AWS Data Engineer - Associate (DEA-C01) Concise Guide

This concise guide covers all the exam topics in summary form. While it is a helpful resource, I recommend not relying solely on this guide. Instead, use it alongside the steps and resources listed on the blog post to ensure a better preparation for your exam.

Data Engineering Fundamentals 3

Fundamentals of Data Analysis - 1 3

Analytics types and techniques: 3

Machine Learning: 3

5 Vs of Big Data 4

1. Volume 4

2. Variety 4

3. Velocity 9

4. Veracity 12

5. Value 13

Fundamentals of data analysis - 2 13

Data Lake 13

Data Warehouse 13

Modern Data Movements 15

AWS Services for Modern Architectures: 17

Domain 1: Data Ingestion and Transformation 18

Data Engineering Fundamentals 18

1.1 Performing Data Ingestion 20

1.2 Transforming and Processing Ingested Data 23

1.3 Orchestrating Data Pipelines 26

1.4 Applying Programming Concepts in Data Pipelines 29

Domain 2: Data Store Management 32

2.1 Choosing the Right Data Store 32

2.2 Understanding Data Cataloging Systems 35

2.3 Managing the Lifecycle of Data 37

2.4 Designing Data Models and Schema Evolution 40

Domain 3: Data Operations and Support 43

3.1 Automating Data Processing with AWS Services 43

3.2 Analysing Data Using AWS Services 45

3.3 Maintaining and Monitoring Data Pipelines in AWS 48

3.4 Ensuring Data Quality: A Critical Task in Data Engineering 51

Domain 4: Data Security and Data Governance 54

4.1 AWS Authentication and Authorisation Concepts 54

4.2 Understanding Authorisation Mechanisms in AWS 59

4.3 Key Concepts in Data Encryption 61

4.4 Data Anonymization, Masking, and Key Salting 62

4.5 Understanding Data Privacy and Governance in AWS 64

Data Engineering Fundamentals

Fundamentals of Data Analysis - 1

How to take a business decision with my current data?
It's necessary to understand the patterns on it. This can achieved by implementing a process called *data analytics*, in which uses tools and techniques to find new value from raw data or/and implement the practice of interpreting data with *data analysis*.

Analytics types and techniques:

a. **Descriptive analytics:** meant to describe a scenario. It answers the question of: what happened? It uses historical data and the combination of visual charts to describe and tell a history.
b. **Diagnostic analytics:** meant to analyse and understand how a situation was developed to begin with. It answers the question of: why did it happen? It compares historical data with other datasets and the combination of statistical techniques to answer questions from raw data.
c. **Predictive analytics:** meant to predict future happenings. It uses historical data to forecast about future outcomes combined with machine learning algorithms.
d. **Prescriptive analytics:** basically meant to recommend actions to the predicted outcome. It's useful to create a more engaging and tailored user experience.

Machine Learning:

Definitions:

a. **Artificial intelligence (AI):** smart machine that can perform tasks requiring human intelligence.
b. **Machine learning (ML) model:** computer program that finds patterns in unanalysed datasets.
c. **ML algorithm:** computer program that helps computers understand hidden patterns in data, make predictions about the data and give recommendations about actions to take.

ML and analytics:

ML models can forecast what might happen in the future (*predictive analytics*) and provide a course of action (*prescriptive analytics*).

ML on AWS:

Amazon CodeWhisperer: Amazon CodeWhisperer generates complete functions and code blocks that align with your descriptions. Additionally, Amazon CodeWhisperer analyses the surrounding code, ensuring the generated code matches your style, naming conventions, and seamlessly integrates into the existing context.

5 Vs of Big Data

Companies with large amounts of data struggle with:

---

1. Volume

Indicates the amount of data that will be ingested by the solution - the total size of the data coming in.

Types of sources for ingestion and storage:

- *Transactional data:* customer information, online product purchase and service contracts.
- *Temporary data:* Internet browser cache.
- *Objects:* Images, email messages, text files, social media content, text messages, videos.

AWS Services for Volume:

- **Amazon Simple Storage Service (S3):** Object level storage, ideal for semistructured and unstructured data.
  Some characteristics of Amazon S3:
  1. Decoupling: separate data by their purpose: brown, silver and gold layers are an example of decoupling storage.
  2. Parallelisation: possible to access data in a bucket, in parallel, without impacting other processes.
  3. Centralise datasets: central location to store analytical datasets. It allows to avoid costly process of moving data between storage system and processing system.

- **AWS Lake Formation:** It makes it convenient to ingest, clean, catalog, transform, and secure data and make it available for analysis and ML. Lake Formation automatically configures underlying AWS services to ensure compliance with defined policies.

- **Amazon Redshift:** Data warehouse with columnar indexing.

---

2. Variety

Indicates the number of different sources, and the types of sources, that the solution will use.

Types of data:

- **Structured:** organised in table, with schema and is easy to be analysed. Downside: lack of flexibility.
- **Semistructured:** stored as documents or key-value pairs (JSON, CSV and XML are examples). Downside: difficult to analyse.
- **Unstructured:** stored as files (videos, images are examples). Downside: hardest to analyse.

Data source can also be classified by the type of storage used for it:

- **Structured data store:** relational database management system (RDBMS). Uses SQL to query the data.
- **Semistructured:** Non-relational databases (NoSQL db).
- **Unstructured:** data lakes or object storage solutions.

Within a database, there are two *methods to organise information*:

- **OLTP:** online transaction processing. Write-heavy operation in a db (make use of queries with INSERT, UPDATE and DELETE).

- **OLAP:** online analytical processing. Read-heavy operation in a db (make use of queries with SELECT and AGGREGATE).

Row-based and columnar data storage

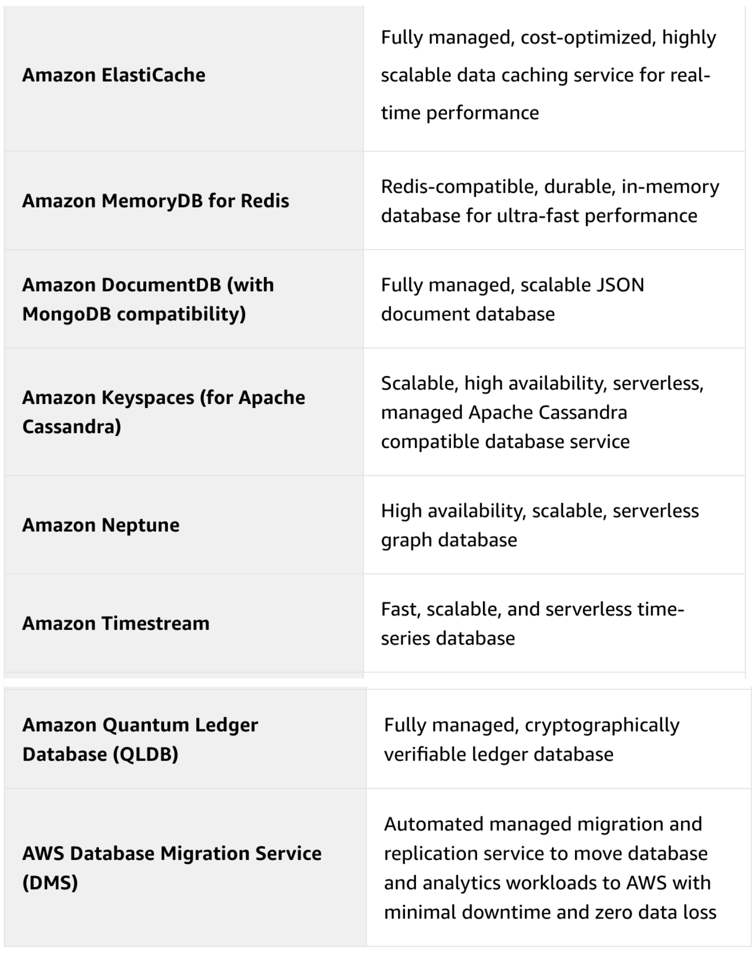
- **For OLTP Systems**: Row-based databases are preferred for their efficiency in handling numerous short, transactional queries that require quick read-write access.
- **For OLAP Systems**: Columnar-based databases are preferred for their ability to handle complex, read-intensive queries over large volumes of data, making them ideal for data analysis and reporting.

Comparing data storage methods

Characteristic	Row-based	Columnar
Representation	Multiple tables, each containing columns and rows	Collection of documents Single table with keys and values
Data design	Normalized relational or dimensional data warehouse.	Denormalized document, wide column or key value
Optimized	Optimized for storage	Optimized for compute
Query style	Language: SQL	Language: Many Uses object querying
Scaling	Scale vertically	Scale horizontally
Implementation	OLTP business systems	Analytical systems

The following table lists AWS services that provide purpose-built data stores.

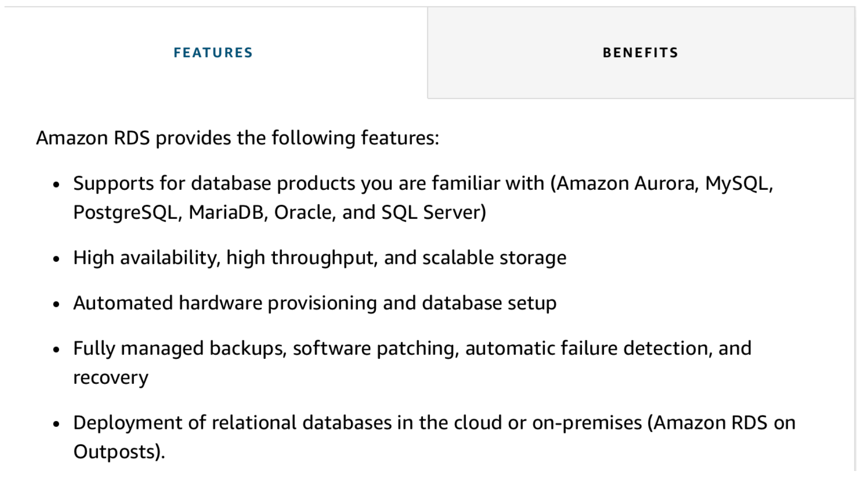
Databases	Description
Amazon Aurora	High performance, high availability, scalable, proprietary serverless relational database management system (RDBMS) with full MySQL and PostgreSQL compatibility
Amazon Relational Database Service (Amazon RDS)	Managed relational database service in the cloud with various database engine options
Amazon Redshift	Cloud-based data warehousing with ML to deliver the best price performance at any scale
Amazon DynamoDB	Fast, flexible, and highly scalable NoSQL database



Amazon ElastiCache	Fully managed, cost-optimized, highly scalable data caching service for real-time performance
Amazon MemoryDB for Redis	Redis-compatible, durable, in-memory database for ultra-fast performance
Amazon DocumentDB (with MongoDB compatibility)	Fully managed, scalable JSON document database
Amazon Keyspaces (for Apache Cassandra)	Scalable, high availability, serverless, managed Apache Cassandra compatible database service
Amazon Neptune	High availability, scalable, serverless graph database
Amazon Timestream	Fast, scalable, and serverless time-series database
Amazon Quantum Ledger Database (QLDB)	Fully managed, cryptographically verifiable ledger database
AWS Database Migration Service (DMS)	Automated managed migration and replication service to move database and analytics workloads to AWS with minimal downtime and zero data loss

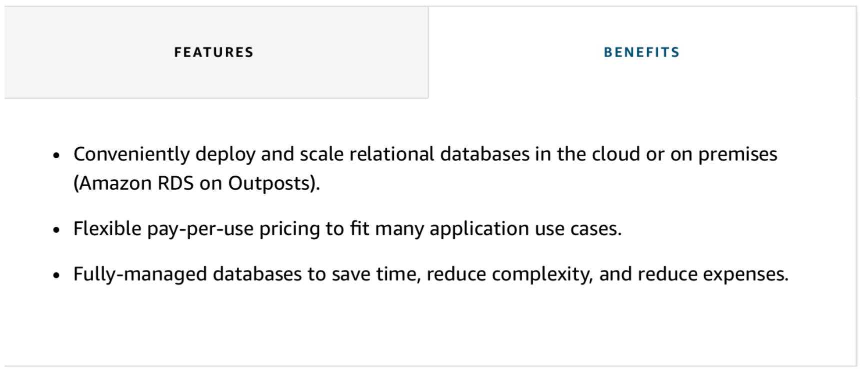
AWS Services for Variety:

1. **Amazon Relational Database Service (Amazon RDS):**



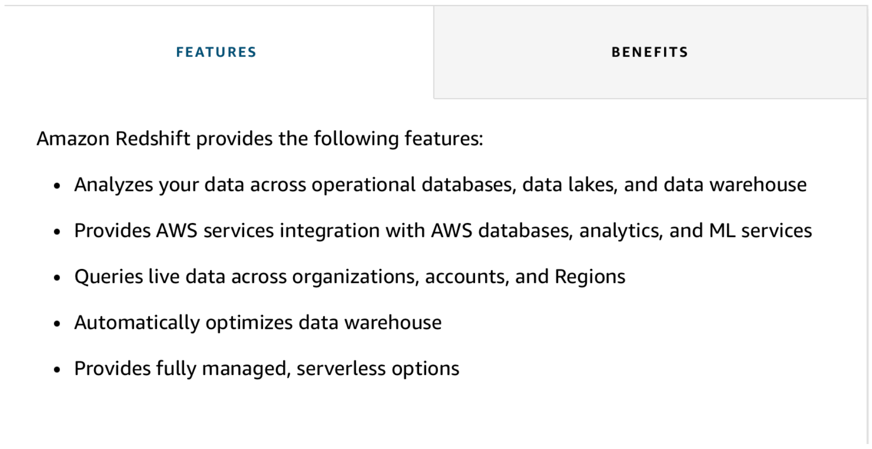
Amazon RDS provides the following features:

- Supports for database products you are familiar with (Amazon Aurora, MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server)
- High availability, high throughput, and scalable storage
- Automated hardware provisioning and database setup
- Fully managed backups, software patching, automatic failure detection, and recovery
- Deployment of relational databases in the cloud or on-premises (Amazon RDS on Outposts).


FEATURES | BENEFITS

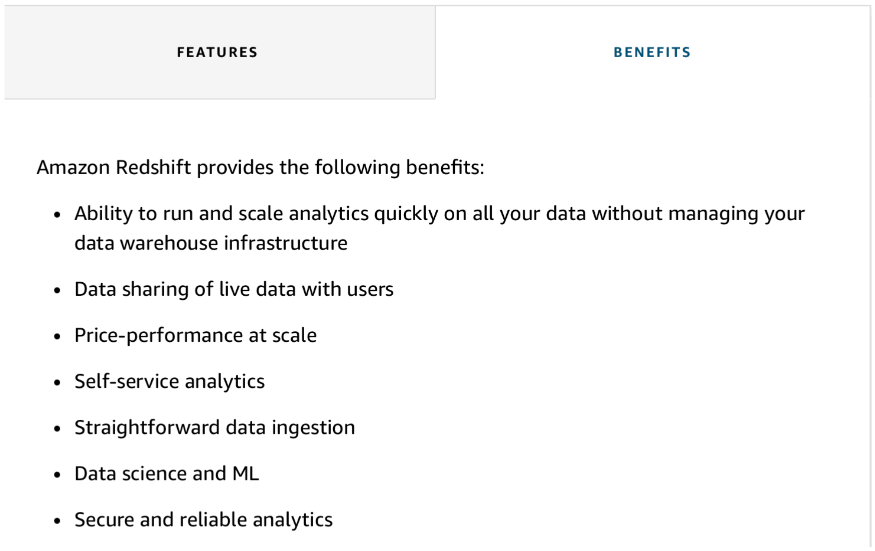
- Conveniently deploy and scale relational databases in the cloud or on premises (Amazon RDS on Outposts).
- Flexible pay-per-use pricing to fit many application use cases.
- Fully-managed databases to save time, reduce complexity, and reduce expenses.

2. **Amazon Redshift:** You can use Amazon Redshift's capabilities to run analytics on structured data within the data warehouse. When using additional features like ***Amazon Redshift Spectrum***, you can also analyse semistructured data stored in Amazon S3 data lakes, providing a comprehensive analytics solution.


