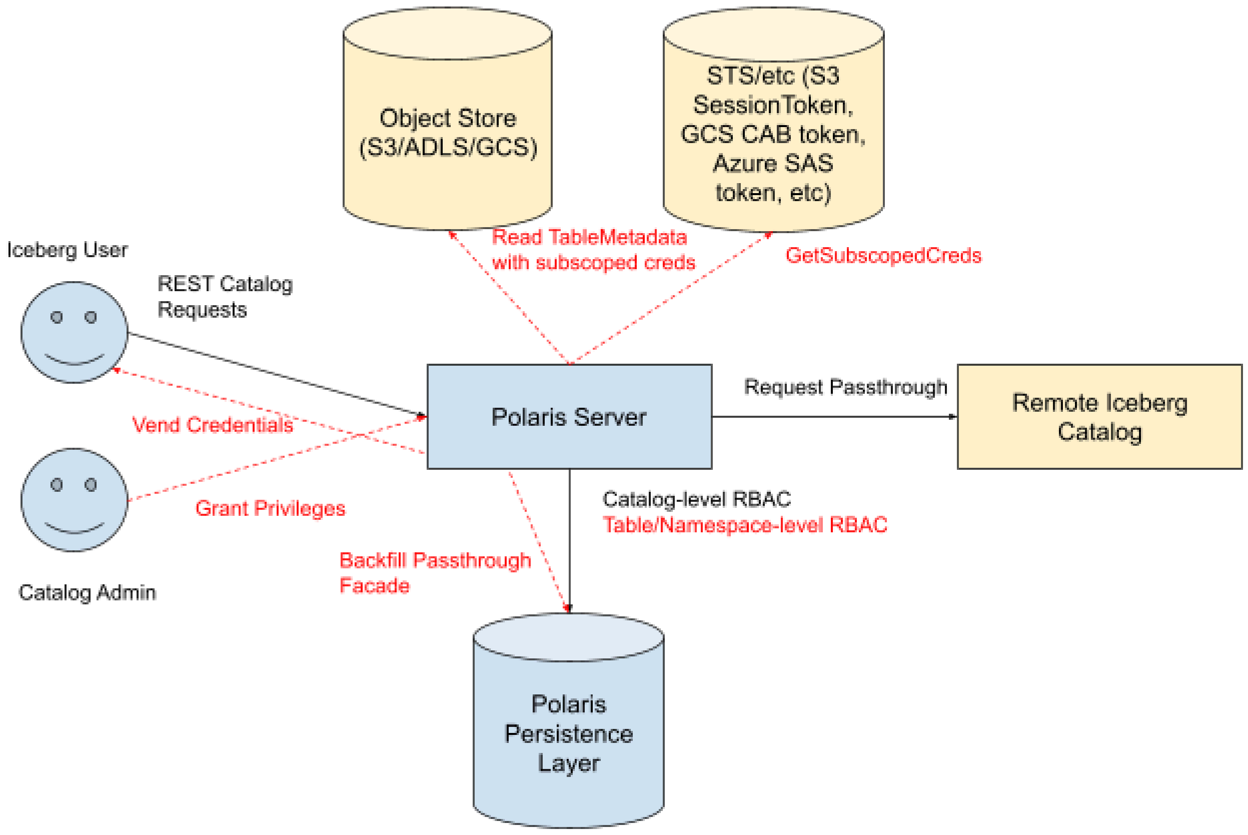
Apache Polaris Federation - Non-REST Remotes, Credential Vending and Table-Level RBAC Outline

Dennis Huo
Jul 16, 2025

Background

This document serves as a lightweight design outline for adding credential vending and table-level RBAC to the Polaris Catalog Federation feature originating in Apache Polaris Catalog Federation Proposal, along with a prerequisite refactor that is relevant for supporting federation non-Iceberg-REST remote catalogs in a secure way.

