Continuous Jobs and File Triggers in Databricks

Ryan Chynoweth
6 min read · Mar 14, 2023

Databricks Workflows provide a robust set of orchestration capabilities natively in the product which eliminates the requirement for third party tooling, reducing complexity and costs on your modern data stack. Databricks recently released two new job triggers: continuous and file arrival.

Continuous jobs enable customers to keep a job constantly running by automatically starting a new job run when the previous run completes. The completion status of the job can be either success or failure. The file arrival trigger is a common requirement for when data is published to cloud storage and customers would like to automatically start a new job run. File arrival triggers are extremely common for event based processing systems.

We will provide details on creating jobs with both continuous and file arrival schedules. Please refer to the associated GitHub repository which contains the JSON configuration for the jobs and the required notebooks.

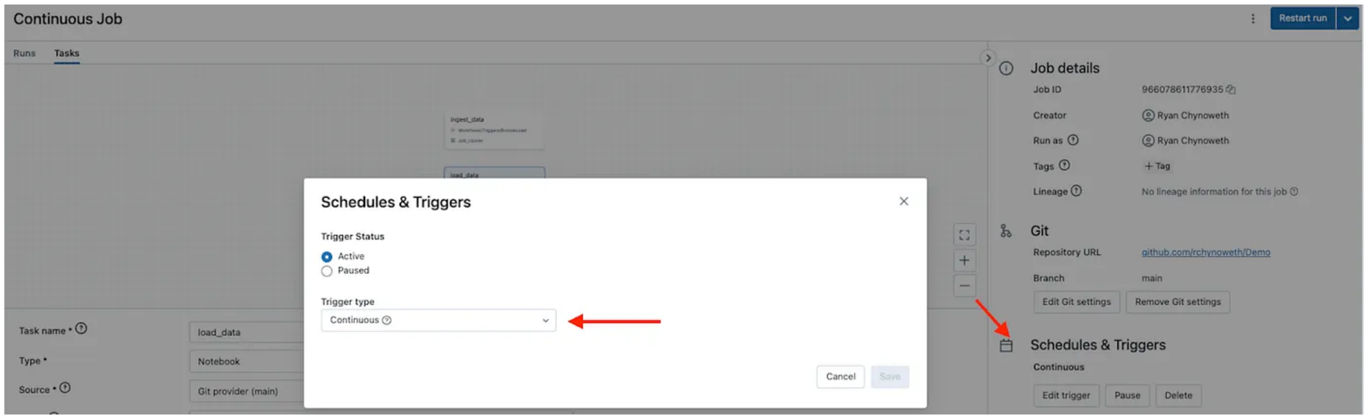
Create a Continuous Job

The most common scenario for using a continuous job schedule is running Spark Structured Streaming jobs. Since it is possible for jobs to fail due to a variety of reasons, such as memory issues or unexpected data schema, engineers ideally need to automatically restart the job with minimal downtime. Historically, users were required to leverage job retries that would automatically restart the job when it fails, but this is not an ideal experience for a variety of reasons.

Continuous jobs can be applied to batch jobs as well. Engineers may want to write a batch script and run it as quickly as possible to keep data up to date. For example, if we have a piece of code that is performing an entire overwrite of an aggregation table we can set this on a continuous schedule to automatically trigger a new job when the previous one finishes. This behavior is especially useful when you would like to run a job on a schedule of less than 60 seconds since the intent of the continuous schedule is to start the new job within that time frame. A continuous schedule is fairly unique, as most schedulers in the modern data stack are limited to running at most once a minute and require outside tooling or loops to run continuously. Databricks enables and abstracts the complexity of continuously running a job.

One huge benefit to continuous jobs is the reuse of compute instances from the previous job run, therefore, the provisioning time for Databricks clusters is cut drastically after the first job run is triggered. For continuous execution with the same compute, some customers would run jobs on an interactive cluster which was less ideal from a cost and managed compute perspective. This process is no longer required and customers can easily reconfigure their current jobs today without any code changes.

