kwapi-g5k Documentation

Release 1.0.0

Clement Parisot

Sep 27, 2017

Contents

Kwapi is a framework designed for acquiring energy consumption and network metrics. It allows to import metrics from various sources and expose them in different ways.

Its architecture is based on a layer of drivers, which retrieve measurements from wattmeters or network switches, and a layer of plugins that collect and process them. The communication between these two layers goes through a bus. In the case of a distributed architecture, a plugin can listen to several drivers at remote locations.

Drivers and plugins are easily extensible to support other types of sources, and provide other services and metrics.

CHAPTER 1

What is the purpose of the project and vision for it?

Kwapi could be used to do:

- Energy monitoring of data centers
- Usage-based billing
- Efficient scheduling
- Network traffic visualisation
- Long-term storage of measurements

It aims at supporting various wattmeters and switches, being scalable and easily extensible.

This documentation offers information on how Kwapi works and how to contribute to the project.

CHAPTER 2

Table of contents

Installing

Installing Kwapi from source

1. Clone the Kwapi git repository to the management server:

```
$ git clone https://github.com/grid5000/kwapi-g5k.git
```

2. Data management use numpy, which cannot be installed vi pip. On Debian/Ubuntu, use:

   $ apt-get install python-numpy

3. Run the Kwapi installer and copy the configuration files:

```
$ pip install kwapi-g5k
$ cp -r kwapi-g5k/etc/kwapi /etc/
```

Installing Kwapi on Grid'5000

1. Create a VM on the site you want to monitor. You can create a domU with Xen or KVM. The command should be similar to this one:

```
$ xen-create-image --hostname=kwapi --ip=ip_address --dist jessie --role=udev
```

   You can find more informations about this procedure here: https://www.grid5000.fr/w/TechTeam:Puppet_4_admin

2. Start your VM:

```
$ xm create /etc/xen/kwapi.cfg
```

3. Install configuration tool on the VM. Puppet manifests and files are available (soon) in **grid5000-puppet** on INRIA Gitlab. You must install the right version of Puppet used in Grid'5000. Installation procedure can be find here: Puppet. After the certificate signing procedure, you should have a new Puppet node named **kwapi.site.grid5000.fr**.
4. Configure Kwapi with Puppet. You have to add additional classes on your new Puppet node. Use hiera YAML file to add **grid5000::kwapi** class on the node.
5. Test your new feature on the VM:

```
$ rake feature:test host=kwapi.site
```

6. Your VM is now configured with latest Grid'5000 version of Kwapi. You can connect on the node to check Kwapi service status.:

```
$ ssh kwapi.site.g5kadmin
$ sudo service kwapi status
```

Running Kwapi modules as daemon

Kwapi can be started, stoped, restarted with the service command:

```
$ sudo service kwapi start|stop|restart
```

This command will start kwapi as a daemon and run the modules indicated in `/etc/kwapi/daemon.conf` file.

Running Kwapi modules in foreground (debugging)

If you want to manage each Kwapi module individually (drivers and plugins), you can run the following commands.

- Start the drivers on all the configured machines:

```
$ kwapi-drivers
```

- Start the forwarder on this machine and a remote machine (optional):

```
$ kwapi-forwarder
```

- Start the API plugin if you want to access metrics with the API:

```
$ kwapi-api
```

- Start the RRD plugin if you want to store data as RRD (mandatory to display graphs in a web browser):

```
$ kwapi-rrd
```

- Start the HDF5 plugin if you want to store fine grained datas:

```
$ kwapi-hdf5
```

- Start Live plugin to active Web visualisation of your mesures:

```
$ kwapi-live
```

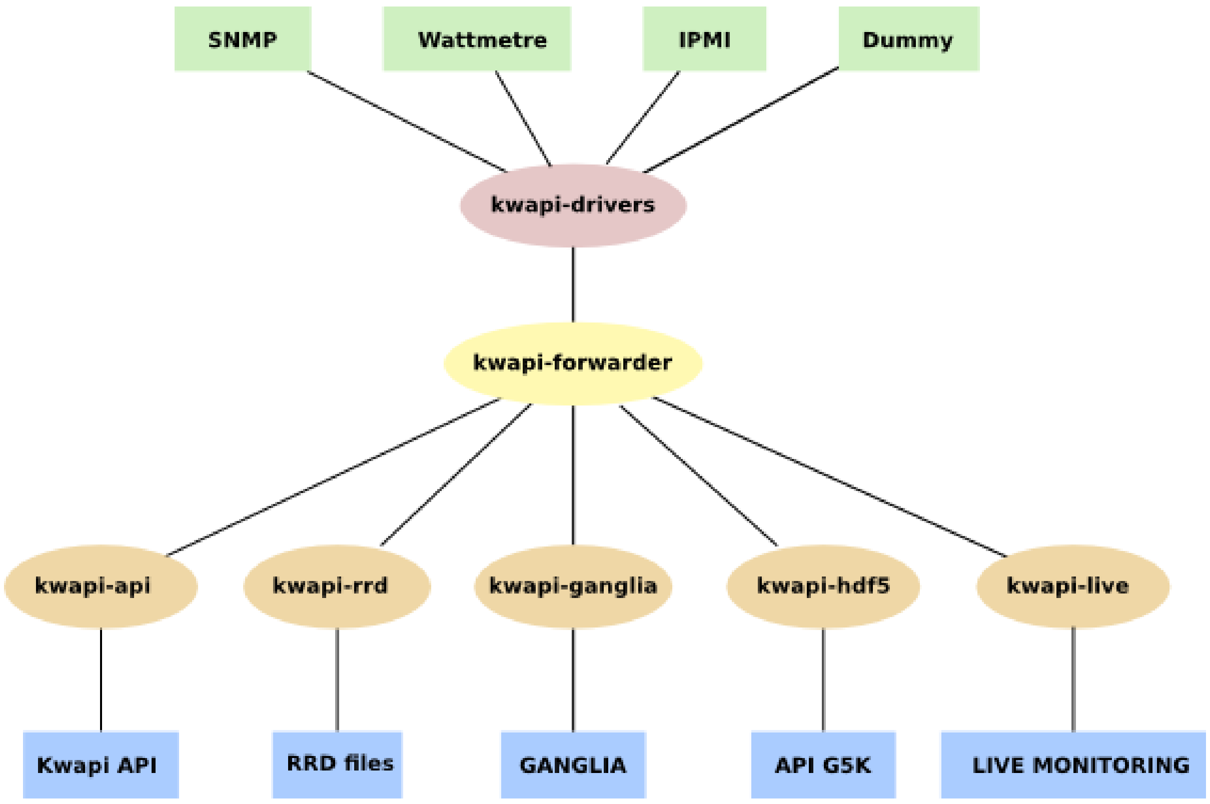
- Start Ganglia plugin to push data to the remote Ganglia server:

```
$ kwapi-ganglia
```

Warning: Don't forget to **stop** Kwapi daemon service before activating modules in foreground or it will result on conflict problems and data corruption !

System Architecture

Overview of the global layered architecture:



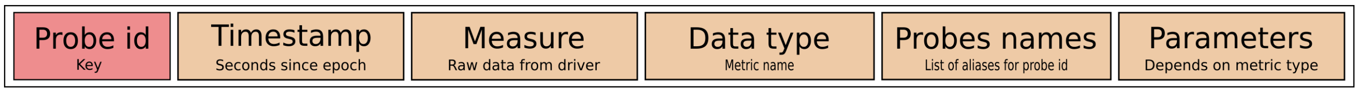
Kwapi drivers

Kwapi drivers are derived from a Driver superclass, itself derived from Thread. So drivers are threads. At least one driver thread is instantiated for each wattmeter or switch you want to monitor. Their constructors takes as arguments a list of probe IDs, probe names (an alias for probe ids) and kwargs (specific arguments).

Driver threads roles are:

1. Setting up a wattmeter or a switch.
2. Listening and decoding received data.
3. Calling a driver superclass method with measurements as argument. This method appends signature to the measurements, and publishes them on the bus.

Message format:



There are several types of drivers already implemented in Kwapi. You can of course implement your own driver.

SNMP drivers

Using the SNMP protocol and the right OIDs, you can retrieve a subtree of values corresponding, for a wattmeter, at the power consumption, for a switch, at the network traffic of his interfaces.

Of course, this driver works with any device that implementis SNMP protocol. So you can retrieve other metrics depending on what you want to monitor.

Wattmeters via SNMP

Kwapi supports different kinds of wattmeters (IPMI, Eaton PDU, Wattsup, etc). Wattmeters communicate via IP networks or serial links. Each wattmeter has one or more sensors (probes). Wattmeters send their values quite often (each second), and they are listen by wattmeter drivers.

Network drivers

Second type of drivers is the network drivers. Giving a switch address and the correct protocol, you can retrieve incoming and outgoing traffic. You have just to configure the right OIDs and allow SNMP requests on the switch. Every port of the given switch is scanned and the current counter value of the interface is assigned to the configured neighbor. This counter is a 64-bits number that correspond to the total number of octets received on the interface, including framing characters.

Dummy drivers

Added as a testing feature, dummy drivers just send the value you ask them to send. They are used to simulate a probe. You can configure them as you wish. Kwapi implements dummynet and dummywatt drivers to simulate network and wattmeter drivers.

IPMI drivers

IPMI drivers use the command line *ipmitool* to retrieve information from IPMI sensors.

JSON url drivers

This driver can be used to get informations directly from JSON structured text of Grid‘5000 API. It can be usefull when you want to import in Kwapi remote metrics from the Metrologie API.

Driver manager

The driver manager is the loader and the checker of driver threads. It loads all drivers according the configuration file, and checks regularly that driver threads are alive. In case of crash, the event is logged and the driver thread is reloaded. We can imagine that a driver will crash if a technician unplug a wattmeter, for example.

Bus

Currently, the internal Kwapi bus is ZeroMQ. Publishers are driver threads, and subscribers are plugins.

