

Vacuum Platform

A. McNab¹

¹*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.

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 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.

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 `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 [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

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

74 5 Image URLs

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

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

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 89 an out of date VM image and then notifying resource providers.

90 6 VacUserData templates

91 In most cases, a generic image such as CernVM is used which then requires further
 92 contextualization as the VM starts using a user_data file supplied by VMLM. The VMLM
 93 must be able to retrieve a template for the user_data file from an HTTPS URL nominated
 94 by the experiment each time a VM is to be created. That is, without any caching. The
 95 VMLM must include an appropriate HTTP User-Agent header indicating the VMLM

196 implementation and version when making this request to allow experiments to monitor
197 which VMLM versions are in use. The VMLM should support both standard CAs and
198 IGTF-endorsed CAs when verifying the X.509 certificates used by the relevant HTTPS
199 webserver.

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

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

104 **##user_data_space##** Space name
105 **##user_data_machinetype##** Name of the machinetype of this VM
106 **##user_data_machine_hostname##** Hostname assigned to the VM by the
107 VMLM
108 **##user_data_manager_version##** A string giving the VMLM version
109 **##user_data_manager_hostname##** Hostname of the VMLM
110 **##user_data_manager_machinefeatures_url##** \$MACHINEFEATURES URL
111 (section 3)
112 **##user_data_manager_jobfeatures_url##** \$JOBFEATURES URL (section 3)
113 **##user_data_manager_joboutputs_url##** \$JOBOUTPUTS URL (section 4)

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

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

123 If the VM requires an X.509 proxy, it must expect that the special pattern
124 **##user_data_option_x509_proxy##** will be replaced by the PEM encoded X.509 cer-
125 tificates and RSA private key which compromise the proxy. VMLM's should provide a
126 mechanism for creating X.509 proxies dynamically for each VM instance from a host or
127 robot certificate owned by the resource provider, with an X.509 proxy lifetime reflecting
128 the maximum VM lifetime.

129 The VM must not assume that any other grid, HEP middleware, or scripts are running
130 as part of the VMLM and able to provide dynamic values for pattern substitutions. For
131 example, it must not require that resource providers provide proxies with VOMS attributes
132 to the VM. If this is needed, the VM should use the proxy provided to obtain the VOMS
133 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 995.¹ 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 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.

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

167 **message_type** “factory_status”
168 **vac_version** Name and software version of Vac
169 **vacquery_version** Name and version of the VacQuery protocol
170 **cookie** Matching the value supplied by the recipient
171 **space** Vac space name
172 **factory** FQDN of the factory
173 **time_sent** Time in Unix seconds
174 **site** Name of the site registered in the GOCDB, or the Vac space name if the
175 site is not registered
176 **total_cpus** Number of (logical) processors available to VMs
177 **running_cpus** Number of processors assigned to running VMs
178 **running_machines** Number of running VMs
179 **total_machines** Maximum possible number of VMs
180 **total_hs06** Total HS06 available to VMs
181 **boot_time** The time when the factory booted up in Unix seconds
182 **factory_heartbeat_time** Time of the last heartbeat created by the VM
183 factory agent in Unix seconds
184 **responder_heartbeat_time** Time of the last heartbeat created by the Vac-
185 Query responder service in Unix seconds
186 **mjf_heartbeat_time** Time of the last heartbeat created by the HTTP Ma-
187 chine/Job Features service in Unix seconds
188 **metadata_heartbeat_time** Time of the last heartbeat created by the HTTP
189 Metadata service in Unix seconds
190 **vac_disk_avail_kb** Free space available in Vac’s workspace, in units of 1024
191 bytes
192 **root_disk_avail_kb** Free space available on the root partition, in units of
193 1024 bytes
194 **vac_disk_avail_inodes** Free inodes available in Vac’s workspace
195 **root_disk_avail_inodes** Free inodes available on the root partition
196 **load_average** The 15 minute load average on the factory
197 **kernel_version** The kernel version of the factory
198 **os_issue** A string identifying the operating system (typically the first line of
199 /etc/issue)
200 **swap_used_kb** Swap space in use on the factory, in units of 1024 bytes
201 **swap_free_kb** Free swap space, in units of 1024 bytes
202 **mem_used_kb** Physical memory in use on the factory, in units of 1024 bytes
203 **mem_total_kb** Free physical memory, in units of 1024 bytes

204 7.2 Machine messages

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

208 7.2.1 machines_query

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

211 **message_type** “machines_query”
212 **vac_version** Name and software version of Vac
213 **vacquery_version** Name and version of the VacQuery protocol
214 **space** Vac space name
215 **cookie** Freely chosen by the sender

