



Data Centres Optimization for Energy-Efficient and Environmentally Friendly INternet

Funding scheme: Specific Targeted Research Projects – STREP
Co-funded by the European Commission within the Seventh Framework Programme
Project no. 609140
Strategic objective: FP7-SMARTCITIES-2013 (ICT-2013.6.2)
Start date of project: October 1th, 2013 (36 months duration)



Deliverable D3.3

ICT Performance and Energy Supervisor component (Implementation)

Due date: 31/12/2015
Submission date: 28/01/2016
Deliverable leader: I2CAT
Author list: Amaia Legarrea (I2CAT), Isart Canyameres (i2CAT), Adrián Roselló (i2CAT), Andrea Gronchi (NXW), Tommaso Zini (NXW), Artemis Voulkidis (SYN), Vangelis Angelou (GRNET)

Dissemination Level

- | | |
|-------------------------------------|---|
| <input type="checkbox"/> | PU: Public |
| <input type="checkbox"/> | PP: Restricted to other programme participants (including the Commission Services) |
| <input type="checkbox"/> | RE: Restricted to a group specified by the consortium (including the Commission Services) |
| <input checked="" type="checkbox"/> | CO: Confidential, only for members of the consortium (including the Commission Services) |

List of Contributors

Participant	Contributor
I2CAT	Amaia Legarrea, Isart Canyameres, Adrián Roselló
NXW	Tommaso Zini, Andrea Gronchi
SYN	Artemis Voulkidis
GRNET	Vangelis Angelou

Amendment History

Version	Date	Partners	Description/Comments
0.1	10/12/2015	NXW	ToC Initial draft
0.2	22/12/2015	NXW, I2CAT, SYN, GRNET	Initial contributions
0.3	05/01/2016	i2CAT	Reviewed input from version 0.2
0.4	18/01/2016	i2CAT	Addressed comments from previous version
0.5	27/01/2016	i2CAT	Final version after internal review
1.0	28/01/2016	IRT	Final version submitted to EC

Table of Contents

List of Contributors	2
Amendment History	3
Table of Contents	4
Figures Summary	6
Tables Summary	7
Abbreviations	8
Executive Summary	9
1. Introduction	10
2. ICT Performance and Energy Supervisor	12
2.1. Module description	12
2.1.1. ICT Performance and Energy Supervisor core	13
2.1.2. Metrics Engine and ICT Topology	15
2.1.3. eCOP Monitor Database	16
2.1.4. DCO Brokers	17
3. Implementation description	19
3.1. ICT Performance and Energy Supervisor core	19
3.1.1. Core Modules	19
3.1.5 Module Dependencies	21
3.1.6 Deployment requirements and testing	22
3.2. Metrics Engine and ICT Topology	22
3.2.1. Basic concepts	23
3.2.2. Module Dependencies	24
3.2.3. Deployment and installation	25
3.2.4. Testing the installation	27
3.3. eCOP Monitor Database	28
3.3.1. Installation and Configuration of the DB	29
3.3.2. Installation and configuration of the eCOP DB Broker	31
3.3.3. Testing the configuration of the eCOP DB Broker	35
3.3.4. Using the eCOP DB Broker	37
3.3.5. Service endpoints and data model	44
3.4. DCO Brokers	56
3.4.1. Installation and configuration of the DCO Brokers	56
3.4.2. Using and testing the DCO Brokers	57
3.4.3. Service endpoints and data model	60
4. Conclusions	61

References 62

Figures Summary

Figure 1-1: DOLFIN overall architecture	10
Figure 2-1: ICT Performance and Energy Supervisor architecture	12
Figure 2-2: ICT Performance and Energy Supervisor Core architecture	13
Figure 2-3 - Metrics Engine block diagram	15
Figure 2-4: The eCOP Monitor Database block diagram.	16
Figure 3-1 - Metrics Engine internal workflow.....	23
Figure 3-2 - eCOP DB APIs used by Metrics Engine	24
Figure 3-3 - Metrics Engine interactions diagram	25
Figure 3-4 - Metrics Engine Swagger web interface.....	28
Figure 3-5: Database schema of the eCOP DB, including the authentication tables, excluding the default Django framework administration tables.....	30
Figure 3-6: Dolfin group creation through the administration page.....	34
Figure 3-7: eCOP DB Broker user creation through the administration page	35
Figure 3-8: Default view of the online documentation module of the eCOP DB Broker.....	36
Figure 3-9: The default view of the online documentation module for acquiring an authorization token	38
Figure 3-10: Documenting the response class and the path-parameters of an API service endpoint.....	39
Figure 3-11: Login page of the eCOP DB Dashboard.	40
Figure 3-12: List of the servers of the hypothetical DC.	40
Figure 3-13: Details of server with serial-number SN-1.	41
Figure 3-14: The measurements tab of the eCOP DB Dashboard.	42
Figure 3-15: List of actions performed by the eCOP Policy Actuator.	42
Figure 3-16: eCOP DB Broker data model hierarchy	55

Tables Summary

Table 2-1: DCO Broker injection API.....	14
Table 2-2: Optimizer-Policy Maker Orders	18
Table 3-1: ICT Performance and Energy Supervisor Core Interactions	21
Table 3-2- Metrics Engine RESTfull APIs	23
Table 3-3: The main endpoints URLs related to the dashboard utilities.	43
Table 3-4: List of API service endpoints related to the cooling elements of a DOLFIN DC.	44
Table 3-5: Data model for the 'cooling' elements.	45
Table 3-6: List of API service endpoints related to the Flavours.....	45
Table 3-7: Data model for the 'cooling' elements.	45
Table 3-8: List of API service endpoints related to the Lightning.....	46
Table 3-9: Data model for the 'lightning' elements.....	46

Abbreviations

API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CXF	Apache Open-source framework: https://cxf.apache.org/
DB	Data Base
DC	Data Centre
DCO	Data Centre Optimization
DoW	Description of Work
eCOP	energy Consumption and Optimization Platform
HTTP	HyperText Transfer Protocol
HW	Hard Ware
ICT	Information Communication Technologies
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MySQL	https://www.mysql.com/
OS	Operating System
RDBMS	Relational Database Management System
REST	Representational State Transfer
SNMP	Simple Network Management Protocol
ToC	Table of Contents
URL	Uniform Resource Locator
VM	Virtual Machine
WP	Work package

Executive Summary

This document states the specifics of the implementation of the DOLFIN ICT Performance and Energy Supervisor as implemented in WP3. This document provides a detailed description of the modules that form part of the ICT Performance and Energy Supervisor, together with the interfaces and the interactions with other modules, as well as the implementation explanation of each of the modules of this part of the DOLFIN system.

This document is fed by the deliverables D2.1[1], D2.2[2], D3.1[3] previously submitted by the DOLFIN consortium. Since the initial design (described in the deliverable D3.1[3]), some of the characteristics and implementation techniques for the ICT Performance and Energy Supervisor have changed. The changes were identified aiming to simplify the overall architecture, focusing on the development and software modelling of the DOLFIN scenarios and the strategic objectives. All those changes have been explained within this document.

The ultimate objective of the present deliverable is to provide a thorough report and analysis of the complete set of performance metrics that DOLFIN system utilizes for optimization purposes, including a complete design and implementation description of ICT Performance and Energy Supervisor modules with its internal and external interfaces and the single modules testing results.

1.Introduction

This document explains the implementation details of the ICT Performance and Energy Supervisor module part of the eCOP of DOLFIN system. The main responsibility of the ICT Performance and Energy Supervisor is to interact with the underlying DC subsystems through the DCOs collecting the relevant information to evaluate the metrics and KPIs of DOLFIN. The module implements mechanisms to efficiently retrieve the data from various sub-systems and distributes such information to the consumers that are interested. The ICT Performance and Energy Supervisor implements specific techniques for data analysis and their representation that will be used by other eCOP modules during their optimization activities.

The overall description of DOLFIN architecture was designed and defined in earlier DOLFIN deliverables (D2.1[1], D2.2[2], D3.1[3], D3.2[4], D4.1[5]) and it is depicted here for reference in Figure 1-1. Our aim is to remain through this figure the relationship from a high level point of view between the ICT Performance and Energy Supervisor and the other modules including their interactions.

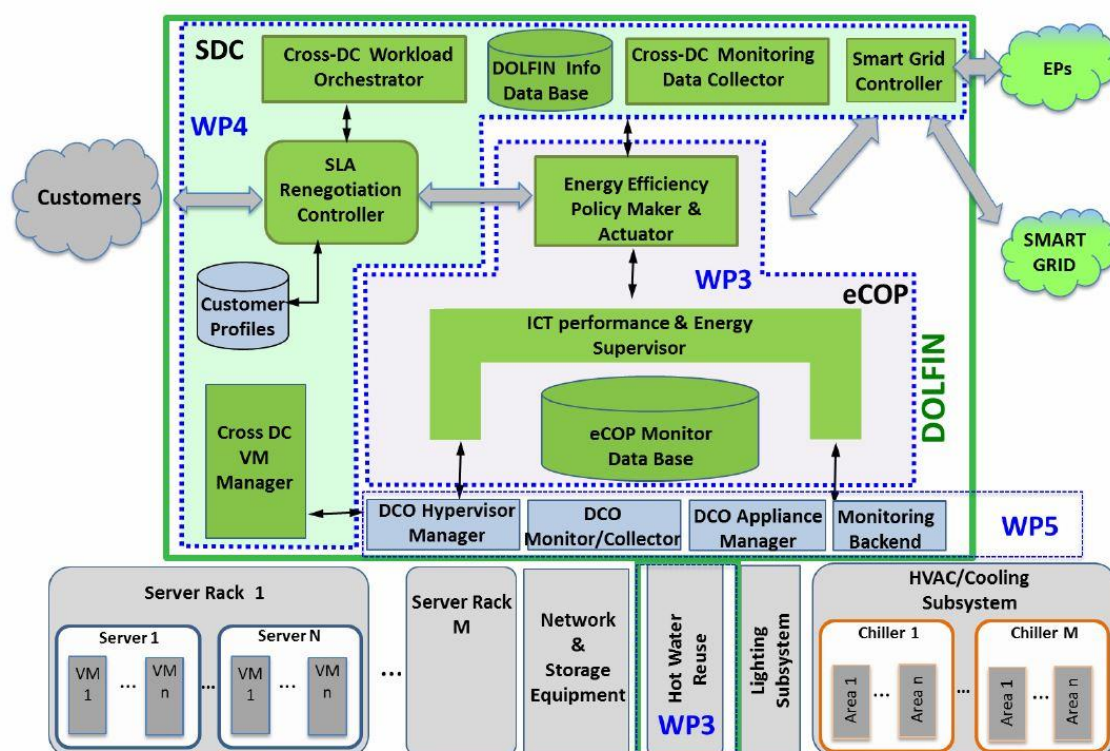


Figure 1-1: DOLFIN overall architecture

The ICT Performance and Energy supervisor is the core optimization of a group of components that implements advanced monitoring, self-evaluation and self-optimisation capabilities. Further specific information about the module described in this document can be consulted in D3.1 [1].

A Git repository have been created to manage all DOLFIN project components, and it could be found at <http://stash.i2cat.net/projects/DOL>. The access to this repository is secured through username and password and hosted by i2CAT.

2. ICT Performance and Energy Supervisor

2.1. Module description

The ICT Performance and Energy Supervisor module has evolved since its original definition in D3.1[3]. Its main responsibilities remain the same but some changes have been introduced in the module implementation:

- The module architecture has changed from event based publisher-subscriber to synchronous producer-consumer based on REST APIs, with the aim to produce a module with a simplified and more stable communication mechanisms.
- In the new model of the DOLFIN architecture, each component is aware of the modules that interact with it and issues calls to their API directly. The ICT Performance and Energy Supervisor core is no longer involved in inter-component communications. Figure 2-1 shows the updated inter-component interactions.
- As a result of the implementation of those changes, the responsibilities of the core module have changed substantially, taking the role of an input data hub and processor. The core waits for data from DCO brokers and passes it through a data processor which then sends the order to calculate a new metric to the Metrics Engine in case this is required.

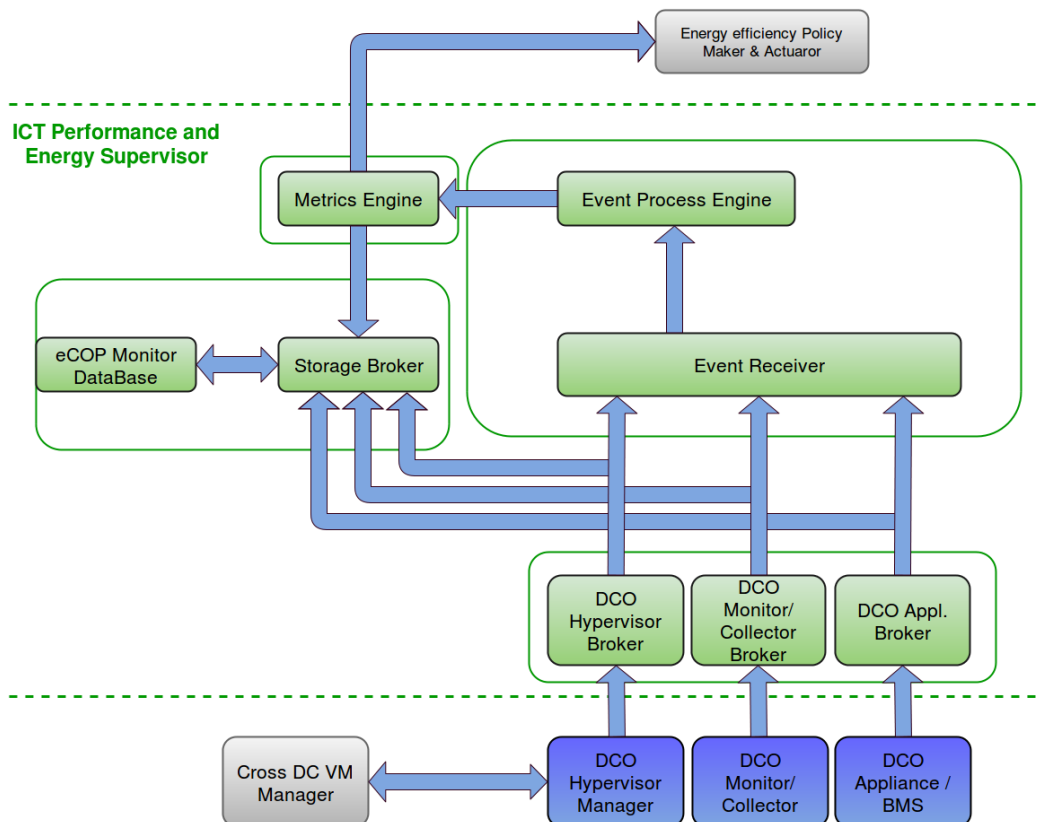


Figure 2-1: ICT Performance and Energy Supervisor architecture

The architectural changes have an impact in the way the module interfaces behave and are implemented although they do not impact the functional operation of the DOLFIN system. Both, the internal and external interfaces are explained in the subsequent sections of this document, starting from the Core, then the Metrics Engine and ICT topology, following the eCOP DB and finally the DCO Brokers.

2.1.1.1. ICT Performance and Energy Supervisor core

The main objective of the ICT Performance and Energy Supervisor Core is to act as a data hub (Event Receiver) that receives all data from multiple DCO brokers and as a data processor (Event Process Engine) that processes the data to identify whether the KPIs should be recalculated to update its values or not. From a functional level point of view, the messages received by the core are treated as events.

Figure 2-2 highlights the Core internal components and interaction amongst them and other modules.

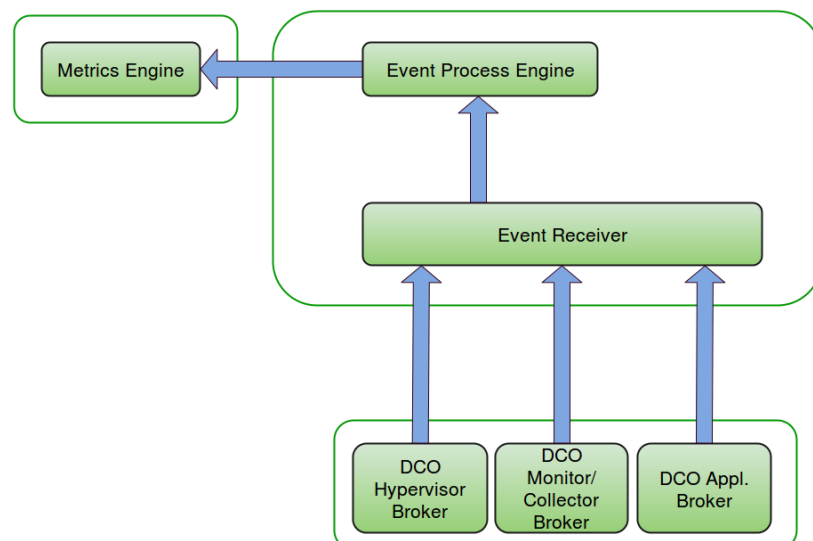


Figure 2-2: ICT Performance and Energy Supervisor Core architecture

2.1.1.1.1. Component responsibilities:

The ICT Performance and Energy Supervisor Core has the following responsibilities:

- Act as an entry point to the events from the DCO brokers.
- Trigger a KPI update calculation in the Metrics Engine whenever this is required.

