



在 Apache Iceberg 上使用 AWS

AWS 规范性指导



AWS 规范性指导: 在 Apache Iceberg 上使用 AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

简介	1
现代数据湖	3
现代数据湖中的高级用例	3
阿帕奇冰山简介	4
AWS 支持 Apache Iceberg	4
Athena SQL 中的 Iceberg 表入门	7
创建未分区的表	7
创建分区表	8
使用单个 CTAS 语句创建表并加载数据	8
插入、更新和删除数据	9
查询冰山表	9
冰山桌解剖图	10
在 Amazon EMR 中使用 Iceberg	12
版本和功能兼容性	12
使用 Iceberg 创建 Amazon EMR 集群	12
在 Amazon EMR 中开发 Iceberg 应用程序	13
使用亚马逊 EMR Studio 笔记本电脑	13
在 Amazon EMR 中运行 Iceberg 职位	14
亚马逊 EMR 的最佳实践	18
在 Iceberg 中工作 AWS Glue	19
使用原生 Iceberg 集成	19
使用自定义 Iceberg 版本	20
Iceberg 的 Spark 配置 AWS Glue	20
AWS Glue 工作最佳实践	22
使用 Spark 处理冰山表	23
创建和写作 Iceberg 表	23
使用 Spark SQL	23
使用 DataFrames API	24
更新 Iceberg 表中的数据	25
更新 Iceberg 表中的数据	26
删除 Iceberg 表中的数据	26
读取数据	27
使用时空旅行	27
使用增量查询	28

访问元数据	28
使用 Trino 处理冰山牌桌	30
EC2 设置上的 Amazon EMR	30
创建 Iceberg 表	31
从冰山桌上读书	32
将数据更新到 Iceberg 表中	32
从 Iceberg 表中删除记录	33
查询 Iceberg 表元数据	33
使用时空旅行	33
将 Iceberg 与 Trino 搭配使用时的注意事项	34
使用 Firehose 处理冰山牌桌	35
使用 Athena SQL 处理 Iceberg 表	36
版本和功能兼容性	36
冰山桌规格支持	36
冰山功能支持	36
使用 Iceberg 桌子	37
使用使用 Iceberg 表来处理冰山表 Pylceberg	38
先决条件	38
连接到数据目录	38
列出和创建数据库	39
创建和写作 Iceberg 表	39
未分区的表	39
分区表	40
读取数据	42
删除数据	43
访问元数据	43
使用时空旅行	43
使用 Iceberg 表格格式规范版本 3	45
版本 3 中的主要功能	45
版本兼容性	45
版本 3 入门	46
先决条件	46
创建版本 3 表	46
启用删除向量	47
使用行谱系进行变更跟踪	47
版本 3 的最佳实践	48

何时使用版本 3	48
优化写入性能	49
优化读取性能	49
迁移策略	49
兼容性注意事项	50
问题排查	50
常见问题	50
获取帮助	51
定价	51
可用性	51
其他资源	51
将现有表迁移到 Iceberg	52
In-place 迁移	52
选项 1：快照过程	53
选项 2：迁移程序	55
在中复制表迁移过程 AWS Glue Data Catalog	59
就地迁移后保持 Iceberg 表同步	59
选择正确的就地迁移策略	61
完整数据迁移	63
选择迁移策略	64
迁移选项摘要	65
优化 Iceberg 工作负载的最佳实践	69
一般最佳实践	69
优化读取性能	70
分区	70
调整文件大小	72
优化列统计信息	73
选择正确的更新策略	74
使用 ZSTD 压缩	74
设置排序顺序	75
优化写入性能	77
设置表格分配模式	77
选择正确的更新策略	77
选择正确的文件格式	78
优化存储	78
启用 S3 Intelligent-Tiering	79

存档或删除历史快照	79
删除孤立文件	82
使用压缩来维护表格	82
冰山压实	82
调整压实行为	84
在 Amazon EMR 上使用 Spark 运行压缩或 AWS Glue	84
使用亚马逊 Athena 进行压缩	85
跑步压实的建议	86
在 Amazon S3 中使用冰山工作负载	87
防止热分区 (HTTP 503 错误)	87
使用 Iceberg 维护操作来释放未使用的数据	87
跨复制数据 AWS 区域	87
监控冰山工作负载	90
Table-level 监控	90
Database-level 监控	92
预防性维护	93
治理和访问控制	94
参考架构	95
每晚批量摄取	95
结合了批量和近实时摄取的数据湖	96
资源	97
贡献者	98
文档历史记录	100
.....	ci

在 Apache Iceberg 上使用 AWS

亚马逊 Web Services ([贡献者](#))

2025 年 11 月 ([文件历史记录](#))

Apache Iceberg 是一种开源表格格式，可在提高性能的同时简化表管理。AWS 亚马逊 EMR AWS Glue、Amazon Athena 和 Amazon Redshift 等分析服务包括对 Iceberg 的原生支持，因此您可以在亚马逊简单存储服务 (Amazon S3) 之上轻松构建交易数据湖。AWS

此外，下一代 Amazon 建立 SageMaker 在[开放的湖库架构之上，该架构](#)统一了数据湖、AWS 数据仓库以及第三方和联合来源的数据访问。Lakehouse 与 Iceberg 完全兼容，让您灵活地使用 Iceberg REST API 访问和查询现有数据。

本技术指南提供了有关如何开始使用 Iceberg 的指导 AWS 服务，并包括在优化成本和性能的 AWS 同时大规模运行 Iceberg 的最佳实践和建议。

无论你是刚开始使用 Iceberg 还是想要优化现有 Iceberg 工作负载的经验丰富的用户 AWS，本指南都能为项目的每个阶段提供宝贵的见解

在本指南中：

- [现代数据湖](#)
- [Athena SQL 中的 Iceberg 表入门](#)
- [在 Amazon EMR 中使用 Iceberg](#)
- [在 Iceberg 中工作 AWS Glue](#)
- [使用 Spark 处理冰山表](#)
- [使用 Trino 处理冰山牌桌](#)
- [使用 Amazon Data Firehose 处理 Iceberg 表](#)
- [使用 Athena SQL 处理 Iceberg 表](#)
- [使用使用 Iceberg 表来处理冰山表 Pylceberg](#)
- [使用 Iceberg 表格格式规范版本 3](#)
- [将现有表迁移到 Iceberg](#)
- [优化 Iceberg 工作负载的最佳实践](#)
- [监控冰山工作负载](#)
- [治理和访问控制](#)

- [参考架构](#)
- [资源](#)
- [贡献者](#)

现代数据湖

现代数据湖中的高级用例

数据存储的演变已从数据库发展到数据仓库和数据湖，每种技术都能满足独特的业务和数据需求。传统数据库擅长处理结构化数据和事务性工作负载，但随着数据量的增加，它们面临着性能挑战。数据仓库的出现是为了解决性能和可扩展性问题，但与数据库一样，它们依赖垂直集成系统中的专有格式。

就成本、可扩展性和灵活性而言，数据湖是存储数据的最佳选择之一。您可以使用数据湖以低成本保留大量结构化和非结构化数据，并将这些数据用于不同类型的分析工作负载，从商业智能报告到大数据处理、实时分析、机器学习和生成式人工智能 (AI)，以帮助指导更好的决策。

尽管有这些好处，但最初设计的数据湖并不是具有类似数据库的功能。数据湖不支持原子性、一致性、隔离性和耐久性 (ACID) 处理语义，您可能需要这些语义才能使用多种不同的技术在成百上千的用户中有效地优化和管理数据。数据湖不为以下功能提供原生支持：

- 随着业务中数据的变化，执行高效的记录级别更新和删除
- 在表增长到数百万个文件和数十万个分区时管理查询性能
- 确保多个并发写入器和读取器的数据一致性
- 防止写入操作在操作中途中失败时数据损坏
- 在没有（部分）重写数据集的情况下，随着时间的推移不断演变表架构

这些挑战在处理变更数据捕获 (CDC) 等用例中变得特别普遍，或者与隐私、数据删除和流数据摄取相关的用例，这可能会导致表格不理想。

使用传统 Hive-format 表的数据湖仅支持对整个文件的写入操作。这使得更新和删除难以实现、耗时且成本高昂。此外，还需要在 ACID-compliant 系统中提供并发控制和保障，以确保数据的完整性和一致性。

这些挑战给用户带来了两难境地：在完全集成但专有的平台之间做出选择，或者选择供应商中立但资源密集型的自建数据湖，需要持续的维护和迁移才能实现其潜在价值。

[为了帮助克服这些挑战，Iceberg 提供了其他类似数据库的功能，简化了数据湖的优化和管理开销，同时仍支持在 Amazon S3 等经济实惠的系统上进行存储。](#)

阿帕奇冰山简介

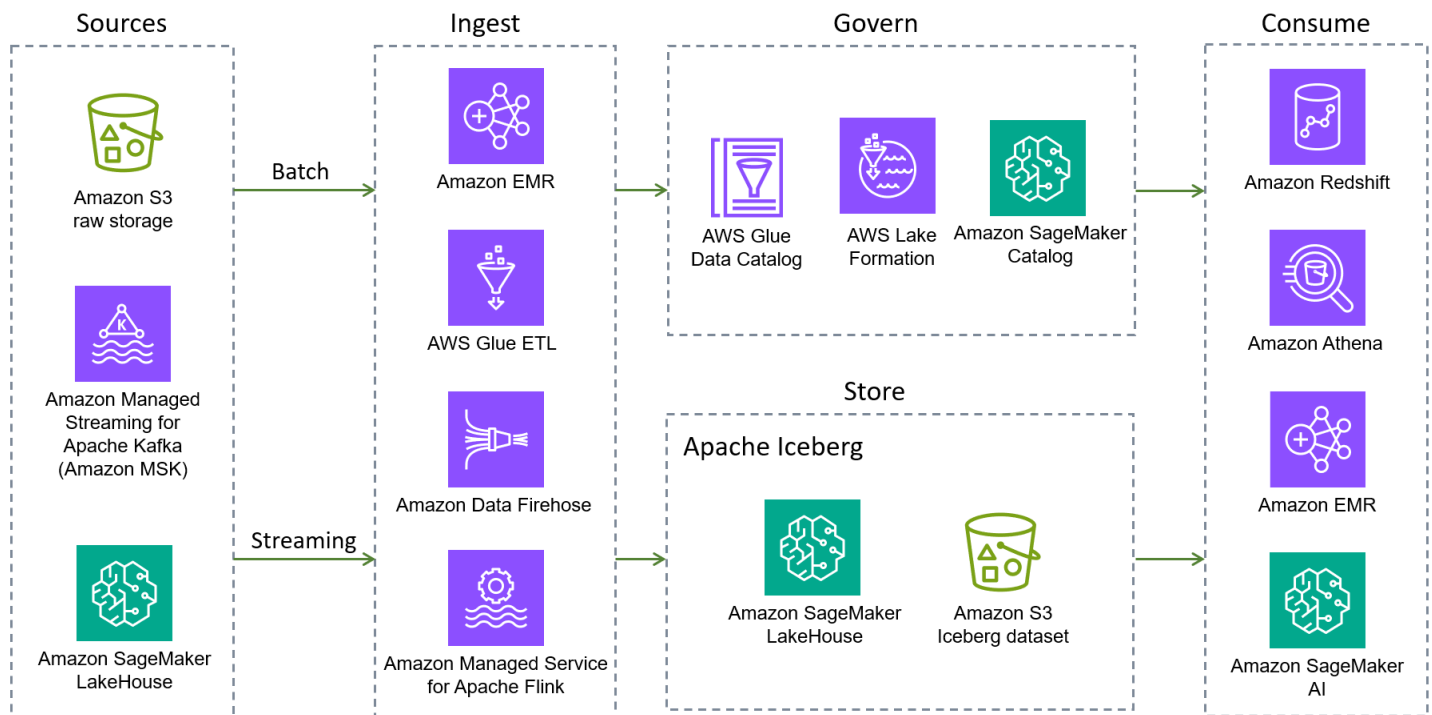
Apache Iceberg 是一种开源表格式，它在数据湖表中提供以前只能在数据库或数据仓库中使用的功能。它专为扩展和性能而设计，非常适合管理超过数百 GB 的表。冰山桌的一些主要特点是：

- 删除、更新和合并。Iceberg 支持用于数据仓库的标准 SQL 命令，用于数据湖表。
- 快速扫描计划和高级筛选。Iceberg 存储诸如分区和列级统计数据之类的元数据，引擎可以使用这些元数据来加快规划和运行查询的速度。
- 完整的架构演变。Iceberg 支持添加、删除、更新或重命名列而不会产生副作用。
- 分区的演变。随着数据量或查询模式的变化，您可以更新表的分区布局。Iceberg 支持更改对表进行分区的列，也支持在复合分区中添加列或从复合分区中删除列。
- 隐藏分区。此功能可防止自动读取不必要的分区。这样用户就不必了解表的分区细节或在查询中添加额外的过滤器。
- 版本回滚。用户可以通过恢复到交易前状态来快速纠正问题。
- 时空旅行。用户可以查询表的特定先前版本。
- 可序列化的隔离。表更改是原子性的，因此读者永远不会看到部分或未提交的更改。
- 并发作家。Iceberg 使用乐观并发来允许多笔交易成功。如果发生冲突，其中一位作者必须重试交易。
- 打开文件格式。[Iceberg 支持多种开源文件格式，包括 Apache Parquet、Apache Avro 和 Apache ORC。](#)

总而言之，使用 Iceberg 格式的数据湖受益于事务一致性、速度、规模和架构演变。有关这些功能和其他 Iceberg 功能的更多信息，请参阅 [Apache Iceberg](#) 文档。

AWS 支持 Apache Iceberg

[Apache Iceberg 由亚马逊 EMR AWS 服务、亚马逊 Athena、亚马逊 Redshift 和亚马逊等提供支持。AWS Glue SageMaker](#) 下图描绘了基于 Iceberg 的数据湖的简化参考架构。



以下内容 AWS 服务 提供原生 Iceberg 集成。还有一些 AWS 服务 可以与Iceberg进行间接交互或通过打包Iceberg库进行交互。

- [Amazon S3](#) 凭借其持久性、可用性、可扩展性、安全性、合规性和审计能力，是构建数据湖的最佳场所。Iceberg 的设计和构建旨在与亚马逊 S3 无缝交互，并为 [Iceberg](#) 文档中列出的许多 Amazon S3 功能提供支持。此外，[Amazon S3 Table](#) s 还提供了第一个具有内置 Iceberg 支持的云对象存储，并简化了大规模表格数据的存储。借助对 Iceberg 的 S3 Tables 支持，您可以使用流行 AWS 的第三方查询引擎轻松查询表格数据。
- [下一代基于开放的 SageMaker湖库架构](#)，该架构统一了 Amazon S3 数据湖、Amazon Redshift 数据仓库以及第三方和联合数据源之间的数据访问。这些功能可帮助您在单个数据副本上构建强大的分析和 AI/ML 应用程序。湖屋与 Iceberg 完全兼容，因此您可以使用 Iceberg REST API 灵活地访问和查询现有数据。
- [Amazon EMR](#) 是一款大数据解决方案，使用 Apache Spark、Flink、Trino 和 Hive 等开源框架，用于千兆字节规模的数据处理、交互式分析和机器学习。亚马逊 EMR 可以在自定义的亚马逊弹性计算云 (Amazon EC2) 集群、亚马逊弹性 Kubernetes Service (亚马逊 EKS) 或亚马逊 EMR Serverless 上运行。AWS Outposts
- [Amazon Athena](#) 是一项基于开源框架的无服务器交互式分析服务。它支持开放表和文件格式，并提供了一种简化、灵活的方式来分析存放的 PB 级数据。Athena 为 Iceberg 的读取、时空旅行、写入和 DDL 查询提供原生支持，并将其用 AWS Glue Data Catalog 于 Iceberg 元数据库。

- [Amazon Redshift](#) 是一个 PB 级的云数据仓库，支持基于集群和无服务器的部署选项。Amazon Redshift Spectrum 可以查询注册并存储 AWS Glue Data Catalog 在亚马逊 S3 上的外部表。Redshift Spectrum 还支持 Iceberg 存储格式。
- [AWS Glue](#) 是一项无服务器数据集成服务，可以更轻松地发现、准备、移动和集成来自多个来源的数据，用于分析、机器学习 (ML) 和应用程序开发。它与 Iceberg 完全集成。具体而言，您可以使用 AWS Glue 作业对 Iceberg 表执行读写操作，通过 [AWS Glue Data Catalog](#) (与 Hive 元存储兼容) 管理表，使用 AWS Glue 爬虫自动发现和注册表，以及通过数据质量功能评估 Iceberg 表中的数据质量。AWS Glue AWS Glue Data Catalog 还支持收集列统计信息、计算和更新 Iceberg 表中每列的不同值 (NDV) 的数量，以及自动表优化 (压缩、快照保留、孤立文件删除)。AWS Glue 还支持将第三方应用程序列表中的零 ETL 集成到 Iceberg AWS 服务中。
- [Amazon Data Firehose](#) 是一项完全托管的服务，用于向亚马逊 S3、亚马逊 Redshift、亚马逊服务、亚马逊无服务器、Splunk、OpenSearch Apache Iceberg 表以及支持的第三方服务提供商拥有的任何自定义 HTTP 或 HTTPS 终端节点 (包括 Datadog、Dynatrace、MongoDB、New Relic LogicMonitor、Coralogix 和 Elastic) 等目的地提供实时流数据。OpenSearch 使用 Firehose，您无需编写应用程序或管理资源。您可以配置数据生成工具向 Firehose 发送数据，然后 Firehose 会将数据自动传输到您指定的目标。还可以配置 Firehose 在传输数据之前转换数据。
- [适用于 Apache Flink 的亚马逊托管服务 Flink](#) 是一项完全托管的亚马逊服务，允许您使用 Apache Flink 应用程序来处理流数据。它支持读取和写入 Iceberg 表，并支持实时数据处理和分析。
- [Amazon SageMaker AI](#) 支持使用 Iceberg 格式在亚马逊 SageMaker AI 功能商店中存储功能集。
- [AWS Lake Formation](#) 提供粗略而精细的访问控制权限来访问数据，包括 Athena 或 Amazon Redshift 使用的 Iceberg 表。要详细了解对 Iceberg 表的权限支持，请参阅 [Lake Formation 文档](#)。

AWS 提供多种支持 Iceberg 的服务，但涵盖所有这些服务超出了本指南的范围。以下各节介绍了 Amazon EMR 和 Athena SQL 上的 Spark (批量和 AWS Glue 结构化直播)。以下部分简要介绍了 Athena SQL 中对 Iceberg 的支持。

开始使用 Amazon Athena SQL 中的 Iceberg 表

亚马逊 Athena 为 Iceberg 提供内置支持。除了设置 Athena 文档[入门](#)部分中详述的服务先决条件外，您无需任何其他步骤或配置即可使用 Iceberg。本节简要介绍如何在 Athena 中创建表格。有关更多信息，请参阅本指南后[面的使用 Athena SQL 处理 Iceberg 表](#)。

您可以使用不同的引擎在上 AWS 创建 Iceberg 表。这些表格可以无缝协作 AWS 服务。要使用 Athena SQL 创建你的第一个 Iceberg 表，你可以使用以下样板代码。

```
CREATE TABLE <table_name> (  
    col_1 string,  
    col_2 string,  
    col_3 bigint,  
    col_ts timestamp)  
PARTITIONED BY (col_1, <<<partition_transform>>>(col_ts))  
LOCATION 's3://<bucket>/<folder>/<table_name>/'  
TBLPROPERTIES (  
    'table_type' = 'ICEBERG'  
)
```

以下各节提供了在 Athena 中创建分区和未分区的 Iceberg 表的示例。有关更多信息，请参阅 At [hena](#) 文档中详细介绍的 Iceberg 语法。

创建未分区的表

以下示例语句自定义样板 SQL 代码，以在 Athena 中创建未分区的 Iceberg 表。您可以将此语句添加到 [Athena](#) 控制台的查询编辑器中以创建表。

```
CREATE TABLE athena_iceberg_table (  
    color string,  
    date string,  
    name string,  
    price bigint,  
    product string,  
    ts timestamp)  
LOCATION 's3://DOC_EXAMPLE_BUCKET/ice_warehouse/iceberg_db/athena_iceberg_table/'  
TBLPROPERTIES (  
    'table_type' = 'ICEBERG'  
)
```

有关使用查询编辑器的分步说明，请参阅 Athena 文档中的[入门](#)。

创建分区表

以下语句使用 Iceberg 的[隐藏](#)分区概念根据日期创建分区表。它使用 `day()` 转换从时间戳列中派生出每日分区（使用 `dd-mm-yyyy` 格式）。Iceberg 不会将此值存储为数据集中的新列。相反，该值是在您写入或查询数据时即时派生的。

```
CREATE TABLE athena_iceberg_table_partitioned (  
    color string,  
    date string,  
    name string,  
    price bigint,  
    product string,  
    ts timestamp)  
PARTITIONED BY (day(ts))  
LOCATION 's3://DOC_EXAMPLE_BUCKET/ice_warehouse/iceberg_db/athena_iceberg_table/'  
TBLPROPERTIES (  
    'table_type' = 'ICEBERG'  
)
```

使用单个 CTAS 语句创建表并加载数据

在前几节的分区和未分区示例中，Iceberg 表创建为空表。您可以使用 `INSERT` 或 `MERGE` 语句将数据加载到表中。或者，您可以使用 `CREATE TABLE AS SELECT` (CTAS) 语句在单个步骤中创建数据并将其加载到 Iceberg 表中。

CTAS 是 Athena 中创建表并在单个语句中加载数据的最佳方式。以下示例说明如何使用 CTAS 从 Athena 中的现有表 (`iceberg_ctas_table`) 创建 Iceberg Hive/Parquet 表 (`hive_table`)。

```
CREATE TABLE iceberg_ctas_table WITH (  
    table_type = 'ICEBERG',  
    is_external = false,  
    location = 's3://DOC_EXAMPLE_BUCKET/ice_warehouse/iceberg_db/iceberg_ctas_table/'  
) AS  
SELECT * FROM "iceberg_db"."hive_table" limit 20  
---  
SELECT * FROM "iceberg_db"."iceberg_ctas_table" limit 20
```

要了解有关 CTAS 的更多信息，请参阅 [Athena CTAS](#) 文档。

插入、更新和删除数据

Athena 支持使用 UPDATE、MERGE INTO、和 M 语句将数据写入 Iceberg 表 INSERT INTO 的不同方式。DELETE FROM

Note

Athena SQL 目前不支持写入时复制方法。UPDATE MERGE INTO、和 DELETE FROM 操作始终使用读取时合并方法进行位置删除，而不管指定的表属性如何。如果您设置了诸如 write.update.mode、write.merge.mode、和之类的表属性 write.delete.mode 以使用写时复制，则查询不会失败，但是 Athena 会忽略这些属性并继续使用读时合并。

