sparkly Documentation

Release 1.0.0

Tubular

Feb 13, 2017

Contents

Sparkly is a lib which makes usage of pyspark more convenient and consistent.

A brief tour on Sparkly features by example:

```
# The main thing and the entry point of the Sparkly lib is SparklyContext
from sparkly import SparklyContext

class CustomSparklyContext(SparklyContext):
   # Install custom spark packages instead of hacking with `spark-submit`:
   packages = ['com.databricks:spark-csv_2.10:1.4.0']

   # Install jars and import udfs from them as simple as:
   jars = ['/path/to/brickhouse-0.7.1.jar'],
   udfs = {
     'collect_max': 'brickhouse.udf.collect.CollectMaxUDAF',
   }


ctx = CustomSparklyContext()

# Operate with easily interchangable URL-like data source definitions,
# instead of untidy default interface:
df = ctx.read_ext.by_url('mysql://<my-sql.host>/my_database/my_database')
df.write_ext('parquet:s3://<my-bucket>/<path>/data?partition_by=<field_name1>,<field_
↪name1>')

# Operate with Hive Metastore with convenient python api,
# instead of verbose Hive queries:
ctx.hms.create_table(
   'my_custom_table',
   df,
   location='s3://<my-bucket>/<path>/data',
   partition_by=[<field_name1>,<field_name1>],
   output_format='parquet'
)

# Make integration testing more convenient with Fixtures and base test classes:
# SparklyTest, SparklyGlobalContextTest, instead of implementing you own spark testing
# mini frameworks:
class TestMyShinySparkScript(SparklyTest):
   fixtures = [
      MysqlFixture('<my-testing-host>', '<test-user>', '<test-pass>', '/path/to/data.
↪sql', '/path/to/clear.sql')
   ]

   def test_job_works_with_mysql(self):
      df = self.hc.read_ext.by_url('mysql://<my-testing-host>/<test-db>/<test-table>?
↪user=<test-usre>&password=<test-password>')
      res_df = my_shiny_script(df)
      self.assertDataFrameEqual(
         res_df,
         {'fieldA': 'DataA', 'fieldB': 'DataB', 'fieldC': 'DataC'},
      )
```

CHAPTER 1

Sparkly Context

1.1 About Sparkly Context

`SparklyContext` class is the main class of the Sparkly library. It encompasses all of this library's functionality. Most of times you want to subclass it to define the various options you desire through class attributes.

Sparkly context have links to other extras of the lib:

Attribute	Link to the doc
`read_ext`	*Read/write utilities for DataFrames*
`hms`	*Hive Metastore Utils*

`Dataframe` pyspark class is also monkey patched with `write_ext` (*Read/write utilities for DataFrames*) attribute for convenient writing.

1.2 Use cases

Setup Custom options

Why: Sometimes you need to customize your spark context more than default. We prefer to define Spark options declaratively rather than using getter/setters for each option.

For example: some useful usecases of this are:

- Optimizing shuffling options, like `spark.sql.shuffle.partitions`
- Setup custom Hive Metastore instead of local.
- Package specific options, like `spark.hadoop.avro.mapred.ignore.inputs.without.extension`

```
from sparkly import SparklyContext
class OwnSparklyContext(SparklyContext):
    options = {
        # Increasing default amount of partitions for shuffling.
        'spark.sql.shuffle.partitions': 1000,
        # setup remote Hive Metastore.
        'hive.metastore.uris': 'thrift://<host1>:9083,thrift://<host2>:9083',
        # setup avro reader to not ignore files without `avro` extension
        'spark.hadoop.avro.mapred.ignore.inputs.without.extension': 'false',
    }

# you can also overwrite or add some options at initialisation time.
ctx = OwnSparklyContext({ ...initialize-time options... })

# you still can update options later if you need.
ctx.setConf('key', 'value')
```

Installing spark dependencies

Why: The default mechanism requires that dependencies be declared when the spark job is submitted, typically on the command line. We prefer a code-first approach where dependencies are actually declared as part of the job.

For example: You want to install cassandra connector to read data for one of your tables.

```
from sparkly import SparklyContext
class OwnSparklyContext(SparklyContext):
    # specifying spark dependencies.
    packages = [
        'datastax:spark-cassandra-connector:1.5.0-M3-s_2.10',
    ]

# dependencies will be installed in context initialization.
ctx = OwnSparklyContext()

# Here is how you now can obtain a Dataframe representing yout cassandra table.
df = ctx.read_ext.by_url('cassandra://<cassandra-host>'
                         '/<db>/<talbe>?consistency=QUORUM&parallelism=16')
```

CHAPTER 4

Using UDFs

Why: By default to use udfs in Hive queries you need to add jars and specify which udfs you wish to use using verbose Hive queries.

For example: You want to import udfs from (brickhouse)[https://github.com/klout/brickhouse] Hive udfs lib.

```
from pyspark.sql.types import IntegerType
from sparkly import SparklyContext

def my_own_udf(item):
    return len(item)

class OwnSparklyContext(SparklyContext):
    # specifying spark dependencies.
    jars = [
        '/path/to/brickhouse.jar'
    ]
    udfs = {
        'collect_max': 'brickhouse.udf.collect.CollectMaxUDAF',
        'my_udf': (my_own_udf, IntegerType())
    }

# dependencies will be installed in context initialization.
ctx = OwnSparklyContext()

ctx.sql('SELECT collect_max(amount) FROM my_data GROUP BY ...')
ctx.sql('SELECT my_udf(amount) FROM my_data')
```

class sparkly.context.**SparklyContext**(*additional_options=None*)

Wrapper around HiveContext to simplify definition of options, packages, JARs and UDFs.

