# COME ORCHESTRARE UNA PIPELINE DI DATA ANALYTICS E BUSINESS INTELLIGENCE VIA STEP FUNCTION

Amazon Athena | Amazon Kinesis Data Firehose | Amazon QuickSight | AWS Glue | Data and Analytics

beSharp | 18 Febbraio 2021

Le pipeline di ETL su AWS di solito hanno un comportamento lineare: si inizia da un servizio e si termina con un altro. Questa volta, tuttavia, vorremmo presentare una configurazione più flessibile, in cui alcuni job ETL potrebbero essere saltati a seconda dei dati. Inoltre, alcuni dei dati trasformati nel nostro datalake verranno interrogati da AWS Athena per generare dashboard di BI in QuickSight, mentre altre partizioni di dati verranno utilizzate per addestrare un rilevamento di anomalie ad-hoc tramite Sagemaker.

Un potente strumento per orchestrare questo tipo di pipeline ETL è il servizio AWS StepFunctions.

In questo articolo, vogliamo mostrarti alcuni dei passaggi coinvolti nella creazione della pipeline citata e quali servizi AWS per l'analisi dei dati si possano utilizzare in scenari quasi in tempo reale per gestire un volume elevato di dati in modo scalabile.

In particolare, esamineremo i connettori e i crawler di AWS Glue, AWS Athena, QuickSight, Kinesis Data Firehose e infine una breve spiegazione su come utilizzare SageMaker per creare previsioni a partire dai dati raccolti. Per saperne di più su Sagemaker puoi anche dare un'occhiata ai nostri altri articoli.

Iniziamo!

## Il nostro setup

In questo esempio, configureremo diversi sensori per inviare dati di temperatura e diagnostici alla nostra pipeline ed eseguiremo diverse analisi BI, per verificarne l'efficienza; useremo infine un modello di Sagemaker per ricercare la presenza di anomalie.

Per mantenere le cose interessanti, vogliamo anche acquisire i dati storici da due posizioni diverse: un bucket S3 e un database che risiede su un'istanza EC2 in una VPC diversa da quella della nostra pipeline ETL.

Useremo diversi job ETL per recuperare ed estrarre i dati puliti dalle tuple a disposizione e AWS Step Functions per orchestrare tutti i crawler e i job.

Kinesis Data Firehose recupererà continuamente i dati dei sensori e con AWS Athena interrogheremo le informazioni, dai dati aggregati e per sensore, per mostrare le statistiche grafiche in Amazon Quicksight.

Ecco un semplice schema che illustra i servizi coinvolti e il flusso completo.

*La nostra infrastruttura*

## Kinesis Data Firehose

Kinesis Data Firehose può essere utilizzato per ottenere dati quasi in tempo reale dai sensori, che sfruttano IoT Core SDK per connettersi ai dispositivi effettivi. Come visto in questo articolo, possiamo creare una "Cosa", generando così un **topic**. Collegandosi a tale **topic**, diversi dispositivi possono raccogliere le proprie metriche tramite Firehose inviando messaggi utilizzando il protocollo MQTT e, se necessario, IoT Core può anche gestire **l'autenticazione** del dispositivo.

Per iniziare a inviare i dati dei sensori, dobbiamo scaricare il kit di connessione dalla pagina AWS IoT seguendo le istruzioni presentate.

*Seleziona OS e linguaggio di programmazione per scaricare il connection kit*

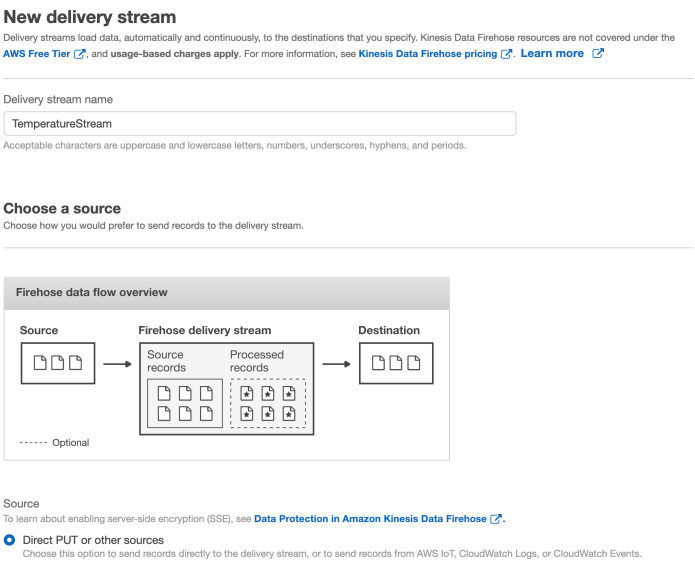
Una volta scaricato, inizializziamo un nuovo progetto Node.js e **installiamo AWS-IoT-device-SDK**. Dopodiché, è possibile eseguire lo script **start.sh** incluso, assicurandosi che tutti i certificati, scaricati insieme al kit, siano nella stessa directory. Ora possiamo creare uno script locale per inviare dati a un topic, passando i moduli richiesti e utilizzando **device.publish ("<topic>", payload)**:

```
const deviceModule = require('aws-iot-device-sdk').device;
const cmdLineProcess = require('aws-iot-device-sdk/examples/lib/cmdline');
…
device.publish('topic', JSON.stringify(payload));
```

