

Tutorial

Load Testing Made Easy

Diana Corbacho

diana.corbacho@erlang-solutions.com



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



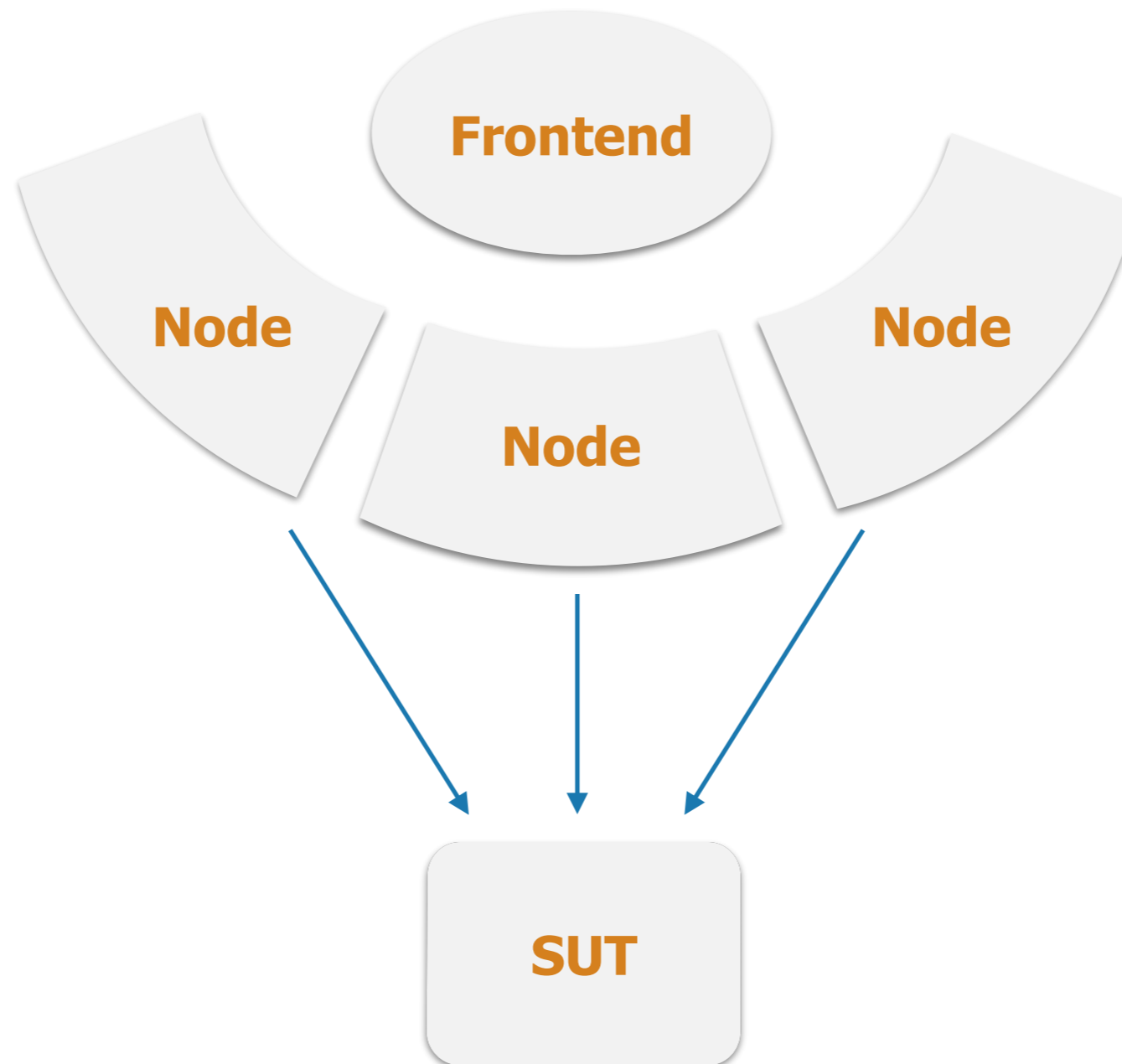
Property-based testing for
web services and internet
applications

FP7 grant €3.39 million

9 partners from 3 countries

Megaload offers...

Scalable cloud platform to run load tests



Megaload offers...

Web interface

Megaload LT™ Load generators Load test

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

Edit here your test cases!

Save them in your local drive and then upload to your chosen load generator

megaload 0.0.1 **STANDBY**

```
103     }
104   ]
105 }
106 },
107 {
108   "scenario": {
109     "id": "main-page",
110     "weight": 1,
111     "actions": [
112       {
113         "http-request": {
114           "plugin_id": "SUT_server",
115           "method": "GET",
116           "path": "/",
117           "assert_status": "200"
118         }
119       }
120     ]
121   }
122 }
123 ]
```

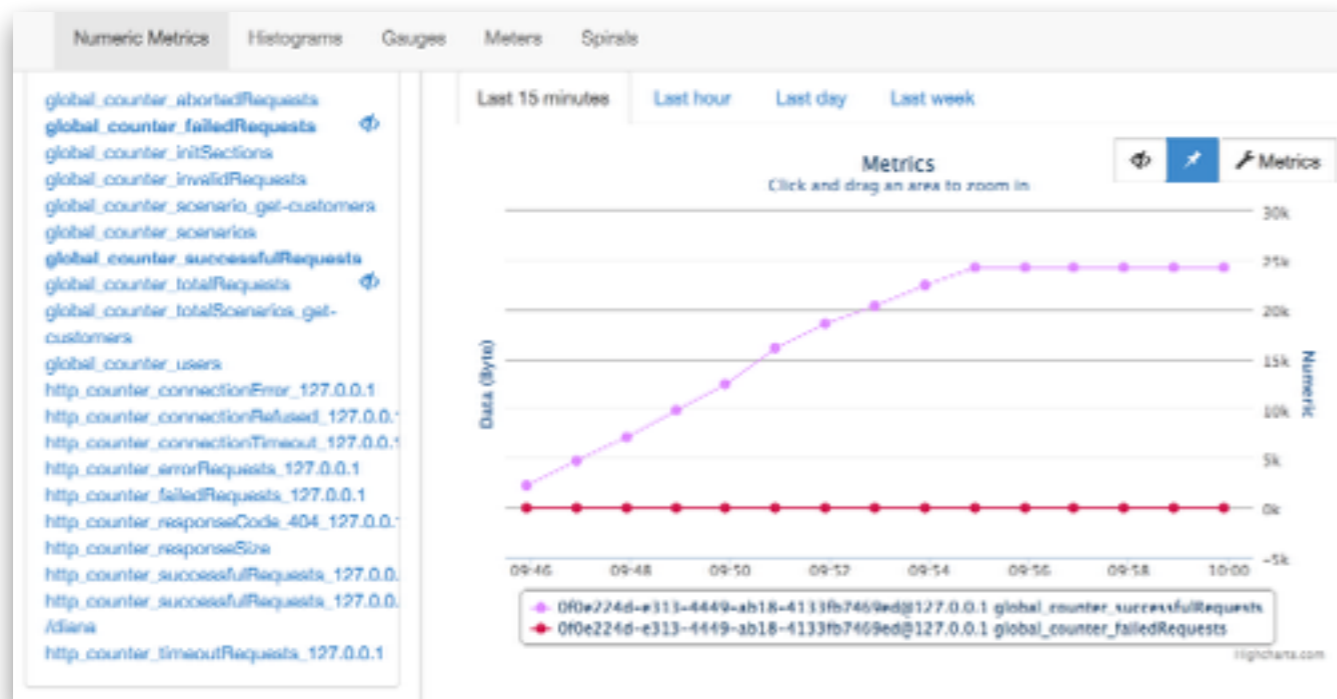
Megaload offers...

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
```

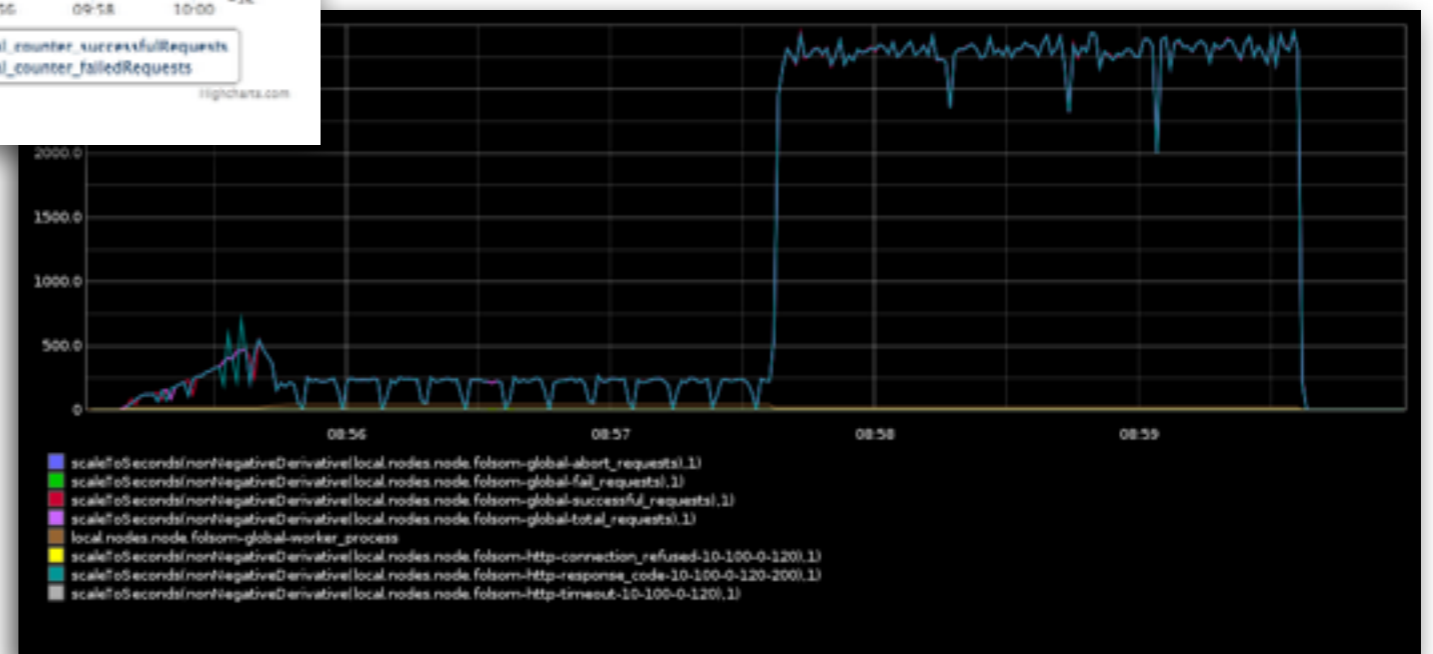
Megaload offers...

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>

```
-module(ml_xmpp).  
  
