How to start your journey into data analytics

Cost Modeling Data Lakes for Beginners

Cost Modeling Data Lakes for Beginners : How to start your journey into data analytics

Table of Contents

Abstract **1**
Abstract 1
Introduction **2**
What should the business team focus on? 2
Defining the approach to cost modeling data lakes **4**
Measuring business value **5**
Building data lakes **7**
Batch processing 7
Real-time processing 9
Understanding what influences data lakes costs **12**
Ingestion 12
Cost factors 12
Streaming 13
Cost factors 13
Cost optimization factors 13
Processing and transformation 14
Cost factors 14
Cost optimization factors 14
Analytics and storage 15
Cost factors 15
Cost optimization factors 15
Visualization and API access 16
Cost factors 16
Cost optimization factors 16
Metadata management, data catalog, and data governance 16
Cost factors 17
Cost optimization practices 17
Overview of cost optimization 17
Use data compression 18
Reduce run frequency 18
Partition data 19
Choose a columnar format 19
Create data lifecycle policies 20
Right size your instances 20

Use Spot Instances ... 20
Use Reserved Instances ... 21
Choose the right tool for the job ... 21
Use automatic scaling ... 22
Choose serverless services ... 22
Cost optimization in analytics services ... 24
Streaming ... 24
Kinesis Data Streams ... 24
Firehose ... 25
Amazon Managed Streaming for Apache Kafka (Amazon MSK) ... 27
Storage ... 30
Amazon Simple Storage Service ... 30
Processing and analytics ... 34
Amazon EMR ... 34
AWS Glue ... 38
Lambda ... 41
Amazon Athena ... 43
Amazon Redshift ... 46
Monitoring data lakes costs ... 49
Conclusion ... 50
Contributors ... 51
Further reading ... 52
Document revisions ... 53
AWS Glossary ... 54

Cost Modeling Data Lakes for Beginners

Publication date: **March 03, 2023** (*Document revisions*)

Abstract

This whitepaper focuses on helping you understand and address the first challenge of your data lake journey: how much will it cost? Understanding cost considerations can inform your organizational business case and decision-making process when evaluating the value realization outcomes for a data lake project.

This whitepaper discusses the challenges of using traditional methodologies for costing data lake projects. It then outlines an approach that enables you to move at speed, realizing value early on the project cycle.

Introduction

Customers want to realize the value held in the data their organization generates. Common use cases include helping them expedite decision making, publishing data externally to foster innovation, or creating new revenue streams by monetizing the data.

Organizations that successfully generate business value from their data, will outperform their peers. An Aberdeen survey saw organizations who implemented a Data Lake outperforming similar companies by 9% in organic revenue growth. These leaders were able to do new types of analytics like machine learning over new sources like log files, data from click-streams, social media, and internet connected devices stored in the data lake. This helped them to identify, and act upon opportunities for business growth faster by attracting and retaining customers, boosting productivity, proactively maintaining devices, and making informed decisions.

To best realize this value using data lakes, customers need a technology cost breakdown for their budget and realize the best value of their solution. But without building the solution, they don't know how much it will cost. This is a common paradox that delays many customers from starting their data lake projects.

A data lake is a common way to realize these goals. There are many considerations along this journey, such as team structure, data culture, technology stack, governance risk, and compliance.

Costing data lakes requires a different approach than delivering them. Customers must focus on identifying and measuring business value early on so that they can start their projects quickly and demonstrate the value back to the business quickly and incrementally.

What should the business team focus on?

- **Measure business value** – Without measuring business value, it's hard to justify any expenditure or drive any value from your testing early on in the project.
- **Prototype rapidly** – Focus your energy on driving business outcomes with any experiments you run.
- **Understand what influences costs** – Analytics projects generally have similar stages, ingestion, processing, analytics, and visualization. Each of these stages has key factors that influence the cost.
- **Cost model a small set of experiments** – Your analytics project is a journey. As you expand your knowledge, your capabilities will change. The sooner you can start experimenting, the sooner

your knowledge will grow. Build a cost model that covers to smallest amount of work to impact your business outcomes and iterate.

Defining the approach to cost modeling data lakes

IT projects historically have well-defined milestones that make cost modeling a fairly simple process. Selected software is usually a commercial off-the-shelf (COTS) product, which can be costed (including licenses) and based on predictable metrics, such as number of users and number of CPUs.

The effort to implement the software is well within the standard skill sets of the IT department, with plenty of experience in similar deployment models to draw on to get an accurate picture of the implementation time. This will all feed into a large design document that can be used to calculate costs.

Therefore, you can expect to use the same cost modeling you have previously used for an analytics project. The challenge here is that an analytics project often doesn't have a clear end. It's a journey where you explore and prepare data in line, with some aspirational business outcomes. This makes it difficult to know the following:

- How much data will be ingested
- Where that data will come from
- How much storage or compute capacity will be needed to make sense of that data
- Which services are required to exploit the data to realize and deliver value throughout the business

However, analytics projects provide the radical growth and business opportunities to enable your organization grow and get a competitive edge. Customers who have realized business benefit include the following companies.

- Siemens mobility, a leader in transport solutions, cut their energy costs by 1015% with analytics projects.
- Kumon, an online talent market for freelancers, managed to increase purchasing conversion by 30% and decrease churn by 40%.
- 3Victors, a leader in travel data analytics, built their business by collecting billions of shopper requests for flights and analyzing them on AWS. This provided actionable real-time insights to travel marketers and revenue managers. Those insights, in turn, enabled 3Victor to improve the customer conversion rate and improve their return on advertising spend.

Measuring business value

The first project on your data lake journey starts by stating what your business goals are. These are often extracted from your business strategy. Business goals are high-level aspirations that support the business strategy, such as “improve customer intimacy”.

Once these business goals are identified, together with your team write a set of business outcomes that would have a positive impact against these goals. Outcomes are targeted and measurable, such as *reduce cost of customer acquisition*.

Finally, we identify a number of metrics that can be measured to validate the success of the experiments. This helps ensure that the right business value is being achieved.

A good example of this is Hyatt Hotels, a leading global hospitality company. Hyatt wanted to improve customer loyalty, improve the success rate of upsells and add-ons, and better guide the users with accommodation recommendations. This fed directly into their business goal to improve their American Customer Satisfaction Index (ACSI). To achieve this, the team at Hyatt identified a requirement to build personalized connections with their customers.

Some example business outcomes could be:

- Number of return visits per 1000 customers per month
- Number of upsells per 100 customers per month
- Number of add-ons per 100 customers per month
- Number of bookings made per day
- Number of negative customer reviews per weeks

This is only an example of one piece of work a data lake could support. After the initial piece of work is delivered, the team can then iterate. For example, as a by-product of delivering the previous feature, they could have identified important information. For example, perhaps Hyatt discovered that commonly searched for add-on purchases (for example, specific spa treatments) were not offered in the chosen location. Or, they might have discovered that customers were searching for accommodation in areas that Hyatt doesn’t yet have a footprint in.

The team could go on to develop new features and service offerings that would help them deliver a better customer experience or help them make decisions that would improve their chances of

choosing the right location to scale their footprint globally to help them deliver against their business goal.

Building data lakes

Because the aim is to get started on your data lake project, let's break down your experiments into the phases that are typical in data analytics projects:

- Data ingestion
- Processing and transformation
- Analytics
- Visualization, data access, and machine learning

By breaking down the problem into these phases, you reduce the complexity of the overall challenge. This makes the number of variables in each experiment lower, enabling you to get model costs more quickly and accurately.

We recommend that you start your analytics project by implementing the foundation of a data lake. This gives you a good structure to tackle analytics challenges and allow great flexibility for the evolution of the platform.

A data lake is a single store of enterprise data that includes raw copies from various source systems and processed data that is consumed for various analytics and machine learning activities that provide business value.

Choosing the right storage to support a data lake is critical to its success. Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data.

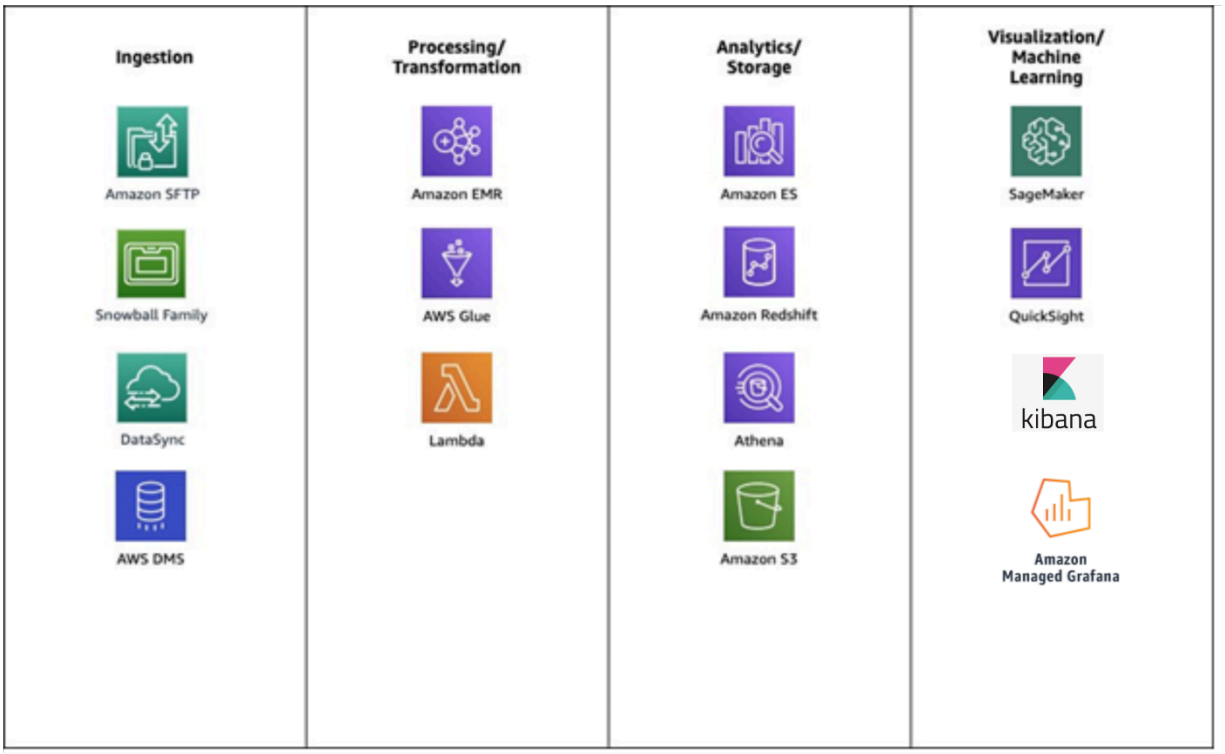
Data lakes generally support two types of processing: batch and real-time. It is common for more advanced users to handle both types of processing within their data lake. However, they often use different tooling to deliver these capabilities. We will explore common architectures for both patterns and discuss how to estimates costs with both.

