MODULE 15 · STUDY NOTES

Data Engineering Patterns

How data characteristics shape a pipeline, and the AWS services that ingest, store, process, and analyze data — from batch ETL with AWS Glue to real-time streaming with Amazon Kinesis, data lakes, and analytics

A data engineering pipeline turns raw, scattered data into trustworthy business insight, and a cloud architect's job is to choose components that fit both the data and the outcome. This module works backward from that outcome. It starts with the **five Vs** — value, veracity, volume, velocity, and variety — the characteristics that shape every infrastructure decision, and the *modernize, unify, innovate* strategy that leads to a **modern data architecture** with a **data lake** at its center. It then walks the four pipeline layers — **ingest, store, process, analyze**, with storage woven through all of them — covering the ingestion patterns (homogeneous vs. heterogeneous ETL/ELT, batch vs. streaming) and the purpose-built AWS services for each layer: AppFlow, DataSync, and Data Exchange to ingest; AWS Glue for batch ETL; the Amazon Kinesis family for real-time streams; S3, Redshift, Aurora, and DynamoDB to store; Amazon EMR for parallel big-data processing; and Athena, QuickSight, and OpenSearch Service to analyze and visualize. It closes by mapping these choices onto the AWS Well-Architected Framework's **Data Analytics Lens**, because the right architecture is never fixed — it is continually re-evaluated as data, cost, and requirements evolve.

LEARNING OBJECTIVES

- Use the AWS Well-Architected Framework to generalize the type of architecture required for common data-ingestion use cases (batch and stream)
- Select a data ingestion pattern appropriate to the characteristics of the data (velocity, volume, and variety)
- Select the appropriate AWS services to ingest and store data for a given use case
- Select the appropriate AWS services to optimize data processing and transformation requirements for a given use case
- Identify when to use different types of AWS data analytics and visualization services based on a given use case

15.1 Data characteristics

Approach every design as a cloud architect: **start with the business case and work backward** to choose the best components — and expect those decisions to evolve as needs are better understood and as you optimize cost and performance. The point is not to collect data for its own sake but to store, organize, and access it so that it earns business value. Five characteristics of the data drive those choices.

TABLE 15.1 The five Vs of data

CHARACTERISTIC	SCOPE	QUESTIONS TO CONSIDER
Value	How processed data provides insight into a business problem	What insights can be gained from the data?
Veracity	How to protect and strengthen the integrity of the data	How accurate, precise, and trusted is the data?
Volume	How much data you need to process	How long must you keep it? What are the access patterns?
Velocity	How quickly data enters and moves through the pipeline	How often is data generated? How fast must it be acted on?
Variety	How many data sources and data types you work with	What format and type is the data? From what sources?

The Vs are drawn in a circle, not a line, because each influences the others and decisions are not made linearly. **Value depends on veracity** — without good data you make bad business decisions. **Volume and velocity together** drive the expected throughput and scaling of the pipeline; for equally high-volume use cases, how fast data arrives and must be processed decides the infrastructure (can the pipeline absorb sudden bursts, such as gaming data at peak times?). **Variety** — structured, semistructured, or unstructured — dictates how data enters the pipeline and how it must be prepared, and combining datasets can enrich analysis while complicating processing. Throughout, keep the **end user** in mind.

> **THE THREE-PRONGED STRATEGY: MODERNIZE, UNIFY, INNOVATE**
>
> - **Modernize** — move to cloud-based infrastructure and purpose-built services to reduce undifferentiated lifting, increase agility, and cut operational effort.
> - **Unify** — create a single source of truth and make data available across the organization, combining the best of data lakes and purpose-built data stores.
> - **Innovate** — apply artificial intelligence and machine learning (AI/ML) to find new insights, reimagine old processes, and create new experiences.

Organizations want to capture all their data and derive value fast, but shuffling data among a data lake and separate stores gets **complex and messy** as it grows. The **modern data architecture** solves this: store data in a **centralized location** and make it available to every consumer for analytics and AI/ML applications. It **integrates a data lake, a data warehouse, and other purpose-built data stores** while enabling **unified governance and seamless data movement** — it is not restricted by separate silos — so the organization makes better decisions with agility.

