# Analytics Lens

## AWS Well-Architected Framework

# Analytics Lens: AWS Well-Architected Framework

# Table of Contents

# Analytics Lens - AWS Well-Architected Framework

Publication date: **May 2020** (*Document Revisions* (p. 55))

## Abstract

This document describes the Analytics Lens for the AWS Well-Architected Framework. The document covers common analytics applications scenarios and identifies key elements to ensure that your workloads are architected according to best practices.

# Introduction

The AWS Well-Architected Framework helps you understand the pros and cons of decisions you make while building systems on AWS.

By using the framework, you learn architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud. It provides a way for you to consistently measure your architectures against best practices and identify areas for improvement. We believe that having well-architected systems greatly increases the likelihood of business success.

In this "Lens" we focus on how to design, deploy, and architect your analytics application workloads in the AWS Cloud. For brevity, we have only covered details from the Well-Architected Framework that are specific to analytics workloads. You should still consider best practices and questions that have not been included in this document when designing your architecture. We recommend that you read the AWS Well-Architected Framework whitepaper.

This document is intended for those in technology roles, such as chief technology officers (CTOs), architects, developers, and operations team members. After reading this document, you will understand AWS best practices and strategies to use when designing architectures for analytics applications and environment.

# Definitions

The AWS Well-Architected Framework is based on five pillars: operational excellence, security, reliability, performance efficiency, and cost optimization. For analytics workloads and environments, AWS provides multiple core components that allow you to design robust architectures for your analytics applications. In this section, we will present an overview of the services that will be used throughout this document.

Organizing data architectures into conceptual "layers" allows appropriate access controls, pipelines, Extract, Transform, and Load (ETL) flows, and integrations specific to the use case. There are six areas that you should consider when building an analytics workload.

**Topics**



## Data Ingestion Layer

The Data ingestion layer is responsible for ingesting data into the central storage for analytics, such as a data lake. It's comprised of services that aid in consuming datasets in batch and real-time streaming modes from external sources, such as website clickstreams, database event streams, financial transactions, social media feeds, IT logs, location-tracking events, IoT telemetry data, on-premises data sources, and cloud-native data stores.

**Amazon Kinesis** is a family of services for ingesting real-time data and provides capabilities to securely load and analyze streaming data and stream data to Amazon Simple Storage Service (Amazon S3) for

long-term storage. We also provide **Amazon Managed Streaming for Apache Kafka (MSK)**, which is a fully managed service that enables you to run highly available and secure Apache Kafka clusters to process streaming data without the need to modify your existing code base.

With **AWS Database Migration Services (DMS)**, you can replicate and ingest existing databases while the source databases remain fully operational. The service supports multiple database sources and targets, including writing data directly to Amazon S3. To accelerate DMS migrations, you can use **AWS Snowball**, which uses secure physical appliances to transfer large amounts of data into and out of the AWS Cloud. You should also investigate **AWS Direct Connect,** which creates a consistent, private network connection between your data center and AWS.

You may also have additional data ingestion points, such as **AWS IoT Core**, a managed platform that can process and route messages at scale into AWS data stores reliably and securely. **AWS DataSync** is a data transfer service that simplifies, automates, and accelerates moving and replicating data between on-premises storage systems such as NFS and AWS storage services such as **Amazon EFS** and Amazon S3 to be ingested into your analytics workload.

# Data Access and Security Layer

The data access and security layer provides a mechanism for accessing data assets while protecting them to ensure that the data is stored securely and access is provided to only those authorized. This layer enables:

- Secure data access to the central data repository (that is, the data lake)
- Secure access to the central Data Catalog
- Fine-grained access control on the Data Catalog's databases, tables, and columns
- Encryption of data assets in transit and at rest

You must use **AWS Identity and Access Management (IAM)** to manage access to AWS services and resources securely. Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources. With **AWS CloudTrail**, you can log, continuously monitor, and retain account activity related to data access actions from those users and roles across your AWS infrastructure. You can also use **Amazon CloudWatch** to collect monitoring and operational data, in the form of logs, metrics, and events, for your analytics workload.

To provide encryption at rest, use **AWS Key Management Service (KMS)**, a secure and resilient service that makes it easy for you to create and control the encryption keys that encrypt your data. Some regulations require you to pair KMS with **AWS CloudHSM**, a cloud-based hardware security module (HSM) which enables you to easily generate and use your own encryption keys. AWS CloudHSM helps you demonstrate compliance with security, privacy, and anti-tamper regulations, such as HIPAA, FedRAMP, and PCI. You can configure KMS to use your CloudHSM cluster as a custom key store instead of the default KMS key store.

**AWS Lake Formation** is an integrated data lake service that makes it easy for you to ingest, clean, catalog, transform, and secure your data and make it available for analysis and machine learning (ML). Lake Formation provides its own permissions model, which augments the AWS IAM permissions model, to configure data access and security policies for data lakes, and audit and control access from AWS analytic and ML services. This centrally defined permissions model enables fine-grained access to data stored in data lakes through a simple grant/revoke mechanism.

# Catalog and Search Layer

The catalog and search layer of your analytics workload manages discovering and cataloging the metadata pertaining to your data assets. This layer also provides search capabilities as data assets grow in quantity and size—scenarios where you want to find a table based on criteria that you define and extract subsets of data—are quite common in analytics applications.

**AWS Glue** is a fully managed extract, transform, and load (ETL) service that makes it easy for customers to prepare and load their data for analytics. You can point AWS Glue to your data stored on AWS, and it discovers your data and stores the associated metadata (for example, its table definition and schema) in the AWS Glue Data Catalog. Once cataloged, your data is immediately searchable, queryable, and available for ETL.

With **Amazon Elasticsearch Service**, you can deploy fully managed Elasticsearch clusters in the AWS Cloud to search your data assets. You get direct access to the Elasticsearch APIs—existing code and applications work seamlessly with the service and includes managed Kibana, integration with Logstash and other AWS services, built-in alerting, and SQL querying.

**Amazon Relational Database Service (Amazon RDS)** makes it easy to set up, operate, and scale a relational database in the cloud. In addition to AWS Glue, you can use Amazon RDS to create an external Hive metastore for EMR. The metastore contains a description of the table and the underlying data on which it is built, including the partition names, data types, and so on.

**Amazon DynamoDB** is a NoSQL data store that can be used to create a high performance, low-cost external index that maps queryable attributes to Amazon S3 object keys. Amazon DynamoDB automatically scales and remains highly available without the need to maintain traditional servers.

# Central Storage Layer

The central storage layer manages the storage of data as it's ingested from a variety of producers and makes it available to downstream applications. This layer is at the core of a data lake and should support housing of all types of data: unstructured, semi-structured, and structured data. As data grows over time, this layer should scale elastically in a secure and cost-effective manner.

In data processing pipelines, data might be stored at intermediate stages of processing, both to avoid needless duplication of work up to that point in the pipeline, as well as to make intermediate data available to multiple downstream consumers. Intermediate data might be frequently updated, stored temporarily, or stored long term, depending on the use case.

**Amazon S3** provides an optimal foundation for central storage because of its virtually unlimited scalability, 99.999999999% (11 "nines") of durability, native encryption, and access control capabilities. As data storage requirements increase over time, data can be transitioned to lower-cost tiers, such as S3 Infrequent Access or Amazon S3 Glacier, through lifecycle policies to save on storage costs while still preserving the original, raw data. You can also use S3 Intelligent-Tiering, which optimizes storage costs automatically when data access patterns change, without performance impact or operational overhead.

Amazon S3 makes it easy to build a multi-tenant environment, where many users can bring their own data analytics tools to a common set of data. This improves both cost and data governance over that of traditional solutions, which commonly require multiple, distributed copies of the data. To enable easy access, Amazon S3 provides RESTful APIs that are simple and supported by Apache Hadoop as well as most major third-party independent software vendors (ISVs) and analytics tool vendors.

With Amazon S3, your data lake can decouple storage from compute and data processing. In traditional Hadoop and data warehouse solutions, storage and compute are tightly coupled, making it difficult to optimize costs and data processing workflows. Amazon S3 allows you to store all data types in their native formats and use as many or as a few virtual servers as you want to process the data. You can also integrate with serverless solutions, such as AWS Lambda, Amazon Athena, Amazon Redshift Spectrum, Amazon Rekognition, and AWS Glue, that allow you to process data without provisioning or managing servers.

**Amazon Elastic Block Store (EBS)** provides persistent block storage volumes for use with Amazon EC2 instances in the AWS Cloud. Each Amazon EBS volume is automatically replicated within its Availability Zone to protect you from component failure, which provides high availability and durability. For analytics workloads, you can use EBS with Big Data analytics engines (such as the Hadoop/HDFS ecosystem or Amazon EMR clusters), relational and NoSQL databases (such as Microsoft SQL Server and MySQL or

Cassandra and MongoDB), stream and log processing applications (such as Kafka and Splunk), and data warehousing applications (such as Vertica and Teradata) running on EC2 instances.

# Processing and Analytics Layer

The processing and analytics layer is responsible for providing tools and services for querying and processing (that is, cleansing, validating, transforming, enriching and normalizing) the datasets to derive business insights in both batch and real time streaming mode. There are many services that can be used for the processing and analytics layer.

**Amazon EMR** is a managed service to easily run and scale Apache Spark, Hadoop, HBase, Presto, Hive, and other big data frameworks across dynamically scalable Amazon EC2 instances and interact with data in other AWS data stores, such as Amazon S3 and Amazon DynamoDB.

**Amazon Redshift** is a fully managed data warehouse that makes it simple and cost-effective to analyze all your data using standard SQL and your existing Business Intelligence (BI) tools. **Redshift Spectrum** is a feature of Amazon Redshift that enables you to run queries against exabytes of unstructured data in Amazon S3, with no loading or ETL required. Redshift Spectrum can execute highly sophisticated queries against an exabyte of data or more—in just minutes.

**Amazon Athena** is an interactive query service that makes it easy to analyze data in Amazon S3 using standard SQL. Athena is serverless, so there is no infrastructure to manage, and you pay only for the queries that you run. Athena integrates with AWS Glue Data Catalog, allowing you to create a unified metadata repository across various services, crawl data sources to discover schemas, populate your catalog with new and modified table and partition definitions, and maintain schema versioning.

With **Amazon Neptune**, you can create a fast, reliable, fully managed graph database that makes it easy to build and run applications that work with highly connected datasets. It supports popular graph models, such as Property Graph and W3C's RDF, and their respective query languages, Apache TinkerPop Gremlin and SPARQL. Amazon Neptune can power graph relationship use cases, such as recommendation engines, fraud detection, knowledge graphs, drug discovery, and network security.

**Amazon SageMaker** is a fully managed machine learning platform that enables developers and data scientists to quickly and easily build, train, and deploy machine learning models at any scale. Data scientists can use it to easily create, train, and deploy ML models against data lake elements.

Existing services can also be used for processing and analytics, including Amazon Kinesis, Amazon RDS, Apache Kafka, and AWS Glue ETL jobs.

# User Access and Interface Layer

The user access and interface layer provides a secure means for user access and an administrative interface for managing users.

**AWS Lambda** lets you run stateless serverless applications on a managed platform that supports microservices architectures, deployment, and management of execution at the function layer.

With **Amazon API Gateway**, you can run a fully managed REST API that integrates with Lambda to execute your business logic and includes traffic management, authorization and access control, monitoring, and API versioning. For example, you can create a data lake API using API Gateway that receives requests via HTTPS. When an API request is made, Amazon API Gateway leverages a custom authorizer (a Lambda function) to ensure that all requests are authorized before data is served.

With **Amazon Cognito**, you can easily add user sign-up, sign-in, and data synchronization to serverless applications. Amazon Cognito user pools provide built-in sign-in screens and federation with Facebook, Google, and Amazon, using Security Assertion Markup Language (SAML). Amazon Cognito Federated Identities lets you securely provide scoped access to AWS resources that are part of your serverless architecture.

# General Design Principles

The Well-Architected Framework identifies a set of general design principles to facilitate good design in the cloud for analytics applications:

- **Automate data ingestion:** Ingestion of data should be automated using triggers, schedules, and change detection. Automating the collection process eliminates error-prone manual processes, allows data to be processed as it arrives, and allows you to create and replicate your systems at low cost. You can leverage the schedulers available in orchestration tools to schedule and trigger ingestion at periodic intervals or run the ingestion scripts continuously for streaming applications.
- **Design ingestion for failures and duplicates**: Ingestion triggered from requests and events must be idempotent, as failures can occur and a given message might be delivered more than once. Include appropriate retries for downstream calls.
- **Preserve original source data**: Ingested raw data should be preserved as is because having raw data in its pristine form allows you to repeat the ETL process in case of failures. No transformation of the original data files should occur during the pipeline execution.
- **Describe data with metadata**: As datasets tend to grow in variety and volume, it's essential that any dataset that makes its way into a data store environment is discoverable and classified. Capture metadata about the data store to ensure that the downstream applications are all able to leverage the ingested datasets. Ensure that this activity is well-documented and automated.
- **Establish data lineage**: Data lineage refers to tracking data origin and its flow between different data systems. Having the ability to view, track, and manage data flow as it moves from source to the destination can greatly simplify tracing any errors along the analytics pipeline and get insights into how data has evolved.
- **Use the right ETL tool for the job**: When it comes to extract, transform, and load (ETL) tools, there are several options: custom built to solve specific problems, assembled from open source projects, and commercially licensed ETL platforms. Select an ETL tool that closely meets your requirements for streamlining the workflow between the source and the destination. Factors to examine include support for complex workflows, APIs and specific languages, connectors to varied data stores, performance, budget, and enterprise scale.
- **Orchestrate ETL workflows:** Automate ETL workflows. In an analytics environment, output from one process or job typically serves as an input to another. Chaining ETL jobs ensures the seamless execution of your ETL workflow while enabling you to track and debug any failures.
- **Tier storage appropriately:** Store data in the optimal tier to ensure that you leverage the best features of the storage services for your analytics applications. There are two basic parameters when it comes to choosing the right storage service for your data: its format, and how often it's accessed. By distributing your datasets into different services and then transitioning from one service to another, you can build a robust backend storage infrastructure for your analytical applications.
- **Secure, protect, and manage your entire analytics pipeline:** Both the data assets and the infrastructure for storing, processing, and analyzing data must be secured. Securing your data assets begins with implementing fine-grained controls that allow authorized users to see, access, process, and modify particular assets, while ensuring that unauthorized users are blocked from taking any actions that would compromise data confidentiality and security. Access roles might change at various stages of an analytics pipeline, requiring you to ensure that only authorized users have access at each stage.
- **Design for scalable and reliable analytics pipelines:** Make analytics execution compute environments reliable and scalable such that the volume or velocity of data does not impact the production pipelines. Provide high data reliability and optimized query performance to support different analytics applications, from batch and streaming ingests, fast ad hoc queries to data science are top priorities when architecting analytics workflows.

# Scenarios

In this section, we cover the five key scenarios that are common in many analytics applications and how they influence the design and architecture of your analytics environment on AWS. We present the assumptions made for each of these scenarios, the common drivers for the design, and a reference architecture for how these scenarios should be implemented.

**Topics**



# Data Lake

A data lake is a centralized repository that allows you to store all your structured and unstructured data at any scale. You can store your data as-is, without having to first structure the data, and run different types of analytics—from dashboards and visualizations to big data processing, real-time analytics, and machine learning—to guide better decisions.

Organizations that successfully generate business value from their data using a data lake are able to do new types of analytics, such as machine learning on data from log files, click-streams, social media, and internet connected devices. This can help you to identify and act upon opportunities for business growth faster by attracting and retaining customers, boosting productivity, proactively maintaining devices, and making informed decisions.

The main challenge with a data lake architecture is that raw data is stored with no oversight of the contents. To make the data usable, you must have defined mechanisms to catalog and secure the data. Without these mechanisms, data cannot be found or trusted, resulting in a “data swamp.” Meeting the needs of diverse stakeholders requires data lakes to have governance, semantic consistency, and access controls.

**Topics**



## Characteristics

- Regardless of the source type, data structure, or amount, all the original source data should be in one place, creating a “Single Source of Truth”.
- A data lake should always decouple storage and compute.
- A data lake should support quick ingestion and consumption, not only in terms of speed, but also to allow flexibility in the data design. Data providers should only have to be told where to put the data, that is, which S3 bucket. The choice of storage structure, schema, ingestion frequency, and data quality should be left to the data producer.

- A data lake supports schema on read. Multiple schemas can be used when ingesting data from a data lake into a compute environment, unlike the fixed structure of records data stored within a data warehouse.
- A data lake should be designed for low-cost storage. This allows historical data to be kept longer at lower cost, as its overall business value declines over time.
- A data lake supports protection and security rules, which provide mechanisms to allow only authorized access to the data, and tracking how data is used as it flows through tiers within the lake.