Batch processing

Batch processing is an efficient way to process large volumes of data. The data being processed is typically not time-critical and is usually processed over minutes, hours, and in some cases, days. Generally, batch processing systems automate the steps of gathering the data, extracting

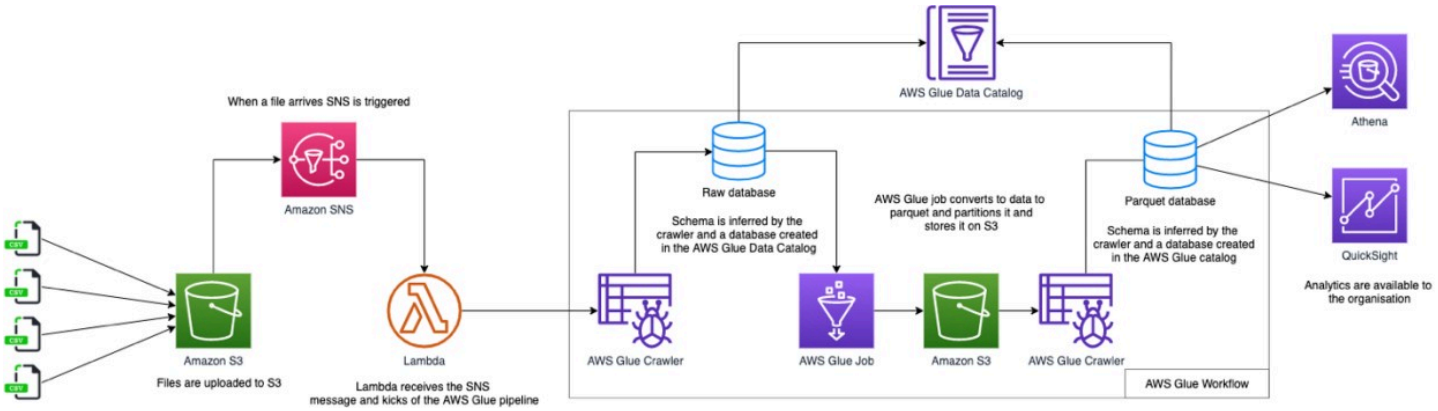
it, processing it, enriching it, and formatting it in a way that can be used by business applications, machine learning applications, or business intelligence reports.

Before we get started, let's look at a common set of services that customers use to build data lakes for processing batch data.



Common services used to build data lakes for batch data

The following example architecture is relatively common. It uses AWS Glue, Amazon Athena, Amazon S3, and Amazon QuickSight.



Example architecture for batch processing

The preceding example shows a typical pipeline to ingest raw data from CSV files. AWS Glue automatically infers a schema to allow the data to be queried. AWS Glue jobs are used to extract, clean, curate, and rewrite the data in an optimized format (Parquet) before exposing visualizations to end users. This is all achieved using serverless technologies that reduce the operational burden to the analytics team.

We are going to explore each of these steps in more detail, in addition to the things you need to consider along the way. But, before we do, let's take a quick look at the other form of processing.

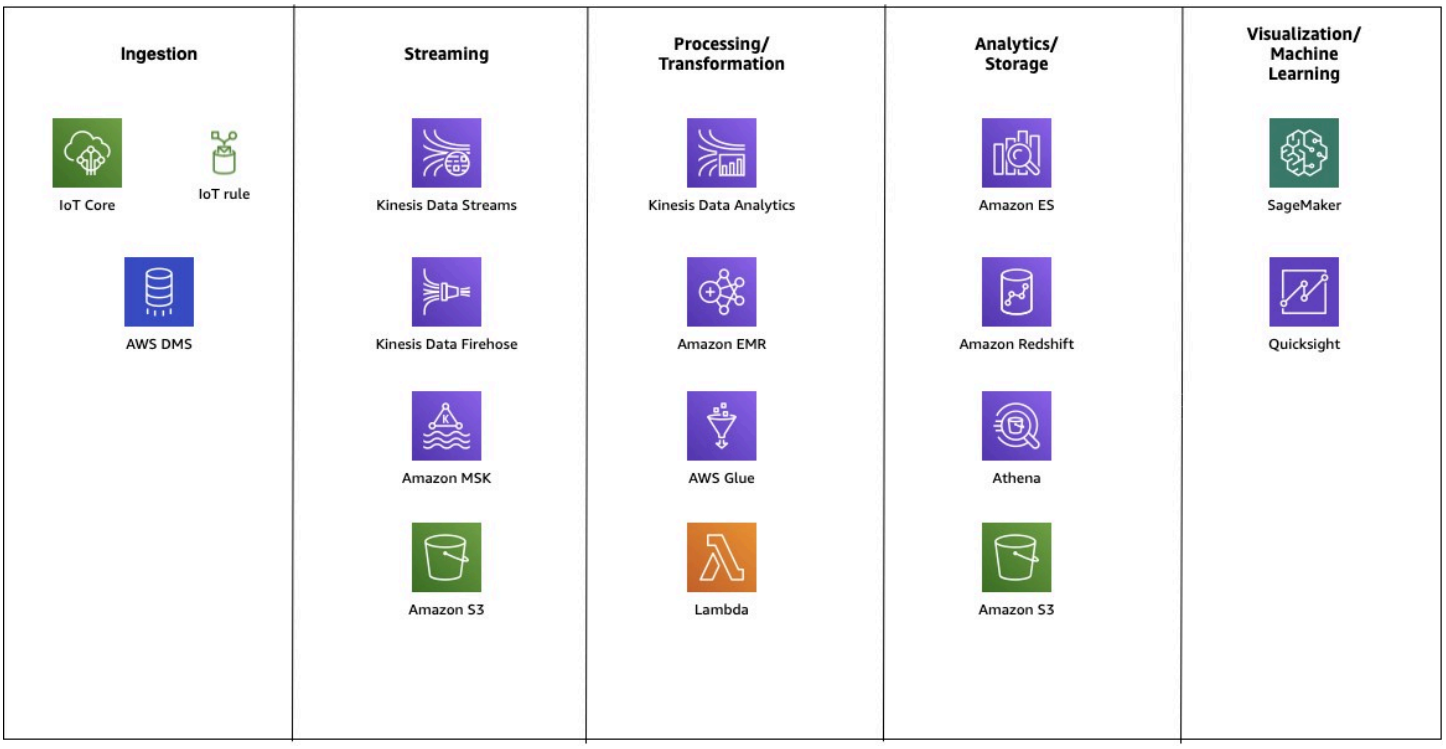
Real-time processing

Real-time processing is a way of processing an unbounded stream of data in order to generate real-time (or nearly real-time) alerts or business decisions. The response time for real-time processing can vary from milliseconds to minutes.

Real-time processing has its own ingestion components and has a streaming layer to stream data for further processing. Examples of real-time processing are:

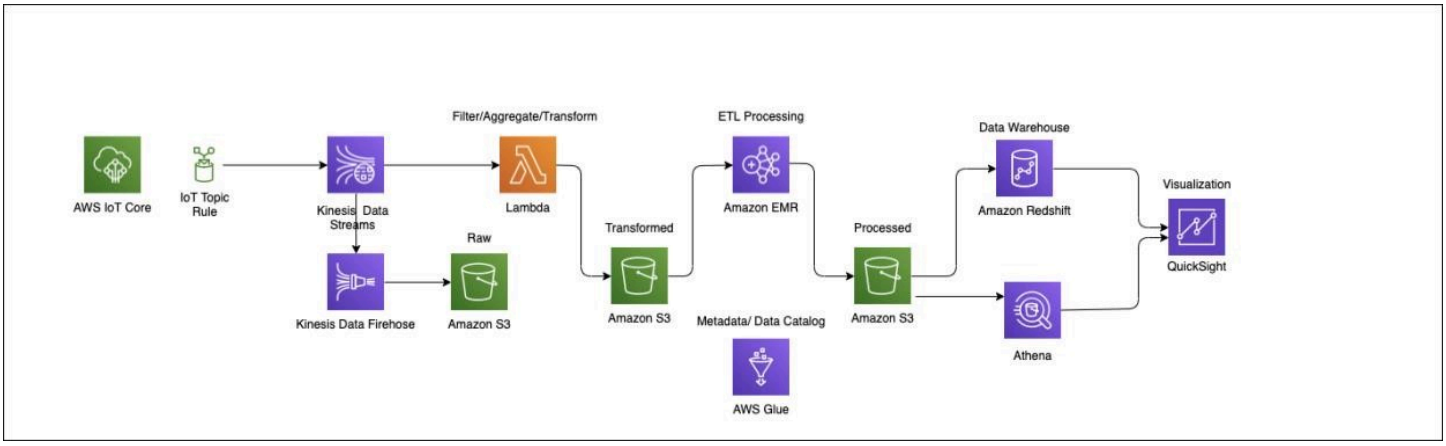
- Processing IoT sensor data to generate alerts for predictive maintenance
- Trading data for financial analytics
- Identifying sentiment using a real-time Twitter feed

Before we get started, let's look at a common set of services that customers use to build data lakes for processing real-time data.



Common services used to build data lakes for real-time data

Our example architecture is relatively simple and uses the following services: Amazon Kinesis, AWS Lambda, AWS Glue, Amazon Athena, Amazon S3 and Amazon QuickSight.



Example architecture for real-time processing

This example shows that many IoT devices send their telemetry to AWS IoT Core. AWS IoT allows users to securely manage billions of connected devices and route those messages to other AWS endpoints. In this case, AWS IoT Core passes the messages Amazon Kinesis, which ingests streaming data at any scale. The raw data is split into two streams, one that writes the raw data to Amazon S3 and a second that uses AWS Lambda (a serverless compute service) to filter, aggregate,

and transform the data before again storing it on Amazon S3. The manipulated data is then cataloged in AWS Glue and made available to end users to run ad hoc queries using Amazon Athena and create visualizations using Amazon QuickSight.

Understanding what influences data lakes costs

Across both real-time and batch processing, data flows through different stages. For each stage, there is an option to use managed services from AWS or to use compute, storage, or network services from AWS with third-party or open source software installed on top it.

In the case of managed services, AWS provides service features like high availability, backups, and management of underlying infrastructure at additional cost. In some cases, the managed services are on-demand serverless based solutions, where the customer is charged only when the service is used.

In case of third-party or open source software, the user pays AWS for infrastructure components being used and does the management and administration work themselves (in addition to license cost, if applicable).

Ingestion

Services required for ingestion depend on the source of data, the frequency at which the data should be consumed for processing to derive business value from it, and the data transfer rate.

Cost factors

Depending on the services you choose, the primary cost factors are:

- **Storage** – These are the costs you pay for storing your raw data as it arrives.
- **Data transfer** – These are the costs you pay for moving the data. Costs can be either bandwidth charges, leased line, or offline transfer. It is worth noting the analytics services should be deployed in the same Region to avoid unnecessary inter-Region data transfer. This improves performance and reduces costs.
- **Managed service costs** – These are the costs you pay (usually per second or per hour) for the service you are using, if you chose a managed service from AWS (for example, AWS IoT or AWS Transfer for SFTP).

