

# Relazione del progetto di Big Data Architectures



**Matteo Marulli, Matteo Ghera<sup>1</sup>**

<sup>1</sup>CORSO DI Laurea Magistrale in Informatica: Curriculum Data Science

2994 Words

Student ID:7005652, 7006227

**Keywords:** Docker, Mesos, Marathon, Zabbix, MySQL, Elasticsearch, Logstash, Kibana, Elastic Machine Learning

15<sup>th</sup> February, 2021

## Abstract

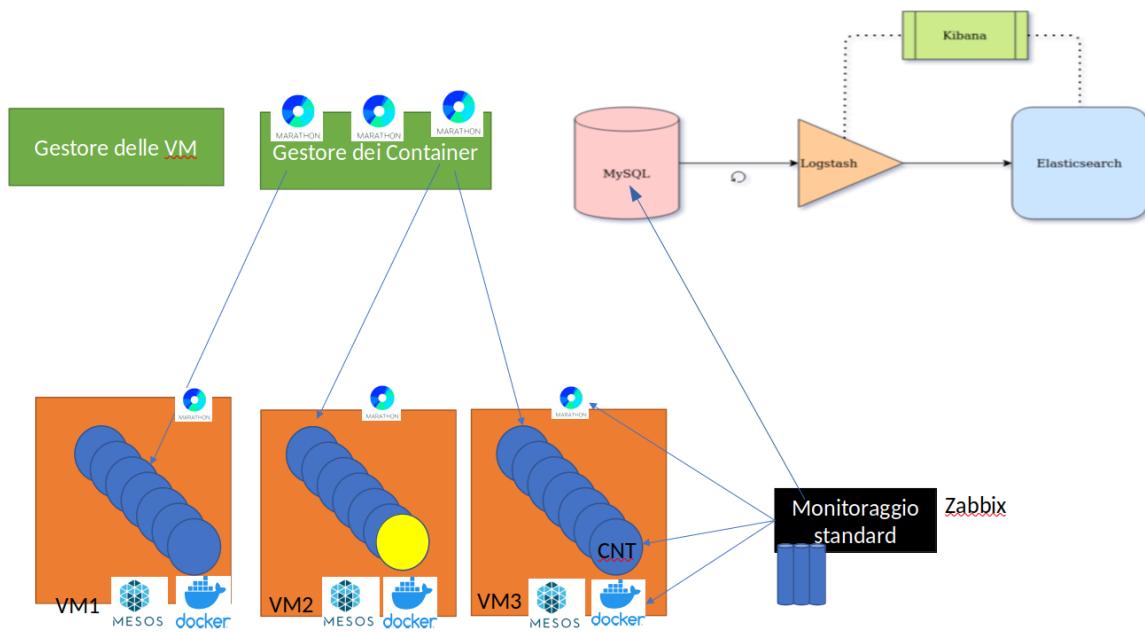
Supponiamo di avere a disposizione su un server diverse macchine virtuali che eseguono applicazioni all'interno di container Docker. Risulta fondamentale avere un sistema di monitoraggio che rilevi la quantità di risorse (CPU, Memoria RAM, Disco, Rete ed Errori) al fine di rilevare utilizzi eccessivi di risorse e possibili malfunzionamenti delle applicazioni in esecuzione. L'architettura per la gestione, il monitoraggio e la rilevazione di anomalie nel consumo di risorse da parte delle applicazioni da noi studiata si compone di diverse componenti:

- **Apache Mesos**, gestore di risorse per cluster di VM;
- **Apache Marathon**, servizio di orchestrazione di container Mesos/Docker;
- **Zabbix**, servizio di monitoraggio;
- **Kibana**, strumento di visualizzazione dei dati per la creazione di dashboard accattivanti. Kibana utilizza come database Elasticsearch a differenza di Zabbix che utilizza Mysql. Noi abbiamo utilizzato Logstash per trasferire i dati da Mysql a Elasticsearch.

Queste componenti sono interconnesse fra di loro e realizzano un flusso di dati che consente di monitorare il consumo di risorse da parte delle diverse applicazioni attraverso le dashboard di Kibana.

## 1 Modelli di anomaly detection sulla base del comportamento di container

L'obiettivo del progetto è realizzare un'applicazione basata su microservizi per il monitoraggio e il rilevamento di anomalie nei container Docker in esecuzione su un data center. Il data center è gestito attraverso Apache Mesos e Apache Marathon. Apache Mesos è un gestore di risorse per cluster di macchine host / VM che amministra il data center come se fosse un'unica entità. Apache Marathon è un servizio di orchestrazione per Mesos con il quale è possibile creare applicazioni che girano su container Docker. Per sviluppare il progetto abbiamo usato diversi strumenti che introduciamo via via per un'analisi più dettagliata. In Appendice 10 abbiamo riportato lo schema dei docker-compose e dei file di configurazione. I file completi si possono trovare all'indirizzo Github: <https://github.com/MattBlue92/progetto-Big-data-architectures-2021>



**Fig. 1.** Schema di progettazione

## 2 Configurazione di Apache Marathon e Apache Mesos

Inizialmente per Mesos e Marathon abbiamo usato l'immagine docker mesos-mini come consigliato dalla documentazione ufficiale di Mesos <http://mesos.apache.org/blog/mesos-mini/>

```
$ docker run --rm --privileged -p 5050:5050 -p 5051:5051
-p 8080:8080 mesos/mesos-mini
```

Il problema di questa soluzione è che usava l'opzione `--privileged` creando l'effetto "Docker in Docker", rompendo così il design di Docker. Inoltre il container aveva accesso al kernel del nostro sistema e ciò è fortemente sconsigliato per motivi di sicurezza. Da questo problema abbiamo deciso di scrivere un docker-compose in cui facciamo partire separatamente tutti i servizi necessari per il corretto funzionamento di Mesos e Marathon, ossia:

- Apache Zookeeper vers: 3.4.13
- Il master di mesos: mesosphere/mesos-master vers:1.7.1
- Lo slave di mesos: mesosphere/mesos-slave vers:1.7.1
- La UI di Marathon: mesosphere/marathon vers:1.8.667

## 3 Creazione di un'applicazione in Marathon

Per prendere confidenza con Marathon abbiamo deciso di creare un'applicazione Marathon che avvia un container Docker dell'immagine Nginx. Le Figure 2 3 4 mostrano le impostazioni che abbiamo scelto per l'applicazione di prova:

**Edit Application**

JSON Mode

<b>General</b>	<b>ID</b> nginx		
Docker Container	CPUs      Memory (MiB)      Disk Space (MiB)      Instances 0,3      128      200      1		
Ports	Command		
Environment Variables	<small>May be left blank if a container image is supplied</small>		
Labels			
Health Checks			
Volumes			
Optional			

**Fig. 2.** Impostazioni per i valori di cpus, memory e space.

**Edit Application**

JSON Mode

<b>General</b>	<b>Image</b> nginx	<b>Network</b> Bridged
Docker Container	<input type="checkbox"/> Force pull image on every launch <input type="checkbox"/> Extend runtime privileges	
Ports		
Environment Variables		

**Fig. 3.** Impostazioni per il pull del immagine nginx e del tipo di rete da usare per l'application

**Edit Application**

JSON Mode

<b>General</b>	<b>Container Port</b> <input type="text" value="22"/>	<b>Protocol</b> <input type="button" value="tcp"/>	<b>Name</b> <input type="text" value="nginx"/> <input type="button" value="+"/> <input type="button" value="-"/>
Docker Container			
Ports			
Environment Variables	<small>Your Docker container will bind to the requested ports and they will be dynamically mapped to \$PORT0 on the host.</small>		

**Fig. 4.** Impostazioni per la porta da usare per la porta

The screenshot shows the Marathon web interface at [localhost:8080/ui/#/apps](http://localhost:8080/ui/#/apps). The left sidebar has sections for STATUS (Running, Deploying, Suspended, Delayed, Waiting), HEALTH (Healthy, Unhealthy, Unknown), and RESOURCES (Volumes). The main area is titled 'APPLICATIONS' and shows a single application named 'nginx'. To the right of the application list, there is a table with columns: CPU, Memory, Status, Running Instances, and Health. The values shown are 0.3, 128 MiB, Running, 1 of 1, and a green bar indicating healthy status.

**Fig. 5.** Risultato finale.



**Fig. 6.** L'applicazione passa dallo stato **Running** allo stato **Delayed**

In Figura 5 l'applicazione Nginx viene creata ed eseguita correttamente ma abbiamo osservato che questa applicazione viene messa in status **delayed** più volte in modo imprevedibile (Figura 6) per poi tornare nello stato **running**. Per risolvere questo problema abbiamo cercato una configurazione del Docker Compose di Marathon/Mesos più stabile rispetto a quella descritta.

La nuova versione del Docker compose è riportata nella Sezione 10.1 e al seguente link <https://github.com/meltwater/docker-mesos>. Il nuovo docker-compose avvia i container di Mesos e di Marathon ed, insieme a questi, avvia diverse applicazioni che noi abbiamo utilizzato come esempio per il monitoraggio dei dati. La lista delle applicazioni Marathon avviate è la seguente (Figura 7):

- **graphite** (1 istanza)
- **web app**, Hello World Python che usa un server Apache (3 istanze)
- **elasticsearch** (3 istanze)
- **hazelcast**, immagine hazelcast: latest (3 istanze)
- **mysql** (1 istanza)
- **rabbitmq**, immagine rabbitmq:management (1 istanza)

## 4 Configurazione di Zabbix e Mysql

Per comodità abbiamo diviso il docker-compose con la configurazione di Mesos/Marathon dal docker-compose contenente la configurazione di Zabbix e Mysql. All'interno di quest'ultimo file abbiamo creato una nuova rete di tipo bridge chiamata **bda\_project** assegnando a ciascun servizio un indirizzo IP. La subnet della rete bda\_project è 172.16.121.0/24. Maggiori dettagli sulla nostra configurazione si può trovare nella Sezione 10.2.

