Let's jabber

About ejabberd!

Ahmed Omar
@spawn_think

The Readers' CHOICE AWARDS 2011

About.com

WINNER

NIMBUZZ

BEST MOBILE IM

APPSOLUTELY FREE®

ovi

FREE

http://get.nimbuzz.com

Agenda

- What's Ejabberd?
- Ejabberd Development
- What's good in there?
- What to watch out for
- Ejabberd @ Nimbuzz
- Q & A

What’s ejabberd?

- The Erlang flavor of Jabber/XMPP servers.

ERLANG

What’s xmpp?

- Extensible Messaging and Presence Protocol
- Who is using it and what for?
- Basics
  - Tcp connection
  - JIDs and resources
  - Roster and subscription states
  - XML stream and stanzas
  - XEPs

Identify yourself

- Bare JID

  user1@server-x

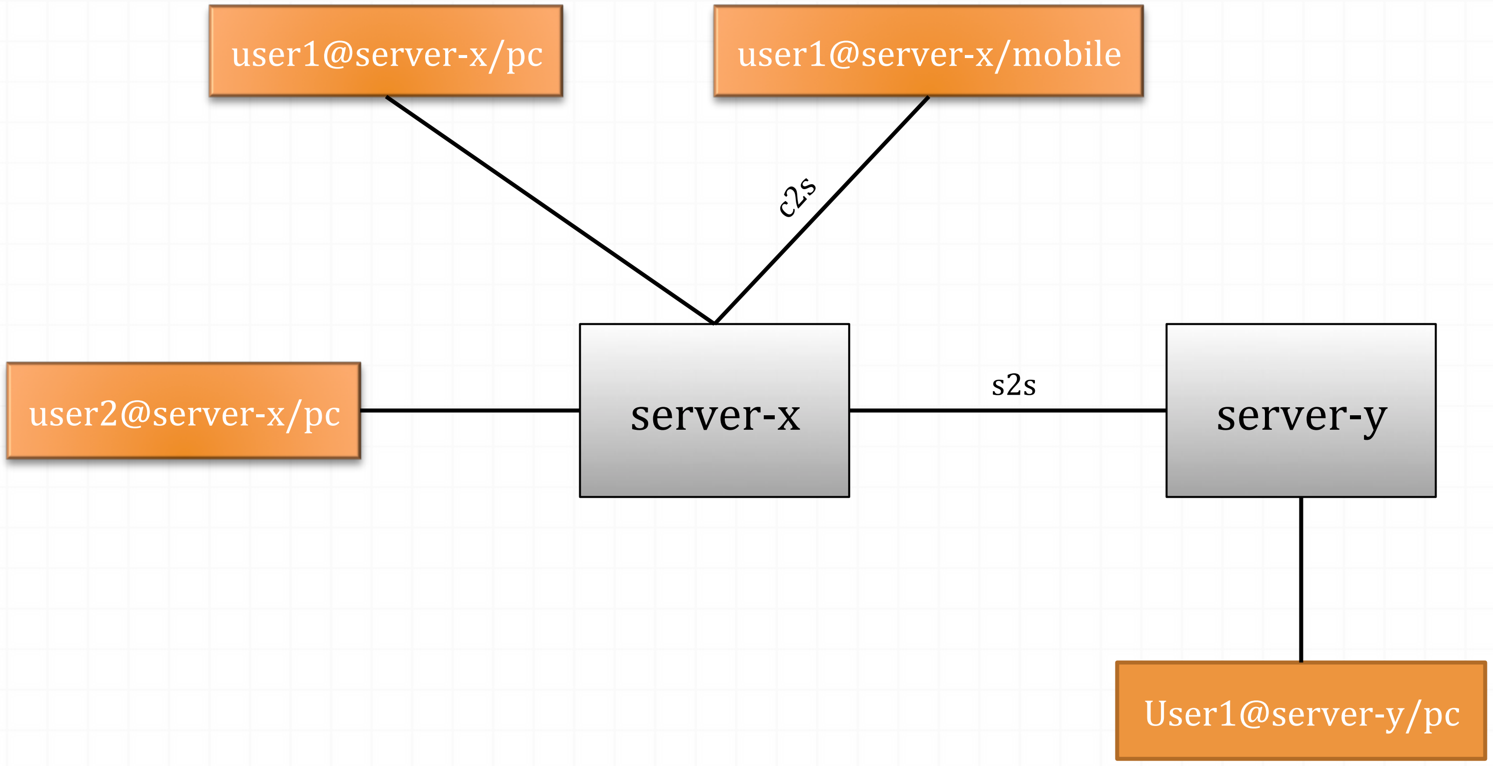
- Full JID (with resource)

  user1@server-x/pc

Subscription states

- "none”
- "to" I’m interested in you, but you are not!
- "from” You are interested in me, but I don’t care!
- "both” Horray, we are friends!