以下语句用于 INSERT INTO 向 Iceberg 表添加数据：

```
INSERT INTO "iceberg_db"."ice_table" VALUES (  
    'red', '222022-07-19T03:47:29', 'PersonNew', 178, 'Tuna', now()  
)  
  
SELECT * FROM "iceberg_db"."ice_table"  
where color = 'red' limit 10;
```

示例输出：

Results (1)								Copy	Download results
Search rows								< 1 > ⚙	
#	color	date	name	price	product	ts			
1	red	222022-07-19T03:47:29	PersonNew	178	Tuna	2023-10-11 11:35:01.298000 UTC			

有关更多信息，请参阅 [Athena 文档](#)。

查询冰山表

您可以使用 Athena SQL 对您的 Iceberg 表运行常规 SQL 查询，如前面的示例所示。

除了通常的查询外，Athena 还支持对 Iceberg 表进行时空旅行查询。如前所述，您可以通过更新或删除 Iceberg 表来更改现有记录，因此可以方便地使用时空旅行查询根据时间戳或快照 ID 回顾表的旧版本。

例如，以下语句更新了的颜色值Person5，然后显示自 2023 年 1 月 4 日起的较早值：

```
UPDATE ice_table SET color='new_color' WHERE name='Person5'

SELECT * FROM "iceberg_db"."ice_table" FOR TIMESTAMP AS OF TIMESTAMP '2023-01-04
12:00:00 UTC'
```

示例输出：

Results (15)

CopyDownload results

Search rows

#	color	date	name	price	product	ts
1	cyan	222022-07-19T03:47:29	Person5	353	Keyboard	2023-01-03 10:15:52.268000 UTC
2	lime	222022-07-19T03:47:29	Person1	833	Towels	2023-01-03 10:15:52.268000 UTC
3	turquoise	222022-07-19T03:47:29	Person1	1319	Shirt	2023-01-03 10:15:52.268000 UTC
4	blue	222022-07-19T03:47:29	Person3	163	Sausages	2023-01-03 10:15:52.268000 UTC

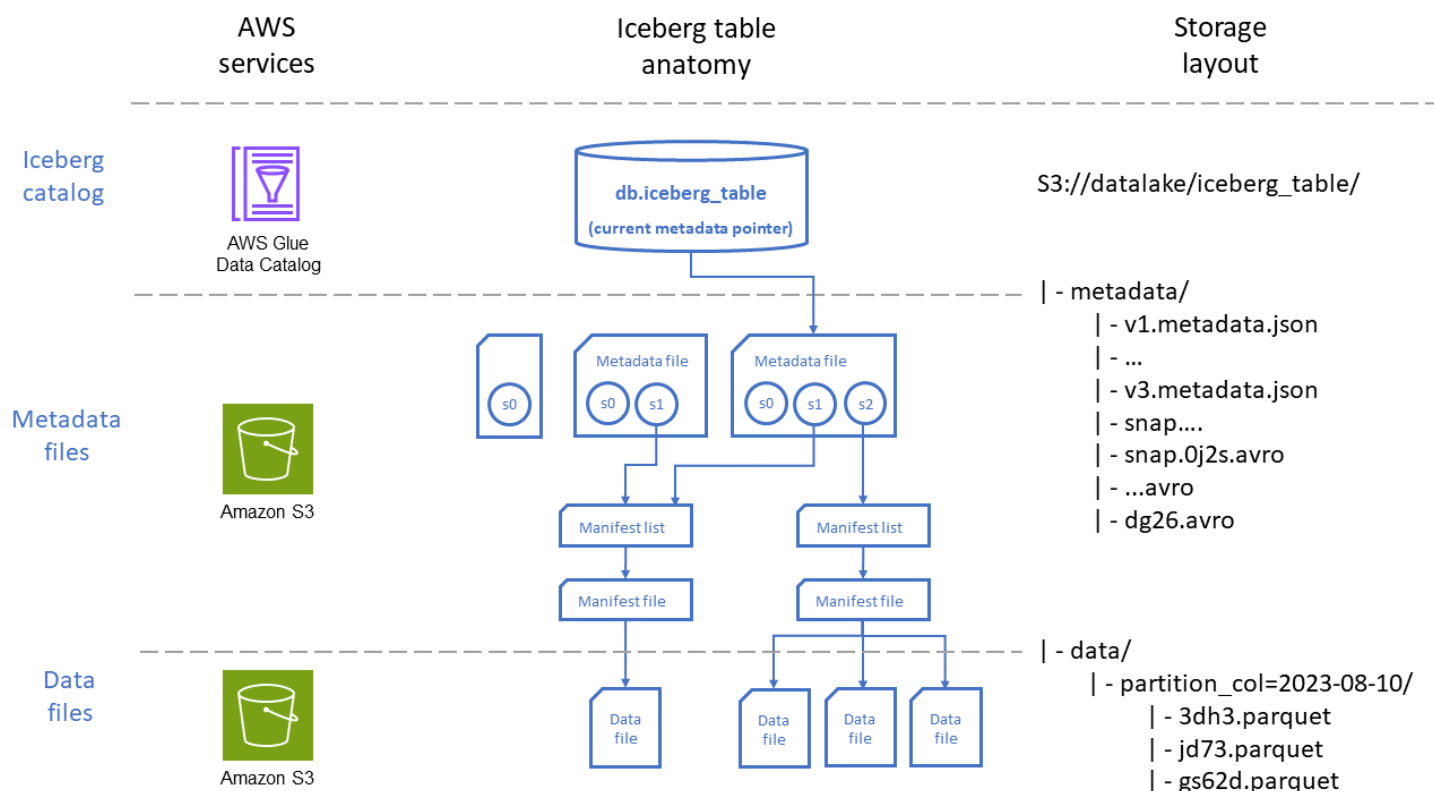
有关时空旅行查询的语法和其他示例，请参阅 [Athena 文档](#)。

冰山桌解剖图

现在，我们已经介绍了使用 Iceberg 桌子的基本步骤，让我们更深入地了解冰山桌子错综复杂的细节和设计。

为了启用本指南[前面描述](#)的功能，Iceberg 设计了数据和元数据文件的分层结构。这些层可以智能地管理元数据，以优化查询计划和执行。

下图通过两个角度描绘了 Iceberg 表的组织结构：AWS 服务 用于存储表的视角和文件在 Amazon S3 中的位置。



如图所示，冰山表由三个主要层组成：

- **Iceberg 目录：** AWS Glue Data Catalog 与 Iceberg 原生集成，对于大多数用例来说，它是运行的工作负载的最佳选择。AWS 与 Iceberg 表（例如 Athena）交互的服务（例如 Athena）使用目录来查找表的当前快照版本，以读取或写入数据。
- **元数据层：** 元数据文件，即清单文件和清单列表文件，用于跟踪表架构、分区策略和数据文件位置等信息，以及列级统计信息，例如存储在每个数据文件中的记录的最小和最大范围。这些元数据文件存储在 Amazon S3 中的表路径中。
 - 清单文件包含每个数据文件的记录，包括其位置、格式、大小、校验和和其他相关信息。
 - 清单列表提供了清单文件的索引。随着表中清单文件数量的增加，将这些信息分成较小的子部分有助于减少查询需要扫描的清单文件数量。
 - 元数据文件包含有关整个 Iceberg 表的信息，包括清单列表、架构、分区元数据、快照文件和其他用于管理表元数据的文件。
- **数据层：** 该层包含包含查询将针对的数据记录的文件。[这些文件可以以不同的格式存储，包括 Apache Parquet、Apache Avro 和 Apache ORC。](#)
 - 数据文件包含表的数据记录。
 - 删除文件在 Iceberg 表中对行级删除和更新操作进行编码。Iceberg 有两种类型的删除文件，如 [Iceberg](#) 文档中所述。这些文件是通过使用读取时合并模式通过操作创建的。

在 Amazon EMR 中使用 Iceberg

Amazon EMR 使用 Apache Spark、Apache Hive、Flink 和 Trino 等开源框架，在云端提供 PB 级的数据处理、交互式分析和机器学习。

Note

本指南使用 Apache Spark 作为示例。

亚马逊 EMR 支持多种部署选项：开启亚马逊 EMR、EKS 上的亚马逊 EMR、EC2 亚马逊 EMR Serverless 和开启亚马逊 EMR。AWS Outposts 要为您的工作负载选择部署选项，请参阅 [Amazon EMR 常见问题解答](#)。

版本和功能兼容性

亚马逊 EMR 版本 6.5.0 及更高版本原生支持 Apache Iceberg。有关每个亚马逊 EMR 版本支持的 Iceberg 版本列表，[请参阅亚马逊 EMR 文档中的 Iceberg 版本](#)历史记录。另请查看在 [Iceberg 中使用集群](#)下的章节，了解不同框架上的 Amazon EMR 支持哪些 Iceberg 功能。

我们建议您使用最新的 Amazon EMR 版本，以便从支持的最新版本的 Iceberg 中受益。本节中的代码示例和配置假设您使用的是 Amazon EMR 版本 emr-7.8.0。

使用 Iceberg 创建 Amazon EMR 集群

要在安装了 Iceberg 的情况下在亚马逊上创建亚马逊 EC2 EMR 集群，请按照亚马逊 [EMR](#) 文档中的说明进行操作。

具体而言，您的集群应按以下分类进行配置：

```
[[
  "Classification": "iceberg-defaults",
  "Properties": {
    "iceberg.enabled": "true"
  }
]]
```

从亚马逊 EMR 6.6.0 开始，您还可以选择在 EKS 上使用 Amazon EMR Serverless 或 Amazon EMR 作为 Iceberg 工作负载的部署选项。

在 Amazon EMR 中开发 Iceberg 应用程序

要为 Iceberg 应用程序开发 Spark 代码，您可以使用[亚马逊 EMR Studio](#)，这是一个基于 Web 的集成开发环境 (IDE)，适用于在亚马逊 EMR 集群上运行的完全托管 Jupyter 笔记本电脑。

使用亚马逊 EMR Studio 笔记本电脑

您可以在 Amazon EMR Studio Workspace 笔记本电脑中以交互方式开发 Spark 应用程序，并将这些笔记本电脑连接到集群上的 Amazon EMR EC2 或 EKS 托管终端节点上的 Amazon EMR。[有关在 EKS 上为亚马逊 EMR 设置 EMR 和 Amazon EMR 设置 EMR 的说明，请参阅 AWS 服务 文档。EC2](#)

要在 EMR Studio 中使用 Iceberg，请按照以下步骤操作：

1. 按照[使用安装了 Iceberg 的集群中的说明，启动启用了 Iceberg 的 Amazon EMR 集群](#)。
2. 设置 EMR 工作室。有关说明，请参阅[设置 Amazon EMR Studio](#)。
3. 打开 EMR Studio Workspace 笔记本并运行以下代码作为笔记本中的第一个单元来配置 Spark 会话以使用 Iceberg：

```
%%configure -f
{
  "conf": {
    "spark.sql.catalog.<catalog_name>": "org.apache.iceberg.spark.SparkCatalog",
    "spark.sql.catalog.<catalog_name>.warehouse": "s3://YOUR-BUCKET-NAME/YOUR-
FOLDER-NAME/",
    "spark.sql.catalog.<catalog_name>.type": "glue",
    "spark.sql.extensions":
    "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
  }
}
```

其中：

- <catalog_name>是你的 Iceberg Spark 会话目录名称。将其替换为您选择的名称，并记得更改与该目录关联的所有配置中的引用。在您的代码中，您可以使用完全限定的表名（包括 Spark 会话目录名称）来引用 Iceberg 表，如下所示：

```
<catalog_name>.<database_name>.<table_name>
```

或者，您可以将默认目录更改为通过设置 `spark.sql.defaultCatalog` 目录名称来定义的 Iceberg 目录。第二种方法使您可以引用不带目录前缀的表，这可以简化查询。

- `<catalog_name>.warehouse` 指向您要存储数据和元数据的 Amazon S3 路径。
 - 要使目录成为 AWS Glue Data Catalog，请将设置 `spark.sql.catalog.<catalog_name>.type` 为 `glue`。对于任何自定义目录实现，都需要使用此密钥来指向实现类。本指南后面的“[一般最佳实践](#)”部分描述了 Iceberg 支持的不同目录。
4. 现在，你可以像开发任何其他 Spark 应用程序一样，开始在笔记本中交互式开发 Iceberg 的 Spark 应用程序。

有关使用亚马逊 EMR Studio 为 Apache Iceberg 配置 Spark 的更多信息，请参阅博客文章“在亚马逊 EMR 上[使用 Apache Iceberg 构建高性能、符合 ACID 且不断演变的数据湖](#)”。

在 Amazon EMR 中运行 Iceberg 职位

为你的 Iceberg 工作负载开发 Spark 应用程序代码后，你可以在任何支持 Iceberg 的 Amazon EMR 部署选项上运行它（参见 [A amazon EM R 常见问题解答](#)）。

与其他 Spark 任务一样，您可以通过添加步骤或以交互方式向主节点提交 Spark 任务来向 EC2 集群上的 Amazon EMR 提交工作。要运行 Spark 作业，请参阅以下 Amazon EMR 文档页面：

- 有关向集群上的 Amazon EMR 提交工作的不同选项的概述以及每个选项的详细说明，请参阅[向 EC2 集群提交工作](#)。
- 有关 EKS 上的 Amazon EMR，请参阅使用[运行 Spark 作业](#)。StartJobRun
- [有关 EMR Serverless 的信息，请参阅运行作业](#)。

以下各节提供了每个 Amazon EMR 部署选项的示例。

亚马逊 EMR 开启 EC2

您可以使用以下步骤提交 Iceberg Spark 作业：

1. 在您的工作站上创建包含以下内容的文件 `emr_step_iceberg.json`：

```
[{
  "Name": "iceberg-test-job",
  "Type": "spark",
  "ActionOnFailure": "CONTINUE",
```

```

    "Args": [
        "--deploy-mode",
        "client",
        "--conf",

        "spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
        "--conf",
        "spark.sql.catalog.<catalog_name>=org.apache.iceberg.spark.SparkCatalog",
        "--conf",
        "spark.sql.catalog.<catalog_name>.type=glue",
        "--conf",
        "spark.sql.catalog.<catalog_name>.warehouse=s3://YOUR-BUCKET-NAME/YOUR-  

FOLDER-NAME/",
        "s3://YOUR-BUCKET-NAME/code/iceberg-job.py"
    ]
}

```

2. 通过自定义以粗体突出显示的 Iceberg 配置选项，修改特定 Spark 作业的配置文件。
3. 使用 AWS Command Line Interface (AWS CLI) 提交步骤。在 `emr_step_iceberg.json` 文件所在的目录中运行该命令。

```
aws emr add-steps --cluster-id <cluster_id> --steps file://emr_step_iceberg.json
```

Amazon EMR Serverless

要向 EMR Serverless 提交 Iceberg Spark 作业，请使用以下命令：AWS CLI

1. 在您的工作站上创建包含以下内容的文件 `emr_serverless_iceberg.json`：

```

{
    "applicationId": "<APPLICATION_ID>",
    "executionRoleArn": "<ROLE_ARN>",
    "name": "iceberg-test-job",
    "jobDriver": {
        "sparkSubmit": {
            "entryPoint": "s3://YOUR-BUCKET-NAME/code/iceberg-job.py",
            "entryPointArguments": []
        }
    },
    "configurationOverrides": {
        "applicationConfiguration": [{

```

```

        "classification": "spark-defaults",
        "properties": {
            "spark.sql.extensions":
"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
            "spark.sql.catalog.<catalog_name>":
"org.apache.iceberg.spark.SparkCatalog",
            "spark.sql.catalog.<catalog_name>.type": "glue",
            "spark.sql.catalog.<catalog_name>.warehouse": "s3://YOUR-BUCKET-NAME/
YOUR-FOLDER-NAME/",
            "spark.jars": "/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar",

            "spark.hadoop.hive.metastore.client.factory.class": "com.amazonaws.glue.catalog.metastore.AWS
        }
    }],
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://YOUR-BUCKET-NAME/emr-serverless/logs/"
        }
    }
}

```

2. 通过自定义以粗体突出显示的 Iceberg 配置选项，修改特定 Spark 作业的配置文件。
3. 使用提交作业 AWS CLI。在 `emr_serverless_iceberg.json` 文件所在的目录中运行命令：

```
aws emr-serverless start-job-run --cli-input-json file://emr_serverless_iceberg.json
```

要使用 EMR Studio 控制台向 EMR Serverless 提交 Iceberg Spark 作业，请执行以下操作：

1. 按照 [EMR 无服务器](#) 文档中的说明进行操作。
2. 要配置 Job，请使用为提供的 Spark 的 Iceberg 配置，AWS CLI 并自定义 Iceberg 的突出显示字段。有关详细说明，请参阅 Amazon EMR [文档中的将 Apache Iceberg 与 EMR Serverless 配合使用](#)。

Amazon EMR on EKS

要在 EKS 上向 Amazon EMR 提交 Iceberg Spark 作业，请使用以下命令：AWS CLI

1. 在您的工作站上创建包含以下内容的文件 `emr_eks_iceberg.json`：

```

{
  "name": "iceberg-test-job",
  "virtualClusterId": "<VIRTUAL_CLUSTER_ID>",
  "executionRoleArn": "<ROLE_ARN>",
  "releaseLabel": "emr-6.9.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://YOUR-BUCKET-NAME/code/iceberg-job.py",
      "entryPointArguments": [],
      "sparkSubmitParameters": "--jars local:///usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [{
      "classification": "spark-defaults",
      "properties": {
        "spark.sql.extensions":
"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
        "spark.sql.catalog.<catalog_name>":
"org.apache.iceberg.spark.SparkCatalog",
        "spark.sql.catalog.<catalog_name>.type": "glue",
        "spark.sql.catalog.<catalog_name>.warehouse": "s3://YOUR-BUCKET-NAME/YOUR-FOLDER-NAME/",
        "spark.hadoop.hive.metastore.client.factory.class":
"com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory"
      }
    }],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "s3MonitoringConfiguration": {
        "logUri": "s3://YOUR-BUCKET-NAME/emr-serverless/logs/"
      }
    }
  }
}

```

2. 通过自定义以粗体突出显示的 Iceberg 配置选项，修改 Spark 作业的配置文件。
3. 使用提交作业 AWS CLI。在 `emr_eks_iceberg.json` 文件所在的目录中运行以下命令：

```
aws emr-containers start-job-run --cli-input-json file://emr_eks_iceberg.json
```

有关详细说明，请参阅 EKS [上的亚马逊 EMR 文档中的 EKS 上将 Apache Iceberg 与 Amazon EMR 搭配使用](#)。

亚马逊 EMR 的最佳实践

本节提供了在 Amazon EMR 中调整 Spark 作业以优化向 Iceberg 表读取和写入数据的一般指南。有关 Iceberg 特定的最佳实践，请参阅本指南后面的“[最佳实践](#)”部分。

- 使用最新版本的亚马逊 EMR — 亚马逊 EMR 通过亚马逊 EMR Spark 运行时提供开箱即用的 Spark 优化。AWS 每个新版本都会提高 Spark 运行时引擎的性能。
- 为您的 Spark 工作负载确定最佳基础架构 — Spark 工作负载可能需要不同类型的硬件来满足不同的作业特征，以确保最佳性能。Amazon EMR [支持多种实例类型](#)（例如计算优化、内存优化、通用型和存储优化），以满足所有类型的处理要求。在载入新工作负载时，我们建议您使用常规实例类型（例如 M5 或 m6g）进行基准测试。监控 CloudWatch 来自 Ganglia 和 Amazon 的操作系统 (OS) 和 YARN 指标，确定峰值负载下的系统瓶颈（CPU、内存、存储和 I/O），然后选择合适的硬件。
- 调整 `spark.sql.shuffle.partitions` — 将该 `spark.sql.shuffle.partitions` 属性设置为集群中的虚拟内核 (vCore) 总数或该值的倍数（通常为 vCore 总数的 1 到 2 倍）。当您使用哈希和范围分区作为写入分配模式时，此设置会影响 Spark 的并行度。它在写入之前请求洗牌以整理数据，从而确保分区对齐。
- 启用托管扩展-对于几乎所有用例，我们建议您启用托管扩展和动态分配。但是，如果您的工作负载具有可预测的模式，我们建议您禁用自动扩展和动态分配。启用托管扩展后，我们建议您使用竞价型实例来降低成本。将竞价型实例用于任务节点，而不是核心或主节点。当您使用竞价型实例时，请使用每个队列具有多种实例类型的实例队列，以确保竞价型可用性。
- 尽可能使用广播联接 — 广播（地图侧）联接是最佳联接，前提是您的一个表足够小，可以容纳最小节点的内存（按顺序排列 MBs），并且您正在执行 `equi (=)` 联接。支持除完全外部联接之外的所有联接类型。广播联接将较小的表作为哈希表广播到内存中的所有工作节点。广播小桌子后，您无法对其进行更改。由于哈希表位于本地 Java 虚拟机 (JVM) 中，因此使用哈希联接可以根据联接条件轻松地将其与大型表合并。广播联接可提供高性能，因为随机播放开销最小。
- 调整垃圾收集器 — 如果垃圾收集 (GC) 周期很慢，可以考虑从默认的并行垃圾收集器切换到 G1GC 以获得更好的性能。要优化 GC 性能，可以微调 GC 参数。要跟踪 GC 性能，您可以使用 Spark 用户界面对其进行监控。理想情况下，GC 时间应小于或等于任务总运行时间的 1%。

在 Iceberg 中工作 AWS Glue

[AWS Glue](#) 是一项无服务器数据集成服务，可以更轻松地发现、准备、移动和集成来自多个来源的数据，用于分析、机器学习 (ML) 和应用程序开发。的核心功能之一 AWS Glue 是它能够以简单且经济实惠的方式执行提取、转换和加载 (ETL) 操作。这有助于对数据进行分类、清理、丰富数据，并在各种数据存储和数据流之间可靠地移动数据。

[AWS Glue 作业](#) 使用 [Apache Spark 或 Pyth on](#) 运行时封装定义转换逻辑的脚本。AWS Glue 作业既可以在批处理模式下运行，也可以在流式传输模式下运行。

在中创建 Iceberg 作业时 AWS Glue，根据的版本 AWS Glue，您可以使用原生 Iceberg 集成或自定义 Iceberg 版本将 Iceberg 依赖项附加到该作业。

使用原生 Iceberg 集成

AWS Glue 3.0、4.0 和 5.0 版本原生支持事务性数据湖格式，例如 Spark 中的 Apache Iceberg、Apache Hudi 和 Linux Foundation Delta Lake。AWS Glue 此集成功能简化了开始在中使用这些框架所需的配置步骤 AWS Glue。

要为您的 AWS Glue 作业启用 Iceberg 支持，请设置作业：为您的 AWS Glue 作业选择任务详细信息选项卡，滚动到“高级属性”下的“作业参数”，然后将密钥设置为，其值设置为iceberg。--datalake-formats

如果您使用笔记本创作作业，则可以使用以下%%configure魔术在第一个笔记本单元格中配置参数：

```
%%configure
{
  "--conf" : <job-specific Spark configuration discussed later>,
  "--datalake-formats" : "iceberg"
}
```