Il file di configurazione di Mysql riportato in Sezione 10.3 attiva, fra le altre cose, lo scheduler per l'esecuzione degli eventi Mysql descritti in Sezione 6.2. Tale file di configurazione si chiama **mysqld.cnf** e viene caricato nella cartella **/etc/mysql/mysql.conf.d** attraverso il docker-compose. Questo file permette di automatizzare l'avvio dello scheduler nel caso in cui il server Mysql venga riavviato per qualche motivo.

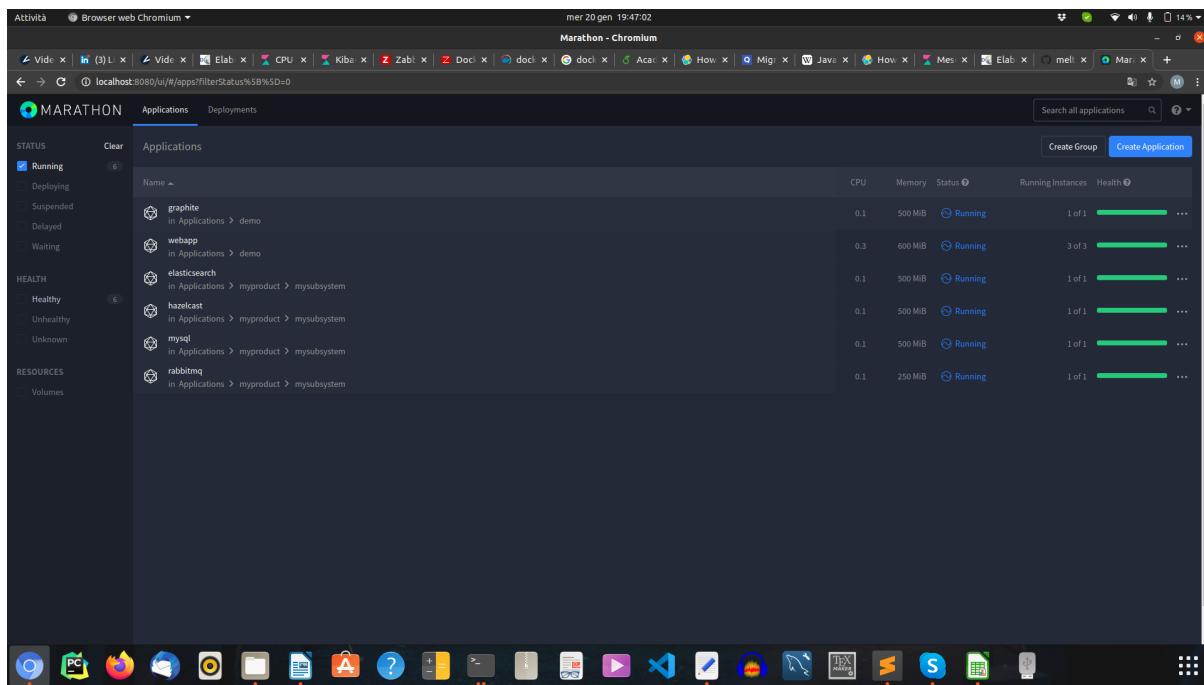


Fig. 7. Lista delle applicazioni avviate con il nuovo Docker Compose

## 5 Configurazione dello Zabbix Server

La configurazione di Zabbix è stata fatta attraverso l'interfaccia grafica disponibile all'indirizzo <http://127.0.0.1>.

Per comodità abbiamo creato un nuovo agente Zabbix all'interno della scheda "Host" del pannello "Configuration". Quest'ultimo è stato configurato come segue:

- **Indirizzo IP:** 172.18.0.1
- **Porta:** 10050
- **Template 1:** Template App Docker

Abbiamo notato che è presente un processo di decimazione nel tempo. Zabbix distingue il concetto di "history", che sta indicare i risultati monitorati per ogni container, dal concetto di "trend", che rappresenta il valore della statistica utilizzata per sintetizzare i risultati ottenuti prima della loro eliminazione. Come richiesto abbiamo settato "History storage period" e "Trend storage period" a 365 giorni mentre il campo "Update interval", che indica ogni quanto Zabbix monitora le applicazioni, l'abbiamo settato a 5 minuti. In particolare dalla regola "Containers discovery" abbiamo eseguito un "Mass Update" di tutti gli "Item prototypes".

### 5.1 Estrazione delle informazioni di Marathon con Zabbix

Attraverso il suo **template app docker**, Zabbix estrae da ogni container molte metriche come **memory usage**, **cpu usage for second**, **network bytes for second** ecc...

Tuttavia al nome del container creato da Marathon viene associato un codice che segue questo pattern:

*mesos-<codice-mesos>-S0.<codice-istanza>,*

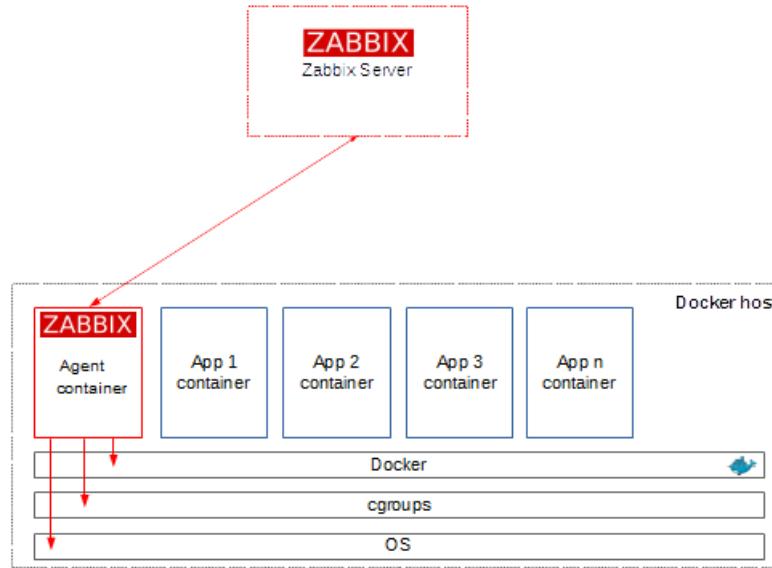
dove *<codice-mesos>* è un codice esadecimale associato al container mentre *<codice-istanza>* è un codice esadecimale che è associata all'istanza dell'applicazione. Questo codice viene riportato su Zabbix in corrispondenza di ogni metrica monitorata. Questo rende complicato risalire al container a cui è associato il valore della metrica monitorata. Per risolvere questo problema abbiamo deciso di recuperare il **Marathon app id**. Il **Marathon app id** è il nome dato all'applicazione creata su Marathon. Per recuperare questa informazione quello che abbiamo fatto è modificare il template app docker creando un nuovo **item prototype** dal nome **MarathonAppId**. Nella creazione del item prototype (Figura: 12) abbiamo seguito le seguenti operazioni:

1. selezionato la voce **Dependent Item** per indicare che l'item è il derivato di un altro item;
2. assegnato una chiave dal nome **docker.container\_info.app\_id["#NAME"]**;
3. specificato la dipendenza con l'item Get info con il master item **Template App Docker: Container #NAME: Get info**;
4. impostato il tipo di informazione da restituire a **Text**;
5. impostato il periodo di conservazione di questa informazione nel database a 365 giorni.

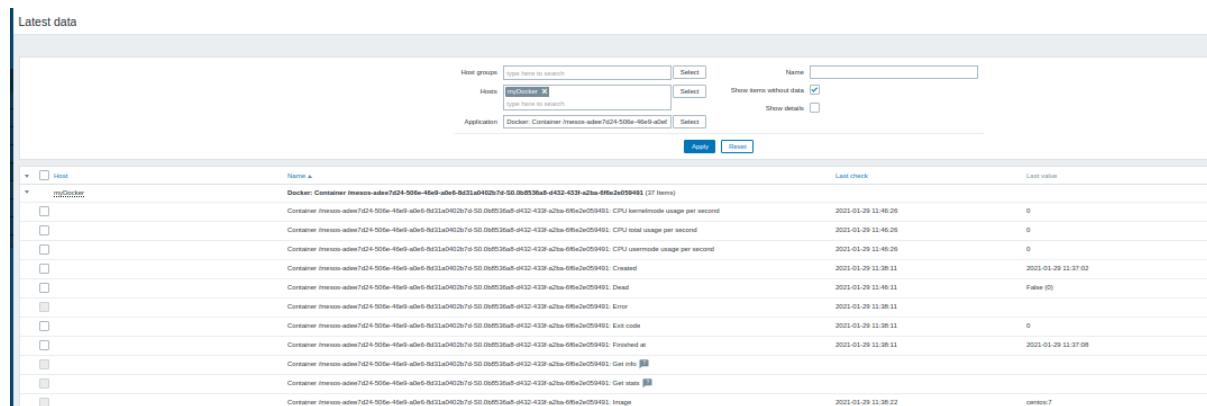
Di seguito abbiamo definito i passi di preprocesing che Zabbix deve eseguire per questo item sull'output del docker inspect (Figura 13):

1. **JSONPath**, indichiamo che l'output è un file JSON e che deve estrarre l'informazione contenuta nel campo **\$.HostConfig.Binds**;
2. Definiamo un'espressione regolare **executorsV([a-z\_0-9]\*)** per estrarre il Marathon App Id e in caso di errore restituiamo il valore *unknown*.

La voce **Docker: Name** del Template App Docker consente di recuperare il nome dell'host/VM/Docker-engine su cui l'agente Zabbix è installato. In caso di più host, quindi, va dichiarato per ciascuno di essi uno Zabbix Agent che recupererà il nome dell'host utilizzando la suddetta voce. La Figura 8 mostra come l'agente Zabbix si interfaccia con l'ambiente sul quale è in esecuzione.



