vPoller Documentation

Release 0.7.1

Marin Atanasov Nikolov

Dec 16, 2018

Contents

vPoller is a distributed VMware vSphere API Proxy, designed for discovering and polling of vSphere objects.

It uses the VMware vSphere API in order to perform discovery and polling of vSphere objects.

vPoller uses the ZeroMQ messaging library for distributing tasks to workers and load balancing of client requests.

vPoller can be integrated with other systems, which require access to vSphere objects, but do not have native support for it.

Possible scenarios where vPoller could be used is integration with monitoring systems as part of the discovery and polling process in order to provide monitoring of your VMware vSphere environment.

vPoller has been tested with VMware vSphere 5.x and with very limited testing on vSphere 4.x

vPoller is Open Source and licensed under the BSD License.

CHAPTER 1

Contributions

vPoller is hosted on Github. Please contribute by reporting issues, suggesting features or by sending patches using pull requests.

CHAPTER 2

Bugs

Probably. If you experience a bug issue, please report it to the vPoller issue tracker on Github

CHAPTER 3

Getting started

A good place to start with vPoller is to go over the *Terminology* page in order to get familiar with the concepts and terms used in vPoller.

Once ready with that go ahead to the *Installation of vPoller* and *Configuration of vPoller* documentations, which provide all the details about how to install and configure vPoller.

Make sure to also check the *Example usage of vPoller* page and see how to run your first vPoller task requests.

CHAPTER 4

Contents

4.1 Installation of vPoller

This document walks you through the installation of vPoller.

There are a number of ways to install vPoller on your system - you could either install vPoller from source from the Github repo, or install via `pip`.

4.1.1 Requirements

On the list below you can see the dependencies of vPoller:

- Python 2.7.x, 3.2.x or later
- pyVmomi
- vconnector
- pyzmq
- docopt

The C client of vPoller also requires the following packages to be installed in order to build it:

- Python development files (on Debian systems this is usually provided by the `python-dev` package)
- ZeroMQ 4.x Library

4.1.2 Installation with pip

In order to install vPoller using `pip`, simply execute this command:

```
$ pip install vpoller
```

If you would like to install vPoller in a `virtualenv`, then follow these steps instead:

```
$ virtualenv vpoller-venv
$ source vpoller-venv/bin/activate
$ pip install vpoller
```

4.1.3 Installation from source

The `master` branch of vPoller is where main development happens.

In order to install the latest version of vPoller follow these simple steps:

```
$ git clone https://github.com/dnaeon/py-vpoller.git
$ cd py-vpoller
$ sudo python setup.py install
```

If you would like to install vPoller in a `virtualenv` follow these steps instead:

```
$ virtualenv vpoller-venv
$ source vpoller-venv/bin/activate
$ git clone https://github.com/dnaeon/py-vpoller.git
$ cd py-vpoller
$ python setup.py install
```

This should take care of installing all dependencies for you as well.

In order to install one of the stable releases of vPoller please refer to the page of vPoller stable releases.

4.1.4 Installing the C client of vPoller

vPoller comes with two client applications - a Python and a C client.

In order to use the C client of vPoller you need to make sure that you have the ZeroMQ 4.x library installed as the C client is linked against it.

Here is how to install the ZeroMQ 4.x library on your system from source:

```
$ git clone https://github.com/zeromq/zeromq4-x.git
$ cd zeromq4-x
$ ./autogen.sh
$ ./configure
$ make && sudo make install && make clean
$ sudo ldconfig
```

After that building the vPoller C client is as easy as this:

```
$ cd py-vpoller/extra/vpoller-cclient
$ make
```

You should now have the `vpoller-cclient` executable in your current directory built and ready for use.

4.2 Configuration of vPoller

The default configuration file of vPoller resides in a single file and it's default location is `/etc/vpoller/vpoller.conf`.

Below is an example `vpoller.conf` file that you can use:

```
[proxy]
frontend     = tcp://*:10123
backend      = tcp://*:10124
mgmt         = tcp://*:9999

[worker]
db           = /var/lib/vconnector/vconnector.db
proxy        = tcp://localhost:10124
mgmt         = tcp://*:10000
helpers      = vpoller.helpers.zabbix, vpoller.helpers.czabbix
tasks        = vpoller.vsphere.tasks

[cache]
enabled      = True
maxsize      = 0
ttl          = 3600
housekeeping = 480
```

The table below provides information about the config entries used along with a description for each of them.

Section	Option	Description
proxy	frontend	Endpoint to which clients connect and send tasks for processing
proxy	backend	Endpoint to which workers connect and get tasks for processing
proxy	mgmt	Management endpoint, used for management tasks of the `vPoller Proxy`
worker	db	Path to the `vconnector.db` SQLite database file
worker	proxy	Endpoint to which workers connect and get tasks for processing
worker	mgmt	Management endpoint, used for management tasks for the `vPoller Worker`
worker	helpers	Helper modules to be loaded and used for post-processing of any result data
worker	tasks	Task modules to be loaded by the `vPoller Worker`
cache	enabled	If True then `vPoller Worker` will use a cache for the vSphere managed objects
cache	maxsize	Upperbound limit on the entries stored in the cache
cache	ttl	The TTL in seconds after which a cached object is considered as expired
cache	housekeeping	Time in minutes to perform periodic cache housekeeping

4.2.1 Configuring vSphere Agents for the Workers

The `vSphere Agents` are the ones that take care of establishing connections to the vSphere hosts and perform discovery and polling of vSphere objects.

A `vPoller Worker` can have as many `vSphere Agents` as you want, which means that a single `vPoller Worker` can be used to monitor multiple vSphere hosts (ESXi hosts, vCenter servers).

Connection details (username, password, host) about each `vSphere Agent` are stored in a SQLite database and are managed by the vconnector-cli tool.

Note: The example commands below use the `root` account for configuring a vSphere Agent for a vCenter Server.

The `root` account in a vCenter Server by default has full administrative privileges.

If security is a concern you should use an account for your vSphere Agents that has a restricted set of privileges.

First let's initialize the `vConnector` database file:

```
$ sudo vconnector-cli init
```

By default the `vconnector.db` database file resides in `/var/lib/vconnector/vconnector.db`, unless you specify an alternate location from the command-line.

Now, let's add one `vSphere Agent`, which can later be used by our `vPoller Worker`.

This is how to add a new `vSphere Agent` using `vconnector-cli`:

```
$ sudo vconnector-cli -H vc01.example.org -U root -P p4ssw0rd add
```

When you create a new `vSphere Agent` by default it will be disabled, so in order to use that agent in your `vPoller Worker` you should enable it first.

This is how to enable a `vSphere Agent`:

```
$ sudo vconnector-cli -H vc01.example.org enable
```

At any time you can view the currently registered `vSphere Agents` by running the `vconnector-cli get` command, e.g.:

```
$ sudo vconnector-cli get
+------------------+------------+------------+-----------+
| Hostname         | Username   | Password   |   Enabled |
+==================+============+============+===========+
| vc01.example.org | root       | p4ssw0rd   |         1 |
+------------------+------------+------------+-----------+
```

As the `vconnector.db` database contains connection details about your VMware vSphere hosts in order to avoid any leak of sensitive data you would want to secure this file and make it readable only by the user, which runs the `vPoller Worker`.

4.3 vPoller Services

vPoller consists of a number of components, each responsible for a specific task.

This page describes how to manage the `vpoller-proxy` and `vpoller-worker` services.

Please refer to the *Terminology* page for more information about the vPoller components and their purpose.

In a production environment you would want to have these services running as daemons and started at boot-time. At the end of this documentation we will see how to use a process control system such as Supervisord for managing the `vpoller-proxy` and `vpoller-worker` services.

4.3.1 Starting and stopping the vPoller Proxy

In order to start the `vpoller-proxy` service simply execute the command below:

```
$ vpoller-proxy start
```

After you start the `vpoller-proxy` service you should see something similar, which indicates that the `vpoller-proxy` has started successfully and is ready to distribute tasks to the `vPoller Workers`.

```
$ vpoller-proxy start
[2014-09-05 13:26:04,807 - INFO/MainProcess] Starting Proxy Manager
[2014-09-05 13:26:04,808 - INFO/MainProcess] Creating Proxy Manager sockets
[2014-09-05 13:26:04,808 - INFO/MainProcess] Starting Proxy process
[2014-09-05 13:26:04,809 - INFO/MainProcess] Proxy Manager is ready and running
[2014-09-05 13:26:04,810 - INFO/VPollerProxy-1] Proxy process is starting
[2014-09-05 13:26:04,810 - INFO/VPollerProxy-1] Creating Proxy process sockets
[2014-09-05 13:26:04,810 - INFO/VPollerProxy-1] Proxy process is ready and running
```

In order to stop the `vpoller-proxy` service simply hit `Ctrl+C`, which would gracefully shutdown the service.

Another way to stop the `vpoller-proxy` service is to use the management interface and send a shutdown signal to the service.

Here is how to shutdown a `vpoller-proxy` using the management interface:

```
$ vpoller-proxy --endpoint tcp://localhost:9999 stop
```

4.3.2 Starting and stopping the vPoller Worker

In order to start the `vpoller-worker` service simply execute the command below:

```
$ vpoller-worker start
```

After you start the `vpoller-worker` service you should see something similar, which indicates that the `vpoller-worker` has started successfully and is ready to process task requests.

```
[2014-09-05 04:26:38,136 - INFO/MainProcess] Starting Worker Manager
[2014-09-05 04:26:38,138 - INFO/MainProcess] Starting Worker processes
[2014-09-05 04:26:38,138 - INFO/MainProcess] Concurrency: 1 (processes)
[2014-09-05 04:26:38,139 - INFO/MainProcess] Worker Manager is ready and running
[2014-09-05 04:26:38,141 - INFO/VPollerWorker-1] Worker process is starting
[2014-09-05 04:26:38,142 - INFO/VPollerWorker-1] Creating Worker sockets
[2014-09-05 04:26:38,144 - INFO/VPollerWorker-1] Worker process is ready and running
```

By default when you start the `vpoller-worker` service it will create `worker processes` equal to the number of cores available on the target system.

In order to control the concurrency level and how many worker processes will be started use the `--concurrency` option of `vpoller-worker`.

Here is an example command, which will start `vpoller-worker` with 4 worker processes.

```
$ vpoller-worker --concurrency 4 start
[2014-09-05 04:29:56,680 - INFO/MainProcess] Starting Worker Manager
[2014-09-05 04:29:56,682 - INFO/MainProcess] Starting Worker processes
[2014-09-05 04:29:56,682 - INFO/MainProcess] Concurrency: 4 (processes)
[2014-09-05 04:29:56,689 - INFO/VPollerWorker-1] Worker process is starting
[2014-09-05 04:29:56,694 - INFO/VPollerWorker-1] Creating Worker sockets
[2014-09-05 04:29:56,691 - INFO/VPollerWorker-2] Worker process is starting
[2014-09-05 04:29:56,698 - INFO/VPollerWorker-2] Creating Worker sockets
[2014-09-05 04:29:56,693 - INFO/VPollerWorker-3] Worker process is starting
[2014-09-05 04:29:56,700 - INFO/VPollerWorker-3] Creating Worker sockets
[2014-09-05 04:29:56,703 - INFO/VPollerWorker-3] Worker process is ready and running
[2014-09-05 04:29:56,698 - INFO/VPollerWorker-4] Worker process is starting
[2014-09-05 04:29:56,703 - INFO/MainProcess] Worker Manager is ready and running
[2014-09-05 04:29:56,704 - INFO/VPollerWorker-1] Worker process is ready and running
```

(continues on next page)

(continued from previous page)

```
[2014-09-05 04:29:56,706 - INFO/VPollerWorker-4] Creating Worker sockets
[2014-09-05 04:29:56,705 - INFO/VPollerWorker-2] Worker process is ready and running
[2014-09-05 04:29:56,710 - INFO/VPollerWorker-4] Worker process is ready and running
```

In order to stop the `vpoller-worker` service simply hit `Ctrl+C`, which would gracefully shutdown the service.

Another way to stop the `vpoller-worker` service is to use the management interface and send a shutdown signal to the service.

Here is how to shutdown a `vpoller-worker` using the management interface:

```
$ vpoller-worker --endpoint tcp://localhost:10000 stop
```

4.3.3 Using the vPoller Management Interfaces

When you start `vpoller-proxy` and `vpoller-worker` a management endpoint is available for sending management tasks to the services.

At any time you can request status information from your vPoller services by sending a request to the management interface.

This is how you could get status information from your `vpoller-proxy`:

```
$ vpoller-proxy --endpoint tcp://localhost:9999 status
```

And this is how you could get status information from your `vpoller-worker`:

```
$ vpoller-worker --endpoint tcp://localhost:10000 status
```

4.3.4 Managing vPoller Services with Supervisord

When running vPoller in a production environment you would want to have the `vpoller-proxy` and `vpoller-worker` services running as daemons and started at boot-time.

In this section we will see how to use Supervisord for managing the vPoller services.

First, make sure that you have Supervisord installed on your system.

After that create the following config file and place it into your `Supervisord include` directory.

```
[program:vpoller-proxy]
command=/usr/bin/vpoller-proxy start
redirect_stderr=true
stdout_logfile=/var/log/vpoller/vpoller-proxy.log
autostart=true
;user=myusername
stopsignal=INT

[program:vpoller-worker]
command=/usr/bin/vpoller-worker start
redirect_stderr=true
stdout_logfile=/var/log/vpoller/vpoller-worker.log
autostart=true
;user=myusername
stopsignal=INT
```

Now reload `Supervisord` by executing these commands:

```
$ sudo supervisorctl reread
$ sudo supervisorctl reload
```

In order to start the `vpoller-proxy` and `vpoller-worker` services you would use the `supervisorctl` tool.

This is how to start the vPoller services:

```
$ sudo supervisorctl start vpoller-proxy
$ sudo supervisorctl start vpoller-worker
```

And this is how to stop the vPoller services:

```
$ sudo supervisorctl stop vpoller-proxy
$ sudo supervisorctl stop vpoller-worker
```

For more information about what you can do with `Supervisord`, please refer to the official documentation of Supervisord.

4.4 vPoller Helpers

The `vPoller Helpers` were implemented in order to provide an easy way for connecting your applications to vPoller.

A result messages returned by the `vpoller-worker` is always in JSON format. This could be okay for most applications, which require to process a result message, but in some cases you might want to receive the result in different formats and feed the data into your application.

Using the `vPoller Helpers` you are able to convert the result message to a format that your application or system understands.

The table below summarizes the currently existing and supported `vPoller Helpers` along with a short description:

vPoller Helper	Description
vpoller.helpers.zabbix	Helper which returns result in Zabbix-friendly format
vpoller.helpers.czabbix	vPoller Zabbix helper for C clients
vpoller.helpers.csvhelper	Helper which returns result in CSV format
vpoller.helpers.cclient	Helper for vPoller C clients

The `vPoller Helpers` are simply Python modules and are loaded by the `vPoller Workers` upon startup.

4.4.1 Enabling helpers

In order to enable helpers in your `vPoller Workers` you need to specify in the `vpoller.conf` file the helper modules, which you wish to be loaded and available to clients.

Here is a sample `vpoller.conf` file which includes the `helpers` configuration option for loading the `zabbix` helper module in your `vPoller Worker`:

```
[proxy]
frontend = tcp://*:10123
backend  = tcp://*:10124
mgmt     = tcp://*:9999
```

(continues on next page)

(continued from previous page)

```

[worker]
db        = /var/lib/vconnector/vconnector.db
proxy     = tcp://localhost:10124
mgmt      = tcp://*:10000
helpers   = vpoller.helpers.zabbix
```

Multiple vPoller helpers can be loaded as well by separating them using a comma.

Here's an example `vpoller.conf` file which loads multiple helpers in your `vPoller Worker`:

```
[proxy]
frontend = tcp://*:10123
backend  = tcp://*:10124
mgmt     = tcp://*:9999

[worker]
db        = /var/lib/vconnector/vconnector.db
proxy     = tcp://localhost:10124
mgmt      = tcp://*:10000
helpers   = vpoller.helpers.zabbix,vpoller.helpers.czabbix
```

4.4.2 vPoller Zabbix Helper

One of the `vPoller Helpers` is the Zabbix vPoller Helper module, which can translate a result message to Zabbix LLD format and return values ready to be used in Zabbix items as well.

Here is an example of using the `Zabbix vPoller Helper`, which will convert a result message to Zabbix-friendly format:

```
$ vpoller-client --method datastore.discover --vsphere-host vc01.example.org \
             --helper vpoller.helpers.zabbix
```

The `*.discover` methods of vPoller when used with the Zabbix helper, would return data ready in Zabbix LLD format.

When using the `*.get` methods of vPoller with the Zabbix helper, the result would be a single property value, making it suitable for use in Zabbix items.

