Ingestion & Transformation

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

FROM SOURCE TO USABLE DATA

Ingest Batch or stream → **Transform** Clean and enrich → **Validate** Quality gates → **Load** Publish safely

01 Data Ingestion and Transformation 34%

Choose the ingestion path

- **ETL** transforms before loading; **ELT** loads first and transforms in the destination. Choose batch or streaming from freshness, volume, replay and operational requirements.
- **AWS DMS** supports full load and ongoing change data capture from supported databases. Configure transaction logging, retention, connectivity and target mappings. **AWS SCT** converts eligible schemas and assesses conversion gaps; schema conversion and data migration are separate tasks.
- **DataSync** automates scheduled file and object transfers with validation. **Transfer Family** supplies managed file-transfer protocols; **S3 File Gateway** exposes file shares backed by S3. Transfer completion can trigger downstream processing.
- **AppFlow** integrates supported SaaS sources with AWS destinations. **AWS Data Exchange** distributes third-party datasets; neither replaces database transaction-log replication.

Kinesis partitioning and producers

- **Kinesis Data Streams** retains records for independent consumers and replay. A partition key maps records to a shard; ordering is scoped to the shard, not the whole stream.
- **Hot shards** often indicate skewed partition keys. Choose a well-distributed key while preserving required entity ordering; more shards alone cannot spread one hot key.
- **PutRecords** batches API submissions; inspect per-record failures and retry with backoff. **KPL aggregation** combines small user records to improve efficiency, with compatible consumer deaggregation.
- **Provisioned capacity** requires shard sizing and scaling. **On-demand** reduces capacity administration; monitor throttling and key skew under either mode.

Consumers, duplicates and replay

- **Enhanced fan-out** gives registered Kinesis consumers dedicated read throughput. Standard consumers share shard read capacity. Separate consumers can process the same stream independently.
- **KCL** manages shard leases, worker coordination and checkpoints. Checkpoint after durable processing; failures or resharding can cause records to be read again.

Ingestion & Transformation

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

FROM SOURCE TO USABLE DATA

Ingest
Batch or stream → **Transform**
Clean and enrich → **Validate**
Quality gates → **Load**
Publish safely

- **At-least-once delivery** requires idempotent sinks. Embed stable event IDs, deduplicate or upsert, and handle partial failures. Processing checkpoints alone do not eliminate producer duplicates.
- **Replay** needs sufficient stream retention or an archived raw copy. Isolate backfills from live processing capacity and use the same deduplication keys across both paths.

Buffered delivery with Data Firehose

- **Amazon Data Firehose** delivers streaming data to configured destinations such as S3, Redshift and OpenSearch. Buffering trades delivery latency for efficient writes; include it in the end-to-end freshness budget.
- **Lambda transformations** can normalize records before delivery. For supported format conversion, Firehose uses a Glue schema to convert JSON into **Parquet or ORC**; convert CSV to JSON first when required.
- **Redshift delivery** stages data in S3 and loads it with COPY. Direct Redshift streaming ingestion follows a separate path and avoids this S3 staging step.
- **CloudWatch Logs subscription filters** select and forward log events. Account for log envelopes, compression and destination format; monitor delivery failures and retain failed records for recovery.

Stateful streams and Kafka

- **Managed Service for Apache Flink** supports stateful streaming joins, aggregations and event-time windows. **Watermarks** track event-time progress; define handling for late and out-of-order events.
- **Tumbling windows** do not overlap; **sliding windows** overlap; **session windows** group activity separated by inactivity. Checkpoints recover operator state; end-to-end guarantees also depend on sources and sinks.
- **Amazon MSK** preserves Kafka APIs and topic/partition semantics. Kafka ordering is per partition. **MSK Connect** runs connectors; choose provisioned or serverless capacity against feature and workload needs.
- **MSK Replicator** can replicate between supported clusters for regional resilience. Plan failover, offsets, authentication and duplicate handling; asynchronous replication does not imply a fixed global latency or global ordering guarantee.

Glue transformation mechanics

- **Glue Spark jobs** distribute large transformations; **Python shell jobs** suit smaller non-distributed scripts. **Glue Studio** offers visual authoring, and interactive sessions support iterative notebook development.
- **DynamicFrames** represent semi-structured records and choice types. **ResolveChoice** resolves ambiguous types; **ApplyMapping** selects, renames and casts fields. Convert to a Spark DataFrame for Spark SQL and richer transformations.