Streaming

This stage is only applicable for real-time processing. This stage is primarily responsible for ingesting the unbounded stream of data and providing guaranteed delivery for downstream processing.

Cost factors

The primary costs of this stage are:

- **Data transfer** – These are the costs you pay for the rate at which data is consumed by the data streaming service.
- **Streaming service costs** – These are the costs you pay (usually per second or per hour) for AWS management of Amazon Kinesis or Amazon MSK service that is being used, including instance cost of Amazon MSK.
- **Storage cost** – This is the cost of storing data in streaming service until data is consumed by its consumers and processed.

Cost optimization factors

Ingesting and processing real-time streaming data requires the infrastructure to support the aggregation of the source events, processing of streams, and making the data available for consumption. The AWS streaming ETL services such as Kinesis Data Streams, Kinesis Firehose, and Amazon Managed Kafka Services (MSK), reduces the administration cost. AWS manages the infrastructure, storage, networking, and configuration in your streaming ETL pipeline.

We recommend that you consider the following actions to reduce the cost when using the following services:

- Kinesis Data Streams
- Kinesis Firehose
- Amazon Managed Streaming for Apache Kafka(Amazon MSK)

Processing and transformation

The cost of processing depends on number of factors, such as the size of the data being processed, the complexity of the transformations (which leads to the choice of tool), and how the data is stored (format and compression).

Cost factors

The primary cost at this stage includes:

- **Processing unit cost** - This is the cost of compute and memory required to process the data. If you use a service like Amazon EMR, the Amazon EC2 instance used to process the data will incur the cost.
- **Managed service costs** - These are the costs you pay (usually per second or per hour) for the management of the service. For a serverless service like AWS Glue, this will be billed based on the utilization of the service.
- **Storage cost** - This is the cost of storing the processed data.

Cost optimization factors

AWS analytics services take care of many of the operational tasks of data processing and transformation. The managed analytics services from AWS offer advanced features without additional licensing fees and lower the customer's cost of operations by eliminating the need for a dedicated team of experts to manage their clusters. As a result, customers using the managed analytics services will realize the benefit of Operational efficiency, Performance, Availability, and Scalability.

We recommend that you consider the following actions to reduce the cost when using the following services:

- Amazon EMR
- AWS Glue
- Lambda
- Amazon Athena
- Amazon Redshift

Analytics and storage

The business value of data lakes is derived using tools in analytics stage. Cost in this stage correlates directly to the amount of data collected and analytics done on the data to derive the required value.

Cost factors

The primary cost of this stage includes:

- **Processing Unit cost** - This is the cost of instances used by the analytics tool being used. Data Warehouse solution like Amazon Redshift or analytics solutions on Amazon EMR or operational analytics on Amazon OpenSearch Service are all billed based on the instance type and number of nodes used.
- **Storage cost** - The data stored for analysis either on Amazon S3 or on the nodes of the analytics tool itself contribute to this cost.
- **Scanned data cost** - This is applicable only for serverless service like Athena and Amazon Redshift Spectrum, where the cost is based on the data scanned by the analytics queries.

Cost optimization factors

AWS provides reliable, scalable, and inexpensive storage and compute building blocks combined with fully managed value-added analytics services to help customers quickly gain insights into their data. In addition, the AWS-managed storage and analytics services help customers simplify the storage and analytics of their data by taking care of the infrastructure administration and management.

We recommend that you consider the following actions to reduce the cost when using the following services:

- Amazon EMR
- AWS Glue
- Lambda
- Amazon Athena
- Amazon Redshift
- S3

Visualization and API access

Visualization is used to graphically present the enormous amount of data in data lake in a human consumable format. API access refers to allowing developers to integrate your data in their apps to make it meaningful for consumers. The rate at which data is refreshed or request affects the number of queries being fired (and cost) in the analytics stage.

Cost factors

The primary cost of this stage includes:

- **Per user license cost** - Most of SaaS visualization tools charge per user license cost.
- **Processing unit cost** - The cost required to host the visualization software is applicable only for hosted visualization tools.
- **In-memory processing cost** - This is the cost of the visualization need to load huge amount of data in memory for quick performance. The amount of data required to be loaded also has a direct impact on cost, for example, Super-fast, Parallel, In-memory Calculation Engine (SPICE) in Amazon QuickSight.
- **Number of requests through an API** - As developers integrate with their apps the number of requests made for data will increase.

Cost optimization factors

We recommend that you consider the following actions to reduce cost:

- Choose the right tool for the job
- Choose serverless services
- Use automatic scaling
- Use Reserved Instances
- Right size your instances

Metadata management, data catalog, and data governance

In addition to the previous data lake stages, you need the following components to make use of the data effectively and securely.

- **Metadata management** - Responsible for capturing technical metadata (data type, format, and schema), operational metadata (interrelationship, data origin, lineage) and business metadata (business objects and description)
- **Data catalog** – An extension of metadata that includes features of data management and search capabilities
- **Data governance** – Tools that manage the ownership and stewardship of data along with access control and management of data usage

Cost factors

The primary cost of this stage includes:

- **Processing cost** – This is the cost associated with processing required to generate the catalog or metadata of data stored. In governance tools, this cost depends on number of governance rules that is being processed.
- **Storage cost** – This is the cost associated with amount of metadata and catalog stored.
- **License cost** – If there is any third party involved in providing these services, it will include the cost of that license.

Cost optimization practices

We recommend that you consider the following actions to reduce cost:

- Choose the right tool for the job
- Choose serverless services
- Reduce run frequency
- Partition data
- Choose a columnar format

Overview of cost optimization

Across stages, the following are general practices to optimize cost of the services used.

Use data compression

There are many different formats for compression. You should be selective about which one you choose to make sure it is supported by the services you are using.

The formats provide different features for example some formats support splitting meaning they support parallelization better, whilst others have better compression but at the cost of performance. Compressing the data, reduces the storage cost and also reduces the cost for scanning data.

To give an example of how compression can help reduce costs we took a CSV dataset and compressed it using bzip2. This reduced the data from 8 GB to 2 GB a 75% reduction. If we extrapolate this compression ratio to a larger dataset, 100 TB the savings are significant:

- Without compression: $2,406.40 per month
- With compression: $614.40 per month This represents a saving of 75%.

> **Note**
>
> Compression also reduces the amount of data scanned by compute services such as Amazon Athena. An example of this covered in the Columnar format section.

Reduce run frequency

In this scenario, let's assume we generate 1 TB of data daily. We could potentially stream this data, because it's available, or choose to batch it and transfer it in one go. Let's assume we have 1 Gbps bandwidth available for the transfer from our source system, for example on-premises systems. A single push of 1 TB daily would take around two hours. In this case we'll allow three hours for some variance in speed.

Also, in this scenario, we will use AWS SFTP to transfer the data. In the first scenario, we need the AWS SFTP service constantly running. In the second scenario, we only need it running for three hours a day to complete our transfer.

- Running constant transfers as the data is available: $259.96 per month
- Running a single batch transfer daily: $68.34 per month This represents a saving of just over 73%.

Partition data

Partitioning data allows you to reduce the amount of data that is scanned within queries. This helps both reduce cost and improve performance. To demonstrate this, we used the GDELT dataset. It contains around 300 million rows stored in many CSV files. We stored one copy un-partitioned and a second copy partitioned by year, month, day.

```
SELECT count(*) FROM "gdeltv2"."non-partioned|partitioned" WHERE year =
2019;
```

Unpartitioned:

- Cost of query (102.9 GB scanned): $0.10 per query
- Speed of query: 4 minutes 20 seconds Partitioned:
- Cost of query (6.49 GB scanned): $0.006 per query
- Speed of query: 11 seconds

This represents a saving of 94% and improved performance by 95%.

Choose a columnar format

Choosing a columnar format such as parquet, ORC or Avro helps reduce disk I/O requirements which can reduce costs and drastically improve performance in data lakes. To demonstrate this, we will again use GDELT. In both examples there is no partitioning but one dataset is stored as CSV and one as parquet.

```
SELECT * FROM "gdeltv2"."csv|parquet" WHERE globaleventid =
```

CSV:

- Cost of query (102.9 GB scanned): $0.10 per query
- Speed of query: 4 minutes 20 seconds Partitioned:
- Cost of query (1.04 GB scanned): $0.001 per query
- Speed of query: 12.65 seconds

This represents a saving of 99% and improved performance by 95%.

Create data lifecycle policies

With Amazon S3, you can save significant costs by moving your data between storage tiers. In data lakes we generally keep a copy of the raw in-case we need to reprocess data as we find new questions to ask of our data. However, often this data isn't required during general operations. Once we have processed our raw data for the organization, we can choose to move this to a cheaper tier of storage. In this example we are going to move data from standard Amazon S3 to Amazon S3 Glacier.

For our example, we will model these costs with 100TB of data.

- Standard Amazon S3: **$2,406.40** per month
- Amazon S3 Glacier: **$460.80** per month This represents a saving of just over 80%.

Right size your instances

AWS services are consumed on demand. This means that you pay only for what you consume. Like many AWS services, when you run the job you define how many resources you want and you also choose the instance size you want. You also have ability to stop the instance when it's not being used and you can spin it up based on specific schedule. For example, transient EMR clusters can be used for scenarios where data has to be processed once a day or once a month.

Monitor instance metrics with analytic workloads and downsize the instances if they are over provisioned.

Use Spot Instances

Amazon EC2 Spot Instances let you take advantage of unused Amazon EC2 capacity in the AWS Cloud. Spot Instances are available at up to a 90% discount compared to ondemand prices. For example, using spot helps the Salesforce save up to 40 percent over Amazon EC2 Reserved Instance pricing and up to 80 percent over on-demand pricing. Spot is a great use case for batch when time to insight is less critical.

In the following example, we have modeled a transient Amazon EMR cluster that takes six hours to complete its job.

- On-Demand Instance: $106.56 per month
- Spot Instance (estimated at a conservative 70%): $31.96 per month This represents a saving of just over 70%.

Use Reserved Instances

Amazon EC2 Reserved Instances (RI) provide a significant discount (up to 72%) compared to On-Demand pricing and provide a capacity reservation when used in a specific Availability Zone. This can be useful if some services used in the data lake will be constantly utilized.

A good example of this can be Amazon OpenSearch Service. In this scenario we'll model a 10 node Amazon OpenSearch Service cluster (3 master + 7 data) with both on-demand and reserved pricing. In this model, we'll use c5.4xlarge instances with a total storage capacity of 1 TB.

- On-demand: $ 7,420.40 per month
- RI (three years, all up front): $ 3,649.44 per month This represents a saving of just over 50%.

Choose the right tool for the job

There are many different ways to interact with AWS services. Picking the right tool for the job, helps reduce cost.

If you are using Kinesis Data Streams and pushing data into it, you can choose between the AWS SDK, the Kinesis Producer Library (KPL), or the Kinesis Agent. Sometimes the option is based on the source of the data and sometimes the developer can choose.

