DATA+AI
SUMMIT 2022

US Healthcare Price Transparency in Coverage

Utilizing Databricks and Delta Lake

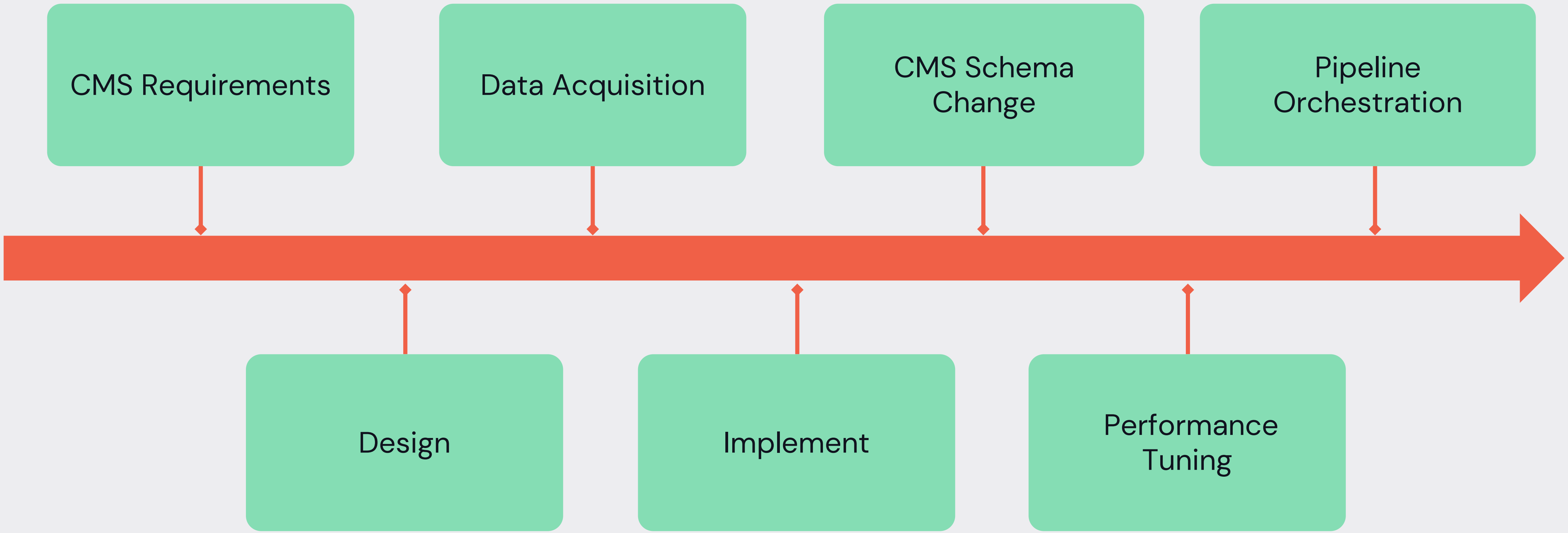
Ross Silberquit
IT Principal Engineer, Cigna

Narayanan Hariharasubramanian
IT Principal Director, Cigna

ORGANIZED BY databricks

Journey

A tale of Linear and Horizontal Progression



Who are we?

About us

- A global health services company with the mission of improving the health, well-being, and peace of mind of those we serve by making health care simple, affordable, and predictable.
  - easy to get care – letting you choose how, when, and where you want it.
  - A more affordable health care by partnering with providers who provide quality, cost-effective care.
  - A comprehensive health care coverage with “no surprises.”

Cigna®

cigna.com/about-us/

What is Price Transparency?

Phase One - Machine Readable Files (MRF)

Requirements

- CMS mandated[1] all health insurance payers publicly post MRF files with contracted provider rates for all procedure codes.
- 3 Types of files:
  - In-Network Contracted Rates
  - Out-of Network Allowed Amounts
  - Table Of Contents
- MRF schema set by CMS[2].

