**1) Driver OOM (Out-of-Memory)**

**How to find it (UI path)**

1. **Workflows → Jobs → Runs → (your run) → View output**
2. Click **Spark UI** (top right).
3. **Executors → Driver (logs)** → open **stderr** for OutOfMemoryError / Java heap space / GC overhead limit exceeded.
4. **SQL** tab → open latest statement → check **Physical Plan** for CollectLimit, Broadcast, Exchange right before failure.

**Common symptoms**

* Run aborts early; no tasks finish.
* toPandas(), collect(), or huge display() results.
* Very large **broadcast** or driver-side aggregations.

**Likely root causes**

* Pulling too much to the driver (collect/toPandas).
* Driver planning very large broadcast or JSON serialization spikes.
* Too many result rows in notebook cell.

**Fix fast (choose the lowest-risk first)**

* **Stop collecting to driver**: write to Delta and **sample**:
* df.write.mode("overwrite").saveAsTable("tmp.safe\_preview");
* spark.table("tmp.safe\_preview").limit(20).display()
* **Cap output**: df.limit(1000).toPandas() only for true samples.
* **Push work to executors**: replace Python UDFs with SQL/Builtin/Pandas UDFs.
* **Tune memory only if needed** (cluster policy): raise **Driver Memory/Overhead** one step.
* **Avoid massive broadcasts**: ensure the “small” side really is small; or drop the hint.

**Verify**

* Re-run and watch **Spark UI → Executors → Driver** memory/GC; **SQL** tab shows no Collect of huge datasets; job completes.

**2) Shuffle Fetch Failures (e.g., FetchFailedException)**

**How to find it (UI path)**

1. **Workflows → Jobs → Runs** → the run will show **failed stage**.
2. **Spark UI → Stages** → failed stage shows red; open it.
3. In the **Event Timeline** and **Task** table, look for **Failed Tasks = X**; open one → bottom shows **FetchFailed** with shuffle block / lost executor details.
4. **Executors** tab → check if an executor died (exit reason, lost shuffle blocks).

**Common symptoms**

* Job fails late in a wide stage (join/agg) with **lots of shuffle**.
* Error mentions **shuffle block** / **connection** / **executor lost**.
* Retries help intermittently.

**Likely root causes**

* Executor loss during shuffle (node preemption, OOM, long GC).
* Network timeouts or too-large in-flight shuffle fetches.
* Oversized partitions due to skew (one reducer fetching huge blocks).

**Fix fast**

* **Stabilize & retry** (often transient): re-run once.
* **Reduce per-task shuffle pressure**: increase partitions (smaller blocks):
* spark.conf.set("spark.sql.adaptive.enabled","true") # AQE
* spark.conf.set("spark.sql.shuffle.partitions","auto") # or bump (e.g., 600–1200)
* **Broadcast the small side** to avoid shuffle on that table:
* SELECT /\*+ BROADCAST(dim) \*/ ...
* **Address skew** (see Section 3): salt keys or let AQE split skewed partitions.
* **Soften network constraints** (use sparingly if you must):
* spark.conf.set("spark.network.timeout","600s")
* # (advanced) spark.reducer.maxSizeInFlight=48m → 24m to reduce in-flight
* **Health of executors**: ensure cluster isn’t undersized; prevent executor OOM (see next).

**Verify**

* **Stages** page now shows **no failed tasks**; **Shuffle read time** drops; **Executors** page shows stable executor count.

**3) Stragglers & Tail Latency (mitigate with Speculation)**

**How to spot stragglers (UI path)**

1. **Spark UI → Stages** → open slow stage.
2. **Summary Metrics**: Big spread between median and max task time.
3. **Task table**: one/few tasks many× slower than peers.
4. **SQL** tab physical plan often shows Exchange → shuffle; stragglers are usually reducers for skewed keys.

**Likely root causes**

* **Data skew**: a reducer got most of one hot key’s rows.
* **Noisy executor**: hardware hiccup or transient GC.
* **Cold I/O** or remote block locality issues.

**Mitigations (order of preference)**

**A) Fix the cause (best)**

* **Enable AQE + skew split**:
* spark.conf.set("spark.sql.adaptive.enabled","true")
* spark.conf.set("spark.sql.adaptive.skewJoin.enabled","true")
* **Broadcast** small side to eliminate shuffle on one input.
* **Repartition by balanced key** or **range**:
* df.repartition(800, "date") # example; size by bytes/target ~128MB
* **Salt hot keys** (last resort for extreme skew).

