Relazione del progetto di Big Data Architectures

Matteo Marulli, Matteo Ghera[1]

[1]Corso di Laurea Magistrale in Informatica: Curriculum Data Science

2809 Words Student ID:7005652, 7006227

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

29th January, 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 eccessi di risorse e possibili malfunzionamenti delle app 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 e infine per analizzare e trovare le anomalie nei dati abbiamo usato Elastic Machine Learning .

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 dashbord e gli strumenti messi a disposizione da 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 di 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 che girano su container docker. Per sviluppare il progetto abbiamo usato diversi strumenti che introduciamo via via per un'analisi più dettagliata. In Appendice 8 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

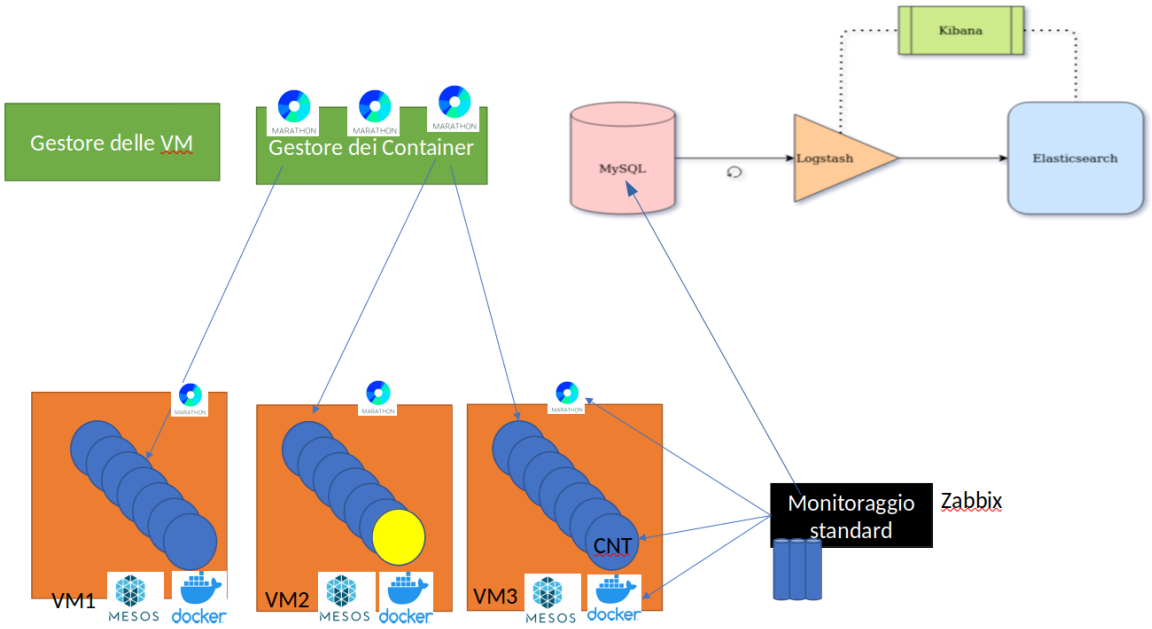

Fig. 1. Schema di progettazione

2 Configurazione di Apache Mesos e Marathon

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 tutti i servizi necessari per il corretto funzionamento di Mesos e Marathon, ossia:

- Apache Zookeper 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

Per comodità abbiamo assegnato a ciascuno di essi un indirizzo IP all'interno di una rete di tipo bridge chiamata **bda_project**. L'indirizzo IP di questi servizi è 172.16.121.X dove $X = 2, ..., 5$. Maggiori dettagli sulla nostra configurazione si può trovare nella Sezione 8.2.

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:

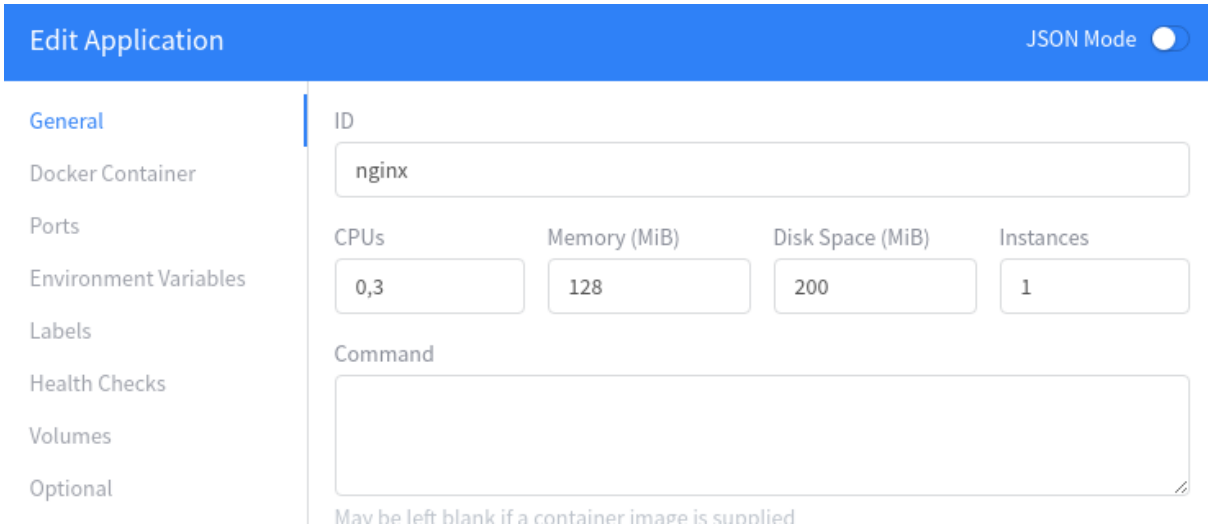

Fig. 2. Impostazioni per i valori di cpus, memory e space.

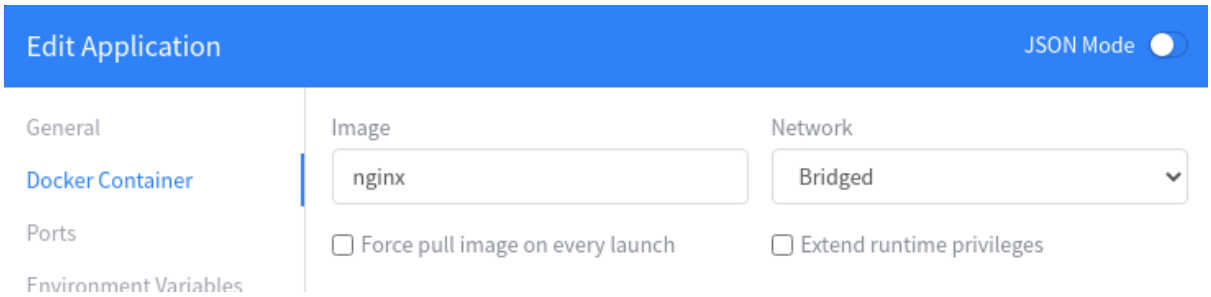

Fig. 3. Impostazioni per il pull del immagine nginx e del tipo di rete da usare per l'application