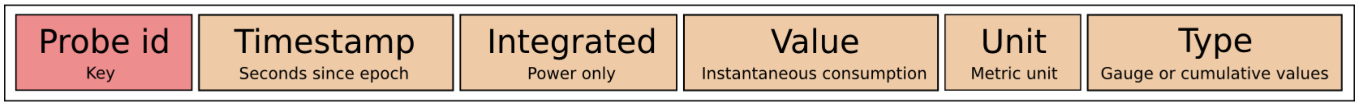
Kwapi plugins

Kwapi API plugin

Kwapi API offers a REST API. This API is linked in Grid‘5000 API and adopts its standard to expose live measures to the users. Such data can then be imported in experiments by just sending a request to the API. This plugin contains a collector that computes kWh for power, interface counters for network and an API based on Flask.

Collector

The collector stores these values for each probe:



Fields:

- Probe id: could be the hostname of the monitored machine. But it is a bit more complicated because a probe can monitor several machines (PDU).
- Timestamp: is updated when a new value is received.
- Integrated (power only): is computed by taking into account the new value in watt, and the elapsed time since the previous update.
- Value: offers the possibility to know instantaneous consumption or traffic of a device, without having to query two times a probe in a small interval to deduce it. This could be especially useful if a probe has a large refresh interval: there is no need to wait its next value.
- Unit: metric unit. For example ‘W’ stands for *watt* in power API.
- Type: metric type can be ‘Gauge’ or ‘Cumulative’. It indicates if measures are retrivied as counter or not and if an integrated value can be calculated

No history is kept by this plugin. Storage is offered with other plugins. The collector is cleaned periodically to prevent a deleted probe from being stored indefinitely in the collector. So when a probe has not been updated for a long time, it is deleted.

API

Verb	URL	Parameters	Expected result
GET	/probe-ids/		Returns all known probe IDs.
GET	/probes/		Returns all information about all known probes.
GET	/probes/<probe>/	probe id	Returns all information about this probe (id, timestamp, value, type, integrated).
GET	/probes/<probe>/<metric>/	probe id metric {energy, network_in, network_out}	Returns the probe’s metric value.

Probe id *must be* in the format **<site>.<probe>** where **site** is the site on which Kwapi is running (could be retrieve with the second field of hostname command), and probe is the probe id. You can also use a probe name instead of a probe id.

Kwapi RRD plugin

It stores information from the drivers directly in RRD files. The advantage of such files is that they permits to render efficiently Graphs with various scales. The size of the databases are constant. One database per probe and per metric is created. By default, RRD files are stored in */var/lib/kwapi/kwapi-rrd*.

This plugin create and update automatically the RRD files, depending on the values he receives from the drivers.

As each plugin, he needs:

- Probe id: probe identifier (could be different than probe name)
- Timestamp: time of the measure, given by the driver, unix format timestamp
- Measure: measure
- Data type: metric name
- Probes names: list of aliases that will be mapped to this probe id
- Parameters: other informations about the metrics

Kwapi Live plugin

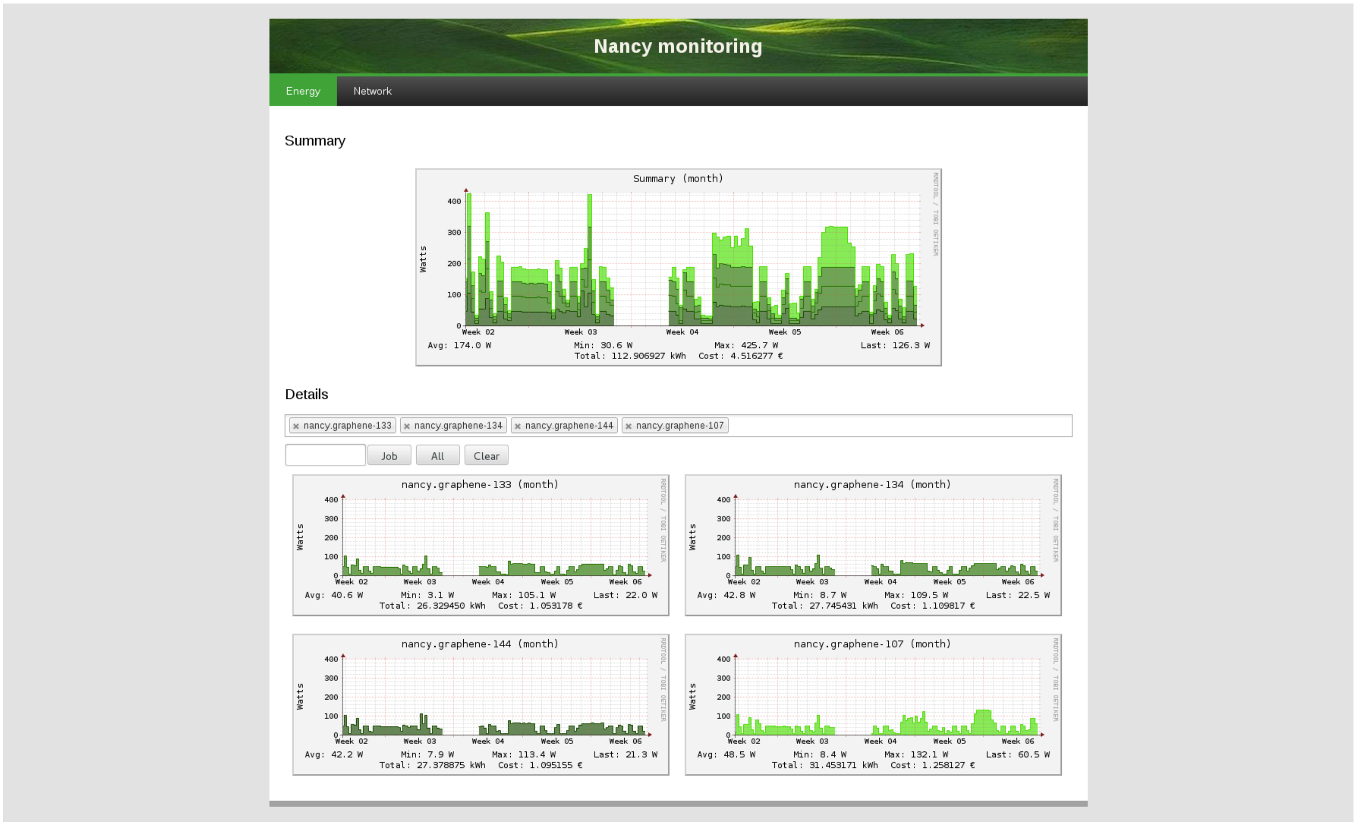
Web interface

The visualization plugin provides a web interface with power consumption and network traffic graphs. It is based on Flask and RRDtool.

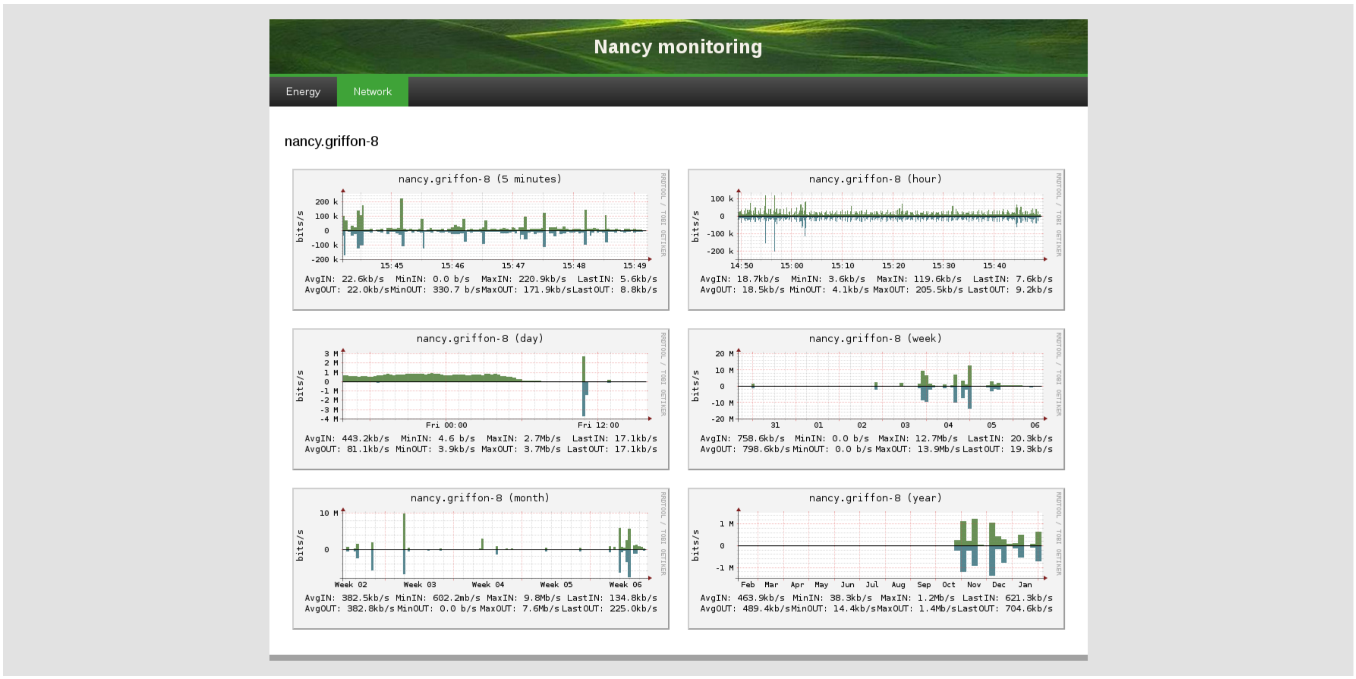
Verb	URL	Parameters	Expected result
GET	/<metric>/last/<period>/	metric { energy, network } period { minute, hour, day, week, month, year }	Returns a webpage with a summary graph and all probe graphs.
GET	/<metric>/probe/<probe>/	metric { energy, network } probe id	Returns a webpage with all graphs about this probe (all periods).
GET	/<metric>/summary-graph/<start>/<end>	metric { energy, network } start timestamp end timestamp	Return a summary graph of the metric evolution about this period
GET	/<metric>/graph/<probe>/<start>/<end>	metric { energy, network } probe id start timestamp end timestamp	Returns a graph about this probe.
POST	/zip/[probes]	probe id or list of probe ids	Returns a zip file containing all data for in RRD format and all graphs for each scale.
GET	/nodes/<job_id>/<metric>	job_id: Grid‘5000 OAR job ID metric { energy, networks }	Returns list of probe ids for the specified job.

