Load Testing At Scale

Aaron Seigo – aseigo@mykolab.com - 12/2017

Who is this fella?

Initial release: Haida Gwaii, Canada (up by Alaska), 1975

Software developer: 1992

Linux user: 1994

Free software career: 1997

KDE: 2000

Free Software Hippy: 2004

Currently: Nomoko AG

Load Testing At Scale

Aaron Seigo – aseigo@mykolab.com - 12/2017

The Bigger The System …

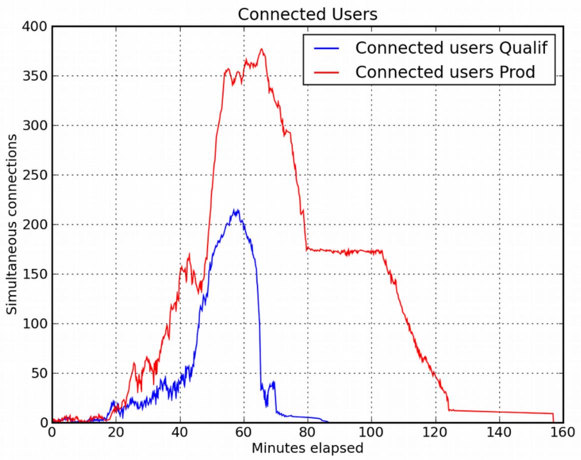
What to measure?

User sessions

- Completes a session before starting the next
- Limiting is based on number of users
- Built to emulate interaction

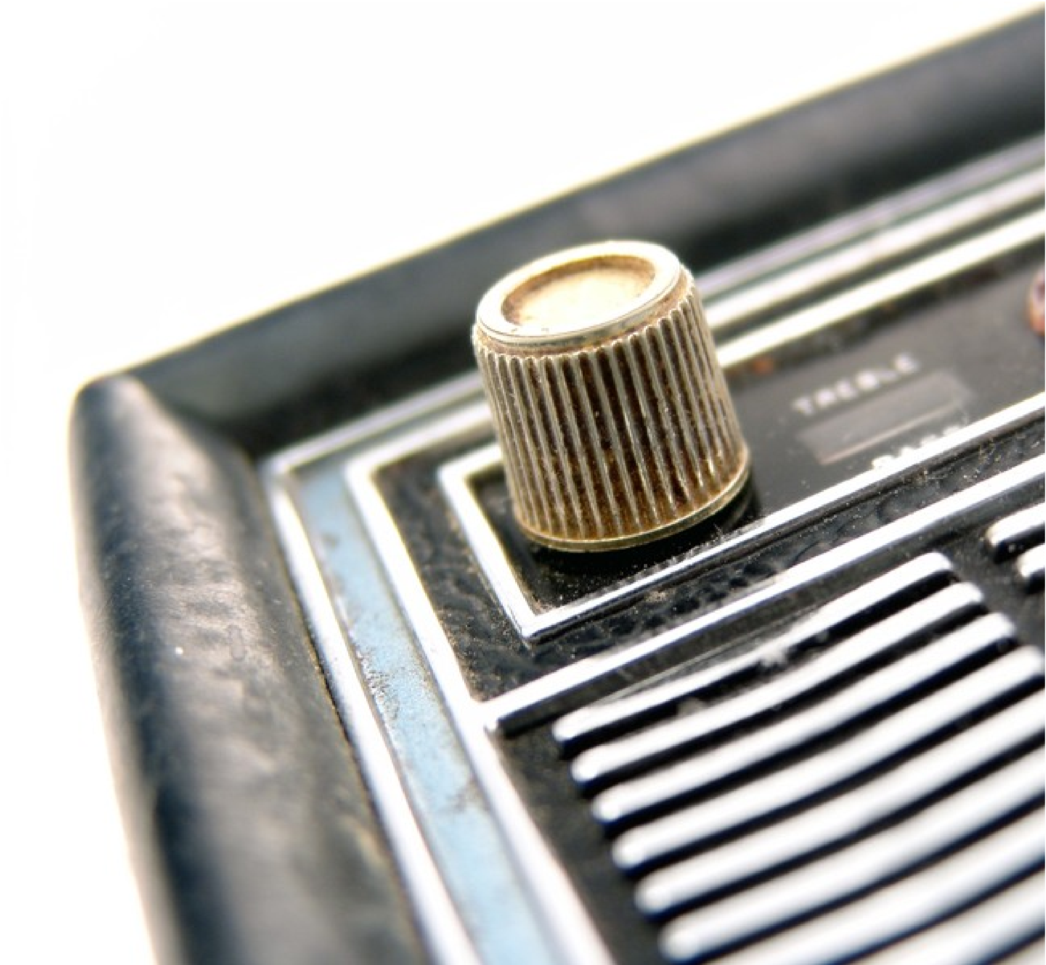
Requests

- Launches as many requests in parallel as
- Built to emulate load



How to measure?

- Quantifiable
- Repeatable
- Scalable
- Automation friendly



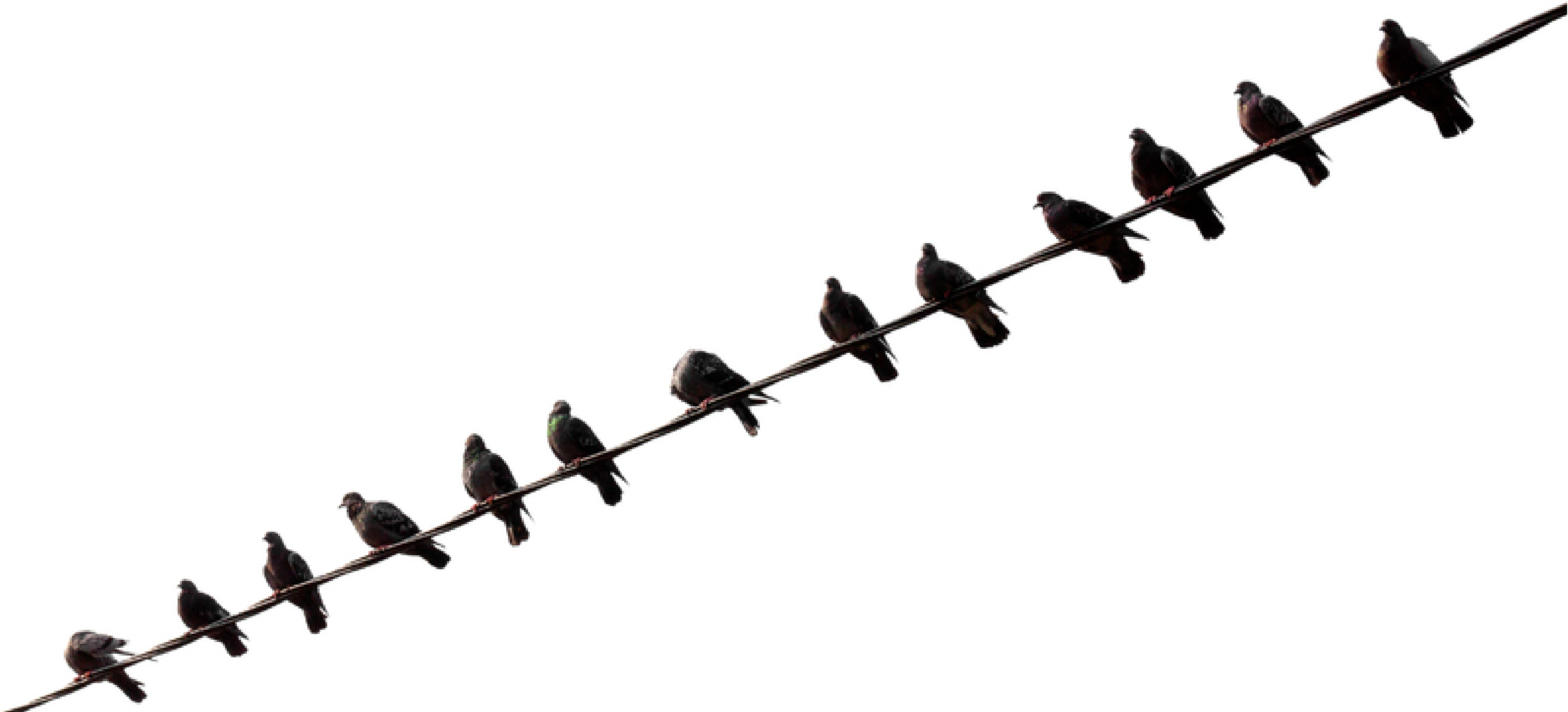
The Heroes Of Our Story

The Heroes Of Our Story

MZBENCH

basho_bench

They all have something in common…



They all have something in common…

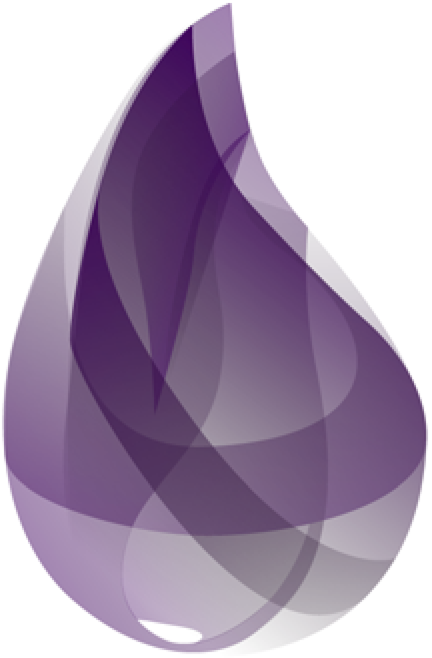
Erlang … ?

Yes, Erlang!

- Has a 30+ year history
- Majority of mobile data processed by Erlang

WhatsApp
bet365
Pinterest
pagerduty
Julius Bär
RabbitMQ
SQUARE ENIX
ejabberd
ERICSSON
CouchDB
relax
b/r
bleacher report