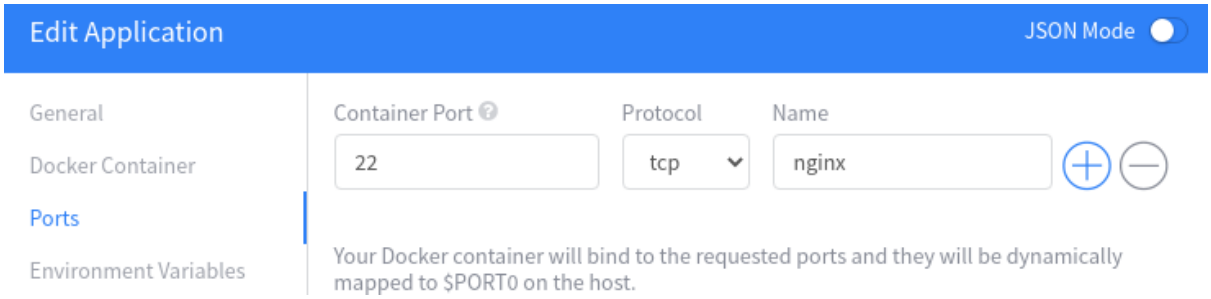

Fig. 4. Impostazioni per la porta da usare per la porta

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

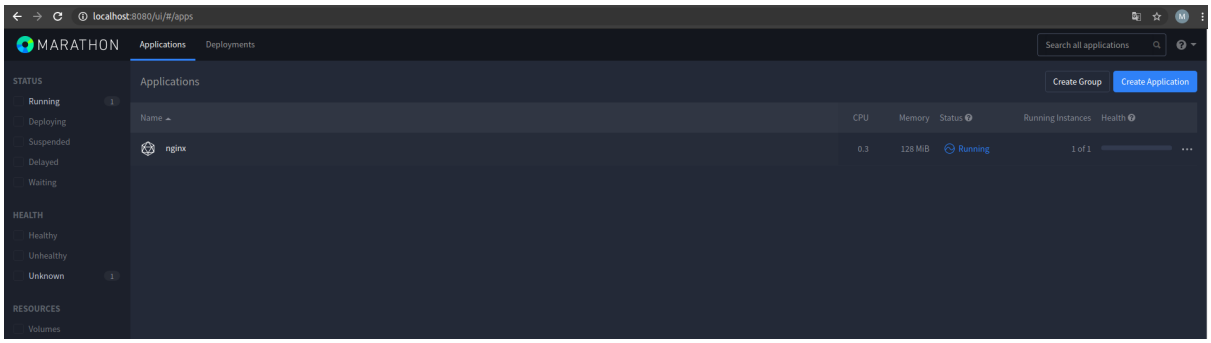

Fig. 5. Risultato finale.

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

configurazione del Docker Compose di Marathon/Mesos più stabile rispetto a quella utilizzata inizialmente.

La nuova versione del Docker compose è riportata nella Sezione 8.2 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

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:

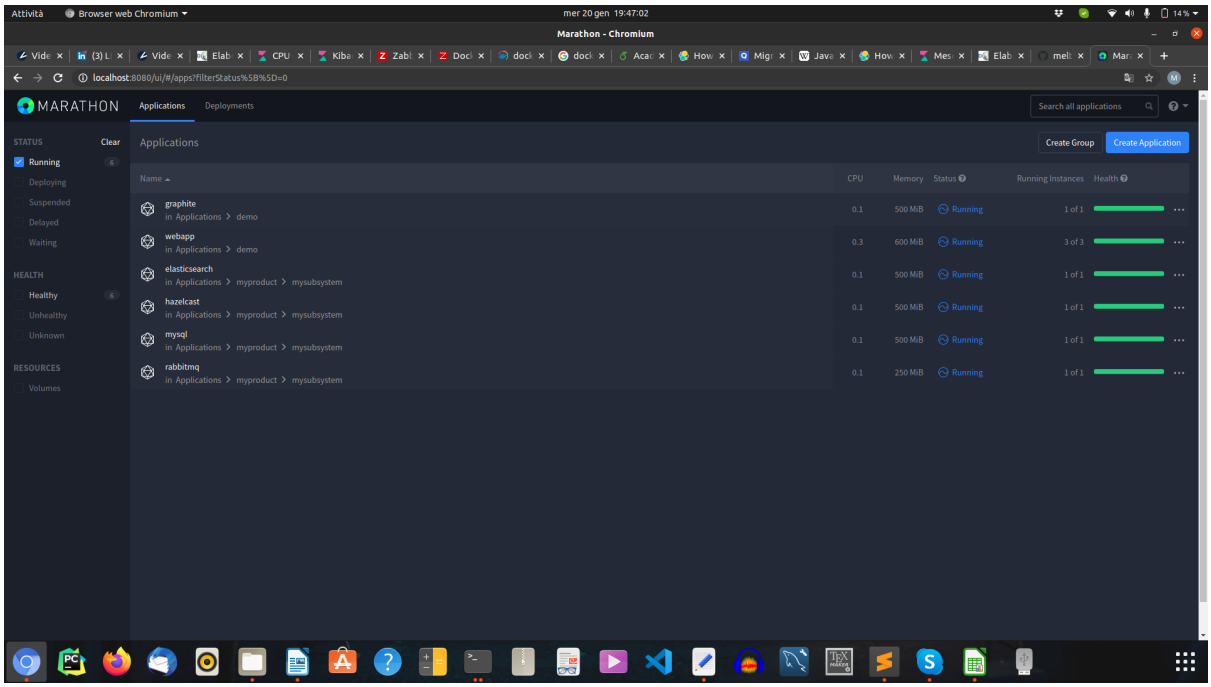

Fig. 7. Lista delle applicazioni avviate con il nuovo Docker Compose

- **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".

4.1 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:

$$mesos\text{-}<codice\text{-}mesos>\text{-}S0.<codice\text{-}instanza>,$$

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 che si da 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 preproccesing 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 e id 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.

5 Monitoraggio delle applicazioni

5.1 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.

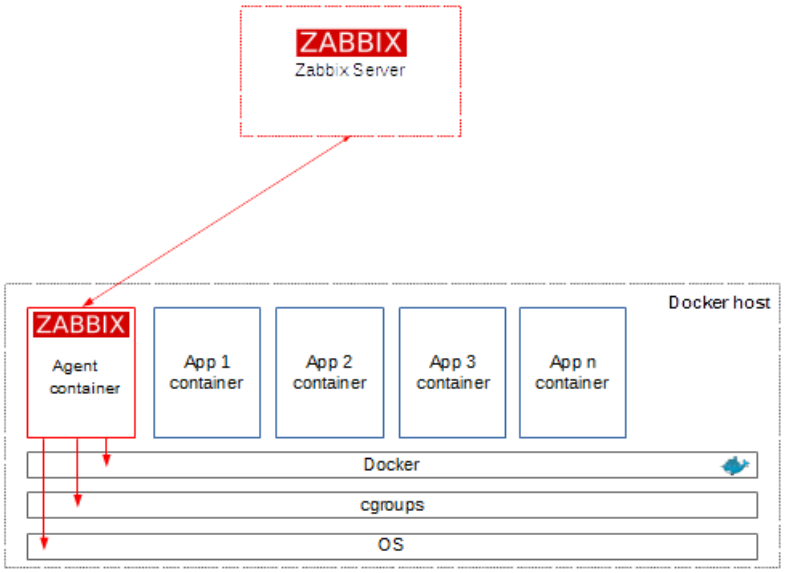

Fig. 8. Schema della configurazione dello Zabbix Agent su un host (VM o Docker)

