Labs Proposal: Zerobus - file mode (Previously named Managed File Push)

[Searching for a project name in cc google should yield no prior results. Prod.workload_insights table should not have mentions of the same project]

We would like to provide a code-free and one-stop solution to allow customers to continuously upload structured data files to Databricks and automatically convert them to delta tables. One Pager

Owners	Chi Yang Chavdar Botev	Status	Design sketch
Product (team)	Elise Georis	Artifact	DAB
Repository name	link	Language	Python
Target release	2025-8-31	Go link	go/ingestion/filepush

Please fill all of the fields before submitting a labs project. This document is a lightweight mix between http://go/designdoctemplate, http://go/prd, and a launch checklist. It is meant to make a more streamlined high-quality project for labs.

Review status

[Instructions]

Reviewer (team)	LGTM blockers [?]	LGTM	Updated
@Name (Labs)		☐	YYYY-MM-DD
Elise Georis (PM)		☑	Jul 24, 2025
Alex Khachaturian		☐	LPP-XXX
@Name (Security)		☐	SDR-XXX

Stakeholders include someone that represents the customer(s) of this project and SMEs of any top-level architecture or processes that this project proposes changing.

Page: Everything You Need to Know About Security Design Reviews (SDRs) (go/sdr)
Page: SDR Process and Faqs | SDR FAQs

Part 1: Request for comments

Customer evidence

[We're not publishing labs just for one customer. To publish a lab project, we need at least 10 customers who want it.]

☑ ~~More than 10 customers want it right now~~

Tl;dr

- Many customers write files to cloud storage with the sole goal of ingesting them into Databricks. This staging is an unnecessary step, and it introduces a ton of pain around cloud storage and Auto Loader.
- We spoke with **30+** customers who want to push files directly to an API that abstracts the staging location + pipeline—avoiding management of cloud storage and Auto Loader entirely. Details >>
    - (Auto Loader remains a priority, and we're still improving it in parallel.)
- This complements the Zerobus record mode that's in PrPr; we intend to position these as the same product, with a file mode and a record mode.

Details

- **The primary value prop of push-based file ingestion is simpler architecture and lower the management costs.**
    - *"The lion's share of our compute is shuffling data around from source system to source system... If we could guarantee that data could always land, and we could skip S3 directly... as long as it costs less than the infra now, we'd be willing to pay for it."* - Nike
    - *"The reason we want to pursue this is to reduce costs. Get rid of the things in between. Cost takes priority."* - ShareChat
    - *"The fewer icons, the better."* - Cherry Baekert
    - *"In an ideal world, we'd ingest the data from the services into a pipeline that's managed for us - and write into the bronze table."* - Skyscanner
    - *"We'd like a more managed process that's directly integrated with Databricks, versus first doing Azure, then calling a DBX job."* - Calamos
- **Customers also see it as an opportunity to decentralize development.**
    - *"We want to delegate to the home teams as much as possible. If they can write to one data destination that we manage—and the SLAs, validations, governance etc. are on them—that's an evolution of what we're currently doing. We're too much at the center, managing everything."* - ADP
    - *"We want to make the central data platform lightweight. To push the intricacy of the logic of transformation, etc. down to the producer. Even the event-to-table contract should be on their side, effectively."* - ShareChat
- **Surprising learning: they may even be able to work with third-party data producers to push files to the API.**
    - *"Today, third parties like Sprinklr land it in S3, and we pick it up from there. But they've been asking for an API where they can push data to."* - United
    - *"We have a good working relationship with the source teams."* - Helen
- **For Databricks, this is an opportunity to mitigate the shift-left risk. Providers like Confluent or Fivetran are writing directly to UC-managed Delta or Iceberg—and**

bypassing Auto Loader or COPY INTO entirely. By introducing this API, we can at least make it as easy and price-competitive to use Databricks for that same job.

