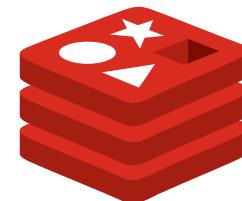




Embedding Lua scripts for Redis in C
& other lessons learned



redis



<https://nchan.slact.net>



NCHAN: What is it?

- Third Party Nginx Module
- Buffering Pub/Sub server for web clients
- Subscribe via Long-Polling, Websocket, EventSource / SSE, Chunked-Transfer, multipart/mixed
- Publish via HTTP and Websocket
- Storage in-memory & on-disk, or in **Redis**.
- Uses channels to coordinate publishers and subscribers.



Some Features

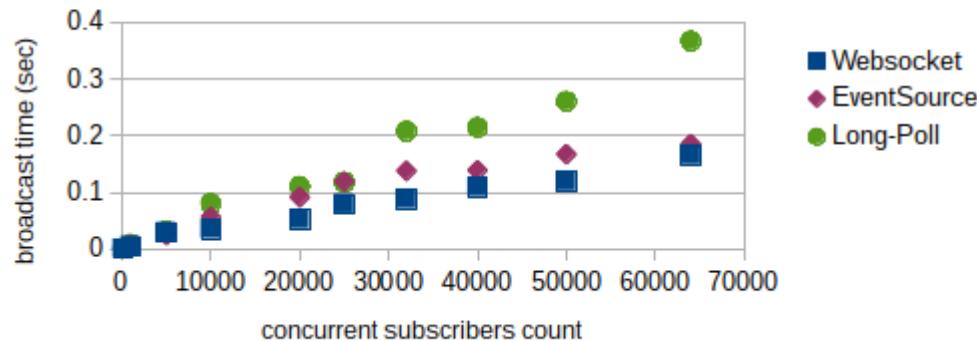
- Channel ID derived from publisher / subscriber request.
- Per-channel configurable message expiration.
- Multiplexed subscriptions.
- Access controls based on channel metadata or upstream application response.
- Resumable subscriber connections with no-loss, no-repetition delivery guarantees.



Scalability

Total Subscriber Response Times Benchmark
(as measured from within Nchan)

Tests run on a dual-CPU Xeon L5630 with 8 HT cores, using 8 Nginx workers.



- It's pretty fast...
 - 30K websocket responses per 100ms
 - Handles connections as well as Nginx, because it *is* Nginx.
- Scales vertically with more CPU and RAM bandwidth
- Scales horizontally by sharding subscribers...
...or by using **Redis**...

An aside on Nginx configs

```
#very basic nchan config
worker_processes 5;
http {
    server {
        listen 80;

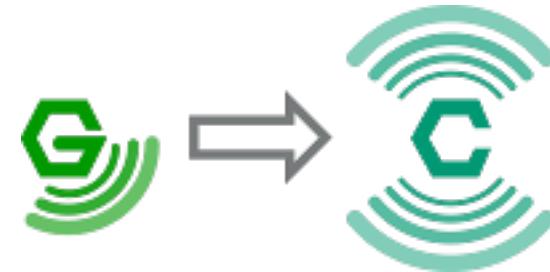
        location ~ /sub/(.+)$ {
            nchan_subscriber;
            nchan_channel_id $1;
        }

        location ~ /pub/(.+)$ {
            nchan_publisher;
            nchan_channel_id $1;
        }
    }
}
```

```
curl -v http://localhost/sub/foo
*   Trying 127.0.0.1...
*   Connected to localhost (127.0.0.1) port 80 (#0)
> GET /sub/broadcast/foo HTTP/1.1
> Host: localhost:80
> User-Agent: curl/7.48.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.9.15
< Date: Mon, 25 Apr 2016 22:21:07 GMT
< Content-Type: application/x-www-form-urlencoded
< Content-Length: 5
< Last-Modified: Mon, 25 Apr 2016 22:21:07 GMT
< Connection: keep-alive
< Etag: 0
< Vary: If-None-Match, If-Modified-Since
<
* Connection #0 to host localhost left intact
hi%
```

```
curl -X POST http://localhost:8082/pub/foo -d hi
queued messages: 1
last requested: 0 sec. ago
active subscribers: 1
last message id: 1461622867:0
```

Some history...



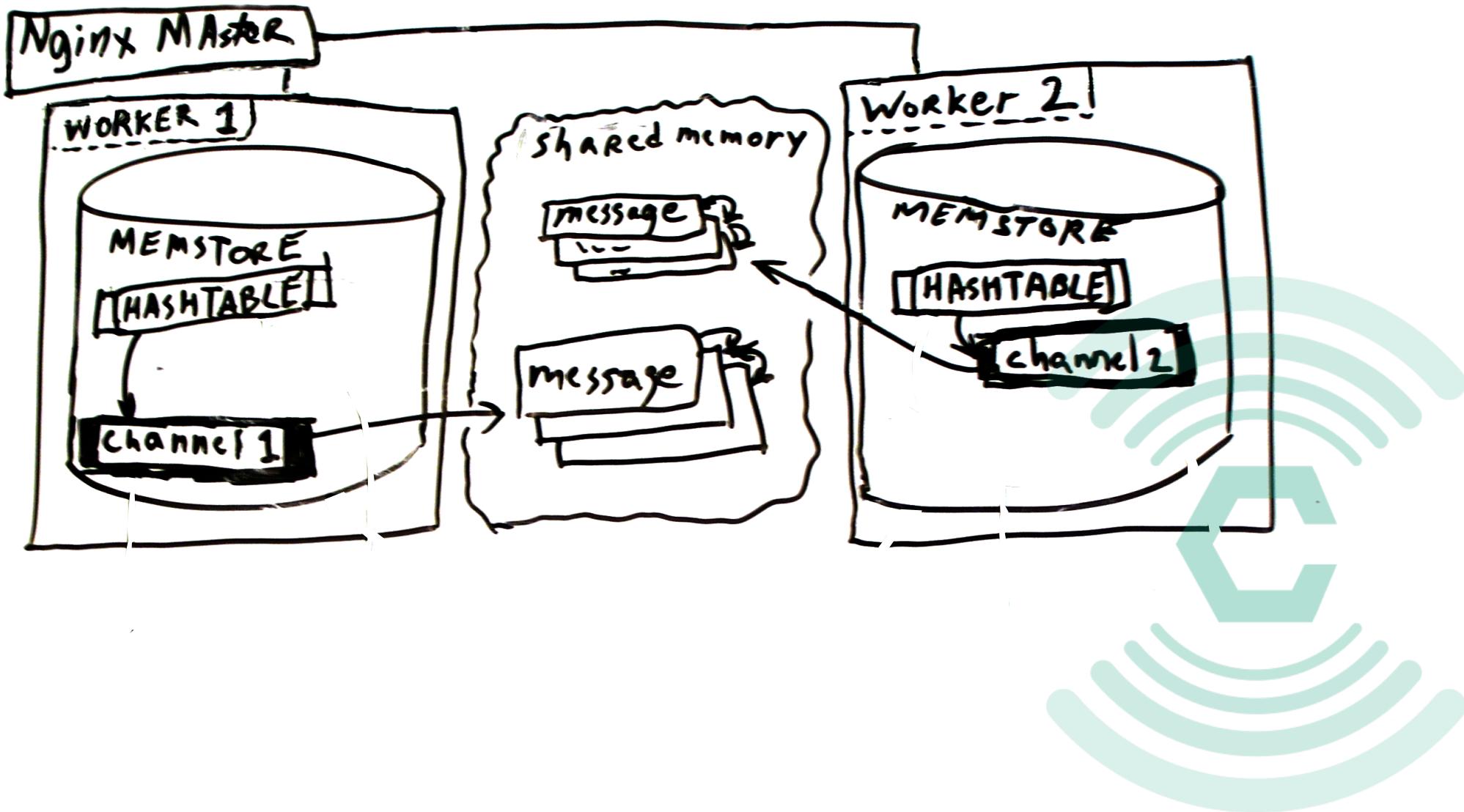
nginx_http_push_module (2009-2011)

