
Data Analytics Lens

AWS Well-Architected Framework

Data Analytics Lens: AWS Well-Architected Framework

Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Introduction	1
How to use this lens	1
Workload context checklist	2
Design principles by pillar	3
Operational excellence	3
Security	3
Reliability	3
Performance efficiency	3
Cost optimization	3
Sustainability	3
Pillars of the Well-Architected Framework	4
Operational excellence	4
1 – Monitor the health of the analytics application workload	4
2 – Modernize deployment of the analytics jobs and applications	6
Security	9
3 – Designing data platforms for governance and compliance	9
4 – Implement data access control	17
5 – Control the access to workload infrastructure	21
Reliability	24
6 – Design resilience for analytics workload	24
7 – Govern data and metadata changes	27
Performance efficiency	29
8 – Choose the best-performing compute solution	30
9 – Choose the best-performing storage solution	31
10 – Choose the best-performing file format and partitioning	33
Cost optimization	36
11 – Choose cost-effective compute and storage solutions based on workload usage patterns	36
12 – Build financial accountability models for data and workload usage	39
13 – Manage cost over time	41
14 – Use optimal pricing models based on infrastructure usage patterns	44
Sustainability	46
15 – Sustainability implementation guidance	46
Scenarios	64
Data discovery	64
Characteristics	64
Modern data architecture	67
Characteristics	68
Reference architecture	69
Configuration notes	70
Batch data processing	71
Characteristics	72
Reference architecture	73
Configuration notes	74
Streaming ingest and stream processing	75
Characteristics	75
Reference architecture	77
Operational analytics	81
Characteristics	82
Reference architecture	83
Configuration notes	84
Data visualization	85
Characteristics	85
Reference architecture	86
Configuration notes	87

Conclusion 88

Contributors 89

Document revisions 90

Notices 91

AWS glossary 92

Data Analytics Lens

Publication date: **November 17, 2022** ([Document revisions \(p. 90\)](#))

This document describes the AWS Well-Architected Data Analytics Lens, a collection of customer-proven best practices for designing well-architected analytics workloads. The Data Analytics Lens contains insights that AWS has gathered from real-world case studies, and helps you learn the key design elements of well-architected analytics workloads along with recommendations for improvement. The document is intended for IT architects, developers, and team members who build and operate analytics systems.

How to use this lens

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, cost-effective, and sustainable 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. Also consider best practices and questions that have not been included in this document when designing your architecture. We therefore 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.

There are many methods for implementing a data platform, therefore there are many options your organization must consider before designing a solution. In this whitepaper, we detail specific architectural approaches that can meet your design objectives. You must consider whether the proposed solution meets your organizational architecture principles and whether your team resources have the experience to implement such a solution.

Workload context checklist

To understand your business's context better, you must gather the following information.

ID	Priority	Workload Context
<input type="checkbox"/> C1	Required	Name of the workload
<input type="checkbox"/> C2	Required	Description that contains the business purposes, key performance indicators (KPIs), and the intended users of the workload
<input type="checkbox"/> C3	Required	Review owner who leads the lens review
<input type="checkbox"/> C4	Required	Workload owner who is responsible for maintaining the workload
<input type="checkbox"/> C5	Required	Business stakeholders who sponsor the workload
<input type="checkbox"/> C6	Required	Business partners who have a stake in the workload, such as information security, finance, and legal
<input type="checkbox"/> C7	Recommended	Architecture design document that describes the workload
<input type="checkbox"/> C8	Recommended	AWS account IDs associated with the workload
<input type="checkbox"/> C9	Recommended	Regulatory compliance requirements relevant to the workload (if any)

Design principles arranged by pillar

These are the design principles outlined in this paper organized by pillar of the AWS Well-Architected Framework.

Operational excellence (p. 4)

- [1 – Monitor the health of the analytics application workload \(p. 4\)](#)
- [2 – Modernize deployment of the analytics jobs and applications \(p. 6\)](#)

Security (p. 9)

- [3 – Designing data platforms for governance and compliance \(p. 9\)](#)
- [4 – Implement data access control \(p. 17\)](#)
- [5 – Control the access to workload infrastructure \(p. 21\)](#)

Reliability (p. 24)

- [6 – Design resilience for analytics workload \(p. 24\)](#)
- [7 – Govern data and metadata changes \(p. 27\)](#)

Performance efficiency (p. 29)

- [8 – Choose the best-performing compute solution \(p. 30\)](#)
- [9 – Choose the best-performing storage solution \(p. 31\)](#)
- [10 – Choose the best-performing file format and partitioning \(p. 33\)](#)

Cost optimization (p. 36)

- [11 – Choose cost-effective compute and storage solutions based on workload usage patterns \(p. 36\)](#)
- [12 – Build financial accountability models for data and workload usage \(p. 39\)](#)
- [13 – Manage cost over time \(p. 41\)](#)
- [14 – Use optimal pricing models based on infrastructure usage patterns \(p. 44\)](#)

Sustainability (p. 46)

- [15 – Sustainability implementation guidance \(p. 46\)](#)

Pillars of the Well-Architected Framework

This section describes the design principles, best practices, and improvement suggestions that are relevant when designing your workload architecture. For brevity, only questions that are specific to analytics workloads are included in the Data Analytics Lens. We recommend you also read and apply the guidance found in each Well-Architected pillar. The pillars include topics related to foundational best practices for operational excellence, security, performance efficiency, reliability, cost optimization, and sustainability that are relevant to all workloads.

Pillars

- [Operational excellence \(p. 4\)](#)
- [Security \(p. 9\)](#)
- [Reliability \(p. 24\)](#)
- [Performance efficiency \(p. 29\)](#)
- [Cost optimization \(p. 36\)](#)
- [Sustainability \(p. 46\)](#)

Operational excellence

The operational excellence pillar includes the ability to support development and run workloads effectively, gain insight into your operations, and continually improve supporting processes and procedures that deliver business value.

Best practices

- [1 – Monitor the health of the analytics application workload \(p. 4\)](#)
- [2 – Modernize deployment of the analytics jobs and applications \(p. 6\)](#)

1 – Monitor the health of the analytics application workload

How do you measure the health of your analytics workload? Data analytics workloads often involve multiple systems and process steps working in coordination. It is imperative that you monitor not only individual components but also the interaction of dependent processes to ensure a healthy data analytics workload.

ID	Priority	Best practice
□ BP 1.1	Required	Validate the data quality of source systems before transferring data for analytics.

ID	Priority	Best practice
<input type="checkbox"/> BP 1.2	Required	Monitor the planned arrival of recent data from the source systems.
<input type="checkbox"/> BP 1.3	Required	Define and measure the schedules of recurring analytics jobs.
<input type="checkbox"/> BP 1.4	Required	Define and measure the runtime durations of analytics jobs.

For more details, refer to the following information:

- AWS Compute Blog: [Monitoring and troubleshooting serverless data analytics applications](#)
- AWS Big Data Blog: [Building a serverless data quality and analysis framework with Deequ and AWS Glue](#)

Best practice 1.1 – Validate the data quality of source systems before transferring data for analytics

Data quality can have an intrinsic impact on the success or failure of your organization's data analytics projects. To avoid committing significant resources to process potentially poor-quality data, your organization should understand the quality of the source data, and monitor the changes to data quality throughout the data pipeline.

Data source validation can often be performed quickly on a subset of the latest data range to look for data defects. Such defects include missing values, anomalous data, or wrong data types that could fail the analytics job completion or lead to completion of the job with inaccurate results.

For more details refer to following document:

- AWS Blog: [How to Architect Data Quality on the AWS Cloud](#)

Suggestion 1.1.1 – Implement data quality validation mechanisms

The critical attributes of data quality that should be measured and tracked through your environment are completeness, accuracy, and uniqueness. Validating and measuring your data quality using metrics is important to build trust in your data.

For more details, refer to the following information:

- AWS Big Data Blog: [Enforce customized data quality rules in AWS Glue DataBrew.](#)
- AWS Big Data Blog: [Build a data quality score card using AWS Glue DataBrew, Amazon Athena, and Amazon QuickSight.](#)

Suggestion 1.1.2 – Notify stakeholders and use business logic to determine how to remediate data that is not valid

It's usually more efficient to impute missing values, but in other cases it's more efficient to block processing until the data quality issue can be resolved at source.

Suggestion 1.1.3 – Score and share the quality of your datasets

To improve the ongoing trust in data quality and adoption of your organization's datasets, consider creating a data quality matrix that can be accessed by the relevant teams advertising the quality score

of your datasets and potential issues with the data. This information can be incorporated in your Data Catalog.

Best practice 1.2 – Monitor the planned arrival of recent data from the source systems

Under normal operations, data processing produces outputs to meet business deadlines. A single component of a data processing pipeline might be healthy because it is running effectively. However, if no data arrives, reporting deadlines could be impacted.

Suggestion 1.2.1 – Monitor for the arrival or availability of new data. Notify if the expected duration between data arrival is too long

When event-driven systems rely on incoming data to initiate an analytics processing job, if new data does not arrive, then the job will not begin. Implement systems to track the time between data checkpoints. If the analytics job does not start within the expected time window, alert the support team.

Best practice 1.3 – Define and measure the schedules of recurring analytics jobs

Recurring analytics jobs (jobs that run on a schedule) have a defined start time. Emit metrics indicating when the jobs have started and notify if the job has not started within a predefined amount of time.

Suggestion 1.3.1 – Monitor the start time for recurring analytics processing jobs compared to the expected start time

When the start of the job has exceeded a defined time window past the expected start time, emit an alarm.

Best practice 1.4 – Define and measure the runtime durations of analytics jobs

Consider the configuration of analytics job status (runtime or run status) or use metrics that indicate when data processes have started and ended to calculate runtime durations. The notification is set to check whether the job has started but has not ended within the expected threshold.

Suggestion 1.4.1 – Monitor the start and end times for analytics processing

To improve the reliance of your analytics workloads, it is essential to monitor and alert for each step of your analytics pipeline. This can be performed by using CloudWatch Logs or by taking advantage of features supported by AWS services, such as Amazon Redshift systems tables.

2 – Modernize deployment of the analytics jobs and applications

How do you deploy jobs and applications in a controlled and reproducible way? Using modern development practices, such as continuous integration/continuous delivery (CI/CD), can help ensure that changes are rolled out in a controlled and repeatable way.

Your team should use test automation to verify infrastructure, code changes, and data updates at every stage of your deployment lifecycle.

ID	Priority	Best practice
<input type="checkbox"/> BP 2.1	Recommended	Use version control for job and application changes.
<input type="checkbox"/> BP 2.2	Recommended	Create test data and provision staging environment.
<input type="checkbox"/> BP 2.3	Recommended	Test and validate analytics jobs and application deployments.
<input type="checkbox"/> BP 2.4	Recommended	Build standard operating procedures for deployment, test, rollback, and backfill tasks.

For more details, refer to the following information:

- AWS DevOps Blog: [Complete CI/CD with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline](#)
- AWS Big Data Blog: [AWS serverless data analytics pipeline reference architecture](#)
- AWS Whitepaper: [Building a Cloud Operating Model](#)
- AWS Big Data Blog: [Build a DataOps platform to break silos between engineers and analysts](#)

Best practice 2.1 – Use version control for job and application changes

Version control systems support tracking changes and the ability to revert to previous versions of an analytics system should changes cause unintended consequences. Your team should version control code repositories for both analytics infrastructure as code (IaC) and analytics applications logic.

Suggestion 2.1.1 – Use infrastructure as code and version control systems so that a failed deployment can be rolled back to a previous good state

Follow software development best practices when building analytics systems. For example, deploy resources using code templates, such as AWS CloudFormation or Hashicorp Terraform, so that all deployments occur exactly as intended. Use version control systems (for example, code repositories such as AWS CodeCommit or GitHub) to hold current and previous versions of your code templates. Using these tools, if a new change results in unwanted outcomes, you can easily roll back to the previous code template.

For more details, refer to the following information:

- AWS Whitepaper: [Introduction to DevOps on AWS](#)
- AWS Blog: [Automate building an integrated analytics solution with AWS Analytics Automation Toolkit](#)

Best practice 2.2 – Create test data and provision staging environment

Using a known and unchanging dataset for test purposes helps ensure that when changes are made to the analytics environment or analytics application code, test results can be compared to previous versions.

Confirming that the test datasets accurately represent real-world data allows the analytics workload developer to confirm the outcomes from the analytics job, as well as comparing test results to previous versions.

Your organization should use a staging environment for user access testing. Your organization should create logically separated AWS accounts for your development, test, staging, and production environments depending upon your development standards.

For more details, refer to the following information:

AWS Whitepaper: [Establishing your best practice AWS environment](#)

Suggestion 2.2.1 – Use a curated dataset to test application logic and performance improvements

Analytics projects that are being developed should use the same curated dataset to compare results between tests of different versions of your code. Using the same dataset for all tests allows demonstrating improvement over time, as well as making it easier to recognize regressions in your code.

To help control access to sensitive data, your organization should use data masking techniques when restoring development data to non-production environments. More information on data minimization techniques can be found in [Security \(p. 9\)](#).

For more details, refer to the following information:

AWS Database Blog: [Data Masking using AWS DMS \(AWS Data Migration Service\)](#)

Suggestion 2.2.2 – Use a random sample of recent data to validate application edge cases and help ensure that regressions have not been introduced

Use a statistically valid random sample of recent data to confirm that the analytics solution continues to perform under real-world conditions. Using a sample of recent data also allows you to recognize whether your dataset characteristics have shifted, or whether anomalous data has recently been introduced to your data.

Best practice 2.3 – Test and validate analytics jobs and application deployments

Before making changes in production environments, use standard and repeatable automated tests to validate performance and accuracy of results.

Suggestion 2.3.1 – Establish separate staging environments to test changes before going live

Use separate environments, such as development, test, and production, to allow feature development to be introduced without disrupting production systems. Test changes for accuracy and performance before changes are deployed into the production environment.

Suggestion 2.3.2 – Automate the deployment and testing when infrastructure and applications changes occur

Use services, such as AWS CodePipeline and AWS Step Functions, to test changes programmatically and validate performance results.

For more details refer to the following information:

- AWS Perspective Guidance: [Deploy an AWS Glue job with an AWS CodePipeline CI/CD pipeline](#)
- AWS DevOps Blog: [How to unit test and deploy AWS Glue jobs using AWS CodePipeline](#)

Best practice 2.4 – Build standard operating procedures for deployment, test, rollback, and backfill tasks

Standard operating procedures for deployment, test, rollback, and data backfill tasks allow faster deployments, reduce the number of errors that reach production. Using a standard approach also makes remediation easier if a deployment results in unintended consequences.

Suggestion 2.4.1 – Document and use standard operating procedures for implementing changes in your analytics workload

Standard operating procedures allow teams to make changes confidently, thus avoiding repeatable mistakes and reducing the chance of human error.

Suggestion 2.4.2 – Use automation to perform changes to underlying analytics infrastructure or application logic

Automated tests can determine when changes have unintended consequences and can roll back without human intervention.

Security

The security pillar encompasses the protection of data, systems, and assets to take advantage of cloud technologies to improve your security.

Best practices

- [3 – Designing data platforms for governance and compliance \(p. 9\)](#)
- [4 – Implement data access control \(p. 17\)](#)
- [5 – Control the access to workload infrastructure \(p. 21\)](#)

3 – Designing data platforms for governance and compliance

How do you protect data in your organization's analytics workload? Privacy by Design (PbD) is an approach in system engineering that takes privacy into account throughout the whole engineering process. PbD especially focuses on systems or applications that capture and process personal data. Many countries or political unions enforce data protection regulations. The main data protection regulations are: GDPR (General Data Protection Regulation), CCPA (California Consumer Privacy), LGPD (Lei geral da Protecao de Dados Pessooasis in Brazil), POPIA (South Africa), Australian Privacy Act and DPA (UK Data Protection Act).

As an organization you must have an understanding what data protection regulations you must adhere to and implement them into your solution accordingly. If your organization operates across territories, then you must adhere to multiple data regulations.

This whitepaper covers the common themes shared amongst these regulations; however this is not an exhaustive list. Therefore you must consult your organization's Data Protection Office to determine what additional regional and company-wide data protection and data governance requirements must be implemented.

For more details regarding the different types of data protection regulations, refer to the following:

- GDPR - [General Data Protection Regulation Center](#)
- CCPA - [California Consumer Privacy Act](#)
- LGPD - [The General Data Protection Law](#)
- POPIA - [South Africa Data Privacy](#)

ID	Priority	Best practice
<input type="checkbox"/> BP 3.1	Required	Privacy by design.
<input type="checkbox"/> BP 3.2	Required	Classify and protect data
<input type="checkbox"/> BP 3.3	Required	Understand data classifications and their protection policies.
<input type="checkbox"/> BP 3.4	Required	Identify the source data owners and have them set the data classifications.
<input type="checkbox"/> BP 3.5	Required	Record data classifications into the Data Catalog so that analytics workload can understand.
<input type="checkbox"/> BP 3.6	Required	Implement encryption policies.
<input type="checkbox"/> BP 3.7	Required	Implement data retention policies for each class of data in the analytics workload.
<input type="checkbox"/> BP 3.8	Recommended	Enforce downstream systems to honor the data classifications.

For more details, refer to the following information:

- AWS GDPR Center: [Introducing the New GDPR Center and “Navigating GDPR Compliance on AWS” Whitepaper](#)
- AWS Database Blog: [Best practices for securing sensitive data in AWS data stores](#)
- AWS Security Blog: [Discover sensitive data by using custom data identifiers with Amazon Macie](#)
- Amazon Macie User Guide: [What is Amazon Macie?](#)
- AWS Key Management Service Developer Guide: [What is AWS Key Management Service?](#)
- AWS Whitepaper: [Data Classification: Secure Cloud Adoption](#)

Best practice 3.1 – Privacy by Design

Privacy by Design is an approach in system engineering that takes privacy into account throughout the whole engineering process. It especially focuses on systems or applications that capture and process personal data.

There is an increased focus on ensuring that personal data is processed lawfully, fairly, and in a transparent manner in relation to the data subject. Another concern is that the data processing is adequate, relevant, and limited in relation to the purpose for which the information is used.

Suggestion 3.1.1 – Data minimization

Organizations should only receive, process, and store information that is relevant for the task rather than processing all information when only a portion of the file is required. For example, if a client provided a full extract of all information from their source system containing sensitive personal information, and if a portion of the file is deemed irrelevant in meeting the overall project requirements, the remainder of the file should not be stored or processed.

Data minimization coincides with data access controls in that applying data minimization rules can be implemented using data access controls. A suggestion is to create and maintain a data access matrix aligned with your data classification catalogs. This helps ensure that the correct groups of people have access to the right data. As most compliant frameworks encourage evidence that rules have been applied, a data access matrix can demonstrate to auditors that your organization has gone through the proper thought process to determine who can access what information.

Data minimization can be applied at the point of capture. It can also be applied at the point of access by presenting a restricted data model or implementing role-based access controls (RBAC). For more information on controlling data access, see [4 – Implement data access control \(p. 17\)](#).

Test and user acceptance test (UAT) environments, as well as training model datasets, must have a restricted dataset and not contain any personal information. If the structure of the data model must remain the same as production, then consider anonymizing or masking information to meet your data minimization requirements.

It is common practice to create test and development environments using a backup of production and restore to the respective development or test environment. If this is the case, anonymization of personally identifiable information (PII) and other sensitive information must occur using inbuilt logic or services such as AWS Glue DataBrew to obfuscate the information.

For more details, refer to the following documentation:

- Amazon Redshift RBAC - [Amazon Redshift role-based access controls](#)
- AWS Lake Formation RBAC - [Lake Formation role-based access controls](#)
- Amazon Athena RBAC - [Amazon Athena fine-grained access controls](#)
- AWS Glue DataBrew - [AWS Glue DataBrew](#) Visual Data Preparation

Suggestion 3.1.2 – Anonymization, pseudonymization, and tokenization

Suggestion 3.1.2.1 – Anonymization

Anonymization is defined as the process of turning data into a form that does not identify individuals and where identification is not likely to take place.

This results in changing personal data into data that is no longer personal. An important factor in this process is that the anonymization must be irreversible. The anonymized value should be supported by the current field data type, have similar length, and retain some characteristics of the original value. For example, if a Vehicle Registration Number such as OU51 SMR was being anonymized, the result would look similar to BB88 9AA.

Organizations need the ability to anonymize full datasets as well as single records. Single record anonymization functionality can help deliver right to erasure and meet data retention requirements. In this case, full batch anonymization is typically used when obfuscating development and UAT environments.

The function to anonymize information should support the flexibility to anonymize certain fields, but not all.

Operational databases, reporting databases, and analytical data marts should all be considered for anonymization, although reports and analytical cubes should never typically contain PII information regardless.

Audit the reason why information was anonymized, for example, data portability, or data retention removal. The time, date, and user ID of when and who the anonymization process has affected should be recorded in an audit table.

For more details, refer to the following information:

- AWS Big Data Blog: [Anonymize and manage data in your data lake with Amazon Athena and AWS Lake Formation](#)

Suggestion 3.1.2.2 – Pseudonymization

Pseudonymized data is not the same as anonymized data.

When data has been pseudonymized, it still retains a level of detail in the target data that allows tracking back of the data to its original state. With anonymized data, the level of detail is reduced rendering a reverse compilation impossible. Pseudonymization is the processing of personal data in such a way that the data can only be attributed to a specific data subject by using additional information. To pseudonymize a dataset, the additional information must be kept separately and subject to technical and organizational measures to ensure non-attribution to an identified or identifiable person.

In summary, pseudonymized data is a privacy-enhancing technique where directly identifying data, such as IP addresses and contact information, are held separately and securely from processed data to ensure non-attribution. Similar to anonymization, referential integrity must not be affected. Therefore, both of the following are required: an audit trail of the pseudonymization process, and a pseudonymization function that supports both single item and batch processing.

Suggestion 3.1.2.3 – Tokenization

Tokenization, when applied to data security, is the process of substituting a sensitive data element with a non-sensitive equivalent. This is referred to as a token, which has no extrinsic or exploitable meaning or value. The token is a reference that maps back to the sensitive data through a tokenization system. Tokenization is typically used in finance to tokenize the payment account number (PAN).

For more details, refer to the following information:

- AWS Blog – [AWS Glue DataBrew detection data masking transformations](#)

Suggestion 3.1.3 – Rights of the individual, citizen, or subject

Your organization should consider the process to address the rights of the individual, citizen, or subject for their respective regional regulation.

Suggestion 3.1.3.1 – Subject Access Request (SAR)

This particular right is for an individual to request information from the data controller, that is, how their personal data is being processed. If an individual's information is being processed, the personal data and associated metadata must be provided to that individual.

If the individual's information is stored in a database, then an automated process, such as a stored procedure or User-Defined Function (UDF), should be developed to answer the Subject Access Request (SAR). There will, however, be situations when the individual's information is stored in Amazon S3. If the information is stored in Amazon S3, the proposed solution to identify which S3 object contains the respective information is to build a lookup table in a database containing the reference number, individual contact details, and the S3 object location. This approach allows your organization to ingest the information into Amazon EMR, infer the schema using Apache Spark, and extract the information required to fulfill the request. Alternatively, your organization must process all S3 objects to identify the information to fulfill the request.

If your regional regulations require that your organization handle a right to data portability request, then the SAR logic can double up to support that as well.

For more details, refer to the following information:

- Apache Spark Documentation – [Inferring the Schema Using Reflection](#)

Suggestion 3.1.3.2 – Right to be forgotten or erasure

Individuals have the right to erasure (the right to be forgotten), where an individual can request that all of their personal data is erased by the data controller organization. In some countries, there are instances where the data controller can refuse to comply with a right to erasure request, such as where the data is used for financial governance.

The right to erasure does not strictly mean that the individual's information must be deleted. Instead, it can be permanently masked so that the personal data is no longer in the clear and the update is irreversible.

The organization must consider all data repositories when responding to a SAR as an individual's information can reside in back up and source system databases. All these records must have the individual's information removed or anonymized.

If there are concerns about the impact of database referential integrity being affected by removing the individual's information, then you can consider anonymization of the specific data attributes for the given individual. There are benefits to anonymization, such as being able to maintain an audit history of what actions have been performed against the individual by referencing a system ID. The same steps that are performed in production environments must also be run in UAT, development, OLTP, and back up repositories.

The schedule of running the procedure in the other environments depends on the refresh schedules of those other environments.

Best practice 3.2 – Classify and protect data

How do you classify and protect data in analytics workload? Because analytics workloads ingest data from source systems, the owner of the source data should define the data classifications. As the analytics workload owner, you should honor the source data classifications and implement the corresponding data protection policies of your organization. Share the data classifications with the downstream data consumers to permit them to honor the data classifications in their organizations and policies as well.

Data classification helps to categorize organizational data based on sensitivity and criticality, which then helps determine appropriate protection and retention controls on that data.

Best practice 3.3 – Understand data classifications and their protection policies

Data classification in your organization is key to determining how data must be protected while at rest and in transit. For example, since an analytics workload necessarily copies and shares data between operations and systems, we recommend that access be controlled to certain data classifications. Such a data protection strategy helps to prevent data loss, theft, and corruption, and helps to minimize the impact caused by malicious activities or unintended access.

Suggestion 3.3.1 – Identify classification levels

Use the [Data Classification whitepaper](#) to help you identify different classification levels. Four common levels used are restricted, confidential, internal, and public, however, these levels can vary based on the industry and compliance requirements of your organization.

Suggestion 3.3.2 – Define access rules

The data owners should define the data access rules based on the sensitivity and criticality of the data. For example, with AWS Lake Formation, you can define and enforce access controls that operate at the table, column, row, and cell level for all the users that access your data lake.

