Apache Iceberg: Revolutionizing Data Lakes on AWS

I. Introduction to Apache Iceberg

Apache Iceberg is an open-source table format designed for managing enormous analytical datasets, often at petabyte scale, within data lakes.[1] Originating from Netflix and Apple to address the limitations of existing data lake table formats, it is now a vibrant Apache Software Foundation project.[1] The fundamental purpose of Iceberg is to bring the robust capabilities traditionally associated with data warehouses—such as ACID (Atomicity, Consistency, Isolation, Durability) transactions and standard SQL operations—to the flexible and cost-effective environment of data lakes.[1]

Traditional data lakes, often built on Apache Hive, suffered from several critical shortcomings. These included a lack of transactional guarantees, leading to potential data corruption or inconsistency, especially with concurrent operations. Their metadata management was often insufficient for performance at scale, and schema evolution was unpredictable, frequently resulting in data correctness errors. Furthermore, partitioning was a manual and error-prone process.[4] Apache Iceberg was engineered to solve these specific problems.

The key benefits that Iceberg offers stem directly from its innovative design. It provides an open standard, ensuring interoperability across a wide range of compute engines. Full ACID transactions guarantee data integrity. Schema evolution is flexible and safe, allowing table structures to change without rewriting underlying data. Time travel capabilities enable querying historical versions of data and rolling back to previous states. Performance is significantly enhanced through intelligent metadata and features like hidden partitioning, which abstracts the physical data layout from users.[1] This focus on correctness and reliability transforms the data lake from a potentially unreliable collection of files into a dependable, database-like foundation for analytics.[4] This shift is pivotal for organizations aiming to build modern data architectures that are both agile and robust.

II. Apache Iceberg Architecture

The architecture of Apache Iceberg is composed of three fundamental layers: the Catalog layer, the Metadata layer, and the Data layer.[5] Each layer plays a distinct role in managing table data and enabling Iceberg's advanced features.

A. The Catalog Layer

The Iceberg catalog is the entry point for all operations on Iceberg tables. Its primary responsibility is to track the current metadata pointer for each table, which is essential for enabling atomic operations like commits.[5] The catalog manages table names and organizes them into namespaces (analogous to schemas or databases in traditional systems).[6]

It is important to distinguish the Iceberg catalog from the Hive Metastore (HMS). While HMS can be used as one type of Iceberg catalog, Iceberg's design significantly reduces reliance on the traditional limitations of HMS, such as its single point of failure for table state or its performance bottlenecks with large numbers of partitions.[2]
Iceberg supports several types of catalogs:

- **File-based catalogs:** The Hadoop Catalog is an example, storing metadata pointers in files on a distributed file system like HDFS or cloud object stores.[6]
- **Service-based catalogs:** These leverage external services. Common examples include AWS Glue Data Catalog, a JDBC-accessible database, the Hive Metastore itself, and Project Nessie (which provides Git-like semantics for data).[6]
- **REST-based catalogs:** This is an increasingly popular approach, defined by an open API specification. Implementations like the open-source Project Polaris (incubating at Apache, originally from Snowflake) and Snowflake's internal catalog use this API, allowing various engines to interact with Iceberg tables through a standardized HTTP interface.[6]

The choice of catalog is a critical architectural decision. It directly impacts multi-engine interoperability, transactional guarantees, and overall system manageability. In the AWS ecosystem, the AWS Glue Data Catalog is a prevalent and often recommended choice due to its managed nature, serverless capabilities, and tight integration with other AWS analytics services like Athena, EMR, and Lake Formation.[6] This integration simplifies the setup and operational aspects of using Iceberg on AWS.

B. The Metadata Layer

The metadata layer is where Iceberg's intelligence truly resides, meticulously tracking the structure and state of tables. This layer consists of several key components: metadata files (commonly named metadata.json), manifest lists, and manifest files.[2]

- **Metadata Files (metadata.json):** Each Iceberg table has a current metadata file. This JSON file is the root of the table's metadata tree and contains vital information such as the table's current schema, the active partition specification, the history of snapshots (versions), and a pointer to the current snapshot ID.[3] A crucial aspect of Iceberg's design is immutability: every change to the table (e.g., a DML operation, schema change) results in the creation of a *new* metadata.json file, and the catalog atomically updates its pointer to this new file. Old metadata files are preserved, enabling time travel.
- **Snapshots:** A snapshot represents the complete state of a table at a specific point in time. Every DML or DDL operation that modifies the table creates a new snapshot.[2] Each snapshot points to a manifest list.
- **Manifest Lists:** These are Avro files that itemize all the manifest files constituting a particular snapshot. For each manifest file, the manifest list stores its path, its size, references to the snapshot it belongs to, and, importantly, partition bounds information. This allows query engines to quickly prune entire manifest files if their partition bounds do not overlap with a query's filters.[2]

- **Manifest Files:** Manifest files are also Avro files. Each manifest file lists a subset of the data files that make up a snapshot. For every data file, the manifest contains its path, storage format (e.g., Parquet, ORC, Avro), partition membership information, and detailed column-level statistics such as min/max values, null counts, and total counts for each column within that data file.[2]

This hierarchical structure—metadata file pointing to a manifest list, which points to multiple manifest files, which in turn point to the actual data files—is fundamental to Iceberg's efficiency. It allows query engines to perform aggressive metadata pruning. Instead of listing all files in a directory (a common bottleneck in traditional data lakes, especially on object stores), an engine can navigate this metadata tree, using the statistics and partition information at each level to eliminate irrelevant files and even entire manifest files from consideration, significantly reducing the I/O required for query planning and execution.[2]

C. The Data Layer

The data layer consists of the actual data files where the table's records are stored. Iceberg itself is format-agnostic regarding the data files, supporting popular columnar formats like Apache Parquet and Apache ORC, as well as row-based formats like Apache Avro.[1] These files are typically stored in distributed object storage systems such as Amazon S3, Azure Data Lake Storage, or Google Cloud Storage, or in HDFS.[1]

For handling row-level updates and deletes, Iceberg (specifically in its v2 specification and beyond) introduces the concept of **delete files**. Instead of immediately rewriting data files upon a delete or update, Iceberg can write separate delete files that record which rows are deleted. There are different types of delete files [5]:

- **Positional delete files:** Identify records to be deleted by their file path and position within that file.
- **Equality delete files:** Identify records to be deleted based on matching column values.
- **Deletion vectors (v3 spec):** A more optimized form of positional deletes, offering significant performance improvements.[5]

These delete files are applied at query time (merge-on-read) or can be periodically merged with base data files during compaction maintenance operations (copy-on-write).

The clear separation of the data layer (the actual data files) from the metadata layer is a cornerstone of Iceberg's architecture. This decoupling is what enables powerful features like schema evolution and partition evolution without the need to rewrite the underlying data files, a significant improvement over older table formats where such changes were often prohibitively expensive.[1]

III. Core Features of Apache Iceberg

Apache Iceberg offers a rich set of features designed to bring reliability, performance, and flexibility to data lake operations. These features collectively address many of the historical pain points associated with managing large-scale analytical datasets.

A. ACID Transactions

One of Iceberg's most significant contributions is bringing ACID (Atomicity, Consistency, Isolation, Durability) transactional guarantees to data lakes.[1]

- **Atomicity:** Operations are all-or-nothing. A commit to an Iceberg table either fully succeeds, making all its changes visible, or it fails, leaving the table in its previous state. This is achieved by atomically swapping the pointer in the catalog to a new metadata file upon a successful commit.[5]
- **Consistency:** Transactions bring the data from one valid state to another. Iceberg ensures this by maintaining a consistent view of the data through snapshots.
- **Isolation:** Concurrent operations do not interfere with each other. Iceberg provides snapshot isolation, meaning readers see a consistent version of the table as of the start of their query, unaffected by concurrent writes. Writers operate on isolated snapshots and commit their changes atomically.[1]
- **Durability:** Once a transaction is committed, its changes are permanent and will not be lost, as they are persisted in metadata files and the underlying data files on durable storage.[5]

Iceberg employs an optimistic concurrency control model. Multiple writers can operate simultaneously, and conflicts (e.g., if two writers try to update the same data) are typically detected and resolved at commit time.[1] The ability to perform ACID transactions reliably is foundational for modern data architectures where data is constantly being ingested, updated, and queried by multiple processes and users simultaneously. It ensures data integrity and trustworthiness, which are paramount for analytical workloads.[3]

B. Schema Evolution

Iceberg provides robust and flexible schema evolution capabilities, allowing users to modify a table's schema without rewriting existing data files.[1] Supported operations include:

- Adding new columns
- Dropping existing columns
- Renaming columns
- Reordering columns
- Updating column types (type promotion, e.g., INT to BIGINT, FLOAT to DOUBLE)

This is possible because Iceberg tracks columns by unique IDs assigned when a column is added to the schema, rather than by column names.[2] When the schema evolves, a new version of the schema is recorded in the table's metadata. Query engines use the table's schema history to correctly interpret data files written with older schema versions. Iceberg also supports schema evolution for nested structures within complex types.[2] This safe and low-impact schema evolution is a game-changer for agile data development. As business requirements change and new data sources emerge, data structures can adapt without the need for costly and time-consuming data migrations or the risk of breaking existing queries and applications.[1]

C. Partition Evolution

Similar to schema evolution, Iceberg allows the partitioning scheme of a table to evolve over time without requiring the rewriting of existing data.[2] For example, a table initially partitioned by day might later be changed to be partitioned by hour, or a new partition field might be added.
When a partition specification changes:

- Existing data files remain physically organized according to their original partition scheme.
- New data written to the table will use the new partition specification.
- Iceberg's metadata tracks the history of partition specifications, allowing query engines to correctly plan queries across data written with different partitioning schemes.[5]

This capability addresses a significant limitation of traditional Hive tables, where changing the partitioning scheme often necessitated a full rewrite of the table, a daunting task for large datasets.[2] With Iceberg, partition evolution becomes a metadata-only operation for existing data, providing crucial flexibility for long-term data management and query optimization.

