

The Delta Lake Series Streaming

Using Delta Lake to express
computation on streaming data



What's inside?

The Delta Lake Series of eBooks is published by Databricks to help leaders and practitioners understand the full capabilities of Delta Lake as well as the landscape it resides in. This eBook, **The Delta Lake Series – Streaming**, focuses on a number of use cases that showcase Delta Lake's robust streaming capabilities so you can use them to your benefit.

What's next?

After reading this eBook, you'll not only understand what Delta Lake offers, but you'll also understand how to use Delta Lake to express computation on streaming data in the same way you express a batch computation on static data.

Here's what you'll find inside

Introduction

What is Delta Lake?

Chapter

01

How Delta Lake Solves Common Pain Points in Streaming

Chapter

02

USE CASE #1: Simplifying Streaming Stock Data Analysis Using Delta Lake

Chapter

03

USE CASE #2: How Tilting Point Does Streaming Ingestion Into Delta Lake

Chapter

04

USE CASE #3: Building a Quality of Service Analytics Solution for Streaming Video Services



What is Delta Lake?

[Delta Lake](#) is a unified data management system that brings data reliability and fast analytics to cloud data lakes. Delta Lake runs on top of existing data lakes and is fully compatible with Apache Spark™ APIs.

At Databricks, we've seen how Delta Lake can bring reliability, performance and lifecycle management to data lakes. With Delta Lake, there will be no more malformed data ingestion, difficulties deleting data for compliance, or issues modifying data for data capture.

With Delta Lake, you can accelerate the velocity that high-quality data can get into your data lake, and the rate that teams can leverage that data with a secure and scalable cloud service.

How Delta Lake Solves Common
Pain Points in Streaming

CHAPTER 01



01

How Delta Lake Solves Common Pain Points in Streaming



The pain points of a traditional streaming and data warehousing solution can be broken into two groups: data lake and data warehouse pains.

Data lake pain points

While data lakes allow you to flexibly store an immense amount of data in a file system, there are many pain points including (but not limited to):

- Consolidation of streaming data from many disparate systems is difficult.
- Updating data in a data lake is nearly impossible, and much of the streaming data needs to be updated as changes are made. This is especially important in scenarios involving financial reconciliation and subsequent adjustments.
- Query speeds for a data lake are typically very slow.
- Optimizing storage and file sizes is very difficult and often requires complicated logic.

Data warehouse pain points

The power of a data warehouse is that you have a persistent performant store of your data. But the pain points for building modern continuous applications include (but are not limited to):

- Constrained to SQL queries (i.e., no machine learning or advanced analytics).
- Accessing streaming data and stored data together is very difficult, if at all possible.
- Data warehouses do not scale very well.
- Tying compute and storage together makes using a warehouse very expensive.

How Delta Lake on Databricks solves these issues

[Delta Lake](#) is a unified data management system that brings data reliability and performance optimizations to cloud data lakes. More succinctly, Delta Lake combines the advantages of data lakes and data warehouses with Apache Spark™ to allow you to do incredible things.

- Delta Lake, along with Structured Streaming, makes it possible to analyze streaming and historical data together at high speeds.
- When Delta Lake tables are used as sources and destinations of streaming big data, it is easy to consolidate disparate data sources.
- Upserts are supported on Delta Lake tables.
- Delta Lake is ACID compliant, making it easy to create a compliant data solution.
- Easily include machine learning scoring and advanced analytics into ETL and queries.
- Decouples compute and storage for a completely scalable solution.

In the following use cases, we'll share what this looks like in practice.📍





USE CASE #1

Simplifying Streaming Stock Data
Analysis Using Delta Lake

CHAPTER 02

02 Simplifying Streaming Stock Data Analysis Using Delta Lake



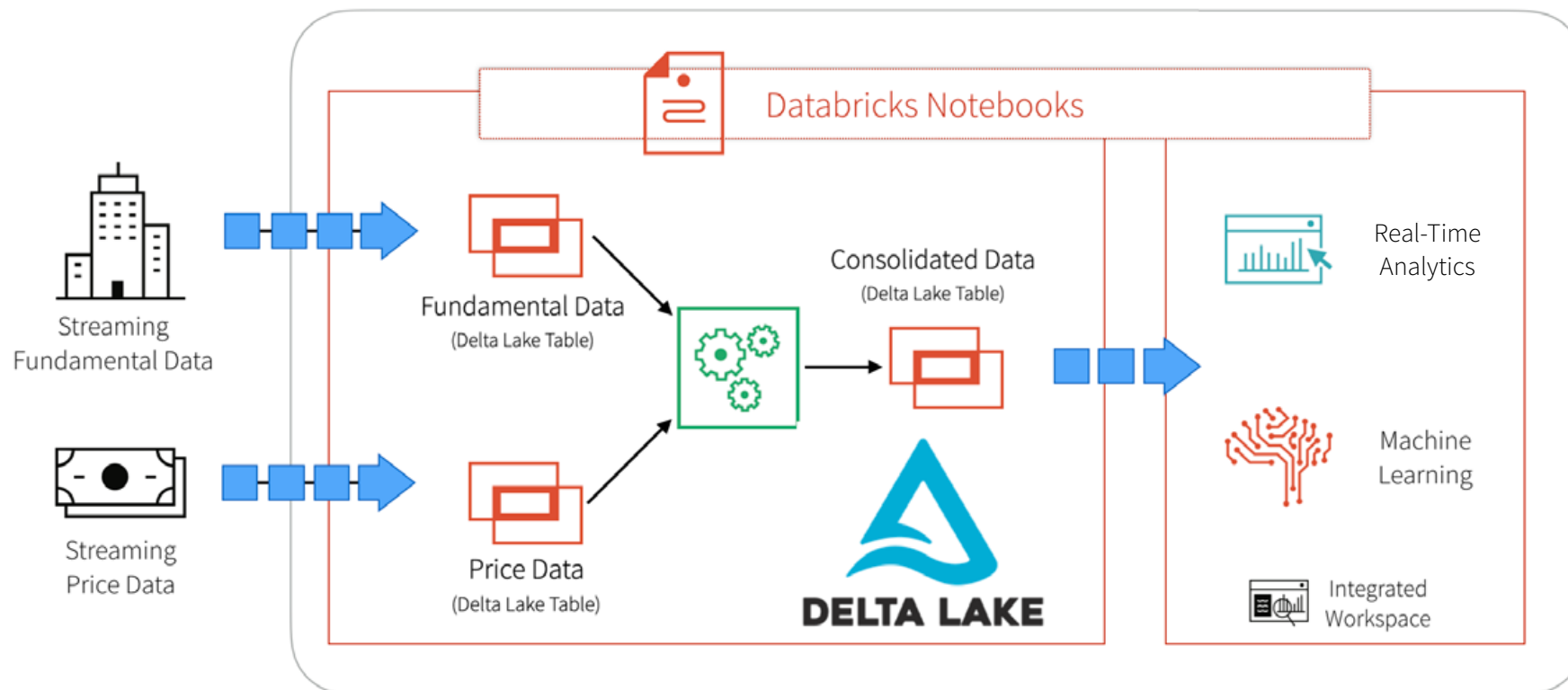
Real-time analysis of stock data is a complicated endeavor. After all, there are many challenges in maintaining a streaming system and ensuring transactional consistency of legacy and streaming data concurrently.

Thankfully, [Delta Lake](#) helps solve many of the pain points of building a streaming system to analyze stock data in real time. In this section, we'll share how to simplify the streaming of stock data analysis using Delta Lake.

In the following diagram, you can see a high-level architecture that simplifies this problem. We start by ingesting two different sets of data into two Delta Lake tables. The two data sets are stock prices and fundamentals.

After ingesting the data into their respective tables, we then join the data in an ETL process and write the data out into a third Delta Lake table for downstream analysis.

Delta Lake helps solve these problems by combining the scalability, streaming and access to the advanced analytics of Apache Spark with the performance and ACID compliance of a data warehouse.



Implement your streaming stock analysis solution with Delta Lake

Delta Lake and Apache Spark do most of the work for our solution; you can try out the full [notebook](#) and follow along with the code samples below.

As noted in the preceding diagram, we have two data sets to process – one for fundamentals and one for price data. To create our two Delta Lake tables, we specify the `.format('delta')` against our Databricks File System (DBFS) locations.

```
# Create Fundamental Data (Databricks Delta table)
dfBaseFund = spark \
  .read \
  .format('delta') \
  .load('/delta/stocksFundamentals')

# Create Price Data (Databricks Delta table)
dfBasePrice = spark \
  .read \
  .format('delta') \
  .load('/delta/stocksDailyPrices')
```

While we're updating the `stockFundamentals` and `stocksDailyPrices`, we will consolidate this data through a series of ETL jobs into a consolidated view (`stocksDailyPricesWFund`).

