Microsoft Cloud Workshop

Cosmos DB Real Time Advanced Analytics
Whiteboard design session trainer guide
November 2019

Contents

Trainer information

Thank you for taking time to support the whiteboard design sessions as a trainer!

Role of the trainer

An amazing trainer:

- Creates a safe environment in which learning can take place.
- Stimulates the participant's thinking.
- Involves the participant in the learning process.
- Manages the learning process (on time, on topic, and adjusting to benefit participants).
- Ensures individual participant accountability.
- Ties it all together for the participant.
- Provides insight and experience to the learning process.
- Effectively leads the whiteboard design session discussion.
- Monitors quality and appropriateness of participant deliverables.

- Effectively leads the feedback process.

Whiteboard design session flow

Each whiteboard design session uses the following flow:

Step 1: Review the customer case study (15 minutes)

Outcome

Analyze your customer's needs.

- Customer's background, situation, needs and technical requirements
- Current customer infrastructure and architecture
- Potential issues, objectives and blockers

Step 2: Design a proof of concept solution (60 minutes)

Outcome

Design a solution and prepare to present the solution to the target customer audience in a 15-minute chalk-talk format.

- Determine your target customer audience.
- Determine customer's business needs to address your solution.
- Design and diagram your solution.
- Prepare to present your solution.

Step 3: Present the solution (30 minutes)

Outcome

Present solution to your customer:

- Present solution
- Respond to customer objections
- Receive feedback

Wrap-up (15 minutes)

- Review preferred solution

Before the whiteboard design session: How to prepare

Before conducting your first whiteboard design session:

- Read the Student guide (including the case study) and Trainer guide.

- Become familiar with all key points and activities.
- Plan the point you want to stress, which questions you want to drive, transitions, and be ready to answer questions.
- Prior to the whiteboard design session, discuss the case study to pick up more ideas.
- Make notes for later.

During the whiteboard design session: Tips for an effective whiteboard design session

Refer to the Trainer guide to stay on track and observe the timings.

Do not expect to memorize every detail of the whiteboard design session.

When participants are doing activities, you can **look ahead to refresh your memory**.

- **Adjust activity and whiteboard design session pace** as needed to allow time for presenting, feedback, and sharing.
- **Add examples, points, and stories** from your own experience. Think about stories you can share that help you make your points clearly and effectively.
- **Consider creating a "parking lot"** to record issues or questions raised that are outside the scope of the whiteboard design session or can be answered later. Decide how you will address these issues, so you can acknowledge them without being derailed by them.

Have fun! Encourage participants to have fun and share!

Involve your participants. Talk and share your knowledge but always involve your participants, even while you are the one speaking.

Ask questions and get them to share to fully involve your group in the learning process.

Ask first, whenever possible. Before launching into a topic, learn your audience's opinions about it and experiences with it. Asking first enables you to assess their level of knowledge and experience, and leaves them more open to what you are presenting.

Wait for responses. If you ask a question such as, "What's your experience with (fill in the blank)?" then wait. Do not be afraid of a little silence. If you leap into the silence, your participants will feel you are not serious about involving them and will become passive. Give participants a chance to think, and if no one answers, patiently ask again. You will usually get a response.

Cosmos DB real-time advanced analytics whiteboard design session student guide

Abstract and learning objectives

Woodgrove Bank, who provides payment processing services for commerce, is looking to design and implement a PoC of an innovative fraud detection solution. They want to provide new services to their merchant customers, helping them save costs by applying machine learning and advanced analytics to detect fraudulent transactions. Their customers are around the world, and the right solutions for them would minimize any latencies experienced using their service by distributing as much of the solution as possible, as closely as possible, to the regions in which their customers use the service.

In this whiteboard design session, you will work in a group to design the data pipeline PoC that could support the needs of Woodgrove Bank.

At the end of this workshop, you will be better able to design solutions that leverage the strengths of Cosmos DB in support of advanced analytics solutions that require high throughput ingest, low latency serving and global scale in combination with scalable machine learning, big data and real-time processing capabilities.

Step 1: Review the customer case study

Outcome

Analyze your customer's needs.

Timeframe: 15 minutes

Directions: With all participants in the session, the facilitator/SME presents an overview of the customer case study along with technical tips.

1. Meet your table participants and trainer.
2. Read all of the directions for steps 1-3 in the student guide.
3. As a table team, review the following customer case study.

Customer situation

Woodgrove Bank, who provides payment processing services for commerce, is looking to design and implement a PoC of an innovative fraud detection solution. They know from experience and through contacts in the financial industry that there is a constant arms race between fraudsters and banks. Thanks to increasingly powerful and easily accessible technology, financial crime is on the rise. Payment processing companies, like Woodgrove Bank, and their merchant customers risk financial losses due to fraud.

They also risk fines from failing to detect or even prevent criminal acts like money laundering or terror financing. Woodgrove forecasts reaching over USD $10 Billion in assets over the upcoming fiscal year, placing them within the stricter regulatory purview of institutions classified by the US government as "big banks". This means that they will be subject to regulatory fines over and above the fraud loss, putting their business at greater risk.

While all forms of fraud are on the rise, like ATM fraud, card transaction fraud, payment fraud, Woodgrove Bank would like to focus on online fraud. In the most basic terms, online fraud is committed when an unauthorized user impersonates another user by taking over their account, using malware, or hijacking internet sessions and uses the impersonated credentials to make purchase transactions. When dealing with millions of transactions, it is both crucial and challenging to detect and monitor fraud in real-time across all transactions. Doing so helps prevent additional losses and detect widespread attacks.

Given this focus on online fraud, they want to provide new services to their merchant customers, helping them save costs by applying machine learning and advanced analytics to detect fraudulent transactions. Their customers are around the world, and the right solutions for them would minimize any latencies experienced using their service by distributing as much of the solution as possible, as closely as possible, to the regions in which their customers use the service. This is the solution for which they would like to implement a PoC.

In flagging fraudulent transactions, they know there are tradeoffs between being overly aggressive and mistakenly identifying innocuous transactions as fraudulent, and not being aggressive enough such that they miss transactions that represent real fraud. According to Mari Stephens, Chief Information Officer (CIO), Woodgrove Bank, they would rather miss a fraudulent event in their automated system, than mistakenly identify innocuous transactions as fraudulent because the latter will frustrate both their merchant customer and the end customers and potentially lose their business. However, they want to balance this by doing as much as they can to detect fraud while minimizing the customer frustration. To address this, they believe the PoC will need to handle transactions at two "speeds". First, they want to screen transactions for fraud as they happen, only blocking a transaction if the system is very confident it is fraudulent. Second, they want to perform a more in-depth, offline fraud sweep of transactions to identify suspicious transactions. These are transactions which are potentially fraud, for which they will notify the merchant that they should perform additional verification with the end customer before completing the order. This deeper analysis that is performed in batch may use a slightly different ML model, but since it is a more intensive run, it needs to be run in batch and score transactions a little less leniently than the real-time scoring model. Remember, Woodgrove wants to minimize false positives during real-time scoring, but do a deeper analysis of the transactions later on and possibly tag those that were not blocked as suspicious.

They have decades worth of historical transaction data (including transactions identified as fraudulent) that they believe would be helpful in the fraud detection PoC. This data is in tabular format and can be exported to CSV files if needed.

The analysts at Woodgrove Bank are very interested in the recent notebook-driven approach to performing data science and data engineering tasks, and would prefer a solution that features notebooks as the standard way to explore data, prepare data, model, and define the logic for scheduled processing.

Woodgrove's current process

Woodgrove Bank provides a RESTful API that their merchant customers use to submit payments. The POC you design should not interrupt this process in any way. The solution you design needs to run side-by-side and augment their current process without changing their current workflow. Currently, as payments flow through their API endpoints, a series of cardholder verification steps are executed, such as matching the cardholder's billing address to their account. Once this validation check has completed, Woodgrove returns an authorization ID to the merchant, along with a status (accepted, rejected, declined, etc.). The payment details are entered into a relational database and the back-end payment process continues. There may be an opportunity to modify this process down the road, but that is not the focus of the POC.