--datalake-formats中的iceberg配置对 AWS Glue 应于基于您的版本的特定 Iceberg AWS Glue 版本：

AWS Glue 版本	默认 Iceberg 版本
5.0	1.7.1

AWS Glue 版本	默认 Iceberg 版本
4.0	1.0.0
3.0	0.13.1

使用自定义 Iceberg 版本

在某些情况下，您可能需要保留对任务的 Iceberg 版本的控制权，并按照自己的节奏对其进行升级。例如，升级到更高版本可以解锁对新功能和性能增强的访问权限。要将特定的 Iceberg 版本与一起使用 AWS Glue，您可以提供自己的 JAR 文件。

在实现自定义 Iceberg 版本之前，请查看 AWS Glue 文档的[AWS Glue 版本](#)部分，以验证与您的 AWS Glue 环境的兼容性。例如，AWS Glue 5.0 需要与 Spark 3.5.4 兼容。

例如，要运行使用 Iceberg 版本 1.9.1 的 AWS Glue 作业，请按照以下步骤操作：

1. 获取所需的 JAR 文件并将其上传到亚马逊 S3：
 - a. [从 Apache Maven 存储库中下载 iceberg-spark-runtime-3.5_2.12-1.9.1.jar 和-1.9.1.jar。iceberg-aws-bundle](#)
 - b. 将这些文件上传到您指定的 S3 存储桶位置（例如 `s3://your-bucket-name/jars/`）。
2. 按如下方式为 AWS Glue 作业设置作业参数：
 - a. 在 `--extra-jars` 参数中指定两个 JAR 文件的完整 S3 路径，用逗号分隔它们（例如，`s3://your-bucket-name/jars/iceberg-spark-runtime-3.5_2.12-1.9.1.jar,s3://your-bucket-name/jars/iceberg-aws-bundle-1.9.1.jar`）。
 - b. 请勿包含 `iceberg` 作为 `--datalake-formats` 参数的值。
 - c. 如果您使用 AWS Glue 5.0，则必须将 `--user-jars-first` 参数设置为 `true`。

Iceberg 的 Spark 配置 AWS Glue

本节讨论为 Iceberg 数据集创作 AWS Glue ETL 作业所需的 Spark 配置。您可以使用 `--conf` Spark 密钥以及以逗号分隔的所有 Spark 配置键和值列表来设置这些配置。你可以使用笔记本中的 `%configure` 魔法，也可以使用 AWS Glue Studio 控制台的 Job 参数部分。

```
%glue_version 5.0
```

```
%%configure
{
  "--conf" : "spark.sql.extensions=org.apache.iceberg.spark.extensions...",
  "--datalake-formats" : "iceberg"
}
```

使用以下属性配置 Spark 会话：

- `<catalog_name>` 是你的 Iceberg Spark 会话目录名称的名称。将其替换为您选择的名称，并记得更改与该目录关联的所有配置中的引用。在您的代码中，您可以使用完全限定的表名（包括 Spark 会话目录名称）来引用 Iceberg 表，如下所示：

`<catalog_name>.<database_name>.<table_name>`

或者，您可以将默认目录更改为通过设置 `spark.sql.defaultCatalog` 目录名称来定义的 Iceberg 目录。您可以使用第二种方法来引用不带目录前缀的表，这样可以简化查询。

- `<catalog_name>.<warehouse>` 指向您要存储数据和元数据的 Amazon S3 路径。
- 要使目录成为 AWS Glue Data Catalog，请将设置 `spark.sql.catalog.<catalog_name>.type` 为 `glue`。对于任何自定义目录实现，都需要使用此密钥来指向实现类。有关 Iceberg 支持的目录，请参阅本指南后面的“[一般最佳实践](#)”部分。

例如，如果您有一个名为的目录 `glue_iceberg`，则可以使用多个 `--conf` 密钥配置作业，如下所示：

```
%%configure
{
  "--datalake-formats" : "iceberg",
  "--conf" :
  "spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
  --conf spark.sql.catalog.glue_iceberg=org.apache.iceberg.spark.SparkCatalog --
  conf spark.sql.catalog.glue_iceberg.warehouse=s3://<your-warehouse-dir>/ --conf
  spark.sql.catalog.glue_iceberg.type=glue"
}
```

或者，您可以使用代码将上述配置添加到 Spark 脚本中，如下所示：

```
spark = SparkSession.builder\

  .config("spark.sql.extensions","org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions")
  .config("spark.sql.catalog.glue_iceberg",
  "org.apache.iceberg.spark.SparkCatalog")\
```

```
.config("spark.sql.catalog.glue_iceberg.warehouse", "s3://<your-warehouse-dir>/")\n.config("spark.sql.catalog.glue_iceberg.type", "glue") \n.getOrCreate()
```

AWS Glue 工作最佳实践

本节提供了调整 Spark 作业 AWS Glue 以优化 Iceberg 表中数据的读取和写入的一般指南。有关 Iceberg 特定的最佳实践，请参阅本指南后面的“[最佳实践](#)”部分。

- 尽可能使用最新版本 AWS Glue 并进行升级 — 新版本 AWS Glue 提供性能改进、启动时间缩短和新功能。它们还支持最新的 Iceberg 版本可能需要的较新 Spark 版本。有关可用 AWS Glue 版本及其支持的 Spark 版本的列表，请参阅[AWS Glue 文档](#)。
- 优化 AWS Glue 作业内存-按照 AWS 博客文章“[优化内存管理](#)”中的建议进行操作 AWS Glue。
- 使用 AWS Glue Auto Scaling — 启用 Auto Scaling 时，AWS Glue 会根据您的工作负载自动动态调整 AWS Glue 工作人员的数量。这有助于降低高峰负荷期间 AWS Glue 的工作成本，因为当工作量较小且员工处于闲置状态时，可以 AWS Glue 缩减员工人数。要使用 AWS Glue Auto Scaling，您需要指定 AWS Glue 任务可以扩展到的最大工作人员数量。有关更多信息，请参阅 AWS Glue 文档 AWS Glue 中的[使用 auto scaling](#)。
- 使用所需的 Iceberg 版本 — Iceberg AWS Glue 的原生集成最适合入门 Iceberg。但是，对于生产工作负载，我们建议您添加库依赖项（如[本指南前面](#)所述），以完全控制 Iceberg 版本。这种方法可以帮助您在 AWS Glue 工作中受益于最新的 Iceberg 功能和性能改进。
- 启用 Spark 用户界面进行监控和调试 — 您还可以使用[中的 Spark 用户界面 AWS Glue](#)来检查 Iceberg 作业，方法是在有向无环图 (DAG) 中可视化 Spark 作业的不同阶段并详细监控作业。Spark UI 提供了一种有效的方法来进行故障排除和优化 Iceberg 作业。例如，您可以识别具有大量洗牌或磁盘溢出的瓶颈阶段，以确定调整机会。有关更多信息，请参阅 AWS Glue 文档中的[使用 Apache Spark 网页用户界面监控作业](#)。

使用 Apache Spark 处理冰山表

本节概述了如何使用 Apache Spark 与 Iceberg 表进行交互。示例是可以在 Amazon EMR 上运行的样板代码，或者。AWS Glue

注意：与 Iceberg 表交互的主要接口是 SQL，因此大多数示例都将 Spark SQL 与 DataFrames API 结合使用。

创建和写作 Iceberg 表

你可以使用 Spark SQL 和 Spark DataFrames 来创建数据并将其添加到 Iceberg 表中。

使用 Spark SQL

要编写 Iceberg 数据集，请使用标准的 Spark SQL 语句，例如 CREATE TABLE 和 INSERT INTO

未分区的表

以下是使用 Spark SQL 创建未分区的 Iceberg 表的示例：

```
spark.sql(f"""
    CREATE TABLE IF NOT EXISTS {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_nopartitions (
        c_customer_sk          int,
        c_customer_id          string,
        c_first_name            string,
        c_last_name             string,
        c_birth_country         string,
        c_email_address         string)
    USING iceberg
    OPTIONS ('format-version'='2')
""")
```

要将数据插入到未分区的表中，请使用标准 INSERT INTO 语句：

```
spark.sql(f"""
INSERT INTO {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_nopartitions
SELECT c_customer_sk, c_customer_id, c_first_name, c_last_name, c_birth_country,
       c_email_address
FROM another_table
```

```
""")
```

分区表

以下是使用 Spark SQL 创建分区 Iceberg 表的示例：

```
spark.sql(f"""
    CREATE TABLE IF NOT EXISTS {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_withpartitions (
        c_customer_sk          int,
        c_customer_id          string,
        c_first_name           string,
        c_last_name            string,
        c_birth_country        string,
        c_email_address        string)
    USING iceberg
    PARTITIONED BY (c_birth_country)
    OPTIONS ('format-version'='2')
""")
```

要使用 Spark SQL 将数据插入到分区 Iceberg 表中，请使用标准 INSERT INTO 语句：

```
spark.sql(f"""
INSERT INTO {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_withpartitions
SELECT c_customer_sk, c_customer_id, c_first_name, c_last_name, c_birth_country,
       c_email_address
FROM another_table
""")
```

Note

从 Iceberg 1.5.0 开始，向分区表中插入数据时，hash 写入分配模式为默认模式。有关更多信息，请参阅 Iceberg 文档中的[编写分发模式](#)。

使用 DataFrames API

要编写 Iceberg 数据集，你可以使用 DataFrameWriterV2 API。

要创建 Iceberg 表并向其写入数据，请使用 `df.writeTo(t)` 函数。如果该表存在，请使用该 `.append()` 函数。如果不是，请使用 `.create()`。以下示例使用 `.createOrReplace()`，其变体 `.create()` 等效于 `CREATE OR REPLACE TABLE AS SELECT`。

未分区的表

要使用 API 创建和填充未分区的 Iceberg 表，请执行以下操作：DataFrameWriterV2

```
input_data.writeTo(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_nopartitions") \
    .tableProperty("format-version", "2") \
    .createOrReplace()
```

要使用 API 将数据插入到现有的未分区 Iceberg 表中，请执行以下操作：DataFrameWriterV2

```
input_data.writeTo(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_nopartitions") \
    .append()
```

分区表

要使用 API 创建和填充分区 Iceberg 表，请执行以下操作：DataFrameWriterV2

```
input_data.writeTo(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_withpartitions") \
    .tableProperty("format-version", "2") \
    .partitionedBy("c_birth_country") \
    .createOrReplace()
```

要使用 API 将数据插入分区的 Iceberg 表，请执行DataFrameWriterV2以下操作：

```
input_data.writeTo(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_withpartitions") \
    .append()
```

更新 Iceberg 表中的数据

以下示例说明如何更新 Iceberg 表中的数据。此示例修改c_customer_sk列中包含偶数的所有行。

```
spark.sql(f"""
UPDATE {CATALOG_NAME}.{db.name}.{table.name}
SET c_email_address = 'even_row'
WHERE c_customer_sk % 2 == 0
""")
```

此操作使用默认的写入时复制策略，因此它会重写所有受影响的数据文件。

更新 Iceberg 表中的数据

更新数据是指在单个事务中插入新的数据记录并更新现有的数据记录。要将数据插入到 Iceberg 表中，请使用语句。SQL `MERGE INTO`

以下示例将表格{`UPSERT_TABLE_NAME`}的内容放入表中：`{TABLE_NAME}`

```
spark.sql(f"""
    MERGE INTO {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME} t
    USING {UPSERT_TABLE_NAME} s
    ON t.c_customer_id = s.c_customer_id
    WHEN MATCHED THEN UPDATE SET t.c_email_address = s.c_email_address
    WHEN NOT MATCHED THEN INSERT *
    """)
```

- 如果中{`UPSERT_TABLE_NAME`}已经存在{`TABLE_NAME`}具有相同值的客户记录`c_customer_id`，则该{`UPSERT_TABLE_NAME`}记录`c_email_address`值将覆盖现有值（更新操作）。
- 如果中的客户记录在中{`UPSERT_TABLE_NAME`}不存在{`TABLE_NAME`}，则会将该{`UPSERT_TABLE_NAME`}记录添加到{`TABLE_NAME`}（插入操作）。

删除 Iceberg 表中的数据

要从 Iceberg 表中删除数据，请使用`DELETE FROM`表达式并指定与要删除的行匹配的筛选器。

```
spark.sql(f"""
    DELETE FROM {CATALOG_NAME}.{db.name}.{table.name}
    WHERE c_customer_sk % 2 != 0
    """)
```

如果过滤器与整个分区匹配，Iceberg 会执行仅限元数据的删除并将数据文件保留在原处。否则，它只会重写受影响的数据文件。

`delete` 方法获取受`WHERE`子句影响的数据文件，并在不包含已删除记录的情况下创建这些文件的副本。然后，它会创建一个指向新数据文件的新表快照。因此，已删除的记录仍存在于表的旧快照中。例如，如果您检索表的上一个快照，则会看到刚刚删除的数据。有关出于清理目的删除不必要的旧快照以及相关数据文件的信息，请参阅本指南后面的[“使用压缩维护文件”](#)一节。

读取数据

您可以使用 Spark SQL 和 Spark 在 Spark 中读取 Iceberg 表的最新状态。DataFrames

使用 Spark SQL 的示例：

```
spark.sql(f"""
SELECT * FROM {CATALOG_NAME}.{db.name}.{table.name} LIMIT 5
""")
```

使用 DataFrames API 的示例：

```
df = spark.table(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}").limit(5)
```

使用时空旅行

Iceberg 表中的每个写入操作（插入、更新、更新插入、删除）都会创建一个新快照。然后，您可以使用这些快照进行时空旅行，以回到过去，检查表的过去状态。

有关如何使用 snapshot-id 和计时值检索表快照历史记录的信息，请参阅本指南后面的[访问元数据](#)部分。

以下时空旅行查询根据特定内容显示表的状态 snapshot-id。

使用 Spark SQL：

```
spark.sql(f"""
SELECT * FROM {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME} VERSION AS OF {snapshot_id}
""")
```

使用 DataFrames API：

```
df_1st_snapshot_id = spark.read.option("snapshot-id", snapshot_id) \
    .format("iceberg") \
    .load(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}") \
    .limit(5)
```

以下时空旅行查询根据在特定时间戳之前创建的最后一次快照显示表的状态，以毫秒 () 为单位。as-of-timestamp

使用 Spark SQL :

```
spark.sql(f"""
SELECT * FROM dev.{db.name}.{table.name} TIMESTAMP AS OF '{snapshot_ts}'
""")
```

使用 DataFrames API :

```
df_1st_snapshot_ts = spark.read.option("as-of-timestamp", snapshot_ts) \
    .format("iceberg") \
    .load(f"dev.{DB_NAME}.{TABLE_NAME}") \
    .limit(5)
```

使用增量查询

您还可以使用 Iceberg 快照以增量方式读取附加的数据。

注意：目前，此操作支持从append快照中读取数据。它不支持从replace、overwrite或删除之类的操作中获取数据。此外，Spark SQL 语法不支持增量读取操作。

以下示例检索在快照start-snapshot-id (不包括) 和end-snapshot-id (含) 之间附加到 Iceberg 表的所有记录。

```
df_incremental = (spark.read.format("iceberg")
    .option("start-snapshot-id", snapshot_id_start)
    .option("end-snapshot-id", snapshot_id_end)
    .load(f"glue_catalog.{DB_NAME}.{TABLE_NAME}")
)
```

访问元数据

Iceberg 通过 SQL 提供对其元数据的访问。您可以通过查询命名空间来访问任何给定表 (<table_name>) 的元数据<table_name>.<metadata_table>。有关元数据表的完整列表，请参阅 Iceberg 文档中的[检查表](#)。

以下示例说明如何访问 Iceberg 历史元数据表，该表显示了 Iceberg 表的提交 (更改) 历史记录。

使用亚马逊 EMR Studio 笔记本上的 Spark SQL (有%%sql魔法) :

```
Spark.sql(f"""
SELECT * FROM {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}.history LIMIT 5
""")
```

使用 DataFrames API :

```
spark.read.format("iceberg").load("{CATALOG_NAME}.{DB_NAME}.
{TABLE_NAME}.history").show(5,False)
```

示例输出 :

Type:

Table

Pie

Scatter

Line

Area

Bar

made_current_at	snapshot_id	parent_id	is_current_ancestor
2023-01-09 02:50:17.547000+00:00	7501027970051178613	6598755163776233735	True
2023-01-12 05:39:29.567000+00:00	7069175828427777019	7501027970051178613	True
2023-01-12 05:39:58.807000+00:00	5173022175861138222	7069175828427777019	True
2023-01-12 05:40:18.499000+00:00	3703414997660223390	5173022175861138222	True
2023-01-12 05:40:41.827000+00:00	3807904412292252460	3703414997660223390	True

使用 Trino 处理冰山牌桌

本节介绍如何在 Amazon EMR 上使用 [Trino](#) 设置和操作 Iceberg 表。示例是您可以在 EC2 集群上的 Amazon EMR 上运行的样板代码。本节中的代码示例和配置假设您使用的是 Amazon EMR 版本 emr-7.9.0。

EC2 设置上的 Amazon EMR

1. 创建包含以下内容的 `iceberg.properties` 文件。如果语句中未明确指定新表的默认存储格式，则该 `iceberg.file-format=parquet` 设置将确定新 CREATE TABLE 表的默认存储格式。

```
connector.name=iceberg
iceberg.catalog.type=glue
iceberg.file-format=parquet
fs.native-s3.enabled=true
```

2. 将 `iceberg.properties` 文件上传到 S3 存储桶。
3. 创建一个引导操作，该操作从您的 S3 存储桶中复制 `iceberg.properties` 文件并将其作为 Trino 配置文件存储在您将要创建的 Amazon EMR 集群上。请务必将 `<S3-bucket-name>` 使用您的 S3 存储桶名称替换。

```
#!/bin/bash
set -ex
sudo aws s3 cp s3://<S3-bucket-name>/iceberg.properties /etc/trino/conf/catalog/
iceberg.properties
```

4. 创建安装了 Trino 的 Amazon EMR 集群，并将前一个脚本的执行指定为引导操作。以下是创建集群的示例 AWS Command Line Interface (AWS CLI) 命令：

```
aws emr create-cluster --release-label emr-7.9.0 \
--applications Name=Trino \
--region <region> \
--name Trino_Iceberg_Cluster \
--bootstrap-actions '[{"Path":"s3://<S3-bucket-name>/bootstrap.sh","Name":"Add
iceberg.properties"}]' \
--instance-groups
' [{"InstanceGroupType":"MASTER","InstanceCount":1,"InstanceType":"m5.xlarge"},
{"InstanceGroupType":"CORE","InstanceCount":3,"InstanceType":"m5.xlarge"}]' \
--service-role "<IAM-service-role>" \
```

```
--ec2-attributes '{"KeyName":"<key-name>","InstanceProfile":"<EMR-EC2-instance-profile>"}'
```

你在哪里替换：

- <S3-bucket-name>使用您的 S3 存储桶名称
- <region>根据你的具体情况 AWS 区域
- <key-name>用你的 key pair。如果密钥对 (key pair) 不存在，则会将其创建。
- <IAM-service-role>使用遵循[最低权限原则的 Amazon EMR 服务角色](#)。
- <EMR-EC2-instance-profile>使用您的[实例配置文件](#)。

5. 初始化 Amazon EMR 集群后，您可以通过运行以下命令来初始化 Trino 会话：

```
trino-cli
```

6. 在 Trino CLI 中，您可以通过运行以下命令来查看目录：

```
SHOW CATALOGS;
```

创建 Iceberg 表

要创建 Iceberg 表，可以使用 CREATE TABLE 语句。 以下是创建使用 Iceberg 隐藏分区的分区表的示例：

```
CREATE TABLE iceberg.iceberg_db.iceberg_table (  
    userid int,  
    firstname varchar,  
    city varchar)  
WITH (  
    format = 'PARQUET',  
    partitioning = ARRAY['city', 'bucket(userid, 16)'],  
    location = 's3://<S3-bucket>/<prefix>');
```

Note

如果您未指定格式，则将使用您在上一节中配置的iceberg.file-format值。

要插入数据，请使用INSERT INTO命令。示例如下：

```
INSERT INTO iceberg.iceberg_db.iceberg_table (userid, firstname, city)
VALUES
  (1001, 'John', 'New York'),
  (1002, 'Mary', 'Los Angeles'),
  (1003, 'Mateo', 'Chicago'),
  (1004, 'Shirley', 'Houston'),
  (1005, 'Diego', 'Miami'),
  (1006, 'Nikki', 'Seattle'),
  (1007, 'Pat', 'Boston'),
  (1008, 'Terry', 'San Francisco'),
  (1009, 'Richard', 'Denver'),
  (1010, 'Pat', 'Phoenix');
```

从冰山桌上读书

您可以使用SELECT语句读取 Iceberg 表的最新状态，如下所示：

```
SELECT * FROM iceberg.iceberg_db.iceberg_table;
```

将数据更新到 Iceberg 表中

您可以使用MERGE INTO语句执行 upsert 操作（同时插入新记录和更新现有记录）。示例如下：

```
MERGE INTO iceberg.iceberg_db.iceberg_table target
USING (
  VALUES
    (1001, 'John Updated', 'Boston'),      -- Update existing user
    (1002, 'Mary Updated', 'Seattle'),     -- Update existing user
    (1011, 'Martha', 'Portland'),          -- Insert new user
    (1012, 'Paulo', 'Austin')              -- Insert new user
) AS source (userid, firstname, city)
ON target.userid = source.userid
WHEN MATCHED THEN
  UPDATE SET
    firstname = source.firstname,
    city = source.city
WHEN NOT MATCHED THEN
  INSERT (userid, firstname, city)
  VALUES (source.userid, source.firstname, source.city);
```

从 Iceberg 表中删除记录

要从 Iceberg 表中删除数据，请使用 `DELETE FROM` 表达式并指定与要删除的行匹配的筛选器。示例如下：

```
DELETE FROM iceberg.iceberg_db.iceberg_table WHERE userid IN (1003, 1004);
```

查询 Iceberg 表元数据

Iceberg 通过 SQL 提供对其元数据的访问。您可以通过查询命名空间来访问任何给定表 (`<table_name>`) 的元数据 "`<table_name>.$<metadata_table>`"。有关元数据表的完整列表，请参阅 Iceberg 文档中的[检查表](#)。