Example:

```
from pyspark.sql.types import IntegerType
import sparkly


class MyContext(sparkly.SparklyContext):
    options = {'spark.sql.shuffle.partitions': '2000'}
    packages = ['com.databricks:spark-csv_2.10:1.4.0']
    jars = ['../path/to/brickhouse-0.7.1.jar']
    udfs = {
        'collect_max': 'brickhouse.udf.collect.CollectMaxUDAF',
        'my_python_udf': (lambda x: len(x), IntegerType()),
```

```
    }


hc = MyContext()
hc.read_ext.cassandra(...)
```

options
dict[str,str] – Configuration options that are passed to SparkConf. See the list of possible options.

packages
list[str] – Spark packages that should be installed. See https://spark-packages.org/

jars
list[str] – Full paths to jar files that we want to include to the context. E.g. a JDBC connector or a library with UDF functions.

udfs
dict[str,str|typing.Callable] – Register UDF functions within the context. Key - a name of the function, Value - either a class name imported from a JAR file

or a tuple with python function and its return type.

has_jar (*jar_name*)
Check if the jar is available in the context.

Parameters **jar_name** (*str*) – E.g. "mysql-connector-java"

Returns bool

has_package (*package_prefix*)
Check if the package is available in the context.

Parameters **package_prefix** (*str*) – E.g. "org.elasticsearch:elasticsearch-spark"

Returns bool

Read/write utilities for DataFrames

Sparkly isn't trying to replace any of existing storage connectors. The goal is to provide a simplified and consistent api across a wide array of storage connectors. We also added the way to work with *abstract data sources*, so you can keep your code agnostic to the storages you use.

5.1 Cassandra

Sparkly relies on the official spark cassandra connector and was successfully tested in production using versions 1.5.x and 1.6.x.

Package	https://spark-packages.org/package/datastax/spark-cassandra-connector
Configuration	https://github.com/datastax/spark-cassandra-connector/blob/b1.6/doc/reference.md

```
from sparkly import SparklyContext


class MyContext(SparklyContext):
    # Feel free to play with other versions
    packages = ['datastax:spark-cassandra-connector:1.6.1-s_2.10']

hc = MyContext()

# To read data
df = hc.read_ext.cassandra('localhost', 'my_keyspace', 'my_table')
# To write data
df.write_ext.cassandra('localhost', 'my_keyspace', 'my_table')
```

5.2 CSV

Sparkly relies on the csv connector provided by Databricks.

Note: Spark 2.x supports CSV out of the box. We highly recommend you to use the official api.

Package	https://spark-packages.org/package/databricks/spark-csv
Configuration	https://github.com/databricks/spark-csv#features

```
from sparkly import SparklyContext


class MyContext(SparklyContext):
    # Feel free to play with other versions
    packages = ['com.databricks:spark-csv_2.10:1.4.0']

hc = MyContext()

# To read data
df = hc.read_ext.csv('/path/to/csv/file.csv', header=True)
# To write data
df.write_ext.csv('/path/to/csv/file.csv', header=False)
```

5.3 Elastic

Sparkly relies on the official elastic spark connector and was successfully tested in production using versions 2.2.x and 2.3.x.

Package	https://spark-packages.org/package/elastic/elasticsearch-hadoop
Configuration	https://www.elastic.co/guide/en/elasticsearch/hadoop/current/configuration.html

```
from sparkly import SparklyContext


class MyContext(SparklyContext):
    # Feel free to play with other versions
    packages = ['org.elasticsearch:elasticsearch-spark_2.10:2.3.0']

hc = MyContext()

# To read data
df = hc.read_ext.elastic('localhost', 'my_index', 'my_type', query='?q=awesomeness')
# To write data
df.write_ext.elastic('localhost', 'my_index', 'my_type')
```

5.4 MySQL

Basically, it's just a high level api on top of the native jdbc reader and jdbc writer.

Jars	https://dev.mysql.com/downloads/connector/j/
Configuration	https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-configuration-properties.html

Note: Sparkly doesn't contain any jars inside, so you will have to take care of this. Java connectors for mysql could be found on https://dev.mysql.com/downloads/connector/j/. We usually place them within our service/package codebase in *resources* directory. It's not the best idea to place binaries within a source code, but it's pretty convenient.

```
from sparkly import SparklyContext
from sparkly.utils import absolute_path
```

```
class MyContext(SparklyContext):
    # Feel free to play with other versions.
    jars = [absolute_path(__file__, './path/to/mysql-connector-java-5.1.39-bin.jar')]

hc = MyContext()

# To read data
df = hc.read_ext.mysql('localhost', 'my_database', 'my_table',
                       options={'user': 'root', 'password': 'root'})
# To write data
df.write_ext.mysql('localhost', 'my_database', 'my_table', options={
    'user': 'root',
    'password': 'root',
    'rewriteBatchedStatements': 'true',  # improves write throughput dramatically
})
```

5.5 Kafka

Sparkly's reader and writer for Kafka are built on top of the official spark package for Kafka and python library kafka-python . The first one allows us to read data efficiently, the second covers a lack of writing functionality in the official distribution.

