

MATRIX™ System

Montai AtTRIBUTE eXpander

Scalable Compound Property Generation

750 Million Compounds | Under 2 Hours

Processed end to end on a single coordinated run

Thomas George Thomas | Rody Arantes

BioIT World 2026

Montai Therapeutics | A Flagship Pioneering Company

The Challenge | Why MATRIX System

Generating compound properties at scale was the bottleneck of drug discovery

INPUTS WE PROCESS

Multi-source compound libraries

Vendor Catalogs

Commercial compound libraries

Public Sources

Open chemistry databases

Private Sources

Internal proprietary collections

BEFORE MATRIX System

2-3 days/property = ~100 days

to generate 40 properties for 115 million compounds

- Data scientists processed data locally in Jupyter notebooks on their laptops
- Some properties skipped: compute time up to 6 months
- Stakeholders waited weeks for usable data
- Long turnaround slowed analysis and decisions
- No incremental path: every new source restarted the clock

Decision Velocity | Weeks vs Hours

What MATRIX System changes for the scientists who actually need the data

TWO EVERYDAY TRIGGERS

1

Add new compounds

Vendor catalog or private library lands in the data lake

2

Add a new property

Scientist needs a new RDKit, ML or custom attribute

BEFORE

Decisions delayed for weeks

- Days of implementation work
- Days of property generation
- Stakeholders waiting, blocked



WITH MATRIX System

Decisions enabled in hours

- Minutes of implementation work
- Under 2 hours of generation
- Stakeholders unblocked, daily

