OCCIMON — An OCCI-Monitoring proof of concept

Augusto Ciuffoletti

August 26, 2014

This document will first guide you into the task of configuring a virtual network, and implementing a simple monitoring infrastructure using the **OCCIMON** demo package. Next we explain hot to install the Eclipse (TM) project with the source of the **OCCIMON** package.

1 The virtual network

The virtual network is built using VirtualBox(TM). It is made of four hosts:

pc the guest node (`192.168.5.2`), containing the description of the monitoring infrastructure as a set of OCCI-JSON files

router a virtual machine (`192.168.5.1`) that acts as router and DHCP server

c1 a virtual machine (`192.168.5.3`) that is the target of the monitoring activity

sensor1 a virtual machine (`192.168.5.6`) that is the sensor in our monitoring infrastructure

The four hosts share the same virtual Ethernet, a "Host Only" network in VirtualBox terminology. In addition, the router VM has a "NAT" interface as a facility to connect to network to the Internet (not strictly needed for operation).

2 The virtual machines

The VMs run a Linux Debian with no graphical interface. Their image can be retrieved from ... The code needed to run the experiment can be downloaded from "here" (demo.tgz) (less than 1MB).

On the `router`, a tiny Python script (`httpServer.py`, included in demo.tgz tarball) implements the repository of OCCI-JSON documents using the SimpleHTTPServer module:

```
#!/usr/bin/python
import SimpleHTTPServer
import SocketServer

Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
httpd = SocketServer.TCPServer(("", 6789), Handler)
print "Server ready..."
httpd.serve_forever()
```

The script is launched in the directory hosting the OCCI-JSON configuration files. For our convenience, in the same directory we have also the executable jars useful for the experiment.

```
augusto$laplenovo:~experiment/ ./httpServer.py
Server ready...
```

The files urn:uuid:* contain the description of a target compute resource (`c2222`), of the sensor resource (`s1111`), and of the collector link (`2345`).

The OpenJDK is installed on each VM:

```
user@c1:~$ java -version
java version "1.7.0_25"
OpenJDK Runtime Environment (IcedTea 2.3.10) (7u25-2.3.10-1~deb7u1)
OpenJDK Client VM (build 23.7-b01, mixed mode, sharing)
```

The official JDK works as well.

3 The experiment

Run the following commands in this order:

1. on the router virtual host (`router`), launch the web server:

   ```
   httpServer.py
   ```

2. on the compute virtual host (`c1`), launch the metric container:

   ```
   java -jar MetricContainer-v2.jar urn:uuid:c2222 http://router:6789/
   ```

3. on the sensor virtual host (`sensor1`), launch the sensor resource:

   ```
   java -jar Demo-v2.jar urn:uuid:s1111 http://router:6789/
   ```

4. on the guest computer, start a UDP receiver (e.g., netcat):

   ```
   nc -ul 8888
   ```

To have a significant output, it is appropriate to background a CPU consuming task on c1 before launching the metric container, e.g.:

```
( while true; do sleep 1; find / -name x > /dev/null 2>&1; done ) &
```

4 Description of the experiment

The compute resource `urn:uuid:c2222` is equipped with a MetricContainer, a feature that allows sensors to instantiate measurement tools on the compute resource. The feature is available as a mixin that complements the compute resource.

The sensor resource `urn:uuid:s1111` instructs the MetricContainer to run two distinct measurements that are included in the same collector with id `urn:uuid:2345`:

- the CPU load
- the reachability of the guest computer (`192.168.5.2`)

They are represented as **metric** mixins attached to the collector.

The first metric is averaged using an exponentially weighted moving average, that is represented as a **aggregator** mixin. It is delivered to a UDP socket on the guest PC (port 8888) according with a **publisher** mixin.

The second metric is simply recorded locally on a (fake) log file: in this case there is no aggregation step, and the functionality is described with a **publisher** mixin.

