Tutorial

Load Testing Made **Easy**

Diana Corbacho
diana.corbacho@erlang-solutions.com

Erlang SOLUTIONS

Load testing

Responsiveness and stability under a particular workload

Load testing, easy?

Generate massive amounts of traffic is difficult

hardware, software, labour

Load testing, plug and play?

In-house load testing requires knowledge outside of the testing domain

Load testing, suitable tools?

Very specific use cases at huge scale

Combinations of protocols

Integration with the environment

Customisation, customisation, customisation

What can we do?

Challenge existing tools

- SaaS
- Commercial products
- Open source

Build a new tool

Hope for the best, the system won't crash

What did we do?

We built

MEGALOAD

Testers

Developers

CI

Megaload

How did we do it?

www.prowess-project.eu

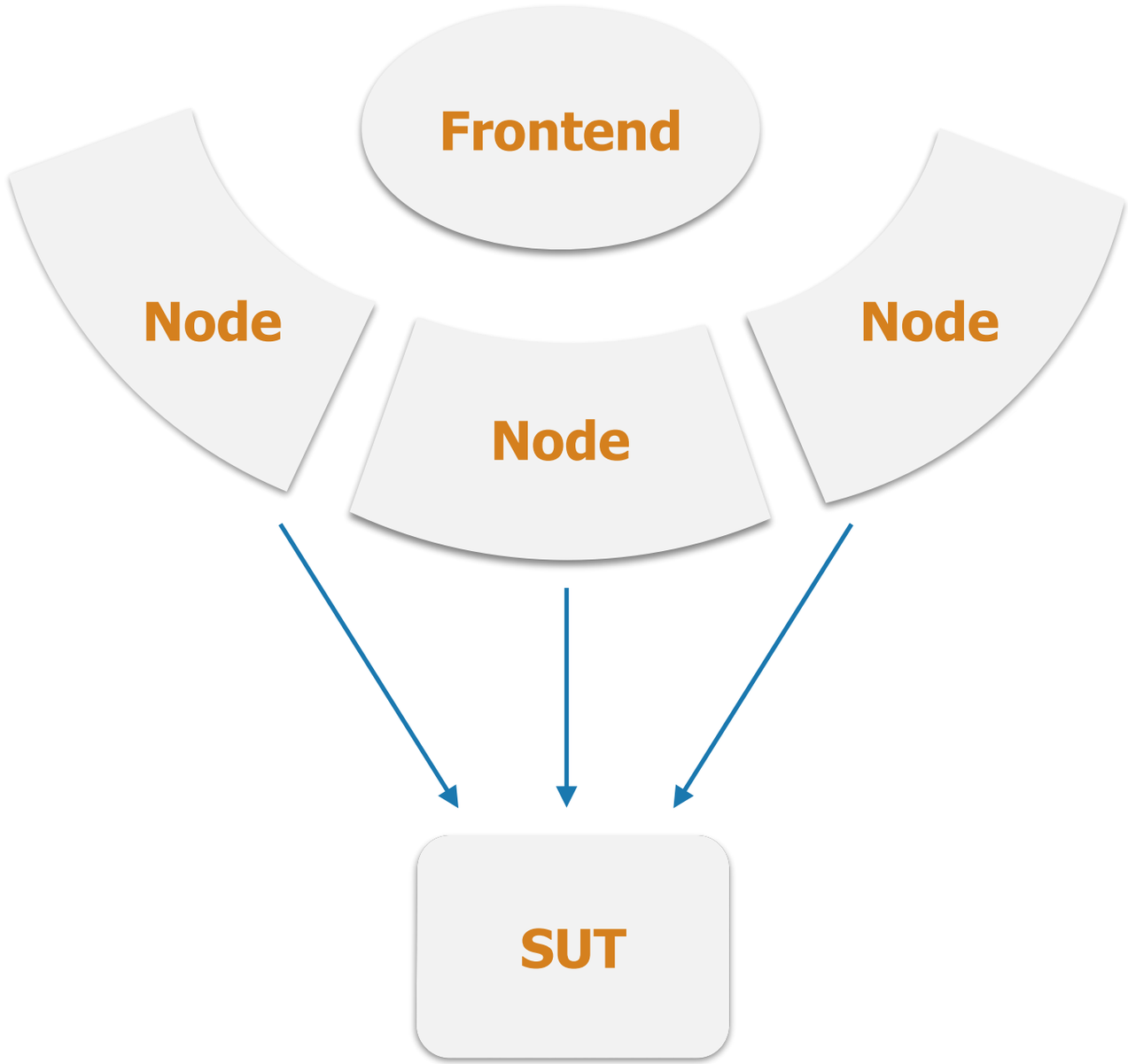
Property-based testing for web services and internet applications

FP7 grant €3.39 million

9 partners from 3 countries

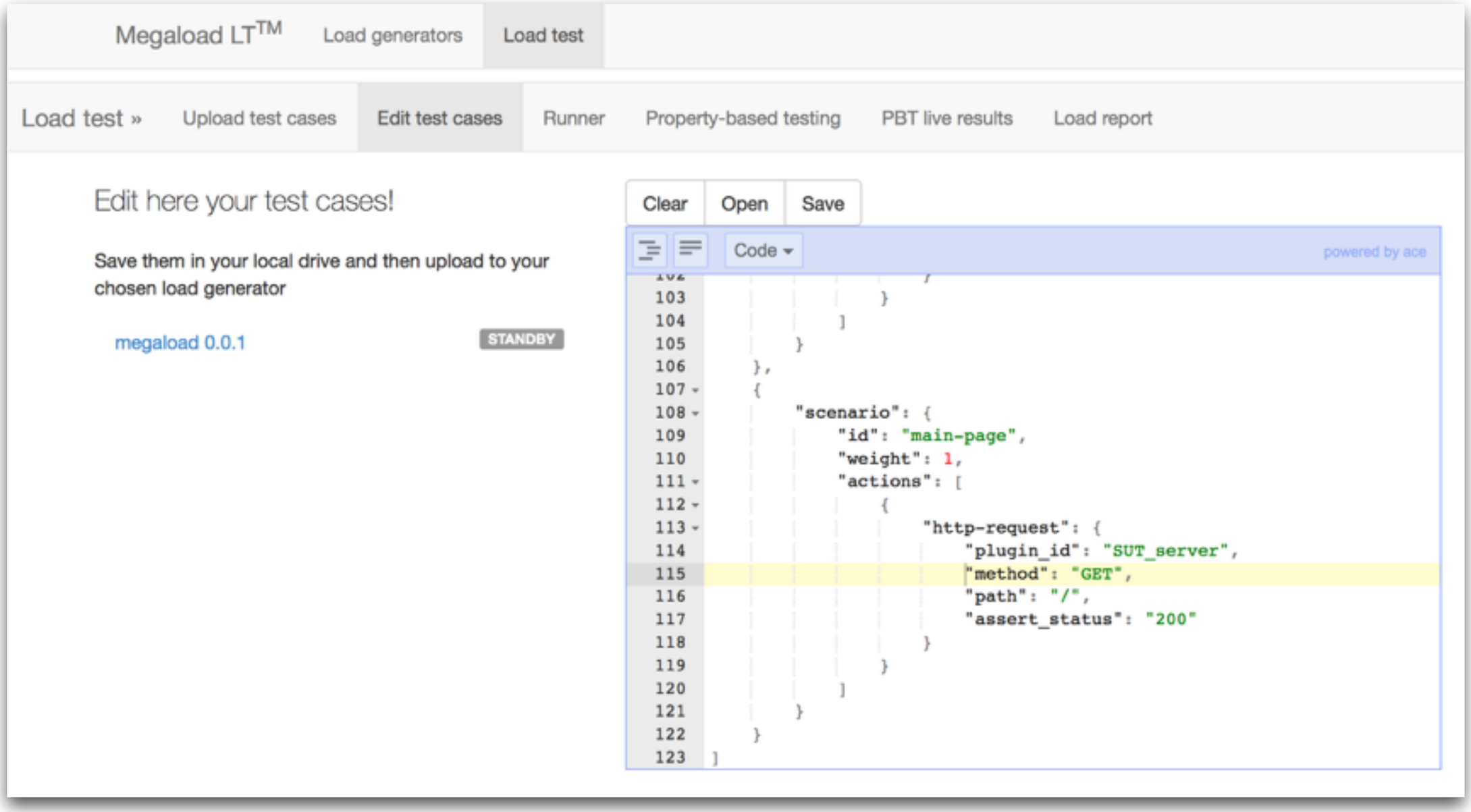
Megaload offers...

Scalable cloud platform to run load tests



Megaload offers...

Web interface



Megaload offers...

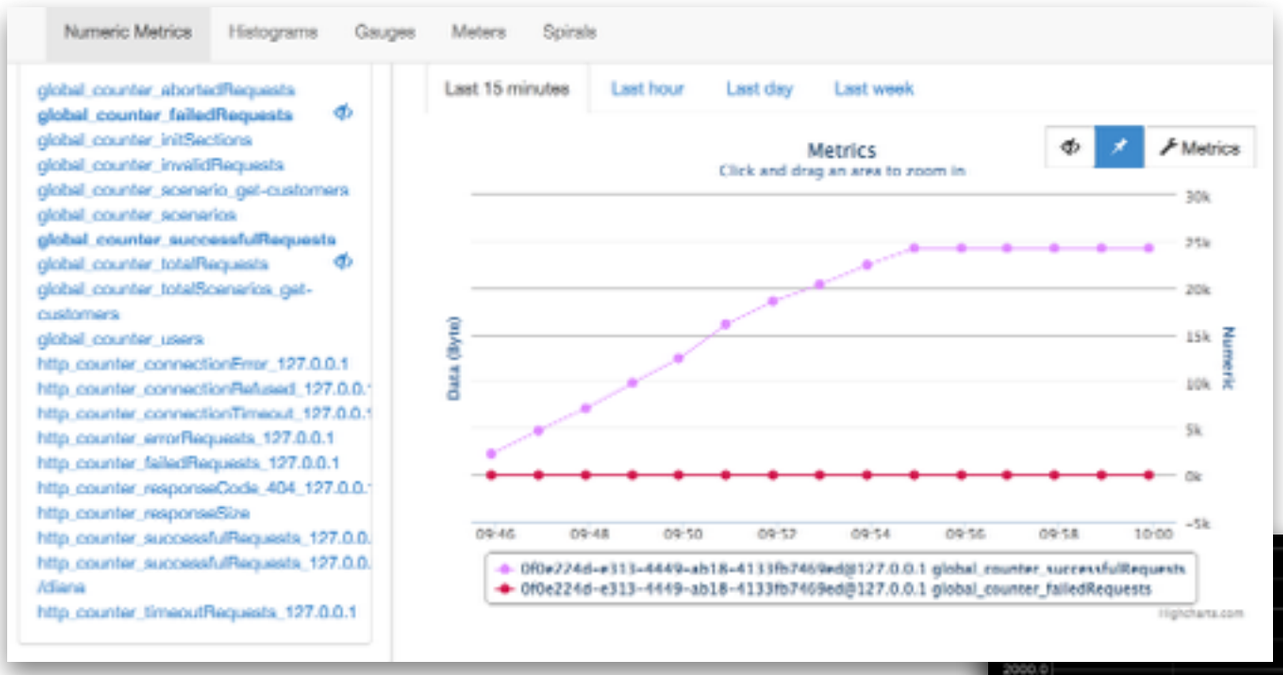
RESTful API

```
$ curl --data '{"id":"SUT-test"}' \
  > http://127.0.0.1:8888/megaload/start_load/
64a052a0-8010-42db-b89e-aae70e869942 \
  > -H 'Content-Type: application/json;charset=utf-8'

< HTTP/1.1 200 OK
< connection: keep-alive
< server: Cowboy
< date: Mon, 19 May 2014 16:25:58 GMT
< content-length: 0
< content-type: application/json
```

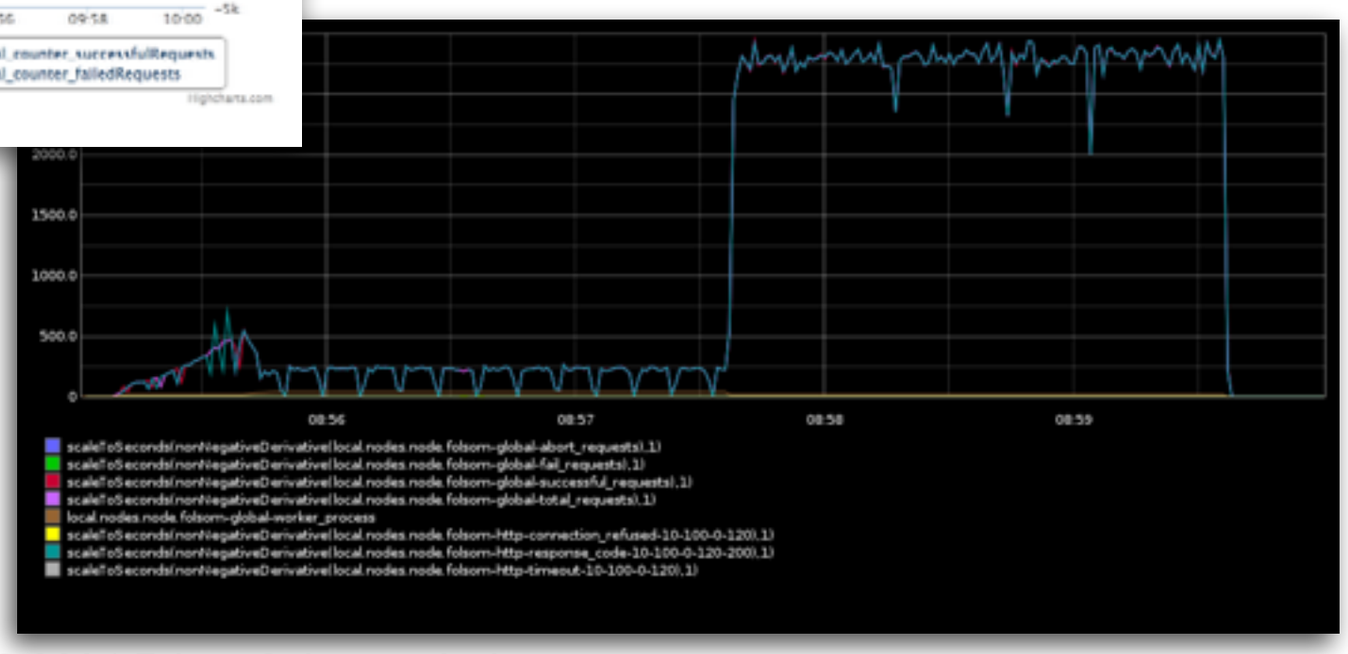
Megaload offers...

Real-time statistics and graphs



web interface

graphite



Megaload offers...

maybe!

DSL to ease test description

```
{
    "scenario": {
        "id": "main-page",
        "weight": 1,
        "actions": [
            {
                "http-request": {
                    "plugin_id": "SUT-server",
                    "method": "GET",
                    "path": "/",
                    "headers": {
                        "Cookie": "PREF=ID=1345ds2345d3",
                        "Connection": "keep-alive"
                    },
                    "assert_status": "200"
                }
            }
        ]
    }
}
```

Megaload offers...

Plugins & Extensions - Highly customisable

XMPP plugin - https://github.com/esl/escalus

```
{
    "xmpp-send": {
        "plugin_id": "userA",
        "packet": {
            "xmpp_stanza-presence": {
                "message": "available"
            }
        }
    }
}
```

```
-module(ml_xmpp).

-compile({parse_transform, exprecs}).
-export_records(['xmpp-send']).

-export([validation_spec/1, send/3]).

validation_spec('xmpp-send') ->
    [
     {plugin_id, mandatory, binary, false},
     {packet, mandatory, binary, true}
    ].

send(#'xmpp-send'{packet = Packet}, GlobalTestId, #transport{} = Transport) ->
    EPacket = loader_run_plugin:eval_action(GlobalTestId, Packet),
    {ok, escalus_connection:send(Transport, EPacket), Transport}.
```

Megaload offers...

