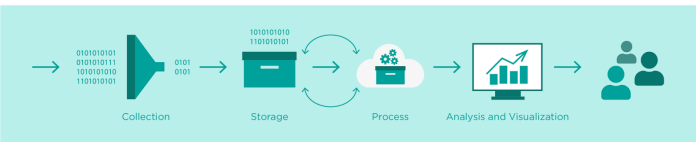
ACQUISIRE E ANALIZZARE DATI IOT SU AWS: UN SEMPLICE ESEMPIO CON AURORA E ATHENA

Amazon Athena | Amazon Kinesis Data Firehose | Amazon Rekognition | Amazon S3 | Aurora Serverless | AWS Glue Databrew | Data and Analytics | Data Ingestion | Data Visualization | Dataset | Internet of Things (IoT)

beSharp | 11 Dicembre 2020

Con l'avvento dell'Internet of Things, il numero di device connessi sta aumentando in modo esponenziale, così come la quantità di dati da essi generati. Per questo l'acquisizione e l'analisi dei dati è diventato uno degli argomenti più scottanti dell'attuale panorama IT. AWS offre un'ampia gamma di servizi che ci consentono di importare, raccogliere, archiviare, analizzare e visualizzare enormi quantità di dati in modo rapido ed efficiente.



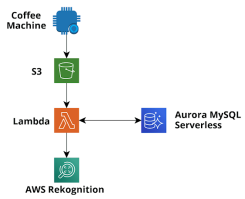
In questo breve articolo andremo a presentare un'applicazione molto semplice, ma reale che abbiamo sviluppato per rappresentare la pipeline di acquisizione e analisi dei dati nel contesto di eventi e conferenze AWS e IoT.

Abbiamo modificato una macchina da caffè moderna tramite l'utilizzo di componenti elettronici personalizzati quali un Raspberry Pi Zero e una microcamera, affinché, al premere del pulsante di erogazione, scattasse una foto alla persona di fronte e la caricasse su S3. A seguito dell'upload della foto, Amazon Rekognition, triggerato da AWS Lambda, si occupa di analizzarla, individuando parametri quali: – esiste la persona nella foto? – la persona ha gli occhiali? – ha la barba? – ha la baffi? – sta sorridendo?

A termine dell'analisi dell'immagine, se questa contiene il volto di una persona, un record viene scritto dalla funzione Lambda in un db serverless Aurora MySQL insieme all'output del modello di Machine Learning di Rekognition.

Infine abbiamo sviluppato una semplicissima applicazione web collegata al database per la visualizzazione delle statistiche e una query con AWS Athena che pulisce i dati e li sposta in un bucket S3 come file parquet.

Di seguito è mostrato uno schema dell'infrastruttura proposta.



Passiamo ora a descrivere i passaggi di un comune processo di acquisizione e trasformazione dei dati e come li stiamo organizzando nella nostra applicazione.

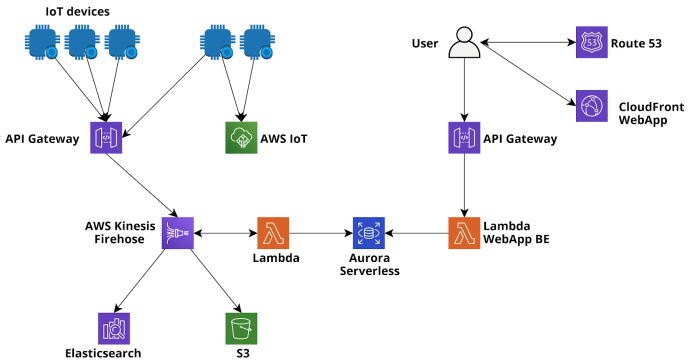
La fase di importazione / archiviazione

In AWS un flusso di inserimento dati molto comune consiste nell'utilizzare AWS IoT Core (Secure MQTT) o Api Gateway (API REST o Websocket) come entry point di dati, connettendolo direttamente a Kinesis Firehose (utilizzando regole IoT o Api gateway Service integrations) e infine sfruttare le potenti funzionalità di Firehose per il buffering dei dati, la trasformazione del buffer (funzioni AWS Lambda), la crittografia del flusso (AWS KMS), la compressione dei dati (GZIP) e la consegna dei dati a batch di messaggi compressi e crittografati automaticamente sia ad uno storage di tipo duraturo (S3) che a un data warehouse (AWS Redshift) per query analitiche complesse sull'enorme quantità di dati raccolti.