**B) Turn on speculation (tail-cutting)**

* Good when tasks are **idempotent** (most map/shuffle read tasks). Avoid for tasks with non-idempotent side effects.
* Settings:
* spark.conf.set("spark.speculation","true")
* spark.conf.set("spark.speculation.quantile","0.75") # when 25% fastest tasks finish
* spark.conf.set("spark.speculation.multiplier","1.5") # outlier if 1.5× median
* spark.conf.set("spark.speculation.minTaskRuntime","60s") # avoid short tasks
* Expect a small compute overhead (duplicates) but shorter stage tail.

**Verify**

* **Stage duration** drops; **Task time distribution** tightens; fewer long tails.
* **Jobs** page total runtime improves; **Executors** show minor increase in task attempts (expected).

**Quick Extras (often related)**

**Executor OOM (not the driver)**

* **UI:** Spark UI → Executors → look for killed executors; **stderr** shows OOM.
* **Fix:** Increase partitions (smaller tasks), **broadcast** joins, reduce columns early, prefer built-in functions over UDFs, consider raising executor memory/overhead one step.
* **Verify:** No new OOMs; **Spill** metrics fall in Stages.

**File smallness (tiny file storms)**

* **UI:** SQL tab → scan time high, many files; **Delta** history shows frequent small writes.
* **Fix:** Batch writes, OPTIMIZE compaction (Databricks), target ~**128MB** files.
* **Verify:** Fewer files scanned; query wall-clock improves.

**Triage Checklist (copy/paste)**

**When a run fails:**

1. Jobs → Runs → **Spark UI** → note **failing stage** and error type.
2. **Stages** → open stage → check **Failed Tasks**, **Shuffle Read/Write**, **Spill**.
3. **Executors** → driver/executor logs for OOM/exit reasons.
4. **SQL** → physical plan: broadcast? exchange? big collect?
5. Apply the targeted fix (above), **re-run**, and compare **stage/task metrics**.

**Config Snippets (safe starters)**

# Adaptive + skew handling

spark.conf.set("spark.sql.adaptive.enabled","true")

spark.conf.set("spark.sql.adaptive.skewJoin.enabled","true")

# Partition sizing (or 'auto' on newer DBR)

spark.conf.set("spark.sql.shuffle.partitions","600") # tune to keep partitions ~128MB

# Speculation (tail cutting)

spark.conf.set("spark.speculation","true")

spark.conf.set("spark.speculation.quantile","0.75")

spark.conf.set("spark.speculation.multiplier","1.5")

spark.conf.set("spark.speculation.minTaskRuntime","60s")

**A) Streaming & Ingestion**

**1) Checkpoint missing/corrupt (Structured Streaming)**

**UI:** Workflows → Jobs → Runs → Spark UI → **Driver logs** (+ **Storage** tab if caching used)  
**Symptoms:** Stream won’t start, throws checkpoint/offset errors; repeated restarts.  
**Root causes:** Checkpoint path deleted/overwritten; wrong cluster perms; schema change incompatible with state.  
**Fix:**

* If safe: **rename** or **reset** checkpoint dir and **backfill** from source.
* Ensure **checkpointLocation** points to a stable, writeable path (S3/ABFS); fix ACLs.
* For schema changes, add **explicit casts** or migrate state with a one-time batch.  
  **Verify:** Stream starts cleanly; **Spark UI → Streaming Query** (if available) shows steady progress; lag drops.

**2) Watermark / state-store explosion**

**UI:** Spark UI → **SQL** (open query) → **Plan** shows stateful ops; **Stages** shows very long tasks with huge spill; **Driver logs** mention “state store memory”  
**Symptoms:** Throughput collapses over time; OOM; long GC; disk spill heavy.  
**Root causes:** Missing/too-large watermark; unbounded state from late data; wide windows.  
**Fix:**

* Set reasonable **withWatermark(…, '10 minutes')** and trim window widths.
* Periodically **checkpoint/compact** results to break lineage.
* Route very late data to a **dead-letter** table for replay.  
  **Verify:** Batch duration stabilizes; **state operator size** (in progress logs) plateaus.

**3) Autoloader schema inference / evolution failures**

**UI:** Jobs → Runs → **Output**; **Driver logs** show schema evolution exceptions  
**Symptoms:** Stream errors after a new field appears; Bronze stops ingesting.  
**Root causes:** No schema evolution options; schema location not writeable; incompatible type change.  
**Fix:**

* Use **cloudFiles.schemaLocation** with write perms.
* Set **evolution** opts (e.g., cloudFiles.inferColumnTypes=true) and explicitly cast/normalize in Silver.
* For breaking changes, land raw to **\_rescued\_data** and handle in Silver.  
  **Verify:** New columns appear; ingestion resumes; **Bronze** table shows fresh files.

