

# Wobserver

Easy to Integrate Monitoring and Debugging

Ian Luites

May 5, 2017

ElixirConf EU 🍄





# The man behind the beard

Backend developer at Square Enix (London).  
We started using Elixir in production.





# Table of contents

- 1 Wobserver
- 2 Integration
- 3 Extending
- 4 A look into the code
- 5 Reflection
- 6 Questions

# Wobserver



# Wobserver: Introduction

## What?

*W[eb]-observer*

*Wobserver*

Web based metrics, monitoring, and observer



# Wobserver: Introduction

## Why?

Observing the internal state of Elixir applications is easy through the use of `:observer`.

It allows for easier debugging and is a great tool for understanding the inner workings of an application.

Sadly enough it is not always possible to use.





# Wobserver: Introduction

## Why? Limited access

- ❑ Micro services
- ❑ Remote deployment (Firewalls, load balancers)
- ❑ Docker containers
- ❑ Lazy dev ops with arbitrary CI rules...







It is possible to connect to nodes through SSH, but...

You will need to:

- ☐ have SSH access to the machine.
- ☐ setup the Erlang VM to only listen on the tunneled port.
- ☐ iterate node by node.



# Wobserver: Introduction

Solution: wobserver

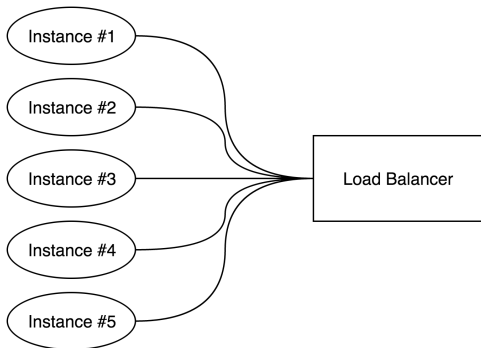
`:wobserver` provides a web interface for easy remote access using websockets.



# Wobserver: Introduction

## What about the load balancer?

### Application

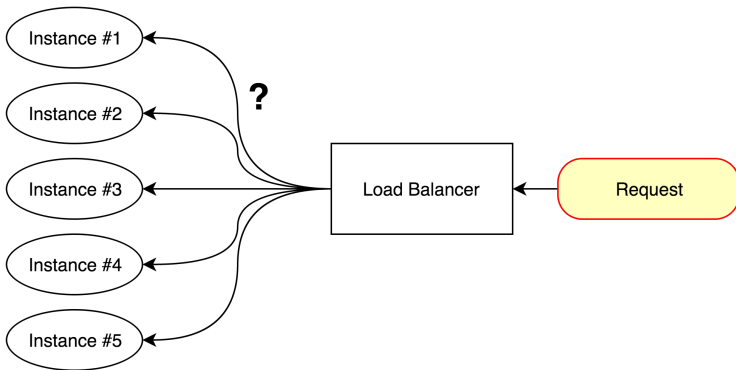




# Wobserver: Introduction

## What about the load balancer?

### Application

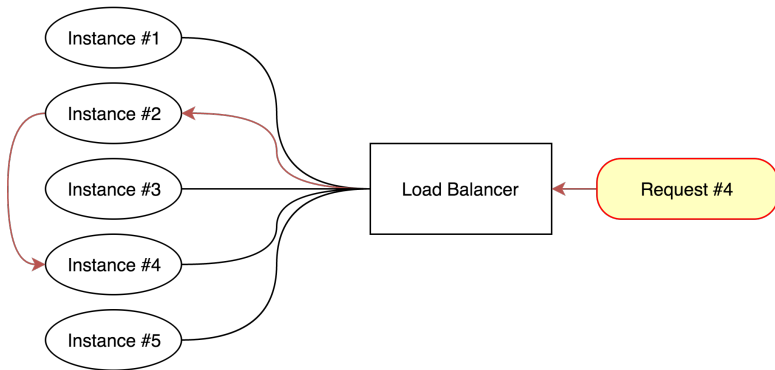




# Wobserver: Introduction

## What about the load balancer?

### Application





`:wobserver` provides the following information in the web-interface:

- ☐ System overview
- ☐ Load charts
- ☐ Memory allocators
- ☐ Application diagram
- ☐ Process list
- ☐ Port list
- ☐ ETS table browser



A JSON API is available to allow for easy automation.

- ☐ Node listing
- ☐ System information
- ☐ Websocket fallback
- ☐ Accessing information from specific nodes

It also functions as a fallback for the web UI, when websockets fail.



`:wobserver` adds a `/metrics` endpoint.

It formats system metrics for *Prometheus*, but can be configured to format metrics in any format.

More metrics can be added dynamically by hooking into `:wobserver`.





