any shutdown message given by the last VM to run in
224 this slot
225 **shutdown_time** Unix time of the shutdown_message

226 7.3 Machinetype messages

227 The machinetypes_query (plural) and machinetype_status (singular) messages are used by
228 factories to gather information from neighbours within the same Vac space about what
229 they are running, and outcomes of recently started VMs which have finished.

230 7.3.1 machinetypes_query

231 The machinetypes_query message is sent to a factory to request a machinetype_status
232 message for each of the machinetypes it supports.

233 **message_type** “machinetypes_query”
 234 **vac_version** Software version of Vac
 235 **vacquery_version** Version of the VacQuery protocol
 236 **space** Vac space name
 237 **cookie** Freely chosen by the sender

238 **7.3.2 machinetype_status**

239 machinetype_status messages are returned in response to machinetypes_query messages
 240 directed to a factory. They may also be generated spontaneously and sent to a VacMon
 241 service as described in section 7.4.

242 **message_type** “factory_status”
 243 **vac_version** Software version of Vac
 244 **vacquery_version** Version of the VacQuery protocol
 245 **cookie** Matching the value supplied by the recipient
 246 **space** Vac space name
 247 **factory** FQDN of the factory
 248 **num_machinetypes** Number of machinetype_status messages to expect from
 249 this factory
 250 **time_sent** Time in Unix seconds
 251 **machinetype** Name of the machinetype
 252 **total_hs06** Total HEPSPEC06 of all the processors allocated to running VMs
 253 for this machinetype on this factory
 254 **total_machines** Number of running VMs for this machinetype on this factory
 255 **num_before_fizzle** Number of running VMs which have not yet reached
 256 fizzle_seconds
 257 **shutdown_message** Shutdown message given by the most recently created
 258 VM for this machinetype on this factory which has finished
 259 **shutdown_time** Unix time of the shutdown_message
 260 **shutdown_machine** Name of the VM slot associated with the shut-
 261 down_message

262 **7.4 VacMon services**

263 VacMon services receive factory_status and machinetype_status messages from Vac daemons
 264 on UDP port 8884. These may be used for Ganglia-style monitoring of individual sites or
 265 groups of sites. As VacQuery messages as JSON documents, they may be conveniently
 266 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”.

In this case, VMLMs 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 APEL records generated by the VMLM, an underlying cloud infrastructure may also be instrumented to submit 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 GLUE2

VMLMs should publish status information as JSON documents using the GLUE 2.0 schema, at an HTTPS URL advertised in the GOCDB service entry for the space, in the Grid Information / URL field.

296 **10 Summary**

297 Vacuum platform interfaces have been described ...

298 **References**