|  |  | |
| --- | --- | --- |

🛫 Flight Delays — End-to-End (Git + Docker Spark + MySQL + QA)

I keep **Activities under C:\Git\…** and **Tools under C:\Tools\…**.  
Everything below is **copy-paste-ready**, first-person, small diffs.

# 🎯 Problem Statement & Objective

☐ **What:** Stand up **Spark (PySpark)** + **MySQL** in Docker (project name **FlightDelays**), load departuredelays.csv, persist to MySQL, and publish **QA test scripts** (PySpark + SQL) in a GitHub repo **DepartureDelays**.

☐ **Why:** Prove **ingest → query → persist → validate**, with a repo and reproducible containers for the team.

# 📜 0) File Download (when the site needs auth cookies)

### Step 1: Get cURL & Session.Cookies

| How I get the cURL even when the CSV opens in a new tab  1) Start from the course page that contains the Departure Delays link   * Open the page that shows the “Video 0.31…” text and the **Departure Delays** link. * Make sure you’re already signed in (Google SSO) and the link opens for you.   2) Open Chrome DevTools on that page   * Press **F12** (or **Ctrl+Shift+I**). * Go to the **Network** tab. * Click the **🧹 Clear** button once so we start with a clean list.   3) Click the Departure Delays link (CSV opens in a new tab)   * The CSV opens in a new browser tab (viewer/spreadsheet UI).   4) Confirm you’ve selected the actual CSV request   * Click the **/download** row. * In **Headers → Response Headers** you should see:    + Content-Type: text/csv   + Content-Disposition: inline; filename="departuredelays.csv" (or attachment; filename=...)      * In **Preview/Response**, the content starts with the CSV header:  date,delay,distance,origin,destination.   If those two are present, you’ve got the **real CSV** request (not a script or icon).   * What Chrome just did:   + It took the exact request you made **while you were logged in**.   + It exported it as a ready-to-run PowerShell script that creates a  Microsoft.PowerShell.Commands.WebRequestSession.   + It **serialized your current session cookies** into lines like  Session.Cookies.Add((New-Object System.Net.Cookie("…", "…", "/", "classroom.emeritus.org"))).   + Those cookie lines are the **reason your PowerShell call stays authenticated**. |
| --- |
|  |
|  |

Copy as Bash cURL

| 5) Copy the fully-formed request as a Copy as cURL (bash) script   * Right-click that **/download** row → **Copy → Copy as cURL (bash)** |
| --- |
| curl 'https://classroom.emeritus.org/courses/12271/files/3904250/download' \  -H 'accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7' \  -H 'accept-language: en-US,en;q=0.9,hi;q=0.8' \  -H 'cache-control: max-age=0' \  -b '\_fbp=fb.1.1761277618656.223950538269579280; log\_session\_id=0558af137acb74e1e247b7e2ef375cf3; \_clck=n026bf%5E2%5Eg0j%5E0%5E2123; \_clsk=1i5uvoc%5E1761634095333%5E1%5E1%5Ei.clarity.ms%2Fcollect; lux\_uid=176169832985493108; \_legacy\_normandy\_session=dFrwT2lTavSgvfpLU5N6NQ+ENROXkfACJsMcg3bLnD6DrWuBOAEr\_aVqn4-3qVkJF4TR62eWefM33ldLGpg2tMcVovxAUHZxeExNa4FuziR\_\_ekz1Ma4H-MNDWwK5JbJ5-BatePS7HAtXi9KDubYNEnZpRVVqQ1Ez4q18vrBppSv-SCboESfxgrT-MvnzG7L-395TPTVPhYn3S-no4zuQabjA2GM5CqVwujimv1NVHKXXPBGhfwD2kw7g9mVblZZYhCbD6TdQ1EsrPnXfqxn7HzS6WndsFpeHz6hlW6XBMHp38jjh0ClrN5qwt0\_QZ\_0Yz3oqCuHfwWvfh7toyzDxGTtp4O41dOYA2S2cBCCjt2WqcA0V5OXbKzcTpF1Tus18utmdlkUS6SO21okxKEn0fF1tTm0QKHly\_27DegBpCMt3I3PNVXQsCklaFW4ou5IJlmvdCSDOs-JKwbma3cCWgUian2Uia2lBUXM0s4mxkKmXgx48Gr3a38G7IanxGd7FUlZqGBjJ2RCogZyHpW56L0a1rfa3aEgodGhK5LEKFgfWFxKEjkZFuxQDiifU5W-AAqCoLWZbYPM5LHaYJNpzm7aHJ1A7edruJadTY8b2XZcQ.AmPFJtLl4eW4e5yBFIgz5qnEvYc.aQFjqA; \_normandy\_session=dFrwT2lTavSgvfpLU5N6NQ+ENROXkfACJsMcg3bLnD6DrWuBOAEr\_aVqn4-3qVkJF4TR62eWefM33ldLGpg2tMcVovxAUHZxeExNa4FuziR\_\_ekz1Ma4H-MNDWwK5JbJ5-BatePS7HAtXi9KDubYNEnZpRVVqQ1Ez4q18vrBppSv-SCboESfxgrT-MvnzG7L-395TPTVPhYn3S-no4zuQabjA2GM5CqVwujimv1NVHKXXPBGhfwD2kw7g9mVblZZYhCbD6TdQ1EsrPnXfqxn7HzS6WndsFpeHz6hlW6XBMHp38jjh0ClrN5qwt0\_QZ\_0Yz3oqCuHfwWvfh7toyzDxGTtp4O41dOYA2S2cBCCjt2WqcA0V5OXbKzcTpF1Tus18utmdlkUS6SO21okxKEn0fF1tTm0QKHly\_27DegBpCMt3I3PNVXQsCklaFW4ou5IJlmvdCSDOs-JKwbma3cCWgUian2Uia2lBUXM0s4mxkKmXgx48Gr3a38G7IanxGd7FUlZqGBjJ2RCogZyHpW56L0a1rfa3aEgodGhK5LEKFgfWFxKEjkZFuxQDiifU5W-AAqCoLWZbYPM5LHaYJNpzm7aHJ1A7edruJadTY8b2XZcQ.AmPFJtLl4eW4e5yBFIgz5qnEvYc.aQFjqA; \_csrf\_token=gt8sARKFGMemU0Jni07HT5v0K1M%2BMvsJrpMNmjJsVSP3q0V7XLJPpNYVdiXPHbUc6YcdIwdj1Hr31kDgdx8%2BUw%3D%3D' \  -H 'dnt: 1' \  -H 'priority: u=0, i' \  -H 'referer: https://classroom.emeritus.org/courses/12271/pages/video-0-dot-31-using-pyspark-to-query-flight-data?module\_item\_id=2207404' \  -H 'sec-ch-ua: "Google Chrome";v="141", "Not?A\_Brand";v="8", "Chromium";v="141"' \  -H 'sec-ch-ua-mobile: ?0' \  -H 'sec-ch-ua-platform: "Windows"' \  -H 'sec-fetch-dest: document' \  -H 'sec-fetch-mode: navigate' \  -H 'sec-fetch-site: same-origin' \  -H 'sec-fetch-user: ?1' \  -H 'upgrade-insecure-requests: 1' \  -H 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36' |

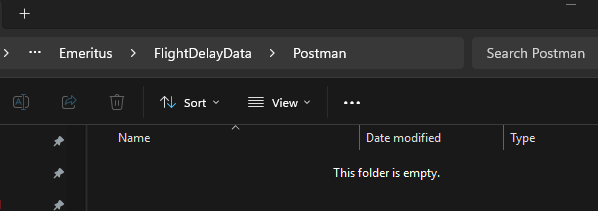
**–OR–**

Copy as Powershell

| 5) Copy the fully-formed request as a PowerShell script (with cookies)   * Right-click that **/download** row → **Copy → Copy as PowerShell** |
| --- |
| $session = New-Object Microsoft.PowerShell.Commands.WebRequestSession  $session.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36"  $session.Cookies.Add((New-Object System.Net.Cookie("\_fbp", "fb.1.1761277618656.223950538269579280", "/", ".emeritus.org")))  $session.Cookies.Add((New-Object System.Net.Cookie("log\_session\_id", "0558af137acb74e1e247b7e2ef375cf3", "/", "classroom.emeritus.org")))  $session.Cookies.Add((New-Object System.Net.Cookie("\_clck", "n026bf%5E2%5Eg0j%5E0%5E2123", "/", ".emeritus.org")))  $session.Cookies.Add((New-Object System.Net.Cookie("\_clsk", "1i5uvoc%5E1761634095333%5E1%5E1%5Ei.clarity.ms%2Fcollect", "/", ".emeritus.org")))  $session.Cookies.Add((New-Object System.Net.Cookie("lux\_uid", "176169832985493108", "/", "classroom.emeritus.org")))  $session.Cookies.Add((New-Object System.Net.Cookie("\_legacy\_normandy\_session", "dFrwT2lTavSgvfpLU5N6NQ+ENROXkfACJsMcg3bLnD6DrWuBOAEr\_aVqn4-3qVkJF4TR62eWefM33ldLGpg2tMcVovxAUHZxeExNa4FuziR\_\_ekz1Ma4H-MNDWwK5JbJ5-BatePS7HAtXi9KDubYNEnZpRVVqQ1Ez4q18vrBppSv-SCboESfxgrT-MvnzG7L-395TPTVPhYn3S-no4zuQabjA2GM5CqVwujimv1NVHKXXPBGhfwD2kw7g9mVblZZYhCbD6TdQ1EsrPnXfqxn7HzS6WndsFpeHz6hlW6XBMHp38jjh0ClrN5qwt0\_QZ\_0Yz3oqCuHfwWvfh7toyzDxGTtp4O41dOYA2S2cBCCjt2WqcA0V5OXbKzcTpF1Tus18utmdlkUS6SO21okxKEn0fF1tTm0QKHly\_27DegBpCMt3I3PNVXQsCklaFW4ou5IJlmvdCSDOs-JKwbma3cCWgUian2Uia2lBUXM0s4mxkKmXgx48Gr3a38G7IanxGd7FUlZqGBjJ2RCogZyHpW56L0a1rfa3aEgodGhK5LEKFgfWFxKEjkZFuxQDiifU5W-AAqCoLWZbYPM5LHaYJNpzm7aHJ1A7edruJadTY8b2XZcQ.AmPFJtLl4eW4e5yBFIgz5qnEvYc.aQFjqA", "/", "classroom.emeritus.org")))  $session.Cookies.Add((New-Object System.Net.Cookie("\_normandy\_session", "dFrwT2lTavSgvfpLU5N6NQ+ENROXkfACJsMcg3bLnD6DrWuBOAEr\_aVqn4-3qVkJF4TR62eWefM33ldLGpg2tMcVovxAUHZxeExNa4FuziR\_\_ekz1Ma4H-MNDWwK5JbJ5-BatePS7HAtXi9KDubYNEnZpRVVqQ1Ez4q18vrBppSv-SCboESfxgrT-MvnzG7L-395TPTVPhYn3S-no4zuQabjA2GM5CqVwujimv1NVHKXXPBGhfwD2kw7g9mVblZZYhCbD6TdQ1EsrPnXfqxn7HzS6WndsFpeHz6hlW6XBMHp38jjh0ClrN5qwt0\_QZ\_0Yz3oqCuHfwWvfh7toyzDxGTtp4O41dOYA2S2cBCCjt2WqcA0V5OXbKzcTpF1Tus18utmdlkUS6SO21okxKEn0fF1tTm0QKHly\_27DegBpCMt3I3PNVXQsCklaFW4ou5IJlmvdCSDOs-JKwbma3cCWgUian2Uia2lBUXM0s4mxkKmXgx48Gr3a38G7IanxGd7FUlZqGBjJ2RCogZyHpW56L0a1rfa3aEgodGhK5LEKFgfWFxKEjkZFuxQDiifU5W-AAqCoLWZbYPM5LHaYJNpzm7aHJ1A7edruJadTY8b2XZcQ.AmPFJtLl4eW4e5yBFIgz5qnEvYc.aQFjqA", "/", "classroom.emeritus.org")))  $session.Cookies.Add((New-Object System.Net.Cookie("\_csrf\_token", "gt8sARKFGMemU0Jni07HT5v0K1M%2BMvsJrpMNmjJsVSP3q0V7XLJPpNYVdiXPHbUc6YcdIwdj1Hr31kDgdx8%2BUw%3D%3D", "/", "classroom.emeritus.org")))  Invoke-WebRequest -UseBasicParsing -Uri "https://classroom.emeritus.org/courses/12271/files/3904250/download" `  -WebSession $session `  -Headers @{  "authority"="classroom.emeritus.org"  "method"="GET"  "path"="/courses/12271/files/3904250/download"  "scheme"="https"  "accept"="text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7"  "accept-encoding"="gzip, deflate, br, zstd"  "accept-language"="en-US,en;q=0.9,hi;q=0.8"  "cache-control"="max-age=0"  "dnt"="1"  "priority"="u=0, i"  "referer"="https://classroom.emeritus.org/courses/12271/pages/video-0-dot-31-using-pyspark-to-query-flight-data?module\_item\_id=2207404"  "sec-ch-ua"="`"Google Chrome`";v=`"141`", `"Not?A\_Brand`";v=`"8`", `"Chromium`";v=`"141`""  "sec-ch-ua-mobile"="?0"  "sec-ch-ua-platform"="`"Windows`""  "sec-fetch-dest"="document"  "sec-fetch-mode"="navigate"  "sec-fetch-site"="same-origin"  "sec-fetch-user"="?1"  "upgrade-insecure-requests"="1"  } |

### Step 2: Download file using Postman

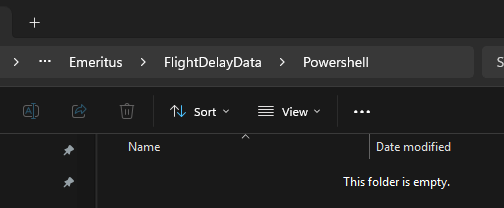
C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Postman



| **Postman (still works without Interceptor)**  1. In Chrome DevTools, on the **download** row (the one whose Headers show Content-Type: text/csv and Content-Disposition: filename="departuredelays.csv"), right-click → **Copy → Copy as PowerShell**. 2. In Postman: **Import → Raw text** → paste → **Import**.   cURL from Bash works   1. Open the imported request → **Settings** tab → ensure **Follow redirects = ON**. 2. Click **Cookies** (top right) and confirm classroom.emeritus.org has cookies (Postman will import them from the script). 3. Click **▼ Send → Send and Download**, save to C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Postman |
| --- |

### Step 3: Download file using Powershell

C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Powershell



| **Do this (PowerShell-only, copy-paste)**   # 1) Target path (your standard downloads folder)  $TARGET = "C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Powershell\departuredelays.csv"    # 2) Session from DevTools "Copy as PowerShell" (you already have this block)  $session = New-Object Microsoft.PowerShell.Commands.WebRequestSession  $session.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36"  # ... keep all the Cookies.Add(...) lines exactly as Chrome gave you ...    # 3) Download the file (key bits: Accept header, Follow redirects, OutFile)  Invoke-WebRequest `  -Uri "https://classroom.emeritus.org/courses/12271/files/3904250/download" `  -WebSession $session `  -Headers @{  "Accept" = "text/csv,\*/\*"  "Referer" = "https://classroom.emeritus.org/courses/12271/pages/video-0-dot-31-using-pyspark-to-query-flight-data?module\_item\_id=2207404"  } `  -MaximumRedirection 10 `  -OutFile $TARGET    # 4) Quick verify  Get-Item $TARGET | Select-Object Name, Length  Get-Content $TARGET -TotalCount 3 |
| --- |
| **Why this works:**   * -OutFile writes bytes to disk (otherwise you’ll end up with an HTML preview saved as response.html). * Accept: text/csv,\*/\* nudges the server to send the CSV directly. * -MaximumRedirection allows the login-protected download hop. * Your copied **cookies** keep you authenticated. |

# 🧱 1) Folder & File Setup (Activity under C:\Git, Tools already under C:\Tools)

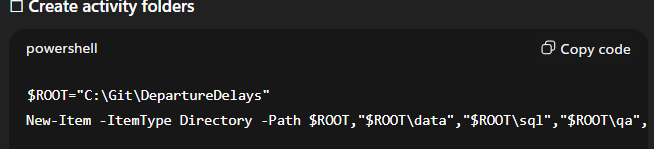
☐ **Roots I use**

* + C:\Git\DepartureDelays ← activity project
  + C:\Tools\… ← (already have Docker Desktop, MySQL shell, etc.)

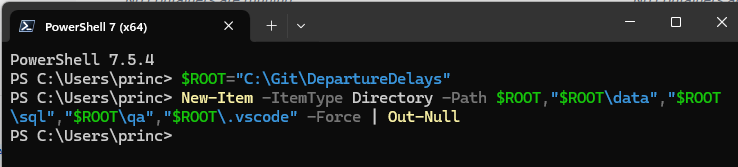
☐ **Create activity folders**

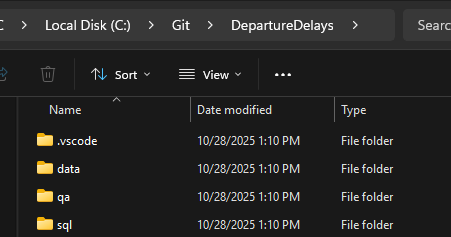
* I’m setting my **project home** to C:\Git\DepartureDelays so everything for this activity stays in one place.
* The first line saves that path in a PowerShell variable called **$ROOT** (so I can reuse it).
* The second line **creates folders**: the main project plus **data** (CSV files), **sql** (database scripts), **qa** (tests), and **.vscode** (editor settings).
* The **-Force** flag makes the command safe to re-run—it won’t error if folders already exist.
* **Out-Null** just hides the “created directory” messages to keep my console clean.

