Cloud-Based Multi-Sensor Simulator

Luigi Priano 1000002081.

Professori: Giuseppe Pappalardo - Andrea Francesco Fornaia.

Dipartimento: D.M.I. Informatica LM-18 A.A. 2024/2025

Data documento: 10/09/2024.

Indice

1 Introduzione al progetto

Il progetto "Cloud-Based Multi-Sensor Simulator" è pensato per creare un sistema che simula il funzionamento dei sensori degli pneumatici di un veicolo, concentrandosi su due parametri fondamentali: la temperatura e la pressione. L'obiettivo è fornire un modo efficace per monitorare in tempo reale questi valori, cruciali per garantire la sicurezza e le prestazioni del veicolo. Il traguardo generale del progetto è sviluppare un'architettura client-server completamente basata su cloud, con la flessibilità di moltiplicare sia i client che i server utilizzando i container. Questo approccio consente di scalare facilmente il sistema in base alle necessità, aggiungendo nuovi client che simulano i sensori degli pneumatici e nuovi server per gestire il carico di dati in crescita.

2 Architettura del progetto

Questa architettura rappresenta un sistema distribuito in cloud che sfrutta Kubernetes per gestire i container delle applicazioni e orchestrare i flussi di dati. Nella parte sinistra dell'immagine sono rappresentati diversi client, ciascuno ospitato in un container orchestrato da Kubernetes. Questi client inviano dati inerenti ai sensori a un sistema di messaggistica basato su "Cloud Pub/Sub", un servizio di messaggistica asincrona, progettato per ricevere e distribuire grandi quantità di messaggi in maniera efficiente. Il server è anch'esso containerizzato e gestito da Kubernetes, il suo ruolo è quello di consumare i messaggi dal servizio Pub/Sub, elaborarli e quindi inoltrare i risultati a Elasticsearch, utilizzato per indicizzare e analizzare i dati. Questo processo permette di gestire in modo scalabile l'elaborazione di grandi quantità di dati in tempo quasi reale. A seguire, i dati indicizzati da Elasticsearch vengono visualizzati e analizzati tramite Kibana, una piattaforma che consente di creare dashboard interattive e report dettagliati. Sarà grazie a Kibana se i dati verranno visualizzati tramite grafici in tempo reale. Di seguito viene mostrata l'architettura per facilitarne la comprensione.

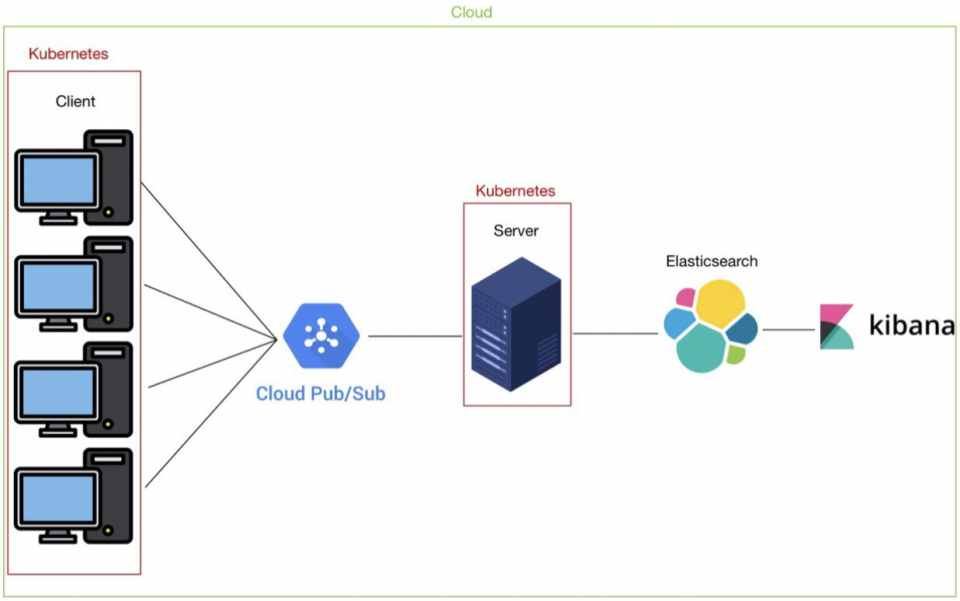


Fig. 1: Pipeline dell'architettura

2.1 Google Cloud

La piattaforma Cloud che è stata adottata nel progetto è Google Cloud. Questo servizio, offerto da Google, è una suite di servizi modulari di cloud computing tra cui archiviazione dati, analisi dei dati e apprendimento automatico, oltre ad una serie di strumenti di gestione. Funziona sulla stessa infrastruttura che Google utilizza internamente per i suoi prodotti per gli utenti finali. Google Cloud offre Infrastructure as a service (IAAS), Platform as a service (PAAS) ed ambienti serverless, nel progetto è stato usato un approccio misto tra IAAS e PAAS. L'uso di IaaS si riflette principalmente nell'uso di Kubernetes per orchestrare container e gestire le risorse di calcolo in cloud. In questo caso, hai il controllo dell'infrastruttura sottostante: i server virtuali, lo storage, e le risorse di rete che Kubernetes utilizza per gestire e distribuire i container. L'uso di PAAS è chiaramente visibile nell'integrazione di servizi gestiti come Cloud Pub/Sub, Elasticsearch e Kibana. Questi sono servizi di piattaforma che ti permettono di concentrarti sullo sviluppo e l'implementazione delle tue applicazioni, senza preoccuparti di gestire l'infrastruttura sottostante.

2.2 Kubernetes

Kubernetes, spesso abbreviato in K8s, è una piattaforma open-source per l'orchestrazione di container che automatizza molti processi necessari per eseguire applicazioni containerizzate in produzione. Pensato inizialmente da Google e ora mantenuto dalla Cloud Native Computing Foundation (CNCF), Kubernetes è progettato per aiutare gli sviluppatori e gli operatori a gestire applicazioni su larga scala, riducendo significativamente la complessità di gestione. In sostanza, Kubernetes si occupa di tutto ciò che riguarda l'esecuzione dei container, come il deployment, la scalabilità, il networking e il bilanciamento del carico, senza la necessità di interventi manuali. Kubernetes automatizza queste operazioni attraverso il suo cluster manager. Ogni cluster è costituito da un insieme di nodi, dove i container vengono eseguiti all'interno di entità chiamate "Pod". Il cluster manager monitora costantemente lo stato dei nodi e dei Pod, prendendo decisioni in tempo reale per garantire che l'infrastruttura rimanga sempre operativa e resiliente. Un vantaggio dell'uso di Kubernetes è la sua capacità di effettuare il bilanciamento del carico e la gestione del traffico in modo nativo. Kubernetes utilizza un servizio chiamato "Service" per definire un set di Pod e un endpoint univoco, consentendo così la comunicazione tra i diversi componenti dell'applicazione o tra l'applicazione stessa e l'esterno del cluster. Questo approccio facilita la scalabilità e assicura che ogni richiesta venga gestita dal container disponibile e meno carico. Nel progetto, Kubernetes è stato utilizzato per orchestrare i container dei client e del server, consentendo il loro deployment efficiente su Google Cloud.