FEATURES | BENEFITS

Amazon Redshift provides the following features:

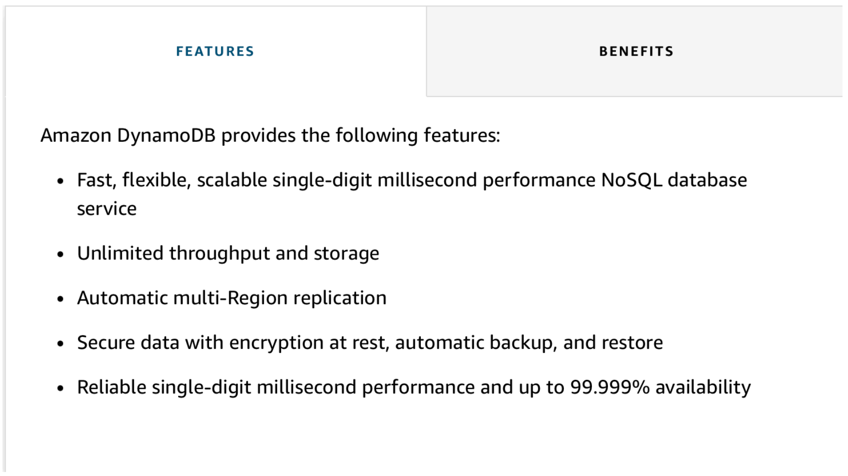
- Analyzes your data across operational databases, data lakes, and data warehouse
- Provides AWS services integration with AWS databases, analytics, and ML services
- Queries live data across organizations, accounts, and Regions
- Automatically optimizes data warehouse
- Provides fully managed, serverless options


FEATURES | BENEFITS

Amazon Redshift provides the following benefits:

- Ability to run and scale analytics quickly on all your data without managing your data warehouse infrastructure
- Data sharing of live data with users
- Price-performance at scale
- Self-service analytics
- Straightforward data ingestion
- Data science and ML
- Secure and reliable analytics

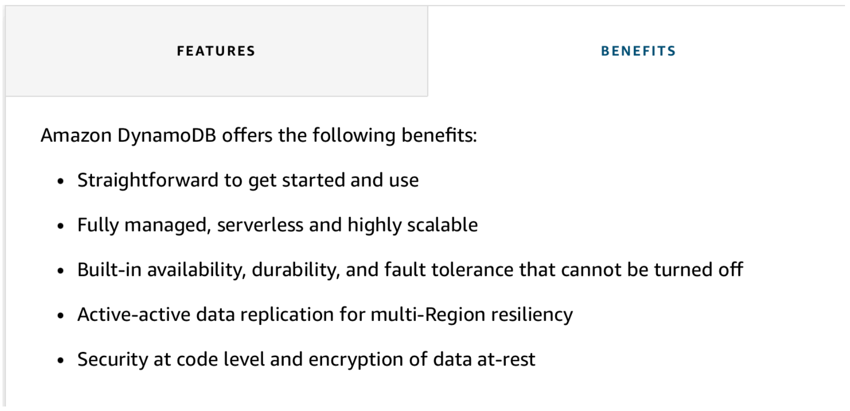
3. **Amazon DynamoDB:**



FEATURES | BENEFITS

Amazon DynamoDB provides the following features:

- Fast, flexible, scalable single-digit millisecond performance NoSQL database service
- Unlimited throughput and storage
- Automatic multi-Region replication
- Secure data with encryption at rest, automatic backup, and restore
- Reliable single-digit millisecond performance and up to 99.999% availability

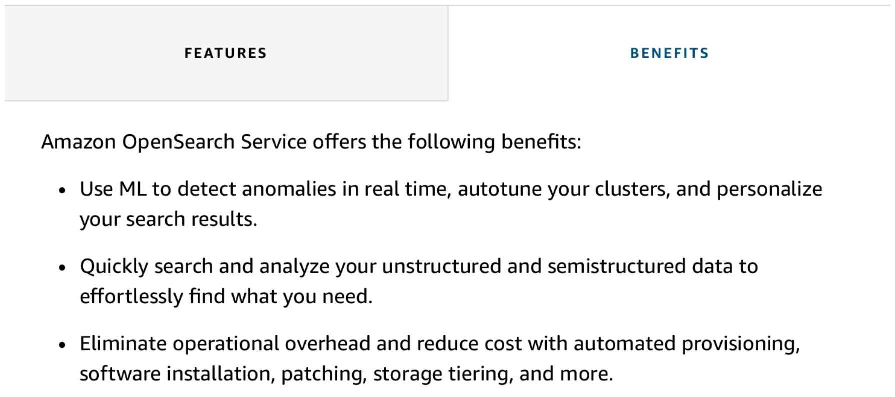


FEATURES | BENEFITS

Amazon DynamoDB offers the following benefits:

- Straightforward to get started and use
- Fully managed, serverless and highly scalable
- Built-in availability, durability, and fault tolerance that cannot be turned off
- Active-active data replication for multi-Region resiliency
- Security at code level and encryption of data at-rest

4. **Amazon OpenSearch Service:** OpenSearch is an open source, distributed search and analytics suitable for a broad set of use cases. OpenSearch provides a fast and highly scalable system for exploring and visualising data, with convenient OpenSearch Dashboards. Amazon OpenSearch Service supports integration with streaming data from Amazon S3 buckets, Amazon Kinesis Data Streams, and DynamoDB Streams.



FEATURES | BENEFITS

Amazon OpenSearch Service offers the following benefits:

- Use ML to detect anomalies in real time, autotune your clusters, and personalize your search results.
- Quickly search and analyze your unstructured and semistructured data to effortlessly find what you need.
- Eliminate operational overhead and reduce cost with automated provisioning, software installation, patching, storage tiering, and more.

3. Velocity

Indicates the speed of data flowing through to be processed.

Types of data processing:

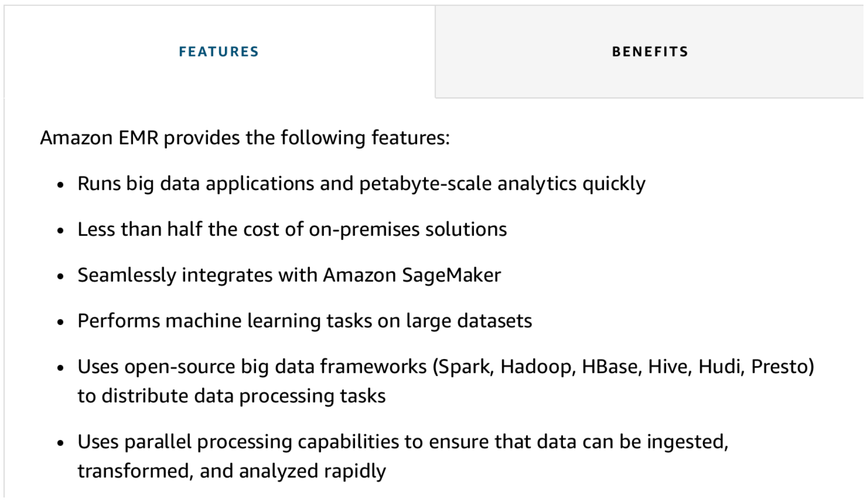
- Batch mode
- Stream mode

Four types of velocity to process data:

- *Scheduled:* it represents data that is processed in a very large volume on a regularly scheduled basis.
- *Periodic:* is a batch of data that is processed at irregular times.
- *Near real-time:* it represents streaming data that is processed in small individual batches. The batches are continuously collected and then processed within minutes of the data generation.
- *Real time:* it represents streaming data that is processed in very small individual batches. The batches are continuously collected and then processed within milliseconds of the data generation.

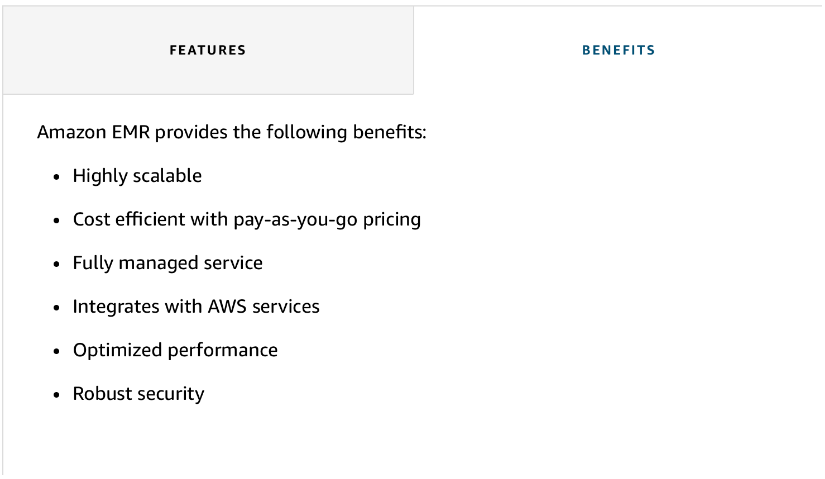
AWS Services for Velocity

1. **Amazon EMR:** Big data solution for petabyte-scale data processing, interactive analytics and ML.



Amazon EMR provides the following features:

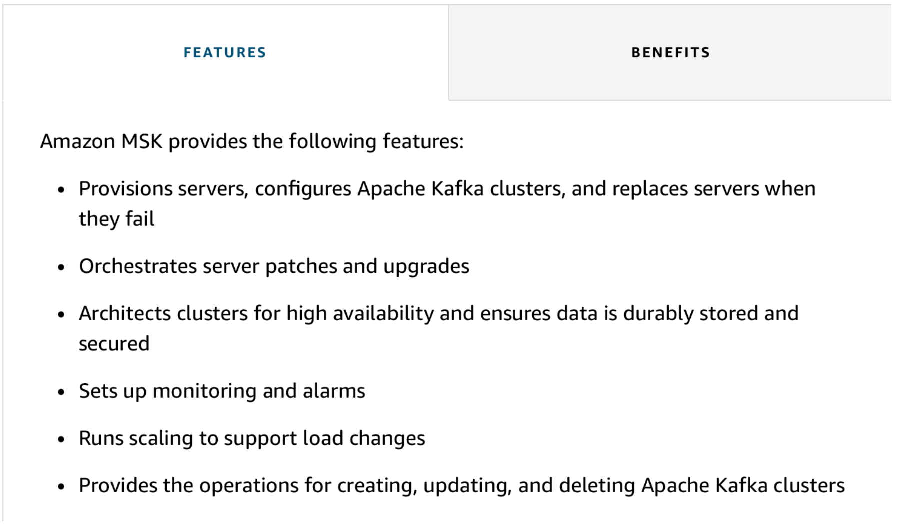
- Runs big data applications and petabyte-scale analytics quickly
- Less than half the cost of on-premises solutions
- Seamlessly integrates with Amazon SageMaker
- Performs machine learning tasks on large datasets
- Uses open-source big data frameworks (Spark, Hadoop, HBase, Hive, Hudi, Presto) to distribute data processing tasks
- Uses parallel processing capabilities to ensure that data can be ingested, transformed, and analyzed rapidly


FEATURES | BENEFITS

Amazon EMR provides the following benefits:

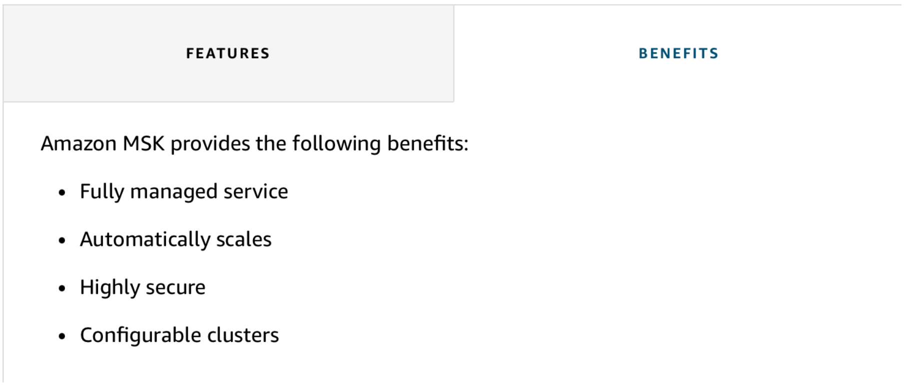
- Highly scalable
- Cost efficient with pay-as-you-go pricing
- Fully managed service
- Integrates with AWS services
- Optimized performance
- Robust security

2. **Amazon Managed Streaming for Apache Kafka (Amazon MSK):** Fully managed, highly available apache kafka service.


FEATURES | BENEFITS

Amazon MSK provides the following features:

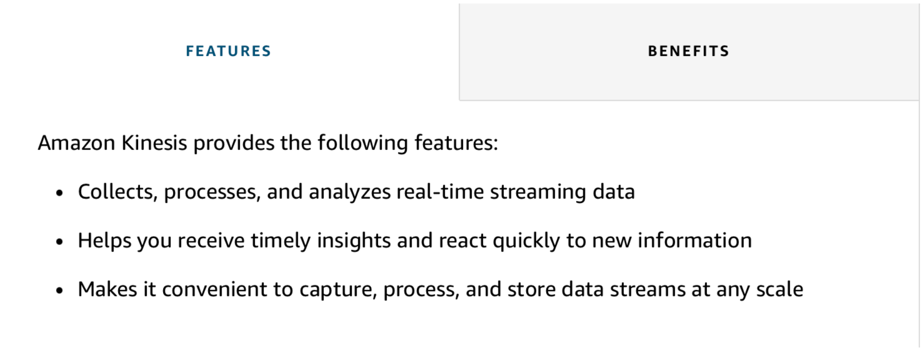
- Provisions servers, configures Apache Kafka clusters, and replaces servers when they fail
- Orchestrates server patches and upgrades
- Architects clusters for high availability and ensures data is durably stored and secured
- Sets up monitoring and alarms
- Runs scaling to support load changes
- Provides the operations for creating, updating, and deleting Apache Kafka clusters


FEATURES | BENEFITS

Amazon MSK provides the following benefits:

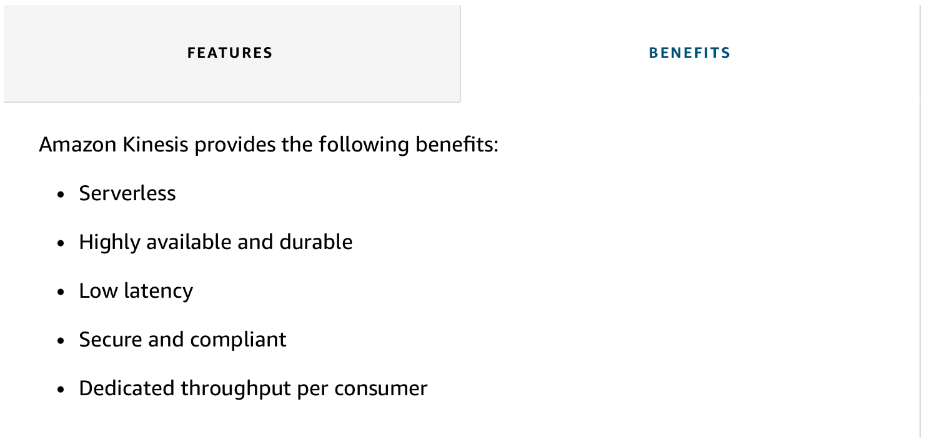
- Fully managed service
- Automatically scales
- Highly secure
- Configurable clusters

3. **Amazon Kinesis:** Amazon Kinesis Data Streams ingests and stores data from real-time streams so it can be prepared for analysis by downstream services. Amazon Kinesis Data Streams works with AWS Lambda to create event-driven, serverless workflows.


FEATURES | BENEFITS

Amazon Kinesis provides the following features:

- Collects, processes, and analyzes real-time streaming data
- Helps you receive timely insights and react quickly to new information
- Makes it convenient to capture, process, and store data streams at any scale



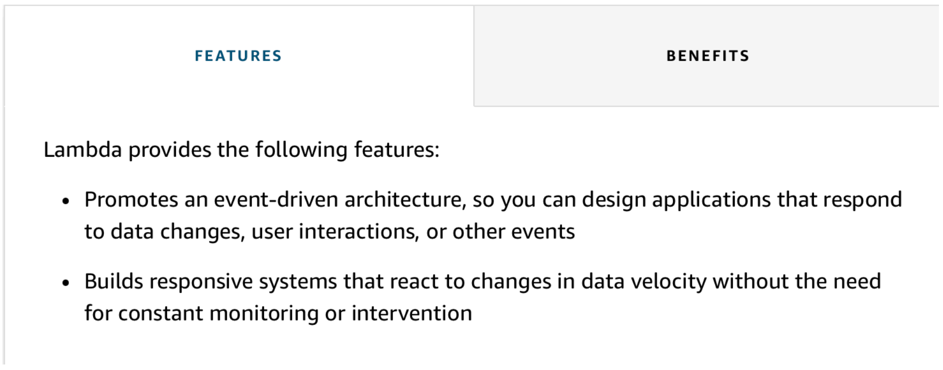
Amazon Kinesis provides the following benefits:

- Serverless
- Highly available and durable
- Low latency
- Secure and compliant
- Dedicated throughput per consumer

Data streams services on AWS:

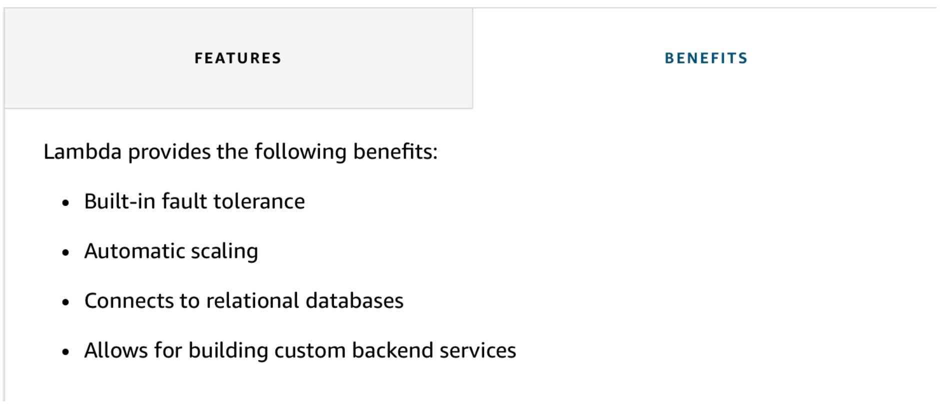
Service	Description
Amazon Kinesis Data Streams	A data streaming service that continuously captures data in real time from hundreds of thousands of sources.
Amazon Kinesis Data Firehose	Near real-time analytics with existing business intelligence tools by capturing, transforming, and loading data streams into AWS data stores.
Amazon Managed Service for Apache Flink	Build and run Apache Flink applications, and query and analyze streaming data without setting up infrastructure and clusters.

4. **AWS Lambda:** AWS Lambda is a serverless compute service that helps address velocity challenges in data processing with real-time and event-driven data processing. Events and changes in data can generate quick responses, without the need to manage infrastructure.