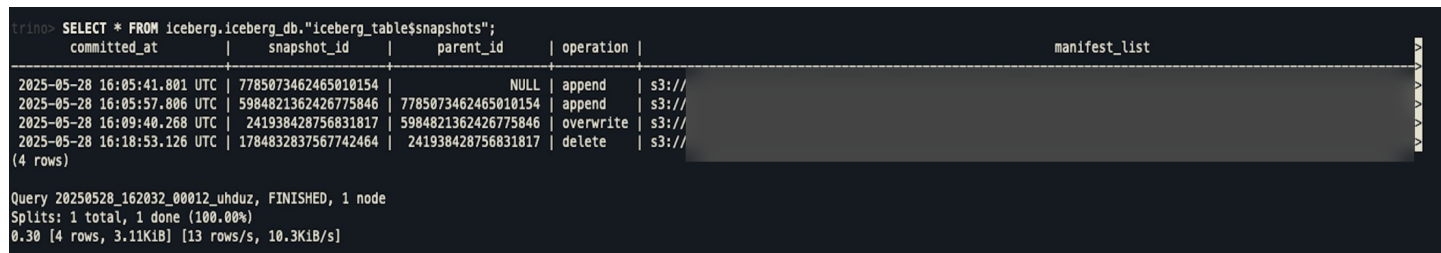
以下是用于检查 Iceberg 元数据的示例查询列表：

```
SELECT FROM iceberg.iceberg_db."iceberg_table$snapshots";
SELECT FROM iceberg.iceberg_db."iceberg_table$history";
SELECT FROM iceberg.iceberg_db."iceberg_table$partitions";
SELECT FROM iceberg.iceberg_db."iceberg_table$files";
SELECT FROM iceberg.iceberg_db."iceberg_table$manifests";
SELECT FROM iceberg.iceberg_db."iceberg_table$refs";
SELECT * FROM iceberg.iceberg_db."iceberg_table$metadata_log_entries";
```

例如，此查询：

```
SELECT * FROM iceberg.iceberg_db."iceberg_table$snapshots";
```

提供了输出：



The screenshot shows a terminal window with a SQL query and its output. The query is `SELECT * FROM iceberg.iceberg_db."iceberg_table$snapshots";`. The output is a table with columns: `committed_at`, `snapshot_id`, `parent_id`, `operation`, and `manifest_list`. There are 4 rows of data. Below the table, it says "(4 rows)". At the bottom, there is a status message: "Query 20250528_162032_00012_uhduz, FINISHED, 1 node Splits: 1 total, 1 done (100.00%) 0.30 [4 rows, 3.11KiB] [13 rows/s, 10.3KiB/s]"

committed_at	snapshot_id	parent_id	operation	manifest_list
2025-05-28 16:05:41.801 UTC	7785073462465010154	NULL	append	s3://
2025-05-28 16:05:57.806 UTC	5984821362426775846	7785073462465010154	append	s3://
2025-05-28 16:09:40.268 UTC	241938428756831817	5984821362426775846	overwrite	s3://
2025-05-28 16:18:53.126 UTC	1784832837567742464	241938428756831817	delete	s3://

使用时空旅行

Iceberg 表中的每个写入操作（插入、更新、更新或删除）都会创建一个新快照。然后，您可以使用这些快照进行时空旅行，以回到过去，检查表的过去状态。

以下时空旅行查询根据特定内容显示表的状态snapshot_id：

```
SELECT *  
FROM iceberg.iceberg_db.iceberg_table FOR VERSION AS OF 241938428756831817;
```

以下时空旅行查询根据特定的时间戳显示表的状态：

```
SELECT *  
FROM iceberg.iceberg_db.iceberg_table FOR TIMESTAMP AS OF TIMESTAMP '2025-05-28  
16:09:40.268 UTC'
```

将 Iceberg 与 Trino 搭配使用时的注意事项

Trino 对 Iceberg [表的写入操作遵循读时合并](#)的设计，因此它们会创建位置删除文件，而不是重写受更新或删除影响的整个数据文件。如果要使用写时复制方法，可以考虑使用 Spark 进行写入操作。

使用 Amazon Data Firehose 处理 Iceberg 表

Amazon Data Firehose 是一项无服务器的无代码服务，用于将来自日志、亚马逊日志 AWS WAF、Amazon Kinesis 数据流和适用于 Apache Kafka 的亚马逊托管流媒体（亚马逊 MSK）等 20 多个来源的数据流传输到亚马逊 S3、Amazon Redshift、Snowflake 和 Splunk 等目的地。CloudWatch AWS IoT

您可以使用 Firehose 将流数据直接传送到亚马逊 S3 中的 Apache Iceberg 表。使用 Firehose，您可以将记录从单个流路由到不同的 Apache Iceberg 表，并自动对表中的记录应用插入、更新和删除操作。Firehose 保证精确一次送到冰山牌桌。此功能需要使用 AWS Glue Data Catalog。

Firehose 还可以直接将流数据传输到亚马逊 S3 表。这些表提供了针对大规模分析工作负载进行了优化的存储，并包括可持续提高查询性能和降低表格数据存储成本的功能。

有关如何设置 Firehose 流以向 Apache Iceberg 表传送数据的信息，请参阅 Firehose 文档中的[设置 Firehose 流](#)或博客[文章使用亚马逊数据 Firehose 将实时数据流传输到亚马逊 S3 中的 Apache Iceberg 表](#)。

使用 Athena SQL 处理 Iceberg 表

Amazon Athena 为 Apache Iceberg 提供了内置支持，不需要额外的步骤或配置。本节详细概述了支持的功能，并提供了有关使用 Athena 与 Iceberg 表交互的高级指导。

版本和功能兼容性

冰山桌规格支持

Apache Iceberg 表规范指定了 Iceberg 表的行为方式。Athena 支持表格格式版本 2，因此您使用控制台、CLI 或 SDK 创建的任何 Iceberg 表本质上都使用该版本。

[如果您使用使用其他引擎创建的 Iceberg 表，例如亚马逊 EMR 上的 Apache Spark 或 AWS Glue，请务必使用表属性来设置表格式版本。](#)作为参考，请参阅本指南前面的[“创建和编写 Iceberg 表”](#)一节。

冰山功能支持

您可以使用 Athena 读取和写入 Iceberg 表。当您使用、和DELETE FROM语句更改数据时 UPDATEMERGE INTO，Athena 仅支持模式。 merge-on-read此属性无法更改。要使用更新或删除数据 copy-on-write，您必须使用其他引擎，例如亚马逊 EM AWS Glue R 上的 Apache Spark 或。下表汇总了 Athena 中对 Iceberg 功能的支持。

		DDL 支持		DML 支持		AWS Lake Formation 为了安全起见（可选）
表格格式		创建表	架构演变	读取数据	写入数据	行/列访问控制
Amazon Athena	版本 2	✓	✓	✓	X C opy-on-write	✓
					✓ M erge-on-read	✓

 Note

- Athena 不支持增量查询。
- 在 Athena 中，无论表属性中的写入时复制 (CoW) 设置如何，更新、删除和合并操作始终默认为读时合并 (MoR)，因为不支持 CoW。

使用 Iceberg 桌子

要快速开始在 Athena 中使用 Iceberg，请参阅本指南前面的“在 Athena SQL [中使用 Iceberg 表入门](#)”一节。

下表列出了限制和建议。

场景	限制	建议
生成表 DDL	使用其他引擎创建的 Iceberg 表可以具有在 Athena 中未公开的属性。对于这些表，无法生成 DDL。	在创建表的引擎中使用等效语句（例如，Spark 的 SHOW CREATE TABLE 语句）。
写入冰山表的对象中的随机 Amazon S3 前缀	默认情况下，使用 Athena 创建的 Iceberg 表会启用该属性。write.object-storage.enabled	要禁用此行为并获得对 Iceberg 表属性的完全控制权，请使用其他引擎（例如 Amazon EMR 上的 Spark 或）创建 Iceberg 表。AWS Glue
递增查询	Athena 目前不支持。	要使用增量查询来启用增量数据摄取管道，请在 Amazon EMR 上使用 Spark 或。AWS Glue

使用使用 Iceberg 表来处理冰山表 Pylceberg

本节介绍如何使用 [Pylceberg](#) 与 Iceberg 表进行交互。[提供的示例是样板代码](#)，您可以使用正确配置的凭证在 [Amazon Linux 2023 EC2 实例](#)、[AWS Lambda 函数](#) 或任何 Python 环境中运行这些代码。[AWS](#)

先决条件

Note

这些示例使用 [Pylceberg 1.9.1](#)。

要使用 Pylceberg，您需要 Pylceberg 并适用于 Python (Boto3) 的 AWS SDK 安装。以下是如何设置要使用的 Python 虚拟环境的示例，Pylceberg 以及 AWS Glue Data Catalog：

1. 使用 [pip python 软件包安装程序](#) 进行下载 [Pylceberg](#)。您还需要与 [Boto3](#) 进行交互。AWS 服务您可以使用以下命令配置本地 Python 虚拟环境进行测试：

```
python3 -m venv my_env
cd my_env/bin/
source activate
pip install "pyiceberg[pyarrow,pandas,glue]"
pip install boto3
```

2. 运行打开 python Python 外壳并测试命令。

连接到数据目录

要开始在中使用 Iceberg 表 AWS Glue，您首先需要连接到。AWS Glue Data Catalog

该 `load_catalog` 函数通过创建一个作为所有 Iceberg 操作的主接口的 [目录](#) 对象来初始化与数据目录的连接：

```
from pyiceberg.catalog import load_catalog
region = "us-east-1"

glue_catalog = load_catalog(
```

```
'default',
**{
    'client.region': region
},
type='glue'
)
```

列出和创建数据库

要列出现有数据库，请使用以下 `list_namespaces` 函数：

```
databases = glue_catalog.list_namespaces()
print(databases)
```

要创建新数据库，请使用以下 `create_namespace` 函数：

```
database_name="mydb"
s3_db_path=f"s3://amzn-s3-demo-bucket/{database_name}"

glue_catalog.create_namespace(database_name, properties={"location": s3_db_path})
```

创建和写作 Iceberg 表

未分区的表

以下是使用函数创建未分区的 Iceberg 表的示例：`create_table`

```
from pyiceberg.schema import Schema
from pyiceberg.types import NestedField, StringType, DoubleType

database_name="mydb"
table_name="pyiceberg_table"
s3_table_path=f"s3://amzn-s3-demo-bucket/{database_name}/{table_name}"

schema = Schema(
    NestedField(1, "city", StringType(), required=False),
    NestedField(2, "lat", DoubleType(), required=False),
    NestedField(3, "long", DoubleType(), required=False),
)
```

```
glue_catalog.create_table(f"{database_name}.{table_name}", schema=schema,
    location=s3_table_path)
```

你可以使用该 `list_tables` 函数来检查数据库中的表列表：

```
tables = glue_catalog.list_tables(namespace=database_name)
print(tables)
```

你可以使用 `append` 函数 and 在 Iceberg 表中插入数据：PyArrow

```
import pyarrow as pa
df = pa.Table.from_pylist([
    {"city": "Amsterdam", "lat": 52.371807, "long": 4.896029},
    {"city": "San Francisco", "lat": 37.773972, "long": -122.431297},
    {"city": "Drachten", "lat": 53.11254, "long": 6.0989},
    {"city": "Paris", "lat": 48.864716, "long": 2.349014},
],
)

table = glue_catalog.load_table(f"{database_name}.{table_name}")
table.append(df)
```

分区表

以下是使用函数和创建带有[隐藏分区的分区](#) Iceberg 表的示例：`create_tablePartitionSpec`

```
from pyiceberg.schema import Schema
from pyiceberg.types import (
    NestedField,
    StringType,
    FloatType,
    DoubleType,
    TimestampType,
)

# Define the schema
schema = Schema(
    NestedField(field_id=1, name="datetime", field_type=TimestampType(),
        required=True),
    NestedField(field_id=2, name="drone_id", field_type=StringType(), required=True),
    NestedField(field_id=3, name="lat", field_type=DoubleType(), required=False),
```

```

        NestedField(field_id=4, name="lon", field_type=DoubleType(), required=False),
        NestedField(field_id=5, name="height", field_type=FloatType(), required=False),
    )

from pyiceberg.partitioning import PartitionSpec, PartitionField
from pyiceberg.transforms import DayTransform

partition_spec = PartitionSpec(
    PartitionField(
        source_id=1, # Refers to "datetime"
        field_id=1000,
        transform=DayTransform(),
        name="datetime_day"
    )
)

database_name="mydb"
partitioned_table_name="pyiceberg_table_partitioned"
s3_table_path=f"s3://amzn-s3-demo-bucket/{database_name}/{partitioned_table_name}"

glue_catalog.create_table(
    identifier=f"{database_name}.{partitioned_table_name}",
    schema=schema,
    location=s3_table_path,
    partition_spec=partition_spec
)

```

您可以采用与未分区表相同的方式向分区表中插入数据。分区是自动处理的。

```

from datetime import datetime
arrow_schema = pa.schema([
    pa.field("datetime", pa.timestamp("us"), nullable=False),
    pa.field("drone_id", pa.string(), nullable=False),
    pa.field("lat", pa.float64()),
    pa.field("lon", pa.float64()),
    pa.field("height", pa.float32()),
])

data = [
    {
        "datetime": datetime(2024, 6, 1, 12, 0, 0),
        "drone_id": "drone_001",
        "lat": 52.371807,

```

```
        "lon": 4.896029,
        "height": 120.5,
    },
    {
        "datetime": datetime(2024, 6, 1, 12, 5, 0),
        "drone_id": "drone_002",
        "lat": 37.773972,
        "lon": -122.431297,
        "height": 150.0,
    },
    {
        "datetime": datetime(2024, 6, 2, 9, 0, 0),
        "drone_id": "drone_001",
        "lat": 53.11254,
        "lon": 6.0989,
        "height": 110.2,
    },
    {
        "datetime": datetime(2024, 6, 2, 9, 30, 0),
        "drone_id": "drone_003",
        "lat": 48.864716,
        "lon": 2.349014,
        "height": 145.7,
    },
]

df = pa.Table.from_pylist(data, schema=arrow_schema)

table = glue_catalog.load_table(f"{database_name}.{partitioned_table_name}")
table.append(df)
```

读取数据

您可以使用该 Pylceberg `scan` 函数从 Iceberg 表中读取数据。您可以筛选行、选择特定列并限制返回的记录数。

```
table= glue_catalog.load_table(f"{database_name}.{table_name}")
scan_df = table.scan(
    row_filter=(
        f"city = 'Amsterdam'"
    ),
    selected_fields=("city", "lat"),
```

```
    limit=100,  
).to_pandas()  
  
print(scan_df)
```

删除数据

该 Pylceberg `delete` 函数允许您使用以下方法从表中删除记录 `delete_filter` :

```
table = glue_catalog.load_table(f"{database_name}.{table_name}")  
table.delete(delete_filter="city == 'Paris'")
```

访问元数据

Pylceberg 提供了多种访问表元数据的函数。您可以通过以下方式查看有关表快照的信息 :

```
#List of snapshots  
table.snapshots()  
  
#Current snapshot  
table.current_snapshot()  
  
#Take a previous snapshot  
second_last_snapshot_id=table.snapshots()[-2].snapshot_id  
print(f"Second last SnapshotID: {second_last_snapshot_id}")
```

有关可用元数据的详细列表，请参阅 Pylceberg 文档的[元数据](#)代码参考部分。

使用时空旅行

您可以使用时空旅行的表快照来访问表的先前状态。以下是查看上次操作之前的表状态的方法 :

```
second_last_snapshot_id=table.snapshots()[-2].snapshot_id  
  
time_travel_df = table.scan(  
    limit=100,  
    snapshot_id=second_last_snapshot_id  
)  
.to_pandas()
```

```
print(time_travel_df)
```

有关可用函数的完整列表，请参阅 Pylceberg [Python API](#) 文档。

使用 Iceberg 表格格式规范版本 3

Apache Iceberg 表格格式规范的最新版本是版本 3。此版本引入了用于构建 PB 级数据湖的高级功能，提高了性能并减少了运营开销。它解决了版本 2 中遇到的常见性能瓶颈，尤其是在批量更新和合规性删除操作方面。

AWS 为 Iceberg 版本 3 规范中定义的删除向量和行谱系提供支持。Apache Spark 在以下 AWS 服务版本中提供了这些功能。

AWS 服务	版本 3 支持
适用于 Apache Spark 的亚马逊 EMR	亚马逊 EMR 版本 7.12 及更高版本
AWS Glue	是
AWS Glue: I ceberg REST API , 表格维护	是
亚马逊 SageMaker 统一工作室笔记本电脑	是
亚马逊 S3 表格 : I ceberg REST API , 表格维护	是
亚马逊 Athena (Trino)	否

版本 3 中的主要功能

删除向量将版本 2 中使用的位置删除文件替换为存储为 Puffin 文件的高效二进制格式。这消除了随机批量更新和《通用数据保护条例》(GDPR) 合规删除带来的写入放大，并显著降低了维护新数据的开销。处理高频更新的组织将看到写入性能立即得到改善，并通过减少小文件来降低存储成本。

行沿袭可在行级别实现精确的变更跟踪。您的下游系统可以通过递增方式处理更改，从而加快数据管道并降低更改数据捕获 (CDC) 工作流的计算成本。此内置功能消除了实施自定义变更跟踪的需求。

版本兼容性

版本 3 保持了与版本 2 表的向后兼容性。AWS 服务同时支持版本 2 和版本 3 表，因此您可以：

- 对版本 2 和版本 3 的表运行查询。

- 无需重写数据，即可将现有版本 2 表升级到版本 3。
- 运行跨版本 2 和版本 3 快照的时空旅行查询。
- 使用架构演变和跨表版本的隐藏分区。

版本 3 入门

先决条件

在使用版本 3 的表之前，请确保您已具备以下条件：

- AWS 账户 具有适当的 AWS Identity and Access Management (IAM) 权限。
- 访问一项或多项 AWS 分析服务（亚马逊 EMR、AWS Glue Amazon SageMaker Unified Studio 笔记本电脑或 Amazon S3 表）。
- 用于存储表数据和元数据的 S3 存储桶。
- 用于开始使用 Amazon S3 表的表存储桶，或者如果您正在构建自己的 Iceberg 基础架构，则是通用的 S3 存储桶。
- 已配置的 AWS Glue 目录。

创建版本 3 表

创建新表

要创建新的 Iceberg 版本 3 表，请将 `format-version` 表格属性设置为 3。

使用 Spark SQL：

```
CREATE TABLE IF NOT EXISTS myns.orders_v3 (  
  order_id bigint,  
  customer_id string,  
  order_date date,  
  total_amount decimal(10,2),  
  status string,  
  created_at timestamp  
)  
USING iceberg  
TBLPROPERTIES (  
  'format-version' = '3'
```

```
)
```

将版本 2 表升级到版本 3

无需重写数据，即可以原子方式将现有版本 2 表升级到版本 3。

使用 Spark SQL：

```
ALTER TABLE myns.existing_table
SET TBLPROPERTIES ('format-version' = '3')
```

Important

版本 3 是单向升级。表从版本 2 升级到版本 3 后，无法通过标准操作将其降级回版本 2。

升级期间发生的操作：

- 新的元数据快照是以原子方式创建的。
- 现有的 Parquet 数据文件可以重复使用。
- 行谱系字段已添加到表元数据中。

升级后：

- 下一次压缩操作将删除旧版本 2 的删除文件。
- 新的修改将使用版本 3 的删除矢量文件。

升级不会对行系变更跟踪记录进行历史回填。

启用删除向量

要利用删除向量进行更新、删除和合并，请配置您的写入模式。

使用 Spark SQL：

```
ALTER TABLE myns.orders_v3
SET TBLPROPERTIES ('format-version' = '3',
                    'write.delete.mode' = 'merge-on-read',
                    'write.update.mode' = 'merge-on-read',
```

```
'write.merge.mode' = 'merge-on-read'  
)
```

这些设置可确保更新、删除和合并操作创建删除矢量文件，而不是重写整个数据文件。

使用行谱系进行变更跟踪

版本 3 会自动添加行谱系元数据字段以跟踪更改。

使用 Spark SQL：

```
# Query with parameter value provided  
last_processed_sequence = 47  
  
SELECT  
    id,  
    data,  
    _row_id,  
    _last_updated_sequence_number  
FROM myns.orders_v3  
WHERE _last_updated_sequence_number > :last_processed_sequence
```

该 `_row_id` 字段唯一标识每行，并 `_last_updated_sequence_number` 跟踪该行的上次修改时间。使用这些字段可以：

- 识别已更改的行以进行增量处理。
- 跟踪数据沿革以确保合规性。
- 优化 CDC 管道。
- 通过仅处理更改来降低计算成本。

版本 3 的最佳实践

何时使用版本 3

在以下情况下，可以考虑升级到版本 3 或从版本 3 开始：

- 您经常执行批量更新或删除。
- 您需要满足 GDPR 或合规删除要求。
- 您的工作负载涉及高频 Upsert。

- 您需要高效的 CDC 工作流程。
- 您想降低小文件的存储成本。
- 您需要更好的变更跟踪功能。

优化写入性能

- 为更新密集型工作负载启用删除向量：

```
SET TBLPROPERTIES (  
  'write.delete.mode' = 'merge-on-read',  
  'write.update.mode' = 'merge-on-read',  
  'write.merge.mode' = 'merge-on-read'  
)
```

- 配置适当的文件大小：

```
SET TBLPROPERTIES (  
  'write.target-file-size-bytes' = '536870912' -- 512 MB  
)
```

优化读取性能

- 使用行谱系进行增量处理。
- 使用时空旅行无需复制即可访问历史数据。
- 启用统计数据收集以更好地进行查询规划。

迁移策略

从版本 2 迁移到版本 3 时，请遵循以下最佳实践：

- 首先在非生产环境中进行测试，以验证升级过程和性能。
- 在低活动期间进行升级，以最大限度地减少对并发操作的影响。
- 监控初始性能，并在升级后跟踪指标。
- 升级后运行压缩以合并已删除的文件。
- 更新您的团队文档以反映版本 3 的功能。

兼容性注意事项

- 引擎版本-确保所有访问该表的引擎都支持版本 3。
- 第三方工具-在升级之前，请验证您的工具的版本 3 兼容性。
- Backup 策略 — 测试基于快照的恢复程序。
- 监控-更新针对版本 3 特定指标的监控仪表板。

问题排查

常见问题

错误：“不支持格式版本 3”

- 确认您的引擎版本支持版本 3。有关详细信息，请参阅本节开头的[表格](#)。
- 检查目录兼容性。
- 请确保您使用的是最新版本的 AWS 服务。

升级后性能下降

- 确认没有压实失败。有关更多信息，请参阅 Amazon S3 文档[中的 S3 表的日志记录和监控](#)。
- 确认删除向量已启用。应设置以下属性：

```
SET TBLPROPERTIES (  
  'write.delete.mode' = 'merge-on-read',  
  'write.update.mode' = 'merge-on-read',  
  'write.merge.mode' = 'merge-on-read'  
)
```

您可以使用以下代码验证表属性：

```
DESCRIBE FORMATTED myns.orders_v3
```

- 查看您的分区策略。过度分区可能会导致文件变小。运行以下查询以获取表格的平均文件大小：

```
SELECT avg(file_size_in_bytes) as avg_file_size_bytes  
FROM myns.orders_v3.files
```

与第三方工具不兼容

- 验证该工具是否支持版本 3 规范。
- 考虑为不支持的工具维护版本 2 表。
- 请联系工具供应商，了解其版本 3 支持时间表。

获取帮助

- 有关 AWS 服务特定问题，请联系[AWS 支持](#)。
- 要从 Iceberg 社区获得帮助，请使用 [Iceberg Slack 频道](#)。
- 有关使用 AWS 服务 管理分析工作负载的信息，请参阅[上的 Analytics AWS](#)。

