

SCALING ERLANG WEB APPLICATIONS

100 TO 100K USERS AT ONE WEB SERVER

Fernando Benavides (*@elbrujohalcon*)

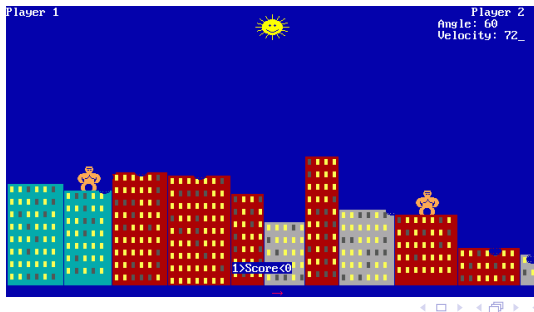
Inaka Labs

March 19, 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



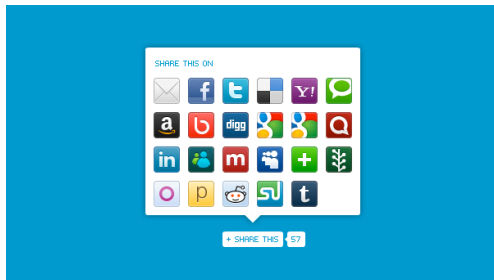
HELLO WORLD!

- 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?



THE CHALLENGE

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

Examples:

- Social sites
- Chat sites
- Sports sites



THE CHALLENGE

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

Examples:

- Social sites
- Chat sites
- Sports sites



THE CHALLENGE

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

Examples:

- Social sites
- Chat sites
- Sports sites



THE CHALLENGE

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

Examples:

- Social sites
- Chat sites
- Sports sites



THE CHALLENGE

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

Examples:

- Social sites
- Chat sites
- Sports sites



THE CHALLENGE

We will try to improve the way we use

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

*We will **not** deal with*

- Multiple machines/nodes
- Database choices and/or implementations



THE CHALLENGE

We will try to improve the way we use

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

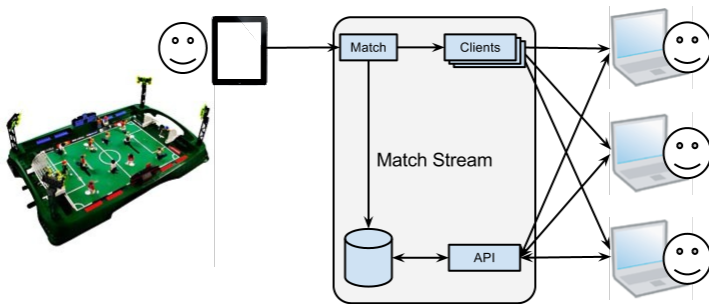
*We will **not** deal with*

- Multiple machines/nodes
- Database choices and/or implementations



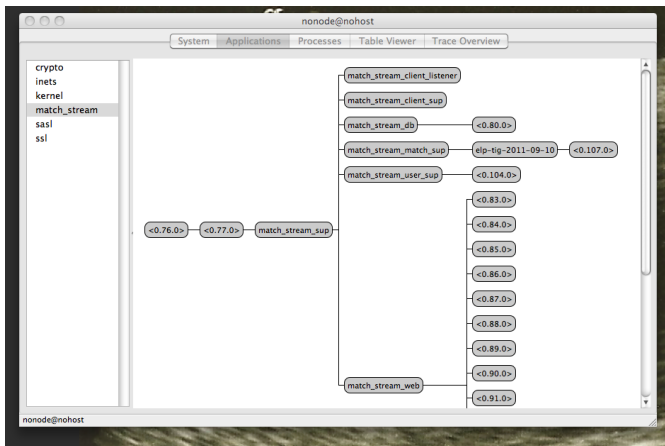
MATCH STREAM

THE SAMPLE PROJECT



MATCH STREAM

INITIAL ARCHITECTURE



THE PLAN



GENERAL RULES

THINGS YOU NEED TO KNOW WHEN SCALING YOUR SYSTEMS

- Start with a system that **works**
- Automate your clients
- Keep a human watching
- Be patient



GENERAL RULES

THINGS YOU NEED TO KNOW WHEN SCALING YOUR SYSTEMS

- Start with a system that **works**
- Automate your clients
- Keep a human watching
- Be patient