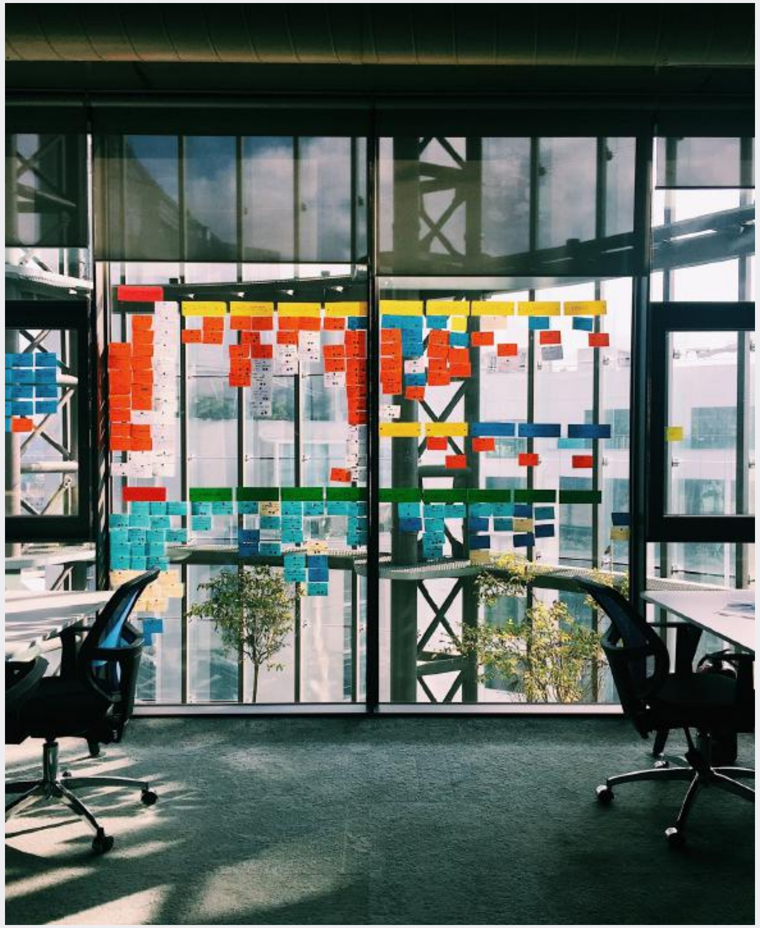


Photo by İrfan Simsar on Unsplash

1. https://www.govinfo.gov/app/details/FR-2020-11-12/2020-24591 2. https://github.com/CMSgov/price-transparency-guide

Price Transparency

Phase One - Machine Readable Files (MRF)

Requirements – contd...

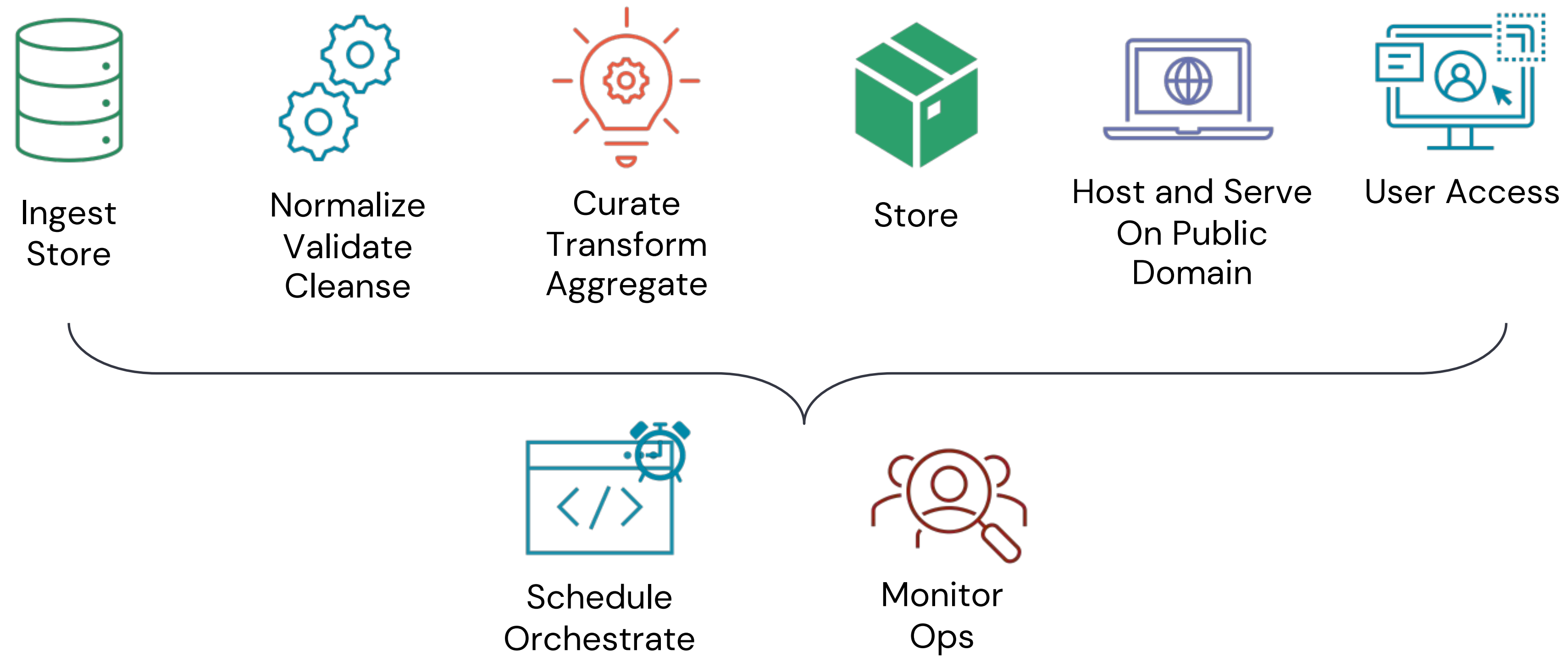
- JSON files, one for each plan[1]
- Generated monthly
- Hosted on public domain
- Include data from partners


```
      "negotiated_rates": [{
        "provider_groups": [{
          "npi": [1111111111, 2222222222, 3333333333, 4444444444,
          "tin":{
            "type": "ein",
            "value": "11-1111111"
          }
        },{
          "npi": [1111111111, 2222222222, 3333333333, 4444444444,
          "tin":{
            "type": "ein",
            "value": "22-2222222"
          }
        }],
        "negotiated_prices": [{
          "negotiated_type": "negotiated",
          "negotiated_rate": 123.45,
          "expiration_date": "2022-01-01",
          "service_code": ["18", "19", "11"],
          "billing_class": "professional",
          "billing_code_modifier": ["AS"]
        },{
```


1. https://github.com/CMSgov/price-transparency-guide/blob/master/examples/in-network-rates/in-network-rates-fee-for-service-single-plan-sample.json

