Developing a user-friendly OpenResty application

OpenResty Con 2017 - Beijing

$ whoami

Thibault Charbonnier

- Lead Engineer @ **Kong** (https://konghq.com)
- Main contributor @ **Kong API Gateway** (https://github.com/Kong/kong)
- OpenResty Contributor on my spare time
- https://github.com/thibaultcha
- https://chasum.net

A user-friendly OpenResty application?

Initial assumption: Most OpenResty applications that we know of seem to be private, deployed on internal infrastructures.

If we were to ship an on-premise OpenResty application, it should be **easy to install and deploy**:

- ➔ NGINX processes only (no other daemons in the system)
- ➔ Minimal libraries dependencies (pure LuaJIT/OpenResty)
- ➔ Horizontally scalable (clustering)
- ➔ Platform agnostic

A user-friendly OpenResty application?

Example: recurring background jobs

➔ Cronjob

```
*/5 * * * * curl -XGET 'http://localhost:8000/job?hello=world'
```

vs

➔ ngx.timer API

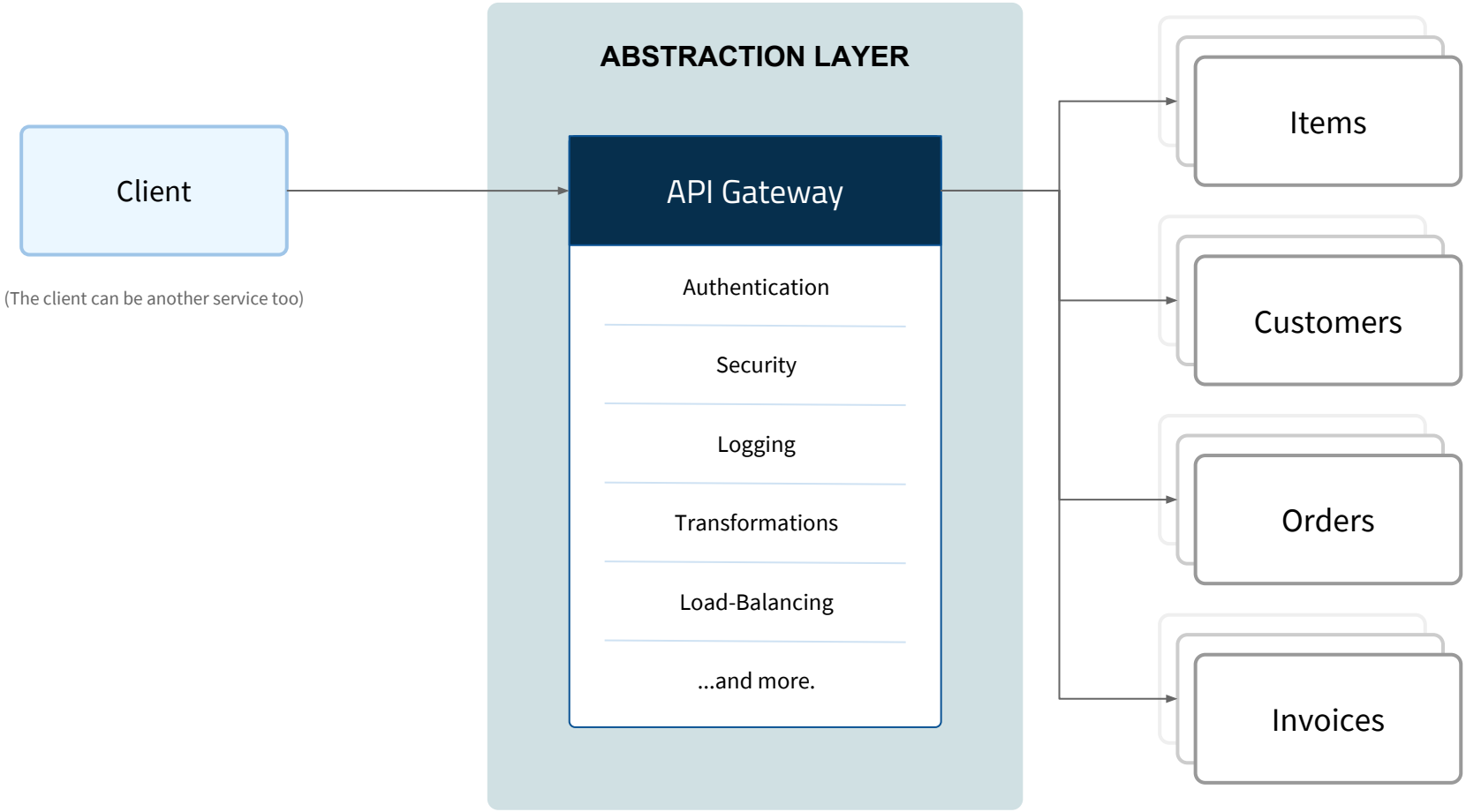
```
ngx.timer.every(60 * 5, do_job(), "world")
```

What is Kong?

Short introduction to API Gateways

What is an API Gateway?

