



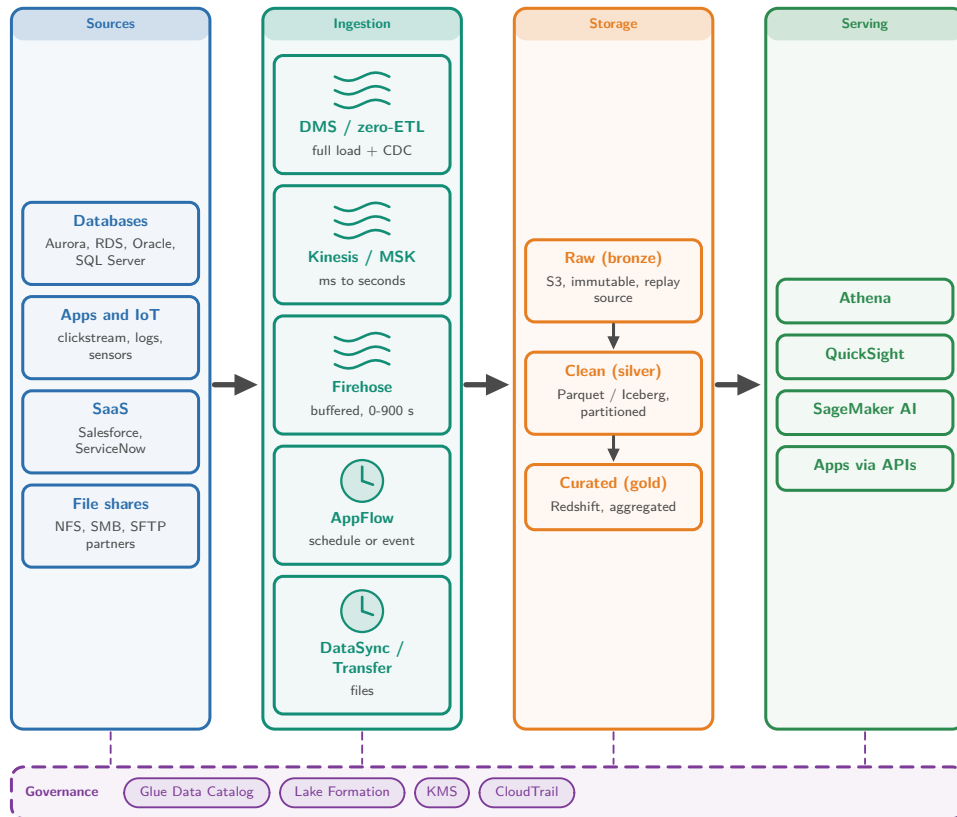
## DEA-C01 Domain 1a: Data Ingestion, Batch and Streaming

AWS DEA-C01 exam guide task 1.1, AWS service docs, as of 2026 | Kinesis Data Streams, Data Firehose, MSK, DMS, AppFlow, DataSync, Transfer Family, Snowball, S3 events and EventBridge; throughput limits, replay, fan-out, schedules and triggers

DEA-C01 Domain 1 34 % Kinesis MSK DMS

### © What task 1.1 asks you to do

DEA-C01 Domain 1 (data ingestion and transformation) is the heaviest domain at **34 %** of scored questions. Task statement 1.1, *perform data ingestion*, expects you to pick the ingestion service for a source and a latency, read its throughput and retention limits, choose between push and pull, schedule or trigger it, and make it replayable. This sheet covers 1.1; sheet 02 covers transformation (1.2), orchestration (1.3) and programming concepts (1.4).



Ingestion paths into an S3 data lake and Redshift: pick the box for the source and the latency