Test assertions

Status | Assertions | Assertion details | Response time | Rate | Statistics

Summary

Test case **SUT-test** has Passed

Overview

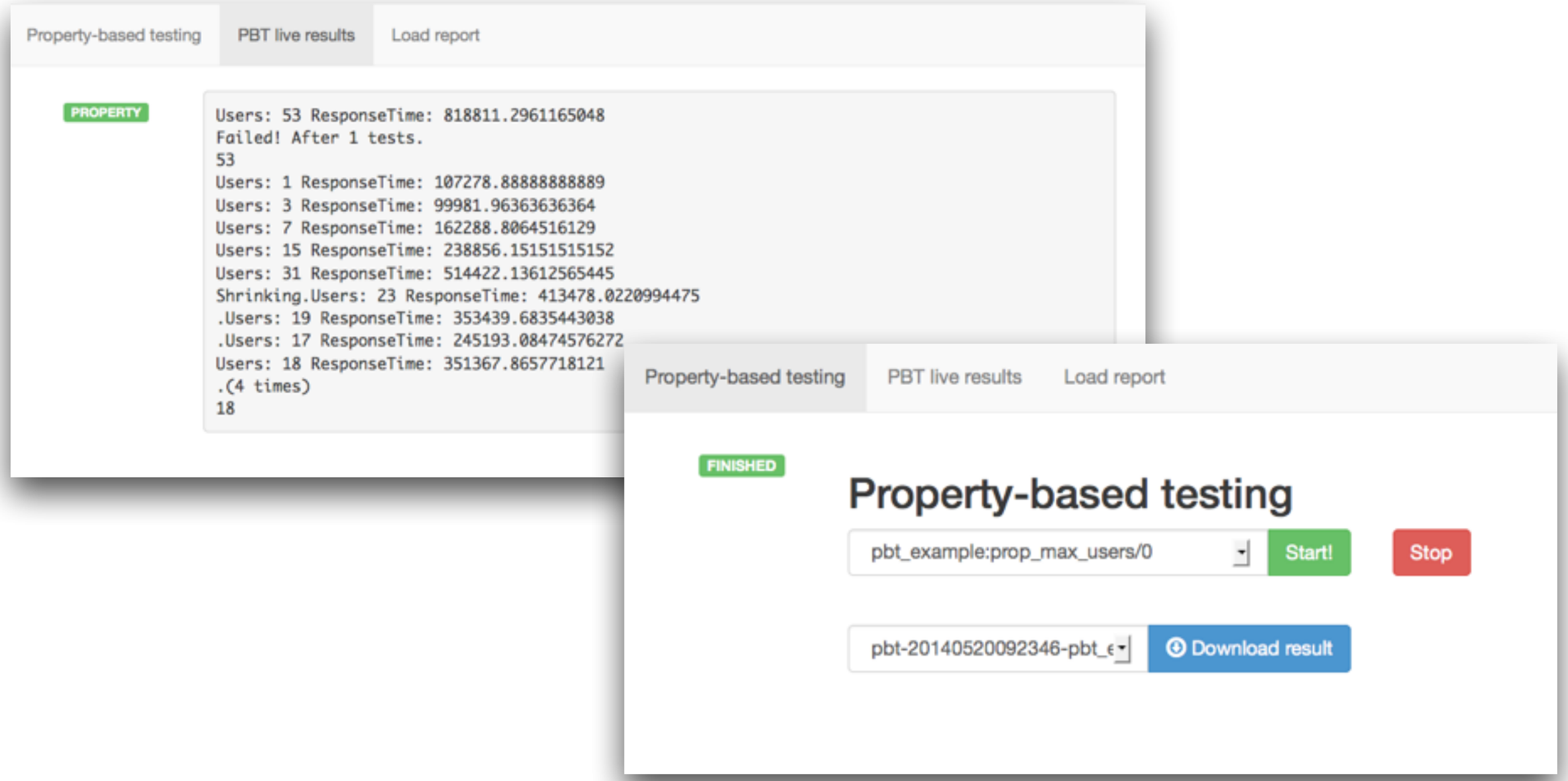
Test Identifier	Status
SUT-test	Passed

Phase Identifier	Status
1 SUT_low	Passed
2 SUT_high	Passed

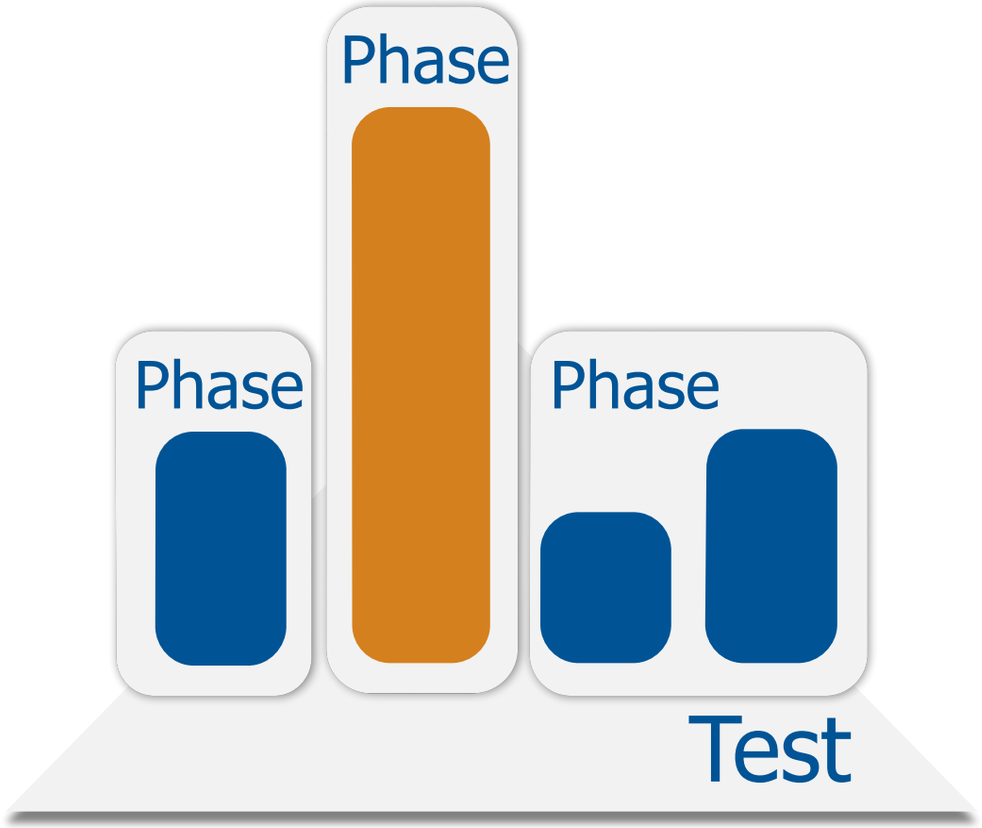
```
"test": {
    "id": "SUT-test",
    "phases": [ "SUT_low", "SUT_high" ],
    "plugins": [ "SUT_server" ],
    "assertions": [
        {
            "assert-histogram": {
                "id": "http_histogram_responseTime",
                "statistic": "mean",
                "filter": {
                    "lt": 500000
                }
            }
        }
    ]
}
}
```

Megaload offers...

Property-based testing applied to load testing



What is a test?



A test is a phase container

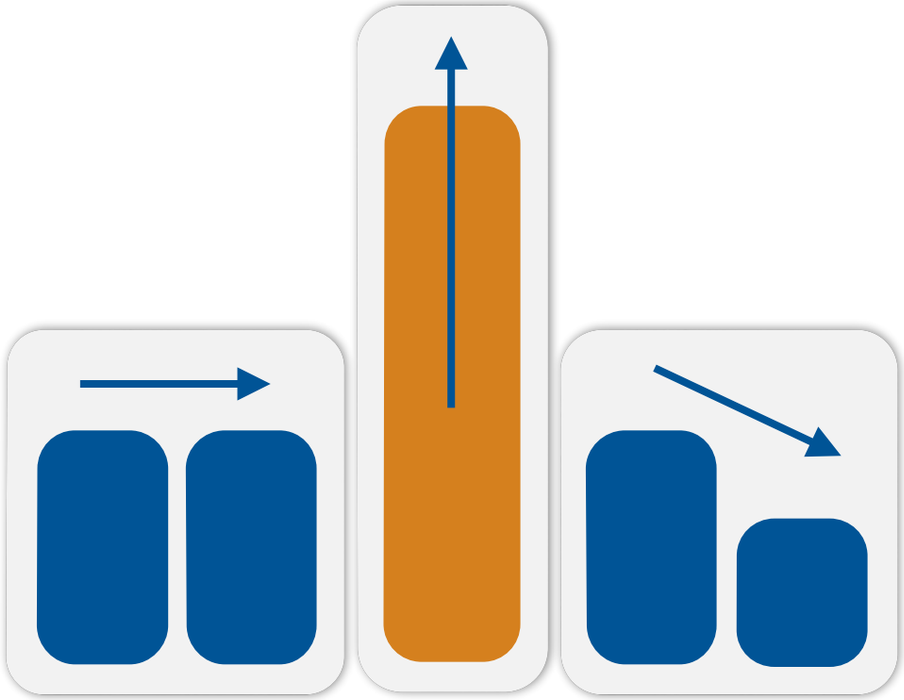
Phase

Load profile

Users

Rate

Scenarios



Scenario

Test actions

Data input

Variable

Custom counter

Think time

JsonPath

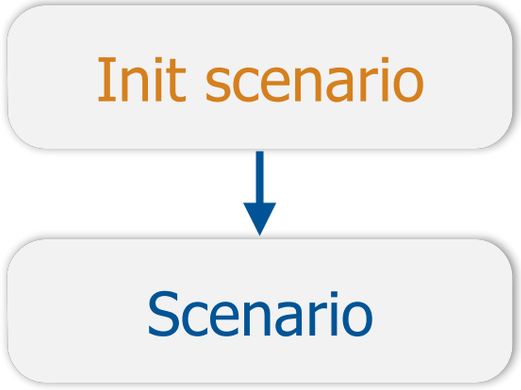
Requests

```
GET /index.html HTTP/1.1
User-Agent: curl/7.30.0
Host: 10.100.0.122:8888
Accept: */*
```

```
<message from="userA@example.net"
to=“userB@example.com">
<body>Hi!</body>
</message>
```

