# LAB 08: Lakeflow Jobs — Triggers, Dependencies & Orchestration

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

---

## Scenario

> *"Orchestrate your Medallion pipeline by creating multi-task Databricks Jobs with dependencies, triggers, and monitoring. First build a pipeline in the UI (Workshop), then solve orchestration concept questions (Practice)."*

---

## Objectives

After completing this lab you will be able to: - Create and configure multi-task Jobs in Databricks UI - Set task dependencies (DAG structure) - Configure Table Update Triggers for event-driven orchestration - Understand Job configuration JSON - Write CRON expressions for scheduling - Design DAGs with fan-out/fan-in patterns - Use Repair Runs for partial re-execution - Choose between Job Clusters and All-Purpose Clusters

---

## Prerequisites

- Cluster running and attached to notebook

- Medallion notebooks from `materials/medallion/` available

---

# Section 1: Workshop — Creating & Running Jobs in Databricks UI

Follow the trainer's instructions step by step.

## Step 1: Create a New Job

- Go to **Workflows** → **Jobs** → **Create Job**

## Step 2: Set the Job Name

- Name it `<your_name>_customer_pipeline`

## Step 3: Configure Job Parameters

- `catalog` = `retailhub_<your_name>`
- `source_path` = `/Volumes/retailhub_<your_name>/default/datasets`

## Step 4: Add First Task — `bronze_customer`

- Type: **Notebook** task
- Source: `materials/medallion/bronze_customers`
- No dependencies (first task)

## Step 5: Add Second Task — `silver_customer`

- Type: **Notebook** task
- Source: `materials/medallion/silver_customers`
- **Depends on:** `bronze_customer`

### Step 6: Create Second Job — `orders_pipeline`

- 4 tasks with dependencies:
  - `bronze_orders` → `silver_orders` → `gold_daily_orders` → `gold_summary`

### Step 7: Configure Table Update Trigger

- Set `orders_pipeline` to trigger automatically when `silver_customers` table is updated

### Step 8: Run the Pipeline

1. Run `customer_pipeline` first
2. `orders_pipeline` should trigger automatically via Table Update Trigger
3. Monitor execution in the **Runs** tab

---

## Section 2: Practice — Orchestration Concepts

Open **`LAB_08_code.ipynb`** and complete the `# TODO` cells.

| Task | What to do | Key concept |
|---|---|---|
| **Task 1** | Job Configuration JSON | Read config and extract: num_tasks, first_task, timeout, retries |
| **Task 2** | Trigger Types | Match scenarios to triggers: scheduled, file_arrival, continuous, manual |
| **Task 3** | CRON Expressions | Write CRON for: daily 6AM, hourly, weekdays 8AM, every 15min |
| **Task 4** | DAG Design | Define task dependencies (fan-out / fan-in pattern) |
| **Task 5** | Repair Runs | Identify which tasks re-run after `build_fact_tables` failure |
| **Task 6** | Task Values | |

| Task | What to do | Key concept |
|---|---|---|
| | | `dbutils.jobs.taskValues` — set and get values between tasks |
| **Task 7** | System Tables SQL | Query `system.lakeflow.job_run_timeline` for FAILED runs |
| **Task 8** | Cluster Selection | Choose job_cluster vs all_purpose for different scenarios |

---

## Detailed Hints

### Task 1: Job Config JSON

- Count the tasks in the `tasks` array
- First task = task with empty `depends_on` list
- Timeout and retries are in the task-level configuration

### Task 2: Trigger Types

- `scheduled` — runs on a CRON schedule
- `file_arrival` — triggers when new files land in a path
- `continuous` — runs continuously (streaming)
- `manual` — ad hoc, run by user

### Task 3: CRON

- Format: `minute hour day_of_month month day_of_week`
- `0 6 * * *` = daily at 6:00 AM
- `0 * * * *` = every hour at :00
- `1-5` = Monday through Friday
- `*/15` = every 15 minutes

### Task 4: DAG Dependencies

- `ingest` has no dependencies (empty list)
- `build_dim_tables` and `build_fact_tables` both depend on `ingest` (fan-out)
- `generate_report` depends on both dim and fact (fan-in)
- `send_notification` depends on `generate_report`

### Task 5: Repair Runs

- Repair Run re-executes the failed task and all downstream tasks
- Successful upstream tasks (ingest, dim) are NOT re-run → saves cost

### Task 6: Task Values

- `dbutils.jobs.taskValues.set(key, value)` in source task
- `dbutils.jobs.taskValues.get(taskName, key)` in downstream task

### Task 7: System Tables

- Table: `system.lakeflow.job_run_timeline`
- Filter: `result_state = 'FAILED'`
- Date filter: `start_time >= current_date() - INTERVAL 7 DAYS`

### Task 8: Cluster Selection

- **Job cluster**: cheaper, auto-terminates, for production ETL + ML training
- **All-purpose cluster**: interactive, for development + ad-hoc analysis

---

## Summary

In this lab you: - Built multi-task Jobs with dependencies in the Databricks UI - Configured Table Update Triggers for event-driven orchestration - Practiced CRON expressions and trigger types - Designed DAGs with fan-out/fan-in

patterns - Explored Repair Runs for cost-efficient re-execution - Used Task Values for inter-task communication - Queried System Tables for job monitoring

> **Exam Tip:** Repair Runs only re-execute the failed task and its downstream dependencies — upstream tasks are skipped. Job Clusters are cheaper for production. `system.lakeflow.job_run_timeline` is the key system table for job monitoring.

> **What's next:** In LAB 09 you will implement governance controls — column masks, row filters, and privilege management with Unity Catalog.