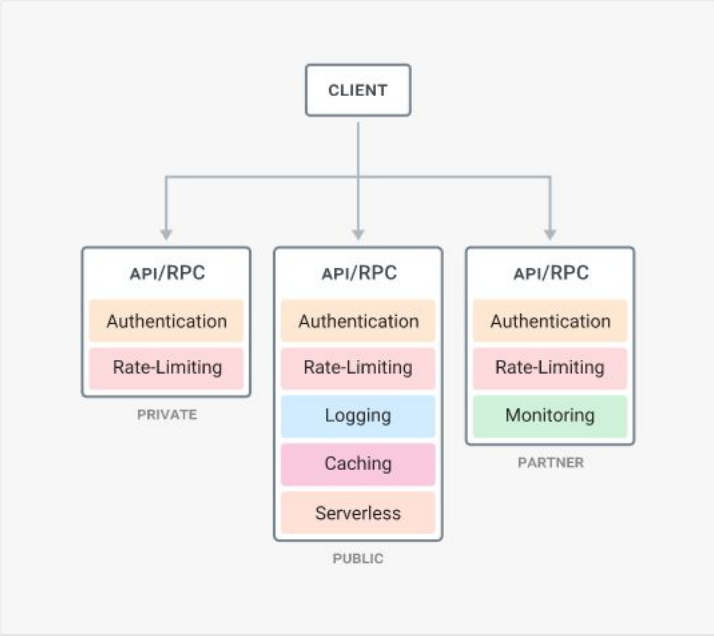
It's a reverse proxy, sitting between your clients and your upstream services



What is an API Gateway?

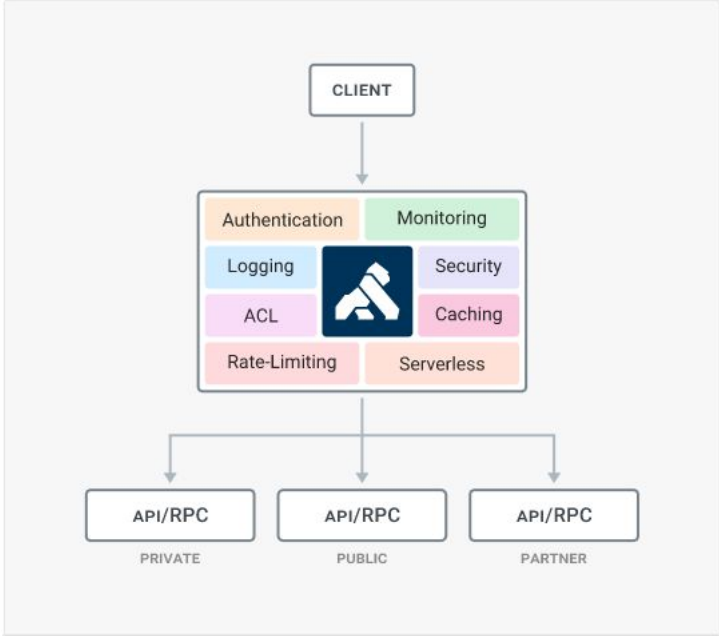
Reduce Code Duplication, Orchestrate Common Functionalities

Legacy Architecture



- ✗ Common functionality is duplicated across multiple services
- ✗ Systems tend to be monolithic and hard to maintain
- ✗ Difficult to expand without impacting other services
- ✗ Productivity is inefficient because of system constraints

Kong Architecture



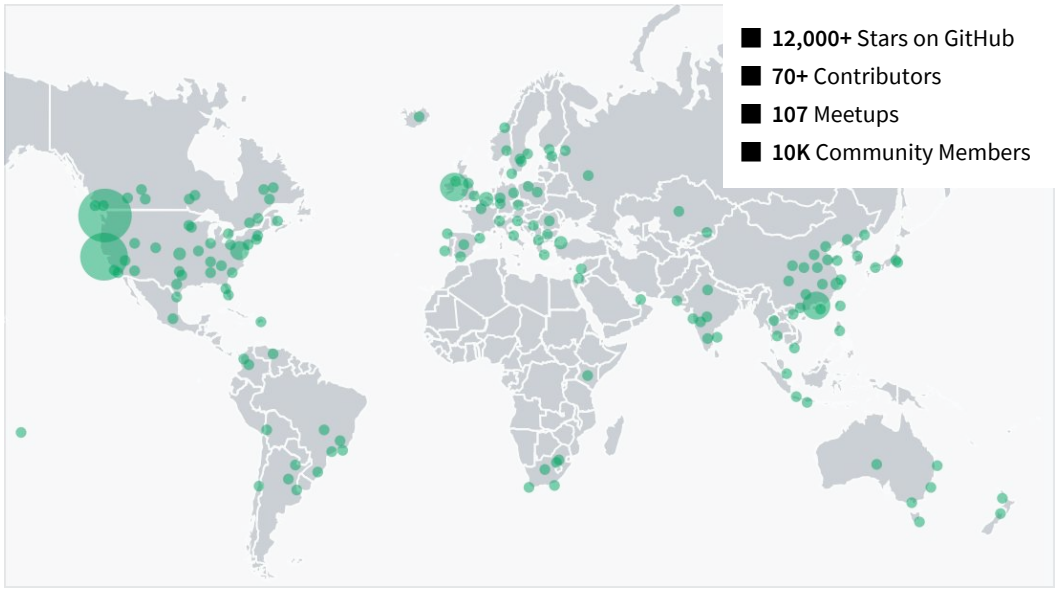
- ✓ Kong centralizes and unifies functionality into one place
- ✓ Build efficient distributed architectures ready to scale
- ✓ Expand functionality from one place with a simple command
- ✓ Your team is focused on the product, Kong does the REST

