PBS Gestor documentation

version

Altair

August 21, 2019

Contents

Welcome to PBS Gestor's documentation!

gestor

gestor module

PBS Gestor: Convert the PBS accounting logs to PostgreSQL database.

This Python script is the PBS Gestor daemon. It reads PBS accounting log messages, parses them to get the job attributes and inserts them into the Reporting database (PostgreSQL).

Pre-requisites for running PBS Gestor:

- Python, version 2.7 or 3.6: https://www.python.org/
- PBS server accounting logs access
- PostgreSQL, version 9, such as 9.2 or 9.6: https://www.postgresql.org/

Pre-set-up before running PBS Gestor:

- PBS is running
- PostgreSQL has host-based authentication (HBA) configuration usually in /var/lib/pgsql/data/pg_hba.conf
- PostgreSQL server daemon is running
- Configuration file ~/.config/pbs_gestor/pbs_gestor_config.json contains PostgreSQL authentication data, such as: hostname, port, username, password, database name

Functions:

- set_logger: sets the logger for the application.
- parser: transforms PBS logs into format for PostgreSQL database.
- event_type: determines type of event in log message
- date_range: utility function providing range of dates between start date and end date, including start date and end date
- main: primary function which run first and calls the other functions as needed.

class `gestor.`**`StrToDate`**(`option_strings`, `database_handler`, `dest=None`, `nargs=None`, `**kwargs`)

Bases: **`argparse.Action`**

Parse day given by user and set it into namespace as date.

`gestor.`**`date_range`**(`start_date`, `end_date`)

Create range of dates between start date and end date, inclusively.

Parameters:
- **start_date** – the first date in date range
- **end_date** – the last date in date range

Returns: list of dates from start date to end date

Return type: date range

`gestor.`**`detect_log_switch`**(`pbs_log_handler`, `database_handler`, `logarguments`)

Detect switch of log handler between logs and record it.

Parameters:
- **pbs_log_handler** – Log handler
- **logdate** – log which is supposed to be in processing
- **start** – time when processing of log started

Returns: log which is currently in processing start: time now when it started to be processed count: lines processed in log file

Return type: logdate

gestor.**event_type**(parsed_event)
Determine type of event in log message.

Parameters: **parsed_event** (*dict*) – job attributes
Returns: type of event
Return type: _event_type(text)

gestor.**log_line_parser**(job)
Parse most of the contents of log line and get the job attributes.

Parameters: **job** (*tuple*) – A tuple of three dictionaries formed after the processing of the log message
Returns: job attributes required in the reporting database.
Return type: attrs(dict)

gestor.**main**(system_config)
Read the PBS accounting logs, process and record to SQL database.
Create PBS Log handler instances to read the PBS accounting logs. Each log line is processed to create a tuple of three dictionaries. The tuple is sent to log_line_parser() to create dictionary of job attributes. The dictionary is written to SQL database.

Parameters: **system_config** – configuration object.
Returns: None

gestor.**none_to_today**(checkdate)
Check whether date is existing, if not, returns today instead.

Parameters: **checkdate** – date to be checked
Returns: either given date or current date
Return type: checkdate

gestor.**parse_input**(database_handler)
Parse arguments provided and return a list of log handlers.

Parameters:
- **args** – list of arguments provided by user to application
- **database_handler** – connection to database, in order to be able to look up the last log file which was ever scanned

Returns: list of log handlers
Return type: pbs_log_handlers

gestor.**set_logger**()
Open and load the logger configuration file, validate and set the logging config.

Parameters: **None** –
Returns: None
Raises: **ConfigurationError**

pbs_gestor package

Subpackages

pbs_gestor.model package

Submodules

pbs_gestor.model.exceptions module

Provide custom exceptions for the PBS Gestor modules.

Classes:

- BaseError: Base class for Exceptions.
- ConfigurationError: Exception class for Config Errors.
- PBSConfigNotFoundError: Exception to be raised when PBS's config file is not found.

exception pbs_gestor.model.exceptions.**BaseError**(message)
Bases: **exceptions.Exception**
Base exception class to be inherited by other exceptions.

exception pbs_gestor.model.exceptions.**ConfigurationError**(message)
Bases: **pbs_gestor.model.exceptions.BaseError**
Exception class for Configuration related issues.

exception pbs_gestor.model.exceptions.**DatabaseError**(message)
Bases: **pbs_gestor.model.exceptions.BaseError**
Exception class for database related operations.

exception **ConnectionError**(message)
Bases: **pbs_gestor.model.exceptions.BaseError**
Exception class for database connection failure.

exception **TableCreationError**(message)
Bases: **pbs_gestor.model.exceptions.BaseError**
Exception class for database table creation failure.

exception pbs_gestor.model.exceptions.**LogLineError**(message)
Bases: **pbs_gestor.model.exceptions.BaseError**
Exception class if log line is in wrong format.

exception pbs_gestor.model.exceptions.**PbsConfigNotFoundError**(message)
Bases: **pbs_gestor.model.exceptions.BaseError**
Exception class if PBS configuration file is not found.

pbs_gestor.model.orm_lib module

ORMs Library (uses sqlalchemy).