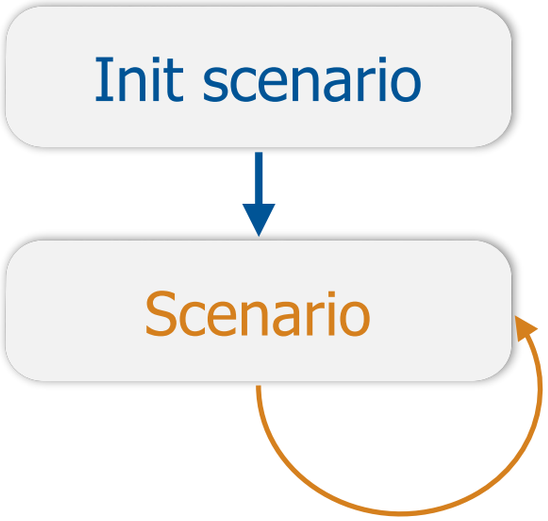
Scenario

Setup scenarios



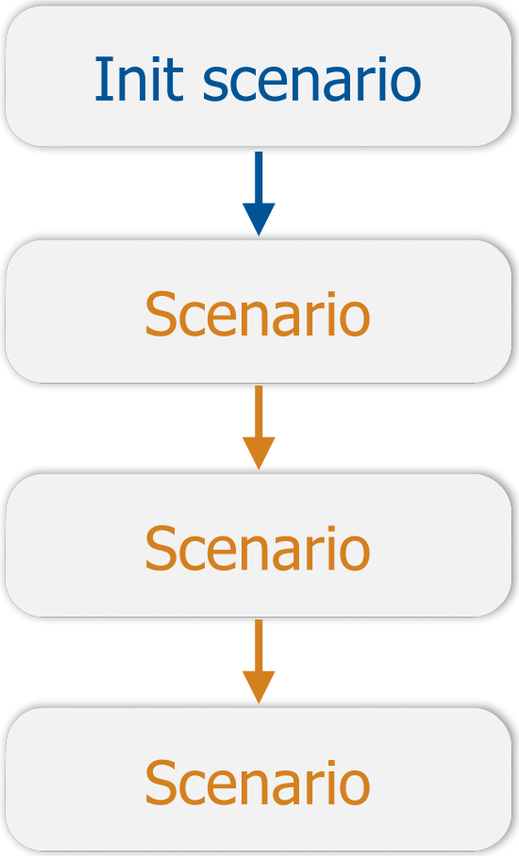
Scenario

Repeat scenarios



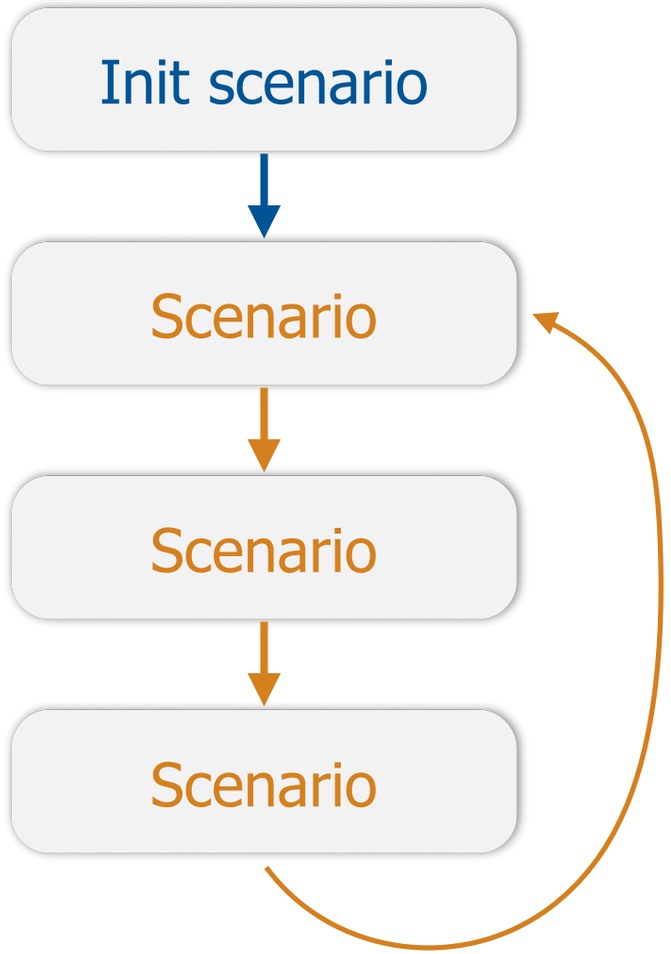
Scenario

Combine scenarios



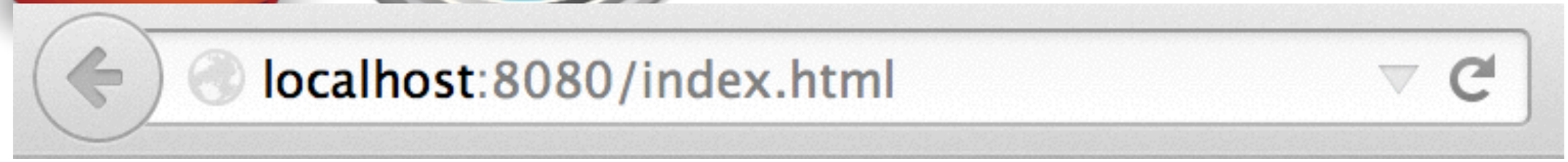
Scenario

Combine + repeat



We know the basic concepts …

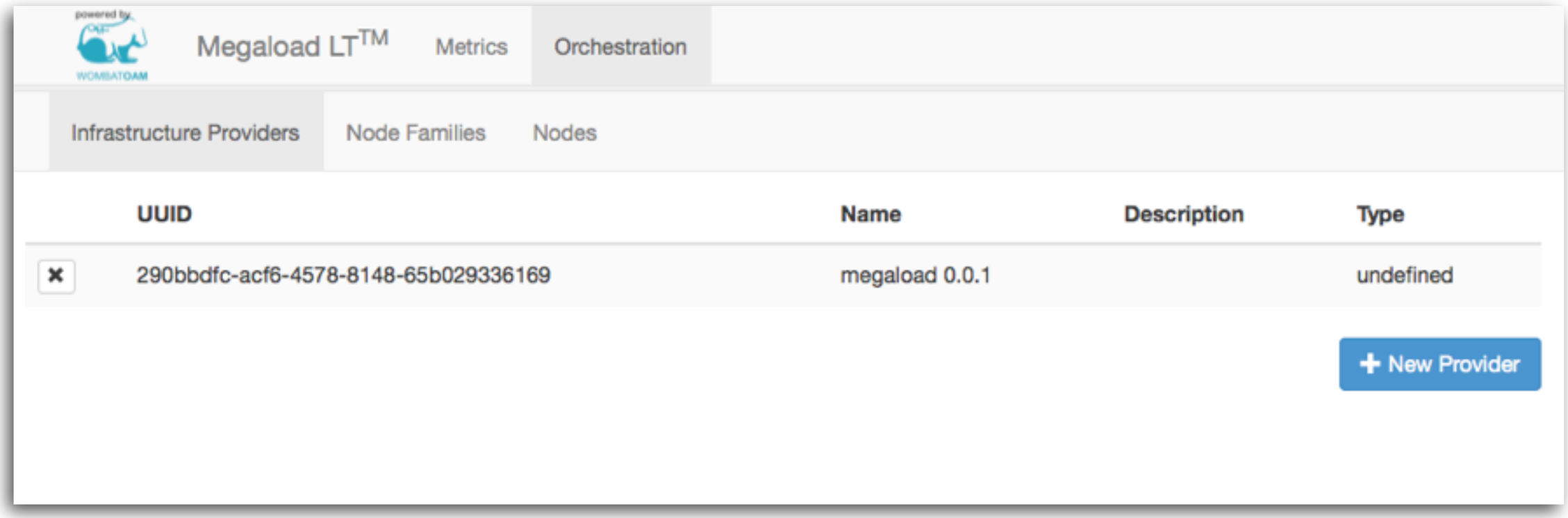
Get started



Not yet supported

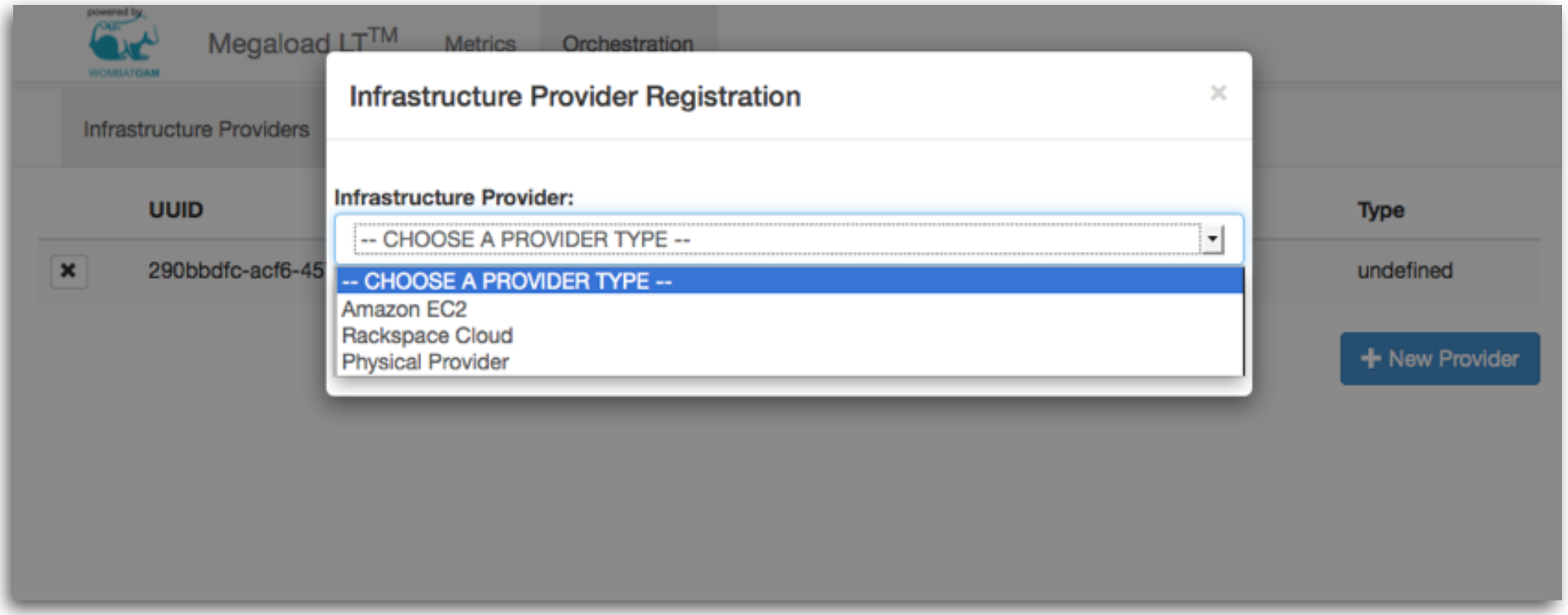
Deployment

Step 1: Add infrastructure provider



Deployment

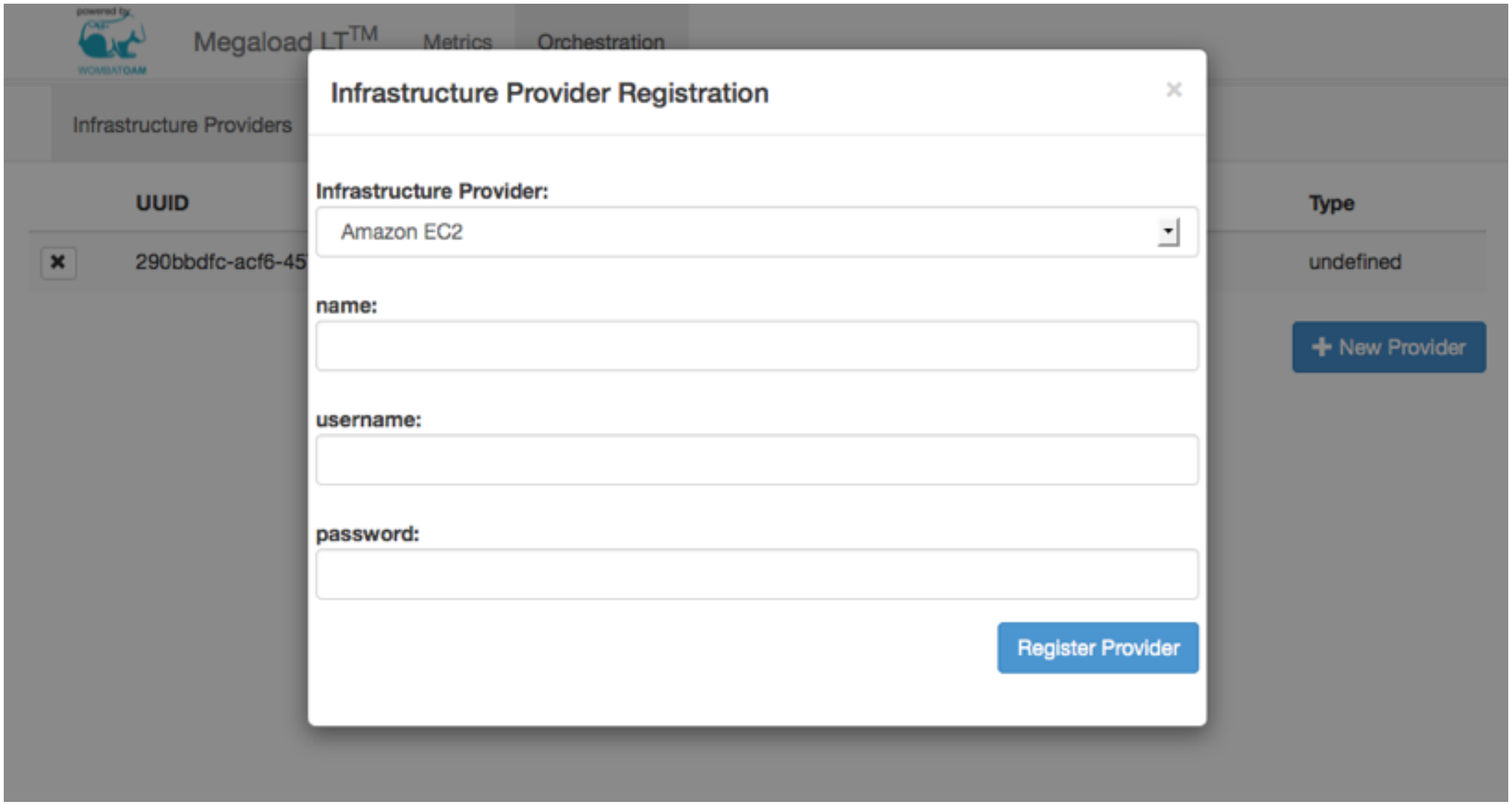
Public & private clouds or physical providers



Uses libcloud

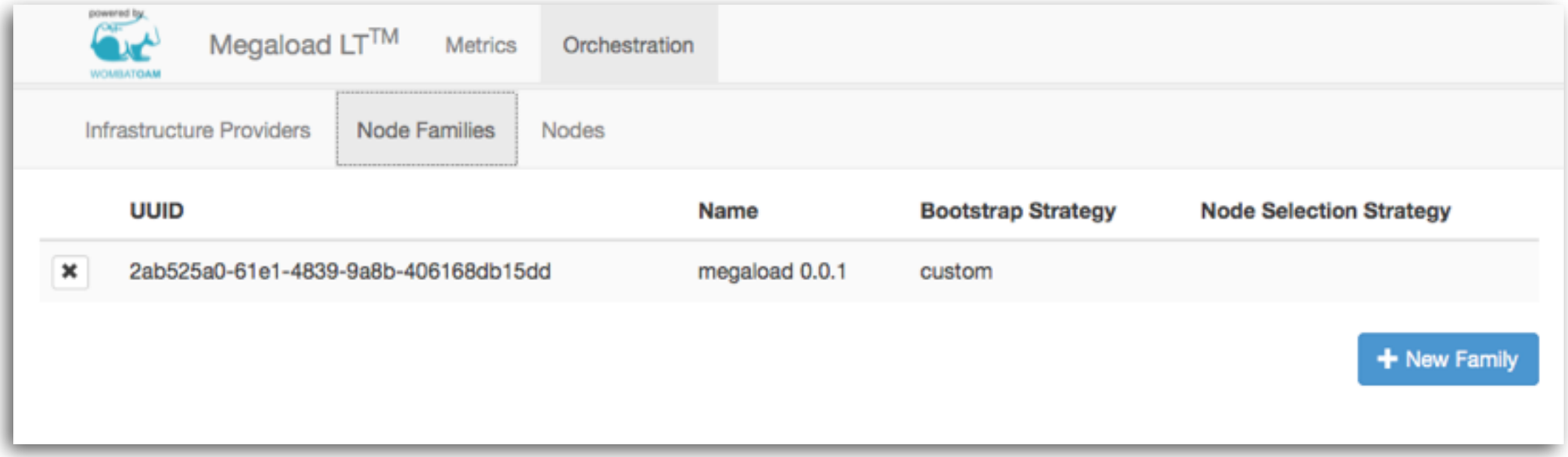
Deployment

Amazon EC2



Deployment

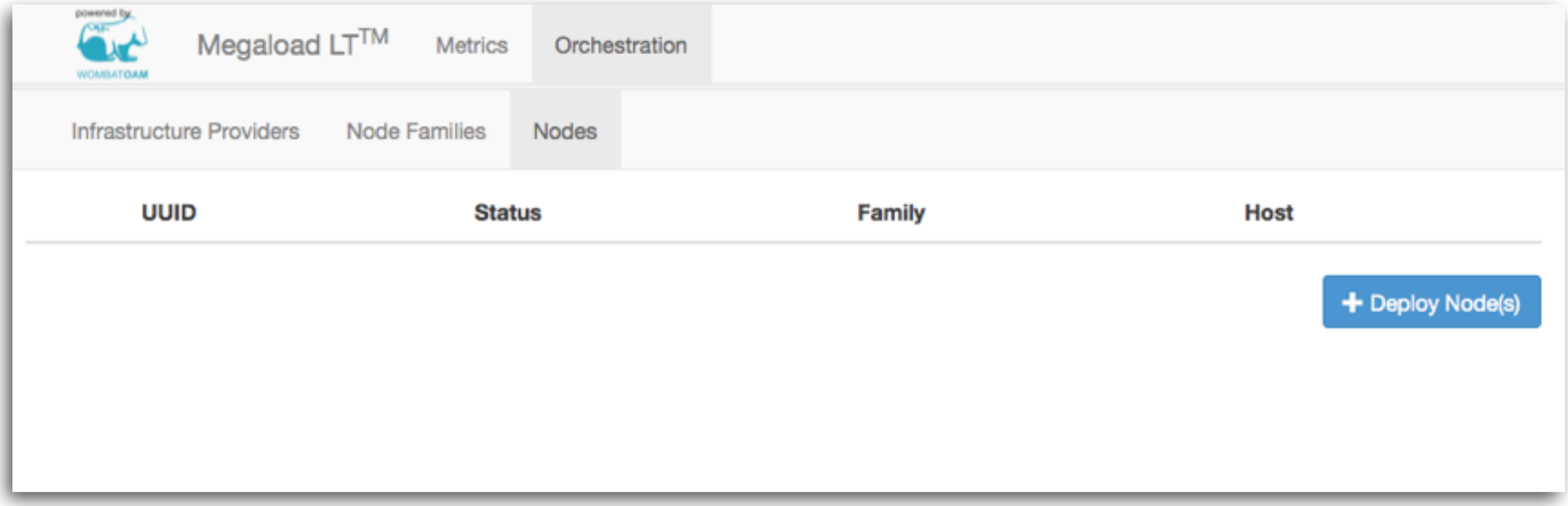
Step 2: Add node family



A node family is a cluster of nodes

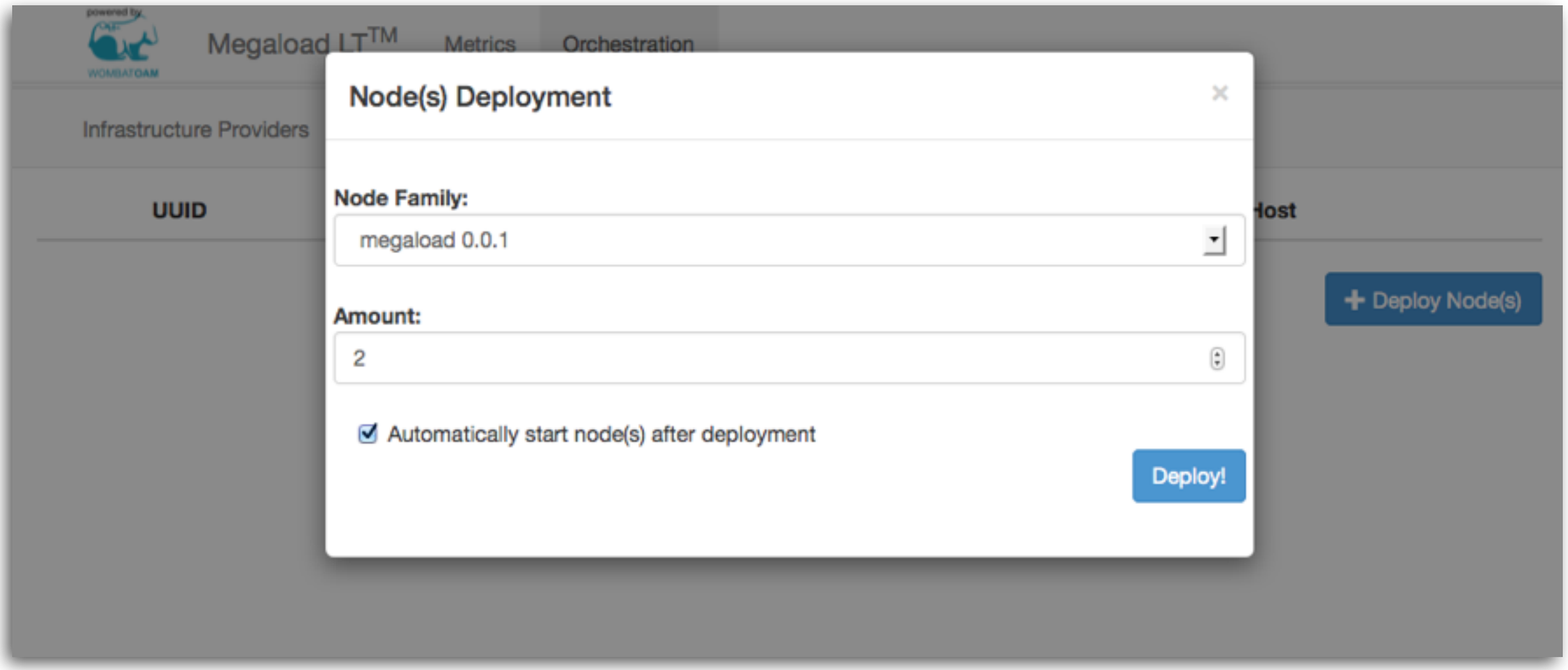
Deployment

Step 3: Deploy nodes



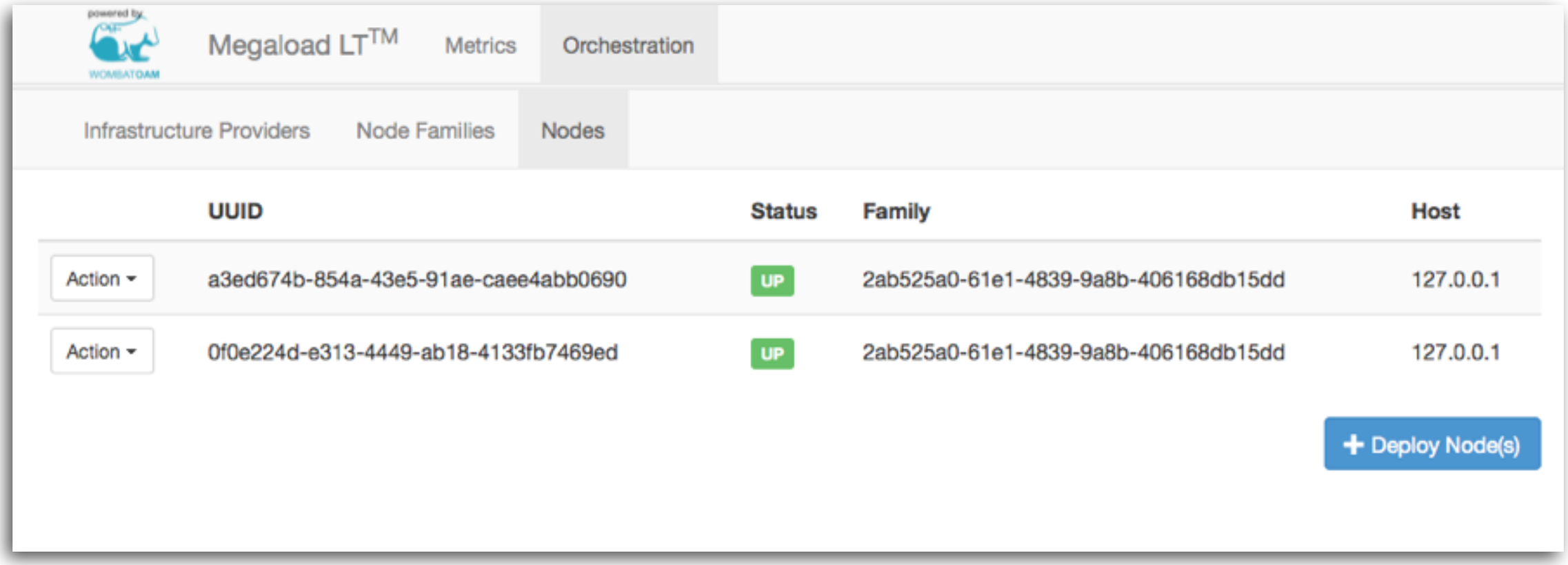
Deployment

Deploy and start 2 nodes



Deployment

Megaload is up!



What can we test?

Web service - small online shop

List costumers

Get costumer

HTTP + JSON

Online shop

GET http://<IP>:5050/customers

```
{"data" : [
            {"id" : "goku",
             "email" : "goku@dragon.ball",
             "account balance" : 500,
              "currency" : "GBP"
            },
            {…},
            {…}
          ],
 "count" : 4
}
```

Online shop

GET http://<IP>:5050/customer/goku

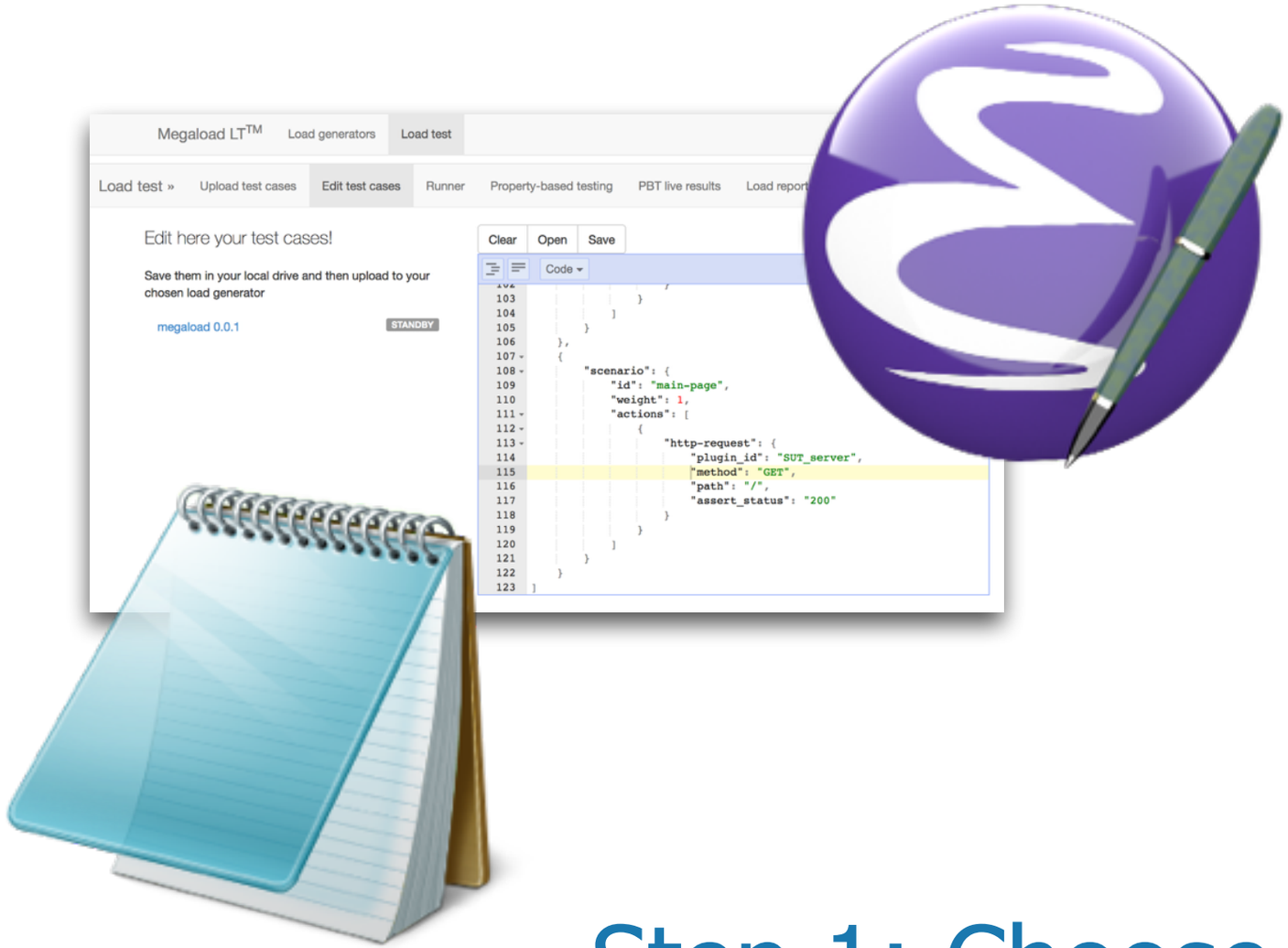
GET http://<IP>:5050/customer/bulma

```
{"id" : "bulma",
   "email" : "bulma@dragon.ball",
   "account_balance" : 750000,
   "currency" : "GBP"
 }
```

GET http://<IP>:5050/customer/vegeta

GET http://<IP>:5050/customer/piccolo

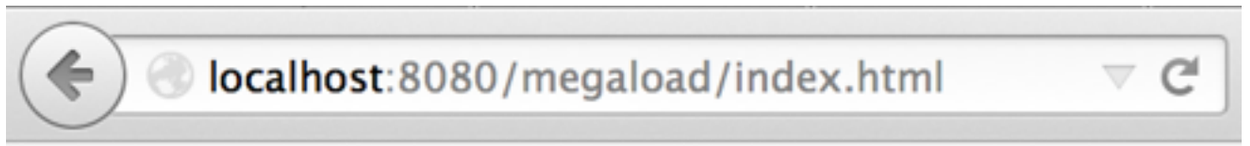
Let’s write our test!