# Reference Architecture

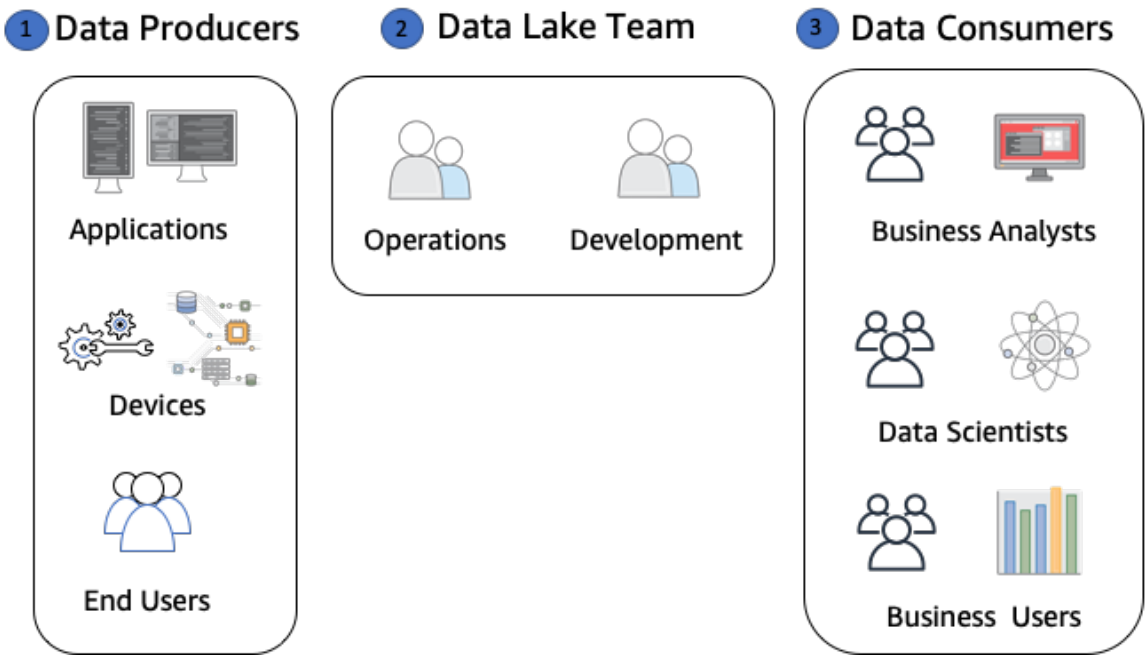


*Figure 1: Organizational Dynamics of a Data Lake*

- **Data Producers** are the organizational revenue generators. These entities, be it logical (software) or physical (Application users) are not mandated to conform to any contract (schema, frequency, structure, etc.) associated with the data they produce—except for where to store the data they produce in the data lake.
- **Data Lake Team** is often made up of an *operations team*, which defines the security and policy mechanisms that are mandated within the data lake, and the *development team* that supports ongoing schema and ETL management.
- **Data Consumers** retrieve data from the data lake using the mechanisms authorized by the Data Lake Team, and further iterate on that data to meet business needs.

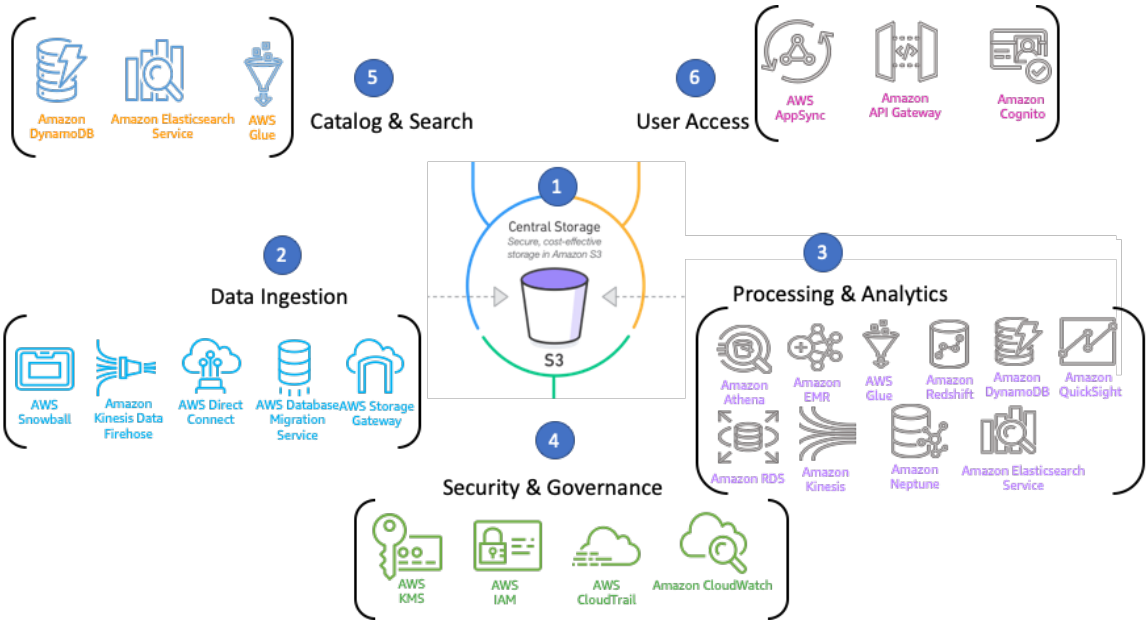


*Figure 2: High-Level Data Lake Technical Reference Architecture*

1. **Amazon S3** is at the core of a data lake on AWS. Amazon S3 supports the object storage of all the raw and iterative datasets that are created and used by ETL processing and analytics environments. The high-level structure housing the data within Amazon S3 is organized in two or three tiers depending on the business requirements of the data lake
   - **Tier 1 Storage (Raw Data) –** this storage tier is one or more S3 buckets that host the raw data coming from the ingestion services. It's important that the data stored in this tier is maintained and preserved in its original form and that no data transformation occurs.
   - **Tier 2 Storage Analytics Optimized –** this storage tier hosts the iterative datasets resulting from the transformation of the raw data in Tier 1 into common columnar formats (such as Parquet, ORC, or Avro) using ETL Job processing (for example, Spark on EMR, or AWS Glue). Organizing data into partitions and storing it in a columnar format allows compute environments that ingest Tier 2 data to achieve better performance at lower costs.
   - **Tier 3 Use-Case-Specific Data Mart (optional) –** the data hosted in this tier is a subset from Tier 2 that has been organized for specific use case data marts. Tier 3 data often has high access and security constraints. Depending on the use case, the data is served from more applicable compute and analytics environments, such as Amazon EMR, Amazon Redshift, Amazon Neptune, and Amazon Aurora.
2. **The Data Ingestion** component of the data lake architecture represents a process to persist data in Amazon S3. Data originating from this section of the data lake is intended to be Tier 1 raw data. The data producing services do not have any constraints, structure conformity requirements, or contracts on the data, except for a predetermined S3 bucket where data is stored.
3. **Processing & Analytics** consists of AWS services that are used to process or ETL data from Tier 1 and deliver the processed data to Tier 2. The data in Tier 2 can then be consumed by an analytics or machine learning environment which does interactive querying or machine learning and model building.
4. **Protect & Secure** is an overarching abstraction that integrates into multiple services that comprise the data lake. This data lake section is intended to communicate the importance of taking into account Authentication, Authorization, Auditing, Compliance and Encryption at Rest/Transit as they apply to the ingestion, storage, and processing/analytics portions of the data lake. Most of these security concepts are integrated into each data processing service as a feature set to protect the access and integrity of the data the services are managing. For more information on how different security

focused services, such as KMS, IAM, and CloudTrail, are integrated into ingestion and analytics services like Kinesis and Amazon Redshift, see the Security Pillar section.

5. **The Catalog & Search** component ensures that the metadata about the datasets hosted in the storage portion (Amazon S3) of the data lake are captured, governed, maintained, indexed, and searchable. This often requires an organizational mandate on the minimal set of metadata (that is, description, tags, usage instructions, etc.) required for each dataset managed in the data lake catalog. This also ensures that all the individual objects that make up a data lake dataset are mapped to the overarching dataset metadata record. You can leverage such AWS services as DynamoDB, Amazon Elasticsearch Service, and the AWS Glue Data Catalog to track and index the metadata of the datasets hosted within your data lake. Using AWS Lambda functions that are triggered by Amazon S3 during new or updated object events, you can easily keep your catalog up to date.
6. **Access & User Interface**

   On AWS, services including **Amazon API Gateway**, **AWS Lambda**, and **AWS Directory Service** enable you to track and securely execute:

   - The creation, modification, and deletion of new data lake datasets and their corresponding metadata.
   - The creation and management of ETL jobs that create, update, and coalesce iterative datasets that would transform raw Tier 1 Data into columnar, compressed and performant Tier 2 Data.
   - Create, delete, or modify analytics compute environments that ingest Tier 2 data for follow-on querying, visualization development, and insights analysis.

## Configuration Notes

- Decide on a location for data lake ingestion (that is, S3 bucket). Select a frequency and isolation mechanism that meets your business needs.
- For Tier 2 Data, partition the data with keys that align to common query filters. This enables pruning by common analytics tools that work on raw data files and increases performance.
- Choose optimal file sizes to reduce Amazon S3 round trips during compute environment ingestion:
  - Recommended: 512 MB – 1 GB in a columnar format (ORC/Parquet) per partition.
- Perform frequent scheduled compactions that align to the optimal file sizes noted previously.
  - For example, compact into daily partitions if hourly files are too small.
- For data with frequent updates or deletes (that is, mutable data):
  - Temporarily store replicated data to a database like Amazon Redshift, Apache Hive, or Amazon RDS until the data becomes static, and then offload it to Amazon S3, or
  - Append the data to delta files per partition and compact it on a scheduled basis using AWS Glue or Apache Spark on EMR.
- With Tier 2 and Tier 3 Data being stored in Amazon S3:
  - Partition data using a high cardinality key. This is honored by Presto, Apache Hive, and Apache Spark and improves the query filter performance on that key.
  - Sort data in each partition with a secondary key that aligns to common filter queries. This allows query engines to skip files and get to requested data faster.

# Batch Data Processing

Most analytics applications require frequent batch processing, for example, to update data stores with pre-aggregated results to make reporting queries faster and simpler for end users. Batch systems must be built to scale to all sizes of data and grow proportionally to the size of the data set being processed.

The rapid expansion of data sources and sizes demands a batch data processing system that is flexible without compromise. Business requirements might dictate that batch data processing jobs be bound by an SLA, or have certain budget thresholds. These requirements should determine the characteristics of the batch processing architecture.

On AWS, services including **Amazon EMR, AWS Glue, and AWS Batch** enable you to run distributed compute frameworks across dynamically scalable EC2 instances or fully managed environments. In a batch processing context, Amazon EMR provides the capability to read and write data to Amazon S3, NoSQL on DynamoDB, SQL databases on Amazon RDS, Amazon Redshift, Hadoop Distributed File Systems (HDFS), and more. Popular frameworks such as Spark, MapReduce, and Flink can be used to distribute work over dynamically scaled clusters. AWS Batch can be used to run standalone or array jobs using Docker containers deployed on dynamically scalable container compute infrastructure. With AWS Glue, you can run Spark jobs without managing any EC2 instances.

By reading and writing data from external sources that are separate from the batch processing compute environments, you decouple storage from compute. In doing so, you can examine running Amazon EMR, AWS Batch, and AWS Glue resources only when jobs are scheduled. This presents a paradigm shift from the traditional model for batch processing systems. Now, you can run compute resources transiently only when they are actually processing the data. Amazon EMR and AWS Batch are also highly integrated with EC2 Spot Instances, allowing you to run EC2 instances and save on cost compared to On-Demand Instance EC2 Instance prices.

**Topics**



# Characteristics

- You want to create a batch data processing system with minimal management of clusters and compute resources.
- You want to reduce the time it takes for your business or end users to run insightful queries, and make it easier for them to understand the data.
- You want to run batch data processing compute resources only when they are needed, and shut them down when they are not.

# Reference Architecture

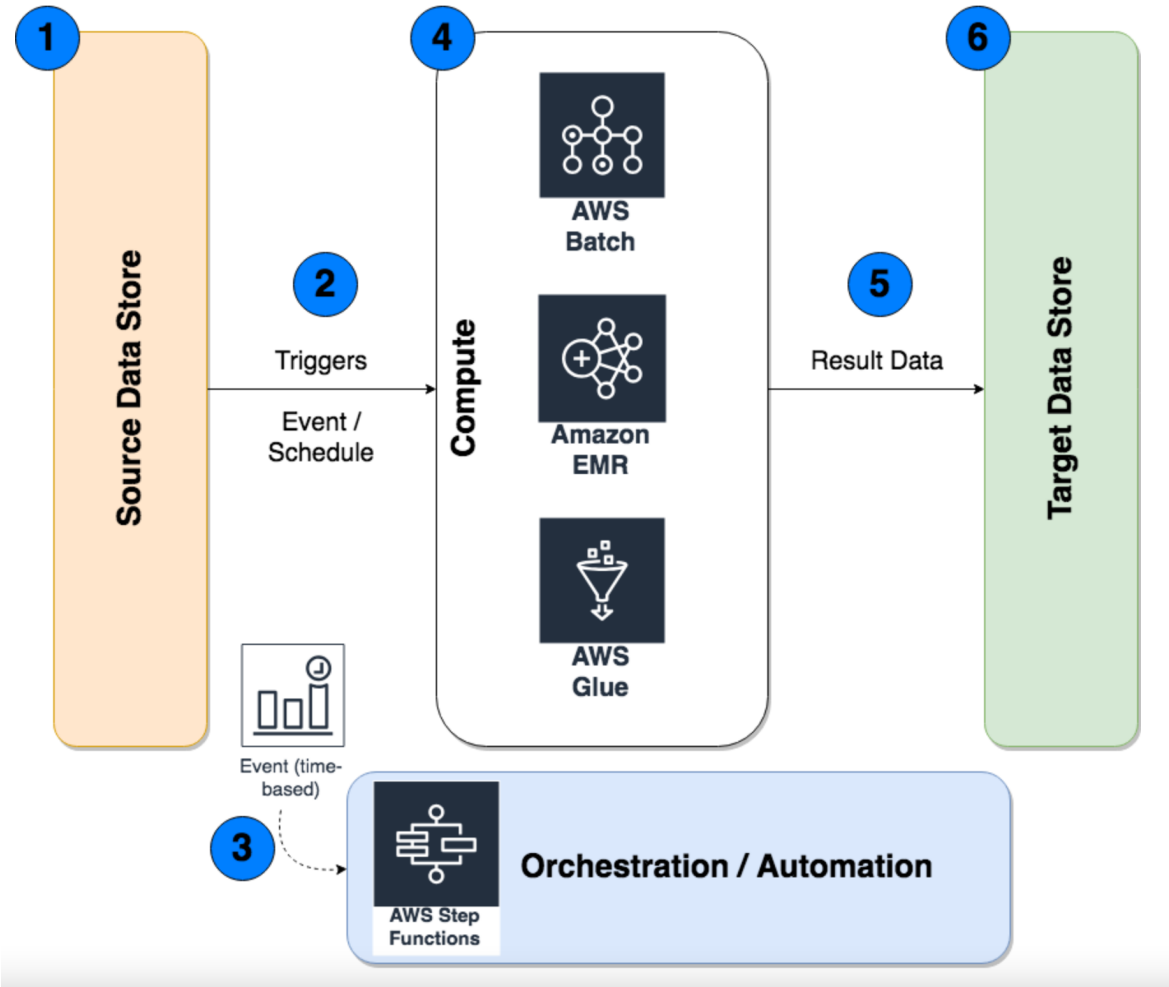


*Figure 3: High-Level Batch Data Processing Architecture*

1. Batch data processing systems typically require a persistent data store for source data. This is important when considering reliability of your system. Persisting your source datasets in durable storage enables you to retry processing jobs in case of failure and unlock new value streams in the future. On AWS, you have an extensive set of options for source data storage. **Amazon S3**, **Amazon RDS, Amazon DynamoDB, Amazon EFS, Amazon Elasticsearch Service, Amazon Redshift,** and **Amazon Neptune** are managed database and storage services you can use as source data stores. You also have the option of using Amazon EC2 and EBS to run your own database or storage solutions. See the ***Data Lake*** and ***Building an Efficient Storage Layer for Analytics*** scenarios for deeper dives into these options.
2. Batch data processing systems should be automated and scheduled for reliability, performance efficiency, and cost optimization. You can use **Amazon CloudWatch Events** to trigger downstream jobs based on schedules (for example, once a day) or events (for example, when new files are uploaded).
3. It's common for batch data processing jobs to have multiple steps—some of which might happen in sequence or in parallel. Using an orchestration service, such as **AWS Step Functions, makes it easy** to implement automated workflows for simple and complex processing jobs. With AWS Step Functions you can build distributed data processing applications using visual workflows. Within an AWS Step Functions workflow, you can use Lambda functions and native service integrations to trigger Amazon

EMR steps, AWS Glue ETL Jobs, AWS Batch jobs, Amazon SageMaker jobs, and custom jobs on Amazon EC2 or on premises.