10,000 Foot View

Data Flow



The Data

The power of scalability

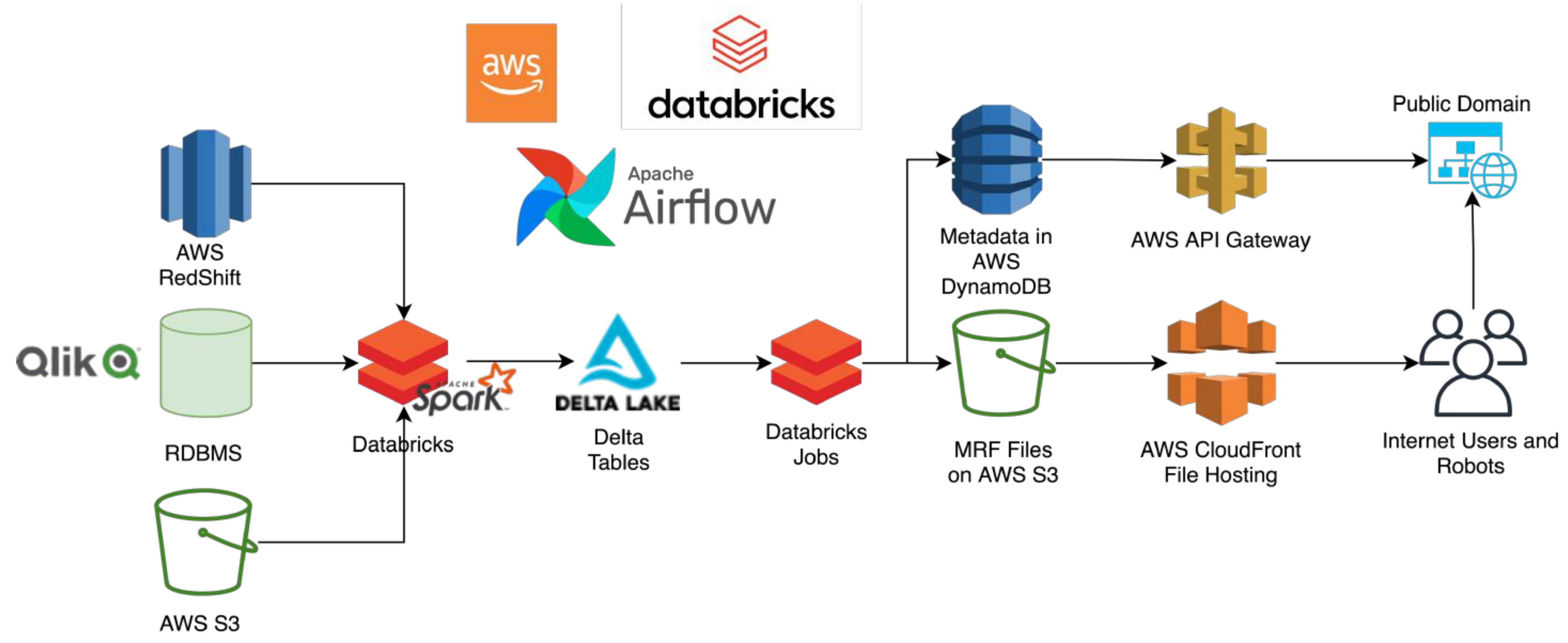
Photo by benjamin lehman on Unsplash

- Thousands of Providers
- Tens of Thousands of Billing Codes (CPT | ICD 10)
- Hundreds of Plans
- Billions of records
- More than 60TB of Data

Let’s dig in

Components

Using Databricks on AWS



Databricks Components

Spark to the rescue

Code

- Scala / Python / Bash
- Notebooks
- Packaged whl's & jar's

CICD / Infrastructure

- REST API and Terraform
  - Cluster orchestration
  - Policy enforcement
  - Job runs (via Airflow)
- Docker image with Databricks CLIs for integration testing
- Automated customer onboarding and provisioning.

Apache Spark

- Dataset/frame APIs (Scala)
- pySpark
- Used for :
  - Normalization
  - Validation
  - Cleansing
  - Aggregation
  - Joins

Solution Architecture

Automation with Databricks API

databricks

/clusters/*

- Nitro instances for HIPAA compliance
- Instance types for storage and memory optimized nodes from i3en.xlarge to 24xlarge and r5dn.xlarge to 8xlarge
- Auto-provisioned instance pools
- Latest runtimes 10.x

/policies/clusters/*

- Glue catalog integration
- S3 based logging destination
- Init scripts to auto-tag EC2 instances for FinOps
- T-Shirt sizing for wide-range of capacity requirements.

/jobs/*

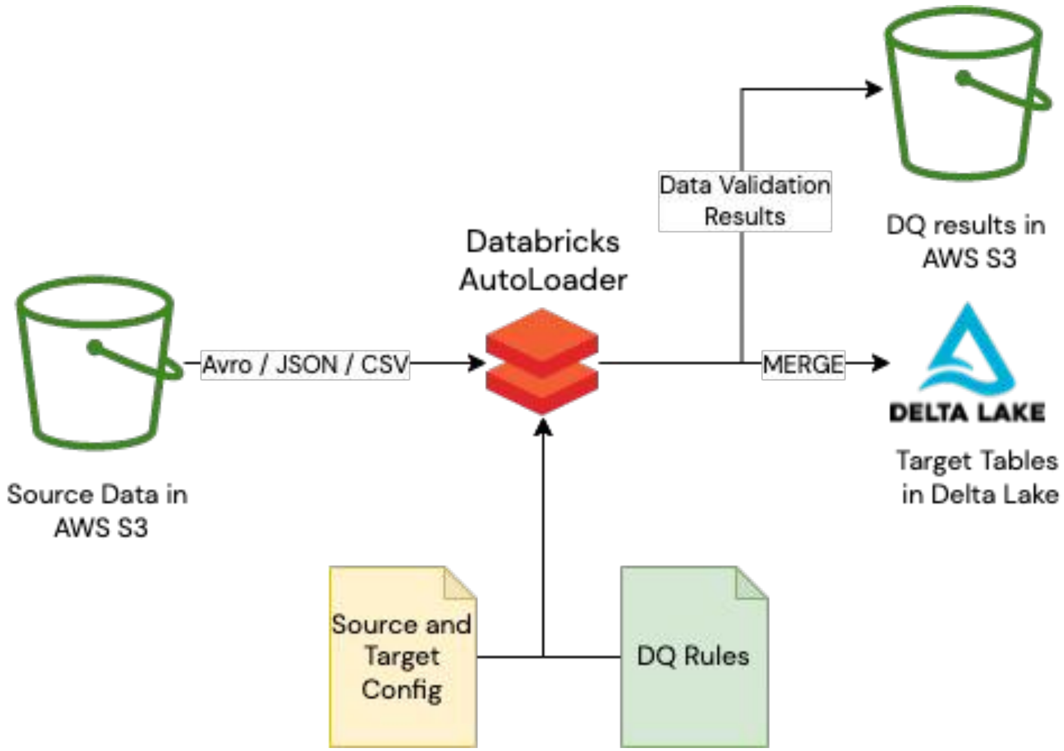
- Python WHL
- Scala JAR
- Notebooks
- Service Principals and Managed cluster and Job permissions