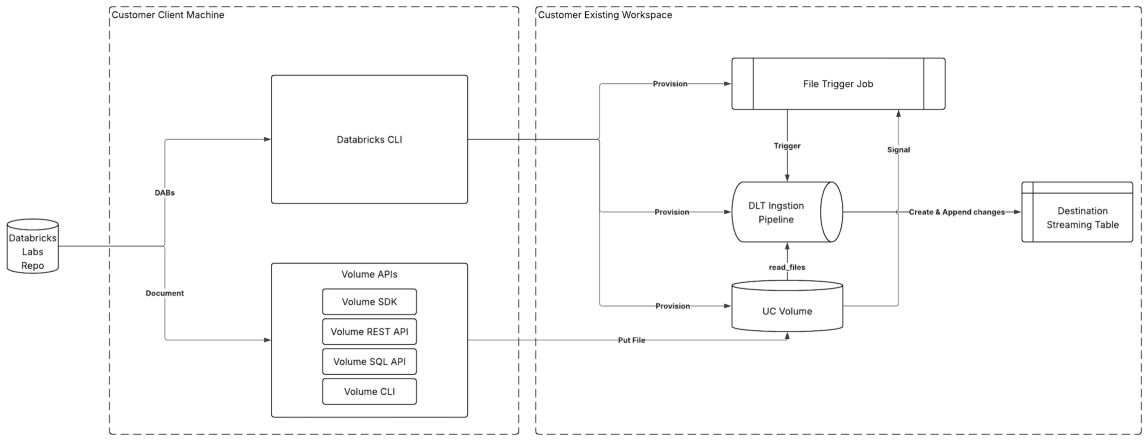
Go-To-Market (GTM) Plan

- ☐ **Blog(s)** Launch blog
- ☐ **Webinar(s)** Data Engineering Big Rocks
- ☐ **Present at conference(s)** DAIWT
- ☐ **Present to partner(s)** Partner PAB
- ☐ Publish to a 3rd party registry
- ☐ **Develop and deploy a Sales Play** Working with industry solutions team
- ☐ **Other** Standard sales enablement materials (INVESTech, internal guides, L100s, etc.)

Markitecture

[Sell this architecture]

- ☑ ~~Diagram is here~~



Graduation strategy

There's demand for a dedicated API, and we hypothesize that this has a distinct spot in our ingestion product matrix. However, before we build yet another ingestion API, we need to validate

that and better understand customers' use cases at scale. Taking an incremental approach balances this need for iteratively improving the product without overloading the ingestion stack. If this is validated, we intend to build a managed API as part of the Zerobus product line.

1. **Provide a notebook that approximates the file push CUJ.**
    - No new APIs.
    - Notebook creates staging.
2. **Grow adoption via GTM and iterate quickly based on feedback—adding features based on a stack rank of customer demand.**
    - Auto staging
    - Push client
    - ...
    - See draft PRD here.
3. **Pending customer demand, build a fully managed API and launch this as part of the Lakeflow Connect Zerobus product line.**

[All Labs should aim to be of high enough quality that it can easily be attached to the platform and provide a user experience that is indistinguishable from the features which are delivered by the product itself. Some of these projects may eventually make it into the product itself. One example - is the terraform provider, another example is dbx.

Please describe why this part of the product is missing and what gaps it fills. Mention the long-term strategy.]

- ☑ ~~You are motivated to keep the project healthy in the long term. See http://go/labs/healthy for more details.~~
- ☑ ~~At least one of the engineering teams could potentially take over this project into the main product~~
  ~~Ingestion team will~~
- ☑ ~~At least one of the Product Managers support the idea of this project~~
  ~~Ingestion team PM Elise~~
- ☐ Labs guide and PM sign off on requirements and proposal

Project kickoff

- ☑ ~~Add your doc to the projects folder.~~
- ☑ ~~Submit this doc to labs@databricks.com.~~

Administrative actions

These actions are performed by your Labs guide. Find one after PM and engineering counterparts sign off on the proposal. SLA is 2 weeks.