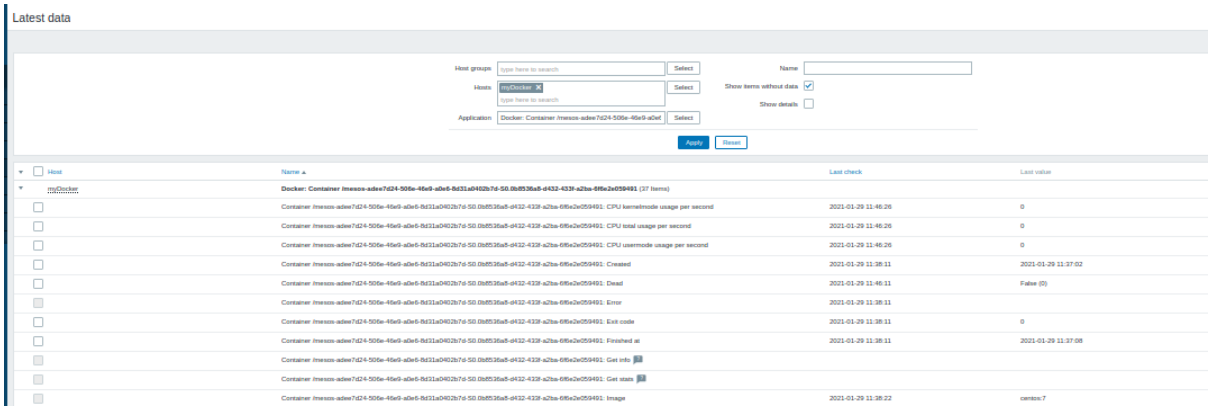

Fig. 9. Latest Data

history	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
history_uint	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
history_text	MarathonAppId
history_str	Docker: Name

Table 1
Tabelle delle metriche di Template App Docker

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.

5.2 Query Mysql

Come ho gia 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.

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

La vista **host** è utilizzata per recuperare le informazioni sull'agente Zabbix, come l'IP e e il nome dell'agente. L'agente viene installato su ogni macchina virtuale per monitorare i

container in esecuzione su quella macchina virtuale. In questo modo si può recuperare l'IP della macchina virtuale.

```
####CREAZIONE DELLA VISTA CHE CONTIENE LE INFORMAZIONI DELL'HOST ZABBIX####
create or replace view host as
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;
```

La vista **info** è utilizzata per recuperare i nomi assegnati dall'utente ai diversi container in esecuzione.

```
########CREAZIONE DELLA VISTA CHE CONTIENE IL NOME DEL HOST########
create or replace view info as
(select distinct SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 1), ':', -1)
   as name, h.value
from items i inner join  history_text h on h.itemid=i.itemid
where SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 2), ':', -1)=' MarathonAppId'
and h.value != 'unknown');
```

La vista **containerCPU_Net** è utilizzata per recuperare i dati sulla CPU e Network oltre al valore delle altre metriche memorizzate nella tabella *history*.

```
########VISTA CHE CONTIENE I NOMI DELLE APPLICAZIONI CREATE DA MARATHON########
#####NELLA VISTA SONO CONTENUTI INFORMAZIONI COME LA CPU, INIZIO/FINE DELLA VITA
#####DEL CONTAINER, METRICHE RELATIVE ALLA RETE
create or replace view containerCPU_Net as
select host.host as hostname, host.ip as host_ip, info.value as container,
SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, 'S',
   2), 'S', -1), ':', 1), ':', -1) as instance_id,
SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 2), ':', -1) as metrics,
from_unixtime(h.clock) as time, h.value
from items i,  history h, info, host
where i.itemid=h.itemid and i.hostid =host.hostid
and i.name regexp '^(Container /mesos)'
and SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 1), ':', -1)=info.name;
```

La vista **containerRAM** è utilizzata per recuperare i dati sulla memoria RAM oltre al valore delle altre metriche memorizzate nella tabella *history_uint*.

```
########VISTA CHE CONTIENE I NOMI DELLE APPLICAZIONI CREATE DA MARATHON########
#####NELLA VISTA SONO CONTENUTI INFORMAZIONI COME LA CPU, INIZIO/FINE DELLA VITA
#####DEL CONTAINER, METRICHE RELATIVE ALLA RETE
#####INFORMAZIONI SUI CONTAINER ATTIVI, FERMI E TOTALI DA DOCKER
create or replace view containerRAM as
```

```
select host.host as hostname, host.ip as host_ip,
IF(SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX(i.name,
  'S', 2), 'S', -1), ':', 1), ':', -1)='Docker','Docker',  info.value) as
  container,
SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, 'S',
  2), 'S', -1), ':', 1), ':', -1) as instance_id,
SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 2), ':', -1) as metrics,
from_unixtime(h.clock) as time, h.value
from items i, history_uint h, info, 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'
or (i.name regexp '^(Container /mesos)'
and SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 1), ':', -1)=info.name));
```

La vista **unionTable** unisce i valori delle metriche recuperate con la vista *containerCPU_Net* e quelle recuperate con la vista *containerRAM*. Questa vista consente di caricare i dati del monitoraggio da Mysql a Logstash con una singola query.

```
######CREAZIONE DELLA VISTA CHE UNISCE LE RIGHE DELLA TABELLA A CON QUELLA
######DELLA TABELLA B
create or replace view unionTable as select * from containerCPU_Net
union all select * from containerRAM;
```

La vista **finalData** permette di creare per ogni metrica una colonna dedicata in cui copiare i dati della metrica.

```
#####CREAZIONE DI UNA VISTA CHE CREA UNA COLONNA PER OGNI METRICA
#####CON IL VALORE CORRETTO DELLA METRICA PER CONTAINER(RIGA)
    create or replace view finalData as (
  select
    hostname, host_ip, host_dns, host_port,
    docker_name, container, instance_id, time, metrics, value,
    case when metrics = " Finished at " then value end as " Finished at",
    case when metrics = " Started at " then value end as " Started at",
    case when metrics = " CPU kernelmode usage per second "
    then value end as " CPU kernelmode usage per second",
    case when metrics = " Throttled time" then value end as " Throttled time",
    case when metrics = " CPU total usage per second "
    then value end as " CPU total usage per second",
    case when metrics = " CPU usermode usage per second "
    then value end as " CPU usermode usage per second",
    case when metrics = " Networks bytes received per second"
```

```
    then value end as " Networks bytes received per second",
    ............
    ............
    ............
    case when metrics = " Memory usage" then value end as " Memory usage",
  from unionTable
);

#####QUERY FINALE CHE RIDUCE IL NUMERO DI RIGE ATTRAVERSO LA GROUPBY
select * from finalData fd ;
```

La query precedente recupera i dati monitorati dal database Zabbix. Si recupera per ogni container Mesos tutti i valori di tutte le metriche definite nel template, il nome assegnato da Marathon all'applicazione che ha generato il container e le metriche globali del Docker Engine.

L'esecuzione della query precedente è lenta quando è eseguita su grandi volumi di dati. Abbiamo dunque pensato di utilizzare una strategia diversa che evita di ottenere una query enorme con tutti i dati di tutte le metriche. Per velocizzare l'esecuzione delle query abbiamo creato tante viste una per ogni metrica. Le viste create hanno tutte la seguente struttura:

```
create or replace view <metric_name> as
select host.host as hostname, host.ip as host_ip, info.value as container,
SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX(SUBSTRING_INDEX(i.name,
   'S', 2), 'S', -1), ':', 1), ':', -1) as instance_id,
SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 2), ':', -1) as metrics,
from_unixtime(h.clock) as time, h.value as value
from items i,  <table_metric_name> h, info, host
where i.name regexp '^(Container /mesos)'
and i.name regexp '( CPU kernelmode usage per second)$'
and SUBSTRING_INDEX(SUBSTRING_INDEX(i.name, ':', 1), ':', -1)=info.name
and i.itemid=h.itemid and i.hostid =host.hostid;
```

dove *metric_name* è il nome della metrica e *table_metric_name* è il nome della tabella Mysql dove sono memorizzati i dati delle metrica. Purtroppo questa soluzione non avvantaggia l'esecuzione degli algoritmi di anomaly detection presenti su Kibana e crea dei problemi nella creazione delle dashboard.