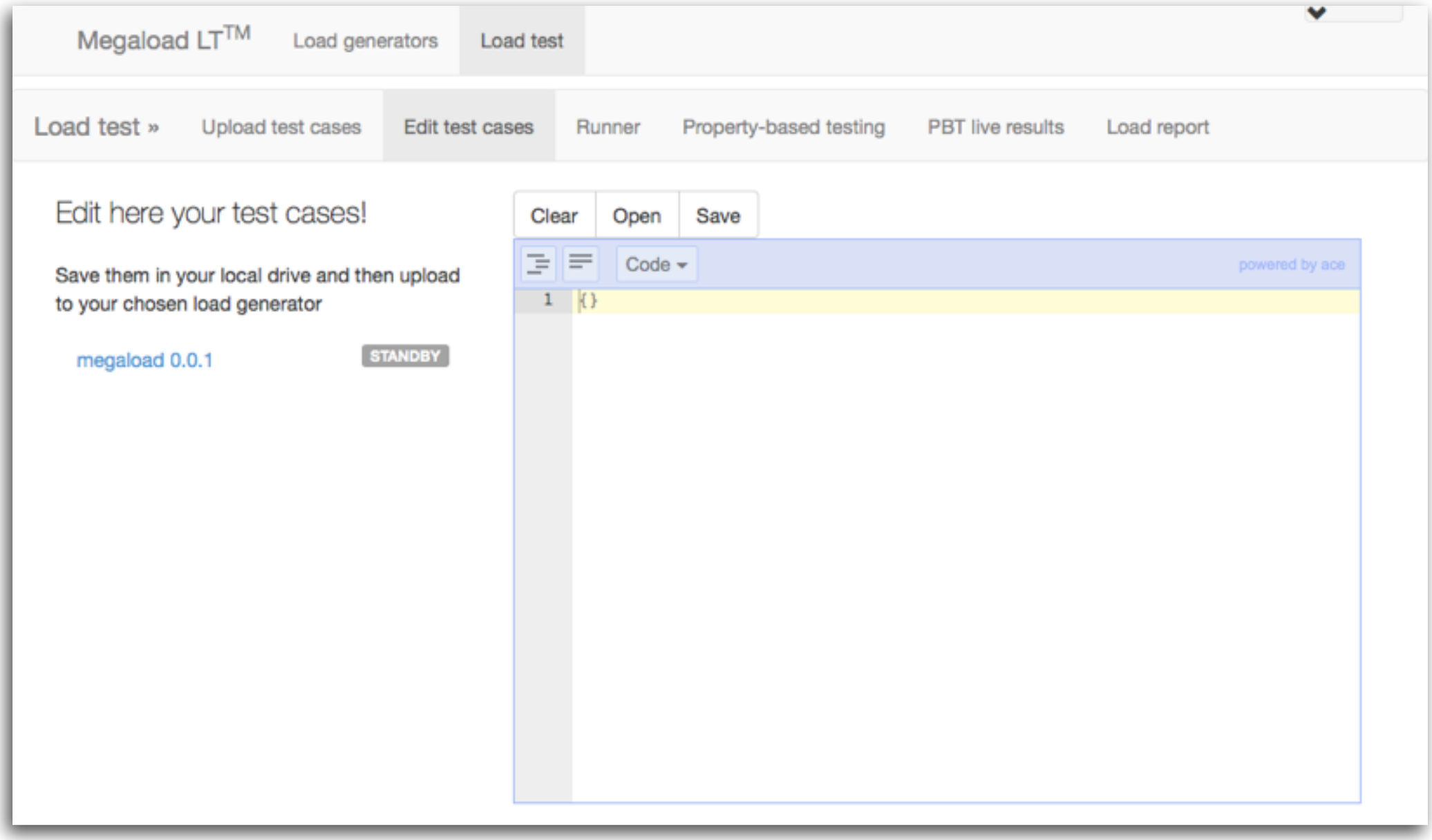


Step 1: Choose your text editor

Step 1: Choose your text editor

Embedded editor at





Step 2: Define a plugin

Plugin is the target server SUT

```
{
 "plugin" :
     {
      "id" : <ID>,
      "plugin_info" : <SPECIFIC_PLUGIN_INFO>
     }
}
```

Step 2: Define a plugin

Give it a name

```
{
 "plugin" :
     {
      "id" : "SUT-server",
      "plugin_info" : <SPECIFIC_PLUGIN_INFO>
     }
}
```

Step 2: Define a plugin

Write the plugin details "plugin_info"

```
{"http-plugin" :
        {"servers" :
            [
               {"host" : "127.0.0.1",
                "port" : 80,
                "ssl" : false
               }
            ],
         "stats_per_url" : true
        }
}
```

Step 2: Define a plugin

```
{
 "plugin" :
     {
      "id" : "SUT-server",
      "plugin_info" : {"http-plugin" :
                                  {"servers" :
                                       [
                                    { "host" : "127.0.0.1",
                                      "port" : 80,
                                      "ssl" : false }
                                       ],
                                   "stats_per_url" : true
                                  }
                              }
     }
}
```

Your plugin is ready!

Step 3: Define a scenario

Easy first, scenario with one request

```
{
 "scenario" :
     {
      "id" : <ID>,
      "weight" : <WEIGHT>,
      "keepalive" : <REUSE_CONNECTION>,
       "actions" : <ACTIONS>
     }
}
```

Step 3: Define a scenario

Give it a name

```
{
 "scenario" :
     {
      "id" : "get-customers",
      "weight" : <WEIGHT>,
      "keepalive" : <REUSE_CONNECTION>,
       "actions" : <ACTIONS>
     }
}
```

Step 3: Define a scenario

Give it a weight

```
{
 "scenario" :
     {
      "id" : "get-customers",
      "weight" : 1,
      "keepalive" : <REUSE_CONNECTION>,
       "actions" : <ACTIONS>
     }
}
```

Step 3: Define a scenario

Set to repeat scenario & reuse open connections

```
{
 "scenario" :
     {
      "id" : "get-customers",
      "weight" : 1,
      "keepalive" : true,
       "actions" : <ACTIONS>
     }
}
```

Step 3: Define a scenario

Add an action - HTTP request

```
{
 "http-request" :
     {
      "plugin_id" : <PLUGIN_ID>,
      "method" : <HTTP_REQUEST_METHOD>,
       "path" : <HTTP_REQUEST_URI>
     }
}
```

Step 3: Define a scenario

Request the main page

```
{
 "http-request" :
     {
      "plugin_id" : "SUT-server",
      "method" : "GET",
       "path" : "/customers"
     }
}
```

Step 3: Define a scenario

Your scenario is ready!

```
{
 "scenario" :
    {
     "id" : "get-customers",
     "weight" : 1,
     "keepalive" : true,
     "actions" : [{"http-request" :
                    {
                     "plugin_id" : "SUT-server",
                     "method" : "GET",
                     "path" : "/customers"
                    }
                  }]
    }
}
```

Step 4: Define a phase

Load profile

```
{
 "phase" :
     {
      "id" : <ID>,
      "interval" : <RAMP_UP_INTERVAL>,
      "duration" : <TEST_DURATION>,
      "max_proc" : <MAX_CONCURRENT_SCENARIOS>,
      "max_rate" : <MAX_RATE>,
      "scenarios" : <SCENARIOS>
     }
}
```

Step 4: Define a phase

Give it a name

```
{
 "phase" :
     {
      "id" : "SUT-low",
      "interval" : <RAMP_UP_INTERVAL>,
      "duration" : <TEST_DURATION>,
      "max_proc" : <MAX_CONCURRENT_SCENARIOS>,
      "max_rate" : <MAX_RATE>,
      "scenarios" : <SCENARIOS>
     }
}
```

Step 4: Define a phase

Give it a ramp up interval in milliseconds

```
{
 "phase" :
     {
      "id" : "SUT-low",
      "interval" : 100,
      "duration" : <TEST_DURATION>,
      "max_proc" : <MAX_CONCURRENT_SCENARIOS>,
      "max_rate" : <MAX_RATE>,
      "scenarios" : <SCENARIOS>
     }
}
```

Step 4: Define a phase

Give it a duration in milliseconds

```
{
 "phase" :
     {
      "id" : "SUT-low",
      "interval" : 100,
      "duration" : 60000,
      "max_proc" : <MAX_CONCURRENT_SCENARIOS>,
      "max_rate" : <MAX_RATE>,
      "scenarios" : <SCENARIOS>
     }
}
```

Step 4: Define a phase

Give it a maximum of concurrent scenarios

```
{
 "phase" :
     {
      "id" : "SUT-low",
      "interval" : 100,
      "duration" : 60000,
      "max_proc" : 100,
      "max_rate" : <MAX_RATE>,
      "scenarios" : <SCENARIOS>
     }
}
```

Step 4: Define a phase

Give it a maximum rate - requests per second

```
{
 "phase" :
     {
      "id" : "SUT-low",
      "interval" : 100,
      "duration" : 60000,
      "max_proc" : 100,
      "max_rate" : 250,
      "scenarios" : <SCENARIOS>
     }
}
```

Step 4: Define a phase

Set it to run the scenario from step 3

Your phase is ready!

```
{
 "phase" :
     {
      "id" : "SUT-low",
      "interval" : 100,
      "duration" : 60000,
      "max_proc" : 100,
      "max_rate" : 250,
      "scenarios" : ["get-customers"]
     }
}
```

Step 5: Define a test

Test is a container of phases

```
{
 "test" :
     {
      "id" : <ID>,
      "phases" : <PHASES>,
      "plugins" : <PLUGINS>
     }
}
```

Step 5: Define a test

Give it a name

```
{
 "test" :
     {
      "id" : "SUT-test",
      "phases" : <PHASES>,
      "plugins" : <PLUGINS>
     }
}
```

Step 5: Define a test

Set it to run the phase from step 4

```
{
 "test" :
     {
      "id" : "SUT-test",
      "phases" : ["SUT-low"],
      "plugins" : <PLUGINS>
     }
}
```

Step 5: Define a test

Set it to use the plugin from step 2

Your test is ready!

```
{
 "test" :
     {
      "id" : "SUT-test",
      "phases" : ["SUT-low"],
      "plugins" : ["SUT-server"]
     }
}
```

Step 6: Compose a valid JSON

Megaload input is a list of JSON objects

```
[
  {"test" : … },
  {"plugin" : … },
  {"phase" : … },
  {"scenario" : … }
]
```

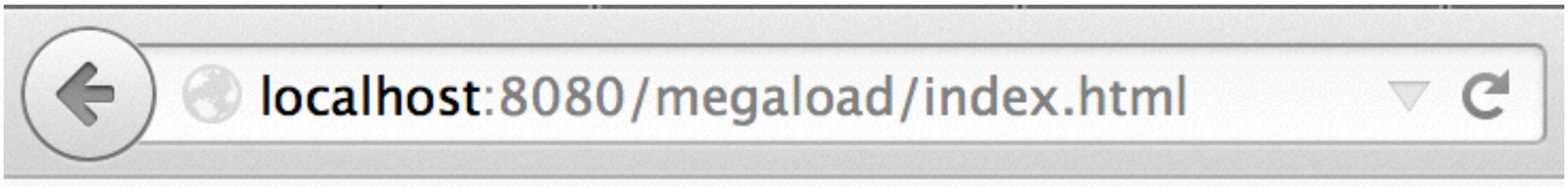
Step 7: Save your test

You will need to upload the text file

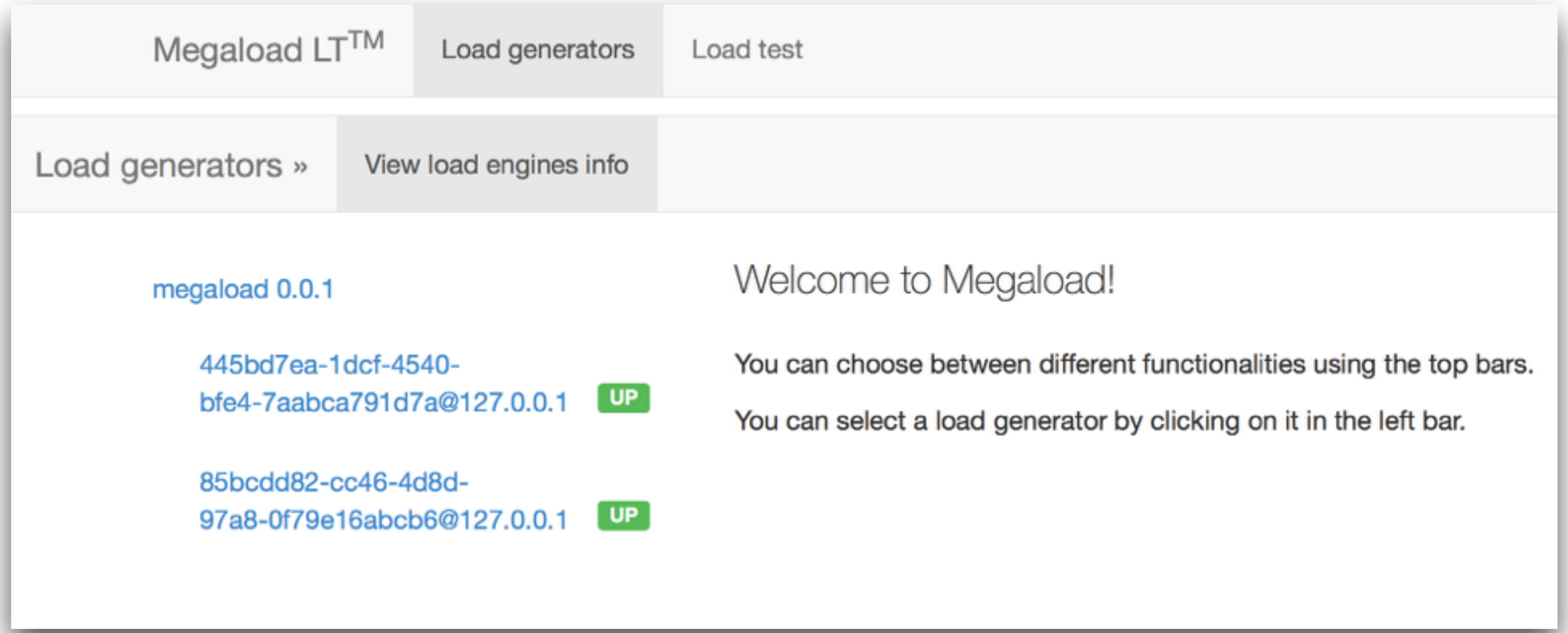
`SUT-test-example.json`

Our test case is ready, let's run it!

