# Understanding Apache Hudi: Is Hudi Always Running or Managed via Spark Jobs?

## Introduction

Apache Hudi is a data lake framework that provides capabilities like data versioning, incremental updates, and real-time query abilities. However, one of the common questions asked is whether Hudi is 'always running' like a service or not. This document explores whether Hudi operates as an always-on service or is controlled through Spark jobs.

## Batch Processing with Hudi

In batch processing, Hudi is used to write data in predefined intervals by running Spark jobs that load new data into the Hudi table.

### Example: Batch ETL Job

Let's say you are reconciling transactions daily and writing the data into a Hudi table:

from pyspark.sql import SparkSession  
from pyspark.sql import functions as F  
  
# Create a Spark session  
spark = SparkSession.builder .appName("Batch Hudi Job") .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") .getOrCreate()  
  
# Load new transactions data from a CSV file  
transactions\_df = spark.read.format("csv") .option("header", "true") .load("s3://bucket/transactions/daily\_transactions.csv")  
  
# Specify Hudi options  
hudi\_options = {  
 'hoodie.table.name': 'transactions\_hudi',  
 'hoodie.datasource.write.recordkey.field': 'transaction\_id',  
 'hoodie.datasource.write.precombine.field': 'timestamp',  
 'hoodie.datasource.hive.sync.enable': 'true',  
 'hoodie.datasource.hive.sync.database': 'default',  
 'hoodie.datasource.hive.sync.table': 'transactions\_hudi',  
 'hoodie.datasource.write.operation': 'upsert',  
 'hoodie.datasource.write.table.type': 'MERGE\_ON\_READ',  
 'hoodie.datasource.hive.sync.partition\_extractor\_class': 'org.apache.hudi.hive.MultiPartKeysValueExtractor'  
}  
  
# Write the DataFrame to the Hudi table  
transactions\_df.write.format("hudi") .options(\*\*hudi\_options) .mode("append") .save("s3://bucket/hudi/tables/transactions\_hudi")

In this scenario, a Spark job is scheduled to run at specific intervals (daily or weekly) and updates the Hudi table with new data. The job stops after completion and is not continuously running.

## Real-Time Streaming with Hudi

In streaming, Hudi can be integrated with frameworks like Apache Kafka and Spark Structured Streaming. In this scenario, a Spark Structured Streaming job continuously consumes data from Kafka, writing it to a Hudi table in near real-time.

### Example: Real-Time Streaming with Kafka and Hudi

Let’s assume we are receiving transaction data continuously through Kafka and need to write it to a Hudi table as it arrives:

from pyspark.sql import SparkSession  
from pyspark.sql.functions import from\_json, col  
from pyspark.sql.types import StructType, StringType  
  
# Create a Spark session  
spark = SparkSession.builder .appName("Streaming Hudi Job") .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") .getOrCreate()  
  
# Define the schema for the Kafka messages (assuming JSON format)  
schema = StructType() .add("transaction\_id", StringType()) .add("amount", StringType()) .add("timestamp", StringType())  
  
# Read from the Kafka topic (streaming)  
kafka\_df = spark.readStream.format("kafka") .option("kafka.bootstrap.servers", "localhost:9092") .option("subscribe", "transactions\_topic") .load()  
  
# Parse the JSON messages from Kafka  
parsed\_df = kafka\_df.select(from\_json(col("value").cast("string"), schema).alias("data")) .select("data.\*")  
  
# Hudi options for streaming  
hudi\_options = {  
 'hoodie.table.name': 'transactions\_hudi',  
 'hoodie.datasource.write.recordkey.field': 'transaction\_id',  
 'hoodie.datasource.write.precombine.field': 'timestamp',  
 'hoodie.datasource.write.table.type': 'MERGE\_ON\_READ',  
 'hoodie.upsert.shuffle.parallelism': 2,  
 'hoodie.insert.shuffle.parallelism': 2  
}  
  
# Write the stream to a Hudi table  
parsed\_df.writeStream .format("hudi") .options(\*\*hudi\_options) .outputMode("append") .option("checkpointLocation", "s3://bucket/hudi/checkpoints/") .start("s3://bucket/hudi/tables/transactions\_hudi") .awaitTermination()

In this streaming scenario, the Spark job is continuously running, consuming data from Kafka and writing it to the Hudi table in near real-time. Here, the job is effectively 'always on' until it is manually terminated.

## Conclusion

Hudi itself is not an 'always-on' service. It is a storage layer that integrates with Spark jobs to handle data ingestion, updates, and queries. Whether Hudi appears to be 'always on' depends on how the Spark jobs using Hudi are structured. In batch processing, the Spark jobs run at specific intervals, while in streaming scenarios, the Spark job can be continuously running.