… and its supermodel child



elixir

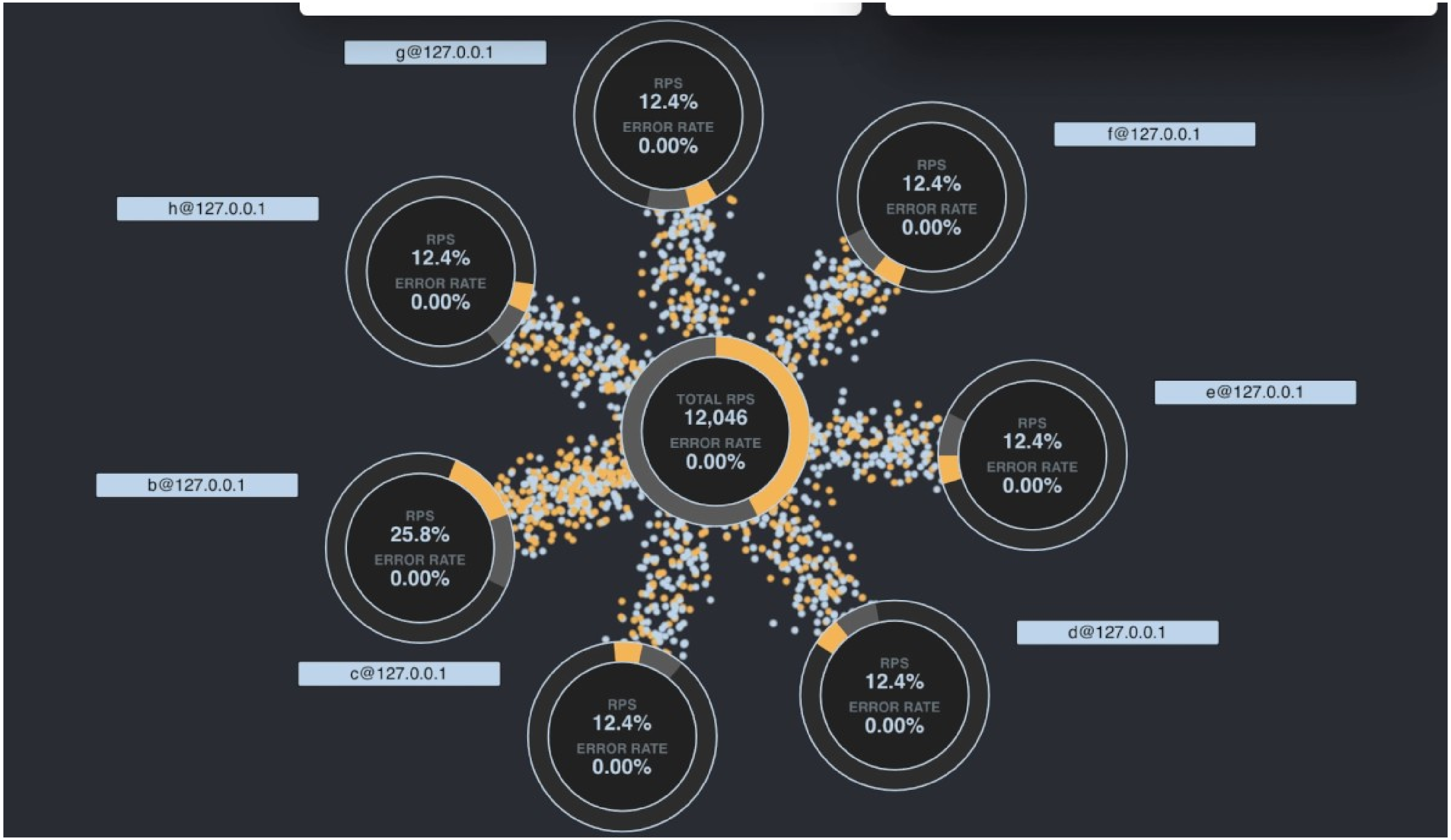
Why Erlang / Elixir?

- Concurrency and clustering built into the language

```
input
|> Stream.chunk_every(3, 1, :discard)
|> Flow.from_enumerable()
|> Flow.partition()
|> Flow.reduce(fn -> %{} end,
               fn v, acc -> Map.update(acc, v, 1, &(&1 + 1)) end)
|> Flow.reject(fn {_, 1} -> true
                       _ -> false end)
```

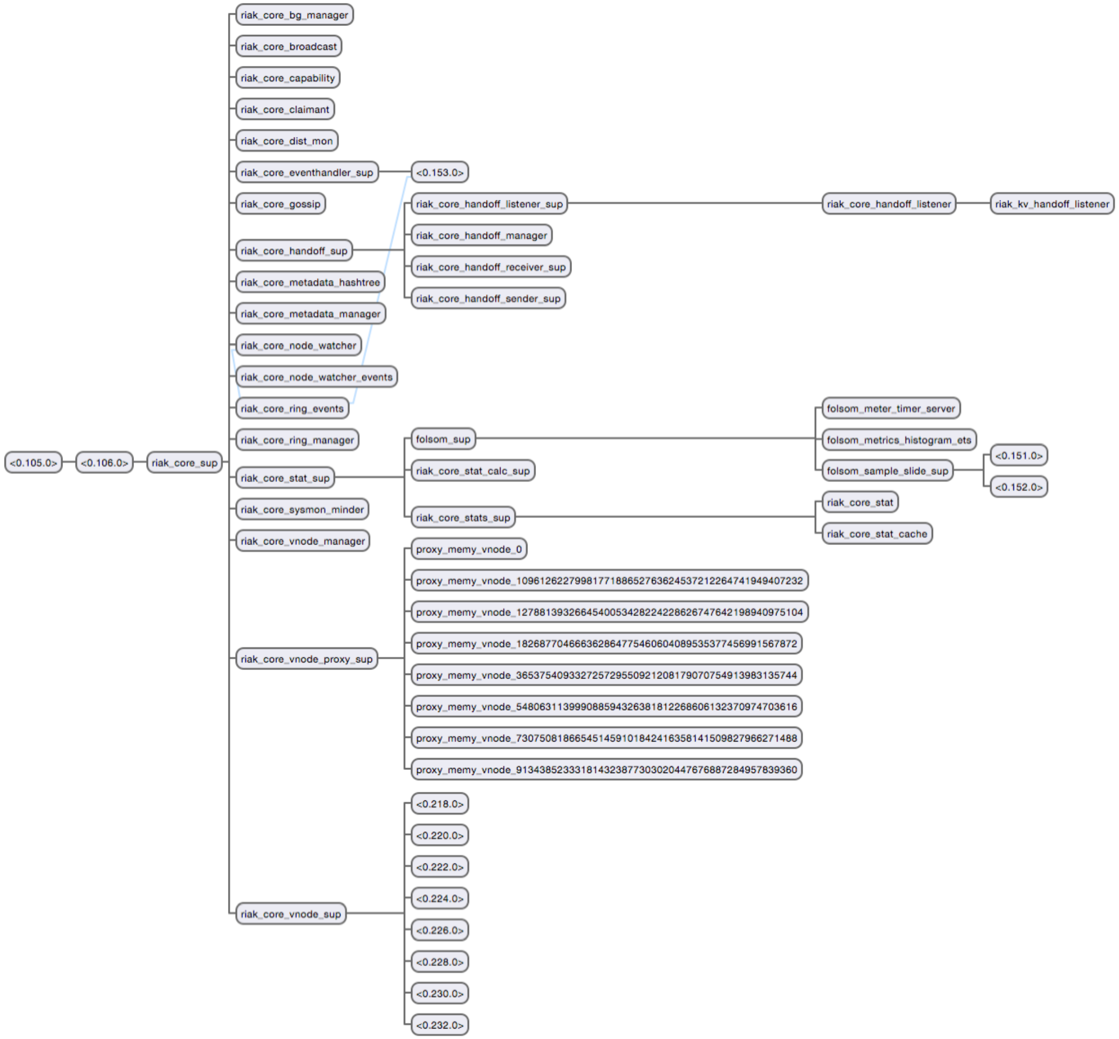
Why Erlang / Elixir?

- Concurrency and clustering built into the language



Why Erlang / Elixir?

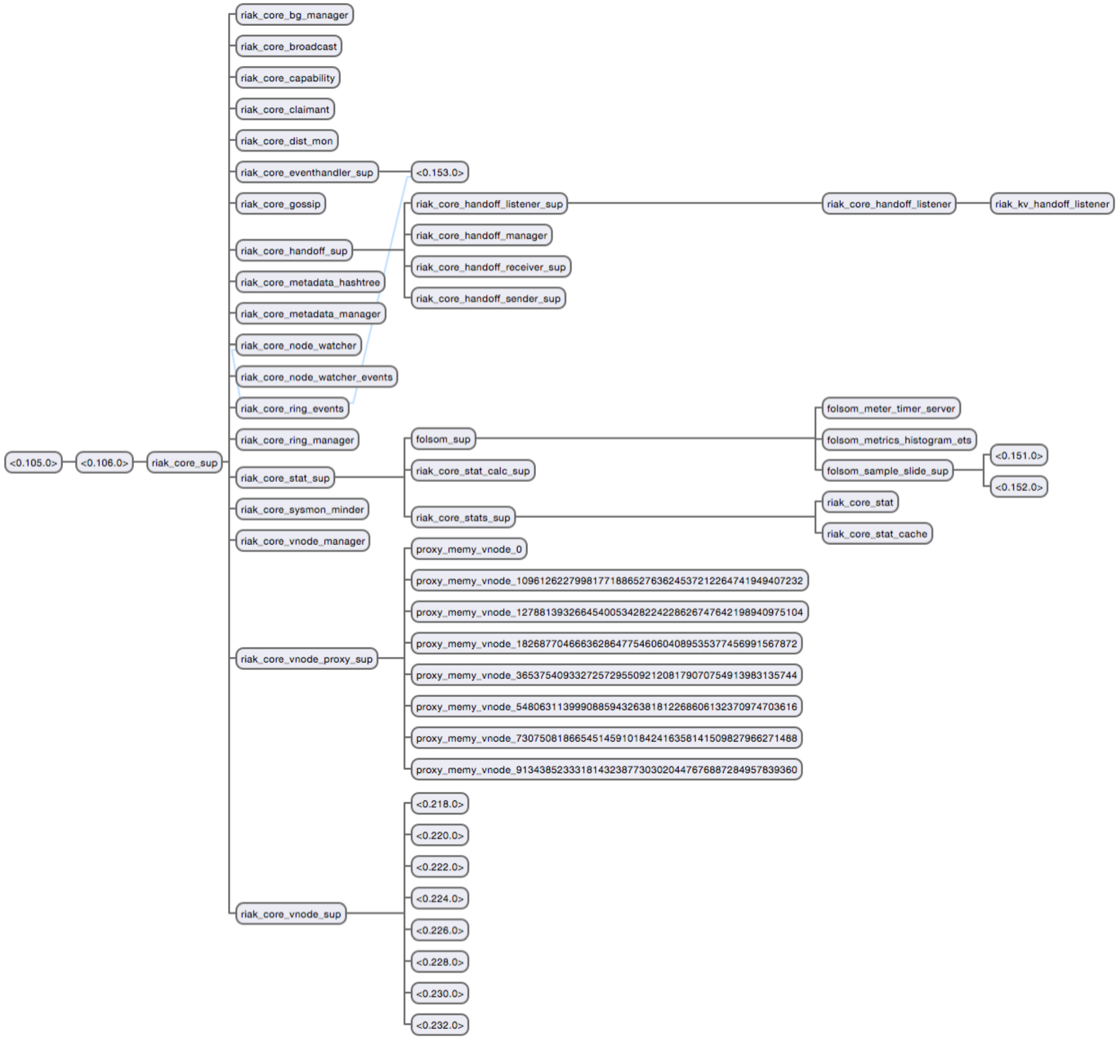
- Concurrency and clustering built into the language



<0.105.0>
<0.106.0>
riak_core_sup
riak_core_bg_manager
riak_core_broadcast
riak_core_capability
riak_core_claimant
riak_core_dist_mon
riak_core_eventhandler_sup
<0.153.0>
riak_core_gossip
riak_core_handoff_sup
riak_core_handoff_listener_sup
riak_core_handoff_listener
riak_kv_handoff_listener
riak_core_handoff_manager
riak_core_handoff_receiver_sup
riak_core_handoff_sender_sup
riak_core_metadata_hashtree
riak_core_metadata_manager
riak_core_node_watcher
riak_core_node_watcher_events
riak_core_ring_events
riak_core_ring_manager
riak_core_stat_sup
folsom_sup
folsom_meter_timer_server
folsom_metrics_histogram_ets
folsom_sample_slide_sup
<0.151.0>
<0.152.0>
riak_core_stat_calc_sup
riak_core_stats_sup
riak_core_stat
riak_core_stat_cache
riak_core_sysmon_minder
riak_core_vnode_manager
riak_core_vnode_proxy_sup
proxy_memy_vnode_0
proxy_memy_vnode_1096126227998177188652763624537212264741949407232
proxy_memy_vnode_1278813932664540053428224228626747642198940975104
proxy_memy_vnode_182687704666362864775460604089535377456991567872
proxy_memy_vnode_365375409332725729550921208179070754913983135744
proxy_memy_vnode_548063113999088594326381812268606132370974703616
proxy_memy_vnode_730750818665451459101842416358141509827966271488
proxy_memy_vnode_913438523331814323877303020447676887284957839360
riak_core_vnode_sup
<0.218.0>
<0.220.0>
<0.222.0>
<0.224.0>
<0.226.0>
<0.228.0>
<0.230.0>
<0.232.0>

Why Erlang / Elixir?

