

Orchestrating High-Throughput Pipelines: Airflow Meets Apache Beam

Ashwin Sampathkumar

Apache Beam | Airflow Summit 2026

Session Agenda (30 Minutes)

AIRFLOW SUMMIT | ONLINE RECONNECT

1. Intro to Apache Beam (Distributed Data & ML)

- Why Airflow workers OOM on heavy document parsing & embedding
- Beam's unified Batch + Streaming model
- Distributed ML via RunInference & Dynamic Work Rebalancing

⚡ Unified Batch + Streaming ML

2. Integrating Beam with Airflow DAGs

- Control Plane (Airflow) vs. Elastic Data Plane (Beam) BeamRunPythonPipelineOperator & Airflow 3 Assets
- Orchestrating zero-downtime Blue/Green model migrations

🌀 Zero-Slot Deferrable DAGs

3. Live Demo: AI Shopping Concierge

- HuggingFace Amazon Reviews 2023 streaming ingestion
- CoGroupByKey Catalog + Reviews join & SHA-256 upserts
- Live Black Friday price drop & 0-ms model upgrade in Vector DB

🛍️ Large Data Vector Sync



Why Do I need Apache Beam when I can do RAG with Airflow/Python

Direct Airflow RAG: With In-Worker Python

AIRFLOW SUMMIT | ONLINE RECONNECT

📄 **1. Small Corpus**
Up to 5,000 Docs / Day

→

🐍 **2. Airflow @task**
Single Worker (< 2GB RAM)

→

🧠 **3. Sync Embedding**
In-Task REST API Calls

→

🗄️ **4. Vector Upsert**
PgVectorHook / SQL Lookup

< 5,000

DOCS / DAY SWEET SPOT

Fits comfortably inside a single Celery or KubernetesPodOperator task without hitting execution timeouts.

< 2 GB

WORKER RAM FOOTPRINT

Simple Markdown/text splitting and small SQL lookups execute in-memory without OOM kills (SIGKILL -9).

1 DAG

ZERO EXTRA INFRA

Fastest path from prototype to production using standard Airflow @task decorators.

🎯 IDEAL DIRECT AIRFLOW RAG USE CASES:

- 📘 Internal Wiki / Notion Sync
- 📋 HR & Legal Policy Assistant
- 📰 Small Daily News Digest

The Scale Wall: When RAG Needs Apache Beam

AIRFLOW SUMMIT | ONLINE RECONNECT

🚨 WHY SINGLE WORKER AIRFLOW BREAKS AT SCALE: 💥 Worker Pod OOM | ⚠️ 1,024 Task Map DB Bottleneck | 🛑 Foundational Model HTTP 429 Quota Storms

📚 100K – 100M+ Docs (10 GB – Multi-TB)

Unified Batch + Streaming

Process multi-TB historical backfills and continuous real-time Kafka/PubSub events with one Beam pipeline.

⚖️ Heavy PDFs & Skewed Sizes

Liquid Sharding (Zero Stragglers)

Dataflow Dynamic Work Rebalancing splits straggler bundles mid-flight so a 500-page PDF never blocks workers.

🔗 Massive Distributed Joins

CoGroupByKey (10M SKUs × 500M Reviews)

Out-of-core distributed shuffle joins product catalogs with buyer reviews without exhausting worker RAM.

⚡ High QPM & GPU Inference

RunInference Dynamic Batching (32–128/RPC)

Stateful request batching + dedicated worker GPUs cut RPC overhead 50x and eliminate HTTP 429 errors.

🏢 ENTERPRISE BEAM + AIRFLOW WORKLOADS: 🛍️ E-Commerce Catalogs + Reviews • 📊 SEC Filings & Financial RAG • 🌐 Continuous Streaming Knowledge Bases



Intro to Apache Beam with Dataflow: Distributed Processing & ML at Scale

Apache Beam with Dataflow

Powerful Managed Platform + Rich Open-Source SDK



Apache Beam

RICH OPEN SOURCE SDK

Unified Programming Model
Single intuitive API for processing both batch and streaming pipelines seamlessly.

