Manuale d'uso Modulo SCE
4.35.1

Versione 2.2
Data: 25/06/2014

Progetto iCaro
La piattaforma cloud per l'accelerazione del business delle PMI toscane
[CUP 6408.30122011.026000074]

Informazioni sul documento

ID Deliverable	DE4.35.1
Titolo Deliverable	Manuale d'uso Modulo SCE
ID Attività	
N. Versione / Revisione	2.2
Natura: Bozza / Definitivo	Finale
Partner responsabile	DISIT
Distribuzione: Riservato / Pubblico	Pubblico
Riferimenti Autore	Daniele Cenni, Claudio Badii
Data redazione	18/04/2014
Riferimenti revisore	Paolo Nesi
Data revisione	30/06/2014
Riferimenti soggetto che approva	Paolo Nesi
Data approvazione e consegna	30/06/2014

Controllo delle revisioni

Oggetto	Numero	Data
Prima stesura e revisione	1.0	23/03/2014
Seconda stesura	2.0	18/05/2014
Terza stesura	2.1	18/06/2014
finale	2.2	30/06/2014

Nota di riservatezza

Il presente documento sarà utilizzato esclusivamente ai fini del progetto ICARO, ha carattere riservato e non potrà quindi essere divulgato se non in seguito ad esplicita autorizzazione scritta da parte dell'ATS, salvo il caso in cui di richieste di ottemperare ad obblighi di legge o a richieste di pubbliche autorità.

Indice

1 Home 5
2 Jobs 7
3 Triggers 8
4 New Job 10
4.1 Job Data 11
4.2 Trigger Data 12
5 New Job (dormant) 14
6 New Trigger 15
7 Nodes Status 17
8 Nodes Log 19
9 Log 21
10 Programmatically call the Scheduler 22
10.1 Insert a Job 22
10.2 Updating a Job 23
10.3 Other operations 23
11. Configuration Tester 26
11.1 Home 26
11.2 Create a DataCenter 26
11.2.1 Host Machine 27
11.2.1.1 Monitor Info 29
11.2.1.2 Local Network 30
11.2.1.3 Local Storage 31
11.2.1.4 Shared Storage 31
11.2.2 External Storage, Firewall, Router 32
11.2.2.1 Shared Storage 33
11.2.3 Create XML 34
11.2.3.1 Eccezioni nella generazione del file XML 36
11.3 Create a BusinessConfiguration 38
11.3.1 Choose DataCenter 38
11.3.2 Create Virtual Machine 39
11.3.3 IcaroApplication 43
11.3.3.1 Icaro Service 44

11.3.3.2 SLAgreement, SLObjective, SLAction e SLMetric 46
11.3.3.3 Creator 47
11.3.4 Create XML 48
11.3.4.1 Eccezioni nella generazione del file XML 49
11.3.4.2 Eccezioni nell'inserimento del file XML nella KB 49
11.4 Create ServiceMetrics 50
11.4.1 Create XML 51
11.5 Simulate DataCenter 51
11.5.1 Real Time Simulation 52
11.5.2 Faster Simulation 53
11.6 Analyze Metrics 54

Legenda Acronimi e sigle

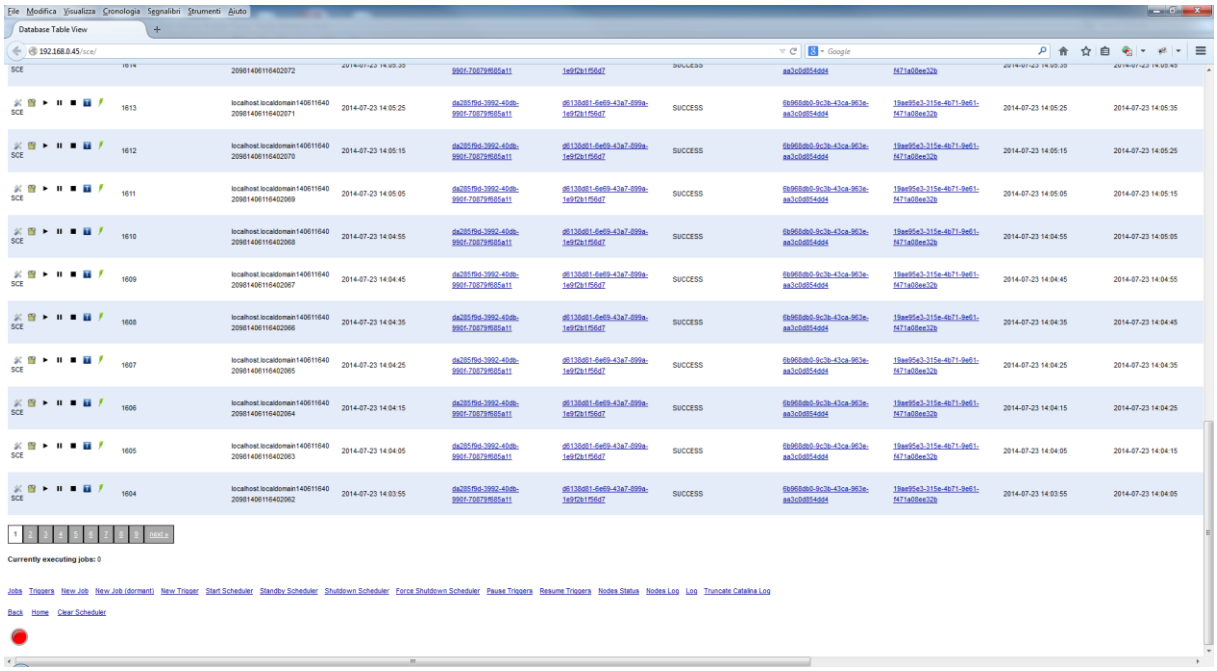
The Smart Cloud Engine Scheduler presents a list of menu items at the bottom of the main page.

- **Jobs**, views the list of currently registered jobs in the scheduler;
- **Triggers,** views the list of currently registered triggers in the scheduler;
- **New Job**, form for creating a new job with an associated trigger, and registering them into the scheduler;
- **New Job (dormant)**, form for creating a new job without an associated trigger, and registering it into the scheduler;
- **New Trigger**, form for creating a new trigger, and registering them into the scheduler;
- **Start Scheduler**, starts the Scheduler's threads that fire triggers. When a scheduler is first created it is in 'stand-by' mode, and will not fire triggers. The misfire/recovery process will be started, if it is the initial call to this action on this scheduler instance;
- **Standby Scheduler**, temporarily halts the Scheduler's firing of triggers. When 'Start Scheduler' is called (to bring the scheduler out of stand-by mode), trigger misfire instructions will NOT be applied during the start - any misfires will be detected immediately afterward. The scheduler can be re-started at any time;
- **Shutdown Scheduler**, halts the Scheduler's firing of triggers, and cleans up all resources associated with the Scheduler, waiting jobs to complete (the scheduler cannot be re-started and requires Tomcat restart);
- **Force Shutdown Scheduler**, halts the Scheduler's firing of triggers, and cleans up all resources associated with the Scheduler (the scheduler cannot be re-started and requires Tomcat restart);
- **Pause Triggers**, pauses all triggers, after using this method 'Resume Triggers' must be called to clear the scheduler's state of 'remembering' that all new triggers will be paused as they are added;
- **Resume Triggers**, resumes (un-pauses) all triggers on every group;
- **Nodes Status**, views the nodes status;
- **Nodes Log**, views the nodes status log;
- **Log**, views the log;
- **Truncate Catalina**, truncate the catalina log file of the scheduler;
- **Back**, return back;
- **Home**, link to home page;
- **Clear Scheduler**, clears (deletes) all scheduling data: all Jobs, Triggers, Calendars;

In the following, the sections of the SCE Scheduler web interface are described.

1 Home

The main status page lists the table of the events occurring in the scheduler, sorted (default) by descending ID.



The columns of the table are:

- **SCHEDULER NAME**, the name of the scheduler that executed the reported event. This column also reports the following icons (from left to right)
  - *Edit Job*, edits the details of the job (loads the **newJob.php** page);
  - *Delete Job*, deletes the job from the scheduler;
  - *Resume Job*, resumes a paused job;
  - *Pause Job*, pauses a job;
  - *Interrupt Job*, stops the job;
  - *View Triggers*, views the triggers associated to a job;
  - *Trigger Job*, immediately fires a triggers associated to a job;
- **ID**, the autoincrement integer ID;
- **FIRE_INSTANCE_ID**, the unique firing ID;
- **DATE**, the date at which occurred the reported event;
- **JOB NAME**, the name of the job;
- **JOB GROUP**, the group of the trigger;
- **STATUS**, the status of the reported event (i.e., FIRED, RUNNING, SUCCESS, FAILED);
- **TRIGGER NAME**, the name of the trigger;
- **TRIGGER GROUP**, the group of the trigger;
- **PREV FIRE TIME**, the previous time at which the reported trigger fired;
- **NEXT FIRE TIME**, the next time at which the reported trigger will fire;
- **REFIRE COUNT**, how many times the trigger associated to this event will be fired from now on (0 indicates forever);
- **RESULT**, the result returned by the job;
- **SCHEDULER INSTANCE ID**, the scheduler instance unique ID;
- **IP ADDRESS**, the IPv4 addresses of the scheduler. In case of multiple IP addresses, all of them are reported here;

The page also reports the main details about the current scheduler status (e.g., memory, cpu usage, jobs running).

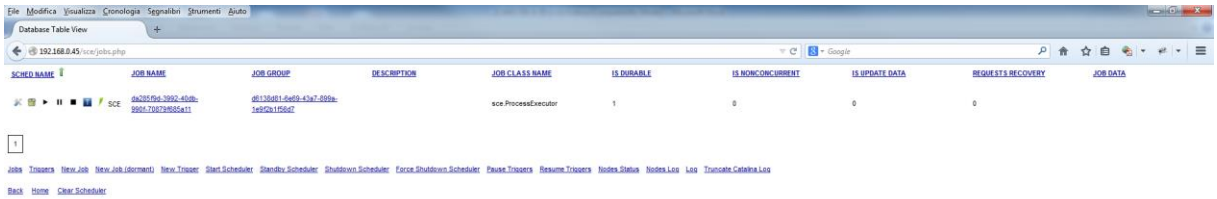
The results of the page are paginated, and can be sorted (ascending or descending) by every column of the table.

At the bottom of the status page, there is a **Push Mode** button that makes the web page automatically refresh every 3 seconds.

When in push mode, each column of the table includes a text box, that can be used to filter the results.

2 Jobs

The jobs page lists the table of the jobs registered in the scheduler.



The columns of the table are:

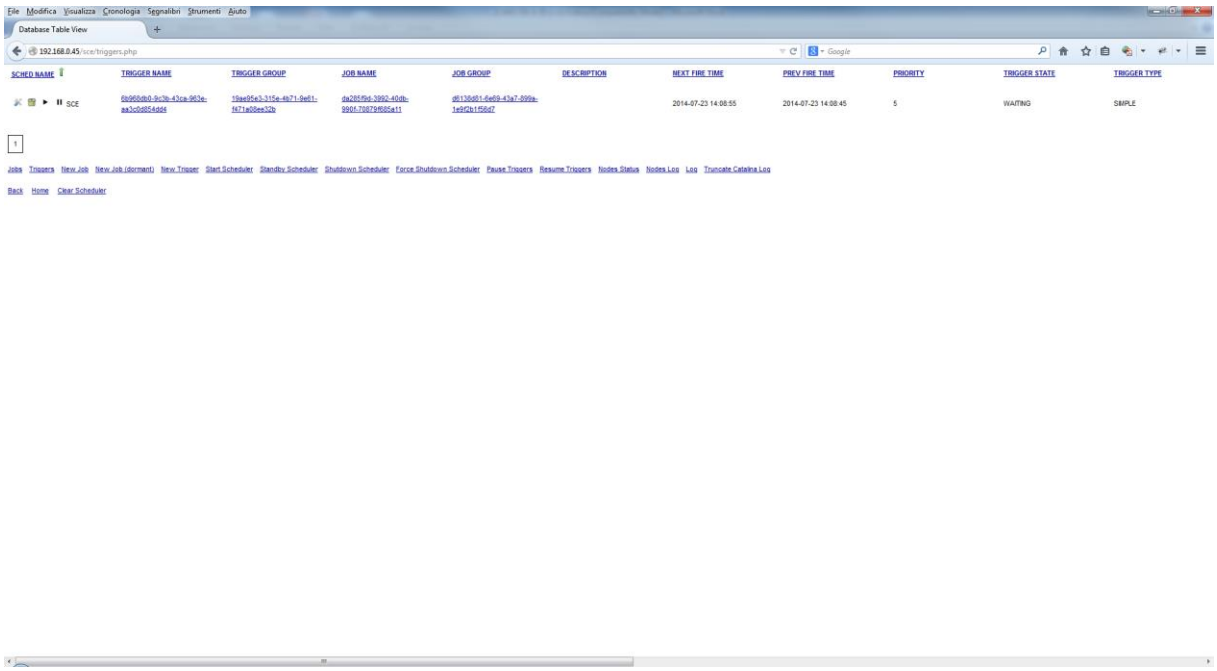
- **SCHED NAME**, the name of the scheduler that executed the reported event. This column also reports the following icons (from left to right)
  - *Edit Job*, edits the details of the job (loads the **newJob.php** page);
  - *Delete Job*, deletes the job from the scheduler;
  - *Resume Job*, resumes a paused job;
  - *Pause Job*, pauses a job;
  - *Interrupt Job*, stops the job;
  - *View Triggers*, views the triggers associated to a job;
- **JOB NAME**, the name of the job. Clicking on the value reported in this field, a job can be edited;
- **JOB GROUP**, the group of the trigger. Clicking on the value reported in this field, a job can be edited;

- **DESCRIPTION**, the description of the job;
- **JOB CLASS NAME**, the class name of the job (i.e., RESTJob, RESTJobStateful, RESTXMLJob, RESTXMLStateful, ProcessExecutorJob, ProcessExecutorStateful). A Job instance can be defined as "stateful" or "non-stateful". Non-stateful jobs only have their JobDataMap stored at the time they are added to the scheduler. This means that any changes made to the contents of the job data map during execution of the job will be lost, and will not seen by the job the next time it executes.
- **IS DURABLE**, reports whether the job is durable or not. A durable job should remain stored after it is orphaned. If a job is non-durable, it is automatically deleted from the scheduler once there are no longer any active triggers associated with it. In other words, non-durable jobs have a life span bounded by the existence of its triggers;
- **IS NONCONCURRENT**, reports the nonconcurrency attribute of the job. If set to true (1) then the job is disallowed to execute concurrently (new triggers that occur before the completion of the current running job will be delayed);
- **IS UPDATE DATA**, reports wheter the job can update its job data during execution;
- **REQUESTS RECOVERY**, in clustering mode, this parameter is set to true to ensure job fail-over. If a job 'requests recovery', and it is executing during the time of a 'hard shutdown' of the scheduler (i.e. the process it is running within crashes, or the machine is shut off), then it is re-executed when the scheduler is started again;
- **JOB DATA**, the job data included in the job data map;

The results of the page are paginated, and can be sorted (ascending or descending) by every column of the table.

3 Triggers

The triggers page lists the table of the triggers registered in the scheduler.



The columns of the table are:

- **SCHED NAME**, the name of the scheduler that executed the reported event. This column also reports the following icons (from left to right)
    - **Edit Trigger**, edits the details of the trigger (loads the **newTrigger.php** page);
    - **Delete Trigger**, deletes the job from the scheduler;
    - **Resume Trigger**, resumes a paused trigger;
    - **Pause Trigger**, pauses a trigger;
- **TRIGGER NAME**, the name of the trigger;
- **TRIGGER GROUP**, the group of the trigger;
- **JOB NAME**, the name of the job. Clicking on the value reported in this field, a job can be edited;
- **JOB GROUP**, the group of the trigger. Clicking on the value reported in this field, a job can be edited;
- **DESCRIPTION**, the description of the trigger;
- **NEXT FIRE TIME**, the next time at which the reported trigger will fire;
- **PREV FIRE TIME**, the previous time at which the reported trigger fired;
- **PRIORITY**, the priority of execution of the trigger (the higher the priority, the higher the probability for the trigger to be fired on time);
- **TRIGGER STATE**, the status of the trigger
    - *WAITING*, the normal state of a trigger, waiting for its fire time to arrive and be acquired for firing by a scheduler;
    -
    - *PAUSED*, means that one of the scheduler's pause methods was used. The trigger is not eligible for being fired until it is resumed;
    - *ACQUIRED*, a scheduler node has identified this trigger as the next trigger it will fire - may still be waiting for its fire time to arrive. After it fires the trigger will be updated (per its repeat settings, if any) and placed back into the *WAITING* state (or be deleted if it does not repeat again);
    - *BLOCKED*, the trigger is prevented from being fired because it relates to a StatefulJob that is already executing. When the stateful job completes its execution, all triggers relating to that job will return to the *WAITING* state;
- **TRIGGER TYPE**, the trigger type (*SIMPLE*);
- **START TIME**, reports the date when the trigger started firing;
- **END TIME**, reports the date when the trigger will stop firing;
- **CALENDAR NAME**, the name of the calendar associated to the trigger;
- **MISFIRE INSTR**, the misfire instruction of the trigger
    - *MISFIRE_INSTRUCTION_IGNORE_MISFIRE_POLICY*, instructs the scheduler that the Trigger will never be evaluated for a misfire situation, and that the scheduler will simply try to fire it as soon as it can, and then update the Trigger as if it had fired at the proper time. If a trigger uses this instruction, and it has missed several of its scheduled firings, then several rapid firings may occur as the trigger attempt to catch back up to where it would have been. For example, a simple trigger that fires every 15 seconds which has misfired for 5 minutes will fire 20 times once it gets the chance to fire;

