Table of Contents

Introduction

This doc can be used as a reference for defining the config files used in metadata driven unified data analytics framework. This doc also provides instructions on how to leverage system manager parameter store inside the config files.

Schema.Json

Help For schemas - applicable to data coming from files, databases, streams and api's. Please refer to the below explanation for defining attributes

Defined below are various elements supported in schema.json

"schemas": [
{

"type": "customerdata", //*This should be same as the type defined in SourceDestinationConfig file*

"enrichWithTimestampData":true, //*Use this to append a timestamp for every record that got processed by the framework before persisting it into sink. Every record will have create_time_stamp and snapshotif(timeinms) as added columns before persisting into sink*

"recordLength":48, // *Only specify this if you are reading a text file, this is a must to specify the record length for text files*

"deriveby": "Index", // *Incoming data values of dataset row can be derived by Index or Name. Index or Name can be used for dataset created from CSV,*
but for DB the values are derived by Name, for text files, values are derived by Index.

Parameters a schema attribute accepts

attributes:[
{
name": "", // *Name of the column of the dataset read from Sounrce*.
"updatedName": "" // *You can specify a value here if you want to use a new column name before persisting the data in sink*

"defaultvalue":""// *This value will be used if the incoming source data has null or blank in it.*

"mandatory": false // *if specified true, then records with missing values will not be persisted to sink and will be moved to error folder*

"type": "string", // *type is case sensitive (valid types are string, integer, biginteger,decimal,bigdecimal double, long, boolean,date, timestamp)*

range":"0-10", // *This is mandatory if you are reading a text file or its optional*

"index": 0 // *This is a mandatory element and 0 means 1st column of the row*

"scale": "5" // *scale Is applicable to type double, decimal and bigdecimal*

"format": "yyyy-MM-ddHH:mm:ss", //*Format is mandatory for date or timestamp field*
}
]

A sample schema is shown below

```
{
  "version": 1,
  "schemas": [
    {
      "type": "customerdataintheformoftextfile",
      "enrichWithTimestampData":true,
      "recordLength":48,
      "deriveby": "Index",

      "attributes": [
        {

          "name": "firstName", //
          "updatedName": "updatedFirstName"
          "mandatory": false, //
          "type": "string",
          "range":"0-10",
          "index": 0 // This is a mandatory element and 0 means 1st column of the row
        },
        {
        // This is how you define a decimal field, put "range" attribute if reading from text file
          "name": "amountindecimal",
          "mandatory": false,
          "type": "decimal",
          "range":"10-20",
          "index": 1,
          "scale": "5" // scale Is applicable to type double, decimal and bigdecimal
        },
        {
          // This is how you define a date field, put "range" attribute if reading from text file
          "name": "dateofbirthindateformat",
          "mandatory": false,
          "type": "date",
```

```
    "format": "yyyy-dd-MM",// Format should be specified for date field
    "index": 2
   },
   {
   // This is how you define a timestamp field, put "range" attribute if reading from text file
    "name": "timeofbirthintimestampformat",
    "mandatory": false,
    "type": "timestamp",
    "format": "yyyy-MM-ddHH:mm:ss", //Format should be specified for timestamp field
    "index": 3
   },
   ]

   }
  ]
}
```

PreDefinedSchemaLocation

//Help for defining PreDefinedSchemaLocation

If you have sample json data or parquet data that captures all the attributes of the incoming data then you don't have to define a schema in schema.json.

Store the sample file as sourcetype.json or sourcetype.parquet (replace sourcetype with the actual value of "Name" attribute defined in sourceDestinationsetupconfig.json)
in a s3 bucket and specify the path of the s3 bucket in predefineSchemaLocation column of DynamoDB when persisting through UI.

If using S3 as the schema holder then store the sample files under prefix /predefinedschemalocation

SourceDestinationConfigSetup – For Source Database (JDBC)

Sample Config file for databases based source

```
{
  "version": 1,
  "source": {
    "databases": [ // This signifies, the input source is a database based source (Only JDBC type is upported)
      {
        "sourcetype": "customerdbdata", //Name could be anything and will be used while submitting the job to the framework. If there is a schema defined for this source in schema.json, make sure the name remains the same in schema.json
        "format": "jdbc",
        "dbInputOptions": { The framework supports several   attributes for reading the data from jdbc, refer to this https://spark.apache.org/docs/latest/sql-data-sources-jdbc.html for more information
          "url": "", //The JDBC URL to connect to. The source-specific connection properties may be specified in the URL. e.g., jdbc:postgresql://localhost/test?user=fred&password=secret
          "dbtable": "",//The JDBC table that should be read from or written into. Note that when using it in the read path anything that is valid in a FROM clause of a SQL query can be used. For example, instead of a full table you could also use a subquery in parentheses. It is not allowed to specify `dbtable` and `query` options at the same time.
          "query":"", //A query that will be used to read data into Spark.Specify this if you want to use custome query instead of directly reading from table by specifying a dbtable option. An example qeury will be like select c1, c2 from t1,
                    //you can also specify select c1, c2 from t1 where c3= {c3val}.c3val can be passed while invoking the job (c3val#actualvalue)
                    //
          "driver": "com.sap.db.jdbc.Driver",     //The class name of the JDBC driver to use to connect to this URL.
          "user": "{dbusername}", //It is advisable to use a  placeholder here and define the placeholder in system parameters of AWS in the form of /udaf/dbusername and the actual value correspoding to this placeholder. The value will be used for processing
          "password": "{dbpassword}}" //It is advisable to use a  placeholder here and define the placeholder in system parameters of AWS in the form of /udaf/dbusername and the actual value correspoding to this placeholder. The value will be used for processing


        },
        "dbOutputOptions": [ //Any Object (S3) or File (HDFS) based Sink can be provided here.This is useful when you want to enable a datalake or a deltalke using delta or hudi format
          {
            "outputFormat": "parquet", // The output will be saved as parquet in s3 or hdfs depending on the output directory  specified. Supported values are parquet,avro,orc,delta,hudi,deltacdc (use deltacdc if you  want the input data to be updated in
```

your delta lake),hudicdc (use hudicdc if you are want the input to be updated in your Hudi lake),json,csv

"savemode": "Append", // Apend or Overwrite. This is Spark related

"numberOfOutputPartitions": "1",this is spark specific, more number of partitions will parallelize the execution. The code uses coalesce. So this value cannot be greate than the

"outputDirectory": "s3://output/", Output directory to write the processed data to.This could be any object (s3) or file level destination.

