

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

THE CHALLENGE

What do we have to deal with?

THE PLAN

How do we face it?

THE TIPS AND TRICKS

What have we learned from it?

OUTLINE

THE CHALLENGE

What do we have to deal with?

THE PLAN

How do we face it?

THE TIPS AND TRICKS

What have we learned from it?

OUTLINE

THE CHALLENGE

What do we have to deal with?

THE PLAN

How do we face it?

THE TIPS AND TRICKS

What have we learned from it?

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
- Underlying system configurations

*We will **not** deal with*

- Multiple machines/nodes
- Databases

We will focus on

- OTP behaviours
- TCP connections
- mochiweb
- Underlying 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

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)).
```