This repository Search Explore Gist Blog Help sethc23

openresty / **drizzle-nginx-module**

Watch 21 Star 177 Fork 23

an nginx upstream module that talks to mysql and drizzle by libdrizzle
http://wiki.nginx.org/NginxHttpDrizzleModule

421 commits **3** branches **58** releases **4** contributors

Code
Issues 7
Pull Requests 1
Wiki
Pulse
Graphs

branch: master drizzle-nginx-module / +

doc: claimed that we work with nginx 1.7.4.

agentzh authored 20 days ago latest commit 19a622d4ed

doc	doc: claimed that we work with nginx 1.7.4.	20 days ago
src	minor coding style fixes around the labels.	20 days ago
t	check for alert messages in error.log in keepalive.t.	2 years ago
util	minor code cleanup.	2 years ago
.gitignore	updated .gitignore a bit.	a year ago
Changes	added the Changes file.	3 years ago
README.markdown	doc: claimed that we work with nginx 1.7.4.	20 days ago
config	fixed the automatic libdrizzle searching algorithm in the config file…	3 years ago
valgrind.suppress	suppressed a valgrind false positive in libdl.	5 months ago

HTTPS clone URL

`https://github.com`

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

README.markdown

Name

drizzle-nginx-module - Upstream module for talking to MySQL and Drizzle directly

This module is not distributed with the Nginx source. See the installation instructions.

Table of Contents

Status

This module is already production ready and is powering the Taobao LineZing site.

Version

This document describes ngx_drizzle v0.1.7 released on 19 December 2013.

Synopsis

```
http {
    ...

    upstream cluster {
        # simple round-robin
        drizzle_server 127.0.0.1:3306 dbname=test
             password=some_pass user=monty protocol=mysql;
        drizzle_server 127.0.0.1:1234 dbname=test2
             password=pass user=bob protocol=drizzle;
    }
```

```
        upstream backend {
            drizzle_server 127.0.0.1:3306 dbname=test
                 password=some_pass user=monty protocol=mysql;
        }

        server {
            location /mysql {
                set $my_sql 'select * from cats';
                drizzle_query $my_sql;

                drizzle_pass backend;

                drizzle_connect_timeout    500ms; # default 60s
                drizzle_send_query_timeout 2s;    # default 60s
                drizzle_recv_cols_timeout  1s;    # default 60s
                drizzle_recv_rows_timeout  1s;    # default 60s
            }

            ...

            # for connection pool monitoring
            location /mysql-pool-status {
                allow 127.0.0.1;
                deny all;

                drizzle_status;
            }
        }
    }
```

Back to TOC

Description

This is an nginx upstream module integrating libdrizzle into Nginx in a non-blocking and streamming way.

Essentially it provides a very efficient and flexible way for nginx internals to access MySQL, Drizzle, as well as other RDBMS's that support the Drizzle or MySQL wired protocol. Also it can serve as a direct REST interface to those RDBMS backends.

This module does not generate human-readable outputs, rather, in a binary format called Resty-DBD-Stream (RDS) designed by ourselves. You usually need other components, like rds-json-nginx-module, rds-csv-nginx-module, or lua-rds-parser, to work with this module. See Output Format for details.

Back to TOC

Keepalive connection pool

This module also provides a builtin per-worker connection pool mechanism for MySQL or Drizzle TCP connections.

Here's a sample configuration:

```
    upstream backend {
        drizzle_server 127.0.0.1:3306 dbname=test
             password=some_pass user=monty protocol=mysql;
        drizzle_keepalive max=100 mode=single overflow=reject;
    }
```

For now, the connection pool uses a simple LIFO algorithm to assign idle connections in the pool. That is, most recently (successfully) used connections will be reused first the next time. And new idle connections will always replace the oldest idle connections in the pool even if the pool is already full.

See the drizzle_keepalive directive for more details.

