PBS Gestor documentation

version

Altair

September 18, 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.

`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.`**`get_input`** ()

Get arguments provided.

Parameters: **args** – list of arguments provided by user to application

Returns: input argument tilldate (str): input argument

Return type: fromdate (str)

`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`** (`fromdate`, `tilldate`, `database_handler`)

Parse arguments provided and return a list of log handlers.

Parameters:

- **tilldate** (*fromdate,*) – 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** –

`gestor.`**`str_to_date`** (`fromday`, `dath`)

Parse day given by user and convert it to date.

pbs_gestor package

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

Subpackages

pbs_gestor.model package

Exceptions and SQLalchemy-based ORM Library.

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: **`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.`**`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

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

Hold structure of database, constants and utility functions to be used across PBS Gestor modules.

Include schema, tables and views.

Classes corresponding to tables:

- PbsJob: This table contains jobs and their attributes: name of the job, running time, vnode, user who started it… - The table contains 'CONSTRAINT unq_pbsjob UNIQUE (ji_jobid)' so that duplicate records are not created for the same job.
- PbsJobArr: This table contains job resources: assigned to jobs (l_* resource names), or used by jobs (u_* resource names). One column contains unique job identifier for relationship to PbsJob table, another column contains name of resource, and the last column contains value of resource; names and values of resources are represented as text.
- PBSLog: This supplementary table contains log handler records of which logs were processed when.

Upon connecting to the PostgreSQL database

- for example, with the command psql -h hostname -p port -U username -d databasename with default database connection settings it would be psql -h localhost -p 5432 -U postgres -d pbsjoblogsdb you would be able to query the contents of the tables inside the database, such as:
- SELECT * FROM schema.pbsjob;

OR

- SELECT * FROM schema.pbsjob WHERE ji_user='username'; where username is name of a user whose job history you want to look up

OR

- SELECT count(*) FROM schema.pbsjob WHERE ji_user='username'; if you want to count how many jobs the user has run in the past

OR

- any other SQL query

Additionally, there is also a View created, with name specified in the configuration file, inside the same schema as tables.

- PbsFlatView contains a subset of information from PbsJobArr, pivoting the table with PostgreSQL's Crosstab function from Tablefunc extension so that requested and used resources would be organised into the following columns: 'mem', 'ncpus', 'walltime', 'cput', 'nodect', 'cpupercent', et cetera instead of being contained in a Entity–attribute–value model as in PbsJobArr table. See: https://www.postgresql.org/docs/9.2/tablefunc.html#AEN152349 Resources requested by job are prefixed with l, to be differentiated from resources used by job. It is also joined with PbsJob table so that Job ID would be available as one of the columns. See INNER JOIN on the following documentation page: https://www.postgresql.org/docs/9.2/queries-table-expressions.html#QUERIES-JOIN The values for the resources, such as ncpus or walltime, are cast to appropriate data types, so that arithmetic operations can be performed on them.

This view can be queried similarly to tables, for example:

- SELECT * FROM schema.pbsflatviewreqjoin WHERE l_nodect > 1; displays list of all jobs which requested/used more than one node.

- **For troubleshooting, try to connect to the database with the same settings**

  (hostname, port, username, databasename) as are inside pbs_gestor_config.json file.

  - For ease of troubleshooting, application prints out the location from which it is reading the configuration file pbs_gestor_config.json . As noted earlier, if you move the file to a different location, please create or update the environment variable PBS_GESTOR_CONF with the location of pbs_gestor_config.json .

  Check whether schema 'schema' exists by querying list of schemas with '\dn' command inside psql client.

  Check whether tables exist with '\dt schema.*' command.

  Check whether tablefunc extension is installed with the following command:

  - \dx+ tablefunc

    If it says 'Did not find any extension named "tablefunc"', then Tablefunc extension is not installed and Crosstab function is not available. If the Tablefunc extension is installed successfully, then there will be list of objects in extension "tablefunc", such as connectby, crosstab and normal_rand.

  OR

  - SELECT count(*) FROM information_schema.routines WHERE routine_name LIKE 'crosstab%';

    If it is 0, then Tablefunc extension is not installed and Crosstab function is not available. If it is above 0 - for example, 6 - then Crosstab function is available, which is good.

  If Tablefunc extension is installed, then you may also wish to check where it is installed:

  - SELECT routine_schema FROM information_schema.routines WHERE routine_name LIKE 'crosstab%';

  OR

  - SELECT specific_schema FROM information_schema.routines WHERE routine_name LIKE 'crosstab%';

    (these two similar commands usually return the same result)

  to see which schema contains the installed Crosstab function.

  - Application expects this result to be either 'schema' or, failing that, something like 'public' so that the function is inside PostgreSQL's search_path. See: https://www.postgresql.org/docs/9.2/runtime-config-client.html#GUC-SEARCH-PATH

* **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 - ~/.config/pbs_gestor/psb_gestor_config.json

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

class `pbs_gestor.utils.`**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.utils.**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.utils.**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

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.

Index

Python Module Index