Kong

Open Source API Gateway

Built with **OpenResty.**

- Open Source
- Extensible via Plugins (**60+** available)
- **Sub-millisecond** latency on most use-cases
- Platform Agnostic
- Horizontally Scalable



https://github.com/Mashape/kong

https://getkong.org

Kong

```
init_by_lua_block {
    kong = require 'kong'
    kong.init()
}
...
location / {
    set $upstream_scheme '';
    set $upstream_uri    '';

    rewrite_by_lua_block { kong.rewrite() }

    access_by_lua_block { kong.access() }

    proxy_http_version 1.1;
    proxy_pass $upstream_scheme://kong_upstream$upstream_uri;

    header_filter_by_lua_block { kong.header_filter() }

    body_filter_by_lua_block { kong.body_filter() }

    log_by_lua_block { kong.log() }
}
```

Kong v0.1 dependencies

- Proxy + Lua middleware → **OpenResty**
- A command line interface (CLI) → **PUC-Rio Lua 5.1** ☹
- DNS resolution → **dnsmasq** ☹
- Clustering of nodes → **Serf** ☹
- Generate UUIDs → **libuuid** ☹
- Database → **Cassandra** □

Lots of dependencies for users to install...

Kong v0.1 dependencies

Processes

- Kong's CLI
- NGINX
- dnsmasq
- serf

Ports

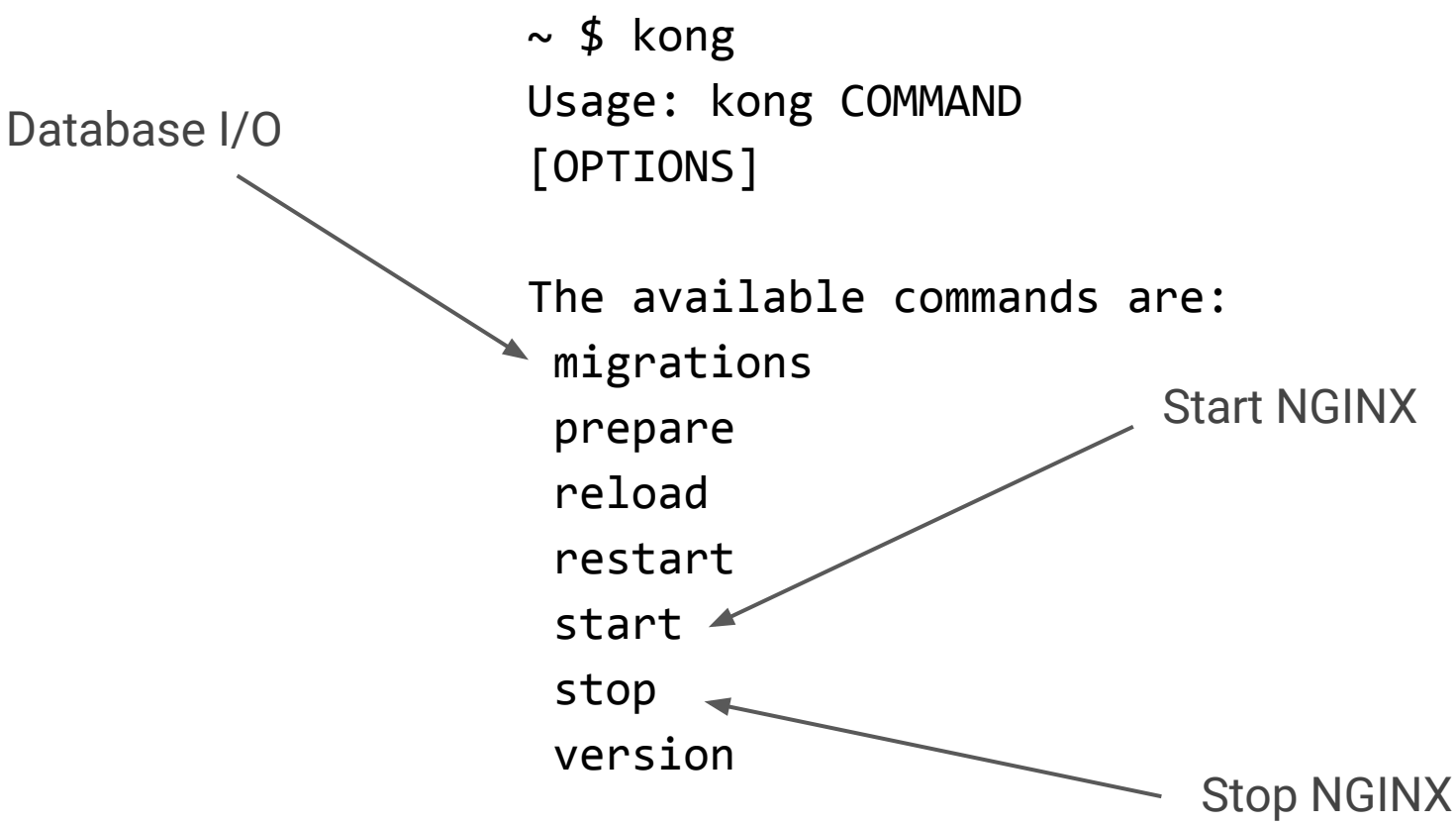
- 80 + 8001 (NGINX)
- 8053 (dnsmasq)
- 7946 + 7373 TCP/UDP (serf)

Less than ideal for dockerized environments or firewall rules...

