Unstructured Data Support in Polaris

Author: Yufei Gu
Create Date: Dec 5, 2024
Update Date: Jan 14, 2025

Background

The rapid growth of AI/ML and data-intensive applications has significantly increased the need for managing unstructured data files, like images, logs, and videos. A scalable and efficient solution for querying and managing unstructured data metadata is now essential.

Polaris plays a critical role in managing and querying large-scale data across different processing engines. Adding unstructured data support will enable Polaris to handle diverse data types efficiently, ensuring high performance and reliability.

A table-like entity like volumes can be used for organizing and managing unstructured data. Volumes provide a way to group related data files logically, similar to directories or containers. This structure offers key advantages:

1. **Better Organization**: Grouping files logically simplifies metadata management.
2. **Access Control**: Volumes help enforce isolation and enable policies for better security and governance. Polaris can ensure the correct **credential vending** based on the privilege.
3. **Scalability**: They provide a flexible framework for advanced use cases like federated queries and staging.

Goals

- Define a new table-like entity named volume in Polaris to support unstructured data.
- Users should be able to add/delete/update files in a volume without any additional API calls. For example, using aws s3 cli to add files.
- Volume access control and credential vending.
- Volume metadata query (file size, timestamp, original url, public url, md5, etc)
- Define APIs to create/drop/query volumes.

Non-Goals

- Transactional features, like versioning/snapshotting/branching, and transitional tracking. Transactional tracking is possible, but users won't be able to leverage the existing tool to add, update and remove files. So it is out of scope. We could have another proposal if the transactional use case is needed.

- Sharding/Partition: all files under a volume are organized freely without any structure.
- Volume/Directory table Sync Between Polaris and Other Catalogs: Iceberg tables in Polaris can be synchronized to other catalogs like Snowflake Horizon, AWS Glue. Do we need the same functionality for directory tables? Probably not. Syncing directory tables between different catalogs doesn't offer significant benefits. Each catalog can create a directory table that points to the same underlying storage location. Since there are no strict consistency guarantees on these tables, ensuring synchronization across catalogs may introduce unnecessary complexity.

Design

A volume belongs to a namespace under a Polaris catalog. The volume's storage type will be decided by the storage type of the owner catalog, specifically, the following 4 types will be supported:

1. AWS s3
2. Azure object store
3. GCS
4. Local file system, mainly for testing purposes.

Here is a command to create a new catalog in Polaris, once a catalog is created, its storage type, role and location are fixed.

```
None
polaris catalogs create \
  --storage-type s3 \
  --default-base-location s3://example-bucket/my_data \
  --role-arn ${ROLE_ARN} \
  my_catalog
```

Volume Properties
1. Location
2. File format, optional

Directory Table

The metadata of a volume will be backed by a read-only Iceberg table in Polaris, which is called directory table. The directory table is optional for a volume. It's enabled by default, but users can choose to use a volume without a directory table.
1. Clients/Engines can query it as an Iceberg table.
2. Table refresh is taken care of by Polaris from the user perspective. Please note, it still needs an engine to write the directory table. The engine itself doesn't belong to Polaris.

3. Table refresh happens asynchronously, and it will be triggered automatically by Polaris. The refresh interval is potentially configurable.
4. The directory table is automatically created once a volume is created. We could put it a special place under the catalog location, for example, `s3://example-bucket/my_catalog_location/_direcotry_tables/t1`

Here is the architecture diagram to show it works.