Per velocizzare l'esecuzione delle query abbiamo provato a definire degli indici per i campi delle tabelle MySQL che vengono richiamati all'interno della clausola WHERE della query. Le chiavi primarie delle tabelle hanno già un indice ad esse associato, gli unici campi sprovvisti di indici sono il campo *name* della tabella *item* e il campo *value* della tabella *history_text*:

```
CREATE INDEX index_name ON item(name);
CREATE INDEX index_name ON history_text(value(3072));
```

5.3 Tabelle dei dati

Le viste che vengono create dalle query della sottosezione **Query Mysql** hanno la seguente forma :

- hostname: indica il nome dell' host zabbix che ha fatto il rilevamento dati
- host_ip: indica l'indirizzo ipv4 dell'host zabbix
- time: indica il timestamp della rilevazione del host zabbix
- container: indica il nome dell'applicazione
- instanceID: indica il nome reale del container che inizia per mesos
- metrics: nome delle metrica rilevata da zabbix per il container
- value: valore della metrica

Al momento non sappiamo quale metriche siano utili perché non abbiamo ancora visualizzato i grafici dei monitorati da ogni metrica. Quando sapremo quali metriche monitorare, modificheremo la query per prendere le metriche necessarie eliminando la colonna metrics e values. Tuttavia abbiamo provato una rappresentazione alternativa che crea 33 query una per ogni singola metrica analizzata invece di un'unica query gigante. Le colonne metrics e value sono sostituite da tante colonne, una per ogni metrica. Ognuna di esse contiene i dati di quella metrica specifica.

6 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
- **elasticsearc** per salvare i dati
- **Kibana** per visualizzare e lanciare tools sui dati(es **Elastic Machine Learning**)

Questi servizi sono stati installati tutti con un docker compose dal nome **docker-compose-zabbix-kibana-logstasch.yml**.

6.1 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 questo schema proposto nella Figura 10. Inizialmente quello che facevamo era di chiedere a Logstash di eseguire periodicamente una query su Mysql e recuperare i nuovi dati. Questa query viene salvata all'interno del file **logstash.conf**.

Il file logstash.conf si compone di tre pipeline: una di input, una di filtraggio e una di output. Nella pipeline 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**. La query da far eseguire a Mysql è la stessa query che si trova nella Sezione

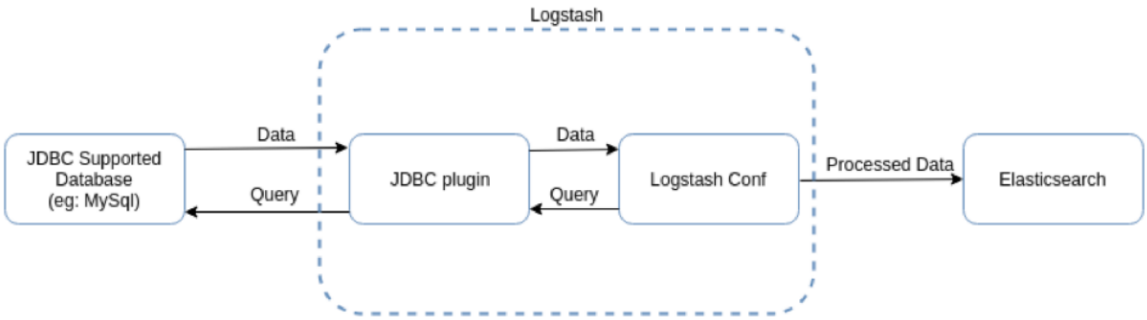

Fig. 10. Schema di interazione tra Mysql (con Jdbc), logstasch ed elastisearch

5.2 e viene eseguita periodicamente impostando lo scheduler con la sintassi **rufus**:"* * * * *", che consente di recuperare i dati da mysql ogni minuto. La pipeline di output invece si occupa di accogliere i dati migrati in un **type** (che in elasticsearch è l'equivalente della tabella) chiamato **mesos-marathon** e per ogni **documento** (riga) assegna un **document_id** univoco in modo da garantire unicità del documento. Si osservi infine che nella query si prendono 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**

Successivamente, a causa della lentezza nell'esecuzione di una singola query, abbiamo modificato il file di configurazione di Logstash per eseguire più di una query contemporaneamente. In particolare, abbiamo modificato la pipeline di input inserendo una configurazione del JDBC una per ogni query. Per distinguere il valori ottenuti da una query da quelli ottenuti da un'altra abbiamo utilizzato un indice Elasticsearch. La pipeline di output è stata modificata inserendo una configurazione diversa per ogni indice. Abbiamo inoltre aggiunto la pipeline filter per forzare il cast a float del campo value.

6.2 Kibana

Dopo aver caricato i dati su elasticsearch, abbiamo creato una semplice dashboard Figura 11 ancora in sviluppo. La 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 cloude 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

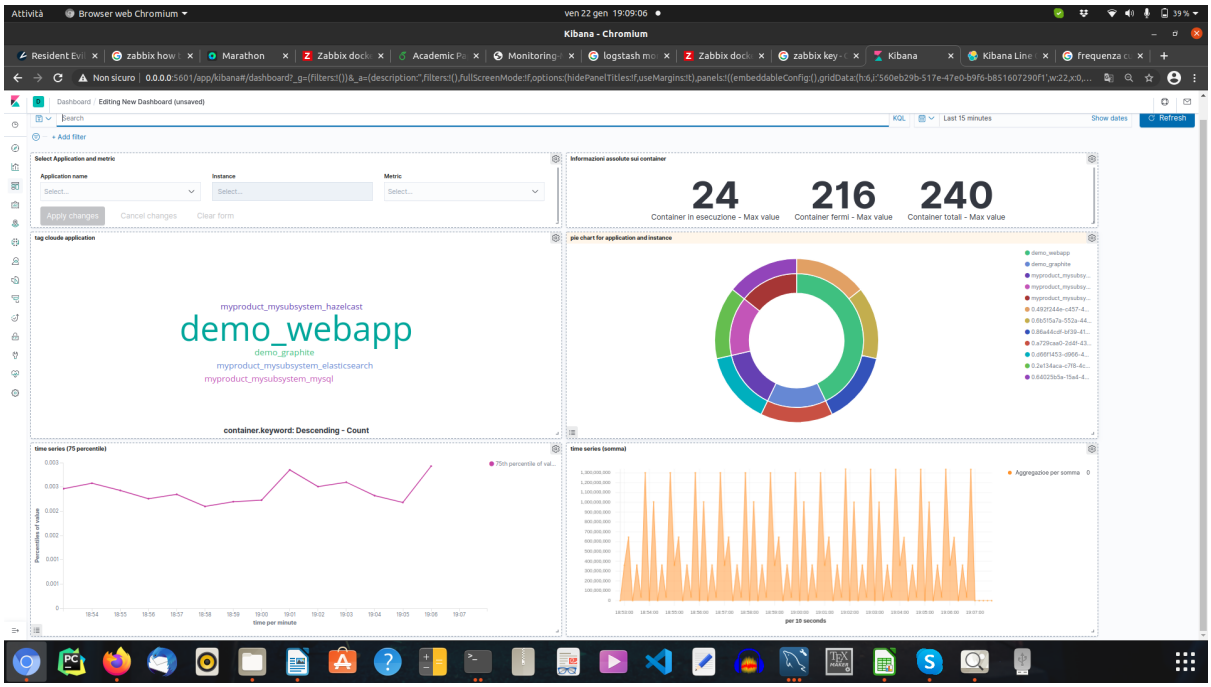

Fig. 11. Dashboard chiamata mesos-marathon-docker dashboard

6.3 Anomaly detection con Kibana

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 preso in analisi una singola metrica e il **multi job** dove viene presa più di una metrica per condurre una analisi multi-variata.