Using the latest KPL enables you to use a native capability to aggregate multiple messages/events into a single PUT unit (Kinesis record aggregation). Kinesis data streams shards support up to 1000 records per second or 1-MB throughput. Record aggregation by KPL enables customers to combine multiple records into a single Kinesis Data Streams record. This allows customers to improve their per shard throughput.

For example, if you have a scenario of pushing 100,000 messages/sec with 150 bytes in each message into a Kinesis data stream, you can use KPL to aggregate them into 15 Kinesis data stream records. The following is the difference in cost between using (latest version) KPL and not.

- Without KPL (latest version): $4,787.28 per month
- With KPL (latest version): $201.60 per month This represents a saving of 95%.

Use automatic scaling

Automatic scaling is the ability to spin resources up and down based on the need. Building application elasticity enables you to incur cost only for your fully utilized resources.

Building EC2 instances using an Auto Scaling group can provide elasticity for Amazon EC2 based services where applicable. Even for serverless services like Kinesis where shards are defined per stream during provisioning, AWS Application Auto Scaling provides ability to automatically add or remove shards based on utilization.

For example, if you are using Kinesis streams to capture user activity for an application hosted in specific Region the streaming information might vary between day and night. During day times, when the user activity is higher you might need more shards compared to night times when the user activity would be very low. Being able to configure AWS Application Auto Scaling based on your utilization enables you to optimize your cost for Kinesis.

Data flowing at rate of 50,000 records/sec with each record having 2 K bytes, will require 96 shards. If the data flow reduces to 1000 records/sec for eight hours during night, it would only need two shards.

- Without AWS Application Auto Scaling: $1068.72 per month
- With AWS Application Auto Scaling (downsizing): $725.26 per month

This represents a saving of 32%.

Choose serverless services

Serverless services are fully managed services that incur costs only when you use them. By choosing a serverless service, you are eliminating the operational cost of managing the service.

For example, in scenarios where you want to run a job to process your data once a day, using serverless service incurs a cost only when the job is run. In comparison, using a self-managed service incurs a cost for the provisioned instance that is hosting the service.

Running a Spark job in AWS Glue to process a CSV file stored in Amazon S3 requires

10 minutes of 6 DPU's and cost $0.44. To execute the same job on Amazon EMR, you need at least three m5.xlarge instances (for high availability) running at a rate of $0.240 per hour.

- Using a serverless service: $13.42 price per month

- Not using a serverless service: $527.04 price per month This represents a saving of 97%.

Similar savings are applicable between running Amazon Kinesis vs Amazon MSK and between Amazon Athena vs Amazon EMR.

For more information on cost optimization in detail, see the ??? section.

Cost optimization in analytics services

Streaming

Kinesis Data Streams

The fundamental unit of scaling for Kinesis Data Streams is a shard. A Kinesis data stream consists of several shards, and each shard provides a specific data ingestion and delivery rate (1 MB/s write and 2 MB/s reads). Each Kinesis data stream (on-demand mode) gets a default capacity of 4 MB/s (4000 records/s) for writes and can burst up to 200 MB/s (200K records/s).

On-demand mode is suitable for unpredictable traffic. In on-demand mode, Kinesis automatically manages the number of shards needed to provide the required throughput. For example, a data stream in on-demand mode observes peak write throughput for the last 30 days and automatically provisions capacity for double that rate. If a new peak is observed, capacity is adjusted to reflect that peak. As a result, aggregate read throughput for the whole stream increases proportionally to write throughput. Please note that if the traffic increases more than double the previous peak within 15 minutes, write throttling can occur. Requests that are throttled require a retry mechanism.

Provisioned mode is suitable for predictable traffic volumes that are easy to forecast. In provisioned capacity mode, customers must provide an appropriate number of shards needed for the application. However, the number of shards can be dynamically modified using UpdateShardCount API. Please be aware of these limitations while configuring AWS Auto Scaling for Kinesis Data Streams.

How to optimize?

- Monitor your total number of records or data ingested per stream and adjust the number of shards. One shard can process up to 1000 records per second, so monitoring the data ingestion rate will give you the exact number of shards needed. Any unused/idle shard still incurs costs, so identifying and merging those shards is recommended.
- Use provisioned capacity mode for predictable workloads. Provisioned capacity mode is more cost-effective than the on-demand mode. For a given uniform data ingestion rate, the on-demand mode can be many times more expensive than provisioned mode. However, the provisioned capacity mode has added overhead of monitoring and scaling shards as demands

change. AWS auto-scaling can be used to automate Kinesis shard scaling for provisioned mode. Please note that Kinesis on-demand mode has autoscaling built-in.

- For spiky workloads during a specific time of the day, consider switching capacity modes to on-demand during spikes and provisioned capacity modes during consistent hours. Please note that switching modes are limited to twice per 24-hour period.
- Avoid hot or cold shards and evenly spread the load across all shards using appropriate partition keys. Enhanced shard-level monitoring would give you visibility into the incoming traffic. You can then merge any cold shards or split hot shards to ensure every shard is working to its max capacity. On-demand mode automatically does this for customers by merging cold shards or splitting hot shards.
- Avoid enhanced fan-out if possible. For example, an enhanced fan out gives dedicated 2 MB/Sec throughput per shard, it also costs a minimum of $8.43 per shard per day (@$0.05/GB in US-East-1 or US-West-2). However, enhanced fan-out is recommended for applications with many consumers for a single data stream with varying read rates. Otherwise, consider MSK as an option for many enhanced fan-out subscribers.
- Use Kinesis record aggregation to pump more data with a 1000 records/sec/shard limit. This can offer a significant cost advantage, especially if you have many small files (<1 KB).
- Kinesis data stream charges for data storage longer than 24 hours. While retaining data for up to 365 days is possible, KDS is not meant for long-term data storage. Without any business or regulatory requirement, data retention policies should be minimized per the application needs.

Firehose

Kinesis Firehose and Data Analytics are serverless and fully managed services, so you don't have to worry about scaling. However, you must pay attention to the quotas(limits) for Firehose and Data Analytics. Please also note that increasing the Firehose quota significantly more than your actual utilization can increase costs for the destination services. For example, Firehose will deliver data in smaller batches to the destination, which would impact API costs to your downstream destination, such as S3.

For Apache Flink applications, you can use the application autoscaling. For autoscaling based on custom metrics, you can use the custom autoscaling policy of Amazon CloudWatch.

How to optimize?

Firehose

- Kinesis data ingestion is billed with 5 KB increments. Therefore, please use PutRecordBatch API or aggregatedRecordSizeBytes (if you use the Kinesis agent) to keep the record size =< 5 KB.
- Since you are billed for S3 request costs, minimize those requests by increasing the buffer (typically, the buffer should be greater than the amount of data you ingest into the delivery stream in 10 seconds). The Max buffer size is 128 MB.
- Similar to buffer size, increase buffer interval as much as possible (depending on real-time or near real-time requirements). The Max buffer interval is 15 minutes.
- Compress incoming data into Parquet or ORC before delivering to the destination to save on storage costs. The default algorithm is SNAPPY, which prioritizes decompression speed rather than compression ratio,

Managed Service for Apache Flink

- For Managed Service for Apache Flink, KPU usage can vary significantly based on data volume, velocity, and complexity. Test your applications with a production-like load to calculate KPU requirements.
- While simple stateless applications can deliver 100s of MB/sec/KPU throughput, with complex machine learning algorithms, the throughput can be much less (to the tune of <1MB/sec/KPU). Consider this complexity variable while estimating cost – testing each application for cost is the most accurate way.
- Consider your need for running analytics on streaming, real-time data. If real-time is not a must-have requirement, using Kinesis Streams with S3 and Athena would be a lower-cost alternative.
- If you use Data Analytics for machine learning purposes such as anomaly detection, experiment with different values for numberofTrees, shinglesize, etc. More precise computation increases the cost (in terms of KPU), and at some point, that return on investment flattens out. Suppose real-time anomaly detection is not required; you can also use OpenSearch (near real-time), Quicksight, or Sagemaker for Anomaly detection, all of which use RCF, a Sagemaker algorithm for anomalous data points.

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Amazon MSK provisioned has two options to expand cluster storage in response to increased usage – automatic and manual scaling. Increased usage may occur due to increased throughput from producers and data retention requirements. The size or family of brokers is adjusted to the MSK cluster's compute capacity based on changes in workloads without interrupting cluster I/O. Amazon MSK Serverless is an option for applications with intermittent or unpredictable workloads. Vertical scaling can be increased or decreased, but this will not work if the number of partitions per broker exceeds the maximum number specified for the new instance size.

- **Automatic Scaling:** Amazon MSK can automatically scale using an auto-scaling policy where the target disk utilization and maximum scaling capacity are set. It is important to note that a storage scaling action can occur only once every six hours. Verify if the region you are using MSK supports automatic scaling in the Automatic Scaling documentation. Manual scaling can be used if the chosen region is not listed.
- **Manual Scaling:** Amazon MSK can scale up broker storage via the Console or CLI. It is important to note that storage scaling has a cool-down period of at least six hours between events. Even though the operation makes additional storage available immediately, the service performs optimizations on your cluster that can take up to 24 hours or more. The duration of these optimizations is proportional to your storage size.
- **MSK Serverless:** The Serverless is an option for Amazon MSK's cluster type, where it automatically provisions and scales capacity without thinking about right-sizing or scaling. Consider using a serverless cluster if your applications need the on-demand streaming capacity that scales up and down automatically. However, serverless performance is directly affected by the number of partitions, so partitions must be created accordingly to match the number of producers and consumers.

How to optimize?

Right size broker instances

To choose the right size and number of active broker instances and storage used, work backward to determine the use-case's throughput, availability, durability, and latency requirements. Begin with an initial sizing of the cluster based on throughput, storage, and durability requirements. Next, scale horizontally or use provisioned throughput to increase the write throughput of the cluster. Finally, scale vertically to increase the number of consumers that can consume from the cluster or

to facilitate in-transit in-cluster encryption. It's also important to consider the disaster recovery strategy required and if the clusters should reside in 2 or 3 availability zones.

It's vital to understand that the number of brokers cannot be decreased after the cluster is created. When designing for high availability, determine how many availability zones the brokers need. With Amazon MSK, you can only scale in units that match your number of deployed AZs - if you deploy with 3 AZs, you can only scale up by three brokers at a time

Choose the broker type based on the maximum number of partitions required. If the number of partitions per broker exceeds the maximum value, the following operations will not be allowed on the cluster:

- Updating the cluster configuration
- Updating the Apache Kafka version for the cluster
- Updating the cluster to a smaller broker type

The kafka.t3.small broker type for Amazon MSK is ideal for getting started with Amazon MSK, looking to test in a development environment, or having low throughput production workloads.

For guidance on choosing the correct number and type of brokers, see right-size your cluster in the documentation. In addition, an MSK Sizing and Pricing spreadsheet can be used to determine the correct number of brokers and understand the cost breakdown.