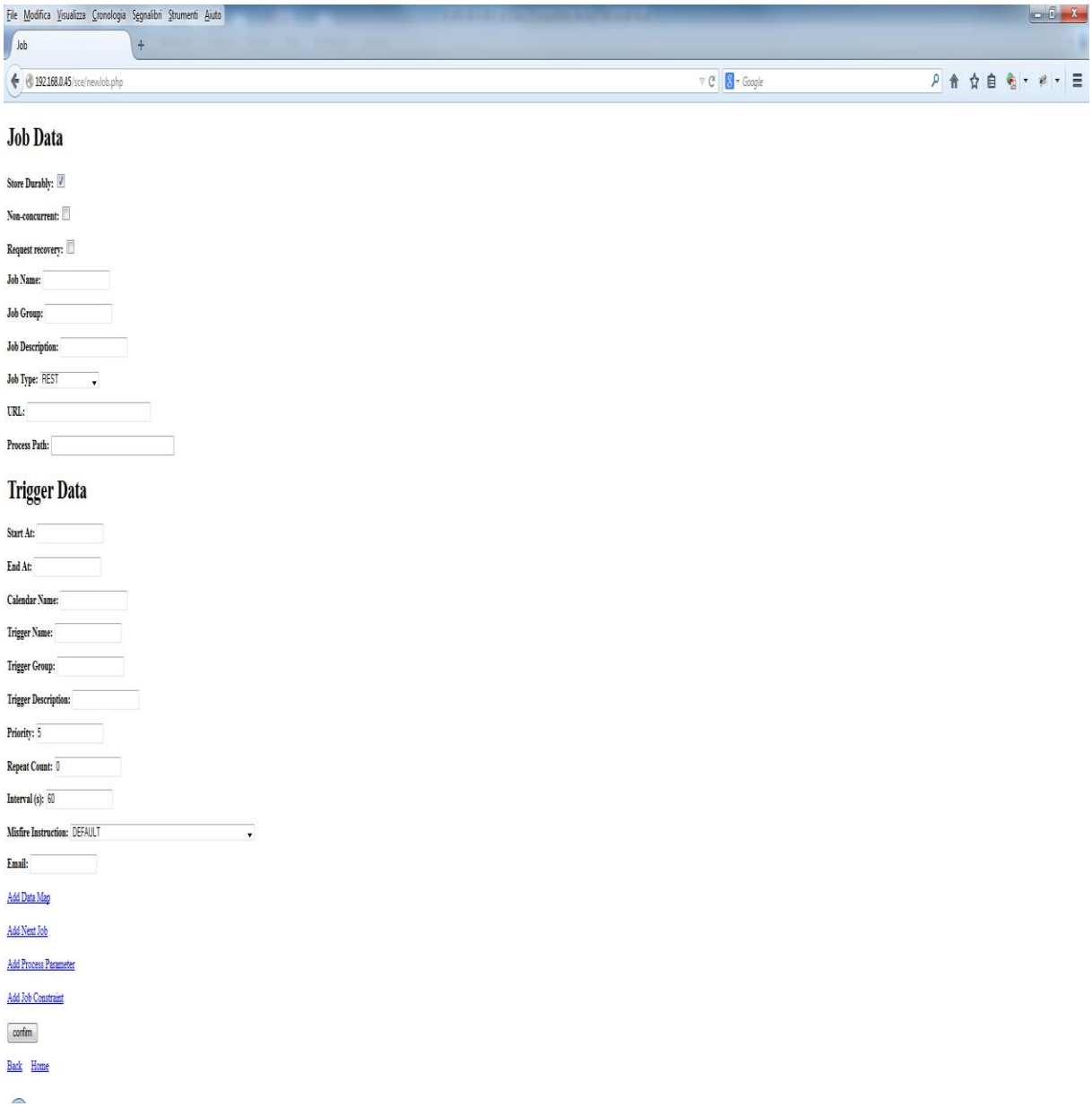
- *MISFIRE_INSTRUCTION_SMART_POLICY*, instructs the scheduler that upon a misfire situation, the update after misfire method will be called on the Trigger to determine the mis-fire instruction, which logic will be trigger-implementation-dependent;
- *MISFIRE_INSTRUCTION_FIRE_NOW*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be fired now by scheduler. This instruction should typically only be used for "one-shot" (non-repeating) triggers. If it is used on a trigger with a repeat count > 0 then it is equivalent to the instruction;
- *MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_EXISTING_REPEAT_COUNT*, instructs the Scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to "now" (even if the associated Calendar excludes "now") with the repeat count left as-is. This does obey the trigger end-time however, so if "now" is after the end-time the Trigger will not fire again. Use of this instruction causes the trigger to "forget" the start-time and repeat-count that it was originally setup with (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time);
- *MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_REMAINING_REPEAT_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to "now" (even if the associated Calendar excludes "now") with the repeat count set to what it would be, if it had not missed any firings. This does obey the trigger end-time however, so if "now" is after the end-time the Trigger will not fire again. Use of this instruction causes the trigger to "forget" the start-time and repeat-count that it was originally setup with. Instead, the repeat count on the trigger will be changed to whatever the remaining repeat count is (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time). This instruction could cause the Trigger to go to the "*COMPLETE*" state after firing "now", if all the repeat-fire-times where missed;
- *MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_REMAINING_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to the next scheduled time after "now" - taking into account any associated Calendar, and with the repeat count set to what it would be, if it had not missed any firings. This instruction could cause the trigger to go directly to the "*COMPLETE*" state if all fire-times where missed;
- *MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_EXISTING_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to the next scheduled time after "now" - taking into account any associated Calendar, and with the repeat count left unchanged. This instruction could cause the Trigger to go directly to the "*COMPLETE*" state if the end-time of the trigger has arrived;

- **JOB DATA**, the job data included in the job data map;

The results of the page are paginated, and can be sorted (ascending or descending) by every column of the table.

4 New Job

This page includes a form for creating new jobs and registering them in the scheduler.



The form includes the following fields and checkboxes:

4.1 Job Data

- **Store Durably**, set whether or not the job should remain stored after it is orphaned. If a job is non-durable, it is automatically deleted from the scheduler once there are no longer any active triggers associated with it. In other words, non-durable jobs have a life span bounded by the existence of its triggers;
- **Non-concurrent**, set the job to disallow to execute concurrently (new triggers that occur before the completion of the current running job will be delayed);
- **Request recovery**, in clustering mode, this parameter must be set to true to ensure job fail-over. If a job 'requests recovery', and it is executing during the time of a 'hard shutdown' of the scheduler (i.e. the process it is running within crashes, or the machine is shut off), then it is re-executed when the scheduler is started again;
- **Job Name**, set the job name;

- **Job Group**, set the job group;
- **Job Description**, set the job description;
- **Job Type**, set the job type (i.e., REST, RESTXML, ProcessExecutor);
- **URL**, set the url to be called (enabled for REST-like jobs);
- **Process Path**, set the path of process to be executed (enabled for ProcessExecutor-like jobs);

4.2 Trigger Data

- **Start At**, set the time the trigger should start at - the trigger may or may not fire at this time - depending upon the schedule configured for the trigger;
- **End At**, set the time at which the trigger will no longer fire - even if it's schedule has remaining repeats;
- **Calendar Name**, set the name of the Calendar that should be applied to this trigger's schedule;
- **Trigger Name**, set the trigger name;
- **Trigger Group**, set the trigger group;
- **Trigger Description**, set the given (human-meaningful) description of the Trigger;
- **Priority**, set the priority of execution of the trigger (the higher the priority, the higher the probability for the trigger to be fired on time);
- **Repeat Count**, set how many times this trigger will be fired (0 indicates forever);
- **Interval (s)**, set the repeat interval in seconds;
- **Misfire Instruction**, the misfire instruction of the trigger
    - *MISFIRE_INSTRUCTION_IGNORE_MISFIRE_POLICY*, instructs the scheduler that the Trigger will never be evaluated for a misfire situation, and that the scheduler will simply try to fire it as soon as it can, and then update the Trigger as if it had fired at the proper time. If a trigger uses this instruction, and it has missed several of its scheduled firings, then several rapid firings may occur as the trigger attempt to catch back up to where it would have been. For example, a simple trigger that fires every 15 seconds which has misfired for 5 minutes will fire 20 times once it gets the chance to fire;
    - *MISFIRE_INSTRUCTION_SMART_POLICY*, instructs the scheduler that upon a misfire situation, the update after misfire method will be called on the Trigger to determine the mis-fire instruction, which logic will be trigger-implementation-dependent;
    - *MISFIRE_INSTRUCTION_FIRE_NOW*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be fired now by scheduler. This instruction should typically only be used for "one-shot" (non-repeating) triggers. If it is used on a trigger with a repeat count > 0 then it is equivalent to the instruction;
    - *MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_EXISTING_REPEAT_COUNT*, instructs the Scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to "now" (even if the associated Calendar excludes "now") with the repeat count left as-is. This does obey the trigger end-time however, so if "now" is after the end-time the Trigger will not fire again. Use of this instruction causes the trigger to "forget" the start-time and repeat-count that it was originally setup with (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time);

- *MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_REMAINING_REPEAT_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to "now" (even if the associated Calendar excludes "now") with the repeat count set to what it would be, if it had not missed any firings. This does obey the trigger end-time however, so if "now" is after the end-time the Trigger will not fire again. Use of this instruction causes the trigger to "forget" the start-time and repeat-count that it was originally setup with. Instead, the repeat count on the trigger will be changed to whatever the remaining repeat count is (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time). This instruction could cause the Trigger to go to the "*COMPLETE*" state after firing "now", if all the repeat-fire-times where missed;
- *MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_REMAINING_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to the next scheduled time after "now" - taking into account any associated Calendar, and with the repeat count set to what it would be, if it had not missed any firings. This instruction could cause the trigger to go directly to the "*COMPLETE*" state if all fire-times where missed;
- *MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_EXISTING_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to the next scheduled time after "now" - taking into account any associated Calendar, and with the repeat count left unchanged. This instruction could cause the Trigger to go directly to the "*COMPLETE*" state if the end-time of the trigger has arrived;

- **Email**, set the email where to send a notification message upon job completion;

In addition to these fields, there are four links that can be used to add data to the job

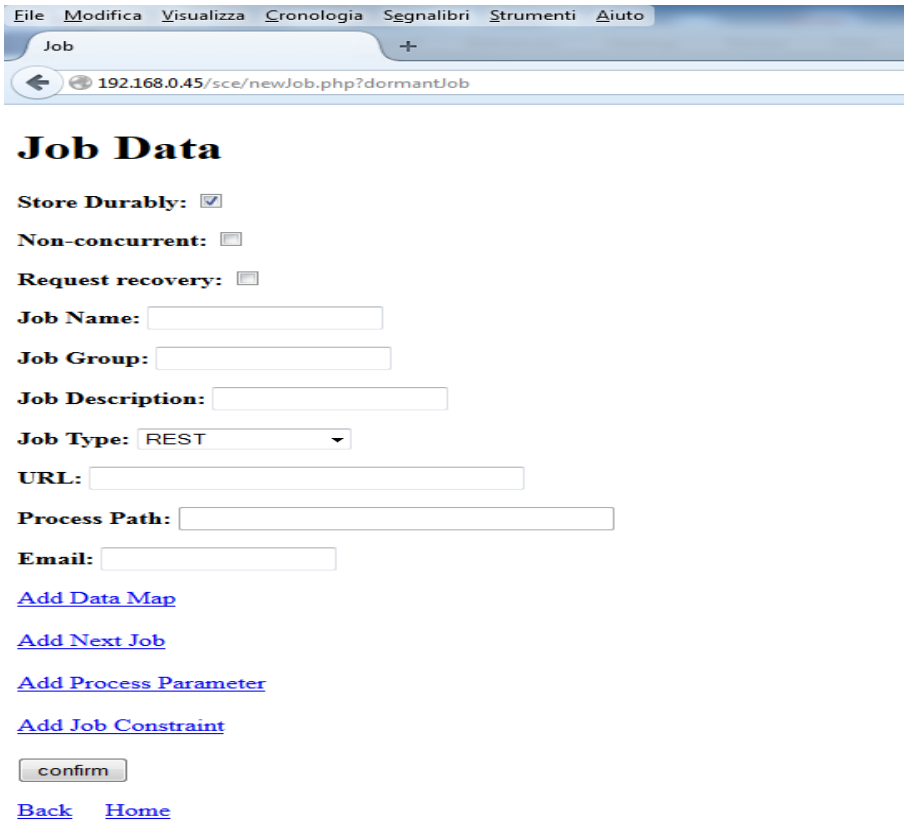
- **Add Data Map**, clicking on this link the user can specify a key-value pair for the job;
- **Add Next Job**, clicking on this link the user can specify a conditional expression that execute another job (i.e., IF RESULT - THEN TRIGGER);
- **Add Process Parameter**, clicking on this link the user can specify process parameters (key = label, value = parameter value) for the job. This must be used only when defining jobs of type ProcessExecutor/ProcessExecutorStateful;
- **Add Job Constraint**, clicking on this link the user can specify job contraints using operators ("==", "!=", "<", ">", "<=", ">=")
  - *OS Architecture*, the architecture of the operating system (e.g., amd64);
  - *Available Processors*, the number of available cpu(s);
  - *OS Name*, the operating system name (e.g., Windows , Linux, Mac);
  - *System Load Average*, the system load average for the last minute. The system load average is the sum of the number of runnable entities queued to the available processors and the number of runnable entities running on the available processors averaged over a period of time. The way in which the load average is calculated is operating system specific but is typically a damped time-dependent average. If the load average is not available, a negative value is returned. This value is designed to provide a hint about the system load and may be queried frequently. The load

average may be unavailable on some platform where it is expensive to implement this method;
- *OS Version*, the operating system version (e.g., 3.13.0-24-generic);
- *Committed Virtual Memory Size*, the amount of virtual memory that is guaranteed to be available to the running process in bytes, or -1 if this operation is not supported;
- *Free Physical Memory Size*, the amount of free physical memory in bytes;
- *Free Swap Space Size*, the amount of free swap space in bytes;
- *Process Cpu Load*, the recent cpu usage for the Java Virtual Machine process. This value is a double in the [0.0, 1.0] interval. A value of 0.0 means that none of the CPUs were running threads from the JVM process during the recent period of time observed, while a value of 1.0 means that all CPUs were actively running threads from the JVM 100% of the time during the recent period being observed. Threads from the JVM include the application threads as well as the JVM internal threads. All values between 0.0 and 1.0 are possible depending of the activities going on in the JVM process and the whole system. If the Java Virtual Machine recent CPU usage is not available, the method returns a negative value;
- *Process Cpu Time*, the cpu time used by the process on which the Java virtual machine is running in nanoseconds. The returned value is of nanoseconds precision but not necessarily nanoseconds accuracy. This value reports -1 if the platform does not support this operation;
- *System Cpu Load*, the recent cpu usage for the whole system. This value is a double in the [0.0, 1.0] interval. A value of 0.0 means that all CPUs were idle during the recent period of time observed, while a value of 1.0 means that all CPUs were actively running 100% of the time during the recent period being observed. All values betweens 0.0 and 1.0 are possible depending of the activities going on in the system. If the system recent cpu usage is not available, the method returns a negative value;
- *Total Physical Memory Size*, the total amount of physical memory in bytes;
- *Total Swap Space Size*, the total amount of swap space in bytes;
- *IP Address*, the IP address of the scheduler;

Clicking on the submit button, will cause the form data to be submitted to the scheduler. A notification popup will appear, reporting the success (timestamp of the accepted request) or failure (error message) from the scheduler.

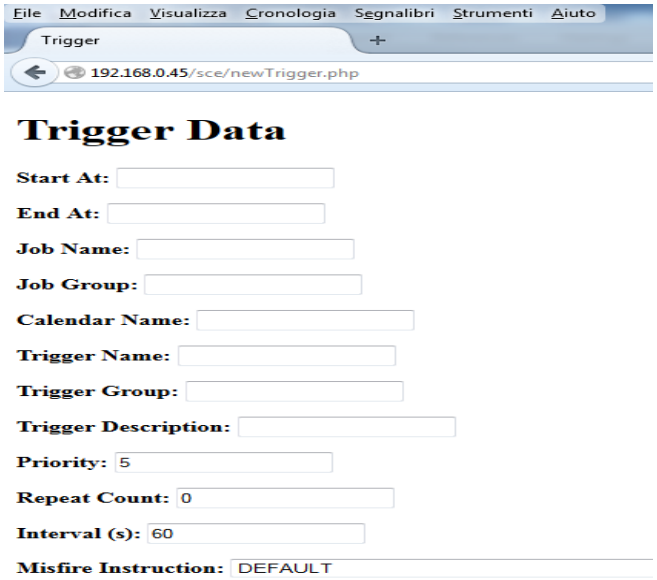
5 New Job (dormant)

This page is the same as New Job, excepting that it does not allow to specify a trigger associated to the job.



6 New Trigger

This page includes a form for creating new triggers and registering them in the scheduler.



The form includes the following fields and checkboxes:

- **Start At**, set the time the trigger should start at - the trigger may or may not fire at this time - depending upon the schedule configured for the trigger;

- **End At**, set the time at which the trigger will no longer fire - even if it's schedule has remaining repeats;
- **Job Name**, set the name of the job which should be fired by the produced trigger;
- **Job Group**, set the group of the job which should be fired by the produced trigger;
- **Calendar Name**, set the name of the Calendar that should be applied to this trigger's schedule;
- **Trigger Name**, set the trigger name;
- **Trigger Group**, set the trigger group;
- **Trigger Description**, set the given (human-meaningful) description of the trigger;
- **Priority**, set the priority of execution of the trigger (the higher the priority, the higher the probability for the trigger to be fired on time);
- **Repeat Count**, set how many times this trigger will be fired (0 indicates forever);
- **Interval (s)**, set the repeat interval in seconds;
- **Misfire Instruction**, the misfire instruction of the trigger
    - *MISFIRE_INSTRUCTION_IGNORE_MISFIRE_POLICY*, instructs the scheduler that the Trigger will never be evaluated for a misfire situation, and that the scheduler will simply try to fire it as soon as it can, and then update the Trigger as if it had fired at the proper time. If a trigger uses this instruction, and it has missed several of its scheduled firings, then several rapid firings may occur as the trigger attempt to catch back up to where it would have been. For example, a simple trigger that fires every 15 seconds which has misfired for 5 minutes will fire 20 times once it gets the chance to fire;
    - *MISFIRE_INSTRUCTION_SMART_POLICY*, instructs the scheduler that upon a misfire situation, the update after misfire method will be called on the Trigger to determine the mis-fire instruction, which logic will be trigger-implementation-dependent;
    - *MISFIRE_INSTRUCTION_FIRE_NOW*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be fired now by scheduler. This instruction should typically only be used for "one-shot" (non-repeating) triggers. If it is used on a trigger with a repeat count > 0 then it is equivalent to the instruction;
    - *MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_EXISTING_REPEAT_COUNT*, instructs the Scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to "now" (even if the associated Calendar excludes "now") with the repeat count left as-is. This does obey the trigger end-time however, so if "now" is after the end-time the Trigger will not fire again. Use of this instruction causes the trigger to "forget" the start-time and repeat-count that it was originally setup with (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time);
    - *MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_REMAINING_REPEAT_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to "now" (even if the associated Calendar excludes "now") with the repeat count set to what it would be, if it had not missed any firings. This does obey the trigger end-time however, so if "now" is after the end-time the Trigger will not fire again. Use of this instruction causes the trigger to "forget" the start-time and repeat-count that it was originally setup with. Instead, the repeat count on the

trigger will be changed to whatever the remaining repeat count is (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time). This instruction could cause the Trigger to go to the "*COMPLETE*" state after firing "now", if all the repeat-fire-times where missed;

- *MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_REMAINING_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to the next scheduled time after "now" - taking into account any associated Calendar, and with the repeat count set to what it would be, if it had not missed any firings. This instruction could cause the trigger to go directly to the "*COMPLETE*" state if all fire-times where missed;

- *MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_EXISTING_COUNT*, instructs the scheduler that upon a misfire situation, the simple trigger wants to be re-scheduled to the next scheduled time after "now" - taking into account any associated Calendar, and with the repeat count left unchanged. This instruction could cause the Trigger to go directly to the "*COMPLETE*" state if the end-time of the trigger has arrived;

Clicking on the submit button, will cause the form data to be submitted to the scheduler. A notification popup will appear, reporting the success (true) or failure (error message) from the scheduler.