**Fig. 8.** Schema della configurazione dello Zabbix Agent su un host (VM o Docker)



**Fig. 9.** Latest Data

<b>history</b>	Finished at, Started at, CPU kernelmode usage per second, Throttled time, CPU total usage per second, CPU usermode usage per second, Networks bytes received per second, Networks incoming packets dropped per second, Networks errors received per second, Networks packets received per second, Networks bytes sent per second, Networks outgoing packets dropped per second, Networks errors sent per second, Networks packets sent per second
<b>history_uint</b>	Docker: Containers paused, Docker: Containers running, Docker: Containers stopped, Docker: Containers total, Created, Restart count, Dead, Exit code, OOMKilled, Paused, Pid, Restarting, Running, Throttled periods, Throttling periods, Memory commit bytes, Memory commit peak bytes, Memory maximum usage, Memory private working set, Memory usage, Online CPUs
<b>history_text</b>	MarathonAppId
<b>history_str</b>	Docker: Name

**Table 1**

Tabelle delle metriche di Template App Docker

## 6 Monitoraggio delle applicazioni

### 6.1 Latest Data

La Figura 9 mostra i dati recuperati da Zabbix. Si osservi che il container è identificato con una stringa che soddisfa il pattern descritto sopra. Si noti anche che per ogni container monitorato abbiamo la ripetizione del nome del container rilevato da Zabbix seguito dalla metrica corrispondente. Ciascuna riga che compare nella visualizzazione del pannello Latest Data, ossia ogni coppia nome container-metrica, rappresenta un'item. Cliccando sul link "Graph" o "History" si visualizza l'intero storico dei dati monitorati per l'item selezionato.

I dati monitorati sono memorizzati all'interno di un database MySQL per item e per istante temporale. Zabbix memorizza i dati monitorati in verticale, cioè una riga per ogni rilevamento. Questa tipologia di memorizzazione dei dati in verticale, tipica dei database SQL, ha creato molti problemi nella scrittura delle query in quanto porta ad avere query molto lunghe con tempi di risposta elevati.

## 6.2 Query Mysql

Come abbiamo già detto, Zabbix inserisce i dati del monitoraggio in un database Mysql. Le tabelle Mysql che memorizzano i dati che ci interessano sono *history*, *history\_uint*, *history\_text* e *history\_str*. La tabella *history* memorizza i dati relativi a CPU e Network dei container, la tabella *history\_uint* memorizza i dati della memoria RAM dei container mentre le tabelle *history\_str* e *history\_text* memorizzano i dati di tipo stringa, come il nome della macchina virtuale o il nome dell'applicazione Marathon. La tabella 1 riporta tutte le metriche suddivise fra le tabelle Zabbix. Dal momento che questa tabelle crescono molto velocemente e sono coinvolte nelle join, abbiamo pensato di modificarle creando degli indici B-Tree\* per aumentare le prestazioni delle query.

```
#####CREAZIONE DEGLI INDICI B-TREE#####
ALTER TABLE items ADD FULLTEXT(name);
ALTER TABLE history_text ADD FULLTEXT(value);
ALTER TABLE history ADD INDEX(clock);
ALTER TABLE history_uint ADD INDEX(clock);
```

Alle tabelle di Zabbix abbiamo pensato di affiancare delle tabelle aggiuntive per facilitare e rendere più rapido il recupero dei dati con le viste. Abbiamo scritto una procedura chiamata *createHost* che ha il compito di creare una tabella dove sono raccolte tutte le informazioni delle VM dell'infrastruttura. Per la loro natura, i dati presenti in questa tabella non cambiano molto frequentemente per questo riteniamo che sia sufficiente aggiornare le informazioni sugli host una sola volta al giorno.

```
DELIMITER //

CREATE PROCEDURE createHost()
BEGIN
drop table if exists host;
drop view if exists host;
create table host
select h.hostid as hostid, h.host as host, id.ip as ip
from hosts h inner join interface id on h.hostid = id.hostid;
alter table host add primary key (hostid);
END //

DELIMITER ;
```

---

\*Zabbix già di suo crea delle tabelle ottimizzate con gli indici, ma alcuni campi che vengono coinvolti nelle nostre query ne sono sprovvisti.

```
CREATE EVENT create_host_event ON SCHEDULE EVERY 1 DAY
DO
CALL createHost();
```

Abbiamo creato anche una tabella info che ha il compito di recuperare il Marathon App Id ed identificare l'applicazione e il container mesos/docker che la esegue. Riteniamo che sia sufficiente aggiornare la lista delle applicazioni in esecuzione una volta l'ora.

```
DELIMITER //

CREATE PROCEDURE createInfo()
BEGIN
drop view if exists info;
drop table if exists info;
create table info
select distinct
CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 1), ':', -1), ':%') as name,
REPLACE(SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 1), ':', -1),
'Container /mesos-', '') as mesos_id,
h.value as value
from items i inner join history_text h on h.itemid=i.itemid
where i.name like CONCAT('%','MarathonAppId')
and h.value not like 'unknown';
alter table info add primary key (name);
alter table info add fulltext(name);
END //
```

DELIMITER ;

```
CREATE EVENT create_info_event ON SCHEDULE EVERY 1 DAY
DO
CALL createInfo();
```

Per recuperare i dati salvati da Zabbix nel database Mysql abbiamo definito le seguenti viste.

La vista **containerCPU** è utilizzata per recuperare le informazioni sull'utilizzo della CPU per ogni container. L'utilizzo della CPU è indicato da una percentuale.

```
####CREAZIONE DELLA VISTA CHE CONTIENE LE INFORMAZIONI DELL'USO
DELLA CPU DA OGNI CONTAINER####
```

```
create or replace view containerCPU as
select host.host as hostname, host.ip as host_ip, info.value as container,
info.mesos_id as mesos_id, i.name as name, h.clock as clock, h.value as value_CPU
from items i, history h, info, host
where i.itemid=h.itemid and i.hostid =host.hostid
and i.name like 'Container /mesos%CPU total usage per second'
and i.name like info.name;
```

La vista **containerRAM** è utilizzata per recuperare le informazioni sul consumo della memoria RAM da parte di ogni container. Il consumo della memoria RAM è indicato in byte.

```
####CREAZIONE DELLA VISTA CHE CONTIENE LE INFORMAZIONI DELL'USO
DELLA RAM DA OGNI CONTAINER#####
create or replace view containerRAM as
select host.host as hostname, host.ip as host_ip, info.value as container,
info.mesos_id as mesos_id, i.name as name, h.clock as clock, h.value as value_RAM
from items i, history_uint h, info, host
where i.itemid=h.itemid and i.hostid =host.hostid
and i.name like 'Container /mesos%Memory usage' and i.name like info.name;
```

La vista **containerNetworkIO\_Sent** è utilizzata per recuperare la quantità di dati inviati sulla rete da ogni container. Questo valore è misurato in byte.

```
####CREAZIONE DELLA VISTA CHE CONTIENE LE INFORMAZIONI SUL INVIO DI DATI
SULLA RETE DA OGNI CONTAINER#####
create or replace view containerNetworkIO_Sent as
select host.host as hostname, host.ip as host_ip, info.value as container,
info.mesos_id as mesos_id, i.name as name, h.clock as clock, h.value as value_NET_IO_Sent
from items i, history h, info, host
where i.itemid=h.itemid and i.hostid =host.hostid
and i.name like 'Container /mesos%Networks bytes sent per second'
and i.name like info.name;
```

La vista **containerNetworkIO\_Received** è utilizzata per recuperare la quantità di dati ricevuti dalla rete da ogni container. Questo valore è misurato in byte.

```
####CREAZIONE DELLA VISTA CHE CONTIENE LE INFORMAZIONI SULLA RICEZIONE DI DATI
SULLA RETE DA OGNI CONTAINER#####
create or replace view containerNetworkIO_Received as
select host.host as hostname, host.ip as host_ip, info.value as container,
info.mesos_id as mesos_id, i.name as name, h.clock as clock,
h.value as value_NET_IO_Received
from items i, history h, info, host
```

```
where i.itemid=h.itemid and i.hostid =host.hostid
and i.name like 'Container /mesos%Networks bytes received per second'
and i.name like info.name;
```

La vista **containerDocker** è utilizzata per recuperare le informazioni sui container che girano sulle diverse VM. Per ogni VM-Host viene recuperato il numero di container in esecuzione, il numero di container in pausa, il numero di container fermi e il numeri complessivo di container in un certo istante di tempo.

```
#####CREAZIONE DELLA VISTA CHE CONTIENE LE STATISTICHE SU OGNI HOST#####
create or replace view containerDocker as
select host.host as hostname, host.ip as host_ip,
i.name as name, h.clock as clock, h.value as value
from items i, history_uint h, host
where i.itemid=h.itemid and i.hostid =host.hostid
and (i.name = 'Docker: Containers paused' or i.name ='Docker: Containers running'
or i.name= 'Docker: Containers stopped' or i.name= 'Docker: Containers total');
```

Le query Mysql eseguite nel file di configurazione di Logstash per recuperare le informazioni sull'uso della CPU, sulla memoria RAM e il numero di bytes inviati e ricevuti dalla rete sono definite come segue:

```
select hostname, host_ip,
SUBSTRING_INDEX(SUBSTRING_INDEX(host_ip, '.', 4), '.', -1) as host_ip_dett,
container, mesos_id,
SUBSTRING_INDEX(SUBSTRING_INDEX(name, ':', 2), ':', -1) as metrics,
from_unixtime(clock) as time, value
from <nameView>
where clock>UNIX_TIMESTAMP(:sql_last_value)
and clock<UNIX_TIMESTAMP(NOW())
```

dove <nameView> è il nome della vista associata alla metrica. La query per recuperare le informazioni sul numero di container complessivi, in esecuzione, stoppati ed in pausa è la seguente:

```
select hostname, host_ip,
SUBSTRING_INDEX(SUBSTRING_INDEX(host_ip, '.', 4), '.', -1) as host_ip_dett,
SUBSTRING_INDEX(SUBSTRING_INDEX(name, ':', 2), ':', -1) as metrics,
from_unixtime(clock) as time,
value from containerDocker
where clock>UNIX_TIMESTAMP(:sql_last_value)
and clock<UNIX_TIMESTAMP(NOW())
```

La condizione `clock>UNIX_TIMESTAMP(:sql_last_value)` and `clock<UNIX_TIMESTAMP(NOW())` è necessaria per recuperare solo i dati più recenti da Mysql (Vedi Sez. [7.1](#)).

