

Colombo Lorenzo 885895

Corso di Architettura Dati:

Progettazione e analisi di un sistema di logging
distribuito su cluster Cassandra



Sommario

1. Introduzione	3
2. Architettura del Sistema	4
2.1 Generatore di log (Servizi simulati).....	4
2.2 Cluster Cassandra distribuito:	4
2.3 Client per interrogazione e analisi:	5
3. Setup del Cluster Cassandra	6
3.1 Configurazione dei nodi	6
3.2 Configurazione del keyspace e della tabella	7
4. Caricamento dei Dati	9
4.1 Script di Generazione dei Log	9
4.2 Esecuzione Distribuita	9
4.3 Configurazione del Cluster	10
4.4 Inserimento dei Dati	10
5. Test di Lettura e Scrittura	11
5.1 Operazioni Svolte.....	11
5.2 Risultati ottenuti.....	11
5.3 Analisi dati ottenuti	12
6. Gestione dei Fault (Caduta Nodo)	14
6.1 Procedura eseguita.....	14
6.2 Risultati.....	14
7. Conclusioni	16

1. Introduzione

Il presente progetto ha l'obiettivo di simulare un sistema di logging distribuito utilizzando Apache Cassandra. L'idea è emulare il flusso continuo di log generati da diversi servizi, archiviandoli in un cluster Cassandra per analizzarne le capacità di scalabilità, replica e tolleranza ai guasti.

Il sistema sarà testato inizialmente con due nodi, per poi aggiungerne dinamicamente un terzo e osservare come il cluster reagisce in termini di prestazioni e distribuzione dei dati. Verranno inoltre condotte prove di carico e interrogazione con diverse configurazioni, valutando l'impatto di politiche di query differenti e la gestione dei fallimenti di nodo.

Questo progetto intende dimostrare in modo pratico l'efficacia di Cassandra nella gestione di dati distribuiti e nell'adattamento a scenari dinamici e reali.

2. Architettura del Sistema

Il progetto si basa su un'architettura distribuita che simula un sistema di raccolta e gestione di log da più servizi, sfruttando Cassandra per garantire scalabilità e affidabilità nella memorizzazione dei dati.

2.1 Generatore di log (Servizi simulati)

Il sistema prevede l'utilizzo di script Python che simulano l'attività di più servizi distribuiti, ciascuno dei quali genera in modo continuo messaggi di log. Questi log contengono informazioni standardizzate, come:

- Istante in cui è avvenuto l'evento (timestamp);
- Nome del servizio;
- Livello di severità (INFO, WARN, ERROR);
- Messaggio descrittivo.

Questi log vengono inviati direttamente a Cassandra, simulando uno scenario realistico di raccolta dati in tempo reale da più punti della rete.

2.2 Cluster Cassandra distribuito:

Il cuore dell'architettura è rappresentato da un cluster Cassandra inizialmente composto da due nodi, configurati tramite Docker (cassandra1, cassandra2). I nodi comunicano tra loro tramite *Gossip Protocol*, condividendo informazioni sulla topologia e lo stato del cluster.

Ogni nodo partecipa attivamente alla replica e alla gestione dei dati, e viene utilizzato l'EndpointSnitch di tipo GossipingPropertyFileSnitch, che permette a Cassandra di essere consapevole della topologia di rete e ottimizzare le repliche di conseguenza.

Successivamente, viene aggiunto dinamicamente un terzo nodo (cassandra3) al cluster già attivo, per testare le capacità di scaling orizzontale di Cassandra e verificare la redistribuzione automatica dei dati.

2.3 Client per interrogazione e analisi:

Per effettuare query e analisi sui dati raccolti, vengono utilizzati diversi strumenti, inclusi metodi automatizzati per facilitare l'interazione e la visualizzazione dei dati:

- **Interfaccia cqlsh:** Utilizzata per ispezionare le tabelle direttamente da un container Docker, permettendo di eseguire manualmente query di selezione e analizzare i dati in tempo reale.
- **Script Python con driver ufficiale Cassandra (cassandra-driver):** Automatizza l'esecuzione delle query e la misurazione delle performance, come i tempi di inserimento e la latenza delle query. Gli script Python interagiscono direttamente con il cluster Cassandra, raccogliendo e analizzando i dati in modo efficiente.

