superpyrate Documentation

Release 0.2.0.post0.dev12+ng0a39794

Will Usher

Aug 22, 2016

Contents

This is the documentation of **superpyrate**.

CHAPTER 1

Contents

1.1 Setup

1.1.1 Configuration

After *installation*, it is necessary to conduct three separate configuration tasks for:

- logging
- luigi
- postgres

Setup and initialise a remote luigi server to host the scheduler. In the UCL shipping group, this is currently hosted on the 'linux box'

Logging

The log output of the superpyrate pipeline is extremely verbose when set to DEBUG level. For production, it is strongly recommended that the level is set to INFO only to avoid the generation of GB-scale log files.

This is easily done using a python logging configuration file, and referencing the path to the logging config in the luigi configuration file.

The contents of the python logging file should look something like this:

```
[loggers]
keys=root,luigi

[handlers]
keys=consoleHandler,fileHandler

[formatters]
keys=fileFormatter,consoleFormatter

[logger_root]
level=INFO
handlers=consoleHandler

[logger_luigi]
level=INFO
handlers=consoleHandler,fileHandler
qualname=luigi.interface
```

```
propagate=0

[handler_consoleHandler]
class=StreamHandler
level=WARNING
formatter=consoleFormatter
args=(sys.stdout,)

[handler_fileHandler]
class=FileHandler
level=INFO
formatter=fileFormatter
args=('logfile.log',)

[formatter_fileFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
datefmt=

[formatter_consoleFormatter]
format=%(levelname)s - %(message)s
datefmt=
```

Luigi

The luigi configuration file can be placed anywhere and is specified through the environment variable `LUIGI_CONFIG_PATH`.

More information about the configuration file can be found in the luigi documentation.

Here's an example configuration file:

```
[resources]
postgres=12
matlab=120
postgres_linuxbox=100

[core]
email-type=html
error-email=user@ucl.ac.uk
default-scheduler-host=123.45.678.910
default-scheduler-port=8028
logging_conf_file=/home/username/logging.conf

[worker]
keep_alive=False
ping_interval=30
```

Postgres

The postgres configuration file `postgresql.conf` resides in the data folder specified by the environment variable `PGDATA`.

This file contains a number of server specific settings which dramatically affect the performance of the database, particular for memory and processor intensive operations such as index generation, clustering and bulk copying of data.

Various postgres configuration files are stored in a github repository.

1.1.2 Setup for ingest

Setting up the database:

```
# Initialise the database server using Scratch for the data
initdb -D $HOME/Scratch/data
# Spin up the database server
pg_ctl -D $HOME/Scratch/data -l logfile start

# Create the test database
createdb test_aisdb
# Use the following command to access the database schema and tables
#psql --host=localhost --port=5432 --username=test_ais --dbname=test_aisdb

psql -U postgres -c "create extension postgis"
psql -c "create database test_aisdb;" -U postgres
psql -U postgres -c "CREATE USER test_ais WITH PASSWORD 'test_ais' SUPERUSER;"
psql -U postgres -c "GRANT ALL PRIVILEGES ON DATABASE test_aisdb to test_ais;"
```

Setup virtual python environment using conda:

```
wget http://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh \
-O miniconda.sh
chmod +x miniconda.sh && ./miniconda.sh -b -p $HOME/miniconda
export PATH=$HOME/miniconda/bin:$PATH
conda update --yes conda

# Configure the conda environment and put it in the path using the
# provided versions
conda create -n testenv --yes python=$PYTHON_VERSION pip scipy pandas numpy psycopg2␣
↪sphinx pylint
source activate testenv
```

Before installing superpyrate, you'll need to setup your git account. Enter the following commands:

```
git config --global user.name "YOUR NAME"
git config --global user.email "YOUR EMAIL ADDRESS"
```

To access git from Legion, you'll need to setup a certificate and ssh access to git. You can follow the instructions here:

```
cd $HOME
git clone https://github.com/UCL-ShippingGroup/superpyrate.git
cd superpyrate
pip install -r requirements.txt
python setup.py develop
```

It is also a good idea to get the legion-scripts which help with running superpyrate:

```
cd $HOME
git clone https://github.com/UCL-ShippingGroup/legion-scripts.git
```

Add a configuration file for py:mod:*luigi*:

```
[core]
default-scheduler-host=128.40.181.109
```

```
default-scheduler-port=8028
logging_conf_file=/home/ucftxyz/logging.conf

[worker]
keep_alive=False
ping_interval=30
```

1.1.3 Installation

Using Legion, loading postgres is as simple as loading the required modules:

If you have legion-scripts installed, then you can just run the `load_postgres.sh` script.