Back to TOC

Last Insert ID

If you want to get LAST_INSERT_ID, then ngx_drizzle already returns that automatically for you when you're doing a SQL insert query. Consider the following sample `nginx.conf` snippet:

```
    location /test {
        echo_location /mysql "drop table if exists foo";
        echo;
        echo_location /mysql "create table foo (id serial not null, primary key (id), val r
        echo;
        echo_location /mysql "insert into foo (val) values (3.1415926);";
        echo;
        echo_location /mysql "select * from foo;";
        echo;
    }

    location /mysql {
        drizzle_pass backend;
        drizzle_module_header off;
        drizzle_query $query_string;
        rds_json on;
    }
```

Then request `GET /test` gives the following outputs:

```
    {"errcode":0}
    {"errcode":0}
    {"errcode":0,"insert_id":1,"affected_rows":1}
    [{"id":1,"val":3.1415926}]
```

You can see the `insert_id` field (as well as the `affected_rows` field in the 3rd JSON response.

Back to TOC

Directives

Back to TOC

drizzle_server

syntax: *drizzle_server <host> user=<user> password=<pass> dbname=<database>*

syntax: *drizzle_server <host>:<port> user=<user> password=<pass> dbname=<database> protocol=<protocol> charset=<charset>*

default: *no*

context: *upstream*

Directive assigns the name and the parameters of server. For the name it is possible to use a domain name, an address, with an optional port (default: 3306). If domain name resolves to several addresses, then all are used.

The following options are supported:

user= `<user>` MySQL/Drizzle user name `<user>` for login.

password= `<pass>` Specify mysql password `<pass>` for login. If you have special characters like `#` or spaces in your password text, then you'll have to quote the whole key-value pair with either single-quotes or double-quotes, as in

```
drizzle_server 127.0.0.1:3306 user=monty "password=a b#1"
        dbname=test protocol=mysql;
```

dbname= `<database>` Specify default MySQL database `<database>` for the connection. Note that MySQL does allow referencing tables belonging to different databases by qualifying table names with database names in SQL queries.

protocol= `<protocol>` Specify which wire protocol to use, `drizzle` or `mysql` . Default to `drizzle` .

charset= `<charset>` Explicitly specify the character set for the MySQL connections. Setting this option to a non-empty value will make this module send out a `set names '<charset>'` query right after the mysql connection is established. If the default character encoding of the MySQL connection is already what you want, you needn't set this option because it has extra runtime cost. Here is a small example:

```
drizzle_server foo.bar.com:3306 user=monty password=some_pass
                                dbname=test protocol=mysql
                                charset=utf8;
```

Please note that for the mysql server, "utf-8" is not a valid encoding name while `utf8` is.

Back to TOC

drizzle_keepalive

syntax: *drizzle_keepalive max=<size> mode=<mode>*

default: *drizzle_keepalive max=0 mode=single*

context: *upstream*

Configures the keep-alive connection pool for MySQL/Drizzle connections.

The following options are supported:

max= `<num>` Specify the capacity of the connection pool for the current upstream block. The value *must* be non-zero. If set to `0` , it effectively disables the connection pool. This option is default to `0` .

mode= `<mode>` This supports two values, `single` and `multi` . The `single` mode means the pool does not distinguish various drizzle servers in the current upstream block while `multi` means the pool will merely reuse connections which have identical server host names and ports. Note that even under `multi`, differences between `dbname` or `user` parameters will be silently ignored. Default to `single`.

overflow= `<action>` This option specifies what to do when the connection pool is already full while new database connection is required. Either `reject` or `ignore` can be specified. In case of `reject`, it will reject the current request, and returns the `503 Service Unavailable` error page. For `ignore`, this module will go on creating a new database connection.

Back to TOC

drizzle_query

syntax: *drizzle_query <sql>*

default: *no*

context: *http, server, location, location if*

Specify the SQL queries sent to the Drizzle/MySQL backend.

Nginx variable interpolation is supported, but you must be careful with SQL injection attacks. You can use the set_quote_sql_str directive, for example, to quote values for SQL interpolation:

```
location /cat {
    set_unescape_uri $name $arg_name;
    set_quote_sql_str $quoted_name $name;

    drizzle_query "select * from cats where name = $quoted_name";
    drizzle_pass my_backend;
}
```

Back to TOC

drizzle_pass

syntax: *drizzle_pass <remote>*

default: *no*

context: *location, location if*

phase: *content*

This directive specifies the Drizzle or MySQL upstream name to be queried in the current location. The `<remote>` argument can be any upstream name defined with the drizzle_server directive.

