307401 Big Data

Introduction to AWS Data Engineering

2024-2

What is Data Engineering?

- Data engineering is the process of designing, building, and maintaining the infrastructure and systems that enable the collection, storage, processing, and analysis of large amounts of data.
- It involves tasks such as data integration, data transformation, data storage and retrieval, data quality assurance, and data pipeline development.
- Data engineering is a crucial component of data-driven organizations and enables them to effectively leverage the power of data to gain insights, make informed decisions, and drive business growth

End-to-End Data Engineering Architecture

- Data engineering architecture refers to the design and structure of the systems and processes used to manage and process data in an organization.
- A well-designed data engineering architecture enables efficient, reliable, and scalable data processing, storage, and retrieval.

A typical data engineering architecture may involve several layers, including:

- **Data Sources:** This layer includes all the systems and applications that generate data, such as databases, sensors, and web services.
- **Data Ingestion:** This layer is responsible for collecting and aggregating data from various sources and bringing it into the data processing pipeline. Tools such as Apache Kafka, Apache Flume, and AWS Kinesis are often used for this purpose.
- **Data Storage:** This layer involves storing data in a structured, semi-structured, or unstructured format, depending on the nature of the data. Common data storage systems include relational databases, NoSQL databases, and data lakes.
- **Data Processing:** This layer is responsible for transforming, cleaning, and enriching the data to make it useful for downstream analytics and machine learning. Tools such as Apache Spark, Apache Beam, and AWS Glue are commonly used for this purpose.
- **Data Analytics:** This layer involves using various tools and techniques to analyze the data and gain insights. This may involve data visualization, machine learning, and other data analysis techniques.
- **Data Delivery:** This layer is responsible for delivering the data and insights to end-users or downstream applications, such as dashboards, reports, and APIs.
- Overall, a good data engineering architecture should be scalable, reliable, secure, and flexible enough to accommodate changing data requirements and business needs.

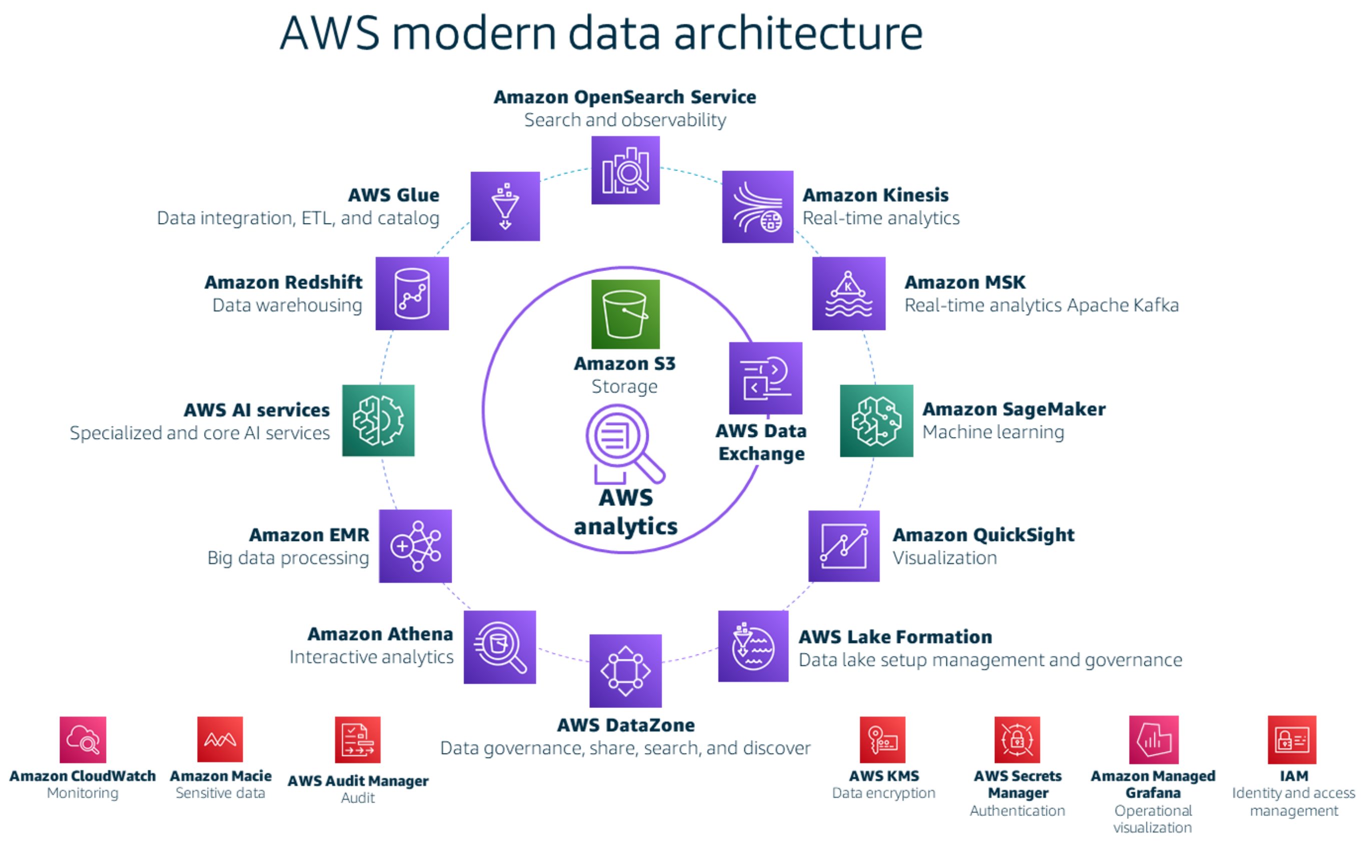
The Day-to-Day Responsibilities of a Data Engineer

The day-to-day responsibilities of a data engineer can vary depending on the organization, but here are some common tasks that a data engineer may perform:

- **Data Ingestion:** Collecting data from various sources and systems and bringing it into the data processing pipeline. This may involve setting up ETL (Extract, Transform, Load) pipelines or working with tools such as ADF, AZCopy, FTP Tools, API Calls, Apache Kafka, Apache Flume, or AWS Kinesis.
- **Data Storage:** Designing and implementing data storage solutions that can accommodate large volumes of structured, semi-structured, or unstructured data in data lake like Azure Data Lake Gen2, AWS S3, Google Cloud Storage, HDFS. This may involve working with Cloud Data Warehouses like Azure SQL DWH, AWS RedShift, Google Big Query and snowflake databases, NoSQL databases.
- **Data Transformation:** Transforming and cleaning the data to make it useful for downstream analytics and machine learning. This may involve using tools such as Apache Spark,Databricks Spark SQL, Synapse Analytics with SQL/Spark, Apache Beam, or AWS Glue.
- **Data Quality:** Ensuring the quality and consistency of the data by implementing data validation, verification, and monitoring processes using pyspark Transformations in Databricks, synapse analytics or any other cloud Data Warehouses using SQL.
- **Performance Optimization:** Improving the performance of data processing pipelines by optimizing queries, improving data partitioning, or using caching strategies.
- **Data Security:** Ensuring the security of the data by implementing encryption, access controls, and other security measures.
- **Documentation:** Creating and maintaining documentation for the data processing pipelines and data storage systems to help ensure that they can be easily understood and maintained by other members of the team. Collaboration: Working closely with data analysts, data scientists, and other stakeholders to understand their requirements and to ensure that the data engineering systems meet their needs. Overall, the day-to-day responsibilities of a data engineer involve designing, building, and maintaining the infrastructure and systems that enable an organization to effectively manage and process large volumes of data.

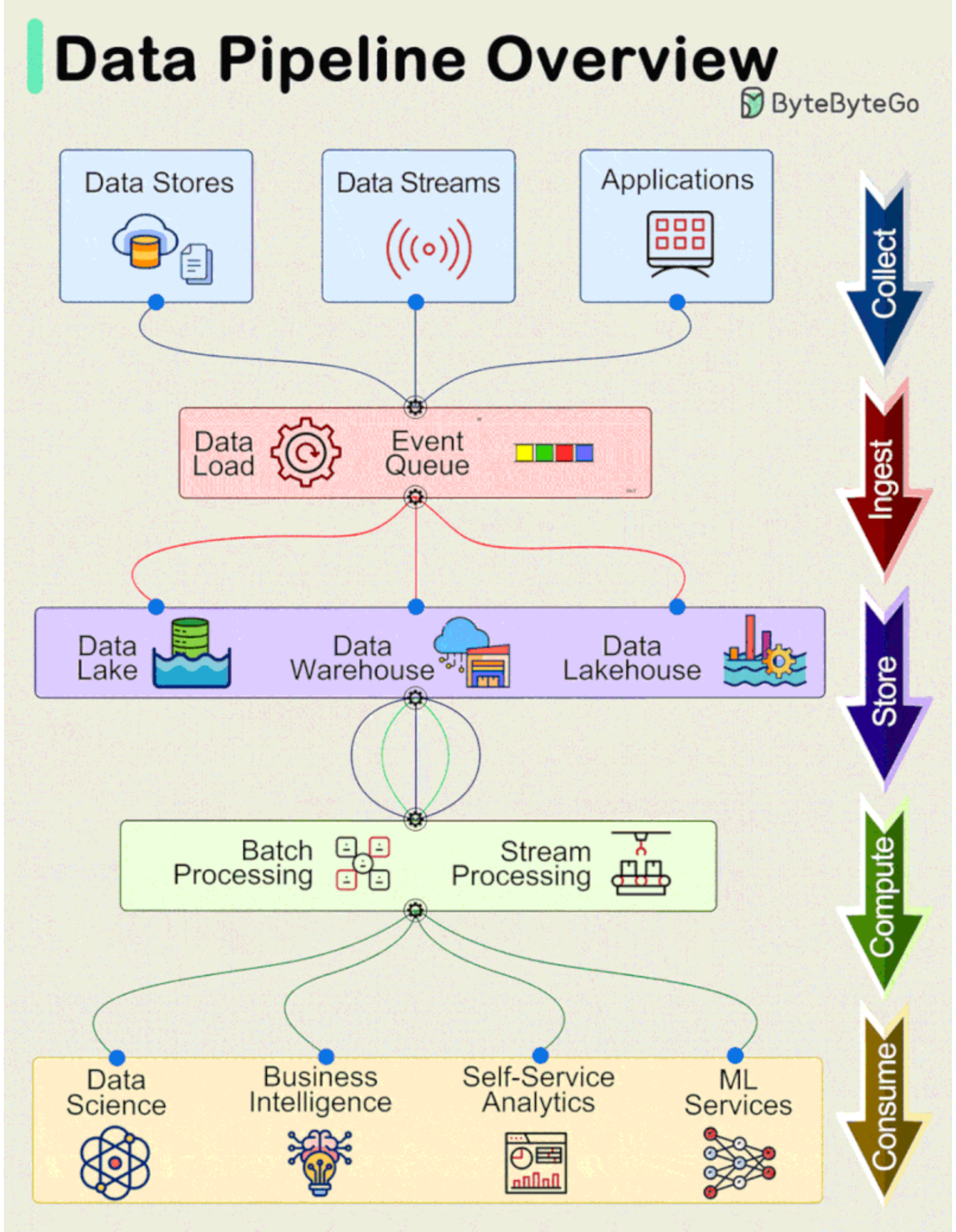
Overview of AWS Data Engineering

- **Core AWS Services**:
  - **S3**: Scalable object storage for raw/staged/curated data.
  - **Glue**: Serverless data integration (ETL), data cataloging.
  - **Athena**: Serverless querying on S3 using SQL.
  - **Redshift**: Data warehousing and analytics.
  - **Kinesis**: Real-time data streaming.
  - **Lambda**: Serverless compute for lightweight ETL.
  - **EMR**: Managed Hadoop/Spark clusters.
  - **Data Pipeline / Step Functions**: Workflow orchestration.



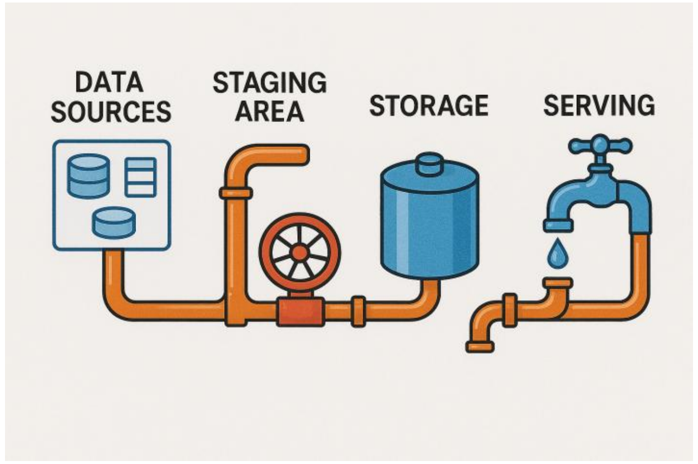
Institute of Analytics
Recognized Programme

University of Petra



Data Pipeline

- Data pipeline is the infrastructure that you build to support data-driven decision-making.
- At the most basic level, any data pipeline infrastructure must be able to:
  1. Bring data in
  2. Store it
  3. Provide the means to work with the data to derive insights.
- As the data engineer or data scientist, it's your job to build the pipeline to figure out what's appropriate and how you're going to do it.
- Often this will include **exploring and experimenting** to get to know the data and the best way to derive value from it.

