Document	Deliverable Project XLcloud / Magellan FSN – AAP Cloud Computing #1		
Planned date	07/03/2013	**Nature**	Internal
Delivery date	07/03/2013		
Statut	Final	**Version**	Revision n°1

Document Properties

Document Title	Energy Efficiency Architecture
Task number	4.2.1
Responsible	François Rossigneux
Author(s) / contributor(s)	Julien Carpentier Jean-Patrick Gelas Laurent Lefèvre François Rossigneux
Document Status	Final
Version	Revision n°1

Summary

The main task was to design an architecture to retrieve power consumption data from wattmeters, and send them to OpenStack Ceilometer.

The wattmeter drivers get power consumption data from various wattmeters, and send them on a bus.

The plugins listen the bus, and process the received data (make them available through an API, building graphs...).

We have also developed a pollster for Ceilometer, so that it can query our API plugin.

Keywords

energy, wattmetre, api

Energy Efficiency Architecture

Sommaire

1 Introduction

1.1 Purpose

This software design specification provides an overview of design and architecture of the proposed Energy Efficiency Architecture (EEA) for XLcloud project.

1.2 Scope

OpenStack virtual machines scheduling doesn't take into account the energy efficiency criteria. So we need to retrieve power consumption data, and keep them at the disposal of OpenStack. EEA was designed to provide an architecture that collects wattmeters measurements, and offers an API for OpenStack.

2 System Overview

As presented in the following figure, the servers are clustered in each datacenter, and each datacenter has an internal administration network.These servers are monitored with wattmeters.

Eaton ePDU		Voltage: 230 V Current: 16 A Outlets: C13, C19 Interface: serial, ethernet (SNMP) Measure frequency: 1 value every 5 seconds Measure precision: 1 W
Schleifenbauer ePDU		Voltage: 230 V Current: 16 A Outlets: C13, C19 Interface: ethernet (SNMP, Modbus, MySQL) Measure frequency: 1 value every 3 seconds Measure precision: below 0.1 W
Dell iDRAC6 : IPMI		Measures: internal sensors (Dell proprietary sensors) Interface: IPMI Measure frequency: 1 value every 5 seconds Measure precision: 7 W
Wattmetre (OmegaWatt)		Measure frequency:1 value per second Measure precision: below 1 W Interface: serial

On service machines, Kwapi-drivers retrieves power consumption data and sends them to Kwapi-API.

Then OpenStack queries Kwapi-API and stores power consumption data.

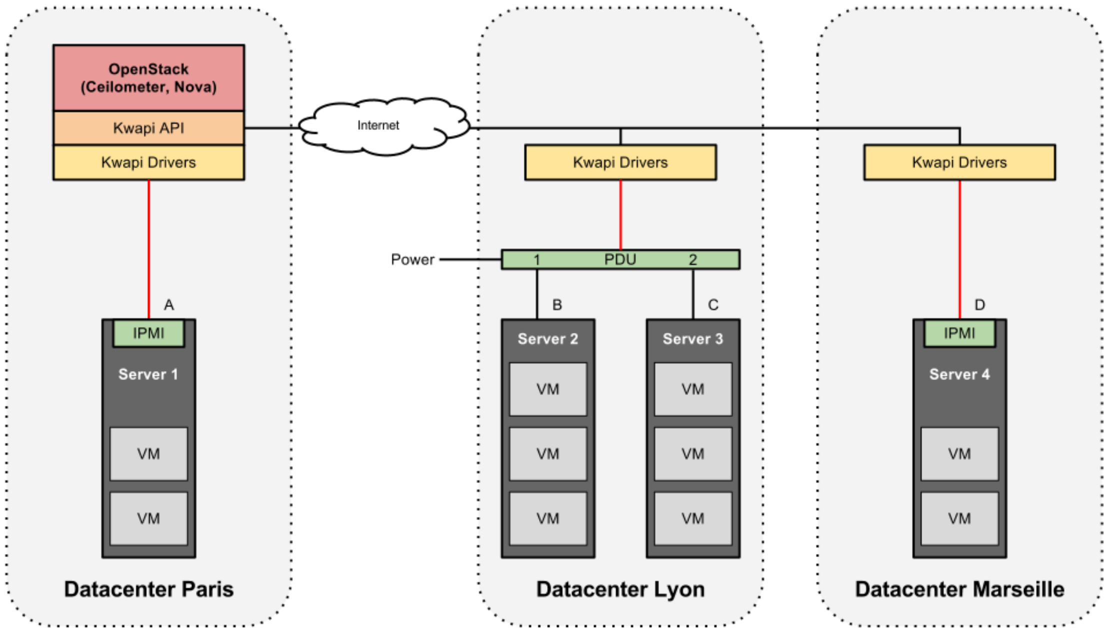

Figure 1: Datacenter architecture overview

This figure reveals several layers, and the resulting software architecture could be summarized by the following figure.