Nginx variables can also be interpolated into the `<remote>` argument, so as to do dynamic backend routing, for example:

```
upstream moon { drizzle_server ...; }

server {
    location /cat {
        set $backend 'moon';

        drizzle_query ...;
        drizzle_pass $backend;
    }
}
```

Back to TOC

drizzle_connect_timeout

syntax: *drizzle_connect_time <time>*

default: *drizzle_connect_time 60s*

context: *http, server, location, location if*

Specify the (total) timeout for connecting to a remote Drizzle or MySQL server.

The `<time>` argument can be an integer, with an optional time unit, like `s` (second), `ms` (millisecond), `m` (minute). The default time unit is `s`, i.e., "second". The default setting is `60s`.

Back to TOC

drizzle_send_query_timeout

syntax: *drizzle_send_query_timeout <time>*

default: *drizzle_send_query_timeout 60s*

context: *http, server, location, location if*

Specify the (total) timeout for sending a SQL query to a remote Drizzle or MySQL server.

The `<time>` argument can be an integer, with an optional time unit, like `s` (second), `ms` (millisecond), `m` (minute). The default time unit is `s`, ie, "second". The default setting is `60s`.

Back to TOC

drizzle_recv_cols_timeout

syntax: *drizzle_recv_cols_timeout <time>*

default: *drizzle_recv_cols_timeout 60s*

context: *http, server, location, location if*

Specify the (total) timeout for receiving the columns metadata of the result-set to a remote Drizzle or MySQL server.

The `<time>` argument can be an integer, with an optional time unit, like `s` (second), `ms` (millisecond), `m` (minute). The default time unit is `s`, ie, "second". The default setting is `60s`.

Back to TOC

drizzle_recv_rows_timeout

syntax: *drizzle_recv_rows_timeout <time>*

default: *drizzle_recv_rows_timeout 60s*

context: *http, server, location, location if*

Specify the (total) timeout for receiving the rows data of the result-set (if any) to a remote Drizzle or MySQL server.

The `<time>` argument can be an integer, with an optional time unit, like `s` (second), `ms` (millisecond), `m` (minute). The default time unit is `s`, ie, "second". The default setting is `60s`.

Back to TOC

drizzle_buffer_size

syntax: *drizzle_buffer_size <size>*

default: *drizzle_buffer_size 4k/8k*

context: *http, server, location, location if*

Specify the buffer size for drizzle outputs. Default to the page size (4k/8k). The larger the buffer, the less streammy the outputing process will be.

Back to TOC

drizzle_module_header

syntax: *drizzle_module_header on|off*

default: *drizzle_module_header on*

context: *http, server, location, location if*

Controls whether to output the drizzle header in the response. Default on.

The drizzle module header looks like this:

```
X-Resty-DBD-Module: ngx_drizzle 0.1.0
```

Back to TOC

drizzle_status

syntax: *drizzle_status*

default: *no*

context: *location, location if*

phase: *content*

When specified, the current Nginx location will output a status report for all the drizzle upstream servers in the virtual server of the current Nginx worker process.

The output looks like this:

```
worker process: 15231

upstream backend
  active connections: 0
  connection pool capacity: 10
  overflow: reject
```

```
    cached connection queue: 0
    free'd connection queue: 10
    cached connection successfully used count:
    free'd connection successfully used count: 3 0 0 0 0 0 0 0 0 0
    servers: 1
    peers: 1

  upstream backend2
    active connections: 0
    connection pool capacity: 0
    servers: 1
    peers: 1
```

Note that, this is *not* the global statistics if you do have multiple Nginx worker processes configured in your `nginx.conf` .