-compile({parse_transform, exprs}).  
-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}.
```

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


Megaload offers...

Test assertions

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

The image displays two overlapping screenshots of the Megaload Property-based testing (PBT) interface. The top screenshot shows the 'PBT live results' tab with a 'PROPERTY' status and a log of test results. The bottom screenshot shows the 'FINISHED' status with control buttons for starting, stopping, and downloading results.

Property-based testing | PBT live results | Load report

PROPERTY

```
Users: 53 ResponseTime: 818811.2961165048
Failed! After 1 tests.
53
Users: 1 ResponseTime: 107278.88888888889
Users: 3 ResponseTime: 99981.96363636364
Users: 7 ResponseTime: 162288.8064516129
Users: 15 ResponseTime: 238856.15151515152
Users: 31 ResponseTime: 514422.13612565445
Shrinking.Users: 23 ResponseTime: 413478.0220994475
.Users: 19 ResponseTime: 353439.6835443038
.Users: 17 ResponseTime: 245193.08474576272
Users: 18 ResponseTime: 351367.8657718121
.(4 times)
18
```

Property-based testing | PBT live results | Load report

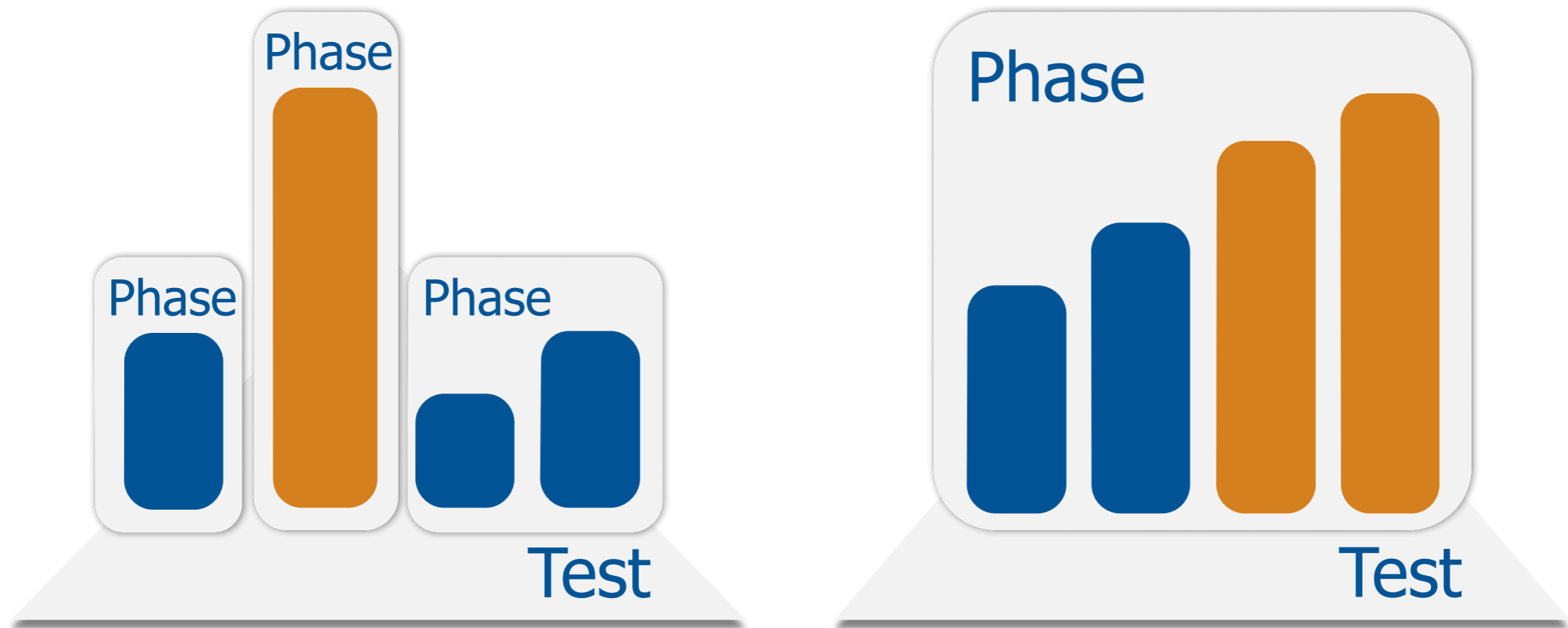
FINISHED

Property-based testing

pbt_example:prop_max_users/0

pbt-20140520092346-pbt_ε

What is a test?



A test is a phase container

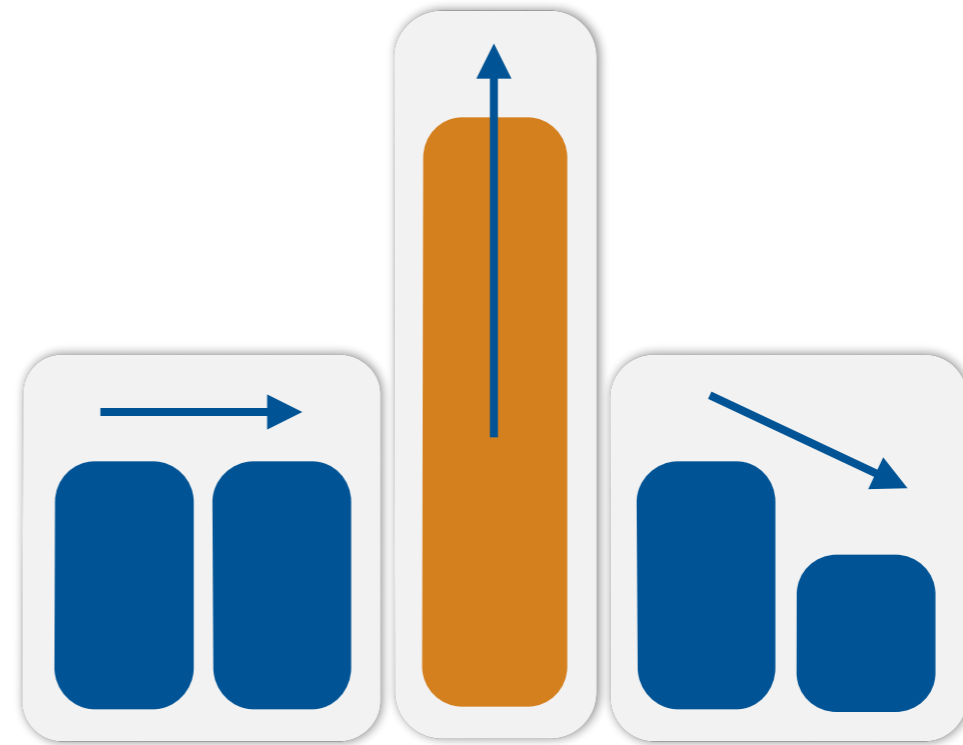
Phase

Load profile

Users

Rate

Scenarios



Scenario

Test actions

Data input

Variable

Custom counter

Think time

JsonPath

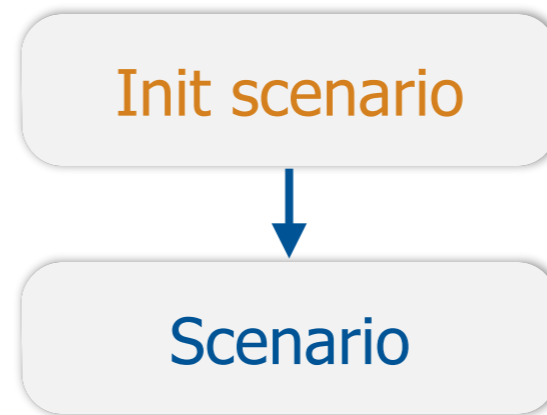
```
GET /index.html HTTP/1.1
User-Agent: curl/7.30.0
Host: 10.100.0.122:8888
Accept: */*
```

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

Requests

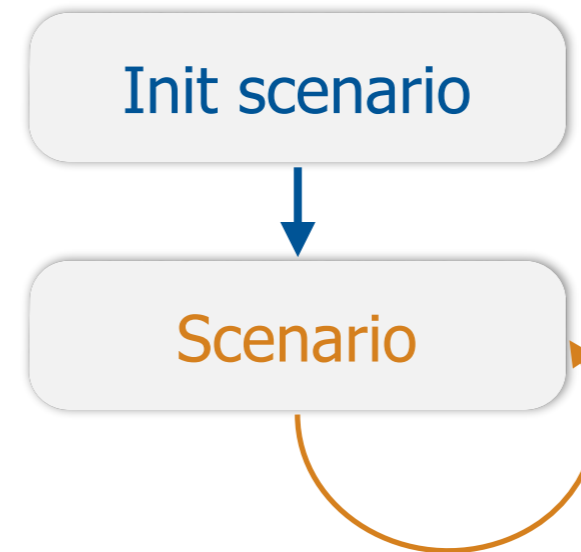
Scenario

Setup scenarios



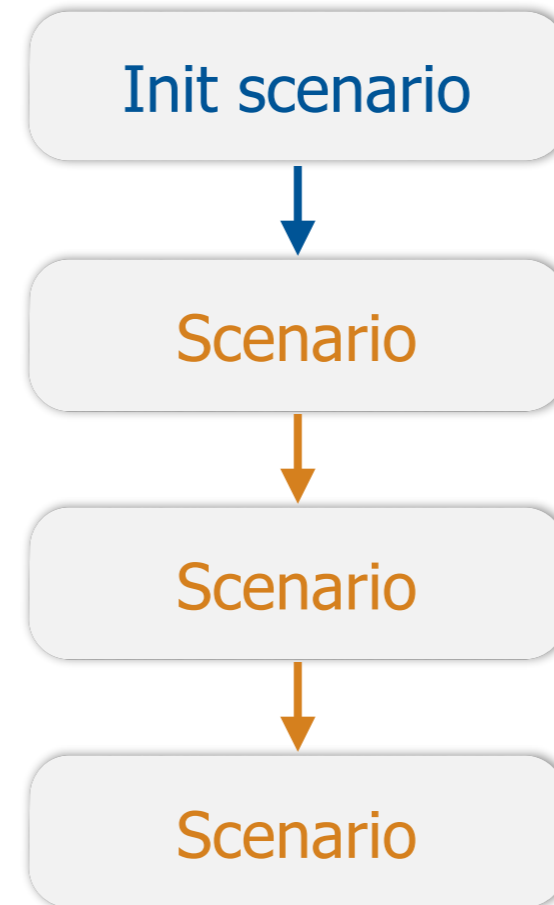
Scenario

Repeat scenarios



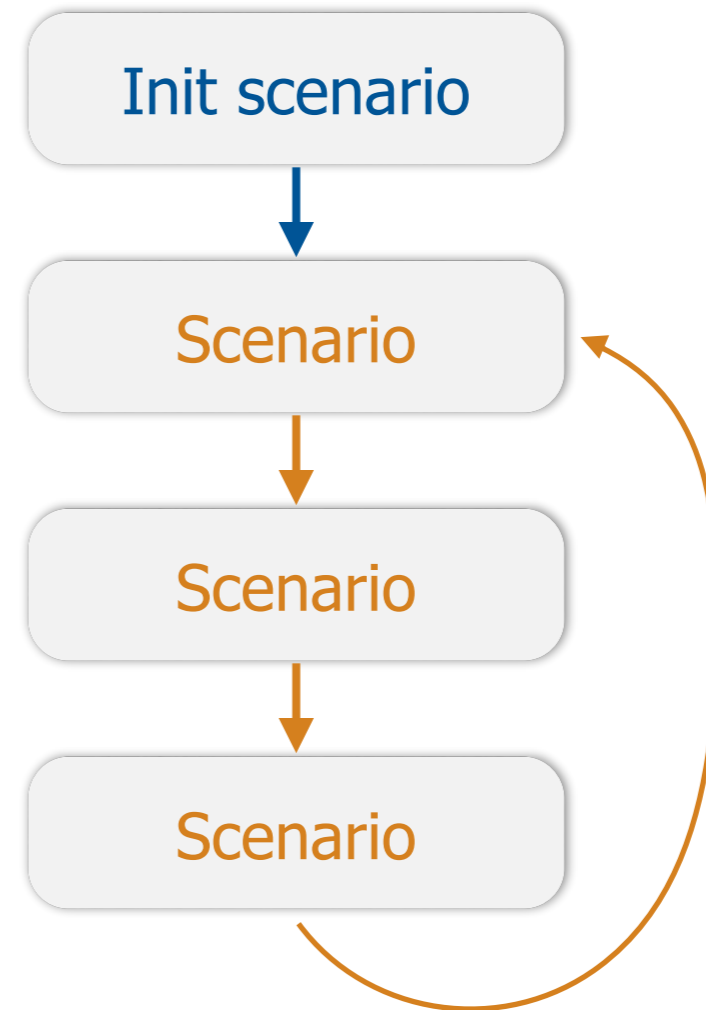
Scenario

Combine scenarios



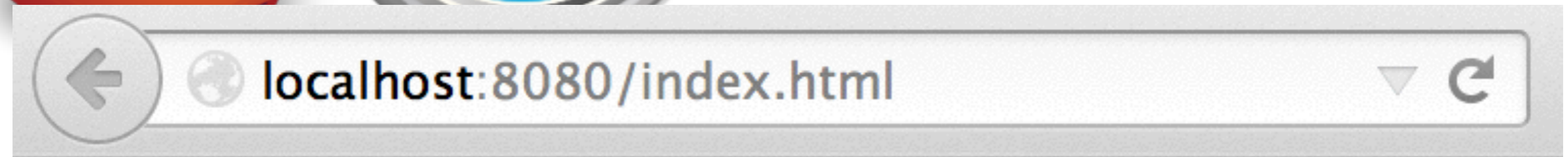
Scenario

Combine + repeat



We know the basic concepts ...

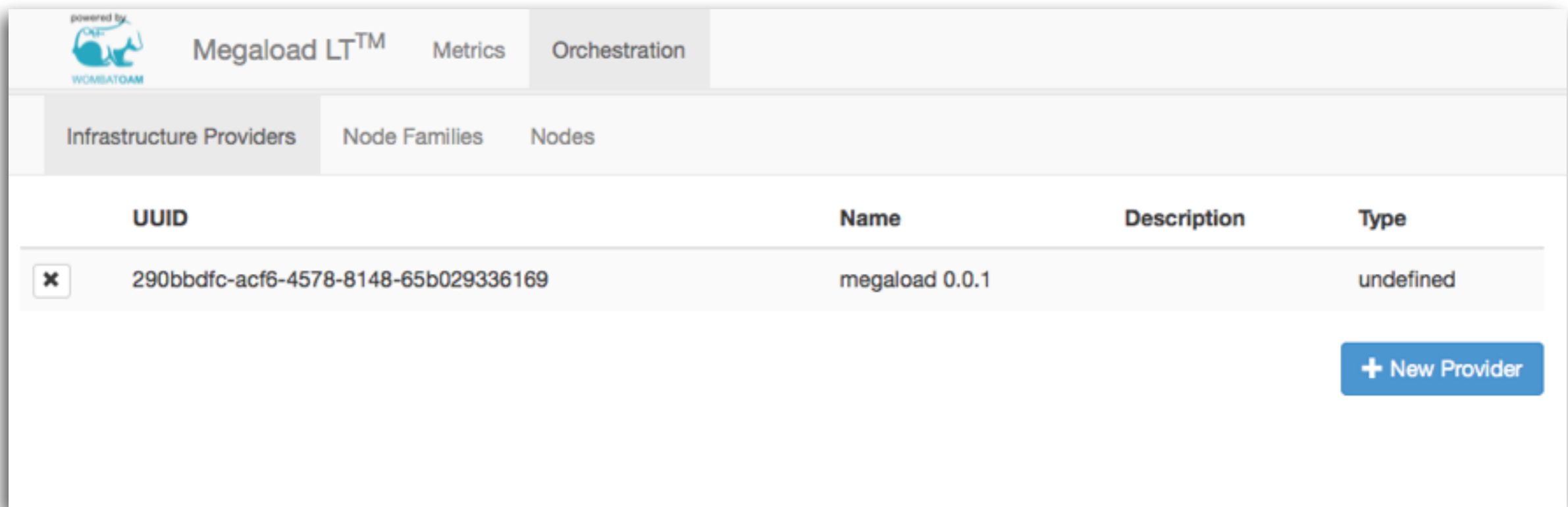
Get started




Not yet supported