D. Hidden Partitioning

Iceberg introduces the concept of hidden partitioning, where partition values are derived from source column values through transformation functions, rather than being stored as separate physical columns in the data files.[1] Examples of transform functions include:

- day(ts): Extracts the day from a timestamp column ts.
- month(ts), year(ts), hour(ts): Similar time-based transforms.
- bucket(N, col): Partitions by the hash of col into N buckets.
- truncate(L, col): Truncates a string or numeric column col to length L.

Because these partition values are generated by Iceberg based on the table's partition specification and the data itself, users can query the table using natural predicates on the source columns (e.g., WHERE event_timestamp >= '2023-01-01' AND event_timestamp < '2023-01-02') without needing to know the exact physical partition column names or values (e.g., dt='2023-01-01').[2] The query engine, using Iceberg's metadata, automatically maps these predicates to the underlying partitions.
Hidden partitioning simplifies query writing for end-users, as they interact with the logical table structure. It also enables more efficient partition pruning by query engines and is a key enabler for partition evolution, as the physical data files do not need to store redundant partition columns.[1]

E. Time Travel & Version Rollback

Iceberg's snapshot-based architecture inherently supports time travel and version rollback.[1] Every operation that modifies a table (e.g., INSERT, UPDATE, DELETE, MERGE, schema change) creates a new snapshot, which represents the state of the table at that specific commit time.

- **Time Travel:** Users can query historical versions of a table by referencing a specific

snapshot ID or a timestamp. This allows for:
    - Auditing data changes over time.
    - Debugging data issues by examining previous states.
    - Reproducing analytical results or machine learning model training based on data as it existed at a particular point.
- **Version Rollback:** If an erroneous operation corrupts the table or produces incorrect results, administrators can easily roll the table back to a previous, valid snapshot.

These capabilities are invaluable for data governance, operational stability, and debugging. They provide a safety net for data operations and enhance the reproducibility of data-driven analyses.[1]

F. Performance Optimizations

Iceberg is designed for high performance on large analytical workloads, achieved through several mechanisms centered around its intelligent metadata management:

- **Metadata Pruning:** During query planning, engines use the information in manifest lists (partition bounds) and manifest files (column-level statistics like min/max values, null counts) to prune away files and even entire partitions that do not contain relevant data for the query.[1] This significantly reduces the amount of data that needs to be scanned.
- **No File Listing:** Unlike Hive tables that often require listing files in directories (an expensive operation on object stores), Iceberg queries directly access the data files listed in the relevant manifests after the planning phase.[2]
- **Data Skipping:** The column-level statistics stored in manifest files allow engines to skip reading data files (or parts of files in columnar formats like Parquet/ORC) if the query predicates fall outside the min/max range of values for the queried columns in those files.[1]
- **Sorted Tables:** Iceberg supports defining a sort order for tables. Writing data pre-sorted according to this order can further enhance data skipping and improve the performance of queries with filters on the sorted columns.[2]

Collectively, these optimizations ensure that query engines can be highly selective about the data they process, leading to faster query execution times and reduced computational costs, especially for petabyte-scale tables.[1]

IV. Apache Iceberg in the AWS Ecosystem

Apache Iceberg has gained significant traction within the Amazon Web Services (AWS) ecosystem, with AWS strategically positioning it as a cornerstone for building modern data lakes and lakehouse architectures.[9] This commitment is evident through native support and deep integrations across a suite of AWS analytics and data services. The core services enabling Iceberg solutions on AWS include Amazon S3 for durable storage, AWS Glue Data Catalog for metadata management, Amazon EMR, Amazon Athena, and Amazon Redshift for compute and query processing, and AWS Lake Formation for centralized data governance.[9] This comprehensive support allows organizations to construct robust, end-to-end data pipelines leveraging Iceberg's capabilities within a familiar and powerful cloud environment.[9]

A. Overview of Iceberg on AWS

AWS actively promotes Apache Iceberg as a critical component for constructing "modern data lakes," which essentially embody the lakehouse paradigm by combining the scalability and flexibility of data lakes with the transactional and performance features of data warehouses.[9] The breadth of native Iceberg support across services like Amazon EMR, Amazon Athena, AWS Glue, Amazon Redshift, Amazon S3 Tables, and AWS Lake Formation underscores a strategic direction towards Iceberg as a foundational open table format on the AWS platform.[9] This widespread adoption within AWS services encourages users to build their data architectures on Iceberg, offering confidence in long-term viability and seamless interoperability. Such an ecosystem allows different teams and tools to collaborate on the same datasets; for instance, data might be ingested by Apache Flink or Spark on Amazon EMR, cataloged by AWS Glue, queried interactively by Amazon Athena and Amazon Redshift, and secured by AWS Lake Formation, all interacting with the same Iceberg tables. This not only simplifies pipeline development but also reduces vendor lock-in at the table format level, fostering a more open and flexible data strategy on AWS.

B. Storage: Amazon S3

Amazon Simple Storage Service (S3) serves as the primary storage backend for both the data files (e.g., Parquet, ORC, Avro) and the metadata files (manifests, manifest lists, table metadata JSON) of Apache Iceberg tables on AWS.[5] S3's inherent scalability, durability, and cost-effectiveness make it an ideal foundation for data lakes.
Iceberg's S3FileIO implementation is specifically designed for optimized interaction with Amazon S3. It supports features such as:

- **Progressive Multipart Uploads:** For efficient writing of large files.
- **Server-Side Encryption (SSE):** Including SSE-S3, SSE-KMS, and SSE-C for data protection.
- **Access Control Lists (ACLs):** For granular permission management at the object level.
- **Object Store File Layout:** An optional layout (write.object-storage.enabled=true) that helps distribute files across different S3 prefixes by appending a hash to file paths. This can mitigate S3 request throttling issues that might occur with traditional Hive-style partitioning where many files share the same prefix.[19]

C. Cataloging: AWS Glue Data Catalog

The AWS Glue Data Catalog is the most commonly used and officially recommended catalog for Apache Iceberg deployments on AWS.[6] It is a fully managed, Hive-compatible metastore that integrates seamlessly with a wide array of AWS services, including EMR, Athena, Redshift, and Lake Formation.[9]
Key configuration properties when using Glue Data Catalog with Iceberg include [19]:

- catalog-impl or type: Set to org.apache.iceberg.aws.glue.GlueCatalog or glue respectively.

- warehouse: Specifies the root S3 path for Iceberg tables.
- io-impl: Defines the FileIO implementation, defaulting to org.apache.iceberg.aws.s3.S3FileIO for S3.
- glue.id: The AWS account ID for the Glue Catalog, enabling cross-account catalog access.
- **Optimistic Locking:** Iceberg leverages Glue's optimistic locking mechanism for concurrent updates. This requires AWS SDK version 2.17.131 or newer. For older SDK versions, or for enhanced robustness, a DynamoDB-based lock manager (lock-impl = org.apache.iceberg.aws.dynamodb.DynamoDbLockManager and lock.table = <your-dynamodb-lock-table-name>) can be configured to ensure atomic transactions across multiple writers or engines.[19]
- glue.skip-archive: A boolean property (defaults to true, meaning skip archival) that controls whether older versions of Glue table metadata are archived by AWS Glue. For streaming use cases with frequent commits, keeping this as true is generally advisable to avoid accumulating a large number of Glue table versions.[19]

The choice of the AWS Glue Data Catalog simplifies metadata management significantly, as it's a serverless component that scales automatically and is deeply embedded within the AWS analytics stack.

D. Compute & Query Engines on AWS

AWS offers a variety of compute and query engines that are compatible with Apache Iceberg, catering to different processing needs, from batch ETL and streaming to interactive SQL and serverless analytics.