4. **AWS Batch, AWS Glue,** and **Amazon EMR** provide managed services and frameworks for batch job execution that fit your specific use case. You have different options for running jobs. For simple jobs that can run in Docker containers—such as video media processing, machine learning training, and file compression—AWS Batch provides a convenient way to submit jobs as Docker containers to a container compute infrastructure on Amazon EC2. For Apache Spark jobs in PySpark or Scala, you can use AWS Glue, which runs Spark jobs in a fully managed Spark environment. For other massively parallel processing jobs, Amazon EMR provides frameworks like Spark, MapReduce, Hive, Presto, Flink, and Tez that run on Amazon EC2 instances in your VPC.
5. Similar to the source data stores, batch jobs require reliable storage to store job outputs or results. You can use the AWS SDK for interacting with Amazon S3 and DynamoDB, and use common file protocols and JDBC connections to store results in file systems or databases.
6. Result datasets are commonly persisted and later accessed by visualization tools, such as **Amazon QuickSight,** APIs, and search-based queries. Based on the access pattern, you might choose data stores that best fit your use-case. See the ***Data Lake*** and ***Building an Efficient Storage Layer for Analytics*** scenarios for deeper dives into these storage options.

# Configuration Notes

1. **Use Batch Processing jobs to prepare large, bulk datasets for downstream analytics.** For large, complex datasets, you may need to provide that data to end users and analysts in such a way that simplifies queries for them. In contrast, these users may find it difficult to query the raw data when they are trying to find simple aggregations. For example, you may want to preprocess a daily sales summarized view of your data for the previous day's sales. This provides users with a table that has fewer rows and columns, making it easier and faster for business users to query the data.
2. **Avoid lifting and shifting batch processing to AWS**. By lifting and shifting traditional batch processing systems into AWS, you risk running over-provisioned resources on Amazon EC2. For example, traditional Hadoop clusters are often over-provisioned and idle in an on-premises setting. Use AWS managed services, such as AWS Glue, Amazon EMR, and AWS Batch, to simplify your architecture and remove the undifferentiated heavy lifting of managing clustered and distributed environments.

   By effectively leveraging these services with modern batch processing architectures that separate storage and compute, you present opportunities to save on costs by eliminating idle compute resources or underutilized disk storage space. Performance can also be improved by using EC2 instance types that are optimized for your specific batch processing tasks, rather than using multi-purpose persistent clusters.
3. **Automate and orchestrate everywhere.** In a traditional batch data processing environment, it's a best practice to automate and schedule your jobs in the system. In AWS, you should leverage automation and orchestration for your batch data processing jobs in conjunction with the AWS APIs to spin up and tear down entire compute environments as well, so that you are only charged when the compute services are in use. For example, when a job is scheduled, a workflow service, such as AWS Step Functions, would use the AWS SDK to provision a new EMR cluster, submit the work, and terminate the cluster after the job is complete.
4. **Use Spot Instances to save on flexible batch processing jobs.** Leverage Spot Instances when you have flexible job schedules, can retry jobs, and can decouple the data from the compute. Use Spot Fleet, EC2 Fleet, and Spot Instance features in EMR and AWS Batch to manage Spot Instances.
5. **Continuously monitor and improve batch processing.** Batch processing systems evolve rapidly as data source volumes increase, new batch processing jobs are authored, and new batch processing frameworks are launched. Instrument your jobs with metrics, timeouts, and alarms to have the metrics and insight to make informed decisions on batch data processing system changes.

# Streaming Ingest and Stream Processing

Ingesting and processing real-time streaming data requires scalability, reliability, and low latency to support a variety of applications. These applications include activity tracking, transaction order processing, click-stream analysis, data cleansing, metrics generation, log filtering, indexing, social media analysis, IoT device data telemetry and metering. These applications are often spiky and process thousands of events per second.

With AWS, you can take advantage of the managed streaming data services offered by Amazon Kinesis, or deploy and manage your own streaming data solution in the cloud on Amazon EC2. See the definitions section for details on the streaming services that can be deployed on AWS.

Consider the following characteristics as you design your stream processing pipeline for real-time ingestion and continuous processing.

**Topics**



## Characteristics

- **Scalable:** For real-time analytics, you should plan a resilient infrastructure that can adapt to changes in the rate of data flowing through the stream. Scaling is typically performed by an administrative application that monitors shard and partition data-handling metrics. You want the workers to automatically discover newly added shards or partitions, and distribute them equitably across all available workers to process.
- **Durable:** Real-time streaming systems should provide high availability and data durability. For example, Amazon Kinesis Data Streams replicates data across three Availability Zones providing the high durability that streaming applications need.
- **Replayable reads:** Streaming processing systems should provide ordering of records, as well as the ability to read or replay records in the same order to multiple consumers reading from the stream.
- **Fault-tolerance, checkpoint, and replay**: *Checkpointing* refers to recording the farthest point in the stream that data records have been consumed and processed. If the application crashes, it can resume reading the stream from that point instead of having to start at the beginning.
- **Enable multiple processing applications in parallel:** The ability for multiple applications to consume the same stream concurrently is an essential characteristic of a stream processing system. For example, you have one application that updates a real-time dashboard and another that archives data to Amazon Redshift. You want both applications to consume data from the same stream concurrently and independently.
- **Messaging Semantics:** In a distributed messaging system, components might fail independently. Different messaging systems implement different semantic guarantees between a producer and a consumer in the case of such a failure. The most common message delivery guarantees implemented are:
  - **At most once:** Messages that could not be delivered, or are lost, are never redelivered.
  - **At least once:** Message might be delivered more than once to the consumer.
  - **Exactly once:** Message is delivered exactly once.

  Depending on your application needs, you need to choose a message delivery system that supports the required semantics.
- **Security:** Streaming ingest and processing systems need to be secure by default. You need to restrict access to the streaming APIs and infrastructure as well as encrypt data at rest and in transit in the

stream. With Kinesis Data Streams, only the account and data stream owners have access to the Kinesis resources they create. Kinesis supports user authentication to control access to the data. You can use AWS IAM policies to selectively grant permissions to users and groups of users. You can securely put and get your data from Kinesis through SSL endpoints using the HTTPS protocol. If you run Apache Kafka, to ensure the security of your Kafka cluster, you need to deploy HTTPS, maintain certificate authorities, and configure the Kafka instances to use SSL to encrypt data in transit.

# Reference Architecture

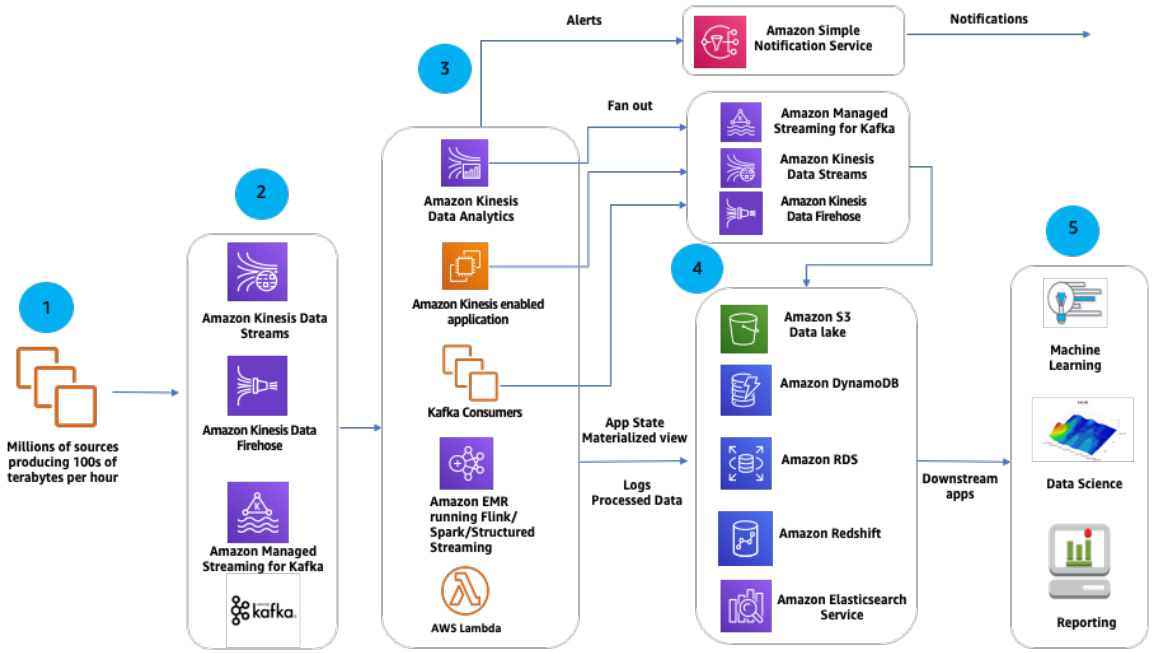


*Figure 4: Streaming Data Analytics Reference Architecture*

1. **Data Producers:** Multiple producers generate data continuously that might amount to terabytes of data per day. Producers can use Kinesis Agent, which is a standalone Java software application, to collect and send data to Amazon Kinesis Data Streams or Amazon Kinesis Data Firehose. The agent continuously monitors a set of files and sends new data to your delivery stream. The agent handles file rotation, checkpointing, and retry upon failures and delivers all of your data in a reliable, timely, and simple manner. If you use an operating system that is not compatible with the Kinesis agent, you could, alternatively, use Kinesis Producer Library (KPL) to achieve high write throughput to a Kinesis data stream. The KPL is an easy-to-use, highly configurable library that helps you write to a Kinesis data stream. It acts as an intermediary between your producer application code and the Kinesis Data Streams API actions. Producers can also use Kafka Producers to send messages to a Kafka cluster.
2. **Streaming Ingest:** Kinesis Data Streams and Kinesis Data Firehose can ingest and process large streams of data records. Choose Kinesis Data Streams for rapid and continuous data intake and aggregation as the response time for the data intake and processing is in real time. Use Kinesis Data Firehose, a fully managed streaming service, to transform and deliver real-time streaming data to destinations such as Amazon S3, Amazon Redshift, Amazon Elasticsearch Service (Amazon ES), and Splunk.

   If you use Apache Kafka, you can deploy the cluster on Amazon EC2 to provide a high performance, scalable solution for ingesting streaming data. AWS offers many different instance types and storage option combinations for Kafka deployments. Alternatively, you can use Amazon Managed Streaming for Apache Kafka (Amazon MSK) to build and run production applications on Apache Kafka without needing Apache Kafka infrastructure management expertise.

3. **Stream Processing:** Real-time data streams can be processed sequentially and incrementally on a record-by-record basis over sliding time windows using a variety of services.

   For example, with Kinesis Data Analytics, you can process and analyze streaming data using standard SQL in a serverless way. The service enables you to quickly author and run SQL code against streaming sources to perform time series analytics, feed real-time dashboards, and create real-time metrics. The SQL query results can then be configured to fan out to external destinations such as Kinesis Data Firehose or Kinesis Data Streams.

   For data ingested using Kinesis Data Streams, you can develop a consumer application using the Kinesis Client Library (KCL). Although you can use the Kinesis Data Streams API to get data from a Kinesis data stream, we recommend using the design patterns and code for consumer applications provided by the KCL. You can use Kinesis Data Streams as a fully managed event source for Lambda functions.

   Another popular way to process streaming data is with Kinesis Data Firehose. Kinesis Data Firehose can invoke your Lambda function to transform incoming source data and deliver the transformed data to destinations. You can enable Kinesis Data Firehose data transformation when you create your delivery stream.

   If you work in a Hadoop environment, you can process streaming data using multiple options—Spark Streaming, Apache Flink or Structured Streaming. Customers can use streaming ingest solutions, such as Kinesis Data Streams, Amazon MSK, or Apache Kafka running on Amazon EC2 with Apache Spark Streaming, for fault-tolerant stream processing of live-data streams, and Spark SQL, which allows Spark code to execute relational queries, to build a single architecture to process real-time and batch data.

   Apache Flink is a streaming dataflow engine that you can use to run real-time stream processing on high-throughput data sources. Flink supports event time semantics for out-of-order events, exactly-once semantics, backpressure control, and APIs optimized for writing both streaming and batch applications. Additionally, Flink has connectors for third-party data sources, such as Amazon Kinesis, Apache Kafka, Amazon ES, Twitter Streaming API, and Cassandra. Amazon EMR supports Flink as a YARN application so that you can manage resources along with other applications within a cluster. Flink-on-YARN allows you to submit transient Flink jobs, or you can create a long-running cluster that accepts multiple jobs and allocates resources according to the overall YARN reservation.

   With Apache Kafka as the streaming source, you can also use Spark Structured Streaming on Amazon EMR. Structured Streaming is a fault-tolerant stream processing engine built over Spark SQL.
4. **Alerts, messaging fan-out and storage:** Processed streaming data can be fed to a real-time predictive analytics system to derive inferences which in turn can be used for alerting using Amazon SNS. Messages that are processed can also be fanned out to Kinesis Data Streams, Kinesis Data Firehose, Kinesis Data Analytics, or AWS Lambda to generate new streams for further processing. Further down in the pipeline, applications can perform simple aggregation based on streaming data and emit processed data into Amazon S3. Other patterns include storing the real-time data in Amazon Redshift for complex analytics or DynamoDB for querying events or Amazon ES for full text search.
5. **Downstream analytics:** Data processed on the fly using streaming technologies can be persisted to serve real-time analytics, machine learning, alerts, and additional custom actions.

# Configuration Notes

1. **Aggregate records before sending to Kinesis Data Streams.** When using Kafka, ensure that the messages are accumulated on the producer side.
2. **When working with Kinesis Data Streams, use KCL to de-aggregate records.** KCL takes care of many of the complex tasks associated with distributed computing—such as load balancing across multiple instances, responding to instance failures, checkpointing processed records, and reacting to re-sharding.

3. **Leverage AWS Spot Instances to process streaming data cost effectively.** You can also process the data using AWS Lambda with Kinesis, and Kinesis Record Aggregation & Deaggregation Modules for AWS Lambda.
4. **Monitor Kinesis data stream metrics using Amazon CloudWatch.** With Kinesis data streams you can get basic stream level metrics as well as shard level metrics.

# Lambda Architecture

The analytics Lambda Architecture pattern seeks to unite big data streaming and batch processing systems into aggregated views. Traditional approaches to batch and stream processing sacrifice performance to use common tools. Other traditional approaches consisting of entangled systems are used without giving up performance, but are difficult to manage and require software domain-specific expertise.

AWS services, including Amazon EMR, Amazon Kinesis Data Streams, Amazon Kinesis Data Firehose, Amazon Kinesis Data Analytics, AWS Glue, and Amazon Athena, make it easy to bring stream and batch processing results together in a single serving layer for your end users, without having to manage complex infrastructure or cluster environments.

**Topics**



## Characteristics

1. You have both batch data processing and streaming data that is interrelated, and integrating the two would provide faster insights for your business.
2. You have already combined batch and streaming data, but the process is slow, complex, or difficult to manage (managing multiple open source stacks, like Apache Kafka, Apache Hadoop, etc., running on Amazon EC2 instances.)

**Example:** Streaming data from IoT sensor devices, and batch processing data from a device settings database and combining these two.

# Reference Architecture

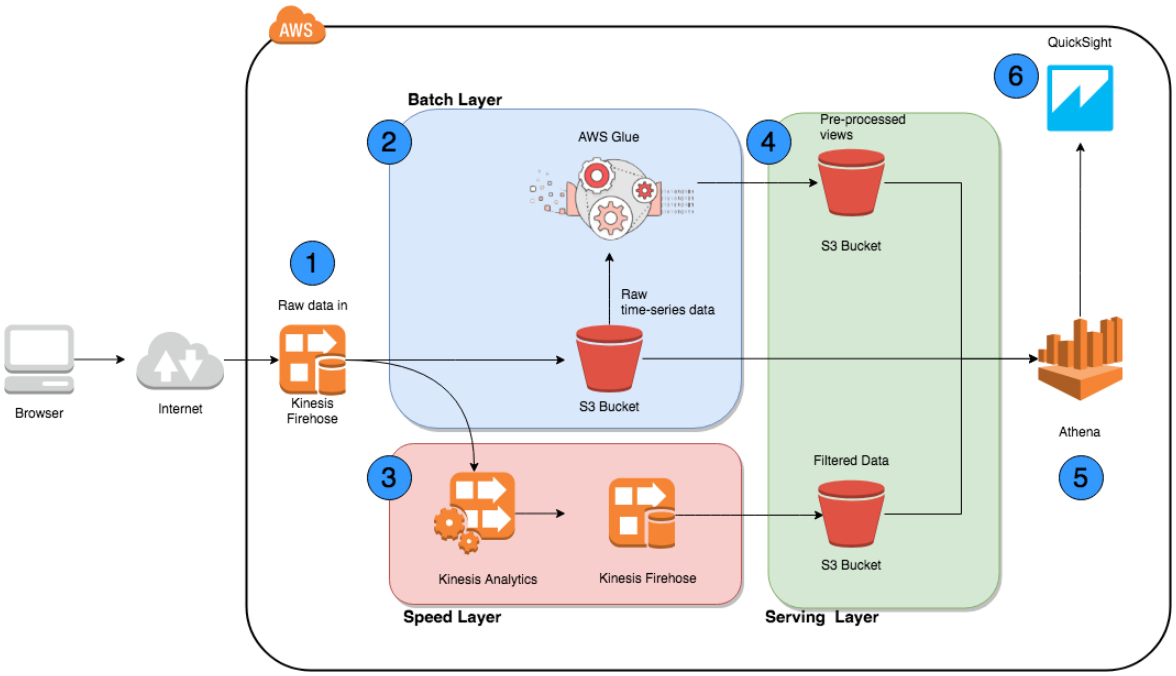


*Figure 5: Lambda Architecture*

Note that this architecture does not use any Amazon EC2 instances for running clusters or distributed streaming or processing frameworks. In the batch layer, AWS Glue is used for serverless ETL and batch processing using Spark Jobs. Glue can be used to extract data from the Live Database, and for doing the Batch Layer processing for combining the Live Database data with the streaming data for complex pre-processed views. In the Speed Layer, streaming data is read in real time from Amazon Kinesis Data Firehose by Amazon Kinesis Data Analytics, where you can join the streaming data from a reference dataset in Amazon S3 (extracted from the Live Database). Amazon Kinesis Data Analytics is used here to join the data in real time, filter data or run machine learning anomaly detection algorithms, and persist the results in Amazon S3. Amazon S3 is used as the Serving Layer, and Amazon Athena and Amazon QuickSight are used to query the various processed data.