GENERAL RULES

THINGS YOU NEED TO KNOW WHEN SCALING YOUR SYSTEMS

- Start with a system that **works**
- Automate your clients
- **Keep a human watching**
- Be patient



GENERAL RULES

THINGS YOU NEED TO KNOW WHEN SCALING YOUR SYSTEMS

- Start with a system that **works**
- Automate your clients
- Keep a human watching
- Be patient



STAGE 1

FINDING THE INITIAL BOUNDARIES

GOALS

- Test the system as it is
- Find N and C

STEPS

- Create the automated testers
- Choose N and C
- Test the API and long-lived connections independently
- Test both together
- Repeat until you find the highest N and C



STAGE 1

FINDING THE INITIAL BOUNDARIES

GOALS

- Test the system as it is
- Find N and C

STEPS

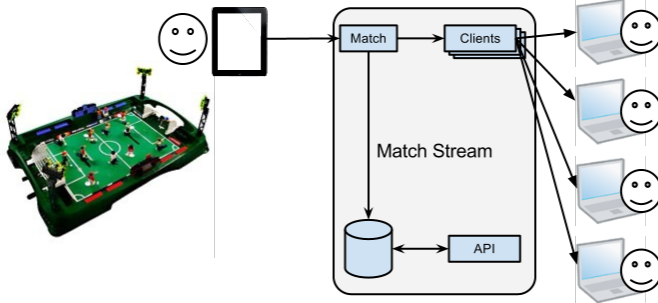
- Create the automated testers
- Choose N and C
- Test the API and long-lived connections independently
- Test both together
- Repeat until you find the highest N and C



STAGE 1

FINDING THE INITIAL BOUNDARIES

RESULTS



$N = 1024 / C = 4$



STAGE 2

BLACKBOX TESTS

GOALS

- Improve the system environment
- Find the highest N and C without altering the code

STEPS

- Check kernel variables
- Check system limits
- Check Erlang VM parameters
- Repeat from Stage 1



STAGE 2

BLACKBOX TESTS

GOALS

- Improve the system environment
- Find the highest N and C without altering the code

STEPS

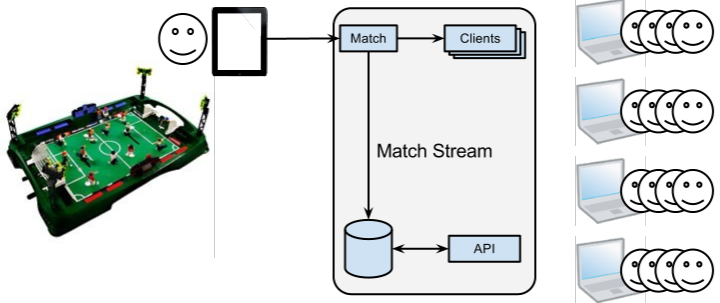
- Check kernel variables
- Check system limits
- Check Erlang VM parameters
- Repeat from Stage 1



STAGE 2

BLACKBOX TESTS

RESULTS



$N = 4096 / C = 4$



STAGE 3

ERLANG TUNING

GOALS

- Tune up **your** system
- Find the highest N and C for **one node**

STEPS

- Find a problem
- Fix it using the list of *Tips and Tricks*
- If not there, add it
- Repeat **from Stage 1**



STAGE 3

ERLANG TUNING

GOALS

- Tune up **your** system
- Find the highest N and C for **one node**

STEPS

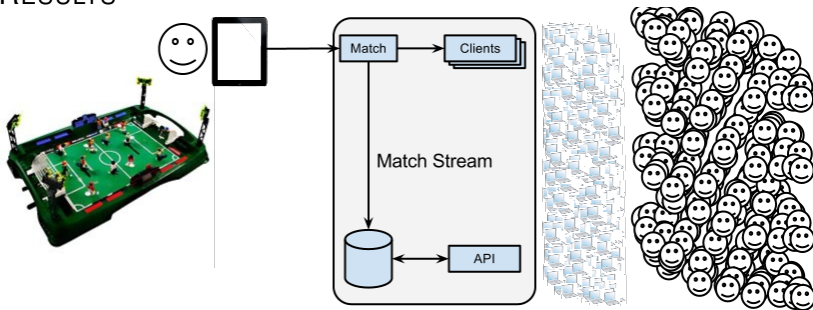
- Find a problem
- Fix it using the list of *Tips and Tricks*
- If not there, add it
- Repeat **from Stage 1**