Create DepartureDelays

  
$ROOT="C:\Git\DepartureDelays"

New-Item -ItemType Directory -Path $ROOT,"$ROOT\data","$ROOT\sql","$ROOT\qa","$ROOT\.vscode" -Force | Out-Null

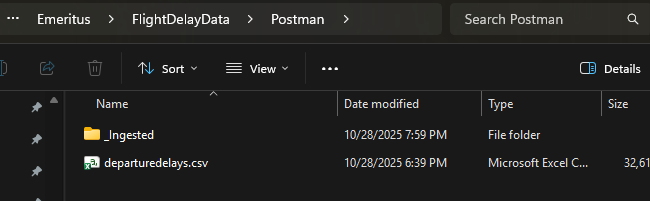


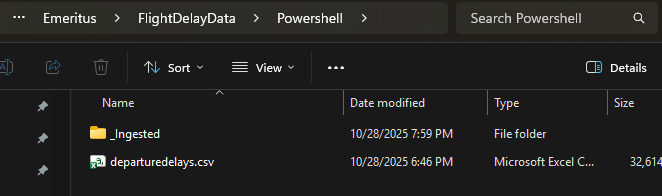


**☐ Create \_Ingested**

Creates C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Postman\\_Ingested if it doesn’t exist.

| # ---- OPTION A:  **Recommended (keeps $Postman & $Powershell as base folders):**    # base folders  $Postman = "C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Postman"  $Powershell = "C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Powershell"  # ingested subfolders  $PostmanIngested = Join-Path $Postman "\_Ingested"  $PowershellIngested = Join-Path $Powershell "\_Ingested"  # create all (idempotent)  New-Item -ItemType Directory -Path $Postman, $Powershell, $PostmanIngested, $PowershellIngested -Force | Out-Null |
| --- |
| # ---- OPTION B:  **If you really want your two variables to point directly to \_Ingested (works, just less flexible):**    $Postman = "C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Postman\\_Ingested"  $Powershell = "C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Powershell\\_Ingested"  New-Item -ItemType Directory -Path $Postman, $Powershell -Force | Out-Null |





**☐ Copy CSV into the project**

PowerShell snippet that does exactly that (creates \_Ingested, copies the CSV into your project, then moves the original into \_Ingested with a timestamped name).

All of the following in same Powershell

| # === Paths as Variable ===      $srcRoot = "C:\Git\\_3rdPartyDownloadFiles\Emeritus\FlightDelayData\Postman"  $dstRoot = "C:\Git\DepartureDelays\data"  $ingested = Join-Path $srcRoot "\_Ingested" |
| --- |
| **Process Single / All Files**  Copies departuredelays.csv to C:\Git\DepartureDelays\data.  Moves the original into \_Ingested as departuredelays\_YYYYMMDD\_HHMMSS.csv so you can track exactly when it was ingested.   | # ---- OPTION A: single specific file ----  # --- File to process ---  $fileName = "departuredelays.csv"      # Get the file (set $f so it isn't $null)  $f = Get-Item -LiteralPath (Join-Path $srcRoot $fileName) -ErrorAction SilentlyContinue  if ($null -eq $f) {  Write-Error "File not found: $(Join-Path $srcRoot $fileName)"  return  }  # 1) COPY to project data folder  $dst = Join-Path $dstRoot $f.Name  Copy-Item -LiteralPath $f.FullName -Destination $dst -Force      # 2) MOVE original into \_Ingested with timestamped name  $ts = (Get-Date).ToString("yyyyMMdd\_HHmmss")  $newName = "{0}\_{1}{2}" -f [IO.Path]::GetFileNameWithoutExtension($f.Name), $ts, $f.Extension  $ingestedPath = Join-Path $ingested $newName  Move-Item -LiteralPath $f.FullName -Destination $ingestedPath -Force  Write-Host "Processed: $($f.Name)" -ForegroundColor Green  Write-Host " → Copied to: $dst"  Write-Host " → Moved to: $ingestedPath" | | --- | | **– OR –** | | The “Option B” comment lets you switch to processing **all** CSVs in the Postman folder if needed.  # ---- OPTION B: all CSVs in the folder (uncomment next line and comment out the line above) ----    # Get ALL CSVs in the source folder  $files = Get-ChildItem -Path $srcRoot -Filter \*.csv -File  if (-not $files) {  Write-Host "No CSV files found in $srcRoot" -ForegroundColor Yellow  return  }  foreach ($f in $files) {  try {  # 1) COPY to project data folder  $dst = Join-Path $dstRoot $f.Name  Copy-Item -LiteralPath $f.FullName -Destination $dst -Force  # 2) MOVE original to \_Ingested with a unique timestamp  $ts = Get-Date -Format "yyyyMMdd\_HHmmssfff" # includes milliseconds  $newName = '{0}\_{1}{2}' -f [IO.Path]::GetFileNameWithoutExtension($f.Name), $ts, $f.Extension  $ingestedPath = Join-Path $ingested $newName  Move-Item -LiteralPath $f.FullName -Destination $ingestedPath -Force  Write-Host "Processed: $($f.Name)" -ForegroundColor Green  Write-Host " → Copied to: $dst"  Write-Host " → Moved to : $ingestedPath"  }  catch {  Write-Host "Failed: $($f.Name) — $($\_.Exception.Message)" -ForegroundColor Red  }  } | |
| # Quick verification    Write-Host "`nContents of $dstRoot:" -ForegroundColor Cyan  Get-ChildItem $dstRoot -Filter \*.csv | Select-Object Name, Length, LastWriteTime | Format-Table  Write-Host "`nIngested files:" -ForegroundColor Cyan  Get-ChildItem $ingested -Filter \*.csv | Select-Object Name, LastWriteTime | Sort-Object LastWriteTime -Descending | Format-Table |

☐ **Add placeholders (we’ll fill next)**

* + C:\Git\DepartureDelays\docker-compose.yaml
  + C:\Git\DepartureDelays\sql\01\_init\_db.sql
  + C:\Git\DepartureDelays\qa\tests\_pyspark.py
  + C:\Git\DepartureDelays\qa\tests\_sql.sql
  + C:\Git\DepartureDelays\.gitignore
  + C:\Git\DepartureDelays\README.md

# 🐳 2) Containers — FlightDelays (Spark + MySQL, professional names)

I keep a single Compose.

Project name: **flightdelays\_stack**;

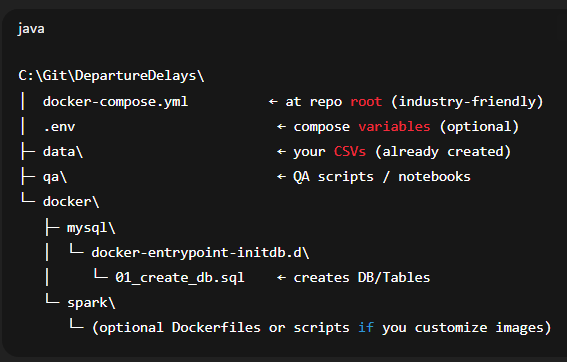
containers: **flight\_spark\_master**, **flight\_spark\_worker\_1**, **flight\_mysql**.

**☐ What’s standard?**

* **Keep docker-compose.yml at the repo root** (most teams expect docker compose up to work from the project top-level).
* Put build files, init SQL, and helper scripts under a **/docker/** subfolder.
* Data you want to mount stays in existing project folders (e.g., data/, qa/).

This gives short, memorable commands and a tidy repo.

**☐ Recommended layout**

****

C:\Git\DepartureDelays\

│ docker-compose.yml ← at repo root (industry-friendly)

│ .env ← compose variables (optional)

├─ data\ ← your CSVs (already created)

├─ qa\ ← QA scripts / notebooks

└─ docker\

├─ mysql\

│ └─ docker-entrypoint-initdb.d\

│ └─ 01\_create\_db.sql ← creates DB/Tables

└─ spark\

└─ (optional Dockerfiles or scripts if you customize images)

**☐ Create folders (copy-paste PowerShell)**

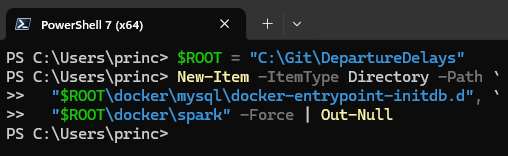


$ROOT = "C:\Git\DepartureDelays"

New-Item -ItemType Directory -Path `

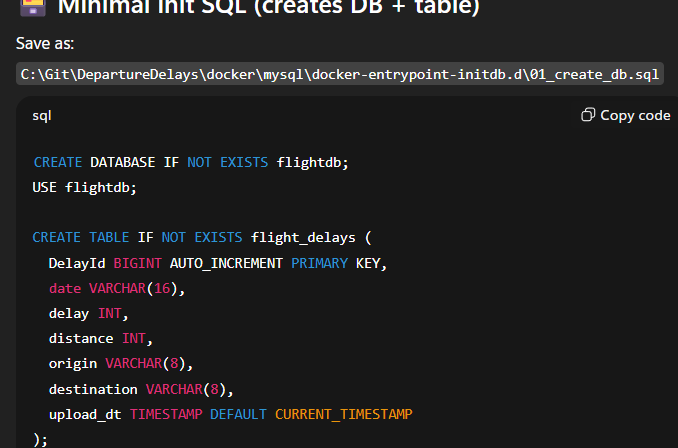
"$ROOT\docker\mysql\docker-entrypoint-initdb.d", `

"$ROOT\docker\spark" -Force | Out-Null



**☐ Minimal init SQL (creates DB + table)**

Save as: C:\Git\DepartureDelays\docker\mysql\docker-entrypoint-initdb.d\01\_create\_db.sql



CREATE DATABASE IF NOT EXISTS flightdb;

USE flightdb;

CREATE TABLE IF NOT EXISTS flight\_delays (

DelayId BIGINT AUTO\_INCREMENT PRIMARY KEY,

date VARCHAR(16),

delay INT,

distance INT,

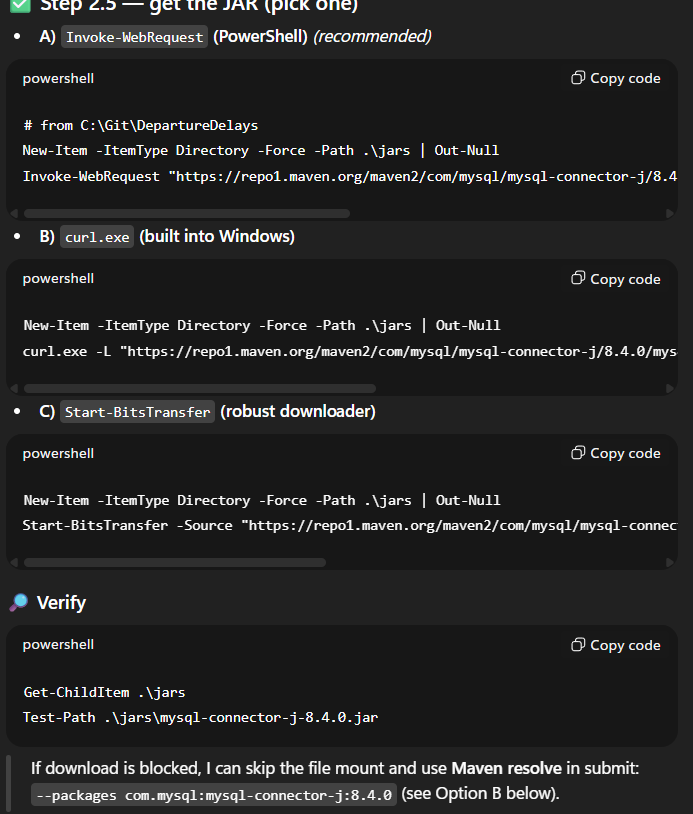
origin VARCHAR(8),

destination VARCHAR(8),

upload\_dt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**☐ Download the MySQL driver JAR (PowerShell)**



**A) Invoke-WebRequest (PowerShell)** *(recommended)*

# from C:\Git\DepartureDelays

New-Item -ItemType Directory -Force -Path .\jars | Out-Null

Invoke-WebRequest "https://repo1.maven.org/maven2/com/mysql/mysql-connector-j/8.4.0/mysql-connector-j-8.4.0.jar" -OutFile ".\jars\mysql-connector-j-8.4.0.jar"

**B) curl.exe (built into Windows)**

New-Item -ItemType Directory -Force -Path .\jars | Out-Null

curl.exe -L "https://repo1.maven.org/maven2/com/mysql/mysql-connector-j/8.4.0/mysql-connector-j-8.4.0.jar" -o ".\jars\mysql-connector-j-8.4.0.jar"

**C) Start-BitsTransfer (robust downloader)**

New-Item -ItemType Directory -Force -Path .\jars | Out-Null

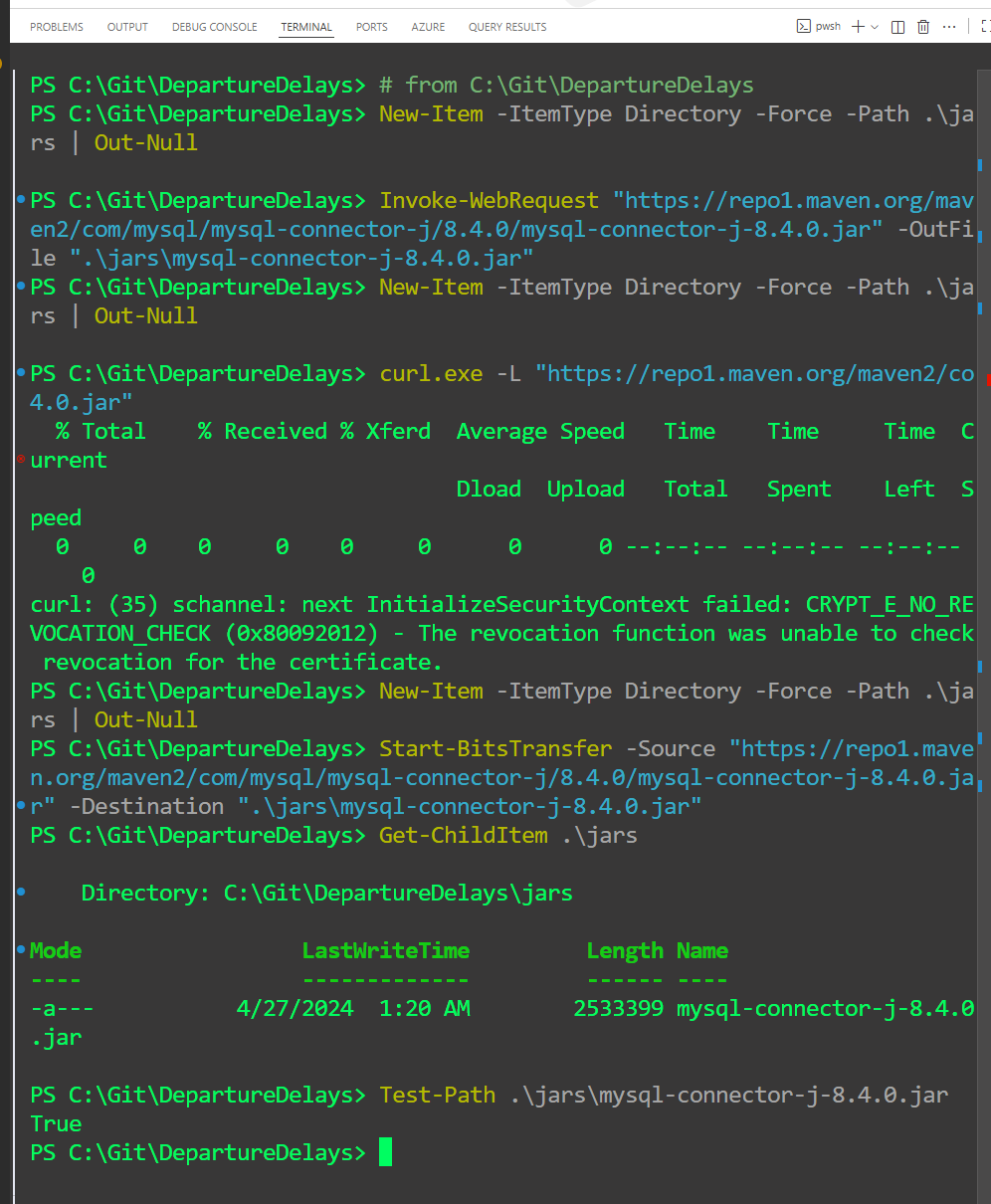
Start-BitsTransfer -Source "https://repo1.maven.org/maven2/com/mysql/mysql-connector-j/8.4.0/mysql-connector-j-8.4.0.jar" -Destination ".\jars\mysql-connector-j-8.4.0.jar"

**🔎 Verify**

Get-ChildItem .\jars

Test-Path .\jars\mysql-connector-j-8.4.0.jar

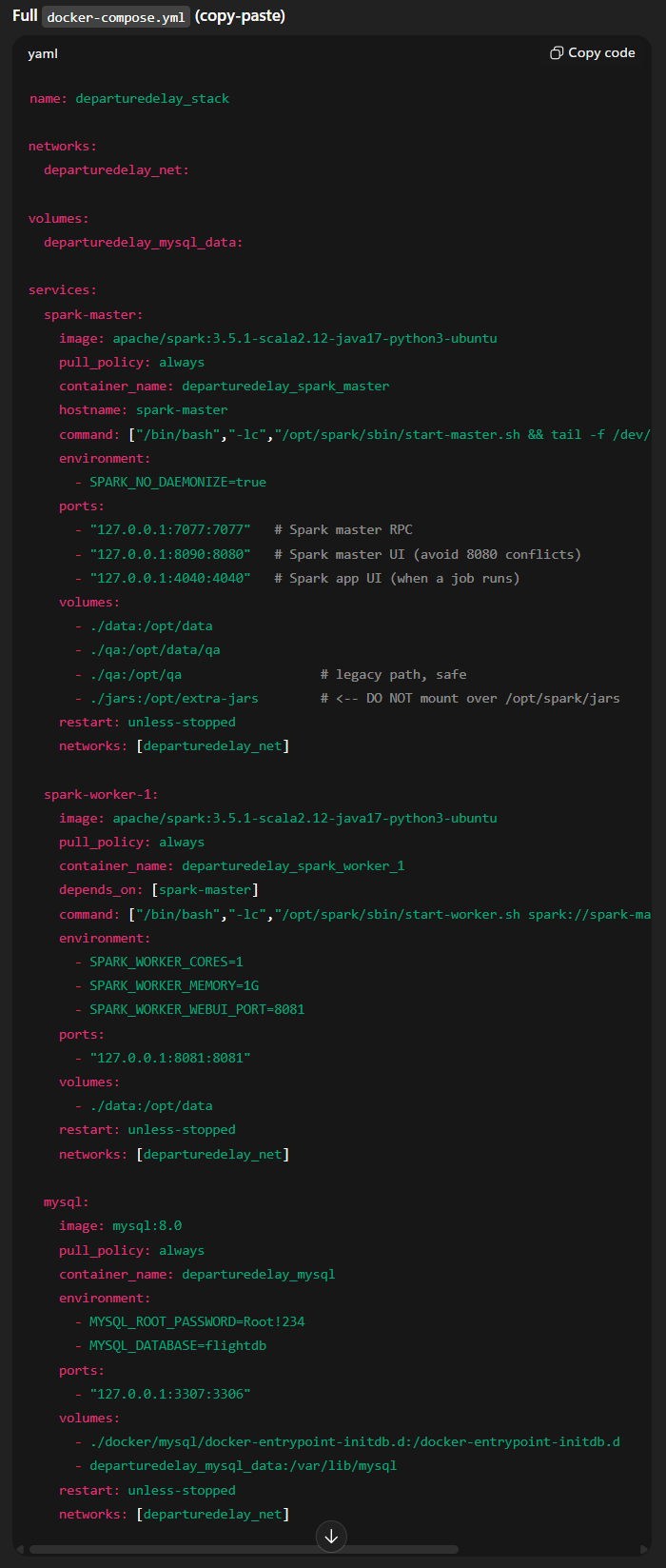
If download is blocked, I can skip the file mount and use **Maven resolve** in submit:  
 --packages com.mysql:mysql-connector-j:8.4.0 (see Option B below).



