

AWS Glue Elastic Views! Framework per ETL e Aggregazione (quasi) senza codice

14 Aprile 2021 - 12 min. read

Amazon DynamoDB

Amazon S3

AWS Glue Elastic Views

Data and Analytics

Data Ingestion

Data Lake

Dataset

ETL

Serverless



Introduzione

L'**ETL** rappresenta uno step fondamentale in un processo di Machine Learning in quanto è il trampolino di lancio su cui si basa tutto il set di dati per la definizione del modello, per questo i **data scientist e gli esperti MLOps pianificano attentamente i job e le pipeline per gestire l'estrazione dei dati dai database**, spesso di natura diversa, **pulendo e normalizzando i dati** ed infine, **generando un data lake** per migliorare ulteriormente i dati durante il processo di indagine.

Di solito, questo processo prevede diversi passaggi, il coordinamento della loro esecuzione, l'accesso a diversi database con diverse tecnologie, la preparazione di molti script, la conoscenza di diversi linguaggi per interrogare i dati rilevanti e così via.

Prendersi cura di tutti questi passaggi è un compito arduo e richiede molta esperienza e, naturalmente, tempo, minando l'efficienza dell'intero progetto che si deve gestire.

AWS, che sta progredendo molto rapidamente negli ultimi due anni nello **sviluppo di strumenti e servizi per aiutare nelle attività di machine learning**, questa volta ci porta un altro importante servizio in soccorso: **AWS Elastic Views**.

AWS Elastic Views consente a un utente di richiedere dati da diverse fonti in modo completamente indipendente dalla loro natura, di eseguire query per i dati in un linguaggio compatibile con SQL e di inviare tutti i dati interrogati a una destinazione, tipicamente S3 o un'altra destinazione dati, ed infine a produrre un data lake con cui lavorare.

Alcuni dei principali vantaggi sono:

- Essere in grado di interrogare un database qualsiasi o un datastream di diversa natura in linguaggio PartiQL, diventando di fatto un sistema di aggregazione, senza la necessità di scrivere workload ETL complessi e personalizzati.
- Utilizzo di comandi potenti come JOIN per aggiungere funzionalità di aggregazione a origini dati che di solito non hanno tale capacità.

Lo scopo di questo articolo è guidare il lettore nell'esplorazione di alcuni dei fattori chiave che rendono questo servizio qualcosa di cui essere definitivamente consapevoli nei propri progetti di Machine Learning.

Esploreremo in profondità ciò che è in grado di fare AWS Elastic Views, considerando però che è ancora in fase beta, quindi si dovrà richiedere l'accesso ad AWS per l'anteprima.

Cominciamo!

Come funziona

Cominciamo il nostro viaggio capendo che **cos'è AWS Glue Elastic Views, e come funziona**. Per prima cosa, diamo uno sguardo allo schema fornito da AWS:



Courtesy of AWS - AWS Glue Elastic Views inputs and outputs

Come mostrato nell'immagine, il punto focale di questo servizio è rappresentato dalla **Materialized View**, che è un sistema per astrarre il set di dati da qualsiasi tipo di sorgente: ad esempio Amazon Aurora, RDS o DynamoDB. Ciò consente di mantenere le informazioni sincronizzate senza l'uso effettivo di un Glue Crawler, come ci saremmo aspettati dai nostri altri articoli sui workload ETL ([qui](#) e [qui](#) alcuni esempi).

Ma diamo un'occhiata in dettaglio alle caratteristiche principali e come possono essere utili.

Usare SQL come mezzo per creare una view

AWS Glue Elastic Views consente a uno sviluppatore di creare viste materializzate su diverse origini di dati, utilizzando query SQL per aggregare i dati. AWS Glue Elastic Views attualmente supporta Amazon DynamoDB, Redshift, S3 e Elasticsearch Service. Inoltre, AWS ha in programma di aggiungere ancora più origini di dati in futuro.