The key set of not-yet-built related features are highlighted in red:


Object Store (S3/ADLS/GCS)
STS/etc (S3 SessionToken, GCS CAB token, Azure SAS token, etc)
Read TableMetadata with subscoped creds
GetSubscopedCreds
Iceberg User
REST Catalog Requests
Vend Credentials
Polaris Server
Request Passthrough
Remote Iceberg Catalog
Grant Privileges
Catalog Admin
Catalog-level RBAC
Table/Namespace-level RBAC
Backfill Passthrough Facade
Polaris Persistence Layer

Three key pieces relate as follows, in order of their inter-dependencies:

- **Credential Vending**
  - Today, a Federated Catalog contains a StorageConfig but it is unused; minting a vended credential requires extracting machinery from the IcebergCatalog class to use in a "decorator" pattern on top of the remote catalog client
- **Non-REST Remote Catalogs**
  - Iceberg REST is unique in that the LoadTableResponse already contains the full body of the TableMetadata; other catalogs like GlueCatalog/HiveCatalog only provide the metadata *filename*, and "environment-based" configuration is used to instantiate FileIOs to read TableMetadata in an insecure way
  - Beyond "credential vending" to provide a credential back to a caller, we must also standardize the instantiation of the *internal* FileIO to match the behavior of the Polaris IcebergCatalog
- **Table/Namespace-Level RBAC**
  - Instantiation of the Catalog "decorator" that will be used for storage-credential management requires the creation of "synthetic" PolarisEntities backfilled into a PolarisResolutionManifest
  - A similar backfill of "synthetic" PolarisEntities is required during Namespace/Table-level grant assignment to provide the securables attached to GrantRecords

Design Details

Credential Vending

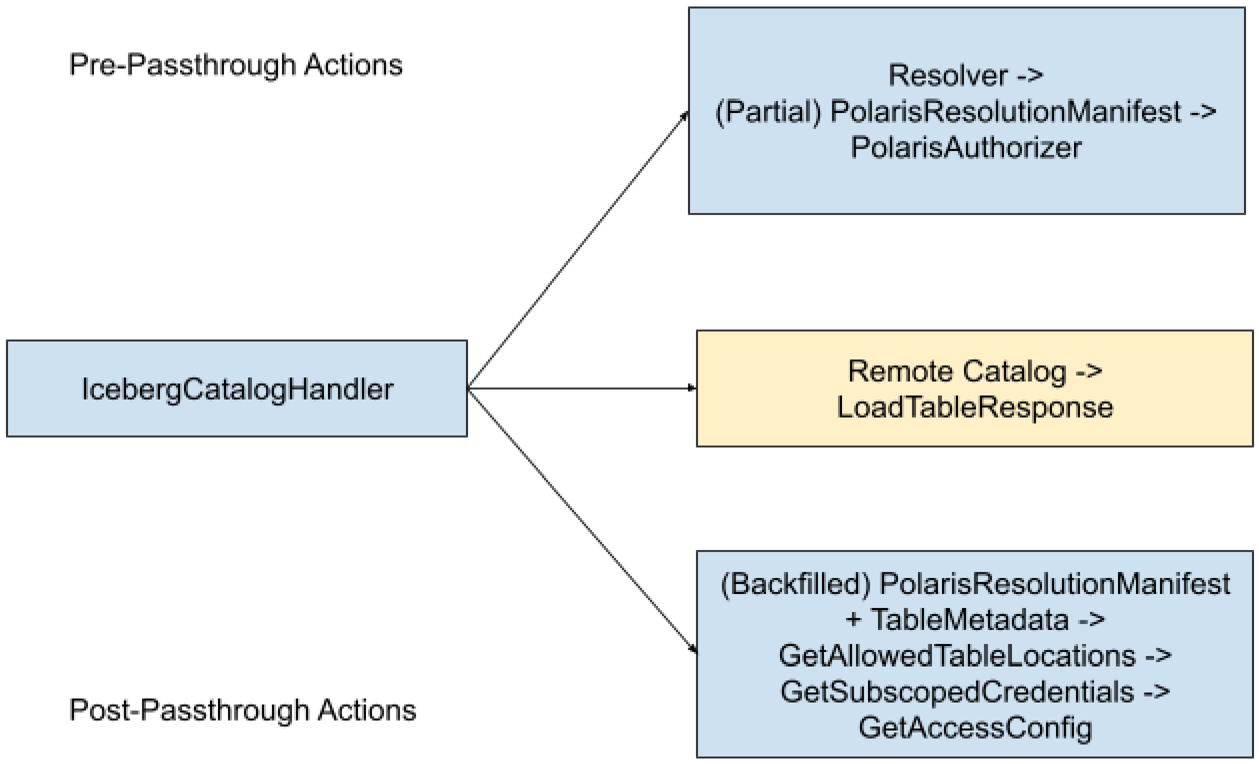
When a federated catalog client is used directly as the `baseCatalog` in the IcebergCatalogHandler, the credential-vending logic which looks for "`instanceof SupportsCredentialDelegation`" will fail, so no credentials will be vended:

```
    if (table instanceof BaseTable baseTable) {
      TableMetadata tableMetadata = baseTable.operations().current();
      return Optional.of(
          buildLoadTableResponseWithDelegationCredentials(
                  tableIdentifier, tableMetadata, actionsRequested, snapshots)
              .build());
    } else if (table instanceof BaseMetadataTable) {
      // metadata tables are loaded on the client side, return NoSuchTableException for now
      throw new NoSuchTableException("Table does not exist: %s",
tableIdentifier.toString());
    }

    throw new IllegalStateException("Cannot wrap catalog that does not produce BaseTable");
  }

  private LoadTableResponse.Builder buildLoadTableResponseWithDelegationCredentials(
      TableIdentifier tableIdentifier,
      TableMetadata tableMetadata,
      Set<PolarisStorageActions> actions,
      String snapshots) {
    LoadTableResponse.Builder responseBuilder =
        LoadTableResponse.builder().withTableMetadata(tableMetadata);
    if (baseCatalog instanceof SupportsCredentialDelegation credentialDelegation) {
      LOGGER
          .atDebug()
          .addKeyValue("tableIdentifier", tableIdentifier)
          .addKeyValue("tableLocation", tableMetadata.location())
          .log("Fetching client credentials for table");
      AccessConfig accessConfig =
          credentialDelegation.getAccessConfig(tableIdentifier, tableMetadata, actions);
```