To create a continuous job please refer to the job configuration file located here. Please see below on how to select and set a continuous schedule for a job in Databricks Workflows. It is worth noting that the job below has two parallel tasks, which means that the job will be marked completed when all tasks are completed before kicking off the next job run. Lastly, if a job fails five times in a row, then Databricks will automatically pause the job to ensure that you are not caught in an infinite loop of retries while paying for unnecessary compute clusters.



Schedules & Triggers

Trigger Status

[✓ Active] Paused

Trigger type: Continuous

Task name: load_data
Type: Notebook
Source: Git provider (main)

Job details
Job ID: 966078611776935
Creator: Ryan Chynoweth
Run as: Ryan Chynoweth
Tags: + Tag
Lineage: No lineage information for this job

Git
Repository URL: github.com/rchynoweth/Demo
Branch: main

Schedules & Triggers
Continuous

Continuous Job Schedule UI

Create a File Trigger Job

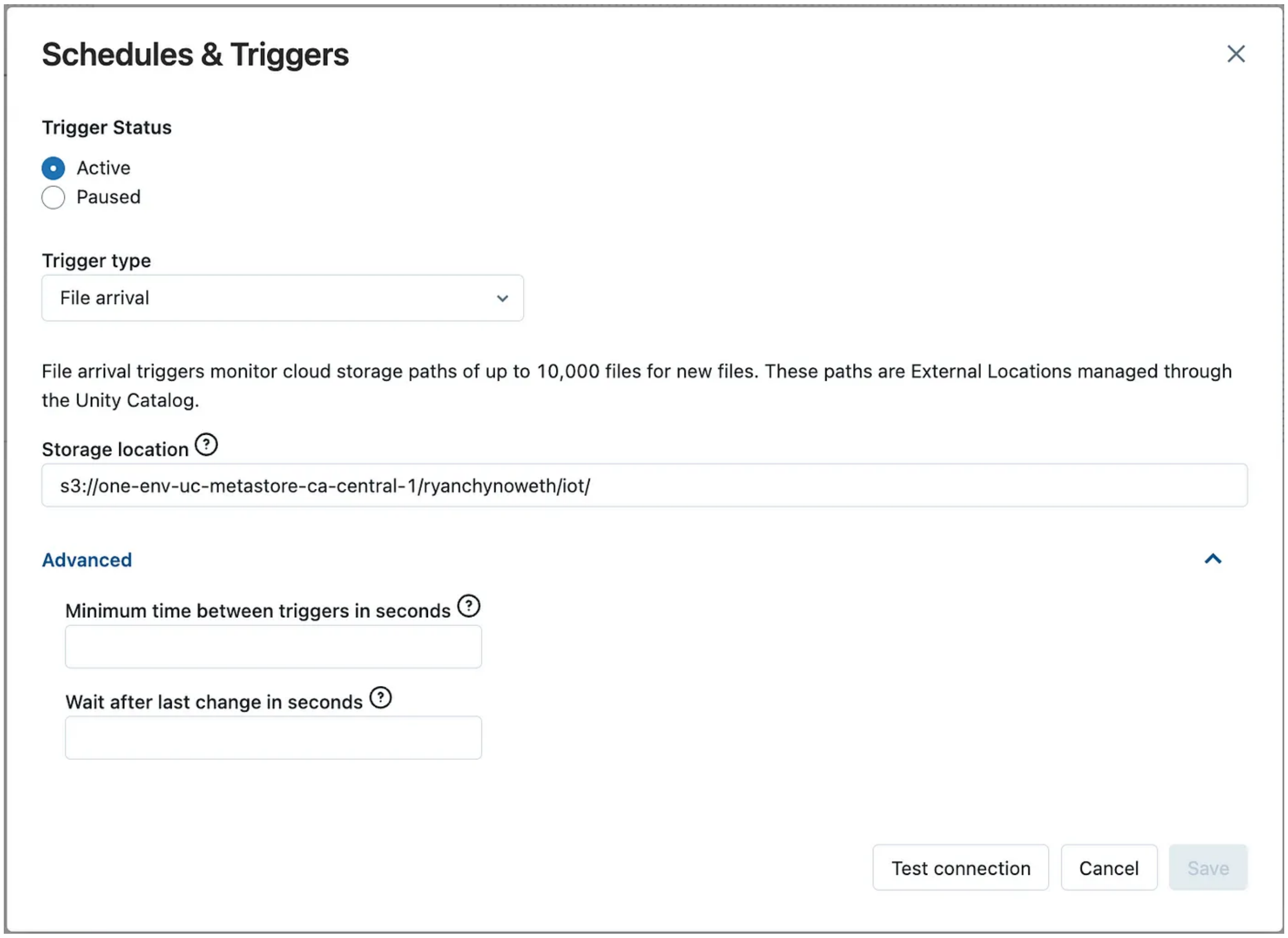
Customers will often use external tooling to connect to source systems (e.g. on-premise databases) and publish the data as a file to cloud object storage (e.g. S3). Once data is in cloud storage a Databricks job needs to be started to perform transformations on that data and load into a table.