### 🐳 docker-compose.yaml (root)

**☐ C:\Git\DepartureDelays\docker-compose.yaml (**continuous running containers**)**

Use the service name spark-master (Docker’s built-in DNS) and pin the worker UI port.



name: departuredelay\_stack

networks:

departuredelay\_net:

volumes:

departuredelay\_mysql\_data:

services:

spark-master:

image: apache/spark:3.5.1-scala2.12-java17-python3-ubuntu

pull\_policy: always

container\_name: departuredelay\_spark\_master

hostname: spark-master

command: ["/bin/bash","-lc","/opt/spark/sbin/start-master.sh && tail -f /dev/null"]

environment:

- SPARK\_NO\_DAEMONIZE=true

ports:

- "7077:7077"

- "8080:8080"

- "4040:4040"

volumes:

- ./data:/opt/data

- ./qa:/opt/data/qa

- ./jars:/opt/extra-jars # host jars -> container

restart: unless-stopped

networks: [departuredelay\_net]

spark-worker-1:

image: apache/spark:3.5.1-scala2.12-java17-python3-ubuntu

pull\_policy: always

container\_name: departuredelay\_spark\_worker\_1

depends\_on: [spark-master]

command: ["/bin/bash","-lc","/opt/spark/sbin/start-worker.sh spark://spark-master:7077 && tail -f /dev/null"]

environment:

- SPARK\_WORKER\_CORES=1

- SPARK\_WORKER\_MEMORY=1G

- SPARK\_WORKER\_WEBUI\_PORT=8081

ports:

- "8081:8081"

volumes:

- ./data:/opt/data

restart: unless-stopped

networks: [departuredelay\_net]

mysql:

image: mysql:8.0

pull\_policy: always

container\_name: departuredelay\_mysql

environment:

- MYSQL\_ROOT\_PASSWORD=Root!234

- MYSQL\_DATABASE=flightdb

ports:

- "3307:3306"

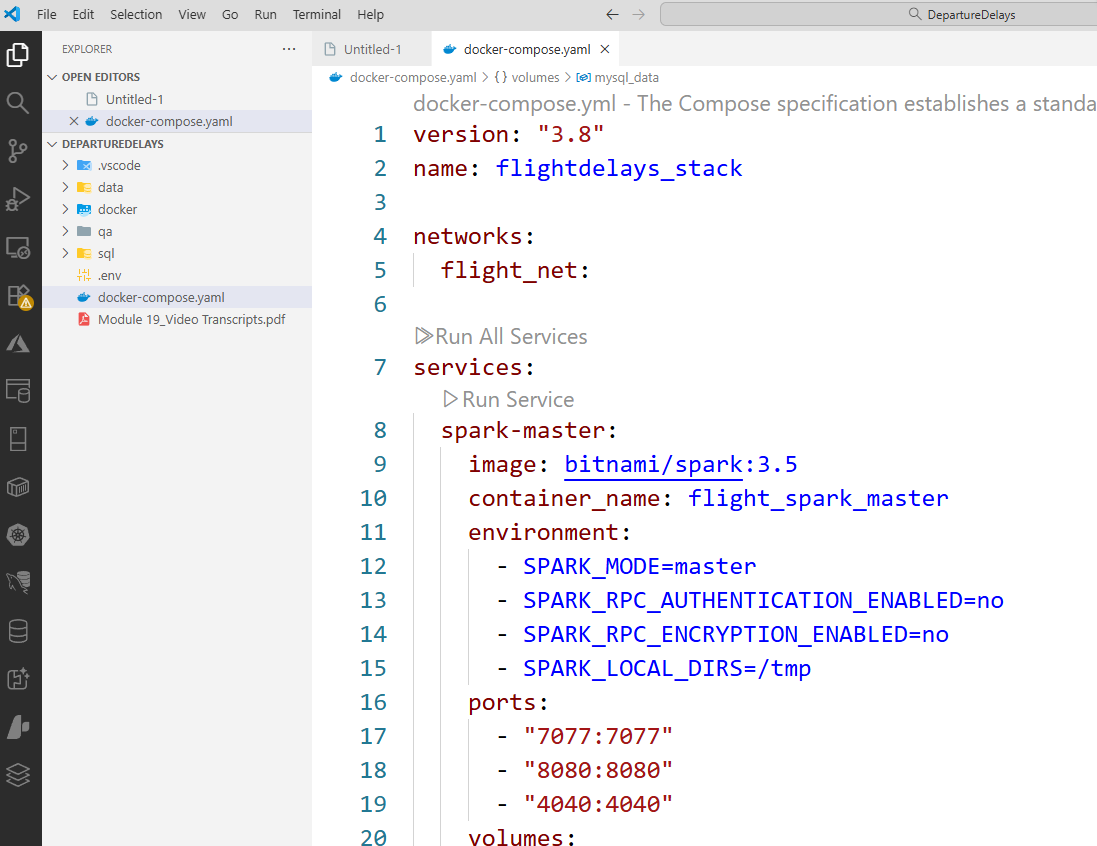
volumes:

- ./docker/mysql/docker-entrypoint-initdb.d:/docker-entrypoint-initdb.d

- departuredelay\_mysql\_data:/var/lib/mysql

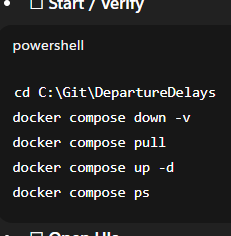
restart: unless-stopped

networks: [departuredelay\_net]



### ▶️ Docker: Start clean, fetch images, start, verify

| **Optional (Image Pull)** Compose will pull images automatically when you run docker compose pull or docker compose up (especially with pull\_policy: always in the YAML). Pre-pulling is only to be explicit or to avoid first-run delays.  docker pull apache/spark:3.5.1-scala2.12-java17-python3-ubuntu  docker pull mysql:8.0 |
| --- |

****

why: I switch to the repo folder so relative paths in docker-compose.yml (like ./data) resolve correctly.

cd “C:\Git\DepartureDelays”

# stop & remove containers, volumes, network

why: I stop and remove containers and their named/anonymous volumes for a clean slate (data loss in volumes is expected).

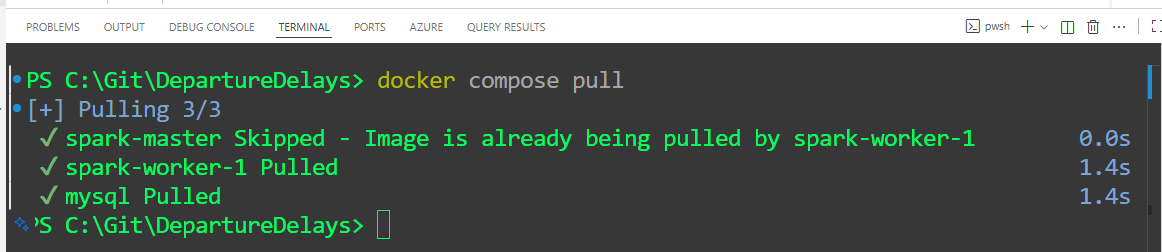
docker compose down -v

|  | | |
| --- | --- | --- |
|  | | |
|  |  |  |

# pulls all the latest images declared in the YAML

why: I fetch the latest images from the registry (useful when images update upstream). Optional if I just built recently.

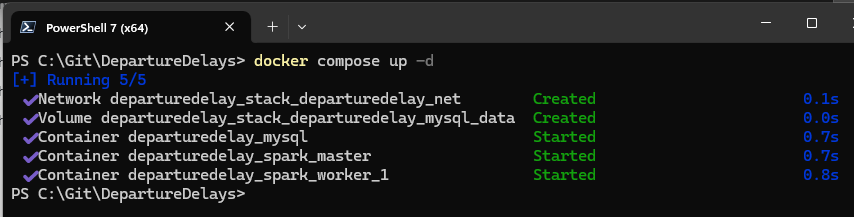
docker compose pull

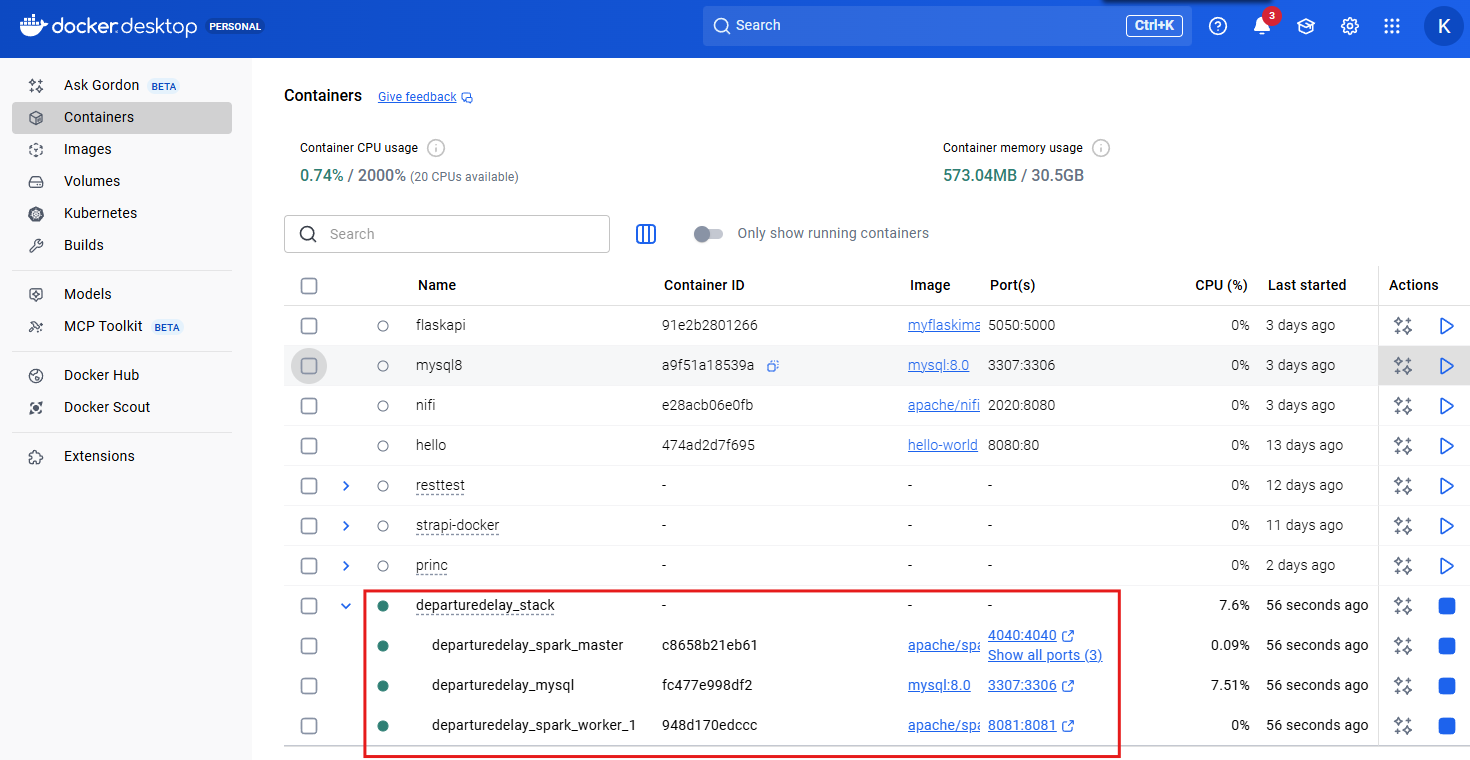


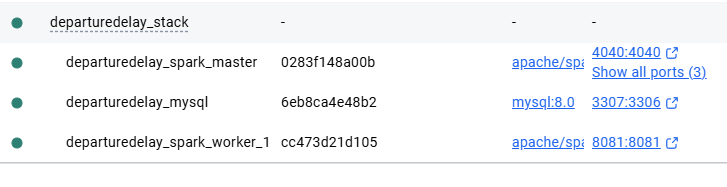
# start in background

why: I (re)build layers if needed and start everything in the background (detached).

docker compose up -d







| # Very Important: Do this to confirm everything’s fine |
| --- |
| Check the master is **Up**, not restarting:  docker ps --filter name=departuredelay\_spark\_master  docker logs departuredelay\_spark\_master --tail=60 |

~~# status of services, states, and short info~~

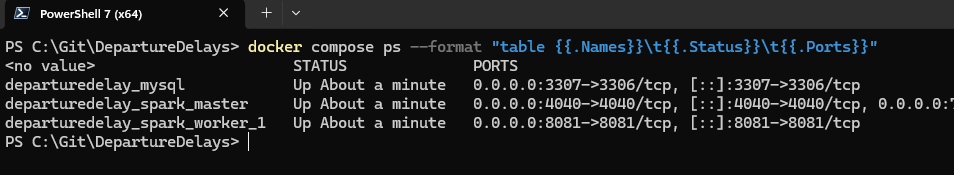
~~why: I get a quick status list of services, states, and short info.~~

~~docker compose ps~~

# table with just Name / Status / Ports

why: I render a neat table with just Name / Status / Ports for fast visual checks.

docker compose ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"

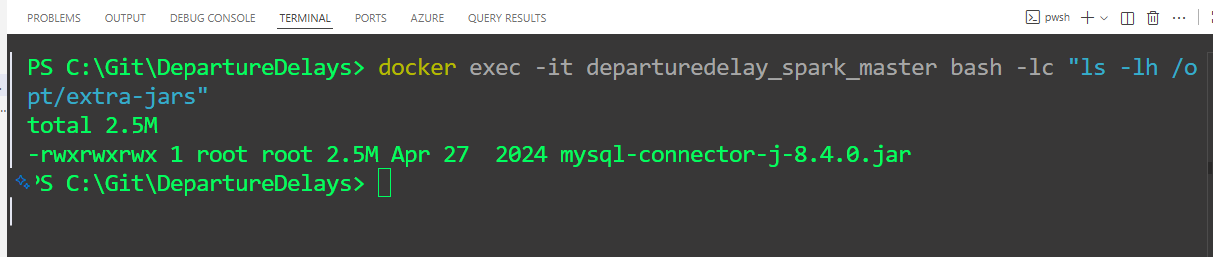


# mysql-connector (if applicable)

why:

docker exec -it departuredelay\_spark\_master bash -lc "ls -lh /opt/extra-jars"

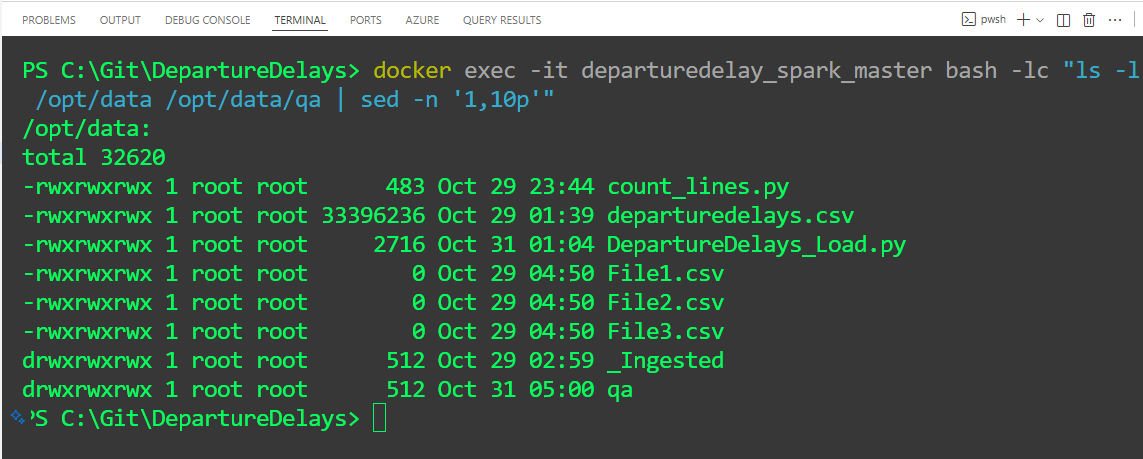
**# expect: mysql-connector-j-8.4.0.jar**