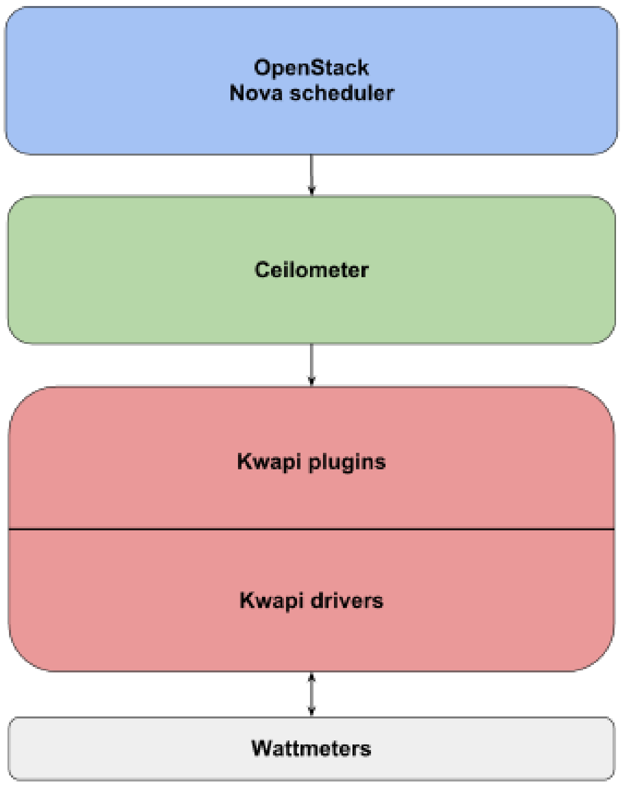

Figure 2: Software architecture overview

The produced software is named Kwapi, and has two main layers : a drivers layer, and a plugins layer.

The drivers layer retrieves measurements from wattmeters, and sends them on a bus.
(generally over network, but sometimes on a local bus if drivers and plugins are on the same machine)

The plugins layer includes several plugins that listen on the bus. There is currently two plugins : the first one to provide an API for OpenStack Ceilometer, and the second one to provide a visualization interface.