2.2.1 Client

Il client ha come principale obiettivo, come già descritto, la simulazione di un sensore. Quest'ultimo simula la temperatura e la conseguente pressione di uno pneumatico. I punti principali del codice sono i seguenti:

- **Credenziali account di servizio google**: la variabile `service_account_file` specifica il percorso del file JSON contenente le credenziali di servizio per l'autenticazione

su Google Cloud. Questo file è utilizzato per configurare le credenziali dell'applicazione, permettendo al codice di autenticarsi correttamente con Google Cloud utilizzando l'ambiente variabile `GOOGLE_APPLICATION_CREDENTIALS`. L'account di servizio, dentro Google Cloud, è stato impostato con i permessi completi per Pub/Sub e Kubernetes;

- **Connessione al servizio Pub/Sub**: sono state definite alcune variabili di configurazione per il progetto e il topic Pub/Sub: `project_id` identifica il progetto su Google Cloud, mentre `topic_id` rappresenta il topic Pub/Sub a cui verranno inviati i messaggi;

- **Simulazione dei dati**: all'interno della funzione `publish_sensor_data()`, viene simulato un ciclo infinito che genera e pubblica dati di temperatura e pressione. La temperatura iniziale è impostata a 20 gradi Celsius. Ad ogni iterazione, la temperatura viene modificata casualmente con un incremento o decremento tra -3 e 3 gradi, mantenendola sempre nel range tra 15 e 50 gradi Celsius. La pressione viene calcolata in base alla temperatura corrente. Partendo da una pressione di base di 2.0 bar, la pressione aumenta di 0.05 bar per ogni grado Celsius sopra i 20 gradi e diminuisce di 0.05 bar per ogni grado sotto i 20 gradi;

- **Pubblicazione su Pub/Sub**: i dati simulati vengono convertiti in formato JSON e codificati in UTF-8 per poter essere trasmessi in rete. Una volta pronti i dati, il codice li pubblica su Google Cloud Pub/Sub.

Per creare l'immagine dello script, sono stati sviluppati sia il file Docker che il file YAML per caricare i dati in Kubernetes. In questo caso, all'interno di quest'ultimo file è specificato che devono essere configurati 4 client.

2.2.2 Server

Lo scopo primario del server è quello di ascoltare i messaggi all'interno della coda del servizio Google Cloud Pub/Sub, elaborarli e inoltrarli ad Elasticsearch. Anche in questo codice sono presenti le variabili per il collegamento a Google Cloud e al servizio Pub/Sub. Le parti fondamentali del codice sono le seguenti:

- **Connessione ad Elasticsearch**: per la connessione al servizio di Elasticsearch sono state usate le variabili `cloud_id` e `api_key`. La prima è il collegamento al progetto dentro il servizio di Elastic, la seconda variabile è utile per poter accedere dentro al progetto;

- **Pulizia dei dati dentro Elasticsearch**: la funzione `clear_elasticsearch_indices` cancella tutti gli indici in Elasticsearch, questa è utile per inizializzare il sistema in uno stato pulito;

- **Elaborazione dei messaggi**: è presente una funzione chiamata `callback`, questa viene chiamata ogni volta che arriva un messaggio nella coda del servizio Pub/Sub. All'interno viene mappato l'id del client per poter organizzare i dati dentro Elasticsearch e viene aggiunta anche la variabile `timestamp` per visualizzare i dati in tempo reale. Inoltre vengono gestiti gli eventuali errori.

Anche per questo codice sono stati sviluppati sia il file Docker che il file YAML per caricare i dati in Kubernetes. In questo caso viene specificato la distribuzione di un solo server.

2.2.3 Distribuzione su Kubernetes

Per entrambi i codici menzionati, è stato necessario eseguire una serie di comandi specifici per distribuire e gestire i dati all'interno di un cluster Kubernetes. Questi comandi sono stati utilizzati per configurare e implementare le risorse necessarie, assicurando che i servizi potessero operare in modo efficiente e scalabile nell'ambiente Kubernetes. Di seguito è presente una descrizione di ciascun comando e delle relative funzioni, che illustrano il processo di distribuzione e gestione dei dati nel cluster.

Il comando presente di seguito costruisce un'immagine Docker utilizzando le istruzioni del Dockerfile.server o Dockerfile.client e l'etichetta come ultima (latest), preparandola per essere caricata in un registro di container.

```
docker build -f Dockerfile.server|client -t
gcr.io/cloud-project-433315/server|client:latest .
```

Successivamente viene caricata l'immagine in un container remoto, in questo caso viene specificata che l'immagine da usare è l'ultima e il container si trova all'interno di Google Cloud.

```
docker push gcr.io/cloud-project-433315/server|client:latest
```

Per finire è presente un comando che applica una configurazione a un cluster Kubernetes, questo può essere utilizzato per creare o aggiornare risorse nel cluster. Questo comando, quindi, legge il file server-deployment.yaml o client-deployment.yaml e applica le configurazioni descritte nel file al cluster Kubernetes. Questo comporta la creazione o l'aggiornamento di un deployment che gestisce i pod e i servizi necessari per eseguire l'applicazione server o client.

```
kubectl apply -f server|client-deployment.yaml
```

Per istanziare il cluster all'interno di Kubernetes si può utilizzare il seguente comando specificando la zona dove far partire il cluster, il numero di nodi e l'abilitazione dell'ip alias.

```
gcloud container clusters create sensor-cluster \
--zone us-central1-a \
--num-nodes 5 \
--enable-ip-alias
```

Per configurare kubectl al cluster creato bisogna specificare il progetto di Google Cloud e la zona.

```
gcloud container clusters get-credentials sensor-cluster \
--zone us-central1-a \
--project cloud-project-433315
```

2.3 Google Cloud Pub/Sub

Google Cloud Pub/Sub è un servizio di messaggistica asincrono e scalabile che disaccoppia i servizi che producono messaggi dai servizi che li elaborano, consente, inoltre, ai servizi di comunicare in modo asincrono, con latenze dell'ordine di 100 millisecondi. Pub/Sub viene utilizzato per l'analisi dei flussi di dati e le pipeline di integrazione dei dati per caricare e distribuire i dati. È efficace anche come middleware orientato alla messaggistica per l'integrazione dei servizi o come coda per caricare in contemporanea le attività. Pub/Sub consente di creare sistemi di producer e consumer di eventi, chiamati publisher e sottoscritti. Gli editori comunicano con i sottoscrittori in modo asincrono trasmettendo eventi, anziché tramite chiamate di procedura remota sincrona (RPC). Gli usufruitori del servizio sono quindi Publisher e Subscriber, descritti come:

- **Publisher**: il compito principale di un publisher è inviare messaggi a un topic. I messaggi possono contenere dati, eventi o altre informazioni che devono essere distribuite ai subscriber. I messaggi possono essere in qualsiasi formato, ma comunemente sono rappresentati come JSON, XML o semplici stringhe di testo, nel caso del progetto sono formattati nel formato JSON. I publisher possono inviare messaggi in modo asincrono, il che significa che non devono attendere la conferma di ricezione da parte dei subscriber prima di inviare altri messaggi. Dopo aver inviato un messaggio, il publisher riceve una conferma che il messaggio è stato accettato dal servizio Pub/Sub. Non deve preoccuparsi della consegna finale ai subscriber, poiché questo è gestito automaticamente dal servizio. Quando invia un messaggio, il publisher specifica il topic a cui il messaggio deve essere inviato, nel caso del progetto si tratta del topic `sensor-data`;
- **Subscriber**: è un componente che riceve e processa i messaggi inviati a un topic. Il suo ruolo principale è ascoltare attivamente i messaggi pubblicati su un topic e gestirli in modo appropriato, garantendo che nessun dato venga perso e che tutte le informazioni necessarie siano processate. Il subscriber si iscrive a una subscription (sottoscrizione) associata a un topic. Una subscription rappresenta un canale attraverso cui i messaggi, pubblicati su un topic, sono consegnati ai subscriber, all'interno del topic sopraccitato si trova la sottoscrizione sotto il nome di `sensor-data-sub`. Sono presenti due tipi di Subscription:
  - **Pull Subscription**: in questo modello il subscriber contatta periodicamente il servizio Pub/Sub per verificare la presenza di nuovi messaggi e li recupera per l'elaborazione. È un approccio che offre un maggiore controllo al subscriber, che decide quando e quanto spesso richiedere i messaggi. All'interno del progetto è stato adottato questo modello;
  - **Push Subscription**: in quest'altro modello il servizio Pub/Sub invia automaticamente (push) i messaggi a un endpoint HTTP o HTTPS specificato dal subscriber. Questo approccio è utile quando si desidera una latenza minima e si ha un endpoint sempre disponibile che può gestire le richieste in tempo reale.

Il servizio Pub/Sub esiste anche in una versione "lite" ovvero una versione semplificata e meno costosa di Pub/Sub, pensata per casi d'uso dove il costo è una priorità e i requisiti di scalabilità e disponibilità sono più prevedibili o limitati. Le differenze principali sono le seguenti:

- **Scalabilità manuale**: a differenza di Pub/Sub, Pub/Sub Lite richiede che l'utente configuri manualmente la capacità in termini di partizioni e throughput;

- **Topic a livello di zona**: i dati sono archiviati all'interno di una singola zona, senza replica su altre zone;

- **Topic a livello di regione**: i dati vengono replicati in modo asincrono in una seconda zona, fornendo una maggiore ridondanza rispetto all'archiviazione a livello di zona, ma con un livello di affidabilità inferiore rispetto a Pub/Sub standard;

- Un ulteriore differenza con il servizio completo è la necessità di utilizzare il pre-provisioning, specificando in anticipo la capacità di throughput e la capacità di archiviazione. Questo rende l'architettura non scalabile automaticamente ed inoltre abbassa i costi poichè vengono mantenute sempre le stesse risorse.

In conclusione è consigliato l'approccio di questo servizio "lite" solo per quelle applicazioni in cui la riduzione dei costi giustifica il lavoro operativo aggiuntivo e un livello di affidabilità inferiore. Non avendo una gestione dei costi limitata all'interno del progetto è stato adoperata la versione completa e non la versione appena citata.

2.4 Elasticsearch

Elasticsearch è un motore di ricerca e analisi distribuito, open source, che permette di archiviare, cercare e analizzare grandi volumi di dati in tempo reale. Originariamente creato nel 2010 da Shay Banon, Elasticsearch è costruito sulla libreria di ricerca Apache Lucene, ma offre un'interfaccia molto più semplice e intuitiva per l'utilizzo delle sue potenti funzionalità. È una delle tecnologie principali all'interno del cosiddetto "ELK stack" (Elasticsearch, Logstash, Kibana), una suite molto popolare per la gestione e l'analisi dei log e dei dati. Elasticsearch si distingue per diverse caratteristiche e funzionalità che lo rendono ideale per una varietà di casi d'uso, dal monitoraggio delle applicazioni all'analisi dei dati provenienti da IoT o social media. Ecco alcune delle sue peculiarità principali:

- **Ricerca in tempo reale**: una delle caratteristiche più potenti di Elasticsearch è la sua capacità di fornire risultati di ricerca quasi istantanei. È progettato per indicizzare rapidamente grandi volumi di dati e renderli immediatamente disponibili per le query di ricerca;

- **Indicizzazione e query avanzate**: Elasticsearch permette di indicizzare e interrogare i dati in modo estremamente flessibile. Supporta una vasta gamma di tipi di dati, inclusi testo, numeri, date, dati geografici e dati strutturati o non strutturati. Si possono eseguire query complesse, come ricerche full-text, filtri, aggregazioni e operazioni di analisi geospaziale, il tutto con tempi di risposta molto rapidi;

- **Integrazione con altri plugin**: Elasticsearch è molto flessibile e può essere utilizzato in combinazione con altri strumenti. Ad esempio, fa parte del "ELK Stack" insieme a Logstash (per la raccolta e trasformazione dei dati) e Kibana (per la visualizzazione e il monitoraggio dei dati). Questo permette di creare Dashboard interattive e aiuta nello studio dei dati;

- **Utilizzo delle API**: Elasticsearch offre semplici API basate su REST, una semplice interfaccia HTTP, e utilizza documenti JSON privi di schema, facilitando l'avvio e la creazione rapida di applicazioni per un'ampia varietà di casi d'uso. Queste permettono di eseguire operazioni di ricerca, indicizzazione, aggiornamento e cancellazione dei dati in modo semplice e intuitivo.

Esistono delle varianti di Elasticsearch gestite da terzi come Amazon OpenSearch Service. Quest'ultimo offre le stesse funzionalità di Elasticsearch, ma con la comodità di un servizio completamente gestito, in cui AWS si occupa della scalabilità, gestione, backup e aggiornamenti. All'interno di Elasticsearch, i dati provenienti dai sensori vengono gestiti in modo efficiente per garantire una rapida ricerca, analisi e visualizzazione. Quando il server riceve i dati dai sensori, essi vengono indicizzati in Elasticsearch, utilizzando un insieme specifico di campi:

- **ID**: ogni sensore ha un identificativo univoco che permette di distinguere e tracciare i dati provenienti da quel particolare dispositivo;
- **Timestamp**: questo campo registra il momento esatto in cui i dati sono stati raccolti. Questo è fondamentale per abilitare la visualizzazione quasi in tempo reale sulla dashboard;
- **Temperature e Pressure**: questi sono i valori effettivi rilevati dai sensori. Elasticsearch indicizza questi dati numerici, consentendo di eseguire ricerche rapide e di creare aggregazioni per visualizzare le tendenze o i cambiamenti nel tempo.