Copiare autonomamente da una fonte dati ad un datalake target

AWS Glue Elastic Views gestisce per noi il lavoro più pesante di copia e aggregazione dei dati da tutte le origini dati fino agli archivi di destinazione, senza dover scrivere codice personalizzato o utilizzare strumenti ETL e linguaggi di programmazione complessi o sconosciuti, con un effetto benefico sia sul tempo che sull'efficienza del progetto. AWS Glue Elastic Views riduce il tempo necessario per combinare e replicare i dati negli archivi dati da mesi a minuti, secondo AWS.

Mantenere i dati nella destinazione sempre aggiornati, automaticamente

Mantenere i dati sincronizzati di solito richiede la creazione e la manutenzione di crawler, AWS Glue Elastic Views, invece, monitora continuamente le modifiche ai dati negli archivi dati iniziali e, quando si verifica una modifica, Elastic Views aggiorna automaticamente le destinazioni. Ciò garantisce che le applicazioni che accedono ai dati utilizzando Elastic Views dispongano sempre dei dati più aggiornati.

Avvisare quando avviene un cambiamento nei dati di un data store sorgente

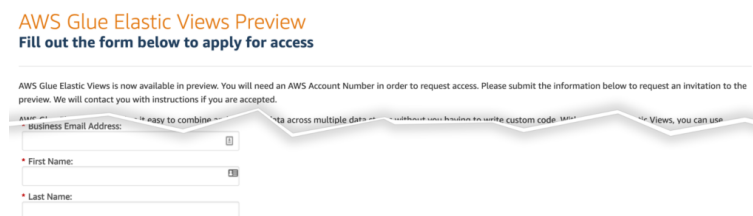
AWS Glue Elastic Views avvisa in modo proattivo gli sviluppatori quando viene apportata una modifica al modello di dati in uno degli archivi dati di origine, in modo che possano aggiornare le loro visualizzazioni per adattarsi a questa modifica velocemente.

Serverless

AWS Glue Elastic Views è completamente serverless e aumenta o diminuisce automaticamente la propria capacità per adattarsi automaticamente ai carichi di lavoro. Non c'è hardware o software da gestire e, come sempre, un utente paga solo per le risorse che utilizza.

Registrarsi per la preview gratuita

Essendo un servizio ancora in beta è necessario registrarsi per la preview gratuita: per farlo, è sufficiente andare a questo [indirizzo](#) e registrarsi, compilando l'apposito form.



The image shows a registration form for the AWS Glue Elastic Views Preview. At the top, it says "AWS Glue Elastic Views Preview" in orange, followed by "Fill out the form below to apply for access". Below this is a paragraph of text: "AWS Glue Elastic Views is now available in preview. You will need an AWS Account Number in order to request access. Please submit the information below to request an invitation to the preview. We will contact you with instructions if you are accepted." The form itself has a white background with a light blue border. It contains three input fields: "Business Email Address:" with a small icon to its right, "First Name:" with a small icon to its right, and "Last Name:" with a small icon to its right. Each field has a red asterisk to its left, indicating it is a required field.

Spaccato del modulo di registrazione alla preview

Verranno richiesti i dettagli personali e aziendali, nonché un'introduzione di base al problema che si desidera risolvere utilizzando AWS Glue Elastic Views. È bene Assicurarsi

di fornire motivazioni ragionevoli, poiché casi d'uso interessanti aumentano le possibilità di essere selezionati per l'anteprima.

Di solito, AWS risponde entro una settimana e se idoneo per l'anteprima, il seguente messaggio verrà inviato alla propria email.

Dear Customer,
We are excited to welcome you to the AWS Glue Elastic Views preview. You can now access AWS Glue Elastic Views using your account below.
Account: [REDACTED]
Before getting started, everyone on your team who is participating in the preview should review the preview usage notes below.

