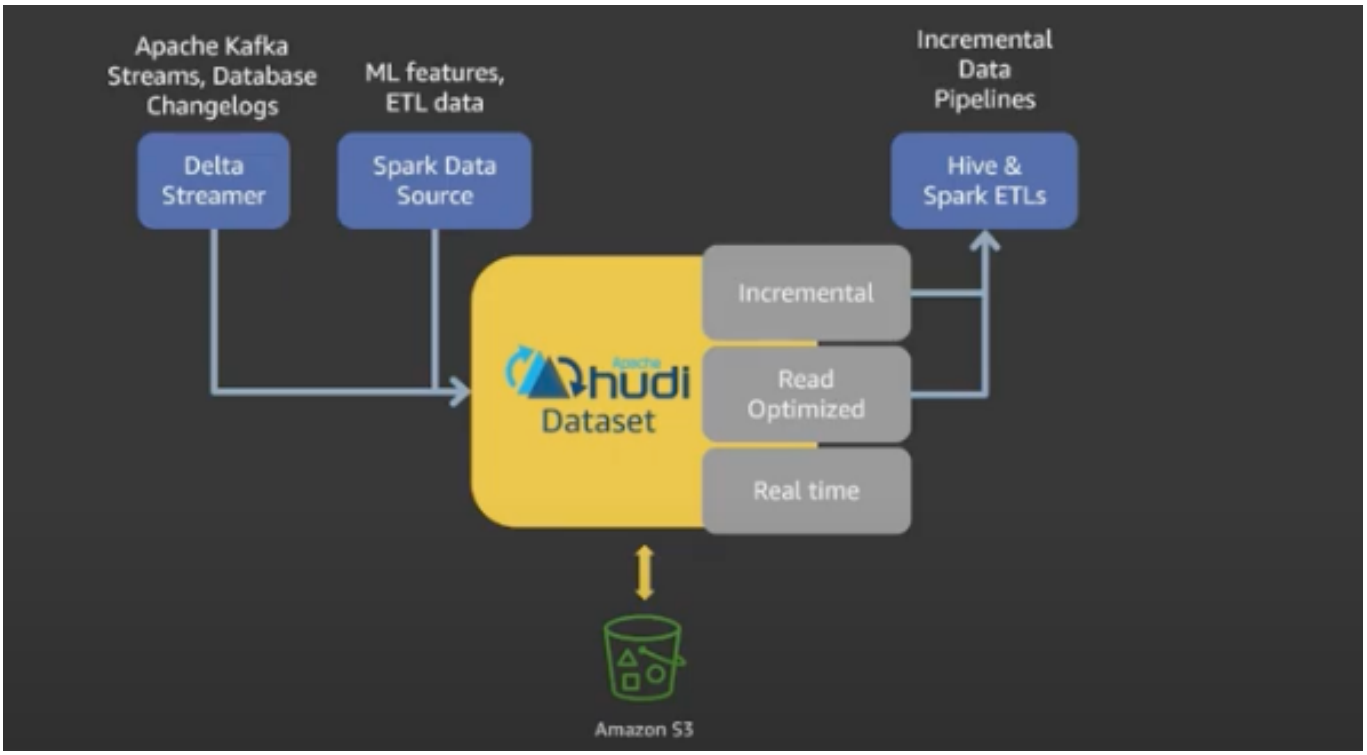
Apache Hudi

Apache Hudi and Data Privacy

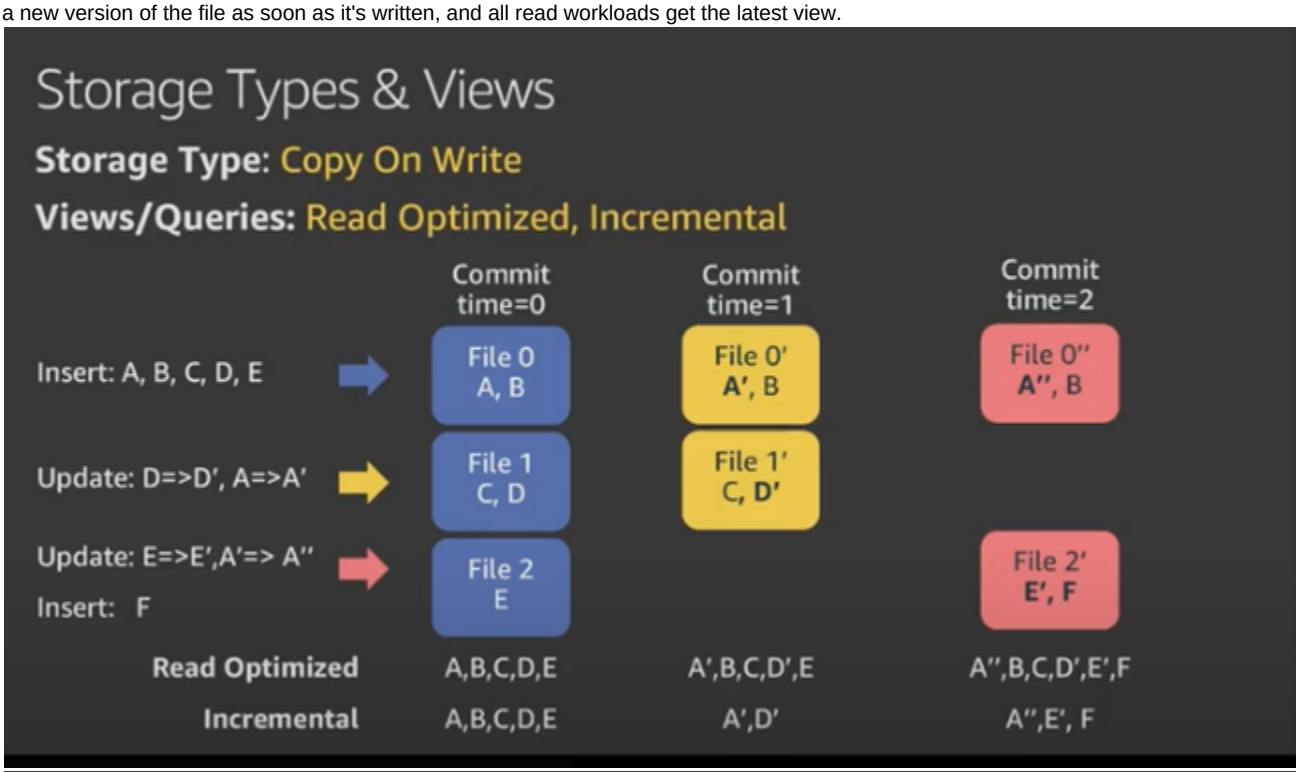


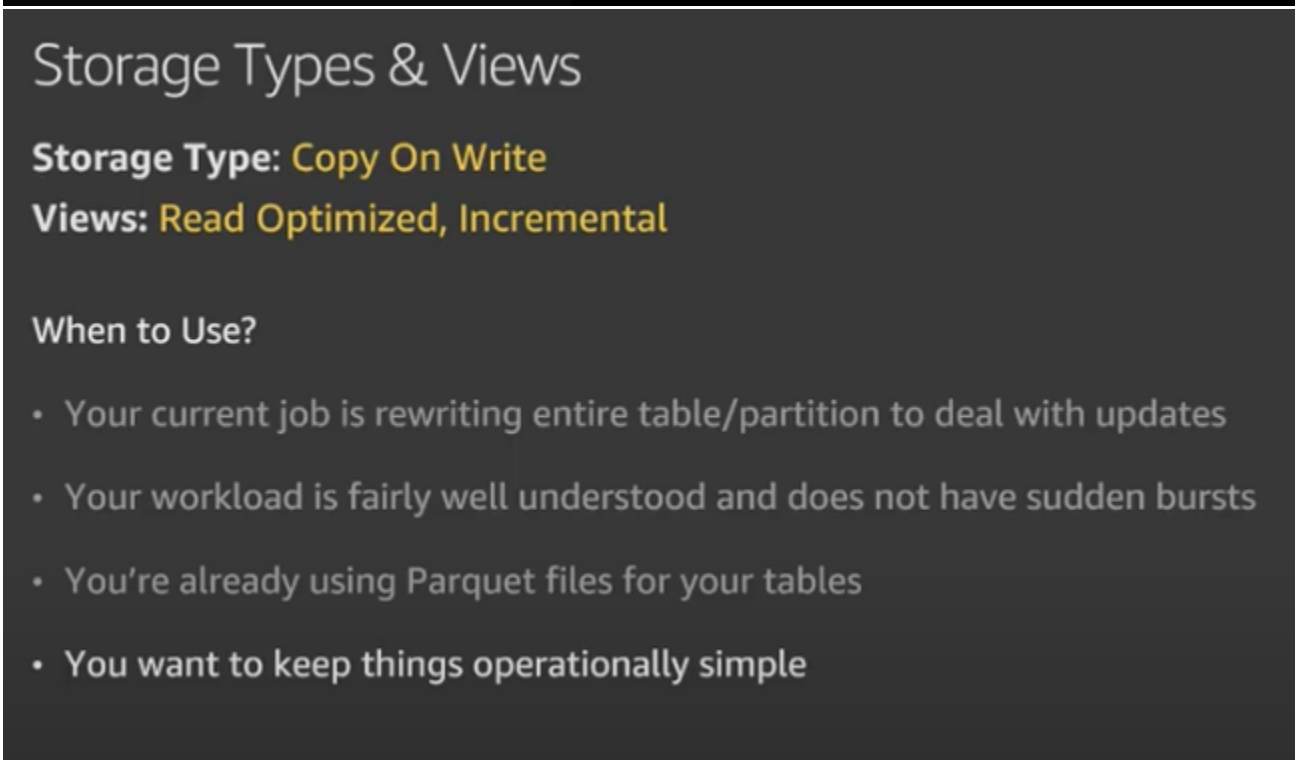
- Hudi is designed to behave like an RDBMS for writes and Bigdata for reads
- **Migration -** Hudi supports 2 modes when migrating/bootstrapping parquet tables.
    - **METADATA_ONLY** : In this mode, record level metadata alone is generated for each source record and stored in new bootstrap location.