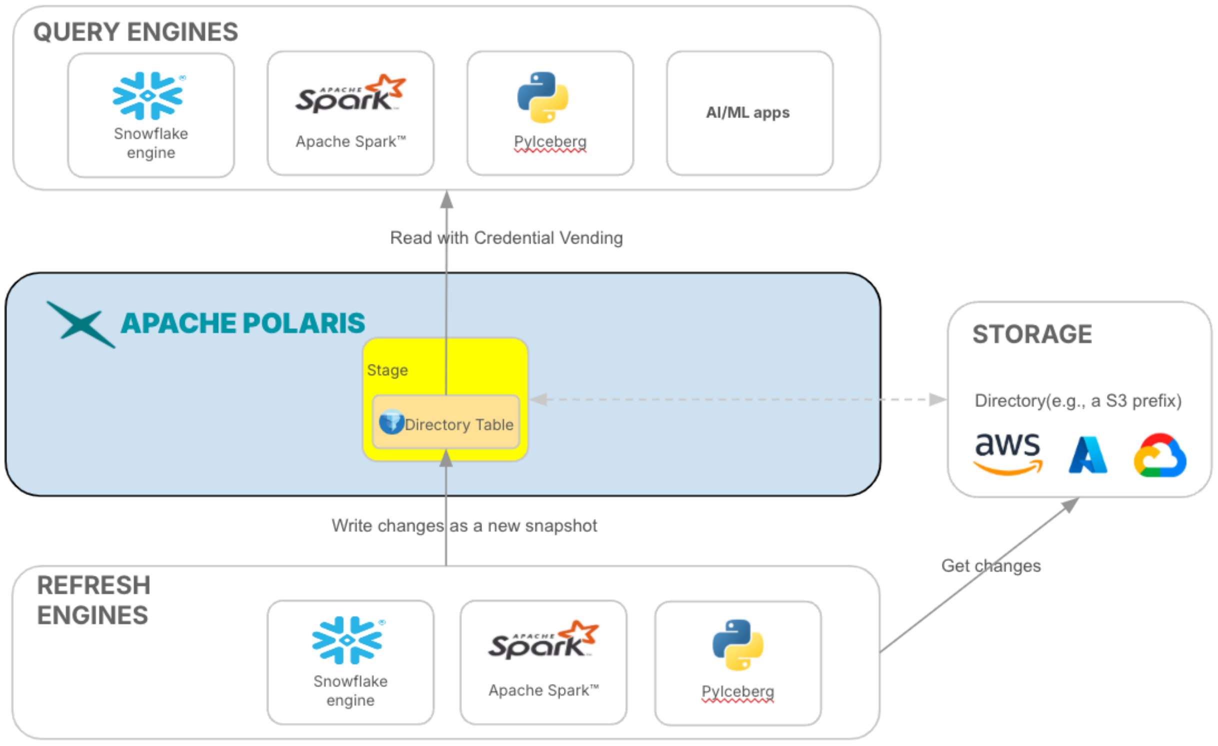

Table schema

```
CREATE TABLE dir_table ( relative_path STRING, size BIGINT,
last_modified TIMESTAMP, md5 STRING, file_url STRING )
```

Two options to refresh the directory tables by an engine

Option 1: Design a delegation service interface in Polaris to invoke a workload in a remote engine.
Option 2: The same model as TMS jobs like compaction and snapshot expiration. Polaris defines the refresh policy for a volume. TMS takes care of the fresh schedule.

REST Endpoints

Since the Iceberg REST specification does not currently cover directory-based file management, and it's unlikely that the Iceberg REST specification will cover this use case in the future, we need to introduce new APIs for this specific use case. These APIs will extend Polaris to handle directory-based tables natively, supporting the CRUD operations listed above.

The following are examples of the REST endpoints that will be introduced to enable this functionality:

- Create a new volume

  POST `/v1/{prefix}/namespaces/{namespace}/volume/`

- Drops a volume, purge options to indicate if the files should be deleted

  Delete `/v1/{prefix}/namespaces/{namespace}/volume/{volume}`

- Retrieves a list of files under the directory with certain filtering, like regex.

  GET/POST `/v1/{prefix}/namespaces/{namespace}/volume/{volume}`

Open questions

1. Do we allow users to list volumes under a namespace using the same API?
    a. The directory table of a volume is an Iceberg table. This makes it possible to unify both.
    b. The table list will be mixed with normal Iceberg tables and directory tables.