2.1.1.1.2. Component interfaces

In order to achieve these two responsibilities, the Event Receiver offers a REST API that the DCO Brokers use to report events with a specific format. This API is very similar to the one defined in D3.1[3], the only difference is that it is synchronous and does not provide subscription related methods. The API exposed by the core component is detailed in the table below.

DCO Broker injection API				
Description	Used by DCO Brokers to post event updates toward the ICT Performance and Energy Supervisor core			
Endpoint Name	/api/eventreceiver			
Allowed methods	POST			
Request Parameters	Parameter	Optional	Type	Description
	time	N	Date	Specifies the moment in time when the broker performed the measurement
	resource	N	String	Specifies the type of resource the measurement belongs to
	resourceId	N	String	Identifies the resource the measurement belongs to
	type	N	Enum	Specifies the measurement type. It must take one of the following values: DCPWR, ITPWR, DCBTUcold, HVACPWR, HVACflow, HCACgap, ExpectedDCPWR, ExpectedDCPWRren, BaselineDCPWR, DCPWRReuse, GRIDPWR.
	value	N	Double	Specifies the measurement value
Provides response	NO			
Response Parameters	none			
Requester	DCO Brokers			
Managed errors	400 - Bad request: invalid message content 404 - Not Found 415 - Unsupported media type			
Example	<pre>{ "time": "2015-12-25T08:00:00Z", "resourceId": "1", "resource": "DC-1", "type": "DCPWR", "value": "250" }</pre>			

Table 2-1: DCO Broker injection API

The Event Process Engine, consumes the REST API that is offered by the Metrics Engine component to trigger calculation of a set of KPIs. This API is described in Metrics engine section below.

The Event Receiver and the Event Process Engine interact with each other through direct java calls, not made accessible to other components.

2.1.2. Metrics Engine and ICT Topology

The Metrics Engine produces updated metric values based on pre-defined KPIs that are made available to the ICT Performance and Energy Supervisor core sub-component and to the rest of the DOLFIN modules. This module has been designed to run on-demand, triggered when certain criteria are met in the ICT Performance and Energy Supervisor; it computes the metrics in one run, processing big amounts of data.

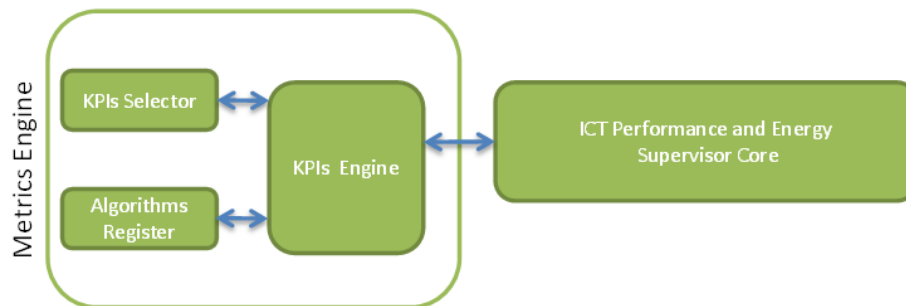


Figure 2-3 - Metrics Engine block diagram

The calculations in the Metrics Engine can be triggered either by an external module requesting a set of synchronous events (e.g., generated by a *cron* entry, that calls the process which performs the operation) or directly called by the user. This calculation can be periodic, as in the first instance, or user-defined as in the second instance.

In a normal working mode, the Metrics Engine is triggered by the Event Process Engine when a specific set of events coming from the DCO Brokers and processed by the Event Receiver Engine requires certain KPIs to be updated. If this is case, the Event Process Engine passes the relevant data to the Metrics Engine to request the activation of the metrics recalculation depending on the type of events received.

The Metrics Engine has been implemented to be sufficiently generic and supports not only a fixed number of metrics, but it also has the possibility to expand them through the implementation of a sort of plugged-in algorithm modules. In this way, new metrics can be incorporated ad-hoc, allowing the developer to concentrate on the algorithms implementation.

At this stage of the implementation, three different metrics (Power Usage Effectiveness, Cooling Effectiveness Ratio, Energy Reuse Effectiveness) are supported and have been validated during the preliminary integration tests, in order to have a set of metrics to perform integration tests with the eCOP Database Broker and other related components:

Power Usage Effectiveness (PUE) -> describes the ratio between the power that has been used by the IT Equipment in contrast to the power used by the rest of the facilities (Indicates how much power is used by the non-IT specific appliances of a DC).

$$PUE(t) = \frac{\text{Total DC Power Consumption}(inst.)}{\text{IT Equipment Power Consumption}(inst.)} = \frac{DCPWR(t)}{ITPWR(t)}$$

Cooling Effectiveness Ratio (CER) -> indicates the ratio between the total heat removed by the cooling system and the electrical energy used for cooling (Evaluates the HVAC effectiveness while cooling the devices and the room).

$$CER(t) = \frac{\text{Total heat removal}(inst.)}{\text{HVAC Power consumption}(inst.)} = \frac{DCBTU_{cold}(t)}{HVACPWR(t)}$$

Energy Reuse Effectiveness (ERE) -> indicates the ratio between the energy consumed in a Data Centre and the re-usage of wasted energy of the same Data Centre.

$$ERE(t) = \frac{DCPWR(t) - DCPWR_{reuse}(t)}{ITPWR(t)}$$

Note that this metric is equivalent to PUE, in the case that the DC does not reuse the wasted energy.

The ICT Topology Graph Database contains an updated version of the data that describes the physical distribution and topology of the DC assets. The data structure is mainly hierarchical, in such a way that can be represented in graph structures (e.g., topology, container, device, measure, etc.).

In addition to the implementation of the Metrics Engine, the Topology module has been developed as an integrated part of Metrics Engine, given their close relationship. With regard to the initial design (in [2]), instead of creating a complete topology DB, bringing along all its abstract structures and loading the computation and operations with the aim of replacing the information already calculated somewhere else, a lightweight real-time topology structure is maintained internally to the Metrics Engine. This approach has been introduced since that some relevant information that initially were in charge of the Topology module, have been moved on the eCOP DB.

2.1.3. eCOP Monitor Database

The eCOP Monitor Database, or hereafter simply eCOP DB, is responsible for the successful persistence and easy extraction of data related to the measurements taken from the DOLFIN monitoring system. The block diagram of this component is depicted in Figure 2-4.

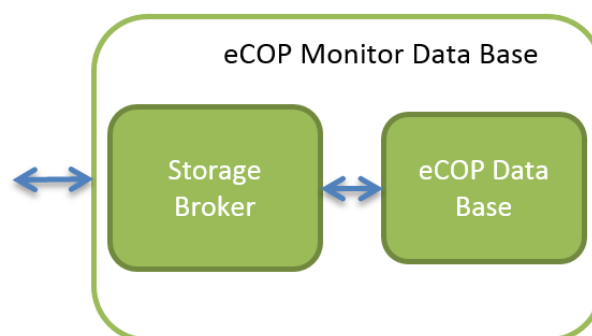


Figure 2-4: The eCOP Monitor Database block diagram.

The eCOP DB consists of a DB for actually storing the data and a RESTful API Broker that wraps this DB so that data extraction is made easier for the rest of DOLFIN's interfacing components.

The core functionality of this component is briefly described in [2], whereas details on its implementation along with the entirety of the supported interfaces are delivered in this document, in paragraph 3.3, page 28.

2.1.4. DCO Brokers

The DCO Brokers form the underlying adaptor layer of the Event Receiver and the Policy Actuator (which implementation will form part of D3.4 and is responsible for the implementation of the actions identified by the Optimizer). In this context, the Policy Actuator translates the optimizer plan into commands, addressed first to the DCO Hypervisor and, subsequently, to the DCO Appliance Manager. The Event Receiver uses the DCO Brokers to poll the infrastructure for status and receive events from infrastructure components

As described in detail in D3.1[3], the DCO Brokers should execute appropriate actions and report the status of execution and correct completion of every issued command, as tracked by the Event Receiver. Any incongruence with the established plan of work will make the sequence abort and an error will be reported. The policy is flagged as failed.

Therefore, the DCO Brokers are responsible to:

1. Decompose Policy Actuator issued commands into Hypervisor or Application specific commands, preserving mapping between the potentially incompatible data models and/or semantics.
2. Abstract low-level transient failures or perform alternate execution paths. As the execution of the decomposed commands is delegated to the DCO Brokers, small inconsistencies and non-fatal errors should be handled at this low-level, so as to simplify the execution of an optimization plan.
3. Report the result of the execution of each command and all DC events to the Event Receiver in a comprehensive manner, performing where necessary the reverse mapping translation (from step 1) between DOLFIN and the underlying DC SW/HW infrastructure components.

According to D3.1 [3], and regarding the available testbeds as described in D5.1[6], the supported DCO Hypervisors by the Brokers include the Openstack/libvirt/QEMU/KVM software stack [7] and VLSP [5]. The DCO Appliance Brokers will be based on the SNMP protocol [8], but given the specificity of HW equipment available at each testbed DC, certain integration effort will be required to tailor the Broker to the precise SNMP calls that DC appliances require.

As a minimum, the set of commands that the Hypervisor and Appliance brokers should be able to perform on account of the Policy Actuator and track for the benefit of the Event Receiver are described in Table 2-2.

Action	Description
VM_migrate	Migrate VM to Server
VM_shift	Postpone the instantiation of VM until time period
DVFS	Scale down the voltage and frequency of Server by given percentage
Serv_term	Enter Server into hibernation mode

Serv_oper	Enter Server into operation mode
Ancil_term	Enter ancillary equipment into hibernation mode
Ancil_oper	Enter ancillary equipment into operation mode
Air_cond_temper	Set air conditioning equipment to given temperature
VM_relocate	Request the relocation of list of VMs to synergetic DC

Table 2-2: Optimizer-Policy Maker Orders

3. Implementation description

3.1. ICT Performance and Energy Supervisor core

3.1.1. Core Modules

The ICT Performance & Energy Supervisor Core is divided in two submodules: the Event Receiver and the Process Engine, each one having a specific differentiated function in the whole DOLFIN system. These two submodules are explained in the following sections.

3.1.1.1. *Event Receiver*

The Event Receiver is the entity that receives the notifications from the DCO Brokers regarding metric measurements. The Event Receiver exposes a REST API, (already described in section 2.1.1). The REST API has been built and published using Apache CXF library, which is an open source services framework providing mechanisms to easily build and develop web services using frontend programming APIs, like JAX-WS and JAX-RS.

The module implementation follows a two-step validation for each received message. The first one, provided by CXF, validates the format of the message according to the developer's annotations included in the model classes. Those annotations allow specifying the format of the message (in our case, JSON) and the expected "schema" in terms of expected fields and possible values associated to them.

The second validation step is done inside the Event Receiver implementation. Once the correct format of the message has been assured by CXF library, the module inspects the type of metric the DCO Broker reported. The Event Receiver contains a list of anticipated metrics that are expected to be received. If the type of metric is unknown, the second validation step ignores the received message, so the process of recalculation in the rest of components of the ICT Performance & Energy Supervisor is not executed. Alternatively, in case the metric is recognised, the Event Receiver sends the order to the Event Process Engine to analyse the metric value.

3.1.1.2. *Event Process Engine*

This module is responsible for processing all metrics messages coming from the Event Receiver. The Event Process Engine contains a set of all KPIs managed by the Metrics Engine, and for each KPI, the list of all kind of metrics involved. Hence, it is responsible for inspecting the metric message and filtering the subset of KPIs that could potentially be affected by the new value. This information is stored in a Map object, where the *key* is the metric type and the *value* is a List instance with the KPIs affected by this metric type.

The output of such process is used to trigger KPIs calculations on the Metrics Engine. The Event Process Engine specifically details which ones should be recalculated by Metrics Engine. This notification is remotely done through the REST Web Service exposed by Metrics Engine module,

described in section 3.1.2 of this deliverable. CXF also provides methods to build Web Services clients, which are used by Event Process Engine to build a REST client for this purpose. In order to develop the client, the module includes classes and annotations inside such classes describing the API nature (expected format, message fields, allowed values, etc.)

3.1.2 API Documentation

The southbound API of the ICT Performance & Energy Supervisor is online documented under the following URL <http://host:port/swagger>. It contains the list of all available REST resources and for each of them, an extensive description of the interactions between the client and these resources. For each REST resource it describes:

- Resource URL.
- HTTP method (GET, PUT, POST, DELETE...)
- Parameter types (Body, Path, Query)
- Expected content-type (JSON, XML, Plain text...)
- Data model expected in body field.
- Returned status codes with description.

The documentation has been automatically generated using Swagger library. Swagger is an open source library providing interactive documentation, client SDK generation and discoverability of REST APIs. It is fully integrated with CXF. The users can obtain all the required information to implement a client for the Event Receiver module and test interactively the API contract by sending sample messages through the graphical client generated by Swagger.

3.1.3 Error Handling

All potential exceptions should be anticipated, detected and resolved as soon as possible during the processes, the core component of the ICT Performance & Energy Supervisor take cares of these potential situations:

- The format of the metric message sent by a DCO Broker is not correct.
- The metric message sent by DCO broker does not contain the metric type, which is a mandatory information.
- The metric type is unknown for the Event Process Engine.
- Metrics Engine is not available.
- Metrics Engine returns an exception after sending the list of affected KPIs.

DCO Brokers only receive an error report in the first two cases, which is automatically done by the CXF library and the annotations included in the classes used to create the web service.

Given that currently, the rest of potential errors do not compromise the module integrity, no other exceptions are launched from the core component, but they will be logged for debugging purposes.

3.1.4 Logging

The ICT Performance & Supervisor core module logs all its activities in a local file. In order to do that, it makes use of the Log4j library. Log4j is an open source logging library for printing log output in local and remote destinations. The core is fully configurable at runtime using external configuration files.

The core module of the ICP Performance & Energy Supervisor provides its own configuration file for the log4j library, performing customized configurations on the log file location, the maximum size and file appenders of the logging, the logging level and the logging messages format.

The Event Receiver and the Event Process Engine submodules log a set of actions to the log file, such as:

- Event Receiver receives a metric report from DCO Broker.
- Event Receiver has processed the metric message.
- The reported metric type is unknown for the Event Process Engine.
- Event Receiver notifies KPI(s) to Metrics Engine.
- Metrics Engine is not available.
- Metrics Engine returns an exception after sending the list of affected KPIs.

Those actions are logged into different levels according to their value or the impact they have in the system.

3.1.5 Module Dependencies

The ICT Performance and Energy Supervisor core module interacts with other DOLFIN's components in order to collect metrics and report the list of KPIs they affect, if any. These interactions have been listed in Table 3-1.

Module Name	Direction	Motivation	Protocol
DCO Brokers	To ICT Performance & Energy Supervisor core	DCO Brokers report performance utilization data and energy consumption, and information about the actual performance of the VMs utilizing resources of the DC IT and non-IT equipment along with information on the actual energy consumption of this equipment.	REST
Metrics Engine	From ICT Performance & Energy Supervisor core	ICT Performance & Energy Supervisor core reports KPIs affected by collected metrics for recalculation.	REST

Table 3-1: ICT Performance and Energy Supervisor Core Interactions

No further direct dependencies exist between the ICT Performance and Energy Supervisor core module and other DOLFIN components.

3.1.6 Deployment requirements and testing

The outcome of the compilation of the core module is a Java Archive (JAR). This file format is S.O. independent, meaning it is compatible with Windows, Linux and MacOS. Java Runtime Environment (JRE) is the only mandatory set of libraries that the user should install in its computer in order to execute the module.

The program requires a configuration file to load some properties. The implementation contains a sample configuration file with all the required properties called “eventprocess.conf”. The administrator should configure the base URL of the southbound API and also the IP address of the Metrics Engine in order to build its client.