| Source and need                                           | Service                               | Why this one                                                                                         |
|-----------------------------------------------------------|---------------------------------------|------------------------------------------------------------------------------------------------------|
| Clickstream, sub-second, several custom consumers, replay | Kinesis Data Streams                  | ordered per key, retention 24 h up to 365 days, many consumers                                       |
| Logs to S3 in Parquet with the least code                 | Amazon Data Firehose                  | serverless, buffers, converts JSON → Parquet/ORC with a Glue table schema                            |
| Existing Kafka producers, Kafka APIs, connectors          | Amazon MSK (or MSK Serverless)        | lift Kafka unchanged; MSK Connect runs Debezium, S3 sink                                             |
| Ongoing replication of an Oracle or SQL Server database   | AWS DMS full load + CDC               | reads the transaction log; DMS Schema Conversion for engine changes                                  |
| Aurora / RDS / DynamoDB into Redshift, no pipeline        | zero-ETL integration                  | near-real-time replication managed by AWS, no Glue job                                               |
| Salesforce objects nightly to S3                          | Amazon AppFlow                        | SaaS connector, incremental pulls, no code agent, parallel transfer, checksum verification, schedule |
| Millions of files from an on-premises NAS                 | AWS DataSync                          |                                                                                                      |
| Partners push files over SFTP                             | AWS Transfer Family                   | managed SFTP/FTPS/AS2 endpoint writing to S3 or EFS                                                  |
| Hundreds of TB, weak network                              | AWS Snowball Edge                     | offline device shipped to AWS (Snowmobile is retired)                                                |
| React to every new object in S3                           | S3 Event Notifications or EventBridge | push events to Lambda, SQS, SNS, Step Functions                                                      |

### ⚡ Kinesis Data Streams: limits that drive the design

|                 |                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------|
| Write per shard | 1 MB/s or 1,000 records/s, whichever is hit first                                                     |
| Read per shard  | 2 MB/s shared by all polling consumers, 5 GetRecords calls/s; EFO: 2 MB/s per consumer per shard      |
| Record size     | up to 1 MB payload by default (base64 data blob + partition key)                                      |
| Retention       | 24 hours default, extend to 7 days or up to 365 days (long-term retention costs extra)                |
| Capacity modes  | provisioned (you split/merge shards) or on-demand (scales automatically, pay per GB written and read) |
| Ordering        | guaranteed within a shard, per partition key; not across shards                                       |
| Delivery        | at-least-once: producers retry, so consumers must be idempotent (dedupe on an event ID)               |

Ingest 6 MB/s · 4,500 records/s · 3 consumers · average record 1.37 KB (= D × 1,024 / N)  
Per shard: 1 MB/s or 1,000 records/s in; 2 MB/s out shared by all consumers, or 2 MB/s per consumer with enhanced fan-out

SHARDS NEEDED FOR EACH LIMIT

|                            |                                                               |                    |
|----------------------------|---------------------------------------------------------------|--------------------|
| Write bandwidth            | $\lceil D / 1 \rceil = \lceil 6 / 1 \rceil$                   | 6 shards           |
| Write records              | $\lceil N / 1,000 \rceil = \lceil 4,500 / 1,000 \rceil$       | 5 shards           |
| Read, shared (3 consumers) | $\lceil D \times C / 2 \rceil = \lceil 6 \times 3 / 2 \rceil$ | 9 shards ← binding |
| Read, enhanced fan-out     | $\lceil D / 2 \rceil = \lceil 6 / 2 \rceil$                   | 3 shards           |

where D = ingest MB/s, N = records per second, C = consumers reading the whole stream; the stream needs the LARGEST count (the binding limit).

99 shards with shared reads, 6 with enhanced fan-out; shared reads chosen

Above a limit the stream throttles writes or reads (throughput exceeded); re-sharding splits or merges units. Hot partition keys can still overload a single unit.

Shards needed per limit for 6 MB/s, 4,500 records/s and 3 consumers: the read side binds unless consumers use enhanced fan-out

⚠️ Producers get ProvisionedThroughputExceededException

HIGHCOMMON

LIKELY CAUSES

1. Hot shard: skewed partition keys

2. Too few shards for the peak

3. Many small PutRecord calls instead of batches

HOW TO CONFIRM

Per-shard metrics (enhanced shard-level monitoring) show one shard at the write limit

```
aws kinesis describe-stream-summary --stream-name clicks
```

FIX

✔️ Use a high-cardinality key (user ID, random suffix)