Per portarci avanti e prendere dimestichezza con la fase di analytics abbiamo usato come dati di prova la tabella che ha una colonna per ogni metrica, cioè quella ottenuta con una sola query.mAl momento il dataset di prova ha pochi dati e per entrambi i job non vengono rilevate anomalie.

Come abbiamo detto abbiamo provato anche a definire dataset diversi uno per ogni metrica. Questi sono stati ottenuti configurando Logstash per l'esecuzione di più di una query. Kibana non supporta più i tipi e per questo unisce tutti i dati provenienti da MySQL in un unico indice. Purtroppo non siamo ancora riusciti a filtrare i dati da Elastic Machine Learning e quindi non abbiamo ancora provato questo strumento sui dati ottenuti con la nuova metodologia di import.

7 Problemi risolti

Durante il progetto abbiamo dovuto trovare uno 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*.

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.

8 Appendice

8.1 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-0.27.1-2.0.226
  command: '/usr/bin/chronos run_jar --http_port 4400 --zk_hosts localhost:2181 --master zk
  net: host

chronossubmit:
  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'
```

8.2 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'
    environment:
      - MYSQL_ROOT_PASSWORD=carryontech
      - MYSQL_DATABASE=zabbix
      - MYSQL_USER=zabbix
      - MYSQL_PASSWORD=carryontech

  zabbix-server:
    container_name: zabbix-server
    image: zabbix/zabbix-server-mysql:ubuntu-5.0.1
    networks:
      - bda_project
    links:
      - mysql
      - elasticsearch:elasticsearch
    depends_on:
      - elasticsearch
    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/zabbix_server.conf:/etc/zabbix/zabbix_server.conf'
#      - ./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
      - elasticsearch:elasticsearch
    depends_on:
      - elasticsearch
    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:/var/www/html/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

elasticsearch:
  container_name: elasticsearch
  image: elasticsearch:7.5.0
  networks:
    bda_project:
      ipv4_address: 172.16.121.11
  environment:
    - bootstrap.memory_lock=true
    - discovery.type=single-node
    - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
  ulimits:
    memlock:
      soft: -1
      hard: -1
  ports:
    - 9200:9200
    - 9300:9300
  stdin_open: true
  tty: true
  logging:
    driver: "json-file"
```

```
    options:
      max-size: "10m"
      max-file: "50"

kibana:
  container_name: kibana
  image: kibana:7.5.0
  networks:
    bda_project:
      ipv4_address: 172.16.121.12
  ulimits:
    memlock:
      soft: -1
      hard: -1
  ports:
    - 5601:5601
  links:
    - elasticsearch:elasticsearch
  depends_on:
    - elasticsearch


logstash:
  container_name: logstash
  privileged: true
  networks:
    bda_project:
      ipv4_address: 172.16.121.13

  build:
    dockerfile: ${PWD}/Dockerfile
    context: ${PWD}/
  environment:
    - LOGSTASH_JDBC_URL=jdbc:mysql://mysql:3306/zabbix?useSSL=false
    - LOGSTASH_JDBC_DRIVER=com.mysql.cj.jdbc.Driver
    - LOGSTASH_JDBC_DRIVER_JAR_LOCATION=/usr/share/logstash/logstash-core/lib/jars/mysql-c
    - LOGSTASH_JDBC_USERNAME=zabbix
    - LOGSTASH_JDBC_PASSWORD=carryontech
    - LOGSTASH_ELASTICSEARCH_HOST=http://elasticsearch:9200
  volumes:
    #- ./mysql-jdbc-input-plugin.conf:/usr/share/logstash/pipeline/logstash.conf
```

```
      - ./logstash/logstash.conf:/usr/share/logstash/pipeline/logstash.conf
    ports:
      - 9600:9600
      - 5044:5044

    depends_on:
      - elasticsearch
      - kibana
      - mysql
    links:
      - elasticsearch:elasticsearch
      - mysql:mysql
      - kibana:kibana

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

8.3 Docker file per logstasch

```
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
```

8.4 Configurazione file logstash.conf per single query

```
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 => "* * * * *"
    # our query
        statement => " select * from (query Sezione 5.2) as query
           where (time > :sql_last_value AND time < NOW()) order by time asc "
        use_column_value => true
        tracking_column => time
        tracking_column_type => "timestamp"
    }
  }
output {
  stdout { codec => json_lines }
  elasticsearch {
    hosts => "elasticsearch:9200"
    index => "mysql-migrate"
    document_id => "%{container} %{metrics} %{time}"
    document_type => "zabbixdata"
  }
}
```

8.5 Configurazione file logstash.conf per multi-query

```
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 => "* * * * *"
# our query
    statement => "select * from Finished_at where (time > :sql_last_value
       AND time < NOW()) order by time asc"
    tracking_column => time
    tracking_column_type => "timestamp"
    use_column_value => true
    type => "finished_at"
    last_run_metadata_path => "/home/finished_at"
}

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 * from Started_at where  (time > :sql_last_value
       AND time < NOW()) order by time asc"
    tracking_column => time
    tracking_column_type => "timestamp"
    use_column_value => true
    type => "started_at"
    last_run_metadata_path => "/home/started_at"
}

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/cpu_kernelmode_usage_per_second"
    # our query
        statement => "select * from CPU_kernelmode_usage_per_second where
           (time > :sql_last_value AND time < NOW()) order by time asc"
        tracking_column => time
        tracking_column_type => "timestamp"
        use_column_value => true
        type => "cpu_kernelmode_usage_per_second"
        last_run_metadata_path => "/home/cpu_kernelmode_usage_per_second"
    }

.....

    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 * from DockerTotal where (time > :sql_last_value
           AND time < NOW()) order by time asc"
        tracking_column => time
        tracking_column_type => "timestamp"
        use_column_value => true
        type => "dockertotal"
        last_run_metadata_path => "/home/dockertotal"
    }
  }

  filter {
      mutate {
        convert => {
          "value" => "float"
           }
      }
    }
```

```
output {
   if[type] == "finished_at" {
        elasticsearch {
            hosts => "elasticsearch:9200"
            index => "migrate_finished_at"
            document_id => "%{container} %{metrics} %{time}"
        }
    }
    if[type] == "started_at" {
        elasticsearch {
            hosts => "elasticsearch:9200"
            index => "migrate_started_at"
            document_id => "%{container} %{metrics} %{time}"
        }
    }
    if[type] == "cpu_kernelmode_usage_per_second" {
        elasticsearch {
            hosts => "elasticsearch:9200"
            index => "migrate_cpu_kernelmode_usage_per_second"
            document_id => "%{container} %{metrics} %{time}"
        }
    }

....