Avere sempre tutti i dati acquisiti salvati in AWS S3 è un passaggio essenziale, non solo come salvavita in caso di problemi con altri archivi più caldi ma anche per creare un data lake condiviso che può essere successivamente analizzato con Athena, EMR, Glue Jobs, Glue Databrew e anche strumenti esterni.

Inoltre è possibile utilizzare Firehose per fornire direttamente i dati ad AWS ElasticSearch per analisi in tempo reale e, se necessario, è anche molto facile fornire i batch di dati importati ad un database relazionale (ad esempio Aurora Serverless Postgres / MySQL) utilizzando AWS Data Migration Service o funzioni Lambda basate su eventi. La migrazione dei dati inseriti (o di un'aggregazione di essi) ad un database relazionale esistente è spesso molto utile se si ha bisogno di arricchire un'applicazione legacy esistente che già utilizza il database.

Nel caso si decida di utilizzare le funzioni Lambda per spostare i dati importati su Aurora, metodo solitamente più veloce e scalabile, è possibile sfruttare direttamente le funzioni Lambda di trasformazione di Firehose o una funzione differente attivata ogni volta che Firehose scrive un oggetto su S3.



Una peculiarità di Firehose consiste nella possibilità di aggiungerlo anche in un secondo momento! Nella nostra semplice applicazione non lo utilizziamo immediatamente, pertanto le immagini e le analisi vengono salvate direttamente in S3 e AWS Aurora Serverless MySQL da Lambda Functions, nel caso in cui il flusso generato dall'applicazione dovesse crescere potremmo successivamente integrarlo senza difficoltà alcuna!

Fase di analisi

Una volta archiviati i dati, è il momento di analizzarli. In questo caso le metodologie possono differire notevolmente. Gli esempi più comuni vanno dalle semplici query eseguite in database relazionali, ai job analitici complessi eseguiti nei data warehouse di Redshift, oppure all'elaborazione in tempo reale utilizzando EMR o ElasticSearch.

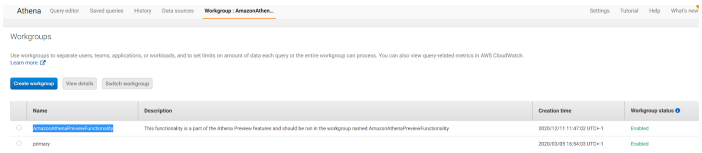
Nel nostro caso possiamo semplicemente eseguire query utilizzando il backend della nostra applicazione web e visualizzare i risultati nel browser.

Tuttavia, in futuro potremmo essere interessati ad eseguire query molto più avanzate sui nostri dati e magari fare qualche check sulla qualità del dato o training per modelli di Machine Learning. Per rendere possibili queste espansioni, occorre quindi spostare i dati da Aurora a S3 per analizzarli con job Glue e Databrew e, se necessario, caricarli facilmente con Apache Spark da Glue o AWS EMR. Per fare ciò possiamo seguire diversi strade: ad esempio potremmo usare il servizio AWS DataMigration per spostare i dati su S3 come file Parquet oppure potremmo creare un Glue Job, caricare i dati usando Glue Connection da RDS con Spark e poi scriverli in S3.

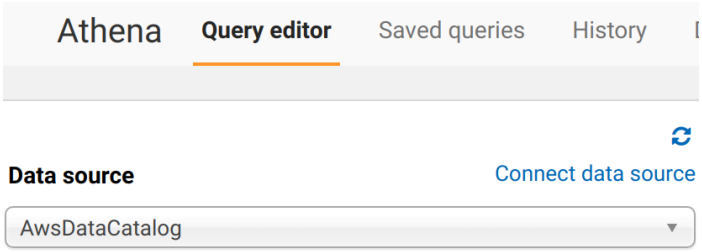
Dopo questo passaggio, sarebbe necessario eseguire un crawler Glue per creare un DataCatalog che verrà utilizzato da Athena e Glue per query e jobs.

