Best practices for designing and implementing modern data-centric architecture use cases

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Best practices for designing and implementing modern data-centric architecture use cases

Table of Contents

Best practices for designing and implementing modern data-centric architecture use cases

Apoorva Patrikar, Amazon Web Services (AWS)

May 2023 (document history)

Organizations are increasingly moving away from application-centric architectures to embrace data-centric architectures where IT infrastructure, application development, and even business processes are designed around data requirements. In a data-centric archicture, data is a core IT asset, and you design your IT systems and processes to optimize your data.

This guide offers best practices for designing a modern data-centric archicture for your use case. You can use these best practices to modernize your data pipelines and the data engineering operations that support that pipeline. This guide also provides an overview of the lifecycle of data in a data pipeline. By understanding this lifecycle, you can build data pipelines that optimize your data.

You can use this guide to overcome the following challenges that many organizations face when designing a data-centric architecture for data pipelines:

- **Aversion to storing multiple versions of the same dataset** – It's not uncommon to frequently process data multiple times, but this approach has its limitations. In fact, it's often less resource intensive and more cost effective to avoid processing data multiple times. This guide shows you the benefit of taking a different approach that focuses on storing processed data in multiple stages.
- **Reluctance to embrace data lakes** – It can be difficult to sort through the marketing claims around data lakes, and it can also be challenging to figure out if your organization has the skills and resources required for incorporating a data lake into your IT systems and processes. This guide can help you understand how a data lake can be a useful component in your data-centric architecture.
- **Hiring enough data engineers** – Market trends suggest that data scientists are expected to perform data engineering tasks in many organizations even though they don't have the right data engineering skills. This skills gaps can have an impact on your time-to-market plans. This guide can help you better understand what data engineering skills are essential for designing a data-centric architecture.

- **Lack of knowledge about using AWS services for horizontal processing** – Horizontal or distributed processing enables a cluster to process chunks of data in parallel by mapping tasks to multiple nodes and collecting the result before sending it transparently to the user. The move toward horizontal processing represents a shift around how data is viewed and processed. This shift affects not just application logic or the application itself but also the way organizations work with data. For example, horizontal processing affects central storage, task distribution, and modularization. Horizontal processing also favors larger chunks of data for read-write operations. This guide explains how horizontal processing can work for your data pipeline.

Targeted business outcomes

This guide can help you and your organization achieve the following business outcomes:

- Better planning for your data architecture projects
- More secure governance of the data lifecycle
- Faster deployment of data pipeline projects
- Higher-quality architecture and engineering of data pipeline solutions

Data engineering principles

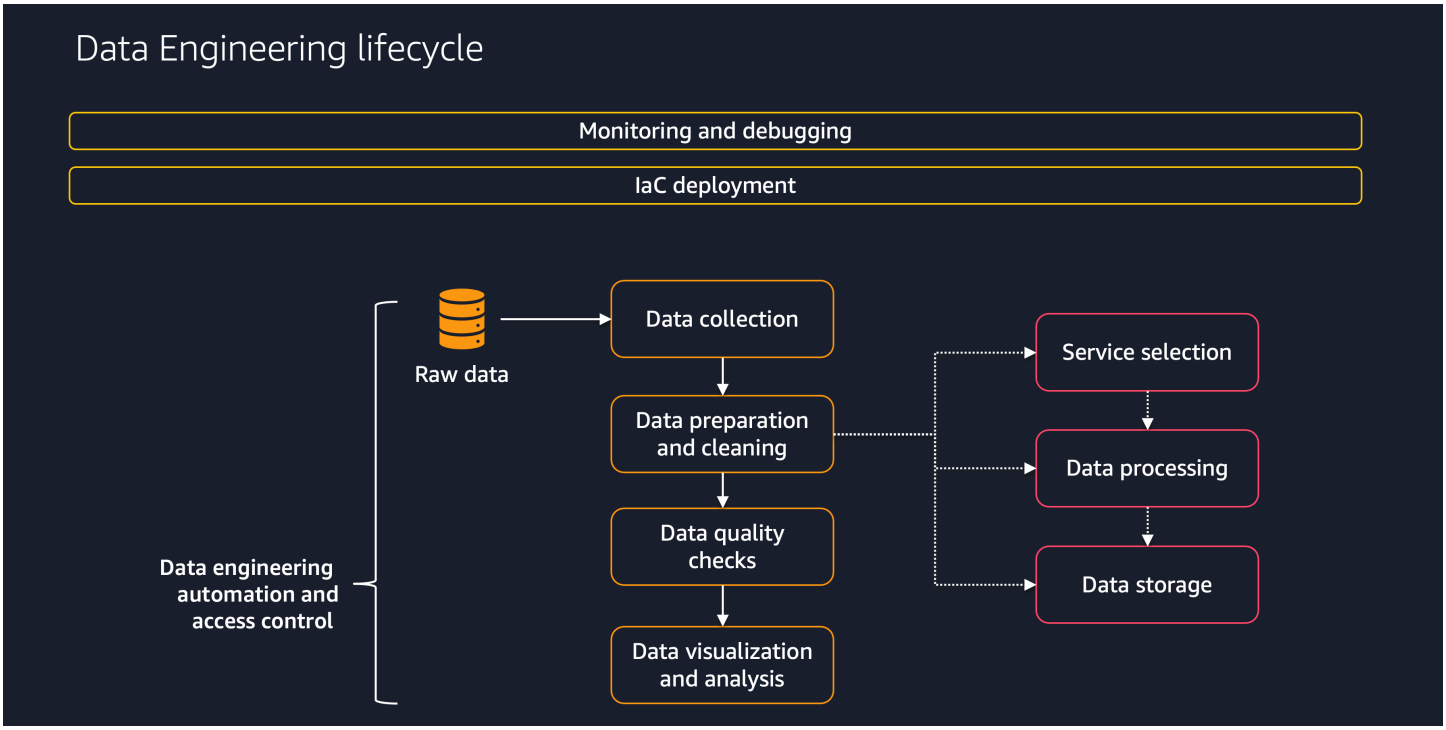
We recommend that you adopt the principles in the following table when you build an architecture for a modern data pipeline.