Optimize storage

To optimize for storage, consider that the broker storage cannot be decreased, only increased. It is also important to choose the right storage volume type. Amazon MSK does not reduce cluster storage in response to reduced usage and does not support decreasing the size of storage volumes.

Amazon MSK can be configured to expand your cluster's storage automatically in response to increased usage using Application Auto-Scaling policies. Your auto-scaling policy sets the target disk utilization and the maximum scaling capacity. The alternate method is manual scaling through the console or Command Line Interface (CLI). If it is a hard requirement to reduce the size of the cluster storage, you migrate the existing cluster to a cluster with smaller storage.

When choosing the storage volume type, consider the required IOPS and throughput for the workload. If necessary, customers can choose to attach multiple volumes to a broker to increase the throughput beyond what can be delivered by a single volume. See Best practices for right-sizing your Apache Kafka clusters to optimize performance and cost for more information.

It is recommended to adjust data retention policies and delete inactive segments of logs. Consuming messages doesn't remove them from the log. To free up disk space regularly, customers can specify a retention period, which dictates how long messages stay in the log. To avoid running out of disk space for messages, create a CloudWatch alarm at a threshold of 80% that watches the KafkaDataLogsDiskUsed metric. When the value of this metric reaches or exceeds 80%, use automatic scaling, reduce the message retention period, and delete new topics.

Reduce throttling on underlying infrastructure and increase throughput

By default, Amazon MSK exposes three metrics that indicate when this throttling is applied to the underlying infrastructure. These can be used for continuous optimization of your Amazon MSK clusters.

- **BurstBalance** - Indicates the remaining balance of I/O burst credits for EBS volumes. If this metric starts to drop, consider increasing the size of the EBS volume to increase the volume baseline performance. If Amazon CloudWatch isn't reporting this metric for your cluster, your volumes are larger than 5.3 TB and no longer subject to burst credits.
- **CPUCreditBalance** - Relevant for brokers from the T3 family and indicates the number of available CPU credits. When this metric drops, brokers consume CPU credits to burst beyond their CPU baseline performance. Consider changing the broker type to the M5 family.
- **TrafficShaping** - A high-level metric indicating the number of packets dropped due to exceeding network allocations. Finer detail is available when the PER_BROKER monitoring level is configured for the cluster. Scale up brokers if this metric is elevated during your typical workloads.

Monitor the load and CPU Utilization

Amazon MSK strongly recommends that you maintain the total CPU utilization for your brokers (defined as CPU User + CPU System) under 60%. However, suppose the CPU Utilization trends are greater than 60%. In that case, it is recommended to scale up for performance efficiency, but this may increase cost due to the additional broker instance that will be provisioned. Instead, you can use Amazon CloudWatch metric math to create a composite metric and set an alarm that gets triggered when the composite metric reaches an average CPU utilization of 60%. When this alarm is triggered, there are three options:

- The recommended approach is to update the broker type to the next larger type (vertical scaling). However, it is essential to note that the brokers will go offline in a rolling fashion and temporarily reassigns partition leadership to other brokers.

- If there are topics with messages not keyed and ordering is not necessary to consumers, add brokers to expand the cluster and add partitions to existing topics with the highest throughput. This option should be used if the granularity of managing resources and costs is required. In addition, this option should be used if the CPU load is significantly greater than 60%, as it does not result in an increased load on existing brokers.
- Expand clusters by adding brokers and reassigning existing partitions using the partition reassignment tool (Kafka-reassign-partitions.sh). However, this is not recommended if CPU utilization is >70% because replication causes additional CPU load and network traffic.

Storage

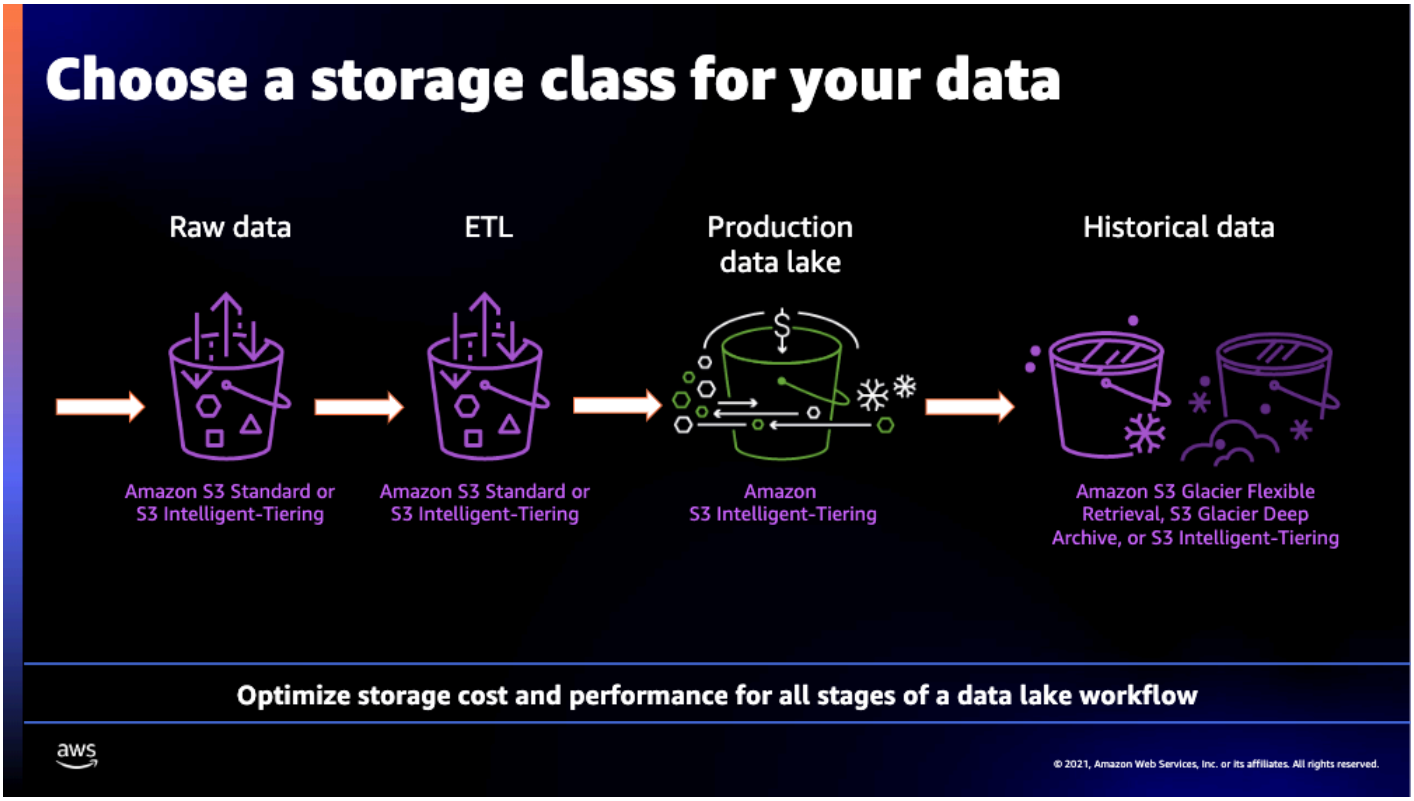
Amazon Simple Storage Service

How to optimize

Storage Classes

S3 storage classes – S3 offers a range of storage classes designed for different use cases. Choosing a suitable storage class can help you be more performant and save a lot of costs. Considerations when choosing a storage class

- **Predictable access patterns** - Datasets such as medical records, media streaming, learning resources, and user-generated content such as photographs are frequently accessed during a specific time and then rarely accessed. For use cases like these, you can plan (lifecycle policies) to move to a lower-cost storage class optimized for infrequent or archive access once the usage of these objects reduce. For predictable workloads, leverage S3 Storage Class Analysis to monitor access patterns across objects and decide when to transition data to the proper storage class to optimize costs.
- **Unpredictable access patterns** - Datasets such as long tail data, data lakes, and data analytics have unpredictable usage patterns. The access patterns for these use cases are highly variable over the year. They can range from little to no access to data being read multiple times in a month or even a single day, leading to high retrieval charges if stored in infrequent access or archive storage class. Use S3 intelligent tiering for such workloads.



Optimize storage cost and performance for all stages of a data lake workflow

- **Cheaper storage class does not guarantee lower S3 costs** - Along with the storage fee, there is a fee for data retrieval, minimum duration, and API calls for a few storage classes. For example, S3 Glacier Instant Retrieval is the ideal storage class if you access data once per quarter. However, although the storage price is lower than S3 Standard-IA, there is an added cost to access the data. Therefore, there is a break-even point where if you're storing data that is accessed too frequently, it makes sense to keep that in S3 Standard-IA (cheaper storage and no retrieval fee). In addition, S3 Glacier Instant Retrieval has a minimum storage duration of 90 days. So, if you upload a file you expect to delete within 90 days, you will be charged a prorated early deletion fee. We recommend you choose the S3 Standard or the S3 Standard-IA storage classes in such cases.

You should upload directly to S3 Glacier if you know the data is accessed not more than once a quarter.

Refer to Amazon cost optimization for more details on Amazon S3 cost optmization

Amazon S3 lifecycle policies

A lifecycle policy automatically move objects in your bucket from one storage class to another. Regular data cleanup into distinct storage classes is crucial in the big data processing. Data lakes have different data layers like source, raw, process, and business data. Once the data is processed, the source and raw data layers are rarely accessed. As a result, transitioning them to infrequent and archive storage class could save money. You can avoid the manual cleanup process with lifecycle policies to save on costs.

Create a lifecycle rule to automatically transition your objects to Standard-IA storage class, archive them to Glacier storage class, and remove them after a specified period.

Lifecycle transition costs are directly proportional to the number of objects transitioned, so reduce the number of objects by aggregating or zipping and compressing them before moving them to archive tiers. Apply life cycle policies to the source, staging, and other data sets that are processed and would no longer be needed.

Create lifecycle policies to clean up incomplete multipart uploads. Also, you can organize data to make life cycle policies simpler and more effective. For example, keep all source data under one prefix so you can have a single lifecycle policy for all the source data sets rather than one for each source prefix. In addition, use lifecycle policies to delete many objects to avoid S3 API costs. Finally, if your bucket is versioned, ensure there is a rule action for both current and non-current objects to either transition or expire.

Type of encryption

Choosing appropriate server-side encryption techniques: S3 encryption helps you protect data against cybercriminals; this is especially important for sensitive data. You can encrypt an object using either Amazon S3-managed keys (SSE-S3), AWS KMS keys stored in AWS Key Management Service (AWS KMS) (SSE-KMS) or using customer-provided keys (SSE-C).

SSE-KMS is for users who want more control over the encryption keys. Enable Amazon S3 bucket keys while using SSE KMS. Bucket Keys decrease the number of transactions from Amazon S3 to AWS KMS, thereby reducing the AWS KMS costs by up to 99 percent. Enabling bucket keys could also help avoid throttling on the KMS limits.