For more details, refer to the following information:

- AWS Security Blog: [How to scale your authorization needs by using attribute-based access control with S3.](#)
- AWS Big Data Blog: [Create a secure data lake by masking, encrypting data, and enabling fine-grained access with AWS Lake Formation.](#)
- AWS Big Data Blog: [Control data access and permissions with AWS Lake Formation and Amazon EMR.](#)
- AWS Big Data Blog: [Enforce column-level authorization with Amazon QuickSight and AWS Lake Formation.](#)

Suggestion 3.3.3 – Identify security zone models to isolate data based on classification

Design the security zone models from AWS account levels down to AWS resource levels. For example, consider building AWS multi-account models to isolate different classes of data from AWS account level. Or, you can consider separating out development and test resources from production ones from AWS account level or from resource levels.

For more details, refer to the following information:

- AWS Whitepaper: [An Overview of the AWS Cloud Adoption Framework.](#)
- AWS Whitepaper: [Organizing Your AWS Environment Using Multiple Accounts.](#)
- AWS Whitepaper: [Security Pillar – AWS Well-Architected Framework.](#)

Suggestion 3.3.4 – Identify sensitive information and define protection policies

Discover sensitive data by using custom data identifiers in Amazon Macie. Based on the sensitivity and criticality of the data, implement data protection policies to prevent unauthorized access. Due to compliance requirements, data might be masked or deleted after processing in some cases.

For more details, refer to the following information:

- AWS Blog: [Introducing PII data identification and handling using AWS Glue DataBrew](#)
- AWS Blog: [Create a secure data lake by masking, encrypting data, and enabling fine-grained access with AWS Lake Formation](#)

Best practice 3.4 – Identify the source data owners and have them set the data classifications

Identify the owners of the source data, such as data lake administrators, business data owners, or IT storage administrators. Confirm what level of protection is required for the data within the analytics workload.

Data classifications follow the data as it moves throughout the analytics workflow to ensure that the data is protected, and to determine who and what systems are allowed to access the data. By following the organization's classification policies, the analytics workload should be able to differentiate the data protection implementations for each class of data. Because each organization has different kinds of classification, the analytics workload should provide a strong logical boundary between processing data of different sensitivity levels. These classifications include *restricted*, *confidential*, and *sensitive*.

Suggestion 3.4.1 – Assign owners per each dataset

A dataset, or a table in relational database technology, is a collection of data. A Data Catalog is a collection of metadata that helps centralize share, search, and manage permissions. The administrator of

the analytics workload should know who are the owners for each dataset, and should assign the dataset ownership in the Data Catalog.

Suggestion 3.4.2 – Define attestation scope and reviewer as additional scope for sensitive data

As the owner of the analytics workload, you should know the data owner for each dataset. For example, when a dataset classified as highly sensitive has permission issues within the organization, you might have to talk to the dataset owners and have them resolve the issues.

Suggestion 3.4.3 – Set expiry for data ownership and attestation, and have owners reconfirm periodically

As businesses change, the data owners and the data classifications might change as well. Run campaigns periodically, such as quarterly or yearly, to request each of the dataset owners to reconfirm that they are still the right owners, and that the data classifications are still accurate.

Best practice 3.5 – Record data classifications into the Data Catalog so that analytics workloads can understand

Allow processes to update the Data Catalog so it can provide a reliable record of where the data is located and its precise classification. To protect the data effectively, analytics systems should know the classifications of the source data so that the systems can govern the data according to business needs. For example, if the business requires that confidential data be encrypted using team-owned private keys, such as from AWS Key Management Service (AWS KMS), then the analytics workload should be able to determine which data is classified as confidential by referencing its data catalog.

Suggestion 3.5.1 – Use tags to indicate the data classifications

Use tags to designate the classification of sensitive data in data stores. Tags allow discoverability of data sensitivity without directly exposing the underlying data. They also can be used to authorize access in [attribute-based access control \(ABAC\)](#) schemes.

For more details, refer to the following information:

- AWS Lake Formation Developer Guide: [What Is AWS Lake Formation?](#)
- AWS Whitepaper: [Tagging Best Practices](#)

Suggestion 3.5.2 – Record lineage of data to track changes in the Data Catalog

Data lineage is a relation among data and the processing systems. For example, the data lineage tells where the source system of the data has come from, what changes occurred to the data, and which downstream systems have access to it. Your organization should be able to discover, record, and visualize the data lineage from source to target systems.

For more details, refer to the following information:

- AWS Big Data Blog: [Metadata classification, lineage, and discovery using Apache Atlas on Amazon EMR](#)

Best practice 3.6 – Implement encryption policies

Data encryption is a way of translating data from plaintext (unencrypted) to ciphertext (encrypted). Encryption is a critical component of a *defense in depth* strategy. Therefore, it is highly recommended that your organization implement a well-designed encryption and key management system by separating access to the decryption key from access to your data to provide data protection.

Suggestion 3.6.1 – Implement encryption policies for data at rest and in transit

Each analytics solution provides encryption methods. Review the viable encryption methods of your solutions and implement as necessary.

For more details, refer to the following information:

- AWS Key Management Service Best Practices Whitepaper: [AWS managed and customer-managed CMKs](#)
- AWS Big Data Blog: [Best Practices for Securing Amazon EMR](#)
- AWS Big Data Blog: [Encrypt Your Amazon Redshift Loads with Amazon S3 and AWS KMS](#)
- AWS Big Data Blog: [Encrypt and Decrypt Amazon Kinesis Records Using AWS KMS](#)
- AWS Partner Network (APN) Blog: [Data Tokenization with Amazon Redshift and Protegrity](#)

Best practice 3.7 – Implement data retention policies for each class of data in the analytics workload

The business's data classification policies determine how long the analytics workload should retain the data and how long backups should be kept. These policies help ensure that every system follows the data security rules and compliance requirements. The analytics workload should implement data retention and backup policies according to these data classification policies. For example, if the policy requires every system to retain the operational data for five years, the analytics systems should implement rules to keep the in-scoped data for five years. More information on data retention can be found in [Sustainability](#) (p. 46).

Suggestion 3.7.1 – Create backup requirements and policies based on data classifications

Data backup should be based on business requirements, such as recovery point objective (RPO), recovery time objective (RTO), data classifications, and the compliance and audit requirements.

Suggestion 3.7.2 – Create data retention requirement policies based on the data classifications

Avoid creating blanket retention policies. Instead, policies should be tailored to individual data assets based on their retention requirements.

For more details, refer to the following information:

- AWS Big Data Blog: [Building a cost efficient, petabyte-scale lake house with Amazon S3 Lifecycle rules and Amazon Redshift Spectrum: Part 1](#)
- AWS Big Data Blog: [Retaining data streams up to one year with Amazon Kinesis Data Streams](#)
- AWS Big Data Blog: [Retain more for less with UltraWarm for Amazon OpenSearch Service](#)

Suggestion 3.7.3 – Create data version requirements and policies

Implement a process that captures the data version to address, based on compliance, security, and operational requirements.

For more details, refer to the following information:

- AWS Storage Blog: [Reduce storage costs with fewer noncurrent versions using Amazon S3 Lifecycle](#)
- AWS Storage Blog: [Simplify your data lifecycle by using object tags with Amazon S3 Lifecycle](#)
- AWS Database Blog: [Implementing version control using Amazon DynamoDB](#)

Best practice 3.8 – Enforce downstream systems to honor the data classifications

Since other data-consuming systems will access the data that the analytics workload shares, the workload should require the downstream systems to implement the required data classification policies. For example, if the analytics workload shares the data that is required to be encrypted using customer managed private keys in AWS Key Management Service (AWS KMS), then the downstream systems should also acknowledge and implement such a data protection policy.

This helps to ensure that the data is protected throughout the data pipelines.

Suggestion 3.8.1 – Have a centralized, shareable catalog with cross-account access to ensure that data owners manage permissions for downstream systems

Downstream systems can run on independent AWS accounts, different from the AWS account running the majority of the analytics workload. Downstream systems should be able to discover the data, acknowledge the required data protection policies, and enforce those policies across the analytics platform.

To allow the downstream systems to use the data from analytics workload, the analytics workload should provide cross-account access based on least privileges for each dataset.

For more details, refer to the following information:

- AWS Big Data Blog: [Cross-account AWS Glue Data Catalog access with Amazon Athena](#)
- AWS Big Data Blog: [How JPMorgan Chase built a data mesh architecture to drive significant value to enhance their enterprise data platform](#)

Suggestion 3.8.2 – Monitor the downstream systems' eligibility to access classified data from the analytics workload

Monitor the downstream systems' eligibility to handle sensitive data. For example, you do not want development or test Amazon Redshift clusters to read sensitive data from the analytics workload. If your organization runs a program that certifies which systems are eligible to process various classes of data, periodically verify that each downstream system's data processing eligibility levels are correct and the list of data that it accesses are appropriate.

4 – Implement data access control

How do you manage access to data within your organization's source, analytics, and downstream systems?

An analytics workload is a centralized repository of data from different source systems. As the analytics workload owner, you should honor the source systems' access management policies when connecting to, and ingesting from, the source systems.

ID	Priority	Best practice
<input type="checkbox"/> BP 4.1	Required	Allow data owners to determine which people or systems can access data in analytics and downstream workloads.
<input type="checkbox"/> BP 4.2	Required	Build user identity solutions that uniquely identify people and systems.
<input type="checkbox"/> BP 4.3	Required	Implement the required data authorization models.

ID	Priority	Best practice
<input type="checkbox"/> BP 4.4	Recommended	Establish an emergency access process to ensure that admin access is managed and used when required.
<input type="checkbox"/> BP 4.5	Recommended	Track data and database changes.

For more details, refer to the following documentation:

- AWS Lake Formation Developer Guide: [Lake Formation Access Control Overview](#)
- Amazon Athena User Guide: [AWS Identity and Access Management in Amazon Athena](#)
- Amazon Athena User Guide: [Enabling Federated Access to the Amazon Athena API](#)
- Amazon Redshift Database Developer Guide: [Managing database security](#)
- Amazon EMR Management Guide: [AWS Identity and Access Management for Amazon EMR](#)
- Amazon EMR Management Guide: [Use Kerberos authentication](#)
- Amazon EMR Management Guide: [Use an Amazon EC2 key pair for SSH credentials](#)

Best practice 4.1 – Allow data owners to determine which people or systems can access data in analytics and downstream workloads

Data owners are the people that have direct responsibility for data protection. For instance, the data owners want to determine which data is publicly accessible, or which data is restricted access to whom or what systems. The data owners should be able to provide data access rules, so that the analytics workload can implement the rules.

Suggestion 4.1.1 – Identify data owners and assign roles

Data ownership is the management and oversight of an organization's data assets to help provide business users with high-quality data that is easily accessible in a consistent manner. Because the analytics workload consolidates multiple datasets into a central place, each dataset is owned by different teams or people. So, it is important for the analytics workload to identify which dataset is owned by whom to have the owners control the data access permissions.

Suggestion 4.1.2 – Identify permission using a permission matrix for users and roles based on actions performed on the data by users and downstream systems

To aid in identifying and communicating data-access permissions, an Access Control Matrix is a helpful method to document which users, roles, or systems have access to which datasets, and to describe what actions they can perform. Below is a sample matrix for two users, and two roles for two schemas with a table in them:

Table 1: Example Access Control Matrix for Users and Roles

Permissions	Read	Write
Schema 1	User1, User2, Role1, Role2	Role1
Schema 1 / Table 1	User1, User2, Role1, Role2	Role2
Schema 2	User1, User2, Role1, Role2	User1, Role1
Schema 2 / Table 2v	User1, User2, Role1, Role2	User2, Role2

The matrix format can help identify the least permissions that are required by various resources and to avoid overlaps. An Access Control Matrix should be thought of as an abstract model of permissions at a given point in time. Periodically review the actual access permissions against the permission matrix document to ensure accuracy.

Best practice 4.2 – Build user identity solutions that uniquely identify people and systems

To control data access effectively, the analytics workload should be able to uniquely identify the people or systems. For example, the workload should be able to tell who accessed to the data by looking at the user identifiers (such as user names, tags, or IAM role names) with confidence that the identifier represents only one person or system.

For more details, refer to the following information:

- AWS Big Data Blog: [Amazon Redshift identity federation with multi-factor authentication](#)
- AWS Big Data Blog: [Federating single sign-on access to your Amazon Redshift cluster with PingIdentity](#)
- AWS Database Blog: [Get started with Amazon OpenSearch Service: Use Amazon Cognito for Kibana access control](#)
- AWS Partner Network (APN) Blog: [Implementing SAML AuthN for Amazon EMR Using Okta and Column-Level AuthZ with AWS Lake Formation](#)
- AWS CloudTrail User Guide: [How AWS CloudTrail works with IAM](#)

Suggestion 4.2.1 – Centralize workforce identities

It's a best practice to centralize your workforce identities, which allows you to federate with AWS Identity and Access Management (IAM) using AWS IAM Identity Center (successor to AWS Single Sign-On) or another federation provider. In Amazon Redshift, IAM roles can be mapped to Amazon Redshift database groups. In Amazon EMR, IAM roles can be mapped to an Amazon EMR security configuration or an Apache Ranger Microsoft Active Directory group-based policy. In AWS Glue, IAM roles can be mapped to AWS Glue Data Catalog resource policies.

AWS analytics services – such as Amazon OpenSearch Service and Amazon DynamoDB – allow integration with Amazon Cognito for authentication. Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Apple, Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0 and OpenID Connect.

For more details, refer to the following information:

- AWS Big Data Blog: [Federate Database User Authentication Easily with IAM and Amazon Redshift](#)
- AWS Big Data Blog: [Federating single sign-on access to your Amazon Redshift cluster with PingIdentity](#)
- Amazon EMR Management Guide: [Allow AWS IAM Identity Center \(successor to AWS Single Sign-On\) for Amazon EMR Studio](#)

Best practice 4.3 – Implement the required data access authorization models

User authorization determines what actions that a user is permitted to take on the data or resource. The data owners should be able to use the authorization methods to protect their data as needed. For example, if the data owners must control which users are allowed to view certain columns of data, the analytics workload should provide column-wise data access authorization along with user group management for an effective control.

Suggestion 4.3.1 – Implement IAM policy-based data access controls

Limit access to sensitive data stores with IAM policies where possible. Provide systems and people with rotating short-term credentials via role-based access control (RBAC).

For more details, refer to the following information:

- AWS Big Data Blog: [Restrict access to your AWS Glue Data Catalog with resource-level IAM permissions and resource-based policies.](#)

Suggestion 4.3.2 – Implement dataset-level data access controls

As dataset owners require independent rules of granting data access, you should build the analytics workloads to have the dataset owners control the data access per each dataset level. For example, if the analytics workload hosts a shared Amazon Redshift cluster, the owners of the individual table should be able to authorize the table read and write independently.

For more details, refer to the following information:

- AWS Big Data Blog: [Validate, evolve, and control schemas in Amazon MSK and Amazon Kinesis Data Streams with AWS Glue Schema Registry.](#)
- Amazon Redshift: [Amazon Redshift announces support for Row-Level Security \(RLS\) Streams with AWS Glue Schema Registry.](#)

Suggestion 4.3.3 – Implement column-level data access controls

Care should be taken that end users of analytics applications are not exposed to sensitive data. Downstream consumers of data should only access the limited view of data necessary for that analytics purpose. Enforce that sensitive data is not exposed using column-level restrictions, for example, mask the sensitive columns to downstream systems so an accidental exposure is avoided.

For more details, refer to the following information:

- AWS Big Data Blog: [Allow fine-grained permissions for Amazon QuickSight authors in AWS Lake Formation](#)
- AWS Partner Network (APN) Blog: [Implementing SAML AuthN for Amazon EMR Using Okta and Column-Level AuthZ with AWS Lake Formation](#)
- AWS Big Data Blog: [Implementing Authorization and Auditing using Apache Ranger on Amazon EMR](#)

Best practice 4.4 – Establish an emergency access process to ensure that admin access is managed and used when required

Emergency access allows expedited access to your workload in the unlikely event of an automated process or pipeline issue. This will help you rely on least privilege access, but still provide users the right level of access when they require it.

Suggestion 4.4.1 – Ensure that risk analysis is performed on your analytics workload by identifying emergency situations and a procedure to allow emergency access

Identify the potential events that can happen from source systems, analytics workload, and downstream systems. Quantify the risk of each event such as likelihood (low, medium, or high) and the size of the business impact (small, medium, or large).

For example, after you identified priority risks, discuss with the source and downstream system owners on how to allow analytics workload access to the source and downstream systems to continue the data processing business.

Best practice 4.5 – Track data and database changes

Data auditing involves monitoring a database to track the actions of a user or process, and to audit the changes that have occurred to the data.

Suggestion 4.5.1 – Database triggering for data auditing

A database trigger is procedural code that is automatically run in response to certain events on a particular table or view in a database. Database triggers can then be used to update an audit table with the changes that have occurred. The types of information that should be included in the auditing process include: the original and updated value of what has been updated, the process or stored procedure that made the update, and the time and date the update occurred.

Suggestion 4.5.2 – Enable advanced auditing

If your database engine supports auditing as a native feature, you should enable the feature to record and audit database events such as connections, disconnections, tables queried, or types of queries issued.

Suggestion 4.5.3 – AWS Lake Formation time travel queries

AWS Lake Formation governed tables support the use of time travel queries. Using time travel queries, your organization can query a table of a previous time. A good example is: if you need to know the status of an individual at a certain time, then a time travel query allows you to select a date range to return the value that existed at that time, rather than the current value.

Suggestion 4.5.4 – Change Data Capture (CDC)

CDC records INSERTs, UPDATEs, and DELETEs applied to relational database tables, and makes a log available of which relational database objects changed, where, and when. These change tables contain columns that reflect the column structure of the source table you have chosen to track, along with the metadata required to understand the changes that have been made.

For more details, refer to the following information:

- AWS CloudTrail - [Secure Standardized Logging](#)
- Amazon RDS Aurora - [Advanced Auditing with an Amazon Aurora](#)
- Amazon RDS Aurora - [Configuring an audit log to capture database activities for Amazon RDS](#)
- AWS Lake Formation - [Implement time travel with Governed Tables](#)
- AWS Database Migration Service (AWS DMS) [AWS Database Migration Service](#)

5 – Control the access to workload infrastructure

How do you protect the infrastructure of the analytics workload? Analytics environments change based on the evolving requirements of data processing and data distribution. Ensuring the environment is accessible with the least permissions necessary is essential in delivering a secure platform. Automate the auditing of environment changes and generate alerts in case of abnormal environment access.

ID	Priority	Best practice
<input type="checkbox"/> BP 5.1	Required	Prevent unintended access to the infrastructure.

ID	Priority	Best practice
<input type="checkbox"/> BP 5.2	Required	Implement least privilege policies for source and downstream systems.
<input type="checkbox"/> BP 5.3	Required	Monitor the infrastructure changes and the user activities against the infrastructure.
<input type="checkbox"/> BP 5.4	Required	Secure the audit logs that record every data or resource access in analytics infrastructure.

Best practice 5.1 – Prevent unintended access to the infrastructure

Grant least privilege access to infrastructure to help prevent inadvertent or unintended access to the infrastructure. For example, make sure that anonymous users are not allowed to access to the systems, and that the systems are deployed into isolated network spaces. Network boundaries isolate analytics resources and restrict network access. Network access control lists (NACLs) act as a firewall for controlling traffic in and out. To reduce the risk of inadvertent access, define the network boundaries of the analytics systems and only allow intended access.

Suggestion 5.1.1 – Ensure that resources in the infrastructure have boundaries

Use infrastructure boundaries for services such as databases. Place services in their own VPC private subnets that are configured to allow connections only to needed analytics systems.

Use [AWS Identity and Access Management \(IAM\) Access Analyzer](#) for all AWS accounts that are centrally managed through [AWS Organizations](#). This allows security teams and administrators to uncover unintended access to resources from outside their AWS organization within minutes.

You can proactively address whether any resource policies across any of your accounts violate your security and governance practices by allowing unintended access.

Best practice 5.2 – Implement least privilege policies for source and downstream systems

The principle of least privilege works by giving only enough access for systems to do the job. Set an expiry on temporary permissions to ensure that re-authentication occurs periodically. The system actions on the data should determine the permission and granting permissions to other systems should not be permitted.

Suggestion 5.2.1 – Ensure that permissions are least for the action performed by user/system

Identify the minimum privileges that each user or system requires, and only allow the permissions that they need. For example, if a downstream system requests to read an Amazon Redshift table from an analytics workload, only give the read permission for the table using Amazon Redshift user privilege controls.

For more details, refer to the following information:

- AWS Security Blog: [Techniques for writing least privilege IAM policies](#)
- Amazon Redshift Database Developer Guide: [Managing database security](#)
- AWS Security Blog: [IAM Access Analyzer makes it easier to implement least privilege permissions by generating IAM policies based on access activity](#)

Suggestion 5.2.2 – Implement the two-person rule to prevent accidental or malicious actions

Even if you have implemented the least privilege policies, someone must have critical permissions for the business, such as the ability to delete datasets from analytics workloads.

The two-person rule is a safety mechanism that requires the presence of two authorized personnel to perform tasks that are considered important. It has its origins in military protocol, but the IT security space has also widely adopted the practice.

By implementing the two-person rule, you can have additional prevention of accidental or malicious actions of the people who have critical permissions.

Best practice 5.3 – Monitor the infrastructure changes and the user activities against the infrastructure

As the infrastructure changes over time, you should monitor what has been changed by whom. This is to ensure that such changes are deliberate and the infrastructure is still protected.

Suggestion 5.3.1 – Monitor the infrastructure changes

You want to know every infrastructure change and want to know that such changes are deliberate. Monitor the infrastructure changes using available methods on your team. For example, you can implement an operation procedure to review the infrastructure configurations every quarter of the year. Or, you can use AWS services that assist you to monitor the infrastructure changes with less effort.

For more details, refer to the following documentation:

- AWS Config Developer Guide: [What Is AWS Config?](#)
- Amazon Inspector User Guide: [What is Amazon Inspector?](#)
- Amazon GuardDuty User Guide: [Amazon S3 protection in Amazon GuardDuty](#)

Suggestion 5.3.2 – Monitor the user activities against the infrastructure

You want to know who is changing the infrastructure and when, so that you can see that any given infrastructure change is performed by an authorized person or system. To do so, as examples, you can implement an operation procedure to review the AWS CloudTrail audit logs every quarter of the year. Or you can implement near real time trend analysis using AWS services such as Amazon CloudWatch Logs Insights.

For more details, refer to the following information:

- AWS CloudTrail User Guide: [Monitoring CloudTrail Log Files with Amazon CloudWatch Logs](#)
- AWS Management and Governance Blog: [Analyzing AWS CloudTrail in Amazon CloudWatch](#)

Best practice 5.4 – Secure the audit logs that record every data or resource access in analytics infrastructure

Audit logs are records that record events as they happened and should be immutable. These logs provide proof of actions and help in identifying misuse. The logs provide a baseline for analysis or for an audit when initiating an investigation. By using a fault-tolerant storage for these logs, it is possible to recover them even when there is a failure in the auditing systems. Access permissions to these logs must be restricted to privileged users. Also log audit log access to help in identifying unintended access to audit data.

Suggestion 5.4.1 – Ensure that auditing is active in analytics services and are delivered to fault-tolerant persistent storage

Review the available audit log features of your analytics solutions, and configure the solutions to store the audit logs to fault-tolerant persistent storage. This helps ensure that you have complete audit logs for security and compliance purposes.

For more details, refer to the following information:

- AWS Management and Governance Blog: [AWS CloudTrail Best Practices](#)
- Amazon Redshift Cluster Management Guide: [Database audit logging](#)
- Amazon OpenSearch Service (successor to Amazon OpenSearch Service) Developer Guide: [Monitoring audit logs in Amazon OpenSearch Service](#)
- AWS Technical Guide – Build a Secure Enterprise Machine Learning Platform on AWS: [Audit trail management](#)
- AWS Big Data Blog: [Build, secure, and manage data lakes with AWS Lake Formation](#)

Reliability

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This includes the ability to operate and test the workload through its total lifecycle. This whitepaper provides in-depth, best practice guidance for implementing reliable analytics workloads on AWS.

Best practices

- [6 – Design resilience for analytics workload \(p. 24\)](#)
- [7 – Govern data and metadata changes \(p. 27\)](#)

6 – Design resilience for analytics workload

How do you design analytics workloads to withstand and mitigate failures?

ID	Priority	Best practice
<input type="checkbox"/> BP 6.1	Required	Create an illustration of data flow dependencies.
<input type="checkbox"/> BP 6.2	Required	Understand the business requirements of analytics and ETL jobs.
<input type="checkbox"/> BP 6.3	Required	Monitor analytics systems to detect analytics or ETL job failures.
<input type="checkbox"/> BP 6.4	Required	Notify stakeholders about analytics or ETL job failures.
<input type="checkbox"/> BP 6.5	Recommended	Automate the recovery of analytics and ETL job failures.
<input type="checkbox"/> BP 6.6	Recommended	Build a disaster recovery (DR) plan for the analytics infrastructure and the data.

For more details, refer to the following documentation:

- AWS Glue Developer Guide: [Running and Monitoring AWS Glue](#)
- AWS Glue Developer Guide: [Monitoring with Amazon CloudWatch](#)

- AWS Glue Developer Guide: [Monitoring AWS Glue Using Amazon CloudWatch Metrics](#)
- AWS Prescriptive Guidance – Patterns: [Orchestrate an ETL pipeline with validation, transformation, and partitioning using AWS Step Functions](#)
- AWS Support Knowledge Center: [How can I use a Lambda function to receive SNS alerts when an AWS Glue job fails a retry?](#)
- AWS Glue Developer Guide: [Repairing and Resuming a Workflow Run](#)
- AWS Data Pipeline Developer Guide: [Cascading Failures and Reruns](#)

Best practice 6.1 – Create an illustration of data flow dependencies

Work with business stakeholders to create a visual illustration of the data pipeline. Identify the systems that interact with each dependency. The key architecture components that are expected to be captured are data acquisition, ingestion, data transformation, data processing, data storage, and data consumption. All system dependencies need owners. Agree within your organization who owns which dependency.