Principle	Example	Use case
Flexibility	Use microservices	FastGo enjoys flexibility and scalability with a microservices architecture on AWS (AWS case study)
Reproducibility	Use infrastructure as code (IaC) to deploy your services	Part 3: How NatWest Group built auditable, reproducible, and explainable ML models with Amazon SageMaker (AWS Machine Learning Blog)
Reusability	Use libraries and references in a shared manner	Create and reuse governed datasets in Amazon QuickSight with new Dataset-as-a-Source feature (AWS Big Data Blog)
Scalability	Choose service configurations to accommodate any data load	Designing a data lake for growth and scale on the AWS Cloud (AWS Prescriptive Guidance)
Auditability	Keep an audit trail by using logs, versions, and dependencies	How Parametric Built Audit Surveillance using AWS Data Lake Architecture (AWS Architecture Blog)

Data lifecycle

To build a data pipeline, you must first ingest data into AWS from an external or internal data source, such as a file server, database, storage bucket, or from an API call. The ingested data may or may not go through transformation, such as anonymization, column dropping, or data cleaning.

This section provides an overview of the stages in the data lifecycle process, as shown in the following diagram.

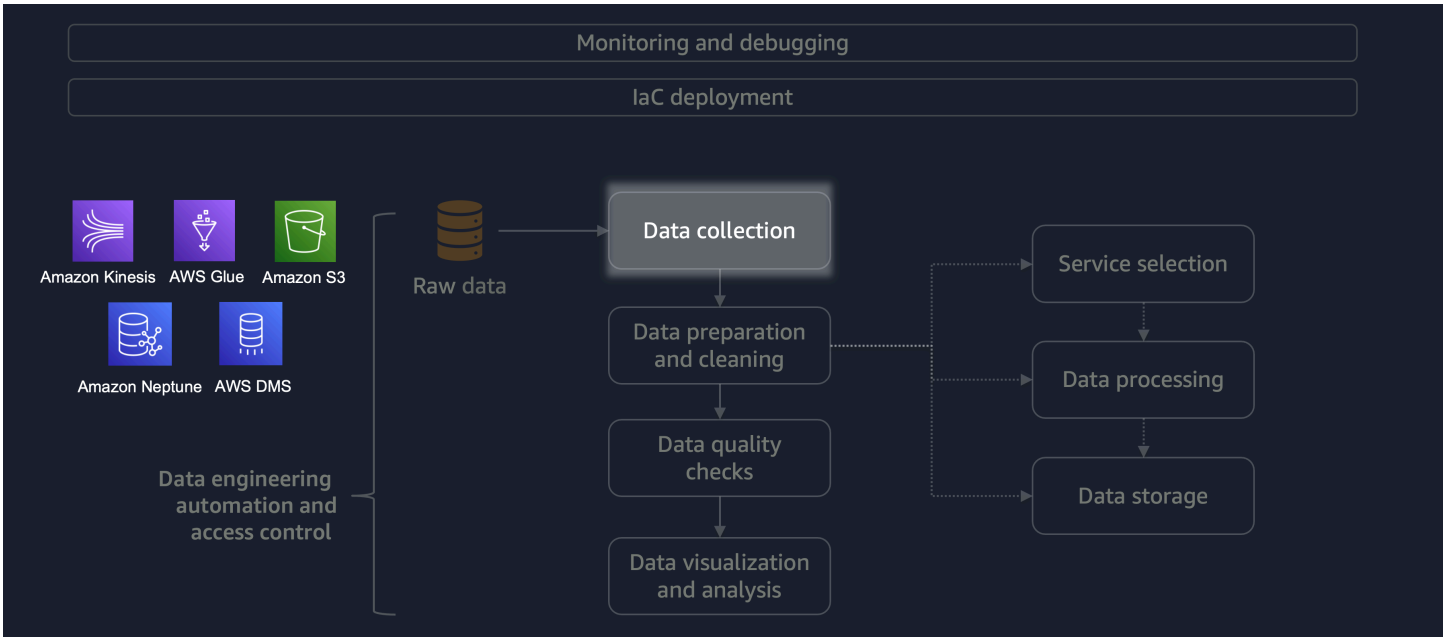


These stages include the following:

- Data collection
- Data preparation and cleaning
- Data quality checks
- Data visualization and analysis
- Monitoring and debugging
- IaC deployment
- Automation and access control

Data collection

You can collect data from a variety of sources within AWS, but it's important to choose the right data collection tool for your use case. The following diagram shows how the data collection stage fits into the data engineering automation and access control lifecycle.