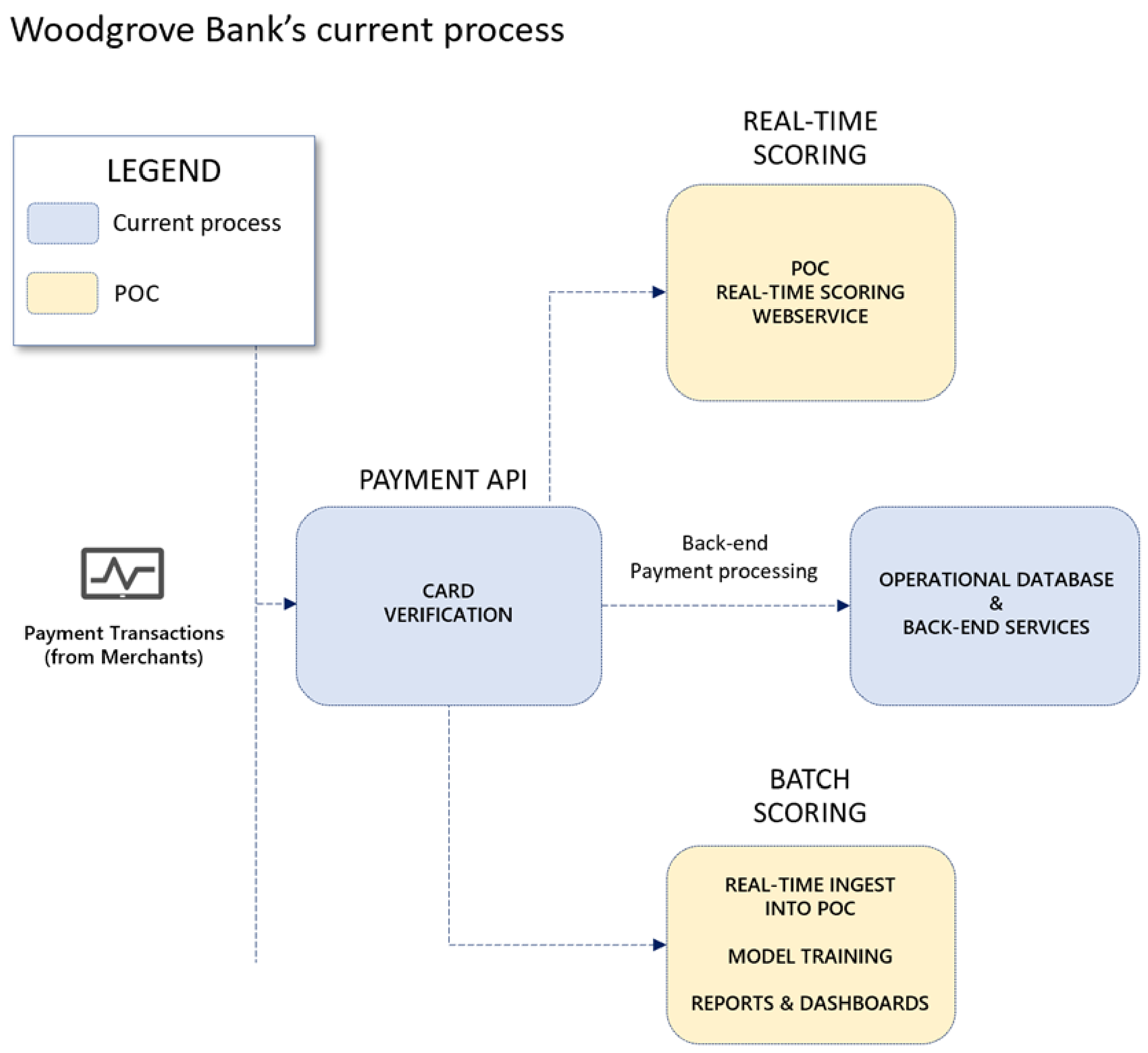
The customer is asking for 2 additions to their current process:

- A RESTful API that can be called for immediate scoring on a transaction to see whether it should be blocked due to reasonably high-level confidence that it is fraudulent. Remember, this step should have a low number of false positives. The batch process that conducts a deeper sweep should flag suspicious transactions that were not blocked by this initial check.

- A real-time data ingestion pipeline they can pass data to at the time they save the payment transaction data from within their API. This should sit side-by-side with their current process, not change it.

To clarify, the requirement for real-time scoring of the payment transaction as fraudulent is not the same as the real-time ingest of all payment transaction data.

Below is a simple diagram Woodgrove Bank provided of their current process (blue boxes), showing where they would like you to fit in the new POC components (yellow boxes).


Woodgrove Bank's current process
LEGEND
Current process
POC
REAL-TIME SCORING
POC REAL-TIME SCORING WEBSERVICE
PAYMENT API
CARD VERIFICATION
Back-end Payment processing
OPERATIONAL DATABASE & BACK-END SERVICES
Payment Transactions (from Merchants)
BATCH SCORING
REAL-TIME INGEST INTO POC
MODEL TRAINING
REPORTS & DASHBOARDS

Customer needs

1. Need to provide fraud detection services to our merchant customers, using incoming payment transaction data to provide early warning of fraudulent activity.

2. We would like to schedule offline scoring of “suspicious activity” using our trained model, and make that data globally available in regions closest to our customers through our web applications.

3. We want the ability to analyze all transactions over time, so we need to be able to store data from transaction sources into long-term storage, without interfering with jobs reading the data set.

4. We would like to use a standard platform that supports our near-term data pipeline needs while providing a long-term standard for data science, data engineering, and development.

Customer objections

1. It's not clear to us if we can only use Cosmos DB as our web app's database, or if we should consider using it in other parts of our advanced analytics data pipeline such as for real-time transaction ingest or for serving of offline processed data.

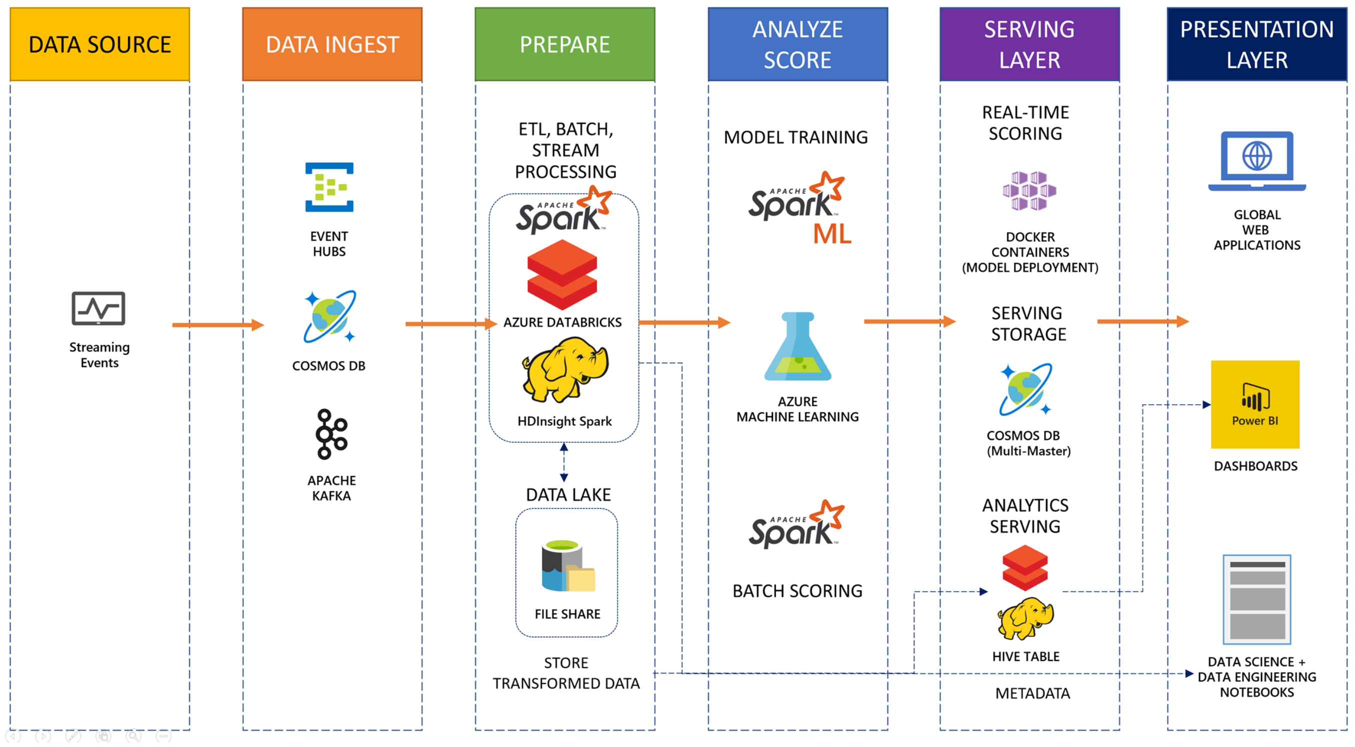
2. Does Cosmos DB integrate with open source big data analytics like Apache Spark?