Portable & Open
Write once and run anywhere (Dataflow, Flink, Spark, or DirectRunner in Colab).

Multi-Language & ML Ecosystem
First-class SDKs for Python, Java, Go, SQL, and turnkey ML transforms (RunInference).



Dataflow

POWERFUL MANAGED PLATFORM

Fully Managed & Serverless
Zero infrastructure to provision; launches ephemerally from Airflow Deferrable Operators.

Liquid Sharding & Autoscaling
Dynamic work rebalancing splits straggler tasks mid-bundle and autoscales VMs/GPUs.

Enterprise RAG & AI Ready
Built-in observability, Dataflow Prime right-fitting, and native AI + DB I/O supports.

What is Apache Beam?

AIRFLOW SUMMIT | ONLINE RECONNECT

📦 1. PCollection
Bounded & Unbounded Data

Immutable distributed dataset. The exact same pipeline code processes a 50TB historical GCS backfill (Batch) or real-time Kafka/PubSub events (Streaming).

⚙️ 2. PTransform & DoFn
Distributed Compute & Joins

Parallel transforms: ParDo for custom section chunking, CoGroupByKey for massive distributed joins and GroupIntoBatches for DB writes.

🚀 3. Dataflow Runner
Liquid Sharding & Scale

Serverless execution: Dynamic Work Rebalancing automatically splits straggler tasks (e.g. a 500-page PDF vs 1KB JSON) and scales workers down to zero.

```
# Apache Beam Pipeline DAG: Composable, Type-Safe Distributed Stages
(p | 'ReadCatalog' >> beam.io.ReadFromText(catalog_gcs)     # 1. Parallel GCS Sharded Read
   | 'ChunkDocs'   >> beam.ParDo(BuildRetailChunksDoFn())   # 2. Section-Aware Chunking + SHA-256 Hash
   | 'BatchEmbed'  >> RunInference(vertex_model_handler)    # 3. Dynamic Batching (32-128 chunks/RPC)
   | 'UpsertDB'    >> beam.ParDo(VectorDBUpsertDoFn()))     # 4. Idempotent Upsert + Zombie Pruning
```

Apache Beam in Production: Alphabet & Industry Leaders













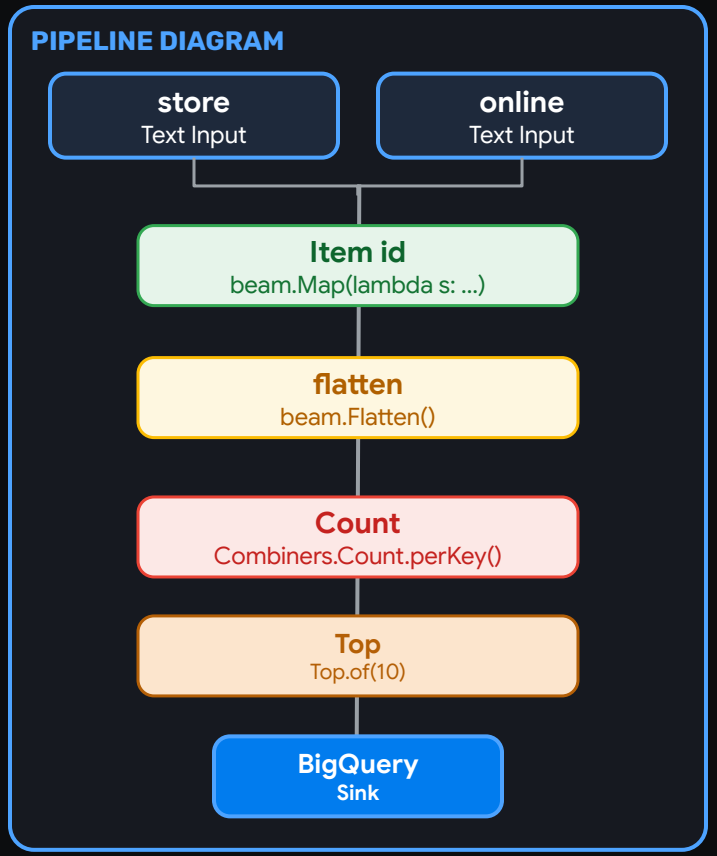




⚡ Powers Planet-Scale Workloads: 10M+ vCPUs Managed Daily | Exabytes of Batch & Streaming Data | Real-Time ML & RAG