Probe id *must be* in the format **<site>.<probe>** where **site** is the site on which Kwapi is running (could be retrieve with the second field of hostname command), and probe is the probe id. You can also use a probe name instead of a probe id.

Webpage with a summary graph and all probe graphs:


Nancy monitoring
Energy
Network
Summary
Summary (month)
Avg: 174.0 W Min: 30.6 W Max: 425.7 W Last: 126.3 W
Total: 112.906927 kWh Cost: 4.516277 €
Details
nancy.graphene-133
nancy.graphene-134
nancy.graphene-144
nancy.graphene-107
Job
All
Clear
nancy.graphene-133 (month)
Avg: 40.6 W Min: 3.1 W Max: 105.1 W Last: 22.0 W
Total: 26.329450 kWh Cost: 1.053178 €
nancy.graphene-134 (month)
Avg: 42.8 W Min: 8.7 W Max: 109.5 W Last: 22.5 W
Total: 27.745431 kWh Cost: 1.109817 €
nancy.graphene-144 (month)
Avg: 42.2 W Min: 7.9 W Max: 113.4 W Last: 21.3 W
Total: 27.378875 kWh Cost: 1.095155 €
nancy.graphene-107 (month)
Avg: 48.5 W Min: 8.4 W Max: 132.1 W Last: 60.5 W
Total: 31.453171 kWh Cost: 1.258127 €

Webpage with scales summaries of a probe:


Nancy monitoring
Energy
Network
nancy.griffon-8
nancy.griffon-8 (5 minutes)
AvgIN: 22.6kb/s MinIN: 0.0 b/s MaxIN: 220.9kb/s LastIN: 5.6kb/s
AvgOUT: 22.0kb/s MinOUT: 330.7 b/s MaxOUT: 171.9kb/s LastOUT: 8.8kb/s
nancy.griffon-8 (hour)
AvgIN: 18.7kb/s MinIN: 3.6kb/s MaxIN: 119.6kb/s LastIN: 7.6kb/s
AvgOUT: 18.5kb/s MinOUT: 4.1kb/s MaxOUT: 205.5kb/s LastOUT: 9.2kb/s
nancy.griffon-8 (day)
AvgIN: 443.2kb/s MinIN: 4.6 b/s MaxIN: 2.7Mb/s LastIN: 17.1kb/s
AvgOUT: 81.1kb/s MinOUT: 3.9kb/s MaxOUT: 3.7Mb/s LastOUT: 17.1kb/s
nancy.griffon-8 (week)
AvgIN: 758.6kb/s MinIN: 0.0 b/s MaxIN: 12.7Mb/s LastIN: 20.3kb/s
AvgOUT: 798.6kb/s MinOUT: 0.0 b/s MaxOUT: 13.9Mb/s LastOUT: 19.3kb/s
nancy.griffon-8 (month)
AvgIN: 382.5kb/s MinIN: 602.2mb/s MaxIN: 9.8Mb/s LastIN: 134.8kb/s
AvgOUT: 382.8kb/s MinOUT: 0.0 b/s MaxOUT: 7.6Mb/s LastOUT: 225.0kb/s
nancy.griffon-8 (year)
AvgIN: 463.9kb/s MinIN: 38.3kb/s MaxIN: 1.2Mb/s LastIN: 621.3kb/s
AvgOUT: 489.4kb/s MinOUT: 14.4kb/s MaxOUT: 1.4Mb/s LastOUT: 704.6kb/s

In the menu bar, you can choose the period the metric you want to display. For each metric you can select a timescale (last minutes, hour, day, week, month or year). By clicking on a probe, you can display all graphs available for this probe, with different resolutions.

You can select several probes and display a stacked summary of their consumption. Use the job field to automatically monitor probes corresponding to your job (select the correct probes and adapt timescale)

You can export all RRD data and PNG graphs with the *Download all probes RRD* link or just the selected probe graphs with the *Download selected probes RRD* link.

Graphs

The summary graph shows the total measurements for the selected metric (sum of all the probes). Each colour corresponds to a probe.

The energy legend contains:

- Minimum, maximum, average and last measures.
- Integrated measure (energy consumed (kWh) or network traffic (Kb/s)).
- Cost if any.

The networl legend contains:

- Minimum, maximum, average and last measures for IN and OUT traffic
- IN network bandwith in **green** and upside of the graph
- OUT network bandwith in **blue** and downside of the graph

File sizes:

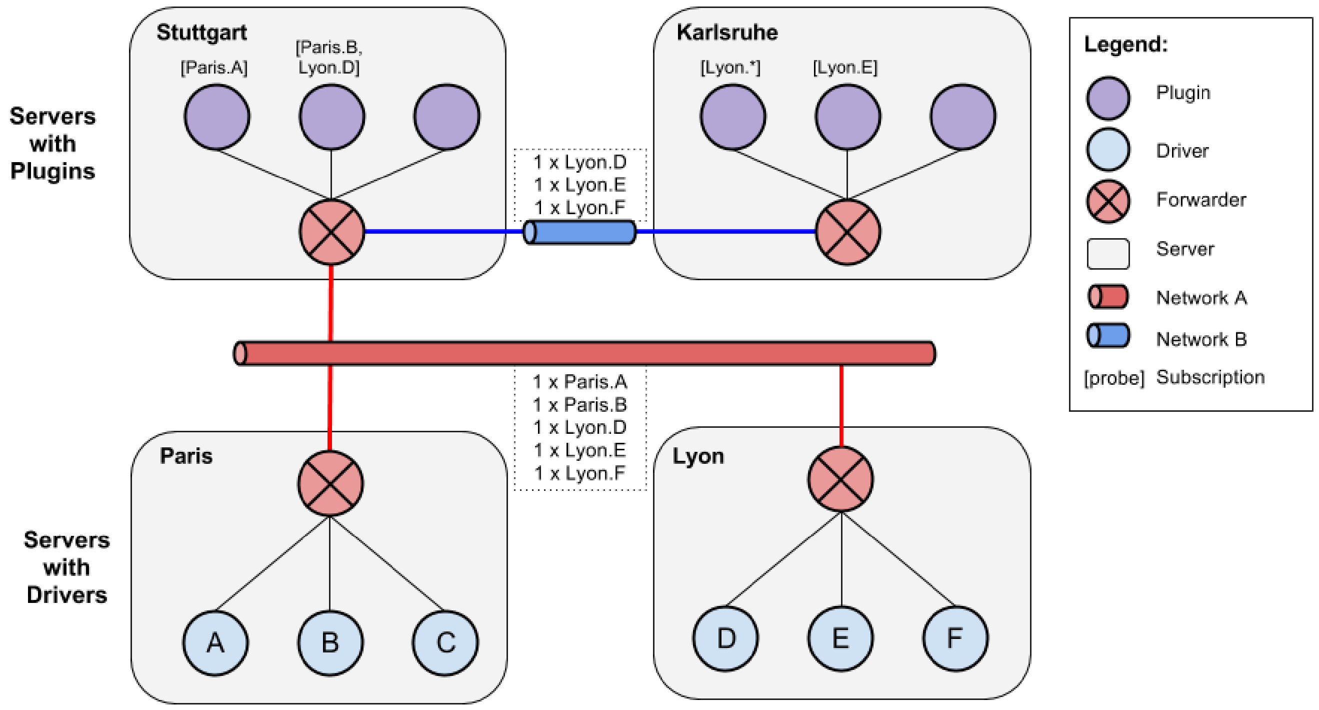
- RRD file: 10 Ko.
- Probe graph: 12 Ko.
- Summary graph: 24 Ko.

A cache mechanism prevents graphs from being rebuilt uselessly.

Kwapi forwarder

The forwarder aims at decreasing the network traffic: if multiple plugins listen the same probe, the metric is sent once on the network, and the forwarder duplicate it and sends a copy to each listeners. The forwarder can also be installed on a gateway machine, in order to connect isolated networks.

The following diagram shows these two features:



Using the forwarder is optional, and the plugins can be configured to subscribe directly to the drivers. Direct subscribing without using the forwarder is recommanded if the drivers and the plugins are running on the same machine.

Kwapi HDF5

Kwapi HDF5 is used to store fine grained metric with Kwapi. Each measure returned by the drivers are stored in an HDF5 file on the server. The main advantages of this database are: * Very large datasets: store several months of power consumption of numerous probes * Fast access * Hierarchical store: data can be groupped by site or cluster * Parallel writing * Compressed file for low storage cost * Heterogeneous data support

You can configure the split period of your HDF5 files in the configuration file (1 file per month or less) depending on how much data you want to save.

Collector

The HDF5 Collector is composed of one Writter by metric with their proper buffers and a single queue per metric where all the

- The update function put the new received value in the queue that correspond to his metric.
- Each Collector iterate on his corresponding queue and for each new measure, writes an entry in his internal buffer
- When a Collector buffer is full, it writes his values to the database on the disk depending on the current date
- If the plugin is stopped, a **STOP** flag his added in all the queues
- When the Collector receive a **STOP** flag, buffers are flushed on the disk and Collector exits

A *chunk_size* parameter. In order to limit the stress of writing data to the disk each seconds, we introduced a *chunk_size* parameters that ensure that each write to the disk (flush of HDF5 file) is done only when this amount of data is reached for a metric type.

Warning: When you stop HDF5 plugin, a final FLUSH is done to write last data to the disk. **Be sure to wait to the end of this final step or it might results in corrupted data!**

API

REST API permits to retrieve measures from those databases. Unlike RRD database, HDFStore store raw measures and datas are not alterated. API is very similar to Kwapi-API. It format is very Grid‘5000 specific because it was developped with the aim to be include to Grid‘5000 Metric API.

Verb	URL	Parameters	Expected result
GET	/		Returns all known metrics.
GET	/<metric>/	metric { energy, network_in, network_out }	Returns all known probe IDs for the metric.
POST	/<metric>/timeseries/[job-idlprobes]	metric { energy, network } job_id probes: list of probes	Return all data for the selected probes and selected range. Selection is made with job_id or probe name given.