Once installed, the ICT Performance & Energy Supervisor core could be executed by running following command:

```
$ java -jar ./target/event-process-engine-0.0.1-SNAPSHOT.jar PATH_TO_CONFIG_FILE
```

In case Linux based OS is being used, simplify the process by making use of the provided bash scripts: start.sh and stop.sh.

```
$ start.sh PATH_TO_CONFIG_FILE
```

All runtime dependencies are included inside the JAR file, so it doesn't require to install any additional library on the system. The installation can be tested by accessing the REST API reference page. Bear in mind that BASE_URL was set in configuration indexed when starting the server.

```
http://server\_base\_url/swagger
```

In order to stop the execution, just interrupt the software process or, in a Linux based machine, execute the stop.sh script.

3.2. Metrics Engine and ICT Topology

This module is the DOLFIN eCOP Database Broker, providing RESTful access to the contents of the DOLFIN eCOP DB.

Metrics Engine (and ICT Topology included) sources are written in Python 3.x. The two main functions of this module are:

- a Web Server, which waits for metrics calculation requests

- a REST API compliant client, which triggers the server for metric calculation

3.2.1. Basic concepts

The Metrics Engine maintains the information of all possible metrics, which can be calculated and stored in the eCOP DB to achieve different energy efficiency and resource management goals.

Metrics affecting all DC parts are calculated based on the values contained in the eCOP DB, following the procedures specified in the Algorithms Register component. This involves values collection and some sort of action in the modules required to calculate them.

The Metrics Engine web server waits for POST requests coming from the trigger component; after picking DC data values from the eCOP DB, it performs its computations to get aggregated values, calculates the metrics and then it stores the values into the eCOP DB.

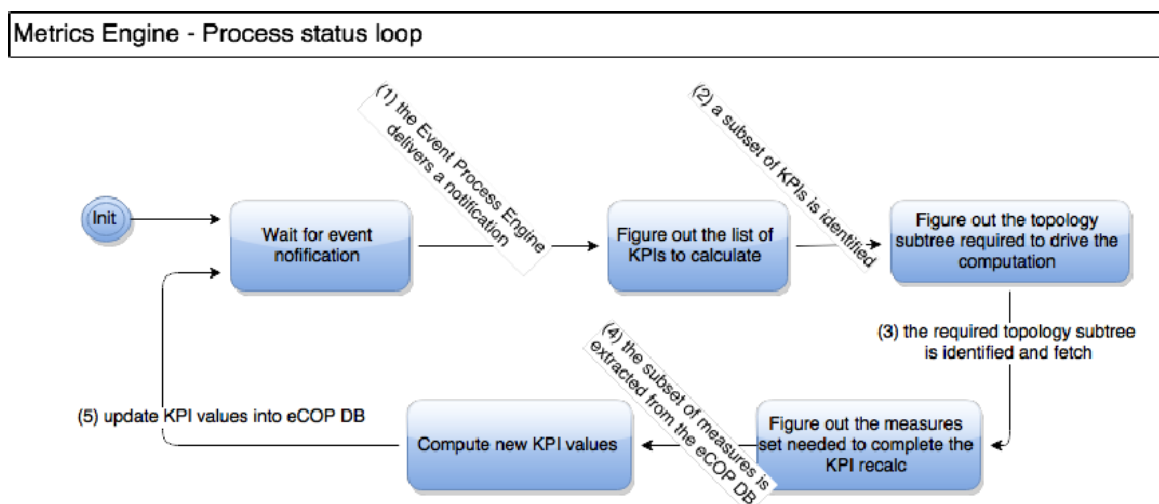


Figure 3-1 - Metrics Engine internal workflow

The trigger component is a simple module which executes a POST request to the server URL defined by the REST API specifications, activating the new metric computation.

A list of Metrics Engine REST APIs is provided in Table 3-2:

API Service Endpoint URL	Description
<metrics_engine_URL>/v1/kpi/	Get a list of supported KPI
<metrics_engine_URL>/v1/kpi/{kpi_name}	Get the value of a specified KPI
<metrics_engine_URL>/v1/kpi/recompute/{kpi_name}	Trigger the Metrics Engine to recompute the specific KPI value

Table 3-2- Metrics Engine RESTfull APIs

The Metrics Engine also communicates with the eCOP Database Broker component to get the required data to calculate aggregates and metrics and to store the new KPIs values.

Figure 3-2 states the list of main eCOP DB REST APIs used by the Metrics Engine:

measurement_types		Show/Hide	List Operations	Expand Operations	Raw
measurements		Show/Hide	List Operations	Expand Operations	Raw
GET	/api/measurements/				
POST	/api/measurements/				
GET	/api/measurements/{pk}/				
PUT	/api/measurements/{pk}/				
PATCH	/api/measurements/{pk}/				
DELETE	/api/measurements/{pk}/				
GET	/api/measurements/by-type/{type}/				Get a generic measurement based on the type of the measurement
GET	/api/measurements/by-type/{type}/{start}/{end}/				Get a generic measurement based on the type of the measurement and a date range
GET	/api/measurements/by-type-resource/{type}/{resource}/				Get a generic measurement based on the type of the measurement and a resource type
GET	/api/measurements/by-type-resource/{type}/{resource}/{id}/				Get a generic measurement based on the type of the measurement for a specific resource
GET	/api/measurements/by-type-resource/{type}/{resource}/{id}/{start}/				Get a generic measurement for a specific resource and a time range starting from now and ending {end} hours before
GET	/api/measurements/by-type-resource/{type}/{resource}/{id}/{start}/{end}/				Get a generic measurement based on the type of the measurement for a specific resource and for a time range
GET	/api/measurements/by-resource/{resource}/				Get a generic measurement based on the resource type
GET	/api/measurements/by-resource/{resource}/{id}/				Get a generic measurement for a specific resource
GET	/api/measurements/by-resource/{resource}/{id}/{start}/{end}/				Get a generic measurement for a specific resource and a time range
GET	/api/measurements/aggregate/power/{type}/{time}/				Get the instantaneous power consumption of a(part of a) DC at a specific time
GET	/api/measurements/aggregate/energy/{type}/{start}/{end}/				Get the energy consumption of a(part of a) DC during a specific time range

Figure 3-2 - eCOP DB APIs used by Metrics Engine

These APIs are used by the Metrics Engine module that has the task to interface with the eCOP DB component, in order to retrieve the data needed for its computations.

3.2.2. Module Dependencies

Metrics Engine module gets its input from the Event Process Engine and the eCOP DB. The Event Process Engine is part of the ICT Performance and Energy Supervisor core module and has been thoroughly explained in section 3.1. The ICT Performance and Energy Supervisor Core exposes a REST API consumed by other modules such as the DCO Hypervisor Broker, the DCO Monitor/Collector Broker and the DCO Appliance Broker, reporting data about performance utilization and energy consumption. If any of the reported information affects relevant KPIs, the Event Process Engine notifies the Metrics Engine which ones are the KPIs that should be recalculated.

The eCOP Database Broker, provides RESTful access to the contents of the DOLFIN eCOP DB. The diagram in Figure 3-3 explains the interaction between the Metrics Engine and eCOP DB components:

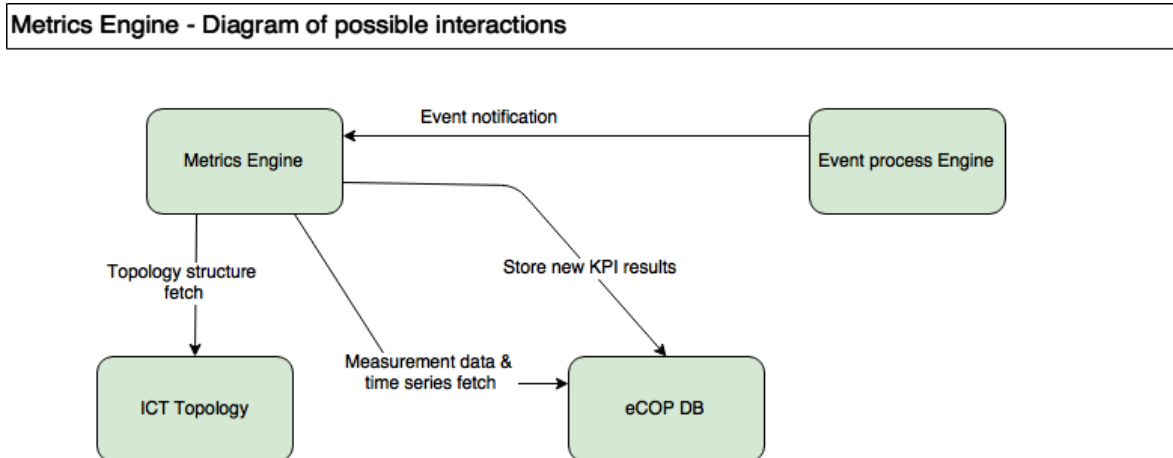


Figure 3-3 - Metrics Engine interactions diagram

In order to download and install these two required components, please refer to the instructions in the repository:

- http://stash.i2cat.net/scm/dol/ecop_event_process_engine.git
- <http://stash.i2cat.net/scm/dol/ecop-db-broker.git>

See included README.md file for installation.

3.2.3. Deployment and installation

Below you may find the installation instructions for Ubuntu systems. Please note that this installation procedure has been verified on Ubuntu 14.04.x x86_64 systems with eCOP DB installed.

Sources are available on a Git repository at:

http://stash.i2cat.net/scm/dol/ecop_metrics_engine.git

Preparing system:

```

$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install git python3.4 python-pip python-dev mysql-
server libmysqlclient-dev build-essential
  
```

Cloning Git repository for Metrics Engine component:

```

$ git clone http://stash.i2cat.net/scm/dol/ecop_metrics_engine.git
  
```

Cloning Git repository for DOLFIN Utils modules:

```
$ git clone http://stash.i2cat.net/scm/dol/dolfin_utils.git
```

Preparing installation directory:

```
$ sudo mv ecop_metrics_engine /opt/  
$ cd /opt/ecop_metrics_engine
```

Installing Python dependencies using PIP tool:

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

Executing post installation triggers:

```
$ python /opt/ecop_metrics_engine/bootstrap_metrics.py
```

This command will insert a `crontab` line to periodically trigger metrics:

```
$ add_cron_entry trigger_kpieng \  
    "*/2 * * * * python /opt/ecop_metrics_engine/trigger_metrics.py  
&& /var/log/trigger_metrics.log"
```

Launching the Metrics Engine server:

```
$ sudo -m dolfin.kpieng.server
```

When launching the Metrics Engine server some useful option are available:

- Use the `--debug` parameter to enable better error reporting and also to start the Swagger UI, available at <http://127.0.0.1:8081/docs/>
- Other alternatives:
 - `--ecop_db_access_url <URL>`
 - `--ecop_db_access_user <ECOP_DB_ACCESS_USER>`
 - `--ecop_db_access_pass <ECOP_DB_ACCESS_PASSWORD>`
 - `--help`
 - `--logging=debug`
 - `--server_port=<PORT_NUMBER>`

- `--server_addr=0.0.0.0` (to bind on all interfaces)

The server component should be running (by default) at port 8081 of your machine. You may test it by issuing the following command to see if the documentation page is properly fetched:

```
$ curl localhost:8081/docs
```

To complete the installation some prerequisites should be satisfied:

- Git
- Python 3.4+
- MySQL
- `dolphin-utils` python package (will be explicitly required during the install phase)

Other dependencies included in `requirements.txt` file (automatically installed using `'pip install'` command, previously mentioned):

- `iso8601` (v0.1.11)
- `requests` (v2.8.1)
- `six` (v1.10.0)
- `tornado` (v4.2.1)
- `scipy` (v0.16.1)
- `numpy` (v1.10.1)

3.2.4. Testing the installation

To test the Metrics Engine installation and configuration assuming that the procedure described in the previous paragraph has been followed (remembering to launch `dolphin.kpieng.server` with `--debug` option), one should visit the page

`http://<METRICS_ENGINE_HOSTNAME_OR_IP>/docs`

which constitutes the landing page of the online documentation module (based on Swagger tool) supporting the Metrics Engine operation (see Figure 3-4).

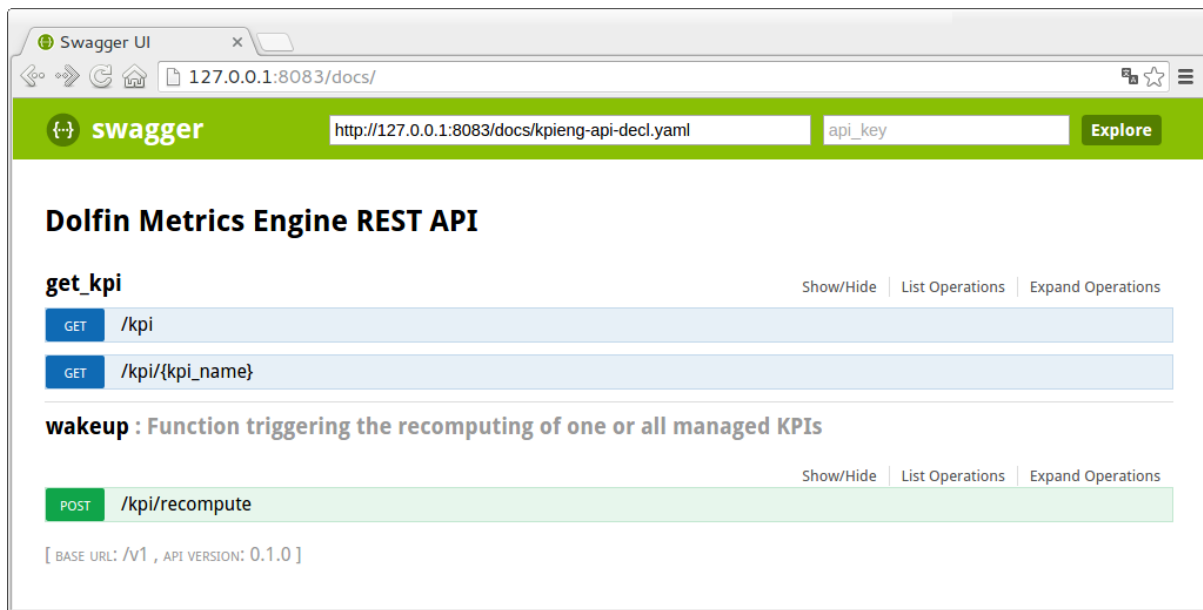


Figure 3-4 - Metrics Engine Swagger web interface

User can use *get_kpi* APIs to test the Metrics Engine server activity (for example triggering the /kpi GET command). Additionally, instead of using Swagger UI to test the metrics recomputing, user could open a console and launch the following command

```
$ python trigger_metrics.py
```

This command triggers the metrics showing no output in case of correct installation; it will return an output only in the case of raising an error.

3.3. eCOP Monitor Database

The eCOP DB has been implemented on top of a MySQL¹ database, running in an Ubuntu Linux² 14.04.3 instance. The eCOP DB Broker implementation has been based on the Django Framework³ using Python⁴ as the programming language for the development.

The architecture of the eCOP DB and the associated Broker is the one documented in the deliverable D3.1 [1]. The data model documented in D3.1 [1] has slightly changed and has been greatly enriched, in an attempt to provide a feature-rich data logging experience to the users of DOLFIN. In the next paragraphs, the installation and configuration instructions both for the eCOP DB and the associated broker are presented, followed by the presentation of the enriched eCOP DB Broker data models.

¹ MySQL website, <https://www.mysql.com/>

² Ubuntu Linux website, <http://www.ubuntu.com/>

³ Django Framework website, <https://www.djangoproject.com/>

⁴ Python website, <https://www.python.org/>

3.3.1. Installation and Configuration of the DB

The eCOP DB has been designed as a relational DB using the MySQL DB as the RDBMS of choice⁵. To install MySQL on an Ubuntu Linux installation, one should issue the following commands:

```
# Update the system software
$ sudo apt-get update
$ sudo apt-get upgrade
# Actually install MySQL
$ sudo apt-get install mysql-server
```

After having installed the RDBMS, one needs to secure the DB installation:

```
$ sudo mysql_secure_installation
```

Next, the DB schema creation should take place by issuing the following commands:

```
$ mysql -uroot -p
Enter password:
mysql> create schema db;
mysql> exit;
```

The actual DB schema declaration along with the tables required for the eCOP DB to operate as expected may be found in DOLFIN's source code repository (granted availability of a valid set of credentials) at <http://stash.i2cat.net/projects/DOL/repos/ecop-db-broker/browse/db/ecop.sql>.

In order to generate the proper DB tables structure using this file, assuming that it has been downloaded under the directory `~`, the following command should be issued⁶:

```
$ mysql -uroot -p < ~/ecop.sql
```

After this step Figure 3-5 depicts the eCOP DB schema including the authentication tables, excluding the default Django framework administration tables.

Note that although the aforementioned schema declaration and installation procedure is valid, the recommended way of generating the eCOP DB tables is through the configuration of the eCOP DB Broker, documented in the following paragraphs.

⁵ The forthcoming installation and configuration analysis is also valid (with minimal differentiations) when using MySQL variants like MariaDB.