Let's assume that a JSON file is published to cloud storage every 15 minutes and by nature there is slight variation in when the file arrives, let's say +/- 1 minute. Since the file being published is decoupled from the Databricks pipeline, the job in Databricks would have to accommodate the variability of data availability resulting in delayed processing (if data arrives early), skipped processing (if data arrives late), or reliance on another tool to trigger the job. With file arrival triggers these issues simply go away and allow engineers to natively build pipelines that process data as soon as it lands in storage.

When using the file arrival trigger the external location is checked at most once a minute but this can be configured as needed. For example, if it is possible for multiple files to be published in a short window of time, then you can provide a wait time between triggers to group processing of files in a single job run.

A limitation for file triggers is that it can only monitor an external location with less than 10,000 files which simply means that if you accrue a larger number of files you should edit the scope of file path to ignore older files. Please note that the 10,000 file limit applies to the path within the external location, therefore, if the external location is at the base directory of the cloud object store then the 10,000 limit can be applied at each subdirectory individually. It is common for raw data being published to cloud storage to follow a temporal directory structure (i.e. /yyyy/mm/dd/file.json) which would allow users to restrict directories for a given time period.

See below for a screenshot of the file trigger configuration.


Schedules & Triggers

Trigger Status

Active [✓]

Paused

Trigger type

File arrival

File arrival triggers monitor cloud storage paths of up to 10,000 files for new files. These paths are External Locations managed through the Unity Catalog.

Storage location

s3://one-env-uc-metastore-ca-central-1/ryanchynoweth/iot/

Advanced

Minimum time between triggers in seconds

Wait after last change in seconds

File Arrival Trigger UI

In order to leverage file triggers for a job, the workspace must be associated with a Unity Catalog (UC) metastore. The data sinks do not need to be registered in UC, however, the file arrival job schedule requires the storage credential and external location objects that are provided by UC. This enables Databricks to authenticate and connect to an external location that needs to be monitored for new files. We quickly discuss how to create those objects below.

Create a Storage Credential

To create a storage credential a user can use the Databricks UI as outlined in the documentation or the Databricks CLI. We will use the Databricks CLI to create the credential in AWS with the following command. Please note that you must be an account owner in order to create a storage credential.

```
 databricks unity-catalog storage-credentials create — name <credential
name> — aws-iam-role-arn <AWS IAM Role>
```

Create an External Location

With Databricks Unity Catalog, users can create external location objects to easily access data in cloud object storage. Prior to Unity Catalog this configuration was done using cluster configurations, therefore, each time a cluster was created it required a specific set of parameters to connect to an external storage. Unity catalog simplifies this process by enabling access controls at the account level and applying permissions to users and groups as opposed to clusters.

