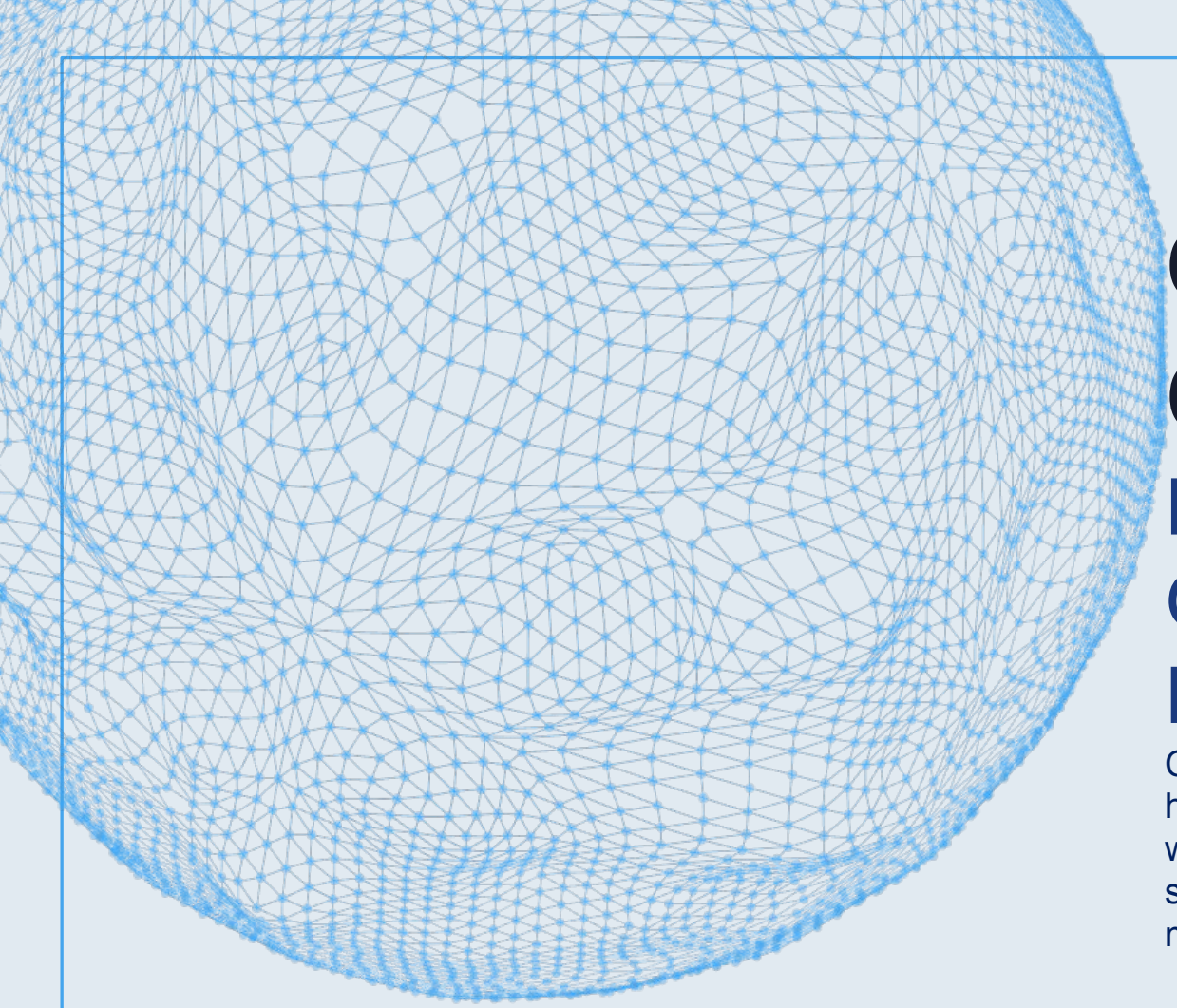


CLOUD COMPUTE COST SHOWDOWN:

HOW WE SAVED MILLIONS OPTIMIZING AWS EMR, DATABRICKS WORKLOADS

Cloud costs spiraling out of control. In this talk, we'll break down how we saved millions optimizing AWS EMR and Databricks workloads without sacrificing performance. Learn real world strategies, cost benchmarking insights, and best practices to maximize efficiency and slash cloud spend Don't miss it

SESHENDRANATH BALLA VENKATA
SENIOR MANAGER OF DATA ENGINEERING.