Back to TOC

Variables

This module creates the following Nginx variables:

Back to TOC

$drizzle_thread_id

This variable will be assigned a textual number of the underlying MySQL or Drizzle query thread ID when the current SQL query times out. This thread ID can be further used in a SQL kill command to cancel the timed-out query.

Here's an example:

```
drizzle_connect_timeout 1s;
drizzle_send_query_timeout 2s;
drizzle_recv_cols_timeout 1s;
drizzle_recv_rows_timeout 1s;

location /query {
    drizzle_query 'select sleep(10)';
    drizzle_pass my_backend;
    rds_json on;

    more_set_headers -s 504 'X-Mysql-Tid: $drizzle_thread_id';
}

location /kill {
    drizzle_query "kill query $arg_tid";
    drizzle_pass my_backend;
    rds_json on;
}

location /main {
    content_by_lua '
        local res = ngx.location.capture("/query")
        if res.status ~= ngx.HTTP_OK then
            local tid = res.header["X-Mysql-Tid"]
            if tid and tid ~= "" then
                ngx.location.capture("/kill", { args = {tid = tid} })
            end
            return ngx.HTTP_INTERNAL_SERVER_ERROR;
        end
        ngx.print(res.body)
    '
}
```

where we make use of headers-more-nginx-module, lua-nginx-module, and rds-json-nginx-module too. When the SQL query timed out, we'll explicitly cancel it immediately. One pitfall here is that you have to add these modules in this order while building Nginx:

- lua-nginx-module
- headers-more-nginx-module
- rds-json-nginx-module

Such that, their output filters will work in the *reversed* order, i.e., first convert RDS to JSON, and then add our `X-Mysql-Tid` custom header, and finally capture the whole (subrequest) response with the Lua module. You're recommended to use the OpenResty bundle though, it ensures the module building order automatically for you.

Back to TOC

Output Format

This module generates binary query results in a format that is shared among the various Nginx database driver modules like ngx_postgres. This data format is named `Resty DBD Stream` (RDS).

If you're a web app developer, you may be more interested in

- using rds-json-nginx-module to obtain JSON output,
- using rds-csv-nginx-module to obain Comma-Separated-Value (CSV) output,
- or using lua-rds-parser to parse the RDS data into Lua data structures.

For the HTTP response header part, the `200 OK` status code should always be returned. The `Content-Type` header *must* be set to `application/x-resty-dbd-stream` . And the driver generating this response also sets a `X-Resty-DBD` header. For instance, this module adds the following output header:

```
X-Resty-DBD-Module: drizzle 0.1.0
```

where `0.1.0` is this module's own version number. This `X-Resty-DBD-Module` header is optional though.

Below is the HTTP response body format (version 0.0.3):

Back to TOC

RDS Header Part

The RDS Header Part consists of the following fields:

uint8_t endian type (1 means big-endian and little endian otherwise)

uint32_t format version (v1.2.3 is represented as 1002003 in decimal)

uint8_t result type (0 means normal SQL result type, fixed for now)

uint16_t standard error code

uint16_t driver-specific error code

uint16_t driver-specific error string length

u_char * driver-specific error string data

uint64_t database rows affected

uint64_t insert id (if none, 0)

uint16_t column count

Back to TOC

RDS Body Part

When the `column count` field in the RDS Header Part is zero, then the whole RDS Body Part is omitted.

The RDS Body Part consists of two sections, Columns and Rows.

Columns

The columns part consists of zero or more column data. The number of columns is determined by `column count` field in RDS Header Part.

Each column consists of the following fields

uint16_t non-zero value for standard column type code and for the column list terminator and zero otherwise.

uint16_t driver-specific column type code

uint16_t column name length

u_char * column name data

Rows

The rows part consists of zero or more row data, terminated by a 8-bit zero.

Each row data consists of a Row Flag and an optional Fields Data part.

Row Flag

uint8_t valid row (1 means valid, and 0 means the row list terminator)

Fields Data

The Fields Data consists zero or more fields of data. The field count is predetermined by the `column number</code) specified in RDS Header Part.`

uint32_t field length ((uint32_t) -1 represents NULL)

u_char * field data in textual representation), is empty (0) if field length == (uint32_t) -1

Back to TOC

RDS buffer Limitations

On the nginx output chain link level, the following components should be put into a single `ngx_buf_t` struct:

- the header
- each column and the column list terminator
- each row's valid flag byte and row list terminator
- each field in each row (if any) but the field data can span multiple bufs.