# Wobserver: Installation

## Hex

- 1 Add `{:wobserver, "~> 0.1"}` to the dependencies.
- 2 `mix deps.get`
- 3 Done, available on *port 4001*



# Wobserver: Installation

## Build

### 1 Clone

`https://github.com/shinyscorpion/wobserver.git`

### 2 `mix deps.get`

### 3 Done?



`:wobserver` bakes in assets to make it easier to use with distillery.

These assets are not in the repo and need to be build locally.

- 1 npm install
- 2 mix build



# Wobserver: Installation

Build: mix build

`:wobserver` uses *Gulp* for the asset pipeline.

After building the assets `mix build` will take the generated output and put it into a module.

# Integration



It is possible to integrate `:wobserver` with an existing cowboy project.

`:wobserver` is written as a Plug, so to integrate we need to:

- ☐ prevent starting another cowboy.
- ☐ forward requests to `:wobserver` router.



Initial setup of the project.

- 1 `mix new --sup cowboy_wobserver`
- 2 Add `{:wobserver, "~> 0.1"}` to the dependencies.
- 3 Add `{:cowboy, "~> 1.0.0"}` to the dependencies.
- 4 Add `{:plug, "~> 1.0"}` to the dependencies.
- 5 `mix deps.get`



# Integration: Plug

## Cowboy: Router

Create a router for our project. (Empty for now)

ROUTER.EX

```
1 defmodule CowboyWobserver.Router do
2   use Plug.Router
3
4   plug :match
5   plug :dispatch
6
7   forward "/wobserver", to: Wobserver.Web.Router
8 end
```





Setup the application to start Cowboy with our router.

## APPLICATION.EX

```
18     children = [  
19         Cowboy.child_spec(  
20             :http,  
21             CowboyWobserver.Router,  
22             [],  
23             options  
24         )  
25     ]
```



And handle the websocket.

## APPLICATION.EX

```
9      options = [  
10         dispatch: [  
11             {:_ , [  
12                 {"/wobserver/ws", Wobserver.Web.Client, []},  
13                 {:_ , Cowboy.Handler, {CowboyWobserver.Router, []}  
14             ]}  
15         ],  
16     ]
```



# Integration: Plug

## Cowboy: Done

Done.

*`http://localhost:4000/wobserver`*

Everything works, right?





We still need to:

- ☐ Disable standalone mode.
- ☐ Point node discovery in the right direction.

### CONFIG.EXS

```
3 config :wobserver,  
4   mode: :plug,  
5   remote_url_prefix: "/wobserver"
```



# Integration: Phoenix

## Project Setup

- 1 `mix phx.new phoenix_wobserver`
- 2 Add `{:wobserver, "~> 0.1"}` to the dependencies.
- 3 `mix deps.get`
- 4 `mix phx.server`

### Result:

Application `http://localhost:4000/`

Wobserver `http://localhost:4001/`



# Integration: Phoenix

## Plug mode

Add the following lines of code:

### CONFIG.EXS

```
25 config :wobserver,  
26   mode: :plug,  
27   remote_url_prefix: "/wobserver"
```

### ROUTER.EX

```
16 forward "/wobserver", Wobserver.Web.Router
```

### ENDPOINT.EX

```
4 socket "/wobserver", Wobserver.Web.PhoenixSocket
```



## Problem

`:wobserver` is used to observe nodes that can not be connected to using traditional methods.

We would still like to view all nodes through a single instance of `:wobserver`.



## Solution

:wobserver proxies all requests (interface and api) to the correct node.

The metrics are not proxied, but aggregated.

But how does :wobserver find the other nodes?





### CONFIG.EXS

```
config :wobserver,  
  discovery: :dns,  
  discovery_search: "google.nl"
```



### DISCOVERY.EX

```
167  @spec dns_discover(search :: String.t) :: list(Remote.t)
168  defp dns_discover(search) when is_binary(search) do
169      search
170      |> String.to_charlist
171      |> :inet_res.lookup(:in, :a)
172      |> Enum.map(&dns_to_node/1)
173  end
```



# Integration: Node Discovery

## Custom

### CONFIG.EXS

```
config :wobserver,  
  discovery: :custom,  
  discovery_search: &MyApp.discover/0
```

or

```
config :wobserver,  
  discovery: :custom,  
  discovery_search: fn -> [] end
```



### REMOTE.EX

```
12  @typedoc "Remote node information."
13  @type t :: %__MODULE__ {
14      name: String.t,
15      host: String.t,
16      port: integer,
17      local?: boolean,
18  }
```

Every `:wobserver` (one per node) has a separate list of nodes. Only the node local to the `:wobserver` instance is marked with `local?`.