With the following code snippet, we can determine the start and end date of available data and then combine the price and fundamentals data for that date range into DBFS.

```
# Determine start and end date of available data
row = dfBasePrice.agg(
    func.max(dfBasePrice.price_date).alias("maxDate"),
    func.min(dfBasePrice.price_date).alias("minDate")
).collect()[0]
startDate = row["minDate"]
endDate = row["maxDate"]

# Define our date range function
def daterange(start_date, end_date):
    for n in range(int((end_date - start_date).days)):
        yield start_date + datetime.timedelta(n)

# Define combinePriceAndFund information by date and
def combinePriceAndFund(theDate):
    dfFund = dfBaseFund.where(dfBaseFund.price_date == theDate)
    dfPrice = dfBasePrice.where(
dfBasePrice.price_date == theDate
    ).drop('price_date')
    # Drop the updated column
    dfPriceWFund = dfPrice.join(dfFund, ['ticker']).drop('updated')
```

```
# Save data to DBFS
dfPriceWFund
.write
.format('delta')
.mode('append')
.save('/delta/stocksDailyPricesWFund')

# Loop through dates to complete fundamentals + price ETL process
for single_date in daterange(
startDate, (endDate + datetime.timedelta(days=1))
):
    print 'Starting ' + single_date.strftime('%Y-%m-%d')
    start = datetime.datetime.now()
    combinePriceAndFund(single_date)
    end = datetime.datetime.now()
    print (end - start)
```

Now we have a stream of consolidated fundamentals and price data that is being pushed into [DBFS](#) in the `/delta/stocksDailyPricesWFund` location. We can build a Delta Lake table by specifying `.format("delta")` against that DBFS location.

```
dfPriceWithFundamentals = spark
.readStream
.format("delta")
.load("/delta/stocksDailyPricesWFund")

// Create temporary view of the data
dfPriceWithFundamentals.createOrReplaceTempView("priceWithFundamentals")
```

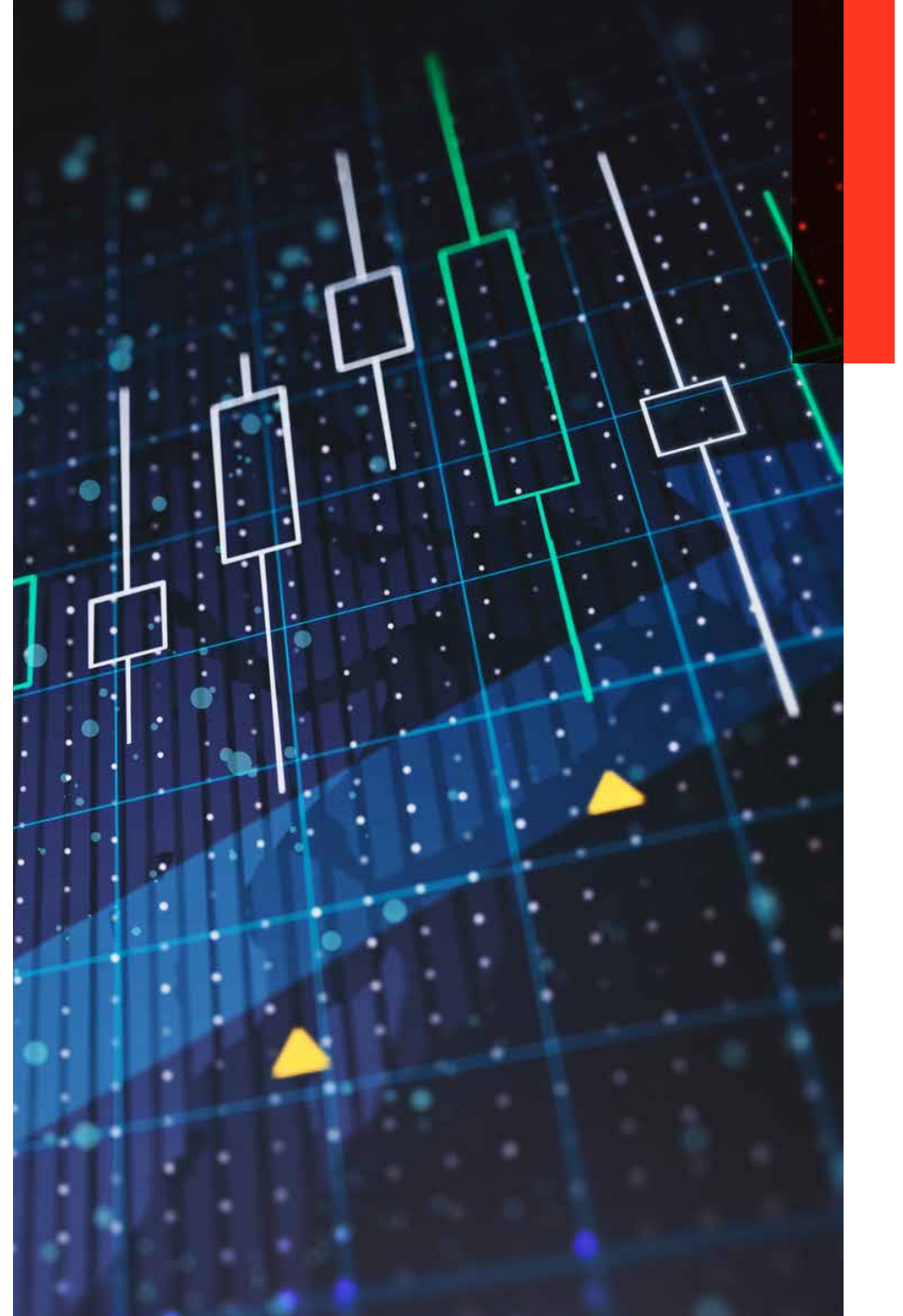
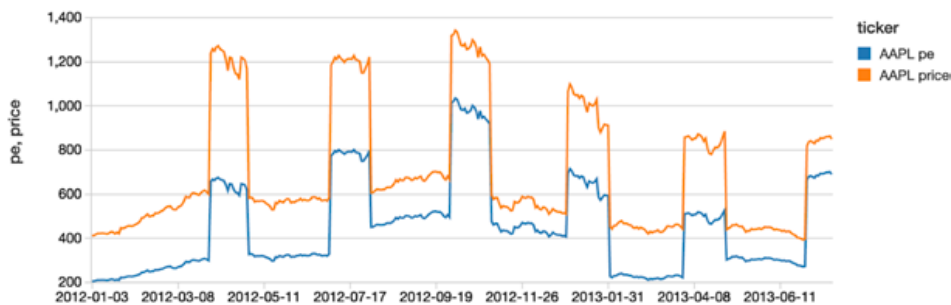

Now that we have created our initial Delta Lake table, let's create a view that will allow us to calculate the price/earnings ratio in real time (because of the underlying streaming data updating our Delta Lake table).

```
%sql
CREATE OR REPLACE TEMPORARY VIEW viewPE AS
select ticker,
       price_date,
       first(close) as price,
       (close/eps_basic_net) as pe
from priceWithFundamentals
where eps_basic_net > 0
group by ticker, price_date, pe
```

Analyze streaming stock data in real time

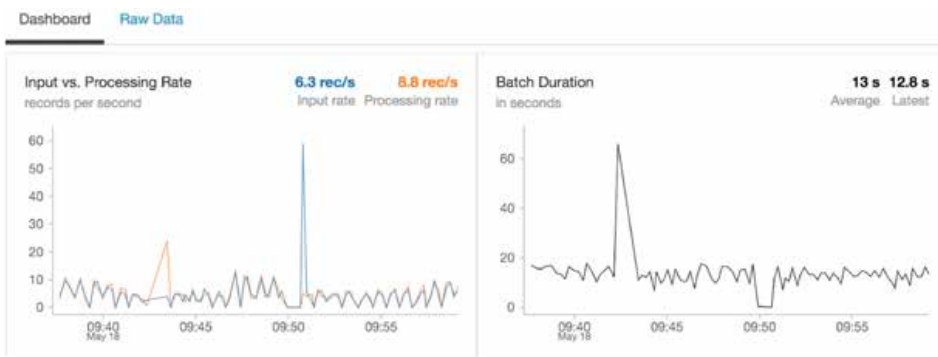
With our view in place, we can quickly analyze our data using Spark SQL.

```
%sql
select *
from viewPE
where ticker == "AAPL"
order by price_date
```

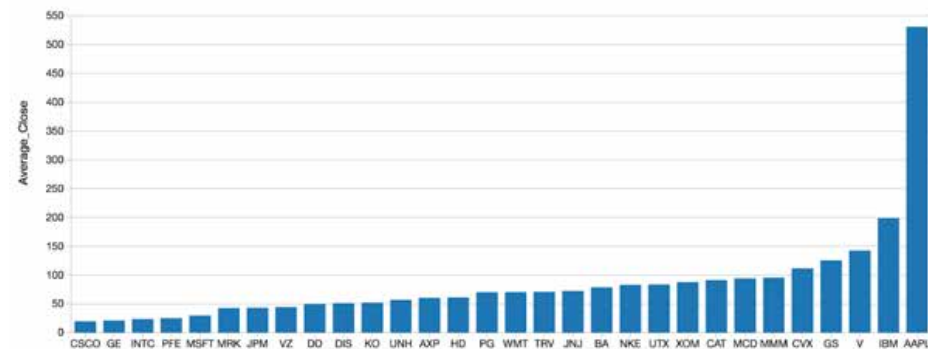




As the underlying source of this consolidated data set is a Delta Lake table, this view isn't just showing the batch data but also any new streams of data that are coming in as per the following streaming dashboard.



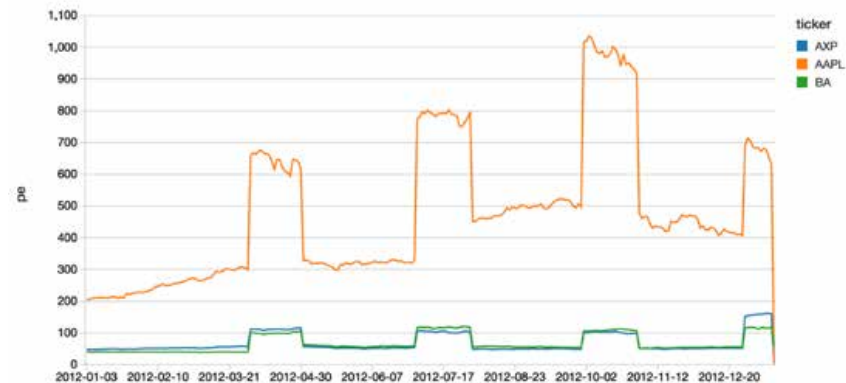
Underneath the covers, Structured Streaming isn't just writing the data to Delta Lake tables but also keeping the state of the distinct number of keys (in this case ticker symbols) that need to be tracked.



