pypuppetdbquery Documentation

Release 0.9.1

Chris Boot

June 27, 2016

Contents

Contents:

CHAPTER 1

pypuppetdbquery

A port of Erik Dalén's PuppetDB Query language to Python. This module is designed to be paired with pypuppetdb but does not depend on it directly.

This module is a Python implementation of the query language also implemented in puppet-puppetdbquery (in Ruby) and node-puppetdbquery (in JavaScript/NodeJS).

Please see the pypuppetdbquery documentation courtesy of Read the Docs and Sphinx.

1.1 Installation

You can install this package from source or from PyPi.

```
$ pip install pypuppetdbquery
```

```
$ git clone https://github.com/bootc/pypuppetdbquery
$ python setup.py install
```

If you wish to hack on it clone the repository but after that run:

```
$ pip install -r requirements.txt
$ pip install -r requirements-dev.txt
```

This will install all the runtime requirements of pypuppetdbquery and the dependencies for the test suite and generation of documentation.

1.2 Usage Example

```
import pypuppetdb
import pypuppetdbquery

pdb = pypuppetdb.connect()

pdb_ast = pypuppetdbquery.parse(
    '(processorcount=4 or processorcount=8) and kernel=Linux')

for node in pdb.nodes(query=pdb_ast):
    print(node)
```

For further examples, see the Examples section of the pypuppetdbquery documentation.

1.3 License

CHAPTER 2

API Documentation

All the user-facing components of this package are intended to be found in the pypuppetdbquery package documentation. The other sub-modules and the test suite are documented here for completeness.

2.1 pypuppetdbquery package

This Python package contains a port of Erik Dalén's PuppetDB Query language to Python.

All the user-facing components of this package are intended to be found in this module itself rather than any of the sub-modules.

pypuppetdbquery.**parse**(*s*, *json=True*, *mode='nodes'*, *lex_options=None*, *yacc_options=None*)

Parse a PuppetDBQuery-style query and transform it into a PuppetDB "AST" query.

This function is intented to be the primary entry point for this package. It wraps up all the various components of this package into an easy to consume format. The output of this function is designed to be passed directly into `pypuppetdb`.

For examples of the query syntax see puppet-puppetdbquery and node-puppetdbquery by Erik Dalén.

Parameters

- **s** (*str*) – The query to parse and transform
- **mode** (*str*) – The PuppetDB endpoint being queried
- **json** (*bool*) – Whether to JSON-encode the PuppetDB AST result
- **lex_options** (*dict*) – Options passed to `ply.lex.lex()`
- **yacc_options** (*dict*) – Options passed to `ply.yacc.yacc()`

pypuppetdbquery.**query_fact_contents**(*pdb*, *s*, *facts=None*, *raw=False*, *lex_options=None*, *yacc_options=None*)

Helper to query PuppetDB for fact contents (i.e. within structured facts) on nodes matching a query string.

Adjusts the query to return only those strucutred fact keys requested in the function call.

The facts listed in the *facts* list are run through the query parser and treated as "identifier paths". This means the same rules apply as for within the query language, e.g. `foo.bar` or `foo.*` or even `foo.~"bar.*"`.

If *raw* is *False* (the default), the return value is a `dict` with node names as keys containing a `dict` of flattened fact paths to fact values. If *True* it returns raw query output: a list of dictionaries (see the PuppetDB fact-contents documentation).

Note: This function can only be used to search deeply within structured facts. It cannot return a whole structured fact, only individual elements within—but you can return all the elements within a structured fact if you want by using a regex match.

Parameters

- **pdb** (*pypuppetdb.api.BaseAPI*) – pypuppetdb connection to query from
- **s** (*str*) – The query string (may be empty to query all nodes)
- **facts** (*Sequence*) – List of fact paths to search for
- **raw** (*bool*) – Whether to skip post-processing the facts into a dict structure grouped by node
- **lex_options** (*dict*) – Options passed to `ply.lex.lex()`
- **yacc_options** (*dict*) – Options passed to `ply.yacc.yacc()`

`pypuppetdbquery.`**query_facts**(*pdb*, *s*, *facts=None*, *raw=False*, *lex_options=None*, *yacc_options=None*)

Helper to query PuppetDB for facts on nodes matching a query string.