1. **Streaming data Producers**: Data generated continuously may generate terabytes per day, and is collected and sent to Kinesis Data Streams or Kinesis Data Firehose.
2. **Batch layer:** Using AWS Glue, you analyze the raw data from Amazon S3 in batch-oriented fashion to look at the datasets over time against the historical data, and store results back in Amazon S3.
3. **Speed layer**: Using Amazon Kinesis Data Analytics, you analyze and filter the data to detect abnormalities in real time.
4. **Serving layer:** Raw data, preprocessed views, and real-time filtered data is available in Amazon S3 for direct querying in the Serving layer.
5. **Amazon Athena provides serverless SQL queries on top of Amazon S3 to power visualizations, dashboards, and ad hoc queries.**
6. Finally, use Amazon Athena and Amazon QuickSight together to query and visualize the data and build a dashboard that can be shared with other users of Amazon QuickSight in your organization.

## Configuration notes:

1. **Leverage serverless compute and analytics services where possible.** This removes the undifferentiated heavy lifting of managing distributed and clustered environments. In AWS, services like Kinesis Data Firehose, Amazon Kinesis Data Analytics, AWS Glue, Amazon S3, Athena, and QuickSight are fully managed and do not require patching, installation of software, back-ups, etc.
2. **Leverage Amazon EMR for frameworks outside of the serverless scope.** You may find your Speed Layer calculations are better suited in something like Apache Flink or Spark Streaming. You can use Amazon EMR for managing cluster environments for those applications.
3. **Determine the right data to combine at the right time.** Not all use-cases demand the use of Lambda Architectures. Work backward from your business requirements for the speed and data freshness that is required for the views in the serving layer.
4. **Create pre-processed views for your end users.** For some users, presenting access to raw streaming and master data can be complicated and overwhelming at best, and inefficient at worst (for example, SQL queries that select all columns without filtering). Creating pre-processed views (also known as materialized views in other domains) simplifies queries and also improves performance. You can align business expectations for freshness of the views.

For more information and a hands-on tutorial on Lambda Architectures, see the following blog post: Unite Real-Time and Batch Analytics Using the Big Data Lambda Architecture, Without Servers!

# Data Science

In a typical simplified data science workflow, data scientists perform an iterative process of model development by preparing data for machine learning, training their algorithms on a training dataset, evaluating algorithm performance against a separate validation dataset, refining their data preparation and algorithms for re-training, then packaging their algorithms into a production deployment framework.

AWS provides several levels of abstraction for services used in machine learning. At the most abstract level, AWS AI services such as Amazon Polly, Amazon Lex, Amazon Comprehend, and Amazon Rekognition provide pre-trained API endpoints that can serve inferences against input data. In the middle tier of abstracted services, platform services such as Amazon Machine Learning, Amazon SageMaker, EMR with SparkML, and Mechanical Turk (for data labeling) are used to accelerate algorithm development without the heavy lifting required to self-manage underlying resources. Finally, the least abstracted tier of frameworks and infrastructure include the ability to spin up CPU-optimized and GPU-optimized AWS Deep Learning AMIs with pre-configured popular frameworks such as TensorFlow, PyTorch, and Apache MXNet. The selection of which machine learning stack to use will depend on the business problem to be solved as well as development team knowledge and experience.

Data science algorithm development is beyond the scope of this document, however common scenarios for preparing, capturing, and utilizing big data for machine learning purposes is presented.

In most business contexts, data is stored in a central repository or distributed across several repositories. This repository could be Amazon S3, a Hadoop framework such as EMR, a relational database such as RDS, a NoSQL database such as DynamoDB, a data warehouse, or a combination such as a Hive-compatible meta-store. However, the raw data collected during the course of business is rarely sufficient clean for machine learning applications. Most machine learning algorithms have strict requirements that data meet certain conditions, such as avoiding null values. Preparing data for machine learning is often the most time consuming portion of a data scientist's job. It is a best practice to develop robust and automated data pipelines to aggregate and transform the raw data a business collects into machine learning capable datasets. Further augmentation of the datasets is often advantageous to improve model performance, for example feature engineering by normalizing or combining columns of data, tokenizing a corpus of text, or pre-processing video into individual frames.

On AWS, managed ETL services such as Glue jobs, EMR, or Data Pipeline reduce the heavy lifting of managing servers required to transform data as well as provide multi-step orchestration of data transformation. By treating the AWS infrastructure as code and utilizing the AWS API endpoints for these services, data scientists can version control the development of their cleansing pipelines as well as document best practices for other data scientists.

Staging, cleansing and feature engineering into sequential pipelines has advantages for hastening model development. More computation and memory are often required to cleanse and sample data from an extremely large raw dataset into a smaller clean dataset than is required to perform feature engineering on the smaller dataset itself. Follow the Well-Architected pillar of performance efficiency to utilize the appropriate instance types for the different stages of the pipeline. At the endpoint of the ETL cleansing pipeline it is a best practice for data scientists to store their cleansed data in a separate data store, such as Amazon S3. A separate data store for machine learning model development ensures data is consistent between model training runs, which provides a stable baseline for evaluating model improvement. Separating compute and storage is a best practice of the cost-optimization Well-Architected pillar.

To solve business problems, the machine learning model will be deployed to serve inferences against new data. Another best practice is to track and improve model performance by augmenting the raw data store with additional labeled data specific to the machine learning context, known as data augmentation. This can be accomplished by recording the new data, the inference, and the “ground truth” of the actual data point. For example, whether or not the inferred “best” advertisement for a particular patron converted to a sale. The ground truth data is invaluable for measuring and improving model performance through retraining with the augmented data. In some cases, ground truth may not be known for some time, so implementing a tracking methodology that links inference to outcome should be considered.

**Topics**



# Characteristics

- You want to use machine learning to solve a business problem
- You have large amounts of data with relationships that are not easy to visualize using traditional warehouse tools
- You want to identify relationships and trends in sparse datasets, for example recommending products
- You want to make predictions based on historical data, for example predicting user churn
- You want to improve a business function in real time, for example serving the most likely advertisement to convert a sale
- You want to improve a business function in real time, for example predicting a part failure from sensor data

# Reference Architecture

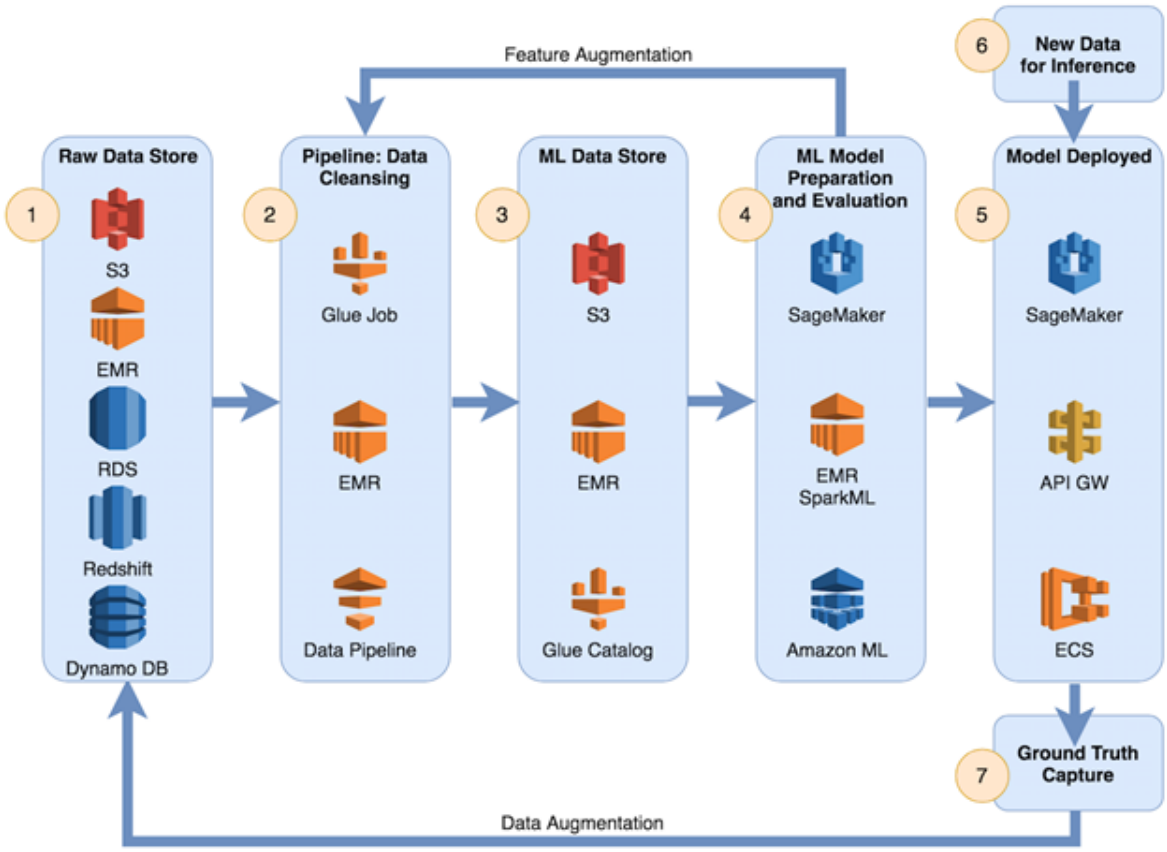


*Figure 6: Data Science Pipeline Reference Architecture*

1. Raw data store. Most frequently, the raw data store is a data lake where the data is ingested and housed.
2. A data-cleansing pipeline converts raw data into machine-learning-ready datasets.
3. Data used for training stored in another repository, for example written to Amazon S3, or an HDFS store, or metastore.
4. ML model is developed using any number of frameworks, for example Amazon SageMaker, SparkML on EMR, GPU-instances running Deep Learning AMIs, R, Python, or Scala.
5. ML model is packaged into a production endpoint. This often sits behind an API to make inferences.
6. In production, new data is served to the ML deployment. Inferences are used to solve business needs.
7. Data capture during production deployment is saved and stored with the appropriate labels. This data is used periodically to re-train the model.

# Configuration notes

- Machine Learning data often needs to be cleansed of missing fields, labeled, and features “engineered” to reduce or combine extraneous columns/features. Additionally, ML training often needs only a subset of the data available, so a down-sampling method is often chosen to speed model training and thereby shorten cycles of learning/refinement.

- During model preparation, data consistency is important so model evaluation can be compared against previous versions. A training dataset and a “hold-out” or validation dataset may be stored separately for model training and evaluation, respectively.
- For cost optimization of compute resources, select the right instance type for training and making inferences. For example, a GPU-optimized instance may speed training neural-network models, but a general-purpose instance is sufficient for serving API inferences from the NN model.
- Utilizing fan-out resources such as Amazon SageMaker or SPOT market resources, multiple versions of the same model can be trained simultaneously with different tuning parameters, known as hyper-parameter optimization, vastly accelerating the development cycle.
- Data Scientists often have their preferred method of developing models. Notebook-style IDEs such as Jupyter Notebook, Zeppelin, or R can present the IDE in the user's local browser, while commands are executed on remote resources within the AWS Cloud.
- Try to avoid moving datasets frequently between the cloud and the data scientist's local workstation. A preferred pattern is to leave all data at rest in Amazon S3, load data as needed for development, retaining ETL pipelines.
- Utilizing API Gateway, Amazon SageMaker, or ECS for deploying your models allows routing incoming traffic to different versions of the ML model, thus verifying performance in production before routing all traffic to the new version.

# Multi-tenant Analytics

Multi-tenant architectures allow organizations to share resources among multiple users/parties while providing controls to keep the tenants isolated and secure. In analytics, multi-tenant designs facilitate many use cases such as running workloads to support multiple departments on a data lake, testing multiple versions of an application and/or testing single application with different datasets allowing full utilization of the expensive cluster resources. In addition, multi-tenant design can be economical as the resources and associated maintenance costs will be shared by multiple tenants.

**Topics**



## Characteristics:

**Resource Isolation:** In a multitenant environment, there should be a logical isolation of the resources consumed by each tenant. This ensures that a tenant’s analytic activities do not interfere or impede another tenant’s tasks.

**Data security:** A tenant’s data assets can be housed on a variety of data stores. Segregate tenants’ data from each other by employing one of the two mechanisms – 1) separate schemas and/or databases for each of the tenants 2.) designing databases to be shared by multiple tenants with data partitioned via a column that represents a unique tenant identifier. Authentication, authorization and auditing methods should be implemented to ensure data security.

**Metered Usage:** Tenants should only pay for the storage and processing they consume in a multi-tenant analytics environment. This often requires careful monitoring and measuring of various components and subsystems used by each tenant.

**Scalability:** As new tenants are onboarded, the multi-tenant analytics environment should easily scale to meet higher processing and storage requirements.