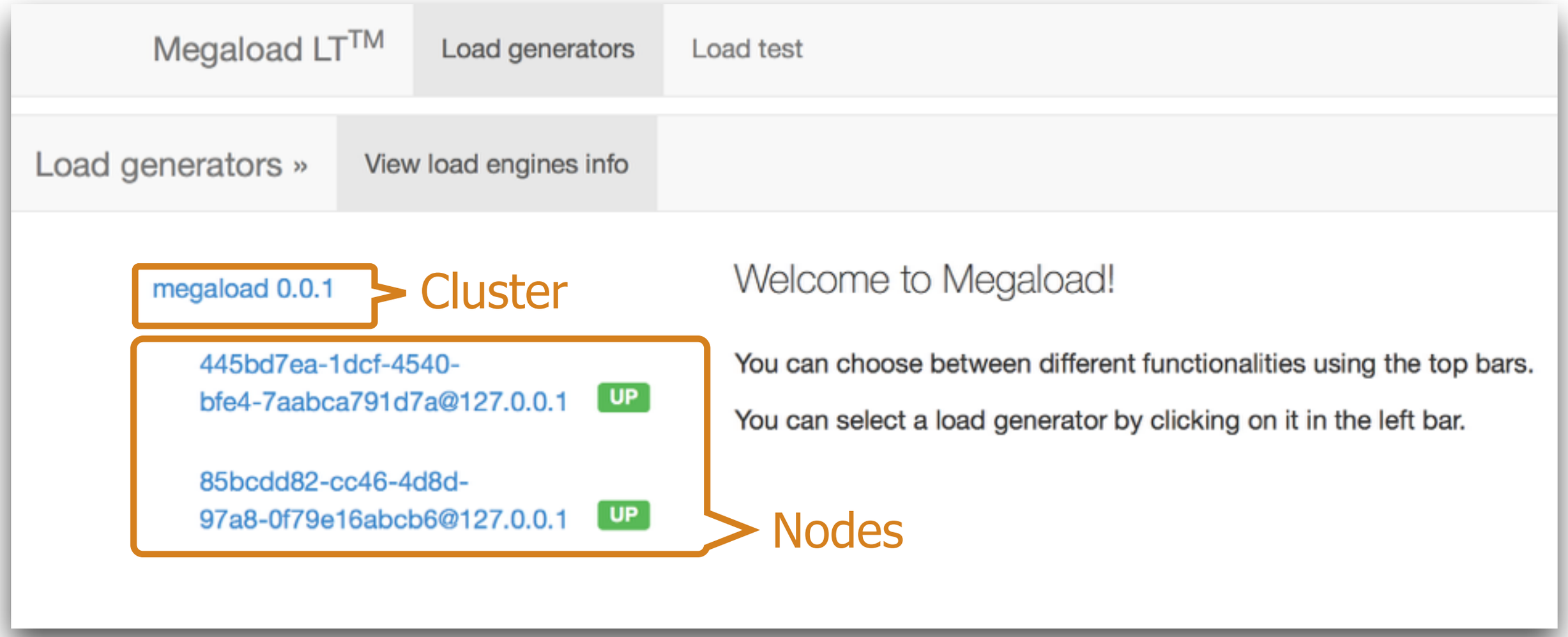
Go to Megaload dashboard



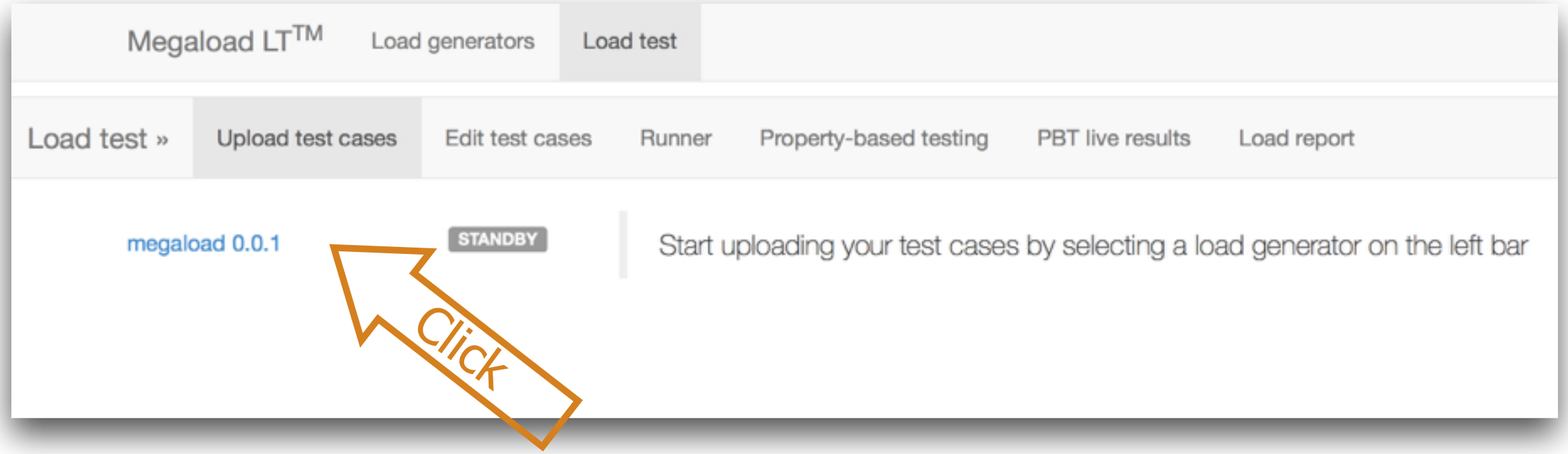
Main page



Main page

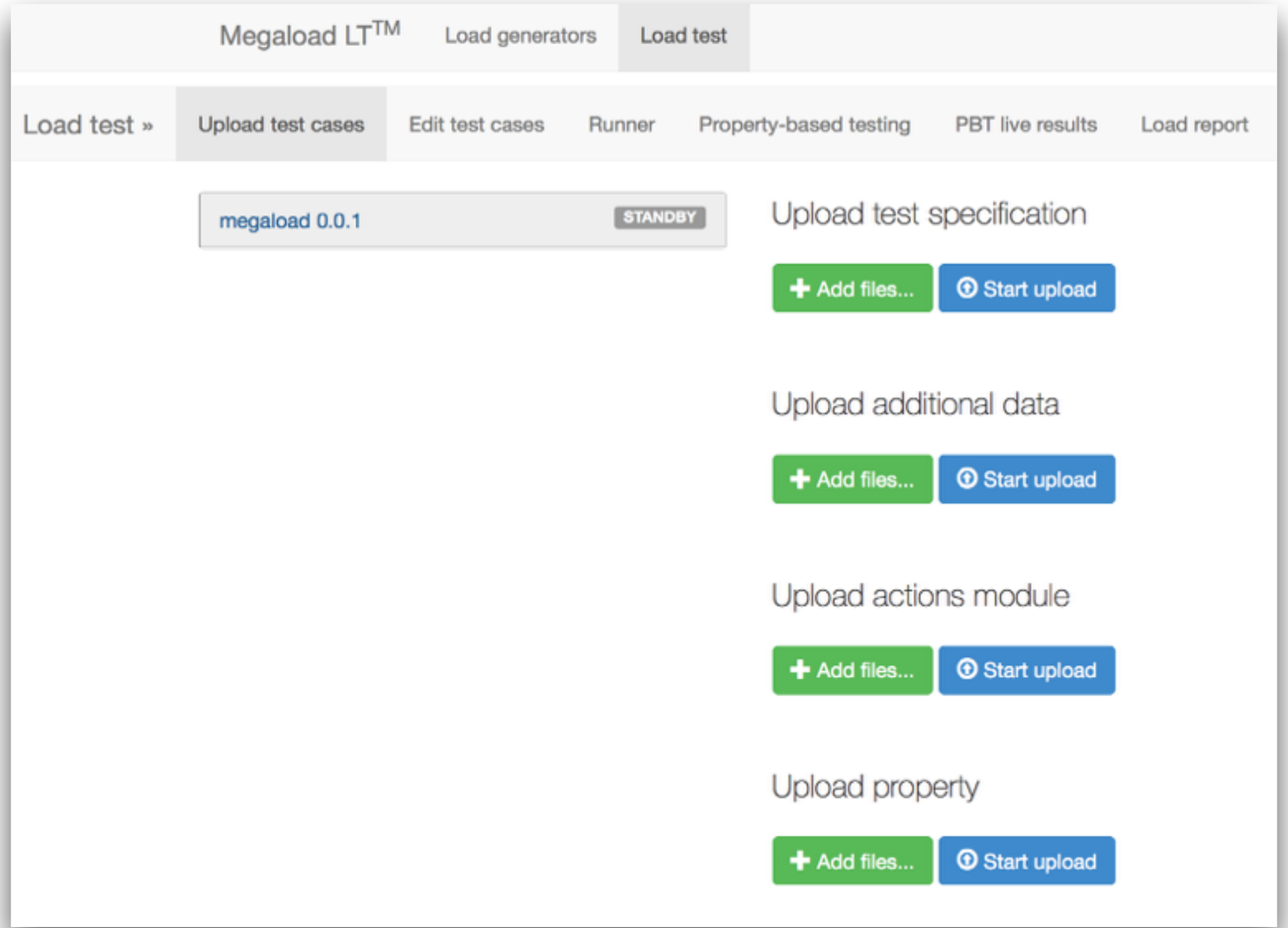


Upload test case



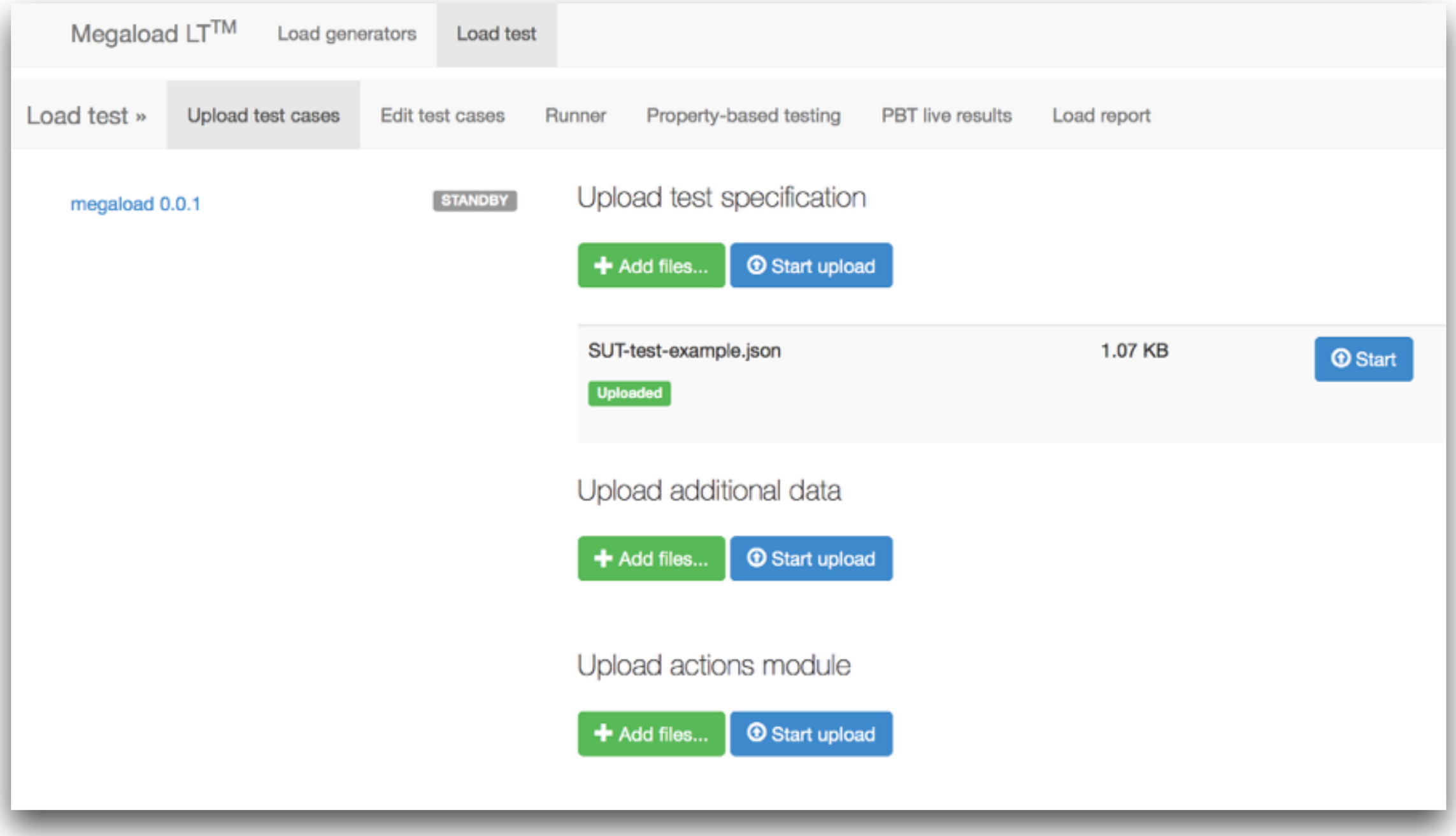
Always select your load generator in the left bar

Upload test case

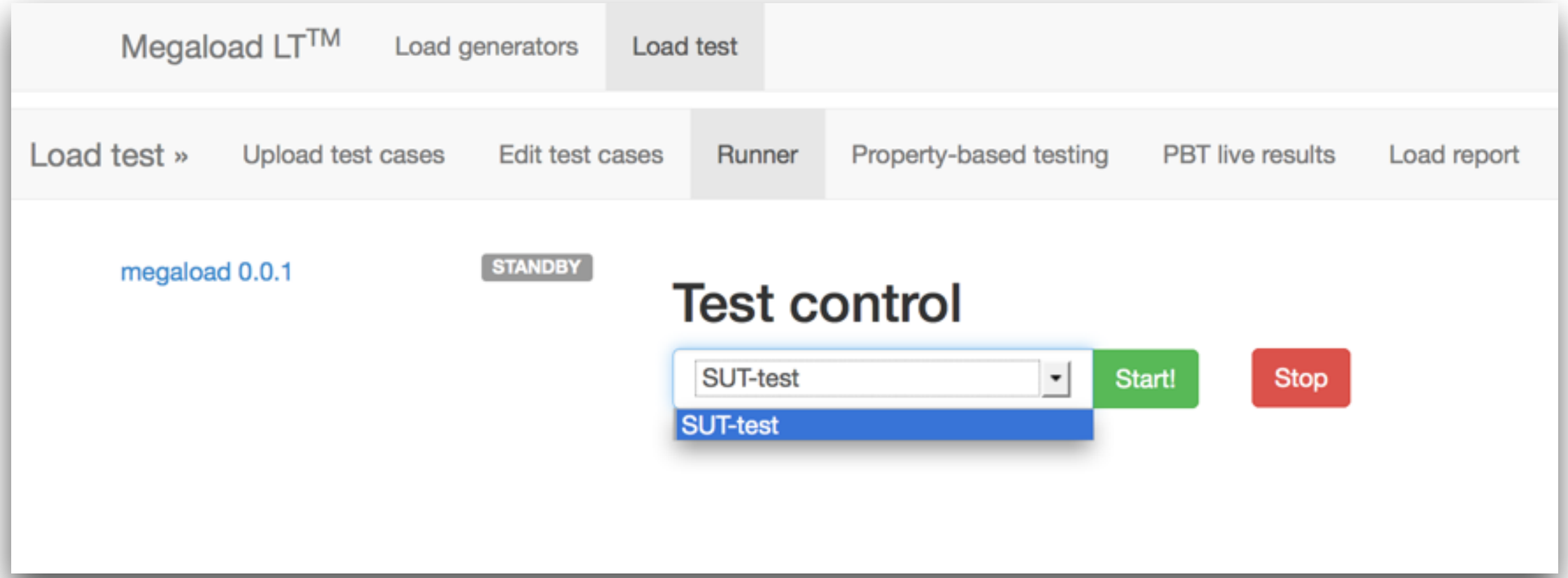


Upload test case

Your test case is uploaded!

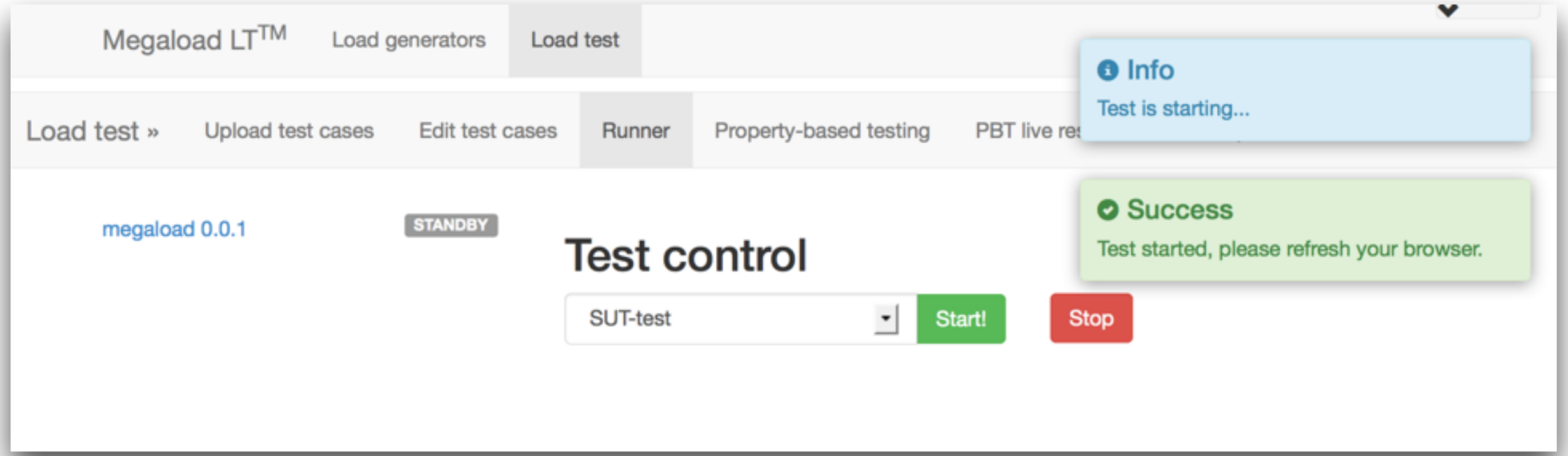


Start load test



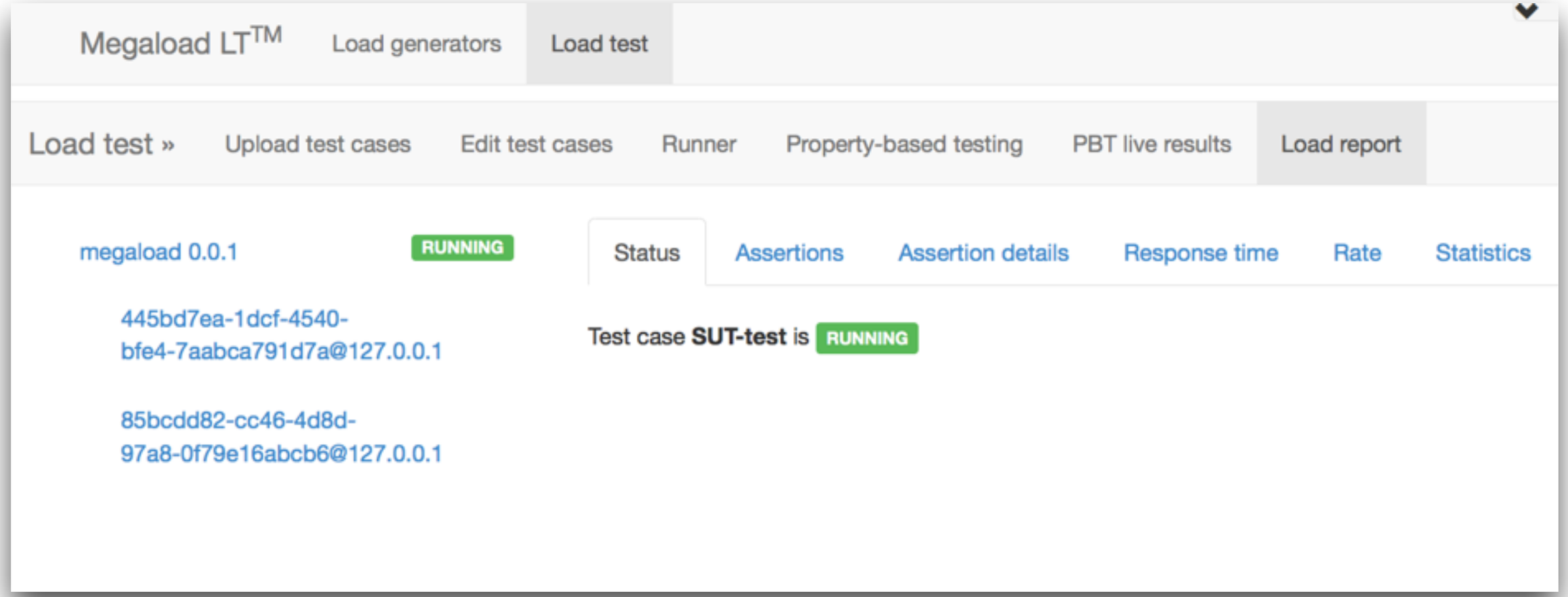
Start load test

Your test case is running!