Delta Lake

Reusable Notebooks

Based on Databricks Autoloader

- Autoloader in Directory Listing Mode
- Data Validation Rules and Reporting
- DeltaLake MERGE to target
- Current and History View of Data
- Filters and Additional Columns using in-built UDFs
- Registers tables to Glue Catalog



Data Storage and Processing

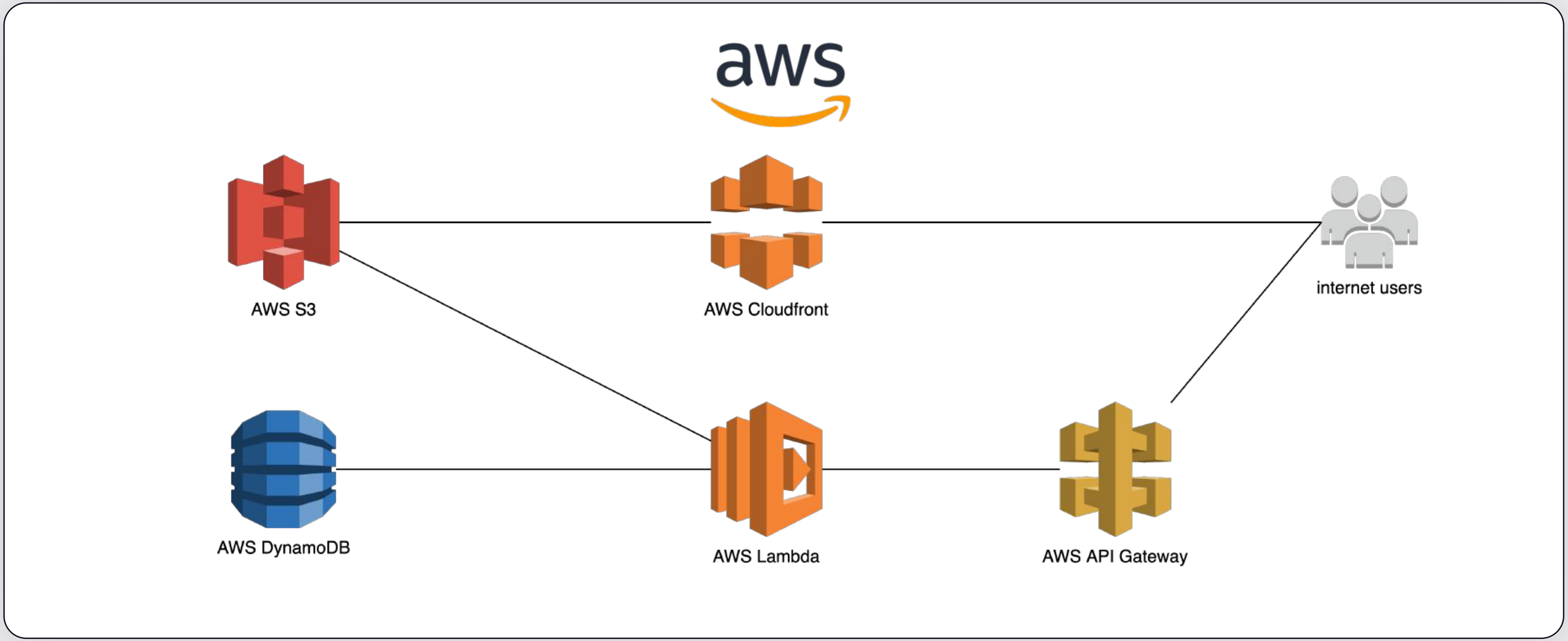
Why Delta Lake?