Lambda provides the following features:

- Promotes an event-driven architecture, so you can design applications that respond to data changes, user interactions, or other events
- Builds responsive systems that react to changes in data velocity without the need for constant monitoring or intervention



FEATURES | BENEFITS

Lambda provides the following benefits:

- Built-in fault tolerance
- Automatic scaling
- Connects to relational databases
- Allows for building custom backend services

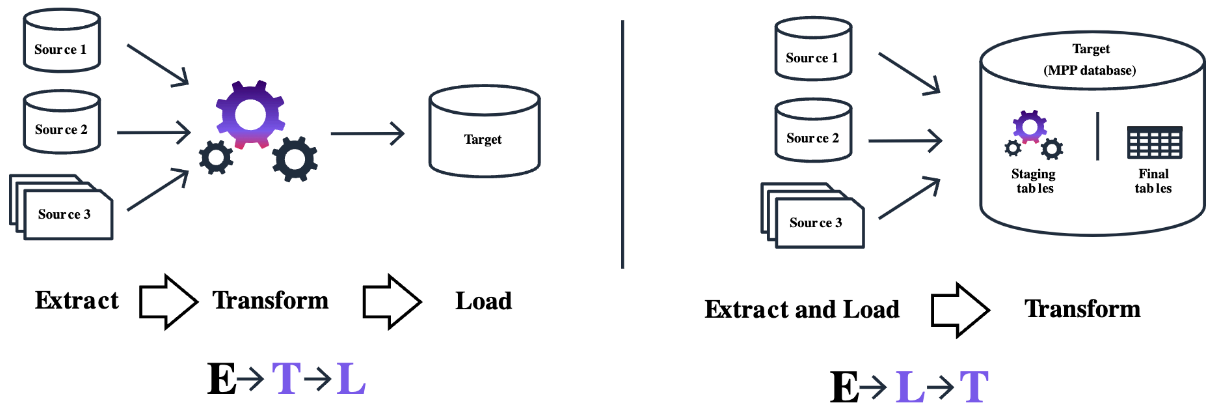
4. Veracity

Indicates how accurate, precise and trusted the data is.

Veracity deals with the trustworthiness of the data, and integrity is one of the key factors influencing that trustworthiness. Data integrity is applied in different ways in each phase of the data lifecycle. The data lifecycle includes creation, aggregation, storage, access, sharing, and archiving.

A comparison of the ETL and ELT processes

Data scientists mainly use ETL to load legacy databases in the data warehouse, and use ELT with modern databases.



AWS Services for Veracity

1. **Amazon EMR:** Amazon EMR helps you overcome data quality, accuracy, and integrity challenges. This service provides a robust data collection and processing platform to analyse vast amounts of data.
2. **AWS Glue:** AWS Glue helps you overcome data quality and data integrity challenges. AWS Glue is a serverless data integration and managed ETL service. AWS Glue can be used as a metastore for your final transformed data by using the AWS Glue Data Catalog. You can manage data quality in datasets with AWS Glue Data Quality. This service analyses your chosen dataset and recommends data quality rules that you can optimise.

3. **AWS Glue DataBrew:** t is a visual data preparation tool that helps data analysts and data scientists clean and normalise data to prepare it for analytics and ML.
4. **Amazon DataZone:** Amazon DataZone is a service designed to enable data sharing, discovery, and governance across an organisation. It helps in creating a data catalog where data producers can share data, and data consumers can discover and access data efficiently and securely.

---

5. Value

Indicates the ability to extract valuable information from the data that has been stored and analysed.

Types of reports used in data analytics:

- Dashbord
- Interactive report
- Static report

AWS Services for Value

1. Amazon Quicksight
2. **Amazon SageMaker:** Build, train and deploy ML models for any use case with fully managed infrastructure, tools and workflows. Amazon SageMaker JumpStart is an ML hub with foundation models, built-in algorithms, and prebuilt ML solutions.
3. Amazon Athena
4. **Amazon Bedrock:** Build and scale generative AI applications with foundation models.

Fundamentals of data analysis - 2

Data Lake

Data Lake is a centralised repository that an organisation can use to store data at scale, both structured and unstructured, in its original format. It delivers data ready to be consumed by data analytics by undergoing the raw data to cleaning processes with ETL tools.

In AWS environment, this can be done using AWS Lake Formation, to easily deploy a well governed and secure data lake infrastructure. Apart from this, with AWS, this centralised repository is normally a s3 bucket, in which stores the objects, that gain a schema with glue crawlers and can be manipulated with Athena.

Data Warehouse

A data warehouse is a central repository of information that is specially designed for analytics. Data flows into a data warehouse from business applications, databases, and other sources. Data warehouse users access the data through business intelligence (BI) tools, SQL clients, and other analytics applications. A data warehouse powers reports, dashboards, and analytics tools by storing data efficiently. It minimises the input and output (I/O) of data and delivers query results quickly to hundreds and thousands of users concurrently.

Amazon Redshift features:

Auto-copy from Amazon S3

The Amazon Redshift auto-copy from Amazon S3 feature automates data ingestion into Amazon Redshift. This feature continuously ingests data as soon as new objects are created in Amazon S3 with no custom coding or manual ingestion activities.

Streaming ingestion

The Amazon Redshift streaming ingestion feature ingests hundreds of megabytes of data per second from Amazon Kinesis Data Streams or Amazon Managed Streaming for Apache Kafka (Amazon MSK). You can define a schema or choose to ingest semi-structured data with the SUPER data type to query data in real time.

Aurora zero-ETL with Amazon Redshift

Amazon Aurora now supports zero-ETL integration with Amazon Redshift to enable near real-time analytics and ML using Amazon Redshift on petabytes of transactional data from Aurora. Within seconds of transactional data being written into Aurora, the data is available in Amazon Redshift. You don’t have to build and maintain complex data pipelines to perform ETL operations.

With this zero-ETL integration, you can analyze data from multiple Aurora database clusters in the same new or existing Amazon Redshift instance. You can derive holistic insights across many applications or partitions.

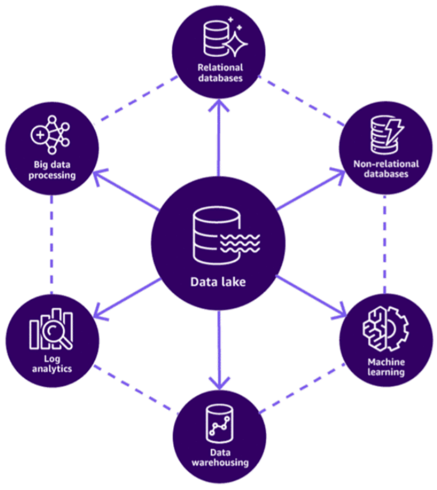
With near real-time access to transactional data, you can use capabilities of Amazon Redshift to derive insights from transactional and other data:

- Built-in ML
- Materialized views
- Data sharing
- Federated access to multiple data stores and data lakes

Modern Data Movements

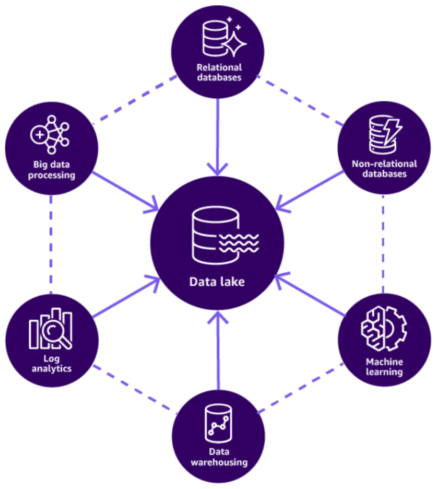
INSIDE-OUT	OUTSIDE-IN	AROUND THE PERIMETER	SHARING ACROSS

Inside-out data movement keeps data in a central repository like a data lake and moves compute to it rather than transferring data across systems. This approach minimizes unnecessary data movement. Key enablers are technologies like serverless functions, query engines, and orchestration tools that collect processing workloads with the source data. A shared data catalog provides metadata to enable querying in place. By analyzing, transforming, and moving data only when necessary, this pattern reduces duplication and movement costs while optimizing processing performance.



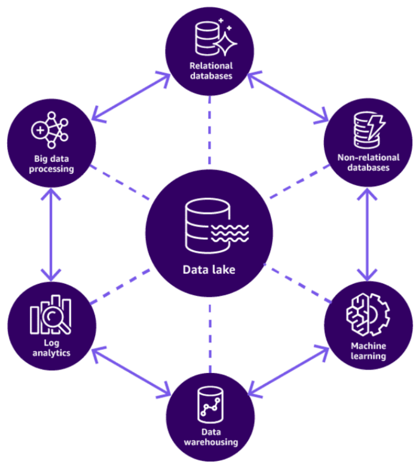
INSIDE-OUT	OUTSIDE-IN	AROUND THE PERIMETER	SHARING ACROSS

Outside-in data movement refers to the process of bringing data from purpose-built systems into an organization's data lake. An example is when a customer copies query results for regional product sales from their data warehouse into their data lake to run product recommendation algorithms against a larger dataset using ML.



INSIDE-OUT	OUTSIDE-IN	AROUND THE PERIMETER	SHARING ACROSS

Around the perimeter data movement is moving data from one purpose-built data store to another. An example is when a customer copies the product catalog data stored in their database to their search service. This makes it easier to look through their product catalog and offload the search queries from the database.



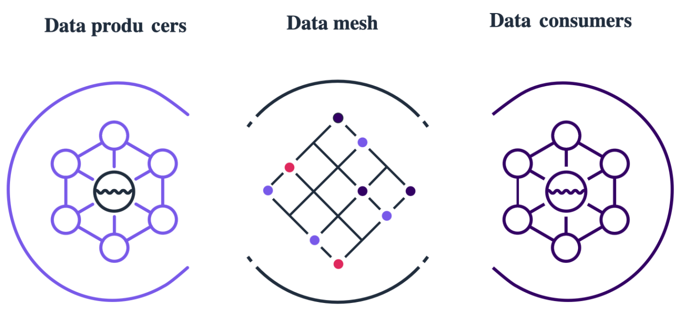
INSIDE-OUT	OUTSIDE-IN	AROUND THE PERIMETER	SHARING ACROSS

Sharing across data movement involves the seamless transfer of data between different applications. An example of sharing across data movement is using a data mesh architecture. Data as a product is a central design goal of data mesh. A business-aligned domain registers as a node in a mesh, publishes their data products to a central governance catalog, and discovers data products to consume through central governance services. The enterprise manages a central set of governance services to support enterprise-wide data discovery, reporting, and auditing.

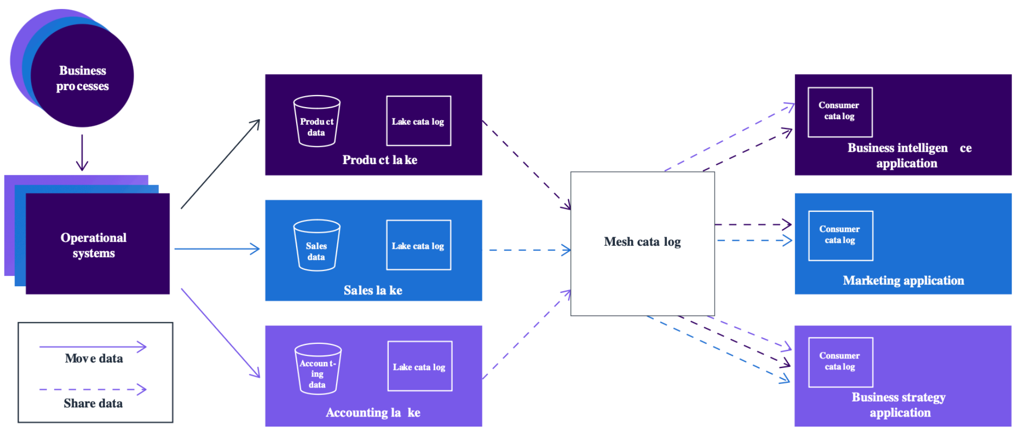
Data mesh: A data mesh architecture consists of data teams that build and run the platform. They build security controls, run the onboarding, and provide training.

Data producers: Data producers consist of business domain teams that want to share their data. They have domain expertise and manage data ownership and governance. They ensure data quality and manage the metadata to make it easy to find their data.

Data consumers: Data consumers make use of the data mesh to fit their business function. They want to be able to find data easily. Data consumers use data to run business priorities and develop business analytics to find new insights.



The following diagram illustrates a data mesh architecture.



Amazon Datazone: Amazon DataZone helps data producers to share data. It also helps data consumers find and consume the data they need and govern data with standards and policies in a data mesh architecture.

AWS Services for Modern Architectures:

1. For Scalable data lakes

- S3
- Lake Formation
- AWS Glue Data Catalog: centralised metadata catalog

2. For purpose built analytics

- Amazon Managed Service for Apache Flink
- Amazon QuickSight
- Amazon OpenSearch Service
- Amazon Redshift: By using federated queries in Amazon Redshift, you can query and analyse data across operational databases, data warehouses, and data lakes. With the Federated Query feature, you can integrate queries from Amazon Redshift on live data in external databases with queries across your Amazon Redshift and Amazon S3 environments.
- Amazon SageMaker: You can use Amazon SageMaker Canvas to generate accurate predictions without having to write code of your own. You can access prebuilt foundation models (FMs) from Amazon, publicly available FMs, or customise your own models. Use SageMaker Canvas for sentiment analysis, object detection, or document analysis. T
- Amazon EMR
- AWS Services for AI
- AWS Services for generative AI
- Amazon Athena
- Amazon RDS

- Amazon Aurora: Amazon Aurora is serverless and scalable MySQL and PostgreSQL compatible OLTP database. It is high performance, highly available, and fully managed. It has many integrations with other AWS services to perform analytics and extract insights.
- Amazon DynamoDB

3. For unified Data Access

- AWS Glue
- Amazon Kinesis Firehose

4. For unified data governance

- AWS Lake Formation: Use Lake Formation to centrally manage your security, access control, and audit trails. You can get **fine-grained row-level security** for your data.
- Amazon DataZone

Domain 1: Data Ingestion and Transformation

Data Engineering Fundamentals

Before diving into the domains and task statements for the AWS Data Engineering certification, it is important to understand the core fundamentals of data engineering and the role of a data engineer.

What is a Data Engineer?

A data engineer is responsible for acquiring, storing, and preparing data for consumption by data scientists, analysts, and other stakeholders. There are many definitions, but at its core, a data engineer focuses on:

- **Getting Data**: Acquiring data from various sources.
- **Storing Data**: Using appropriate storage solutions.
- **Preparing Data**: Transforming and organising data for analysis and machine learning.

What is Data Engineering?

Data engineering involves the development, implementation, and maintenance of systems and processes that ingest raw data and produce high-quality, consistent data for analysis, machine learning, and other uses. It encompasses several key areas:

- **Security**
- **Data Management**
- **Orchestration**
- **Data Architecture**
- **Software Engineering**
- **Operations**

The primary goal is to take raw data and make it easy and reliable to work with, integrating it across datasets and domains. To achieve this, data engineers use various methods, tools, and services such as:

- **Streaming**
- **Extract, Transform, and Load (ETL)**
- **Data Warehouses**
- **Data Lakes**

Data Pipelines

A crucial concept in data engineering is the **data pipeline**, which is a method for moving data from a source to a destination. As data moves through the pipeline, it can be transformed and optimised, ensuring that it is ready to be used when it reaches its destination.

AWS Data Engineering Tools

In this course, we will cover AWS data engineering tools that help build data pipelines, manage data transfer and transformation, and ensure efficient data storage. AWS offers a wide range of services that allow data engineers to create complex data analytic pipelines. Understanding these tools is essential for effective data engineering.

- It's important to note that there are multiple AWS services to achieve a specific outcome.
- As a data engineer, you need to choose the best services based on your requirements to build robust datasets that serve as the source of truth.

Data Engineering Lifecycle

The data engineering lifecycle focuses on the data itself and the end goals and requirements needed from that data. The key stages of the data engineering lifecycle are:

1. **Generation**: Creating or collecting raw data.
2. **Storage**: Efficiently storing the raw data.
3. **Ingestion**: Bringing data into the processing environment.
4. **Transformation**: Cleaning, transforming, and preparing the data.
5. **Serving**: Making the data available for use by analysts, data scientists, and applications.

Difference Between Data Engineering Lifecycle and Data Lifecycle

The **data engineering lifecycle** is a subset of the broader **data lifecycle** and focuses on the stages that a data engineer directly controls. In the entire data lifecycle, you must also consider:

- **Security**
- **Orchestration**
- **Data Architecture**
- **Other Aspects of Data Management**

Evaluating Data Tools and Their Integration

A critical aspect of data engineering is evaluating different data tools and understanding how they work together throughout the data lifecycle. You should also understand:

- **How Data is Produced**: Knowing the systems that generate data.
- **How Data is Consumed**: Understanding how analysts and scientists will use the data.

Managing and reviewing data pipelines is crucial to continually improve and optimise for costs, scalability, agility, reuse, and more.

Software Engineering in Data Engineering

Software engineering is a vital part of the data lifecycle. As a data engineer, you need to be familiar with:

- **Technologies**: Hadoop, Apache Spark, Teradata, Hive, etc.
- **Fundamentals**: Networking, distributed computing, storage, etc.
- **Programming Languages**: Proficiency in different programming languages is essential.

AWS Data Engineering Toolkit

Traditionally, big data processing systems were built in data centers. AWS now offers multiple services for data engineers to build complex data analytic pipelines, such as:

- **Data Transfer and Transformation Services**
- **Storage Solutions**
- **Data Processing Tools**

Knowing which AWS service to use for a specific requirement is crucial to building robust datasets that answer a wide range of questions.

