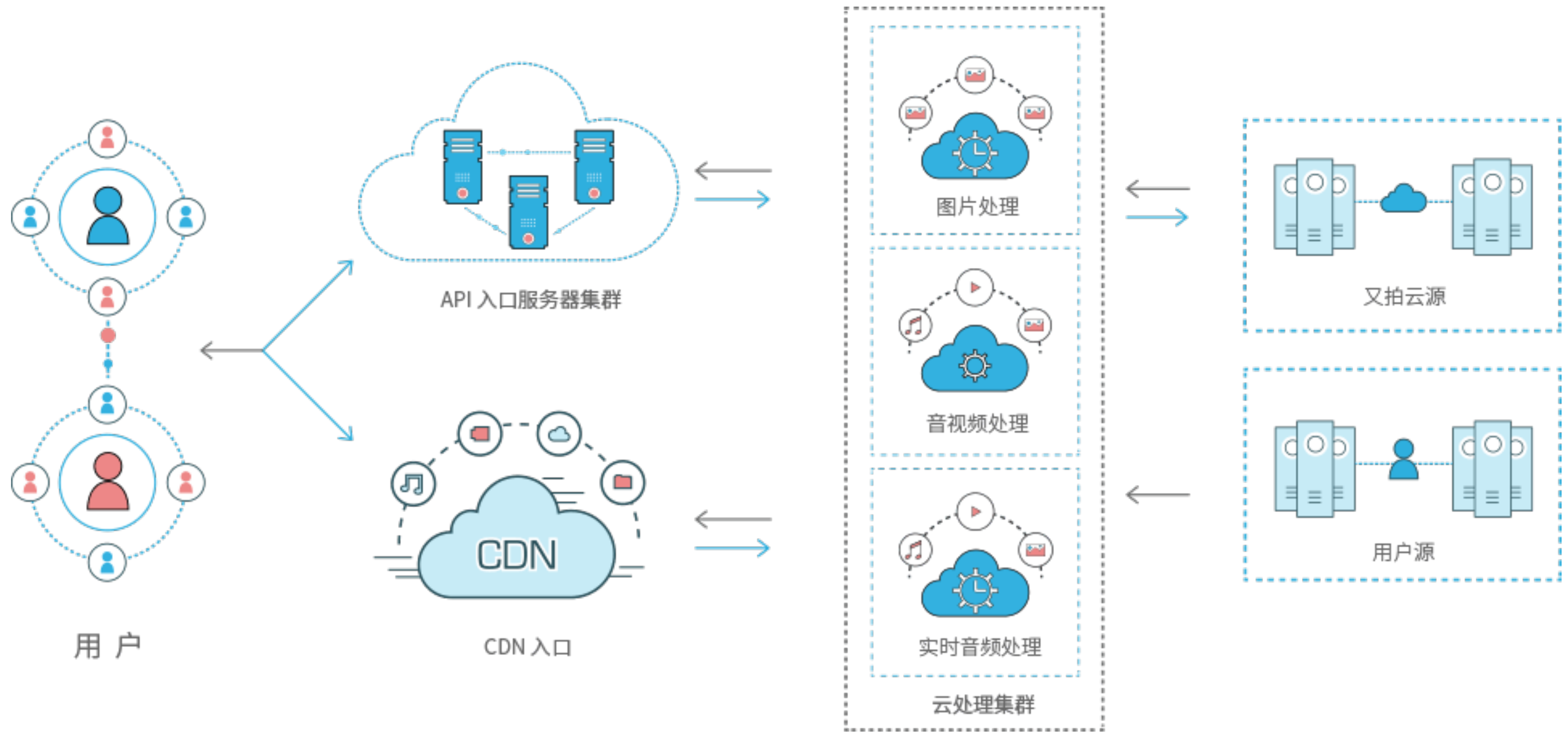
OpenResty在云处理服务集群中的应用

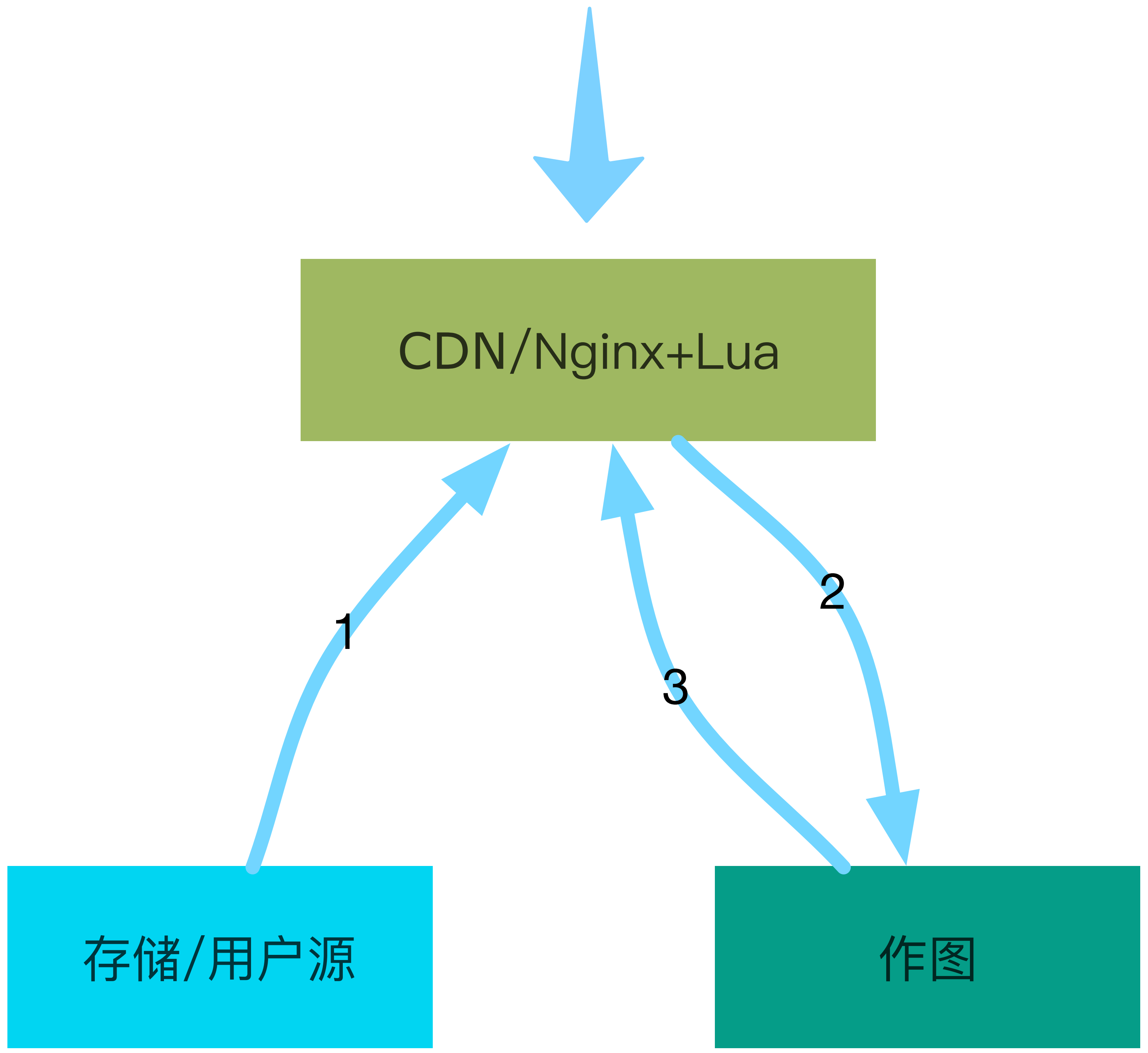
叶靖 (yejingx) @ 又拍云 杭州

2016.12.10 OpenResty Con

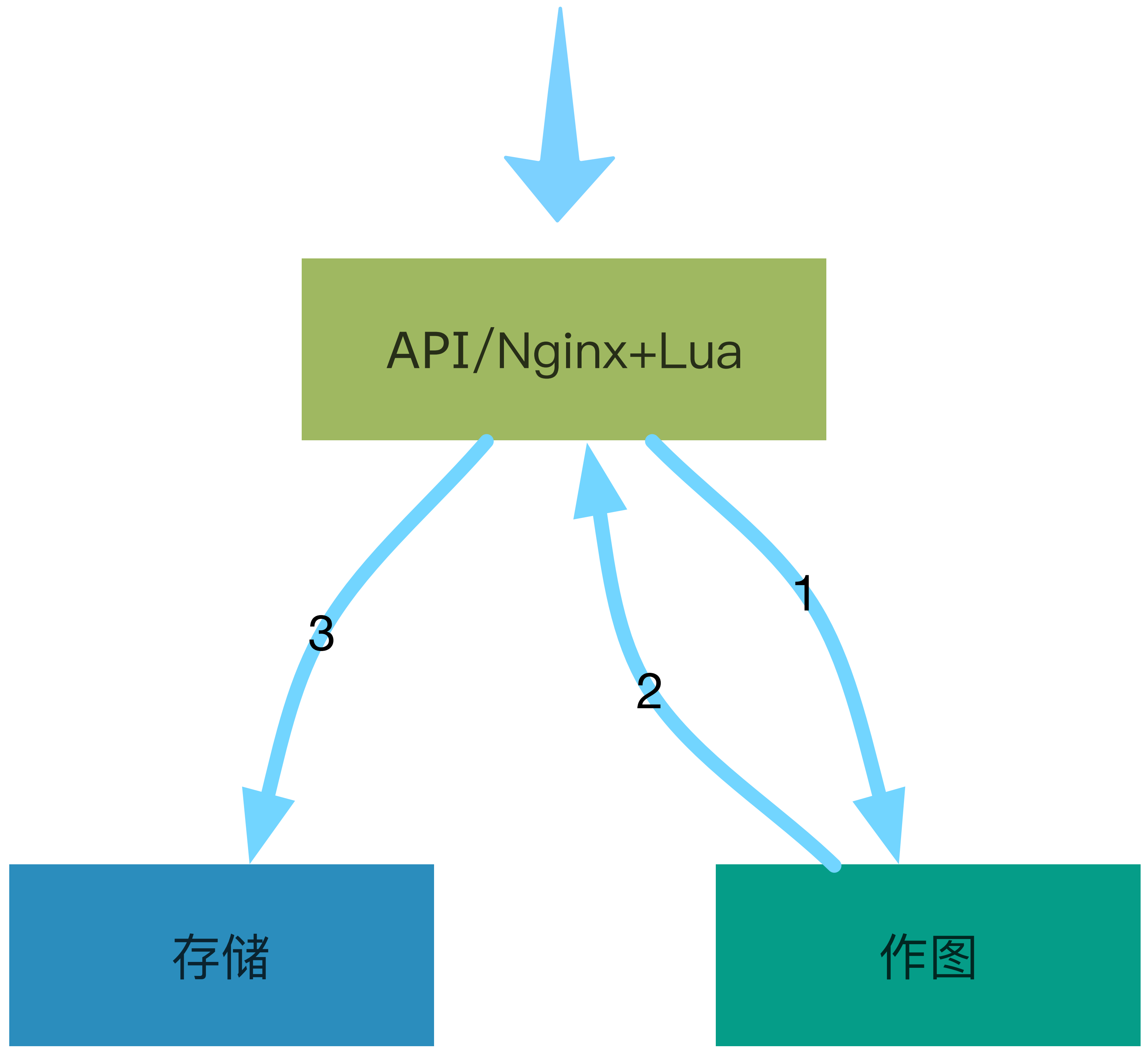
云处理服务



处理请求的数据流



http://yejingx.b0.upaiyun.com/cat.jpg!**/fw/200**



```
curl -T cat.jpg http://v0.api.upyun.com/yejingx/cat.jpg \
      -H "X-Gmkerl-Thumb: /fw/200"
```

