metlog-psutils Documentation

Release 0.1

Victor Ng

May 03, 2013

CONTENTS

metlog-psutils is a plugin extension for *Metlog <http://github.com/mozilla-services/metlog-py>*. metlog-psutils provides details about network connections, memory usage, cpu usage and even some thread details.

This plugin works best on Linux. Running the plugin under OSX will skip some functionality and will require root privileges.

More information about how Mozilla Services is using Metlog (including what is being used for a router and what endpoints are in use / planning to be used) can be found on the relevant spec page.

CHAPTER

ONE

CONFIGURATION

Configuration is normally handled through Metlog's configuration system using INI configuration files. A psutil plugin must use the *metlog_psutils.psutil_plugin:config_plugin* as the provider of the plugin. The suffix of the configuration section name is used to set the method name on the Metlog client. Any part after *metlog_plugin_* will be used as the method name.

In the following example, we will bind a method *procinfo* into the Metlog client where we will allow network messages to be sent to the Metlog server.

```
[metlog_plugin_procinfo]
provider=metlog_psutils.psutil_plugin:config_plugin
net=True
```

Currently supported details options are:

- net - details for each network connection
- io - counters for bytes read, written and the # of syscalls
- cpu - CPU time used by user space and kernel
- mem - memory usage for RSS and VMS
- threads - CPU usage for user/system per thread

CHAPTER

TWO

USAGE

Obtaining a client can be done in multiple ways, please refer to the metlog documentation for complete details.

That said, if you are impatient you can obtain a client using *get_client*. We strongly suggest you do not do this though.

```
from metlog.holder import get_client
```

Logging your process details involves telling the plugin which details you would like to log. For each type of detail you would like to log, you must *explicitly* tell the logger that you would like that information. This is done to allow the suppression of log details through the configuration file.

Using the above example, the following snippet will log network details.

```
from metlog.holder import get_client
client = get_client('metlog_psutil')
client.procinfo(net=True)
```

The call to procinfo will send network details to the backend Metlog server. The transmission of those network details can be entirely suppressed through the configuration file. This is useful in cases where collecting data is not useful due to excessive logging or if the logs are simply not useful. An example with network messages disabled is illustrated below.

```
[metlog_plugin_procinfo]
provider=metlog_psutils.psutil_plugin:config_plugin
net=False
```

CHAPTER

THREE

DETAILED MESSAGE LAYOUT

The plugin will only send the process details that you explicitly ask for. These details are formatted as JSON blobs that are formatted such that a statsd message can be generated on the metlog server side. The following is an example that creates network, cpu, io, memory and thread statistics.

Each statsd messages is namespaced with *psutil.<group>.<hostname>.<pid>*

Note that hostnames have periods replaced with underscore characters so that statsd and graphite will properly namespace the messages.

Open sockets are represented with a two part key.

The key is comprised of:

- the host and port of the socket on the server side. Note that the IP address has periods converted into underscores.
- the TCP connection status

For the following examples, the hostname of the server is *MyHostName* and the process ID of the parent process is 9973.

3.1 Open sockets

An example of a server socket that is listening

```
{'ns': 'psutil.net.MyHostName.9973',
 'key': '127_0_0_1:50007.LISTEN',
 'rate': 1,
 'value': 1}
```

This will then get serialized into a statsd message in form of:

```
psutil.net.MyHostName.9973.127_0_0_1:50007.LISTEN|1|1
```

3.2 CPU time

CPU information for seconds in user space, seconds in kernel space, and percentage of CPU time used is represented as

```
{'ns': 'psutil.cpu.MyHostName.9973',
 'key': 'user',
 'rate': '',
```

```
 'value': 0.12 }

{'ns': 'psutil.cpu.MyHostName.9973',
 'key': 'sys',
 'rate': '',
 'value': 0.02 }

{'ns': 'psutil.cpu.MyHostName.9973',
 'key': 'pcnt',
 'rate': '',
 'value': 0.0 }
```

These are formatted into the following statsd messages:

```
psutil.cpu.MyHostName.9973.user|0.12
psutil.cpu.MyHostName.9973.sys|0.02
psutil.cpu.MyHostName.9973.pcnt|0.0
```