This is how to retrieve a property of a `Datastore` object using the Zabbix helper:

```
$ vpoller-client --method vm.get --vsphere-host vc01.example.org \
             --name vm01.example.org --properties runtime.powerState \
             --helper vpoller.helpers.zabbix
```

4.4.3 vPoller CSV Helper

Another vPoller helper is the `vPoller CSV helper` which translates a result message in CSV format.

Here is an example how to get all your Virtual Machines and their `runtime.powerState` property in CSV format:

```
$ vpoller-client --method vm.discover --vsphere-host vc01.example.org \
             --properties runtime.powerState \
             --helper vpoller.helpers.csvhelper
```

And here is a sample result from the above command:

```
name,runtime.powerState
vpoller-vm-1,poweredOn
vpoller-vm-2,poweredOn
freebsd-vm-1,poweredOn
zabbix-vm-1.04-dev,poweredOn
```

Here is one post that you can read which makes use of the `vPoller CSV Helper` in order to export data from a vSphere environment and plot some nice graphs from it.

- Exporting Data From a VMware vSphere Environment For Fun And Profit

4.5 Example usage of vPoller

This page provides some examples how vPoller can be used to perform various operations like discovery and polling of VMware vSphere objects.

Please also refer to the *Supported methods by vPoller* documentation for the full list of supported vPoller methods you could use.

The property names which we use in these examples can be found in the official VMware vSphere API documentation.

Each vSphere managed object has specific properties, which are documented in the official documentation.

The examples here serve for demonstration purpose only and do not provide all the properties you could use and get from vSphere objects, so make sure to refer to the official vSphere documentation when looking for a specific property name.

There are also a number of posts about how vPoller is being used for various purposes, which you could also read at the following links:

- VMware vSphere CLI tips & tricks with vPoller
- VMware monitoring with Zabbix, Python & vPoller
- Exporting Data From a VMware vSphere Environment For Fun And Profit

4.5.1 Getting vSphere "about" info

Using the `about` vPoller method you can retrieve information about your vSphere host such as API version, vendor, build number, etc.

Here is an example of using the vPoller `about` method:

```
$ vpoller-client --method about --vsphere-host vc01.example.org \
             --method about --properties version,fullName,apiVersion,vendor
```

4.5.2 Datacenter examples

Here is how to discover all `Datacenter` objects from your vSphere environment:

```
$ vpoller-client --method datacenter.discover --vsphere-host vc01.example.org
```

An example command that would get the *summary.overallStatus* property of a specific `Datacenter`:

```
$ vpoller-client --method datacenter.get --vsphere-host vc01.example.org \
             --name datacenter01 --properties name,overallStatus
```

4.5.3 ClusterComputeResource examples

A `ClusterComputeResource` managed object is what you are used to refer to simply as `cluster` in vSphere. The examples commands below show how to discover and get properties for your vSphere clusters.

An example command to discover all `ClusterComputeResource` managed objects from your vSphere environment:

```
$ vpoller-client --method cluster.discover --vsphere-host vc01.example.org
```

And here is how to get the `overallStatus` property for a specific `ClusterComputeResource` managed object:

```
$ vpoller-client --method cluster.get --vsphere-host vc01.example.org \
             --name cluster01 --properties name,overallStatus
```

4.5.4 HostSystem examples

`HostSystem` managed objects in vSphere are your ESXi hosts.

Here is an example how to discover all your ESXi hosts from your vSphere environment:

```
$ vpoller-client --method host.discover --vsphere-host vc01.example.org
```

And here is an example command to get the `runtime.powerState` property for a specific `HostSystem` object:

```
$ vpoller-client --method host.get --vsphere-host vc01.example.org \
             --name esxi01.example.org --properties runtime.powerState
```

An example command to get all Virtual Machines registered on a specific ESXi host:

```
$ vpoller-client --method host.vm.get --vsphere-host vc01.example.org \
             --name esxi01.example.org
```

Here is how you can get all datastores used by a specific ESXi host:

```
$ vpoller-client --method host.datastore.get --vsphere-host vc01.example.org \
             --name esxi01.example.org
```

4.5.5 VirtualMachine examples

An example command to discover all `VirtualMachine` managed objects from your vSphere environment:

```
$ vpoller-client --method vm.discover --vsphere-host vc01.example.org
```

Another example showing how to get the `runtime.powerState` property of a Virtual Machine:

```
$ vpoller-client --method vm.get --vsphere-host vc01.example.org \
             --name vm01.example.org --properties runtime.powerState
```

This is how you could discover all disks in a Virtual Machine. Note, that this method requires that you have VMware Tools installed and running on the target Virtual Machine:

```
$ vpoller-client --method vm.disk.discover --vsphere-host vc01.example.org \
             --name vm01.example.org
```

If you want to get information about a disk in a Virtual Machine you could use the `vm.disk.get` vPoller method. This is how to get the `freeSpace` property for a Virtual Machine disk:

```
$ vpoller-client --method vm.disk.get --vsphere-host vc01.example.org \
             --name vm01.example.org --properties freeSpace --key /var
```

In order to find out the host on which a specific Virtual Machine is running on you could use the `vm.host.get` vPoller method:

```
$ vpoller-client --method vm.host.get --vsphere-host vc01.example.org \
             --name vm01.example.org
```

The example below shows how to retrieve information about the network that a Virtual Machine is using along with information about it's IP address and MAC address:

```
$ vpoller-client --method vm.guest.net.get --vsphere-host vc01.example.org \
             --name vm01.example.org --properties network,ipAddress,macAddress
```

If you want to see which datastores your Virtual Machine is using you can use the `vm.datastore.get` vPoller method, e.g.:

```
$ vpoller-client --method vm.datastore.get --vsphere-host vc01.example.org \
             --name vm01.example.org
```

Using the `vm.process.get` vPoller method we can get a list of all processes running in a Virtual Machine. Note, that we need to supply a username and password when using the `vm.process.get` method, which are used for authentication in the guest system.

```
$ vpoller-client --method vm.process.get --vsphere-host vc01.example.org \
             --name vm01.example.org --properties name,owner,pid,cmdLine \
             --guest-username root --guest-password p4ssw0rd
```

Note: The above example command uses the `root` user for authentication in the guest system. It is recommended that you use a user with restricted privileges when using the `vm.process.get` vPoller method if security is a concern.

4.5.6 Datastore examples

Here is an example command which will discover all `Datastore` managed objects from your vSphere environment:

```
$ vpoller-client --method datastore.discover --vsphere-host vc01.example.org
```

This example command below would return the `summary.capacity` property for a specific `Datastore` object.

```
$ vpoller-client --method datastore.get --vsphere-host vc01.example.org \
             -name ds:///vmfs/volumes/5190e2a7-d2b7c58e-b1e2-90b11c29079d/ \
             --properties summary.capacity
```

This example command will give you all hosts, which are using a specific `Datastore`.

```
$ vpoller-client --method datastore.host.get --vsphere-host vc01.example.org \
             --name ds:///vmfs/volumes/5190e2a7-d2b7c58e-b1e2-90b11c29079d/
```

4.5.7 Viewing established Sessions

vPoller can also be used for viewing the established sessions to your vSphere hosts.

Note: Viewing vSphere sessions by unauthorized parties may be considered as a security hole, as it may provide an attacker with information such as Session IDs, which could be used for spoofing a user's session.

If security is a concern make sure that your `vSphere Agents` are configured to use an account with restricted set of privileges, which cannot view the established vSphere sessions.

Here is an example command that will return the established sessions for your vSphere host:

```
$ vpoller-client --method session.get --vsphere-host vc01.example.org
```

4.5.8 Getting vSphere Events

With vPoller you can also retrieve vSphere events.

This is how you can retrieve the last registered event from your vSphere host:

```
$ vpoller-client --method event.latest --vsphere-host vc01.example.org
```

4.5.9 Getting vSphere Alarms

Using the `*.alarm.get` vPoller methods we can retrieve the triggered vSphere alarms on a `Datacenter`, `ClusterComputeResource`, `HostSystem`, `VirtualMachine` and `Datastore` level.

Here is how you could retrieve all triggered alarms for a `Datacenter`.

```
$ vpoller-client --method datacenter.alarm.get --vsphere-host vc01.example.org \
             --name MyDatacenter
```

An here is an example result from the above command, showing the triggered alarms for our Datacenter.

```
{
  "success": 0,
  "result": [
    {
      "overallStatus": "red",
      "time": "2015-02-13 09:16:50.916096+00:00",
      "key": "alarm-4.host-30",
      "entity": "esxi01.example.org",
      "acknowledged": false,
      "acknowledgedByUser": null,
      "info": "Host memory usage"
    }
  ],
```

(continues on next page)

(continued from previous page)

```
  "msg": "Successfully retrieved alarms"
}
```

4.5.10 Performance metrics

Note: If you are running a vSphere 6.0 environment and experience issues with real-time performance metrics, make sure to check [KB-2119264](http://kb.vmware.com/kb/2119264) for more details.

Using vPoller you can retrieve various performance metrics from your VMware vSphere environment.

In the following examples we will see how we can use vPoller in order to discover the supported performance metrics in our vSphere environment and also how to retrieve real-time and historical statistics from different performance providers - ESXi hosts, Virtual Machines, Datastores, Clusters, etc.

For more information about the performance metrics in a VMware vSphere environment, please make sure to check the VMware vSphere API documentation and especially the PerformanceManager documentation where you can find information about the supported performance counters, the existing counter groups, description of each counter, etc.

First, let's see how to obtain all performance counters that are supported in our vSphere environment. Using the `perf.metric.info` vPoller method we can retrieve a list of all supported performance counters from our vSphere environment.

```
$ vpoller-client --method perf.metric.info --vsphere-host vc01.example.org
```

The result of the above command should contain all performance metrics which are supported on the VMware vSphere host `vc01.example.org`.

Below is an example of a single counter as returned from the `perf.metric.info` vPoller method.

```
{
   "perDeviceLevel": 3,
   "level": 1,
   "key": 143,
   "nameInfo": {
     "label": "Usage",
     "key": "usage",
     "summary": "Network utilization (combined transmit-rates and receive-rates)␣
↪during the interval"
   },
   "groupInfo": {
     "label": "Network",
     "key": "net",
     "summary": "Network"
   },
   "unitInfo": {
     "label": "KBps",
     "key": "kiloBytesPerSecond",
     "summary": "Kilobytes per second"
   },
   "statsType": "rate",
   "rollupType": "average"
 }
```

You can find a sample file with all performance metrics as discovered on a VMware vSphere host in the perf-metric-info.json example file.

In order to obtain information about the supported performance metrics for a specific performance provider (e.g. ESXi host, Virtual Machine, Datastore, etc.) you can use the respective `*.perf.metric.info` vPoller methods, e.g. `vm.perf.metric.info`, `host.perf.metric.info`, etc.

The following example shows how to get the available performance metrics for a Virtual Machine:

```
$ vpoller-client --method vm.perf.metric.info --vsphere-host vc01.example.org \
             --name vm01.example.org
```

You can see an example result of using the `vm.perf.metric.info` method in the vm-perf-metric-info.json example file, which shows the available performance metrics for a specific Virtual Machine.

In the vm-perf-metric-info.json example file you will see that each discovered performance metric has a `counterId` and `instance` attribute, e.g.:

```
{
    "counterId": "net.usage.kiloBytesPerSecond.average",
    "instance": "vmnic0"
}
```

The `counterId` part from the above example counter is comprised of the `<groupInfo.key>.<nameInfo.key>.<unitInfo.key>.<rollupType>` attributes from our counter as discovered by the `perf.metric.info` vPoller method.

We can also request specific counters only when using the `*.perf.metric.info` methods. For example if we are only interested in the `net.usage.kiloBytesPerSecond.average` counter, then we would execute this command instead, which would return all counters and their instances for our performance provider:

```
$ vpoller-client --method vm.perf.metric.info --vsphere-host vc01.example.org \
             --name vm01.example.org --counter net.usage.kiloBytesPerSecond.average
```

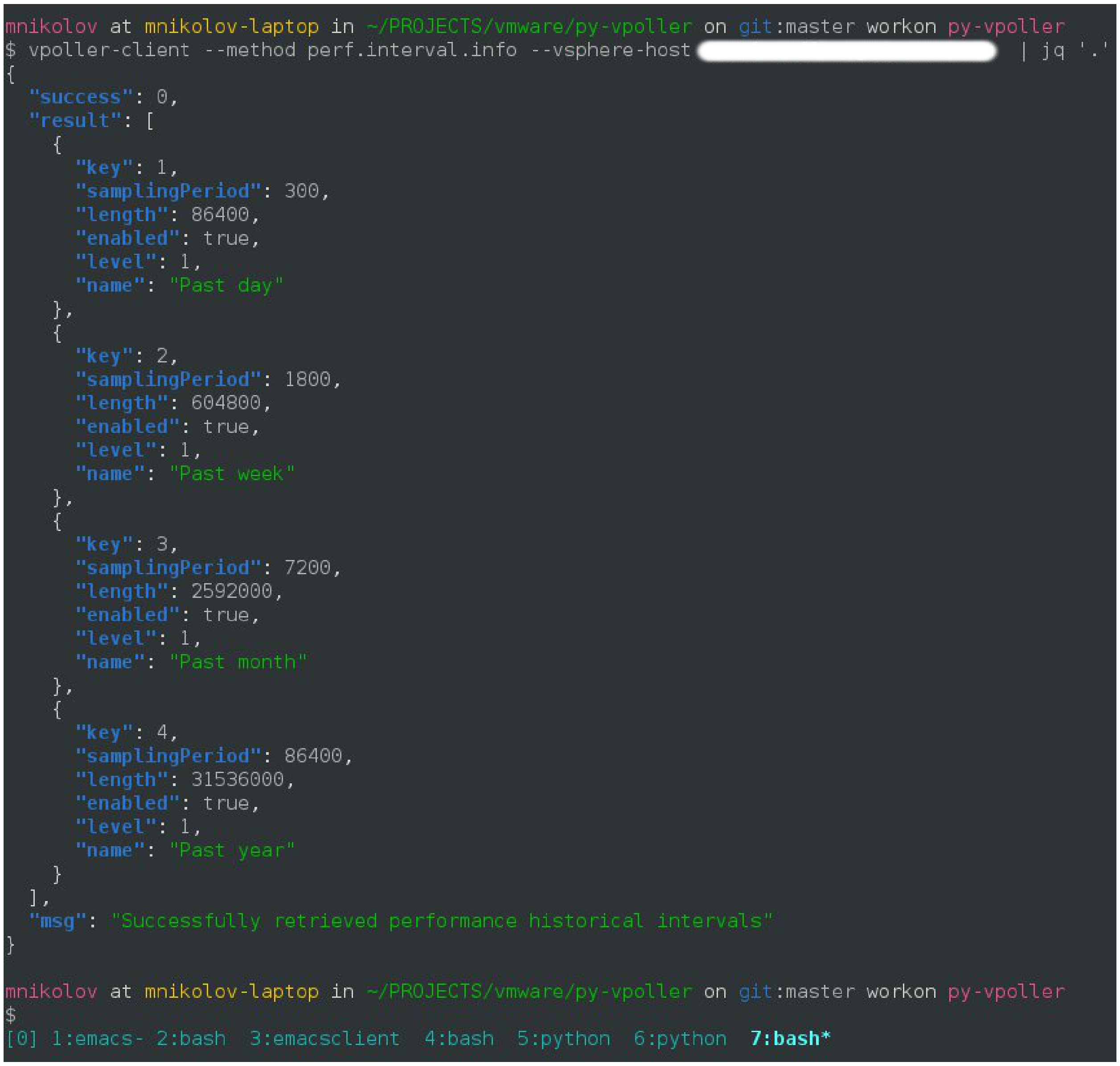
And here's an example result after executing the above command.

```
{
  "result": [
    {
      "counterId": "net.usage.kiloBytesPerSecond.average",
      "instance": "4000"
    },
    {
      "counterId": "net.usage.kiloBytesPerSecond.average",
      "instance": "vmnic0"
    },
    {
      "counterId": "net.usage.kiloBytesPerSecond.average",
      "instance": "vmnic1"
    },
    {
      "counterId": "net.usage.kiloBytesPerSecond.average",
     "instance": ""
    }
  ],
  "msg": "Successfully retrieved performance metrics",
  "success": 0
}
```

We can also get the existing historical performance intervals by using the `perf.interval.info` vPoller method, e.g.:

```
$ vpoller-client --method perf.interval.info --vsphere-host vc01.example.org
```

On the screenshot below you can see an example of retrieving the historical performance intervals on a vSphere host.



```
mnikolov at mnikolov-laptop in ~/PROJECTS/vmware/py-vpoller on git:master workon py-vpoller
$ vpoller-client --method perf.interval.info --vsphere-host [redacted] | jq '.'
{
  "success": 0,
  "result": [
    {
      "key": 1,
      "samplingPeriod": 300,
      "length": 86400,
      "enabled": true,
      "level": 1,
      "name": "Past day"
    },
    {
      "key": 2,
      "samplingPeriod": 1800,
      "length": 604800,
      "enabled": true,
      "level": 1,
      "name": "Past week"
    },
    {
      "key": 3,
      "samplingPeriod": 7200,
      "length": 2592000,
      "enabled": true,
      "level": 1,
      "name": "Past month"
    },
    {
      "key": 4,
      "samplingPeriod": 86400,
      "length": 31536000,
      "enabled": true,
      "level": 1,
      "name": "Past year"
    }
  ],
  "msg": "Successfully retrieved performance historical intervals"
}

mnikolov at mnikolov-laptop in ~/PROJECTS/vmware/py-vpoller on git:master workon py-vpoller
$
[0] 1:emacs- 2:bash  3:emacsclient  4:bash  5:python  6:python  7:bash*
```

The historical performance intervals are used when we need to retrieve historical metrics from performance providers.