Package	https://mvnrepository.com/artifact/org.apache.spark/spark-streaming-kafka_2.10
Configuration	http://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html

Note:

- To use the Kafka functionality **sparkly** needs the **kafka-python** library which is an optional dependency. So you need to install **sparkly** with **kafka** extras: ` pip install sparkly[kafka] `
- When working via DataFrame api, it is expected to be organized as a structure with two top level keys for key and value: ` schema=StructType([StructField('key',...),StructField('value',...)]) ` and then ` df = ctx.createDataFrame(data,schema=schema) `
- This functionality was tested on Kafka version **0.10.x**, which is the most recent to the moment. It was not tested on Kafka **0.8.x** for which needs another package version, which does not have Api used in Sparkly.

```
import json
from sparkly import SparklyContext

class MyContext(SparklyContext):
    packages = [
        'org.apache.spark:spark-streaming-kafka_2.10:1.6.1',
    ]

hc = MyContext()

# To read data from kafka in json as Dataframe.

#   1. Define the schema of the data you read.
df_schema = StructType([
    StructField('key', StructType([
        StructField('id', StringType(), True)
    ])),
```

```
    StructField('value', StructType([
        StructField('name', StringType(), True),
        StructField('surname', StringType(), True),
    ]))
])

#   2. Specify the schema as the reader parameter.
df = hc.read_ext.kafka(
    'kafka.host',
    topic='my.topic',
    key_deserializer=lambda item: json.loads(item.decode('utf-8')),
    value_deserializer=lambda item: json.loads(item.decode('utf-8')),
    schema=df_schema,
)

# To write data to kafka in json from Dataframe
df.write_ext.kafka(
    'kafka.host',
    topic='my.topic',
    key_serializer=lambda item: json.dumps(item).encode('utf-8'),
    value_serializer=lambda item: json.dumps(item).encode('utf-8'),
)
```

5.6 Universal reader/writer

The *DataFrame* abstraction is really powerful when it comes to transformations. You can shape your data from various storages using exactly the same api. For instance, you can join data from Cassandra with data from Elasticsearch and write the result to MySQL.

The only problem - you have to explicitly define sources (or destinations) in order to create (or export) a *DataFrame*. But the source/destination of data doesn't really change the logic of transformations (if the schema is preserved). To solve the problem, we decided to add the universal api to read/write *DataFrames*:

```
from sparkly import SparklyContext

class MyContext(SparklyContext):
    packages = [
        'datastax:spark-cassandra-connector:1.6.1-s_2.10',
        'com.databricks:spark-csv_2.10:1.4.0',
        'org.elasticsearch:elasticsearch-spark_2.10:2.3.0',
    ]

hc = MyContext()

# To read data
df = hc.read_ext.by_url('cassandra://localhost/my_keyspace/my_table?consistency=ONE')
df = hc.read_ext.by_url('csv:s3://my-bucket/my-data?header=true')
df = hc.read_ext.by_url('elastic://localhost/my_index/my_type?q=awesomeness')
df = hc.read_ext.by_url('parquet:hdfs://my.name.node/path/on/hdfs')

# To write data
df.write_ext.by_url('cassandra://localhost/my_keyspace/my_table?consistency=QUORUM&
↪parallelism=8')
df.write_ext.by_url('csv:hdfs://my.name.node/path/on/hdfs')
df.write_ext.by_url('elastic://localhost/my_index/my_type?parallelism=4')
```

```
df.write_ext.by_url('parquet:s3://my-bucket/my-data?header=false')
```

5.7 Controlling the load

From the official documentation:

> Don't create too many partitions in parallel on a large cluster; otherwise Spark might crash your external database systems.
>
> link: <https://spark.apache.org/docs/2.0.1/api/java/org/apache/spark/sql/DataFrameReader.html>

It's a very good advice, but in practice it's hard to track the number of partitions. For instance, if you write a result of a join operation to database the number of splits might be changed implicitly via *spark.sql.shuffle.partitions*.

To prevent us from shooting to the foot, we decided to add *parallelism* option for all our readers and writers. The option is designed to control a load on a source we write to / read from. It's especially useful when you are working with data storages like Cassandra, MySQL or Elastic. However, the implementation of the throttling has some drawbacks and you should be aware of them.

The way we implemented it is pretty simple: we use *coalesce* on a dataframe to reduce an amount of tasks that will be executed in parallel. Let's say you have a dataframe with 1000 splits and you want to write no more than 10 task in parallel. In such case *coalesce* will create a dataframe that has 10 splits with 100 original tasks in each. An outcome of this: if any of these 100 tasks fails, we have to retry the whole pack in 100 tasks.

Read more about coalesce

5.8 Reader API documentation

class `sparkly.reader.`**`SparklyReader`**(*hc*)

A set of tools to create DataFrames from the external storages.

> **Note:** This is a private class to the library. You should not use it directly. The instance of the class is available under *SparklyContext* via *read_ext* attribute.

`by_url`(*url*)

Create a dataframe using *url*.