Provide API's to communicate with the Reporting Database(PostgreSQL).

Classes:

- BaseORMLib: Base class for database-related operations using sqlalchemy.
- Helpers: Helper class for for utility functions.

class pbs_gestor.model.orm_lib.**BaseORMLib**(tables, views, config, schema=None, connection_retries=2)
Bases: **object**
Deal with database: provide API's for database connection, etc.

***tables**
Dictionary holding the table name mapped to their table class.

Type: dict

***config**
The configuration object for database communication.

Type: dict

***schema**
String object holding the schema name.

Type: str

***create**
Flag used to specify whether to attempt to create table and schema.

Type: bool

***connection_retries**
Number of times to try connecting to database before

Type: int

exception is thrown.

*** _create**
Creates schema and table if the don't exist.

*** _create_table**
Creates table if they don't exist.

*** _create_schema**
Creates schema if they don't exist.

*** _database_engine**
Creates a engine from the database configs provided.

*** _set_session**
Creates a new session which is used to communicate with the database.

*** _reset_session**
Closes the old session and creates a new session.

*** _commit**
Commits changes to the database.

*** _rollback**
Rolls back the changes in case any exception is encountered.

*** _close**
Close the Reporting database connection.

*** _insert**
Performs insert within a transaction.

*** _is_session_valid**
Checks the session is valid or not.

*** _merge_by_query**
Performs merge based on the query dictionary.

last_table_ordered_column (obj**)**
Perform query for the first row of table ordered by column.

Parameters: **obj** –

Returns: instance

class pbs_gestor.model.orm_lib.**CreateView** (name, select)
Bases: **sqlalchemy.sql.base.Executable**, **sqlalchemy.sql.elements.ClauseElement**
Create an SQL view, not table.

class pbs_gestor.model.orm_lib.**Helpers**

Bases: **object**
Define various utility functions related to database operation.

*** schema_ref**
Concatenates schema to table name

static **schema_ref**(schema, table)
Concatenate schema name to table name.

Parameters:
- **schema** (*str*) – Schema name.
- **table** (*str*) – Table name.

Returns: Schema_name.Table_name

Return type: (str)

static **timestamp_to_iso_format**(timestamp)
Convert timestamp, if existing, to UTC ISO format.

Parameters: **timestamp** –

Returns: date&time

pbs_gestor.model.orm_lib.**visit_create_view**(element, compiler, **kw)
Create SQL view using raw SQL.

Module contents

Hold the basic structure of the Reporting Database tables.

Classes:

- PbsJob: This holds the structure of table with jobs and their attributes: name of the job, running time, vnode, user who started it, et cetera.
- PbsJobArr: This holds the structure of table with job resources: assigned to jobs, or used by jobs.
- PBSLog: This holds the structure of table with log handler records of which logs were processed when.

class pbs_gestor.model.**PbsJob**(**kwargs)
Bases: **sqlalchemy.ext.declarative.api.Base**
Hold the structure of table with jobs.
Describe when a job was added into a queue, started, or finished, and other attributes (except resources).

attributes = *['ji_jobid', 'ji_jobname', 'ji_user', 'ji_group', 'ji_project', 'ji_sv_name', 'ji_queue', 'ji_priority', 'ji_cr_time', 'ji_quetime', 'ji_runcount', 'ji_eligible_time', 'ji_start_time', 'ji_end_time', 'ji_sessionid', 'ji_exitstat', 'ji_exechost', 'ji_execvnode']*

ji_cr_time

ji_eligible_time

ji_end_time

ji_exechost

ji_execvnode

ji_exitstat

ji_group

ji_jobid

ji_jobname

ji_pbsjobidx

ji_priority

ji_project

ji_quetime

ji_queue

ji_runcount

ji_sessionid

ji_start_time

ji_sv_name

ji_user

p_key = *['ji_pbsjobidx']*

pbsjobarr

class pbs_gestor.model.**PbsJobArr**(**kwargs)
Bases: **sqlalchemy.ext.declarative.api.Base**
Holds table structure: resources assigned to jobs, or used by jobs.

attributes = *['ji_pbsjobidx', 'ji_arrresource', 'ji_arrvalue']*

ji_arrresource

ji_arrvalue

ji_pbsjobarridx

ji_pbsjobidx

p_key = *['ji_pbsjobarridx']*

class pbs_gestor.model.**PbsLog**(**kwargs)
Bases: **sqlalchemy.ext.declarative.api.Base**
This class holds the logs of past runs of loghandler.
These records are utilised to make sure that the same logs are not processed over and over again.

attributes = *['filename', 'start', 'end']*

end

filename

idx

p_key = *['idx']*

start

Submodules

pbs_gestor.pbs_loghandler module

PBS Log Handler: API's to read and process the PBS Accounting logs.

This module can be extended to read and process other types of logs related to PBS. Ex- Mom/Server logs.