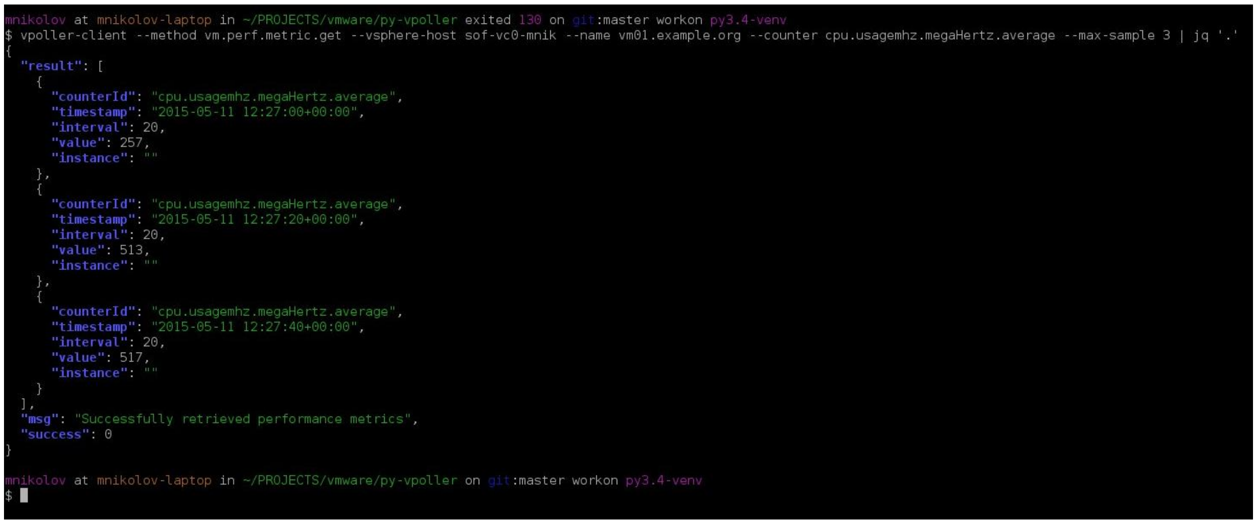
Now, that we know how to get the available performance metrics for our performance providers, let's now see how to retrieve the actual performance counters for them.

In the following example we will see how to get the CPU usage in MHz for a specific Virtual Machine.

And here is how we would get three samples of the `CPU usage in MHz` performance metric.

```
$ vpoller-client --method vm.perf.metric.get --vsphere-host vc01.example.org \
              --name vm01.example.org --max-sample 3 --counter cpu.usagemhz.megaHertz.
↪average
```

Here is an example result of the above command.



```
mnikolov at mnikolov-laptop in ~/PROJECTS/vmware/py-vpoller exited 130 on git:master workon py3.4-venv
$ vpoller-client --method vm.perf.metric.get --vsphere-host sof-vc0-mnik --name vm01.example.org --counter cpu.usagemhz.megaHertz.average --max-sample 3 | jq '.'
{
  "result": [
    {
      "counterId": "cpu.usagemhz.megaHertz.average",
      "timestamp": "2015-05-11 12:27:00+00:00",
      "interval": 20,
      "value": 257,
      "instance": ""
    },
    {
      "counterId": "cpu.usagemhz.megaHertz.average",
      "timestamp": "2015-05-11 12:27:20+00:00",
      "interval": 20,
      "value": 513,
      "instance": ""
    },
    {
      "counterId": "cpu.usagemhz.megaHertz.average",
      "timestamp": "2015-05-11 12:27:40+00:00",
      "interval": 20,
      "value": 517,
      "instance": ""
    }
  ],
  "msg": "Successfully retrieved performance metrics",
  "success": 0
}

mnikolov at mnikolov-laptop in ~/PROJECTS/vmware/py-vpoller on git:master workon py3.4-venv
$ █
```

We can also retrieve the performance metrics for an instance, e.g. get the CPU usage per core.

If we want to retrieve the performance metrics for a specific instance we would execute a similar command instead:

```
$ vpoller-client --method vm.perf.metric.get --vsphere-host vc01.example.org \
             --name vm01.example.org --counter cpu.usagemhz.megaHertz.average --max-
↪sample 3 --instance 0
```

We could also retrieve some interesting statistics from our Datacenters and Clusters as well.

The `vim.Datacenter` and `vim.ClusterComputeResource` managed entities support historical statistics only, so in order to retrieve any performance metrics from them we should specify a valid historical performance interval.

In the following examples we will see how to retrieve performance counter `vmop.numPoweron.number.latest`, which would give us the number of the Virtual Machine power on operations for the past day.

```
$ vpoller-client --method datacenter.perf.metric.get --vsphere-host vc01.example.org \
             --name MyDatacenter --counter vmop.numPoweron.number.latest --perf-
↪interval "Past day"
```

Another example showing how to get performance counter `mem.consumed.kiloBytes.average`, which returns the amount of host physical memory consumed by a virtual machine, host, or cluster.

```
$ vpoller-client --method host.perf.metric.get --vsphere-host vc01.example.org \
            --name esxi01.example.org --counter mem.consumed.kiloBytes.average
```

As our last examples we will see how to retrieve various performance metrics for `vim.Datastore` managed entities.

Note: Some of the `vim.Datastore` performance metrics are retrieved by using the `datastore.perf.metric.get` vPoller method, while others are available via the `host.perf.metric.get`, where a datastore metric is retrieved by using the Datastore instance.

A `vim.Datastore` performance provider by itself provides only historical performance statistics.

Most of the real-time statistics (e.g. `datastoreIops`) are accessed via a `vim.HostSystem` performance provider.

The example below shows how to retrieve the `datastoreIops` for a specific datastore.

First we will discover all instances of performance counter `datastore.datastoreIops.number.average` for our example ESXi host `esxi01.example.org`.

```
$ vpoller-client --method host.perf.metric.info --vsphere-host vc01.example.org \
      --name esx01.example.org --counter datastore.datastoreIops.number.average
```

Example result from the above command is shown below, which contains all instances of counter `datastore.datastoreIops.number.average`.

```
 {
  "success": 0,
  "result": [
    {
      "counterId": "datastore.datastoreIops.number.average",
      "instance": "5481d059-dbd6de3d-2215-d8d385bf2110"
    },
    {
      "counterId": "datastore.datastoreIops.number.average",
      "instance": "5485af07-7326ddc0-6afc-d8d385bf2110"
    },
    {
      "counterId": "datastore.datastoreIops.number.average",
      "instance": "5485af4f-4dbf72e3-4980-d8d385bf2110"
    }
  ],
  "msg": "Successfully retrieved performance metrics"
}
```

If we are interested in finding out the Datastore name for the `5481d059-dbd6de3d-2215-d8d385bf2110` instance from the above example result we could use the `datastore.get` vPoller method to do so. For example:

```
$ vpoller-client --method datastore.get --vsphere-host vc01.example.org \
            --name ds:///vmfs/volumes/5481d059-dbd6de3d-2215-d8d385bf2110/ --
↪properties name
```

And the result from the above command would give us the Datastore name.

```
{
 "success": 0,
 "result": [
   {
     "name": "datastore1",
     "info.url": "ds:///vmfs/volumes/5481d059-dbd6de3d-2215-d8d385bf2110/"
   }
 ],
 "msg": "Successfully retrieved object properties"
}
```

Now, let's get back to the `datastoreIops` metric and retrieve it.

```
$ vpoller-client --method host.perf.metric.get --vsphere-host vc01.example.org \
            --name esxi01.example.org --counter datastore.datastoreIops.number.
↪average --instance 5481d059-dbd6de3d-2215-d8d385bf2110
```

And here is an example result from the above command:

```
{
  "success": 0,
```

(continues on next page)

(continued from previous page)

```
  "result": [
    {
      "instance": "5481d059-dbd6de3d-2215-d8d385bf2110",
      "value": 84,
      "interval": 20,
      "counterId": "datastore.datastoreIops.number.average",
      "timestamp": "2015-02-10 15:48:40+00:00"
    }
  ],
  "msg": "Successfully retrieved performance metrics"
}
```

4.6 Using the API

In this document we will see some examples on how to use the vPoller API.

You can use these examples for connecting your project to vPoller and send task requests for processing.

4.6.1 Sending task requests for processing

Connecting your Python project to vPoller is easy.

If you only need to be able to talk to vPoller and send task requests then using the `VPollerClient` class is the way to go.

The `VPollerClient` sends task requests to the `vPoller Proxy`, which distributes the task requests to the `vPoller Workers`.

Here is how you can send task requests from your Python project to vPoller for processing.

```
>>> from vpoller.client import VPollerClient
>>> msg = {'method': 'vm.discover', 'hostname': 'vc01.example.org'}
>>> client = VPollerClient(endpoint='tcp://localhost:10123')
>>> result = client.run(msg)
>>> print result
```

The above example code will send a task request for discovering all Virtual Machine managed objects from the `vc01.example.org` vSphere host.

And here is what the above example code does:

1. Imports the `VPollerClient` class from the `vpoller.client` module
2. Creates a message that will be sent to the `vPoller Proxy` endpoint. The message contains information such as the `method` to be processed, the vSphere hostname, and any additional details required for processing the task request.
3. We instantiate a `VPollerClient` object and set the endpoint to which the client will connect and send the task request for processing.
4. Using the `run()` method of a `VPollerClient` instance we send the task request over the wire and wait for response.

The `VPollerClient` class comes with builtin mechanism for automatic retry if it doesn't receive a response after some period of time.

In order to control the `retry` and `timeout` settings of a `VPollerClient` object you can instantiate a client object this way:

```
>>> client = VPollerClient(
...     endpoint='tcp://localhost:10123',
...     retries=1,
...     timeout=1000
... )
```

Note, that the `timeout` argument used above is in milliseconds.

Here is another example which would get the `runtime.powerState` property for a specific Virtual Machine:

```
>>> import json
>>> from vpoller.client import VPollerClient
>>> msg = {
...     'method': 'vm.get',
...     'hostname': 'vc01.example.org',
...     'name': 'vm01.example.org',
...     'properties': ['name', 'runtime.powerState']
... }
>>> client = VPollerClient(endpoint='tcp://localhost:10123')
>>> result = client.run(msg)
>>> print json.dumps(result, indent=4)
{
    "msg": "Successfully retrieved object properties",
    "result": [
        {
            "runtime.powerState": "poweredOn",
            "name": "vm01.example.org"
        }
    ],
    "success": 0
}
```

As you can see we have successfully retrieved the `runtime.powerState` property for our Virtual Machine, which shows that our Virtual Machine is powered on.

For a full list of supported vPoller methods which you can use, please refer to the *Supported methods by vPoller* documentation.

You are also advised to check the vpoller.agent module, which is pretty well documented and provides information about each vPoller method and the expected message request in order to begin processing the task.

4.6.2 Executing vPoller tasks locally

Using the `VPollerClient` class as we've seen in the previous section of this document sends task requests to the `vPoller Proxy`, which distributes the tasks to any connected `vPoller Worker`.

This was a remote operation, where a client simply sends a task request and waits for a response.

You could also use vPoller in order to execute tasks locally, which means that no task is send over the wire and all the hard work is done on the local system.

Here is an example of interfacing with the `vSphere Agents`, which provides us with an interface to execute vPoller tasks locally.

The example below is equivalent to the examples in the previous section, except for one thing - it will be executed locally on the system running this code, and it will not be processed by a remote worker.

```
>>> from vpoller.agent import VSphereAgent
>>> agent = VSphereAgent(
...     user='root',
...     pwd='p4ssw0rd',
...     host='vc01.example.org'
... )
>>> agent.connect()
>>> result = agent.vm_discover(msg={})
>>> print result
```

4.6.3 Interfacing with vPoller from other languages

Connecting to vPoller from other languages is easy as well.

vPoller uses the ZeroMQ messaging library as the communication layer, so in theory every language that comes with ZeroMQ bindings should be able to interface with vPoller.

You can find below a simple example of using Ruby for sending a task request to vPoller:

```
#!/usr/bin/env ruby

require 'json'
require 'rubygems'
require 'ffi-rzmq'

# Message we send to vPoller
msg = {:method => "vm.discover", :hostname => "vc01.example.org"}

# Create the ZeroMQ context and socket
context = ZMQ::Context.new(1)
socket = context.socket(ZMQ::REQ)

puts "Connecting to vPoller ..."
socket.connect("tcp://localhost:10123")

puts "Sending task request to vPoller ..."
socket.send_string(msg.to_json)

result = ''
socket.recv_string(result)

puts "Received reply from vPoller: #{result}"
```

You might also want to check the vpoller.client module for example code that you can use in order to implement a `VPollerClient` class in your language of choice.

4.7 vPoller Integration With Zabbix

One of the nice things about vPoller is that it can be easily integrated with other systems.

In this documentation we will see how we can integrate vPoller with Zabbix in order to start monitoring our VMware vSphere environment.

Note: This document is about VMware monitoring with vPoller and Zabbix, and **NOT** about VMware monitoring with stock Zabbix.

If you are looking for VMware monitoring with stock Zabbix, please refer to the official Zabbix documentation.

4.7.1 Why use vPoller with Zabbix and not just use stock Zabbix for VMware monitoring?

There are many things that can be put here describing the reasons and motivation why you might prefer having vPoller with Zabbix integration instead of stock Zabbix, but eventually this would end up being one long (and probably boring) story to write and tell.

You can read this post here, which outlines some very good reasons why you might want to have vPoller with Zabbix instead of stock Zabbix when it comes to VMware vSphere monitoring.

4.7.2 Prerequisites

This documentation assumes that you already have Zabbix installed and configured.

Next thing you need to make sure is that you have vPoller installed, configured and already running.

If you haven't installed and configured vPoller yet, please refer to the *Installation of vPoller* and *Configuration of vPoller* documentations first.

4.7.3 Enabling the vPoller Helpers for Zabbix

In order to be able to integrate vPoller with Zabbix we need to enable some of the vPoller helpers first.

Make sure that these vPoller helpers are enabled in your `vPoller Workers`:

- vpoller.helpers.zabbix
- vpoller.helpers.czabbix

For more information about how to enable the helpers in your `vPoller Workers`, please refer to the *vPoller Helpers* documentation.

4.7.4 Importing the vPoller templates in Zabbix

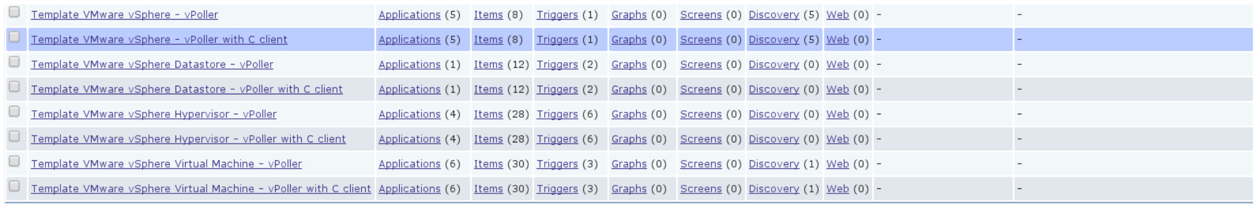
You can grab the latest vPoller templates for Zabbix from the Github repo of vPoller.

Note: Some of the Zabbix items from the vPoller templates are disabled by default. It is recommended that you review each vPoller template and enable or disable the items that you need or don't need at all.

In the vPoller templates for Zabbix directory you will find two directories:

- *vpoller-templates-externalchecks* - contains legacy templates to be used only with Zabbix external scripts
- *vpoller-templates-native* - contains the templates with native vPoller support for Zabbix. It is recommended that you always use the native vPoller support for Zabbix.
- *vpoller-templates-native-2.4* - same as the *vpoller-templates-native* templates, but for Zabbix 2.4.x releases.

Once you import the templates you should see the newly imported vPoller templates.