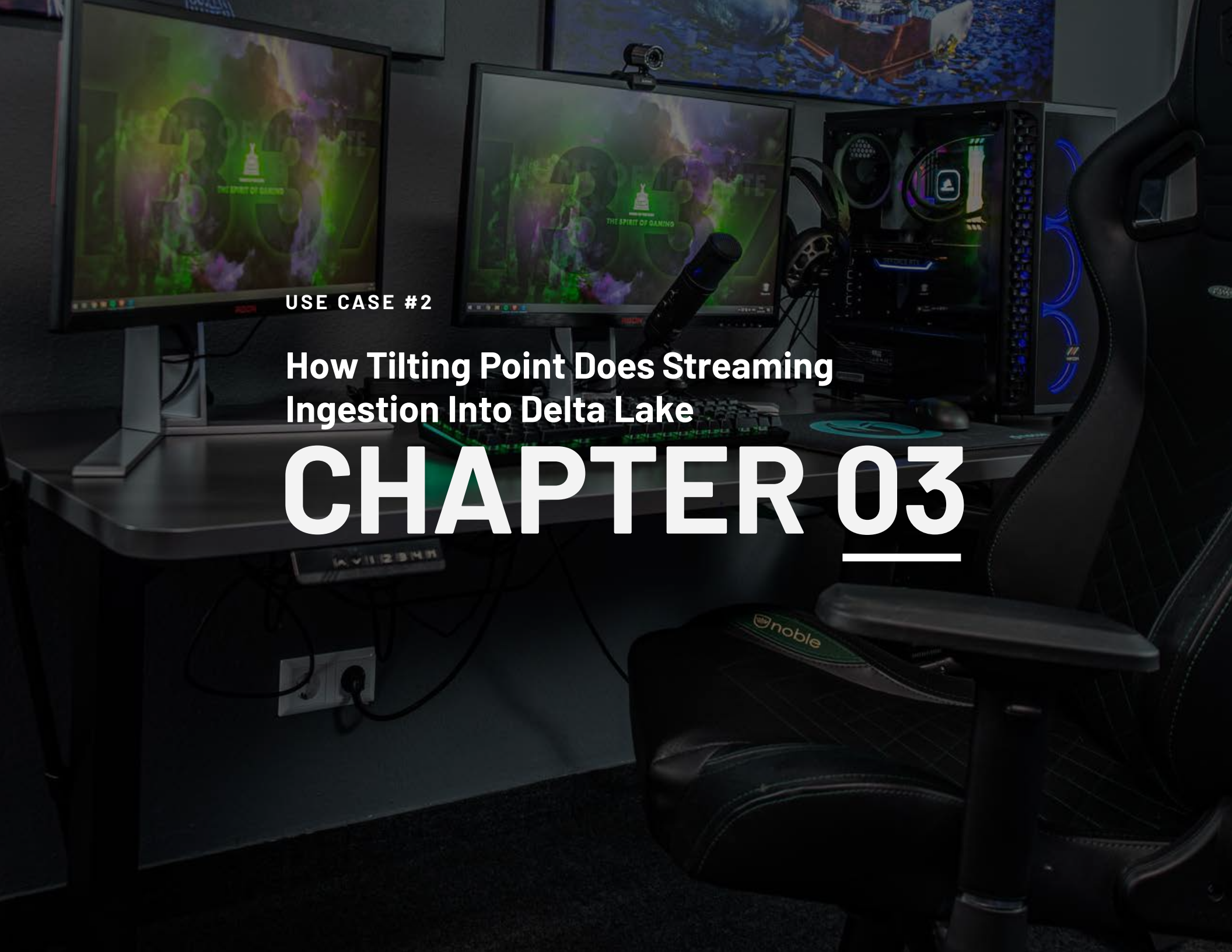
Because you are using Spark SQL, you can execute aggregate queries at scale and in real time.

```
%sql
SELECT ticker, AVG(close) as Average_Close
FROM priceWithFundamentals
GROUP BY ticker
ORDER BY Average_Close
```

In closing, we demonstrated how to simplify streaming stock data analysis using [Delta Lake](#). By combining Spark Structured Streaming and Delta Lake, we can use the Databricks integrated workspace to create a performant, scalable solution that has the advantages of both data lakes and data warehouses.



The [Databricks Unified Data Platform](#) removes the data engineering complexities commonly associated with streaming and transactional consistency, enabling data engineering and data science teams to focus on understanding the trends in their stock data.



USE CASE #2

How Tilting Point Does Streaming
Ingestion Into Delta Lake

CHAPTER 03

03

How Tilting Point Does Streaming Ingestion Into Delta Lake

Tilting Point is a new-generation games partner that provides top development studios with expert resources, services and operational support to optimize high-quality live games for success. Through its user acquisition fund and its world-class technology platform, Tilting Point funds and runs performance marketing management and live games operations to help developers achieve profitable scale.

By leveraging Delta Lake, Tilting Point is able to leverage quality data and make it readily available for analytics to improve the business. Diego Link, VP of Engineering at Tilting Point, provided insights for this use case.

The team at Tilting Point was running daily and hourly batch jobs for reporting on game analytics. They wanted to make their reporting near real-time, getting insights within 5-10 minutes.

They also wanted to make their in-game LiveOps decisions based on real-time player behavior for giving real-time data to a bundles-and-offer system, provide up-to-the-minute alerting on LiveOPs changes that actually might have unforeseen detrimental effects and even alert on service interruptions in game operations. The goal was to ensure that the game experience was as robust as possible for their players.

Additionally, they had to store encrypted Personally Identifiable Information (PII) data separately in order to maintain GDPR compliance.

How data flows and associated challenges

Tilting Point has a proprietary software development kit that developers integrate with to send data from game servers to an ingest server hosted in AWS. This service removes all PII data and then sends the raw data to an Amazon Firehose endpoint. Firehose then dumps the data in JSON format continuously to S3.

To clean up the raw data and make it available quickly for analytics, the team considered pushing the continuous data from Firehose to a message bus (e.g., Kafka, Kinesis) and then using [Apache Spark's Structured Streaming](#) to continuously process data and write to Delta Lake tables.

While that architecture sounds ideal for low latency requirements of processing data in seconds, Tilting Point didn't have such low latency needs for their ingestion pipeline. They wanted to make the data available for analytics in a few minutes, not seconds. Hence they decided to simplify our architecture by eliminating a message bus and instead use S3 as a continuous source for their structured streaming job. But the key challenge in using S3 as a continuous source is identifying files that changed recently.

Listing all files every few minutes has two major issues:

- **Higher latency:** Listing all files in a directory with a large number of files has high overhead and increases processing time.
- **Higher cost:** Listing lots of files every few minutes can quickly add to the S3 cost.

Leveraging Structured Streaming with blob store as source and Delta Lake tables as sink

To continuously stream data from cloud blob storage like S3, Tilting Point uses [Databricks' S3-SQS source](#). The S3-SQS source provides an easy way to incrementally stream data from S3 without the need to write any state management code on what files were recently processed.



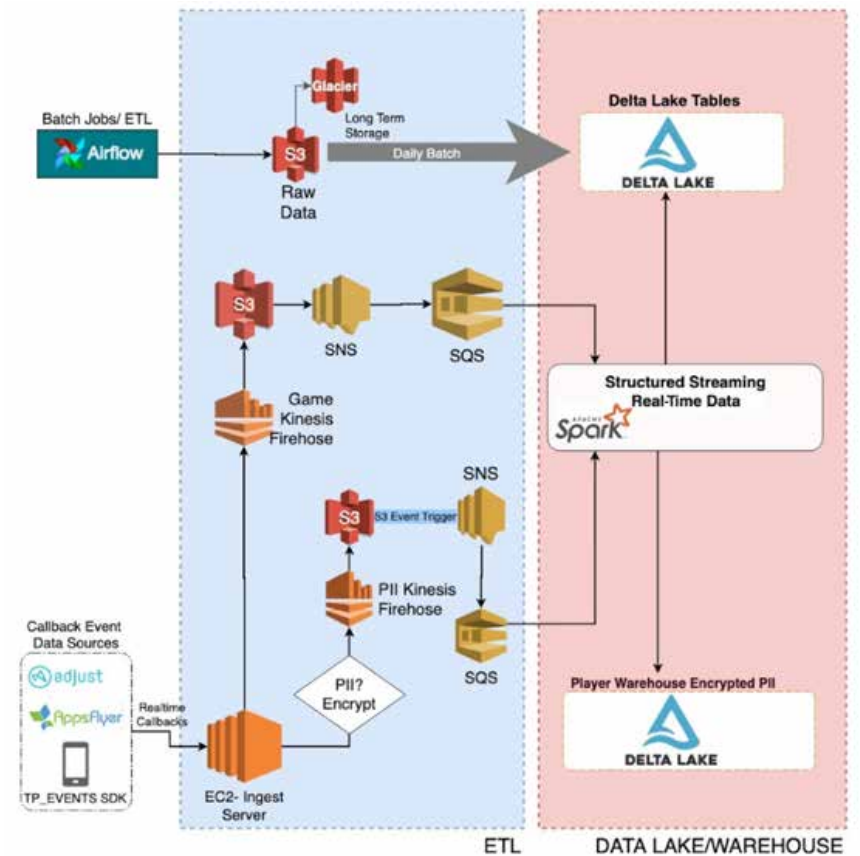


This is how Tilting Point's ingestion pipeline looks:

- [Configure Amazon S3 event notifications](#) to send new file arrival information to SQS via SNS.
- Tilting Point uses the S3-SQS source to read the new data arriving in S3. The S3-SQS source reads the new file names that arrived in S3 from SQS and uses that information to read the actual file contents in S3. An example code below:

```
spark.readStream \
  .format("s3-sqs") \
  .option("fileFormat", "json") \
  .option("queueUrl", ...) \
  .schema(...) \
  .load()
```

- Tilting Point's structured streaming job then cleans up and transforms the data. Based on the game data, the streaming job uses the foreachBatch API of Spark streaming and writes to 30 different Delta Lake tables.
- The streaming job produces lots of small files. This affects performance of downstream consumers. So, an optimize job runs daily to compact small files in the table and store them as right file sizes so that consumers of the data have good performance while reading the data from Delta Lake tables. Tilting Point also runs a weekly optimize job for a second round of compaction.