> **COMMON PITFALL**
>
> Data-infrastructure decisions are **not linear**. Evaluate the five Vs *together* for each use case, and revisit them over time — the right architecture evolves as data volumes grow, requirements sharpen, and cost and performance are optimized.

15.2 Data pipelines

At the most basic level, any pipeline must **bring data in, store it, and provide the means to work with it** to derive insight. The four elements are **ingestion, storage, processing, and analysis**, and storage spans all of them. Real

pipelines are **iterative** — a linear pipeline is just a simplified view — so start with the end in mind (the business problem) and let the data's characteristics shape the pipeline. **Ingestion** is extracting data from its source and loading it into the pipeline to be stored or analyzed — copying data from a source data store to a target data store.

When data is ingested **as-is**, the pattern is **homogeneous**: move data from a source store to a destination store while **keeping the same format or storage-engine type** — for example, migrating an on-premises NoSQL database into an Amazon RDS for MySQL database with no transformation, or landing raw text files in an area that preserves the original copies. Because raw data is often inconsistent, imprecise, and repetitive, data **will almost always be transformed** somewhere in the pipeline to match the schema of its target. When transformation is needed, ingestion is **heterogeneous** — an ETL or an ELT pattern.

ETL – EXTRACT, TRANSFORM, LOAD	ELT – EXTRACT, LOAD, TRANSFORM
Structured data → data warehouse	*Unstructured data → data lake*
▪ Extract structured data ▪ **Transform** it to match the destination schema ▪ **Load** into structured storage for defined analytics ▪ Stores data ready to analyze — saves the analyst time	▪ Extract unstructured (or structured) data ▪ **Load** into the destination close to raw form ▪ **Transform** as needed for each analytics scenario ▪ More raw data on hand — flexibility to create new queries

Ingestion and processing also split along **batch vs. streaming**. **Batch** runs every command on the *entire batch* of data — on demand, on a schedule, or on an event — and computes results over **complete datasets**, supporting deep analysis of large datasets. **Streaming** handles an **unbounded** stream: a continuous, incremental sequence of small data packets generated by thousands of sources at once, processed as a series of events for **real-time analytics**, with metrics and reports updated incrementally.

TABLE 15.2 Batch vs. streaming processing

FEATURE	BATCH PROCESSING	STREAMING PROCESSING
Processing cycles	Run infrequently, typically during off-peak hours	Run continuously
Compute	Requires high computing power	Low compute plus a reliable, low-latency network
Use case	Sales transactions analyzed overnight; reports sent to branches by morning	Product recommendation — data must be analyzed immediately

> **COMMON PITFALL**
>
> If data must be analyzed in **near real time**, choose **streaming** ingestion, not batch. And remember the ingestion rule: homogeneous ingestion needs **no transformation** (source and destination match); only heterogeneous ingestion transforms, before load (**ETL**) or after load (**ELT**).

15.3 AWS tools to ingest data

The ingestion layer uses **purpose-built services organized by source type**, each reducing undifferentiated lifting. Match the tool to where the data lives — SaaS applications, file shares, or third-party data providers.

TABLE 15.3 Purpose-built ingestion services

SERVICE	INGESTS FROM	KEY TRAITS
Amazon AppFlow	SaaS apps (Salesforce, SAP, Google Analytics, Facebook Ads, Zendesk, ServiceNow)	Transfers data between SaaS and AWS (S3, Redshift); reuses integrations via APIs; transforms (filter, mask, validate, partition, aggregate, catalog)
AWS DataSync	File shares; on-premises data centers	Fully managed migration; copies file/object data to and from S3, EFS, FSx; optimized for speed; encryption + integrity validation; preserves metadata; scheduled hourly/daily/weekly
AWS Data Exchange	Third-party data providers	Find, subscribe to, and use third-party data via files, tables, and APIs; comprehensive catalog; use licensed data in production without building pipelines

Amazon AppFlow ends the months-long grind of building and managing SaaS connectors — you integrate applications immediately. **AWS DataSync** moves large volumes to and from on-premises centers and between AWS storage services, arriving securely, intact, and ready to use. **AWS Data Exchange** bridges providers and subscribers: pharmaceutical companies license life-expectancy benchmarks, retailers subscribe to weather data for inventory planning, and restaurants use location data to choose where to expand.