⁶ As explained next, the proper way of generating the right DB schema for the eCOP DB is not via the provided SQL script but, rather, through the configuration of the ECOP DB Broker presented in paragraph 3.3.2.7.

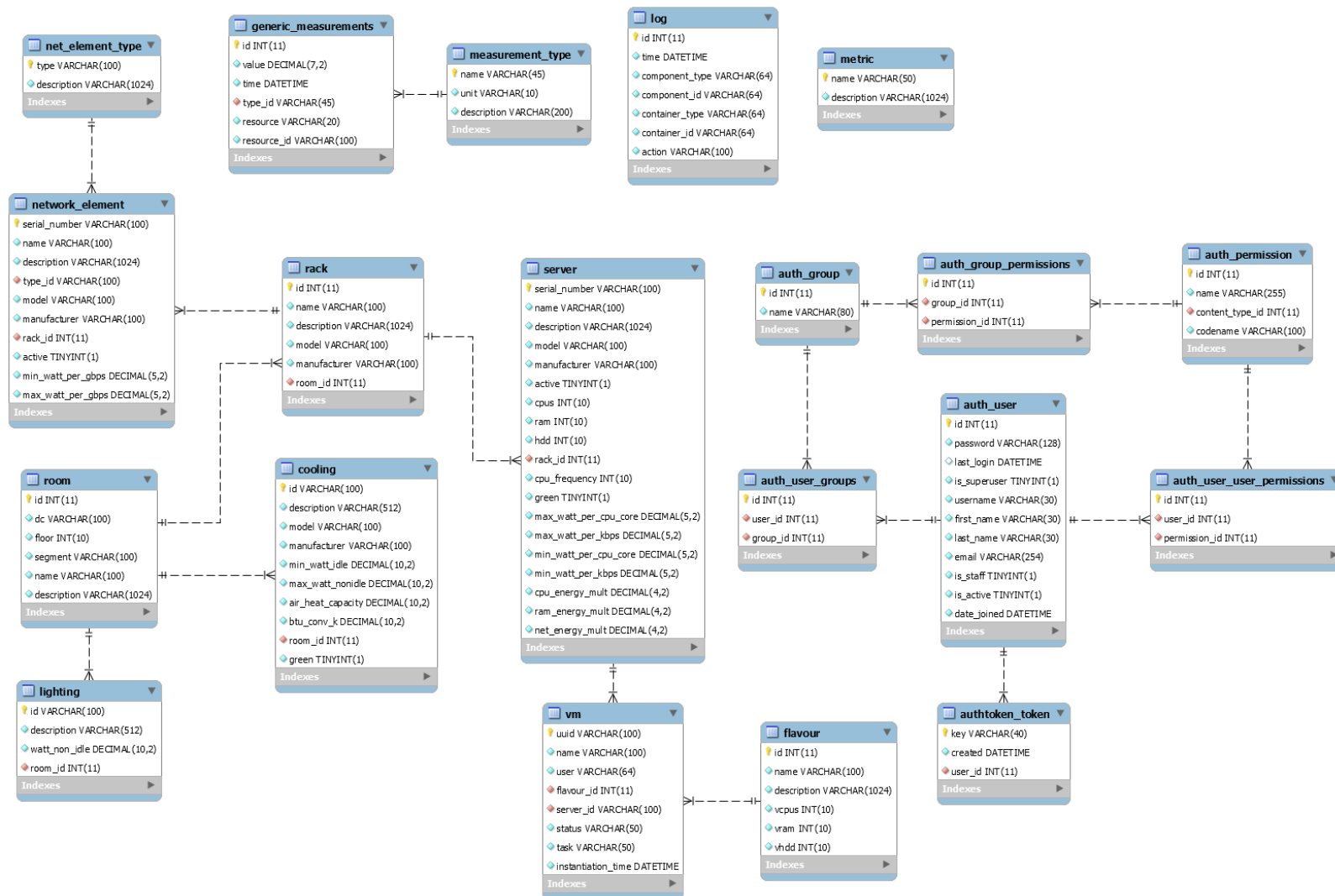


Figure 3-5: Database schema of the eCOP DB, including the authentication tables, excluding the default Django framework administration tables.

3.3.2. Installation and configuration of the eCOP DB Broker

Assuming that the eCOP DB has been successfully installed and configured (see paragraph 3.3.1 for details and discussion), for the installation of the eCOP DB Broker the following commands should be issued:

```
# Update the system software
$ sudo apt-get update
$ sudo apt-get upgrade
# Install necessary software dependencies to get and build the broker
$ sudo apt-get install git python-pip python-dev libmysqlclient-dev \
  build-essential
# Get the eCOP DB Broker code
$ git clone http://stash.i2cat.net/scm/dol/ecop-db-broker.git
# Move the code to /opt and install the necessary software dependencies
$ sudo mv ecop-db-broker/ /opt/ecop_db_broker
$ cd /opt/ecop_db_broker
$ sudo pip install -r requirements.txt
```

After the successful installation of all the software dependencies, the eCOP DB Broker should be configured in order to connect to the eCOP DB. All configuration options are located in the file `/opt/ecop_db_broker/ecop_db_broker/settings.py`. The list of options that should be configured in order to get a valid instance of the eCOP DB Broker is documented in the following paragraphs.

3.3.2.1. Configuration of the DB connection

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'ecop',
        'USER': 'root',
        'PASSWORD': '<password>',
        'HOST': 'localhost',
        'PORT': '3306'
    }
}
```

The `DATABASES` section of the Broker settings should be configured to use the DB installation described in paragraph 3.3.1. If a username other than 'root' is used⁷, this username should be placed instead of 'root'. Evidently, the password of the DB user should be entered instead of

⁷ This is actually also the recommended way for accessing the DB contents.

'<password>'. If the DB and the Broker are not running in the same server, then the HOST and PORT options should be appropriately configured to match the configuration of the DB.

3.3.2.2. *Configuration of the login redirect URL*

```
LOGIN_REDIRECT_URL = '/dolphin/db/html/login/'
```

The LOGIN_REDIRECT_URL setting should be changed if the Broker is running under a BASE URL path other than `http://<BROKER_HOSTNAME_OR_IP>/dolphin/db/`. When using the default configuration presented in this document, this setting should not be changed.

Indicatively, if the Broker is running under the base URL path `http://<BROKER_HOSTNAME_OR_IP>/`, then the LOGIN_REDIRECT_URL setting should be set to `'/html/login/'`.

3.3.2.3. *Configuration of the static files URL*

```
STATIC_URL = '/dolphin/db/static/'
```

The STATIC_URL setting should be changed if the Broker is running under a BASE URL path other than `http://<BROKER_HOSTNAME_OR_IP>/dolphin/db/`. When using the default configuration presented in this document, this setting should not be changed.

Indicatively, if the Broker is running under the base URL path `http://<BROKER_HOSTNAME_OR_IP>/`, then the STATIC_URL setting should be set to `'/static/'`.

3.3.2.4. *Configuration of the SWAGGER online documentation module*

```
'api_path': '/dolphin/db/',
'base_path': '<BROKER_HOSTNAME_OR_IP>/dolphin/db/docs/',
```

These two settings should be changed if the Broker is running under a BASE URL path other than `http://<BROKER_HOSTNAME_OR_IP>/dolphin/db/`. When using the default configuration presented in this document, the two settings should not be changed.

Indicatively, if the Broker is running under the base URL path `http://<BROKER_HOSTNAME_OR_IP>/`, then the two settings should be configured as follows:

```
'api_path': '/',
'base_path': '<BROKER_HOSTNAME_OR_IP>/docs/',
```

3.3.2.5. *Configure CORS headers*

```
CORS_ORIGIN_ALLOW_ALL = True
CORS_ORIGIN_REGEX_WHITELIST = ('^(http?://)?192\.168\.1\.\d+\w*$',)
```

If one needs to configure the eCOP DB Broker to allow CORS, then the relative CORS headers should be included in the responses of the Broker. In order to configure the respective Broker behaviour, one should appropriately configure the above two settings⁸.

3.3.2.6. *Integration with Apache server and configuration [optional, recommended]*

In order to deploy the eCOP DB Broker with Apache and mod_wsgi, the following commands should be issued:

```
# Update the system software
sudo apt-get update
sudo apt-get upgrade
# Actually install Apache and mod_wsgi
sudo apt-get install apache2 libapache2-mod-wsgi
```

After having installed the necessary software, the file `/etc/apache2/sites-enabled/ecop-db-broker.conf` should be created and, then, edited in order to let Apache know of the eCOP DB Broker installation. The contents of the file should be as follows.

```
WSGIPassAuthorization On
WSGIScriptAlias /dolphin/db /opt/ecop_db_broker/ecop_db_broker/wsgi.py

Alias /dolphin/db/static/ /opt/ecop_db_broker/static/

<Directory /opt/ecop_db_broker/static>
    Require all granted
</Directory>

<Directory /opt/ecop_db_broker/ecop_db_broker>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>
```

Evidently, if the eCOP DB Broker should be configured to run in a base URL path, the `WSGIScriptAlias` and the static `Alias` settings should change accordingly⁹.

⁸ For more information on how to configure CORS in Django, the interested reader is request to refer to <https://github.com/ottoyiu/django-cors-headers>.

3.3.2.7. Model Synchronization with the DB

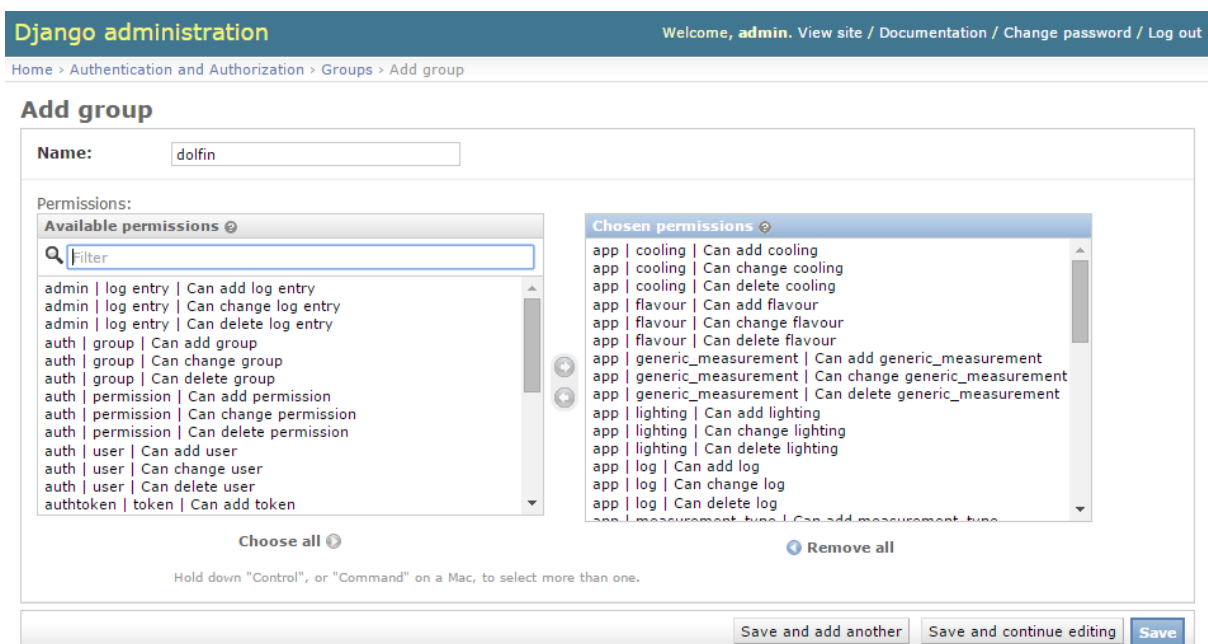
After the successful configuration of the eCOP DB Broker software, the following commands should be issued in order to synchronize the Broker Models with the DB.