7 Nodes Status

The nodes status page lists the current status of each node included in the scheduler cluster. Each node updates its status every 60 seconds.

The columns of the table are:

- **ID**, the autoincrement integer ID;
- **DATE**, the date at which occurred the reported event;
- **IP ADDRESS**, the IPv4 addresses of the scheduler. In case of multiple IP addresses, all of them are reported here;
- **SCHEDULER INSTANCE ID**, the scheduler instance unique ID;
- **CPU LOAD**, the recent cpu usage for the whole system. This value is a double in the [0.0, 1.0] interval. A value of 0.0 means that all CPUs were idle during the recent period of time observed, while a value of 1.0 means that all CPUs were actively running 100% of the time during the recent period being observed. All values between 0.0 and 1.0 are possible depending of the activities going on in the system. If the system recent cpu usage is not available, the method returns a negative value;

- **FREE PHYSICAL MEMORY**, the amount of free physical memory in bytes;
- **JOBS EXECUTED**, the number of jobs executed since the scheduler started;
- **SCHEDULER NAME**, the name of the scheduler that executed the reported event;
- **RUNNING SINCE**, the time since when the scheduler is running;
- **CLUSTERED**, reports whether the scheduler is working in cluster mode;
- **PERSISTENCE**, reports whether the scheduler is working in persistence mode (i.e., using a job store);
- **REMOTE SCHEDULER**, reports whether the scheduler supports remote commands through RMI;
- **CURRENTLY EXECUTING JOBS**, reports the number of currently executing jobs;
- **CPU LOAD JVM**, the recent cpu usage for the Java Virtual Machine process. This value is a double in the [0.0, 1.0] interval. A value of 0.0 means that none of the CPUs were running threads from the JVM process during the recent period of time observed, while a value of 1.0 means that all CPUs were actively running threads from the JVM 100% of the time during the recent period being observed. Threads from the JVM include the application threads as well as the JVM internal threads. All values between 0.0 and 1.0 are possible depending of the activities going on in the JVM process and the whole system. If the Java Virtual Machine recent CPU usage is not available, the method returns a negative value;
- **SYSTEM LOAD AVERAGE**, the system load average for the last minute. The system load average is the sum of the number of runnable entities queued to the available processors and the number of runnable entities running on the available processors averaged over a period of time. The way in which the load average is calculated is operating system specific but is typically a damped time-dependent average. If the load average is not available, a negative value is returned. This value is designed to provide a hint about the system load and may be queried frequently. The load average may be unavailable on some platform where it is expensive to implement this method;
- **OPERATING SYSTEM VERSION**, the operating system version (e.g., 3.13.0-24-generic);
- **COMMITTED VIRTUAL MEMORY**, the amount of virtual memory that is guaranteed to be available to the running process in bytes, or -1 if this operation is not supported;
- **OPERATING SYSTEM NAME**, the operating system name (e.g., Windows , Linux, Mac);
- **FREE SWAP SPACE**, the amount of free swap space in bytes;
- **PROCESS CPU TIME**, the cpu time used by the process on which the Java virtual machine is running in nanoseconds. The returned value is of nanoseconds precision but not necessarily nanoseconds accuracy. This value reports -1 if the platform does not support this operation;
- **TOTAL PHYSICAL MEMORY**, the total amount of physical memory in bytes;
- **NUMBER OF PROCESSORS**, the number of available cpu(s);
- **OPERATING SYSTEM ARCHITECTURE**, the architecture of the operating system (e.g., amd64);
- **TOTAL SWAP SPACE**, the total amount of swap space in bytes;
- **IS SCHEDULER STANDBY**, reports whether the scheduler is in standby;
- **IS SCHEDULER SHUTDOWN**, reports whether the scheduler is shutdown;
- **IS SCHEDULER STARTED**, reports whether the scheduler is running;
- **TOTAL DISK SPACE**, reports the total disk space of the scheduler's host machine;
- **UNALLOCATED DISK SPACE**, reports the unallocated disk space of the scheduler's host machine;

- **USABLE DISK SPACE**, reports the usable disk space of the scheduler's host machine;

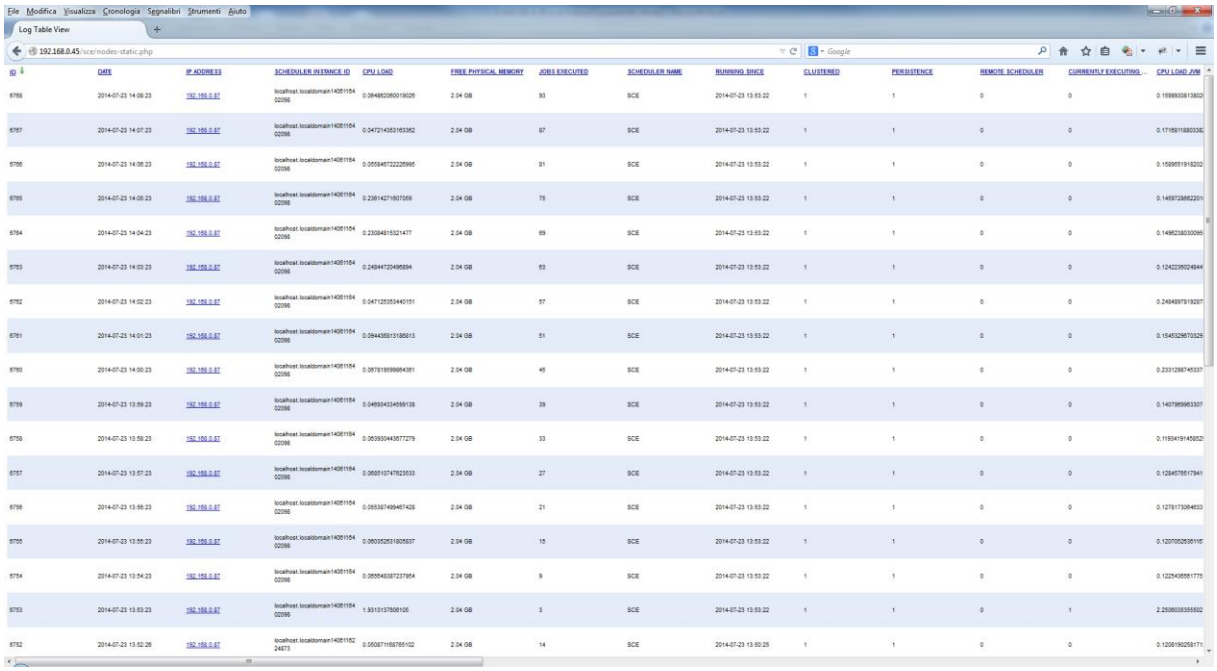
The results of the page are paginated, and can be sorted (ascending or descending) by every column of the table.

At the bottom of the status page, there is a **Push Mode** button that makes the web page automatically refresh every 3 seconds.

When in push mode, each column of the table includes a text box, that can be used to filter the results.

8 Nodes Log

The nodes status page lists the history status of each node included in the scheduler cluster. Each node updates its status every 60 seconds.



The columns of the table are:

- **ID**, the autoincrement integer ID;
- **DATE**, the date at which occurred the reported event;
- **IP ADDRESS**, the IPv4 addresses of the scheduler. In case of multiple IP addresses, all of them are reported here;
- **SCHEDULER INSTANCE ID**, the scheduler instance unique ID;
- **CPU LOAD**, the recent cpu usage for the whole system. This value is a double in the [0.0, 1.0] interval. A value of 0.0 means that all CPUs were idle during the recent period of time observed, while a value of 1.0 means that all CPUs were actively running 100% of the time during the recent period being observed. All values between 0.0 and 1.0 are possible depending of the activities going on in the system. If the system recent cpu usage is not available, the method returns a negative value;
- **FREE PHYSICAL MEMORY**, the amount of free physical memory in bytes;

- **JOBS EXECUTED**, the number of jobs executed since the scheduler started;
- **SCHEDULER NAME**, the name of the scheduler that executed the reported event;
- **RUNNING SINCE**, the time since when the scheduler is running;
- **CLUSTERED**, reports whether the scheduler is working in cluster mode;
- **PERSISTENCE**, reports whether the scheduler is working in persistence mode (i.e., using a job store);
- **REMOTE SCHEDULER**, reports whether the scheduler supports remote commands through RMI;
- **CURRENTLY EXECUTING JOBS**, reports the number of currently executing jobs;
- **CPU LOAD JVM**, the recent cpu usage for the Java Virtual Machine process. This value is a double in the [0.0, 1.0] interval. A value of 0.0 means that none of the CPUs were running threads from the JVM process during the recent period of time observed, while a value of 1.0 means that all CPUs were actively running threads from the JVM 100% of the time during the recent period being observed. Threads from the JVM include the application threads as well as the JVM internal threads. All values between 0.0 and 1.0 are possible depending of the activities going on in the JVM process and the whole system. If the Java Virtual Machine recent CPU usage is not available, the method returns a negative value;
- **SYSTEM LOAD AVERAGE**, the system load average for the last minute. The system load average is the sum of the number of runnable entities queued to the available processors and the number of runnable entities running on the available processors averaged over a period of time. The way in which the load average is calculated is operating system specific but is typically a damped time-dependent average. If the load average is not available, a negative value is returned. This value is designed to provide a hint about the system load and may be queried frequently. The load average may be unavailable on some platform where it is expensive to implement this method;
- **OPERATING SYSTEM VERSION**, the operating system version (e.g., 3.13.0-24-generic);
- **COMMITTED VIRTUAL MEMORY**, the amount of virtual memory that is guaranteed to be available to the running process in bytes, or -1 if this operation is not supported;
- **OPERATING SYSTEM NAME**, the operating system name (e.g., Windows , Linux, Mac);
- **FREE SWAP SPACE**, the amount of free swap space in bytes;
- **PROCESS CPU TIME**, the cpu time used by the process on which the Java virtual machine is running in nanoseconds. The returned value is of nanoseconds precision but not necessarily nanoseconds accuracy. This value reports -1 if the platform does not support this operation;
- **TOTAL PHYSICAL MEMORY**, the total amount of physical memory in bytes;
- **NUMBER OF PROCESSORS**, the number of available cpu(s);
- **OPERATING SYSTEM ARCHITECTURE**, the architecture of the operating system (e.g., amd64);
- **TOTAL SWAP SPACE**, the total amount of swap space in bytes;
- **IS SCHEDULER STANDBY**, reports whether the scheduler is in standby;
- **IS SCHEDULER SHUTDOWN**, reports whether the scheduler is shutdown;
- **IS SCHEDULER STARTED**, reports whether the scheduler is running;
- **TOTAL DISK SPACE**, reports the total disk space of the scheduler's host machine;
- **UNALLOCATED DISK SPACE**, reports the unallocated disk space of the scheduler's host machine;
- **USABLE DISK SPACE**, reports the usable disk space of the scheduler's host machine;

The results of the page are paginated, and can be sorted (ascending or descending) by every column of the table.

At the bottom of the status page, there is a **Push Mode** button that makes the web page automatically refresh every 3 seconds.

When in push mode, each column of the table includes a text box, that can be used to filter the results.

9 Log

The log page lists the status history of the scheduler.

The columns of the table are:

- **ID**, the autoincrement integer ID;
- **DATE**, the date at which occurred the reported event;
- **FIRE_INSTANCE_ID**, the unique firing ID;
- **JOB NAME**, the name of the job;
- **JOB GROUP**, the group of the trigger;
- **TRIGGER NAME**, the name of the trigger;
- **TRIGGER GROUP**, the group of the trigger;
- **STATUS**, the status of the reported event (i.e., FIRED, RUNNING, COMPLETE, SUCCESS, FAILED);
- **PREV FIRE TIME**, the previous time at which the reported trigger fired;
- **NEXT FIRE TIME**, the next time at which the reported trigger will fire;
- **REFIRE COUNT**, how many times the trigger associated to this event will be fired from now on (0 indicates forever);
- **RESULT**, the result returned by the job;

- **SCHEDULER INSTANCE ID**, the scheduler instance unique ID;
- **SCHEDULER NAME**, the name of the scheduler that executed the reported event
- **IP ADDRESS**, the IPv4 addresses of the scheduler. In case of multiple IP addresses, all of them are reported here;
- **LOGGER**, the logging class (i.e., LoggingTriggerHistoryPluginCustom);
- **LEVEL**, the message level (i.e., INFO, DEBUG, ERROR);
- **MESSAGE**, the human-meaningful event log message;

The results of the page are paginated, and can be sorted (ascending or descending) by every column of the table.

At the bottom of the status page, there is a **Push Mode** button that makes the web page automatically refresh every 3 seconds.

When in push mode, each column of the table includes a text box, that can be used to filter the results

10 Programmatically call the Scheduler

Calling the scheduler to insert/update/delete/pause/resume a job requires encapsulating data in an associative array and encode it in JSON, before posting to the webservice as the following associative array: array ('json' => jsonArray), where jsonArray is the json encoded array.

Endpoint URL: http://hostname:8080/SmartCloudEngine/index.jsp

10.1 Insert a Job

Here are the keys of the associative array to be built, with example values. Those with # are reserved keys.

"**startAt**":"1411035453000"
"**endAt**":"1411164000000"
"**withDescription**":"a"
"**withIdentityNameGroup**":["a","a"]
"**withPriority**":"5"
"**repeatForever**":"true"
"**withIntervalInSeconds**":"60"
"**storeDurably**":"true"
"**requestRecovery**":"true"
"**withJobIdentityNameGroup**":["a","a"]
"**withJobDescription**":"a"
"**id**":"scheduleJob"
"**jobClass**":"ProcessExecutorJob"
"**jobDataMap**":{
    "#notificationEmail":daniele.cenni@unifi.it
    "#isNonConcurrent":"true"
    "#nextJobs":
        "**operator**":"!="
        "**result**":"1"
        "**jobName**":"nextjob1"

```
            "jobGroup":"nextgroup1
        "#processParameters":
            "processPath":"/var/www/script.sh"
            "process1":"param1"
            "process2":"param2"
        "#jobConstraints":
            ["operator":"=="
            "result":"1"
            "jobName":"nextjob1"
            "jobGroup":"nextgroup1"
            "systemParameterName":"osArch"
            "value":"linux"]
            ["operator":"=="
            "result":"1"
            "jobName":"nextjob1"
            "jobGroup":"nextgroup1"
            "systemParameterName":"ipAddress"
            "value":"192.168.0.10"]
}
```


Here is a JSON example for inserting a job:


```
{"startAt":"1411035453000","endAt":"1411164000000","withDescription":"a","withIdentityNameGr
oup":["a","a"],"withPriority":"5","repeatForever":"true","withIntervalInSeconds":"60","storeDurably"
:"true","requestRecovery":"true","withJobIdentityNameGroup":["a","a"],"withJobDescription":"a","i
d":"scheduleJob","jobClass":"ProcessExecutorJob","jobDataMap":{"#notificationEmail":"daniele.cenn
i@unifi.it","#isNonConcurrent":"true","#nextJobs":"[{\\"operator\\":\\"!=\\",\\"result\\":\\"1\\",\\"j
obName\\":\\"nextjob1\\",\\"jobGroup\\":\\"nextgroup1\\"}]","#processParameters":"[{\\"processP
ath\\":\\"\\\\\\\/var\\\\\\\/www\\"},{\\"process1\\":\\"param1\\"},{\\"process2\\":\\"param2\\"}]","#
jobConstraints":"[{\\"operator\\":\\"==\\",\\"result\\":\\"1\\",\\"jobName\\":\\"nextjob1\\",\\"jobG
roup\\":\\"nextgroup1\\",\\"systemParameterName\\":\\"osArch\\",\\"value\\":\\"linux\\"},{\\"oper
ator\\":\\"==\\",\\"result\\":\\"1\\",\\"jobName\\":\\"nextjob1\\",\\"jobGroup\\":\\"nextgroup1\\",
\\"systemParameterName\\":\\"ipAddress\\",\\"value\\":\\"192.168.0.10\\"}]"}}
```


10.2 Updating a Job

Example of JSON to be posted for updating a job's data:


```
{"withIdentityNameGroup":["8022721e-b81d-4490-8137-f70e4e285480","5dc84377-150e-4a80-
a962-
f1157143ff1f"],"storeDurably":"true","requestRecovery":"false","withJobIdentityNameGroup":["job1
","group1"],"id":"updateJob","jobClass":"ProcessExecutorJob","jobDataMap":{"#isNonConcurrent":"t
rue","#nextJobs":"[{\\"operator\\":\\"!=\\",\\"result\\":\\"1\\",\\"jobName\\":\\"job3\\",\\"jobGrou
p\\":\\"group2\\"}]","#processParameters":"[{\\"processPath\\":\\"\\\\\\\/var\\\\\\\/www\\\\\\\/html
\\\\\\\/sce\\\\\\\/test\\\\\\\/javaBashTest.sh\\"}]"}}
```


10.3 Other operations

Some examples of JSON calls to perform other operations on the scheduler.