Back to TOC

Status Code

If the MySQL error code in MySQL's query result is not OK, then a 500 error page is returned by this module, except for the table non-existent error, which results in the `410 Gone` error page.

Back to TOC

Caveats

- Other usptream modules like `upstream_hash` and HttpUpstreamKeepaliveModule *must not* be used with this module in a single upstream block.
- Directives like server *must not* be mixed with drizzle_server either.
- Upstream backends that don't use drizzle_server to define server entries *must not* be used in the drizzle_pass directive.

Back to TOC

Trouble Shooting

- When you see the following error message in `error.log`:

  ```
  failed to connect: 15: drizzle_state_handshake_result_read:
    old insecure authentication mechanism not supported in upstream, ...
  ```

  then you may checkout if your MySQL is too old (at least 5.x is required) or your mysql config file explicitly forces the use of old authentication method (you should remove the `old-passwords` line from your `my.cnf` and add the line `secure_auth 1`).

- When you see the following error message in `error.log`:

  ```
  failed to connect: 23: Access denied for user 'root'@'ubuntu'
    (using password: YES) while connecting to drizzle upstream, ...
  ```

  You should check if your MySQL account does have got TCP login access on your MySQL server side. A quick check is to use MySQL's official client to connect to your server:

  ```
  mysql --protocol=tcp -u user --password=password -h foo.bar.com dbname
  ```

```
Note that the `--protocol=tcp` option is required here, or your MySQL client may use Un
```

Back to TOC

Known Issues

- Calling mysql procedures are currently not supported because the underlying libdrizzle library does not support the `CLIENT_MULTI_RESULTS` flag yet :(But we'll surely work on it.
- Multiple SQL statements in a single query are not supported due to the lack of `CLIENT_MULTI_STATEMENTS` support in the underlying libdrizzle library.
- This module does not (yet) work with the `RTSIG` event model.

Back to TOC

Installation

You're recommended to install this module as well as rds-json-nginx-module via the ngx_openresty bundle:

http://openresty.org

The installation steps are usually as simple as `./configure --with-http_drizzle_module && make && make install` (But you still need to install the libdrizzle library manually, see http://openresty.org/#DrizzleNginxModule for detailed instructions.

Alternatively, you can compile this module with Nginx core's source by hand:

- You should first install libdrizzle 1.0 which is now distributed with the drizzle project and can be obtained from https://launchpad.net/drizzle. The latest drizzle7 release does not support building libdrizzle 1.0 separately and requires a lot of external dependencies like Boost and Protobuf which are painful to install. The last version supporting building libdrizzle 1.0 separately is `2011.07.21`. You can download it from http://agentzh.org/misc/nginx/drizzle7-2011.07.21.tar.gz . Which this version of drizzle7, installation of libdrizzle 1.0 is usually as simple as

```
tar xzvf drizzle7-2011.07.21.tar.gz
cd drizzle7-2011.07.21/
./configure --without-server
make libdrizzle-1.0
make install-libdrizzle-1.0
```

```
Ensure that you have the `python` command point to a `python2` interpreter. It's known
```

```
File "config/pandora-plugin", line 185
    print "Dependency loop detected with %s" % plugin['name']
                                             ^
SyntaxError: invalid syntax
make: *** [.plugin.scan] Error 1
```

```
You can fix this by pointing `python` to `python2`.
```

- Download the latest version of the release tarball of this module from drizzle-nginx-module file list.
- Grab the nginx source code from nginx.org, for example, the version 1.7.4 (see nginx compatibility), and then build the source with this module:

```
wget 'http://nginx.org/download/nginx-1.7.4.tar.gz'
tar -xzvf nginx-1.7.4.tar.gz
cd nginx-1.7.4/

# if you have installed libdrizzle to the prefix /opt/drizzle, then
# specify the following environments:
# export LIBDRIZZLE_INC=/opt/drizzle/include/libdrizzle-1.0
# export LIBDRIZZLE_LIB=/opt/drizzle/lib

# Here we assume you would install you nginx under /opt/nginx/.
./configure --prefix=/opt/nginx \
            --add-module=/path/to/drizzle-nginx-module

make -j2
make install
```

You usually also need rds-json-nginx-module to obtain JSON output from the binary RDS output generated by this upstream module.