*** DEFAULT_PBS_CONF**
The default path of the the PBS configuration file - /etc/pbs.conf

Type: str

*** PBS_JOB_VARS**
key is job variable and value is job's usage value.

Type: dict

*** rsrc_types**
The two types of resources from PBS logs

Type: list

are predefined in this list - 'Resource_List' and 'resources_used'.

The purpose of defining these two in a list so that it can be

extended to include other PBS resource types('resources_default',

'resources_available') etc.

Classes:

- **PbsLogHandler: class to provide the API's for reading and** processing of logs.

class pbs_gestor.pbs_loghandler.**PbsLogHandler**(day='today')
Bases: **object**
Read and process the PBS accounting logs, return job's attributes.
By default, start with reading the log file of the current date and then wait indeifinitely, processing logs as they arrive, and automatically switch to the next file as date changes. When called with a day different from today, read and process just one log file.

pbs_log_path
PBS Accounting log path

Type: str

log_file_name
starts with current date.

Type: str

logger
Application level logging object

Type: object

get_accounting_path()
returns the accounting files path of PBS

readline()
starts a generator which reads continuously today's

accounting log file.

process_log_line ()
processes the log line to create dicts.

static **get_accounting_path** ()
Find PBS accounting path under PBS_HOME/server_priv/accounting.
Get the PBS_HOME path from the default PBS conf file or else from the PBS conf file path set in the environment variable "PBS_CONF_FILE".
Append the PBS_HOME path with the accounting path and returns the final path.

Parameters: **None** –
Returns: str - pbs_accounting_path

get_first_log ()
Find the earliest/oldest log available for processing.

Parameters: **None** –
Returns: filename

is_update_needed ()
Check whether log file name needs to be updated.

Parameters: **None** –
Returns: True/False
Return type: bool

open_file ()
Open a file, and return it as a file object.
If not found, keeps retrying till found - or in manual mode, skips it,
because it's possible that some dates are missing in the past logs.

Returns: Returns a file object.
Return type: file

process_log_line (log_msg**)**
Process log line to form data structures for post-processing.

Parameters: **log_msg** – The log message to be processed
Returns: A tuple of three dictionaries formed after the processing of the log message.
Return type: tuple

readline ()
Read line from file, if there is a line, else wait or exit.
Check date and, if needed, update the instance variable "log_file_name" to read the current date's log file.

Parameters: **None** –
Yields: str - Log Line string read from PBS Accounting log file

Example

for log_line in read():
process(line)

update_log_file_name ()
Update the log file name.

pbs_gestor.reporting_database_connect module

Reporting Database Library: communicate with Reporting Database.

Classes:

- **ReportingDBLib: Derived from `BaseORMLib` used to communicate with the Reporting** Database (PostgreSQL).

class pbs_gestor.reporting_database_connect.**ReportingDBLib**(config)
Bases: **pbs_gestor.model.orm_lib.BaseORMLib**
Add to the Reporting DB records of jobs, and of log handler runs.

***config**
The configuration object for database communication.

Type: dict

*** _alter_config**
Modifies the configuration dictionary.

*** _save_job_info_data_mapper**
Modifies the data by applying user defined functions.

*** _insert_pbs_job_data**
Inserts data to pbsjob table, as defined in TABLES.

*** _insert_pbs_job_arr_data**
Inserts data to pbsjobarr table, as defined in TABLES.

*** _save_job_info**
Saves job info to Reporting Database.

*** _save_log_info**
Saves log info to Reporting Database, inserting it into pbslog table, as defined in TABLES.

*** is_connected_database**
Connection to database active or not.

*** write**
Method used to write to database.

*** read**
Method not yet implemented.

CONNECTION_RETRIES = *2*

close()
Close the session gracefully.

is_connected_database()
Check whether connection to database is active or not.

lastscan()
Find the last ever scan by log handler, using pbslog table.

Parameters: **None** –
Returns: the largest/latest date in the pbslog table
Return type: filename

read(key)
Read is not yet implemented for this module.

write(key, data)
Save data into Reporting Database.

Parameters:
- **key** (*str*) – key is used to map to respective function to write the data.
- **data** (*dict*) – Data to be written to database.

Returns: None

Raises:
- **`ValueError`**
- **`KeyError`**

pbs_gestor.utils module

Constants or utility functions to be used across modules of PBS Gestor.

*** CONFIGS_DIR_PATH**
The path where all the configuration files

required by PBS Gestor reside. The current codebase works with

having the configs directory and having the json files like

pbs_gestor_config.**json**

*** GESTOR_CONFIG**
Dictionary of pbs gestor related config

*** DEFAULT_PBS_GESTOR_CONF**
The default path of the the PBS Gestor

configuration file - gestor/configs/psb_gestor_config.json

*** LOGGING_CONFIG**
The logger configuration for PBS Gestor

pbs_gestor.utils.**create_user_config**(config_file)
Create the user's config file.

pbs_gestor.utils.**get_config**()
Find and read configuration file and return its contents.

Module contents

PBS Gestor Src utilities module: Read logs and record to database.

Index

Python Module Index

g

p