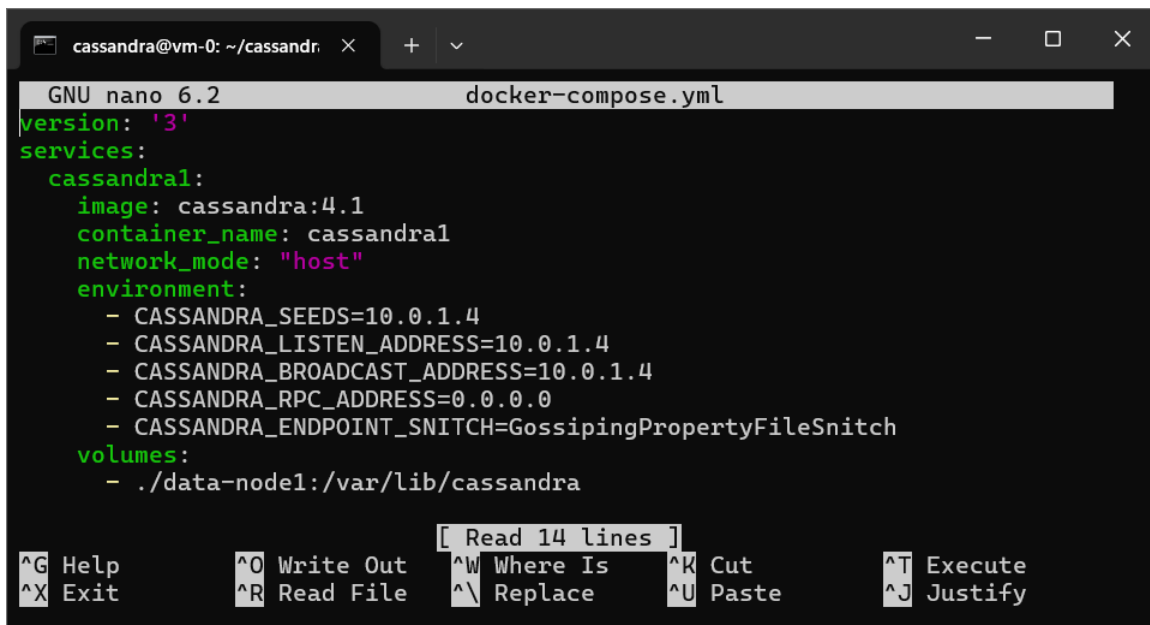
Questa architettura distribuita permette di monitorare e analizzare facilmente i dati e le prestazioni del sistema in tempo reale.

3. Setup del Cluster Cassandra

L'infrastruttura centrale del progetto è costituita da un cluster Apache Cassandra formato inizialmente da due nodi (cassandra1, cassandra2) distribuiti su due macchine virtuali Linux distinte, con successiva espansione a un terzo nodo (cassandra3) su una terza macchina virtuale distinta.

3.1 Configurazione dei nodi

Ogni nodo è configurato attraverso un file docker-compose.yml, differenziato per IP e nome contenitore. Tutti i nodi appartengono alla stessa rete e condividono lo stesso nodo seed (cassandra1).



```
GNU nano 6.2 docker-compose.yml
version: '3'
services:
  cassandra1:
    image: cassandra:4.1
    container_name: cassandra1
    network_mode: "host"
    environment:
      - CASSANDRA_SEEDS=10.0.1.4
      - CASSANDRA_LISTEN_ADDRESS=10.0.1.4
      - CASSANDRA_BROADCAST_ADDRESS=10.0.1.4
      - CASSANDRA_RPC_ADDRESS=0.0.0.0
      - CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch
    volumes:
      - ./data-node1:/var/lib/cassandra
```

[Read 14 lines]