Put things together



XML Stream

```
<stream>
      …….
      <presence/>
      <message/>
      <iq/>
      ……..
</stream>
```

XMPP stanzas examples

```
<iq type='set' id='bind_1'>
 <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
  <resource>pc</resource>
 </bind>
</iq>

<iq type='result' id='bind_1'>
 <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
  <jid>user1@server-x/pc</jid>
 </bind>
</iq>
```

XMPP stanzas examples

Initial Presence → Ask/Tell the world.

```
<presence/>      →    <presence type='probe’
                                from=‘user1@server-x/pc'
                                to=’user2@server-x’/>
```

and also

```
                      <presence
                                from=‘user1@server-x/pc'
                                to=’user2@server-x’/>
```

XMPP stanzas examples

Wanna be friends?

```
<presence to=‘user3@server-x’ type=‘subscribe’/>
```

Sure!

```
<presence to=‘user3@server-x’ type=‘subscribed’/>
```

Or… Go Away!

```
<presence to=‘user3@server-x’ type=‘unsubscribed’/>
```

XMPP stanzas examples

Let’s jabber!

```
<message
    to='user3@server-x'
    from ='user1@server-x/pc'
    type = 'chat'>
<body> Hey </body>
</message>
```

Ejabberd core

- ejabberd_router
- ejabberd_local
- ejabberd_sm
- Jlib
- Xml
- gen_mod

Ejabberd Development

Ejabberd Development

- Modules : gen_mod

```
-module(mod_example).
-behaviour(gen_mod).
-export([start/2, stop/1]).

start(Host, _Opts) ->
	ok.

stop(Host) ->
	ok.
```

Ejabberd Development

- Events and Hooks.
- Routes.
- IQ handlers. (ejabberd_local, ejabberd_sm).
- HTTP requests handlers.

Ejabberd Development

- Hook it!

```
start(Host, _Opts) ->
      ejabberd_hooks:add(privacy_check_packet, Host,
                ?MODULE, check_packet, 25)
stop(Host) ->
      ejabberd_hooks:delete(privacy_check_packet, Host,
                ?MODULE, check_packet, 25),

check_packet(_Flag, User, Server, PrivacyList,
         {From, To, Stanza}, Dir) ->
……
      allow | deny.
```

Ejabberd Development

- Or register a route

```
ejabberd_router:register_route(Domain).
```

Ejabberd Development

- or Handle IQs for a specific namespace

```
gen_iq_handler:add_iq_handler(ejabberd_local,
                              Host,
                              ?NS_LAST,
                              ?MODULE,
                              process_local_iq,
                              IQDisc).
```

Ejabberd Development

- Add your module to ejabberd.cfg

```
{modules,
    [
      ...
      {mod_example, []},
      ...
 ]}.
```

So, What’s good in there?

What's good?

- Flexibility
    - Quite easy to set up/modify a cluster.
    - Support for external services.
    - On the fly configurations.
    - Easy module development.
- Power
- Scalability …. With caution

Any limitations?

Limitations?

- Not much options when it comes to back end.
- Not enough monitoring tools.
- No advanced logging.
- Not all XEPs are implemented.

Anything bad?

Nothing!

It’s written in Erlang, so it must be scalable, robust, fault-tolerant and kicks a**!

Right?

Erlang is not enough

- Q: Do Erlang programs scale?
- A: Wrong question,

Well designed, and well written programs scale.