> WORKED EXAMPLE – ON-PREMISES RESEARCH DATA WITH DATASYNC
>
> Researchers keep data on an on-premises server and need it in AWS for archiving and analytics, synchronized daily. A **DataSync agent** (an AWS-managed extension of the service) reads the source storage after a one-time setup; you then create as many transfer tasks as needed to any Amazon S3 storage class, Amazon FSx, or Amazon EFS — and can use DataSync again to move data among those AWS services later.

15.4 Processing batch data with AWS Glue

Batch processing periodically completes high-volume, repetitive jobs — think of a banking system that receives transactions all day but **collects and processes them at the end of each day** to send reports to stakeholders and third-party systems. Batch makes repetitive tasks efficient and uses compute cost-effectively: compute-intensive work such as backing up, filtering, and sorting runs in batches during **off-peak times** (end of day, overnight) rather than on individual records.

> WHEN TO CONSIDER BATCH PROCESSING
>
> - For **reporting** purposes (daily and weekly).
> - When dealing with **large datasets**.
> - When the use case emphasizes **aggregating or transforming** data, rather than real-time analysis.
> - Example domains: medical research — computational chemistry, clinical modeling, molecular dynamics, genomic sequencing — where no real-time analysis is needed.

AWS Glue is a serverless **data-integration service** that automates and performs **ETL** tasks as part of ingesting data into a batch *or* streaming pipeline. It reads and writes across many systems and databases — integrating with **Amazon S3, DynamoDB, Redshift, RDS, and DocumentDB** — and simplifies both batch and streaming ingestion. Treat it as a multi-purpose data-integration toolkit.

TABLE 15.4 AWS Glue components

COMPONENT	PURPOSE
ETL jobs	Author, run, and monitor data-transformation jobs — visually in AWS Glue Studio (no code) or in Jupyter-style job notebooks; Glue Streaming ETL speeds availability of stream data
Data Catalog	Data discovery and organization — stores metadata and schema for sources and ETL targets (table definitions, physical location, business attributes, change history); stores no datasets
Glue DataBrew	No-code visual data preparation — clean and normalize data with 250+ prebuilt transformations (filter anomalies, standardize formats, correct invalid values)
Glue Data Quality	Assesses a dataset, computes statistics, recommends quality rules, monitors, and alerts when quality deteriorates — catches missing, stale, or bad data before it hurts decisions

Glue crawlers run on your data stores (Amazon S3, relational databases such as RDS), **derive a schema**, and populate the Data Catalog by creating or updating catalog tables. Once structured or semistructured data has a schema in the catalog, it is available for ETL **and** immediately queryable in **Amazon Athena, Amazon EMR, and Amazon Redshift Spectrum**, giving every Glue user a common view of the data. While cataloging, Glue also flags data-type discrepancies and provides notifications.

> WORKED EXAMPLE – A BATCH ETL PIPELINE FOR A GAME
>
> A gaming company generates a few gigabytes of player data daily (average session length, in-game purchases, days since last played) and wants to predict session length. The game server pushes data to an **S3 bucket every 6 hours**; **Glue crawlers** run on the player logs and hand Glue ETL a Data Catalog; each 6-hour load triggers a **Glue ETL job** that aggregates log data per player into **1-minute intervals**; the transformed data lands in an aggregated S3 bucket for multiple analytics applications to access.

Glue is also a **data-transformation** service. A common job converts CSV to Apache Parquet (`.csv` → `.parquet`): CSV is the most common tabular format but is inefficient to store or manipulate beyond about 15 GB, whereas **Parquet** stores data **columnar**, compresses and encodes efficiently, and is **splittable for parallel processing** — speeding analytics and, over time, saving storage space, cost, and time. Glue and DataBrew can also perform advanced transformations such as **PII detection and removal**: scan the data, detect entities (a passport or Social Security number), then remediate by masking the data or storing the detection result for further inspection.

> COMMON PITFALL
>
> Use **AWS Glue when the analytics use case does *not* require real-time aggregation or transformation.** Glue covers schema identification, cataloging, preparation and cleaning, ETL authoring, and data-quality assessment — the batch story. Real-time needs point to the Amazon Kinesis family instead.