^G Help	^O Write Out	^W Where Is	^K Cut	^T Execute
^X Exit	^R Read File	^\ Replace	^U Paste	^J Justify

```
cassandra@vm-2: ~/cassandr
GNU nano 6.2 docker-compose.yml
version: '3'
services:
  cassandra2:
    image: cassandra:4.1
    container_name: cassandra2
    network_mode: "host"
    environment:
      - CASSANDRA_SEEDS=10.0.1.4
      - CASSANDRA_LISTEN_ADDRESS=10.0.1.5
      - CASSANDRA_BROADCAST_ADDRESS=10.0.1.5
      - CASSANDRA_BROADCAST_RPC_ADDRESS=10.0.1.5
      - CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch
    volumes:
      - ./data-node2:/var/lib/cassandra

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify
```

```
cassandra@vm-0: ~/cassandr
cassandra@vm-0:~/cassandr$ sudo docker exec -it cassandra1 nodetool status
Datacenter: dc1
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address      Load          Tokens   Owns (effective)  Host ID                               Rack
UN  10.0.1.4     140 KiB       16       100.0%            7135a3fd-2e6c-48e4-9886-e84fc3ede09c rack1
UN  10.0.1.5     139.68 KiB   16       100.0%            4b0cf3c4-ab08-41df-bb39-fba6d9d5a063 rack1
```

```
cqlsh:log_keyspace> select * from logs ;

id | level | message | service | timestamp
---|---|---|---|---
c878c07f-52b3-4d58-89e4-8ceb4c383837 | ERROR | Log inviato da 10.0.1.6 | notifications | 2025-05-20 14:28:25.830000+0000
1642dfec-a399-44d5-b6d1-5b46e05c97a7 | INFO | Log message inviato da 10.0.1.5 | auth | 2025-05-20 14:28:11.032000+0000
1c5986ce-769b-41b1-a1cb-7414d23e4b57 | INFO | Log inviato da 10.0.1.6 | billing | 2025-05-20 14:28:17.763000+0000
9af2f436-cda0-4eee-8b6f-8ae952180ec2 | ERROR | Log inviato da 10.0.1.6 | notifications | 2025-05-20 14:28:31.564000+0000
941f0720-b432-4a2a-81a9-b04b5c0a538e | INFO | Log message inviato da 10.0.1.5 | billing | 2025-05-20 14:27:28.300000+0000
bb121109-108e-4449-be5f-69deb09a9adb | WARNING | Log inviato da 10.0.1.6 | billing | 2025-05-20 14:27:57.582000+0000
bdfd1eee-b5fe-4739-9d56-0fbceab77e17 | WARNING | Log message inviato da 10.0.1.5 | auth | 2025-05-20 14:27:31.096000+0000
79472f3e-ebab-4252-94bc-65bf08abb6f2 | INFO | Log message inviato da 10.0.1.5 | auth | 2025-05-20 14:27:49.900000+0000
d711f81d-e6e6-41ed-a319-fb2a7a7abb38 | WARNING | Log inviato da 10.0.1.6 | billing | 2025-05-20 14:28:05.279000+0000
3b8427ed-8bab-4171-980d-c4c74a376404 | ERROR | Log message inviato da 10.0.1.5 | billing | 2025-05-20 14:27:45.568000+0000
b53dd42c-4db2-440b-b091-16a2289b58e1 | INFO | Log message inviato da 10.0.1.5 | auth | 2025-05-20 14:28:22.375000+0000
44aa2ba5-6c3e-412f-ab0a-537db09ca71f | ERROR | Log message 0 | notifications | 2025-05-20 14:26:20.635000+0000
0b85218b-5404-4fe7-8ffa-b0a26b7a3467 | ERROR | Log message inviato da 10.0.1.5 | auth | 2025-05-20 14:27:36.121000+0000
46ebdf7f-9978-4b5a-a205-8d2c3affa445 | WARNING | Log message inviato da 10.0.1.5 | notifications | 2025-05-20 14:28:15.639000+0000
35b61b07-24bb-433f-ac2c-da51b47086b7 | WARNING | Log message inviato da 10.0.1.5 | billing | 2025-05-20 14:27:57.837000+0000
8458becc-572c-43ab-a2ca-6dd89757eec3 | ERROR | Log message inviato da 10.0.1.5 | notifications | 2025-05-20 14:28:07.439000+0000
73882ba2-fc30-43c2-84ec-43cc90bb5fab | ERROR | Log inviato da 10.0.1.6 | auth | 2025-05-20 14:28:13.516000+0000

(17 rows)
cqlsh:log_keyspace> |
```