## 6.3 Tabelle dei dati

Il significato dei campi che compaiono nelle query è il seguente:

- hostname, indica il nome dell' host zabbix che ha fatto il rilevamento dati;
- host\_ip, indica l'indirizzo ipv4 dell'host zabbix;
- host\_ip\_dett, riporta le ultime tre cifre dell'ip4;
- time, indica il timestamp della rilevazione dell'host zabbix;
- container, indica il nome dell'applicazione;
- mesosID, indica il nome reale del container che inizia per mesos;
- metrics, nome delle metrica rilevata da zabbix per il container;
- value, valore della metrica.

## 7 Lo stack EKL

Abbiamo osservato che creare delle dashboard accattivanti su Zabbix non è semplice. Per questo motivo abbiamo deciso di utilizzare Kibana che permette di creare visualizzazioni accattivanti in modo più semplice. Per fare questo abbiamo usato i seguenti strumenti:

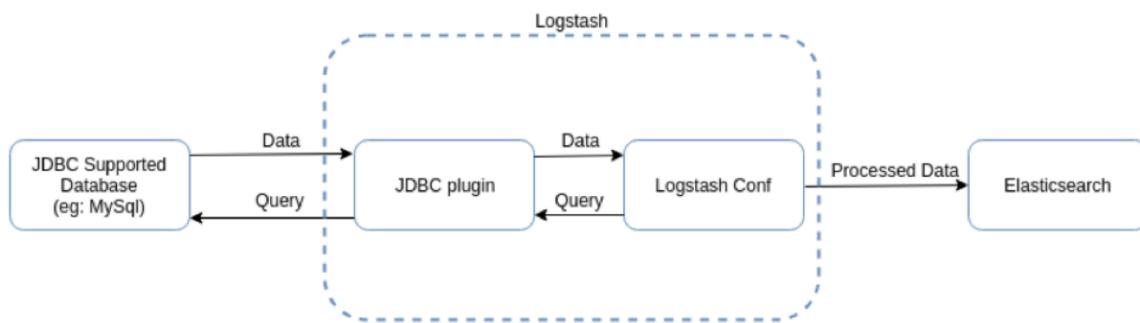
- **logstash** per la fetch dei dati;
- **elasticsearch** per salvare i dati;
- **Kibana** per creare e visualizzare le dashboard.

Questi servizi sono stati installati tutti con il docker compose dal nome **docker-compose-zabbix-kibana-logstash.yml** (Sezione [10.2](#)).

### 7.1 Logstash

Per poter creare una dashboard che monitori i container, occorre che i dati salvati nel database Mysql da Zabbix migrino verso il database di Elasticsearch seguendo lo schema proposto nella Figura [10](#). Il responsabile di questa operazione è Logstash che viene configurato attraverso il file **logstash.conf** (Sezione [10.5](#)).

Il file logstash.conf si compone di tre step collegati tra di loro attraverso una pipeline: uno di input, uno di filtraggio e uno di output. Nello step di *input* abbiamo inserito i dati per interagire con Mysql (per esempio username, password, nome database ecc..) e il path per dire a logstash dove si trova il jar del JDBC. Quest'ultimo era stato precedentemente copiato, con l'istruzione **COPY** da Dockerfile, dalla nostra cartella **./logstash/mysql-connector-java-8.0.11.jar** alla cartella del container **/usr/share/logstash/logstash-core/lib/jars/mysql-connector-java.jar**. Le query da far eseguire a Mysql sono quelle descritte in Sezione [6.2](#) e vengono eseguite periodicamente impostando lo scheduler con la sintassi **rufus:"\*/3 \* \* \* \*"**, che consente di recuperare i dati da mysql ogni tre minuti.



**Fig. 10.** Schema di interazione tra Mysql (con Jdbc), logstasch ed elastisearch

Lo step di *output* invece si occupa di accogliere i dati in un **type** (in Elasticsearch è l'equivalente della tabella) diverso per ogni metrica considerata. Ad ogni **documento** (riga) si assegna un **document\_id** univoco in modo da garantire l'unicità del documento. Si osservi infine che nella query si considerano sempre gli ultimi dati in quanto questo è garantito dalla condizione **time > :sql\_last\_value AND time < NOW()**. Il file logstash.conf è stato copiato nel container creando nel docker compose il volume `./logstash/logstash.conf:/usr/share/logstash/pipeline/logstash.conf`

Come già detto, abbiamo impostato il file di configurazione di Logstash per eseguire più di una query contemporaneamente. Per distinguere i valori ottenuti da una query da quelli ottenuti da un'altra abbiamo utilizzato gli indici Elasticsearch. Abbiamo inoltre aggiunto alla pipeline lo step *filter* per forzare il cast ad integer del campo *host\_ip\_dett*.

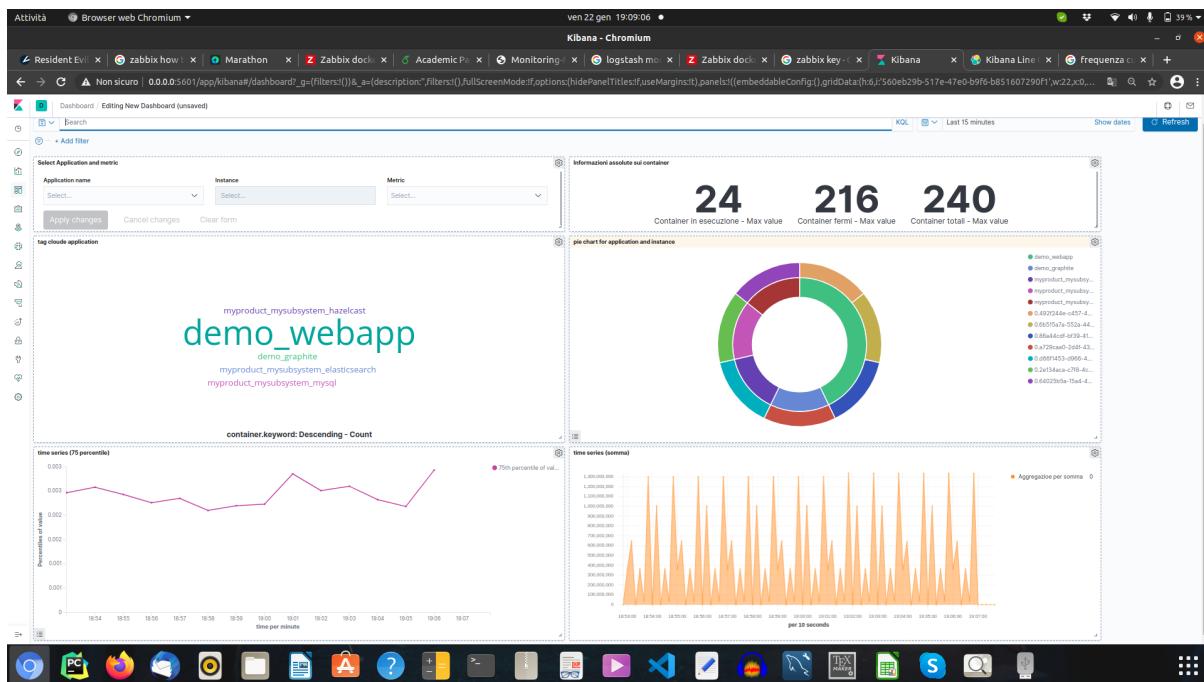
## 7.2 Kibana

Dopo aver caricato i dati su Elasticsearch, abbiamo creato una semplice dashboard sperimentale. La Figura 11 riporta il primo prototipo di dashboard da noi creato che poi abbiamo arricchito quando siamo passati all'ambiente contenente dati reali. Il prototipo di dashboard si compone di:

- un pannello per selezionare l'applicazione, l'istanza e la metrica desiderata, si veda la Figura 14
- un visualizzatore dei container in esecuzione, stoppati e totali, si veda la figura 15
- un tag cloud delle applicazioni che sono in esecuzione, si veda la Figura 16
- un pie chart che mostra la corrispondenza tra le applicazioni e i container che le eseguono, si veda la Figura 17
- una time series aggregata per il 75-percentile rispetto alla metrica e applicazione scelta, si veda la Figura 18
- una time series aggregata per la voce somma rispetto alla metrica e applicazione scelta, si veda la Figura 19

## 8 Verso Snap4City

Una volta che l'esecuzione del procedimento in locale è avvenuto con successo, abbiamo provveduto ad installare il tutto all'interno della piattaforma **Snap4City**, grazie al prezioso aiuto fornito dal



**Fig. 11.** Prima dashboard creata su Kibana attraverso il nostro ambiente di sviluppo

Professor Nesi e dal Professor Bellini. In particolare, la migrazione è stata effettuata caricando ed eseguendo sull'infrastruttura del **disit-lab** il file **docker-compose-zabbix-kibana-logstash.yml** e rieseguendo i passi descritti nelle sezioni precedenti. Abbiamo quindi riconfigurato il Template App Docker di Zabbix in esecuzione sul disit-lab, eseguito le query Mysql attraverso **phpMyAdmin** (sui nostri pc usavamo MySQL Workbench) e atteso che Logstash recuperasse i dati. Per quanto riguarda la dashboard, siamo partiti dalla nostra dashboard sperimentale per poi crearne una più ricca e funzionale. Si osservi la Figura 20. Alla nuova dashboard abbiamo usato un look and feel scuro e creato dei nuovi grafici che visualizzano nel tempo il numero di container in esecuzione, in pausa, fermi e totali per ogni host dell'infrastruttura. Si osservi la Figura 21. Inoltre c'è stato richiesto di tracciare gli spostamenti delle varie applicazioni (sono circa 460) tra i diversi host riportati in Figura 22. Abbiamo infine aggiunto delle visualizzazioni per il consumo della CPU, della RAM, della quantità di byte ricevuti e inviati attraverso la rete. Le Figure 23 , 26, 24, 25 mostrano l'andamento delle time-series ottenute. Abbiamo avuto alcuni problemi di visualizzazione dei dati: impostando una finestra temporale maggiore delle ultime otto ore, Kibana creava troppi bucket e questi eccedevano il massimo numero di bucket consentiti impostato per default a 10.000 bucket. Per risolvere questo problema abbiamo impostato il limite dei bucket a 65.000 attraverso la Kibana console in questo modo:

```
PUT _cluster/settings
{
  "transient": {
    "search.max_buckets": 65000
  }
}
```

## 8.1 Sviluppi futuri

Tra i tanti tool che Kibana mette a disposizione per lavorare con i big data, c'è anche uno strumento chiamato **Elastic Machine Learning** per fare appunto Machine Learning. Questo strumento mette a disposizione solo algoritmi unsupervised per il clustering per rilevare pattern nei dati o algoritmi di anomaly detection per time series. Elastic Machine Learning mette a disposizione due tipi di job per condurre un task di anomaly detection. Il **single job** dove viene presa in considerazione una singola metrica e il **multi job** dove vengono prese in considerazione più metriche per condurre una analisi multi-variata. Il nostro lavoro sulla piattaforma Snap4city si è concluso con la creazione della pipeline dei dati. Tuttavia per gioco abbiamo utilizzato questo strumento sui dati raccolti dai nostri computer ma non abbiamo trovato alcuna anomalia. Ciò è dovuto molto probabilmente al fatto che le applicazioni Marathon in esecuzione sui nostri computer non eseguivano alcuna attività.