Ingestion & Transformation

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

FROM SOURCE TO USABLE DATA

Ingest Batch or stream → **Transform** Clean and enrich → **Validate** Quality gates → **Load** Publish safely

- **DataBrew recipes** provide reusable visual cleaning and preparation steps; profile jobs summarize data characteristics. Standardize fields, validate formats and explicitly handle malformed records rather than assuming a cast repairs them.
- **Exact deduplication** uses business keys and deterministic survivor rules. **FindMatches** addresses fuzzy entity matching without a shared unique identifier and requires labeled examples and quality evaluation.
- **Custom libraries** must be packaged and supplied to the job runtime. Glue connections hold connectivity details; they do not automatically install application dependencies. EMR bootstrap actions can install cluster dependencies.

Orchestrate dependencies and failures

- **Glue workflows** coordinate Glue crawlers, jobs and triggers. Use conditional triggers for dependencies; a downstream join step must wait for all required upstream successes.
- **Step Functions** coordinates multiple AWS services with explicit state, retries and catches. Use a supported **.sync** integration to wait for a long-running job instead of keeping a Lambda function waiting.
- **Task** performs work; **Choice** branches on data; **Parallel** runs defined branches; **Map** iterates over a collection; **Wait** delays. Distributed Map can process S3 datasets and enforce tolerated failure thresholds.
- **Standard workflows** support long-running orchestration and callback task tokens. **Express workflows** suit short, high-volume executions; they do not support every Standard integration pattern.
- **Amazon MWAA** manages Apache Airflow for Python DAGs, operators, scheduling and retries. It suits existing Airflow workflows and heterogeneous dependencies. **EventBridge** routes events or starts work; it does not replace a dependency graph.

Event-driven code and decoupling

- **S3 ObjectCreated notifications** can filter by key prefix or suffix. Route to Lambda, a queue or EventBridge as appropriate. Design for duplicate events and keep output writes from retriggering the same pipeline.
- **Lambda** suits bounded event processing with a maximum execution time of 15 minutes. Raising memory also increases available CPU; measure runtime and cost together.
- **Reserved concurrency** reserves and caps a function's concurrency. **Provisioned concurrency** prepares execution environments to reduce cold starts. **Layers** share versioned dependencies; functions must adopt a new layer version.
- **SQS retention** controls message lifetime; **visibility timeout** controls how long an in-flight message is hidden. A **DLQ** isolates repeatedly failing messages, but does not save messages that simply expire while unprocessed.

Ingestion & Transformation / Data Stores

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

FROM SOURCE TO USABLE DATA

Ingest Batch or stream → **Transform** Clean and enrich → **Validate** Quality gates → **Load** Publish safely

SQL transformations and ML integration

- **WHERE** filters input rows; **GROUP BY** defines aggregation groups; **HAVING** filters aggregates. **COUNT(*)** counts rows, while COUNT(column) ignores nulls. Use **IS NULL**, not equality to NULL.
- **Joins** can multiply records when keys are non-unique. A left join preserves left-side rows; a WHERE predicate on the right side can remove unmatched rows. Validate key uniqueness and expected output counts.
- **CTAS** creates a table from query output; **WITH NO DATA** creates an empty result schema where supported. **INSERT INTO ... SELECT** appends to an existing table. **MERGE** applies matched and unmatched actions using a join condition.
- **ORDER BY event_time DESC LIMIT n** selects recent rows; ordering by an aggregate selects the largest groups instead. Regex **^** anchors the start, **$** the end and **|** expresses alternatives.
- **Redshift ML** exposes model inference through SQL, including supported SageMaker integrations. **Bedrock Knowledge Bases** retrieves source passages for grounded generation with citations; evaluate retrieval, language support and output accuracy.

02 Data Store Management 26%

Metadata, crawlers and schema

- **Glue Data Catalog** stores database, table, schema and partition metadata; it does not store the underlying dataset. Athena, Glue and supported EMR engines can share this metadata.
- **Crawlers** discover supported sources and infer schemas; **classifiers** recognize formats. Keep compatible schemas, formats and directory structures together to avoid unintended table proliferation.
- **Schema-change policies** control how crawlers update metadata. A crawler discovers changes but does not validate business correctness or rewrite old records. Catalog updates and data transformations are separate operations.
- **Glue Schema Registry** manages streaming schema versions and compatibility. It is distinct from the Data Catalog's table metadata and from a business glossary.