I dati inviati sono strutturati in formato JSON con la seguente struttura:

```
{
   "timestamp": "YYYY-MM-DD HH:MM:SS",
   "room_id": "XXXX",
   "temperature": 99
}
```

Per creare un flusso di consegna di Firehose, andiamo alla dashboard del servizio **Kinesis Firehose** nella console Web di AWS, facciamo clic su "Crea flusso di consegna", selezioniamo un nome, quindi "Direct PUT or other sources" come in figura:




*Creare una nuova delivery stream di Firehose*

Lasciamo "Data transformation" e "Record format conversion" come di default. Selezioniamo una destinazione di S3 come target. Ricordiamoci di definire anche una **IoT Rule** per inviare i messaggi IoT a Firehose mediante delivery stream.

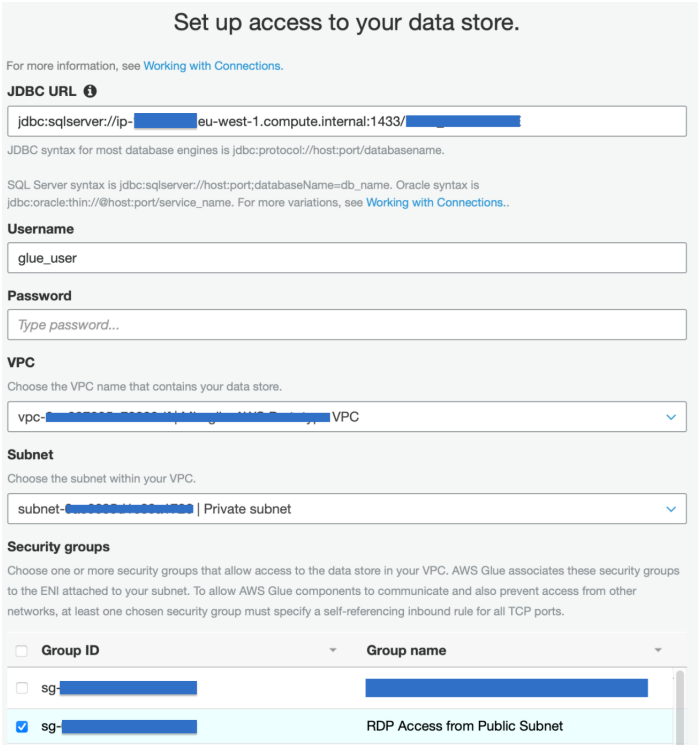
## Glue crawlers e connettori

AWS Glue può essere utilizzato per estrarre e trasformare dati da una moltitudine di origini dati diverse, grazie alla possibilità di definire diversi tipi di connettori.

**Database su istanza EC2**

Vogliamo essere in grado di generare un Glue Data Catalog da un database Microsoft SQL Server, che risiede su un'istanza EC2 in un' altra VPC. Per fare ciò, dobbiamo creare una connessione JDBC, che può essere eseguita facilmente accedendo alla pagina del servizio AWS Glue e aggiungendo una nuova connessione; questa si trova nella sezione "Catalogo dati – Database" del menu della barra laterale.

Basta aggiungere un nome alla connessione (che verrà utilizzata dal relativo Crawler Job), l'URL JDBC, seguendo la giusta convenzione per ORACLE DB, nome utente e password, VPC e sottorete richiesti.