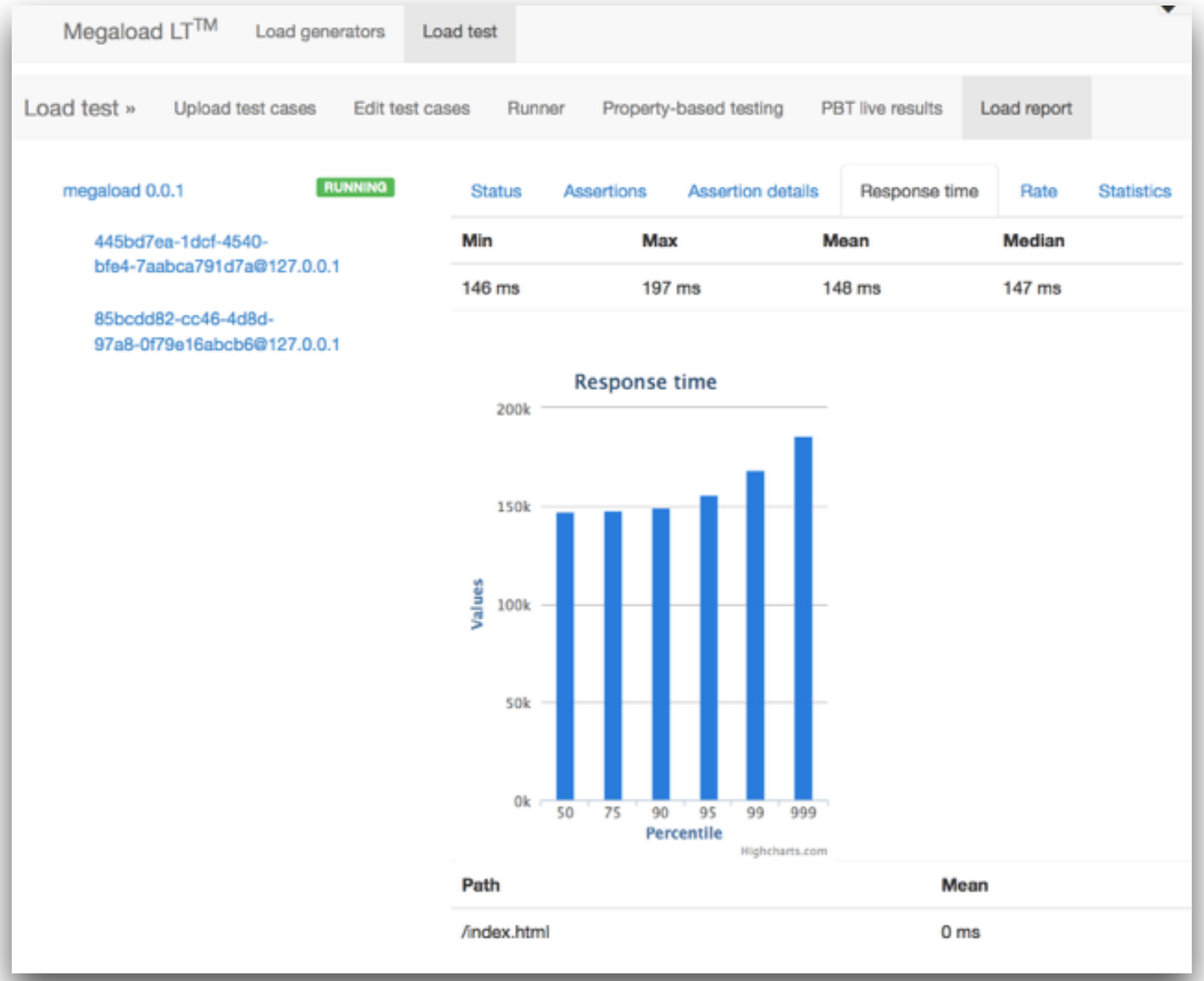
Load test results

Test state



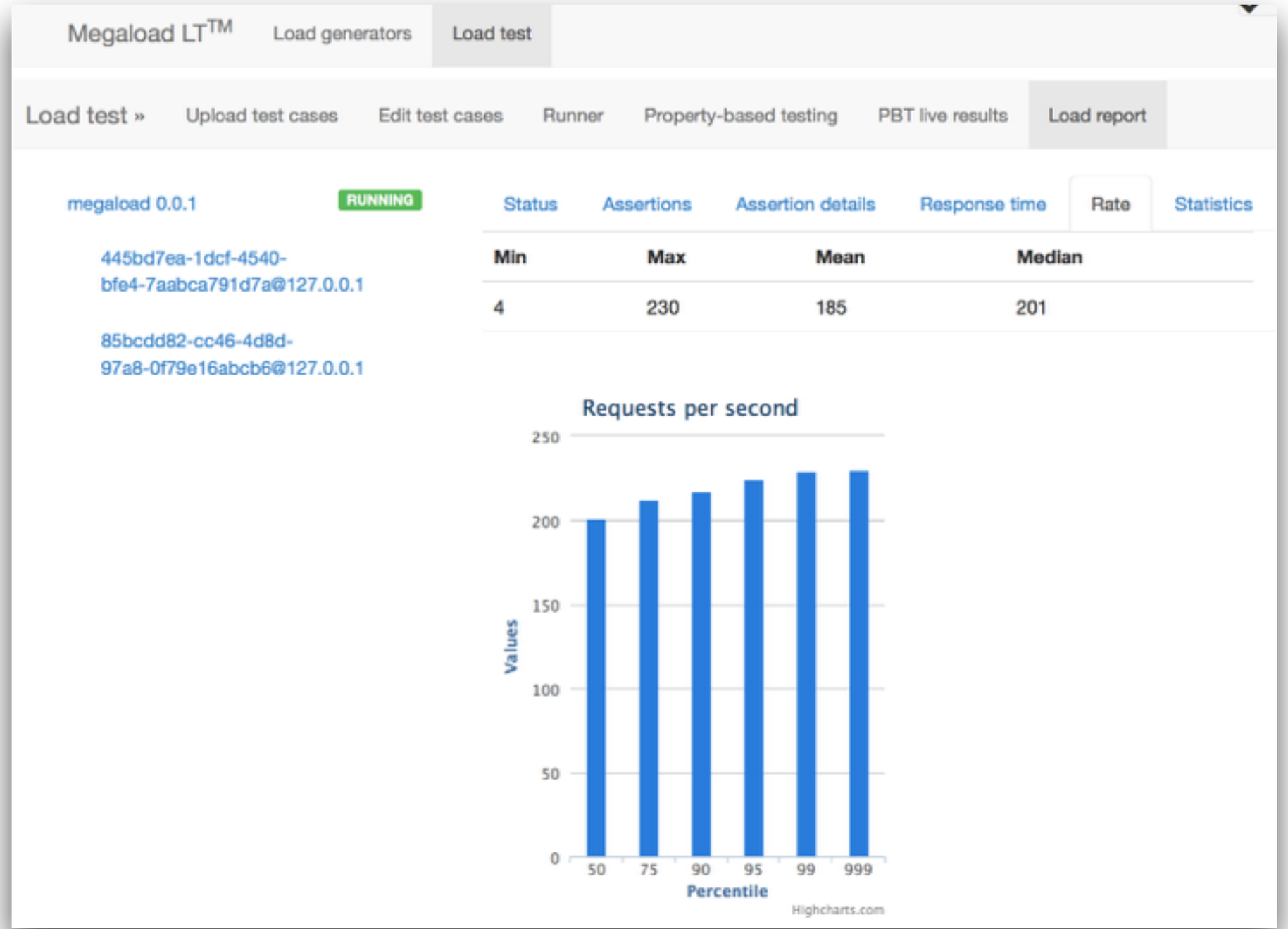
Load test results

Response time

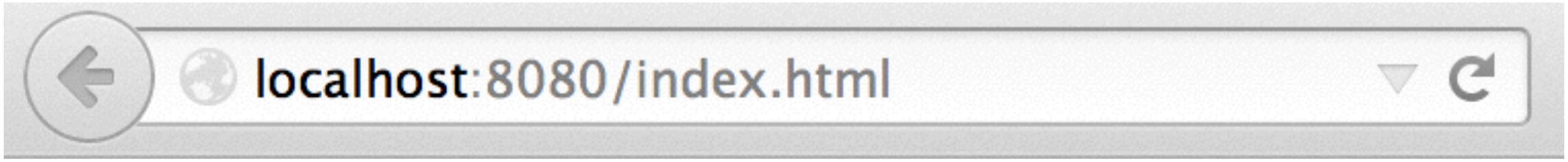


Load test results

Rate

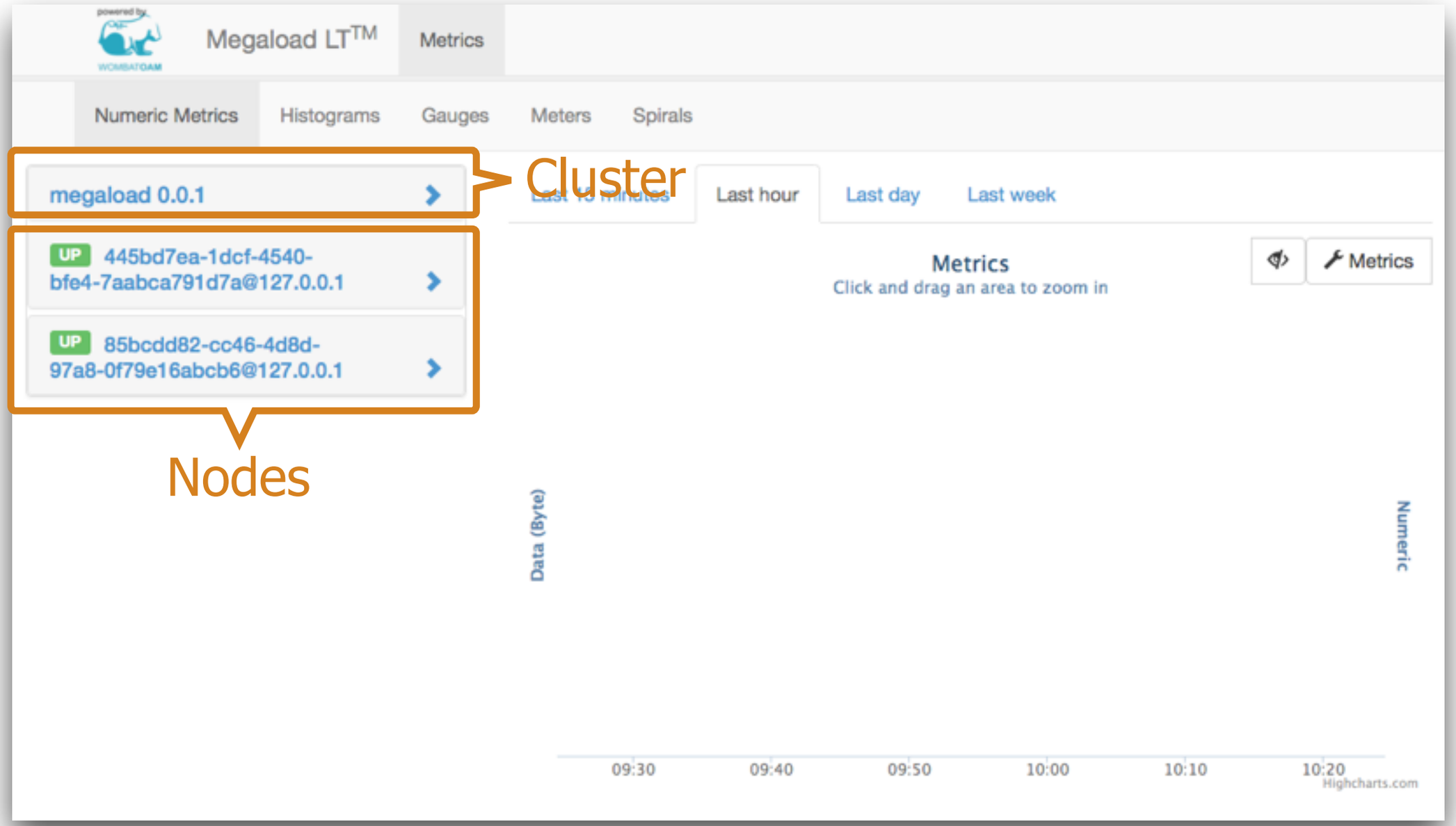


Do you want more metrics?