Property Types | What MATRIX System Generates

RDKit, custom code, and ML inference, all riding the same pipeline

280

ATTRIBUTES PER COMPOUND

all generated in a single coordinated run

Computed

RDKit + Custom Code

Processed SMILES

Carbon Counts

Morgan Fingerprint

Murcko Scaffolds

Filters & Flags

Substructure Detection

Rd Filters

Substructure Flags

Camptothecin Substructure

Transition Metals

Predicted

ML Inference

ML Predictions

NP Likeness

Diversity Clusters

Tanimoto to Anthromolecule

Our Mission | Measurable Outcomes

From weeks of bottleneck to an automated, modular platform

OBJECTIVE



Accelerate

Attribute generation and delivery



Scale

Build a robust, distributed pipeline



Enable

Faster insights and decisions

KEY OUTCOMES

Streamlined

End-to-end automation

Modular

Scalable architecture

Resilient

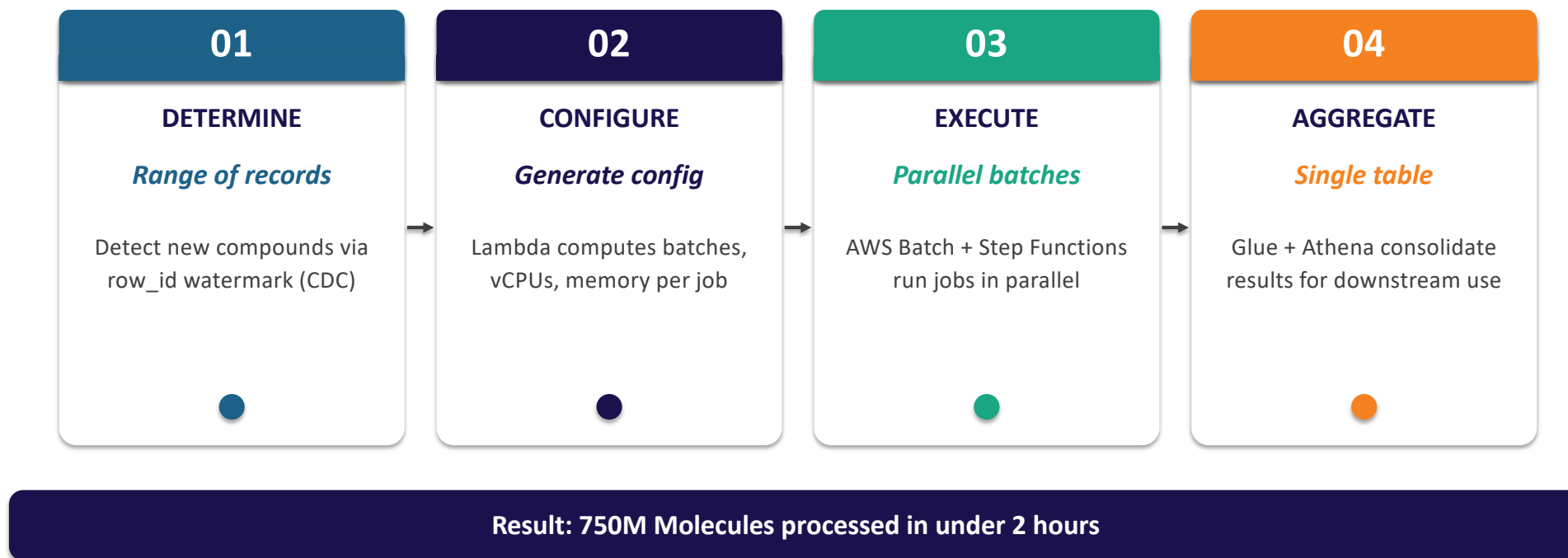
Fault tolerant pipeline

Fast

Property generation under 2 hours

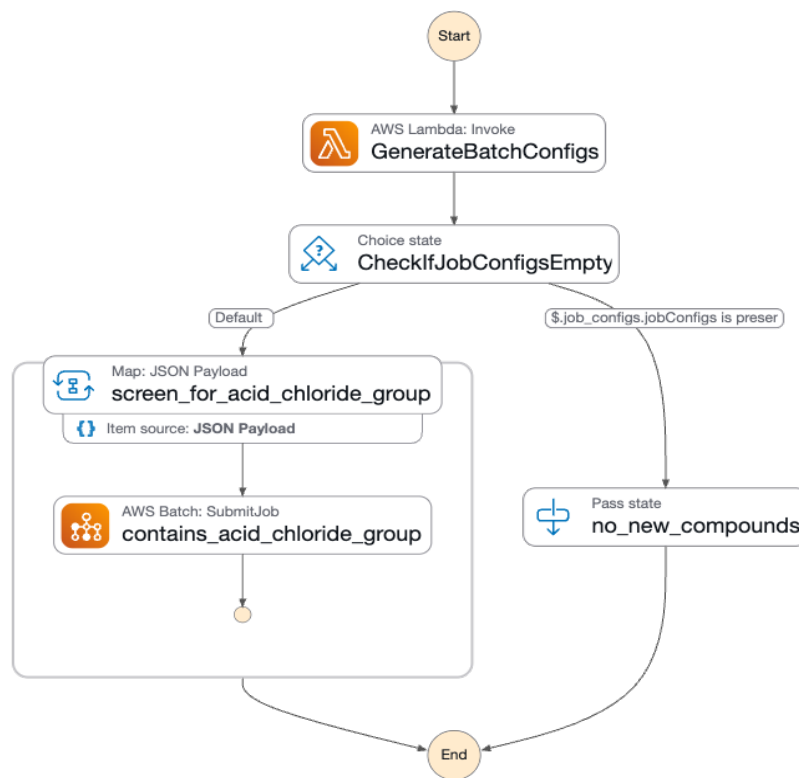
MATRIX System | How It Works

Four orchestrated stages convert raw SMILES into 280 enriched properties



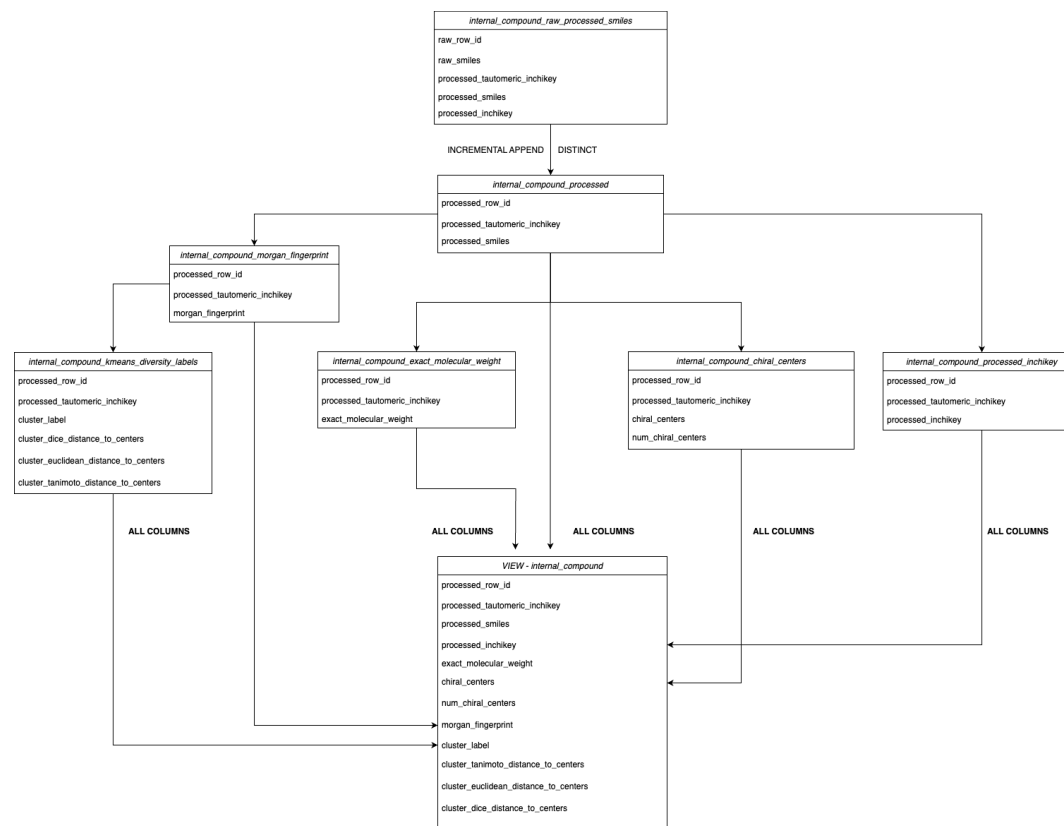
MATRIX System | Architecture

AWS Step Functions orchestrate Lambda, Batch, and Map states across the property-generation workflow



Sample | Data Model

Five tables, one view: how MATRIX System organizes generated properties



dbt | Incremental Modeling & Lineage

SQL-based transformations with first-class lineage tracking and incremental builds