定价

- [亚马逊 EMR 计算和存储定价](#)
- [亚马逊 SageMaker 定价](#)
- [AWS Glue 任务运行和数据目录定价](#)
- [S3 表存储和请求定价](#)

可用性

在 Amazon EMR AWS Glue、AWS Glue Data Catalog 和 S3 表格运行的所有 AWS 区域 地方，都支持 Iceberg 表格式规范版本 3。有关区域可用性，请参阅[按区域划分的 AWS 服务](#)。

其他资源

- [Apache Iceberg 文档](#)
- [Apache Iceberg 桌规格](#)
- [将表格数据从 Amazon S3 迁移到 S3 表的指南](#)
- [教程：S3 表入门](#)

将现有表迁移到 Iceberg

本节重点介绍如何将现有 Hive-style 表格迁移到 Iceberg 格式。[它适用于使用传统 Hive-compatible 格式（例如 Apache Parquet 或 Apache ORC）的表。](#)此信息不适用于已经使用现代表格格式的表，例如 Linux Foundation Delta Lake 或 Apache Hudi。

要将当前的 Hive-style 表迁移到 Iceberg 格式，您可以使用就地迁移或完整数据迁移：

- [In-place 迁移](#)是在现有数据文件之上生成 Iceberg 的元数据文件的过程。
- [完整数据迁移](#)会创建 Iceberg 元数据层，并将现有数据文件从原始表重写到新的 Iceberg 表。

以下各节详细概述了每种迁移方法，包括分步说明和实施注意事项。有关这些迁移策略的更多信息，请参阅 Iceberg 文档的[表迁移](#)部分。

在查看了就地和完整数据迁移方法的详细信息后，请参阅以下两个关键部分，以帮助你制定决策：

- [选择迁移策略](#)可通过一系列问题和场景提供指导，帮助您根据具体要求和用例确定最合适的迁移方法。
- [迁移选项摘要](#)提供了一个全面的表格，比较了不同迁移选项的关键特征和注意事项。此表可作为快速参考指南，并提供功能比较，以帮助您了解方法之间的技术权衡。

In-place 迁移

In-place 迁移无需重写所有数据文件。取而代之的是生成 Iceberg 元数据文件并将其链接到您的现有数据文件。这种方法通常更快、更具成本效益，特别是对于具有兼容文件格式（例如 Parquet、Avro 和 ORC）的大型数据集或表。

Note

In-place 迁移到 [Amazon S3 表](#)时不能使用迁移。

Iceberg 为实现就地迁移提供了两个主要选项：

- 使用[快照](#)过程创建新的 Iceberg 表，同时保持源表不变。有关更多信息，请参阅 Iceberg 文档中的[快照表](#)。

- 使用[迁移](#)过程创建新的 Iceberg 表作为源表的替换。有关更多信息，请参阅 Iceberg 文档中的[迁移表](#)。尽管此过程适用于 Hive Metastore (HMS)，但它目前与 Hive Metastore 不兼容。AWS Glue Data Catalog 本指南后 AWS Glue Data Catalog 面部分的“[复制表迁移过程](#)”提供了一种解决方法，可使用数据目录实现类似的结果。

使用 snapshot 或执行就地迁移后 migrate，某些数据文件可能仍处于未迁移状态。当写入者在迁移期间或迁移之后继续写入源表时，通常会发生这种情况。要将这些剩余文件合并到 Iceberg 表中，可以使用 [add_files](#) 过程。有关更多信息，请参阅 Iceberg 文档中的[添加文件](#)。

假设你有一个在 Athena 中创建并填充的 Parquet-based products 表，如下所示：

```
CREATE EXTERNAL TABLE mydb.products (  
    product_id INT,  
    product_name STRING  
)  
PARTITIONED BY (category STRING)  
STORED AS PARQUET  
LOCATION 's3://amzn-s3-demo-bucket/products/';  
  
INSERT INTO mydb.products  
VALUES  
    (1001, 'Smartphone', 'electronics'),  
    (1002, 'Laptop', 'electronics'),  
    (2001, 'T-Shirt', 'clothing'),  
    (2002, 'Jeans', 'clothing');
```

以下各节说明如何在此表中使用 snapshot 和 migrate 过程。

选项 1：快照过程

该 snapshot 过程创建一个新的 Iceberg 表，该表具有不同的名称，但会复制源表的架构和分区。此操作使源表在操作期间和操作之后都完全不变。它可以有效地创建表的轻量级副本，这对于测试场景或数据探索特别有用，而不会冒修改原始数据源的风险。这种方法可以实现一个过渡期，在这个过渡期，原始表和 Iceberg 表都保持可用（参见本节末尾的注释）。测试完成后，您可以通过将所有写入者和读者过渡到新表来将新 Iceberg 表移至生产环境。

你可以在任何亚马逊 EMR 部署模式中使用 Spark 来运行该 snapshot 过程（例如，EC2 上的 Amazon EMR、EKS 上的 Amazon EMR、EMR Serverless）和。AWS Glue

要使用 snapshot Spark 程序测试就地迁移，请执行以下步骤：

1. 启动 Spark 应用程序并使用以下设置配置 Spark 会话：

- "spark.sql.extensions":"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
- "spark.sql.catalog.spark_catalog":"org.apache.iceberg.spark.SparkSessionCatalog"
- "spark.sql.catalog.spark_catalog.type":"glue"
- "spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AWSGlueMetastoreClientFactory"

2. 运行以下snapshot过程创建指向原始表数据文件的新 Iceberg 表：

```
spark.sql(f"""
CALL system.snapshot(
  source_table => 'mydb.products',
  table => 'mydb.products_iceberg',
  location => 's3://amzn-s3-demo-bucket/products_iceberg/'
)
""")
).show(truncate=False)
```

输出 DataFrame 包含 imported_files_count (已添加的文件数)。

3. 通过查询来验证新表：

```
spark.sql(f"""
SELECT * FROM mydb.products_iceberg LIMIT 10
""")
).show(truncate=False)
```

注意

- 运行该过程后，对源表进行的任何数据文件修改都将使生成的表失去同步。您添加的新文件将在 Iceberg 表中不可见，而您删除的文件将影响 Iceberg 表中的查询功能。要避免同步问题，请执行以下操作：
 - 如果新的 Iceberg 表用于生产用途，请停止所有写入原始表的进程，并将它们重定向到新表。
 - 如果您需要过渡期，或者如果新的 Iceberg 表用于测试目的，请参阅本节[后面的就地迁移后保持 Iceberg 表同步，以获取有关维护表同步](#)的指导。
- 使用该snapshot过程时，该gc.enabled属性将在创建的 Iceberg 表的表属性false中设置为。此设置禁止诸如expire_snapshotsremove_orphan_files、或PURGE选项之

类 DROP TABLE 的操作，这些操作会以物理方式删除数据文件。仍然允许不直接影响源文件的 Iceberg 删除或合并操作。

- 要使您的新 Iceberg 表完全正常运行，并且对物理删除数据文件的操作没有限制，您可以将 `gc.enabled` 表属性更改为 `true`。但是，此设置将允许影响源数据文件的操作，这可能会破坏对原始表的访问。因此，只有在不再需要维护原始表的功能时才更改该 `gc.enabled` 属性。例如：

```
spark.sql(f"""
ALTER TABLE mydb.products_iceberg
SET TBLPROPERTIES ('gc.enabled' = 'true');
""")
```

选项 2：迁移程序

该 `migrate` 过程创建一个新的 Iceberg 表，该表的名称、架构和分区与源表相同。当此过程运行时，它会锁定源表并将其重命名为 `<table_name>_BACKUP_`（或由 `backup_table_name` 过程参数指定的自定义名称）。

Note

如果将 `drop_backup` 过程参数设置为 `true`，则原始表将不会保留为备份。

因此，`migrate` 表过程要求在执行操作之前停止所有影响源表的修改。在运行该 `migrate` 过程之前：

- 停止所有与源表交互的写入器。
- 修改原生不支持 Iceberg 的读者和作者以启用 Iceberg 支持。

例如：

- Athena 继续工作，无需任何修改。
- Spark 需要：
 - 要包含在类路径中的 Iceberg Java 存档 (JAR) 文件（参见本指南前面章节中的“[在 Amazon EMR 中使用 Iceberg](#)”和“[使用 Iceberg](#)”）。AWS Glue
 - SparkSessionCatalog 以下 Spark 会话目录配置（用于添加 Iceberg 支持，同时维护非 Iceberg 表的内置目录功能）：

- "spark.sql.extensions":"org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
- "spark.sql.catalog.spark_catalog":"org.apache.iceberg.spark.SparkSessionCatalog"
- "spark.sql.catalog.spark_catalog.type":"glue"
- "spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClient"

运行该过程后，您可以使用新的 Iceberg 配置重新启动写入器。

目前，该migrate过程与不兼容 AWS Glue Data Catalog，因为数据目录不支持该RENAME操作。因此，我们建议您仅在使用 Hive Metastore 时使用此过程。如果您使用的是数据目录，请参阅[下一节](#)以了解替代方法。

你可以在所有亚马逊 EMR 部署模型（EC2 上的 Amazon EMR、EKS 上的 Amazon EMR、EMR Serverless）上运行该migrate过程，但它需要配置与 Hive Metastore 的连接。建议选择 EC2 上的 Amazon EMR，因为它提供了内置的 Hive Metastore 配置，可以最大限度地降低设置复杂性。

要使用 migrate Spark 程序从配置了 Hive Metastore 的 EC2 集群上的 Amazon EMR 进行就地迁移，请执行以下步骤：

1. 启动 Spark 应用程序并将 Spark 会话配置为使用 Iceberg Hive 目录实现。例如，如果你使用的是 pyspark CLI：

```
pyspark --conf
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.type=hive
```

2. 在 Hive Metastore 中创建products表。这是源表，它已经存在于典型的迁移中。

- a. 在 Hive Metastore 中创建products外部 Hive 表以指向 Amazon S3 中的现有数据：

```
spark.sql(f"""
CREATE EXTERNAL TABLE products (
    product_id INT,
    product_name STRING
)
PARTITIONED BY (category STRING)
STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/products/';
""")
```

```
)
```

b. 使用MSCK REPAIR TABLE以下命令添加现有分区：

```
spark.sql(f"""
MSCK REPAIR TABLE products
""")
```

c. 通过运行SELECT查询来确认表中是否包含数据：

```
spark.sql(f"""
SELECT * FROM products
""")
).show(truncate=False)
```

示例输出：

```
>>> spark.sql(f"""
... SELECT * FROM products
... """)
... ).show(truncate=False)
+-----+-----+-----+
|product_id|product_name|category|
+-----+-----+-----+
|1001      |Smartphone  |electronics|
|1002      |Laptop      |electronics|
|2001      |T-Shirt     |clothing  |
|2002      |Jeans       |clothing  |
+-----+-----+-----+
```

3. 使用 Iceberg migrate 程序：

```
df_res=spark.sql(f"""
CALL system.migrate(
table => 'default.products'
)
""")

df_res.show()
```

输出 DataFrame 包含migrated_files_count (添加到 Iceberg 表中的文件数量)：

```
>>> df_res.show()
+-----+
|migrated_files_count|
+-----+
|                2|
+-----+
```

4. 确认备份表已创建：

```
spark.sql("show tables").show()
```

示例输出：

```
>>> spark.sql("show tables").show()
+-----+-----+-----+
|namespace|      tableName|isTemporary|
+-----+-----+-----+
|  default|    products|      false|
|  default|products_backup_|      false|
+-----+-----+-----+
```

5. 通过查询 Iceberg 表来验证操作：

```
spark.sql(f"""
SELECT * FROM products
""")
).show(truncate=False)
```

注意

- 运行该过程后，如果当前所有查询或写入源表的进程未正确配置 Iceberg 支持，则这些进程都将受到影响。因此，我们建议您按照以下步骤操作：
 - 迁移前使用源表停止所有进程。
 - 执行迁移。
 - 使用正确的 Iceberg 设置重新激活进程。
- 如果在迁移过程中修改了数据文件（添加了新文件或删除了文件），则生成的表将不同步。有关同步选项，请参阅本节[后面的就地迁移后使 Iceberg 表保持同步](#)。

在中复制表迁移过程 AWS Glue Data Catalog

您可以按照以下步骤在中复制迁移过程的结果 AWS Glue Data Catalog（备份原始表并将其替换为 Iceberg 表）：

1. 使用快照过程创建新的 Iceberg 表，该表指向原始表的数据文件。
2. 备份数据目录中的原始表元数据：
 - a. 使用 [GetTable](#) API 检索源表定义。
 - b. 使用 [GetPartitions](#) API 检索源表分区定义。
 - c. 使用 [CreateTable](#) API 在数据目录中创建备份表。
 - d. 使用 [CreatePartition](#) 或 [BatchCreatePartition](#) API 将分区注册到数据目录中的备份表。
3. 将 `gc.enabled` Iceberg 表属性更改为 `false`，以启用全表操作。
4. 删除原始表。
5. 在表根位置的元数据文件夹中找到 Iceberg 表元数据 JSON 文件。
6. 使用 [register_table 过程在数据目录中注册新表](#)，并使用该过程创建的原始表名和 `metadata.json` 文件的位置：`snapshot`

```
spark.sql(f"""
CALL system.register_table(
    table => 'mydb.products',
    metadata_file => '{iceberg_metadata_file}'
)
""")
).show(truncate=False)
```

就地迁移后保持 Iceberg 表同步

该 `add_files` 过程提供了一种灵活的方法将现有数据合并到 Iceberg 表中。具体而言，它通过在 Iceberg 的元数据层中引用现有数据文件（例如 Parquet 文件）的绝对路径来注册它们。默认情况下，该过程会将所有表分区中的文件添加到 Iceberg 表，但您可以有选择地添加来自特定分区的文件。这种选择性方法在以下几种情况下特别有用：

- 在初始迁移后向源表中添加新分区时。
- 在初始迁移后向现有分区中添加或删除数据文件时。但是，重新添加修改后的分区需要先删除分区。本节稍后将提供有关此内容的更多信息。

以下是在执行就地迁移 (snapshot或migrate) 后使用该add_file过程以使新的 Iceberg 表与源数据文件保持同步的一些注意事项：

- 将新数据添加到源表中的新分区时，请使用带有partition_filter选项的add_files过程有选择地将这些新增数据合并到 Iceberg 表中：

```
spark.sql(f"""
CALL system.add_files(
  source_table => 'mydb.products',
  table => 'mydb.products_iceberg',
  partition_filter => map('category', 'electronics')
).show(truncate=False)
```

或者：

```
spark.sql(f"""
CALL system.add_files(
  source_table => '`parquet`.`s3://amzn-s3-demo-bucket/products/`',
  table => 'mydb.products_iceberg',
  partition_filter => map('category', 'electronics')
).show(truncate=False)
```

- 当您指定partition_filter选项时，该add_files过程会扫描整个源表或特定分区中的文件，并尝试将其找到的所有文件添加到 Iceberg 表中。默认情况下，check_duplicate_files过程属性设置为true，如果文件已存在于 Iceberg 表中，则程序将无法运行。这很重要，因为没有内置选项可以跳过先前添加的文件，禁用check_duplicate_files会导致文件被添加两次，从而创建重复的文件。将新文件添加到源表时，请按照以下步骤操作：

1. 对于新分区，add_files与一起使用partition_filter，仅从新分区导入文件。
2. 对于现有分区，请先从 Iceberg 表中删除该分区，然后对该分区重新运行add_files，指定.partition_filter例如：

```
# We initially perform in-place migration with snapshot
spark.sql(f"""
CALL system.snapshot(
  source_table => 'mydb.products',
  table => 'mydb.products_iceberg',
  location => 's3://amzn-s3-demo-bucket/products_iceberg/'
)
""")
).show(truncate=False)
```

```
# Then on the source table, some new files were generated under the
category='electronics' partition. Example:
spark.sql("""
INSERT INTO mydb.products
VALUES (1003, 'Tablet', 'electronics')
""")

# We delete the modified partition from the Iceberg table. Note this is a metadata
operation only
spark.sql("""
DELETE FROM mydb.products_iceberg WHERE category = 'electronics'
""")

# We add_files from the modified partition
spark.sql("""
CALL system.add_files(
  source_table => 'mydb.products',
  table => 'mydb.products_iceberg',
  partition_filter => map('category', 'electronics')
)
""").show(truncate=False)
```

 **Note**

每个add_files操作都会生成一个包含附加数据的新的 Iceberg 表快照。

选择正确的就地迁移策略

要选择最佳的就地迁移策略，请考虑下表中的问题。

问题	建议	说明
您是否想在不重写数据的情况下快速迁移，同时保持 Hive 和 Iceberg 表可供测试或逐步过渡使用？	snapshot程序之后是add_files 程序	使用该snapshot过程通过克隆架构并引用数据文件来创建新的 Iceberg 表，而无需修改源表。使用此add_files 过程合并迁移后添加或修改的分

问题	建议	说明
		区，注意重新添加修改后的分区需要先删除分区。
你是否在使用 Hive Metastore，是否想在不重写数据的情况下立即用 Iceberg 表替换 Hive 表？	migrate程序之后是add_files 程序	使用该migrate过程创建 Iceberg 表，备份源表，然后用 Iceberg 版本替换原始表。 注意：此选项与 Hive Metastore 兼容，但不兼容。AWS Glue Data Catalog 使用此add_files 过程合并迁移后添加或修改的分区，注意重新添加修改后的分区需要先删除分区。

问题	建议	说明
您是否正在使用 AWS Glue Data Catalog Hive 表，是否需要立即用 Iceberg 表替换 Hive 表，而不必重写数据？	调整migrate程序，然后是add_files 程序	复制migrate过程行为： 1. snapshot用于创建 Iceberg 表。 2. 使用 AWS Glue API 备份原始表元数据。 3. gc.enabled 在 Iceberg 表属性上启用。 4. 删除原始表。 5. register_table 用于使用原始名称创建新的表条目。 注意：此选项需要手动处理元数据备份 AWS Glue 的 API 调用。 使用此add_files 过程合并迁移后添加或修改的分区，注意重新添加修改后的分区需要先删除分区。

完整数据迁移

完整数据迁移会重新创建数据文件和元数据。与就地迁移相比，这种方法需要更长的时间，并且需要更多的计算资源。但是，完整数据迁移为提高表格质量和优化数据存储和访问模式提供了重要机会。

在完整数据迁移期间，您可以执行多项有益的操作，例如数据验证以确保完整性和正确性，修改架构以更好地满足当前要求，以及调整分区策略以提高查询性能。您还可以对数据进行重新排序以优化常见的访问模式，实现 Iceberg 隐藏分区以提高查询效率，并根据需要执行文件格式转换（例如，从 CSV 到 Parquet）。

这些功能使完整数据迁移非常适合过渡到 Iceberg 格式以及全面完善和优化您的数据存储策略。尽管完整数据迁移需要更多的前期时间和资源，但由此带来的数据质量、组织和查询性能的改善可以带来长期的好处。要实现完整数据迁移，请使用以下选项之一：

- 在 SparkCREATE TABLE ... AS SELECT (亚马逊 EMR AWS Glue或) 或 Athena 中使用 (CTAS) 声明。您可以使用和子TBLPROPERTIES句为新 Iceberg 表设置分区规格PARTITIONED BY和表属性。您可以根据需要更改新表的架构和分区，而不必从源表继承它们。
- 使用 Amazon EM AWS Glue R 上的 Spark 或者，从源表中读取数据并将数据写成新的 Iceberg 表。有关更多信息，请参阅 Iceberg 文档中的[创建表](#)。

选择迁移策略

过渡到 Iceberg 格式时，在就地迁移和完全迁移之间做出选择至关重要。要确定最适合您特定需求的方法，请考虑以下问题和建议：

问题	建议
数据文件格式是什么（例如，CSV 或 Apache Parquet）？	• 如果您的表格文件格式为 Parquet、ORC 或 Avro，请考虑就地迁移。 • 对于其他格式，例如 CSV、JSON 等，请使用完整数据迁移。
是否要更新或整合表架构？	• 如果您想使用 Iceberg 原生功能来改进表架构，请考虑就地迁移。例如，您可以在迁移后重命名列。（可以在 Iceberg 元数据层中更改架构。） • 如果您因为不再需要整列而想要删除它们，我们建议您使用完整数据迁移。
更改分区策略会给表带来好处吗？	• 如果 Iceberg 的分区方法满足您的要求（例如，使用新的分区布局存储新数据，而现有分区保持原样），请考虑就地迁移。 • 如果要在表中使用隐藏分区，请考虑完全数据迁移。有关隐藏分区的更多信息，请参阅“最佳实践”部分。
添加或更改排序顺序策略会给表格带来好处吗？	• 添加或更改数据的排序顺序需要重写数据集。在这种情况下，可以考虑使用完整数据迁移。

问题	建议
	对于重写所有表分区成本高得令人望而却步的大型表，可以考虑使用就地迁移，并对最常访问的分区运行压缩（启用排序）。
表格里有很多小文件吗？	- 将小文件合并成较大的文件需要重写数据集。在这种情况下，可以考虑使用完整数据迁移。 - 对于重写所有表分区成本高得令人望而却步的大型表，可以考虑使用就地迁移，并对最常访问的分区运行压缩（启用排序）。

迁移选项摘要

下表汇总了每种迁移选项的主要特征和注意事项。

功能	In-place 迁移 快照	In-place 迁移 migrate	完整数据迁移 CTAS 或（创建表格 + 插入）
作为迁移过程一部分的数据布局改进			
• Re-sort 数据	⊗否	⊗否	⊙是
• 更改分区（例如，使用 Iceberg	⊗否	⊗否	⊙是

功能	In-place 迁移 快照	In-place 迁移 migrate	完整数据迁移 CTAS 或 (创建表格 + 插入)
隐藏分区)			
• 更改表架构	⊗否	⊗否	☑是
• 优化文件大小	⊗否	⊗否	☑是
• 在添加数据之前验证现有数据的架构	⊗否	⊗否	☑是
支持的文件格式	Parquet、Avro、ORC	Parquet、Avro、ORC	Parquet、Avro、ORC、JSON、CSV
用冰山表替换源表	⊗否 (创建新表，但通过其他步骤可以替换源表)	☑是 (创建备份表并用 Iceberg 表替换源表)	⊗否 (创建新表)
源表影响			

功能	In-place 迁移 快照	In-place 迁移 migrate	完整数据迁移 CTAS 或 (创建表格 + 插入)
Iceberg 表上的文件删除操作 (expapsho 操作，使用清除功能删除表)	损坏源表	损坏备份表	安全，来源不受影响
冰山桌子冲击			
如果删除源表文件会产生影响	腐化冰山桌	腐化冰山桌	对冰山桌子没有影响