- 1. Amazon EMR (Elastic MapReduce)
  Amazon EMR provides managed clusters for big data processing frameworks like Apache Spark, Apache Flink, Apache Hive, and Trino, all of which can work with Iceberg tables.
    - **Apache Spark:** This is the primary and most feature-rich engine for Iceberg on EMR. It supports full DDL and DML operations, batch processing, and structured streaming with Iceberg tables.[9] Configuration involves including the iceberg-spark-runtime package and setting up the AWS Glue Catalog as the Spark catalog.
      Scala
      ```
      // Example Spark SQL configuration for Glue Catalog
      spark.sql("SET
      spark.sql.catalog.my_glue_catalog=org.apache.iceberg.spark.SparkCatalog")
      spark.sql("SET
      spark.sql.catalog.my_glue_catalog.catalog-impl=org.apache.iceberg.aws.glue.Glu
      eCatalog")
      spark.sql("SET
      spark.sql.catalog.my_glue_catalog.warehouse=s3://your-bucket/warehouse/")
      spark.sql("SET
      ```

spark.sql.catalog.my_glue_catalog.io-impl=org.apache.iceberg.aws.s3.S3FileIO")
19

   - **Apache Flink:** Well-suited for real-time analytics and streaming data ingestion into Iceberg tables, as well as batch processing.[9] It uses the flink-iceberg connector and can be configured with the AWS Glue Catalog. An example Flink SQL CREATE CATALOG statement:
     SQL
     ```
     CREATE CATALOG glue_catalog_for_iceberg WITH (
       'type'='iceberg',
       'warehouse'='s3://your-bucket/flink-iceberg-warehouse/',
       'catalog-impl'='org.apache.iceberg.aws.glue.GlueCatalog',
       'io-impl'='org.apache.iceberg.aws.s3.S3FileIO',
       'lock-impl'='org.apache.iceberg.aws.glue.DynamoLockManager', --
     Recommended for concurrent writes
       'lock.table'='FlinkIcebergLockTable'
     );
     ```
     33
   - **Apache Hive:** Can be used to query Iceberg tables on EMR, leveraging the iceberg-hive-runtime and typically configured with the AWS Glue Data Catalog.[19]
   - **Trino (formerly PrestoSQL):** Provides fast, interactive SQL querying capabilities for Iceberg tables, often used for ad-hoc analytics and federated queries across multiple data sources. It integrates via an Iceberg connector plugin, commonly using the AWS Glue Data Catalog.[9] To configure Trino with Glue Catalog for Iceberg on EMR, an iceberg.properties file in the Trino catalog directory would typically specify:
     Properties
     ```
     connector.name=iceberg
     iceberg.catalog.type=glue
     # hive.metastore.uri=thrift://<emr-master-ip>:9083 -- Only if using Hive Metastore
     directly
     ```
     40
- 2. Amazon Athena
  Amazon Athena offers built-in, serverless query capabilities for Iceberg tables stored in S3 and cataloged in AWS Glue. It supports a wide range of operations including DDL (Data Definition Language) for creating and altering tables, DML (Data Manipulation Language) such as INSERT, UPDATE, DELETE, and MERGE, Create Table As Select (CTAS), time travel queries, and schema evolution.9 UPDATE, MERGE INTO, and DELETE FROM operations in Athena use a merge-on-read approach with positional deletes.10
- 3. Amazon Redshift Spectrum
  Amazon Redshift Spectrum extends Redshift's querying capabilities to data stored externally in Amazon S3, including Iceberg-formatted tables. This allows users to query Iceberg tables directly from their Redshift data warehouse without needing to load the

data into Redshift.9 Setup involves creating an external schema in Redshift that links to the AWS Glue Data Catalog where the Iceberg tables are registered.15 Currently, Redshift Spectrum provides read-only access to Iceberg tables and leverages optimizations like metadata pruning, columnar storage, and predicate pushdown for efficient querying.15

- 4. Serverless Ingestion/Processing with PyIceberg & AWS Lambda
  For lightweight, event-driven data processing or scheduled table updates, PyIceberg can be used in conjunction with AWS Lambda functions and an AWS Glue Iceberg REST endpoint.14 In this architecture, a Lambda function utilizes the PyIceberg library to interact with Iceberg tables (e.g., create tables, insert data) via the Glue REST catalog.46 A key consideration for this pattern is managing potential concurrency conflicts if multiple Lambda invocations attempt to write to the same Iceberg table simultaneously.46

The following table summarizes the Iceberg feature support across these key AWS analytics services:

Table: Apache Iceberg Feature Support Across AWS Analytics Services

AWS Service	Catalog Used	Read	Write (Append)	Update/Delete/Merge	Schema Evolution	Partition Evolution	Time Travel	Maintenance Ops (e.g., Compaction)
Amazon EMR (Spark)	AWS Glue, Hive, etc.	Yes	Yes	Yes	Yes	Yes	Yes	Yes (via Spark procedures)
Amazon EMR (Flink)	AWS Glue, Hive, etc.	Yes	Yes	Yes (Upsert)	Yes	Yes	Yes	Limited (depends on Flink support)
Amazon EMR (Hive)	AWS Glue, Hive	Yes	Yes	Limited	Limited	No	Limited	No (via Hive directly)
Amazon EMR (Trino)	AWS Glue, Hive, etc.	Yes	Yes (INSERT)	Limited	Yes	Yes	Yes	No (via Trino directly)
Amazon Athena	AWS Glue	Yes	Yes	Yes	Yes	Limited (via DDL)	Yes	Yes (OPTIMIZE, VACUUM)
Amazon Redshift Spectrum	AWS Glue	Yes	No	No	N/A (reads schema)	N/A (reads partitions	N/A	No

						)		
AWS Glue ETL (Spark)	AWS Glue	Yes	Yes	Yes	Yes	Yes	Yes	Yes (via Spark procedures)
PyIceberg /Lambda (Glue REST)	AWS Glue (REST)	Yes	Yes	Limited	Yes	Limited	Yes	Limited (via PyIceberg API)

Sources: [9]
This matrix helps in selecting the appropriate AWS service based on the specific Iceberg operations required. For instance, complex data transformations and comprehensive maintenance are best handled by Spark on EMR or AWS Glue ETL. Serverless querying and DML are well-supported by Athena, while Redshift Spectrum is ideal for federated queries from an existing data warehouse environment.

V. Hands-On Guide: Working with Iceberg Tables on AWS

This section provides practical examples of creating, writing to, querying, and evolving Apache Iceberg tables using key AWS services, primarily Amazon Athena and Amazon EMR with Spark. These examples assume that the AWS Glue Data Catalog is being used.

A. Creating Tables

- Using Amazon Athena:
  Athena allows direct creation of Iceberg tables using SQL DDL. The TBLPROPERTIES ('table_type'='ICEBERG') clause is essential.10
  - **Example (Partitioned Table):** This creates a table partitioned by the day of event_ts (hidden partitioning) and explicitly by event_type.
    SQL
    ```
    CREATE TABLE my_athena_iceberg_table (
      event_ts timestamp,
      user_id bigint,
      event_type string,
      details string
    )
    PARTITIONED BY (day(event_ts), event_type)
    LOCATION 's3://your-bucket/warehouse/my_athena_iceberg_table/'
    TBLPROPERTIES ('table_type'='ICEBERG', 'format'='parquet');
    ```
    10
  - **CTAS (Create Table As Select) Example:** This creates an Iceberg table and

populates it with data from source_table, partitioned by the day of event_ts.
SQL
```
CREATE TABLE my_iceberg_ctas_table WITH (
   table_type = 'ICEBERG',
   location = 's3://your-bucket/warehouse/my_iceberg_ctas_table/',
   is_external = false, -- Creates a managed table in Glue
   partitioning = ARRAY['day(event_ts)']
) AS
SELECT event_ts, user_id, event_type, details FROM source_table WHERE
year(event_ts) = 2023;
```
10

- Using Amazon EMR (Spark SQL):
  Spark SQL on EMR, when configured with the AWS Glue Catalog, can also create Iceberg tables.28
  - **Example (Partitioned Table):** This example assumes a Spark session configured to use my_glue_catalog.
    SQL
```
-- Assuming Spark session is configured with my_glue_catalog
CREATE TABLE my_glue_catalog.my_db.my_spark_iceberg_table (
   event_ts timestamp,
   user_id bigint,
   event_type string,
   details string
)
USING iceberg
PARTITIONED BY (days(event_ts), event_type) -- Iceberg transform syntax
LOCATION 's3://your-bucket/warehouse/my_spark_iceberg_table/'
TBLPROPERTIES ('format-version'='2'); -- Recommended for full v2 features
```
28