Incremental Models

Process only new SMILES on each run, never recompute history

Lineage Graph

Every column traces back to a raw source for full auditability

Tested by Default

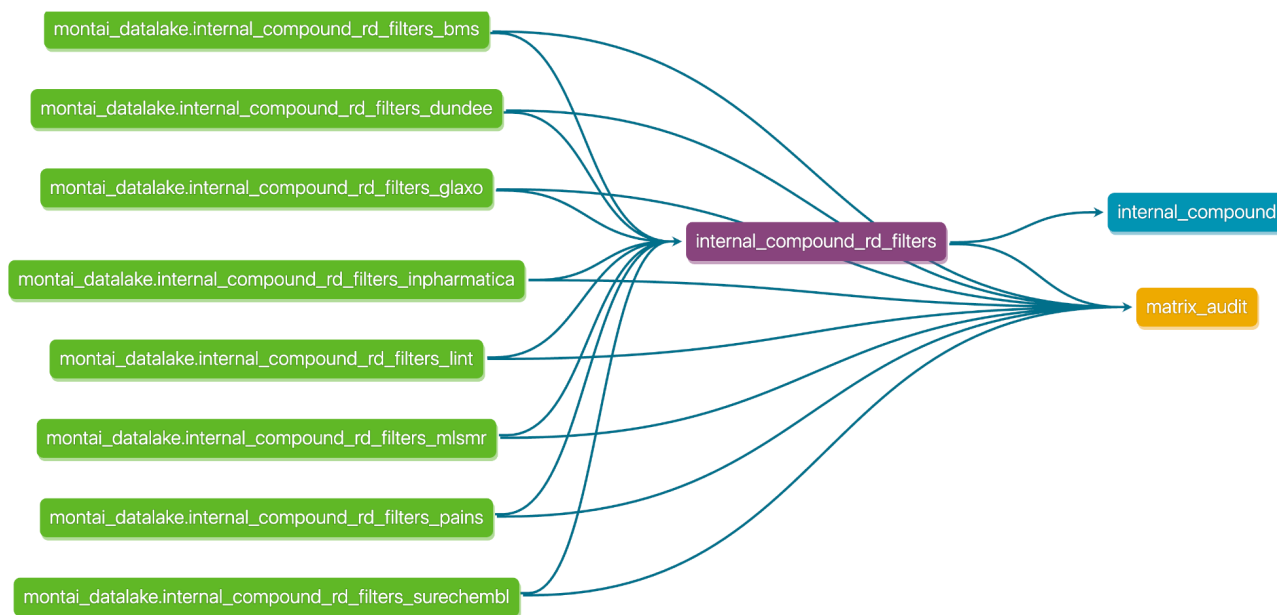
Schema and data tests run on every model build

