## Auto-Terminate Long-Running Databricks Job

### Scenario:

You have a **scheduled Auto Loader or Delta pipeline job**.  
Normally, it completes in **~20 minutes**.  
But occasionally, due to:

* cluster issues
* input data spikes
* hanging tasks

…the job runs **much longer** than expected.

### Goal:

If the job runs **more than 60 minutes**, automatically:

* Alert the team
* Cancel or restart the job
* Log diagnostic info for debugging

## Architecture Overview

1. **Databricks Job** runs a streaming or batch pipeline.
2. A **monitoring script** (scheduled via Databricks Workflow or external orchestrator like Airflow) checks job runtime.
3. If runtime exceeds a threshold (e.g., 60 min), it:
   * Sends a Slack/email alert
   * Cancels the job (optional)
   * Logs metadata to Delta for diagnostics

## Step-by-Step Solution Using Databricks REST API

### 1. ****Monitor Running Jobs via REST API****

python

import requests

import json

from datetime import datetime, timezone

# Replace with your workspace and token

DATABRICKS\_INSTANCE = "https://<your-workspace>.cloud.databricks.com"

TOKEN = "dapi...."

# Job ID to monitor

JOB\_ID = 1234

MAX\_RUNTIME\_MINUTES = 60

headers = {"Authorization": f"Bearer {TOKEN}"}

# Get list of active runs

resp = requests.get(f"{DATABRICKS\_INSTANCE}/api/2.1/jobs/runs/list?job\_id={JOB\_ID}", headers=headers)

runs = resp.json().get("runs", [])

if runs:

latest\_run = runs[0]

run\_id = latest\_run["run\_id"]

start\_time = datetime.fromtimestamp(latest\_run["start\_time"] / 1000, tz=timezone.utc)

duration = (datetime.now(timezone.utc) - start\_time).total\_seconds() / 60

if duration > MAX\_RUNTIME\_MINUTES:

print(f"Job {JOB\_ID} is running too long: {duration:.2f} minutes.")

# Optional: cancel the job

cancel\_resp = requests.post(

f"{DATABRICKS\_INSTANCE}/api/2.1/jobs/runs/cancel",

headers=headers,

json={"run\_id": run\_id}

)

print("Job canceled:", cancel\_resp.status\_code)

### 2. ****Automate this Script****

* Schedule as a separate Databricks **workflow** every 15 min
* Or trigger it from **Azure Logic App / Airflow / Lambda**

### 3. ****Extend: Log Issues to Delta Table****

python

CopyEdit

log\_df = spark.createDataFrame([{

"job\_id": JOB\_ID,

"run\_id": run\_id,

"start\_time": start\_time.isoformat(),

"duration\_minutes": duration,

"detected\_at": datetime.now(timezone.utc).isoformat()

}])

log\_df.write.mode("append").saveAsTable("monitoring.job\_runtime\_exceptions")

## Optional: Add Dashboarding

Use Power BI or Databricks SQL to visualize:

* Jobs exceeding runtime threshold
* Frequency of terminations
* Trends over time

## Summary

|  |  |
| --- | --- |
| **Component** | **Tool** |
| Job monitor | REST API |
| Alert/termination | Python script or workflow |
| Logging | Delta table |
| Dashboard | Power BI / DB SQL |