✔️ Split the hot shard or switch to on-demand

✔️ Batch with PutRecords or the KPL with aggregation

✔️ Retry with exponential backoff

⚠️ Consumer lag (GetRecords.IteratorAgeMilliseconds) keeps growing

MEDIUMCOMMON

LIKELY CAUSES

1. Lambda batch fails and is retried forever

2. Consumers share 2 MB/s per shard

HOW TO CONFIRM

IteratorAge rising while incoming bytes are flat

FIX

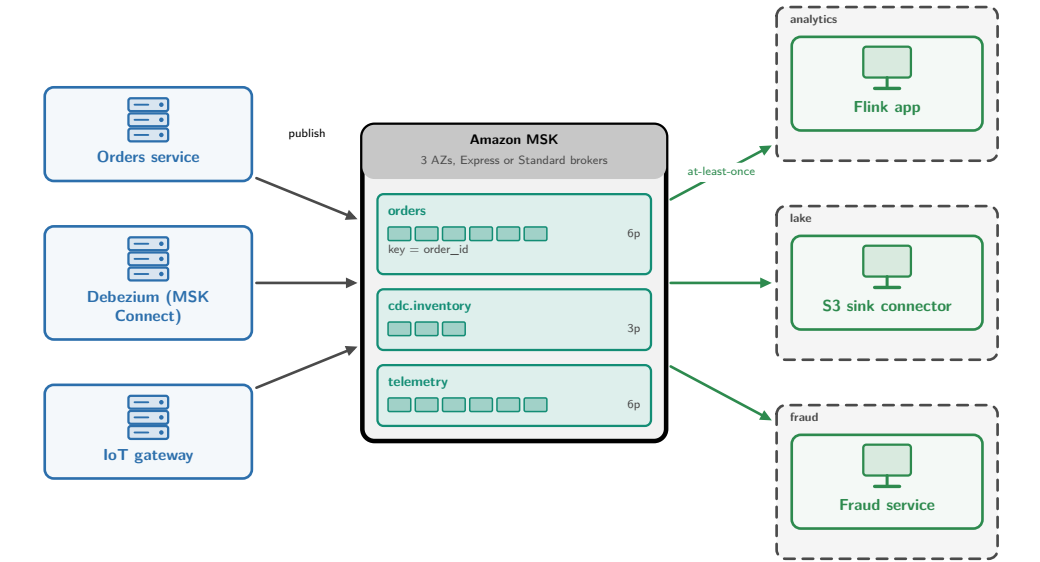
✔️ Set BisectBatchOnFunctionError, MaximumRetryAttempts and an on-failure destination

✔️ Register consumers with enhanced fan-out

✔️ Raise ParallelizationFactor (up to 10 per shard) on the Lambda event source mapping

| Command                                                                                                   | What it does                                      |
|-----------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| aws kinesis create-stream --stream-name clicks --stream-mode-details StreamMode=ON_DEMAND                 | on-demand stream, no shard planning               |
| aws kinesis put-record --stream-name clicks --partition-key u-42 --data <base64>                          | write one record; the key picks the shard         |
| aws kinesis increase-stream-retention-period --stream-name clicks --retention-period-hours 168            | keep 7 days for replay                            |
| aws kinesis register-stream-consumer --stream-arn <arn> --consumer-name fraud                             | enhanced fan-out consumer with its own throughput |
| aws kinesis update-shard-count --stream-name clicks --target-shard-count 8 --scaling-type UNIFORM_SCALING | reshard a provisioned stream                      |

🔗 Firehose, MSK and Managed Flink



Amazon MSK: producers write to partitioned topics, each consumer group reads every partition once

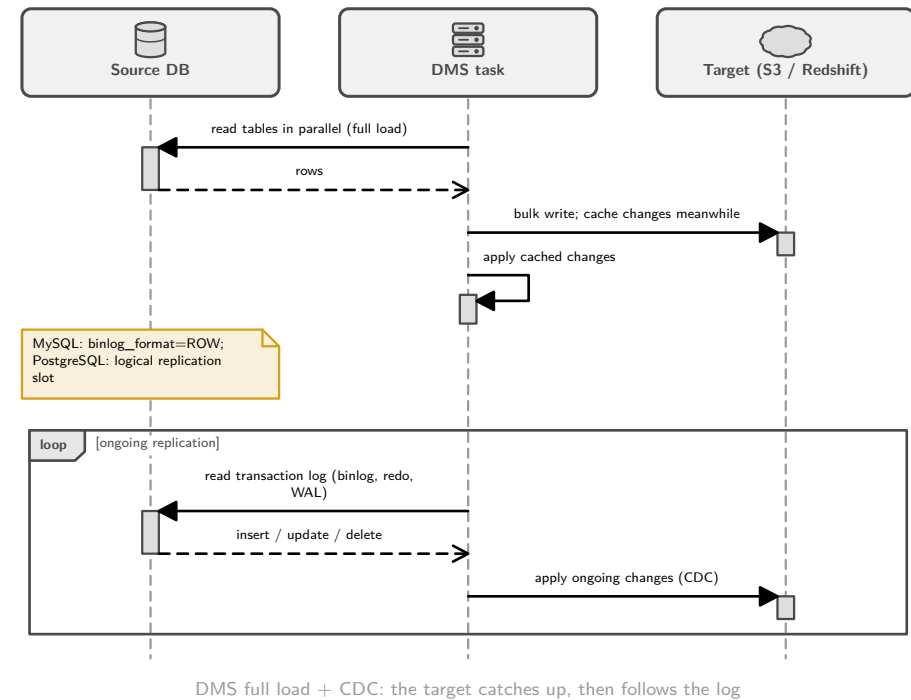
| Kinesis Data Streams                          | Amazon MSK                                                                             | Amazon Data Firehose                                                           |
|-----------------------------------------------|----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| AWS API, shards<br>retention 24 h to 365 days | Kafka API, partitions<br>retention by time or size, tiered<br>storage for long history | no consumer code at all<br>no retention: buffer then deliver                   |
| on-demand or provisioned                      | provisioned (Standard, Express) or<br>Serverless                                       | serverless, pay per GB ingested                                                |
| Lambda, KCL, Flink, Firehose<br>consumers     | any Kafka client, Flink, MSK Con-<br>nect                                              | destinations: S3, Redshift,<br>OpenSearch, Splunk, Snowflake,<br>Iceberg, HTTP |
| choose for AWS-native real time               | choose when Kafka already exists<br>or for Kafka ecosystem                             | choose for 'load into S3/Redshift<br>with least effort'                        |