"partitioncolumns": "" //Partitioning columns (Spark specific, if the data could be partitioned based on certain columns, the name of the column can be specified here for fast processing)

}

],

"dbErrorOptions": {

// This is not madatory, Only Specify the options here if you want records that fail validation to be persisted in s3 or hdfs. You can add several custom attributes here and then override BaseErrorHandler class to provide you own custom error handling

"outputFormat": "csv",

"savemode": "Append",

"numberOfOutputPartitions": "1"

"outputDirectory": "s3://error/"

},

"sparkoptions": [// Optional - Refer to this doc https://spark.apache.org/docs/latest

{

"type": "conf", // Spark configuration settings.

"options": {}

},

{

"type": "read",

"options": {}

},

{

"type": "write", //Spark write setting. Used extensively if persisting in Hudi or Delta lake. Please refer to hudhelp or deltahelp doc.

"options": {}

}

],

// hooks are kind of like sinks where you can persist the data to and also read data from

"hooks": [

{

"hooktype": "athena", // Support registering the s3 data into glue and retrieving it using Athena. Currently support is provided for json,csv,parquet,orc,avro,delta

"hookoptions": [

{

"type": "read", // use this to specify options to read the data from sinks. Default class don't provide an implementation. Override BaseAthenaHandler and specify custom implementation

```
            "options": {}
          },
          {
            "type": "write",
            "options": {
              "athenaDB": "deltadb", // Athena database. This has to exist in Athena or framework will not work
              "athenaTable": "customers", framework will create the table with the specified name if it doesn't exist
              "athenaDir": "s3://dms-ingestion-encrypted-levis/athena/", This is mandatory and athena expects it. Specify a path here.
              "region": "us-west-2", Region to use and is mandatory
              "tblproperties": "", specify custom table properties. Refer to AWS athena doc for examples/
            }
          }
        ]
      },
      {
        "hooktype": "redshift", //Redshift sink.
        "hookoptions": [
          {
            "type": "read",
            "options": {}
          },
          {
            "type": "write", Refer to  link provided for various attributes that can be provided as spark options for writing dataset to redshift and explanation https://docs.databricks.com/data/data-sources/aws/amazon-redshift.html
            "options": {
              "url": "jdbc:redshift://demoredshiftcluster.****.us-east-1.redshift.amazonaws.com:5439/demodb?user={redshiftdbuser}&password={redshiftdbpwd}", //user name and password should be stored in system parameters in the form of /udaf/key and key should be used here in the format {key}
              "aws_iam_role": "arn:aws:iam::*****:role/vfcredshiftRole",
              "dbtable": "customerredshiftdata",
              "tempdir": "s3a://temp/data"
            }
          }
        ]
      },
      {
        "hooktype": "dynamodb", // Suports dynamodb sink.
        "hookoptions": [
          {
            "type": "read",
```

```
            "options": {}
          },
          {
            "type": "write", The framework uses https://github.com/awslabs/emr-dynamodb-connector. Please refer to it for explanation of each attribute
            "options": {
              "dynamodb.servicename": "dynamodb",
              "dynamodb.output.tableName": "****",
              "dynamodb.endpoint": "dynamodb.us-east-1.amazonaws.com",
              "dynamodb.regionid": "us-east-1",
              "dynamodb.throughput.write": "1",
              "dynamodb.throughput.write.percent": "1",
              "mapred.output.format.class": "org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat",
              "mapred.input.format.class": "org.apache.hadoop.dynamodb.read.DynamoDBInputFormat"
            }
          }
        ]
      },
      {
        "hooktype": "jdbc", Any JDBC Database Sink
        "hookoptions": [
          {
            "type": "read",
            "options": {}
          },
          {
            "type": "write", // Refer to this https://spark.apache.org/docs/latest/sql-data-sources-jdbc.html for various DB write spark options that are supported
            "options": {
              "url": "jdbc:postgresql://testinstance.****.us-east-1.rds.amazonaws.com:5432/testdb",
              "dbtable": "ricohmaster",
              "user": "{postgresuser}",  Use System manager to store the values for the user name and pwd. The system parameter key should be in the form of /udaf/postgresuser
              "password": "{postgrespwd}",Use System manager to store the values for the user name and pwd. The system parameter key should be in the form of /udaf/postgrespwd
              "numPartitions": "10"
            }
          }
        ]
      },
      {
        "hooktype": "elasticsearch", Elastic search sink
        "hookoptions": [
```

```
            {
              "type": "read",  These are  spark options for reading from elastic search. refer to this for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
              "options": {
                "index": "index/ricohmaster",
                "es.nodes.wan.only": "true",
                "es.port": "443",
                "es.net.ssl": "true",
                "es.nodes": "https://vpc-testdomain-***.us-east-1.es.amazonaws.com"
              }
            },
            {
              "type": "write",These are  spark options for writing data to elastic search. refer to this for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
              "options": {
                "mode": "Append",
                "save": "index/ricohmaster",
                "es.nodes.wan.only": "true",
                "es.port": "443",
                "es.net.ssl": "true",
                "es.nodes": "https://vpc-testdomain-****.us-east-1.es.amazonaws.com"
              }
            }
          ]
        }
      ]
    }
  ]
}
```

SourceDestinationConfigSetup – For Source FilesOrObjectbased (E.g-S3/HDFS)

