DATAVERSITY DEMO DAY · DATA ARCHITECTURE & INTEGRATION · SEPT 16, 2026

Turn real-time data into AI-ready data with unified data integration

Kavya Nagarajan

Product Manager, IBM watsonx.data integration

IBM

The forces redefining enterprise data integration

Real-time expectations

Milliseconds now define competitive advantage

AI is raising the bar

Trusted, complete data is now a baseline requirement

Rising complexity

More sources, more formats, more places data lives

Where the pressure shows up

How fragmented modernization increases complexity, risk, and instability in data operations.

Latency impacting real-time decisions

Pipeline instability from schema drift

Manual approaches slow issue detection

Rising operational overhead per workload

Why a unified approach — not just another tool

1

Agent-first

Empower every user to build pipelines faster

Build pipelines using natural language through an agentic experience — business users self-serve, data engineers automate the manual work.

2

Real-time, observable, governed

One control plane, every workload

Run batch, streaming, and replication pipelines from a single plane, with AI-powered observability built in from day one.

3

Hybrid by design

Execute data wherever it resides

Process data across any cloud, geography, or on-prem environment — minimizing latency and eliminating costly rewrites.

WHERE WE START

Real-time streaming

Streaming

Continuous, low-latency ingestion and processing in real time.

- React to events as they happen, not on a schedule
- Power real-time analytics, alerting, and downstream apps
- The same engines you'll see live in a moment

WHAT YOU'LL SEE NEXT

Let's see it live.

- Build a streaming pipeline from Kafka in real time
- Set up replication to keep systems continuously in sync
- All from the same control plane

IBM

BACK TO THE PLATFORM

Agentic, not just automated

Agentic AI

Teams design and operate pipelines at scale using agentic capabilities — across everything you've seen today.

- Natural-language pipeline creation for self-serve users
- Autonomous observability that flags and resolves issues before they cascade
- Speed and self-service — without giving up governance or control

BACK TO THE PLATFORM

See everything, in one place

Observability

End-to-end visibility into every pipeline, regardless of how it runs.

- Batch and streaming jobs in the same dashboard
- Lineage, freshness, and SLA tracking built in — not bolted on
- Catch problems before they cascade into an outage

WHAT YOU'LL SEE NEXT

Let's see it live, again.

- Use an AI agent to build a batch pipeline from cloud storage into a database
- Run the pipeline and watch it execute end-to-end
- Set up an alert to see unified observability in action

Faster where it counts

A new streaming engine (in tech preview)

The power of Flink without it's complexity.

- Engineered to move more data, with lower latency and lower operational cost at scale
- Abstracts away Flink's operational complexity — no scarce distributed-systems expertise required
- The only solution pairing a **mature visual pipeline builder with real Flink execution** — competitors leave you writing and maintaining Flink code by hand

HAND-CODED ON FLINK

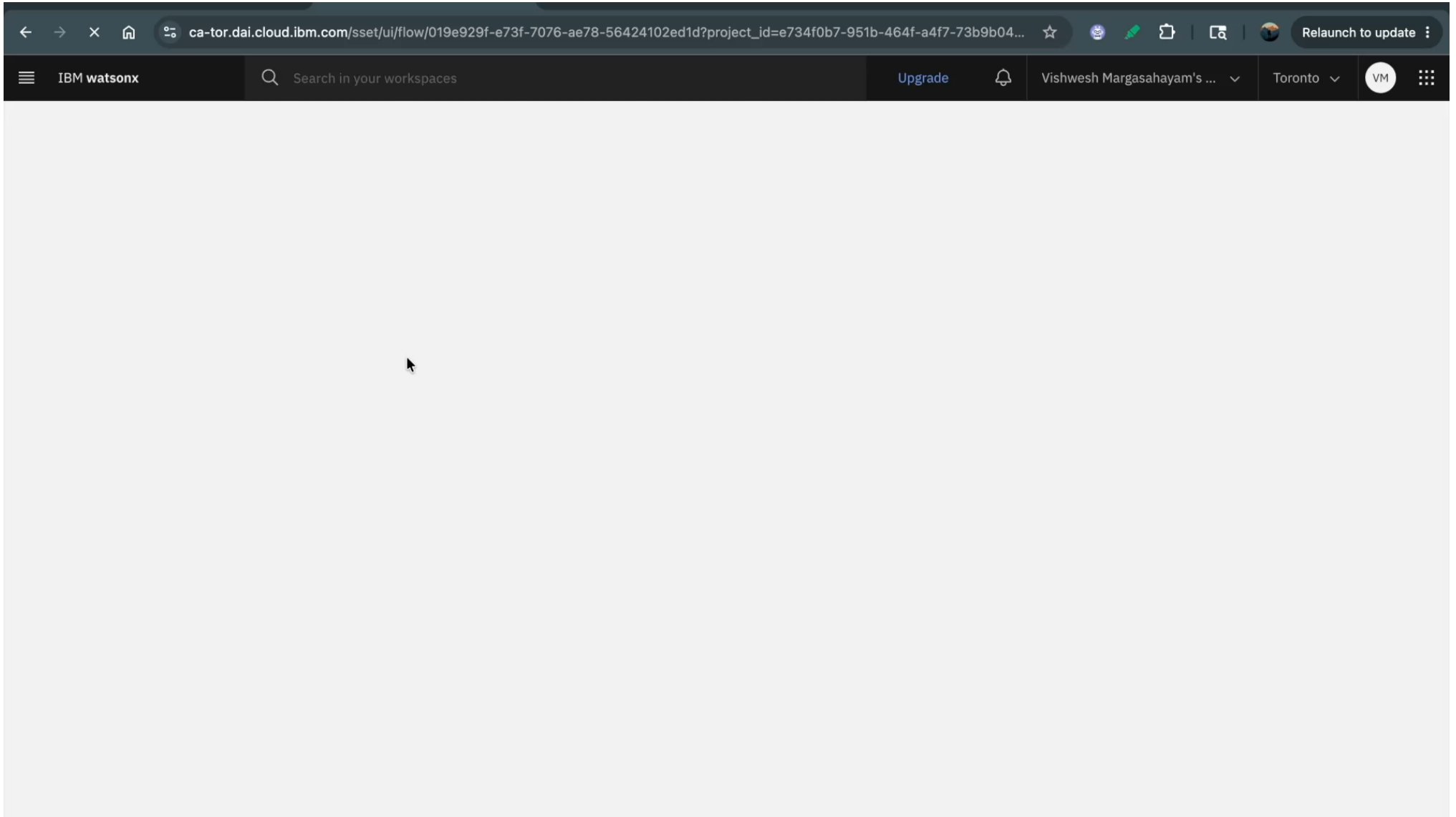
Building a real-time pipeline stage by stage in Java