Architecture showing continuous data ingest into Delta Lake tables

The above Delta Lake ingestion architecture helps in the following ways:

- **Incremental loading:** The S3-SQS source incrementally loads the new files in S3. This helps quickly process the new files without too much overhead in listing files.
- **No explicit file state management:** There is no explicit file state management needed to look for recent files.
- **Lower operational burden:** Since we use S3 as a checkpoint between Firehose and Structured Streaming jobs, the operational burden to stop streams and re-process data is relatively low.
- **Reliable ingestion:** Delta Lake uses [optimistic concurrency control](#) to offer ACID transactional guarantees. This helps with reliable data ingestion.
- **File compaction:** One of the major problems with streaming ingestion is tables ending up with a large number of small files that can affect read performance. Before Delta Lake, we had to set up a different table to write the compacted data. With Delta Lake, thanks to ACID transactions, we can compact the files and rewrite the data back to the same table safely.
- **Snapshot isolation:** Delta Lake's snapshot isolation allows us to expose the ingestion tables to downstream consumers while data is being appended by a streaming job and modified during compaction.
- **Rollbacks:** In case of bad writes, [Delta Lake's Time Travel](#) helps us roll back to a previous version of the table.

In this section, we walked through Tilting Point's use cases and how they do streaming ingestion using Databricks' S3-SQS source into Delta Lake tables efficiently without too much operational overhead to make good quality data readily available for analytics.👏



USE CASE #3

**How to Build a Quality of Service Analytics
Solution for Streaming Video Services**

CHAPTER 04

04 How to Build a Quality of Service Analytics Solution for Streaming Video Services



As traditional pay TV [continues to stagnate](#), content owners have embraced direct-to-consumer (D2C) subscription and ad-supported streaming for monetizing their libraries of content. For companies whose entire business model revolved around producing great content, which they then licensed to distributors, the shift to now owning the entire glass-to-glass experience has required new capabilities, such as building media supply chains for content delivery to consumers, supporting apps for a myriad of devices and operating systems, and performing customer relationship functions like billing and customer service.

With most services renewing on a monthly basis, subscription service operators need to prove value to their subscribers at all times. General quality of streaming video issues (encompassing buffering, latency, pixelation, jitter, packet loss and the blank screen) have significant business impacts, whether it's increased [subscriber churn](#) or [decreased video engagement](#).

When you start streaming, you realize there are so many places where breaks can happen and the viewer experience can suffer. There may be an issue at the source in the servers on-premises or in the cloud; in transit at either the CDN level or ISP level or the viewer's home network; or at the playout level with player/client issues. What breaks at $n \times 104$ concurrent streamers is different from what breaks at $n \times 105$ or $n \times 106$. There is no pre-release testing that can quite replicate real-world users and their ability to push even the most redundant systems to their breaking point as they

channel surf, click in and out of the app, sign on from different devices simultaneously and so on. And because of the nature of TV, things will go wrong during the most important, high-profile events drawing the largest audiences. If you start [receiving complaints on social media](#), how can you tell if they are unique to that one user or rather regional or a national issue? If national, is it across all devices or only certain types (e.g., possibly the OEM updated the OS on an older device type, which ended up causing compatibility issues with the client)?

Identifying, remediating and preventing viewer quality of experience issues becomes a big data problem when you consider the number of users, the number of actions they are taking and the number of handoffs in the experience (servers to CDN to ISP to home network to client). Quality of Service (QoS) helps make sense of these streams of data so you can understand what is going wrong, where and why. Eventually you can get into predictive analytics around what could go wrong and how to remediate it before anything breaks.

Databricks Quality of Service solution overview

The aim of this solution is to provide the core for any streaming video platform that wants to improve their QoS system. It is based on the [AWS Streaming Media Analytics Solution](#) provided by AWS Labs, which we then built on top of to add Databricks as a Unified Data Analytics Platform for both the real-time insights and the advanced analytics capabilities.

[By using Databricks](#), streaming platforms can get faster insights by always leveraging the most complete and recent data sets powered by robust and reliable data pipelines. This decreases time to market for new features by accelerating data science using a collaborative environment. It provides support for managing the end-to-end machine learning lifecycle and reduces operational costs across all cycles of software development by having a unified platform for both data engineering and data science.



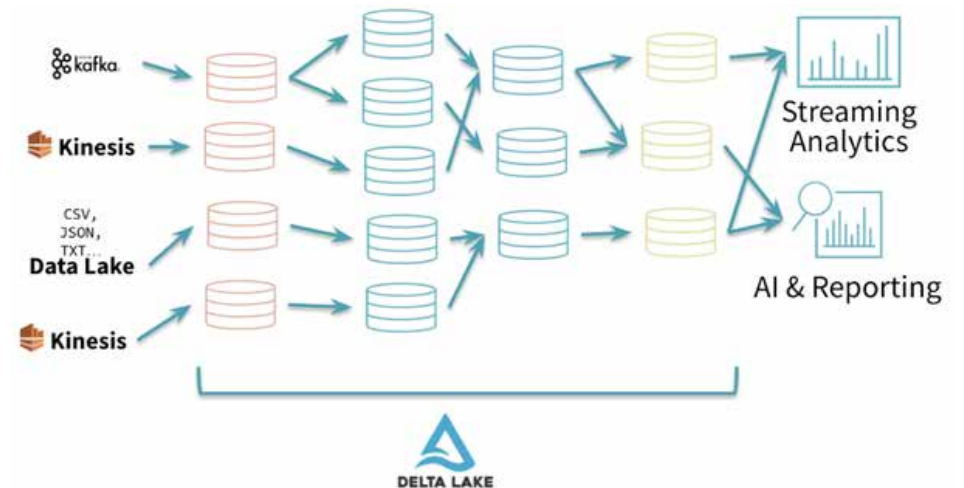


Video QoS solution architecture

With complexities like low-latency monitoring alerts and highly scalable infrastructure required for peak video traffic hours, the straightforward architectural choice was the Delta Architecture – both standard big data architectures like Lambda and Kappa Architectures have disadvantages around the operational effort required to maintain multiple types of pipelines (streaming and batch) and lack support for a unified data engineering and data science approach.

The Delta Architecture is the next-generation paradigm that enables all the data personas in your organization to be more productive:

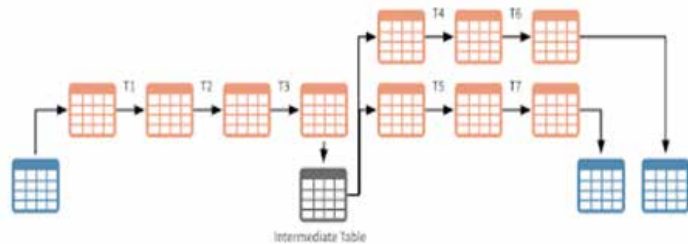
- Data engineers can develop data pipelines in a cost-efficient manner continuously without having to choose between batch and streaming
- Data analysts can get near real-time insights and faster answers to their BI queries
- Data scientists can develop better machine learning models using more reliable data sets with support for time travel that facilitates reproducible experiments and reports



Delta Architecture using the “multi-hop” approach for data pipelines

Writing data pipelines using the Delta Architecture follows the best practices of having a multi-layer “multi-hop” approach where we progressively add structure to data: “Bronze” tables or Ingestion tables are usually raw data sets in the native format (JSON, CSV or txt), “Silver” tables represent cleaned/transformed data sets ready for reporting or data science, and “Gold” tables are the final presentation layer.

For the pure streaming use cases, the option of materializing the DataFrames in intermediate Delta Lake tables is basically just a trade-off between latency/SLAs and cost (an example being real-time monitoring alerts vs. updates of the recommender system based on new content).