Fig. 2: Esempio di dati all'interno di Elasticsearch

Come è facile osservare in figura sono presenti dei campi detti "meta", questi sono campi speciali che vengono utilizzati per immagazzinare informazioni aggiuntive o di contesto su un documento indicizzato. Gli unici campi meta utilizzati nel progetto sono:

- **_id**: ovvero un identificativo univoco che viene auto generato da Elasticsearch stesso;
- **_index**: questo valore viene impostato dal server. Elasticsearch utilizza gli indici per organizzare e memorizzare i dati. Ogni insieme di dati proveniente da un sensore viene memorizzato in un indice specifico, il che permette di attivare diversi grafici e visualizzazioni nella dashboard;

2.5 Kibana

Kibana è un potente strumento di visualizzazione e analisi dei dati progettato per integrarsi perfettamente con Elasticsearch. Fa parte dell'Elastic Stack (ELK Stack), Kibana fornisce un'interfaccia utente intuitiva e interattiva per esplorare, visualizzare e gestire i dati archiviati in Elasticsearch. La caratteristica principale di Kibana è l'interrogazione e l'analisi dei dati. Inoltre, permette di visualizzare i dati in modi alternativi utilizzando mappe di calore, grafici a linee, istogrammi, grafici a torta e mappe geospaziali. Con vari metodi, è possibile cercare i dati memorizzati in Elasticsearch per la diagnostica di sensori nel campo dell'Internet Of Things (IOT) oppure i log provenienti da diversi sistemi e/o applicazioni. La possibilità di costruire e condividere rapidamente dashboard dinamiche che catturano in tempo reale i dati salvati in Elasticsearch lo rende uno strumento molto utile in molti scenari. Le voci principali di Kibana sono le seguenti:

- **Analytics**: sono presenti molte funzionalità relative alla visualizzazione e analisi dei dati. In particolare, la pagina Discover permette un'esplorazione interattiva dei dati, in cui è possibile filtrare i dati per periodo temporale e/o visualizzare solo i campi di nostro interesse. Nel progetto sono stati presi tutti i dati con la voce `sensor-data-*` in modo da prendere tutti gli indici che iniziano con `sensor-data-`. Un'altra sezione di Analytics è Dashboard nella quale è possibile creare delle Dashboard personalizzate in base allo studio da sottoporre. All'interno di ogni Dashboard sono presenti le così dette "Visualizzazioni" ovvero i grafici che sfruttano i dati presenti dentro Discover. Queste visualizzazioni possono essere create con le voci Canvas o Maps. È possibile sviluppare degli algoritmi di Machine Learning utilizzando i dati all'interno di Elasticsearch, mentre, per finire è presente la voce "Graphs" che mostra eventuali relazioni tra i dati;

- **Enterprise search**: in questione sezione si possono creare soluzioni per la ricerca testuale. Infatti, mediante un crawler interno non è necessario inserire manualmente in Elasticsearch i dati da indicizzare, ma il tutto verrà fatto in modo automatico. Differentemente Workspace Search creare un motore di ricerca collegato alle risorse di un'organizzazione;

- **Observability**: questo menù è focalizzato sul monitoraggio, analisi e visualizzazione degli eventi che sono catturati e salvati in Elastisearch. La sezione Logs permette di visualizzare in tempo reale le fonti dei log che sono state definite e filtrare i dati di interesse. Mediante le pagine della sezione Metrics si possono analizzare e visualizzare le metriche della propria architettura (CPU, utilizzo RAM, ecc.) oltre ad impostare la ricezione di avvisi qualora si verificassero delle anomalie. Il sistema Application Performance Monitoring (APM) supporta gli utenti a monitorare le applicazioni e i servizi raccogliendo alcune metriche relative alle prestazioni e agli errori. La sezione Uptime monitora proattivamente la disponibilità dei siti e/o servizi, e verifica la validità dei certificati SSL;

- **Security**: Elastic Security combina le funzionalità di rilevamento delle minacce di un sistema SIEM (security information and event management) con la prevenzione e le capacità di risposta degli endpoint. Queste capacità analitiche e di protezione, sfruttando la potenza di Elasticsearch, questo consente di difendere l'organizzazione dalle minacce prima che si verifichino danni e perdite;

- **Management**: in fine all'interno di questa sezione è possibile eseguire diverse operazioni. Nella pagina DevTools si possono sottomettere le query di interesse per verificare la sintassi e i risultati. Diversamente nella pagina Integrations si possono esplorare diversi agent per collezionare i dati da più di 100 servizi diversi. In modo analogo Fleet fornisce un'interfaccia per aggiungere e gestire le integrazioni per i servizi e le piattaforme più popolari, nonché per gestire una flotta di agent. In Stack Monitoring è possibile monitorare lo stato del proprio stack. Infine, nella pagina di Stack Management si possono gestire diverse funzionalità tra cui: i modelli degli indici, i backups, gli utenti con i relativi ruoli, i workspaces, gli oggetti salvati, i tasks di alerting e di machine learning.

3 Visualizzazioni Dashboard

Per l'esecuzione del progetto, come già specificato, è stato configurato un sistema composto da un server e quattro client. Per valutare l'efficacia del sistema e la sua capacità di gestire un flusso continuo di dati, è stato eseguito un test della durata di 20 minuti. Durante questo periodo, i client hanno inviato messaggi a una frequenza di circa 23-24 invii ogni 30 secondi, come visibile nella figura sottostante, accumulando un totale di 965 messaggi.

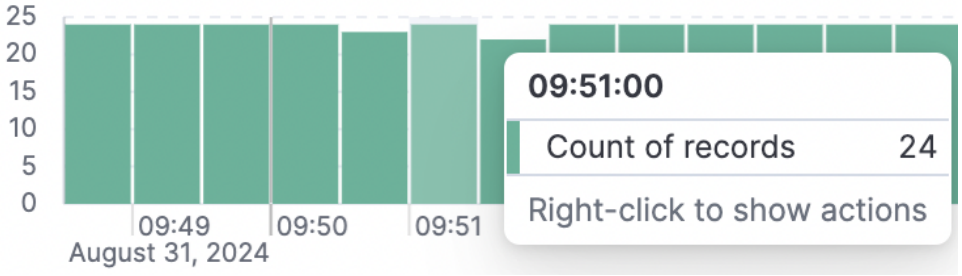


Fig. 3: Messaggi registrati ogni 30 secondi

Questo volume di dati ha permesso di testare vari aspetti critici del sistema, inclusa la capacità del server di elaborare e indicizzare i messaggi in tempo reale senza ritardi significativi. Il test ha rivelato come il sistema gestisce un flusso elevato di messaggi e ha dimostrato l'efficacia della coda Pub/Sub nel distribuire i messaggi al server in modo affidabile. Tramite Pub/Sub si è riuscito ad osservare un picco di invii pari a 0,85 messaggi al secondo, con una media di 0,64. Usufruendo di Kibana sono state create sei visualizzazioni principali, due delle quali moltiplicate per ogni sensore.

3.1 Media della pressione globale

Il grafico 'Media della pressione globale' mostra sull'asse delle ascisse il timestamp suddiviso in intervalli di 30 secondi, mentre sull'asse delle ordinate riporta la media della pressione rilevata da tutti i sensori. Questo grafico può essere utile per monitorare la pressione media dei pneumatici nel tempo. Permette di identificare eventuali variazioni anomale o improvvise della pressione che potrebbero indicare una perdita d'aria, un problema di gonfiaggio o una differenza di pressione tra i pneumatici.

