Efficient Near Real-Time Event Ingestion using DLT: Insights & Lessons

Kavin-Engineer@nextdoor

nextdoor

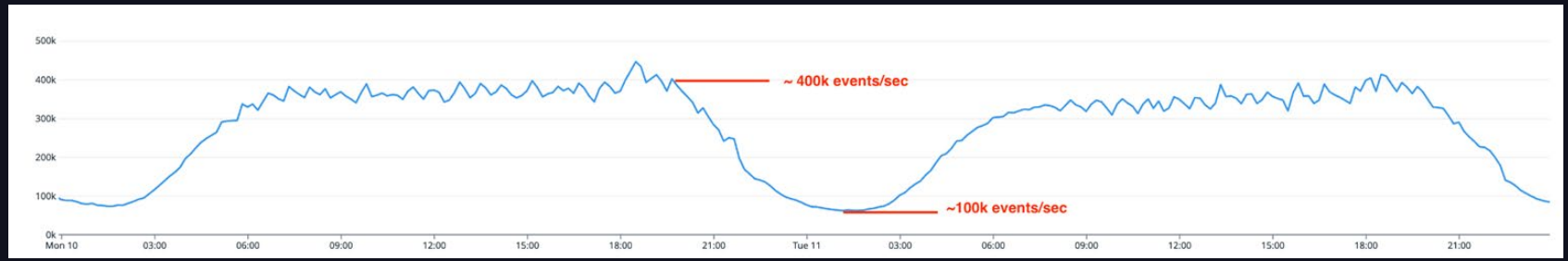
Introduction

- Nextdoor's mission is to create a kinder world by connecting neighbors and real-world connections
- We operate in the US, Canada, Europe & Australia today and have over 43 million weekly active users
- We run on AWS cloud today
- We are hiring !!! Apply @ https://about.nextdoor.com/careers/

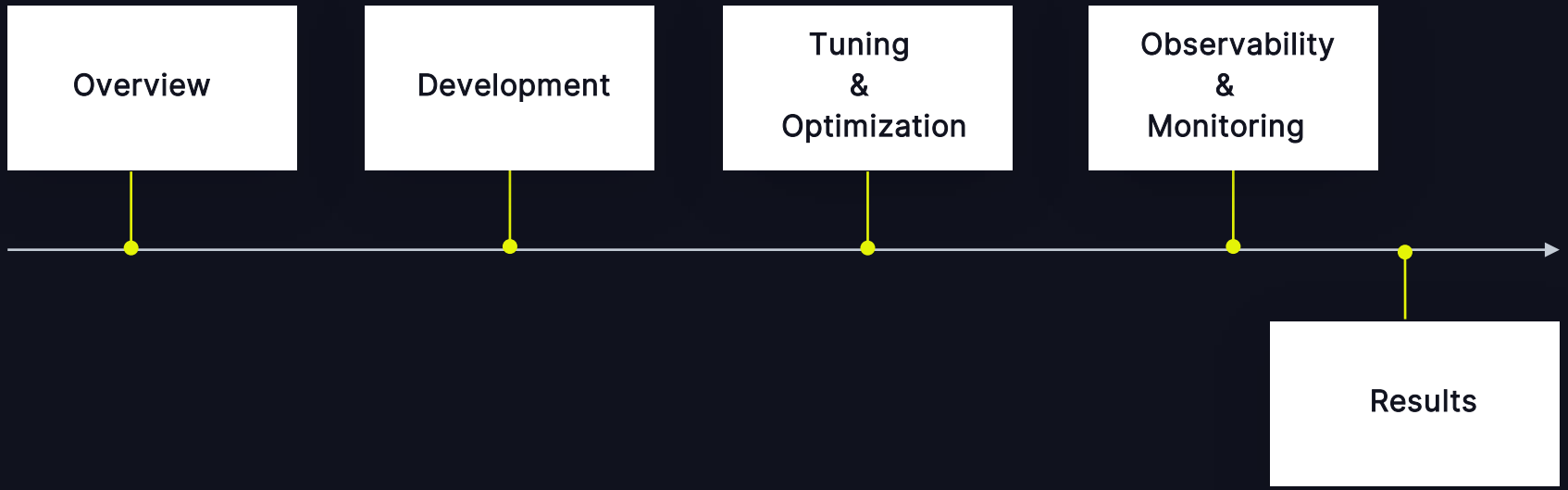
nextdoor

Events

- App is hosted in 4 AWS regions
- Up to `400k events / sec`
- Includes client (impression, click, etc.,) & server events (requests, ab_tests.,etc)