AWS provides the following data collection tools:

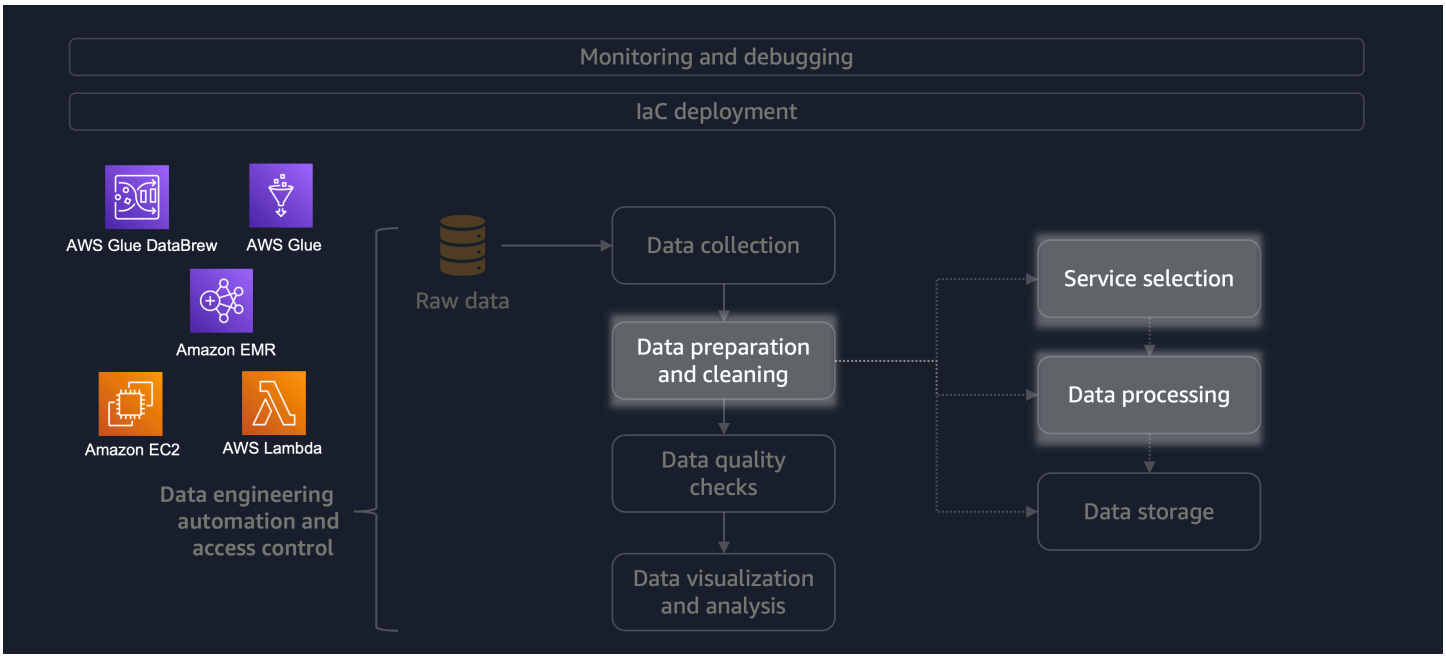
- Amazon Kinesis helps you collect streaming data. Kinesis also offers seamless integration and processing capabilities.
- AWS Database Migration Service (AWS DMS) helps you ingest data from relational databases. AWS DMS has configuration options and direct connections between on-premises and database services, such as Amazon Simple Storage Service (Amazon S3), that are hosted on AWS.
- AWS Glue is an extract, transform, and load (ETL) tool that helps you ingest unstructured data.

There are several use cases for collecting unstructured or semi-structured data by using Amazon S3 for storage. For example, a manufacturing site's data collection use case could require historical data to be ingested for machine history data as XML files, event data as JSON files, and purchase data from a relational database. This use case could also require that all three data sources must be joined.

Before you start the data ingestion process, we recommend that you understand what data must be ingested, and then choose the right tool to collect this data.

Data preparation and cleaning

Data preparation and cleaning is one of the most important yet most time-consuming stages of the data lifecycle. The following diagram shows how the data preparation and cleaning stage fits into the data engineering automation and access control lifecycle.



Here are some examples of data preparation or cleaning:

- Mapping text columns to codes
- Ignoring empty columns
- Filling empty data fields with `0`, `None`, or `''`
- Anonymizing or masking personally identifiable information (PII)

If you have a large workload that has a variety of data, then we recommend that you use Amazon EMR or AWS Glue for your data preparation and cleaning tasks. Amazon EMR and AWS Glue both work with unstructured, semi-structured, and relational data, and both can use Apache Spark to create a `DataFrame` or `DynamicFrame` to work with horizontal processing. Moreover, you can use AWS Glue DataBrew to clean and process data with a no-code approach. Additionally, DataBrew can profile your dataset with column statistics, provide data lineages, and include data quality rules for all or specified columns.