Writing Code: Apache Beam Python SDK (Batch)

Composable PTransforms for Distributed Ingestion & Aggregation



pipeline.py

```python
…
# Read and extract Key-Value pairs
storeSales = p | beam.io.ReadFromText("purchases-store")
       | …
       | beam.Map(lambda s: (s.itemId, s.value))
onlineSales = p | beam.io.ReadFromText("purchases-online")
       | …
       | beam.Map(lambda s: (s.itemId, s.value))
# Merge, count per key, and extract top 10
topSales = (storeSales, onlineSales)
       | beam.Flatten()
       | beam.Combiners.Count.perKey()
       | beam.Combiners.Top.of(10, key=lambda x: x[1])
# Write output to BigQuery sink
topSales | beam.io.WriteToBigQuery(topSales)
```

Writing Code: Apache Beam Python SDK (Streaming)

Seamless Transition from Batch to Windowed Pub/Sub Streams (2 Lines Changed!)



pipeline.py

```python
# Read and extract Key-Value pairs
storeSales = p | beam.io.ReadFromPubSub("purchases-store")
       | …
       | beam.Map(lambda s: (s.itemId, s.value))
onlineSales = p | beam.io.ReadFromPubSub("purchases-online")
       | …
       | beam.Map(lambda s: (s.itemId, s.value))
# Merge, window, count, and extract top 10 items
topSales = (storeSales, onlineSales)
       | beam.Flatten()
       | beam.WindowInto(beam.window.FixedWindows(10))
       | beam.Combiners.Count.perKey()
       | beam.Combiners.Top.of(10, key=lambda x: x[1])
# Write output to BigQuery sink
topSales | beam.io.WriteToBigQuery(topSales)
```

Apache Beam with Dataflow ML Processing Chain

Three turnkey Apache Beam transforms that power production RAG & ML pipelines.

1. Enrichment

Add context to each element: product details, customer features, history.

Declarative lookups into DBs, BigQuery, AI Model Feature Store and more, with built-in rate limiting.

2. MLTransform

Prepare the data and generate embeddings with one interface.

The same artifacts are used in training and serving, so there is no skew.

3. RunInference

Run any model, local or remote.

Batching, model sharing, metrics, retries and model updates come built in.

You declare what each stage does. Beam with Dataflow handles ***Batching***, ***Autoscaling***, ***Retries*** and ***Metrics***.

Beam ML: Enrichment Transform

What it is

- Low-code declarative turnkey transforms for joining streaming data to data stores.
- **Out of the box support:** for most DBs and Models

Benefits

- **Declarative:** Define the what, not the how.
- **Rate limiting:** Automatic back-offs and autoscaler integration.

Use Cases

Real-time recommendations: Join users' live clickstream with historic data.

Anomaly detection: Join IoT telemetry data with historic information.

FSI Index building: Join instrument ticks with historic metrics in near real-time.

```
output = (p
   | "Fetch" >> beam.io.ReadFromPubSub(data_stream)
   ...
   | "Enrich" >> Enrichment(bigtable_handler)
```

Beam ML: MLTransform (Data Prep & Embeddings)

Overview & Benefits

Enables customers to easily prepare data for ML model training.

- **Declarative:** Reduce boilerplate and avoid managing underlying libraries.
- **Composability:** Chain multiple processing transforms with one interface.
- **Zero Skew:** Exact same artifacts across training and serving.

Use Cases

Feature Engineering: Preprocess text, numerical, and categorical features.

Training & Serving Parity: Guarantee exact same transforms, eliminating skew.