Best practice 6.2 – Understand the business requirements of analytics and ETL jobs

There are three common design patterns when moving data from source systems to a target data store. Source and target data stores include: data warehouse, data lake, object storage, file storage, database – relational or non-relational, among others. The first pattern is ETL, which transforms the data before it is loaded into the target data source. The second pattern is ELT, which loads the data into the central data repository (like data lake or data warehouse) and uses the familiar SQL semantics and power of the massively parallel processing (MPP) framework to perform the transformations. The third pattern ETLT is a hybrid of both. This performs a transform to meet *entry* quality criteria, loads into the data storage, and later transforms when needed.

Best practice 6.3 – Monitor analytics systems to detect analytics or ETL job failures

Detect job failures as soon as possible. Pinpointing where and how the error occurred is critical for notifications and corrective actions.

Suggestion 6.3.1 – Monitor and track job errors from different levels, including infrastructure, ETL workflow, and ETL application code

Failures can occur at all levels of the analytics system. Each task in the analytics workload should be instrumented to provide metrics indicating the health of the task. Monitor the emitted metrics and alarm if any components fail.

Suggestion 6.3.2 – Establish end-to-end monitoring for the complete analytics and ETL pipeline

End-to-end monitoring allows tracking the flow of data as it passes through the analytics system. In many cases, data processing might be dependent on application logic, such as sampling a subset of data from a data stream to check accuracy. Properly identifying and monitoring the end-to-end flow of data allows detecting at which step the analytics and ETL job fails.

Suggestions 6.3.3 – Determine what data was processed when the job failed

Failures in data processing systems can cause data integrity or data quality issues. Determine what data was being processed at the time of failure and perform quality checks of both the input and output data.

Suggestions 6.3.4 – Classify the severity of the job failures based on the type of failure and the business impact

Classifying the severity of different job failures will help you prioritize remediation and guide the notification requirements to key stakeholders.

Best practice 6.4 – Notify stakeholders about analytics or ETL job failures

Analytics and ETL job failures can impact the SLAs for delivering the data on time for downstream analytics workloads. Failures might cause data quality or data integrity issues as well. Notifying all stakeholders about the job failure as soon as possible is important for remediation actions needed. Stakeholders may include IT operations, help desk, data sources, analytics, and downstream workloads.

For more details, refer to the following information:

- AWS Well-Architected: [Design your Workload to Withstand Component Failures](#)

Suggestions 6.4.1 – Establish automated notifications to predefined recipients

Use services such as Amazon Simple Notification Service (Amazon SNS) to send automated emails, SMS alerts, or both in the event of failure. Store the alert logs in an immutable data store for future reference.

Suggestions 6.4.2 – Do not include sensitive data in notifications

Automated alerts often include indicators of useful information for troubleshooting the failure. Ensure PII and sensitive information, such as medical or financial information, is not shared in failure notifications.

Suggestions 6.4.3 – Integrate the analytics job failure notification solution with the enterprise operation management system

Where possible, integrate automated notifications into existing operations management tools. For example, an operations support ticket can be automatically filed in the event of a failure. That same ticket can automatically be resolved if the analytics system recovers on retry.

Suggestions 6.4.4 – Notify IT operations and help desk teams of any ETL job failures

Normally, the IT operations team should be the first contact for production workload failures. The IT operations team troubleshoots and attempts to recover the failed job, if possible. It is also helpful to notify the IT help desk of system failures that have an end user impact. These can include issues with the data warehouse used by the business intelligence (BI) analysts.

Best practice 6.5 – Automate the recovery of analytics and ETL job failures

Many factors can cause analytics and ETL jobs to fail. Job failures can be recovered using automated recovery solutions, however, others might require manual intervention. Designing and implementing an automated recovery solution can help reduce the impact of the job failures and streamline IT operations.

Suggestions 6.5.1 – Determine what types of job failures can be recovered safely using an automated recovery solution

Failures can be remediated by rerunning the job differently with understanding of the failure type using application logic. Identify failure types, the root cause of each type of failure, and whether automation can be used to recover from failure.

Suggestions 6.5.2 – Discover recovery procedures that work for multiple failure types

Configure automatic retries to handle intermittent network disruptions. Configure managed scaling to ensure that there are sufficient resources available for jobs to complete within specific time limits.

Suggestions 6.5.3 – Limit the number of automatic reruns and create log entries for the automatic recovery attempts and results

Track the number of reruns an automated recovery can attempt. Limit the number of reruns to avoid unnecessary reruns and resources. Log recovery attempts and outcomes to identify failure trends and drive future improvements.

Suggestion 6.5.4 – Design the job recovery solution based on the delivery SLA

Build systems that can meet SLA requirements even if jobs must be retried or manually recovered. If it's not practical to build systems that can recover within the agreed SLA, ensure that fail-back methods provide information that SLAs will not be met. For example, instead of sending a daily report to arrive by email at 6 AM, provide an email notice that the daily report will not be delivered that day.

Best practice 6.6 – Build a disaster recovery (DR) plan for the analytics infrastructure and the data

Discuss with business stakeholders to understand maximum amount of data loss (RPO) and maximum amount of service loss (RTO).

Suggestion 6.6.1 – Confirm the business requirement of the disaster recovery (DR) plan

Agree with the business shareholders what the internal and external SLAs are for your analytics processes. For example, not all business reports are business critical so it's important that your DR plans are aligned with the severity of the outage.

Suggestion 6.6.2 – Design the disaster recovery (DR) solution for each layer of the solution

Review the architecture for your data and analytics pipeline and select the DR pattern that meets your DR requirements.

Suggestion 6.6.3 – Implement and test your backup solution based on the RPO and RTO

Backup solutions must be implemented to reduce data loss. Periodically test your backup and restore solution to ensure that they are working. Your organization's backup is only as good as its last restore.

7 – Govern data and metadata changes

How do you govern data and metadata changes? Controlled changes are not only necessary for infrastructure, but also required for data quality assurance. If the data changes are uncontrolled, it

becomes difficult to anticipate the impact of these changes. It also makes downstream systems harder to manage data quality issues of their own.

ID	Priority	Best practice
<input type="checkbox"/> BP 7.1	Required	Build a central Data Catalog to store, share, and track metadata changes.
<input type="checkbox"/> BP 7.2	Required	Monitor for data quality anomalies.
<input type="checkbox"/> BP 7.3	Required	Trace data lineage.

Best practice 7.1 – Build a central Data Catalog to store, share, and track metadata changes

Building a central Data Catalog to store, share, and manage metadata across the organization is an integral part of data governance. This will promote standardization and reuse. Tracing metadata change history in the central Data Catalog helps you manage and control version changes in the metadata. A Data Catalog is often required for auditing and compliance.

Suggestion 7.1.1 – Changes on the metadata in the Data Catalog should be controlled and versioned

Use the Data Catalog change tracking features. For example, when the schema changes, AWS Glue Data Catalog will track the version change. You can use AWS Glue to compare schema versions, if needed. In addition, we recommend a change control process that only allows those authorized to make schema changes in your Data Catalog.

Best practice 7.2 – Monitor for data quality anomalies

Good data quality is critical to the downstream data workloads and data consumers, however, bad data can impact the accuracy of analytics insights and ML predictions. Monitor data quality and detect data anomalies as early as possible.

Suggestion 7.2.1 – Include a data quality check stage in the ETL pipeline as early as possible

A data quality check helps ensure that bad data is identified and fixed as soon as possible to prevent bad data from propagating downstream.

Suggestion 7.2.2 – Understand the nature of your data and determine the types of data anomalies that must be monitored and fixed based on the business requirements

The analytics workload can process various types of data, such as structured, unstructured, picture, audio, and video formats. Some data might arrive to the workload periodically, or some might constantly arrive in real time. It is pragmatic to assume that data does not always arrive to the analytics workload in perfect shape, and only a portion – not the whole set – of data matters to your workload.

Understand the characteristics of data, and determine what forms of data anomaly you want to remediate. For example, if you expect the data always contains an important attribute like customer ID, you can define that a datum is abnormal if it doesn't contain the customer_id attribute. Common data

anomalies include duplicate data, missing data, incomplete data, incorrect data format, and different measurement units.

Suggestion 7.2.3 – Select an existing data quality solution or develop your own based on the requirements

Some data quality solutions can only detect single field data quality issues. Other solutions can handle complex stateful data quality issues related to multiple fields.

Best practice 7.3 – Trace data lineage

Have a clear understanding about where your organization's data is coming from, how the data is transformed, who and what systems have access to the data, and how the data is used, is critical to increasing the business value of data. To achieve this goal, data lineage should be tracked, managed, and visualized.

Suggestion 7.3.1 – Track and control data lineage information

Data lineage information should include where data has come from, where the data is going, and who has access to the data. Data changes and the business logic used should also be tracked in the data lineage.

Suggestion 7.3.2 – Use visualization tools to investigate data lineage

Data lineage can become complicated when multiple systems are interacting with each other. Building a data lineage tool to visualize data lineage can reduce troubleshooting time and help identify downstream dependencies.

Suggestion 7.3.3 – Build a data lineage report to satisfy compliance and audit requirements

If some derestriction data lineage is required for compliance or audit purposes, your organization should either build a data lineage process using AWS services or investigate third-party applications.

For more details, refer to the following information:

- AWS data lineage blog: [Build data lineage for data lakes using AWS Glue, Amazon Neptune, and Spline](#)

Performance efficiency

The performance efficiency pillar focuses on the efficient use of resources to meet requirements as demand changes and technologies evolve. Performance optimization is not a one-time activity. It is an incremental and continual process of confirming business requirements, measuring the workload performance, identifying under-performing components, and tuning the components to meet your business needs.

Performance optimization should start with your organization's requirements, such as the business users of the analytics workload. Let the business stakeholders define the performance requirements and SLAs that must be met, then determine the computing requirements meeting their performance needs.

Best practices

- [8 – Choose the best-performing compute solution \(p. 30\)](#)
- [9 – Choose the best-performing storage solution \(p. 31\)](#)

- [10 – Choose the best-performing file format and partitioning \(p. 33\)](#)

8 – Choose the best-performing compute solution

How do you select the best-performing options for your analytics workload?

The definition of best-performing will mean different things to different stakeholders, so gathering all stakeholders' input in the decision process is key. Define performance and cost goals by balancing business and application requirements. Then evaluate the overall efficiency of the compute solution against those goals using metrics emitted from the solution.

ID	Priority	Best practice
<input type="checkbox"/> BP 8.1	Recommended	Identify analytics solutions that best suit your technical challenges.
<input type="checkbox"/> BP 8.2	Recommended	Provision compute resources to the location of the data storage.
<input type="checkbox"/> BP 8.3	Recommended	Define and measure the computing performance metrics.
<input type="checkbox"/> BP 8.4	Recommended	Continually identify under-performing components and fine-tune the infrastructure or application logic.

For more details, refer to the following information:

- AWS Whitepaper – Overview of Amazon Web Services: [Analytics](#)
- AWS Big Data Blog: [Building high-quality benchmark tests for Amazon Redshift using Apache JMeter](#)
- AWS Big Data Blog: [Top 10 performance tuning techniques for Amazon Redshift](#)

Best practice 8.1 – Identify analytics solutions that best suit your technical challenges

AWS has multiple analytics processing services that are built for specific purposes. These include Amazon Redshift for data warehousing, Amazon Kinesis for streaming data, and Amazon QuickSight for data visualization. Your organization should consider each step of the data analytics process as an opportunity to identify the right tool for the job.

Suggestion 8.1.1 – Identify the requirements based on the collected business metrics

Applications and services are designed to overcome specific challenges. It's essential that your organization identifies the right tool for the right job to meet your business and technical requirements. Choosing the incorrect technology can introduce performance issues, especially when processing data at scale.

For more details, refer to the following information:

- AWS Right Tool for the Job: [Databases on AWS: The Right Tool for the Right Job](#)
- AWS Right Tool for the Job: [How to Choose the Right Database](#)

Best practice 8.2 – Provision the compute resources to the location of the data storage

Data analytics workloads require moving data through a pipeline, either for ingesting data, processing intermediate results, or producing curated datasets. It is easier and more efficient to select the location of data processing services near where the data is stored. This approach is preferred to frequently copying or streaming large amounts of data to the processing location. For example, if an Amazon Redshift cluster frequently ingests data from a data lake, ensure that the Amazon Redshift cluster is in the same Region as your data lake S3 buckets.

Suggestion 8.2.1 – Migrate or copy primary data stores from on-premises environments to AWS so that cloud compute and storage are closely located

Avoid repeatedly transferring the same datasets from on-premises data stores to the cloud. Instead, create copies of your data near the analytics platform to avoid data transfer latency and improve overall performance of the analytics solution. Similarly, if data resides in a different AWS Region from the analytics systems, move either the analytics systems or the data to the same Region.

Suggestion 8.2.2 – Consider where your analytics resources are placed

Your organization should think about the locality of data and the locality of where the resources are to process that data. Where possible, your organization should consider using a permanent Region for all data analytics processing as this will help with data transferring overhead.

Best practice 8.3 – Define and measure the computing performance metrics

Define how you will measure performance of the analytics solutions for each step in the process. For example, if the computing solution is a transient Amazon EMR cluster, you can take the following approach. Define the performance as the Amazon EMR job runtime from the launch of the EMR cluster, process the job, then shut down the cluster. As another example, if the computing solution is an Amazon Redshift cluster that is shared by a business unit, you can define the performance as the runtime duration for each SQL query.

Suggestion 8.3.1 – Define performance efficiency metrics

Collect and use metrics to scale the resources to meet business requirements. By doing so, your team can track unexpected spikes to make future improvements.

Best practice 8.4 – Continually identify under-performing components and fine-tune the infrastructure or application logic

After you have defined the performance measurement, you should identify which infrastructure components or jobs are running below the performance criteria. Performance fine-tuning will vary per AWS service. For example, if it is an Amazon EMR cluster, you should change to a larger cluster instance type, or increase the number of cluster nodes. However, for an Amazon Redshift cluster, you can either increase the number of cluster nodes to increase parallel computing capacity, or fine-tune the SQL queries that are running below the performance criteria.

9 – Choose the best-performing storage solution

How do you select the best-performing storage options for your workload?

An analytics workload's optimal storage solution is determined by various aspects such as:

- Compute engine (Amazon EMR, Amazon Redshift, Amazon RDS, and so on)
- Access patterns (random or sequential)
- Required throughput
- Access frequency (online, offline, archival)
- Update patterns (append only, in-place update) availability
- Durability requirements

Choose the best-performing storage solution for your analytics workload's own characteristics.

ID	Priority	Best practice
<input type="checkbox"/> BP 9.1	Highly recommended	Identify critical performance criteria for your storage workload.
<input type="checkbox"/> BP 9.2	Highly recommended	Evaluate and confirm the available storage options for your compute solution.
<input type="checkbox"/> BP 9.3	Recommended	Choose the optimal storage based on access patterns, data growth, and the performance metrics.

For more details, refer to the following information:

- Amazon Elastic Compute Cloud User Guide for Linux Instances: [Amazon EBS volume types](#)
- Amazon Redshift Database Developer Guide: [Amazon Redshift best practices for loading data PDF](#)
- Amazon EMR Management Guide: [Instance storage](#)
- Amazon Simple Storage Service User Guide: [Best practices design patterns: Optimizing Amazon S3 performance](#)

Best practice 9.1 – Identify critical performance criteria for your storage workload

In data analytics, data throughput is often a constraining factor to enable your workloads to run effectively. Throughput is measured by the amount of information that has successfully moved through the network, compute, or storage layers. Improving throughput in each of these layers will result in better query performance.

Suggestion 9.1.1 – Use CloudWatch metrics or self-managed performance monitoring tools to determine if the analytics system performance is limited by compute, storage, or networking

Use a metric collection and reporting system, such as Amazon CloudWatch, to analyze the performance characteristics of the analytics system. Evaluate the measured performance metrics relative to system reference documentation to characterize the system constraints for the workload as a percentage of maximum performance.

Best practice 9.2 – Evaluate and confirm the available storage options for your compute solution

Many AWS data analytics services allow you to use more than one type of storage. For example, Amazon Redshift allows access to data stored in the compute nodes in addition to data stored in Amazon

S3. When performing research on each data analytics service, evaluate all of the storage options to determine the most performance efficient solution that meets business requirements.

Suggestion 9.2.1 – Review the available storage options for the analytics services being considered

For example, Amazon EMR provides local storage via HDFS file system and Amazon S3 as an external storage via EMRFS. For more information, refer to the AWS documentation for your compute solution:

- Amazon EMR Management Guide: [Work with storage and file systems](#)
- Amazon Redshift Cluster Management Guide: [Overview of Amazon Redshift clusters](#)
- Amazon OpenSearch Service Developer Guide: [Managing indices in Amazon OpenSearch Service](#)
- Amazon Aurora User Guide: [Overview of Aurora storage](#)

Suggestion 9.2.2 – Evaluate the performance of the selected storage option

To ensure that the overall analytics system design meets your non-functional requirements, evaluate the performance by running simulated real-world tests in a test environment.

Best practice 9.3 – Choose the optimal storage based on access patterns, data growth, and the performance metrics

Storage options for data analytics can have performance tradeoffs based on access patterns and data size. For example, in Amazon S3, it is much more efficient to retrieve a single 200GB compressed object which contains many small 1-KB log lines than to retrieve millions of individual 1-KB log objects. Evaluate your workload needs and usage patterns to determine if the method or location of storing your data can improve the overall efficiency of your solution.

Suggestion 9.3.1 – Identify available solution options for the performance improvement

When data I/O is limiting performance and business requirements are not being met, improve I/O through the options available within that service. For example, with EBS volumes of GP3 type, increase Provisioned IOPS or throughput, or for Amazon Redshift, increase the number of nodes.

10 – Choose the best-performing file format and partitioning

How do you select the best-performing file formats and partitioning? Selecting the best-performing file format and data partitioning for data-at-rest can have a large impact on the overall analytics workload efficiency.

ID	Priority	Best practice
<input type="checkbox"/> BP 10.1	Recommended	Select format based on data write frequency and patterns for append-only vs. in-place update.
<input type="checkbox"/> BP 10.2	Recommended	Choose data formatting based on your data access pattern
<input type="checkbox"/> BP 10.3	Recommended	Use file compression to reduce number of files and to improve file I/O efficiency.

ID	Priority	Best practice
<input type="checkbox"/> BP 10.4	Recommended	Partition your data to avoid unnecessary file reads.

For more details, refer to the following information:

- Amazon Redshift Database Developer Guide: [Creating data files for queries in Amazon Redshift Spectrum](#)
- Amazon EMR Release Guide: [Hudi](#)
- AWS Big Data Blog: [Apply record level changes from relational databases to Amazon S3 data lake using Apache Hudi on Amazon EMR and AWS Database Migration service](#)

Best practice 10.1 – Select format based on data write frequency and patterns for append-only vs. in-place update

Review your data storage access patterns and performance requirements for a streaming data ingestion case. Instead of storing many small data files individually, periodically batch the small files into a single larger compressed file. This approach will provide better data retrieval performance when using EMR.

Suggestion 10.1.1 – Understand your analytics workload data's write characteristics

If storing data in Amazon S3, evaluate if an append-only method, such as Apache Hudi, is right for your needs.

Best practice 10.2 – Choose data formatting based on your data access pattern

Choosing the right data type for your workload is important. There are many different data types available to support your workload, however, choosing the wrong data type can severely impact the performance of your analytics workload.

Suggestion 10.2.1 – Decide the correct data format for your analytics workload

You can work on unstructured, semi-structured, and structured data formats (CSV, JSON, or columnar formats such as Apache Parquet and Apache ORC) without the need to aggregate your data. Alternately, you can load the data into a database. You can use Amazon Athena to run ad-hoc queries using ANSI SQL on the data stored in Amazon S3.

For example, instead of reshaping the data when a query is processed (as an RDBMS system does), a NoSQL database (such as Amazon DynamoDB) can organize the data so that its shape in the database corresponds with what will be queried. This is a key factor in increasing speed and scalability.

Amazon Redshift data sharing enables instant, granular, and high-performance data access across Amazon Redshift clusters without the need to copy or move it. Data sharing provides live access to data so that you can always see the most up-to-date and consistent information while it's being updated in the data warehouse.

Suggestion 10.2.2 – API-driven data access pattern constraints, such as the amount of data retrieved per API call, can impact overall performance

Refer to [Service endpoints and quotas](#) for API quotas per service.

Suggestion 10.2.3 – Use data, results, and query cache to improve performance and reduce reads from the storage tier

Caching services can speed up the responses to common queries and reduce the load on the storage tier. Use Amazon ElastiCache, Amazon DynamoDB Accelerator (DAX), API gateway caching, or other relevant caching services.

Best practice 10.3 – Use file compression to reduce number of files and to improve file I/O efficiency

Store data in a compressed format to reduce the burden on the underlying storage host and network. For example, for columnar data stored in Amazon S3, use a compatible compression algorithm that supports parallel reads, such as bzip2.

We recommend that your organization test the performance and storage overhead of both uncompressed and compressed datasets to determine best fit prior to implementing this approach.

Suggestion 10.3.1 – Compress data to reduce the transfer time

When storage read/write performance becomes a bottleneck, use compression to reduce data transfer time. Consider the tradeoffs between compute time needed to perform compression and decompression versus the storage I/O bottleneck in your estimates of overall improvements in performance efficiency.

Suggestion 10.3.2 – Evaluate the available compression options for each resource of the workload

Compressing data can improve the performance as there are fewer bytes transferred between the disk and compute layers. The trade-off using this approach is that it requires more compute for data compression and decompression. You can, however, obtain a net efficiency improvement if compression performs as well as or better than uncompressed data transfer time. Compression also requires much less storage, depending on the data type in use, thus saving on data storage latency and costs.

Best practice 10.4 – Partition your data to avoid unnecessary file reads

Storing your data in structured partitions will allow compute to identify the location of only that portion of the data relevant to the query. Determine the most frequent query parameters and store this data in the appropriate location suited to your data retrieval needs. For example, if an analytics workload regularly generates daily, weekly, and monthly reports, then store your data using partitions with a year/month/day format.

Suggestion 10.4.1 – Store data partitioned based on columns with incremental processing constraints

This partitioning scheme is most often used with data processing needed for different time windows, such as daily, weekly, or monthly reports.

Suggestion 10.4.2 – Store data partitioned based on reducing number of files or compaction

Store data that is accessed frequently (*hot* data) and data that is accessed infrequently (*cold* data) into separate partitions. For example, if you have a regulatory reason to archive data, you can store the data in an S3 bucket with a prefix such as /archive. You can configure a S3 Lifecycle policy to move it automatically into Amazon S3 Glacier. Or, if you have too many small files to process, you can

consolidate such small files into a S3 bucket with a prefix such as /raw-data, and you can generate a pre-processed, big-size-but-less-number of files into a S3 bucket with a prefix of /pre-processed for faster analytical processing on these files.

Suggestion 10.4.3 – Store data partitioned based on columns frequently used in queries for filtering

This partitioning scheme is most often implemented on your access patterns, therefore if the WHERE clause in a SELECT statement commonly uses DATE, then your partitioning strategy should be based upon DATE.

Suggestion 10.4.4 – Store data partitioned based on time attributes with earlier data stored in tiers that are accessed infrequently

Use the tiering capabilities of the storage service to put infrequently-accessed data into the tier that is most appropriate for the workload. For example, in an Amazon Redshift data warehouse, data that is accessed infrequently can be stored in Amazon S3. Then you can query it with Amazon Redshift Spectrum, while more frequently-accessed data can be stored in local Amazon Redshift storage.

Cost optimization

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle to optimize cost. Cost optimization is a key effort, from the initial design of your first proof of concept, to the ongoing operation of production workloads. It's a years-long, continual process. Choose the right solution and pricing model. Build cost-aware systems that allow you to achieve business outcomes and minimize costs. To perform cost optimization over time, you should identify data, infrastructure resources, and analytics jobs that can be removed or downsized.

Determine the 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 you decide where to focus engineering resources for development, and to perform a return on investment (ROI) estimation for the analytics portfolio as a whole.

Best practices

- [11 – Choose cost-effective compute and storage solutions based on workload usage patterns \(p. 36\)](#)
- [12 – Build financial accountability models for data and workload usage \(p. 39\)](#)
- [13 – Manage cost over time \(p. 41\)](#)
- [14 – Use optimal pricing models based on infrastructure usage patterns \(p. 44\)](#)

11 – Choose cost-effective compute and storage solutions based on workload usage patterns

How do you select the compute and storage solution for your analytics workload? Your initial design choice could have significant cost impact. Understand the resource requirements of your workload, including its steady-state and spikiness, and then select the solution and tools that meet your requirements. Avoid over-provisioning to allow more cost optimization opportunities.

ID	Priority	Best practice
<input type="checkbox"/> BP 11.1	Recommended	Decouple storage from compute.

ID	Priority	Best practice
<input type="checkbox"/> BP 11.2	Recommended	Plan and provision capacity for predictable workload usage.
<input type="checkbox"/> BP 11.3	Recommended	Use On-Demand Instance capacity for unpredictable workload usage.

For more details, refer to the following information:

- Amazon Elastic Compute Cloud User Guide for Linux Instances: [Get recommendations for an instance type](#)
- AWS Cost Management and Optimization – AWS Cost Optimization: [Right Sizing](#)
- AWS Whitepaper – Right Sizing: Provisioning Instances to Match Workloads: [Tips for Right Sizing](#)

Best practice 11.1 – Decouple storage from compute

It's common for data assets to grow exponentially year over year. However, your compute needs might not grow at the same rate. Decoupling storage from compute allows you to manage the cost of storage and compute separately, and implement different cost optimization features to minimize cost.