## 9 Problemi risolti

Durante il progetto abbiamo dovuto trovare una soluzione per le seguenti problematiche:

1. Recuperare l' ID di Marathon per identificare l'applicazione/container in esecuzione, in quanto Zabbix rileva solo l' ID di Mesos e questo è poco significativo per l'identificazione dell'applicazione/container.

*Risolto eseguendo una espressione regolare che viene eseguita sul JSON restituito dall'item Get Info del template utilizzato;*

2. Recuperare l'IP e la porta del container. Zabbix non riporta un item per tracciare l'IP e la porta del container. Quest'informazione sarebbe disponibile all'interno della voce *Env* del docker inspect. Tuttavia questa parte non compare all'interno del JSON restituito dall'item Get Info di Zabbix. Sembra che il risultato dell'item non riporti tutte le informazioni presenti nel docker inspect.

*Per quanto riguarda l'IP dell'host abbiamo osservato che l'agente Zabbix in esecuzione sull'host riporta il suo nome sotto la metrica Docker: Name del template utilizzato. Non sembra possibile invece recuperare la porta dell'applicazione in quanto conosciuta solo da Marathon. Si consiglia di eseguire una GET con l'indirizzo di Marathon seguito da /v2/apps/ e dal nome dell'applicazione;*

3. Recuperare la quantità di spazio su disco utilizzato da ciascun container in quanto non siamo riusciti a trovare un item di Zabbix che monitori questa risorsa. Il template "Template App Docker" mette a disposizione l'item "Docker: Volume size" ma quest'ultimo monitora la quantità di spazio su disco utilizzato globale ma non del singolo container.

*Non è possibile farlo con il Template App Docker finora utilizzato;*

4. Capire come disegnare delle timeseries in Kibana senza aggregazione.

*In Kibana purtroppo è obbligatorio utilizzare una funzione di aggregazione per disegnare le timeseries. Noi abbiamo utilizzato la funzione di aggregazione MAX e la funzione di aggregazione SUM che calcolano rispettivamente il massimo e il totale dei valori presenti*

*all'interno di un bucket, cioè di un intervallo temporale. In un successivo colloquio con il professore abbiamo capito che l'aggregazione è necessaria ed un eventuale grafico senza aggregazione non sarebbe corretto.*

Le difficoltà descritte in questa sezione ed incontrate durante lo svolgimento di questo progetto sono dovute in parte alla scarsa qualità della documentazione del template da noi utilizzato e di Zabbix in generale.

## 10 Appendice

### 10.1 docker-compose-mesos.yml

```
zookeeper:
  image: mesoscloud/zookeeper:3.4.6-centos-7
  ports:
    - "2181:2181"
    - "2888:2888"
    - "3888:3888"
  environment:
    SERVERS: 'server.1=127.0.0.1'
    MYID: '1'

mesosmaster:
  image: mesosphere/mesos-master:1.0.0
  net: host
  volumes:
    - /var/log/mesos/master
  environment:
    MESOS_ZK: 'zk://localhost:2181/mesos'
    MESOS_QUORUM: '1'
    MESOS_CLUSTER: 'local'
    MESOS_HOSTNAME: 'localhost'
    MESOS_LOG_DIR: '/var/log/mesos/master'

mesosslave:
  image: mesosphere/mesos-slave:1.0.0
  net: host
  privileged: true
  volumes:
    - /var/log/mesos/slave
    - /sys:/sys
  # /cgroup is needed on some older Linux versions
  #   - /cgroup:/cgroup
  # /usr/bin/docker is needed if you're running an older docker version
  #   - /usr/local/bin/docker:/usr/bin/docker:r
  #     - /var/run/docker.sock:/var/run/docker.sock:rw
  environment:
    MESOS_MASTER: 'zk://localhost:2181/mesos'
    MESOS_PORT: '5051'
    MESOS_LOG_DIR: '/var/log/mesos/slave'
```

```
MESOS_CONTAINERIZERS: 'docker,mesos'
MESOS_EXECUTOR_REGISTRATION_TIMEOUT: '5mins'
MESOS_EXECUTOR_SHUTDOWN_GRACE_PERIOD: '90secs'
MESOS_DOCKER_STOP_TIMEOUT: '60secs'
MESOS_WORK_DIR: '/tmp'

# If your workstation doesn't have a resolvable hostname/FQDN then $MESOS_HOSTNAME needs to
#      MESOS_HOSTNAME: 10.87.1.123

marathon:
  image: mesosphere/marathon:v1.3.3
  net: host
  environment:
    MARATHON_ZK: 'zk://localhost:2181 marathon'
    MARATHON_MASTER: 'zk://localhost:2181/mesos'
    MARATHON_EVENT_SUBSCRIBER: 'http_callback'
    MARATHON_TASK_LAUNCH_TIMEOUT: '300000'

marathonsubmit:
  build: marathon-submit
  net: host
  environment:
    MARATHON_URL: http://localhost:8080
  volumes:
    - './marathon-submit:/submit'
# Enables using local development repo of lighter
#      - '../lighter:/lighter'

chronos:
  image: mesosphere/chronos:chronos-2.5.0-0.1.20160223054243.ubuntu1404-mesos-0.27.1-2.0.22
  command: '/usr/bin/chronos run_jar --http_port 4400 --zk_hosts localhost:2181 --master zk
  net: host

chronossSubmit:
  build: chronos-submit
  net: host
  environment:
    CHRONOS_URL: 'http://localhost:4400'

gateway:
  image: meltwater/proxymatic:latest
#  build: ../proxymatic
```

```
net: host
environment:
  MARATHON_URL: 'http://localhost:8080'
  EXPOSE_HOST: 'true'
  VHOST_DOMAIN: 'localdomain'
  VERBOSE: 'true'

secretary:
  image: meltwater/secretary:latest
  net: host
  volumes:
    - './marathon-submit/site/keys:/keys'
```

## 10.2 docker-compose-zabbix-kibana-logstasc.yml

```
version: "3"

services:
  #####Servizi per Zabbix#####
  mysql:
    container_name: mysql
    image: mysql:5.7
    networks:
      - bda_project
    ports:
      - '3306:3306'
    networks:
      bda_project:
        ipv4_address: 172.16.121.6
    volumes:
      - './zabbix/mysql:/var/lib/data'
      - './zabbix/config/myqlld.cnf:/etc/mysql/mysql.conf.d/myqlld.cnf'
    environment:
      - MYSQL_ROOT_PASSWORD=carryontech
      - MYSQL_DATABASE=zabbix
      - MYSQL_USER=zabbix
      - MYSQL_PASSWORD=carryontech

  phpmyadmin:
    depends_on:
      - mysql
    image: phpmyadmin/phpmyadmin
    restart: always
    ports:
      - '8081:80'
    links:
      - mysql
    environment:
      PMA_HOST: mysql
      MYSQL_ROOT_PASSWORD: carryontech
    networks:
      bda_project:
        ipv4_address: 172.16.121.9

  zabbix-server:
```

```
container_name: zabbix-server
image: zabbix/zabbix-server-mysql:ubuntu-5.0.1
networks:
  - bda_project
links:
  - mysql
user: root
restart: always
ports:
  - '10051:10051'
networks:
  bda_project:
    ipv4_address: 172.16.121.7
volumes:
  - './zabbix/alertscripts:/usr/lib/zabbix/alertscripts'
#  - './zabbix/config:/etc/zabbix'
#  - ./zabbix/data:/home/ZabbixExport:z
environment:
  - DB_SERVER_HOST=mysql
  - MYSQL_DATABASE=zabbix
  - MYSQL_USER=zabbix
  - MYSQL_PASSWORD=carryontech
depends_on:
  - mysql

zabbix-frontend:
  container_name: zabbix-frontend
  image: zabbix/zabbix-web-apache-mysql:ubuntu-5.0.1
  networks:
    - bda_project
  links:
    - mysql
  restart: always
  ports:
    - '80:8080'
    - '443:8443'
  networks:
    bda_project:
      ipv4_address: 172.16.121.8
  environment:
    - DB_SERVER_HOST=mysql
```

```
- MYSQL_DATABASE=zabbix
- MYSQL_USER=zabbix
- MYSQL_PASSWORD=carryontech
- PHP_TZ=Europe/Rome
depends_on:
- mysql
# volumes:
#   - './zabbix/config/zabbix.conf.php:/usr/share/zabbix/conf/zabbix.conf.php'

zabbix-agent:
  container_name: zabbix-agent
  image: zabbix/zabbix-agent2:alpine-5.0.1
  user: root
  networks:
    bda_project:
      ipv4_address: 172.16.121.10
  links:
    - zabbix-server
  restart: always
  privileged: true
  volumes:
    - /var/run:/var/run
  ports:
    - '10050:10050'
  environment:
    - ZBX_HOSTNAME=Zabbix server
    - ZBX_SERVER_HOST=172.16.121.7

networks:
  bda_project:
    driver: bridge
    ipam:
      driver: default
      config:
        -
          subnet: 172.16.121.0/24
```