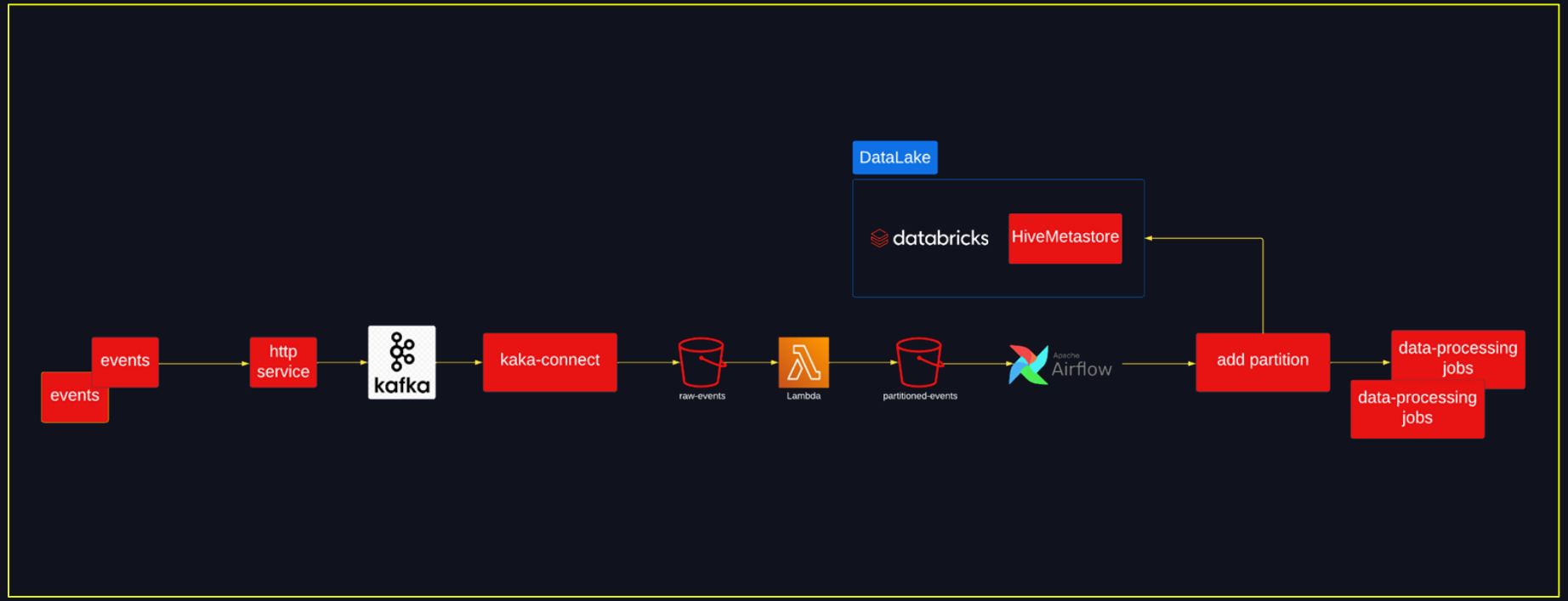


DLT adoption phases



Overview

Nextdoor's event ingestion pipeline pre DLT



Overview

Nextdoor's event ingestion pipeline pre DLT

- Http service published events to one Kafka topic per region
- Kafka-connect application dumped data to S3 bucket ~1 minute interval
- AWS Lambda partitioned the event and writes data to another S3 bucket with retry
- Hourly job in Airflow ran to add the partitions in HiveMetastore