```
# Synchronize the DB with the Broker models
# Remember to create a superuser
$ sudo python manage.py syncdb
$ sudo python manage.py makemigrations
$ sudo python manage.py migrate
# [Optional] Restart the http service if configured to run with Apache
$ sudo service apache2 restart
```

The Broker should now be ready for use.

3.3.2.8. Adding users and groups for use through the eCOP DB Broker

With regard to the configuration part, it is assumed that a superuser has been created during the synchronization with the of the Broker models with the DB (see previous paragraph). Then, one should navigate from a browser to the eCOP DB Broker administration page, namely http://<BROKER_HOSTNAME_OR_IP>/dolfin/db/admin/, click the "Groups" link and create a new group with the name 'dolfin' and choose all permissions starting with "app", as depicted in Figure 3-6.



Django administration Welcome, admin. View site / Documentation / Change password / Log out

Home > Authentication and Authorization > Groups > Add group

Add group

Name:

Permissions:

Available permissions

Q Filter

- admin | log entry | Can add log entry
- admin | log entry | Can change log entry
- admin | log entry | Can delete log entry
- auth | group | Can add group
- auth | group | Can change group
- auth | group | Can delete group
- auth | permission | Can add permission
- auth | permission | Can change permission
- auth | permission | Can delete permission
- auth | user | Can add user
- auth | user | Can change user
- auth | user | Can delete user
- authtoken | token | Can add token

Choose all

Chosen permissions

- app | cooling | Can add cooling
- app | cooling | Can change cooling
- app | cooling | Can delete cooling
- app | flavour | Can add flavour
- app | flavour | Can change flavour
- app | flavour | Can delete flavour
- app | generic_measurement | Can add generic_measurement
- app | generic_measurement | Can change generic_measurement
- app | generic_measurement | Can delete generic_measurement
- app | lighting | Can add lighting
- app | lighting | Can change lighting
- app | lighting | Can delete lighting
- app | log | Can add log
- app | log | Can change log
- app | log | Can delete log

Remove all

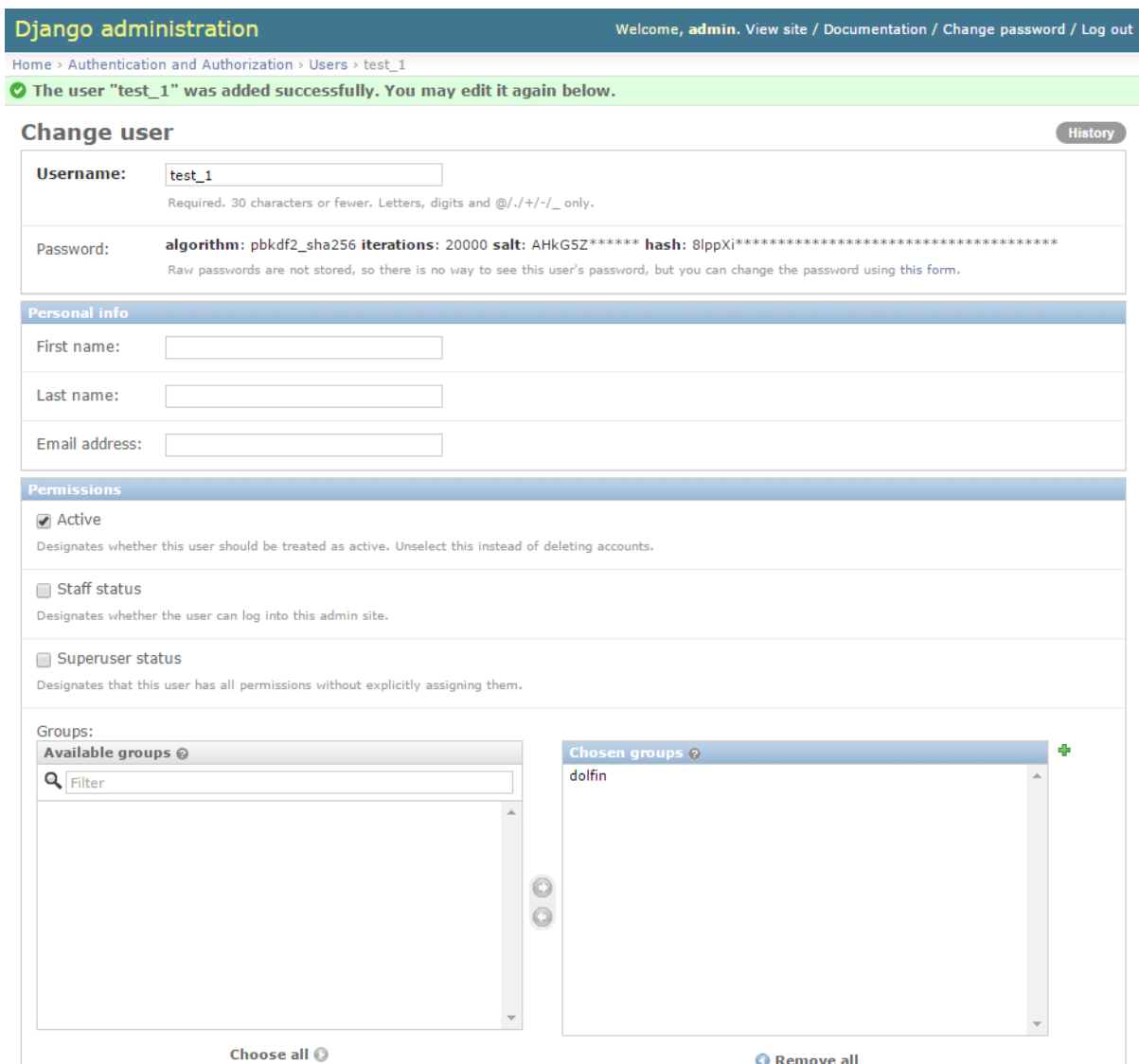
Hold down "Control", or "Command" on a Mac, to select more than one.

Save and add another Save and continue editing Save

Figure 3-6: Dolfin group creation through the administration page

⁹ For information on how to configure Apache and mod_wsgi to work with Django, the interested reader is requested to refer to <https://docs.djangoproject.com/en/1.9/howto/deployment/wsgi/modwsgi/>.

Any user willing to use the eCOP DB Broker should belong to this group. To create a new eCOP DB Broker user, navigate to the administration page, click on 'Users' and then 'Add User'. Upon creation of the user, edit the user to add her into the 'dolfin' group as depicted in Figure 3-7.



Django administration Welcome, **admin**. [View site](#) / [Documentation](#) / [Change password](#) / [Log out](#)

[Home](#) > [Authentication and Authorization](#) > [Users](#) > [test_1](#)

✓ The user "test_1" was added successfully. You may edit it again below.

Change user History

Username:
Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **algorithm:** pbkdf2_sha256 **iterations:** 20000 **salt:** AHkG5Z***** **hash:** 8lppXi*****
Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name:

Last name:

Email address:

Permissions

☒ **Active**
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☐ **Staff status**
Designates whether the user can log into this admin site.

☐ **Superuser status**
Designates that this user has all permissions without explicitly assigning them.

Groups:

Available groups

Chosen groups

dolfin

[Choose all](#)

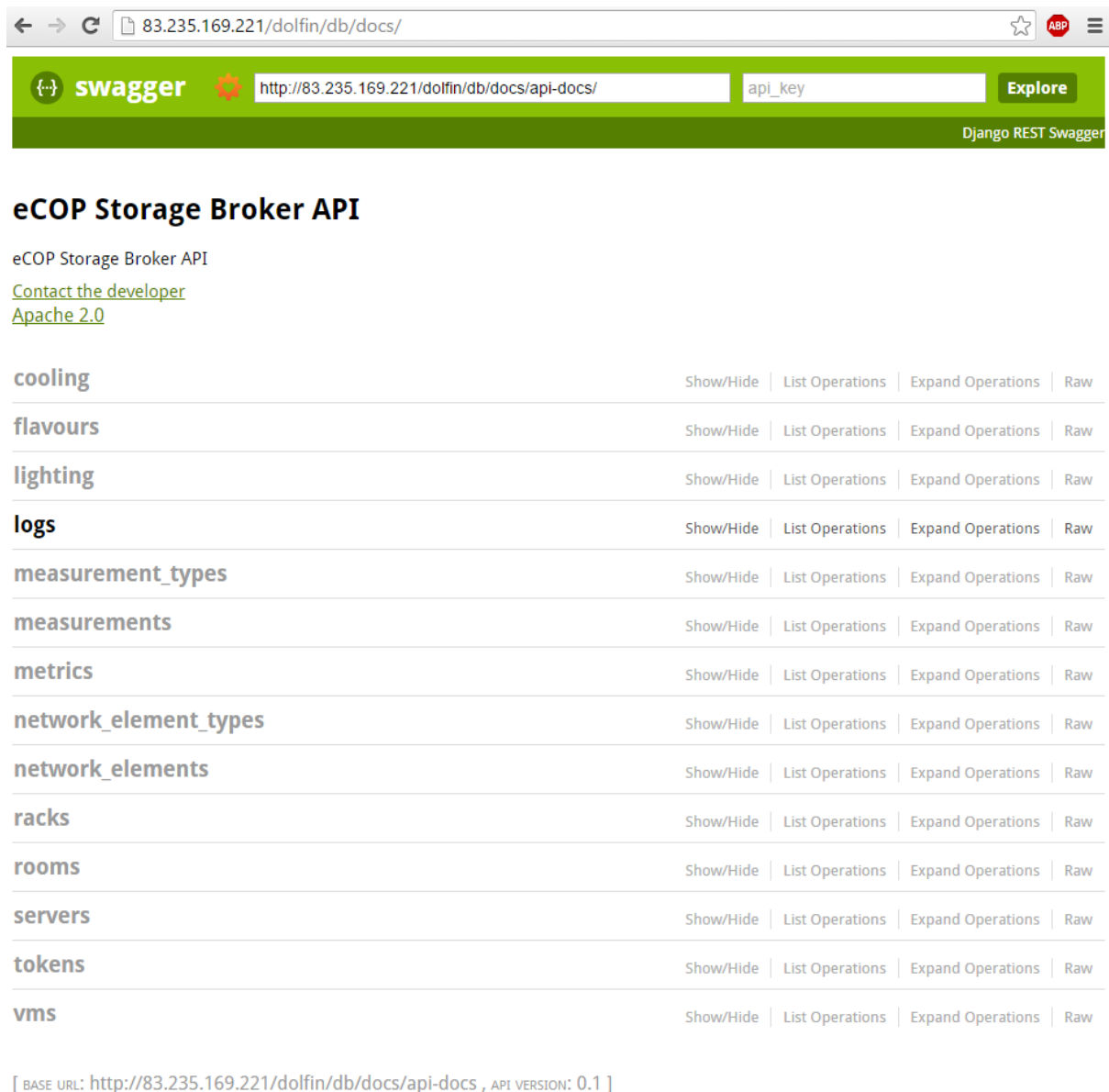
[Remove all](#)

Figure 3-7: eCOP DB Broker user creation through the administration page

3.3.3. Testing the configuration of the eCOP DB Broker

In order to test the eCOP DB Broker installation and configuration assuming that the default configuration presented in the previous paragraphs has been followed, one should visit the page http://<BROKER_HOSTNAME_OR_IP>/dolfin/db/docs/, which constitutes the landing page of the online documentation module supporting the eCOP DB Broker operation, depicted in Figure 3-8.

Find the instructions about how to use the online documentation module, in the page of Swagger-UI¹⁰.



eCOP Storage Broker API

eCOP Storage Broker API

[Contact the developer](#)
[Apache 2.0](#)

cooling	Show/Hide	List Operations	Expand Operations	Raw
flavours	Show/Hide	List Operations	Expand Operations	Raw
lighting	Show/Hide	List Operations	Expand Operations	Raw
logs	Show/Hide	List Operations	Expand Operations	Raw
measurement_types	Show/Hide	List Operations	Expand Operations	Raw
measurements	Show/Hide	List Operations	Expand Operations	Raw
metrics	Show/Hide	List Operations	Expand Operations	Raw
network_element_types	Show/Hide	List Operations	Expand Operations	Raw
network_elements	Show/Hide	List Operations	Expand Operations	Raw
racks	Show/Hide	List Operations	Expand Operations	Raw
rooms	Show/Hide	List Operations	Expand Operations	Raw
servers	Show/Hide	List Operations	Expand Operations	Raw
tokens	Show/Hide	List Operations	Expand Operations	Raw
vms	Show/Hide	List Operations	Expand Operations	Raw

[BASE URL: <http://83.235.169.221/dolphin/db/docs/api-docs> , API VERSION: 0.1]

Figure 3-8: Default view of the online documentation module of the eCOP DB Broker.

While testing the Broker API the following commands should be issued:

```
# Install curl command
$ sudo apt install curl
$ curl -X POST -H "Content-Type: application/json" -d '{"username":
"test","password": "test"}'
# Here comes the response with the token
{"token": "581e76b175a6b48c78bb39ee598284e7183affdb", "created_at": "2015-12-
18T14:05:36.393866+00:00", "expires_at": "2015-12-19T14:05:36.393866+00:00"}
```

¹⁰ Swagger UI webpage, <http://swagger.io/swagger-ui/>.

```
# Request the racks of the DC, using the token already acquired
$ curl -H "Authorization: Token 581e76b175a6b48c78bb39ee598284e7183affdb"
http://<BROKER_HOSTNAME_OR_IP>/dolfin/db/api/racks/
# No racks have been added yet in a vanilla installation
[]
```

The use of the online documentation system is documented in the following paragraphs.

3.3.4. Using the eCOP DB Broker

The eCOP DB Broker currently supports 94 API service-oriented endpoint URLs (services), 37 management ones (the Django defaults¹¹) and 8 endpoints related to the eCOP DB Dashboard.

3.3.4.1. *Using the online documentation module*

As already documented and depicted in Figure 3-8, the eCOP DB Broker features an online documentation module based on the open Swagger framework¹². In order to acquire a token from the online documentation module, one should navigate with a web-browser at http://<BROKER_HOSTNAME_OR_IP>/dolfin/db/docs/ and click on the tokens endpoint. The view should be as in Figure 3-9.

¹¹ Django admin webpage, <https://docs.djangoproject.com/en/1.9/ref/contrib/admin/>

¹² Swagger webpage, <http://swagger.io/>

tokens Show/Hide List Operations Expand Operations Raw

POST `/api/tokens/` Get an authentication token

Implementation Notes
Get an authentication token

Response Class
[Model](#) [Model Schema](#)
object

Response Content Type: `application/json`

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authentication Request	<pre>{ "username": "a_username", "password": "a_password" }</pre>	The authentication request to grant	body	Model Model Schema <pre>{ "username": "", "password": "" }</pre>

Parameter content type: `application/json`

Error Status Codes

HTTP Status Code	Reason
200	OK
204	No content
301	Moved permanently
401	Unauthorized
403	Forbidden
404	Content not found

[Try it out!](#) [Hide Response](#)

Figure 3-9: The default view of the online documentation module for acquiring an authorization token

As it can be observed in Figure 3-9, through the online documentation module one can see the HTTP method supported by the particular relative API service endpoint URL (grey box), the API service description (yellow box) and the HTTP error status codes (blue box). As this is a POST API endpoint, the data model of the entity to be posted (in JSON format) is depicted through the model schema description (red box). When clicking on it, the request body (green box) is filled in with an empty entity structure to be POSTed. When hitting the Try out button, the request is directed to `http://<BROKER_HOSTNAME_OR_IP>/dolfin/db/api/tokens/` (note that the path `/api/tokens/` is the one documented as the relative API service endpoint URL (grey box)). The token acquired should be, then, put into the `api_key` field of the page (located on the upper right part of it, as depicted in Figure 3-8).

When clicking on another, more complex endpoint e.g. to get the power measurements of a particular rack during a time frame, the online documentation system also documents the various expected path-URL parameters required (blue box) and the expected response format (red box), as shown in Figure 3-10.

GET `/api/measurements/by-type-resource/{type}/{resource}/{id}/{start}/{end}/`

Get a generic measurement based on the type of the measurement for a specific resource and for a time range

Implementation Notes

Get a generic measurement based on the type of the measurement for a specific resource and for a time range

Response Class

Model | Model Schema