The main idea behind the method is to unify data access interface for different formats and locations. A generic schema looks like:

```
format:[protocol:]//host[:port][/location][?configuration]
```

Supported formats:

- CSV `csv://`
- Cassandra `cassandra://`
- Elastic `elastic://`
- MySQL `mysql://`
- Parquet `parquet://`
- Hive Metastore table `table://`

Query string arguments are passed as parameters to the relevant reader.

For instance, the next data source URL:

```
cassandra://localhost:9042/my_keyspace/my_table?consistency=ONE
    &parallelism=3&spark.cassandra.connection.compression=LZ4
```

Is an equivalent for:

```
hc.read_ext.cassandra(
    host='localhost',
    port=9042,
    keyspace='my_keyspace',
    table='my_table',
    consistency='ONE',
    parallelism=3,
    options={'spark.cassandra.connection.compression': 'LZ4'},
)
```

More examples:

```
table://table_name
csv:s3://some-bucket/some_directory?header=true
csv://path/on/local/file/system?header=false
parquet:s3://some-bucket/some_directory
elastic://elasticsearch.host/es_index/es_type?parallelism=8
cassandra://cassandra.host/keyspace/table?consistency=QUORUM
mysql://mysql.host/database/table
```

Parameters **`url`** (*`str`*) – Data source URL.

Returns pyspark.sql.DataFrame

`cassandra` (*host*, *keyspace*, *table*, *consistency=None*, *port=None*, *parallelism=None*, *options=None*)

Create a dataframe from a Cassandra table.

Parameters

- **`host`** (*`str`*) – Cassandra server host.
- **`keyspace`** (*`str`*) –
- **`table`** (*`str`*) – Cassandra table to read from.
- **`consistency`** (*`str`*) – Read consistency level: `ONE`, `QUORUM`, `ALL`, etc.
- **`port`** (*`int | None`*) – Cassandra server port.
- **`parallelism`** (*`int | None`*) – The max number of parallel tasks that could be executed during the read stage (see *Controlling the load*).
- **`options`** (*`dict[str, str] | None`*) – Additional options for *org.apache.spark.sql.cassandra* format (see configuration for *Cassandra*).

Returns pyspark.sql.DataFrame

`csv` (*path*, *custom_schema=None*, *header=True*, *parallelism=None*, *options=None*)

Create a dataframe from a CSV file.

Parameters

- **`path`** (*`str`*) – Path to the file or directory.

- **custom_schema** (*pyspark.sql.types.DataType*) – Force custom schema.
- **header** (*bool*) – The first row is a header.
- **parallelism** (*int|None*) – The max number of parallel tasks that could be executed during the read stage (see *Controlling the load*).
- **options** (*dict[str,str]|None*) – Additional options for *com.databricks.spark.csv* format. (see configuration for *CSV*).

Returns pyspark.sql.DataFrame

elastic(*host*, *es_index*, *es_type*, *query=''*, *fields=None*, *port=None*, *parallelism=None*, *options=None*)

Create a dataframe from an ElasticSearch index.

Parameters

- **host** (*str*) – Elastic server host.
- **es_index** (*str*) – Elastic index.
- **es_type** (*str*) – Elastic type.
- **query** (*str*) – Pre-filter es documents, e.g. '?q=views:>10'.
- **fields** (*list[str]|None*) – Select only specified fields.
- **port** (*int|None*) –
- **parallelism** (*int|None*) – The max number of parallel tasks that could be executed during the read stage (see *Controlling the load*).
- **options** (*dict[str,str]*) – Additional options for *org.elasticsearch.spark.sql* format (see configuration for *Elastic*).

Returns pyspark.sql.DataFrame

kafka(*host*, *topic*, *offset_ranges=None*, *key_deserializer=None*, *value_deserializer=None*, *schema=None*, *port=9092*, *parallelism=None*, *options=None*)

Creates dataframe from specified set of messages from Kafka topic.

Defining ranges:

- If *offset_ranges* is specified it defines which specific range to read.
- If *offset_ranges* is omitted it will auto-discover it's partitions.

The *schema* parameter, if specified, should contain two top level fields: *key* and *value*.

Parameters *key_deserializer* and *value_deserializer* are callables which get's bytes as input and should return python structures as output.

Parameters

- **host** (*str*) – Kafka host.
- **topic** (*str|None*) – Kafka topic to read from.
- **offset_ranges** (*list[(int, int, int)*) – List of partition ranges [(partition, start_offset, end_offset)].
- **key_deserializer** (*function*) – Function used to deserialize the key.
- **value_deserializer** (*function*) – Function used to deserialize the value.
- **schema** (*pyspark.sql.types.StructType*) – Schema to apply to create a Dataframe.

- **port** (*int*) – Kafka port.
- **parallelism** (*int | None*) – The max number of parallel tasks that could be executed during the read stage (see *Controlling the load*).
- **options** (*dict | None*) – Additional kafka parameters, see KafkaUtils.createRDD docs.

Returns pyspark.sql.DataFrame

Raises `InvalidArgumentError`

mysql (*host*, *database*, *table*, *port=None*, *parallelism=None*, *options=None*)
Create a dataframe from a MySQL table.

Should be usable for rds, aurora, etc. Options should include user and password.