Data Stores

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

CHOOSE THE SERVING PATTERN

- **S3** — Analytical lake
- **Redshift** — Warehouse
- **DynamoDB** — Key access
- **Catalog** — Metadata

Partition discovery and pruning

- **Partition by common predicates**, such as event date or category. Hive-style paths use key=value segments. Queries must filter partition columns to benefit from pruning; excessive cardinality creates metadata and small-file overhead.
- **ALTER TABLE ADD PARTITION** or Glue partition APIs register known locations directly. **MSCK REPAIR TABLE** discovers compatible missing Hive-style partitions; it does not remove stale ones. Use **DROP PARTITION** to remove obsolete metadata.
- **Event-based crawlers** can consume S3 events from SQS to focus on changed locations. Configure crawler execution and first-run discovery; event collection alone does not continuously update the catalog.
- **Partition projection** computes expected partition locations from table properties for Athena. **Glue partition indexes** speed catalog partition filtering. Projection does not discover schema changes and is not a shared partition list for all engines.

File formats and lake organization

- **Parquet and ORC** are columnar: projection and predicate pushdown reduce reads for selective analytics. **CSV** is row-oriented text; **JSON** supports nested records; **Avro** is a schema-bearing row format.
- **Compression** reduces storage and scanned bytes. Parallelism depends on format, codec and block structure; one huge unsplittable compressed text file can bottleneck processing.
- **Compact small files** to reduce listing, planning and task overhead. Retain enough reasonably balanced files for parallel readers; one file per day is not automatically optimal.
- **Table locations** determine which S3 objects an engine reads. Crawler exclusion patterns do not filter later Athena reads. Keep incompatible file types outside the same table location and verify paths when queries return no data.

Transactional tables on S3

- **Iceberg, Hudi and Delta Lake** add table metadata and transactional operations over object storage. Match engine and runtime support before relying on updates, deletes, schema evolution or time travel.
- **Iceberg snapshots** represent committed table states. Use table-aware writes and MERGE operations; moving or deleting underlying files manually can break table consistency.
- **Copy-on-write** rewrites affected data files for simpler reads. **Merge-on-read** records changes for readers to reconcile, reducing write work but adding read and maintenance cost.
- **Compaction** combines small data files. **Snapshot expiration** removes old table history; **orphan cleanup** removes unreferenced files. Set retention around active readers and recovery needs; use Glue table optimizers where supported.

Data Stores

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

CHOOSE THE SERVING PATTERN

- **S3** — Analytical lake
- **Redshift** — Warehouse
- **DynamoDB** — Key access
- **Catalog** — Metadata

Load, export and merge in Redshift

- **COPY** loads bulk data in parallel. Use one command over a balanced file set; a **manifest** explicitly lists inputs across locations. Avoid serial single-file loads and row-by-row inserts for bulk ingestion.
- **UNLOAD** exports query results to S3, including columnar output options. The Redshift role needs destination S3 and KMS permissions; SQL login permission alone does not authorize the export.
- **Staging tables plus MERGE** update existing business keys and insert new ones. Deduplicate staging keys first and use appropriate transactions; removing duplicates inside one batch does not prevent duplicates from reruns.
- **Redshift streaming ingestion** reads supported Kinesis or MSK sources through an external schema into a materialized view. **Zero-ETL integrations** replicate eligible operational sources; configure supported versions, tables and target databases.

Redshift distribution and sort design

- **KEY distribution** can colocate large joins on the same well-distributed join key. Check cardinality, null concentration and workload skew; a frequently used key can still be a poor distribution key.
- **EVEN** spreads rows without join colocation. **ALL** replicates a table to each compute node and can help small, slowly changing dimensions. **AUTO** lets Redshift manage supported physical choices.
- **Sort keys** enable block skipping for filters. A compound key most benefits predicates on its leading columns; SQL predicate text order is not what controls this benefit.
- **Primary and foreign keys** describe relationships but are generally informational in Redshift. Ensure data integrity before declaring them, because the optimizer can rely on the declared relationships.