Sample config file for File/Object based sources

```
{
  "version": 1,
  "source": {
    "files": [ // This signifies, the input source is  file based such as HDFS or Object based source such as S3
      {
        "sourcetype": "customer_full",//Name could be anything and will be used while submitting the job to the framework.If there is a schema defined for this source in schema.json, make sure the name remains the same in schema.json
```

```
"fileInputOptions": {
  "fileInputDir": "s3://metadata-framwork/full_load/",  //hdfs directory  or s3 bucket for reading source files
  "fileInputFormat": "csv",format of the file (such as csv,json,xml,Parquet,avro,orc)
},
"fileOutputOptions": [ //Any Object (S3) or File (HDFS) based Sink can be provided here.This is useful when you want to enable a datalake or a deltalke using delta or hudi format
  {

    "outputFormat": "parquet", // The output will be saved as parquet in s3 or hdfs depending on the output directory  specified. Supported values are parquet,avro,orc,delta,hudi,deltacdc (use deltacdc if you  want the input data to be updated in your delta lake),hudicdc (use hudicdc if you are  want the input to be updated in your Hudi lake),json,csv
    "savemode": "Append", // Apend or Overwrite. This is Spark related
    "numberOfOutputPartitions": "1",this is spark specific, more number of partitions will parallelize the execution. The code uses coalesce. So this value cannot be greate than the
    "outputDirectory": "s3://output/",// Output directory to write the processed data to.This could be any object (s3) or file level(hdfs) destination.
    "partitioncolumns": "" //Partitioning columns (Spark specific, if the data could be partitioned  based on certain columns, the name of the column can be specified here for fast processing)
  }
],
"fileErrorOptions": {
  // This is not madatory, Only Specify the options here if you want records that fail validation to be persisted in s3 or hdfs. You can add several custom attributes here and then override BaseErrorHandler class to provide you own custom error handling.
  "savemode": "Append",
  "numberOfOutputPartitions": "1",
  "outputDirectory": "s3://metadata-framwork/dms-ingestion/full-load/dbo/error-output/BIN_DETAIL/"
},
"filePostProcessingOptions": {
  // Optional-Attributes can be specified here and be used for provided any custome postporcessing logic by Overriding postProcess method of BaseDataLakeSourceProcessor
},
"sparkoptions": [ // Optional - Spark related file or object based read or write options. Refer to this doc https://spark.apache.org/docs/latest
  {
    "type": "conf", // Spark configuration settings.
    "options": {}
  },
  {
    "type": "read",
    "options": {}
```

```
      },
      {
        "type": "write",
        "options": {}
      }
    ],
    // hooks are kind of like sinks where you can persist the data to and also read data from. Follwing are supported in the framework
    "hooks": [
      {
        "hooktype": "athena", // Support registering the s3 data into glue and retrieving it using Athena. Currently support is provided for json,csv,parquet,orc,avro,delta
        "hookoptions": [
          {
            "type": "read", // use this to specify options to read the data from sinks. Default class don't provide an implementation. Override BaseAthenaHandler and specify custom implementation
            "options": {}
          },
          {
            "type": "write",
            "options": {
              "athenaDB": "deltadb", // Athena database. This has to exist in Athena or framework will not work
              "athenaTable": "customers", framework will create the table with the specified name if it doesn't exist
              "athenaDir": "s3://dms-ingestion-encrypted-levis/athena/", This is mandatory and athena expects it. Specify a path here.
              "region": "us-west-2", Region to use and is mandatory
              "tblproperties": "", specify custom table properties. Refer to AWS athena doc for examples/
            }
          }
        ]
      },
      {
        "hooktype": "redshift", //Redshift sink.
        "hookoptions": [
          {
            "type": "read",
            "options": {}
          },
          {
            "type": "write", Refer to  link provided for various attributes that can be provided as spark options for writing dataset to redshift and explanation https://docs.databricks.com/data/data-sources/aws/amazon-redshift.html
```

```
            "options": {
              "url": "jdbc:redshift://demoredshiftcluster.****.us-east-
1.redshift.amazonaws.com:5439/demodb?user={redshiftdbuser}&password={redshiftdbpwd}",
              "aws_iam_role": "arn:aws:iam::****:role/vfcredshiftRole",
              "dbtable": "customerredshiftdata",
              "tempdir": "s3a://temp/data"
            }
          }
        ]
      },
      {
        "hooktype": "dynamodb", // Suports dynamodb sink.
        "hookoptions": [
          {
            "type": "read",
            "options": {}
          },
          {
            "type": "write", The framework uses https://github.com/awslabs/emr-dynamodb-
connector. Please refer to it for explanation of each attribute
            "options": {
              "dynamodb.servicename": "dynamodb",
              "dynamodb.output.tableName": "***",
              "dynamodb.endpoint": "dynamodb.us-east-1.amazonaws.com",
              "dynamodb.regionid": "us-east-1",
              "dynamodb.throughput.write": "1",
              "dynamodb.throughput.write.percent": "1",
              "mapred.output.format.class":
"org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat",
              "mapred.input.format.class":
"org.apache.hadoop.dynamodb.read.DynamoDBInputFormat"
            }
          }
        ]
      },
      {
        "hooktype": "jdbc", any JDBC database Sink
        "hookoptions": [
          {
            "type": "read",
            "options": {}
          },
          {
            "type": "write", // Refer to this https://spark.apache.org/docs/latest/sql-data-sources-
jdbc.html for various write options that are supported
            "options": {
```

```
            "url": "jdbc:postgresql://testinstance.****.us-east-
1.rds.amazonaws.com:5432/testdb",
            "dbtable": "****",
            "user": "{postgresuser}",  Use System manager to store the values for the user name
and pwd. The system parameter key should be in the form of /udaf/postgresuser
            "password": "{postgrespwd}",Use System manager to store the values for the user
name and pwd. The system parameter key should be in the form of /udaf/postgrespwd
            "numPartitions": "10"
          }
         }
        ]
       },
       {
        "hooktype": "elasticsearch", Elastic search sink
        "hookoptions": [
         {
          "type": "read",  These are  spark options for reading from elastic search. refer to this
for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
          "options": {
           "index": "index/ricohmaster",
           "es.nodes.wan.only": "true",
           "es.port": "443",
           "es.net.ssl": "true",
           "es.nodes": "https://vpc-testdomain-****.us-east-1.es.amazonaws.com"
          }
         },
         {
          "type": "write",These are  spark options for writing data to elastic search. refer to this
for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
          "options": {
           "mode": "Append",
           "save": "index/ricohmaster",
           "es.nodes.wan.only": "true",
           "es.port": "443",
           "es.net.ssl": "true",
           "es.nodes": "https://vpc-testdomain-****.us-east-1.es.amazonaws.com"
          }
         }
        ]
       }
      ]
     }
    ]
   }
}
```