- [ ] Request labs-oss@databricks.com to create a repository in https://github.com/databrickslabs for the project.
- [ ] Request labs-oss@databricks.com to enable LABS_PYPI_TOKEN for a new repository.
- [ ] The project has a GitHub team added https://github.com/orgs/databrickslabs/teams
- [ ] **Collaborators and teams:** GitHub team has the Maintain rights on the project repository
- [ ] **Collaborators and teams:** Owner of the project has the ability to add more people to the project team
- [ ] **Branches:** The main branch has a protection rule
- [ ] **Branches:** The main branch requires a pull request before merging
- [ ] **Branches:** Restrict who can push to the main branch: Organization administrators, repository administrators, and users with the Maintain role (default)
- [ ] **Tags:** There's tag protection rule for **v***.
- [ ] **Actions / General:** Require PR approval for first-time contributors

Part 2: Design sketch

[Instructions]

- [ ] Labs reviewer and PM sign off on requirements and proposal
- [x] ~~Submit an SDR ticket for mandatory security and other reviews~~
  ~~Created: Atlassian~~
- [ ] Required stakeholders sign-off on Design sketch

Today if customers want to ingest files, they will need to configure a bucket from cloud providers to use as file staging location, then create a DLT pipeline and write code to configure file ingestion. They will also need to manage the folder structure of the staging location. This CUJ involves jumping between the cloud provider's web console and Databricks' workspace, and probably communicating with the organization's administrator.

With this project, customers just need to run a command to create the destination delta table, and then put the files to the designated path returned by the command. Within a few minutes, the data in the file will be ingested to the delta table, as structured rows. They could then leverage the full power of Databricks for analytics on the file data.

[Instructions, you can use https://mermaid.live/ and markdown integration in GitHub to version control the diagrams, you can also use https://excalidraw.com/ to make lo-fi diagrams to copy into these documents]
A DAB template is introduced to create necessary resources for managed file push:

1. A UC volume for staging the file
2. An Ingestion Pipeline to read files from the staging location and convert to Delta
3. A job with file arrival trigger to start the pipeline when new file is detected

In addition, we will provide the helper scripts to help customer interact with the DAB:

1. Guide customer to set the necessary parameters and deploy the DAB
2. Put file to volume, given a full qualified table name
3. Embed spark conf to help us track the usage (see Project Tracking section)
4. DAB resource clean up

Requirements

- Provide a solution to serve users with only the following knowledge:
  - SQL concepts (e.g. catalog, schema, table) and writing SQL queries
  - The files to upload and to convert to Delta tables
- Provide a boilerplate for user to quickly iterate through Autoloader configurations and fix the parsing issues

Proposed CUJ

Prerequisite

1. A workspace that has opt-in the following features
   a. Unity Catalog
   b. Serverless Compute
2. A host machine with the following dependencies installed and configured
   a. Databricks CLI
   b. Python 3
3. A catalog with managed location to hold the file push schema and other resources

Create Table & Push File

Step 1 - Define a file push schema

```shell
# Usage: create_filepush_schema <catalog_name>.<schema_name>
python create_filepush_schema.py filepush_catalog.filepush_schema
```

Step 2 - Upload file to table

Only need to specify the full qualified table name.

```shell
# Usage: push_file_to_table <catalog_name>.<schema_name>.<table_name>
<file_path>
python push_file_to_table.py filepush_catalog.filepush_schema.newtable
~/Downloads/prodfood_all_pipelines.csv
```

Debug Table

Step 0 - Inspect the destination table content

Wait for the table to be created and inspect the ingested data. If no data is presented, or if the data is malformatted, continue the following steps.

Step 1 - Open the read_files kernel file for the table

```shell
# Usage: debug_table.py <catalog_name>.<schema_name>.<table_name>
python debug_table.py filepush_catalog.filepush_schema.newtable
```

This opens the workspace editor page to the table's SQL kernel file.