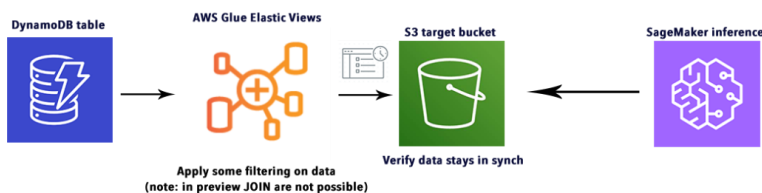
La mail di preview

Dopo essersi registrati, per accedere a Glue Elastic View è necessario cliccare sul link fornito nella mail.

Ora siamo pronti per iniziare il nostro primo workload ETL con Elastic View, invece di utilizzare script Spark standard o Glue Crawler.

Entriamo nel vivo del nostro test

Il modo migliore per capire le possibilità di questo servizio è metterci alla prova con esso. Quindi abbiamo deciso di creare un semplice caso d'uso per presentare come potrebbe essere utilizzato per semplificare i propri workload ETL.



Schema del nostro esempio semplificato

Panoramica

L'idea di base è popolare una tabella DynamoDB con alcuni dati di test ottenuti da fonti di dati gratuite. Vogliamo estrarre e manipolare i dati da questa tabella per dimostrare

come Glue Elastic Views può aggiungere efficacemente funzionalità ETL a DynamoDB, che è storicamente un po' debole su questo aspetto.

Quindi vogliamo inviare questi dati a S3 e verificare che sia possibile aggiornarli in tempo reale per riflettere i cambiamenti, dimostrando come Glue Elastic Views può accelerare efficacemente il processo di indagine di una pipeline di Machine Learning.

Nota: al momento della scrittura di questo articolo, sfortunatamente, l'operazione JOIN non è supportata per PartiQL in AWS Glue Elastic Views, quindi abbiamo optato per testare alcune operazioni matematiche e logiche, oltre a convalidare tutti i passaggi per sincronizzare la tabella DynamoDB con il bucket S3.

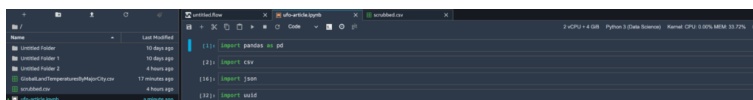
Dataset

Per il nostro esempio, abbiamo deciso di utilizzare un semplice **set di dati sugli avvistamenti UFO**. Vogliamo mettere questo file CSV in una tabella DynamoDB, come detto prima, e applicare alcune operazioni di filtraggio sui campi di latitudine e longitudine, o anche città. Il risultato verrà utilizzato per rispondere alla seguente domanda di esempio: "In che modo gli avvistamenti UFO variano a seconda delle diverse zone?".

Questo è, ovviamente, solo un semplice esempio, non ha implicazioni pratiche a parte dimostrare alcune funzionalità di AWS Glue Elastic Views.

Importare i dati in DynamoDB

Vogliamo creare la tabella per DynamoDB, e per fare questo, abbiamo definito un semplice script, in SageMaker Studio, che si sobbarchi l'operazione per noi.



SageMaker Studio: interfaccia

Fondamentalmente andiamo a leggere i dati dal file CSV usando Pandas, convertiamo le righe in JSON, ma prima di farlo, aggiungiamo anche una colonna "hash" chiamata **id**, perché DynamoDB ha bisogno di una chiave primaria per ogni item.

```
for record in json_list:
    if record['longitudine'] and record['latitudine']:
        record['id'] = sha256(str(record).encode()).hexdigest()
```

Il set di dati sugli UFO presentava anche alcuni problemi che dovevano essere risolti: l'header "longitudine" aveva degli spazi da rimuovere e le colonne di latitudine e longitudine dovevano essere convertite in formato stringa rimuovendo le voci NaN.

Infine, abbiamo utilizzato boto3 per creare una tabella corrispondente al CSV.

```
dynamodb = boto3.resource('dynamodb') table =
dynamodb.Table('article_ufo_sightings') with table.batch_writer() as batch: ...
batch.put_item(Item=record)
```

Il codice completo può essere esplorato [qui](#).