SourceDestinationConfigSetup – For Source Kafka

Sample Config file for Kafka Based Source

```
{
  "version": 1,
  "source": {
    "streams": [ //This signifies, the input source is stream based such as kinesis or kafka or file structured streaming.
      {
        "sourcetype": "levikafka", //Name could be anything and will be used while submitting the job to the framework
        "streamtype": "kafka", //Can be either kafka or kinesis or filestreaming.
        "inputFormat": "json", // Support for Only json is provided
        "flattenJson": "true", // If input data is nested and needs to be flattened. Optional and default is false
        "streamInputOptions": {
          // Framework uses spark kafka structured streaming. Refer to this more details on the attributes https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html
          "kafka.bootstrap.servers": "b-3.levikafka.a0vgly.c2.kafka.us-west-2.amazonaws.com:9092,b-2.levikafka.a0vgly.c2.kafka.us-west-2.amazonaws.com:9092,b-1.levikafka.a0vgly.c2.kafka.us-west-2.amazonaws.com:9092",
          "subscribe": "levi_xstore_topic",
          "startingOffsets": "latest"
        },
        "streamOutputOptions": [ //Any Object (S3) or File (HDFS) based Sink can be provided here.This is useful when you want to enable a datalake or a deltalke using delta or hudi format
          {
            "outputFormat": "parquet",// The output will be saved as parquet in s3 or hdfs depending on the output directory  specified. Supported values are parquet,avro,orc,delta,hudi,deltacdc (use deltacdc if you  want the input data to be updated in your delta lake),hudicdc (use hudicdc if you are  want the input to be updated in your Hudi lake),json,csv
            "savemode": "Append",// Apend or Overwrite. This is Spark related
            "outputDirectory": "s3://metadata-framwork/real-time-ingestion/ingestion_output/",Output directory to write the processed data to.This could be any object (s3) or file level(hdfs) destination.
            "checkpointLocation": "s3://metadata-framwork/ingestion_checkpoint3/", // Checkpint location for storing offsets. This is mandatory
            "partitioncolumns": "" //Partitioning columns (Spark specific, if the data could be partitioned  based on certain columns, the name of the column can be specified here for fast processing)
          }
        ],
        "streamErrorOptions": {
```

```
        // This is not madatory, Only Specify the options here if you want records that fail validation to be persisted in s3 or hdfs. You can add several custom attributes here and then override BaseErrorHandler class to provide you own custom error handling.
        "savemode": "Append",
        "outputFormat": "csv",
        "savemode": "Overwrite",
        "numberOfOutputPartitions": "1",
        "outputDirectory": "s3://metadata-framwork/real-time-ingestion/error_output/"
      },
      "sparkoptions": [ // Optional -  Refer to this doc https://spark.apache.org/docs/latest
        {
          "type": "conf", // Spark configuration settings.
          "options": {}
        },
        {
          "type": "read",
          "options": {}
        },
        {
          "type": "write",
          "options": {}
        }
      ],
      // hooks are kind of like sinks where you can persist the data to and also read data from. Follwing are supported in the framework
      "hooks": [
        {
          "hooktype": "athena", // Support registering the s3 data into glue and retrieving it using Athena. Currently support is provided for json,csv,parquet,orc,avro,delta
          "hookoptions": [
            {
              "type": "read", // use this to specify options to read the data from sinks. Default class don't provide an implementation. Override BaseAthenaHandler and specify custom implementation
              "options": {}
            },
            {
              "type": "write",
              "options": {
                "athenaDB": "deltadb", // Athena database. This has to exist in Athena or framework will not work
                "athenaTable": "customers", framework will create the table with the specified name if it doesn't exist
                "athenaDir": "s3://dms-ingestion-encrypted-levis/athena/", This is mandatory and athena expects it. Specify a path here.
                "region": "us-west-2", Region to use and is mandatory
```

```
            "tblproperties": "", specify custom table properties. Refer to AWS athena doc for
examples/
          }
         }
        ]
       },
       {
        "hooktype": "redshift", //Redshift sink.
        "hookoptions": [
         {
          "type": "read",
          "options": {}
         },
         {
          "type": "write", Refer to  link provided for various attributes that can be provided as
spark options for writing dataset to redshift and explanation
https://docs.databricks.com/data/data-sources/aws/amazon-redshift.html
          "options": {
           "url": "jdbc:redshift://demoredshiftcluster.****.us-east-
1.redshift.amazonaws.com:5439/demodb?user={redshiftdbuser}&password={redshiftdbpwd}",
           "aws_iam_role": "arn:aws:iam::****:role/vfcredshiftRole",
           "dbtable": "customerredshiftdata",
           "tempdir": "s3a://temp/data"
          }
         }
        ]
       },
       {
        "hooktype": "dynamodb", // Suports dynamodb sink.
        "hookoptions": [
         {
          "type": "read",
          "options": {}
         },
         {
          "type": "write", The framework uses https://github.com/awslabs/emr-dynamodb-
connector. Please refer to it for explanation of each attribute
          "options": {
           "dynamodb.servicename": "dynamodb",
           "dynamodb.output.tableName": "*****",
           "dynamodb.endpoint": "dynamodb.us-east-1.amazonaws.com",
           "dynamodb.regionid": "us-east-1",
           "dynamodb.throughput.write": "1",
           "dynamodb.throughput.write.percent": "1",
           "mapred.output.format.class":
"org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat",
```

```
            "mapred.input.format.class":
"org.apache.hadoop.dynamodb.read.DynamoDBInputFormat"
           }
          }
         ]
        },
        {
         "hooktype": "jdbc", any JDBC database Sink
         "hookoptions": [
          {
           "type": "read",
           "options": {}
          },
          {
           "type": "write", // Refer to this https://spark.apache.org/docs/latest/sql-data-sources-
jdbc.html for various write options that are supported
           "options": {
            "url": "jdbc:postgresql://testinstance.***.us-east-1.rds.amazonaws.com:5432/testdb",
            "dbtable": "ricohmaster",
            "user": "{postgresuser}",  Use System manager to store the values for the user name
and pwd. The system parameter key should be in the form of /udaf/postgresuser
            "password": "{postgrespwd}",Use System manager to store the values for the user
name and pwd. The system parameter key should be in the form of /udaf/postgrespwd
            "numPartitions": "10"
           }
          }
         ]
        },
        {
         "hooktype": "elasticsearch", Elastic search sink
         "hookoptions": [
          {
           "type": "read",  These are  spark options for reading from elastic search. refer to this
for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
           "options": {
            "index": "index/ricohmaster",
            "es.nodes.wan.only": "true",
            "es.port": "443",
            "es.net.ssl": "true",
            "es.nodes": "https://vpc-testdomain-viq2aomgojnfoeainxo3uvhese.us-east-
1.es.amazonaws.com"
           }
          },
          {
           "type": "write",These are  spark options for writing data to elastic search. refer to this
for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
```

```
            "options": {
              "mode": "Append",
              "save": "index/ricohmaster",
              "es.nodes.wan.only": "true",
              "es.port": "443",
              "es.net.ssl": "true",
              "es.nodes": "https://vpc-testdomain-viq2aomgojnfoeainxo3uvhese.us-east-
1.es.amazonaws.com"
            }
          }
        ]
      }
    ]
  }
 ]
}
```