- Longpoll-only
- Storage was in shared memory, using an (ugly) global mutex
- Gradually refactored in the course of the last 2 years.
- Rebuilt into Nchan in 2015

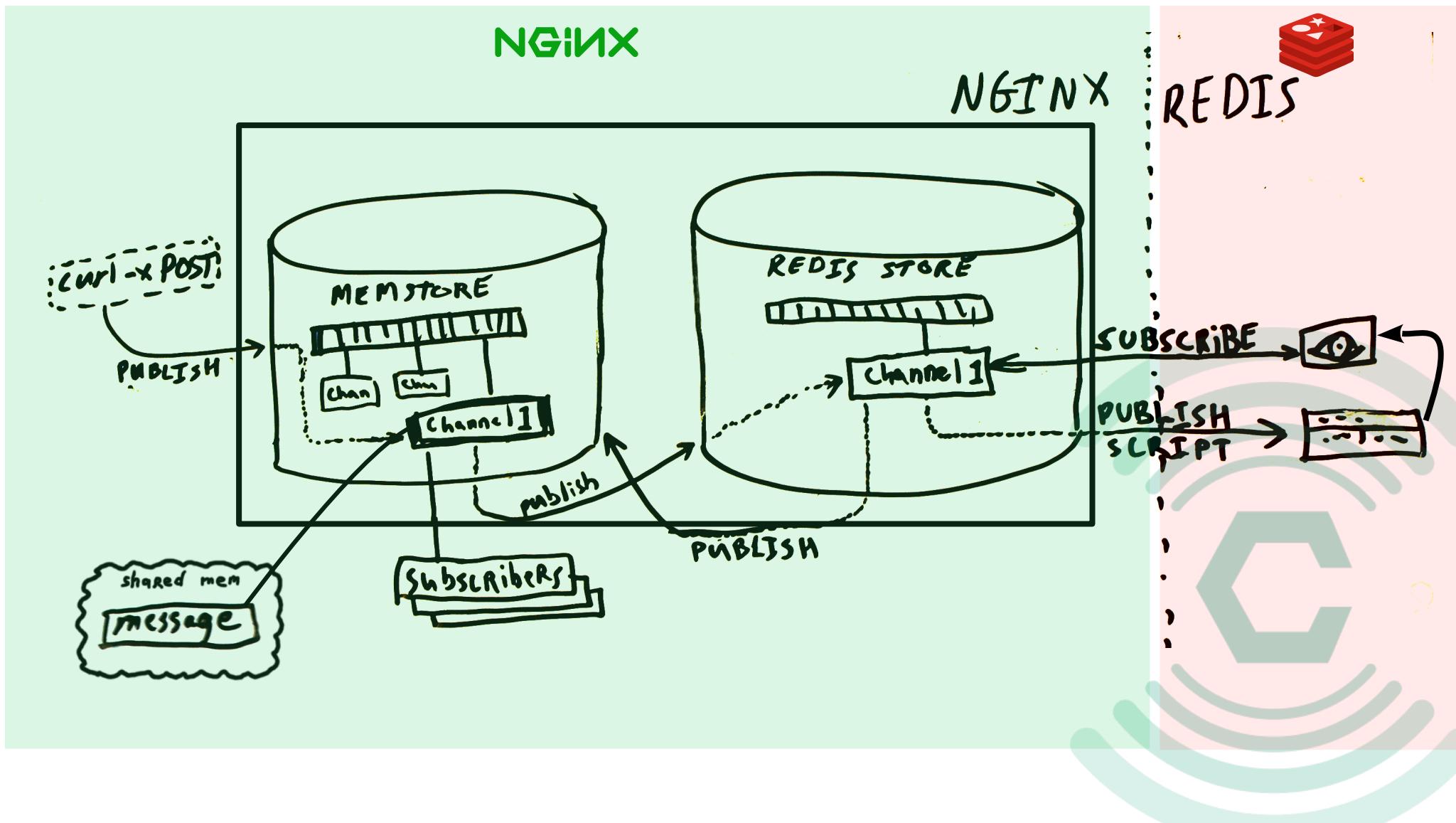
src	7 items
nginx-source	8 items
nginx	0 B
ngx_http_push_module_ipc.c	9.8 KiB
ngx_http_push_module_setup.c	14.5 KiB
ngx_http_push_module.c	45.2 KiB
ngx_http_push_module.h	10.1 KiB
ngx_http_push_rbtree_util.c	7.7 KiB



