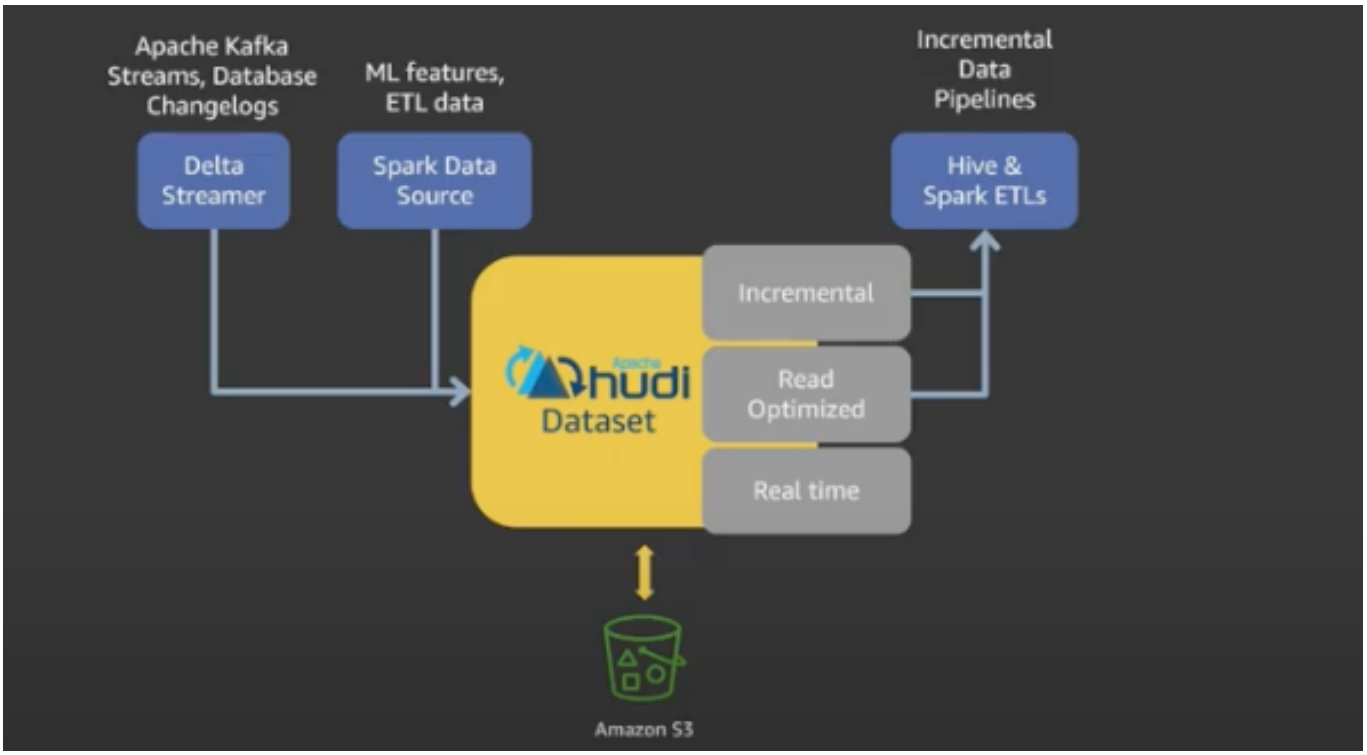
Apache Hudi

Apache Hudi and Data Privacy



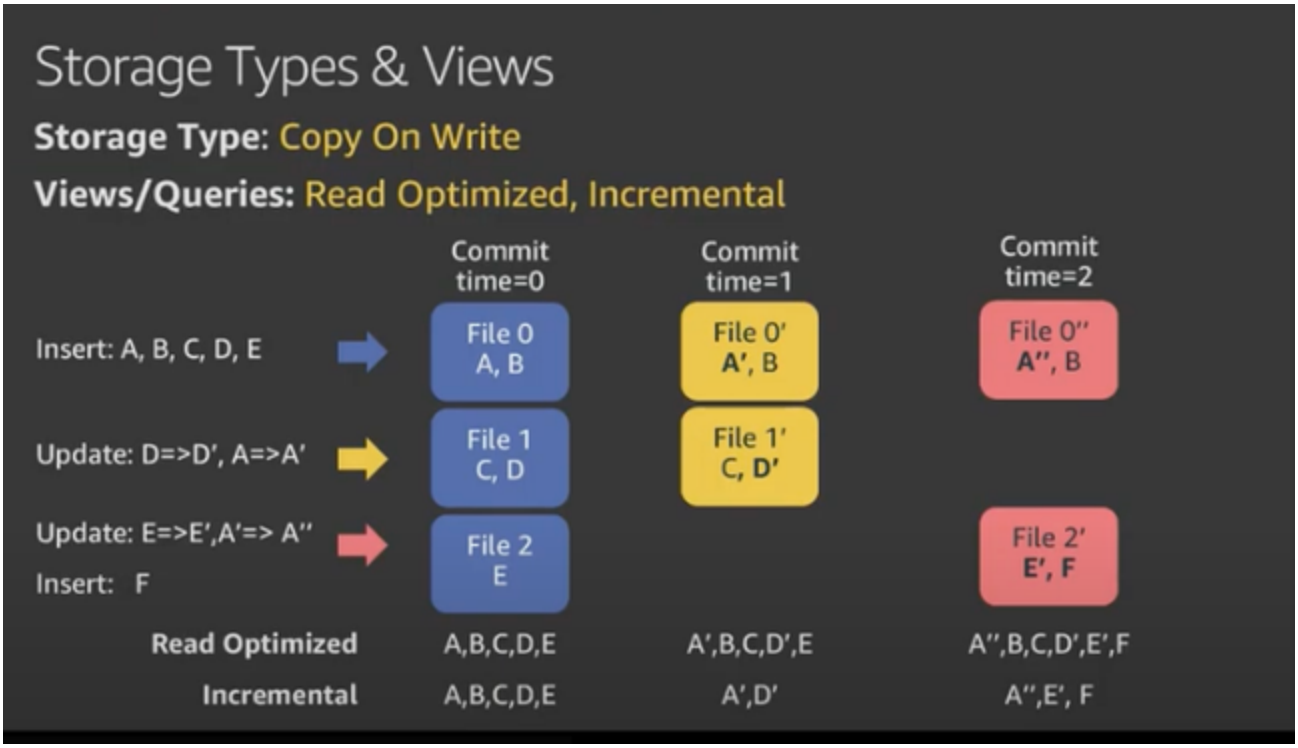
- Hudi supports in place updates, inserts and deletes in a Big data system. There are couple of way to use Hudi:
  - Convert all current datasets/parquet to Apache Hudi parquet files.
  - Apache Hudi without having to convert existing Parquet data in our data lake - As part of this operation, Hudi generates metadata only. It writes the metadata in a separate file that corresponds to each data file in the dataset. The original data is left as-is and not copied over. This is available in EMR.
- 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