Parameters

- **host** (*str*) – MySQL server address.
- **database** (*str*) – Database to connect to.
- **table** (*str*) – Table to read rows from.
- **port** (*int | None*) – MySQL server port.
- **parallelism** (*int | None*) – The max number of parallel tasks that could be executed during the read stage (see *Controlling the load*).
- **options** (*dict[str, str] | None*) – Additional options for JDBC reader (see configuration for *MySQL*).

Returns pyspark.sql.DataFrame

5.9 Writer API documentation

class sparkly.writer.**SparklyWriter** (*df*)
A set of tools to write DataFrames to the external storages.

Note: We don't expect you to be using the class directly. The instance of the class is available under *DataFrame* via *write_ext* attribute.

by_url (*url*)
Write a dataframe to a destination specified by *url*.

The main idea behind the method is to unify data export interface for different formats and locations. A generic schema looks like:

```
format:[protocol:]//host[:port][/location][?configuration]
```

Supported formats:

- CSV `csv://`
- Cassandra `cassandra://`
- Elastic `elastic://`
- MySQL `mysql://`

- Parquet `parquet://`

Query string arguments are passed as parameters to the relevant writer.

For instance, the next data export URL:

```
elastic://localhost:9200/my_index/my_type?&parallelism=3&mode=overwrite
    &es.write.operation=upsert
```

Is an equivalent for:

```
hc.read_ext.elastic(
    host='localhost',
    port=9200,
    es_index='my_index',
    es_type='my_type',
    parallelism=3,
    mode='overwrite',
    options={'es.write.operation': 'upsert'},
)
```

More examples:

```
csv:s3://some-s3-bucket/some-s3-key?partitionBy=date,platform
cassandra://cassandra.host/keyspace/table?consistency=ONE&mode=append
parquet:///var/log/?partitionBy=date
elastic://elastic.host/es_index/es_type
mysql://mysql.host/database/table
```

Parameters **url** (*str*) – Destination URL.

cassandra(*host*, *keyspace*, *table*, *consistency=None*, *port=None*, *mode=None*, *parallelism=None*, *options=None*)
Write a dataframe to a Cassandra table.

Parameters

- **host** (*str*) – Cassandra server host.
- **keyspace** (*str*) – Cassandra keyspace to write to.
- **table** (*str*) – Cassandra table to write to.
- **consistency** (*str | None*) – Write consistency level: `ONE`, `QUORUM`, `ALL`, etc.
- **port** (*int | None*) – Cassandra server port.
- **mode** (*str | None*) – Spark save mode, http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes
- **parallelism** (*int | None*) – The max number of parallel tasks that could be executed during the write stage (see *Controlling the load*).
- **options** (*dict[str, str]*) – Additional options to *org.apache.spark.sql.cassandra* format (see configuration for *Cassandra*).

csv(*path*, *header=False*, *mode=None*, *partitionBy=None*, *parallelism=None*, *options=None*)
Write a dataframe to a CSV file.

Parameters

- **path** (*str*) – Path to the output directory.

- **header** (*bool*) – First row is a header.
- **mode** (*str|None*) – Spark save mode, http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes
- **partitionBy** (*list[str]*) – Names of partitioning columns.
- **parallelism** (*int|None*) – The max number of parallel tasks that could be executed during the write stage (see *Controlling the load*).
- **options** (*dict[str, str]*) – Additional options to *com.databricks.spark.csv* format (see configuration for *CSV*).

elastic (*host*, *es_index*, *es_type*, *port=None*, *mode=None*, *parallelism=None*, *options=None*)

Write a dataframe into an ElasticSearch index.

Parameters

- **host** (*str*) – Elastic server host.
- **es_index** (*str*) – Elastic index.
- **es_type** (*str*) – Elastic type.
- **port** (*int|None*) –
- **mode** (*str|None*) – Spark save mode, http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes
- **parallelism** (*int|None*) – The max number of parallel tasks that could be executed during the write stage (see *Controlling the load*).
- **options** (*dict[str, str]*) – Additional options to *org.elasticsearch.spark.sql* format (see configuration for *Elastic*).

kafka (*host*, *topic*, *key_serializer*, *value_serializer*, *port=9092*, *parallelism=None*, *options=None*)

Writes dataframe to kafka topic.

The schema of the dataframe should conform the pattern:

```
>>>  StructType([
...      StructField('key', ...),
...      StructField('value', ...),
...  ])
```

Parameters *key_serializer* and *value_serializer* are callables which get's python structure as input and should return bytes of encoded data as output.

Parameters

- **host** (*str*) – Kafka host.
- **topic** (*str*) – Topic to write to.
- **key_serializer** (*function*) – Function to serialize key.
- **value_serializer** (*function*) – Function to serialize value.
- **port** (*int*) – Kafka port.
- **parallelism** (*int|None*) – The max number of parallel tasks that could be executed during the write stage (see *Controlling the load*).
- **options** (*dict|None*) – Additional options.

mysql (*host*, *database*, *table*, *port=None*, *mode=None*, *parallelism=None*, *options=None*)

Write a dataframe to a MySQL table.

Should be usable for rds, aurora, etc. Options should include user and password.