Suggestion 11.1.1 – Use low-cost compute resources, such as Amazon EC2 Spot Instances, when applicable

Decouple storage from compute allows stateless logic implementation, consider using Amazon EC2 Spot Instances for additional cost savings. Spot Instances are also an excellent option for reducing costs for development or quality assurance (QA) applications.

Suggestion 11.1.2 – Use Amazon Redshift RA3 instances types

Amazon Redshift RA3 instance types support the ability to decouple the compute and storage. This allows your Amazon Redshift storage to scale independently from your compute resources, which improves cost efficiencies for your data warehousing workloads.

Suggestion 11.1.3 – Use EMR file system (EMRFS)

The EMR file system (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. By using EMRFS, your organization is only charged for the storage used, rather than paying for overprovisioned and underutilized HDFS EBS storage.

Suggestion 11.1.4 – Use Amazon S3 Select and Amazon S3 Glacier Select to reduce data retrieval

Amazon S3 Select and Amazon S3 Glacier Select allow applications to retrieve only a subset of data from an object by using simple SQL expressions. This reduces the overall amount of information that has been requested and reduces downstream data duplication.

Best practice 11.2 – Plan and provision capacity for predictable workload usage

For well-defined workloads, planning capacity ahead based on average usage pattern helps improve resource utilization and avoid over provisioning. For a spiky workload, set up automatic scaling to meet user and workload demand.

Suggestion 11.2.1 – Choose the right instance type based on workload pattern and growth ratio

Consider resource needs, such as CPU, memory, and networking that meet the performance requirements of your workload. Choose the right instance type and avoid overprovisioning. An optimized EC2 instance runs your workloads with optimal performance and infrastructure cost. For example, choose the smaller instance if your growth ratio is low as this allows more granular incremental change.

Suggestion 11.2.2 – Choose the right sizing based on average or medium workload usage

Right sizing is the process of matching instance types and sizes to your workload performance and capacity requirements at the lowest possible cost. It's also the process of looking at deployed instances and identifying opportunities to downsize without compromising capacity or other requirements that will result in lower costs.

Suggestion 11.2.3 – Use automatic scaling capability to meet the peak demand instead of over provisioning

Analytics services can scale dynamically to meet demand. Then, after the demand has dropped below a certain threshold, the service will remove the resources that are no longer needed.

Best practice 11.3 – Use On-Demand Instance capacity for unpredictable workload usage

Serverless solutions are either charged by the amount of data processed or the compute resource used, but only when there is a workload actively running. Compare this approach to using long running infrastructure, but where you don't have to pay for idle resources.

Suggestion 11.3.1 – Use Amazon Athena for ad hoc SQL workloads

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.

Suggestion 11.3.2 – Use AWS Glue instead of Amazon EMR for infrequent ETL jobs

AWS Glue is a fully managed ETL (extract, transform, and load) service that makes it simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores and data streams. With AWS Glue jobs, you pay only for the resources used during the ETL process. In contrast, Amazon EMR is typically used for frequently running jobs requiring semipersistent data storage.

Suggestion 11.3.3 – Use On-Demand Instance resources for transient workloads or short-term development and testing needs

Use On-Demand Instance resources, such as EC2 Spot Instances on EMR, to perform experimental or test data processing at lower cost. During the initial planning and testing phases of analytics workloads, using On-Demand Instance resources provides flexibility to iterate through many alternative architectures until all aspects of a well-architected analytics project are optimized for the job. Some examples include selecting the right instance type or using a serverless architecture to meet the required time-duration of analytics pipeline runs.

12 – Build financial accountability models for data and workload usage

How do you measure and attribute the analytics workload financial accountability? As your business continues to evolve, so will your analytics workload. Data analytics systems and the data generated from them will grow over time into a mix of both shared and isolated-team resources. Your organization should establish a financial attribution model for these resources. Teams will understand how their use of data analytics influences costs to the business and this promotes a culture of accountability and frugality. Creating a financial accountability model will allow departments to cross-charge departments for shared resources.

ID	Priority	Best practice
<input type="checkbox"/> BP 12.1	Recommended	Measure data storage and processing costs per user of the workload.
<input type="checkbox"/> BP 12.2	Recommended	Allow downstream systems to use their own resources for analytics jobs.
<input type="checkbox"/> BP 12.3	Recommended	Build a common, shared processing system and measure the cost per analytics job.
<input type="checkbox"/> BP 12.4	Recommended	Build an internal charge back model for analytics workload usage.
<input type="checkbox"/> BP 12.5	Recommended	Restrict and record resource allocation permissions using AWS Identity and Access Management (IAM).

For more details, refer to the following information:

- AWS Cloud Financial Management Blog: [Cost Allocation Blog Series #1: Cost Allocation Basics That You Need to Know](#)
- AWS Cloud Enterprise Strategy Blog: [Who Pays? Decomplexifying Technology Charges](#)
- AWS Cloud Enterprise Strategy Blog: [Strategy for Efficient Cloud Cost Management](#)
- AWS Cloud Financial Management Blog: [Trends Dashboard with AWS Cost and Usage Reports, Amazon Athena, and Amazon QuickSight](#)
- AWS Well-Architected Labs: [Cost Optimization](#)

Best practice 12.1 – Measure data storage and processing costs per user of the workload

Data analytics workloads have recurring stable costs and per-use costs, for example, a weekly reporting job with relatively static data storage fees or periodic unpredictable processing runtime fees. Your organization should establish a financial attribution mechanism that captures data storage and workload usage when analytics systems are run. Using this approach, your end users (business unit, team, or individual) can be notified of their consumption at regular intervals.

Suggestion 12.1.1 – Use tagging or other attribution methods to identify workload and data storage ownership

Collaboration between business, IT, and finance team to agree on cost allocation, cost ownership, cost charging, and budget management. Create budget tracking policy for storage and workload using

tagging. Agree on the governance approach to implement policy (that is, central and decentralize), billing allocation, charge back, and budget reporting.

For more details, refer to the following information:

- AWS Cloud Financial Management Blog: Cost [Tagging and Reporting with AWS Organizations](#)
- AWS Billing and Cost Management and Cost Management User Guide: [Reporting your budget metrics with budget reports](#), [Configuring AWS Budgets actions](#) and [Creating an Amazon SNS topic for budget notifications](#)

Best practice 12.2 – Allow downstream systems to use their own resources for analytics jobs

Teams can establish their own data analytics resources that support their analytical needs locally, rather than extracting information and transferring it to a central location. Decide when teams will benefit from building local analytics resources balancing required agility and team skillset with the need for a centralized analytics platform.

Suggestion 12.2.1 – Perform regular reviews of analytics operations to determine if the business will benefit from teams managing their own infrastructure

In some cases, teams will want to maintain and manage their own infrastructure. Individual team ownership allows greater flexibility and agility when performing system design choices with fewer dependencies. Furthermore, individual team ownership provides clear cost visibility to the team that own the resources.

Best practice 12.3 – Build a common, shared processing system and measure cost per analytics job

It is more efficient to have a shared processing system where individual teams can send requests for data analytics. Your organization should measure the number of requests by individual teams so that each teams' request can be attributed a cost.

Suggestion 12.3.1 – Perform regular reviews of analytics operations to determine if the business will benefit from shared analytics infrastructure

A centralized team will maintain and manage infrastructure for the benefit of multiple groups. Shared infrastructure allows greater utilization of the resources deployed and concentrated skillsets for infrastructure management. Centralized data repositories deliver the benefit of data being enriched easily by another data store. It also provides an advantage for teams to have a single endpoint to connect to the data. Organizations have reported that having a central location to perform analytics makes delivering against their compliance and data governance requirement much simpler.

Suggestion 12.3.2 – Implement a per-request tracking mechanism that allows utilization attribution to the different teams

With a shared resource, some businesses want to maintain the cost visibility of the teams using those resources. Implementing a per-request tracking mechanism will allow measurement of the associated resource billing metrics during the job and will attribute those portions of the job to the requester. The metrics collected and the associated users will be aggregated into a cost visibility report.

Best practice 12.4 – Build an internal chargeback model for analytics workload usage

After establishing the best practices of measuring and associating the teams' consumption, your organization should build an internal chargeback model. This will indicate each team's financial responsibility for the analytics data that they consume.

Suggestion 12.4.1 – Implement a cost-visibility and internal billback method to aggregate the team's utilization of analytics resources

Notify teams of their analytics usage costs periodically. Build dashboards that provide teams visibility into how their work impacts costs to the business using a self-service approach.

Best practice 12.5 – Restrict and record resource allocation permissions using AWS Identity and Access Management (IAM)

Create specific IAM roles to provide authorization to provision certain resources.

Suggestion 12.5.1 – Create a cost governance framework that uses specialized IAM roles, rather than individual users, to provision costly infrastructure

Restrict the authorization to launch costly resources to specific IAM roles. For example, certain instances types can only be provisioned by certain teams to reduce unnecessary expenditure.

Suggestion 12.5.2 – Track IAM usage logs to determine overall usage-per-user and role

Track the usage across users and roles to get a clear understanding of resource usage. As part of your cost-allocation governance, automatically process the IAM logs so that cost allocation is properly attributed to the relevant department.

13 – Manage cost over time

How do you manage the cost of your workload over time? To ensure that you always have the most cost-efficient workload, periodically review your workload to discover opportunities to implement new services, features, and components. It is common for analytics workloads to have an ever-growing number of users and exponential growth of data volume. Implement a standardized process across your organization to identify and remove unused resources, such as unused data, infrastructure, and ETL jobs.

ID	Priority	Best practice
<input type="checkbox"/> BP 13.1	Recommended	Remove unused data and infrastructure.
<input type="checkbox"/> BP 13.2	Recommended	Reduce overprovisioning infrastructure.
<input type="checkbox"/> BP 13.3	Recommended	Reduce repeating analytics jobs with no business value.
<input type="checkbox"/> BP 13.4	Recommended	Evaluate and adopt new cost-effective solutions.

For more details, refer to the following information:

- AWS Database Blog: [Safely reduce the cost of your unused Amazon DynamoDB tables using On-Demand mode.](#)
- AWS Management and Governance Blog: [Controlling your AWS costs by deleting unused Amazon EBS volumes.](#)
- AWS Database Blog: [Implementing DB Instance Stop and Start in Amazon RDS.](#)
- AWS Big Data Blog: [Lower your costs with the new pause and resume actions on Amazon Redshift.](#)
- AWS Partner Network (APN) Blog: [Scaling Laravel Jobs with AWS Batch and Amazon EventBridge.](#)
- AWS Glue Developer Guide: [Tracking Processed Data Using Job Bookmarks.](#)

Best practice 13.1 – Remove unused data and infrastructure

Delete data that is out of its retention period, or not needed anymore. Delete intermediate-processed data that can be removed without business impacts. If the output of analytics jobs is not used by anyone, consider removing such jobs so that you don't waste resources.

Suggestion 13.1.1 – Track data freshness

In many cases, maintaining a metadata repository for tracking data movement will be worthwhile. This is not only to instill confidence in the quality of the data, but also to identify infrequently updated data, and unused data.

Suggestion 13.1.2 – Delete data that is out of its retention period

Data that is past its retention period should be deleted to reduce unnecessary storage costs. Identify data through the metadata catalog that is outside its retention period. To reduce human effort, automate the data removal process. If data is stored in Amazon S3, use Amazon S3 Lifecycle configurations to expire data automatically.

Suggestion 13.1.3 – Delete intermediate-processed data that can be removed without business impacts

Many steps in analytics processes create intermediate or temporary datasets. Ensure that intermediate datasets are removed if they have no further business value.

Suggestion 13.1.4 – Remove unused analytics jobs that consume infrastructure resources but no one uses the job results

Periodically review the ownership, source, and downstream consumers of all analytics infrastructure resources. If downstream consumers no longer need the analytics job, stop the job from running and remove unneeded resources.

Best practice 13.2 – Reduce overprovisioning infrastructure

Workload resource utilization can change over time, especially with the growth of data or after process optimization has occurred. Your organization should review resource usage patterns and determine if you require the same infrastructure footprint to meet your business goals.

Suggestion 13.2.1 – Evaluate whether compute resources can be downsized

Investigate your resource utilization by inspecting the metrics provided by Amazon CloudWatch. Evaluate whether the resources can be downsized to one-level smaller within the same instance class. For example, reduce Amazon EMR cluster nodes from m5.16xlarge to m5.12xlarge, or the number of instances that make up the cluster.

Suggestion 13.2.2 – Move infrequently used data out of a data warehouse into a data lake

Data that is infrequently used can be moved from the data warehouse into the data lake. From there, the data can be queried in place or joined with data in the warehouse. Use services such as Amazon Redshift Spectrum to query and join data in the Amazon S3 data lake, or Amazon Athena to query data at rest in Amazon S3.

Suggestion 13.2.3 – Merge low utilization infrastructure resources

If you have several workloads that all have low-utilization resources, determine if you can combine those workloads to run on shared infrastructure. In many cases, using a pooled resource model for analytics workloads will save on infrastructure costs.

Suggestion 13.2.4 – Move infrequently accessed data into low-cost storage tiers

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

Transitioning between storage tiers is achieved using Amazon S3 Lifecycle policies. These automatically transition objects into another tier with lower cost, and will even delete expired data. Amazon S3 Intelligent-Tiering will analyze the data access patterns and automatically move objects between tiers.

Best practice 13.3 – Reduce repeating analytics jobs with no business value

It is a common human behavior to expect frequently refreshed data. However, the frequency of running analytics jobs should be considered in the context of its business value. There is no need to refresh data repeatedly if the data is not used frequently.

Suggestion 13.3.1 – Use the lowest acceptable frequency for data processing

Data processing requirements must be considered in the business context. There is no value in processing data faster than it will be consumed or delivered. For example, in a sales analytics workload, it might not be necessary to perform analytics on each transaction as it arrives. In some cases, only hourly reports are needed by business management. Batch processing the transactions will be more efficient and can reduce unnecessary infrastructure costs between batch processing jobs.

Best practice 13.4 – Evaluate and adopt new cost-effective solutions

As AWS releases new services and features, it's a best practice to review your existing architectural decisions to ensure that they remain cost effective. If a new or updated service can support the same workload but in a much cheaper way, consider implementing the change to reduce cost.

Suggestion 13.4.1 – Set Service Quotas to control resource usage

Some AWS services allow setting Service Quotas per account. Service Quotas should be established to prevent runaway infrastructure deployment by accident. Ensure that Service Quotas are set high enough to cover the expected peak usage.

Suggestion 13.4.2 – Pause and resume resources if the workload is not always required

Use automation to pause and resume resources when the resource is unneeded. For example, stop development and test Amazon RDS instances that are not used after working hours.

Suggestion 13.4.3 – Switch to a new service or take advantage of new features that can reduce cost

AWS consistently adds new capabilities to enable your organization to leverage the latest technologies to experiment and innovate more quickly. Your organization should review new service releases frequently to understand the price and performance, and determine if such features can improve cost reduction.

14 – Use optimal pricing models based on infrastructure usage patterns

How do you choose the financially-optimal pricing models of the infrastructure? Consult with your finance team and choose optimal purchasing options, such as On-Demand Instances, Reserved Instances, or Spot Instances. Understand the infrastructure usage patterns of the analytics workload. You can optimize the cost by purchasing reserved capacity with upfront payment by using Spot Instances, or by paying Amazon EC2 usage via On-Demand Instance pricing models. Evaluate the available purchasing models of the analytics infrastructure of your choice and determine the optimal payment models.

ID	Priority	Best practice
<input type="checkbox"/> BP 14.1	Recommended	Evaluate the infrastructure usage patterns then choose payment options accordingly.
<input type="checkbox"/> BP 14.2	Recommended	Consult with your finance team and determine optimal payment models.

For more details, refer to the following information:

- AWS Cloud Enterprise Strategy Blog: [Managing Your Cost Savings with Amazon Reserved Instances](#).
- AWS Big Data Blog: [How Goodreads offloads Amazon DynamoDB tables to Amazon S3 and queries them using Amazon Athena](#).
- AWS Big Data Blog: [Best practices for resizing and automatic scaling in Amazon EMR](#).
- AWS Big Data Blog: [Work with partitioned data in AWS Glue](#).
- AWS Big Data Blog: [Using Amazon Redshift Spectrum, Amazon Athena, and AWS Glue with Node.js in Production](#).
- AWS Compute Blog: [10 things you can do today to reduce AWS costs](#).
- AWS Billing and Cost Management and Cost Management User Guide: [Using Cost Allocation Tags](#).
- AWS Well-Architected Framework: [Cost Optimization Pillar](#).
- AWS Whitepaper: [Laying the Foundation: Setting Up Your Environment for Cost Optimization](#).
- AWS Whitepaper: [Amazon EC2 Reserved Instances and Other AWS Reservation Models](#).
- AWS Whitepaper: [Overview of Amazon EC2 Spot Instances](#).
- AWS Whitepaper: [Right Sizing: Provisioning Instances to Match Workloads](#).
- AWS Whitepaper: [AWS Storage Optimization](#).
- Amazon Redshift: [Purchasing Amazon Redshift reserved nodes](#).

Best practice 14.1 – Evaluate the infrastructure usage patterns and choose your payment options accordingly

On-demand resources provide immense flexibility with pay-as-you-go payment models across multiple scenarios and scales. Alternately, Reserved Instances provide significant cost saving for workloads that have steady resource utilization. Perform regular workload resource usage analysis. Choose the best pricing model to ensure that you don't miss cost optimization opportunities and maximize your discounts.

Suggestion 14.1.1 – Evaluate available payment options of the infrastructure resources of your choice

Review the pricing page for specific AWS services. Each service will list the billing metrics, such as runtime or gigabytes processed, as well as any discount options for dedicated usage. In addition, many AWS analytics services offer discounted payment terms, Reserved Instances, or Savings Plans, in exchange for a specific usage commitment. Almost all AWS services offer the payment for usage on demand, meaning you only pay for what you use.

Suggestion 14.1.2 – For steady, permanent workloads, obtain Reserved Instances or Savings Plans price discounts instead of paying On-Demand Instance pricing

Reserved Instances give you the option to reserve some AWS resources for a one- or a three-year term. In turn, you will receive a significant discount compared with the On-Demand Instance pricing. Workloads that have consistent long-term usage are good candidates for the Reserved Instance payment option.

Suggestion 14.1.3 – Use On-Demand Instance resources during development and in pre-production environments

Development and pre-production environments frequently change and often do not require 100% availability. On-Demand Instance resources are the best option in cases where workload utilization is unpredictable, is frequently changed, or is only used for portions of the day.

Best practice 14.2 – Consult with your finance team and determine optimal payment models

If you use reserved-capacity pricing options, you can reduce the infrastructure cost without modifying your workload architectures. Collaborate with your finance team on the planning and use of purchase discounts.

Make informed decisions regarding various cost factors. These include the amount of capacity to reserve, the reserve term length, and the choice of upfront payments for their corresponding discount rates. The finance team should assist your team in determining the best long-term and reserved-capacity pricing options. This is because these options affect your IT budget plans, such as which month is the right moment to pay an upfront charge.

Suggestion 14.2.1 – Consolidate the infrastructure usage to maximize the coverage of reserved capacity price options

Reserved Instances and Savings Plan purchases apply automatically to the resources that will receive the largest discount benefit. To maximize your discount utilization, consolidate resources in accounts within an AWS Organization structure. Allow the purchase commitments to apply to other AWS accounts within your organization if they are unused in the account for which they are purchased.

Sustainability

Organizations and government departments play a critical role in conserving natural resources and protecting global ecosystems by reducing the use of materials or resources.

The practice of designing and building sustainable cloud workloads is understanding what emissions are attributable to your IT usage has on your organization's carbon footprint. You can then apply these design principles and best practices to reduce these emissions.

Sustainability in the cloud is a continuous effort focused primarily on energy reduction and efficiency across all components of a workload. You can do this by achieving the maximum benefit from the resources provisioned and minimizing the total resources required. This effort can range from the initial selection of an efficient programming language, adoption of modern algorithms, use of efficient data storage techniques, deploying to correctly sized and efficient compute infrastructure, and minimizing requirements for high-powered end user hardware.

How do you ensure the services and the infrastructure deployed to ingest, process, and analyze data have been designed with sustainability as an architectural principle? This section details how to design your data platforms using sustainability architectural best practices to reduce your organization's data analytics carbon footprint.

Throughout this section, we explore various best practices to help understand how they can reduce your organization's analytics carbon footprint. Implementing each of the best practices involves resource trade-offs. Your organization should examine these best practices, both holistically and individually, and agree on whether they will be beneficial in meeting your sustainability goals. Data compression, for instance, will minimize your storage footprint. But, as a trade-off, more computing power is required to decompress the data. It is advised that your company tests the best practices recommendations to determine the level of storage vs. compute trade-off and identify which approach is most sustainably beneficial.

Best practice

- [15 – Sustainability implementation guidance \(p. 46\)](#)

15 – Sustainability implementation guidance

Think about sustainability being a non-functional requirement when designing your systems. Determine how necessary sustainability best practices baked into your development lifecycle are, because sustainability best practice can be applied across all workloads, not just data and analytics.

ID	Priority	Best practice
<input type="checkbox"/> BP 15.1	Recommended	Baseline your organization's sustainability improvements.
<input type="checkbox"/> BP 15.2	Recommended	Encourage a trading carbon for data culture.
<input type="checkbox"/> BP 15.3	Recommended	Encourage a data minimization culture to reduce your organization's data impacts.
<input type="checkbox"/> BP 15.4	Recommended	Implement data retention processes to remove redundant data from your analytics environment.
<input type="checkbox"/> BP 15.5	Recommended	Optimize your data modeling and data storage for efficient data retrieval.
<input type="checkbox"/> BP 15.6	Recommended	Prevent unnecessary data movement between systems and applications.

ID	Priority	Best practice
<input type="checkbox"/> BP 15.7	Recommended	Efficiently manage your analytics infrastructure to reduce underutilized resources.

Best practice 15.1 – Baseline your organization’s sustainability improvements

As an organization, you will want to track your progress towards your sustainability goals. By tracking your environmental impacts, you can build a picture of what changes have made the biggest differences over time.

How do you track your analytics carbon footprint?

Suggestion 15.1 – Baseline the carbon footprint of your workload using the AWS Customer Carbon Footprint Tool

Baselining your analytics carbon footprint at the start of your optimization’s journey is important as it will provide you an understanding of how your changes have reduced your organization’s analytics carbon footprint. If you are an AWS user, your organization can use the AWS Customer Carbon Footprint Tool. The AWS Customer Carbon Footprint Tool is a data tracking and visualization tool that reports on your AWS accounts carbon usage.

Your organization should maintain an audit trail of the changes that your team have made, when they were made, and the impact that the changes had on the carbon footprint of each workload.

For more details, refer to the following information:

- AWS Carbon Footprint: [AWS Customer Carbon Footprint Tool](#)
- AWS Carbon Footprint overview: [AWS Customer Carbon Footprint Tool Overview](#)
- [Sustainability Pillar Improvement Process](#)

Best practice 15.2 – Encourage a trading carbon for data culture

If data is the new oil, should we not be just as worried about the impact data usage has on our planet? Senior leaders often require accurate and timely information to make informed decisions on the latest changes affecting their industry and business. Organizations strive to have the most up-to-date, and accurate information available. As a result, data engineers have developed more processes and increased the frequency of these processes to meet management demand.

Sustainability is not about removing your organization’s data. It’s about the availability or service level of your data assets. For example, organizations may not have considered the long-term sustainability impact of refreshing a report every n-times, or enforcing a full refresh of a data warehouse. In both cases, your organization is not removing data but considering the availability of your organization’s data assets.

Organizations should review the reports, the refresh frequency, and the KPIs per content included and decide whether it’s essential, and understand what the sustainability cost is. How does your organization evaluate the value of data assets against their carbon cost?

Suggestion 15.2.1 – Review your management information reports and decide on the optimal refresh rate

Business reporting requirements vary from one business group to another. One reporting dashboard might have to be run on demand, meaning that you are running a report with your own customized filter

at the point of accessing the report. Another set of report dashboards, however, might require scheduled updates, meaning that the report is run in the background at a scheduled time with pre-defined filter. Knowing the business requirement helps you to decide which refresh approach is required, and can reduce your analytics carbon footprint.

Suggestion 15.2.2 – Review your business reports and identify metrics that are no longer required

As organizations evolve, so do the management information dashboards. Over time, these dashboards will contain metrics that are no longer of business importance. These should be regularly reviewed and updated to ensure only the information required is generated. Efficient reporting will have a positive impact on your sustainability goals. Your organization can identify similar goals across teams/departments to reduce the number of separate reports and thereby reduce duplication and overlap.

Consider measuring your data assets by using proxy metrics. A proxy metric is a data point that represents the value of something through data. For example, your organization can use a proxy metric to determine how valuable and how frequently-accessed your business reports are. Then you can trade off these proxy metrics with carbon metrics to determine the sustainability impact of the solution.

The best way for you to evaluate success in optimizing your workloads for sustainability is to use proxy measures. It is worth noting that there is an obvious goal to reduce the overall values for many of these measures, but it is also important to reduce values per unit of work. As your business grows, naturally some of these values will increase, simply to meet the demands of your business. Reviewing these KPIs per unit of work helps clarify progress, even while the business grows (for example, reduce the number of vCPUs per 1,000 transactions).

To monitor and quantify improvements, track the metrics in the following table that contribute towards each pillar of the modern data architecture. This is not an exhaustive list of metrics. The idea is to use them as guidance and look for inefficiencies to optimize. The information linked will provide you with additional metrics that can be used to identify potential areas for improvement.