3. Properly selecting the right algorithm and training a model using the optimal set of parameters can take a lot of time. Is there a way to speed up this process?

4. We are concerned about how much it costs to use Cosmos DB for our solution. What is the real value of the service, and how do we set up Cosmos DB in an optimal way?

5. How do we optimize our indexes for both write-heavy and read-heavy workloads?

Infographic for common scenarios



Step 2: Design a proof of concept solution

Outcome

Design a solution and prepare to present the solution to the target customer audience in a 15-minute chalk-talk format.

Timeframe: 60 minutes

Business needs

Directions: With all participants at your table, answer the following questions and list the answers on a flip chart:

1. Who should you present this solution to? Who is your target customer audience? Who are the decision makers?

2. What customer business needs do you need to address with your solution?

Design

Directions: With all participants at your table, respond to the following questions on a flip chart:

High-level architecture

1. Without getting into the details (the following sections will address the particular details), diagram your initial vision for handling the top-level requirements for payment fraud detection, including stream capture and processing, long-term storage, model training, global distribution of the model for real-time scoring and of the pre-scored fraud data, and dashboards.

Globally distributed data

1. Which data storage service would you recommend for storing the suspicious transactions? Remember, Woodgrove Bank wants to minimize access latency for their global customers. Be specific about how data is replicated.

2. How does your chosen service handle scaling to meet varying levels of demand across different regions? Can you set specific capacity for specific regions?

3. Distributed databases that replicate data to multiple locations have some potential delay between when you write a record and when that record is available for reading. What options does your chosen service have to ensure the data is not "stale" when read? Are there any tradeoffs between reducing the window between writes, and if so, how do they apply to Woodgrove Bank's situation?

Data ingest

1. What are your recommended options for ingesting payment transaction events as they occur in a scalable way that can be easily processed while maintaining event order with no data loss?

2. Of the ingest options you identified previously, which would you recommend for the scenario?

Data pipeline processing

1. Woodgrove Bank indicated that they would like a unified way to process both streaming data and batch data on a platform that can also support their data science, data engineering, and development needs. Which platform would you recommend, and why?

2. The big data systems Woodgrove Bank used in the past were only able to append new data to the end of existing data sets. This meant each time they had update, they would actually create a duplicate row containing the changed data and then have to author queries to merge those rows so that they had a clean view of the current state of the data. How will your chosen platform cope with this challenge?

3. How will your chosen data processing platform connect to and process data from your chosen data ingest solution for streaming data?

4. What configuration would you need to apply to your solution to allow it to restart any stream processing in the case the job is stopped?

5. What specific secrets might their processing solution want to store? How would they securely store and access those secrets?

Long-term data storage

1. As incoming data is processed, refined, and scored, all of the transactions need to be persisted to long-term storage for analysis, model training and validation, and reporting. This storage needs to handle long-term growth, be fast enough to rapidly ingest new data while simultaneously handling reads against the same data set without interference, and act as a reliable data source for dashboards and reports. Which is your recommended long-term data storage solution, keeping in mind its role within your selected data pipeline processing platform?

2. How do you ensure your data is continuously optimized within your chosen long-term data storage solution, given the requirements to store inserts, updates, and deletes while avoiding generating very small, un-optimized files?

3. Woodgrove Bank wants to retain all raw data (bronze layer), then parse that data into query tables (silver layer) which can be joined with dimension tables, such as account information. They also would like to have summary tables (gold layer) containing business-level aggregates used for their dashboards and reports. How would you support these requirements in your long-term storage solution?

Model training and deployment

1. Describe how your chosen data processing platform will support machine learning model training and deployment. The model will need to be trained on and validated against historical payment transaction data that includes known fraudulent transactions.

2. How will you schedule regular batch scoring of fraud data using the trained model, and make that data available to Woodgrove Bank's web applications at a global scale?

Dashboards and reporting

1. Woodgrove Bank's business analysts would like to have a set of dashboards they can monitor that provide real-time views of fraud trends at a global scale. Thinking back to how your proposed solution provides a set of summary (gold) tables containing business-level aggregates, what do you propose using to meet this requirement? Be specific about how this solution will be put in place and which features it supports.

2. How do you propose giving access to this same data to Woodgrove Bank's data scientists and data engineers within the data processing environment wherein they can craft complex queries and data visualizations?

Prepare

Directions: With all participants at your table:

1. Identify any customer needs that are not addressed with the proposed solution.
2. Identify the benefits of your solution.
3. Determine how you will respond to the customer's objections.

Prepare a 15-minute chalk-talk style presentation to the customer.

Step 3: Present the solution

Outcome

Present a solution to the target customer audience in a 15-minute chalk-talk format.

Timeframe: 30 minutes

Presentation

Directions:

1. Pair with another table.
2. One table is the Microsoft team and the other table is the customer.
3. The Microsoft team presents their proposed solution to the customer.
4. The customer makes one of the objections from the list of objections.
5. The Microsoft team responds to the objection.
6. The customer team gives feedback to the Microsoft team.
7. Tables switch roles and repeat Steps 2-6.

Wrap-up

Timeframe: 15 minutes

Directions: Tables reconvene with the larger group to hear the facilitator/SME share the preferred solution for the case study.

Additional references

Description	**Links**
Introduction to Cosmos DB	https://docs.microsoft.com/azure/cosmos-db/introduction
About Event Hubs	https://docs.microsoft.com/azure/event-hubs/event-hubs-about
What is Azure Databricks?	https://docs.microsoft.com/azure/azure-databricks/what-is-azure-databricks
Azure Databricks Delta Lake	https://docs.azuredatabricks.net/delta/index.html

Description	Links
Using Azure Databricks as a Power BI data source	https://docs.azuredatabricks.net/user-guide/bi/power-bi.html
Introduction to Azure Data Lake Storage	https://docs.microsoft.com/azure/storage/blobs/data-lake-storage-introduction
What is Azure Machine Learning service?	https://docs.microsoft.com/azure/machine-learning/service/overview-what-is-azure-ml
Access ADLS with Azure Databricks using Spark	https://docs.microsoft.com/azure/storage/blobs/data-lake-storage-use-databricks-spark?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json
What is Azure Key Vault?	https://docs.microsoft.com/azure/key-vault/key-vault-overview
Scaling throughput in Azure Cosmos DB	https://docs.microsoft.com/azure/cosmos-db/scaling-throughput
Partitioning and horizontal scaling in Azure Cosmos DB	https://docs.microsoft.com/azure/cosmos-db/partition-data
Consistency levels in Azure Cosmos DB	https://docs.microsoft.com/azure/cosmos-db/consistency-levels
Apache Spark Connector to Cosmos DB	https://docs.microsoft.com/en-us/azure/cosmos-db/spark-connector
Azure Machine Learning SDK for Python	https://docs.microsoft.com/python/api/overview/azure/ml/intro?view=azure-ml-py

Cosmos DB real-time advanced analytics whiteboard design session trainer guide

Step 1: Review the customer case study

- Check in with your table participants to introduce yourself as the trainer.
- Ask, "What questions do you have about the customer case study?"
- Briefly review the steps and timeframes of the whiteboard design session.
- Ready, set, go! Let the table participants begin.

Step 2: Design a proof of concept solution

- Check in with your tables to ensure that they are transitioning from step to step on time.
- Provide some feedback on their responses to the business needs and design.
    - Try asking questions first that will lead the participants to discover the answers on their own.

- Provide feedback for their responses to the customer's objections.
    - Try asking questions first that will lead the participants to discover the answers on their own.

Step 3: Present the solution

- Determine which table will be paired with your table before Step 3 begins.
- For the first round, assign one table as the presenting team and the other table as the customer.
- Have the presenting team present their solution to the customer team.
    - Have the customer team provide one objection for the presenting team to respond to.
    - The presentation, objections, and feedback should take no longer than 15 minutes.
    - If needed, the trainer may also provide feedback.

Wrap-up

- Have the table participants reconvene with the larger session group to hear the facilitator/SME share the following preferred solution.

