Disaster Monitor

Territori Aperti: Nuove tecnologie per la raccolta, la preparazione e l'analisi dei dati.

Dott. Roberto Felicioni
Resp. Scientifico Prof. Giovanni Stilo

Giugno 2022

Parte I
Analisi e Design

1 Disaster Monitor Application

Il borsista Roberto Felicioni si è occupato inizialmente di una fase esplorativa, con l'obiettivo di realizzare un'applicazione per il monitoraggio dei flussi sociali relativi ai disastri naturali, chiamato *Disaster Monitor*.

L'idea alla base è stata quella di creare un monitor (tramite web app) per la visualizzazione di dati relativi ai disastri naturali, come ad esempio è stato fatto dalla John Hopkins University rispetto alla recente pandemia da Sars-CoV-2[1].

Come sorgente di dati, lo studio si è focalizzato sui messaggi che gli utenti scrivono sui social network. Quindi è stata considerata come sorgente il social network Twitter, ma l'analisi potrà essere estesa anche ad altri social network e ulteriori sorgenti di dati (e.g. Open Data).

Successivamente sono state quindi delineate le funzionalità e le componenti che l'applicazione dovrà esporre ed avere. In sintesi, l'applicazione esporrà le seguenti funzionalità: deve essere possibile specificare un insieme definito (ma potenzialmente mutevole) di parole chiave. Tale insieme di parole definisce un topic di interesse e sarà usato per collezionare in tempo reale il flusso dei dati relativo ad esso. Il flusso di dati viene quindi salvato sia nel suo formato originale, sia preprocessato in un formato che ne permetta un'agevole interrogazione. Partendo da questo secondo formato, con opportune interrogazioni ed elaborazioni, verranno prodotte delle viste di dettaglio utili ad avere una visione accurata dell'evento che si sta verificando.
Una successiva prima analisi ha portato quindi ad indentificare tre componenti principali.

- La prima deve essere in grado di collezionare dati da diverse sorgenti (ad esempio le API di Twitter) e deve permettere di predisporre diversi flussi di analisi: ad esempio un flusso che si occupi di memorizzare i dati nella loro forma orginale mentre un altro che potrà effettuare un primo processing e memorizzare i dati in un datastore evoluto.

- Al centro della seconda componente, che si occuperà dello storage dei dati permettendone una facile interrogazione, deve essere presente un datastore evoluto che esponga funzionalità di interrogazione. Tale tecnologia sarà alla base della logica applicativa dell'intera applicazione che verrà sviluppata in seguito.
- La terza componente si occuperà di realizzare le visualizzazioni grafiche. Tali visualizzazioni dovranno essere interattive e correlate. Vale a dire che un evento avrà diverse viste ed indicatori che ne permettano la comprensione.

Dal punto di vista tecnico il primo passo per la realizzazione dell'applicazione è stato quello di fare una ricognizione delle tecnologie esistenti utili alla sua realizzazione. Tali tecnologie devono soddisfare in primis i seguenti vincoli:

i) la licenza di uso deve essere open source;

ii) deve essere possibile fare il deployment di tali tecnologie su un server locale e quindi non includere tecnologie e servizi cloud.

1.1 Tecnologie per la Raccolta dati

Per la raccolta di dati, sono state prese in considerazione diverse tecnologie.

La prima è stata **Apache Flume**[2], un servizio distribuito per la raccolta, l'aggregazione e il trasporto di dati. Il suo punto di forza è la sua architettura, che lo rende adatto alla gestione di flussi di dati in modo complesso e modulare (pipeline). Però sembra che questa tecnologia sia stata accantonata dalla fondazione che si occupa del suo sviluppo in quanto l'ultima versione disponibile, la 1.9.0, è stata rilasciata oltre tre anni fa. Con questo tool è risultato, inoltre, più impegnativo del previsto agganciare le API di Twitter. Pertanto si è deciso di valutare ulteriori alternative.

La seconda tecnologia presa in considerazione è stata **Apache Flink**[3], uno strumento capace di intercettare flussi di eventi, preprocessarli e inviarli a diverse destinazioni per successive elaborazioni. Apache Flink è manutenuto con costanza dalla fondazione Apache (l'ultima versione è stata rilasciata a marzo 2022) ed è parte fondante di diversi progetti alla gestione dei big data. Anche se la piattaforma risulta altamente estensibile, la maggiore criticità risiede nelle risorse e nelle conoscenze che richiede per essere padroneggiata. Visti i tempi ristretti del progetto, si è pertanto proceduto a valutare ulteriori tecnologie.

