Lua & co

Plan

- A propos de moi
- Intro à Lua
- Utilisé où, pour faire quoi ?
- Quelques notions
- Luajit / luajit ffi
- Lua pour le web : Openresty

A propos de moi

du dev , de l'admin sys & réseaux

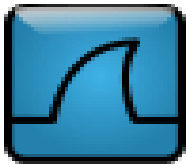
Et Depuis 2012

Formateur @ Afpa

Introduction a Lua

- Créé en 1993 par Roberto Lerusalimschy, Waldemar Celes, Luiz Henrique de Figueiredo – au Bresil
- Langage de script rapide, léger (24000 loc écrit en ANSI C), facilement intégrable.
- Multi-paradigme : procédural, OO, fonctionnel
- Tourne sur toutes les distributions Unix et Windows, smartphones (Android, iOS, Symbian, Windows Phone), sur processeurs embarqués (ARM, Rabbit), IBM mainframes, etc.
- C<->Lua bindings
- Licence MIT

Utilisé où ?



Utilisé comme :

- Extension
- Monitoring
- Scripting (DSL)
- Proxy – Load balancing
- Gestion des caches
- Web Application Firewall
- Robotique
- Game play
- Etc...

Notions

```lua
local str = "je suis une chaine de charactères"
local isOK = true
local nombre = 5
local decimal = 5.2
local car = 'T'
a, b, c = 1, 2, 3 --variable globale a==1, b==2, c==3
local unetable = {1,2,3} --tableau simple dimension ou cle valeur

function kikou()
end

local kikou2 = function()
end

for var = start, valend, step do
 --
end

if (a == 1) then
  print(b) --affiche 2
end
```

Notions suite...

```lua
--commentaire
local prenom = "lionel"
local affiche = print

affiche(prenom)

local personne = {
  prenom = "lionel",
  developeur = true,
  poids = 3750,
}
personne.dit_bonjour = function()
    print('kikou')
end

affiche(personne.poids)

personne.dit_bonjour()

```

```lua
--exemple tableau unidimensionnel
a = {7,12,4}
--iteration
for i, value in ipairs(a) do
  print(i, value)
end

--une fonction peut retourner plusieurs valeurs
function connect()
  --connect blabla...
  if pb then return nil, err
  else return db_handle end
end

local db, err = connect()
if not db then
    print("oulala:"..err)
end
```

Notions suite...

- Concept de Metatable
  - Permet de changer le comportement des tables
    - Surcharge des opérateurs « +, -, /, * » (exemple Tbl1 + tbl2)
    - Comparaisons
    - Heritage, interface (OOP)

Notions suite… OOP

```lua
-- mymodule.lua
local M = {} -- public interface

-- private
local x = 1
local function baz() print 'test' end

function M.foo() print("foo", x) end

function M.bar()
  M.foo()
  baz()
  print "bar"
end

return M

-- Example usage:
local MM = require 'mymodule'
MM.bar()
```

```lua
-- Meta class
local Shape = {}
local mt = { __index = Shape }

function Shape.new (self, side)

   local area = side*side;
   return setmetatable({area = area}, mt)
end

-- Base class method printArea
function Shape.printArea (self)
   print("The area is ",self.area)
end

-- Creating an object
myshape = Shape:new(10)
myshape:printArea()
```

Notions fin

- Multithreading : les **coroutines** :
  - pas multithread au sens OS, non pré-emptif. Permet les opérations non bloquantes sur l'I/O.
- Bizarreries :
  - Indice des tableaux commence par **1** !
  - Différent de 1 => if (i~=1) then … end
  - Pas d'opérateur ternaire (x>3 ? 1 : 0;)
  - Pas de raccourci pour l'incrément (pas de i++ ni i+=)
    - => i = i + 1
  -

Luajit

- Compilateur JIT pour lua
- Super rapide ! : 3 a 100x
- Et petit : < 100 KB
- Integration facilitée avec les librairies natives C : => ffi (foreign function include)