# confirm the host mounts

why: listing files inside the Spark container to confirm Windows folders are mounted correctly. I confirm the host mounts are visible inside the Spark master (/opt/data, /opt/data/qa).**Command (runs only after master is up):**

docker exec -it departuredelay\_spark\_master bash -lc "ls -l /opt/data /opt/data/qa | sed -n '1,10p'"



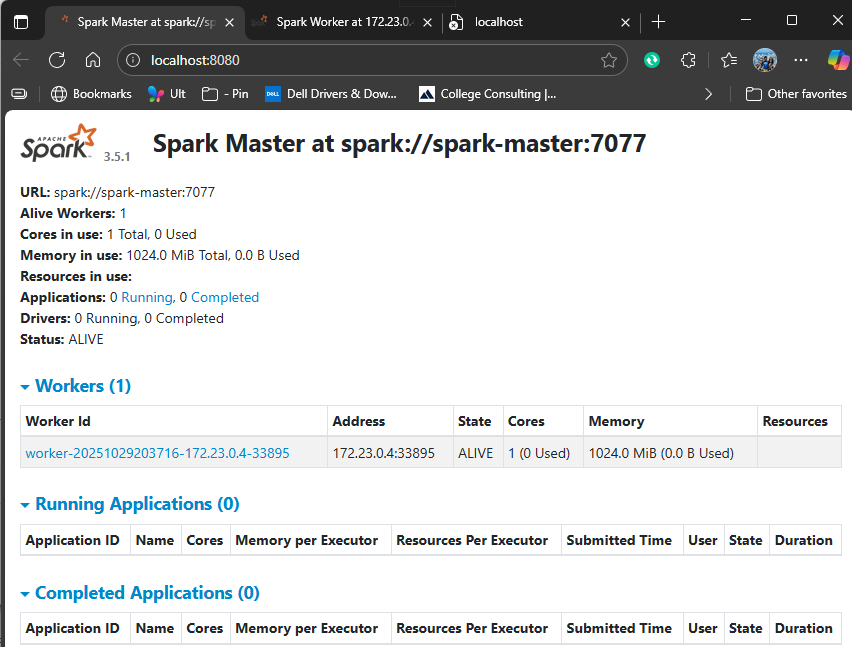
**☐ Open UIs**

**Port cheat-sheet (for your notes)**

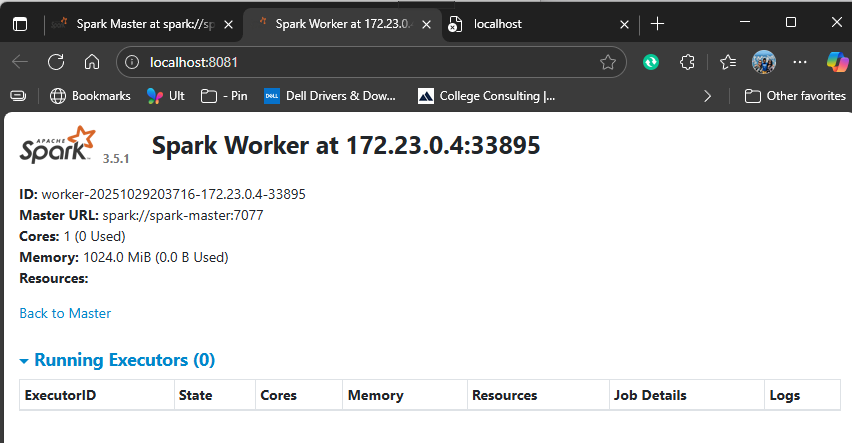
* **7077**: Spark master RPC (workers & drivers connect here)
* **8080**: Spark Master Web UI
* **8081**: Spark Worker Web UI
* **4040**: Spark Application UI (appears only while a job runs)
* **3307 → 3306**: MySQL host→container mapping (we’ll use this when wiring Spark ↔ MySQL)
* 🔗 **Spark Master:**<http://localhost:8080>

Or

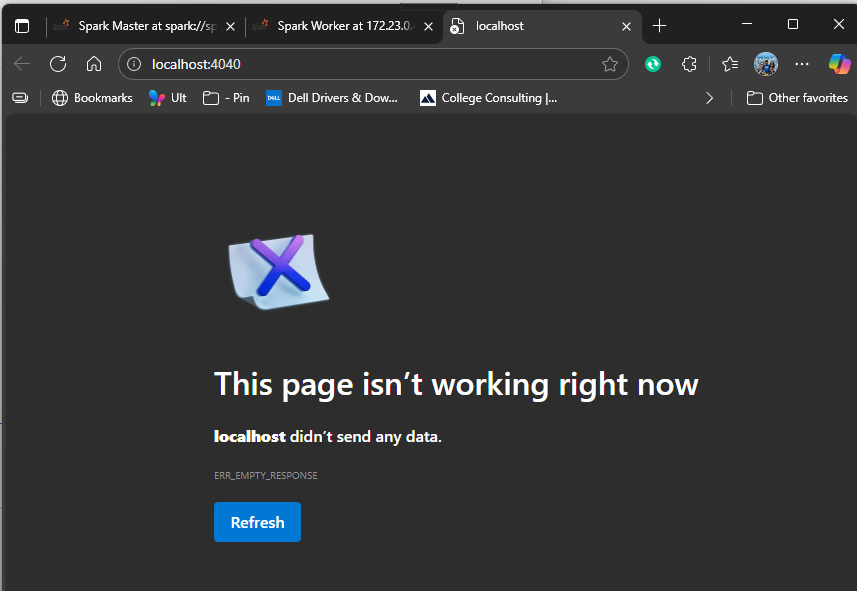
http://localhost:8090/



* 🔗 **Spark Worker:**<http://localhost:8081>



* 🔗 **Spark App UI (when jobs run):**<http://localhost:4040>



ℹ️ 4040 is per-application UI; it only appears when a Spark job is running. The blank page is expected right now.

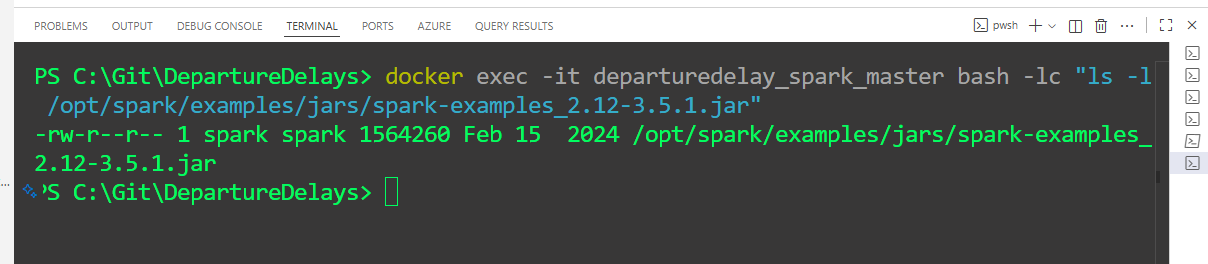
### ✅Sanity Check

**☐ Next step (make 4040 appear + sanity test the cluster)**

**A) Run a built-in job (SparkPi) against the master**

**✅ Sanity check: jar exists**

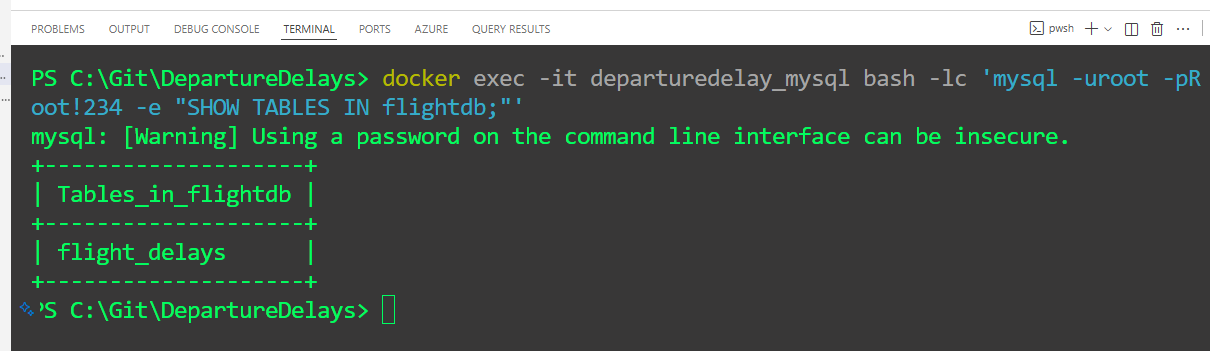
docker exec -it departuredelay\_spark\_master bash -lc "ls -l /opt/spark/examples/jars/spark-examples\_2.12-3.5.1.jar"



**✅ Sanity check:** Point to an existing table (if named differently)

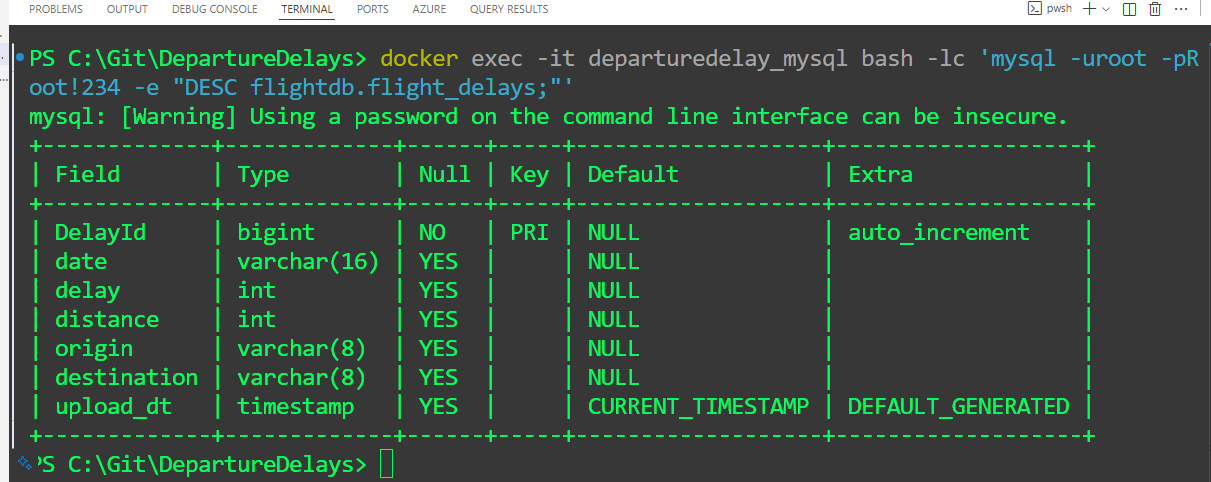
# list tables in flightdb

docker exec -it departuredelay\_mysql bash -lc 'mysql -uroot -pRoot!234 -e "SHOW TABLES IN flightdb;"'



**✅ Optional sanity check (schema)**

docker exec -it departuredelay\_mysql bash -lc 'mysql -uroot -pRoot!234 -e "DESC flightdb.flight\_delays;"'

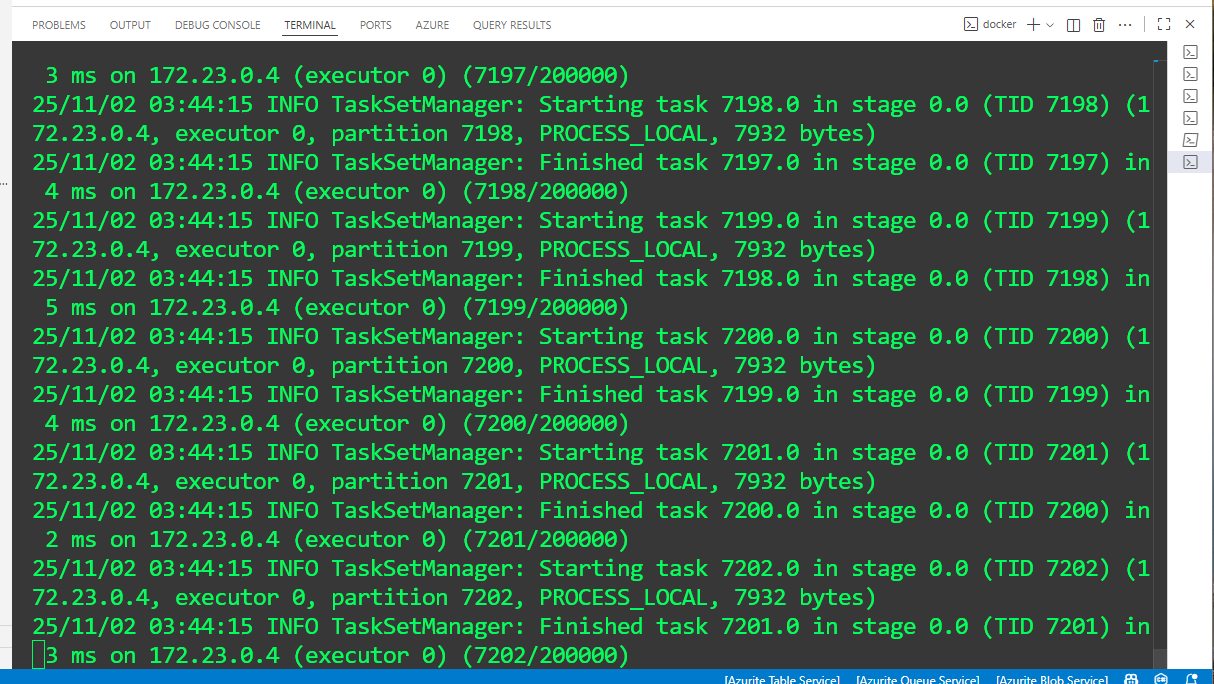


**▶️ Run SparkPi (one-liner)**

4040 doesn’t stay up all the time, that’s expected.

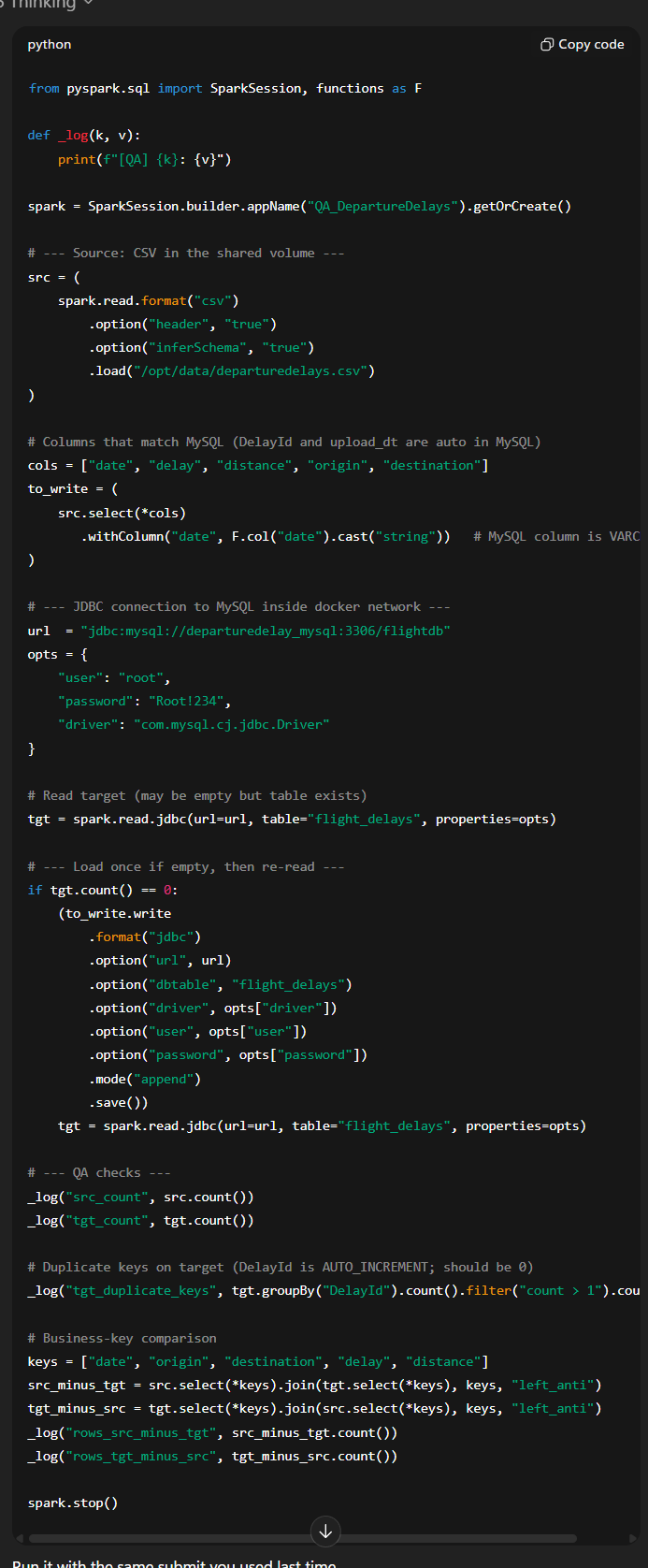
* Spark Master UI (8080/8090): stays up as long as the master container is running.
* 4040 UI: per-application. It only exists while a Spark job is running (then it shuts down). Nothing’s wrong.
* While it runs, I open **http://localhost:4040** (or 4041 if Spark auto-bumps).