La terza e ultima tecnologia considerata per questa parte di progetto è stata **Logstash**[4]. Questo è uno strumento che permette di raccogliere dati da una varietà di fonti, trasformarli contestualmente in memoria e inviarli a una o più destinazioni. Questa tecnologia è altamente fruibile, offrendo una buona integrazione con altre tecnologie che potranno essere scelte per lo storage e il processamento. Le iniziali difficoltà che sono emerse nella corretta gestione dei flussi sono state successivamente superate e pertanto siamo confidenti che tale tecnologia possa essere utilizzata efficacemente, data la sua grande flessibilità, per questa fase del progetto.

1.2 Tecnologie per lo storage e il processamento

Per lo storage e il processamento, si è proceduto a valutare la tecnologia offerta da **Elasticsearch**[5]. Tale tecnologia è composta da un servizio distribuito di ricerca e analisi alta-

mente scalabile. Inoltre Elasticsearch è open source e permette un'efficiente indicizzazione del dato. Elasticsearch risulta flessibile ed integrabile con le altre tecnologie dell'ecosistema big data. Considerate il tipo di interrogazioni che verranno fatte dalla logica applicativa dell'applicazione in fase di visualizzazione, le funzionalità offerte da Elasticsearch risultano essere preferibili a quelle offerte da un semplice database management system tradizionale. Pertanto, abbiamo ritenuto che tale tecnologia sia ottimale per sviluppare questa fase del progetto.

1.3 Tecnologie per la Visualizzazione

Per la scelta della componente da usare ai fini di visualizzazione sono state considerate due tecnologie.

La prima, **Kibana**[6], è stata considerata perché, insieme a Logstash ed Elasticsearch, costituisce un eco sistema completo di strumenti chiamato Elastic Stack; è quindi molto semplice l'intergazione con le tecnologie scelte per realizzare le prime due fasi. Questo punto di forza è però anche una debolezza: infatti Kibana risulta essere uno strumento poco felssibile dal punto di vista della possibile integrazione con altre tecnologie. Ad esempio: se in una fase successiva dello sviluppo si decidesse di sostituire Elasticsearch con uno strumento differente, risulterebbe conseguentemente più impegnativo usare Kibana. E la parte di visualizzazione dovrebbe essere pertanto adattata in maniera consistente o nuovamente realizzata.

Il secondo strumento considerato è stato **Grafana**[7] che, come Kibana, è un'applicazione per la visualizzazione e l'analisi interattiva dei dati. Questa tecnologia, rispetto a Kibana, risulta essere più potente, flessibile e meno vincolata ad una specifica componente di storage e processamento. Di conseguenza si deciso di preferire Grafana per realizzare la parte di visualizzazione.

1.4 Architertura

La figura 1 fornisce una vista di insieme dell'architettura dell'applicazione che si è delineata. Nello specifico, Logstash si occupa del data gathering ed è quindi usato per intercettare il flusso di dati in ingresso, relativo ad un determinato tipo di evento (specificato tramite una collezione di parole). Tale flusso è salvato nel suo formato originale ed è inviato alla componente di storage e indexing costituita da Elasticsearch. La seconda componente, costituita da Elasticsearch, si occupa di costruire un indice a partire dal flusso di dati ricevuto ed espone un'interfaccia per l'interrogazione puntuale dei dati. La parte di visualizzazione è realizzata tramite Grafana e funge da elemento intermediario fra l'interfaccia che ElasticSearch fornisce per interrogare l'indice, da un lato, e l'utente che interagisce con l'applicativo tramite le opportune viste comprese in una dashboard, dall'altro lato.

1.5 Sviluppo

In riferimento alle tecnologie scelte per lo sviluppo del *Disaster Monitor*: Logstash, Elasticsearch e Grafana si è proceduto ad effettuare una prima fase di testing, deployment