To create an external location we will use SQL, please provide the appropriate parameters to the query below.

```
 CREATE EXTERNAL LOCATION <location_name>
URL ‘s3://<bucket>’
WITH (STORAGE CREDENTIAL `<storage_credential_name>`);
```

This can also be done using the Databricks CLI with the following command.

```
 databricks unity-catalog external-locations create — name <location name> —
url <storage url> — storage-credential-name <credential name>
```

Conclusion

Both of the new schedule types in Databricks workflows seek to make the product even more simple and cost effective by enabling customers to build solutions with a single tool. We discussed how to use the file arrival and continuous triggers in Databricks Workflows. For a more detailed analysis of workflow features, I encourage you to read this blog by another Solutions Architect at Databricks.

Databricks Databricks Workflows Etl

Written by Ryan Chynoweth

Edit profile

312 Followers

Senior Solutions Architect Databricks — anything shared is my own thoughts and opinions

More from Ryan Chynoweth

DELTA SHARING

en standard for secure data sh

dustry's first open protocol for secure data sharing, mak
er organizations regardless of which computing platfor

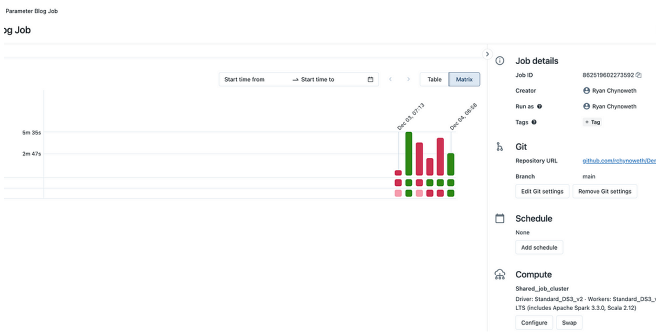
Ryan Chynoweth

Delta Sharing: An Implementation Guide for Multi-Cloud Architecture

Introduction

8 min read · 3 days ago

3 2


Parameter Blog Job

g Job

Start time from → Start time to

Table Matrix

Job details
Job ID 862519602273592
Creator Ryan Chynoweth
Run as Ryan Chynoweth
Tags + Tag

Git
Repository URL github.com/rchynoweth/Der
Branch main
Edit Git settings Remove Git settings

Schedule
None
Add schedule

Compute
Shared_job_cluster
Driver: Standard_DS3_v2 · Workers: Standard_DS3_
LTS (includes Apache Spark 3.3.0, Scala 2.12)
Configure Swap

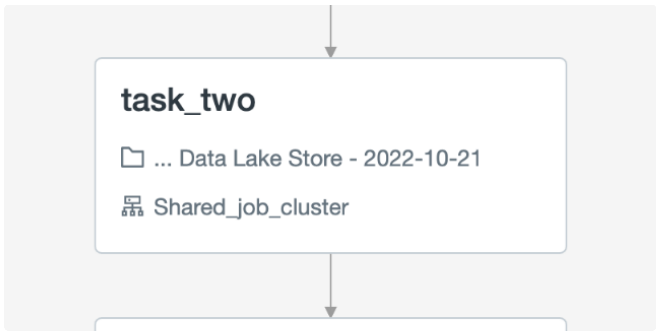
Ryan Chynoweth

Task Parameters and Values in Databricks Workflows

Databricks provides a set of powerful and dynamic orchestration capabilities that are…

11 min read · Dec 7, 2022

50 3



Ryan Chynoweth

Converting Stored Procedures to Databricks

Special thanks to co-author Kyle Hale, Sr. Specialist Solutions Architect at Databricks.