*JDBC – parametri di connessione*

Per stabilire una connessione glue al database, dobbiamo creare una nuova VPC dedicata che verrà utilizzata solo da Glue. La VPC è connessa a quella che contiene il data-warehouse tramite peering VPC, ma sono possibili anche altre opzioni, ad esempio avremmo potuto utilizzare AWS Transit Gateway. Una volta stabilito il peering, dobbiamo ricordarci di aggiungere le rotte corrette, sia alla sottorete Glue che a quella del DB, in modo che le VPC possano scambiare traffico e di aprire il security group del DB, per consentire il traffico in entrata sulla porta pertinente, al security group di Glue nella nuova VPC.

**Dati su S3**

I dati su S3 non richiedono un connettore e possono essere configurati direttamente dalla console di AWS Glue. Creiamo un nuovo crawler, selezionando "data stores" per il **tipo di origine del crawler**; quindi selezioniamo anche "Crawl all folder". Dopodiché è solo questione di impostare il

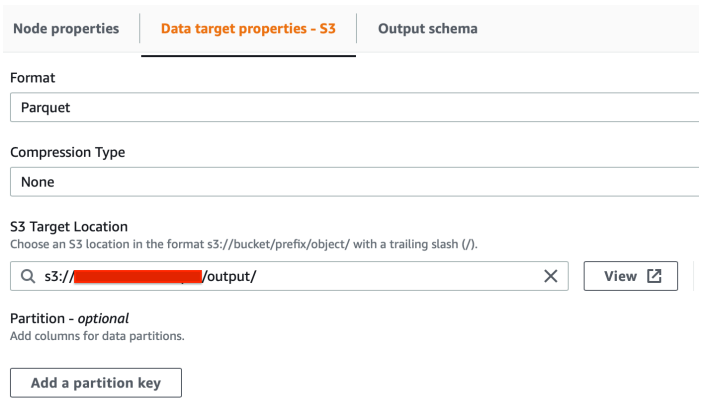
bucket S3, il ruolo IAM corretto e creare un nuovo Schema di GLue per questo crawler. Impostare anche “Run on demand”.

**Glue Job**

I Glue Jobs sono i passaggi della pipeline ETL. Consentono di estrarre, trasformare e salvare i dati in un datalake. Nel nostro esempio, vorremmo mostrare due diversi approcci: job **gestiti da AWS Glue Studio** e mediante l’utilizzo di **codice personalizzato**. Entrambi i job verranno successivamente richiamati da AWS Step Function.

Per i dati storici su S3, possiamo definire i job da Glue Studio. Per S3 selezionare le seguenti opzioni nell’ordine:

1. Nella pagina **Manage Jobs**, selezioniamo sorgente e destinazione da aggiungere alle opzioni del nodo. Quindi, scegliamo S3 come Source e comeTarget.
2. Clicchiamo su “S3 Data source”, quindi selezioniamo il bucket di origine.
3. Nella tab “Node Properties”, inseriamo un nome. Clicchiamo poi sulla tab “Data source properties – S3” nel pannello dei dettagli del nodo. Selezioniamo il nostro schema dalla lista di database nel Glue Data Catalog. Selezioniamo quindi la tabella corretta dal Catalogo.
4. Verifichiamo che il mapping sia corretto.
5. Nel Nodo “S3 Data target”, selezioniamo il bucket di destinazione, CSV come formato (parquet è meglio, ma abbiamo bisogno di CSV per il Random Cut forest), nessuna compressione.




*Proprietà del nodo di destinazione*

Per estrarre i dati dalla nostra istanza EC2, invece, abbiamo bisogno di un custom job. Per crearlo, dobbiamo scrivere noi stessi uno script, ma non preoccupatevi: è piuttosto semplice! Ecco i punti chiave che si devono conoscere per creare uno Spark Job con Glue: il processo ETL è composto da 6 aree distinte nello script:

**Import delle librerie**

Set base necessario al funzionamento dello script:

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrame
```

**Prepariamo i connettori e altre variabili**

Da usare all'interno dello script:

```
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
```

**Recuperate i Dynamic Frame dal Glue Catalog ottenuto tramite un Crawler**

I dynamic frame vengono utilizzati per fare query e trasformare i dati

```
rooms_temperatures_df = glueContext.create_dynamic_frame.from_catalog(database = "raw
_temperatures", table_name = "temperatures", transformation_ctx = "temperature_transfo
rms").toDF()
rooms_temperatures_df.createOrReplaceTempView("TEMPERATURES")
```

L'ultima linea di codice permette di modificare un dynamic frame.

**Applichiamo le operazioni di SQL**

Per estrarre informazioni specifiche