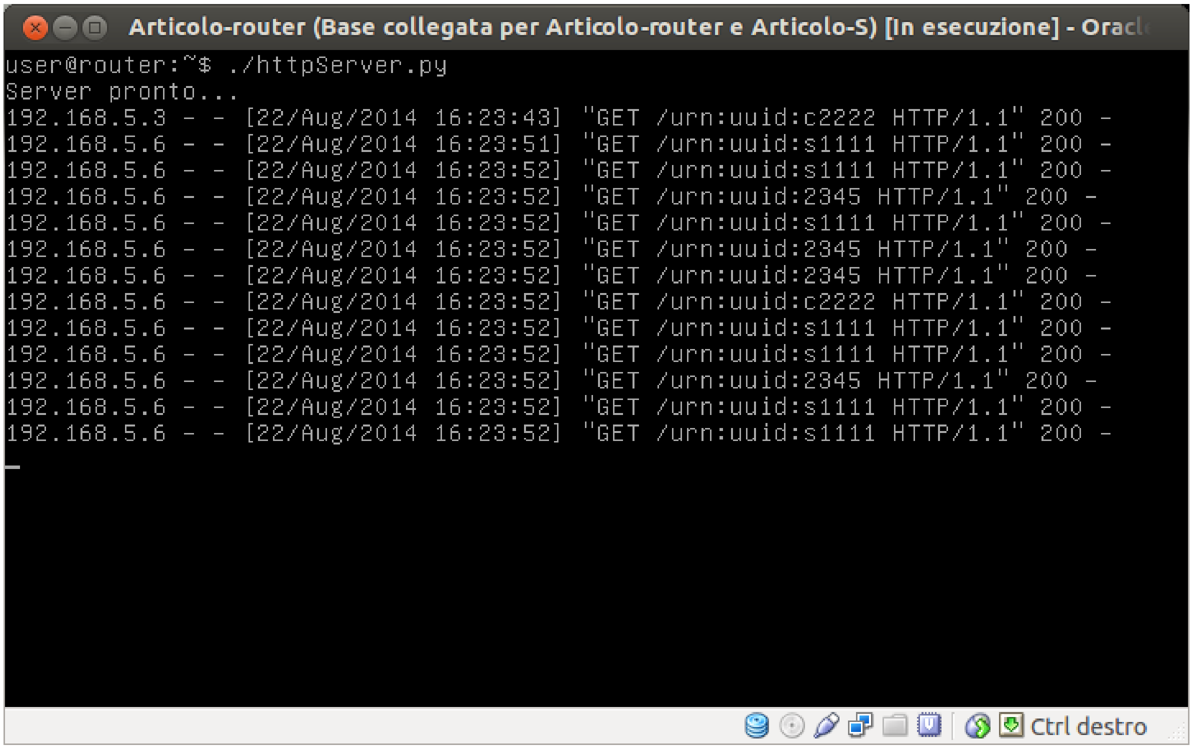

Figure 1: Screenshot on router terminal

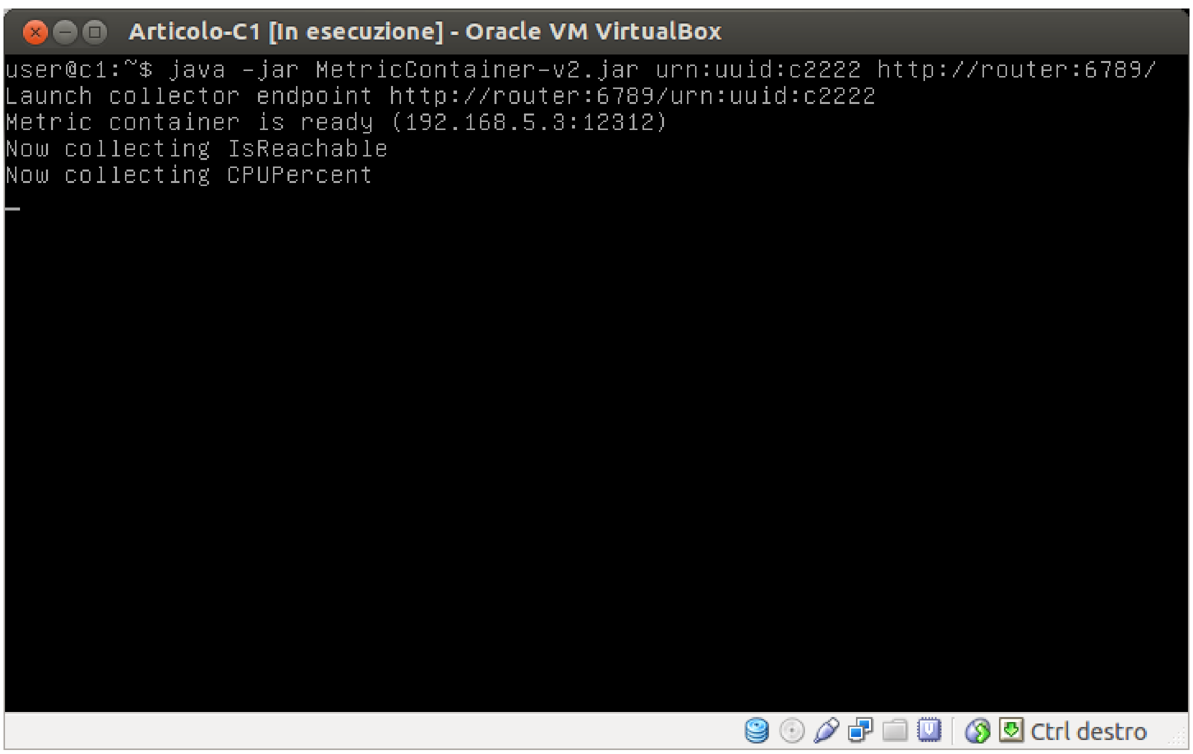

Figure 2: Screenshot on compute resource terminal

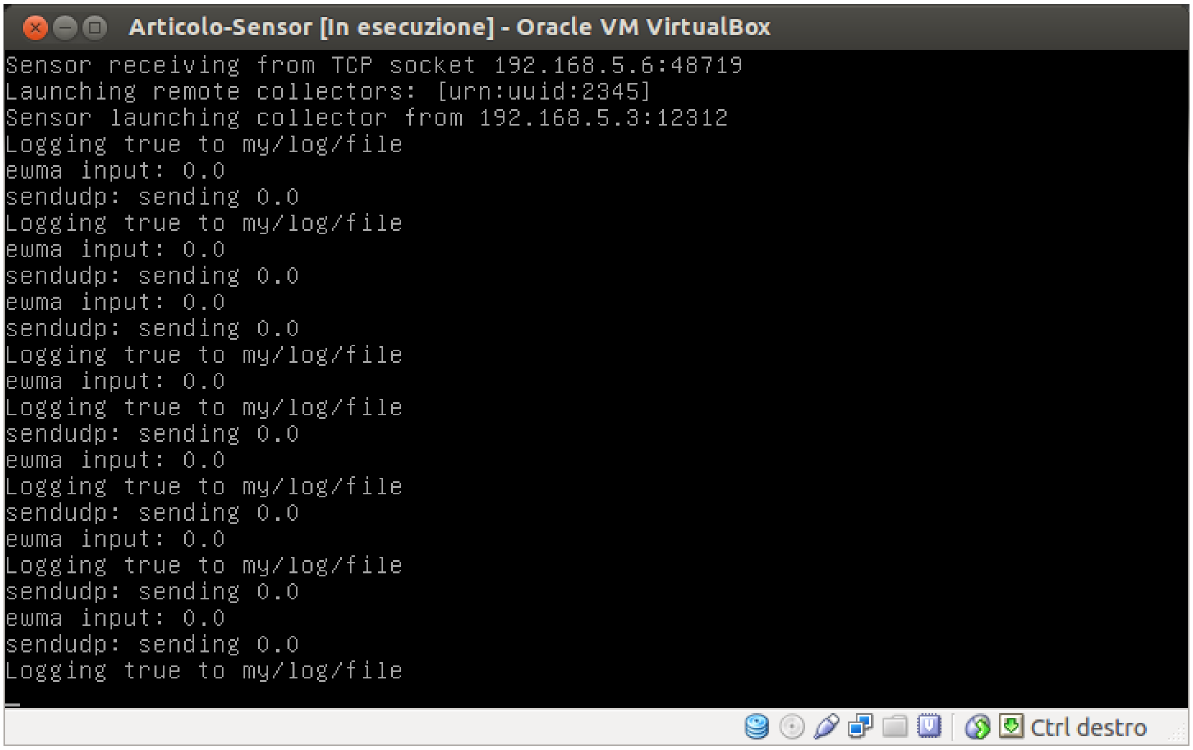

Figure 3: Screenshot on sensor resource terminal

5 Browsing the source code

The code of the demo monitoring facility has been developed with Eclipse (TM). The recommended way to explore and improve the code is by using this same develpment tool.

The `Demo-v2.jar` contains the Eclipse project of the demo monitoring facility.

To install the package in Eclipse 3.8:

- File → Import → General → Existing projects into workspace: in the dialog select "Select archive file", browse your files to find the `Demo-v2.jar` file, and finish
- check (in Window → Preferences) that the JDK is installed
- install (Java → Properties → Java Build Path → Libraries) the libraries:
  - json-simple-1.1.1.jar
  - jsoup-1.7.3.jar
- in the Project Explorer window, move all six packages into the `src` directory

The following is a summary of the packages in the project.

collector implements the `MetricContainer` application that controls the execution of the measurement plugins, the **Collector** endpoint on **Resource** side. In our demo it is controlled via RMI calls issued by the **Collector** endpoint on **Sensor** side. It launches measurement tools implemented by a pool of `MetricMixin` threads.

metric contains the implementation of the metric mixins. They all share the same interface `MetricMixin` and extend a Callable. The arguments passed for their creation are the attributes (class and mixin attributes), the output channel carrying the measurements, and the measurement period.