In general, the responsibilities of the Catalog Federation layer can be divided into **Pre-Passthrough Actions** and **Post-Passthrough Actions**.


Pre-Passthrough Actions
Resolver -> (Partial) PolarisResolutionManifest -> PolarisAuthorizer
IcebergCatalogHandler
Remote Catalog -> LoadTableResponse
(Backfilled) PolarisResolutionManifest + TableMetadata -> GetAllowedTableLocations -> GetSubscopedCredentials -> GetAccessConfig
Post-Passthrough Actions

In the current MVP of Catalog Federation, since the remote-catalog client acts directly as the `baseCatalog` to which the request handling is delegated, there is no way to reuse Polaris shared logic for post-passthrough actions.

Detailed Implementation Items

To support credential vending in a way that aligns with the rest of Polaris, we must minimally:

- Add a decorating delegator implementation of Iceberg's Catalog (with SupportsNamespaces, SupportsViews, etc) which wraps a remote Catalog client of arbitrary type
- Make the decorating delegator class implement `SupportsCredentialDelegation`
- In `IcebergCatalogHandler::initializeCatalog` return the wrapped catalog instead of the raw remote catalog client
- Extract shared helpers like getLocationsAllowedToBeAccessed for the decorating delegator to use in its implementation of `getAccessConfig`