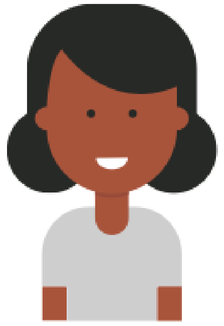


Collect data

Store and process data

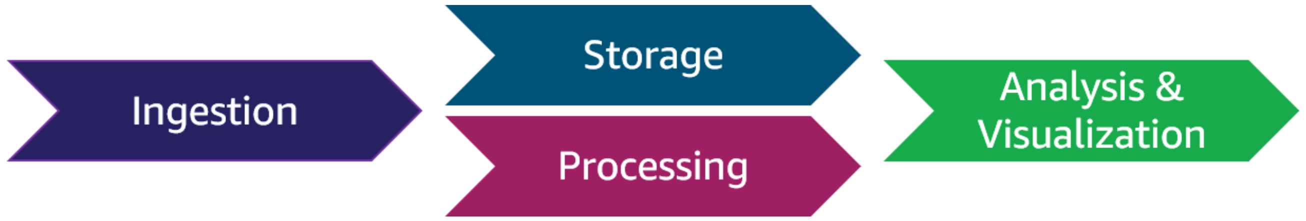
Build something useful with data

Common data pipeline questions

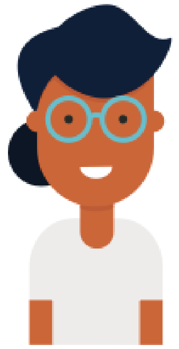


Data engineer

- Does the organization have the data that addresses the need? Where is the data stored and in what format?
- Will I need to combine data from multiple sources?
- What is the type and quality of data? What will be the source of truth?
- What are the security requirements for this data? Who needs access to it and in what state?
- What types of mechanisms are needed to transfer the data from its locations into the pipeline?
- How much data is there, and how frequently is it updated or processed?
- How important is speed when data is requested?



- What can the data tell me?
- How will I evaluate the results?
- What kind of visualization do I need?
- Which formats and tools are the analysts familiar with?
- Do I need a big data framework?
- Which type of AI/ML models fit?
- What is the simplest way to implement AI/ML?



Data scientist

Work Backwards when designing your infrastructure

- The number of variations and options to create a data pipeline are broad.
- The key to designing an effective decision-making infrastructure is to Start With The Business Problem to be solved or decision to be made, then build the pipeline that best suits that use case.

2. What data do you need to support this?

1. What decision are you trying to make?

Weigh the trade-offs of cost, speed, and accuracy.

The Benefits Realized

Traditional Approach (Technology First)

A retail chain might start by building a general-purpose data warehouse that collects all store data. They spend months designing complex ETL processes to load all available data from point-of-sale systems, inventory management, customer loyalty programs, and financial systems into a central repository. After significant investment, they have a comprehensive data warehouse but struggle to derive specific business value from it.

Working Backward Approach (Business Problem First)

Instead, let's see how working backward would look for the same retailer:

1. **Define the business problem**: Store managers need to reduce stockouts of high-demand items while minimizing excess inventory of slow-moving products.
2. **Identify the decision to be made**: Determine optimal reorder points and quantities for each product at each store location.
3. **Define the data needs**:
   - Historical sales data by product/store (hourly granularity)
   - Current inventory levels
   - Supplier lead times
   - Seasonal trends
   - Local event data (that might impact demand)
   - Weather forecast data
4. **Design the pipeline**:
   - Streaming integration with POS systems for near real-time sales data
   - API connections to inventory management systems
   - Integration with external weather and local event APIs
   - A specialized analytics database optimized for time-series forecasting
   - ML models specifically designed for demand prediction
   - Mobile dashboards for store managers with actionable reorder recommendations

The Expected Benefits by Starting Backwards

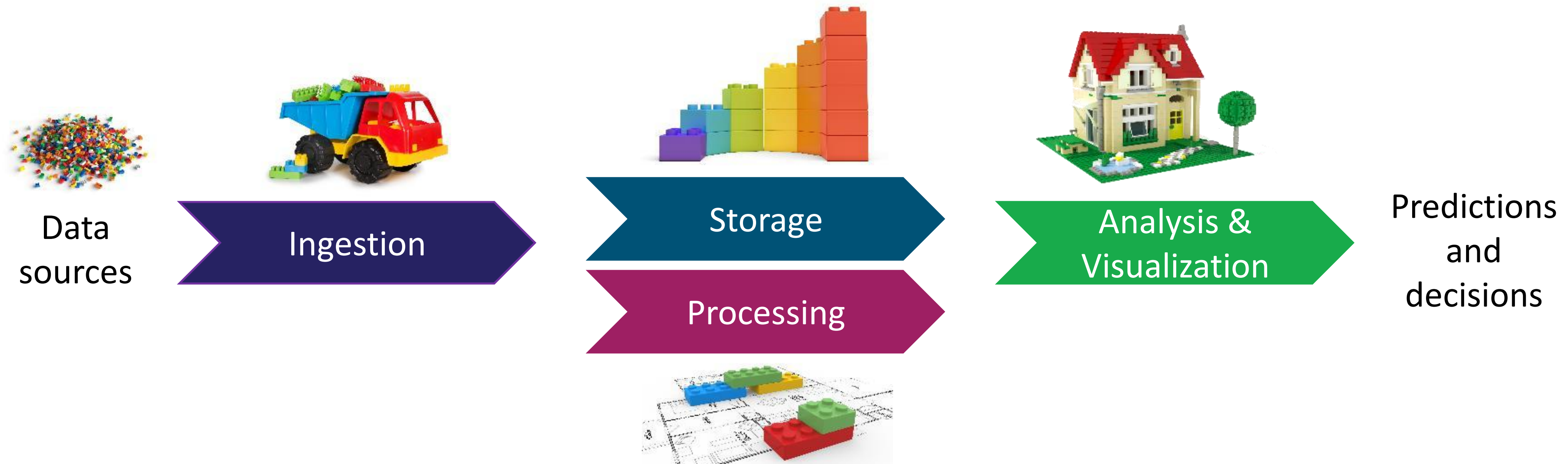
By working backward from the specific inventory optimization problem, the following can be acheived:

1. Faster time to value
2. Right-sized infrastructure
3. Measurable ROI
4. Iterative improvement
5. Better technology decisions

The cloud makes it easier to start up different infrastructure to serve different use cases. An organization doesn't need to invest in an infrastructure that tries to serve all business needs with a single database, server, or technology stack. For each use case, start with the end in mind, and build the data pipeline that supports it.

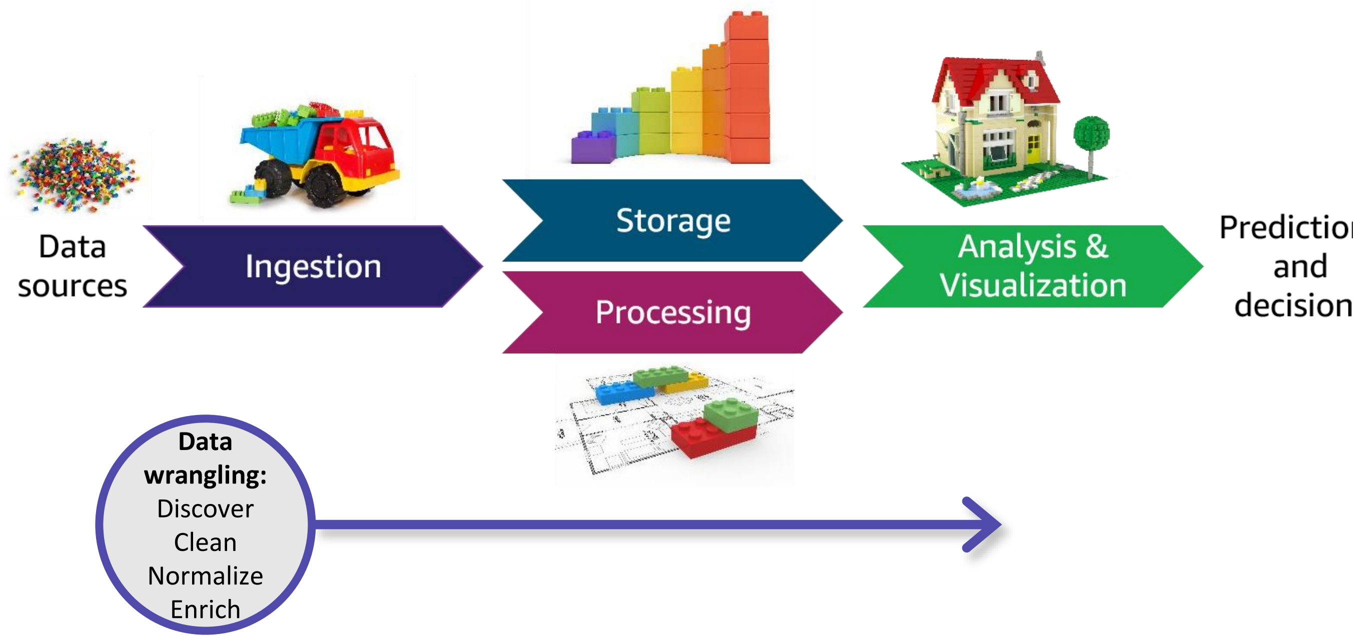
Layers of the pipeline infrastructure

- The infrastructure layers that you need to build include methods to ingest and store data from the data sources that you have identified.
- You need to make the stored data accessible for decision-making processes.
- You must also create the infrastructure to process, analyze, and visualize data by using tools that are appropriate to the use case.
- To build an appropriate infrastructure, you will need to understand the nature of the data and the intended type of processing and analysis to be performed.



Actions taken with data in the pipeline

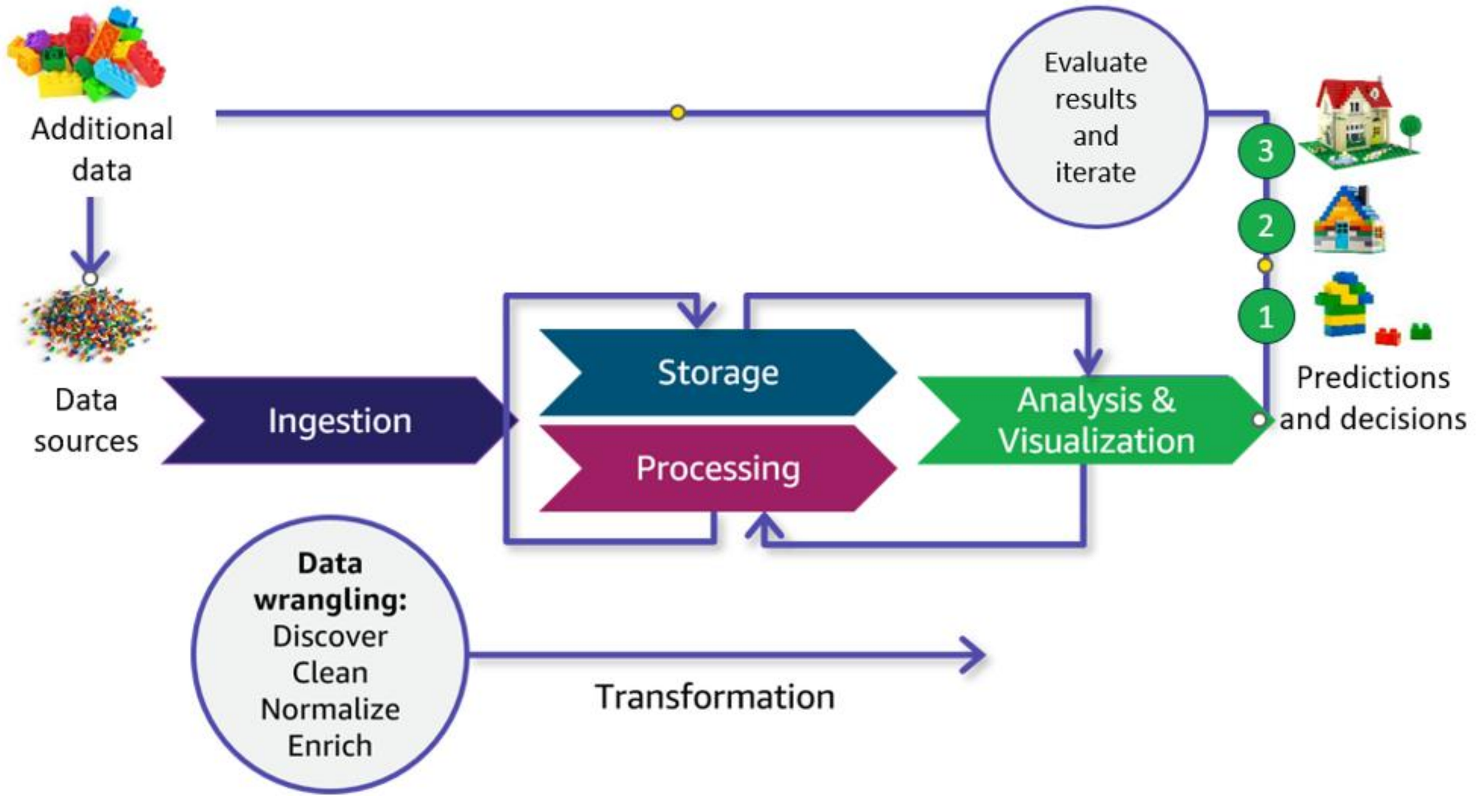
- The pipeline must support the ability to discover details about the data and how it fits (or doesn't fit) your intended outcomes.
- You might start with a hypothesis and use BI tools or other mechanisms to discover more information about the data and then experiment and see where it takes you.
- Data will almost always be transformed as it moves through the pipeline. This might include modifying the format to support a specific analysis tool or replacing values (for example, zeroes in place of nulls).
- *Data wrangling* is a term that is used to describe the ways in which data is manipulated and transformed from its raw state into more meaningful states that downstream processes or users can use.



Transformation

Iterative processing through the pipeline

- Another key characteristic of deriving insights by using your data pipeline is that the process will almost always be iterative.
- For example, in this illustration, the initial iteration (number 1) yielded a result that wasn't as defined as was desired. Therefore, the data scientist refined the model and reprocessed the data to get a better result (number 2).
- After reviewing those results, they determined that additional data could improve the detail available in their result, so an additional data source was tapped and ingested through the pipeline to produce the desired result (number 3).



