Scaling Erlang Web Applications

100 to 100K users at one web server

Fernando Benavides (*@elbrujohalcon*)

Inaka Labs

March 6, 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

We will work on the scalability of a *web* project that has an *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 an *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 an *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 an *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 an *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 system 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 a 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 a 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
- Design the system topology
- Find *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

Kernel Variables

```
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
sysctl -w net.ipv4.tcp_syncookies=1
sysctl -w net.ipv4.tcp_mem="50576   64768   98152"
sysctl -w net.core.netdev_max_backlog=2500
sysctl -w net.netfilter.nf_conntrack_max=1233000
```

Open Files Limit

```
ulimit -n 999999
```

Erlang VM tweaks

+P Number of Processes

+K Kernell Polling

-SMP SMP Support

OS TWEAKS

Kernel Variables

```
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
sysctl -w net.ipv4.tcp_syncookies=1
sysctl -w net.ipv4.tcp_mem="50576   64768   98152"
sysctl -w net.core.netdev_max_backlog=2500
sysctl -w net.netfilter.nf_conntrack_max=1233000
```

Open Files Limit

```
ulimit -n 999999
```

Erlang VM tweaks

+P Number of Processes

+K Kernell Polling

-SMP SMP Support

OS TWEAKS

Kernel Variables

```
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
sysctl -w net.ipv4.tcp_syncookies=1
sysctl -w net.ipv4.tcp_mem="50576   64768   98152"
sysctl -w net.core.netdev_max_backlog=2500
sysctl -w net.netfilter.nf_conntrack_max=1233000
```

Open Files Limit

```
ulimit -n 999999
```

Erlang VM tweaks

+P Number of Processes

+K Kernell Polling

-SMP SMP Support

Connection tweaks

Backlog

- Allow more concurrent connections
- Remember HTTP *runs on* TCP

Connections

- Don't use just one of them
- Check inbound and outbound connections

Connection tweaks

Backlog

- Allow more concurrent connections
- Remember HTTP *runs on* TCP

Connections

- Don't use just one of them
- Check inbound and outbound connections

GEN_EVENT

SUP_HANDLER

- Don't use it
- Monitor the processes instead

LONG DELIVERY QUEUES

- Use *repeaters*

GEN_EVENT

SUP_HANDLER

- Don't use it
- Monitor the processes instead

Long Delivery Queues

- Use *repeaters*

GEN_SERVERS

Call Timeouts

Remember `gen_server:reply/2`

Memory Footprint

Remember `hibernate`

Long init/1

Use 0 timeout

GEN_SERVERS

Call Timeouts

Remember `gen_server:reply/2`

Memory Footprint

Remember `hibernate`

Long init/1

Use 0 timeout

GEN_SERVERS

Call Timeouts

Remember `gen_server:reply/2`

Memory Footprint

Remember `hibernate`

Long init/1

Use 0 timeout

SUPERVISORS

- Sometimes `simple_one_for_one` supervisors get overburdened because they have too many children
- Try a supervisor hierarchy with several managers below the main supervisor
- Turn `supervisor:start_child/2` calls into something like

```
supervisor:start_child(
  list_to_atom("module-name_" ++
                integer_to_list(random:uniform(#ofSupervisors))).
```

Timers

- Don't use the `timer` module
- Use `erlang:send_after`

Logging

- Don't log too much
- Use a good logging system

Registration

Sometimes it's better to register processes instead of keeping track of their pids manually

Timers

- Don't use the `timer` module
- Use `erlang:send_after`

Logging

- Don't log too much
- Use a good logging system

Registration

Sometimes it's better to register processes instead of keeping track of their pids manually

Timers

- Don't use the `timer` module
- Use `erlang:send_after`

Logging

- Don't log too much
- Use a good logging system

Registration

Sometimes it's better to register processes instead of keeping track of their pids manually

Summary

- It is an iterative process
- It worked awesomely for us in both experimental and real-life systems
- It's no silver bullet
- The list of *Tips and Tricks* grows constantly over time

Summary

- It is an iterative process
- It worked awesomely for us in both experimental and real-life systems
- It's no silver bullet
- The list of *Tips and Tricks* grows constantly over time

Summary

- It is an iterative process
- It worked awesomely for us in both experimental and real-life systems
- It's no silver bullet
- The list of *Tips and Tricks* grows constantly over time

Summary

- It is an iterative process
- It worked awesomely for us in both experimental and real-life systems
- It's no silver bullet
- The list of *Tips and Tricks* grows constantly over time

Other stuff

that we left out of this presentation

- Adding nodes
- Choosing databases
- System specific improvements
- Measuring tools

Any questions?

Thanks!