CLI

Choosing an interpreter

Build an OpenResty CLI



Build an OpenResty CLI

```
#!/usr/bin/env lua

require("kong.cmd.init")(arg)
```

- No FFI (LuaJIT only)
- No support for cosockets (LuaSocket + LuaSec fallback)
- Missing `ngx.*` API

Lots of fragmentation between our OpenResty and CLI code ☹

Build an OpenResty CLI

```
#!/usr/bin/env luajit

require("kong.cmd.init")(arg)
```

- LuaJIT FFI available
- No support for cosockets (LuaSocket + LuaSec fallback)
- Missing `ngx.*` API

An improvement but fragmentation is still very much of an issue ☹

Build an OpenResty CLI

```
#!/usr/bin/env resty

require("kong.cmd.init")(arg)
```

- Runs in `timer` context thanks to https://github.com/openresty/resty-cli
- Cosockets available
- `ngx.*` API available
- LuaJIT FFI available

We can reuse our OpenResty and CLI code

No PUC-Rio Lua dependency

Build an OpenResty CLI

- Proxy + Lua middleware → **OpenResty**
- A command line interface (CLI) → ~~PUC-Lua 5.1~~ **OpenResty** 
- DNS resolution & Load balancing → **dnsmasq**
- Clustering of nodes → **Serf**
- Generate UUIDs → **libuuid**
- Database → **Cassandra**

Using resty-cli in busted

Using resty-cli in busted

Kong's test framework is busted since 2014

https://github.com/Olivine-Labs/busted

```
describe('Busted unit testing framework', function()
  it('should be easy to use', function()
    assert.truthy('Yup.')
  end)

  it('should have lots of features', function()
    -- deep check comparisons!
    assert.same({ table = 'great'}, { table = 'great' })

    -- or check by reference!
    assert.is_not.equals({ table = 'great'},
                         { table = 'great'})

    assert.falsy(nil)
    assert.error(function() error('Wat') end)
  end)
end)
```

Using resty-cli in busted

We changed the interpreter from PUC-Rio Lua to resty-cli

```
-- ./rbusted
#!/usr/bin/env resty

-- Busted command-line runner
require 'busted.runner'({ standalone = false })
```

https://github.com/thibaultcha/lua-resty-busted

Using resty-cli in busted

```lua
-- t/sanity_spec.lua
describe("openresty script", function()
  it("should run in ngx_lua context", function()
    assert.equal(0, ngx.OK)
    assert.equal(200, ngx.HTTP_OK)
  end)

  it("should yield", function()
    ngx.sleep(3)
    assert.is_true(1 == 1)
  end)
end)
```

Using resty-cli in busted

Improving busted for OpenResty development

```
~ $ rbusted --o=tap t/sanity_spec.lua
ok 1 - openresty script should run in ngx_lua context
ok 2 - openresty script should yield
1..2
```

Now we can test our OpenResty code with busted!

UUID generation

And PRNG seeding

Removing the libuuid dependency

- PUC-Rio Lua - https://github.com/Tieske/uuid
  - Slowest implementation
  - Generates invalid v4 UUIDs
- libuuid binding (Lua C API) - https://github.com/Kong/lua-uuid
  - Safe underlying implementation
  - External dependency
- libuuid binding (LuaJIT FFI) - https://github.com/bungle/lua-resty-uuid
  - Safe underlying implementation
  - External dependency
- LuaJIT - https://github.com/thibaultcha/lua-resty-jit-uuid
  - Seems to be the fastest implementation
  - Uses LuaJIT's PRNG

Removing the libuuid dependency

```
LuaJIT 2.1.0-beta1 with 1e+06 UUIDs
UUID v4 (random) generation
1. resty-jit-uuid   took:   0.064228s    0%
2. FFI binding      took:   0.093374s   +45%
3. C binding        took:   0.220542s   +243%
4. Pure Lua         took:   2.051905s   +3094%

UUID v3 (name-based and MD5) generation if supported
1. resty-jit-uuid   took:   1.306127s

UUID v5 (name-based and SHA-1) generation if supported
1. resty-jit-uuid   took:   4.834929s

UUID validation if supported (set of 70% valid, 30% invalid)
1. resty-jit-uuid (JIT PCRE enabled)    took:   0.223060s
2. FFI binding                          took:   0.256580s
3. resty-jit-uuid (Lua patterns)        took:   0.444174s
```

