openresty / lua-upstream-nginx-module

Nginx C module to expose Lua API to ngx_lua for Nginx upstreams

22 commits 1 branch 2 releases 2 contributors

branch: master lua-upstream-nginx-module / +

Merge branch 'master' of github.com:agentzh/lua-upstream-nginx-module

agentzh authored on Oct 22, 2014 latest commit cecdb5f2a8

src	bugfix: upstream names did not support taking a port number. thanks m…	10 months ago
t	tests: sina.com is a domain we have no control which may resolve to j…	4 months ago
util	initial checkin: methods get_upstreams() and get_servers() now work.	a year ago
.gitignore	initial checkin: methods get_upstreams() and get_servers() now work.	a year ago
README.md	Modified README.md: fixed get_backup_peers example	6 months ago
config	initial checkin: methods get_upstreams() and get_servers() now work.	a year ago
valgrind.suppress	suppressed a valgrind false positive in libdl.	11 months ago

README.md

Name

ngx_lua_upstream - Nginx C module to expose Lua API to ngx_lua for Nginx upstreams

Table of Contents

Status

This module is still under active development and is considered production ready.

Synopsis

```
http {
    upstream foo.com {
```

```
        server 127.0.0.1 fail_timeout=53 weight=4 max_fails=100;
        server agentzh.org:81;
    }

    upstream bar {
        server 127.0.0.2;
    }

    server {
        listen 8080;

        # sample output for the following /upstream interface:
        # upstream foo.com:
        #     addr = 127.0.0.1:80, weight = 4, fail_timeout = 53, max_fails = 100
        #     addr = 106.187.41.147:81, weight = 1, fail_timeout = 10, max_fails = 1
        # upstream bar:
        #     addr = 127.0.0.2:80, weight = 1, fail_timeout = 10, max_fails = 1

        location = /upstreams {
            default_type text/plain;
            content_by_lua '
                local concat = table.concat
                local upstream = require "ngx.upstream"
                local get_servers = upstream.get_servers
                local get_upstreams = upstream.get_upstreams

                local us = get_upstreams()
                for _, u in ipairs(us) do
                    ngx.say("upstream ", u, ":")
                    local srvs, err = get_servers(u)
                    if not srvs then
                        ngx.say("failed to get servers in upstream ", u)
                    else
                        for _, srv in ipairs(srvs) do
                            local first = true
                            for k, v in pairs(srv) do
                                if first then
                                    first = false
```

```
                                ngx.print("    ")
                            else
                                ngx.print(", ")
                            end
                            if type(v) == "table" then
                                ngx.print(k, " = {", concat(v, ", "), "}")
                            else
                                ngx.print(k, " = ", v)
                            end
                        end
                        ngx.print("\\n")
                    end
                end
            end
        ';
    }
}
```

Functions

Back to TOC

get_upstreams

`syntax: names = upstream.get_upstreams()`

Get a list of the names for all the named upstream groups (i.e., explicit `upstream {}` blocks).

Note that implicit upstream groups created by `proxy_pass` and etc are excluded.

Back to TOC

get_servers

```
syntax: servers = upstream.get_servers(upstream_name)
```

Get configurations for all the servers in the specified upstream group. Please one server may take multiple addresses when its server name can be resolved to multiple addresses.

The return value is an array-like Lua table. Each table entry is a hash-like Lua table that takes the following keys:

- addr

  socket address(es). can be either a Lua string or an array-like Lua table of Lua strings.

- backup
- fail_timeout
- max_fails
- weight

Back to TOC

get_primary_peers

```
syntax: peers = upstream.get_primary_peers(upstream_name)
```

Get configurations for all the primary (non-backup) peers in the specified upstream group.

The return value is an array-like Lua table for all the primary peers. Each table entry is a (nested) hash-like Lua table that takes the following keys:

- current_weight
- effective_weight

- fail_timeout
- fails
- id

  Identifier (ID) for the peer. This ID can be used to reference a peer in a group in the peer modifying API.

- max_fails
- name

  Socket address for the current peer

- weight
- accessed

  Timestamp for the last access (in seconds since the Epoch)

- checked

  Timestamp for the last check (in seconds since the Epoch)

Back to TOC

get_backup_peers

```
syntax: peers = upstream.get_backup_peers(upstream_name)
```

Get configurations for all the backup peers in the specified upstream group.

The return value has the same structure as get_primary_peers function.

Back to TOC

set_peer_down

```
syntax: ok, err = upstream.set_peer_down(upstream_name, is_backup, peer_id, down_value)
```

Set the "down" (boolean) attribute of the specified peer.

To uniquely specify a peer, you need to specify the upstream name, whether or not it is a backup peer, and the peer id (starting from 0).

Note that this method only changes the peer settings in the current Nginx worker process. You need to synchronize the changes across all the Nginx workers yourself if you want a server-wide change (for example, by means of ngx_lua's ngx.shared.DICT).

Below is an example. Consider we have a "bar" upstream block in `nginx.conf`:

```
upstream bar {
    server 127.0.0.2;
    server 127.0.0.3 backup;
    server 127.0.0.4 fail_timeout=23 weight=7 max_fails=200 backup;
}
```

then

```
upstream.set_peer_down("bar", false, 0, true)
```

will turn down the primary peer corresponding to `server 127.0.0.2`.

Similarly,

```
upstream.set_peer_down("bar", true, 1, true)
```

will turn down the backup peer corresponding to `server 127.0.0.4 ...`.

You can turn on a peer again by providing a `false` value as the 4th argument.

Back to TOC

TODO

- Add API to add or remove servers to existing upstream groups.

Back to TOC

Compatibility

The following versions of Nginx should work with this module:

- **1.5.x** (last tested: 1.5.12)

Back to TOC

Installation

This module is bundled and enabled by default in the OpenResty bundle. And you are recommended to use OpenResty.

1. Grab the nginx source code from nginx.org, for example, the version 1.5.12 (see nginx

compatibility),

2. then grab the source code of the ngx_lua as well as its dependencies like LuaJIT.
3. and finally build the source with this module:

```
wget 'http://nginx.org/download/nginx-1.5.12.tar.gz'
tar -xzvf nginx-1.5.12.tar.gz
cd nginx-1.5.12/

# assuming your luajit is installed to /opt/luajit:
export LUAJIT_LIB=/opt/luajit/lib

# assuming you are using LuaJIT v2.1:
export LUAJIT_INC=/opt/luajit/include/luajit-2.1

# Here we assume you would install you nginx under /opt/nginx/.
./configure --prefix=/opt/nginx \
    --with-ld-opt="-Wl,-rpath,$LUAJIT_LIB" \
    --add-module=/path/to/lua-nginx-module \
    --add-module=/path/to/lua-upstream-nginx-module

make -j2
make install
```

If you are using ngx_openresty, then you can just add this module to OpenResty like this:

```
./configure --add-module=/path/to/lua-upstream-nginx-module
make -j2
make install
```

And you are all set. This module will get bundled into OpenResty in the near future.

Back to TOC

Author

Yichun "agentzh" Zhang (章亦春) agentzh@gmail.com, CloudFlare Inc.

Back to TOC

Copyright and License

SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Back to TOC

See Also

- the ngx_lua module: http://github.com/openresty/lua-nginx-module#readme
- the lua-resty-upstream-healthcheck library which makes use of the Lua API provided by this module.

Back to TOC