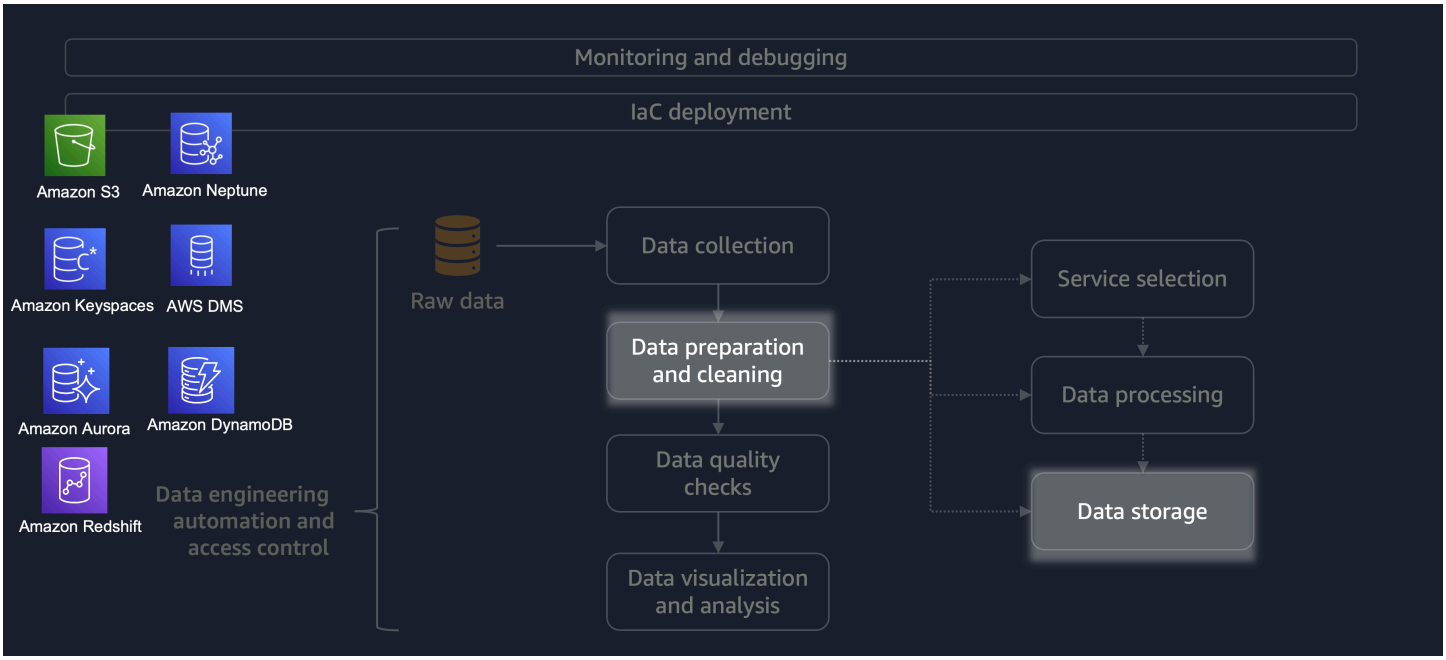
For smaller workloads that don't require distributed processing and can be completed in under 15 minutes, we recommend that you use AWS Lambda for data preparation and cleaning. Lambda is a cost-effective and lightweight option for smaller workloads. For highly secure data that can't enter the cloud, we recommend that you perform data anonymization on Amazon Elastic Compute Cloud (Amazon EC2) instances by using an AWS Outposts server.

It's essential to choose the right AWS service for data preparation and cleaning and to understand the tradeoffs involved with your choice. For example, consider a scenario where you're choosing from AWS Glue, DataBrew, and Amazon EMR. AWS Glue is ideal if the ETL job is infrequent. An infrequent job takes place once a day, once a week, or once a month. You can further assume that your data engineers are proficient in writing Spark code (for big data use cases) or scripting in general. If the job is more frequent, running AWS Glue constantly can get expensive. In this case, Amazon EMR provides distributed processing capabilities and offers both a serverless and server-based version. If your data engineers don't have the right skillset or if you must deliver results fast, then DataBrew is a good option. DataBrew can reduce the effort to develop code and speed up the data preparation and cleaning process.

After the processing is completed, the data from the ETL process is stored on AWS. The choice of storage depends on what type of data you're dealing with. For example, you could be working with non-relational data like graph data, key-value pair data, images, text files, or relational structured data.

As shown in the following diagram, you can use the following AWS services for data storage:

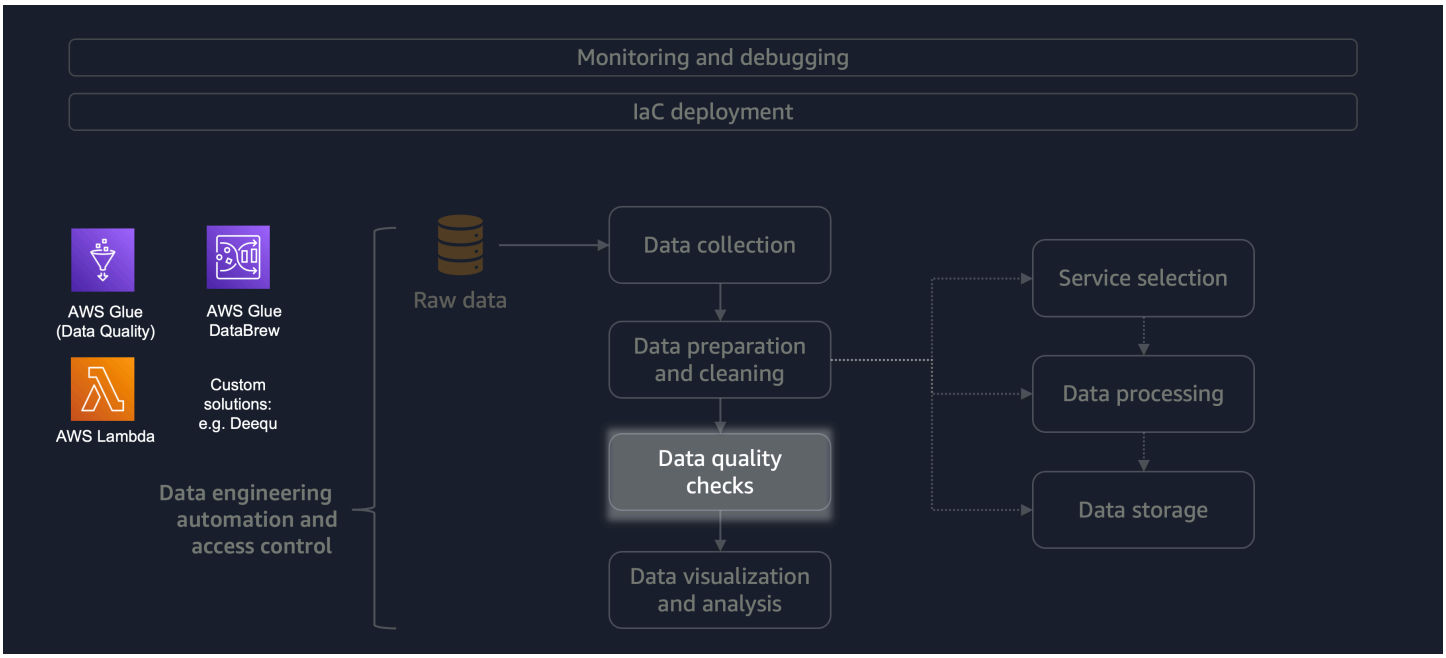
- Amazon S3 stores unstructured data or semi-structured data (for example, Apache Parquet files, images, and videos).
- Amazon Neptune stores graph datasets that you can query by using SPARQL or GREMLIN.
- Amazon Keyspaces (for Apache Cassandra) stores datasets that are compatible with Apache Cassandra.
- Amazon Aurora stores relational datasets.
- Amazon DynamoDB stores key-value or document data in a NoSQL database.
- Amazon Redshift stores workloads for structured data in a data warehouse.