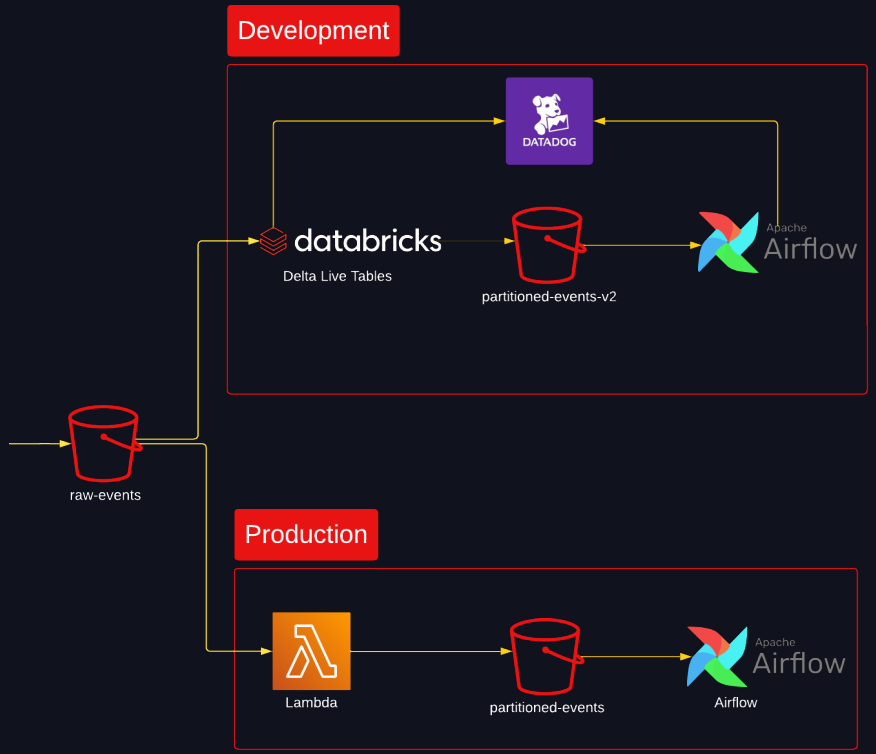
Overview

Goals

- Functional
  - Events to reach Data Lake near real-time to enable quicker analysis
  - At most once instead of at least once event delivery
- Non-functional
  - No increase in compute and/or storage cost
  - Insights into ingestion

Overview

Vision



- After deciding we wanted to stream in data we decided to try DLT with Autoloader
- DLT is Databricks managed and offers schema evolution, exactly once per streaming table
- Autoloader - Incremental ingests files from a cloud storage like S3
- Datadog integration for observability and monitoring

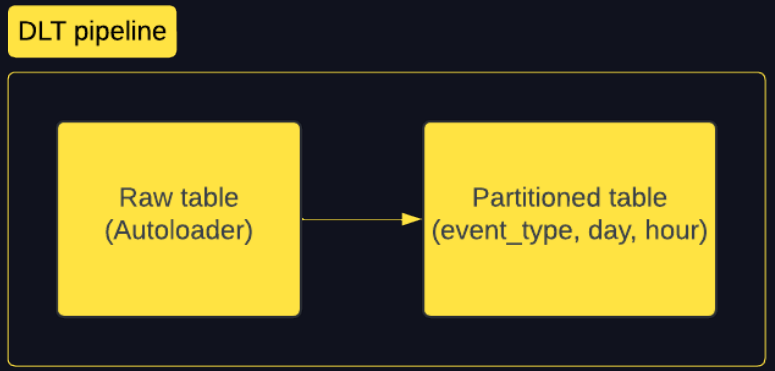
Development

Considerations

- Chose Python Interface over SQL
- Staging data - `< 1%` of production event volume
- Autoloader has lots of default setting
  - `cloudFiles.format : json`
- Node types
  - `driver -> c5.2xlarge, workers -> m5.2xlarge`
- Tags for cost attribution

Development

DLT tables



- Pretty loose schema
  - json_body: string, headers Struct<event: string, event_ts: bigint, ingestion_ts: bigint>
- Hive style Partition
  - `event_type={some_event_type}/day=dd-MM-YY/hour=HH`

Development

Staging

Development

Production

- Driver ran into OOM issues
- Each micro batch was taking several minutes to run

Development

What went wrong

- Spark Driver ran into OOM issues
  - Autoloader directory listing caused driver to do `diff of extremely high number` of S3 prefixes
- Each Spark micro batch was taking way longer to run
  - We were processing events from `60 days ago` and running large micro batches
- Other - EC2 Spot termination
  - Spot termination was causing `spike` in latency and `increased` cost
- Other - DLT failed because of missing checkpoints
  - S3 lifecycle policy `removed checkpoints` making the DLT crash hard

Tuning & Optimization

Autoloader tuning

- `cloudFiles.useNotifications: true` - turns on FileNotification instead of Directory listing

- `cloudFiles.queueUrl: <sqs-queue-url>` - If you have existing SQS queue
- `cloudFiles.maxFilesPerTrigger: 2000` - default is 1000.
- `cloudFiles.includeExistingFiles: false` - default is true, if you only care about new data