    - **FULL_RECORD** : In this mode, record level metadata is generated for each source record and both original record and metadata for each record copied.
- Hudi is acronym for **H**adoop **U**pserts an**D** **I**ncrementals
- EMR 5.28 includes Hudi 0.5.0 and is is compatible with spark hive and presto. Support hive metastore and aws glue catalog for storing metadata
- **How can Hudi help**
    - Data privacy law compliance
    - Consuming real time data streams and applying CDC - capture data change logs, with databases
    - Reinstating late arriving data
    - Tracking data changes and rollback

Storage Types

1. Copy On Write
2. Merge On Read

Copy On Write(CoW)

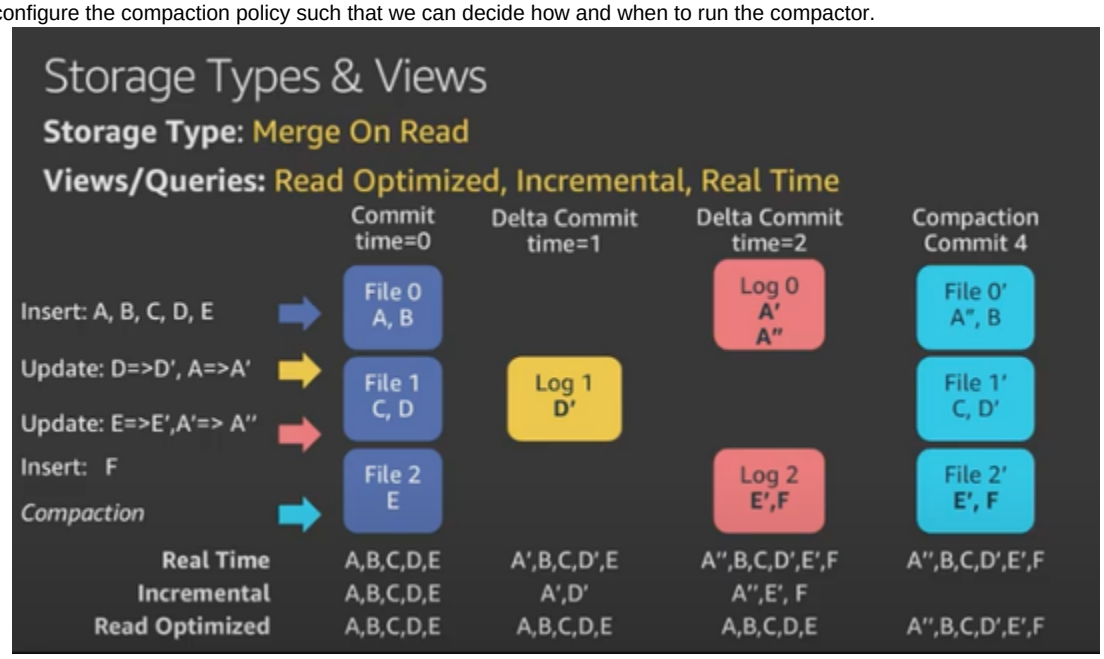
- This is the default storage type, which creates a new version of the file and stores the output in Parquet format. This is useful when you want to have the UPSERT version ready as soon as the new data is written. This is **great for read-heavy workloads** as you can create a new version of the file as soon as it's written, and all read workloads get the latest view.
- 