By using the right service with the correct configurations, you can store your data in the most efficient and effective way. This minimizes the effort involved in data retrieval.

Data quality checks

Data quality is an integral yet often overlooked part of the data cleaning process. The following diagram shows how data quality checks fit into the data engineering automation and access control lifecycle.

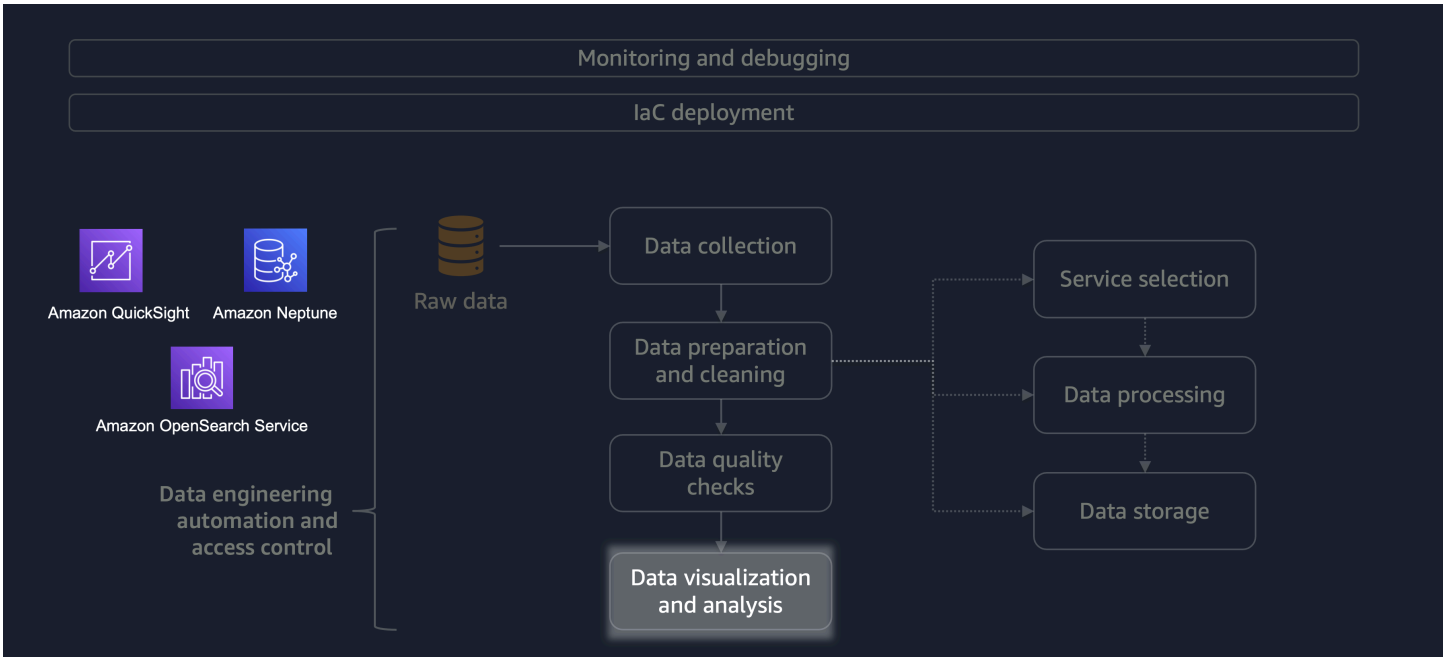


The following table provides an overview of different data quality solutions based on use case.