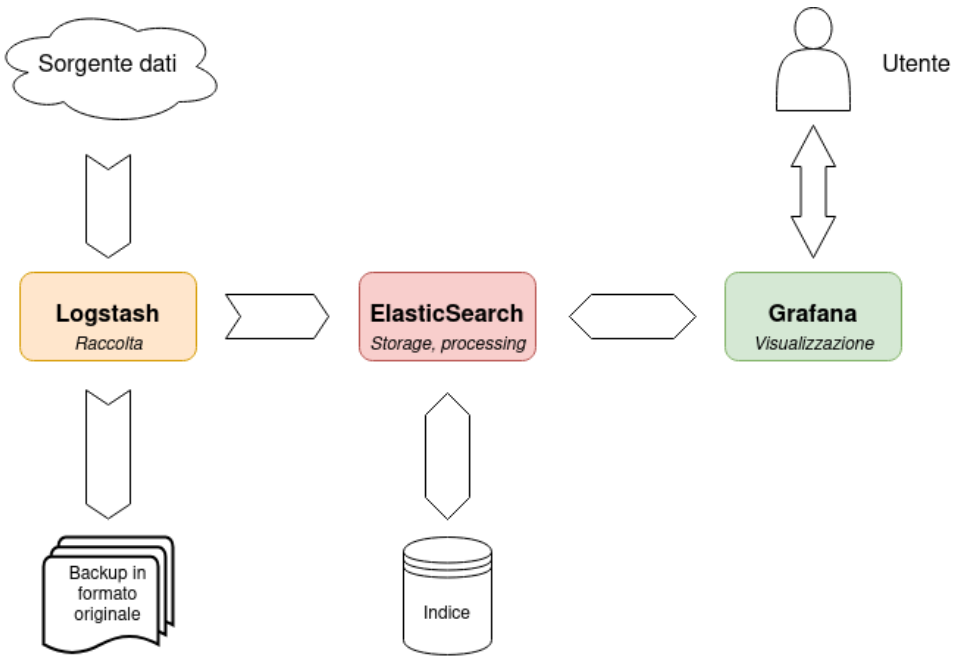

Figura 1: Diagramma di rappresentazione del flusso dei dati tra le varie componenti dell'applicazione web

e configurazione delle stesse. In particolare sono state effettuale le seguenti operazioni di deployment basate su Docker[8]:

- installazione dell'ambiente Docker;
- scaricamento installazione e configurazione dell'immagine docker di Logstash (versione 7.17.0);
- scaricamento, installazione e configurazione dell'immagine docker di Elasticsearch (versione 7.17.0);
- scaricamento, installazione e configurazione dell'immagine docker di Grafana (versione OSS 8.2.0);
- configurazione dei flussi per il collezionamento e salvataggio della sorgente tweet su filessystem locale;
- configurazione dei flussi per il collezionamento e salvataggio della sorgente tweet su in ElasticSearch;
- configurazione dell'interfaccia di base tramite Grafana per l'interrogazione di ElasticSearch, sono state effettuate tre viste di test;

Attualmente è stato così possibile realizzare un'applicazione multi contenitore attraverso la quale, tramite i file di configurazione, si sono realizzati con successo i collegamenti fra le diverse componenti in modo da collezionare i dati, salvarli, renderli fruibili e realizzare delle prime visualizzazioni personalizzate.

Parte II
Implementazione

1.6 Architettura

Una volta realizzata l'applicazione come nel punto 1.5, si è valutato se fosse possibile sviluppare la logica applicativa usando le funzionalità esistenti di Elasticsearch e Grafana, con esito negativo. Nel dettaglio, non si è trovato un modo semplice per ottenere, data una finestra temporale definita, l'insieme delle parole più significative in quella finestra, dove per significative si intende che ad ogni parola viene assegnato un punteggio (noi abbiamo usato il tf-idf) e si selezionano quelle parole che hanno un punteggio maggiore.

Si è pensato quindi di inserire nell'architettura un nuovo componente, in grado di gestire quella parte di logica applicativa che non è possibile eseguire (perlomeno in modo semplice) usando esclusivamente Elasticsearch e Grafana.

Questa nuova componente sarà scritta nel linguaggio Python e consisterà in un web server capace di interrogare Elasticsearch, elaborarne le risposte ed esporre delle REST API, attraverso le quali Grafana potrà avere accesso a queste elaborazioni.

Python è stato scelto perché, da un lato, ha una sintassi semplice e chiara; dall'altro, sono disponibili una grande quantità di librerie e framework che ne permettono un'estensione enorme. Per questi motivi, ci è sembrata la scelta più sensata da fare.

Nella figura 2 è mostrata l'architettura finale, dopo l'aggiunta di questo ulteriore componente.