Warehouse models and shared data

- **Fact tables** hold events and measures; **dimension tables** describe entities. A star schema simplifies analytical joins. Choose normalization or denormalization for query patterns and update requirements.
- **Redshift Spectrum** queries external S3 data through catalog metadata. **Federated queries** access supported live relational sources. Combine local, external and federated data while accounting for source load and transfer costs.
- **Data sharing** exposes live Redshift data to authorized consumers using separate compute. Configure producer shares, consumer authorization and database permissions; it is distinct from exporting a copy to S3.
- **SUPER** stores semi-structured values for PartiQL navigation. **Schemas** organize objects and grants. **RA3** separates managed storage from compute; **Serverless** reduces cluster administration. The asynchronous **Redshift Data API** runs SQL over HTTPS without a persistent JDBC connection.

Data Stores

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

CHOOSE THE SERVING PATTERN

- **S3** — Analytical lake
- **Redshift** — Warehouse
- **DynamoDB** — Key access
- **Catalog** — Metadata

DynamoDB access patterns and retention

- **DynamoDB** supports key-based operational workloads at low latency. A **Query** requires a partition-key value and can constrain a sort key. A **Scan** examines the table or index; filter expressions do not avoid the initial reads.
- **GSIs** enable alternate partition and sort keys and use eventually consistent reads. **LSIs** retain the table partition key with another sort key and are created with the table. Design indexes from concrete access patterns.
- **On-demand capacity** suits unpredictable demand. **Provisioned capacity** supports predictable throughput with auto scaling; scheduled scaling can prepare for abrupt known peaks. Distribute keys to avoid hot partitions.
- **TTL** uses an item expiration timestamp in epoch seconds; deletion is asynchronous. Filter expired records when needed, and do not treat TTL as a precise deletion deadline. **Streams** can feed change processing or archival; **DAX** caches supported reads.

Match storage to the workload

- **Aurora and RDS** serve relational transactions; read replicas offload reads rather than write CPU. **OpenSearch** supports indexed full-text search and near real-time log analytics. **Timestream** models time-series measurements.
- **Neptune** serves graph relationships with languages such as **Gremlin** and **SPARQL**. **DocumentDB** provides MongoDB compatibility; **Keyspaces** provides Cassandra compatibility. Match data model, APIs and access patterns.
- **EBS** persists block data independently of an instance when deletion settings allow it. **Instance store** is ephemeral. Elastic Volumes can modify eligible volume types and performance; snapshots provide recovery history.
- **EFS** supplies shared file access, including supported Lambda mounts. Plan VPC reachability and mount targets. **DataSync** can consolidate regional files before local processing when data residency requires it.

S3 storage classes and protection

- **Standard** favors frequent access; **Standard-IA** and **Glacier Instant Retrieval** retain immediate reads with different cost tradeoffs. **One Zone-IA** has a different availability and durability failure boundary.
- **Glacier Flexible Retrieval and Deep Archive** need restore workflows before ordinary reads. Choose against required retrieval time, minimum storage duration, request costs and object size; keep active query data in a directly readable class.
- **Intelligent-Tiering** suits uncertain access patterns. Optional archive tiers change retrieval behavior. **Lifecycle rules** transition or expire objects; versioned buckets also need noncurrent-version and delete-marker cleanup rules.

Data Stores / Operations & Support

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

OPERATE WITH EVIDENCE

Observe Metrics and logs → **Diagnose** Find bottlenecks → **Improve** Tune or scale → **Recover** Replay safely

- **Versioning** enables recovery from overwrites and delete markers. **Object Lock compliance mode** prevents deletion of protected versions even by root during retention; governance mode permits authorized bypass. A legal hold remains until removed.
- **Storage Lens** summarizes storage usage, including noncurrent versions and incomplete multipart uploads. **S3 Inventory** supplies scheduled object listings. Use lifecycle cleanup for abandoned multipart uploads as well as obsolete versions.