SSE-S3 is the server-side encryption for Amazon S3. There are no additional fees for using server-side encryption with Amazon S3-managed keys (SSE-S3). Amazon S3 encrypts each object with a unique key and encrypts the key with a key that it rotates regularly.

With server-side encryption with customer-provided keys (SSE-C), users manage the encryption keys and Amazon S3 manages the encryption, as it writes to disks, and decryption, when you access your objects.

Client-Side Encryption: You could also choose to encrypt data client-side and upload the encrypted data to S3. In this case, users manage the encryption process, the encryption keys, and related tools.

Data transfer out

As of today, AWS does not charge for data transfer in, data transferred between S3 buckets and to any AWS service within the same region, and data transfer out to Amazon Cloudfront. Different regions have different data transfer-out costs. Choose your region wisely. If possible, deploy your workloads within the same region to avoid inter-region data transfer fees.

Amazon S3 Storage lens

Amazon S3 storage lens is an analytics solution that provides visibility into object storage with point-in-time metrics, trend lines, and actionable recommendations. The Amazon S3 Storage lens will help you discover anomalies and identify cost efficiencies

File formats

Choose appropriate file formats. The file type, compression strategy, and partitioning you use on S3 will significantly impact performance and cost. Storing data in its raw format takes up a lot of space, is inefficient, and is expensive. Instead, use file formats such as parquet, Avro, and orc to speed up read/writes and compress data, using less storage space. Parquet and ORC are suitable for columnar storage and frequent reads over writes; they also compress better than Avro. On the other hand, Avro is suitable for row-based storage and is faster for frequent writes than reads.

Batch operations

S3 Batch operations is a managed service that lets you manage billions of objects at scale. You can change object metadata and properties, copy or replicate objects between buckets, replacing object tag sets, modify access controls and restore archived objects from the S3 glacier. You need not write code, set up servers (compute charges) or figure out how to partition the loads between servers, worry about S3 throttling and pay no compute charge because AWS will handle all these tasks for you.

Processing and analytics

Amazon EMR

There are four deployment options for Amazon EMR:

- Amazon EMR on Amazon EC2
- Amazon EMR on Amazon EKS
- EMR Outposts
- EMR Serverless

Amazon EMR on Amazon EC2 we scale by the cluster. You can define an Amazon EMR cluster with the number of nodes supporting the capacity needed for your workload. To right-size your Amazon EMR cluster, refer to the Cluster configuration and guidelines best practices. You can start with a smaller number of core nodes and increase/decrease the number of task nodes to meet your workload requirements. You can also use the Amazon EMR Managed Scaling or custom autoscaling policies on Amazon EMR. You can specify the minimum and the maximum number of nodes and let Amazon EMR adjust the number of nodes it needs as it optimizes cost and performance.

Amazon EMR on Amazon EKS scales by the job corresponding to the number of vCPU and Memory allocation.

You can create and run Amazon EMR clusters on AWS Outposts. Amazon EMR on AWS Outposts is ideal for low latency workloads running close to on-premises data and applications. Scaling could be challenging due to the limited availability of Amazon EC2 on Outpost. S3 is the only supported option for Amazon EMR on Outposts. S3 on Outposts is not supported for Amazon EMR on AWS Outposts. EMR on outpost could get expensive since spot instances are not available.

Amazon EMR Serverless is a serverless managed service. You need not worry about scaling. Serverless can be cost-effective for cases where you have unpredictable workloads.

How to optimize

This whitepaper primarily focuses on optimization techniques for Amazon EMR on Amazon EC2 and Amazon EMR on EKS. The following are the considerations.

EMR on EC2

Type of infrastructure matching your workload

Choose Amazon EC2 if you know your job profile. If you plan to create a multi-tenant environment where multiple teams and software versions are required to run your applications, use Amazon EKS. If your workload is adhoc or you are unsure of what infrastructure capacity it needs, use Amazon EMR Serverless. AWS Outpost if you need to deploy in a low latency hybrid environment or if you have any restrictions requiring Amazon EMR to run in your own data center.

Spot instances

In an Amazon EMR cluster, you have the master, core, and task nodes. The Master and Core nodes are necessary for any job processing; therefore, you either need to run this on-demand, with different purchase options and cost savings, which we will discuss later. Spot instances allow you to use unused EC2 capacity at a discounted price. Still, they have the possibility of being interrupted after a two-minute of notice when Amazon EC2 needs to reclaim its capacity back. The Task node is an ideal candidate for spot instances as it reduces the cost of failure since this node does not have the data. However, when a spot instance terminates, Amazon EMR has to reallocate nodes. Since the core nodes have HDFS, Amazon EMR needs significant rework on Hadoop replication.

The Amazon EMR Amazon EC2 cluster has an Instance Fleet configuration, allowing you to use various spot instances in provisioning EC2. For example, if you use an Allocation strategy for instance fleets and create a cluster using the AWS CLI or the Amazon EMR API, you can specify up to 30 Amazon EC2 instance types per instance fleet. In the allocation strategy, you can have two options on how Spot instances will be allocated. First is the spot pool, which checks which Availability Zone has the largest capacity for spot instances. Second is the on-demand, where it checks for the lowest price.

Reserved instances and Savings Plan

Depending on the workload, RIs for OnDemand instances can be a reasonable cost-saving strategy. As of today, you cannot purchase RIs for a few hours a day, so you need to plan to use RIs for 24 hours a day for x days for maximum benefits. You can re-purpose the instances for other workloads when you finish running your Amazon EMR workloads. There is a break-even point in switching to RI. If you use your cluster at least 70% of the time, that is the break-even point. And the number can change on the type of instance.

Savings Plans is a flexible pricing model offering lower prices than On-Demand pricing in exchange for a specific usage commitment (measured in $/hour) for one or three years. AWS offers three

types of Savings Plans – Compute Savings Plans, Amazon EC2 Instance Savings Plans, and Amazon SageMaker Savings Plans. Compute Savings Plans apply to usage across Amazon EC2, AWS Lambda, and AWS Fargate. The Amazon EC2 Instance Savings Plans apply to Amazon EC2 usage, and Amazon SageMaker Savings Plans apply to Amazon SageMaker usage. Use Cost explorer to sign up for a 1- or 3-year Savings Plan and manage your plans by taking advantage of recommendations, performance reporting, and budget alerts.

Considerations for using Graviton

Amazon EMR now supports Graviton instance types. On Graviton2 instances, Amazon EMR runtime for Apache Spark provides an additional cost savings of up to 30% and improved performance of up to 15% relative to equivalent previous generation instances. Additionally, for Apache Spark, TPC-DS3 TB benchmark queries run up to 32 times faster using Amazon EMR runtime. Refer to the supported instance types.

Storage using EMFRS

Choose wisely between HDFS and EMRFS, and both have their benefits. While EMRFS provides scalability, durability, and persistent storage, performance might take a hit due to higher latency in Ec2/S3 and back. HDFS offers better performance. It's best used for caching the results produced by intermediate job-flow steps.

Use the latest releases of EBS Volumes. The latest volumes usually provide better IOPS and offer more price performance compared to previous releases.

Amazon S3 data transfer charges

Avoid Amazon S3 cross-region data transfer charges by creating the same region's Amazon EMR clusters and S3 buckets.

Job optimization

Configure jobs for multiple checkpoints and rerun from the last successful checkpoint. This reduces the cost of the duration of the rerun, thereby reducing the cost of failure;

Monitoring and right-sizing

Choosing the right type and then the proper sizing of Amazon EC2 is critical. First, you must understand your workload's characteristics. Is your job memory intensive or CPU intensive? What are your storage needs? You then can choose from a variety of instances offered by AWS. Each

Amazon EC2 instance type has a specific purpose and price, C for compute-optimized, R for memory-optimized, and M/T for general purpose.

Amazon EMR provides several tools you can use to gather information about your cluster. You can access information about the cluster from the console, the CLI, or programmatically. The standard Hadoop web interfaces and log files are available on the master node. You can choose to archive the application and cluster logs on Amazon S3. You can also use monitoring services such as CloudWatch and Ganglia to track the performance of your cluster. Application history is available from the console using the "persistent" application UIs for Spark History, persistent YARN timeline server, and Tez user interfaces. These services are hosted off-cluster, so you can access application history for 30 days after the cluster terminates.

Elasticity

You can automatically or manually adjust the number of Amazon EC2 instances available to an Amazon EMR cluster in response to workloads with varying demands. To use automatic scaling, you have two options. EMR Managed scaling or custom automatic scaling policy. Each scaling has one type of application (spark, hive, or YARN). The metric you choose to monitor and the rate of scale in and out are vital factors in choosing one over the other.

Terminating unused clusters

You can now specify the number of idle hours and minutes, after which the cluster can auto-terminate. The default idle time is one hour. Enable termination protection to ensure Amazon EC2 instances are not shut down by an accident or error. In addition, termination protection is beneficial if your cluster might have data stored on local disks that you need to recover before the instances are terminated.

EMR on EKS

You can leverage Amazon EMR on Amazon EKS as a deployment option to run Apache Spark applications on Amazon EKS cost-effectively. EMR on Amazon EKS can provide up to 61% lower costs and up to 68% performance improvement for spark workloads. It includes multiple performance optimization features, such as Adaptive Query Execution (AQE), dynamic partition pruning, flattening scalar subqueries, bloom filter join, and more.

You could use Spot instances and Arm-based Graviton E2 instances to optimize costs further. Leverage graceful executor decommissioning feature that makes Amazon EMR on Amazon EKS workloads more robust by enabling Spark to anticipate Spot Instance interruptions. Amazon EMR

on Amazon EKS can further reduce job costs via critical stability and performance improvements without recomputing or rerun impacted Spark jobs.

Finally, you can use familiar command line tools such as kubectl to interact with a job processing environment and observe Spark jobs in real-time, reducing development time.

AWS Glue

The main factors that influence the cost for AWS Glue are:

Number and size of jobs

With AWS Glue ETL jobs, you only pay for the time your ETL job takes to run. There are no resources to manage, no upfront costs, and you are not charged for startup or shutdown time. You are charged hourly based on the number of Data Processing Units (DPUs) used to run your ETL job. A single Data Processing Unit (DPU) provides four vCPU and 16 GB of memory. AWS bills for jobs and development endpoints in increments of 1 second, rounded up to the nearest second. Depending on the type of job that you are running, Glue may require a minimum number of DPUs to be provisioned and used to run the job. For example, Apache Spark and Spark Streaming jobs require a minimum of 2 DPUs. By default, Glue will allocate 10 DPU to each Apache Spark job and 2 DPU to each Spark streaming job. Jobs using AWS Glue version 0.9 or 1.0 have a 10-minute minimum billing duration, while jobs that use Glue versions 2.0 and later have a 1-minute minimum. For Python shell jobs, you can allocate either 0.0625 DPUs or 1 DPU. By default, Glue will allocate 0.0625 DPU, and the jobs have a 1-minute minimum billing duration.

AWS Glue Data Catalog

With AWS Glue Data Catalog, you are charged based on the storage of objects and the requests made to the Data Catalog. An AWS AWS Glue Data Catalog object is a table, table version, partition, or database. The first million objects stored are free, and the first million requests per month are free.