3 System Architecture

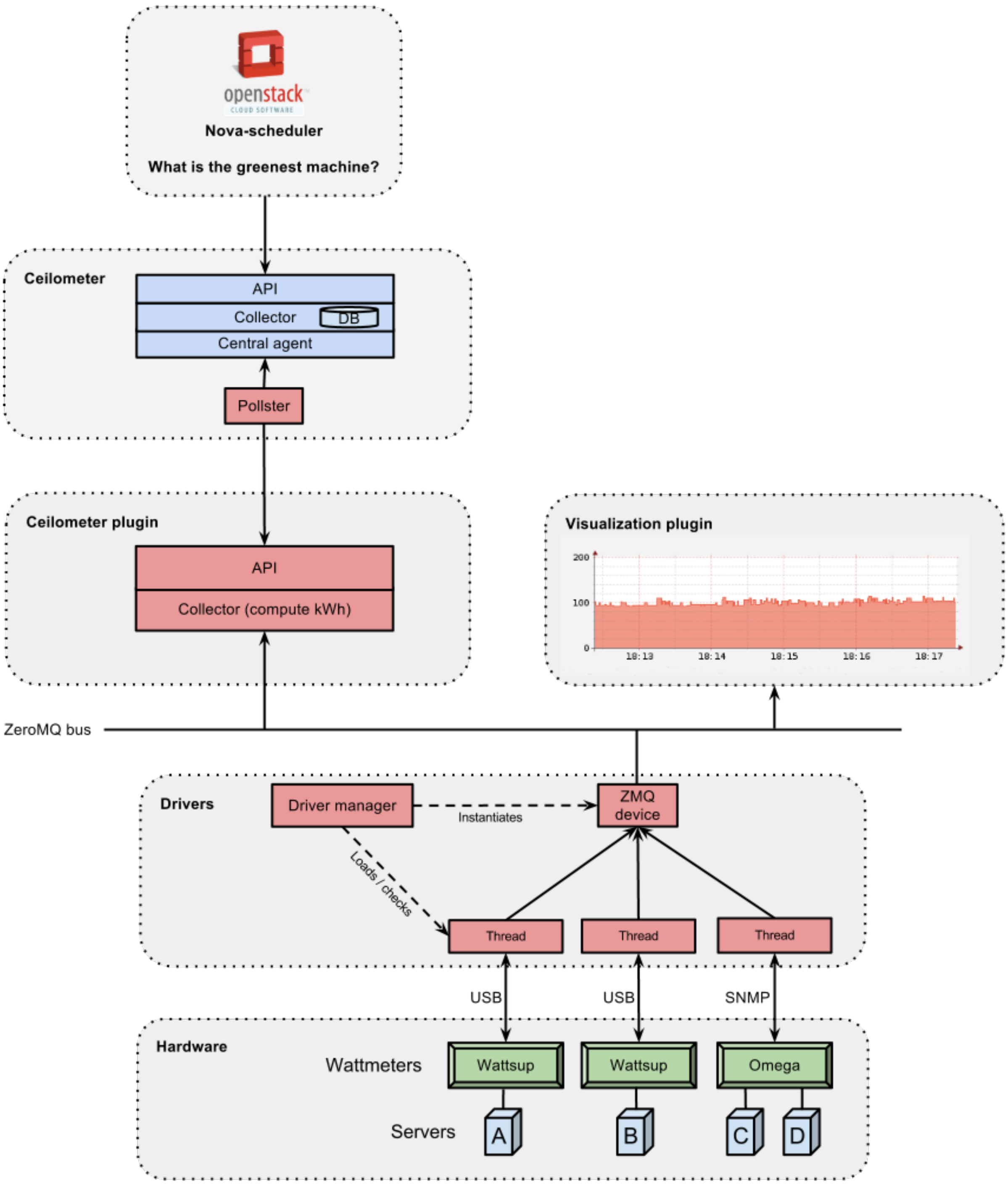

Figure 3: Software detailed architecture

3.1 Kwapi Drivers

3.1.1 Driver classes (IPMI, Wattsup, OmegaWatt, etc) and superclass

Kwapi supports different kinds of wattmeters (IPMI, 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. Wattmeter 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. Their constructors takes as arguments a list of probe IDs, and kwargs (specific arguments).

Driver threads roles are:

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

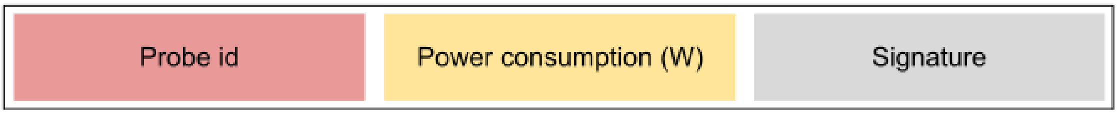

Figure 4: Bus message format

3.1.2 Driver manager

The driver manager is used as loader and checker for driver threads. It loads all drivers according to the configuration file, and checks regularly that driver threads are alive. In case of crash, 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.

3.2 Bus

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

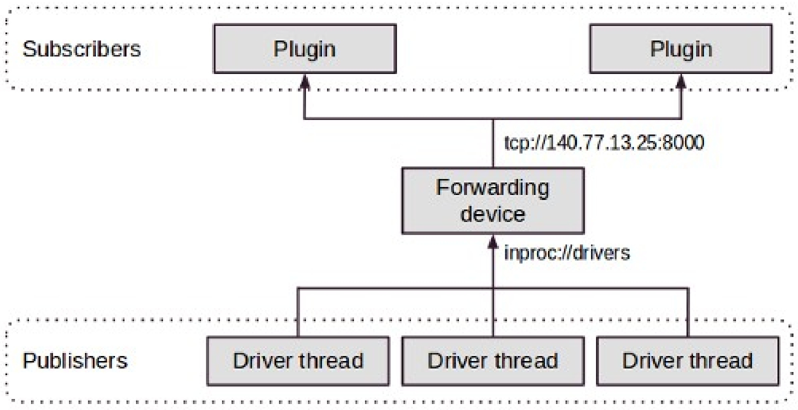

Figure 5: Bus design pattern

Plugins can easily listen multiple data sources:

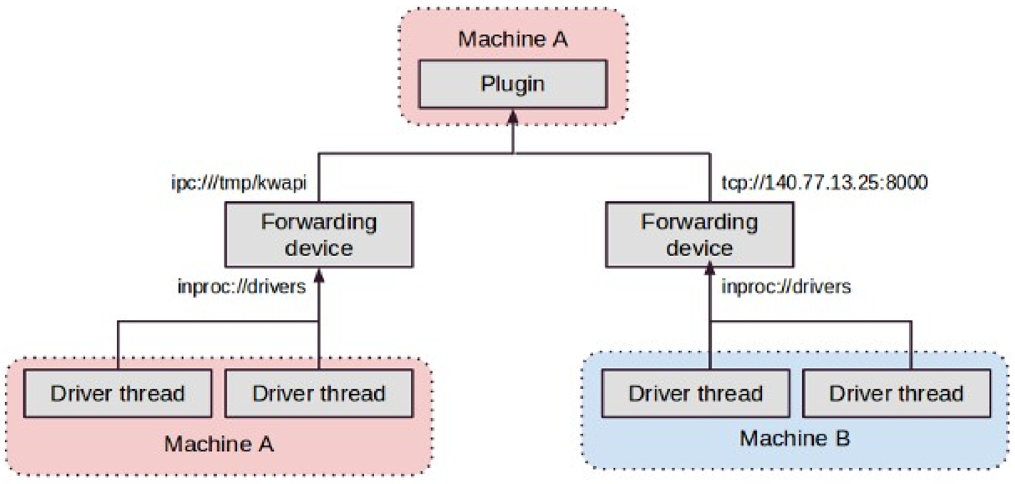

Figure 6: Multiple source example

In the near future, we plan to use OpenStack RPC framework.

3.3 Kwapi Plugins

3.3.1 API Plugin for Ceilometer

API plugin allows Ceilometer pollster to get consumption data through a REST API. This plugin contains a collector that computes kWh, and an API based on Flask (A Python Microframework).

Collector

The collector stores these values for each probe:

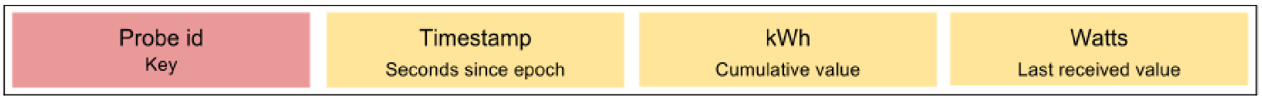

Figure 7: Collector record

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 (Managed PDU).
- Timestamp is updated when a new value is received.
- KWh is computed by taking into account new consumption value, and the elapsed time since the previous update.
  It allows Ceilometer to compute average consumption for the duration it wants (knowing the kWh consumed and the time elapsed since its last check).
- Watts field offers the possibility to know live consumption 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 for the next value.

There is no watt logs to avoid duplicating the storage architecture of Ceilometer. The collector is cleaned periodically to prevent a deleted probe from being stored indefinitely in the collector. So if a probe is not updated for a long time, it is removed.

API

Verb	URL	Parameters	Expected result
GET	/v1/	-	Returns detailed information about this specific version of the API.
GET	/v1/probe-ids/	-	Returns all known probe IDs.
GET	/v1/probes/	-	Returns all information about all known probes.
GET	/v1/probes/<probe>/	probe id	Returns all information about this probe (id, timestamp, kWh, W).
GET	/v1/probes/<probe>/<meter>/	probe id meter { timestamp, kwh, w }	Returns the probe meter value.

Authentication

The pollster provides a token (X-Auth-Token). The API plugin checks the token (Keystone request), and if the token is valid, requested data are sent. Responses are not signed because Ceilometer trusts Kwapi plugin.

Ceilometer pollster

Ceilometer pollster is started periodically by Ceilometer central agent. It knows the Kwapi URL by doing a

Keystone request (endpoint-get). It queries probe values through Kwapi API, using the GET /v1/probes/ call, so that it gets all detailed informations about all probes in just one query. For each probe, it creates a counter

object and publishes it on the Ceilometer bus.

Published counters:

- Energy (cumulative type): represents kWh.
- Power (gauge type): represents watts.

Counter timestamps are Kwapi timestamps, so that Ceilometer doesn't store wrong data if a probe is not updated. Ceilometer handles correctly the case where a probe value is reset (kWh decrease), because of its cumulative type.