Production ML Pipelines: Chain vocabulary, normalization, and embeddings.

```
# Beam MLTransform pipeline example
p
| beam.Create(data)
| MLTransform(
    write_artifact_location=artifact_location)
     .with_transform(
       ComputeAndApplyVocabulary(columns=['x']))
```

Turnkey RAG in Beam: VertexAITextEmbeddings

```
# 1. Define chunker & Vertex AI embedding transforms
chunker = LangChainChunker(
    document_field='content', text_splitter=splitter)

vertex_embedder = VertexAITextEmbeddings(
    model_name='text-embedding-005',
    columns=['content'], project=project)

# 2. Construct Dataflow ML RAG pipeline
with beam.Pipeline(options=pipeline_options) as p:
  (p
   | 'ReadCatalog' >> beam.Create(PRODUCTS_DATA)
   | 'ChunkDocs'   >> chunker
   | 'EmbedChunks' >> MLTransform(
       write_artifact_location=gcs_artifact_uri)
       .with_transform(vertex_embedder)
   | 'WriteAlloyDB' >> VectorDatabaseWriteTransform(
       AlloyDBVectorWriterConfig(connection_config))
  )
```

Document Chunking

LangChainChunker: Splits raw document content into chunked text segments prior to vector processing.

Vector Embeddings

VertexAITextEmbeddings: Leverages **text-embedding-005** chained inside Beam's composable MLTransform.

Vector Store Ingestion

Streams processed vector embeddings seamlessly into any vector store using the Beam IOs

RAG Support: Knowledge Ingestion & Real-Time Streaming

Overview & Capabilities

- **Built-in Transforms:** Single line-of-code to integrate embedding generation directly into pipeline.
- **Flexible Models:** Choice of Vertex AI or BYOM for embedding creation.
- **Vector Store Support:** Native writes to many Vetor DBs including, Bigtable, Spanner, Postgres & more.

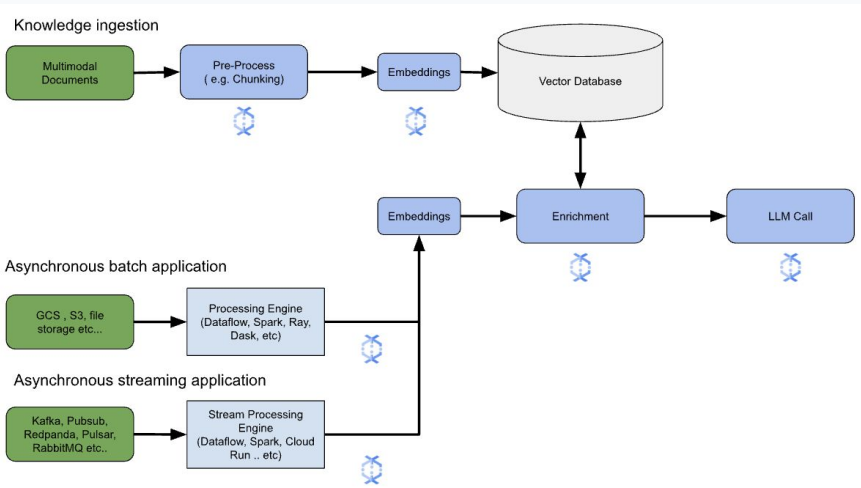
Key Benefits

- **Simplified Preprocessing:** Flexible chunking with powerful Dataflow processing.
- **Lightweight Models:** Leverage fast models (e.g. Gemma) for local embeddings.
- **Unified Pipeline:** Same code for knowledge ingestion and async streaming serving.
- **Zero Lock-in:** Easily switch vector databases or ML models without code changes.

```
MLTransform(write_artifact_location=gcs_uri)

.with_transform(VertexAITextEmbeddings('text-embedding-005')
)
```

RAG Pipeline Architecture



Beam ML: RunInference Transform

What is RunInference?

Enables Dataflow pipelines to seamlessly execute and manage ML model inference with native framework support.