FlashSaleRevenueJob.java

```java
public class FlashSaleRevenueJob {
  public static void main(String[] args) throws Exception {
    StreamExecutionEnvironment env =
        StreamExecutionEnvironment.getExecutionEnvironment();
    env.enableCheckpointing(10_000, CheckpointingMode.EXACTLY_ONCE);
    env.getCheckpointConfig().setMinPauseBetweenCheckpoints(5_000);
    env.getCheckpointConfig().setCheckpointStorage(
        "s3://checkpoints/flash-sale-job/");
    env.setStateBackend(new EmbeddedRocksDBStateBackend());
    env.setRestartStrategy(RestartStrategies
        .fixedDelayRestart(3, Time.seconds(10)));

    KafkaSource<Order> source = KafkaSource.<Order>builder()
        .setBootstrapServers(KAFKA_BROKERS)
        .setTopics("orders-demo")
        .setStartingOffsets(OffsetsInitializer.latest())
        .setValueOnlyDeserializer(new OrderDeserializer())
        .build();

    WatermarkStrategy<Order> watermarks = WatermarkStrategy
        .<Order>forBoundedOutOfOrderness(Duration.ofSeconds(5))
        .withTimestampAssigner((o, ts) -> o.getEventTime());

    DataStream<Order> orders = env
        .fromSource(source, watermarks, "orders-source")
        .filter(o -> o.getAmount() > 500);

    DataStream<EnrichedOrder> enriched = AsyncDataStream
        .unorderedWait(orders, new CustomerLookupFunction(),
            1000, TimeUnit.MILLISECONDS, 100);

    DataStream<EnrichedOrder> withCampaign = AsyncDataStream
        .unorderedWait(enriched, new CampaignLookupFunction(),
            1000, TimeUnit.MILLISECONDS, 100);

    withCampaign
        .keyBy(o -> Tuple3.of(o.getCategory(), o.getTier(),
            o.getCampaignName()))
        .window(TumblingEventTimeWindows.of(Time.seconds(10)))
        .aggregate(new RevenueAggregator(),
            new WindowMetricsFunction())
        .addSink(JdbcSink.sink(
            "INSERT INTO flash_sale_metrics VALUES (?,?,?,?,?,?)",
            new MetricsStatementBuilder(),
            JdbcExecutionOptions.builder()
                .withBatchSize(200).build(),
            new JdbcConnectionOptionsBuilder()
                .withUrl(DB_URL).build()));

    env.execute("flash-sale-revenue-10s");
  }
}
```

~45 lines of Java — watermarks, state, checkpointing, and serialization all hand-managed

A first look at watsonx.data integration for Flink



Real-time to AI ready – at every speed.

- **Unified control plane**

  Batch, streaming, replication, and observability — one place, not four.

- **A new engine, real performance**

  watsonx.data integration for Flink raises the ceiling on streaming throughput — coming soon.

- **Agentic, not just automated**

  Faster pipeline delivery without giving up governance or control.

Thank you. Let's take your questions.