Step 2 - Iterate until the data is parsed correctly

The following script can be run with any SQL warehouse, independent of the DLT pipeline.

```sql
-- filepush_schema_newtable_readfiles_kernel.sql
SELECT
```

```
  *
FROM
  read_files(

'/Volumes/chi_catalog/filepushexperiment/filepushexperiment_volume/newtable/',
    ignoreCorruptFiles => 'true', -- If a different malformed file pushed and
RocksDB has it enlisted, this will ignore the error when the file is deleted.
    ignoreMissingFiles => 'true' -- If a different file format is accidentally
pushed and RocksDB has it enlisted, this will ignore the error when the file is
deleted.
    -- Do not change anything above
    -- Add any additional options below
    -- Example:
    ,
    -- header => 'true',
    escape => '"'
  )
```

Step 3 - Save the change and trigger a full refresh on the table

Full refresh to fix the data that is already ingested.

Shell

```
# Usage: fullrefresh_table.py <catalog_name>.<schema_name>.<table_name>
python fullrefresh_table.py filepush_catalog.filepush_schema.newtable
```

Update Table

This should be the same as the Debug Table CUJ. Users are supposed to interact with the SQL concept only. This is a limitation on purpose.

Delete Table

Shell

```
# Usage: delete_table.py <catalog_name>.<schema_name>.<table_name>
python delete_table.py filepush_catalog.filepush_schema.newtable
```

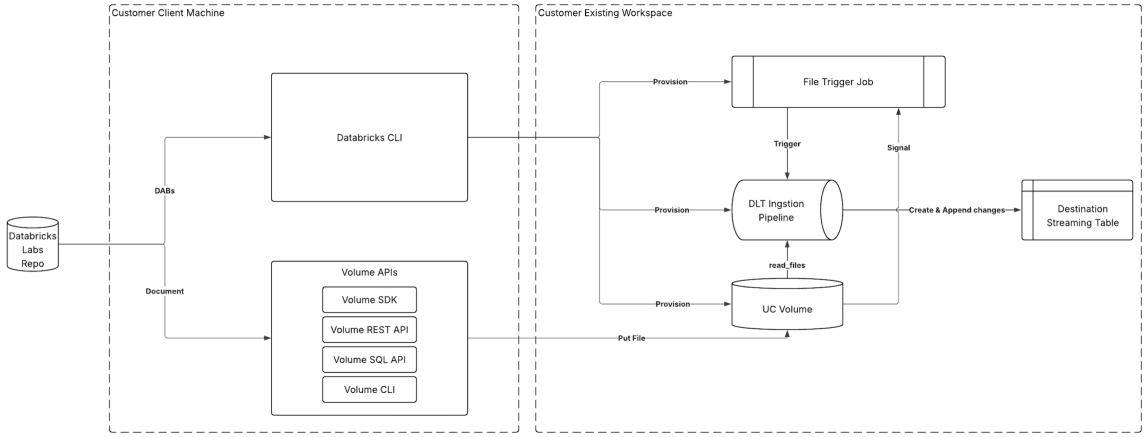
Cleanup

Drop all deployed resource, including all created tables and uploaded files.

```shell
# Usage: drop_filepush_schema.py <catalog_name>.<schema_name>
python drop_filepush_schema.py filepush_catalog.filepush_schema
```

Release Criteria

Detail Design



With the DAB being deployed, 4 types of resource will be created, and they have the following mapping relationships and functionalities.
Schema : Pipeline : Job : Volume = 1 : 1 : 1 : 1

- A schema that acts as a boundary of resources, and to hold all file-push-created tables.
- A volume that serves as the staging location of user data files.
- A DLT pipeline to manage all tables under a designated schema. It will scan for new table folders and create append flows that target individual tables.
- A job that is configured with the file trigger to monitor the new files of all tables under the data path.

Besides these, an upload client SDK would be provided in the DAB package. The client SDK does **not** need to have knowledge of the DAB.

Autoloader Kernel File

A SQL file that could be run isolated from the DLT pipeline. This acts as a "boilerplate" for customers to quickly debug Autoloader configuration issues. Once they finish the exploration, save the file and the change can be reflected in the destination table after a full table refresh.