# Reference Architecture

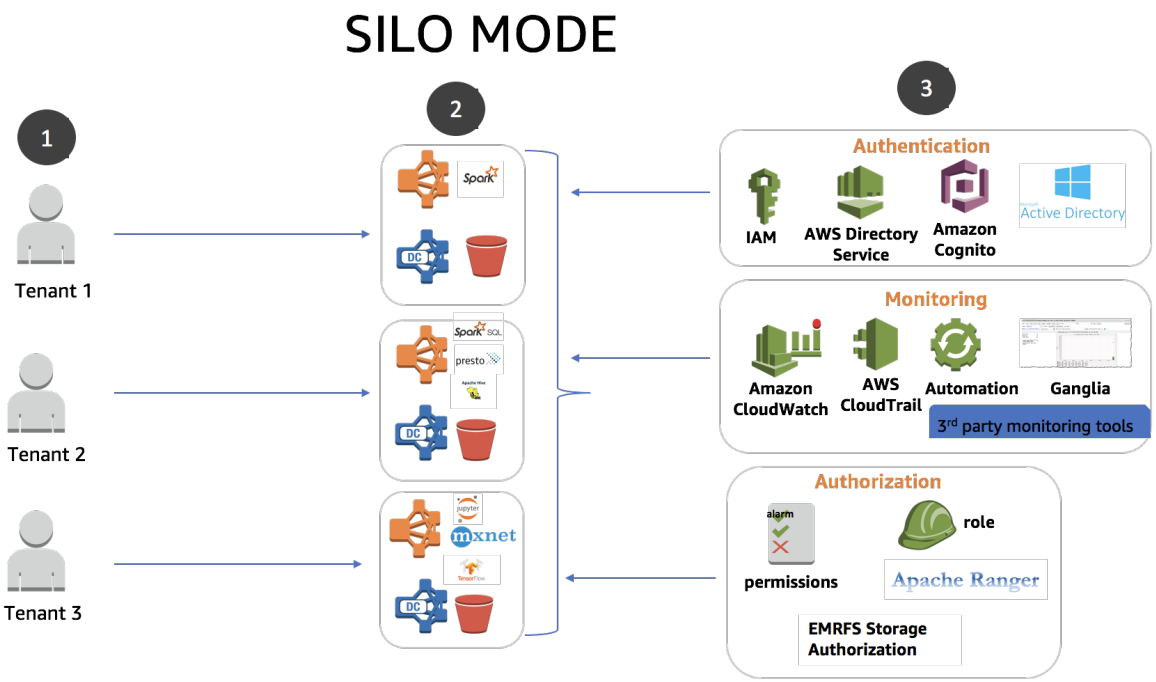


*Figure 7: Reference architecture for multi-tenant analytics on AWS (silo mode)*

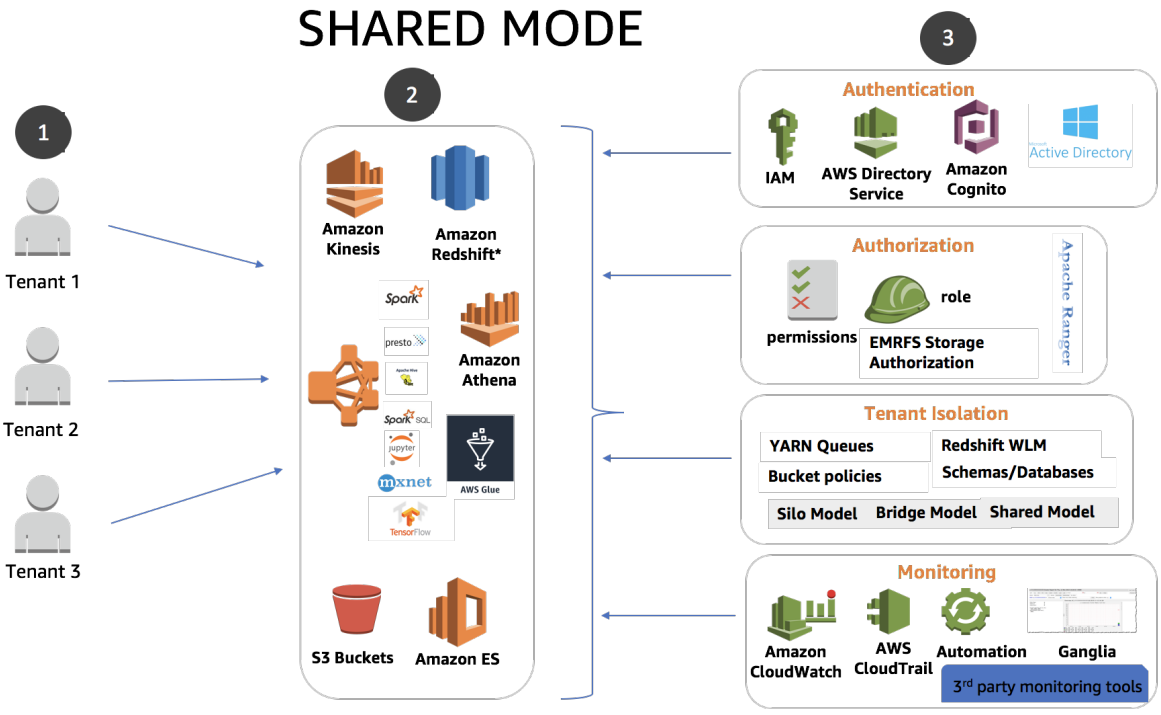


*Figure 8: Reference architecture for multi-tenant analytics on AWS (shared mode)*

There are two basic models that are commonly used when partitioning tenant usage and data in multi-tenant analytics architecture. Each partitioning model takes a very different approach to managing, accessing, and separating tenant data.

**Silo/dedicated model**: Each tenant gets their own dedicated resource and the data is fully isolated from any other tenant data. All constructs that are used to represent the tenant's data are considered logically "unique" to that client, meaning that each tenant will generally have a distinct representation, monitoring, management, and security footprint.

1. Tenants - A data engineer, an analyst and a data scientist who all need resources to run their activities
2. Each tenant launches their own clusters. A data engineer installs tools like Spark and Hive to manipulate and store the processed results in Amazon S3. An analyst might run tools such as Spark SQL and Presto to explore datasets and send the Query results to his own S3 bucket. And a data scientist might be using the EMR cluster to run ML or Deep Learning frameworks
3. Different authentication and isolation patterns based on the analytics tool of choice

**Shared/pool model:** This is a true multi-tenant model where the tenants share all of the analytic system's infrastructure constructs. Tenant data is placed into a common database and all tenants share a common representation (schema). This requires the introduction of a partitioning key that is used to scope and control access to tenant data. Analytics resources comprised of clusters are shared among the tenants using constructs like queues and work load management modules. This model tends to simplify a SaaS solution's provisioning, management, and update experience. It also fits well with the continuous delivery and agility goals that are essential to SaaS providers.

1. Tenants - A data engineer, an analyst and a data scientist who all need resources to run their activities
2. A large multi-node cluster with all the tools and frameworks installed can support a variety of users. In addition, this infrastructure can also be used by end users who can launch edge nodes to run their data science platforms. Despite the cost effectiveness, sharing a cluster can be a cause for concern as a tenant might monopolize resources and cause the SLA to be missed for other tenants. For example, an analyst can issue long running query on Presto or Hive and much of the cluster resources. Or a data scientist might train a model over massive amounts of data
3. Different authentication and isolation patterns based on the analytics tool of choice

# Configuration Notes:

Architecting secure multi-tenant environment in analytics requires careful consideration on several fronts which includes

- **Authentication of tenants**: A multi-tenant architecture requires managing identities belonging to different tenants. Users should be able to authenticate themselves to the analytics application using their AWS IAM or managed credentials.
- **Authorization to datasets:** Once authenticated, users should only be able to access their own datasets. In multi-tenant analytics this means providing users with just enough privileges to access their data but not those of other tenants.
- **Isolation of resources for tenants:** In multi-tenant applications, each tenant is isolated from others through a well-defined partitioning strategy. Degree of isolation among tenants is determined by the chosen partitioning style - silo or shared.
- **Monitoring and metrics, centralized management of tenants and billing:** For multi-tenant analytic designs, it is imperative to have a management layer to monitor the operations, metrics, performance and billing construct to efficiently manage tenants.

# The Pillars of the Well-Architected Framework

This section describes each of the pillars, and includes definitions, best practices, questions, considerations, and key AWS services that are relevant when architecting solutions for analytics applications.

For brevity, we have only included questions that are specific to analytics workloads. Questions that have not been included in this document should still be considered when designing your architecture. We recommend that you read the AWS Well-Architected Framework whitepaper.

**Topics**



## Operational Excellence Pillar

The **operational excellence** pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures.

**Topics**



### Design Principles

In the cloud, a number of principles drive operational excellence. The design principles from the AWS Well-Architected Framework whitepaper are recommended and do not vary for analytics workloads.

### Definition

There are three best practice areas for operational excellence in the cloud:

- Prepare
- Operate
- Evolve

In addition to what is covered by the Well-Architected Framework concerning processes, runbooks, and game days, there are specific areas you should look into to drive operational excellence within analytics applications.

# Best Practices

**Topics**



## Prepare

To drive operational excellence in analytics workloads, you must understand your pipelines and their expected behaviors. You will then be able to design them to provide insight into their status and build procedures to support them.

To prepare for operational excellence, you need to consider the following:

- **Operational priorities:** Analytics teams typically have roles such as Data Engineer, Data Analyst, and Data Scientist. These teams need to have a shared understanding of the analytics pipelines, their role in it, and the shared business goals to set the priorities that will enable business success. You also need to consider external regulatory and compliance requirements for data assets and processing engines, which might influence your priorities and the choice of tools used in the pipeline. Use your priorities to focus your operations improvement efforts where they will have the greatest impact (for example, developing team skills, improving analytics application performance measuring techniques, automating and orchestrating batch and real-time pipelines, or enhancing monitoring). Update your priorities as needs change.
- **Design for operations:** The design of your analytics pipeline should include how it is deployed, updated, and operated. Having a good CI/CD pipeline can help your organization discover bugs before they reach production and deliver updates more frequently. It can also help developers write quality code and automate the ETL job release management process, mitigate risk, and run operations in a more predictable fashion. In AWS, you can view your entire workload (that is, applications, infrastructure, policy, governance, and operations) as code. It can all be defined in and updated using code. This means you can apply the same engineering discipline that you use for application code to every element of your stack.
- **Operational readiness:** A successful implementation of an analytics pipeline involves many essential steps that take into account risks and benefits. Thorough planning that includes preserving raw data assets, creating runbooks that document pipeline activities and playbooks that guide your processes for issue resolution, monitoring and debugging capabilities are critical to being operationally ready. You should also test your procedures, failure scenarios, and the success of your responses (for example, by holding game days and testing prior to going live) to identify areas that you need to address.

| **ANALYTICS_OPS 01: How do you select the most suitable analytics services and solutions?** |
| --- |
| |

It's important to classify your workload and choose the services that are best suited for your analytics applications. On AWS, there are many services available to simplify your analytics workloads in the cloud, whether your analytics use case falls under business intelligence, machine learning, NoSQL, and more.

Select analytics services for your use cases based on operational criteria, such as speed/performance, batch/real time, interactive/automated, and self-managed/serverless/AWS-managed. For example, for Apache Spark jobs focused on core ETL activities, AWS Glue simplifies your architecture by removing the burden of managing Apache Spark clusters. For highly customized workflows such as real time streaming using Kinesis Data Streams or for leveraging Amazon EC2 Spot Instances, Amazon EMR may be a more

suitable service. Amazon Redshift and Redshift Spectrum are complementary and are very suitable for business intelligence and reporting queries that expand into your data lake in Amazon S3.

**ANALYTICS_OPS 02: How do you create and deploy your analytics pipeline?**

Implementing an analytics pipeline comprises services for acquiring datasets, high-performance clusters for data manipulation, as well as services for the querying and visualization of data. As part of your preparation, you should automate the creation, configuration, and deployment of the services for seamless pipeline operations. You can pre-provision tooling and a clean environment using AWS CloudFormation. This allows you to not only provision the infrastructure using templates but also to carry out forensics in a safe, isolated environment. Organizations that transform their ETL applications to cloud-based, serverless ETL architectures benefit from a seamless, end-to-end continuous integration and continuous delivery (CI/CD) pipeline—from source code, to build, to deployment, and to product delivery. Having a CI/CD pipeline can help your organization discover bugs before they reach production and can deliver updates more frequently. It can also help developers write quality code, automate the ETL job release management process, mitigate risk, and more.

## Operate

Operational success is the achievement of outcomes measured by the metrics you define. By understanding the operational health of your analytics application, you can identify when it's impacted by operational events and respond appropriately.

To operate successfully, you need to consider the following:

- Understand operational health of the entire analytics pipeline
- Responding to Events

**ANALYTICS_OPS 03: How do you monitor the health of the analytics pipeline?**

Your team must be able to easily understand the operational health of your analytics workload. Use metrics based on key-performance indicators (KPIs) and operational outcomes to gain useful insights. Use these metrics to implement dashboards with business and technical viewpoints that can help team members make informed decisions. On AWS, capabilities are available to help you bring together and analyze your workload and operations logs so that you can know your operating status and gain insight from operations over time.

Many AWS managed services, such as Amazon RDS and Amazon Redshift, provide service-specific metrics that can be integrated into CloudWatch Dashboards. For example, by monitoring the average number of read and write operations per second on a data warehouse, you can determine usage patterns and scale your capacity accordingly. Metrics, including queue length/depth, are important and provide information about the number of queries or requests waiting to be processed by the data warehouse at any given time.

Send log data from your clusters to Amazon CloudWatch Logs and define baseline metrics to establish normal operating patterns. Create Amazon CloudWatch dashboards that present system- and business-level views of your metrics. For processing frameworks such as Amazon EMR, you should also install the Ganglia monitoring system, which is a scalable distributed monitoring system for high performance computing systems, such as clusters and grids.

You can also ingest your CloudWatch Logs log data into Amazon Elasticsearch Service (Amazon ES) and then use the built-in support for Kibana to create dashboards and visualizations of your operational health (for example, order rates, connected users, and transaction times). In the AWS shared

responsibility model, portions of monitoring are delivered to you through the AWS Service Health Dashboard (SHD) and the Personal Health Dashboard (PHD). These dashboards provide alerts and remediation guidance when AWS is experiencing events that might affect you. Customers with Business and Enterprise Support subscriptions also get access to the Amazon PHD API, enabling integration to their event management systems. AWS also has support for third-party log analysis systems and business intelligence tools through the AWS service APIs and SDKs (for example, Grafana, Kibana, and Logstash). Right-sizing capacity in your analytics pipeline should not be based on guess work—instead use operational metrics to make informed decisions.

**ANALYTICS_OPS 04: How do you manage operational events for your analytics application?**

An analytics pipeline consists of many moving parts across ingestion, storage, analysis, and visualization of your data. Failure in any of these individual layers of your data and analytics pipeline can have downstream impact on business critical applications that depend on your analytics layer. Ensuring that you have remediation steps in place for such events helps ensure business continuity and meeting your availability SLAs. Anticipate operational events, both planned (for example, sales promotions, deployments, and failure tests) and unplanned (for example, surges in utilization and component failures).

Create and use runbooks and playbooks to avoid confusion and methodically respond to alerts consistently. Defined alerts must be owned by a role or a team that is accountable for the response and escalations. Understand the business impact of your system components and use this knowledge to prioritize efforts where needed. Perform a root cause analysis (RCA) after events, and then prevent the recurrence of failures and document workarounds. Know when a human decision is needed before an action is taken and, when possible, have that decision made before the action is required. You should have critical manual procedures available for use when automated procedures fail.

AWS simplifies your event response by providing tools supporting all aspects of your workload and operations programmatically. These tools allow you to script responses to operations events and trigger their execution in response to monitored data. In AWS, you can improve recovery time by replacing failed components with known good versions, rather than trying to repair them. There are multiple ways to automate the execution of runbook and playbook actions on AWS.

To respond to an event from a state change in your AWS resources, or your own custom events, you should create CloudWatch rules to trigger responses through Amazon CloudWatch targets (for example, Lambda functions, Amazon Simple Notification Service (Amazon SNS) topics, Amazon Elastic Container Service (Amazon ECS) tasks, Step Functions, or AWS Systems Manager Automation). AWS also supports third-party systems through the AWS service APIs and SDKs. There are a number of tools provided by partners and third parties that allow for monitoring, notifications, and responses. Some of these tools include New Relic, Splunk, Loggly, SumoLogic, and Datadog.

**ANALYTICS_OPS 05: How are you evolving your data and analytics workload while minimizing the impact of change?**

As new technologies are introduced, it is common for organizations to upgrade their data and analytics stack to newer versions or replace a service for ingestion, processing, or visualization with a managed or serverless alternative. Decoupling storage from the compute layer for data assets, using an external metadata store, and using versioned configuration artifacts in Amazon S3 buckets enable you to upgrade and re-launch clusters and resume processing.

If you need more fine-grained control over configuration within analytics shared across multiple applications functions, consider the AWS Systems Manager Parameter Store feature over environment variables. Use AWS Secrets Manager to store database credentials and other “secrets” so you can easily rotate, manage, and retrieve the secrets and credentials from your analytics application code.

### Evolve

See the Well-Architected Framework whitepaper for best practices in the **evolve** area for operational excellence that apply to analytics applications.

## Resources

Refer to the following resources to learn more about our best practices for operational excellence.

### Documentation & Blogs

- Orchestrate multiple ETL jobs using AWS Step Functions and AWS Lambda
- Implementing continuous integration and delivery of serverless AWS Glue ETL Applications using AWS Developer Tools
- Implementing Dynamic ETL Pipelines Using AWS Step Functions
- The Right Way to Store Secrets using Parameter Store
- Tutorial - Storing and Retrieving secrets

### Whitepapers

- Operational Excellence Pillar

### Videos

- Walking the Tightrope: Balancing Innovation, Reliability, Security
- A Day in the Life of a Netflix Engineer III

# Security Pillar

The security pillar includes the ability to protect information, systems, and assets while delivering business value through risk assessments and mitigation strategies.

**Topics**



## Definition

There are five best practice areas for security in the cloud:

- Identity and access management
- Detective controls
- Infrastructure protection
- Data protection
- Incident response

Managed services address some of today's biggest security concerns with analytics applications environments, as they minimize certain forms of infrastructure management tasks, such as operating system patching, binary patching, etc. Although the attack surface is reduced compared to non-managed analytics application architectures, Open Web Application Security Project (OWASP) recommendations and application security best practices still apply.

The questions in this section are designed to help you address specific ways an attacker could try to gain access to or exploit misconfigured permissions that could lead to abuse. The practices described in this section strongly influence the security of your entire cloud platform and so should not only be validated carefully but also reviewed frequently.

The infrastructure protection best practice area will not be described in this document because the practices from the AWS Well-Architected Framework still apply.

# Design Principles

In the cloud, there are a number of principles that help you strengthen your system's security. In particular, the following is emphasized for analytics workloads. See also the design principles in the AWS Well-Architected Framework whitepaper.

- **Enable data traceability**: Monitor, alert, and audit actions and changes to the lineage of data in your environment in real time. Integrate metrics with systems to automatically respond and take action.