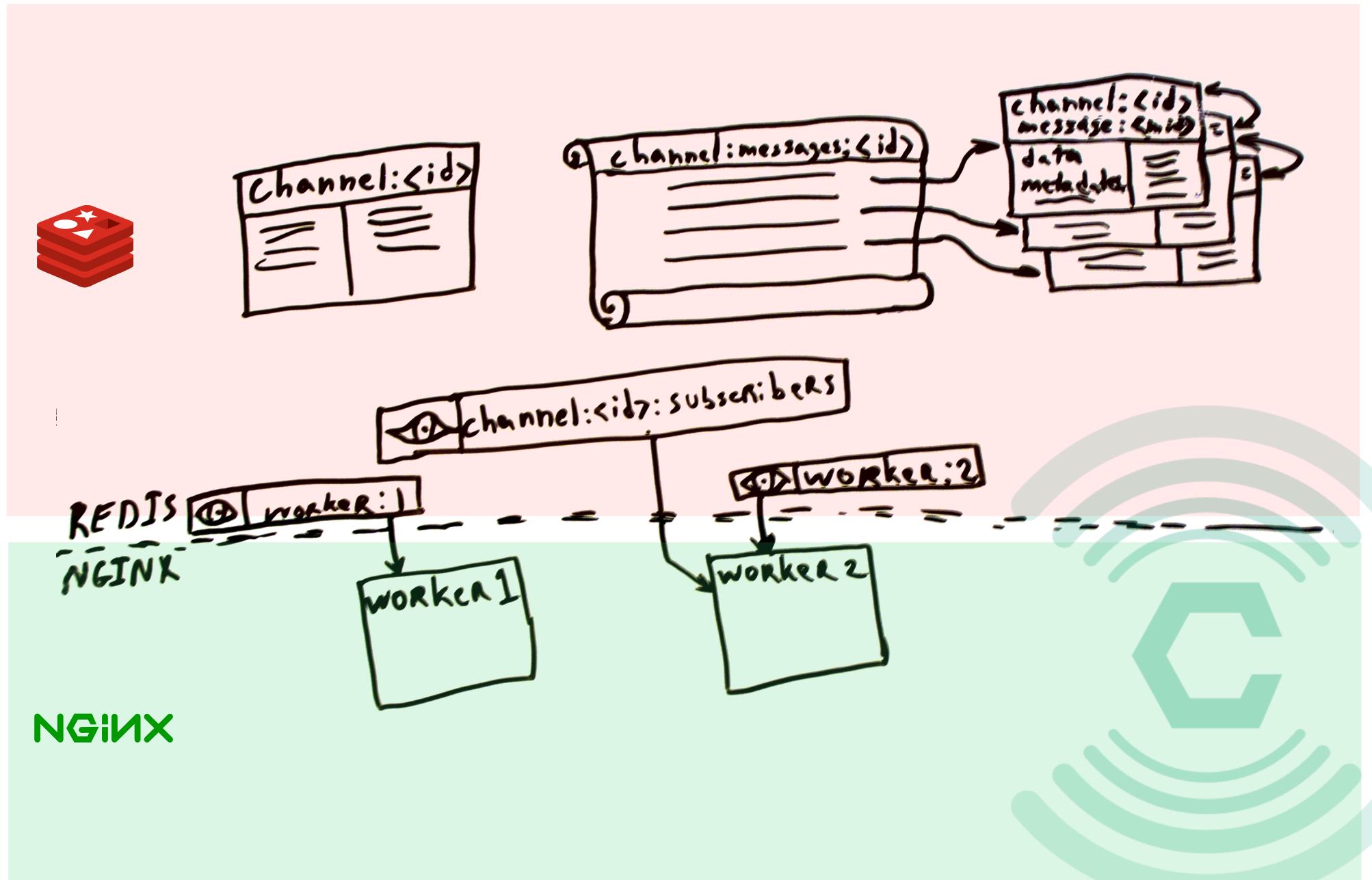
Architecture Overview: Memory Store



Architecture Overview: Memory & Redis Store



Redis Data Architecture



hi redis

- Nginx uses a custom event loop
- hiredis has adapters for all the standard event libraries, but not for Nginx
- Fortunately, there are nginx-hiredis adapters out there already:
 - https://github.com/wandenberg/redis_nginx_adapter
 - https://github.com/alexly/redis_nginx_module
- Each Nginx worker uses 3 connections to redis:
 - 1 asynchronous, for running scripts
 - 1 asynchronous, for PUBSUB
 - 1 synchronous, for use when shutting down



Lua Scripts

- Great for cutting down roundtrips, but...
- No easy way to call scripts from within scripts.
- No way to share functions.
- No way to reuse code.



The Two Forms of Redis Scripts:

1. The All-in-One Script

`geo.lua: EVALSHA <hash> <keys> <COMMAND> <args...>`

(<https://github.com/RedisLabs/geo.lua>)

- Necessary for function reuse.
- A bit difficult to write and debug.



The Two Forms of Redis Scripts:

2. Split Scripts

- One script per ‘command’
- Useful when little functional overlap between ‘commands’
- (Arguably) easier to write and debug.
- DRYDRY: Prepare to repeat yourself.



Gluing it Together with Scripts

- Scripts can be tested with a high-level language before embedding
- How to embed Lua scripts in C?
- How to load scripts?
- How to do error checking?

>--	nginx-source	7 items
<	store	9 items
>	memory	2 items
<	redis	8 items
>	scripts	10 items
--	AI delete.lua	1.7 KiB
--	AI find_channel.lua	768 B
--	Gemfile	58 B
--	Gemfile.Lock	103 B
--	AI get_message.lua	4.8 KiB
--	AI publish_status.lua	945 B
--	AI publish.lua	5.8 KiB
--	AI subscriber_register.lua	1.6 KiB
--	AI subscriber_unregister.lua	1.4 KiB
--	testscripts.rb	6.3 KiB
--	genlua.rb	1.5 KiB
--	redis_lua_commands.h	23.4 KiB
--	redis_nginx_adapter.c	7.0 KiB
--	redis_nginx_adapter.h	574 B
--	store.c	52.6 KiB
--	store.h	2.1 KiB
--	uthash.h	60.0 KiB
--	ngx_http_push_module_ipc.c	5.9 KiB
--	ngx_http_push_module_ipc.h	314 B
--	ngx_http_push_store.h	1.1 KiB

testscripts.rb :

testing lua with ruby

```
#!/usr/bin/ruby
require 'digest/sha1'
require "redis"
require 'open3'
require 'minitest'
require 'minitest/reporters'
require "minitest/autorun"
require 'securerandom'

REDIS_HOST="127.0.0.1"
REDIS_PORT=8537
REDIS_DB=1

class PubSubTest < Minitest::Test
  @@redis=nil
  @@scripts= {}
  @@files=  {}
  @@scripts= {}
  @@hashes=  {}

  def self.test_order; :alpha; end

  def self.luac
    if @@scripts
      @@scripts.each do |name, script|
        Open3.popen2e('luac', "-p", @@files[name]) do |stdin,
        stdouterr, process|
          raise stdouterr.read unless process.value.success?
        end
      end
    else
      raise "scripts not loaded yet"
    end
  end
end
```

```
def self.loadsscripts
  @@scripts.each do |name, script|
    begin
      h=@@redis.script :load, script
      @@hashes[name]=h
    rescue Redis::CommandError => e
      e.message.gsub!(/:(\s+)(user_script):(\d+)/, ":"+
      \n#{name}.lua:\2:")
      def e.backtrace; []; end
      raise e
    end
  end
end

def setup
  unless @@redis
    @@redis=Redis.new(:host => REDIS_HOST, :port =>
REDIS_PORT, :db => REDIS_DB)

Dir[ "#{File.dirname(__FILE__)}/*.lua" ].each do |f|
  scriptname=File.basename(f, ".lua").to_sym
  @@scripts[scriptname]=IO.read f
  @@files[scriptname]=f
end
self.class.luac
self.class.loadsscripts
end

def redis; @@redis; end
def hashes; @@hashes; end
#here be tests
end
```

testscripts.rb : minitest is pretty nice



Embedding

- Import scripts as C strings:

```
--input: keys: [], values: [ channel_id ]
--output: channel_hash {ttl, time_last_seen, subscribers,
messages} or nil
-- finds and return the info hash of a channel, or nil of
channel not found
local id = ARGV[1]
local key_channel='channel:'..id

redis.call('echo', ' ##### FIND_CHANNEL ##### ')

if redis.call('EXISTS', key_channel) ~= 0 then
  local ch = redis.call('hmget', key_channel, 'ttl',
'time_last_seen', 'subscribers', 'fake_subscribers')
  if(ch[4]) then
    --replace subscribers count with fake_subscribers
    ch[3]=ch[4]
    table.remove(ch, 4)
  end
  for i = 1, #ch do
    ch[i]=tonumber(ch[i]) or 0
  end
  table.insert(ch, redis.call('llen',
"channel:messages:"..id))
  return ch
else
  return nil
end
```



```
"--input: keys: [], values: [ channel_id ]\n"
"--output: channel_hash {ttl, time_last_seen, subscribers,
messages} or nil\n"
"-- finds and return the info hash of a channel, or nil of
channel not found\n"
"local id = ARGV[1]\n"
"local key_channel='channel:'..id\n"
"\n"
"redis.call('echo', ' ##### FIND_CHANNEL ##### ')\\n"
"\n"
"if redis.call('EXISTS', key_channel) ~= 0 then\\n"
"  local ch = redis.call('hmget', key_channel, 'ttl',
'time_last_seen', 'subscribers', 'fake_subscribers')\\n"
"  if(ch[4]) then\\n"
"    --replace subscribers count with fake_subscribers\\n"
"    ch[3]=ch[4]\\n"
"    table.remove(ch, 4)\\n"
"  end\\n"
"  for i = 1, #ch do\\n"
"    ch[i]=tonumber(ch[i]) or 0\\n"
"  end\\n"
"  table.insert(ch,
redis.call('llen', \\\"channel:messages:\\\"..id))\\n"
"  return ch\\n"
"else\\n"
"  return nil\\n"
"end\\n"
```

(must have the same hash)

Error Handling?

```
127.0.0.1:6379> evalsha "f738535cb8488ef039e747d144a5634b8408c7c5" 0
```

```
(error) ERR Error running script (call to f_f738535cb8488ef039e747d144a5634b8408c7c5):  
@enable_strict_lua:15: user_script:1: Script attempted to access unexisting global variable  
'foobar'
```

Script hash is known, but no script name...

- Let's use it to lookup the script name by hash!
- So we need to embed the script *name* and *hash* along with the source...