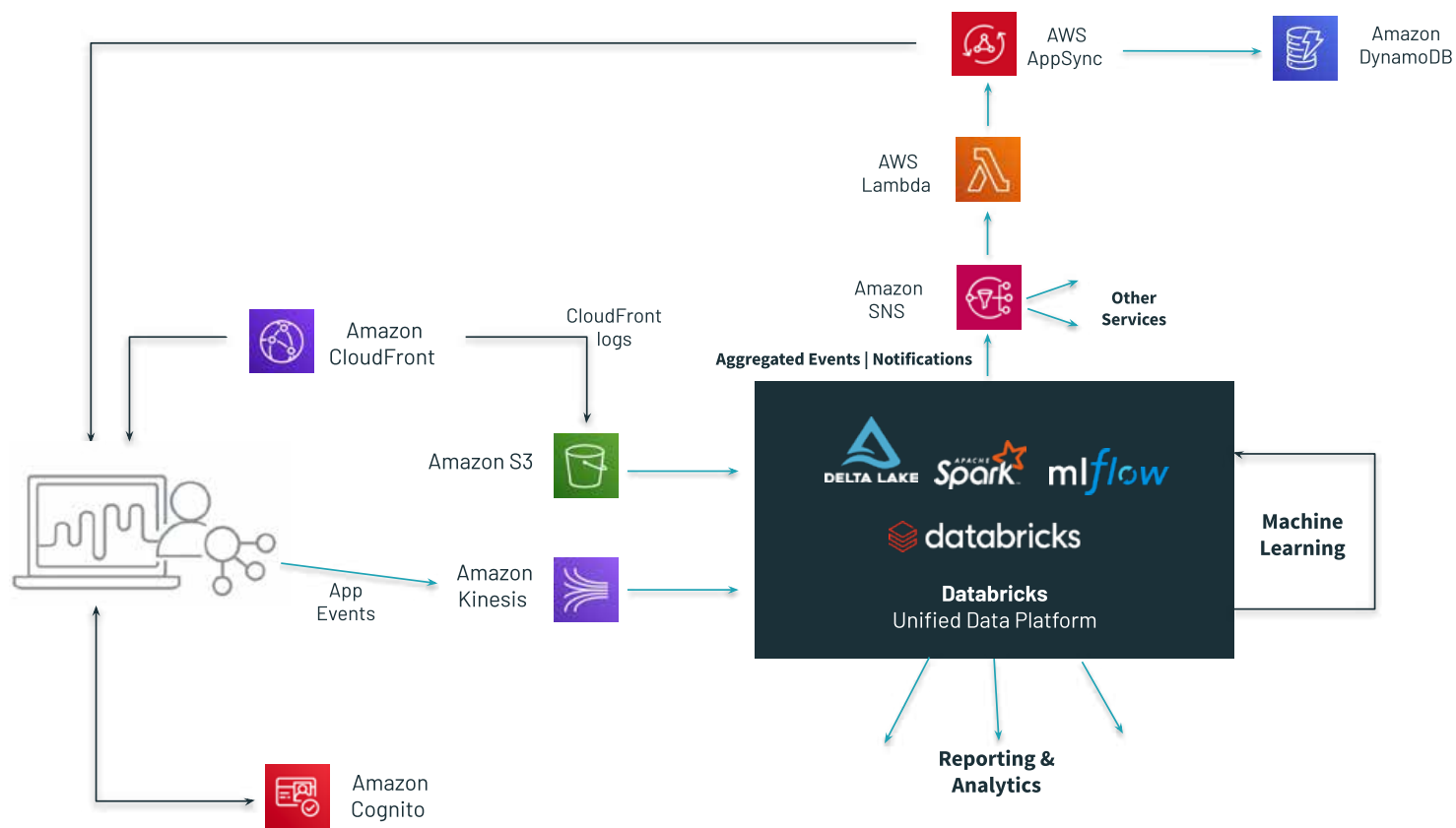


A streaming architecture can still be achieved while materializing DataFrames in Delta Lake tables

The number of “hops” in this approach is directly impacted by the number of consumers downstream, complexity of the aggregations (e.g., Structured Streaming enforces certain limitations around chaining multiple aggregations) and the maximization of operational efficiency.

The QoS solution architecture is focused around best practices for data processing and is not a full video-on-demand (VoD) solution – with some standard components like the “front door” service Amazon API Gateway being avoided from the high-level architecture in order to keep the focus on data and analytics.





High-level architecture for the QoS platform

Making your data ready for analytics

Both sources of data included in the QoS solution (application events and CDN logs) are using the JSON format, great for data exchange – allowing you to represent complex nested structures, but not scalable and difficult to maintain as a storage format for your data lake / analytics system.

In order to make the data directly queryable across the entire organization, the Bronze to Silver pipeline (the “make your data available to everyone” pipeline) should transform any raw formats into Delta Lake and include all the quality checks or data masking required by any regulatory agencies.



type	ts	jsonData
stream	2020-04-08T09:53:58.864+0000	{\"metricType\":\"stream\", \"at\":\"105.077308\", \"rtt\":350, \"connection_type\":\"4g\", \"package\":\"hls\", \"resolution\":\"640x360\", \"fps\":\"29.970\", \"avg_bitrate\":1330203, \"duration\":597, \"cdn_tracking_id\":\"pxiujwunmzflqxa8soetfjrjvzfnfsovfnh8epu0-6szy38sv1vg==\", \"user_id\":\"us-west-2:610f9e88-55e8-4a80-9b63-936446e6df2d\", \"video_id\":\"bigbuckbunny\", \"playlist_type\":\"live\", \"timestamp\":1586339638864}
stream	2020-04-08T09:54:00.396+0000	{\"metricType\":\"stream\", \"at\":\"415.111827\", \"rtt\":528, \"connection_type\":\"3g\", \"package\":\"hls\", \"resolution\":\"640x360\", \"fps\":\"29.970\", \"avg_bitrate\":1350743, \"duration\":735, \"cdn_tracking_id\":\"jahzaxapw4y-xrhjoescatcytahbjvduwfgcvawqiyhu4pagrnrya==\", \"user_id\":\"us-west-2:cc7af3ef-6cf9-4da1-8274-3212db46be48\", \"video_id\":\"tearsofsteel\", \"playlist_type\":\"live\", \"timestamp\":1586339640396}
stream	2020-04-08T09:54:01.332+0000	{\"metricType\":\"stream\", \"at\":\"410.095889\", \"rtt\":591, \"connection_type\":\"4g\", \"package\":\"hls\", \"resolution\":\"640x360\", \"fps\":\"29.970\", \"avg_bitrate\":1330203, \"duration\":597, \"cdn_tracking_id\":\"svynftjqsprpmuy-zewwi4u3pbdn2sfsgcaodm-2cgdpfihhy4zlhka==\", \"user_id\":\"us-west-2:83cb8ebe-4f29-4822-80a2-feb10095a9a2\", \"video_id\":\"bigbuckbunny\", \"playlist_type\":\"live\", \"timestamp\":1586339641332}
stream	2020-04-08T09:53:53.498+0000	{\"metricType\":\"stream\", \"at\":\"385.057842741\", \"rtt\":523, \"connection_type\":\"not available\", \"package\":\"hls\", \"resolution\":\"640x360\", \"fps\":\"29.970\", \"avg_bitrate\":1350743, \"duration\":735, \"cdn_tracking_id\":\"h3dplndfpy9acagudbie3kwck9gknh98fjdu7uqhikigwagc8ga==\", \"user_id\":\"us-west-2:2f7000b3-1e01-4cba-a0ef-d49b8952633c\", \"video_id\":\"tearsofsteel\", \"playlist_type\":\"live\", \"timestamp\":1586339633498}
buffer	2020-04-	{\"metricType\":\"buffer\", \"buffer_type\":\"firstbuffer\", \"at\":\"385.057842741\", \"rtt\":523, \"connection_type\":\"not available\", \"time_millisecond\":386983, \"cdn_tracking_id\":\"h3dplndfpy9acagudbie3kwck9gknh98fjdu7uqhikigwagc8ga==\", \"user_id\":\"us-west-2:2f7000b3-1e01-4cba-

Showing the first 1000 rows.

Raw format of the app events

Video applications events

Based on the architecture, the video application events are pushed directly to Kinesis Streams and then just ingested to a Delta Lake append-only table without any changes to the schema.

Using this pattern allows a high number of consumers downstream to process the data in a streaming paradigm without having to scale the throughput of the Kinesis stream. As a side effect of using a Delta Lake table as a sink (which supports [optimize!](#)), we don't have to worry about the way the size of the processing window will impact the number of files in your target table — known as the “small files” issue in the big data world.

Both the timestamp and the type of message are being extracted from the JSON event in order to be able to partition the data and allow consumers to choose the type of events they want to process. Again combining a single Kinesis stream for the events with a Delta Lake “Events” table reduces the operational complexity while making things easier for scaling during peak hours.

Schema:

col_name	data_type
browserfamily	string
bytes	string
cdn_source	string
isbot	boolean
origin	string
location	string
logdate	date
logtime	string
osfamily	string
requestid	string
ip	string
resulttype	string
year	int
month	int
day	int
hour	int

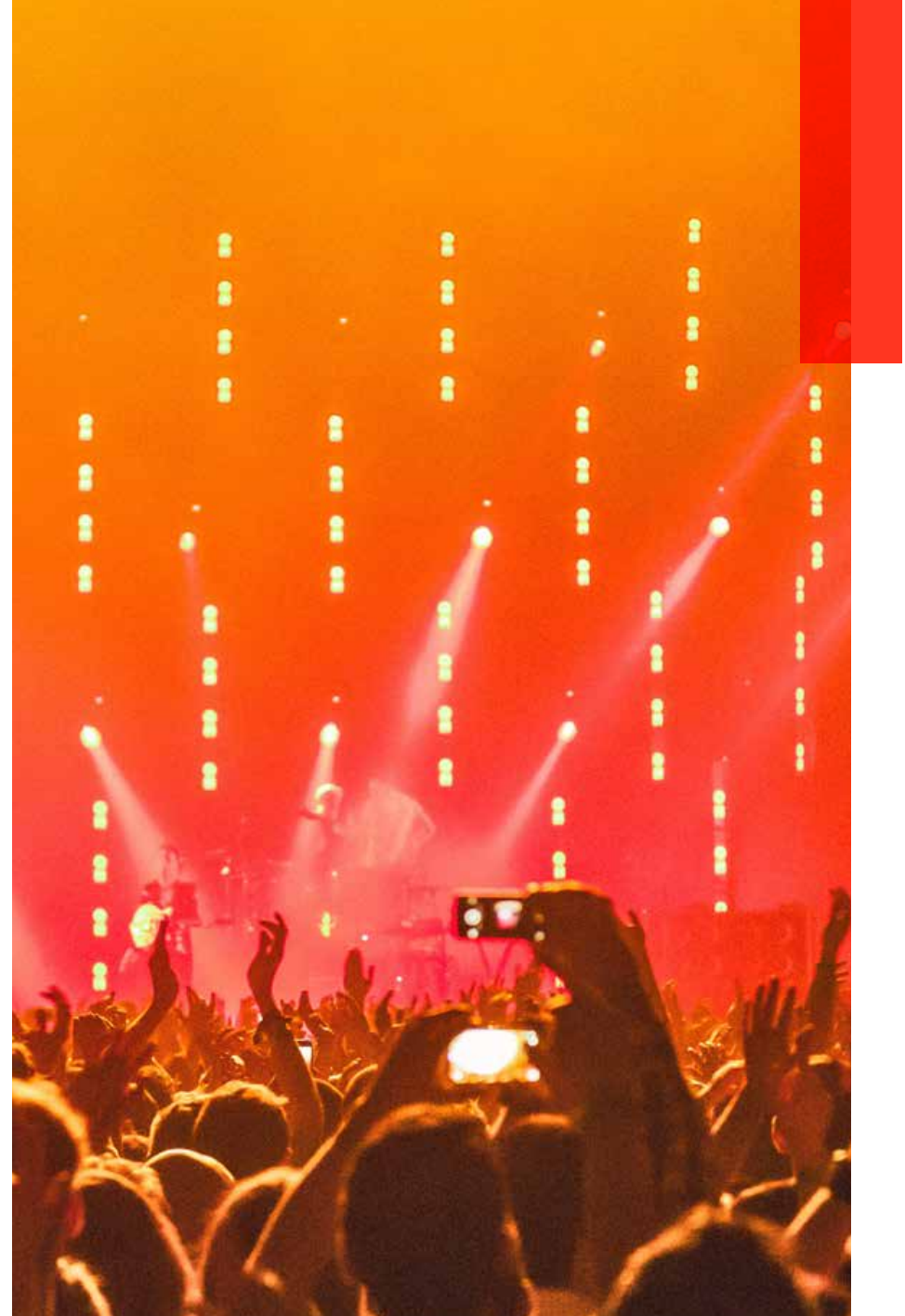
All the details are extracted from JSON for the Silver table