SourceDestinationConfigSetup – For Source Kinesis

Sample config file for Kinesis Based Source

```
{
  "version": 1,
  "source": {
    "streams": [ //This signifies, the input source is stream based such as kinesis or kafka or file
structured streaming.
      {
        "sourcetype": "levikinesis", //Name could be anything and should be used while submitting
the job to the framework
        "streamtype": "kinesis", //Can be either kafka or kinesis or filestreaming.
        "inputFormat": "json", // Support for Only json is provided
        "flattenJson": "true", // If input data is nested and needs to be flattened. Optional and
default is false
        "streamInputOptions": {
          // Framework uses spark Kinesis structured streaming. Refer to this more details on the
attributes https://databricks.com/blog/2017/08/09/apache-sparks-structured-streaming-with-
amazon-kinesis-on-databricks.html
          "streamName": "devices",
          "subscribe": "levisstream",
          "initialPosition": "earliest",
          "region":"us-east-1"
        },
        "streamOutputOptions": [ //Any Object (S3) or File (HDFS) based Sink can be provided
here.This is useful when you want to enable a datalake or a deltalke using delta or hudi format
          {
```

```
        "outputFormat": "parquet",// The output will be saved as parquet in s3 or hdfs depending on the output directory  specified. Supported values are parquet,avro,orc,delta,hudi,deltacdc (use deltacdc if you  want the input data to be updated in your delta lake),hudicdc (use hudicdc if you are  want the input to be updated in your Hudi lake),json,csv
        "savemode": "Append",// Apend or Overwrite. This is Spark related
        "outputDirectory": "s3://metadata-framwork/real-time-ingestion/ingestion_output/",Output directory to write the processed data to.This could be any object (s3) or file level(hdfs) destination.
        "checkpointLocation": "s3://metadata-framwork/ingestion_checkpoint3/", // Checkpint location for storing offsets. This is mandatory
        "partitioncolumns": "" //Partitioning columns (Spark specific, if the data could be partitioned  based on certain columns, the name of the column can be specified here for fast processing)
      }
    ],
    "streamErrorOptions": {
      // This is not madatory, Only Specify the options here if you want records that fail validation to be persisted in s3 or hdfs. You can add several custom attributes here and then override BaseErrorHandler class to provide you own custom error handling.
      "savemode": "Append",
      "outputFormat": "csv",
      "savemode": "Overwrite",
      "numberOfOutputPartitions": "1",
      "outputDirectory": "s3://metadata-framwork/real-time-ingestion/error_output/"
    },
    "sparkoptions": [ // Optional -  Refer to this doc https://spark.apache.org/docs/latest
      {
        "type": "conf", // Spark configuration settings.
        "options": {}
      },
      {
        "type": "read",
        "options": {}
      },
      {
        "type": "write",
        "options": {}
      }
    ],
      // hooks are kind of like sinks where you can persist the data to and also read data from. Follwing are supported in the framework
      "hooks": [
        {
          "hooktype": "athena", // Support registering the s3 data into glue and retrieving it using Athena. Currently support is provided for json,csv,parquet,orc,avro,delta
          "hookoptions": [
```

```
        {
          "type": "read", // use this to specify options to read the data from sinks. Default class
don't provide an implementation. Override BaseAthenaHandler and specify custom
implementation
          "options": {}
        },
        {
          "type": "write",
          "options": {
            "athenaDB": "deltadb", // Athena database. This has to exist in Athena or
framework will not work
            "athenaTable": "customers", framework will create the table with the specified
name if it doesn't exist
            "athenaDir": "s3://dms-ingestion-encrypted-levis/athena/", This is mandatory and
athena expects it. Specify a path here.
            "region": "us-west-2", Region to use and is mandatory
            "tblproperties": "", specify custom table properties. Refer to AWS athena doc for
examples/
          }
        }
      ]
    },
    {
      "hooktype": "redshift", //Redshift sink.
      "hookoptions": [
        {
          "type": "read",
          "options": {}
        },
        {
          "type": "write", Refer to  link provided for various attributes that can be provided as
spark options for writing dataset to redshift and explanation
https://docs.databricks.com/data/data-sources/aws/amazon-redshift.html
          "options": {
            "url": "jdbc:redshift://demoredshiftcluster.****.us-east-
1.redshift.amazonaws.com:5439/demodb?user={redshiftdbuser}&password={redshiftdbpwd}",
            "aws_iam_role": "arn:aws:iam::142269675036:role/vfcredshiftRole",
            "dbtable": "customerredshiftdata",
            "tempdir": "s3a://temp/data"
          }
        }
      ]
    },
    {
      "hooktype": "dynamodb", // Suports dynamodb sink.
      "hookoptions": [
```

```
        {
          "type": "read",
          "options": {}
        },
        {
          "type": "write", The framework uses https://github.com/awslabs/emr-dynamodb-connector. Please refer to it for explanation of each attribute
          "options": {
            "dynamodb.servicename": "dynamodb",
            "dynamodb.output.tableName": "***",
            "dynamodb.endpoint": "dynamodb.us-east-1.amazonaws.com",
            "dynamodb.regionid": "us-east-1",
            "dynamodb.throughput.write": "1",
            "dynamodb.throughput.write.percent": "1",
            "mapred.output.format.class": "org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat",
            "mapred.input.format.class": "org.apache.hadoop.dynamodb.read.DynamoDBInputFormat"
          }
        }
      ]
    },
    {
      "hooktype": "jdbc", any JDBC database Sink
      "hookoptions": [
        {
          "type": "read",
          "options": {}
        },
        {
          "type": "write", // Refer to this https://spark.apache.org/docs/latest/sql-data-sources-jdbc.html for various write options that are supported
          "options": {
            "url": "jdbc:postgresql://testinstance.cjarlxl94ak0.us-east-1.rds.amazonaws.com:5432/testdb",
            "dbtable": "***",
            "user": "{postgresuser}",  Use System manager to store the values for the user name and pwd. The system parameter key should be in the form of /udaf/postgresuser
            "password": "{postgrespwd}",Use System manager to store the values for the user name and pwd. The system parameter key should be in the form of /udaf/postgrespwd
            "numPartitions": "10"
          }
        }
      ]
    },
    {
```

```
            "hooktype": "elasticsearch", Elastic search sink
            "hookoptions": [
             {
               "type": "read",  These are  spark options for reading from elastic search. refer to this
for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
               "options": {
                 "index": "index/ricohmaster",
                 "es.nodes.wan.only": "true",
                 "es.port": "443",
                 "es.net.ssl": "true",
                 "es.nodes": "https://vpc-testdomain-***.us-east-1.es.amazonaws.com"
               }
             },
             {
               "type": "write",These are  spark options for writing data to elastic search. refer to
this for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
               "options": {
                 "mode": "Append",
                 "save": "index/ricohmaster",
                 "es.nodes.wan.only": "true",
                 "es.port": "443",
                 "es.net.ssl": "true",
                 "es.nodes": "https://vpc-testdomain-***.us-east-1.es.amazonaws.com"
               }
             }
           ]
         }
       ]
     }
   ]
  }
 }
```