Deployment

Step 1: Add infrastructure provider



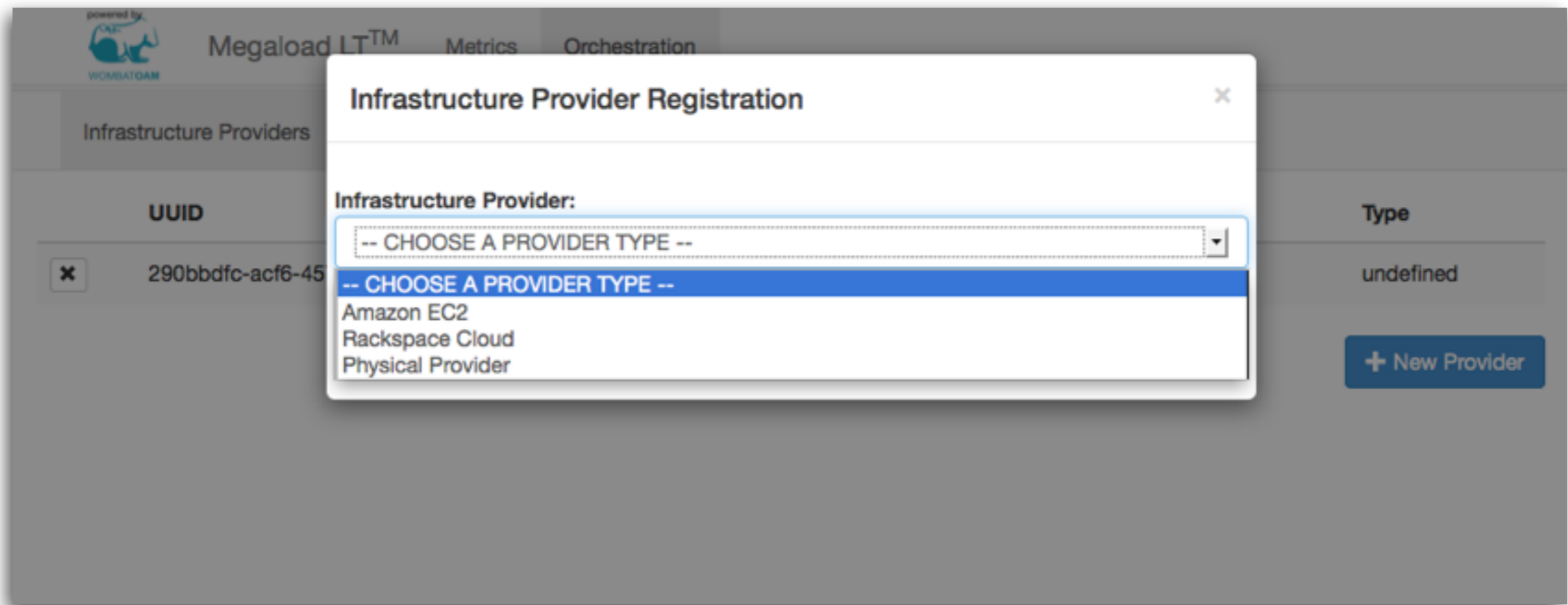
The screenshot shows the Megaload LT™ interface. At the top left, there is a logo for WOMBATOAM with the text "powered by". The main header includes "Megaload LT™", "Metrics", and "Orchestration". Below this, there are tabs for "Infrastructure Providers", "Node Families", and "Nodes". The "Infrastructure Providers" tab is active, displaying a table with the following columns: "UUID", "Name", "Description", and "Type". A single entry is shown with the UUID "290bbdfc-acf6-4578-8148-65b029336169", Name "megaload 0.0.1", and Type "undefined". A blue button labeled "+ New Provider" is located at the bottom right of the table area.

UUID	Name	Description	Type
 290bbdfc-acf6-4578-8148-65b029336169	megaload 0.0.1		undefined

[+ New Provider](#)

Deployment

Public & private clouds or physical providers



Uses



Deployment

Amazon EC2

The screenshot shows a web interface for Megaload LT™ with a modal dialog titled "Infrastructure Provider Registration". The dialog has a close button (X) in the top right corner. It contains the following fields:

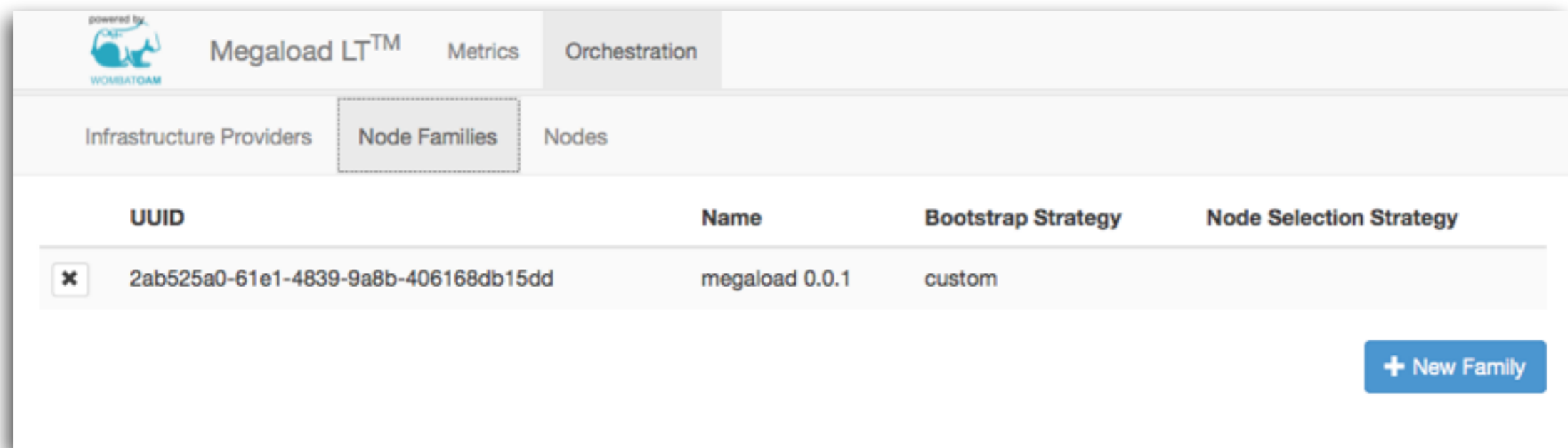
- Infrastructure Provider:** A dropdown menu with "Amazon EC2" selected.
- name:** An empty text input field.
- username:** An empty text input field.
- password:** An empty text input field.

At the bottom right of the dialog is a blue button labeled "Register Provider".

In the background, the main interface shows a table of "Infrastructure Providers" with a header "UUID" and a row containing a UUID "290bbdfc-acf6-45" and a close button (X). To the right, there is a "Type" field with the value "undefined" and a blue button labeled "+ New Provider".

Deployment

Step 2: Add node family



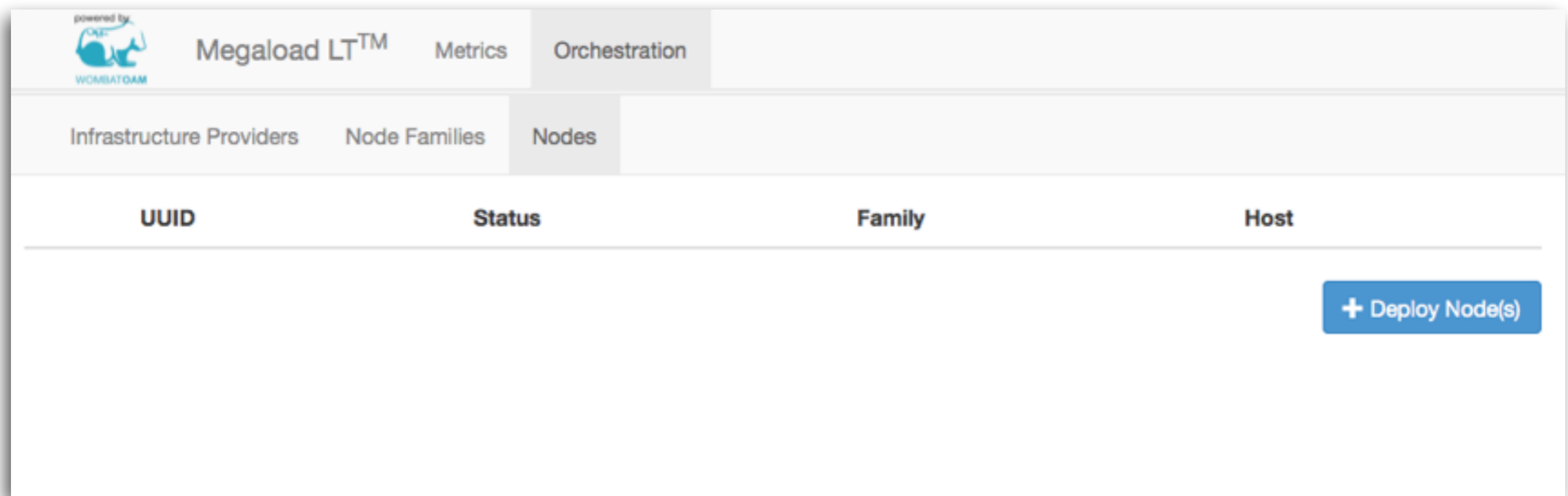