**B) Delta Lake & Table Management**

**4) Optimistic concurrency (Delta commit conflicts)**

**UI:** Jobs → Runs → **Output** shows “ConcurrentModificationException”; Data → Data Explorer → table → **History**  
**Symptoms:** MERGE/UPDATE fails under write contention; retries succeed later.  
**Root causes:** Multiple writers to same partitions; large MERGE touching many files.  
**Fix:**

* **Partition** to reduce overlap; **upsert** in **smaller batches**.
* Use **idempotent** keys; add **retry** with exponential backoff.
* Consider **applyChangesInto** (DLT) or staging tables to stage diffs.  
  **Verify:** Fewer conflicts in logs; **history** shows successful commits; job retries drop.

**5) VACUUM removed files needed for time travel / streaming**

**UI:** Data Explorer → table → **History**; SQL editor to check TBLPROPERTIES  
**Symptoms:** Time travel fails; stream queries error on missing files.  
**Root causes:** Too-aggressive **VACUUM**; retention shorter than readers/checkpoints require.  
**Fix:**

* Increase delta.deletedFileRetentionDuration & delta.logRetentionDuration.
* **Pause VACUUM** until consumers updated; reprocess from source if needed.  
  **Verify:** New vacuums keep required versions; no “file not found” in stream logs.

**6) Transaction log / metadata bloat (too many small files)**

**UI:** Data Explorer → **History** (massive commits); SQL → DESCRIBE DETAIL (lots of files); Jobs failing during OPTIMIZE  
**Symptoms:** Slow planning; huge commit times; OPTIMIZE/MERGE sluggish.  
**Root causes:** Tiny-file storm from micro-batches; overly granular partitioning.  
**Fix:**

* Schedule **OPTIMIZE** compaction; adjust **micro-batch** size; lift **maxFilesPerTrigger**.
* Reconsider partition strategy; target ~**128MB** files.  
  **Verify:** Query planning time down; fewer files scanned; **OPTIMIZE** runtime normalizes.

**C) Permissions, UC & External Storage**

**7) Permission denied (Unity Catalog / RBAC)**

**UI:** Data → Data Explorer → object → **Permissions**; **Lineage** for upstream table perms  
**Symptoms:** PERMISSION\_DENIED / cannot read/write table.  
**Root causes:** Missing object grants, missing **USAGE** on catalog/schema, role not inherited.  
**Fix:**

* Grant **USAGE** on catalog/schema + **SELECT/MODIFY** on table/view.
* Use **roles** (group-level) not individuals.  
  **Verify:** Same query succeeds from target principal; **Permissions** shows expected grants.

**8) External Location / S3 access failures**

**UI:** Admin Console (or UC) → **External locations** & **Storage credentials**; Jobs → **Driver logs** show 403/AccessDenied  
**Symptoms:** Table reads/writes fail with S3 403; location cannot be listed.  
**Root causes:** IAM role trust wrong (ExternalId), bucket policy missing, path mismatch.  
**Fix:**

* Re-check **IAM trust** (Databricks AWS account + ExternalId).
* Ensure bucket policy grants list/get/put on required prefixes.
* Validate **url** exactly matches bucket/prefix in External Location.  
  **Verify:** External Location **Validate** passes; read/write succeeds.

**9) Secrets / credentials not found**

**UI:** Admin → **Secret scopes**; Jobs → **Output** shows Secret not found  
**Symptoms:** Notebook fails resolving dbutils.secrets.get.  
**Root causes:** Wrong scope/key; unauthorized principal; scope not in this workspace.  
**Fix:**

* Create/verify scope and key; grant **READ** to job principal / cluster.
* Reference with exact **scope/key** names.  
  **Verify:** Secret resolves (print length only); downstream JDBC/REST calls succeed.

**D) Compute, Environment & Dependency Issues**

**10) Pool exhaustion / autoscaling limits**

**UI:** Compute → **Pools** (capacity/idle); Jobs → run timeline shows pending state long  
**Symptoms:** Runs stuck in **Pending**; long spin-up.  
**Root causes:** Pool at capacity; autoscaling minimum too low; quota limits.  
**Fix:**

* Increase **pool capacity**; raise **min workers** for busy hours; schedule staggering.
* For quotas, request increase or reduce concurrency.  
  **Verify:** New runs start within expected time; pool metrics stable.