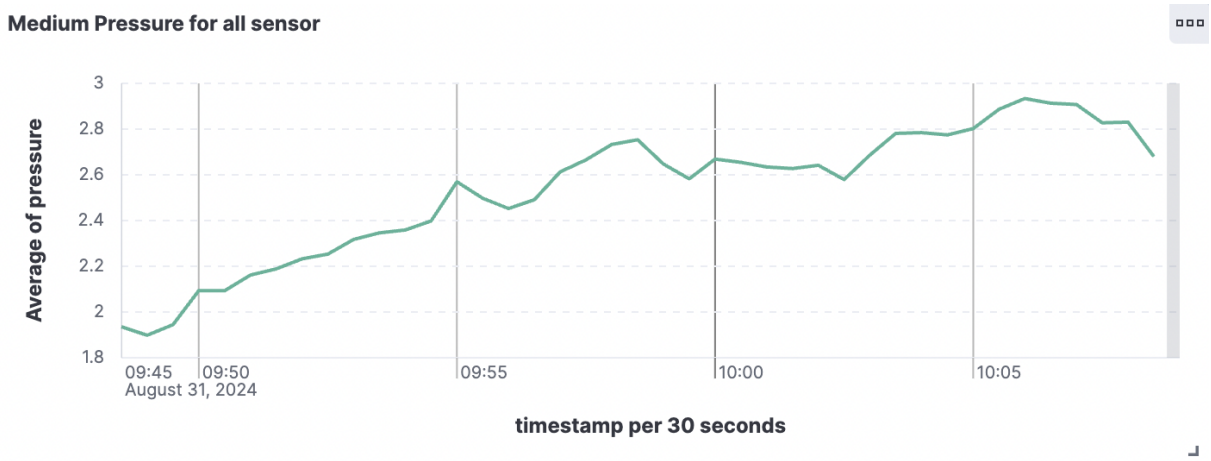


Fig. 4: Media pressione globale

3.2 Media della temperatura globale

Il seguente grafico, a differenza del precedente, presenta sull'asse delle ordinate la media delle temperature. Questo grafico è utile per monitorare la temperatura media dei pneumatici durante l'uso del veicolo. La temperatura è un indicatore critico delle condizioni di funzionamento dei pneumatici: un surriscaldamento può indicare un sovraccarico, o un'eccessiva velocità, mentre temperature troppo basse possono influire sull'aderenza e sulla capacità di frenata.

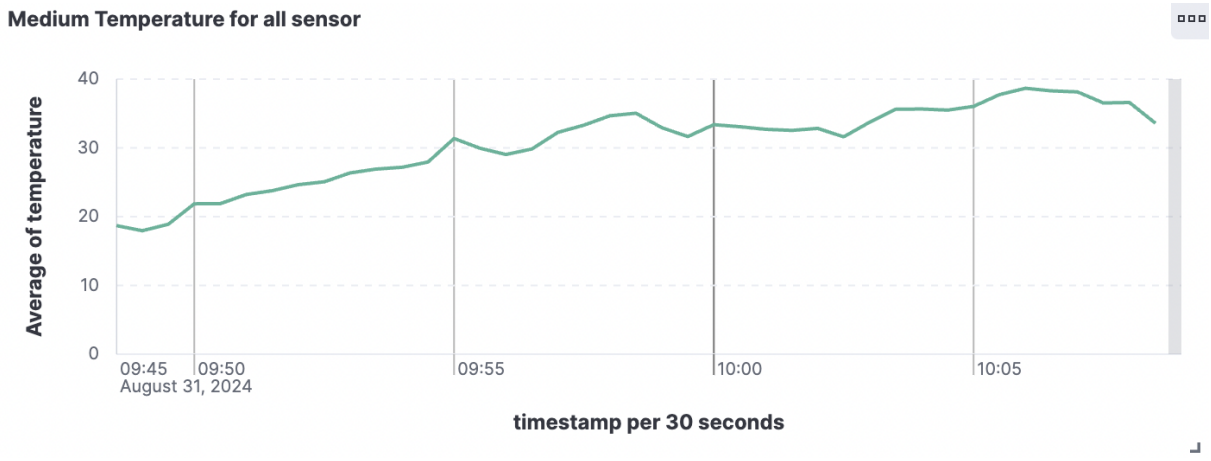


Fig. 5: Media temperatura globale

3.3 Pressione differente per ogni sensore

Il grafico seguente è simile a quello della media della pressione globale, ma mostra i valori di pressione rilevati separatamente da ciascun sensore. È evidente la presenza di quattro pressioni distinte, una per ogni sensore, contraddistinte dai differenti colori. Questo grafico consente di monitorare individualmente la pressione di ciascun pneumatico in tempo reale. Può essere utilizzato per identificare rapidamente eventuali differenze di pressione tra i pneumatici.

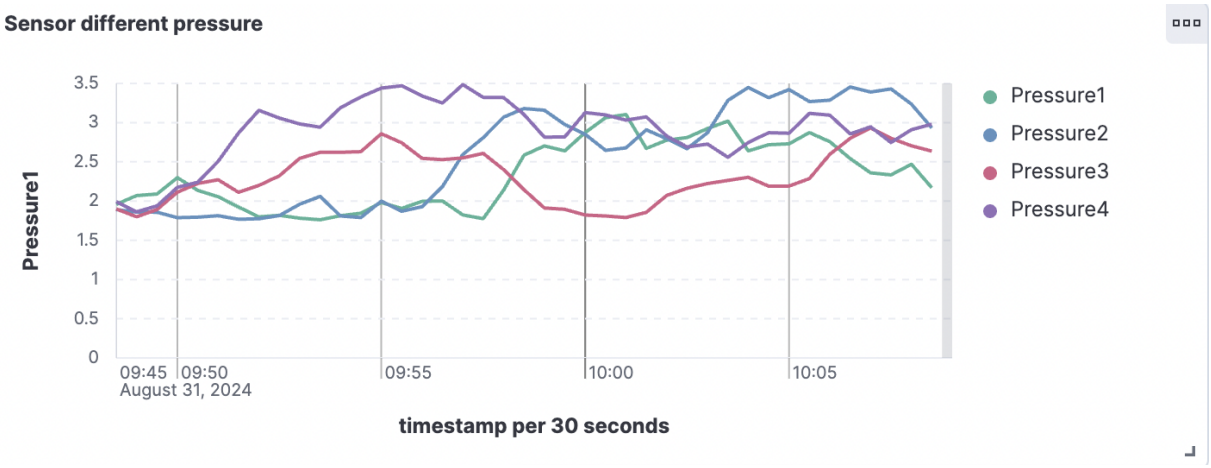


Fig. 6: Pressione per ogni sensore

3.4 Temperatura differente per ogni sensore

In questo grafico la temperatura non è globale ma suddivisa per ogni sensore, anche in questo caso sono presenti quattro temperature differenti. Tramite l'utilizzo di questo grafico è possibile osservare eventuali cali o picchi di temperatura dovuti ad eccessiva velocità, slittamenti o rallentamenti. Questi nell'ambito del motorsport possono comportare cali di prestazione, nell'ambito urbano possono indicare problemi di gonfiaggio.