03 Data Operations and Support 22%

Athena efficiency and execution plans

- **Athena** queries S3 with SQL without managing a cluster. Use columnar formats, compression, partition filters and selective column lists to reduce scanned data and latency; keep query data near the engine.
- **EXPLAIN** shows a plan without running the query. **EXPLAIN ANALYZE** runs it and reports execution statistics, so it can incur scan charges. Distinguish planning time, queue time and execution time.
- **Join statistics** help the optimizer choose a plan. For a distributed hash join, a small build side reduces memory pressure; check the plan rather than assuming table size or SQL text order always determines execution.
- **Workgroups** separate query configuration, history, permissions and cost controls. Data scan limits can stop excessive queries; capacity reservations address compute contention but do not repair inefficient SQL.
- **Result reuse** can avoid rerunning an eligible unchanged query within an accepted freshness interval. **Federated Query connectors** access supported external sources; include connector, source and spill costs. Spark workgroups support notebook-based analysis.

Redshift tuning and maintenance

- **WLM** separates workloads and manages queueing and resources. **Concurrency scaling** adds capacity for eligible concurrent workloads when enabled for queues; it is not a general fix for one inefficient query.
- **Materialized views** persist query results to avoid repeated work. Choose automatic, scheduled or explicit refresh against freshness needs. **Query editor v2** can schedule SQL, including stored procedure calls. Ordinary views store definitions and recompute results.
- **ANALYZE** updates optimizer statistics; **ANALYZE COMPRESSION** recommends encodings. **VACUUM** reclaims deleted space or restores sort order; REINDEX addresses interleaved-sort maintenance. Automatic maintenance may reduce manual work.
- **STL_ALERT_EVENT_LOG**, query plans and supported system views help identify skew, redistribution, missing statistics and expensive operations. Use **Redshift Advisor** recommendations and measure changes against the workload.

Operations & Support

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

OPERATE WITH EVIDENCE

Observe Metrics and logs → **Diagnose** Find bottlenecks → **Improve** Tune or scale → **Recover** Replay safely

Glue and EMR performance diagnosis

- **Spark UI** exposes stages, tasks, executor memory, shuffle and spill. Uneven task durations or one full disk can indicate skew; aggregate CPU alone can hide it.
- **Partition pruning**, predicate pushdown and early projection reduce work. **DynamicFrame file grouping** reduces small-input overhead for supported formats. Repartition for balanced work; avoid excessive tiny partitions.
- **Shuffle-heavy joins and sorts** can exhaust local disk. Address skew with better keys or salting, and use supported S3 shuffle storage where appropriate. Broadcast only a genuinely small side that fits memory.
- **Scale up** workers for per-executor memory or disk pressure; **scale out** for parallelizable load. Glue auto scaling and EMR managed scaling adjust capacity. Use **Flex** for eligible nonurgent Glue jobs; use Spot for interruption-tolerant EMR capacity and S3 for persistent data.

Observe the whole pipeline

- **CloudWatch metrics and alarms** detect operational thresholds. Track throughput, errors, lag, retries and freshness across every stage. **EventBridge** can route Glue or Step Functions state changes to team-specific **SNS standard topics**.
- **Kinesis iterator age** measures consumer lag; throughput-exceeded metrics identify throttling. Monitor Firehose delivery success and freshness. **DMS CDCLatencySource** and **CDCLatencyTarget** help separate source capture delay from target application delay.
- **Logs** explain individual failures: inspect Lambda application logs, Glue driver/executor logs, EMR Spark/YARN container logs and MWAA task logs. Scheduler and DAG-processing logs diagnose scheduling and import failures.
- **OpenTelemetry** collects traces; **Fluent Bit** can forward container logs. Correlate trace/request IDs across services. OpenSearch supports searchable logs and dashboards; database load and wait diagnostics help isolate RDS CPU bottlenecks.

Rules, profiles and quality gates

- **Glue Data Quality** uses **DQDL rulesets** for completeness, uniqueness, allowed values, ranges and cross-dataset checks. **ReferentialIntegrity** tests key membership in reference data; **DatasetMatch** compares corresponding datasets.
- **Evaluate Data Quality** integrates checks into ETL. Define whether a failed rule rejects records, quarantines output or fails the job; evaluating a rule does not automatically prevent publishing bad data.
- **Completeness** measures required values; **validity** measures conformance to allowed formats or ranges. Test row counts, nulls, duplicate keys, date ordering, reference coverage and freshness. A plausible count alone does not prove a complete delivery.
- **Anomaly detection** supplements explicit rules with unexpected metric changes. Reusable profiles and rules scale across datasets; version and deploy rules through APIs. Quality results can be surfaced through a configured business catalog integration.