Preferred target audience

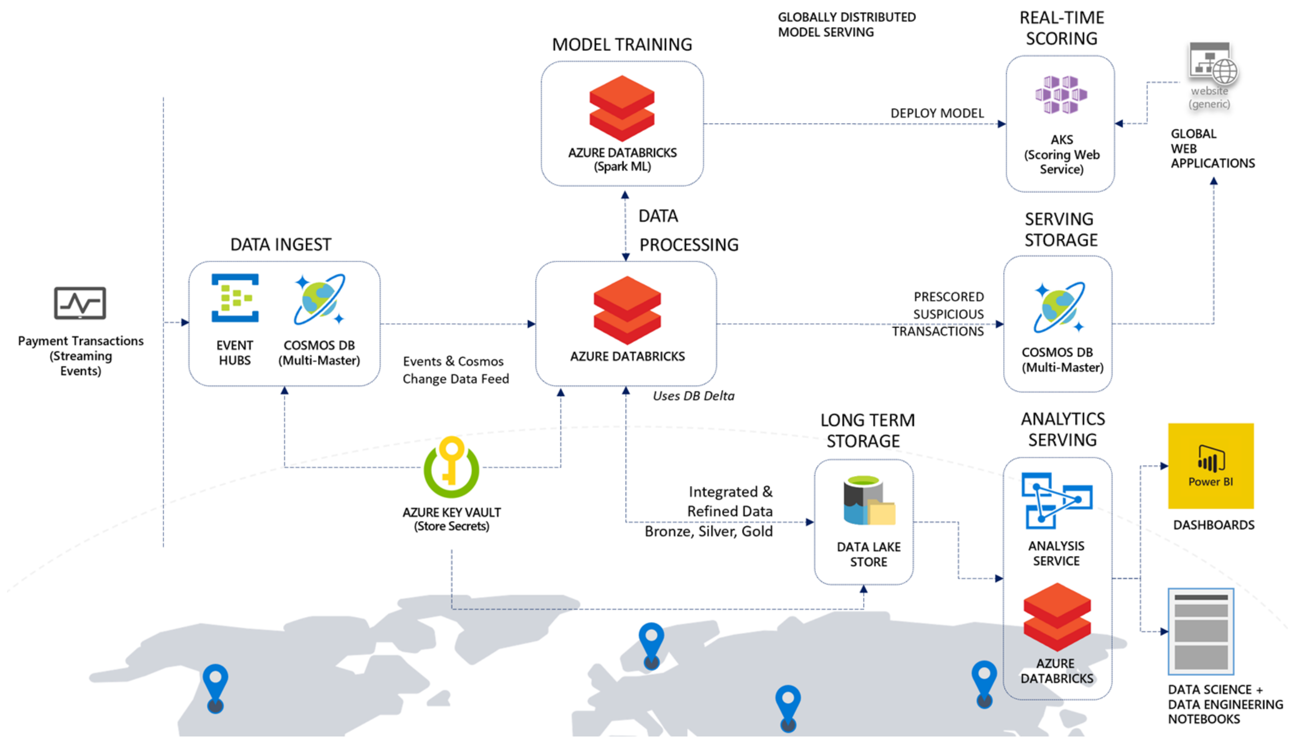
Mari Stephens, Chief Information Officer (CIO), Woodgrove Bank

The primary audience is the business decision makers and technology decision makers. From the case study scenario, this includes Mari Stephens, CIO for Woodgrove Bank. Usually, we talk to the infrastructure managers who report to the chief information officers (CIOs). We also speak with application sponsors (like a vice president [VP] line of business [LOB], or chief marketing officer [CMO]), or to those who represent the business unit IT or developers that report to application sponsors.

Preferred solution

High-level architecture

1. Without getting into the details (the following sections will address the particular details), diagram your initial vision for handling the top-level requirements for payment fraud detection, including stream capture and processing, long-term storage, model training, global distribution of the model for real-time scoring and of the pre-scored fraud data, and dashboards.



The solution begins with the payment transaction systems writing transactions to Azure Cosmos DB. Using the built-in change feed feature in Cosmos DB, the transactions can be read as a stream of incoming data within an Azure Databricks notebook, using the `azure-cosmosdb-spark` connector, and stored long-term within an Azure Databricks Delta Lake table backed by Azure Data Lake Storage. The Delta tables efficiently manage inserts and updates (e.g., upserts) to the transaction data. Tables created in Databricks over this data can be accessed by business analysts using dashboards and reports in Power BI, by using Power BI's Spark connector. Alternately, semantic models can be stored in Azure Analysis Service to serve data to Power BI, eliminating the need to keep a dedicated Databricks cluster running for reporting. Data scientists and engineers can create their own reports against Databricks tables, using Azure Databricks notebooks.

Azure Databricks also supports training and validating the machine learning model, using historical data stored in Azure Data Lake Storage. The model can be periodically re-trained using the data stored in Delta Lake tables or other historical tables. The Azure Machine Learning service is used to deploy the trained model as a real-time scoring web service running on a highly available Azure Kubernetes Service cluster (AKS cluster). The trained model is also used in scheduled offline scoring through Databricks jobs, and the "suspicious activity" output is stored in Azure Cosmos DB so it is globally available in regions closest to Woodgrove Bank's customers through their web applications.

Finally, Azure Key Vault is used to securely store secrets, such as account keys and connection strings, and serves as a backing for Azure Databricks secret scopes.

> **Note**: The preferred solution is only one of many possible, viable approaches.

Globally distributed data

1. Which data storage service would you recommend for storing the suspicious transactions? Remember, Woodgrove Bank wants to minimize access latency for their global customers. Be specific about how data is replicated.

Azure Cosmos DB is well-suited for delivering large amounts of data in a fast and reliable way through data centers around the world. In addition, there is a connector available for reading and writing to Cosmos DB from Spark clusters, like those in Azure Databricks, making it a good candidate for both data ingest and as a data serving layer.

It is a simple process to add or remove geographical regions associated with a Cosmos DB account at any time with a few clicks, or programmatically through a single API call. Because Cosmos DB automatically indexes the data stored within a Cosmos container upon ingestion, users can query the data without having to deal with a schema or the complications of index management in a globally distributed setup.

When Woodgrove Bank configures Cosmos DB, they should take care to select an appropriate partition key for their data. They should select a partition key which provides even distribution of storage and throughput (measured in requests per second) at any given time to avoid storage and performance bottlenecks. For instance, they might choose to partition by account number or customer region. This key should be present in the bulk of queries for read-heavy scenarios to avoid excessive fan-out across numerous partitions. This is because each document with a specific partition key value belongs to the same logical partition, and is also stored in and served from the same physical partition. Each physical partition is replicated across geographical regions, resulting in global distribution.

2. How does your chosen service handle scaling to meet varying levels of demand across different regions? Can you set specific capacity for specific regions?

   In Azure Cosmos DB, provisioned throughput is represented as request units/second (RUs). RUs measure the cost of both read and write operations against your Cosmos DB container. Because Cosmos DB is designed with transparent horizontal scaling (e.g., scale out) and multi-master replication, you can very quickly and easily increase or decrease the number of RUs to handle thousands to hundreds of millions of requests per second around the globe with a single API call.

   Cosmos DB allows you to increment/decrement the RUs in small increments of 1000 at the database level, and in even smaller increments of 100 RU/s at the container level. It is recommended that you configure throughput at the container granularity for guaranteed performance for the container all the time, backed by SLAs. Other guarantees that Cosmos DB delivers are 99.999% read and write availability all around the world, with those reads and writes being served in less than 10 milliseconds at the 99th percentile.

   When you set a number of RUs for a container, Cosmos DB ensures that those RUs are available in all regions associated with your Cosmos DB account. When you scale out the number of regions by adding a new one, Cosmos will automatically provision the same quantity of RUs in the newly added region. You cannot selectively assign different RUs to a specific region. These RUs are provisioned for a container (or database) for all associated regions.

3. Distributed databases that replicate data to multiple locations have some potential delay between when you write a record and when that record is available for reading. What options does your chosen service have to ensure the data is not "stale" when read? Are there any tradeoffs between reducing the window between writes, and if so, how do they apply to Woodgrove Bank's situation?

   Most distributed databases offer two consistency levels: strong and eventual. These live at different ends of a spectrum, where strong consistency often results in slower transactions because it synchronously writes data to each replica set. This guarantees that the reader will always see the most recent committed version of the data. Eventual consistency, on the other hand, asynchronously writes to each replica set with no ordering guarantee for reads. The replicas eventually converge, but the risk is that it can take several reads to retrieve the most up-to-date data.