Back to TOC

Compatibility

If you're using MySQL, then MySQL `5.0 ~ 5.5` is required. We're not sure if MySQL `5.6+` work; reports welcome!

This module has been tested on Linux and Mac OS X. Reports on other POSIX-compliant systems will be highly appreciated.

The following versions of Nginx should work with this module:

- 1.7.x (last tested: 1.7.4)
- 1.6.x
- 1.5.x (last tested: 1.5.8)
- 1.4.x (last tested: 1.4.4)
- 1.3.x (last tested: 1.3.7)
- 1.2.x (last tested: 1.2.9)
- 1.1.x (last tested: 1.1.5)
- 1.0.x (last tested: 1.0.8)
- 0.8.x (last tested: 0.8.55)
- 0.7.x >= 0.7.44 (last tested version is 0.7.67)

Earlier versions of Nginx like `0.6.x` and `0.5.x` will *not* work.

If you find that any particular version of Nginx above `0.7.44` does not work with this module, please consider reporting a bug.

Back to TOC

Community

Back to TOC

English Mailing List

The openresty-en mailing list is for English speakers.

Back to TOC

Chinese Mailing List

The openresty mailing list is for Chinese speakers.

Back to TOC

Report Bugs

Please submit bug reports, wishlists, or patches by

1. creating a ticket on the issue tracking interface provided by GitHub,
2. or sending an email to the OpenResty community.

Back to TOC

Source Repository

Available on github at openresty/drizzle-nginx-module.

Back to TOC

Test Suite

This module comes with a Perl-driven test suite. The test cases are declarative too. Thanks to the Test::Nginx module in the Perl world.

To run it on your side:

```
$ PATH=/path/to/your/nginx-with-echo-module:$PATH prove -r t
```

Because a single nginx server (by default, `localhost:1984`) is used across all the test scripts (`.t` files), it's meaningless to run the test suite in parallel by specifying `-jN` when invoking the `prove` utility.

Back to TOC

TODO

- add the MySQL transaction support.

- add multi-statement MySQL query support.
- implement the "drizzle_max_output_size" directive. When the RDS data is larger then the size specified, the module will try to terminate the output as quickly as possible but will still ensure the resulting response body is still in valid RDS format.
- implement the `drizzle_upstream_next` mechanism for failover support.
- add support for multiple "drizzle_query" directives in a single location.
- implement *weighted* round-robin algorithm for the upstream server clusters.
- add the `max_idle_time` option to the drizzle_server directive, so that the connection pool will automatically release idle connections for the timeout you specify.
- add the `min` option to the "drizzle_server" directive so that the connection pool will automatically create that number of connections and put them into the pool.
- add Unix domain socket support in the `drizzle_server` directive.
- make the drizzle_query directive reject variables that have not been processed by a drizzle_process directive. This will pretect us from SQL injections. There will also be an option ("strict=no") to disable such checks.

Back to TOC

Changes

The changes of every release of this module can be obtained from the ngx_openresty bundle's change logs:

http://openresty.org/#Changes

Back to TOC

Authors

- chaoslawful (王晓哲)
- Yichun "agentzh" Zhang (章亦春) , CloudFlare Inc.
- Piotr Sikora , CloudFlare Inc.

Back to TOC

Copyright & License

Back to TOC

See Also

- rds-json-nginx-module
- rds-csv-nginx-module
- lua-rds-parser
- The ngx_openresty bundle
- DrizzleNginxModule bundled by ngx_openresty
- postgres-nginx-module
- lua-nginx-module
- The lua-resty-mysql library based on the lua-nginx-module cosocket API.