# Best Practices

**Topics**



## Identity and Access Management

Identity and access management are key parts of an information security program, ensuring that only authorized and authenticated users are able to access your resources, and only in a manner that you intend. For example, you should define principals (that is, users, groups, services, and roles that take action in your account), build out policies aligned with these principals, and implement strong credential management. These privilege-management elements form the core of authentication and authorization. Analytics applications often require multiple interdependent services with differing access patterns and methods, so it's also important to appropriately scope the service-level inherited access those services provide throughout your environment.

| **ANALYTICS_SEC 1: How do you authenticate access to your analytics applications within your organization?** |
| --- |
| |

Authentication is the process of an entity (a user or application) proving its identity to an application or system. Big data services within AWS offer multiple authentication mechanisms and you can choose the authentication mechanism that best fits your organization.

AWS Identity and Access Management (IAM) supports federating authentication with Active Directory or other directory services using LDAP. Many large organizations implement a federated authentication

mechanism. IAM roles are assumed by the federated user, which then provide authentication to downstream services. For example, in Amazon Redshift, IAM roles can be mapped to Amazon Redshift DB Groups, Amazon EMR IAM roles can be mapped to an EMR Security configuration or Apache Range AD group-based policy, and in AWS Glue, IAM roles can be mapped to Glue catalog resource policies.

Here are some authentication options for each service. Refer to the service-specific documentation for more details:

| Service | Authentication Options |
| --- | --- |
| AWS Lake Formation | IAM authentication, Lake Formation permissions for Data Catalog access |
| Amazon Athena | IAM authentication, SSH Keys for JDBC connections, Federated identity, cross-account, EC2 instance profile |
| Amazon Redshift | IAM Authentication, Native database authentication |
| Amazon EMR | IAM Authentication, LDAP Authentication (for HiveServer2, Hue, Presto, Zeppelin), Kerberos authentication, SSH Keys, Apache Knox |

| ANALYTICS_SEC 2: How do you authorize access to the analytics services within your organization? |
| --- |
| |

Authorization is the act of allowing or denying an identity to perform an action. This requires authentication first, as determining what an identity can do requires that the identity has been validated. Authorization for data can be broken up into storage authorization, metadata authorization and application authorization.

Analytics services and applications are often shared among teams within an organization. How you decide to organize authorization to services depends on whether you pursue a “fine-grained” or “coarse-grained” approach to user segmentation. With a fine-grained approach, teams share AWS account access with common resources, such as a large Amazon Redshift cluster owned by the organization, while individual databases are authorized by team at the IAM policy level. In contrast, the coarse-grained approach segments authorization by team, and each team maintains control of their individual accounts and all resources required for their job function within their account. Coarse-grained collaboration access is provided via cross-account roles that can be assumed by other teams. In this approach, for example, the “data lake team” maintains control of S3 buckets in one account, while the “data engineering team” uses the AWS Glue service in a separate account, where AWS Glue is authorized via cross-account roles with read-only access to read from the S3 data lake. The coarse-grained approach is preferred by Amazon to ease security management within organizations with a large number of users.

Here are some storage authorization options for each service. Refer to the service-specific documentation for more details:

| Service | Storage Authorization Options |
| --- | --- |
| Amazon S3 | S3 bucket policies and ACLs allow defining fine grained permission on S3 objects |
| Amazon EMR | Use EMRFS authorization for datasets in Amazon S3 via Security Configuration in the Amazon EMR service. |

| Service | Storage Authorization Options |
| --- | --- |
| | Enable "Secure Mode" in Hadoop. Use Hadoop ACLs. |

Metadata Authorization rules are the access-control rules you specify at the catalog or metadata level that store the definitions of tables, views, etc. Here are some metadata authorization options for each service. Refer to the service-specific documentation for more details:

| Service | Metadata Authorization Options |
| --- | --- |
| Apache Hive on Amazon EMR | SQL Standard-Based Hive Authorization,<br><br>Apache Ranger on EC2 can be used for fine-grained access control (limited to Apache Hive) |
| Amazon Redshift | SQL Standard-Based Authorization using grants to users and groups. |
| AWS Glue | Allows attaching fine-grained access control policies to catalog items |

Application Authorization rules are the rules that define what actions each user can perform on the resources for each service, for example, run a Glue Crawler, run a Glue ETL job, launch or terminate EMR Clusters, etc. Application Authorization Rules are defined in IAM across services. Here are some application authorization options for each service. Refer to the service-specific documentation for more details:

| Service | Application Authorization Options |
| --- | --- |
| Amazon Redshift | IAM policies define who can list, read, create and manage Amazon Redshift resources. |
| AWS Glue | IAM policies define who can read and modify Glue resources like Crawlers, Jobs etc. |
| Amazon EMR | IAM policies define who can list and modify Amazon EMR clusters |
| Amazon Athena | IAM policies assigned to the user define who can read, run query etc. in Amazon Athena |
| Amazon Kinesis | IAM policies assigned to the user define who can create, modify Amazon Kinesis Streams, Amazon Kinesis Data Firehose Delivery Streams, etc. |
| Amazon QuickSight | IAM Policies assigned to user define who is an Amazon QuickSight administrator or author or reader. |

## Detective controls

| ANALYTICS_SEC 3: How are you storing, monitoring and analyzing access and query logs? |
| --- |
| |

It's important to monitor and analyze data access and query logs. A common practice in large organizations is to use a central log management service, like Elasticsearch with Kibana to store and analyze log files. Amazon S3 is the best choice for long-term retention and archiving of log data, allowing easy search and discovery using Amazon Athena. VPC Flow Logs enable you to monitor connections to and between services used in your analytics application.

Here is a list of how common AWS services used in an analytics application store access and query logs. Refer to the service-specific documentation for more details:

| Service | Log Storage Options |
|---|---|
| Amazon Athena | Stores query requests in CloudTrail, Query results stored in Amazon S3 |
| Amazon S3 | Buckets can be enabled to store access logs |
| Amazon Redshift | Audit Logging can be enabled |
| Amazon EMR | Stores logs in Amazon S3. Custom Query logs for Presto, Hive etc. can be saved to Elasticsearch, Amazon S3, etc., using log4j appenders. |
| AWS Glue | Stores logs in Amazon CloudWatch |

## Infrastructure protection

See the Well-Architected Framework whitepaper for best practices in the **infrastructure protection** area for security that apply to analytics applications.

## Data Protection

Before architecting any analytics system, foundational practices that influence security should be in place. For example, data classification provides a way to categorize organizational data based on levels of sensitivity, and encryption protects data by way of rendering it unintelligible to unauthorized access. These methods are important because they support objectives such as preventing financial loss or complying with regulatory obligations.

| **ANALYTICS_SEC 4: How are you securing data at rest?** |
|---|
| |

Ensuring sensitive data is encrypted at rest is paramount to creating and maintaining a proper security posture. Analyze security and compliance requirements for your analytics application to determine how to encrypt your data at rest. When encrypting your data, design a proper key rotation policy. Equally important, use a key management system to ensure that you encrypt and decrypt your data in the proper way. For key management, use an external key management system or a hardware security module (HSM) to ensure that there is no unauthorized access to those keys. Keeping appropriate ration policy is also important to ensure the keys are not compromised.

Encrypting data within AWS starts by defining an encryption key management strategy. Questions to answer are: Do I have to manage my encryption keys?", "Do I need dedicated key management hardware?", and "Do I have to manage my keys on premises?" The following chart can help you choose the best solution for key management for your data at rest.

| Do I have to manage my encryption keys? | Do I need dedicated key management hardware? | Do I have to manage my keys on premises? | Strategy |
|---|---|---|---|
| No | No | No | Use AWS service managed |
| Yes | No | No | Use AWS KMS |
| Yes | Yes | No | Use AWS CloudHSM |
| Yes | No | Yes | Use your own KMS |
| Yes | Yes | Yes | Use your own HSM |

Amazon S3 allows both server-side encryption (SSE) as well as client-side encryption (CSE) for protecting data at rest. For SSE, three mutually exclusive options are available: Server-side encryption with Amazon S3-managed keys (SSE-S3), Server-side encryption with AWS KMS-managed keys (SSE-KMS), and Server-side encryption with customer-provided keys (SSE-C). For CSE, you have the following options: Use an AWS KMS-managed customer master key, or use a client-side master key.

Amazon EMR encryption of data at rest is configured with a managed "Security Configuration" object. This configuration file will configure EMRFS and local-volume encryption at rest. If needed, configure HDFS transparent encryption.

Amazon Redshift can configure encryption by selecting the "Encrypt database" option. Selecting this option will ensure encryption of data at rest as well as encryption of backups. Options for the key management store are AWS KMS and HSM.

AWS Glue supports data encryption at rest for ETL jobs and development endpoints. You can configure ETL jobs and development endpoints to use AWS Key Management Service (KMS) keys to write encrypted data at rest. You can also encrypt the metadata stored in the AWS Glue Data Catalog using keys that you manage with AWS KMS. Additionally, you can use AWS KMS keys to encrypt job bookmarks and the logs generated by crawlers and ETL jobs. Encryption settings for crawlers, ETL jobs, and development endpoints can be configured using the security configurations in Glue. AWS Glue Data Catalog encryption can be enabled via the settings for the AWS Glue Data Catalog.

| **ANALYTICS_SEC 5: How are you securing data in transit?** |
|---|
| |

Data security is also required for data in transit. Here are some options for encrypting data in transit for common AWS services used in analytics application. Refer to the service-specific documentation for more details:

| Point "A" | Point "B" | Data Flow Protection |
|---|---|---|
| Enterprise data sources | Amazon S3 | Encrypted with SSL/TLS; S3 requests signed with AWS Sigv4 |
| Amazon S3 | Amazon EMR Amazon Redshift Amazon Athena | Encrypted with SSL/TLS |
| Amazon Athena | Clients | JDBC connections use SSL/TLS by default |

| Point “A” | Point “B” | Data Flow Protection |
|---|---|---|
| Amazon EMR | Clients | Encrypted with SSL/TLS; varies with Hadoop application client |
| Amazon Redshift | Clients | Supports SSL/TLS; Requires configuration |
| Apache Hadoop on Amazon EMR | | • Hadoop RPC encryption<br>• HDFS block data transfer encryption<br>• KMS over HTTPS is not enabled by default with Hadoop KMS<br><br>May vary with Amazon EMR release (such as Tez and Spark in release 5.0.0+) |

| **ANALYTICS_SEC 6: How are you protecting sensitive data within your organization?** |
|---|
| |

Sensitive data is data that must be protected from visibility—even within your organization. Examples of sensitive data are those that have a high potential for abuse or fraud and includes financial information, like credit card numbers, personally-identifying information, like government ID numbers, and those records required by law to maintain tighter access controls. When designing an analytics workflow, understand how and where sensitive information is being processed and take measures to protect it.

Before sensitive data can be protected, it must first be identified as sensitive data. Amazon Macie is a security service that uses machine learning to automatically discover, classify, and protect sensitive data in AWS. Amazon Macie recognizes sensitive, data such as personally identifiable information (PII) or intellectual property, and provides you with dashboards and alerts that give you visibility into how this data is being accessed or moved.

Data stored in Amazon S3 can be tagged with designations to indicate the sensitivity, and the type of sensitive data that is present. Access to that sensitive data can be limited using IAM policies that contain Condition clauses. Amazon Athena uses permissions granted from the storage layer, so by controlling access to data in Amazon S3, you can restrict users from querying it using Athena. Similarly, sensitive data access in Amazon EMR and Athena can be restricted using tags and IAM policies. Sensitive data in Amazon Redshift can be restricted by creating views that do not contain sensitive data columns, and only allowing access to users to those non-sensitive views.

Explicitly denying permission for tagging actions is an important consideration. This prevents users from granting permissions to themselves that you did not intend to grant through tags.

## Incident Response

Even with extremely mature preventive and detective controls, your organization should still put processes in place to respond to and mitigate the potential impact of security incidents. The architecture of your workload strongly affects the ability of your teams to operate effectively during an incident, to isolate or contain systems, and to restore operations to a known good state. Putting in place the tools and access ahead of a security incident, then routinely practicing incident response through game days, will help you ensure that your architecture can accommodate timely investigation and recovery. For

further recommendations on incident response, see the AWS Well-Architected Framework Security Pillar whitepaper.

| **ANALYTICS_SEC 7: How are you defining and enforcing policies to respond to security alerts and incidents?** |
| --- |
| |

It's critical to respond to security alerts and events related to your analytical applications within a pre-defined and accepted SLA. Data is your most important asset and if there is a breach in the security of your analytics infrastructure, your entire data asset could be compromised if the breach is not contained in a timely manner. Hence, it's important to proactively identify these events and remediate them as soon as possible. For example, it's important to scan for sensitive information, like authentication keys and passwords to your databases, being embedded within public or private code repositories that could compromise access to those databases. Moreover, if you notice unusual access patterns in a given time period from your data stores, you should investigate it in an urgent and timely manner. Every organization is different and it's important to work closely with your security and compliance teams to decide on the appropriate SLAs for each of the security-related events.

AWS provides multiple services to define, enforce, and respond to security alerts and incidents. AWS CloudTrail is a service that enables governance, compliance, operational auditing, and risk auditing of your AWS account. With CloudTrail, you can log, continuously monitor, and retain account activity related to actions across your AWS infrastructure. Amazon Macie is a security service that makes it easy for you to discover, classify, and protect sensitive data in Amazon S3. Macie collects AWS CloudTrail events and Amazon S3 metadata such as permissions and content classification. Amazon GuardDuty is a managed threat detection service that continuously monitors for malicious or unauthorized behavior to help protect your AWS accounts and workloads. Amazon GuardDuty gives you intelligent threat detection by collecting, analyzing, and correlating billions of events from AWS CloudTrail, Amazon VPC Flow Logs, and DNS Logs across all of your associated AWS accounts.

# Resources

Refer to the following resources to learn more about our best practices for operational excellence.

## Documentation & Blogs

- Connect to Amazon Athena with federated identities using temporary credentials
- Best Practices for Securing Amazon EMR
- IAM Policies and Bucket Policies and ACLs! Oh, My! (Controlling Access to S3 Resources)
- Security, Protecting and Managing Data
- Video: Securing Enterprise Big Data Workloads on AWS
- Video: Best Practices to Secure Data Lake on AWS

## Whitepapers

- Amazon Web Services: Overview of Security Processes
- Big Data Analytics Options
- Security Pillar: AWS Well-Architected Framework

## Partner Solutions

- AWS Big Data Partner solutions

## Third-Party Tools

- Lumada Data Catalog
- Collibra
- Okera
- Apache Atlas
- Apache Knox
- Apache Ranger

# Reliability Pillar

The reliability pillar includes the ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or transient network issues.

**Topics**



## Definition

There are three best practice areas for reliability in the cloud:

- Foundations
- Change management
- Failure management

To achieve reliability, a system must have a well-planned foundation and monitoring in place, with mechanisms for handling changes in demand, requirements, or mitigating the risk of data store failure. The system should be designed to detect failure and, ideally, automatically heal itself.

## Design Principles

In the cloud, a number of principles help you increase reliability. In particular, the following are emphasized for analytics workloads. For more information, refer to the design principles in the AWS Well-Architected Framework whitepaper.

- **Manage the lifecycle of data assets, transitioning and expiration:** Apply a governance process to how datasets and assets are maintained. Establish a review cycle on the relevance and freshness of a dataset. Ensure that operational maintenance cycles on managed datasets are being maintained. Such governance can include how data is updated or refreshed, institutional input tracking on data value and use, and criteria and process for data expiration including the use of tiered storage for cost management of data.
- **Enforce Data Hygiene**: When managing datasets for institutional use, apply mechanisms to assure data model standards are defined and enforced. Every dataset being used within an institutional workload should have a governance model applied to it with a repository that records its control, access, cleanliness standard, usage lineage, and overall management

- **Preserve data lineage:** Derived datasets are mutable; original data is not. The process of ingesting data into a data lake begins with the unmodified original “raw” data from which all downstream processes and manipulations derive. This raw dataset may undergo multiple transformations before it can be used by downstream analytical applications. It's important to maintain traceability of data attributes as the data moves through each layer of the analytical system with data lineage capture. A metadata repository that maps and tracks the schema changes can be used to capture such lineage information.

# Best Practices

**Topics**



## Foundations

See the Well-Architected Framework whitepaper for best practices in the **foundations** area for reliability that apply to analytics applications.

## Change Management

| **ANALYTICS_REL 1: How do you track changes to the metadata in your data warehouse?** |
| --- |
| |

You should have a centralized metadata repository to track changes in the source and target. There should be a convenient path to propagate those changes to downstream systems that depend on the source and target systems. For example, if a column in the source table changes, the ETL job that reads the column should be modified to read from the modified column and any metric calculation in the reporting system that depends of the column should be modified.

It’s also essential to have the ability to track how the value of a target metric is calculated from the source as the data traverses multiple layers of the data warehouse. This helps in debugging any errors in the value of the metrics. The system should have the capability to provide an audit trail of changes to the metadata over a period of time.