Caution: PRNG seeding in NGINX workers

```
init_by_lua_block {
    --math.randomseed(ngx.time()) <-- AVOID
}

init_worker_by_lua_block {
    math.randomseed(ngx.time() + ngx.worker.pid())
    math.randomseed = function()end -- ensure we prevent re-seeding
}
```

- Be wary of calling `math.randomseed()` in `init_by_lua`
- Seeding in `init_worker_by_lua` is safer
- Still, some external dependencies may call `math.randomseed()` again

A possible solution: https://github.com/openresty/lua-resty-core/pull/92

Build an OpenResty CLI

- Proxy + Lua middleware → **OpenResty**
- A command line interface (CLI) → ~~**PUC-Lua 5.1**~~ **OpenResty**
- DNS resolution & Load balancing → **dnsmasq**
- Clustering of nodes → **Serf**
- Generate UUIDs → ~~**libuuid**~~ **OpenResty** 
- Database → **Cassandra**

DNS Resolution

DNS Resolution

NGINX does not use the system resolver, and needs a user-specified **name server**.

More often than not, this deceives users:

- ➔ Ignores /etc/resolv.conf ☹
- ➔ Ignores /etc/hosts ☹
- ➔ No support for SRV records ☹
- ➔ Unusable with `balancer_by_lua` ☹

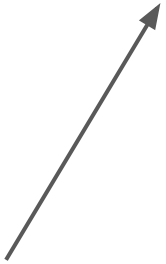
DNS Resolution

Temporary solution: **dnsmasq**

```
~ $ kong start --vv
...
2017/03/01 14:45:35 [debug] found 'dnsmasq' executable at /usr/sbin/dnsmasq
2017/03/01 14:45:35 [debug] starting dnsmasq: /usr/sbin/dnsmasq -p 8053 --pid-file=/usr/local/kong/pids/dnsmasq.pid -N -o
--listen-address=127.0.0.1
```

```
http {
    resolver 127.0.0.1:8053 ipv6=off;

    ...
}
```



dnsmasq daemon

- Parses /etc/resolv.conf
- Parses /etc/hosts
- Support for SRV records
- Still unusable with `balancer_by_lua` ☹
- New dependency ☹

DNS Resolution

To remove our dnsmasq dependency, and use balancer_by_lua, we must resolve DNS records in the Lua land.

Part of the solution: https://github.com/openresty/lua-resty-dns

- Pure Lua, bundled with OpenResty
- Resolves, A, AAAA, CNAME, SRV records (and more)
- No `/etc/hosts` parsing ☹
- No `/etc/resolv.conf` parsing ☹
- No results caching ☹
- No DNS load-balancing ☹

DNS Resolution

lua-resty-dns-client - https://github.com/Kong/lua-resty-dns-client
Author: Thijs Schreijer (@tieske)

- Built on top of lua-resty-dns
- Parses `/etc/hosts`
- Parses `/etc/resolv.conf`
- Built-in cache & asynchronous querying
- Built-in DNS load-balancing
- 

DNS Resolution

- Proxy + Lua middleware → **OpenResty**
- A command line interface (CLI) → ~~**PUC-Lua 5.1**~~ **OpenResty**
- DNS resolution & Load balancing → ~~**dnsmasq**~~ **OpenResty**
- Clustering of nodes → **Serf**
- Generate UUIDs → ~~**libuuid**~~ **OpenResty**
- Database → **Cassandra**

Clustering

Clustering

- Kong nodes connected to the **same database** (PostgreSQL or Cassandra) share the same configuration.
- To limit database traffic, Kong nodes maintain their own **cache**.
- **lua-shared-dict** + **lua-resty-lock** allow Kong to avoid the “dogpile effect” (cache stampede).

```
http {
    lua_shared_dict kong_cache ${{MEM_CACHE_SIZE}};
    ...
}
```