Luajit : ffi - exemple

```c
//libsuper.c
int superfonctionbionic(int a, int b)
{
    return a + b;
}
```

```lua
local ffi = require("ffi")
ffi.cdef[[
typedef struct { uint8_t red, green, blue, alpha; } rgba_pixel;
int printf(const char *fmt, ...);
int superfonctionbionic(int a, int b);
]]

local img = ffi.new("rgba_pixel[?]",1)
img[1].red = 251
img[1].green = 150
img[1].blue = 89
img[1].alpha = 255
ffi.C.printf("red color is %g !\n", img[1].red)

local bioniclib = ffi.load('./libsuper.so')
local res = bioniclib.superfonctionbionic(1,2)

print(res)
--res = 3
```

Outils & support

- Outils :
  - Gestionnaire de paquet : Luarocks
  - Tests unitaires : « busted » ou « luaunit » et d’autres…
  - Debugger :IDE ZeroBrane Studio
- Supports
  - Mailing list
  - irc.freenode.net #lua
  - Workshops

Lua pour le web

- Pléthore de solutions,
  - du 100 % Lua :
    - Xavante, Pegasus
  - Extension de serveur existant :
    - LightHttpd, Apache, Nginx, Tornado...
  - NodeJs like :
    - Luvit

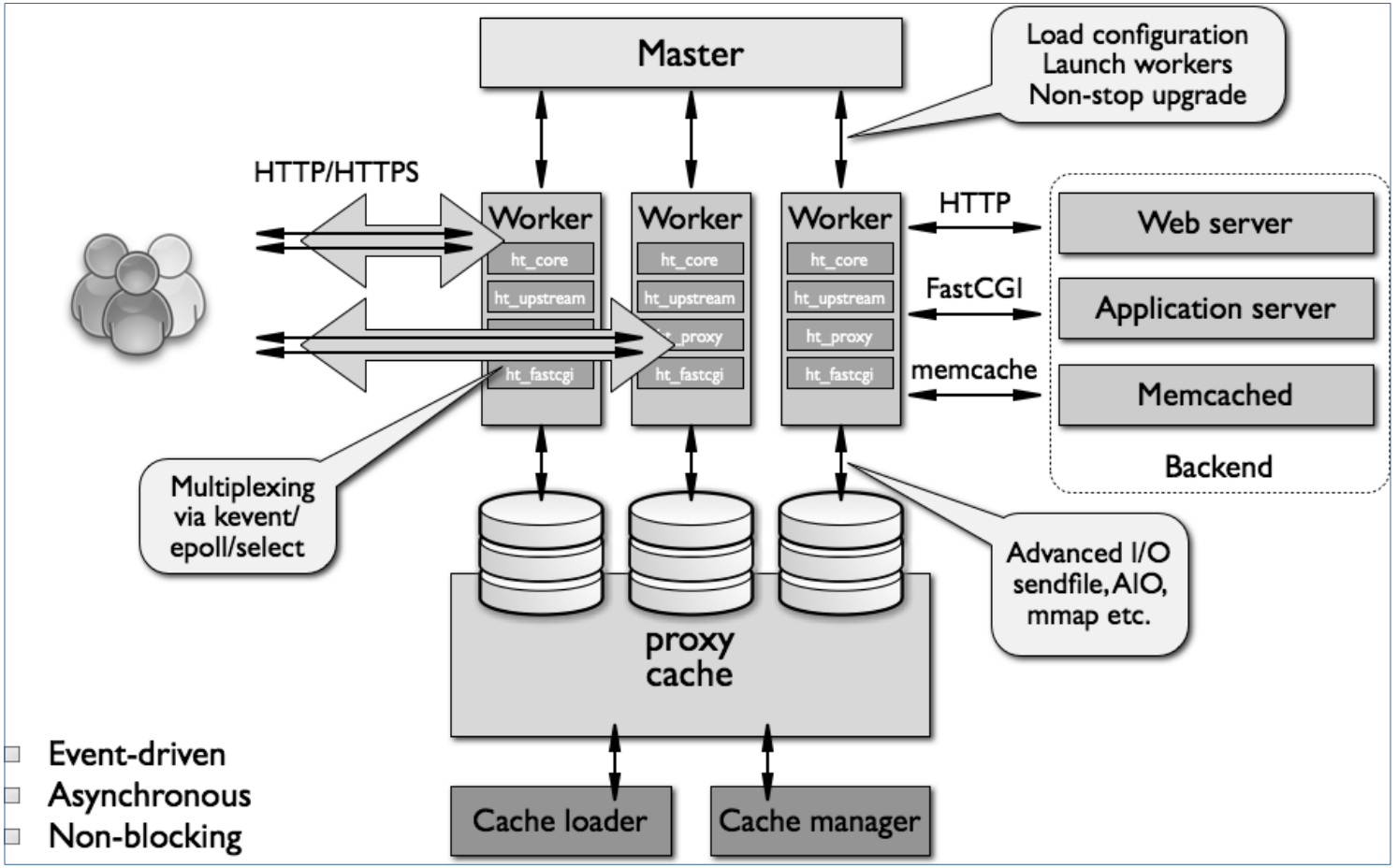
Openresty

Openresty = Nginx + Luajit + Modules

Nginx

- HTTP server, reverse proxy, proxy mail
- Modèle asynchrone
- Empreinte mémoire faible
- Modulaire
- Extensible avec des modules écrit en C

Nginx modèle



Source : http://www.aosabook.org/en/nginx.html

Nginx – serveur min.

```
worker_processes  1;
error_log logs/error.log;
events {
    worker_connections 1024;
}
http {
    server {
        listen 8000;
        location / {

          index index.html;
          root public/;
    }
}
```

```
$> curl http://0.0.0.0:8000/index.html
```

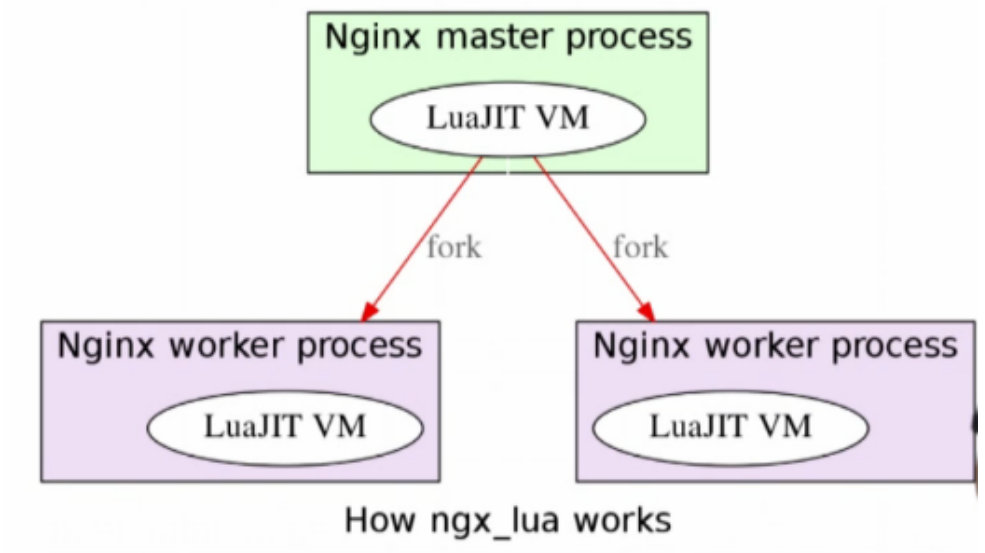
Openresty

- Créé et maintenu principalement par Yichun Zhang (@agentzh) .(Taobao.com puis Clouflare)
- En production sur des sites à fort trafic (Alibaba group, Clouflare)
- Modules I/O non bloquants pour :
  - Memcached
  - Redis
  - MySQL / Drizzle
  - PostgreSQL
- Websockets, CLI, Json, etc..