Otherwise, we assume install on linux. For MacOSx, Windows and other architectures, refer to the packages and documentation for postgres available on the website.

1.1.4 Compiling from source

First, install postgres:

```
cd $HOME
wget https://ftp.postgresql.org/pub/source/v9.5.2/postgresql-9.5.2.tar.gz
gunzip postgresql-9.5.2.tar.gz
tar xf postgresql-9.5.2.tar
cd postgresql-9.5.2
# Install a local version of postgres in the user directory on the login node
./configure --prefix=$HOME/pgsql
./make -s
./make install
./make clean
# Setup path
echo 'PATH=$HOME/pgsql/bin:$PATH' >> ~/.bash_profile
```

Installing postgis, is painful:

```
# Obtain, compile and install postgis and its requirements (GEOS, PROJ4, GDAL)
cd $HOME
svn checkout http://svn.osgeo.org/geos/trunk geos-svn
cd geos-svn
./autogen.sh
./configure --prefix=$HOME/geos
./make -s
./make install
echo 'PATH=$HOME/geos/bin:$PATH' >> ~/.bash_profile
echo 'export LD_LIBRARY_PATH=$HOME/geos/lib:$LD_LIBRARY_PATH' >> ~/.bash_profile


cd $HOME
wget http://download.osgeo.org/proj/proj-4.9.1.tar.gz
tar xf proj-4.9.1.tar.gz
cd proj-4.9.1
./configure --prefix=$HOME/proj4
./make
./make install
echo 'PATH=$HOME/proj4/bin:$PATH' >> ~/.bash_profile
echo 'export LD_LIBRARY_PATH=$HOME/proj4/lib:$LD_LIBRARY_PATH' >> ~/.bash_profile
```

```
cd $HOME
wget http://download.osgeo.org/gdal/2.1.0/gdal-2.1.0.tar.gz
tar xf gdal-2.1.0.tar.gz
cd gdal-2.1.0
./configure --prefix=$HOME/gdal
./make
./make install
echo 'export PATH=/$HOME/gdal/bin:$PATH' >> ~/.bash_profile
echo 'export LD_LIBRARY_PATH=$HOME/gdal/lib:$LD_LIBRARY_PATH' >> ~/.bash_profile
echo 'export GDAL_DATA=$HOME/gdal/share/gdal' >> ~/.bash_profile
echo 'export PATH' >> ~/.bash_profile
# Test
#% gdalinfo --version
# See below for installation of Python bindings


wget http://download.osgeo.org/postgis/source/postgis-2.2.2.tar.gz
tar xf postgis-2.2.2.tar.gz
cd postgis-2.2.2
./configure --prefix=$HOME/postgis
./make
./make install
```