☐	Template VMware vSphere - vPoller	Applications (5)	Items (8)	Triggers (1)	Graphs (0)	Screens (0)	Discovery (5)	Web (0)	-	-
☐	Template VMware vSphere - vPoller with C client	Applications (5)	Items (8)	Triggers (1)	Graphs (0)	Screens (0)	Discovery (5)	Web (0)	-	-
☐	Template VMware vSphere Datastore - vPoller	Applications (1)	Items (12)	Triggers (2)	Graphs (0)	Screens (0)	Discovery (0)	Web (0)	-	-
☐	Template VMware vSphere Datastore - vPoller with C client	Applications (1)	Items (12)	Triggers (2)	Graphs (0)	Screens (0)	Discovery (0)	Web (0)	-	-
☐	Template VMware vSphere Hypervisor - vPoller	Applications (4)	Items (28)	Triggers (6)	Graphs (0)	Screens (0)	Discovery (0)	Web (0)	-	-
☐	Template VMware vSphere Hypervisor - vPoller with C client	Applications (4)	Items (28)	Triggers (6)	Graphs (0)	Screens (0)	Discovery (0)	Web (0)	-	-
☐	Template VMware vSphere Virtual Machine - vPoller	Applications (6)	Items (30)	Triggers (3)	Graphs (0)	Screens (0)	Discovery (1)	Web (0)	-	-
☐	Template VMware vSphere Virtual Machine - vPoller with C client	Applications (6)	Items (30)	Triggers (3)	Graphs (0)	Screens (0)	Discovery (1)	Web (0)	-	-

4.7.5 Native vPoller support for Zabbix

Native vPoller support for Zabbix makes it possible for Zabbix to talk natively to vPoller via a Zabbix loadable module

Native vPoller support for Zabbix is available only for Zabbix release versions 2.2.x or above, as loadable modules in Zabbix were introduced since the 2.2.x release of Zabbix.

Now, let's see how to build, install and configure the vPoller loadable module for Zabbix.

First, make sure that you have the ZeroMQ 4.x library installed as the vPoller loadable module for Zabbix is linked against it.

Here is how to install the ZeroMQ 4.x library on your system from source:

```
$ git clone https://github.com/zeromq/zeromq4-x.git
$ cd zeromq4-x
$ ./autogen.sh
$ ./configure
$ make && sudo make install && make clean
$ sudo ldconfig
```

Next thing you need to do is get the Zabbix source package for your Zabbix release from the Zabbix Download page. We need the source package of Zabbix in order to build the vPoller loadable module.

Get the source package for your Zabbix release. For instance if you are running Zabbix version 2.2.5 you should download the source package for version 2.2.5 of Zabbix.

In the example commands below we are using the source package for Zabbix version 2.2.5.

```
$ tar zxvf zabbix-2.2.5.tar.gz
$ cd zabbix-2.2.5
$ ./configure
```

The next step we need to do is to grab the vPoller loadable module for Zabbix from the Github repo of vPoller and place the module in the `zabbix-2.2.5/src/modules` directory where you have unpacked the Zabbix source package.

```
$ cp -a py-vpoller/extra/zabbix/vpoller-module zabbix-2.2.5/src/modules
```

Building the vPoller module for Zabbix is now easy.

```
$ cd zabbix-2.2.5/src/modules/vpoller-module
$ make
```

Running the `make(1)` command will create the shared library `vpoller.so`, which can now be loaded by your Zabbix Server, Proxy and Agents.

Let's now load the `vpoller.so` module in the Zabbix Server during startup. In order to load the module you need to edit your `zabbix_server.conf` file and update the `LoadModulePath` and `LoadModule` configuration options. Below is an example snippet from the `zabbix_server.conf` file, which loads the `vpoller.so` module.

```
####### LOADABLE MODULES #######

### Option: LoadModulePath
#       Full path to location of server modules.
#       Default depends on compilation options.
#
# Mandatory: no
# Default:
LoadModulePath=/usr/local/lib/zabbix

### Option: LoadModule
#       Module to load at server startup. Modules are used to extend functionality of␣
↪the server.
#       Format: LoadModule=<module.so>
#       The modules must be located in directory specified by LoadModulePath.
#       It is allowed to include multiple LoadModule parameters.
#
# Mandatory: no
# Default:
LoadModule=vpoller.so
```

Make sure that you copy the `vpoller.so` module, which you've built to your `LoadModulePath` directory.

```
$ sudo cp zabbix-2.2.5/src/modules/vpoller-module/vpoller.so /usr/local/lib/zabbix
```

Once ready with the configuration changes make sure to restart any service for which you've just updated the config file.

You can verify that the `vpoller.so` module has been successfully loaded by inspecting your Zabbix logs. In the log snippet below you can see that our Zabbix Server has successfully loaded the `vpoller.so` module.

```
13352:20140910:080628.011 Starting Zabbix Server. Zabbix 2.2.5 (revision 47411).
13352:20140910:080628.012 ****** Enabled features ******
13352:20140910:080628.012 SNMP monitoring:           YES
13352:20140910:080628.012 IPMI monitoring:           YES
13352:20140910:080628.012 WEB monitoring:            YES
13352:20140910:080628.012 VMware monitoring:         YES
13352:20140910:080628.012 Jabber notifications:      YES
13352:20140910:080628.012 Ez Texting notifications:  YES
13352:20140910:080628.012 ODBC:                      YES
13352:20140910:080628.012 SSH2 support:              YES
13352:20140910:080628.012 IPv6 support:              YES
13352:20140910:080628.012 ******************************
13352:20140910:080628.012 using configuration file: /etc/zabbix/zabbix_server.conf
13352:20140910:080628.013 Loading vPoller module configuration file /etc/zabbix/
↪vpoller_module.conf
13352:20140910:080628.015 loaded modules: vpoller.so
```

The vPoller loadable module for Zabbix can use an optional configuration file which allows you to manage some of the vPoller settings, such as the task timeout, retries and endpoint of the `vPoller Proxy` to which task requests are being sent.

The configuration of the `vpoller.so` module resides in the `/etc/zabbix/vpoller_module.conf` file and you can find a sample configuration file in the vPoller loadable module for Zabbix directory from the Github repo.

4.7.6 The Zabbix vPoller Key

Once loaded the vPoller module for Zabbix exposes a single key of type `Simple check` that can be used by your Zabbix items and is called `vpoller[*]`.

The `vpoller[*]` Zabbix key has the following form:

```
vpoller[method, hostname, name, properties, <key>, <username>, <password>, <counter-
↪name>, <instance>, <perf-interval>]
```

And the parameters that `vpoller[*]` key expects are these.

Parameter	Description	Required
method	vPoller method to be processed	True
hostname	VMware vSphere server hostname	True
name	Name of the vSphere object (e.g. VM name, ESXi name)	True
properties	vSphere object properties to be collected by vPoller	True
<key>	Additional information to be passed to vPoller	False
<username>	Username to use when logging into the guest system	False
<password>	Password to use when logging into the guest system	False
<counter-name>	Performance counter name to be retrieved	False
<instance>	Performance counter instance	False
<perf-interval>	Historical performance interval	False

Note that some of the above parameters are mandatory and some are optional depending on what vPoller method you are requesting to be processed.

If your Zabbix Agents are also loading the `vpoller.so` module you can use the `zabbix_get(8)` tool from the command-line in order to send task requests to vPoller.

Here is one example that uses `zabbix_get(8)` in order check the power state of VM using the `vpoller[*]` key.

```
$ zabbix_get -s 127.0.0.1 -p 10050 -k "vpoller[vm.get, vc01.example.org, ns01.example.
↪org, runtime.powerState]"
"poweredOn"
```

4.7.7 Setting up vPoller externalscripts for Zabbix

Note: This section of the documentation provides instructions how to install the vPoller `externalscripts` in Zabbix.

It is recommended that you always use the `native vPoller support for Zabbix` when integrating vPoller with Zabbix, and use `externalscripts` only if you cannot have the native vPoller support for Zabbix, e.g. you are running an older Zabbix release which doesn't support loadable modules or the loadable module is not available for your platform.

Get the `vpoller-zabbix` and `cvpoller-zabbix` wrapper scripts from the links below and place them in your Zabbix `externalscripts` directory:

- https://github.com/dnaeon/py-vpoller/blob/master/extra/zabbix/externalscripts/vpoller-zabbix
- https://github.com/dnaeon/py-vpoller/blob/master/extra/zabbix/externalscripts/cvpoller-zabbix

You can also find user-contributed `vpoller-zabbix` and `cvpoller-zabbix` wrapper scripts, which come with more features and safety checks at the links below:

- https://github.com/dnaeon/py-vpoller/blob/master/contrib/zabbix/externalscripts/vpoller-zabbix
- https://github.com/dnaeon/py-vpoller/blob/master/contrib/zabbix/externalscripts/cvpoller-zabbix

Using any of these wrapper scripts should be fine.

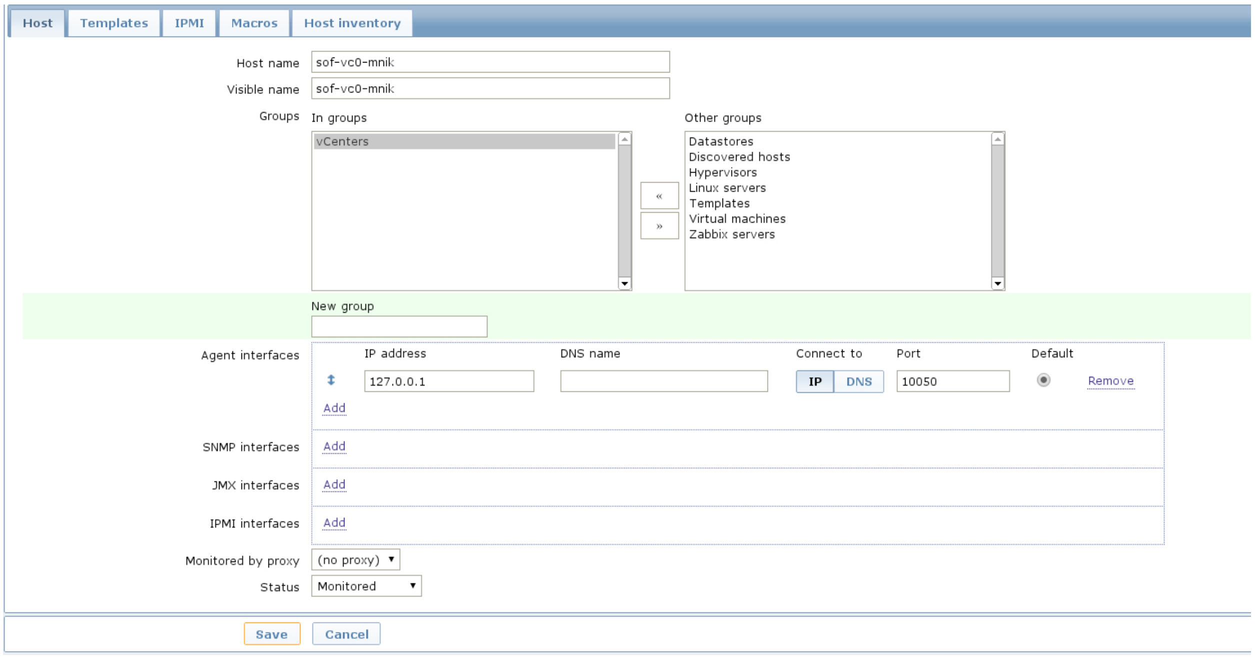
Place the `vpoller-zabbix` and `cvpoller-zabbix` wrapper scripts into your Zabbix `externalscripts` directory and make sure they are executable as well:

```
$ sudo chmod 0755 $externalscripts/vpoller-zabbix $externalscripts/cvpoller-zabbix
```

4.7.8 Monitoring your VMware environment with vPoller and Zabbix

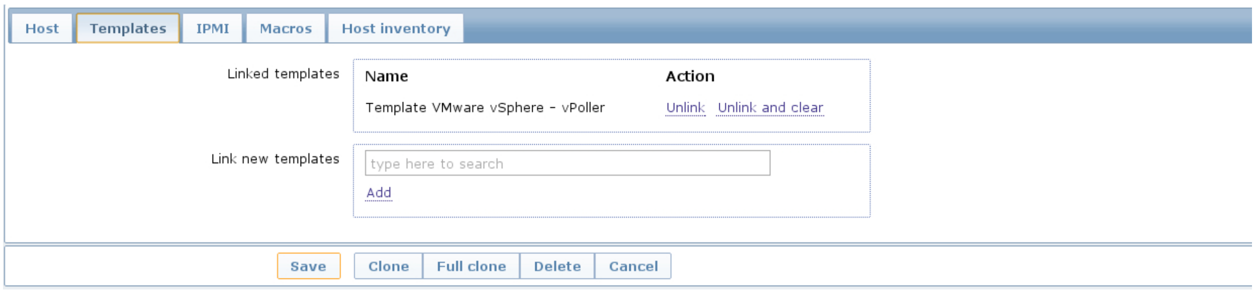
Time to start monitoring our VMware vSphere environment with vPoller and Zabbix. Let's go ahead and add a VMware vCenter server and get some data out of it.

Login to your Zabbix frontend and navigate to `Configuration -> Hosts`, then at the top-right corner click on the `Create host` button. Fill in the hostname of the vCenter we are going to monitor and add it to a group, e.g. vCenters in my case.



Host name: sof-vc0-mnik
Visible name: sof-vc0-mnik
Groups: In groups: vCenters
Other groups: Datastores, Discovered hosts, Hypervisors, Linux servers, Templates, Virtual machines, Zabbix servers
New group:
Agent interfaces: IP address 127.0.0.1; DNS name; Connect to IP; Port 10050; Default ●
SNMP interfaces: Add
JMX interfaces: Add
IPMI interfaces: Add
Monitored by proxy: (no proxy)
Status: Monitored

Next, click on the `Templates` and link the `Template VMware vSphere - vPoller` template if you are using vPoller with external checks support or use the `Template VMware vSphere - vPoller Native` template for native vPoller support in Zabbix.

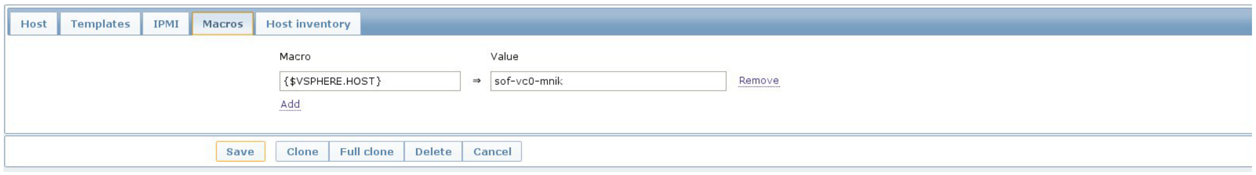


Linked templates:

Name	Action
Template VMware vSphere - vPoller	Unlink Unlink and clear

Link new templates: type here to search

The last thing we need to do is add a Zabbix macro to our vSphere host. Navigate to the `Macros` tab and add the `{$VSPHERE.HOST}` macro which value should be the hostname of the vSphere host you are adding to Zabbix.



Once done, click the `Save` button and you are ready.

Soon enough Zabbix will start sending requests to vPoller which would discover your vSphere objects (ESXi hosts, Virtual Machines, Datastores, etc) and start monitoring them.

4.7.9 Importing vSphere objects as regular Zabbix hosts

In the previous section of this documentation we have seen how we can use Zabbix with vPoller working together in order to perform monitoring of our VMware vSphere environment.

The way we did it is by using vPoller in order to discover VMware vSphere objects and then use the Zabbix Low-level discovery protocol in order to create hosts based on the discovered data.

While `Zabbix Low-level discovery` is a powerful feature of Zabbix which you could use in order to automate the process of discovering and adding hosts to your Zabbix server, it still has some limitations and disadvantages.

One disadvantage of using Zabbix LLD is that once a host is being created by a Zabbix Discovery Rule that host becomes immutable - you cannot manually change or update anything on the host, unless these changes come from the discovery rule or the host profile applied to the host.

You can imagine that this might be a bit of frustrating when you want to group your hosts in a better way for example, which obviously you cannot do since this host is now immutable.

Linking additional templates to a discovered host is also not possible, which is another big issue. Now that you've discovered your VMware Virtual Machines you probably wanted to link some additional templates to them, but you will soon discover that this is not possible either.

You cannot even add more interfaces to your hosts if needed… Like mentioned earlier - your host is immutable, so that means no changes at all after your hosts have been discovered with a Zabbix LLD rule.

So, what can we do about it?

Well, we can solve this issue! And vPoller is going to help us do that! :)

We are going to use the zabbix-vsphere-import tool, which can discover and import vSphere objects as regular Zabbix hosts - that means that all vSphere objects (ESXi hosts, Virtual Machines, Datastores, etc.) which were imported by the zabbix-vsphere-import tool would be regular Zabbix hosts, which you could update - adding the host to groups you want, linking arbitrary templates to it, etc.

