**10. DAG Reliability & Retry Logic (Airflow Production Hardening)**

**Scenario:**  
You’ve built an Airflow DAG to ingest raw files → transform into Delta.  
Now, you must make it **robust against failures**:

* What if a task fails due to network hiccup?
* What if retries reprocess data twice?
* How to avoid DAGs being stuck waiting for a file?

**Baseline Approach (Expected in Interviews):**

1. **Retries & backoff**

* Configure retries for transient errors.
* Use exponential backoff to avoid retry storms.

default\_args = {

"retries": 3,

"retry\_delay": timedelta(minutes=5),

"retry\_exponential\_backoff": True,

"max\_retry\_delay": timedelta(minutes=60)

}

1. **Idempotent tasks**

* All ETL tasks must be re-runnable without causing duplicates.
* Use Delta MERGE for upserts, not INSERT-only.
* External side effects (API calls, audit inserts) must use unique IDs.

1. **Sensors**

* Use reschedule mode for efficiency (mode="reschedule").
* Example: FileSensor waiting for \_READY marker file.

1. **Timeouts**

* Prevent zombie tasks with execution timeouts.

task = PythonOperator(

task\_id="process\_data",

python\_callable=process\_fn,

execution\_timeout=timedelta(hours=2)

)

1. **SLAs & Alerts**

* SLA = “task must finish within 2 hours of schedule”.
* Send alert if missed.

**Advanced considerations:**

* **Retries vs reschedules:**
  + Retry → reruns the whole task (could repeat processing).
  + Reschedule → task just waits and checks later (safe for sensors).
* **Concurrency control:**
  + max\_active\_runs=1 to prevent overlap.
  + Pools/queue to throttle heavy tasks.
* **Partial failures:**
  + Downstream should not run unless upstream succeeded (trigger rules).
  + For optional paths, use trigger\_rule="all\_done".
* **Backfill robustness:**
  + Parameterize DAG (execution\_date) → ensures reprocessing works without duplication.
* **Monitoring:**
  + Track retries in Airflow UI / Prometheus.
  + Add custom logging: row counts, hash of processed files.

**Follow-up Q&A (Interview style)**

**Q1. How do you prevent retries from reprocessing the same data twice?**  
👉 By designing **idempotent tasks**: use MERGE (upsert) instead of INSERT, generate audit keys, and checkpoint streaming jobs. For APIs, send unique request IDs.

**Q2. What’s the difference between retries and reschedules?**  
👉 Retries re-execute the operator code (may duplicate work). Reschedules are used for sensors — they don’t fail, just “sleep and wake up” to check again.

**Q3. How do you detect & avoid zombie tasks?**  
👉 Use execution\_timeout and dagrun\_timeout. Set alerting on tasks running too long.

**Q4. What’s your approach if one task consistently fails after retries?**  
👉 Fail fast and alert. Don’t keep retrying forever. Move bad input to quarantine and continue processing healthy data.

**Q5. How do you handle retries in a DAG with multiple dependent tasks?**  
👉 Retries are task-specific. Downstream waits until upstream succeeds. If upstream keeps failing, DAG fails. For independent branches, use trigger\_rule="all\_done".

**Cheat Sheet (Compressed memory):**

* **Retries:** 2–3, exponential backoff, max delay.
* **Idempotency:** MERGE not INSERT; unique IDs for audit.
* **Sensors:** reschedule mode, avoid busy-wait.
* **Timeouts:** execution\_timeout + dagrun\_timeout.
* **Concurrency:** max\_active\_runs=1, pools for throttling.
* **SLAs:** detect late jobs, alert.
* **Monitoring:** row counts, hashes, retry metrics.

✅ So now you’ve seen:

* **Problem 5:** DAG design & orchestration (pipeline structure).
* **Problem 10:** DAG reliability & retries (pipeline hardening).

They complement each other — architecture vs robustness.