14 min read · Dec 29, 2022

116 5

Ryan Chynoweth

Recursive CTE on Databricks

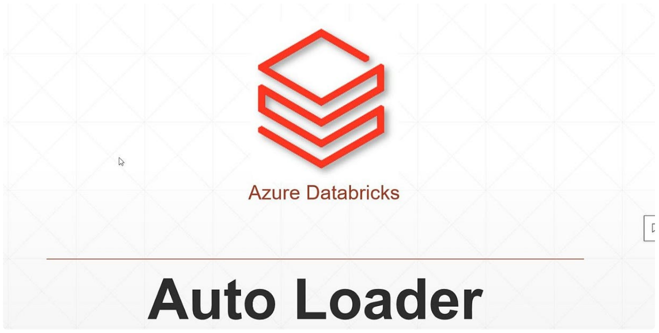
Introduction

3 min read · Apr 20, 2022

33

See all from Ryan Chynoweth

Recommended from Medium



SIRIGIRI HARI KRISHNA in Towards Dev

Auto Loader

Autoloader simplifies reading various data file types from popular cloud locations like...

5 min read · Dec 9, 2023

4 1

Daan Rademaker

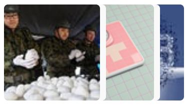
Do-it-yourself, building your own Databricks Docker Container

In my previous LinkedIn article, I aimed to persuade you of the numerous advantages o...

7 min read · Oct 16, 2023

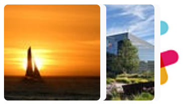
26

Lists



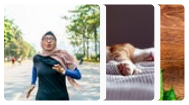
Staff Picks

547 stories · 597 saves



Stories to Help You Level-Up at Work

19 stories · 395 saves

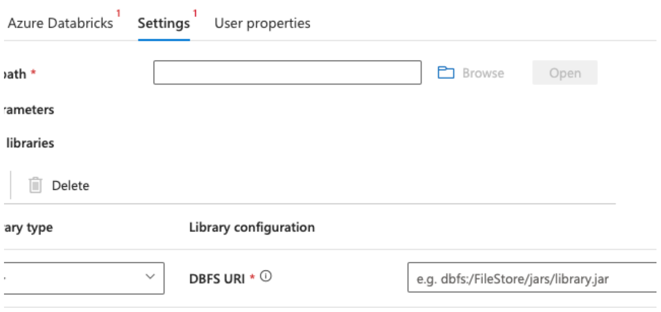


Self-Improvement 101

20 stories · 1146 saves

Productivity 101

20 stories · 1047 saves



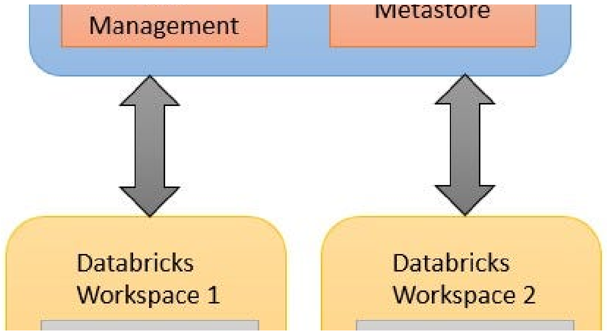
Matt Bradley

Azure Data Factory tips for running Databricks jobs

Following on from an earlier blog around using spot instances with Azure Data...

4 min read · Nov 2, 2023

14



Oindrila Chakraborty

Introduction to “Unity Catalog” in Databricks

What is “Unity Catalog”?

10 min read · Aug 14, 2023

18 2

Chetanya-Patil

Submitting Databricks job using DatabricksSubmitRunOperator in...

# Create and trigger a one-time run using operators

2 min read · Dec 1, 2023

Nnaemezue Obi-Eyisi

Unveiling the Secrets: External Tables vs. External Volumes in...

While reviewing the Databricks documentation about Unity Catalog, I came...

· 7 min read · Sep 25, 2023

5

39 1

See more recommendations