Modular SQL

Each property is its own composable model

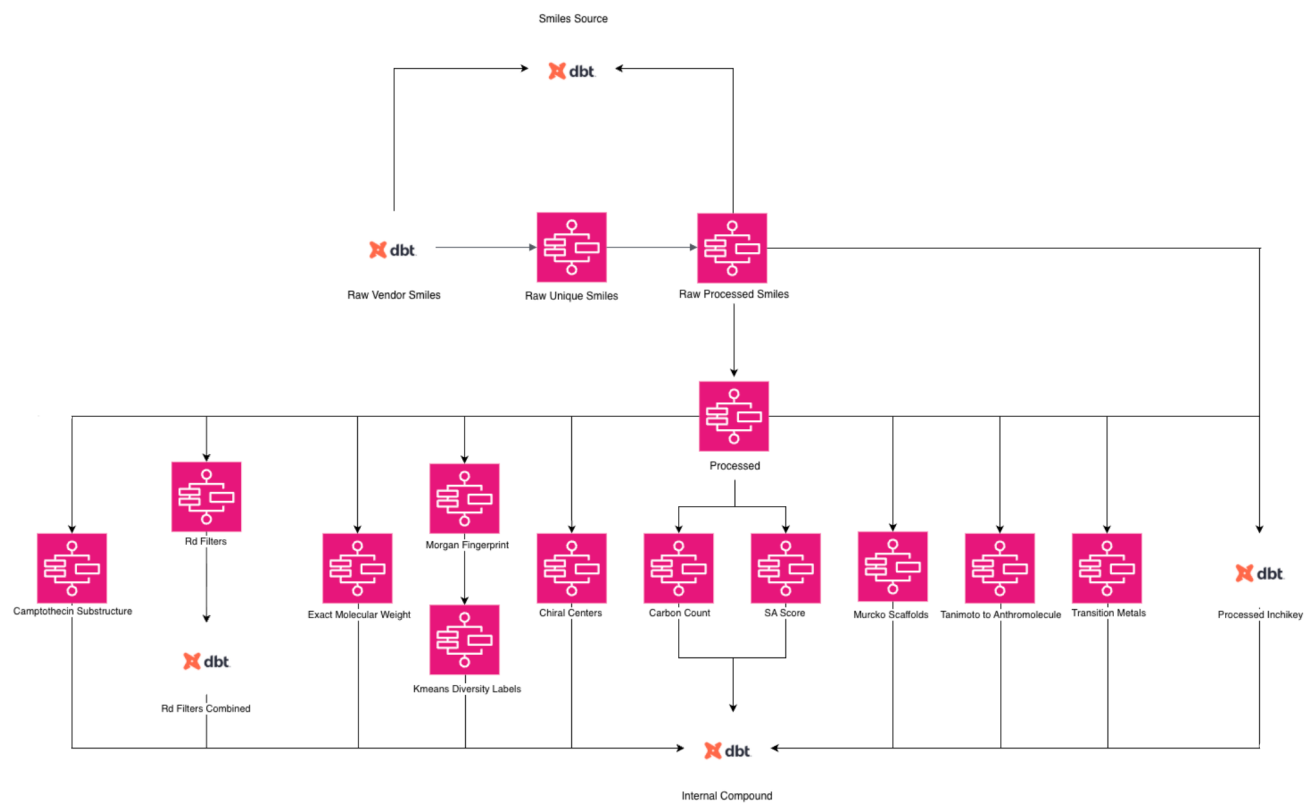
Incremental Modeling | Data Lineage

dbt data docs render the lineage graph: every vendor rd_filters source flows into matrix_audit and the unified internal_compound asset



Pipeline | Step Functions + dbt Models

Step Functions run the property generation jobs; Dagster updates the dbt models that compose the final Internal Compound table