    if[type] == "dockertotal" {
        elasticsearch {
            hosts => "elasticsearch:9200"
            index => "migrate_dockertotal"
            document_id => "%{container} %{metrics} %{time}"
        }
    }
    stdout {
        codec => json_lines
    }

}
```

9 Zabbix gallery

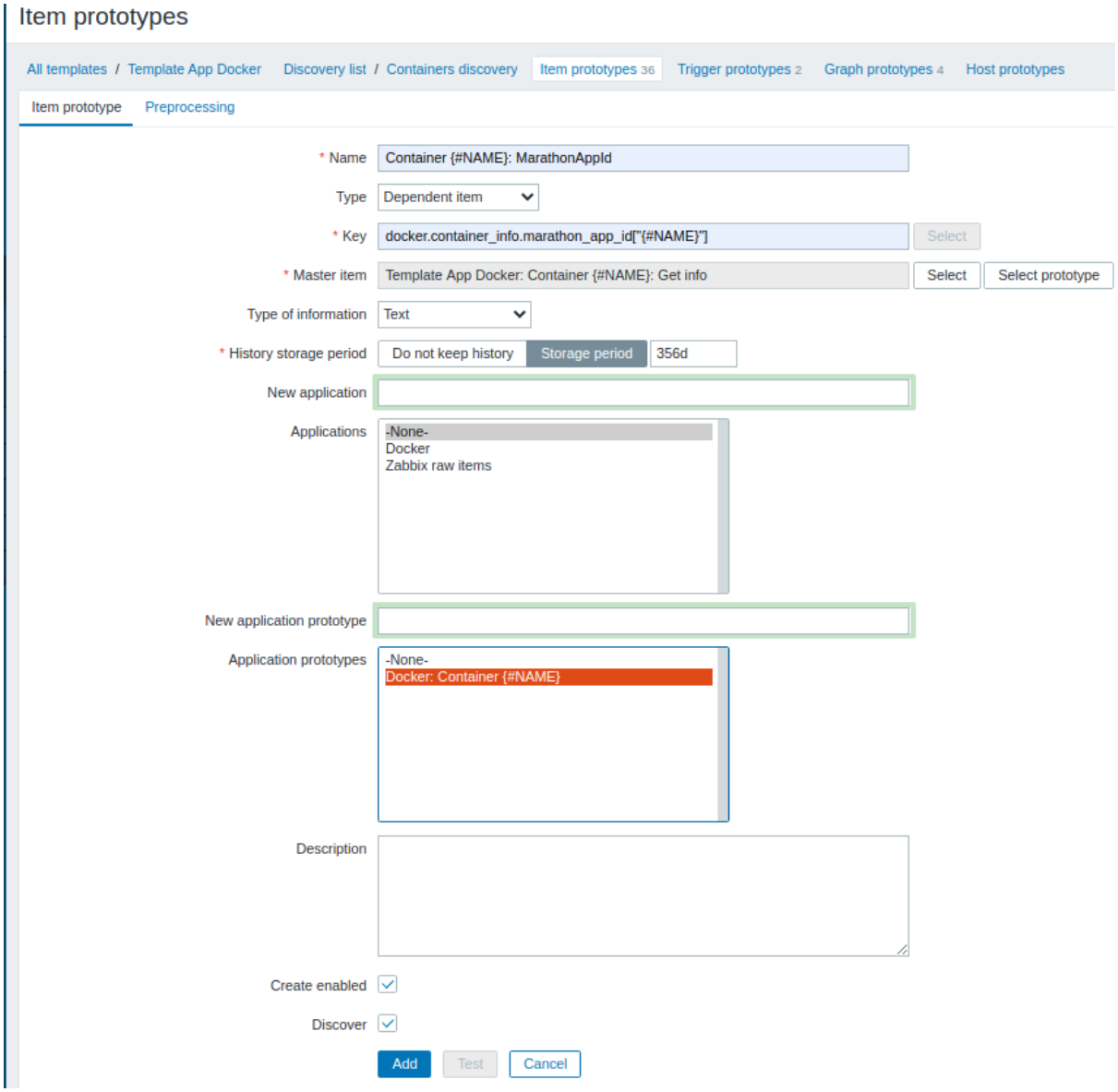

Fig. 12. Configurazione del prototype item per recuperare il marathon app id.