Volume Folder Structure

```
None
schema_volume/
├── data/
│   ├── table1/
│   │   ├── table1_data1.csv
│   │   └── table1_data2.csv
│   └── table2/
│       └── table2_datax.csv
├── kernel/
│   ├── schema_volume_table1_readfiles_kernel.sql
│   └── schema_volume_table2_readfiles_kernel.sql
└── baddata/
│   ├── table1/
│   └── table2/
└── archived/
    ├── table1/
    └── table2/
```

Purpose of immediate directories

- data/
  - Holds raw uploaded data files
  - The upload client will create new directories if the table does not exist
  - The file trigger will monitor this directory. This should capture all old or new tables
- kernel/
  - Holds Autoloader kernel script file
  - Debug script will open corresponding kernel file
- baddata/
  - Holds data that fails the criteria, e.g. corrupted files
- archived/
  - Holds data moved by CleanSource

Volume Path Property

Record the volume path information to the following objects, this way the client SDK is decoupled with the DAB config, and could resolve the upload path "statelessly"

1. Schema, as DBPROPERTIES

About this schema

Name	filepushexperiment
Catalog Name	chi_catalog
Owner	chi.yang@databricks.com
Properties	filepush: volume_path: "/Volumes/chi_catalog/filepushexperiment/filepushexperiment_volume/"

2. Pipeline, as configuration

Pipeline settings UI | JSON | YAML

```
{
  "id": "0166f103-39eb-49f0-8a50-148d2e1ba82a",
  "pipeline_type": "WORKSPACE",
  "name": "filepushexperiment_pipeline",
  "configuration": {
    "filepush.volume_path": "/Volumes/chi_catalog/
    filepushexperiment/filepushexperiment_volume/"
  },
```

3. Table, as table property

Type	STREAMING_TABLE
Table Id	1285e114-92e4-4b55-b0c7-75f65a92b995
Pipeline Id	0166f103-39eb-49f0-8a50-148d2e1ba82a
Created At	Aug 19, 2025, 05:45 PM
Created By	chi.yang@databricks.com
Updated At	Aug 19, 2025, 05:45 PM
Updated By	chi.yang@databricks.com
Delta Runtime Properties Kvpairs	(empty)
Properties	pipelines: pipelineId: "0166f103-39eb-49f0-8a50-148d2e1ba82a" filepush: volume_path: "/Volumes/chi_catalog/filepushexperiment/filepushexperiment_volume/newtable/"

Dynamic Table Discovery

In every run, the Lakeflow pipeline source code performs the following operations:

1. Scans for all directories in the data folder, and use the directory name as table name
2. Initiates or read kernel file for each of the immediate sub-directory
3. Run flows for each of the immediate sub-directory

Limitations

Interference with CleanSource

Cleansource may empty the table data folder, causing the DLT source code to remove the table definition from the next update. But since the destination table is not dropped, they will be recovered when the next file arrives and the table annotation reappears in the source code. Source
Alternatively, we can make the source code scan the kernel directory and keep a stub of table definition.

Table Folder Access Control

Volume does not provide folder level access control. Therefore, table owners could accidentally upload the data to a different table under the schema and potentially break the ingestion.

Flow Resolution Issue Fails the DLT Update

Although disconnected flows do not interfere with each other, a DLT update run might fail if the flow cannot be resolved due to mixed file format, file name parsing failure, etc.

[Optional] Major alternatives

[Instructions]

Alternative: Per table pipeline

Table : Pipeline : Job : Volume = 1 : 1 : 1 : 1, this way the permission control model is more manageable at table level, as the Volume access is in sync with the table.

Alternative: DAB command as client interface

Instead of creating Python scripts that invoke DAB commands, make the CUJ use DAB commands directly. This way the multiplatform support comes free from Databricks CLI. If running custom code is needed, we build it as a job and run it directly in the workspace.

Alternative: Use DLT append flow

DLT append flow enables the flexibility to redirect the flow to a different streaming table, add backfill flow to the same table.

[Optional] Uncertainties and dependencies

[Instructions]

-

Project Tracking

How do you intend to track the usage of this project via our logfood telemetry?