First, let's create the config file which zabbix-vsphere-import will be using. Below is an example config file used by `zabbix-vsphere-import` tool:

```
---
vsphere:
  hostname: vc01.example.org

vpoller:
  endpoint: tcp://localhost:10123
  retries: 3
  timeout: 3000
```

(continues on next page)

(continued from previous page)

```
zabbix:
  hostname: http://zabbix.example.org/zabbix
  username: Admin
  password: zabbix

  vsphere_object_host:
    proxy: zbx-proxy.example.org
    templates:
      - Template VMware vSphere Hypervisor - vPoller Native
    macros:
      VSPHERE.HOST: vc01.example.org
    groups:
      - Hypervisors

  vsphere_object_vm:
    templates:
      - Template VMware vSphere Virtual Machine - vPoller Native
    macros:
      VSPHERE.HOST: vc01.example.org
    groups:
      - Virtual Machines

  vsphere_object_datastore:
    templates:
      - Template VMware vSphere Datastore - vPoller Native
    macros:
      VSPHERE.HOST: vc01.example.org
    groups:
      - Datastores
```

In the example config file above we have defined various config entries - Zabbix server, Zabbix Proxy which will be used, vPoller settings and also templates to be linked for the various vSphere objects.

As you can see the format of the configuration file allows for flexible setup of your discovered vSphere objects.

Time to import our vSphere objects as regular Zabbix hosts. To do that simply execute the command below:

```
$ zabbix-vsphere-import -f zabbix-vsphere-import.yaml
```

Here is an example output of running the zabbix-vsphere-import tool:

```
$ zabbix-vsphere-import -f zabbix-vsphere-import.yaml
[2014-09-06 10:33:28,420] - INFO - Connecting to Zabbix server at http://zabbix.
↪example.org/zabbix
[2014-09-06 10:33:28,537] - INFO - [vSphere ClusterComputeResource] Importing objects␣
↪to Zabbix
[2014-09-06 10:33:28,814] - INFO - [vSphere ClusterComputeResource] Number of objects␣
↪to be imported: 1
[2014-09-06 10:33:28,814] - INFO - [vSphere ClusterComputeResource] Creating Zabbix␣
↪host group 'cluster01'
[2014-09-06 10:33:28,904] - INFO - [vSphere ClusterComputeResource] Import of objects␣
↪completed
[2014-09-06 10:33:28,904] - INFO - [vSphere HostSystem] Importing objects to Zabbix
[2014-09-06 10:33:29,122] - INFO - [vSphere HostSystem] Number of objects to be␣
↪imported: 2
[2014-09-06 10:33:29,289] - INFO - [vSphere HostSystem] Creating Zabbix host 'esxi01.
↪example.org'
```

(continues on next page)

(continued from previous page)

```
[2014-09-06 10:33:30,204] - INFO - [vSphere HostSystem] Creating Zabbix host 'esxi02.
↪example.org'
[2014-09-06 10:33:30,658] - INFO - [vSphere HostSystem] Import of objects completed
[2014-09-06 10:33:30,658] - INFO - [vSphere VirtualMachine] Importing objects to␣
↪Zabbix
[2014-09-06 10:33:30,775] - INFO - [vSphere VirtualMachine] Number of objects to be␣
↪imported: 9
[2014-09-06 10:33:30,935] - WARNING - Unable to find Zabbix host group 'Virtual␣
↪Machines'
[2014-09-06 10:33:30,936] - INFO - Creating Zabbix host group 'Virtual Machines'
[2014-09-06 10:33:33,965] - INFO - [vSphere VirtualMachine] Creating Zabbix host
↪'ubuntu-14.04-dev'
[2014-09-06 10:33:34,956] - INFO - [vSphere VirtualMachine] Creating Zabbix host
↪'centos-6.5-amd64'
[2014-09-06 10:33:35,945] - INFO - [vSphere VirtualMachine] Creating Zabbix host 'sof-
↪vc0-mnik'
[2014-09-06 10:33:36,441] - INFO - [vSphere VirtualMachine] Creating Zabbix host
↪'test-vm-01'
[2014-09-06 10:33:36,934] - INFO - [vSphere VirtualMachine] Creating Zabbix host 'sof-
↪dev-d7-mnik'
[2014-09-06 10:33:37,432] - INFO - [vSphere VirtualMachine] Creating Zabbix host
↪'ubuntu-12.04-desktop'
[2014-09-06 10:33:43,430] - INFO - [vSphere VirtualMachine] Creating Zabbix host
↪'zabbix-vm-2'
[2014-09-06 10:33:43,929] - INFO - [vSphere VirtualMachine] Creating Zabbix host
↪'zabbix-vm-1'
[2014-09-06 10:33:44,432] - INFO - [vSphere VirtualMachine] Creating Zabbix host
↪'VMware vCenter Server Appliance'
[2014-09-06 10:33:44,937] - INFO - [vSphere VirtualMachine] Import of objects␣
↪completed
[2014-09-06 10:33:44,937] - INFO - [vSphere Datastore] Importing objects to Zabbix
[2014-09-06 10:33:45,046] - INFO - [vSphere Datastore] Number of objects to be␣
↪imported: 1
[2014-09-06 10:33:45,339] - INFO - [vSphere Datastore] Creating host 'ds:///vmfs/
↪volumes/5190e2a7-d2b7c58e-b1e2-90b11c29079d/'
[2014-09-06 10:33:45,607] - INFO - [vSphere Datastore] Import of objects completed
```

Generally you would want to run the import perhaps once an hour (e.g. from `cron(8)`), so that your Zabbix server is in sync with your vSphere environment.

If you are importing your vSphere objects in Zabbix using the `zabbix-vsphere-import` tool make sure to disable any Zabbix LLD discovery rules in order to avoid any conflicts between them.

4.7.10 Agent-less process monitoring in Virtual Machines

Another cool feature of vPoller is the ability to perform process monitoring inside VMware Virtual Machines without the need of having Zabbix Agents (or any other software) installed and running on your systems.

This can be quite handy in situations where you don't have the Zabbix Agents installed or you are not even allowed to install any software on your Virtual Machines.

A good example is a service provider where customers request that specific process availability be monitored in Virtual Machines, but don't want to have any third-party software installed on the customers' systems.

In case you are wondering how we perform the agent-less process monitoring of VMware Virtual Machines using vPoller, you may want to check the vSphere API documentation for GuestProcessManager().

Let's see now how we can use vPoller with Zabbix integration in order to provide agent-less process monitoring for our Virtual Machines.

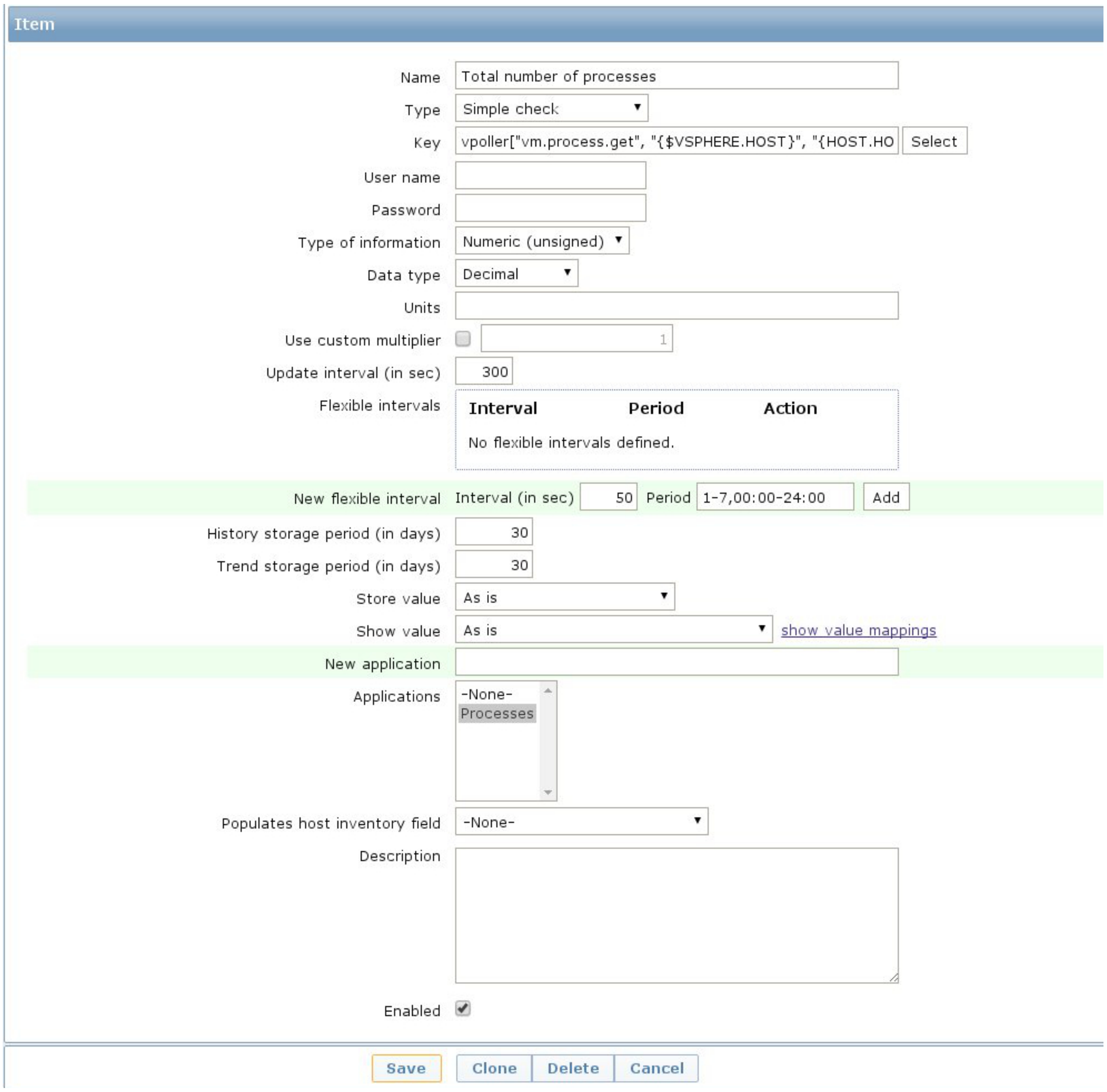
First we will create a Zabbix item that will monitor the total number of processes in a Virtual Machine and then we will see how we can monitor the availability for certain processes.

The Zabbix key that we will use for agent-less process monitoring is of type `Simple check` and has the following format:

```
vpoller["vm.process.get", "{$VSPHERE.HOST}", "{HOST.HOST}", "cmdLine", "", username,␣
↪password]
```

In the above Zabbix key the `username` and `password` parameters should be a valid username and password that can login to the guest system.

On the screenshot below we are creating a new Zabbix item that will monitor the total number of processes in our Virtual Machine.

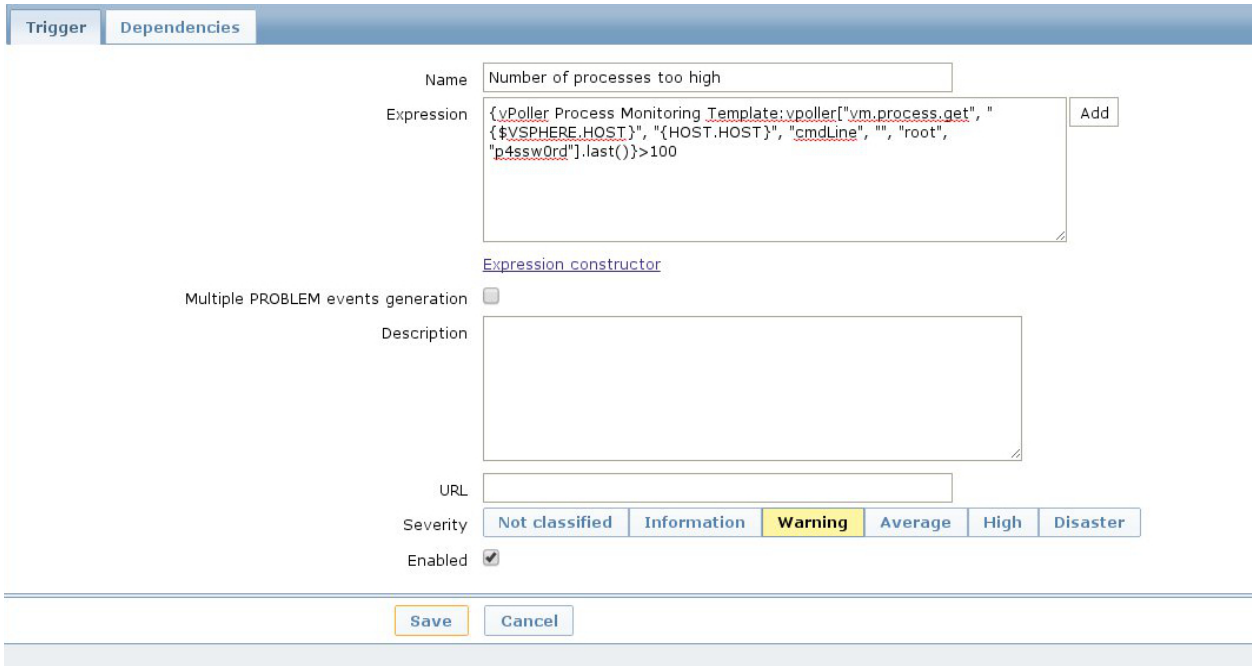


Name	Total number of processes
Type	Simple check
Key	vpoller["vm.process.get", "{$VSPHERE.HOST}", "{HOST.HO
User name	
Password	
Type of information	Numeric (unsigned)
Data type	Decimal
Units	
Use custom multiplier	1
Update interval (in sec)	300
Flexible intervals	Interval Period Action — No flexible intervals defined.
New flexible interval	Interval (in sec) 50 Period 1-7,00:00-24:00
History storage period (in days)	30
Trend storage period (in days)	30
Store value	As is
Show value	As is
New application	
Applications	-None- Processes
Populates host inventory field	-None-
Description	
Enabled	✓

The key that we've used for monitoring the total number of processes in our guest system is this:

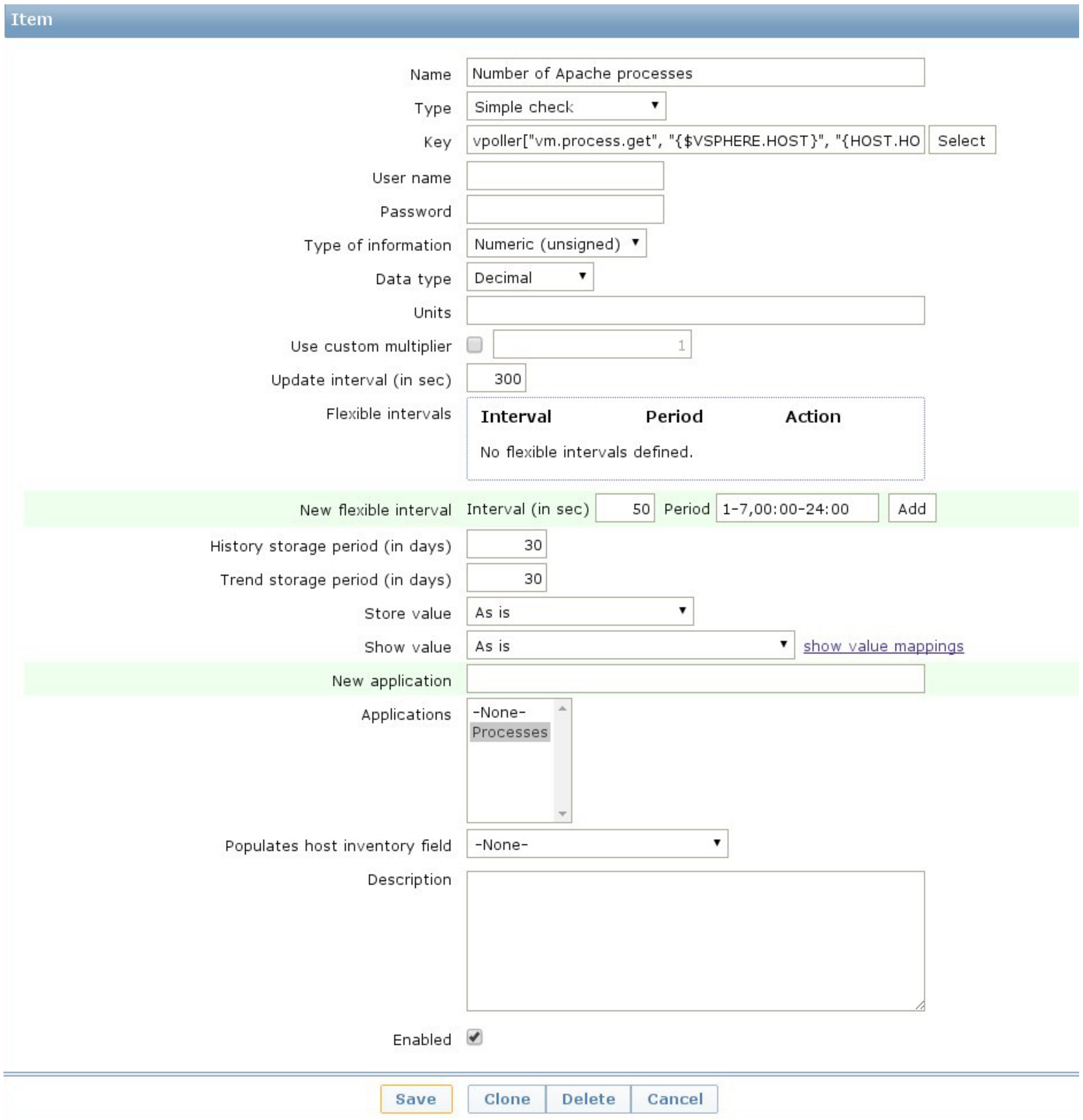
```
vpoller["vm.process.get", "{$VSPHERE.HOST}", "{HOST.HOST}", "cmdLine", "", root,␣
↪p4ssw0rd]
```

We can also create a trigger for our item which will go into certain state whenever the total number of processes exceeds a certain value.