docker exec -it departuredelay\_spark\_master bash -lc "/opt/spark/bin/spark-submit --master spark://spark-master:7077 --class org.apache.spark.examples.SparkPi /opt/spark/examples/jars/spark-examples\_2.12-3.5.1.jar 200000"



| ✅ While this runs, open **http://localhost:4040** → you should see the Spark UI for this app.  ✅ In **http://localhost:8080**, the app should show under *Running/Completed Applications*. |
| --- |

#### ▶️tests\_pyspark.py



from pyspark.sql import SparkSession, functions as F

def \_log(k, v):

print(f"[QA] {k}: {v}")

spark = SparkSession.builder.appName("QA\_DepartureDelays").getOrCreate()

# --- Source: CSV in the shared volume ---

src = (

spark.read.format("csv")

.option("header", "true")

.option("inferSchema", "true")

.load("/opt/data/departuredelays.csv")

)

# Columns that match MySQL (DelayId and upload\_dt are auto in MySQL)

cols = ["date", "delay", "distance", "origin", "destination"]

to\_write = (

src.select(\*cols)

.withColumn("date", F.col("date").cast("string")) # MySQL column is VARCHAR

)

# --- JDBC connection to MySQL inside docker network ---

url = "jdbc:mysql://departuredelay\_mysql:3306/flightdb"

opts = {

"user": "root",

"password": "Root!234",

"driver": "com.mysql.cj.jdbc.Driver"

}

# Read target (may be empty but table exists)

tgt = spark.read.jdbc(url=url, table="flight\_delays", properties=opts)

# --- Load once if empty, then re-read ---

if tgt.count() == 0:

(to\_write.write

.format("jdbc")

.option("url", url)

.option("dbtable", "flight\_delays")

.option("driver", opts["driver"])

.option("user", opts["user"])

.option("password", opts["password"])

.mode("append")

.save())

tgt = spark.read.jdbc(url=url, table="flight\_delays", properties=opts)

# --- QA checks ---

\_log("src\_count", src.count())

\_log("tgt\_count", tgt.count())

# Duplicate keys on target (DelayId is AUTO\_INCREMENT; should be 0)

\_log("tgt\_duplicate\_keys", tgt.groupBy("DelayId").count().filter("count > 1").count())

# Business-key comparison

keys = ["date", "origin", "destination", "delay", "distance"]

src\_minus\_tgt = src.select(\*keys).join(tgt.select(\*keys), keys, "left\_anti")

tgt\_minus\_src = tgt.select(\*keys).join(src.select(\*keys), keys, "left\_anti")

\_log("rows\_src\_minus\_tgt", src\_minus\_tgt.count())

\_log("rows\_tgt\_minus\_src", tgt\_minus\_src.count())

spark.stop()

**▶️ Run my QA script (one-liner)**

docker exec -it departuredelay\_spark\_master bash -lc '/opt/spark/bin/spark-submit --master spark://spark-master:7077 --jars /opt/extra-jars/mysql-connector-j-8.4.0.jar --driver-class-path /opt/extra-jars/mysql-connector-j-8.4.0.jar /opt/data/qa/tests\_pyspark.py'

|  |
| --- |
|  |
|  |

#### ▶️count\_lines.py

**☐ Sanity check with a tiny PySpark job**

1. On host, create C:\Git\DepartureDelays\data\count\_lines.py:

# why: tiny job to prove submit + worker execution

# why: hold Spark UI open long enough to inspect

# why: keep Spark UI up long enough to view on 4040

from pyspark.sql import SparkSession

import time

spark = SparkSession.builder.appName("count\_lines\_ui").getOrCreate()

print("UI:", spark.sparkContext.uiWebUrl) # prints actual 404x inside container

df = spark.range(0, 100\_000\_000)

print("Rows:", df.count())

time.sleep(120) # keep UI alive for 2 minutes

spark.stop()

* **File:** data/count\_lines.py (exactly what you have in the screenshot)
* **Purpose (why):**
  + Hold the Spark **app UI (4040/4041...)** open for demos.
  + Quick “driver→master→worker” sanity (no MySQL/JDBC needed).

**▶️ How I run it (with your container names)**

docker exec departuredelay\_spark\_master /opt/spark/bin/spark-submit `

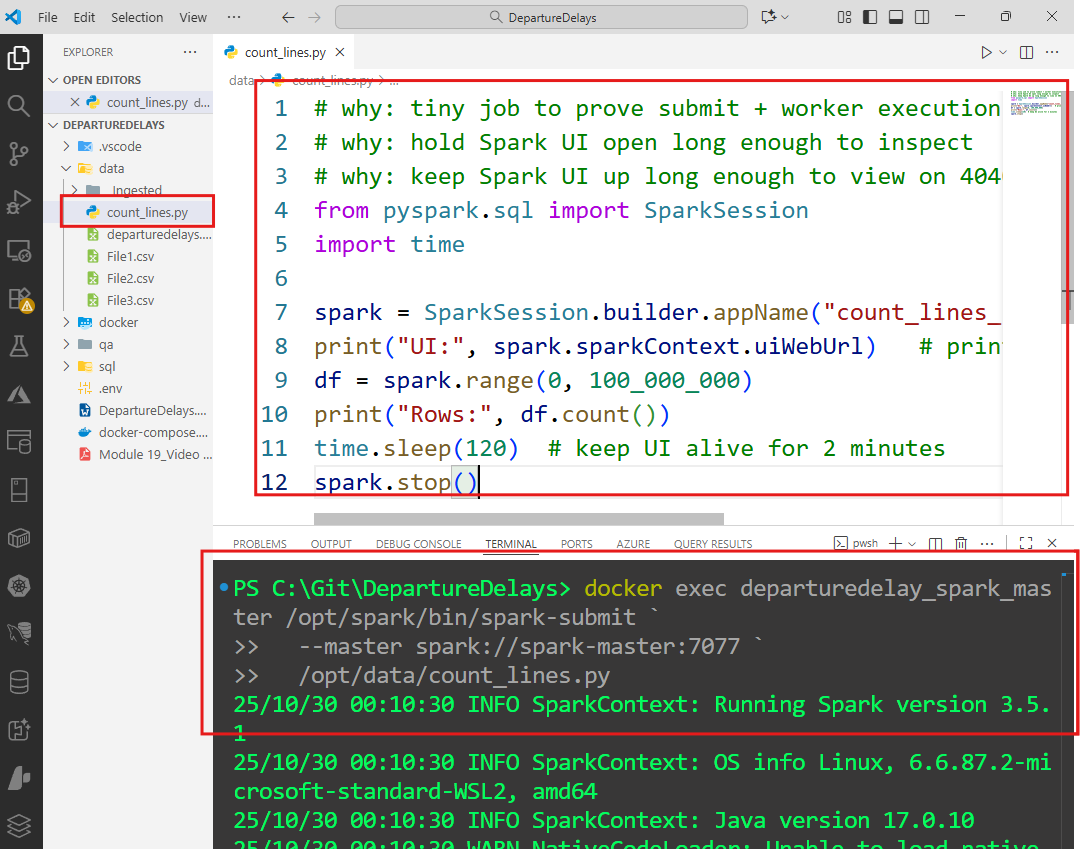
--master spark://spark-master:7077 `

/opt/data/count\_lines.py

* Open [**http://localhost:4040**](http://localhost:4040) right away (Spark might choose 4041+; the script prints the actual uiWebUrl).

You should see:

* Logs with "Rows: 100000"
* Active stages/executors in **4040**
* App listed in **8080** (Master UI)



**🧭 Where it fits in the flow**

* **Before** loader/QA: run count\_lines.py once to validate the cluster + show 4040.
* **After** that, proceed with:
  1. Create DB/table in **departuredelay\_mysql**
  2. Run **departure\_delays\_loader.py** (CSV → MySQL)
  3. Run **QA** scripts (PySpark + SQL)
  4. **Git push** repo + send email links

You already have all those steps from me with your exact container names. If you’re good with keeping count\_lines.py as a utility, I’ll move ahead assuming this flow:

* Keep data/count\_lines.py (UI keeper)
* Run it when you want 4040 alive
* Use the loader + QA + Git steps I gave next

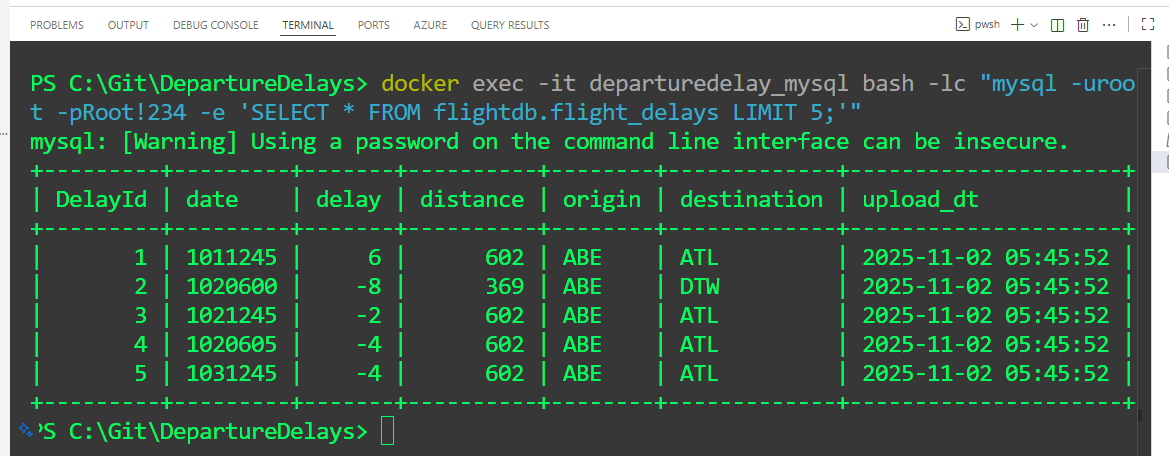
#### ▶️Row count in MySQL (container):

docker exec -it departuredelay\_mysql bash -lc "mysql -uroot -pRoot!234 -e 'SELECT COUNT(\*) AS rows\_in\_table FROM flightdb.flight\_delays;'"



#### ▶️Peek a few rows:

docker exec -it departuredelay\_mysql bash -lc "mysql -uroot -pRoot!234 -e 'SELECT \* FROM flightdb.flight\_delays LIMIT 5;'"

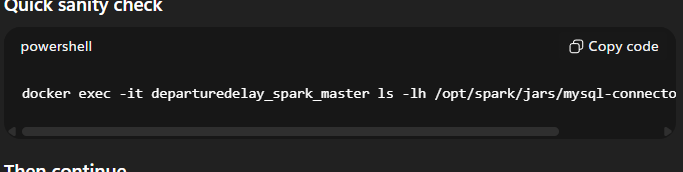


# 🐍 3) PySpark — Load CSV → Query → Persist to MySQL

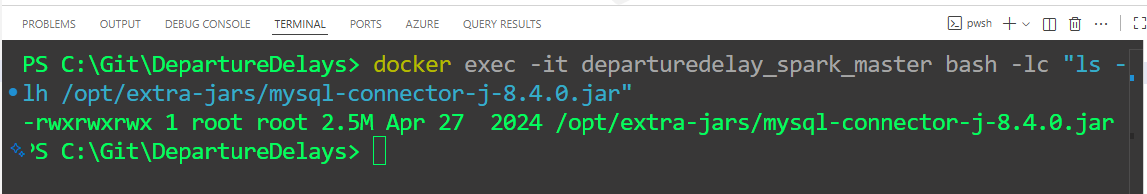
* You’ll **read** the departuredelays.csv file with PySpark (fast, distributed DataFrame engine).
* You’ll **add two columns**: a unique surrogate key (DelayId) and an upload\_date.
* You’ll **run a couple quick queries** (just to prove you can filter/aggregate).
* Then you’ll **write the result into MySQL** (table **Flight Delays**) through JDBC — all **from the Spark master container**.
* While the job runs, the **Spark App UI** appears on **4040** (or 4041+) and disappears when the job ends — that’s expected.

Using your exact names from the screenshots:  
**stack:** departuredelay\_stack  
**containers:** departuredelay\_spark\_master, departuredelay\_spark\_worker\_1, departuredelay\_mysql

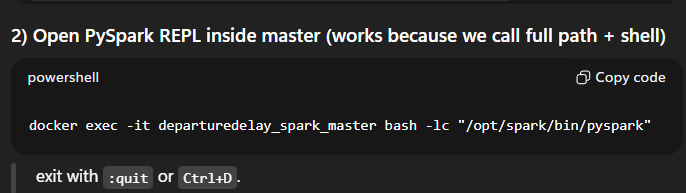
**☐** Quick sanity check



docker exec -it departuredelay\_spark\_master bash -lc "ls -lh /opt/extra-jars/mysql-connector-j-8.4.0.jar"

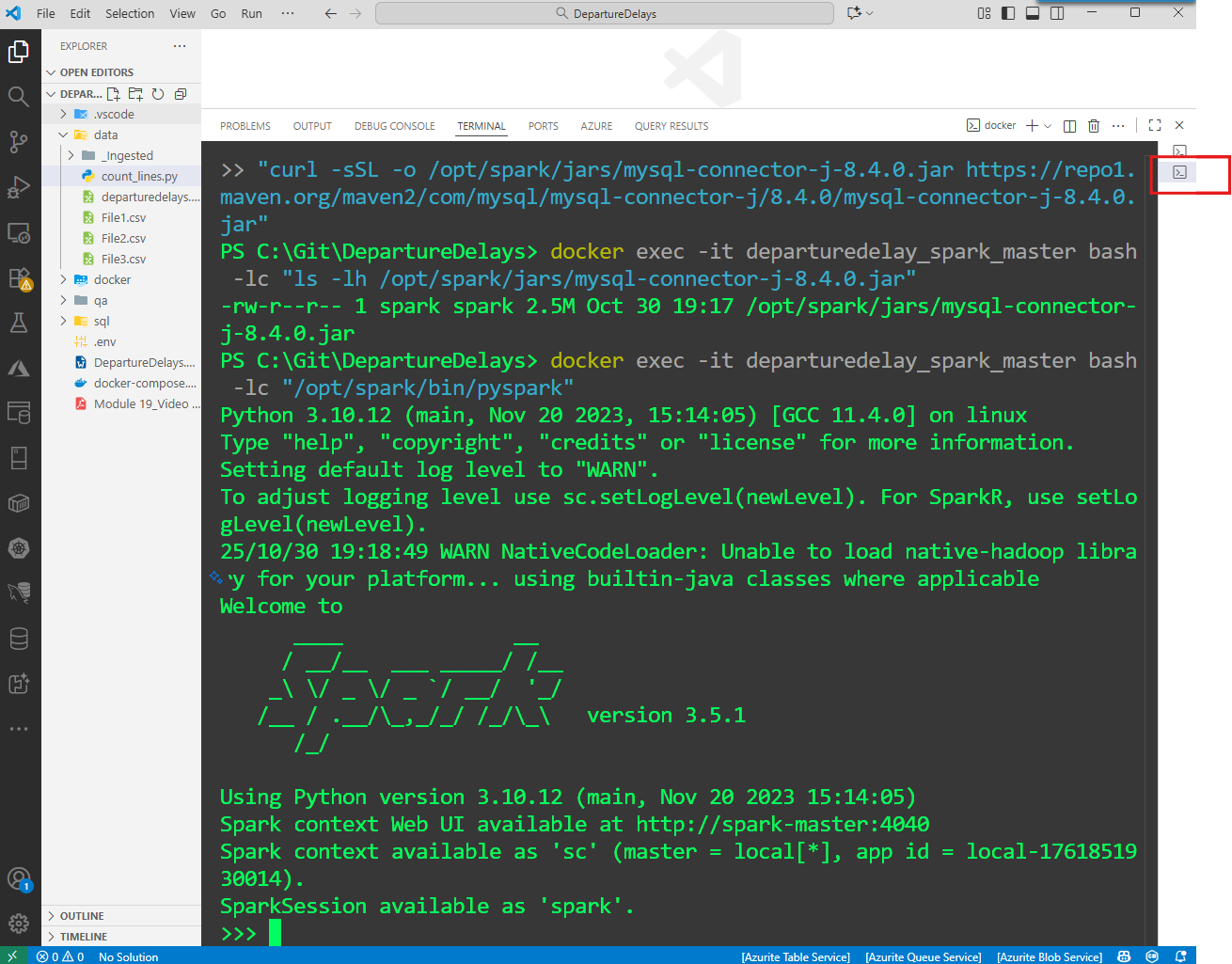


**☐ Open PySpark REPL inside master (works because we call full path + shell)**



docker exec -it departuredelay\_spark\_master bash -lc "/opt/spark/bin/pyspark"

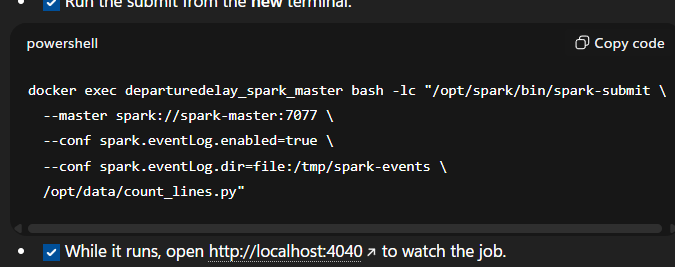
exit with :quit or Ctrl+D.