Range of data is selected with the *from* and *to* parameters. Resolution by default is 1 and can’t be changed.

Kwapi Ganglia

This plugin his pretty simple. You first have to run a Ganglia server somewhere. It have to accept data from the remote Kwapi server. For this, edit the configuration file according to Ganglia Documentation. Check for the Ganglia **multicast address**.

The single parameter of this plugin is the *ganglia_server* address. Edit this field in the configuration file to point your remote Ganglia server. All data received from your drivers will be sended to the server. Actual configuration just send single power probes consumption to Ganglia.

Metric	Remote name	Parameters
power	pdu	• units: Watts • type: uint16 • value: int(metrics) • hostname: ip:hostname (ex: 192.168.1.1:griffon-54.nancy.grid5000.fr) • spoof: True
network_in	None	
network_out	None	

You can modiy this parameters in the config file. Float values for metrics will be supported.

Configuration Options

Kwapi drivers specific

The following table lists the Kwapi drivers specific options in the drivers configuration file. For information we are listing the configuration elements that we use after the Kwapi drivers specific elements.

Parameter	Default	Note
probes_endpoint	ipc:///tmp/kwapi-drivers	Endpoint where the drivers send their measurements ipc://<file> or tcp://<host>:<port>
enable_signing	true	Enable message signing between drivers and plugins
metering_secret	change this or be hacked	Secret value for signing metering messages
check_drivers_interval	60	Check drivers at the specified interval and restart them if they are crashed

The configuration file contains a section for each wattmeter.

A sample configuration file can be found in drivers.conf.

Kwapi plugin API specific

The following table lists the Kwapi API specific options in the API configuration file. For information we are listing the configuration elements that we use after the Kwapi API specific elements.

Parameter	Default	Note
api_port	5000	API port
probes_endpoint	ipc:///tmp/kwapi-forwarder	Endpoint where the measurements are received
signature_checking	true	Enable the verification of signed metering messages
driver_metering_secret	change this or be hacked	Secret value for verifying signed metering messages
cleaning_interval	300	Delete the probes that have not been updated during the specified interval

A sample configuration file can be found in api.conf.

Kwapi plugin RRD specific

The following table lists the Kwapi RRD specific options in the RRD configuration file. For information we are listing the configuration elements that we use after the Kwapi RRD specific elements.

Parameter	Default	Note
probes_endpoint	ipc:///tmp/kwapi-forwarder	Endpoint where the measurements are received
signature_checking	true	Enable the verification of signed metering messages
driver_metering_secret	change this or be hacked	Secret value for verifying signed metering messages
rrd_dir	/var/lib/kwapi/kwapi-rrd	The directory where are stored RRD files

A sample configuration file can be found in rrd.conf.

Kwapi plugin Live specific

The following table lists the Kwapi Live specific options in the Live configuration file. For information we are listing the configuration elements that we use after the Kwapi Live specific elements.

Parameter	Default	Note
live_port	8080	Port used to display webpages
probes_endpoint	ipc:///tmp/kwapi-forwarder	Endpoint where the measurements are received
signature_checking	true	Enable the verification of signed metering messages
driver_metering_secret	change this or be hacked	Secret value for verifying signed metering messages
png_dir	/var/lib/kwapi/kwapi-png	The directory where are stored PNG files
rrd_dir	/var/lib/kwapi/kwapi-rrd	The directory where are stored RRD files
currency	€	The currency symbol used in graphs
kwh_price	0.125	The kWh price used in graphs
hue	100	The hue of the graphs
max_watts	400	The maximum value of the summary graph
refresh_interval	5	The webpage auto-refresh interval

A sample configuration file can be found in live.conf.

Warning: Be sure that *rrd_dir* directory is the same in RRD Plugin and Live plugin

Kwapi plugin Ganglia specific

The following table lists the Kwapi Ganglia specific options in the Ganglia configuration file. For information we are listing the configuration elements that we use after the Kwapi API specific elements.

Parameter	Default	Note
ganglia_server	udp://239.2.11.71:8649	Ganglia server address
probes_endpoint	ipc:///tmp/kwapi-forwarder	Endpoint where the measurements are received
signature_checking	true	Enable the verification of signed metering messages
driver_metering_secret	change this or be hacked	Secret value for verifying signed metering messages

A sample configuration file can be found in ganglia.conf.

General options

The following is the list of options that we use:

Parameter	Default	Note
log_file		Log output to a named file
verbose	true	Print more verbose output

Kwapi forwarder specific

The following table lists the Kwapi forwarder specific options in the forwarder configuration file. For information we are listing the configuration elements that we use after the Kwapi forwarder specific elements.

Parameter	Default	Note
forwarder_endpoint	ipc:///tmp/kwapi-forwarder	Endpoint where the measurements are forwarded and where the plugins subscriptions are received
probes_endpoint	ipc:///tmp/kwapi-drivers	Endpoint where the drivers send their measurements. ipc://<file> or tcp://<host>:<port>

A sample configuration file can be found in forwarder.conf.

Kwapi Daemon specific

The following table lists the Kwapi service specific options in the daemon configuration file.

Set a parameter to **false** will not start the corresponding plugin/driver when you start the service.

Warning: Always run *service kwapi stop* **BEFORE** modifying any of the following parameters !

Parameter	Default	Note
KWAPI_DRIVERS	true	Start Kwapi drivers in kwapi service
KWAPI_FORWARDER	true	Start Kwapi forwarder in kwapi service
KWAPI_API	true	Start Kwapi api in kwapi service
KWAPI_RRD	true	Start Kwapi rrd in kwapi service
KWAPI_HDF5	true	Start Kwapi hdf5 in kwapi service
KWAPI_LIVE	true	Start Kwapi live in kwapi service
KWAPI_GANGLIA	true	Start Kwapi ganglia in kwapi service

A sample configuration file can be found in daemon.conf.

Usage

Install Kwapi on your site

See *Installing Kwapi from source* to know how to install Kwapi.

Configuration

Warning: Configuration files are only read once when the plugin/driver is started. You have to restart the plugin to load a new configuration.

Configure the drivers

See the *Configuration* section for specific information on each driver configuration. If you are on Grid‘5000 you can use the specific *kwapi-g5k-conf* tool to generate the configuration from the Grid‘5000 API.

Configure the plugins

Some plugins have specific options that you can configure (the size of HDF5 files for example). You can retrieve those options in the specific *<plugin_name>.conf* of Kwapi.

Choose which drivers/plugins to start

By using the *daemon.conf* configuration file, you can specify which driver you want to run. By default (empty file), no Kwapi driver or plugin will run at all. They will be launch in the order defined in the file.

Launch Kwapi

Start the system service to see Kwapi in action. You can check Kwapi status with the system command like an usual service *service kwapi status*.

Debug Kwapi execution

By default, all logs go to */var/log/kwapi/*. The file *kwapi.log* contains errors on kwapi service or errors that are not catch in drivers or plugins. There is 1 log file per driver/service + 1 log file for dedicated scripts (kwapi-g5k-conf and kwapi-g5k-check).

- HDF5 are quite big. Be sure to have sufficient space for them.
- Check that user kwapi have access to databases and log files under */var/lib/kwapi* and */var/log/kwapi*.
- A chunk size has been defined to limit stress of disk access introduced by HDF5 plugin. You can tune its value if you have problems with disk access.

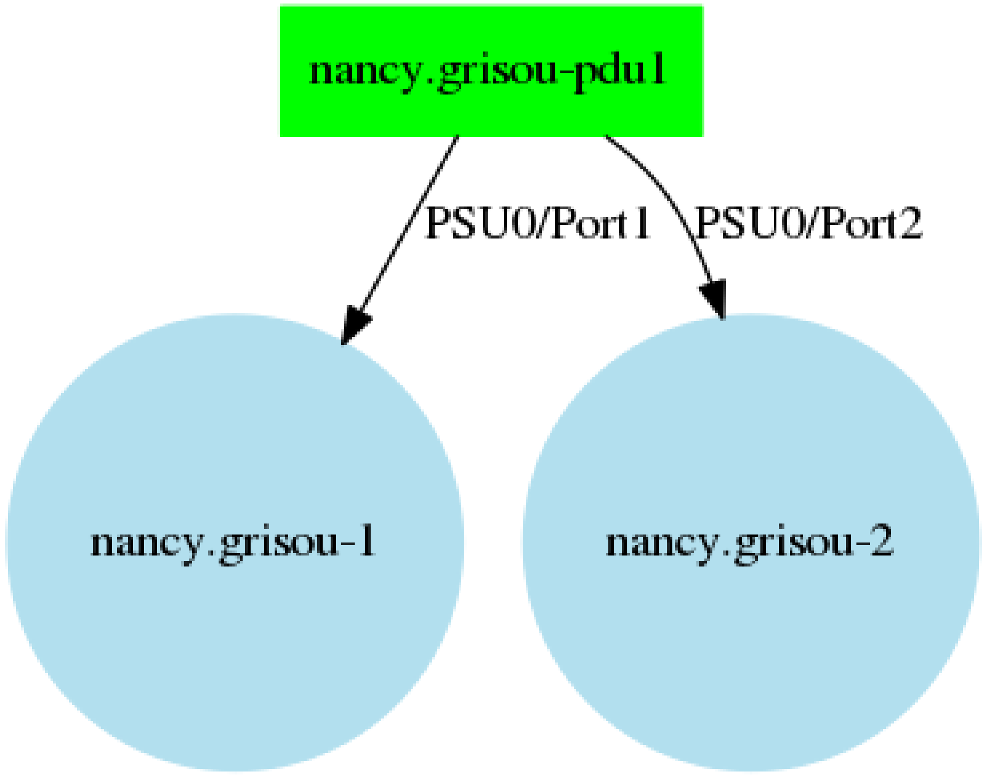
Exemples

Power configuration examples

Here you can find multiple examples of driver configuration for Kwapi

1 PDU 2 nodes

2 nodes connected to 1 PDU.