处理请求的特点

- **多个upstream**
  - proxy_pass 不好用了
  - 需要 cosocket 接管 Nginx 数据流
- **要流式，也要缓冲**
  - 不能读全部 body 到内存
  - ngx.req.init_body / ngx.req.append_body / ngx.req.finish_body
- **失败重试**
  - 需要在 lua 代码里管理 upstream

接管数据流

lua-resty-httpipe

流式连接多个upstream

```
local r0, err = hp:request("127.0.0.1", 8080, {
                                     method = “GET",
                                     path = "/image",
                                     stream = true })

local r1, err = r0.pipe:request(“127.0.0.1", 9090, {
                                     method = “POST",
                                     path = "/imgprocess" })


ngx.status = r1.status
ngx.print(r1.body)
```

流式的缓冲

利用闭包对 ngx.req.socket() 进行封装

```
local req_reader = httpipe:get_client_body_reader()

ngx.req.init_body()

local downstream_reader = function()
  local chunk = req_reader(8192)
  if chunk then
    ngx.req.append_body(chunk)
  end
  return chunk
end

httpipe:send_request{ body=downstream_reader, …}
ngx.req.finish_body()

httpipe:read_response{…}
```

重试和 upstream 管理

lua-resty-checkups v0.1.0

被动健康检查

max_fails / fail_timeout
fail_timeout秒内失败max_fails次则把该上游标记为
fail_timeout 秒内不可用

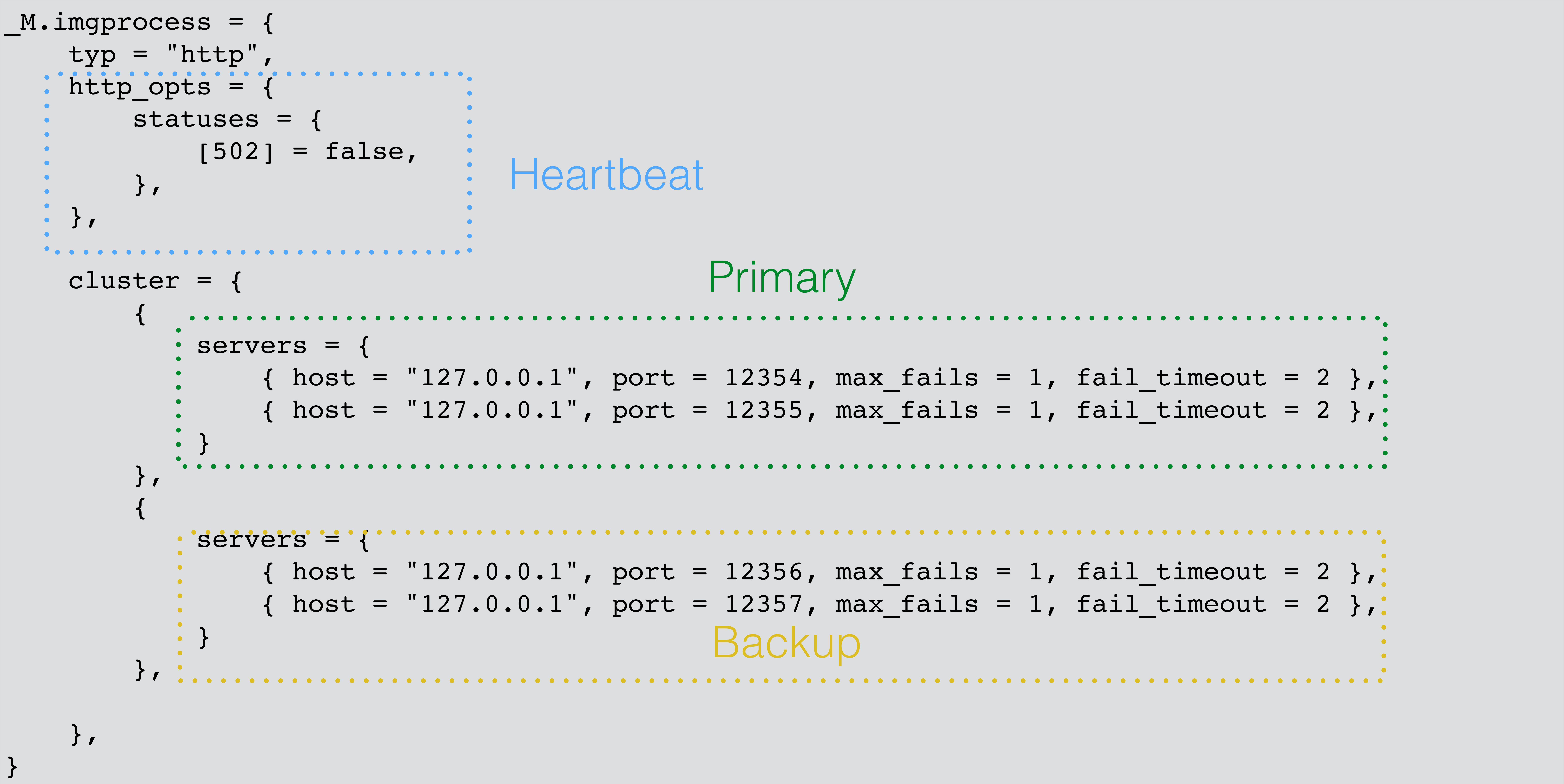
主动健康检查

heartbeat
定时给上游发送心跳包检测服务是否存活支持
tcp, http, mysql, redis

负载均衡算法

wrr / 一致性哈希 / 主备 / 多数据中心

checkups 配置



```
_M.imgprocess = {
    typ = "http",
    http_opts = {
        statuses = {
            [502] = false,
        },
    },
    cluster = {
        {
            servers = {
                { host = "127.0.0.1", port = 12354, max_fails = 1, fail_timeout = 2 },
                { host = "127.0.0.1", port = 12355, max_fails = 1, fail_timeout = 2 },
            }
        },
        {
            servers = {
                { host = "127.0.0.1", port = 12356, max_fails = 1, fail_timeout = 2 },
                { host = "127.0.0.1", port = 12357, max_fails = 1, fail_timeout = 2 },
            }
        },

    },
}
```