- 

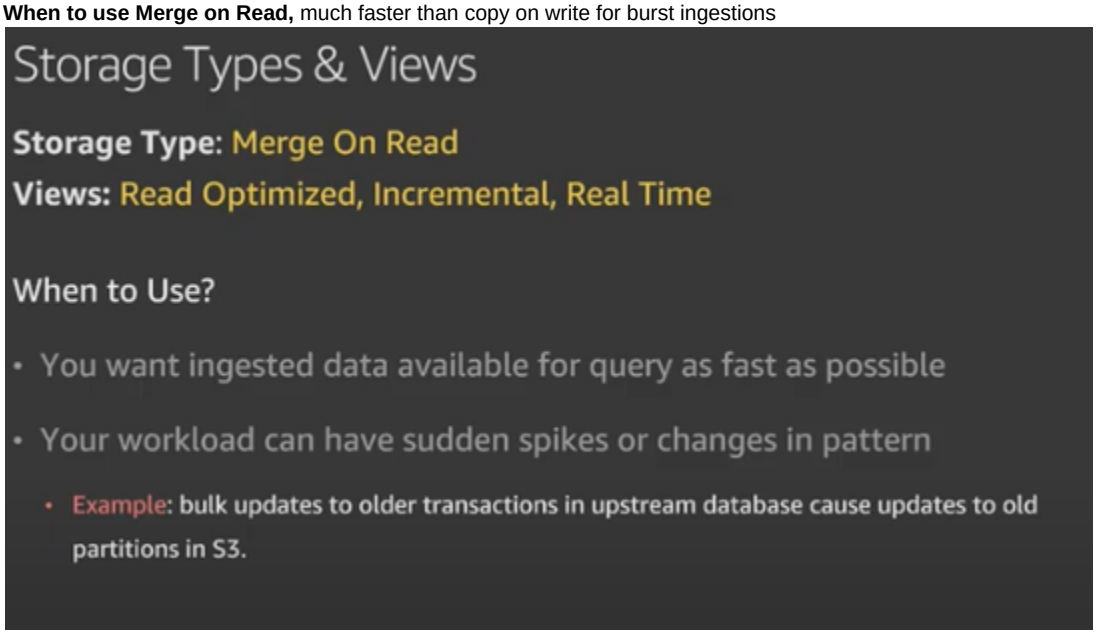
- If you have burst writes, then COW is not best choice as it will create lots of file versions. See above File 0, has 3 version - File 0, File 0' and File 0''
- Based on the hudi cleanup policy applied File0, File 0' are purged.

Merge On Read(MoR)

- This storage type is helpful for write-heavy workloads, where the merging does not happen during the write process but instead happens on demand when a read request comes in. This stores data in a combination of Parquet and row-based Avro formats. **Each new update creates a row-based incremental delta file and is compacted when needed to create a new version of the file in Parquet format.** You can configure the compaction policy such that we can decide how and when to run the compactor.



- **When to use Merge on Read,** much faster than copy on write for burst ingestions
-