On AWS, use the AWS Glue Data Catalog to maintain a repository of metadata for source and target systems. The AWS Glue Data Catalog also integrates with other AWS services including Athena and EMR so that you can maintain consistency between the query environments and ETL environments. For example, the file that is loaded into Amazon S3 using a Glue ETL job can be a part of the AWS Glue Data Catalog, which is shared with Athena so it can be seamlessly queried without a need to move the file into a different service. The AWS Glue Data Catalog can also be used as a Hive metastore running on Amazon EMR.

| **ANALYTICS_REL 2: How do you maintain the configuration settings of your analytics environment to ensure consistency across versions?** |
| --- |
| |

Maintain the configuration parameters and environment as code by templatizing resources including databases, Hadoop clusters, and query tools. This allows for easy setup and consistent deployment of the big data environment.

Tracking changes to the configuration using a code repository enables you to track changes in the environment and also to roll back to an earlier version of the environment in case there are issues with the configuration changes.

AWS Config allows you to track changes in the configurations of AWS services. In addition, you can maintain consistency between various big data environments by using CloudFormation, which enables you to easily manage the infrastructure for AWS resource provisioning and updates. You can also use AWS OpsWorks, which is a Chef-based configuration and automation service.

## Failure Management

| **ANALYTICS_REL 3: How do you recover from failures in your primary data warehouse?** |
| --- |
| |

Downtime in a primary data warehouse can be catastrophic for a business. The data warehouse powers the downstream analytical applications, some of them mission critical, and a proper recovery strategy needs to be in place so that it can recover from failures in a timely manner, consistent with the organization's recovery time objective (RTO) and recovery point objective (RPO).

AWS provides you with the opportunity to design fault tolerant architectures when it comes to databases. For example, Amazon Redshift in a multi-node cluster continuously monitors itself for node failures and in the event of a failure, automatically creates a copy of the data and replicates it on a healthy node, while ensuring that the queries are served. Amazon RDS has Cross-Region Replication and Multi-AZ deployment options that allow you to maintain a synchronous copy of the database in another Availability Zone while ensuring read performance. RDS and Amazon Redshift support automated snapshots to durable object storage should the data warehouse need to be recovered from a catastrophic failure.

| **ANALYTICS_REL 4: How do you recover from failures of your ETL jobs?** |
| --- |
| |

ETL jobs allow data to be ingested from multiple disparate source systems, transformed for downstream consumption and loaded into a data mart for reporting. An ETL job can be a batch or stream workload, depending on the velocity of data ingestion. Moreover, ETL job starts can be schedule-based or event-based. A failure in the ETL step could result in outdated or incorrect metrics in your analytical application. It is essential to ensure that the ETL jobs continue to run and can recover in case of failures.

AWS Glue is a serverless service that allows you to run batch ETL jobs without the need for maintaining servers. This reduces the chances of failures in the batch ETL job considerably. Similarly, Amazon Kinesis Data Streams and Amazon Kinesis Data Firehose are serverless stream processing services that reduce the risk of failures in stream-based ETL. AWS Lambda is a service that allows you to run code without worrying about servers. It can be used to process small event-based or schedule-based ETL logic or can also be used as a trigger for longer running batch ETL jobs.

# Resources

Refer to the following resources to learn more about our best practices for operational excellence.

## Documentation & Blogs

- AWS Glue - Running and Monitoring AWS Glue
- Amazon EMR – View and Monitor a Cluster

- Monitoring the Amazon Kinesis Data Streams Service with Amazon CloudWatch

### Whitepapers

- Reliability Pillar

# Performance Efficiency Pillar

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve.

**Topics**



## Definition

Performance efficiency in the cloud is composed of four areas:

- Selection
- Review
- Monitoring
- Tradeoffs

Take a data-driven approach to selecting a high-performance architecture. Gather data on all aspects of the architecture, from the high-level design to the selection and configuration of resource types. By reviewing your choices on a recurring basis, you can ensure that you are taking advantage of the continually evolving AWS Cloud. Monitoring ensures that you are aware of any deviance from expected performance and can take action on it. Finally, you can make tradeoffs in your architecture to improve performance, such as using compression or caching, or relaxing consistency requirements.

The review and trade-offs best practice areas will not be described in this document because the practices from the AWS Well-Architected Framework still apply.

## Design Principles

When designing for analytics workloads in the cloud, a number of principles help you achieve performance efficiency:

- **Use data profiling to improve performance.** Store prepared data in an appropriate environment based on data access and query retrieval patterns. Use business and application requirements to define performance and cost optimization goals. Published data should have service design goals (such as data refresh rate.) Included in a data profile are data statistics (sums, averages), column skew (disproportional frequency of a value), and missing data. These characteristics of the data profile can affect query performance and partitioning strategies, and should be monitored.
- **Continuously optimize data storage.** Data storage optimization—especially compression, partitioning columns, distribution keys, and sort keys—need to be continuously evaluated and improved as query patterns change.

# Best Practices

**Topics**



## Selection

**ANALYTICS_PERF 01: How do you select file formats and compression to store your data?**

In many analytics applications, the data loading performance is the bottleneck to data processing. By compressing data at rest, and utilizing AWS services that can read compressed data formats, you can lower the cost for storage, as well as improve performance by reducing the data scanned by queries. Using a storage format that allows partitioning data can further reduce the amount of data scanned, improving performance efficiency.

There are a variety of storage options related to big data processing in Amazon S3. You have the option of storing data in open formats, such as delimited text (for example, CSV), ORC, Parquet, Avro, and SequenceFile. You might also choose to compress the data as GZIP, BZIP, LZO, Snappy, and more. Each of these formats and compression algorithms provides characteristics that can optimize your workload for performance as well as storage and query costs.

Analytics applications using frameworks running on Amazon EMR, Athena, AWS Glue, and Redshift Spectrum benefit from columnar formats and compression for data storage in Amazon S3. Amazon Redshift storage also presents options for compressed storage. Amazon Redshift column compression and encoding can improve query performance. Compression also reduces your storage footprint, therefore reducing your Amazon Redshift cost. Use splittable compression where possible, as it allows workers running in parallel to read and work on different chunks of data from a single object.

Use columnar formats where possible. Columnar formats improve performance by reducing the data scanned. When you query data, you are commonly only retrieving a few columns out of what could be hundreds of columns. Columnar file formats allow big data processing engines to retrieve only the data from columns that are requested.

**ANALYTICS_PERF 02: How do you determine compute options and sizing for your analytics applications?**

Run performance tests specific to each workload to determine the sizing of your architecture and any bottlenecks, and to ensure that you can meet target SLAs for data loading, processing, and freshness.

Analytics services on AWS offer a large number of compute options optimized for CPU, GPU, memory, and storage. Selecting suitable instance counts and instance types for your big data applications can potentially provide performance improvements and cost savings. For example, many analytics workloads

are time sensitive to meet business operations reporting requirements. Performance metrics should be examined in the context of the overall analytics workload end-to-end completion time, while monitoring individual steps in the analytics pipeline.

Monitor your application resources, such as EC2 instances, EMR clusters, RDS instances, Amazon Redshift clusters, ElastiCache, and AWS Glue to ensure that you are meeting performance objectives without being over- or under-provisioned.

**ANALYTICS_PERF 03: How do you organize or partition your data in your data lake?**

By partitioning your data, you can reduce the amount of data scanned by each query, thus improving performance and reducing cost.

Monitor user query behavior to find columns frequently used in filters and “group by” statements. You can select these columns for partitioning for best results. You can partition your data by any column. A common practice is to partition the data based on time, often leading to a multi-level partitioning scheme. For example, if you have data coming in every hour, you might decide to partition by year, month, day, and hour. If you have data coming from many different sources but loaded only one time per day, you might partition by a data source identifier and date. If you have multiple business groups or customers querying the data, you might also consider partitioning by business group ID or customer ID columns in your data.

Avoid over-partitioning, as it can lead to added overhead and smaller files in your S3 buckets, which can be detrimental to performance. Referring to the example of partitioning by year, month, day, and hour, this partitioning scheme might be too coarse if most queries are filtering on year. Partitioning is use-case dependent and might even be fluid in your organization, as one size may not fit all use cases.

## Review

See the Well-Architected Framework whitepaper for best practices in the **review** area for performance efficiency that apply to analytics applications.

## Monitoring

**ANALYTICS_PERF 04: How are you monitoring the performance of data ingestion, batch jobs, and ETL jobs?**

Monitoring analytics applications ensures that you are using empirical data to determine horizontal scale-out strategies, compute selection, and that actively manage service limits that may impact your performance.

## Ingestion Monitoring

AWS provides services such as Kinesis Data Streams and Kinesis Data Firehose, AWS Database Migration Service (AWS DMS), and AWS Glue to simplify your large-scale data ingestion. You can also develop your own ingestion applications using frameworks on Amazon EMR or Amazon EC2. Like other applications and infrastructure, these services should be monitored to ensure they are being scaled appropriately.

For example, Kinesis Data Streams should be monitored at the stream and shard level to ensure appropriate resharding takes place, and that you are scaling the Kinesis producer and consumer applications to keep up with demand. Glue ETL Jobs should be monitoring for Data Processing Unit

(DPU) memory and CPU consumption to ensure you have enough DPUs for jobs to succeed in an efficient time-frame. AWS DMS offers continuous change data capture replication, in which case you must select DMS Replication instance types that are suitable for the desired throughput.

## Batch and ETL Job Monitoring

Many analytics applications include batch and ETL jobs to run complex calculations or aggregations, simulations, forecasting, or to train machine learning models on huge datasets in an offline scenario. These jobs can often take hours or even days.

These jobs should ordinarily be orchestrated and scheduled in an automated fashion. In addition to relying on compute and storage utilization metrics for the batch and ETL jobs, you should also instrument your job orchestration and scheduling systems to provide end to end metrics and alerts for jobs. Although this is not an exhaustive list, you should consider monitoring job execution time, job SLAs, and overall job compute utilization.

## CloudWatch Metrics and Alerts

When using higher-level AWS analytics services, you should leverage the many built-in CloudWatch Metrics and create alerts. These alerts can be used both for notification purposes, as well as automating the scaling policies on your applications themselves.

For big data applications running on Amazon EC2, you might need to instrument for additional monitoring metrics and logging. You can use CloudWatch Logs and Custom Metrics to transmit custom application metrics to Amazon CloudWatch. You can also engage with our APN Partners who can help you simplify and enhance your monitoring and logging for analytics application profiling.

| **ANALYTICS_PERF 05: How are you monitoring the performance of your queries?** |
| --- |
| |

Monitoring query and retrieval performance over time is important so that you can continuously re-evaluate whether you are using appropriate data stores, databases, and file formats for your use cases. Most analytics applications ultimately persist ingested data and processed results in a durable location for querying or retrieval, such as queries from applications, ad hoc SQL queries on a data lake, machine learning training jobs, business intelligence queries from a data warehouse, and more.

You should monitor query performance of services such as Amazon Redshift, Athena, Amazon EMR, DynamoDB, in order to make informed decisions on performance optimization. This will vary depending on the AWS features and services you are using and the overall architecture.

## Tradeoffs

See the Well-Architected Framework whitepaper for best practices in the **tradeoff** area for performance efficiency that apply to analytics applications.

# Resources

Refer to the following resources to learn more about our best practices for performance efficiency.

## Documentation & Blogs

- AWS Glue - Running and Monitoring AWS Glue
- Amazon EMR – View and Monitor a Cluster

- How Realtor.com Monitors Amazon Athena Usage with AWS CloudTrail and Amazon QuickSight
- Top 10 Performance Tuning Tips for Amazon Athena
- Monitoring the Amazon Kinesis Data Streams Service with Amazon CloudWatch
- Unit Tests for Data

**Whitepapers**

- Performance Efficiency Pillar

# Cost Optimization Pillar

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment.

**Topics**



## Definition

There are four best practice areas for cost optimization in the cloud:

- Cost-effective resources
- Matching supply and demand
- Expenditure awareness
- Optimizing over time

As with the other pillars, there are trade-offs to consider. For example, do you want to optimize for speed to market or for cost? In some cases, it's best to optimize for speed—going to market quickly, shipping new features, or simply meeting a deadline—rather than investing in upfront cost optimization. Design decisions are sometimes guided by haste as opposed to empirical data, as the temptation always exists to overcompensate "just in case" rather than spend time benchmarking for the most cost-optimal deployment. This often leads to drastically over-provisioned and under-optimized deployments. The following sections provide techniques and strategic guidance for the initial and ongoing cost optimization of your deployment.

Analytics workloads primarily differ from traditional workloads in their scale, both in the storage of data and the computational requirements. The business drivers that derive value from analytics often have time-sensitive response requirements that can push architectures toward over-provisioning. However, considering the following questions can result in a cost-optimized analytics platform.

## Design Principles

In the cloud, the cost optimization design principles from the AWS Well-Architected Framework whitepaper are recommended and do not vary for analytics workloads.

# Best Practices

**Topics**



## Cost-Effective Resources

| **ANALYTICS_COST 01: How are you choosing the right compute solution for analytics applications?** |
| --- |
| |

During the initial planning and testing phases of analytics workloads, using on-demand resources provides immense flexibility to iterate through many alternative architectures until all aspects of a well-architected analytics project are optimized, such as selecting the right tool for the job to meet the required time-duration of analytics pipeline runs. On-demand resource billing allows low-commitment testing across multiple scenarios and scales. At the end of the testing period, a project should have an understood set of resource requirements: the number, type, and size of resources, as well as the steady-state or spikiness of the workload.

Reserved Instance commitments provide significant cost savings for those resources that have steady utilizations. Reserved capacity can be purchased for DynamoDB, RDS, Amazon Redshift, Amazon EMR, and ElastiCache.

For transient workloads, or those that implement stateless logic, consider using Amazon EC2 Spot Instances for additional cost savings. You can tightly integrate Spot Instances with other AWS services, such as Amazon EMR, Amazon EC2 Auto Scaling, AWS Auto Scaling, CloudFormation, AWS Batch, Data Pipeline, and Amazon Elastic Container Service (Amazon ECS). You also can integrate with non-AWS services, including Jenkins, Bamboo, Alces Flight, and Terraform. Spot Instances are an excellent option for reducing costs for development or quality assurance (Dev/QA) applications.

On Amazon EMR, instance fleets give you a wider variety of options and automated intelligence for instance provisioning. For example, you can now provide a list of up to five instance types with corresponding weighted capacities. Amazon EMR automatically provisions On-Demand and Spot Instance capacity across these instance types when creating your cluster. This can make it easier and more cost effective to quickly obtain and maintain your desired capacity for your clusters.

| **ANALYTICS_COST 02: How are you optimizing data stored in your data lake for cost?** |
| --- |
| |

It's very common for data assets to grow exponentially year over year in a data lake. With this exponential growth, it's important for you to optimize your data lake for cost.

Amazon S3 offers a variety of storage classes that enable you to optimize costs over time. You could consider a lifecycle plan that migrates data from S3 Standard to S3-Infrequent Access (or in some cases S3-Infrequent Access Single-AZ) to Amazon S3 Glacier class to be a sufficient cost-savings measure. Newer services that allow directly querying data in Amazon S3, such as Redshift Spectrum, Athena, Amazon S3 Select, and S3 Glacier Select, allow you to change how data should be stored for analytics. In many cases, adding a hive-style partitioning structure to the Amazon S3 prefix naming convention,

compressing your data, and storing data in columnar formats can improve query speed as well as reduce the cost of query transactions.

Amazon Athena is a serverless query service that makes it easy to analyze data directly in Amazon S3 using standard SQL. With Amazon Athena, you only pay for the queries that you run. You are charged based on the amount of data scanned per query. You can achieve significant cost savings and performance gains by compressing, partitioning, or converting your data to a columnar format, because each of those operations reduces the amount of data that Athena needs to scan to execute a query. Amazon Athena supports a wide variety of data formats, such as CSV, TSV, JSON, and text files, and also supports open source columnar formats, such as Apache ORC and Apache Parquet. Athena also supports compressed data in Snappy, Zlib, LZO, and GZIP formats. By compressing, partitioning, and using columnar formats, you can improve performance and reduce your costs.

S3 Select and S3 Glacier Select enable applications to retrieve only a subset of data from an object by using simple SQL expressions. You can use Amazon S3 Select to query objects in Apache Parquet format, JSON Arrays, and GZIP or BZIP2 compression for CSV and JSON objects. GZIP and BZIP2 are the only compression formats that Amazon S3 Select supports.

Amazon Redshift also includes Redshift Spectrum, allowing you to directly run SQL queries against exabytes of unstructured data in Amazon S3. Amazon Redshift Spectrum charges you by the amount of data that is scanned from Amazon S3 per query. Parquet stores data in a columnar format, so Amazon Redshift Spectrum can eliminate unneeded columns during the scan. Various tests have shown that partitioned Parquet files are not only performing faster, but they are also much more cost-effective than non-partitioned row-based CSV files. Redshift Spectrum supports multiple structured and semistructured data formats. Refer to the Redshift Spectrum documentation for all supported formats.

For MapReduce workloads, consider storing data in Amazon S3 and using the Amazon EMR File System (EMRFS). EMRFS is an implementation of HDFS that all Amazon EMR clusters use for reading and writing regular files from Amazon EMR directly to Amazon S3. EMRFS provides the convenience of storing persistent data in Amazon S3 for use with Hadoop while also providing features like consistent view and data encryption. With EMRFS, it is much easier to launch transient Amazon EMR clusters, saving money by only paying for the time the cluster is used.