Use case	Solution	Example
No-code solution to add column-level or table-level quality conditions	AWS Glue DataBrew	Checks if all column values are between 1 and 12, or if a table or column is empty
Custom code added to an AWS Glue job or a no-code solution (in preview) to add column-level or table-level quality conditions	AWS Glue Data Quality	Checks if the column `first_name` is not null, or if the column `phone_number` contains only numbers or a "+" operator and/or statistical functions, such as average or sum
Custom checks	ETL of choice, such as AWS Lambda, AWS Glue, or Amazon EMR	Checks if the value of column A is always greater than the corresponding value of column B and column C, or if the value of column `continent` is always geographically correct and derived from the `city` column
Sophisticated solution with a metrics report, constraint validation, and constraint suggestions	Deequ	Checks if the `CompletenessConstraint` for the Completeness of column metric `review_id` is equal to 1

Data visualization and analysis

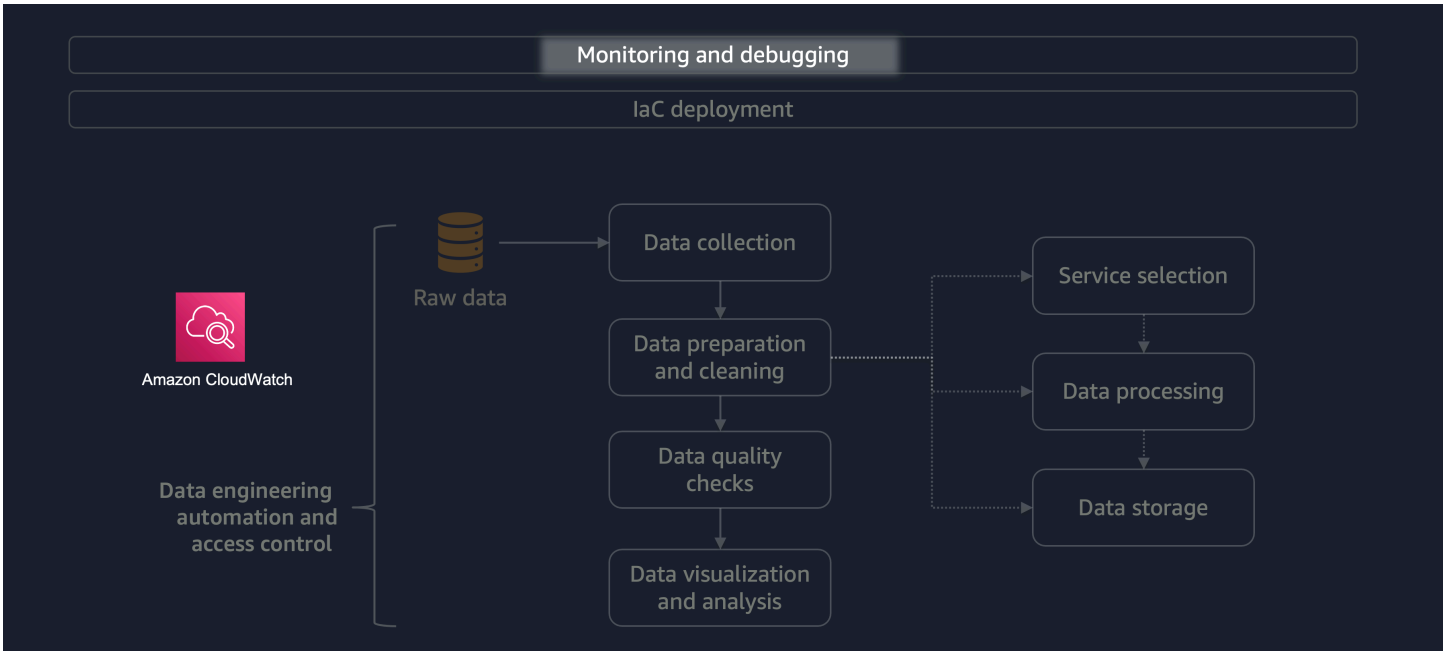
After you complete your data quality checks, then you can move to the data analysis or visualization stage, as shown in the following diagram.



In this stage, you can use Quick Sight for creating graphs or charts, Neptune for graph database operations and visualization, or OpenSearch for open-source search and analytics. Typically, you can also feed clean data into data science or machine learning (ML) workflows by using Amazon SageMaker pipelines or simple reads from Amazon S3. The data visualization and analysis stage concludes the sequential portion of the data engineering pipeline.

Monitoring and debugging

Certain phases in the data lifecycle are not sequential but consistently present. This is true for the monitoring and debugging stage, as shown in the following diagram.