1.2 License

```
The MIT License (MIT)

Copyright (c) 2016 Will Usher

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

1.3 Developers

- Will Usher <w.usher@ucl.ac.uk>

1.4 Changelog

1.4.1 Version 0.1

- Feature A added
- FIX: nasty bug #1729 fixed
- add your changes here!

1.5 superpyrate

1.5.1 superpyrate package

Submodules

superpyrate.db_setup module

Sets up the tables in a newly created database, ready for data ingest

superpyrate.db_setup.**main**()

superpyrate.db_setup.**make_options**()

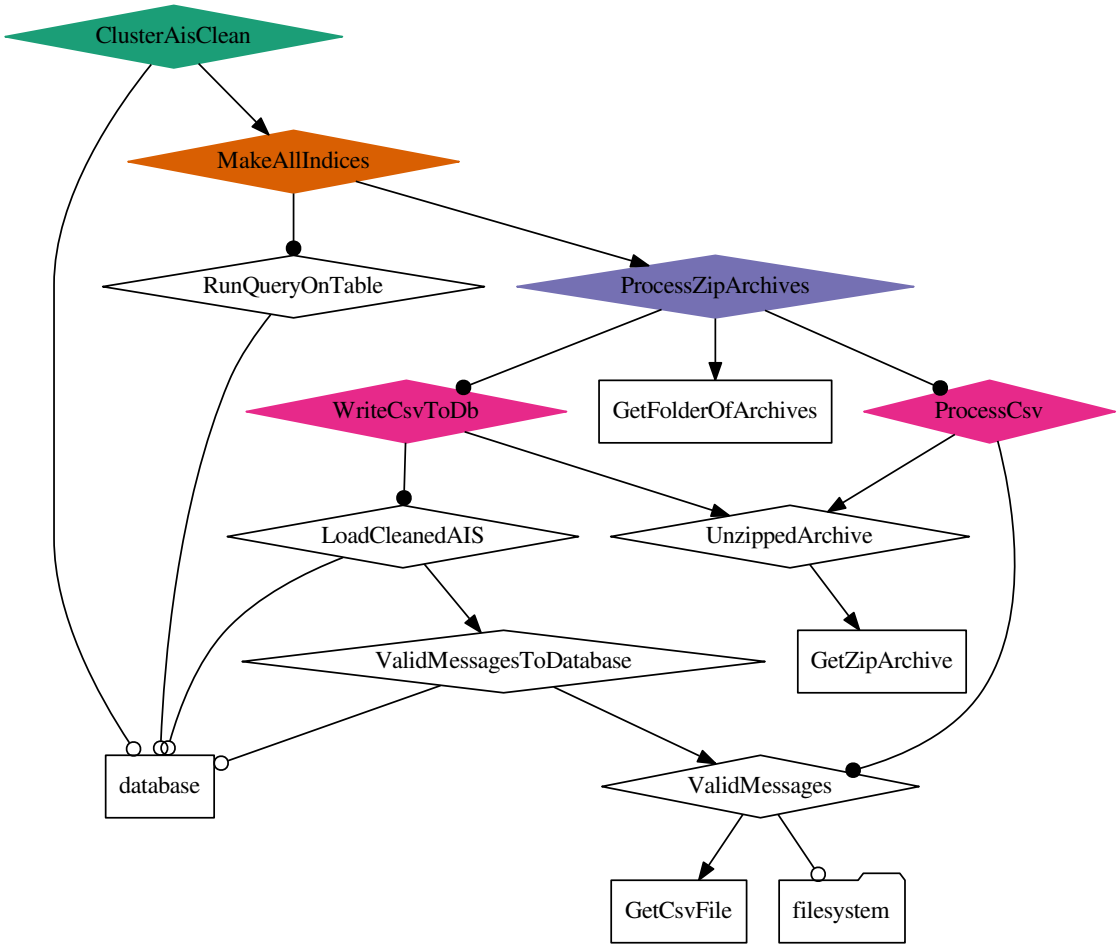
superpyrate.pipeline module

Runs an integrated pipeline from raw zip file to database tables.

This mega-pipeline is constructed out of three smaller pipelines and now brings together tasks which:

1. Unzip individual AIS archives and output the csv files
2. Validate each of the csv files, processing using a derived version of the pyrate code, outputting vaidated csv files
3. Using the postgres *copy* command, ingest the validated data directly into the database

Because the exact contents of the zipped archives are unknown until they are unzipped, tasks are spawned dynamically.



Entry points

It is not necessary to run the entire pipeline, although there is little harm in doing so, as luigi manages the process so that individual tasks are idempotent. This means that a task only runs if required. Luigi only runs the tasks necessary to produce the files which are required for the specified entry point.

For example, to run the entire pipeline, producing a full ingested and clustered database, run:

```
luigi --module superpyrate.pipeline ClusterAisClean
      --workers 12
      --folder-of-zips /folder/of/zips/
      --with_db
```

If only the validated csv files are required, run:

```
luigi --module superpyrate.pipeline ProcessZipArchives
      --workers 12
      --folder-of-zips /folder/of/zips/
      --with_db
```

Working folder

The working folder `LUIGIWORK` must contain two subfolders - files and tmp. The `files` subfolder contains the `unzipped` and `cleancsv` folders, with all of the respective temporary files stored within. The `tmp` subfolder contains `processcsv`, `writecsv`, `archives` and `database` folders and contains files which are generated by the tasks which do not produce an actual file as output, rather spawn child-tasks.

Environment Variables

LUIGIWORK the working folder for the files

DBHOSTNAME hostname for the database e.g. localhost

DBNAME the name of the database

DBUSER the name of the user with access to the database

DBUSERPASS the password of the database user

class superpyrate.pipeline.**ClusterAisClean**(**args*, ***kwargs*)
Bases: *superpyrate.pipeline.ClusterAisClean*

Clusters the ais_clean table over the disk on the mmsi index

requires(*self*)

task_namespace = None

class superpyrate.pipeline.**GetCsvFile**(**args*, ***kwargs*)
Bases: luigi.task.ExternalTask

csvfile = <luigi.parameter.Parameter object>

output()

task_namespace = None

class superpyrate.pipeline.**GetFolderOfArchives**(**args*, ***kwargs*)
Bases: luigi.task.ExternalTask

Returns the folder of zipped archives as a luigi.file.LocalTarget

folder_of_zips = <luigi.parameter.Parameter object>

output()

run()

task_namespace = None

class superpyrate.pipeline.**GetZipArchive**(**args*, ***kwargs*)
Bases: luigi.task.ExternalTask

Returns a zipped archive as a luigi.file.LocalTarget

output()

task_namespace = None

zip_file = <luigi.parameter.Parameter object>

class superpyrate.pipeline.**LoadCleanedAIS**(**args*, ***kwargs*)
Bases: luigi.postgres.CopyToTable

Update `ais_sources` table with name of CSV file processed

After the valid csv files are successfully written to the database, this function updates the `sources` table with the name of the file which has been written

column_separator = ‘,’

csvfile = <luigi.parameter.Parameter object>

database = ‘’

host = ‘’

null_values = (None, ‘’)

password = ‘’

requires()

run()

table = ‘ais_sources’

task_namespace = None

user = ‘’

class superpyrate.pipeline.**MakeAllIndices**(**args*, ***kwargs*)
Bases: *superpyrate.pipeline.MakeAllIndices*

Creates the indices required for a specified table

The list of indices are derived from the table specification in `pyrate`

Parameters **table** (*str, default='ais_clean'*) –

requires(*self*)

task_namespace = None

class superpyrate.pipeline.**ProcessCsv**(**args*, ***kwargs*)
Bases: luigi.task.Task

Yields *ValidMessages*

output()
Dummy files are placed in a folder of the same name as the zip file

The files are placed in a subdirectory of `LUIGIWORK` called `tmp/processcsv`

requires()

run()

task_namespace = None

zip_file = <luigi.parameter.Parameter object>

class superpyrate.pipeline.**ProcessZipArchives**(**args*, ***kwargs*)
Bases: luigi.task.Task

Dynamically spawns *WriteCsvToDb* or *ProcessCsv* depending on database

Parameters **with_db** (*bool*) – Indicate whether a database is available for writing csv files

Yields

- *WriteCsvToDb*
- *ProcessCsv*

folder_of_zips = <luigi.parameter.Parameter object>

output()

requires()

run()

task_namespace = None

with_db = <luigi.parameter.BoolParameter object>

class superpyrate.pipeline.**RunQueryOnTable**(**args*, ***kwargs*)
Bases: luigi.postgres.PostgresQuery

Runs a query on a table in the database

Used for passing in utility type queries to the database such as creation of indices etc.

Parameters

- **query** (*str*) – A legal sql query
- **table** (*str, default='ais_clean'*) – A table on which to run the query

database = ''

host = ''

password = ''

query = <luigi.parameter.Parameter object>

table = <luigi.parameter.Parameter object>

task_namespace = None

update_id = <luigi.parameter.Parameter object>

user = ''

class superpyrate.pipeline.**UnzippedArchive**(**args*, ***kwargs*)
Bases: luigi.contrib.external_program.ExternalProgramTask

Unzips the zipped archive into a folder of AIS csv format files the same name as the original file

Parameters **zip_file** (*str*) – The absolute path of the zipped archives

output()
Outputs the files into a folder of the same name as the zip file

The files are placed in a subdirectory of LUIGIWORK called files/unzipped

program_args()
Runs 7zip to extract the archives of AIS files

Notes

e Unzip all ignoring folder structure (i.e. to highest level)

-o Output folder

-y Answer yes to everything

requires()

task_namespace = None

zip_file = <luigi.parameter.Parameter object>

class superpyrate.pipeline.**ValidMessages**(**args*, ***kwargs*)
Bases: luigi.task.Task

Takes AIS messages and runs validation functions, generating valid csv files in folder called 'cleancsv' at the same level as unzipped_ais_path

csvfile = <luigi.parameter.Parameter object>

output()
Validated files are named as the original csv file

The files are placed in a subdirectory of LUIGIWORK called files/cleancsv

requires()

run()

task_namespace = None

class superpyrate.pipeline.**ValidMessagesToDatabase**(**args*, ***kwargs*)
Bases: luigi.postgres.CopyToTable

Writes the valid csv files to the postgres database

