intake_spark Documentation

Release 0.1.2+1.g360d91c.dirty

Joseph Crail

Jul 02, 2021

CONTENTS:

1 Quickstart **3**
- 1.1 Installation . . . 3
- 1.2 Usage . . . 3

2 API Reference **5**

3 Indices and tables **11**

Index **13**

This package enables the Intake data access and catalog system to read from Apache Spark.

This package provides Spark RDD and DataFrame drivers, and passes chained set of arguments on to pyspark, and there is also a driver to view the Spark catalog (often with tables provided by Hive) as Intake entries.

Be sure to read the quickstart on how to provide the context/parameters required by Spark and how to phrase a Spark loading invocation for Intake.

This package is required by Intake for use of any `.to_spark()` methods.

CHAPTER

ONE

QUICKSTART

`intake-spark` provides quick and easy access to data via Apache Spark

1.1 Installation

To use this plugin for intake, install with the following command:

```
conda install -c conda-forge intake-spark
```

1.2 Usage

1.2.1 Establishing a Context

Operations on a Spark cluster are achieved via a "context", which is a python-side client to the remote system. All use of this package require a valid context in order to work. There are two ways to establish the context:

- use the function `intake_spark.base.SparkHolder.set_class_session()`, passing for `context=` and `session=` the respective existing objects that have been created previously using conventional means. In this case, Spark connection-specific parameters are not encoded in the catalog. Since intake-spark uses `getOrCreate()`, the existence of the objects should be enough for them to be picked up at data access time, but we still recommend calling the set function explicitly. Note that if only using RDDs and no SQl functionality, the `session` need not be created or provided.
- the same function can also take a set of parameters, in which case intake-spark will attempt to create the context and session for you, passing the parameters (master, app_name, executor_env, spark_home and config parameters) on to Spark. In this case, the parameters can be stored in a catalog (the `context_kwargs` parameter, a dictionary). If providing an empty set of parameters, Spark will create the default local context, which is useful only for testing.

```
# in-code example
from intake_spark.base import SparkHolder
import intake
SparkHolder.set_class_session(master='spark://myhost:7077', app_name='intake', hive=True)
```

1.2.2 Encoding Spark calls

Spark calls are often expressed as a chain of attribute look-ups with some being called as methods with arguments. To encode these for Intake, we make each stage of the chain an element in a list, starting from the Context or Session for the RDD and DataFrame versions of the driver. For example, the following encodes `sc.textfile('s3://bucket/files*.txt')`, i.e., a single element of attribute lookup

```
source = intake.open_spark_rdd([
    ['textFile', ['s3://bucket/files*.txt',]],
    ['map', [len, ]]
    ])
rdd = source.to_spark()
```

Here `rdd` will be a pySpark RDD instance.

A more complicated example, for encoding `spark.read.option("mergeSchema", "true").parquet("data/test_table")` for use with Intake

```
source = intake.open_spark_dataframe([
    ['read', ],
    ['option', ["mergeSchema", "true"]],
    ['parquet', ["data/test_table",]]
    ])
df = source.to_spark()
```

Note that you *can* pass functions using this formalism, but only encode python built-ins into a YAML file, e.g., `len => !!python/name:builtins.len ''`.

1.2.3 Using in a catalog

The above example could be expressed in YAML syntax as follows

```
sources:
  spark_dataframe:
    args:
      args:
      - - read
      - - option
        - - mergeSchema
          - 'true'
      - - parquet
        - - data/test_table
      context_kwargs:
        master: "spark://myhost:7077"
        app_name: intake
    description: ''
    driver: spark_dataframe
    metadata: {}
```

Note the complex nesting pattern, and that in this case we are including arguments for creating an appropriate SparkContext and Session on the fly, if one has not already been made.

CHAPTER

TWO

API REFERENCE

intake_spark.spark_sources.SparkRDD(*args, ...)	A reference to an RDD definition in Spark
intake_spark.spark_sources.SparkDataFrame(...)	A reference to a DataFrame definition in Spark
intake_spark.spark_cat.SparkTablesCatalog(...)	Intake automatically-generate catalog for tables stored in Spark

class intake_spark.spark_sources.**SparkRDD**(**args*, ***kwargs*)

A reference to an RDD definition in Spark

RDDs are list-of-things objects, evaluated lazily in Spark.

Examples

```
>>> args = [('textFile', ('text.*.files', )),
...         ('map', (len,))]
>>> context = {'master': 'spark://master.node:7077'}
>>> source = SparkRDD(args, context)
```

The output of *source.to_spark()* is an RDD object holding the lengths of the lines of the input files.

Attributes

cache

cache_dirs

cat

classname

description

dtype

entry

gui Source GUI, with parameter selection and plotting

has_been_persisted

hvplot Returns a hvPlot object to provide a high-level plotting API.

is_persisted

plot Returns a hvPlot object to provide a high-level plotting API.

plots List custom associated quick-plots

shape

Methods

`__call__`(**kwargs)	Create a new instance of this source with altered arguments
`close`()	Close open resources corresponding to this data source.
`configure_new`(**kwargs)	Create a new instance of this source with altered arguments
`describe`()	Description from the entry spec
`discover`()	Open resource and populate the source attributes.
`export`(path, **kwargs)	Save this data for sharing with other people
`get`(**kwargs)	Create a new instance of this source with altered arguments
`persist`([ttl])	Save data from this source to local persistent storage
`read`()	Materialise the whole RDD into a list of objects
`read_chunked`()	Return iterator over container fragments of data source
`read_partition`(i)	Returns one of the partitions of the RDD as a list of objects
`to_dask`()	Return a dask container for this data source
`to_spark`()	Return the spark object for this data, an RDD
`yaml`()	Return YAML representation of this data-source

get_persisted	
set_cache_dir	

read()
Materialise the whole RDD into a list of objects

read_partition(*i*)
Returns one of the partitions of the RDD as a list of objects

to_spark()
Return the spark object for this data, an RDD

class intake_spark.spark_sources.**SparkDataFrame**(**args*, ***kwargs*)
A reference to a DataFrame definition in Spark

DataFrames are tabular spark objects containing a heterogeneous set of columns and potentially a large number of rows. They are similar in concept to Pandas or Dask data-frames. The Spark variety produced by this driver will be a handle to a lazy object, where computation will be managed by Spark.