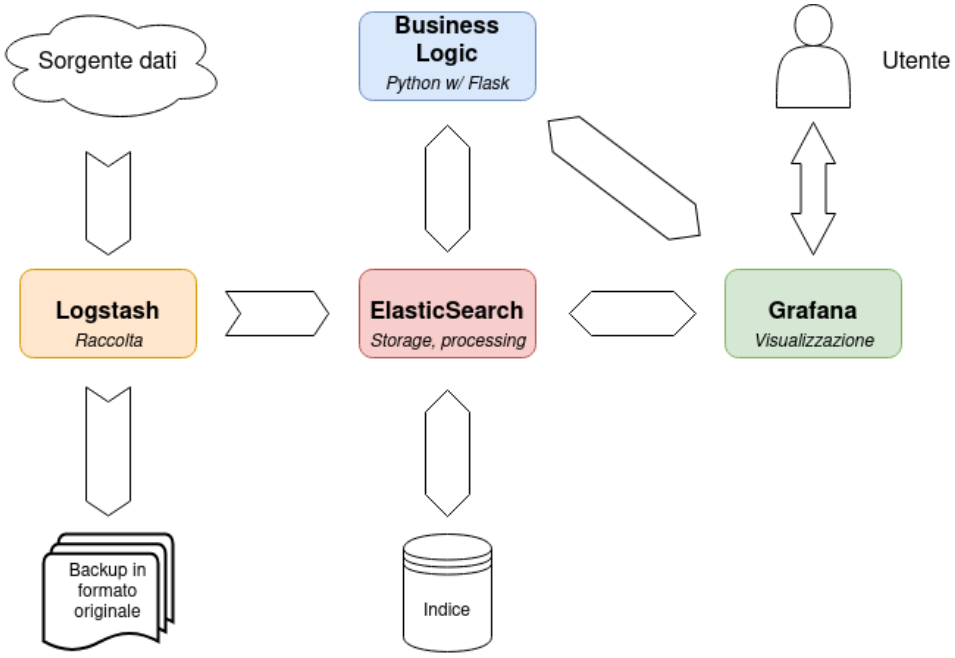

Figura 2: Diagramma aggiornato di rappresentazione del flusso dei dati tra le varie componenti dell'applicazione web

Nel seguito, descriveremo con un livello di dettaglio leggermente superiore come è configurata e cosa fa ognuna delle componenti.

1.7 Infrastruttura (Docker)

Docker[8] è un software che offre la possibilità di creare applicazioni in modo rapido, raccogliendo i vari componenti software che compongono l'applicazione in unità standardizzate chiamate container.

Docker non è parte dell'architettura e non sarà usato nella versione finale della web application. Abbiamo scelto di usarlo durante lo sviluppo dell'applicazione per una serie di vantaggi:

- separa l'applicazione in sviluppo dal resto dei programmi in esecuzione sul computer;
- permette di avviare l'intera applicazione semplicemente lanciando docker con un opportuno file di configurazione;
- bastano poche modifiche del file di configurazione per aggiungere un container o per aggiornarlo a una versione più recente;
- Logstash, Elasticsearch e Grafana hanno tutti e tre dei container ufficiali, aggiornati di frequente e disponibili nel Docker Hub Container Image Library[9];
- ogni container ha al suo interno tutto il necessario per il corretto funzionamento del software che ospita, quindi non è necessario preoccuparsi delle loro dipendenze software.

Attraverso Docker Compose (un tool di Docker che permette la costruzione e l'esecuzione di applicazioni multi container) è possibile eseguire insieme questi tre container.

L'unico elemento descritto nell'architettura che non viene eseguito attraverso docker è la componente di business logic scritta in Python.

1.8 Persistenza e Ricerca (Logstash ed Elasticsearch)

Logstash[4] è uno strumento che permette di raccogliere dati da diverse fonti (nel nostro caso, al momento ci siamo focalizzati su Twitter), li elabora al volo e li invia a una o più destinazioni.

Il file di configurazione di Logstash si divide in tre parti: input, filter e output. Queste tre parti hanno dei nomi autoesplicativi e, nel caso della nostra applicazione, hanno il seguente contenuto:

- **input** usa il plugin Twitter già presente in Logstash; al suo interno sono inserite le credenziali per accedere alle API di Twitter, insieme alla lista delle parole sulle quali Logstash si metterà in ascolto e al formato desiderato (nel nostro caso: JSON);
- **filter** contiene la componente di elaborazione dell'input che logastash offre; nel nostro caso, si limita a duplicare il flusso di input per la fase di output;
- **output** salva su file uno dei due flussi (in un formato compresso, nell'architettura è indicato come "backup in formato originale") e manda ad Elasticsearch l'altro dei due flussi, andando a popolare un indice indicato da noi.

Elasticsearch[5] gestisce l'indice che contiene i messaggi che gli arrivano da Logstash e fornisce a Grafana e al server web in Python una API attraverso la quale è possibile interrogare l'indice.