15.5 Processing real-time data

Streaming data is emitted at high volume in a continuous, incremental manner with the goal of **low-latency processing**. An online gaming company, for instance, collects streaming data about player interactions, analyzes it in real time, and offers incentives and dynamic experiences to keep players engaged. Streaming suits any scenario with

new, dynamic data generated continually — companies often begin simple (collecting system logs, rolling min-max computations) and evolve toward sophisticated near-real-time processing, such as tracking brand sentiment across social media streams and responding in a timely fashion.

ELEMENTS OF A STREAMING PIPELINE

- **Sources** — streaming data can come from many origins at once.
- **Producers** collect data from sources and deliver it into the stream service.
- **Stream** — the mechanism that delivers producer-created records to be read by consumers.
- **Consumers** read data from streaming storage and process it.
- **Destinations** — the processed data can go to one or many targets.

TABLE 15.5 AWS streaming services

SERVICE	ROLE
Amazon Data Firehose	Captures and transforms streaming data in near real time; delivers to storage/analytics destinations via micro-batches; no code required
Amazon Kinesis Data Streams	Collects and ingests large streams of log and event records in real time; temporary storage with replay
Amazon Managed Service for Apache Flink	Serverless; queries and analyzes streaming data in real time (for example, aggregation or anomaly detection)
Amazon Kinesis Video Streams	Securely streams video from connected devices to AWS for analytics, ML, and playback
Amazon MSK	Fully managed Apache Kafka, deployable in a VPC

Amazon Data Firehose captures and transforms streaming data in near real time and delivers it to common destinations — **Amazon S3, Redshift, OpenSearch Service**, or any supported HTTP endpoint — with **no coding**. It delivers via **micro-batch cycles**: the service waits a short window (milliseconds up to several seconds) before writing a batch, balancing latency and throughput — quicker than batch, not as fast as pure streaming — and loads new data to the destination **within 60 seconds**. Firehose can also convert input from **JSON to Apache Parquet** (or ORC) before storing, since these columnar formats save space and query faster than row-oriented JSON.

For faster processing, **Amazon Kinesis Data Streams** ingests stream log and event data, provides **temporary storage (up to 365 days)**, delivers in real time within 60 seconds, and lets **multiple consumers replay** the same stream simultaneously. **Producers** write to the stream via the AWS SDK, the **Amazon Kinesis Agent**, or the **Kinesis Producer Library (KPL)**, which simplifies retry, batching, and write optimization. **Consumers** read with the **Kinesis Client Library (KCL)** — which handles distributed processing (load-balancing record processing across instances, checkpointing processed records, and handling instance failures) — or with the Kinesis SDKs for finer control.

Amazon Managed Service for Apache Flink is serverless and builds end-to-end stream-processing applications, analyzing data with **SQL or Apache Flink** to deliver insights in seconds or minutes; common uses are streaming ETL, continuous metric generation, responsive real-time analytics, and interactive querying of data streams, and it scales automatically. **Amazon Kinesis Video Streams** securely streams video from millions of connected devices (smartphones, security cameras, dashcams, drones), elastically provisioning the ingest infrastructure and durably storing, encrypting, and indexing the video. **Amazon MSK** is fully managed Apache Kafka that runs your existing Kafka

apps and tools without code changes — ideal to reduce licensing costs if you already know Kafka, though it needs more setup and engineering than the Kinesis services.

> **WORKED EXAMPLE – CAFÉ CLICKSTREAM WITH FIREHOSE**
>
> The café owners want to see when users browse and click the online menu, and they do **not** need subsecond delivery — so **Kinesis Data Firehose** (the simplest service, with connectors for the source and destination) fits. A static website on **Amazon S3** serves the menu → the browser sends clickstream data to **Amazon API Gateway** → **Firehose** ingests it → the data lands in an **S3 bucket**, where analysis and visualization tools read it.

> **WORKED EXAMPLE – MEDICAL-DEVICE MONITORING WITH KINESIS DATA STREAMS + FLINK**
>
> Medical devices monitor patients, and any fluctuation must be reported in **milliseconds** — so timing-critical ingestion uses **Kinesis Data Streams**. KDS ingests the IoT sensor data → **Amazon Managed Service for Apache Flink** performs **anomaly detection** → on an anomaly, an **AWS Lambda** function fires → Lambda sends a mobile notification to the medical professional to evaluate the device.