SourceDestinationConfigSetup – For Source File Structured Streaming

Sample config file for file based structured streaming. This will be a continuously running job, processing new files as and when they are dumped into the s3 bucket or hdfs.

```
{
  "version": 1,
  "source": {
    "streams": [  //This signifies, the input source is stream based such as kinesis or kafka or file
structured streaming.
      {
```

```
"sourcetype": "customer_cdc_filestreming",//Name could be anything and should be used while submitting the job to the framework
"streamtype": "filestreaming", // This signifies , the job will run continuosly and will listen for file arrivals.
"inputFormat": "csv",
"streamInputOptions": {
 "fileInputDir": "s3://metadata-framwork/cdc/" //hdfs directory  or s3 bucket for reading source files. It will be continously listening for files arriving to this bucket.
},
"streamOutputOptions": [  //Any Object (S3) or File (HDFS) based Sink can be provided here.This is useful when you want to enable a datalake or a deltalke using delta or hudi format
 {
  "outputFormat": "deltacdc", // The output will be saved as delta in s3 or hdfs depending on the output directory  specified. Supported values are parquet,avro,orc,delta,hudi,deltacdc (use deltacdc if you  want the input data to be updated in your delta lake),hudicdc (use hudicdc if you are  want the input data to be updated in your Hudi lake),json,csv
  "savemode": "Append",// Apend or Overwrite. This is Spark related
  "cdccolumn": "op", //This is applicable only when updating the data into delta lake (E.g CDC data from DMS). This column value can be either I (Insert), U(Update), D(Delete). Valid when outputFormat is deltacdc.Ensure the input data has the column that matches the name specified here
  "keyColumn": "id", // This is applicable only when updating the data into delta lake using delta format. This is the key that will be used when updating or deleting the data in delta table. Valid when outputFormat is deltacdc
  "numberOfOutputPartitions": "1",
  "outputDirectory": "s3://dms-ingestion-encrypted-levis/delta_output/customers/",Output directory to write the processed data to.This could be any object (s3) or file level(hdfs) destination.
  "checkpointLocation": "s3://dms-ingestion-encrypted-levis/check_point/" // Checkpint location for storing offsets. This is mandatory
 }
],
"streamErrorOptions": { // Records will be persisted in the below location if validation fails
 "outputFormat": "csv",
 "savemode": "Append",
 "numberOfOutputPartitions": "1",
 "outputDirectory": "s3://error/"
},
"sparkoptions": [
 {
  "type": "conf", // Spark configuration settings.
  "options": {}
 },
 {
  "type": "read",
  "options": {
```

```
          "startingOffsets": "earliest" //earliest or latest (latest means will listen for only new file arrivals)
          }
        },
        {
          "type": "write",
          "options": {}
        }
      ],
      // hooks are kind of like sinks where you can persist the data to and also read data from. Follwing are supported in the framework
      "hooks": [
        {
          "hooktype": "athena", // Support registering the s3 data into glue and retrieving it using Athena. Currently support is provided for json,csv,parquet,orc,avro,delta
          "hookoptions": [
            {
              "type": "read", // use this to specify options to read the data from sinks. Default class don't provide an implementation. Override BaseAthenaHandler and specify custom implementation
              "options": {}
            },
            {
              "type": "write",
              "options": {
                "athenaDB": "deltadb", // Athena database. This has to exist in Athena or framework will not work
                "athenaTable": "customers", framework will create the table with the specified name if it doesn't exist
                "athenaDir": "s3://dms-ingestion-encrypted-levis/athena/", This is mandatory and athena expects it. Specify a path here.
                "region": "us-west-2", Region to use and is mandatory
                "tblproperties": "", specify custom table properties. Refer to AWS athena doc for examples/
              }
            }
          ]
        },
        {
          "hooktype": "redshift", //Redshift sink.
          "hookoptions": [
            {
              "type": "read",
              "options": {}
            },
            {
```

```
            "type": "write", Refer to  link provided for various attributes that can be provided as spark options for writing dataset to redshift and explanation https://docs.databricks.com/data/data-sources/aws/amazon-redshift.html
            "options": {