Azure Cosmos DB was designed with control over the tradeoffs between read consistency, availability, latency, and throughput. This is why Cosmos DB offers five consistency levels: strong, bounded staleness, session, consistent prefix, and eventual. As a general rule of thumb, you can get about 2x read throughput for session, consistent prefix, and eventual consistency models compared to bounded staleness or strong consistency.

The Session consistency level is the default, and is suitable for most operations. It provides strong consistency for the session (application or connection), where all reads are current with writes from that session. Data from other sessions come in the correct order, but aren't guaranteed to be current. Session consistency level provides a balance of good performance and good availability at half the cost of both strong consistency and bounded staleness. As mentioned before, session provides about 2x read throughput compared to these two stronger consistency levels as well.

One thing to consider in your design is that you may get stronger consistency guarantees in practice. Read-consistency is tied to the ordering and propagation of the write/update operations. If there are no writes being made to the data set, then the consistency level is not a factor. Put another way, if no new writes are being made to the data set at the same time the reads are happening, then all reads, regardless of consistency level, will show the latest data.

In the case of Woodgrove Bank, Cosmos DB is being used for storing suspicious transactions that they identify by performing scheduled batch processing against all transactions. In this case, there are very few writes (which would happen in batch as the suspicious transactions are written out) compared to the number of reads (which might happen each time a customer reviews the flagged transactions). Because of this, for most of the read-heavy workload, all reads will get the latest data anyway. Therefore, a consistency level of Session will suffice for these documents, resulting in higher read throughput (approximately 2x faster) compared to strong and bounded staleness.

Data ingest

1. What are your recommended options for ingesting payment transaction events as they occur in a scalable way that can be easily processed while maintaining event order with no data loss?

   There are a couple of options in Azure for ingesting real-time streaming payment transactions. Which one you choose will depend on various factors, including the rate of flow (how many transactions/second), data source and compatibility, level of effort to implement, and long-term storage needs:

   1. **Event Hubs** is a Big Data streaming platform and event ingestion service, capable of ingesting millions of events per second. It supports multiple consumers (event processors), and is automatically scalable. Event Hubs is a good candidate for this architecture for the following reasons:

      - Fully managed PaaS with little configuration or management overhead.
      - Highly scalable to process millions of events per second. Use the Auto-inflate feature to automatically scale the number of throughput units to meet usage needs.

- Contains an optional Apache Kafka endpoint, allowing for event processing from existing Kafka-based applications. This also allows for simple integration with Apache Spark clusters, such as those hosted in Azure Databricks.
- Simultaneously supports real-time and batch processing through Event Hubs Capture. This feature allows one to easily capture and store all events in their raw form to either Azure Blob storage or Azure Data Lake Store for long-term retention and micro-batch processing.
- Event publishers (systems sending payment transaction data) can publish events using HTTPS, AMQP 1.0, or Apache Kafka 1.0 and above.
- Event consumers can process streams using .NET, Java, Python, Go, or Node.js.

Things to be cautious about are:

- Event Hubs guarantees consistency and event ordering *per partition*. Since it is important to keep payment transactions in order when processing them, you can guarantee ordering by setting a partition key on the event, or use a `PartitionSender` object to only send events to a certain partition. This has scaling implications if you plan to use more than one partition. It also has uptime implications if the partition you are trying to send to is unavailable. If no partition is defined, then the data is distributed across available partitions. You may choose to ensure ordering by region or merchant, as an example, by creating a partition for the region or merchant. The potential downside to this approach is finite number of available partitions for a given event hub (32 max by default, higher via quota increase request). Another option is to aggregate events within the processing application by time-stamping the event with a custom sequence number. This requires state to be kept within the processing application.

2. **Azure Cosmos DB** outputs a sorted list of documents that were changed in the order in which they were modified or inserted, as they are being written, by reading from its change feed. Like Event Hubs, the change feed output can be distributed across one or more consumers for parallel processing. Azure Cosmos DB is a good candidate for this architecture for the following reasons:

   - Fully managed PaaS with little configuration or management overhead.
   - Cosmos DB is highly scalable, and is already being used to store pre-scored fraud data.
   - An Apache Spark connector is available, allowing Azure Databricks clusters to directly access the change feed with very little code.
   - Cosmos DB with change feed enabled acts as both a raw data store for batch processing and stream processing.
   - Event publishers can publish events to Cosmos DB using .NET, Java, Node.js, and Python, using a number of APIs, such as SQL, Cassandra, MongoDB, Gremlin, and Azure Table Storage.
   - The change feed feature can only be used by the SQL and Gremlin APIs of Cosmos DB. Woodgrove will be using the SQL API, so they will be able to use the change feed feature.
   - Cosmos DB is globally accessible across many Azure regions, bringing it closer to distributed event publishers and consumers.
   - In a multi-region Azure Cosmos account, if a write-region fails over, the change feed will work across the manual failover operation and it will be contiguous.

- Cosmos DB Allows you to set a time-to-live (TTL) value, in seconds, on a container or on individual documents. This value tells Cosmos DB when to expire, or delete, the document(s) automatically. This setting can help save in storage costs by removing what you no longer need. Typically, this is used on hot data, or data that must be expired after a period of time due to regulatory requirements.

Things to be cautious about are:

- Similar to Event Hubs, feed item ordering is guaranteed per logical partition key. There is no global guaranteed order across the partition key values. Also similar to Event Hubs, changes are available in parallel across all logical partition keys of a container, allowing for parallel processing by multiple consumers. As discussed in the previous section (globally distributed data), choosing an appropriate partition key for Cosmos DB is a critical step for ensuring balanced reads and writes, scaling, and, in this case, in-order change feed processing per partition. While there are no limits, per se, on the number of logical partitions, a single logical partition is allowed an upper limit of 10 GB of storage. Logical partitions cannot be split across physical partitions. For the same reason, if the partition key chosen is of bad cardinality, you could potentially have skewed storage distribution. For instance, if one logical partition becomes larger faster than the others and hits the maximum limit of 10 GB, while the others are nearly empty, the physical partition housing the maxed out logical partition cannot split and could cause an application downtime.

2. Of the ingest options you identified previously, which would you recommend for the scenario?

While it is certainly possible to combine Azure Cosmos DB and Event Hubs for data ingest, this may result in unnecessary complexity, especially considering how closely their features (and challenges) align. For instance, it is possible to ingest all data into Cosmos DB and send events to Event Hubs through an intermediary event processor, such as Azure functions, in order to further process the event hub data downstream by consumers that can integrate with Event Hubs, but have no way to use Cosmos DB's change feed. However, this additional layer of abstraction is not necessary for the scenario of Woodgrove Bank. All things considered, the best approach is to select one after weighing the pros and cons of each. Remember, your decision should be based on the following factors: the rate of flow (how many transactions/second), data source and compatibility, level of effort to implement, and long-term storage needs.

Both services are capable of acting as the ingestion service. They both support a high rate of flow, and both have options for long-term storage needs. However, Event Hubs requires an extra step for long-term storage, which is handled through Event Hubs Capture, whereas Cosmos DB already provides this storage for raw data. Considering the level of effort to implement, this favors Cosmos DB since transactional data is already being written to a database, those applications or APIs can be updated to also write to Cosmos DB, or switch over to Cosmos DB entirely as the single database. Event Hubs requires adding an event service to their architecture and learning how to use it from their current systems.

Another point to consider is that Woodgrove Bank would like to ingest and serve data across multiple regions around the globe, and have that data synchronized as well. With Cosmos DB, you can simply add additional regions at any time either programmatically through its APIs, or through the "Replicate data globally" Cosmos DB settings in the portal. However, Event Hubs does not have this same option.

You have to create new Event Hubs in each region you wish to send data to, modify your sending applications, as well as your stream processing applications to account for the additional service endpoints.

A final decision point for using Cosmos DB for ingestion, is that it offers flexible message retention through its time-to-live (TTL) settings. This value can be set at the container level by applying a TTL value for all documents within, or on individual messages as they are sent. This optimization helps save in storage costs by automatically expiring (deleting) the documents after the specified period of time. For instance, you can set the TTL to 60 days to allow Woodgrove Bank to keep the streaming data available for that amount of time so they can reprocess in Azure Databricks, or query the raw data within the container as needed.