TABLE 15.6 Choosing a streaming ingestion service

SERVICE	CHOOSE WHEN	RETENTION
Firehose	Simplest option feeding a Firehose-approved destination; plug-and-play AWS integration	Undelivered data max 24 h; no store or replay of delivered data
Kinesis Data Streams	You can't use a Firehose destination or need more control; requires code customization (KPL/KCL)	Up to 365 days for replay (24 h by default)
Amazon MSK	Open-source Kafka to cut licensing; already familiar with Kafka; manage more infrastructure	Longer, configurable retention

> **COMMON PITFALL**
>
> The tie-breakers among streaming services are **business use case, engineering effort, and retention period.** Firehose is least complex; **Amazon MSK is the most complex.** Only **Kinesis Data Streams** offers long replay (up to 365 days); Firehose does not store delivered data at all.

15.6 Storage in the data pipeline

Throughout the pipeline, data is stored securely, and in a modern data architecture the **data lake** sits at the center — made of **data storage, a data catalog, and security access**. A data lake is a **centralized repository for all structured and unstructured data at any scale**, stored **as-is** in open, standards-based formats at low cost — you need not structure the data first. Around it sit relational databases, nonrelational (NoSQL) databases, big-data processing, machine learning, log analytics, and a data warehouse, all able to exchange data.

DATA MOVEMENT IN AND OUT OF THE LAKE

- **Inside-out** — a service (big-data processing, ML) needs part of the lake's data; it is copied, moved, or query-filtered *from* the lake *to* the service (e.g., clickstream collected in the lake, a portion sent to a log-analytics service for trend analysis).
- **Outside-in** — an analytics solution needs the data *in* the lake (to form part of a bigger dataset); it is copied, moved, or filtered *to* the lake from a database source.
- **Direct** — if two services integrate directly, move, copy, or query data between them without routing through the lake.

TABLE 15.7 AWS data lake building blocks

LAYER	AWS SERVICE	ROLE
Data storage	Amazon S3	Low-cost, scalable object storage at the core of the lake
Catalog & transformation	AWS Glue	Data Catalog holds the schemas used to read the lake; Glue jobs transform data
Management & security	AWS Lake Formation	Central console to discover sources, manage the lake, and configure IAM access permissions
Access engine	Amazon Athena	SQL query engine that lets analytics and visualization tools query the lake

Other services round out the architecture: **Amazon Aurora and RDS** store relational, normalized data; **DynamoDB** stores key-value NoSQL data; **Amazon EMR** transforms or aggregates big datasets in parallel; **Amazon SageMaker** trains models to recognize patterns; **OpenSearch Service** indexes data for fast retrieval; and **Amazon Redshift**, the data warehouse, stores historical data and splits a SQL query to run in parallel across datasets spanning decades.

TABLE 15.8 Data warehouse vs. data lake

FEATURE	DATA WAREHOUSE	DATA LAKE
Data sources	Business applications and databases	IoT devices, websites, mobile apps, social media, business apps
Schema	Structured — schema-on-write	Unstructured — schema-on-read
Price	Higher-cost storage	Low-cost storage
Data quality	Curated/processed/aggregated — single source of truth	Raw transactional events, or transformed data
Analytics use case	Batch reporting, business intelligence, visualizations	Logs, data discovery, and profiling

WORKED EXAMPLE – STORAGE FOR A BANK APPLICATION

Match each data type to a store. **Individual customer transactions** are **OLTP** — small record writes, 24/7 availability, massive concurrency — so a relational or NoSQL store fits: **Aurora or DynamoDB**. The **daily transaction total per customer** is **OLAP** — computed and kept in a data warehouse for multi-year spending-pattern reports in columnar tables — so **Amazon Redshift**. **User activity logs for fraud analysis** go to the data lake (**Amazon S3**) or, for an indexed solution, **OpenSearch Service**. **Application error logs** take the simplest path — data-lake storage on **Amazon S3**.