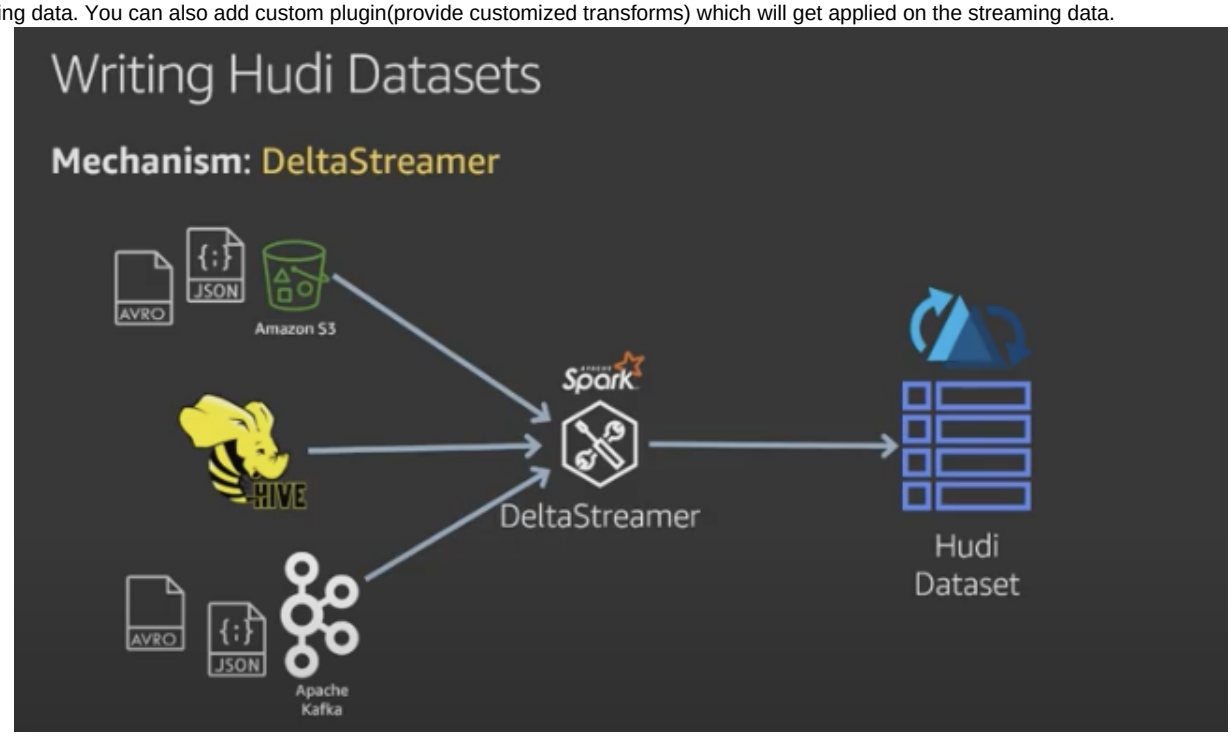
Hudi Logical Views

- **Read optimized view**: This includes the latest compacted data from MoR tables and the latest committed data from CoW tables. This view does not include the delta files that are not committed or compacted yet.
- **Incremental view**: This is helpful for downstream ETL jobs as it provides an incremental change view from CoW tables.
- **Real-time view**: This view is helpful when you plan to access the latest copy of data, which merges the columnar Parquet files and the row-based Avro delta files. Only with possible with **MoR.**

Writing Hudi Datasets

- **DeltaStreamer** - listens to Avro or Json files in S3 and triggers a DeltaStreamer, similarly it can listen to events on Kafka topic either in Avro or Json format and stream these changes to hoodie dataset. You may want use it as self managed ingestion tool with automatic

compactions and checkpointing. As it does constant check pointing, you can start and stop your Delta Streamer and it would start/catch-up from where the delta streamer left off as it does frequent checkpointing. You can also apply sql based transformation on the streaming data. You can also add custom plugin(provide customized transforms) which will get applied on the streaming data.