Qui tuttavia mostreremo un percorso diverso e talvolta molto più flessibile per esportare i dati in modo pulito e catalogarli da un database relazionale: **Athena custom data source.**

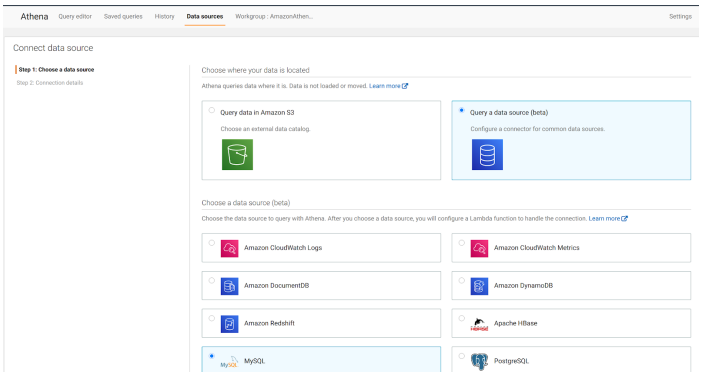
Per impostazione predefinita, Athena viene fornito con l'integrazione S3 – Glue Data Catalog, ma di recente AWS ha dato la possibilità di aggiungere un'origine dati personalizzata, ad esempio database connessi tramite JDBC, AWS CloudWatch o l'esecuzione di query su S3 ma utilizzando un metastore Apache Hive personalizzato. Nel nostro caso siamo interessati a connetterci a MySQL Aurora Serverless. Per fare ciò dobbiamo andare su Athena Home, configurare un workgroup denominato AmazonAthenaPreviewFunctionality e quindi aggiungere una path S3 di output delle query di Athena:



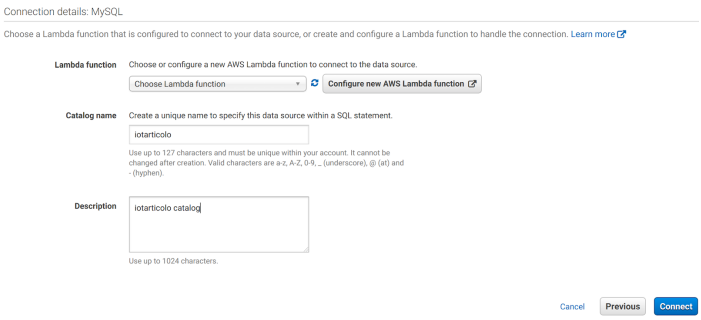
Dopodiché possiamo tornare alla home di Athena e selezionare Connect Data Source:



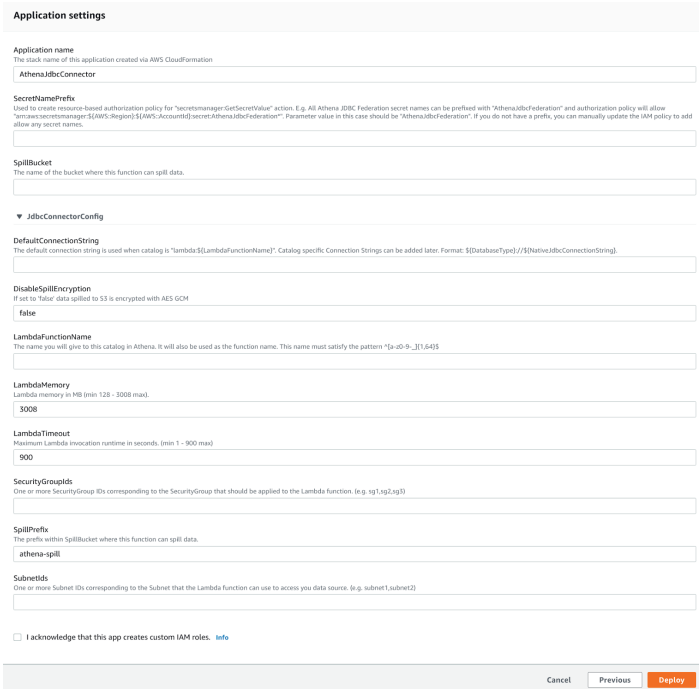
Ci viene presentata una pagina web dove dobbiamo selezionare il tipo di sorgente dati: optiamo per Query a data source (beta) e MySQL:



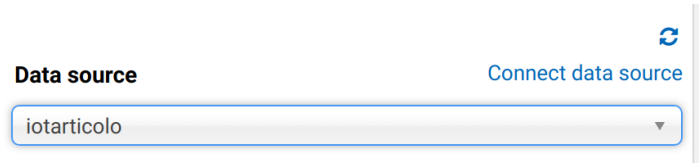
A questo punto ci viene chiesto di inserire il nome e la descrizione del nuovo catalogo e di selezionare o creare una funzione Lambda per gestire la connessione. Impostiamo il nome desiderato e facciamo clic su Configura nuova funzione AWS Lambda.



Viene presentata questa pagina in cui bisogna inserire l'uri di connessione JDBC per Aurora e selezionare la subnet e il security group per la funzione Lambda che Athena utilizzerà per stabilire la connessione JDBC. Vanno scelti accuratamente, altrimenti la Lambda non raggiungerà l'istanza Aurora!