Parameters

- **host** (*str*) – MySQL server address.
- **database** (*str*) – Database to connect to.
- **table** (*str*) – Table to read rows from.
- **mode** (*str|None*) – Spark save mode, http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes
- **parallelism** (*int|None*) – The max number of parallel tasks that could be executed during the write stage (see *Controlling the load*).
- **options** (*dict*) – Additional options for JDBC writer (see configuration for *MySQL*).

sparkly.writer.**attach_writer_to_dataframe**()

A tiny amount of magic to attach write extensions.

Hive Metastore Utils

6.1 About Hive Metastore

Hive metastore is a database storing metadata about Hive tables, which you operate in your Sparkly (Hive) Context. Read more about Hive Metastore

To configure a SparklyContext to work with your Hive Metastore, you have to set *hive.metastore.uris* option. You can do this via hive-site.xml file in spark config ($SPARK_HOME/conf/hive-site.xml), like this:

```
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://<n.n.n.n>:9083</value>
  <description>IP address (or fully-qualified domain name) and port of the metastore host</description>
</property>
```

or set it dynamically in SparklyContext options, like this:

```
class MySparklyContext(SparklyContext):
    options = {
        'hive.metastore.uris': 'thrift://<n.n.n.n>:9083',
    }
```

After this your sparkly context will operate on the configured Hive Metastore.

6.2 Use cases

6.2.1 Check for existence

Why: sometimes logic of your program may depend on existance of a table in a Hive Metastore. **For example**: to know if we should create a new table, or we need to replace an existing one.

```
from sparkly import SparklyContext
hc = SparklyContext()
assert(hc.hms.table('my_table').exists() in {True, False})
```

6.2.2 Create a table in hive metastore

Why: You may want to unify access to all your data via Hive Metastore tables. To do this you generally need to perform 'CREATE TABLE ...' statement for each data you have. To simplify this we implemented this method which generates the CREATE TABLE statements by passed parameters and executes them on Hive Metastore.

Input: table name, data on some data storage hdfs or s3, stored in some specific format (parquet, avro, csv, etc.)

Output: table available in HiveMetastore

```
from sparkly import SparkeContext
# input
hc = SparklyContext()
df = hc.read_ext.by_url('parquet:s3://path/to/data/')
# operation
hc.hms.create_table(
     'new_shiny_table',
     df,
     location='s3://path/to/data/',
     partition_by=['partition', 'fields'],
     output_format='parquet'
)
new_df = hc.read_ext.by_url('table://new_shiny_table')
```

6.2.3 Replace table in hive metastore

Why: some times you want to quickly replace data underlying some table in Hive Metastore. For example, if you exported a new snapshot of your data to a new location and want to point Hive Metastore table to this new location. This method avoids downtime during which data in the table won't be accessible. It first creates a new table separately (slow operation) and then operating on meta data (quick renaming operation).

Input: table name to replace, data schema, location, partitioning, format.

Output: updated table in Hive Metastore.

```
from sparkly import SparkeContext
# input
hc = SparklyContext()
df = hc.read_ext.by_url('csv:s3://path/to/data/new/')
# operation
table = hc.hms.replace_table(
    'old_table',
    df,
    location='s3://path/to/data/',
    partition_by=['partition', 'fields'],
)
```

6.2.4 Operating on table properties

Why: some times you want to assign some metadata to your table like creation time, last update, purpose, data source, etc. Table properties is a perfect place for this. Generally you have to execute Sql queries and parse results to manipulate table properties. We implemented a more convenient interface on top of this.

Set/Get property

```
from sparkly import SparklyContext
hc = SparklyContext()
table = hc.hms.table('my_table')
table.set_property('foo', 'bar')
assert table.get_property('foo') == 'bar'
assert table.get_all_properties() == {'foo': 'bar'}
```

Note properties may only have string keys and values, so you have to think on serialization from other data types by yourself.

6.3 API documentation

class `sparkly.hive_metastore_manager.`**`SparklyHiveMetastoreManager`**(*hc*)

A set of tools to interact with HiveMetastore.

`create_table`(*table_name*, *schema*, *location*, *partition_by=None*, *table_format=None*, *properties=None*)

Creates table in Hive Metastore.

Parameters

- **`table_name`** (*`str`*) – name of new Table.
- **`schema`** (*`pyspark.sql.dataframe.DataFrame`*) – schema.
- **`location`** (*`str`*) – location of data.
- **`partition_by`** (*`list`*) – partitioning columns.
- **`table_format`** (*`str`*) – default is parquet.
- **`properties`** (*`dict`*) – properties to assign to the table.

Returns Table

`get_all_tables`()

Returns all tables available in metastore.

Returns list

`replace_table`(*table_name*, *schema*, *location*, *partition_by=None*, *table_format=None*)

Replaces table *table_name* with data represented by schema, location.

Parameters

- **`table_name`** (*`str`*) – Table name.
- **`schema`** (*`pyspark.sql.dataframe.DataFrame`*) – schema.
- **`location`** (*`str`*) – data location, ex.: s3://path/tp/data.
- **`partition_by`** (*`list`*) – fields the data partitioned by.

Returns Table

class `sparkly.hive_metastore_manager.`**`Table`**(*hms*, *table_name*)

Represents a table in HiveMetastore.

Provides meta data operations on a Table.

`df`()

Returns dataframe for the managed table.

Returns pyspark.sql.dataframe.DataFrame

exists()
Checks if table exists.

Returns bool

get_all_properties()
Returns all table properties.

Returns Property names to values.