1.9 Business logic (Python e Flask)

Questa componente sarà scritta in Python, usando il micro-framework Flask[10]. Attraverso Flask è possibile realizzare un web server che espone delle REST API, le quali possono essere interrogate da Grafana. La scelta di realizzare un web server con Flask è stata dettata dalla volontà, da un lato, di usare Python; dall'altro, di avere la possibilità di esporre delle REST API. Flask è leggero, scalabile, semplice da usare ed è stato quindi considerato migliore delle alternative, come ad esempio Django.

Al momento, gli entry point delle REST API sono due. Il primo restituisce i dieci messaggi più importanti in un certo intervallo temporale; il secondo fornisce alcuni cluster di parole.

Questi cluster sono il risultato della seguente elaborazione, sviluppata in più punti:

- si seleziona un certo intervallo temporale (durante lo sviluppo si è utilizzato un intervallo di 10 minuti, ma in produzione si pensa di usare finestra di 10 giorni) e si chiede ad Elasticsearch l'elenco delle 10000 parole più usate in questo intervallo.
- si divide la finestra in 10 intervalli di uguale durata (nel caso la finestra sia di 10 minuti, gli intervalli sono da 1 minuto) e, tramite una query ad Elasticsearch, per ogni parola si ottiene il numero di occorrenze per ogni suddivisione della finestra, ottenendo di fatto per ogni parola una serie temporale.
- a questo punto si calcola, per ogni parola, il tf-idf ottenuto usando come term frequency il numero di occorrenze nella suddivisione (1 minuto) e come document frequency il numero di occorrenze nell'intervallo (10 minuti). Per ogni parola si usa la suddivisione che contiene il numero maggiore di occorrenze.
- si selezionano quindi le 1000 parole con il tf-idf più elevato
- su questa selezione, si rappresentano le serie temporali usando SAX e si selezionano quelle parole che seguono un pattern di collective attention: (a+b?bb?a+)?(a+b?bba*)?
- sulle parole che rimangono, viene effettuato un k-clustering sulla edit distance della rappresentazione SAX delle serie temporali delle parole. L'idea è quella di raggruppare nello stesso cluster quelle parole che hanno un simile pattern nel tempo.

Questa operazione non è effettuata in modo continuo (al momento il codice viene lanciato a mano, nella versione finale l'esecuzione avverrà verosimilmente una volta al giorno, in modo automatico).

Il risultato di questa elaborazione è un numero variabile di cluster, ciascuno contenente un insieme di parole.

1.10 Interfaccia Grafica (Grafana)

Grafana[7] è il componente che fornisce la visualizzazione delle informazioni all'utente, attraverso delle infografiche costruite a partire da sorgenti eterogenee di dati (nel nostro caso, le fonti saranno Elasticsearch e la componente in Python). Nella figura 3 è mostrato come si presenta la dashboard principale della nostra applicazione web.