- 

Writing Pyspark Dataframe as a Hudi datasets

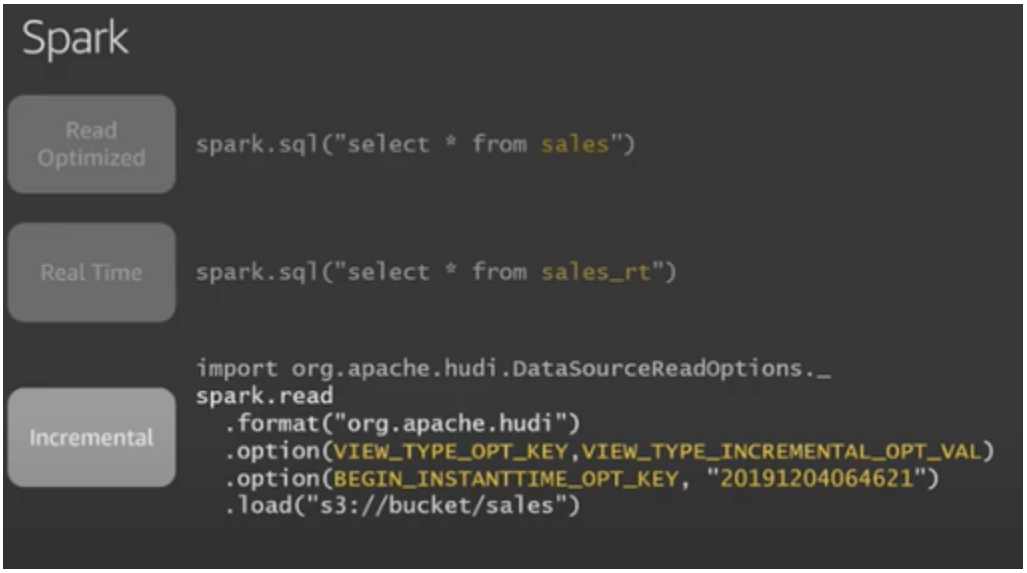
When you create the hudi dataset you can tell hudi to create table metadata in Hive metastore or AWS glue catalog

- ```
  inputDF.write()
         .format("org.apache.hudi")
         .options(opts)
         .option(TABLE_NAME, "sales").
         .mode(SaveMode.Append)
         .save("s3://bucket/sales");
  ```

Querying Hudi Datasets
  ```

Engine	Views	When to use?
Spark SQL	• Read Optimized • Real time • Incremental	• Notebooks & Data Science • Machine Learning • Custom data pipelines • Best support for incremental & streaming ETL
Hive	• Read Optimized • Real time • Incremental	• Data warehousing ETL • Incremental ETL pipelines
Presto	• Read Optimized	• Interactive & ad hoc queries

Querying with Spark



BEGIN_INSTANTTIME_OPT_KEY the begin commit time

VIEW_TYPE_OPT_KEY,VIEW_TYPE_INCREMENTAL_OPT_VAL incremental view

PRECOMBINE_FIELD_OPT_KEY used to rank/deduplicate data within the same batch. Mostly a timestamp, if multiple records have the same keys, then the data row with the latest timestamp gets precedence and is used, the other one is discarded.

Note: Presto support read only optimized views, *this may not be true in 2022*

Hudi CLI

Hudi has extensive CLI which help you manage the connect to a dataset and more. For instance you can look at compactions that are scheduled, you can unschedule these, show the commits - connect to a hudi dataset and say **commits show**, it will show you the commit timeline for this dataset, based on this you can rollback to a previous commit time.

Hive ?

Apache Hive is a modern data warehouse technology that enables reading, writing, and managing large datasets in distributed storage, typically within a Hadoop cluster, all using SQL. What it really means Hive is a data processing tool used on top of Hadoop and exposed through a SQL-like interface making it very accessible to a wide audience of analysts, data scientists, and engineers.

You typically interact with data using Hive by issuing a SQL query via clients such as Hive Web Interface, JDBC Driver, Hive CLI, or Thrift clients like beeline. Hive SQL queries that are submitted to the Hive Server use metadata in the metastore and decompose it into a query plan as a compiled MapReduce job. The MapReduce job is then executed on the Hadoop cluster which processes and aggregates the resultset data.

Just like with a traditional RDBMS you can organize tables of data in a databases but, in Hive there are two forms of tables: managed tables and external tables.

Managed Tables

Hive fully owns all data when it comes to *managed tables* and the files representing a table's data are stored on disk within the Hadoop HDFS file system under /user/hive/warehouse/databasename.db/tablename/ which should soley be controlled only using Hive functionality.

External Tables

Hive provides the ability to store metadata about the schema of files maintained outside of the control of Hive which is what is meant by an *extern al table*. This means that you can use Hive to processes these external datasets using the metadata data you provide about the structure of files but, Hive does not have control over another program altering them.

HUDI Cleanup

Hudi Cleanup command

[hoodie]$ hdfs dfs -mkdir -p /applications/hudi/lib

[hoodie]$ hdfs dfs -copyFromLocal /usr/lib/hudi/hudi-utilities-bundle.jar /applications/hudi/lib/hudi-utilities-bundle.jar

[hoodie]$ mkdir ~/hudi

[hoodie]$ hdfs dfs -get /applications/hudi/lib ~/hudi

[hoodie]$ spark-submit --class org.apache.hudi.utilities.HoodieCleaner ~/hudi/lib/hudi-utilities-bundle.jar --props s3://s3-data-tst-hudi-repo/hudi/cleanup/hudi_cleanup.properties --target-base-path s3://s3-data-tst-hudi-repo/hudi/event_log/ --spark-master yarn

If you want to keep the latest 3 file versions, you need to add these specific configs to the properties, in out case we only need to keep the latest version of the files so it is 1:

```
hoodie.cleaner.policy=KEEP_LATEST_FILE_VERSIONS
# hoodie.cleaner.fileversions.retained=3
hoodie.cleaner.fileversions.retained=1
```

Ref:

https://hudi.apache.org/docs/configurations#hoodiecleanerpolicy, `hoodie.cleaner.policy`
https://hudi.apache.org/docs/configurations#hoodiecleanerfileversionsretained, `hoodie.cleaner.fileversions.retained=`

```
public enum HoodieCleaningPolicy {
  KEEP_LATEST_FILE_VERSIONS, KEEP_LATEST_COMMITS, KEEP_LATEST_BY_HOURS}
```

How to find my spark-master

On master node terminal run **spark-shell**

```
Spark context available as 'sc' (master = yarn, app id =
application_1660226703437_0006).
```

spark-master value is **yarn**

In case you want to run it along with ingesting data, configs are available which enable you to run it synchronously or asynchronously, please configure the below:

hoodie.clean.automatic=true

hoodie.clean.async=true

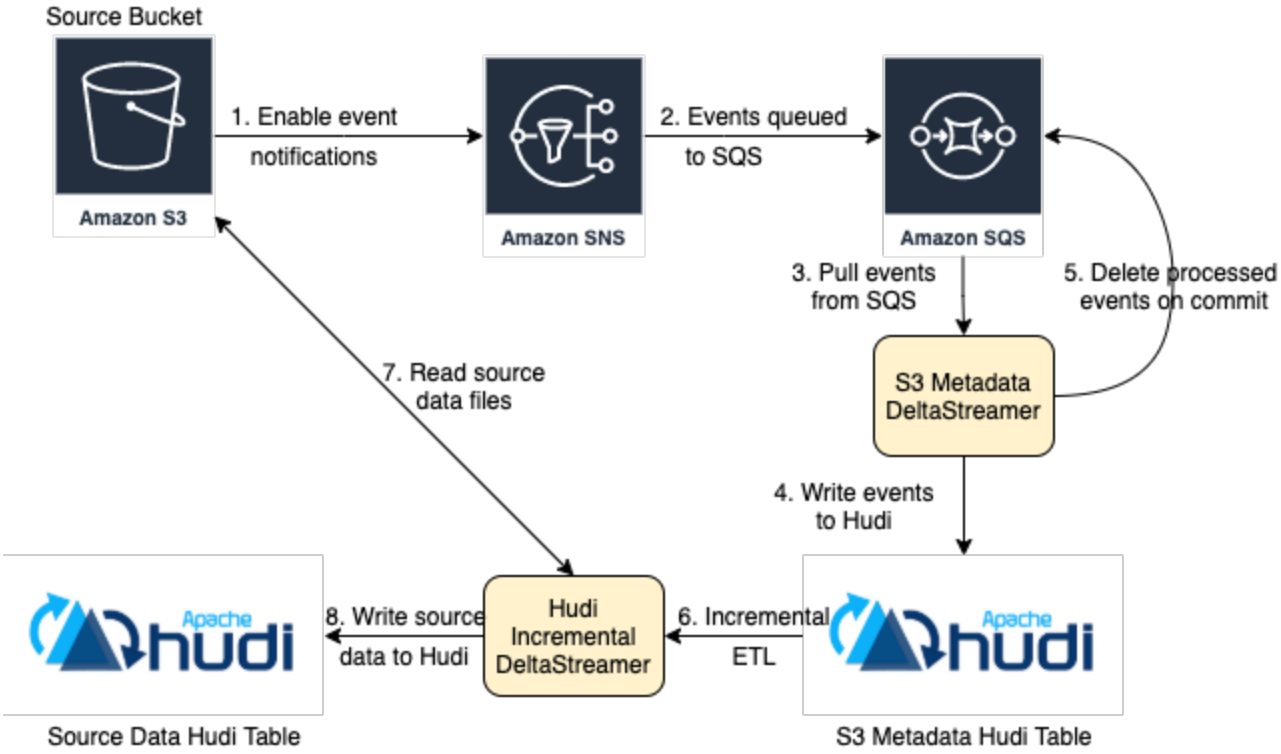
hoodie.cleaner.commits.retained=1 # default value is 10

ref: https://hudi.apache.org/docs/hoodie_cleaner/

Other Options - DeltaLake /Apache HUDI/Iceberg

Databricks Delta Lake	Uber Hudi	Netflix Iceberg
DeltaLake leverages Spark Partitioning and to do ***Partition Pruning*** while upserting /reading data but there is ***no concept of Primary Key*** in Delta Table and all columns are treated equally.	When writing data into HUDI, you model the records like how you would on a key-value store - specify a key field (unique for a single partition/across dataset), a partition field (denotes partition to place key into) and preCombine/combine logic that specifies how to handle duplicates in a batch of records written. This model enables HUDI to enforce primary key constraints like you would get on a database table. It also helps HUDI to build indexes on PRIMARY KEY (recordKey)	
	Hudi further optimizes compactions by utilizing Key Indexing to efficiently keep track of which files contain stale records.	
Column level update - DeltaLake provides a good API to do the database like Updates. For eg if I need to update all "field_1" to from "0.0" to "20.0", we can do like this.	In HUDI, there is no direct way to update any column value. However you can load all the data as spark dataframe , update the dataframe and then **"upsert"** to update the values.	