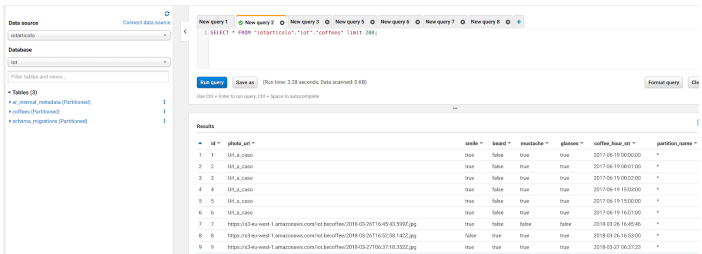
Il prefisso del Secret viene utilizzato per memorizzare le credenziali del database in AWS Secret Manager, questo è essenziale per l'ambiente di produzione e se lo si lascia vuoto, l'integrazione non verrà creata. Dopo aver selezionato deploy e selezionato la Lambda appena creata nella dashboard di Athena, verrà creato un nuovo catalogo diverso dallo standard AwsGlueCatalog:



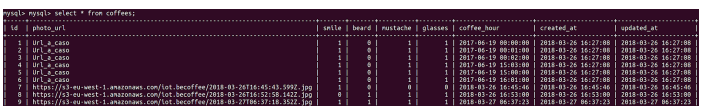
Tuttavia all'inizio Databeses e tabelle non appariranno. Controllando i log della Lambda su CloudWatch si troverà un errore del tipo:

```
Catalog is not supported in multiplexer. After registering the catalog in Athena, mus
t set 'iotarticolo_connection_string' environment variable in Lambda. See JDBC connect
or README for further details.: java.lang.RuntimeException
```

Impostiamo quindi la variabile di environment richiesta per la funzione Lambda utilizzando la stessa stringa di connessione JDBC usata come stringa DefaultConnection nel passaggio precedente. Dopodiché la connessione funzionerà e si potrà interrogare il DB direttamente da Athena! Ottimo!



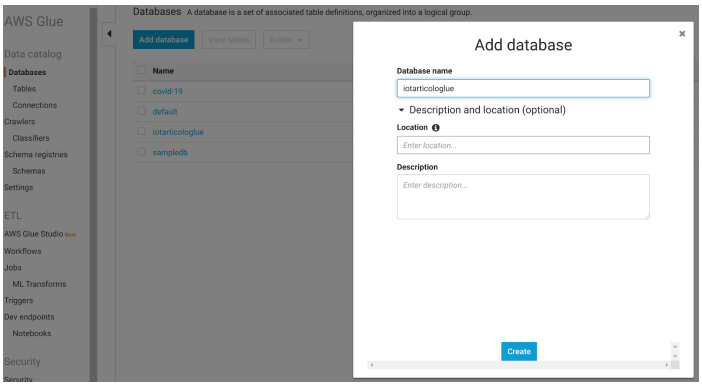
Ad uno sguardo più attento, però, notiamo immediatamente che qualcosa è in conflitto con i dati: ecco una schermata di ciò che possiamo leggere direttamente da MySQL:



Come si può vedere Athena è abbastanza intelligente da convertire i dati tinyint(1) in bool ma non riesce a leggere le colonne datetime da mysql. Ciò è dovuto ad un problema molto noto con il connettore jdbc e la soluzione più semplice è creare un nuovo campo dove datetime è una stringa in formato datetime Java:

```
UPDATE coffees SET coffees.coffee_hour_str=DATE_FORMAT(coffee_hour, '%Y-%m-%d %H:%i:%
s');
ALTER TABLE coffees ADD COLUMN coffee_hour_str VARCHAR(255) AFTER coffee_hour;
```

A questo punto Athena potrà leggere il nuovo campo. Ed ora siamo pronti per un bellissimo trucco: andiamo nella dashboard di Glue e creiamo un nuovo Database: un database è solo un contenitore logico per metadati, si può scegliere il nome che si preferisce.