Clustering

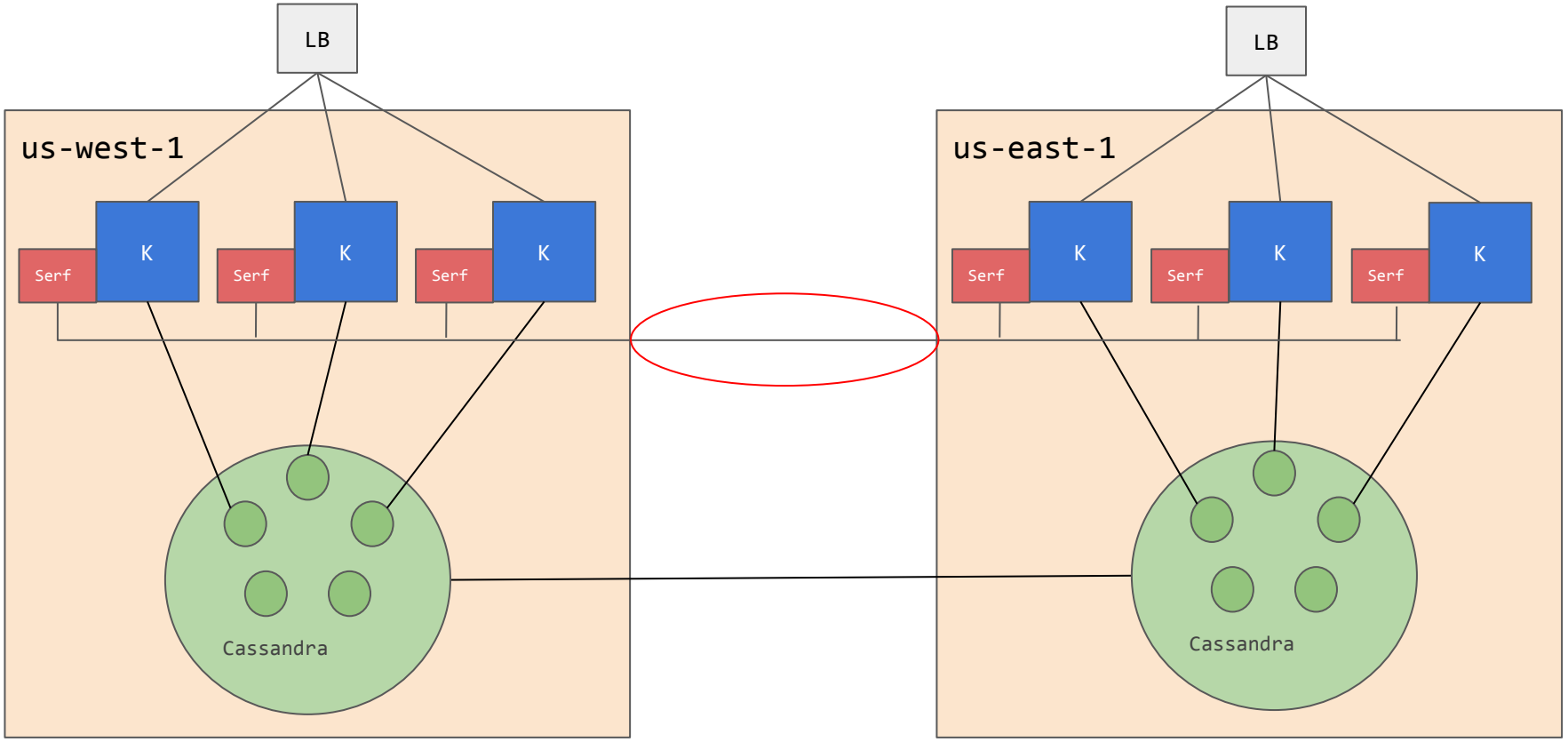
Temporary solution: **Serf** (https://www.serf.io/)

```
~ $ kong start --vv
...
2017/05/22 14:30:13 [debug] found 'serf' executable in $PATH
2017/05/22 14:30:13 [debug] starting serf agent: nohup serf agent -profile 'wan' -bind '0.0.0.0:7946' -log-level 'err' -rpc-addr
'127.0.0.1:7373' -event-handler
'member-join,member-leave,member-failed,member-update,member-reap,user:kong=/usr/local/kong/serf/serf_event.sh' -node
'dev_0.0.0.0:7946_470b634076b94e2aa6a0bb7bce7673f7' > /usr/local/kong/logs/serf.log 2>&1 & echo $! > /usr/local/kong/pids/serf.pid
2017/05/22 14:30:14 [verbose] serf agent started
2017/05/22 14:30:14 [verbose] auto-joining serf cluster
2017/05/22 14:30:14 [verbose] registering serf node in datastore
2017/05/22 14:30:14 [verbose] cluster joined and node registered in datastore
```

- Provides inter-nodes gossiping
- New dependency ☹
- Additional ports and firewall rules ☹
- Additional cross-datacenter communication ☹