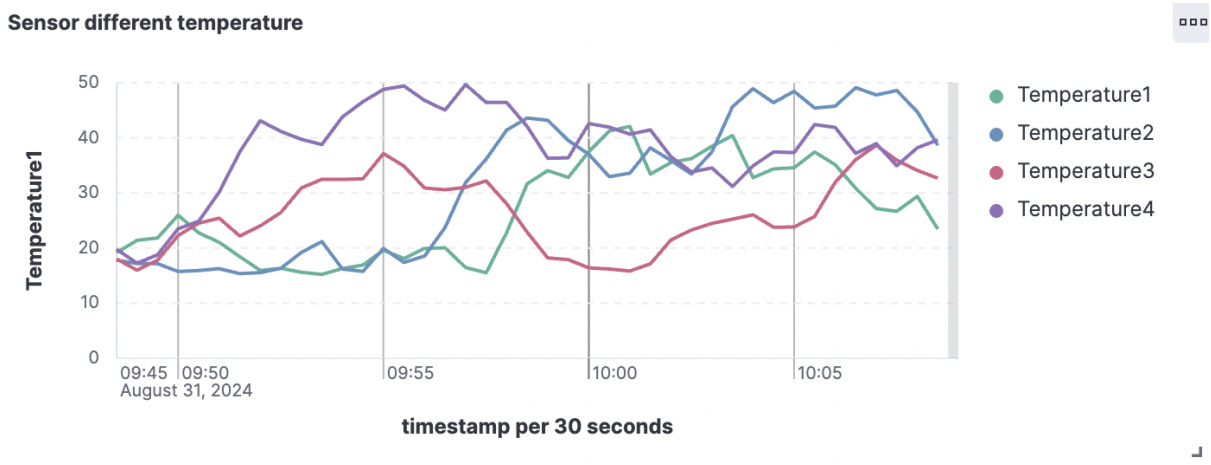


Fig. 7: Temperatura per ogni sensore

3.5 Pressione per ogni sensore

Il grafico della pressione di un singolo pneumatico, misurata dal relativo sensore, mostra in tempo reale il comportamento dello pneumatico. Questo consente di generare allarmi in caso di valori eccessivamente alti o bassi, permettendo di monitorare con precisione le condizioni del pneumatico. I colori del grafico variano dal blu (pressione bassa, indicativa di un pneumatico troppo freddo) al rosso (pressione alta, indicativa di un pneumatico troppo caldo). I range di colore sono stati scelti equamente tra 1.75 (minimo) e 3.5 (massimo), ma possono essere adattati in base alla gravità della situazione.

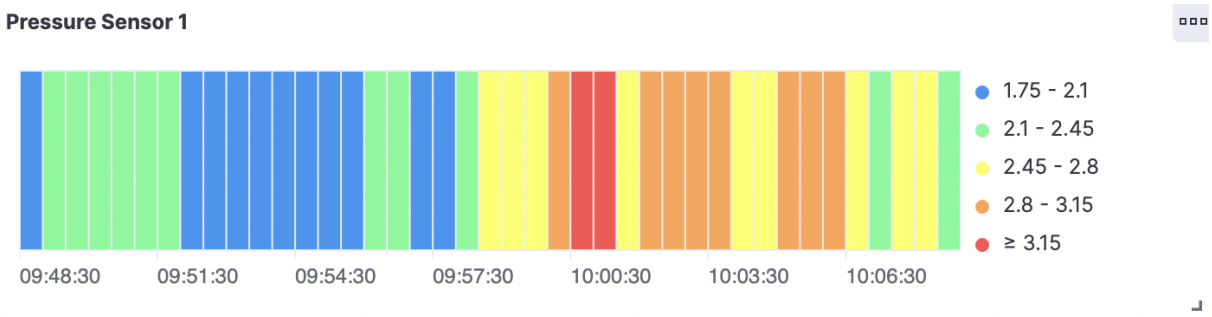


Fig. 8: Esempio pressione sensore 1

3.6 Temperatura per ogni sensore

Come nel grafico precedente, in questo caso la temperatura rappresenta un dato più affidabile rispetto alla pressione, poiché fornisce un'indicazione fisica più accurata sullo stato della gomma. Il range va da 15 (minimo, pneumatico molto freddo) a 50 (massimo, pneumatico molto caldo). Anche in questo caso, i valori sono suddivisi equamente, ma possono essere modificati in base alle necessità dell'analista. È possibile dimostrare che, dividendo equamente i valori di temperatura e pressione, i grafici risultanti tendono a essere simili. Per questo motivo, è consigliabile differenziare i range tra i due grafici. Nel progetto attuale, trattandosi di una simulazione, i dati di pressione vengono calcolati sulla base dei valori di temperatura. In un contesto reale, tuttavia, questi due parametri potrebbero non variare necessariamente in modo proporzionale.

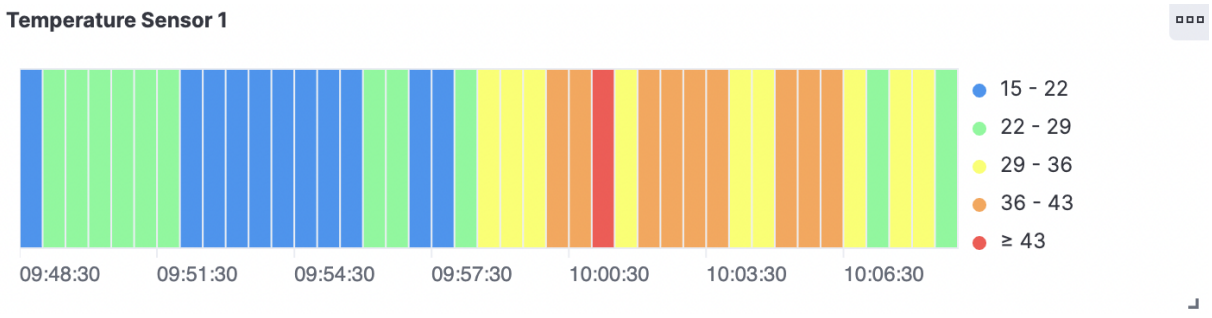
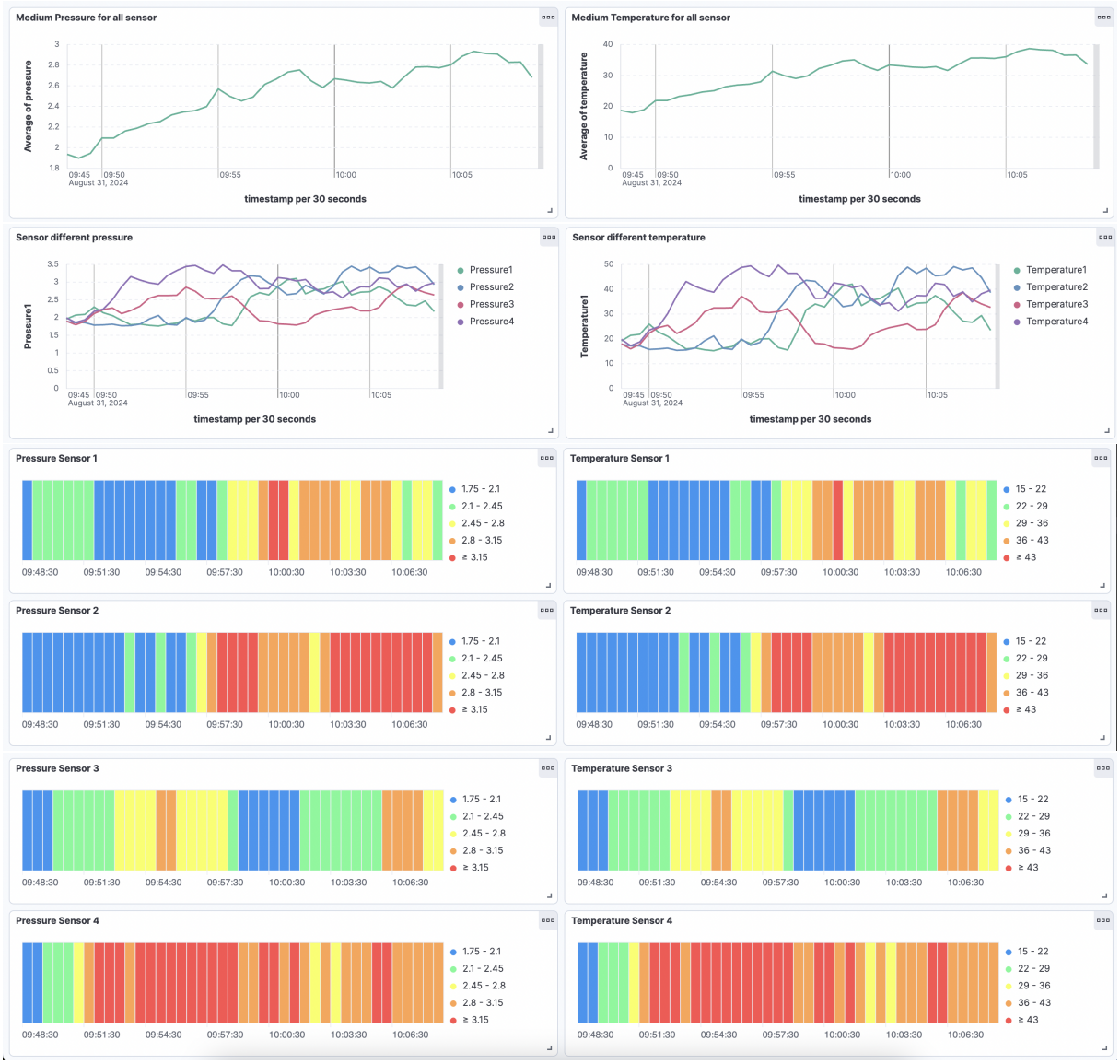