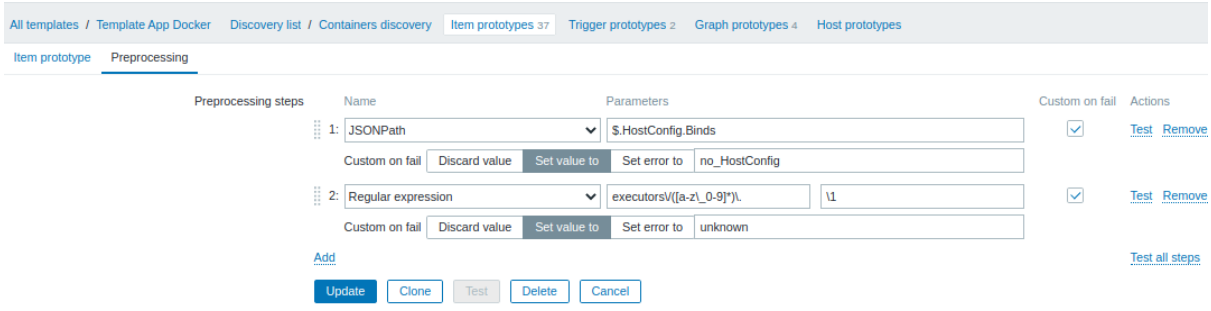

Fig. 13. Configurazione dei passi di pre processing per recuperare il marathon app id con proto item personalizzato.

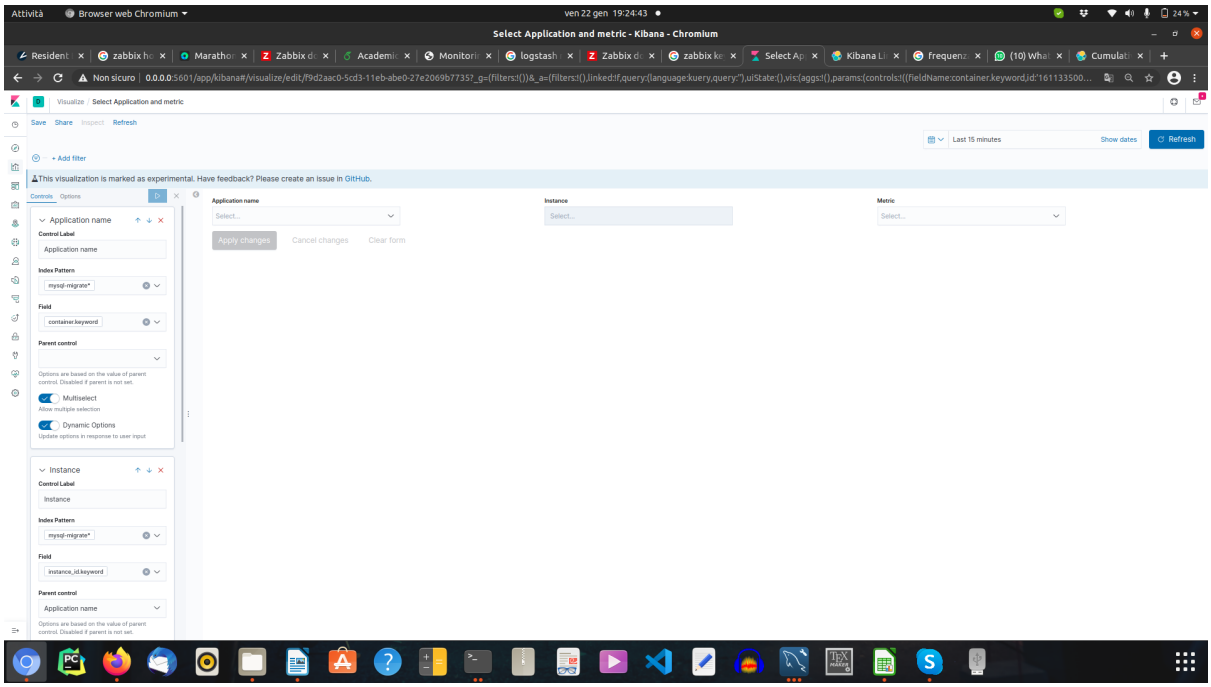

Fig. 14. Selettore dell'applicazione e delle metrica

10 Kibana gallery

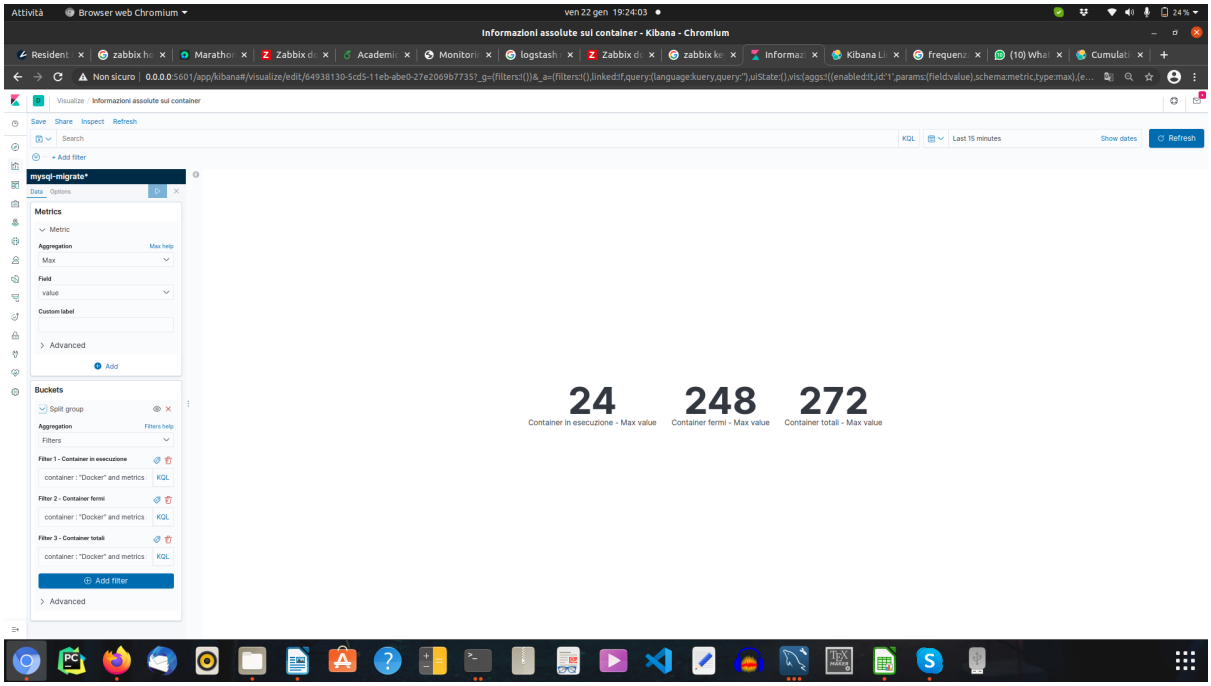

Fig. 15. Visualizzatore dei container attivi, stoppati e totali

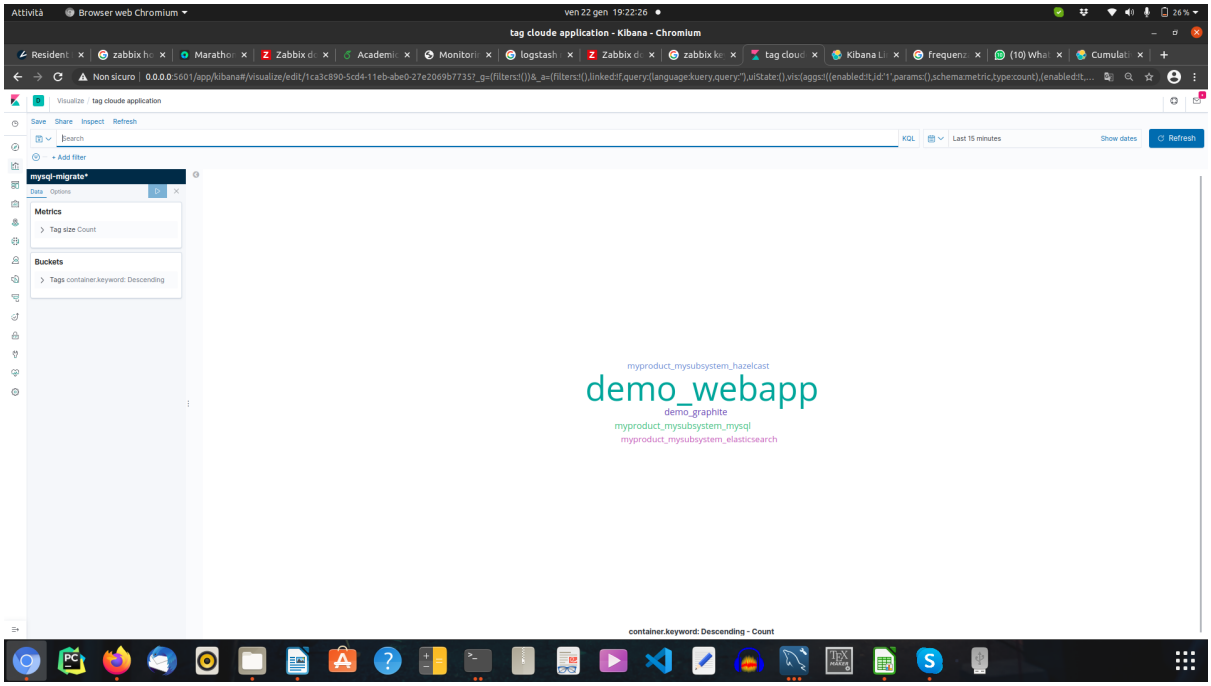