Glue crawlers

For Glue crawlers, there is an hourly rate based on the crawler runtime to discover data and populate the AWS Glue Data Catalog. You are billed in increments of 1 second, rounded up to the nearest second, with a 10-minute minimum duration for each crawl. Use of AWS Glue crawlers is optional, and you can populate the AWS AWS Glue Data Catalog directly through the API.

How to optimize?

File sizing and formatting

The first step to optimizing your Glue ETL and crawl jobs is ensuring your source data is formatted correctly. For Spark jobs, it is important that your data set is splittable or can fit into the Spark executor memory. Standard file formats to choose from are Apache Parquet or Apache ORC, as they are splittable and have columnar formats. Additionally, you should ensure high-quality data to avoid Glue crawlers misinterpreting data or having incorrect columns in the data. You can also use Glue DataBrew to clean and normalize data. Glue DataBrew is a feature of AWS Glue that can help reduce the time it takes to prepare data and make intelligent suggestions to help fix data quality issues. It is important to note that Glue DataBrew can incur costs based on the number of nodes used to run your Glue DataBrew job. Therefore, having optimally sized and formatted files can reduce the processing time for Glue jobs and consequently reduce the cost of running Glue.

Combine small files into larger files

It takes longer to crawl a large number of small files than a small number of large files. The reason is the crawler must list each file and read the first megabyte of each new file. However, suppose you know the pattern or schema of the data being crawled, and the data is stored in Amazon S3. In that case, you can combine multiple files by setting the groupFiles and groupSize parameters (https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-etl-connect.html#aws-glue-programming-etl-connect-s3, https://docs.aws.amazon.com/glue/latest/dg/grouping-input-files.html). By combining smaller files into larger files, you can reduce your crawler's runtime and costs.

Glue crawler configurations

Enable your Glue crawlers to use incremental crawls so that you crawl only new subsets of data, and your crawler will not extraneously perform data discovery on files that have already been crawled. The incremental crawls will allow for shorter Glue crawls that will reduce cost. Additionally, it would help if you only run your AWS Glue crawlers when you are aware of new schema changes in your dataset; your AWS Glue crawler should not constantly be running.

Choose specific files or partitions of data to scan by using include paths and exclude patterns. A crawler will start by evaluating the required included path, and for Amazon S3, MongoDB, Amazon DocumentDB, and relational data stores, you must specify an exclude path. Exclude paths will tell the crawler to skip specific files or paths and can reduce the number of files a crawler needs to list. The specific path to scan can make the crawler run faster and reduce cost.

You can run multiple crawlers simultaneously instead of having one crawler running on the entire data store. AWS Glue version 3.0 or later supports autoscaling for AWS Glue ETL and streaming ETL jobs (excludes Python shell jobs). When crawlers are run in parallel, job runtime can be shortened, which lowers the cost of running Glue jobs.

Data partitioning

Use data partitioning to improve performance on your Glue jobs. By default, DynamicFrames are not partitioned when written. However, DynamicFrames now supports native partitioning using the partitionKeys option. You can also use pushdown predicates to filter on partitions without having to list and read all the files in the dataset. Data partitioning allows you to list and read only what you need into a DynamicFrame, instead of reading the whole dataset and filtering inside the DynamicFrame. These functionalities allow you to restrict and reduce what is read into a DynamicFrame, reducing the total data processing done.

Job monitoring and efficient data processing

Monitor your Glue jobs using job metrics in AWS Glue to help you plan for DPU capacity for your Glue jobs and determine the optimal DPU capacity. AWS Glue also has a feature called AWS Glue job run insights that simplifies job debugging and optimization for your Glue jobs. In addition, AWS Glue provides the Spark UI and CloudWatch logs and metrics to monitor your Glue jobs. With Glue job run insights, you get information about your job executions, including the line number where a Glue job script fails, root cause analysis, and recommended actions. For example, the last Spark action executed before a job failure.

Improve your data processing for large datasets by using Workload partitioning with Bounded Execution to reduce common errors like inefficient Spark scripts, distributed in-memory execution of large-scale transformations, and dataset abnormalities. Again, this is supported for S3 data sources only.

Shuffling is essential in Spark jobs whenever data is rearranged between partitions. This is because wide transformations like join, groupByKey, reduceByKey, and repartition require information from other partitions to complete processing. Spark will gather the required data from each partition and combine it into a new partition. With Glue 2.0, you can use Amazon S3 to store Spark shuffle and spill data, which will disaggregate compute and storage for your Spark jobs and give complete elasticity and low-cost shuffle storage.

Use job bookmarks to track data that has already been processed in previous runs of an ETL job. Job bookmarks will help AWS Glue maintain state information and prevent reprocessing old data, reducing the amount of data scanned and compute resources utilized, therefore reducing costs.

Minimize usage of development endpoints

Development endpoints are environments that can be used to develop and test AWS Glue scripts. These are optional to use and have an additional cost associated with them. Alternatively, you can develop your jobs locally to save on endpoint costs.

Lambda

The first time you invoke a Lambda function, it creates an instance of the function to process the event. After that, the instance stays active to process additional events after finishing the response. If you invoke the function again before the first event is complete, Lambda initializes another instance, and the function processes the two events concurrently. After that, Lambda creates new instances as needed as more events come in and drop instances if the demand reduces.

A function's concurrency is the number of instances that can serve requests at any time. The concurrency varies by region, and you can increase the limit using AWS Service Quota Dashboard. You can find more details in the Lambda Function Scaling.

How to optimize?

Right-sizing

- Use Compute Optimizer and Lambda Power tuner to right-size the functions and memory configurations, especially after new deployments. Compute Optimizer would take days to gather historical data. Lambda Power tuning can be executed right away and will assist in prioritizing cost vs. execution time.
- Remove unneeded Codes and Libraries to reduce function size. Use Lambda layers to share code and libraries used from multiple functions. This would help reduce memory footprint and, hence, the GB per second cost.
- There might be some use cases where Lambda is not the most efficient solution. For example, you might want to evaluate some alternatives for long-running jobs, large payload (> 6 MB), or scheduled workloads. One such use case is covered in this document. AWS Batch is another alternative for scheduled jobs.

Performance efficiency

- A 128 MB function that runs for 10 seconds costs more than a 1 GB function that runs for 1 second, so less memory is not always cheaper. A critical aspect of right-sizing is comparing

execution duration vs. the cost of increasing memory. Ensure Compute Optimizer has one to two weeks of data before executing the recommendations.

- Use X-Ray and CodeGuru to identify bottlenecks and expensive Lambda codes. These tools are not free but can provide significant value in reducing costs and identifying bottlenecks. For example, use X-Ray analytics to identify and review subsets of requests with higher latency. Amazon CodeGuru uses machine learning for automated code review and application performance profiling to identify the most expensive lines of code. It also provides recommendations to improve code quality. Typically, you want to use X-ray in production and CodeGuru in Development environments.
- Use CloudWatch Lambda Insights for a deeper understanding of usage and bottlenecks/saturation points, if any. However, shipping custom metrics to CloudWatch using put-metrics needs a synchronous blocking call which adds to the idle wait time. Use Embedded Metric Format (EMF) for asynchronous calls instead.
- It is important to maximize the amount of time your application spends processing. To minimize the time it is waiting on other resources, identify areas where one Lambda function is waiting for a response from another lambda or other downstream AWS service. Avoid using Lambda as an orchestrator. Instead, use dedicated orchestration tools like Step functions, which can reduce Lambda execution times waiting for a response from another Lambda function or other AWS services. It would also avoid Lambda idle wait times. Additionally, try to break down a Lambda into smaller functions to reduce I/O waits.
- Identify Lambda functions as a proxy to AWS SDK calls. You can replicate those functions by direct SDK integrations in the workflow/step functions or API gateway/Eventbridge/AppSync.
- You can reduce the number of invocations by effectively using Event filtering. Currently, you can use event filtering with Kinesis, DynamoDB, and SQS event sources. A Lambda function will be executed only when a particular event pattern is matched. This also eliminates the need to use filtering logic in the application code.
- Watch out for infinite loops between your Lambda function and S3 bucket and avoid this recursion problem, as it can significantly hit your Lambda invocation cost.

Operational efficiency

- Whenever possible, use AWS Graviton2 for your functions, which provides 34% better price/performance. Popular runtimes such as Node.JS, Python, Ruby, and Java are already supported on the Graviton2 platform. Use Lambda Power tuning to compare costs between x86 and Graviton2. You can upgrade to Graviton2 in place without redeploying (but testing is required).

- For a consistent pattern of traffic, use Lambda provisioned concurrency. This will help reduce Lambda cold starts and avoid burst throttling. Provisioned concurrency can result in ~16% cost savings if fully utilized. Additionally, you can use provisioned concurrency in conjunction with AWS autoscaling to adjust automatically for peaks and troughs.
- Be aware of log storage costs (in CloudWatch). Use log retention policies for every log group. If you need to retain logs for a long time, move them to S3 and use retention policies.
- Remove logging that does not add value to investigations to save on CloudWatch data ingestion costs and log storage costs. Turn off debug logs in production. Instead, use structured formats such as JSON. Don't use CloudWatch for trace data; X-Ray is a better tool.

Pricing model

- If you have a consistent monthly usage for your Lambda functions, use Compute Savings Plan for your baseline Lambda usage. Using the Savings plan, customers can save up to 17% (applicable to both on-demand and provisioned concurrency). The best practice is to analyze Lambda usage for a consistent monthly baseline and apply provisioned concurrency and savings plans to that baseline usage.
- Customers can now take advantage of Lambda's tiered pricing. There is no action needed to avail of this feature. However, please note that this new pricing model is per region and platform (x86 and ARM64). So, consolidating them into one would maximize the savings if customers are running functions using both platforms.

Amazon Athena

Athena scales automatically to handle queries, so you do not need to worry about the underlying infrastructure and how it scales. Athena will also automatically execute queries in parallel without worrying about managing or tuning clusters.

Athena provides connectors for enterprise data sources, including Amazon DynamoDB, Amazon Redshift, Amazon OpenSearch Service, MySQL, PostgreSQL, Redis, and other popular third-party data stores. Athena's data connectors allow you to generate insights from multiple data sources using Athena's easy-to-use SQL syntax without needing to move your data with ETL scripts. In addition, data connectors run as AWS Lambda functions and can be enabled for cross-account access, allowing you to scale SQL queries to hundreds of end-users.

How to optimize?

Athena configurations

- **Use workgroups** - Workgroups can be used to separate users, teams, applications, or workloads to limit the amount of data each query or the entire workgroup can process and help track costs. Resource-level identity-based policies can control access to a specific workgroup and view query-related metrics in Amazon CloudWatch.
- **Minimize data scans** - Athena does not cache query results, meaning repeated queries will re-scan any data, even if it has been scanned before. Data scans can increase the overall Athena cost as data scanned will directly impact the cost. Avoid running the same query multiple times if the table data hasn't changed.