- Using AWS Glue ETL (Spark DataFrame API):
  AWS Glue ETL jobs can create and write to Iceberg tables using the Spark DataFrameWriterV2 API.28
  - **Example (Create and Write):** This Python snippet assumes input_df is an existing Spark DataFrame and the Glue job is configured for Iceberg with glue_catalog.
    Python
```
from pyspark.sql import functions as F

# Assuming 'input_df' is a Spark DataFrame
# and Glue job is configured for Iceberg and Glue Catalog (e.g., 'glue_catalog')
input_df.writeTo("glue_catalog.my_db.my_glue_etl_table") \
  .tableProperty("format-version", "2") \
  .partitionedBy("event_type", F.days(F.col("event_ts"))) \
```

.createOrReplace() # Use.create() if table should not be replaced if it exists
9

B. Writing & Querying Data

- **Writing Data (DML):**
  - **Amazon Athena:** Supports INSERT INTO, UPDATE, MERGE INTO, and DELETE FROM statements. UPDATE, MERGE, and DELETE use a merge-on-read approach with positional deletes.[10]
    - **Example MERGE INTO in Athena:**
      SQL
      ```
      MERGE INTO my_athena_iceberg_table t
      USING source_updates s ON t.user_id = s.user_id AND day(t.event_ts) =
      day(s.event_ts)
      WHEN MATCHED AND s.is_deleted = true THEN DELETE
      WHEN MATCHED THEN UPDATE SET t.details = s.new_details
      WHEN NOT MATCHED THEN INSERT (event_ts, user_id, event_type, details)
                    VALUES (s.event_ts, s.user_id, s.event_type, s.details);
      ```
      10
  - **Amazon EMR (Spark SQL):** Also supports INSERT INTO/OVERWRITE, MERGE INTO, UPDATE, and DELETE FROM. MERGE INTO, UPDATE, and row-level DELETE often require Iceberg Spark extensions to be enabled.[26]
    - **Example MERGE INTO in Spark SQL:**
      SQL
      ```
      -- Assuming source_updates_view provides new/updated records
      MERGE INTO my_glue_catalog.my_db.my_spark_iceberg_table t
      USING source_updates_view s ON t.user_id = s.user_id AND date(t.event_ts)
      = date(s.event_ts)
      WHEN MATCHED AND s.operation = 'DELETE' THEN DELETE
      WHEN MATCHED AND s.operation = 'UPDATE' THEN UPDATE SET t.details =
      s.new_details
      WHEN NOT MATCHED AND s.operation = 'INSERT' THEN INSERT *;
      ```
      26
- Querying Data:
  Both Athena and Spark SQL use standard SELECT statements. Effective querying relies on Iceberg's metadata for predicate pushdown and partition pruning.10
- Time Travel Queries:
  Iceberg allows querying the state of a table at a specific point in time or a specific snapshot ID.
  - **Amazon Athena:**
    SQL
    ```
    SELECT * FROM my_athena_iceberg_table FOR TIMESTAMP AS OF TIMESTAMP
    '2023-01-15 10:00:00 UTC';
    ```

```
SELECT * FROM my_athena_iceberg_table FOR VERSION AS OF
1234567890123456789;
```
10

- **Amazon EMR (Spark SQL):**
  SQL
  ```
  SELECT * FROM my_glue_catalog.my_db.my_spark_iceberg_table VERSION AS OF
  1234567890123456789;
  SELECT * FROM my_glue_catalog.my_db.my_spark_iceberg_table TIMESTAMP AS
  OF '2023-01-15 10:00:00';
  ```
  28

C. Schema & Partition Evolution Examples

- **Schema Evolution:**
  - **Amazon Athena:** Supports adding, changing type, and renaming columns via ALTER TABLE statements.[42]
    - **Example (Athena - Add Column):**
      SQL
      ```
      ALTER TABLE my_athena_iceberg_table ADD COLUMNS (new_metric
      double);
      ```
  - **Amazon EMR (Spark SQL):** Also supports schema changes via ALTER TABLE.[48]
    - **Example (Spark SQL - Change Column Type):**
      SQL
      ```
      ALTER TABLE my_glue_catalog.my_db.my_spark_iceberg_table ALTER
      COLUMN user_id TYPE string;
      ```
      48
- Partition Evolution (EMR - Spark SQL):
  Iceberg allows changing a table's partition specification without rewriting existing data. New data will conform to the new specification, while old data remains queryable using its original partitioning.48
  - **Example (Spark SQL - Add Partition Field):**
    SQL
    ```
    -- Add 'event_source' as a new, unpopulated partition field
    ALTER TABLE my_glue_catalog.my_db.my_spark_iceberg_table ADD PARTITION
    FIELD event_source;

    -- Example: Change existing timestamp partition from days(event_ts) to
    hours(event_ts)
    -- Note: The exact syntax for REPLACE PARTITION FIELD might vary or require
    specific Spark versions/configurations.
    -- A common approach for complex changes is to define a new spec and apply it.
    -- For simplicity, adding a field is more straightforward:
    ```

ALTER TABLE my_glue_catalog.my_db.my_spark_iceberg_table ADD PARTITION FIELD bucket(16, user_id);
[27]

The practicality of these evolution operations, particularly more complex partition changes, can differ between engines. While Iceberg's specification defines how these evolutions work at the format level, the specific DDL syntax and completeness of support can vary. Amazon EMR with Spark often provides more comprehensive DDL capabilities for advanced Iceberg features compared to Athena, which might have limitations on certain table property modifications or advanced DDLs.[27] Consequently, for intricate schema or partition management tasks on AWS, users might find Spark (on EMR or via AWS Glue ETL) to be the more versatile tool, even if their primary query interface is Athena. This underscores the importance of understanding the specific capabilities of each engine's Iceberg integration on AWS, as the choice of tool for management tasks may differ from that used for routine querying.

VI. Essential Maintenance & Best Practices on AWS

Operating Apache Iceberg tables efficiently at scale, especially in a dynamic cloud environment like AWS, necessitates regular maintenance and adherence to best practices. These practices ensure optimal query performance, control storage costs, and maintain data integrity.

A. Table Maintenance: Compaction, Snapshot Expiration, Orphan File Removal

Iceberg's design, while powerful, can lead to performance degradation or increased costs if not properly maintained. Key maintenance operations include:

- **1. Compaction (Rewriting Data Files / Bin Packing):**
  - **Problem:** Frequent small write operations, common in streaming ingestion or small batch updates, can lead to a proliferation of small data files. This "small file problem" negatively impacts query performance due to increased metadata overhead and the higher latency associated with opening many S3 objects.[2]
  - **Solution:** Periodically run data compaction (often called bin packing) to combine these small data files into fewer, larger, and more optimally sized files (e.g., target sizes between 128MB and 1GB, though >100MB is a general guideline). Compaction also serves to merge delete files (generated by merge-on-read DML operations) with their corresponding base data files, physically removing deleted rows.[2]
  - **Implementation on AWS:**
    - **Amazon Athena:** Use the OPTIMIZE <table> REWRITE DATA USING BIN_PACK command. This is suitable for ad-hoc or scheduled compaction managed by Athena.[42]
    - **Amazon EMR / AWS Glue (using Spark):** Execute the rewrite_data_files

      Spark procedure: CALL catalog_name.system.rewrite_data_files(table => 'db.table', options => map('target-file-size-bytes','536870912')). This offers more control over compaction strategies (e.g., sorting) and resource allocation.[31]
    - **AWS Glue Data Catalog Automatic Compaction:** AWS Glue can be configured to automatically monitor Iceberg tables and trigger compaction processes when certain thresholds (e.g., number of small files, size of delete files) are met.[14]
- **2. Snapshot Expiration:**
  - **Problem:** Every write operation (DML, DDL, compaction) creates a new table snapshot. While this enables time travel, retaining an excessive number of old snapshots indefinitely leads to metadata bloat in the catalog and prevents the cleanup of old, unreferenced data files, thereby increasing S3 storage costs.[47]
  - **Solution:** Periodically expire old snapshots based on retention policies (e.g., keep snapshots for the last 30 days, or retain the last N snapshots). Expiring a snapshot removes it from the table's metadata history, making it no longer available for time travel and allowing the underlying data and metadata files that are uniquely referenced by that expired snapshot (and no other active snapshot) to be candidates for physical deletion.[47]
  - **Implementation on AWS:**
    - **Amazon Athena:** Use the VACUUM <table> command. The behavior of VACUUM is controlled by table properties like vacuum_max_snapshot_age_seconds (e.g., TBLPROPERTIES ('vacuum_max_snapshot_age_seconds'='2592000') for 30 days) and vacuum_min_snapshots_to_keep.[42]
    - **Amazon EMR / AWS Glue (using Spark):** Execute the expire_snapshots Spark procedure: CALL catalog_name.system.expire_snapshots(table => 'db.table', older_than => 'timestamp_expression', retain_last => <N>).[47]
    - **AWS Glue Data Catalog Automatic Snapshot Retention:** This feature can be enabled and configured to manage snapshot expiration automatically based on defined policies.[14]
- **3. Orphan File Removal:**
  - **Problem:** Failed write operations, aborted jobs, or sometimes even normal snapshot expiration might leave behind data or metadata files in S3 that are no longer referenced by any valid Iceberg table snapshot. These are known as orphan files and consume storage unnecessarily.[47]
  - **Solution:** Periodically scan the table's S3 location and remove files that are not tracked by any current or retained snapshot's metadata. Caution is needed to ensure a safe retention period (e.g., older than a few days) to avoid deleting files from in-progress writes.[47]
  - **Implementation on AWS:**
    - **Amazon Athena:** The VACUUM <table> command in Athena also handles

the removal of orphan files, typically based on the snapshot retention settings and a safety grace period.[42]