3.2 Configurazione del keyspace e della tabella

Per memorizzare i log generati dai servizi simulati, è stato creato un keyspace dedicato denominato log_keyspace. La configurazione del keyspace prevede l'utilizzo della strategia di replica SimpleStrategy con un Replication Factor pari a 3, in modo da garantire la replicazione dei dati su tutti i nodi del cluster. Questa scelta permette di ottenere un buon compromesso tra disponibilità, tolleranza ai guasti (anche eventuali

catastrofi naturali) e durabilità dei dati, assicurando che ogni log venga replicato su tutti e tre i nodi attivi del cluster.

La creazione del keyspace è stata effettuata mediante:

```
CREATE KEYSPACE IF NOT EXISTS log_keyspace WITH replication = {  
  'class': 'SimpleStrategy',  
  'replication_factor': 3  
};
```

Successivamente, è stata definita una tabella logs all'interno del keyspace, con lo scopo di raccogliere i log generati dai diversi servizi. La struttura della tabella prevede i seguenti campi:

- id: identificatore univoco (UUID) del log, utilizzato come chiave primaria.
- service: nome del servizio che ha generato il log.
- level: livello di severità del log (es. INFO, WARN, ERROR).
- message: messaggio di log.
- timestamp: data e ora in cui il log è stato generato.

La creazione della tabella è stata eseguita con il seguente comando:

```
CREATE TABLE IF NOT EXISTS logs (  
  id UUID PRIMARY KEY,  
  service TEXT,  
  level TEXT,  
  message TEXT,  
  timestamp TIMESTAMP  
);
```

Con una configurazione di questo tipo si ha un database adatto a uno scenario di logging distribuito, in cui i log vengono continuamente scritti e letti su diversi nodi del cluster. La presenza di un replication factor elevato assicura che i dati siano resistenti alla perdita anche in caso di guasto di uno o più nodi.

4. Caricamento dei Dati

Per testare la distribuzione, la replica e la resilienza del cluster Cassandra, è stato sviluppato uno script Python che simula il comportamento di un sistema di logging distribuito. Tale script genera e carica continuamente dati di log nel keyspace `log_keyspace`.

4.1 Script di Generazione dei Log

Lo script, realizzato in Python tramite la libreria `cassandra-driver`, simula la generazione di log da parte di microservizi. Per ogni iterazione, vengono scelti casualmente:

- uno tra i servizi: `auth`, `billing`, `notifications`
- uno tra i livelli di log: `INFO`, `WARNING`, `ERROR`
- un messaggio identificativo sequenziale

Ogni log è corredato da un identificatore univoco (UUID) e un timestamp UTC, ed è inserito nella tabella `logs` del keyspace `log_keyspace` tramite una query `INSERT`.

4.2 Esecuzione Distribuita

Lo script è stato caricato ed eseguito contemporaneamente su due nodi del cluster:

- Nodo 1 (IP: `10.0.1.5`)
- Nodo 2 (IP: `10.0.1.6`)

Questa esecuzione parallela simula un contesto realistico in cui più fonti indipendenti inviano log al sistema. In questo modo si genera una scrittura concorrente distribuita, che consente di osservare la capacità del cluster di gestire accessi simultanei da più client, verificando il corretto bilanciamento del carico tra i nodi attivi e il comportamento del meccanismo di replica in presenza di scritture parallele

4.3 Configurazione del Cluster

Inizialmente il cluster Cassandra è composto da due nodi attivi:

- 10.0.1.4
- 10.0.1.5

Un terzo nodo (10.0.1.6) verrà aggiunto dinamicamente in seguito, verificando l'effettiva capacità di Cassandra di ridistribuire i dati nel cluster.