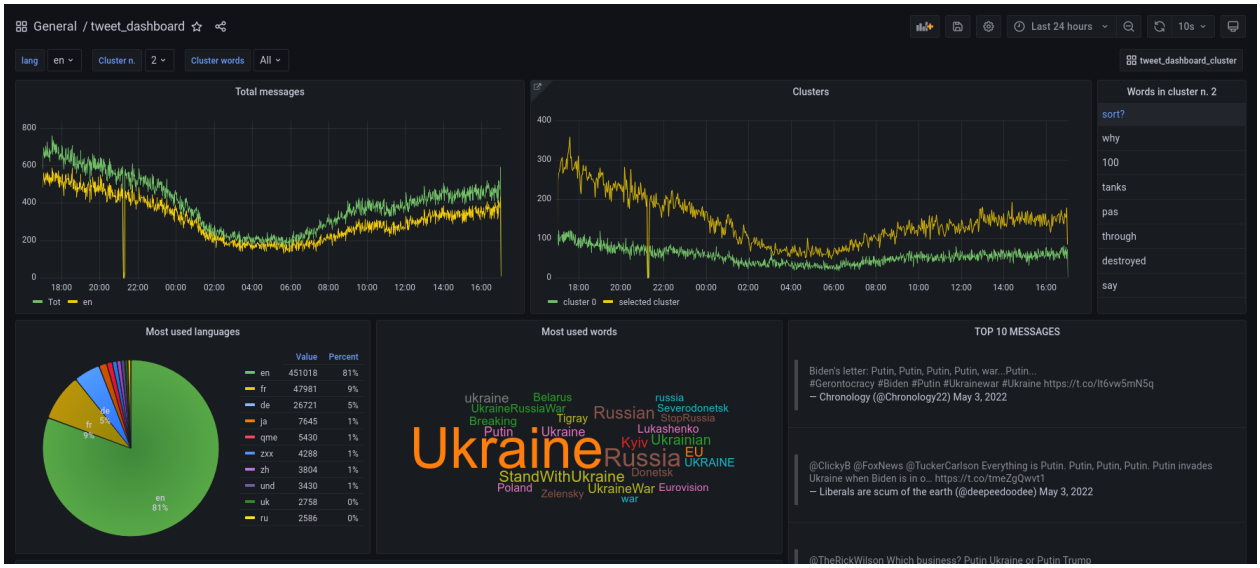

Figura 3: Screenshot della dashboard principale

Vediamo nel dettaglio cosa è presente nella dashboard.

Nell'angolo in alto a sinistra della dashboard (Figura 4) sono presenti il nome della dashboard (in questo caso: tweet_dashboard) e sono poi visibili e selezionabili tre variabili: lang (permette di selezionare una specifica lingua), Cluster n. (permette di selezionare uno specifico cluster) e cluster words (stabilito un cluster, permette di selezionare un sottoinsieme delle parole che compongono il cluster).

Subito in basso, sempre sulla sinistra, sono visibili due serie temporali (figura 5). La prima (Tot, in verde) rappresenta la quantità di messaggi scritti ogni minuto in totale, mentre la seconda (en, in giallo) rappresenta la stessa cosa, ma limitatamente ai messaggi in lingua inglese.

Spostandoci sulla destra, abbiamo (figura 6) una serie temporale che rappresenta, per ogni minuto, il numero di messaggi scritti che contengono le parole di uno specifico cluster.

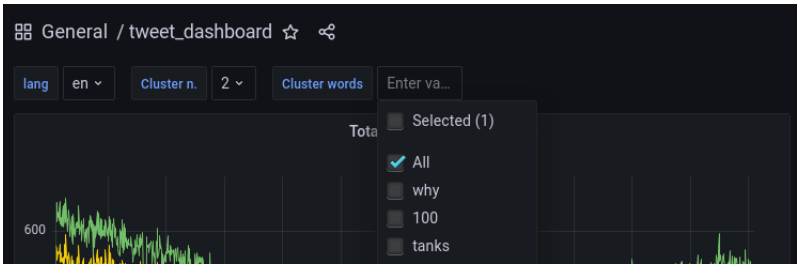

Figura 4: Dettaglio top-left della dashboard

Più in basso (figura 7) abbiamo, sulla sinistra, un grafico a torta che rappresenta le lingue principali in cui sono scritti i messaggi mentre, sulla destra, è mostrata una tag cloud con le parole più usate.

Ancora più a destra infine (figura 8) sono mostrati i messaggi considerati più rappresentativi.

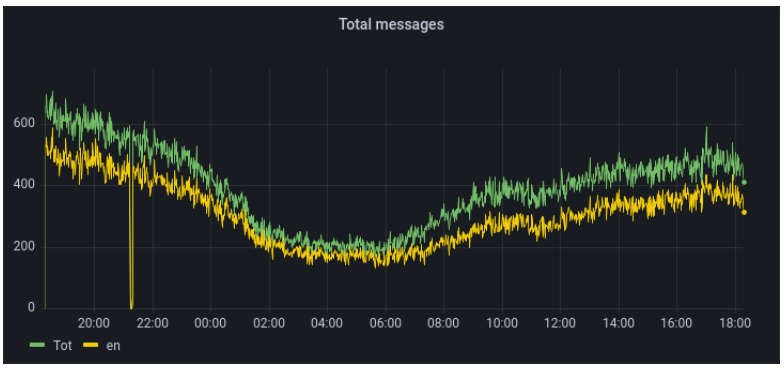

Figura 5: Dettaglio sulla serie temporale di tutti i messaggi

Figura 6: Dettaglio sulla serie temporale dei messaggi legati a uno specifico cluster