The screenshot shows the Megaload LT™ Orchestration interface. The top navigation bar includes "powered by WOMBATOAM", "Megaload LT™", "Metrics", and "Orchestration". Below this, there are tabs for "Infrastructure Providers", "Node Families" (which is highlighted with a dashed border), and "Nodes". The main content area displays a table with the following columns: "UUID", "Name", "Bootstrap Strategy", and "Node Selection Strategy". A single row is visible in the table with the following data: UUID: 2ab525a0-61e1-4839-9a8b-406168db15dd, Name: megaload 0.0.1, Bootstrap Strategy: custom. A blue button labeled "+ New Family" is located in the bottom right corner of the table area.

UUID	Name	Bootstrap Strategy	Node Selection Strategy
<input type="checkbox"/> 2ab525a0-61e1-4839-9a8b-406168db15dd	megaload 0.0.1	custom	

A node family is a **cluster** of nodes

Deployment

Step 3: Deploy nodes



The screenshot displays the Megaload LT™ web interface. At the top left, there is a logo for 'powered by WOMBATGAM'. The main navigation bar includes 'Megaload LT™', 'Metrics', and 'Orchestration'. Under 'Orchestration', there are sub-tabs for 'Infrastructure Providers', 'Node Families', and 'Nodes'. The 'Nodes' tab is currently selected. Below the navigation, there is a table with the following headers: 'UUID', 'Status', 'Family', and 'Host'. The table body is currently empty. In the bottom right corner of the table area, there is a blue button with a plus sign and the text '+ Deploy Node(s)'.

Deployment

Deploy and start 2 nodes

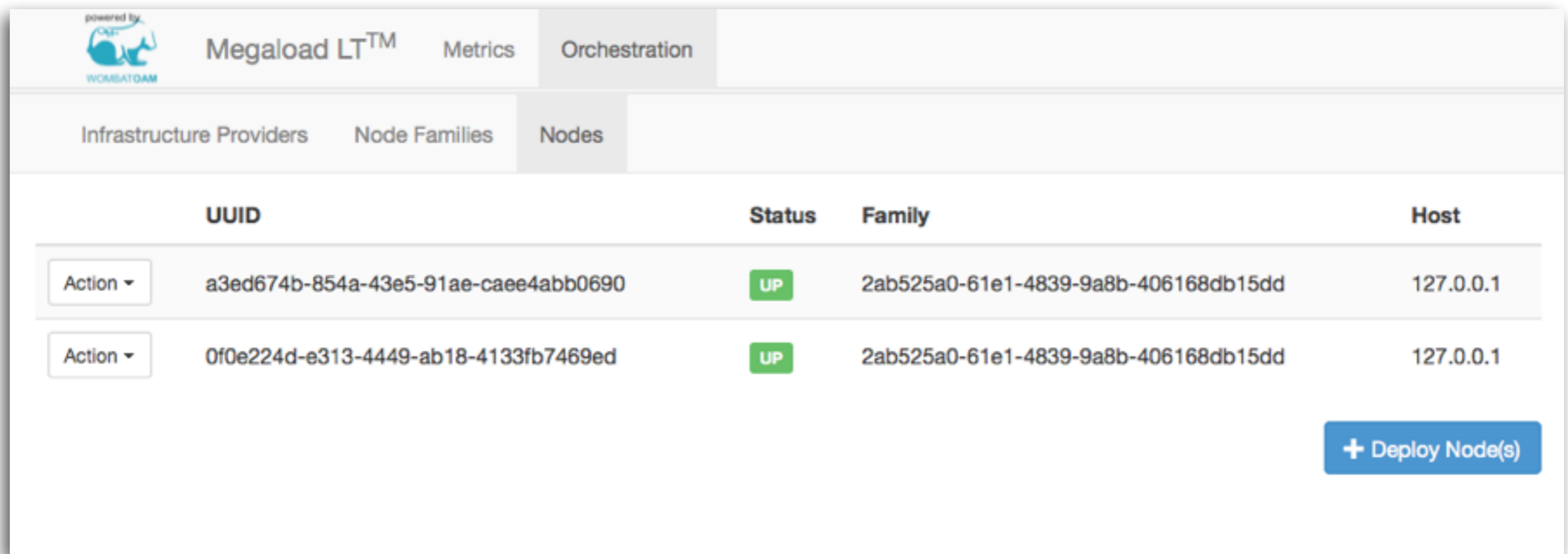
The screenshot displays the Megaload LT™ interface. At the top, there is a navigation bar with 'Megaload LT™', 'Metrics', and 'Orchestration'. Below this, a table is partially visible with columns for 'Infrastructure Providers' and 'UUID'. A modal dialog box titled 'Node(s) Deployment' is open in the center. It contains the following elements:

- Node Family:** A dropdown menu currently showing 'megaload 0.0.1'.
- Amount:** A text input field containing the number '2'.
- Automatically start node(s) after deployment
- Deploy!** A blue button to execute the deployment.

In the background, a '+ Deploy Node(s)' button is visible on the right side of the interface.

Deployment

Megaload is up!