- {"**id**":"**isInStandbyMode**"} // wheter the scheduler is in standby
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **isShutdown**"} // wheter the scheduler was shutdown
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **isStarted**"} // wheter the scheduler is started
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **deleteJob**", "**jobName**": jobName, "**jobGroup**": jobGroup} // delete a job
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **resumeJob**"} // resume a job
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **pauseJob**", "**jobName**": jobName, "**jobGroup**": jobGroup} // pause a job
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **interruptJob**", "**jobName**": jobName, "**jobGroup**": jobGroup} // stop a job
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **unscheduleJob**", "**triggerName**": triggerName, "**triggerGroup**": triggerGroup} // delete a trigger
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **triggerJob**", "**jobName**": jobName, "**jobGroup**": jobGroup} // delete a job
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **deleteJob**", "**jobName**": jobName, "**jobGroup**": jobGroup} // delete a job
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **resumeTrigger**", "**triggerName**": triggerName, "**triggerGroup**": triggerGroup} // resume a trigger
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":" **pauseTrigger**", "**triggerName**": triggerName, "**triggerGroup**": triggerGroup} // pause a trigger
    - **Response**: {"0":["response"],"1":t/f} // t/f = ["true", "false"]
- {"**id**":"**startScheduler**"} // start the scheduler

- **Response**: {"0":["response"],"1":t/f} // t/f = [“true”, “false”]

- {“**id**”:“**standbyScheduler**”} // standby the scheduler
  - **Response**: {"0":["response"],"1":t/f} // t/f = [“true”, “false”]
- {“**id**”:“**shutdownScheduler**”} // shutdown the scheduler (wait for running jobs to complete)
  - **Response**: {"0":["response"],"1":t/f} // t/f = [“true”, “false”]
- {“**id**”:“**forceShutdownScheduler**”} // start the scheduler
  - **Response**: {"0":["response"],"1":t/f} // t/f = [“true”, “false”]
- {“**id**”:“**clearScheduler**”} // clear the scheduler (delete all jobs and triggers)
  - **Response**: {"0":["response"],"1":t/f} // t/f = [“true”, “false”]
- {“**id**”:“**pauseAll**”} // pause all jobs and triggers
  - **Response**: {"0":["response"],"1":t/f} // t/f = [“true”, “false”]
- {“**id**”:“**resumeAll**”} // resume all jobs and triggers
  - **Response**: {"0":["response"],"1":t/f} // t/f = [“true”, “false”]
- {“**id**”:“**truncateCatalinaLog**”} // truncate catalina log
  - **Response**: {"0":["response"],"1":t/f} // t/f = [“true”, “false”]

11. Configuration Tester

Come si può vedere nel derivable D2.8.2, all'interno del componente Smart Cloud, oltre ad avere lo Smart Cloud Engine, si ha il Configuration Tester e in questo capitolo andremo a descrivere come è possibile usare tale strumento.

11.1 Home

Quando si accede all'applicazione web che realizza il componente Configuration Tester, si ha davanti una pagina bianca con una barra di navigazione sulla quale si può scegliere quale operazione eseguire fra quelle presenti (Figura 11.1).

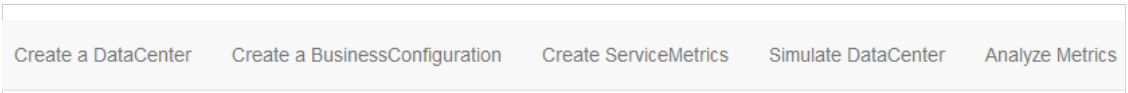


Figura 10.1: Operazioni che è possibile eseguire con il Configuration Tester

11.2 Create a DataCenter

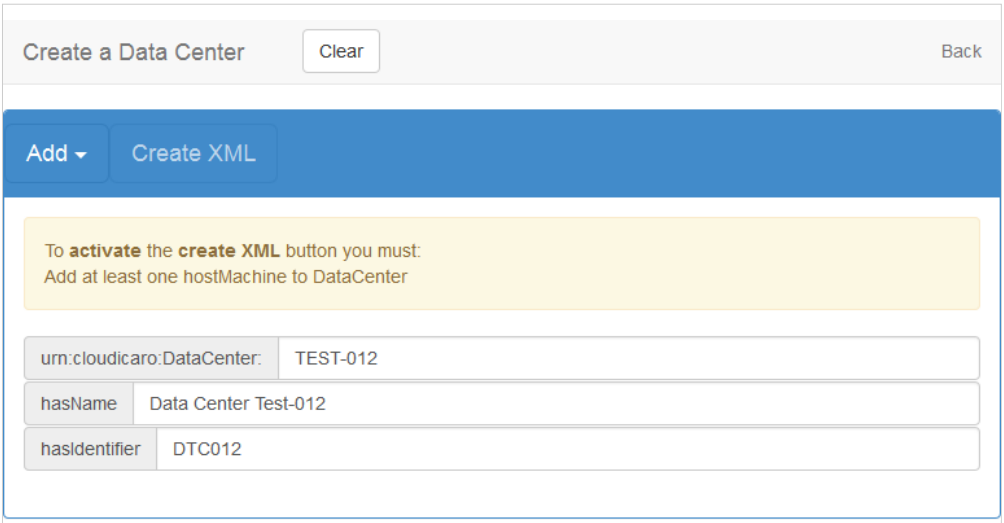


Figura 11.2: Pagina principale per la creazione di un dataCenter

La parte principale di ogni piattaforma cloud è il data center ed è necessario, quindi, avere a disposizione uno strumento che consenta di inserire all'interno della KB un nuovo dataCenter da poter analizzare. Questa operazione può essere fatta utilizzando l'operazione "Create a DataCenter" presente nella barra iniziale del Configuration Tester. Una volta cliccato sul link dell'operazione si accede alla pagina mostrata in Figura 11.2.

Attraverso il form di Figura 11.2 è possibile inserire le informazioni proprie del dataCenter:

- **urn:cloudIcaro:DataCenter:** la KB è realizzata attraverso un'ontologia e in questo tipo di database ogni entità è identificata univocamente attraverso un URI. Per quanto riguarda i dataCenter la parte iniziale dell'URI è sempre la stessa, ma la parte finale è variabile e può essere scelta dall'utente, che sta creando il dataCenter, inserendola in questo campo
- **hasName:** in questo campo è possibile inserire il nome che si vuole assegnare al dataCenter che si sta creando

- **hasIdentifier:** è possibile anche associare un identificativo al dataCenter e si può fare utilizzando questo campo

Come si può vedere nella Figura 11.2 queste informazioni non sono sufficienti per la creazione di un dataCenter: l'alert di colore giallo indica quali sono le altre entità che devono necessariamente essere inserite affinché sia possibile creare il file XML rappresentante il dataCenter appena creato. Cliccando sul pulsante "Add" è possibile aggiungere ulteriori entità (Figura 11.3) al dataCenter, sia che queste siano necessarie, sia che queste siano opzionali, alla creazione del file XML.

Il pulsante "Clear", visibile in alto sulla barra di navigazione nella Figura 11.3, consente di resettare tutti i form riempiti: una volta premuto non sarà possibile recuperare i dati inseriti.

Il link "Back", visibile in alto a destra sulla barra di navigazione nella Figura 11.3, consente di tornare alla Home per poter scegliere un'altra operazione da eseguire con l'applicazione.

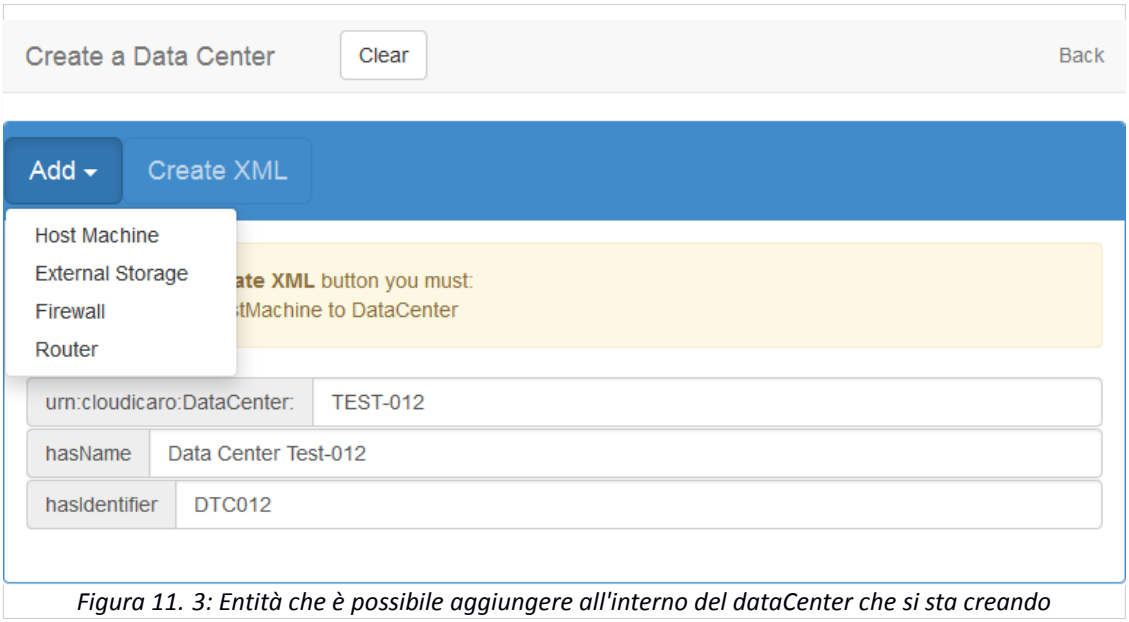


Figura 11. 3: Entità che è possibile aggiungere all'interno del dataCenter che si sta creando

Iniziamo a vedere come si possono aggiungere le entità al dataCenter che si sta creando, partendo dall'entità hostMachine che è necessaria per la creazione del file XML.

11.2.1 Host Machine

Per velocizzare l'inserimento delle informazioni, il form per la creazione delle entità hostMachine (Figura 11.4), consente di eseguire l'inserimento di gruppi di hostMachine anziché l'inserimento di ogni singola hostMachine.

A differenza dei dati inseriti per il dataCenter, in questo caso non è necessario inserire l' URI delle hostMachine: essendo un inserimento multiplo l'URI viene generato automaticamente in base: al dataCenter dove si trovano le hostMachine, al numero del gruppo di hostMachine che viene creato e al numero di hostMachine presenti nel gruppo. Per esempio, se un gruppo di hostMachine viene generato con il form in Figura 11.4 all'interno di un dataCenter generato con le informazioni della Figura 11.3, saranno generati i seguenti URI:

urn:cloudicaro:HostMachine:Test-012_HM01

urn:cloudicaro:HostMachine:Test-012_HM02

.

.

urn:cloudicaro:HostMachine:Test-012_HM010

*G*li altri campi devono essere riempiti con le seguenti informazioni:

- **prefixName:** il prefisso del nome di tutte le hostMachine che verranno create da questo form. A tale prefisso, come fatto per l'URI, sarà aggiunto un numero finale identificativo della hostMachine all'interno del gruppo
- **prefixIdentifier:** il prefisso dell'identificativo di tutte le hostMachine che verranno create da questo form. A tale prefisso, come fatto per l'URI, sarà aggiunto un numero finale identificativo della hostMachine all'interno del gruppo
- **CPUType:** il tipo di CPU che si trova all'interno di ogni hostMachine del gruppo (p.es. Intel Xeon)
- **domain:** il dominio all'interno del quale si trovano le hostMachine. Al momento la KB accetta come dominio "DC01" (Figura 11.4)
- **username:** l'username di accesso alle hostMachine. Sarà inserito uguale per tutte le hostMachine del gruppo.
- **password:** la password di accesso alle hostMachine. Sarà inserita uguale per tutte le hostMachine del gruppo.
- **# Host:** il numero di host machine che verranno create con tutte le caratteristiche inserite negli altri campi
- **CPU:** il numero di CPU che ha al suo interno ogni hostMachine
- **CPUSpeed:** la velocità in Ghz che ha ogni CPU inserita nel campo precedente
- **memorySize:** la dimensione della memoria principale ('RAM') in GB presente all'interno di ogni hostMachine
- **capacity:** capacità generica di ogni hostMachine
- **operatingSystem:** sistema operativo installato in ogni hostMachine (si possono scegliere i sistemi operativi previsti all'interno del dataCenter reale)
- **monitorState:** indica se il monitor stia o meno funzionando su ogni hostMachine. Si può scegliere fra i valori: Enabled che indica il monitor funzionante e Disabled che indica il monitor disabilitato

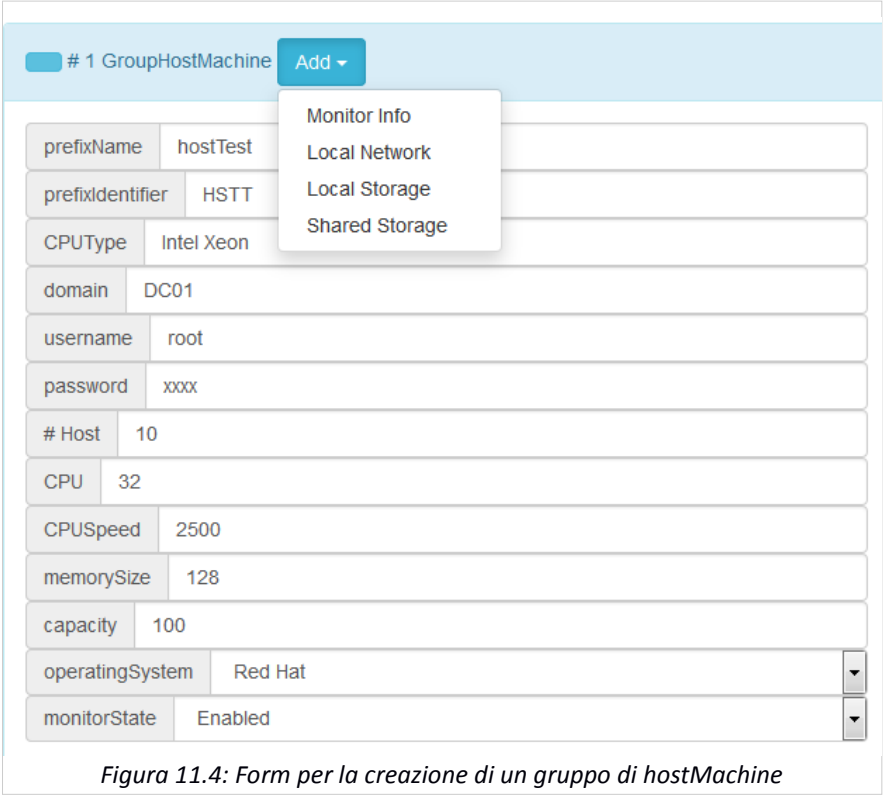


Figura 11.4: Form per la creazione di un gruppo di hostMachine

Come il dataCenter anche ogni hostMachine può avere delle entità associate. Vediamo in dettaglio quali sono le entità che possono essere aggiunte.

11.2.1.1 Monitor Info

Questa entità rappresenta informazioni utili per l'SM che deve monitorare le metriche delle host machine reali presenti nel dataCenter. Con la creazione del gruppo di hostMachine ad ogni hostMachine del gruppo sarà associata una propria entità monitorInfo con gli stessi dati delle altre.

Il form per l'inserimento di un'entità monitorInfo è quello che si può vedere in Figura 11.5 e le informazioni che devono essere inserite sono le seguenti:

- **metricName:** nome della metrica che l' SM deve tenere sotto controllo
- **arguments:** argomenti che possono risultare utili al controllo effettuato dal SM
- **warningValue:** valore che può innescare della azioni volte a prevenire problemi se raggiunto dalla metrica
- **criticalValue:** valore che può innescare della azioni volte a prevenire problemi se raggiunto dalla metrica
- **maxCheckAttemps:** numero di tentativi che devono essere effettuati sul valore della metrica prima di intraprendere azioni correttive

- **monitorState:** indica se il monitor stia o meno funzionando su tale monitor info. Si possono scegliere due valori: Enabled che indica il monitor funzionante e Disabled che indica il monitor disabilitato

- **checkMode:** indica in che modalità debba essere effettuato il controllo. Si possono scegliere due valori: Passive e Active

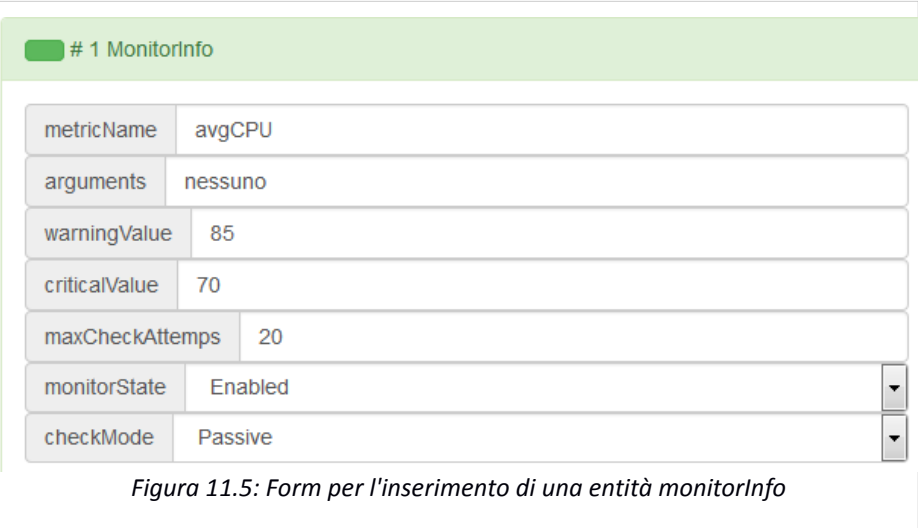


Figura 11.5: Form per l'inserimento di una entità monitorInfo

11.2.1.2 Local Network

Ogni host machine può avere uno o più networkAdapter: variano in base al numero di localNetwork alle quali si connette. Considerato che si sta generando un gruppo di hostMachine e non una hostMachine singola, si è ritenuto opportuno non inserire lo stesso indirizzo IP per ogni hostMachine: si inseriscono i dati sulle localNetwork alle quali accede il gruppo di hostMachine e l'applicazione:

- genererà in automatico gli indirizzi IP da associare ai networkAdapter delle hostMachine

- controllerà che non esistano due localNetwork inserite con lo stesso URI e dati diversi

- controllerà che la disponibilità di indirizzi IP di ogni localNetwork riesca a soddisfare la richiesta di indirizzi IP in base al numero di entità che devono avere un networkAdapter per accedere ad una determinata localNetwork.

I dati di ogni localNetwork possono essere inseriti attraverso il form di Figura 11.6 inserendo i seguenti dati:

- **networkAddress:** l'indirizzo della localNetwork che si sta creando

- **subNetMask:** la subnet mask della localNetwork che si sta creando. Attraverso tale valore si può sapere quanti indirizzi IP è possibile generare all'interno della localNetwork che si sta inserendo.

