INTRODUCTION TO EXTENDING EJABBERD

USING ERLANG FOR THE FIRST TIME

Kyle Burton / @kyleburton

INCORPORATE CHAT INTO OUR PRODUCT

- Quickly: Use Standard Software
- Be Robust: Support Clustering and HA
- Support multi-person Rooms

EJABBERD

Met all of our initial requirements, then things changed...

EJABBERD EXTENDED

- See who is in a room
- Peek at the messags in a room
- Post a message to a room on behalf of a user
- Pre-create Rooms
- Create user credentials

CHOICES CHOICES

Do we use the existing tools and extend XMPP itself?

Do we create an alternate interface into eJabberd's internals?

1: EXTEND XMPP

- PRO: can use existing libs (bosh, smack)
- CON: smack is flaky from the JVM
- CON: normal users could make these calls
- CON: may break standard clients (pidgin)
- CON: eJabberd prevents the same user from logging in multiple times (dealbreaker!)

2: REST API

- PRO: easy to call into from the JVM
- PRO: easy to map actions to RESTful URLs
- PRO: mod_restul_admin has 1 of our required features already
- CON: have to hack Erlang (PRO)

REST API, I CHOOSE YOU!

Simplicity wins.

EJABBERD INTERNALS

eJabberd is built on Erlang's OTP Behaviors

OTP BEHAVIORS

- gen_sever
- gen_fsm
- gen_event
- supervisor

GEN_FSM

mod_muc_room is a gen_fsm.

It has a state which contains what we need:

- list of users
- message history

WHAT NOW?

All the pieces were now laid out for us, so how do we hack eJabberd and Erlang?

A strange new world awaited us...

ERLANG: LANGUAGE SEMANTICS

- Expression Based
- Single Assignment, [generally] Immutable Types
- Pattern Matching and Destructuring
- Function Clauses
- Syntax: comma, semi-colon; and period.
- Data Types: atoms, numbers, lists, tuples, binaries, pid, function
- There is no ***String*** type!

ERLANG: RUNTIME SEMANTICS

- iolists
- recursion and process state
- processes
- message passing
- the mnesia distributed database
- code path
- hot reloading code
- upgrading a process

LESSONS LEARNED

- Erlang Cookie
- JSON in a land with no string
- mod_restful_debug
- REPL
- Remote Shell
- `ejabberdctl live`

LESSON: COOKIE

ejabberd didn't use

`$HOME/.erlang.cookie`

like all the other kids, it used

`/var/lib/ejabberd/.erlang.cookie`

instead :/

JSON

`mod_restful_mochijson2` was hard for us to see how it worked.

Not having The `string==list(integer)` equivalence in our heads added to the confusion.

MOD_RESTFUL_DEBUG

We wrote a simple `gen_server` that held a dictionary so we could message it to store or retrieve data. We then used this from inside the other code to capture values, and with `ejabberdctl debug` we could query and see the values.

This was a huge help to our productivity.

REPL

Having access to Erlang's repl was nice, but...

EJABBERDCTL DEBUG

The remote shell, or having a shell that was directly interacting with the server was even better.

Between the remote shell and our debugging service, it was almost like swank.

EJABBERDCTL LIVE

This runs ejabberd in the foreground, not as adaemon. This was invaluable when we broke the service at startup:

- we borked the configuration file
- we messed up the nodename and couldn't connect the debug console
- we tried to call a module:function that didn't exist.

HOTLOADING CODE

Recompile, copy the *.beam files into the proper place and then:

```
l(module_name).
nl(module_name).
```

DEMO

Go flail on the keyboard and make mistakes.

- Extend Ejabberd
- Add Modules
- Code, Reload and Interact
- mod_muc_room stores history in a dict not a ble, so there is no persistance :(

QUESTIONS?

THANK YOU!

REFERENCES

- erlang docs: http://www.erlang.org/doc/
- iolists: http://prog21.dadgum.com/70.html