The screenshot shows the Megaload LT™ interface. At the top, there is a logo for WOMBATDAM and the text "powered by". Below this, there are navigation tabs: "Metrics" and "Orchestration". Under "Orchestration", there are sub-tabs: "Infrastructure Providers", "Node Families", and "Nodes". The "Nodes" tab is active. Below the tabs is a table with the following columns: "Action", "UUID", "Status", "Family", and "Host". There are two rows of data, both with a status of "UP". A blue button labeled "+ Deploy Node(s)" is located at the bottom right of the table.

Action	UUID	Status	Family	Host
Action ▾	a3ed674b-854a-43e5-91ae-caee4abb0690	UP	2ab525a0-61e1-4839-9a8b-406168db15dd	127.0.0.1
Action ▾	0f0e224d-e313-4449-ab18-4133fb7469ed	UP	2ab525a0-61e1-4839-9a8b-406168db15dd	127.0.0.1

+ Deploy Node(s)

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

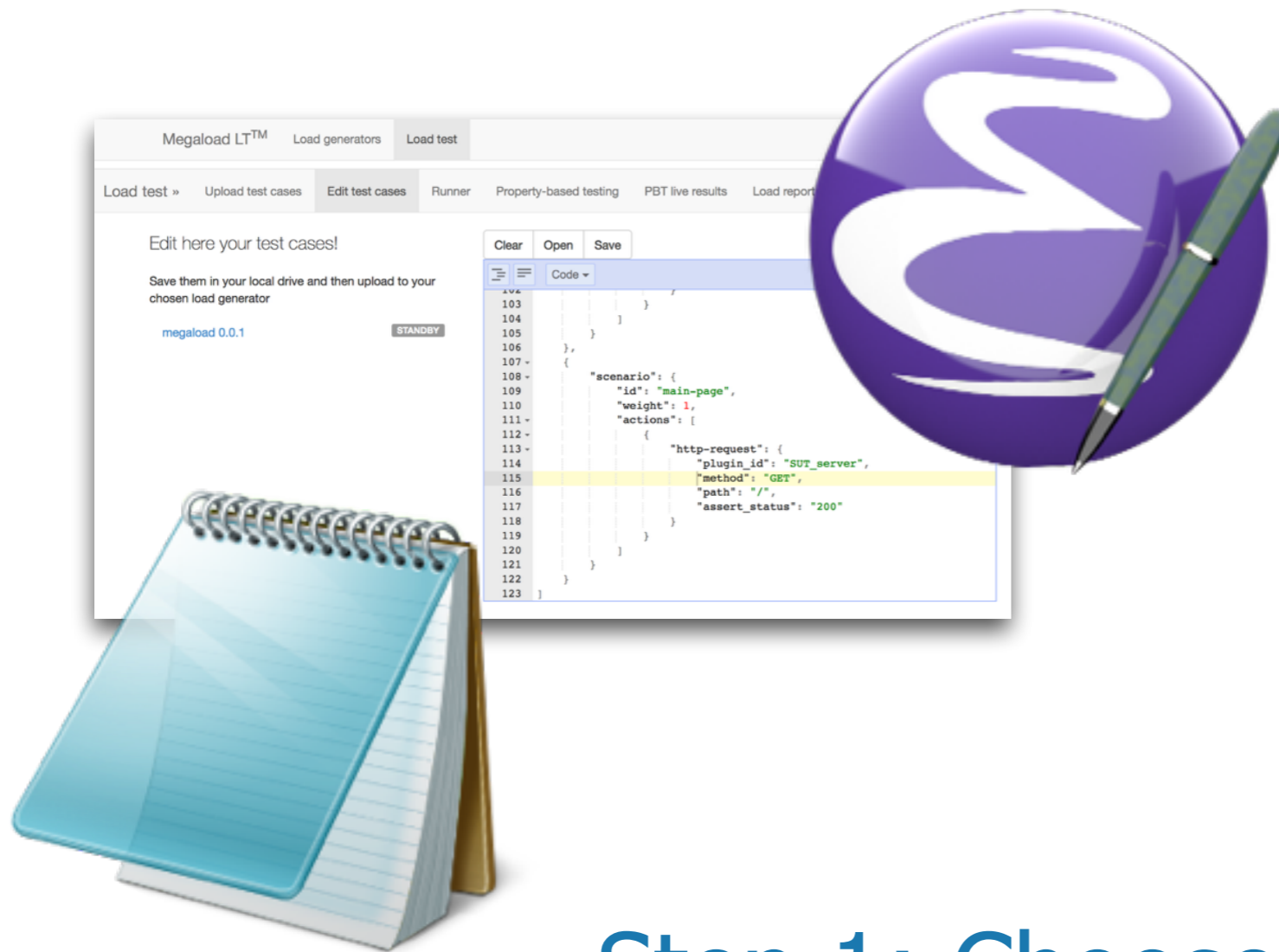
GET http://<IP>:5050/customer/bulma

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

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

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

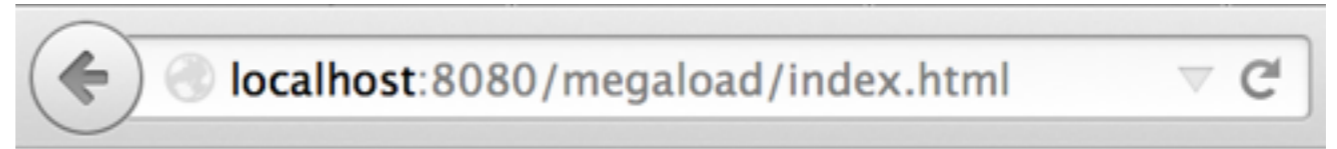
Let's write our test!



Step 1: Choose **your** text editor

Step 1: Choose your text editor

Embedded editor at



A screenshot of the Megaload LT™ web interface. The top navigation bar includes "Megaload LT™", "Load generators", and "Load test". Below this, a secondary navigation bar has "Load test »", "Upload test cases", "Edit test cases", "Runner", "Property-based testing", "PBT live results", and "Load report". The main content area is titled "Edit here your test cases!". It contains instructions: "Save them in your local drive and then upload to your chosen load generator". Below this, it shows "megaload 0.0.1" and a "STANDBY" button. On the right, there is an embedded text editor with a toolbar containing "Clear", "Open", and "Save" buttons. The editor's header shows "Code" and "powered by ace". The editor content shows a single line with the number "1" and a pair of curly braces "{}".

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

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

Your **phase** is ready!

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" : ... }  
]
```