4.4 Inserimento dei Dati

Ogni log è inserito mediante la seguente query CQL:

```
session.execute("""
    INSERT INTO logs (id, service, level, message, times>
    VALUES (%s, %s, %s, %s, %s)
    """, (log_id, service, level, message, timestamp))
```

L'esecuzione contemporanea su due nodi ha prodotto una quantità significativa di log distribuiti su più partizioni.

5. Test di Lettura e Scrittura

Per valutare le prestazioni e la scalabilità del sistema Cassandra, sono stati eseguiti test di scrittura e lettura, su un keyspace con replication factor di 3, variando il numero di nodi attivi nel cluster (da 2 a 3) e utilizzando differenti livelli di consistenza: ONE, QUORUM e ALL.

Ho testato ONE, QUORUM e ALL perché coprono l'intero range possibile di consistenza: bassa, media e alta. Sono anche i livelli più rappresentativi e usati in contesti reali. Altri livelli (es. TWO, THREE) sono varianti meno generali e utili solo in casi specifici.

5.1 Operazioni Svolte

- Scrittura: sono stati generati e inseriti 500.000 log per ciascun livello di consistenza (ONE, QUORUM, ALL), utilizzando uno script Python.
- Lettura: sono state eseguite una query su campo secondario (WHERE service = 'auth') con ALLOW FILTERING, e una query su chiave primaria (WHERE id = 'addcd3c3-87f5-406e-919a-6fd6a472b278') per ciascun livello di consistenza e per entrambe le configurazioni di cluster (3 nodi e 2 nodi attivi).

5.2 Risultati ottenuti

- Scrittura:

NODI ATTIVI	CONSISTENCY LEVEL	NUMERO LOG SCRITTI	TEMPO TOTALE (S)	TEMPO MEDIO PER INSERIMENTO (MS)
3	ONE	500.000	282.694	0.57
3	QUORUM	500.000	371.087	0.74
3	ALL	500.000	381.429	0.76
2	ONE	500.000	271.340	0.54
2	QUORUM	500.000	376.691	0.75
2	ALL	ERROR	-	-

- Lettura - Query su service (ALLOW FILTERING):

NODI ATTIVI	CONSISTENCY LEVEL	TEMPO (MS)	RISULTATI TROVATI	ESITO
3	ONE	3860.11	200.005	Successo
3	QUORUM	5338.84	200.005	Successo
3	ALL	6123.41	200.005	Successo
2	ONE	3583.05	200.005	Successo
2	QUORUM	5241.89	200.005	Successo
2	ALL	33.20	0	Errore

- Lettura - Query su chiave primaria id:

NODI ATTIVI	CONSISTENCY LEVEL	TEMPO (MS)	RISULTATI TROVATI	ESITO
3	ONE	5.09	1	Successo
3	QUORUM	15.19	1	Successo
3	ALL	12.30	1	Successo
2	ONE	1.26	1	Successo
2	QUORUM	1.75	1	Successo
2	ALL	51.66	0	Errore

5.3 Analisi dati ottenuti

- Scrittura:
 - I tempi aumentano all'aumentare della consistenza richiesta.
 - ONE è significativamente più veloce, poiché richiede la risposta da un solo nodo.
 - QUORUM e ALL sono più lenti ma forniscono maggiore affidabilità.
- Lettura:
 - Prestazioni: Le query su chiave primaria (id = ...) sono significativamente più veloci rispetto a quelle su attributi

secondari (service = 'auth' con ALLOW FILTERING), indipendentemente dal livello di consistenza.

- Consistenza: Le query con ONE sono le più rapide ma meno affidabili. QUORUM rappresenta un buon compromesso tra prestazioni e consistenza. Le query con ALL falliscono se non tutti i nodi sono disponibili.
 - Efficienza delle query: Le query che non utilizzano la chiave primaria, come quelle con ALLOW FILTERING, comportano una scansione completa dei dati, risultando in prestazioni inferiori in quanto Cassandra è ottimizzato per accessi tramite chiave primaria e l'uso di ALLOW FILTERING dovrebbe essere limitato a casi specifici e con dataset di dimensioni contenute.
- Scalabilità e fault tolerance:
 - Cassandra continua a funzionare correttamente anche in presenza della caduta di un nodo, purché non si richieda consistenza ALL (in particolare con replication factor = 3 non riesce a trovare alcun dato).
 - La distribuzione dei dati e la replica permettono di bilanciare carico e tollerare guasti, ma è importante scegliere il giusto livello di consistenza in base ai requisiti del sistema.