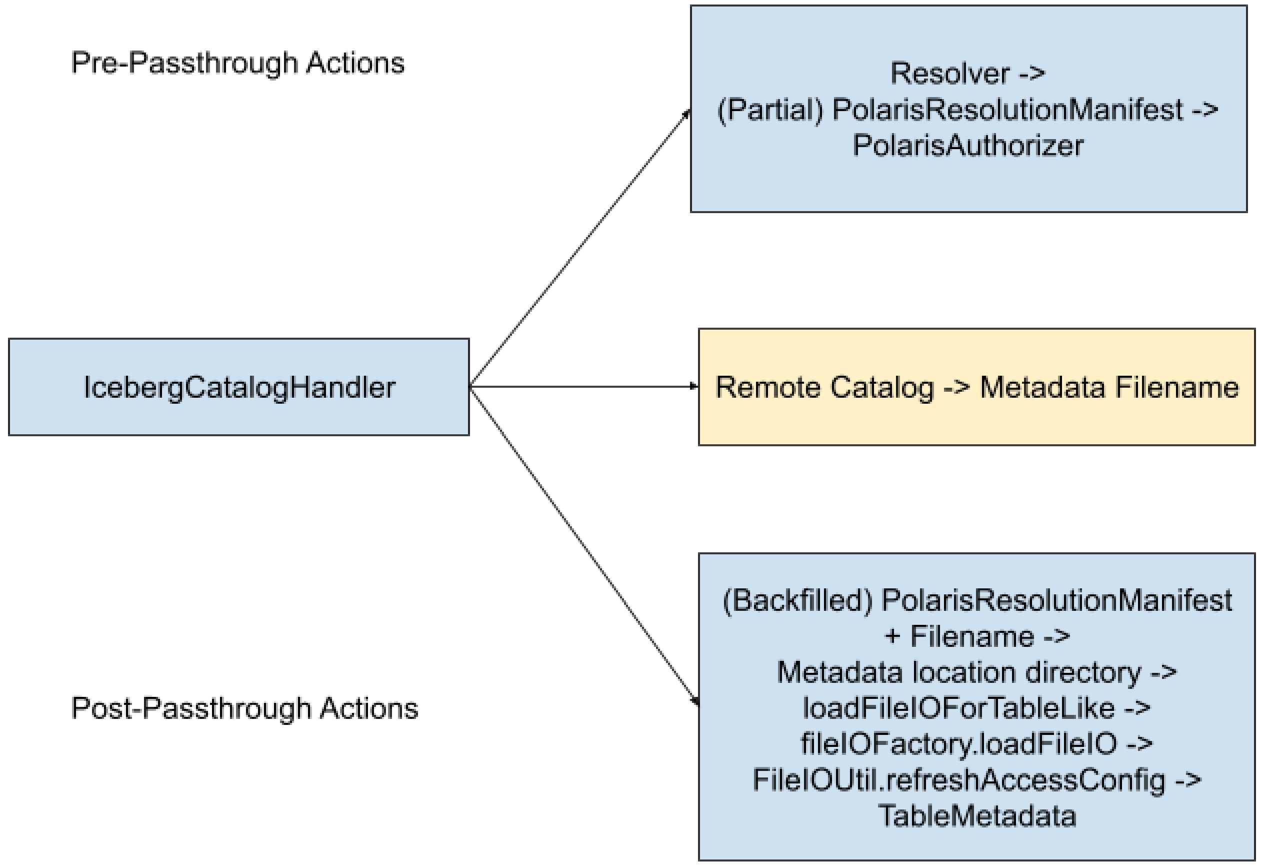
To better refactor shared logic and pave the way for other federation post-processing actions, ideally we'll also:

- Add a helper method which can take the low-level result of a remote loadTable response to "backfill" a PolarisResolutionManifest within the decorating delegator class

    - In contrast to the IcebergCatalog which takes a PolarisResolutionManifestCatalogView in its constructor, the decorating delegator should be able to produce a *synthetic* version of such a view as a Post-Passthrough Action
    - This synthetic PolarisResolutionManifestCatalogView will serve as a precursor for future JIT-Creation of Polaris Passthrough Facade entities
- Extract the full logic of getAccessConfig into a shared class (possibly an abstract base class) to be shared by the decorating delegator and the existing IcebergCatalog
    - Some refactoring may be required to allow helper methods like findStorageInfo, etc., to operate on a caller-specified PolarisResolutionManifestCatalogView

Non-REST Remote Catalogs

Extending the credential-vending work whereby we prepare a decorating delegator to perform credential-vending as a "post-passthrough action", the Non-REST Remote Catalog becomes another special case of post-passthrough processing in that reading the TableMetadata from the object store in itself becomes post-passthrough:


Pre-Passthrough Actions
Resolver -> (Partial) PolarisResolutionManifest -> PolarisAuthorizer
IcebergCatalogHandler
Remote Catalog -> Metadata Filename
Post-Passthrough Actions
(Backfilled) PolarisResolutionManifest + Filename -> Metadata location directory -> loadFileIOForTableLike -> fileIOFactory.loadFileIO -> FileIOUtil.refreshAccessConfig -> TableMetadata

In the absence of this flow, we currently by-default allow non-REST Catalog client libraries to exercise "custom" environment-based FileIO loading, which is insecure for many reasons:

- Circumvents the core Polaris FileIOFactory centralization which is often used for things like FileIO-layer throttling, metering, auditing, etc.
- May use ObjectStore SDK libraries to search the "default environment path" for credentials:
    - May look for environment variables like AWS_ACCESS_KEY_SECRET
    - May try to contact the "local cloud VM metadata server" for short-lived credentials
    - May search the local filesystem for credentials like ~/.awscredentials

Detailed Implementation Items

- Refactor IcebergCatalog to extract shared `loadFileIOForTableLike` logic
- Optional - Better centralize the interactions with `FileIOUtil.refreshAccessConfig` - currently both `IcebergCatalog::getAccessConfig` and `DefaultFileIOFactory::loadFileIO` directly call `FileIOUtil.refreshAccessConfig` but we may want to be able to change both callsites just by wrapping the remote catalog in our decorating delegator class

Table/Namespace-Level RBAC

Once we've solidified the pattern of producing a BackfilledPolarisResolutionManifest based on the initial PartialPolarisResolutionManifest + Contents of Remote Catalog response, we have two options:

- Backfill only a dummy entity in persistence without ever performing a passthrough to the remote catalog
    - Pros
        - Easy to implement
        - Doesn't require additional requests to the remote catalog just to generate grant records
        - Also fixes in-passing an existing bug where attempts at table-level/namespace-level grants on a federated catalog *may* just incorrectly apply the grant to the base CatalogEntity instead
    - Cons
        - No validation of existence or state of entities
        - No way to attach grants to a strict "version" of an entity; grants effectively become "by-name"
        - If authorization adds features like attribute-based access control, we won't have the actual entity attributes available to support that
- Require actually fetching a successful response from the remote catalog before producing the synthetic passthrough facade entity

    - Pros
        - Better aligned with future plans to have a strictly validated passthrough-facade entity
    - Cons
        - Requires more substantial work to be able to initialize a remote Catalog client within PolarisAdminService

It's feasible to begin with only backfilling a dummy entity in persistence and add remote-catalog mediated backfill later.

Detailed Implementation Items

- Modify PolarisAdminService to support "partial" PolarisResolutionManifest results in the "pre-passthrough action" phase prior to authorization
- TBD: Create a synthetic PolarisResolutionManifest with the full leaf path based naively on the fully-qualified entity name, without contacting the remote catalog
- Continue to apply the same existing grant logic
- Phase-2: Mimic the catalog-resolution logic from IcebergCatalogHandler to allow federated-catalog client creation in PolarisAdminService
- Phase-2: After authorizing the PartialPolarisResolutionManifest, issue the necessary remote calls to validate existence/state of securables on which we will be adding grants
- Phase2: Commit the Backfilled entities into the persistence layer

Longer-Term

Beyond the immedate features of credential vending and Table RBAC, the formalization of “Pre-Passthrough” and “Post-Passthrough” actions/transformations/decorations outlined here are the building blocks for the longer-term features relating to Catalog Migration mentioned in the “Future Directions and Catalog Migration” section of the original proposal.

The Catalog decorator pattern and “entity backfill” lead directly to features eventually serving the migration use case, including:

- Freshness-aware table loading
- TableMetadata caching
- Snapshot shredding, “snapshots=refs” filtering

The details of these features are out of scope of this document, but will be discussed in further Polaris Federation roadmap documents/designs.