**11) Library / environment conflicts (Py4J, version skew)**

**UI:** Jobs → **Cluster** → Libraries; **Driver logs** show import errors / ClassNotFound  
**Symptoms:** Fails on import; UDFs crash; version mismatch errors.  
**Root causes:** Conflicting wheel versions; incompatible Scala/Spark; mixing pip/cluster-installed libs.  
**Fix:**

* Pin versions in **requirements.txt** / wheel; avoid mixing install methods.
* Use **init scripts** or **Repos**-based package mgmt; align to DBR runtime.  
  **Verify:** Import succeeds; job runs; UDFs stable.

**12) Driver/executor local disk full (ephemeral)**

**UI:** Spark UI → **Executors** (storage/memory); **Driver logs** mention “No space left on device”  
**Symptoms:** Stages fail writing shuffle/temp; checkpoint writes fail.  
**Root causes:** Excess spill; huge temp logs; oversized broadcast; runaway checkpoints.  
**Fix:**

* Reduce shuffle pressure (more partitions, broadcast small side).
* Clean/rotate logs; move checkpoints to S3/ABFS; increase local disk size (cluster type).  
  **Verify:** No new disk errors; spill metrics drop.

**13) Network / S3 throttling (429/5xx)**

**UI:** Driver logs show S3 503/SlowDown; **Stages** show retries; long shuffle read  
**Symptoms:** Intermittent failures; retries succeed sometimes.  
**Root causes:** Burst traffic, tight maxInFlight, transient provider issues.  
**Fix:**

* Back off: bump spark.network.timeout, reduce spark.reducer.maxSizeInFlight a notch; retry.
* Stagger heavy jobs; consider **pool** warm nodes.  
  **Verify:** Fewer transient fetch failures; job stabilizes.

**E) SQL Warehouses, DLT, Scheduling**

**14) SQL Warehouse capacity / scaling failures**

**UI:** SQL Warehouses → **Events/Logs**; Query History → error text  
**Symptoms:** Queries queue or fail to start; “insufficient capacity.”  
**Root causes:** Warehouse min/max set too low; region capacity constraint.  
**Fix:**

* Raise **max clusters**; enable **serverless** if allowed; adjust size/T-shirt.  
  **Verify:** Queries start promptly; queue time down.

**15) DLT pipeline halts (expectation failures / configuration)**

**UI:** Workflows → DLT Pipelines → **Event log**; **Expectations** tab  
**Symptoms:** Pipeline red/amber; rows dropped or pipeline stops on violation.  
**Root causes:** Expectations set to **FAIL**; schema mismatch; source missing.  
**Fix:**

* Set expectation action to **DROP** or **QUARANTINE** while triaging.
* Fix source schema; add quarantine sink for bad rows.  
  **Verify:** Runs turn green; expectation metrics look sane.

**16) Job concurrency / overlap**

**UI:** Workflows → Job → **Settings** (max concurrent runs)  
**Symptoms:** New runs skip/cancel or clash on locks; table write conflicts.  
**Root causes:** Concurrency > 1 on non-idempotent steps; checkpoint contention.  
**Fix:**

* Set **Max concurrent runs = 1** for those jobs; add **mutex** resource pattern; separate checkpoints.  
  **Verify:** No overlapping writes; retries drop.

**F) Data Modeling / Query Pitfalls**

**17) Partition misalignment / no pruning**

**UI:** SQL → **Query Profile** shows full-scan; **Plan** lacks “Dynamic Pruning”  
**Symptoms:** Scans all files; slow filter queries.  
**Root causes:** Filter not on partition column; non-deterministic expressions; different data types.  
**Fix:**

* Align filters with **partition columns**; cast to correct types; enable **DPP** (usually on by default).  
  **Verify:** Fewer files scanned; query time improved.

**18) Exploding joins (cartesian / accidental many-to-many)**

**UI:** SQL → Plan shows **CartesianProduct** or huge shuffle; **Stages** bytes blow up  
**Symptoms:** Stage time/memory skyrockets; OOM.  
**Root causes:** Missing join condition; low-cardinality cross joins; duplicate keys.  
**Fix:**

* Confirm join keys; pre-aggregate/deduplicate; use **semi/anti** joins when appropriate.  
  **Verify:** Shuffle sizes normal; job completes.

**Quick UI Map (cheat sheet)**

* **Jobs → Runs → View output**: high-level failures, logs link.
* **Spark UI**:
  + **SQL**: actual physical plan, operator times, DPP/broadcast indicators.
  + **Stages**: shuffle/spill, failed tasks, skew/tail.
  + **Executors**: OOM, GC, lost executors, disk.
  + **Storage**: cache usage.
* **Data Explorer (Unity Catalog)**: **History**, **Lineage**, **Permissions**.
* **Compute → Pools/Clusters**: capacity, autoscaling events.
* **SQL Warehouses**: scaling/capacity events.
* **DLT Pipelines**: event log, expectations.