Heartbeat

Primary

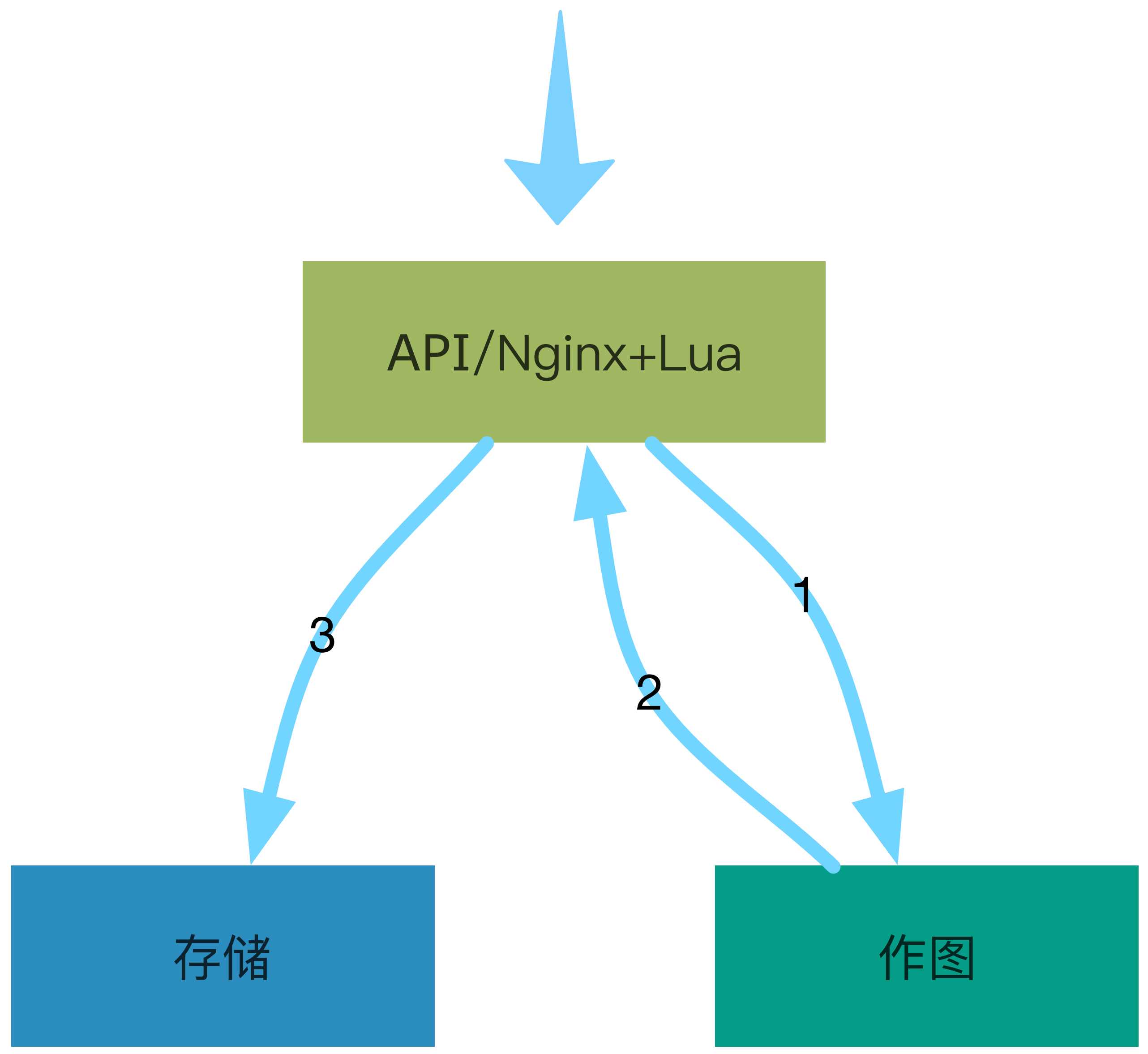
Backup

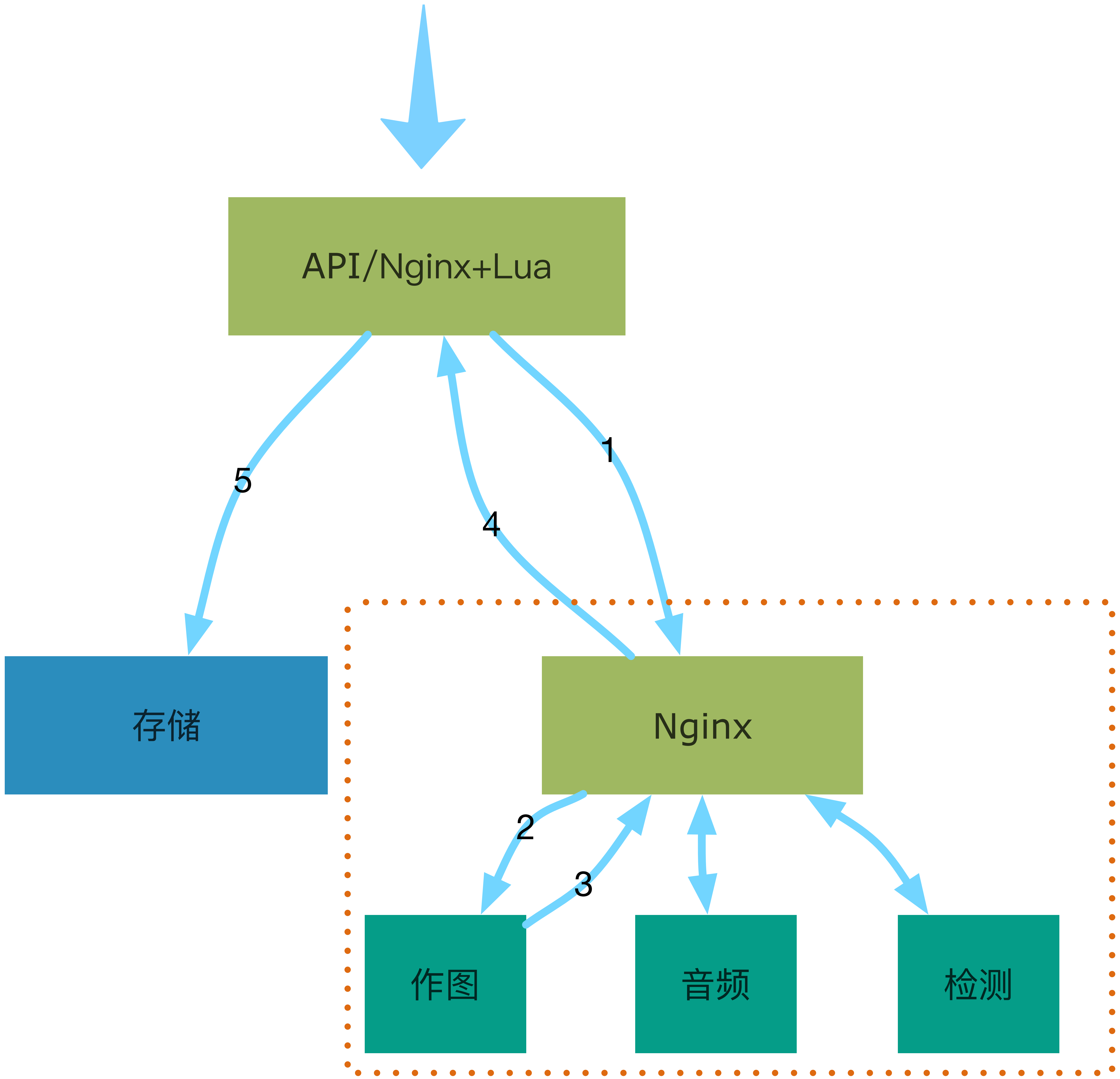
checkups 选择 upstream

```
syntax: res, err = ready_ok(key, callback, opts?)
```

```
local ok, err = checkups.ready_ok("imgprocess", function(host, port)
  httpipe:request(host, port, {method="GET", path="/echo"})
end)
```

处理类型增加





增加 Nginx 作为处理服务代理

集群规模增加

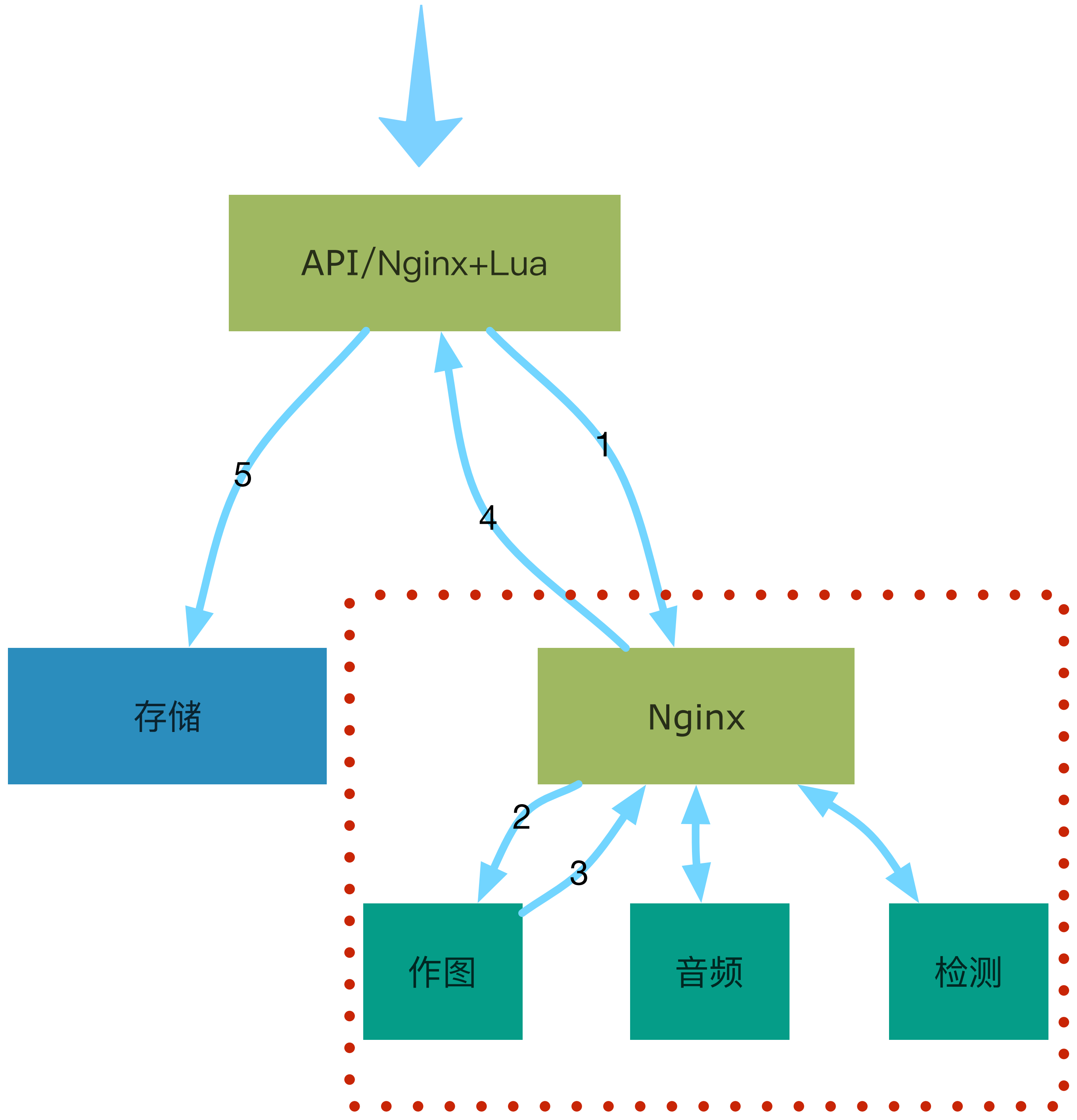
- **更新**
  - 运维脚本切流量
  - 修改 upstream.conf
  - reload
  - 更新一次几个小时
- **扩容**
  - 无 / 依赖太复杂
- **异常机器摘除**
  - 更新
  - 响应慢

如何走出困境?

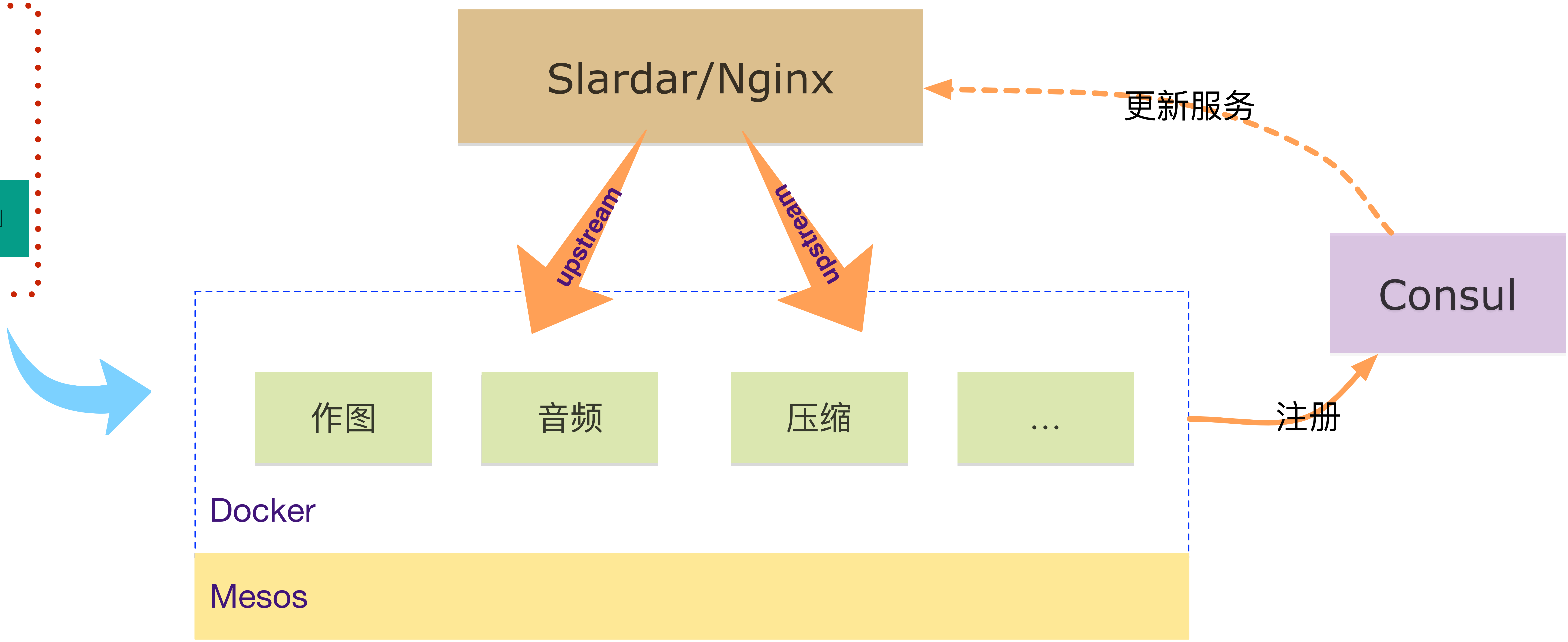
?