(The price of having a simple server is offloading complexity to the client)



genlua.rb

input: script files

output: C structs with script src, hashes, and names

example/

— delete.lua

```
-- deletes first key
redis.call('del', KEYS[1])
```



— echo.lua

```
-- echoes the first argument
redis.call('echo', ARG[1])
```



```
typedef struct {
    //deletes first key
    char *delete;

    //echoes the first argument
    char *echo;

} redis_lua_scripts_t;

static redis_lua_scripts_t redis_lua_hashes = {
    "c6929c34f10b0fe8eaba42cde275652f32904e03",
    "8f8f934c6049ab4d6337cfa53976893417b268bc"
};

static redis_lua_scripts_t redis_lua_script_names = {
    "delete",
    "echo",
};

static redis_lua_scripts_t redis_lua_scripts = {
    //delete
    "--deletes first key\n"
    "redis.call('del', KEYS[1])\n",

    //echo
    "--echoes the first argument\n"
    "redis.call('echo', ARG[1])\n"
};
```

Introducing hsss: Hash-Safe Script Splinterer

- Ruby gem for Lua Script and hash embedding into C source.

```
> gem install hsss
```

```
> hsss --help
Hash-Safe Script Splinterer, a Lua Script and hash embedder into C source.
Good for putting Redis Lua scripts in your C headers.
```

```
Usage: hsss [options] files
```

--format [split whole]	Output as separate or a single struct
--struct [redis_lua_scripts_t]	C struct name
--row-struct [redis_lua_script_t]	Hash+name+script struct for 'whole' format.
--scripts [redis_lua_scripts]	Scripts variable (split or whole format)
--hashes [redis_lua_hashes]	Hashes variable (split format)
--names [redis_lua_script_names]	Script names variable (split format)
--count [redis_lua_scripts_count]	integer script count variable
--no-count	Omit script count variable
--each-macro [REDIS LUA SCRIPTS EACH]	Iterator macro
--no-each	Omit the iterator macro
--no-parse	Skip using luac to check script syntax
--prefix PREFIX	Prefix default names with this

hsss features

- runs luac -p on each script to validate syntax
 - (almost) doesn't matter if using lua 5.1, 5.2 or 5.3
- Customizable variable and struct names
- Iterator macro
- Two output formats: *whole* and *split*



hsss output

```
> hsss --format whole example/*.lua
```

```
// don't edit this please, it was auto-generated by hsss
// https://github.com/slact/hsss
```

```
typedef struct {
    char *name;
    char *hash;
    char *script;
} redis_lua_script_t;

typedef struct {
    //deletes first key
    redis_lua_script_t delete;

    //echoes the first argument
    redis_lua_script_t echo;

} redis_lua_scripts_t;

static redis_lua_scripts_t redis_lua_scripts = {
    {"delete", "c6929c34f10b0fe8eaba42cde275652f32904e03",
     "--deletes first key\n"
     "redis.call('del', KEYS[1])\n"},

    {"echo", "8f8f934c6049ab4d6337cfa53976893417b268bc",
     "--echoes the first argument\n"
     "redis.call('echo', ARG[1])\n"}
};

const int redis_lua_scripts_count=2;
```

```
#define REDIS LUA_SCRIPTS_EACH(script) \
for((script)=(redis_lua_script_t *)&redis_lua_scripts; (script) < \
(redis_lua_script_t *)(&redis_lua_scripts + 1); (script)++)
```

```
> hsss --format split example/*.lua
```

```
// don't edit this please, it was auto-generated by hsss
// https://github.com/slact/hsss
```

```
typedef struct {
    //deletes first key
    char *delete;

    //echoes the first argument
    char *echo;

} redis_lua_scripts_t;

static redis_lua_scripts_t redis_lua_hashes = {
    "c6929c34f10b0fe8eaba42cde275652f32904e03",
    "8f8f934c6049ab4d6337cfa53976893417b268bc"
};

static redis_lua_scripts_t redis_lua_script_names = {
    "delete",
    "echo",
};

static redis_lua_scripts_t redis_lua_scripts = {
    //delete
    //--deletes first key\n"
    "redis.call('del', KEYS[1])\n",

    //echo
    //--echoes the first argument\n"
    "redis.call('echo', ARG[1])\n"
};

const int redis_lua_scripts_count=2;
```

```
#define REDIS LUA_SCRIPTS_EACH(script_src, script_name, script_hash) \
for((script_src)=(char **)&redis_lua_scripts, (script_hash)=(char \
**)redis_lua_hashes, (script_name)=(char **)&redis_lua_script_names; \
(script_src) < (char **)&redis_lua_scripts + 1; (script_src)++, \
(script_hash)++, (script_name)++)
```

The rest of the talk (mostly) assumes the ‘whole’ format.

LOADing scripts from hsss output

```
static void redisLoadScriptCallback(redisAsyncContext *c, void *r, void *privdata) {
    redis_lua_script_t *script = privdata;

    redisReply *reply = r;
    if (reply == NULL) return;
    switch(reply->type) {
        case REDIS_REPLY_ERROR:
            ngx_log_error(NGX_LOG_ERR, ngx_cycle->log, 0, "nchan: Failed loading redis lua script %s : %s", script->name, reply->str);
            break;
        case REDIS_REPLY_STRING:
            if(ngx_strncmp(reply->str, script->hash, REDIS LUA_HASH_LENGTH)!=0) {
                ngx_log_error(NGX_LOG_ERR, ngx_cycle->log, 0, "nchan Redis lua script %s has unexpected hash %s (expected %s)", script->name, reply->str, script->hash);
            }
            break;
    }
}

static void redisInitScripts(redisAsyncContext *c){
    redis_lua_script_t *script;

    REDIS LUA_SCRIPTS_EACH(script) {
        redisAsyncCommand(c, &redisLoadScriptCallback, script, "SCRIPT LOAD %s", script->script);
    }
}
```

EVALSHAing scripts

--format whole

```
redisAsyncCommand(asyncContext, asyncCommandCallback, privdata, "EVALSHA  
%s 1 foo", redis_lua_scripts.get_message_from_key.hash);
```

--format split

```
redisAsyncCommand(asyncContext, asyncCommandCallback, privdata, "EVALSHA  
%s 1 foo", redis_lua_script_hashes.get_message_from_key);
```

```
static void asyncCommandCallback(redisAsyncContext *c, void *r,  
void *privdata) {  
    redisReply *reply = (redisReply *)r;  
    if(reply != NULL && reply->type == REDIS_REPLY_ERROR) {  
        redisCheckErrorCallback(c, r, privdata);  
    } else {  
        //script response handling logic  
    }  
}
```

Async error handler

```
(error) ERR Error running script (call to f_738535cb8488ef039e747d144a5634b8408c7c5):  
@enable_strict_lua:15: user_script:1: Script attempted to access unexisting global variable  
'foobar'
```

```
static void redisCheckErrorCallback(redisAsyncContext *c, void *r, void *privdata) {  
    static const ngx_str_t script_error_start= ngx_string("ERR Error running script (call to f_");  
    redisReply *reply = (redisReply *)r;  
    if(reply != NULL && reply->type == REDIS_REPLY_ERROR) {  
        if(ngx_strncmp(reply->str, script_error_start.data, script_error_start.len) == 0 && (unsigned ) reply->len > script_error_start.len + REDIS_LUA_HASH_LENGTH) {  
            char *hash = &reply->str[script_error_start.len];  
            /* This is the part highlighted in blue */  
            REDIS_LUA_SCRIPTS_EACH(script) {  
                if (ngx_strncmp(script->hash, hash, REDIS_LUA_HASH_LENGTH)==0) {  
                    ngx_log_error(NGX_LOG_ERR, ngx_cycle->log, 0, "REDIS_SCRIPT_ERROR: %s :%s", script->name,  
&reply->str[script_error_start.len + REDIS_LUA_HASH_LENGTH + 2]);  
                    return;  
                }  
            }  
            ngx_log_error(NGX_LOG_ERR, ngx_cycle->log, 0, "REDIS_SCRIPT_ERROR: (unknown): %s", reply->str);  
        }  
        else {  
            ngx_log_error(NGX_LOG_ERR, ngx_cycle->log, 0, "REDIS_REPLY_ERROR: %s", reply->str);  
        }  
    }  
}
```

hsss in your build script

Shell script
(actual snippet from nchan)

```
#!/bin/zsh

#...
rdstore_dir=${MY_PATH}/../src/store/redis
bundle exec hsss \
  --format whole \
  --no-count \
  ${rdstore_dir}/scripts/*.lua > ${rdstore_dir}/redis_lua_commands.h
if ! [ $? -eq 0 ]; then
  echo "failed generating redis lua scripts";
  exit 1
fi
#...
```