- Resilience and durability are built into the language



<0.105.0>
<0.106.0>
riak_core_sup
riak_core_bg_manager
riak_core_broadcast
riak_core_capability
riak_core_claimant
riak_core_dist_mon
riak_core_eventhandler_sup
<0.153.0>
riak_core_gossip
riak_core_handoff_sup
riak_core_handoff_listener_sup
riak_core_handoff_listener
riak_kv_handoff_listener
riak_core_handoff_manager
riak_core_handoff_receiver_sup
riak_core_handoff_sender_sup
riak_core_metadata_hashtree
riak_core_metadata_manager
riak_core_node_watcher
riak_core_node_watcher_events
riak_core_ring_events
riak_core_ring_manager
riak_core_stat_sup
folsom_sup
folsom_meter_timer_server
folsom_metrics_histogram_ets
folsom_sample_slide_sup
<0.151.0>
<0.152.0>
riak_core_stat_calc_sup
riak_core_stats_sup
riak_core_stat
riak_core_stat_cache
riak_core_sysmon_minder
riak_core_vnode_manager
riak_core_vnode_proxy_sup
proxy_memy_vnode_0
proxy_memy_vnode_1096126227998177188652763624537212264741949407232
proxy_memy_vnode_1278813932664540053428224228626747642198940975104
proxy_memy_vnode_182687704666362864775460604089535377456991567872
proxy_memy_vnode_365375409332725729550921208179070754913983135744
proxy_memy_vnode_548063113999088594326381812268606132370974703616
proxy_memy_vnode_730750818665451459101842416358141509827966271488
proxy_memy_vnode_913438523331814323877303020447676887284957839360
riak_core_vnode_sup
<0.218.0>
<0.220.0>
<0.222.0>
<0.224.0>
<0.226.0>
<0.228.0>
<0.230.0>
<0.232.0>

Why Erlang / Elixir?

- Highly predictable latency
  - Pre-emptive multitasking
  - Long-running uninteruptables (>1ms) scheduled separately
- Easy to extend
  - Hot code loading, even over the network!
  - Java, C/C++, Rust, Python, Ruby …
- Fantastic APIs for networking and protocol parsing

Why Erlang / Elixir?

- Fantastic developer productivity
  - Benefits of functional programming, pragmatically
  - Hard problems (e.g. threading) handled for us
  - Excellent tooling
    - Package management
    - Build and deploy tools
    - Testing frameworks

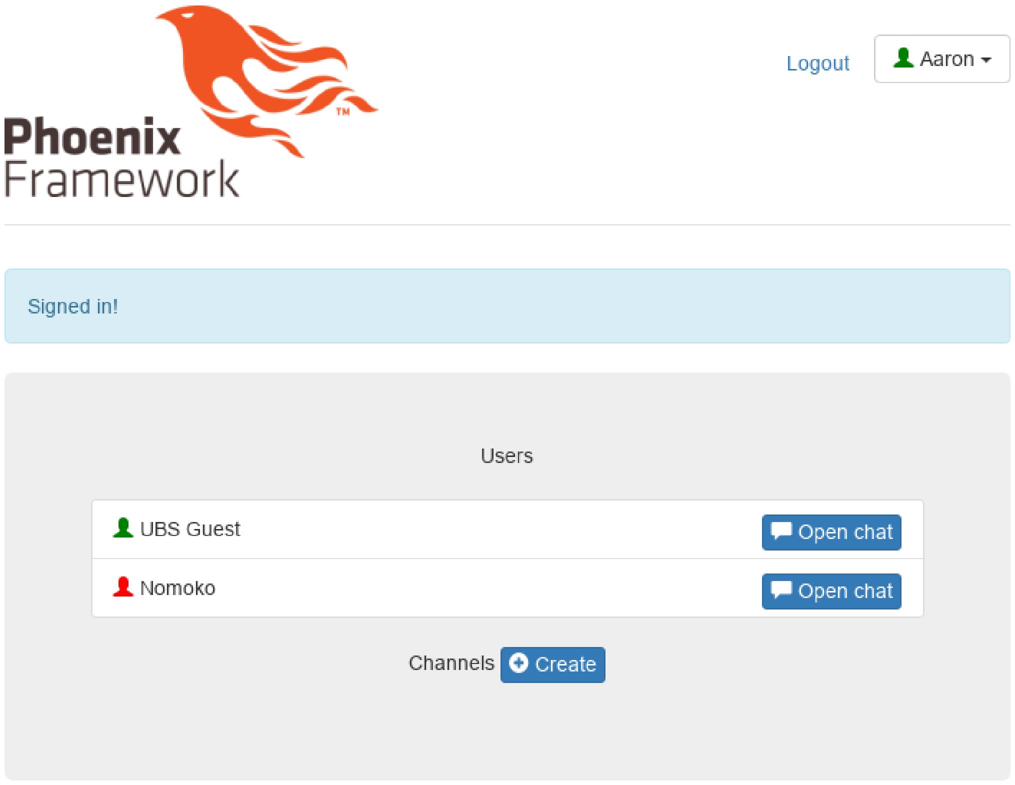
Erlang / Elixir!

Erlang / Elixir!

Erlang & Elixir !

Testing An Application

- VM with 2 vCPU / 2 GB RAM for the application



Signed in!

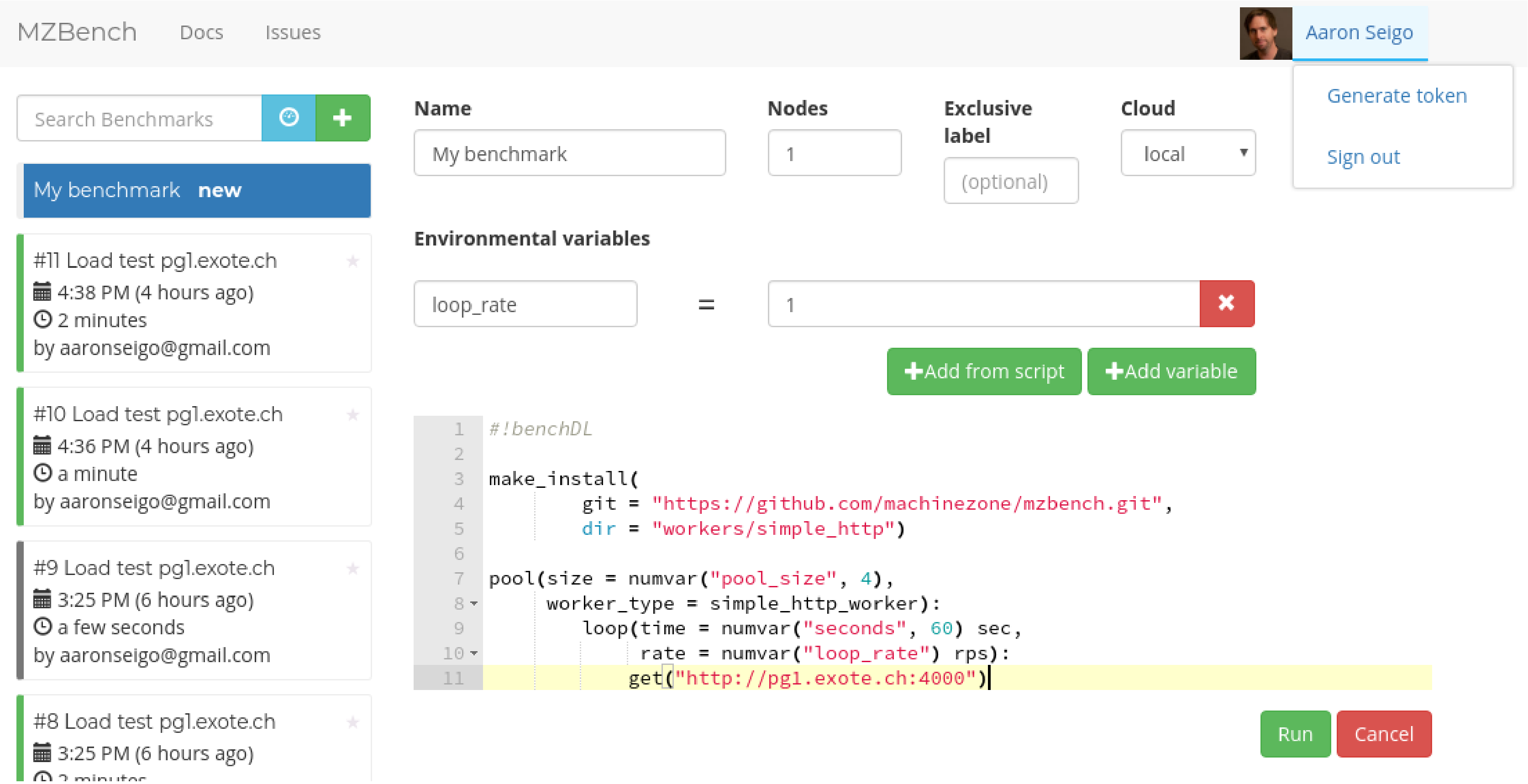
Users

UBS Guest

Nomoko

Testing An Application

- VM with 4 vCPU / 4 GB RAM for the load tests



My benchmark **new**

#11 Load test pgl.exote.ch
4:38 PM (4 hours ago)
2 minutes
by aaronseigo@gmail.com

#10 Load test pgl.exote.ch
4:36 PM (4 hours ago)
a minute
by aaronseigo@gmail.com

#9 Load test pgl.exote.ch
3:25 PM (6 hours ago)
a few seconds
by aaronseigo@gmail.com

#8 Load test pgl.exote.ch
3:25 PM (6 hours ago)

Name: My benchmark

Nodes: 1

Exclusive label: (optional)

Cloud: local

Environmental variables

loop_rate = 1

```
#!benchDL

make_install(
        git = "https://github.com/machinezone/mzbench.git",
        dir = "workers/simple_http")

pool(size = numvar("pool_size", 4),
     worker_type = simple_http_worker):
        loop(time = numvar("seconds", 60) sec,
             rate = numvar("loop_rate") rps):
            get("http://pgl.exote.ch:4000")
```

A distributed Load Testing Tool
TSUNG

A distributed Load Testing Tool

TSUNG

- Models user sessions
- Cluster support as well as local concurrency
- Multi-protocol
  - HTTP, Websocket, WebDAV, XMPP, PostgreSQL, MySQL, AMQP, MQTT, LDAP, raw sockets
- Record-for-replay
- XML configuration, amazingly flexible
- Fantastic documentation
- Latest release 1.7.0 in August 2017 (1st release in 2001!)