容器化改造

改造之后



终于可以快速扩容了 :)



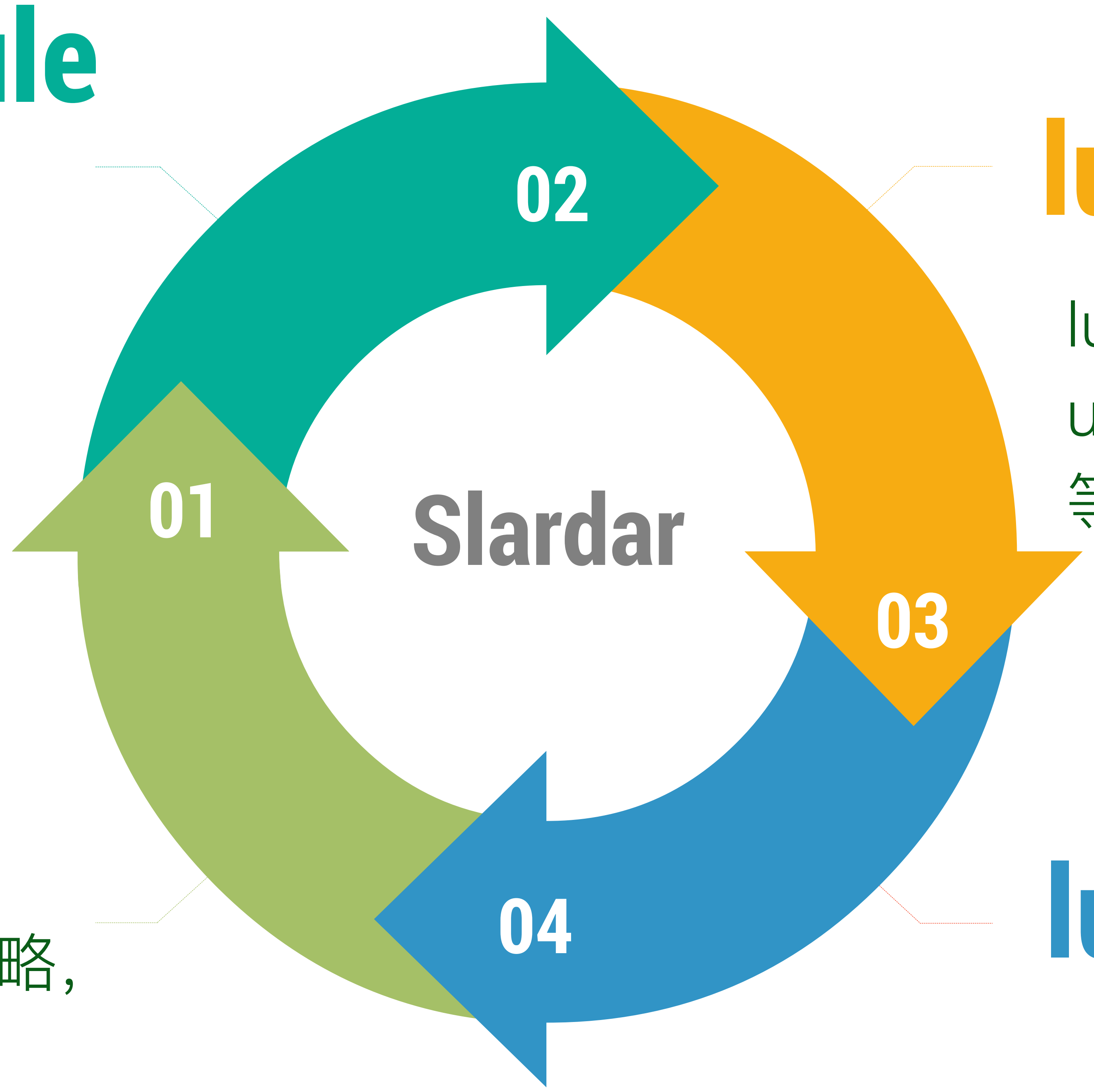
还有一个问题

Consul 里的服务如何更新到 Nginx ?

常见的方案

- **consul-template / etcd + confd**
  - 监听 Consul 中的变化
  - 触发重新生成 upstream.conf
  - Reload Nginx
- **ngx_http_dyups_module**
  - C 实现
  - 能过 HTTP 接口查询、增加、删除 upstream
  - 纯 lua 方案无法使用 / 无法与 checkups 结合
  - 开发效率比不上 lua

我们的轮子 - Slardar



lua-nginx-module

在Nginx 里用lua写逻辑，
balance_by_lua_*对
upstream选择进行hook

lua-resty-checkups

lua版的upstream，实现了动态
upstream管理、主被动健康检查
等功能

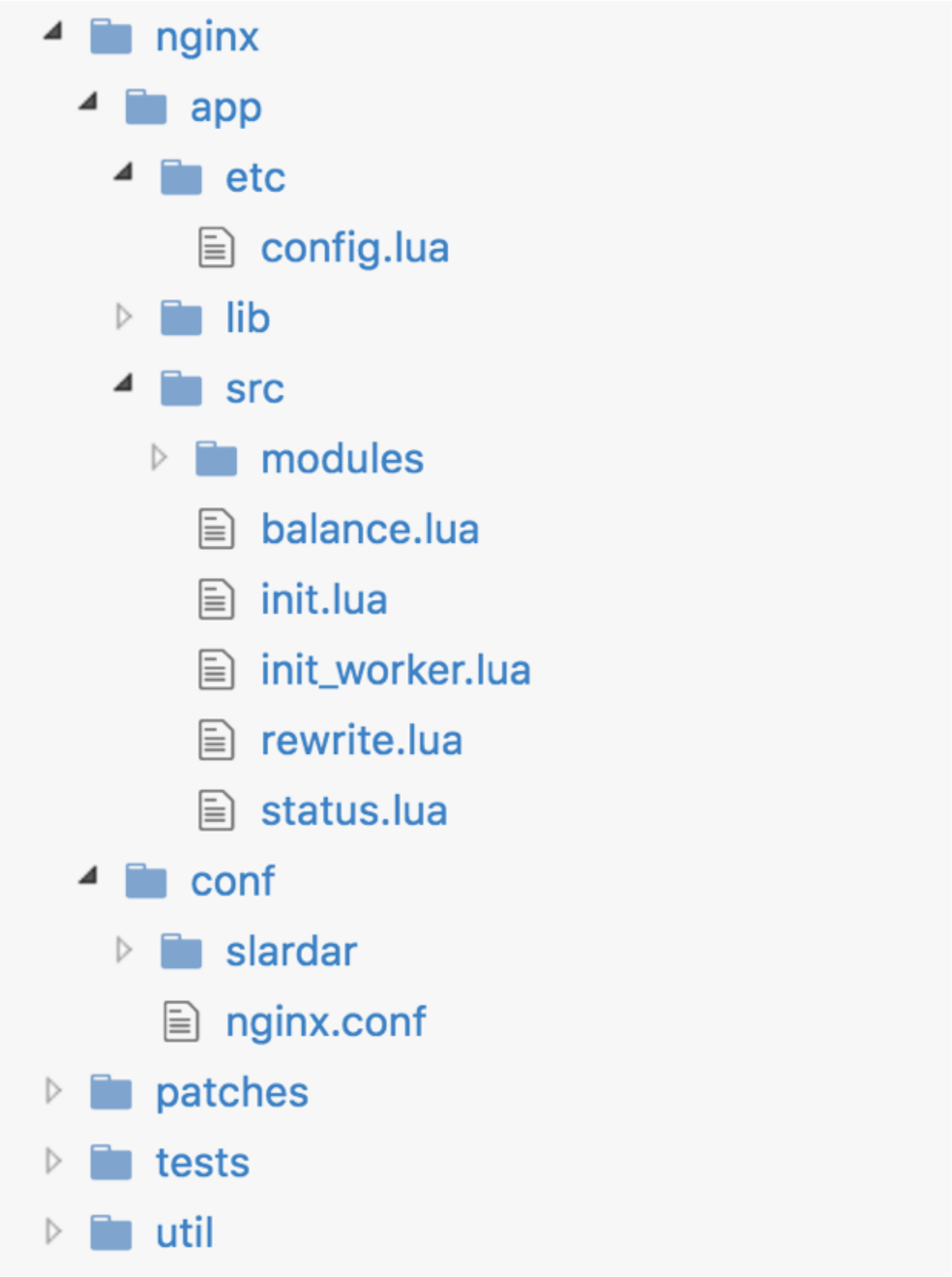
Nginx

proxy_* 指令和upstream重试策略，
如proxy_next_upstream_retry
和proxy_next_upstream_timeout

luasocket

lua的阻塞网络库
用于在启动时从consul拉upstream列表

又拍云 ngx_lua 项目组织



- nginx
  - app
    - etc
      - config.lua
    - lib
    - src
      - modules
      - balance.lua
      - init.lua
      - init_worker.lua
      - rewrite.lua
      - status.lua
  - conf
    - slardar
    - nginx.conf
- patches
- tests
- util