Event Hubs has a similar feature called message retention. You can set the value between one and seven days, or a maximum of four weeks if you contact Microsoft support. Setting the TTL for documents saved to Cosmos DB individually for any length of time desired (even beyond 7 days) is an advantage Cosmos DB has over Event Hubs when used for ingesting streaming data.

Data pipeline processing

1. Woodgrove Bank indicated that they would like a unified way to process both streaming data and batch data on a platform that can also support their data science, data engineering, and development needs. Which platform would you recommend, and why?

   The recommended platform that meets these needs for this solution is Azure Databricks. When it comes to working with big data in a unified way, whether you process it real-time as it arrives or in batches, Apache Spark provides a fast and capable engine that also supports data science processes, like machine learning and advanced analytics. Built as a joint effort by the team that created Apache Spark and Microsoft, Azure Databricks provides data science and engineering teams with a single platform for Big Data processing and Machine Learning.

   There are a lot of features Azure Databricks offers over top of standard Spark installations. The key features that make this a good choice for Woodgrove Bank are:

   - Integrates with Azure Active Directory for single sign-on and RBAC in certain scenarios.
   - Contains collaborative features such as the workspace that contains both private and shared folders, integrated change tracking of notebooks and integration with git source control systems like GitHub, and granular user and role-based permissions.
   - It is possible to start and stop clusters either manually or automatically, based on usage.
   - Supports running scheduled jobs for executing notebooks and libraries on a schedule.
   - Integrates with Azure Key Vault, which serves as a backing store for secrets within Azure Databricks, including automatic redaction of those secrets when users attempt to output them in a notebook.
   - Includes Databricks Delta Lake, which supports features Woodgrove Bank is looking for, such as the ability to upsert data into tables, optimize data storage, and simultaneously read data from tables which are being written to by streaming and batch processes.
   - Join disparate data sets found in data lakes.
   - Train and evaluate machine learning models at scale.

2. The big data systems Woodgrove Bank used in the past were only able to append new data to the end of existing data sets. This meant each time they had updates, they would actually create a duplicate row containing the changed data and then have to author queries to merge those rows so that they had a clean view of the current state of the data. How will your chosen platform cope with this challenge?

   Use Databricks Delta Lake. Delta Lake is a Spark table with built-in reliability and performance optimizations.

   You can read and write data stored in Delta Lake using the same familiar Apache Spark SQL batch and streaming APIs you use to work with Hive tables or Databricks File Store (DBFS) directories. Delta Lake provides the following functionality:

   - **ACID transactions** - Multiple writers can simultaneously modify a data set and see consistent views.
   - **DELETES/UPDATES/UPSERTS** - Writers can modify a data set without interfering with jobs reading the data set.
   - **Automatic file management** - Data access speeds up by organizing data into large files that can be read efficiently.
   - **Statistics and data skipping** - Reads are 10-100x faster when statistics are tracked about the data in each file, allowing Delta Lake to avoid reading irrelevant information.

3. How will your chosen data processing platform connect to and process data from your chosen data ingest solution for streaming data?

   Whether you have chosen to ingest your data using Azure Cosmos DB with change feed, or Event Hubs, you can connect to either of these directly from Spark. For Cosmos DB, use the `azure-cosmosdb-spark` connector, which lets you easily read to and write from Azure Cosmos DB via Spark DataFrames in either Python or Scala. For Event Hubs, either use the `azure-eventhubs-spark` library, or enable Kafka on your event hub and use the Spark-Kafka adapter (supports Kafka v2.0+), available as of Spark v2.4.

   Because the payment transactions will be arriving in real time, you will want to use Spark Structured Streaming to process the data. Think of a stream of data as a table to which data is continuously appended. In streaming, the problems of traditional data pipelines are exacerbated. Specifically, with frequent meta data refreshes, table repairs and accumulation of small files in intervals measured in seconds or minutes. Many small files result because data may be streamed in at low volumes with short triggers. Delta Lake is uniquely designed to address these needs.

4. What configuration would you need to apply to your solution to allow it to restart any stream processing in the case the job is stopped?

   When defining a Delta Lake streaming query, one of the options that you need to specify is the location of a checkpoint directory.

   ```
   .writeStream.format("delta").option("checkpointLocation", <path-to-checkpoint-directory>) ...
   ```

   This is a structured streaming feature. It stores the current state of your streaming job. Should your streaming job stop for some reason and you restart it, it will continue from where it left off.

Please note, if you do not have a checkpoint directory, when the streaming job stops, you lose all state around your streaming job and upon restart, you start from scratch.

5. What specific secrets might their processing solution want to store? How would they securely store and access those secrets?

   Specific secrets that they may need to be accessed by Azure Databricks are account names and keys for Azure Data Lake Storage, Cosmos DB connection strings or access keys, and Azure Machine Learning service account keys.

   To securely store these secrets, use Azure Key Vault, in addition to Key Vault-backed secret scopes within Azure Databricks. Azure Key Vault provides a service that allows you to securely centralize application secrets. The benefit of it being a centralized store of secrets, is that you only need to define those secrets, like connection strings or account keys, in one place which can be accessed by several Azure services as well as custom applications.

   Azure Databricks has two types of secret scopes: Key Vault-backed and Databricks-backed. These secret scopes allow you to store secrets, such as database connection strings, securely. If someone tries to output a secret to a notebook, it is replaced by `[REDACTED]`. This helps prevent someone from viewing the secret or accidentally leaking it when displaying or sharing the notebook.

Long-term data storage

1. As incoming data is processed, refined, and scored, all of the transactions need to be persisted to long-term storage for analysis, model training and validation, and reporting. This storage needs to handle long-term growth, be fast enough to rapidly ingest new data while simultaneously handling reads against the same data set without interference, and act as a reliable data source for dashboards and reports. Which is your recommended long-term data storage solution, keeping in mind its role within your selected data pipeline processing platform?

   Although our proposed solution uses Delta Lake to store data within a Delta Lake table, we still need to select the appropriate storage service it uses under the covers. For this, use Azure Data Lake Storage Gen2 (ADLS Gen2). ADLS Gen2 is a set of capabilities dedicated to big data analytics, built on Azure Blob storage. Data Lake Storage Gen2 is the result of converging the capabilities of Microsoft's two existing storage services, Azure Blob storage and Azure Data Lake Storage Gen1. Features from Azure Data Lake Storage Gen1, such as file system semantics, directory, and file level security and scale are combined with the low-cost, tiered storage, and high availability/disaster recovery capabilities from Azure Blob storage.

   ADLS Gen2 makes Azure Storage the foundation for building enterprise data lakes on Azure. Designed from the start to service multiple petabytes of information while sustaining hundreds of gigabits of throughput, ADLS Gen2 allows you to easily manage massive amounts of data.

   ADLS Gen2 allows you to manage and access data just as you would with a Hadoop Distributed File System (HDFS). The new Azure Blob Filesystem (ABFS) driver allows Azure Databricks to access data stored in ADLS Gen2. This driver is optimized specifically for big data analytics, and overcomes the inherent deficiencies of the previous WASB driver.

   With Delta Lake managing the files stored within ADLS, the files will be optimized through re-indexing and combining numerous small files into fewer, read-optimized large files. Other enhancements provided by Delta Lake allow for simultaneously handling reads and writes against the same data set without interference. The Delta Lake table can be accessed by BI dashboard and reporting services that support reading Hive tables, such as Power BI.

2. How do you ensure your data is continuously optimized within your chosen long-term data storage solution, given the requirements to store inserts, updates, and deletes while avoiding generating very small, un-optimized files?

   Historical and new data is often written in very small files and directories. This will especially be the case when dealing with real-time payment transaction data. The result is that a query on this data may be very slow due to network latency or volume of file metadata. The solution is to compact many small files into one larger file. Delta Lake has a mechanism for compacting small files. It supports the `OPTIMIZE` operation, which performs file compaction. Small files are compacted together into new larger files up to 1GB. The 1GB size was determined by the Databricks optimization team as a trade-off between query speed and run-time performance.

   `OPTIMIZE` is not run automatically because you must collect many small files first. Run `OPTIMIZE` more often if you want better end-user query performance. Since `OPTIMIZE` is a time consuming step, run it less often if you want to optimize cost of compute hours. To start with, run `OPTIMIZE` on a daily basis (preferably at night when fewer jobs are run), and determine the right frequency for your particular business case. You can use Databricks Jobs to schedule automatically running the `OPTIMIZE` operations. In the end, the frequency at which you run `OPTIMIZE` is a business decision.