6. Gestione dei Fault (Caduta Nodo)

Per testare la resilienza del sistema in scenari di guasto, è stata effettuata una simulazione della caduta di un nodo nel cluster Cassandra composto da 3 nodi, con replication factor = 3. Lo scopo era verificare se il sistema fosse in grado di continuare a garantire disponibilità e corretta replica dei dati anche in condizioni di degrado.

6.1 Procedura eseguita

- Simulazione del guasto del nodo cassandra3 tramite comando Docker stop;
- Conferma mediante nodetool status della corretta caduta del nodo;
- Esecuzione di query e scritture con diversi livelli di consistenza per osservare il comportamento del cluster con un nodo non disponibile;
- Riavvio del nodo con docker start;
- Conferma mediante nodetool status del corretto riavvio del nodo.

6.2 Risultati

CONSISTENCY LEVEL	SCRITTURA CON 1 NODO DOWN	LETTURA CON 1 NODO DOWN	COMPORAMENTO ATTESO	COMPORAMENTO OSSERVATO
ONE	Funziona	Funziona	Basta una replica disponibile	Funziona
QUORUM	Funziona	Funziona	Richiede 2 repliche su 3	Funziona
ALL	Fallisce	Fallisce	Richiede tutte le repliche disponibili	UnavailableException

Le operazioni con ONE e QUORUM sono state completate con successo, a conferma della tolleranza ai guasti garantita da Cassandra. Le operazioni con ALL, invece, hanno restituito un'eccezione di indisponibilità (UnavailableException), poiché il numero di nodi disponibili era insufficiente a garantire la consistenza richiesta.

Dopo il riavvio del nodo caduto, il cluster ha automaticamente rilevato e reintegrato il nodo nel sistema, tramite il protocollo di gossip.

7. Conclusioni

Questo progetto ha permesso di esplorare in modo approfondito le potenzialità di Apache Cassandra in un contesto distribuito, focalizzandosi su un caso d'uso realistico: la raccolta e gestione di log provenienti da più servizi. L'implementazione del cluster, prima con due nodi e successivamente con l'aggiunta di un terzo, ha evidenziato l'efficacia della scalabilità orizzontale del sistema, che è stato in grado di ridistribuire automaticamente i dati senza interruzioni del servizio.

L'analisi delle prestazioni ha mostrato come Cassandra privilegi la velocità nelle operazioni di scrittura, risultando particolarmente adatto a scenari con grandi volumi di dati in ingresso. I test sulle letture, condotti con diversi livelli di consistenza (ONE, QUORUM, ALL), hanno evidenziato la necessità di un compromesso tra rapidità e coerenza: ONE offre prestazioni elevate ma minore affidabilità, mentre ALL garantisce la massima accuratezza a discapito del tempo di risposta.

La tolleranza ai guasti è emersa chiaramente durante la simulazione della caduta di un nodo: Cassandra ha mantenuto la disponibilità delle operazioni a livelli di consistenza più permissivi, confermando la robustezza della sua architettura distribuita. L'aggiunta del terzo nodo ha inoltre dimostrato come il sistema possa essere esteso dinamicamente senza disservizi.

Nel complesso, Cassandra si è rivelato un sistema estremamente adatto alla gestione di dati distribuiti e ad alta disponibilità. La possibilità di configurare il livello di consistenza rende il sistema flessibile, permettendo di adattarlo alle esigenze specifiche di ogni applicazione, sia essa orientata alla velocità, all'affidabilità o a un bilanciamento tra le due. Queste caratteristiche rendono Cassandra una scelta solida per applicazioni moderne che richiedono scalabilità, resilienza e prestazioni affidabili.