https://github.com/upyun/slardar

lua-resty-checkups v0.2.0

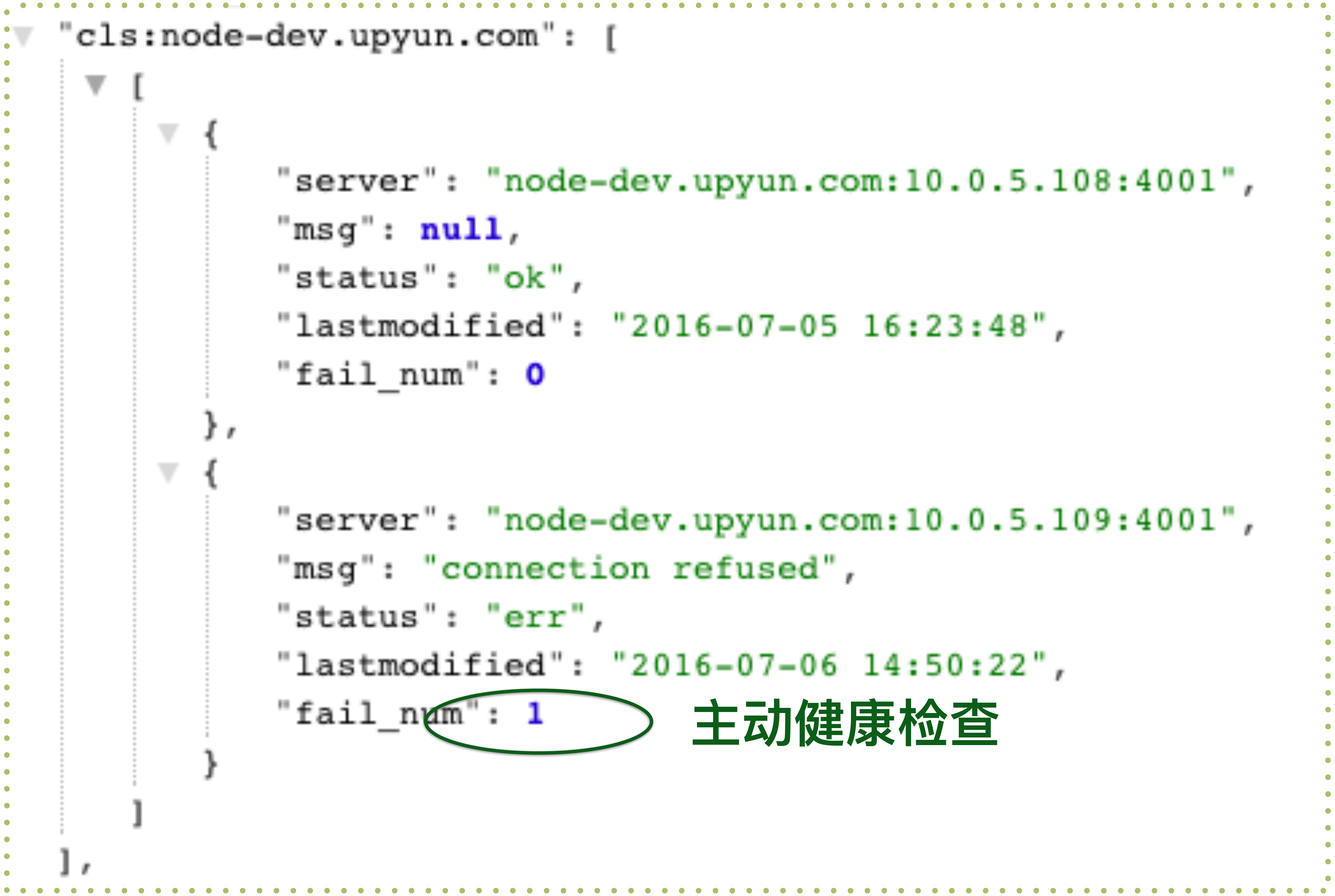
动态upstream管理 ············ update_upstream / delete_upstream
基于共享内存实现worker间同步

通过 HTTP 接口动态更新 upstream 列表:

```
curl -d \
  ’{“servers”:[ \
   {"host":"10.0.5.108", "port": 4001}, \       —> node docker 1
   {“host":"10.0.5.109", "port": 4001}], \      —> node docker 2
   ”keepalive": 20}’\
  127.0.0.1:1995/upstream/node-dev.upyun.com
```

upstream状态

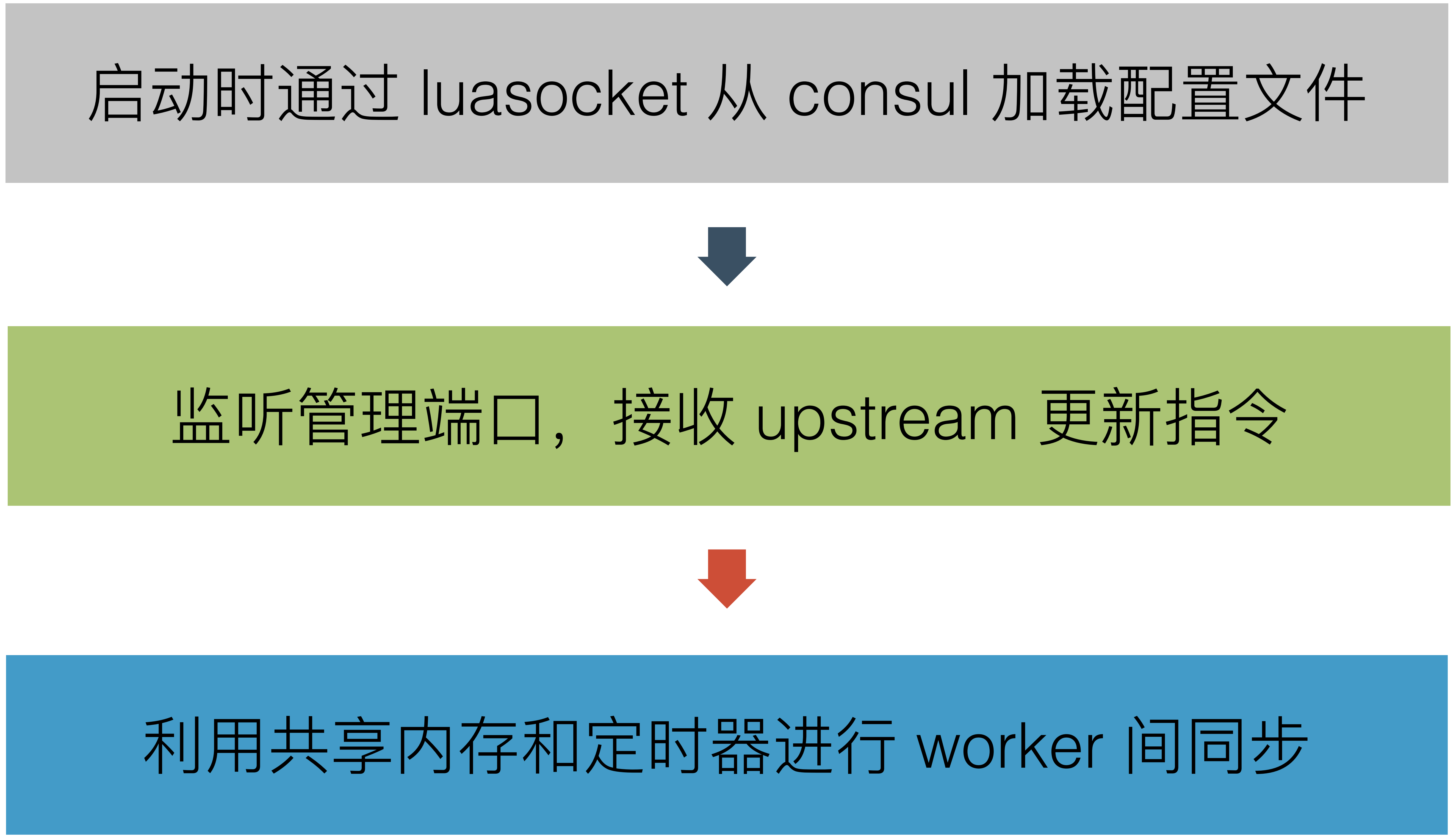
http://127.0.0.1:1995/status



```
▼ "cls:node-dev.upyun.com": [
    ▼ [
        ▼ {
              "server": "node-dev.upyun.com:10.0.5.108:4001",
              "msg": null,
              "status": "ok",
              "lastmodified": "2016-07-05 16:23:48",
              "fail_num": 0
          },
        ▼ {
              "server": "node-dev.upyun.com:10.0.5.109:4001",
              "msg": "connection refused",
              "status": "err",
              "lastmodified": "2016-07-06 14:50:22",
              "fail_num": 1
          }
      ]
  ],
```

主动健康检查

Slardar - 动态 upstream 管理

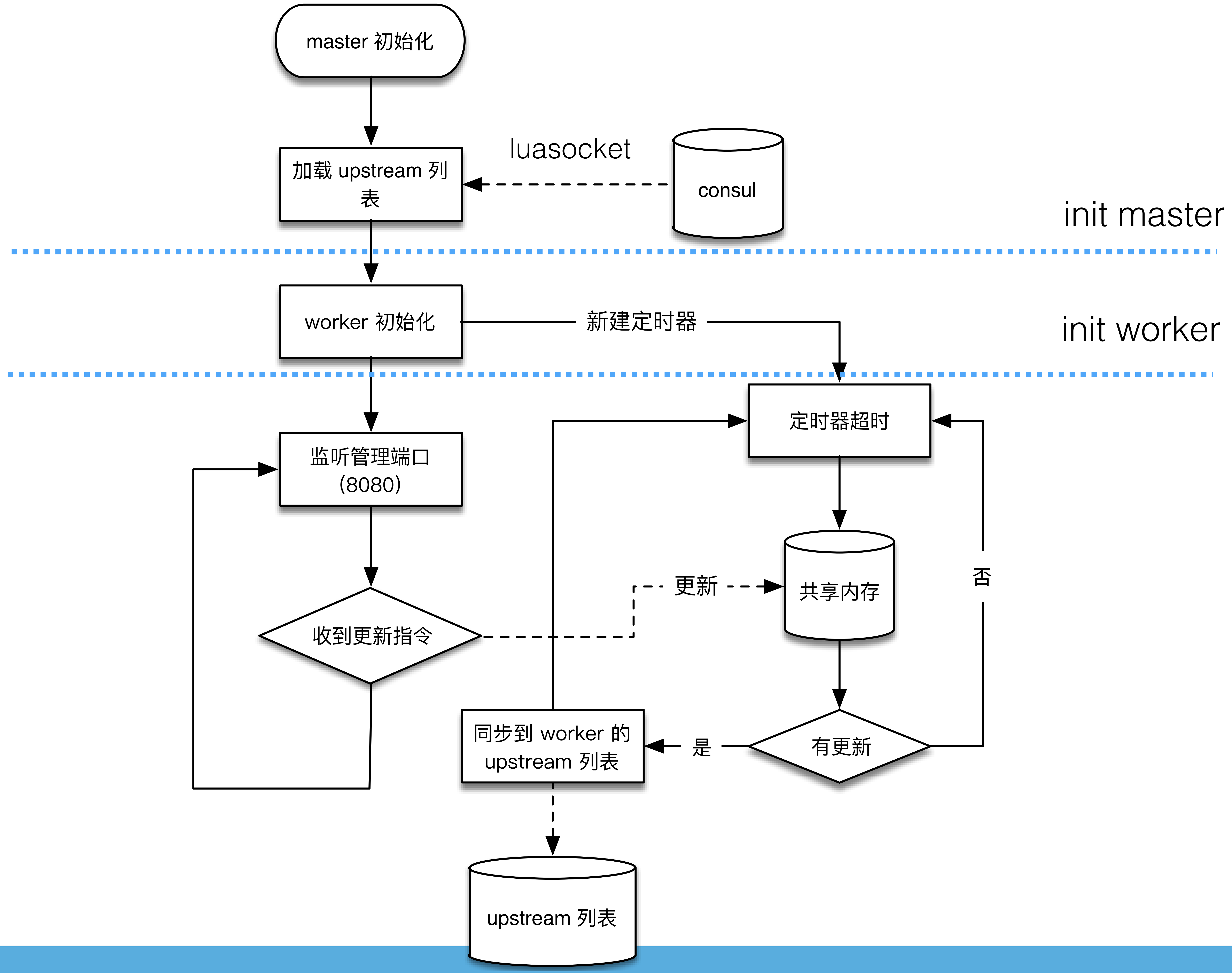


启动时通过 luasocket 从 consul 加载配置文件

监听管理端口，接收 upstream 更新指令

利用共享内存和定时器进行 worker 间同步

Slardar - 动态 upstream 管理



Slardar - 兼容 proxy_pass

balance_by_lua_*

upstream.conf:

```
upstream common {
    server 0.0.0.1;
    balancer_by_lua_file app/src/slardar_balance.lua;
}
```

