Scaling Erlang Web Applications

100 to 100K users at one web server

Fernando Benavides (*@elbrujohalcon*)

Inaka Labs

February 27, 2012

Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you

Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you

Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you

Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you

Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you

Outline

Outline

Outline

Outline

We will work on the scalability of a *web* project that has a *HTTP API* and keeps clients *connected* to the server for *long periods* of time.

Examples:

- Social sites
- Chat sites
- Sports sites

We will work on the scalability of a *web* project that has a *HTTP API* and keeps clients *connected* to the server for *long periods* of time.

Examples:

- Social sites
- Chat sites
- Sports sites

We will work on the scalability of a *web* project that has a *HTTP API* and keeps clients *connected* to the server for *long periods* of time.

Examples:

- Social sites
- Chat sites
- Sports sites

We will work on the scalability of a *web* project that has a *HTTP API* and keeps clients *connected* to the server for *long periods* of time.

Examples:

- Social sites
- Chat sites
- Sports sites

We will work on the scalability of a *web* project that has a *HTTP API* and keeps clients *connected* to the server for *long periods* of time.
Examples:

- Social sites
- Chat sites
- Sports sites

We will focus on

- OTP behaviours
- TCP connections
- mochiweb
- Underlaying system configurations

*We will **not** deal with*

- Multiple machines/nodes
- Databases

We will focus on

- OTP behaviours
- TCP connections
- mochiweb
- Underlaying system configurations

*We will **not** deal with*

- Multiple machines/nodes
- Databases

The Plan

General Considerations

- Create a system that works
- Automate your clients
- Keep a human watching
- Be patient

General Considerations

- Create a system that works
- Automate your clients
- Keep a human watching
- Be patient

General Considerations

- Create a system that works
- Automate your clients
- Keep a human watching
- Be patient

General Considerations

- Create a system that works
- Automate your clients
- Keep a human watching
- Be patient

General Considerations

- Create a system that works
- Automate your clients
- Keep a human watching
- Be patient

Goals

- Test the system as it is
- How many users can the system handle as is?
- Find *N* and *C*

Steps

- Choose *N* and *C*
- Test the API
- Test long-lived connections
- Test both
- Repeat with higher values for *N* and *C*

Steps

- Choose *N* and *C*
- Test the API
- Test long-lived connections
- Test both
- Repeat with higher values for *N* and *C*

Goals

- Improve the environment
- Tune-In the machine(s)
- Don't touch the code

Steps

- Check kernel variables
- Check system limits
- Check Erlang VM parameters

Goals

- Tune up your system
- Discover scalability issues and fix them
- Find the biggest *N* and *C* for one node

Steps

- Choose *N* and *C* to fail
- Find the problem
- Fix it
- Add it to the list of *Tips and Tricks*
- Repeat with higher values for *N* and *C*

Steps

- Choose *N* and *C* to fail
- Find the problem
- Fix it
- Add it to the list of *Tips and Tricks*
- Repeat with higher values for *N* and *C*

Goals

- Get the system ready to work on many nodes
- Decide if they should be connected or not
- Find the *N* and *C* per node

Steps

- Get the second node running
- Choose *N* and *C*
- Try interconnected instances
- Try independent instances
- Repeat with higher values for *N* and *C*

Steps

- Get the second node running
- Choose *N* and *C*
- Try interconnected instances
- Try independent instances
- Repeat with higher values for *N* and *C*

OS TWEAKS

TODO: Copy from the article

Erlang tweaks

TODO: Copy from the article on listeners TODO: Copy from the article on inbound TCP connections TODO: Copy from the article on outbound TCP connections

GEN_EVENT

TODO: Copy from the article on sup_handler TODO: Copy from the article on long delivery queues

GEN_SERVERS

TODO: Copy from the article on timing out TODO: Copy from the article on too much memory TODO: Copy from the article on taking too long to initiliaze

SUPERVISORS

TODO: Copy from the article

Process Registration

TODO: Copy from the article

Timers

TODO: Copy from the article

Logging

TODO: Copy from the article

Summary

TODO: Summary

Other stuff

that we left out of this presentation

TODO: List of other scalability stuff we left out

Any questions?

```
-spec fact(integer()) -> integer().
fact(N) ->
        lists:fold(fun(X, F) ->
                           F * X
                   end, 1, lists:seq(1,N)).
```