3.3.2 Visualization Plugin

Web interface

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

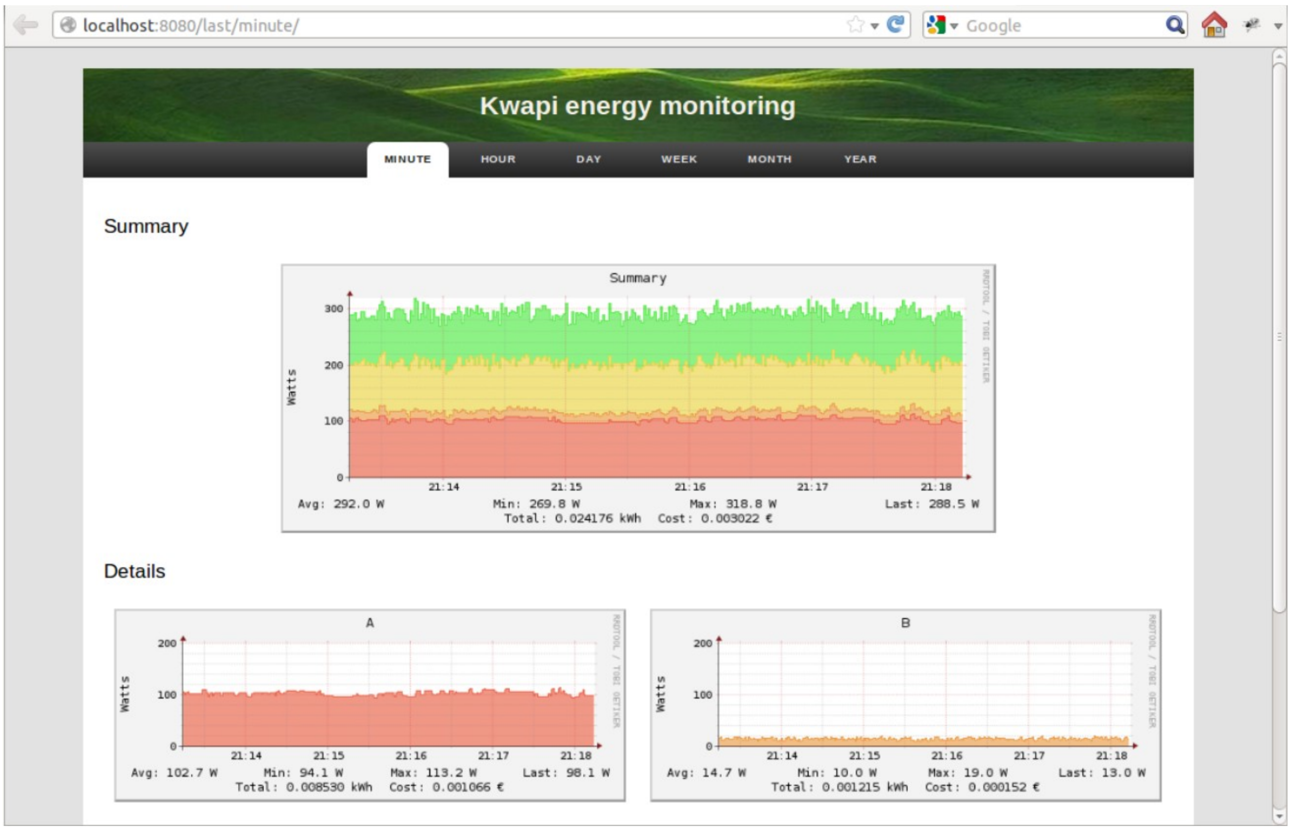

Figure 8: Aggregate and all probes consumption graphs

In the menu bar, you can choose the period for which you want to display graphs (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. Graphs are refreshed every five seconds, but bandwidth is saved by using the HTTP 304 response codes.

Verb	URL	Parameters	Expected result
GET	/last/<period>/	period { minute, hour, day, week, month, year}	Returns a webpage with an aggregate graph and all probe graphs.
GET	/probe/<probe>/	probe id	Returns a webpage with all graphs about this probe (all periods).
GET	/graph/<period>/	period { minute, hour, day, week, month, year}	Returns an aggregate graph about this period.
GET	/graph/<period>/<probe>/	period { minute, hour, day, week, month, year} probe id	Returns a graph about this probe.

Graphs

The aggregate graph shows the total power consumption (sum of all the probes). Each colour corresponds to a probe.

The legend contains:

- Minimum, maximum, average and last power consumption.
- Energy consumed (kWh).
- Cost.

File sizes:

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

These is a cache mechanism: recent graphs are not rebuilt uselessly.

4 Code organization

Repository: https://github.com/stackforge/kwapi

The main directory contains the sub-directories *drivers* and *plugins*.

In *drivers* directory, we found:

- *driver_manager.py*: loading and checking
- *driver.py:* driver superclass
- Several driver classes (for example *ipmi.py*, *wattsup.py*, etc)

The *plugin* directory a sub-directory for each plugin:

- *api*: API plugin for Ceilometer
- *rrd*: visualization plugin

The main directory also contain common code:

- *openstack*: common code (used for logging and configuration)
- *security.py*: used for message signing between drivers and plugins