Slardar - 兼容 proxy_pass

app/src/slardar_balance.lua:

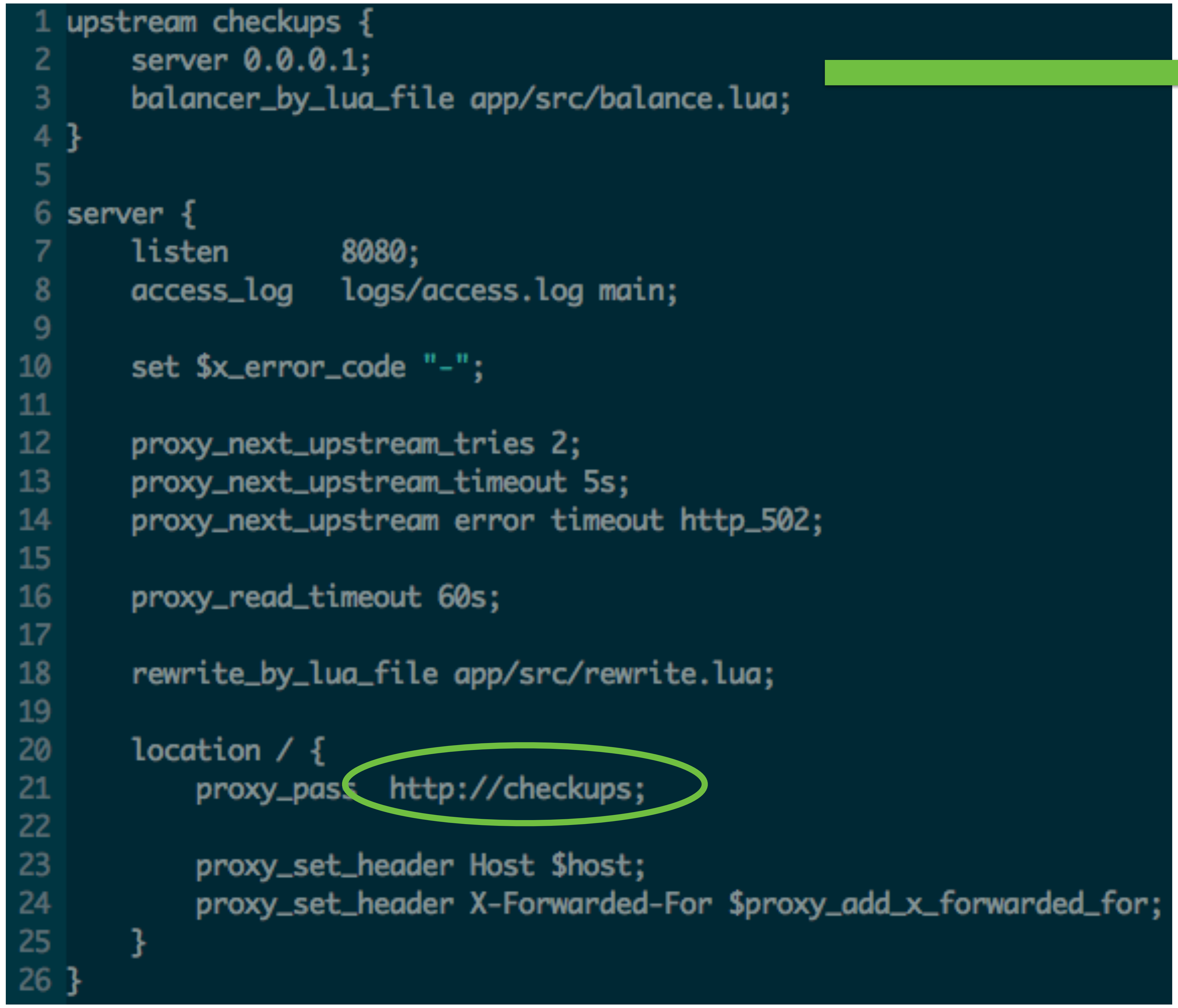
```
local status, code = balancer.get_last_failure()
if status == "failed" then
    local last_peer = ngx.ctx.last_peer
    -- mark last_peer failed
    checkups.feedback_status(skey, last_peer.host, last_peer.port, true)
end

local peer = checkups.select_peer(ngx.var.host)
ngx.ctx.last_peer = peer

balancer.set_current_peer(peer.host, peer.port)
balancer.set_more_tries(1)
```

优势

lua-resty-checkups ＋ balance_by_lua_*

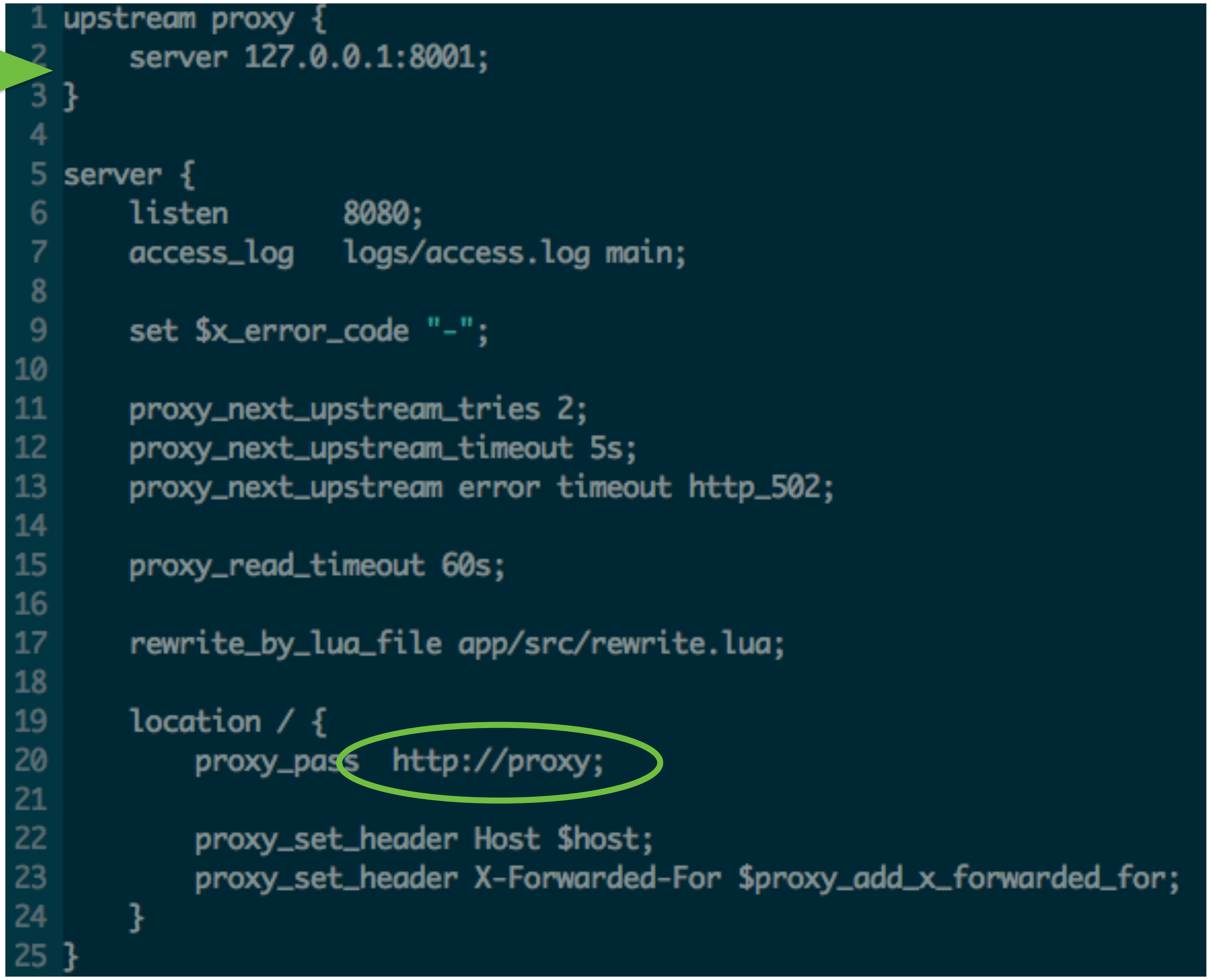
- **纯 lua 实现，不依赖第三方 C 模块**
  - 二次开发非常高效，减少维护负担
- **可以用 Nginx 原生的 proxy_***
  - proxy_next_upstream_tries / proxy_next_upstream_timeout
  - proxy_xxx
- **适用于几乎任何 ngx_lua 项目**
  - 可同时满足纯lua方案与c方案