Fig. 9: Esempio temperatura sensore 1

3.7 Esempio completo di Dashboard



4 Costi dei servizi

Questa sezione esamina i costi dei servizi descritti in precedenza, che possono variare in base al tempo di utilizzo, alle risorse istanziate e alle prestazioni richieste. I principali servizi a pagamento del progetto sono il cluster Kubernetes, il servizio Pub/Sub e Elasticsearch su cloud. Ognuno di questi sarà analizzato in base al rapporto costo per ora, per poi determinare il costo effettivo del progetto. Di seguito sono riportati i costi principali:

- **Kubernetes**: i costi di Kubernetes su Google Cloud (Google Kubernetes Engine, GKE) possono variare in base a diversi fattori, come il tipo di macchina virtuale (VM) utilizzata per i nodi del cluster, la quantità di risorse allocate, la regione in cui è distribuito il cluster, e le funzionalità opzionali attivate.
  - **Tipo di nodi**: i costi dei cluster Kubernetes dipendono principalmente dalla dimensione e dal tipo dei nodi (macchine virtuali) che li compongono. Nodi con più CPU, RAM o GPU sono più costosi rispetto a nodi con meno risorse;
  - **Numero di nodi**: aumentare il numero di nodi nel cluster porta a un aumento proporzionale dei costi. Più nodi significano più risorse utilizzate e, quindi, costi più elevati;
  - **Regioni e zone**: i costi possono anche variare a seconda della regione o della zona in cui vengono distribuiti i cluster. Alcune regioni potrebbero essere più costose rispetto ad altre a causa della domanda, della disponibilità di risorse, o di specifici costi infrastrutturali;
  - **Costi di rete**: questi possono variare in base al traffico in uscita dal cluster, soprattutto se i dati vengono trasferiti tra regioni diverse o fuori dal cloud.

  Il risparmio sui costi di questo servizio può essere ottenuto ridimensionando i nodi in base alle reali necessità del sistema, utilizzando l'auto-scaling per ottimizzare dinamicamente il numero di nodi e le risorse in base al carico di lavoro, scegliendo regioni o zone meno costose e ottimizzando l'uso delle risorse per sfruttare al meglio quelle già istanziate. Se l'implementazione richiede risorse aggiuntive, queste possono essere aumentate tramite l'auto-scaling o manualmente, modificando il tipo o il numero di nodi. In tal caso, i costi del servizio aumenteranno proporzionalmente in base alle modifiche effettuate;

- **Google Cloud Pub/Sub**: i costi del servizio possono variare in base a diversi fattori, come il volume dei dati trasferiti, il numero di richieste di pubblicazione (pub) e di consegna (sub), la memoria necessaria per archiviare i messaggi in coda, e le risorse impiegate durante le operazioni. Di seguito vengono approfonditi questi aspetti:
  - **Volume di dati trasferiti**: i costi di Pub/Sub sono principalmente influenzati dal volume di dati trasferiti. Ogni messaggio pubblicato e consegnato genera un costo in base alla quantità di dati trasmessi. I costi sono calcolati per gigabyte (GB) trasferiti, sia per i messaggi pubblicati che per quelli consegnati ai Subscriber;
  - **Operazioni di pubblicazione e consegna**: i costi includono sia le operazioni di pubblicazione (pub) che quelle di consegna (sub), in pratica ogni messaggio

pubblicato nel topic ha un costo. Inoltre se sono presenti diversi Subscriber il costo per ogni ricezione è differente per ognuno di loro;

- **Retention dei messaggi**: Pub/Sub permette di conservare i messaggi per un certo periodo (fino a 7 giorni) prima che vengano automaticamente eliminati. La conservazione dei messaggi ha un costo basato sulla quantità di dati conservati e sul tempo di retention;
- **Throughput e latenza**: il costo del servizio può variare anche in base al throughput (numero di messaggi al secondo) e alla latenza desiderata.

I costi del servizio possono essere ridotti, come già menzionato, utilizzando la versione "Lite" di Pub/Sub. In questo modello, la capacità delle risorse viene definita in base alle esigenze specifiche, rendendo i costi delle risorse più prevedibili rispetto alla versione standard. Inoltre, i costi di archiviazione a lungo termine sono inferiori e il servizio è limitato geograficamente a una sola regione o a due regioni con richieste asincrone. Un'ulteriore strategia di risparmio consiste nel ridurre il numero di richieste di pubblicazione (pub) o consegna (sub) e nel diminuire la quantità di dati nei singoli pacchetti.

- **Elastic**: anche per Elastic, i costi variano in base a diversi fattori, tra cui le risorse utilizzate, la larghezza di banda richiesta e lo storage impiegato, compresi eventuali backup e snapshot. Di seguito vengono spiegati in dettaglio questi aspetti:

  - **Risorse di calcolo**: i costi principali sono legati alle risorse di calcolo utilizzate dai nodi Elasticsearch. Questo include CPU e memoria. Le istanze delle macchine virtuali (VM) che ospitano i nodi Elasticsearch sono fattori chiave nella determinazione dei costi. Maggiore è la potenza di calcolo e la quantità di memoria richiesta, maggiore sarà il costo;
  - **Storage dei dati**: Elastic Cloud su Google Cloud addebita i costi in base allo storage dei dati. Questo include sia lo storage per i dati indicizzati (come documenti e log) sia per i backup e le snapshot;
  - **Traffico di rete**: i costi di rete sono associati al traffico di dati tra i nodi Elasticsearch e tra i client e i server Elasticsearch. I dati in uscita dalla rete Google Cloud verso l'esterno o tra diverse regioni possono comportare costi aggiuntivi;
  - **Backup e snapshot**: Elastic Cloud offre opzioni di backup e snapshot per la protezione dei dati. Questi servizi comportano costi aggiuntivi basati sulla frequenza e sulla durata del backup. Maggiore è il numero di snapshot e la loro durata, maggiori saranno i costi.