3. Woodgrove Bank wants to retain all raw data (bronze layer), then parse that data into query tables (silver layer) which can be joined with dimension tables, such as account information. They also would like to have summary tables (gold layer) containing business-level aggregates used for their dashboards and reports. How would you support these requirements in your long-term storage solution?

   To support this, you would first define the storage paths for the mounted Azure Data Lake Storage account for each layer (bronze, silver, and gold). These will be used to store files for each of the related Delta Lake tables. In addition, define a path to your checkpoint directory (`checkpointLocation` option) for stream processing, so you can continue where you left off if your stream processing job stops for any reason. For example:

```
basePath       = "/mnt/adlsGen2/payment-transactions-streaming"
bronzePath     = basePath + "/transactionsRaw.delta"
silverPath     = basePath + "/transactions.delta"
goldPath       = basePath + "/transactionsSummary.delta"
checkpointPath = basePath + "/checkpoints"
```

> Note: The bronze path may not be needed, depending on how you configured your data ingestion layer. For instance, if you are using Event Hubs, you can configure Event Hubs Capture to store the raw data directly to ADLS. You will process the stream in Databricks, make any transformations to the data, and store the transformed data into your query (silver) tables. If you are using Azure Cosmos DB, the container you use for ingest will act as the bronze path

> containing raw data. Then, just like you do for Event Hubs, you will process the stream, this time from the Cosmos change feed, in Databricks, transform the data, and store it in the silver tables.

Here is an example of creating a query table (silver) by reading a stream into a Delta Lake query directory (`silverPath`), while also transforming the data:

```
from pyspark.sql.functions import unix_timestamp

(spark.readStream
    .format("delta")
    .load(str(bronzePath))
    .select(col("transactionID"),
            col("accountID"),
            col("transactionAmount"),
            col("transactionCurrencyCode"),
            unix_timestamp(col("timestamp"), "yyyy-MM-
dd'T'HH:mm:ss.SSSX").cast("timestamp").alias("timestamp"),
            col("transactionDeviceId"),
            col("transactionIPaddress"))
    .writeStream
    .format("delta")
    .option("checkpointLocation", checkpointPath + "/silver")
    .outputMode("append")
    .start(silverPath)
)
```

Now we can create a Delta Lake table, referring to the silver path location:

```
spark.sql("DROP TABLE IF EXISTS Transactions")

spark.sql("""
    CREATE TABLE Transactions
    USING Delta
    LOCATION '{}'
""".format(silverPath))
```

All of this code can be run from within one or more Databricks notebooks, either manually or on a schedule.

Model training and deployment

1. Describe how your chosen data processing platform will support machine learning model training and deployment. The model will need to be trained on and validated against historical payment transaction data that includes known fraudulent transactions.

   Azure Databricks supports machine learning training at scale. This means that it can handle the historical payment transaction data, which Woodgrove Bank said it can provide as a series of CSV files, and transform that data as needed for cleanup and feature selection. A large portion of that data will be used for training, and the rest can be used to validate the performance of the trained model.

For model deployment, use Azure Machine Learning service and the Azure Machine Learning SDK to host a trained machine learning model and automatically deploy the model to an Azure Kubernetes Service (AKS) cluster. Creating the AKS cluster is a one-time process for your workspace, where after you can reuse it for multiple deployments as the model gets updated through re-training. The basic steps are as follows:

1. Register the model in the workspace model registry.
2. Build a Docker image, including:
   - Download the registered model from the registry.
   - Create a dockerfile, with a Python environment based on the dependencies you specify in the environment yaml file.
   - Add your model files and the scoring script you supply in the dockerfile.
   - Build a new Docker image using the dockerfile.
   - Register the Docker image with the Azure Container Registry associated with the workspace.
3. Deploy the Docker image to Azure Kubernetes Service (AKS).
4. Start up a new container (or containers) in AKS.

The above can be done through a series of scripts for automation. To deploy globally, modify the script to deploy to AKS clusters hosted within different regions.

2. How will you schedule regular batch scoring of fraud data using the trained model, and make that data available to Woodgrove Bank's web applications at a global scale?

   The models that they have trained within Azure Databricks notebooks can be saved to ADLS or to Azure Machine Learning service. These saved models can then be re-loaded by a notebook or a job that executes a notebook to batch score the “suspicious transactions” on a scheduled basis. The notebook logic would use the Cosmos DB Spark connector to push the scored results out to the globally distributed set of containers, making the suspicious transactions “locally” available to authorized consuming applications.

Dashboards and reporting

1. Woodgrove Bank's business analysts would like to have a set of dashboards they can monitor that provide real-time views of fraud trends at a global scale. Thinking back to how your proposed solution provides a set of summary (gold) tables containing business-level aggregates, what do you propose using to meet this requirement? Be specific about how this solution will be put in place and which features it supports.

   - Option 1: No additional services required, but with a comparatively higher cost:

     Power BI can query any tables created in Azure Databricks. This connection is facilitated via a JDBC connector, whose connection information is provided in the Get Data UI of Power BI by selecting the Spark connector. While the data for these tables may be stored in ADLS, the experience to the analysts is similar to them querying a table from a more traditional relational database. With the table connection in place, analysts can build reports and dashboards using Power BI.

- Option 2: Additional service and synchronization required, but lower cost option:

  A more cost-effective approach to reporting is to store the summary (gold) table data in Azure Analysis Services, which you connect to from Power BI. This eliminates having to have a dedicated Databricks cluster running at all times for reporting and analysis. Azure Analysis Services is a fully managed platform as a service (PaaS) for storing your data in a tabular semantic data model. You will need to synchronize the data to your semantic models in Azure Analysis Services to keep your dashboard and reports up-to-date. This can be done as an extra step during stream processing, using rolling aggregates over a time window, or by batch processing on a schedule, using an Azure Databricks job or an orchestration service like Azure Data Factory.

2. How do you propose giving access to this same data to Woodgrove Bank's data scientists and data engineers within the data processing environment wherein they can craft complex queries and data visualizations?

   Data scientists and data engineers would use Azure Databricks notebooks to craft complex queries and data visualizations.

Checklist of preferred objection handling

1. It's not clear to us if we can only use Cosmos DB as our web app's database, or if we should consider using it in other parts of our advanced analytics data pipeline such as for real-time transaction ingest or for serving of offline processed data.

   Cosmos DB was created from the ground-up as a distributed database service that transparently handles the complexity of running within multiple regions around the world. Because providing data to customers around the globe is a key requirement, Cosmos DB is an ideal choice for ingesting data where it arrives, and serving the data where it is requested. It's major advantage when operating at a global scale is its high concurrency with low latency and predictable results. This combination is unique to Cosmos DB and ideal for Woodgrove Bank's needs. The change feed feature of Cosmos DB makes it useful for both storing raw transaction data as it is written, and notifying consumers, like Azure Databricks, of changes as they occur for real-time processing.

2. Does Cosmos DB integrate with open source big data analytics like Apache Spark?

   Yes, the `azure-cosmosdb-spark` connector can be used to read and write to Cosmos DB, and is also capable of using the change feed to react to events as they occur. Visit https://github.com/Azure/azure-cosmosdb-spark to learn how to use the connector and to access sample code.