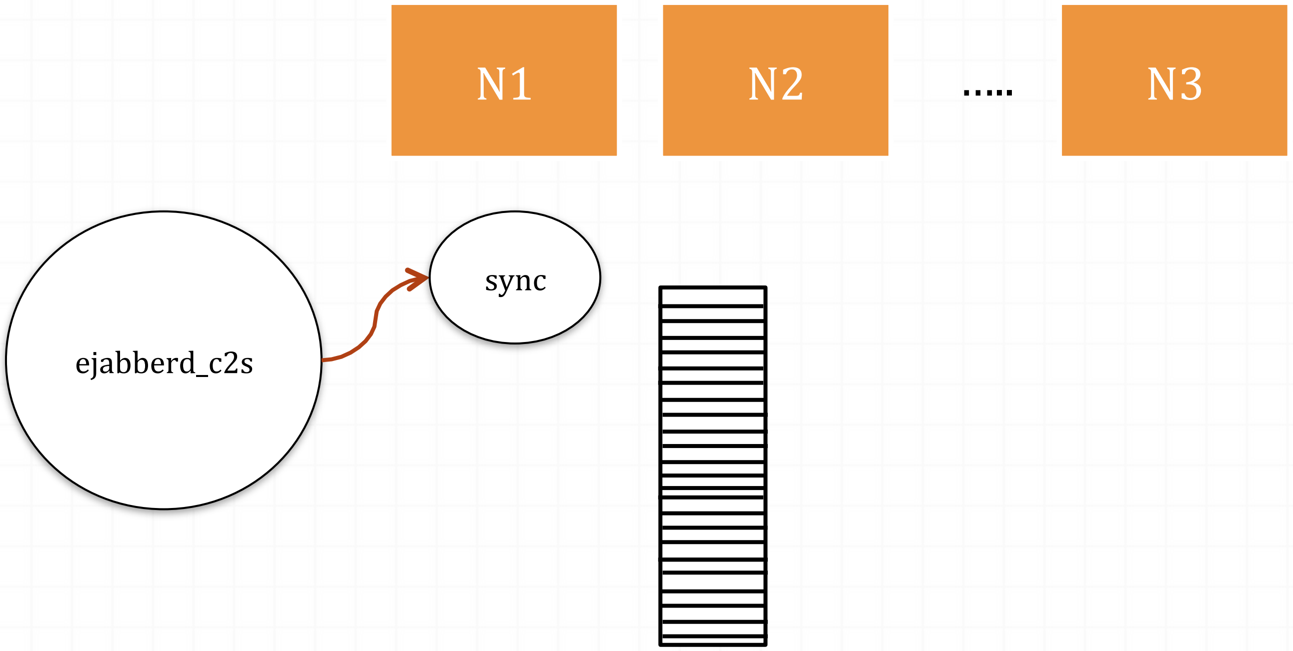
It's just much easier to achieve that in Erlang.

Watch out!

- Memory
  - Strings (ejabberd-3.0 will use binaries, but still alpha).
  - Queues … Queues … Queues
- Mnesia scalability
- OTP-less design
- Maintainability & Testing

Shoot yourself in the foot (a.k.a queues)

- One process handles all requests for the session.
- Mnesia operations and sync dirty.



OTP-less design

- ejabberd is not an otp app.
- No proper OTP release.
- ejabberd is down, and node is still up and in the cluster? fail fast vs. hiding failures?

Recovery from failures

Just use supervisors, right?

```erlang
add_iq_handler(Component, Host, NS, Module, Function, Type) ->
    case Type of
        no_queue ->
            Component:register_iq_handler(Host, NS, Module, Function, no_queue);
        one_queue ->
            {ok, Pid} = supervisor:start_child(ejabberd_iq_sup,
                                               [Host, Module, Function]),
            Component:register_iq_handler(Host, NS, Module, Function,
                                          {one_queue, Pid});
        {queues, N} ->
            Pids =
                lists:map(
                  fun(_) ->
                          {ok, Pid} = supervisor:start_child(
                                        ejabberd_iq_sup,
                                        [Host, Module, Function]),
                          Pid
                  end, lists:seq(1, N)),
            Component:register_iq_handler(Host, NS, Module, Function,
                                          {queues, Pids});
        parallel ->
            Component:register_iq_handler(Host, NS, Module, Function, parallel)
    end.
```

gen_iq_handler.erl

```
    IQSupervisor =
        {ejabberd_iq_sup,
         {ejabberd_tmp_sup, start_link,
          [ejabberd_iq_sup, gen_iq_handler]},
         permanent,
         infinity,
         supervisor,
         [ejabberd_tmp_sup]},
```