Nota: avremmo potuto utilizzare AWS Glue anche per questa attività, inserendo il file CSV in un bucket S3 di origine, utilizzando poi Glue Crawler per importare i dati, ma poiché abbiamo già trattato i lavori ETL con questo servizio in altri [articoli](#), abbiamo optato per una soluzione più semplice non essendo questo il fulcro dell'esempio.

Creare la tabella in DynamoDB

Siamo semplicemente andati alla console DynamoDB, abbiamo cliccato su "create table" e abbiamo utilizzato le semplici impostazioni segnate in immagine. Una nota però: applichiamo la modalità di capacità **on-demand** per velocizzare la generazione della tabella.

Create DynamoDB table Tutorial

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name*

Primary key* Partition key

Add sort key

Table settings

Read/write capacity mode

Select on-demand if you want to pay only for the read and writes you perform, with no capacity planning required. Select provisioned to save on throughput costs if you can reliably estimate your application's throughput requirements. See the [DynamoDB pricing page](#) and [DynamoDB Developer Guide](#) to learn more.

Read/write capacity mode can be changed later.

Provisioned (free-tier eligible)

On-demand

Settaggi per la tabella - id come chiave primaria e on-demand per la capacity mode

Aggiungiamo **id** come chiave primaria per la tabella UFO sightings

Creare una Materialized View dalla tabella

Prima di generare la vista, dovevamo aggiungere la tabella DynamoDB come sorgente in AWS Glue Elastic Views; per farlo, siamo andati alla console principale, abbiamo selezionato “Tables” a sinistra e cliccato su “Create Table”. Quindi abbiamo selezionato la nuova tabella DynamoDB, generata nei passaggi precedenti.

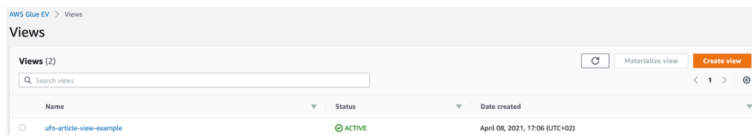


Tabella di partenza per la Materialized View

Il passaggio successivo è stato applicare alcuni filtri per creare il nostro set di dati di destinazione finale, dipendente dalla vista.

Andando sulla scheda “Views” sul lato sinistro della console ne abbiamo creata una nuova. Qui ci è stata presentata la possibilità di scrivere codice PartiQL personalizzato: esattamente quello che volevamo!

Abbiamo aggiunto il seguente codice nell’editor per abilitare la nostra vista materializzata:

```
SELECT id, Latitude, Longitude FROM article_ufo_sightings.article_ufo_sightings
```

Abbiamo anche dovuto scrivere tutti gli attributi che volevamo esportare nei file di parquet di destinazione (sembra che AWS Glue Elastic Views generi batch di file di parquet nella directory di output).

Come il lettore può osservare dal codice sopra, abbiamo evitato di richiedere apposta informazioni utili: volevamo mostrare che è possibile modificare la vista in tempo reale dopo che la materialized view è stata creata.

Inviare i dati ad Amazon S3

Al momento della scrittura di questo articolo, S3 è una delle tre opzioni disponibili come target, insieme a ElasticSearch e Redshift. Nel nostro caso S3 è la destinazione ideale, in quanto vogliamo che il set di dati finale venga consumato da SageMaker.

Siamo andati dentro la View e abbiamo cliccato su “Materialized View”, quindi abbiamo selezionato “Glue EV” come supporto per sbloccare “S3” come target: lì abbiamo aggiunto il bucket “article-ufo-materialized-views” e selezionato default come crittografia; abbiamo aggiunto un ruolo IAM adatto per l’esecuzione.