    - **Amazon EMR / AWS Glue (using Spark):** Execute the remove_orphan_files Spark procedure: CALL catalog_name.system.remove_orphan_files(table => 'db.table', older_than => 'timestamp_expression').[47]
    - **AWS Glue Data Catalog Automatic Orphan File Deletion:** This can be configured alongside snapshot retention to automatically clean up unreferenced files.[14]

- **4. Metadata File Compaction (Manifests):**
  - **Problem:** Similar to data files, a high frequency of small commits can lead to a large number of small manifest files. While manifest lists help, excessive manifest files can still slow down query planning.[50]
  - **Solution:** Use the rewrite_manifests Spark procedure to combine smaller manifest files into fewer, larger ones, reducing the metadata overhead for query planners.[50]
  - **Implementation on AWS (EMR/Glue - Spark):** CALL catalog_name.system.rewrite_manifests(table => 'db.table').[50] This is typically run after data compaction.

The necessity of these maintenance tasks cannot be overstated. Iceberg's architecture, which involves creating new metadata files and snapshots for each write, can lead to significant accumulation if not managed. Without regular compaction, the proliferation of small data files and manifests degrades query performance. Without snapshot expiration and orphan file removal, storage costs can escalate due to unreferenced data and bloated metadata history.[2] Therefore, organizations adopting Iceberg must implement a robust strategy for automating these maintenance routines to ensure their data lake remains performant and cost-effective. AWS provides a range of tools, from managed Glue features to flexible EMR procedures and Athena commands, to facilitate this, but the onus of configuration, scheduling, and monitoring these processes remains with the user.[14]

B. Automating Maintenance on AWS

Automating Iceberg table maintenance is crucial for production environments. AWS provides several mechanisms to achieve this:

- **AWS Glue Data Catalog Automatic Optimization:** As highlighted, the Glue Data Catalog offers built-in features to automate compaction, snapshot retention, and orphan file deletion for Iceberg tables. These can be configured at the table level through the AWS console, CLI, or SDK, allowing Glue to manage these tasks based on predefined schedules or thresholds.[14] This is often the simplest approach for standard maintenance needs.
- **Custom Automation with EMR, Lambda, and Orchestration Services:** For more granular control, custom scheduling, or complex maintenance logic, a common pattern involves orchestrating Spark-based maintenance procedures on Amazon EMR.
  - **AWS Glue ETL Jobs:** Scheduled Glue ETL jobs running Spark scripts can execute

Iceberg maintenance procedures.[49]

- **Amazon EMR with AWS Step Functions or Amazon EventBridge + AWS Lambda:** This pattern offers robust orchestration:
    1. An **Amazon EventBridge rule** triggers an **AWS Lambda function** on a defined schedule (e.g., daily, weekly) or based on specific events (e.g., a high number of small files detected via a monitoring script).[65]
    2. The **Lambda function** can contain logic to identify which Iceberg tables require maintenance (e.g., by querying Iceberg metadata tables, checking table properties, or from a predefined list). It then prepares the necessary parameters for the maintenance job.
    3. The Lambda function submits a **step to a running Amazon EMR cluster** or provisions a **transient EMR cluster** specifically for the maintenance task. Using transient clusters can be cost-effective as you only pay for the cluster when it's performing work.[65]
    4. The **EMR step** executes a Spark application (e.g., a Spark SQL script or a Scala/Python Spark job) that calls the relevant Iceberg Spark procedures (rewrite_data_files, expire_snapshots, remove_orphan_files, rewrite_manifests).[52]
    5. If a transient cluster was used, it automatically terminates after the job completes.
    6. **AWS Step Functions** can be used to create more complex workflows involving multiple EMR steps, conditional logic, error handling, and parallel execution of maintenance tasks for different tables or partitions.[65]
- While **PyIceberg with Lambda** can perform some metadata operations, heavier maintenance tasks like data compaction are typically better suited for the distributed processing power of Spark on EMR or Glue.[46]

C. Performance Tuning & Cost Optimization on AWS

Optimizing performance and managing costs are ongoing concerns for any data lake.

- **Performance Tuning Strategies:**
    - **File Format and Compression:** Apache Parquet is generally the preferred file format for analytical workloads with Iceberg due to its columnar nature. Zstandard (ZSTD) compression often provides a good balance between compression ratio and decompression speed.[5]
    - **Optimal File Sizes:** Aim for data file sizes greater than 100MB, typically in the range of 128MB to 512MB (or even up to 1GB depending on the workload). This is controlled via the write.target-file-size-bytes table property and achieved through compaction.[47]
    - **Effective Partitioning:** Design partitioning schemes that align with common query patterns. Utilize Iceberg's hidden partitioning and partition evolution capabilities. For high-cardinality columns (e.g., user IDs), consider transforms like bucket(). For low-cardinality fields (e.g., region), explicit partitions can be

effective.[47]

  - **Predicate Pushdown & Partition Pruning:** Ensure that queries are written to leverage these features, and that query engines are configured to take full advantage of Iceberg's metadata.[47]
  - **Metadata Caching:** Caching frequently accessed metadata files (manifests, manifest lists) can reduce query latency, especially for repeated queries.[47]
  - **EMR/Spark Configuration:** Tune Spark configurations on EMR, such as executor memory, number of cores, shuffle partitions, and parallelism, to match the workload and cluster resources. Use appropriate EMR instance types for I/O or compute-intensive tasks.[47]
- **Cost Optimization Measures:**
  - **Regular Maintenance:** As discussed, compaction and snapshot expiration are crucial for reducing S3 storage costs by eliminating redundant data and small file overhead.[47]
  - **S3 Lifecycle Policies:** For tables with long time-travel requirements, consider using S3 Lifecycle Policies to transition older, infrequently accessed data files (associated with retained but old snapshots) to more cost-effective S3 storage tiers like S3 Glacier Instant Retrieval or S3 Glacier Flexible Retrieval, if query patterns allow for slightly higher retrieval latency for historical data.[47]
  - **Amazon EMR Cost Controls:** Utilize EMR autoscaling to dynamically adjust cluster size based on workload. Leverage Spot Instances for fault-tolerant workloads to significantly reduce compute costs. Use transient EMR clusters for sporadic maintenance or ETL jobs.[47]
  - **Amazon Athena:** Being a pay-per-query service, costs are directly related to the amount of data scanned. Well-maintained and optimized Iceberg tables (compacted, effectively partitioned) lead to less data scanned and thus lower Athena costs.[47]
  - **AWS Glue Automatic Optimization:** Enabling these features can reduce the operational effort and potentially optimize storage and compute associated with manual maintenance jobs.[14]

D. Data Governance with AWS Lake Formation

Data governance is critical for data lakes, and AWS Lake Formation provides a centralized service to manage security and access control for Iceberg tables.

- **Fine-Grained Access Control:** Lake Formation enables defining and enforcing fine-grained permissions for Iceberg tables that are registered in the AWS Glue Data Catalog. This includes permissions at the database, table, column, and even row and cell levels (using data filters).[9]
- **Centralized Management:** Permissions are managed centrally in Lake Formation and are consistently enforced across various AWS analytics services that integrate with it, such as Amazon EMR, Amazon Athena, Amazon Redshift Spectrum, and AWS Glue ETL jobs.

- **Hybrid Access Mode:** For organizations migrating to Lake Formation or needing to support existing IAM-based access patterns, Lake Formation's hybrid access mode allows IAM permissions to coexist with Lake Formation permissions for the same data resources. This facilitates a phased adoption of Lake Formation governance.[48]
- **Setup:** The typical setup involves:
    1. Registering the Amazon S3 locations where Iceberg table data resides with Lake Formation.
    2. Granting permissions (e.g., SELECT, INSERT, ALTER, DESCRIBE) on Glue databases and tables to specific IAM roles or users through Lake Formation.