- **urn:cloudIcaro:LocalNetwork:** come già visto per il dataCenter anche in questo caso si deve inserire la parte variabile dell'URI relativo alla localNetwork. Tale parte variabile deve essere univoca rispetto a tutte le altre localNetwork
- **name:** il nome che si vuole associare alla localNetwork
- **identifier:** l'identificativo che si vuole associare alla localNetwork

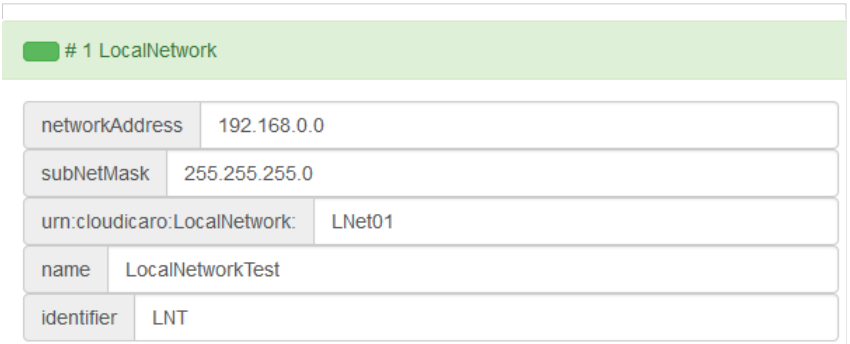


Figura 10.6: Form per l'inserimento di una entità local network

11.2.1.3 Local Storage

Questa entità si comporta esattamente come l'entità monitorInfo: ogni entità creata sarà inserita all'interno di ogni host machine del gruppo che si sta generando. Ogni entità localStorage, però, a differenza di monitorInfo avrà un proprio URI che viene calcolato in automatico dall'applicazione, tenendo conto del dataCenter e dell'hostMachine all'interno dei quali si trova e dal numero di localStorage creati all'interno di un gruppo di hostMachine:

urn:cloudicaro:LocalStorage:Test-012_HM01_LS0

Le informazioni che devono essere inserite nel form di Figura 11.7 sono:

- **name:** il nome del localStorage
- **identifier: I**'identificativo del localStorage
- **diskSize:** la dimensione in GB del disco fisso contenuto nel localStorage

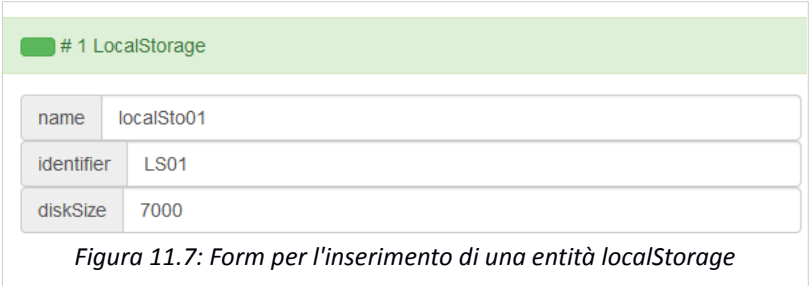


Figura 11.7: Form per l'inserimento di una entità localStorage

11.2.1.4 Shared Storage

A differenza delle altre entità associate ad una hostMachine, l'entità Shared Storage non viene definita completamente all'interno del form per la creazione di un gruppo i hostMachine. Si può

solamente inserire l'URI di uno sharedStorage da associare ad ogni hostMachine del gruppo che si sta generando, attraverso il campo visibile in Figura 11.8.

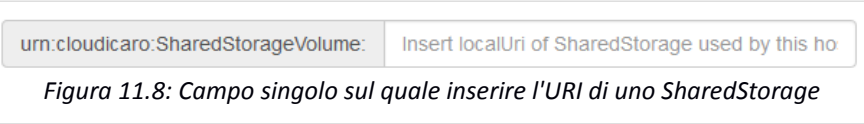


Figura 11.8: Campo singolo sul quale inserire l'URI di uno SharedStorage

In seguito, vedremo dove sia possibile definire tale entità e i controlli che sono eseguiti affinché l'URI di uno sharedStorage inserito nel form di creazione di un gruppo di hostMachine, sia uguale ad uno sharedStorage effettivamente creato.

11.2.2 External Storage, Firewall, Router

Dato che i form per l'inserimento delle entità externalStorage, firewall e router sono identici, a parte una voce aggiuntiva nel menù a tendina del form dell' externalStorage, relativa all'inserimento dell'entità sharedStorage accennata nel precedente paragrafo, si descrivono insieme e si possono vedere in Figura 11.8.

Anche per queste entità vale quanto affermato per le hostMachine: per velocizzare l'inserimento delle entità del dataCenter da testare, non si esegue un inserimento per ogni singola entità ma si creano gruppi di entità in modalità automatica. Naturalmente se si crea un gruppo con numerosità 1 si creerà una singola entità.

Le informazioni da inserire per la creazione di queste entità sono:

- **prefixName:** il nome che deve avere l'entità che si sta creando. Come per l'entità hostMachine il testo che si inserisce sarà solamente il prefisso del nome seguito da un numero che identifica il gruppo e l'entità stessa all'interno del gruppo
- **prefixIdentifier:** l'identificativo che deve avere l'entità che si sta creando. Come per l'entità hostMachine il testo che si inserisce sarà solamente il prefisso dell'identificativo seguito da un numero che identifica il gruppo e l'entità stessa all'interno del gruppo
- **model:** il modello (p.es. Cisco xxxx, WesternDigital 12345) dell'entità che si sta creando
- **# Entità: i**l numero di entità che si desidera creare con le caratteristiche presenti negli altri campi
- **monitorState:** indica se il monitor stia o meno funzionando su ogni entità creata. Si può scegliere fra i valori: Enabled che indica il monitor funzionante e Disabled che indica il monitor disabilitato

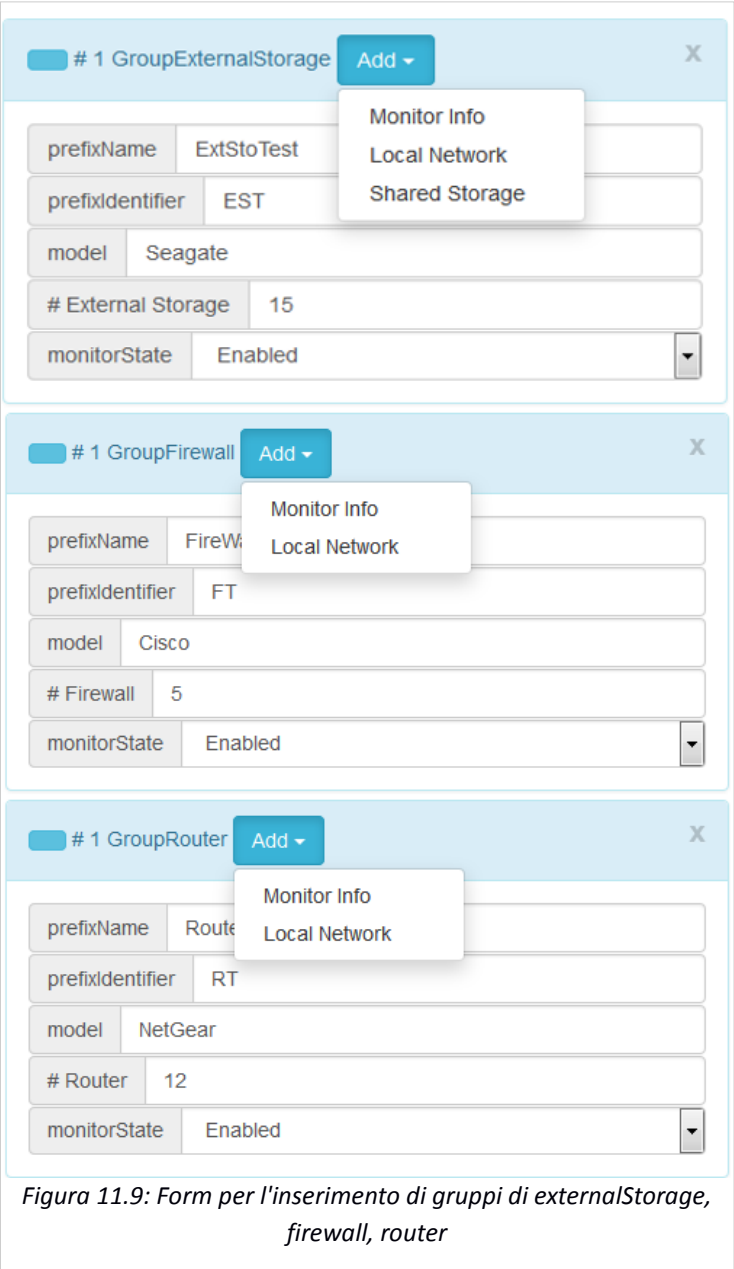


Figura 11.9: Form per l'inserimento di gruppi di externalStorage, firewall, router

Per le entità che si possono aggiungere come monitorInfo e localNetwork si rimanda alla descrizione effettuata per le stesse entità associate alle hostMachine nei paragrafi: "11.2.1.1 Monitor Info" e "11.2.1.2 Local Network".

11.2.2.1 Shared Storage

Come visto nel paragrafo "11.2.1.4 Shared Storage" è possibile associare ad ogni hostMachine uno sharedStorage, ma all'interno del form di creazione di un gruppo di hostMachine è possibile solamente inserire l'URI di tale sharedStorage.

All'interno del form di un externalStorage è possibile creare tale sharedStorage in quanto quest'ultimo deve necessariamente far parte di un externalStorage. Il form per la creazione di questa entità si può vedere nella Figura 11.10.

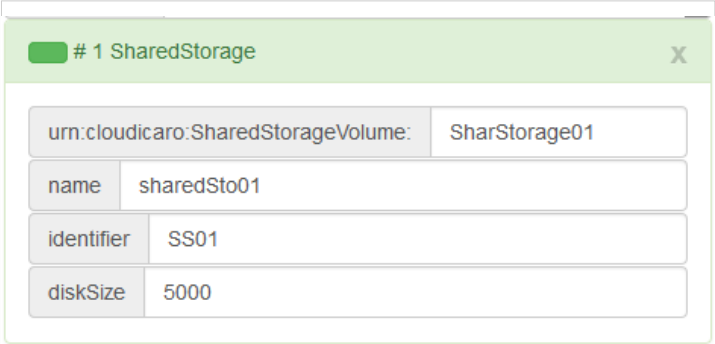


Figura 11.10: Form di inserimento di un'entità SharedStorage

Come si può vedere i dati da inserire sono gli stessi di un'entità localStorage, con l'aggiunta dell'URI necessario per essere associato ad una hostMachine.

Quindi i dati da inserire sono:

- **urn:cloudIcaro:SharedStorageVolume:** l'URI dello sharedStorage che può essere associato ad una hostMachine. Ogni externalStorage sarà associato alla singola entità generata da questo form.
- **name:** il nome del sharedStorage
- **identifier:** l'identificativo del sharedStorage
- **diskSize**: la dimensione in GB del disco fisso contenuto nel sharedStorage

11.2.3 Create XML

Se vengono inserite tutte le informazioni richieste dall'alert visibile in Figura 11.2: il pulsante "Create XML" si attiva (Figura 11.11), l'alert scompare ed è possibile eseguire la creazione dell'XML che rappresenta il dataCenter relativo alle informazioni inserite.

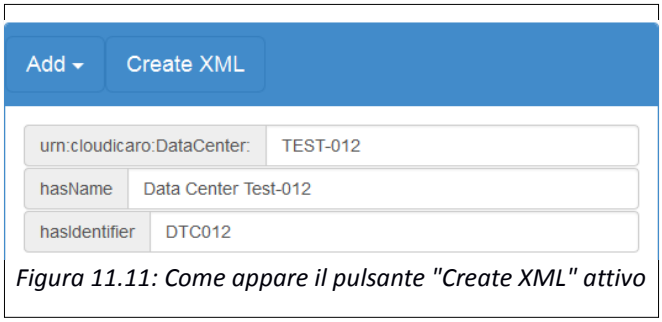


Figura 11.11: Come appare il pulsante "Create XML" attivo

Una volta premuto il pulsante "Create XML", se non ci sono problemi (vedi paragrafo successivo), si accede alla pagina visibile in Figura 11.12.

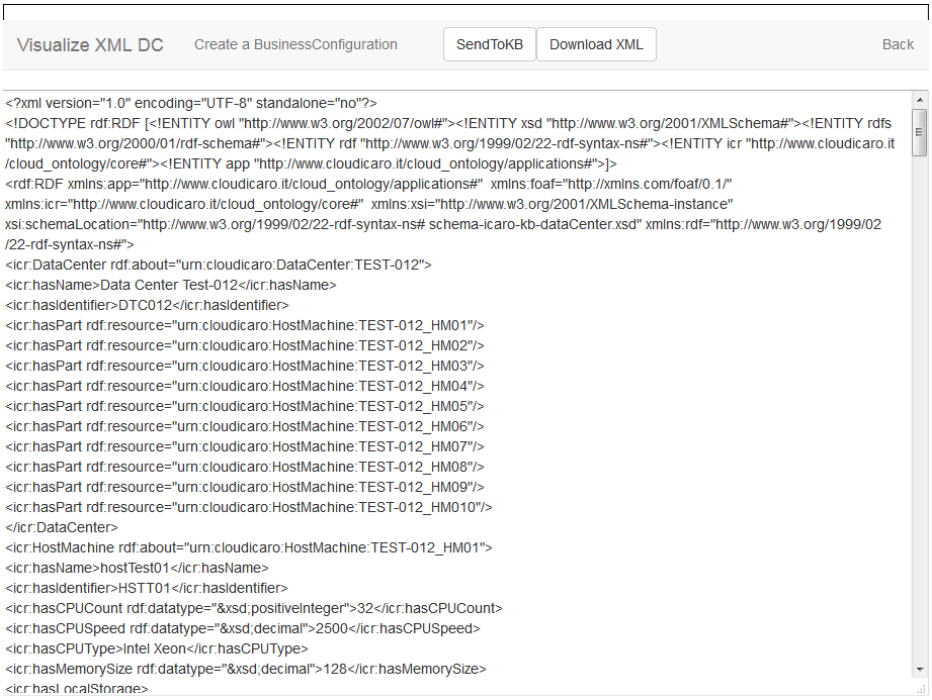


Figura 11.12: Pagina di creazione di un file XML relativo all'entità dataCenter

In questa pagina è visibile un'area di testo contenente il file XML generato: tale testo non è modificabile, ma è selezionabile per poter essere copiato in una qualsiasi altra applicazione che consente di manipolare testi.

Cliccando il pulsante "Download XML" il file XML viene automaticamente inviato al browser, che si sta utilizzando per accedere alla applicazione web, ed è possibile aprirlo direttamente o scaricarlo sul proprio computer. Il file scaricato è un semplice file di testo con estensione .xml e con filename

datacenterYYYY-MM-DDTHH_MM_SS

indicante quale sia l'entità che è contenuta all'interno dell'XML (in questo caso un dataCenter) e il timestamp di quando il file è stato inviato al browser.

Cliccando il pulsante "SendToKB" viene aperto il "modalPanel" in Figura 11.13 dove si può inserire l'indirizzo IP e la porta ai quali è possibile trovare l'API REST esposta dalla KB per effettuare l'operazione di inserimento di un dataCenter.

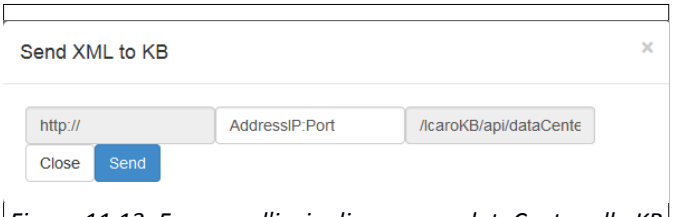


Figura 11.13: Form per l'invio di un nuovo dataCenter alla KB

Se l'invio alla KB del file XML del dataCenter ha esito positivo viene visualizzato un panel come quello di Figura 11.14.

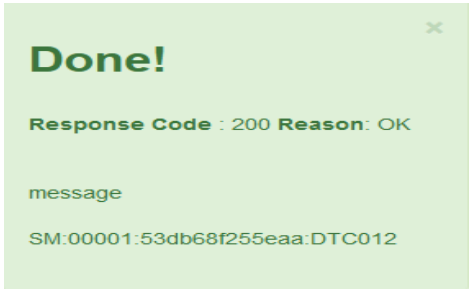


Figura 11.14: Messaggio che indica che l'invio del dataCenter è avvenuto con successo

Se l'invio alla KB del file XML del dataCenter ha esito negativo, perché all'indirizzo IP e alla porta inseriti non si trova un' istanza della KB, allora viene visualizzato un panel come quello di Figura 11.15.

Figura 11.15: Messaggio che indica che l'invio del dataCenter non è avvenuto

Il link "Back" porta alla pagina di creazione di un dataCenter dove vengono mostrati i dati precedentemente inseriti che possono essere modificati. Una volta apportate le modifiche desiderate verrà generato un nuovo XML e si verrà indirizzati nuovamente nella pagina di visualizzazione del file XML creato.

Il link "Create a BusinessConfiguration" serve per creare un'entità businessConfiguration sul dataCenter che si è appena generato. Per sapere come creare una businessConfiguration attraverso l'applicazione qui descritta vedere il capitolo successivo.

11.2.3.1 Eccezioni nella generazione del file XML

Come scritto nel paragrafo "11.2.1.2 Local Network" l'applicazione:

- controllerà che non esistano due localNetwork inserite con lo stesso URI e dati diversi
- controllerà che la disponibilità di indirizzi IP di ogni localNetwork riesca a soddisfare la richiesta di indirizzi IP in base al numero di entità che devono avere un networkAdapter per accedere ad una determinata localNetwork.

E come anticipato nel paragrafo "11.2.1.4 Shared Storage" l'applicazione:

- esegue i controlli affinché l'URI di uno sharedStorage inserito nel form di creazione di un gruppo di hostMachine, sia uguale ad uno sharedStorage effettivamente creato.

Se i controlli che esegue l'applicazione non vanno a buon fine vengono generati dei messaggi di errore per avvertire l'utente che i dati inseriti non sono corretti. Questi messaggi vengono generati in una nuova pagina dalla quale si può tornare a quella di inserimento dei dati e modificarli per evitare che si generino nuove eccezioni.