1.1 Performing Data Ingestion

Before diving into the specifics of data ingestion, let's revisit the **Data Engineering Lifecycle**, which involves several stages: **generation, storage, ingestion, transformation, and serving**. In this section, we will briefly cover the first two stages (generation and storage) and then focus on **ingestion**, which is the key topic of this task statement.

1. Data Engineering Lifecycle Overview

1. **Generation Stage**:
   - This is where data originates—your source system. Examples include IoT devices, transactional databases, application message queues, etc.
   - Data engineers do not control the source system but must understand how it works:
     - **Characteristics**: Data types, schema, frequency, velocity, persistence, and potential duplicates.
     - **Considerations**: What is the schema of the source data? Does it change over time? What data quality issues might arise?
2. **Storage Stage**:
   - Selecting an appropriate storage solution is crucial for handling data ingested from the source system into AWS.

  - Different use cases require different AWS storage services (e.g., Amazon S3, Redshift).
  - This stage is covered in detail under **Domain 2: Data Store Management**.

2. Ingestion Stage

The ingestion stage involves gathering data from source systems and ingesting it into a suitable environment for processing and analysis.

- **Key Considerations for Ingestion**:
  - **Use Cases**: What are the use cases for the ingested data?
  - **Destination**: Where will this data go after ingestion?
  - **Frequency and Volume**: What is the frequency and volume of data being ingested?
  - **Format**: What format is the data in (e.g., JSON, CSV, TSV)?
- **Data Ingestion Concepts**:
  - **Batch vs. Streaming**: Understand the differences between batch and streaming ingestion.
  - **Push vs. Pull**: Know when to use a push model (data is pushed from the source) versus a pull model (data is pulled by the destination).
- **Replayability**:
  - **Event-Driven Design**: Consider designing ingestion pipelines in an event-driven manner to support replayability in case of failures.
  - **AWS Services for Event Collection**: Amazon S3, Amazon Kinesis, and Amazon EventBridge are useful for collecting and initiating events when new data arrives or changes occur.
  - **Testing and Validation**: Test the replayability of data pipelines using different scenarios to validate idempotence and avoid data loss, duplication, or integrity issues.
- **Best Practices for Ingestion Pipelines**:
  - **Checkpoint Mechanisms**: Track the progress and state of data processing to enable resumption from the last successful point.
  - **Data Versioning**: Store raw or intermediate data in scalable storage like Amazon S3 to retain data for compliance, governance, and replayability.
  - **Logging and Monitoring**: Capture logs, errors, and metrics to track progress and identify failures.
  - **Infrastructure as Code (IaC)**: Use tools like AWS CloudFormation or AWS CDK to automate the deployment and configuration of ingestion pipelines.

3. Steps to Perform Data Ingestion

Data ingestion involves several steps, starting with data producers and ending with data consumers:

- **Producers**: These generate data in the source system (e.g., databases, mobile devices).
- **Ingestion Tool**: This gathers data from producers.
- **Consumers**: These are the end points that use the ingested data (e.g., Amazon EC2, AWS Lambda, Amazon Redshift).

Example: Ingesting Data into AWS Using Amazon S3:

- Application writes customer interactions and reviews to an S3 bucket in TSV format.
- **Data Science Team**: Use **Amazon Athena** to query and analyse the data using SQL.
  - **Configuration**: Register the TSV data with Athena and run ad-hoc queries.
  - **Optimisation**: Convert TSV data to Apache Parquet for better performance.
- **Business Intelligence Team**: Insert TSV data into **Amazon Redshift** and use **Redshift Spectrum** to combine queries for data in the S3 lake.

Handling Large Data Volumes:

- Example: Ingest large daily gzip files into an Amazon Redshift cluster.
- **Optimisation**: Split the gzip file into smaller files to enable parallel processing and use distribution keys to improve load times.

4. AWS Services for Data Ingestion

AWS offers a variety of services for different ingestion scenarios:

- **Transactional Data Systems**:
  - **DynamoDB**, **Amazon RDS**: For quickly storing and retrieving small amounts of data.
  - **AWS Database Migration Service (DMS)**: To ingest transactional data and handle continuous replication.
- **Streaming Data Ingestion**:
  - **Amazon Kinesis** (Data Streams, Data Firehose, Video Streams): For real-time ingestion and processing.
  - **Amazon MSK** (Managed Streaming for Apache Kafka): For real-time analytics.
  - **Amazon AppFlow**: For ingesting data from SaaS services to S3 or Redshift.
  - **AWS Transfer Family**: For ingesting data using FTP/SFTP protocols.
  - **AWS DataSync**: For ingesting data from on-premises storage.
- **Batch Data Ingestion**:
  - **Amazon EMR**: Deploy Hadoop frameworks and use JDBC drivers to load data from relational databases.
  - **AWS Glue**: Fully managed ETL service that supports data processing and transfer.

5. Managing Stateful and Stateless Data Transactions

- **Stateful Transactions**: Store information about the current state.
  - **Examples**: **Amazon ElastiCache for Redis**, **Amazon RDS**.
- **Stateless Transactions**: Do not store session information or rely on past state.
  - **Examples**: **AWS Lambda**, **Amazon API Gateway**, **Amazon S3**.

6. Understanding the Five V's of Data

When determining an ingestion solution, consider the **Five V's of Data**:

1. **Variety**: Types of data (structured, unstructured, semi-structured).
   - AWS Services: Amazon RDS, S3, AWS Glue, Amazon Comprehend.

2. **Volume**: Amount of data to be transferred and stored.
    - AWS Services: S3, EBS, Redshift, DynamoDB.
3. **Velocity**: Speed of data ingestion and processing.
    - AWS Services: Kinesis, MSK, OpenSearch, Lambda.
4. **Veracity**: Quality, completeness, and accuracy of data.
    - Consider scenarios where data might be missing or inconsistent.
5. **Value**: Importance of data in providing business insights.
    - Focus on ensuring ingested data adds business value.

1.2 Transforming and Processing Ingested Data

This section focuses on the **fourth step in the data engineering lifecycle**, which is **transformation**. Up until this stage, we have generated data in the source system, chosen an appropriate storage solution, and ingested the data into that storage. Now, we need to transform this data to make it useful for downstream applications.

1. Understanding Data Transformation

Data Transformation is the process of converting raw data from its original form into a more usable format for analysis, reporting, machine learning, and other downstream use cases. Without proper transformation, data remains inert and unfit for meaningful insights.

- **Tools for Data Transformation**:
    - **Queries**: Fundamental for retrieving and manipulating data.
    - **Data Modelling**: Designing how data is structured and related to the real world.
    - **Transformations**: Applying various processes to make data useful for analysis.

Data Modelling is essential before building data systems. It helps structure data to represent real-world entities and relationships and ensures consistency and usability.

2. OLTP vs. OLAP Data Modelling and Querying

- **OLTP (Online Transaction Processing)**:
    - Optimised for real-time transactional processing and updates.
    - Uses normalised or denormalised data models.
    - Smaller storage requirements and shorter response times.
    - Ideal for order processing, payments, and customer data management.
- **OLAP (Online Analytical Processing)**:
    - Optimised for complex data analysis and reporting.
    - Uses star schema, snowflake schema, or other analytical models.
    - Large storage requirements, from terabytes to petabytes.
    - Suitable for predicting customer behaviour and trend analysis.

3. Why Transform Data?

Transforming data helps in:

- **Mapping Data Types**: Converting strings to numeric or date formats.

- **Standardising Formats**: Ensuring consistent formats across datasets.
- **Removing Bad Data**: Cleaning data by removing errors or inconsistencies.
- **Schema Changes and Aggregation**: Applying transformations like normalisation, aggregation, or feature engineering for machine learning.

Key considerations for data transformation include cost, return on investment, and business value.

4. Data Preparation and Wrangling

Data Preparation or **Data Wrangling** involves cleaning and organising raw data for analysis. This can include:

- **Adding Calculated Fields**
- **Applying Filters**
- **Changing Field Names or Data Types**
- **Joining Tables**: Using SQL queries for multi-table operations.

Light transformations can occur during ingestion using services like **Kinesis Data Firehose**, which can convert data formats. However, more complex transformations may require advanced processing tools.

5. AWS Services for Data Transformation and Processing

- **Serverless Processing**:
  - **AWS Glue**: A serverless ETL service that uses Python or Spark engines for transformations. It includes the **AWS Glue Data Catalog** for managing metadata.
- **Distributed Computing**:
  - **Amazon EMR**: Managed service for running big data frameworks like Spark, Hive, HBase, Presto, Pig, etc., for large-scale data processing.
  - **AWS Batch and AWS Step Functions**: For managing distributed computing tasks.
- **Data Processing Examples Using AWS Services**:
  - **AWS Lambda**: Suitable for lightweight transformations and validations. For example, a Lambda function can be triggered to process incoming CSV files, validate their format, and move them to another S3 bucket for batch processing.
  - **Amazon EMR with Spark**: Provides a managed Hadoop environment for running big data frameworks. You can configure clusters, write Spark applications, and use Spark's APIs for data processing tasks like aggregation, filtering, and machine learning.

6. Choosing Between AWS Glue and Amazon EMR

- **AWS Glue**: Ideal for serverless ETL tasks, data cleansing, enrichment, and data movement. Suitable when you need a managed solution without deep configuration.
- **Amazon EMR**: A more flexible and configurable environment for running Hadoop and Spark workloads. It provides better control over the processing environment.

7. Example Architecture for Data Transformation

- **Scenario**: Using an Apache web server on an Amazon EC2 instance to visualize data in Kibana.
    - **Kinesis Agent**: Installed on EC2 to send data to **Kinesis Data Firehose**.
    - **Kinesis Data Firehose**: Sends data to an S3 bucket for storage and Amazon Managed Service for Apache Flink for processing.
    - **Amazon Managed Service for Apache Flink**: Processes the data with SQL commands or machine learning models and sends the transformed data to another Firehose stream.
    - **Amazon OpenSearch Service**: Receives data from Firehose for indexing and visualisation in Kibana.

8. Orchestration in Data Transformations

Orchestration combines different operations such as transformations, transfers, and updates across multiple datasets and systems. It ensures efficient data movement, schema updates, and data wrangling.

9. Debugging and Optimising Data Transformations

- **Troubleshooting Steps**:
    - Check logs generated by AWS services.
    - Verify data quality and integrity.
    - Validate transformation logic and source data.
    - Use debugging modes in AWS Glue or smaller datasets for debugging.
- **Optimisation Techniques**:
    - Profile and analyse execution to identify bottlenecks.
    - Use efficient algorithms, partitioning strategies, and caching.
    - Implement incremental processing and retries for transient failures.

10. Enriching Data with AWS Services

Example scenario: Enriching logs from **CloudWatch Logs** with data from **DynamoDB** and using the output for further study.

- **Solution**:
    - Create a **Lambda function** to enrich logs with DynamoDB data.
    - Use **Kinesis Data Firehose** to collect and transform the enriched logs and store them in Amazon S3.

11. Connecting to Data Sources with JDBC and ODBC

- **JDBC (Java Database Connectivity)**: A Java API for connecting to relational databases in AWS. Requires JDBC driver, security configuration, and connection URL setup.
- **ODBC (Open Database Connectivity)**: A standardised API for connecting to data sources, including relational databases. Requires ODBC driver installation and DSN (Data Source Name) setup.

12. Creating Data APIs Using AWS Services

- **AWS Glue or Amazon EMR**: To perform data processing or transformation.
- **API Gateway**: To create and manage APIs for data access. It offers authentication, rate limiting, caching, and request/response transformation.
- **AWS Lambda**: For processing and serving data via API Gateway.
- **Security and Authentication**: Implement measures using IAM, AWS Certificate Manager, API keys, OAuth, etc.
- **Caching and Performance**: Use ElastiCache or API Gateway caching features to optimize performance.
- **Monitoring and Scalability**: Use CloudWatch for monitoring and API Gateway configurations for scaling.

1.3 Orchestrating Data Pipelines

Orchestrating data pipelines is a key task in managing data architecture, which involves designing systems that support evolving data needs. In this section, we'll cover key concepts and AWS services for orchestrating data pipelines, including best practices for building scalable, resilient, and efficient pipelines.

1. Understanding Data Architecture

- **Data Architecture**: The design of systems to support an organisation's data needs, including both operational (functional requirements) and technical (how data is ingested, stored, transformed, and served) aspects.
- **Considerations**:
  - Monolithic vs. Microservices
  - Single vs. Multi-Tenant
  - Event-Driven Architectures, Serverless, Containers
  - Performance, Security, Control, Flexibility, and Cost

For further reading on data architecture, explore the AWS Well-Architected Framework and its six pillars.

2. Types of Data Architectures

- **Common Data Architectures**:
  - **Data Warehouse**: Centralized repository for structured data, optimized for querying and reporting.
  - **Data Lake**: Stores raw, unprocessed data in its original format, suitable for large-scale analytics.
  - **Data Lakehouse**: Combines elements of data lakes and warehouses to support both structured and unstructured data.
  - **Lambda Architecture**: Combines batch and real-time processing.
  - **Kappa Architecture**: Focuses solely on real-time processing.
  - **Data Mesh**: Decentralized architecture that treats data as a product, managed by domain-oriented teams.

3. What is a Data Pipeline?

- **Data Pipeline**: A collection of data processing tasks that are run in a specific order, potentially involving multiple stages of transformation and enrichment.
- **Workflow**: Defines the sequencing of tasks, which can be run sequentially or in parallel, depending on dependencies and requirements.

4. Orchestration in Data Pipelines

- **Data Orchestration**: Ensures that data flows properly from ingestion to processing and storage, managing dependencies and execution order.
- **Core Concepts for Orchestration**:
  - Task sequencing and dependencies
  - Automation of task execution
  - Error handling and retries
  - Monitoring and alerting

5. AWS Services for Orchestrating Data Pipelines

- **AWS Step Functions**: A serverless orchestration service that uses state machines to define workflows, handle errors, and manage retries.
  - **Use Cases**: Orchestrating complex workflows, managing retries, integrating with AWS services like Lambda, EMR, and more.
  - **Example Workflow**: Initiate with Step Functions, trigger Lambda to run Spark jobs on EMR, wait for job status updates, and handle success or failure states.
- **AWS Data Pipeline**: A managed service for scheduling and running data workflows, allowing ETL tasks between AWS and on-premises data sources.
  - **Use Cases**: Simple ETL tasks, data movement between AWS services, batch processing.
  - **Integration**: Supports JDBC connections to read/write data from other data stores like on-premises databases.
- **AWS Glue Workflows**: Part of AWS Glue, it builds and orchestrates ETL jobs involving AWS Glue components such as crawlers, jobs, and triggers.
  - **Use Cases**: Ideal when working solely with AWS Glue components for data processing.
  - **Example Workflow**: Use Glue crawlers to catalog data, run Glue jobs for data transformation, and handle notifications on job success or failure.
- **Amazon Managed Workflows for Apache Airflow (MWAA)**: A managed version of Apache Airflow that allows defining complex workflows as Directed Acyclic Graphs (DAGs).
  - **Use Cases**: Suitable for more complex data pipelines that involve multiple tools and services.
  - **Example Workflow**: Define DAGs in Python to run tasks like Glue jobs, Lambda functions, or custom scripts.

6. Pipeline Triggering and Scheduling

- **Triggering Pipelines**:
    - **Event-Driven**: Pipelines that run in response to events, such as new data uploads to S3.
    - **Schedule-Based**: Pipelines that run at regular intervals (e.g., daily, weekly).
- **Handling Pipeline Failures**:
    - **Common Failures**:
        - Data Quality Issues: Incorrect file formats or corrupt data.
        - Code Errors: Logic or syntax errors introduced in updates.
        - Endpoint Errors: Network or permission issues.
        - Dependency Errors: Failures due to upstream dependencies.
    - **Failure Handling**: Use orchestration tools like Step Functions to configure retry strategies, specify backoff rates, and handle errors gracefully.

7. Monitoring and Notifications

- **Monitoring**: Use CloudWatch to monitor pipeline health, performance, and detect failures.
- **Notifications**: Use Amazon SNS to send alerts for pipeline failures or issues.
    - **Example**: Configure SNS topics to send email notifications when a Step Function state machine fails.

8. Best Practices for Building Data Pipelines

- **Use Distributed Processing**: Leverage frameworks like Spark on EMR or AWS Glue for large-scale data processing.
- **Implement Auto Scaling**: Ensure optimal performance and cost efficiency by scaling resources automatically based on demand.
- **Partition Data**: Use data partitioning techniques to distribute workloads evenly and improve processing times.
- **Use Fault-Tolerant Storage**: Store data in reliable services like Amazon S3 or Amazon EFS.
- **Implement Backups**: Use AWS Backup or S3 versioning to safeguard data.
- **Error Handling and Retries**: Define robust error handling and retry mechanisms in orchestration tools like Step Functions.
- **Data Validation and Quality Checks**: Implement checks and automated testing to ensure data integrity.
- **CI/CD for Pipelines**: Use continuous integration and deployment practices to manage and deploy pipeline changes efficiently.

9. AWS Services for Different Complexity Levels

- **Less Complex Environments**:
    - **AWS Data Pipeline**: Simpler orchestration needs, less complex workflows.
    - **Step Functions**: For simple workflows with basic state transitions and error handling.

- **More Complex Environments**:
    - **AWS Glue Workflows**: For ETL pipelines involving multiple Glue components.
    - **Amazon MWAA**: For complex, multi-step workflows that benefit from DAG definitions in Apache Airflow.