```
result = glueContext.sql("")
```

Nel nostro caso, abbiamo bisogno di generare 3 risultati distinti, uno per ogni room, e per questo usiamo un semplice **WHERE room_id = <value>**

**Applichiamo il nostro mapping**

Per generare uno schema di conversione

```
dynamicFrameResult = DynamicFrame.fromDF(result, glueContext, "Result")
applymapping = ApplyMapping.apply(frame = dynamicFrameResult, mappings = [("temp", "b
igint", "temp","bigint"), ("room_id", "string", "room_id","string"), ("timestamp", "s
tring", "timestamp","string")])
```

**Salviamo di nuovo su S3**

Per poter manipolare i dati in seguito

```
to_be_written = glueContext.write_dynamic_frame.from_options(frame = applymapping, co
nnection_type = "s3", connection_options = {"path": "s3://", "partitionKeys": ["times
tamp"]}, format = "csv", transformation_ctx = "to_be_written")
job.commit()
```

## Step Function

La Step Function rappresenta il nucleo, la logica della nostra soluzione di esempio. Il suo scopo principale è gestire tutti i lavori ETL, mantenerli sincronizzati e gestire gli errori. Un vantaggio è che possiamo usare la Step Function per regolare i dati iniettati nel bucket S3 centrale, dove salviamo tutti i valori puliti.

Per iniziare, questo è lo schema della step function che abbiamo usato per questo esempio:

*La nostra pipeline di esempio*

Nel nostro esempio ci sono un paio di hint interessanti che vorremmo condividere su Step Function; in primo luogo, abbiamo 2 crawler loop principali: il primo, ha branch e gestisce 2 crawler contemporaneamente (uno standard per S3 e uno per il database EC2 che è quello personalizzato); il secondo prende tutti i dati recuperati sia dalle sorgenti di dati storici che da quella in real-time (da Kinesis Firehose) ed estrae i set di dati per room, che verranno poi utilizzati con Amazon SageMaker.

Poiché i crawler sono asincroni, non possiamo aspettarli, quindi abbiamo dovuto creare 2 cicli di attesa per entrambi gli step di esecuzione.

AWS Lambda viene utilizzato per chiamare le API di AWS Glue per avviare i job che abbiamo configurato in precedenza.

Per darvi qualche spunto, ecco alcune parti interessanti descritte nel file JSON che rappresenta la macchina a stati.

```
"Type": "Parallel",
  "Branches": [
        {
          "StartAt": "Import Raw from EXTERNAL_DB",
          "States": {
            "Import Raw from EXTERNAL_DB": {
              "Type": "Task",
              "Resource": "arn:aws:states:::glue:startJobRun.sync",
```

In AWS Step Function, we can launch tasks in parallel (for us, the two historical data glue jobs) using “Type: Parallel” and “Branches”. Also after the key “Branches”, it is possible to retrieve the parallel result.

In AWS Step Function, possiamo avviare attività in parallelo (per noi, i due processi glue sui dati storici) utilizzando “Type: Parallel” e “Branches”. Inoltre dopo la chiave “Branches”, vediamo come è possibile recuperare il risultato dei job in parallelo.

```
"ResultPath": "$.ParallelExecutionOutput",
"Next": "Start LAKE_DATA Crawler"
```

Possiamo eseguire un Glue Job sincrono definito nella console, passando il nome del job stesso e anche abilitando la generazione di un Glue catalog durante il processo.

```
"Parameters": {
                "JobName": "EXTERNAL_DB_IMPORT_TO_RAW",
                "Arguments": {
                  "--enable-glue-datacatalog": "true",
```

Possiamo inoltre risolvere le eccezioni del codice direttamente in Step Function portandoci in uno step di errore mediante la chiave “Catch”:

```
"Catch": [
        {
          "ErrorEquals": [
            "States.TaskFailed"
          ],
          "Next": "Data Pipeline Failure"
        }
],
```

Poiché non abbiamo un modo standard per attendere il completamento dei lavori, utilizziamo l’output dei lavori paralleli e un ciclo di attesa di Step Functions per verificare se l’operazione è

stata eseguita; per questo, usiamo la chiave “Wait”:

```
"Wait for LAKE_DATA Crawler": {
      "Type": "Wait",
      "Seconds": 5,
      "Next": "Check LAKE_DATA Crawler"
},
```

Il resto del flusso è praticamente una ripetizione di questi componenti.

Il fatto interessante è che possiamo applicare alcune condizioni di partenza per alterare l’esecuzione del flusso, come evitare alcuni lavori se non necessari al momento o anche eseguire un’altra macchina a stati da un passo preciso per prendere il nostro esempio e modularizzare le parti più complicate.

## Athena e Quicksight

Athena può generare tabelle che possono essere interrogate utilizzando il linguaggio SQL standard, non solo: i risultati delle query Athena possono essere importati in Amazon QuickSight per generare rapidamente grafici e report, basati sui tuoi dati.

Nel nostro flusso di lavoro, è possibile eseguire query Athena sul bucket S3 di destinazione che contiene sia i dati della temperatura globale sia quelli specifici dei sensori. Vediamo rapidamente come fare:

1. Se abbiamo già creato un Glue Crawler, avremo un Datasource e una table.
2. Selezioniamo il database e la tabella nella dashboard di Athena, nella sidebar a sinistra (abbiamo utilizzato temperatures_db e temperatures, ottenuti dai nostri crawlers).
3. Creiamo una semplice query che possa essere utilizzata poi in QuickSight per mostrare un grafico, ad esempio, una semplice “SELECT * FROM temperatures”.

Tramite questi 3 step, Athena genererà il risultato della query come mostrato in figura:

*Athena – query di esempio*

Un paio di trucchi interessanti quando si lavora con Athena:

- Evitiamo di usare il carattere “-” nel nome del database, usiamo “_” invece.
- Per quanto ci sia possibile evitiamo che i crawlers scansionino cartelle contenenti file con schema differenti, possiamo farlo solo con quelli che condividono lo stesso schema (ad esempio

per generare delle partition).

Quicksight può leggere query di Athena e presentare grafici e diagrammi da esse. È molto semplice: andiamo alla pagina del servizio Quicksight e seguiamo uno dei tanti tutorial a riguardo, tenendo presente alcune cose importanti:

- Quicksight non è **direttamente incluso nelle risorse del proprio account, bisogna abilitare una subscription** a pagamento (ci sono 60 giorni di trial però).
- Per poter accedere ad Athena, **Quicksight necessita che il proprio ruolo ottenga accesso completo ad Athena.**
- Un certificato SSL valido deve essere rilasciato e presente, per esempio utilizzando Amazon ACM.

Se non vogliamo, o non possiamo, utilizzare Quicksight, possiamo sempre chiamare direttamente le API di Athena e creare la nostra dashboard customizzata da zero.

## SageMaker: Random Cut Forest anomaly detection

L'algoritmo di apprendimento automatico che esploreremo in questo articolo si chiama Random Cut Forest. L'algoritmo prende un insieme di data point casuali (Random), li taglia allo **stesso numero di punti** e crea alberi (Cut). Infine, controlla tutti gli alberi insieme (Forest) per verificare se un particolare data point deve essere considerato un'anomalia.

In generale, un albero è un modo ordinato di memorizzare dati numerici e, per crearlo, suddividiamo casualmente i data point fino a quando è possibile isolare il punto che stiamo testando per determinare se si tratta di un'anomalia. Ogni volta che suddividiamo i punti viene creato un nuovo livello dell'albero.

SageMaker offre un'implementazione managed di Random Cut Forest che accetta data points in formato CSV. Possiamo recuperare facilmente i dataset con:

```
data_location = f"s3://{bucket}/{key}"
df=pd.read_csv(data_location,delimiter=',')
```

I dati contengono un **timestamp**, il **valore della temperatura** in C ° e un **room_id**, che identifica una particolare stanza in cui è stato installato il sensore. Abbiamo già utilizzato la nostra Step Function per dividere i dati provenienti da stanze diverse in modo da poter passare direttamente il CSV all'Estimator.

*Sample data*

Abbiamo fatto riferimento a questo articolo per verificare come i dati possano essere passati all'Estimator. Stando alla documentazione ufficiale, dobbiamo passare 3 iperparametri principali:

- **num_samples_per_tree** – il numero di punti estrapolati in modo casuale da passare ad ogni albero. **1/num_samples_per_tree** dovrebbe approssimare il valore stimato di **anomalies/points** nel dataset.
- **num_trees** – il numero di alberi creati nella foresta. Ogni albero impara da un modello differente, generato da sample di dati differenti.
- **feature_dim** – la dimensione di ogni data point.

L'Estimator è definito in questo modo:

```
import sagemaker
from sagemaker import RandomCutForest

execution_role = sagemaker.get_execution_role()
sagemaker_session = sagemaker.Session()
bucket = “”
prefix = “”

rcf = RandomCutForest(
    role=execution_role,
    instance_count=1,
    instance_type="ml.m4.xlarge",
    data_location=f"s3://{bucket}/{prefix}",
    output_path=f"s3://{bucket}/{prefix}/output",
    num_samples_per_tree=512,
    num_trees=50,
)
rcf.fit(rcf.record_set(df.value.to_numpy().reshape(-1, 1)))
```

Alcune informazioni da tenere in considerazione sono che generiamo **execution_role** e **sagemaker_session** utilizzando i metodi incorporati di SageMaker. Per il nostro training utilizziamo un'istanza **ml.m4xlarge**, mentre per l'inferenza abbiamo utilizzato una **ml.c5.xlarge** come suggerito dalla documentazione. Non sprechiamo crediti per le istanze GPU poiché l'algoritmo RCF non tiene conto della GPU.

Per il deploy possiamo utilizzare l'approccio standard:

```
rcf.deploy(initial_instance_count=1, instance_type="ml.m4.xlarge")
```

E così ci siamo! Abbiamo raggiunto la fine di questo workflow. Vediamo alcune referenze e riassumiamo quanto analizzato fin'ora.

## Referenze


- https://docs.aws.amazon.com/step-functions/latest/dg/create-sample-projects.html
- https://docs.aws.amazon.com/glue/latest/dg/connection-using.html
- https://docs.aws.amazon.com/glue/latest/dg/connection-defining.html
- https://docs.aws.amazon.com/glue/latest/dg/populate-add-connection.html
- https://docs.aws.amazon.com/athena/latest/ug/glue-athena.html
- https://docs.aws.amazon.com/quicksight/latest/user/create-a-data-set-athena.html
- https://docs.aws.amazon.com/sagemaker/latest/dg/randomcutforest.html
- https://docs.aws.amazon.com/vpc/latest/peering/what-is-vpc-peering.html
- https://aws.amazon.com/blogs/big-data/derive-insights-from-iot-in-minutes-using-aws-iot-amazon-kinesis-firehose-amazon-athena-and-amazon-quicksight/


## Takeaways

In questo articolo, abbiamo visto molti servizi di AWS perfettamente adatti per l'analisi dei dati quando si tratta di scenari quasi in tempo reale. Abbiamo discusso di AWS Step Function e di come può essere utilizzata per orchestrare flussi di lavoro non lineari, offrendo agli sviluppatori la possibilità di avere più scelte nella manipolazione ed estrazione dei dati per diversi tipi di analisi.

AWS Glue si è dimostrato sufficientemente flessibile da prendersi cura di origini di dati residenti in luoghi diversi: istanze EC2, S3 e in account diversi. È stata una scelta perfetta, anche per la semplicità di impostare Spark Job. Abbiamo visto in particolare come connettersi a un'origine dati utilizzando una connessione JDBC.

Athena ha dimostrato di essere lo strumento perfetto per estrarre i risultati ETL per la fruizione da parte della Business Intelligence e Quicksight è la scelta più ovvia per mostrare i risultati, poiché è nativamente compatibile con le query di Athena.

Come in molti altri scenari che abbiamo affrontato, Kinesis Data Firehose è stato utilizzato anche per trasferire dati quasi in tempo reale a S3 da una fonte non AWS.

Abbiamo anche visto come Amazon S3 sia sempre un must quando si tratta di flussi di lavoro di big data, problemi di machine learning e creazione di data lake. I suoi standard di durabilità, oltre alla compatibilità con qualsiasi altro servizio AWS, lo rendono la scelta perfetta sia per l'archiviazione a lungo termine che per il buffer intermedio.

Per concludere, abbiamo fornito alcuni suggerimenti su come manipolare i dati in SageMaker per eseguire inferenze per il rilevamento di anomalie.

Questo conclude il nostro viaggio di oggi, come sempre sentitevi liberi di commentare e raggiungerci per discutere qualsiasi domanda, dubbio o idea che vi venga in mente. Saremo lieti di rispondere il prima possibile!

Fino alla nostra prossima storia, stay safe e #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