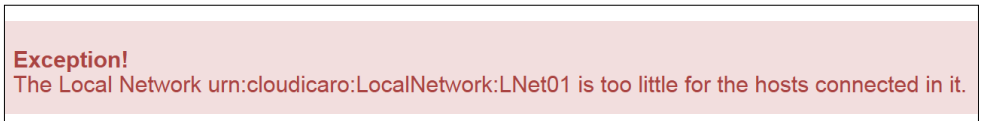


Figura 11.16: Eccezione generata se una localNetwork non ha abbastanza indirizzi IP

Se la localNetwork che viene inserita non ha a disposizione abbastanza indirizzi IP per tutte le entità che sono associate a tale rete, allora viene generata l'eccezione in Figura 11.16.

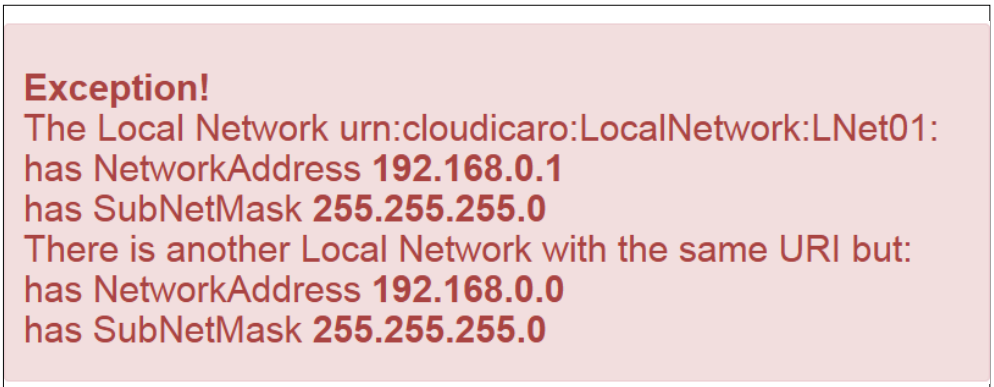


Figura 11.17: Eccezione generata se sono state inserite due LocalNetwork con lo stesso URI,

Se un utente vuole che due gruppi di hostMachine siano associati alla stessa localNetwork deve inserire in ognuno dei due gruppi la stessa localNetwork con gli stessi dati. Se durante questo inserimento le due localNetwork avessero uguali URI sarebbero considerate la stessa rete dall'applicazione e gli indirizzi IP e le subNetMask dovrebbero essere identici per non generare l'eccezione che si può vedere in Figura 11.17.

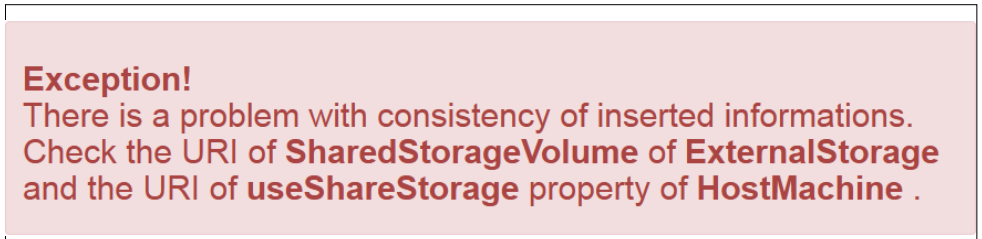


Figura 11.18: Eccezione generata se lo SharedStorage associato ad un'hostMachine non è stato creato in nessuna ExternalStorage

L'ultima eccezione che si può generare, come la precedente, controlla che i dati inseriti in due posizioni diverse nei form di inserimento del dataCenter, concordino fra di loro. Ad ogni gruppo di hostMachine si può associare uno SharedStorage, ma questo deve essere definito all'interno di un externalStorage. Quando uno sharedStorage associato ad un gruppo di hostMachine non coincide con uno sharedStorage creato all'interno di un externalStorage viene generata l'eccezione di Figura 11.18.

11.3 Create a BusinessConfiguration

Come visto nel paragrafo "11.2.3 Create XML" è possibile creare una businessConfiguration direttamente dalla pagina di visualizzazione di un dataCenter appena generato. In questo caso si accede direttamente al form che consente di creare la businessConfiguration. Invece, se clicchiamo sul link "Create a BusinessConfiguration", che si trova nella barra di navigazione della pagina principale dell'applicazione (Figura 11.1), si viene reindirizzati ad una pagina "intermedia" dove è possibile scegliere un dataCenter, presente in KB, all'interno del quale inserire una businessConfiguration.

11.3.1 Choose DataCenter

In questa pagina è possibile inserire indirizzoIP e porta (Figura 11.19) ai quali si trova una KB funzionante.

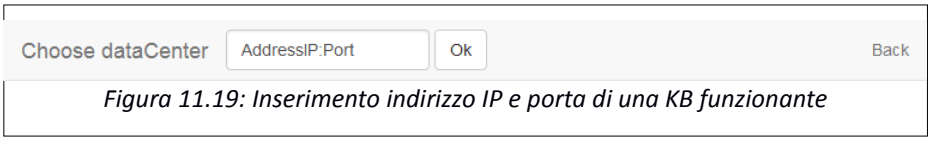


Figura 11.19: Inserimento indirizzo IP e porta di una KB funzionante

Se i dati inseriti risultano corretti viene mostrata una lista contenente i dataCenter presenti nella KB indicata (Figura 11.20).

Se questa KB non è quella desiderata è possibile modificare i dati inseriti attraverso il pulsante "Change".

Se si vuole conoscere hostMachine, virtualMachine e icaroService contenuti in uno dei dataCenter della lista, si deve premere il pulsante "Load", una volta selezionato il dataCenter di interesse.

Visualizzate le entità di un dataCenter, se si decide di creare una businessConfiguration all'interno di quest'ultimo, si deve cliccare il pulsante "Next" per accedere al form di inserimento di una nuova businessConfiguration.

N.B.: Quando si preme il pulsante Next viene considerato come dataCenter sul quale inserire la nuova businessConfiguration l'ultimo che è stato recuperato attraverso il pulsante "Load". Selezionare un dataCenter dalla lista senza premere il pulsante "Load" non lo recupera e viene considerato il dataCenter precedente, già caricato in memoria. Per questo motivo quando appare la lista di scelta dei dataCenter, il pulsante "Next non è visibile fino a che non viene caricato un dataCenter dalla KB.

11.3.2 Create Virtual Machine

Una volta scelto e recuperato dalla KB il dataCenter desiderato, si può creare la nuova businessConfiguration. Per farlo, si devono per prima cosa inserire le virtualMachine sulle quali dovranno appoggiarsi i servizi. Le virtualMachine possono essere inserite tramite il form visibile in Figura 11.21.

Figura 11.21: Form per l'inserimento di gruppi di virtualMachine

Il form ha molti campi uguali a quelli presenti nel form per la creazione di un gruppo di hostMachine, ma ce ne sono alcuni nuovi, vediamo quali sono le informazioni da inserire in tutti i campi:

- **externalIPAddress:** indirizzo IP che ha la virtualMachine verso l'esterno della rete e che può essere utilizzato per accedervi
- **monitoringIPAddress:** indirizzo IP che ha la virtualMachine e sul quale viene effettuato il monitoraggio

- **prefixName:** il prefisso del nome di tutte le virtualMachine che verranno create da questo form. A tale prefisso, come fatto per l'URI, sarà aggiunto un numero finale identificativo della virtualMachine all'interno del gruppo
- **prefixIdentifier:** il prefisso dell'identificativo di tutte le virtualMachine che verranno create da questo form. A tale prefisso, come fatto per l'URI, sarà aggiunto un numero finale identificativo della virtualMachine all'interno del gruppo
- **domain:** il dominio all'interno del quale si trovano le virtualMachine. Al momento la KB accetta come dominio "DC01" (Figura 11.21)
- **username:** l'username di accesso alle virtualMachine. Sarà inserito uguale per tutte le virtualMachine del gruppo.
- **password:** la password di accesso alle virtualMachine. Sarà inserita uguale per tutte le virtualMachine del gruppo.
- **# Virtual:** il numero di virtualMachine che verranno create con tutte le caratteristiche inserite negli altri campi
- **CPU:** il numero di CPU che ha al suo interno ogni virtualMachine
- **CPUReservation:** la velocità minima in Mhz che ha ogni CPU inserita nel campo precedente
- **CPULimit:** la velocità massima in Mhz che ha ogni CPU inserita nel campo precedente. Questo limite può essere impostata in base a quanta CPU viene utilizzata da tutte le virtualMachine presenti in una hostMachine
- **memorySize: i**ndica quanta memoria, di quella presente fisicamente nell'hostMachine, può essere utilizzata al massimo da ogni virtualMachine del gruppo che si sta creando
- **memoryReservation:** indica quanta memoria, di quella presente fisicamente nell'hostMachine, viene riservata per ogni virtualMachine del gruppo che si sta creando. Questa memoria è sempre disponibile per la virtualMachine e non dipende dalle altre virtualMachine che stanno eseguendo nella stessa hostMachine
- **memoryLimit: i**ndica quanta memoria, di quella presente fisicamente nell'hostMachine, può essere utilizzata effettivamente dalla virtualMachine. Questo valore dipende da quanta memoria stanno usando le altre virtualMachine che si trovano nella stessa hostMachine
- **operatingSystem:** sistema operativo installato in ogni virtualMachine (si possono scegliere i sistemi operativi previsti all'interno del dataCenter reale)
- **monitorState:** indica se il monitor stia o meno funzionando su ogni virtualMachine. Si può scegliere fra i valori: Enabled che indica il monitor funzionante e Disabled che indica il monitor disabilitato

- **arePartOf:** si deve scegliere in quale hostMachine risiederà il gruppo di virtualMachine che si sta creando. Si può scegliere una delle hostMachine presenti nel dataCenter che si è selezionato nella pagina precedente

- **isStoredOn:** consente di scegliere in quale storage della hostMachine selezionata al punto precedente devono essere salvate tutte le virtualMachine del gruppo che si sta creando. Si può scegliere sia un localStorage che un virtualStorage associato alla hostMachine scelta nel campo precedente

Anche in questo caso non è necessario inserire l' URI delle virtualMachine: essendo un inserimento multiplo l'URI viene generato automaticamente in base: al dataCenter e all'hostMachine dove si trovano le virtualMachine, al numero del gruppo di virtualMachine che viene creato e al numero di virtualMachine presenti nel gruppo. Per esempio, se un gruppo di virtualMachine viene generato con il form in Figura 11.21, saranno generati i seguenti URI:

urn:cloudicaro:HostMachine:Test-OneHost_HM01_VM01

urn:cloudicaro:HostMachine:Test-OneHost_HM01_VM02

.

.

urn:cloudicaro:HostMachine:Test-OneHost_HM01_VM05

Come per le altre entità presenti nella KB, anche alle virtualMachine è possibile associare altre sotto entità come: monitorInfo, localNetwork e virtualStorage (Figura 11.22).

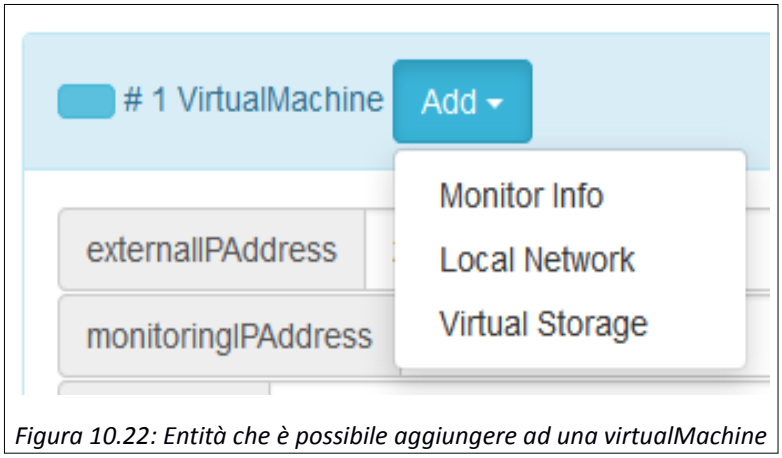


Figura 10.22: Entità che è possibile aggiungere ad una virtualMachine

Per vedere come aggiungere tali entità e per i dettagli delle informazioni da inserire si può consultare i paragrafi "11.2.1.1 Monitor Info", "11.2.1.2 Local Network" e "11.2.1.3 Virtual Storage", in quanto i form e le informazioni da inserire non variano rispetto alle entità che è possibile aggiungere ad un hostMachine. L'unica cosa che cambia è l'URI generato automaticamente per un virtualStorage rispetto a quello generato per un localStorage, che sarà del tipo:

urn:cloudicaro:VirtualStorage:Test-OneHost_HM01_VM02_VS0

Come il form per la creazione di dataCenter anche questo form mostra un alert di colore giallo che informa l'utente quali siano le entità da inserire per abilitare il pulsante "Next" che consente di andare alla pagina successiva per creare la nuova businessConfiguration.

Il pulsante "Clear" permette di resettare la pagina cancellando tutte le informazioni inserite e i form aggiunti al form iniziale.

Il link "Back" consente di tornare alla pagina di scelta di un dataCenter nel caso l'utente volesse cambiarlo con un altro presente nella stessa KB o volesse cambiare la KB scelta in precedenza.

Scelto il dataCenter sul quale creare la nuova businessConfiguration e create le virtualMachine sulle quali inserire gli IcaroService, viene spiegato come creare una businessConfiguration attraverso il form che si può vedere in Figura 11.23.

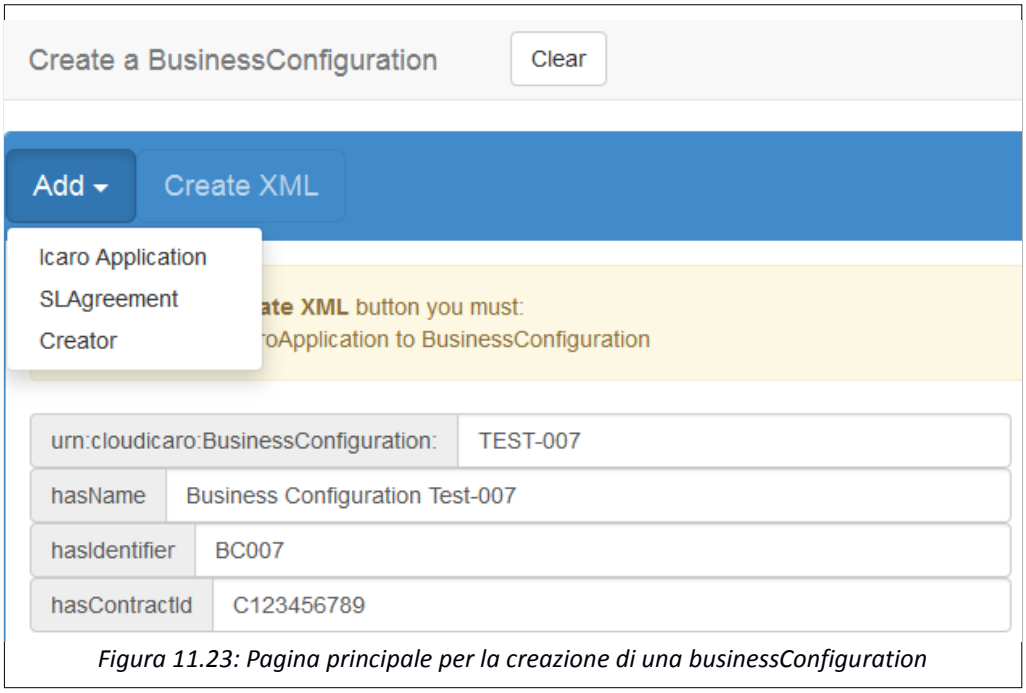


Figura 11.23: Pagina principale per la creazione di una businessConfiguration

Attraverso il form di Figura 11.23 è possibile inserire le informazioni proprie della businessConfiguration

- **urn:cloudIcaro:BusinessConfiguration:** la KB è realizzata attraverso un'ontologia e in questo tipo di database ogni entità è identificata univocamente attraverso un URI. Per quanto riguarda le businessConfiguration la parte iniziale dell'URI è sempre la stessa, ma la parte finale è variabile e può essere scelta dall'utente che sta creando il data center inserendola in questo campo

- **hasName:** in questo campo è possibile inserire il nome che si vuole assegnare alla businessConfiguration che si sta creando

- **hasIdentifier:** è possibile anche associare un identificativo alla businessConfiguration e si può fare utilizzando questo campo

- **hasContractId:** identifica l'id del contratto associato alla businessConfiguration per poter risalire all'utente che ha sottoscritto questa businessConfiguration

Come si può vedere nella Figura 11.23 queste informazioni non sono sufficienti per la creazione di una businessConfiguration: l'alert di colore giallo indica quali sono le altre entità che devono necessariamente essere inserite affinché sia possibile creare il file XML rappresentante la businessConfiguration appena creata. Cliccando sul pulsante "Add" è possibile aggiungere ulteriori

entità (Figura 11.23) alla businessConfiguration, sia che queste siano necessarie, sia che queste siano opzionali, alla creazione del file XML.

Il pulsante "Clear", visibile in alto sulla barra di navigazione nella Figura 11.23, consente di resettare tutti i form riempiti: una volta premuto non sarà possibile recuperare i dati inseriti.

Il link "Back" consente di tornare alla pagina precedente per poter modificare le virtualMachine create.

Iniziamo a vedere come si possono aggiungere le entità alla businessConfiguration che si sta creando, partendo dall'entità icaroApplication che è necessaria per la creazione del file XML.

11.3.3 IcaroApplication

Il form che consente di aggiungere un'entità icaroApplication alla businessConfiguration che si sta creando si può vedere in Figura 11.24.

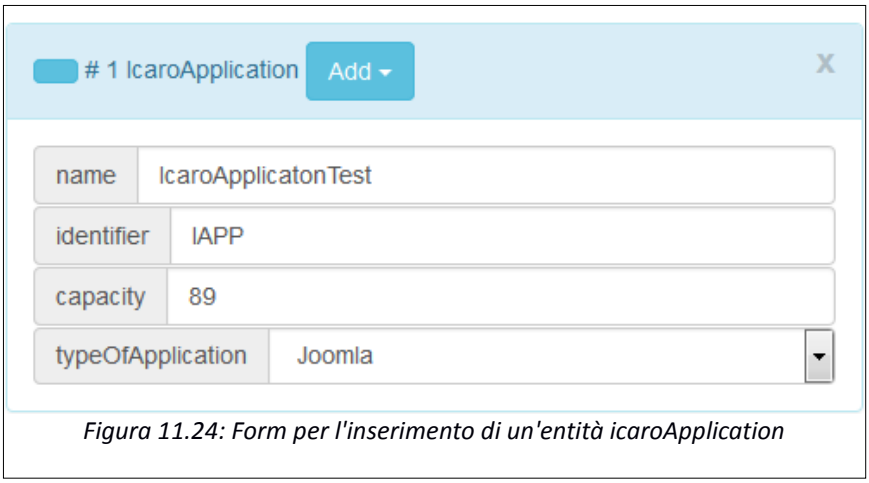


Figura 11.24: Form per l'inserimento di un'entità icaroApplication

I campi che devono essere inseriti sono i seguenti:

- **name:** in questo campo è possibile inserire il nome che si vuole assegnare alla icaroApplication che si sta inserendo
- **identifier:** è possibile anche associare un identificativo alla icaroApplication e si può fare utilizzando questo campo
- **capacity:** identifica una generica quantità contenuta dalla icaroApplication
- **typeOfApplication:** si può scegliere la tipologia di icaroApplication che deve essere inserita. Le applicazioni che sono presenti nella lista sono quelle che possono essere istanziate all'interno della piattaforma cloud icaro.