Fig. 16. Tag cloude per visualizzare le applicazioni

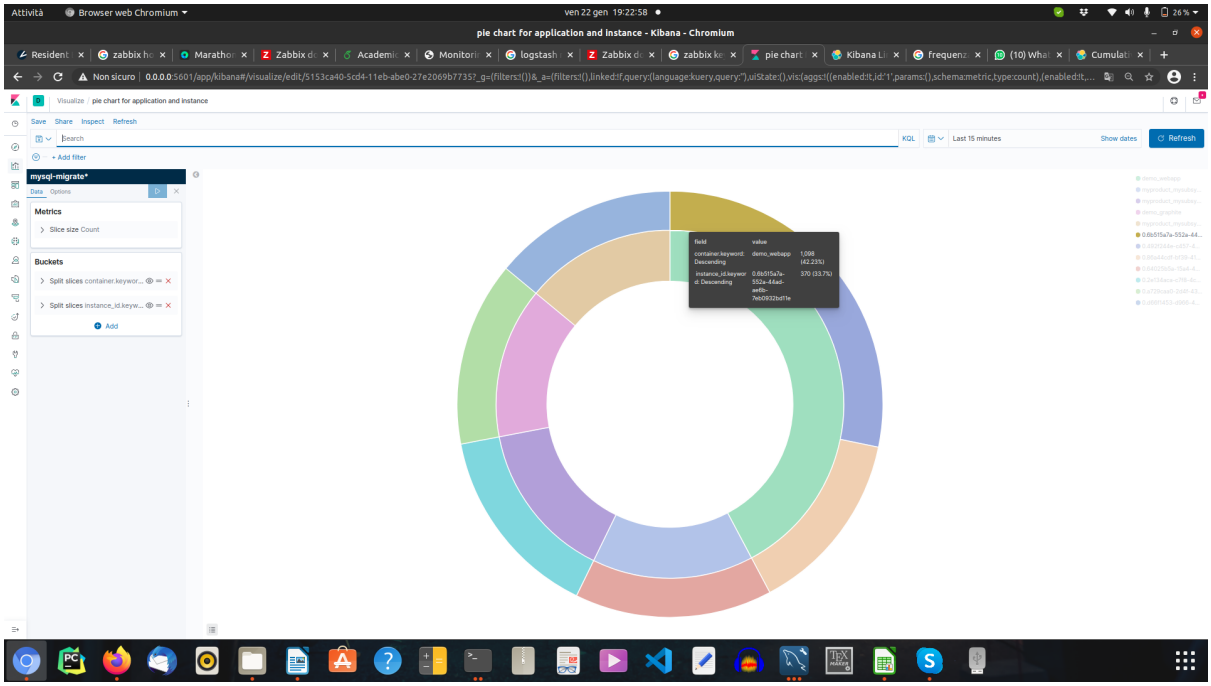

Fig. 17. Piechart che mostrano i container in esecuzione che eseguono le diverse applicazioni

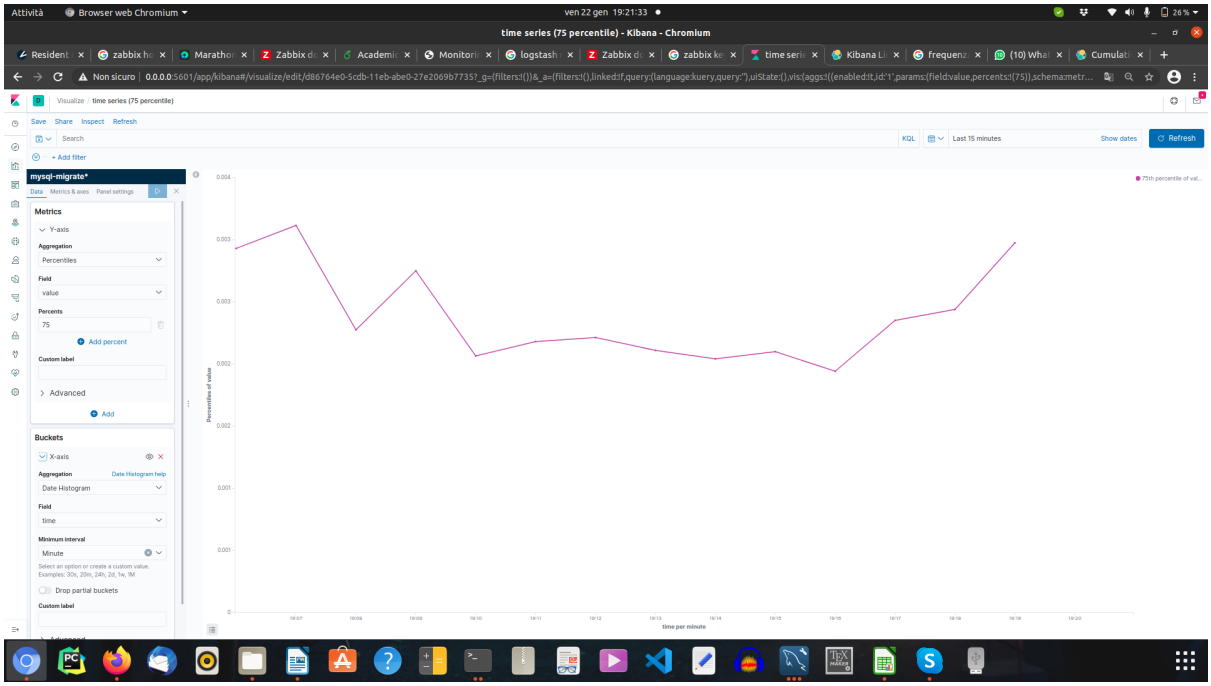

Fig. 18. Time series che plotta il 75-percentile per il container e metrica scelta

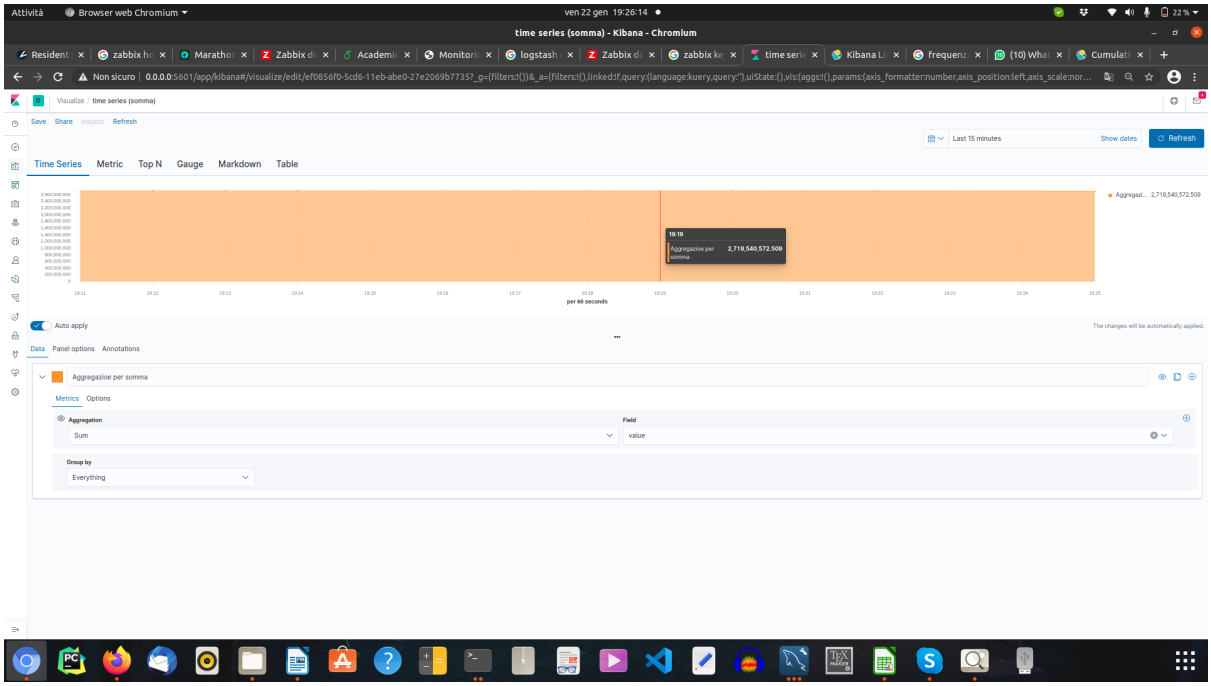

Fig. 19. Time series aggregata per somma per il container e metrica scelta