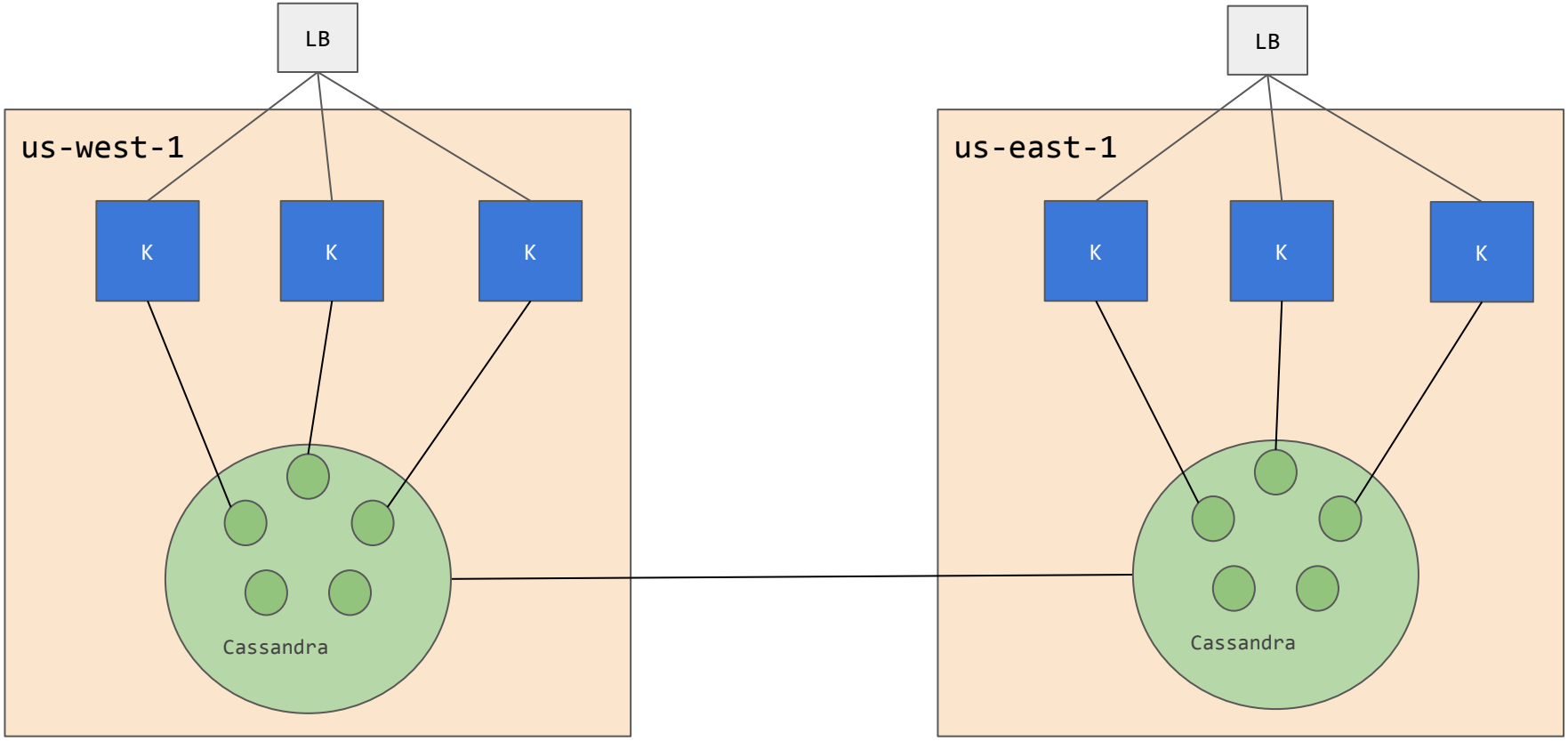
Clustering

The overhead of the OpenResty + Serf pattern



Clustering

Our desired high-level view of a Kong cluster



Clustering

We removed our Serf dependency by introducing a pub/sub mechanism between OpenResty and PostgreSQL/Cassandra.

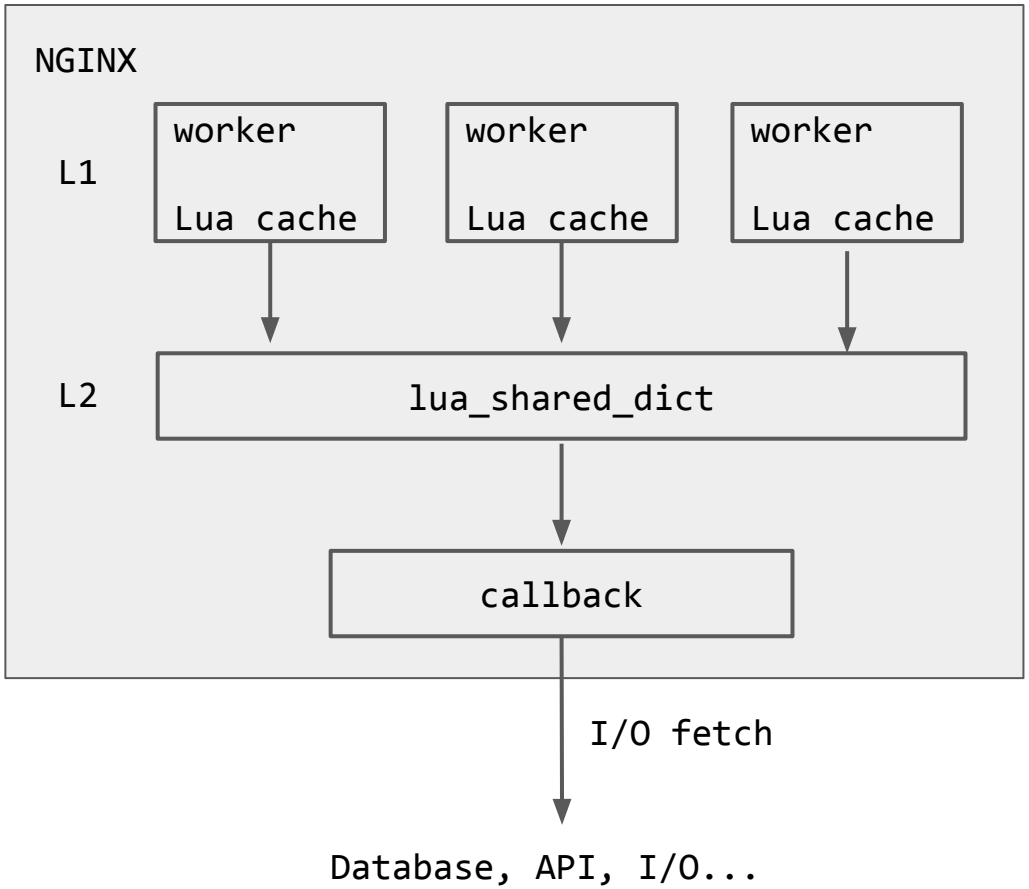
- Workers write in a “channel” with `broadcast(channel, data, nbf)`
- Other workers subscribe to it with `subscribe(channel, callback)`
- A combination of `ngx.timer` and `lua-resty-lock` allows for a safe polling mechanism
- Introducing new configuration properties: `db_update_frequency/db_update_propagation/db_cache_ttl`
- Upon invalidation event received: `ngx.shared.cache:delete(key)`