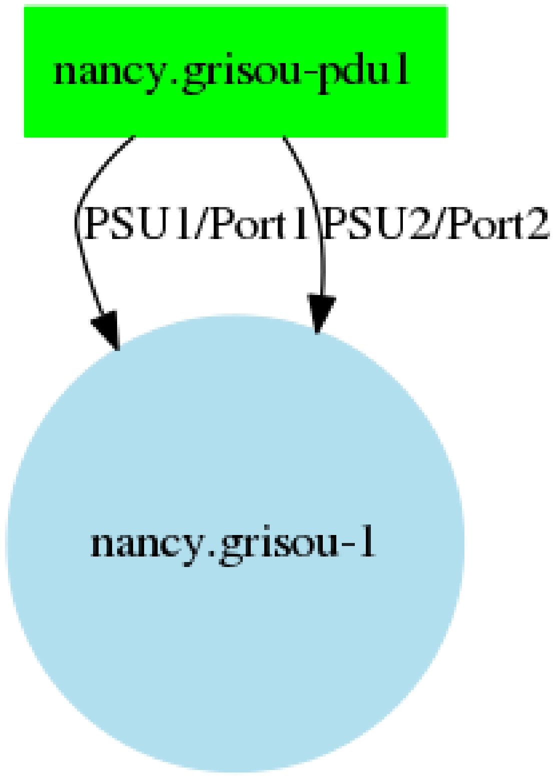


Listing 2.1: drivers.conf

```
[grisou-pdu1]
probes = ['nancy.grisou-pdu1.1', 'nancy.grisou-pdu1.2' ]
probes_names = ['nancy.grisou-1', 'nancy.grisou-2']
data_type = {'name':'power', 'type':'Gauge', 'unit':'W'}
```

1 PDU, 1 node, multiple PSU

1 Node with 2 PSU (Power Supply Unit) connected to 1 PDU.

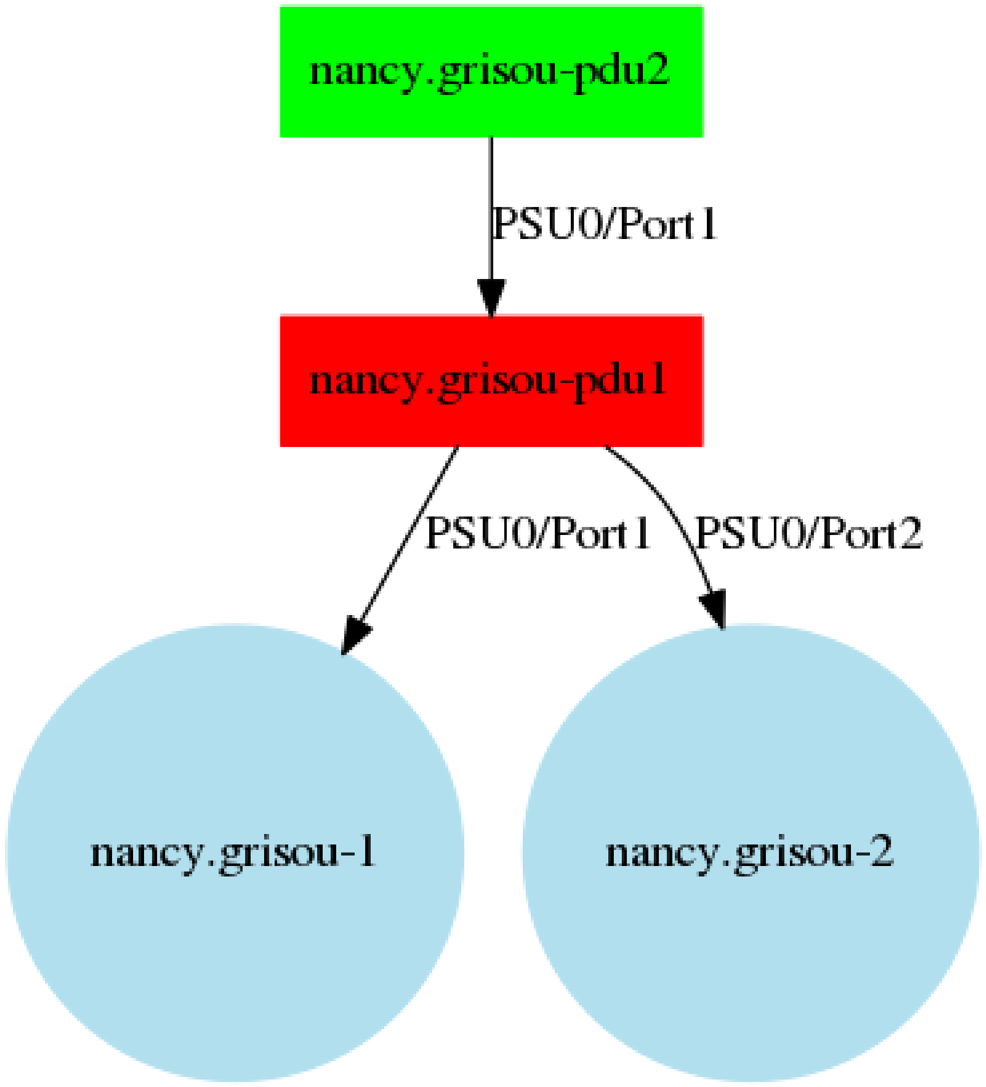


Listing 2.2: drivers.conf

```
[grisou-pdu1]
probes = ['nancy.grisou-pdu1.1', 'nancy.grisou-pdu1.2' ]
probes_names = ['nancy.grisou-1', 'nancy.grisou-1']
data_type = {'name':'power', 'type':'Gauge', 'unit':'W'}
```

1 PDU, shared measure, 2 nodes

2 nodes connected to a non-monitorable PDU. The non-monitorable PDU in red is connected to a monitorable PDU. It is equivalent to a shared consumption measure for the 2 nodes.



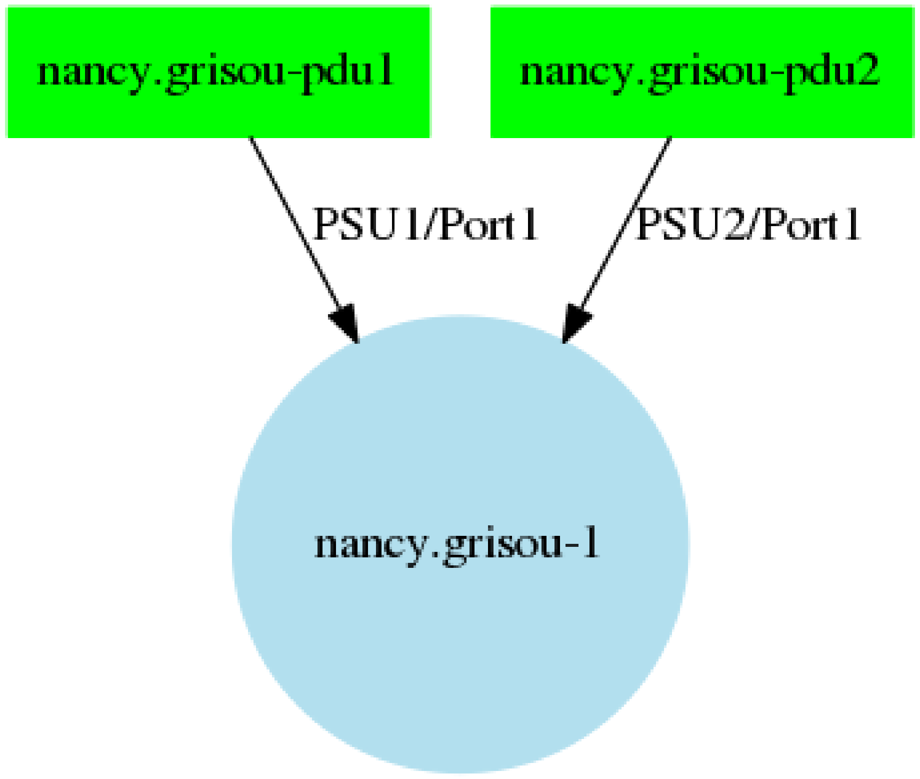
Listing 2.3: drivers.conf

```
# No info on pdu1
[grisou-pdu2]
probes = ['nancy.grisou-pdu2.1', ]
probes_names = [['nancy.grisou-1', 'nancy.grisou-2']]
```

```
data_type = {'name':'power', 'type':'Gauge', 'unit':'W'}
```

2 PDU, 1 node, 2 PSU

1 node with 2 PSU connected to 2 different PDU.



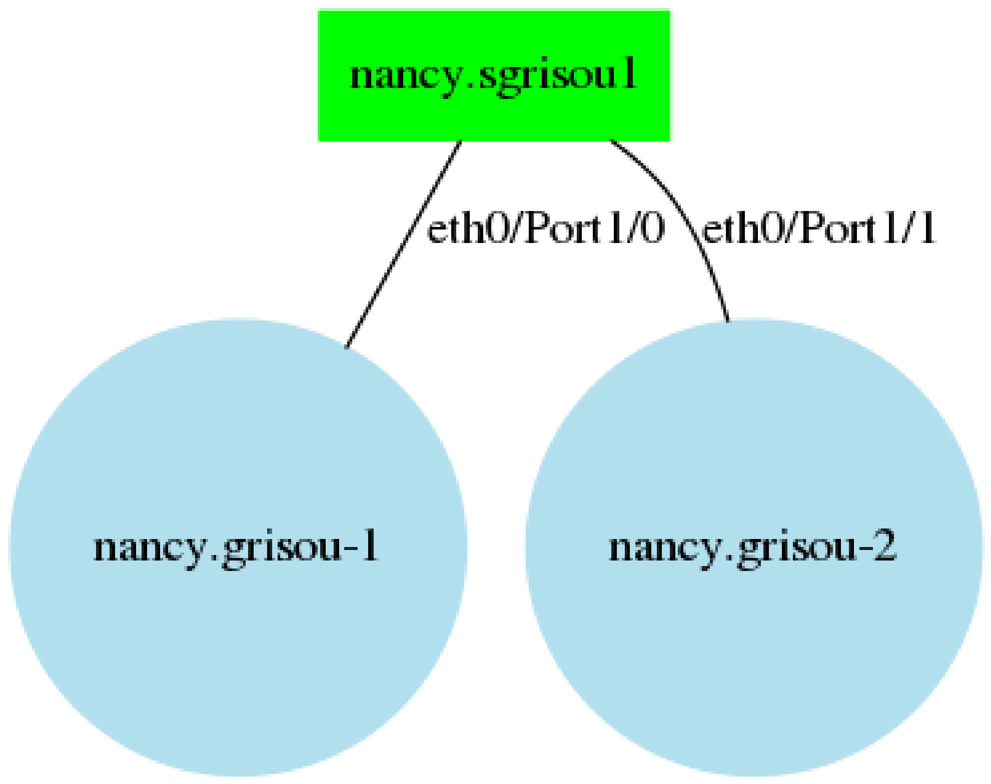
Listing 2.4: drivers.conf

```
[grisou-pdu1]
probes = ['nancy.grisou-pdu1.1', ]
probes_names = [['nancy.grisou-1']]
data_type = {'name':'power', 'type':'Gauge', 'unit':'W'}

[grisou-pdu2]
probes = ['nancy.grisou-pdu2.1', ]
probes_names = [['nancy.grisou-1']]
data_type = {'name':'power', 'type':'Gauge', 'unit':'W'}
```

Network configuration examples

1 Switch 2 nodes

2 nodes connected to 1 switch.