- Scales well for billions of records
- MERGE capabilities to maintain current view
- Great for sharing data with other teams
- Enables streaming use cases to process CDC data from sources and from DeltaLake using Change Data Feed
- Intermediary results of aggregations and joins improving performance
- Manual & Auto-optimization, Z-ORDERing for improved performance
- Makes reporting and metrics faster with its metadata capabilities

Serving Layer

Solution Architecture

AWS & Other



Performance Tuning

Optimization

With Apache Spark & Delta Lake

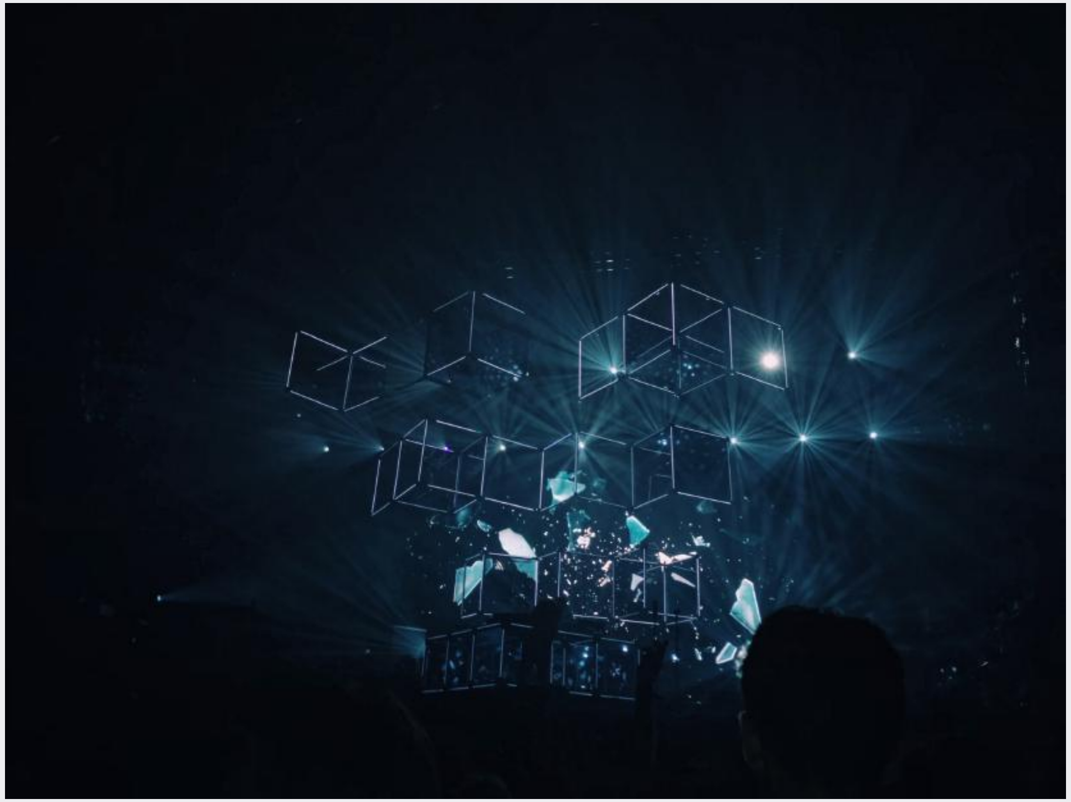


Photo by fabio on Unsplash

Techniques

- Pre-partitioning sources on low-cardinality fields
- Z-Ordering
- Delta Lake – Optimize
- Window Functions
- Aggregators
- Choosing the right EC2 instance types

Optimization

General Best Practices

What we tried

- – using all Datasets (strictly typed)
- – more shuffle partitions (smaller task chunks to avoid OOM)
- – removing distincts (replaced with window functions to avoid a second shuffle)
- – using delta format for initial write out (delta cache boost)
- – breaking the job up into smallest possible steps

Photo by Kolleen Gladden on Unsplash

Optimization

Aggregators

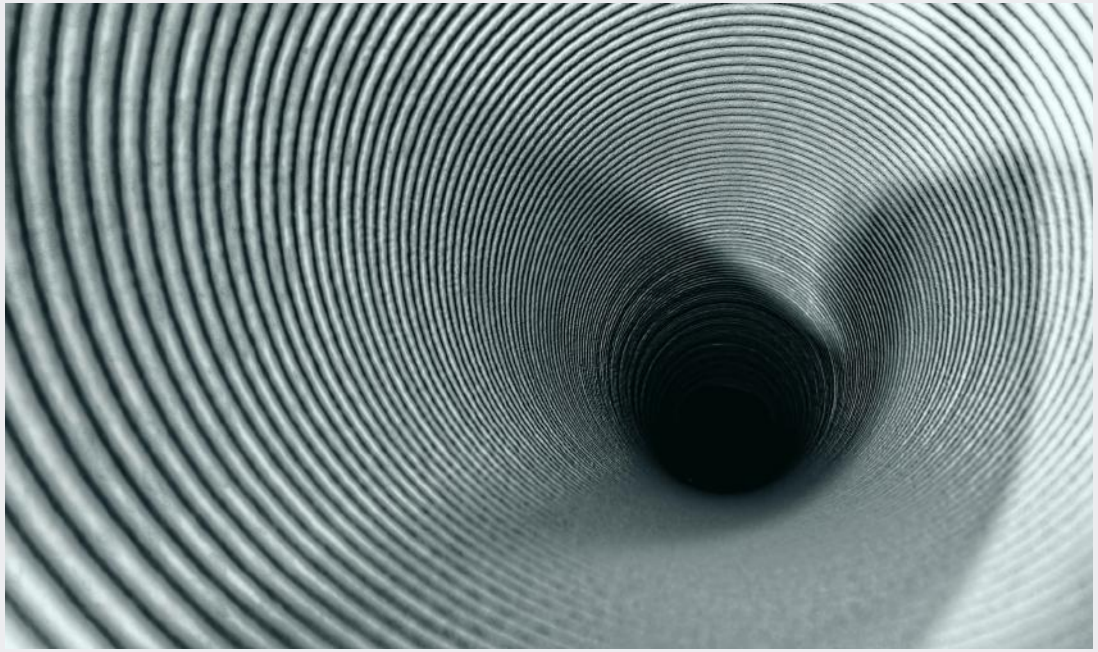


Photo by Ricardo Gomez Angel on Unsplash

Aggregator[-IN, BUF, OUT]

- ***IN*** - The input type for the aggregation.
- ***BUF*** - The type of the intermediate value of the reduction.
- ***OUT*** - The type of the final output result.