There are various methods to track the usage of a Databricks Labs Project and attribute it to a number of DBUs. One or more of these can be employed. The preferred order is:

1. Library Import Tracking
2. Spark Config Parameter Tracking
3. REST API tracking

Method	**Description**	**Limitations**
Library Import	Whenever a library is imported from a package, that telemetry is tracked against the cluster it was imported on.	DLT clusters do not track this for some reason (investigating) DBSQL clusters cannot import libraries
Spark Config	Spark Configs can be tracked when used on clusters	Cannot be set via spark.conf.set(). Rather, it can only be detected if set upon cluster creation. Cannot contain anything that is considered PII, like table name. Value is unable to be redacted.
REST API	Whenever a labs project is installed via the CLI, it's name is included in the *http_user_agent* in the header. Other API calls to other Databricks endpoints can include the project name in the *http_user_agent*, along with other information, which can be detected.	Currently, the cluster_id is always redacted in the logs, which makes attributing labs projects to DBUs nearly impossible. The lowest granularity therefore that can be attributed is to workspace.

Proposed Project Tracking

Track	**Method**	**Description / Action Item**	**Action Taken**
☐	Library Import	Need to add library name to python_module_allowlist	☐
☑	Spark Config	Need to add spark config to spark config allowlist	☐
☐	REST API	Need to include project name in http_user_agent calls via the SDK method: *databricks.sdk.config.with_user_agent_extra()*	☐

Part 3: Launch checklist

API Clients

[Please remove this section if the project is a library imported in a notebook]

- ☐ Unique user-agent string is designed for telemetry
- ☐ User-agent prefix is added to Labs ETL for telemetry

Notebook Libraries

[Please remove this section if the project is an API client]

- ☐ Package prefix is unique and didn't appear in prod.workload_insights table
- ☐ Package prefix is added to Labs ETL for telemetry
- ☐ Check if the library does not exist in PyPI

Java/Scala projects

[Please remove this section if the project is written in Python]

- ☐ Maven is used as a build system (security scanning requirement)
- ☐ JaCoCo plugin is used
- ☐ Scalaguide is used (for Scala-projects)
- ☐ com.databricks.labs namespace is used

Python projects

[Please remove this section if the project is written in Java or Scala]

- ☐ There's no project with a similar name on PyPI
- ☐ PyTest is used. See dbx or arcuate for examples.
- ☐ Setup.py has correct metadata. See arcuate for example.
- ☐ Style checks are enforced via black
- ☐ PyPI release is published
- ☐ PyPI release process is automated in CD, no manual release process is allowed
- ☐ Test coverage metrics are measured and presented publicly
- ☐ Reporting via codecov.io is required for http://go/labs/dash telemetry

Administration

[These actions are performed by Labs administrators]

- ☐ Dependabot is enabled

- [ ] (Python) LABS_PYPI_TOKEN added for this new repository
- [ ] Lgtm.io is integrated
- [ ] CodeQL checks are integrated

Enablement

[Please add enablement materials and link them to this section]

- [ ] Enablement deck is ready
- [ ] Enablement video pitch is ready
- [ ] Schedule internal enablement session

Continuous integration

[Testing is fully automated]

- [ ] Github Actions are invoked for every pull request
- [ ] Code coverage is greater than 80%
- [ ] Codecov.io is integrated

Security & Legal

- [ ] Submit an SDR ticket for mandatory security and other reviews
- [ ] Fill in http://go/designdoc/legal and submit LPP-XXX, follow up with legal-product@databricks.com

Longevity

Projects **cannot** reach the v1.0 version since it's a labs project. To reach v1.0 means to get production-grade support and to be included in the main product. Project must always be http://go/labs/healthy:

- [ ] Code review is complete
- [ ] The release process is documented. See the arcuate.
- [ ] The release branch is locked to prevent force-pushes
- [ ] Initial release starts with v0.0.1 or v0.1.0
- [ ] End-user documentation is available. See datagen for example

Marketing

- [ ] Blog request is submitted - http://go/blogprocess
- [ ] Marketing request is submitted to add it to https://databricks.com/learn/labs