Creando un'icaroApplication, come succede per altre entità già viste, non è necessario inserire l'URI di tale entità, in quanto quest'ultima viene creata in base al tipo di applicazione che si sta inserendo, la businessConfiguration all'interno della quale si sta dichiarando e il numero di icaroApplication contenute dalla businessConfiguration. Nel caso in cui una icaroApplication venga inserita con le informazioni della Figura 11.24 e della Figura 11.23 avremo come URI generata:

urn:cloudicaro:Joomla:Test-007_0

Come è possibile vedere in Figura 11.25 è possibile aggiungere ulteriori entità ad una icaroApplication.

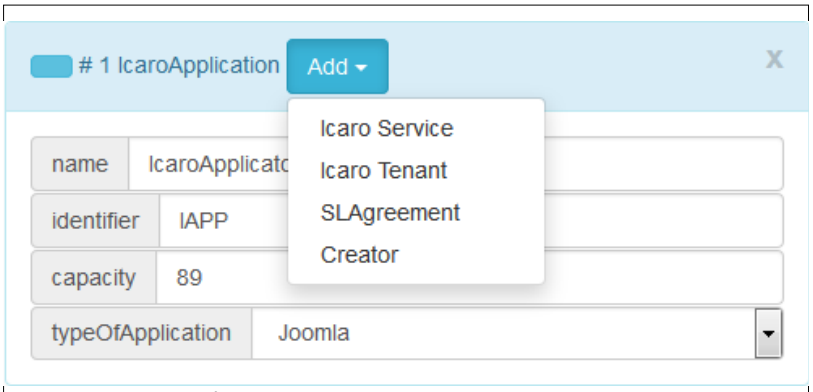


Figura 11.26: Entità che possono essere aggiunte ad una icaroApplication

N.B. Al momento, anche se presente nel menù a tendina, non è possibile aggiungere una entità icaroTenant: viene visualizzato il form per l'inserimento ma i dati non vengono raccolti per la creazione del file XML. Quindi analizzeremo le informazioni che devono essere inserite per le altre entità.

11.3.3.1 Icaro Service

Ogni icaroApplication ha bisogno di alcuni servizi per funzionare e se questi servizi non vengono inseriti, allora l'applicazione genererà dei messaggi appositi per avvertire l'utente di questa mancanza (si possono vedere nei seguenti paragrafi).

Il form con il quale si possono inserire informazioni relative alle entità icaroService si può vedere in Figura 11.27 e i campi da riempire sono i seguenti:

- **monitorIPAddress:** indirizzo IP dove si può trovare l'icaroService e sul quale viene effettuato il monitoraggio
- **name:** in questo campo è possibile inserire il nome che si vuole assegnare all'icaroService che si sta creando
- **identifier:** è possibile anche associare un identificativo alla icaroApplication e si può fare utilizzando questo campo
- **processName:** il nome del processo con il quale l'icaroService viene eseguito all'interno della virtualMachine
- **username:** l'username di accesso all'icaroService
- **password:** la password di accesso all'icaroService

- **typeOfService:** in questo campo si vincola la scelta ai tipi di servizi che consentono l'esecuzione di tutte le icaroApplication previste all'interno del cloud icaro. L'utente è responsabile di scegliere quei servizi che consentono l'esecuzione della icaroApplication che sta inserendo in un determinato momento.

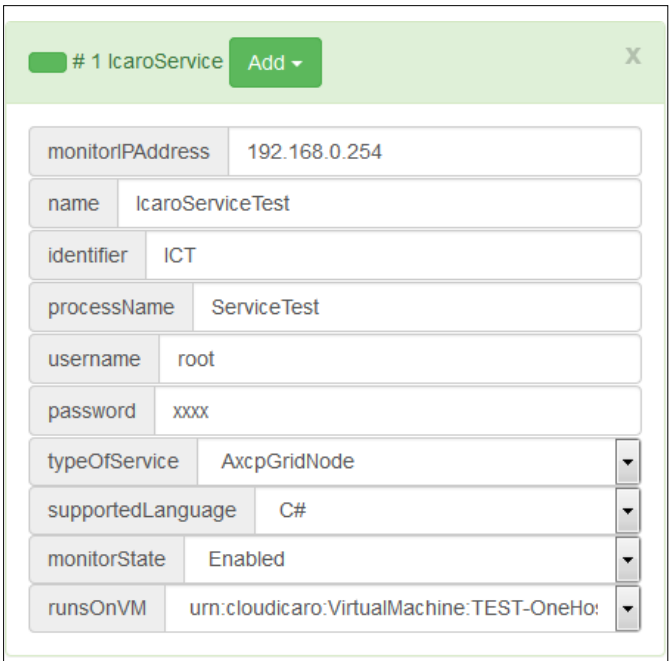


Figura 11.27: Form per l'inserimento dell'entità IcaroService

- **supportedLanguage:** è possibile scegliere quale linguaggio supporta l'icaroService che si sta creando. La scelta è vincolata ai linguaggi che si decide di supportare all'interno della piattaforma cloud icaro.

- **MonitorState:** indica se il monitor stia o meno funzionando sull'icaroService che si sta creando. Si può scegliere fra i valori: Enabled che indica il monitor funzionante e Disabled che indica il monitor disabilitato

- **runsOnVM:** si deve indicare in quale virtualMachine si vuole che sia reso funzionante l'icaroService che si sta generando. La scelta è vincolata alle virtualMachine inserite al passo precedente.

Come si può vedere dalla Figura 11.28 è possibile aggiungere ulteriori entità ad un icaroService.

Infatti, è possibile aggiungere un'entità monitorInfo, un'entità TCP Port e un'entità UDP Port. Per quanto riguarda l'entità monitorInfo e le informazioni che devono essere inserite si può vedere il paragrafo "11.2.1.1 Monitor Info".

Le altre due entità contengono semplicemente un campo numerico dove si può indicare il numero della porta TCP e/o UDP alla quale è possibile trovare l'icaroService.

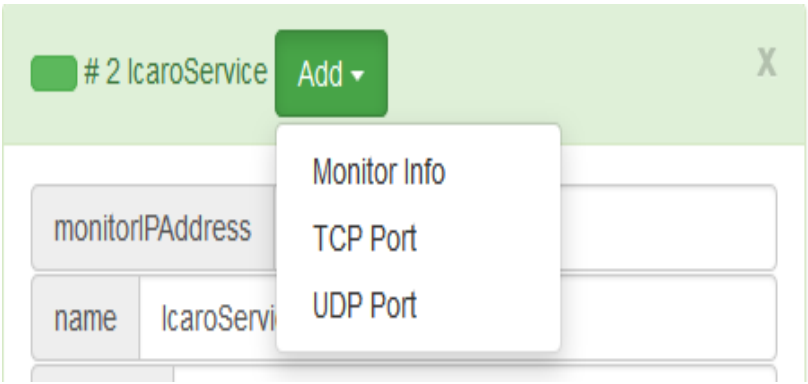


Figura 11.28: Entità che è possibile aggiungere ad un Icaro Service

11.3.3.2 SLAgreement, SLObjective, SLAction e SLMetric

Come previsto dalla KB ad ogni icaroApplication può essere associato uno SLAgreement e questa entità può essere inserita tramite il form presente in Figura 11.29.

Le informazioni da inserire all'interno del form sono:

- **startTime:** data che indica da quando lo SLAgreement inizia a valere
- **endTime:** data che indica fino a quando lo SLAgreement vale

Lo SLAgreement è univoco per ogni icaroApplication e non ne può essere creato più di uno.

Però possono essere aggiunti a tale entità un numero indefinito di SLObjective, a loro volta composti da SLAction, che rappresentano le azioni che devono essere intraprese dal sistema se non vengono rispettate le clausole delle SLAMetric (Figura 11.29).

Come si può vedere (Figura 11.29), l'entità SLObjective non ha campi propri ma è un semplice "contenitore" per le entità SLAction e SLMetric.

Le informazioni che possono essere inserite in un'entità SLAction sono:

- **name:** il nome dell'azione che viene eseguita se i vincoli imposti da SLMetric non vengono rispettati
- **callURI: se** l'azione che si deve eseguire è possibile chiamarla attraverso una URI, in questo campo si deve inserire tale URI

Le informazioni che si possono inserire in un'entità SLMetric sono:

- **name:** il nome della metrica che viene memorizzata all'interno della KB ed è oggetto del controllo per rispettare questo SLAgreement
- **unit:** l'unità di misura che viene associata al seguente campo value

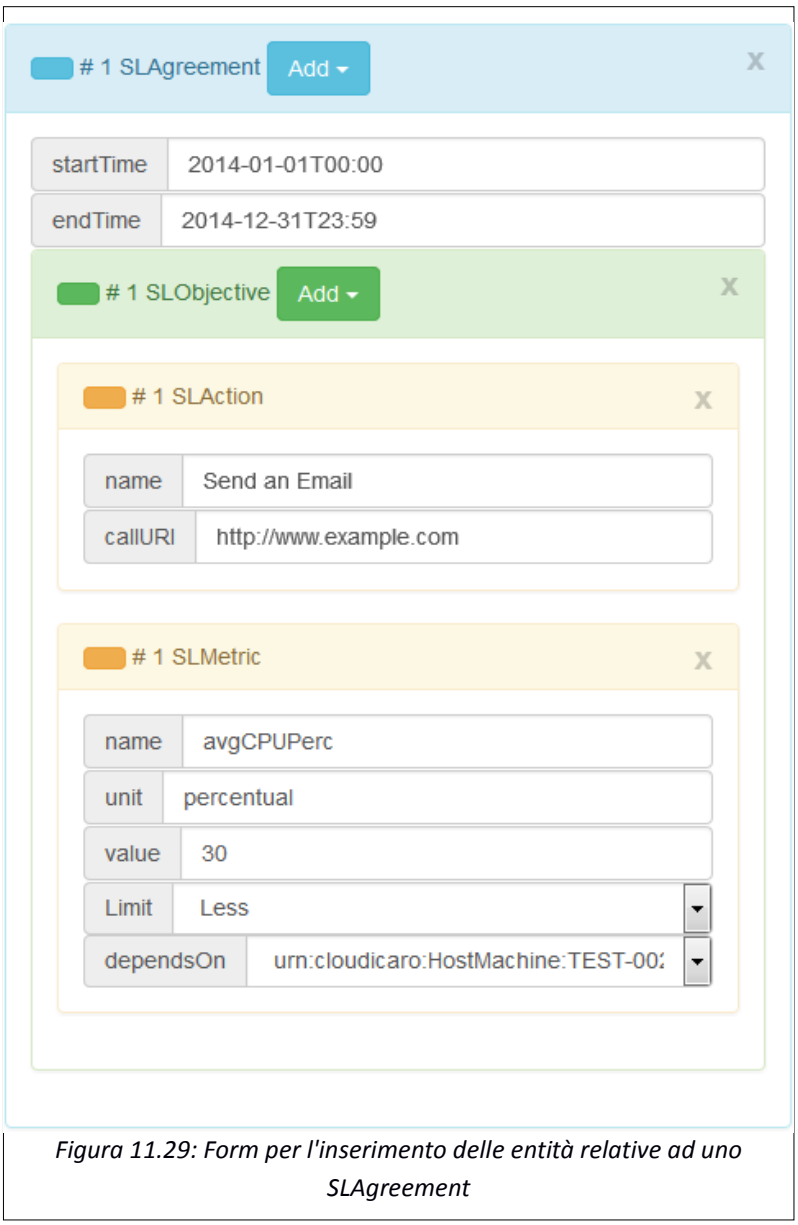


Figura 11.29: Form per l'inserimento delle entità relative ad uno SLAgreement

- **value:** il valore che deve essere rispettato in base al limit inserito in seguito
- **limit:** indica la condizione che devono rispettare le metriche memorizzate all'interno della KB rispetto al valore inserito precedentemente (campo "value"). I valori che possono essere scelti sono: Greater, Less o Equal che corrispondono ai simboli >, < e =.
- **dependsOn:** naturalmente la stessa metrica può essere misurata su più risorse contenute nella piattaforma cloud e in questo campo si deve inserire quale sia la risorsa di interesse per rispettare questa clausola dell'SLAgreement. Al momento è possibile scegliere fra le VirtualMachine e le HostMachine come risorse da monitorare.

11.3.3.3 Creator

Una icaroApplication può essere associata al suo creator. Per fare ciò, si devono inserire le informazioni seguenti, nel form che si può vedere in Figura 11.30:

- **urn:cloudIcaro:User:** si deve inserire l'URI che si desidera per il creatore che si sta inserendo
- **name:** nome del creatore che si sta inserendo nel form
- **email:** email del creatore che si sta inserendo nel form

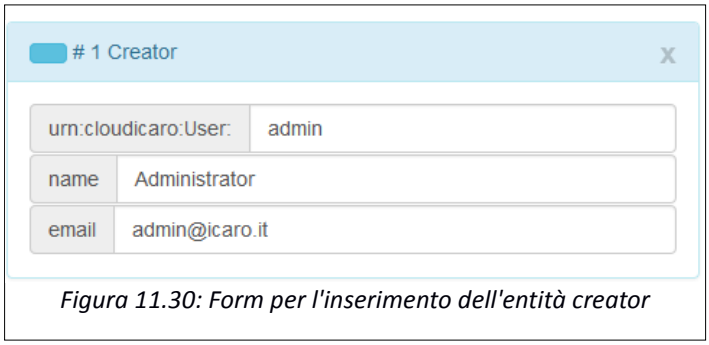


Figura 11.30: Form per l'inserimento dell'entità creator

Questa entità è simile alla entità localNetwork vista nei paragrafi precedenti: si inseriva una localNetwork per creare in automatico le entità networkAdapter da associare alle entità che si connettevano a quella rete e si controllava che non ci fossero due localNetwork con lo stesso URI e con diverse informazioni inserite.

Infatti, quando si inseriscono i dati in questo form la vera entità che viene generata è quella User che sarà associata alle altre entità per realizzare la proprietà hasCreator.

Inoltre, se l'utente inserito è creator di due icaroApplication allora si dovrà eseguire il controllo che non esistano due User con lo stesso URI e diversi name e email.

Nel paragrafo precedente si è descritto le informazioni e le entità necessarie alla creazione di un'entità icaroApplication all'interno di una businessConfiguration. Oltre a tale entità si possono aggiungere ad una businessConfiguration le entità SLAgreement e Creator.

Siccome tali entità hanno lo stesso form delle entità che si possono aggiungere ad una icaroApplication si consiglia di vedere i paragrafi precedenti per sapere come inserire tali informazioni all'interno dei form.

11.3.4 Create XML

Per quanto riguarda la creazione del file XML relativo alla businessConfiguration inserita e alla pagina alla quale si accede una volta creato tale file si può vedere il paragrafo "11.2.3 Create XML" relativo alla creazione del file XML di un dataCenter.

Le uniche modifiche riguardano il fatto che nella barra di navigazione non è presente il link per la creazione di una nuova businessConfiguration, dato che se ne è appena creata una ed è differente la API REST che viene invocata per l'inserimento della businessConfiguration, appena generata, dentro la KB.

11.3.4.1 Eccezioni nella generazione del file XML

Nel paragrafo precedente si è affermato che l'applicazione dovrà effettuare dei controlli nel caso in cui siano inseriti due creatori (p. es. uno della businessConfiguration e uno di una icaroApplication) per:

- evitare che esistano due User con lo stesso URI e diversi name e email

Nel caso in cui il controllo abbia esito negativo, cioè esistono due entità User/Creator con lo stesso URI ma differenti name e/o email, viene mostrato all'utente il messaggio di Figura 11.31.

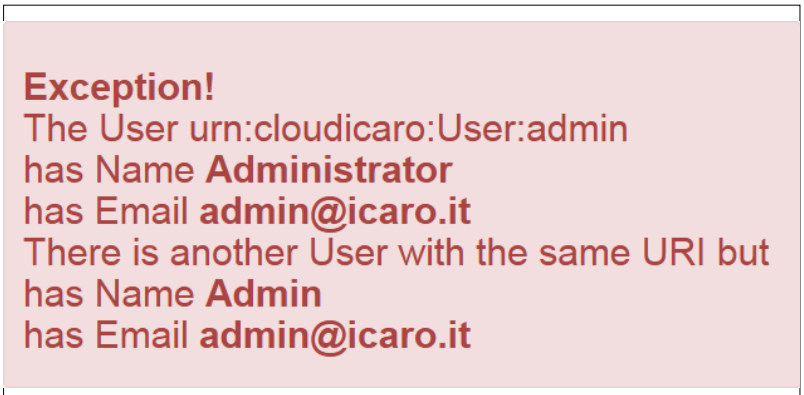


Figura 11.31: Eccezione generata se sono stati inseriti due User con lo stesso URI, ma con gli altri dati diversi

Letto il messaggio è possibile tornare alla pagina precedente per effettuare le modifiche necessarie ad evitare di ricevere nuovamente il messaggio che avverte dell'errore fatto in fase di inserimento.

11.3.4.2 Eccezioni nell'inserimento del file XML nella KB

Come detto all'inizio del paragrafo "11.3.3.1 Icaro Service", ogni icaroApplication ha bisogno di alcuni servizi per funzionare e se questi servizi non vengono inseriti, allora l'applicazione genererà dei messaggi appositi per avvertire l'utente di questa mancanza.

Tali messaggi, a differenza delle eccezioni che vengono generate dall'applicazione, vengono generate durante l'inserimento del file all'interno della KB, in quanto riguardano regole inserite nella KB e non nello XSD-SCHEMA perla generazione dei file XML.