Key takeaways: The data pipeline – infrastructure for data-driven decisions

- A data pipeline provides the infrastructure for data-driven decision-making.
- When designing a pipeline, start with the business problem to be solved and work backwards to the data.
- The pipeline includes layers to ingest, store, process, and analyze and visualize data passing through it.
- Data wrangling refers to how data is acted upon as it passes through the pipeline. Tasks include discovery, cleaning, normalization, transformation, and augmentation.
- Data is processed iteratively to evaluate and improve upon results.

amazon
ATHENA

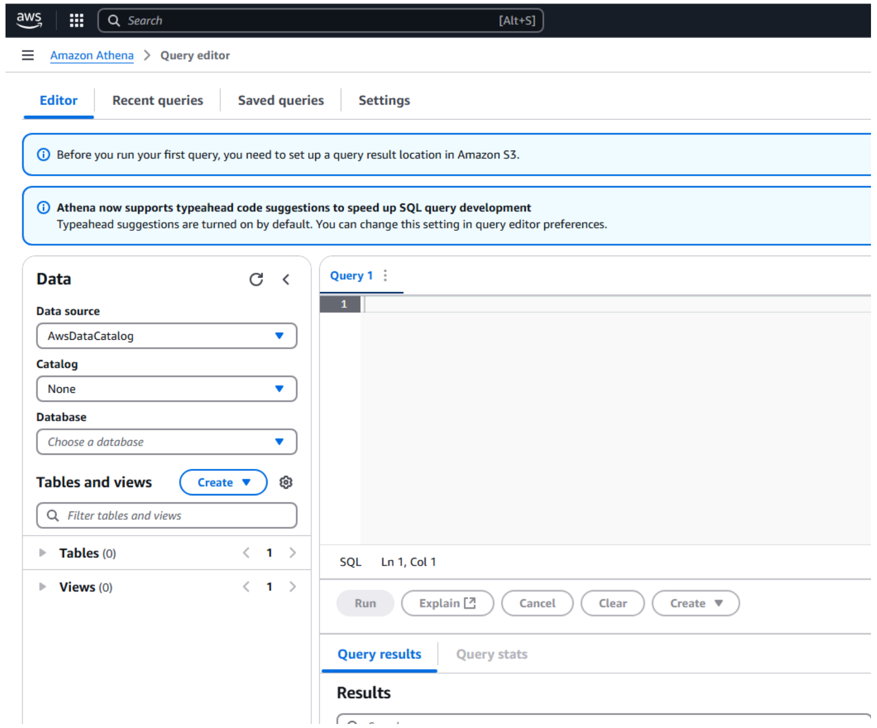
Introduction to Amazon Athena

Definition:

- AWS Athena is a serverless query service for directly analyzing data in Amazon S3 using standard SQL.
- No infrastructure to set up or manage — you pay only for the data scanned per query.
- **Ad hoc querying**: instantly run queries without ETL pipelines.

Trino and Athena

- Athena is built on the open-source Trino query engine
- Trino (formerly PrestoSQL) enables fast, scalable query processing
- Distributed SQL query engine for big data processing


Before you run your first query, you need to set up a query result location in Amazon S3.

Athena now supports typeahead code suggestions to speed up SQL query development
Typeahead suggestions are turned on by default. You can change this setting in query editor preferences.

Data source: AwsDataCatalog
Catalog: None
Database: Choose a database

Athena Core Components

1. Data Sources:

- Primarily queries data stored in Amazon S3.
- Supports various data formats, including CSV, JSON, ORC, Avro, and Parquet.

2. Query Engine:

- Built on Presto, an open-source distributed SQL query engine optimized for low-latency data analysis.

3. Integration with AWS Glue:

- Utilizes AWS Glue Data Catalog for managing metadata and schema definitions.

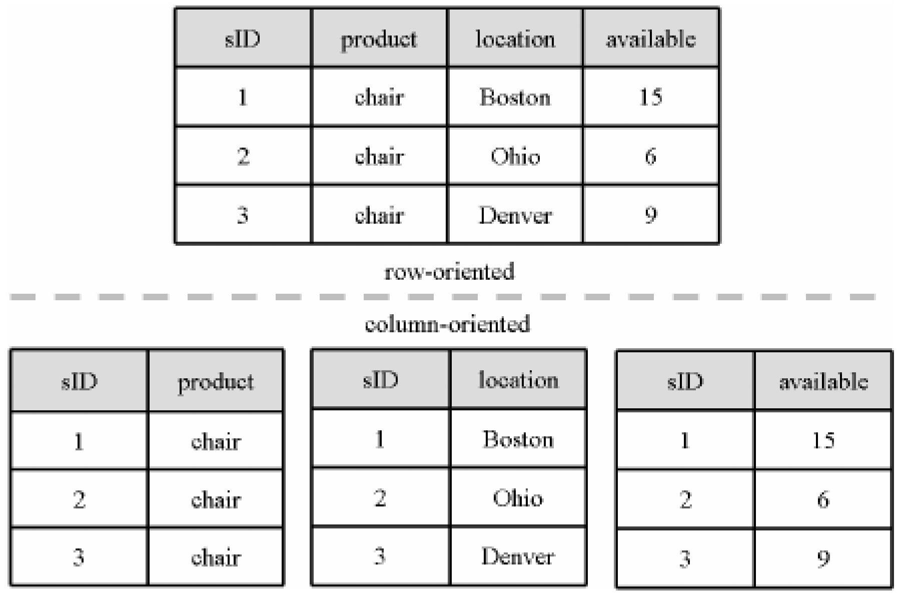
Common Data File Formats

Format	Text / Binary	Row vs Column	Schema Handling	Compression	Splittable	Typical Use Cases
CSV	Plain-text	Row	None (external schema)	No (gzip/zip external)	No (gzipped) / Yes (plain)	Spreadsheets, exports
JSON / NDJSON	Plain-text	Row (hierarchical)	Self-describing (names and values)	No (gzip/bz2 common)	No (compressed) / Yes (NDJSON)	APIs, logs, documents
Avro	Binary	Row	Required (schema in header)	Yes (Snappy/Deflate)	Yes	Kafka, service exchange
Parquet	Binary	Column	Required (embedded)	Yes (Snappy/Gzip/Zstd)	Yes	Analytics, data lakes
ORC	Binary	Column	Required (embedded)	Yes (Zlib/Snappy/Zstd)	Yes	Hive, warehousing
XML	Plain-text	Tree-structured	Self-describing (tags)	No (gzip external)	No (compressed) / Yes (plain)	Legacy configs/docs

Data Storage and Compression

Columnar Storage:

- Data is stored in columns, optimizing query performance and reducing I/O.



sID	product	location	available
1	chair	Boston	15
2	chair	Ohio	6
3	chair	Denver	9

row-oriented

column-oriented

sID	product
1	chair
2	chair
3	chair

sID	location
1	Boston
2	Ohio
3	Denver

sID	available
1	15
2	6
3	9

How Athena Works

1. Amazon S3 Bucket

Stores raw or preprocessed data in formats like CSV, JSON, Parquet, ORC, or Avro.

2. AWS Glue Crawler

Automatically scans the data in S3 to detect schema and partition information.

3. AWS Glue Data Catalog

Stores metadata (tables, columns, types, partitions) that define how Athena interprets the data.

4. Amazon Athena

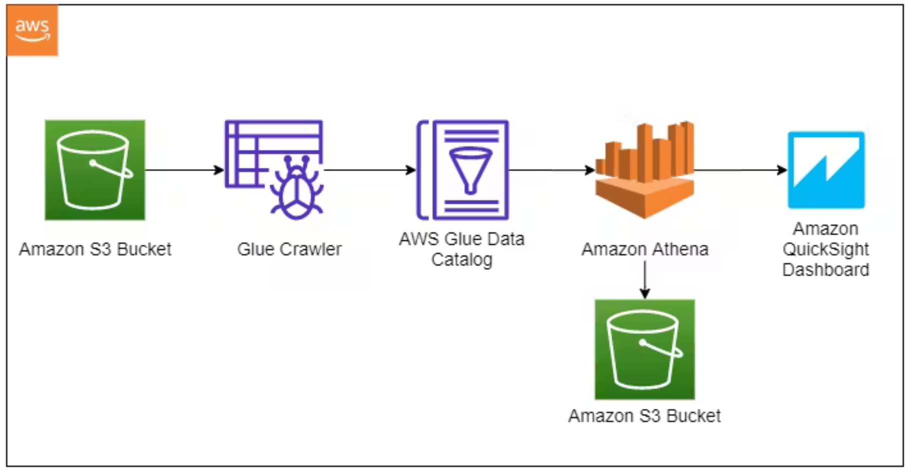
A serverless SQL query engine that uses the Glue Data Catalog to execute SQL queries directly on S3 data.

5. Amazon S3 (Query Results)

Athena writes the query results back to another (or the same) S3 bucket for further use.

6. Amazon QuickSight Dashboard

Connects to Athena to visualize the results, build dashboards, and generate business insights.



Optimizing Queries with Compression, Partitions and Buckets

Athena's cost is based on data scanned — so optimizing file formats and partitions can massively reduce costs in Big Data environments.

Golden Rule: Less data scanned = Faster query + Lower cost

Optimization Methods:

Compression

- **Purpose:** Store data using compressed formats (e.g., GZIP, Parquet) to:
- Save S3 storage space.
- Reduce the amount of data Athena scans (saving cost).

Buckets

- **Definition:** Logical grouping of related records based on **high-cardinality** fields.
- **Purpose:** Speed up query performance by minimizing full dataset scans.
- **Example:** Splitting taxi trips into separate buckets by each day's timestamp.

Partitioning

- **Definition:** Splitting data into logical directories based on **low-cardinality** fields.
- **Example:** Partitioning taxi trips by **payment type** (Credit card, Cash).

Handling Large Files

1. Use appropriate file formats (Parquet/ORC)
2. Implement effective partitioning strategies
3. Apply bucketing for high-cardinality columns
4. Use columnar compression
5. Control query scope with filters

Create a Table with Partition in Athena

```
CREATE EXTERNAL TABLE sales_records ( product_id INT, sales_amount DECIMAL(10, 2), transaction_date DATE )
PARTITIONED BY (year INT, month INT)
LOCATION 's3://your-bucket/sales-data/';
```

Amazon Glue

Introduction to AWS Glue

What is AWS Glue?

- AWS Glue is a fully managed, serverless data integration service that simplifies the process of discovering, preparing, and combining data for analytics, machine learning, and application development.

Key Features:

- **Serverless Architecture:** No infrastructure to manage; AWS handles provisioning and scaling.
- **Data Cataloging:** Centralized metadata repository for all data assets.
- **Automated ETL:** Automatically generates code to extract, transform, and load data.
- **Support for Multiple Data Sources:** Integrates with various AWS and third-party data sources.

Core Components of AWS Glue

1. **AWS Glue Data Catalog:** A centralized metadata repository that stores table definitions, job metadata, and other control information to manage your ETL environment.

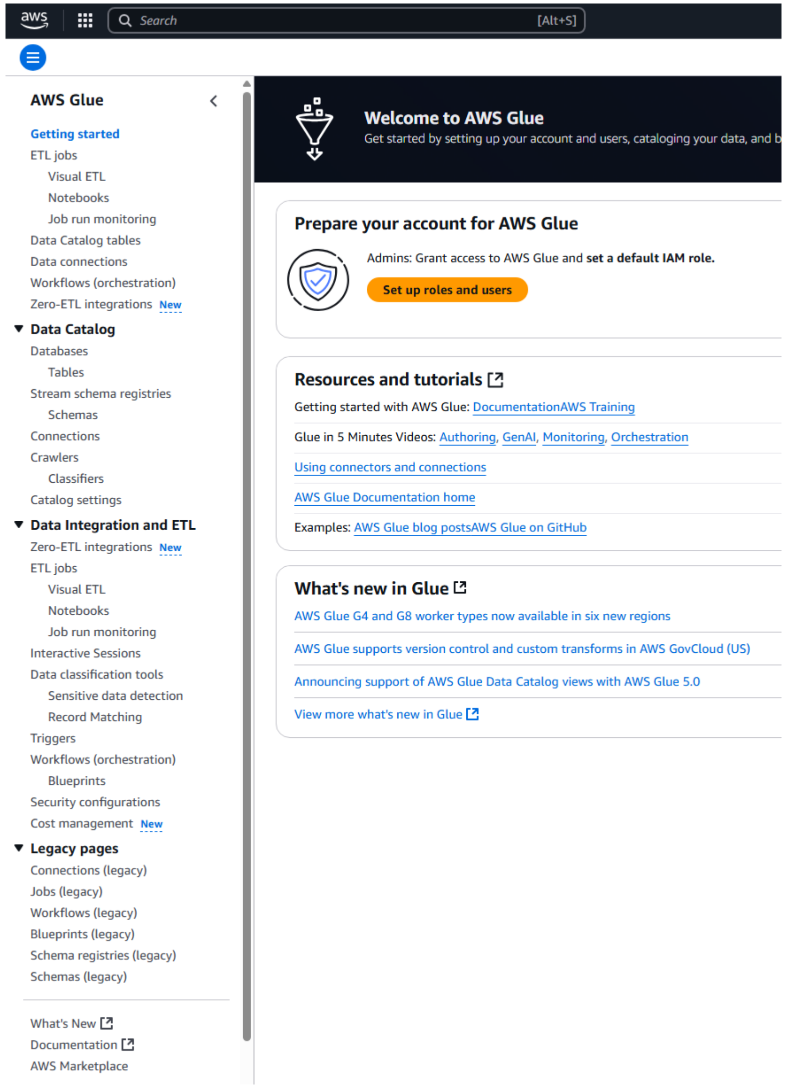
2. **Crawlers:** Automated processes that scan data sources to infer schemas and populate the Data Catalog.