**ANALYTICS_COST 03: How are you tiering your storage for cost effectiveness?**

Tiered storage transitions datasets to different storage tiers based on the usage patterns to optimize storage cost. For example, “hot data” (frequently accessed data) is generally kept in in-memory stores (caches), “warm data” (less frequently accessed data) in object stores, databases and “cold data” in magnetics drives or tapes for archival. It is highly recommended that you consider tiering your storage for your analytics workload.

AWS provides multiple options to choose from when selecting a storage layer for your analytical applications. Selecting the correct storage layer is the foundation for your analytical infrastructure and you should use the right service to achieve the desired scale. For details on AWS storage offerings, refer to this link.

The chart below shows how data can be tiered and moved between the different AWS storage services.

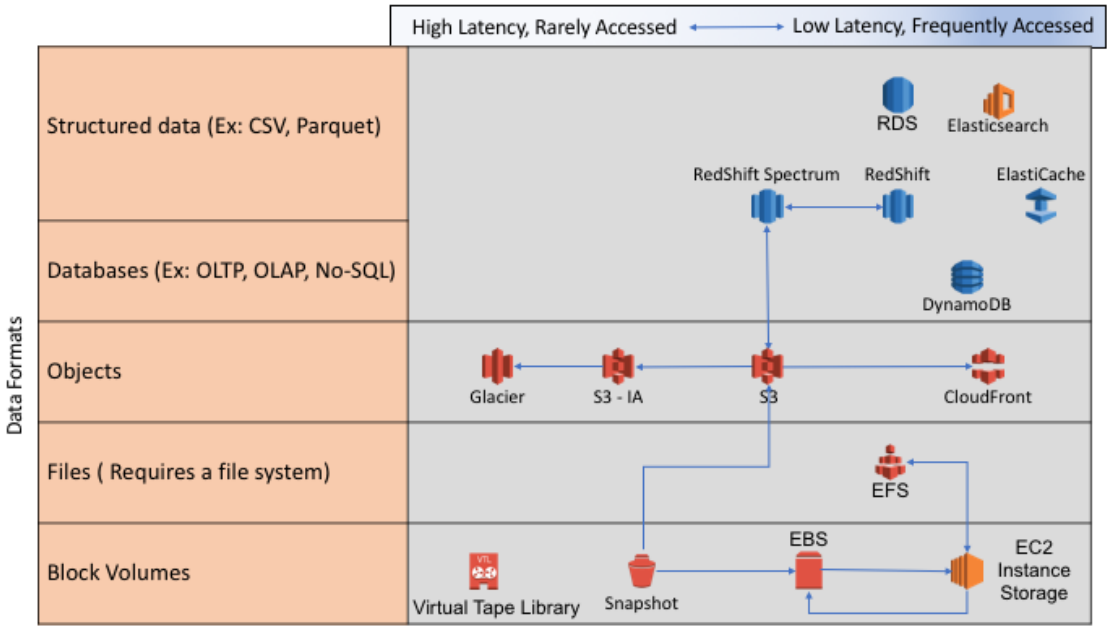


*Figure 9: Tiered Storage on AWS*

The left column represents the common data formats that you would build analytical applications on. The right side lays out the different AWS services. The services on the right are for low latency frequently accessed data and the services on the left are for high latency rarely accessed datasets. The arrows denote how you can move from data from one service to the other.

- **Structured Data and Databases:** This layer represents datasets that exist in structured formats like CSV, Parquet and ORC or datasets that are stored in OLAP or OLTP databases. These are ideal for fast query performance and are the desired format for low latency applications. As noted in the above chart, structured data residing on S3 can be used seamlessly with Redshift Spectrum to create external tables. The data in the external tables can be merged with Amazon Redshift physical layer tables via SQL statements. Other services we have in this layer include ElastiCache that allows in memory processing of datasets for faster response times, Elasticsearch that allows you to build search capabilities on datasets, RDS for storing data in OLTP format and Amazon DynamoDB for storing data queried using NoSQL techniques. You can choose from these services to store data that is frequently accessed and requires low latency retrievals.
- **Objects:** Amazon S3 is the object storage service from AWS that's high on durability. It interfaces with other AWS compute services like EC2, EMR, Lambda, DynamoDB and Amazon Redshift to allow for scalable compute architectures. S3 provides different options for storage tiers that you can choose from. S3 Standard is used for objects that are frequently accessed and S3 Infrequent Access (S3 IA), which has a lower storage cost, are for objects that are less frequently accessed. S3 IA does have a retrieval fee associated to it so objects should be moved to this layer only when it's not accessed frequently. Objects that are almost never accessed can be archived to Glacier, which is the archival service from AWS.
- **Files:** For customer looking for hosting shared file systems on AWS, Amazon Elastic File System (EFS) is a great option. EFS provides a POSIX interface and an NFS mount point for the files system that can be mounted on multiple instances in parallel. It's great for scaling big data workloads horizontally as the same file can now be processed in parallel using multiple compute instances. Moreover, EFS storage grows with customer's usage so they do not need to pre-provision volumes which may result in over or under provisioning of file storage. For certain analytics workloads, Amazon FSx for Lustre provides a high-performance file system that works natively with Amazon S3.

- **Block Volumes:** Amazon EBS allows customer to attach a block storage to EC2 compute instances. These block volumes can exist beyond the lifecycle of the instance as EBS snapshots. The snapshots are saved on S3 and can be mounted on any EC2 instance for computing on the files stored in the volume. You can use Amazon Data Lifecycle Manager (Amazon DLM) to automate the creation, retention, and deletion of snapshots taken to back up your Amazon EBS volumes.

Transitioning between layers is achieved using lifecycle policies. You author lifecycle policies which are used to move objects automatically between these different layers. Lifecycle policies can be built using S3 storage management features that provide insights into how often a certain object has been accessed over a period of time. S3 also provides the S3 Intelligent-Tiering storage class. This storage class automatically transitions objects into two access tiers: frequent access and infrequent access, based on the access pattern on the object. This tiering happens automatically without you taking additional action. Any object that hasn't been accessed for 30 consecutive days is moved to infrequent access tier and when it's accessed again, it is moved back to frequent access tier.

For objects that need to be distributed across the globe and/or require caching near the request source for even lower latencies, customer can use Amazon CloudFront which is a content delivery network (CDN) service from AWS. It is available at multiple AWS edge locations and can be used to cache objects from S3.

When you are building a data lake on AWS, it's important to look at the various AWS storage options highlighted above and choose the right service for the datasets.

## Matching Supply and Demand

| **ANALYTICS_COST 04: What is your data lifecycle plan?** |
| --- |
| |

When designing a data lake or data analytics project, considerations such as required access patterns, transaction concurrency, as well as acceptable transaction latency will influence where data is stored. It is equally important to consider how often older data will be accessed, and to have a data lifecycle plan to migrate data tiers from hotter storage to colder, less-expensive storage, while still meeting all business objectives.

For example, when analyzing log data, perhaps the previous 90 days must stay accessible for query in S3 Standard class, whereas data between 91-365 days old can be migrated to S3 Infrequently Accessed class, and data older than 365 days can be migrated to S3 Glacier class. Determining when and how frequently data is accessed will indicate when data can be migrated between storage tiers, and can be codified into a data lifecycle plan.

Several AWS capabilities help you evaluate your current data storage needs as well as plan for implementing a data lifecycle plan. S3 Inventory and S3 Analytics allow identifying objects stored in S3 as well as analyze storage access patterns to help you decide when to transition the right data to the right storage class. For example, with S3 Analytics you can answer the following questions: How much of my storage did I retrieve? What percentage of my storage did I retrieve? How much of my storage is infrequently accessed? How old are the objects in my storage?

Additionally, for data warehouse workloads, infrequently accessed data in the warehouse can be migrated from local storage to S3. When the cold data is required for analytics, use services such as Athena or Redshift Spectrum to join data in relational or warehouse storage with directly-queried data located in S3. In this manner, the amount of storage provisioned in the database or data warehouse can be significantly less than would be necessary to store all the data locally.

| **ANALYTICS_COST 05: How are you calculating your individual data-processing step costs?** |
| --- |
| |

It is important to consider analytics workflow costs at each individual data processing step or individual pipeline branch. The benefit of understanding analytics workflow costs at this granular level will help determine where to focus engineering resources for development, as well as perform return-on-investment calculations for the analytics portfolio as a whole. For example, if a high-cost ETL job is run hourly, but the downstream analytics is performed nightly, determining the effective cost of the pipeline steps would provide justification to reduce the frequency of the ETL job.

In this context, "data processing step" is defined as an event which results in a response dataset, such as a SQL query, a MapReduce function, a data munge, an ML model inference, or even an import/export job. AWS provides many options for your analytics applications, so considering performance without considering cost may leave you over-provisioned for your business requirement.

Factors that will impact data processing costs include: data storage location, data processing frequency, ETL or job concurrency, data size per processing cycle, processing duration and/or allowed latency, data format, network traffic costs, as well as upstream and downstream processes.

A best practice when considering cost-optimizing a big data workflow is to create a data pipeline model. For each step calculate the cost of that step considering the above factors. For example, storing data in S3 Standard Infrequently Accessed class (S3-IA) costs less than storing data in S3 Standard while the data is at rest. However, when considering the transaction costs of repeatedly retrieving data from S3-IA, the combined cost of storage + retrieval will point to S3 Standard as the lower-cost option. Similar calculations can be performed for storing and transacting on data in Amazon Redshift vs. Redshift Spectrum, Athena vs RDS, etc. Each AWS service has been designed to serve a particular need and the cost of that service can guide a user to the optimum solution.

AWS detailed billing reports as well as third-party solutions from APN Partners can provide fine-grained cost breakdowns by resource to assist in calculating individual data processing step costs.

## Expenditure Awareness

**ANALYTICS_COST 06: How are you tracking the "freshness" of your data used by your analytics application?**

In conjunction with having a data lifecycle plan, it's important to track data "freshness." In many cases, maintaining a metadata repository for tracking data movement will be worthwhile, not only to instill confidence in the quality of the data, but also to identify infrequently updated data. Infrequently updated data can often move from relatively more expensive low-latency storage to relatively less expensive high-latency storage. Consider using an automated system to move data between tiers.

For example, consider a business that updates sales reports on a daily, a monthly, and an annual cadence. After transactions are aggregated for the daily report, the data can be migrated from an RDBMS to Amazon S3 where the data can be ingested in a batch-processing system such as Amazon EMR or am AWS Glue Job.

Herd is one open-sourced metadata catalog tool created by the Financial Industry Regulatory Authority (FINRA). In addition, there are commercial vendors who provide metadata management and governance solutions.

**ANALYTICS_COST 07: How are you managing data transfer costs in your analytics application?**

In AWS, data transfer charges apply to data transferred out of AWS to the internet or over Direct Connect, data transferred between Regions, and data transferred between Availability Zones. Data transfer into AWS and data transfer within an AZ is free.

It's a best practice to scale applications horizontally (by adding additional hosts) rather than vertically (by increasing host memory or CPU) to reduce the risk of a lost host taking down the application. Similarly, another best practice is to place horizontally-scaled hosts in multiple Availability Zones to reduce the risk of a single AZ outage taking down the application. Applications that are "chatty," or have significant network traffic between hosts, should be monitored to provide visibility of the network traffic costs and, if necessary, to justify re-architecting the application to reduce network traffic while still maintaining required availability and resiliency goals. For transient workloads, it might be more cost effective to scale horizontally within a single AZ and rerun if needed.

Special consideration should be given to applications where data is hosted in only one Region or in an on-premise data store but consumed in AWS. In many cases, it's cost effective to move data closer to the analytics tools in the cloud rather than to pay data transfer charges. Another option for applications repeatedly reading the same data fields is to place a data cache service between the application and the on-premises data store, such as Amazon ElastiCache.

VPC Flow Logs are one tool for monitoring data traffic in your AWS environment. VPC Flow Logs is a feature that enables you to capture information about the IP traffic going to and from a VPC, a subnet, or a specific network interface. Flow log data for a monitored network interface is recorded as flow log records, which are log events consisting of fields that describe the traffic flow source, destination, and bytes of traffic, among other fields. Periodically collecting and analyzing VPC Flow Logs can provide significant insight into characterizing analytics application network transit costs (especially in a multitenant environment). It's not recommended to continuously collect and store VPC Flow Logs records for already characterized applications, due to the volume of records and the cost of storage for these low-utility records outside of periodic characterization.

**ANALYTICS_COST 08: What is your cost allocation strategy for resources consumed by your analytics application?**

It's important to provide users and financial stakeholders visibility into costs for your analytics workload. This is especially true for high-business-value analytics workloads that may require significant computation to achieve results within short windows. The cost-allocation strategy you choose can differ when using a siloed or shared-tenancy architecture.

In the siloed approach to analytics, individual teams or business units retain the use of AWS resources for their line-of-business purposes. Cost allocation can be performed in several ways with the two primary methods being cost-allocation tagging, and multi-zccount strategy. In cost-allocation tagging, AWS resources are tagged with one or more metadata key-value pairs, where the key designates the cost metric and the value designates the line of business or bill-back group. For example, a key-value pair could associate one key "Cost Center" to the value "BI Tools" and another key "Deployment Stage" to the value "Test." Using multiple key-value pairs allows fine-grained cost reporting. Another approach to use with siloed analytics workloads is a multi-account strategy for higher levels of separation of security and billing.

AWS Organizations is a service that allows a nested hierarchy of groups of AWS accounts, where all account billing is consolidated under the top-most level master-payer account. All billable activity within each account is exposed as a separate line item on the monthly bill. Comparing the two methods for billing purposes, the multi-account strategy allows more simplicity of billing separation without enforcing tags on all resources, but with the added complexity of managing users and resources that may be redundant among multiple accounts.

In the shared-tenancy approach to analytics, an analytics platform is deployed in a manner that enables multiple lines of business to use it. Similarly, a shared-tenancy platform is often used to provide SaaS analytics to your customers. Cost-allocation in a shared-tenancy model requires using a combination of logging and cost modeling to determine the correct price per transaction or workload. Recommended metrics to capture in addition to the entity requesting the work include runtime duration, read/write capacity units, and data scanned. AWS CloudTrail and CloudWatch Logs enable the capture of API calls

and event logs, respectively, such that a cost-allocation model can be utilized to provide a per-session charge-back.

## Optimizing Over Time

See the Well-Architected Framework whitepaper for best practices in the **optimizing over time** area for cost optimization that apply to analytics applications.

# Resources

Refer to the following resources to learn more about our best practices for operational excellence.

## Documentation & Blogs

- Managing Your Cost Savings with Amazon Reserved Instances
- How Goodreads offloads Amazon DynamoDB tables to Amazon S3 and queries them using Amazon Athena
- Best practices for resizing and automatic scaling in Amazon EMR
- Work with partitioned data in AWS Glue
- Using Amazon Redshift Spectrum, Amazon Athena, and AWS Glue with Node.js in Production
- Documentation: Using Cost Allocation Tags

## Whitepapers

- Cost Optimization Pillar of the AWS Well-Architected Framework
- Laying the Foundation: Setting Up Your Environment for Cost Optimization
- Amazon EC2 Reserved Instances and Other Reservation Models
- Leveraging Amazon EC2 Spot Instances at Scale
- Creating a Culture of Cost Transparency and Accountability
- Right Sizing: Provisioning Instances to Match Workloads
- AWS Storage Optimization

# Conclusion

This lens provides architectural guidance for designing and building reliable, secure, efficient, and cost-effective analytics workloads in the cloud. We captured common architectures and overarching analytics design tenets. The document also discussed the well-architected pillars through the analytics lens providing you with a set of questions to consider for new or existing analytics architectures. Applying the framework to your architecture helps you build robust, stable, and efficient systems, leaving you to focus on running analytics pipelines and pushing the boundaries of the field to which you're committed. The analytics landscape is continuing to evolve as the ecosystem of tooling and processes grows and matures. As this evolution occurs, we will continue to update this paper to help you ensure that your analytics applications are well-architected.

# Contributors

The following individuals and organizations contributed to this document:

- Radhika Ravirala, Sr. Specialist Solutions Architect, Amazon Web Services
- Laith Al-Saadoon, Sr. Solutions Architect, Amazon Web Services
- Wallace Printz, Sr. Solutions Architect, Amazon Web Services
- Ujjwal Ratan, Principal ML/AI Life Sciences Solutions Architect, Amazon Web Services
- Neil Mukerje, EMR Solutions Architect, Amazon Web Services

# Further Reading

For additional information, see the following:

- AWS Well-Architected Framework
- Data Lakes and Analytics on AWS
- Big Data on AWS
- AWS Big Data Blog
- Data and Analytics Solutions space

# Document Revisions

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

| update-history-change | update-history-description | update-history-date |
| --- | --- | --- |
| Initial publication (p. 55) | Analytics Lens first published. | May 20, 2020 |

# Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.