Examples

```
>>> args = [
...    ['read', ],
...    ['format', ['csv', ]],
...    ['option', ['header', 'true']],
...    ['load', ['data.*.csv', ]]
...    ]
>>> context = {'master': 'spark://master.node:7077'}
>>> source = SparkDataFrame(args, context)
```

The output of *source.to_spark()* contains a spark object pointing to the parsed contents of the indicated CSV files

Attributes

cache

cache_dirs

cat

classname

description

dtype

entry

gui Source GUI, with parameter selection and plotting

has_been_persisted

hvplot Returns a hvPlot object to provide a high-level plotting API.

is_persisted

plot Returns a hvPlot object to provide a high-level plotting API.

plots List custom associated quick-plots

shape

Methods

`__call__`(**kwargs)	Create a new instance of this source with altered arguments
`close`()	Close open resources corresponding to this data source.
`configure_new`(**kwargs)	Create a new instance of this source with altered arguments
`describe`()	Description from the entry spec
`discover`()	Open resource and populate the source attributes.
`export`(path, **kwargs)	Save this data for sharing with other people
`get`(**kwargs)	Create a new instance of this source with altered arguments
`persist`([ttl])	Save data from this source to local persistent storage

continues on next page

Table 3 – continued from previous page

read()	Read all of the data into an in-memory Pandas data-frame
read_chunked()	Return iterator over container fragments of data source
read_partition(i)	Returns one partition of the data as a pandas data-frame
to_dask()	Return a dask container for this data source
to_spark()	Return the Spark object for this data, a DataFrame
yaml()	Return YAML representation of this data-source

get_persisted	
set_cache_dir	

read()
Read all of the data into an in-memory Pandas data-frame

read_partition(*i*)
Returns one partition of the data as a pandas data-frame

to_spark()
Return the Spark object for this data, a DataFrame

class intake_spark.spark_cat.**SparkTablesCatalog**(**args*, ***kwargs*)
Intake automatically-generate catalog for tables stored in Spark

This driver will query Spark's Catalog object for any tables, and create an entry for each which, when accessed, will instantiate SparkDataFrame sources. Commonly, these table definitions will come from Hive.

Attributes

auth

cache

cache_dirs

cat

classname

description

dtype

entry

gui Source GUI, with parameter selection and plotting

has_been_persisted

hvplot Returns a hvPlot object to provide a high-level plotting API.

is_persisted

kwargs

plot Returns a hvPlot object to provide a high-level plotting API.

plots List custom associated quick-plots

shape

Methods

`__call__`(**kwargs)	Create a new instance of this source with altered arguments
`close`()	Close open resources corresponding to this data source.
`configure_new`(**kwargs)	Create a new instance of this source with altered arguments
`describe`()	Description from the entry spec
`discover`()	Open resource and populate the source attributes.
`export`(path, **kwargs)	Save this data for sharing with other people
`filter`(func)	Create a Catalog of a subset of entries based on a condition
`force_reload`()	Imperative reload data now
`from_dict`(entries, **kwargs)	Create Catalog from the given set of entries
`get`(**kwargs)	Create a new instance of this source with altered arguments
`items`()	Get an iterator over (key, source) tuples for the catalog entries.
`keys`()	Entry names in this catalog as an iterator (alias for __iter__)
`persist`([ttl])	Save data from this source to local persistent storage
`pop`(key)	Remove entry from catalog and return it
`read`()	Load entire dataset into a container and return it
`read_chunked`()	Return iterator over container fragments of data source
`read_partition`(i)	Return a part of the data corresponding to i-th partition.
`reload`()	Reload catalog if sufficient time has passed
`save`(url[, storage_options])	Output this catalog to a file as YAML
`serialize`()	Produce YAML version of this catalog.
`to_dask`()	Return a dask container for this data source
`to_spark`()	Provide an equivalent data object in Apache Spark
`values`()	Get an iterator over the sources for catalog entries.
`walk`([sofar, prefix, depth])	Get all entries in this catalog and sub-catalogs
`yaml`()	Return YAML representation of this data-source

get_persisted	
search	
set_cache_dir	

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

R

read() (*intake_spark.spark_sources.SparkDataFrame method*), 8
read() (*intake_spark.spark_sources.SparkRDD method*), 6
read_partition() (*intake_spark.spark_sources.SparkDataFrame method*), 8
read_partition() (*intake_spark.spark_sources.SparkRDD method*), 6

S

SparkDataFrame (*class in intake_spark.spark_sources*), 6
SparkRDD (*class in intake_spark.spark_sources*), 5
SparkTablesCatalog (*class in intake_spark.spark_cat*), 8

T

to_spark() (*intake_spark.spark_sources.SparkDataFrame method*), 8
to_spark() (*intake_spark.spark_sources.SparkRDD method*), 6