### 10.3 mysqld.cnf

```
# Copyright (c) 2014, 2016, Oracle and/or its affiliates. All rights reserved.
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License, version 2.0,
# as published by the Free Software Foundation.
#
# This program is also distributed with certain software (including
# but not limited to OpenSSL) that is licensed under separate terms,
# as designated in a particular file or component or in included license
# documentation. The authors of MySQL hereby grant you an additional
# permission to link the program and your derivative works with the
# separately licensed software that they have included with MySQL.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License, version 2.0, for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
#
# The MySQL Server configuration file.
#
# For explanations see
# http://dev.mysql.com/doc/mysql/en/server-system-variables.html

[mysqld]
pid-file = /var/run/mysqld/mysqld.pid
socket = /var/run/mysqld/mysqld.sock
datadir = /var/lib/mysql
event_scheduler=ON
#log-error = /var/log/mysql/error.log
# By default we only accept connections from localhost
#bind-address = 127.0.0.1
# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0
```

## 10.4 Docker file per Logstash

```
FROM docker.elastic.co/logstash/logstash:7.3.2
```

```
# install dependency
RUN /usr/share/logstash/bin/logstash-plugin install logstash-input-jdbc
RUN /usr/share/logstash/bin/logstash-plugin install logstash-filter-aggregate
RUN /usr/share/logstash/bin/logstash-plugin install logstash-filter-jdbc_streaming
RUN /usr/share/logstash/bin/logstash-plugin install logstash-filter-mutate

# copy lib database jdbc jars
COPY ./logstash/mysql-connector-java-8.0.11.jar /usr/share/logstash/
logstash-core/lib/jars/mysql-connector-java.jar
```

## 10.5 Configurazione file logstash.conf

```
input {
    jdbc {
        jdbc_driver_library => "${LOGSTASH_JDBC_DRIVER_JAR_LOCATION}"
        jdbc_driver_class => "${LOGSTASH_JDBC_DRIVER}"
        jdbc_connection_string => "${LOGSTASH_JDBC_URL}"
        jdbc_user => "${LOGSTASH_JDBC_USERNAME}"
        jdbc_password => "${LOGSTASH_JDBC_PASSWORD}"
        schedule => "*/3 * * * *"

        # our query
        statement => "select hostname, host_ip, SUBSTRING_INDEX(SUBSTRING_INDEX(host_ip, '.') , '.', -1) as host, port, type, tracking_column, tracking_column_type, use_column_value, last_run_metadata_path"
        tracking_column => time
        tracking_column_type => "timestamp"
        use_column_value => true
        type => "containercpu"
        last_run_metadata_path => "/home/containercpu"
    }
}

jdbc {
    jdbc_driver_library => "${LOGSTASH_JDBC_DRIVER_JAR_LOCATION}"
    jdbc_driver_class => "${LOGSTASH_JDBC_DRIVER}"
    jdbc_connection_string => "${LOGSTASH_JDBC_URL}"
    jdbc_user => "${LOGSTASH_JDBC_USERNAME}"
    jdbc_password => "${LOGSTASH_JDBC_PASSWORD}"
    schedule => "*/3 * * * *"
```

```
# our query
    statement => "select hostname, host_ip, SUBSTRING_INDEX(SUBSTRING_INDEX(host_ip, '.',
    tracking_column => time
    tracking_column_type => "timestamp"
    use_column_value => true
    type => "containernetworkio_sent"
    last_run_metadata_path => "/home/containernetworkio_sent"
}

jdbc {
    jdbc_driver_library => "${LOGSTASH_JDBC_DRIVER_JAR_LOCATION}"
    jdbc_driver_class => "${LOGSTASH_JDBC_DRIVER}"
    jdbc_connection_string => "${LOGSTASH_JDBC_URL}"
    jdbc_user => "${LOGSTASH_JDBC_USERNAME}"
    jdbc_password => "${LOGSTASH_JDBC_PASSWORD}"
    schedule => "*/* * * *"
    last_run_metadata_path => "/home/containernetworkio_received"
# our query
    statement => "select hostname, host_ip, SUBSTRING_INDEX(SUBSTRING_INDEX(host_ip, '.',
    tracking_column => time
    tracking_column_type => "timestamp"
    use_column_value => true
    type => "containernetworkio_received"
    last_run_metadata_path => "/home/containernetworkio_received"
}

jdbc {
    jdbc_driver_library => "${LOGSTASH_JDBC_DRIVER_JAR_LOCATION}"
    jdbc_driver_class => "${LOGSTASH_JDBC_DRIVER}"
    jdbc_connection_string => "${LOGSTASH_JDBC_URL}"
    jdbc_user => "${LOGSTASH_JDBC_USERNAME}"
    jdbc_password => "${LOGSTASH_JDBC_PASSWORD}"
    schedule => "*/* * * *"
# our query
    statement => "select hostname, host_ip, SUBSTRING_INDEX(SUBSTRING_INDEX(host_ip, '.',
    tracking_column => time
    tracking_column_type => "timestamp"
    use_column_value => true
    type => "containerram"
    last_run_metadata_path => "/home/containerram"
}
```

```
jdbc {  
    jdbc_driver_library => "${LOGSTASH_JDBC_DRIVER_JAR_LOCATION}"  
    jdbc_driver_class => "${LOGSTASH_JDBC_DRIVER}"  
    jdbc_connection_string => "${LOGSTASH_JDBC_URL}"  
    jdbc_user => "${LOGSTASH_JDBC_USERNAME}"  
    jdbc_password => "${LOGSTASH_JDBC_PASSWORD}"  
    schedule => "*/3 * * * *"  
  
    # our query  
    statement => "select hostname, host_ip, SUBSTRING_INDEX(SUBSTRING_INDEX(host_ip, '.'). . .  
    tracking_column => time  
    tracking_column_type => "timestamp"  
    use_column_value => true  
    type => "containerdocker"  
    last_run_metadata_path => "/home/containerdocker"  
}  
  
  
jdbc {  
    jdbc_driver_library => "${LOGSTASH_JDBC_DRIVER_JAR_LOCATION}"  
    jdbc_driver_class => "${LOGSTASH_JDBC_DRIVER}"  
    jdbc_connection_string => "${LOGSTASH_JDBC_URL}"  
    jdbc_user => "${LOGSTASH_JDBC_USERNAME}"  
    jdbc_password => "${LOGSTASH_JDBC_PASSWORD}"  
    schedule => "*/3 * * * *"  
  
    # our query  
    statement => "select hostname, host_ip, SUBSTRING_INDEX(SUBSTRING_INDEX(host_ip, '.'). . .  
    tracking_column => time  
    tracking_column_type => "timestamp"  
    use_column_value => true  
    type => "containererror"  
    last_run_metadata_path => "/home/containererror"  
}  
  
}  
  
filter {  
    mutate {  
        convert => {  
            "host_ip_dett" => "integer"  
        }  
    }  
}
```

```
    }

}

output {

  if[type] == "containercpu" {
    elasticsearch {
      hosts => "elasticsearch:9200"
      index => "containercpu"
      document_id => "%{container} %{metrics} %{time}"
    }
  }

  if[type] == "containernetworkio_sent" {
    elasticsearch {
      hosts => "elasticsearch:9200"
      index => "containernetworkio_sent"
      document_id => "%{container} %{metrics} %{time}"
    }
  }

  if[type] == "containernetworkio_received" {
    elasticsearch {
      hosts => "elasticsearch:9200"
      index => "containernetworkio_received"
      document_id => "%{container} %{metrics} %{time}"
    }
  }

  if[type] == "containerram" {
    elasticsearch {
      hosts => "elasticsearch:9200"
      index => "containerram"
      document_id => "%{container} %{metrics} %{time}"
    }
  }

  if[type] == "containerdocker" {
    elasticsearch {
      hosts => "elasticsearch:9200"
      index => "containerdocker"
      document_id => "%{container} %{metrics} %{time}"
    }
  }
}
```

```
        }
    }

    if[type] == "containererror" {
        elasticsearch {
            hosts => "elasticsearch:9200"
            index => "containererror"
            document_id => "%{container} %{metrics} %{time}"
        }
    }

    stdout {
        codec => json_lines
    }
}
```

## 11 Zabbix gallery

### Item prototypes

The screenshot shows the 'Item prototypes' configuration page in Zabbix. The top navigation bar includes links for 'All templates', 'Template App Docker', 'Discovery list', 'Containers discovery', 'Item prototypes 36', 'Trigger prototypes 2', 'Graph prototypes 4', and 'Host prototypes'. The current tab is 'Item prototype'.