Parameters **original_csvfile** (*luigi.Parameter*) – The raw csvfile containing AIS data

cols = ['MMSI', 'Time', 'Message_ID', 'Navigational_status', 'SOG', 'Longitude', 'Latitude', 'COG', 'Heading', 'IMO'

column_separator = ','

columns = ['mmsi', 'time', 'message_id', 'navigational_status', 'sog', 'longitude', 'latitude', 'cog', 'heading', 'imo', 'drau

copy(*cursor*, *clean_file*)

database = ''

host = ''

null_values = (None, '')

original_csvfile = <luigi.parameter.Parameter object>

password = ''

requires()

rows()
Return/yield tuples or lists corresponding to each row to be inserted.

Yields **row** (*iterable*)

run()
Inserts data generated by rows() into target table.

If the target table doesn't exist, self.create_table will be called to attempt to create the table.

table = 'ais_clean'

task_namespace = None

user = ''

class superpyrate.pipeline.**WriteCsvToDb**(**args*, ***kwargs*)
Bases: *superpyrate.pipeline.WriteCsvToDb*

Dynamically spawns *LoadCleanedAIS* to load valid csvs into the database

requires(*_self*)

task_namespace = None

superpyrate.pipeline.**get_environment_variable**(*name*)
Tries to access an environment variable, and handles the error by replacing the value with a dummy value (an empty string)

superpyrate.pipeline.**get_working_folder**(*folder_of_zips=None*)

Parameters **folder_of_zips** (*str*) – The absolute path of the folder of zips e.g. /tests/fixtures/testais/

Returns **working_folder** – The path of the working folder. This is either set by the environment variable LUIGIWORK, or if empty is computed from the arguments

Return type str

superpyrate.pipeline.**setup_working_folder**()
Setup the working folder structure for the entire luigi pipeline

The working folder ('LUIGIWORK') must contain two subfolders - files and tmp. The *files* subfolder contains the *unzipped* and *cleancsv* folders, with all of the respective temporary files stored within. The *tmp* subfolder contains *processcsv*, *writecsv*, *archives* and *database* folders and contains files which are generated by the tasks which do not produce an actual file as output, rather spawn child-tasks.

superpyrate.task_countfiles module

Holds the luigi tasks which count the number of rows in the files

Records the number of clean and dirty rows in the AIS data, writing these stats to the database and finally producing a report of the statistics

1. Count the number of rows in the raw csv files (in files/unzipped/<archive>)
2. Count the number of rows int the clean csv files (in files/cleancsv/)
3. Write the clean rows in the clean column of ais_sources
4. Write the dirty (raw - clean) rows into the dirty column of ais_sources

class superpyrate.task_countfiles.**CountLines**(**args*, ***kwargs*)
Bases: *superpyrate.task_countfiles.CountLines*

Counts the number of lines for all the csvfiles in a folder

Writes all the counts and filenames to a delimited file with the name of the folder

Parameters **folder_name** (*str*) – The absolute path of the csv file

requires(*_self*)

task_namespace = None

class superpyrate.task_countfiles.**DoIt**(**args*, ***kwargs*)
Bases: luigi.task.Task

folder_of_zips = <luigi.parameter.Parameter object>

output()

requires()

run()

task_namespace = None

with_db = <luigi.parameter.BoolParameter object>

class superpyrate.task_countfiles.**GetCountsForAllFiles**(**args*, ***kwargs*)
Bases: *superpyrate.task_countfiles.GetCountsForAllFiles*

Counts the rows in all clean (validated) and raw files

requires(*_self*)

task_namespace = None

class superpyrate.task_countfiles.**ProduceStatisticsReport**(**args*, ***kwargs*)
Bases: *superpyrate.task_countfiles.ProduceStatisticsReport*

Produces a report of the data statistics

requires(*_self*)

task_namespace = None

superpyrate.tasks module

Contains the code for validating AIS messages

superpyrate.tasks.**learn_columns**(*read_cols*, *required_cols*, *csv_or_xml='csv'*)
Tries to match the read columns with the list given

Parameters

- **read_cols** (*dict*) – The columns read from the csv file
- **required_cols** (*list*) – A list of the columns required from the csv file
- **csv_or_xml** (*str, default='csv'*) –

superpyrate.tasks.**produce_valid_csv_file**(*inputf*, *outputf*)

Parameters

- **input_file** – File path to a large CSV file of AIS data
- **output_file** – File path for a CSV file containing validated and cleaned data

superpyrate.tasks.**readcsv**(*fp*, *forced_col_map=None*, *columns=None*)
Yields a dctionary of the subset of columns required

Reads each line in CSV file, checks if all columns are available, and returns a dictionary of the subset of columns required (as per AIS_CSV_COLUMNS).

If row is invalid (too few columns), returns an empty dictionary.

Parameters

- **fp** (*TextIOWrapper*) – An open TextIOWrapper (returned by *open()*)
- **forced_col_map** (*dict*) – A dictionary mapping the keys defined in columns to columns with different names
- **columns** (*dict, default=AIS_CSV_COLUMNS*) – A dictionary of columns

Yields **rowsubset** (*dict*) – A dictionary of the subset of columns as per *columns*

superpyrate.tasks.**unfussy_reader**(*csv_reader*)

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

S

C

D

F

G

H

L

M

N

O

P

Q

R

S

T

U

V

W

Z