CDN logs

The CDN logs are delivered to S3, so the easiest way to process them is the Databricks Auto Loader, which incrementally and efficiently processes new data files as they arrive in S3 without any additional setup.

```
auto_loader_df = spark.readStream.format("cloudFiles") \  
    .option("cloudFiles.format", "json") \  
    .option("cloudFiles.region", region) \  
    .load(input_location)  
  
anonymized_df = auto_loader_df.select('*', ip_\  
anonymizer('requestip').alias('ip'))\  
    .drop('requestip')\  
    .withColumn("origin", map_ip_to_location(col('ip')))  
  
anonymized_df.writeStream \  
    .option('checkpointLocation', checkpoint_location) \  
    .format('delta') \  
    .table(silver_database + '.cdn_logs')
```

As the logs contain IPs – considered personal data under the GDPR regulations – the “make your data available to everyone” pipeline has to include an anonymization step. Different techniques can be used, but we decided to just strip the last octet from IPv4 and the last 80 bits from IPv6. On top, the data set is also enriched with information around the origin country and the ISP provider, which will be used later in the Network Operation Centers for localization.





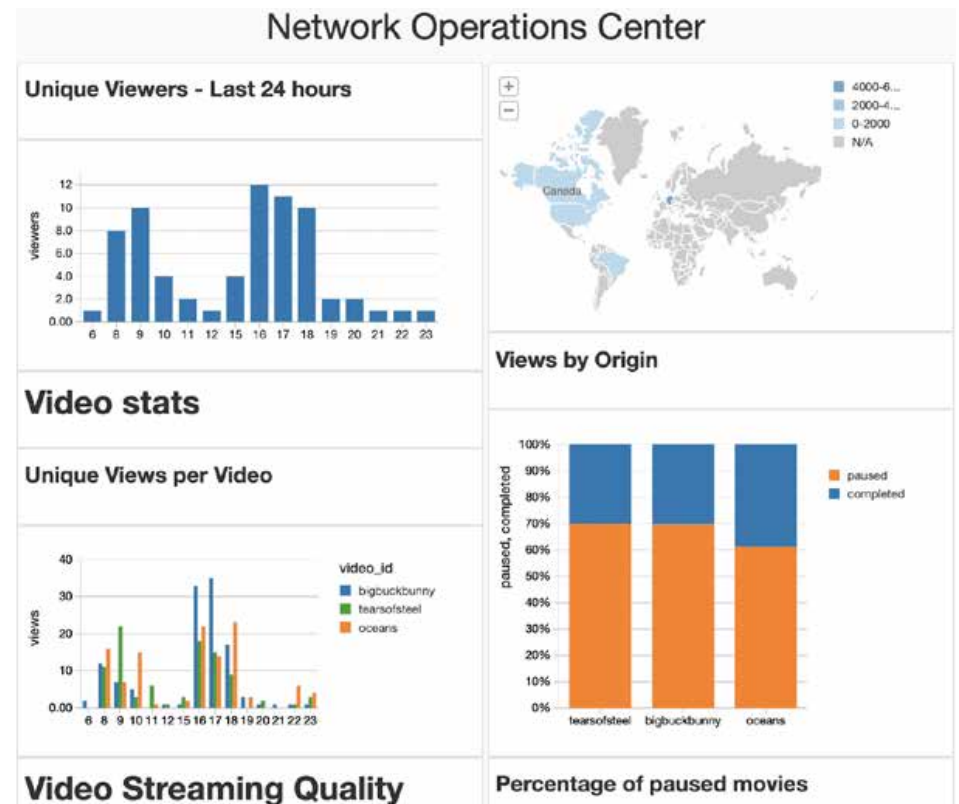
Creating the Dashboard / Virtual Network Operation Centers

Streaming companies need to monitor network performance and the user experience as near real-time as possible, tracking down to the individual level with the ability to abstract at the segment level, easily defining new segments such as those defined by geos, devices, networks and/or current and historical viewing behavior.

For streaming companies that has meant adopting the concept of Network Operation Centers (NOC) from telco networks for monitoring the health of the streaming experience for their users at a macro level, flagging and responding to any issues early on. At their most basic, NOCs should have dashboards that compare the current experience for users against a performance baseline so that the product teams can quickly and easily identify and attend to any service anomalies.

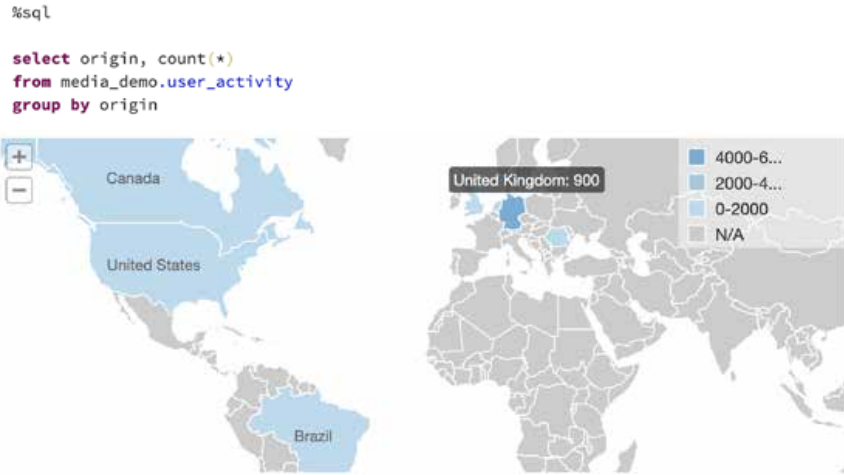
In the QoS solution we have incorporated a [Databricks dashboard](#). BI tools can also be effortlessly connected in order to build more complex visualizations, but based on customer feedback, built-in dashboards are, most of the time, the fastest way to present the insights to business users.

The aggregated tables for the NOC will basically be the Gold layer of our Delta Architecture – a combination of CDN logs and the application events.