The integration of Lake Formation with Apache Iceberg represents a significant advancement in providing enterprise-grade, centralized security and governance for open data lakes on AWS. Historically, data lakes often faced challenges in implementing consistent and granular access controls. Lake Formation addresses this by providing a unified control plane over data stored in S3, using the AWS Glue Data Catalog as the central metastore.[48] For Iceberg tables, this means that administrators can define precise access policies for different users or applications (e.g., data scientists, analysts, ETL processes) accessing the same underlying data via EMR, Athena, or Redshift, ensuring that they only see and interact with the data they are authorized for.[69] This capability is crucial for fostering data democratization securely and meeting increasingly stringent compliance and regulatory requirements. The hybrid access mode further simplifies adoption for existing data lakes by allowing a gradual transition to Lake Formation's governance model without disrupting current workflows.[48]

VII. Iceberg in Context: Comparisons & Future Outlook

Understanding Apache Iceberg's position relative to other table formats and its trajectory within the AWS ecosystem provides valuable context for architectural decisions.

A. Brief: Iceberg vs. Delta Lake vs. Hudi

Apache Iceberg, Delta Lake, and Apache Hudi are the leading open table formats, all aiming to supercharge data lakes by imbuing them with data warehouse-like capabilities such as ACID transactions, schema evolution, and time travel.[2] While they share common goals, they have different origins, strengths, and ecosystem focuses.

- **Apache Iceberg:** Originating from Netflix and Apple, Iceberg is renowned for its strong emphasis on open standards, broad multi-engine interoperability, exceptionally robust schema and partition evolution capabilities, and proven performance at petabyte scale. It boasts a large and diverse developer community.[1]
- **Delta Lake:** Developed by Databricks, Delta Lake offers tight integration with the Apache Spark ecosystem. It emphasizes simplicity, ease of use, and features like data quality constraints (e.g., NOT NULL) enforced during writes.[8]
- **Apache Hudi:** Created at Uber, Hudi (Hadoop Upserts Deletes and Incrementals) excels in use cases requiring real-time or near real-time data ingestion, efficient upserts (updates and inserts), and incremental data processing.[12]

When to Choose Iceberg (Recap):
Iceberg is often the preferred choice for:

- Large-scale analytical workloads requiring high performance and reliability.
- Environments demanding strong interoperability across multiple query and processing engines (e.g., Spark, Flink, Trino, Presto, Athena).
- Use cases where flexible and non-disruptive schema and partition evolution are critical.
- Building cloud-native data lakes with an emphasis on open standards and avoiding vendor lock-in at the table format level.[12]

The following table provides a high-level comparison:

Table: Iceberg vs. Delta Lake vs. Hudi: High-Level Comparison

Feature/Aspect	**Apache Iceberg**	**Delta Lake**	**Apache Hudi**
Primary Ecosystem	Multi-engine (Spark, Flink, Trino, Presto, Hive, Athena, etc.)	Spark (strong Databricks alignment)	Spark, Flink (strong streaming/incremental focus)
ACID Transactions	Yes (Optimistic Concurrency)	Yes (Optimistic Concurrency)	Yes (Optimistic Concurrency)
Schema Evolution	Full, robust, non-breaking	Yes, with schema enforcement	Yes
Partition Evolution	Yes, non-breaking	Limited (often requires rewrite for existing data)	Limited (can be complex)
Time Travel	Yes (Snapshot-based)	Yes (Version-based)	Yes (Snapshot-based)
Engine Interoperability	High, designed for openness	Growing, but historically Spark-centric	Growing, good Spark/Flink support
Key Strength/Use Case	Large-scale analytics, multi-engine, open standard, robust evolution	Spark-native simplicity, data quality constraints	Real-time ingestion, upserts, incremental processing
Community/Openness	Apache Software Foundation, large diverse community	Linux Foundation, strong Databricks backing	Apache Software Foundation, active community

Sources: [2]

This comparison helps clarify Iceberg's unique value. While all formats bring improvements, Iceberg's design prioritizes openness and robust metadata management, making it particularly suitable for complex, evolving data landscapes that need to be accessed by diverse tools, a common scenario in large enterprises and on platforms like AWS.

B. Key Use Cases on AWS

Leveraging AWS services, Apache Iceberg enables a variety of powerful use cases:

- **Data Warehousing and Lakehouse Modernization:** Organizations are increasingly using Iceberg on Amazon S3 as the foundation for their analytical platforms, effectively building data lakehouses. This allows them to combine the cost-efficiency and scalability of S3 with the performance and transactional capabilities of Iceberg, queried by engines like Athena, EMR, and Redshift Spectrum.[3]
- **Transactional Data Lakes:** Iceberg's ACID transactions and support for row-level updates and deletes make it ideal for managing data lakes that require frequent modifications. This is crucial for use cases like implementing GDPR "right to be forgotten" requirements, managing Slowly Changing Dimensions (SCDs) in customer or product tables, or correcting data errors without full table rewrites.[3]
- **Real-time Analytics and Streaming Ingestion:** Combined with stream processing engines like Apache Flink or Spark Structured Streaming on Amazon EMR, Iceberg can serve as a reliable sink for real-time data. Ingested data becomes immediately available for querying, enabling real-time dashboards, monitoring, and near real-time analytical applications.[3]
- **Quantitative Finance Research Platforms:** The financial industry leverages Iceberg on AWS for high-performance research platforms. Iceberg's query performance, efficient data correction capabilities (updates/deletes), and time travel features (for backtesting and auditability) are highly valued in this domain.[76]
- **Data Lake Migration:** Iceberg provides a robust target format for migrating from older data lake technologies, such as Hive. Companies like Natural Intelligence have successfully migrated their Hive data lakes to Iceberg on AWS, using services like AWS Glue, EMR, and EventBridge to manage the transition and synchronization, thereby gaining Iceberg's advanced features and performance benefits.[27]

These use cases demonstrate the practical value of Iceberg in solving real-world data challenges, with AWS providing the scalable infrastructure and integrated services to support these solutions.

C. The Future of Iceberg and AWS Innovations

Apache Iceberg is a rapidly evolving project with a vibrant community and strong industry backing, including significant contributions and innovations from AWS.

- **Iceberg Community Developments:**
    - **Table Specification V3:** The community is actively working on Iceberg Table Specification V3, which is expected to introduce significant enhancements such as native support for semi-structured data (a VARIANT type), official support for views and materialized views, more efficient binary delete vectors (improving on positional/equality deletes), and row lineage tracking for better data traceability.[77]
    - **Beyond V3:** Visionary ideas for future versions (e.g., V4) include single-file commits (which could optimize writes for very small or frequent updates, beneficial for streaming), and more adaptive metadata structures to further enhance performance and reduce overhead, particularly for smaller table scenarios.[77]

    - **Expanding Language Ecosystem:** There's a growing focus on non-JVM integrations, with libraries like PyIceberg (Python) and efforts in Go gaining traction. This broadens Iceberg's accessibility to a wider range of developers and tools.[14]
    - **Native Geospatial Type Support:** The introduction of native geospatial types into Iceberg is underway, aiming to standardize storage and enable optimized geospatial analytics directly within the Iceberg ecosystem, leveraging engines like Apache Spark with Apache Sedona.[78]
- **AWS Innovations and Trends:**
    - **Amazon S3 Tables:** AWS recently announced Amazon S3 Tables, a managed offering for Iceberg tables on S3. This service aims to simplify operations by automating maintenance tasks like compaction and snapshot management, providing an easier on-ramp to using Iceberg.[11]
    - **AWS Glue Data Catalog Enhancements:** AWS continues to invest heavily in the Glue Data Catalog's Iceberg capabilities. This includes more advanced automatic optimization features like improved compaction (handling delete files, nested types, partial progress commits), automated snapshot retention, and orphan file deletion, as well as support for partition evolution.[14]
    - **Deeper AWS Lake Formation Integration:** Expect continued enhancements in fine-grained access control and governance for Iceberg tables through AWS Lake Formation, making it easier to secure data lakes.[48]
    - **Broader Query Engine Support for REST Catalogs:** The Iceberg REST Catalog specification is gaining adoption. For example, DuckDB recently added support, which is relevant for querying AWS services like Amazon S3 Tables and Amazon SageMaker Lakehouse that may expose Iceberg tables via this standard interface.[11]

These developments, both from the open-source community and AWS, point towards a future where Apache Iceberg becomes even more powerful, easier to manage, and applicable to a broader array of data management challenges. The focus on reducing operational overhead (e.g., small file problem, maintenance complexity) and expanding capabilities (e.g., semi-structured data, geospatial support) will likely accelerate Iceberg's adoption further.[11] As Iceberg matures and simplifies, it increasingly solidifies its position as the de facto open standard for data lake tables, potentially displacing older formats and even challenging proprietary data warehouse formats in a wider range of analytical scenarios.