Tuning & Optimization

Cluster tuning

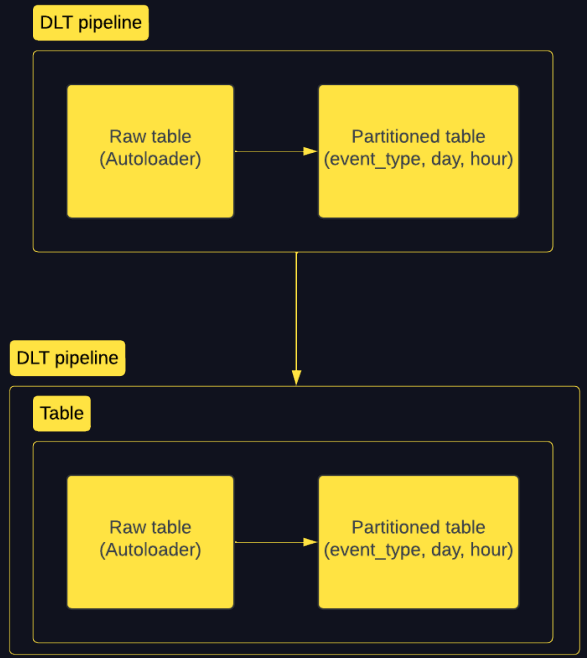
- Switched to On Demand instances
  - To avoid Spot termination and decrease cost and latency
  - It turned out that On Demand worked out much cheaper
- Instance pools for driver & worker for faster recovery
- Fleet pools for workers for better availability
- On Demand worked out cheaper than Spot because of less interruptions

Tuning & Optimization

Storage cost

- Duplicative data between raw and partitioned table costing 2X storage cost

- Solution: Combine into one DLT table with autoloader and partitioning



Observability & Monitoring

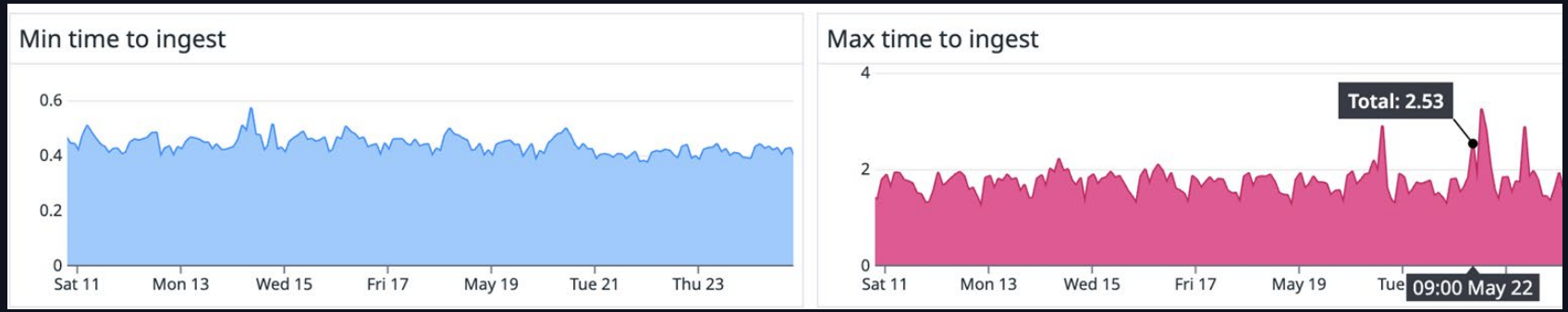
Default metrics

- Latency can be different based on each data that's processed in micro batch. It is a great metric to monitor too to track regressions
- Tune the autoloader for number of files and/or bytes till processing rate & input rate should ideally be very close

Observability & Monitoring

Custom metrics

- Spark observe API to send data specific metrics - real easy to use
- Min time to ingest an event from Nextdoor's servers to get ingested to DataLake `~24 seconds`
- The last event to reach DataLake is between `~2 - 3 minutes`



Observability & Monitoring

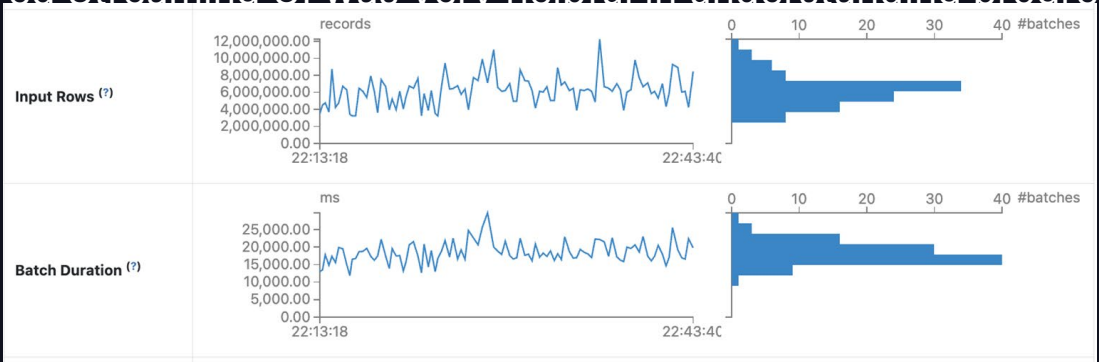
SQS metrics

- SQS queue depth is good metric to monitor to ensure DLT is up and picking up new messages

