

BearerAuthManager: a dependency-free AuthManager for the Iceberg REST Client

Author: Alex Dutra
Status: Proposed
Created: 2026-10-27

1. Overview

This proposal introduces `BearerAuthManager`, a new `AuthManager` implementation.

Its only job is to attach an `Authorization: Bearer <token>` header to REST requests, where the token comes either from 1) a file that the manager re-reads periodically, or 2) a static configuration property.

The main intended use case is around token files: `BearerAuthManager` is a natural fit for environments where the token is produced externally, for example by a Kubernetes service account projected volume, a cloud identity webhook, a sidecar (such as a Vault Agent, or the [SPIFFE helper](#)), or just a home-grown shell script.

A static token use case is also provided for simplicity; it should be reserved for trivial setups, testing, exploration, or for migration purposes, as it is inherently less secure.

This manager implements [RFC 6750](#) and is only loosely related to OAuth2: the token can be any opaque string. The manager never contacts an authorization server and never performs any OAuth2 grant of any kind; it also does not implement token refreshes. Obtaining and renewing the token becomes the responsibility of a producer process that lives outside the Iceberg REST client.

Because token production is decoupled from token consumption, `BearerAuthManager` does not require any third-party dependencies, and in particular does not require any OAuth2 client library.

With `BearerAuthManager`, authentication becomes a node-local process. This is the same philosophy behind other `AuthManager` implementations that read credentials from the node's environment, such as the `GoogleAuthManager`.

No token and no credentials are broadcast from control plane to data plane. Instead, what is broadcast is a *path* property pointing to a file on disk, that the node can read. Since it's just a string, it can be serialized like any other catalog property. And since it doesn't carry a secret, it is also safer.

Related Work

This proposal is an implementation of [Issue #13550](#) (July 2025). The issue explicitly lists Lakekeeper as a potential consumer of this feature, because it natively supports authentication via Kubernetes service account tokens.

[PR #17389](#): this existing PR also aims at implementing #13550, but by modifying the existing `OAuth2Manager`. This proposal instead introduces a new, greenfield manager, which is simpler and safer (no risk of regressions).

2. Goals and non-goals

Goals for the new manager:

- Ability to read a bearer token from a plain text file.
- Ability to re-read the token file periodically, with a configurable reload interval.
- Validation of the file contents, so that a malformed file cannot corrupt the `Authorization` header.
- Ability to read static tokens from configuration, for simpler setups.
- No third-party dependencies or client libraries required.

Non-goals:

- Ability to pass different token file paths to different nodes: all nodes will read from the same path (but may read different tokens from it).
- Token sources other than a file on disk or a static token from configuration. In particular, environment variables will not be supported, because they are less secure, cannot be rotated, and in a JVM hosting several catalogs, they would send one catalog's token to another.
- It's also a non-goal to expose token sources as a new public API, even if this could be done later if there is demand.
- Reading the token from a remote location, e.g. HTTP or S3, or over a unix socket. The file must be local to the node for security reasons; the manager will not implement any network protocol to fetch it.
- Per-user or per-session tokens: OAuth2 flows in general, and U2M (user-to-machine) ones in particular, are out of scope; besides, the so-called "contextual" auth sessions are not propagated to worker nodes. (This is a non-goal for the `BearerAuthManager` itself; it doesn't mean that the token producer isn't allowed to produce tokens from a U2M grant capturing the user's identity. That's entirely left for the engine / platform to figure out.)

- "Rich" files, e.g. JSON-formatted, and the ability to extract the token from it. The file is expected to contain the token verbatim, with no decoration.
- Reacting to a 401 error by re-reading the file and retrying: requires an `AuthSession` hook that does not exist today. See [PR #14241](#). See also §4.5 below.

3. Configuration

`BearerAuthManager` will be selected with `rest.auth.type=bearer`.

The proposed configuration properties are:

Property	Default	Description
<code>rest.auth.bearer.token-file</code>		Path to a file containing the bearer token. Required unless <code>rest.auth.bearer.token</code> is set; mutually exclusive with it.
<code>rest.auth.bearer.token</code>		A static bearer token. Mutually exclusive with <code>rest.auth.bearer.token-file</code> .
<code>rest.auth.bearer.token-file.reload-interval</code>	<code>PT1M</code>	How long a token read from the file is used before the file is read again. An ISO-8601 duration, e.g. <code>PT1M</code> for 1 minute.

Validation rules:

- Exactly one of `rest.auth.bearer.token-file` or `rest.auth.bearer.token` must be set. Setting both, or none, is an error.
- A token file value must be a path on the local filesystem. The path is allowed to be relative.
- The token file, if provided, must exist and be readable at the time of auth session initialization. If it does not, the session fails to initialize.
- The minimum reload interval is 1 second. Configuring a lower interval should result in an error.