10. Example Use Case: Orchestrating a Data Lake Pipeline

- **Scenario**: Building a serverless data lake with Amazon S3 as the primary store.
    - **Ingestion**: Use Kinesis Data Firehose for streaming data, AWS DMS for relational data, and DataSync for on-premises files.
    - **Cataloging**: Use AWS Glue crawlers triggered by Lambda functions to catalog new data.
    - **Processing**: Use AWS Glue ETL jobs to transform data into Parquet format and store it in a processed zone.
    - **Monitoring**: Set up CloudWatch events and SNS notifications for job status updates.

1.4 Applying Programming Concepts in Data Pipelines

This section focuses on applying programming concepts to make data pipelines efficient and scalable. We'll explore how data engineers can use programming tools and best practices in AWS to optimise data transformations, processing, and orchestration.

1. Overview of Data Pipelines and ETL Processes

- **ETL Process**: The ETL (Extract, Transform, Load) process involves extracting data from various sources, transforming it into a useful format, and loading it into a data warehouse, data lake, or data mart. Combining these steps operationalises and automates data processing.

- **AWS Engines for Data Transformations**:
    - **SQL**: Standard, widely known language for data transformation, particularly suitable when working in environments focused on SQL.
    - **Spark**: More versatile for complex data processing requirements, including high throughput and low latency scenarios.

2. Using SQL vs. Spark for Data Transformations

- **SQL**:
    - Best for environments that heavily rely on relational databases and data warehouses like Amazon Redshift.
    - Suitable when transformations are performed directly within the data warehouse (ELT approach).
- **Spark**:
    - Supports complex data processing, machine learning, and streaming.
    - Can be used in batch and real-time processing scenarios.

    - Can run on AWS Glue (serverless), Amazon EMR (managed cluster), ECS, EKS, or third-party services like Databricks.
- **Choosing Between SQL and Spark**:
    - Use **SQL** for transformations directly in Amazon Redshift using the ELT approach.
    - Use **Spark** for offloading transformations to a Spark cluster, which allows the data warehouse to focus on query performance for end users.

3. Optimising Data Processing with AWS Engines

- **Amazon Redshift**: Uses SQL for querying and optimising data transformations directly within the warehouse.

- **Amazon EMR**: Processes large datasets using frameworks like Hadoop, Spark, HBase, and Presto on customisable clusters.

- **AWS Data Pipeline**: Defines and schedules data workflows across AWS services and on-premises resources, including tasks like data movement and transformations.

- **AWS Glue Studio**: Provides a visual interface for designing ETL jobs with SQL-based transformations, useful for users without coding skills in Spark.

- **AWS Glue DataBrew**: Helps in applying transformations directly to data files, simplifying data preparation tasks.

4. Optimising Code for Data Ingestion and Transformation

Strategies to Optimize Code:

1. **Parallel Processing**: Distribute workloads across multiple compute resources to speed up processing.
2. **Batch Processing**: Reduce overhead by processing large data volumes in batches.
3. **Optimised Data Formats**: Use columnar formats like Parquet or ORC for better compression and faster data access.
4. **Filtering and Partitioning**: Load and process only necessary data, partitioned by relevant attributes (e.g., date, region).
5. **Compression**: Use built-in compression options in AWS services to reduce storage needs and transfer speeds.
6. **Resource Allocation**: Tune memory and parallelism settings to optimise resource usage and prevent bottlenecks.
7. **Query Optimisation**: Implement indexing, query rewriting, and incremental processing to improve performance.

Monitoring and Automation:

- Use AWS CloudWatch to monitor and optimise data processing performance.
- Implement Infrastructure as Code (IaC) with AWS CloudFormation, AWS CDK, or AWS SAM for automated and repeatable deployments.
- Use CI/CD pipelines for automated testing and deployment of data pipelines.

5. Applying Programming Concepts in AWS Services

- **AWS Lambda**: Write functions to handle data ingestion tasks like parsing and transforming data on new uploads to S3.
- **Amazon EMR and AWS Glue**: Use Python or Scala with Spark to define distributed data processing jobs.
- **Amazon Kinesis**: Real-time data processing using the Kinesis Data Streams API with AWS SDKs.
- **AWS Glue ETL Framework**: Write custom Python scripts to perform complex transformations like data cleansing, aggregation, and normalisation.
- **AWS Step Functions**: Build serverless workflows using state machines to manage orchestration and conditions across AWS services.

6. Optimising AWS Lambda Functions for Data Processing

- **Best Practices for AWS Lambda**:
  - Use AWS SDKs and libraries for optimised code.
  - Allocate appropriate memory and CPU resources based on workload.
  - Configure concurrency settings to match performance requirements.
  - Use asynchronous invocations and provision concurrency to reduce cold starts.
  - Monitor function performance and optimise configurations to reduce latency and increase throughput.
  - Consider potential latency when accessing resources in Amazon VPC.

7. Handling Failures and Retry Strategies

- **Common Pipeline Failures**:
  - **Data Quality Issues**: Incorrect file formats or corrupt data causing processing errors.
  - **Code Errors**: Syntax or logic errors introduced during updates.
  - **Endpoint Errors**: Network issues or permission errors affecting data flow.
  - **Dependency Errors**: Failures due to dependencies within or between pipelines.
- **Failure Handling**:
  - Configure retries with exponential backoff rates in orchestration tools like Step Functions.
  - Use AWS SNS to send alerts on pipeline failures or issues, ensuring timely response and resolution.

8. Understanding Data Structures and Algorithms

- **Importance in Data Engineering**:
  - **Data Structures**: Organise and store data efficiently (e.g., databases, distributed file systems, hash tables).
  - **Algorithms**: Perform operations like sorting, searching, or processing data (e.g., MapReduce for distributed data processing).
- **Optimising Data Pipelines**:

    - Choose appropriate data structures and algorithms to enhance performance and scalability.
    - Use techniques like partitioning, caching, and parallel processing to improve data processing efficiency.

Example: Using Amazon EMR with HDFS and MapReduce to process large datasets and extract insights, optimising performance through data partitioning and parallel processing.

9. Examples of Applying Programming Concepts in AWS

- **Data Ingestion**: Use Lambda to trigger data ingestion workflows, parse and transform data, and load it into data stores.
- **Data Processing**: Use Amazon EMR with Spark or AWS Glue to run distributed data processing tasks on large datasets.
- **Real-Time Processing**: Implement Kinesis for real-time data streams and processing using custom code with AWS SDKs.
- **Data Transformation**: Use AWS Glue to perform complex transformations and data preparation with Python-based ETL scripts.
- **Storage and Retrieval**: Utilize Amazon RDS, Redshift, and S3 for storing structured, semi-structured, and unstructured data, accessing them using SQL or AWS SDKs.
- **Workflow Orchestration**: Use Step Functions to define and manage complex workflows across multiple AWS services.

Domain 2: Data Store Management

2.1 Choosing the Right Data Store

Introduction: Importance of Choosing the Right Storage Solution

As a data engineer, selecting the appropriate data store is crucial. This involves understanding the different AWS storage platforms and options, as well as the underlying components, characteristics, performance considerations, durability, and costs associated with each. The first step in choosing the proper storage solution for your data architecture is knowing the use cases, the way data is stored, and how it will be retrieved.

Overview of AWS Storage Services

AWS offers various storage services, each with unique characteristics designed to meet different use cases:

- **Amazon S3 (Object Storage):**
    - **Characteristics:** Highly available, scalable, and durable.
    - **Use Cases:** Storing static content, backups, media files, data lakes, etc.

- **Amazon EFS (File Storage):**
    - **Characteristics:** Fully managed, scalable file storage service that can be shared across multiple EC2 instances.
    - **Use Cases:** Shared file storage across multiple instances, web hosting, content management systems, big data processing, etc.
    - **Other File Storage Options:** Amazon FSx for Windows File Server, Lustre, NetApp ONTAP.
- **Amazon EBS (Block Storage):**
    - **Characteristics:** Provides persistent block-level storage volumes attached to EC2 instances.
    - **Use Cases:** Running databases, transactional applications, applications requiring low-latency, high-performance storage.
- **Amazon RDS (Managed Relational Database Service):**
    - **Characteristics:** Managed service for MySQL, PostgreSQL, Oracle, SQL Server, etc.
    - **Use Cases:** Hosting and managing relational databases.
- **DynamoDB (Managed NoSQL Database Service):**
    - **Characteristics:** Offers single-digit millisecond latency at any scale.
    - **Use Cases:** Applications requiring low latency read/write operations, real-time data, gaming, IoT, dynamic scaling.
- **Amazon Redshift (Managed Data Warehouse Service):**
    - **Characteristics:** Petabyte-scale data warehouse service for analytics and business intelligence workloads.
    - **Use Cases:** Storing and managing structured data, handling large volumes of high-velocity semi-structured and unstructured data.
- **AWS Lake Formation:**
    - **Characteristics:** Managed service to build, secure, and manage data lakes.
    - **Use Cases:** Combining structured and unstructured data into a central repository, crawling, cataloging, and preparing data for analytics.

Factors to Consider When Choosing a Data Store

When selecting a data store, consider the following factors:

- **Performance Requirements:** Input/output operations per second (IOPS), scalability, durability, and data access patterns.
- **Specific Use Cases:** For example, Amazon FSx for Lustre is ideal for high-performance computing, data analytics, and machine learning workloads that require large-scale parallel data processing.

Case Studies and Questions

1. **High-Performance Parallel File System:**
    - **AWS Service:** Amazon FSx for Lustre.

    - **Use Cases:** High-performance computing, data analytics, machine learning with large-scale parallel data processing.
2. **In-Memory Caching and Low Latency Access:**
    - **AWS Service:** ElastiCache.
    - **Use Cases:** Caching frequently accessed data, session storage, real-time analytics.
3. **Using Amazon Redshift for Data Warehousing:**
    - **Scenario:** Need to store 30 TB of uncompressed data, with thousands of aggregated queries daily, and complex joins involving a small subset of columns.
    - **Solution:** Amazon Redshift is ideal due to its columnar data storage, which optimises query performance by reducing input/output, and its support for SQL joins.
4. **Data Migration to Amazon Redshift:**
    - **Scenario:** Migrating data into a Redshift cluster table.
    - **Solution:** Use the `COPY` command in Redshift to load data from sources like S3 or EMR. Handle data conversions using supported parameters to avoid errors during migration.
5. **Optimising Analytical Workloads with Amazon Redshift Spectrum:**
    - **Scenario:** Storing terabytes of historical data in Amazon S3 and running analytics on recent data stored in Amazon RDS.
    - **Solution:** Export older data to S3, run daily jobs to export current data from RDS to Redshift, and use Redshift Spectrum for queries that join older and recent data.
6. **Managing Petabytes of Data as Key-Value Pairs:**
    - **Scenario:** Need to process and analyse petabytes of data stored as key-value pairs with immediate access.
    - **Solution:** Analyse data with Hive on Amazon EMR and store the results in a DynamoDB table, using a SQL-like interface.

Managing Data Access and Permissions

- **Security and Access Management:**
    - **IAM:** Control access to clusters and resources by creating IAM users with specific permissions and roles.
    - **Security Groups and KMS:** Add encryption and manage encryption keys using AWS Key Management Service (KMS) for added data protection.
- **Parameter Groups:**
    - **Use Case:** Configure database parameters, including locking mechanisms to control concurrency and prevent data conflicts.
- **Database Auditing:**
    - **Use Case:** Track and log access and changes to the database, monitor unauthorised access attempts, and configure recurring monitoring schedules.

2.2 Understanding Data Cataloging Systems

Introduction: Importance of Data Catalogs

Data cataloging is crucial for managing and organising data within an organisation. It helps to ensure that data, which might be stored across various locations, is accessible, understandable, and usable for analysis and decision-making. A data catalog serves as a central repository for metadata, storing information about the data's location, schema, description, and runtime metrics. This centralisation facilitates tasks like writing federated queries, performing ETL operations, and more.

Key Concepts in Data Cataloging

1. **What is a Data Catalog?**
   - A data catalog is a metadata repository that provides information about the data, such as its location, schema, and runtime metrics. It plays a crucial role in organising data for easy discovery and use, especially in complex data environments where data is spread across multiple sources.
2. **Importance of Data Catalogs in Data Engineering**
   - As a data engineer, you need to set up and maintain data catalogs that integrate with various data pipelines and storage systems. This ensures that data is well-organised, secure, and easily accessible for analysis.
3. **Data Security and Governance**
   - Even with efficient data pipelines and the best data consumption tools, data is only valuable if it is secure and governed correctly. Ensuring that data catalogs are integrated with security and governance frameworks is crucial for maintaining data quality and trust.

Setting Up and Managing Data Catalogs

1. **Data Catalogs in AWS Glue**
   - **Purpose:** The AWS Glue Data Catalog serves as an index to the location, schema, and runtime metrics of your data. It supports ETL jobs and allows for easy querying of data using services like Athena.
   - **Components:**
     - Metadata Tables: Each table specifies a single data store, with metadata about the data's structure.
     - Crawlers: Typically used to automate the discovery of data and update the Data Catalog.
2. **Using Athena with Data Catalogs**
   - Athena queries data stored in a data lake using the databases and table definitions in the Data Catalog. The catalog provides details like the S3 location of data files, file format types, and partition information.
3. **Role of AWS Lake Formation**
   - **Key Features:**
     - Provides an interface for the Data Catalog.
     - Adds key-value properties at the column level, enhancing data management and security.

        - Enables role-based access controls and cross-account data access for complex data lake environments.

Examples and Practical Applications

1. **Role-Based Access Control with Lake Formation**
    - **Scenario:** You have a central data lake in Amazon S3 that holds data for four AWS accounts. You need to ensure role-based access so that each business unit can only access their data.
    - **Solution:**
        - Build a data lake in each account, catalog the data across multiple accounts to a central account using Lake Formation, update S3 bucket policies with the Lake Formation service-linked role, and use Lake Formation permissions to manage access.
2. **Ingesting and Cataloging Data Automatically**
    - **Scenario:** After new data ingestion, you want to automatically update the Data Catalog.
    - **Solution:** Use an AWS Glue crawler to scan the data source and update the Data Catalog with the latest metadata.
3. **Synchronising Partitions with Data Catalog**
    - **Scenario:** You have large datasets in Amazon S3, partitioned by criteria like date or region.
    - **Solution:**
        - Create a database in AWS Glue.
        - Configure a crawler for the partitioned data source.
        - Define the partitioning schema.
        - Run and schedule the crawler to keep the Data Catalog updated.
        - Use Athena to query the data based on partition criteria efficiently.

Creating Source or Target Connections in AWS Glue

1. **Purpose:**
    - Managing data assets by creating connections to various data sources or targets within your AWS environment.
2. **Examples:**
    - **Connecting to an S3 Bucket:** Provide the S3 bucket path and optional patterns.
    - **Connecting to Amazon RDS:** Provide the database endpoint, port, credentials, and other relevant information.
3. **Using Connections:**
    - After creating a connection, it can be used in AWS Glue crawlers or ETL jobs to access and catalog data, ensuring that your data is organised and discoverable.

Understanding Metadata and Data Catalog Components

1. **Metadata Components:**
    - **Schema Information:** Structure of the data, including data types and field names.
    - **Data Source Information:** Origin of the data, such as a database or data lake.
    - **Timestamps:** Creation and modification times.
    - **Data Quality Metrics:** Information on data accuracy, completeness, and consistency.
    - **Access Controls:** Define who owns the data and who can access it.
2. **Data Catalog Components:**
    - **Tables and Databases:** Stores metadata about data structures.
    - **Partitioning Information:** Helps in efficient data querying.
    - **Statistics and Metadata:** Automatically collected by AWS Glue crawlers.
    - **Data Lineage:** Tracks data flow between datasets.
    - **Tagging and Labels:** For categorisation and search.
    - **Integration with Analytics Tools:** Allows users to work with the data.

Security and Compliance in Data Cataloging

1. **Classifying Data Based on Sensitivity:**
    - **Data Classification:** Categorising data into different levels of sensitivity to apply appropriate security controls.
    - **Tools and Services:**
        - **IAM:** Controls access to AWS services and data.
        - **AWS KMS:** Manages encryption keys for data security.
        - **Amazon VPC and Security Groups:** Isolates and controls resource access.
        - **Service Control Policies (SCPs):** Manages permissions within AWS Organizations.
        - **Amazon Macie:** Automatically discovers, classifies, and protects sensitive data.
2. **Implementing Security and Compliance Measures:**
    - Use a combination of IAM roles, encryption, VPCs, security groups, and Macie to ensure data security and compliance with organisational policies.

2.3 Managing the Lifecycle of Data

In the introduction to Domain 2, we discussed the importance of data management throughout the data lifecycle, from the source system to its final stage. This brings us to the third task statement for Domain 2: managing the lifecycle of data.

Data Management During Transformation

Transformation is a critical stage in data management because it creates new datasets that will also need to be managed. It's essential to consider the following questions:

- How important is this data?
- How long should I keep this data?
- How often will this data be accessed?