Return type dict

get_property(*name*, *to_type=None*)
Gets table property.

Parameters

- **name** (*str*) – Name of the property.
- **to_type** (*type*) – Type to coarce to, str by default.

Returns any

set_property(*name*, *value*)
Sets table property.

Parameters

- **name** (*str*) – Name of the property.
- **value** (*str*) – Value of the proporty.

Returns Self.

Return type *Table*

Schema management

This package contains utilities for converting string to spark schema definition. This might be useful for:

- Specifying schema as (command line) parameter.
- More convenient interface for specifying schema by hands.

7.1 Use cases

7.1.1 Init Dataframe from data

Why: Sometimes you know the schema of the data, but format is not recognized by spark. Then you can read it as raw python data and apply the known schema to it. Sparkly utility will make schema definition easy and not hardcoded.

For example: You have custom format file without any type information, but types could be are easily derived.

```
from sparkly.schema_parser import generate_structure_type, parse_schema

data = ... parse data from file ...
schema_as_string = 'name:string|age:int'  # Note: you can get this from command line,␣
↪for example
spark_schema = generate_structure_type(parse_schema(schema_as_string))
df = ctx.createDataframe(data, spark_schema)
```

`sparkly.schema_parser.`**`parse`**`(`*schema*`)`

Converts string to Sparke schema definition.

Usages:

```
>>> parse('a:struct[a:struct[a:string]]').simpleString()
'struct<a:struct<a:string>>'
```

Parameters **`schema`** (`str`) – Schema definition as string.

Returns StructType

Raises `UnsupportedDataType` – In case of unsupported data type.

Generic Utils

These are generic utils used across the Sparkly library.

`sparkly.utils.`**`absolute_path`**(*file_path*, **rel_path*)

Returns absolute path to file.

Usage:

```
>>> absolute_path('/my/current/dir/x.txt', '..', 'x.txt')
'/my/current/x.txt'
```

```
>>> absolute_path('/my/current/dir/x.txt', 'relative', 'path')
'/my/current/dir/relative/path'
```

```
>>> import os
>>> absolute_path('x.txt', 'relative/path') == os.getcwd() + '/relative/path'
True
```

Parameters

- **file_path** (*str*) – file
- **rel_path** (*list[str]*) – path parts

Returns str

`sparkly.utils.`**`kafka_get_topics_offsets`**(*host*, *topic*, *port=9092*)

Returns available partitions and their offsets for the given topic.

Parameters

- **host** (*str*) – Kafka host.
- **topic** (*str*) – Kafka topic.
- **port** (*int*) – Kafka port.

Returns **[** – [(partition, start_offset, end_offset)].

Return type int, int, int

CHAPTER 9

Exceptions

exception sparkly.exceptions.**FixtureError**
Happen when testing data setup or teardown fails.

exception sparkly.exceptions.**InvalidArgumentError**
Happen when invalid parameters are passed to a function.

exception sparkly.exceptions.**SparklyException**
Base exception of sparkly lib.

exception sparkly.exceptions.**UnsupportedDataType**
Happen when schema defines unsupported data type.

CHAPTER 10

Integration Testing Base Classes

10.1 Base testing classes

There are two main testing classes in Sparkly:

- **SparklyTest:**
  - Instantiates Sparkly context specified in *context* attribute.
  - The context will be available via *self.hc*.
- **SparklyGlobalContextTest:**
  - Reuses single SparklyContext for all tests for performance boost.

Example:

```
from sparkly import SparklyContext
from sparkly.test import SparklyTest

class MyTestCase(SparklyTest):
    context = SparklyContext
    def test(self):
        df = self.hc.read_ext.by_url(...)
        self.assertDataFrameEqual(
            df, [('test_data', 1)], ['name', 'number']
        )

...

class MyTestWithReusableContext(SparklyGlobalContextTest):
    context = SparklyContext
    def test(self):
        df = self.hc.read_ext.by_url(...)

...
```

10.2 Fixtures

Fixtures is term borrowed from testing in Django framework. A fixture will load data to a database upon text execution.

There are couple of databases supported in Sparkly:

- Mysql (requires: *PyMySql*)
- Elastic
- Cassandra (requires: *cassandra-driver*)

Example:

```
from sparkly.test import MysqlFixture, SparklyTest

class MyTestCase(SparklyTest):
    ...
    fixtures = [
        MysqlFixture('mysql.host',
                     'user',
                     'password',
                     '/path/to/setup_data.sql',
                     '/path/to/remove_data.sql')
    ]
    ...
```

class sparkly.testing.**CassandraFixture**(*host*, *setup_file*, *teardown_file*)
Fixture to load data into cassandra.

Notes

- Depends on cassandra-driver.

Examples

```
>>> class MyTestCase(SparklyTest):
...     fixtures = [
...         CassandraFixture(
...             'cassandra.host',
...             absolute_path(__file__, 'resources', 'setup.cql'),
...             absolute_path(__file__, 'resources', 'teardown.cql'),
...         )
...     ]
...
```

```
>>> class MyTestCase(SparklyTest):
...     data = CassandraFixture(
...         'cassandra.host',
...         absolute_path(__file__, 'resources', 'setup.cql'),
...         absolute_path(__file__, 'resources', 'teardown.cql'),
...     )
...     def setUp(self):
...         data.setup_data()
...     def tearDown(self):
...         data.teardown_data()
...
```

```
>>> def test():
...     fixture = CassandraFixture(...)
...     with fixture:
```

```
...         test_stuff()
...
```

class `sparkly.testing.`**`ElasticFixture`**(*host*, *es_index*, *es_type*, *mapping=None*, *data=None*, *port=None*)

Fixture for elastic integration tests.