3. **Jobs:** Scripts that define the ETL process, written in Python or Scala, which can be generated automatically or customized.

4. **Triggers:** Mechanisms to initiate jobs based on schedules or events.

5. **Workflows:** Orchestrate multiple jobs and crawlers to build complex ETL pipelines.

6. **Transformers**: Built-in and custom data transformation functions.



Welcome to AWS Glue

Get started by setting up your account and users, cataloging your data, and b

Prepare your account for AWS Glue

Admins: Grant access to AWS Glue and **set a default IAM role.**

Resources and tutorials

Getting started with AWS Glue: DocumentationAWS Training

Glue in 5 Minutes Videos: Authoring, GenAI, Monitoring, Orchestration

Using connectors and connections

AWS Glue Documentation home

Examples: AWS Glue blog postsAWS Glue on GitHub

What's new in Glue

AWS Glue G4 and G8 worker types now available in six new regions

AWS Glue supports version control and custom transforms in AWS GovCloud (US)

Announcing support of AWS Glue Data Catalog views with AWS Glue 5.0

View more what's new in Glue

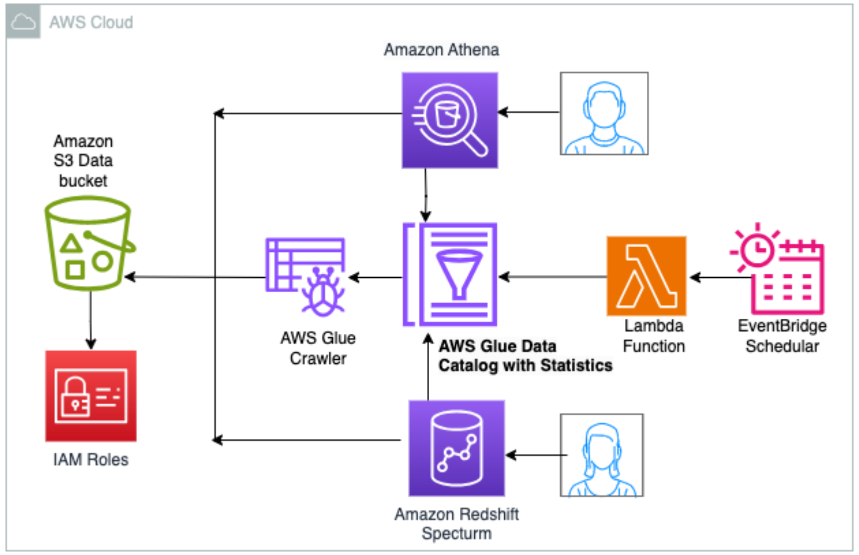
AWS Glue Data Catalog

Purpose:

- Acts as a persistent metadata store for all your data assets, enabling data discovery and schema management.

Features:

- **Schema Versioning:** Tracks changes to data schemas over time.
- **Integration:** Works seamlessly with Amazon Athena, Redshift Spectrum, and Amazon EMR.
- **Security:** Supports fine-grained access control using AWS Identity and Access Management (IAM).



AWS Glue Crawlers

Functionality:

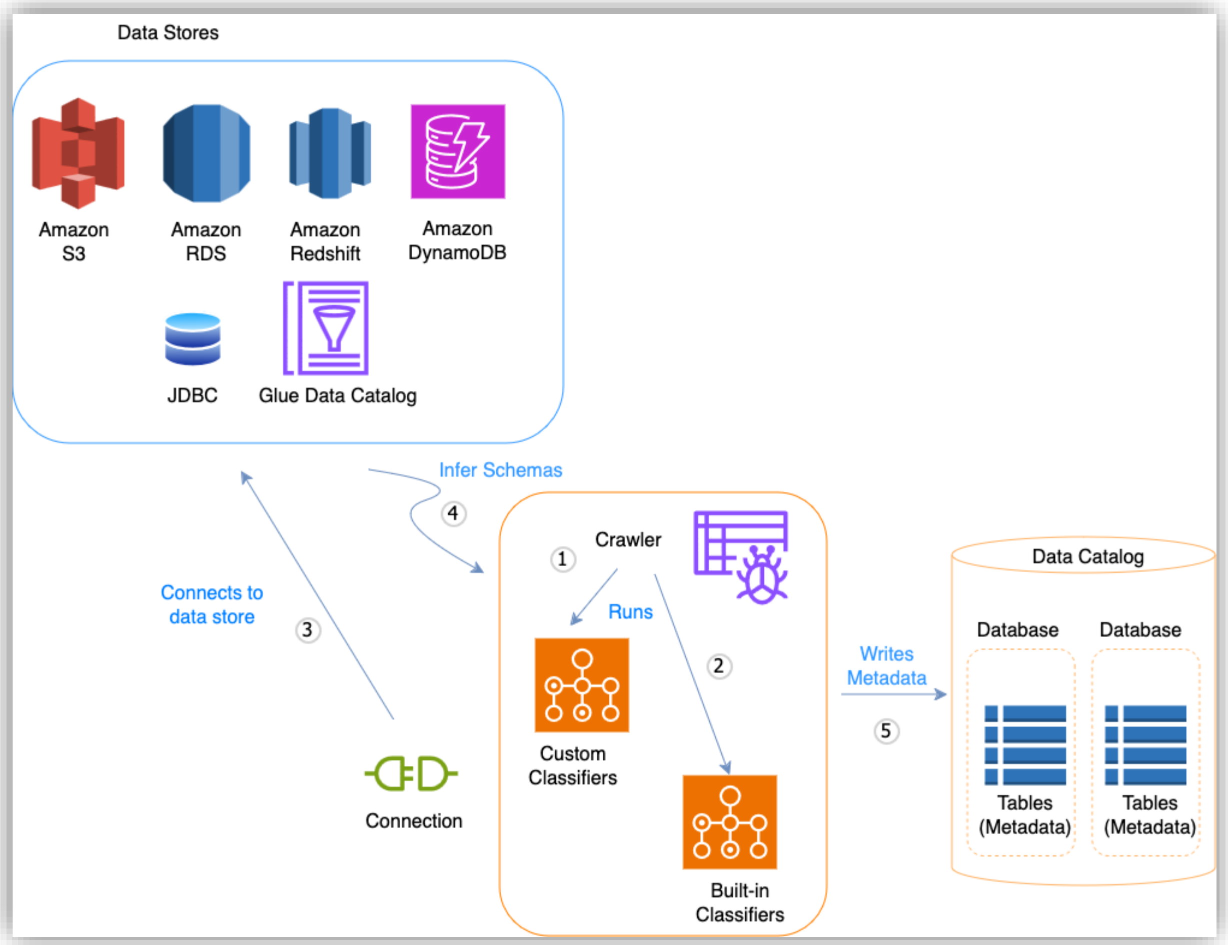
- Connect to various data sources, infer schemas, and populate the Data Catalog with metadata.

Supported Data Stores:

- Amazon S3, Amazon RDS, Amazon Redshift, DynamoDB, and more.

Custom Classifiers:

- Define custom logic to classify data formats not supported by default.



AWS Glue Jobs

Types of Jobs:

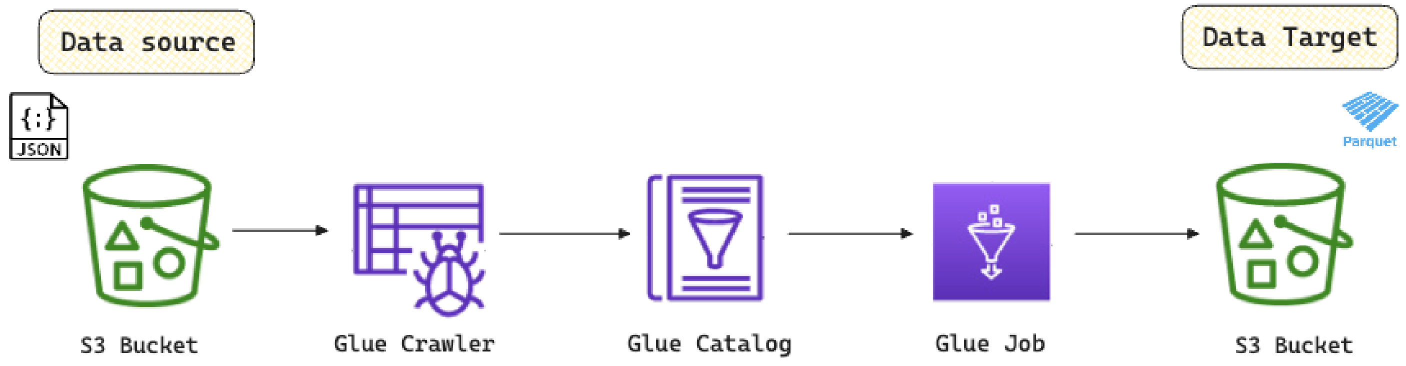
- **ETL Jobs:** Perform extract, transform, and load operations.
- **Streaming Jobs:** Process real-time data streams using Apache Spark Structured Streaming.

Development Options:

- **AWS Glue Studio:** Visual interface to create and manage jobs.
- **Script Editor:** Write custom scripts in Python or Scala.

Job Monitoring:

- Track job runs, monitor logs, and set up alerts for failures or delays.



Triggers and Workflows

Triggers:

- **Scheduled Triggers:** Initiate jobs at specified times.
- **Event-Based Triggers:** Start jobs in response to events like new data arrival.

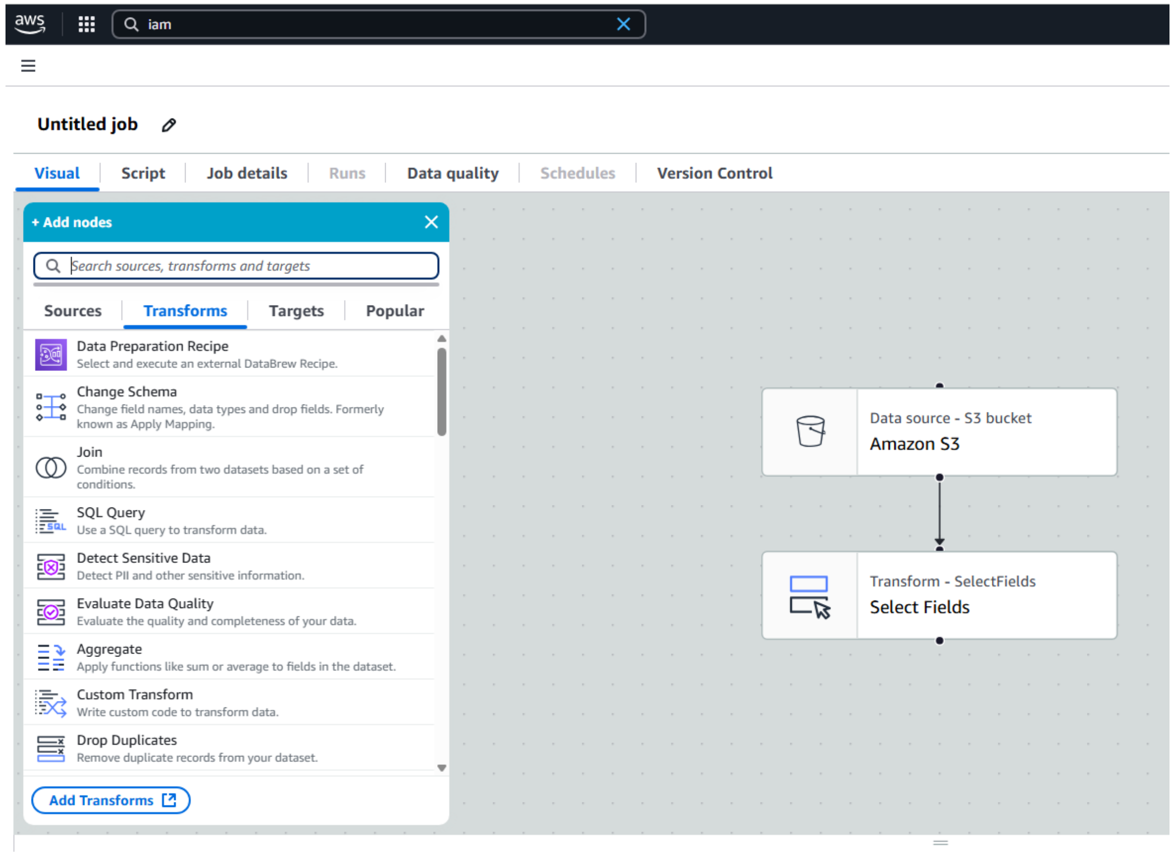
Workflows:

- Define a sequence of jobs and crawlers with dependencies to automate complex ETL processes.