性能对比



```
upstream checkups {
    server 0.0.0.1;
    balancer_by_lua_file app/src/balance.lua;
}

server {
    listen        8080;
    access_log    logs/access.log main;

    set $x_error_code "-";

    proxy_next_upstream_tries 2;
    proxy_next_upstream_timeout 5s;
    proxy_next_upstream error timeout http_502;

    proxy_read_timeout 60s;

    rewrite_by_lua_file app/src/rewrite.lua;

    location / {
        proxy_pass  http://checkups;

        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

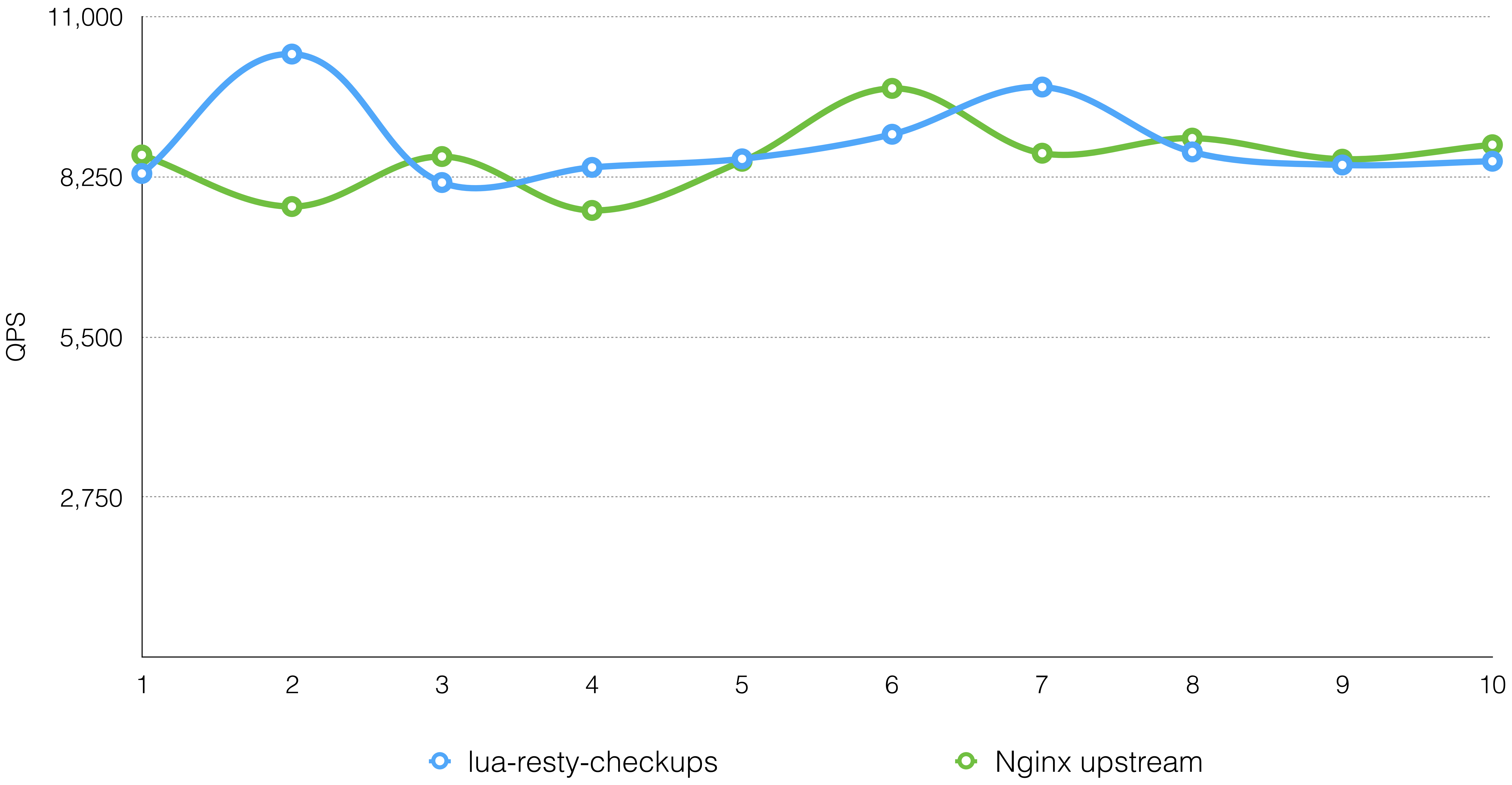
lua-resty-checkups



```
upstream proxy {
    server 127.0.0.1:8001;
}

server {
    listen        8080;
    access_log    logs/access.log main;

    set $x_error_code "-";

    proxy_next_upstream_tries 2;
    proxy_next_upstream_timeout 5s;
    proxy_next_upstream error timeout http_502;

    proxy_read_timeout 60s;

    rewrite_by_lua_file app/src/rewrite.lua;

    location / {
        proxy_pass  http://proxy;

        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Nginx upstream

性能对比


11,000
8,250
5,500
2,750
QPS
1
2
3
4
5
6
7
8
9
10
lua-resty-checkups
Nginx upstream

Slardar - 动态lua代码加载

对请求做改写

执行简单的参数检查，节省带宽

E.X. 禁止删除操作：

```
curl -d ‘

if ngx.get_method() == “DELETE” and ngx.var.host == “admin.upyun.com” then
    return ngx.exit(403)
end

‘ 127.0.0.1:1995/lua/script.admin.upyun.com
```

Slardar - 动态lua代码加载

lua-resty-load

to be open sourced...

loadfile	loadstring	setfenv
加载本地lua代码	从consul或HTTP 请求body加载代码	设置代码的执行环境

Slardar - 动态配置

config.lua

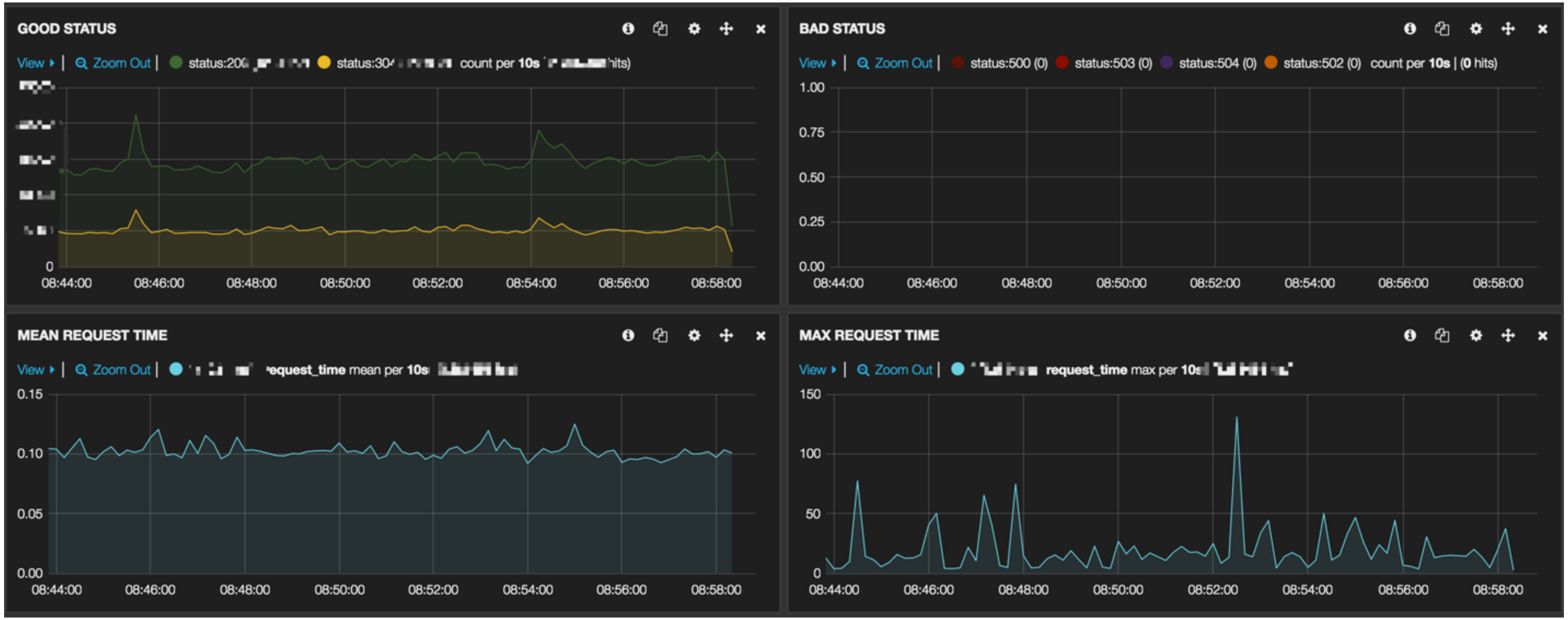
```
local _M = {}

_M.limit = {
    imgprocess = {
        rate = 100,
        burst = 20,
    },
    audio = {
        rate = 10,
        burst = 2,
    },
}

_M.topics = {
    naga = 1,
    compress = 1,
}

return _M
```

lua-resty-shcache

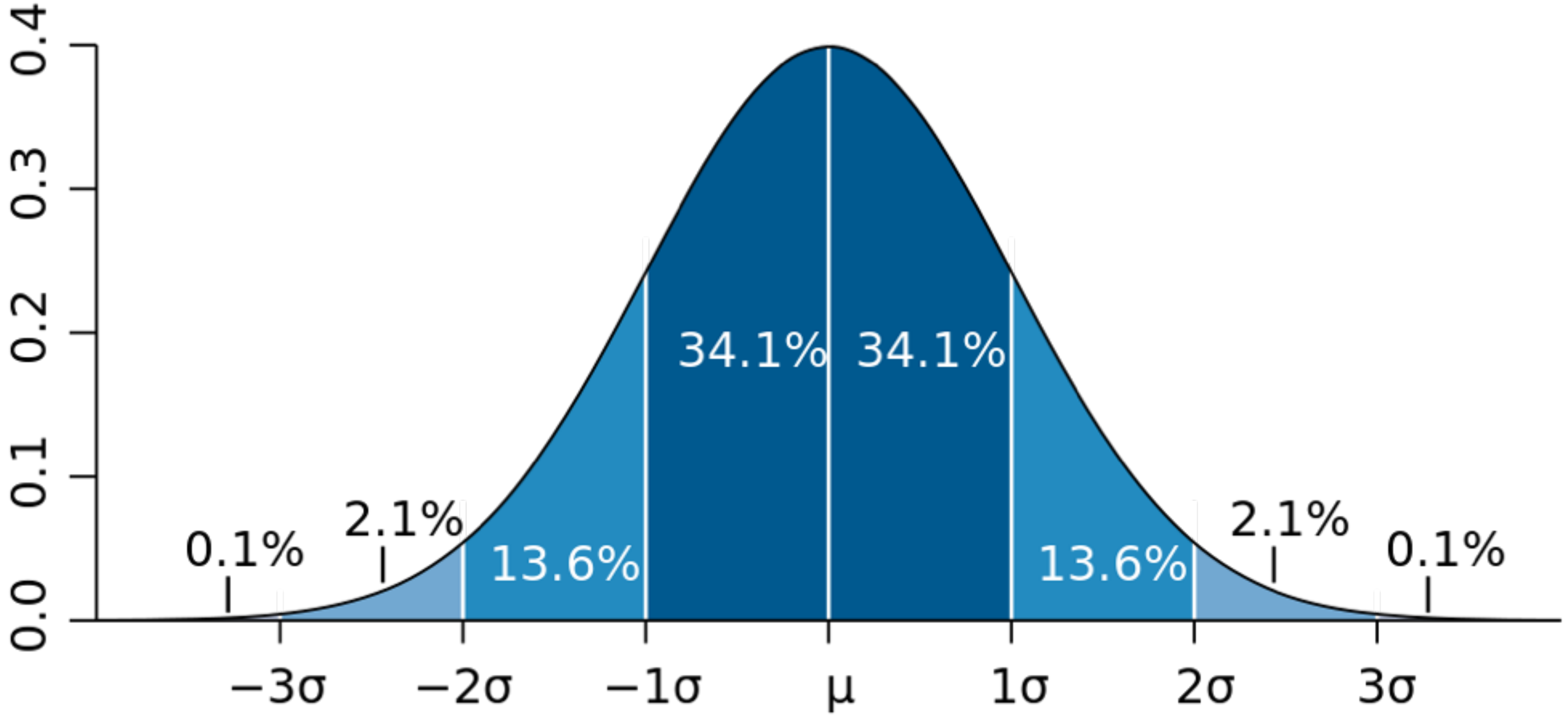
```
config = require “config”
consul = require “consul”

setmetatable(config, {
    __index = consul.load_config,
})
```