Table 2. – Sustainability Metrics for the Modern Data Architecture

Area	Optimization goals	Metrics, tools, and reference documentation
Data ingestion	<ul style="list-style-type: none">• Avoid unnecessary ingestion.• Reduce ingestion size.• Maximize utilization.	<p>Right size DMS instances – Choose the appropriate instance DMS Replication Instance and Task Metrics – CPUAllocated, CPUUtilization, WriteThroughput, ReadThroughput, CDCLatencySource, CDCLatencyTarget.</p> <p>CloudWatch Metrics – Amazon Kinesis – ReadProvisionedThroughputExceeded, WriteProvisionedThroughputExceeded</p> <p>CloudWatch Metrics – Amazon MSK – BurstBalance, CpuIdle, CpuIoWait, KafkaAppLogsDiskUsed, KafkaDataLogsDiskUsed, MemoryUsed</p>
Data lake	<ul style="list-style-type: none">• Maximum utilization• Optimize data lifecycle management	<p>Amazon S3 Storage Lens – Visualize storage usage and activity trends</p>

Area	Optimization goals	Metrics, tools, and reference documentation
		CloudWatch Metrics for Amazon S3 – BucketSizeBytes
Unified data governance	<ul style="list-style-type: none"> Reduce unnecessary cataloging 	Use CloudTrail events for monitoring crawler runs and job runs. Use appropriate Crawler properties
Data movement	<ul style="list-style-type: none"> Right-size worker nodes Monitor shuffling and optimize partitioning Reduce or stop unnecessary data movement 	AWS Glue CloudWatch Metrics – glue.driver.aggregate.recordsRead, glue.driver.aggregate.shuffleBytesWritten, glue.driver.aggregate.shuffleLocalBytesRead, glue.driver.BlockManager.disk.diskSpaceUsed ASW Glue Job Monitoring console – DPU Usage, WorkerNode Breakdown

Area	Optimization goals	Metrics, tools, and reference documentation
Purpose-built analytics and insights	<ul style="list-style-type: none"> • Maximize resource utilization • Right size instances • Reduce Storage • Reduce amount of data scanned 	<ul style="list-style-type: none"> • Amazon Redshift Cluster Performance Data – CPU utilization, Percentage disk space used, Read throughput, Write throughput, Write latency, Query duration, Query throughput • Amazon Redshift Database Performance – Query throughput, Query duration • Amazon Redshift Query History (optimize queries) – Query Runtime, CPU utilization, Storage Capacity Used • Amazon Redshift Advisor Recommendations • Amazon Redshift Spectrum Queries – System views: SVL_S3QUERY, SVL_S3QUERY_SUMMARY • CloudWatch Metrics for Amazon EMR () – IsIdle, HDFS utilization, S3BytesRead, S3BytesWritten, MemoryTotalMB, MemoryAllocatedMB • CloudWatch Metrics for Amazon OpenSearch Service (Cluster Metrics) – CPUUtilization, FreeStorageSpace, ClusterUsedSpace, JVMMemoryPressure, DiskThroughputThrottle • CloudWatch Metrics for Amazon Athena – ProcessedBytes, QueryQueueTime, TotalExecutionTime • CloudWatch Metrics for Amazon SageMaker – CPUUtilization, GPUUtilization, GPUMemoryUtilization, MemoryUtilization, and DiskUtilization <p>Kinesis Data Analytics Application Metrics – CPUUtilization, containerCPUUtilization, containerMemoryUtilization, containerDiskUtilization, idleTimeMsPerSecond, heapMemoryUtilization</p>

Suggestion 15.2.3 – Review your data pipelines run frequency

Data pipelines are responsible for transferring data throughout your analytics platform. Working backwards from your business requirements, decide whether your organization can decrease the run frequency of the data pipelines.

Best practice 15.3 – Encourage a data minimization culture to reduce your organization's data impacts

Analytics relies heavily on large volumes of data being stored and processed. Minimizing the amount of data stored and processed can have a positive impact on your organization's analytics environmental impact. Encourage a data minimization or *just enough data* mindset in your organization's development team to reduce the overall amount of data processed and therefore reduce the amount of energy used.

As your data assets grow, your organization might implement an improvement process. Select target data assets for continual improvements, adopt successful improvements, and quantify the success of the improvement against your carbon metrics. This helps ensure that your organization does not unintentionally increase its analytics carbon footprint by allowing unrestricted data growth. As an example, your organization's data growth may increase 3x and your customer base increase 5x, which, typically speaking, means that your carbon trade-off is in credit.

More information on building and implementing an Improvement process can be found in the [Sustainability Pillar whitepaper](#).

How do you minimize the amount of data that is processed?

Suggestion 15.3.1 – Minimize the amount of data extracted from your source systems and supplied to your data warehouse environment

Data warehousing plays an important role in providing meaningful insights to your reporting layers and analytics. Data warehousing is the ingestion and merging of multiple data sources to create a single optimized data model, typically using the denormalized data modeling approach where is aggregated to provide faster query response times. It is encouraged that your organization applies these principles of building a data warehouse.

For more details, refer to the following information:

- Amazon Redshift development guide: [Database Developer Guide](#)

When designing your source data extraction processes, it is recommended that your organization should only extract data required for the data warehouse and onwards. This results in less data being transferred over the network, less data being loaded into the data warehouse, and lessens the requirement to remove unnecessary data from your data warehouse.

When extracting data from your source datastore, your organization should use a date range to extract only data that has been added or updated in the source datastore since the last data extract. This is called *delta updates*. This approach reduces the amount of static unchanged data shared between dependencies.

Designing and building an efficient data model requires upfront consideration. Your development team should ensure that the optimal row-level granularity (for example customer level, address level, or product level) and data attributes will reduce unnecessary deduplication and filtering further downstream.

Most reporting applications support data editing and data filtering capabilities. Therefore, your development teams can develop a subset of data within the business tool minimizing the amount of data required for a report refresh.

For more details, refer to the following information:

- Amazon QuickSight: [Creating datasets](#)

Suggestion 15.3.2 – Apply data type best practices when developing database tables

Data types are an attribute associated with data that tells a computer system how to interpret its value. Understanding data types ensures that data is collected in the preferred format and the value of each property is as expected. For example, if your data attribute contains a single binary value (1,0) then it would be inefficient to store the binary value in a CHAR(MAX) column because retrieving more bytes from storage becomes inefficient and expensive to process.

For more details, refer to the following information:

- Amazon Redshift best practices: [Amazon Redshift best practices for designing tables](#)
- Amazon Athena data types: [Data types in Amazon Athena](#)
- Amazon Redshift data types: [Data types](#)
- Amazon QuickSight: [Supported data types and values](#)

Suggestion 15.3.3 – Review your application APIs to understand whether all data must be shared with your streaming applications

APIs play an important role in connecting and sharing data between software applications, databases and other systems. Application developers should consider the size of an event payload submitted to these systems.

Organizations require the ability to run analytics on real-time data. To do so, organizations send data to streaming services. Streaming services, such as Amazon Managed Streaming for Apache Kafka (Amazon MSK) and Amazon Kinesis, allow organizations to run analytics on real-time streams of information. It is important that the data being shared with such streaming services is reviewed through the improvement process, because the more data provided in the payload will require more resources to store and process the data. Reducing the network, storage, and compute resources required to process unnecessary data can help towards reducing your organization's analytics environmental impact.

Review data that is captured by the application and pushed to the streaming platform to identify data attributes that can be removed. Also identify opportunities to store commonly used transforms to create values that can be computed once. Review your Kafka topic and identify if it's duplicated data of whether a single topic is enough to deliver to multiple dependencies. Through the Improvement process you should consider data volumes and the value of your assets, and measure these against your organization's proxy metrics.

If it is not possible to reduce data at the point of data capture, as a developer, you can use AWS Lambda to trim event payloads of data attributes that are not required for downstream processing. However, as an organization, you should balance the trade-off of compute cost of removing the data versus retaining the original data values. This is not a binary option but should be measured over time to determine if it would be worthwhile removing data.

For more details, refer to the following information:

- AWS Lambda: [Using AWS Lambda with Amazon Kinesis](#)

Implement a monitor and alert strategy to get a clear picture of data growth over time. Take action on any significant data growth by understanding what additional attributes have been added to the event payload. Alerts should be implemented on thresholds, such as 3x data growth, or create an internal metric that your organization should expect to increase the overall data footprint aligned with new customers.

For more details, refer to the following information:

- Amazon Kinesis: [Monitoring the Amazon Kinesis Data Streams Service with Amazon CloudWatch](#)

Suggestion 15.3.4 – Reduce the amount of data migrated from one environment to another

Migrating data from one environment to another is a common exercise. Your organization should consider data minimization when migrating from one environment to another as migration requires additional network, storage, and compute resources for migrating unwanted information. Your organization should regularly review all information that is in scope of the migration and determine whether it is necessary for future workloads, rather than defaulting to a *migrate all* approach.

If your organization maintains a data catalog, a review of the data assets by a Business Analyst (BA) prior to migrating the data should be performed to understand whether the data is required by the business.

For more details, refer to the following information:

- AWS Data Migration: [Top 10 Data Migration](#)
- AWS Data Migration (video): [Top 10 Data Migration Best Practices](#)

Suggestion 15.3.5 – Apply the optimal data model for your data access patterns

Understanding your data access patterns will help determine which data modeling technique is most suitable. There are many different data modeling techniques but this whitepaper focuses on the two most popular model: Normalization and Denormalization.

Normalization is the method of arranging the data in a data model to reduce redundant data and improve query efficiency. This method involves designing the tables and setting up relationships between those tables according to certain rules. The redundancy and inconsistent dependency can be removed using these rules to make it more flexible. Typically, normalized data models are used in online transaction processing (OLTP) and are supported by relational databases that store the database data in rows.

Denormalization is a technique used on a previously normalized OLTP database by removing complex primary and foreign key relationships and removing data redundancy. Denormalized data models are typically used in online analytical processing (OLAP) where the data is stored in columnar-oriented databases, such as Amazon Redshift. OLAP is designed for multidimensional analysis of data in a data warehouse, which contains both transactional and historical data. In computing, denormalization is the process of improving the read performance of a database that will benefit your organization's sustainability goals.

As a denormalized data model is a subset of your normalized data model, it generally performs better for data warehouse workloads. However, the sustainability trade-off is in the amount of compute required to convert your normalized data model into a denormalized data model and the additional data storage. The cost of transforming and aggregating many billions of rows is expensive, therefore, your organization should review regularly what attributes need transforming and remove any logic from those attributes that don't need processing.

A method of determining whether denormalization is a better approach to your workload is to review the number of join operations that you are using to return a result. Denormalization, in theory, reduces the number of joins, therefore, reducing the number of resources required to return results.

Regardless of the approach, understanding your applications and how your user base interact with your data stores is very important in designing the correct data model to use to reduce your organization's analytics environmental impact.

For more details, refer to the following information:

- Modern data architecture: [Build a modern data architecture on AWS with Amazon AppFlow, AWS Lake Formation, and Amazon Redshift](#)

Best practice 15.4 – Implement data retention processes to remove redundant data from your analytics environment

The retention of data should be informed, relevant, and limited to what is necessary for the purposes for which the data is processed. Storing data indefinitely and without purpose can cause significant storage and processing overhead that can impact your organization's analytics environmental impact. Ensure that the period for which the data should be stored is limited and reviewed on a regular basis.

How can you remove unnecessary data from an object store?

Suggestion 15.4.1 – Define and implement a data lifecycle process for data at rest (data lake)

Implement a lifecycle management process that will either remove data that is no longer required, or archive data into less resource-intensive storage.

When removing data from an object store, your organization should consider the following design points;

- The data retention removal process should run on a regular basis
- The data retention removal process should remove data from all buckets, sub-directories and prefixes.
- The data retention removal process should take an audit of what data has been removed, when it was removed, and who performed the removal process. This audit data should be tracked in an immutable audit log for auditing purposes.
- Production, user acceptance test (UAT), and development (DEV) environments must be included and adhere to the agreed retention policy across all environments.
- Consider other locations where data might be stored, such as SFTP locations.
- Classify your organization's data by data temperature, such as *hot* for frequently accessed, and *cold* for infrequently accessed. After data has been classified by temperature, your organization should implement a strategy to move data into the respective S3 bucket storage classes. For example, cold data could be moved to Amazon S3 Glacier storage class. For an illustration of data temperatures, see [Optimizing your AWS Infrastructure for Sustainability, Part II: Storage](#).

For more details, refer to the following information:

- Amazon S3 Lifecycle Management: [Managing your storage lifecycle](#)

How can you remove unnecessary data from a database?

Suggestion 15.4.2 – Remove unnecessary data from a database

To effectively remove information from a database, your organization should track when the data was loaded into the database and when the last customer interaction occurred, such as a purchase or other activity. This tracking helps you accurately identify when data should be removed.

- The data retention removal process should run frequently, but should not be run excessively, as excessive deletion can increase compute resources that could mitigate the benefit of removing the data from your database.

- The data retention removal process should remove data from all databases and tables.
- The data retention removal process should retain an audit of what data has been removed, when it was removed, and who performed the removal process. This audit data should be tracked in an immutable audit log for auditing purposes.
- If your database enforces referral integrity, you should redact only the data and retain the primary and foreign keys.

For more details, refer to the following information:

- Amazon Redshift: [Amazon Redshift Stored Procedures](#)
- Amazon Redshift: [DELETE Statement](#)
- Amazon Redshift: [Scheduling a query on the Amazon Redshift console](#)

Suggestion 15.4.3 – Develop a process to remove data from your streaming application when no longer required

The primary use-case of a streaming application is to transfer information from source to target, but they can also be used for storing historic data. Your organization might find that storing events in your streaming application is an efficient method, and replaying information into the target environment if information in that environment has become corrupted. However, the approach of using your streaming application as a data repository can lead to data becoming redundant and needing to be removed. Your organization should consider implementing a data retention policy to mitigate this concern.

For more details, refer to the following information:

- Amazon Kinesis: [Changing the Data Retention Period](#)
- Amazon Managed Streaming for Apache Kafka: [Adjust data retention parameters](#)

Best practice 15.5 – Optimize your data modeling and data storage for efficient data retrieval

How your data is organized in a data store, database, or file system can have an impact on the amount of resources that are required to store, process, and analyze the data. As developers, we can use development methods to improve the speed of data retrieval by enforcing specific encoding rules, or using indexes to optimize the efficiencies of the data seek and retrieval process.

How can your organization reduce the resources required to store, process, and analyze your organization's data in a sustainable manner?

Reducing data that a database system scans to return a result is an efficient way in reducing your organization's analytics environmental impact. This approach requires less resources to scan the disk to retrieve the information to service the request, and reduces the amount of provisioned storage required to service the workload. There are different methods that database engines use to optimize the amount of information scanned. In this section, we describe how Amazon Athena uses partitioning on Amazon S3 to reduce scanned data and how Amazon Redshift uses sort keys and distribution styles to do the same.

Suggestion 15.5.1 – Implement an efficient partitioning strategy for your data lake

Partitioning plays a crucial role when querying data in Amazon Athena or Amazon Redshift Spectrum since it limits the volume of data scanned resulting in dramatically accelerating queries and reducing query runtime. Folders, where data is stored in Amazon S3 and which are physical entities, are mapped to partitions in metastore, which are logical entities. Amazon Athena uses these partitions to retrieve

the list of folders that contain relevant data for a query. This partitioning list instructs Amazon Athena where to locate the data rather than searching all S3 buckets to retrieve the result, resulting in reducing compute resource and your organization's analytics environmental impact.

For more details, refer to the following information:

- Amazon S3 and Amazon Athena: [Bucketing vs partitioning](#)
- Amazon Athena: [Partitioning data in Amazon Athena](#)

Suggestion 15.5.2 – Configure distribution styles and sort keys in your Amazon Redshift database

Amazon Redshift sort keys determine the order in which rows in a table are stored on the disk. Query performance is improved when Amazon Redshift sort keys are properly used as it enables the query optimizer to read fewer blocks of data filtering out the majority of it. By reading fewer blocks of data, this approach can result in a reduction of compute resources required.

For more details, refer to the following information:

- Amazon Redshift: [Choose the best sort key](#)

Choosing the right distribution styles is crucial when it comes to optimizing Amazon Redshift performance, as the distribution style is how the data is distributed across the nodes in Amazon Redshift cluster. This means that the database engine can evenly distribute data across the nodes of the cluster and reduce the amount of computation resources required to return a result.

For more details, refer to the following information:

- Amazon Redshift: [Automate your Amazon Redshift performance tuning with automatic table optimization](#)
- Amazon Redshift: [Distribution styles](#)

Suggestion 15.5.3 – Enable results and query plan caching

It is advisable that if your data warehousing application supports results and query plan caching, then it should be enabled. By enabling results caching, your data warehouse will hold the results of a query in cache for faster data retrieval by not requiring the database engine to scan the disk to return the result. Most data engines support the ability to cache the query plan negating the need to access the query optimizer to generate the query plan, by returning the query plan from cache. This will save on compute resource and reduce the environmental impact.

For more details, refer to the following information:

- Amazon Redshift: [Performance optimization](#)

Suggestion 15.5.4 – Enable data compression to reduce storage resources

Your organization should consider compressing data in both object stores, such as Amazon S3, and if supported, in your organization's database systems. By compressing data, your organization is reducing the amount of storage and networking resources required for the workload. Database systems can decompress the data at a rate that is almost unnoticeable to the end user or application. As the data is compressed and then decompressed, this will also reduce the retrieval time of the database engine to fetch all the data from the storage array leading to a potential reduction in compute resources.

For more details, refer to the following information:

- **Amazon Redshift compression and encoding:** [Amazon Redshift Engineering's Advanced Table Design Playbook: Compression Encodings](#)
- **Amazon Redshift file compression parameter:** [File compression parameters](#)
- Amazon Redshift Compression: [Compression encodings](#)
- Amazon DynamoDB Compression: [Using data compression](#)
- Amazon Athena Compression Support: [Amazon Athena compression support](#)

Suggestion 15.5.5 – Format your information to reduce unnecessary storage and processing compute

There are many different data formats that your data can be structured in, such as CSV, JSON, ORC, or PARQUET. Each file format has been designed to overcome specific technical challenges, therefore, a single file format will most likely not meet all your requirements. To reduce storage throughput and computation overhead, your organization should test a number of data formats to understand what the most efficient data type is for your workload. More information on this can be found in the data lake optimization section.

Suggestion 15.5.6 – Avoid using unnecessary functions when returning results to the end user

Arranging your query results by an aggregate or in-column order can be costly due to the compute resources required. Try avoiding using functions such as ORDER BY when the results of your query don't need to be in order. Think about how the information must be represented and run the system accordingly, rather than using unnecessary compute.

Suggestion 15.5.7 – Prevent running costly FLATTEN operations on JSON objects

Increasingly, analysts are required to pull insights from semi-structured data such as JSON files. Flattening JSON objects to fit into a structured format can be costly as the FLATTEN function will scan and arrange the whole dataset. Your organization should consider ingesting JSON data into an Amazon Redshift SUPER data type field to support the ability to query JSON entities without the need to flatten the whole dataset.

For more details, refer to the following information:

- Amazon Redshift: [Ingesting and querying semi-structured data in Amazon Redshift](#)

Best practice 15.6 – Prevent unnecessary data movement between systems and applications

Moving data around your organization can be very costly as it requires compute, networking, and storage resources. This can be particularly costly for analytics workloads as they generally require large quantities of information. When businesses move data around their organization, they increase the risk of creating duplicate data, which can impact your storage resource.

How does your organization mitigate the unnecessary data movement from one part of your organization to another?

Suggestion 15.6.1 – Implement data virtualization techniques to query information where the data resides

Data virtualization plays an important role since it can reduce the amount of data that is copied and thus duplicated in your data access layer. In data virtualization, only the data that is required to service

the request is copied from the source location into the data virtualization layer and temporarily cached in memory. This data is then used to service the user's request. The sustainability trade-off with data virtualization is that data can be transferred between the data source and the data virtualization layer more times than if the dataset was copied (and duplicated) once daily. This approach can also lead to additional computation of results.

For more details, refer to the following information:

- Use Amazon Athena for data virtualization: [Amazon Athena](#)
- Running Presto and Trino on Amazon EMR: [Presto and Trino](#)

Suggestion 15.6.2 – Reduce the flow of data between application and database by implementing pushdown predicates

Predicate pushdown is important as it reduces the amount of data passed between a file system, database, and BI layers. Your database will evaluate filter predicates in the query against metadata stored in the storage files and determine if a filter is then applied when pulling the information from disk. In other words, pushing down parts of the query to where the data is stored. Predicate pushdowns can be used in services such as Amazon Athena where Amazon Athena will use the S3 storage metadata to support with filtering the data that is returned to the database engine.

For more details, refer to the following information:

- Use pushdown predicated with Amazon Athena: [Top 10 Performance Tuning Tips for Amazon Athena](#)
- Optimizing EMR Spark with leveraging pushdown predicates: [Optimize Spark performance](#)

Suggestion 15.6.3 – Prevent data movement by leveraging pre-calculated materialized views

A materialized view can reduce the amount of data shared between your data warehouse and reporting layers by pre-computing the results of a pre-defined query. Materialized views are especially useful for speeding up queries that are predictable and repeated. Instead of performing resource-intensive queries against large tables (such as aggregates or multiple joins), applications can query a materialized view and retrieve a precomputed result set, therefore, saving on compute resource and reducing an organization's analytics environmental impact.

For more details, refer to the following information:

- Amazon Redshift: [Creating materialized views in Amazon Redshift](#)

Suggestion 15.6.4 – Reduce the flow of data between an operational database and a data warehouse by using federated querying

A federated query enables your organization to directly query data stored in external databases that enables data analysts, engineers, and data scientists to perform SQL queries across data stored in relational, non-relational, object, and custom data sources. With federated querying, you can submit a single SQL query and analyze data from multiple sources running on premises or hosted in the cloud, which reduces data latency in reporting. Federated querying can reduce the amount of information shared between data stores, however, the sustainability trade-off is that your organization could transfer the same information multiple times rather than a once-off single bulk copy of all information on a daily basis. Your organization should frequently review your federated querying patterns to identify whether it's more sustainable to use federated query or single bulk copies. To do this, your organization could review the amount of data that has been queried in a week, versus calculating the size of a full extract, and implement the approach that processes the least amount of data.

For more details, refer to the following information:

- Amazon Redshift: [Querying data with federated queries in Amazon Redshift](#)
- Amazon Athena: [Using Amazon Athena Federated Query](#)

Suggestion 15.6.5 – Reduce the flow of data between an object store and a data warehouse by using Amazon Redshift Spectrum

Data temperatures can have a bearing on where your organization's data resides. Data can be stored in the database engine backed by high performance storage or on less energy intensive storage, such as object storage. If your data is split across both Amazon S3 and Amazon Redshift, then your organization can use Amazon Redshift Spectrum. Amazon Redshift Spectrum allows users to query external data within Amazon S3.

For more details, refer to the following information:

- Amazon Redshift: [Optimize data patterns using Amazon Redshift Data Sharing](#)
- Amazon Redshift: [Getting started with Amazon Redshift Spectrum](#)
- Amazon Redshift: [Querying external data using Amazon Redshift Spectrum](#)

Suggestion 15.6.6 – Decrease the amount of data shared between Amazon Redshift clusters by using data sharing

Amazon Redshift data sharing plays an important role in sharing the requirement to move and copy data between Amazon Redshift clusters. Data sharing allows the administrator to share databases, tables, and views with a consumer Amazon Redshift cluster. The consumer Amazon Redshift cluster can query live data where the data resides in the producer cluster, thus reducing data movement and data copying.

For more details, refer to the following information:

- Amazon Redshift: [Amazon Redshift Data Sharing](#)

Suggestion 15.6.7 – Decrease the amount of information shared across accounts by using AWS Lake Formation

The ability to share and control data across AWS accounts is important to gather a holistic view of department and operational insights. Limiting the amount of information shared across accounts can have a positive impact on reducing your analytics carbon footprint as it will reduce excessive data sharing and data duplication.

For more details, refer to the following information:

- AWS Lake Formation: [AWS Lake Formation](#)
- AWS Lake Formation: [Cross-account access in Lake Formation](#)

Best practice 15.7 – Efficiently manage your analytics infrastructure to reduce underutilized resources

Ensuring your organization has the correct amount of resource provisioned for your workload is a difficult and challenging task. The common approach for ensuring your organization has the sufficient number of resources available for unpredicted peaks is to overprovision your resources, however, this

approach generally leads to resources being underutilized thus consuming energy when they are not needed.

For a wider understanding of optimizing infrastructure for sustainability refer to the following information:

- Well-Architected Sustainability: [Optimizing your AWS Infrastructure for Sustainability, Part I: Compute](#)
- Well-Architected Sustainability: [Optimizing your AWS Infrastructure for Sustainability, Part II: Storage](#)

How does your organization ensure efficient infrastructure usage?

Suggestion 15.7.1 – Leverage Amazon EMR managed scaling

Implementing Amazon EMR managed scaling will automatically size cluster resources based on the workload for best performance. Amazon EMR managed scaling monitors key metrics, such as CPU and memory usage, and optimizes the cluster size for best resource utilization. As a focus of sustainability is reducing resources, Amazon EMR managed scaling helps ensure you have the right number of resources at the right time.

For more details, refer to the following information:

- Amazon EMR managed scaling: [Using EMR managed scaling in Amazon EMR](#)
- Amazon EMR managed scaling: [Introducing Amazon EMR-Managed Scaling – Automatically Resize Clusters to Lower Cost](#)

Suggestion 15.7.2 – Leverage EMRFS to provide your organization with flexible cluster management

EMR supports two types of file system, HDFS (Hadoop Distributed File System) and EMRFS (Elastic MapReduce File System). The main difference between them is that HDFS is attached to the EMR nodes whereby EMRFS is decoupled from the EMR cluster. By using EMRFS, you get all the benefits of Amazon S3, therefore you only pay for what you use and you need not overprovision to ensure your organization has enough spare storage for future EMR data requirements. This is important as less storage will reduce your organization's analytics environmental impact. Your organization can freely stop clusters because the data is stored in a cluster that is not attached to the nodes.