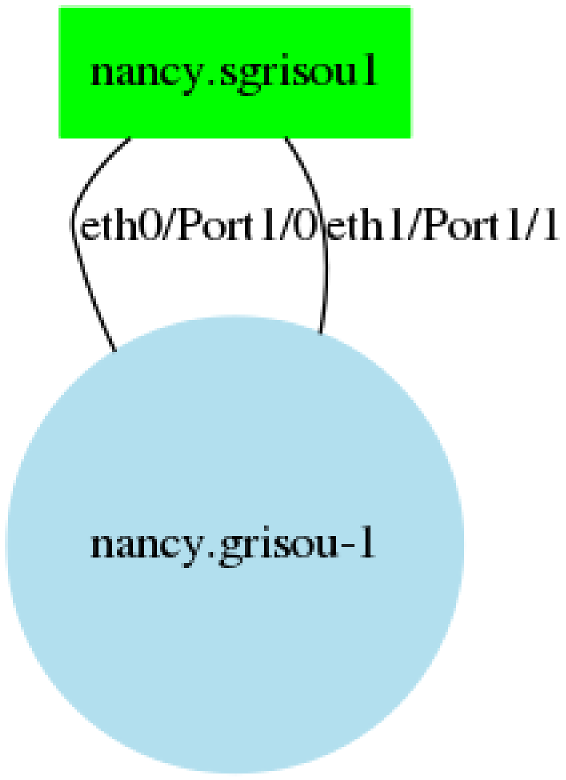


Listing 2.5: drivers.conf

```
[sgrisou1-IN]
probes = ['nancy.sgrisou1.1-0', 'nancy.sgrisou1.1-1']
probes_names = ['nancy.grisou-1', 'nancy.grisou-2']
data_type = {'name':'network_in', 'type':'Cumulative', 'unit':'B'}

[sgrisou1-OUT]
probes = ['nancy.sgrisou1.1-0', 'nancy.sgrisou1.1-1']
probes_names = ['nancy.grisou-1', 'nancy.grisou-2']
data_type = {'name':'network_out', 'type':'Cumulative', 'unit':'B'}
```

1 Switch, 1 node, multiple network interfaces

1 Node with 2 network interfaces (eth0 and eth1) connected to 1 switch.

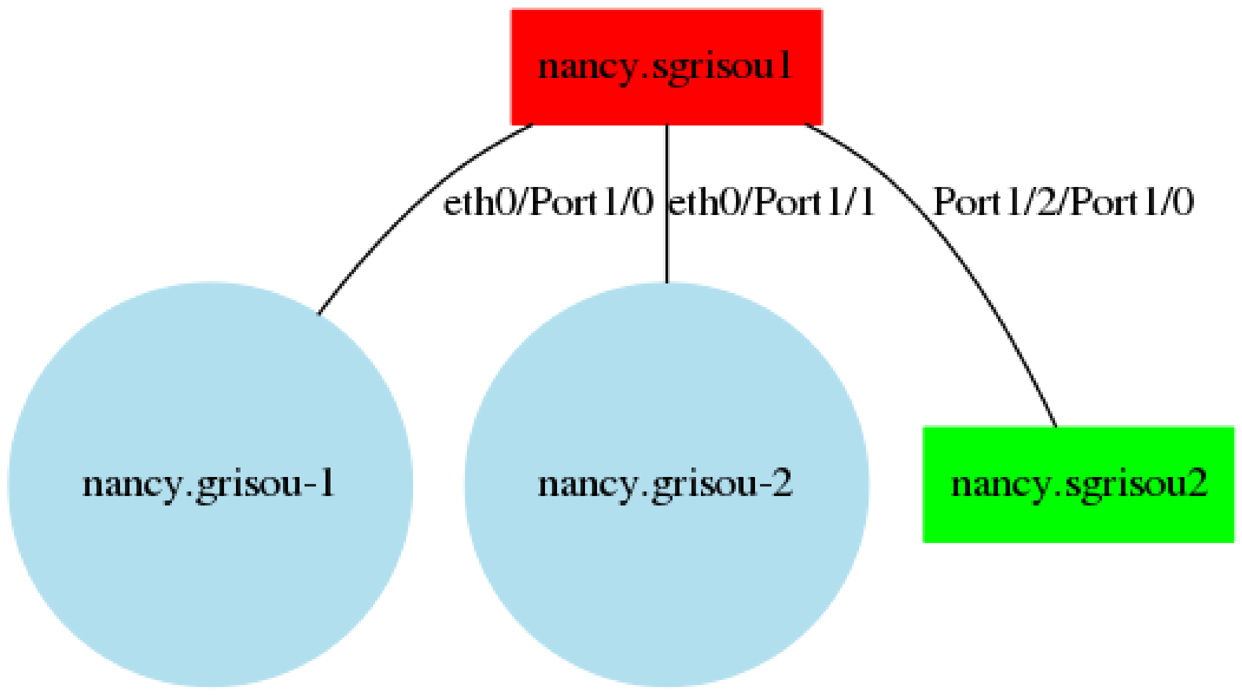


Listing 2.6: drivers.conf

```
[sgrisou1-IN]
probes = ['nancy.sgrisou1.1-0', 'nancy.sgrisou1.1-1']
probes_names = ['nancy.grisou-1', 'nancy.grisou-1-eth1']
data_type = {'name':'network_in', 'type':'Cumulative', 'unit':'B'}

[sgrisou1-OUT]
probes = ['nancy.sgrisou1.1-0', 'nancy.sgrisou1.1-1']
probes_names = ['nancy.grisou-1', 'nancy.grisou-1-eth1']
data_type = {'name':'network_out', 'type':'Cumulative', 'unit':'B'}
```

1 switch, shared measure, 2 nodes

2 nodes connected to a non-monitorable switch. The non-monitorable switch in red is connected to a monitorable switch in green. It is equivalent to a shared bandwitch measure for the 2 nodes.

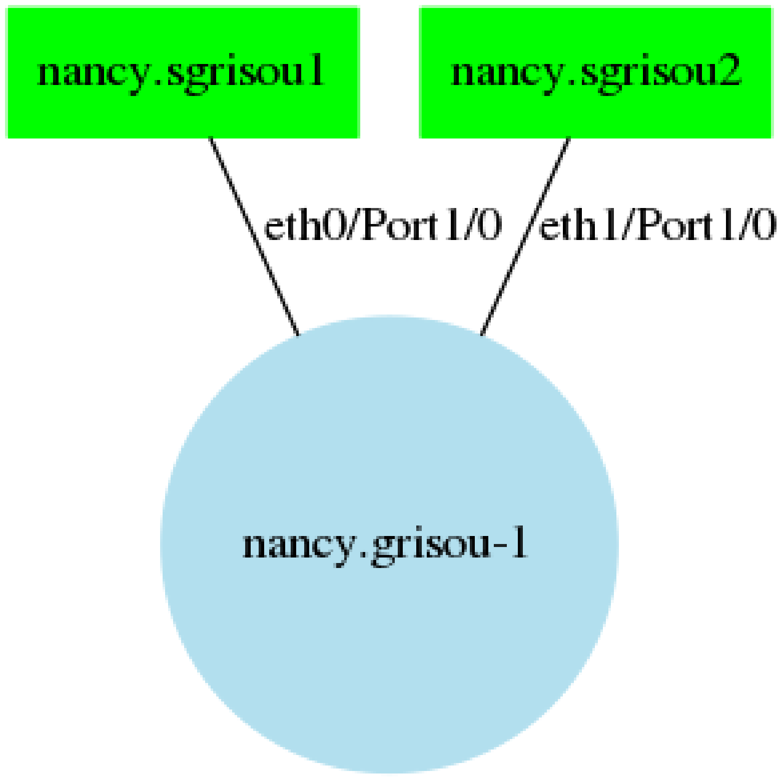


Listing 2.7: drivers.conf

```
# No info for sgrisou1
[sgrisou2-IN]
probes = ['nancy.sgrisou2.1-0', ]
probes_names = [['nancy.grisou-1', 'nancy.grisou-2']]
data_type = {'name':'network_in', 'type':'Cumulative', 'unit':'B'}

[sgrisou2-OUT]
probes = ['nancy.sgrisou2.1-0', ]
probes_names = [['nancy.grisou-1', 'nancy.grisou-2']]
data_type = {'name':'network_out', 'type':'Cumulative', 'unit':'B'}
```

2 switches, 1 node, 2 network interfaces

1 node with 2 network interfaces connected to 2 different switches.