ejabberd_sup.erl

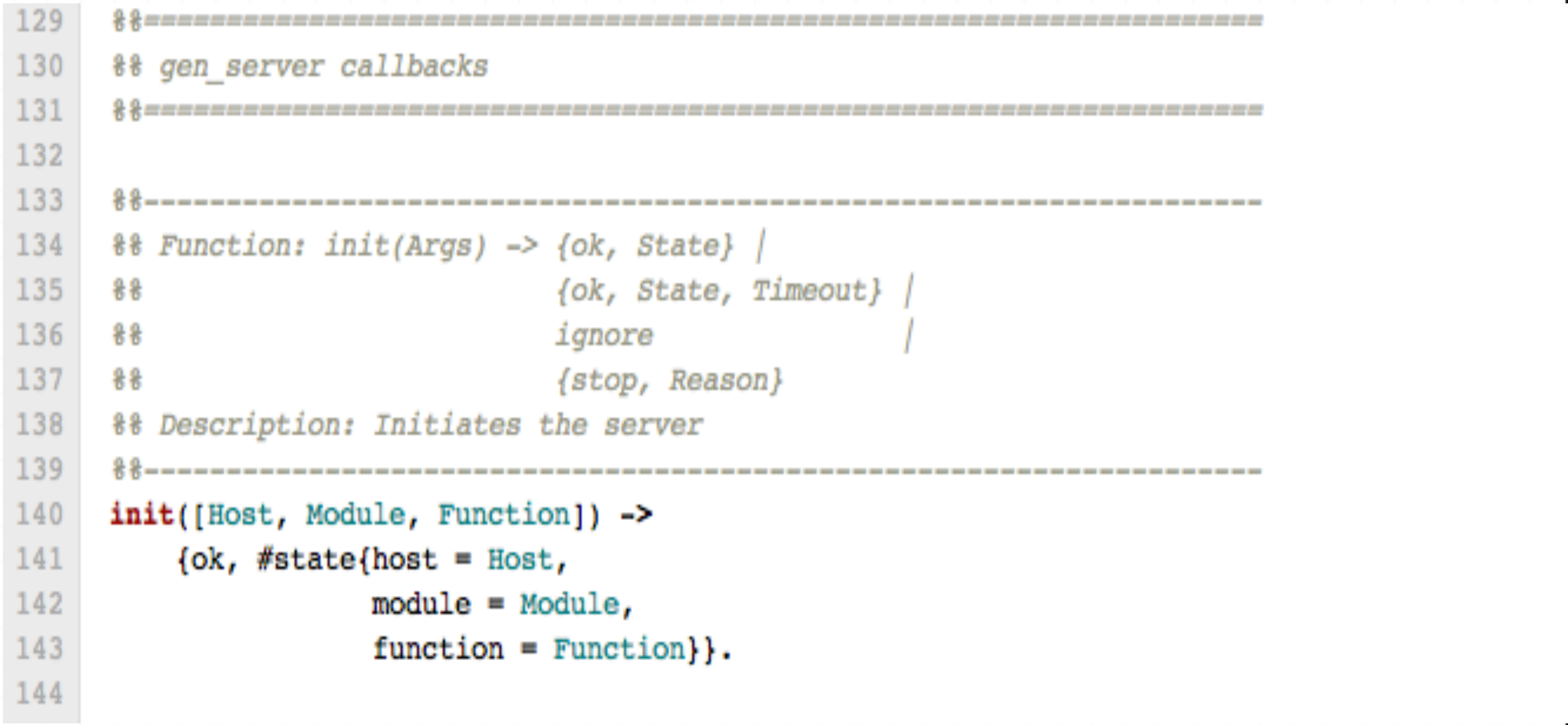
```
-module(ejabberd_tmp_sup).
-author('alexey@process-one.net').

-export([start_link/2, init/1]).

start_link(Name, Module) ->
    supervisor:start_link({local, Name}, ?MODULE, Module).


init(Module) ->
    {ok, {{simple_one_for_one, 10, 1},
          [{undefined, {Module, start_link, []},
            temporary, brutal_kill, worker, [Module]}]}}.
```