For more details, refer to the following information:

- Amazon EMR: [EMR File System \(EMRFS\)](#)

Suggestion 15.7.3 – Use Amazon Redshift resizing options to better manage your cluster resources

Being able to increase your database resources on demand allows you to operate with minimal infrastructure overhead, which reduces your overall analytics carbon footprint. Amazon Redshift supports two types of infrastructure resizing: Classic and Elastic resize. The different types of Amazon Redshift resizing can allow your organization to change the number of nodes and the node types. Elastic resizing is much more efficient and often requires less time to complete than a classic resize. To compare, classic resize operations provision a new cluster while copying over data from your source cluster. These are important trade-offs when considering resizing for sustainability purposes.

For more details, refer to the following information:

- Amazon Redshift cluster scaling: [How do I resize an Amazon Redshift cluster?](#)

Suggestion 15.7.4 – Stop and restart your Amazon Redshift cluster using the pause and resume feature

Using the pause and resume feature of Amazon Redshift can reduce the number of resources that power your cluster when the cluster is not in use. Your organization can then resume the Amazon Redshift cluster when there is a requirement to query the data. Your organization can use the pause and resume features to better manage resources throughout your ETL pipeline, for example, after the transformation steps of your ETL job have completed, your organization can pause the Amazon Redshift cluster that was responsible for performing the transformation. This is a good use of resources for ETL that can reduce your analytics carbon footprint.

For more details, refer to the following information:

- Amazon Redshift pause and resume: [Lower your costs with the new pause and resume actions on Amazon Redshift](#)
- Amazon Redshift pause and resume: [Pausing and resuming clusters](#)

Suggestion 15.7.5 – Use serverless for unpredictable workloads

Serverless is ideal when it is difficult to predict compute needs, such as with variable workloads, periodic workloads with idle time, and steady-state workloads with spikes. This is different from provisioned clusters as provisioned typically consume more resources. This approach is also a good fit for ad-hoc analytics that have to get started quickly and for test and development environments.

For more details, refer to the following information:

- Amazon Redshift Serverless: [Amazon Redshift Serverless](#)
- AWS-Managed Streaming for Apache Kafka Serverless: [Amazon MSK Serverless](#)
- Amazon EMR Serverless: [Amazon EMR Serverless](#)

Suggestion 15.7.6 – Leverage AWS Glue and AWS Glue DataBrew for data profiling and transformation

In the context of your data analytics pipeline, data transformation tasks can consume significant amounts of resources because the data must be transformed in a timely manner to ensure the data is of a format or quality that it can be used to drive insights. What does your organization do with the resources that have been used to transform data? Your organization should consider serverless technologies that decommission resources once they have completed their respective tasks.

For more details, refer to the following information:

- AWS Glue: [AWS Glue Overview](#)
- AWS DataBrew: [AWS DataBrew Overview](#)

Suggestion 15.7.7 – Leverage transient EMR cluster for data transform task

Transient clusters are an efficient way to reduce your static infrastructure footprint, which has a positive impact on meeting your sustainability goals. In the context of your data pipeline, an EMR cluster can be automatically provisioned to transform a dataset, and after the transform task has completed, the EMR cluster can be decommissioned. This approach can be performed for multiple pipeline and multiple different transformation tasks.

For more details, refer to the following information:

- Amazon EMR: [Launch a Spark job in a transient EMR cluster using a Lambda function](#)

Suggestion 15.7.8 – Run EMR on your Amazon Elastic Kubernetes Services (Amazon EKS) environment for optimal application infrastructure usage

Running your applications on correctly provisioned infrastructure is important to reduce your analytics carbon footprint. Amazon EMR on Amazon EKS supports the efficient usage of infrastructure when running EMR workloads, such as Spark jobs. Amazon EMR on Amazon EKS ensures that there is little or no additional infrastructure being used to run the Spark jobs because the application is containerization within the limits of your organization's Amazon EKS environment.

For more details, refer to the following information:

- Amazon EMR on Amazon EKS: [Amazon EMR on Amazon EKS](#)

Suggestion 15.7.9 – Process logs using an event driven solution

Event-driven architecture refers to a system of loosely coupled microservices that transfer data between each other. Typically, the data is in the form of a transaction such as an event or log. Loosely coupled microservices enable independent resource provisioning to ensure the right number of resources are available at every step of the event-driven solution. This approach provides your organization the ability to efficiently manage your workload resources therefore reduce your organization's analytics carbon footprint.

For more information, refer to the [event driven architecture blog](#)

Resources

Documentation and blogs

- AWS Customer Carbon Footprint: [AWS Customer Carbon Footprint Tool](#)
- Amazon QuickSight: [Creating datasets](#)
- Amazon Athena data types: [Data types in Amazon Athena](#)
- Amazon Redshift data types: [Data types](#)
- Amazon QuickSight: [Supported data types and values](#)
- Amazon QuickSight: [Using AWS Lambda with Amazon Kinesis](#)
- Amazon Kinesis: [Monitoring the Amazon Kinesis Data Streams Service with Amazon CloudWatch](#)
- AWS Data Migration: [Top 10 Data Migration](#)
- Amazon S3 Lifecycle Management: [Managing your storage lifecycle](#)
- Amazon Kinesis: [Changing the Data Retention Period](#)
- AWS-Managed Service Kafka: [Adjust data retention parameters](#)
- Amazon S3 and Amazon Athena: [Bucketing vs partitioning](#)
- Amazon Athena: [Partitioning data in Amazon Athena](#)
- Amazon Redshift development guide: [Database Developer Guide](#)
- Amazon Redshift: [Amazon Redshift Stored Procedures](#)
- Amazon Redshift: [DELETE Statement](#)
- Amazon Redshift: [Ingesting and querying semi-structured data in Amazon Redshift](#)
- Amazon Redshift data types: [Data types](#)
- Amazon Redshift: [Scheduling a query on the Amazon Redshift console](#)
- Amazon Redshift: [Choose the best sort key](#)
- Amazon Redshift Serverless: [Amazon Redshift Serverless](#)
- Amazon Redshift: [Automate your Amazon Redshift performance tuning with automatic table optimization](#)

- Amazon Redshift: [Distribution styles](#)
- Amazon Redshift: [Performance optimization](#)
- Amazon Redshift best practices: [Amazon Redshift best practices for designing tables](#)
- Amazon Redshift: [Getting started with Amazon Redshift Spectrum](#)
- Amazon Redshift: [Querying external data using Amazon Redshift Spectrum](#)
- Amazon Redshift file compression parameter: [File compression parameters](#)
- Amazon Redshift Compression: [Compression encodings](#)
- Amazon Redshift: [Creating materialized views in Amazon Redshift](#)
- Amazon Redshift: [Querying data with federated queries in Amazon Redshift](#)
- Amazon Redshift compression and encoding: [Amazon Redshift Engineering's Advanced Table Design Playbook: Compression Encodings](#)
- Modern data architecture: [Build a modern data architecture on AWS with Amazon AppFlow, AWS Lake Formation, and Amazon Redshift](#)
- Amazon DynamoDB Compression: [Using data compression](#)
- Amazon Athena Compression Support: [Amazon Athena compression support](#)
- Use Amazon Athena for data virtualization: [Amazon Athena](#)
- Running Presto and Trino on Amazon EMR: [Presto and Trino](#)
- Use pushdown predicated with Amazon Athena: [Top 10 Performance Tuning Tips for Amazon Athena](#)
- Optimizing EMR Spark with leveraging pushdown predicates: [Optimize Spark performance](#)
- Amazon Athena: [Using Amazon Athena Federated Query](#)
- EMR-Managed Scaling: [Using EMR-Managed scaling in Amazon EMR](#)
- EMR-Managed Scaling: [Introducing Amazon EMR-Managed Scaling – Automatically Resize Clusters to Lower Cost](#)
- Amazon EMR: [EMR File System \(EMRFS\)](#)
- Amazon Redshift cluster scaling: [How do I resize an Amazon Redshift cluster?](#)
- Amazon EMR on EKS: [Amazon EMR on Amazon EKS](#)
- Amazon EMR: [Launch a Spark job in a transient EMR cluster using a Lambda function](#)
-

Whitepapers

- Well-Architected Sustainability: [Optimizing your AWS Infrastructure for Sustainability, Part I: Compute](#)
- Well-Architected Sustainability: [Optimizing your AWS Infrastructure for Sustainability, Part II: Storage](#)

Demonstrations

- AWS Customer Carbon Footprint overview: [AWS Customer Carbon Footprint Tool Overview](#)
- AWS Data Migration (video): [Top 10 Data Migration Best Practices](#)

Scenarios

In this section, we cover the six key scenarios that are common in many analytics applications. We describe how they influence the design and architecture of your analytics environment in 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.

Scenarios

- [Data discovery \(p. 64\)](#)
- [Modern data architecture \(p. 67\)](#)
- [Batch data processing \(p. 71\)](#)
- [Streaming ingest and stream processing \(p. 75\)](#)
- [Operational analytics \(p. 81\)](#)
- [Data visualization \(p. 85\)](#)

Data discovery

Many organizations treat data like an organizational asset, meaning it is no longer the property of individual departments. You want to analyze all types of data to drive actionable insights, be prepared for the unexpected, create new revenue streams, improve customer experience, and increase operational efficiencies.

The data discovery process consists of a number of interactive sessions with various stakeholders within an organization. Sometimes this starts with an initial session to identify new ways to extract value from your data, while at other times, it could be with a specific use-case around what you want to do. You can go straight into identifying key people and diving deep to gather the information that is needed in order to determine your best possible solution.

In either case, the end goal is to maximize the value you get from the data and identify appropriate next steps. This includes how you plan to consume data, what data sources you have and how to ingest that data, and then potentially what types of transformations you may need for the data.

Characteristics

The common issue that can hold you back from maximizing the value of your data is the variety of data silos within your organization. Silos can prevent you from extracting maximum value from all your data with the greatest flexibility. Data warehouses can help with this to a point, but often only a small portion of raw data is brought into the data warehouse. Organizations often end up with multiple data warehouses, so you can still have these silos. There are a number of modern approaches to enterprise-wide analytics that can help solve this – such as data lakes, modern data architectures, and data mesh. If you are not already exploring these modern approaches, this is a good opportunity to learn more about these architectures in the following sections.

Another common issue is whether you can benefit from increasing the velocity at which you ingest and process your data. Many organizations still have a predominantly batch-oriented strategy to processing the data, where a majority of your data is processed on a daily schedule. You can ask yourself questions such as ‘How can we benefit if we have access to more up-to-date data?’ AWS can help you explore options for ingesting streaming data, and for processing data in micro-batches or employ stream processing.

If you have previously explored streaming options in the past, you might have been concerned about the complexity of some of these solutions, but there are many AWS-managed solutions for streaming that significantly reduces much of this complexity.

In general, data discovery consists of five steps:

1. Define the business value

This is the first step in data discovery where you define the business value or opportunity by conducting interactive sessions. Here are a few example questions to define the business opportunity.

- What insights are you getting from the data?
- How would getting insight into data provide value to the business?
- Are you looking to create a new revenue stream from your data?
- What are challenges with your current approach and tool?
- What are you not providing to your customers that you would like to provide?
- Who is the executive level stakeholder for this effort?
- Example-specific use case questions:
 - How does data define your customer acquisition strategy?
 - Would your business benefit from exploring modern approaches to managing fraud detection, predictive maintenance, customer 360, IoT, clickstream, operational analytics, root-cause analysis to reduce mean time to detection and mean time to recovery?
 - How are you continually innovating on behalf of your customers and improving their user experience?

2. Identify your user personas

In this step, you focus on your data consumers, such as business analysts, data engineers, data analysts, and data scientists. Once you have developed your user personas, enable them for purpose-built analytics and machine learning.

Here are few example questions to identify your data consumers.

- Who are the end users?
- What insights are you currently getting from your data?
- What insights are on your roadmap?
- Do you have a multi-tenant data model?
- What are the different consumption models?
 - Which tool or interface do your data consumers use?
 - How real time does the data need to be for this use case (for example, near real time, every 15 minutes, hourly, daily)?
 - What is the total number of consumers for this consumption model?
 - What is the peak concurrency?

3. Identify your data sources

In this step, you focus on your data sources and tools to bring that data into the data platform. This allows you to perform comprehensive analytics and machine learning from a wide variety of data from various data sources.

Data types and sources

Table 3: Typical data sources in an organization

Data type	Example data sources	
Structured data	ERP applications, CRM applications, ERP applications, CMS applications, SaaS applications, SAP applications, line of business (LOB) applications, and SQL databases	
Semi-structured data	Web applications, NoSQL databases, EDI (electronic data interchange), CSV, XML, and JSON documents	
Unstructured data	Video files, audio files, images, IoT data, sensors data, and invoices	
Batch	Internal applications generate structured data at regularly defined schedules	
Streaming data	Sensors, social media, video streams, IoT devices, mobile devices that generate semi-structured and unstructured data as continuous streams	

Here are a few example questions to identify your data consumers.

- How many data sources do you have to support?
 - Where and how is the data generated?
 - What are the different types of your data? (for example, structured, semi-structured, unstructured, batch, streaming)
 - What are the different formats of your data? (for example, JSON, CSV, FHIR)
 - Is your data originating from on premises, a third-party vendor, or the cloud?
 - Is the data source streaming, batch, or micro-batch?
 - What is the rate and volume of ingestion?
 - What is the ingestion interface (for example, API, SFTP, Amazon S3, AWS Marketplace)
- How does your team on-board new data sources?

4. Define your data storage, catalog, and data access needs

In this step, you focus on your data storage, data cataloging, security, compliance, and data access requirements.

Here are few example questions to identify your data storage and data access requirements.

- What data stores do you have?
- What is the purpose of each data store?
- Why that storage method? (for example, files, SQL, NoSQL, data warehouse)
- How do you currently organize your data? (for example, data tiering, partition)

- How much data are you storing now, and how much do you expect to be storing in the future, for example, 18 months from now?
- How do you manage data governance?
- What data regulatory and governance compliance do you face?
- What is your disaster recovery (DR) strategy?

5. Define your data processing requirements

In this step, you focus on your data processing requirements.

Here are few example questions to identify your data processing requirements.

- Do you have to transform or enrich the data before you consume it?
- What tools do you use for transforming your data?
- Do you have a visual editor for the transformation code?
- What is your frequency of data transformation? (for example, real time, micro-batching, overnight batch)
- Are there any constraints with your current tool of choice?

Modern data architecture

Organizations have been building data lakes to analyze massive amounts of data for deeper insights into their data. To do this, they bring data from multiple silos into their data lake, and then run analytics and AI/ML directly on it. It is also common for these organizations to have data stored in specialized data stores, such as a NoSQL database, a search service, or a data warehouse, to support different use cases. To analyze all of the data that is spread across the data lake and other data stores efficiently, businesses often move data in and out of the data lake and between these data stores. This data movement can get complex and messy as the data grows in these data stores.

To address this, businesses need a data architecture that allows building scalable, cost-effective data lakes. The architecture can also support simplified governance and data movement between various data stores. We refer to this as a *modern data architecture*. Modern data architecture integrates a data lake, a data warehouse, and other purpose-built data stores while enabling unified governance and seamless data movement.

As shown in the following diagram, with a modern data architecture, organizations can store their data in a data lake and use purpose-built data stores that work with the data lake. This approach allows access to all of the data to make better decisions with agility.

Modern Data Architecture

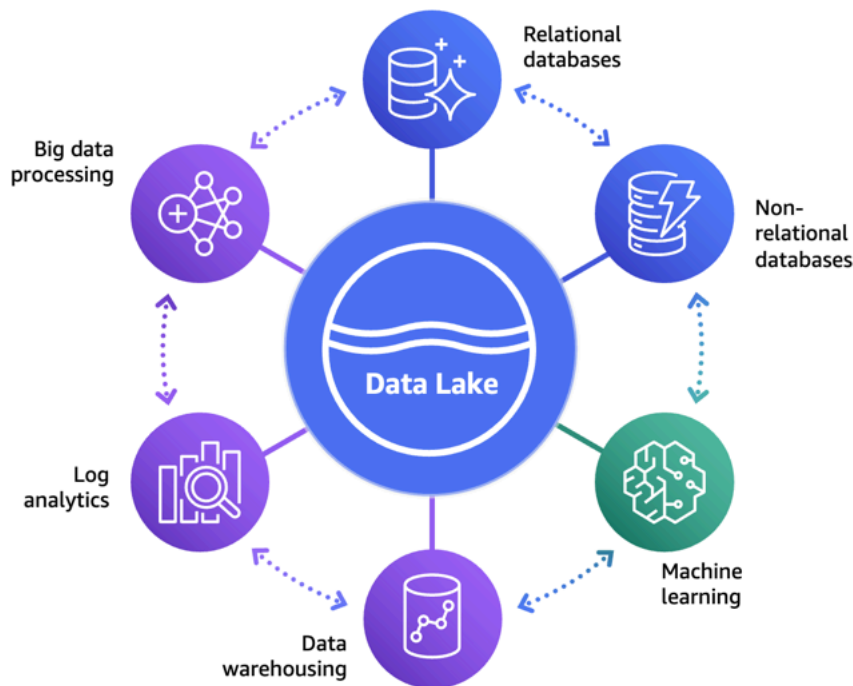


Figure 1: Modern data architecture

There are three different patterns for data movement. They can be described as follows:

Inside-out data movement: A subset of data in a data lake is sometimes moved to a data store, such as an Amazon OpenSearch Service cluster or an Amazon Neptune cluster. This pattern supports specialized analytics, such as search analytics, building knowledge graphs, or both. For example, enterprises send information from structured sources (such as relational databases), unstructured sources (such as metadata, media, or spreadsheets) and other assets to a data lake. From there, it is moved to Amazon Neptune to build a knowledge graph. We refer to this kind of data movement as inside-out.

Outside-in data movement: Organizations use data stores that best fit their applications and later move that data into a data lake for analytics. For example, to maintain game state, player data, session history, and leader boards, a gaming company might choose Amazon DynamoDB as the data store. This data can later be exported to a data lake for additional analytics to improve the gaming experience for their players. We refer to this kind of data movement as outside-in.

Around the perimeter: In addition to the two preceding patterns, there are scenarios where the data is moved from one specialized data store to another. For example, enterprises might copy customer profile data from their relational database to a NoSQL database to support their reporting dashboards. We refer to this kind of data movement as around the perimeter.

Characteristics

Scalable data lake: A data lake should be able to scale easily to petabytes and exabytes as data grows. Use a scalable, durable data store that provides the fastest performance at the lowest cost, supports multiple ways to bring data in, and has a good partner ecosystem.

Data diversity: Applications generate data in many formats. A data lake should support diverse data types—structured, semi-structured, or unstructured.

Schema management: A modern data architecture should support schema on read for a data lake with no strict source data requirement. The choice of storage structure, schema, ingestion frequency, and data quality should be left to the data producer. A data lake should also be able to incorporate changes to the structure of the incoming data that is referred to as schema evolution. In addition, schema enforcement helps businesses ensure data quality by preventing writes that do not match the schema.

Metadata management: Data should be self-discoverable with the ability to track lineage as data flows through tiers within the data lake. A comprehensive Data Catalog that captures the metadata and provides a queryable interface for all data assets is recommended.

Unified governance: A modern data architecture should have a robust mechanism for centralized authorization and auditing. Configuring access policies in the data lake and across all the data stores can be overly complex and error prone. Having a centralized location to define the policies and enforce them is critical to a secure modern data architecture.

Transactional semantics: In a data lake, data is often ingested nearly continuously from multiple sources and is queried concurrently by multiple analytic engines. Having atomic, consistent, isolated, and durable (ACID) transactions is pivotal to keeping data consistent.

Reference architecture

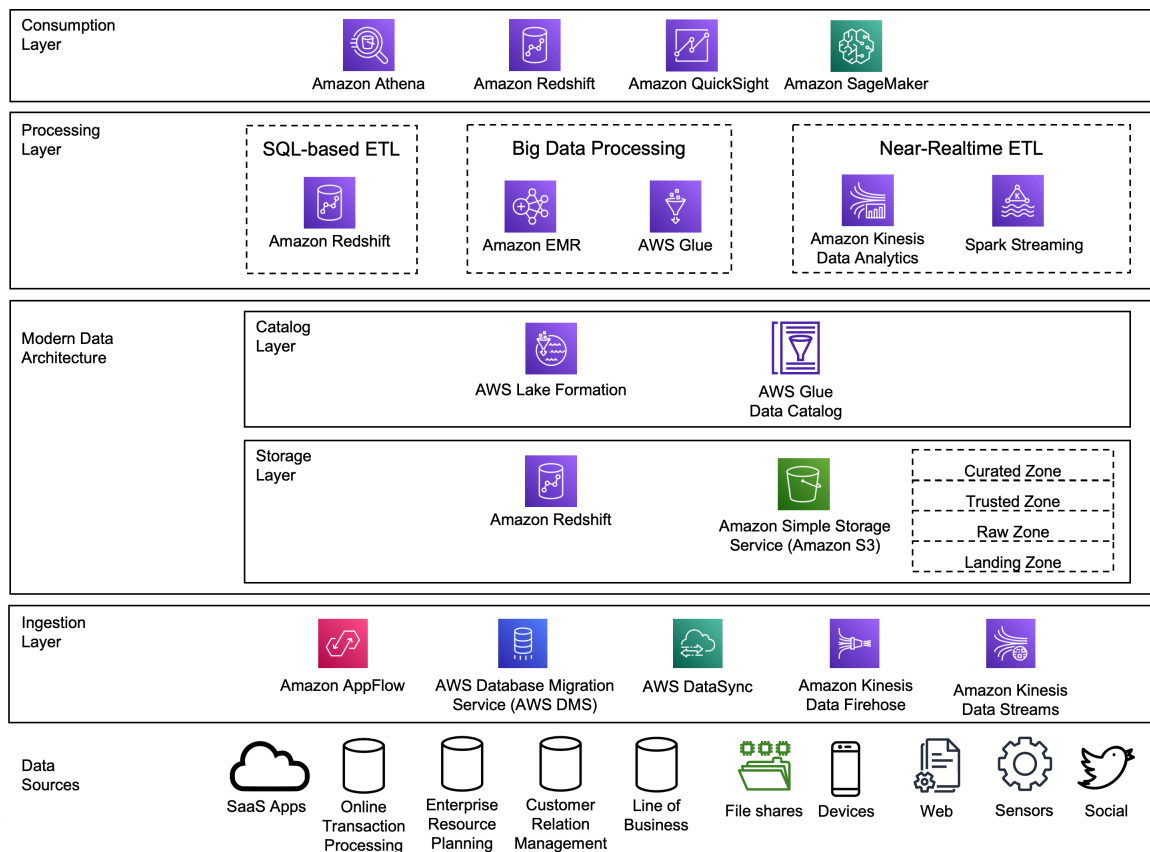


Figure 2: Modern data architecture reference architecture

Configuration notes

- To organize data for efficient access and easy management:
 - The storage layer can store data in different states of consumption readiness, including raw, trusted, conformed, enriched, and modeled. It's important to segment your data lake into landing, raw, trusted, and curated zones to store data depending on its consumption readiness. Typically, data is ingested and stored as is in the data lake (without having to first define schema) to accelerate ingestion and reduce time needed for preparation before data can be explored.
 - Partition data with keys that align to common query criteria.
 - Convert data to an open columnar file format, and apply compression. This will lower storage usage, and increase query performance.
- Choose the proper storage tier based on data temperature. Establish a data lifecycle policy to delete old data automatically to meet your data retention requirements.
- Decide on a location for data lake ingestion, for example, an S3 bucket. Select a frequency and isolation mechanism that meet your business needs.
- Depending on your ingestion frequency and data mutation rate, schedule file compaction to maintain optimal performance.
- Use AWS Glue crawlers to discover new datasets, track lineage, and avoid a data swamp.
- Manage access control and security using AWS Lake Formation, IAM role setting, AWS KMS, and AWS CloudTrail.
- There is no need to move data between a data lake and the data warehouse for the data warehouse to access it. Amazon Redshift Spectrum can directly access the dataset in the data lake.
- For more details, refer to the [Derive Insights from AWS Modern Data](#) whitepaper.

User personas

To get the full value from your modern data architecture, there are various personas who will access the data and perform data analytics. For example, the chief data officer (CDO) of an organization is responsible for driving digital innovation and transformation across lines of business. This CDO should set a data-driven vision for the organization and be a champion of using data, analytics, and AI/ML to inform business decisions.