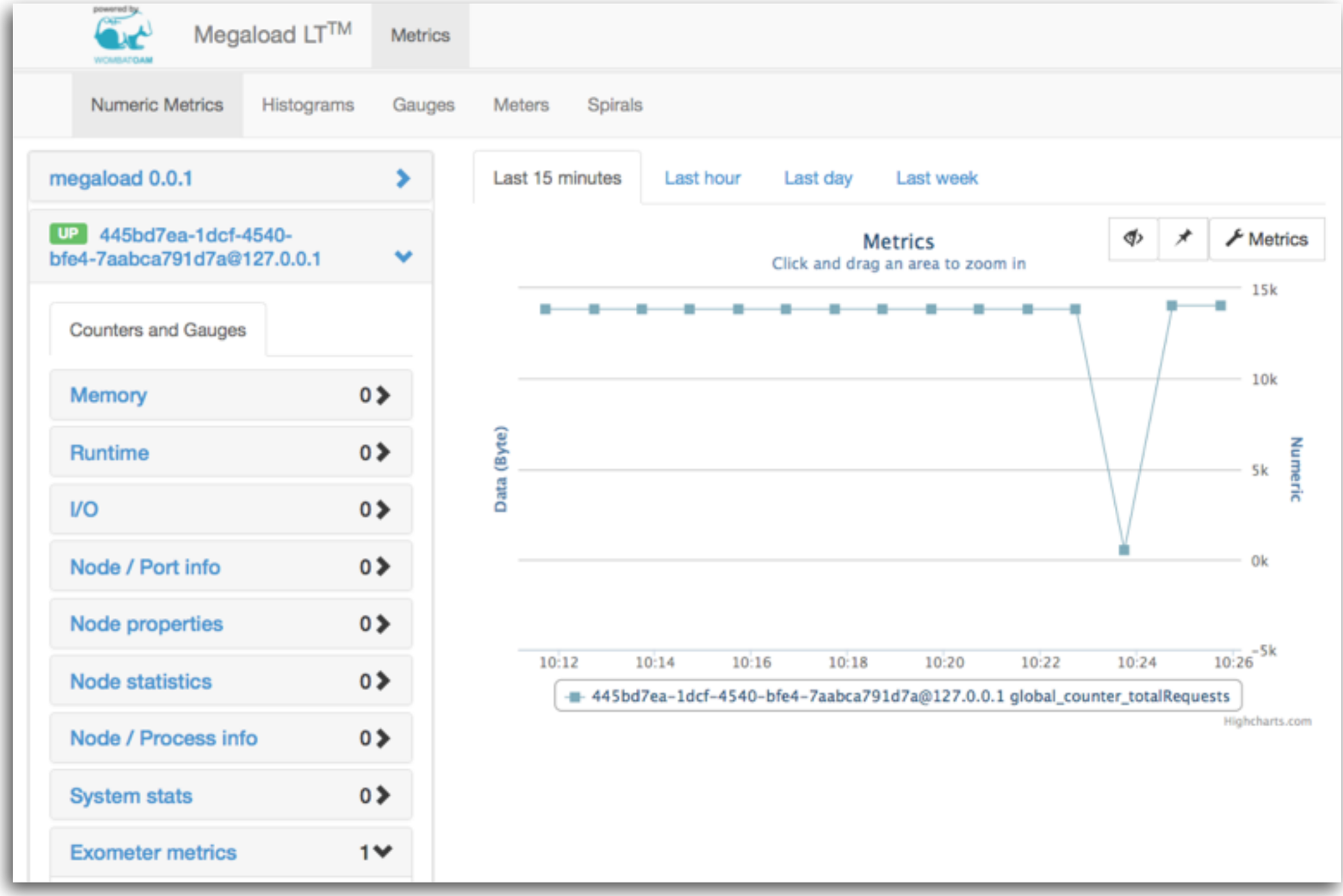
Metrics

Real-time metrics dashboard



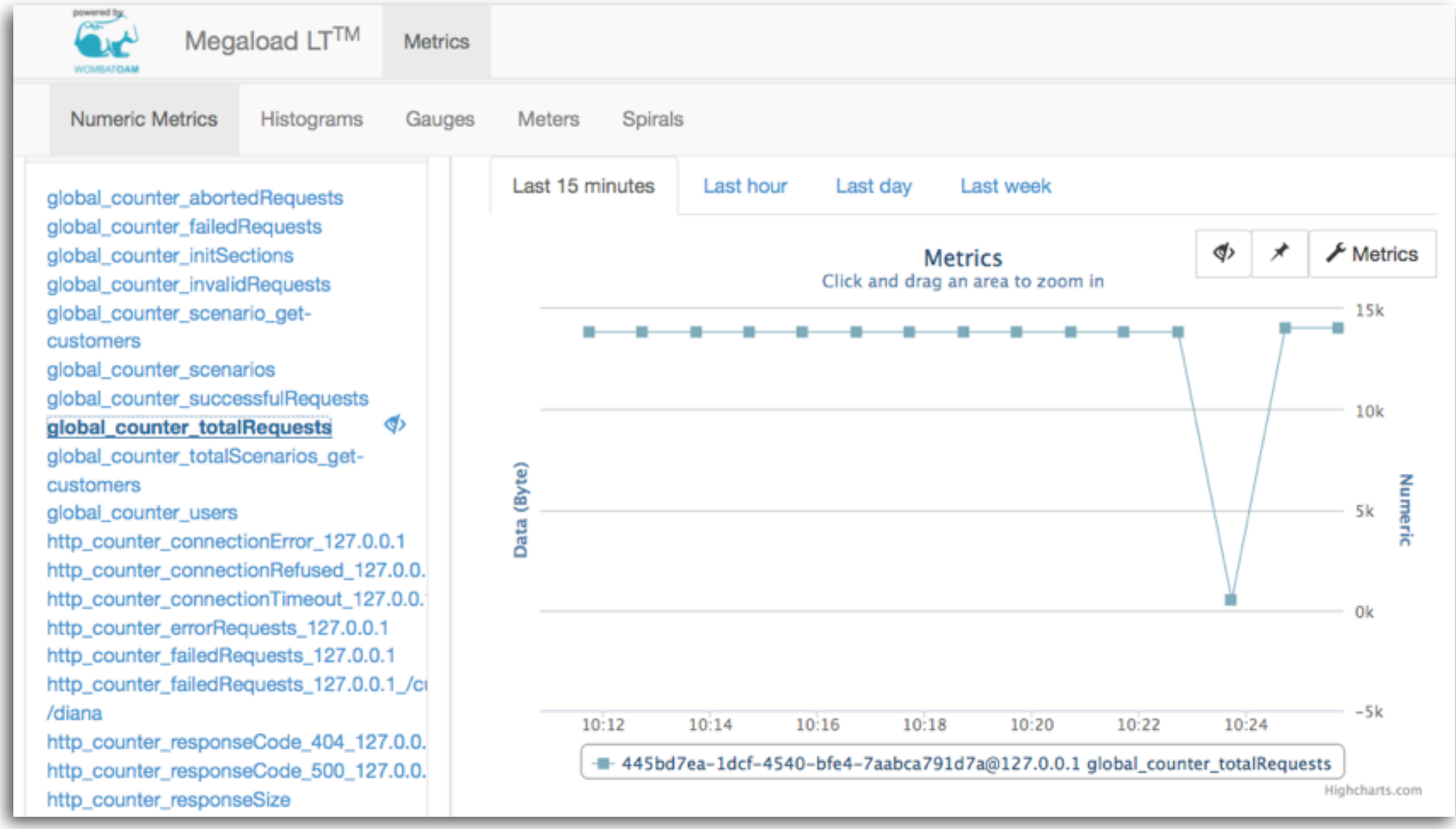
Metrics

Load test counters - node



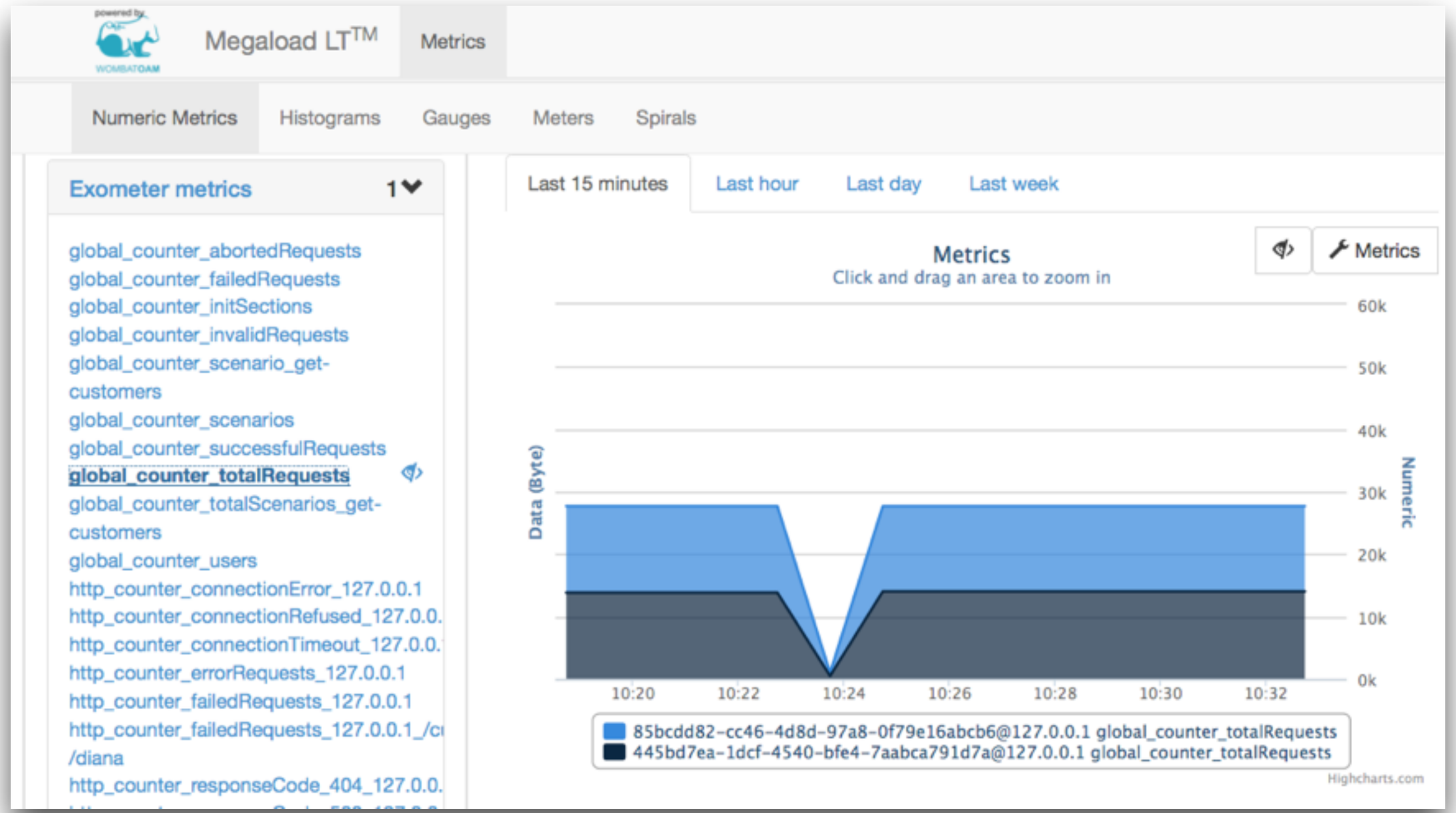
Metrics

Built-in counters



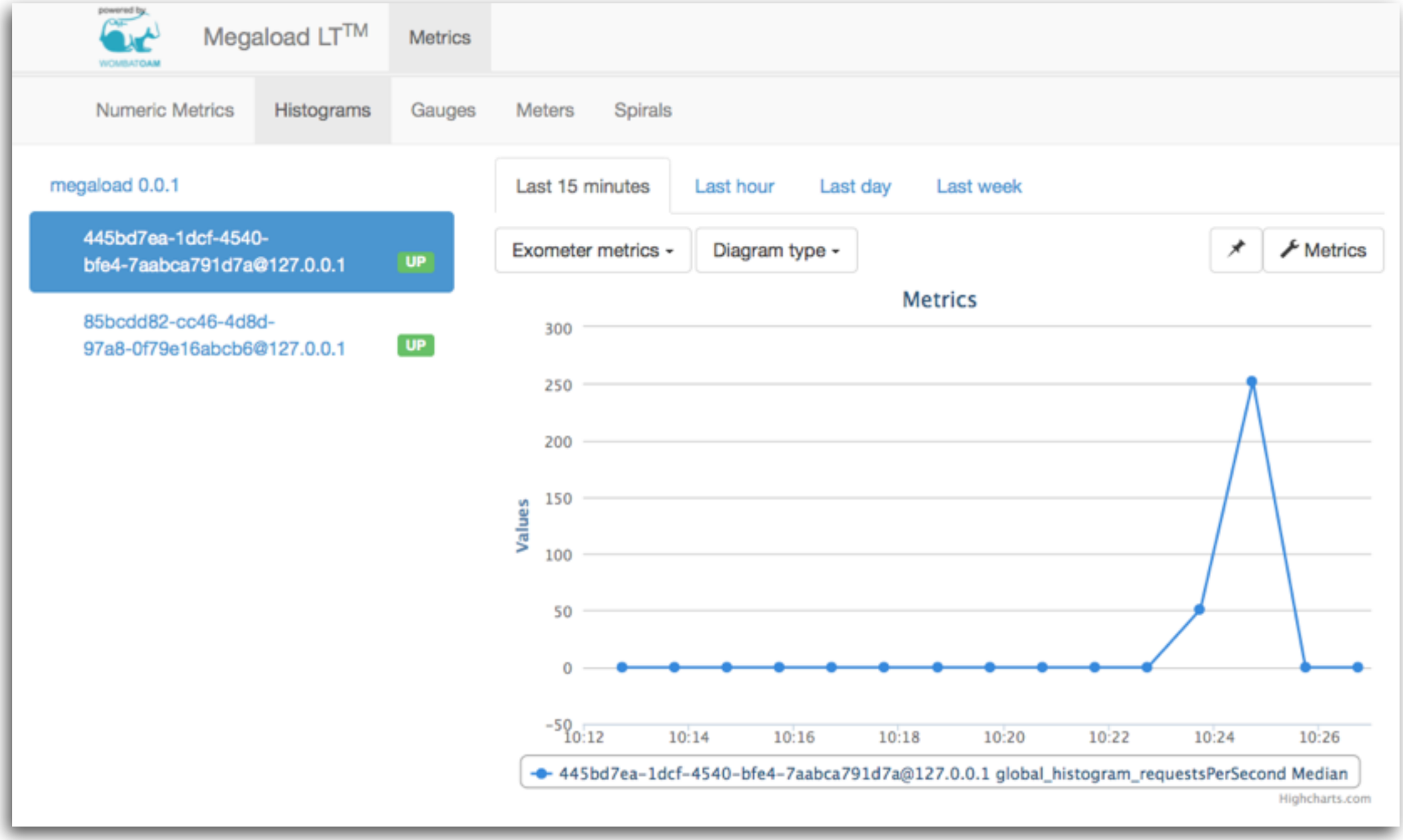
Metrics

Aggregated counters - cluster



Metrics

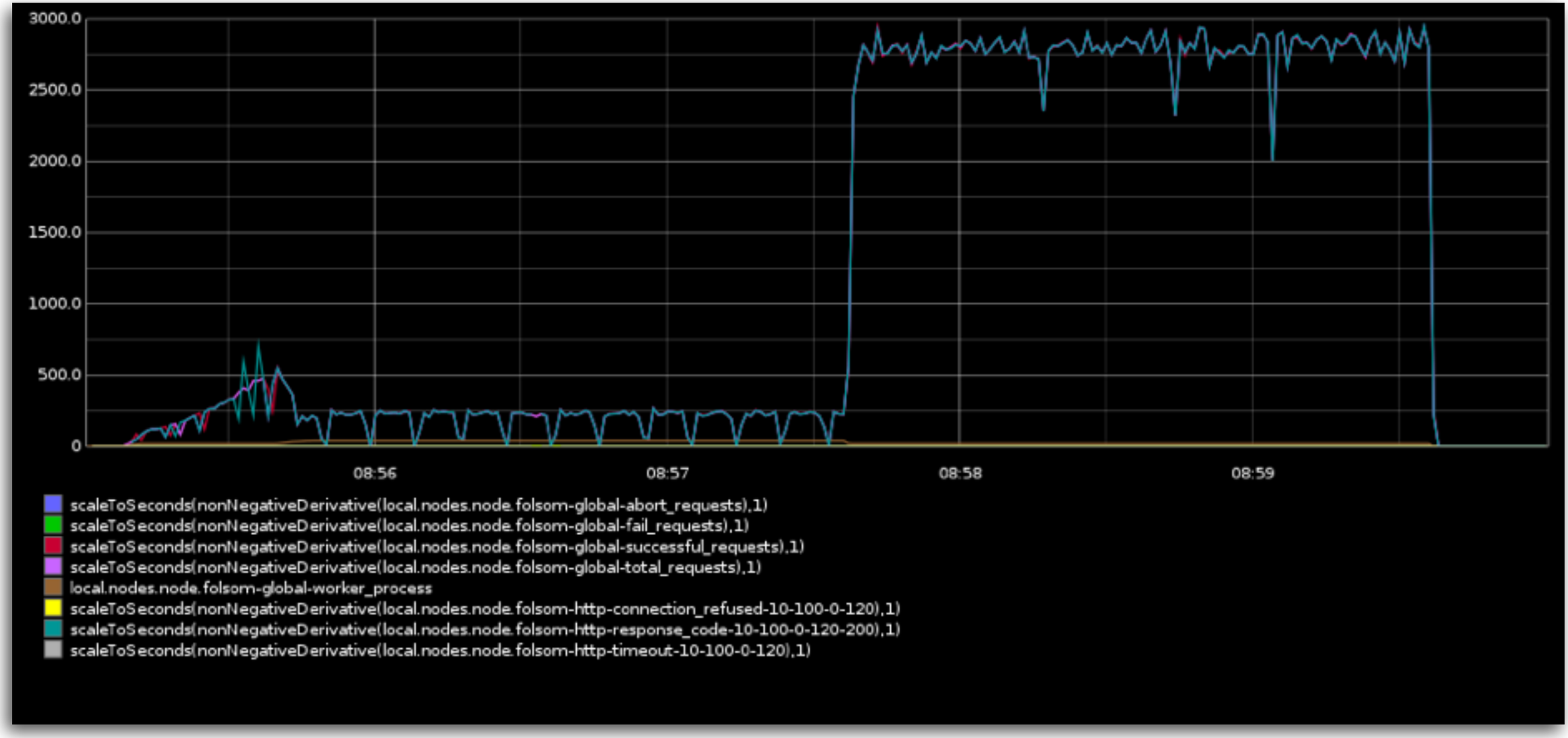
Histograms - node



Need more power?

Flexible, configurable, scalable

graphite



Graphite

Use your own Graphite deployment
Configure Megaload

etc/sys.config

```
{graphite, [
            {carbon_host, "127.0.0.1"},
            {carbon_udp_port, 2003}
           ]}
```

That's all, you got real-time metrics in Graphite

Back to our test case

Test assertions

Let's include some checks in our test

```
{
 "test" :
     {
      "id" : "SUT-test",
      "phases" : ["SUT-low"],
      "plugins" : ["SUT-server"],
      "assertions" : <ASSERTIONS>
     }
}
```

Test assertions

Counters & histograms

```
[
 {"assert-counter" :
    {"id" : "global_counter_failedRequests",
     "metric" : "count",
     "filter" : {"eq" : 0}}
 },
 {"assert-histogram" :
    {"id" : "http_histogram_responseTime",
     "statistic" : "mean",
     "filter" : {"lt" : 150000}}
 }
]
```

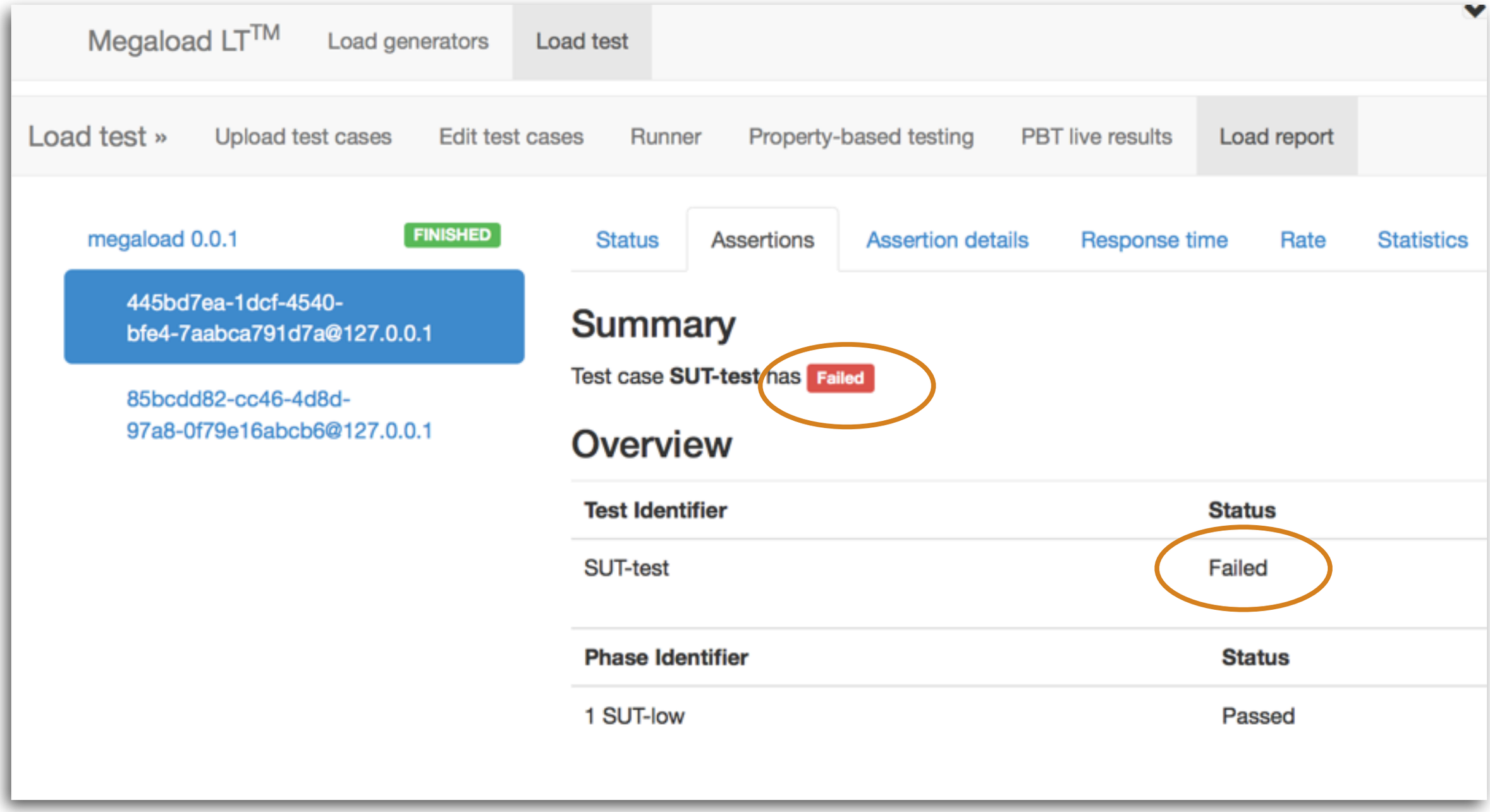
Test assertions

It’s ready to run!

```
{
 "test" :
     {
      "id" : "SUT-test",
      "phases" : ["SUT-low"],
      "plugins" : [“SUT-server”],
      "assertions" : [{"assert-counter" :
                      {"id" : "global_counter_failedRequests",
                       "metric" : "count",
                       "filter" : {"eq" : 0}}
                      },
                  {"assert-histogram" :
                      {"id" : "http_histogram_responseTime",
                       "statistic" : "mean",
                       "filter" : {"lt" : 150000}}
                  }]
     }
}
```