A distributed Load Testing Tool

TSUNG

- Jabber/XMPP
  - 90,000 users on 10 1.5Ghz UltraSPARC IIIi CPUs
  - 2,000,000 users on a m4.10xlarge (40 vCPU / 160GB)
- HTTP and HTTPS
  - 22k+ websocket connections on 4-vCPU/15GB RAM, scaling linearly over a 15 node cluster
  - ~10k requests/second on a m1.small
  - 60k+ websocket connections on 2 Digital Ocean droplets with 10 vCPUs
  - 10 million simultaneous users running on a 75-system cluster, generating more than one million requests per second

TSUNG

A distributed Load Testing Tool

Setting up a cluster is straight-forward:

```
<clients>
  <client host="client1" weight="1" maxusers="800">
    <ip value="10.9.195.12"></ip>
    <ip value="10.9.195.13"></ip>
  </client>
  <client host="client2" weight="3" maxusers="600" cpu="2"/>
</clients>

<servers>
  <server host="server1" port="80" type="tcp" weight="4"></server>
  <server host="server2" port="80" type="tcp" weight="1"></server>
</servers>
```

Defining a wave of users:

```
<arrivalphase phase="1" duration="10" unit="minute">
  <users maxnumber="100" interarrival="0.1" unit="second"></users>
</arrivalphase>

<arrivalphase phase="2" duration="10" unit="minute">
  <users maxnumber="200" arrivalrate="10" unit="second"></users>
</arrivalphase>
```

Sessions definitions are also XML:

```
<sessions>
  <session name="load" weight="1" type="ts_http">
    <request>
      <http url="https://pg1.exote.ch/" method="GET" />
    </request>
   </session>
</sessions>
```

A distributed Load Testing Tool

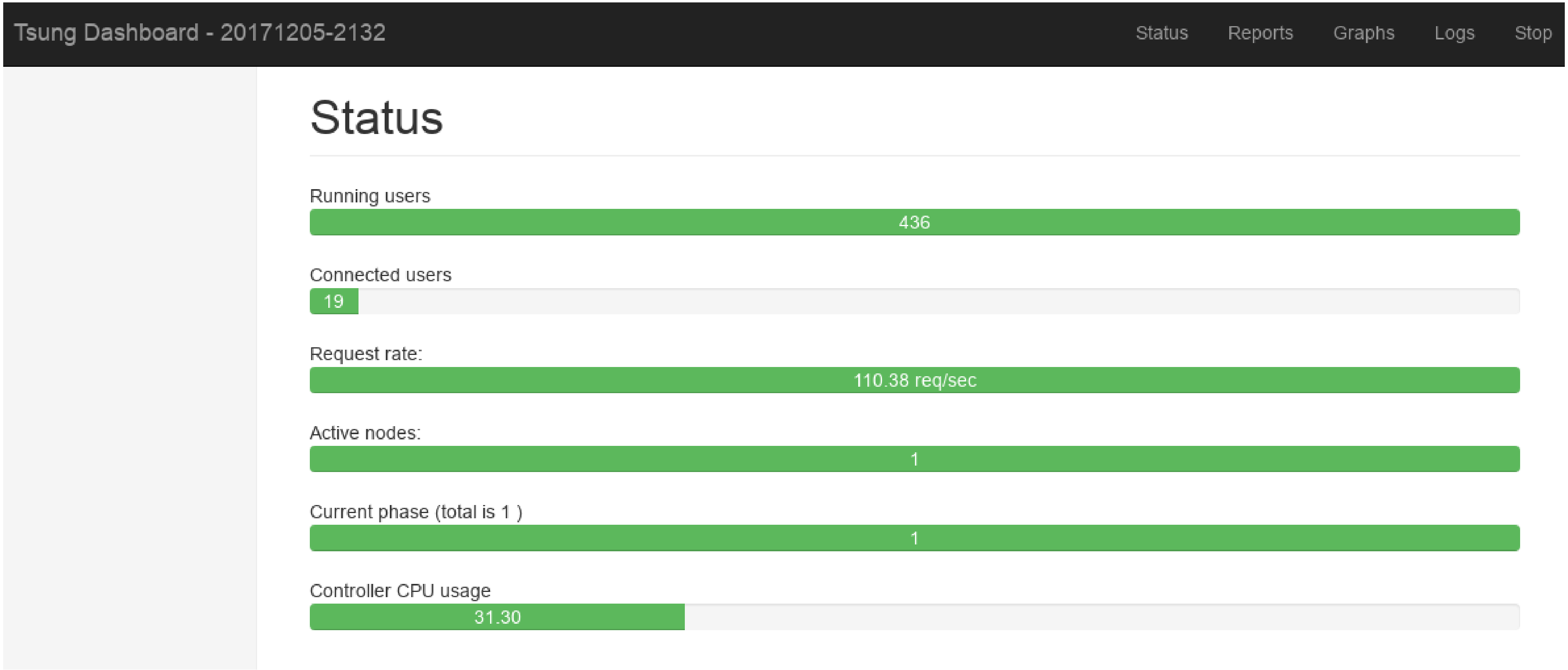
TSUNG

An amazing array of options:

- Load progressions
- Think times
- SSL cyphers and reuse
- Retries, timeouts, etc. etc.
- Monitoring (e.g. SNMP)
- ....