Results

What went well

- Optimized writes wrote bigger files automatically
- Additionally, maintenance cluster performs compaction & vacuum every day
- Autoscaling worked pretty well stabilizing at minimum of 10 & maximum of 20 nodes
- Structured Streaming UI was very helpful in understanding progress, latency etc



Results

Finance was happy

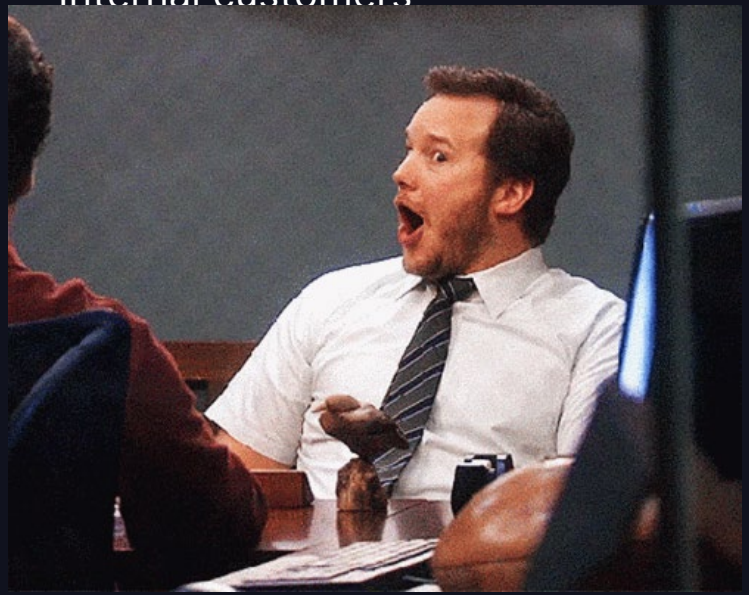
Finance team

- Because we reduced compute cost by 75% with DLT over previous solution

Results

Data users were happy

Internal customers



- Because of increased data freshness enabling analysis, debugging & building near real-time aggregates
- Better query performance than the previous solution because of large files & stats

Results

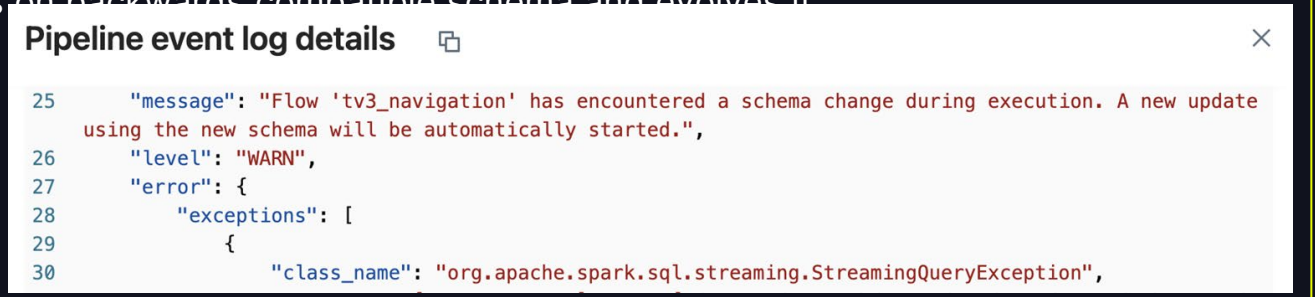
Data Platform team was happy

Data Platform team

- Because of the operational visibility & better monitoring

Next

- We are going to explore Serverless if possible to not have to worry about cloud infrastructure.
- Strongly typed events in Parquet
  - Supporting Structs, Arrays
  - `cloudFiles.format : parquet`
  - DLT auto restarts on backwards compatible schema and evolves it



Thanks

- Nextdoor team
  - Zack Shapiro - Head, Data Platform
  - Sebastian Csar - Software Engineer, Data Platform

- Databricks team
  - Toby Messinger - Solutions Architect
  - Jitesh Soni - Data Architect
  - Janelle Davies - AE