VIII. Conclusion & Key Takeaways

Apache Iceberg has fundamentally transformed the landscape of data lakes, bringing much-needed reliability, performance, and flexibility. Its core value proposition lies in enabling data warehouse-like capabilities—ACID transactions, robust schema and partition evolution, and time travel—directly on open, scalable, and cost-effective cloud storage like Amazon S3. The key benefits are clear:

- **Reliability:** ACID transactions ensure data integrity, even with concurrent operations.

- **Flexibility:** Schema and partition evolution allow data architectures to adapt to changing business needs without disruptive and costly rewrites.
- **Performance:** Intelligent metadata management, including predicate pushdown and partition pruning, significantly accelerates queries on massive datasets.
- **Openness:** As an open standard with a strong community, Iceberg promotes interoperability across a wide array of processing engines and avoids vendor lock-in at the table format level.

The Amazon Web Services ecosystem provides a powerful and comprehensive platform for building and operating Iceberg-based solutions. Key services like Amazon S3 for storage, AWS Glue Data Catalog for metadata management, Amazon EMR and AWS Glue ETL for processing, Amazon Athena and Amazon Redshift for querying, and AWS Lake Formation for governance, all offer robust integration with Apache Iceberg. This synergy allows organizations to construct sophisticated, end-to-end data lakehouse architectures tailored to their specific needs.

Looking ahead, Apache Iceberg is not just a technology for today but a strategic choice for future-proof data architectures. Its ongoing evolution, driven by a vibrant open-source community and complemented by innovations from cloud providers like AWS, promises to further simplify data management, enhance performance, and broaden its applicability. For organizations aiming to unlock the full potential of their data with agility and confidence, Apache Iceberg on AWS presents a compelling and future-oriented path.

To delve deeper, exploring AWS workshops, official documentation, and experimenting with Iceberg on AWS services is highly recommended.[80] The journey towards a modern, efficient, and reliable data lake is significantly advanced by embracing Apache Iceberg.

Referências citadas

1. Apache Iceberg: The Basics | Qlik Blog, acessado em maio 26, 2025, https://www.qlik.com/blog/apache-iceberg-the-basics
2. Apache Iceberg | What is it? Why Use It? | Starburst, acessado em maio 26, 2025, https://www.starburst.io/data-glossary/apache-iceberg/
3. What are Apache Iceberg tables? Benefits and challenges - Redpanda, acessado em maio 26, 2025, https://www.redpanda.com/blog/apache-iceberg-tables-benefits-challenges
4. Why Apache Iceberg — for data lake users – Tabular, acessado em maio 26, 2025, https://www.tabular.io/apache-iceberg-cookbook/introduction-data-lake-perspective/
5. Apache Iceberg Architecture: 3 Core Components to Understand, acessado em maio 26, 2025, https://atlan.com/know/iceberg/apache-iceberg-architecture/
6. What Is Apache Iceberg Data Catalog? Discover Your 2025 Options, acessado em maio 26, 2025, https://atlan.com/know/iceberg/apache-iceberg-data-catalog/
7. The Evolution of Apache Iceberg Catalogs - Dremio, acessado em maio 26, 2025, https://www.dremio.com/blog/the-evolution-of-apache-iceberg-catalogs/
8. The 2025 Comprehensive Guide to Apache Iceberg - DEV Community, acessado em maio 26, 2025,

https://dev.to/alexmercedcoder/the-2025-comprehensive-guide-to-apache-iceberg-2g22
9. Apache Iceberg & AWS: A Practical Guide [2025] - Atlan, acessado em maio 26, 2025, https://atlan.com/know/iceberg/apache-iceberg-aws/
10. Getting started with Apache Iceberg tables in Amazon Athena SQL - AWS Prescriptive Guidance, acessado em maio 26, 2025, https://docs.aws.amazon.com/prescriptive-guidance/latest/apache-iceberg-on-aws/getting-started.html
11. R2 Data Catalog: Managed Apache Iceberg tables with zero egress ..., acessado em maio 26, 2025, https://blog.cloudflare.com/r2-data-catalog-public-beta/
12. Comparing Apache Hudi, Apache Iceberg, and Delta Lake - CloudThat, acessado em maio 26, 2025, https://www.cloudthat.com/resources/blog/comparing-apache-hudi-apache-iceberg-and-delta-lake
13. What is Apache Iceberg? - Iceberg Tables Explained - AWS, acessado em maio 26, 2025, https://aws.amazon.com/what-is/apache-iceberg/
14. Apache Iceberg | AWS Big Data Blog, acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/tag/apache-iceberg/
15. Seamless Analytics with Amazon Redshift and Apache Iceberg on ..., acessado em maio 26, 2025, https://www.cloudthat.com/resources/blog/seamless-analytics-with-amazon-redshift-and-apache-iceberg-on-amazon-s3
16. Climbing the Iceberg with ClickHouse, acessado em maio 26, 2025, https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse
17. Delta Lake and Iceberg communities collide – in a good way - The Register, acessado em maio 26, 2025, https://www.theregister.com/2025/04/15/iceberg_delta_lake_collab/
18. Amazon adds Iceberg to make S3 a more functional data lake - Blocks and Files, acessado em maio 26, 2025, https://blocksandfiles.com/2024/12/10/amazon-makes-s3-a-more-functional-data-lake/
19. AWS - Apache Iceberg - The Apache Software Foundation, acessado em maio 26, 2025, https://apache.github.io/iceberg/docs/1.4.3/aws/
20. AWS - Apache Iceberg™ - The Apache Software Foundation, acessado em maio 26, 2025, https://iceberg.apache.org/docs/latest/aws/
21. AWS - Apache Iceberg™, acessado em maio 26, 2025, https://iceberg.apache.org/docs/nightly/aws/
22. AWS - Apache Iceberg™, acessado em maio 26, 2025, https://iceberg.apache.org/docs/nightly/docs/aws/
23. AWS - Apache Iceberg™, acessado em maio 26, 2025, https://iceberg.apache.org/docs/1.8.1/aws/
24. Top 10 Query Engines for Apache Iceberg: A Complete Comparison ..., acessado em maio 26, 2025, https://estuary.dev/comparison-query-engines-for-apache-iceberg/
25. ClickHouse vs StarRocks vs Presto vs Trino vs Apache Spark™ — Comparing

Analytics Engines - Onehouse, acessado em maio 26, 2025, https://www.onehouse.ai/blog/apache-spark-vs-clickhouse-vs-presto-vs-starrocks-vs-trino-comparing-analytics-engines
26. Spark Writes - Apache Iceberg - The Apache Software Foundation, acessado em maio 26, 2025, https://iceberg.apache.org/docs/1.5.0/spark-writes/
27. How to add partition fields to Iceberg table | AWS re:Post, acessado em maio 26, 2025, https://repost.aws/questions/QUDG1xddnfScaPtJ1Nheelxw/how-to-add-partition-fields-to-iceberg-table
28. Working with Apache Iceberg tables by using Apache Spark - AWS ..., acessado em maio 26, 2025, https://docs.aws.amazon.com/prescriptive-guidance/latest/apache-iceberg-on-aws/iceberg-spark.html
29. Spark Configuration - Apache Iceberg, acessado em maio 26, 2025, https://iceberg.apache.org/docs/1.5.0/spark-configuration/
30. Spark Writes - Apache Iceberg, acessado em maio 26, 2025, https://iceberg.apache.org/docs/1.5.1/spark-writes/
31. Spark Procedures - Apache Iceberg, acessado em maio 26, 2025, https://iceberg.apache.org/docs/1.6.1/spark-procedures/
32. Flink Connector - Apache Iceberg™, acessado em maio 26, 2025, https://iceberg.apache.org/docs/1.4.3/flink-connector/
33. Build a data lake with Apache Flink on Amazon EMR | AWS Big Data ..., acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/build-a-unified-data-lake-with-apache-flink-on-amazon-emr/
34. Introduction - Apache Iceberg™, acessado em maio 26, 2025, https://iceberg.apache.org/docs/nightly/
35. Partitioning - Apache Iceberg™, acessado em maio 26, 2025, https://iceberg.apache.org/docs/latest/partitioning/
36. How to create an Iceberg table using base-2 file layout in Athena? | AWS re:Post, acessado em maio 26, 2025, https://repost.aws/questions/QU6jHRH4bMQ7OihsTYFGjU1g/how-to-create-an-iceberg-table-using-base-2-file-layout-in-athena
37. Creating Apache Iceberg tables - AWS Lake Formation, acessado em maio 26, 2025, https://docs.aws.amazon.com/lake-formation/latest/dg/creating-iceberg-tables.html
38. Use an Iceberg cluster with Hive - Amazon EMR - AWS Documentation, acessado em maio 26, 2025, https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-iceberg-use-hive-cluster.html
39. acessado em dezembro 31, 1969, https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-iceberg-hive.html
40. Use an Iceberg cluster with Trino - Amazon EMR, acessado em maio 26, 2025, https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-iceberg-use-trino-cl