Il ruolo può essere creato utilizzando l’**editor per ruoli e policy di AWS**, tenendo però a mente che, una volta creato, si dovrà modificare la **trust relationship** con il seguente codice per abilitare il ruolo IAM, altrimenti non saremo in grado di vedere il ruolo nel selettore:

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": [
        "materializedviews.amazonaws.com"
      ]
    },
    "Action": "sts:AssumeRole"
  }
]
```

Qui abbiamo invece uno screenshot con i parametri utilizzati:

Target location

Location for materialized view
 Enter a location in Amazon S3 where your materialized view will be stored and updated. Amazon S3 is object storage built to store and retrieve data. [Learn more](#)

S3Uri

Format: s3://bucket/prefix. For example, if you specify s3://myviews/materializedview1, then AWS Glue EV exports the Parquet files to s3://myviews/materializedview1. The Amazon S3 Bucket must be in eu-west-1 Region.

S3 encryption

Bucket default encryption

AES-256
 Use server-side encryption with Amazon S3 managed keys (SSE-S3)

AWS-KMS
 Use server-side encryption with AWS KMS managed keys (SSE-KMS)

Permission
 Specify the IAM role for AWS Glue EV to access your S3 bucket. For KMS encryption, the role must include access to your CMK. [Learn more](#)

Target name

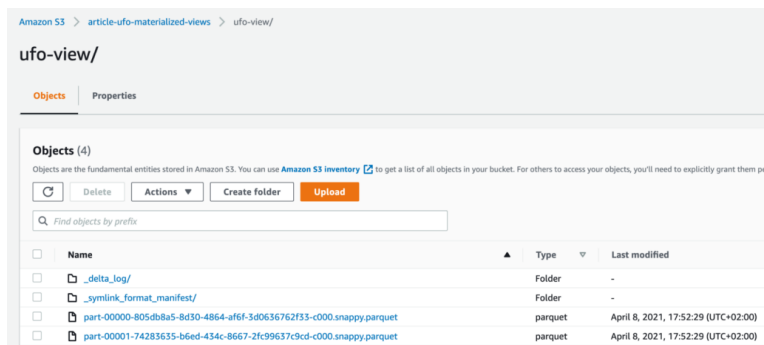
Folder name
 Specify a name for a target. The name must be unique for all targets per service that are in your AWS account in the current Region. For example, if you specify myfolder, then AWS Glue EV exports the Parquet files to s3://bucket/prefix/myfolder.

All letters must be lowercase. Folder names cannot begin with _ or -. Folder names cannot contain spaces, commas, :, ;, *, +, /, \, |, ?, #, %, >, or <.

Materialized View - settaggi di destinazione su S3

Una volta creata, la vista deve essere attivata per sincronizzarsi con il bucket S3; per farlo, siamo andati sia nella tabella che nella vista che avevamo creato, e abbiamo cliccato su “Attiva” nel loro pannello di dettaglio.

Dopo un paio di minuti dall’attivazione, il bucket S3 è stato riempito con i dati risultanti!



I dati di destinazione

Fatto interessante: appena prima di attivare la visualizzazione, il servizio si è lamentato del fatto che alcuni campi non fossero compatibili con il target di output e ci ha dato la possibilità di modificare la visualizzazione al volo con un editor in linea, questo è quello che abbiamo fatto per castare “Latitudine” e “Longitudine” a numero intero:

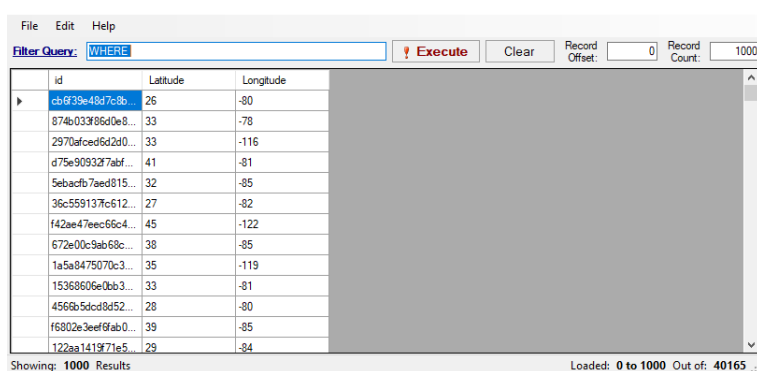
Definition