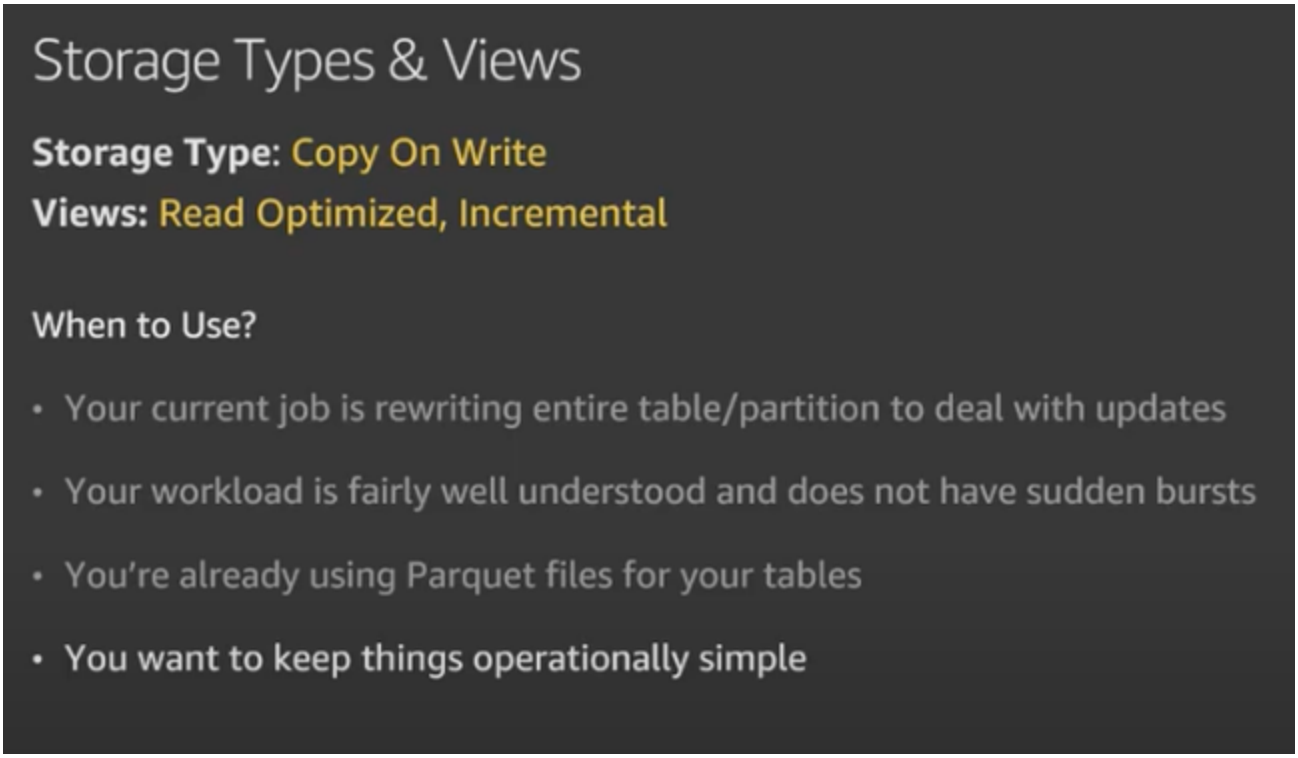
Storage Types

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.
-

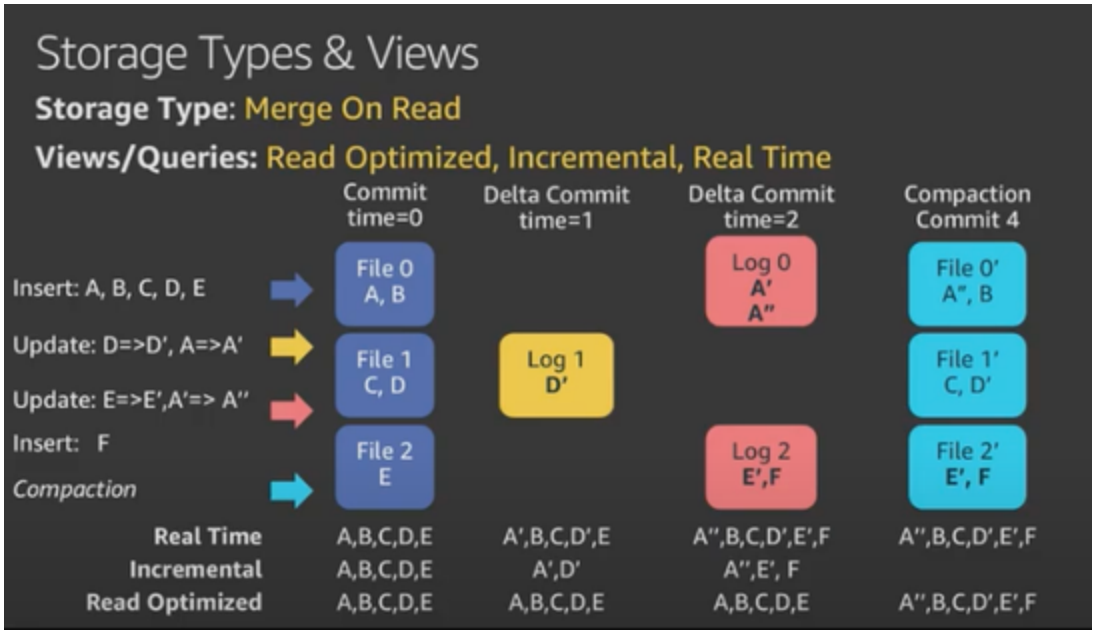
- 

- 

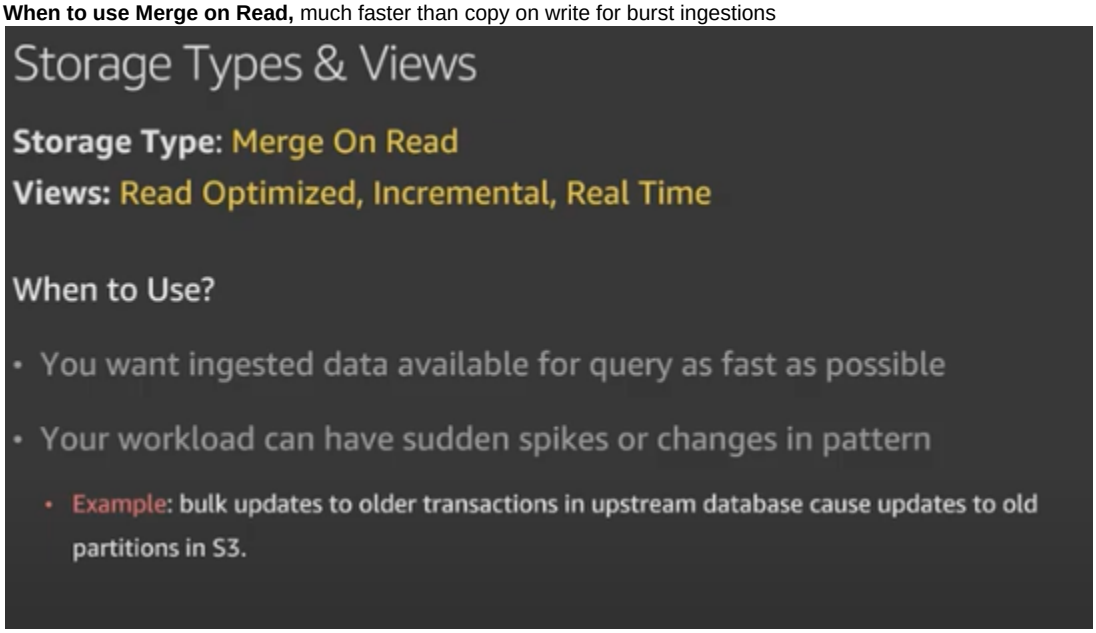
- If you have burst writes, then COW is not best choice as it will create lots of file versions. See above File 0, has 3 version - File 0, File 0' and File 0''

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 it compacts the recent data aggressively.