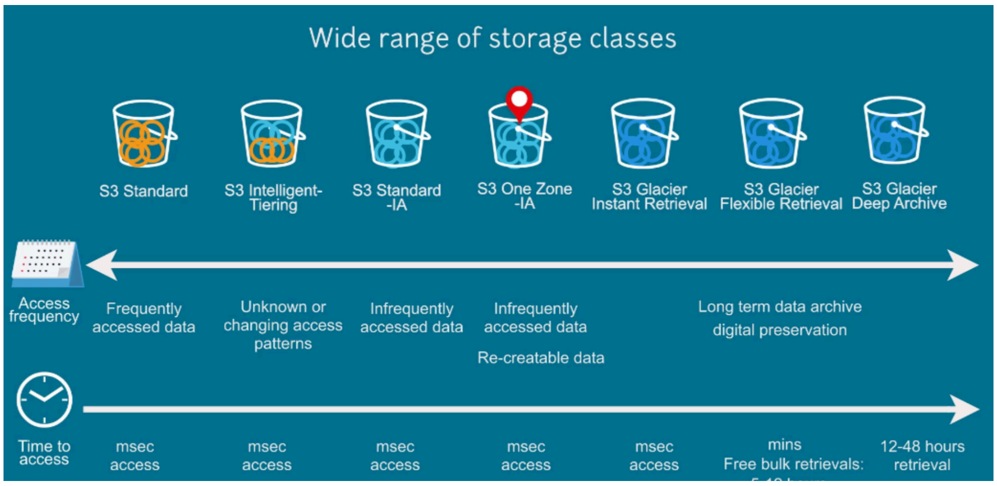
The answers to these questions will guide your decisions regarding storage, access, and cost optimisation.

Storage Categories Based on Access Frequency

Query access patterns differ for datasets depending on how frequently the data is accessed. Storage can generally be categorised into three persistence levels:

1. **Hot Storage** - For frequently accessed, newer data.
2. **Warm Storage** - For data accessed less often.
3. **Cold Storage** - For infrequently accessed, older data.

When selecting the appropriate storage tier, it's vital to consider both the cost and the retrieval time. Storing all data as hot data ensures quick access but comes at a higher cost. Conversely, cold storage reduces costs but increases retrieval times and may have higher access costs.


Wide range of storage classes
S3 Standard
S3 Intelligent-Tiering
S3 Standard -IA
S3 One Zone -IA
S3 Glacier Instant Retrieval
S3 Glacier Flexible Retrieval
S3 Glacier Deep Archive
Access frequency
Frequently accessed data
Unknown or changing access patterns
Infrequently accessed data
Infrequently accessed data
Re-creatable data
Long term data archive digital preservation
Time to access
msec access
msec access
msec access
msec access
msec access
mins Free bulk retrievals:
12-48 hours retrieval

Optimising Storage Costs

To optimise storage costs based on the data lifecycle, AWS offers several tools and features:

- **Amazon S3 Lifecycle Policies**: Use these policies to automatically transition data between storage classes (e.g., from S3 Standard to S3 Glacier) based on the data's age or to delete data to meet business and legal requirements.
- **Amazon Redshift**: You can resize your Redshift cluster, adding or removing nodes or changing node types, to optimise your storage and performance needs as they evolve. Elastic resizing allows you to adjust your cluster without downtime.

This brings us back to an important question: What data do you need to keep, and for how long? Proper data classification is crucial because it helps determine the value of the data you're storing, how it's protected, and who has access to it. Not all data is created equal, so ensuring the correct classification is vital for security.

Data Retention and Regulatory Compliance

Data retention policies must consider whether the data can be recreated and how long it needs to be stored. Regulatory and compliance requirements might dictate that certain data must be retained for a specific period. Archiving strategies and data retention policies must be in place to store and retrieve data as needed to meet these requirements.

Ensuring Data Protection, Resiliency, and Availability

To ensure your data is protected and has appropriate resiliency and availability, implement a combination of best practices and AWS services. AWS offers various services to replicate data across multiple Availability Zones or Regions, ensuring high availability and durability. For example:

- **Amazon S3**: Automatically replicates data across multiple Availability Zones.
- **Amazon RDS**: Supports multi-AZ configurations for automatic failover.

Data protection also includes backup and restore strategies, such as using AWS Backup or configuring automatic snapshots for your databases. Another option is to replicate data across Regions and use versioning in Amazon S3 to protect against accidental deletions or overrides.

Data Lifecycle Management in Practice

Consider a scenario where you have data stored in Amazon S3 that becomes infrequently accessed after six months but must be archived after two years due to regulatory requirements. To optimise storage costs:

1. Use an AWS Glue ETL job to compress, partition, and transform raw data into a columnar format.
2. Query this processed data using Athena.
3. Create an S3 lifecycle policy to move the data from S3 Standard to S3 Standard-Infrequent Access after six months.
4. After two years, transfer the data to Amazon S3 Glacier for long-term archival.

Another practical scenario involves using DynamoDB's time-to-live feature to automatically delete items from a table after a specific timestamp, helping manage data with expiration dates or cleaning up old data.

When managing large-scale datasets, such as those stored in Amazon Redshift, you might notice query performance issues as the data grows. Organising data into time series tables and dropping older tables can improve performance by eliminating the need for large-scale delete processes and subsequent vacuuming, thus optimising both performance and cost.

2.4 Designing Data Models and Schema Evolution

To gain business insights from your data, it must be in a usable form. The process of converting your data into a useful form involves data modelling and design. Data engineers must understand data modelling best practices and apply the appropriate level and type of modelling to the data source for different use cases.

1. Data Modeling in AWS

AWS provides services and tools that increase the flexibility of your data models while retaining logical separation, dimensions, attributes, and more.

- **Amazon Redshift**: Supports star schema, snowflake schema, and data modelling patterns like the Kimball method. It can also implement Inmon and Data Vault data modelling patterns with additional transformations and custom solutions.
- **Amazon Redshift Serverless**: Enables importing and querying data in a data warehouse. You can build schemas and tables, import data visually, and explore database objects using Query Editor version 2.

2. Data Processing Frameworks

Understanding data processing frameworks such as **Apache Spark** is essential. Spark can be combined with AWS services like **Amazon EMR**, **Amazon S3**, and **AWS Glue** to build data processing pipelines to handle structured and unstructured data at scale.

- **Amazon EMR**: Launch Spark clusters on demand, process data in parallel, and scale clusters as needed.
- **Amazon S3**: Read data directly into Spark DataFrames or Resilient Distributed Datasets (RDDs) using `spark.read`API for structured data or `spark.sparkContext.textFile` for raw text data.
- **AWS Glue**: Use Glue data crawlers to automatically infer the schema of structured data in Amazon S3 and create corresponding tables in the Data Catalog. This metadata can then be used by Spark for processing.

Other AWS services that complement Spark and handle various data types include:

- **Amazon Athena**: Used for interactive SQL queries on structured data stored in S3.
- **Amazon Comprehend**: Used for natural language processing tasks on unstructured text data.

3. Data Lineage for Data Governance

Data lineage helps track the origin, transformation, and flow of data throughout its lifecycle. This is crucial for error tracking, accountability, debugging, and ensuring data governance and compliance.

- **Amazon SageMaker ML Lineage Tracking**: Tracks machine learning-related data lineage by capturing metadata about training data, model artefacts, input data, output models, and transformations applied to training data. It can be integrated with AWS Glue and Amazon S3 to enrich data lineage information across data sources and processes.

4. Best Practices for Data Optimisation

Data optimisation techniques such as indexing, partitioning, and compression help improve data retrieval performance and storage efficiency:

- **Indexing Techniques**:
    - **Amazon RDS**: Identify columns frequently used in WHERE clauses or join operations and create indexes.
    - **DynamoDB**: Choose the right primary key and secondary indexes for efficient querying patterns.
- **Partitioning Strategies**:
    - **Amazon S3**: Store large datasets in logical folders based on attributes or categories to manage and query subsets of data using Athena or AWS Glue.
    - **Amazon Redshift**: Use distribution keys and sort keys to distribute data evenly across nodes and minimise data movement during queries.
- **Compression Techniques**:
    - Use supported compression formats such as gzip or bzip2 for S3 objects.
    - In Amazon Redshift, apply compression encoding to columns with repetitive or highly compressible data.
- **Data Serialisation Formats**:
    - Use formats like Parquet or ORC for data stored in Amazon S3 to optimise storage and query performance.
- **Data Caching**:
    - Use **ElastiCache** or **Amazon CloudFront** for cached and frequently accessed data to reduce backend load and speed up data access.
- **Data Archiving and Retention**:
    - Store historical or infrequently accessed data in cost-effective storage tiers like **Amazon S3 Glacier** or **S3 Glacier Deep Archive**.
- **Provisioned Throughput and Auto Scaling**:
    - Use DynamoDB to provision read and write throughput and leverage auto-scaling for automatic capacity adjustments.
- **Query Performance Optimisation**:
    - Use **Amazon RDS Performance Insights** or **Amazon Redshift** query monitoring to identify and address performance bottlenecks.

5. Schema Evolution Techniques

Schema evolution is the process of changing the schema or structure of data while preserving existing data and ensuring backward compatibility. This is particularly common with event data, where fields may be added or removed, or value types might change.

- **DynamoDB**: Automatically handles schema changes; new attributes can be added, and old data without the new attributes remains accessible.

- **Amazon RDS and Aurora**: Schema evolution can be more complex. Techniques include using AWS DMS for schema changes with minimal downtime or stored procedures to update schemas in a controlled manner.
- **Data Lakes (Amazon S3)**: Use a schema-on-read approach where the schema is applied at query time. Tools like **Athena** or **AWS Glue** enable schema evolution by adjusting data interpretation during analysis.
- **Schema Versioning and Registry**:
    - Add versioning information to data records or metadata to manage changes to the schema.
    - **AWS Glue** supports schema versioning to define different versions of data structures.
- **Columnar Data Formats**:
    - Use formats like Avro, Parquet, or ORC that support schema evolution by allowing fields to be added, removed, or modified without breaking existing data.
- **Automated ETL Jobs**:
    - Use **AWS Glue** for automated ETL jobs to transform data from one schema to another.
- **Techniques for Handling Schema Changes**:
    - Use a **dead letter queue**, schema registry, schema validation, schema versioning, backups before changes, and monitoring and alerting to proactively manage schema evolution issues.

6. Database Migration and Schema Conversion

When migrating databases from one engine to another or migrating to AWS, schema conversion is crucial.

- **AWS Schema Conversion Tool (SCT)**: Converts database schemas and code to be compatible with the target database engine for migrations to Amazon RDS or Aurora. It automates data type mapping, stored procedure conversions, and index conversions to streamline the migration process.
- **AWS Database Migration Service (DMS)**: Works with AWS SCT to convert source database schema to be compatible with the target database engine during migration.

7. Modelling Different Data Types in AWS

For the exam, understand how to model structured, semi-structured, and unstructured data in AWS to choose the appropriate storage and processing options:

- **Structured Data**: Use Amazon RDS, Aurora, and Redshift.
- **Semi-Structured Data**: Use DynamoDB, Amazon DocumentDB (MongoDB compatibility), Athena, and S3.
- **Unstructured Data**: Use Amazon S3, Amazon Rekognition, Amazon Transcribe, and Amazon Comprehend.

Domain 3: Data Operations and Support

3.1 Automating Data Processing with AWS Services

Introduction

Automating data processing in AWS involves setting up your infrastructure and data pipelines to run efficiently with minimal manual intervention. This section focuses on how to leverage AWS services like CloudFormation, Lambda, Step Functions, Amazon MWAA (Amazon Managed Workflows for Apache Airflow), and others to automate these processes.

Infrastructure Automation with CloudFormation

- **CloudFormation Templates**: Use CloudFormation to define and deploy your infrastructure. For example, you can create an S3 bucket, set up EventBridge rules to monitor the bucket, and trigger Lambda functions for automated processing.
- **Example Workflow**:
  1. **Create S3 Bucket**: Store your data files.
  2. **Monitor with EventBridge**: Trigger a Lambda function when a new file is added.
  3. **Lambda Functions**:
     - Validate the new file.
     - Launch a Step Function state machine.
     - Process the file with AWS Glue, update the data catalog, and notify via SNS.

Continuous Integration and Deployment

- **CodePipeline**: Automate the deployment of code changes. For example, when a Python file is committed to CodeCommit, use CodeBuild for building and integrating the code, run unit tests, and then deploy the changes automatically.
- **End-to-End Testing**: Automate tests that simulate the entire data pipeline, from file ingestion to final data validation.

Orchestrating Data Pipelines

- **Amazon MWAA (Managed Workflows for Apache Airflow)**:
  - **Apache Airflow**: Define Directed Acyclic Graphs (DAGs) to manage complex workflows.
  - **Tasks**: Include AWS Glue jobs, Spark jobs, or any custom scripts.
- **Step Functions**:
  - **State Machines**: Coordinate different AWS services using a visual workflow. It integrates with Lambda, Glue, EMR, and other services.
  - **Error Handling**: Built-in retry mechanisms and error handling for robust workflows.
- **Comparison**:
  - **MWAA**: Ideal for complex, code-driven workflows.
  - **Step Functions**: Best for serverless, event-driven workflows.

Troubleshooting Data Workflows

- **CloudWatch Logs**: Check for errors or exceptions in CloudWatch.

- **Apache Airflow Logs**: Review logs generated by Airflow tasks.
- **IAM Roles and Permissions**: Ensure correct permissions are configured for MWAA and related AWS services.
- **Dependency Management**: Verify that all required dependencies are installed and accessible.
- **Resource Monitoring**: Adjust MWAA resources (e.g., worker nodes) based on workflow demands.
- **Local Testing**: Test DAGs locally before deploying to MWAA.
- **AWS Support**: Utilise AWS Support for complex troubleshooting.

Consuming and Managing Data APIs

- **Lambda**: Create serverless functions that interact with APIs.
- **API Gateway**: Secure and manage API endpoints, which can trigger Lambda functions or interact with backend services.
- **AWS SDKs**: Use SDKs to interact with external APIs, handle authentication, request signing, and response parsing.

Best Practices:

- **Monitoring and Logging**: Use CloudWatch for monitoring API performance and logging errors.
- **Auto Scaling**: Automatically scale Lambda functions or backend resources to handle varying loads.
- **Caching**: Use API Gateway caching or ElastiCache to reduce backend load.
- **Throttling and Rate Limiting**: Prevent abuse with API Gateway's throttling features.
- **Error Handling**: Implement error handling in Lambda functions.
- **Security**: Use IAM and OAuth for API security.
- **Data Backup and Disaster Recovery**: Implement backup strategies for backend systems and Lambda function configurations.
- **API Versioning**: Manage changes with versioning to maintain backward compatibility.
- **Performance Optimization**: Continuously optimise API integrations using CloudFront, DynamoDB, and Lambda layers.

Processing Data with AWS Services

- **Key Services**:
  - **Amazon EMR**: For big data processing.
  - **Amazon Redshift**: For data warehousing.
  - **AWS Glue**: For ETL processes.
  - **Athena**: For querying data in S3 without infrastructure setup.

Use Case: Use Athena for quick data exploration and analysis. Write SQL queries in the Athena console, and results can be downloaded or saved to S3.

Managing Events and Schedulers

- **EventBridge**:
  - **Event-Driven Architectures**: Automate tasks using event rules and patterns.
  - **Integration with CloudWatch**: Monitor events and set up alarms.

    - **Schema Registry**: Manage event schemas for consistent event-driven architectures.

Example:

- **Data Pipeline Orchestration**: Use a Step Function that begins with a Lambda function checking file extensions, processes supported files, and sends failure notifications for unsupported files.
- **EventBridge Rule**: Create a rule to trigger the Step Function state machine when a new file is added to an S3 bucket.

3.2 Analysing Data Using AWS Services

Introduction

In this section, we focus on AWS services that help data consumers analyse and visualise data. We'll cover essential data manipulation operations, AWS services for data analysis, and practical examples to enhance your understanding.

Key Data Manipulation Operations

For the exam, be sure to understand the following data manipulation operations used in data analysis and reporting:

1. **Data Aggregation**:
    - **Definition**: Summarising or combining data from multiple records into a single value.
    - **AWS Services**:
        - **Amazon Redshift**: Use SQL queries with aggregate functions like `SUM`, `AVG`, `COUNT`.
        - **Athena**: Run SQL queries on data in S3 using `GROUP BY` and aggregate functions.
        - **Amazon OpenSearch Service**: Perform data aggregation with date histograms, term aggregations, and more.
2. **Rolling Average**:
    - **Definition**: Calculates the average of a set of values within a specified window or period to smooth out data fluctuations.
    - **AWS Services**:
        - **Amazon Redshift**: Use window functions to calculate rolling averages.
        - **QuickSight**: Built-in support for calculating rolling averages in visualisations.
3. **Grouping**:
    - **Definition**: Grouping data based on specific attributes or fields, often used with data aggregation.
    - **AWS Services**:
        - **Amazon Redshift**: Use the `GROUP BY` clause in SQL queries.
        - **Athena**: Supports `GROUP BY` for data grouping.
        - **QuickSight**: Group data in visualisations using dimensions and aggregations.

4. **Pivoting**:
    - **Definition**: Transforming data from rows into columns, useful for reshaping data for reports.
    - **AWS Services**:
        - **Amazon Redshift**: Use the `PIVOT` clause with aggregate functions in SQL queries.
        - **QuickSight**: Create pivot tables and visualisations.

AWS Services for Data Analysis and Visualisation

Here are key AWS services that support data analysis and visualisation:

1. **Amazon QuickSight**:
    - Create interactive dashboards and perform ad hoc data analysis.
2. **AWS Glue DataBrew**:
    - A visual data preparation tool for cleaning and normalising data without writing code.
3. **Athena**:
    - Run interactive SQL queries directly on data stored in Amazon S3.
4. **Amazon Redshift**:
    - Analyse large datasets using standard SQL queries.
5. **Amazon EMR**:
    - Process large amounts of data using frameworks like Spark and Hadoop.
6. **AWS Glue**:
    - Discover, catalog, and transform data for analysis; automatically generate ETL code.
7. **AWS Data Pipeline**:
    - Orchestrate and automate the movement and transformation of data between AWS services and on-premises sources.
8. **Lake Formation**:
    - Securely access and analyse data from multiple sources stored in a data lake.