WHAT DETERMINES WHERE DATA IS STORED

- **Cost vs. query time** — Amazon S3 is lowest-cost but less query-efficient; Redshift costs more (a cluster of instances) but queries large datasets efficiently; **Redshift Spectrum** queries S3 directly **without moving data** into the warehouse.
- **Query scope** — partition S3 (e.g., daily logs in dated folders) so a date-filtered query touches only the relevant folders.
- **Retention** — S3 storage classes match access patterns: S3 Standard (general purpose), Intelligent-Tiering (unknown/changing access), Standard-IA and One Zone-IA (infrequent access), and Glacier Instant Retrieval / Flexible Retrieval / Deep Archive (archive).
- **Origin & shape** — OLTP business apps → RDMS (Aurora) for structured, NoSQL (DynamoDB) for semistructured, streams (Kinesis / S3 / MSK) for real time; OLAP analytics → data lake (S3) for raw/unstructured, data warehouse (Redshift) for processed/aggregated.

COMMON PITFALL

Schema-on-write is the data warehouse; schema-on-read is the data lake. A warehouse needs its schema defined *before* data is written (a curated single source of truth); a lake accepts data as-is and applies a schema only when the data is read. Choose storage by data **origin, shape, cost, query scope, and retention period.**

15.7 Parallel processing in the data pipeline

Parallel processing breaks a large dataset into smaller parts, processes each part **concurrently**, and **aggregates** the results into the final output. It exists because a single server lacks the memory and storage to handle a huge dataset; a cluster of hundreds or thousands of servers does the work instead. Counting every word across a million-book library, for example, splits into 10 batches of 100,000 books whose per-word counts are then collated into one result list. The payoff is time: processing 1,000 one-minute log files sequentially takes over 16 hours, but in parallel it takes about 1 minute.

Amazon EMR manages the cluster infrastructure — provisioning, setup, configuration, and tuning — and bundles Apache Hadoop applications including **Hadoop MapReduce** and **Apache Spark**. A cluster can run on **Amazon EC2** (a VPC subnet in a single Availability Zone), pass work to an **Amazon EKS** cluster for Multi-AZ, or run on **AWS Outposts** on-premises; **Amazon EMR Serverless** removes cluster management entirely.

TABLE 15.9 Choosing a parallel-processing solution

REQUIREMENT	AMAZON EMR	EMR SERVERLESS	AWS GLUE
Manage and control clusters	Yes	No	No
Lift and shift legacy Hadoop/Spark to AWS	Yes	No	No
Develop new cloud apps for batch processing	No	Yes	Yes
Has a pay-per-job price model	No	Yes	Yes
Run only Apache Spark jobs	No	No	Yes

An EMR cluster is a set of EC2 **nodes** by role: the **primary node** coordinates data and task distribution and tracks cluster health; **core nodes** run tasks *and* store data in HDFS; optional **task nodes** only run tasks. Clusters can be **long-running or transient** (shut down after processing to save cost — so persist results to external storage first, since cluster storage is terminated too). Data reaches EMR via **HDFS** or the **EMR File System (EMRFS)**, which reads and writes directly to Amazon S3; a processing **step** names the framework (MapReduce or Spark) and a Python, Scala, or PySpark script, and results are exported to destinations such as S3 or Redshift. A common curation flow uses **three data-lake zones**: a drop zone (secured by Lake Formation), an EMR **cleaning** job into an analytics zone, and an EMR **curation** job into a curated zone.

> **COMMON PITFALL**
>
> Match the parallel-processing tool to the team: **Amazon EMR** to manage clusters or lift-and-shift legacy Hadoop/Spark; **EMR Serverless** for a serverless, pay-per-job model when the team has some Hadoop/Spark experience; **AWS Glue** when the team is new to analytics or wants to run **only Apache Spark** jobs.

15.8 Analysis and visualization

Once data is stored and curated, the **consumption tool** depends on who needs it and why — and every choice should apply the **principle of least privilege**, granting only the access a user or system needs to do its job (a business analyst reading one Redshift table gets read permission on that table only).

TABLE 15.10 Matching consumption tools to the job

PERSONA / NEED	AWS SERVICE
Business analyst — interactive dashboard of charts and graphs to share with stakeholders	Amazon QuickSight
Data scientist / engineer — one-off SQL queries over customer-activity files in a data lake	Amazon Athena SQL
DevOps engineer — real-time application-monitoring dashboard	Amazon OpenSearch Service

Amazon QuickSight is a **business intelligence (BI)** tool that scales to hundreds of thousands of users, delivering fast, responsive visualizations from a robust in-memory engine called **SPICE**. A dashboard is a collection of charts, graphs, and insights that you publish, share by link, embed in a web or mobile application, or email on a schedule (Enterprise edition). It connects to RDS, Aurora, Redshift, Athena, and S3, to uploaded Excel or flat files, to on-premises databases, and to SaaS sources; **AutoGraph** picks the right chart type automatically, and **QuickSight Q** answers natural-language questions as visualizations.