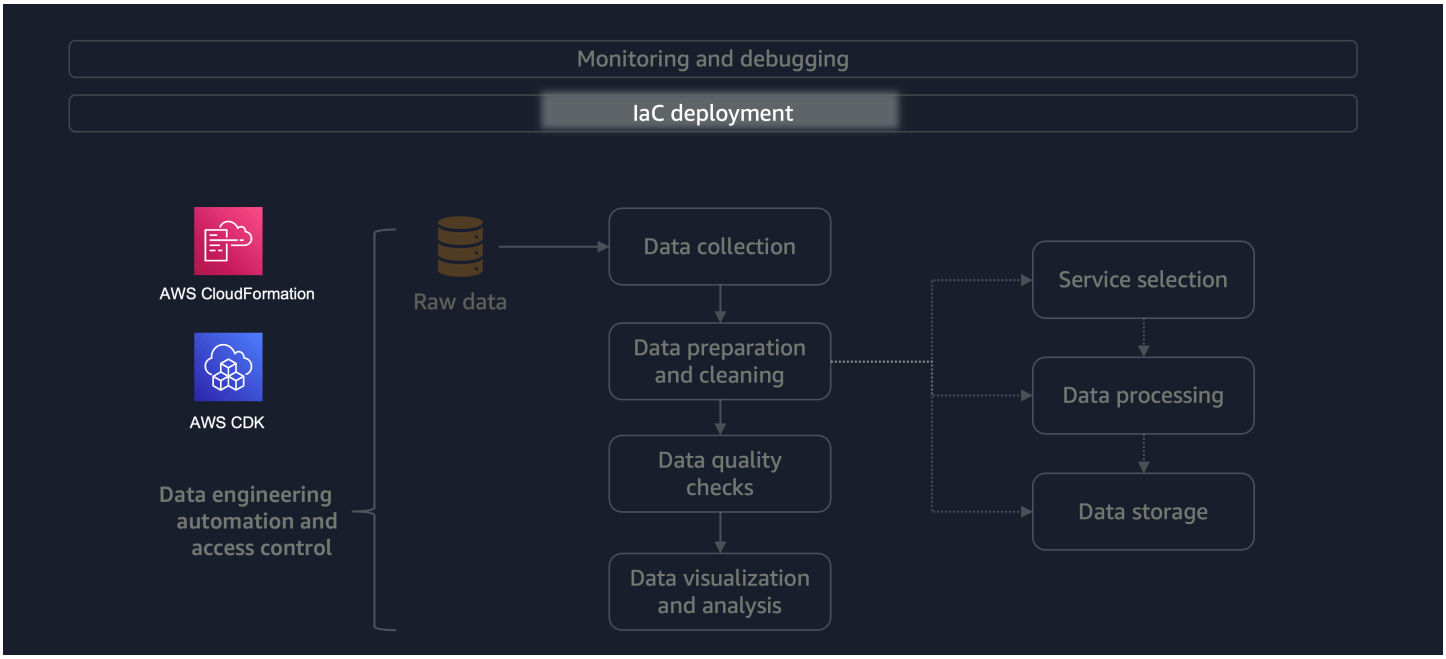


The process of data engineering must be continually monitored for correctness and performance. Amazon CloudWatch plays a crucial role in monitoring data engineering, as it logs every error and info log to its log groups. You can use monitoring to build automated error recovery. For example, you can stop pipelines if you find that your data quality rules are not satisfied, or you can log successful runs and failed runs separately to enable a recovery action. Monitoring improves the overall reliability of the data engineering process (that is, the full ETL process) as well as the data.

Additionally, we recommend that you create CloudWatch dashboards that include the relevant metrics for the monitoring and debugging process. This can help ensure that the data engineering process is running smoothly and as expected. This is important for operations as well as reporting. For example, a CloudWatch dashboard can show users the status of loads to help them understand the reliability of their processes or what percentage of their data was dropped due to low quality or which sources have the maximum failures. A CloudWatch dashboard not only helps you visualize results but also helps you improve processes by identifying the pain points in the ETL process.

IaC deployment

Modern architecture is incomplete without a mechanism for an infrastructure as code (IaC) deployment. The following diagram shows the AWS services related to IaC deployment.



We recommend that any deployed infrastructure is always backed by code using IaC tools. For example, you can use AWS CloudFormation or AWS Cloud Development Kit (AWS CDK). AWS CDK is a wrapper around CloudFormation.

As a best practice, we recommend that you push your code to a code repository of your choice. It's also a best practice to use source control in your code repository so that you have versioning and collaboration capabilities that enable multiple team members to work simultaneously on the same code base, while ensuring that the code integration from different developers into the main branch doesn't result in any conflicts.

Automation and access control

Automation

Pipeline automation is a crucial part of modern data-centric architecture design. To successfully run your production system, we recommend that you have a data pipeline that has a start trigger, connecting steps, and a mechanism for separating failed and passed stages. It's also important to log failures while not hindering the rest of the ETL process.

You can use AWS Glue workflows to create a pipeline. The pipeline supports all AWS Glue jobs, Amazon EventBridge triggers, and crawlers. You can also create workflows from scratch or by using AWS Glue blueprints. A blueprint provides a framework that helps you get started on reusable use

cases. For example, this could be a workflow to import data from Amazon S3 into a DynamoDB table. You can even use parameters to make the blueprint reusable.

If the data pipeline involves more services outside of AWS Glue, then we recommend that you use AWS Step Functions as the orchestrator. Step Functions can create automated workflows, including manual approval steps for security incident response. You can also use Step Functions for large-scale parallel or sequential processing.

Finally, we recommend using EventBridge to insert triggers on schedules, events, or on demand. You can also use EventBridge to create pipelines with filters.

Access control

We recommend that you use AWS Identity and Access Management (IAM) for access control. IAM allows you to specify who or what can access services and resources in AWS and centrally manage fine-grained permissions. Every phase of the lifecycle—from storage to automation to using processing tools—requires the right access permissions. While working with data-centric use cases, you can use AWS Lake Formation to simplify the process of making data available for wide-ranging analytics and also across accounts.

Best practices

We recommend that you follow storage and technical best practices. These best practices can help you get the most out of your data-centric architecture.