AGENDA

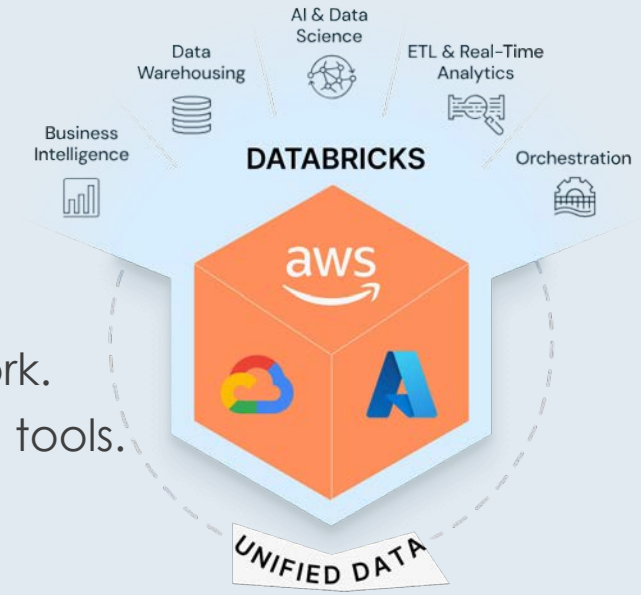
- What is Databricks
- Databricks Cost Optimization
- What is AWS EMR
- AWS EMR differences
- AWS EMR Cost Optimization

What is Databricks

Databricks is a cloud-based, unified analytics platform that integrates big data processing, data engineering, machine learning, and AI using Apache Spark.

Key Features:

Fully Managed Apache Spark – Optimized Spark runtime for high performance.
Delta Lake – ACID-compliant data lake with optimized storage and query performance.
Serverless Compute – Auto-scaling clusters reduce costs and complexity.
Collaborative Notebooks – Interactive notebooks for data science & ML workflows.
Multi-Cloud Support – Available on AWS, Azure, and Google Cloud.



Benefits of Databricks

Cost-Efficient – Optimized resource usage with auto-scaling clusters.
High Performance – Photon Engine accelerates SQL & Spark queries.
Collaboration-Friendly – Supports shared notebooks & real-time teamwork.
Seamless Integration – Connects with AWS, Azure, Google Cloud, and BI tools.

Databricks Cost Optimization

Review EBS Usage	Save between 50% to 80%	• Disk storage is used for Spark shuffle operations where the intermediate data is persisted to disk if the memory is full. • Check your instance types and the allocated disk storage and remove EBS storage for instances that already come with local storage. Eg. i3.4xlarge workers have 3.6 TB of local storage. • EBS Cost per Job=Storage Cost+I/O Cost+Snapshot Cost+Data Transfer Cost.
Understand auto scaling pitfalls	Save up to 25%	• Using Databricks jobs API requests, we were able to know if we have over allocated resources to a job. We are analyzing the RESIZING requests submitted by the job during the lifetime of the job cluster • Disable auto-scaling so that all the workers are available from the start of job execution which eliminates resizing and decreases job run time.
Reducing AWS Config cost	Save up to 50%	• Disabling auto scaling, limits the configuration changes that need to be recorded. • Prioritize using instances with local storage as each time you create, modify, or delete an EBS volume, AWS Config records a CI. This reduces both config and EBS costs. • Move appropriate workloads from m nodes to i nodes. This reduces resource recording and hence the Config costs as most of the recording requests are for EBS volumes.
Choosing the right instance type	Save up to 40%	• We often miss a lot of savings by choosing the wrong instance type.ex: M5.4xlarge vs I3.4xlarge
Choosing Spot Vs On Demand instances	Save 30%	• Cheaper vs pricier than spot instances • Will get terminated on AWS request vs More Savings as most of the cost is refundable under AWS savings plan.