Operations & Support

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

OPERATE WITH EVIDENCE

Observe Metrics and logs → **Diagnose** Find bottlenecks → **Improve** Tune or scale → **Recover** Replay safely

Reliable publication and failure recovery

- **Glue bookmarks** track previously processed source data for supported readers. Initialize the job, keep stable transformation contexts and call **job.commit()** after success. S3 bookmarks use object modification information; JDBC keys need suitable ordering.
- **Bookmarks are not transaction-log CDC** and do not provide target exactly-once writes. Reset or rewind can replay data; clean or upsert targets deliberately. Coordinate concurrency so runs do not corrupt shared state.
- **Validate before publication**: write to a staging location, check quality and completeness, then publish the accepted output. Preserve the last valid dataset and quarantine failures with enough metadata for diagnosis.
- **Retry transient failures** with backoff, bounded attempts and idempotent operations. Configure DLQs for exhausted delivery attempts and alarm on them. An EventBridge Scheduler DLQ captures failed deliveries; event-bus archive/replay is a separate feature.

Serving analytics and repeatable deployment

- **QuickSight SPICE** imports data for fast, scalable dashboard reads; refresh it to meet freshness requirements. **Direct query** reads the source at query time and adds source load. Dataset calculated fields can be materialized in SPICE; level-aware calculations control aggregation grain.
- **Amazon Q in QuickSight** supports natural-language BI workflows. Review generated calculations and reports, and maintain dataset permissions. QuickSight access needs the relevant source, S3 result and KMS permissions plus network reachability.
- **Infrastructure as code** makes pipelines repeatable. Define resources, permissions and S3 event wiring in **AWS SAM**; use **sam build** then **sam deploy**. In Git, fetch current changes and merge or rebase deliberately, resolving conflicts before review.
- **RTO** bounds recovery time; **RPO** bounds data loss. Test backups and restores. AWS Backup retention and vault lock protect supported recovery points; encrypted cross-Region Redshift snapshot copies need destination key permissions and the required snapshot copy grant.

Security & Governance

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

GOVERN EVERY ACCESS PATH

- **Identity** — Authenticate
- **Permissions** — Authorize
- **Encryption** — Protect
- **Audit** — Record access

04 Data Security and Governance 18%

Identity, roles and least privilege

- **Execution roles** give workloads temporary credentials. A **trust policy** controls who may assume a role; permission policies control allowed actions. **iam:PassRole** lets an authorized caller assign an approved role to a service.
- **Cross-account access** generally needs permissions on both the caller and the resource side. Check identity policies, resource policies, explicit denies and KMS key policies. **SCPs** and session policies constrain permissions; they do not grant access by themselves.
- **Lambda resource policies** authorize service invocation; the execution role governs what function code may access. Keep EventBridge invocation permission distinct from a function's data permissions.
- **IRSA** associates a pod service account with an IAM role through the EKS OIDC provider. **EMR runtime roles** scope individual steps. For Glue interactive sessions, check the caller's session permissions, role passing and execution-role trust.

Private paths and connectivity checks

- **Security groups** are stateful; **network ACLs** are stateless. Allow database ports from approved workload groups, include return-path rules where needed, and check DNS, routes and credentials. Merely sharing a subnet does not grant connectivity.
- **S3 gateway endpoints** add S3 routes to associated route tables. Check endpoint policy, bucket policy and IAM permissions together. An S3 bucket does not have a security group.
- **Interface endpoints** use private IPs and security groups for supported services. A VPC-attached Lambda function does not gain internet access simply by using a public subnet; use appropriate endpoints or NAT egress.
- **Glue VPC jobs** need free subnet addresses and appropriate self-referencing security-group traffic between workers. **Redshift enhanced VPC routing** controls COPY/UNLOAD data paths. **Direct Connect** supplies private connectivity; use TLS or other encryption controls separately.

Encryption, checksums and secrets

- **SSE-S3** uses S3-managed keys; **SSE-KMS** uses KMS keys and key permissions; **DSSE-KMS** applies two independent server-side encryption layers. Encryption at rest does not control which rows a user may query.
- **KMS decrypt permission** is separate from S3 read permission. Cross-account encryption access requires an appropriate customer-managed key policy and caller permissions. Grant writers the relevant encryption/data-key operations too.

