LAB 08: Lakeflow Jobs – Triggers, Dependencies & Orchestration

Duration: ~30 min
Day: 3
After module: M08: Lakeflow Jobs & Orchestration
Difficulty: Intermediate

Scenario

> *"The RetailHub Medallion pipeline (LAB 07) is running successfully on-demand. Now it's time to automate it. The business requires: (1) daily scheduled refresh of the pipeline, (2) a validation task that runs AFTER the pipeline completes, (3) an email alert if any task fails. You will configure a multi-task Job with triggers and dependencies in the Databricks UI, then explore the configuration via notebook code."*

Objectives

After completing this lab you will be able to: - Create a multi-task Lakeflow Job with task dependencies - Configure different trigger types (scheduled/cron, file arrival, continuous) - Define task dependency chains (linear and fan-out/fan-in patterns) - Set up retry policies and failure notifications - Use `dbutils.jobs` to pass parameters between tasks - Query system tables to monitor job runs programmatically

---

Prerequisites

- Completed LAB 07 (Lakeflow Pipeline is created in workspace)
- Access to Databricks workspace with Job creation permissions

---

Part 1: Understanding Job Architecture (~5 min)

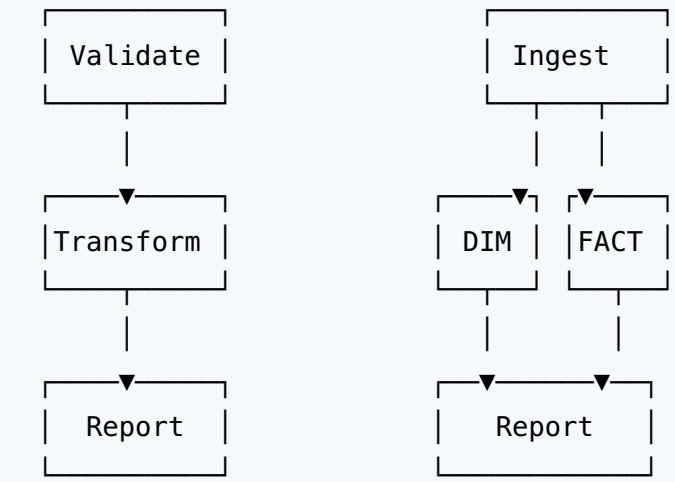
Task 1: Review Job Components

Before creating a Job in the UI, review these key components:

Component	Description	Exam Relevance
Task	A single unit of work (notebook, DLT pipeline, SQL, JAR, Python script)	Know all task types
Dependency	Defines execution order between tasks (task B runs after task A)	DAG structure
Trigger	When the job starts (manual, scheduled, file arrival, continuous)	Trigger types and use cases
Cluster	Each task can use its own cluster or share one (job cluster vs all-purpose)	Cost optimization
Retry Policy	How many times to retry a failed task before marking it as failed	Resilience
Timeout		Resource protection

Component	Description	Exam Relevance
	Maximum duration before a task is killed	

Job Dependency Patterns



> **Exam Tip:** In fan-out/fan-in pattern, the Report task has **two dependencies** (DIM and FACT). It starts only when **both** complete successfully. This is how Databricks handles parallel execution within jobs.

Part 2: Create Multi-Task Job in UI (~10 min)

Task 2: Build the RetailHub Daily Job

Open the Databricks UI and follow these steps:

1. **Navigate:** Workflows > Jobs > Create Job

2. **Task 1 – Pipeline Refresh:**
    - Task name: `refresh_pipeline`
    - Type: **Lakeflow pipeline task**
    - Pipeline: Select your RetailHub pipeline from LAB 07
    - Cluster: Job cluster (Single Node, smallest available)
3. **Task 2 – Validate Results:**
    - Task name: `validate_results`
    - Type: **Notebook task**
    - Notebook: `task_01_validate.ipynb` (from the training setup)
    - **Depends on:** `refresh_pipeline`
    - Cluster: Same job cluster
4. **Task 3 – Generate Report:**
    - Task name: `generate_report`
    - Type: **Notebook task**
    - Notebook: `task_03_report.ipynb`
    - **Depends on:** `validate_results`
    - Cluster: Same job cluster

> **Exam Tip:** A **Job cluster** is created when the Job starts and terminated when it ends. It's cheaper than an all-purpose cluster for scheduled jobs. Multiple tasks can share the same Job cluster.