Load test results

Assertions overview



Load test results

Assertion details

Megaload LT™ | Load generators | Load test

Load test » | Upload test cases | Edit test cases | Runner | Property-based testing | PBT live results | Load report

megaload 0.0.1 FINISHED

445bd7ea-1dcf-4540-bfe4-7aabca791d7a@127.0.0.1

85bcdd82-cc46-4d8d-97a8-0f79e16abcb6@127.0.0.1

Status | Assertions | Assertion details | Response time | Rate | Statistics

Phase | Test

SUT-test

Status	Assertion	Value
false	{"assert-histogram": {"id":"http_histogram_responseTime","statistic":"mean","filter": {"lt":150000}}}	165649
true	{"assert-counter": {"id":"global_counter_failedRequests","metric":"count","filter":{"eq":0}}}	0

Request assertions

Validate response code and content

Automatic update of counters

global_counter_successfulRequests
global_counter_failedRequests

http_counter_successfulRequests_<SUT_IP>
http_counter_failedRequests_<SUT_IP>

Request assertions

Validate response code and content

```
{
 "http-request" :
     {
      "plugin_id" : <PLUGIN_ID>,
      "method" : <HTTP_REQUEST_METHOD>,
      "path" : <HTTP_REQUEST_URI>,
      "assert_status" : <HTTP_RESPONSE_STATUS_CODE>,
      "assert_body" : <ACTION_CHECK>
     }
}
```

Request assertions

Validate response code

```
{
 "http-request" :
     {
      "plugin_id" : "SUT-server",
      "method" : "GET",
      "path" : "/customers",
      "assert_status" : "200"
     }
}
```

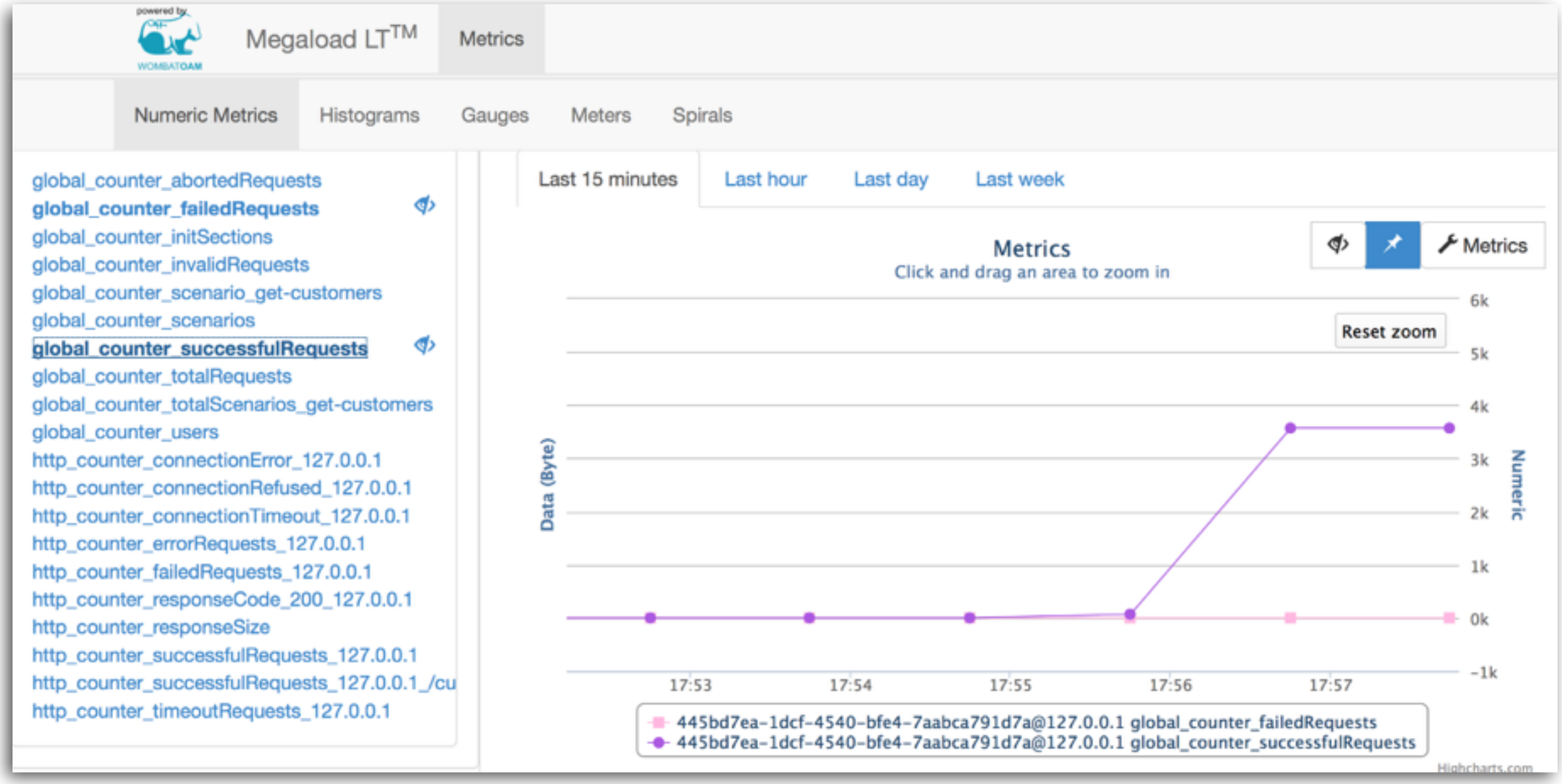
Request assertions

Validate content

```
{
 "http-request" :
     {
      "plugin_id" : "SUT-server",
      "method" : "GET",
      "path" : "/customers",
      "assert_body" :
         {"jsonpath-value" :
                {"path" : "body.count",
                 "value" : 4}}
     }
}
```

Request assertions

This is the result ...



... not very interesting

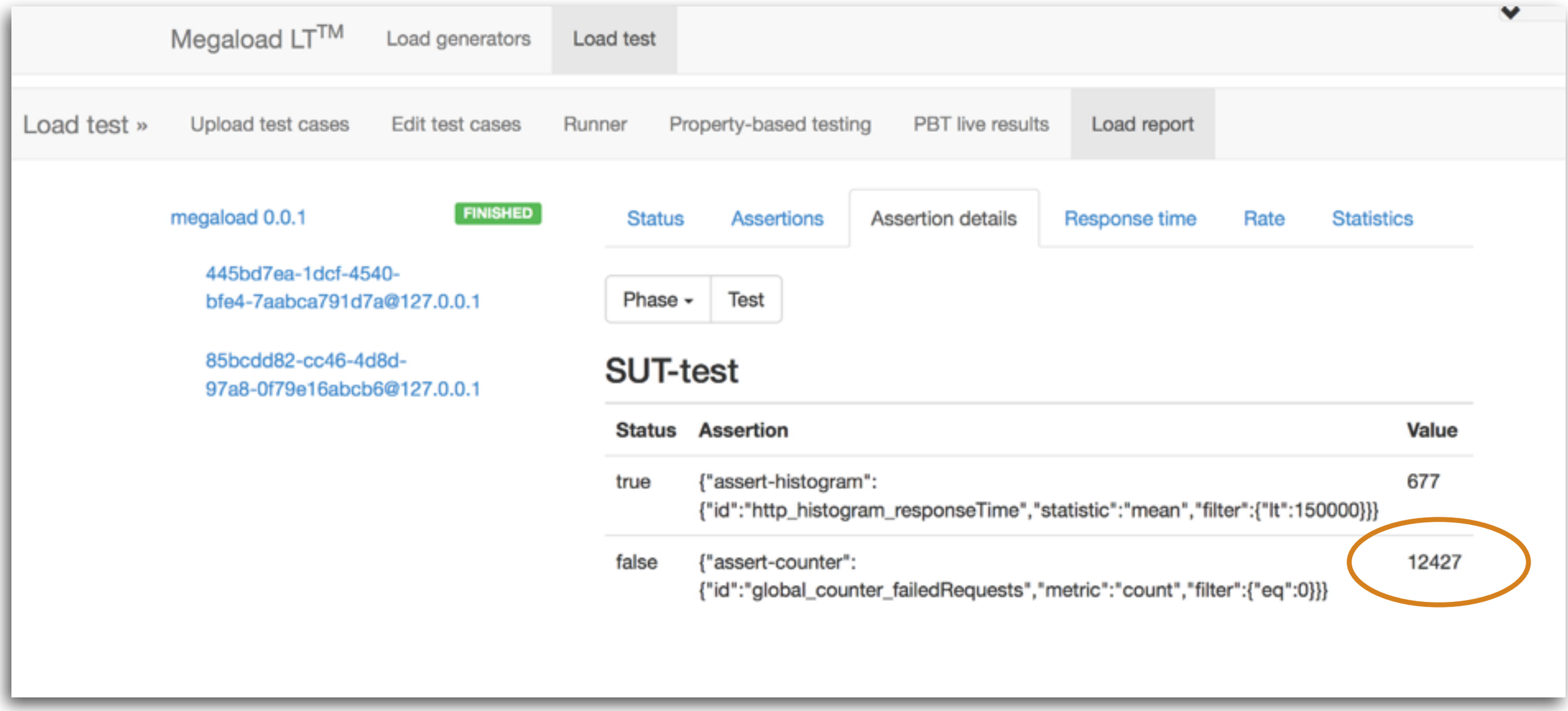
Request assertions

Let's try a different request. Are we customers of the shop?

```
{
 "http-request" :
     {
       "plugin_id" : "SUT-server",
       "method" : "GET",
       "path" : "/customer/diana",
       "assert_status" : "404"
     }
}
```

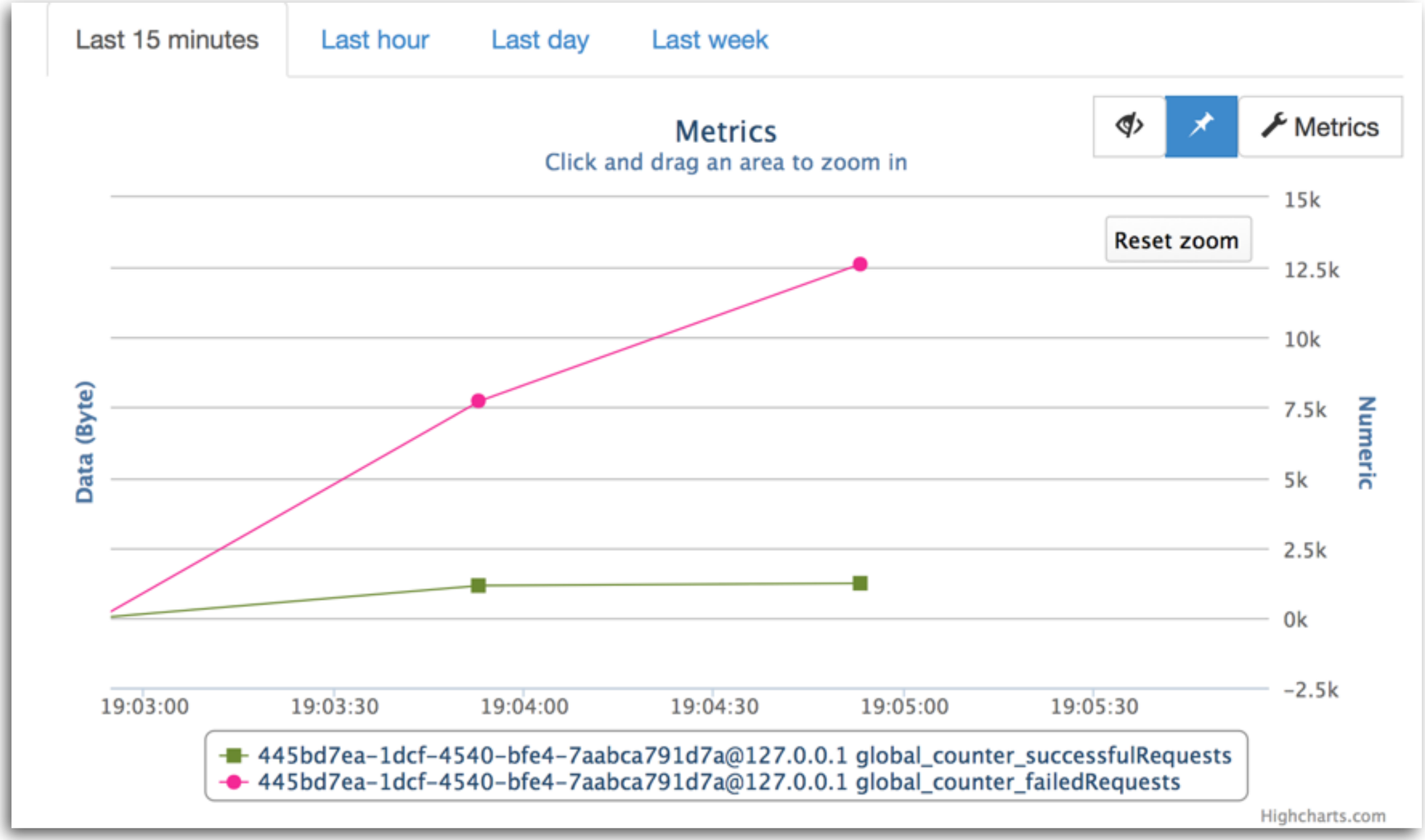
Request assertions

Now we have failing requests



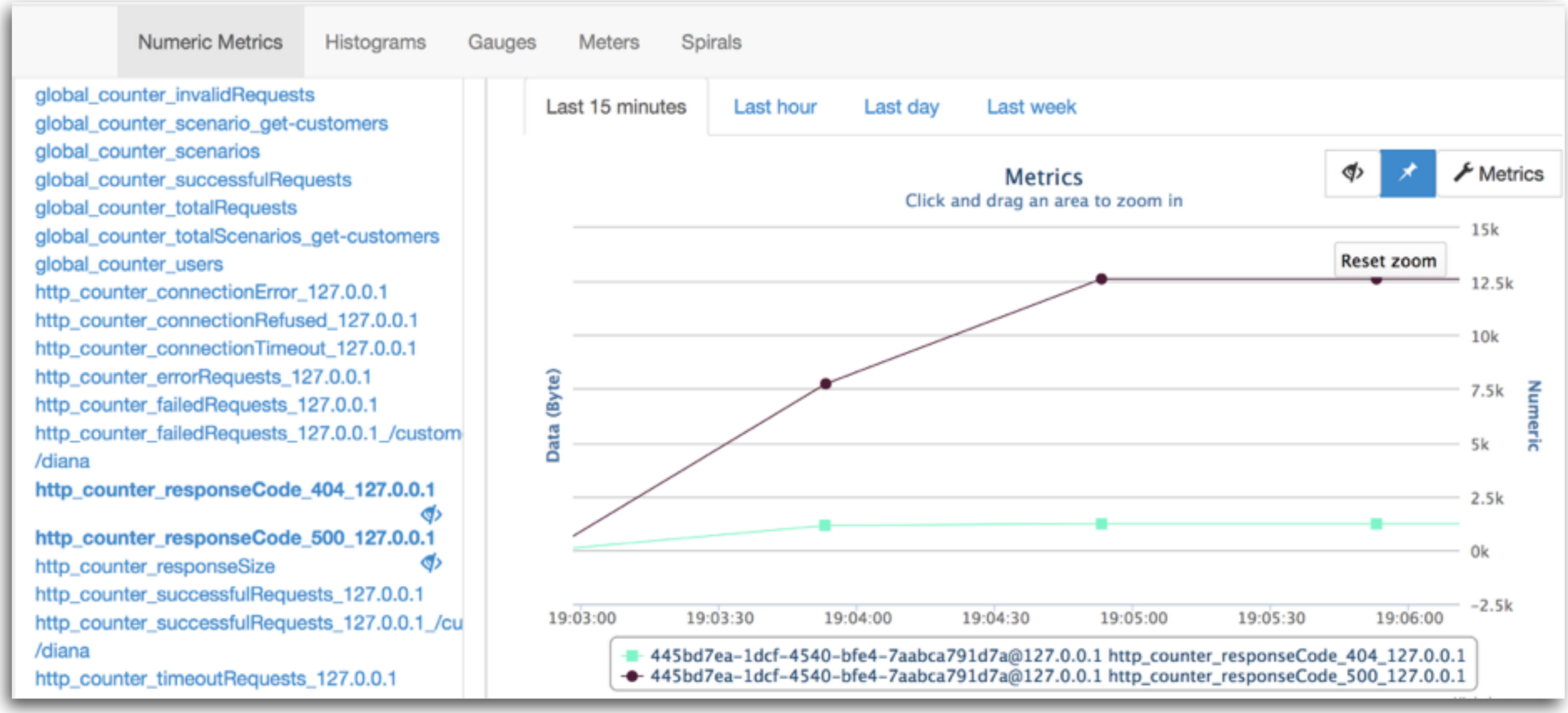
Request assertions

Not all of them failed



Request assertions

We got 505 response codes!



Questions?

Thanks for coming!