216 7.2.2 machine_status

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

219 **message_type** “factory_status”
220 **vac_version** Name and software version of Vac
221 **vacquery_version** Name and version of the VacQuery protocol
222 **cookie** Matching the value supplied by the recipient
223 **space** Vac space name
224 **factory** FQDN of the factory
225 **num_machines** Number of machine_status messages to expect from this
226 factory
227 **time_sent** Time in Unix seconds
228 **machine** Hostname of the VM slot
229 **state** State of the current or most recent VM in this slot, as a string
230 **uuid** Lowlevel UUID, as used by libvirtd
231 **created_time** Unix time of the VM’s creation
232 **started_time** Unix time the VM entered the running state
233 **heartbeat_time** Unix time when the VM was last observed to be running
234 (this is not the same as any heartbeat generated within the VM)
235 **cpu_seconds** CPU seconds used by the VM
236 **cpu_percentage** Recent CPU percentage use. May be over 100% for multit-
237 processor VMs

238 **hs06** Total HEPSPEC06 for the processors assigned to this VM
 239 **machinetype** Name of the machinetype
 240 **shutdown_message** Any shutdown message given by the last VM to run in
 241 this slot
 242 **shutdown.time** Unix time of the shutdown_message

243 **7.3 Machinetype messages**

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

247 **7.3.1 machinetypes_query**

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

250 **message_type** “machinetypes_query”
 251 **vac_version** Name and software version of Vac
 252 **vacquery_version** Name and version of the VacQuery protocol
 253 **space** Vac space name
 254 **cookie** Freely chosen by the sender

255 **7.3.2 machinetype_status**

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

259 **message_type** “factory_status”
 260 **vac_version** Name and software version of Vac
 261 **vacquery_version** Name and version of the VacQuery protocol
 262 **cookie** Matching the value supplied by the recipient
 263 **space** Vac space name
 264 **factory** FQDN of the factory
 265 **num_machinetypes** Number of machinetype_status messages to expect from
 266 this factory
 267 **time_sent** Time in Unix seconds
 268 **machinetype** Name of the machinetype
 269 **total_hs06** Total HEPSPEC06 of all the processors allocated to running VMs
 270 for this machinetype on this factory

271 **total_machines** Number of running VMs for this machinetype on this factory
 272 **num_before_fizzle** Number of running VMs which have not yet reached
 273 fizzle_seconds
 274 **shutdown_message** Shutdown message given by the most recently created
 275 VM for this machinetype on this factory which has finished
 276 **shutdown_time** Unix time of the shutdown_message
 277 **shutdown_machine** Name of the VM slot associated with the shut-
 278 down_message

279 7.4 VacMon services

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

284 8 APEL

285 VMLMs should support reporting of usage to the central APEL service with messages
 286 of the type “APEL-individual-job-message”. These are the records used for conventional
 287 grid sites, rather than those developed for cloud resources.

288 VLMs must include the following in the messages:

289 **FQAN** VOMS FQAN specified by the experiment when configuring the space
 290 **SubmitHost** Must be of the form [space name] + "/" + [vmlm] + "-" +
 291 [VMLM host name], where “vmlm” is a lowercase name for the VMLM
 292 software such as “vac”
 293 **LocalJobId** VM UUID
 294 **LocalUserId** VMLM hostname
 295 **Queue** Name of the machinetype
 296 **GlobalUserName** The space name converted to an X.500 DN with
 297 DC components. For example, vac01.example.com would become
 298 /DC=vac01/DC=example/DC=com
 299 **InfrastructureDescription** Such as APEL-VAC or APEL-VCYCLE, with
 300 APEL and then an uppercase name for the VMLM software.
 301 **Processors** The number of virtual CPUs assigned to the VM.

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

305 In addition to grid-style APEL records generated by the VMLM, an underlying cloud
 306 infrastructure may also be instrumented to submit cloud-style APEL usage records to the

307 cental APEL service. This is especially likely at sites participating in the EGI Federated
308 Cloud. In this case, resource providers must ensure that double counting is avoided by
309 disabling reporting from the VMLM to the central APEL service.

310 **9 GOCDB**

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

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

316 **10 GLUE2**

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

320 **11 Summary**

321 Vacuum platform interfaces have been described ...

322 **References**

- 323 [1] M. Alef et al, HSF-TN-2016-02 “Machine/Job Features Specification” (HEP Software
324 Foundation)