https://github.com/Kong/kong/blob/master/kong/cluster_events.lua

Clustering

lua-resty-mlcache

- Multi-level caching (`lua-resty-cache` + `lua_shared_dict`) with LRU eviction
- TTL and negative (miss) TTL
- Built-in mutex mechanism with `lua-resty-lock` to prevent dogpile effects
- Multiple instances supported



https://github.com/thibaultcha/lua-resty-mlcache

DNS Resolution

- Proxy + Lua middleware → **OpenResty**
- A command line interface (CLI) → ~~**PUC-Lua 5.1**~~ **OpenResty**
- DNS resolution & Load balancing → ~~**dnsmasq**~~ **OpenResty**
- Clustering of nodes → ~~**Serf**~~ **OpenResty** 
- Generate UUIDs → ~~**libuuid**~~ **OpenResty**
- Database → **Cassandra**

Inter-workers communication

Inter-workers communication

Invalidating Lua-land cache (`lua-resty-lru`) requires inter-workers communication, a long-requested OpenResty feature.

lua-resty-worker-events - https://github.com/Kong/lua-resty-worker-events
Author: Thijs Schreijer (@tieske)

- Pub/sub mechanism via `lua_shared_dict`
- Multiple channels
- Automatic polling via `ngx.timer`

Ideally, a binding API for cosockets will one day replace `lua_shared_dict` based solutions!

Conclusion

Conclusion

One by one, we've eliminated all external dependencies. Kong now is a **pure OpenResty application**.

- Proxy + Lua middleware → **OpenResty**
- A command line interface (CLI) → ~~**PUC-Lua 5.1**~~ **OpenResty**
- DNS resolution & Load balancing → ~~**dnsmasq**~~ **OpenResty**
- Clustering of nodes → ~~**Serf**~~ **OpenResty**
- Generate UUIDs → ~~**libuuid**~~ **OpenResty**
- Database → **Cassandra**

Conclusion

We've open sourced several libraries to the OpenResty community!

- https://github.com/Kong/**lua-resty-dns-client**
- https://github.com/Kong/**lua-resty-worker-events**
- https://github.com/thibaultcha/**lua-resty-mlcache**
- https://github.com/thibaultcha/**lua-resty-jit-uuid**
- https://github.com/thibaultcha/**lua-resty-busted**
- And more!
- https://github.com/thibaultcha/**lua-cassandra** (see my LuaConf 2017 talk in Rio de Janeiro: https://youtu.be/o8mbOT3Veeo)
- https://github.com/thibaultcha/**lua-resty-socket**

Conclusion

Wishlist:

- Support for cosockets in `init_by_lua`
- Native inter-workers communication
- Support for SSL client certificates for cosockets (https://github.com/openresty/lua-nginx-module/pull/997)
- Support for `ngx.rawlog()` API (https://github.com/openresty/lua-resty-core/pull/128)
- Support for `/etc/hosts` parsing (https://github.com/openresty/openresty/pull/247)

Thank you!

Questions?

Bonus

lua-cjson empty array encoding

```lua
local cjson = require "cjson"
local rows = {} -- fetch from db

-- before
cjson.encode({ data = rows })

--[[
{
    "data":{}
}
--]]

-- now
setmetatable(rows, cjson.empty_array_mt)
cjson.encode({ data = rows })

--[[
{
    "data":[]
}
--]]
```

https://github.com/openresty/lua-cjson/pull/6

lua-resty-socket

https://github.com/thibaultcha/lua-resty-socket

Compatibility module for cosocket/LuaSocket.

- Automatic fallback to LuaSocket in non-OpenResty, or non-supported OpenResty contexts (e.g. `init_by_lua`)
- Support for SSL via LuaSec fallback
- Full interoperability

```lua
local socket = require "resty.socket"
local sock = socket.tcp()

sock:settimeout(1000) ---> 1000ms converted to 1s if LuaSocket
sock:getreusedtimes(...) ---> 0 if LuaSocket
sock:setkeepalive(...) ---> calls close() if LuaSocket
sock:sslhandshake(...) ---> LuaSec dependency if LuaSocket
```

Friendly error logs with ngx.log

```
local errlog = require "ngx.errlog"
errlog.rawlog(ngx.NOTICE, "hello world")

2017/07/09 19:36:25 [notice] 25932#0: *1 [lua] content_by_lua(nginx.conf:51):5:
 hello world, client: 127.0.0.1, server: localhost, request: "GET /log
HTTP/1.1", host: "localhost"
```

- Raw output to `error_log`
- Customizable stacktrace level report

https://github.com/openresty/lua-resty-core/pull/128

Thank you!

Questions?