```
1 SELECT id, cast(Latitude as integer), cast(Longitude as integer) FROM "ufo-article-view-example"
```

I dati sono ora collegati direttamente con il nostro bucket S3, quindi qualsiasi modifica apportata alla tabella si riflette direttamente dopo alcuni secondi. Fondamentalmente è come avere un Glue Crawler che funziona su richiesta, che si accende e si spegne quando necessario, e senza intervento umano.

Modifichiamo i dati su S3

Volevamo dimostrare che è possibile modificare i dati ottenuti dalla tabella DynamoDB in qualsiasi momento, per questo abbiamo iniziato salvando "file parquet incompleti" come il lettore può osservare qui:



id	Latitude	Longitude
cb9f39e45d72c3b	26	-90
874b033f86d0e8...	33	-78
2970afced6d2d0...	33	-116
d75e909327abf...	41	-81
5ebacfb7aed815...	32	-85
36c559137fc612...	27	-82
f42ae47eec66c4...	45	-122
672e00c9ab69c...	38	-85
1a5a8475070c3...	35	-119
15368606e0bb3...	33	-81
4568b5dcd8d52...	28	-90
f6802e3eef6ab0...	39	-85
122aa141971e5...	29	-84

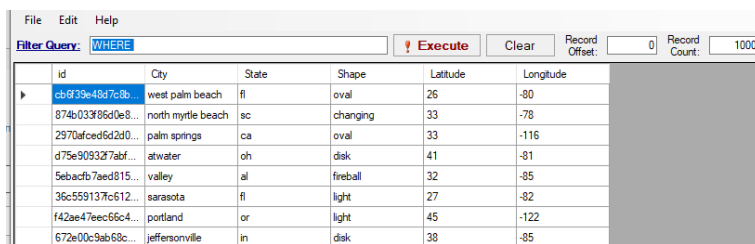
Dataset incompleto con id, Latitudine e Longitudine

Per modificare il nostro set di dati finale dovevamo prima "disattivare" la vista materializzata. Successivamente, è stato possibile definire una nuova vista Materializzata sullo stesso target aggiungendo più colonne. Si noti inoltre che, se si dispone di più visualizzazioni dipendenti l'una dall'altra, è necessario disattivarle ed eliminarle nell'ordine corretto. Forse questa operazione diventerà meno macchinosa al momento del rilascio ufficiale.

Abbiamo modificato la definizione originale della vista materializzata aggiungendo più colonne:

```
SELECT id,cast(City as string),cast(State as string),cast(Shape as string),cast(Latitude as integer),cast(Longitude as integer) FROM article_ufo_sightings.article_ufo_sightings;
```

Nonostante il processo non proprio lineare, queste operazioni hanno richiesto meno di 5 minuti e i nuovi dati sono stati presto resi disponibili nel bucket S3 corretto:



id	City	State	Shape	Latitude	Longitude
cb6f939e48d7c9b...	west palm beach	fl	oval	26	-80
874b033f85d0e8...	north myrtle beach	sc	changing	33	-78
2970afced6d2d0...	palm springs	ca	oval	33	-116
d75e9093d77abf...	atwater	oh	disk	41	-81
5ebacfb7aed815...	valley	al	fireball	32	-85
35c559137fc612...	sarasota	fl	light	27	-82
f42ae47eecd65c4...	portland	or	light	45	-122
672e00c9ab68c...	jeffersonville	in	disk	38	-85

Dataset con tutte le colonne

Ovviamente se le modifiche si trovano nella tabella originale e non nella vista, gli aggiornamenti sono completamente “seamless” come ci si aspetterebbe.

Un altro promemoria: essendo l’interfaccia ancora in fase di anteprima, consigliamo di evitare di lanciare molte operazioni in poco tempo, poiché abbiamo riscontrato diversi bug legati a “race condition” non ancora gestite correttamente.

Ora possiamo farci del Machine Learning!

Per verificare che il datasource target sia sfruttabile per lavori di machine learning, abbiamo anche preparato un semplice Jupiter Notebook di test per SageMaker, applicando alcune semplici analisi di correlazione sul data lake creato. Tutto questo si può vedere in dettaglio nel [notebook](#). L’idea era di verificare se esiste una sorta di correlazione tra luoghi, città e avvistamenti UFO e, sulla base dei dati di esempio, provare a fare alcune semplici inferenze. Ulteriori informazioni su come utilizzare SageMaker per fare inferenza sono state trattate in [questo articolo](#).

Referenze

- <https://pages.awscloud.com/AWS-Glue-Elastic-Views-Preview.html>
- <https://aws.amazon.com/glue/features/elastic-views/>
- <https://aws.amazon.com/glue/faqs/>
- <https://cloudacademy.com/course/aws-reinvent-2020-aws-glue-elastic-views-1209/aws-glue-elastic-views/>

Per concludere

Siamo giunti alla fine di questo viaggio nelle meraviglie di AWS Elastic Views, quindi è il momento di riassumere ciò che abbiamo imparato finora.

Questo servizio AWS si rivela prezioso quando si tratta di lavorare con molte origini dati, soprattutto se di diversa natura, in quanto recupera e interroga tutti i dati con un linguaggio compatibile SQL (PartiQL), evitando la creazione di molti lavori ETL Glue complessi.

È perfetto in tutte quelle situazioni in cui è necessario combinare dati legacy e nuovi, poiché di solito risiedono, come da best practice, su diverse origini dati: quelle più economiche per gli accessi poco frequenti e quelle con bassa latenza per i nuovi dati.

Se vogliamo utilizzare S3 come target, diventa una soluzione **adatta per job di SageMaker o anche per attività che sfruttano i servizi di AWS Managed Machine Learning**.

Se Elasticsearch è il target designato, Elastic Views diventa perfetto per i workload di Business Intelligence.

AWS Elastic Views supporta gli aggiornamenti in tempo reale sui dati, con la possibilità di aggiornare anche un singolo valore per riflettere le modifiche; tutto questo utilizzando un linguaggio SQL semplice e universalmente noto, che offre funzionalità SQL per i database che non le supportano.

Potendo aggiornare un singolo campo, evita di eseguire nuovamente la scansione di tutti i dati in un'origine dati per aggiornare la destinazione scelta.

Infine, vorremmo dare un consiglio: poiché la preview attuale è ancora in una fase molto preliminare, la maggior parte delle funzionalità descritte non sono ancora disponibili completamente per una prova, quindi anche se il prodotto è già utile in diversi casi, è bene sperimentare prima di utilizzarlo per i lavori di produzione o attendere il rilascio pubblico.

Ed eccoci qui! Ci auguriamo che la lettura ti sia piaciuta e che abbia fornito utili spunti. Come sempre, sentiti libero di commentare nella sezione sottostante e [contattaci](#) per qualsiasi dubbio, domanda o idea!

Ci vediamo su **#Proud2beCloud** tra un paio di settimane per un'altra storia!



Alessandro Gaggia

Head of software development di beSharp, Full-Stack developer, mi occupo di garantire lo stato dell'arte di tutta la nostra codebase. Scrivo codice in quasi ogni linguaggio, ma prediligo Typescript. Respiro Informatica, Game design, Cinema, Fumetti e buona cucina. Disegno per passione!



Matteo Moroni

DevOps e Solution Architect di beSharp, mi occupo di sviluppare soluzioni SaaS, Data Analysis, HPC e di progettare architetture non convenzionali a complessità divergente. Appassionato di informatica e fisica, da sempre lavoro nella prima e ho un PhD nella seconda. Parlare di tutto ciò che è tecnico e nerd mi rende felice!

Copyright © 2011-2021 by beSharp srl - P.IVA IT02415160189