**Approach 1:** Utilizing Published Event Logs for Audit Tracking

* Databricks now enables publishing DLT pipeline event logs to the Metastore, making them accessible to all users. This overcomes the limitation of directly accessing the event log API, which is restricted to the respective pipeline owner.
* We can retrieve table-level execution details at the Bronze layer from the published event logs by leveraging **DLT Pipeline Id** and **DLT Run ID**.
* Generic Workflow:

**A diagram of a diagram

AI-generated content may be incorrect.**

* Based on the above generic workflow, the following explains the Bronze notebook task and its role in implementing the audit mechanism for this approach.

A diagram of a diagram

AI-generated content may be incorrect.

* After triggering the DLT pipeline, the Bronze Notebook task will remain in a running state until the execution of the DLT pipeline completes. This will be achieved by continuously polling the DLT API in a loop to monitor the pipeline's status.
* Once the DLT pipeline execution ends, logs for the current run will be retrieved from the published event logs. The extracted logs will then be written to the **PAC\_Bronze\_Audit** table from Bronze notebook.

**Example –**

****

**Limitations of Approach 1:**

* The Bronze notebook task will remain in a running state until the associated DLT pipeline completes execution. This leads to additional costs as one cluster remains active for monitoring the DLT pipeline run, while another cluster runs the actual DLT pipeline execution.

**How Event logs get published?**

* The owner of the pipeline can publish the event log as a public Delta table by toggling the Publish event log to metastore option in the **Advanced** section of the pipeline configuration. We can optionally specify a new table name, catalog, and schema for the event log. Refer the below screenshot.

A screenshot of a computer

AI-generated content may be incorrect.

* When specifying the table name, schema, and catalog, a physical Delta table is created in Unity Catalog. Logs can then be extracted from this table and written to the actual audit table, as demonstrated in Approach 1.
* If a new table name, catalog, and schema are not specified, then a table named as ‘event\_log\_<pipeline\_id>’ will get created in the default schema and catalog named set in the pipeline configuration.

A screenshot of a computer

AI-generated content may be incorrect.

**Approach 2:** Implementing custom audit logic

**A white rectangular box with black text

AI-generated content may be incorrect.**

* The goal is to track and log table level execution details, including Job\_id, Job\_Run\_id, Task\_Run\_id, Table\_Name, Start\_Date\_Time, End\_Date\_Time, Status, and Errors encountered during the DLT pipeline execution. These details will be captured dynamically and stored in the Job\_Task\_Audit table.
* Extract job-level metadata such as job\_id, job\_run\_id, and task\_run\_id from Databricks pipeline configurations. These values in the pipeline configuration will be dynamically updated with the current run IDs in each run by making an API call from the Bronze notebook task before triggering the DLT execution.
* Record timestamps for Start\_Date\_Time and End\_Date\_Time when the execution begins and ends. Capture the Table\_Name being processed and its corresponding status. Log any errors encountered during execution.
* In this approach, we need to maintain two tables: one DLT table and one Delta table. The DLT table is required because, in the event of a failure, the pipeline will not run unless at least one active flow exists in the DLT pipeline run.
* The DLT table stores the latest audit information for the current run and then data from the DLT table gets written to the non-DLT table.

**Limitations in this approach:**

* If a DLT pipeline fails before 'setting up the table' (due to syntax or cluster issues), logging the event in the audit table is not possible.
* DLT table for each pipeline. This is because a DLT table created in one pipeline cannot be used in another.
* We cannot insert logs of every DLT pipeline directly into one DLT table, so a non-DLT table is used instead to combine the logs of all DLT pipelines.