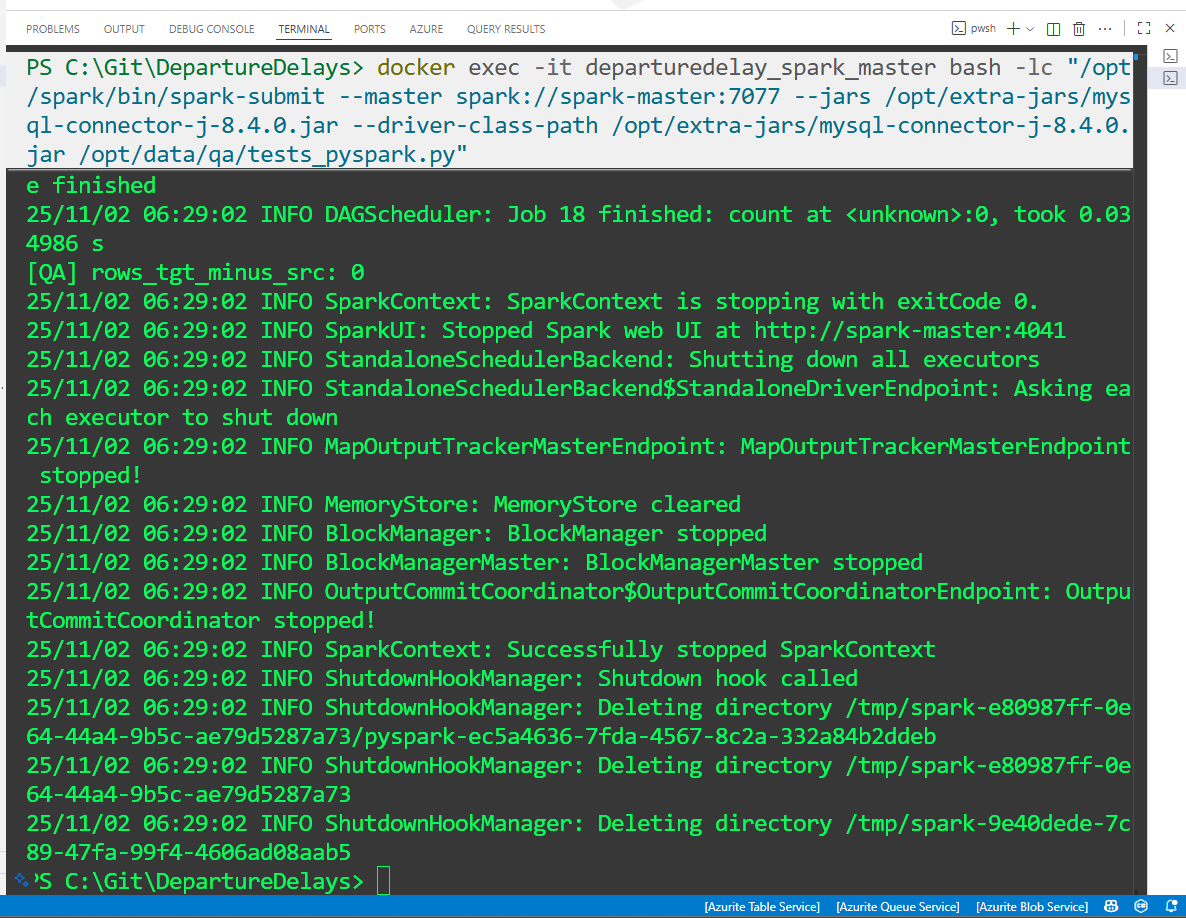
**☐ What I do now**

**Option A — keep the REPL open and use a second terminal (my pick)**

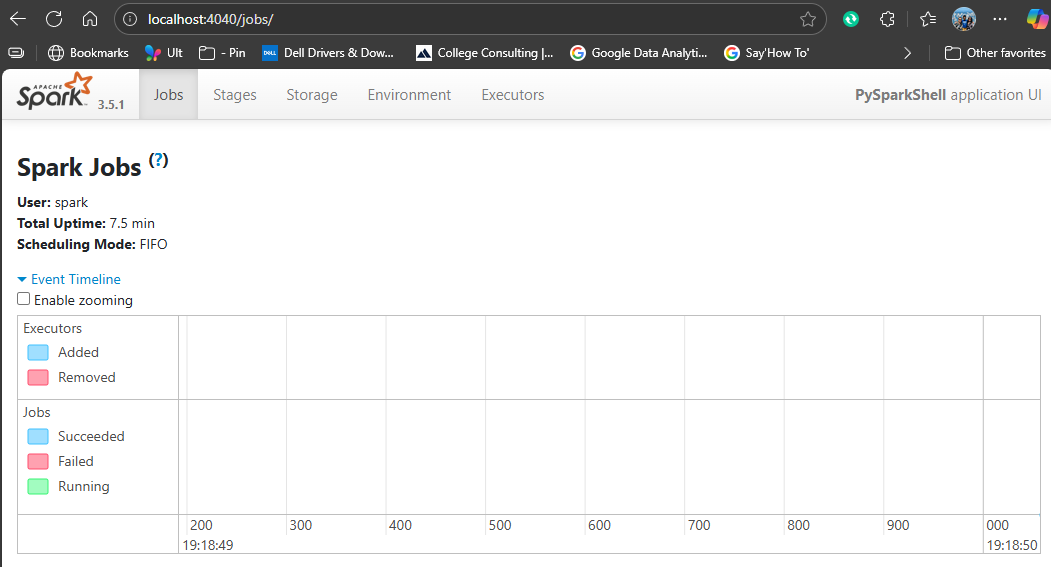
* In VS Code, click **➕** in the Terminal panel to open a **new** tab.
* Run the submit from the **new** terminal:



docker exec -it departuredelay\_spark\_master bash -lc "/opt/spark/bin/spark-submit --master spark://spark-master:7077 --jars /opt/extra-jars/mysql-connector-j-8.4.0.jar --driver-class-path /opt/extra-jars/mysql-connector-j-8.4.0.jar /opt/data/qa/tests\_pyspark.py"



* While it runs, open [**http://localhost:4040**](http://localhost:4040) to watch the job.



## 🏃Run this PySpark block (read CSV → queries → write to MySQL)

**Plan for DepartureDelays\_Load.py (what + why)**

☑️ **Load the CSV into Spark** I pull the spreadsheet (departuredelays.csv) from the shared mount (/opt/data). Spark reads it fast and understands columns/types. *Why:* work with millions of rows reliably, not just in Excel.

☑️ **Add housekeeping columns** I create a **surrogate key** DelayId (a unique number Spark generates) and an **upload\_dt** (today’s date). *Why:* every row becomes uniquely addressable and I can track when I loaded it.

☑️ **Quick sanity peeks** I run tiny select/filter/order queries (e.g., show top delays, SFO→ORD sample). *Why:* quick smell-test that the data looks right before I publish it.

☑️ **Publish to MySQL (table: flightdb.flight\_delays)** Using the MySQL **JDBC driver** we downloaded, I write the cleaned DataFrame to MySQL. I can use **mode("overwrite")** now (repeatable), later switch to **append** for incremental loads. *Why:* the team’s tools (MySQL Workbench, BI dashboards) can hit a durable database, not a loose CSV.

☑️ **Watch it live** While it runs, I can open **http://localhost:4040** to see stages/tasks. *Why:* instant feedback on progress and any bottlenecks.

☑️ **Containers do the plumbing** Spark Master/Worker containers do the compute; the **MySQL** container stores results. They’re on the same Docker network so **departuredelay\_mysql:3306** works as the DB host. *Why:* consistent, reproducible environment.

☑️ **Idempotent + auditable** The script can be re-run safely; the extra columns (DelayId, upload\_dt) make audits and deduping simpler. *Why:* reliable pipelines beat ad-hoc one-offs.

☑️ Outcome A clean, queryable table flightdb.flight\_delays ready for QA checks (row counts, dup checks, source↔target diffs) and for downstream analytics.

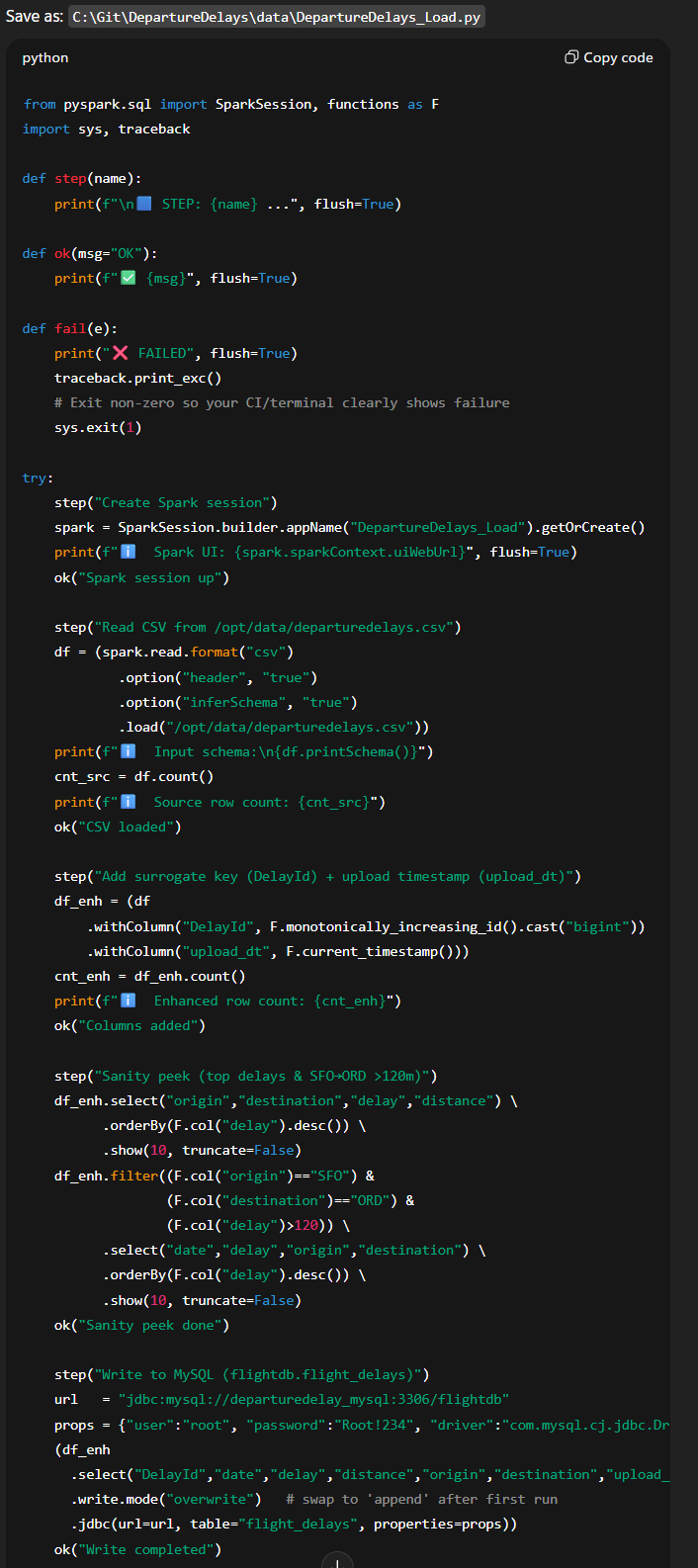
**☐ Quick sanity: make sure the CSV is visible inside master**

docker exec -it departuredelay\_spark\_master bash -lc "ls -lh /opt/data/departuredelays.csv | cat"

You should see a real size (KB/MB). If “No such file” appears, your mount path is wrong.

**☐ Use the working PySpark loader (inside-container connection + safe table name)**

Create this file at C:\Git\DepartureDelays\data\DepartureDelays\_Load.py.



from pyspark.sql import SparkSession, functions as F

import sys, traceback

def step(name):

print(f"\n🟦 STEP: {name} ...", flush=True)

def ok(msg="OK"):

print(f"✅ {msg}", flush=True)

def fail(e):

print("❌ FAILED", flush=True)

traceback.print\_exc()

# Exit non-zero so your CI/terminal clearly shows failure

sys.exit(1)

try:

step("Create Spark session")

spark = SparkSession.builder.appName("DepartureDelays\_Load").getOrCreate()

print(f"ℹ️ Spark UI: {spark.sparkContext.uiWebUrl}", flush=True)

ok("Spark session up")

step("Read CSV from /opt/data/departuredelays.csv")

df = (spark.read.format("csv")

.option("header", "true")

.option("inferSchema", "true")

.load("/opt/data/departuredelays.csv"))

print(f"ℹ️ Input schema:\n{df.printSchema()}")

cnt\_src = df.count()

print(f"ℹ️ Source row count: {cnt\_src}")

ok("CSV loaded")

step("Add surrogate key (DelayId) + upload timestamp (upload\_dt)")

df\_enh = (df

.withColumn("DelayId", F.monotonically\_increasing\_id().cast("bigint"))

.withColumn("upload\_dt", F.current\_timestamp()))

cnt\_enh = df\_enh.count()

print(f"ℹ️ Enhanced row count: {cnt\_enh}")

ok("Columns added")

step("Sanity peek (top delays & SFO→ORD >120m)")

df\_enh.select("origin","destination","delay","distance") \

.orderBy(F.col("delay").desc()) \

.show(10, truncate=False)

df\_enh.filter((F.col("origin")=="SFO") &

(F.col("destination")=="ORD") &

(F.col("delay")>120)) \

.select("date","delay","origin","destination") \

.orderBy(F.col("delay").desc()) \

.show(10, truncate=False)

ok("Sanity peek done")

step("Write to MySQL (flightdb.flight\_delays)")

url = "jdbc:mysql://departuredelay\_mysql:3306/flightdb"

props = {"user":"root", "password":"Root!234", "driver":"com.mysql.cj.jdbc.Driver"}

(df\_enh

.select("DelayId","date","delay","distance","origin","destination","upload\_dt")

.write.mode("overwrite") # swap to 'append' after first run

.jdbc(url=url, table="flight\_delays", properties=props))

ok("Write completed")

step("Post-write verification (read back 5 rows via JDBC)")

df\_check = spark.read.jdbc(url=url, table="flight\_delays", properties=props)

print(f"ℹ️ Target row count (quick): {df\_check.count()}")

df\_check.orderBy(F.col("DelayId").asc()).show(5, truncate=False)

ok("Verified target")

step("Stop Spark")

spark.stop()

ok("All done 🎉")

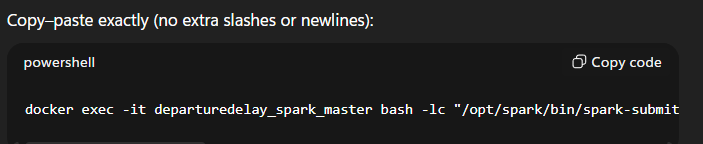
except Exception as e:

fail(e)

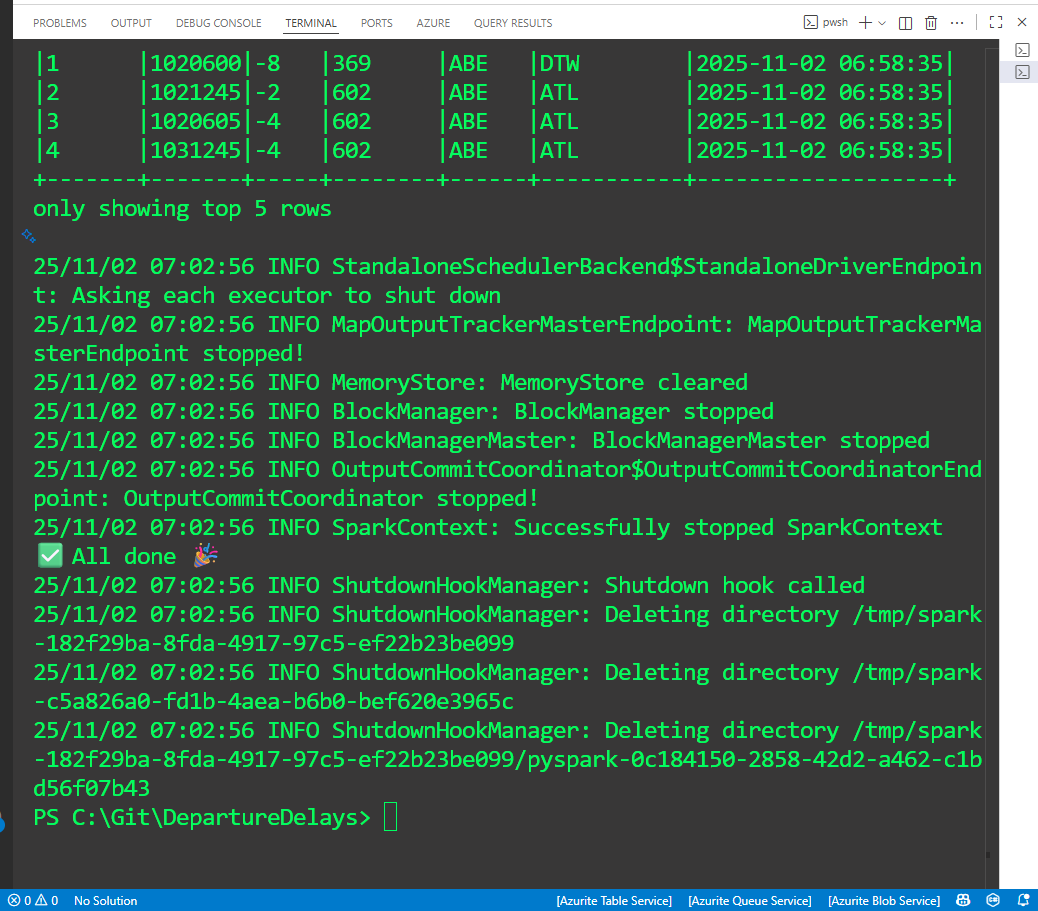
| Truncate (clean slate) from PowerShell:  docker exec -it departuredelay\_mysql bash -lc "mysql -uroot -pRoot!234 -e 'TRUNCATE TABLE flightdb.flight\_delays;'"    The warning is normal for MySQL CLI. The table is now empty. |
| --- |