Databricks Cost Optimization

Using right number of Shuffle partitions

Spark's shuffle operations, such as `groupBy`, `join`, and `reduceByKey`, redistribute data between partitions. The `spark.sql.shuffle.partitions` setting controls this, defaulting to 200. This fixed value often results in inefficient shuffle sizes: too small for small datasets and too large for big ones. Observation : (2*No of cores) property matching the cluster config

Knowing when to Cache data

While caching is a powerful tool, it's crucial to use it judiciously. The Spark UI, can provide insights into memory usage, cache hit ratios, and GC activity, helping us make informed decisions about caching strategies

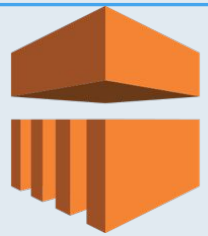
Addressing Duplicates

Spark offers distinct(), dropDuplicates(), and window functions for removing duplicates, varying in flexibility and complexity.
Grouping/aggregation consolidates duplicates from operations, while proactive data ingestion prevents them at the source.
Salting, by adding random prefixes to keys, can mitigate data skew during duplicate removal, improving shuffle performance.

Establish a Robust Data Lifecycle and Cost Control System

By defining lifecycle policies, you can automatically shift objects to more cost-effective storage tiers as they age.
Enable free, machine learning-powered cost anomaly alerts in AWS Cost Explorer. Set custom thresholds to receive notifications when your spending deviates from expected patterns, helping you quickly identify and address unexpected costs.

What is AWS EMR



amazon
EMR

AWS EMR is a cloud-based big data processing service that simplifies running large-scale Apache Spark, Hadoop, Hive, Presto, and other distributed frameworks.

AWS EMR provides multiple deployment options to suit different big data processing needs. The three main types are:The best EMR type depends on your workload, budget, and operational requirements:

1. Use **EMR on EC2** for persistent, large-scale workloads needing fine-tuned performance.
   ❌ Requires cluster management (provisioning, scaling, monitoring).
2. Use **EMR on EKS** for containerized, Kubernetes-managed workloads.
   ❌More complex setup than traditional EMR on EC2.
3. Use **EMR Serverless** for on-demand, pay-per-use processing without cluster management.
   ❌Less control over configurations compared to EC2-based EMR.

EMR on EC2 vs EMR on EKS vs EMR Serverless

Feature	EMR on EC2	EMR on EKS	EMR Serverless
Cluster Control	High	Medium	Low
Scaling	Manual/Auto	Auto	Fully Auto
Best for	Large-scale ETL, persistent workloads	AI/ML, containerized jobs	Ad-hoc, bursty jobs
Cost Efficiency	High (if optimized)	Moderate	Highest (pay-per-use)
Ease of Use	Moderate	Complex	Easiest

AWS EMR Cost Optimization

1. Right-Sizing Clusters Based on Workload Patterns

- Migrated from m5.4xlarge to m5.2xlarge and r5 instances for cost-efficient compute and memory balancing.
- Implemented compute-optimized instances (c5, c6i) for Spark executors, significantly improving performance per dollar spent.
- Transitioned to Graviton-based instances (c7g) where applicable, reducing costs by ~20%.

2. Dynamic Auto-Scaling for Efficient Resource Utilization

- Enabled EMR Managed Scaling to adjust instance count dynamically based on job demand.
- Used Spot Instances for transient jobs, cutting costs by up to 70%.
- Implemented Instance Fleet configuration to leverage the best-priced instances at runtime.

3. Optimizing Apache Spark Performance

- Reduced shuffle data by leveraging AWS Glue Data Catalog and Parquet format, cutting down S3 I/O costs.
- Implemented dynamic allocation for Spark executors, ensuring optimal resource usage per job.
- Adjusted Spark parallelism settings and executor memory allocation to maximize performance while minimizing waste.