Practical Examples

1. **Updating a Data Visualisation Workflow**:
    - **Scenario**: You currently run a script on an EC2 instance every 8 hours to extract, clean, and convert data into CSV files for spreadsheet analysis.
    - **Solution**: Move the script to a Lambda function, use EventBridge to trigger it every 8 hours, and send the processed data to an Amazon OpenSearch cluster. OpenSearch provides a dashboard and search features for real-time analytics.
2. **Cost-Effective Data Ingestion and Reporting**:
    - **Scenario**: Ingest sensor data from IoT devices and provide a daily visual report.

- **Solution**: Use a transient Amazon EMR cluster to perform data transformation tasks nightly, and terminate it after processing. Use QuickSight to generate the daily visual report.

Running SQL Queries on AWS Database Services

AWS supports SQL queries across various services:

1. **Amazon RDS and Aurora**:
    - Use SQL to manage the structure and data in relational databases.
2. **DynamoDB**:
    - Supports **PartiQL**, a SQL-compatible query language for querying NoSQL data.
3. **Amazon DocumentDB**:
    - A fully managed NoSQL database service compatible with MongoDB, using SQL-like queries.
4. **Athena**:
    - Use SQL to query data stored in Amazon S3, including data lakes in formats like CSV, Parquet, or JSON.
5. **AWS Glue**:
    - Use SQL-like transformations to prepare and clean data before loading it into target databases.

Data Analytics and Exploration in Athena Using Spark

- **Running Spark Applications**:
    - Add Spark code in Athena to process data and receive results.
    - Develop Spark applications using Python or Athena notebook APIs.
    - Set up an Amazon EMR cluster with Spark, load data from S3, and use Spark to explore and analyse data interactively.
    - Use Jupyter Notebooks in Amazon EMR to write and execute Spark code for data exploration tasks.

Verifying and Cleaning Data in AWS

AWS provides various services to verify and clean data as part of the data analysis process:

1. **Lambda**: Custom data validation and enrichment.
2. **Athena**: SQL-based data verification.
3. **QuickSight**: Data visualisation and exploration.
4. **Jupyter Notebooks**: Complex data cleaning tasks.
5. **SageMaker Data Wrangler**: Data preparation for machine learning models.

3.3 Maintaining and Monitoring Data Pipelines in AWS

Introduction

Maintaining and monitoring data pipelines is crucial for ensuring the reliability, availability, and performance of your data workflows. This section covers key aspects of monitoring, observability, logging, and troubleshooting in AWS data pipelines.

Key Questions for Data Pipeline Monitoring

- **Data Integrity**:
  - **Input and Output Validation**: How do you ensure that the data being processed and outputted is correct?
  - **Schema Validation**: How do you verify that the table schemas are correct before and after queries?
  - **Data Quality Tests**: Why should you run data quality tests on both input datasets and transformed datasets? Running these tests helps detect issues early, allowing you to flag errors, roll back changes, and investigate root causes.
- **Operations Monitoring**:
  - **System Performance**: How do you ensure that your systems are performing optimally?
  - **Metrics to Monitor**: Key metrics include query queue length, query concurrency, memory usage, storage utilisation, network latency, and disk I/O. Poorly performing queries might require refactoring or tuning, and you may need to optimise the database or scale up compute resources.

AWS Monitoring and Observability Tools

AWS provides a range of services to monitor and observe your data pipelines:

- **CloudWatch**:
  - **Metrics Collection**: Track and collect metrics from your AWS resources.
  - **Logs**: Monitor, store, and access log files. Set up alerts for specific thresholds.
  - **Events**: CloudWatch Events (now part of EventBridge) deliver a real-time stream of system events, enabling automated responses.
- **EventBridge**:
  - **Real-Time Data Streaming**: Connect applications using real-time data from various sources. Route data to targets like Lambda for real-time processing.
- **AWS X-Ray**:
  - **End-to-End Tracing**: Visualise and analyse interactions in large-scale, distributed applications with micro-services.
- **Lambda**:
  - **Invocation Logging**: Automatically logs each invocation. Custom log statements can be added for more detailed logging.

- **Application Load Balancer Logs**:
    - **Access Logs**: Capture detailed information about each request, including request time, client IP, and user agent.
- **Amazon RDS Logs**:
    - **Log Types**: Monitor database activities using error logs, slow query logs, and general logs.
- **CloudTrail**:
    - **API Call Logging**: Records all API calls, providing an audit trail. Logs can be sent to Amazon S3 or CloudWatch Logs for analysis and alerting.

Ensuring Observability

To make your application observable, it must be instrumented to share traces, metrics, and logs:

- **Traces**: Use AWS X-Ray to visualise application requests as they travel through your distributed system.
- **Metrics**: Collect metrics with CloudWatch to monitor performance.
- **Logs**: Centralise logging with CloudWatch Logs and set up automated responses with EventBridge.

Automation and Log Management

- **Log Groups and Streams**: CloudWatch Logs use log groups as containers for log streams, helping organise and control access to your logs.
- **Automation with CloudFormation**: Automate CloudWatch Logs configurations, log retention policies, and infrastructure management with CloudFormation templates.
- **Log Retention**: By default, CloudWatch Logs retains log data indefinitely. You can configure retention policies to manage costs and comply with data retention requirements.

Advanced Traceability with AWS Services

- **AWS Config**:
    - **Configuration Tracking**: Records configuration changes to AWS resources, useful for compliance, auditing, and change management.
    - **Managed Rules**: Predefined rules help evaluate whether AWS resources comply with best practices.
- **VPC Flow Logs**:
    - **Network Traffic Monitoring**: Capture information about IP traffic going to and from network interfaces in your VPC for security and compliance.
- **Security Hub**:
    - **Consolidated Security View**: Aggregates security alerts and compliance status across AWS services like GuardDuty, Inspector, and Config into a single dashboard.

Automated Remediation and Monitoring

- **Lambda for Automation**: Use Lambda functions to automate responses to specific events, such as performing custom log analysis or taking action based on log patterns.
- **EventBridge for Event-Driven Architecture**: Automatically trigger workflows based on specific events in your AWS environment.

Best Practices for Troubleshooting

- **Performance Monitoring**: Use CloudWatch or other monitoring tools to track key performance metrics, establish baselines, and identify anomalies.
- **AWS Health Dashboard**: Before troubleshooting your application, check the AWS Health dashboard for any ongoing service disruptions.
- **Log Analysis**: Analyse application logs for error messages or performance bottlenecks. Use CloudWatch Logs for centralised log management.
- **Service-Specific Logs**: For services like Lambda, RDS, or EC2, check service-specific logs for issues related to resource usage, errors, or timeouts.

Case Study: Troubleshooting Increased Latency

- **Scenario**: An application using Kinesis Data Streams logs data but experiences increased latency during peak hours.

- **Approach**:

    1. **Check CloudWatch Metrics**: Verify CPU utilisation of instances in the auto-scaling group.
    2. **Analyse Kinesis Logs**: Check for provisioned throughput exceeded errors. If none, the data stream and instances are handling the load.
    3. **DynamoDB Write Capacity**: Investigate and potentially increase the provisioned write capacity for the DynamoDB table.
- **Network Performance**: Monitor network traffic using VPC Flow Logs and Network Load Balancer logs to detect latency or packet loss issues.

Optimising Amazon Redshift Queries

- **Scenario**: Avoid latency for short-running queries in Amazon Redshift.
- **Solution**: Use parameter groups to configure workload management queues, ensuring that short-running queries are prioritised over long-running ones.

Managed Services and Error Handling

- **AWS Managed Services**: Services like Amazon EMR and Kinesis Data Streams require you to handle error conditions programmatically, while fully managed services may handle these differently.
- **Debugging Complex Ecosystems**: For services like Amazon EMR, issues can arise from the interplay of open-source software, custom code, and AWS services. Testing and debugging in smaller, controlled environments can help identify problems early.

Real-Time Anomaly Detection

- **Example**: Using Amazon Managed Service for Apache Flink to detect anomalies in a data stream and send alerts. For instance, monitor temperature levels in a data center and trigger a Lambda function to control the air conditioning if an anomaly is detected.

Kinesis Data Firehose Error Handling

- **Error Handling**: Kinesis Data Firehose retries data delivery until the retry duration expires. If it fails to deliver, it backs up the data to an S3 bucket. This ensures data is not lost even if delivery issues occur.

3.4 Ensuring Data Quality: A Critical Task in Data Engineering

Introduction: The Importance of Trust in Data

Ensuring data quality is fundamental in data engineering because trust in your data is crucial. If stakeholders and end-users lose confidence in the data, the overall goals of your projects are likely compromised. This is why it's essential to focus on understanding the use cases, the users, the data being produced, and how it will be served.

Starting a New Data Lifecycle: Key Questions

When embarking on a new data lifecycle, consider the following questions:

- **Who will use this data?**
- **How will they use this data?**
- **How do you collaborate to ensure that you serve the data that is needed?**

A common saying among data engineers is, "Publishing no data is better than publishing bad data." Therefore, automated checks throughout your pipelines are critical. These checks should ensure that if there's an anomaly or error, the pipeline stops and is remediated promptly.

Implementing Data Quality Checks

Data quality checks are essential in any data pipeline. These checks should be integrated before transformation, during transformation, and post-transformation. However, it's crucial to strike a balance—too many checks can lead to false positives, while too few can result in significant data quality issues.

Example Scenario: Handling Duplicate Values

Suppose you encounter a duplicate value in a column that should be your primary key during a data quality check. In this case, you can configure the system to send a notification. Depending on the issue, you might not need to stop the entire process.

Data Profiling: Understanding and Improving Data Quality

Data profiling is a critical part of data management, operations, and preparation. It involves systematically examining data to understand its structure, content, and quality.

Key Benefits of Data Profiling:

- **Identify Missing or Null Values:** Detect and manage missing or null values in datasets.
- **Assess Data Quality:** Identify anomalies, inconsistencies, and potential data quality issues.
- **Understand Data Distribution:** Analyse common values, frequency distributions, and correlations.

- **Summarise Data:** Generate statistical summaries like minimum, maximum, mean, median, and standard deviation.

AWS Services for Data Profiling

Several AWS services and tools are available to perform data profiling:

- **AWS Glue DataBrew:** Automates data profiling tasks, generating data statistics, frequency distributions, and quality insights.
- **Amazon Athena:** Provides basic data profiling through SQL queries that generate statistical summaries and analyse data patterns.
- **Amazon Redshift:** Offers SQL capabilities for data profiling, including calculating data distributions and identifying anomalies.
- **Amazon EMR with Spark:** Enables custom data profiling scripts for in-depth analysis.

Data Validation: Ensuring Quality and Reliability

Data validation is essential for ensuring data completeness, consistency, accuracy, and integrity throughout its lifecycle.

Key Validation Steps:

1. **Completeness:** Check for missing values, nulls, or empty fields.
2. **Consistency:** Validate data against predefined rules or constraints.
3. **Accuracy:** Ensure data accuracy through thorough checks.
4. **Integrity:** Maintain consistent data throughout its lifecycle.

Running Data Quality Checks During Processing

Data quality checks can be integrated into the data processing workflow to identify and address issues like empty fields or non-unique values.

AWS Tools for Data Quality Checks:

- **AWS Glue DataBrew Recipes:** Define rules and validation checks.
- **Amazon EMR with Spark:** Build custom data quality checks.
- **AWS Lambda and Step Functions:** Run quality checks and orchestrate tasks based on validation results.
- **SageMaker Data Wrangler:** Apply data quality checks using filters and transformations.

Handling Duplicate Records: Example Scenario

Suppose you notice duplicate records in an Amazon Redshift table after running AWS Glue jobs. To address this:

- **Create a Staging Table:** Replace existing rows in the Redshift table before persisting new data.
- **Perform Merge Operations:** Use an upset feature to ensure no duplicate records remain in the Redshift tables.

Data Sampling Techniques

Data sampling involves selecting a subset of data from a larger dataset to reduce computational resources and processing time while still gaining insights.

AWS Services for Data Sampling:

- **Amazon S3 Select:** Retrieve specific subsets of data using SQL-like queries.
- **Amazon Athena and Redshift:** Sample data with SQL queries using the LIMIT clause.
- **Amazon EMR with Spark:** Perform systematic, random, or stratified sampling.
- **SageMaker Data Wrangler and AWS Glue DataBrew:** Support data sampling for various use cases.

Addressing Data Skew

Data skew, where some partitions or keys have significantly more data than others, can lead to performance bottlenecks. Implementing mechanisms to detect and manage data skew is crucial.

Detection and Handling of Data Skew:

- **Monitoring and Alerting:** Use CloudWatch and custom metrics to monitor for data skew.
- **AWS Glue Job Bookmarks:** Track job progress and monitor for data skew issues.
- **Dynamic Repartitioning:** Implement techniques in Amazon EMR with Spark to balance the load.
- **Real-Time Detection:** Use Spark Streaming with Kinesis or Apache Kafka to detect and handle data skew as it occurs.

Ensuring Even Data Partitioning

To ensure data is evenly partitioned across nodes or tasks, use strategies like:

- **Partitioning Key and Hash Functions:** Distribute data evenly.
- **Load Balancing Mechanisms:** Monitor and redistribute workloads across nodes.
- **Shuffling Techniques:** Redistribute data during operations like MapReduce.

Data Partitioning and Bucketing

Partitioning and bucketing are techniques to optimise data storage and improve query performance:

- **Partitioning:** Organise data by columns (e.g., date columns) to reduce the amount of data scanned during queries.
- **Bucketing:** Distribute records into categories (buckets) to manage data distribution and improve performance during joins and aggregations.

Join Optimisation in AWS Services

AWS services offer built-in algorithms to handle data skew during join operations:

- **Skew Join Optimisation:** Features in Amazon EMR to automatically detect and handle data skew.
- **Adaptive Join Algorithms:** Amazon Redshift offers adaptive algorithms that switch between join strategies based on data skew.

Domain 4: Data Security and Data Governance

4.1 AWS Authentication and Authorisation Concepts

Fundamental concepts:

- **Authentication**: This is the process of identifying and verifying who you are. It's about ensuring that the person or system trying to access resources is who they claim to be.
- **Authorisation**: Once authenticated, authorisation determines what you, as an identified user or system, can access within the AWS environment.

Authentication in AWS

AWS provides several methods to authenticate users and resources before granting access to services and data. These methods include:

- **Password-Based Authentication**: The most common method where users log in with a username and password.
- **Certificate-Based Authentication**: This method uses SSL/TLS certificates to verify the identity of users and systems.
- **Role-Based Authentication**: Here, users or systems assume roles that grant them temporary credentials to access AWS resources.

How Do You Authenticate in AWS?

AWS Identity and Access Management (IAM) is the service that controls who can sign in (authenticate) and who can access resources (authorise). IAM supports different authentication methods:

- **Root User**: The most powerful user in an AWS account, which should be used sparingly and securely.
- **IAM User**: A specific identity created within your AWS account, typically for individual users or applications.
- **IAM Role**: Unlike users, roles are intended to be assumed temporarily and provide temporary credentials.
- **Federated Identity**: Users can authenticate via an external identity provider, like Google or Active Directory, using AWS IAM Identity Center for single sign-on (SSO).

Different AWS Identities

- **IAM User**: A unique identity within your AWS account with long-term credentials.
- **IAM Groups**: Collections of IAM users that share the same permissions.
- **IAM Roles**: Designed to be assumed by anyone who needs them, roles provide temporary credentials and are not tied to a specific user.

Example: You can use IAM roles to grant cross-account access, allowing trusted users from another AWS account to access resources in your account.

For the Exam: Key Authentication Concepts

Make sure you understand the following:

- **IAM Roles with Temporary Credentials**: Used for scenarios like federated user access or cross-service access.
- **Principal Permissions, Service Roles, and Service-Linked Roles**: Know how they differ and when to use each.

AWS Certificate-Based Authentication

AWS Certificate Manager (ACM) handles the creation, storage, and renewal of SSL/TLS certificates. You can use ACM to:

- Issue and manage public and private certificates for AWS services.
- Import third-party certificates into ACM.
- Export ACM certificates for use in your own public key infrastructure.

Managing Access with IAM Policies

IAM policies are documents that define what actions can be performed on AWS resources. There are several types of policies:

- **IAM Identity-Based Policies**: Attached to users, groups, or roles to define what actions they can perform.
- **Resource-Based Policies**: Directly attached to an AWS resource, like an S3 bucket, to control access.
- **Trust Policies**: Define who or what can assume an IAM role.

Important to Remember: In AWS, if a policy does not explicitly allow an action, the action is denied by default.

Using Managed and Inline Policies

- **AWS Managed Policies**: Pre-created policies managed by AWS, suitable for common permissions.
- **Customer Managed Policies**: Created by you and can be reused across multiple identities.
- **Inline Policies**: One-to-one relationships between a policy and an identity, deleted when the identity is deleted.