Adjusts the query to return only those facts requested in the function call.

The fact names included in *facts* may be names or regular expressions. If the string starts and ends with `/`, it is considered a regular expression (e.g. `/^lsb/`).

If *raw* is *False* (the default), the return value is a `dict` with node names as keys containing a `dict` of fact names to fact values. If *True* it returns raw `pypuppetdb.types.Fact` objects as `pypuppetdb.api.BaseAPI.nodes()` does.

Note: This function can return only full facts, not elements of structured facts. For example, only the whole `os` fact may be returned but not the `os.family` key within the larger structured fact. If you need to do this, look at *query_fact_contents()*.

Parameters

- **pdb** (*pypuppetdb.api.BaseAPI*) – pypuppetdb connection to query from
- **s** (*str*) – The query string (may be empty to query all nodes)
- **facts** (*Sequence*) – List of fact names to search for
- **raw** (*bool*) – Whether to skip post-processing the facts into a dict structure
- **lex_options** (*dict*) – Options passed to `ply.lex.lex()`
- **yacc_options** (*dict*) – Options passed to `ply.yacc.yacc()`

2.2 pypuppetdbquery.ast module

Abstract Syntax Tree (AST) for the PuppetDBQuery language. These simple classes are used by the *pypuppetdbquery.parser.Parser* in order to represent the parsed syntax tree.

class `pypuppetdbquery.ast.`**AndExpression**(*left*, *right*)

Bases: *pypuppetdbquery.ast.BinaryExpression*

class pypuppetdbquery.ast.**BinaryExpression**(*left*, *right*)
Bases: *pypuppetdbquery.ast.Node*

class pypuppetdbquery.ast.**BlockExpression**(*expression*)
Bases: *pypuppetdbquery.ast.UnaryExpression*

class pypuppetdbquery.ast.**Comparison**(*operator*, *left*, *right*)
Bases: *pypuppetdbquery.ast.Expression*

class pypuppetdbquery.ast.**Date**(*value*)
Bases: *pypuppetdbquery.ast.Literal*

class pypuppetdbquery.ast.**Expression**
Bases: *pypuppetdbquery.ast.Node*

class pypuppetdbquery.ast.**Identifier**(*name*)
Bases: *pypuppetdbquery.ast.Node*

class pypuppetdbquery.ast.**IdentifierPath**(*components*)
Bases: *pypuppetdbquery.ast.Node*

class pypuppetdbquery.ast.**Literal**(*value*)
Bases: *pypuppetdbquery.ast.Node*

class pypuppetdbquery.ast.**Node**
Bases: object

class pypuppetdbquery.ast.**NotExpression**(*expression*)
Bases: *pypuppetdbquery.ast.UnaryExpression*

class pypuppetdbquery.ast.**OrExpression**(*left*, *right*)
Bases: *pypuppetdbquery.ast.BinaryExpression*

class pypuppetdbquery.ast.**ParenthesizedExpression**(*expression*)
Bases: *pypuppetdbquery.ast.UnaryExpression*

class pypuppetdbquery.ast.**Query**(*expression*)
Bases: *pypuppetdbquery.ast.Node*

class pypuppetdbquery.ast.**RegexpIdentifier**(*name*)
Bases: *pypuppetdbquery.ast.Identifier*

class pypuppetdbquery.ast.**RegexpNodeMatch**(*value*)
Bases: *pypuppetdbquery.ast.Expression*

class pypuppetdbquery.ast.**Resource**(*res_type*, *title*, *exported*, *parameters=None*)
Bases: *pypuppetdbquery.ast.Expression*

class pypuppetdbquery.ast.**Subquery**(*endpoint*, *expression*)
Bases: *pypuppetdbquery.ast.Node*

class pypuppetdbquery.ast.**UnaryExpression**(*expression*)
Bases: *pypuppetdbquery.ast.Node*

2.3 pypuppetdbquery.evaluator module

class pypuppetdbquery.evaluator.**Evaluator**
Bases: object

Converts a *pypuppetdbquery.ast* Abstract Syntax Tree into a PuppetDB native AST query.