```
GenericMeasurementSerializer {
  id (integer),
  value (string),
  time (string),
  type (string),
  resource (string),
  resource_id (string)
}
```

Response Content Type:

Parameters

Parameter	Value	Description	Parameter Type	Data Type
type	<input type="text" value="power"/>	The name of the measurement type	path	string
resource	<input type="text" value="rack"/>	The type of the resource (e.g. room, server, etc)	path	string
id	<input type="text" value="3"/>	The id of the specific resource	path	string
start	<input type="text" value="2015-12-15T00:00:00Z"/>	The start time of the time range, in ISO8601 format	path	string
end	<input type="text" value="2015-12-16T00:00:00Z"/>	The end time of the time range, in ISO8601 format	path	string

Error Status Codes

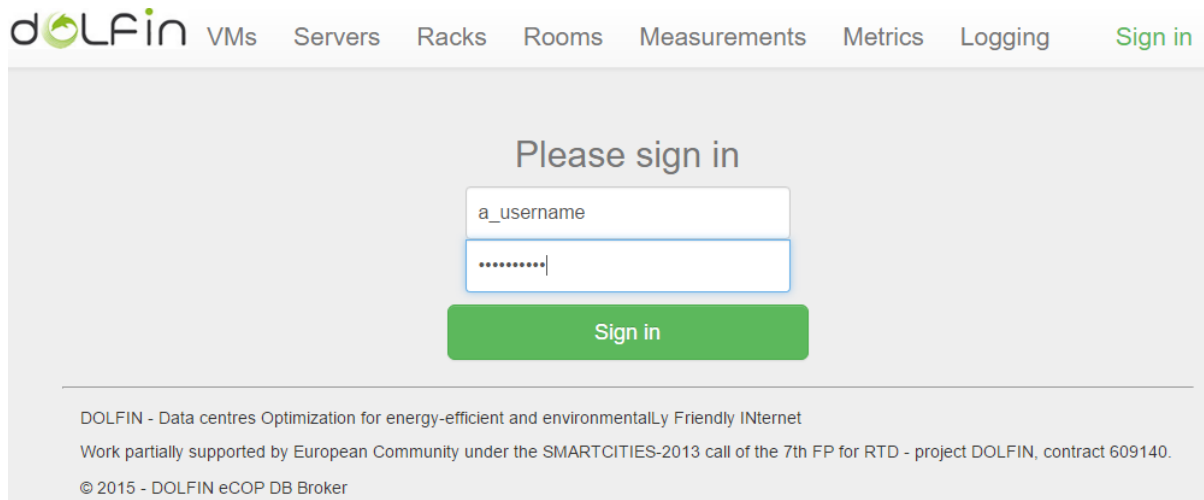
Figure 3-10: Documenting the response class and the path-parameters of an API service endpoint.

The full listing of the supported service APIs is documented in paragraph 3.3.5

3.3.4.2. Using the eCOP DB Dashboard

In order to log into the eCOP DB Dashboard, one should navigate to the page http://<BROKER_HOSTNAME_OR_IP>/dolfin/db/html/login¹³ and provide the credentials that were given when the user was inserted into the DB. Figure 3-11 depicts the landing page of the login service.

¹³ In the context of DOLFIN's 1st prototype, the BROKER_HOSTNAME_OR_IP is 83.235.169.221.



Please sign in

[Sign in](#)

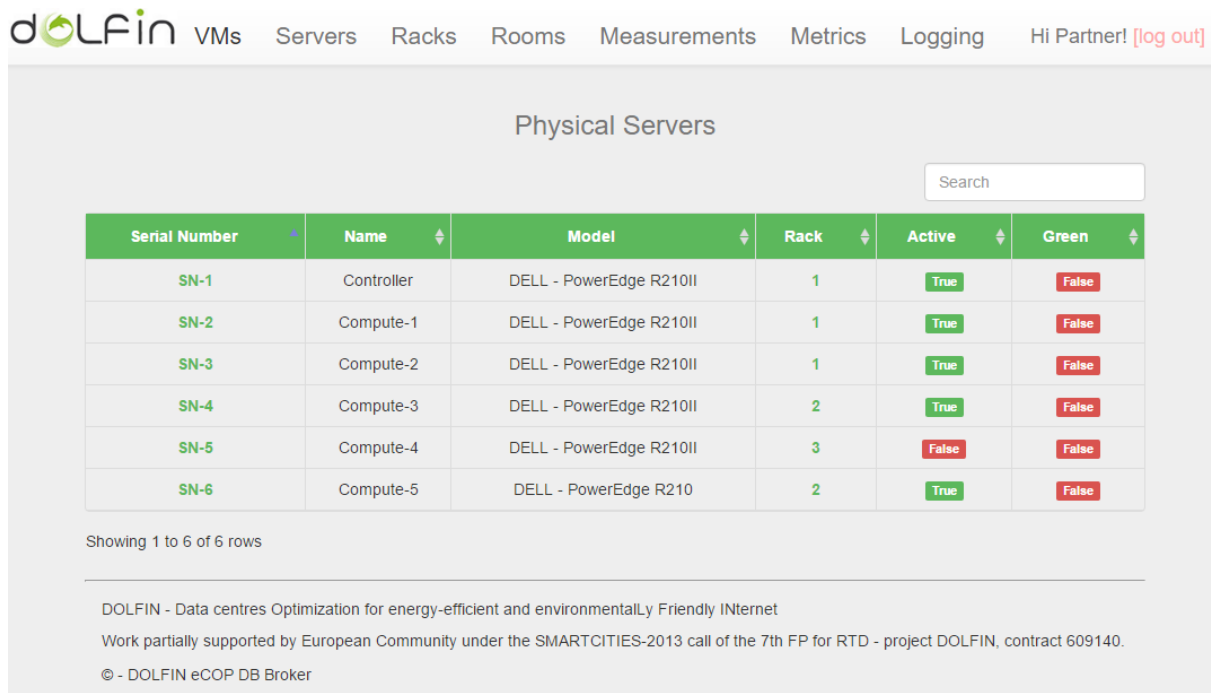
DOLFIN - Data centres Optimization for energy-efficient and environmentally Friendly INternet

Work partially supported by European Community under the SMARTCITIES-2013 call of the 7th FP for RTD - project DOLFIN, contract 609140.

© 2015 - DOLFIN eCOP DB Broker

Figure 3-11: Login page of the eCOP DB Dashboard.

Upon successful login, the user is able to see the list of VMs, servers, racks and rooms of the DC, by clicking on the relevant tabs in the top of the Dashboard and an indicative view of the servers list tab is provided in Figure 3-12.



Physical Servers

Serial Number	Name	Model	Rack	Active	Green
SN-1	Controller	DELL - PowerEdge R210II	1	True	False
SN-2	Compute-1	DELL - PowerEdge R210II	1	True	False
SN-3	Compute-2	DELL - PowerEdge R210II	1	True	False
SN-4	Compute-3	DELL - PowerEdge R210II	2	True	False
SN-5	Compute-4	DELL - PowerEdge R210II	3	False	False
SN-6	Compute-5	DELL - PowerEdge R210	2	True	False

Showing 1 to 6 of 6 rows

DOLFIN - Data centres Optimization for energy-efficient and environmentally Friendly INternet

Work partially supported by European Community under the SMARTCITIES-2013 call of the 7th FP for RTD - project DOLFIN, contract 609140.

© - DOLFIN eCOP DB Broker

Figure 3-12: List of the servers of the hypothetical DC.

When clicking on the serial number of a server, the user is redirected to the details page of the particular server, where its detailed characteristics are provided to the user, including its hardware specifications, the VMs currently hosted in this particular server, any actions that were taken by the eCOP Actuator during the previous time as well as the latest measurements related to this server (the last 1000 measurements are provided). An example of a server details page is depicted in Figure 3-13 (the related actions and the measurements have been omitted from the snapshot for reasons of space). Note that similar pages are available also for the VMs, racks and rooms of the DC.

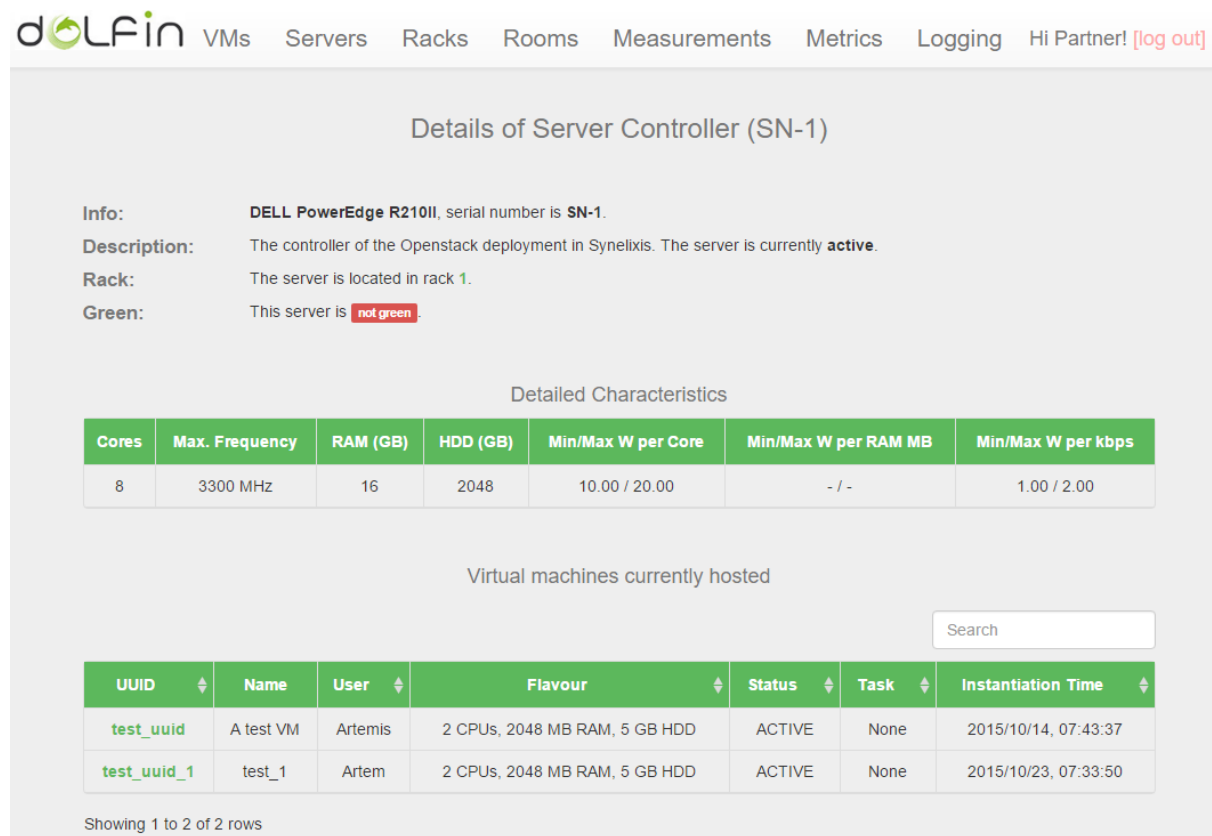


Figure 3-13: Details of server with serial-number SN-1.

Apart from reviewing the DC equipment and infrastructure, the Dashboard also provides real-time information on the monitored elements of the DC, in the measurements tab, depicted in Figure 3-14. Note that through the dropdown menus available, one can determine an arbitrary measurement type, resource, resource id as well as a timeframe to look for values into (e.g. a sliding window of 1 hour in the past).

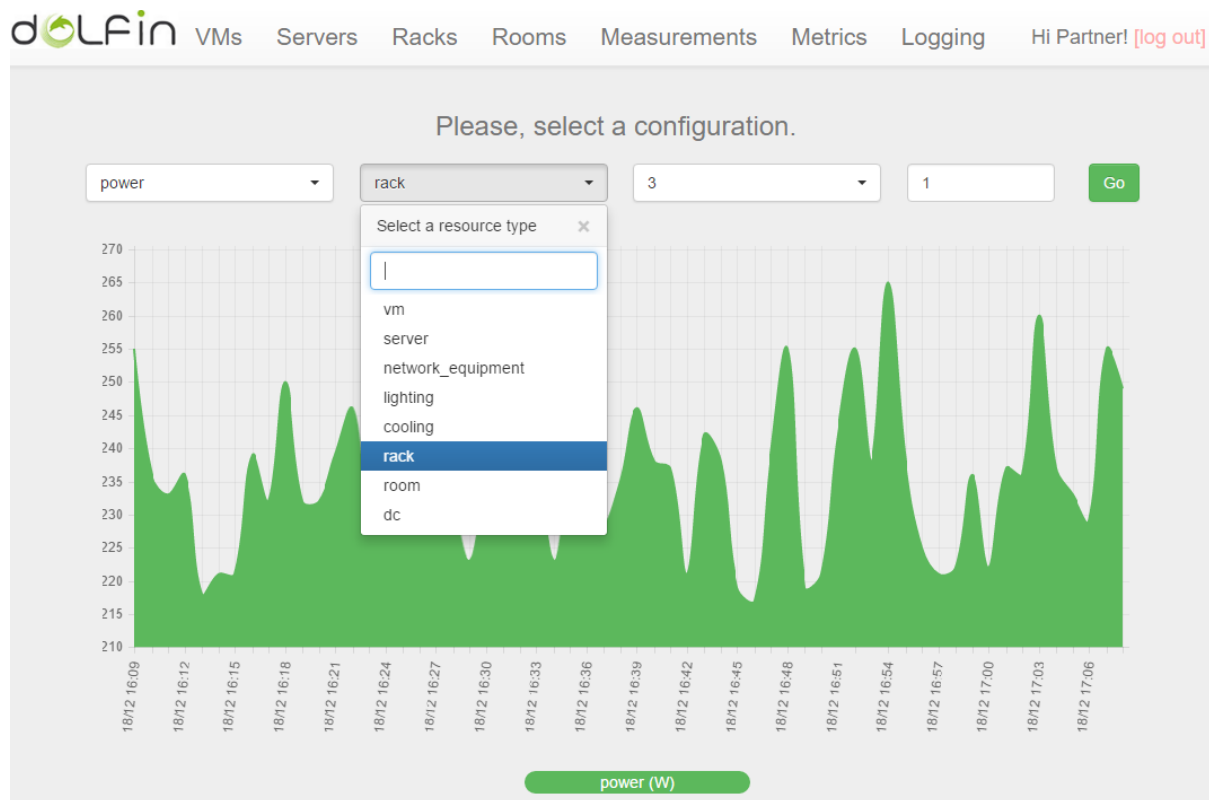
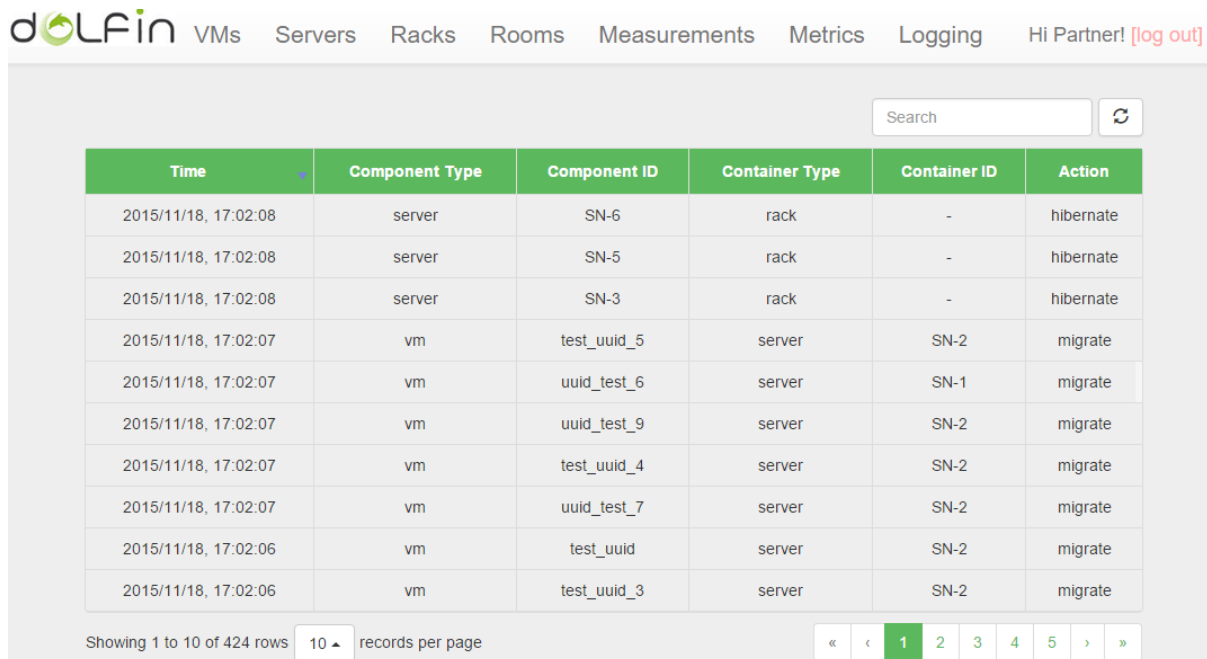


Figure 3-14: The measurements tab of the eCOP DB Dashboard.

Also, the user is able to retrieve a list of actions that were performed by the eCOP Policy Actuator by selecting the Logs tab, depicted in Figure 3-15.



The screenshot shows the DOLFIN web interface with the 'Logs' tab selected. It displays a table of actions performed by the eCOP Policy Actuator. The table has columns for Time, Component Type, Component ID, Container Type, Container ID, and Action. Below the table is a pagination bar showing 'Showing 1 to 10 of 424 rows' and a '10 records per page' dropdown. The table contains 10 rows of data, including actions like 'hibernate' and 'migrate'.

Time	Component Type	Component ID	Container Type	Container ID	Action
2015/11/18, 17:02:08	server	SN-6	rack	-	hibernate
2015/11/18, 17:02:08	server	SN-5	rack	-	hibernate
2015/11/18, 17:02:08	server	SN-3	rack	-	hibernate
2015/11/18, 17:02:07	vm	test_uuid_5	server	SN-2	migrate
2015/11/18, 17:02:07	vm	uuid_test_6	server	SN-1	migrate
2015/11/18, 17:02:07	vm	uuid_test_9	server	SN-2	migrate
2015/11/18, 17:02:07	vm	test_uuid_4	server	SN-2	migrate
2015/11/18, 17:02:07	vm	uuid_test_7	server	SN-2	migrate
2015/11/18, 17:02:06	vm	test_uuid	server	SN-2	migrate
2015/11/18, 17:02:06	vm	test_uuid_3	server	SN-2	migrate

Figure 3-15: List of actions performed by the eCOP Policy Actuator.

An indicative listing of the endpoint URLs related to the capabilities of the dashboard is tabulated in Table 3-3.

API Service Endpoint URL	Description
/dolphin/db/html/logging/	Get a view of the latest actions performed by the eCOP Policy Actuator.
/dolphin/db/html/login/	Log into the dashboard
/dolphin/db/html/measurements/	Get a real-time graphical representation of the monitored values of a DC resource
/dolphin/db/html/metrics/	Get a real-time graphical representation of the supported KPIs of the DC
/dolphin/db/html/racks/	Get the list of racks of the DC
/dolphin/db/html/racks/<id>/	Get details regarding the rack with id <id>
/dolphin/db/html/rooms/	Get the list of rooms of the DC
/dolphin/db/html/rooms/<id>/	Get details regarding room with id <id>
/dolphin/db/html/servers/	Get the list of servers of the DC
/dolphin/db/html/servers/<serial_number>/	Get details regarding server with serial number <serial_number>
/dolphin/db/html/vms/	Get the list of VMs of the DC
/dolphin/db/html/vms/<uuid>/	Get details regarding VM with uuid <uuid>

Table 3-3: The main endpoints URLs related to the dashboard utilities.

3.3.4.3. Using the eCOP DB from the command line

The easiest way to use and test the eCOP DB is from the command line, using the well-known UNIX command `curl`, to activate one of the API service endpoints detailed in paragraph 3.3.5. Indicatively, if one wanted to get more information about the server racks that are available in a DOLFIN-enabled DC, the following commands should be issued:

```
# Install curl command
$ sudo apt install curl
$ curl -X POST -H "Content-Type: application/json" -d '{"username":
"test","password": "test"}'
http://<BROKER_HOSTNAME_OR_IP>/dolphin/db/api/tokens/
# Here comes the response with the token
{"token":"581e76b175a6b48c78bb39ee598284e7183affdb","created_at":"2015-12-
18T14:05:36.393866+00:00","expires_at":"2015-12-19T14:05:36.393866+00:00"}
# Request the racks of the DC, using the token already acquired
$ curl -H "Authorization: Token 581e76b175a6b48c78bb39ee598284e7183affdb"
http://<BROKER_HOSTNAME_OR_IP>/dolphin/db/api/racks/
# No racks have been added yet in a vanilla installation
[]
```

3.3.5. Service endpoints and data model

The eCOP DB Broker has been configured to persist and offer on demand data categorized into and related to the following groups of services:

Next, these groups are presented in detail.

1. cooling
2. flavours
3. lighting
4. logs
5. measurement_types
6. measurements
7. metrics
8. network_element_types
9. network_elements
10. racks
11. rooms
12. servers
13. tokens
14. vms

3.3.5.1. Cooling [/api/cooling/]

The cooling service provides information regarding the cooling elements of the DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/cooling/	Get a list of the cooling elements of the DC
POST	/api/cooling/	Persist a new cooling element
GET	/api/cooling/<pk>/	Get details for the cooling element with id <i>pk</i> .
PUT	/api/cooling/<pk>/	Update a particular cooling element with id <i>pk</i> .
PATCH	/api/cooling/<pk>/	Update a particular cooling element with id <i>pk</i> .
DELETE	/api/cooling/<pk>/	Delete a particular cooling element with id <i>pk</i> .
GET	/api/cooling/by-room/<id>/	Get the cooling elements located in the room with id <i>id</i> .

Table 3-4: List of API service endpoints related to the cooling elements of a DOLFIN DC.

The data model of a cooling element retrieved by the eCOP DB is as follows:

Attribute	Type	Description
id	String	The ID of the cooling element
description	String	The description of the cooling element

model	String	The model of the cooling element
manufacturer	String	The manufacturer of the cooling element
min_watt_idle	Decimal	The minimum consumption of the cooling element, when idle
max_watt_nonidle	Decimal	The maximum consumption of the cooling element
air_heat_capacity	Decimal	The air heat capacity of the cooling element
btu_conv_k	Decimal	The BTU of the cooling element
room	String	The room where the cooling element is located
green	Boolean	Indicates whether this cooling element is green (e.g. powered by green energy sources) or free (outside air cooling)

Table 3-5: Data model for the 'cooling' elements.

3.3.5.1. Flavours (/api/flavours/)

The flavours service provides information regarding the supported VM flavours (virtual hardware configuration). The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/flavours/	Get the list of supported flavour
POST	/api/flavours/	Persist a new flavour
GET	/api/flavours/<pk>/	Get details for the flavour with id <i>pk</i> .
PUT	/api/flavours/<pk>/	Update a particular flavour with id <i>pk</i> .
PATCH	/api/flavours/<pk>/	Update a particular flavour with id <i>pk</i> .
DELETE	/api/flavours/<pk>/	Delete a particular flavour with id <i>pk</i> .

Table 3-6: List of API service endpoints related to the Flavours

The data model of a flavour retrieved by the eCOP DB is as follows:

Attribute	Type	Description
id	Integer	The ID of the flavour (auto-generated)
name	String	The name of the flavour
description	String	The description of the flavour
vcpus	Integer	The number of virtual CPUs of the flavour
vram	Integer	The size of the virtual RAM of the flavour (in MB)
vhdd	Integer	The size of the virtual HDD of the flavour (in GB)

Table 3-7: Data model for the 'cooling' elements.

3.3.5.2. Lighting [/api/lighting/]

The lighting service provides information regarding the supported lighting elements of the DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/lighting/	Get the list of supported lighting elements
POST	/api/lighting/	Persist a new lighting element
GET	/api/lighting/<pk>/	Get details for the lighting element with id <i>pk</i> .
PUT	/api/lighting/<pk>/	Update a particular lighting element with id <i>pk</i> .
PATCH	/api/lighting/<pk>/	Update a particular lighting element with id <i>pk</i> .
DELETE	/api/lighting/<pk>/	Delete a particular lighting element with id <i>pk</i> .

Table 3-8: List of API service endpoints related to the Lightning

The data model of a lighting element retrieved by the eCOP DB is as follows:

Attribute	Type	Description
id	Integer	The ID of the lighting element (auto-generated)
description	String	The description of the lighting element
watt_non_idle	Decimal	The consumption of the lighting element when on
room	String	The ID of the room where the lighting equipment is located

Table 3-9: Data model for the 'lightning' elements.

3.3.5.3. Logs [/api/logs/]

The logs service provides information regarding the actions performed by the various DC elements and, particularly, the actions performed by the eCOP Policy Actuator. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/logs/	Get the list of logs
POST	/api/logs/	Persist a new log
GET	/api/logs/<pk>/	Get details for the log with id <i>pk</i> .
PUT	/api/logs/<pk>/	Update a particular log with id <i>pk</i> .
PATCH	/api/logs/<pk>/	Update a particular log with id <i>pk</i> .
DELETE	/api/logs/<pk>/	Delete a particular log with id <i>pk</i> .
GET	/api/logs/by-component-id/<component_id>/	Get the logs that refer to a particular component (e.g. the uuid of a VM, the serial_number of a server, the id of a lighting element etc.) the id of the component being <component_id>
GET	/api/logs/by-component-	Get the logs that refer to a particular component

	type/<component_type>/	type (e.g. VM, server, rack etc.), the type being <component_type>
GET	/api/logs/by-user-id/<user_id>/	Get the logs that refer to a particular user with id <user_id>

The data model of a particular log retrieved by the eCOP DB is as follows:

Attribute	Type	Description
id	Integer	The ID of the log (auto-generated)
time	String	The time of when the action was logged in ISO8601 format
component_type	String	The type of the component referred to by the action (e.g. VM, server, rack, room etc.)
component_id	String	The ID of the component referred to by the action (e.g. the uuid of a VM, the serial_number of a server etc.)
container_type	String	The type of the entity that contains the component
container_id	String	The id of the entity that contains the component
action	String	The description of the action performed.

3.3.5.4. Measurement types [/api/measurement_types/]

The measurement_types service provides information regarding the supported measurement types for the monitoring of the elements of the DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/measurement_types/	Get the list of supported measurement_types
POST	/api/measurement_types/	Persist a new measurement_type
GET	/api/measurement_types/<pk>/	Get details for the measurement_type with id pk.
PUT	/api/measurement_types/<pk>/	Update a particular measurement_type with id pk.
PATCH	/api/measurement_types/<pk>/	Update a particular measurement_type with id pk.
DELETE	/api/measurement_types/<pk>/	Delete a particular measurement_type with id pk.

The data model of a measurement type retrieved by the eCOP DB is as follows:

Attribute	Type	Description
name	String	The name of the measurement type
unit	String	The unit of the measurement type
description	String	The description of the measurement type

3.3.5.5. Measurements [/api/measurements/]

The measurement service provides information regarding the monitoring of the elements of the DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/measurements/	Get the list of supported measurement
POST	/api/measurements/	Persist a new measurement
GET	/api/measurements/<pk>/	Get details for the measurement with id pk.
PUT	/api/measurements/<pk>/	Update a particular measurement with id pk.
PATCH	/api/measurements/<pk>/	Update a particular measurement with id pk.
DELETE	/api/measurements/<pk>/	Delete a particular measurement with id pk.
GET	/api/measurements/by-type/<type>/	Get a list of measurements based on their type
GET	/api/measurements/by-type/<type>/<start>/<end>/	Get a list of measurements based on the type of the measurements and a date range
GET	/api/measurements/by-type-resource/<type>/<resource>/	Get a list of measurements based on the type of the measurements and a resource type
GET	/api/measurements/by-type-resource/<type>/<resource>/<id>/	Get a list of measurements based on the type of the measurements for a specific resource
GET	/api/measurements/by-type-resource/<type>/<resource>/<id>/<start>/	Get a list of measurements for a specific resource and a time range starting from now and starting <start> hours before
GET	/api/measurements/by-type-resource/<type>/<resource>/<id>/<start>/<end>/	Get a generic measurement based on the type of the measurement for a specific resource and for a time range
GET	/api/measurements/by-resource/<resource>/	Get a generic measurement based on the resource type
GET	/api/measurements/by-resource/<resource>/<id>/	Get a generic measurement for a specific resource
GET	/api/measurements/by-resource/<resource>/<id>/<start>/<end>/	Get a generic measurement for a specific resource and a time range
GET	/api/measurements/aggregate/power/<type>/<time>/	Get the instantaneous power consumption of a (part of a) DC at a specific time
GET	/api/measurements/aggregate/energy/<type>/<start>/<end>/	Get the energy consumption of a (part of a) DC during a specific time range

The data model of a measurement retrieved by the eCOP DB is as follows:

Attribute	Type	Description
-----------	------	-------------

id	Integer	The id of the measurement (auto-generated)
value	String	The value of the measurement
time	String	The time of the measurement in ISO8601 format
type	String	The type of the measurement (must be a valid measurement_type name)
resource	String	The resource type (can be a vm, server, rack etc.)
resource_id	String	The id of the resource (can be the uuid of a vm, the serial_number of a server, the id of a lighting element etc.)

3.3.5.6. Metrics [/api/metrics/]

The metrics service provides information regarding the KPIs of the DC operation. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/metrics/	Get the list of supported metrics
POST	/api/metrics/	Persist a new metric
GET	/api/metrics/<pk>/	Get details for the metric with id pk.
PUT	/api/metrics/<pk>/	Update a particular metric with id pk.
PATCH	/api/metrics/<pk>/	Update a particular metric with id pk.
DELETE	/api/metrics/<pk>/	Delete a particular metric with id pk.

The data model of a metric retrieved by the eCOP DB is as follows:

Attribute	Type	Description
name	String	The name of the metric
description	String	The description of the metric

Note that the 'metric' notation can be used as a valid 'resource' of a measurement object.

3.3.5.7. Network element types [/api/network_element_types/]

The network_element_types service provides information regarding the types of the networking elements of a DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/network_element_types/	Get the list of supported network_element_types
POST	/api/network_element_types/	Persist a new network element type
GET	/api/network_element_types/<pk>/	Get details for the network element type

		with id <i>pk</i> .
PUT	/api/network_element_types/<pk>/	Update a particular network element type with id <i>pk</i> .
PATCH	/api/network_element_types/<pk>/	Update a particular network element type with id <i>pk</i> .
DELETE	/api/network_element_types/<pk>/	Delete a particular network element type with id <i>pk</i> .

The data model of a `network_element_type` object retrieved by the eCOP DB is as follows:

Attribute	Type	Description
type	String	The name of the network element type
description	String	The description of the KPI

3.3.5.8. *Network elements [/api/network_elements]*

The `network_elements` service provides information regarding the networking elements of a DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/network_elements/	Get the list of supported network elements
POST	/api/network_elements/	Persist a new network element
GET	/api/network_elements/<pk>/	Get details for the network element with id <i>pk</i> .
PUT	/api/network_elements/<pk>/	Update a particular network element with id <i>pk</i> .
PATCH	/api/network_elements/<pk>/	Update a particular network element with id <i>pk</i> .
DELETE	/api/network_elements/<pk>/	Delete a particular network element with id <i>pk</i> .
GET	/api/network_elements/by-type/<type>	Get all the network elements of a specific type

The data model of a `network_element` object retrieved by the eCOP DB is as follows:

Attribute	Type	Description
serial_number	String	The serial number of the networking element
name	String	The name of the networking element
description	String	The description of the networking element
type	String	The type of the networking element (must be the name of a <code>network_element_type</code>)

model	String	The model of the networking element
manufacturer	String	The manufacturer of the networking element
rack	String	The rack into which the networking element is located
active	Boolean	Indicates whether the networking element is active
min_watt_per_gbps	Decimal	The minimum energy consumption per Gbps of the networking element (when idle)
max_watt_per_gbps	Decimal	The maximum energy consumption per Gbps of the networking element

3.3.5.9. Racks [/api/racks/]

The racks service provides information regarding the racks of the DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/racks/	Get the list of supported racks
POST	/api/racks/	Persist a new rack
GET	/api/racks/<pk>/	Get details for the rack with id pk.
PUT	/api/racks/<pk>/	Update a particular rack with id pk.
PATCH	/api/racks/<pk>/	Update a particular rack with id pk.
DELETE	/api/racks/<pk>/	Delete a particular rack with id pk.

The data model of a rack object retrieved by the eCOP DB is as follows:

Attribute	Type	Description
id	Integer	The id of the rack (auto-generated)
name	String	The name of the rack
description	String	The description of the rack
model	String	The model of the rack
manufacturer	String	The manufacturer of the rack
room	String	The room into which the rack is located

3.3.5.10. Servers [/api/servers/]

The servers service provides information regarding the servers of the DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/servers/	Get the list of supported servers

POST	/api/servers/	Persist a new server
GET	/api/servers/<pk>/	Get details for the server with serial number pk.
PUT	/api/servers/<pk>/	Update a particular server with serial number pk.
PATCH	/api/servers/<pk>/	Update a particular server with serial number pk.
DELETE	/api/servers/<pk>/	Delete a particular server with serial number pk.
GET	/api/servers/by-cooling-system/<cooling_system>/	Get the active servers for a specific cooling system
GET	/api/servers/by-energy-efficiency/sorted/	Get the servers of the DC, sorted by their energy efficiency

The data model of a server object retrieved by the eCOP DB is as follows:

Attribute	Type	Description
serial_number	String	The serial number of the server
name	String	The name of the server
description	String	The description of the server
rack	String	The rack into which the server is located
model	String	The model of the server
manufacturer	String	The manufacturer of the server
active	Boolean	Indicates whether the server is active
cpus	Integer	The number of CPU cores of the server
ram	Integer	The amount of RAM of the server (in MB)
hdd	Integer	The amount of HDD of the server (in GB)
cpu_frequency	Integer	The CPU frequency of the server CPU cores
min_watt_per_cpu_core	Decimal	The minimum power demand per CPU core of the server
max_watt_per_cpu_core	Decimal	The maximum power demand per CPU core of the server
min_watt_per_kbps	Decimal	The minimum power demand per kbps of the server
max_watt_per_kbps	Decimal	The maximum power demand per kbps of the server
cpu_energy_mult	Decimal	The multiplier of the energy consumption due to CPU utilization of the server (used in DOLFIN energy models)
ram_energy_mult	Decimal	The multiplier of the energy consumption due to RAM utilization of the server (used in DOLFIN energy models)
net_energy_mult	Decimal	The multiplier of the energy consumption due to network utilization of the server (used in DOLFIN energy models)
green	Boolean	Indicates whether the server is green (e.g. is of low energy consumption, or is powered by green energy sources)

3.3.5.11. Tokens [/api/tokens/]

The tokens endpoint is used for users to acquire authorization tokens to use for accessing the contents of the eCOP DB through the Broker. The following API service endpoint has been configured for this endpoint:

HTTP Method	API Service Endpoint URL	Description
POST	/api/tokens/	Get a new authorization token

The data model of entity to be posted in order to acquire a new token is structured as follows:

Attribute	Type	Description
username	String	The username of the user
password	String	The password of the user

The data model of a token object retrieved by the eCOP DB is as follows:

Attribute	Type	Description
token	String	The authorization token
created_at	String	The date when the token was generated in ISO8601 format
expires_at	String	The date when the token ceases to be valid in ISO8601 format

The token should be used in every HTTP request performed against the eCOP DB Broker API service endpoints as a header in the form:

```
Authorization: Token <token>
```

where <token> is the acquired token.

3.3.5.12. VMs [/api/vms/]

The VMs service provides information regarding the VMs of the DC. The following endpoints have been defined for this group:

HTTP Method	API Service Endpoint URL	Description
GET	/api/vms/	Get the list of supported VMs

POST	/api/vms/	Persist a new VM
GET	/api/vms/<pk>/	Get details for the VM with uuid pk.
PUT	/api/vms/<pk>/	Update a particular VM with uuid pk.
PATCH	/api/vms/<pk>/	Update a particular VM with uuid pk.
DELETE	/api/vms/<pk>/	Delete a particular VM with uuid pk.
GET	/api/vms/by-user-id/<user_id>/	Get the list of VMs that belong to a specific user
GET	/api/vms/by-server/<serial_number>/	Get the list of VMs that are hosted in a specific server

The data model of a VM object retrieved by the eCOP DB is as follows:

Attribute	Type	Description
name	String	The name of the VM
uuid	String	The uuid of the VM
user	String	The user of the VM
flavour	String	The flavour of the VM (must be a valid flavour id)
server	String	The server hosting the VM (must be a valid serial number of an existing server)
status	String	The status of the VM
task	String	The current task of the VM
instantiation_time	String	The instantiation time of the VM in ISO8601 format

Figure 3-16 presents an overview of the eCOP DB Broker data models hierarchy.

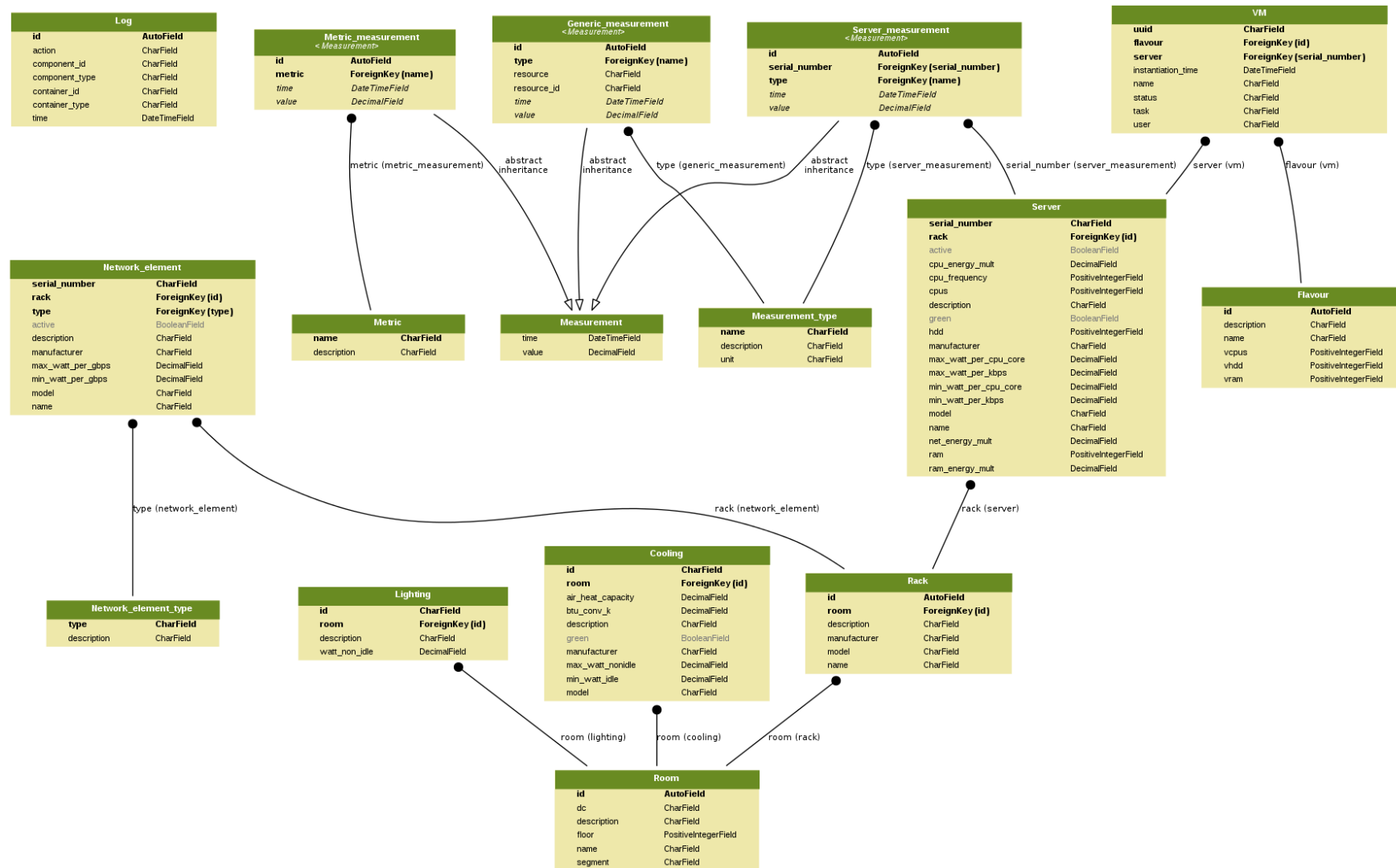


Figure 3-16: eCOP DB Broker data model hierarchy

3.4. DCO Brokers

The DCO Brokers have been developed in conjunction with DOLFIN ICT Performance and Energy Supervisor. The current implementation of the DCO Hypervisor Broker uses Apache jclouds¹⁴ to issue commands to a running OpenStack installation. The use of jclouds however significantly abstracts the target DCO Hypervisor stack and thus the DCO Brokers can be used unaltered for a large collection of private and public cloud infrastructures through their respective APIs (e.g. AWS, CloudStack, ElasticStack, DigitalOcean, Docker etc.).

3.4.1. Installation and configuration of the DCO Brokers

As the DCO Brokers are written in Java, there are two ways to deploy them:

- Deploy the pre-built binaries
- Build from source

If provided with a binary distribution, load the contents to a folder (by default, `brokers`) and proceed to the using and testing section.

To build the DCO Brokers from source on an Ubuntu machine, use the following procedure:

```
# Install maven and the JDK
$ sudo aptitude install maven openjdk-8-jdk
# Untar the brokers distribution
$ tar -zxvf brokers-src.tar.gz
# Build the brokers package
$ cd brokers/openstack
$ mvn package
```

At this point, Maven builds the source code and packages the binaries along with the required libraries in the `target` directory.

Configuration is provided for the brokers at runtime, using standard Java XML properties files. Example files are provided in the `brokers/openstack/src/main/config` directory. At least the following parameters in `broker.xml` should be overridden with the DC specific values:

Parameter	Description	Example Value
tenantname	The administrative tenant name	adminTenant
username	An admin/privileged user	adminUser

¹⁴ Apache jclouds: <https://jclouds.apache.org/>

password	The password for access	adminSECRETpassword
credential	Other credential (e.g. an admin token)	adminToken.txt
identity.endpoint	The endpoint of the OpenStack Identity service (keystone)	http://endpoint:35357/v3.0/
compute.endpoint	The endpoint of the Openstack Compute service (nova)	http://endpoint:5000/v2.0/
region	The region to manage	regionOne
groupname	The instance name template for instances spawned through the broker	dolphin-instances

Optionally, parameters of the `logback.xml` file can be overridden to provide more debugging details.

3.4.2. Using and testing the DCO Brokers

To test the broker independently of the Policy Actuator, a small command line utility can be used to issue low level commands to a running OpenStack installation. An example output to a specific set of test commands is provided below:

```
## Test admin permissions

user@user-desktop:~/brokers/openstack$ java -
Dbroker.properties=src/main/config/broker.xml -Dpassword=<secretpw> -jar
target/openstack-adaptor-0.0.1-jar-with-dependencies.jar authenticate

2015-12-21 12:12:38,018 INFO [g.g.d.o.i.IdentityActions] - Endpoint:
http://endpoint:35357/v3.0/

2015-12-21 12:12:38,025 INFO [g.g.d.o.i.IdentityActions] - Endpoint:
http://endpoint:35357/v3.0/

2015-12-21 12:12:39,698 INFO [g.g.d.o.i.IdentityActions] - Authenticate On Call

2015-12-21 12:12:39,739 INFO [g.g.d.o.i.IdentityActions] - Authenticated

## Create a test server

user@user-desktop:~/brokers/openstack$ java ... createServer

2015-12-21 12:12:56,915 INFO [g.g.d.o.i.IdentityActions] - Endpoint:
http://endpoint:35357/v3.0/

2015-12-21 12:12:56,922 INFO [g.g.d.o.i.IdentityActions] - Endpoint:
http://endpoint:35357/v3.0/

2015-12-21 12:12:58,578 INFO [g.g.d.o.c.ComputeActions] - Create Server%n

2015-12-21 12:13:20,619 INFO [g.g.d.o.c.ComputeActions] - {id=regionOne/4418aaa0-
ac08-43ef-b593-d281e9ea0ec5, providerId=4418aaa0-ac08-43ef-b593-d281e9ea0ec5,
uri=http://termi7:8774/v2/c4841ea2951d41bea90d5b6e1f8e6732/servers/4418aaa0-ac08-
43ef-b593-d281e9ea0ec5, name=my-instances-ec6,
uri=http://termi7:8774/v2/c4841ea2951d41bea90d5b6e1f8e6732/servers/4418aaa0-ac08-
43ef-b593-d281e9ea0ec5, location={scope=HOST,
id=828e766f4fa830b8b3299f6e5c46a252eed9d5eb656924a76dedc3fd,
description=828e766f4fa830b8b3299f6e5c46a252eed9d5eb656924a76dedc3fd,
```

```
parent=regionOne}, group=my-instances, imageId=regionOne/a3bcd3b-b04c-447a-8d68-12e0d24cdb8f, os={family=ubuntu, name=Ubuntu 14.04.2 x64, version=14.04, description=Ubuntu 14.04.2 x64, is64Bit=true}, status=RUNNING, loginPort=22, hostname=my-instances-ec6, privateAddresses=[192.168.5.187], hardware={id=regionOne/0a6c7fe2-a932-4753-985a-b916cdf99811, providerId=0a6c7fe2-a932-4753-985a-b916cdf99811, name=special.4, location={scope=REGION, id=regionOne, description=regionOne, parent=openstack-nova}, processors=[{cores=4.0, speed=1.0}], ram=8192, volumes=[{type=LOCAL, size=20.0, bootDevice=true, durable=true}], supportsImage=ALWAYS_TRUE}, loginUser=ubuntu, userMetadata={jclouds-group=my-instances}}
```

```
2015-12-21 12:13:20,619 INFO [g.g.d.o.c.ComputeActions] - Login: ssh
ubuntu@192.168.5.187
```

```
2015-12-21 12:13:20,619 INFO [g.g.d.o.c.ComputeActions] - Password: Jg6RL25hfLbf
```

```
## List all user virtual servers.
```

```
user@user-desktop:~/brokers/openstack$ java ... listServers
```

```
2015-12-21 12:14:39,636 INFO [g.g.d.o.i.IdentityActions] - Endpoint:
http://endpoint:35357/v3.0/
```

```
2015-12-21 12:14:39,643 INFO [g.g.d.o.i.IdentityActions] - Endpoint:
http://endpoint:35357/v3.0/
```

```
Total Number of Nodes = 5
```

```
{id=regionOne/4418aaa0-ac08-43ef-b593-d281e9ea0ec5, providerId=4418aaa0-ac08-43ef-b593-d281e9ea0ec5, uri=http://termi7:8774/v2/c4841ea2951d41bea90d5b6elf8e6732/servers/4418aaa0-ac08-43ef-b593-d281e9ea0ec5, name=my-instances-ec6, uri=http://termi7:8774/v2/c4841ea2951d41bea90d5b6elf8e6732/servers/4418aaa0-ac08-43ef-b593-d281e9ea0ec5, location={scope=HOST, id=828e766f4fa830b8b3299f6e5c46a252eed9d5eb656924a76dedc3fd, description=828e766f4fa830b8b3299f6e5c46a252eed9d5eb656924a76dedc3fd, parent=regionOne}, group=my-instances, imageId=regionOne/a3bcd3b-b04c-447a-8d68-12e0d24cdb8f, os={family=ubuntu, name=Ubuntu 14.04.2 x64, version=14.04, description=Ubuntu 14.04.2 x64, is64Bit=true}, status=RUNNING, loginPort=22, hostname=my-instances-ec6, privateAddresses=[192.168.5.187], hardware={id=regionOne/0a6c7fe2-a932-4753-985a-b916cdf99811, providerId=0a6c7fe2-a932-4753-985a-b916cdf99811, name=special.4, location={scope=REGION, id=regionOne, description=regionOne, parent=openstack-nova}, processors=[{cores=4.0, speed=1.0}], ram=8192, volumes=[{type=LOCAL, size=20.0, bootDevice=true, durable=true}], supportsImage=ALWAYS_TRUE}, userMetadata={jclouds-group=my-instances}}
```

```
...
```

```
## Pause/unpause a running virtual server in memory
```

```
user@user-desktop:~/brokers/openstack$ java ... pauseServer 4418aaa0-ac08-43ef-b593-d281e9ea0ec5
```

```
2015-12-21 12:15:03,643 INFO [g.g.d.o.i.IdentityActions] - Endpoint:
http://endpoint:35357/v3.0/
```

```
2015-12-21 12:15:03,650 INFO [g.g.d.o.i.IdentityActions] - Endpoint:
http://endpoint:35357/v3.0/
```

```
2015-12-21 12:15:05,253 INFO [g.g.d.o.c.ComputeActions] - Pause Server
```

```
2015-12-21 12:15:07,032 INFO [g.g.d.o.c.ComputeActions] - Server 4418aaa0-ac08-
```

43ef-b593-d281e9ea0ec5 paused.

user@user-desktop:~/brokers/openstack\$ java ... unpauseServer 4418aaa0-ac08-43ef-b593-d281e9ea0ec5

2015-12-21 12:15:21,428 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/

2015-12-21 12:15:21,435 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/

2015-12-21 12:15:23,153 INFO [g.g.d.o.c.ComputeActions] - Unpause Server

2015-12-21 12:15:24,854 INFO [g.g.d.o.c.ComputeActions] - Server 4418aaa0-ac08-43ef-b593-d281e9ea0ec5 unpaused.

Migrate the virtual server to a different HW node

user@user-desktop:~/brokers/openstack\$ java ... migrateServer 4418aaa0-ac08-43ef-b593-d281e9ea0ec5

2015-12-21 12:15:44,500 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/

2015-12-21 12:15:44,506 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/

2015-12-21 12:15:46,209 INFO [g.g.d.o.c.ComputeActions] - Migrate Server

2015-12-21 12:17:09,407 INFO [g.g.d.o.c.ComputeActions] - Server 4418aaa0-ac08-43ef-b593-d281e9ea0ec5 migrated.

Stop the virtual server, verify it is stopped and start it again (in essence, perform a hard VM reboot)

user@user-desktop:~/brokers/openstack\$ java ... stopServer 4418aaa0-ac08-43ef-b593-d281e9ea0ec5

2015-12-21 12:17:47,276 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/

2015-12-21 12:17:47,283 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/

2015-12-21 12:17:48,882 INFO [g.g.d.o.c.ComputeActions] - Stop Server

2015-12-21 12:17:52,000 INFO [g.g.d.o.c.ComputeActions] - Server 4418aaa0-ac08-43ef-b593-d281e9ea0ec5 stopped.

user@user-desktop:~/brokers/openstack\$ java ... listServers

2015-12-21 12:17:56,197 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/

2015-12-21 12:17:56,204 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/

Total Number of Nodes = 5

{id=regionOne/4418aaa0-ac08-43ef-b593-d281e9ea0ec5, providerId=4418aaa0-ac08-43ef-b593-d281e9ea0ec5, uri=http://termi7:8774/v2/c4841ea2951d41bea90d5b6e1f8e6732/servers/4418aaa0-ac08-43ef-b593-d281e9ea0ec5, name=my-instances-ec6,

```
uri=http://termi7:8774/v2/c4841ea2951d41bea90d5b6elf8e6732/servers/4418aaa0-ac08-43ef-b593-d281e9ea0ec5, location={scope=HOST, id=0d9a990490658c316cce0e8c95d3f6ec9bbbe6edf4fd7c8c5708a63b, description=0d9a990490658c316cce0e8c95d3f6ec9bbbe6edf4fd7c8c5708a63b, parent=regionOne}, group=my-instances, imageId=regionOne/a3bcd3b-b04c-447a-8d68-12e0d24cdb8f, os={family=ubuntu, name=Ubuntu 14.04.2 x64, version=14.04, description=Ubuntu 14.04.2 x64, is64Bit=true}, status=SUSPENDED, loginPort=22, hostname=my-instances-ec6, privateAddresses=[192.168.5.187], hardware={id=regionOne/0a6c7fe2-a932-4753-985a-b916cdf99811, providerId=0a6c7fe2-a932-4753-985a-b916cdf99811, name=special.4, location={scope=REGION, id=regionOne, description=regionOne, parent=openstack-nova}, processors=[{cores=4.0, speed=1.0}], ram=8192, volumes=[{type=LOCAL, size=20.0, bootDevice=true, durable=true}], supportsImage=ALWAYS_TRUE}, userMetadata={jclouds-group=my-instances}}
```

...

```
user@user-desktop:~/brokers/openstack$ java ... startServer 4418aaa0-ac08-43ef-b593-d281e9ea0ec5
```

```
2015-12-21 12:18:14,764 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/
```

```
2015-12-21 12:18:14,771 INFO [g.g.d.o.i.IdentityActions] - Endpoint: http://endpoint:35357/v3.0/
```

```
2015-12-21 12:18:16,542 INFO [g.g.d.o.c.ComputeActions] - Start Server
```

```
2015-12-21 12:18:21,010 INFO [g.g.d.o.c.ComputeActions] - Server 4418aaa0-ac08-43ef-b593-d281e9ea0ec5 started.
```

3.4.3. Service endpoints and data model

Applicable to Policy Actuator.

4. Conclusions

This document has presented the ICT Performance and Energy Supervisor detailed implementation and the external interfaces to this module. This document has also described the details of each of the modules implementation providing end-points for their APIs and data models of each of the components of the ICT Performance and Energy Supervisor. The ICT Performance and Energy supervisor will be the core of the DOLFIN system, interacting with the infrastructure components and the overlaying system that holds the intelligence to calculate the most optimum set up of the system and save energy in the first instance.

The components that have been discussed in this deliverable are the:

- ICT Performance and Energy Supervisor Core (or Event Receiver Engine)
- Metrics engine and ICT Topology
- Monitoring DB
- DCO Brokers.

All the methods and end-points for those components have been explained here. Each component has been tested to demonstrate their correct functioning and operation in an isolated environment while looking ahead on the following steps of integration with the rest of the components of the DOLFIN architecture.

References

- [1] D2.1 - DOLFIN_D2.1_IRT_FF_20140207
- [2] D2.2 - DOLFIN_D2.2_UCL_30-09-2014_FF
- [3] D3.1 - DOLFIN_D3.1_NXW_FF-20150430
- [4] D3.2 - DOLFIN_D3.2_SYN_FF_20151109
- [5] D4.1 - DOLFIN_D4.1_UCL_FF-20150430
- [6] D5.1 - DOLFIN_D5.1_GRNET_FF
- [7] OpenStack - <https://www.openstack.org/>
- [8] SNMP Protocol - <https://www.ietf.org/rfc/rfc1157.txt>