Key Benefits

- **Reduce Boilerplate Code:** Simplifies pipeline architecture and model loading logic.
- **Local & Remote Inference:** Works with PyTorch, scikit-learn, TensorFlow, and Vertex AI.
- **Model Hub Integrations:** Out-of-the-box support for TensorHub and Hugging Face.
- **Multi-Model Pipelines:** Chain or run multiple inference models within a single pipeline.

pipeline.py

```
from apache_beam.ml.inference.base import
RunInference

with beam.Pipeline(options=pipeline_options) as p:
  (
     p
     | 'Read'  >> beam.io.ReadFromPubSub(topic=topic)
     | 'Prep'  >> beam.Map(preprocess_chunks)
     | 'Embed' >> RunInference(model_handler)
     | 'Write' >> beam.Map(format_pgvector_rows)
  )
```

RunInference Features — Framework Support

Out-of-the-box ModelHandlers for leading ML frameworks and custom GPU/API endpoints

PyTorch

🔶 TensorFlow & TFX
SavedModel & TensorHub
ModelHandler

🤗 Hugging Face &
Vertex AI Endpoints
Transformers &
Embeddings

dmlc
XGBoost

scikit
learn

➕ Custom ModelHandler
Bring Any Framework
Local GPU or Remote RPC



Integrate Apache Beam with Airflow DAGs

Integrating Beam with Airflow 3 (Cloud Composer 3)

AIRFLOW SUMMIT | ONLINE RECONNECT

1. Airflow 3 Asset & HF Delta Sync

Streams HuggingFace deltas to GCS staging & reads active model Variable.

2. Beam Python Pipeline Launch

Launches retail_rag_pipeline.py + requirements.txt on Dataflow via Triggerer.

3. Recall & Parity Quality Gate

Verifies SKU count parity, DLQ < 0.1% & Recall@5 across versions in Vector DB.

4. Atomic View Cutover (0ms)

Flips active_retail_concierge_rag view & updates active model Variable.

```
launch_rag_upsert = BeamRunPythonPipelineOperator(
    task_id='launch_dataflow_rag_upsert', runner='DataflowRunner', deferrable=True,
    py_file='/home/airflow/gcs/dags/pipeline/retail_rag_pipeline.py',
    pipeline_options={
        'project': PROJECT_ID, 'region': 'us-central1',
        'requirements_file': '/home/airflow/gcs/dags/pipeline/requirements.txt',
        'embedding_version': "{{ var.value.get('active_embedding_model', 'text-embedding-004') }}"})
ingest_hf_to_gcs >> launch_rag_upsert >> verify_recall_gate >> promote_vector_db_serving_view
```

Live Demo: AI Shopping Concierge (Amazon Catalog + Reviews)

Demo Architecture: AI Shopping Concierge

AIRFLOW SUMMIT | ONLINE RECONNECT

🤗 1. HuggingFace Stream Amazon Reviews 2023

- McAuley-Lab/Amazon-Reviews-2023
- raw_meta_Electronics (Title, Price, Specs, Features)
- raw_review_Electronics (Verified Buyer Reviews)

➔

🔥 2. Beam on Dataflow CoGroupByKey + Batch ML

- Joins SKU Catalog + Top 5 Helpful Verified Reviews by parent_asin
- Chunks: OVERVIEW, TECH_SPECS & CUSTOMER_REVIEWS
- Batched Vertex AI Embeddings

➔

🗄️ 3. BigQuery Vector + SQL Search

- Combines SQL predicates (in_stock = TRUE AND price <= $150 AND rating >= 4.2) with HNSW cosine similarity
- Served via active_retail_concierge_rag view

🌀 Cloud Composer 3: retail-concierge-composer | Image: composer-3-airflow-3.3.1-build.5

- Bucket: `GCS Bucket`
  - ├─ `retail_concierge_dags.py` (DAG 1: Incremental Sync | DAG 2: Zero-Downtime Embedding Migration)
  - └─ `pipeline/retail_rag_pipeline.py` + `pipeline/requirements.txt` (Dataflow CoGroupByKey + RunInference + Upsert)

Key Takeaways: Production RAG with Airflow & Beam

AIRFLOW SUMMIT | ONLINE RECONNECT

🌀 1. Orchestrate in Airflow (Control Plane)

🔥 2. Beam Data Plane for Elastic Distributed ML

🗄️ 3. Keep Vector DB Serving Freshness & Zero Downtime Upgrades



Questions?

Ashwin Sampathkumar

Demo Code Github link