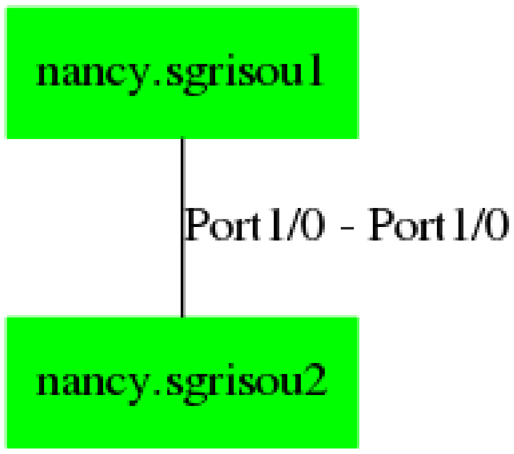


Listing 2.8: drivers.conf

```
[sgrisou1-IN]
probes = ['nancy.sgrisou1.1-0', ]
probes_names = ['nancy.grisou-1',]
data_type = {'name':'network_in', 'type':'Cumulative', 'unit':'B'}

[sgrisou1-OUT]
probes = ['nancy.sgrisou1.1-0', ]
probes_names = ['nancy.grisou-1',]
data_type = {'name':'network_out', 'type':'Cumulative', 'unit':'B'}

[sgrisou2-IN]
probes = ['nancy.sgrisou2.1-0', ]
probes_names = ['nancy.grisou-1']
data_type = {'name':'network_in', 'type':'Cumulative', 'unit':'B'}

[sgrisou2-OUT]
probes = ['nancy.sgrisou2.1-0', ]
probes_names = ['nancy.grisou-1-eth1']
data_type = {'name':'network_out', 'type':'Cumulative', 'unit':'B'}
```

2 switches interlink

2 switches are connected to each other.



Listing 2.9: drivers.conf

```
[sgrisou1-IN]
probes = ['nancy.sgrisou1.1-0', ]
probes_names = ['nancy.sgrisou1_sgrisou2']
data_type = {'name':'network_in', 'type':'Cumulative', 'unit':'B'}

[sgrisou1-OUT]
probes = ['nancy.sgrisou1.1-0', ]
probes_names = ['nancy.sgrisou1_sgrisou2']
data_type = {'name':'network_out', 'type':'Cumulative', 'unit':'B'}

[sgrisou2-IN]
probes = ['nancy.sgrisou2.1-0', ]
probes_names = ['nancy.sgrisou2_sgrisou1']
data_type = {'name':'network_in', 'type':'Cumulative', 'unit':'B'}

[sgrisou2-OUT]
probes = ['nancy.sgrisou2.1-0', ]
probes_names = ['nancy.sgrisou2_sgrisou1']
data_type = {'name':'network_out', 'type':'Cumulative', 'unit':'B'}
```

Grid‘5000 integration

Grid‘5000 tools

There are 2 tools developped in kwapi that can be used in Grid‘5000:

- **kwapi-g5k-conf**
- **kwapi-g5k-check**

kwapi-g5k-conf

This tool is used to generate Kwapi drivers configuration. It uses content of */etc/kwapi/drivers.conf.orig* as an initial content and use the Grid‘5000 API (with execo) to retrieve power configuration and network configuration.

This script is compatible with both *SNMP* and *JSON* drivers. It creates 1 entry per PDU and 1 entry per switch. Only compute nodes are configured with power monitoring. Only nodes, servers, interswitch links, Renater links and addtionnal virtual links described in Grid‘5000 API are configured for network monitoring. Resolution is set at *1 mesure per second* by default.

kwapi-g5k-check

This tool is used to check Kwapi configuration. It uses execo to get Grid‘5000 status and reserve the nodes with the API. It has to be launched from Grid‘5000 frontend.

It will reserve the nodes of a site and stress then one by one. By pulling the measurement API, it will check that power metrics increase for the node concerned by the test. At the end, it will show a list of nodes for which the check fail.

A network version has been added to this script but was never tested on Grid‘5000.

Reference-repository configuration

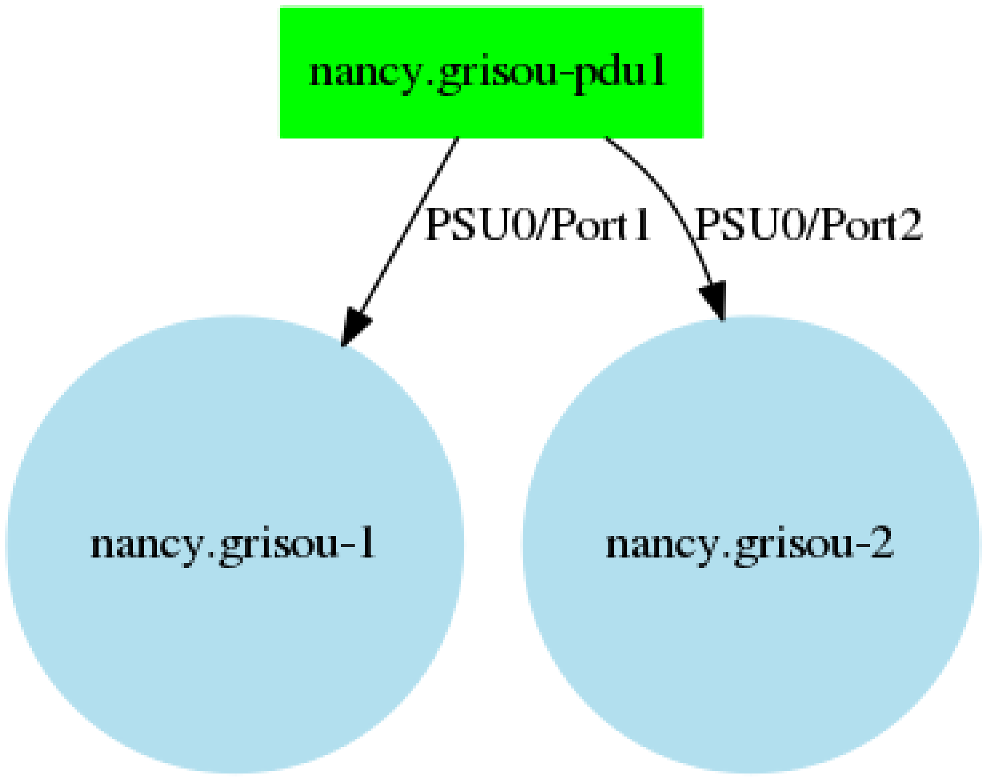
PDU entries in Grid‘5000 API has to be well defined in order to generate correctly the scripts. We will take each examples of the exemple section and see how to expose them through Grid‘5000 API.

Warning: As a reference, you should probably read before:

- https://www.grid5000.fr/mediawiki/index.php/Measurements_tutorial#Finding_nodes_supporting_power_metrics,
- https://www.grid5000.fr/mediawiki/index.php/Measurements_tutorial#Power_probe_naming_and_description
- https://www.grid5000.fr/mediawiki/index.php/TechTeam:Reference_Repository#Monitoring_section

1 PDU 2 nodes

2 nodes connected to 1 PDU.



Listing 2.10: pdus.yaml

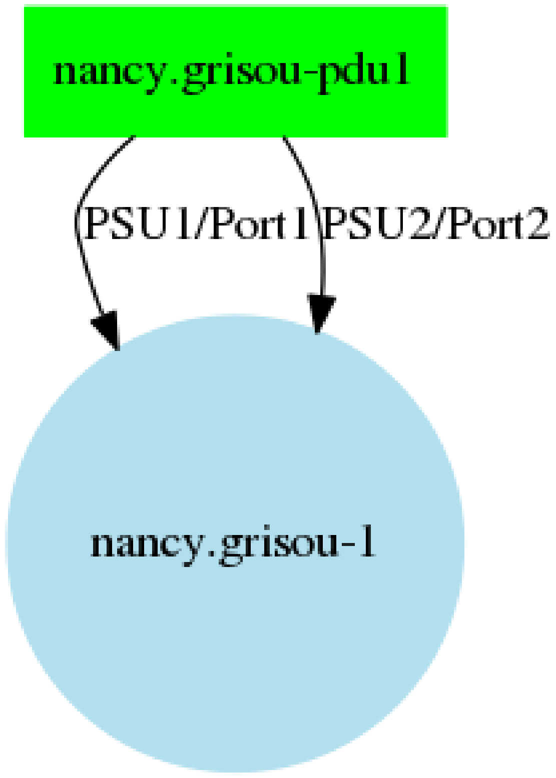
```
---
pdus:
  grisou-pdu1:
    vendor: APC
    model: AP8653
    sensors:
     - power:
        per_outlets: true
        snmp:
          available: true
          total_oids:
            - iso.3.6.1.4.1.318.1.1.12.1.16.0
          unit: W
          outlet_prefix_oid: iso.3.6.1.4.1.318.1.1.26.9.4.3.1.7
        resolution: 1
```

Listing 2.11: grisou.yaml

```
---
nodes:
  grisou-[1-2]:
    monitoring:
      wattmeter: true
      metric: power
  grisou-1:
    pdu:
      - uid: grisou-pdu1
        port: 1
  grisou-2:
    pdu:
      - uid: grisou-pdu1
        port: 2
```