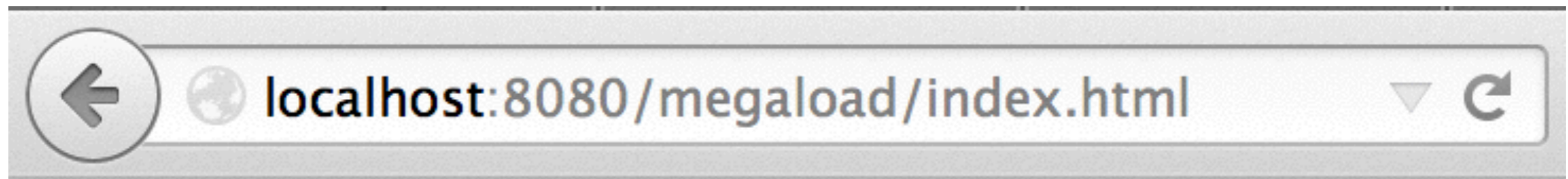
Step 7: Save your test

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

The screenshot shows the main page of Megaload LT™. At the top, there is a navigation bar with three items: 'Megaload LT™', 'Load generators', and 'Load test'. Below this, there is a secondary navigation bar with 'Load generators »' and 'View load engines info'. The main content area is split into two columns. The left column displays a list of load generators under the heading 'megaload 0.0.1'. Two entries are visible, each with a unique ID and IP address, and a green 'UP' status indicator. The right column contains a welcome message and instructions on how to use the interface.

Megaload LT™	Load generators	Load test
Load generators »	View load engines info	

megaload 0.0.1

- 445bd7ea-1dcf-4540-bfe4-7aabca791d7a@127.0.0.1 **UP**
- 85bcdd82-cc46-4d8d-97a8-0f79e16abcb6@127.0.0.1 **UP**

Welcome to Megaload!

You can choose between different functionalities using the top bars.
You can select a load generator by clicking on it in the left bar.

Main page

The screenshot shows the main page of Megaload LT. At the top, there are three navigation tabs: "Megaload LT™", "Load generators", and "Load test". Below these, there are two more tabs: "Load generators »" and "View load engines info". The main content area displays a cluster of nodes. A callout box labeled "Cluster" points to the "megaload 0.0.1" label. Another callout box labeled "Nodes" points to the list of two nodes, each with a green "UP" status indicator. To the right of the nodes, there is a welcome message and instructions on how to use the interface.

Megaload LT™ Load generators Load test

Load generators » View load engines info

megaload 0.0.1 Cluster

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

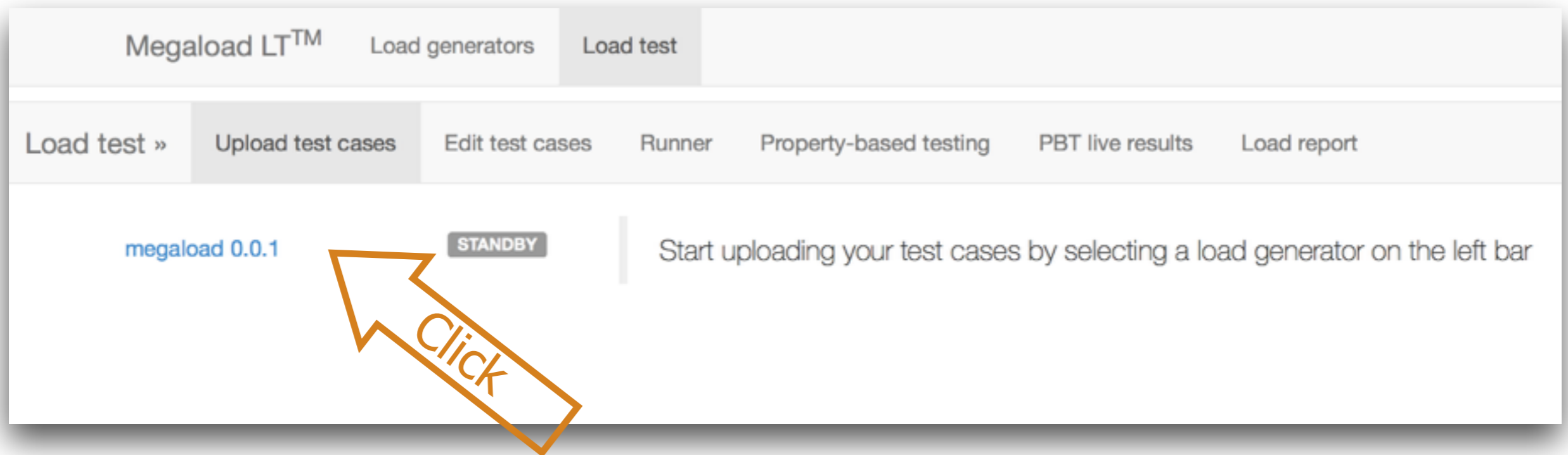
85bcdd82-cc46-4d8d-97a8-0f79e16abcb6@127.0.0.1 UP

Nodes

Welcome to Megaload!

You can choose between different functionalities using the top bars.
You can select a load generator by clicking on it in the left bar.

Upload test case



Always select your load generator in the left bar

Upload test case

The screenshot displays the Megaload LT™ web interface. At the top, there are navigation tabs: 'Megaload LT™', 'Load generators', and 'Load test'. Below these, a secondary navigation bar includes 'Load test »', 'Upload test cases' (which is highlighted), 'Edit test cases', 'Runner', 'Property-based testing', 'PBT live results', and 'Load report'. The main content area shows a card for 'megaload 0.0.1' with a 'STANDBY' status. To the right of this card are four sections for uploading different types of test data: 'Upload test specification', 'Upload additional data', 'Upload actions module', and 'Upload property'. Each of these sections contains a green '+ Add files...' button and a blue 'Start upload' button.

Upload test case

Your **test case** is uploaded!

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 **STANDBY**

Upload test specification

+ Add files... Start upload

SUT-test-example.json	1.07 KB	Start
-----------------------	---------	-------

Uploaded

Upload additional data

+ Add files... Start upload

Upload actions module

+ Add files... Start upload

Start load test

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 **STANDBY**

Test control

SUT-test Start! Stop

SUT-test

Start load test

Your test case is running!

The screenshot displays the Megaload LT™ web interface. At the top, there are navigation tabs: "Megaload LT™", "Load generators", and "Load test". Below this, a secondary navigation bar includes "Load test »", "Upload test cases", "Edit test cases", "Runner", "Property-based testing", and "PBT live res". The main content area shows "megaload 0.0.1" on the left, a "STANDBY" status indicator, and a "Test control" section. The "Test control" section features a dropdown menu with "SUT-test" selected, a green "Start!" button, and a red "Stop" button. On the right side, there are two notification boxes: a blue "Info" box stating "Test is starting..." and a green "Success" box stating "Test started, please refresh your browser."