**☐ Submit the job from host into the master**

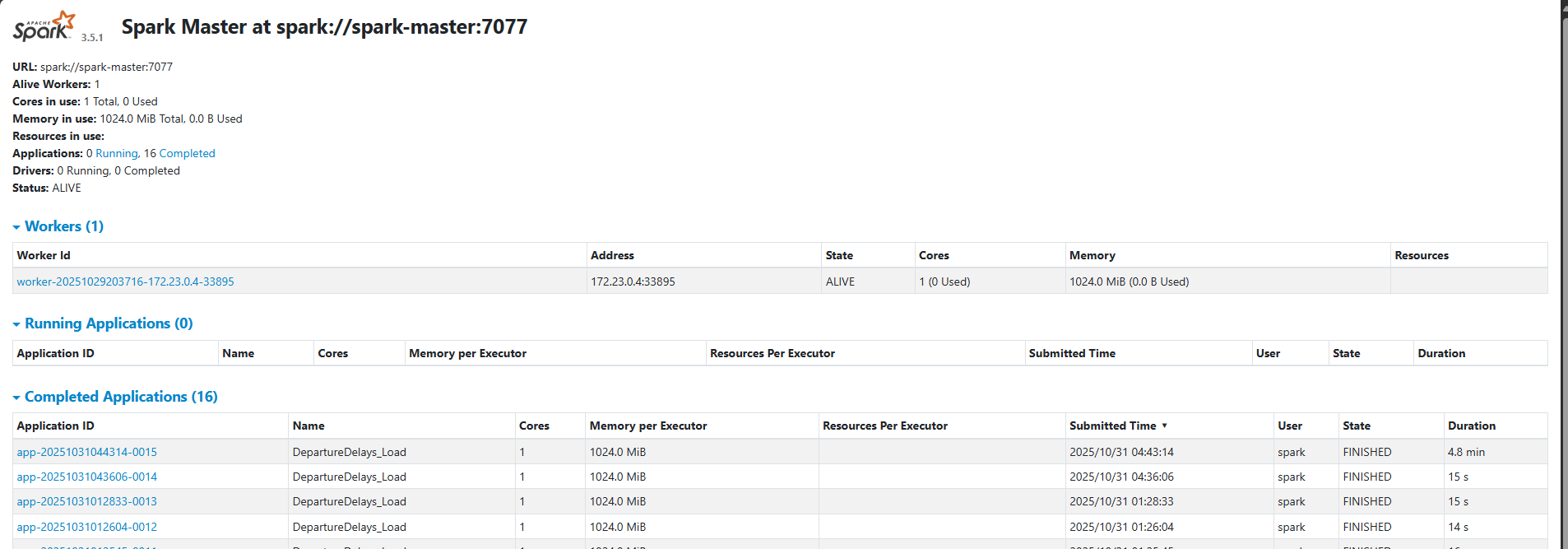
Copy–paste exactly (no extra slashes or newlines):



docker exec -it departuredelay\_spark\_master bash -lc "/opt/spark/bin/spark-submit --deploy-mode client --master spark://spark-master:7077 --jars /opt/extra-jars/mysql-connector-j-8.4.0.jar --driver-class-path /opt/extra-jars/mysql-connector-j-8.4.0.jar /opt/data/DepartureDelays\_Load.py"

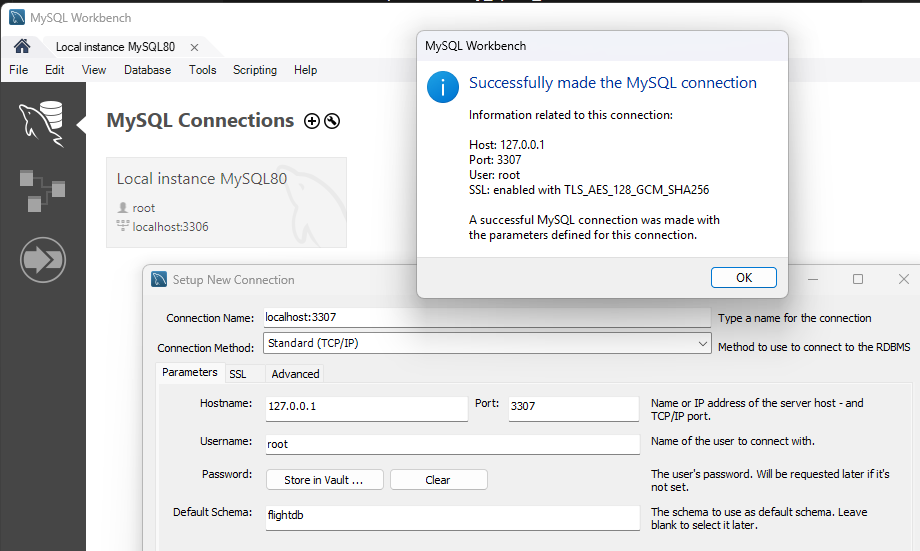


* While it runs, open [**http://localhost:4040**](http://localhost:4040) (Spark UI for this app).
* Master UI: [**http://localhost:8080**](http://localhost:8080) should list the running/completed app.

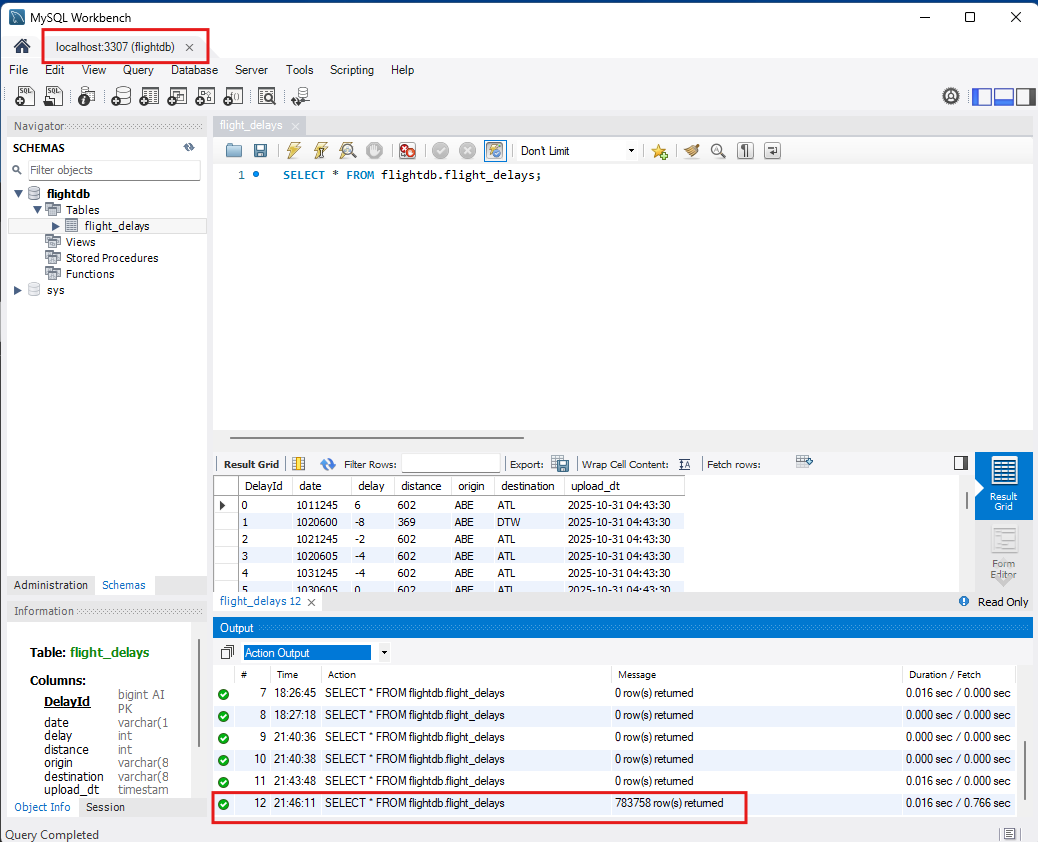


| **Re-run** the Spark job (when you want to load file with same name) |
| --- |
| To **re-run** the Spark job you only need this one command from your **VS Code terminal** (PowerShell):  docker exec departuredelay\_spark\_master bash -lc "/opt/spark/bin/spark-submit \  --master spark://spark-master:7077 \  --conf spark.eventLog.enabled=true \  --conf spark.eventLog.dir=file:/tmp/spark-events \  /opt/data/DepartureDelays\_Load.py"  Quick checklist before re-run   * Containers are up (you should see departuredelay\_spark\_master, departuredelay\_spark\_worker\_1, departuredelay\_mysql running). * The JDBC jar exists (you already validated):   docker exec departuredelay\_spark\_master bash -lc "ls -lh /opt/spark/jars/mysql-connector-j-8.4.0.jar" * Your script path is still /opt/data/DepartureDelays\_Load.py (mapped from C:\Git\DepartureDelays\data\DepartureDelays\_Load.py).   Tips  Want to **append** instead of overwrite on repeat runs? In the script, change:   .write.mode("overwrite")  to:   .write.mode("append")  Copy–paste exactly (no extra slashes or newlines):    docker exec -it departuredelay\_spark\_master bash -lc "/opt/spark/bin/spark-submit --deploy-mode client --master spark://spark-master:7077 --jars /opt/extra-jars/mysql-connector-j-8.4.0.jar --driver-class-path /opt/extra-jars/mysql-connector-j-8.4.0.jar /opt/data/DepartureDelays\_Load.py"  Watch progress at [**http://localhost:4040**](http://localhost:4040) while it runs.  Verify in MySQL Workbench (connection localhost:3307): |

**☐ Verify in MySQL**

* From Workbench: connect to **localhost:3307**, user **root**, password **Root!234**, DB **flightdb**.  
  
* Run:

SELECT \* FROM `flightdb`.`Flight Delays`

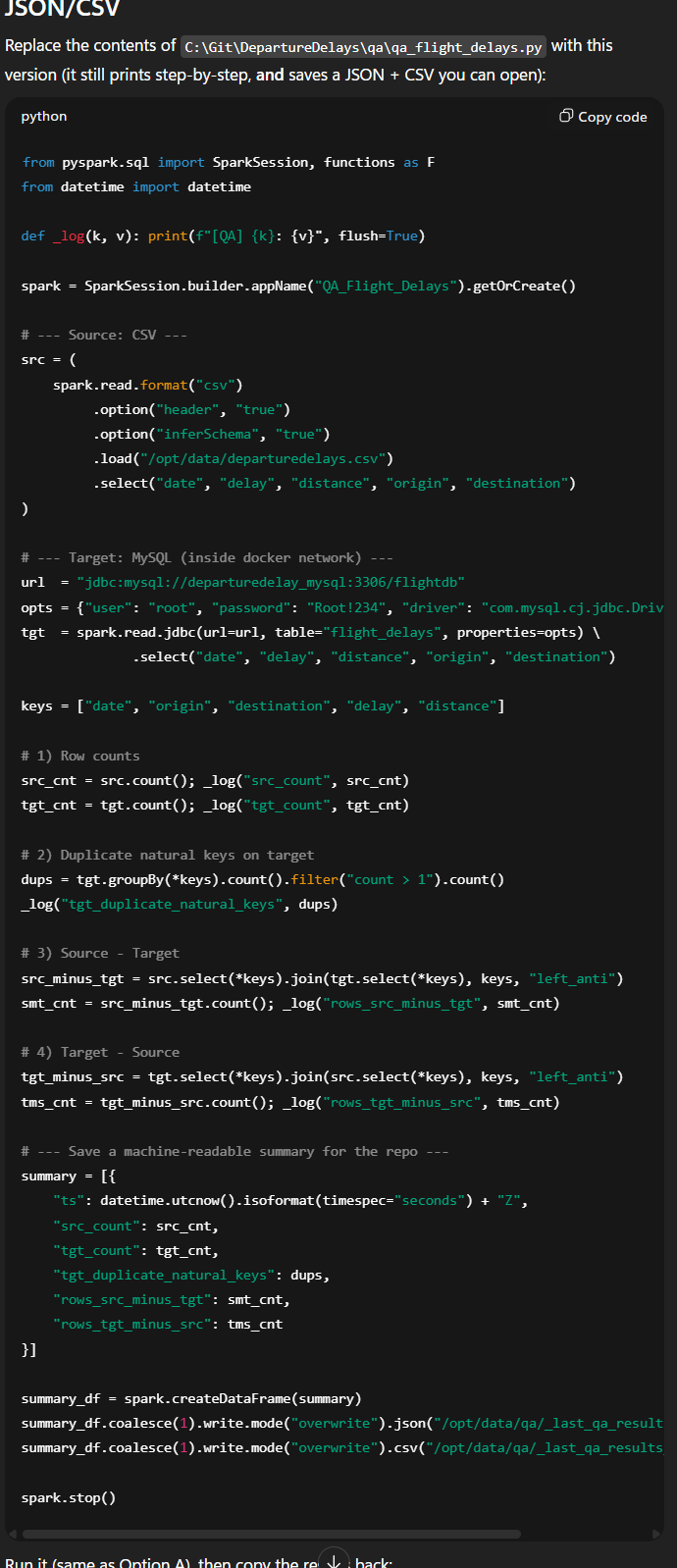


## 🧪 QA mini-suite (CSV vs MySQL)

Put these into your repo so QA can re-run from Docker.  
Folder (**C:\Git\DepartureDelays\qa**) already exists in your layout.

### PySpark QA (source↔target checks) → qa/qa\_flight\_delays.py

Create output in 3 formats (log, \*.json, \*.csv)



from pyspark.sql import SparkSession, functions as F

from datetime import datetime

def \_log(k, v): print(f"[QA] {k}: {v}", flush=True)

spark = SparkSession.builder.appName("QA\_Flight\_Delays").getOrCreate()

# --- Source: CSV ---

src = (

spark.read.format("csv")

.option("header", "true")

.option("inferSchema", "true")

.load("/opt/data/departuredelays.csv")

.select("date", "delay", "distance", "origin", "destination")

)

# --- Target: MySQL (inside docker network) ---

url = "jdbc:mysql://departuredelay\_mysql:3306/flightdb"

opts = {"user": "root", "password": "Root!234", "driver": "com.mysql.cj.jdbc.Driver"}

tgt = spark.read.jdbc(url=url, table="flight\_delays", properties=opts) \

.select("date", "delay", "distance", "origin", "destination")

keys = ["date", "origin", "destination", "delay", "distance"]

# 1) Row counts

src\_cnt = src.count(); \_log("src\_count", src\_cnt)

tgt\_cnt = tgt.count(); \_log("tgt\_count", tgt\_cnt)

# 2) Duplicate natural keys on target

dups = tgt.groupBy(\*keys).count().filter("count > 1").count()

\_log("tgt\_duplicate\_natural\_keys", dups)

# 3) Source - Target

src\_minus\_tgt = src.select(\*keys).join(tgt.select(\*keys), keys, "left\_anti")

smt\_cnt = src\_minus\_tgt.count(); \_log("rows\_src\_minus\_tgt", smt\_cnt)

# 4) Target - Source

tgt\_minus\_src = tgt.select(\*keys).join(src.select(\*keys), keys, "left\_anti")

tms\_cnt = tgt\_minus\_src.count(); \_log("rows\_tgt\_minus\_src", tms\_cnt)

# --- Save a machine-readable summary for the repo ---

summary = [{

"ts": datetime.utcnow().isoformat(timespec="seconds") + "Z",

"src\_count": src\_cnt,

"tgt\_count": tgt\_cnt,

"tgt\_duplicate\_natural\_keys": dups,

"rows\_src\_minus\_tgt": smt\_cnt,

"rows\_tgt\_minus\_src": tms\_cnt

}]

summary\_df = spark.createDataFrame(summary)

summary\_df.coalesce(1).write.mode("overwrite").json("/opt/data/qa/\_last\_qa\_results\_json")

summary\_df.coalesce(1).write.mode("overwrite").csv("/opt/data/qa/\_last\_qa\_results\_csv", header=True)

spark.stop()

**🧭 How to Run this**

| **Run the QA script with spark-submit inside the master**  docker exec -it departuredelay\_spark\_master bash -lc '/opt/spark/bin/spark-submit --deploy-mode client --master spark://spark-master:7077 --jars /opt/extra-jars/mysql-connector-j-8.4.0.jar --driver-class-path /opt/extra-jars/mysql-connector-j-8.4.0.jar /opt/data/qa/postIngestionDataCompare.py 2>&1 | tee /opt/data/qa/\_last\_qa\_run.log' |
| --- |