功能	In-place 迁移 快照	In-place 迁移 migrate	完整数据迁移 CTAS 或 (创建表格 + 插入)
如果在源表位置添加新文件会产生影响	在新桌子上不可见 (需要将分区与合并 add_files)	在新桌子上不可见 (需要将分区与合并 add_files)	在新桌子上不可见 (需要 INSERT INTO 到新桌子)
成本	低	低	更高 (完整数据重写)
迁移速度	快速	快速	慢一点
可用于迁移到 Amazon S3 表	⊗否	⊗否	⊙是
需要手动 DDL	⊗否 (架构和分区是从源表中复制的)	⊗否 (架构和分区是从源表中复制的)	如果使用 CTAS，则只需要指定分区
最佳用途	无需重写数据即可快速迁移，允许并行使用 Hive 和 Iceberg 进行测试或逐步过渡。	当可以立即切换时，无需重写数据即可在原地替换 Hive 表。	通过数据重写实现全面的冰山优化。非常适合重新设计分区或架构，或者改善布局 and 性能。如果可能，请务必推荐。

优化 Apache Iceberg 工作负载的最佳实践

Iceberg 是一种表格格式，旨在简化数据湖管理并增强工作负载性能。不同的用例可能会优先考虑不同的方面，例如成本、读取性能、写入性能或数据保留，因此 Iceberg 提供了配置选项来管理这些权衡。本节提供了优化和微调 Iceberg 工作负载以满足您的要求的见解。

主题

- [一般最佳实践](#)
- [优化读取性能](#)
- [优化写入性能](#)
- [优化存储](#)
- [使用压缩来维护表格](#)
- [在 Amazon S3 中使用冰山工作负载](#)

一般最佳实践

无论您的用例如何，当您在上使用 Apache Iceberg 时 AWS，我们都建议您遵循这些一般的最佳实践。

- 使用 Iceberg 格式版本 2。

默认情况下，Athena 使用 Iceberg 格式版本 2。

[当您在 Amazon EMR 上使用 Spark 或 AWS Glue 创建 Iceberg 表时，请按照 Iceberg 文档中所述指定格式版本。](#)

- 使用 AWS Glue Data Catalog 作为您的数据目录。

Athena 默认使用 AWS Glue Data Catalog。

当您在 Amazon EMR 上使用 Spark 或 AWS Glue 使用 Iceberg 时，请在你的 Spark 会话中添加以下配置以使用。AWS Glue Data Catalog 有关更多信息，请参阅本指南 AWS Glue 前面的 [“Iceberg 的 Spark 配置”](#) 部分。

```
"spark.sql.catalog.<your_catalog_name>.type": "glue"
```

- 使用 a AWS Glue Data Catalog s 锁定管理器。

默认情况下，Athena 对 Iceberg 表使用 AWS Glue Data Catalog 作为锁管理器。

当您在 Amazon EMR 上使用 Spark 或 AWS Glue 使用 Iceberg 时，请务必将您的 Spark 会话配置为使用 AWS Glue Data Catalog 作为锁定管理器。有关更多信息，请参阅 Iceberg 文档中的[乐观锁定](#)。

- 使用 Zstandard (ZSTD) 压缩。

Iceberg 的默认压缩编解码器是 gzip，可以使用 table 属性对其进行修改。write.<file_type>.compression-codec Athena 已经使用 ZSTD 作为 Iceberg 表的默认压缩编解码器。

一般而言，我们建议使用 ZSTD 压缩编解码器，因为它可以在 GZIP 和 Snappy 之间取得平衡，并且在不影响压缩比的情况下提供良好的 read/write 性能。此外，还可以根据需要进行调整。有关更多信息，请参阅 Athena 文档中的[Athena 中的 ZSTD 压缩级别](#)。

Snappy 可能提供最佳的整体读取和写入性能，但压缩率低于 GZIP 和 ZSTD。如果您优先考虑性能，即使这意味着要在 Amazon S3 中存储更大的数据量，Snappy 也可能是最佳选择。

优化读取性能

本节讨论了可以调整的表属性，这些属性可以独立于引擎来优化读取性能。

分区

与 Hive 表一样，Iceberg 使用分区作为索引的主要层，以避免读取不必要的元数据文件和数据文件。列统计数据也被视为索引的辅助层，以进一步改进查询计划，从而缩短总体执行时间。

对您的数据进行分区

要减少查询 Iceberg 表时扫描的数据量，请选择与预期读取模式一致的平衡分区策略：

- 确定查询中经常使用的列。这些是理想的分区候选对象。例如，如果您通常查询特定日期的数据，则分区列的自然示例就是日期列。
- 选择低基数分区列，以避免创建过多的分区。分区过多会增加表中的文件数量，从而对查询性能产生负面影响。根据经验，“分区过多”可以定义为大多数分区中的数据大小小于设定值的 2-5 倍的情况。target-file-size-bytes

Note

如果您通常通过对高基数列（例如，可以包含数千个值的列）使用过滤器进行查询，请使用带有存储桶转换的 Iceberg 隐藏分区功能，如下一节所述。id

使用隐藏分区

如果您的查询通常根据表列的派生项进行筛选，请使用隐藏分区，而不是显式创建新列作为分区。有关此功能的更多信息，请参阅 [Iceberg 文档](#)。

例如，在具有时间戳列的数据集中（例如，2023-01-01 09:00:00），使用分区转换从时间戳中提取日期部分并即时创建这些分区，而不是使用解析后的日期创建新列（例如，2023-01-01）。

隐藏分区最常见的用例是：

- 当数据包含 @@ 时间戳列时，按日期或时间进行分区。Iceberg 提供多种变换来提取时间戳的日期或时间部分。
- 在 @@ 列的哈希函数上进行分区，此时分区列的基数很高，会导致分区过多。Iceberg 的存储桶转换通过在分区列上使用哈希函数，将多个分区值组合成更少的隐藏（存储桶）分区。

有关所有可用 [分区转换](#) 的概述，请参阅 Iceberg 文档中的分区转换。

通过使用常规 SQL 函数（如 `year()` 和 `month()`），用于隐藏分区的列可以成为查询谓词的一部分。`month()` 谓词也可以与诸如 `BETWEEN` 和 `AND` 之类的运算符组合使用。

Note

Iceberg 无法对产生不同数据类型的函数执行分区修剪；例如，`substring(event_time, 1, 10) = '2022-01-01'`

使用分区演变

当现有的 [分区策略不是最佳时](#)，请使用 [Iceberg 的分区演变](#)。例如，如果您选择的每小时分区结果太小（每个分区只有几兆字节），请考虑切换到每日或每月分区。

当最初不清楚表的最佳分区策略，并且您想在获得更多见解时完善分区策略时，可以使用这种方法。分区演变的另一个有效用途是，当数据量发生变化并且当前的分区策略随着时间的推移而变得不那么有效时。

有关如何演变分区的说明，请参阅 Iceberg 文档中的 [ALTER TABLE SQL 扩展](#)。

调整文件大小

优化查询性能包括最大限度地减少表中的小文件数量。为了获得良好的查询性能，我们通常建议保留大于 100 MB 的 Parquet 和 ORC 文件。

文件大小也会影响 Iceberg 表的查询计划。随着表中文件数量的增加，元数据文件的大小也随之增加。较大的元数据文件可能会导致查询计划变慢。因此，当表大小增加时，请增加文件大小以缓解元数据的指数级扩展。

使用以下最佳实践在 Iceberg 表中创建大小合适的文件。

设置目标文件和行组的大小

Iceberg 提供了以下用于调整数据文件布局的关键配置参数。我们建议您使用这些参数来设置目标文件大小和行组或行标大小。

参数	默认值	评论
<code>write.target-file-size-bytes</code>	512MB	此参数指定 Iceberg 将创建的最大文件大小。但是，写入某些文件的大小可能小于此限制。
<code>write.parquet.row-group-size-bytes</code>	128MB	Parquet 和 ORC 都将数据分块存储，因此引擎可以避免在某些操作中读取整个文件。
<code>write.orc.stripe-size-bytes</code>	64 MB	
<code>write.distribution-mode</code>	无，适用于 Iceberg 1.1 及更低版本	Iceberg 要求 Spark 在写入存储之前在其任务之间对数据进行排序。

参数

默认值

评论

哈希，从 Iceberg 1.2 版本开始

- 根据您的预期表格大小，请遵循以下一般准则：
 - 小表（最多几千兆字节）-将目标文件大小减至 128 MB。还要减小行组或条带的大小（例如，减至 8 或 16 MB）。
 - 中型到大型表（从几千兆字节到几百千兆字节不等）-默认值是这些表的良好起点。如果您的查询选择性很强，请调整行组或条带大小（例如，调整到 16 MB）。
 - 非常大的表（数百 GB 或 TB）-将目标文件大小增加到 1024 MB 或更多，如果您的查询通常会提取大量数据，请考虑增加行组或条带大小。
- 为确保写入 Iceberg 表的 Spark 应用程序创建大小合适的文件，请将该 `write.distribution-mode` 属性设置为 `hash` 或 `range`。有关这些模式之间差异的详细说明，请参阅 Iceberg 文档中的 [编写分发模式](#)。

以上仅为通用准则。我们建议您运行测试以确定最适合您的特定表和工作负载的值。

定期进行压实

上表中的配置设置了写入任务可以创建的最大文件大小，但不能保证文件会有该大小。为确保文件大小合适，请定期运行压缩，将小文件合并成较大的文件。有关运行压实的详细指导，请参阅本指南后面的 [冰山压实](#)。

优化列统计信息

Iceberg 使用列统计信息来执行文件修剪，从而通过减少查询扫描的数据量来提高查询性能。要从列统计数据中受益，请确保 Iceberg 收集查询过滤器中经常使用的所有列的统计信息。

默认情况下，根据表属性的 `write.metadata.metrics.max-inferred-column-defaults` 定义，Iceberg 仅收集 [每个表中前 100 列](#) 的统计数据。如果您的表包含超过 100 列，并且您的查询经常引用前 100 列之外的列（例如，您可能筛选第 132 列的查询），请确保 Iceberg 收集这些列的统计数据。有两种方法可以实现此目的：

- 创建 Iceberg 表时，对列进行重新排序，使查询所需的列位于设置的列范围内 `write.metadata.metrics.max-inferred-column-defaults`（默认为 100）。

注意：如果您不需要 100 列的统计数据，则可以将 `write.metadata.metrics.max-inferred-column-defaults` 配置调整为所需的值（例如 20），然后对列进行重新排序，使需要读取和写入查询的列位于数据集左侧的前 20 列之内。

- 如果您在查询筛选器中仅使用几列，则可以禁用指标收集的整体属性，并有选择地选择要为其收集统计数据的各个列，如以下示例所示：

```
.tableProperty("write.metadata.metrics.default", "none")
.tableProperty("write.metadata.metrics.column.my_col_a", "full")
.tableProperty("write.metadata.metrics.column.my_col_b", "full")
```

注意：当对这些列上的数据进行排序时，列统计信息最为有效。有关更多信息，请参阅本指南后面的[“设置排序顺序”](#)部分。

选择正确的更新策略

当您的用例可以接受较慢的写入操作时，使用写入时复制策略来优化读取性能。这是 Iceberg 使用的默认策略。

Copy-on-write 从而提高读取性能，因为文件是以读取优化的方式直接写入存储的。但是，与读时合并相比，每次写入操作花费的时间更长，消耗的计算资源也更多。这在读取和写入延迟之间进行了典型的权衡。通常，写入时复制非常适合将大多数更新并置在同一个表分区中的用例（例如，用于每日批量加载）。

Copy-on-write 配置（`write.update.modewrite.delete.mode`、和 `write.merge.mode`）可以在表级别进行设置，也可以在应用程序端独立设置。

使用 ZSTD 压缩

您可以使用 `table` 属性修改 Iceberg 使用的压缩编解码器。`write.<file_type>.compression-codec` 我们建议您使用 ZSTD 压缩编解码器来提高表的整体性能。

默认情况下，Iceberg 1.3 及更早版本使用 GZIP 压缩，与 ZSTD 相比，GZIP 压缩的 read/write 性能较慢。

注意：某些引擎可能使用不同的默认值。使用 At [hena](#) 或 [Amazon EMR 7.x 版本创建的 Iceberg 表就是这种情况](#)。

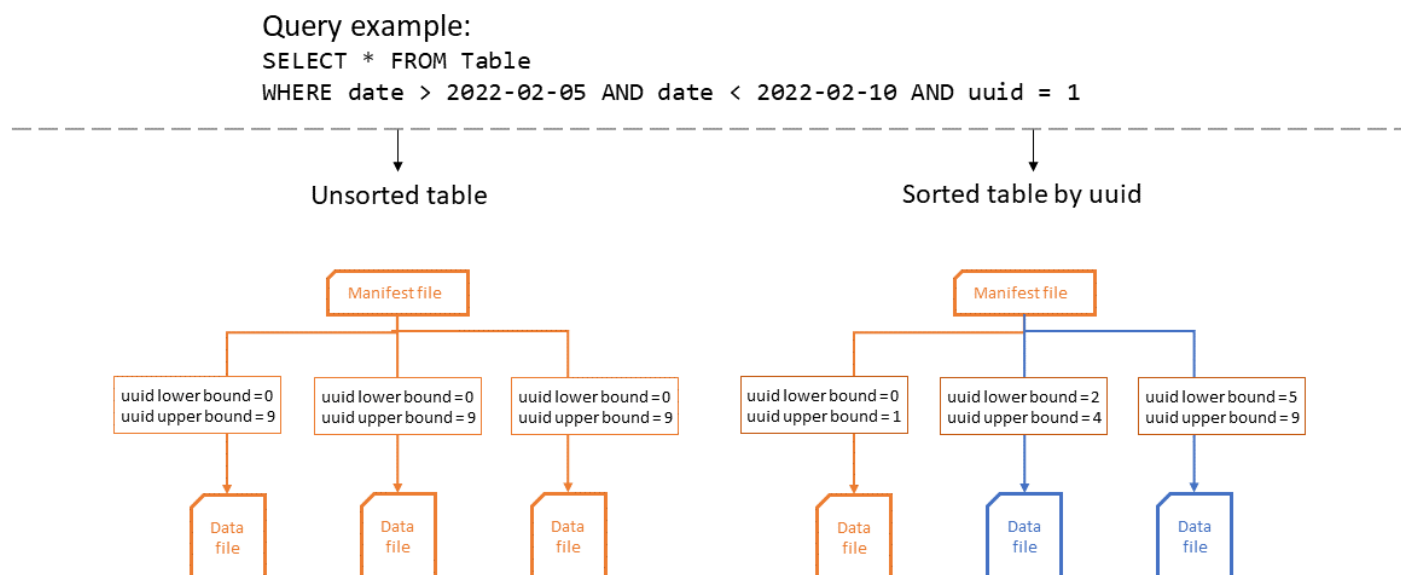
设置排序顺序

为了提高 Iceberg 表的读取性能，我们建议您根据查询过滤器中经常使用的一列或多列对表进行排序。排序与 Iceberg 的列统计信息相结合，可以显著提高文件修剪的效率，从而加快读取操作的速度。排序还可以减少使用查询筛选器中排序列的 Amazon S3 查询请求的数量。

通过使用 Spark 运行数据定义语言 (DDL) 语句，可以在表级别设置分层排序顺序。有关可用选项，请参阅 [Iceberg 文档](#)。设置排序顺序后，写入者会将此排序应用于 Iceberg 表中的后续数据写入操作。

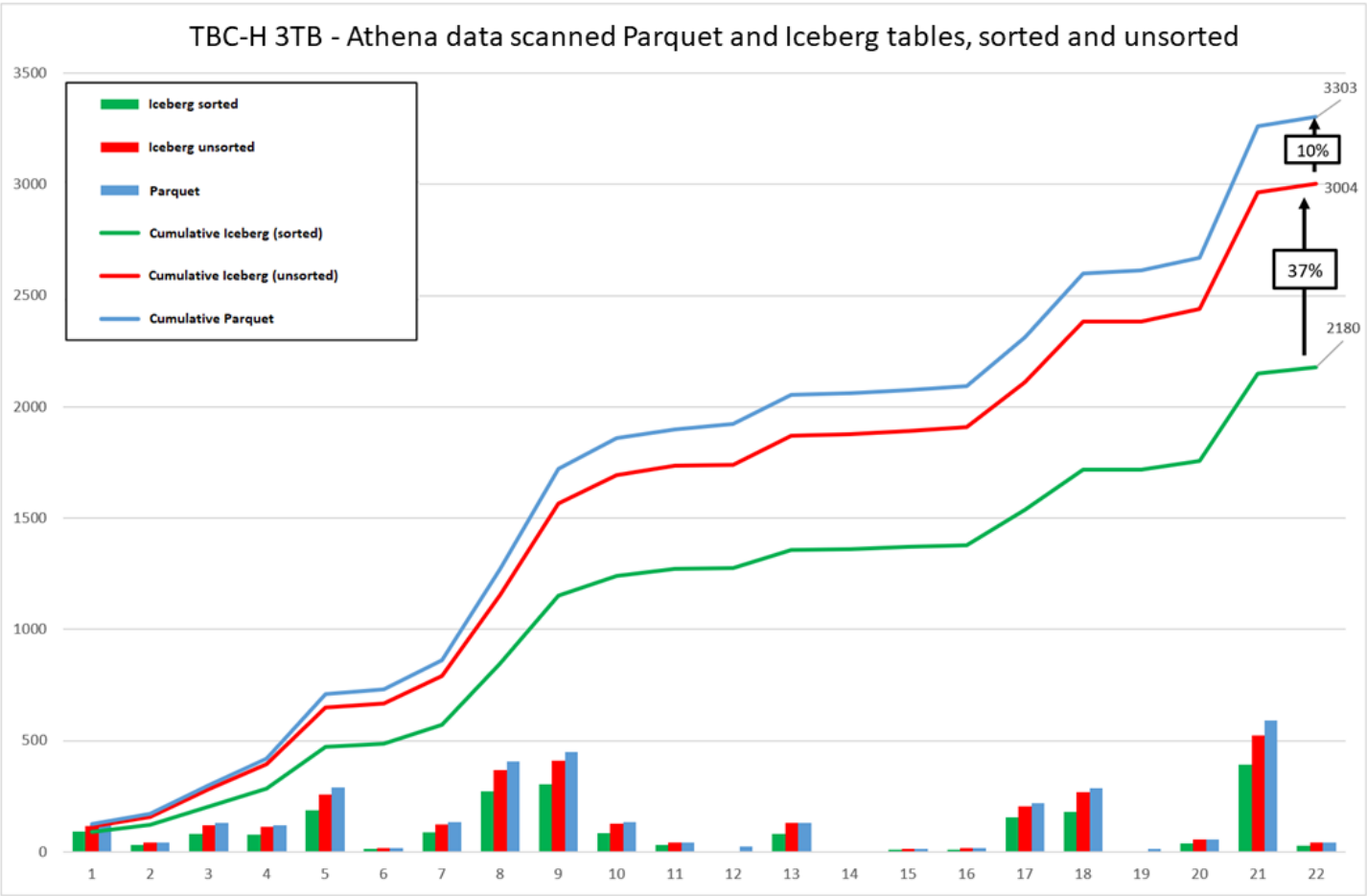
例如，在大多数查询按日期 (yyyy-mm-dd) 分区的表中 uuid，您可以使用 DDL 选项 `Write Distributed By Partition Locally Ordered` 来确保 Spark 写入范围不重叠的文件。

下图说明了对表进行排序时如何提高列统计数据的效率。在示例中，排序后的表只需要打开一个文件，并且可以最大限度地受益于 Iceberg 的分区和文件。在未排序的表中，任何数据文件中都 uuid 可能存在任何数据，因此查询必须打开所有数据文件。



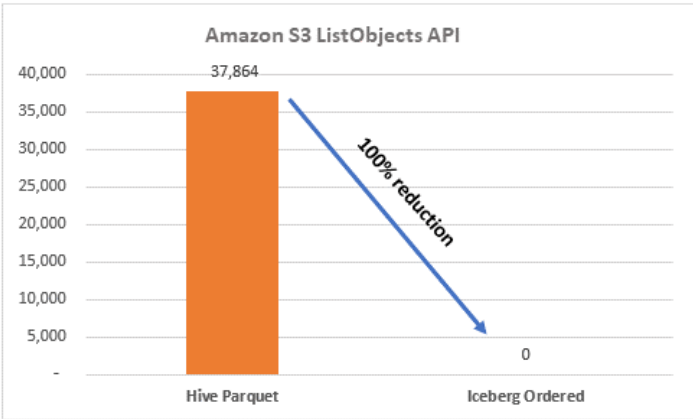
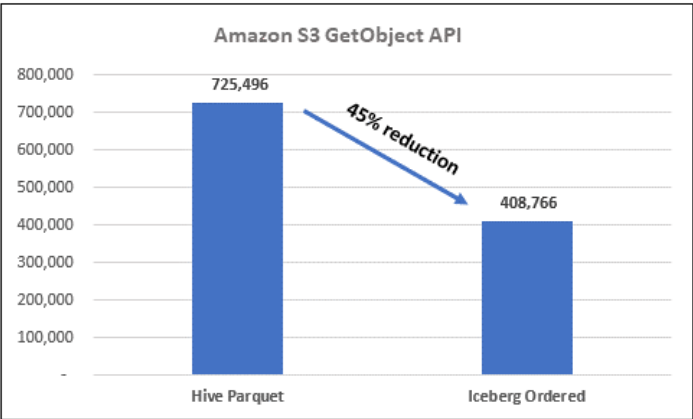
更改排序顺序不会影响现有的数据文件。您可以使用 Iceberg compaction 对它们应用排序顺序。

如下图所示，使用 Iceberg 排序的表格可能会降低工作负载成本。



这些图表汇总了运行Hive (Parquet) 表与冰山排序表的 TPC-H 基准测试的结果。但是，对于其他数据集或工作负载，结果可能会有所不同。

TPC-H 3TB - 22 queries



优化写入性能

本节讨论表属性，您可以调整这些属性以优化 Iceberg 表的写入性能，不受引擎影响。

设置表格分配模式

Iceberg 提供多种写入分配模式，用于定义写入数据在 Spark 任务中的分布方式。有关可用模式的概述，请参阅 Iceberg 文档中的[编写分发模式](#)。

对于优先考虑写入速度的用例，尤其是在流式工作负载中，`write.distribution-mode`请设置为`none`。这样可以确保 Iceberg 不会请求额外的 Spark 洗牌，并且可以在 Spark 任务中可用时写入数据。此模式特别适用于 Spark 结构化流媒体应用程序。

Note

将写入分配模式设置为`none`往往会生成大量小文件，从而降低读取性能。我们建议定期进行压缩，将这些小文件整合到大小合适的文件中，以提高查询性能。

选择正确的更新策略

当您的用例可以接受对最新数据执行较慢的读取操作时，使用读时合并策略来优化写入性能。

当您使用读时合并时，Iceberg 会将更新和删除内容作为单独的小文件写入存储空间。读取表格时，读者必须将这些更改与基础文件合并，以返回最新的数据视图。这会导致读取操作的性能下降，但会加快更新和删除的写入速度。通常，读时合并非常适合流式处理包含更新的工作负载，或者更新较少且分布在多个表分区中的作业。