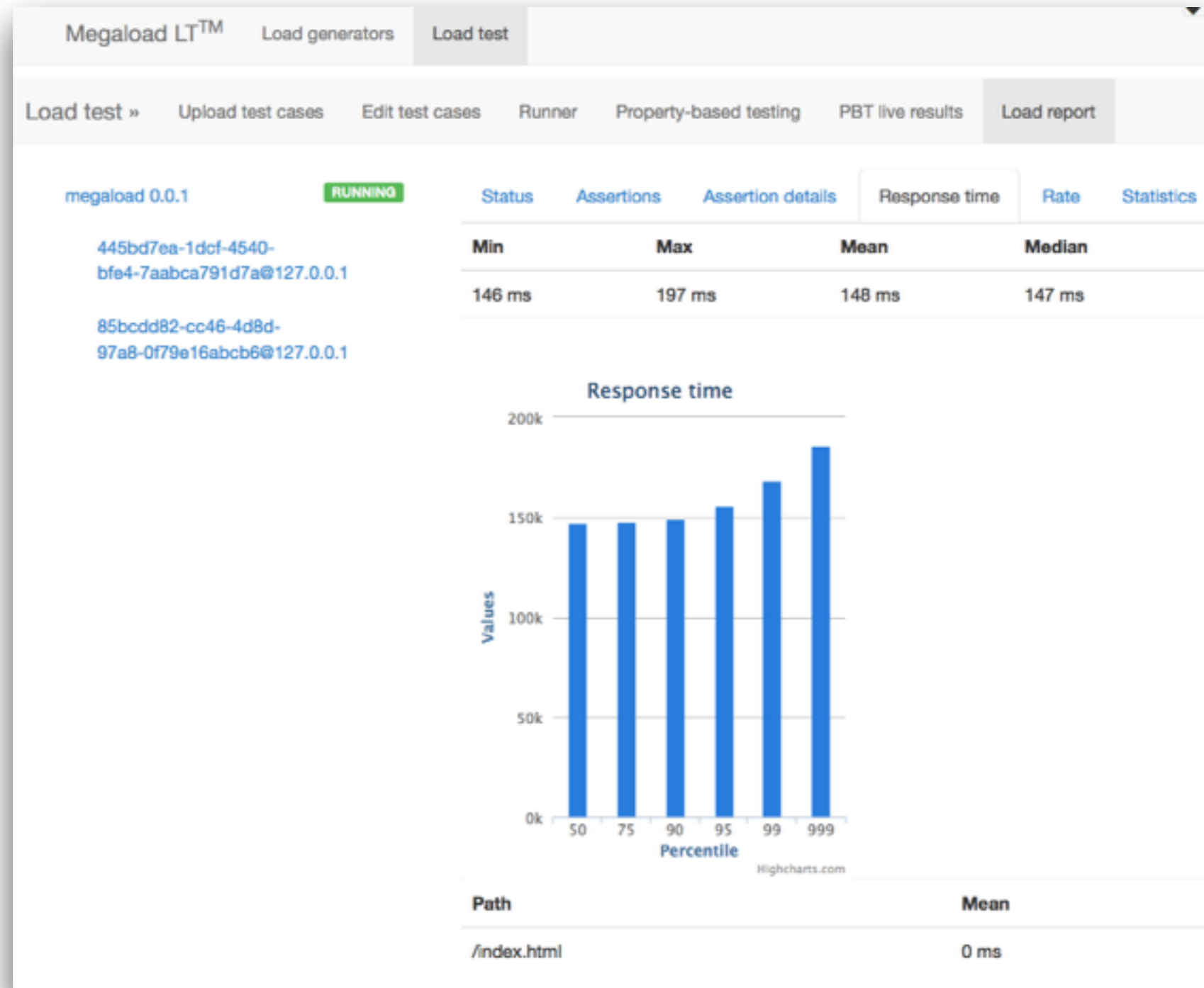
Load test results

Test state

The screenshot displays the Megaload LT™ interface. At the top, there are navigation tabs: 'Megaload LT™', 'Load generators', and 'Load test'. Below this is a secondary navigation bar with options: 'Load test »', 'Upload test cases', 'Edit test cases', 'Runner', 'Property-based testing', 'PBT live results', and 'Load report'. The main content area shows a test instance 'megaload 0.0.1' with a green 'RUNNING' status. Below the instance name are two UUIDs: '445bd7ea-1dcf-4540-bfe4-7aabca791d7a@127.0.0.1' and '85bcdd82-cc46-4d8d-97a8-0f79e16abcb6@127.0.0.1'. A 'Status' tab is selected, showing the message 'Test case SUT-test is RUNNING'. Other tabs include 'Assertions', 'Assertion details', 'Response time', 'Rate', and 'Statistics'.

Load test results

Response time

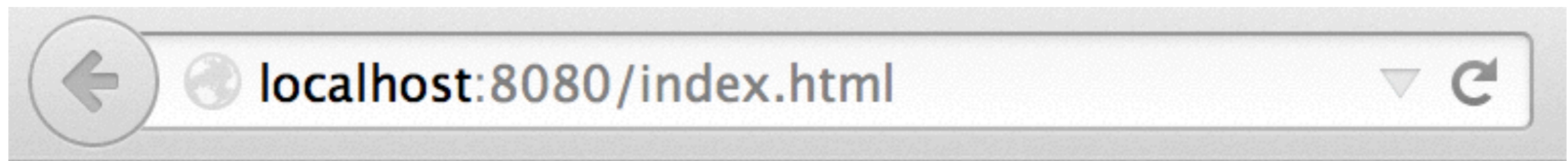


Load test results

Rate

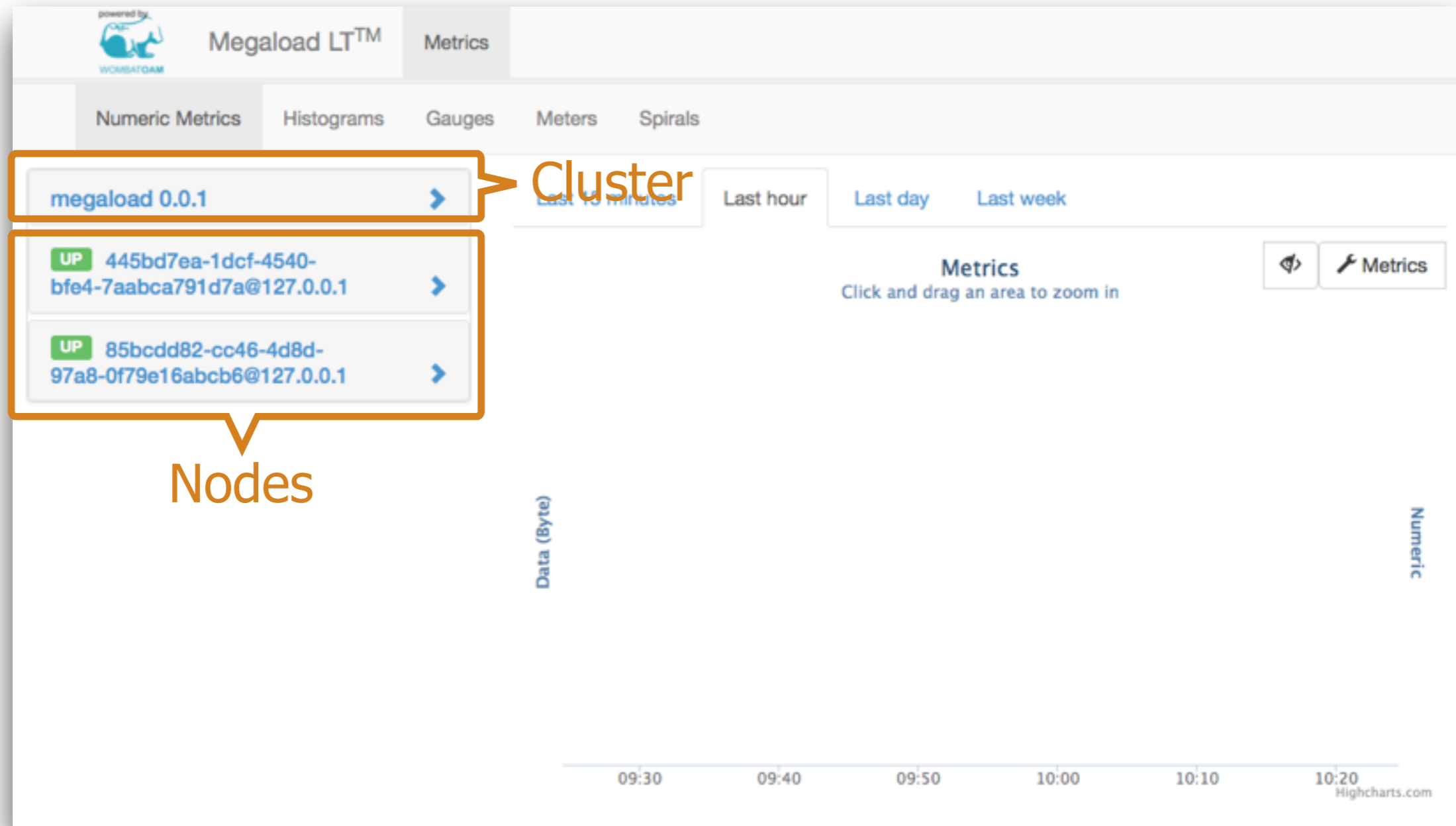


Do you want more metrics?