Table 4: Key personas for a modern data architecture

Personas	Responsibility	Areas of interest	Modern data architecture purpose-built AWS services
Chief data officer (CDO)	Build a culture of using data to solve problems and accelerate innovation.	Data quality, data governance, data and AI strategy, evangelize the value of data to the business.	AWS Lake Formation, Amazon OpenSearch Service
Data architect	Driven to architect technical solutions to meet business needs. Focuses on solving complex data challenges to help the	Data pipeline, data processing, data integration, data governance, and data catalogs.	AWS Glue, Amazon EMR, Amazon Redshift, Amazon Athena, Amazon OpenSearch Service

	CDO deliver on their vision.		
Data engineer	Deliver usable, accurate dataset to organization in a secure and performant manner.	Variety of tools to build data pipeline, ease of use, configuration, and maintenance.	AWS Glue, Amazon EMR, Amazon Kinesis, Amazon Redshift, Amazon Athena, Amazon OpenSearch Service
Data security officer	Data security, privacy, and governance must be strictly defined and adhered to.	Keeping information secure. Comply with data privacy regulations and protecting personally identifiable information (PII), applying fine-grained access controls and data masking.	AWS Lake Formation, AWS Identity and Access Management (IAM).
Data scientist	Construct the means for extracting business-focused insight from data quickly for the business to make better decision.	Tools that simplify data manipulation, and provide deeper insight than visualization tools. Tools that help build the ML pipeline.	Amazon SageMaker, Amazon Athena, Amazon QuickSight, AWS Glue Studio, AWS Glue DataBrew
Data analyst	React to market conditions in real time, must have the ability to find data and perform analytics quickly and easily.	Querying data and performing analysis to create new business insights, producing reports and visualizations that explain the business insights.	Amazon Athena, Amazon QuickSight, AWS Glue Studio, Amazon Redshift

Batch data processing

Most analytics applications require frequent batch processing that allows them to process data in batches at varying intervals. For example, processing daily sales aggregations by individual store and then writing that data to the data warehouse on a nightly basis can allow business intelligence (BI) reporting queries to run faster. Batch systems must be built to scale for all sizes of data and to scale seamlessly to the size of the dataset being processed by various job runs.

It is important for the batch processing system to be able to support disparate source and target systems. These include processing various data formats, seamlessly scaling out to process peak data volumes, orchestrating jobs using workflow, providing a simple way to monitor the jobs, and most importantly offering an ease-of-use development framework that accelerates job development. Business requirements might dictate that batch data processing jobs be bound by an SLA, or have certain budget thresholds. Use these requirements to determine the characteristics of the batch processing architecture.

On AWS, analytic services such as Amazon EMR, Amazon Redshift, Lake Formation blueprints, and [AWS Glue](#) family services, namely Glue ETL, [Glue Workflows](#), and [AWS Glue DataBrew](#) allow you to run batch data processing jobs at scale for all batch data processing use cases and for various personas. These personas include data engineers, data analysts, and data scientists. While there are some overlapping

capabilities between these services, knowing the core competencies and when to use which service or services allows you to accomplish your objectives in the most effective way.

Characteristics

Following are the key characteristics that determine how you should plan when developing a batch processing architecture.

Ease of use development framework: This is one of the most important characteristics that allows personas of ETL developers and data engineers, data analysts, and data scientists to improve their overall efficiencies. An ETL developer benefits from a hybrid development interface that helps them to use the best of both—developing part of their job and switching to writing customized complex code where applicable. Data analysts and data scientists spend much of their time preparing data for actual analysis, or capturing feature engineering data for their machine learning models. You can improve their efficiencies in data preparation by adopting a no-code data preparation interface. This helps them normalize and clean data up to 80% faster compared to traditional approaches to data preparation.

Support disparate source and target systems: Your batch processing system should support different types of data sources and targets between relational, semi-structured, non-relational, and SaaS providers. When operating in the cloud, a connector ecosystem can benefit you by seamlessly connecting to various sources and targets, and can simplify your job development.

Support various data file formats: Some of the commonly seen data formats are CSV, Excel, JSON, Apache Parquet, Apache ORC, XML, and Logstash Grok. Your job development can be accelerated and simplified if the batch processing services can natively profile these various file formats, and infer schema automatically (including complex nested structures) so that you can focus more on building transformations.

Seamlessly scale out to process peak data volumes: Most batch processing jobs experience varying data volumes. Your batch processing job should scale out to handle peak data spikes and scale back in when the job completes.

Simplified job orchestration with job bookmarking capability: The ability to develop job orchestration with dependency management, and the ability to author the workflow using API, CLI, and a graphical user interface allows for a robust CI/CD integration.

Ability to monitor and alert on job failure: This is an important measure for ease of operational management. Having quick and easy access to job logs, and a graphical monitoring interface to access job metrics can help you identify errors and tuning opportunities quickly for your job. Coupling that with an event-driven approach to alert on job failure will be invaluable for easing operational management.

Provide a low-cost solution: Costs can quickly get out of control if you do not plan correctly. A pay-as-you-go pricing model for both compute and authoring jobs can help you overcome hefty costs upfront and allows you to pay only for what you use instead of overpaying to accommodate for peak workloads. Use automatic scaling to accommodate spiky workloads when necessary. Using Spot Instances where applicable can bring your costs down for workloads where they are a good fit.

Reference architecture

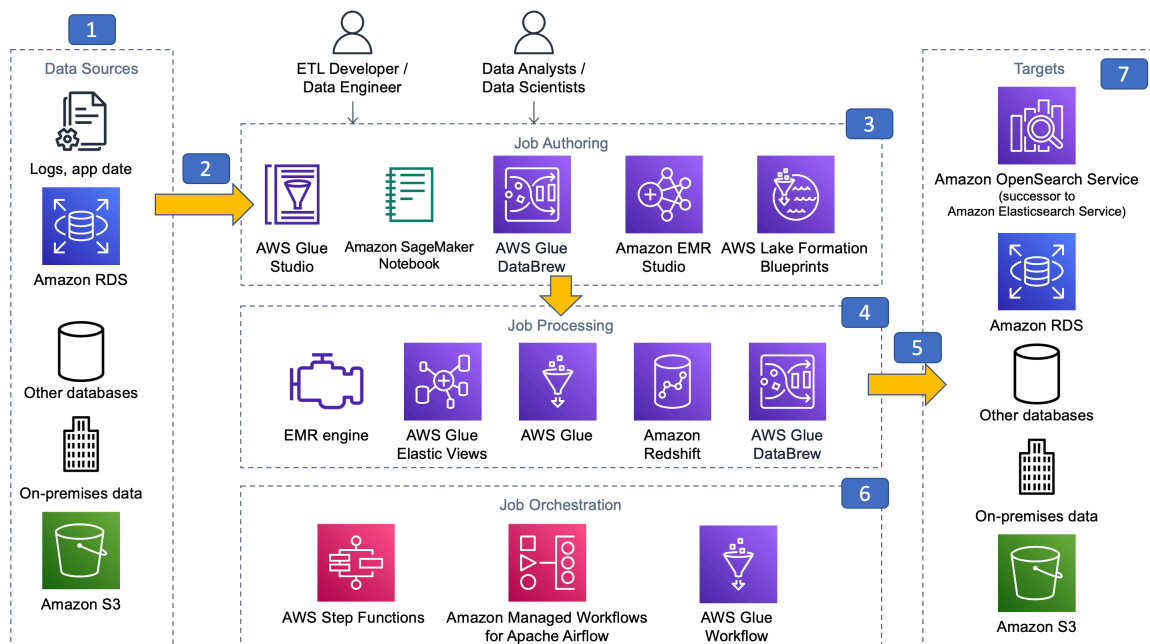


Figure 3: Batch data processing reference architecture

1. Batch data processing systems typically require a persistent data store for source data. When developing batch data processing applications on AWS, you can use data from various sources, including your on-premises data stores, Amazon RDS, Amazon S3, DynamoDB, and any other databases that are accessible in the cloud.
2. Data processing jobs need access to a variety of data stores to read data. You can use AWS Glue connectors from the AWS Marketplace to connect to a variety of data stores, such as Google BigQuery, and SAP HANA. You also can connect to SaaS application providers, such as Salesforce, ServiceNow, and Google Analytics, using AWS Glue DataBrew and Amazon AppFlow. In addition, you can always rely on the custom JDBC capability in Apache Spark and connect to any JDBC-compliant data store from Amazon EMR or AWS Glue jobs.
3. Choosing the right authoring tool for the job simplifies job development and improves agility.
 - a. You can use AWS Glue Studio or Glue interactive sessions when authoring jobs for the AWS Glue Spark runtime engine.
 - b. Use AWS Glue blueprints when you create a self-service parametrized job for analysts and control what data the analyst is allowed to access.
 - c. Use Amazon EMR notebooks for interactive job development and scheduling notebook jobs against Amazon EMR.
 - d. Use Amazon SageMaker notebook when working within SageMaker development and pre-processing data using Spark on EMR.
 - e. Use AWS Glue DataBrew from the AWS Management Console or from a Jupyter notebook for no-code development experience.
 - f. Use Lake Formation blueprints to quickly create batch data ingestion jobs to rapidly build a data lake in AWS.
4. Choosing the right processing engine for your batch jobs allows you to be flexible with managing costs and lowering operational overhead. Amazon EMR, AWS Glue (Streaming) ETL and Amazon Redshift offer the ability to scale seamlessly based on your job runtime metrics using managed scaling, automatic scaling, and concurrency scaling features for read and write, respectively. Amazon

EMR and Amazon Redshift offer both server-based and serverless architectures while the other services depicted in the reference architecture are fully serverless. Amazon EMR (server-based) allows you to use Spot Instances for suitable workloads that can further save your costs. A good strategy is to complement these processing engines to meet the business objectives of the SLA, functionality, and lower TCO by choosing the right engine for the right job.

5. Batch processing jobs usually require writing processed data to a target persistent store. This store can reside anywhere between AWS, on-premises environments, or other cloud providers. You can use the rich connector interface AWS Glue offers to write data to various target platforms, such as Amazon S3, Snowflake, and Amazon OpenSearch Service. You can also use the native Spark JDBC connector feature and write data to any supported JDBC target.
6. All batch jobs require a workflow that can handle dependency checks to ensure no downstream impacts and have a bookmarking capability that allows them to resume where they left off in the event of a failure or at the next run of the job. When using AWS Glue as your batch job processing engine, you can use the native workflow capability to help you create a workflow with a built-in state machine to track the state of your job across the entire workflow. AWS Glue jobs also support a bookmarking capability that keeps track of what it has processed and what will be processed during next run. Similarly, AWS Lake Formation blueprints support a bookmarking capability when processing incremental data. With Amazon EMR Studio, you can schedule notebook jobs. When using any of the analytic processing engines, you can build job workflows using an external scheduler, such as AWS Step Functions or Amazon Managed Workflows for Apache Airflow (Amazon MWAA) that allows you to interoperate between any service including external dependencies.
7. Batch processing jobs write data output to a target data store, which can be anywhere in the AWS Cloud, on premises, or at another cloud provider. You can use the AWS AWS Glue Data Catalog to crawl the supported target databases to simplify writing to your target database.

Configuration notes

Use AWS AWS Glue Data Catalog as a central metastore for your batch processing jobs, regardless of which AWS analytics service you use as a processing engine. Batch processing jobs cater to a variety of workloads ranging from running several times an hour or day, to running monthly or quarterly. The data volumes vary significantly and so do the consumption patterns on the processed dataset. Always work backwards to understand the business SLAs and develop your job accordingly. The central Data Catalog makes it easy for you to use the right analytic service to meet your business SLAs and other objectives, thereby creating a central analytic ecosystem.

Avoid lifting and shifting server-based batch processing systems to AWS. By lifting and shifting traditional batch processing systems into AWS, you risk running overprovisioned resources on Amazon EC2. For example, traditional Hadoop clusters are often overprovisioned and idle in an on-premises setting. Use AWS Managed Services, such as AWS Glue, Amazon EMR, and Amazon Redshift, to simplify your architecture using a modern data architecture pattern and remove the undifferentiated heavy lifting of managing clustered and distributed environments.

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 use automation and orchestration for your batch data processing jobs in conjunction with the AWS APIs to spin up and tear down entire compute environments, 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 shut down the cluster after the job is complete. Similarly, you can use Terraform or a CloudFormation template to achieve similar functionality.

Use Spot Instances and Graviton-based instance types on EMR to save costs and get better price performance ratio. Use Spot Instances when you have flexible SLAs that are resilient to job reruns upon failure and when there is a need to process very large volumes of data. Use Spot Fleet, EC2 Fleet, and Spot Instance features in Amazon EMR to manage Spot Instances.

Continually monitor and improve batch processing jobs. 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

Processing real time streaming data requires throughput scalability, reliability, high availability, and low latency to support a variety of applications and workloads. Some examples include: streaming ETL, real-time analytics, fraud detection, API microservices integration, fraud detection activity tracking, real-time inventory and recommendations, and click-stream, log file, and IoT device analysis.

Streaming data architectures are built on five core constructs: data sources, stream ingestion, stream storage, stream processing, and destinations. Each of these components can be created and launched using AWS Managed Services and deployed and managed as a purpose-built solution on Amazon EC2, Amazon Elastic Container Service (Amazon ECS), or Amazon Elastic Kubernetes Service (Amazon EKS).

Examples of each of these components include:

Data sources: Application and click stream logs, mobile apps, existing transactional relational and NoSQL databases, IoT sensors, and metering devices.

Stream ingestion and producers: Both open source and proprietary toolkits, libraries, and SDKs for Kinesis Data Streams and Apache Kafka to create custom stream producers, AWS service integrations such as AWS IoT Core, CloudWatch Logs and Events, Amazon Kinesis Data Firehose, AWS Data Migration Service (DMS), and third-party integrations.

Stream storage: Kinesis Data Streams, Amazon Managed Streaming for Apache Kafka (Amazon MSK), and Apache Kafka.

Stream processing and consumers: Amazon EMR (Spark Structured Streaming, Apache Flink), AWS Glue ETL Streaming, Kinesis Data Analytics for Apache Flink, third-party integrations, and build-your-own custom applications using AWS and open source community SDKs and libraries.

Downstream destinations: Databases, data warehouses, purpose-built systems such as OpenSearch services, data lakes, and various third-party integrations.

With these five components in mind, next let's consider the characteristics as you design your stream processing pipeline for real-time ingestion and nearly continuous stream processing.

Characteristics

Scalable throughput: For real-time analytics, you should plan a resilient stream storage 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.

Dynamic stream processor consumption and collaboration: Stream processors and consumers should automatically discover newly added Kinesis shards or Kafka partitions, and distribute them equitably across all available resources to process independently or collaboratively as a consumption group (Kinesis Application Name, Kafka Consumer Group).

Durable: Real-time streaming systems should provide high availability and data durability. For example, Amazon Kinesis Data Streams and Amazon Managed Streaming for Apache Kafka (Amazon MSK) replicate data across Availability Zones providing the high durability that streaming applications need.

Replay-ability: Stream storage systems should provide the ordering of records within shards and partitions, as well as the ability to independently read or replay records in the same order to stream processors and consumers.

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 consuming application crashes, it can resume reading the stream from that point instead of having to start at the beginning.

Loosely coupled integration: A key benefit of streaming applications is the construct of loose coupling. The value of loose coupling is the ability of stream ingestion, stream producers, stream processors, and stream consumers to act and behave independently of one another. Examples include the ability to scale consumers outside of the producer configuration and adding additional stream processors and consumers to receive from the same stream or topic as existing stream processors and consumers, but perform different actions.

Allow 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 might 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 can choose a message delivery system that supports one or more of these required semantics.

Security: Streaming ingest and processing systems must be secure by default. You must grant access by using the principal of least privilege to the streaming APIs and infrastructure, and encrypt data at rest and in transit. Both Kinesis Data Streams and Amazon MSK can be configured to use IAM policies to grant least privilege access. For stream storage in particular, allow encryption in transit for producers and consumers, and encryption at rest.

Reference architecture

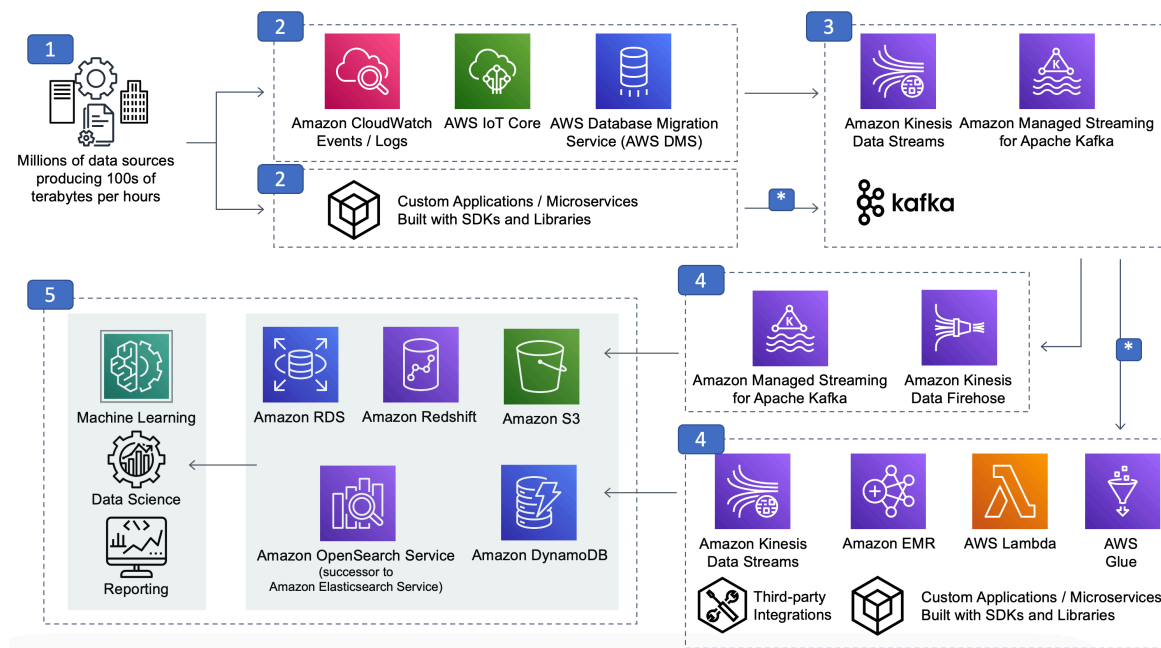


Figure 4: Streaming data analytics reference architecture

The preceding streaming reference architecture diagram is segmented into the previously described components of streaming scenarios:

- Data sources
- Stream ingestion and producers
- Stream storage
- Stream processing and consumers
- Downstream destinations

All, or portions, of this reference architecture can be used for workloads such as application modernization with microservices, streaming ETL, ingest, real-time inventory, recommendations, or fraud detection.

In this section, we will identify each layer of components shown in the preceding diagram with specific examples. The examples are not intended to be an exhaustive list, but rather an attempt to describe some of the more popular options.

The subsequent Configuration notes section provides recommendations and considerations when implementing streaming data scenarios.

We will review the five core components of streaming architecture first, and then discuss these specialized flows.

- 1. Data sources:** The number of potential data sources is in the millions. Examples include application logs, mobile apps and applications with REST APIs, IoT sensors, existing application databases (RDBMS, NoSQL) and metering records.
- 2. Stream ingestion and producers:** Multiple data sources generate data continually that might amount to terabytes of data per day. Toolkits, libraries, and SDKs can be used to develop custom stream

producers to streaming storage. In contrast to custom developed producers, examples of pre-built producers include Kinesis Agent, Change Data Capture (CDC) solutions, and Kafka Connect Source connectors.

3. **Streaming storage:** Kinesis Data Streams, Amazon MSK, and self-managed Apache Kafka are all examples of stream storage options for ingesting, processing, and storing large streams of data records and events. Streaming storage implementations are modeled on the idea of a distributed, immutable commit log. Events are stored for a configurable duration (hours to days to months, or even permanently in some cases). While stored, events are available to any client.
4. **Stream processing and consumers:** 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. Or, put another way, this can be where particular domain-specific logic resides and is computed.

With Kinesis Data Analytics for Apache Flink or Kinesis Data Analytics Studio, you can process and analyze streaming data using standard SQL in a serverless way. The service allows you to quickly author and run SQL queries against streaming sources to perform time series analytics, feed real-time dashboards, and create real-time metrics.

If you work in an Amazon EMR environment, you can process streaming data using multiple options—Apache Flink or Spark Structured Streaming.

Finally, there are options for AWS Lambda, third-party integrations, and build-your-own custom applications using AWS SDKs, libraries, and open-source libraries and connectors for consuming from Kinesis Data Streams, Amazon MSK, and Apache Kafka.

5. **Downstream destinations:** Data can be persisted to durable storage to serve a variety of use cases including ad hoc analytics and search, machine learning, alerts, data science experiments, and additional custom actions.

A special note on the data flow lanes noted with asterisk (*). There are two examples that both involve bidirectional flow of data to and from layer #3 streaming storage.

The first example is the bidirectional flow of in-stream ETL between stream processor (#4) that uses one or more raw event sources from stream storage (#3) and performs, filtering, aggregations, joins, etc., and writes results back to streaming storage to a refined (that is, curated, hydrated) result stream or topic (#3) where it can be used by a different stream processor or downstream consumer.

The second bidirectional flow example is the ubiquitous application modernization microservice design (#2) that often use a streaming storage layer (#3) for decoupled microservice interaction.

The key takeaway here is for us to not presume that the streaming event flows exclusively from left-to-right over time in the reference architecture diagram.

Configuration notes

As explored so far, we know streaming data architects have options for implementing particular components in their stack, for example, different options for streaming storage, streaming ingest, and streaming producers. While it's impractical to provide in-depth recommendations for each layer's options in this whitepaper, there are some high-level concepts to consider as guide posts, which we will present next.

For more in-depth analysis of a particular layer in your design, consider exploring the provided links within the following guidelines.

Streaming application guidelines

Determine business requirements first. It's always a best practice and practical to focus on your workload's particular needs first, rather than starting with a feature-by-feature comparison between the

technical options. For example, we often see organizations prioritizing Technical Feature A vs. Technical Feature B before determining their workload's requirements. This is the wrong order. Determine your workload's requirements first because AWS has a wide variety of purpose-built options at each streaming architecture layer to best match your requirements.

Technical comparisons second. After business requirements have been clearly established, the next step is to match your business requirements with the technical options that offer the best chance for success. For example, if your team has few technical operators, serverless might be a good option.

Other technical questions about your workload might be whether you require a large number of independent stream processors and consuming applications, that is, one vs. many stream processors and consumers. What kind of manual or automatic scaling options are available to match business requirement throughput, latency, SLA, RPO, and RTO objectives? Is there a desire to use open source-based solutions? What are the security options and how well do they integrate into existing security postures? Is one path easier or more straightforward to migrate to versus another, for example, self-managed Apache Kafka to Amazon MSK.

To learn more about your options for various layers in the reference architecture stack, refer to the following:

- **Streaming ingest and producers** — Can be workload-dependent and use AWS service integrations, such as AWS IoT Core, CloudWatch Logs and Events, AWS Data Migration Service (AWS DMS), and third-party integrations (Refer to [Writing Data to Amazon Kinesis Data Streams in Amazon Kinesis](#) in the *Amazon Kinesis Data Streams Developer Guide*).
- **Streaming storage** — Kinesis Data Streams, Kinesis Data Firehose, Amazon MSK, and Apache Kafka (Refer to [Best Practices in Amazon Managed](#) in the *Amazon Managed Streaming for Apache Kafka Developer Guide*).
- **Stream processing and consumers** — Kinesis Data Analytics for Apache Flink, Kinesis Data Firehose, AWS Lambda, open source and proprietary SDKs (Refer to [Advanced Topics for Amazon Kinesis Data Streams Consumers in Amazon Kinesis Data Streams Developer Guide](#) and [Best Practices for Kinesis Data Analytics for Apache Flink in the Amazon Kinesis Data Analytics Developer Guide](#)).

For more information, refer to the [Build Modern Data Streaming Architectures on AWS](#) whitepaper.

Remember separation of concerns. Separation of concerns is the application design principle that promotes segmenting an application into distinct, particular area of concerns. For example, your application might require that stream processors and consumers are performing an aggregation computation in addition to recording the computation results to a downstream destination. While it can be tempting to clump both of these concerns into one stream processors or consumers, it is recommended to consider separation instead. It's often better in the segment isolate into multiple stream processors or consumers for operation monitoring, performance tuning isolation, and reducing the downtime blast radius.

Development

Existing or desired skills match. Realizing value from streaming architectures can be difficult and often a new endeavor for various roles within organizations. Increase your chances of success by aligning your team's existing skillsets, or desired skillsets, wherever possible. For example, is your team familiar with Java or do they prefer a different language, such as Python or Go? Does your team prefer a graphical user interface for writing and deploying code? Work backwards from your existing skill resources and preferences to appropriate options for each component.

Build vs. buy (Write your own or use off-the-shelf). Consider whether an integration between components already exists or if you must write your own. Or perhaps both options are available. Many teams new to streaming incorrectly assume that everything must be written from scratch. Instead, consider services such as Kafka Connect Connectors for inbound and outbound traffic, AWS Lambda, and Kinesis Data Firehose.

Performance

Aggregate records before sending to stream storage for increased throughput. When using Kinesis, Amazon MSK, or Kafka, ensure that the messages are accumulated on the producer side before sending to stream storage. This is also referred to as batching records to increase throughput, but at the cost of increased latency.

When working with Kinesis Data Streams, use Kinesis Client Library (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.

Initial planning and adjustment of shards and partitions. The most common mechanism to scale stream storage for stream processors and consumers is through the number of configured shards (Kinesis Data Streams) or partitions (Apache Kafka, Amazon MSK) for a particular stream. This is a common element across Kinesis Data Streams, Amazon MSK, and Apache Kafka, but options for scaling out (and in) the number of shards or partitions vary.

- Amazon Kinesis Data Streams Developer Guide: [Resharding a Stream](#)
- Apache Kafka Documentation - Operations: [Expanding your cluster](#) (also applicable to Amazon MSK)
- Amazon Managed Streaming for Apache Kafka Developer Guide: [Using LinkedIn's Cruise Control for Apache Kafka with Amazon MSK](#) (for partition rebalancing)

Use Spot Instances and automatic scaling to process streaming data cost effectively. You can also process the data using AWS Lambda with Kinesis, Amazon MSK, or both, and Kinesis record aggregation and de-aggregation modules for AWS Lambda. Various AWS services offer automatic scaling options to keep costs lower than provisioning for peak volumes.

Operations

Monitor Kinesis Data Streams and Amazon MSK metrics using Amazon CloudWatch. You can get basic stream and topic level metrics in addition to shard and partition level metrics. Amazon MSK also provides an [Open Monitoring with Prometheus](#) option.