DECAMEL_RE = re.compile('(?!^)([A-Z]+)')
Regular expression used when converting CamelCase class names to underscore_separated names.

evaluate(*ast*, *mode='nodes'*)
Process a parsed PuppetDBQuery AST and return a PuppetDB AST.

The resulting PuppetDB AST is a native Python list. It will need converting to JSON (using `json.dumps()`) before it can be used with PuppetDB.

Parameters

- **ast** (*pypuppetdbquery.ast.Query*) – Root of the AST to evaulate
- **mode** (*str*) – PuppetDB endpoint to target

Returns PuppetDB AST

Return type list

2.4 pypuppetdbquery.lexer module

exception pypuppetdbquery.lexer.**LexException**(*message*, *position*)
Bases: Exception

Raised for errors encountered during lexing.

Such errors may include unknown tokens or unexpected EOF, for example. The position of the lexer when the error was encountered (the index into the input string) is stored in the *position* attribute.

class pypuppetdbquery.lexer.**Lexer**(***kwargs*)
Bases: object

Lexer for the PuppetDBQuery language.

This class uses `ply.lex.lex()` to implement the lexer (or tokenizer). It is used by *pypuppetdbquery.parser.Parser* in order to process queries.

The arguments to the constructor are passed directly to `ply.lex.lex()`.

Note: Many of the docstrings in this class are used by `ply.lex` to build the lexer. These strings are not particularly useful for generating documentation from, so the built documentation for this class may not be very useful.

---

input(*s*)
Reset and supply input to the lexer.

Tokens then need to be obtained using *token()* or the iterator interface provided by this class.

next()
Implementation of `iterator.next()`.

Return the next item from the container. If there are no further items, raise the `StopIteration` exception.

t_ASTERISK = '*'

t_AT = '@'

t_BOOLEAN(*t*)
true|false

t_DOT = '\\.'

t_EQUALS = '='

t_EXPORTED = '@@'

t_FLOAT(*t*)
-?d+.d+

t_GREATERTHAN = '>'

t_GREATERTHANEQ = '>='

t_HASH = '[#]'

t_LBRACE = '{'

t_LBRACK = '\\['

t_LESSTHAN = '<'

t_LESSTHANEQ = '<='

t_LPAREN = '\\('

t_MATCH = '~'

t_NOTEQUALS = '!='

t_NOTMATCH = '!~'

t_NUMBER(*t*)
-?d+

t_RBRACE = '}'

t_RBRACK = '\\]'

t_RPAREN = '\\)'

t_STRING_bareword(*t*)
[-w_:]+

t_STRING_double_quoted(*t*)
"(\.|[^\"])*"

t_STRING_single_quoted(*t*)
'(\.|[^\'])*'

t_error(*t*)

t_ignore = ' \t\n\r\x0c\x0b'

t_keyword(*t*)
not|and|or

token()
Obtain one token from the input.

tokens = ('LPAREN', 'RPAREN', 'LBRACK', 'RBRACK', 'LBRACE', 'RBRACE', 'EQUALS', 'NOTEQUALS', 'MAT
List of token names handled by the lexer.

2.5 pypuppetdbquery.parser module

exception pypuppetdbquery.parser.**ParseException**(*message*, *position*)
Bases: Exception

Raised for errors encountered during parsing.

The position of the lexer when the error was encountered (the index into the input string) is stored in the *position* attribute.

class pypuppetdbquery.parser.**Parser**(*lex_options=None*, *yacc_options=None*)
Bases: object

Parser for the PuppetDBQuery language.