**Preprocessing**

- Name:** Container {#NAME}: MarathonAppId
- Type:** Dependent item
- Key:** docker.container\_info.marathon\_app\_id["{#NAME}"]
- Master item:** Template App Docker: Container {#NAME}: Get info
- Type of information:** Text
- History storage period:** Do not keep history

**New application**: Applications -None-, Docker, Zabbix raw items

**New application prototype**: Application prototypes -None-, Docker: Container {#NAME}

**Description**: (empty)

**Create enabled:**

**Discover:**

**Buttons:** Add, Test, Cancel

**Fig. 12.** Configurazione del prototype item per recuperare il marathon app id.

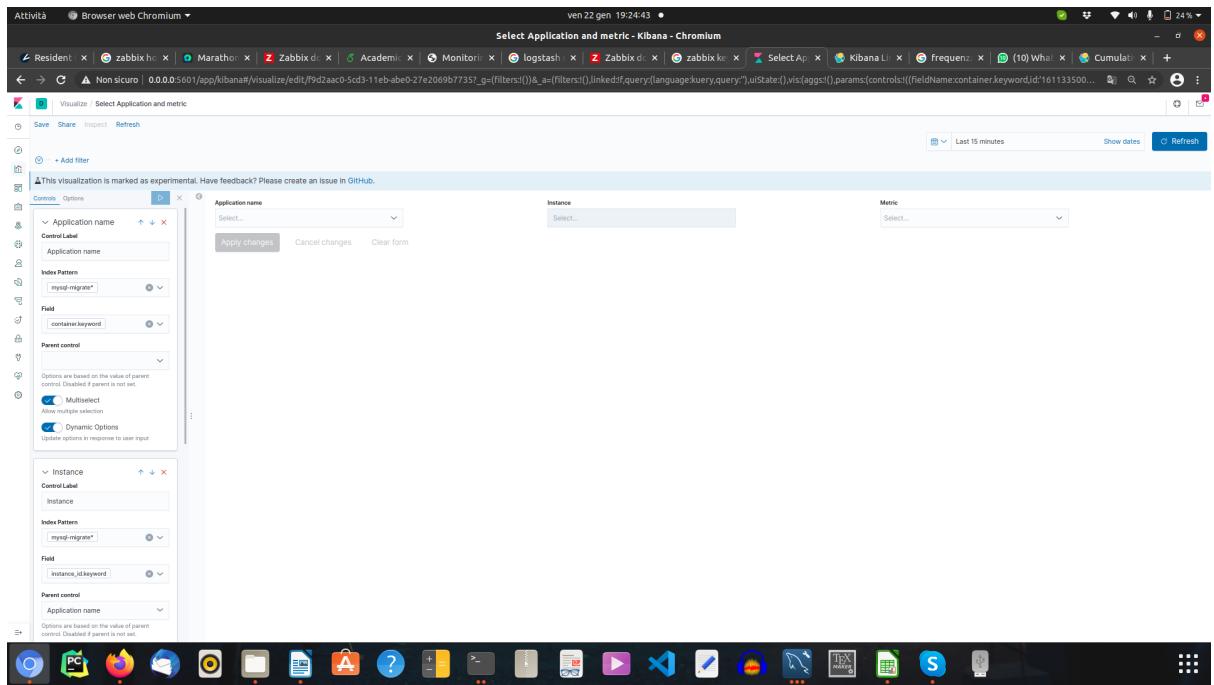
The screenshot shows the 'Preprocessing' configuration page for the prototype item. The top navigation bar is identical to Fig. 12.

**Preprocessing**

Step	Name	Parameters	Custom on fail	Actions
1:	JSONPath	\$.HostConfig.Binds	<input checked="" type="checkbox"/>	<a href="#">Test</a> <a href="#">Remove</a>
2:	Regular expression	executorsV([a-zA-Z_0-9]*).\\1	<input checked="" type="checkbox"/>	<a href="#">Test</a> <a href="#">Remove</a>

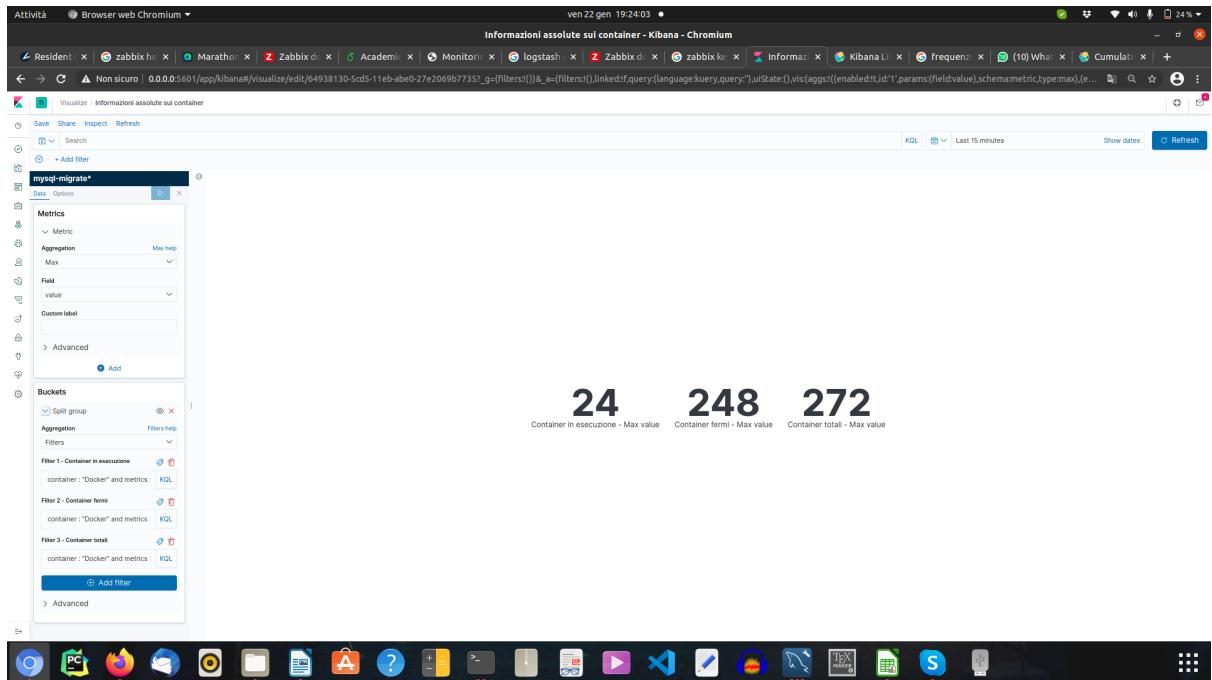
**Buttons:** Add, Update, Clone, Test, Delete, Cancel, Test all steps