Applying IAM Policies

Let's start with a basic JSON IAM policy example that grants read-only access to objects in a specific S3 bucket. Here's what a simple policy might look like:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "AWS": "arn:aws:iam::123456789012:user/ExampleUser"
         },
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::example-bucket/*"
      }
   ]
}
```

Explanation of the JSON Policy Components

1. **Version**:
   - This indicates the version of the policy language. "2012-10-17" is the most recent and widely used version.
2. **Statement**:
   - This is the main section of the policy and can include multiple individual statements. Each statement defines a specific permission.
3. **Effect**:
   - The `Effect` element can either be "Allow" or "Deny." It determines whether the specified action is allowed or denied. In this case, "Allow" means the action is permitted.
4. **Principal**:
   - The `Principal` element specifies the identity (user, role, account, etc.) to which the policy is applied. In this example, the policy applies to the IAM user "ExampleUser" in account `123456789012`.
5. **Action**:
   - The `Action` element defines what actions are allowed or denied. In this example, the action "s3
     " is allowed, which means the user can read objects from the S3 bucket.
6. **Resource**:
   - The `Resource` element specifies the AWS resources to which the actions apply. Here, the policy applies to all objects (`/*`) within the S3 bucket named "example-bucket".

Example of Cross-Account Access

Now, let's consider a scenario where a Glue Crawler in Account A needs to access an S3 bucket in Account B. This requires setting up permissions in both accounts.

Step 1: Create an IAM Role in Account B

In Account B, create an IAM role that grants the necessary permissions to access the S3 bucket. Here's an example of the trust policy for this role:

```
{

  "Version": "2012-10-17",

  "Statement": [

    {
        "Effect": "Allow",

        "Principal": {

          "AWS": "arn:aws:iam::111111111111:role/GlueCrawlerRole"

        },

        "Action": "sts:AssumeRole"

      }

  ]

}
```

- **Principal**: The `Principal` here is the IAM role in Account A (with account ID `111111111111`), which will assume this role in Account B.
- **Action**: The `Action` "sts
  " allows the role from Account A to assume the role in Account B.

Step 2: Attach a Policy to the Role in Account B

Next, attach a policy to the role in Account B that allows access to the S3 bucket:

S3 Access Policy in Account B:

```
{

  "Version": "2012-10-17",

  "Statement": [

    {

        "Effect": "Allow",

        "Action": "s3:GetObject",

        "Resource": "arn:aws:s3:::target-bucket/*"

    }

  ]

}
```

- **Action**: This allows the `s3:GetObject` action, which is needed to read objects in the bucket.
- **Resource**: This policy applies to all objects in the "target-bucket" in Account B.

Step 3: Assume the Role in Account A

In Account A, configure the Glue Crawler to assume the role in Account B. The Glue service in Account A needs a role with a policy that allows it to assume the role in Account B.

Assume Role Policy in Account A:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::222222222222:role/AccessS3InAccountB"
    }
  ]
}
```

- **Action**: The `sts:AssumeRole` action allows the role in Account A to assume the cross-account role in Account B.
- **Resource**: The ARN of the role in Account B that grants access to the S3 bucket.

Final Setup

1. **Account A**:
   - The Glue Crawler role assumes the role in Account B with the necessary permissions to access the S3 bucket.
2. **Account B**:
   - The S3 bucket policy (if needed) should also allow the role in Account B to access the bucket. For more granular control, you can specify the exact actions and resources within the bucket.

IAM with AWS Services

IAM acts as the Identity Provider (IDP) for your AWS account, managing authentication and authorisation:

- **API Gateway**: Requires specific IAM roles and policies for both API creation and execution.
- **CloudFormation**: You can control what users can do with CloudFormation and manage access through VPC endpoints.

Secrets Management in AWS

AWS Secrets Manager is a service designed to protect sensitive information like database credentials and API keys. It supports:

- **Integration with IAM**: Attach access control policies to secrets.
- **Encryption with KMS**: Secrets are encrypted at rest using AWS managed or customer-managed keys.
- **Automatic Rotation**: Schedule secure rotations for secrets, with built-in support for services like Amazon RDS and Amazon Redshift.

Example Problem: If you're using Athena and QuickSight for data analysis, and permissions issues arise after updating a Data Catalog, remember that cross-service access must be configured correctly. This means ensuring QuickSight has the appropriate permissions to access new S3 buckets used by Athena.

4.2 Understanding Authorisation Mechanisms in AWS

As a data engineer, you're responsible for the security of the systems you build and maintain. This involves understanding two key concepts: **authentication** and **authorisation**. While authentication verifies who you are, authorisation determines what you can access within a system once authenticated.

In AWS, authorisation is about ensuring that authenticated users or systems have the least privilege necessary to access resources. This is done using security policies, which can be managed programmatically as code. Just like with application code, these policies can be tested and automated in a pipeline to verify they work as expected.

For example, **IAM Access Analyser** helps you continuously evaluate permissions in policies, identifying resources that might be accessible from outside your AWS account. Additionally, AWS Organisations allows you to enforce policies across multiple accounts.

Authorisation Models in AWS: RBAC and ABAC

In AWS, you control access to resources using policy-based mechanisms, primarily through two models: **Role-Based Access Control (RBAC)** and **Attribute-Based Access Control (ABAC)**. Let's break them down:

Role-Based Access Control (RBAC)

- **RBAC** grants access based on roles. For instance, a "developer" role might allow a user to perform tasks within a data pipeline or application.
- This model is easier to manage when roles align clearly with job functions. However, it can become complex if users need permissions across multiple roles or when roles are hard to define due to complex business logic.

Attribute-Based Access Control (ABAC)

- **ABAC** uses attributes to define permissions. These attributes, known as tags in AWS, can be attached to IAM users, roles, and resources. For example, tags could represent environments like "development" or "production."

- ABAC is more dynamic and allows for contextual, granular access control. However, it can be challenging to implement, as it requires careful definition of attributes and policies.

Example: You could create three roles, each tagged with different values (e.g., "heart," "star," "bolt"). A single policy could then grant access only when both the role and the resource have matching tag values.

Combining RBAC and ABAC

- You can also combine RBAC and ABAC for more granular access control. This hybrid approach uses both roles and attributes to make authorisation decisions.

Managing Database Access in Amazon Redshift

For database users in **Amazon Redshift**, you can manage access and permissions through SQL commands like `CREATE USER` or `ALTER USER`. Here are key points:

- **Superusers** can create or drop users and have ownership permissions for all databases.
- **Role-Based Access Control (RBAC)** in Redshift allows you to secure sensitive data by controlling what users can do at a broad or detailed level.
- **Principle of Least Privilege**: Users should have only the permissions they need to perform their roles. Permissions are managed at the role level, so you don't have to update individual database objects.

Managing Data Lake Permissions with Lake Formation

Lake Formation provides centralised management of access control for data in your data lake:

- You can define security policies at various levels, from databases to individual cells in a table.
- **Tag-Based Access Control**: Lake Formation allows you to manage permissions using custom labels (LF tags), making it easier to govern large datasets across services like Amazon Redshift, Athena, and AWS Glue.
- **Integration with Redshift**: Lake Formation can also be integrated with Amazon Redshift data sharing to centrally manage access at various levels, ensuring secure access without migrating data.

Secrets Management in AWS

In a previous lesson, we discussed using **Secrets Manager**. However, you can also use **Parameter Store** from AWS Systems Manager for storing configuration data and secrets:

- **Parameter Store**: Used for storing data like passwords and database strings. However, it doesn't provide automatic rotation for secrets.
- **Secrets Manager Integration**: You can reference Secrets Manager secrets within Parameter Store to leverage its rotation capabilities.

AWS provides robust tools to secure your data at rest and in transit, ensuring that your data is protected by design and by default. Let’s explore how AWS enables data encryption, masking, and key management to help you secure your applications and comply with regulatory requirements.

4.3 Key Concepts in Data Encryption

1. **Encryption at Rest and In Transit**
   - All AWS services offer encryption for data at rest and in transit. Encryption at rest protects your data from unauthorised access when it is stored, while encryption in transit safeguards data as it moves across networks.
   - **AWS Key Management Service (KMS)**: A managed service that lets you create and control cryptographic keys used to encrypt your data. AWS KMS integrates with most AWS services, allowing seamless encryption and key management across your infrastructure.
2. **AWS KMS Integration**
   - AWS KMS integrates with **AWS CloudTrail** to log the usage of your KMS keys for auditing, compliance, and regulatory requirements.
   - Through the **AWS KMS API**, you can manage keys, including creating custom key stores and performing cryptographic operations.
3. **AWS CloudHSM**
   - If you need direct control over the hardware that generates and stores encryption keys, **AWS CloudHSM** provides a managed hardware security module (HSM).
   - AWS handles hardware provisioning, software patching, and creating encrypted backups, while you manage scaling, crypto accounts, and credentials within the HSM.
4. **TLS for Secure Connections**
   - **Transport Layer Security (TLS)** is used at the application layer to secure communications. AWS services like Network Load Balancer, Application Load Balancer, CloudFront, and API Gateway allow you to upload and manage digital certificates using **AWS Certificate Manager (ACM)**.

Encryption Best Practices for AWS Services

- Understand how to implement data at rest encryption across AWS services such as **Amazon EBS**, **Amazon S3**, **Amazon RDS**, **Amazon Redshift**, **ElastiCache**, **Lambda**, **Amazon EMR**, **AWS Glue**, and **SageMaker**.
- **Amazon SQS** also supports **Server-Side Encryption (SSE)** for secure message transmission.

Example Question: If you create an Amazon EMR cluster with encrypted EBS volumes, is the local disk also encrypted?

- No, the local disk is not automatically encrypted, but you can enable local disk encryption in the security configuration.

4.4 Data Anonymization, Masking, and Key Salting

1. **Data Anonymization**
   - This process removes sensitive information from datasets, ensuring that personal or classified data is no longer identifiable.
2. **Data Masking**
   - Data masking obscures confidential data by replacing it with altered values. This allows you to use realistic-looking data without exposing real data.
3. **Key Salting**
   - Salting involves adding random data to a password before hashing it. This makes it more difficult for attackers to crack hashed passwords.

Tokenisation in AWS

1. **Understanding Tokenisation**
   - **Tokenisation** replaces sensitive data with a random string (token) that has no direct value. Unlike encryption, tokenisation does not use a mathematical algorithm to transform data. Instead, a token vault stores the relationship between the token and the original data, which is usually encrypted for security.
2. **Implementing Tokenisation**
   - **Token vaults** can be constructed in Amazon VPCs to store sensitive information securely while sharing tokens with approved services.

Example: In a secure design, a serverless application could use **API Gateway**, **Lambda**, **Amazon Cognito**, **DynamoDB**, and **AWS KMS** to tokenise sensitive customer data such as social security numbers and credit card details. The tokenised data is stored securely, reducing the complexity and cost of managing access while maintaining data protection.

Advanced Security Features

1. **Dynamic Data Masking (DDM) in Amazon Redshift**
   - **Dynamic Data Masking** allows you to control how sensitive data is displayed to users at query time without modifying the underlying data.
   - You can apply masking expressions to table columns and customise policies to target specific users or roles, enhancing data security without altering the data itself.
2. **Enhanced Security with VPC Endpoints**
   - **VPC endpoints** can be used to ensure that AWS services like DynamoDB and AWS KMS only communicate through private networks, avoiding the public internet.
   - Endpoint policies can enforce specific permissions, while AWS KMS can restrict key management activities to authorised roles.

When managing data in AWS, it's crucial to have a robust logging and audit trail to meet regulatory requirements and ensure data is available when needed. This involves configuring logs, storing them securely, and ensuring that they can be retrieved for auditing and compliance purposes.

Logging and Data Retention

Certain regulations, such as HIPAA (Health Insurance Portability and Accountability Act) and PCI (Payment Card Industry Data Security Standard), require data to be retained for a specified period. AWS provides tools to log data effectively, monitor it, and ensure it's available upon request while also allowing for compliant deletion when no longer needed.

- **CloudWatch Logs**: When you create a log group, it acts as a container for log streams, helping you organise and control access to your logs. By default, logs in CloudWatch Logs are retained indefinitely, but you can configure retention policies to specify how long to retain your log data.
- **CloudTrail**: AWS CloudTrail captures API calls and logs them for auditing and compliance. These logs can be sent to Amazon S3 or CloudWatch Logs, where you can set up alerts, analyse them, and retain them as needed. CloudTrail helps track all API activity across AWS accounts and Regions.

Centralised Logging with CloudTrail Lake

- **CloudTrail Lake**: This service provides centralised logging for CloudTrail events across multiple AWS accounts and Regions. You can create an event data store to log specific event categories, such as CloudTrail events, AWS Config configuration items, or external user activity data.
- **Running SQL Queries**: You can run SQL queries across multiple event data stores in CloudTrail Lake using supported SQL JOIN keywords. This allows for more flexible and detailed analysis of your logs.
- **Monitoring with CloudWatch Metrics**: CloudTrail Lake supports CloudWatch metrics, enabling you to monitor data ingestion into your event data store, helping you understand the volume of data being logged.

Logging Considerations for Amazon EMR

When designing a cluster in Amazon EMR, consider the following:

- **Debugging and Log Storage**: By default, Amazon EMR writes log files to the primary node in the `/mnt/var/log/`directory. You can also archive log files to Amazon S3 for long-term storage.
- **Integration with CloudTrail**: CloudTrail captures all API calls for Amazon EMR, including those made through the console and API. If you configure a CloudTrail trail, you can deliver these logs continuously to an S3 bucket for auditing.

Analysing Logs in AWS

Choosing the right AWS services to analyse logs is crucial for operational efficiency:

- **CloudWatch Logs Insights**: Use this service to search and analyze log data in CloudWatch Logs. It automatically discovers fields in logs from AWS services like Route 53, Lambda, CloudTrail, and VPC. You can perform queries to diagnose issues and validate fixes.

- **Athena CloudWatch Connector**: This connector allows you to query your log data in CloudWatch Logs using SQL. It maps your log groups as schemas and each log stream as a table, making it easier to query all logs within a log group.

- **Streaming Logs to OpenSearch Service**: Configure a CloudWatch Logs log group to stream data in near real-time to an OpenSearch Service cluster using a CloudWatch Log subscription. This setup consolidates system, application, and AWS service logs into a single, highly scalable service.

Developing and Scheduling with CloudWatch

- **Debug Logs**: Data engineers can use CloudWatch to discover logs for the services they run and keep a debug log while developing their applications.

- **Scheduling Services**: Use CloudWatch Events to schedule the services you want to launch at specific times, helping automate operations and reduce manual intervention.

4.5 Understanding Data Privacy and Governance in AWS

Data privacy and governance are critical aspects of managing sensitive information in AWS. This lesson delves into how to protect personally identifiable information (PII) and ensure data sovereignty throughout the data lifecycle.

Protecting Sensitive Data Throughout Its Lifecycle

AWS provides various solutions designed to protect sensitive data during its entire lifecycle, from data ingestion to storage and processing:

- **Data in Transit**: Protect data as it travels over networks using **Transport Layer Security (TLS)**. TLS ensures secure communication channels.
- **Data at Rest**: Use **volume encryption**, **object encryption**, or **database table encryption** to protect stored data.

For sensitive workloads, additional protection techniques like **field-level encryption** can help secure individual data fields within larger application payloads, preserving the overall structure while protecting sensitive information.

Example: AWS CloudFront, combined with Lambda@Edge, can protect sensitive data at the edge of the AWS network, reducing data exposure as it enters the system and minimising the compliance footprint for auditing.

Creating a Secure Data Lake

A secure data lake involves masking, encrypting data, and enabling fine-grained access controls:

- **Data Ingestion and Processing**: Sensitive data can be identified, masked, and encrypted during ingestion before being securely stored in Amazon S3.
- **Fine-Grained Access with Lake Formation**: Use **Lake Formation** to define and enforce fine-grained access permissions, ensuring only authorised users can access specific data fields.

Use Case: For diagnostic devices sending MQTT messages via AWS IoT Core, you could use Kinesis Data Firehose to pre-process data, store it in S3, and use AWS Glue for ETL processing. Amazon Comprehend can identify sensitive data, and Lake Formation can manage access for analysts using Athena.

Data Privacy in ETL Pipelines

When designing ETL pipelines, consider the following:

- **Sensitive Data Scanning**: Implement a scanning process to identify sensitive data before further processing. This can be done on a scheduled basis, adding time to the pipeline but ensuring compliance with data privacy requirements.
- **IAM Policies and Permissions**: Use IAM managed policies to control permissions for Lambda functions that access AWS resources within the pipeline.

Example: In a typical pipeline, raw data is scanned for sensitive information before being processed. If sensitive data is found, an administrator is notified to take action, such as adjusting the data cleanup process.

Data Sovereignty and Backup Strategies

Data sovereignty is crucial for ensuring that data remains within allowed Regions, and it ties into your backup and disaster recovery strategies:

- **Backup Plans**: AWS Backup allows you to define backup plans that specify the frequency and retention of backups across multiple AWS accounts using AWS Organisations.
- **Data Classification**: Use S3 bucket tags to classify data according to its protection level. For example, critical data may require different backup strategies compared to non-sensitive data.

Use Case: In a data lake environment, backing up raw data might be essential, but for PII data, it may be preferable to back up de-identified data instead.

Data Sharing and Permissions in AWS

When sharing data across AWS services, such as Amazon Redshift, it's important to manage permissions carefully:

- **Data Share Lifecycle**: Amazon Redshift applies checks to ensure that users who own or have privileges on data share objects cannot be dropped until permissions are properly managed.
- **Column-Level Access Control**: Use column-level grants and revokes in Amazon Redshift to control who can access specific columns, ensuring that confidential data is protected.

Example: If a new table contains confidential data, you can restrict access to certain columns based on user permissions, enhancing security and compliance.

Managing Configuration Changes with AWS Config

AWS Config tracks and manages configuration changes across your AWS account:

- **Configuration Items**: AWS Config discovers resources, generates configuration items, and tracks changes over time.

- **Config Rules and Lambda**: Each AWS Config rule is associated with a Lambda function that evaluates the rule’s logic. AWS Config can stream changes and notifications to an SNS topic for further action.

Example: Removing an egress rule from a VPC security group triggers AWS Config to update the configuration items for that group and all related resources, ensuring comprehensive change management.