ejabberd_tmp_sup.erl

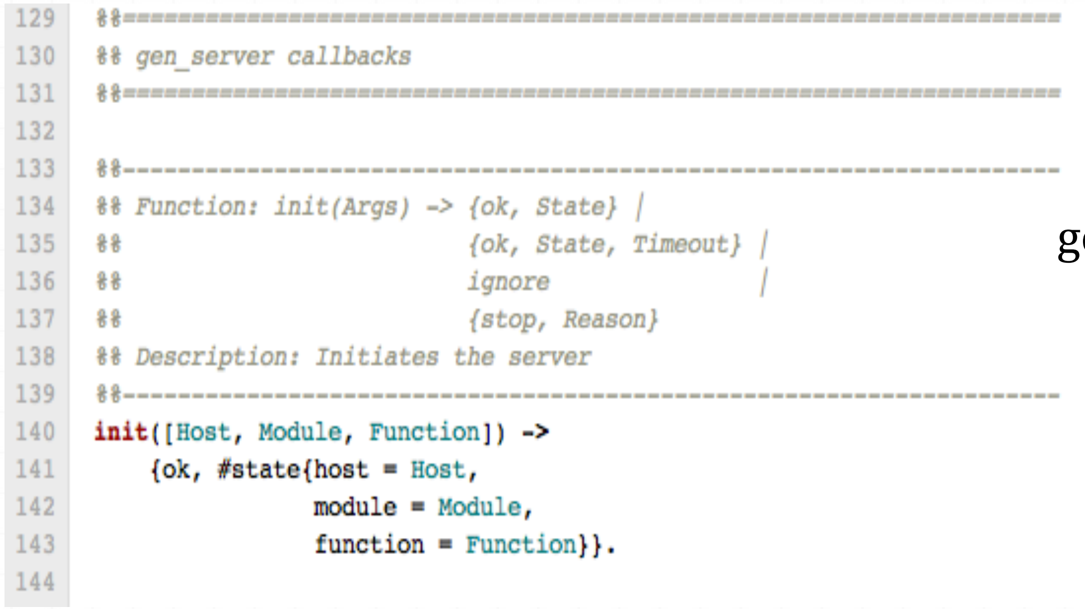

```
%%====================================================================
%% gen_server callbacks
%%====================================================================


%%--------------------------------------------------------------------
%% Function: init(Args) -> {ok, State} |
%%                         {ok, State, Timeout} |
%%                         ignore               |
%%                         {stop, Reason}
%% Description: Initiates the server
%%--------------------------------------------------------------------
init([Host, Module, Function]) ->
    {ok, #state{host = Host,
                module = Module,
                function = Function}}.

```

gen_iq_handler.erl

Where's the data?

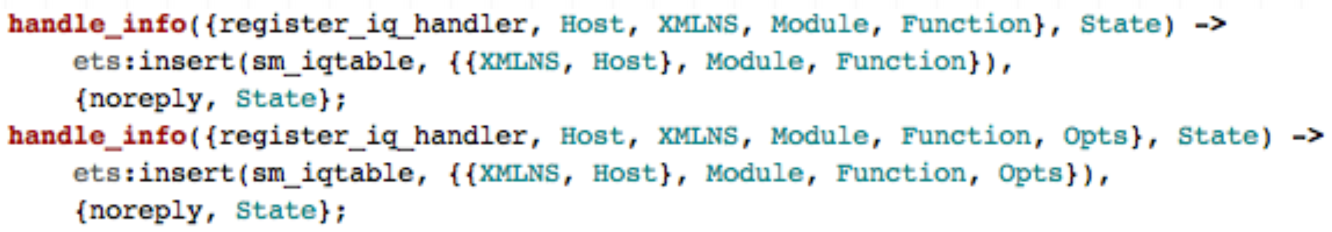


```
%%====================================================================
%% gen_server callbacks
%%====================================================================

%%--------------------------------------------------------------------
%% Function: init(Args) -> {ok, State} |
%%                         {ok, State, Timeout} |
%%                         ignore               |
%%                         {stop, Reason}
%% Description: Initiates the server
%%--------------------------------------------------------------------
init([Host, Module, Function]) ->
    {ok, #state{host = Host,
                module = Module,
                function = Function}}.
```