可以在表级别设置读时合并配置（`write.update.mode`、`write.delete.mode`和`write.merge.mode`），也可以在应用程序端单独设置。

使用 `merge-on-read` 需要定期进行压缩，以防止读取性能随着时间的推移而下降。Compaction 可将更新和删除与现有数据文件进行协调，以创建一组新的数据文件，从而消除读取方面产生的性能损失。默认情况下，除非您将`delete-file-threshold`属性的默认值更改为较小的值，否则Iceberg的压缩不会合并删除文件（请参阅 [Iceberg](#) 文档）。要了解有关压实的更多信息，请参阅本指南后面的[冰山压实](#)部分。

选择正确的文件格式

Iceberg 支持以 Parquet、ORC 和 Avro 格式写入数据。镶木地板是默认格式。Parquet 和 ORC 是列式格式，可提供卓越的读取性能，但通常写入速度较慢。这代表了读取和写入性能之间的典型权衡。

如果写入速度对您的用例很重要，例如在流媒体工作负载中，请考虑在编写器的选项 Avro 中设置为 write-format，以 Avro 格式写入。由于 Avro 是一种基于行的格式，因此它以较慢的读取性能为代价提供更快的写入时间。

要提高读取性能，请定期进行压缩，将小 Avro 文件合并并转换为较大的 Parquet 文件。压实过程的结果受制于 write.format.default 表格设置。Iceberg 的默认格式是 Parquet，因此，如果你用 Avro 编写然后运行压缩，Iceberg 会将 Avro 文件转换为 Parquet 文件。示例如下：

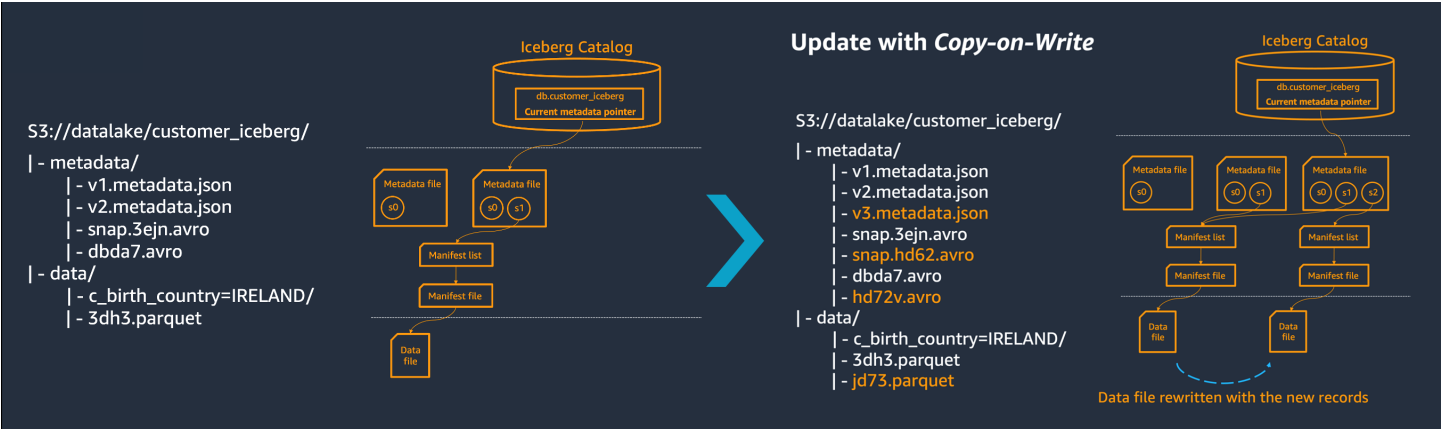
```
spark.sql(f"""
    CREATE TABLE IF NOT EXISTS glue_catalog.{DB_NAME}.{TABLE_NAME} (
        Col_1 float,
        <<<...other columns...>>
        ts timestamp)
    USING iceberg
    PARTITIONED BY (days(ts))
    OPTIONS (
        'format-version'='2',
        write.format.default='parquet')
""")

query = df \
    .writeStream \
    .format("iceberg") \
    .option("write-format", "avro") \
    .outputMode("append") \
    .trigger(processingTime='60 seconds') \
    .option("path", f"glue_catalog.{DB_NAME}.{TABLE_NAME}") \
    .option("checkpointLocation", f"s3://{BUCKET_NAME}/checkpoints/iceberg/")

    .start()
```

优化存储

更新或删除 Iceberg 表中的数据会增加数据的副本数量，如下图所示。运行压缩也是如此：它增加了 Amazon S3 中的数据副本数量。这是因为 Iceberg 将所有表的底层文件视为不可变的。



按照本节中的最佳实践来管理存储成本。

启用 S3 Intelligent-Tiering

当访问模式发生变化时，使用 [Amazon S3 Intelligent-Tiering](#) 存储类自动将数据移至最具成本效益的访问层。此选项没有操作开销或对性能的影响。

注意：请勿将 S3 中的可选层（例如存档访问权限和深度存档访问权限）Intelligent-Tiering 与 Iceberg 表一起使用。要存档数据，请参阅下一节中的指南。

您也可以使用 [Amazon S3 生命周期规则](#) 来设置自己的规则，以便将对象移动到另一个 Amazon S3 存储类别，例如 S3 Standard-IA 或 S3 One Zone-IA（请参阅 Amazon S3 文档中[支持的转换和相关限制](#)）。

存档或删除历史快照

对于向 Iceberg 表提交的每笔事务（插入、更新、合并、压缩），都会创建该表的新版本或快照。随着时间的推移，Amazon S3 中的版本数量和元数据文件数量会不断增加。

快照隔离、表回滚和时空旅行查询等功能需要保留表的快照。但是，存储成本会随着您保留的版本数量的增加而增加。

下表描述了您可以实施的设计模式，以根据您的数据保留要求管理成本。

设计模式	解决方案	使用案例
删除旧快照	使用 Athena 中的 VACUUM 语句 删除旧的快照。此操作不会产生任何计算成本。	这种方法会删除不再需要的快照以降低存储成本。您可以根

设计模式

解决方案

使用案例

为特定快照设置保留策略

- [或者，您可以在亚马逊 EMR 上使用 Spark 或 AWS Glue 删除快照。有关更多信息，请参阅 Iceberg 文档中的 `expire\_snapshots`。](#)

1. 在 Iceberg 中使用标签标记特定的快照并定义保留策略。有关更多信息，请参阅 Iceberg 文档中的[历史标签](#)。

例如，您可以在 Amazon EMR 上的 Spark 中使用以下 SQL 语句，每月保留一个快照，为期一年：

```
ALTER TABLE glue_catalog.db.table
CREATE TAG 'EOM-01' AS
OF VERSION 30 RETAIN
365 DAYS
```

2. 在 Amazon EMR 上使用 Spark 或 AWS Glue 删除剩余的未加标签的中间快照。

据数据保留要求配置应保留多少快照或保留多长时间。

此选项对快照执行硬删除。您无法回滚或穿越到过期的快照。

这种模式有助于遵守业务或法律要求，这些要求您显示过去给定时刻的表格状态。通过对特定的带标签的快照设置保留政策，您可以删除已创建的其他（未标记的）快照。这样，您无需保留创建的每个快照即可满足数据保留要求。

设计模式	解决方案	使用案例
存档旧快照	1. 使用 Amazon S3 标签用 Spark 标记对象。(Amazon S3 标签与 Iceberg 标签不同；有关更多信息，请参阅 Iceberg 文档。) 例如： <pre>spark.sql.catalog. my_catalog.s3.delete-enabled=false and \ spark.sql.catalog. my_catalog.s3.delete.tags.my_key=to_archive</pre> 2. 在 Amazon EMR 上使用 Spark 或 删除 AWS Glue 快照。当您使用示例中的设置时，此过程会标记对象并将其与 Iceberg 表元数据分离，而不是将其从 Amazon S3 中删除。 3. 使用 S3 生命周期规则将标记为 to_archive S3 Glacier 存储类 之一的对象过渡。 4. 要查询存档数据，请执行以下操作： • 恢复存档对象 (如果对象已转换为 Amazon Glacier 即时检索存储类别，则无需执行此步骤)。 • 使用 Iceberg 中的 register_table 过程 将快照注册为目录中的表。	这种模式允许您以较低的成本保留所有表版本和快照。 如果不先将这些版本恢复为新表，就无法穿越时空或回滚到存档的快照。出于审计目的，这通常是可以接受的。 您可以将此方法与以前的设计模式相结合，为特定快照设置保留策略。

设计模式

解决方案

使用案例

有关详细说明，请参阅 AWS 博客文章[提高在 Amazon S3 数据湖上构建的 Apache Iceberg 表的操作效率](#)。

删除孤立文件

在某些情况下，Iceberg 应用程序可能会在您提交事务之前失败。这会将数据文件留在 Amazon S3 中。由于没有提交，因此这些文件不会与任何表相关联，因此您可能需要异步清理它们。

要处理这些删除，您可以使用亚马逊 Athena 中的 `VACUUM` 语句。此语句删除快照并删除孤立文件。这非常具有成本效益，因为 Athena 不收取此操作的计算成本。此外，在使用该 `VACUUM` 语句时，您不必安排任何其他操作。

或者，您可以在 Amazon EMR 上使用 Spark 或 AWS Glue 运行该程序。`remove_orphan_files` 此操作会产生计算成本，并且必须单独进行计划。有关更多信息，请参阅 [Iceberg 文档](#)。

使用压缩来维护表格

Iceberg 包含的功能使您能够在向[表写入数据后执行表维护操作](#)。一些维护操作侧重于简化元数据文件，而另一些维护操作则增强了数据在文件中的聚集方式，以便查询引擎可以有效地找到响应用户请求所需的信息。本节重点介绍与压缩相关的优化。

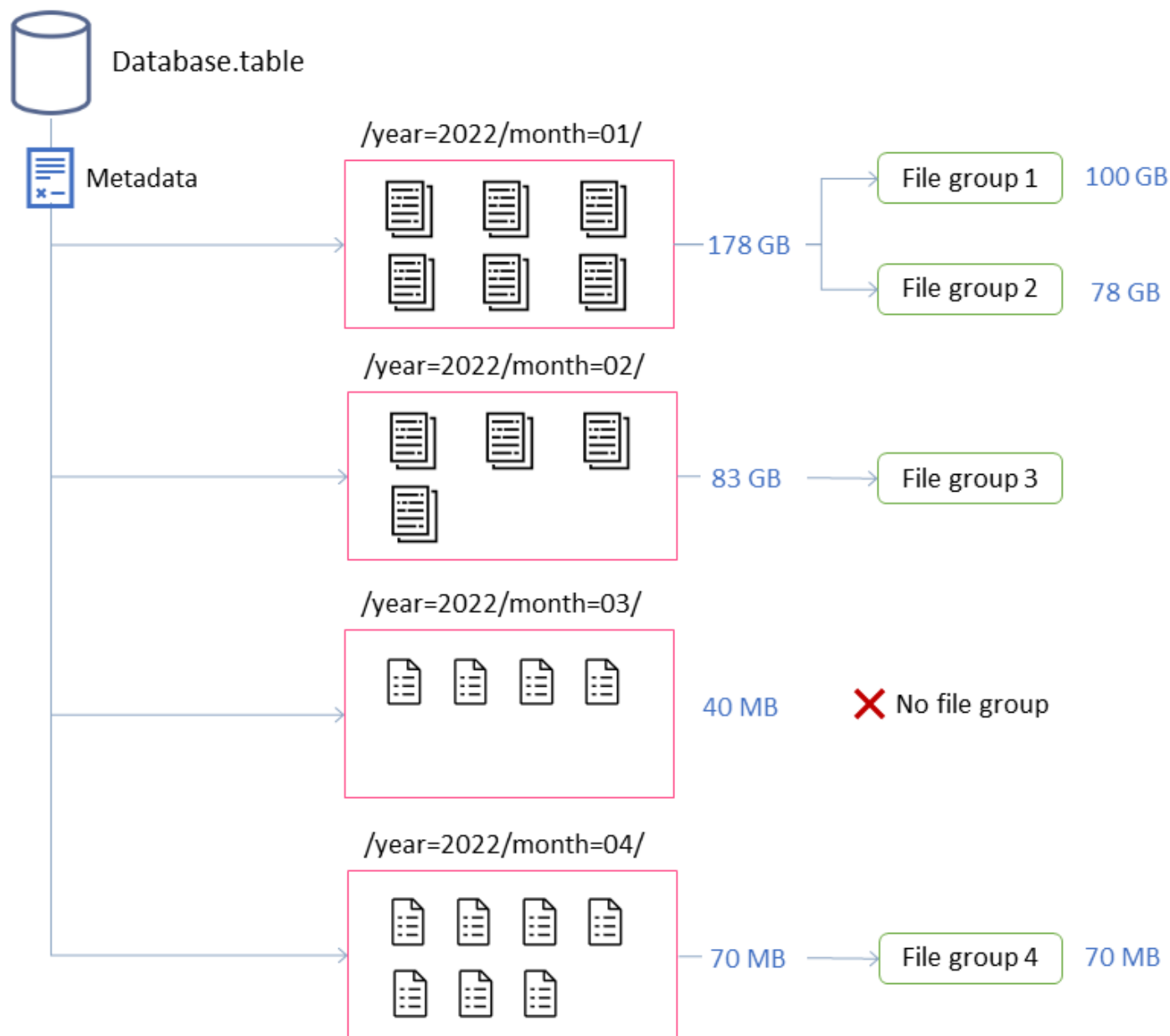
冰山压实

在 Iceberg 中，您可以使用压实来执行四项任务：

- 将小文件合并成通常超过 100 MB 的大文件。这种技术被称为垃圾箱包装。
- 将删除文件与数据文件合并。删除文件是由使用读时合并方法的更新或删除生成的。
- （重新）根据查询模式对数据进行排序。可以在没有任何排序顺序的情况下写入数据，也可以使用适合写入和更新的排序顺序写入数据。
- 使用空间填充曲线对数据进行聚类，以优化不同的查询模式，尤其是 z 顺序排序。

开启 AWS，您可以通过 Amazon Athena 或者在亚马逊 EMR 中使用 Spark 或。AWS Glue

使用 [rewrite_data_files](#) 过程运行压缩时，可以调整多个旋钮来控制压缩行为。下图显示了垃圾箱装箱的默认行为。了解垃圾箱压缩是理解分层排序和排 Z-order 序实现的关键，因为它们是垃圾箱打包接口的扩展，操作方式类似。主要区别在于对数据进行排序或聚类所需的额外步骤。



在此示例中，Iceberg 表由四个分区组成。每个分区的大小不同，文件数量也不同。如果您启动 Spark 应用程序来运行压缩，则该应用程序总共会创建四个要处理的文件组。文件组是 Iceberg 抽象，它表示将由单个 Spark 作业处理的一组文件。也就是说，运行压缩的 Spark 应用程序将创建四个 Spark 作业来处理数据。

调整压实行为

以下关键属性控制如何选择数据文件进行压缩：

- 默认情况下，[MAX_FILE_GROUP_SIZE_BYTES](#) 将单个文件组（Spark 作业）的数据限制设置为 [100 GB](#)。对于没有分区的表或分区跨越数百 GB 的表，此属性尤其重要。通过设置此限制，您可以分解操作以计划工作并取得进展，同时防止群集上的资源耗尽。
- 注意：每个文件组都是单独排序的。因此，如果要执行分区级排序，则必须调整此限制以匹配分区大小。
- [MIN_FILE_SIZE_BYTES](#) 或 [MIN_FILE_SIZE_DEFAULT_RATIO](#) 默认为在表级别设置的目标文件大小的 75%。例如，如果表的目标大小为 512 MB，则任何小于 384 MB 的文件都将包含在要压缩的文件集中。
- [MAX_FILE_SIZE_BYTES](#) 或 [MAX_FILE_SIZE_DEFAULT_RATIO](#) 默认为目标文件大小的 180%。与设置最小文件大小的两个属性一样，这些属性用于识别压缩作业的候选文件。
- [MIN_INP](#) UT_FILES 指定表分区大小小于目标文件大小时要压缩的最小文件数。此属性的值用于确定是否值得根据文件数（默认为 5）压缩文件。
- D@@ [DELETE_FILE_THRESHOLD](#) 指定文件在压缩中包含之前对其执行的最小删除操作次数。除非您另行指定，否则压缩不会将删除文件与数据文件合并。要启用此功能，必须使用此属性设置阈值。此阈值特定于单个数据文件，因此，如果将其设置为 3，则仅当有三个或更多引用数据文件的删除文件时，才会重写该数据文件。

这些属性使您可以深入了解上图中文件组的形成情况。

例如，标记为的分区 month=01 包括两个文件组，因为它超过了 100 GB 的最大大小限制。相比之下，该 month=02 分区包含单个文件组，因为它小于 100 GB。该 month=03 分区不满足默认的最低输入文件要求，即五个文件。因此，它不会被压实。最后，尽管该 month=04 分区包含的数据不足以形成所需大小的单个文件，但由于该分区包含五个以上的小文件，因此文件将被压缩。

您可以为在亚马逊 EMR 上运行的 Spark 设置这些参数，或者。AWS Glue 对于 Amazon Athena，您可以使用以前缀开头 [表的属性](#) 来管理类似的属性)。optimize_

在 Amazon EMR 上使用 Spark 运行压缩或 AWS Glue

本节介绍如何正确调整 Spark 集群的大小以运行 Iceberg 的压缩实用程序。以下示例使用 Amazon EMR Serverless，但您可以在 EC2 或 EKS 上的 Amazon EMR 中或中使用相同的方法。AWS Glue

您可以利用文件组和 Spark 作业之间的关联来规划群集资源。要按顺序处理文件组，考虑到每个文件组的最大大小为 100 GB，可以设置以下 [Spark](#) 属性：

- `spark.dynamicAllocation.enabled = FALSE`
- `spark.executor.memory = 20 GB`
- `spark.executor.instances = 5`

如果要加快压缩速度，则可以通过增加并行压缩的文件组的数量来横向扩展。您还可以使用手动或动态扩展来扩展 Amazon EMR。

- 手动缩放（例如，按系数 4）
 - `MAX_CONCURRENT_FILE_GROUP_REWRITES=4`（我们的因素）
 - `spark.executor.instances=5`（示例中使用的值） $\times 4$ （我们的因子） $= 20$
 - `spark.dynamicAllocation.enabled = FALSE`
- 动态缩放
 - `spark.dynamicAllocation.enabled=TRUE`（默认，无需执行任何操作）
 - `MAX_CONCURRENT_FILE_GROUP_REWRITES = N`（将此值与 `spark.dynamicAllocation.maxExecutors`，默认值为 100；根据示例中的执行器配置，您可以将其设置为 20） N

这些是帮助调整集群规模的指导方针。但是，您还应该监控 Spark 作业的性能，以找到最适合您的工作负载的设置。

使用亚马逊 Athena 进行压缩

[Athena 通过 OPTIMIZE 语句提供了 Iceberg 压缩实用程序作为托管功能的实现。](#) 您可以使用此语句来运行压缩，而不必评估基础架构。

此语句使用垃圾箱打包算法将小文件分组为较大的文件，并将删除文件与现有数据文件合并。要使用分层排序或 z 顺序排序对数据进行聚类，请在 Amazon EMR AWS Glue R 上使用 Spark 或。

可以在创建表时通过在 OPTIMIZE 语句中传递表属性来更改该语句的默认行为，也可以在创建表之后使用该 CREATE TABLE 语句来更改该 ALTER TABLE 语句的默认行为。有关默认值，请参阅 [Athena 文档](#)。

跑步压实的建议

使用案例

按计划运行垃圾箱压实

建议

- 如果您不知道OPTIMIZE表中包含多少小文件，请使用 Athena 中的语句。Athena 的定价模型基于扫描的数据，因此，如果没有要压缩的文件，则不会产生与这些操作相关的成本。为避免在 Athena 表上遇到超时，请按表分区OPTIMIZE运行。
- 如果您希望压缩大量小文件 AWS Glue，请使用 Amazon EMR 或动态扩展。

根据事件对垃圾箱进行装箱压实

- 如果您希望压缩大量小文件 AWS Glue，请使用 Amazon EMR 或动态扩展。

运行压缩来对数据进行排序

- 使用 Amazon EMR 或 AWS Glue，因为排序是一项昂贵的操作，可能需要将数据泄漏到磁盘。

运行压缩以使用 z 顺序排序对数据进行聚类

- 使用 Amazon EMR 或 AWS Glue，因为 z 顺序排序是一项非常昂贵的操作，可能需要将数据泄漏到磁盘。

在由于数据迟到而可能被其他应用程序更新的分区上运行压缩

- 使用 Amazon EMR 或。AWS Glue启用 Iceberg [部分_PROGRESS_ENABLED 属性](#)。使用此选项时，Iceberg 会将压缩输出拆分为多个提交。如果发生冲突（也就是说，如果在压缩运行时更新了数据文件），则此设置将重试限制为包含受影响文件的提交，从而降低了重试成本。否则，您可能需要重新压缩所有文件。

在冷分区（不再接收活动写入的数据分区）上运行压缩

- 使用 Amazon EMR 或。AWS Glue在此rewrite_data_files 过程中，指定排除主动写入分区的where谓词。这种策略可以

使用案例

建议

防止写入器和压缩作业之间的数据冲突，只留下 Iceberg 可以自动解决的元数据冲突。

在 Amazon S3 中使用冰山工作负载

本节讨论了可用于优化 Iceberg 与 Amazon S3 交互的 Iceberg 属性。

防止热分区 (HTTP 503 错误)

在 Amazon S3 上运行的某些数据湖应用程序可以处理数百万或数十亿个对象并处理数 PB 的数据。这可能导致前缀接收大量流量，通常通过 HTTP 503 (服务不可用) 错误检测到这些流量。为防止出现此问题，请使用以下 Iceberg 属性：

- 设置 `write.distribution-mode` 为 `range左hash右`，Iceberg 会写入大文件，从而减少 Amazon S3 请求。这是首选配置，应能解决大多数情况。
- 如果由于工作负载中有大量数据而继续出现 503 错误，则可以在 Iceberg `true` 中 `write.object-storage.enabled` 将其设置为。这指示 Iceberg 对对象名称进行哈希处理，并将负载分配给多个随机的 Amazon S3 前缀。

有关这些属性的更多信息，请参阅 Iceberg 文档中的[写入属性](#)。

使用 Iceberg 维护操作来释放未使用的数据

要管理 Iceberg 表，您可以使用 Iceberg 核心 API、Iceberg 客户端 (例如 Spark) 或托管服务，例如亚马逊 Athena。要从 Amazon S3 中删除旧文件或未使用的文件，我们建议您仅使用 Iceberg 原生 API 来[删除快照](#)、[移除旧的元数据文件](#)和[删除孤立](#)文件。

通过 Boto3、亚马逊 S3 软件开发工具包或 AWS Command Line Interface (AWS CLI) 使用 Amazon S3 API，或者使用任何其他非 Iceberg 方法覆盖或删除 Iceberg 表的 Amazon S3 文件会导致表损坏和查询失败。