Part 3: Configure Triggers (~5 min)

Task 3: Set Up Scheduled Trigger

Configure a CRON-based trigger for daily execution:

1. In Job settings, click **Add trigger**
2. Select **Scheduled**

3. Set the CRON expression: `0 6 * * *` (daily at 06:00 UTC)
4. Timezone: Select your timezone

Common CRON patterns for reference:

Pattern	CRON Expression	Use Case
Every day at 6 AM	`0 6 * * *`	Daily batch refresh
Every hour	`0 * * * *`	Near-real-time updates
Mon-Fri at 8 AM	`0 8 * * 1-5`	Business day reports
Every 15 min	`*/15 * * * *`	Frequent incremental loads
First day of month	`0 0 1 * *`	Monthly aggregations

Task 4: Explore Other Trigger Types

Review (do NOT configure) these additional trigger types:

Trigger Type	When to Use	Configuration
Manual	Ad-hoc runs, testing	Default, no trigger needed
Scheduled (CRON)	Regular batch processing	CRON expression + timezone
File arrival	Event-driven: run when new files land in a Volume/cloud path	Path to monitor + wait time
Continuous	Always-on processing, minimal latency	Restart on completion, with optional pause

> **Exam Tip: File arrival trigger** monitors a cloud storage path. When new files appear, the job starts. This is ideal for event-driven architectures. The trigger checks for files at configurable intervals

> (default: 5 min). Continuous trigger restarts the job immediately after completion – used for streaming-like behavior with notebook tasks.

Part 4: Dependencies & Error Handling (~5 min)

Task 5: Configure Retry and Alerting

In the Job settings:

1. **Retry Policy:**
   - For `validate_results` task: Set max retries to **2**, retry interval **60 seconds**
   - Rationale: Validation may fail due to transient Delta table issues
2. **Timeout:**
   - Set `refresh_pipeline` timeout to **30 minutes**
   - If the pipeline doesn't finish in 30 min, something is wrong
3. **Notifications (Email/Webhook):**
   - Add notification on **Failure**: Your email
   - Add notification on **Success**: Optional (for monitoring)

Task 6: Test Run and Repair

1. **Run the Job manually** (click "Run now")
2. **Observe the DAG** in the Runs tab – watch tasks execute in dependency order
3. If a task fails:
   - Click the failed run
   - Click **Repair run** – this re-runs ONLY the failed task and downstream dependencies
   - Saves time vs. re-running the entire job

> **Exam Tip: Repair run** is a key exam concept. It re-executes only failed and downstream tasks, preserving successful results. This is critical for long-running pipelines where re-running everything would be wasteful.

Part 5: Programmatic Job Monitoring (~5 min)

Task 7: Query System Tables

Open `LAB_08_code.ipynb` and complete the tasks there. You will: - Query `system.lakeflow.job_run_timeline` to see job execution history - Query `system.lakeflow.job_task_run_timeline` for task-level details - Analyze job duration trends and failure rates - Use `dbutils.jobs.taskValues` to pass data between tasks

---

Summary

After completing this lab you have: - Created a multi-task Job with linear task dependencies - Configured a CRON scheduled trigger for daily execution - Understood file arrival and continuous trigger types - Set up retry policies and email notifications - Used Repair Run to re-execute failed tasks efficiently - Queried system tables for job monitoring

---

What’s next: LAB 09

Next you will apply governance and security controls: permissions, row filters, column masking, and Delta Sharing.

Instructions

Open **`LAB_08_code.ipynb`** and complete all `# TODO` cells.
Each task has an `assert` cell to verify your solution.