Makefile

```
IDIR=./
CC=gcc
CFLAGS=-I$(IDIR)
ODIR=./
LIBS=-lhiredis
_DEPS = scripts.h
DEPS = $(patsubst %,$(IDIR)/%,${_DEPS})
_OBJ = embeddy.o
OBJ = $(patsubst %,$(ODIR)/%,${_OBJ})

$(ODIR)/%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

embeddy: scripts.h $(OBJ)
    $(CC) -o $@ $^ $(CFLAGS) $(LIBS)

scripts.h: *.lua
    bundle exec hsss --format whole \
      ./*.lua > scripts.h

.PHONY: clean

clean:
    rm -f $(ODIR)/*.* ~ core $(INCDIR)/*~
```

hsss in Redis, why not?

src/scripting.c

```
void scriptingEnableGlobalsProtection(lua_State *lua) {
    char *s[32];
    sds code = sdseempty();
    int j = 0;

    /* strict.lua from:
    http://metalua.luaforge.net/src/lib/strict.lua.html.
     * Modified to be adapted to Redis. */
    s[j++] = "local dbg=debug\n";
    s[j++] = "local mt = {}\n";
    s[j++] = "setmetatable(_G, mt)\n";
    s[j++] = "mt.__newindex = function (t, n, v)\n";
    s[j++] = "    if dbg.getinfo(2) then\n";
    s[j++] = "        local w = dbg.getinfo(2, \"S\").what\n";
    s[j++] = "        if w ~= \"main\" and w ~= \"C\" then\n";
    s[j++] = "            error(\"Script attempted to create global
variable '\"..tostring(n)..\"'\", 2)\n";
    s[j++] = "        end\n";
    s[j++] = "    end\n";
    s[j++] = "    rawset(t, n, v)\n";
    s[j++] = "end\n";
    s[j++] = "mt.__index = function (t, n)\n";
    s[j++] = "    if dbg.getinfo(2) and
dbg.getinfo(2, \"S\").what == \"C\" then\n";
    s[j++] = "        error(\"Script attempted to access unexisting
global variable '\"..tostring(n)..\"'\", 2)\n";
    s[j++] = "    end\n";
    s[j++] = "    return rawget(t, n)\n";
    s[j++] = "end\n";
    s[j++] = "debug = nil\n";
    s[j++] = NULL;

    for (j = 0; s[j] != NULL; j++) code =
sdscatlen(code, s[j], strlen(s[j]));

    luaL_loadbuffer(lua, code, sdslen(code), "@enable_strict_lua");
    lua_pcall(lua, 0, 0, 0);
    sdsfree(code);
}
```

```
void scriptingInit(int setup) {
    /* [...] */
    /* Add a helper function that we use to sort the multi bulk output of non
     * deterministic commands, when containing 'false' elements. */
{
    char *compare_func =
        "function __redis__compare_helper(a,b)\n"
        "    if a == false then a = '' end\n"
        "    if b == false then b = '' end\n"
        "    return a<b\n"
        "end\n";
    luaL_loadbuffer(lua, compare_func, strlen(compare_func), "@cmp_func_def");
    lua_pcall(lua, 0, 0, 0);
}
/* Add a helper function we use for pcall error reporting.
 * Note that when the error is in the C function we want to report the
 * information about the caller, that's what makes sense from the point
 * of view of the user debugging a script. */
{
    char *errh_func =
        "local dbg = debug\n"
        "function __redis__err_handler(err)\n"
        "    local i = dbg.getinfo(2, 'nS1')\n"
        "    if i and i.what == 'C' then\n"
        "        i = dbg.getinfo(3, 'nS1')\n"
        "    end\n"
        "    if i then\n"
        "        return i.source .. ':' .. i.currentline ..
        "    else\n"
        "        return err\n"
        "    end\n"
        "end\n";
    luaL_loadbuffer(lua, errh_func, strlen(errh_func), "@err_handler_def");
    lua_pcall(lua, 0, 0, 0);
}
/* [...] */
}
```

src/scripts/

compare_func.lua

```
-- helper function that we use to sort the multi bulk output of non-deterministic commands, when containing 'false' elements
function __redis__compare_helper(a,b)
    if a == false then a = '' end
    if b == false then b = '' end
    return a<b
end
```



enable_strict.lua

```
--strict.lua from: http://metalua.luaforge.net/src/lib/strict.lua.html.
-- Modified to be adapted to Redis.
local dbg=debug
local mt = {}
setmetatable(_G, mt)
mt.__newindex = function (t, n, v)
    if dbg.getinfo(2) then
        local w = dbg.getinfo(2, "S").what
        if w ~= "main" and w ~= "C" then
            error("Script attempted to create global variable '"..tostring(n).."'", 2)
        end
    end
    rawset(t, n, v)
end
mt.__index = function (t, n)
    if dbg.getinfo(2) and dbg.getinfo(2, "S").what ~= "C" then
        error("Script attempted to access unexisting global variable '"..tostring(n).."'", 2)
    end
    return rawget(t, n)
end
debug = nil
```



error_handler_func.lua

```
-- a helper function we use for pcall error reporting. Note that when the error is in the C function we want to report the
-- information about the caller, that's what makes sense from the point of view of the user debugging a script.
```

```
local dbg = debug
function __redis__err__handler(err)
    local i = dbg.getinfo(2,'nSl')
    if i and i.what == 'C' then
        i = dbg.getinfo(3,'nSl')
    end
    if i then
        return i.source .. ':' .. i.currentline .. ':' .. err
    else
        return err
    end
end
```



hsss in Redis? Why not.

src/scripting.c

```
#include "luascripts.h"
/* ... */

void scriptingEnableGlobalsProtection(lua_State *lua) {
    luaL_loadbuffer(lua, luaScript.enable_strict, strlen(luaScript.enable_strict), "@enable_strict_lua");
    lua_pcall(lua, 0, 0, 0);
}

void scriptingInit(int setup) {
    /* [...] */

    /* Add a helper function that we use to sort the multi bulk output of non
     * deterministic commands, when containing 'false' elements */
    luaL_loadbuffer(lua, luaScript.compare_func, strlen(luaScript.compare_func), "@cmp_func_def");
    lua_pcall(lua, 0, 0, 0);

    /* Add a helper function we use for pcall error reporting.
     * Note that when the error is in the C function we want to report the
     * information about the caller, that's what makes sense from the point
     * of view of the user debugging a script */
    luaL_loadbuffer(lua, luaScript.error_handler_func, strlen(luaScript.error_handler_func), "@err_handler_def");
    lua_pcall(lua, 0, 0, 0);

    /* [...] */
}
```

hsss in Redis? Why not.

src/Makefile

```
#[...]  
  
luascripts.h: scripts/*.lua  
    hsss --format split \  
        --no-names \  
        --no-hashes \  
        --no-count \  
        --no-each \  
        --struct redisLuaScripts \  
        --scripts luaScript \  
        scripts/*.lua > luascripts.h
```

```
dep: luascripts.h  
    $(REDIS_CC) -MM *.c > Makefile.dep
```

run *make dep* once, then...

hsss in Redis? Why not.