Openresty

- Directives pour executer du code Lua durant les phases Nginx
  - Init phase
  - Rewrite phase.
  - Access phase.
  - Content phase.
  - Log phase
  - Et d’autres…
  - API Lua pour accéder à l’environnement Nginx

Openresty



- Le code Lua est exécuté directement au sein des workers

Openresty – Hello world

```
   location /hellolua {
   default_type 'text/plain';

   content_by_lua_block {
       local name = ngx.var.arg_name or "Anonymous"
       ngx.say("Hello, ", name, "!")
   };
}

$> curl http://localhost/hellolua?name=Lua

Hello, Lua
```

Openresty : Chargement module externe

```
location /xxxx {
    rewrite_by_lua_file /path/to/rewrite.lua;
    access_by_lua_file /path/to/access.lua;
    content_by_lua_file /path/to/content.lua;
}
```

- Les modules sont chargés qu'une seule fois par worker à la première requête

Openresty -sous requête

```
location / {
    content_by_lua '
        local res = ngx.location.capture("/memcached",
            { args = { cmd = "incr", key = ngx.var.uri } }
        )
    ';
}

location /memcached {
    set $memc_cmd $arg_cmd;
    set $memc_key $arg_key;
    memc_pass 127.0.0.1:11211;
}
```

Openresty – cosocket

- Connection E/S non bloquante avec la librairie Cosocket :
  - TCP ou Unix Domain sockets
  - Permet une écriture séquentiel du code tout en s’assurant que l’exécution soit asynchrone ! (voir la joie des callbacks ;-))
  - Mécanisme « Keepalive » pour la réutilisation des connections.

Openresty : client TCP

```
location /memcached {
    content_by_lua_block {'
        local sock = ngx.socket.connect("127.0.0.1", 11211)
        sock:send("SET foo bar 3600\r\n")
        local line = sock:receive()
        if line then
            ngx.say(line)
        end
        sock:setkeepalive()
    };
}
```

```
$> curl http://localhost/memcached
```

```
STORED
```

Openresty : mysql

```
location /todos {
    content_by_lua_block {
    local mysql = require "resty.mysql"
    local db, err = mysql:new()
       if not db then
           ngx.say("failed to instantiate mysql: ", err)
           return
        end

       db:set_timeout(1000) -- 1 sec
       --connect…
       local res, err, errcode, sqlstate = db:query("select * from tasks")
       if not res then
            ngx.say("bad result: ", err, ": ", errcode, ": ", sqlstate, ".")
            return
       end

       local cjson = require "cjson"
       ngx.say("result: ", cjson.encode(res))
    } ;
}
```

Openresty : mini app

```
location /api/v0.1/todos {
        default_type application/json;

        content_by_lua_block {

          local todo_api = require "todo-app"

          --get location after /api/v0.1/todos
          local sub_loc = string.sub(ngx.var.request_uri, string.len(ngx.var.location)+1)
          if sub_loc == "" then sub_loc = "/" end

          local router = require "router"
          local r = router.new()

          r:get("/", todo_api.getAll)
          r:post("/", todo_api.create)
          r:delete("/:id", todo_api.delete)
          r:put("/:id", todo_api.update)

          --execute routes
          r:execute(ngx.var.request_method, sub_loc, ngx.req.get_uri_args())

        }
    }
```

https://github.com/lduboeuf/openresty-todo-app

Openresty - outils

- Profilage en temps réel
- Test suite
- Gestionnaire de package (OPM)
- Mailing list

Openresty : encore plus

- Lapis (http://leafo.net/lapis/) = openresty + libs utils + templating html + sessions
- Sailor (http://sailorproject.org/) = MVC, routing, templating, model generator, etc...

Bonus !

Protocole agnostique : Luvit

- Créé par Tim Caswell (@creationix) , un des premier contributeur de NodeJs.
- En production chez Rackspace pour leurs agents de monitoring.
- Initialement l’idée était de porter Node en Lua avec des api similaires : Gain en mémoire de 20x !
- Repose comme Node sur la lib LibUV : event loop, E/S Asynchrone , ...
- Intègre la librairie OpenSSL, zlib.
- Objectif : pouvoir créer des applications auto-executables, rapides, légères.