              "url": "jdbc:redshift://demoredshiftcluster.*****.us-east-1.redshift.amazonaws.com:5439/demodb?user={redshiftdbuser}&password={redshiftdbpwd}",
              "aws_iam_role": "arn:aws:iam::*****:role/vfcredshiftRole",
              "dbtable": "customerredshiftdata",
              "tempdir": "s3a://temp/data"
            }
          }
        ]
      },
      {
        "hooktype": "dynamodb", // Suports dynamodb sink.
        "hookoptions": [
          {
            "type": "read",
            "options": {}
          },
          {
            "type": "write", The framework uses https://github.com/awslabs/emr-dynamodb-connector. Please refer to it for explanation of each attribute
            "options": {
              "dynamodb.servicename": "dynamodb",
              "dynamodb.output.tableName": "*****",
              "dynamodb.endpoint": "dynamodb.us-east-1.amazonaws.com",
              "dynamodb.regionid": "us-east-1",
              "dynamodb.throughput.write": "1",
              "dynamodb.throughput.write.percent": "1",
              "mapred.output.format.class": "org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat",
              "mapred.input.format.class": "org.apache.hadoop.dynamodb.read.DynamoDBInputFormat"
            }
          }
        ]
      },
      {
        "hooktype": "jdbc", any JDBC database Sink
        "hookoptions": [
          {
            "type": "read",
            "options": {}
          },
          {
```

```
            "type": "write", // Refer to this https://spark.apache.org/docs/latest/sql-data-sources-jdbc.html for various write options that are supported
            "options": {
              "url": "jdbc:postgresql://testinstance.*****.us-east-1.rds.amazonaws.com:5432/testdb",
              "dbtable": "****",
              "user": "{postgresuser}",  Use System manager to store the values for the user name and pwd. The system parameter key should be in the form of /udaf/postgresuser
              "password": "{postgrespwd}",Use System manager to store the values for the user name and pwd. The system parameter key should be in the form of /udaf/postgrespwd
              "numPartitions": "10"
            }
          }
        ]
      },
      {
        "hooktype": "elasticsearch", Elastic search sink
        "hookoptions": [
          {
            "type": "read",  These are  spark options for reading from elastic search. refer to this for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
            "options": {
              "index": "index/ricohmaster",
              "es.nodes.wan.only": "true",
              "es.port": "443",
              "es.net.ssl": "true",
              "es.nodes": "https://vpc-testdomain-****.us-east-1.es.amazonaws.com"
            }
          },
          {
            "type": "write",These are  spark options for writing data to elastic search. refer to this for more info https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html
            "options": {
              "mode": "Append",
              "save": "index/ricohmaster",
              "es.nodes.wan.only": "true",
              "es.port": "443",
              "es.net.ssl": "true",
              "es.nodes": "https://vpc-testdomain-****.us-east-1.es.amazonaws.com"
            }
          }
        ]
      }
    ]
  }
]
```

```
  }
}
```

Enabling DeltaLake using the Framework

Please refer to this doc for more information on Open source Delta Lake https://delta.io/
Below config options show how to enable delta lake using the framework for both full load & cdc

***fileOutputOtions or dbOutputOptions or streamOutputOptions, define the output format as delta or deltacdc**

```
"fileOutputOptions":[
  {
    "outputFormat":"delta", //"delta "for full load and "deltacdc" when  updating the delta lake.
    "savemode":"Append", //Append or Overwrite
    "outputDirectory": "s3://hudi-workshop-113877221460/hudi_ingestion_output/test_new2" // This is where Hudi data is written to for enabling data lake
    "cdccolumn": "op", //This is applicable only when updating the data into delta lake (E.g CDC data from DMS). This column value can be either I (Insert), U(Update), D(Delete). Valid when outputFormat is deltacdc. Ensure the input data has the column that matches the name specified here
    "keyColumn": "id", // This is applicable only when updating the data into delta lake using delta format. This is the key to to use when updating or deleting the data in delta table. Valid when outputFormat is deltacdc
  }
 ]
```

Enabling Hudi using the Framework

Please refer to this doc for information on Hudi

https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hudi.html

Below Config options show how to enable delta lake using the framework for both full load and cdc