IF - you're primarily a Spark shop and expect relatively low write throughput. If you are also already a Databricks customer, Delta Engine brings significant improvements to both read and write performance and concurrency, and it can make sense to double down on their ecosystem.	**IF** - you use a variety of query engines and require flexibility for how to manage mutating datasets. Be mindful that supporting tooling and the overall developer experience can be rough around the edges. Though possible, there is also operational overhead required to install and tune Hudi for real, large-scale production workloads.	**IF** -your primary pain point is not changes to existing records, but the metadata burden of managing huge tables on an object store (more than 10k partitions). Adopting Iceberg will relieve performance issues related to S3 object listing or Hive Metastore partition enumeration. On the contrary, support for deletions and mutations is still preliminary, and there is operational overhead involved with data retention

Ingestion from AWS S3 using Hudi



ref: https://hudi.apache.org/blog/2021/08/23/s3-events-source/

Hudi and Streams

https://blaqfireroundup.wordpress.com/2021/07/29/using-hudi-deltastreamer/

https://aws.amazon.com/blogs/big-data/build-a-serverless-pipeline-to-analyze-streaming-data-using-aws-glue-apache-hudi-and-amazon-s3/

Hudi Community Zoom call

https://hudi.apache.org/community/syncs#monthly-community-call

References

- https://www.youtube.com/watch?v=_ckNyL_Nr1A
- Hudi Workshop Studio
- https://hudi.apache.org/docs/writing_data/ **
- https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hudi-work-with-dataset.html
- https://github.com/nmukerje/EMR-Hudi-Workshop/blob/master/notebooks/Hudi_Pyspark_Example.ipynb
- https://github.com/SirOibaf/hops-examples/blob/hudi_python/notebooks/featurestore/hudi/HudiOnHops.ipynb
- Apache Hudi and Pyspark
- https://hudi.apache.org/docs/configurations/
- https://hudi.apache.org/docs/quick-start-guide/
- https://hudi.apache.org/blog/2020/08/20/efficient-migration-of-large-parquet-tables/
- https://www.onehouse.ai/blog/apache-hudi-native-aws-integrations
- https://aws.amazon.com/blogs/big-data/new-features-from-apache-hudi-available-in-amazon-emr/ **
- https://pages.awscloud.com/rs/112-TZM-766/images/EV_analytics-sprint-week-apache-hundi-amazon-emr_Sep-2020.pdf
- https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hive-metastore-glue.html
- https://thecodinginterface.com/blog/aws-emr-with-apache-hive/
- https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hive-metastore-glue.html *
- https://github.com/PacktPublishing/Simplify-Big-Data-Analytics-with-Amazon-EMR-/blob/main/chapter_11/spark-hudi-etl.ipynb
- https://stackoverflow.com/questions/66552781/spark-datasource-hudi-table-read-using-instant-time
- https://medium.com/swlh/apache-hudi-vs-delta-lake-295c019fe3c5
- https://www.linkedin.com/pulse/datalake-in-depth-comparison-deltalake-apache-hudi-tanu-dua