**🧭 Where to see the step outputs (Complete This Section after Certificate)**

| **Next steps which i haven’t done are… which includes recreating new docker-compose.yaml with following changes…**  **This is done**    **Following not done…**  **Refer to Emeritus 41 Tools 17-19**  **https://chatgpt.com/share/6907a8ae-48c8-8013-92da-50b5ba41cf0e** **✅ Next steps (Option A: write to /opt/qa)****1) Save code (already done)** # write results (1 file each)  summary\_df = spark.createDataFrame(summary)  summary\_df.coalesce(1).write.mode("overwrite").json("/opt/qa/\_last\_qa\_results\_json")  summary\_df.coalesce(1).write.mode("overwrite").csv("/opt/qa/\_last\_qa\_results\_csv", header=True)  spark.stop()   * why: align Spark output path to the mounted host folder.  **2) Confirm compose mount** In docker-compose.yml for **spark-master** (and worker if you run on worker):  services:  spark-master:  # ...  volumes:  - ./data:/opt/data  - ./qa:/opt/qa # <-- required for Option A   * If I just added this line, I’ll recreate:   docker compose up -d --force-recreate **3) Re-run the job** Pick the way I normally run it (any one is fine):  **A. spark-submit inside container**  docker exec -it departuredelay\_spark\_master bash -lc "spark-submit /opt/work/qa/postIngestionDataCompare.py"  **B. python (PySpark) inside container**  docker exec -it departuredelay\_spark\_master bash -lc "python /opt/work/qa/postIngestionDataCompare.py"  **C. VS Code Remote-Containers / Terminal already attached**  spark-submit /opt/work/qa/postIngestionDataCompare.py  # or  python /opt/work/qa/postIngestionDataCompare.py  Replace /opt/work/qa/... with the actual in-container path where the repo is mounted (e.g., /opt/app/qa/...). The **output** path stays /opt/qa/.... **4) Quick verify (use /opt/qa, not /opt/data/qa)** docker exec -it departuredelay\_spark\_master bash -lc '  ls -lh /opt/qa/\_last\_qa\_results\_json;  ls -lh /opt/qa/\_last\_qa\_results\_csv;  echo "--- JSON PREVIEW ---"; head -n 5 /opt/qa/\_last\_qa\_results\_json/part-\*.json;  echo "--- CSV PREVIEW ---"; head -n 5 /opt/qa/\_last\_qa\_results\_csv/part-\*.csv  ' **5) Verify on Windows host** I expect new files here:   * C:\Git\DepartureDelays\qa\\_last\_qa\_results\_json\part-\*.json * C:\Git\DepartureDelays\qa\\_last\_qa\_results\_csv\part-\*.csv  **6) (Optional) Rename single part file to a friendly name** # JSON  Get-ChildItem -Path C:\Git\DepartureDelays\qa\\_last\_qa\_results\_json -Filter "part-\*.json" |  Select-Object -First 1 | ForEach-Object {  Rename-Item -Path $\_.FullName -NewName "last\_qa\_results.json" -Force  }  # CSV  Get-ChildItem -Path C:\Git\DepartureDelays\qa\\_last\_qa\_results\_csv -Filter "part-\*.csv" |  Select-Object -First 1 | ForEach-Object {  Rename-Item -Path $\_.FullName -NewName "last\_qa\_results.csv" -Force  }   * why: coalesce(1) still writes a part-\*.csv + \_SUCCESS. Renaming helps for human consumption.  **7) (Optional) Clean up the old wrong path inside container** docker exec -it departuredelay\_spark\_master bash -lc 'rm -rf /opt/data/qa/\_last\_qa\_results\_json /opt/data/qa/\_last\_qa\_results\_csv || true' **8) (Optional) .gitignore outputs** Add to .gitignore to avoid committing generated files:  qa/\_last\_qa\_results\_json/  qa/\_last\_qa\_results\_csv/ **🔎 Why your previous check showed “No such file or directory”** You verified under /opt/data/qa/.... With Option A we now write to /opt/qa/.... Use the updated verify block above (Step 4). **🧩 If you *prefer* to keep /opt/data/qa in code (Option B)**  * Revert code to /opt/data/qa/...   In compose:   volumes:  - ./data:/opt/data  - ./qa:/opt/data/qa   * Recreate containers and re-run. Use /opt/data/qa/... in the verify commands. |
| --- |

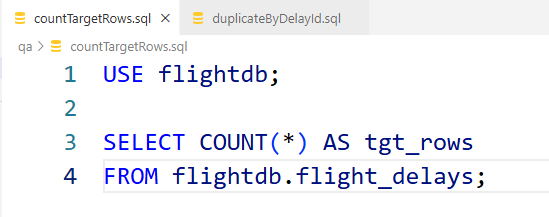
Your script prints clean markers like [QA] src\_count: ... to **stdout**, so you’ll see them in the **Terminal** (not the VS Code “Output” pane). You can also open the saved log and the machine-readable results:

| Show the **last run log**:  docker exec -it departuredelay\_spark\_master bash -lc 'tail -n +1 /opt/data/qa/\_last\_qa\_run.log' |
| --- |

| **Show the JSON/CSV summary files the script wrote:**  docker exec -it departuredelay\_spark\_master bash -lc 'ls -lh /opt/data/qa/\_last\_qa\_results\_json; ls -lh /opt/data/qa/\_last\_qa\_results\_csv'  docker exec -it departuredelay\_spark\_master bash -lc 'cat /opt/data/qa/\_last\_qa\_results\_json/part-\*.json'  You should see lines like:  [QA] src\_count: 1391578  [QA] tgt\_count: 1391578  [QA] tgt\_duplicate\_natural\_keys: 507  [QA] rows\_src\_minus\_tgt: 0  [QA] rows\_tgt\_minus\_src: 0 |
| --- |

### SQL QA

#### qa\countTargetRows.sql



USE flightdb;

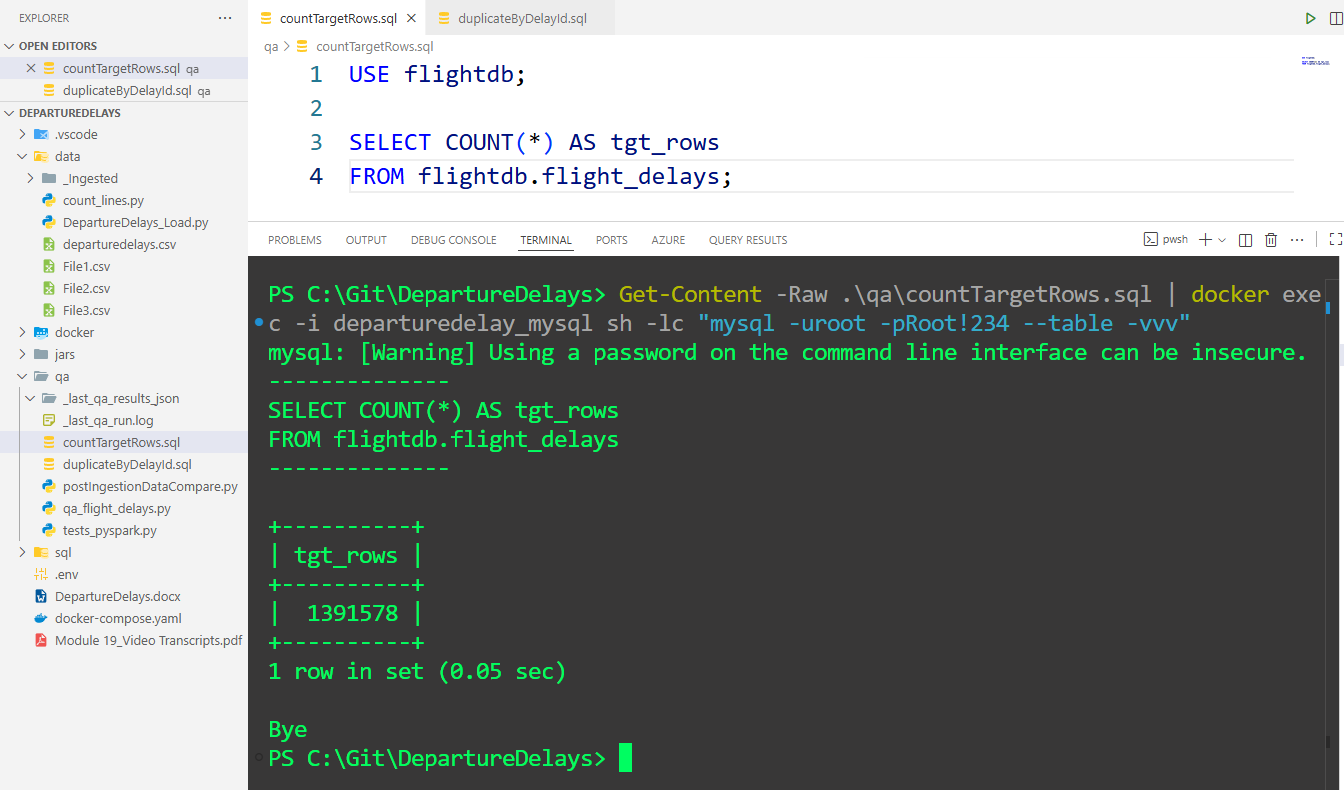
SELECT COUNT(\*) AS tgt\_rows

FROM flightdb.flight\_delays;

Run:

# run from C:\Git\DepartureDelays

Get-Content -Raw .\qa\countTargetRows.sql | docker exec -i departuredelay\_mysql sh -lc "mysql -uroot -pRoot!234 --table -vvv"



#### qa\duplicateByDelayId.sql

| This does not work as there is not output    USE flightdb;  SELECT DelayId, COUNT(\*) AS cnt  FROM flightdb.flight\_delays  GROUP BY DelayId  HAVING COUNT(\*) > 1; |
| --- |

USE flightdb;

SELECT COUNT(\*) AS duplicate\_groups

FROM (

SELECT DelayId

FROM flightdb.flight\_delays

GROUP BY DelayId

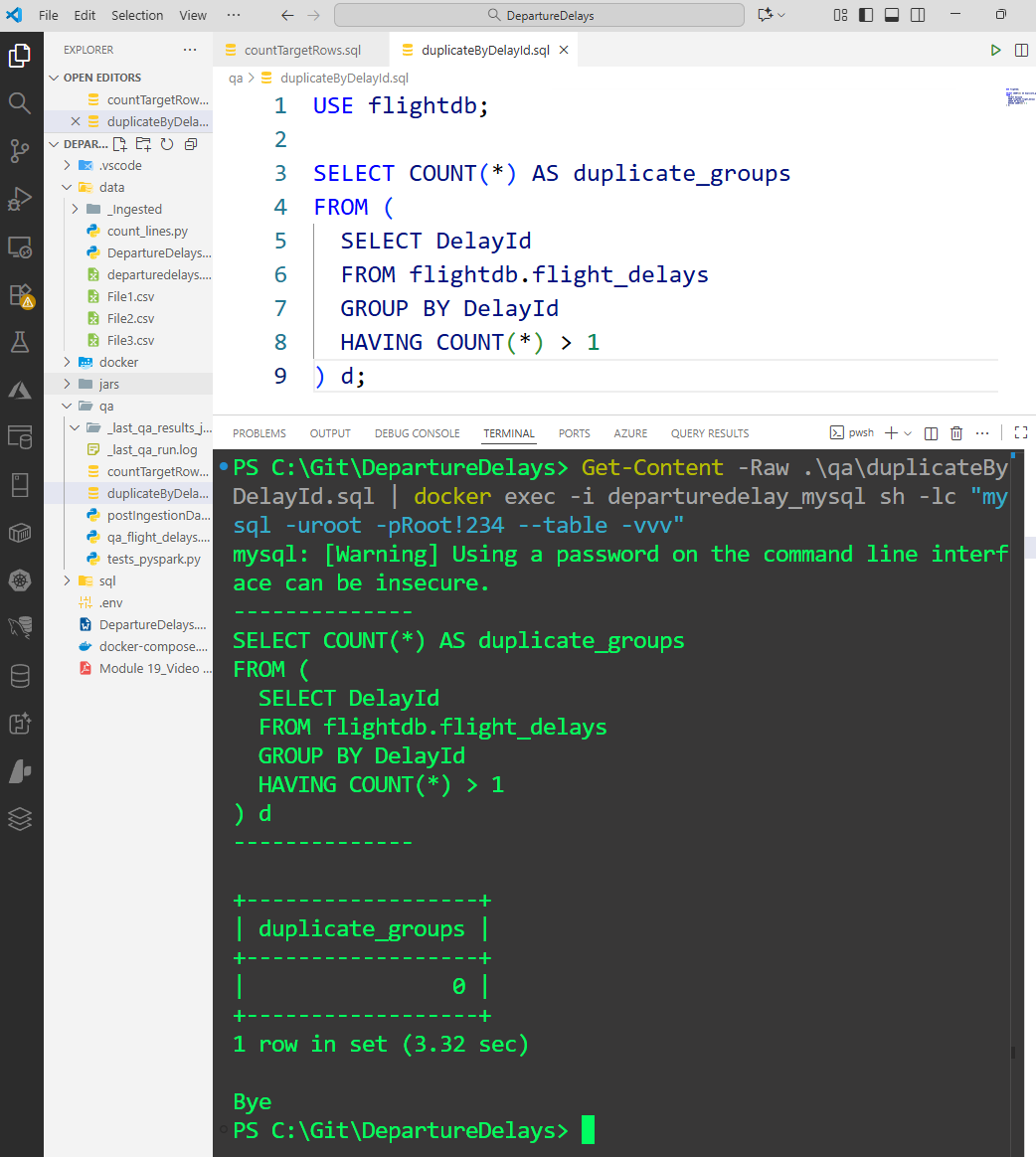
HAVING COUNT(\*) > 1

) d;

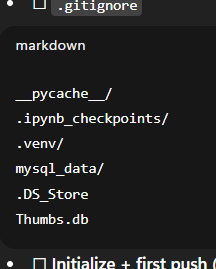
Run:

# run from C:\Git\DepartureDelays

Get-Content -Raw .\qa\duplicateByDelayId.sql | docker exec -i departuredelay\_mysql sh -lc "mysql -uroot -pRoot!234 --table -vvv"



# 🧰 5) Git — First push from VS Code, pull, push final

☐ **.gitignore  
**

\_\_pycache\_\_/

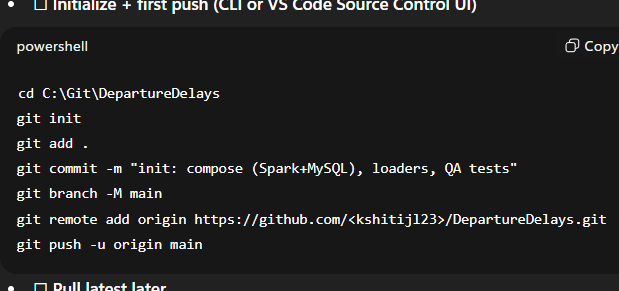
.ipynb\_checkpoints/

.venv/

mysql\_data/

.DS\_Store

Thumbs.db

☐ **Initialize + first push (CLI or VS Code Source Control UI)  
**

cd C:\Git\DepartureDelays

git init

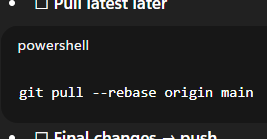
git add .

git commit -m "init: compose (Spark+MySQL), loaders, QA tests"

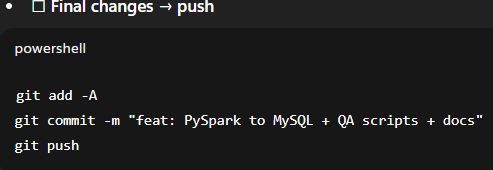
git branch -M main

git remote add origin https://github.com/<kshitijl23>/DepartureDelays.git

git push -u origin main

☐ **Pull latest later  
**

git pull --rebase origin main

☐ **Final changes → push  
**

git add -A

git commit -m "feat: PySpark to MySQL + QA scripts + docs"

git push

# 🌐 6) Links & Email Templates

**Spark Stack (local UIs)**

* + Master: http://localhost:8080
  + Worker: http://localhost:8081
  + Jobs: http://localhost:4040

**Email: Git repo created**Subject: DepartureDelays repo — Spark + MySQL + QA tests

Hi team,

I created the GitHub repo “DepartureDelays” with Docker Compose, PySpark loaders,

and QA scripts. Clone and run from C:\Git\DepartureDelays.

Repo: https://github.com/<kshitijl23>/DepartureDelays

Spark Master UI: http://localhost:8080

QA: docker exec -it flight\_spark\_master spark-submit /opt/qa/tests\_pyspark.py

— Krish

**Email: Docker stack (FlightDelays) is up** Subject: FlightDelays Docker (Spark + MySQL) ready

Spark Master UI: http://localhost:8080, Worker UI: http://localhost:8081.

Compose lives in C:\Git\DepartureDelays. Use spark-submit to run QA tests.

— Krish

# 🧪 7) Verification Checklist

☐ docker compose ps shows **flight\_spark\_master**, **flight\_spark\_worker\_1**, **flight\_mysql** running.

☐ PySpark queries match the screenshots (show(10) outputs).

☐ MySQL row count > 0.

☐ QA logs: src\_count == tgt\_count, tgt\_duplicate\_keys = 0, both minus counts = 0.

# 🧯 8) Troubleshooting (fast)

☐ **Image pull fails:** docker compose pull; confirm no VPN block to Docker Hub.

☐ **pyspark not found:** you’re likely in worker; run on **flight\_spark\_master**.

☐ **JDBC class not found:** ensure /opt/spark/jars/mysql-connector-j-8.4.0.jar.

☐ **Reset stack:  
**docker compose down -v && docker compose up -d

# 📦 9) What I Submit (zip + comment)

* **Zip:** C:\Git\DepartureDelays\FlightDelays\_KrishL\_v1.zip
  + Include: docker-compose.yaml, sql\01\_init\_db.sql, qa\tests\_pyspark.py, qa\tests\_sql.sql, README.md, and **screenshots** (UIs, PySpark results, MySQL count, QA output).

**Comment:**FlightDelays (Docker Spark + MySQL) with PySpark ingest and QA.

Repo: https://github.com/<kshitijl23>/DepartureDelays. See README for steps.

# 🔁 Change Requests (if I need edits later)

* “**Rename containers**”: change container\_name: values and name: at top of compose.
* “**Split stacks**”: I’ll create docker-compose.spark.yaml and docker-compose.mysql.yaml with tiny diffs.

==========================================

Normal

Title

SUBTITLE

# Heading 1

## Heading 2

### Heading 3

#### Heading 4

##### Heading 5

###### Heading 6