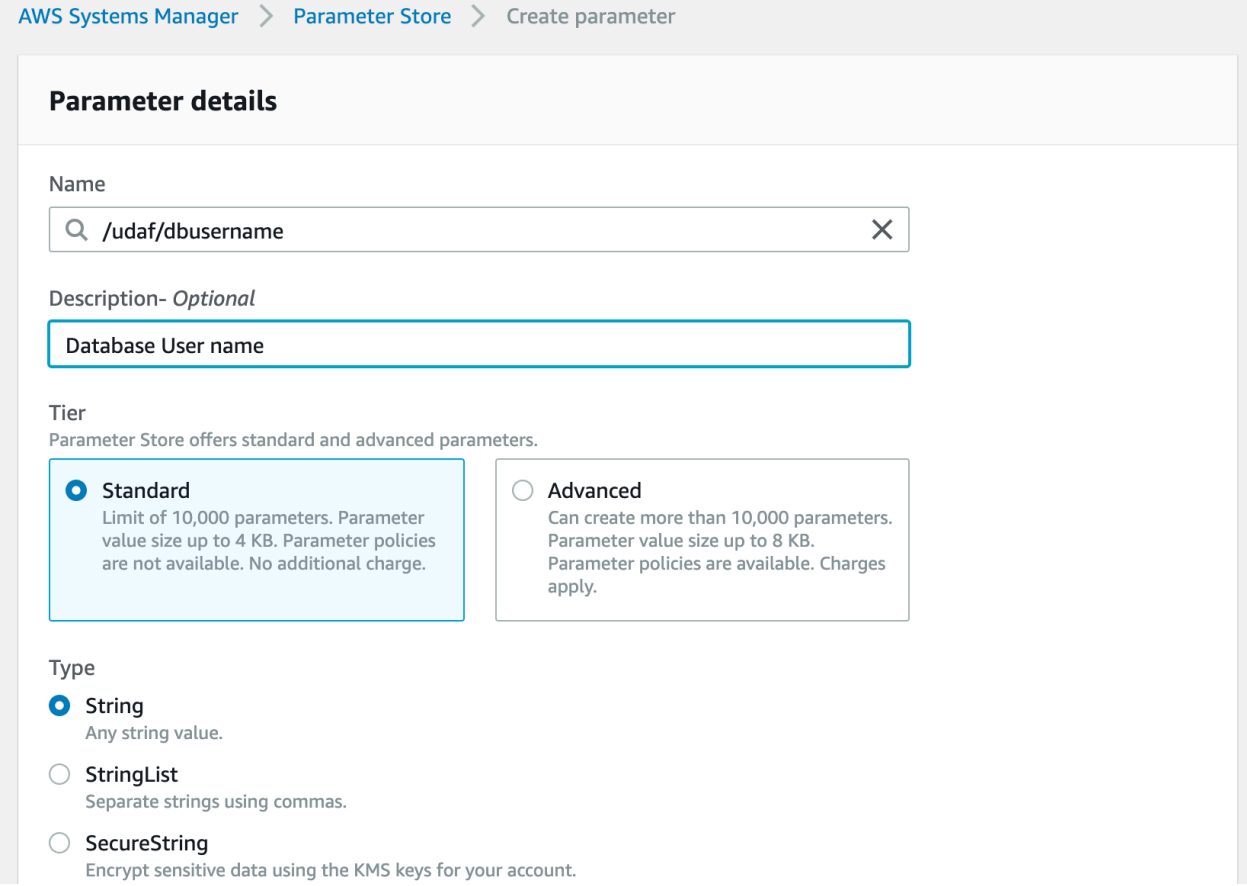
*Define Hudi spark write options under sparkoptions Write

// fileOutputOtions or dbOutputOptions or streamOutputOptions, define the output format as hudi

```
"fileOutputOptions":[
 {
   "outputFormat":"hudi", //hudi for full load and hudicdc for change data capture.
   "savemode":"Append", //Append or Overwrite
   "outputDirectory": "s3://hudi-workshop-113877221460/hudi_ingestion_output/test_new2" //
This is where Hudi data is written to for enabling data lake
   "cdccolumn":"op" // Specify the column name here that would be used for determining insert
(I), updates (U), deletes (D) in change data capture file .Applicable only when outputFormat is
"hudicdc".Ensure the input data has the column that matches the name specified here
 }
 ],

"sparkoptions":[


 {
    "type": "write", Refer to this doc for more info regarding  the below options
https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hudi.html
    "options":
    {
      "hoodie.datasource.write.recordkey.field":"id",
      "hoodie.datasource.write.partitionpath.field":"state_code",
      "hoodie.datasource.write.precombine.field":"account_no",
      "hoodie.datasource.hive_sync.enable":"true",

      "hoodie.datasource.hive_sync.table":"test_new2",
      "hoodie.datasource.hive_sync.partition_fields":"state_code",
      "hoodie.datasource.hive_sync.assume_date_partitioning":"false",

"hoodie.datasource.hive_sync.partition_extractor_class":"org.apache.hudi.hive.MultiPartKeysV
alueExtractor",
      "hoodie.datasource.write.operation":"insert",

      "hoodie.table.name":"test_new2",
      "hoodie.base.path":"s3://hudi-workshop-113877221460/hudi_ingestion_output/test_new2"

    }
 }
 ]
```

Systems Managers Parameter store

Framework Config file (SourceDestinationConfigSetup.json) supports System Manager Parameter Store. Any secrets such as user name and password should be stored in System Managers Parameter Store as shown below. For e.g the below screenshot attached stores the dbuser name value with the key as udaf/dbusername. This can be referred in config files as {dbusername}. Similarly password can be stored in SSM with key as /udaf/dbpassword and can be referred in config files as {dbpassword} . Please ensure the keys start with /udaf/. Currently this support is only provided for config files stored in dynamodb



Supports for Passing values to Config files

Framework supports passing values to config files. You can define placeholder in config files as {key1} and pass the values for key 1 as key1#value1. If there are multiple placeholders in config files such as {key1}, {key2}..{KeyN}, values for the keys can be passed as key1#value1|key2#value2|keyN#valueN. Currently this support is only provided for config files stored in dynamodb.

Framework Runtime and Processing Jobs

Running from UI

The framework runs on EMR. You can use the UI to submit a job for the framework to process. If done through UI, the values to the framework are passed in the form of JSON shown below.

{
"jarPath":"s3://metadata-framwork/code/udaf-levis-v1.jar" //Path to the Jar file,
"sourceType":"kafka#levikafka" //(Source(accepted values are database,kafka,kinesis,filestreaming,file)#("sourcetype": value of the config file))),
"region":"us-west-2",
"key":"dynamodbkeyors3filepath" //Key used for stroing the config file,
"emrInstanceId":"j-39Q84P8LAWU8L" //EMR Instance Id,
"configPlaceholders":"key1#val1|key2#val2" (In cofng file you can define placeholder as {key1} {key2}, they will be replaced with val1 and val2)
}

Running from EMR Console & Master Node

Job can be submitted via steps in AWS EMR console or by directly submitting a job using the command shown below through an orchestrator or from the master node.

spark-submit --class com.udaf.datalake.DataLakeMainApplication s3://metadata-framwork/code/udaf-levis-v1.jar regionname dynamodbkeyors3path kakfa#levikafka key1#val1|key2#val2

Last argument is for replacing config file placeholders and is optional.