  Per risparmiare sui costi di Elastic, è fondamentale ottimizzare la configurazione dei nodi scegliendo risorse adeguate alle esigenze specifiche dell'implementazione. È utile adottare strategie di gestione dei dati, come la rotazione e la compressione, per ridurre il volume di dati memorizzati. Inoltre, è possibile minimizzare il traffico di rete mantenendo i dati all'interno della stessa regione e pianificare backup periodici limitando la durata delle snapshot.

4.1 Costi dell'implementazione

Il costo del progetto può essere suddiviso tra i tre servizi menzionati, analizzando i costi specifici per ciascuno di essi. Questi costi possono essere forniti sia per unità (ad esempio, per singolo messaggio o operazione) sia in base alle tempistiche orarie, consentendo una valutazione dettagliata delle spese complessive.

- **Kubernetes**: per il progetto è stato utilizzato il tipo di nodo "e2-medium", che dispone di 2 vCPUs e 4 GB di memoria, con un costo di 0,0335 € all'ora per singolo nodo. Per garantire che il server e i client avessero sufficienti risorse e non andassero in crash, sono stati impiegati 5 nodi, in questo modo le risorse disponibili sono 10 vCPUs e 20 GB di memoria. Pertanto, il costo totale per l'utilizzo di questi nodi, e di conseguenza del servizio Kubernetes, è di 0,1675 € all'ora;
- **Google Cloud Pub/Sub**: il costo del servizio Google Cloud Pub/Sub è di 0,04 € per ogni milione di messaggi pubblicati e di 0,04 € per ogni milione di messaggi letti tramite sottoscrizioni. Inoltre, le richieste di pubblicazione e lettura dei dati sono fatturate a 8 € per ogni milione di dati processati;
- **Elasticsearch**: la sottoscrizione all'abbonamento di Elastic presenta un costo di 0,000092484 € per unità.

Analizzando il progetto su un periodo di 20 minuti, durante il quale sono state inoltrate 965 richieste al server, i costi possono essere calcolati come segue:

- **Kubernetes**:
  - Costo orario per nodo: 0,0335 €
  - Numero di nodi: 5
  - Costo orario per l'intero cluster: 0,0335 € * 5 = 0,1675 €
  - Costo per 20 minuti: 0,1675 € * 1/3 = 0,05583 €
- **Google Cloud Pub/Sub**:
  - Costo per le richieste di pubblicazione e lettura: 0,00772 €
  - Costo per messaggi pubblicati e letti: 0,000386 €
- **Elasticsearch**:
  - Costo per le richieste: 965 * 0,000092484 = 0,0893 €

Il costo finale dell'applicazione per 20 minuti è pari a:
0,05583 € + 0,008106 € + 0,0893 € = 0,153236 €, circa 0,459708 € l'ora.

È possibile ottenere risparmi utilizzando Google Cloud Pub/Sub Lite e un cluster di dimensioni ridotte. Con Pub/Sub Lite, i costi sono suddivisi come segue: un throughput di 1 Mb al secondo costa 0,01 €, e lo storage di 1 GB costa anch'esso 0,01 €, per un totale di 0,02 €. Questo è più elevato rispetto ai 0,008106 € del servizio completo di Pub/Sub. Tuttavia, Pub/Sub Lite offre vantaggi economici significativi soprattutto con un numero elevato di richieste; nel caso del progetto in esame, non si osserva un risparmio immediato,

ma piuttosto un aumento dei costi. Per quanto riguarda il cluster, è possibile optare per il nodo "e2-small", che offre potenza di calcolo e risorse inferiori rispetto al nodo "e2-medium". Il costo orario per un singolo nodo "e2-small" è di 0,0168 €. Utilizzando 5 nodi, il costo orario totale diventa 0,084 €, e il costo per 20 minuti di utilizzo è di circa 0,028 €. Utilizzando il cluster sopraccitato i costi per 20 minuti sono pari a:
0,028 € + 0,008106 € + 0,0893 € = 0,125406 €, circa 0,376218 € l'ora. Il risparmio ottenuto con queste opzioni è circa del 18,15% rispetto ai costi originali.

5 Conclusione

L'obiettivo principale del progetto era implementare un servizio multi-client e single server direttamente su una piattaforma cloud, con l'intento di ridurre i costi associati alla gestione di infrastrutture fisiche e al personale tecnico. Spostando il servizio su cloud, il progetto ha potuto concentrarsi esclusivamente sul risultato finale, delegando al provider cloud la gestione delle risorse e delle operazioni quotidiane. Nel corso del progetto, è emerso che l'adozione di soluzioni basate su cloud non solo ha permesso di ottimizzare i costi operativi, ma ha anche semplificato la gestione del servizio, eliminando la necessità di amministrare direttamente le macchine e i sistemi sottostanti.

5.1 Sviluppi futuri

Di seguito vengono elencati possibili sviluppi futuri:

- **Scalabilità con Multi-Server**: un possibile sviluppo futuro del progetto prevede l'espansione da un'architettura single server a un'architettura multi-server. Questa modifica potrebbe essere necessaria nel caso in cui il numero di sensori aumenti e un singolo server non sia più in grado di gestire il carico di lavoro. L'adozione di un file YAML configurabile semplificherà il processo di scalabilità, poiché i client e i server comunicano direttamente tramite Pub/Sub, evitando complessità nella gestione delle connessioni tra di loro;
- **Ottimizzazione dei costi**: è essenziale effettuare uno studio approfondito delle risorse utilizzate per identificare opportunità di ottimizzazione dei costi. Con l'aumento del numero di client e server, è cruciale evitare che i costi del servizio crescano in modo esponenziale. Ottimizzare la configurazione delle risorse e adottare soluzioni più economiche, come l'uso di nodi di dimensioni inferiori e strategie di gestione dei dati più efficienti, può contribuire significativamente a mantenere i costi sotto controllo.

Queste migliorie proposte sono fondamentali per assicurare la sostenibilità a lungo termine e l'efficienza economica del servizio. È quindi essenziale pianificare attentamente l'espansione futura e ottimizzare continuamente le risorse utilizzate.

6 Riferimenti

1. Google Cloud Platform, Wikipedia (2024)
2. Cos'è Kubernetes?, Kubernetes (2020)
3. Cos'è Elasticsearch?, AWS (2023)
4. Che cos'è Pub/Sub?, Google (2024)
5. A cosa serve Kibana?, Alessandro Fiori (2022)
6. Prezzi di Google Kubernetes Engine, Google (2024)
7. Prezzi di Pub/Sub, Google (2024)
8. Prezzi di Compute Engine, Google (2024)
9. Repository GitHub progetto