Name	Number of processes too high
Expression	{vPoller Process Monitoring Template:vpoller["vm.process.get", "{$VSPHERE.HOST}", "{HOST.HOST}", "cmdLine", "", "root", "p4ssw0rd"].last()}>100
Multiple PROBLEM events generation	☐
Description	
URL	
Severity	Not classified / Information / [✓ Warning] / Average / High / Disaster
Enabled	✓

Now, let's add a second item which this time will be monitoring the number of Apache processes in our Virtual Machine.



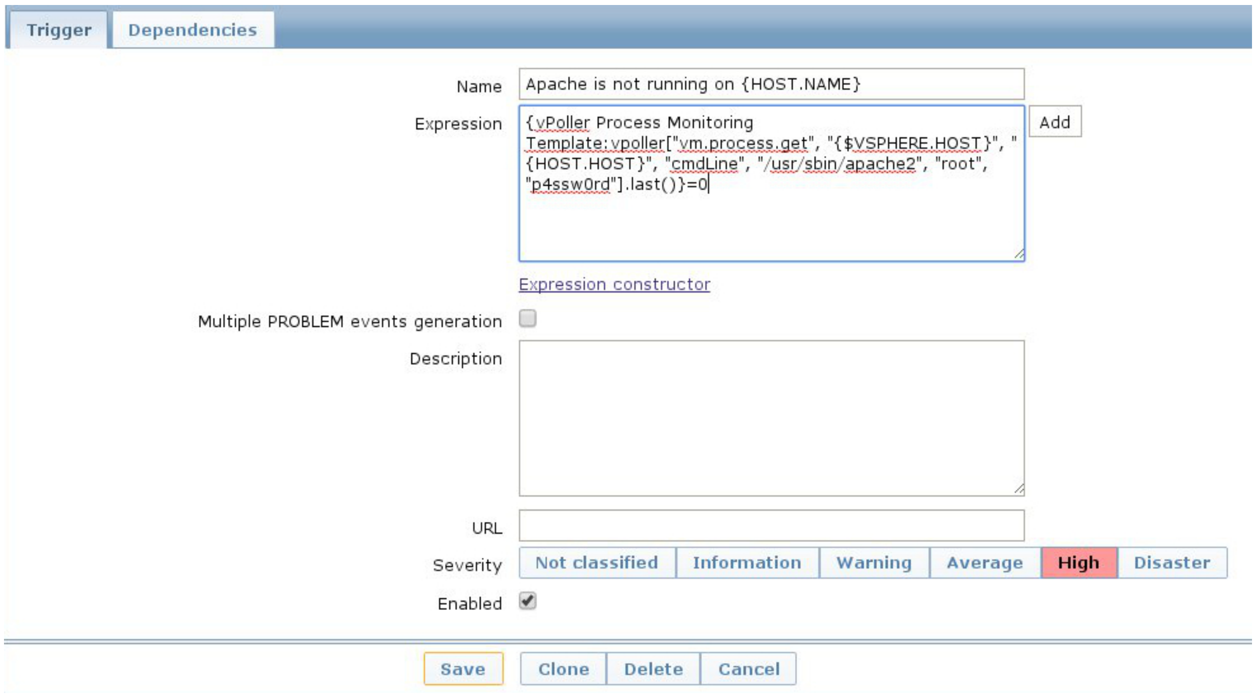
Item

Name	Number of Apache processes
Type	Simple check
Key	vpoller["vm.process.get", "{$VSPHERE.HOST}", "{HOST.HO
User name	
Password	
Type of information	Numeric (unsigned)
Data type	Decimal
Units	
Use custom multiplier	☐ 1
Update interval (in sec)	300
Flexible intervals	Interval Period Action — No flexible intervals defined.
New flexible interval	Interval (in sec) 50 Period 1-7,00:00-24:00
History storage period (in days)	30
Trend storage period (in days)	30
Store value	As is
Show value	As is
New application	
Applications	-None- Processes
Populates host inventory field	-None-
Description	
Enabled	✓

On the screenshot above we have used the following Zabbix key in order to monitor the number of Apache processes in our Virtual Machine.

```
vpoller["vm.process.get", "{$VSPHERE.HOST}", "{HOST.HOST}", "cmdLine", "/usr/sbin/
↪apache2", root, p4ssw0rd]
```

Should we want to be notified in case our process is not running we can create a trigger for our item and set the severity level of the issue.



Name: Apache is not running on {HOST.NAME}

Expression: {vPoller Process Monitoring Template:vpoller["vm.process.get", "{$VSPHERE.HOST}", "{HOST.HOST}", "cmdLine", "/usr/sbin/apache2", "root", "p4ssw0rd"].last()}=0

Multiple PROBLEM events generation: ☐

Description:

URL:

Severity: Not classified | Information | Warning | Average | [✓ High] | Disaster

Enabled: ✓

Note: It is recommended that you use a system account with restricted set of privileges when you perform agent-less process monitoring with vPoller and Zabbix.

You may also want to consider creating a global Zabbix macro for the system account username and password and use it in your Zabbix keys, without having the need to include the username and password in every single process-monitoring item that you want to have.

Global macros in Zabbix can be created by navigating to `Administration -> General -> Macros` in your Zabbix Dashboard.

As a final example on agent-less process monitoring with vPoller and Zabbix we will see how to query the number of process from the command-line using the `zabbix_get(8)` tool.

Here's how to query the total number of processes in a Virtual Machine from the command-line:

```
$ zabbix_get -s 127.0.0.1 \
             -p 10050 \
             -k 'vpoller[vm.process.get, vc01.example.org, vm01.example.org, cmdLine,
↪"", root, p4ssw0rd]'
```

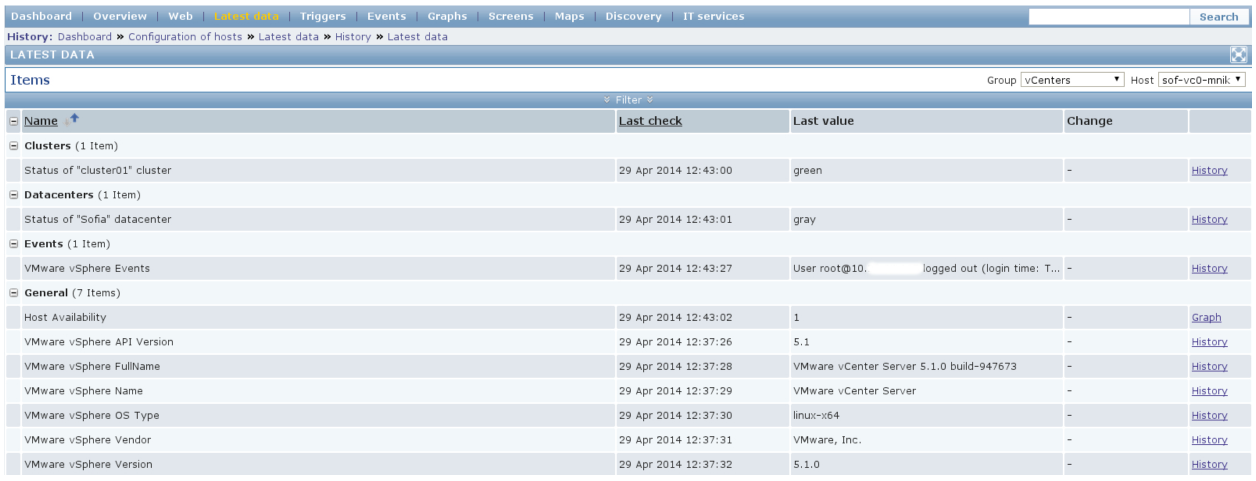
And this is how to query the number of certain processes in a Virtual Machine using `zabbix_get(8)`:

```
$ zabbix_get -s 127.0.0.1 \
             -p 10050 \
             -k 'vpoller[vm.process.get, vc01.example.org, vm01.example.org, cmdLine,
↪"/usr/sbin/apache2", root, p4ssw0rd]'
```

4.7.11 Example screenshots

Let's see some example screenshots of Zabbix monitoring a VMware vSphere environment using vPoller.

Checking the latest data of our vCenter server in Zabbix:

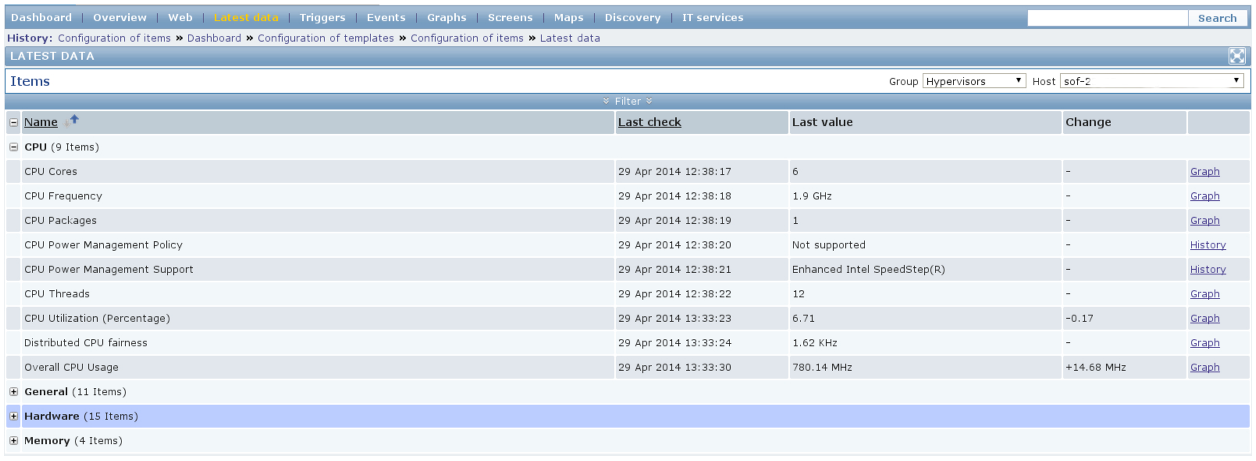

Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services
Search
LATEST DATA
Items
Filter

History: Dashboard » Configuration of hosts » Latest data » History » Latest data

Group vCenters Host sof-vc0-mnik

Name	Last check	Last value	Change	
Clusters (1 Item)				
Status of "cluster01" cluster	29 Apr 2014 12:43:00	green	-	History
Datacenters (1 Item)				
Status of "Sofia" datacenter	29 Apr 2014 12:43:01	gray	-	History
Events (1 Item)				
VMware vSphere Events	29 Apr 2014 12:43:27	User root@10.[redacted] logged out (login time: T...	-	History
General (7 Items)				
Host Availability	29 Apr 2014 12:43:02	1	-	Graph
VMware vSphere API Version	29 Apr 2014 12:37:26	5.1	-	History
VMware vSphere FullName	29 Apr 2014 12:37:28	VMware vCenter Server 5.1.0 build-947673	-	History
VMware vSphere Name	29 Apr 2014 12:37:29	VMware vCenter Server	-	History
VMware vSphere OS Type	29 Apr 2014 12:37:30	linux-x64	-	History
VMware vSphere Vendor	29 Apr 2014 12:37:31	VMware, Inc.	-	History
VMware vSphere Version	29 Apr 2014 12:37:32	5.1.0	-	History

Let's see the latest data for some of our ESXi hosts:

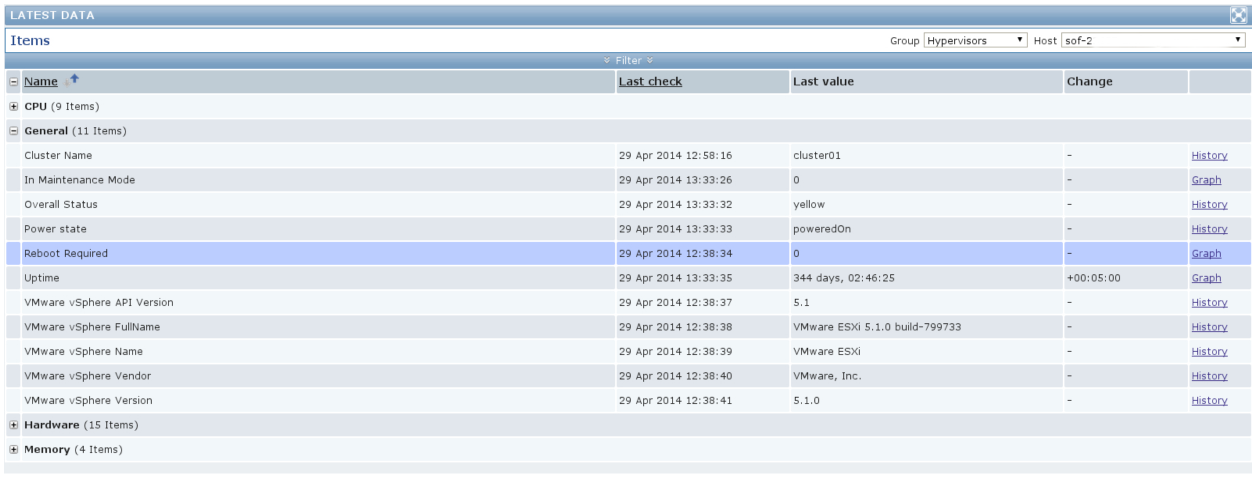

Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services
Search
LATEST DATA
Items
Filter

History: Configuration of items » Dashboard » Configuration of templates » Configuration of items » Latest data

Group Hypervisors Host sof-2

Name	Last check	Last value	Change	
CPU (9 Items)				
CPU Cores	29 Apr 2014 12:38:17	6	-	Graph
CPU Frequency	29 Apr 2014 12:38:18	1.9 GHz	-	Graph
CPU Packages	29 Apr 2014 12:38:19	1	-	Graph
CPU Power Management Policy	29 Apr 2014 12:38:20	Not supported	-	History
CPU Power Management Support	29 Apr 2014 12:38:21	Enhanced Intel SpeedStep(R)	-	History
CPU Threads	29 Apr 2014 12:38:22	12	-	Graph
CPU Utilization (Percentage)	29 Apr 2014 13:33:23	6.71	-0.17	Graph
Distributed CPU fairness	29 Apr 2014 13:33:24	1.62 KHz	-	Graph
Overall CPU Usage	29 Apr 2014 13:33:30	780.14 MHz	+14.68 MHz	Graph
General (11 Items)				
Hardware (15 Items)				
Memory (4 Items)				

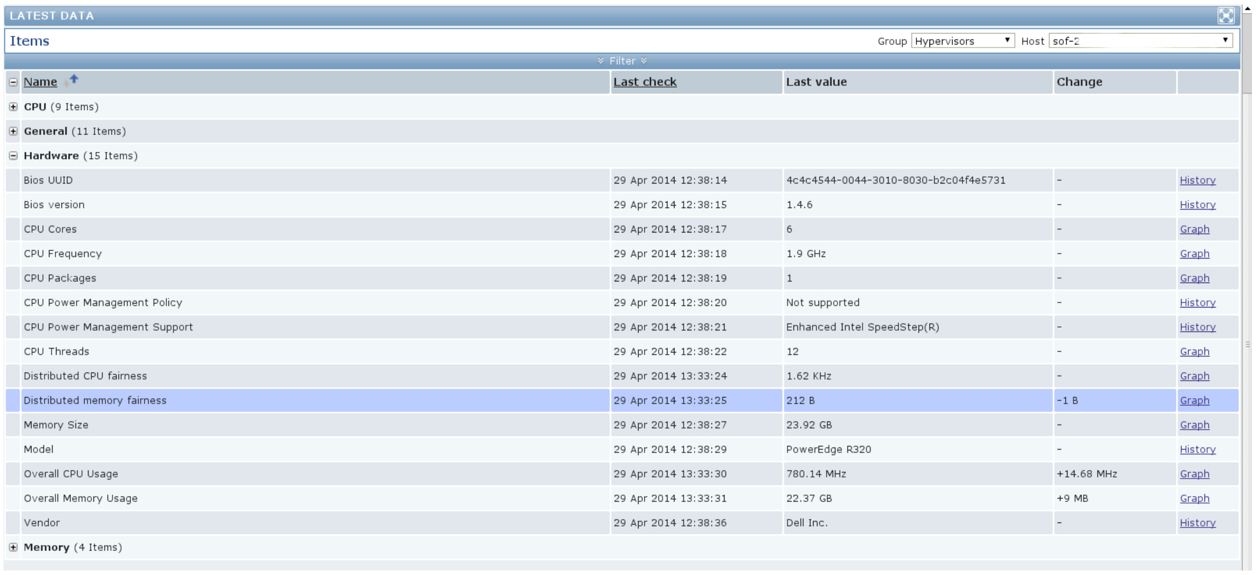
Another screenshot showing information about our ESXi host:



Group Hypervisors Host sof-2

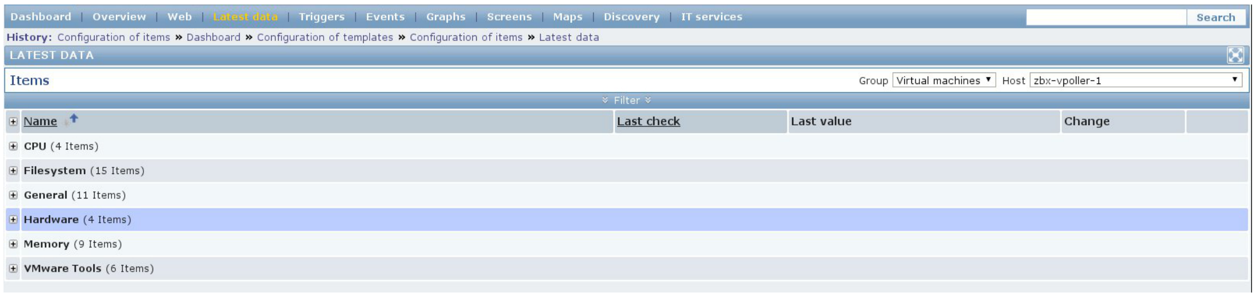
Name	Last check	Last value	Change	
CPU (9 Items)				
General (11 Items)				
Cluster Name	29 Apr 2014 12:58:16	cluster01	-	History
In Maintenance Mode	29 Apr 2014 13:33:26	0	-	Graph
Overall Status	29 Apr 2014 13:33:32	yellow	-	History
Power state	29 Apr 2014 13:33:33	poweredOn	-	History
Reboot Required	29 Apr 2014 12:38:34	0	-	Graph
Uptime	29 Apr 2014 13:33:35	344 days, 02:46:25	+00:05:00	Graph
VMware vSphere API Version	29 Apr 2014 12:38:37	5.1	-	History
VMware vSphere FullName	29 Apr 2014 12:38:38	VMware ESXi 5.1.0 build-799733	-	History
VMware vSphere Name	29 Apr 2014 12:38:39	VMware ESXi	-	History
VMware vSphere Vendor	29 Apr 2014 12:38:40	VMware, Inc.	-	History
VMware vSphere Version	29 Apr 2014 12:38:41	5.1.0	-	History
Hardware (15 Items)				
Memory (4 Items)				

And another screenshot showing hardware related information about our ESXi host:



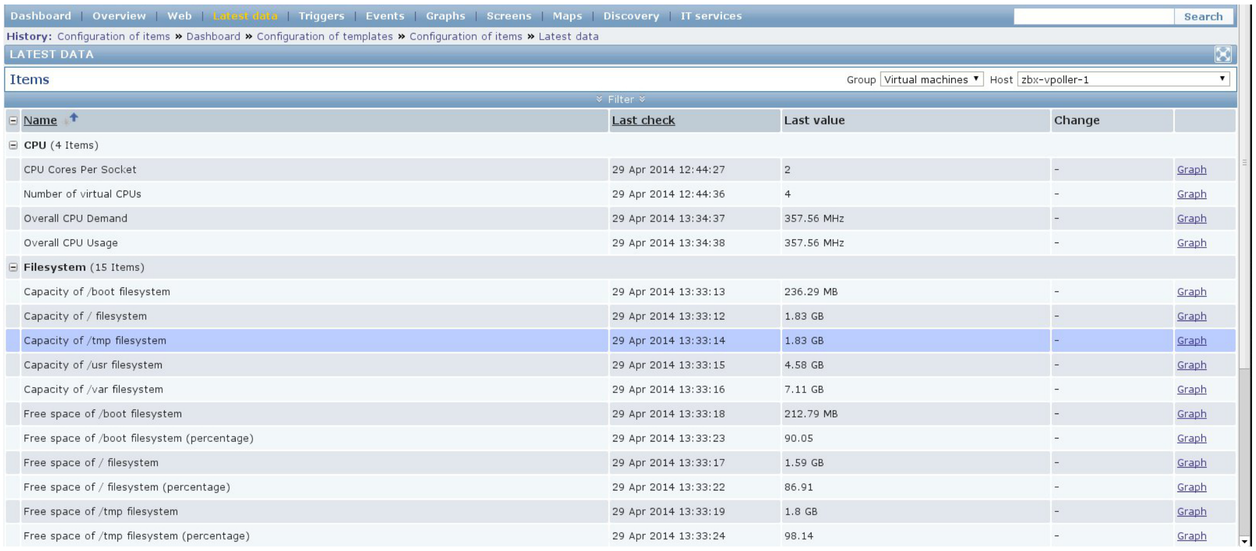
Name	Last check	Last value	Change	
CPU (9 Items)				
General (11 Items)				
Hardware (15 Items)				
Bios UUID	29 Apr 2014 12:38:14	4c4c4544-0044-3010-8030-b2c04f4e5731	-	History
Bios version	29 Apr 2014 12:38:15	1.4.6	-	History
CPU Cores	29 Apr 2014 12:38:17	6	-	Graph
CPU Frequency	29 Apr 2014 12:38:18	1.9 GHz	-	Graph
CPU Packages	29 Apr 2014 12:38:19	1	-	Graph
CPU Power Management Policy	29 Apr 2014 12:38:20	Not supported	-	History
CPU Power Management Support	29 Apr 2014 12:38:21	Enhanced Intel SpeedStep(R)	-	History
CPU Threads	29 Apr 2014 12:38:22	12	-	Graph
Distributed CPU fairness	29 Apr 2014 13:33:24	1.62 KHz	-	Graph
Distributed memory fairness	29 Apr 2014 13:33:25	212 B	-1 B	Graph
Memory Size	29 Apr 2014 12:38:27	23.92 GB	-	Graph
Model	29 Apr 2014 12:38:29	PowerEdge R320	-	History
Overall CPU Usage	29 Apr 2014 13:33:30	780.14 MHz	+14.68 MHz	Graph
Overall Memory Usage	29 Apr 2014 13:33:31	22.37 GB	+9 MB	Graph
Vendor	29 Apr 2014 12:38:36	Dell Inc.	-	History
Memory (4 Items)				

Let's check the latest data for one of our Virtual Machines:


Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services
Search
History: Configuration of items » Dashboard » Configuration of templates » Configuration of items » Latest data
LATEST DATA
Items
Group Virtual machines
Host zbx-vpoller-1
Filter

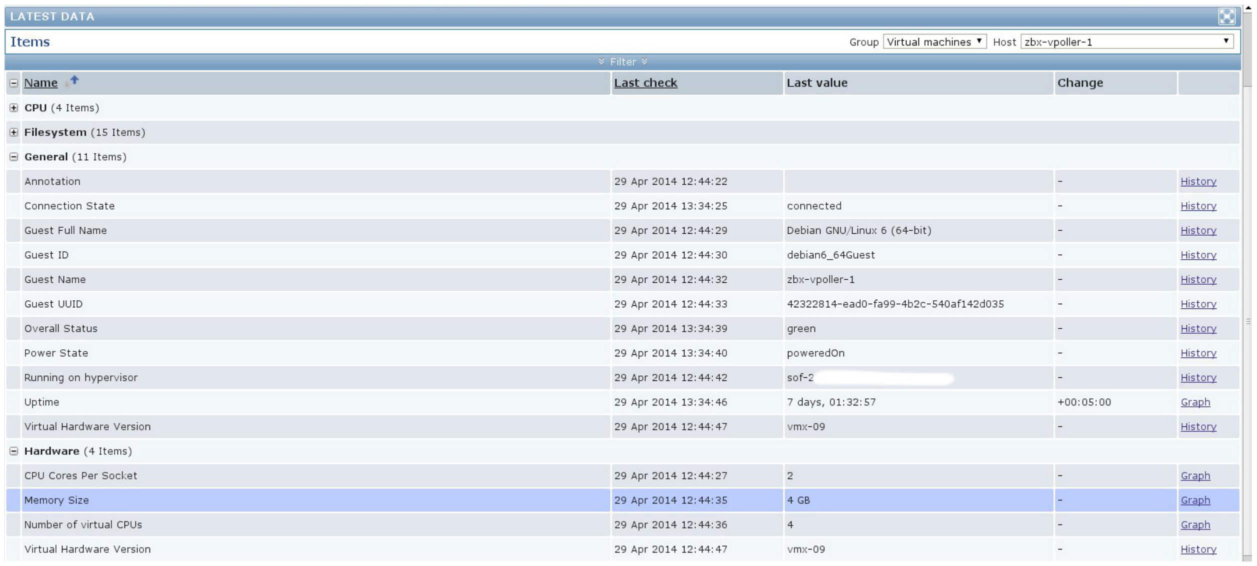
Name	Last check	Last value	Change	
CPU (4 Items)				
Filesystem (15 Items)				
General (11 Items)				
Hardware (4 Items)				
Memory (9 Items)				
VMware Tools (6 Items)				

A screenshot showing information about the file systems in Virtual Machine:


Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services
Search
History: Configuration of items » Dashboard » Configuration of templates » Configuration of items » Latest data
LATEST DATA
Items
Group Virtual machines
Host zbx-vpoller-1
Filter

Name	Last check	Last value	Change	
CPU (4 Items)				
CPU Cores Per Socket	29 Apr 2014 12:44:27	2	-	Graph
Number of virtual CPUs	29 Apr 2014 12:44:36	4	-	Graph
Overall CPU Demand	29 Apr 2014 13:34:37	357.56 MHz	-	Graph
Overall CPU Usage	29 Apr 2014 13:34:38	357.56 MHz	-	Graph
Filesystem (15 Items)				
Capacity of /boot filesystem	29 Apr 2014 13:33:13	236.29 MB	-	Graph
Capacity of / filesystem	29 Apr 2014 13:33:12	1.83 GB	-	Graph
Capacity of /tmp filesystem	29 Apr 2014 13:33:14	1.83 GB	-	Graph
Capacity of /usr filesystem	29 Apr 2014 13:33:15	4.58 GB	-	Graph
Capacity of /var filesystem	29 Apr 2014 13:33:16	7.11 GB	-	Graph
Free space of /boot filesystem	29 Apr 2014 13:33:18	212.79 MB	-	Graph
Free space of /boot filesystem (percentage)	29 Apr 2014 13:33:23	90.05	-	Graph
Free space of / filesystem	29 Apr 2014 13:33:17	1.59 GB	-	Graph
Free space of / filesystem (percentage)	29 Apr 2014 13:33:22	86.91	-	Graph
Free space of /tmp filesystem	29 Apr 2014 13:33:19	1.8 GB	-	Graph
Free space of /tmp filesystem (percentage)	29 Apr 2014 13:33:24	98.14	-	Graph

Another screenshot showing general information about a Virtual Machine:

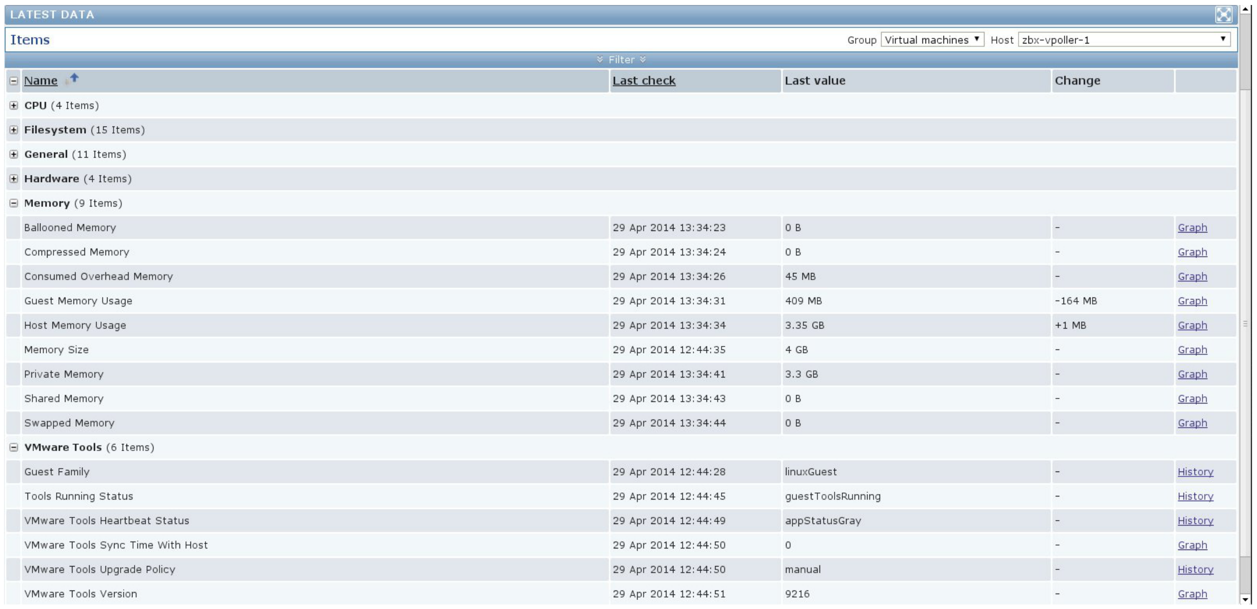


LATEST DATA

Items — Group: Virtual machines — Host: zbx-vpoller-1

Name	Last check	Last value	Change	
⊞ **CPU** (4 Items)				
⊞ **Filesystem** (15 Items)				
⊟ **General** (11 Items)				
Annotation	29 Apr 2014 12:44:22		-	History
Connection State	29 Apr 2014 13:34:25	connected	-	History
Guest Full Name	29 Apr 2014 12:44:29	Debian GNU/Linux 6 (64-bit)	-	History
Guest ID	29 Apr 2014 12:44:30	debian6_64Guest	-	History
Guest Name	29 Apr 2014 12:44:32	zbx-vpoller-1	-	History
Guest UUID	29 Apr 2014 12:44:33	42322814-ead0-fa99-4b2c-540af142d035	-	History
Overall Status	29 Apr 2014 13:34:39	green	-	History
Power State	29 Apr 2014 13:34:40	poweredOn	-	History
Running on hypervisor	29 Apr 2014 12:44:42	sof-2[redacted]	-	History
Uptime	29 Apr 2014 13:34:46	7 days, 01:32:57	+00:05:00	Graph
Virtual Hardware Version	29 Apr 2014 12:44:47	vmx-09	-	History
⊟ **Hardware** (4 Items)				
CPU Cores Per Socket	29 Apr 2014 12:44:27	2	-	Graph
Memory Size	29 Apr 2014 12:44:35	4 GB	-	Graph
Number of virtual CPUs	29 Apr 2014 12:44:36	4	-	Graph
Virtual Hardware Version	29 Apr 2014 12:44:47	vmx-09	-	History

Another screenshot showing information about the memory and VMware Tools for our Virtual Machine:

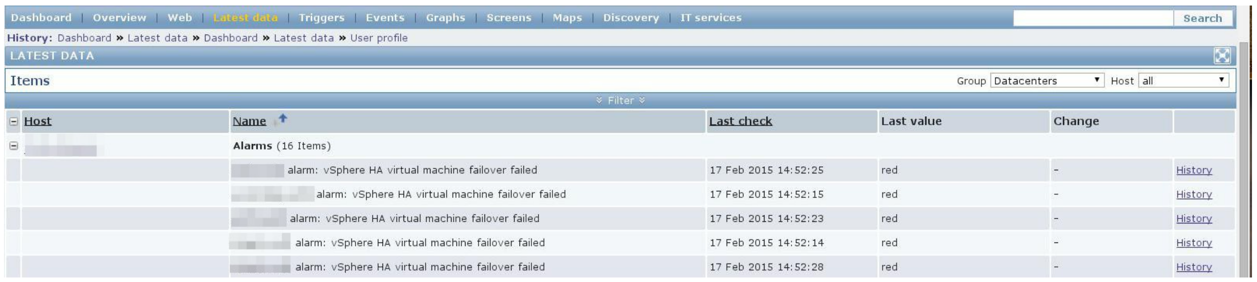


LATEST DATA

Items — Group: Virtual machines — Host: zbx-vpoller-1

Name	Last check	Last value	Change	
⊞ **CPU** (4 Items)				
⊞ **Filesystem** (15 Items)				
⊞ **General** (11 Items)				
⊞ **Hardware** (4 Items)				
⊟ **Memory** (9 Items)				
Ballooned Memory	29 Apr 2014 13:34:23	0 B	-	Graph
Compressed Memory	29 Apr 2014 13:34:24	0 B	-	Graph
Consumed Overhead Memory	29 Apr 2014 13:34:26	45 MB	-	Graph
Guest Memory Usage	29 Apr 2014 13:34:31	409 MB	-164 MB	Graph
Host Memory Usage	29 Apr 2014 13:34:34	3.35 GB	+1 MB	Graph
Memory Size	29 Apr 2014 12:44:35	4 GB	-	Graph
Private Memory	29 Apr 2014 13:34:41	3.3 GB	-	Graph
Shared Memory	29 Apr 2014 13:34:43	0 B	-	Graph
Swapped Memory	29 Apr 2014 13:34:44	0 B	-	Graph
⊟ **VMware Tools** (6 Items)				
Guest Family	29 Apr 2014 12:44:28	linuxGuest	-	History
Tools Running Status	29 Apr 2014 12:44:45	guestToolsRunning	-	History
VMware Tools Heartbeat Status	29 Apr 2014 12:44:49	appStatusGray	-	History
VMware Tools Sync Time With Host	29 Apr 2014 12:44:50	0	-	Graph
VMware Tools Upgrade Policy	29 Apr 2014 12:44:50	manual	-	History
VMware Tools Version	29 Apr 2014 12:44:51	9216	-	Graph

On the screenshot below you can see the discovered triggered alarms for one of our vSphere Datacenters:



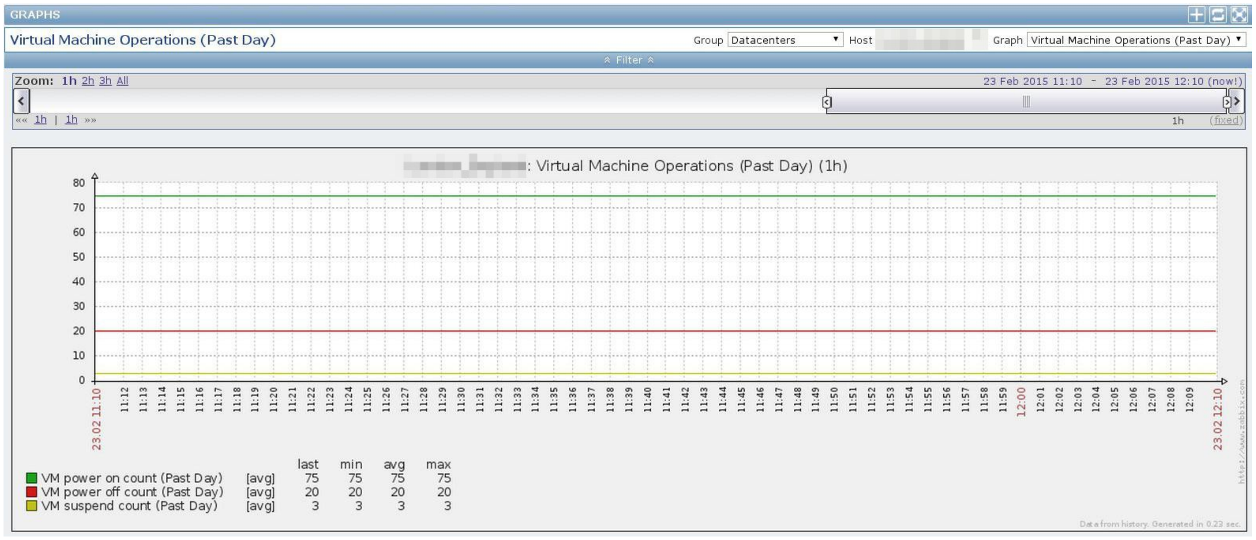
Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services
Search
History: Dashboard » Latest data » Dashboard » Latest data » User profile

LATEST DATA

Items — Group: Datacenters — Host: all

Host	Name	Last check	Last value	Change	
⊟ [redacted]	**Alarms** (16 Items)				
	[redacted] alarm: vSphere HA virtual machine failover failed	17 Feb 2015 14:52:25	red	-	History
	[redacted] alarm: vSphere HA virtual machine failover failed	17 Feb 2015 14:52:15	red	-	History
	[redacted] alarm: vSphere HA virtual machine failover failed	17 Feb 2015 14:52:23	red	-	History
	[redacted] alarm: vSphere HA virtual machine failover failed	17 Feb 2015 14:52:14	red	-	History
	[redacted] alarm: vSphere HA virtual machine failover failed	17 Feb 2015 14:52:28	red	-	History

The screenshot below shows the Virtual Machine operations for the past day for one of our VMware vSphere Datacenters:

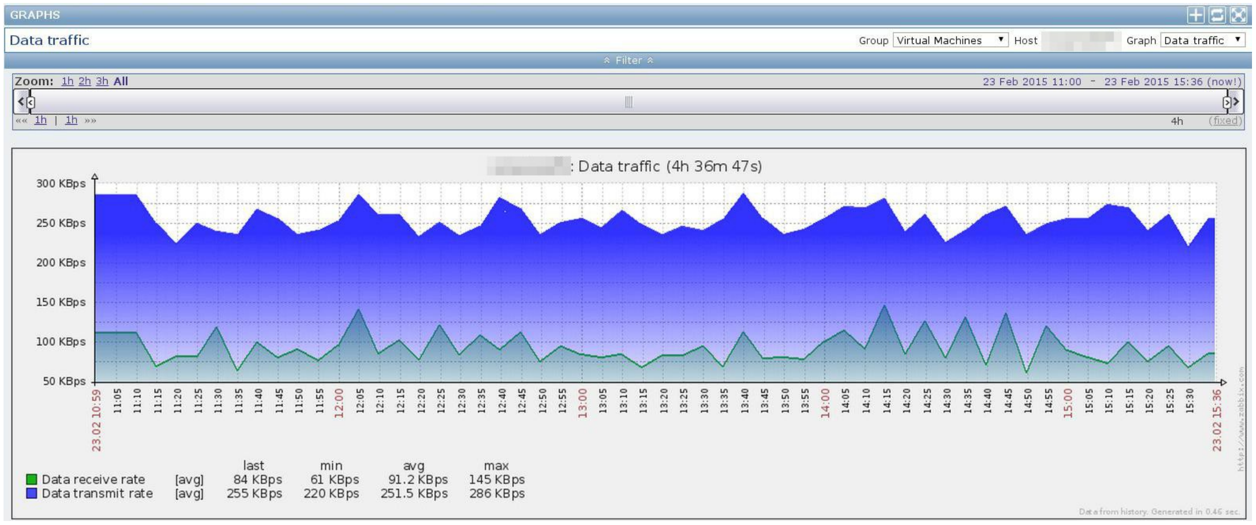

GRAPHS
Group Datacenters
Host
Graph Virtual Machine Operations (Past Day)
Filter
Zoom: 1h 2h 3h All
23 Feb 2015 11:10 - 23 Feb 2015 12:10 (now!)
1h
(fixed)

Virtual Machine Operations (Past Day)

: Virtual Machine Operations (Past Day) (1h)

		last	min	avg	max
🟩 VM power on count (Past Day)	[avg]	75	75	75	75
🟥 VM power off count (Past Day)	[avg]	20	20	20	20
🟨 VM suspend count (Past Day)	[avg]	3	3	3	3

From the screenshot below we can see the data traffic for one of our Virtual Machines.



Data traffic

: Data traffic (4h 36m 47s)

		last	min	avg	max
🟩 Data receive rate	[avg]	84 KBps	61 KBps	91.2 KBps	145 KBps
🟦 Data transmit rate	[avg]	255 KBps	220 KBps	251.5 KBps	286 KBps

4.8 Supported methods by vPoller

The table below lists the supported methods by vPoller along with description for each of them.

vPoller Method	Description
about	Get *about* information for a vSphere host
event.latest	Get the latest registered event from a vSphere host
session.get	Get the established vSphere sessions
perf.metric.info	Get info about all supported performance counters by the vSphere host
perf.interval.info	Get the existing performance historical intervals on the vSphere host
net.discover	Discover all vim.Network managed objects
net.get	Get properties of a vim.Network managed object
net.host.get	Get all HostSystems using a specific vim.Network
net.vm.get	Get all VirtualMachines using a specific vim.Network

Continued on next page

Table 1 – continued from previous page

vPoller Method	Description
datacenter.discover	Discover all vim.Datacenter managed objects
datacenter.get	Get properties of a vim.Datacenter managed object
datacenter.alarm.get	Get all alarms for a vim.Datacenter managed object
datacenter.perf.metric.info	Get the available performance counters for a vim.Datacenter object
datacenter.perf.metric.get	Retrieve performance metrics for a vim.Datacenter object
cluster.discover	Discover all vim.ClusterComputeResource managed objects
cluster.get	Get properties of a vim.ClusterComputeResource managed object
cluster.alarm.get	Get all alarms for a vim.ClusterComputeResource managed object
cluster.perf.metric.info	Get the available performance counters for a vim.ClusterComputeResource
cluster.perf.metric.get	Retrieve performance metrics for a vim.ClusterComputeResource object
resource.pool.discover	Discover all vim.ResourcePool managed objects
resource.pool.get	Get properties of a vim.ResourcePool managed object
host.discover	Discover all vim.HostSystem managed objects
host.get	Get properties of a vim.HostSystem managed object
host.alarm.get	Get all alarms for a vim.HostSystem managed object
host.cluster.get	Get the cluster a vim.HostSystem managed object
host.vm.get	Get all Virtual Machines registered on a vim.HostSystem
host.net.get	Get all Networks available for a specific vim.HostSystem
host.datastore.get	Get all datastores available to a vim.HostSystem
host.perf.metric.info	Get the available performance counters for a HostSystem object
host.perf.metric.get	Retrieve performance metrics for a vim.HostSystem object
vm.alarm.get	Get all alarms for a vim.VirtualMachine managed object
vm.discover	Discover all vim.VirtualMachine managed objects
vm.disk.discover	Discover all guest disks on a vim.VirtualMachine object
vm.guest.net.get	Discover all network adapters on a vim.VirtualMachine object
vm.net.get	Get all Networks used by a specific vim.VirtualMachine
vm.get	Get properties of a vim.VirtualMachine object
vm.datastore.get	Get all datastore used by a vim.VirtualMachine object
vm.disk.get	Get information about a guest disk for a vim.VirtualMachine object
vm.host.get	Get the HostSystem in which a specified vim.VirtualMachine is running on
vm.process.get	Get the running processes in a vim.VirtualMachine
vm.cpu.usage.percent	Get the CPU usage in percentage of a Virtual Machine
vm.perf.metric.info	Get the available performance counters for a VirtualMachine object
vm.perf.metric.get	Retrieve performance metrics for a vim.VirtualMachine object
vm.snapshot.get	Get all snapshots for a vim.VirtualMachine object
datastore.alarm.get	Get all alarms for a vim.Datastore managed object
datastore.discover	Discover all vim.Datastore objects
datastore.get	Get properties of a vim.Datastore object
datastore.host.get	Get all HostSystem objects using a specific datastore
datastore.vm.get	Get all VirtualMachine objects using a specific datastore
datastore.perf.metric.info	Get the available performance counters for a vim.Datastore object
datastore.perf.metric.get	Retrieve performance metrics for a vim.Datastore object
vsan.health.get	Get VSAN health state for a vim.HostSystem object

4.9 Terminology

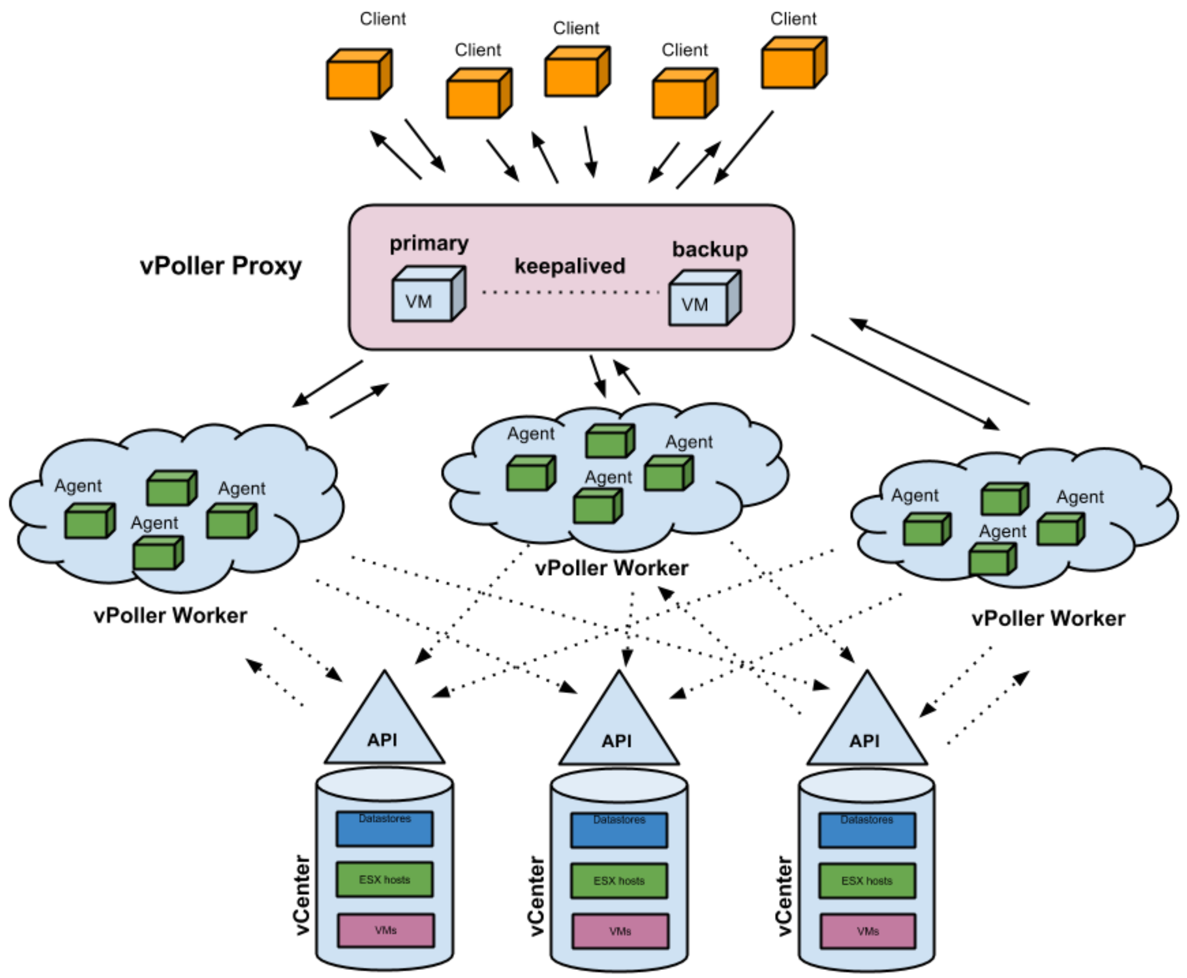
`vPoller Proxy` ZeroMQ proxy which distributes tasks and load balances client requests. The application running the `vPoller Proxy` is `vpoller-proxy`.

vPoller Worker Worker application which processes tasks, such as discovery and polling of vSphere object properties. The `vPoller Worker` receives new tasks for processing from the `backend` endpoint of a `vPoller Proxy`. The application running the `vPoller Worker` is `vpoller-worker`.

vPoller Client Client application used for sending task requests and receiving of results. The `vPoller Client` sends task requests to the `frontend` endpoint of a `vPoller Proxy`. The application running the `vPoller Client` is `vpoller-client` and `vpoller-cclient`, which is the client application written in C.

vSphere Agent The `vSphere Agents` are the ones that take care of establishing connections to the vSphere hosts and perform discovery and polling of vSphere objects. The `vSphere Agents` are running on the `vPoller Workers` and a single `vPoller Worker` can have as many `vSphere Agents` as you'd like. `vSphere Agents` are configured and managed by the `vconnector-cli` tool.

On the image below you can see how each vPoller component relates to the others.



Here is what happens when you send a client task request:

1. A `vPoller Client` sends a task request to the `frontend` endpoint of a `vPoller Proxy`.
2. Task request is received on the `vPoller Proxy` and is dispatched to any connected `vPoller Workers` on the `backend` endpoint.
3. The task request is received on the `vPoller Worker` and given to a `vSphere Agent` which is taking care of the requested vSphere host for processing the request through the VMware vSphere API.

4. The `vSphere Agent` returns any result from the operation to the `vPoller Worker` which in turn sends the result through the `vPoller Proxy` back to the client which requested the task.