**Fig. 13.** Configurazione dei passi di pre processing per recuperare il marathon app id con proto item personalizzato.

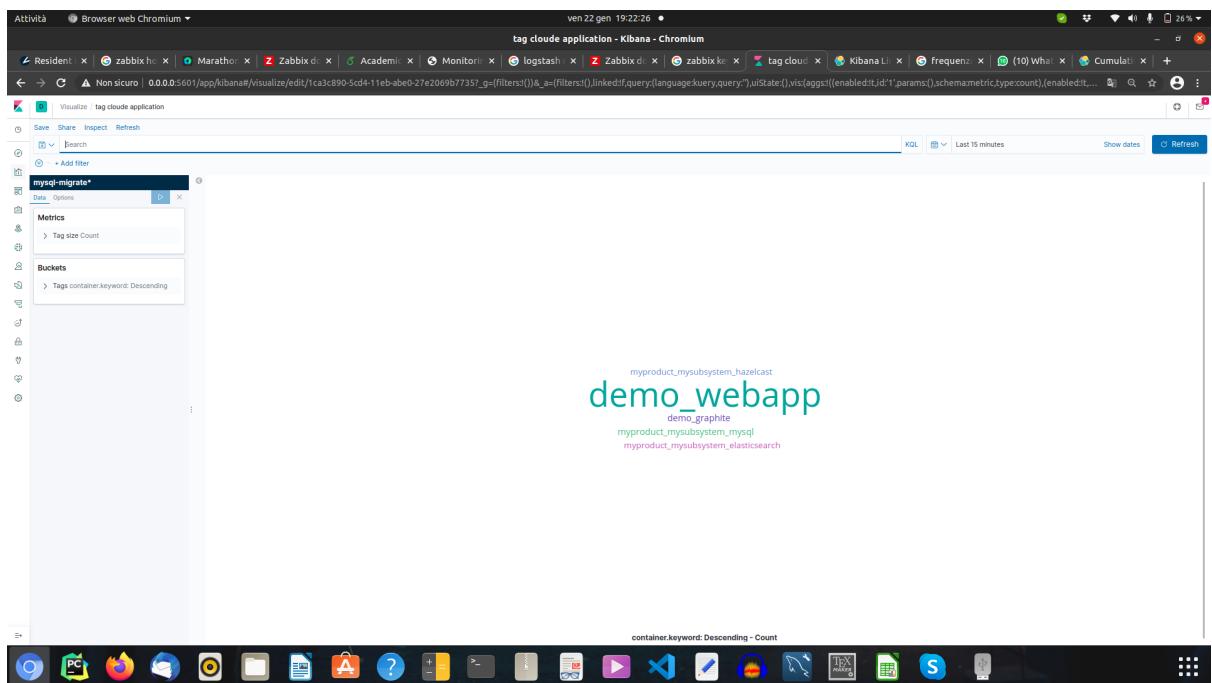


**Fig. 14.** Selettore dell'applicazione e delle metrica

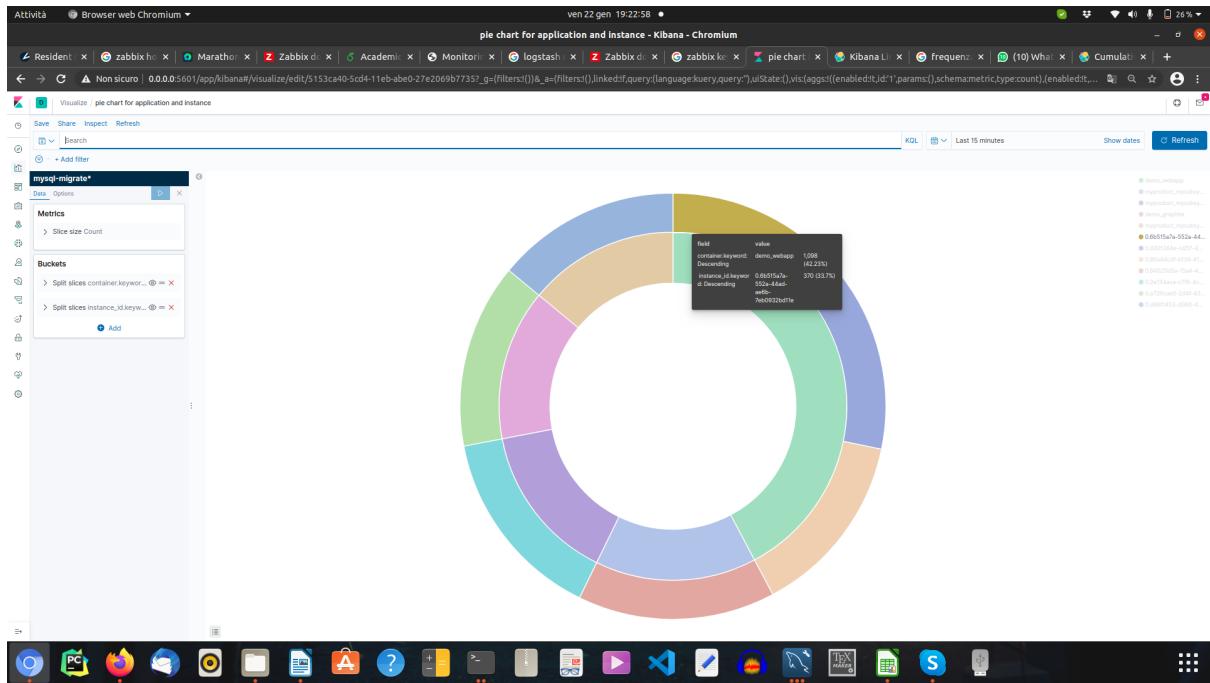
## 12 Kibana gallery



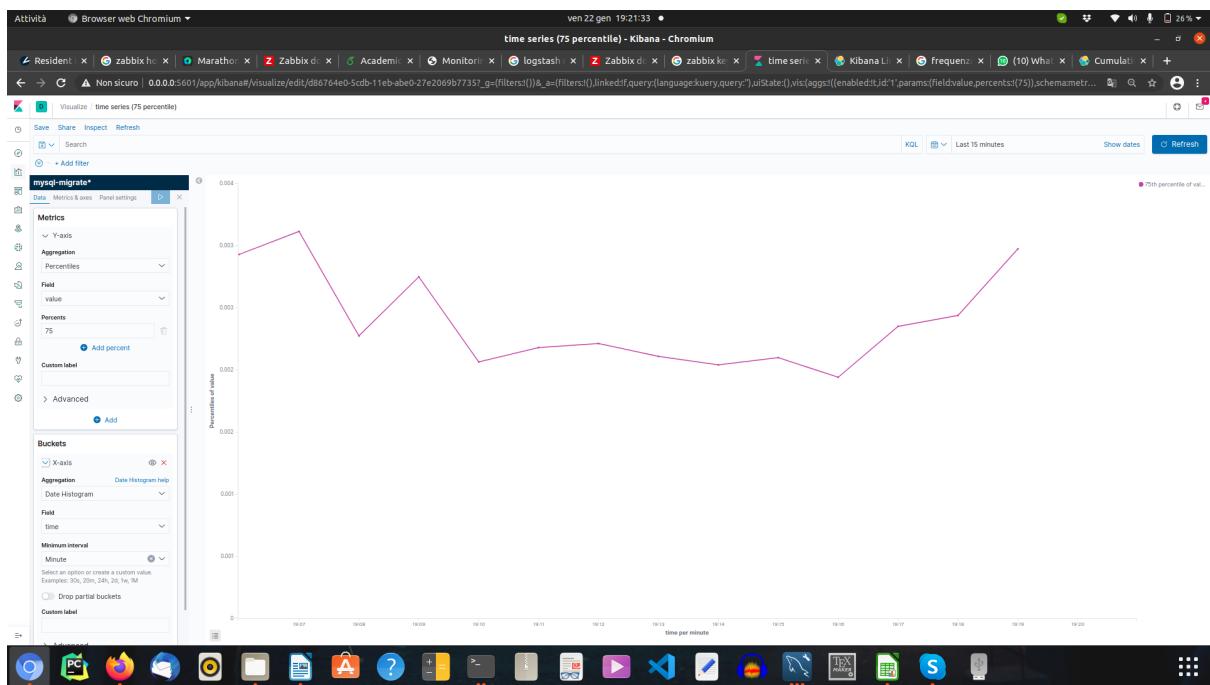
**Fig. 15.** Visualizzatore dei container attivi, stoppati e totali



**Fig. 16.** Tag cloude per visualizzare le applicazioni



**Fig. 17.** Piechart che mostrano i container in esecuzione che eseguono le diverse applicazioni



**Fig. 18.** Time series che plotta il 75-percentile per il container e metrica scelta

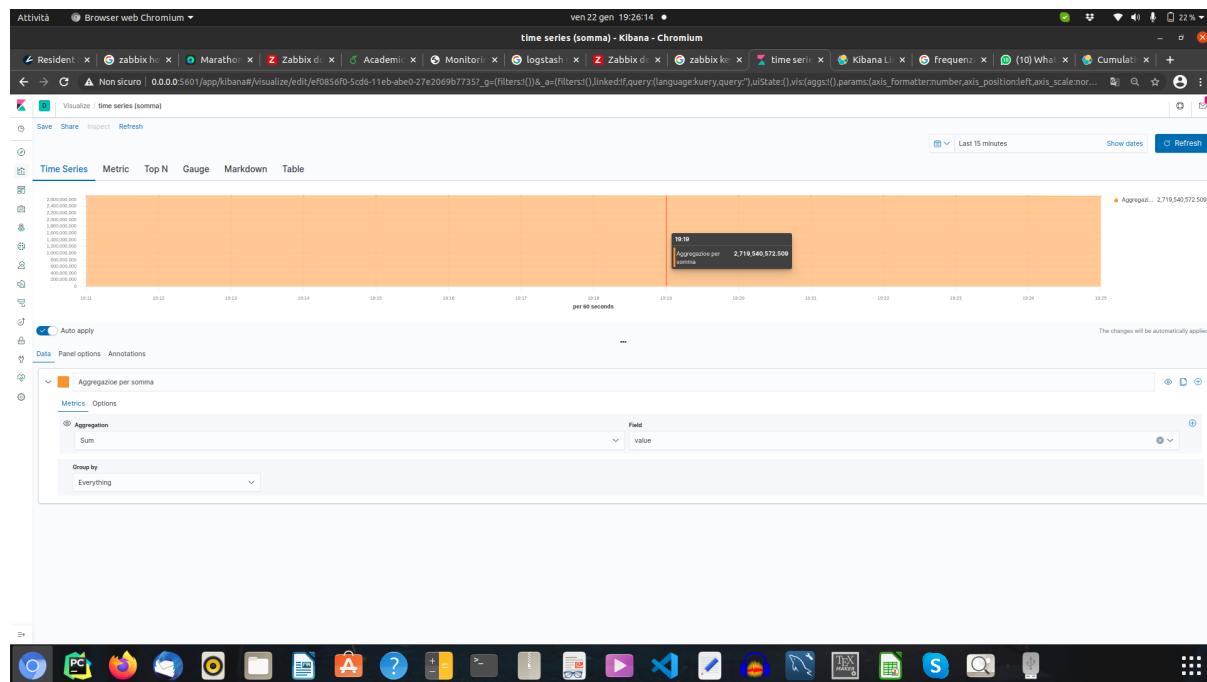
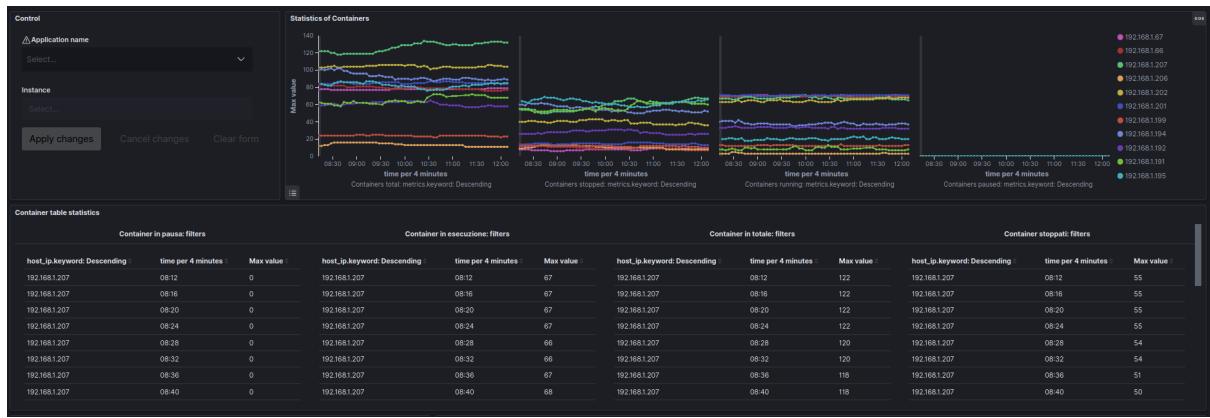


Fig. 19. Time series aggregata per somma per il container e metrica scelta

## 13 Snap4City dashboard





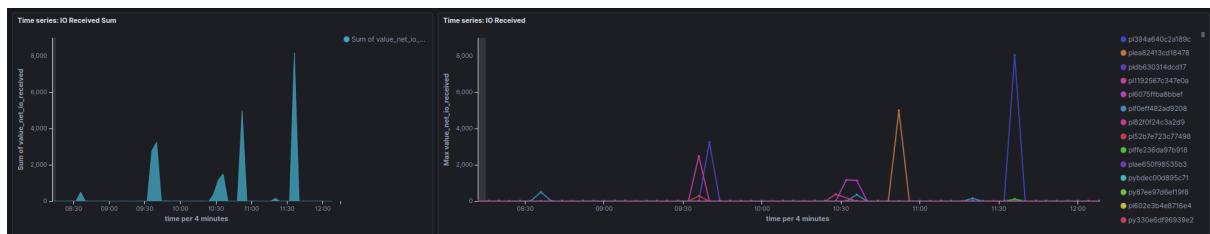
**Fig. 21.** Dashboard-Snap4City per le informazioni dei container



**Fig. 22.** Dashboard-Snap4City per gli spostamenti degli host



**Fig. 23.** Dashboard in esecuzione su Snap4City per il consumo della CPU



**Fig. 24.** Dashboard in esecuzione su Snap4City per la ricezione dati



**Fig. 25.** Dashboard in esecuzione su Snap4City per l'invio dati



**Fig. 26.** Dashboard in esecuzione su Snap4City per il consumo di memoria RAM