<https://asciinema.org/a/b35ki1280vk04pxj2opo13zvg>



<https://github.net/slact/redis>

hsss: embedding Lua scripts made painless

```
> gem install hsss
```

<https://github.com/slact/hsss>

- Highly customizable output
- Iterator macro
- Seamlessly integrates into build tooling
 - Needed only during development, not source distribution
- Onomatopoeic



Debugging

<https://redislabs.com/blog/5-methods-for-tracing-and-debugging-redis-lua-scripts>

- Use 3.2's *redis-cli --lrb* for debugging with breakpoints
- Otherwise, debug output is your (only) option.
- Many ways to log output, but the most powerful is redis-cli monitor + `redis.call('echo', message)`
- Why MONITOR?...



```
redis-cli monitor
OK
1461473912.214865 [0 127.0.0.1:60118] "SUBSCRIBE" "channel:pubsub:test/foo"
1461473912.214908 [0 127.0.0.1:60116] "EVALSHA" "a717de6d8fc25c375e97d095bb21b7cfdec466b6" "0" "test/foo" "--" "300"
1461473912.214933 [0 lua] "echo" " ##### SUBSCRIBER REGISTER SCRIPT ##### "
1461473912.214947 [0 lua] "HINCRBY" "channel:test/foo" "last_subscriber_id" "1"
1461473912.214957 [0 lua] "hincrby" "channel:test/foo" "subscribers" "1"
1461473912.214963 [0 lua] "ttl" "channel:test/foo"
1461473912.214969 [0 lua] "expire" "channel:subscribers:test/foo" "300"
1461473912.214974 [0 lua] "expire" "channel:messages:test/foo" "300"
1461473912.214978 [0 lua] "expire" "channel:test/foo" "300"
1461473912.215015 [0 127.0.0.1:60116] "EVALSHA" "71c1046fd6c7500ba588857fe6fe93480f0b21d6" "0" "test/foo" "0" "0" "FILO"
1461473912.215027 [0 lua] "echo" " ##### GET_MESSAGE ##### "
1461473912.215032 [0 lua] "HGETALL" "channel:test/foo"
1461473912.215043 [0 lua] "lindex" "channel:messages:test/foo" "-1"
1461473912.315520 [0 127.0.0.1:60122] "EVALSHA" "f33207af23f0efab740207e8faf45a29acbb4c0a" "0" "test/foo" "1"
1461473912.315554 [0 lua] "echo" " ##### FAKESUBS ##### "
1461473912.315567 [0 lua] "EXISTS" "channel:test/foo"
1461473912.315578 [0 lua] "HINCRBY" "channel:test/foo" "fake_subscribers" "1"
1461473919.670390 [0 127.0.0.1:60116] "EVALSHA" "eebcaa531438f8ef81c3b8347468d16bcbe5fef9" "0" "test/foo" "1461473919"
"foobar" "" "" "240" "200"
1461473919.670473 [0 lua] "echo" " ##### PUBLISH ##### "
1461473919.670490 [0 lua] "EXISTS" "channel:test/foo"
1461473919.670499 [0 lua] "HGETALL" "channel:test/foo"
1461473919.670525 [0 lua] "echo" "channel present"
1461473919.670542 [0 lua] "echo" "channel current_message absent"
1461473919.670559 [0 lua] "EXISTS" "channel:msg:1461473919:0:test/foo"
[...]
1461473919.670634 [0 lua] "echo" "channel.max_stored_messages was not set, but is now 200"
1461473919.670666 [0 lua] "HMSET" "channel:msg:1461473919:0:test/foo" "content_type" "" "id" "1461473919:0" "data" "foobar"
"prev_tag" "0" "time" "1461473919" "eventsource_event" "" "prev_time" "0" "ttl" "240" "tag" "0"
[...]
1461473919.670829 [0 lua] "echo" "Stored message with id 1461473919:0 => foobar"
1461473919.670840 [0 lua] "PUBSUB" "NUMSUB" "channel:pubsub:test/foo"
1461473919.670858 [0 lua] "PUBLISH" "channel:pubsub:test/foo"
"\x99\x a3msg\xcc\xf0\xceW\x1cR\x7f\x00\x00\x00\x00\x a6foobar\x a0\x a0"
1461473919.670874 [0 lua] "llen" "channel:messages:test/foo"
1461473919.670891 [0 lua] "echo" "channel test/foo ttl: 240, subscribers: 1(fake: 1), messages: 1"
1461473919.671141 [0 127.0.0.1:60116] "EVALSHA" "71c1046fd6c7500ba588857fe6fe93480f0b21d6" "0" "test/foo" "1461473919" "0"
"FILO"
1461473919.671190 [0 lua] "echo" " ##### GET_MESSAGE ##### "
1461473919.671202 [0 lua] "HGETALL" "channel:test/foo"
```

That's why we MONITOR.

- Script parameters, commands executed available “for free”.
- Script names need to be marked **##### CLEARLY AND EMPHATICALLY #####**.
- *redis-cli monitor* output can be piped to grep for more careful real-time analysis.



Echo echo echo echo

- All-purpose debug output

```
local dbg = function(...)
    local arg = {...}
    for i = 1, #arg do
        arg[i]=tostring(arg[i])
    end
    redis.call('echo', table.concat(arg, " "))
end

dbg(' ##### ', "SCRIPT NAME", " ##### ")
dbg(nil, {}, false, {foo=1}, "banana")
```

```
1461476791.535382 [0 lua] "echo" " ##### " SCRIPT NAME " ##### "
1461476791.535399 [0 lua] "echo" "nil {} false table: 0x256bd00 banana"
```

Putting It All Together



subscribe

- Update subscriber count
- Assign subscriber ids (unused for now)
- Manage key expiration

subscriber_register.lua

```
--input: keys: [], values: [channel_id, subscriber_id, active_ttl]
-- 'subscriber_id' can be '-' for new id, or an existing id
-- 'active_ttl' is channel ttl with non-zero subscribers. -1 to persist, >0 ttl in sec
--output: subscriber_id, num_current_subscribers, next_keepalive_time

local id, sub_id, active_ttl, concurrency = ARGV[1], ARGV[2], tonumber(ARGV[3]) or 28, ARGV[4]

--local dbg = function(...) redis.call('echo', table.concat({...})) end

redis.call('echo', ' ##### SUBSCRIBER REGISTER SCRIPT ##### ')

local keys = {
    channel = 'channel'.id,
    messages = 'channel.messages'.id,
    subscribers = 'channel.subscribers'.id
}

local setkeyttl=function(ttl)
    for i,v in pairs(keys) do
        if ttl > 0 then
            redis.call('expire', v, ttl)
        else
            redis.call('persist', v)
        end
    end
end

local random_safe_next_ttl = function(ttl)
    return math.floor(ttl/2 + ttl/2 * math.random())
end

local sub_count
if sub_id == '-' then
    sub_id = tonumber(redis.call('INCRBY', keys.channel, 'last_subscriber_id', 1))
    sub_count=tonumber(redis.call('hincrby', keys.channel, 'subscribers', 1))
else
    sub_count=tonumber(redis.call('hget', keys.channel, 'subscribers'))
end

local next_keepalive
local actual_ttl = tonumber(redis.call('ttl', keys.channel))
if actual_ttl < active_ttl then
    setkeyttl(active_ttl)
    next_keepalive = random_safe_next_ttl(active_ttl)
else
    next_keepalive = random_safe_next_ttl(actual_ttl)
end

return {sub_id, sub_count, next_keepalive}
```