Example: Spark on Kubernetes with a [projected service account token](#) and default reload interval:

None

```
spark.sql.catalog.polaris.uri           = https://example.com/api/catalog
spark.sql.catalog.polaris.rest.auth.type = bearer
spark.sql.catalog.polaris.rest.auth.bearer.token-file = /var/run/secrets/iceberg/token
```

Example of a corresponding Kubernetes pod spec snippet, with a projected service account token volume mounted at `/var/run/secrets/iceberg`:

None

```
volumeMounts:
  - name: iceberg-token
    mountPath: /var/run/secrets/iceberg

volumes:
  - name: iceberg-token
    projected:
      sources:
        - serviceAccountToken:
            path: token
            audience: polaris
            expirationSeconds: 3600
```

3.1 Support for the legacy `token` property

Today, the `OAuth2Manager` is inferred from the presence of either `token` or `credential` in the catalog properties.

Arguably, picking `OAuth2Manager` for `token` is overkill. `BearerAuthManager` is a more natural fit.

However, the `OAuth2Manager` can, at least in theory, refresh that token, while the `BearerAuthManager` cannot. In practice though, the refresh is only possible against catalogs that implement the deprecated internal token endpoint; in any other setup, the token is already non-refreshable.

Therefore, this proposal suggests that `BearerAuthManager` should be inferred instead of `OAuth2Manager` when the legacy `token` property is present. It will be mapped to the `rest.auth.bearer.token` key. The token will not be refreshable.

4. Design

4.1 Proposed structure

The implementation in Java will live in `iceberg-core`. The manager should be the only public class. Other required package-private components are:

- A **token source** abstraction that exposes the current token, and two implementations: a token file source and a static token source.
- A **bearer session** implementing `AuthSession`; it holds a token source and computes the `Authorization` header *at request time* from the source's current token.

4.2 Session kinds

`BearerAuthManager` will follow the guidelines below for the different session kinds it must create:

Session	Behavior
<code>initSession</code>	Default: delegates to <code>catalogSession</code> .
<code>catalogSession</code>	Validates the configuration (see §3), builds the token source, and returns a bearer auth session. Held by the Iceberg REST client for its lifetime.
<code>contextualSession</code>	Default: returns the parent. Per-session tokens are out of scope.
<code>tableSession</code> with a parent	Default: returns the parent. Vended tokens in <code>LoadTableResponse.config()</code> are not honored, in line with their recent deprecation.
"standalone" <code>tableSession</code>	For the S3 remote signer. Token sources will be cached, so that many signer instances in one JVM can access different token files efficiently.
<code>close</code>	Closes the shared sources. No-op for this manager.

See §4.6 below for more details on the vended-credential refresh providers and the S3 remote signer.

See Security considerations (§4.7) below for details about catalog server property overrides.

4.3 Token file source internals

The token file source will follow the strategy of Kubernetes' [client-go](#).

It will require no background thread. Its state will be stored in an immutable holder, capturing the current token and the *monotonic* time it was read at (Java's `System.nanoTime()`). The state holder is swapped atomically.

On each request for the token:

1. If the held token is younger than the interval, return it. Age is computed by subtracting monotonic time readings. This is the fast path.
2. Otherwise, attempt a reload (slow path). Only one thread reloads at a time; concurrent callers return the held token immediately rather than waiting.
3. Reload sequence: apply sanity checks; read the whole file; decode as UTF-8; then validate (see §4.4 below for details). On success, publish a new holder. On any failure, keep the current token, reset its read time so that the next attempt waits a full interval, and log a warning.

Symlinks must be followed. This is by design, and Kubernetes' projected service account volume is a good example: it writes a timestamped directory and a symlink to it, and [atomically swaps the symlink](#) when the token is rotated.

Keeping the last good token allows recovery from transient errors. For example, a producer that writes non-atomically may expose an empty or partial file for an instant; or a transient I/O error may cause the file to be unreadable. These should not turn into a failed authentication. This design choice has one important consequence though: *deleting the token file does not revoke the token*. The client keeps presenting the last token it read until that token expires server-side. Revocation is the authorization server's responsibility.

4.4 Token validation

Applied to both static tokens and file contents:

Check	Rule	Reason
Size cap	A file larger than 256 KiB is rejected before reading	256 KiB is large enough even for JWTs signed with the largest post-quantum schemes. The practical ceiling for token sizes is dictated by the default HTTP header limits applied by web servers.