Dagster | Pipeline Orchestration

Asset-aware orchestration that treats data products as first-class citizens



Asset Graph

Visualize every table, model, and Step Function as a node in one DAG



Scheduled & Event-Driven

Triggers on S3 changes, schedule, or upstream asset materialization



Built-in Observability

Run history, logs, and lineage in a single UI

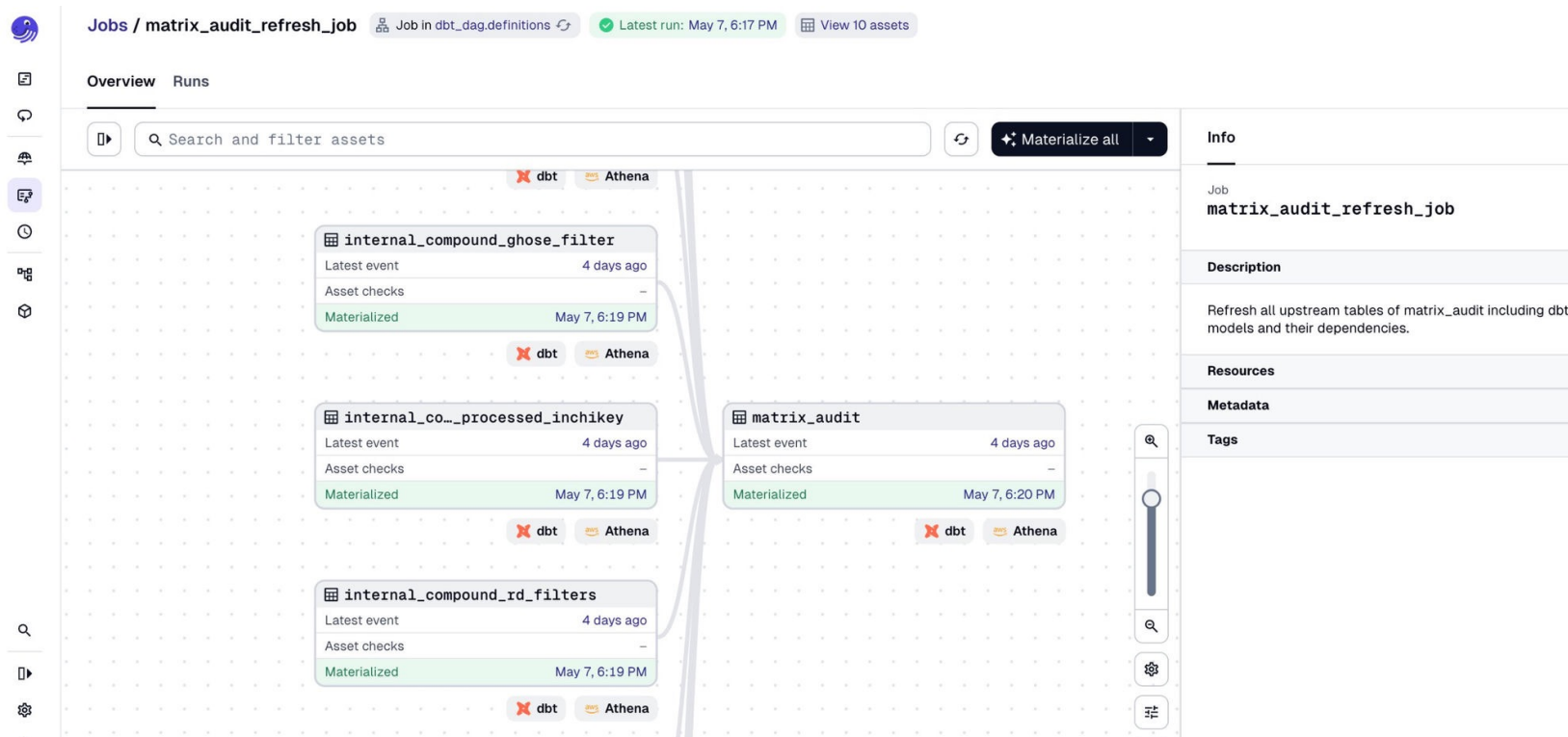


Plugs Into Everything

Native integrations with dbt, AWS, and the Modern Data Stack

Dagster UI | Live Orchestration

Asset graph for `matrix_audit_refresh_job` running in production



Change Data Capture | Compute Only What Changed

Incremental append on row_id eliminates redundant recomputation

WHY CDC

Avoid redundant recomputation

- New sources triggered full recompute of all properties
- Significant compute overhead and delays
- Wasted infrastructure cost on duplicated work