subscriber_unregister.lua

```
--input: keys: [], values: [channel_id, subscriber_id, empty_ttl]
-- 'subscriber_id' is an existing id
-- 'empty_ttl' is channel ttl when without subscribers. 0 to delete immediately, -1 to persist, >0 ttl in sec
--output: subscriber_id, num_current_subscribers

local id, sub_id, empty_ttl = ARGV[1], ARGV[2], tonumber(ARGV[3]) or 28

--local dbg = function(...) redis.call('echo', table.concat({...})) end

redis.call('echo', ' ##### SUBSCRIBER UNREGISTER SCRIPT ##### ')

local keys = {
    channel = 'channel'.id,
    messages = 'channel.messages'.id,
    subscribers = 'channel.subscribers'.id
}

local setkeyttl=function(ttl)
    for i,v in pairs(keys) do
        if ttl > 0 then
            redis.call('expire', v, ttl)
        elseif ttl < 0 then
            redis.call('persist', v)
        else
            redis.call('del', v)
        end
    end
end

local sub_count = 0
if redis.call('EXISTS', keys.channel) ~= 0 then
    sub_count = redis.call('hincrby', keys.channel, 'subscribers', -1)

    if sub_count == 0 and tonumber(redis.call('LEN', keys.messages)) == 0 then
        setkeyttl(empty_ttl)
        sub_count = redis.call('hincrby', keys.channel, 'subscribers', 1)
        if sub_count < 0 then
            return {err='Subscriber count for channel "'..id..'" less than zero: '..sub_count}
        end
        else
            dbg("channel ", id, " already gone")
        end
    end
end

return {sub_id, sub_count}
```

publish.lua

- Store message
- Delete old messages
- Garbage-collect message list
- Update key expiration
- PUBLISH to PUBSUB clients (as msgpack)

```
--input: keys: [], values: [channel_id, time, message, content_type, eventsource_event, msg_ttl, max_msg_buf_size]
--output: message_tag, channel_hash (ttl), time_last_seen, subscribers, messages

local id=ARGV[1]
local time_tonumber=ARGV[2]
local msg=
  id nil
  data: ARGV[3],
  content_type:ARGV[4]
  eventsource_event:ARGV[5],
  ttl: tonumber(ARGV[6]),
  time: time,
  tag: ''
)
if msg_ttl == 0 then
  msg_ttl = 12614400 --4 years
end
local store_at_most_n_messages = tonumber(ARGV[7])
if store_at_most_n_messages == nil or store_at_most_n_messages == '' then
  return (err="Argument 7, max_msg_buf_size, can't be empty")
end
if store_at_most_n_messages == 0 then
  msg.unbuffered = 1
end

local dbg = function(...)
  local arg = {...}
  for i = 1, #arg do
    arg[i]=tostring(arg[i])
  end
  redis.call('echo', table.concat(arg))
end

if type(msg.content_type) == "string" and msg.content_type:find(' ') then
  return (err="Message content-type cannot contain \" \" character")
end

redis.call('echo', ' ##### PUBLISH ##### ')

-- sets all fields for a hash from a dictionary
local hset = function(key, dict)
  if next(dict) == nil then return nil end
  local bulk = {}
  for k, v in pairs(dict) do
    table.insert(bulk, k)
    table.insert(bulk, v)
  end
  return redis.call('HMSET', key, unpack(bulk))
end

local tohash=function(arr)
  if type(arr) == "table" then
    return nil
  end
  local h = {}
  local k=nil
  for v in ipairs(arr) do
    if k == nil then
      k=v
    else
      h[k]=v; k=nil
    end
  end
  return h
end

local key=
{
  time_offset= 'chan:message_time_offset',
  last_message= nil,
  message= 'channel msg %s'.id, --not finished yet
  channel= 'channel'.id,
  messages= 'channel messages'.id,
  subscribers= 'channel subscribers'.id
}

local channel_pubsub = 'channel pubsub'.id

local new_channel
local channel
if redis.call('EXISTS', key.channel) == 0 then
  channel=hset(redis.call('HGETALL', key.channel))
end

if channel==nil then
  --dbg("channel present")
  if channel.current_message ~= nil then
    --dbg("channel.current_message present")
    key.last_message=channel.msg.%s:format(channel.current_message.id)
  else
    --dbg("channel current_message absent")
    key.last_message=nil
  end
  new_channel=false
else
  --dbg("channel missing")
  channel={}
  new_channel=true
  key.last_message=nil
end

--set new message id
if key.last_message then
  local lastmsg = redis.call('HGET', key.last_message, 'time', 'tag')
  local lasttime, lasttag = tonumber(lastmsg[1]), tonumber(lastmsg[2])
  --dbg("New message id: %s, lasttime: %s, lasttag: %s, msg_time: %s, msg_tag: %s", msg.id, lasttime, lasttag, msg.time, msg.tag)
  if lasttime==msg.time then
    msg.tag=lasttag
  end
  msg.prev_time = lasttime
  msg.prev_tag = lasttag
else
  msg.prev_time = 0
  msg.prev_tag = 0
end
msg.id=("%s %s"):format(msg.time, msg.tag)

key.message=key.message:format(msg.id)
if redis.call('EXISTS', key.message) == 0 then
  return (err="Message for channel %s id %s already exists":format(id, msg.id))
end

msg.hset(key.message, msg)
if msg.unbuffered == 1 then
  msg.hset(key.message, msg)
end

--update channel
redis.call('HSET', key.channel, 'current_message', msg.id)
if msg.prev then
  redis.call('HSET', key.channel, 'prev.message', msg.prev)
end
if time then
  redis.call('HSET', key.channel, 'time', time)
end
if not channel.ttl then
  channel.ttl=msg.ttl
  redis.call('HSET', key.channel, 'ttl', channel.ttl)
end

if not channel.max_stored_messages then
  channel.max_stored_messages = store_at_most_n_messages
  redis.call('HSET', key.channel, 'max_stored_messages', store_at_most_n_messages)
  dbg("channel max_stored_messages was not set, but is now %s", store_at_most_n_messages)
end

--write message
hset(key.message, msg)

--check old entries
local oldestmsg=function(list_key, old_fmt)
  local old, oldkey
  local n, del=0, 0
  while true do
    n+=1
    old=redis.call('LINDEX', list_key, -1)
    if not old then
      oldestmsg.old_messages = channel.msg.%s:id()
      return (old.old_messages, old.key)
    end
    local oldkey=old:format(old)
    local old=redis.call('EXISTS', oldkey)
    if err~=then
      return oldkey
    else
      redis.call('RPOP', list_key)
      del+=1
    end
    else
      break
    end
  end
end

local max_stored_msgs = tonumber(redis.call('HGET', key.channel, 'max_stored_messages')) or -1

if max_stored_msgs < 0 then --no limit
  oldestmsg.old_messages = channel.msg.%s:id()
  return (oldestmsg.old_messages, oldestmsg.oldkey)
elseif max_stored_msgs == 0 then
  local stored_messages = tonumber(redis.call('llen', key.messages))
  redis.call('PUSH', key.messages, msg.id)
  if stored_messages > max_stored_msgs then
    local oldmsgid = redis.call('RPOF', key.messages)
    redis.call('DEL', channel.msg.%s:id(), oldmsgid)
  end
  oldestmsg.old_messages = channel.msg.%s:id()
end

--set expiration times for all the things
redis.call('EXPIRE', key.message, msg.ttl)
redis.call('EXPIRE', key.time_offset, channel.ttl)
redis.call('EXPIRE', key.channel.channel, channel.ttl)
redis.call('EXPIRE', key.messages, channel.ttl)
redis.call('EXPIRE', key.subscribers, channel.ttl)

--publish message
local unpacked

if #msg.data < 5+1024 then
  unpacked= (
    msg,
    msg.ttl or 0,
    msg.time,
    tonumber(msg.tag) or 0,
    msg.prev_time or 0,
    msg.prev_tag or 0,
    msg.data or '',
    msg.content_type or '',
    msg.eventsouce_event or ''
  )
else
  unpacked= (
    msgkey,
    msg.time,
    tonumber(msg.tag) or 0,
    key.message
  )
end

local msgpacked

--dbg("Stored message with id %s: %s", msg.id, msg.data)
--now publish to the efficient channel
local channel_pubsub = redis.call('PUBSUB', 'NUMSUB', channel_pubsub)[2]
if tonumber(channel_pubsub) > 0 then
  msgpacked = msgpack.pack(unpacked)
  redis.call('PUBLISH', channel_pubsub, msgpacked)
end
local num_messages = redis.call('llen', key.messages)

--dbg("channel %s, id: %s, ttl: %s, subscribers: %s, channel_subscribers: (%s)", channel.id, channel.id, channel.ttl, channel.subscribers, channel_fake_subscribers or 0, num_messages)
return (msg.tag, tonumber(channel.ttl or msg.ttl), tonumber(channel.time or msg.time), tonumber(channel.fake_subscribers or channel.subscribers or 0), tonumber(num_messages)), new_channel)
```