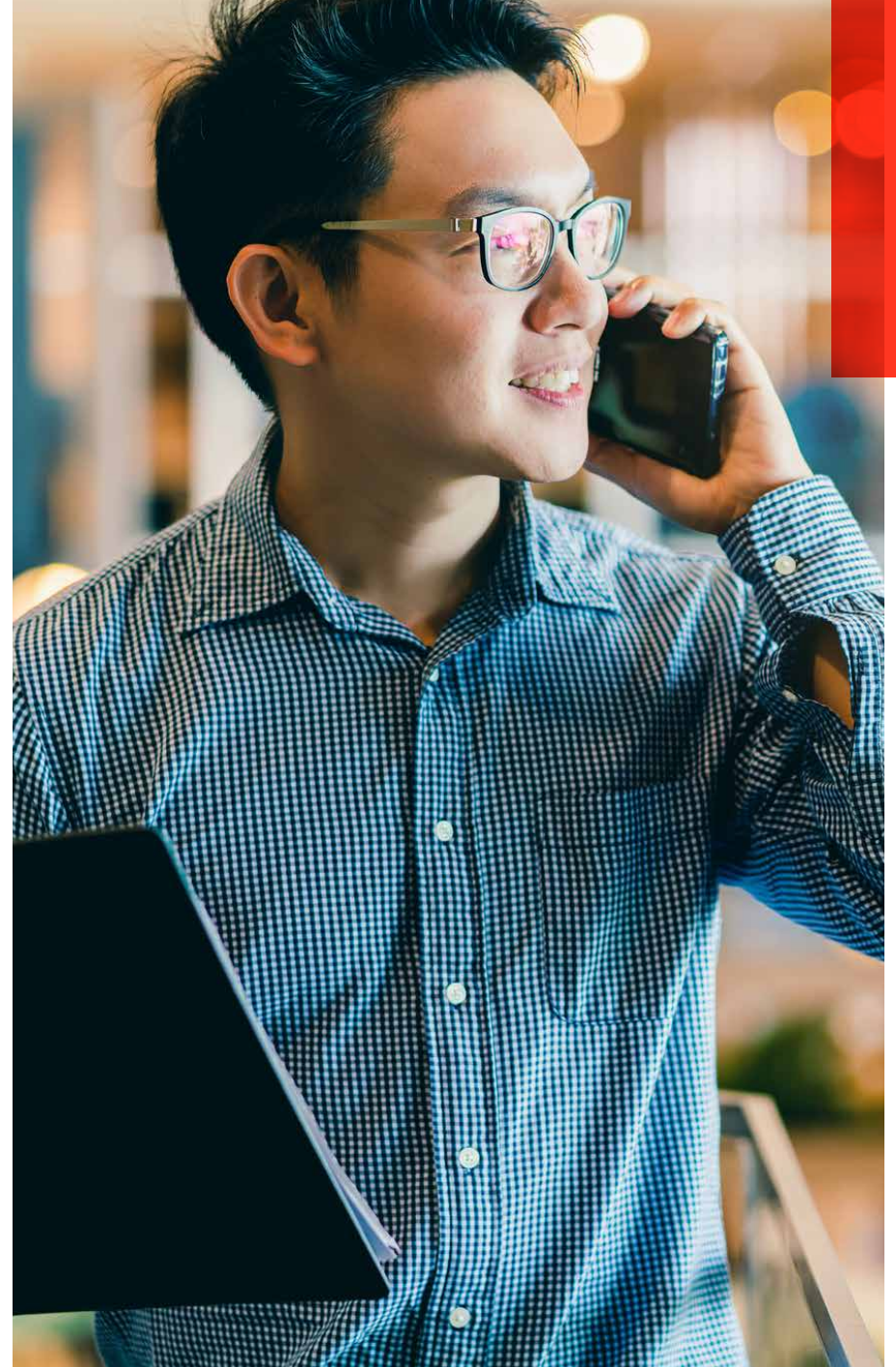
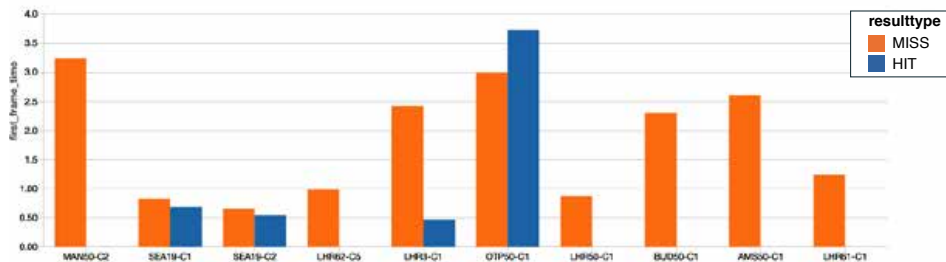
Example of Network Operations Center dashboard

The dashboard is just a way to visually package the results of SQL queries or Python / R transformation – each notebook supports multiple dashboards so in case of multiple end users with different requirements we don't have to duplicate the code – as a bonus the refresh can also be scheduled as a Databricks job.

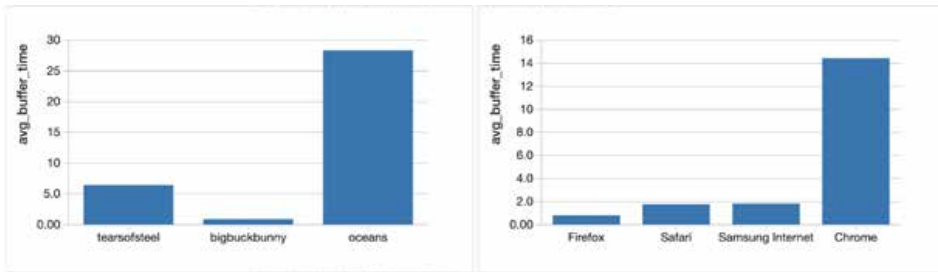


Visualization of the results of a SQL query

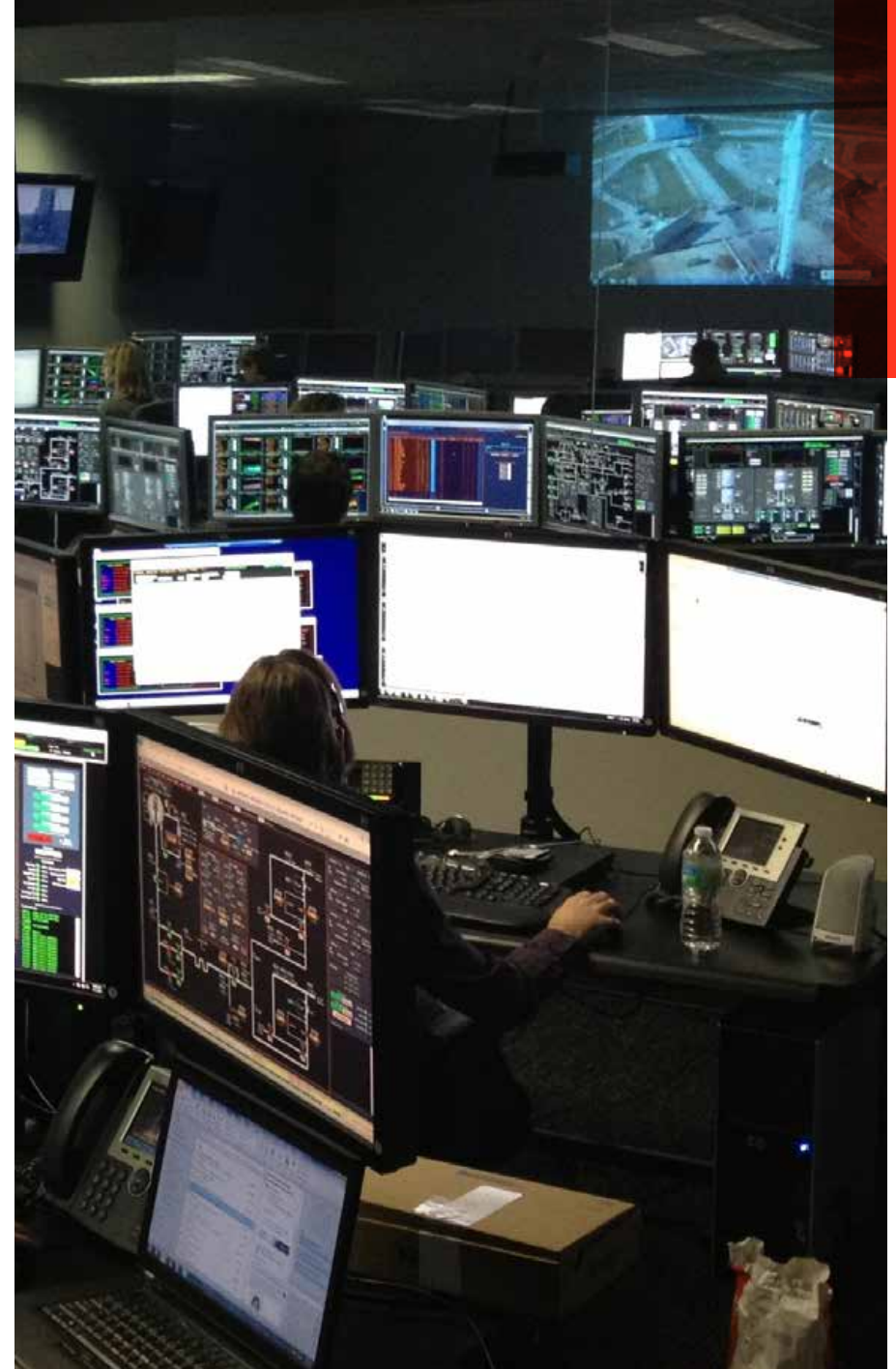
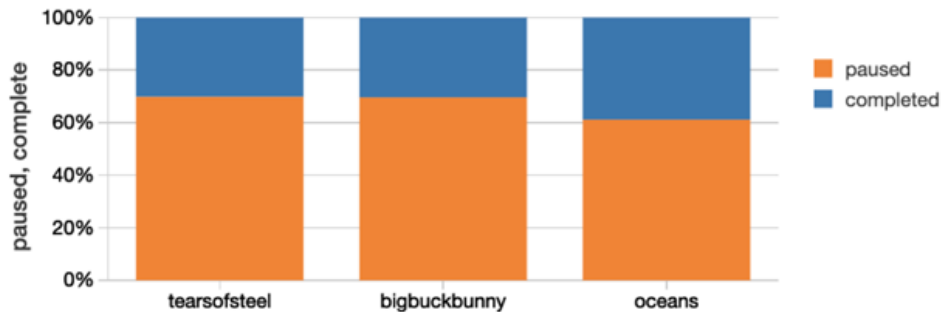
Loading time for videos (time to first frame) allows better understanding of the performance for individual locations of your CDN – in this case the AWS CloudFront Edge nodes – which has a direct impact in your strategy for improving this KPI – either by spreading the user traffic over multi-CDNs or maybe just implementing a



Failure to understand the reasons for high levels of buffering — and the poor video quality experience that it brings — has a significant impact on subscriber churn rate. On top of that, advertisers are not willing to spend money on ads responsible for reducing the viewer engagement — as they add extra buffering on top, so the profits on the advertising business usually are impacted too. In this context, collecting as much information as possible from the application side is crucial to allow the analysis to be done not only at video level but also browser or even type / version of application.



On the content side, events for the application can provide useful information about user behavior and overall quality of experience. How many people that paused a video have actually finished watching that episode / video? What caused the stoppage: The quality of the content or delivery issues? Of course, further analyses can be done by linking all the sources together (user behavior, performance of CDNs / ISPs) to not only create a user profile but also to forecast churn.