This class uses ply.yacc.yacc() to implement the parser. In concert with *pypuppetdbquery.lexer.Lexer*, it produces an Abstract Syntax Tree (AST) as declared in *pypuppetdbquery.ast*.

Parameters

- **lex_options** (*dict*) – Passed as keyword arguments to *pypuppetdbquery.lexer.Lexer*
- **yacc_options** (*dict*) – Passed as keyword arguments to ply.yacc.yacc()

Note: Many of the docstrings in this class are used by ply.yacc to build the parser. These strings are not particularly useful for generating documentation from, so the built documentation for this class may not be very useful.

---

p_block_expr(*p*)
block_expr : LBRACE expr RBRACE

p_boolean(*p*)
boolean : BOOLEAN

p_comparison_expr(*p*)
comparison_expr : identifier_path comparison_op literal

p_comparison_op(*p*)

comparison_op [MATCH]

NOTMATCH
EQUALS
NOTEQUALS
GREATERTHAN
GREATERTHANEQ
LESSTHAN
LESSTHANEQ

p_empty(*p*)
empty :

p_error(*p*)

p_expr(*p*)

expr [resource_expr]

comparison_expr

subquery

p_expr_and(*p*)
expr : expr AND expr

p_expr_identifier_path(*p*)
expr : identifier_path

p_expr_not(*p*)
expr : NOT expr

p_expr_or(*p*)
expr : expr OR expr

p_expr_parenthesized(*p*)
expr : LPAREN expr RPAREN

p_float(*p*)
float : FLOAT

p_identifier(*p*)

identifier [string]

integer

p_identifier_path(*p*)
identifier_path : identifier

p_identifier_path_nested(*p*)
identifier_path : identifier_path DOT identifier

p_identifier_regexp(*p*)
identifier : MATCH string

p_identifier_wild(*p*)
identifier : ASTERISK

p_integer(*p*)
integer : NUMBER

p_literal(*p*)

literal [boolean]

string

integer

float

p_literal_date(*p*)
literal : AT string

p_query(*p*)

query [expr]

empty

p_resource_expr(*p*)
resource_expr : string LBRACK identifier RBRACK

p_resource_expr_exported(*p*)
resource_expr : EXPORTED string LBRACK identifier RBRACK

p_resource_expr_exported_param(*p*)
resource_expr : EXPORTED string LBRACK identifier RBRACK block_expr

p_resource_expr_param(*p*)
resource_expr : string LBRACK identifier RBRACK block_expr

p_string(*p*)
string : STRING

p_subquery_block(*p*)
subquery : HASH string block_expr

p_subquery_comparison(*p*)
subquery : HASH string DOT comparison_expr

parse(*text*, *debug=0*)
Parse the input string and return an AST.

Parameters

- **text** (*str*) – The query to parse
- **debug** (*bool*) – Output detailed information during the parsing process

Returns An Abstract Syntax Tree

Return type *pypuppetdbquery.ast.Query*

precedence = (('left', 'OR'), ('left', 'AND'), ('left', 'EQUALS', 'MATCH', 'LESSTHAN', 'GREATERTHAN'), ('right',
Precedence rules in lowest to highest order

start = 'query'
Non-terminal to use as the starting grammar symbol

2.6 Test Suite

class test_frontend.**TestFrontendParse**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test cases targetting *pypuppetdbquery.parse()*.

test_empty_queries()

test_simple_json()

test_simple_raw()

class test_frontend.**TestFrontendQueryFactContents**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test cases targetting *pypuppetdbquery.query_fact_contents()*.

test_raw_output()

test_with_query_and_facts_list()

test_without_either()

test_without_query()

class test_frontend.**TestFrontendQueryFacts**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test cases targetting *pypuppetdbquery.query_facts()*.