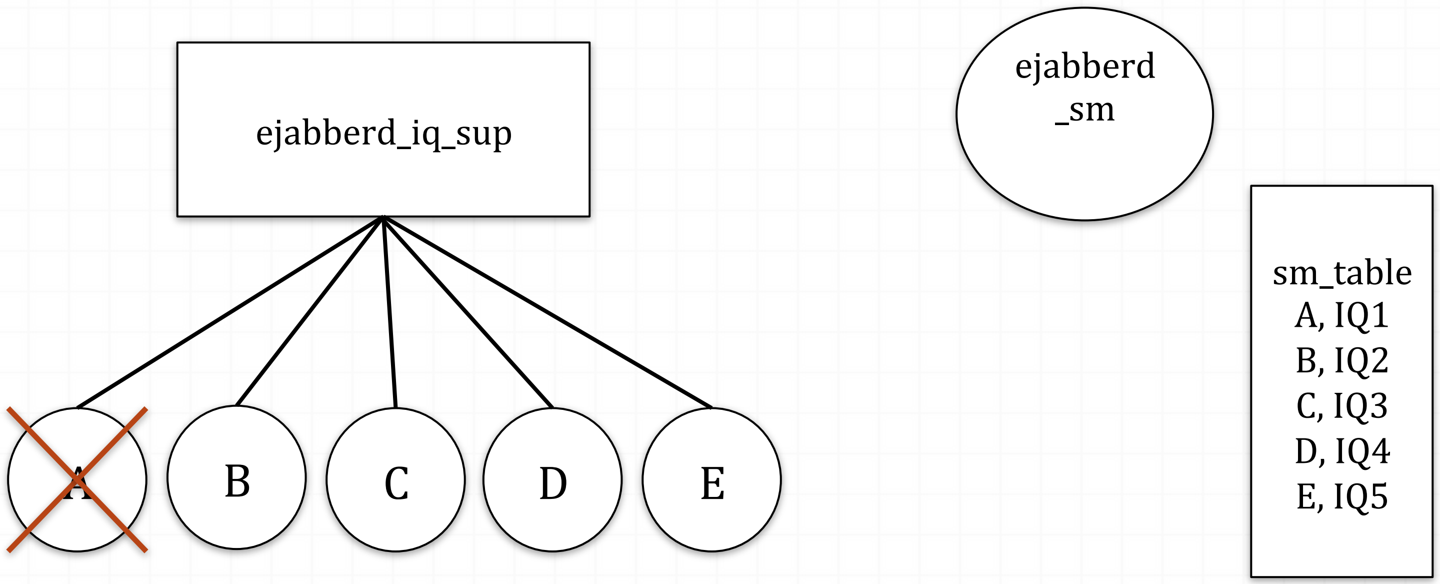
gen_iq_handler.erl

ejabberd_sm.erl



```
handle_info({register_iq_handler, Host, XMLNS, Module, Function}, State) ->
    ets:insert(sm_iqtable, {{XMLNS, Host}, Module, Function}),
    {noreply, State};
handle_info({register_iq_handler, Host, XMLNS, Module, Function, Opts}, State) ->
    ets:insert(sm_iqtable, {{XMLNS, Host}, Module, Function, Opts}),
    {noreply, State};
```

And….


ejabberd_iq_sup
A
B
C
D
E
ejabberd _sm
sm_table
A, IQ1
B, IQ2
C, IQ3
D, IQ4
E, IQ5

Maintainability & Testing

- 2000+ lines modules?
- 200+ lines function?
- Dive into the cases, deeply!
- Not everything is documented
- Comments? hmm
- edoc? shhh!
- Community website, not so up-to-date
- Unit tests? Coverage? ehmmm.....

Ejabberd @ Nimbuzz

Ejabberd @ Nimbuzz

- Our setup. (20+ nodes).
- Backend (DB + caching).
- Protocols (XMPP, HTTP, Thrift).

Tools we use

- Testing :
  - Common test and exmpp
- Monitoring:
  - Munin and Nagios
  - Etop
- Analysis & Profiling :
  - Crashdump viewer (oh yeah!)
  - The mighty Dialyzer
  - Prof family! (eprof, cprof, ...etc)

Conclusion

Ejabberd is a powerful extensible solution.

Still, there’s a great room for improvement.

References

http://xmpp.org/

http://www.ejabberd.im

http://www.process-one.net/docs/ejabberd/guide_en.html

http://www.process-one.net/en/wiki/ejabberd/

Questions?

Or let’s have the break? ☺

You can reach me:

@spawn_think

spawn.think@gmail.com