hermes Documentation

Release 0.1

Transifex

March 04, 2015

Contents

Hermes is a Postgres-talking, event-driven, failure-handling Python library. Its main purpose is to enable the easy implementation of resilient Python processes which require communication with Postgres. It defines a base-layer which you can build as little or as much as you like on top of.

It's been used at Transifex to fulfil a number of roles, one of them including a Postgres -> Elasticsearch river.

CHAPTER 1

Compatibility

*nix operating system which supports the select function.

Postgresql 9.0+ is required to support LISTEN/NOTIFY commands.

CHAPTER 2

Installation

```
pip install hermes-pg
```

CHAPTER 3

Usage

Most users will just need to define some form of process to run when an event is emitted. This can be achieved by defining a processor object and supplying that to the Client object like so:

```
from hermes.components import Component

class Processor(Component):
    def execute(self):
        pass

    def execution
```

CHAPTER 4

Contents

4.1 Client

class hermes.client.**Client**(*dsn*, *watch_path=None*, *failover_files=None*)

Bases: multiprocessing.process.Process, watchdog.events.FileSystemEventHandler

Hermes client. Responsible for Listener and Processor components. Provides functions to start/stop both itself and its components. In addition, it is also capable of receiving file-system events via the 'watchdog' library.

General procedure:

```
1. Starts both the Process and Listener components.
2. Listen and act upon exit/error notifications from components
3. Listen for file-system events and act accordingly.
```

To make the client listen for Postgres 'recovery.conf, recovery.done' events:

```
from hermes.client import Client

dsn = {'database': 'example_db',
       'host': '127.0.0.1',
       'port': 5432,
       'user': 'example',
       'password': 'example'}

watch_path = '/var/lib/postgresql/9.4/main/'
failover_files = ['recovery.done', 'recovery.conf']

client = Client(dsn, watch_path, failover_files)

# Add processor and listener
...

# Start the client
client.start()
```

Or, if you decide you don't want to use a file watcher, then you can omit those parameters. However, the Client will still perform master/slave checks if a problem is encountered:

```
from hermes.client import Client

dsn = {'database': 'example_db',
       'host': '127.0.0.1',
       'port': 5432,
```

```
        'user': 'example',
        'password': 'example'}

client = Client(dsn)

# Add processor and listener
...

# Start the client
client.start()
```

Parameters

- **dsn** – a Postgres-compatible DSN dictionary
- **watch_path** – the directory to monitor for filechanges. If None, then file monitoring is disabled.
- **failover_files** – a list of files which, when modified, will cause the client to call `execute_role_based_procedure()`

add_listener(*listener*)

Parameters **listener** – A `Component` object which will listen for notifications from Postgres and pass an event down a queue.

Raises `InvalidConfigurationException` if the provided listener is not a subclass of `Component`

add_processor(*processor*)

Parameters **processor** – A `Component` object which will receive notifications and run the `execute()` method.

Raises `InvalidConfigurationException` if the provided processor is not a subclass of `Component`

execute_role_based_procedure()

Starts or stops components based on the role (Master/Slave) of the Postgres host.

Implements a binary exponential backoff up to 32 seconds if it encounters a FATAL connection error.

on_any_event(*event*)

Listens to an event passed by 'watchdog' and checks the current master/slave status

Parameters **event** – a `FileSystemEvent`

object passed by 'watchdog' indicating an event change within the specified directory.

run()

Performs a `select()` on the components' error queue. When a notification is detected, the client will log the message and then calculate if the Postgres server is still a Master - if not, the components are shutdown.

start()

Starts the Client, its Components and the directory observer

Raises `InvalidConfigurationException`

terminate()

Terminates each component, itself, and the directory observer.

4.2 Components

class hermes.components.**Component**(*notification_pipe*, *error_strategy*, *exit_queue*, *error_queue*, *pg_connector*)

Bases: multiprocessing.process.Process

cleanup()
Disconnects the attached connector and sets 'self.cleaned' to True.

join(*timeout=None*)
Joins the component if 'self.started' is set to True. Calls 'self.cleanup' regardless of outcome.

run()

terminate()
Terminates the component by putting an entry on the 'exit_queue' and joining the current thread.

4.3 Connectors

class hermes.connectors.**PostgresConnector**(*dsn*, *cursor_factory=<class 'psycopg2.extras.DictCursor'>*)

Postgres-talking connection wrapper. A thin wrapper to encapsulate the complexity of creating, re-creating, and disconnecting from a Postgres database.

Creating a PostgresConnector is done like so:

```
from psycopg2.extras import DictCursor

# Define a Postgres DSN dictionary
dsn = {'database': 'example_db',
       'host': '127.0.0.1',
       'port': 5432,
       'user': 'example',
       'password': 'example'}

cursor_factory = DictCursor

# Pass the DSN to the PostgresConnector's constructor
connector = PostgresConnector(dsn, cursor_factory=cursor_factory)
```

Parameters

- **dsn** – a Postgres-compatible DSN dictionary
- **cursor_factory** – a callable cursor subclass

disconnect()
Disconnects from the Postgres instance unless it is already disconnected.

is_server_master()
Enquires as to whether this server is a master or a slave.

Returns a boolean indicating whether the server is master.

pg_connection
Connects to the Postgres host, if a connection does not exist or is closed, using the the DSN provided in the constructor.

Automatically sets connection isolation level to AUTOCOMMIT.

Returns a `connection` object

pg_cursor
Opens a postgres cursor if it doesn't exist or is closed. Otherwise returns the current cursor.

Returns a psycopg2 `cursor` instance or subclass as defined by the cursor_factory passed to the constructor

4.4 Listeners

class hermes.listeners.**NotificationListener**(*pg_connector*, *notif_channel*, *notif_queue*, *error_strategy*, *error_queue*, *exit_queue*, *fire_on_start=True*)
Bases: `hermes.components.Component`

A listener to detect event notifications from Postgres and pass onto to a processor.

Parameters

- **pg_connector** – a PostgresConnector object.
- **notif_channel** – the string representing the notification channel to listen to updates on.
- **notif_queue** – the notification queue to pass events to
- **error_strategy** – the error strategy that this listener should follow.
- **error_queue** – the error queue to inform the client on.
- **exit_queue** – the queue to listen for exit events on.

4.5 Strategies

class hermes.strategies.**ErrorStrategy**
Abstract strategy for handling errors returned from components

handle_exception(*error*)
An abstract method that must be overidden by subclasses.

Must return a tuple of: (Boolean indicating if the exception was handled, a string message)

4.6 Exceptions

exception hermes.exceptions.**InvalidConfigurationException**
Bases: `exceptions.Exception`

4.7 Changelog

4.7.1 0.1 (2014-10-11)

Initial release.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

h