STAGE 3

ERLANG TUNING

RESULTS



N = 65536 / C = 8192



STAGE 4

ADDING NODES

GOALS

- Find the best system topology
- Find N and C **per node**

STEPS

- Add a node
 - connected; or
 - independent
- Repeat **from Stage 1**



STAGE 4

ADDING NODES

GOALS

- Find the best system topology
- Find N and C **per node**

STEPS

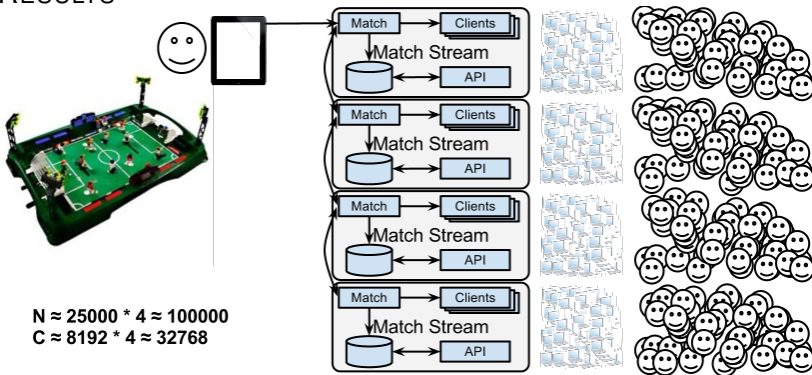
- Add a node
 - connected; or
 - independent
- Repeat **from Stage 1**



STAGE 4

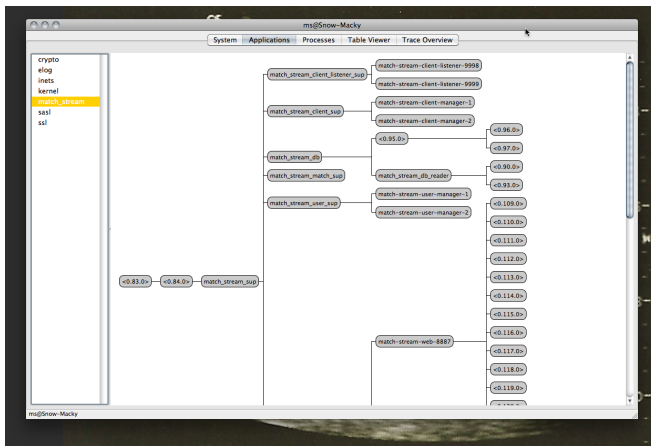
ADDING NODES

RESULTS



MATCH STREAM

FINAL ARCHITECTURE



TIPS AND TRICKS



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

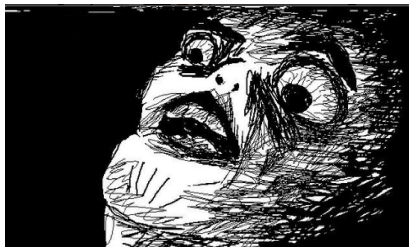
+P Number of Processes
+K Kernell Polling
-SMP SMP Support



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_synccookies=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
```



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_SERVER

CALL TIMEOUTS

Remember `gen_server:reply/2`

MEMORY FOOTPRINT

Remember `hibernate`

LONG INIT/1

Use 0 timeout



GEN_SERVER

CALL TIMEOUTS

Remember `gen_server:reply/2`

MEMORY FOOTPRINT

Remember `hibernate`

LONG INIT/1

Use 0 timeout



GEN_SERVER

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))).
```



OTHER PROCESSES

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
- You can always register processes **both** locally and globally



OTHER PROCESSES

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
- You can always register processes **both** locally and globally



OTHER PROCESSES

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
- You can always register processes **both** locally and globally



SUMMARY

- This 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

- This 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

- This 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

- This 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



SCALING TOPICS

THAT WEREN'T COVERED ON THIS PRESENTATION

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



QUESTIONS



Thanks!

