Building Services With webmachine

Kevin Smith

OPSCODE

RULE THE CLOUD

OPSCODE

Chef

- Infrastructure as code
- Describe server config using Ruby DSL
- Client/Server. Nodes (your servers) run chef-client, talk to Chef server
- It's awesome.

RULE THE CLOUD

OPSCODE

Chef Server API

- Merb, Ruby, Unicorn, Nginx
- Stateless, horizontally scalable
- Talks to
  - CouchDB,
  - authorization service (Erlang),
  - Solr

RULE THE CLOUD

Typical Chef Server API Request

1. User public key for authentication
2. Node data from CouchDB (median 22K, 3rd Qu. 44K)
3. Authoirzation check
4. POST, GET, PUT, DELETE

How much RAM should it use?

60 req/sec × 44K =
2.7MB

2.7MB data + code + copies... 27MB?

OPSCODE

100MB

Concurrency?
One request per worker.

204 MB

per unicorn worker

12 workers per server

8 servers

$12 \times 204\ MB = 2.4\ GB$

$8 \times 2.4\ GB =$

19.2 GB

for pulling JSON out of a database and returning it

What is webmachine?

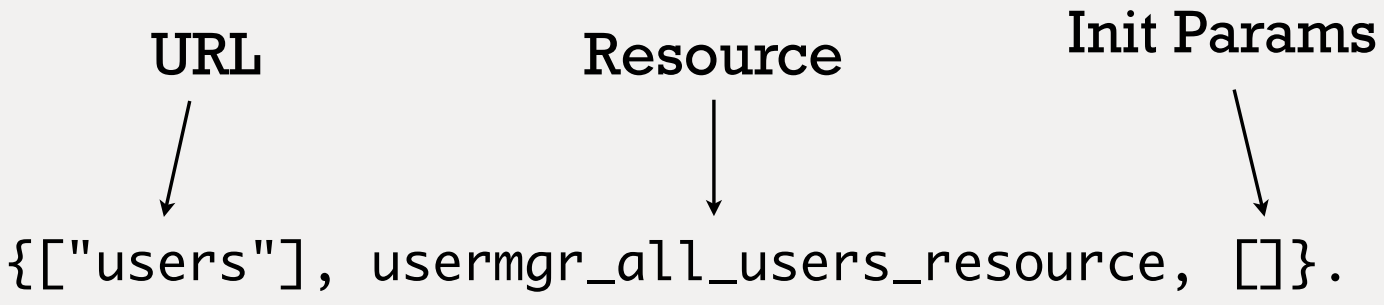
- RESTful behavior "engine"
- Graphical Debugger
- Good performance
- Battle tested in commercial products

webmachine: Major Concepts

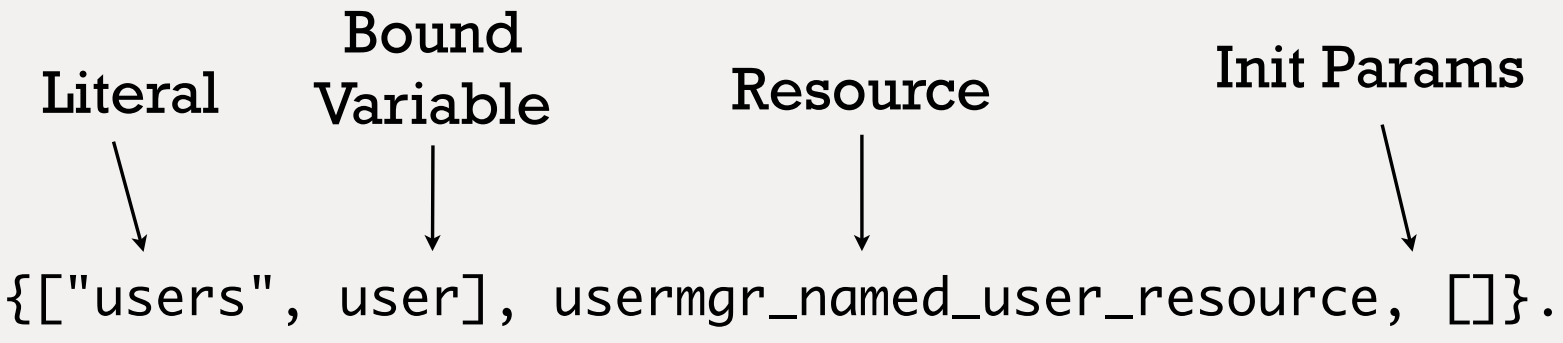
- Resources
- Callbacks
- Routes

OPSCODE

webmachine: Routes



webmachine: Routes



webmachine: Resources

- Corresponds to a single API endpoint
- Implemented as an Erlang module which...
  - Includes `webmachine.hrl`
  - Implements a public `init/1` function
  - .....at a minimum

webmachine: Callbacks

- Functions exported by the resource
- Each callback runs at a designated time
- ~ 30 possible callback functions
- Sane defaults automatically used