1 PDU, 1 node, multiple PSU

1 Node with 2 PSU (Power Supply Unit) connected to 1 PDU.



Listing 2.12: pdus.yaml

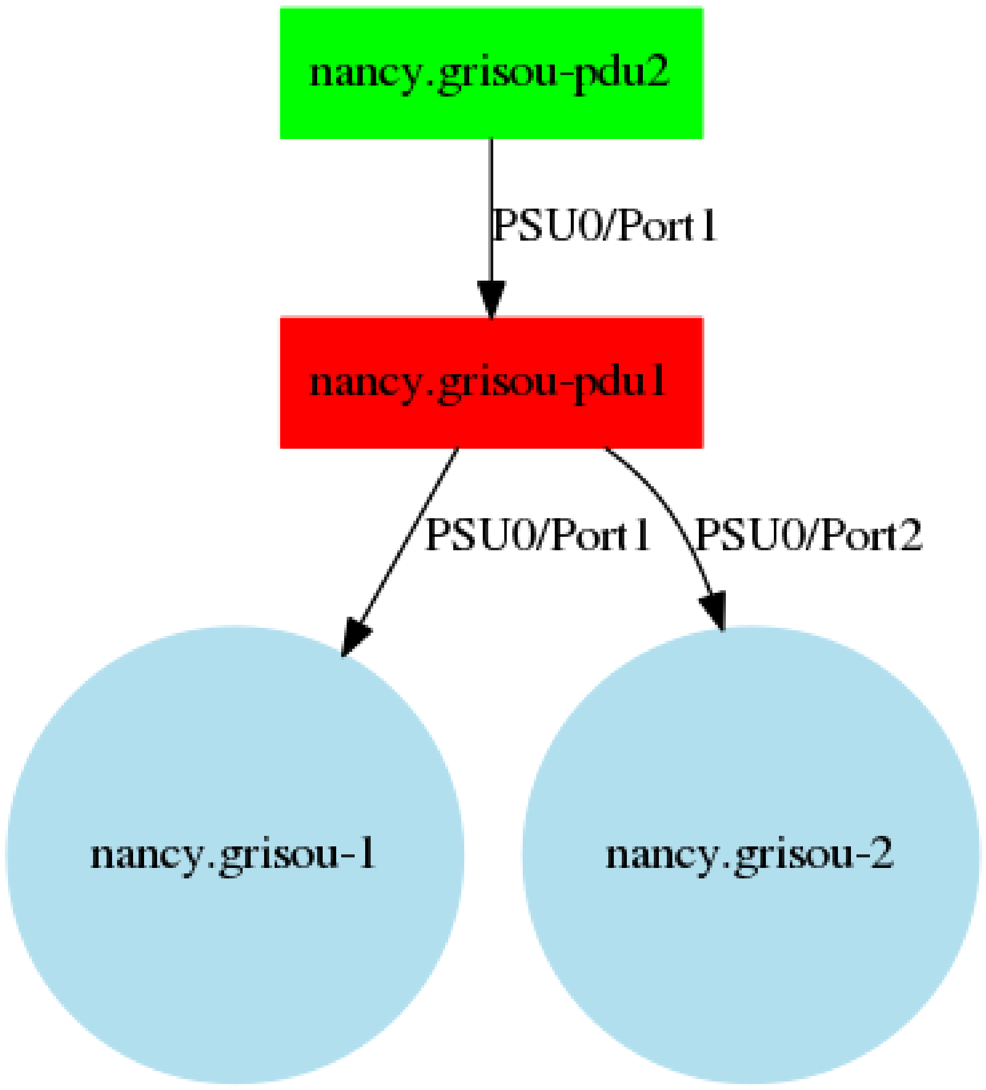
```
---
pdus:
  grisou-pdu1:
    vendor: APC
    model: AP8653
    sensors:
     - power:
        per_outlets: true
        snmp:
          available: true
          total_oids:
            - iso.3.6.1.4.1.318.1.1.12.1.16.0
          unit: W
          outlet_prefix_oid: iso.3.6.1.4.1.318.1.1.26.9.4.3.1.7
        resolution: 1
```

Listing 2.13: grisou.yaml

```yaml
---
nodes:
  grisou-1:
    monitoring:
      wattmeter: multiple
      metric: power
    pdu:
      - uid: grisou-pdu1
        port: 1
      - uid: grisou-pdu1
        port: 2
```

1 PDU, shared measure, 2 nodes

2 nodes connected to a non-monitorable PDU. The non-monitorable PDU in red is connected to a monitorable PDU. It is equivalent to a shared consumption measure for the 2 nodes.



Listing 2.14: pdus.yaml

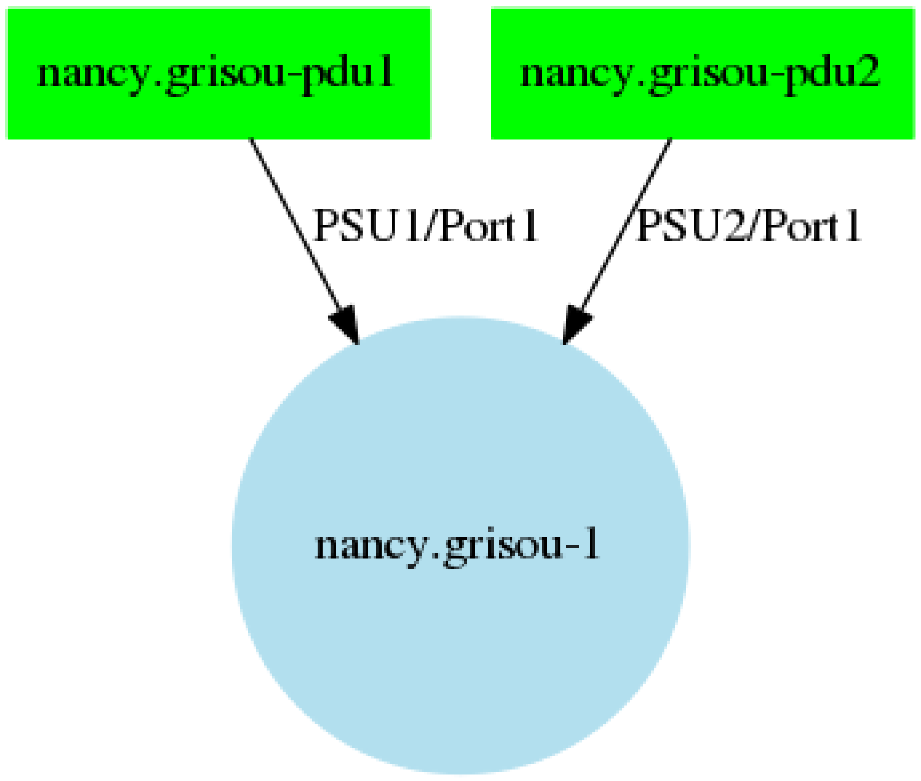
```
---
pdus:
  grisou-pdu1:
    vendor: APC
    model: AP7953
    sensors:
     - power:
        per_outlets: false
        snmp:
          available: true
          total_oids:
            - iso.3.6.1.4.1.318.1.1.12.1.16.0
          unit: W
        resolution: 1
    monitoring:
      wattmeter: true
      metric: power
    pdu:
      - uid: grisou-pdu2
        port: 1
  grisou-pdu2:
    vendor: APC
    model: AP8653
    sensors:
     - power:
        per_outlets: true
        snmp:
          available: true
          total_oids:
            - iso.3.6.1.4.1.318.1.1.12.1.16.0
          unit: W
          outlet_prefix_oid: iso.3.6.1.4.1.318.1.1.26.9.4.3.1.7
        resolution: 1
```

Listing 2.15: grisou.yaml

```
---
nodes:
  grisou-[1-2]:
    monitoring:
      wattmeter: shared
      metric: power
  grisou-1:
    pdu:
      - uid: grisou-pdu1
        port: 1
  grisou-2:
    pdu:
      - uid: grisou-pdu1
        port: 2
```

2 PDU, 1 node, 2 PSU

1 node with 2 PSU connected to 2 different PDU.



Listing 2.16: pdus.yaml

```
---
pdus:
  grisou-pdu1:
    vendor: APC
    model: AP8653
    sensors:
     - power:
        per_outlets: true
        snmp:
          available: true
          total_oids:
            - iso.3.6.1.4.1.318.1.1.12.1.16.0
          unit: W
          outlet_prefix_oid: iso.3.6.1.4.1.318.1.1.26.9.4.3.1.7
        resolution: 1
  grisou-pdu2:
    vendor: APC
    model: AP8653
    sensors:
     - power:
        per_outlets: true
        snmp:
          available: true
          total_oids:
            - iso.3.6.1.4.1.318.1.1.12.1.16.0
          unit: W
          outlet_prefix_oid: iso.3.6.1.4.1.318.1.1.26.9.4.3.1.7
        resolution: 1
```

Listing 2.17: grisou.yaml

```
---
nodes:
  grisou-1:
    monitoring:
      wattmeter: multiple
      metric: power
    pdu:
      - uid: grisou-pdu1
        port: 1
      - uid: grisou-pdu2
        port: 1
```

Kwapi Development

Project Hosting Details

Bug tracker https://intranet.grid5000.fr/bugzilla/

Code Hosting https://github.com/grid5000/kwapi-g5k

Mailing list support-staff Grid‘5000

Grid‘5000 Doc https://www.grid5000.fr/w/Measurements_tutorial

Areas to Contribute

Drivers

Kwapi aims at supporting various drivers. If you have a non-supported wattmeter, you can easily contribute by writing a new one.

Plugins

Kwapi plugins process the metrics. You can contribute by writing new plugins to bring new functionnalities.

Testing

The first version of Kwapi has not yet unit tests and has not seen much run-time in real environments.

Working with the Source

Setting up a Development Sandbox

1. Set up a server or virtual machine.
2. Clone the kwapi project to the machine:

```
$ git clone https://github.com/grid5000/kwapi-g5k.git
$ cd ./kwapi-g5k
```

3. Once this is done, use install option of *setup.py* file to install kwapi locally:

```
$ python setup.py install
```

4. If some dependant packages are missing, fix them with *pip install*:

```
$ pip install -r requirements.txt
```

4. You can start to hack kwapi. If you are preparing a patch, create a topic branch and switch to it before making any changes:

```
$ git checkout -b TOPIC-BRANCH
```

5. Use git to push your changes and ask for a pull request.
6. Package your solution for Debian installation:

```
$ python setup.py --command-packages=stdeb.command bdist_deb
$ cd deb_dist/
```

   All the deb archives are exported in this directory.

7. Import the new generated packages of kwapi-g5k on the remote apt repository.
8. Execute Puppet on the VM to install the latest version of Kwapi or simply run:

```
$ apt-get update && apt-get install python-kwapi-g5k
```

Code Reviews

Kwapi uses the GitHub to hos all code and developer documentation contributions. You can report an issue or a feature request on this repository.

Bugzilla can also be used for API related bugs or device configuration problems.

Glossary

driver Software thread running querying a wattmeter or switch and sending the results to the plugins.

forwarder Component that forwards plugins subscriptions and metrics. Used to minimize the network traffic, or to connect isolated networks through a gateway.

plugin An action triggered whenever a meter reaches a certain threshold.

probe A wattmeter sensor or network device. A wattmeter can have only one probe (usually the IPMI cards), or multiple probes (usually the PDUs). A network device usually have multiples probes that correspond to his network interfaces. One probe is defined for incoming traffic and one for outgoing traffic.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index