TSUNG

A distributed Load Testing Tool



Status

Running users
436

Connected users
19

Request rate:
110.38 req/sec

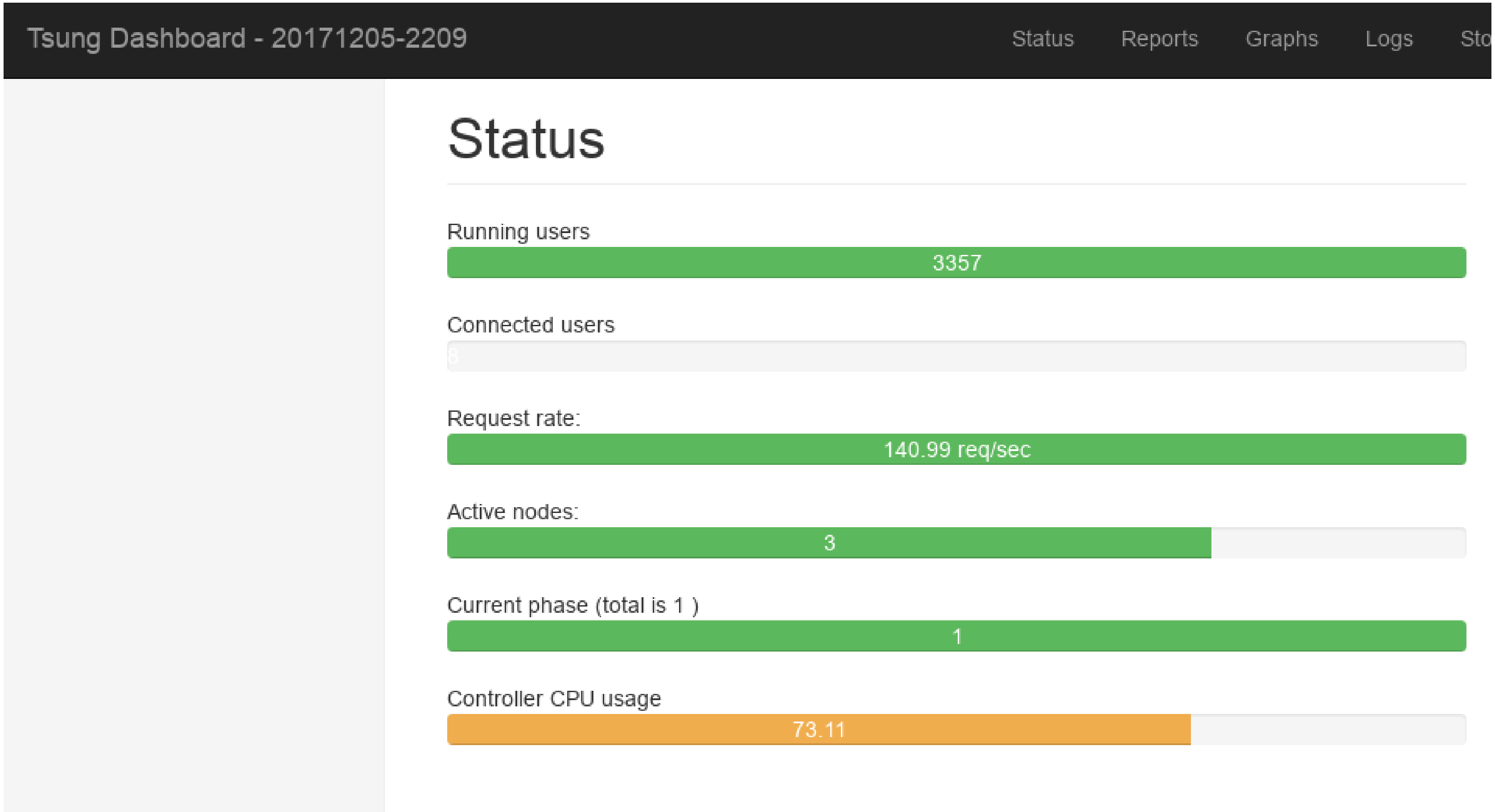
Active nodes:
1

Current phase (total is 1)
1

Controller CPU usage
31.30

A distributed Load Testing Tool

TSUNG



Tsung Dashboard - 20171205-2209

Status

Running users
3357

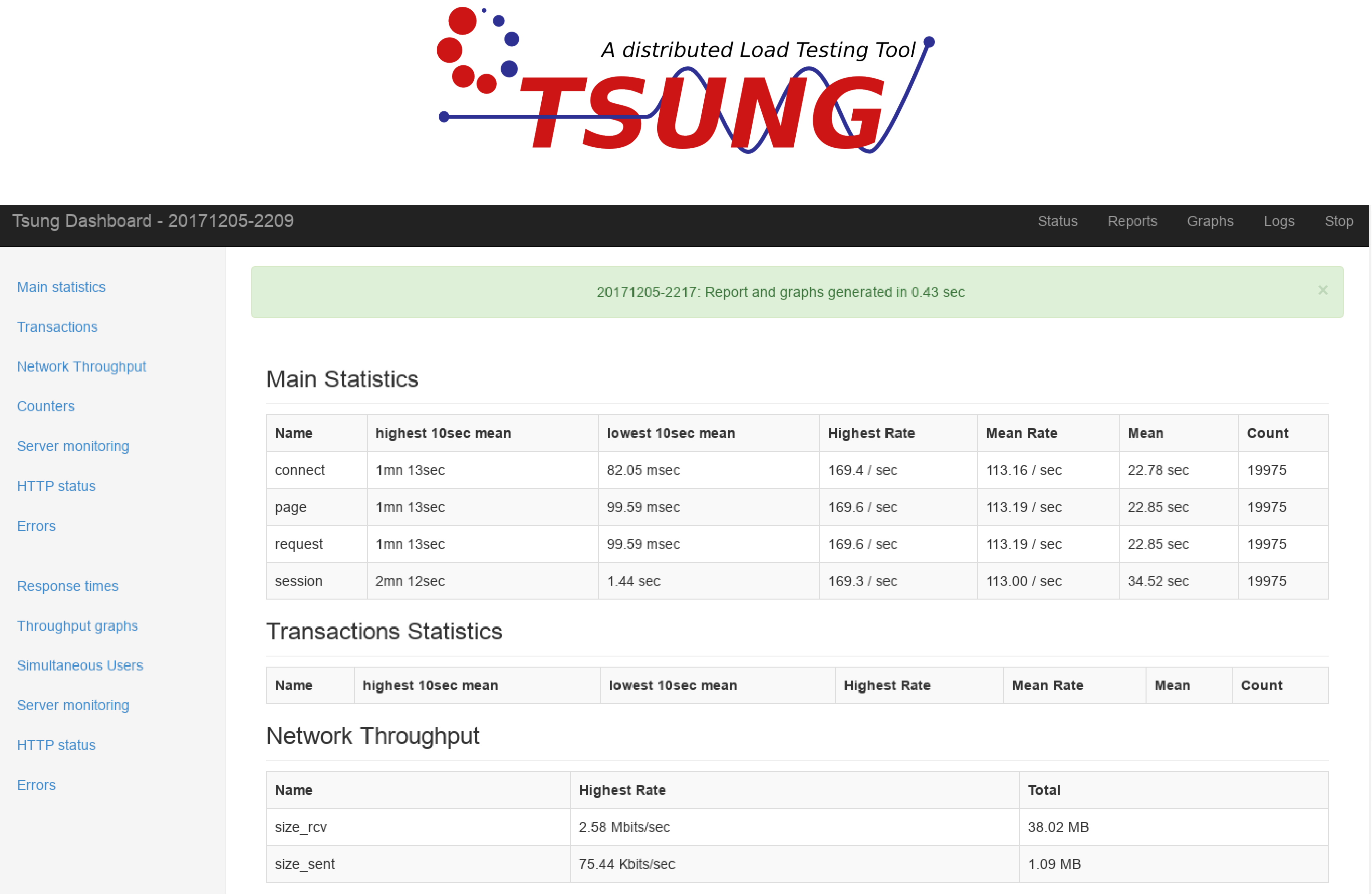
Connected users
8

Request rate:
140.99 req/sec

Active nodes:
3

Current phase (total is 1)
1

Controller CPU usage
73.11



Tsung Dashboard - 20171205-2209

20171205-2217: Report and graphs generated in 0.43 sec

Main Statistics

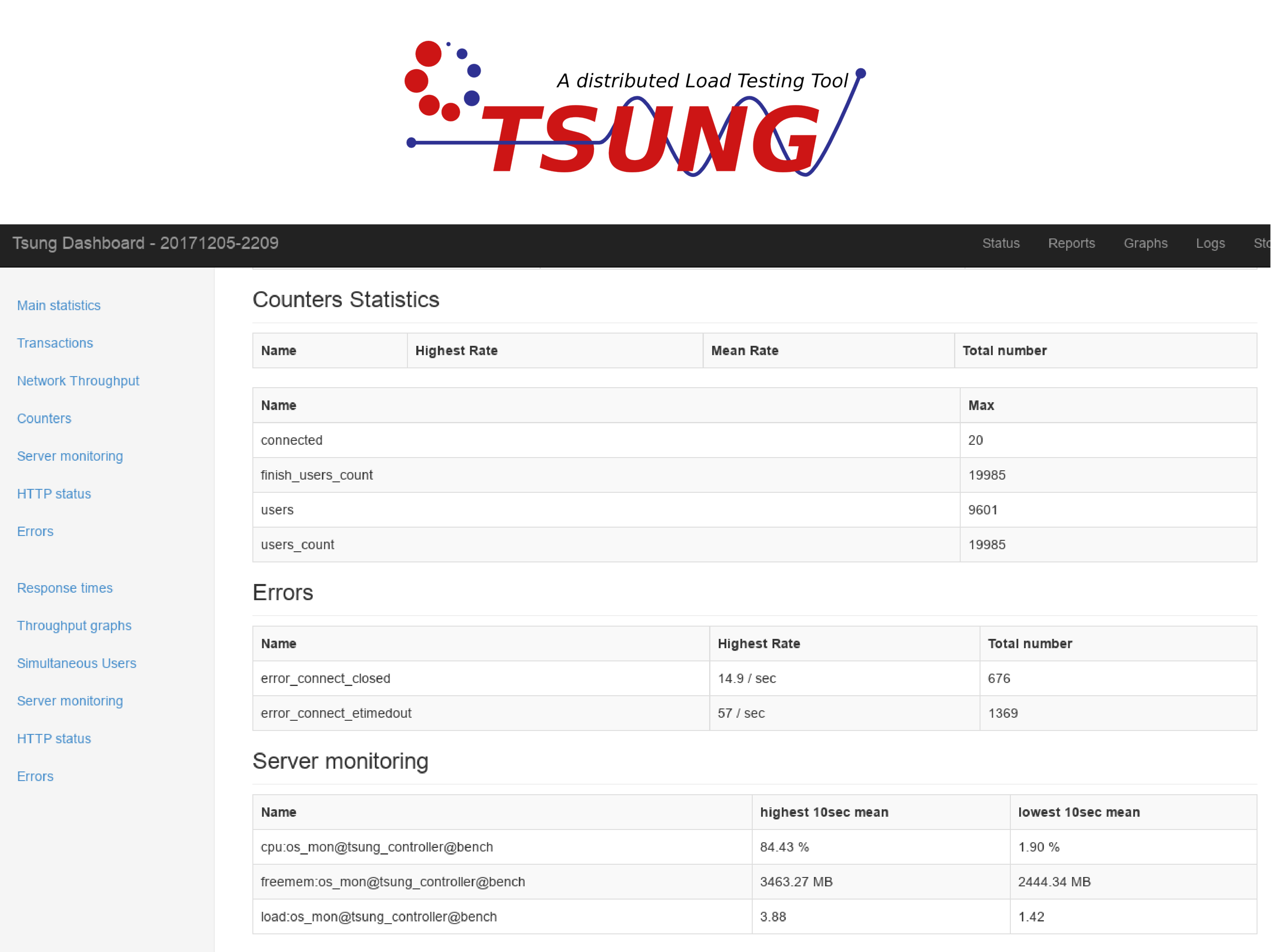
Name	highest 10sec mean	lowest 10sec mean	Highest Rate	Mean Rate	Mean	Count
connect	1mn 13sec	82.05 msec	169.4 / sec	113.16 / sec	22.78 sec	19975
page	1mn 13sec	99.59 msec	169.6 / sec	113.19 / sec	22.85 sec	19975
request	1mn 13sec	99.59 msec	169.6 / sec	113.19 / sec	22.85 sec	19975
session	2mn 12sec	1.44 sec	169.3 / sec	113.00 / sec	34.52 sec	19975

Transactions Statistics

Name	highest 10sec mean	lowest 10sec mean	Highest Rate	Mean Rate	Mean	Count

Network Throughput

Name	Highest Rate	Total
size_rcv	2.58 Mbits/sec	38.02 MB
size_sent	75.44 Kbits/sec	1.09 MB



Counters Statistics

Name	Highest Rate	Mean Rate	Total number

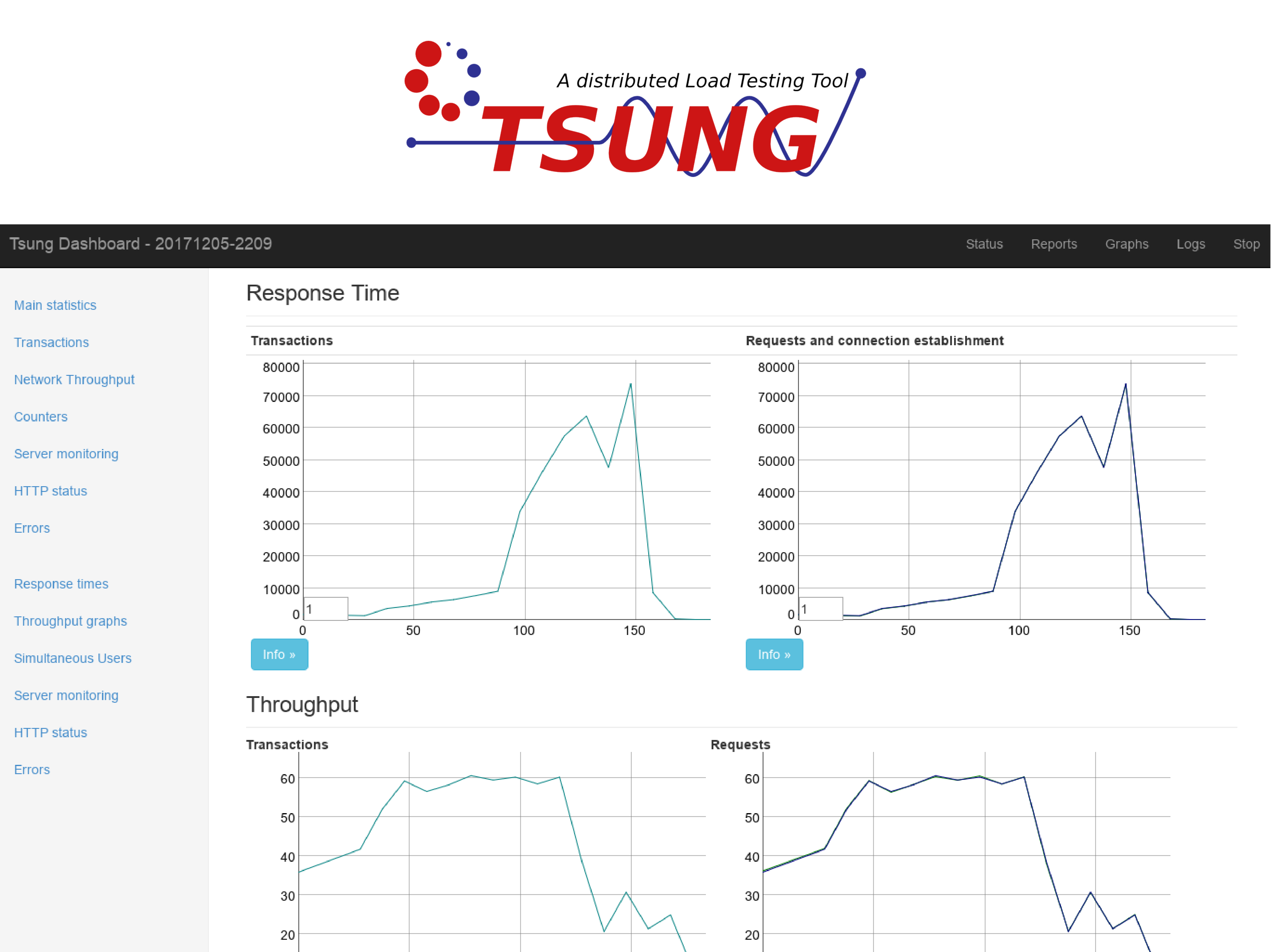
Name	Max
connected	20
finish_users_count	19985
users	9601
users_count	19985

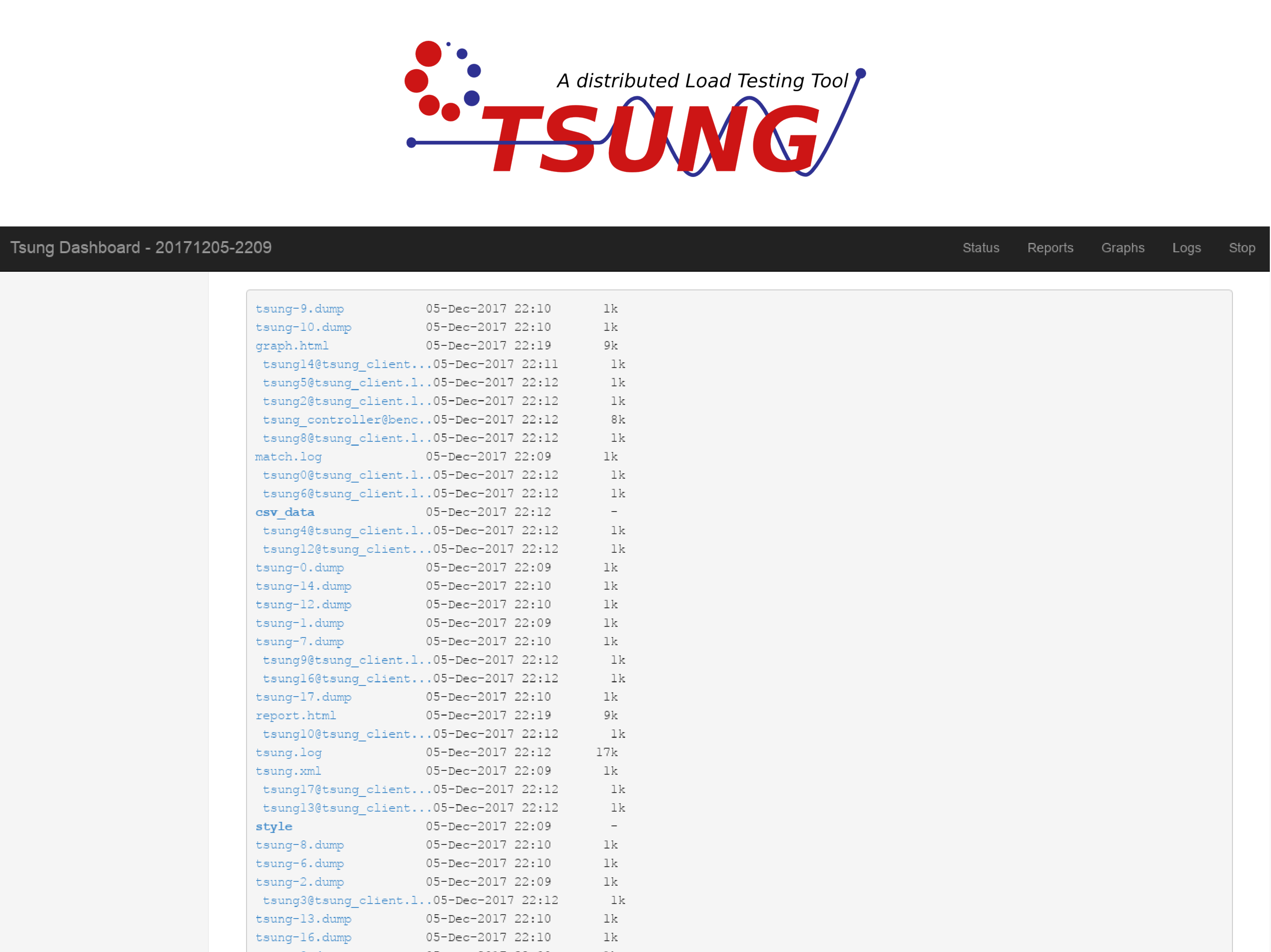
Errors

Name	Highest Rate	Total number
error_connect_closed	14.9 / sec	676
error_connect_etimedout	57 / sec	1369