Check	Rule	Reason
Trim	Leading and trailing whitespace (including a trailing newline) is removed	Tolerate <code>echo</code> and text editors.
Non-empty	Empty after trimming is rejected	An empty <code>Authorization: Bearer</code> header is never accepted.
<code>token68</code> grammar	The token must match the syntax for bearer tokens	This is the HTTP grammar of what follows <code>Bearer</code> in <code>Authorization</code> headers, defined in RFC 9110 §11.2 , identical to RFC 6750's <code>b64token</code> . Required for header safety: only a subset of visible ASCII is allowed; control characters, whitespace, DEL and non-ASCII are rejected.

4.5 Recovering from `401` errors (HTTP authentication challenges)

Not in scope. Today, the REST client turns a `401` into a `NotAuthorizedException` in its error handlers and never informs the `AuthSession` that produced the header, so there is no place for a source to learn that its token was rejected. client-go's behavior (invalidate the cache so that the *next* request re-reads) is thus not achievable.

An even better behavior is to re-read and retry the *same* failed request once; but again, that would need the `AuthSession` hook proposed in [PR #14241](#).

When that lands, the bearer session would implement the hook by forcing a reload on its source. Until then, the reload interval is the only guarantee that the token is valid at all times.

4.6 Usage on data plane nodes (worker nodes)

A hard requirement for all data plane nodes is that *the token file must exist at the configured path on every node that runs a `FileIO`*. On Kubernetes, this holds when all pods share the same volume definition and mounts.

The [S3 remote signer](#) loads an `AuthManager` from the `FileIO` properties and creates a "standalone" `tableSession`. It also injects OAuth2-specific keys (`oauth2-server-uri`, `scope`, `token-refresh-enabled`) and copies `token` and `credential` when present. With the `bearer` manager, those keys are ignored, except `token`. No change is needed in the signer for this proposal; its JVM-wide static manager instance is a pre-existing problem that `bearer` neither worsens nor fixes. See [Issue #16460](#).

`AuthSession` instances returned by `S3V4RestSignerClient` will reference cached token sources, in order to avoid excessive filesystem access. The cache key must include the token source type (file or static), and a changing part that fully identifies the source: either `{"file", <normalized_path>, <reload_interval>}` or `{"static", <token>}`. The reload interval isn't strictly required, but allows to disambiguate the rare case where two different `FileIO` instances point to the same path but with different reload intervals.

The [vended-credential refresh providers](#) (`VendedCredentialsProvider` for S3, `OAuth2RefreshCredentialsHandler` for GCS, `VendedAdlsCredentialProvider` for ADLS) load a manager from the `FileIO` properties and create a `catalogSession`. With the `bearer` manager, each catalog session gets its own token source, without caching: caching would barely save one file read per endpoint invocation, which seems unnecessary. (Note: in spite of its name, `OAuth2RefreshCredentialsHandler` works with any `AuthManager`.)

4.7 Security considerations

- On POSIX filesystems, when the file is readable by *others*, the file source will log a warning once at construction, the way SSH does for private keys. It should not fail: the platform decides permissions (e.g. Kubernetes writes `0600` when all containers share a `runAsUser`, but `0640` with `fsGroup`, and `0644` otherwise). See [KEP-1205 file permissions](#).
- Symlinks are followed by design. The filesystem layout and its security is the platform's responsibility.
- The manager never creates, modifies or deletes anything on the filesystem.
- Overridability of manager properties by a catalog server: the threat model treats the REST catalog server as a trusted control-plane component, see [Issue #16470](#). It could therefore, in theory, supply an override for properties like `token` or `token-file`. But letting the catalog server override these properties does not make sense (especially `token-file`), and presents a security risk. This risk has been extensively debated and the general consensus is that catalogs should not override authentication properties. Therefore, the `BearerAuthManager` will:
 - Accept overrides when it's impossible to distinguish the property provenance (config endpoint, S3 signer, vended-credential refreshers)
 - Ignore any override coming from load-table endpoint specifically (IOW, `tableSession` always returns the parent session).
- The manager must never accidentally log a token. The session and the token sources must print themselves without the token.
- Warning messages should not flood the logs:
 - Runtime issues should log on transitions, not on occurrences. For example, the file source should emit one warning when a reload first fails after a success, but not for every repeated failure.

- Configuration issues should log once per JVM.

5. Roadmap

The proposed roadmap targets the next Iceberg release, 1.13, for the Java implementation.

The core Java implementation fits in one PR for `iceberg-core`, carrying the manager, tests and docs.

Several follow-up tasks can be done after the core PR lands, in order to improve the ecosystem's support for bearer tokens:

1. Support for `BearerAuthManager` in Pylceberg, iceberg-rust and iceberg-go.
2. Support for `BearerAuthManager` in Trino and StarRocks.
3. Support for recovering from `401` errors and HTTP authentication challenges. This means reviving [PR #14241](#).