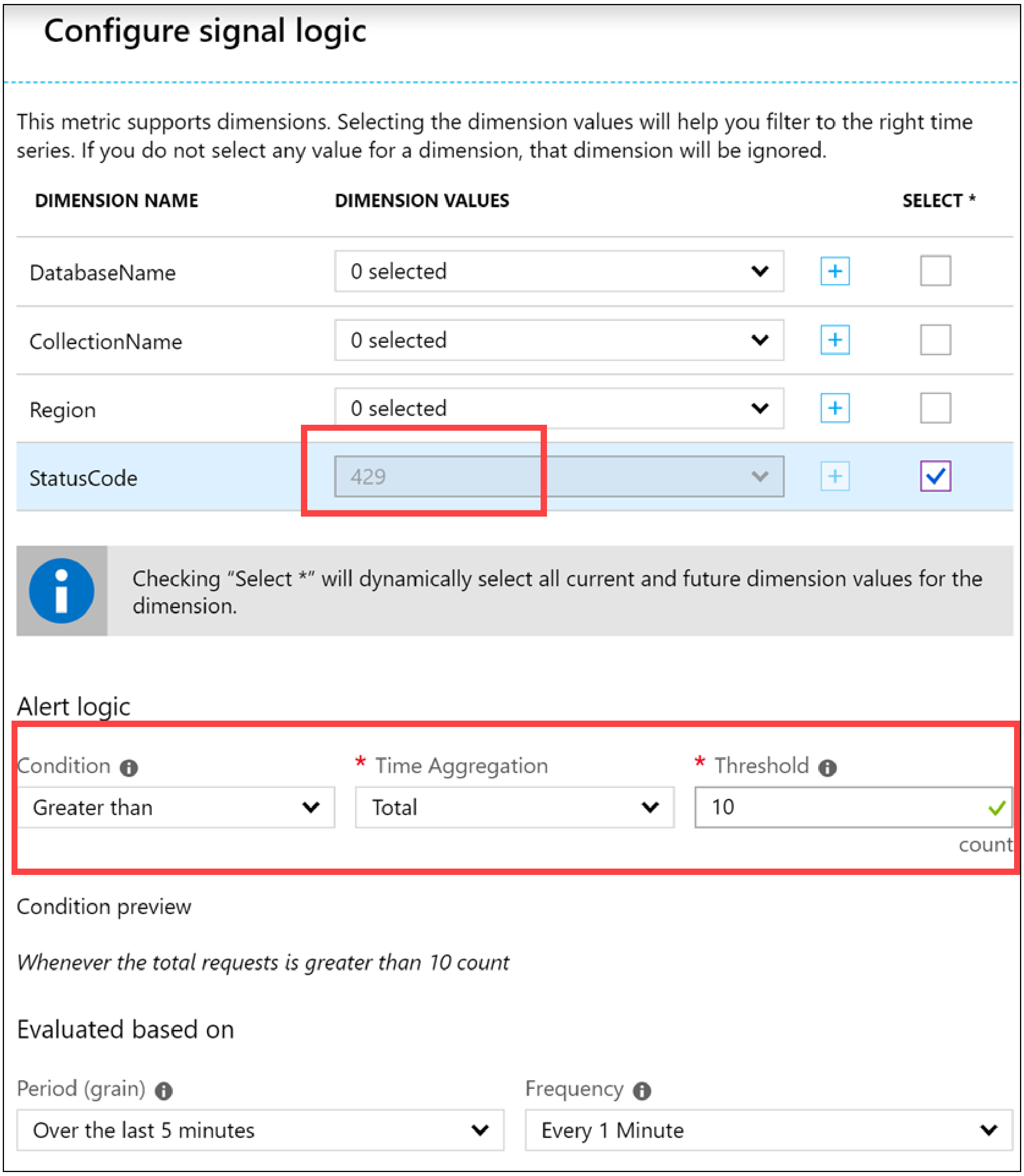
3. Properly selecting the right algorithm and training a model using the optimal set of parameters can take a lot of time. Is there a way to speed up this process?

   The typical approach to model training involves a time-consuming process trying dozens or even hundreds of combinations. Data scientists often try different ways of normalizing the data, different algorithms and different settings for those algorithms (hyperparameters). Data scientists will often setup a grid-search approach that will run multiple independent tests using differing combinations, measuring the performance of each and choosing the combination that provides the best results according to some performance metrics they select. With Azure Machine Learning AutoML, this search process is automated, and greatly simplifies the setup to try the typical combinations and quickly identify the best performing model against a user-selected performance metric. AutoML is used via the Azure Machine Learning Python SDK and can be utilized within Azure Databricks notebooks.

4. We are concerned about how much it costs to use Cosmos DB for our solution. What is the real value of the service, and how do we set up Cosmos DB in an optimal way?

   Azure Cosmos DB's greatest strength is that it provides a multi-model, globally available NoSQL database with high concurrency, low latency, and predictable results. The fact that it transparently synchronizes data to all regions, which can quickly and easily be added at any time, adds value by reducing the amount of development required to read and write the data and removes any need for synchronization.

   The cost of all database operations, such as throughput, CPU, and memory, is normalized by Azure Cosmos DB and is expressed in terms of Request Units (RUs). The cost to read a 1-KB item is 1 Request Unit (1 RU) and the minimum RUs required to consume 1 GB of storage is 40. The cost of writing a 1-KB item is 5 RUs. Many people risk over-allocating RUs to their containers when they may not need such high levels at all times. To save costs, a good tactic is to ramp up the number of RUs during batch processing or other read/write-heavy loads, then reduce the number of RUs afterwards. This can be done automatically or manually through the portal. An example of how this can be automatically done is to monitor Cosmos DB with Azure Monitor and set an alert rule that calls an Azure Function to scale up the number of RUs for that container. Then you would have another process to scale down as needed. You configure Azure Monitor to monitor the total requests for a 429 HTTP status code (which means the requests are throttled), apply alert logic where the total number of these codes are greater than a pre-defined value (like 10) over the last 5 minutes.



Configure signal logic

This metric supports dimensions. Selecting the dimension values will help you filter to the right time series. If you do not select any value for a dimension, that dimension will be ignored.

DIMENSION NAME	DIMENSION VALUES	SELECT *
DatabaseName	0 selected	☐
CollectionName	0 selected	☐
Region	0 selected	☐
StatusCode	429	✓

Checking "Select *" will dynamically select all current and future dimension values for the dimension.

Alert logic

Condition	* Time Aggregation	* Threshold
Greater than	Total	10 count

Condition preview

Whenever the total requests is greater than 10 count

Evaluated based on

Period (grain)	Frequency
Over the last 5 minutes	Every 1 Minute

Cosmos DB also offers flexible time-to-live (TTL) that can be set at the container-level or on individual documents. This value tells Cosmos DB to expire, or delete, a document after a certain amount of seconds. This value can be set to as little as one second, or months into the future. As a result, you can save storage costs for records that are no longer needed. The flexibility of setting TTL at the container or record-level is a unique characteristic of Cosmos DB.

Another pitfall that developers or database administrators experience, especially those who are used to using traditional relational databases, is that they tend to want to create a Cosmos DB container for each "table", or entity type they wish to store. Instead, you should consider storing any number of entity types within the same container. The reason for this is that there is a cost associated with each container that you add. Because containers do not enforce any type of schema, you are able to store entities of the same type with different schemas (likely due to changes over time or from excluding properties with no values), or entirely different entities within that container. A common approach to dealing with different types of entities within a container is to add a string property like `collectionType` so that you can filter query results by that type. For instance, Woodgrove Bank stores transaction and suspicious activity data within the same container. They could assign a value of "Transaction" to the transaction entities, and "SuspiciousActivity" to the suspicious activity entities. Both types and many others can coexist within the container and can easily be filtered by the `collectionType` property.

5. How do we optimize our indexes for both write-heavy and read-heavy workloads?

   Woodgrove Bank should create two or more Cosmos DB containers for their scenario, based on different indexing requirements and access patterns. Let us call the container in which streaming data is ingested, `telemetry`. This container will likely have a higher throughput than the other containers, and its index should also be optimized for write-heavy workloads. To do this, Woodgrove should not use the default index for the 'telemetry' container, but only include paths that they know will be queried further down the processing pipeline. The other containers, which tend to have a more read-heavy workload, can benefit from the default indexing policy.

   The default indexing policy for newly created containers indexes every property of every item, enforcing range indexes for any string or number, and spatial indexes for any GeoJSON object of type Point. This allows you to get high query performance without having to think about indexing and index management upfront. Since we need faster writes for `telemetry`, we exclude unused paths. The use of indexing paths can offer improved write performance and lower index storage for scenarios in which the query patterns are known beforehand, as indexing costs are directly correlated to the number of unique paths indexed.

   The indexing mode for all the Cosmos DB containers should be set to **Consistent**. This means the index is updated synchronously as items are added, updated, or deleted, enforcing the consistency level configured for the account for read queries. The other indexing mode one could choose is None, which disables indexing on the container. Usually this mode is used when your container acts as a pure key-value store, and you do not need indexes for any of the other properties. It is possible to dynamically change the consistency mode prior to executing bulk operations, then changing the mode back to Consistent afterwards, if the potential performance increase warrants the temporary change.

Customer quote (to be read back to the attendees at the end)

"As a bank who is entrusted by customers all around the world with processing online payments, we have to be pro-active in detecting online fraud and protecting those customers. If we do not detect such activity quickly enough, things can spiral out of control, causing losses and in both monetary terms and our customers' trust. Azure has really enabled us to add that layer of security and peace of mind at a grand scale."

--- Mari Stephens, CIO, Woodgrove Bank