CDC minimizes redundant work and optimizes compute resources

STRATEGY: INCREMENTAL APPEND

row_id watermark

01

Assign

unique row_id to every deduplicated compound

02

Compare

latest row_id between source and target tables

03

Append

only new unique compounds on subsequent runs

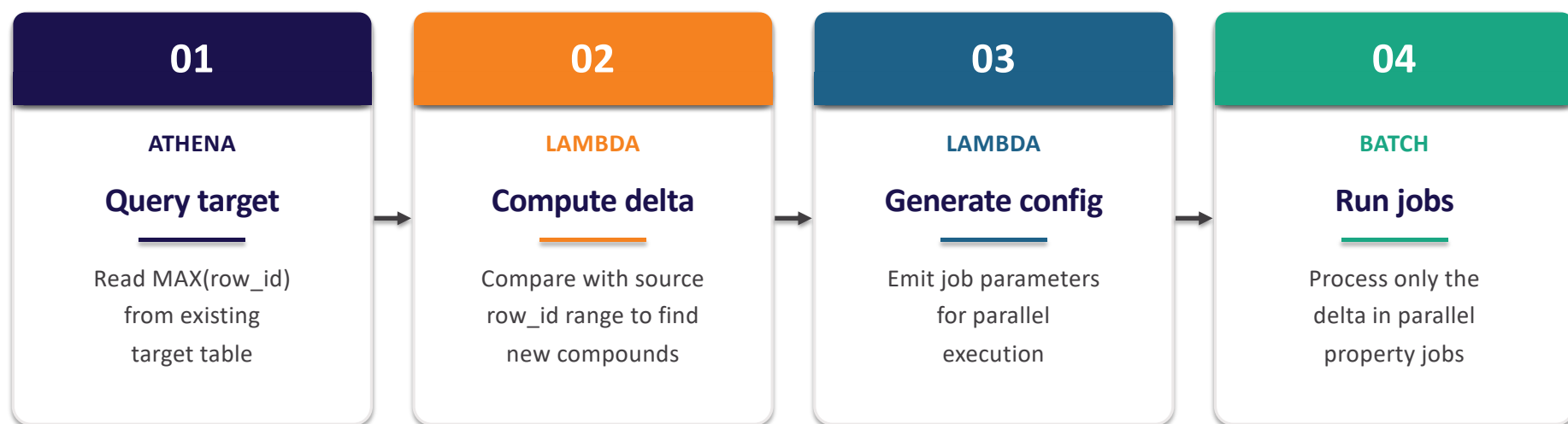
04

Detect

and process new compounds for property generation

CDC | Process Flow

Four steps from existing target table to a parallel batch run, processing only the new rows



BATCH CONFIG PAYLOAD

OFFSET

BATCH SIZE

vCPUs

MEMORY

Result: zero recomputation of historical compounds, only the delta runs

CDC | Modular & Cost Efficient

Independent workflows for each property, recomputed only when needed



Modular Approach: separate workflows for metadata generation, efficient when re-running select properties

Storage Layer | Athena & Glue

Hard limits met head-on with disciplined file layout and ingestion patterns

LIMITATIONS

What we hit

- **Athena partition limit: 100**
Hard ceiling on partitions per query
- **Small files**
Cause major performance overhead in Athena
- **MSCK REPAIR / Crawler**
Required after every partition ingestion
- **Trino / PrestoDB constraints**
Architectural choices, not bugs

OPTIMIZATIONS

How we won

- **No partitioning at ingest**
Avoid partition limit and ingest delays
- **Append with row range filenames**
Prevents collisions and tracks lineage
- **10M rows per file ~128MB**
Sweet spot for Athena scan and parallelism
- **Skip the crawler after first run**
Metadata stable, no re-scan required

AWS Batch | Issues & Optimizations

Three production bottlenecks, three engineered solutions

! 50 TPS Submission Limit

Global cap on AWS Batch job submissions per second

+ FIX

Step Function wait state with random jitter
& AWS Array Jobs scaling to 10,000 children/sec

! Out of Memory + Non-Serializable

Returning RDKit mol objects as columns

+ FIX

Compute properties inside the function,
return primitive results only

! Slow Pandas Writes to S3

Pandas write was the dominant bottleneck

+ FIX

Redesigned writer using PyArrow,
significantly faster S3 writes

Best Practices | What to Take Home

Five rules of thumb for running RDKit-scale workloads on AWS Batch

- 1 Scale wide, not deep**
50,000 vCPU compute environments scale better
- 2 Monitor memory per job**
20 to 50 GB range covers nearly all property workloads
- 3 Match instance to property**
Compute-optimized for fingerprints, general-purpose for filters
- 4 Tag every job**
Auditability is a Day-0 design choice, not Day-N debt
- 5 Optimize Map for 50 TPS**
Step Function jitter + Array Jobs together break the global cap

ACHIEVEMENTS

From 3 Months to 2 Hours

MATRIX System delivered scale, speed, and resilience that legacy workflows could not approach

280

Properties

generated per compound

750M

SMILES

processed end to end

2 hrs

Runtime

from raw to output

6mo to 1hr

Acceleration

the headline number

MATRIX System

Property generation, transformed.

From weeks bottleneck to a 2-hour automated platform

Thank you

Questions?

Thomas George Thomas

PRESENTER

tthomas@montai.com

Rody Arantes

PRESENTER

rarantes@montai.com

Montai Therapeutics | A Flagship Pioneering Company