Server monitoring

Name	highest 10sec mean	lowest 10sec mean
cpu:os_mon@tsung_controller@bench	84.43 %	1.90 %
freemem:os_mon@tsung_controller@bench	3463.27 MB	2444.34 MB
load:os_mon@tsung_controller@bench	3.88	1.42


A distributed Load Testing Tool
TSUNG
Tsung Dashboard - 20171205-2209
Status
Reports
Graphs
Logs
Stop
Main statistics
Transactions
Network Throughput
Counters
Server monitoring
HTTP status
Errors
Response times
Throughput graphs
Simultaneous Users
Server monitoring
HTTP status
Errors
Response Time
Transactions
80000
70000
60000
50000
40000
30000
20000
10000
0
1
0
50
100
150
Info »
Requests and connection establishment
80000
70000
60000
50000
40000
30000
20000
10000
0
1
0
50
100
150
Info »
Throughput
Transactions
60
50
40
30
20
Requests
60
50
40
30
20



A distributed Load Testing Tool
TSUNG

Tsung Dashboard - 20171205-2209

Status Reports Graphs Logs Stop

```
tsung-9.dump            05-Dec-2017 22:10      1k
tsung-10.dump           05-Dec-2017 22:10      1k
graph.html              05-Dec-2017 22:19      9k
 tsung14@tsung_client...05-Dec-2017 22:11      1k
 tsung5@tsung_client.l..05-Dec-2017 22:12      1k
 tsung2@tsung_client.l..05-Dec-2017 22:12      1k
 tsung_controller@benc..05-Dec-2017 22:12      8k
 tsung8@tsung_client.l..05-Dec-2017 22:12      1k
match.log               05-Dec-2017 22:09      1k
 tsung0@tsung_client.l..05-Dec-2017 22:12      1k
 tsung6@tsung_client.l..05-Dec-2017 22:12      1k
csv_data                05-Dec-2017 22:12      -
 tsung4@tsung_client.l..05-Dec-2017 22:12      1k
 tsung12@tsung_client...05-Dec-2017 22:12      1k
tsung-0.dump            05-Dec-2017 22:09      1k
tsung-14.dump           05-Dec-2017 22:10      1k
tsung-12.dump           05-Dec-2017 22:10      1k
tsung-1.dump            05-Dec-2017 22:09      1k
tsung-7.dump            05-Dec-2017 22:10      1k
 tsung9@tsung_client.l..05-Dec-2017 22:12      1k
 tsung16@tsung_client...05-Dec-2017 22:12      1k
tsung-17.dump           05-Dec-2017 22:10      1k
report.html             05-Dec-2017 22:19      9k
 tsung10@tsung_client...05-Dec-2017 22:12      1k
tsung.log               05-Dec-2017 22:12     17k
tsung.xml               05-Dec-2017 22:09      1k
 tsung17@tsung_client...05-Dec-2017 22:12      1k
 tsung13@tsung_client...05-Dec-2017 22:12      1k
style                   05-Dec-2017 22:09      -
tsung-8.dump            05-Dec-2017 22:10      1k
tsung-6.dump            05-Dec-2017 22:10      1k
tsung-2.dump            05-Dec-2017 22:09      1k
 tsung3@tsung_client.l..05-Dec-2017 22:12      1k
tsung-13.dump           05-Dec-2017 22:10      1k
tsung-16.dump           05-Dec-2017 22:10      1k
```

A distributed Load Testing Tool

TSUNG

Website

http://tsung.erlang-projects.org

Docs

http://tsung.erlang-projects.org/user_manual/

Git

https://github.com/processone/tsung

MZBENCH

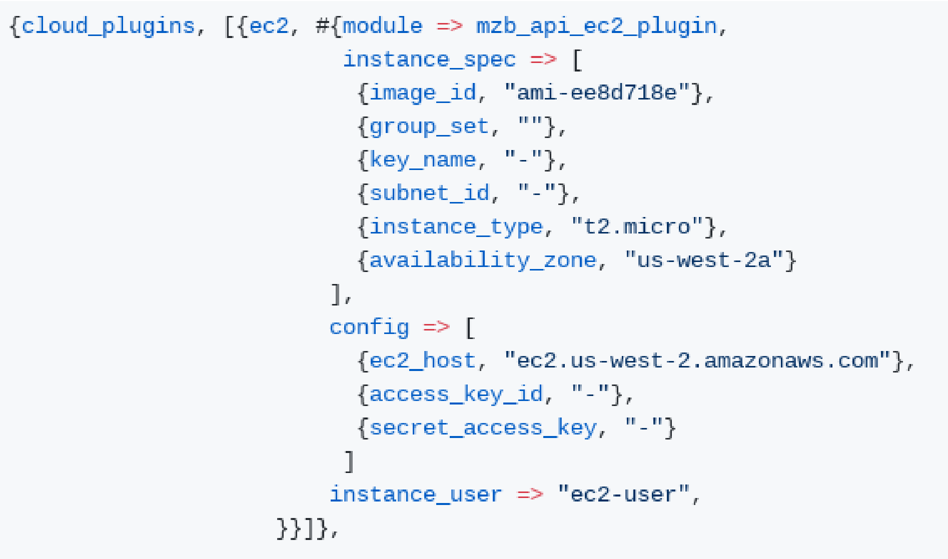
MZBENCH

- Models requests
- Flexible deployment: AWS, docker, rpm/deb, …
- Multi-protocol
  - HTTP, MySQL, PostgreSQL, MongoDB, Cassandra, XMPP, AMQP, raw sockets, shell commands, and TCPKali
- BDL: a Python-ish DSL for test definition
- Great documentation
- Latest release 0.5.2 in April 20017, first in 2015

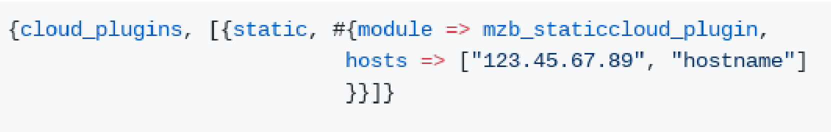
MZBENCH

Clustering is simple:

- mzb_api_ec2_plugin : Allocate hosts from the Amazon EC2 cloud
- mzb_staticcloud_plugin : Allocates hosts from a static pool
- mzb_multicloud_plugin : Allocate hosts from multiple sources by ratio



```
{cloud_plugins, [{ec2, #{module => mzb_api_ec2_plugin,
                         instance_spec => [
                          {image_id, "ami-ee8d718e"},
                          {group_set, ""},
                          {key_name, "-"},
                          {subnet_id, "-"},
                          {instance_type, "t2.micro"},
                          {availability_zone, "us-west-2a"}
                        ],
                        config => [
                          {ec2_host, "ec2.us-west-2.amazonaws.com"},
                          {access_key_id, "-"},
                          {secret_access_key, "-"}
                         ]
                        instance_user => "ec2-user",
                    }}]},
```



```
{cloud_plugins, [{static, #{module => mzb_staticcloud_plugin,
                         hosts => ["123.45.67.89", "hostname"]
                         }}]}
```

MZBENCH

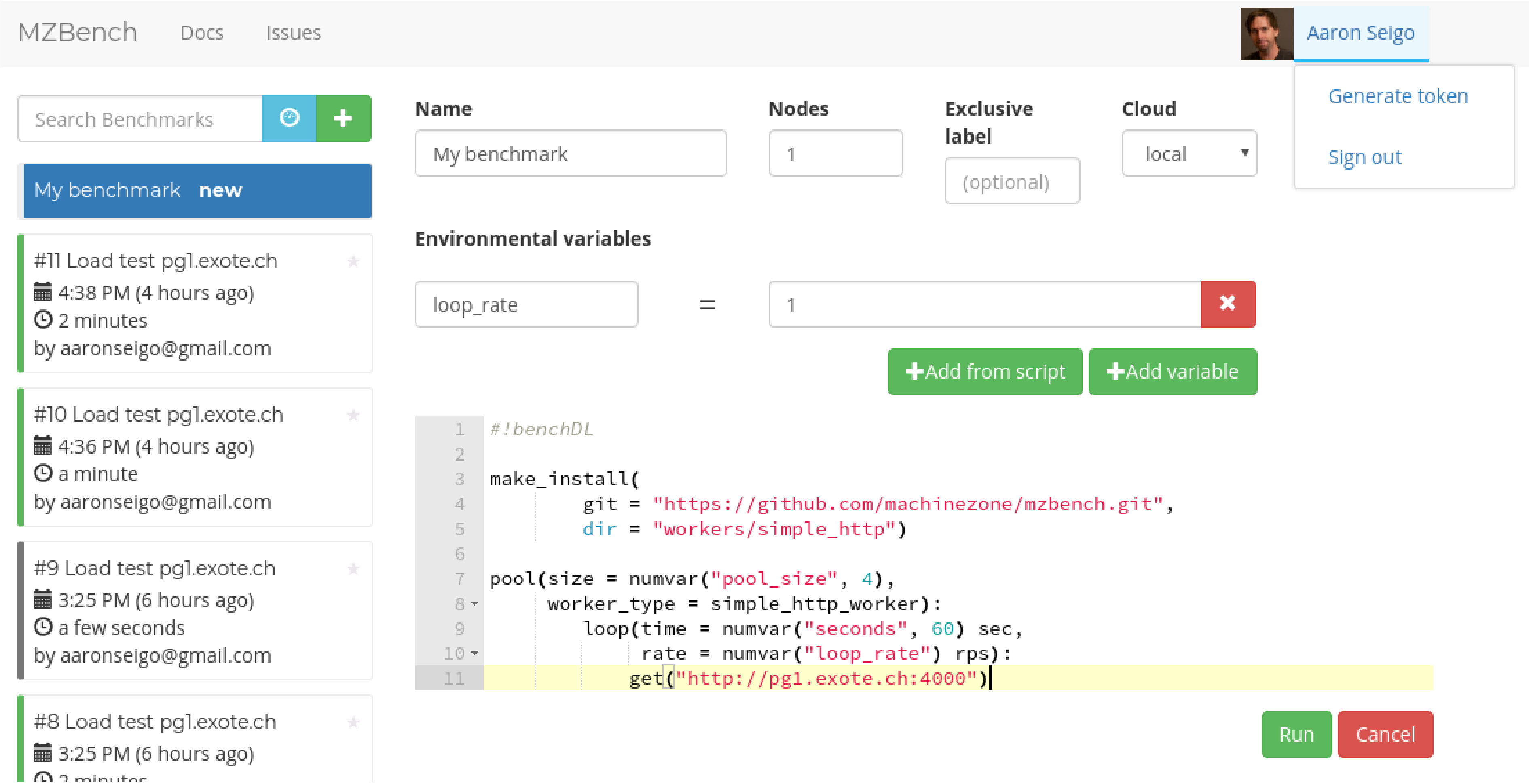
Scenarios are straight forward as well:

```
#!benchDL

make_install(
        git = "https://github.com/machinezone/mzbench.git",
        dir = "workers/simple_http")

pool(size = numvar("pool_size", 4),
     worker_type = simple_http_worker):
        loop(time = numvar("seconds", 60) sec,
             rate = numvar("loop_rate") rps):
            get("http://pg1.exote.ch:4000")
```