Optimizing storage

- **Partitioning data** - Partitioning data will divide your table into parts and keep the related data together based on the columns used for partitioning. Data partitioning can reduce the amount of data scanned per query. When deciding which columns to partition, you should consider which columns may be common filters for queries and how granular your partitions are. Run the EXPLAIN ANALYZE query command to determine if a table is partitioned and use a partitioned column as a filter.
- **Compressing data** - Compress your data to speed up queries, as long as the files are either an optimal size (generally greater than 128MB) or splittable. Splittable files can be read in parallel by the execution engine in Athena, taking less time when reading a splittable file. Ideal file types include Avro, Parquet, and ORC, as they are splittable regardless of the compression codec. Only files compressed with BZIP2 and LZO codec are splittable for text files.
- **Optimal file size** - Files should also not be too small as they may add to the overhead of opening S3 files, listing directories, getting object metadata, setting up data transfer, reading file headers, reading compression dictionaries, etc. Amazon S3 and Amazon Athena currently have a limit of 5500 requests per second, so combining smaller files into a larger file is recommended to avoid throttling or excessive scanning.
- **Partition Projection** - With partition projection, partition values and locations are calculated from configuration rather than read from a repository like AWS AWS Glue Data Catalog. Since in-memory operations tend to be faster than remote operations, partition projects can reduce the runtime of queries against highly partitioned tables. Using partition projection is ideal when your

partitions' schemas are the same or if the tables' schema consistently accurately describes the partition's schemas.

- **Storage lifecycle policies** - The results of Athena queries are stored in S3. Configure lifecycle policies on buckets to reduce storage costs for data that is not used.

Performance and Query Tuning

- **ORDER BY** - Athena will use distributed sort to run the sort operation in parallel on multiple nodes, so if you use the ORDER BY clause to look at the top or bottom N values, use a LIMIT clause to reduce the cost of the sort and save on query runtime.
- **GROUP BY** - The GROUP BY operator distributes rows based on the GROUP BY columns to worker nodes, which hold the GROUP BY values in memory. When using the GROUP BY operator, order the columns by the highest cardinality to the lower. You can also reduce the number of columns included in the SELECT clause to reduce the amount of memory required to store if they are unneeded.
- **JOIN Queries**: Athena will distribute the table on the right to worker nodes and stream the table on the left to join. Therefore, when joining two tables, you should specify the larger table on the left side of the join and the smaller one on the right side. However, suppose you are joining three or more tables. In that case, you can consider joining the largest tables with the smallest ones to reduce the intermediate results and then join the remaining tables.

Function efficiency

- **Minimize the use of window functions** - Window functions are memory intensive and, in general, require an entire dataset to be loaded into a single Athena node for processing. Loading large datasets can risk crashing the node. Instead, consider partitioning the data, filtering the data to run the window function on a subset of the data, or finding an alternative way to construct the query.
- **Use efficient functions** - Some queries have multiple ways to construct the same query. Use more efficient methods to reduce processing done on the data set. For example, use regular expressions instead of LIKE on large strings, or reduce nested functions.
- **Use approximate functions** - If you do not require exact numbers, you can use approximate functions to minimize memory usage; approximate functions will count unique hashes of values instead of entire strings, with a standard error of 2.3%.

- **Include only necessary columns** - Limit your final SELECT statements to only include columns that you need to reduce the amount of data that needs to be processed through the entire query run pipeline. Filtering a query is particularly useful when querying large string-based tables that require multiple joins or aggregations.

Amazon Redshift

An Amazon Redshift data warehouse is a collection of computing resources called nodes, which are organized into a group called a cluster. Each cluster runs an Amazon Redshift engine and contains one or more databases. Amazon Redshift scales by the number of clusters and nodes within each cluster.

With RA3 node types, your compute and storage can scale separately. For example, with previous generation nodes DC2 and DS2, you scale compute and storage depending on the size of your instance. In addition, you can resize your cluster as needed depending on changes to your capacity or performance requirements.

Concurrency Scaling is a feature of Amazon Redshift that allows you to add cluster capacity to support concurrent users and queries without resizing your cluster.

How to optimize?

Additionally, some best practices can help you reduce costs and optimize query performance when running queries on Redshift. Redshift also includes a Redshift Advisor that will provide recommendations and insights into optimizing your workloads. We discuss these recommendations are below: table compression, Workload Management (WLM), and cluster resizing.

Choose the right node size

The first step to optimizing your Redshift clusters is choosing the right node type and instance size to meet your requirements. Keep in mind that Redshift can compress your data up to four times. Redshift will provide suggestions for node types when you start using it based on your needs, like CPU, RAM, storage capacity, and availability, but you can quickly scale up or down as needed.

Use Redshift AQUA

Advanced Query Accelerator, or AQUA, is a distributed and hardware-accelerated cache that improves query performance. AQUA is only supported on RA3 node types but is available at no additional charge and with no code changes.

Use Reserved nodes when possible

If you have steady-state production workloads, you can choose to run Reserved nodes, which can offer additional cost savings. These savings come with 1-year or 3-year term lengths with the option to pay all upfront, partial upfront, or no upfront.

Use the Pause and Resume feature for On-Demand nodes

With Redshift on-demand, you pay for what you use based on the storage and compute depending on the node type used to run your cluster. With On-demand clusters, utilize the pause and resume feature when clusters do not need to be running. On-demand billing will be suspended while the cluster is paused, so you will not be charged when the cluster is paused.

Table compression, partitions, and columnar format

Without data compression, data consume additional storage space and utilize additional disk I/O, which are factors that directly affect your Redshift usage and cost. You can use the ANALYZE COMPRESSION command to get suggestions for compression for your data.

Store data in columnar formats like Apache Parquet or ORC and leverage partitions to reduce the amount of data scanned when data is stored in Amazon S3 for Amazon Redshift Spectrum, as the amount of data scanned is directly correlated to the cost.

Compressing Amazon S3 file objects loaded by the COPY command.

The COPY command integrates with the massively parallel processing (MPP) architecture of Amazon Redshift. It allows you to read and load data in parallel from Amazon S3, Amazon DynamoDB, and text output. You can apply compression when you create and load brand new tables or use the COPY command with COMPUPDATE set to ON to analyze and apply compression automatically.

Cluster Resize

Elastic Resize: Elastic Resize lets you quickly add, remove, or change node types from an existing cluster by automating the process of taking a snapshot, creating a new cluster deleting the old cluster, and renaming the new cluster. You can use this process to upgrade to the new RA3 node type from previous generation node types (DC2 or DS2). The process takes about ten to fifteen minutes to complete, and you will not be able to write to the cluster temporarily while the data is transferring to the new cluster. This process automatically redistributes the data to the new nodes.

Classic Resize: Classic Resize is the manual approach to resizing your cluster and is the predecessor of Elastic Resize; Elastic Resize is the recommended option.

Amazon Redshift Spectrum

Amazon Redshift Spectrum allows you to run queries directly against the data you have stored in Amazon S3, and you are charged only for the bytes scanned, rounded up to the next megabyte. You are charged the standard S3 rates for storing objects in your S3 buckets and requests made against your S3 buckets. Additionally, you can implement an “aging-off” process for historical data or less frequently queried data to save costs, as S3 storage is more cost-effective than Redshift’s direct-attached storage. For any hot and frequently accessed data, it is preferable to keep that data in direct-attached storage for performance.

Workload management (WLM) and concurrency scaling

When you have multiple sessions or users running queries simultaneously, some queries may take longer to run or consume more resources than others and affect the performance of other queries. Workload Management can define multiple query queues and route queries to the appropriate queues at runtime to help handle these situations.

There are options for Automatic WLM and Manual WLM. Automatic WLM will automatically manage how many queries run concurrently and how much memory is allocated to each dispatched query. In addition, you can specify the priority of workloads or users that map to each queue. With Manual WLM, you configure the specific queues used to manage queries. You can also set up rules to route queries to particular queues based on the user running the query or specified labels.

Concurrency scaling is cost-effective for spiky workloads where you require additional capacity for short periods instead of provisioning additional persistent nodes in your cluster. To implement concurrency scaling, you can route queries to Concurrency scaling clusters with a workload manager.

Trusted Advisor Recommendations

Trusted Advisor would run checks against your Amazon Redshift resources in your account to notify you about cost optimization opportunities. These include providing recommendations for purchasing Reserved nodes to optimize based on usage and shutting underutilized clusters.

Monitoring data lakes costs

Once the data lake is built to provide its intended features, we recommend that you measure the cost and tie it back to business value it provides. This enables you to perform a return-on-investment analysis on your analytics portfolio. It also enables you to iterate and tune the resources for optimal cost based on cost optimization techniques discussed earlier in this whitepaper.

To track the cost utilization for your analytic workloads, you need to define your cost allocation strategy. Cost-allocation tagging ensures that you tag your AWS resources with metadata key-value pairs that reflect the business unit the data lake pipeline was built for.

Tags enable you to generate billing reports for the resources associated with a particular tag. This lets you to either do charge-back or return-on-investment analysis.

Another strategy to track your costs is to use multiple AWS accounts and manage them using AWS Organizations. In this approach, every business unit owns their AWS account and provisions and manages their own resources. This lets them track all cost associated with that account for their data lake needs.

By tracking costs and tying it back to your business value, you can complete your cost modeling for your first analytics workload. This process also lets you iterate and repeat the process again of deciding business use cases, defining KPI and building data lake features on top of your already built data lake foundation while monitoring the cost associated with it.

Conclusion

Customers struggle with starting their analytics projects because it is difficult to estimate costs when you have no knowledge or foresight of their unique requirements as an organization. Without a cost estimate, projects fail to get funding and organizations miss the enormous value making data driven decisions can offer.

This whitepaper offers a way forward. We have shown how you can tackle this challenge. This involves breaking down the problem into smaller components that can more easily be sized, costed, and implemented, while delivering measurable business value early on in your project.

You can use the scenarios as templates to build out a picture of what your analytics experiment will look like. The scenarios can help your organization start a journey towards making data-based decisions and drive business value, offering benefits for your organization and its customers.

Contributors

Contributors to this document include:

- Charlie Llewellyn, Solutions Architecture (Public Sector), Amazon Web Services
- Anusha Dharmalingam, Enterprise Solutions Architecture, Amazon Web Services
- Jaymalya Biswas, Solutions Architecture, Amazon Web Services
- Jennifer Hu, Solutions Architecture, Amazon Web Services
- Suraj Shivananda, Solutions Architecture, Amazon Web Services
- Carole Suarez, Solutions Architecture, Amazon Web Services
- Marie Yap, Solutions Architecture, Amazon Web Services

Further reading

The following resources can help you get started in running big data analytics on AWS:

- Well Architected for Analytics
- Big Data on AWS
- AWS Big Data Blog
- Big Data Test Drives
- Big Data on AWS
- Big Data Customer Case Studies

Document Revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Update	Added cost optimization techniques for specific AWS analytics services.	March 3, 2023
Initial publication	Whitepaper published.	November 1, 2020

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.