AWS Glue Studio

Features:

- User-friendly interface to create, run, and monitor ETL jobs without writing code.
- Drag-and-drop functionality to design data flows.
- Automatic code generation with the option to customize scripts.


aws
iam
Untitled job
Visual
Script
Job details
Runs
Data quality
Schedules
Version Control
+ Add nodes
Search sources, transforms and targets
Sources
Transforms
Targets
Popular
Data Preparation Recipe
Select and execute an external DataBrew Recipe.
Change Schema
Change field names, data types and drop fields. Formerly known as Apply Mapping.
Join
Combine records from two datasets based on a set of conditions.
SQL Query
Use a SQL query to transform data.
Detect Sensitive Data
Detect PII and other sensitive information.
Evaluate Data Quality
Evaluate the quality and completeness of your data.
Aggregate
Apply functions like sum or average to fields in the dataset.
Custom Transform
Write custom code to transform data.
Drop Duplicates
Remove duplicate records from your dataset.
Add Transforms
Data source - S3 bucket
Amazon S3
Transform - SelectFields
Select Fields

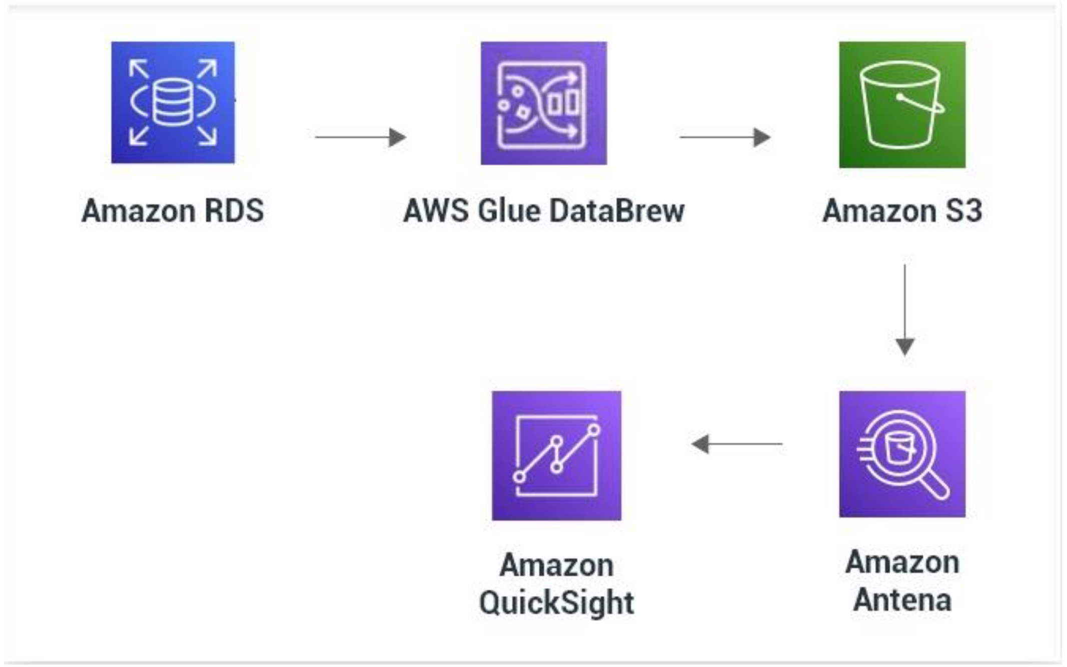
AWS Glue DataBrew

Overview:

- Visual data preparation tool for data analysts and scientists.

Capabilities:

- Clean and normalize data without writing code.
- Apply transformations, handle missing values, and create data profiles.



Module 4 Lab: Querying Data Using Amazon Athena

In this lab, you will learn how to query CSV data stored in Amazon S3 using **Amazon Athena** and **AWS Glue**.

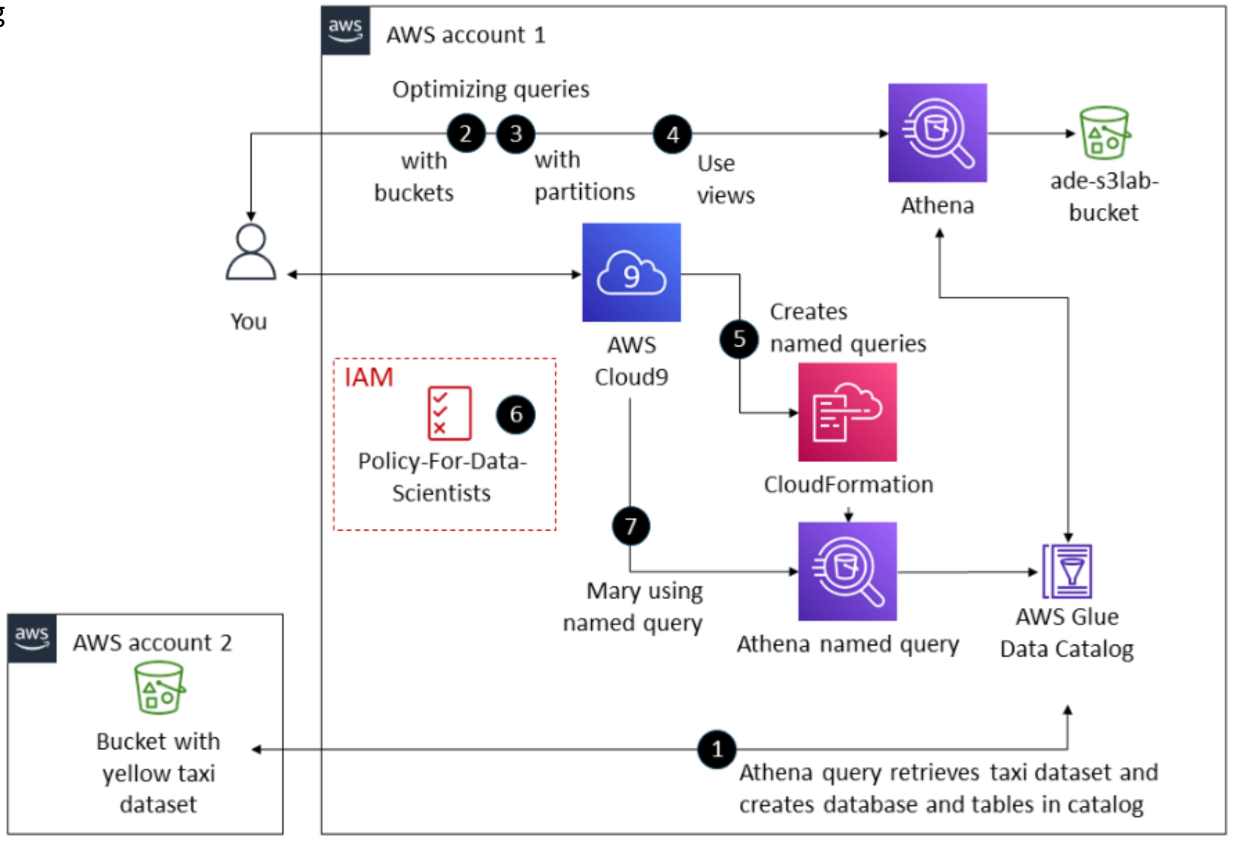
Athena enables SQL queries on S3 data without moving it, while Glue helps define and manage metadata.

You will implement a lab in AWS Academy Data Engineering Course (Module 4) where we simulate the role of a data scientist building a scalable, permission-controlled solution that supports team-wide access following the principle of least privilege.

Objectives

You will:

- Create a Glue database and table via the Athena editor
- Define schema and use bulk column features
- Query and optimize data in S3 with Athena
- Create views and named queries
- Deploy named queries using CloudFormation
- Validate IAM access through AWS CLI
- This training uses a 2017 taxi trip dataset and is restricted to required AWS services only.



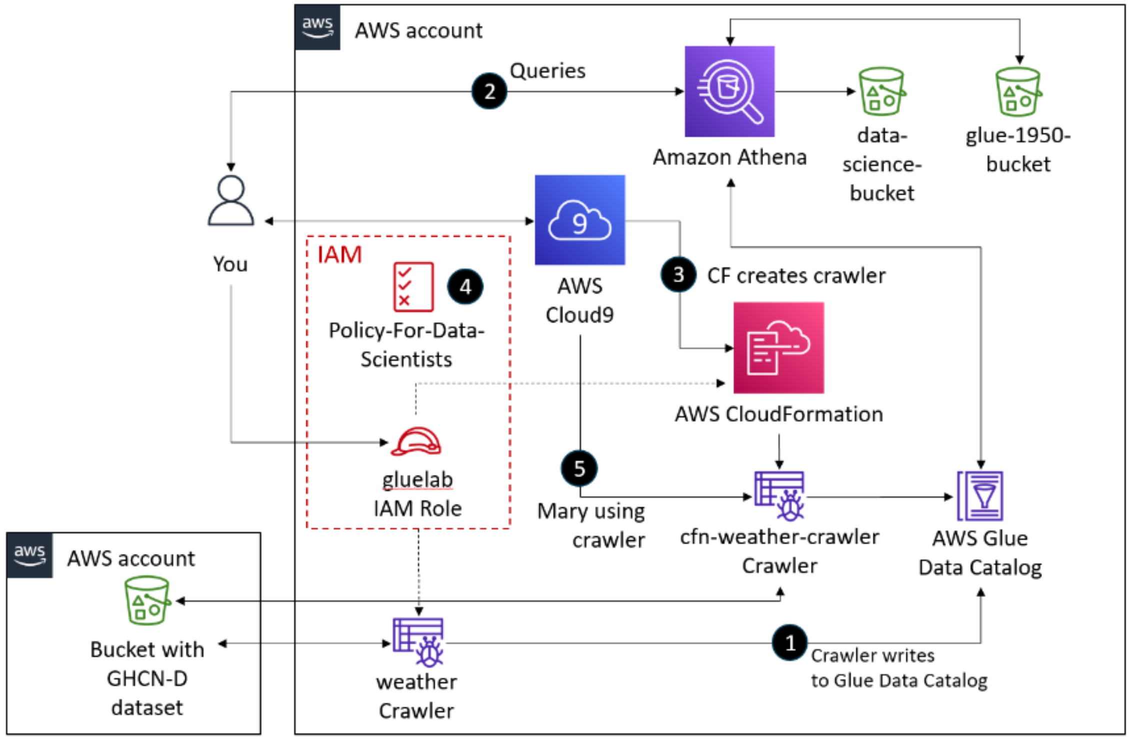
Module 7 Lab: Performing ETL on a Dataset by Using AWS Glue

In this lab, you will learn how to use AWS Glue to import a dataset from Amazon Simple Storage Service (**Amazon S3**).

You will then extract the data, transform its schema, and load the dataset into an **AWS Glue** database for later analysis by using Athena.

After completing this lab, you will be able to do the following:

- Access AWS Glue in the AWS Management Console and create a **crawler**.
- Create an AWS Glue database with tables and a schema by using a crawler.
- Query data in the AWS Glue database by using Athena.
- Create and deploy an AWS Glue crawler by using an AWS CloudFormation template.
- Review an AWS Identity and Access Management (IAM) policy for users to run an AWS Glue crawler and query an AWS Glue database in Athena.
- Confirm that a user with the IAM policy can use the AWS Command Line Interface (AWS CLI) to access the AWS Glue database that the crawler created.
- Confirm that a user can run the AWS Glue crawler when source data changes.



amazon
REDSHIFT

Amazon Redshift

Amazon Redshift is a fully managed data warehouse service offered by Amazon Web Services (AWS).

It is designed for large-scale data storage and fast query performance, making it suitable for data analytics and business intelligence workloads.

Here are some key features and benefits of Amazon Redshift:

Key Features and Benefits:

- **High Performance**:
  - Redshift uses columnar storage and data compression to reduce the amount of data read from disk, improving query performance.
  - Massively parallel processing (MPP) architecture allows Redshift to distribute and parallelize queries across multiple nodes.
- **Scalability**:
  - Redshift allows you to start with a small cluster and scale up to petabytes of data.
  - You can easily resize your Redshift cluster by adding or removing nodes.

Amazon Redshift

- **Cost-Effective**:
  - Pay only for the resources you use with on-demand pricing.
  - Reserved instance pricing and the ability to pause and resume clusters can further reduce costs.
- **Managed Service**:
  - AWS manages the setup, operation, and scaling of Redshift, including backups, patching, and monitoring.
  - Automatic backups and snapshots provide data protection and recovery.
- **Data Integration**:
  - Redshift integrates with various AWS services such as Amazon S3, Amazon RDS, Amazon DynamoDB, and AWS Glue for data ingestion and ETL processes.
  - Supports integration with third-party ETL tools and BI platforms.
- **Advanced Analytics**:
  - Supports SQL-based querying and analytics.
  - Redshift Spectrum allows you to run queries on data stored in Amazon S3 without loading it into Redshift.
  - Machine learning integration with Amazon SageMaker for predictive analytics.

Core Components of Amazon Redshift

1. Clusters:

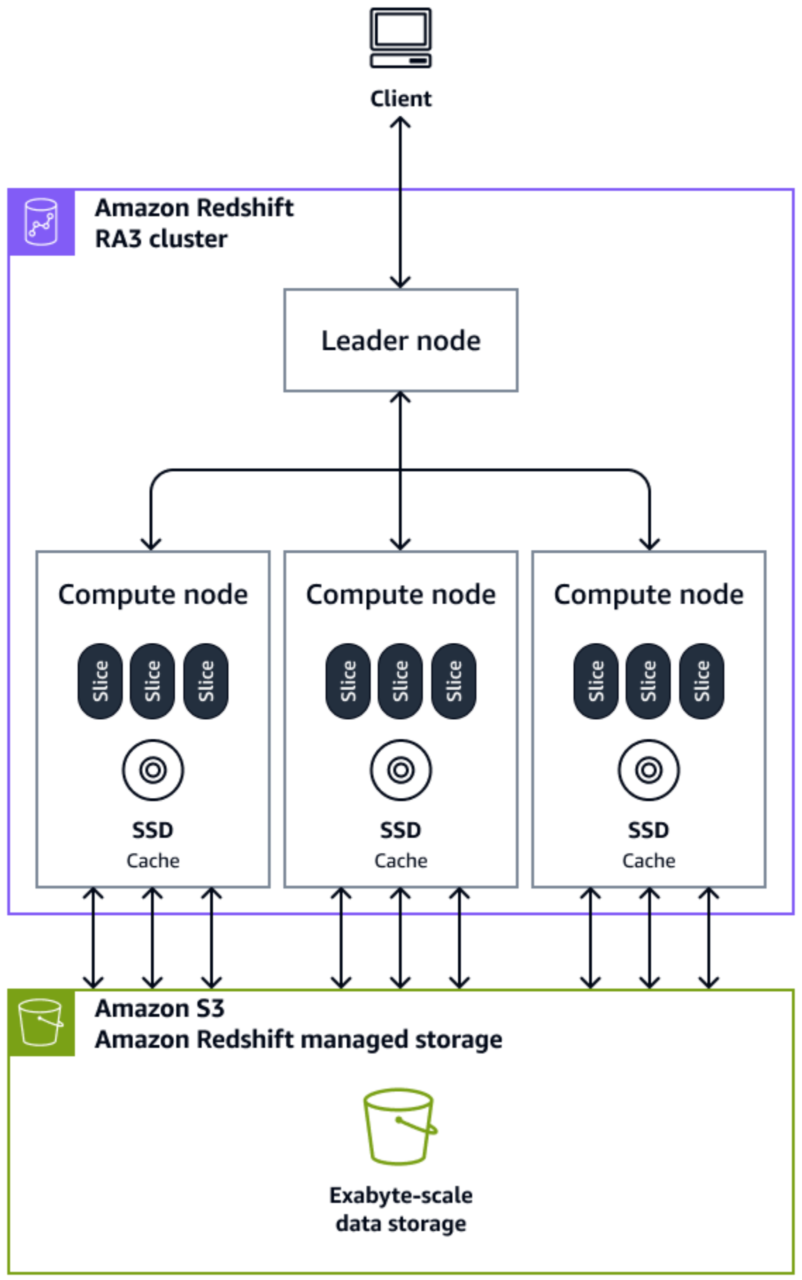
- The core infrastructure component, composed of one or more compute nodes.