MZBENCH



My benchmark **new**

#11 Load test pg1.exote.ch
4:38 PM (4 hours ago)
2 minutes
by aaronseigo@gmail.com

#10 Load test pg1.exote.ch
4:36 PM (4 hours ago)
a minute
by aaronseigo@gmail.com

#9 Load test pg1.exote.ch
3:25 PM (6 hours ago)
a few seconds
by aaronseigo@gmail.com

#8 Load test pg1.exote.ch
3:25 PM (6 hours ago)
2 minutes

Name: My benchmark

Nodes: 1

Exclusive label: (optional)

Cloud: local

Environmental variables

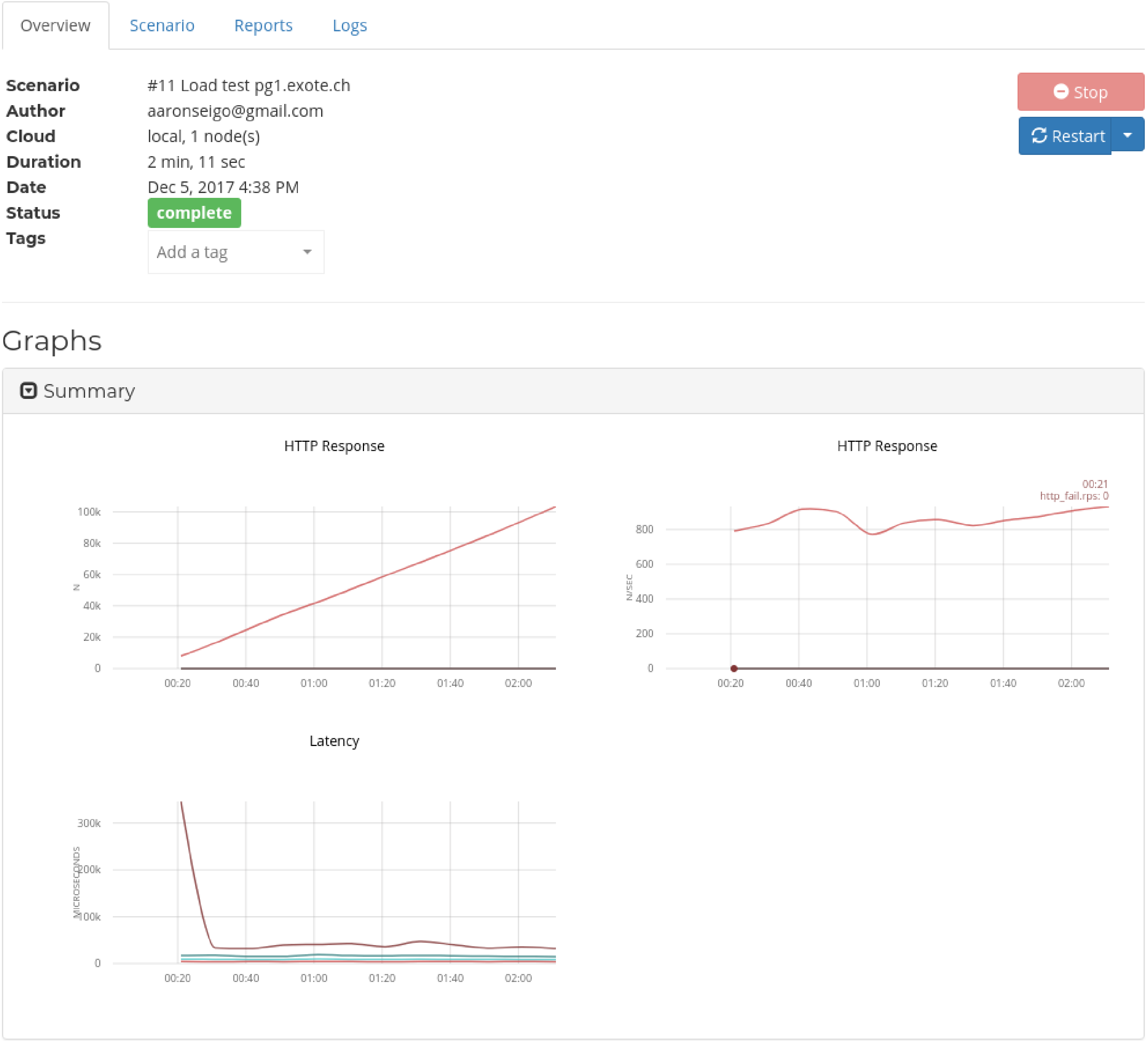
loop_rate = 1

```
#!benchDL

make_install(
        git = "https://github.com/machinezone/mzbench.git",
        dir = "workers/simple_http")

pool(size = numvar("pool_size", 4),
     worker_type = simple_http_worker):
        loop(time = numvar("seconds", 60) sec,
             rate = numvar("loop_rate") rps):
            get("http://pg1.exote.ch:4000")
```

MZBENCH

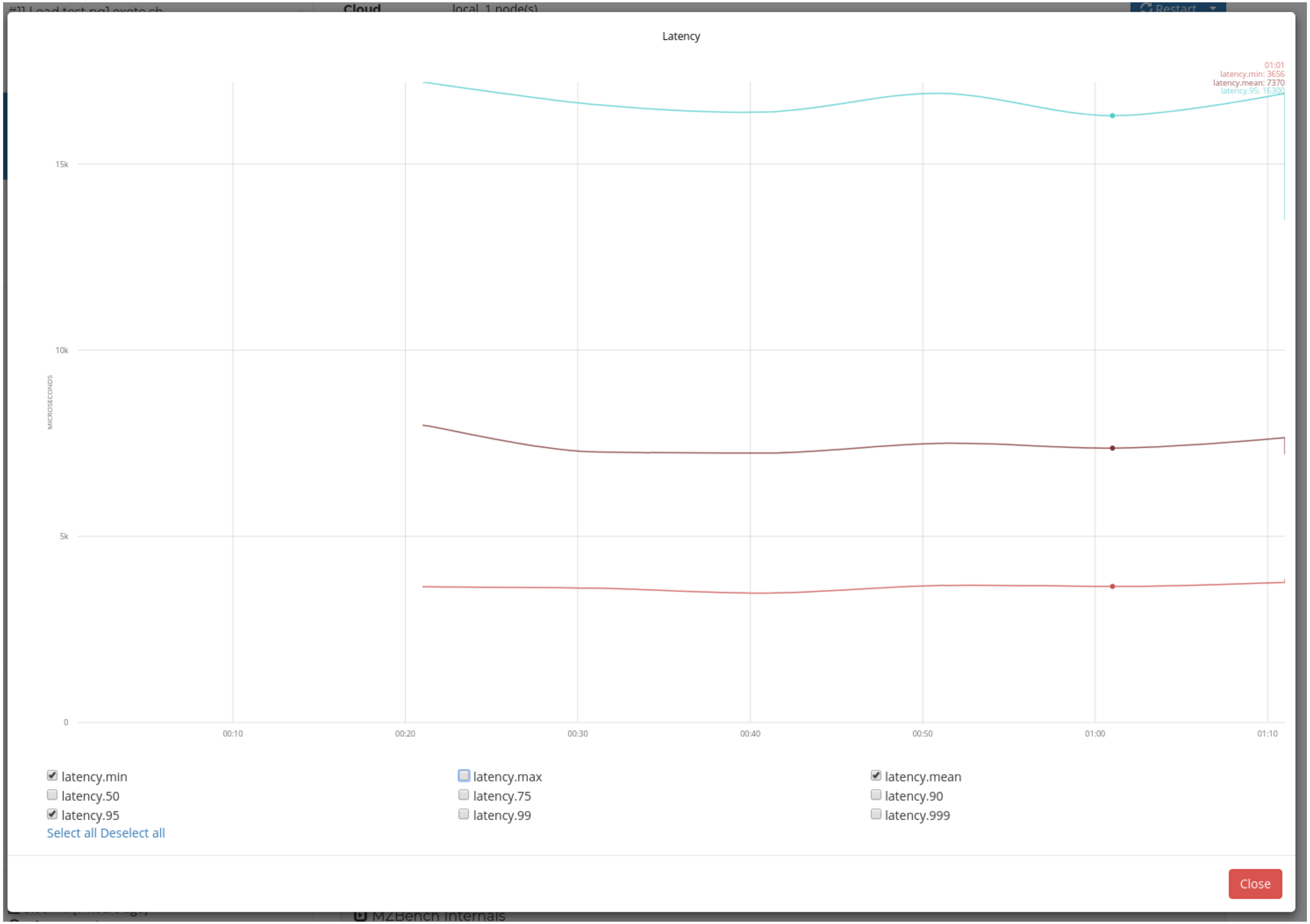


Scenario #11 Load test pg1.exote.ch
Author aaronseigo@gmail.com
Cloud local, 1 node(s)
Duration 2 min, 11 sec
Date Dec 5, 2017 4:38 PM
Status complete
Tags Add a tag

