# How to slash your dbt data pipeline costs with DuckDB

# Table of contents

## 0.1 Introduction

- Name: Joseph Machado
- Work exp: 10 years in Data engineering & platform
- Blog: Startdataengineering

# Chapter 1

# DuckDB + Ephemeral VMs = dirt cheap data processing.

## 1.1 Serverless workflows

- Serverless dbt + duckdb workflow: Start VM -> Pull data into VM -> Process data -> Dump data into destination -> Spin down VM.

| Start VM | → | Pull data into VM | → | Process data | → | Dump data into |
|----------|---|-------------------|---|--------------|---|----------------|

## 1.2 Ephemeral servers are inexpensive

- Most cloud providers allow you to rent virtual machines charged by the hour (e.g., EC2).

- If you are not under tight time constraints, you can bid spot VMS at a much lower rate than on-demand VMS.

- Pulling data into EC2 is cheap.

- Fast data transfer speed (especially with cpp optimized duckdb extensions).

## 1.3 Real project cost computation

- Real data infra (**ref**). Details:

1. **Number of jobs**: 800 independent data pipelines with about 80 source systems
2. **Number of files**: ~400
3. **Average size per file**: ~1GB

- Objective: Reducing overall costs to between \$3,000 and \$5,000 per month, processing about 400GB per day across 800 jobs.
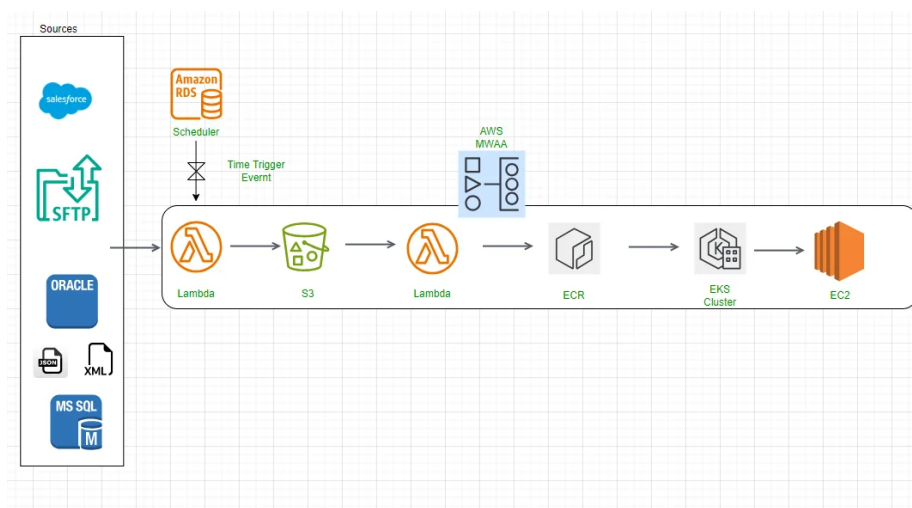
## 1.4



Figure 1.1: Architecture

## 1.5 Cost projection with serverless duckdb + dbt

- Execution time (sample): A non-optimized, expensive pipeline (fact-fact join, joining all dimensions) takes about 2 minutes to run.

  1. Input: ~2GB
  2. RAM: 8GB Cores: 2
  3. Output: ~13GB

- Computing the cost for 800 jobs, assuming one file (500MB) on avg per job ~ 700 USD.

- Data transfer costs & IP costs are negligible at this scale.

## 1.6 Save time and money by keeping the feedback loop short

- With standard dist. Data proc systems, you will need to wait a while (set up cluster, start job, etc.) before you see the results.

- With dbt + duckdb, you can run locally without a complex setup and see results instantly (check out the buenavista package for viewing results when a dbt pipeline is running in duckdb).

# Chapter 2

# Migrate one (or a few pipelines) at a time.

## 2.1 Your first migration will involve some work

- Setting up infra to run, EC2/AWS Lambda/ECS, k8s, etc instead of connecting to a db/engine.

- **Code changes**: Changing db-specific functions, e.g., MERGE and date functions.

- **Error handling**: If you are processing in memory (without persisting intermediate datasets) and your data processing fails, you will have to re-run the entire pipeline.

## 2.2

- **Permissions** are set at the service level, not the USER/ROLE level, as in most data processing systems.

- **Logging system metrics**: most cloud VMs have this setup, e.g., AWS Cloudwatch.

## 2.3 With a template to migrate, the rest of the migration will be more straightforward

- With the infra in place, the migration will be simple.

- Migrate at off-peak times. For example, if you are in e-commerce, don't migrate during Thanksgiving, or if you are in finance, don't migrate at

the end of fiscal year reporting.

## 2.4

- Data pipeline migrations are tricky!
    1. **Code**:  Ensure the code has unit/integration tests (not just DQ checks)
    2. **Data**:  Ensure data has sufficient DQ checks and validate data between old and new systems for a defined period before switching over.

## 2.5    Watch out for pipelines that fully reprocess huge tables.

- If you have pipelines that involve aggregating historical data (e.g., for anomaly detection, handling late-arriving events), you need to handle them. Use one of the methods below:

- **Aggregate past n periods of data** instead of reaggregating the entire data set. For example, if your pipeline processes sales data that comes in every day, instead of reprocessing the past n years' worth of data, consider reprocessing the past 3/6 months of data (depending on later arriving data for your business use case).

## 2.6

- Use **specialized data structures** such as date lists to optimize large aggregation operations.

- **Store aggregated data** in a separate location. For example, if you count rows in a dataset every run, store the counts in a separate table so you don't have to recompute them for historical data each time.

# Chapter 3

# Watch out for integration points!

## 3.1 Data permissions are defined at the service level.

- Most data processing systems have comprehensive data access controls crucial for data governance.

- With dbt + duckdb, we must handle data access at a service (E.g., AWS Lambda can access a specific S3 bucket, etc.) level.

- The inability to handle data permissions at a row level (like what Snowflake can offer) can sometimes be a deal breaker (PII, Sensitive info, etc).

## 3.2 Some popular tools don't officially support DuckDB yet

1. elementary (GH issue)

2. greatexpectations

## 3.3 You need to do some work to dump data into Vendor warehouses

- With dbt, you usually create the output dataset as well.

- When using duckdb for processing, you must dump the data into a destination system (cloud store or another database).

- You will need a system to create the output tables/dump to the cloud store via extensions or code.

## 3.4   Questions

## 3.5   Excited About

1. Motherduck makes serverless infra for proc and analytics simple
2. Ibis dataframe with DuckDB backend
3. Tight Python integrations with Client API
4. dbt duckdb
5. Buena vista library for accessing duckdb tables when dbt is running