Storage best practices for big data

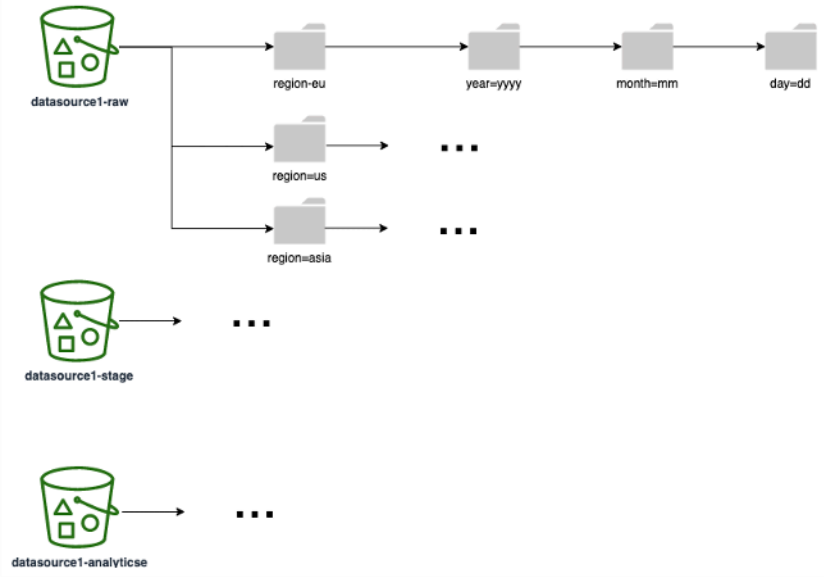
The following table describes a common best practice to store files for a big data processing load on Amazon S3. The last column is an example of a lifecycle policy that you can set. If Amazon S3 Intelligent-Tiering is enabled (which delivers automatic storage cost savings when data access patterns change automatically), then you don't have to manually set the policy.

Data layer name	Description	Example lifecycle policy strategy
Raw	Contains raw, unprocessed data **Note**: For an external data source, the raw data layer is typically a 1:1 copy of the data, but on AWS the data can be partitioned by keys based on AWS Region or date during the ingestion process.	After one year, move files into the S3 Standard-IA storage class. After two years in S3 Standard-IA, archive the files in Amazon Simple Storage Service Glacier (Amazon S3 Glacier). Amazon Glacier (original standalone vault-based service) will no longer accept new customers starting December 15, 2025, with no impact to existing customers. Amazon Glacier is a standalon e service with its own APIs that stores data in vaults and is distinct from Amazon S3 and the Amazon S3 Glacier storage classes. Your existing

		data will remain secure and accessible in Amazon Glacier indefinitely. No migration is required. For low-cost, long-term archival storage, AWS recommends the Amazon S3 Glacier storage classes, which deliver a superior customer experience with S3 bucket-based APIs, full AWS Region availability, lower costs, and AWS service integration. If you want enhanced capabilit ies, consider migrating to Amazon S3 Glacier storage classes by using our AWS Solutions Guidance for transferring data from Amazon S3 vaults to Amazon S3 Glacier storage classes.
Stage	Contains intermediate processed data that's optimized for consumption **Example**: CSV to Apache Parquet converted raw files or data transformations	You can delete data after a defined time period or according to your organizat ion's requirements. You can remove some data derivatives (for example, an Apache Avro transform of an original JSON format) from the data lake after a shorter amount of time (for example, after 90 days).

Analytics	Contains the aggregated data for your specific use cases in a consumption-ready format **Example**: Apache Parquet	You can move data to S3 Standard-IA, and then delete the data after a defined time period or according to your organization's requirements.

The following diagram shows an example of a partitioning strategy (corresponding to one S3 folder/prefix) that you can use across all the data layers. We recommend that you choose a partitioning strategy based on how your data is used downstream. For example, if reports are built on your data (where most common queries on the report filter the results based on region and dates), then make sure to include the regions and dates as partitions to improve query performance and runtime.



Technical best practices

Technical best practices depend on the specific AWS services and processing technologies that you use to design your data-centric architecture. However, we recommend that you keep in mind the following best practices. These best practices apply to typical data processing use cases.

Area	Best practice
SQL	Reduce the amount of data that must be queried by projecting attributes on your data.

	Instead of parsing the entire table, you can use data projection to scan and return only certain required columns in the table. Avoid large joins if possible because joins between multiple tables can significantly impact performance due to their resource-intensive demands.
Apache Spark	Optimize Spark applications with workload partitioning in AWS Glue (AWS Big Data blog). Optimize memory management in AWS Glue (AWS Big Data blog).
Database design	Follow the Architecture Best Practices for Databases (AWS Architecture Center).
Data pruning	Use server-side partition pruning with the `catalogPartitionPredicate` .
Scaling	Understand and implement horizontal scaling.

FAQ

What's the difference between a data pipeline and an ML pipeline?

A data pipeline is a data engineering pipeline that typically ingests, cleans, and processes data to make it compatible or optimized for machine learning (ML) or other analytical and visualization processes. An ML pipeline typically automates the creation of an ML model.

What's the difference between horizontal and vertical scaling?

Horizontal scaling is the addition of hardware to increase processing power and enable the use of clusters (for example, by using Amazon EMR or AWS Glue). Vertical scaling is the increase in processing power of existing hardware (for example, increasing the RAM capacity of an EC2 instance).

Next steps

We recommend that you take the following next steps:

1. Assess your current data engineering process and contact data experts for further guidance and assistance.
2. Adjust your data engineering process to ensure that all the stages of the data lifecycle are optimized for efficiency.
3. Use the data engineering principles and best practices in this guide to design a data-centric architecture and choose the AWS services that are most appropriate for your use case.

Resources

- AWS Data and Analytics Competency Partner
- AWS Glue
- AWS Step Functions
- Introduction to Apache Spark

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication	—	May 5, 2023