3.3 I/O counters

I/O metrics provide bytes read, written and the number of system calls used for read and write operations.

```
{'ns': 'psutil.io.MyHostName.9973',
 'key': 'read_bytes',
 'rate': '',
 'value': 50}

{'ns': 'psutil.io.MyHostName.9973',
 'key': 'write_bytes',
 'rate': '',
 'value': 200}

{'ns': 'psutil.io.MyHostName.9973',
 'key': 'read_count',
 'rate': '',
 'value': 3115}

{'ns': 'psutil.io.MyHostName.9973',
 'key': 'write_count',
 'rate': '',
 'value': 5434}
```

This will then get serialized into a statsd message in form of:

```
psutil.io.MyHostName.9973.read_bytes|50
psutil.io.MyHostName.9973.write_bytes|200
psutil.io.MyHostName.9973.write_count|3115
psutil.io.MyHostName.9973.write_bytes|5434
```

3.4 Memory Usage

Memory stats provide percentage of memory used as well as RSS and VMS usage.

```
{'ns': 'psutil.meminfo.MyHostName.9973',
 'key': 'pcnt',
 'rate': '',
 'value': 2.193876346582101}

{'ns': 'psutil.meminfo.MyHostName.9973',
 'key': 'rss',
 'rate': '',
 'value': 11415552}

{'ns': 'psutil.meminfo.MyHostName.9973',
 'key': 'vms',
 'rate': '',
 'value': 52461568}
```

This will then get serialized into a statsd message in form of:

```
psutil.meminfo.MyHostName.9973.pcnt|2.193876346582101
psutil.meminfo.MyHostName.9973.rss|11415552
psutil.meminfo.MyHostName.9973.vms|52461568
```

3.5 Thread level CPU usage

Thread level CPU usage adds the thread id as a prefix to the key. statsd is provided with CPU usage for user space and kernel space in seconds. The key is prefixed with the thread id so that statistics per thread per process can be monitored. In the following example, CPU stats for thread 17177 are monitored.

```
{'ns': 'psutil.thread.MyHostName.9973',
 'key': '17177.sys',
 'rate': '',
 'value': 0.02}

{'ns': 'psutil.thread.MyHostName.9973',
 'key': '17177.user',
 'rate': '',
 'value': 0.13}
```

This will then get serialized into a statsd message in form of:

```
psutil.thread.MyHostName.9973.17177.sys|0.02
psutil.thread.MyHostName.9973.17177.user|0.13
```

CHAPTER

FOUR

API DOCUMENTATION

class metlog_psutils.psutil_plugin.**LazyPSUtil**(*pid*, *server_addr=None*)

This class can only be used *outside* the process that is being inspected

get_busy_stats()

Get process busy stats.

Return 3 statsd messages for total_cpu time in seconds, total uptime in seconds, and the percentage of time the process has been active.

get_connections()

Return details of each network connection as a list of dictionaries.

Keys in each connection dictionary are:

local - host:port for the local side of the connection

remote - host:port of the remote side of the connection

status - TCP Connection status. One of :

- "ESTABLISHED"
- "SYN_SENT"
- "SYN_RECV"
- "FIN_WAIT1"
- "FIN_WAIT2"
- "TIME_WAIT"
- "CLOSE"
- "CLOSE_WAIT"
- "LAST_ACK"
- "LISTEN"
- "CLOSING"

get_cpu_info()

Return CPU usages in seconds split by system and user for the whole process. Also provides CPU % used for a 0.1 second interval.

Note that this method will *block* for 0.1 seconds.

get_io_counters()

Return the number of bytes read, written and the number of read and write syscalls that have invoked.

get_memory_info()
Return the percentage of physical memory used, RSS and VMS memory used

get_thread_cpuinfo()
Return CPU usages in seconds split by system and user on a per thread basis.

summarize_network(*network_data*)
Summarizes network connection information into something that is friendly to statsd.

From a metrics standpoint, we only really care about the number of connections in each state.

Connections are sorted into 2 buckets

- server connections

For each listening host:port, a dictionary of connection states to connection counts is created

metlog_psutils.psutil_plugin.**check_osx_perm**()
psutil can't do the right thing on OSX because of weird permissioning rules in Darwin.

http://code.google.com/p/psutil/issues/detail?id=108

metlog_psutils.psutil_plugin.**config_plugin**(*config*)
Configure the metlog plugin prior to binding it to the metlog client.

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

m

metlog_psutils.psutil_plugin, **??**