2. Leader Node:

- Manages communications with client programs and all communication with compute nodes.

3. Compute Nodes:

- Handle data processing tasks and store data.



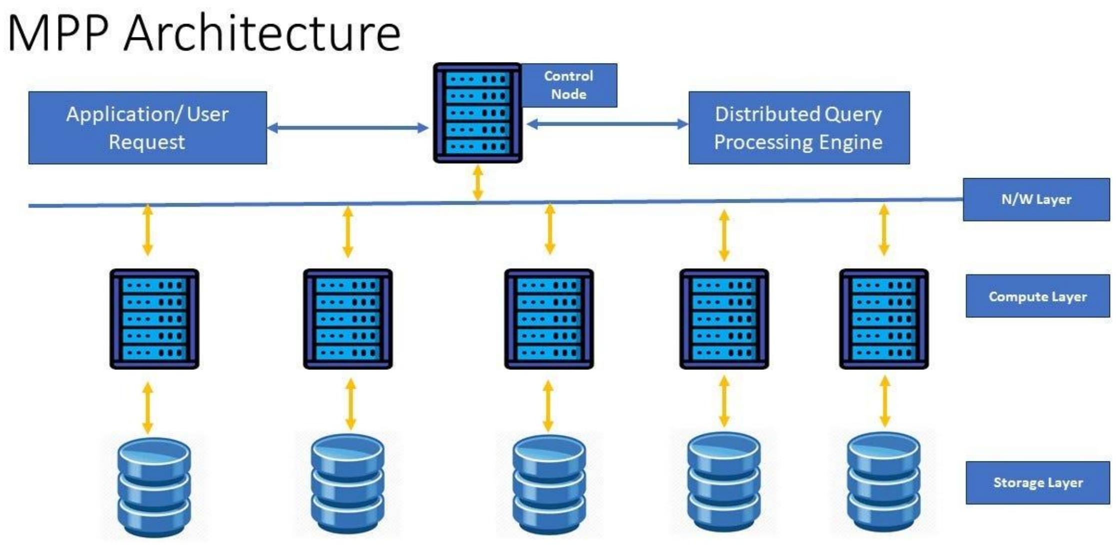
Query Processing and Optimization

Massively Parallel Processing (MPP):

- Distributes data and query load across multiple nodes for efficient processing.

Query Optimization:

- Redshift's query optimizer evaluates multiple execution strategies and selects the most efficient one.



Redshift Deployment Options

1. Provisioned Clusters

With provisioned clusters, you manage your Amazon Redshift resources manually.

This option gives you complete control over cluster sizing, scaling, and maintenance windows.

2. Redshift Serverless

Amazon Redshift Serverless lets you access and analyze data without the configurations of a provisioned data warehouse.

Resources are automatically provisioned, and capacity is intelligently scaled to deliver fast performance for demanding workloads.

Key benefits of Redshift Serverless:

- **i. Automatic resource provisioning**: No need to manually configure cluster size
- **ii. Pay-per-use pricing**: You don't incur charges when the data warehouse is idle
- **iii. Auto-scaling**: Capacity adjusts based on workload demands
- **iv. Simplified management**: No need to monitor or tune the cluster

Redshift Serverless organizes resources using namespaces (for database objects) and workgroups (for compute resources).

Workgroups house compute resources like Redshift Processing Units (RPUs), security groups, and usage limits.

Use Cases

Data Warehousing:

- Centralized repository for structured and semi-structured data.

Business Intelligence:

- Supports integration with BI tools like Tableau, Looker, and Amazon QuickSight.

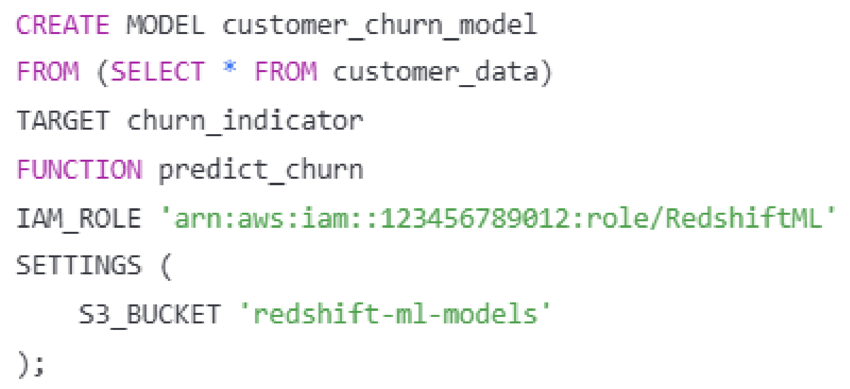
Machine Learning:

- Provides data for training and inference in machine learning models.

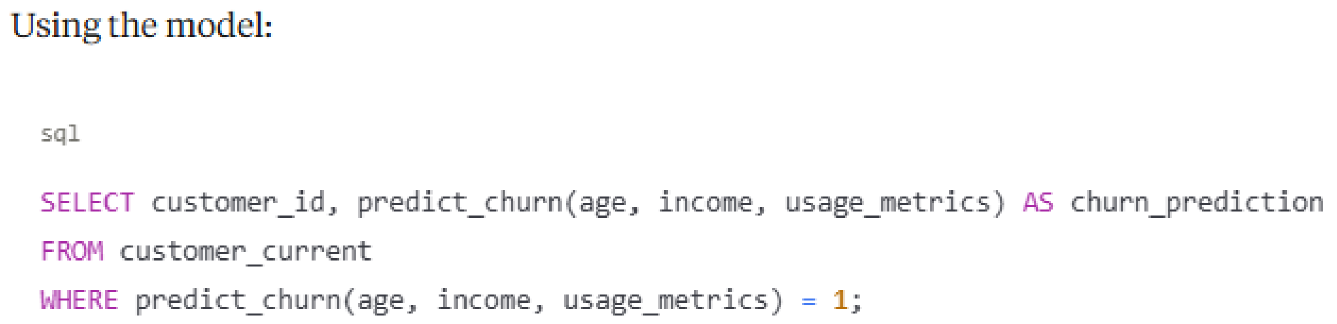
Redshift ML

- Amazon Redshift ML makes it easy for data analysts and database developers to create, train, and apply machine learning models using familiar SQL commands.
- It leverages Amazon SageMaker without requiring users to learn new tools or languages.

Create Customer Churn Model



```sql
CREATE MODEL customer_churn_model
FROM (SELECT * FROM customer_data)
TARGET churn_indicator
FUNCTION predict_churn
IAM_ROLE 'arn:aws:iam::123456789012:role/RedshiftML'
SETTINGS (
    S3_BUCKET 'redshift-ml-models'
);
```



Using the model:

sql

```sql
SELECT customer_id, predict_churn(age, income, usage_metrics) AS churn_prediction
FROM customer_current
WHERE predict_churn(age, income, usage_metrics) = 1;
```

Setting Up AWS Redshift

Setting Up a Provisioned Cluster

1. **Sign in to AWS Console**: Navigate to the Redshift service
2. **Create Cluster**: Choose "Create cluster" and configure settings:
    1. Cluster identifier
    2. Node type
    3. Number of nodes
    4. Database configurations (name, port, credentials)
    5. Network and security settings
    6. Maintenance and backup options
3. **Launch Cluster**: Review settings and create the cluster

----

Setting Up Redshift Serverless

1. **Sign in to AWS Console**: Navigate to the Redshift service
2. **Create Serverless Workspace**: Configure settings:
    1. Namespace name (for database objects)
    2. Workgroup name (for compute resources)
    3. Network and security settings
    4. RPU capacity settings (optional)
3. **Create Workspace**: Review settings and create the serverless workspace

BUSINESS INTELLIGENCE AND DATA ANALYTICS – WE PREDICT THE FUTURE –

Institute of Analytics Recognized Programme

University of Petra

amazon EMR

Introduction to Amazon EMR

What is Amazon EMR?

- Amazon EMR (Elastic MapReduce) is a managed cloud-native **Big Data Platform** provided by Amazon Web Services (AWS).
- It simplifies running big data frameworks, such as **Apache Hadoop** and **Apache Spark**, on AWS to process and analyze vast amounts of data.

Key Features:

- **Scalability:** Handles large-scale data sets, allowing dynamic scaling of resources based on workload demands.
- **Performance:** Utilizes distributed processing frameworks to decrease command execution time.
- **Simplified Management**: AWS handles provisioning, configuration, and tuning of clusters
- **Flexibility**: Supports multiple deployment options and various open-source big data frameworks
- **Integration:** Seamlessly integrates with various AWS services like Amazon S3, Amazon RDS, and Amazon DynamoDB.
- **Security**: Integrates with AWS security services for comprehensive protection

Use Cases

Use Case	Description	Tools/Frameworks	Example
Big Data Processing (ETL)	Transform raw data into usable formats	Apache Spark, Hadoop MapReduce	Daily ETL jobs to process clickstream data and store it in S3
Data Warehousing & Analytics	Perform complex queries on large datasets	Hive, Presto, Trino	Ad-hoc analysis of petabytes of user logs for BI reports
Machine Learning	Train ML models on distributed data	Spark MLlib, integration with SageMaker	Recommendation engine on product-user interaction data
Real-time Stream Processing	Ingest and analyze real-time data streams	Spark Streaming, Apache Flink	Fraud detection on real-time transaction streams
Log and Event Analysis	Analyze system logs for monitoring and security	Hadoop, Spark, Hive	Parsing application logs for error pattern detection
Genomics & Scientific Research	Process large scientific datasets	Custom Hadoop/Spark applications	Genome sequencing analysis across distributed nodes
Graph Processing	Perform graph-based computations	Apache Giraph, Spark GraphX	Social network analysis for influencer detection
Web Indexing & Crawling	Process and index web content	Hadoop, custom MapReduce	Large-scale web crawling and search index creation
Data Lake Analytics	Query data directly from Amazon S3 without loading into a database	Hive, Presto, Trino	Analyzing semi-structured data in an S3-based data lake
Financial Risk Modeling	Run complex financial simulations	Spark, Scala/Python	Monte Carlo simulations for value-at-risk (VaR) analysis

EMR Architecture Overview

Core Components

AWS EMR architecture consists of several layers, each providing specific functionality to the cluster:

1. **Storage Layer**
   1. Amazon S3 (primary data store)
   2. HDFS (Hadoop Distributed File System)
   3. EMRFS (EMR File System)
   4. Local instance storage
2. **Cluster Layer**
   1. **Primary Node (formerly Master Node)**: Manages the cluster and coordinates distribution of data and tasks
   2. **Core Nodes**: Run tasks and store data in HDFS
   3. **Task Nodes**: Run tasks but don't store data
3. **Processing Layer**
   1. Open-source frameworks like Hadoop, Spark, Hive, and others
   2. EMR-optimized versions of these frameworks

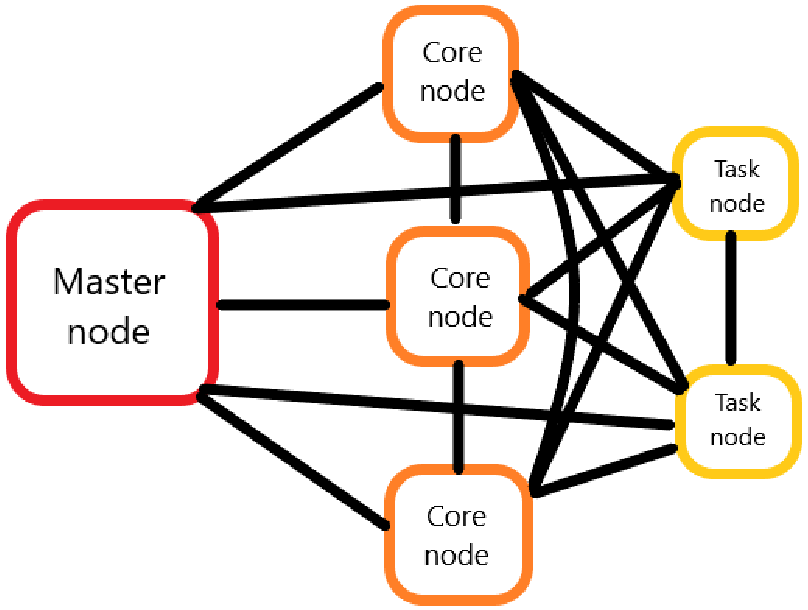
Core Components of Amazon EMR

1. Clusters:

- The central component of Amazon EMR, composed of Amazon EC2 instances.
- Each instance in the cluster is called a **Node**.

2. Node Types:

- **Primary Node:** Manages the cluster by running software components to coordinate the distribution of data and tasks among other nodes for processing.
- **Core Node:** Runs tasks and stores data in the Hadoop Distributed File System (HDFS) on your cluster.
- **Task Node:** Runs tasks but does not store data in HDFS. Task nodes are optional.



