Come trasmettere in maniera efficiente eventi DML su AWS

14 Aprile 2023 - 8 min. read

Amazon Kinesis Data Streams | *AWS Database Migration Service*

In passato, molti workload disponevano di un proprio database grande e monolitico, a cui non solo l'applicazione, ma anche gli strumenti di reporting e il supporto tecnico si collegavano per eseguire query.

Sebbene questo sia vero ancora oggi, le aziende si stanno sempre più muovendo verso l'archiviazione delle singole informazioni su più data source e server. Una buona prassi prevede che solo l'applicazione principale sia in grado di accedere al database, gli strumenti di reporting utilizzino i dati archiviati su un'istanza separata e il monitoraggio e l'analisi dei dati siano eseguiti aggregando i dati provenienti da fonti diverse.

Per far ciò, abbiamo bisogno di trasmettere le modifiche che si verificano sul nostro database verso una o più destinazioni. Oggi daremo un'occhiata a come farlo su AWS.

AWS Database Migration Service (DMS) è un potente strumento per la migrazione dei dati tra varie piattaforme di database. Una delle caratteristiche distintive di AWS DMS è la sua funzionalità Change Data Capture (CDC), che consente lo **streaming in tempo reale delle modifiche apportate a un database di origine verso un database di destinazione**.

Utilizzando AWS DMS, hai la possibilità di collegare un database di destinazione direttamente come endpoint, oppure di utilizzare Amazon Kinesis Data Streams per acquisire ed elaborare i dati in streaming.

Ecco alcune differenze tra i due approcci:

1. Latenza: durante lo streaming dei dati direttamente a un database di destinazione, potrebbe esserci una certa latenza nell'elaborazione e nella scrittura dei dati. Con

Kinesis Data Streams, i dati vengono acquisiti ed elaborati in tempo reale, senza ritardi nell'elaborazione.

2. Scalabilità: Kinesis Data Streams è progettato per gestire grandi volumi di dati in streaming e può essere ridimensionato automaticamente per adattarsi all'aumento del traffico. Quando si effettua lo streaming dei dati direttamente in un database di destinazione, potrebbe essere necessario ridimensionare manualmente il database per gestire i picchi di traffico.
3. Flessibilità: con Kinesis Data Streams puoi facilmente elaborare e analizzare i dati in streaming utilizzando vari servizi AWS, come AWS Glue o AWS Lambda. Durante lo streaming dei dati direttamente in un database di destinazione, potresti avere opzioni limitate per l'elaborazione e l'analisi dei dati.
4. Costo: l'utilizzo di Kinesis Data Streams può comportare costi aggiuntivi per l'elaborazione e l'archiviazione dei dati in streaming, nonché eventuali servizi AWS associati utilizzati per l'elaborazione e l'analisi. Lo streaming di dati direttamente a un database di destinazione potrebbe non comportare costi aggiuntivi, ma potrebbe essere necessario considerare il costo del ridimensionamento del database per gestire l'aumento del traffico.

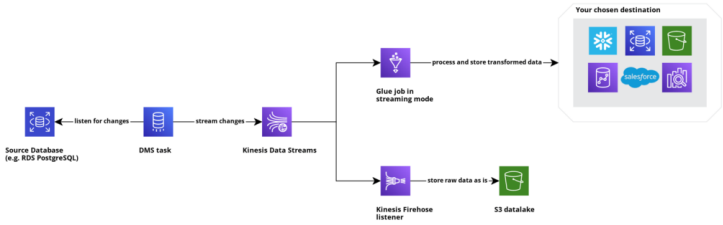
Nel complesso, entrambi gli approcci presentano vantaggi e svantaggi e la scelta migliore dipende dal caso d'uso e dai requisiti specifici.

In questo articolo esploreremo la possibilità di elaborare eventi di inserimento/aggiornamento/eliminazione real-time con l'aiuto di Amazon Kinesis Data Streams.

Setup di una soluzione di streaming di eventi DML su AWS

Creiamo un proof of concept per testare la soluzione di streaming CDC con DMS e Kinesis Data Streams. L'idea è avere un **processo automatizzato che ci offra un modo semplice per replicare le modifiche che si verificano su un database di origine su uno o più motori di destinazione**.

Questo è un diagramma di ciò che costruiremo:



L'ingestion

La prima cosa che dobbiamo fare, se vogliamo abilitare CDC, è **configurare il nostro database di origine** per rendere disponibili tutte le informazioni necessarie al DMS per catturare nuovi eventi. Per molti engine, questo significa eseguire una serie di query che sono ben descritte nella documentazione ufficiale di AWS.