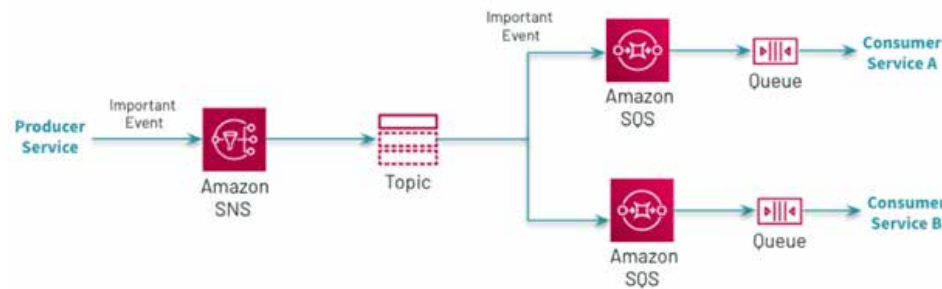


Creating (near) real-time alerts

When dealing with the velocity, volume and variety of data generated in video streaming from millions of concurrent users, dashboard complexity can make it harder for human operators in the NOC to focus on the most important data at the moment and zero-in on root cause issues. With this solution, you can easily set up automated alerts when performance crosses certain thresholds that can help the human operators of the network as well as set off automatic remediation protocols via a Lambda function. For example:

- If a CDN is having latency much higher than baseline (e.g., if it's more than 10% latency vs. baseline average), initiate automatic CDN traffic shifts.
- If more than [some threshold, e.g., 5%] of clients report playback errors, alert the product team that there is likely a client issue for a specific device.
- If viewers on a certain ISP are having higher-than-average buffering and pixelation issues, alert frontline customer representatives on responses and ways to decrease issues (e.g., set stream quality lower).

From a technical perspective, generating real-time alerts requires a streaming engine capable of processing data real time and publish-subscribe service to push notifications.



Integrating microservices using Amazon SNS and Amazon SQS

The QoS solution implements the [AWS best practices for integrating microservices](#) by using Amazon SNS and its integrations with Amazon Lambda (see below for the

updates of web applications) or Amazon SQS for other consumers. The [custom for each writer](#) option makes the writing of a pipeline to send email notifications based on a rule-based engine (e.g., validating the percentage of errors for each individual type of app over a period of time) really straightforward.

```
def send_error_notification(row):

    sns_client = boto3.client('sns', region)

    error_message = 'Number of errors for the App has exceeded the
threshold {}'.format(row['percentage'])

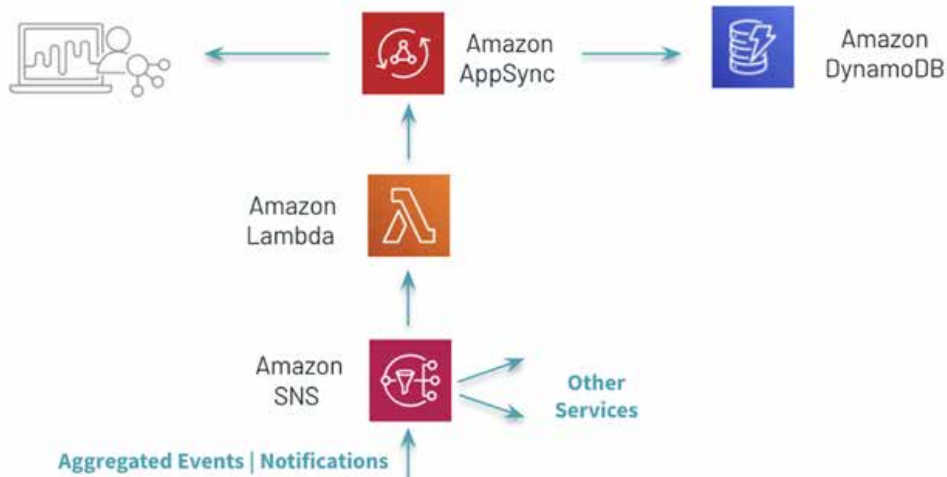
    response = sns_client.publish(
        TopicArn=,
        Message= error_message,
        Subject=,
        MessageStructure='string')

# Structured Streaming Job

getKinesisStream("player_events")\
    .selectExpr("type", "app_type")\
    .groupBy("app_type")\
    .apply(calculate_error_percentage)\
    .where("percentage > {}".format(threshold)) \
    .writeStream\
    .foreach(send_error_notification)\
    .start()
```

Sending email notifications using AWS SNS

On top of the basic email use case, the Demo Player includes three widgets updated in real time using AWS AppSync: the number of active users, the most popular videos and the number of users concurrently watching a video.



Updating the application with the results of real-time aggregations

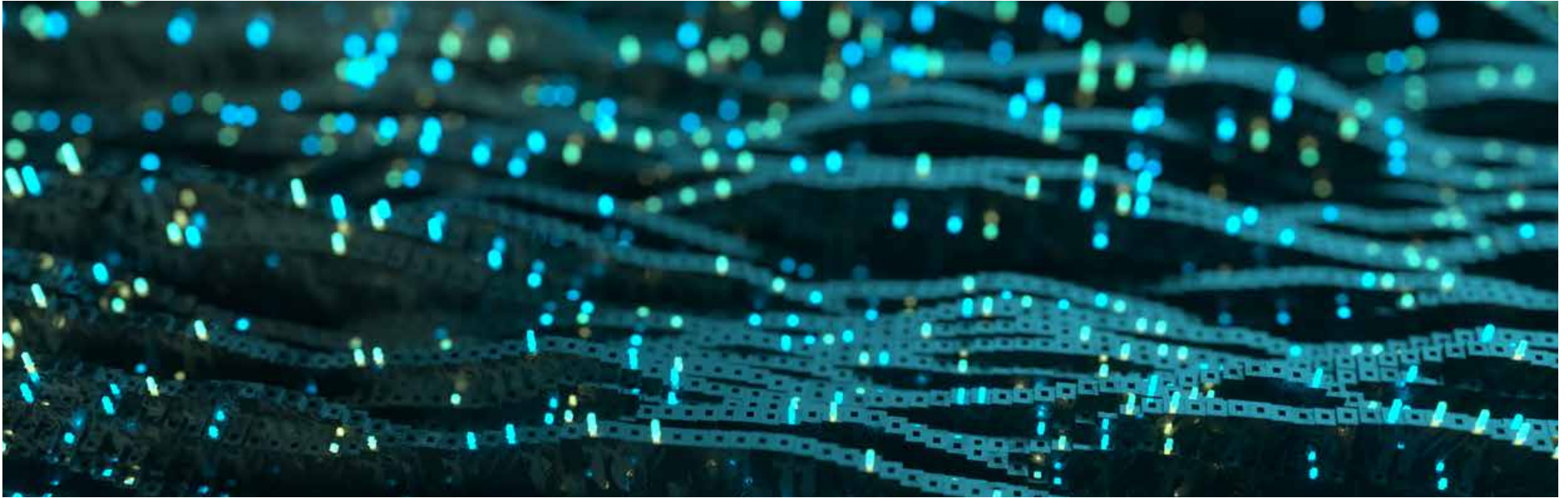
The QoS solution is applying a similar approach — Structured Streaming and Amazon SNS — to update all the values allowing for extra consumers to be plugged in using AWS SQS. This is a common pattern when huge volumes of events have to be enhanced and analyzed; pre-aggregate data once and allow each service (consumer) to make their own decision downstream.

Next steps: machine learning

Manually making sense of the historical data is important but is also very slow. If we want to be able to make automated decisions in the future, we have to integrate machine learning algorithms.

As a Unified Data Platform, Databricks empowers data scientists to build better data science products using features like Runtime for Machine Learning with built-in support for [Hyperopt](#) / [Horvod](#) / [AutoML](#) or the integration with MLflow, the end-to-end machine learning lifecycle management tool.





We have already explored a few important use cases across our customer base while focusing on the possible extensions to the QoS solution.

Point-of-failure prediction and remediation

As D2C streamers reach more users, the costs of even momentary loss of service increases. ML can help operators move from reporting to prevention by forecasting where issues could come up and remediating before anything goes wrong (e.g., a spike in concurrent viewers leads to switching CDNs to one with more capacity automatically).

Customer churn

Critical to growing subscription services is keeping the subscribers you have. By understanding the quality of service at the individual level, you can add QoS as a variable in churn and customer lifetime value models. Additionally, you can create customer cohorts for those who have had video quality issues in order to test proactive messaging and save offers.

Getting started with the Databricks streaming video QoS solution

Providing consistent quality in the streaming video experience is table stakes at this point to keep fickle audiences with ample entertainment options on your platform. With this solution we have sought to create a quick start for most streaming video platform environments to embed this QoS real-time streaming analytics solution in a way that:

- Scales to any audience size
- Quickly flags quality performance issues at key parts of the distribution workflow
- Is flexible and modular enough to easily customize for your audience and your needs, such as creating new automated alerts or enabling data scientists to test and roll out predictive analytics and machine learning

To get started, download the notebooks for the [Databricks streaming video QoS solution](#). For more guidance on how to unify batch and streaming data into a single system, view the [Delta Architecture webinar](#).📺

What's next?

Now that you understand Delta Lake and how its features can improve performance, it may be time to take a look at some additional resources.

Explore subsequent eBooks in the collection >

- [The Delta Lake Series – Fundamentals and Performance](#)
- [The Delta Lake Series – Features](#)
- [The Delta Lake Series – Lakehouse](#)
- [The Delta Lake Series – Customer Use Cases](#)

Do a deep dive into Delta Lake >

- [Getting Started With Delta Lake Tech Talk Series](#)
- [Diving Into Delta Lake Tech Talk Series](#)
- [Visit the site for additional resources](#)

Try Databricks for free >

Learn more >