test_query_facts_in_raw_mode()

test_query_facts_with_facts_list_only()

test_query_facts_with_query_and_facts_list()

test_query_facts_with_query_and_facts_list_regex()

test_query_facts_without_query_or_facts()

class test_integration.**TestIntegration**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test cases for the integrated combination of *pypuppetdbquery.lexer.Lexer*, *pypuppetdbquery.parser.Parser*, and *pypuppetdbquery.evaluator.Evaluator*.

setUp()

test_boolean_values()

test_capitalize_class_names()

test_dates_in_queries()

test_does_not_wrap_subquery_with_mode_none()

test_double_quoted_strings()

test_empty_queries()

test_equals_operator()

test_escape_non_match_parts_on_structured_facts_with_match_op()

test_float_values()

test_greater_than_eq_operator()

test_greater_than_operator()

test_less_than_eq_operator()

test_less_than_operator()

test_match_operator()

test_negate_expressions()

test_node_subqueries()

test_node_subqueries_with_block_of_conditions()

test_node_subqueries_with_fact_query()

test_node_subquery_fact_field()

test_not_equals_operator()

test_not_match_operator()

test_precedence_a()

test_precedence_b()

test_precedence_within_resource_parameter_queries_a()

test_precedence_within_resource_parameter_queries_b()

test_resource_queries_for_exported_resources()

test_resource_queries_for_exported_resources_with_parameters()

test_resource_queries_with_regexp_title_matching()

test_resource_queries_with_tags()

test_resource_queries_with_type_and_title()

test_resource_queries_with_type_title_and_parameters()

test_single_quoted_strings()

test_single_string_expressions()

test_structured_facts()

test_structured_facts_with_array_component()

test_structured_facts_with_match_operator()

test_structured_facts_with_wildcard_operator()

class test_lexer.**TestLexer**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test cases for *pypuppetdbquery.lexer.Lexer*.

setUp()

test_all_tokens()

test_empty_queries()

test_invalid_input()

class test_parser.**TestParster**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test cases for *pypuppetdbquery.parser.Parser*.

setUp()

test_boolean_values()

test_dates_in_queries()

test_double_quoted_strings()

test_empty_queries()

test_float_values()

test_invalid_input()

test_invalid_input_eof()

test_logical_operators()

test_negate_expression()

test_node_subqueries()

test_node_subqueries_with_block_of_conditions()

test_resource_queries_for_exported_resources()

test_resource_queries_for_exported_resources_with_parameters()

test_resource_queries_with_type_and_regexp_identifier()

test_resource_queries_with_type_title_and_parameters()

test_single_string_expressions()

test_structured_facts_with_wildcard_operator()

CHAPTER 3

Examples

Basic query for nodes using `pypuppetdb`:

```
import pypuppetdb
import pypuppetdbquery

pdb = pypuppetdb.connect()

pdb_ast = pypuppetdbquery.parse(
    '(processorcount=4 or processorcount=8) and kernel=Linux')

for node in pdb.nodes(query=pdb_ast):
    print(node)
```

Obtain named facts from nodes matching a query (using `pypuppetdb`):

```
import pypuppetdb
import pypuppetdbquery

pdb = pypuppetdb.connect()

node_facts = pypuppetdbquery.query_facts(
    pdb,
    '(processorcount=4 or processorcount=8) and kernel=Linux',
    ['/^lsb/', 'architecture'])

for node in node_facts:
    facts = node_facts[node]
    print(node, facts)
```

Obtain selected structured fact values from nodes matching a query (using `pypuppetdb`):

```
import pypuppetdb
import pypuppetdbquery

pdb = pypuppetdb.connect()

node_facts = pypuppetdbquery.query_fact_contents(
    pdb,
    '(processorcount=4 or processorcount=8) and kernel=Linux',
    ['system_uptime.days', 'os.lsb.~"dist.*"'])

for node in node_facts:
    facts = node_facts[node]
    print(node, facts)
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

p

t

Q

R

S

T

U