- Strictly typed datasets

Optimization

Window Functions

WindowSpec

- Can be more efficient than traditional group by, assuming you pre-partition data based on the same id (one shuffle only)
- You don't lose any extraneous columns as you would with a traditional group by

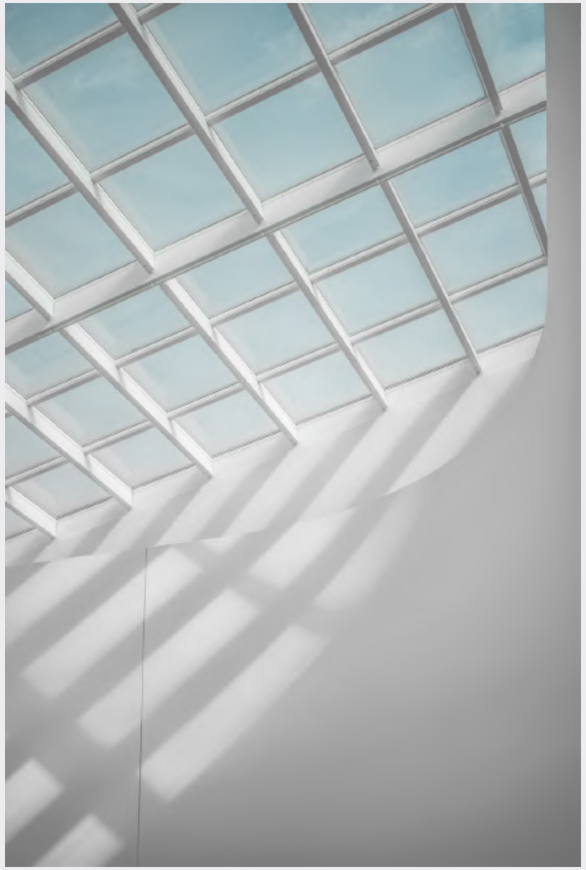


Photo by Jayden So on Unsplash

Window Function Example

Code snippets

```
// Define a window and some helper functions
val proceedure_window: WindowSpec = Window.partitionBy(col("procedure"))
    .orderBy(col("procedure"))
def selectTopOneRecord(window: WindowSpec)(df: DataFrame): Dataset[Row] = {
    df.withColumn("row_number", row_number().over(window))
    .filter(col("row_number") === 1)
    .drop("row_number")
}
def addSetColumn(structCol: String, window: WindowSpec): Column = {
    collect_set(structCol).over(window)
}
```

Window Function Example

Code snippets

```
// Transform a data set
case class RawData(id:Long,name:String,proceedure:String,cost:Double)
case class Provider(id:Long,name:String,cost:Double)
case class ProcedureWithProviderList(proceedure:String,providers:Seq[Provider])
def addProdcureStruct(): Column = { struct(col("id"), col("name"), col("cost"))}
val my_transformed_ds = my_input_ds
      .withColumn("procedures", addProdcureStruct())
      .withColumn("procedures",addSetColumn("procedures",procedure_window))
      .transform(selectTopOneRecord(procedure_window))
      .as[ProcedureWithProviderList]
```

Issue # 1

Even Spark has its limits

CMS In-network rates MRF Schema is not Spark friendly

- Expected file size about 1.6 TB uncompressed and 700 GB compressed.
- Aggregate 1000's of billing codes & more than 2GB of data into an Array column
- Aggregation action forces Spark to pull all partitioned data to the same executor.

```
java.lang.IllegalArgumentException: Cannot
    grow BufferHolder by size XXXXXXXXX
    because the size after growing exceeds
    size limitation 2147483632
```

Solution

Back to basics

Photo by Gabriel Heinzer on Unsplash

Using Databricks as a compute platform

- It's a linux box under the hood!
  - Scalable EC2*
- Notebooks make it easy to mix in OS commands (%sh) with Scala or Python based Spark code
- Mount storage from AWS S3*

* when using Databricks on AWS

Solution

Back to basics

Using Databricks as a compute platform

- Reading JSON data written out from Spark
- Using storage optimized instances
- Stitching together JSON data with bash (%sh) commands in a notebook
- Utilizing Databricks mount points to read / write data
- Monitoring server metrics with Ganglia UI