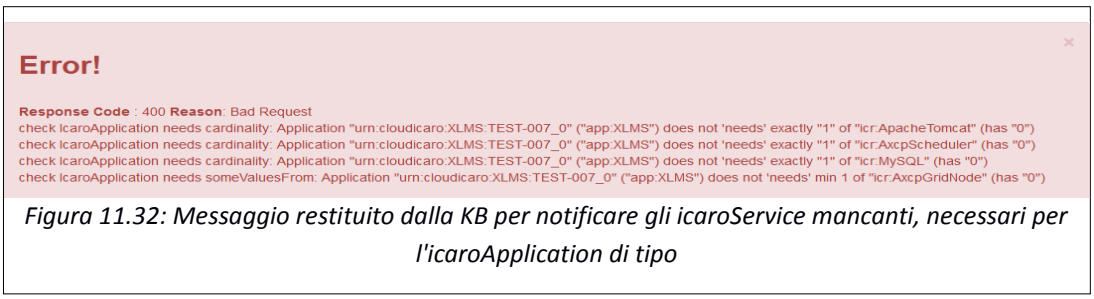


Figura 11.32: Messaggio restituito dalla KB per notificare gli icaroService mancanti, necessari per l'icaroApplication di tipo

Questo porta, per esempio, ad avere il messaggio di Figura 11.32 in risposta al tentativo di invio alla KB di un file XML di una businessConfiguration contenente un icaroApplication di tipo XLMS, della quale non sono stati dichiarati tutti i servizi di cui ha bisogno.

11.4 Create ServiceMetrics

Questa operazione consente di creare un numero predefinito di metriche da associare ad una determinata entità (p.es. una hostMachine o una virtualMachine), decidendo il periodo temporale nel quale devono essere distribuite, i valori che devono avere durante tale periodo e il valore finale che deve essere inserito nell'ultima metrica.

Questa operazione può essere utile se si vuole controllare la corretta rilevazione di problemi, per esempio relativi ad uno "sforamento" dei limiti imposti dallo SLA , attraverso delle metriche inserite in KB.

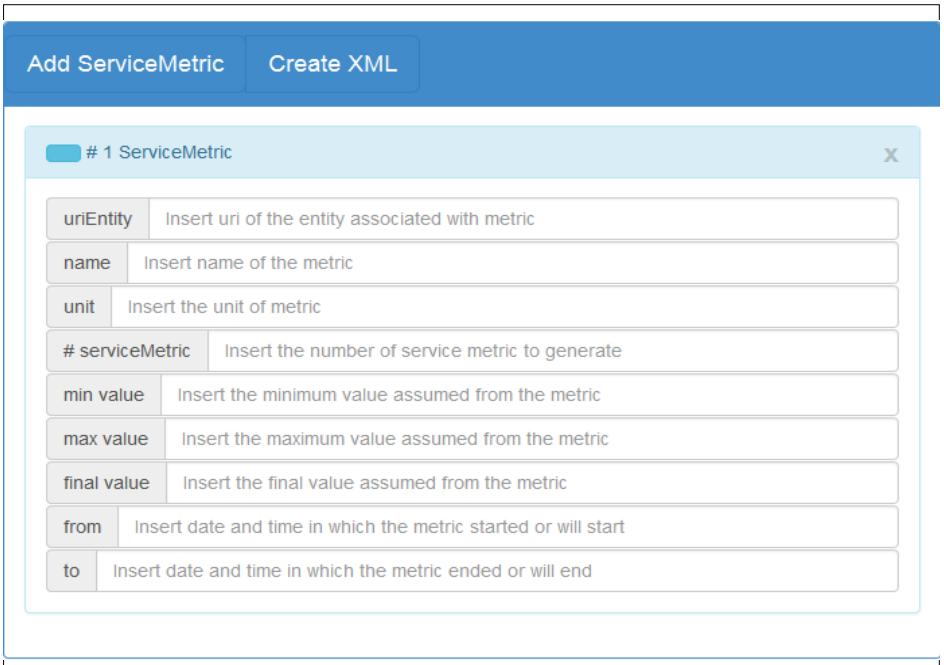


Figura 11.33: Form per l'inserimento delle entità Service Metrics

Per la creazione delle metriche il form che si deve riempire è quello di figura 11.33 e le informazioni necessarie sono le seguenti:

- **uriEntity:** uri dell'entità alla quale si devono riferire le metriche generate automaticamente
- **name:** il nome che si desidera dare a tutte le metriche che verranno generate
- **unit:** l'unità di misura che deve essere associata ai valori inseriti per la metrica
- **# serviceMetric:** il numero di metriche che devono essere generate
- **minValue:** il più piccolo valore che può assumere la metrica
- **maxValue:** il più grande valore che può assumete la metrica
- **finalValue:** il valore che deve assumere la metrica finale che verrà generata. Può anche essere un valore che non rientra fra i valori minValue e maxValue.
- **from:** data di inizio delle metriche.

- **to:** data di fine delle metriche

Il periodo di generazione delle metriche è un periodo fittizio e le metriche verranno generate tutte istantaneamente per essere inviate alla KB.

11.4.1 Create XML

Per quanto riguarda la creazione del file XML relativo alle serviceMetrics inserite e alla pagina alla quale si accede una volta creato tale file si può vedere il paragrafo "11.2.3 Create XML" relativo alla creazione del file XML di un dataCenter.

Le uniche modifiche riguardano il fatto che nella barra di navigazione non è presente il link per la creazione di una nuova businessConfiguration, dato che stiamo creando delle metriche, è differente la API REST che viene invocata per l'inserimento delle serviceMetric, appena generate, dentro la KB e il file XML generato contiene tutte le serviceMetric generate, anziché generare un singolo XML per ogni serviceMetrics.

11.5 Simulate DataCenter

Questa operazione consente di simulare (in modo approssimativo, per il momento) il funzionamento di una piattaforma cloud per:

- generare delle metriche in modalità automatica
- studiare il comportamento della piattaforma cloud nella fase di inserimento di nuove businessConfiguration, in particolare nell'inserimento di nuove virtualMachine e icaroService

La simulazione può essere effettuata in due modalità:

- **RealTime:** il tempo di simulazione è lo stesso del tempo che passa nella realtà. Attraverso questa simulazione si cercherà di studiare il comportamento del cloud durante l'inserimento di nuove businessConfiguration
- **Faster:** in questo caso non si può parlare di un vero e proprio tempo di simulazione. Si scelgono la data iniziale e la data finale della simulazione e in quel lasso di tempo vengono generate delle metriche in modalità automatica per tutte le entità di un determinato dataCenter, scelto in precedenza

La scelta fra le due simulazioni può essere fatta attraverso gli appositi pulsanti visibili in Figura 11.34.

Una volta scelta una delle due tipologie di simulazione si viene reindirizzati sulla pagina di scelta del dataCenter, in quanto dobbiamo scegliere quale dataCenter simulare. Per avere informazioni su tale pagina vedere il paragrafo "11.3.1 Choose DataCenter".

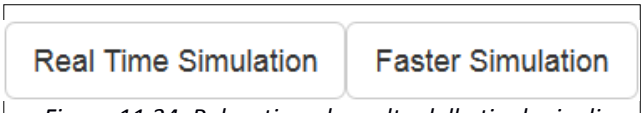


Figura 11.34: Pulsanti per la scelta della tipologia di simulazione da effettuare

11.5.1 Real Time Simulation

La pagina che consente di simulare un dataCenter si può vedere in Figura 11.35.

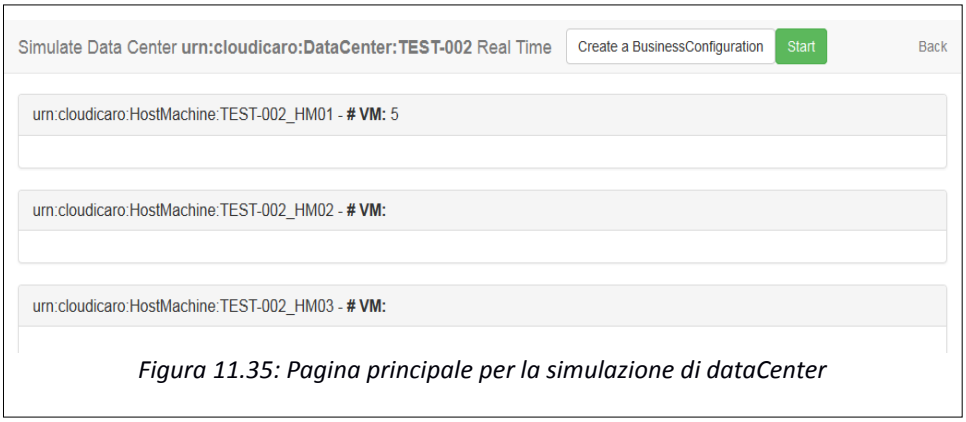


Figura 11.35: Pagina principale per la simulazione di dataCenter

All'interno della pagina possiamo vedere dei "panel" che saranno riempiti con dei grafici una volta iniziata la simulazione. Ad ogni "panel" è associata una hostMachine e nell'header viene indicato quante virtualMachine sono presenti all'interno dell'hostMachine relativa a quel "panel".

Se si vuole iniziare la simulazione si può premere il pulsante "Start".

Una volta iniziata la simulazione, nella pagina appaiono i grafici relativi ai dati della simulazione e i pulsanti della barra di navigazione vengono modificati come si può vedere in Figura 11.36.

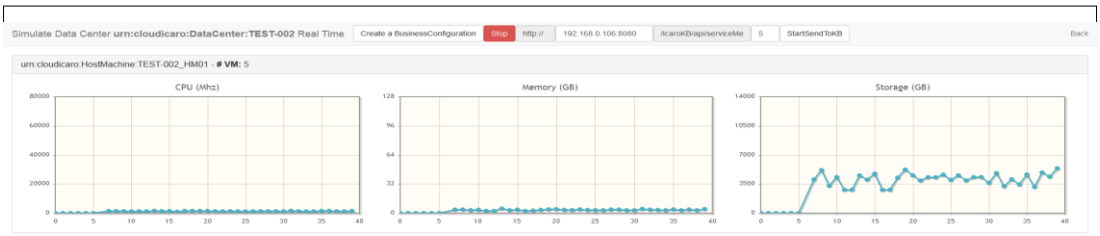


Figura 11.36: Simulazione di un dataCenter con i grafici relativi a CPU, Memory e Storage

Nei tre grafici visibili possiamo vedere l'andamento dei dati simulati di CPU, Memory e Storage relativi ad una singola hostMachine. I grafici vengono aggiornati ogni 5 secondi con il "refresh" della pagina.

Il pulsante "Stop" permette di interrompere istantaneamente la simulazione.

Il form che segue il pulsante "Stop" consente di scegliere una KB (inserendo indirizzo IP e porta di una KB funzionante) e il tempo con il quale i dati simulati vengono campionati e inviati per essere inseriti nella KB scelta. Il timeStamp che verrà inserito all'interno delle metriche inviate alla KB è riferito al momento in cui queste vengono campionate.

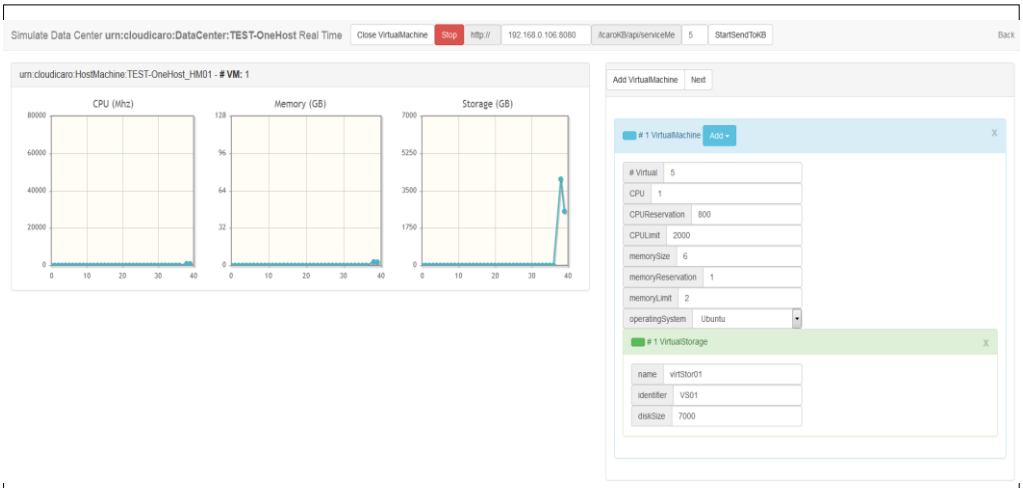


Figura 11.37: Come appare la pagina della simulazione quando si aggiungono virtualMachine

Figura 11.38: Come appare la pagina della simulazione quando si aggiungono businessConfiguration

Il pulsante "Create a BusinessConfiguration", se premuto, per il momento fa apparire un form per l'inserimento di virtualMachine e a seguire un form per l'inserimento di una businessConfiguration Figura 11.37 e Figura 11.38. Si devono implementare le funzioni che consentano di aggiungere queste entità dinamicamente al simulatore.

11.5.2 Faster Simulation

Se non si ha bisogno di effettuare una simulazione in real time, ma si devono simulare dei dati provenienti da un determinato dataCenter, si può utilizzare la Faster Simulation, la quale, a differenza della creazione di serviceMetric:

- genera le metriche per tutte le entità di un dataCenter anziché generarle per una sola entità
- genera le metriche con un intervallo di 5 minuti, anziché redistribuirle in modo uniforme nel periodo indicato
- non prevede limite inferiore o superiore ai valori che possono essere assunti dalla metrica e nemmeno un valore finale

L'interfaccia per la generazione automatica delle metriche si può vedere in Figura 11.39.

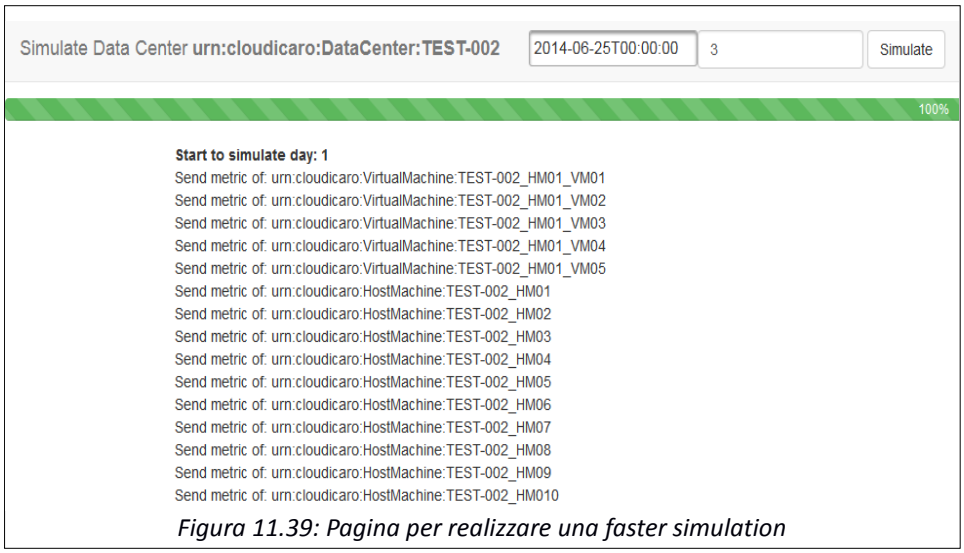


Figura 11.39: Pagina per realizzare una faster simulation

Si deve semplicemente inserire la data dalla quale deve partire la simulazione, inserire il numero di giorni che devono essere simulati e cliccare sul pulsante "simulate". Le metriche saranno create istantaneamente e inviate alla KB dalla quale è stato caricato il dataCenter che si sta simulando.

11.6 Analyze Metrics

Attraverso tale operazione è possibile analizzare le metriche che sono già presenti all'interno della KB. Come già spiegato nel paragrafo "11.3.1 Choose DataCenter" per prima cosa è necessario scegliere e caricare dalla KB il dataCenter che si vuole analizzare.

Una volta che si è caricato un dataCenter potremo vedere la pagina in Figura 11.40.

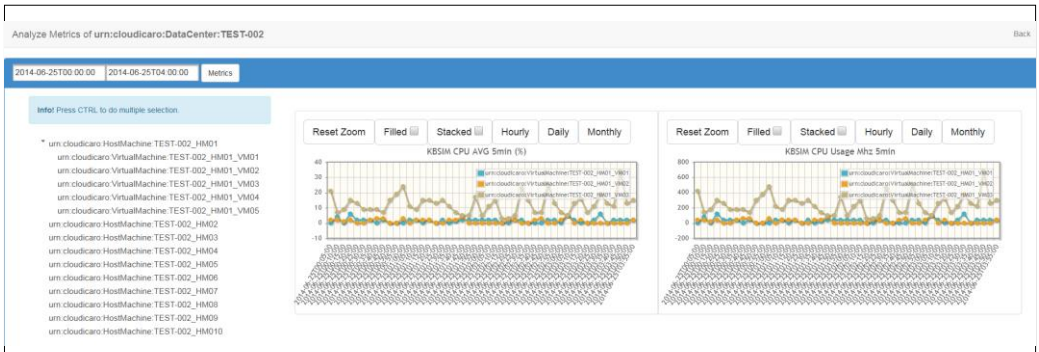


Figura 11.40 Pagina che consente di analizzare le metriche di entità contenute in un determinato dataCenter

Si può notare sulla sinistra l'elenco di hostMachine e virtualMachine contenute all'interno del dataCenter selezionato nella pagina precedente.

Su tale lista si possono selezionare più virtualMachine e/o più hostMachine delle quali analizzare le metriche. Una volta selezionate le entità si devono inserire data di inizio e data di fine dell'analisi che si vuole effettuare.

Premendo il pulsante "Metrics" vengono generati i grafici di Fiugra 11.40. Se due entità hanno una stessa metrica allora i dati verranno raggruppati all'interno di un singolo grafico e si avrà una curva

per ogni entità: in Figura 11.40 abbiamo 3 curve perché si sono selezionate 3 entità che hanno le stesse metriche.

Il pulsante "Reset Zoom" riporta il grafico alla sua risoluzione originaria dopo che si è effettuato uno zoom con il mouse.

Il checkBox "Filled" disegna l'area sottostante di ogni curva che in Figura 11.40 è trasparente.

Il chechBox "Stacked" somma i vari contributi di ogni curva e realizzando un grafico con i dati cumulati fra le tre curve.

Gli ultimi tre bottoni "Hourly", "Daily" e "Monthly" servono per scalare la risoluzione del grafico per avere una visione migliore dell'andamento dei dati.