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

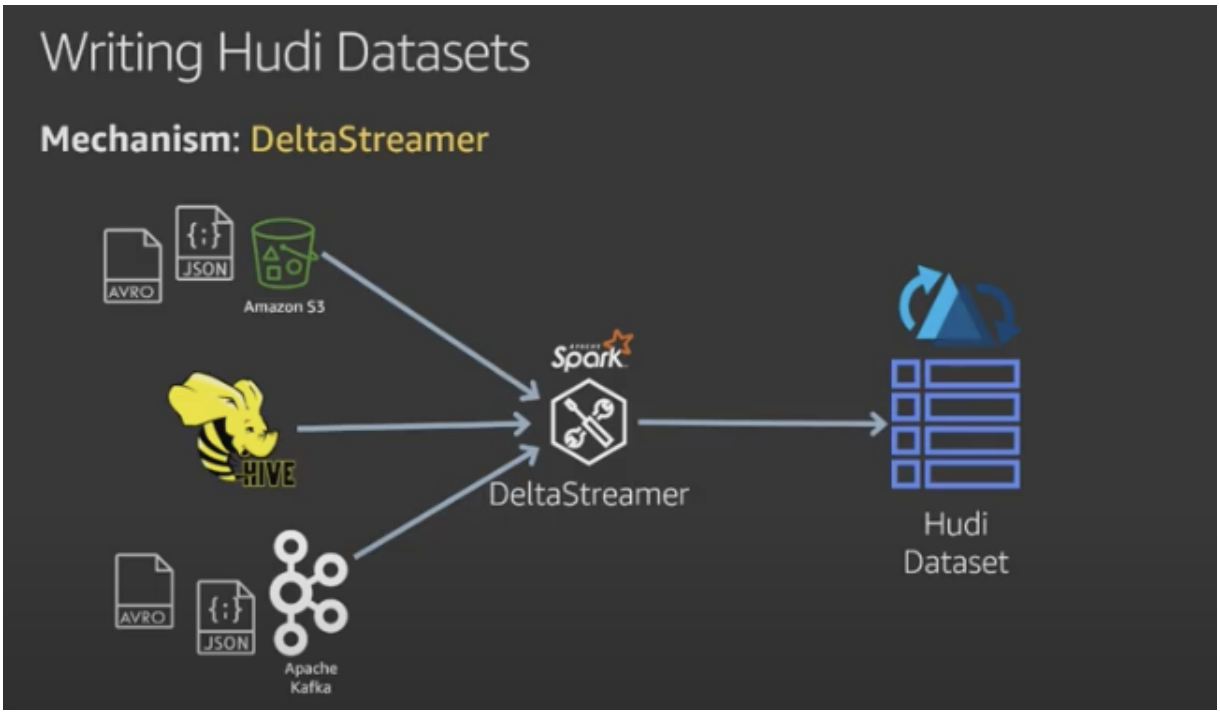


Hudi Logical Views

- **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 **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 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 which will get applied on the streaming data.
    -



Writing Pyspark Dataframe as a Hudi datasets

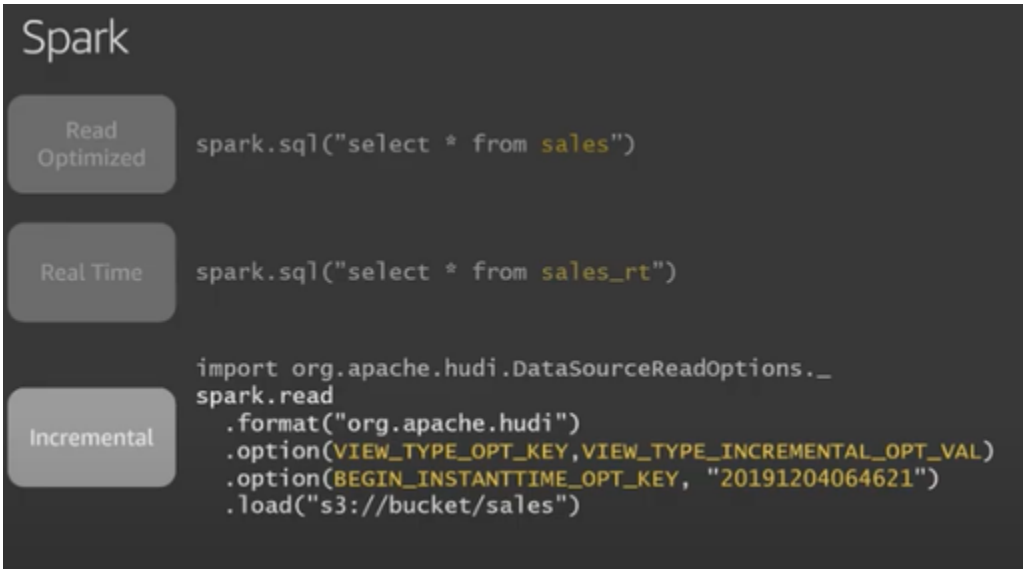
When you create the hudi dataset you can tell hudi to create table metadata in Hive metastore or AWS glue catalog

- ```
  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 on 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.

References

- https://www.youtube.com/watch?v=_ckNyL_Nr1A
- https://hudi.apache.org/docs/writing_data/ **
- https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hudi-work-with-dataset.html
- 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/