Instruction for 2026 Lakeflow Community Connectors Hackathon

Intro

Lakeflow community connectors are packaged as Python source code that defines a configurable SDP (Spark Declarative Pipeline). Through a simple UI flow, users can pull connector source code from a Databricks-Labs repository, create an SDP pipeline, and run it to ingest data from supported sources.

The Python source code consists of four parts:

1. **Source connector implementation**, following a predefined API
2. **Pipeline spec**, defined as a Pydantic class
3. **Configurable ingestion pipeline definition**
4. **Shared utilities and libraries** that package the source implementation together with the pipeline

Developers only need to implement or modify the source connector logic, while connector users configure ingestion behavior by updating the pipeline spec.

Developers for this Hackathon are welcome to improve the common shared library and the pipeline definition if these changes can benefit other source connectors.

Development FAQ:

FAQs: 2026 Lakeflow Community Connectors Hackathon

#2026-community-connector-hackathon

Use the Connector

Pre-build connectors

On Databricks main page, click **"+New"** → **"Add or upload data"**, and then select the source under **"Community connectors"**.

Steps:

1. Click **"+ Create connection"** to set up a new UC connection, **OR** select existing UC connection, and then click **"Next"**.
   a. If creating new, follow the documentation link to get more information on what parameters required for the UC connection.
   b. Do not update the pre-populated **"SourceName"**.
2. Set up ingestion pipeline by configuring **"Pipeline name"**, **"Event log location"** and **"Root path"**, and then click **"Create pipeline".**
   a. **Event log location**: the catalog/schema in UC to store the SDP's event log, as well the default catalog/schema for the ingested tables.
   b. **Root path**: the workspace directory to clone and store the community connector source code.
3. Configure ingestion pipeline spec in SDP editor.
   a. In "ingest.py", update the **"objects"** fields to include tables to be ingested.
4. Run the pipeline or schedule the pipeline similar to other SDPs.

Customize your connector

On Databricks main page, click **"+New"** → **"Add or upload data"**, and then select the **"+ Add Community Connector"** under **"Community connectors"**.

Steps:

1. In the pop-up window, configure **"Source Name"** and **"GitHub Repository URL"**, and then click **"Add Connector"**
   a. **Source Name:** The name string of your customized source
   b. **GitHub Repository URL:** The github url where the connector source code is hosted.
2. Follow the instructions of **Pre-built connectors** above from step 2.

Git Repo

lakeflow-community-connectors: master

Developers of this hackathon should clone the databrickslabs repo and add the source implementation under **sources** directory.

After the review and evaluation, these connectors will be pushed to the main repo so that external users can use them.

Development Guide

From a high-level, there are 4 steps to develop a connector:

1. **Understanding the source:** The goal of this step is to gather all relevant information about the source system (for example, APIs, authentication mechanisms, schemas). This information is used to implement the interface functions required to retrieve data from the source. **A structured template is provided to ensure all important source details are captured consistently**.
2. **Generate Python connector implementation:** Based on the information collected in the first step, developers implement the required API functions defined by the connector interface.
3. **Test-and-fix loop**: This step validates the connector implementation and fixes any issues. Standard test suites are provided, and developers can add additional source-specific tests as needed. **At least some tests must execute against a real source system rather than relying solely on mocks**.
4. **Generate public documentation**: Once the connector is functionally complete, public-facing documentation is generated using a documentation template. This documentation **should follow the provided the template** and explains how to configure and use the connector, helping users adopt it quickly and correctly.

API to Implement

Note: We recommend to use our template to simplify the implementation of Python Data Source API. However, developers can also choose to directly implement Python Data Source API as long as the implementation meets the API contracts of community connectors. See more details in this doc: Lakeflow Community Connectors APIs

Lakeflow community connectors leverage the Python Data Source API as the underlying foundation. Rather than exposing the low-level Python source interfaces, an abstraction layer is built on top to simplify development and automatically assemble a complete SDP pipeline.

See more details here.

```python
class LakeflowConnect:
    def __init__(self, options: dict[str, str]) -> None:
        """
        Initialize the source connector with parameters needed to connect to the source.
        """

    def list_tables(self) -> list[str]:
        """
        List names of all the tables supported by the source connector.
        The list could either be a static list or retrieved from the source via API.
        """

    def get_table_schema(
        self, table_name: str, table_options: dict[str, str]
    ) -> StructType:
        """
        Fetch the schema of a table.
        """

    def read_table_metadata(
        self, table_name: str, table_options: dict[str, str]
    ) -> dict:
        """
        Fetch the metadata of a table.
        """

    def read_table(
        self, table_name: str, start_offset: dict, table_options: dict[str, str]
    ) -> (Iterator[dict], dict):
        """
        Read the records of a table and return an iterator of records and an offset.
        The read starts from the provided start_offset.
        Records returned in the iterator will be one batch of records marked by the offset as its end_offset.
        The read_table function could be called multiple times to read the entire table in multiple batches and
        it stops when the same offset is returned again.
        If the table cannot be incrementally read, the offset can be None if we want to read the entire table in one batch.
        We could still return some fake offsets (cannot checkpointing) to split the table into multiple batches.
        """
```

Pipeline Spec

The community connectors utilize a pipeline spec that is highly similar to the one used for our existing managed ingestion pipelines.

Below are the fields of the spec:

- **connection_name**: Required. The name of a UC connection in string.
- **objects**: Required. Specifying tables to ingest and the destination for these tables in array.
  - **table**: The type of an object.
    - **source_table**: required.
    - **destination_catalog**: optional. If not set, tables will be ingested into the default catalog configured in pipeline setup above.
    - **destination_schema**: optional. If not set, tables will be ingested into the default catalog configured in pipeline setup above.
    - **destination_table**: optional. If not set, the destination table name will be the same as the source table.
    - **table_configuration**:
      - **scd_types**: “SCD_TYPE_1”, “SCD_TYPE_2”, “APPEND_ONLY”
      - Other source-specific optional or required parameters. See documentation of the specific source connector.

Vibe-Coding using Cursor

Instructions are stored under the `prompts/` directory in the Git repo. When using Cursor, developers can ask it to reference these instructions and complete a specific step. **It’s best to keep a human in the loop: have Cursor complete one step at a time, then review the output and make any necessary adjustments before moving on.**

Below are sample prompts developers can use:

- **Step 1 (understand the source)**
  Please refer to `@prompts/vibe_coding_instruction.md` and execute **Step 1 only**. Use this `<link>` to collect raw information about the source API.
- **Step 3 (implement the connector)**
  Refer to `@sources/<your_source>/source_api_doc_template.md` and execute **Step 3** defined in `@prompts/vibe_coding_instruction.md` to implement the connector.

For more details, please see the instructions in this.

Use Claude Code

TO BE ADDED. STAY TUNED.

Appendix

PRD: Lakeflow Connect Community Connectors

[Design Doc] Lakeflow Connect Community Connectors

Instruction of Lakeflow Community Connectors