Security & Governance

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

GOVERN EVERY ACCESS PATH

- **Identity** — Authenticate
- **Permissions** — Authorize
- **Encryption** — Protect
- **Audit** — Record access

- **Secrets Manager** stores and can rotate database credentials. Applications retrieve current secrets at runtime; rotation requires a supported integration or correctly configured rotation function and network access. Avoid hardcoded secrets and log exposure.
- **TLS** protects data in transit; **ACM** manages certificates for supported integrations. SFTP uses **SSH**, while FTPS uses TLS. EMR security configurations define encryption settings at cluster creation. **Checksums**, such as SHA-256, verify transfer integrity; an ETag is not universally an MD5 checksum.

Lake Formation access controls

- **Lake Formation** governs cataloged lake data with database, table and fine-grained grants. Register S3 locations and configure the registration role, data-location permissions and engine access as required.
- **Data filters** restrict rows, columns or cells for supported query engines. A country predicate controls which customer rows an analyst sees; column grants hide fields. S3 object policies cannot express SQL row filters.
- **LF-Tags** classify catalog resources and support scalable tag-based grants. They are governance metadata, not S3 object tags or labels on individual records. Use cross-account sharing and resource links where required.
- **Governed access paths** must remain governed: direct S3 permissions can expose data outside fine-grained query controls. Review IAM, bucket policies, catalog permissions and Lake Formation grants together.

Redshift authentication and data security

- **Federated sign-in** establishes user identity through supported IAM, SAML or IAM Identity Center integrations. Map identities to database permissions; successful authentication does not automatically grant table access.
- **Database RBAC** groups privileges by job function. **Row-level security** filters records; **column-level privileges** limit selected fields; **dynamic data masking** changes presented values according to policy and database role.
- **Masking** can show partial identifiers to support users and full values to authorized auditors without duplicating tables. Protect base tables and alternate access paths, and test combined policies using representative roles.

PII discovery, redaction and policy scope

- **Macie** discovers sensitive information in supported S3 data. It reports findings; it does not automatically redact records or grant access. **Glue Detect PII** and preparation transforms can identify and mask fields before publication.
- **Detection and masking** require defined identifiers, custom patterns where needed, and evaluation for false positives and negatives. Parsing support matters: a proprietary file format may need conversion before inspection.

Security & Governance

AWS Certified Data Engineer - Associate

KNOWLEDGE CHEATSHEET

GOVERN EVERY ACCESS PATH

- **Identity** — Authenticate
- **Permissions** — Authorize
- **Encryption** — Protect
- **Audit** — Record access

- **S3 object ARNs** scope GetObject/PutObject access to prefixes; **s3:prefix** constrains bucket listing. Endpoint-aware bucket conditions such as **aws:SourceVpce** restrict request paths; plan service access exceptions before using broad denies.
- **aws:CurrentTime** supports absolute timestamp conditions, not a recurring business-hours calendar by itself. Regional restrictions, time restrictions and row-level data locality are separate controls; test each against the actual request context.

Data mesh and business discovery

- **Data mesh** assigns domain teams ownership of data products with shared governance standards. Use S3 and catalogs for discovery, Lake Formation for governed lake access, and Redshift sharing for supported warehouse products.
- **Amazon DataZone** adds business glossaries, metadata forms, asset publishing and subscription workflows. Technical schemas describe structure; business metadata explains meaning, ownership and appropriate use.
- **SageMaker Unified Studio** organizes assets, users and projects in domains. **Domain units** delegate business-unit administration; users can discover assets and request access. IAM Identity Center-based domains provide centralized sign-in.

Audit and protect operational evidence

- **CloudTrail management events** capture control-plane API activity. **Data events** capture selected resource data operations, such as S3 object access and SNS publishing; configure the appropriate selectors rather than assuming every operation is logged.
- **Database audit logs** record SQL/user activity that control-plane API logs do not capture. Enable required Redshift user-activity logging and supported RDS engine audit settings, then centralize logs with restricted retention and access.
- **CloudTrail Lake** queries collected audit events. CloudWatch metric filters, alarms and EventBridge rules can route relevant failures; preserve principal and resource context when notifying responders.
- **CloudWatch Logs data protection policies** detect and mask supported sensitive data, with controlled unmask permissions. Avoid writing secrets in the first place; encryption and log-level filtering serve different purposes.