mixin contains the implementation of the publisher and aggregator mixins for the sensor. They share the same interface SensorMixinIF, with the sensor and mixin attributes and two channels, one for input, another for output.

sensor contains the `Sensor` class, the thread that implements the **Sensor** resource. The endpoints of the **Collector** links attached to the Sensor are implemented with a pool of threads in the class `CollectorManager`, also defined in this package. The number of collectors is currently limited to one, but it is easely expandable.

sensorContainer It is the Java application that implements a pool of sensors as `Sensor` threads (currently limited to one, but easely expandable);

systemSpecification it is a package containing a library useful to manipulate the JSON documents.

The project generates two distinct executable applications: one for the (singleton) `sensorContainer` (called `Demo`), another for the `MetricContainer`. Both of them take two arguments:

- the id of an entity (respectively a **Sensor**, or a **Resource**),
- the address of the web repository containing the documents that describe the monitoring infrastructure.

6 Disclaimer

The package is intended as a proof of concept for the content of the OCCI monitoring proposal, not for use in any production or experimental environment. The framework of the proposal is implemented, but many details are left incomplete or not functional. The proof of concept is going to be useful for the revision of the proposal, which appears to be marginal and limited to the introduction of a *MetricContainer* mixin to be added to the monitored resource.

The package will undergo a deep re-engineering step with the purpose of qualifying it as a prototype implementation.