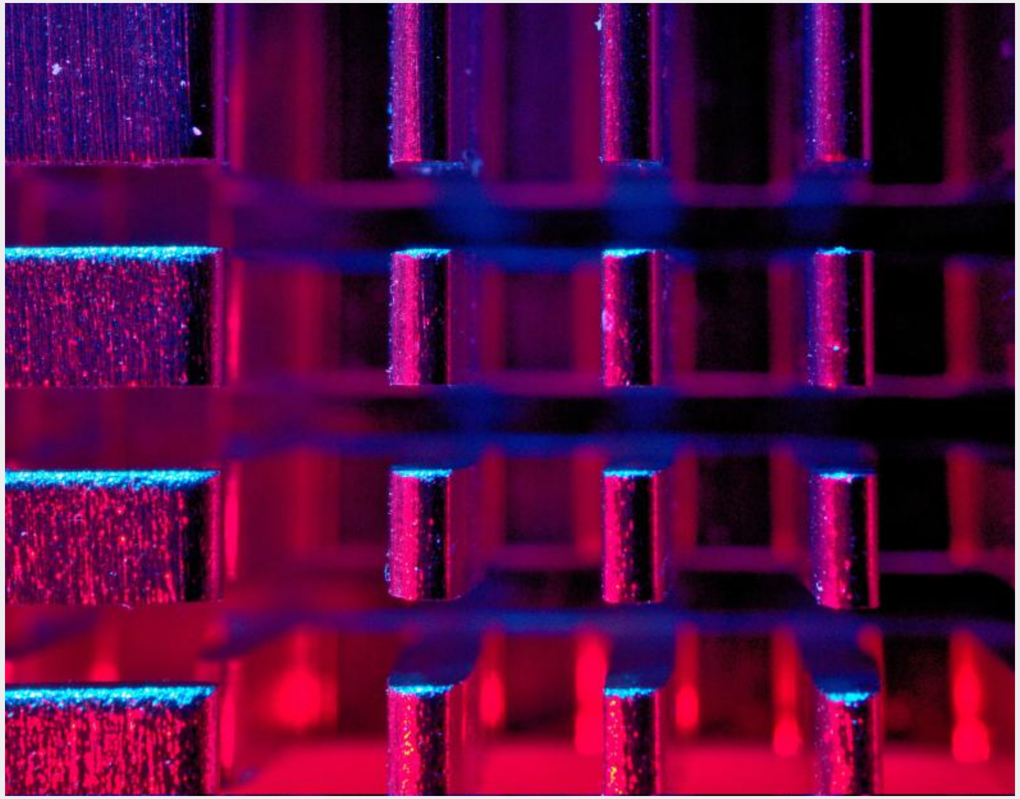


Photo by Michael Dziedzic on Unsplash

Issue # 2

Serving up a large amount of data for a public domain

- Terabytes of data over HTTP
- Products with expected file sizes of 1.6 TB for a single JSON file.
- Public access
- 1-[n] downloads for each file

Photo by Timon Studler on Unsplash

Solution

Serving up a large amount of data for a public domain

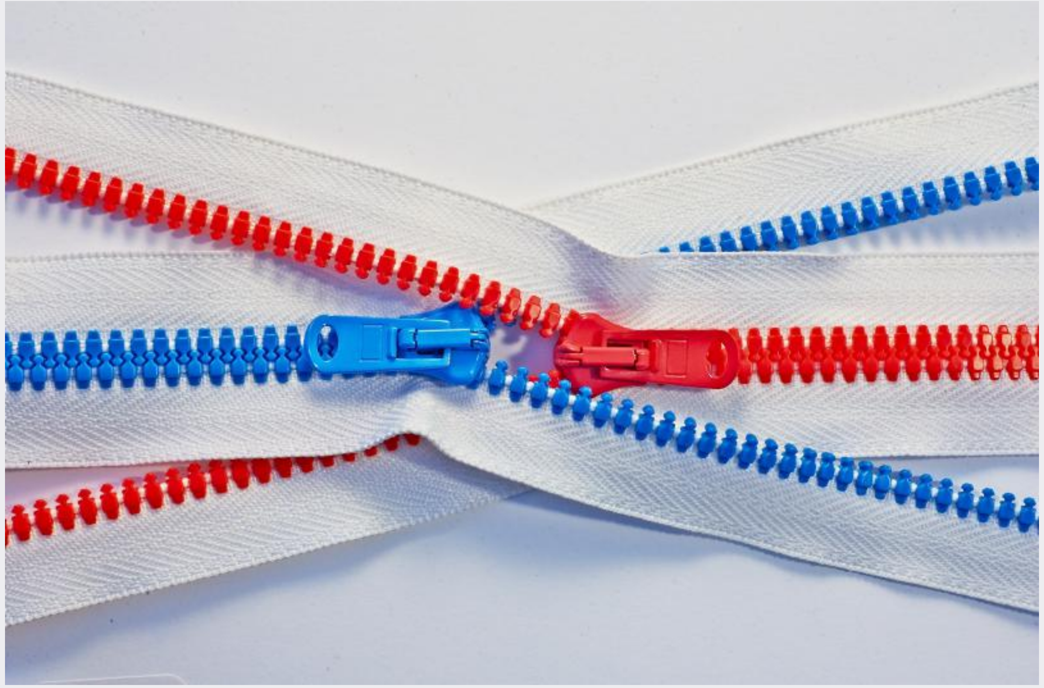


Photo by [Tomas Sobek](#) on [Unsplash](#)

- Compressing data during Spark write and file stitching process
- Using window functions to remove duplicate rows
- Utilizing AWS CloudFront to cache data for downloads