Notes

- Data upload uses bulk api.

Examples

```
>>> class MyTestCase(SparklyTest):
...     fixtures = [
...         ElasticFixture(
...             'elastic.host',
...             'es_index',
...             'es_type',
...             '/path/to/mapping.json',
...             '/path/to/data.json',
...         )
...     ]
...
```

class `sparkly.testing.`**`Fixture`**

Base class for fixtures.

Fixture is a term borrowed from Django tests, it's data loaded into database for integration testing.

`setup_data`()

Method called to load data into database.

`teardown_data`()

Method called to remove data from database which was loaded by *setup_data*.

class `sparkly.testing.`**`KafkaFixture`**(*host*, *port=9092*, *topic=None*, *key_serializer=None*, *value_serializer=None*, *data=None*)

Fixture for kafka integration tests.

Notes

- depends on kafka-python lib.
- json file should contain array of dicts: [{'key': ..., 'value': ...}]

Examples

```
>>> class MyTestCase(SparklyContext):
...     fixtures = [
...         KafkaFixture(
...             'kafka.host', 'topic',
...             key_serializer=..., value_serializer=...,
```

```
...             data='/path/to/data.json',
...         )
...     ]
```

class `sparkly.testing.`**`MysqlFixture`**(*host*, *user*, *password=None*, *data=None*, *teardown=None*)
Fixture for mysql integration tests.

Notes

- depends on PyMySql lib.

Examples

```
>>> class MyTestCase(SparklyTest):
...     fixtures = [
...         MysqlFixture('mysql.host', 'user', 'password', '/path/to/data.sql')
...     ]
...     def test(self):
...         pass
...
```

class `sparkly.testing.`**`SparklyGlobalContextTest`**(*methodName='runTest'*)
Base test case that keeps a single instance for the given context class across all tests.

Integration tests are slow, especially when you have to start/stop Spark context for each test case. This class allows you to reuse Spark context across multiple test cases.

class `sparkly.testing.`**`SparklyTest`**(*methodName='runTest'*)
Base test for spark scrip tests.

Initializes and shuts down Context specified in *context* param.

Example

```
>>> class MyTestCase(SparklyTest):
...     def test(self):
...         self.assertDataFrameEqual(
...             self.hc.sql('SELECT 1 as one').collect(),
...             [{'one': 1}],
...         )
```

`assertDataFrameEqual`(*actual_df*, *expected_data*, *fields=None*, *ordered=False*)
Ensure that DataFrame has the right data inside.

Parameters

- **actual_df** (*pyspark.sql.DataFrame|list[pyspark.sql.Row]*) – Dataframe to test data in.
- **expected_data** (*list[dict]*) – Expected dataframe rows defined as dicts.
- **fields** (*list[str]*) – Compare only certain fields.
- **ordered** (*bool*) – Does order of rows matter?

context

alias of SparklyContext

License

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
```

```
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
```

```
        that You distribute, all copyright, patent, trademark, and
        attribution notices from the Source form of the Work,
        excluding those notices that do not pertain to any part of
        the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part of its
        distribution, then any Derivative Works that You distribute must
        include a readable copy of the attribution notices contained
        within such NOTICE file, excluding those notices that do not
        pertain to any part of the Derivative Works, in at least one
        of the following places: within a NOTICE text file distributed
        as part of the Derivative Works; within the Source form or
        documentation, if provided along with the Derivative Works; or,
        within a display generated by the Derivative Works, if and
        wherever such third-party notices normally appear. The contents
        of the NOTICE file are for informational purposes only and
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

 5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

 6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

 7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

 8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
```

```
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't include
      the brackets!)  The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright 2017 Tubular Labs, Inc.

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.


======================================================================
Sparkly Subcomponents:

The Sparkly project contains subcomponents with separate copyright
notices and license terms. Your use of the source code for the these
subcomponents is subject to the terms and conditions of the following
licenses.

=======================================================================
Apache licenses
=======================================================================

The following dependencies are provided under a Apache license. See project link for
details.
```

```
(Apache License 2.0) Spark (https://github.com/apache/spark)
(Apache License 2.0) cassandra-driver (https://github.com/datastax/python-driver)

========================================================================
BSD-style licenses
========================================================================

The following dependencies are provided under a BSD-style license. See project link␣
↪for details.

(BSD License) mock (https://github.com/testing-cabal/mock)
(PSF License) Sphinx (https://github.com/sphinx-doc/sphinx)

========================================================================
MIT licenses
========================================================================

The following dependencies are provided under the MIT License. See project link for␣
↪details.

(MIT License) sphinx_rtd_theme (https://github.com/snide/sphinx_rtd_theme)
(MIT License) pytest (https://github.com/pytest-dev/pytest)
(MIT License) pytest-cov (https://github.com/pytest-dev/pytest-cov)
(MIT License) PyMySQL (https://github.com/PyMySQL/PyMySQL)
```

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

S

T

U