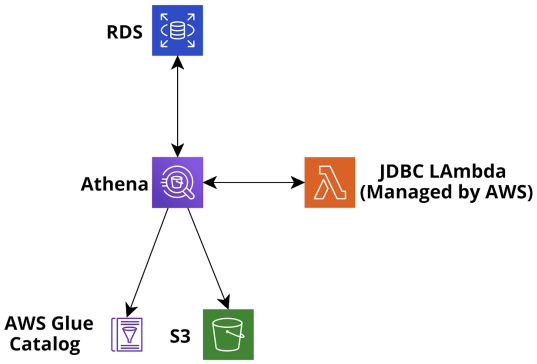
A questo punto possiamo tornare ad Athena ed eseguire una query come questa:

```
CREATE table iotarticologlue.coffees

WITH (

  format='PARQUET', external_location='s3://besharp-athena/coffees_parquet', parquet_c
ompression='GZIP'

) AS SELECT photo_url,smile,beard,mustache,glasses,coffee_hour_str FROM
"iotarticolo"."iot"."coffees"

WHERE photo_url LIKE 'https://%';
```

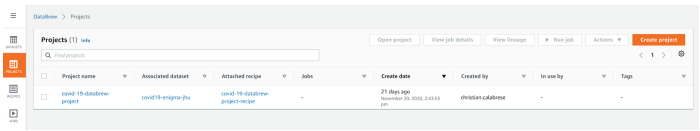
Ciò creerà una nuova tabella nel database che abbiamo appena aggiunto al nostro catalogo di dati Glue e salverà tutti i dati in S3 come file GZIP Parquet. Inoltre è anche possibile, se lo si desidera, cambiare la compressione (es. Snappy o BZIP).

Oltre ad esportare i dati come Parquet, la query andrà ad eliminare quelli con la formattazione errata per l'url di S3.

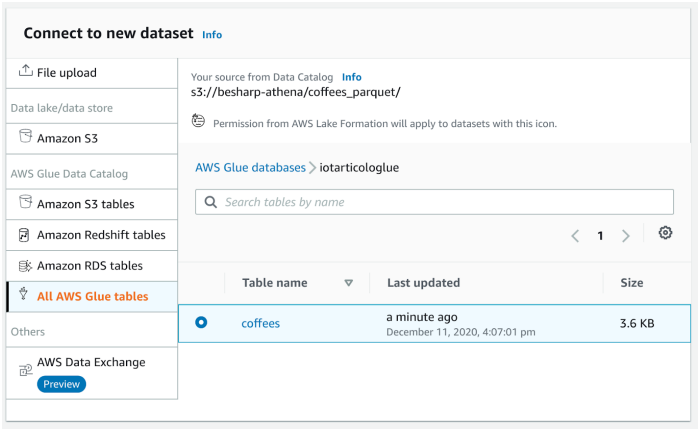
Abbiamo quindi un modo super veloce per esportare il nostro db in S3 come parquet e creare automaticamente il catalogo Glue.



Diventa quindi semplice visualizzare questo nuovo catalogo in AWS Glue Databrew: andiamo alla dashboard di Databrew e creiamo un nuovo progetto

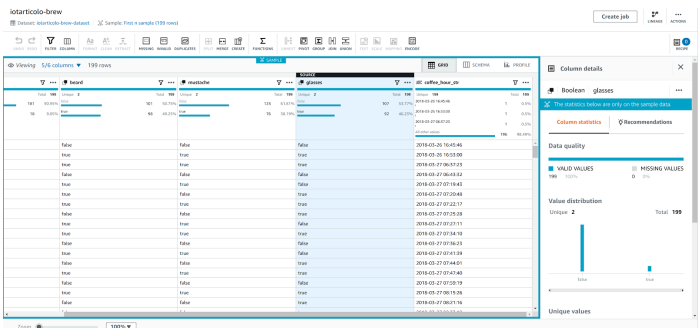


e un nuovo set di dati nella sezione aggiungi dataset.



In caso di errore sarà necessario rinominare il file s3 come .parquet e scansionare nuovamente la tabella con i crawler Glue.

Et voilà una bellissima visualizzazione dei dati del nostro dataset completa di statistiche delle colonne!



Conclusioni

In questo articolo abbiamo descritto un'applicazione IoT molto semplice che utilizza Amazon Rekognition e Amazon Aurora. Abbiamo spiegato come può essere migliorata sfruttando Firehose e infine abbiamo utilizzato Athena per trasformare e pulire i dati raccolti.

Abbiamo visto anche come salvarli molto facilmente come parquet e come possono essere analizzati con Glue Databrew, Athena e altri strumenti AWS come EMR.

Avete mai provato configurazioni simili per i vostri processi di Data Analysis?

Fateci sapere! Saremo felici di offrirvi un caffè...connesso 😀

Per oggi è tutto.

Continuate a seguirci: ci vediamo tra 14 giorni qui su **#Proud2beCloud!**

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it