# Integration: Node Discovery

## Connected nodes

**Question:** How do I get wobserver to see the connected nodes?



**Question:** How do I get wobserver to see the connected nodes?

**Question:** Shouldn't they connect automatically?



# Integration: Node Discovery

## Building the node discovery

So let's add it!



# Integration: Node Discovery

## Creating a node

### nodes\_wobserver.ex

```
15  defp node_info(local \\ false) do
16    %Wobserver.Util.Node.Remote{
17      name: node() |> to_string,
18      host: "localhost",
19      port: Application.get_env(:wobserver, :port),
20      local?: local,
21    }
22  end
```





# Integration: Node Discovery

## Sending node info back to a process

nodes\_wobserver.ex

```
11  def remote_node(caller_pid) do
12    send caller_pid, {:wobserver_node, node_info()}
13  end
```



# Integration: Node Discovery

## Querying the node

### nodes\_wobserver.ex

```
24  defp check_node(remote_node) do
25      Task.async fn ->
26          Node.spawn remote_node,
27                      NodesWobserver,
28                      :remote_node,
29                      [self()]
30
31      receive do
32          {:wobserver_node, data} -> data
33      end
34  end
35  end
```



# Integration: Node Discovery

## Sending node info back to a process

nodes\_wobserver.ex

```
2  def list do
3    [
4      node_info(true) |
5      Node.list
6      |> Enum.map(&check_node/1)
7      |> Enum.map(&Task.await/1)
8    ]
9  end
```



# Integration: Node Discovery

## Building the node discovery

Does the solve the problem?



# Integration: Node Discovery

## Building the node discovery

Does the solve the problem? *kind-a*

# Extending



Add additional metrics or generate metrics during runtime by setting the config.

## **Metric types:**

- counter Numeric, should only increase.

- gauge Numeric, can go up and down.



## CONFIG.EXS

```
3 config :wobserver,  
4   metrics: [  
5     additional: [  
6       marbles: {  
7         &ExtendingWobserver.marbles/0,  
8         :gauge,  
9         "Marbles"  
10      },  
11    ],  
12    generators: [  
13      &ExtendingWobserver.generator/0,  
14    ]  
15  ]
```





Add additional pages or generate pages during runtime by setting the config.

## Page options:

`api_only` Numeric, should only increase.

`refresh` Refresh time factor, 0 for no refresh, higher is slower.



Page data can be given in different formats, but it all must be JSON encodable.

### **Page data:**

- map displayed as key-value table.

- list of maps displayed as table, where each map is a row.

It is also possible to return a map of page data. Each value will be turned into one of the above tables.



## CONFIG.EXS

```
17 config :wobserver,  
18   pages: [  
19     {"Stable", :stable, &ExtendingWobserver.stable/0}  
20   ]
```



# Extending: Library Register

extending\_wobserver.ex

```
54 def register do
55   if Code.ensure_loaded(Wobserver) == {:module, Wobserver}
56     Wobserver.register :page,
57                       {"Stable",
58                       :stable,
59                       &ExtendingWobserver.stable/0}
60     Wobserver.register :metric,
61                       [&ExtendingWobserver.generator/0]
62   end
63 end
```



# Extending: Library

Ok?

Does this work? Even for users without `:wobserver`?



# Extending: Library

Ok?

Does this work? Even for users without `:wobserver`?

*kind-a*



Use `apply/3` or...

Add the following line to `project/0` in your mix file

`mix.exs`

```
11       xref: [exclude: [Wobserver]],
```

# A look into the code





# A look into the code:

## Structure

lib/system

All system statistics.

lib/util

All `:observer` functionality.

lib/util/metrics

All metric formatting.

lib/util/node

Node discovery and inter-connections.

src

All web assets.

gulp.js

The web assets build pipeline.

# Reflection



# Reflection:

## Mistakes made

- ❑ Too many mixed concerns.
  - ❑ Interconnecting nodes
  - ❑ Web interface
  - ❑ Metrics
  - ❑ System monitoring
- ❑ Bad websocket handling
- ❑ Security not really taken into consideration
- ❑ Local nodes...
- ❑ Impractical assets setup

And general Elixir inexperience.



# Reflection:

## Version 0.2?

- ☐ Split concerns.
- ☐ Good node interactivity, through sockets and other methods.
- ☐ Security from the start.
- ☐ Get someone / something to make a nice interface and diagrams.

# Questions



# Questions: Questions?

Questions?

Want to know more about what we do with Elixir @ Square Enix?

Just drop by for a chat.

Want to see the code samples? Download the slides?

**See:** *[github.com/ianLuites/wobserver-elixirconf-2017](https://github.com/ianLuites/wobserver-elixirconf-2017)*