Amazon Athena SQL is a **serverless** query engine and schema-metadata store that runs **standard SQL directly against data in Amazon S3** — structured or unstructured — with no infrastructure to manage and a **pay-per-query** model suited to one-off queries. Built on **Trino and Apache Presto**, it reads CSV, JSON, ORC, Parquet, and Avro; integrates with the Glue Data Catalog; and connects to sources from RDS, DynamoDB, MSK, OpenSearch, and Redshift to on-premises SAP HANA and Db2 and even other clouds' data lakes — enabling **federated queries**. **Athena for Apache Spark** additionally runs big-data parallel workloads, and BI tools connect over JDBC/ODBC drivers.

Amazon OpenSearch Service is a managed, serverless service for interactive **log analytics, real-time application monitoring, and website search**. It **indexes** uploaded data in a domain for fast search and analysis and ships **OpenSearch Dashboards** with every domain. It adds **alerting and ML-based anomaly detection** on streaming data (be notified the moment an outlier appears), and a **Multi-AZ with Standby** deployment gives business-critical workloads high availability and resilience to node or single-AZ failures.

> **WORKED EXAMPLE – VISUALIZING CAFÉ CLICKSTREAM**
>
> The café owners want a shareable clickstream dashboard. A data analyst uses **QuickSight** to set security permissions for **Athena** and the café clickstream **S3** bucket as data sources and builds the dashboard; via the Data Catalog, **Athena** saves the QuickSight-generated queries in a query-result bucket; the analyst **publishes** the dashboard and sends a **URL link**, which the owners open to view it.

> **COMMON PITFALL**
>
> Sample-exam pattern: to **analyze *and* visualize real-time streaming logs with a single service**, choose **Amazon OpenSearch Service**. Athena queries but does not visualize; QuickSight visualizes but needs a separate ingestion service; Amazon Redshift is a data warehouse, not a real-time tool.

15.9 Applying the AWS Well-Architected Framework to data pipelines

The **AWS Well-Architected Data Analytics Lens** collects customer-proven best practices for analytics workloads, distilled from real-world case studies so architects and developers can apply them without becoming subject-matter experts. The review process is continual: **define the workload** (a set of components that together deliver business value), **evaluate it against the pillar design principles** (prioritizing the pillars in order of importance for your case and picking the most important principles per pillar), and **implement best practices**, re-evaluating regularly to add as many as possible over time.

TABLE 15.11 Data Analytics Lens best practices (four emphasized pillars)

PILLAR	BEST-PRACTICE APPROACH
Security	Control access to the workload infrastructure — implement policies of least privilege for source and downstream systems
Performance efficiency	Choose the best-performing compute — identify the analytics service built for the challenge (Redshift for warehousing, Kinesis for streaming, QuickSight for visualization)
Cost optimization	Manage cost over time — remove unused data and infrastructure (use S3 lifecycle configurations to expire data past retention) and reduce overprovisioning (move cold data to the lake, query in place with Redshift Spectrum or Athena)
Reliability	Design resilience for analytics workloads — understand the business requirements of analytics and ETL jobs and the data-movement patterns behind them

For the **reliability** pillar, three patterns move data from source systems to a target store: **ETL** transforms *before* loading; **ELT** loads into the central repository (data lake or data warehouse) and transforms later; and **ETLT**, a hybrid, transforms just enough to meet entry-quality criteria, loads the data, then transforms it further when needed.

> **COMMON PITFALL**
>
> Only **four** of the six Well-Architected pillars are emphasized here — **security, performance efficiency, cost optimization, and reliability** — and the listed best practices are **not exhaustive**. The lens is a *continual* process: prioritize the pillars for your use case and keep re-evaluating as data, cost, and requirements change.