uster.html
41. acessado em dezembro 31, 1969, https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-iceberg-trino.html
42. Create Iceberg tables - Amazon Athena - AWS Documentation, acessado em maio 26, 2025, https://docs.aws.amazon.com/athena/latest/ug/querying-iceberg-creating-tables.html
43. ALTER TABLE SET TBLPROPERTIES - Amazon Athena - AWS Documentation, acessado em maio 26, 2025, https://docs.aws.amazon.com/athena/latest/ug/querying-iceberg-alter-table-set-properties.html
44. Working with Apache Iceberg tables by using Amazon Athena SQL - AWS Prescriptive Guidance, acessado em maio 26, 2025, https://docs.aws.amazon.com/prescriptive-guidance/latest/apache-iceberg-on-aws/iceberg-athena.html
45. Query Apache Iceberg tables - Amazon Athena - AWS Documentation, acessado em maio 26, 2025, https://docs.aws.amazon.com/athena/latest/ug/querying-iceberg.html
46. Accelerate lightweight analytics using PyIceberg with AWS Lambda ..., acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/accelerate-lightweight-analytics-using-pyiceberg-with-aws-lambda-and-an-aws-glue-iceberg-rest-endpoint/
47. Expert Checklist for Apache Iceberg on AWS: From the Essentials to Advanced Optimization, acessado em maio 26, 2025, https://community.aws/content/2rfuYeCPh6A7TzNaBjmnXLwHX75/expert-checklist-for-apache-iceberg-on-aws-from-the-essentials-to-advanced-optimization
48. Read and write Apache Iceberg tables using AWS Lake Formation hybrid access mode, acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/read-and-write-apache-iceberg-tables-using-aws-lake-formation-hybrid-access-mode/
49. Use AWS Glue ETL to perform merge, partition evolution, and ..., acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/use-aws-glue-etl-to-perform-merge-partition-evolution-and-schema-evolution-on-apache-iceberg/
50. amazon web services - How do I manage Apache Iceberg metadata ..., acessado em maio 26, 2025, https://stackoverflow.com/questions/77762787/how-do-i-manage-apache-iceberg-metadata-that-grows-exponentially-in-aws
51. Maintaining tables by using compaction - AWS Prescriptive Guidance, acessado em maio 26, 2025, https://docs.aws.amazon.com/prescriptive-guidance/latest/apache-iceberg-on-aws/best-practices-compaction.html
52. Apache Iceberg Maintenance | IOMETE, acessado em maio 26, 2025, https://iomete.com/resources/reference/iceberg-tables/maintenance
53. Procedures - Apache Iceberg™, acessado em maio 26, 2025,

https://iceberg.apache.org/docs/1.8.0/spark-procedures/
54. Iceberg Spark Procedures for Snapshot and Metadata - IOMETE, acessado em maio 26, 2025, https://iomete.com/resources/reference/iceberg-tables/iceberg-procedures
55. Procedures - Apache Iceberg™, acessado em maio 26, 2025, https://iceberg.apache.org/docs/latest/spark-procedures/
56. Automated optimization for Apache Iceberg tables using Glue Data Catalog | Amazon Web Services - YouTube, acessado em maio 26, 2025, https://www.youtube.com/watch?v=xOXE7AS-pNA
57. Accelerate queries on Apache Iceberg tables through AWS Glue auto compaction, acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/accelerate-queries-on-apache-iceberg-tables-through-aws-glue-auto-compaction/
58. Can I enable optimization in AWS Glue when I create an iceberg table instead of the using the console after the table is created? | AWS re:Post, acessado em maio 26, 2025, https://repost.aws/questions/QUuSNzjdFvSAGPNCvy722nlg/can-i-enable-optimization-in-aws-glue-when-i-create-an-iceberg-table-instead-of-the-using-the-console-after-the-table-is-created
59. AWS Glue Data Catalog offers advanced automatic optimization for ..., acessado em maio 26, 2025, https://aws.amazon.com/about-aws/whats-new/2024/12/aws-glue-data-catalog-automatic-optimization-iceberg-tables/
60. The AWS Glue Data Catalog now supports storage optimization of Apache Iceberg tables, acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/the-aws-glue-data-catalog-now-supports-storage-optimization-of-apache-iceberg-tables/
61. Snapshot retention optimization - AWS Glue - AWS Documentation, acessado em maio 26, 2025, https://docs.aws.amazon.com/glue/latest/dg/snapshot-retention-management.html
62. Enabling orphan file deletion - AWS Glue, acessado em maio 26, 2025, https://docs.aws.amazon.com/glue/latest/dg/enable-orphan-file-deletion.html
63. how to enable in compaction status, Snapshot retention status, orphan file deletion status on aws glue tables.for reference i have · Issue #1829 · apache/iceberg-python - GitHub, acessado em maio 26, 2025, https://github.com/apache/iceberg-python/issues/1829
64. Automated optimization for Apache Iceberg tables | AWS OnAir S05 - YouTube, acessado em maio 26, 2025, https://www.youtube.com/watch?v=wFoDmlyjWbw
65. Getting Started with AWS EMR: Architecture, Pricing, and Best Practices - Cloudchipr, acessado em maio 26, 2025, https://cloudchipr.com/blog/aws-emr
66. Melting the ice — How Natural Intelligence simplified a data lake migration to Apache Iceberg - AWS, acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/melting-the-ice-how-natural-intelligence-simplified-a-data-lake-migration-to-apache-iceberg/

67. Use an Iceberg cluster with Spark - Amazon EMR - AWS Documentation, acessado em maio 26, 2025, https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-iceberg-use-spark-cluster.html
68. S3 Tables with Apache Iceberg: Manage Data at Scale | Decube, acessado em maio 26, 2025, https://www.decube.io/post/s3-tables-apache-iceberg
69. Announcing fine-grained access control via AWS Lake Formation with EMR on EKS, acessado em maio 26, 2025, https://aws.amazon.com/about-aws/whats-new/2025/02/fine-grained-control-aws-lake-formation-emr-eks/
70. Access data in Amazon S3 Tables using PyIceberg through the AWS Glue Iceberg REST endpoint, acessado em maio 26, 2025, https://aws.amazon.com/blogs/storage/access-data-in-amazon-s3-tables-using-pyiceberg-through-the-aws-glue-iceberg-rest-endpoint/
71. Setting up permissions for open table storage formats in Lake ..., acessado em maio 26, 2025, https://docs.aws.amazon.com/lake-formation/latest/dg/otf-tutorial.html
72. Enforce fine-grained access control on Open Table Formats via Amazon EMR integrated with AWS Lake Formation | AWS Big Data Blog, acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/enforce-fine-grained-access-control-on-open-table-formats-via-amazon-emr-integrated-with-aws-lake-formation/
73. acessado em dezembro 31, 1969, https://aws.amazon.com/blogs/big-data/manage-fine-grained-access-control-on-apache-iceberg-tables-with-aws-lake-formation/
74. Delta Lake vs. Apache Iceberg vs. Hudi: Choosing the Best Format, acessado em maio 26, 2025, https://www.hashstudioz.com/blog/delta-lake-vs-apache-iceberg-vs-hudi-choosing-the-right-format-for-big-data-lakes/
75. Why Iceberg + Python is the Future of Open Data Lakes - dltHub, acessado em maio 26, 2025, https://dlthub.com/blog/iceberg-open-data-lakes
76. Build a high-performance quant research platform with Apache ..., acessado em maio 26, 2025, https://aws.amazon.com/blogs/big-data/build-a-high-performance-quant-research-platform-with-apache-iceberg/
77. Highlights from the Iceberg Summit 2025 - Quesma, acessado em maio 26, 2025, https://quesma.com/blog-detail/highlights-iceberg-summit-2025
78. Iceberg Geo Type: Transforming Geospatial Data Management at ..., acessado em maio 26, 2025, https://www.databricks.com/dataaisummit/session/iceberg-geo-type-transforming-geospatial-data-management-scale
79. AWS re:Invent 2018: [NEW LAUNCH!] Intro to AWS Lake Formation - Build a secure data lake (ANT396) - YouTube, acessado em maio 26, 2025, https://www.youtube.com/watch?v=nsiLMqg654s
80. Immersion Day - Data Foundation with Apache Iceberg on AWS (Virtual),

acessado em maio 26, 2025, https://aws-experience.com/emea/north/e/c09f2/immersion-day---data-foundation-with-apache-iceberg-on-aws-virtual
81. Apache Iceberg 101 - Your Guide to Learning Apache Iceberg Concepts and Practices, acessado em maio 26, 2025, https://www.dremio.com/blog/apache-iceberg-101-your-guide-to-learning-apache-iceberg-concepts-and-practices/