Graphs

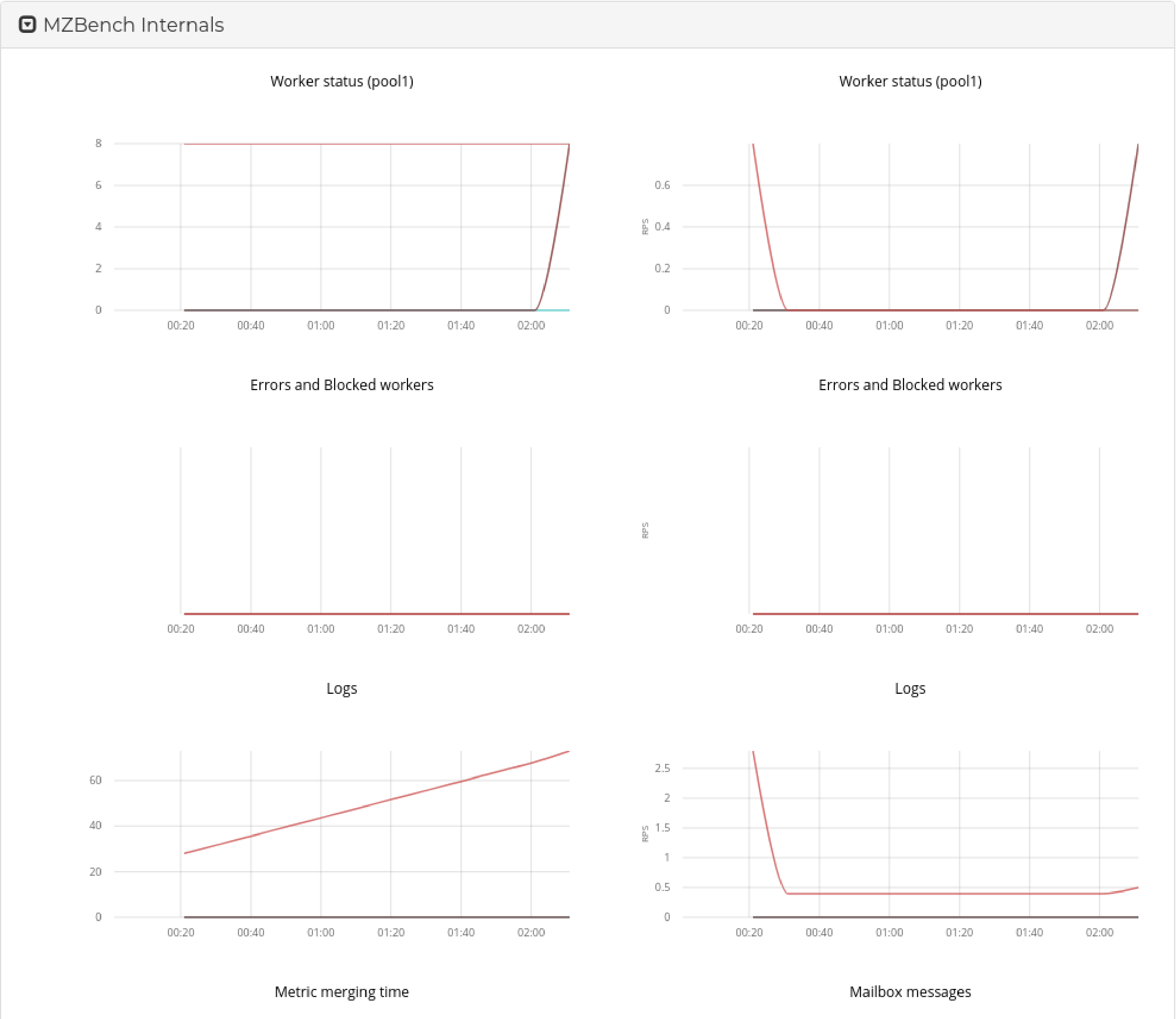
Summary

MZBENCH

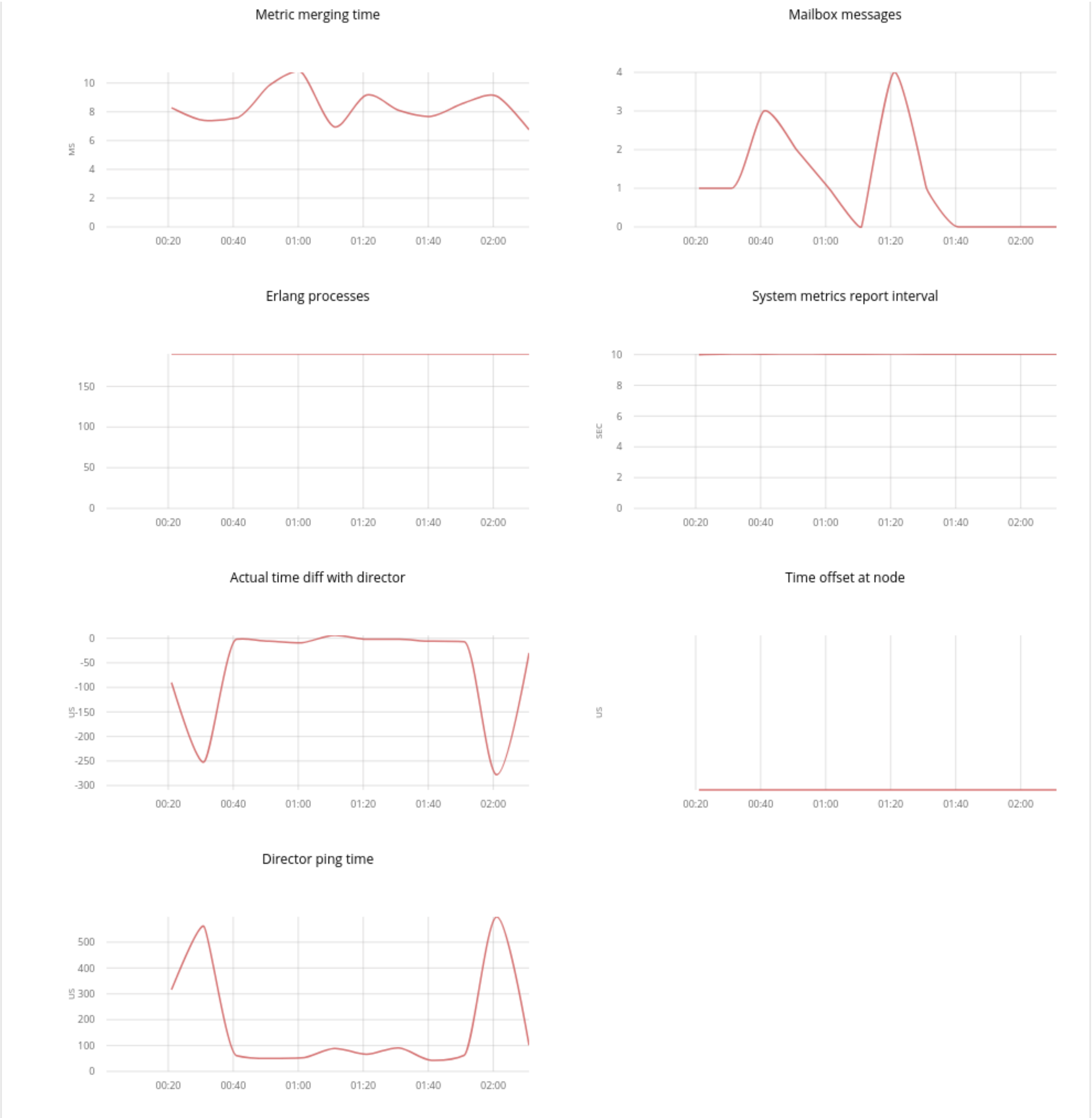


☑ latency.min
☐ latency.max
☑ latency.mean
☐ latency.50
☐ latency.75
☐ latency.90
☑ latency.95
☐ latency.99
☐ latency.999

MZBENCH


MZBench Internals
Worker status (pool1)
8
6
4
2
0
00:20
00:40
01:00
01:20
01:40
02:00
Worker status (pool1)
0.6
0.4
0.2
0
RPS
00:20
00:40
01:00
01:20
01:40
02:00
Errors and Blocked workers
00:20
00:40
01:00
01:20
01:40
02:00
Errors and Blocked workers
RPS
00:20
00:40
01:00
01:20
01:40
02:00
Logs
60
40
20
0
00:20
00:40
01:00
01:20
01:40
02:00
Logs
2.5
2
1.5
1
0.5
0
RPS
00:20
00:40
01:00
01:20
01:40
02:00
Metric merging time
Mailbox messages

MZBENCH



Metric merging time

Mailbox messages

Erlang processes

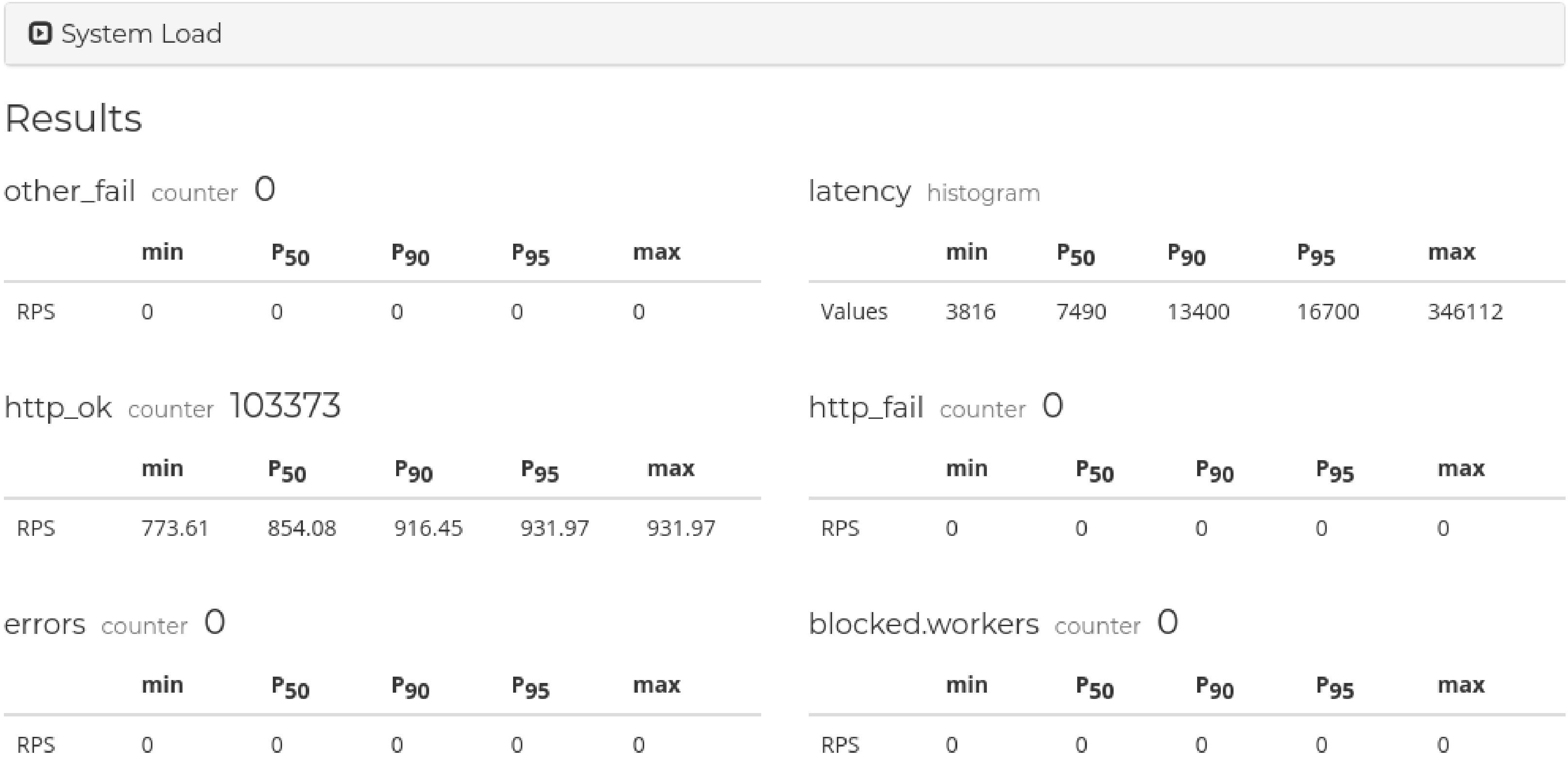
System metrics report interval

Actual time diff with director

Time offset at node

Director ping time

MZBENCH



System Load

Results

other_fail counter 0

	min	P_{50}	P_{90}	P_{95}	max
RPS	0	0	0	0	0

latency histogram

	min	P_{50}	P_{90}	P_{95}	max
Values	3816	7490	13400	16700	346112

http_ok counter 103373

	min	P_{50}	P_{90}	P_{95}	max
RPS	773.61	854.08	916.45	931.97	931.97

http_fail counter 0

	min	P_{50}	P_{90}	P_{95}	max
RPS	0	0	0	0	0

errors counter 0

	min	P_{50}	P_{90}	P_{95}	max
RPS	0	0	0	0	0

blocked.workers counter 0

	min	P_{50}	P_{90}	P_{95}	max
RPS	0	0	0	0	0

MZBENCH

Git

https://github.com/satori-com/mzbench

Documentation

https://github.com/satori-com/mzbench/tree/master/doc

Thank you!

Aaron Seigo – aseigo@mykolab.com - 12/2017