Code

Conditional GETs

- Reduces server-to-client traffic
- Improves performance
- **A REAL PAIN** with webmachine

Conditional GETs

```
-export([generate_etag/2]).
.
.
.

generate_etag(Req, #state{data=Data}) ->
    mochihex:to_hex(crypto:md5(Data)).
```

OPSCODE

Compressed Content

- Reduces traffic
- Faster data transmission times
- Snappier user experience
- **A TOTAL NIGHTMARE** with webmachine

Compressed Content

```
-export([encodings_provided/2]).
.
.
.

encodings_provided(Req, State) ->
    [{“gzip”, fun(X) -> zlib:gzip(X) end},
     {“identity”, fun(X) -> X end}].
```

Code

DRYing Up Endpoints

Boilerplate: A Cautionary Tale

You write a couple of endpoints and they work great.

RULE THE CLOUD

Boilerplate: A Cautionary Tale

Your users are happy.

RULE THE CLOUD

Boilerplate: A Cautionary Tale

Your boss is happy.

RULE THE CLOUD

Boilerplate: A Cautionary Tale

You're happy.

RULE THE CLOUD

Boilerplate: A Cautionary Tale

Life is **GOOD.**

RULE THE CLOUD

Then....

Boilerplate: A Cautionary Tale

The world wakes up and notices your app.

RULE THE CLOUD

Boilerplate: A Cautionary Tale

User signups increase by an order of magnitude.

API use goes through the roof.

RULE THE CLOUD

Boilerplate: A Cautionary Tale

Servers catch fire. Network connections are choked with traffic.

RULE THE CLOUD

OPSCODE

Boilerplate: A Cautionary Tale

RULE THE CLOUD

Boilerplate: A Cautionary Tale

"We need to monitor API performance", you say to your team.

Boilerplate: A Cautionary Tale

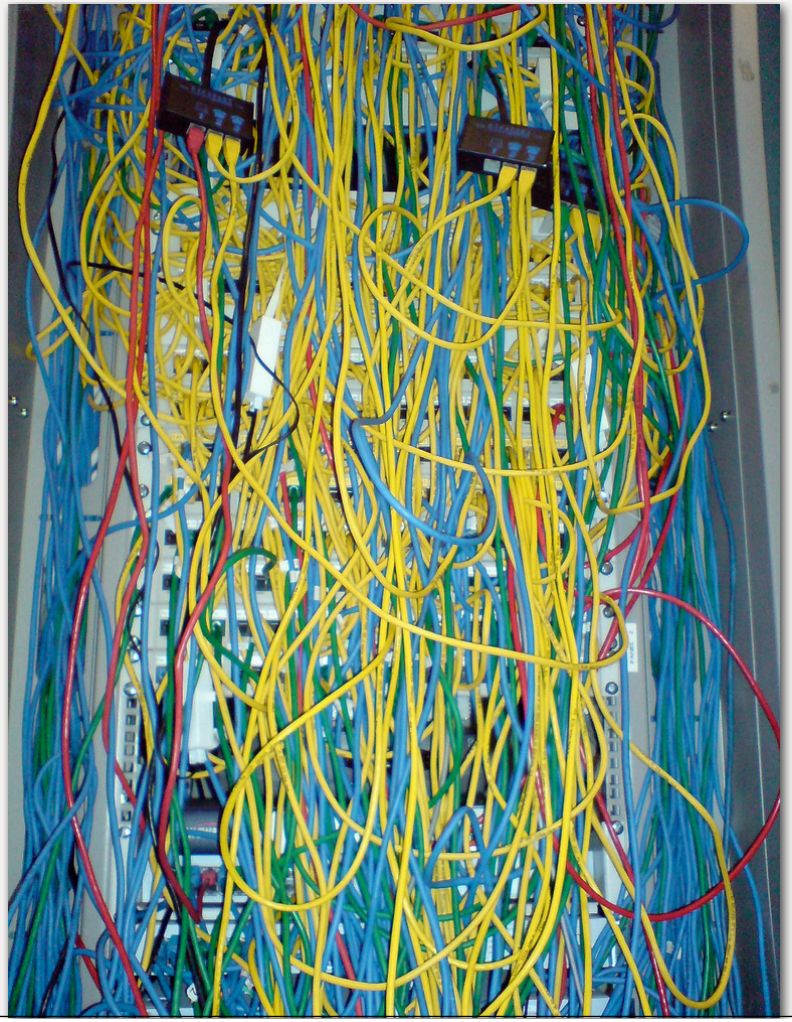
"We'll instrument all the endpoints to log to graphite!"