Dopo aver configurato il database di origine, **creiamo il nostro Kinesis Data Stream**.

Questo passaggio è piuttosto semplice; per richiedere la creazione sono richiesti pochi parametri. In particolare, dobbiamo solo decidere quale sarà la modalità di scaling del nostro data stream:

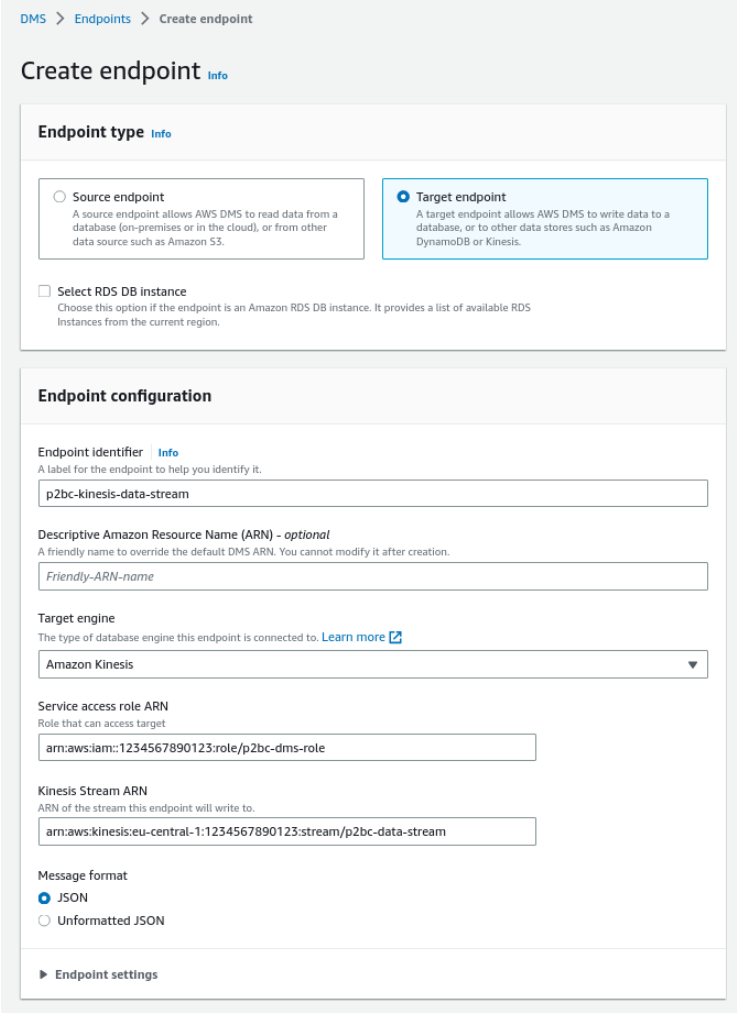
1. Scegliendo l'on-demand delegheremo le operazioni di scaling ad AWS.
2. Scegliendo "provisioned" sarà necessario fornire il **numero di shard** (ovvero il throughput di lettura e scrittura) che rappresenta la dimensione del nostro stream.

Nel mio caso ho scelto l'opzione on-demand. Tieni a mente il concetto di sharding, tornerà più avanti in questo articolo.

Ora che abbiamo configurato i sistemi di origine e di destinazione, dobbiamo **creare su AWS DMS l'istanza e gli endpoint**, che sono fondamentalmente un insieme di configurazioni che il servizio utilizza per interagire con i vari sistemi.

La configurazione di un endpoint DMS è davvero semplice: prima devi scegliere se stai creando un endpoint di origine o di destinazione, quindi specifichi il tipo di engine a cui l'endpoint è connesso e, infine, a seconda dell'engine selezionato, devi fornire alcuni parametri che consentono di configurare la connessione.

Ecco un esempio del form di creazione per l'endpoint Kinesis Data Stream:



AWS DMS richiede un ruolo IAM per poter interagire con Kinesis. Questo ruolo dovrebbe avere almeno queste autorizzazioni:

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Action": [
                                "kinesis:PutRecord",
                                        "kinesis:PutRecords",
                                        "kinesis:DescribeStream"
                                ],
                                "Resource": [
                                        "<your delivery stream arn>"
                                ],
                                "Effect": "Allow"
                }
        ]
}
```

Ora l'ultima risorsa da **creare** sulla console AWS DMS è **il task effettivo che si occuperà di copiare i dati dalla nostra origine a Kinesis**.

Scegli l'istanza di replica DMS in cui desideri creare il tuo task, seleziona gli endpoint creati in precedenza e configura il tipo di migrazione che desideri eseguire: puoi avere una migrazione che acquisisce uno snapshot dei dati archiviati sul motore di origine, oppure è possibile abilitare la Change Data Capture in modo che i nuovi eventi vengano replicati automaticamente nel motore di destinazione. Oppure entrambe le cose.

I replication task hanno tonnellate di parametri di configurazione che puoi modificare per aumentare l'efficienza, la stabilità e la velocità, ma non è questo l'articolo in cui andremo ad approfondirli. Per questa POC non è stato necessario cambiare nulla e tutti i parametri sono stati lasciati al loro valore di default.

La creazione dell'attività richiederà alcuni minuti. Nel frattempo possiamo concentrarci sulla creazione delle risorse per...

L'elaborazione

Nel diagramma dell'architettura ho inserito AWS Glue come servizio di elaborazione. AWS Glue si integra nativamente con molti data warehouse e dà la possibilità di elaborare grandi dataset grazie a Spark. Inoltre, i Glue Job possono essere configurati in modalità streaming, il che significa che sono sempre attivi ed elaborano i dati man mano che arrivano.

Qui puoi trovare un piccolo frammento di codice che può essere una buona base di partenza per la creazione del job:

```python
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.context import SparkContext
from pyspark.sql.functions import from_json, col, lit, current_timestamp
from pyspark.sql import DataFrame

args = getResolvedOptions(sys.argv, ["JOB_NAME", "kinesis_stream_arn"])
job_run_id = args['JOB_RUN_ID']
sc = SparkContext()
glue_context = GlueContext(sc)
spark = glue_context.spark_session
job = Job(glue_context)

def process_batch(data_frame: DataFrame, _batch_id: int):
    job.init(args["JOB_NAME"], args)
    if data_frame.count() <= 0:
        job.commit()
        return
    data_frame.printSchema()
    data_frame.show()
    # Do your processing here
    job.commit()


amazon_kinesis_dataframe = glue_context.create_data_frame.from_options(
    connection_type="kinesis",
    connection_options={
        "typeOfData": "kinesis",
        "streamARN": args["kinesis_stream_arn"],
        "classification": "json",
        "startingPosition": "TRIM_HORIZON",
        "inferSchema": "false",
        "avoidEmptyBatches": "true",
        "schema": "`data` string, `metadata` STRUCT<`timestamp`: TIMESTAMP, `record-type`: STRING,
`operation`: STRING, `partition-key-type`: STRING, `schema-name`: STRING, `table-name`: STRING,
`transaction-id`: BIGINT> NOT NULL"
    },
    transformation_ctx="amazon_kinesis_dataframe",
)

glue_context.forEachBatch(
    frame=amazon_kinesis_dataframe,
    batch_function=process_batch,
    options={
        "windowSize": "100 seconds",
        "checkpointLocation": args["TempDir"] + "/" + args["JOB_NAME"] + "/checkpoint/",
    },
)
```

Puoi copiare il codice dalla GitHub gist.

Questo codice crea sostanzialmente un collegamento tra il Glue Job e un Data Frame che ha come sorgente Kinesis Data Stream. Questo data frame verrà aggiornato ogni 100 secondi e ogni sua nuova versione verrà passata alla funzione *process_batch*. Questa funzione si occupa dell'elaborazione dei nuovi dati.

Durante la creazione del Kinesis Data Frame, possiamo fornire una struttura dei dati in ingresso. Questo schema deve corrispondere al formato dei dati che DMS ci fornisce, che è un JSON che contiene due chiavi:

- data, un altro JSON che contiene i valori dei record creati/aggiornati/eliminati
- metadata, sempre un JSON che contiene informazioni relative all'evento come lo schema e il nome della tabella da cui è scaturito l'evento, la data in cui si è verificato, ...

Come puoi vedere, ho fornito solo uno schema "strutturato" per la parte dei metadati. Questo perché non possiamo conoscere il formato della chiave "data" prima di conoscere il nome dello schema e della tabella.

Dopo aver avviato il nostro nuovo Glue job, dovremmo essere in grado di vedere alcuni log su CloudWatch simili a questi:

```
▼ 2023-03-31T14:36:43.475+02:00       root |-- data: string (nullable = true) |-- metadata: struct (nullable = true) | |-- timestamp: timestamp (nullab…

  root
   |-- data: string (nullable = true)
   |-- metadata: struct (nullable = true)
   |    |-- timestamp: timestamp (nullable = true)
   |    |-- record-type: string (nullable = true)
   |    |-- operation: string (nullable = true)
   |    |-- partition-key-type: string (nullable = true)
   |    |-- schema-name: string (nullable = true)
   |    |-- table-name: string (nullable = true)
   |    |-- transaction-id: long (nullable = true)