Module summary

- Judge data by the five Vs — value, veracity, volume, velocity, variety — considered together; value and veracity govern insight quality, and volume and velocity together drive pipeline throughput and scaling.
- The modernize-unify-innovate strategy leads to a modern data architecture: a central data lake integrated with a data warehouse and purpose-built stores, with unified governance and seamless data movement — no silos.
- A pipeline has four layers — ingest, store, process, analyze — with storage throughout; ingestion copies source to target, either homogeneous (as-is, no transform) or heterogeneous (ETL transforms before load for structured data → warehouse; ELT transforms after load for unstructured data → lake).
- Batch computes over complete datasets (high compute, off-peak, reporting); streaming processes an unbounded sequence of small packets continuously (low compute, low-latency network, real-time).
- Purpose-built ingestion services: Amazon AppFlow (SaaS apps), AWS DataSync (file shares and on-premises), AWS Data Exchange (third-party data).
- AWS Glue is the serverless batch-ETL engine — Data Catalog (metadata only), crawlers (derive schema), DataBrew (no-code prep), Data Quality — and can convert CSV to Parquet and detect/remove PII.
- The Amazon Kinesis family handles streams: Data Firehose (simplest, micro-batch, no replay), Kinesis Data Streams (low latency, replay up to 365 days, KPL/KCL), Managed Service for Apache Flink (real-time analytics/anomaly detection), and Kinesis Video Streams — plus Amazon MSK (managed Kafka).
- On AWS the data lake = S3 (storage) + Lake Formation (management/security/IAM) + Glue (catalog/transform) + Athena (SQL access); schema-on-write is the warehouse, schema-on-read is the lake; choose storage by origin, shape, cost, query scope, and retention (S3 storage classes).
- Parallel processing splits, processes concurrently, then aggregates big data; choose Amazon EMR to manage clusters or lift-and-shift Hadoop/Spark, and EMR Serverless or AWS Glue for serverless pay-per-job (Glue for Spark-only, new-to-analytics teams).
- Analyze and visualize by persona: QuickSight (BI dashboards), Athena SQL (serverless one-off queries), OpenSearch Service (real-time log analytics + dashboards, the single service for real-time analyze-and-visualize); apply least privilege and the Data Analytics Lens pillars (security, performance efficiency, cost optimization, reliability).

Review questions

1. What are the five Vs of data, and which pair together drives pipeline throughput and scaling?

Answer: Value, veracity, volume, velocity, and variety. Volume and velocity together drive the expected throughput and scaling of the pipeline.

2. Distinguish homogeneous from heterogeneous ingestion, and ETL from ELT.

Answer: Homogeneous ingestion moves data as-is with no transformation (same format/engine). Heterogeneous ingestion transforms it: ETL transforms before loading (structured data → data warehouse); ELT loads raw then transforms (unstructured data → data lake).

3. Name the AWS Glue components beyond ETL jobs, and state what the Data Catalog does and does not store.

Answer: Data Catalog, DataBrew, Data Quality, and crawlers. The Data Catalog stores metadata and schema (table definitions, locations, attributes) but stores no datasets — the data stays in S3, RDS, and other stores.

4. A team needs replay of streaming data up to a year and fine control over consumption. Which streaming service, and why not Firehose?

Answer: Kinesis Data Streams — it retains data up to 365 days for replay and allows code-level control (KPL/KCL). Firehose does not store or replay delivered data (undelivered kept a maximum of 24 hours).

5. Why convert CSV to Parquet for large analytics files?

Answer: Parquet is columnar, compresses and encodes efficiently, and is splittable for parallel processing — speeding analytics and saving storage space, cost, and time; CSV is inefficient beyond about 15 GB.

6. Which four AWS services implement a data lake on AWS, and what does each contribute?

Answer: Amazon S3 (storage), AWS Lake Formation (management, security, and IAM access), AWS Glue (Data Catalog and transformation), and Amazon Athena (SQL query access).

7. Contrast schema-on-write with schema-on-read, and give the AWS example of each.

Answer: Schema-on-write requires the schema before data is written — the data warehouse (Amazon Redshift), a curated single source of truth. Schema-on-read applies a schema only when data is read — the data lake (Amazon S3), which stores data as-is.

8. A data engineer must analyze and visualize streaming user-activity logs in real time using a single service. Which service, and why are Athena, QuickSight, and Redshift wrong?

Answer: Amazon OpenSearch Service — it analyzes streaming logs and provides visualization via OpenSearch Dashboards in one service. Athena has no visualization; QuickSight needs a separate ingestion service; Redshift is a data warehouse, not real-time.