AWS EMR Cost Optimization

4. Serverless EMR vs. EKS Compute Processing

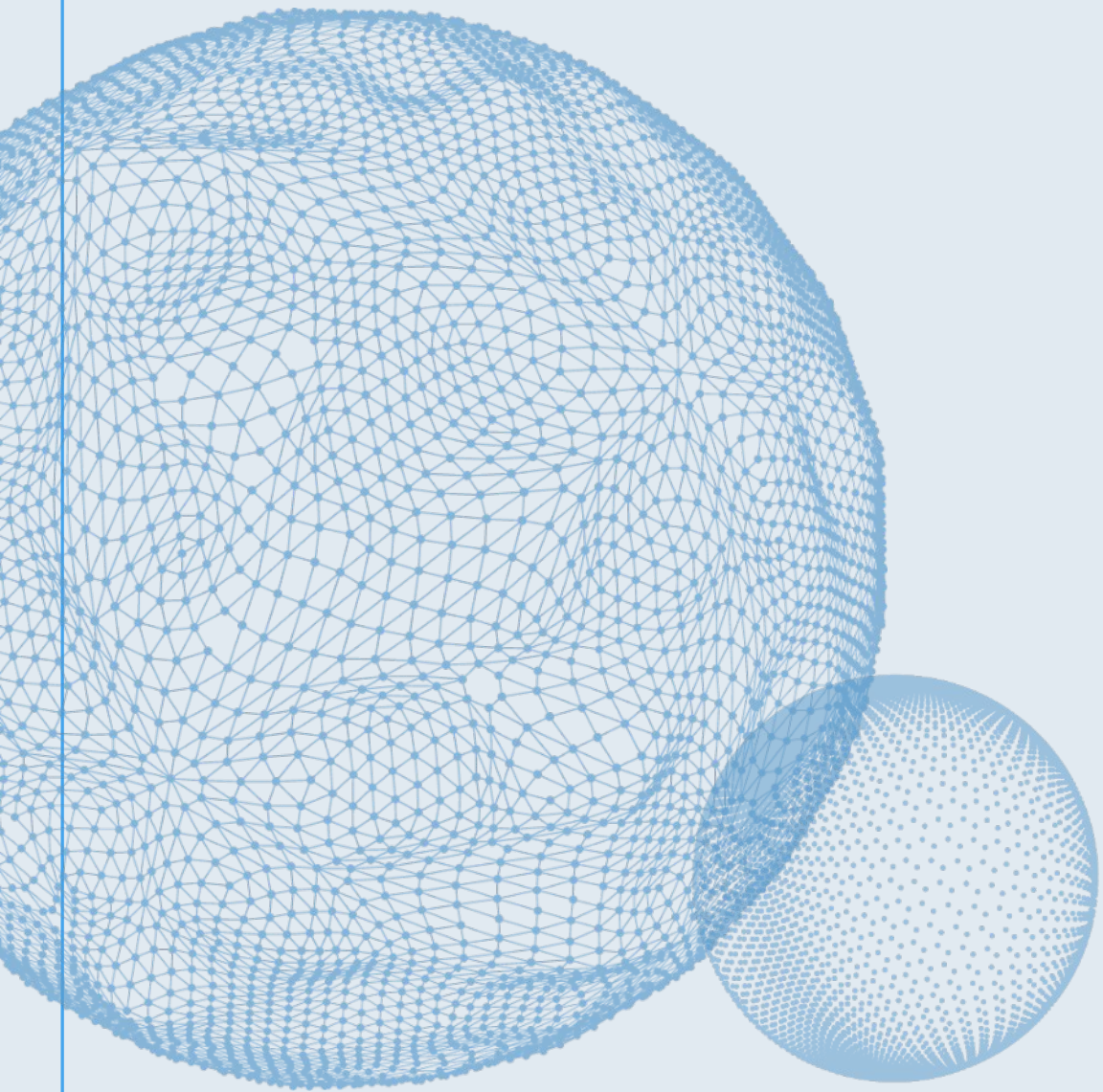
- EKS with Spot instances provided better cost efficiency for predictable workloads due to finer control over infrastructure.
- Serverless EMR worked best for bursty, unpredictable jobs due to its automated scaling.
- Hybrid approach: Migrated recurring workloads to EKS while keeping ad-hoc jobs on Serverless EMR.

5. Data Storage and Access Optimization

- Reduced S3 read/write costs by caching frequently used datasets in Hudi/Iceberg instead of redundant full-table scans.
- Implemented AWS Lake Formation permissions to minimize unnecessary data duplication and enforce row/column-level security.
- Adjusted retention policies, reducing old data storage costs by 40%.

6. Automating Cluster Shutdowns & Scheduling

- Integrated AWS Step Functions to auto-terminate idle clusters post-job completion.
- Implemented job scheduling via Apache Airflow, optimizing cluster lifecycles and avoiding long-running idle clusters.



THANK YOU