Data Storage and Processing

Storage Options:

- **Amazon S3:** EMR File System (EMRFS) allows EMR clusters to directly access data stored in Amazon S3 as if it were a file system like HDFS.
- **HDFS:** A distributed and scalable Hadoop file system enabling data distribution across instances in the cluster.

Processing Frameworks:

- **Apache Hadoop MapReduce:** An open-source programming model for distributed computing.
- **Apache Spark:** A cluster framework and programming model for processing big data workloads, supporting multiple interactive query modules such as SparkSQL

EMR Studio and Notebooks

EMR Studio:

- An integrated development environment (IDE) that makes it easy for data scientists and data engineers to develop, visualize, and debug data engineering and data science applications written in R, Python, Scala, and PySpark.

EMR Notebooks:

- Provides a managed environment based on Jupyter Notebooks for interactive data analysis and visualization.

Deployment Options

AWS EMR offers different deployment options to meet different operational requirements:

1. EMR on EC2

The traditional deployment model where EMR runs on Amazon EC2 instances.

Key features:

- Full control over cluster configuration and instance selection
- Customizable AMIs (Amazon Machine Images)
- Ability to install custom software
- Ideal for workloads requiring specific hardware configurations

Use when:

- You need maximum control and flexibility
- You want to customize underlying infrastructure
- You need specific EC2 instance types for performance optimization

Deployment Options

2. **A serverless deployment** option that eliminates the need to configure, manage, or scale clusters.

Key features:

- Automatic resource provisioning and scaling
- Pay only for resources used by your applications
- Zero infrastructure management
- Currently supports Apache Spark and Apache Hive

Use when:

- You want to simplify operations
- You have variable or unpredictable workloads
- You want to avoid managing clusters
- You prefer focusing on applications rather than infrastructure

Data Processing Frameworks

EMR supports a variety of open-source frameworks for data processing. The latest EMR releases include:

Apache Hadoop

The foundation framework providing:

- HDFS (Hadoop Distributed File System)
- YARN (resource management)
- MapReduce (programming model)

Apache Spark

A unified analytics engine offering:

- In-memory processing
- Support for SQL, streaming, machine learning, and graph processing
- Performance up to 100x faster than traditional MapReduce

Apache Hive

Data warehouse software providing:

- SQL-like interface (HiveQL)
- Structured data querying capabilities
- Integration with various storage system

Data Processing Frameworks

Presto and Trino

Distributed SQL query engines for:

- High-performance interactive analytics
- Querying data across multiple sources

Apache Flink

A stream processing framework offering:

- Real-time data processing
- Event-time processing
- State management

HBase

A NoSQL database providing:

- Random, real-time read/write access
- Storage for large tables with billions of rows

Data Processing Patterns

1. **Batch Processing**:
    1. Processing large volumes of data periodically
    2. ETL jobs for data warehousing
2. **Interactive Analysis**:
    1. SQL queries with Hive, Presto, or Spark SQL
    2. Ad-hoc analysis in notebooks
3. **Streaming Processing**:
    1. Real-time data processing with Spark Streaming or Flink
    2. Processing data from Kinesis or Kafka
4. **Machine Learning**:
    1. Building and deploying models with Spark MLlib
    2. Feature engineering and model scoring

Critical Metrics to Monitor

1. **Cluster Health**:
    1. Node status
    2. Service availability
    3. HDFS health
2. **Resource Utilization**:
    1. CPU usage
    2. Memory consumption
    3. Disk space
    4. Network I/O
3. **Application Metrics**:
    1. Job completion rates
    2. Processing latency
    3. Error rates
    4. Throughput

Cost Management

EMR Pricing Models

1. **On-Demand Instances**:
    1. Pay per hour with no long-term commitments
    2. Higher rate but maximum flexibility
2. **Reserved Instances**:
    1. Lower hourly rate with 1 or 3-year commitments
    2. Significant discount for predictable workloads
3. **Spot Instances**:
    1. Use unused EC2 capacity at steep discounts
    2. Subject to interruption based on capacity availability
4. **EMR Serverless Pricing**:
    1. Pay only for vCPU, memory, and storage resources used
    2. No need to pay for idle capacity

Cost Optimization Strategies

1. **Cluster Management**:
   1. Use transient clusters for batch processing
   2. Leverage auto-termination for idle clusters
   3. Implement managed scaling
2. **Instance Strategy**:
   1. Mix instance types (On-Demand, Spot)
   2. Use Instance Fleets for flexible allocation
   3. Leverage Graviton processors for better price-performance
3. **Storage Optimization**:
   1. Use S3 for persistent storage
   2. Compress data appropriately
   3. Implement lifecycle policies for old data
4. **Workload Optimization**:
   1. Improve job efficiency to reduce runtime
   2. Schedule jobs during off-peak hours
   3. Batch small processing jobs

Cost Monitoring

1. **AWS Cost Explorer**:
    1. Track EMR spending by cluster, user, or project
    2. Analyze cost trends
    3. Create budget alerts
2. **AWS Budgets**:
    1. Set spending limits
    2. Configure alerts for potential overruns
3. **Cost Allocation Tags**:
    1. Tag resources for cost attribution
    2. Track costs by department, project, or environment

Setting Up AWS EMR

Setting Up EMR on EC2

1. **Sign in to AWS Console**: Navigate to the EMR service
2. **Create Cluster**: Choose "Create cluster" and configure:
    1. Cluster name
    2. Release version (applications and versions)
    3. Instance types and count
    4. Networking settings
    5. Security configurations
    6. Bootstrap actions (optional)
3. **Launch Cluster**: Review settings and create

Setting Up EMR Serverless

1. **Sign in to AWS Console**: Navigate to the EMR service
2. **Create Application**:
    1. Choose application type (Spark or Hive)
    2. Select release version
    3. Configure capacity settings (optional)
    4. Set up network and security options
3. **Submit Jobs**: Use the EMR Studio, AWS CLI, or SDK to submit jobs

Hands-On Exercises

Exercise 1: Setting Up an EMR Cluster

1. Launch a basic EMR cluster using the AWS Management Console
2. Connect to the cluster using SSH
3. Explore the cluster web interfaces (YARN, Spark, etc.)
4. Submit a simple job and monitor its execution

Exercise 2: Processing Data with Spark

1. Prepare sample dataset in S3
2. Create a Spark application to process the data
3. Submit the application to EMR
4. Analyze the results and optimize the application

Exercise 3: EMR Serverless Deployment

1. Create an EMR Serverless application
2. Configure pre-initialized capacity
3. Submit Spark or Hive jobs
4. Monitor job execution and resource usage

Exercise 4: Security Implementation

1. Create a security configuration with encryption
2. Set up Kerberos authentication
3. Implement Lake Formation integration
4. Test fine-grained access controls

Exercise 5: Performance Optimization

1. Benchmark a standard EMR configuration
2. Apply performance optimization techniques
3. Measure the impact of optimizations
4. Develop a performance tuning checklist

Lab: Processing Logs by Using Amazon EMR

In this lab, you will use Amazon EMR to process logs that are stored in an S3 bucket.

To process the data in Amazon EMR, you will use Apache Hive.

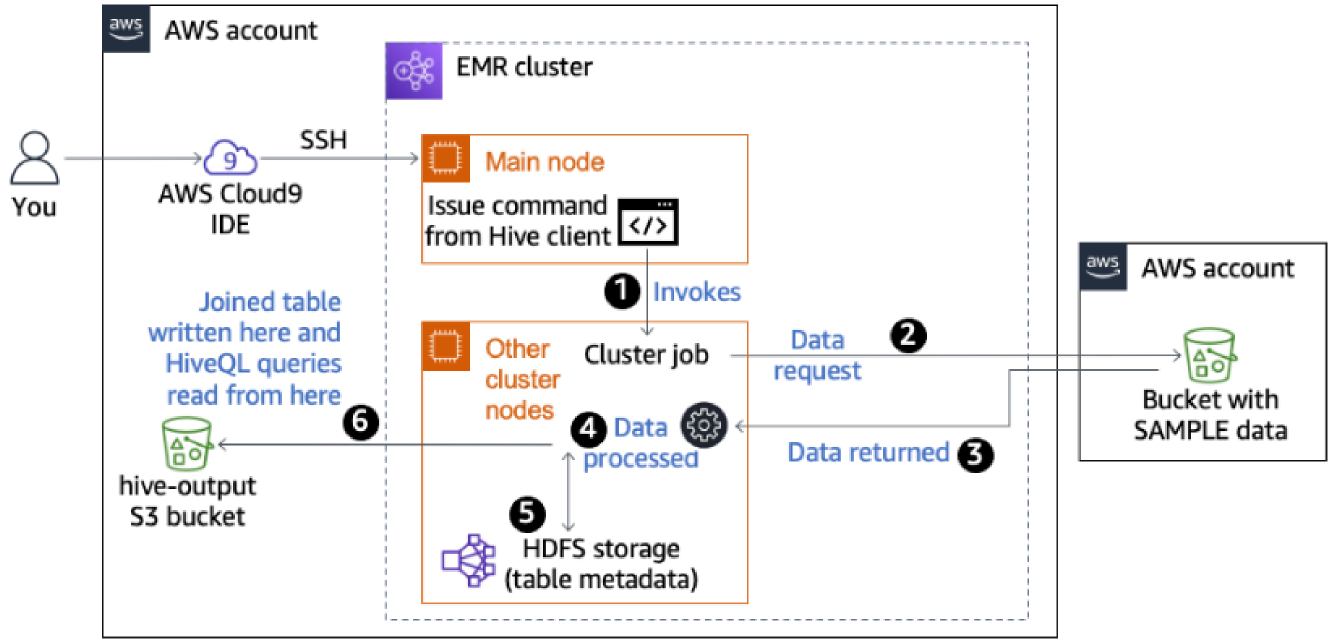
Hive is an open-source data warehouse infrastructure framework that is built on top of Hadoop and is often used to process large datasets.

With Hive, you can query large datasets with SQL-like queries.

When you run a Hive query, the request is translated into a Yet Another Resource Negotiator (YARN) job. YARN is the part of the Hadoop software that coordinates job scheduling.

After completing this lab, you should be able to do the following:

- Launch an EMR cluster through the AWS Management Console.
- Run Hive interactively.
- Use Hive commands to create tables from log data that is stored in Amazon S3.
- Use Hive commands to join tables and store the joined table in Amazon S3.
- Use Hive to query tables that are stored in Amazon S3.



IOA Institute of Analytics
Recognized Programme

University of Petra

amazon
Kinesis

What is Amazon Kinesis?

AWS Kinesis is a suite of services designed to collect, process, and analyze real-time, streaming data at massive scale. This means you can get timely insights and react quickly to new information.

Why use Kinesis?
Imagine you need to:

- Analyze clickstreams from your website in real-time to personalize user experience.
- Process IoT sensor data from thousands of devices.
- Ingest application logs for immediate anomaly detection.
- Feed data into real-time dashboards or machine learning models.
- Kinesis provides the building blocks for these scenarios.

Core Kinesis Services

Kinesis is not a single service but a family. The main ones are:

1. **Kinesis Data Streams (KDS):** The foundational service. It's a massively scalable and durable real-time data streaming service. You build custom applications to process or analyze streaming data.
2. **Kinesis Data Firehose:** The easiest way to reliably load streaming data into data lakes, data stores, and analytics services. It's more about *delivery* than custom processing.
3. **Kinesis Data Analytics:** Allows you to process and analyze streaming data using standard SQL or Apache Flink. It can read from Data Streams or Firehose.
4. **Kinesis Video Streams:** Securely streams video from connected devices to AWS for analytics, machine learning (ML), playback, and other processing.

Kinesis Data Streams (KDS)

Concept: Think of a Kinesis Data Stream as a highly available, durable pipe for your data. Data flows into the stream as **records**.

Key Terminology:

- **Data Record:** The unit of data stored in a Kinesis stream. It consists of a sequence number, a partition key, and a data blob (up to 1MB).
- **Producer:** An application that puts data records into a Kinesis Data Stream (e.g., web servers, IoT devices, mobile apps).
- **Consumer:** An application that gets records from a Kinesis Data Stream and processes them (e.g., an EC2 instance, Lambda function, Kinesis Data Analytics application).
- **Shard:** The base throughput unit of a Kinesis Data Stream.
    - Each shard provides a capacity of 1MB/second data input and 2MB/second data output.
    - One shard can support up to 1000 PUT records per second.
    - You specify the number of shards when you create a stream. You can scale the number of shards up or down.
    - **Ordering is guaranteed within a shard.**
- **Partition Key:** Used to group data by shard within a stream. Kinesis uses the partition key to segregate and route records to different shards. Records with the same partition key go to the same shard. This is crucial for ordered processing of related records.
- **Sequence Number:** A unique identifier for each record within its shard. Kinesis assigns it after you write the record.

Kinesis Data Streams (KDS)

How it Works:

1. **Producers** (e.g., your application server, an IoT device) send data records to your Kinesis Data Stream using the AWS SDK, Kinesis Producer Library (KPL), or Kinesis Agent.
2. When sending a record, you specify the stream name, a **partition key**, and the data itself.
3. Kinesis hashes the partition key and maps it to a specific **shard**.
4. Data is stored in shards for a configurable **retention period** (default 24 hours, max 365 days).
5. **Consumers** (e.g., EC2 instances running custom code, AWS Lambda functions, Kinesis Data Analytics) read data from the shards.
    1. **Shared Throughput Consumers (Classic):** Each consumer gets a fraction of the shard's 2MB/s output. Max 5 consumers per shard this way.
    2. **Enhanced Fan-Out Consumers:** Each consumer gets its own dedicated 2MB/s throughput per shard, allowing many consumers to read from the same stream without impacting each other.