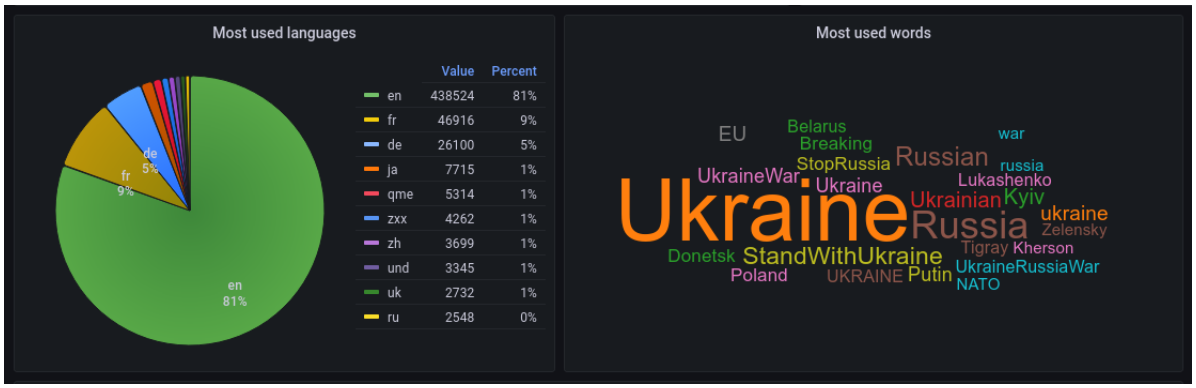

Figura 7: Dettaglio sul grafico a torta e sulla tag cloud

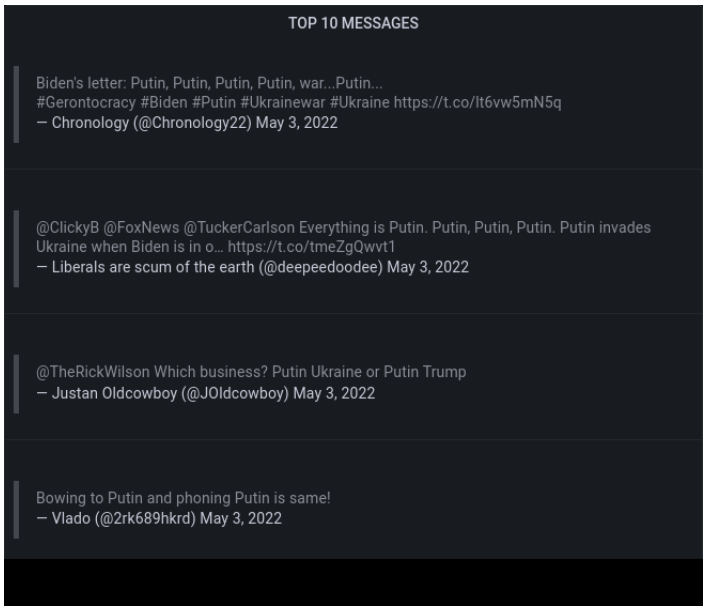

Figura 8: Dettaglio sui messaggi più significativi

1.11 Obiettivi Raggiunti

Attualmente l'applicazione web fornisce due dashboard. La prima, che si può osservare in figura 3, è globale e riguarda tutti i messaggi in uno specifico intervallo di tempo.

Da questa dashboard è possibile scegliere, dalle selezioni in alto a sinistra (vedi figura 4), la lingua tra quelle più usate (ad esempio: inglese, francese, tedesco, giapponese...) in modo da ottenere la visualizzazione della serie temporale dei messaggi in quella lingua (nella figura 5 è possibile osservare la serie temporale complessiva in verde e quella dei messaggi in lingua inglese in giallo; selezionando un'altra lingua viene visualizzata la serie temporale corrispondente ai messaggi scritti in quella lingua).

Inoltre è possibile selezionare è il singolo cluster. La scelta di un cluster in questa dashboard comporta l'aggiornamento della serie temporale sul lato destro (vedi figura 6) con la serie temporale dei messaggi che contengono almeno una parola del cluster selezionato.

Dal terzo menù (visibile in figura 4) è possibile selezionare uno specifico sottoinsieme delle parole parole del cluster e, in questo caso, il grafico a destra sarà relativo solo ai messaggi che contengono almeno una tra le parole del sottoinsieme del cluster.

Grafana fornisce la possibilità, attraverso il suo menù in alto a destra, di selezionare un intervallo temporale diverso da quello mostrato inizialmente. In questo modo, se si è interessati ad osservare un periodo specifico, è possibile selezionarlo e, in questo modo, tutte informazioni visualizzate nella dashboard (serie temporali, grafico a torta delle lingue, tag cloud) faranno riferimento a quella finestra temporale.