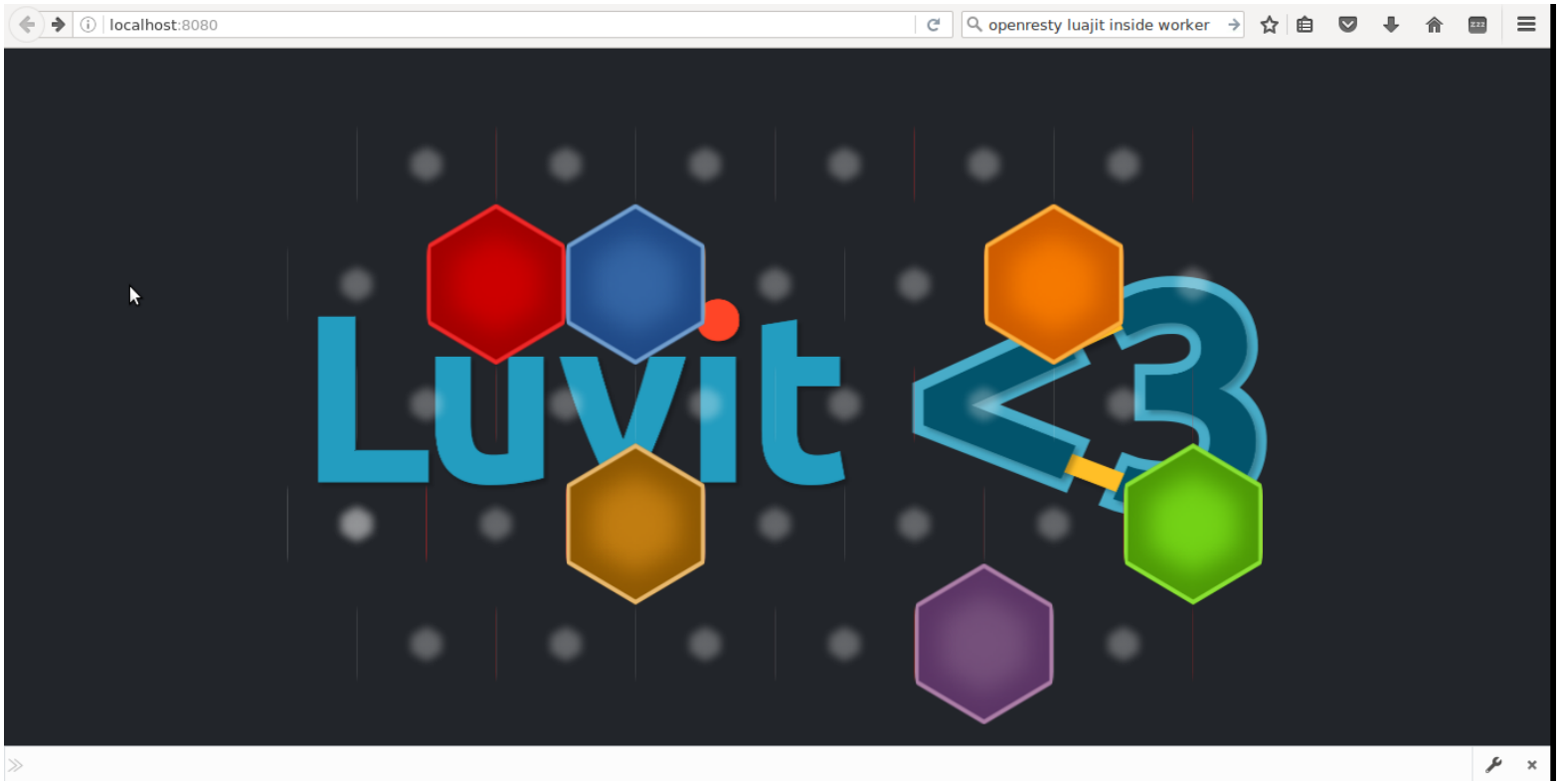
Luvit

- On peut créer une appli « node-style » (basée sur des callbacks) ou séquentielle (mais non bloquante grâce aux co-routines).
- Gestionnaire de package, repository et compilateur : lit (créé avec Luvit…)
  - Exemple : compiler une appli et l'executer

```
git clone git@github.com:creationix/hexes.git
cd hexes
lit install
lit make
./hexes
```

Luvit : exemple

https://github.com/creationix/hexes



Luvit

- Cool si :
  - Pas peur de la doc sporadique
  - Prêt à coder son connecteur ;-) (pour l'instant existe un wrapper pour Prostgresql, Redis).
  - Mettre la main dans le cambouis
- Communauté limitée mais très réactive et sympa ;-) :freenode IRC #luvit , et mailing list

Bah voilà , Merci !