delete.lua

- Does the obvious thing
- PUBLISHes deletion notice to all PUBSUB clients

```
--input: keys: [], values: [ channel_id ]
--output: channel_hash {ttl, time_last_seen, subscribers, messages} or nil
-- delete this channel and all its messages
local id = ARGV[1]
local key_msg= 'channel:msg:%s:'..id --not finished yet
local key_channel='channel:'..id
local messages= 'channel:messages:'..id
local subscribers='channel:subscribers:'..id
local pubsub= 'channel:pubsub:'..id

redis.call('echo', ' ##### DELETE #####')
local num_messages = 0
--delete all the messages right now mister!
local msg
while true do
  msg = redis.call('LPOP', messages)
  if msg then
    num_messages = num_messages + 1
    redis.call('DEL', key_msg:format(msg))
  else
    break
  end
end

local del_msgpack = cmsgpack.pack({{"alert", "delete channel", id}})
for k,channel_key in pairs(redis.call('SMEMBERS', subscribers)) do
  redis.call('PUBLISH', channel_key, del_msgpack)
end

local nearly_departed = nil
if redis.call('EXISTS', key_channel) ~= 0 then
  nearly_departed = redis.call('hmget', key_channel, 'ttl', 'time_last_seen',
'subscribers')
  for i = 1, #nearly_departed do
    nearly_departed[i]=tonumber(nearly_departed[i]) or 0
  end

  --leave some crumbs behind showing this channel was just deleted
  redis.call('setex', "channel_deleted"..id, 5, 1)

  table.insert(nearly_departed, num_messages)
end

redis.call('DEL', key_channel, messages, subscribers)

if redis.call('PUBSUB', 'NUMSUB', pubsub)[2] > 0 then
  redis.call('PUBLISH', pubsub, del_msgpack)
end

return nearly_departed
```

odds and ends

- **add_fakesub**

- updated once per subscriber request
- too much traffic, so binned to 100-ms intervals

- **find_channel**

- used for access controls

- **channel_keepalive**

- key expiry management

- **get_message_from_key**

- easier to deal with than an HMGET

add_fakesub.lua

```
--input: keys: [], values: [channel_id, number]
--output: current_fake_subscribers

redis.call('echo', ' ##### FAKESUBS ##### ')
local id=ARGV[1]
local num=tonumber(ARGV[2])
if num==nil then
    return {err="fakesub number not given"}
end

local chan_key = 'channel:'..id
local exists = false
if redis.call('EXISTS', chan_key) == 1 then
    exists = true
end

local cur = 0

if exists or (not exists and num > 0) then
    cur = redis.call('HINCRBY', chan_key, 'fake_subscribers', num)
    if not exists then
        redis.call('EXPIRE', chan_key, 5) --something small
    end
end

return cur
```

find_channel.lua

```
--input: keys: [], values: [channel_id]
--output: channel_hash {ttl, time_last_seen, subscribers, messages} or nil
-- finds and return the info hash of a channel, or nil if channel not found
local id = ARGV[1]
local key_channel='channel:'..id

redis.call('echo', ' ##### FIND_CHANNEL ##### ')

if redis.call('EXISTS', key_channel) ~= 0 then
    local ch = redis.call('hmget', key_channel, 'ttl', 'time_last_seen', 'subscribers', 'fake_subscribers')
    if(ch[4]) then
        --replace subscribers count with fake_subscribers
        ch[3]=ch[4]
        table.remove(ch, 4)
    end
    for i = 1, #ch do
        ch[i]=tonumber(ch[i]) or 0
    end
    table.insert(ch, redis.call('llen', "channel:messages"..id))
    return ch
else
    return nil
end
```

channel_keepalive.lua

```
--input: keys: [], values: [channel_id, ttl]
-- ttl is for when there are no messages but at least 1 subscriber.
--output: seconds until next keepalive is expected, or -1 for "let it disappear"
redis.call('ECHO', ' ##### CHANNEL KEEPALIVE ##### ')
local id=ARGV[1]
local ttl=tonumber(ARGV[2])
if not ttl then
    return {err="Invalid channel keepalive TTL (2nd arg)"}
end

local random_safe_next_ttl = function(ttl)
    return math.floor(ttl/2 + ttl/2.1 * math.random())
end

local key={
    channel= 'channel:'..id, --hash
    messages= 'channel:messages:'..id, --list
}

local subs_count = tonumber(redis.call('HGET', key.channel, 'subscribers')) or 0
local msgs_count = tonumber(redis.call('LLEN', key.messages)) or 0
local actual_ttl = tonumber(redis.call('TTL', key.channel))

if subs_count > 0 then
    if msgs_count > 0 and actual_ttl > ttl then
        return random_safe_next_ttl(actual_ttl)
    end
    --refresh ttl
    redis.call('expire', key.channel, ttl);
    redis.call('expire', key.messages, ttl);
    return random_safe_next_ttl(ttl)
else
    return -1
end
```

get_message_from_key.lua

```
--input: keys: [message_key], values: []
--output: msg_ttl, msg_time, msg_tag, prev_msg_time, prev_msg_tag, message, content_type, eventsource_event, channel_subscriber_count

local key = KEYS[1]

local ttl = redis.call('TTL', key)
local time, tag, prev_time, prev_tag, data, content_type, es_event = unpack(redis.call('HMGET', key, 'time', 'tag', 'prev_time', 'prev_tag', 'data', 'content_type', 'eventsource_event'))

return {ttl, time, tag, prev_time or 0, prev_tag or 0, data or "", content_type or "", es_event or ""}
```

Nchan wishes and T0D0s

- hiredis:
 - no-copy redisCommand parameter
- Nchan (redis stuff):
 - Configurable key prefix
 - Multiple Redis servers
 - Redis cluster support



Try hsss

```
> gem install hsss
```

- Code and documentation at
<https://github.com/slact/hsss>
- Try Redis + hsss:
<https://github.com/slact/redis>
(unstable branch)





- Thorough documentation and examples at <https://nchan.slact.net>
- Build and run:
 - macs: > brew tap homebrew/nginx
 > brew install nginx-full --with-nchan
 - From source: <http://github.com/slact/nchan>
 - Pre-packaged: <https://nchan.slact.net/#download>

Fin

Contact me: shell@slact.net

slides and notes at

<https://nchan.slact.net/redisconf>

- <https://github.com/slact>
- <https://nchan.slact.net>
- Please support Nchan Development
(if you're interested):
 - *Paypal*: nchan@slact.net
 - *Bitcoin*: 15dLBzRS4HLRwCCVjx4emYkxXcyAPmGxM3