- **Firehose buffering** size 1-128 MiB and interval 0-900 s; the first reached flushes. Zero buffering gives the lowest latency but many small files
- **Firehose transforms** Lambda transformation per batch, record format conversion JSON → Parquet/ORC from a Glue table, dynamic partitioning by keys in the record (S3 prefixes such as year=/month=/)
- **Firehose to Redshift** writes to an intermediate S3 bucket, then runs COPY; failed records go to an error prefix in S3
- **Managed Service for Apache Flink** formerly Kinesis Data Analytics: stateful stream processing (windows, joins) in Java, Python or SQL (Studio notebooks), reading Kinesis or MSK
- **Redshift streaming ingestion** a materialized view over a Kinesis stream or MSK topic, refreshed to land records in Redshift with low latency and no Firehose

### EXAM TIP

Keyword map: 'near real time into S3, least operational overhead' → Firehose. 'Real time, custom processing, replay, multiple consumers' → Kinesis Data Streams. 'Existing Apache Kafka' → MSK. 'Windowed aggregations on a stream' → Managed Service for Apache Flink. Firehose is never 'real time' in the sub-second sense: it buffers.

## Databases, SaaS and files



### DMS setup that works the first time

✓ baseline / mandatory    □ recommended – verify in your context

- ✓ Enable the source's change log: MySQL binary logging in ROW format, PostgreSQL logical replication, Oracle supplemental logging
- ✓ Size the replication instance (or use DMS Serverless capacity units) and place it in the same VPC/region as the source
- ✓ Pick LOB mode: limited LOB (fast, truncates above the max size) or full LOB (slow, complete)
- **SHOULD** Turn on data validation for the tables that matter compares source and target rows after load
- **SHOULD** Writing CDC to S3: add TimestampColumnName and keep the 0p column (I/U/D) so a downstream job can merge
- **MUST** Heterogeneous migration (Oracle → Aurora PostgreSQL): convert the schema first with DMS Schema Conversion

**AppFlow flow trigger** — On demand, on schedule (incremental pulls by a timestamp field) or on event (Salesforce change events); can filter, mask and validate fields on the way.

**DataSync task** — Source location + destination location + options (verify, bandwidth limit, include/exclude filters, schedule); an agent VM is needed for on-premises sources.

**S3 Transfer Acceleration** — Upload to the nearest edge location, then over the AWS backbone to the bucket; helps distant clients send large objects.

### 🔥 Small-file pain

A team sent IoT events straight to S3 with one PUT per event: 40 million objects a day, Athena queries slowed to minutes. Moving to Firehose with 128 MiB / 300 s buffers and Parquet conversion cut objects to a few thousand a day and scanned bytes by about 90 %.