Slardar - 日志与监控

lua-resty-logger-socket → Heka → Kafka → ES



Slardar - 异常机器自动摘除



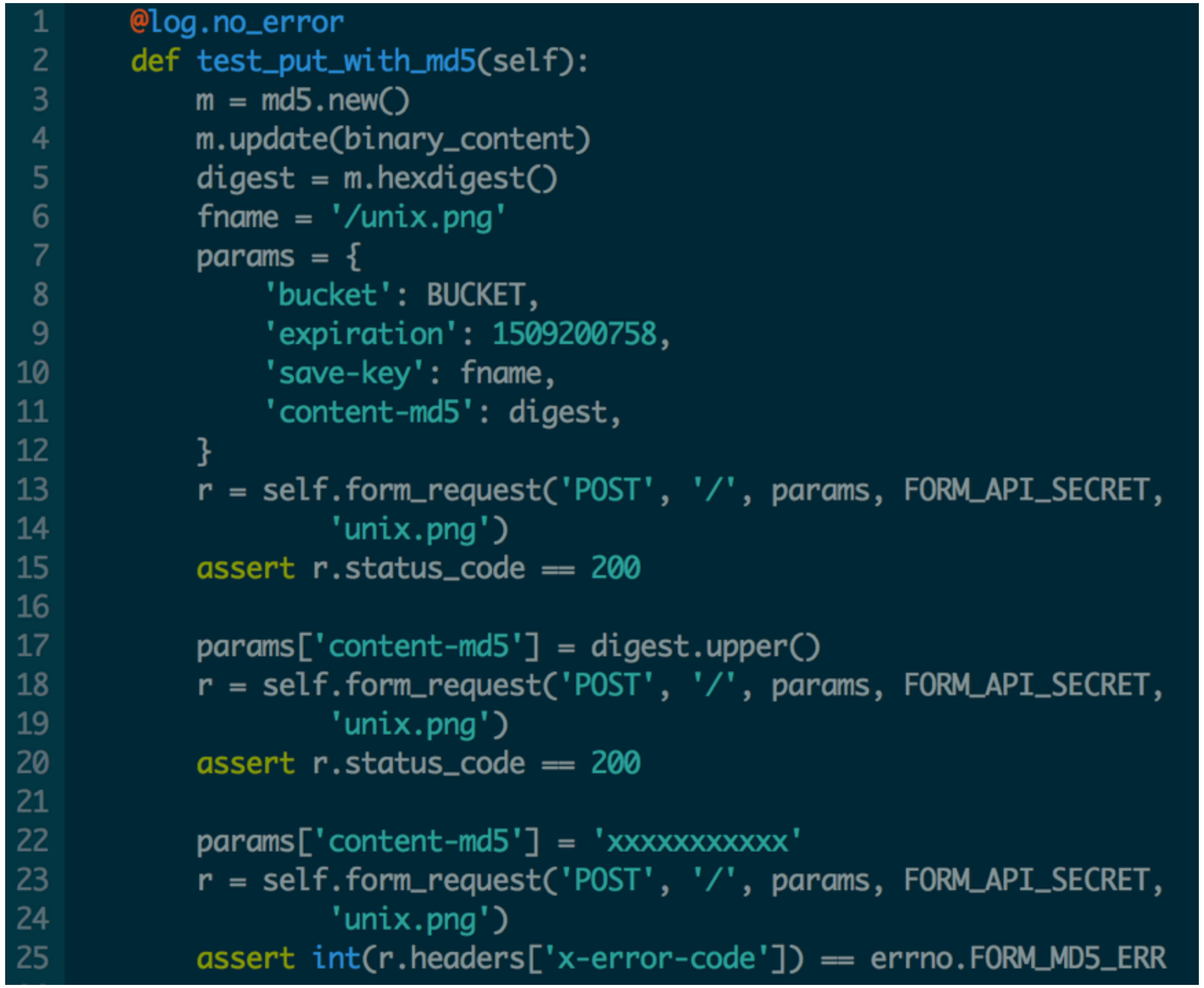
图片来源:https://thecuriousastronomer.wordpress.com/2014/06/26/what-does-a-1-sigma-3-sigma-or-5-sigma-detection-mean/

$$|x_{502} - \mu| > 3\sigma$$ → update_upstream

测试

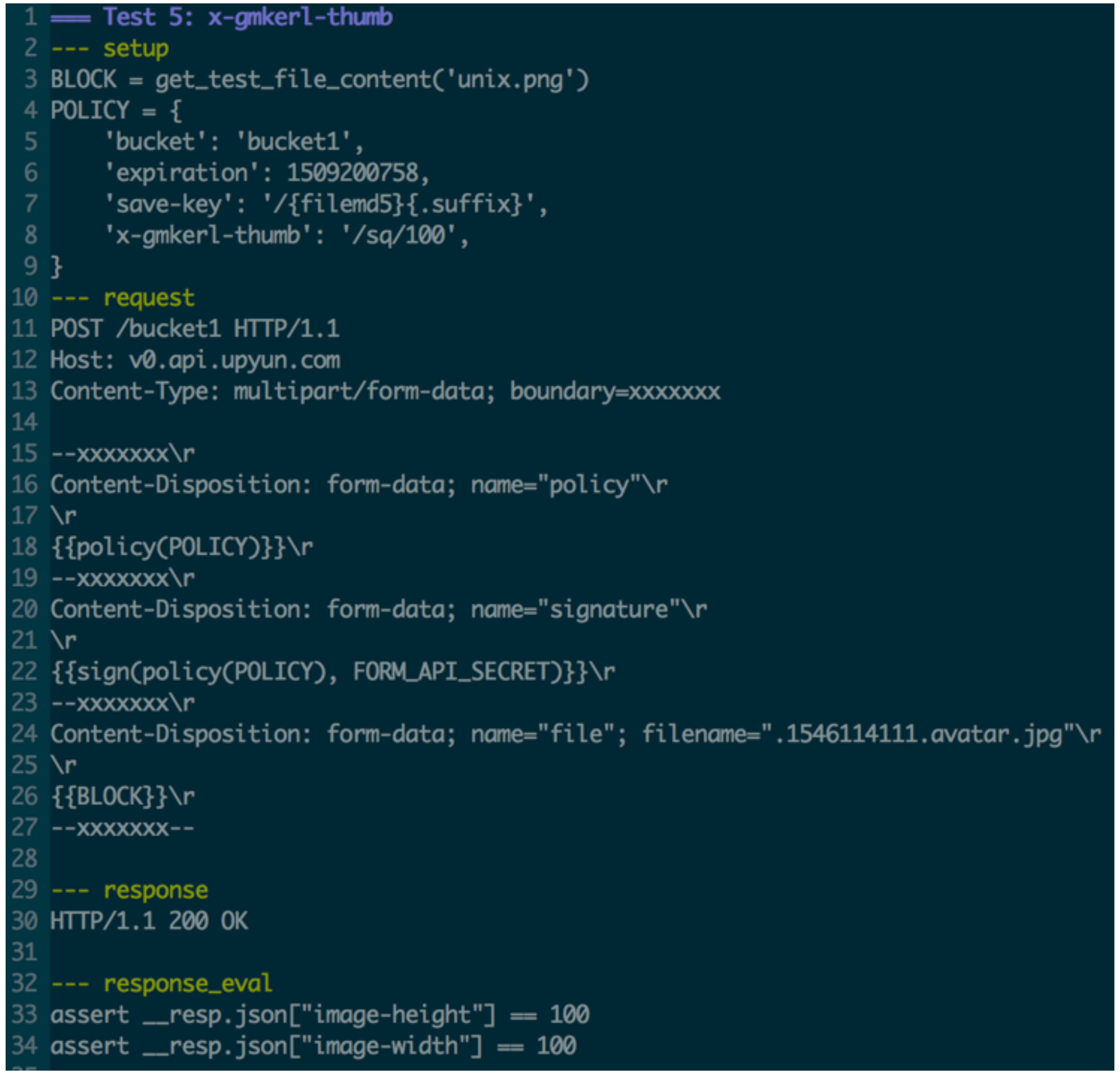
Test::Nginx or Python?

测试 - Inspired by Test::Nginx



```
    @log.no_error
    def test_put_with_md5(self):
        m = md5.new()
        m.update(binary_content)
        digest = m.hexdigest()
        fname = '/unix.png'
        params = {
            'bucket': BUCKET,
            'expiration': 1509200758,
            'save-key': fname,
            'content-md5': digest,
        }
        r = self.form_request('POST', '/', params, FORM_API_SECRET,
                'unix.png')
        assert r.status_code == 200

        params['content-md5'] = digest.upper()
        r = self.form_request('POST', '/', params, FORM_API_SECRET,
                'unix.png')
        assert r.status_code == 200

        params['content-md5'] = 'xxxxxxxxxxx'
        r = self.form_request('POST', '/', params, FORM_API_SECRET,
                'unix.png')
        assert int(r.headers['x-error-code']) == errno.FORM_MD5_ERR
```

python unittest



```
=== Test 5: x-gmkerl-thumb
--- setup
BLOCK = get_test_file_content('unix.png')
POLICY = {
    'bucket': 'bucket1',
    'expiration': 1509200758,
    'save-key': '/{filemd5}{.suffix}',
    'x-gmkerl-thumb': '/sq/100',
}
--- request
POST /bucket1 HTTP/1.1
Host: v0.api.upyun.com
Content-Type: multipart/form-data; boundary=xxxxxxx

--xxxxxxx\r
Content-Disposition: form-data; name="policy"\r
\r
{{policy(POLICY)}}\r
--xxxxxxx\r
Content-Disposition: form-data; name="signature"\r
\r
{{sign(policy(POLICY), FORM_API_SECRET)}}\r
--xxxxxxx\r
Content-Disposition: form-data; name="file"; filename=".1546114111.avatar.jpg"\r
\r
{{BLOCK}}\r
--xxxxxxx--

--- response
HTTP/1.1 200 OK

--- response_eval
assert __resp.json["image-height"] == 100
assert __resp.json["image-width"] == 100
```

ytest

未来 —— 支持 TCP 动态路由

Thanks

Q & A