- Amazon Kinesis Data Streams Developer Guide: [Monitoring Amazon Kinesis Data Streams](#)
- Amazon Managed Streaming for Apache Kafka Developer Guide: [Monitoring an Amazon MSK Cluster](#)

Plan for the unexpected / No single point of failure. Some components in your streaming architecture will offer different options for durability in case of a failure. For example, Kinesis Data Streams replicates to three different Availability Zones. With Apache Kafka and Amazon MSK, producers can be configured to require acknowledgement for partition leader as well as a configurable number of in-sync replica followers before considering the write successful. In these examples, you are able to plan for possible disruptions in your AWS environment, for example, if an Availability Zone goes offline, without possible downtime of your producing and consuming layers.

Security

Authentication and authorization.

- Amazon Managed Streaming for Apache Kafka Developer Guide: [Authentication and Authorization for Apache Kafka APIs](#)
- Amazon Kinesis Data Streams Developer Guide: [Controlling Access to Amazon Kinesis Data Streams Resources Using IAM](#)

Encryption in transit and encryption at rest. Streaming data actively moves from one layer to another, such as from a streaming data producer to stream storage over the internet or through a private network.

Protecting data in transit, enterprises can and often choose to use encrypted connections (HTTPS, SSL, TLS) to protect the contents of data in transit. Many AWS streaming services offer protection of data at rest through encryption.

- AWS Well-Architected Framework Security Pillar: [AWS Identity and Access Management](#)
- AWS Lake Formation Developer Guide: [Security in AWS Lake Formation](#)

Operational analytics

Operational analytics refers to inter-disciplinary techniques and methodologies that aim to measure and improve day-to-day business performance in terms of increasing the efficiency of internal business processes and improving customer experience and value.

Traditional analytics like Business Intelligence (BI) provide each Line of Business (LOB) with insights to identify trends and take decisions based on what happened in the past.

But this is no longer sufficient. To deliver a good customer experience, organizations must continually measure their workload performance and quickly respond to operational inefficiencies for a better customer experience.

By using operational analytics systems, they can initiate such business actions based on the recommendations that the systems provide. They can also automate the execution processes to reduce the human errors. This makes the system go beyond being descriptive to being more prescriptive and even being predictive in nature.

On the other hand, IT infrastructures are becoming increasingly distributed adding more complexity to the workloads in terms of identifying the operational data that captures the system's state, characterize its behavior, and finally rectify potential issues in the pipelines.

Several tools and methodologies have emerged that help companies keep their systems reliable. Every system or application must be instrumented to expose telemetry data that provides operational insights in real or near real time.

The telemetry data can be of different form of signals: logs, traces, and metrics. Traditionally this data came in the form of logs that represent a record of an event happened within an application, server or a system operation. It can be of different types such as: application logs, security logs, system logs, audit trails, and infrastructure logs. Logs are usually used in troubleshooting and generating root-cause analysis for a system or application failure at a specific point in time.

Trace signal captures the user request for resources as it passes through different systems all the way to the destination and the response back to the user. It indicates a causal relationship between all the services being part of a distributed transaction. Organizations used to develop their own trace mechanisms but it is recommended to use existing tools that support a standard trace-context propagation format. The trace-context holds the information that links the producer of a message to its downstream consumers.

Metric data provides a point-in-time measure of the health of the system, such as resource consumptions in terms of CPU utilization. Metric signals offer an overview of the overall system health while reducing the manual effort to build these metrics and store them. With metrics, system operators can be notified in real time about anomalies in production environments and establish automated recovery process in case of a recurrent incident.

The signals mentioned above have different ways to be instrumented and provide different approaches to implement operational analytics use cases. Therefore, organizations must have an operational objective in mind from which they can work backwards to identify what data output they need from their system, which tool is better fit for their business and IT environment and finally what insights are needed to better understand their customers and improve their production resiliency.

Characteristics

Discoverability: The ability of the system to make operational data available for consumption. This involves discovering multiple disparate types of data available within an application that can be used for various ad hoc explorations.

Connectivity: Operational data can emanate from a variety of data sources in different format with disparate volumes. For this reason, the operational system has to provide the capability to seamlessly integrate all the data with the least overhead for production application.

Scalability: The ability of the system to scale up and out to adapt to changes in the operational analytics workload in terms of storage or compute requirements.

Monitoring: You should be able to continuously monitor the operational system performance and get notified about the resource utilization and the overall health of your system.

Security: The access to the operational system must be secure. With Amazon OpenSearch Service, you can configure the domain to be accessible with an endpoint within your VPC or a public endpoint accessible to the internet. In addition to network-based access control, you must set up user authentication and authorization to secure the access to data based on business requirements. OpenSearch Service supports encryption at rest and in Transit.

Data durability: With operational analytics, the use cases differ as to the retention requirements. You should understand your business requirements in terms of analyzing historical data. With Amazon OpenSearch Service, you can retain more data with less cost using the UltraWarm and cold storage tiers.

Automation: The data lifecycle in your operational system should be automated in order to easily onboard new data pipelines and reduce the overhead of managing the lifecycle of the data. With Index State Management (ISM) in Amazon OpenSearch Service, you can create your own policies to automate the lifecycle management of indices stored in the service.

Observability: The ability to understand internal state from the various signal outputs in the system. By providing a holistic view of these various signals along with a meaningful inference it becomes easy to understand how healthy and well performing the overall system is.

User centricity: Each analytics application should address a well-defined operational scope and solve a particular problem at hand. Users of the system often won't understand or care about the analytics process but only see the value the result.

Agility: The system must be flexible enough to accommodate changing needs of an analytics application and offer necessary control to bring in additional data with low overhead.

Reference architecture

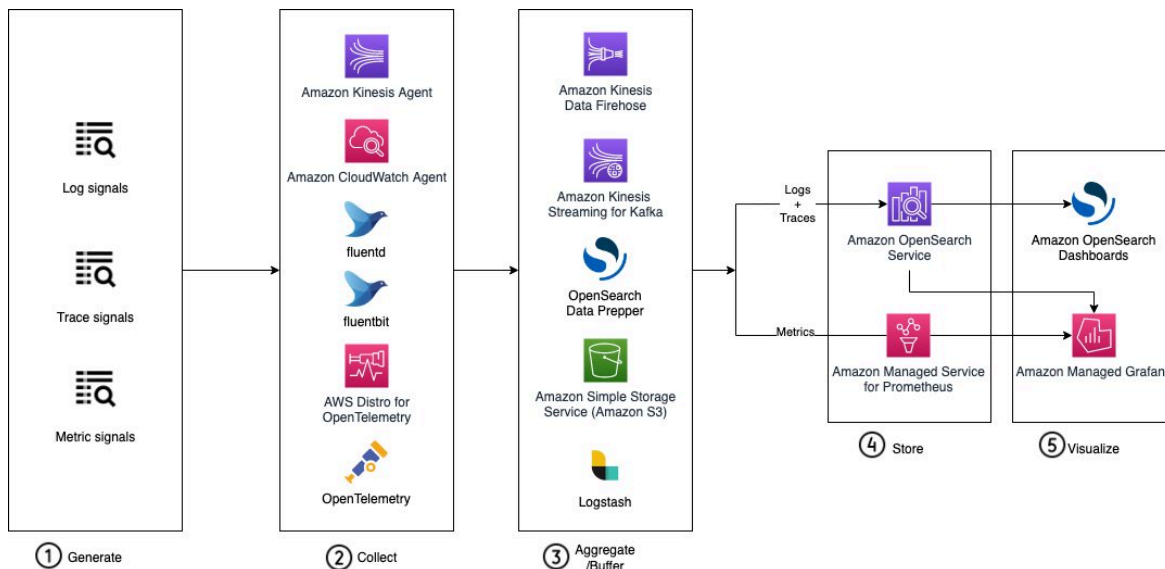


Figure 5: Operational analytics reference architecture

The reference architecture covers the data flow in an operational analytics use case. The Ingestion pipeline contains up to five stages as follows:

1. With your operational and business goal in mind, you should instrument your system/platform to **produce** the relevant type of signals such as various logs, traces, and metrics, and expose the data to a set of collectors. At this stage, you choose open-source instrumentation tools such as [Jaeger](#) or [Zipkin](#). And if you plan to generate different type of signals, we recommend that you include signal correlation beginning with the design step. Open-source tools such as [OpenTelemetry](#) facilitate the context propagation by adding a Trace ID to all logs related to a specific request. This reduces the mean time to problem resolution by enhancing the observability of the system from multiple viewpoints.
2. The second step is to **collect** the telemetry data from the producers and deliver it to the aggregators or buffers. You can use native AWS services (such as [Amazon Kinesis Agent](#), [CloudWatch agents](#), or [AWS Distro for OpenTelemetry](#)) to let you instrument your applications just once, collect and correlate metrics and traces, along with contextual information and metadata about where the application is running. You can also use a number of lightweight shippers such as [Fluentd](#) to collect logs, [Fluentbit](#) to collect both logs and metrics, and open-source [OpenTelemetry](#).
3. Before sending the data to [Amazon OpenSearch Service](#), it is recommended that you **buffer or aggregate** information from the collectors to reduce the overall connections to the domain and use the batch (`_bulk`) API to send batches of documents rather than sending single documents. It is also possible at this stage (or at the collection stage) to transform and aggregate the data for the downstream analytics tools. To do this, you can use AWS services such as [Amazon Kinesis Data Firehose](#) and [Amazon Managed Streaming for Apache Kafka](#). For large-scale environments, you can use [Amazon S3](#) to have a backup the data. It is also possible to use open-source tools such as OpenSearch Data Prepper for trace and log analytics, or you can use the [open source version of Logstash](#) (check compatibility with Amazon OpenSearch Service [here](#)).
4. Amazon OpenSearch Service makes it easy for you to index and **store** telemetry data to perform interactive analytics. Amazon OpenSearch Service is built to handle a large volume of structured and unstructured data from multiple data sources at high ingestion rates. Amazon OpenSearch Service integrates not only with AWS services but also with open-source tools as the ones listed previously. It is also possible to use [Amazon Managed Service for Prometheus](#) to **store** and **query** operational

metrics. The service is integrated with Amazon Elastic Kubernetes Service (Amazon EKS), Amazon Elastic Container Service (Amazon ECS), and AWS Distro for Open Telemetry.

5. Amazon OpenSearch Service dashboard is the default **visualization** tool for data in Amazon OpenSearch Service. It also serves as a user interface for many of the OpenSearch plugins, including Observability, Security, Alerting, Index State Management, and SQL. You can also conduct interactive analysis and visualization on data with [Piped Processing Language \(PPL\)](#), a query interface. You can use [Amazon Managed Grafana](#) to complement Amazon OpenSearch Service on the visualization layer. And you connect Amazon Grafana to Amazon Managed Service for Prometheus to query, **visualize**, alert on, and understand metric data.

Configuration notes

As shared in the previous section, there are different options and a non-exhaustive list of tools that you can choose from to implement an operational analytics pipeline. A list of configuration parameters to take into consideration for a well-architected operational pipeline is provided.

Define operational goals and business requirements: As a best practice, you should always start by identifying your operational goals, and what business outcome you must reach. Think about who are your end users, what are the insights to help drive their decisions, and how they will access these insights. After you define the business requirements, you can start designing your technical pipeline, establishing the integration options in your environment, and reviewing the skill sets you have, to choose the right option.

Choose a data model before ingestion: When bringing data in from disparate sources, especially from structured systems into structureless systems such as OpenSearch, special care must be taken to ensure that the chosen data model provides a frictionless search experience for users.

Ingestion pipeline: You should make sure that your ingestion framework is reusable and extensible to be able to scale and include new use cases on the long term, otherwise, check which parts of your infrastructure would require modernization.

Production ready tools and services: AWS offers a set of managed services that are production ready and which eliminate the operational overhead of managing the infrastructure, such as Amazon OpenSearch Service. As shared in the reference architecture, you can also integrate open source tools, such as OpenSearch Data Prepper, to transform and aggregate the operational data for downstream analytics and visualizations.

Sizing OpenSearch domain: The first step in sizing an OpenSearch cluster is to check your data size, and identify your storage and query requirements. Estimate the number of active shards you will have per index based on your input data, and the shard size that you identify. Then, estimate your vCPU requirements and choose the type of instances that will be able to handle both storage and vCPUs. Plan for time to benchmark the domain with a realistic dataset using [OpenSearch Benchmark](#), tune the configuration and iterate until you meet the performances required in terms of Throughput, Search Latency, and Index Latency. For more information, see [Sizing Amazon OpenSearch Service domains](#) and [Best practices for configuring your Amazon OpenSearch Service domain](#).

Use tiered storage: The value of operational data or any timestamped data generally decreases with the age of the data. Moving aged data into tiered storage can save significant operational cost. Summarized rollups that can condense the data can also help address storage cost.

Performance: There are multiple parameters to consider when thinking about performance and it is always specific to each workload. However, Amazon OpenSearch Service offers features that you can already enable in your domain, such as [Auto-Tune](#) that automatically deploys optional changes to improve cluster speed and stability. Other items to take into consideration include using the `_bulk` API to load data into OpenSearch, and only indexing data fields that need be searchable.

Define security requirements: Make sure to set up your domain inside a virtual Private Cloud (VPC) to secure the traffic to your domain. Apply the least privilege access approach with restrictive access

policies, or with fine-grained access control for OpenSearch dashboards. OpenSearch Service also offers encryption of data at rest and in transit.

Monitor all involved components: Monitor all involved components with metrics in Amazon CloudWatch. With the CloudWatch metrics available for Amazon OpenSearch Service, you can monitor the overall cluster health, you can also check the performance of individual nodes and monitor EBS volume metrics. It is also a best practice to set [CloudWatch alarms](#) to get notified about any issues that your production domain encounters. You can start by setting the following alarms:

- CPUUtilization maximum is $\geq 80\%$ for 15 minutes, 3 consecutive times
- ClusterStatus.yellow maximum is ≥ 1 for 1 minute, 1 consecutive time
- JVMMemoryPressure maximum is $\geq 80\%$ for 5 minutes, 3 consecutive times
- FreeStorageSpace minimum is $\leq 25\%$ of the storage space for 1 minute, 1 consecutive time

Data visualization

Every day, the people in your organization make decisions that affect your business. When they have the right information at the right time, they can make the choices that move your company in the right direction. This gives decision makers the opportunity to explore and interpret information in an interactive visual environment to democratize data and accelerates data-driven insights that are easy to understand and navigate.

Building a BI and data visualization service in the cloud allows you to take advantage of capabilities such as scalability, availability, redundancy, and enterprise grade security. It also lowers the barrier to data connectivity and allows access to a far wider range of data sources—both traditional, such as databases, as well as non-traditional, such as SaaS sources. An added advantage to a cloud-based data visualization service is the elimination of undifferentiated heavy lifting related to managing server infrastructure.

Characteristics

Scalability: Ensure that the underlying BI infrastructure is able to scale up vertically and horizontally both in terms of concurrent users as well as data volume. For example, Amazon QuickSight SPICE, and web applications automatically scale up server capacity to accommodate a large number of concurrent users without any manual intervention in terms of provisioning additional capacity for data, load balancing, and other services.

Connectivity: BI applications must be able to not only connect with data platforms such as traditional data warehouses and databases, but also support connectivity to a data lake and modern data architectures. The application must also have the capacity to connect to non-traditional sources, such as SaaS applications. Typically, data stores are secured behind a private subnet and BI tools and applications must be able to connect in a secure mechanism using strategies, such as VPC endpoints and secure firewalls.

Centralized security and compliance: BI applications must allow for a layered approach for security. This includes: Securing at the perimeter using techniques such as IP allow lists, security groups, ENIs and IAM policies for cloud resource access, securing the data in transit and data at rest using SSL and encryption, and restricting varying levels of access through fine-grained permissions for users to the underlying data and BI assets. The application must also comply with the governmental and industry regulations for the country or region the company is bound by.

Sharing and collaboration: BI applications must support data democratization. They must have features that allow sharing of the dashboards with other users in the company as well as for multiple report authors to collaborate with one another by sharing access to the underlying dataset. Not all BI tools have this capability. Amazon QuickSight allows the sharing of assets, such as data sources, data sets, analyses, dashboards, themes, and templates.

Logging, monitoring, and auditing: BI applications must provide adequate mechanisms to monitor and audit the usage of the application for security (to prevent unwanted access to data assets and other resources) and troubleshooting. Amazon QuickSight can be used with Amazon CloudWatch, AWS CloudTrail, and IAM to track record of actions taken by a user, user role, or an AWS service. This provides the who, what, when, and where of every user action in QuickSight.

Perform advanced analytics

Modern BI applications must be able to discover hidden insights from your data, perform forecasting and what-if analysis, or add easy-to-understand natural language narratives to dashboards. The business users need the ability to perform analytics without deep statistical and machine learning knowledge.

Amazon QuickSight ML Insights provide features that make it easy to discover hidden trends and outliers, identify key business drivers, and perform powerful what-if analysis and forecasting with no technical or ML experience.

Enable self-service business intelligence

The common challenges of BI tools are how to make data more accessible to more people without extensive user training and technical understanding. Data must be available in all format - raw, semi-processed and processed. Self-service BI should allow users to interact with data on an as-needed-basis without involving IT.

Amazon QuickSight Q allows user to ask business questions in natural language and receive answers with relevant visualizations that help them gain insights from the data. QuickSight Q uses machine learning to interpret the intent of a question and analyze the correct data to provide accurate answers to business questions quickly

Reference architecture

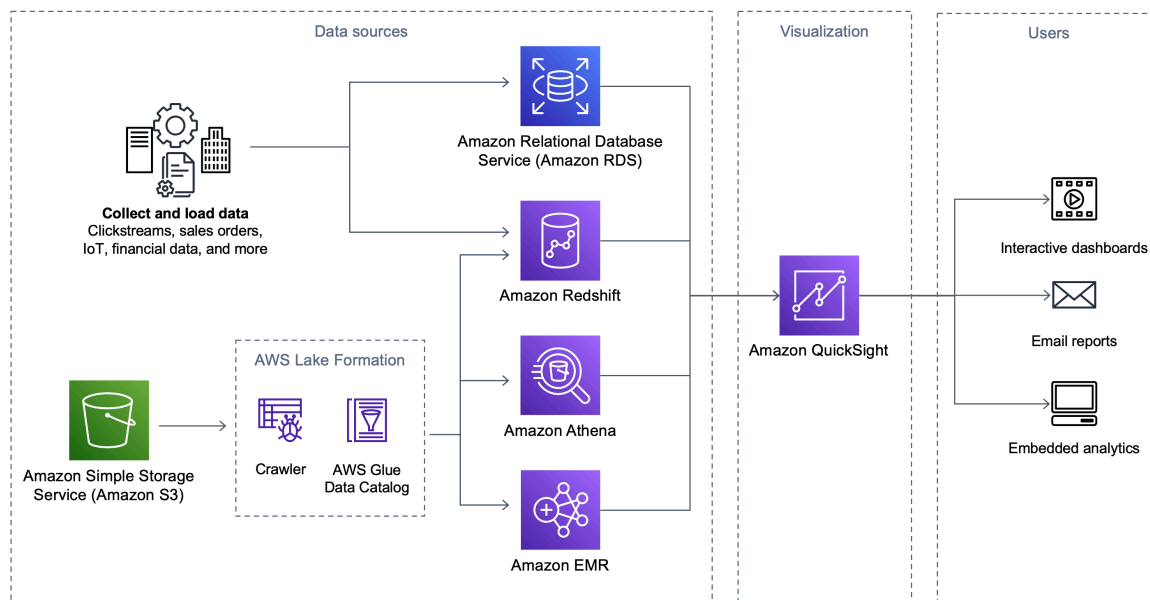


Figure 6: QuickSight dashboard end-to-end design

Data sources: Supports connection with traditional Data Warehouse or databases and also have the capacity to connect to non-traditional sources such as SaaS applications. Supported datasources in QuickSight include Amazon S3, Amazon Redshift, Amazon Aurora, Oracle, MySQL, Microsoft SQL Server, Snowflake, Teradata, Jira, and ServiceNow. Check [here](#) for the complete list of data sources supported in

QuickSight. These data sources could be secured behind a private subnet and QuickSight can connect in a secure mechanism using strategies such as VPC endpoints, and secure firewalls.

Visualization Tool: Amazon QuickSight.

Consumers: Visual dashboard consumers accessing a QuickSight console or embedded QuickSight analytics dashboard.

Configuration notes

Security: Implement the principle of least privilege throughout the visualization application stack. Ensure data sources are connected using VPCs and restrict security groups to only the required protocols, sources, and destinations. Enforce that the users as well as applications in every layer of the stack are given just the right level of access permissions to data and the underlying resources. Ensure seamless integration with identity providers—either industry supported or customized. To ease flow and remove confusion, set up QuickSight and single sign-on (SSO) such that email addresses for end users are automatically synced at their first login. In the case of multi-tenancy, use namespaces for better isolation of principals and other assets across tenants. For example, QuickSight follows the least privilege principle and access to AWS resources such as Amazon Redshift, Amazon S3 or Amazon Athena (common services used in data warehouse, data lake or modern data architectures) can be managed through the QuickSight user interface. Additional security at the user or group level is supported using fine-grained access control through a combination of IAM permissions. Additionally, QuickSight features, such as row level security, column level security, and a range of asset governance capabilities that can be configured directly through QuickSight user interface.

Cost optimization: Accurately identify the volume of dashboard consumers and embedding requirements to determine the optimal pricing model for the given visualization use case. QuickSight offers two different pricing options (capacity and user based) that allows clients to implement cost-effective BI solutions. Capacity pricing allows large-scale implementations and user-based pricing allows clients to get started with minimal investment (Note: SPICE has a 500M records or 500 GB volume per dataset limitation).

Low latency considerations: Use in-memory caching option, such as Memcached, Redis, or the in-memory caching engine in QuickSight called SPICE (Super-fast, Parallel, In-memory Calculation Engine) to prevent latency in dashboard rendering while accommodating any built-in restrictions that the caching technology might have.

Pre-process data views: Ensure that the data is cleansed, standardized, enhanced, and pre-processed to allow analysis within the BI layer. If possible, create pre-processed, pre-combined, pre-aggregated data views for analysis purposes. ETL tools, such as AWS Glue DataBrew, or techniques, such as materialized views, can be employed to achieve this. After uploading the dataset, users can add calculated fields to a dataset during the data preparation or from the analysis page for additional insights provided data.

Conclusion

This lens provided 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 whitepaper also discussed the AWS Well-Architected Framework 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 time to focus on running analytics workloads and pushing the boundaries of the field to which you're committed. The analytics landscape is continuing to evolve as the number of tools and processes grows and matures.

As this evolution occurs, we will continue to update this whitepaper to help you ensure that your analytics applications are well architected.

Contributors

The following individuals and organizations contributed to this version of this whitepaper:

- Russell Jackson: Sr. SSA, Analytics, Amazon Web Services
- Pragnesh Shah: Migration and Modernization PSA, Amazon Web Services
- Raghavarao Sodabathina: Principal Solutions Architect, Amazon Web Services
- Indira Balakrishnan: Principal Specialist SA, Amazon Web Services
- Dhiraj Thakur: Senior Solutions Architect, Amazon Web Services
- Manos Samatas: Principal SSA, Analytics, WWPS EMEA, Amazon Web Services
- Srikanth Sopirala: Principal US Central Analytics SSA, WWSO, Amazon Web Services
- Navneet Srivastava: Principal HCLS DL/A Specialist, AWS BDSI HCLS, Amazon Web Services
- Shana Schipers: Specialist SA, Analytics, AWS WWSO, Amazon Web Services
- Pratik Patel: Enterprise Support Lead (US-Cen), AWS Enterprise Support, Amazon Web Services
- Hajer Bouafif: Specialist SA, Analytics, Amazon Web Services
- Bhasi Mehta: QuickSight SSA, Amazon Web Services
- Padmaja Suren: Technical BDM-Cloud Intel & Pricing, WWPS Solution Architecture, Amazon Web Services
- Srikanth Baheti: Senior Amazon QuickSight Senior Solutions Architect, Amazon Web Services
- Raji Sivasubramaniam: Amazon QuickSight Senior Solutions Architect, Amazon Web Services
- Bruce Ross: Sr Lens Leader, Well-Architected, Amazon Web Services

The following individuals and organizations contributed to earlier versions of this whitepaper:

- Wallace Printz: Senior SA, Amazon Web Services
- Radhika Ravirala: Principal SA, Amazon Web Services
- Indira Balakrishnan: Senior Specialist SA, Amazon Web Services
- Juan Yu: Senior Specialist SA, Amazon Web Services
- Richard Mei: Senior SA, Amazon Web Services
- Shankar Nivas: Principal Data Architect, Amazon Web Services
- Todd McGrath: Senior Specialist SA, Amazon Web Services
- Padmaja Suren: Senior Specialist SA, Amazon Web Services
- Harsha Tadiparthi: Principal Specialist SA, Amazon Web Services
- Muthu Pitchaimani: Analytics Specialist SA, Amazon Web Services
- David Tucker: Senior Manager of Analytics Solutions Architects, Amazon Web Services
- Srikanth Baheti: Senior Amazon QuickSight Senior Solutions Architect, Amazon Web Services
- Raji Sivasubramaniam: Amazon QuickSight Senior Solutions Architect, Amazon Web Services

Document revisions

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

Change	Description	Date
Minor update (p. 90)	Contributors updated.	January 5, 2023
Major update (p. 90)	Added Sustainability content and numerous updates throughout.	November 17, 2022
Minor update (p. 90)	Updates throughout and added placeholder for Sustainability.	September 14, 2022
Whitepaper updated (p. 90)	Title changed to Data Analytics Lens. Major updates applied throughout.	October 29, 2021
Initial publication (p. 90)	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.

© 2022 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.