RULE THE CLOUD

Boilerplate: A Cautionary Tale

Now you have **N** problems.

RULE THE CLOUD



OPSCODE

Boilerplate Code Sucks

- Repetitive
- Copy pasta errors
- Repetitive
- PITA
- Repetitive

Introducing mixer

- Basic mixins for Erlang
- Reuse entire functions with minimal typing
- No runtime overhead (parse transform)
- Explicit and easy to reason about

Code

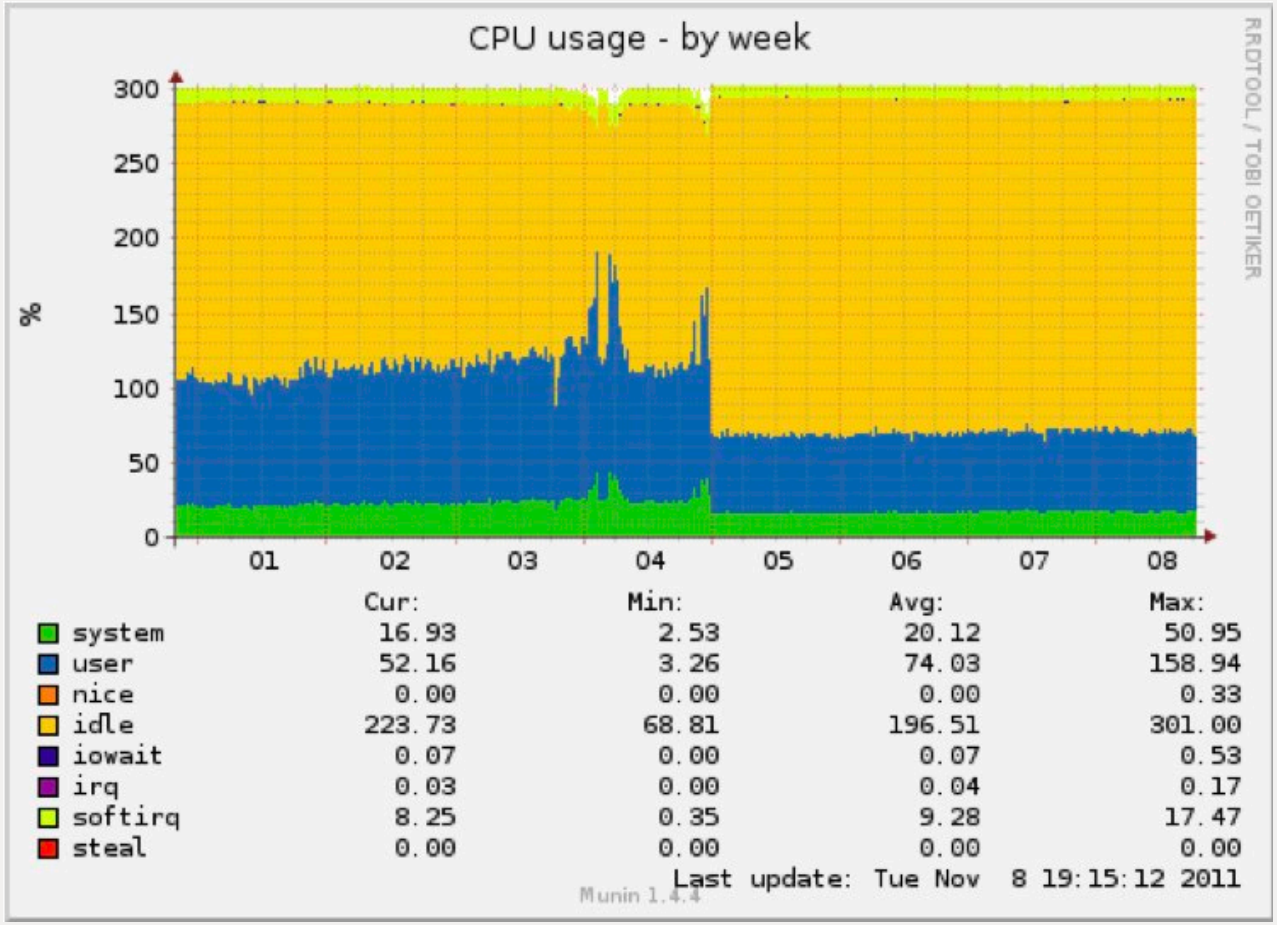
How did we do?

	Erlang	Ruby
idle	19MB	100MB
loaded	75MB	204MB

Erlang	Ruby
600MB	19.2GB

OPSCODE

CPU Usage on Chef Server



Thank You

Email: k@opscode.com

Twitter: kevsmith

RULE THE CLOUD