## 🕒 Triggers, schedules and replay

| Need                                        | Mechanism                                                           | Detail                                                                                                |
|---------------------------------------------|---------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| Run every night at 02:00 UTC                | EventBridge Scheduler or a rule with cron(0 2 * * ? *)              | AWS cron has 6 fields (min hour day-of-month month day-of-week year); one of the day fields must be ? |
| Run every 15 minutes                        | rate(15 minutes)                                                    | rate expressions for fixed intervals                                                                  |
| Start when a file lands                     | S3 → EventBridge rule on Object Created with a prefix/suffix filter | EventBridge must be enabled on the bucket; then route to Step Functions, Glue workflow, Lambda        |
| Process stream records                      | Lambda event source mapping on Kinesis, MSK, DynamoDB Streams, SQS  | batch size, batching window, starting position TRIM_HORIZON / LATEST / AT_TIMESTAMP                   |
| Connect a source to a target with filtering | EventBridge Pipes                                                   | source (SQS, Kinesis, DynamoDB Streams, MSK) → filter → enrich (Lambda, API) → target                 |
| Batch ingestion inside a pipeline           | Glue workflow trigger, Step Functions, MWAA DAG                     | see sheet 02 for orchestration                                                                        |

### 💡 Make every pipeline replayable

Keep the raw records immutable in an S3 raw zone (or in stream retention long enough to cover a bug-fix cycle), make loads idempotent (MERGE on a key, overwrite a partition), and record the last processed position (Kinesis checkpoint in DynamoDB with the KCL, Kafka consumer offsets, Glue job bookmarks). Replay = reset the position and rerun.

### 📄 Worked example: sizing a clickstream

Peak 4,500 events/s of 1.2 KB each, three consumers (Lambda alerts, Flink, Firehose to S3). Ingest =  $4,500 \times 1.2 \text{ KB} = 5.4 \text{ MB/s} \rightarrow 6$  shards for the 1 MB/s write limit (4,500 records/s needs only 5). Reads: three polling consumers  $\times 5.4 \text{ MB/s} = 16.2 \text{ MB/s} \rightarrow 9$  shards at 2 MB/s. Options: 9 provisioned shards, or 6 shards with enhanced fan-out for the two latency-sensitive consumers (Firehose reads as a normal consumer), or on-demand mode if the peak is unpredictable.

## ✓ Myths, recap and self-test

### MYTH

✗ Kinesis guarantees global ordering

✗ Firehose delivers each record instantly

✗ DMS needs no change on the source

✗ On-demand Kinesis has no limits

### FACT

✓ Ordering holds only within a shard

Records with the same partition key stay in order; across keys and shards there is no order.

✓ Firehose buffers by size and time

Latency is the buffer interval (0-900 s) plus delivery; choose Kinesis Data Streams for sub-second consumers.

✓ CDC needs the source's change log enabled

Binary logging in ROW format, logical replication or supplemental logging, and enough log retention.

✓ On-demand still has per-shard and account write limits

It scales automatically from a default write capacity, but hot partition keys still throttle.

## Key takeaways

### 01

Latency picks the family

seconds = streams, minutes = Firehose, hours = batch tools

### 02

Size on the binding limit

MB/s, records/s or the read side

### 03

Keys decide hot shards

high-cardinality partition keys

### 04

CDC needs the log

enable it before the DMS task

### 05

Raw zone = replay

immutable S3 copy of what arrived

Q1 Write limit of one Kinesis shard?

A 1 MB/s or 1,000 records/s, whichever comes first.

Q2 Three consumers each need 2 MB/s per shard without contention: which feature?

A Enhanced fan-out: each registered consumer gets 2 MB/s per shard pushed over HTTP/2.

Q3 Default and maximum Kinesis retention?

A 24 hours default, up to 365 days.

**Q4** Land JSON logs in S3 as Parquet with no code?

**A** Firehose with record format conversion, using a Glue Data Catalog table schema.

**Q5** Replicate an on-premises Oracle database continuously into S3?

**A** AWS DMS task with full load + CDC, S3 target endpoint (Parquet), source supplemental logging on.

**Q6** Pull Salesforce data into S3 every hour with no code?

**A** Amazon AppFlow scheduled flow with incremental transfer.

**Q7** What does a Lambda event source mapping's `BisectBatchOnFunctionError` do?

**A** Splits a failing batch in two and retries each half, isolating the bad record instead of blocking the shard.

**Q8** Which AWS cron field must be ??

**A** Either day-of-month or day-of-week (not both may be set); AWS cron has six fields including year.

**Q9** Aurora MySQL into Redshift with no pipeline to run?

**A** Aurora zero-ETL integration with Amazon Redshift.