▼ 2023-03-31T14:36:43.945+02:00       +--------------------+--------------------+ | data| metadata| +--------------------+--------------------+ |{"INVO…

  +--------------------+--------------------+
  |                data|            metadata|
  +--------------------+--------------------+
  |{"INVOICE_ID":18,...|{2023-03-31 12:33...|
  |{"INVOICE_ID":19,...|{2023-03-31 12:33...|
  |{"INVOICE_ID":20,...|{2023-03-31 12:33...|
  |{"ORDER_ID":129,"...|{2023-03-31 12:33...|
  |{"ORDER_ID":130,"...|{2023-03-31 12:33...|
  |{"ORDER_ID":131,"...|{2023-03-31 12:33...|
  |{"ORDER_ID":132,"...|{2023-03-31 12:33...|
  |{"ORDER_ID":133,"...|{2023-03-31 12:33...|
  |{"ORDER_ID":134,"...|{2023-03-31 12:33...|
  |{"ORDER_ID":135,"...|{2023-03-31 12:33...|
  +--------------------+--------------------+
```

Infine, volevo salvare gli eventi su S3 così come provengono da DMS, in maniera da poter decidere di tornare indietro e rielaborare tutto fino a un certo punto nel tempo. Fortunatamente disponiamo già del nostro Data Stream su Kinesis e possiamo collegarvi più consumer, dunque possiamo facilmente creare un Kinesis Firehose Delivery Stream per aggregare gli eventi e scaricarli su un bucket S3.

Possibili criticità

Quando abbiamo provato per la prima volta questa soluzione, ci siamo imbattuti in un problema di prestazioni legato alla scalabilità del nostro Data Stream: come abbiamo discusso in precedenza, Kinesis utilizza un concetto di **shard** come unità di misura dello scaling. Quando il producer produce un nuovo evento, deve fornire lo shard in cui deve andare l'evento o utilizzare un ID casuale per far decidere a Kinesis.

DMS utilizza sempre lo stesso valore.

In questo modo, utilizza sempre lo stesso shard, quindi anche avendo configurato la modalità di scaling del data stream come on-demand, non vi sarà un'effettiva scalabilità della soluzione a causa di DMS.

Il target endpoint di DMS per Kinesis dispone di un'impostazione che puoi abilitare chiamata PartitionIncludeSchemaTable che permette a DMS di utilizzare il nome dello schema e della tabella come partition key (e quindi come shard).

Questo non è ancora sufficiente per avere una buona scalabilità secondo me: se stai migrando molte tabelle, avrai molti shard, ognuno dedicato alla propria tabella. Se su una delle tue tabelle dovesse verificarsi un picco di eventi, Kinesis non scalerà sufficientemente poichè DMS continuerà a inviare dati sullo stesso shard.

Fortunatamente questo diventa un problema solo in caso di tabelle sorgente molto grandi (stiamo parlando di milioni di eventi ogni giorno). Se il tuo workload è più standard, anche con questo problema DMS probabilmente non vedrai peggiorare le tue prestazioni.

Speriamo che AWS ci metta una patch al più presto :)

Conclusione

Questa infrastruttura è piuttosto semplice da configurare e offre sicuramente molti vantaggi: puoi elaborare i tuoi dati così come vengono generati mantenendo una copia grezza su uno storage separato, è scalabile e flessibile; siccome i Glue Job configurati in modalità streaming non sono economici, puoi sempre sostituirli con una funzione Lambda ottimizzando i costi se non si verificano molti eventi sul database sorgente.

Hai giù utilizzato questa soluzione o una simile? Raccontaci la tua esperienza!

Arrivederci al prossimo articolo su **Proud2beCloud**!

About Proud2beCloud

Proud2beCloud è il blog di beSharp, APN Premier Consulting Partner italiano esperto nella progettazione, implementazione e gestione di infrastrutture Cloud complesse e servizi AWS avanzati. Prima di essere scrittori, siamo Solutions Architect che, dal 2007, lavorano quotidianamente con i servizi AWS. Siamo innovatori alla costante ricerca della soluzione più all'avanguardia per noi e per i nostri clienti. Su Proud2beCloud condividiamo regolarmente i nostri migliori spunti con chi come noi, per lavoro o per passione, lavora con il Cloud di AWS. Partecipa alla discussione!

Mattia Costamagna

Ingegnere DevOps e sviluppatore cloud-native @ beSharp. Adoro passare il mio tempo libero a leggere romanzi e ascoltare musica rock e blues degli anni '70. Sempre alla ricerca di nuove tecnologie e framework da testare e utilizzare. La birra artigianale è il mio carburante!