Automation

DevOps

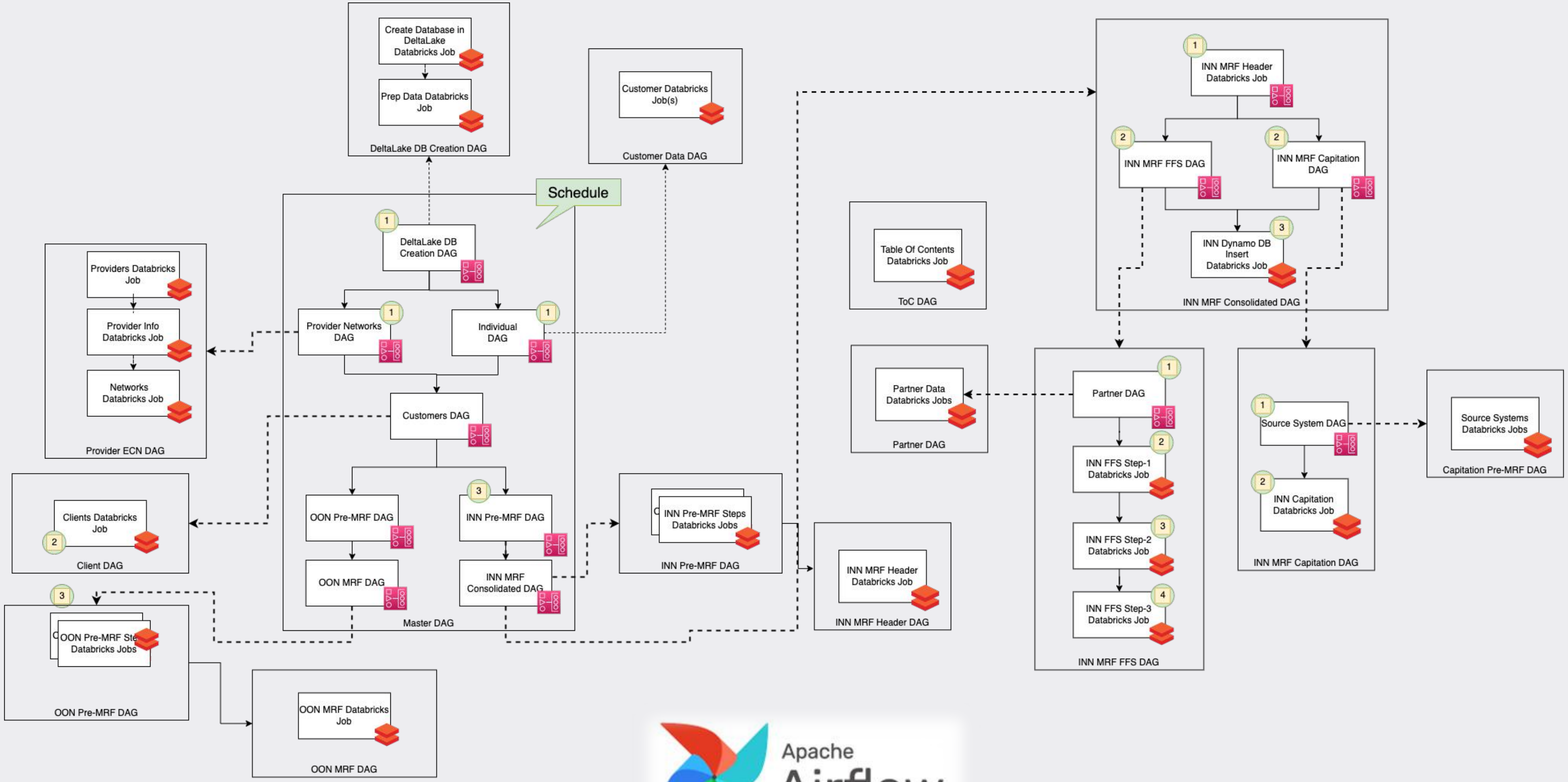
Automate end-to-end pipelines and runtime

Photo by Christophe Dion on Unsplash

Curation-D

- Automated pipeline
  - Databricks Jobs
  - Jenkins
  - Terraform
  - SBT / Plz
- Airflow for Scheduling and Orchestration

Master DAG



Create Database in DeltaLake Databricks Job
Prep Data Databricks Job
DeltaLake DB Creation DAG
Customer Databricks Job(s)
Customer Data DAG
Schedule
DeltaLake DB Creation DAG
Provider Networks DAG
Individual DAG
Customers DAG
OON Pre-MRF DAG
INN Pre-MRF DAG
OON MRF DAG
INN MRF Consolidated DAG
Master DAG
Providers Databricks Job
Provider Info Databricks Job
Networks Databricks Job
Provider ECN DAG
Clients Databricks Job
Client DAG
OON Pre-MRF Steps Databricks Jobs
OON Pre-MRF DAG
OON MRF Databricks Job
OON MRF DAG
INN Pre-MRF Steps Databricks Jobs
INN Pre-MRF DAG
INN MRF Header Databricks Job
INN MRF Header DAG
Table Of Contents Databricks Job
ToC DAG
Partner Data Databricks Jobs
Partner DAG
INN MRF Header Databricks Job
INN MRF FFS DAG
INN MRF Capitation DAG
INN Dynamo DB Insert Databricks Job
INN MRF Consolidated DAG
Partner DAG
INN FFS Step-1 Databricks Job
INN FFS Step-2 Databricks Job
INN FFS Step-3 Databricks Job
INN MRF FFS DAG
Source System DAG
INN Capitation Databricks Job
INN MRF Capitation DAG
Source Systems Databricks Jobs
Capitation Pre-MRF DAG

DATA+AI SUMMIT 2022

Thank you

Ross Silberquit
IT Principal Engineer
linkedin.com/in/rosssilberquit1

Narayanan HariharaSubramanian
IT Principal Director
linkedin.com/in/narayanan-a-h

Appendix

Disclaimer

- All architectural & code samples are for demonstrative purposes only and should not be considered a complete working solution.

Appendix

Sources

- All photos are from Unsplash.com (https://unsplash.com/license)
- Aggregators https://spark.apache.org/docs/latest/sql-ref-functions-udf-aggregate.html#user-defined-aggregate-functions-udafs
- Aggregators sample notebook: https://docs.databricks.com/_static/notebooks/dataset-aggregator.html
- Window functions https://spark.apache.org/docs/latest/sql-ref-syntax-qry-select-window.html

DATA+AI SUMMIT 2022

Thank you

Ross Silberquit
IT Principal Engineer
linkedin.com/in/rosssilberquit1

Narayanan HariharaSubramanian
IT Principal Director
linkedin.com/in/narayanan-a-h