In ultimo è possibile passare, attraverso un pulsante, ad una seconda dashboard. Questa fornisce sostanzialmente le stesse metriche della prima dashboard, ma dettagliate sul cluster che abbiamo selezionato. E' possibile osservare come si presenta questa seconda dashboard in figura 9.

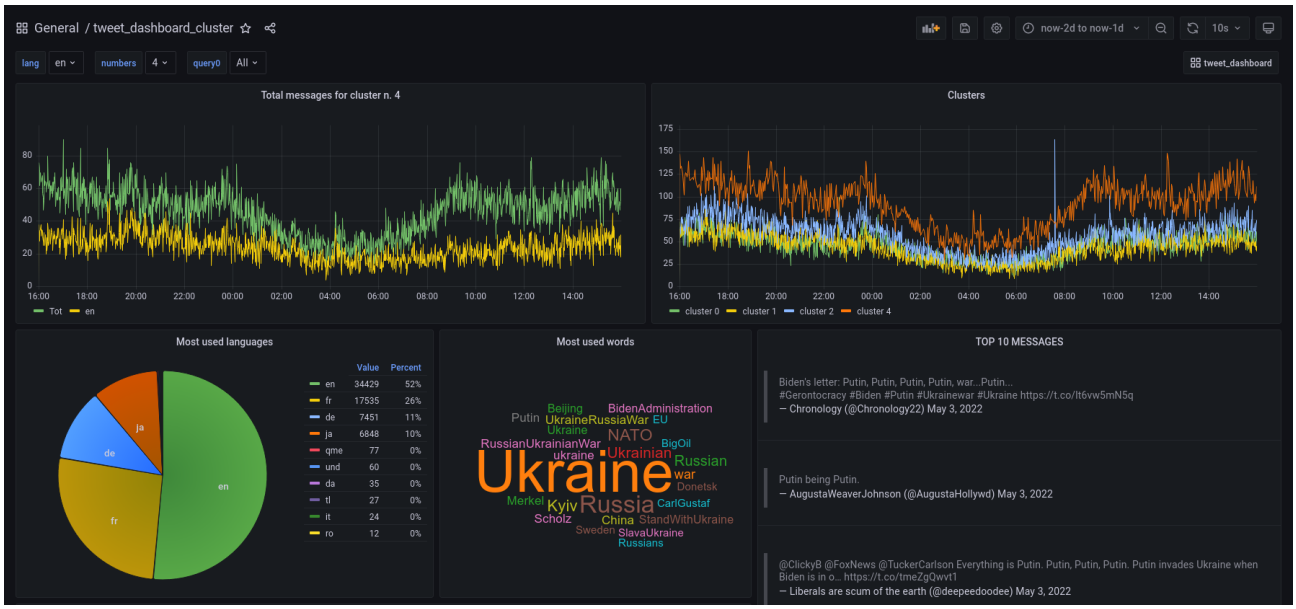

Figura 9: Screenshot della dashboard di un singolo cluster

Riferimenti bibliografici

[1] John Hopkins University and Medicine. COVID19 Map - John Hopkins Coronavirus Resource Center. `https://coronavirus.jhu.edu/map.html`, 2022. [Online il 06 aprile 2022].

[2] The Apache Software Foundation. Welcome to Apache Flume. `https://flume.apache.org/`, 2009-2021. [Online il 06 aprile 2022].

[3] The Apache Software Foundation. Apache Flink: Stateful Computations over Data Streams. `https://flink.apache.org/`, 2014-2022. [Online il 06 aprile 2022].

[4] Elasticsearch B.V. Logstash: Collect, Parse and Transform Logs. `https://www.elastic.co/logstash/`, 2022. [Online il 06 aprile 2022].

[5] Elasticsearch B.V. Elasticsearch: The Official Distributed Search & Analytics Engine. `https://www.elastic.co/elasticsearch/`, 2022. [Online il 06 aprile 2022].

[6] Elasticsearch B.V. Kibana: Explore, Visualize, Discover Data. `https://www.elastic.co/kibana/`, 2022. [Online il 06 aprile 2022].

[7] Grafana Labs. An overview on Grafana. `https://grafana.com/grafana/`, 2022. [Online il 06 aprile 2022].

[8] Docker Inc. Docker. `https://www.docker.com/`, 2022. [Online il 06 aprile 2022].

[9] Docker Inc. Docker Hub Container Image Library. `https://hub.docker.com/`, 2022. [Online il 10 giugno 2022].

[10] Flask. Flask documentation. `https://flask.palletsprojects.com/en/2.1.x/`, 2022. [Online il 10 giugno 2022].