When to Use KDS:

- You need custom real-time data processing.
- You require low latency (sub-second).
- You need ordered processing for specific keys.
- You want to feed data into multiple real-time applications.
- Building real-time dashboards, anomaly detection, real-time metrics.

Kinesis Data Firehose

Concept: Firehose is all about *delivering* streaming data to a destination. It's a fully managed service that requires minimal to no coding for basic delivery.

Key Terminology:

- **Delivery Stream:** The Firehose entity you create.
- **Source:** Where Firehose gets the data (e.g., Kinesis Data Streams, direct PUT from producers, CloudWatch Logs, Events).
- **Destinations:** Where Firehose sends the data:
    - Amazon S3
    - Amazon Redshift (often via S3)
    - Amazon Elasticsearch Service (now Amazon OpenSearch Service)
    - Splunk, a real-time search and analysis engine to quickly and easily search through large volumes of log data
    - Custom HTTP endpoints or supported third-party partner destinations.
- **Data Transformation (Optional):** You can use an AWS Lambda function to transform incoming source data before delivering it (e.g., convert JSON to CSV, filter records).
- **Buffering:** Firehose buffers incoming data (by size or time) before delivering it to the destination to optimize for batch writes.

Kinesis Data Firehose

How it Works:

1. Producers send data to a Firehose Delivery Stream (or Firehose reads from a Kinesis Data Stream).
2. Firehose buffers the data.
3. (Optional) Firehose invokes your Lambda function for data transformation.
4. Firehose delivers the (transformed) data to the configured destination in batches.
5. It handles retries, scaling, and monitoring automatically.

When to Use Firehose:

- You need to load streaming data into S3, Redshift, OpenSearch, Splunk, or HTTP endpoints with minimal effort.
- You want a managed service that handles scaling, retries, and batching.
- Simple transformations (via Lambda) are sufficient.
- You don't need sub-second processing latency for the *delivery* part (buffering introduces some latency).

KDS vs. Firehose:

- **KDS:** For real-time *processing* with custom consumer applications. You manage consumers.
- **Firehose:** For real-time *delivery* to destinations. AWS manages delivery.
- You can use them together: KDS for initial ingestion and fan-out to multiple consumers, one of which could be Firehose for archiving raw data to S3.

Kinesis Data Analytics

Concept: Process and analyze streaming data in real-time using standard SQL or Apache Flink.

Two Flavors:

- **Kinesis Data Analytics for SQL Applications:**
    - Write standard SQL queries on your streaming data.
    - Define a schema for your incoming data.
    - Use concepts like tumbling windows, sliding windows, and anomaly detection functions.
    - Output results to another Kinesis Data Stream, a Firehose delivery stream, or a Lambda function.
- **Kinesis Data Analytics for Apache Flink:**
    - Build sophisticated stream processing applications using Java, Scala, or Python with the Apache Flink framework.
    - Offers more flexibility and power than SQL for complex event processing, stateful computations, etc.
    - Fully managed Flink environment.

Kinesis Data Analytics

How it Works:

1. Define an input: a Kinesis Data Stream or a Kinesis Data Firehose delivery stream.
2. Write your SQL query or Flink application code.
3. (SQL) Kinesis Data Analytics infers a schema or you can define it.
4. (SQL) Your SQL query runs continuously on the incoming data.
5. Define an output (destination): another KDS, Firehose, or Lambda.
6. The service continuously reads from the source, processes data, and sends results to the destination.

When to Use Kinesis Data Analytics:

- You need real-time analytics (e.g., aggregations, filtering, time-series analysis) on streaming data.
- You prefer using SQL for its simplicity for common analytics tasks.
- You need the power of Apache Flink for complex stream processing without managing Flink clusters yourself.
- Real-time dashboards, alerting, generating metrics for leaderboards.

AWS Kinesis Video Streams

Kinesis Video Streams makes it easy to securely stream video from millions of connected devices to AWS for analytics, machine learning, playback, and other processing.

It can ingest data from cameras, smartphones, security cameras, and other video sources.

How it Works:

- **Producers:** Devices (e.g., cameras with the Kinesis Video Streams Producer SDK) or applications that send video data to a Kinesis video stream.
- **Video Stream:** A resource that transports and stores video data. Each stream can contain multiple fragments.
- **Fragment:** A self-contained sequence of media data, typically a few seconds long.
- **Consumers:** Applications that retrieve and process the video data. This can include:
  - Live or on-demand playback using HLS or DASH.
  - Integration with Amazon Rekognition Video for real-time video analysis (e.g., object detection, face recognition).
  - Custom ML models for video analytics.
  - Saving processed video or clips to S3.
- **WebRTC:** Kinesis Video Streams also provides fully managed WebRTC capabilities for real-time, two-way media streaming between web browsers, mobile applications, and connected devices.

Use Cases:

- Live video streaming for events or monitoring.
- Smart home camera solutions.
- Industrial monitoring with video analytics.
- Dashcam video storage and analysis.
- Real-time baby monitors.
- Two-way video conferencing using WebRTC.

Comparison Table

Feature	Data Streams	Firehose	Analytics
Use Case	Custom processing	Easy delivery	Real-time analytics
Scalability	Manual shard management	Auto-scaling	Scales with input
Processing	Application/consumer	No processing	SQL/Flink
Latency	Low (ms)	Higher (seconds)	Low
Destination	Custom	S3, Redshift, etc.	KDS, KDF

Real-World Applications

E-Commerce:
- Real-time inventory tracking
- Customer behavior analysis
- Dynamic pricing strategies
- Fraud detection

Media & Entertainment:
- Viewer engagement metrics
- Content recommendation
- Ad performance tracking
- Live event monitoring

IoT & Manufacturing:
- Sensor data processing
- Predictive maintenance
- Quality control
- Supply chain optimization

Financial Services:
- Transaction monitoring
- Risk assessment
- Compliance reporting
- Market analysis

Amazon OpenSearch Service

Amazon OpenSearch Service

What is OpenSearch?

- A **managed service** that makes it easy to deploy, operate, and scale OpenSearch and legacy Elasticsearch clusters.
- Designed for **search**, **log analytics**, **monitoring**, **observability**, and **application performance monitoring (APM)**.
- Open-source search and analytics engine, Fork of Elasticsearch
- Distributed, RESTful search service

Key Capabilities:

- **Full-text search**: Complex queries on text data
- **Analytics**: Aggregations and visualizations
- **Log analytics**: Real-time log analysis
- **Application monitoring**: Performance insights

OpenSearch Architecture

- **Domain**: A managed OpenSearch cluster.
- **Node:** A server within the domain
- **Index**: Logical namespace for documents.
- **Document**: JSON-formatted data stored in the index.
- **Shard**: Subdivision of an index, improves scalability and parallelism.
- **Replica**: A copy of a shard for fault tolerance.

Data Ingestion Options

- **Amazon Kinesis Data Firehose** (Direct PUT or Kinesis Data Streams)
- **Logstash**: For complex ingestion pipelines
- **Beats/Fluent Bit**: Lightweight log shippers
- **Lambda / SDKs**: Custom ingestion logic
- **CloudWatch Logs Subscriptions**

Indexing and Querying

- **Indexing**: Send data to OpenSearch using HTTP PUT/POST
- **Querying**: Use **DSL (Domain Specific Language)** or **Lucene syntax**
- Queries can be:
  - Full-text (match, match_phrase)
  - Structured (term, range, bool)
  - Aggregations for dashboards

OpenSearch Dashboards

- Visualization layer for OpenSearch (formerly Kibana)
- Create:
  - Line charts
  - Bar/pie charts
  - Maps (if geo-data present)
  - Real-time dashboards
- Use “Discover” to explore raw data
- Use “Visualize” and “Dashboard” for monitoring and insights

Integration with Other AWS Services

- **Amazon Kinesis Firehose**: Real-time streaming to OpenSearch
- **Amazon CloudWatch Logs**: Log subscriptions into OpenSearch
- **AWS Lambda**: Custom processing and ingestion
- **IAM**: Secure access management
- **Cognito**: Dashboard user management

Real-World Example Architecture

- **Log Analytics Pipeline:**

1. Application logs sent to CloudWatch
2. Logs streamed via Lambda to Kinesis Firehose
3. Firehose delivers data to OpenSearch
4. Dashboards used to monitor and analyze logs in real-time

Summary

- OpenSearch Service enables real-time search, analytics, and observability at scale.
- Offers flexible ingestion, powerful querying, secure access, and rich visualizations.
- Integrates easily with AWS services to form complete observability pipelines.

Amazon CloudWatch

What is CloudWatch?

- **Monitoring service**: Collects logs, metrics, and events from AWS resources and applications.
- **Firehose Integration**: CloudWatch logs can be streamed into Kinesis Data Firehose for long-term storage and analysis.
- Monitoring and observability service
- Collects metrics, logs, and events
- Provides insights into AWS resources

Key Components:

1. **Metrics**: Numerical data points over time
2. **Logs**: Text-based application/system logs
3. **Events**: System state changes
4. **Alarms**: Notifications based on thresholds

Amazon CloudWatch

Integration with Streaming:

- Monitors Kinesis Data Firehose metrics
- Captures Lambda function logs
- Tracks OpenSearch performance
- Provides debugging capabilities

Benefits:

- Unified monitoring platform
- Real-time insights
- Automated actions
- Historical data retention

BUSINESS INTELLIGENCE AND DATA ANALYTICS
– WE PREDICT THE FUTURE –

Institute of Analytics
Recognized Programme

University of Petra

LAB

Analyzing and Visualizing Streaming Data with Kinesis Data Firehose, OpenSearch Service, and OpenSearch Dashboards

In this lab, you will:

- Set up an Amazon Kinesis Data Firehose delivery stream.
- Configure the stream to send data to Amazon OpenSearch Service.
- Simulate streaming data to the Firehose delivery stream.
- Visualize the data using OpenSearch Dashboards.

Prerequisites

- AWS account (or AWS Academy lab environment)
- IAM permissions to access Kinesis, OpenSearch, S3, CloudWatch
- Basic understanding of streaming data concepts
- Familiarity with AWS Management Console

Services Overview

- **Amazon Kinesis Data Firehose**
- Fully managed service for delivering real-time streaming data
- Automatically batches, compresses, encrypts, and loads data
- **Amazon OpenSearch Service**
- Search and analytics engine
- Stores and indexes data for fast search and visualization
- **OpenSearch Dashboards**
- Visualization tool for OpenSearch
- Allows creation of dashboards, charts, and graphs

Lab Architecture

1. Data is streamed via Kinesis Data Firehose
2. Firehose sends data to OpenSearch Service
3. OpenSearch indexes the data
4. Dashboards used to visualize insights

Step 1 - Set Up IAM Role

- Create a new IAM role for Kinesis Data Firehose
- Attach policy to allow access to:
    - Amazon OpenSearch Service
    - Amazon S3 (backup)
    - CloudWatch Logs

Step 2 - Create a Kinesis Data Firehose Delivery Stream

- Choose source: Direct PUT
- Destination: Amazon OpenSearch Service
- Backup: Amazon S3
- Transformations: Disabled
- Error logging: Enable CloudWatch

Step 3 - Create OpenSearch Domain

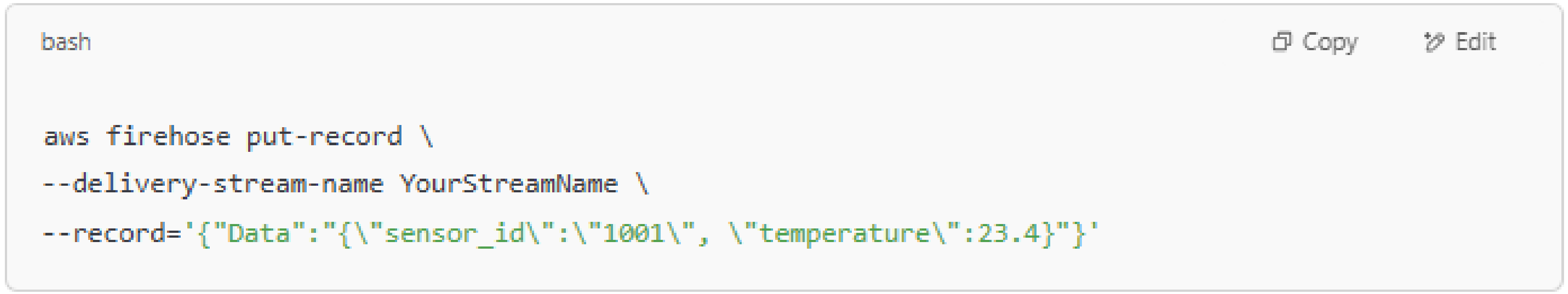
- Select development and testing configuration
- Choose instance type and number
- Enable fine-grained access control
- Set up OpenSearch Dashboards

Step 4 - Update IAM Policy for OpenSearch Access

- Allow Firehose access to OpenSearch domain
- Add trust policy to IAM role if needed

Step 5 - Simulate Streaming Data

- Use AWS CLI or Python script to send JSON records to Firehose:


bash Copy Edit

```bash
aws firehose put-record \
--delivery-stream-name YourStreamName \
--record='{"Data":"{\"sensor_id\":\"1001\", \"temperature\":23.4}"}'
```

Step 6 - Verify Delivery

- Go to OpenSearch Dashboards
- Navigate to “Discover”
- Check if your index is receiving data

Step 7 - Visualize Data

- Create index pattern in OpenSearch Dashboards
- Use “Visualize” to create charts (e.g., bar, pie, line)
- Combine visuals into dashboards

Cleanup

- Delete:
  - Delivery stream
  - OpenSearch domain
  - IAM roles and policies
  - S3 backup bucket (optional)

Summary

- You have:
  - Created a streaming pipeline with Kinesis Firehose
  - Delivered data to OpenSearch
  - Visualized data with OpenSearch Dashboards
- This architecture supports real-time analytics and monitoring