跨复制数据 AWS 区域

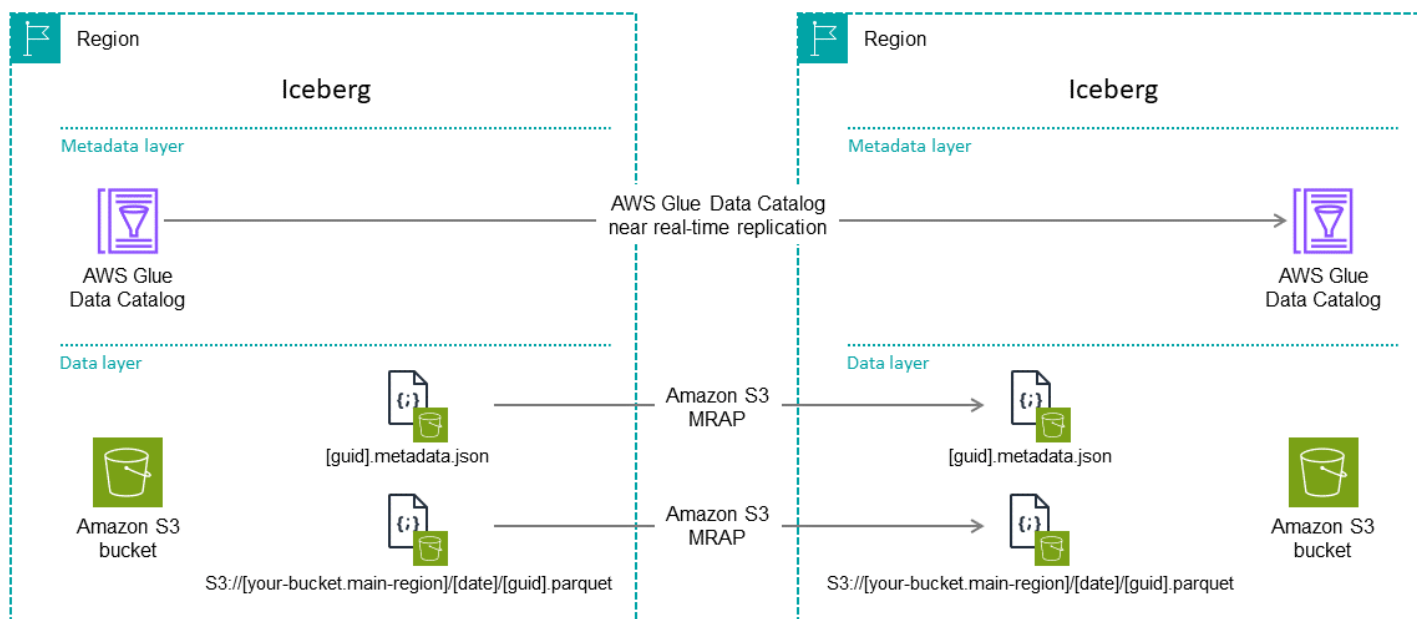
在 Amazon S3 中存储 Iceberg 表时，您可以使用 Amazon S3 中的内置功能，例如[Cross-Region 复制 \(CRR\)](#) 和 [Multi-Region 接入点 \(MRAP\)](#)，在多个表之间复制数据。AWS 区域 MRAP 为应用程序提供了一个全局端点，用于访问位于多个存储桶中的 S3 存储桶。AWS 区域 Iceberg 不支持相对路

径，但您可以使用 MRAP 通过将存储桶映射到接入点来执行 Amazon S3 操作。MRAP 还与 Amazon S3 Cross-Region 复制过程无缝集成，这会导致长达 15 分钟的延迟。您必须同时复制数据和元数据文件。

⚠ Important

目前，Iceberg 与 MRAP 的集成仅适用于 Apache Spark。如果您需要故障转移到辅助服务器 AWS 区域，则必须计划将用户查询重定向到故障转移区域中的 Spark SQL 环境（例如 Amazon EMR）。

CRR 和 MRAP 功能可帮助您为 Iceberg 表构建跨区域复制解决方案，如下图所示。



要设置此跨区域复制架构，请执行以下操作：

1. 使用 MRAP 位置创建表。这样可以确保 Iceberg 元数据文件指向 MRAP 位置而不是物理存储桶位置。
2. 使用 Amazon S3 MRAP 复制 Iceberg 文件。MRAP 支持数据复制，服务级别协议 (SLA) 为 15 分钟。Iceberg 可防止读取操作在复制过程中引入不一致。
3. 使表在辅助区域 AWS Glue Data Catalog 中可用。可从两个选项中选择：
 - 使用 AWS Glue Data Catalog 复制设置用于复制 Iceberg 表元数据的管道。此实用程序在 GitHub [Glue Catalog 和 Lake Formation 权限复制](#) 存储库中可用。这种事件驱动的机制根据事件日志复制目标区域中的表。

- 当您需要进行故障转移时，请在辅助区域中注册表。对于此选项，您可以使用之前的实用程序或 Iceberg regi [ster_table 过程](#) 并将其指向最新的文件。metadata.json

监控 Apache 冰山工作负载

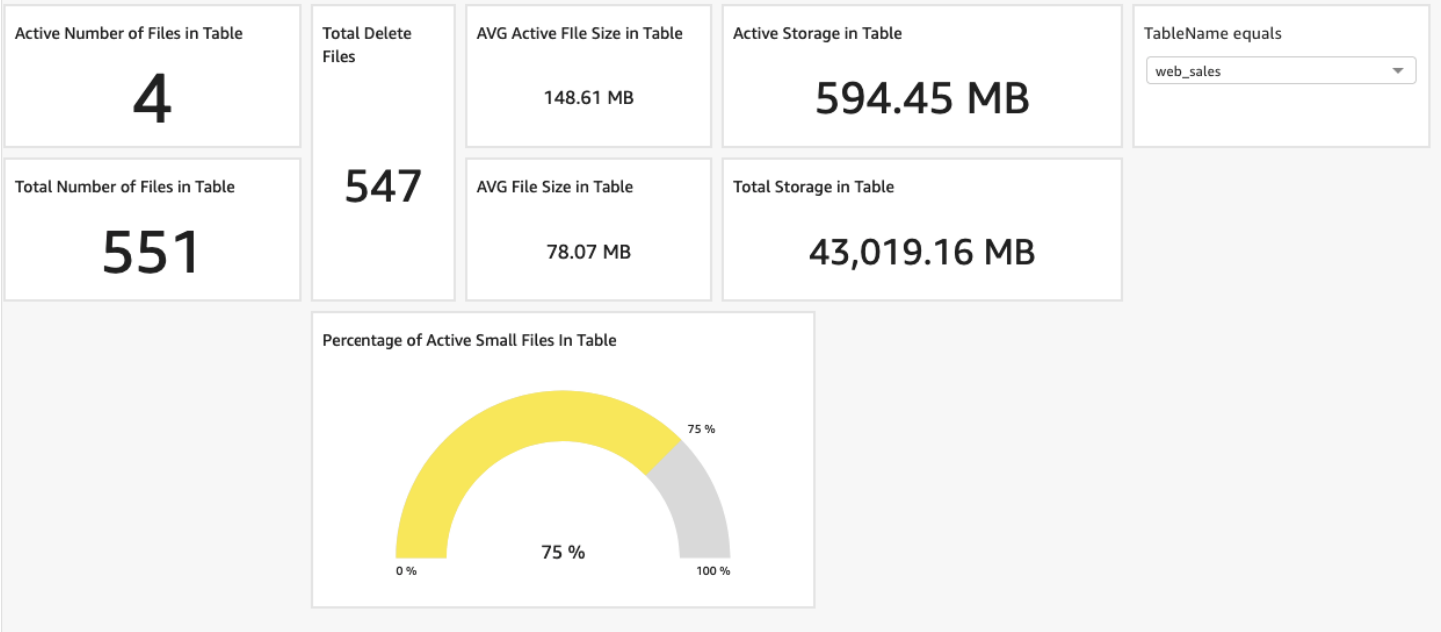
要监控 Iceberg 工作负载，您有两种选择：分析[元数据表](#)或使用[指标报告器](#)。指标报告器在 Iceberg 版本 1.2 中引入，仅适用于 REST 和 JDBC 目录。

如果您正在使用 AWS Glue Data Catalog，则可以通过在 Iceberg 公开的元数据表上设置监控来深入了解 Iceberg 表的运行状况。

监控对于性能管理和故障排除至关重要。例如，当 Iceberg 表中的分区达到一定比例的小文件时，您的工作负载可以启动压缩作业，将这些文件整合为更大的文件。这样可以防止查询速度超过可接受的水平。

Table-level 监控

以下屏幕显示了在 Amazon Quick Sight 中创建的表格监控控制面板。此仪表板使用 Spark SQL 查询 Iceberg 元数据表，并捕获活动文件数量和总存储空间等详细指标。然后，这些信息存储在 AWS Glue 表中以供操作。最后，使用 Amazon Athena 创建了 Quick Sight 控制面板，如下图所示。这些信息可帮助您识别和解决系统中的特定问题。



示例 Quick Sight 仪表板收集了 Iceberg 表的以下关键绩效指标 (KPI)：

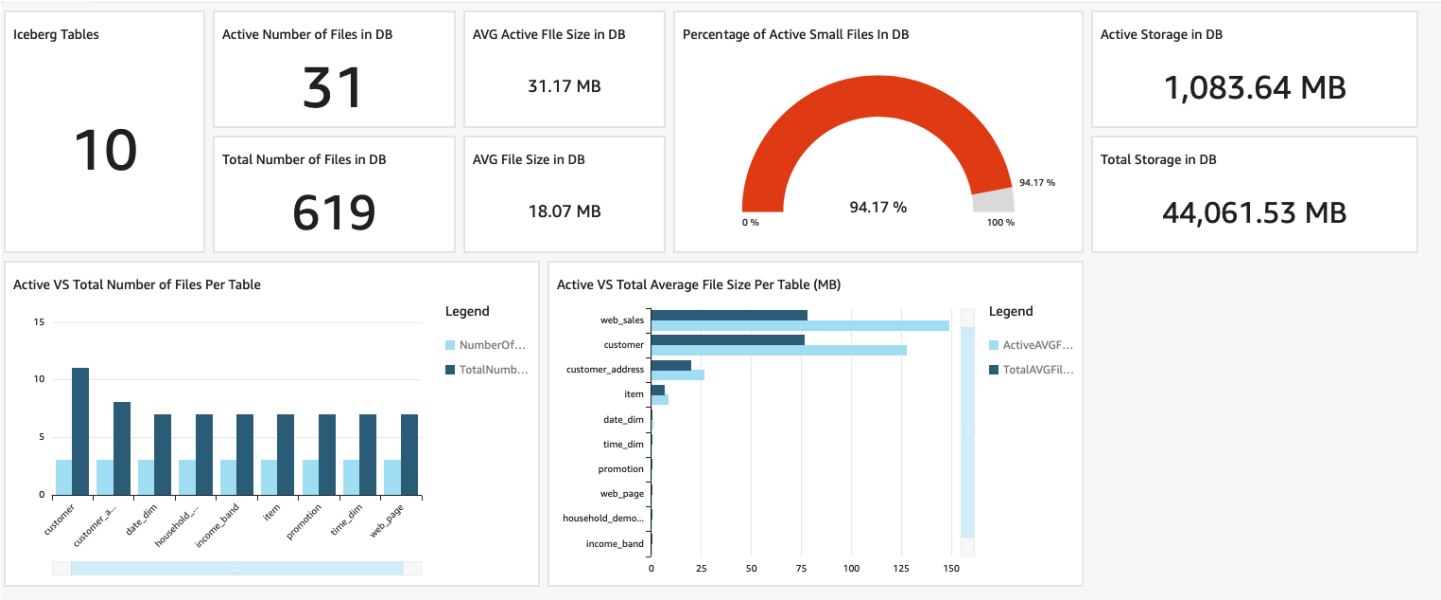
关键绩效指标	描述	Query
文件数量	Iceberg 表中的文件数 (适用于所有快照)	<pre>select count(*) from <catalog.database. table_name>.all_files</pre>
活动文件数	冰山表最后一次快照中的活动文件数	<pre>select count(*) from <catalog.database. table_name>.files</pre>
平均文件大小	Iceberg 表中所有文件的平均文件大小 (以兆字节为单位)	<pre>select avg(file_ size_in_bytes)/100 0000 from <catalog.database. table_name>.all_files</pre>
平均活动文件大小	Iceberg 表中活动文件的平均文件大小 (以兆字节为单位)	<pre>select avg(file_ size_in_bytes)/100 0000 from <catalog.database. table_name>.files</pre>
小文件的百分比	小于 100 MB 的活动文件所占的百分比	<pre>select cast(sum(case when file_size _in_bytes < 100000000 then 1 else 0 end)*100/ count(*) as decimal(1 0,2)) from <catalog.database. table_name>.files</pre>
总存储大小	表中所有文件的总大小，不包括孤立文件和 Amazon S3 对象版本 (如果启用)	<pre>select sum(file_ size_in_bytes)/100 0000 from <catalog.database. table_name>.all_files</pre>

关键绩效指标	描述	Query
活动存储空间总大小	给定表的当前快照中所有文件的总大小	<pre>select sum(file_size_in_bytes)/1000000 from <catalog.database.table_name>.files</pre>

有关创建仪表板的更多信息，请参阅 [Quick Sight 文档](#)。

Database-level 监控

以下示例显示了在 Quick Sight 中创建的监控仪表板，用于概述一组 Iceberg 表的数据库级 KPI。



此控制面板收集以下 KPI：

关键绩效指标	描述	Query
文件数量	Iceberg 数据库中的文件数（适用于所有快照）	此仪表板使用上一节中提供的表格级查询并合并结果。

关键绩效指标	描述	Query
活动文件数	Iceberg 数据库中活动文件的数量（基于 Iceberg 表的最后一次快照）	
平均文件大小	Iceberg 数据库中所有文件的平均文件大小（以兆字节为单位）	
平均活动文件大小	Iceberg 数据库中所有活动文件的平均文件大小（以兆字节为单位）	
小文件的百分比	冰山数据库中小于 100 MB 的活动文件的百分比	
存储空间总大小	数据库中所有文件的总大小，不包括孤立文件和 Amazon S3 对象版本（如果启用）	
活动存储空间总大小	数据库中所有表的当前快照中所有文件的总大小	

预防性维护

通过设置前几节中讨论的监控功能，您可以从预防性角度而不是被动角度进行表维护。例如，您可以使用表级和数据库级指标来计划诸如以下的操作：

- 当表格达到 N 个小文件时，使用垃圾箱压缩对小文件进行分组。
- 当表达到 N 个给定分区中的删除文件时，使用 bin 打包压缩合并删除文件。
- 当总存储空间比活动存储量高 X 倍时，通过移除快照来移除已经压缩的小文件。

Apache Iceberg 的治理和访问控制 AWS

Apache Iceberg 与集成 AWS Lake Formation 以简化数据治理。这种集成允许数据湖管理员为 Iceberg 表分配单元级别的访问权限。有关使用 Amazon Athena 和查询 Iceberg 表的示例，AWS 请参阅[博客文章使用 AWS Lake Formation Amazon Athena 与 Apache Iceberg 表交互以及使用跨账户细粒度权限进行交互](#)。AWS Lake Formation

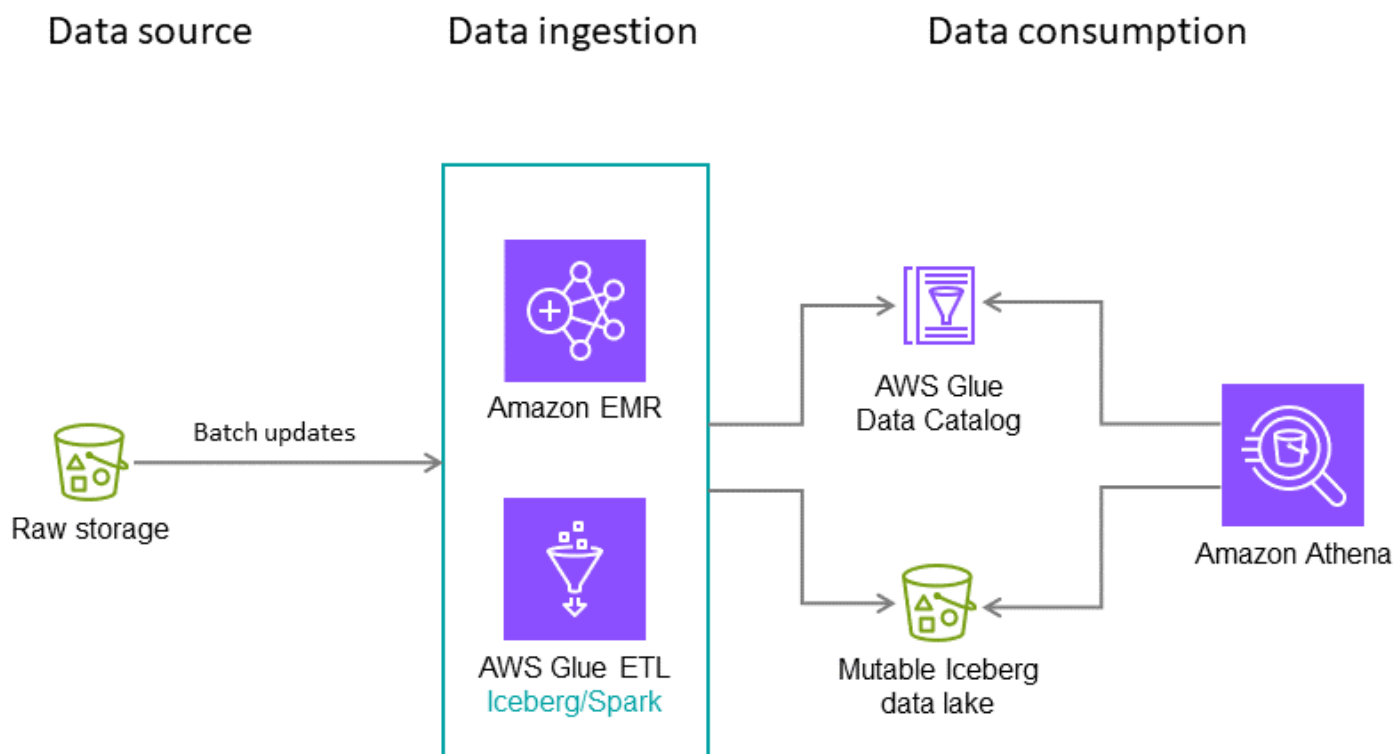
Apache Iceberg 的参考架构 AWS

本节提供了如何在不同用例中应用最佳实践的示例，例如批量摄取和将批量数据摄取和流式数据摄取相结合的数据湖。

每晚批量摄取

对于这个假设的用例，假设你的 Iceberg 桌每晚都会提取信用卡交易。每个批次仅包含增量更新，必须将其合并到目标表中。每年收到几次完整的历史数据。对于这种情况，我们建议使用以下架构和配置。

注意：这只是一个示例。最佳配置取决于您的数据和要求。



建议

- 文件大小：128 MB，因为 Apache Spark 任务以 128 MB 的区块处理数据。
- 写入类型：写时复制。如本指南前面所述，这种方法有助于确保以读取优化的方式写入数据。
- 分区变量：year/month/day。在我们的假设用例中，我们查询最新数据的频率最高，尽管我们偶尔会对过去两年的数据进行全表扫描。分区的目标是根据用例的要求推动快速读取操作。
- 排序顺序：时间戳

- 数据目录：AWS Glue Data Catalog

结合了批量和近实时摄取的数据湖

您可以在 Amazon S3 上配置数据湖，以便跨账户和地区共享批处理和流式传输数据。有关架构图和详细信息，请参阅 AWS 博客文章使用 [Apache Iceberg 构建交易数据湖](#)，[AWS Glue 以及使用 Amazon AWS Lake Formation Athena 进行跨账户数据共享](#)。

资源

- [在 AWS Glue \(AWS Glue 文档 \) 中使用 Iceberg 框架](#)
- [Iceberg](#) (亚马逊 EMR 文档)
- [使用 Apache Iceberg 表 \(亚马逊 Athena 文档 \)](#)
- [Amazon S3 文档](#)
- [快速文档](#)
- [Glue Catalog 和 Lake Formation 权限复制](#) (GitHub 存储库)
- [Apache Iceberg 文档](#)
- [Apache Spark 文档](#)

贡献者

以下人员撰 AWS 写、共同撰写和审阅了本指南。

贡献者

- Stefano Sandona，大数据解决方案架构师
- Imtiaz (Taz) Sayed，分析解决方案架构师技术负责人
- Shana Schippers，大数据解决方案架构师
- Prashant Singh，Amazon EMR 软件开发工程师
- Arun A K，大数据和 ETL 解决方案架构师
- 弗朗西斯科·莫里洛，流媒体解决方案架构师
- Suthan Phillips，分析架构师，亚马逊 EMR
- Sercan Karaoglu，解决方案架构师
- Yonatan Dolan，分析专家
- 盖伊·巴查尔，解决方案架构师
- Sofia Zilberman，流媒体解决方案架构师
- Dan Stair，专业解决方案架构师
- Sakti Mishra，解决方案架构师
- Ron Ortloff，亚马逊 S3 首席产品经理
- 张大卫，分析解决方案架构师

审稿人

- Rick Sears，亚马逊 EMR 总经理
- Linda OConnor，亚马逊 EMR 专家
- Ian Meyers，亚马逊 EMR 总监
- Vinita Ananth，亚马逊 EMR 产品管理总监
- 杰森·伯科维茨，产品经理 AWS Lake Formation
- Mahesh Mishra，亚马逊 Redshift 产品经理
- 弗拉基米尔·兹拉特金，大数据解决方案架构经理
- Karthik Prabhakar，分析架构师，亚马逊 EMR

- Vijay Jain , 产品经理
- Anupriti Warade , 亚马逊 S3 产品经理
- Ajit Tandale , 解决方案架构师, 数据
- Gwen Chen , 产品营销经理
- 关山则隆 , 首席架构师, 大数据

文档历史记录

下表介绍了本指南的一些重要更改。如果您希望收到有关未来更新的通知，可以订阅 [RSS 源](#)。

变更	说明	日期
新章节	添加了有关 使用 Iceberg 表格格式规范版本 3 的信息。	2025 年 11 月 26 日
更正	更正了 快照过程 部分中有关gc.enabled 设置的信息。	2025 年 10 月 24 日
补充	添加了有关使用 Trino 和处理 Iceberg 表的新章节 Pylceberg ，并扩展了有关 将表迁移到 Iceberg 的部分。	2025 年 8 月 12 日
更新	对整个指南中的信息进行了增强和澄清，以反映最新版本的 Amazon EMR 和 Apache Iceberg。AWS Glue	2025 年 7 月 14 日
补充	添加了有关使用 Amazon Data Firehose 处理 Iceberg 表的 新章节 。	2025 年 2 月 20 日
初次发布	—	2024 年 4 月 30 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。