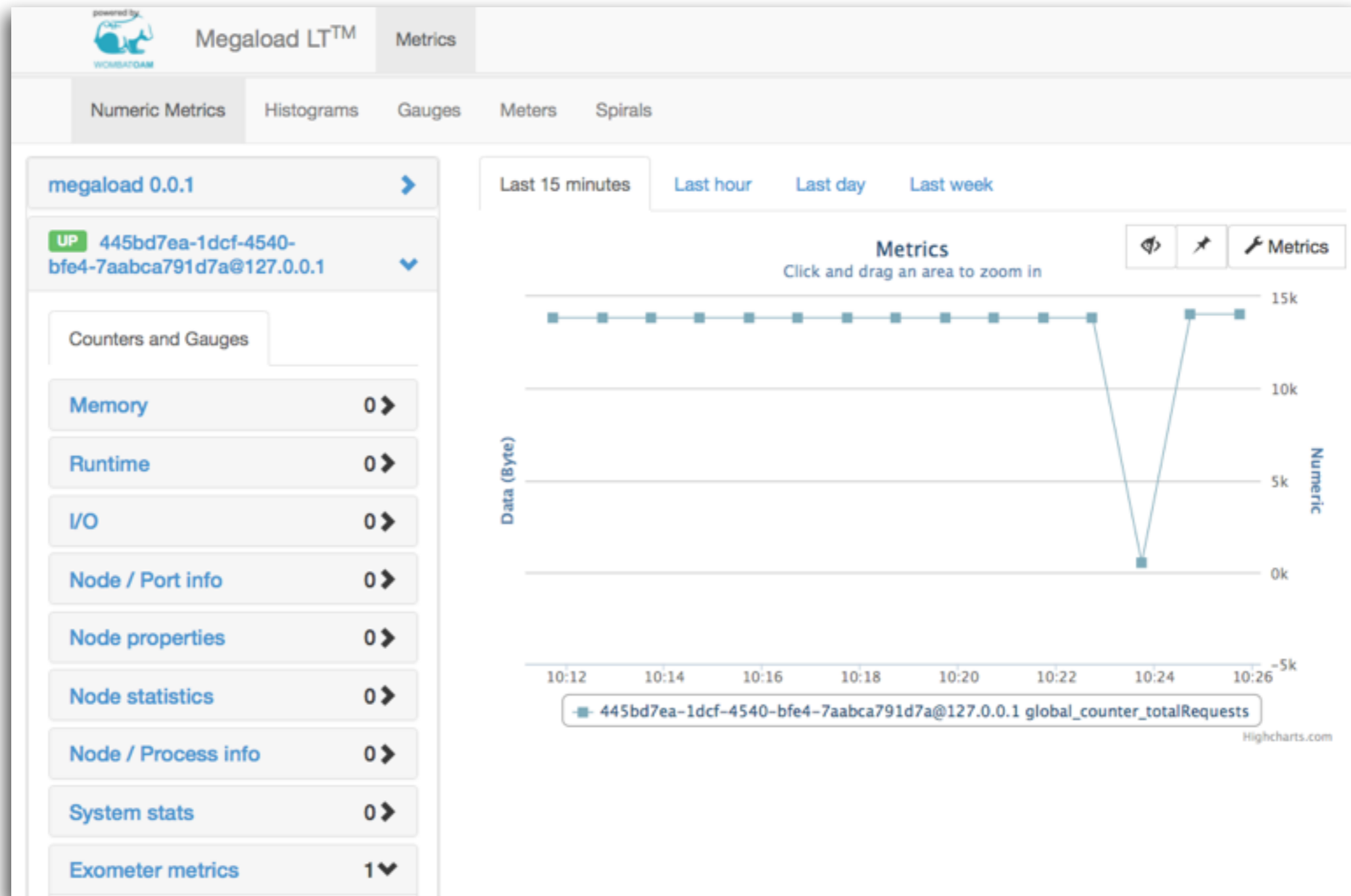
Metrics

Real-time metrics dashboard



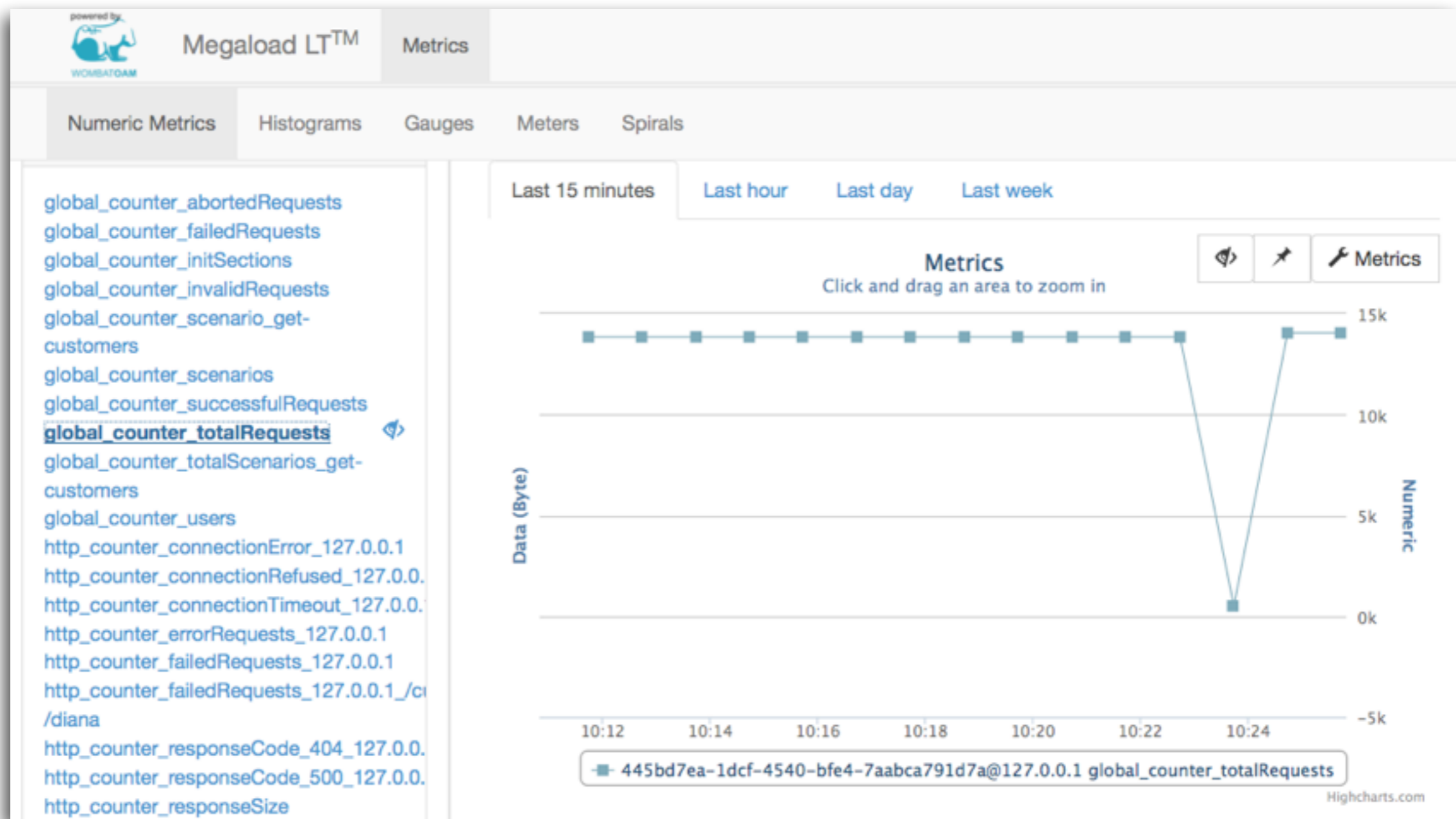
Metrics

Load test counters - node



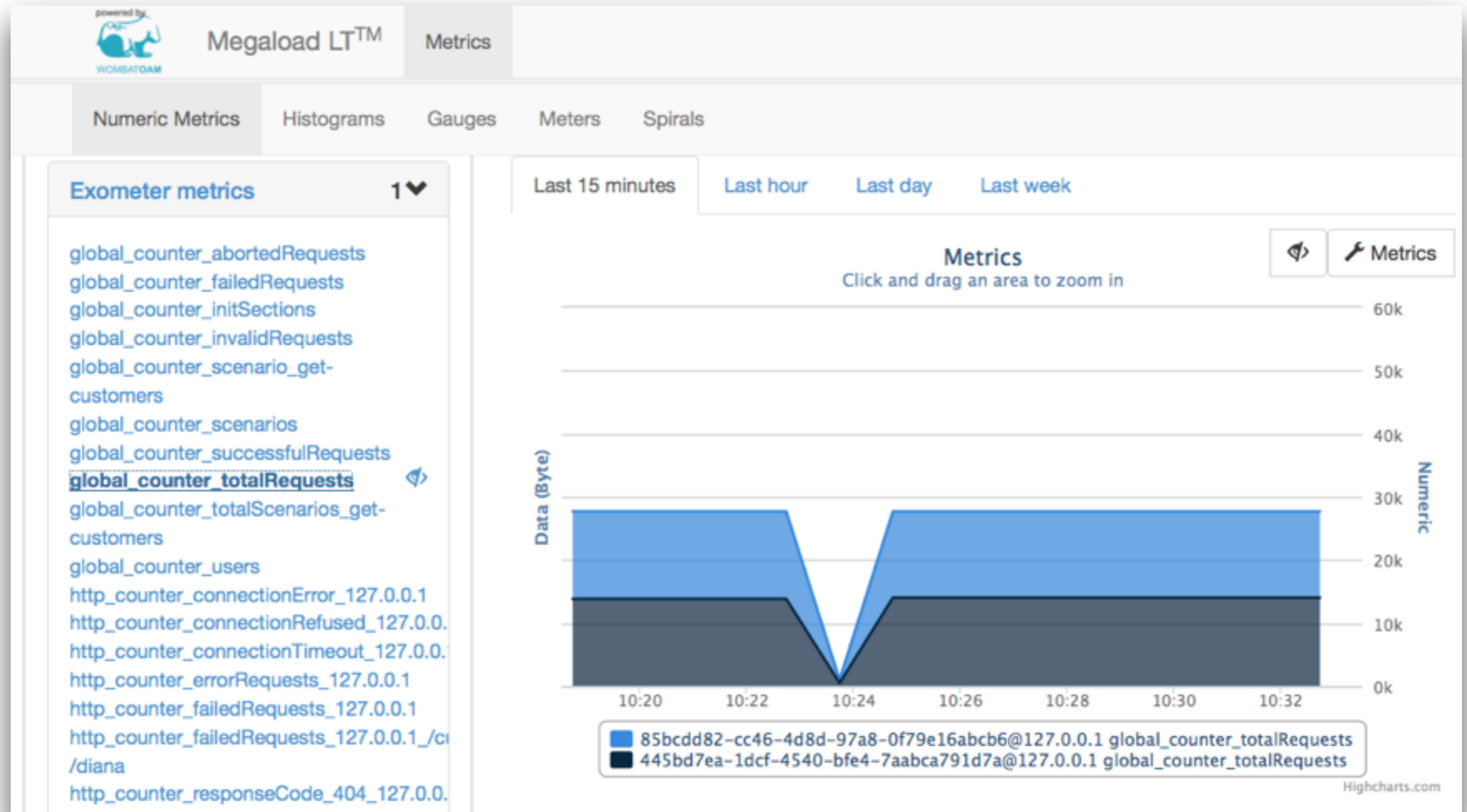
Metrics

Built-in counters



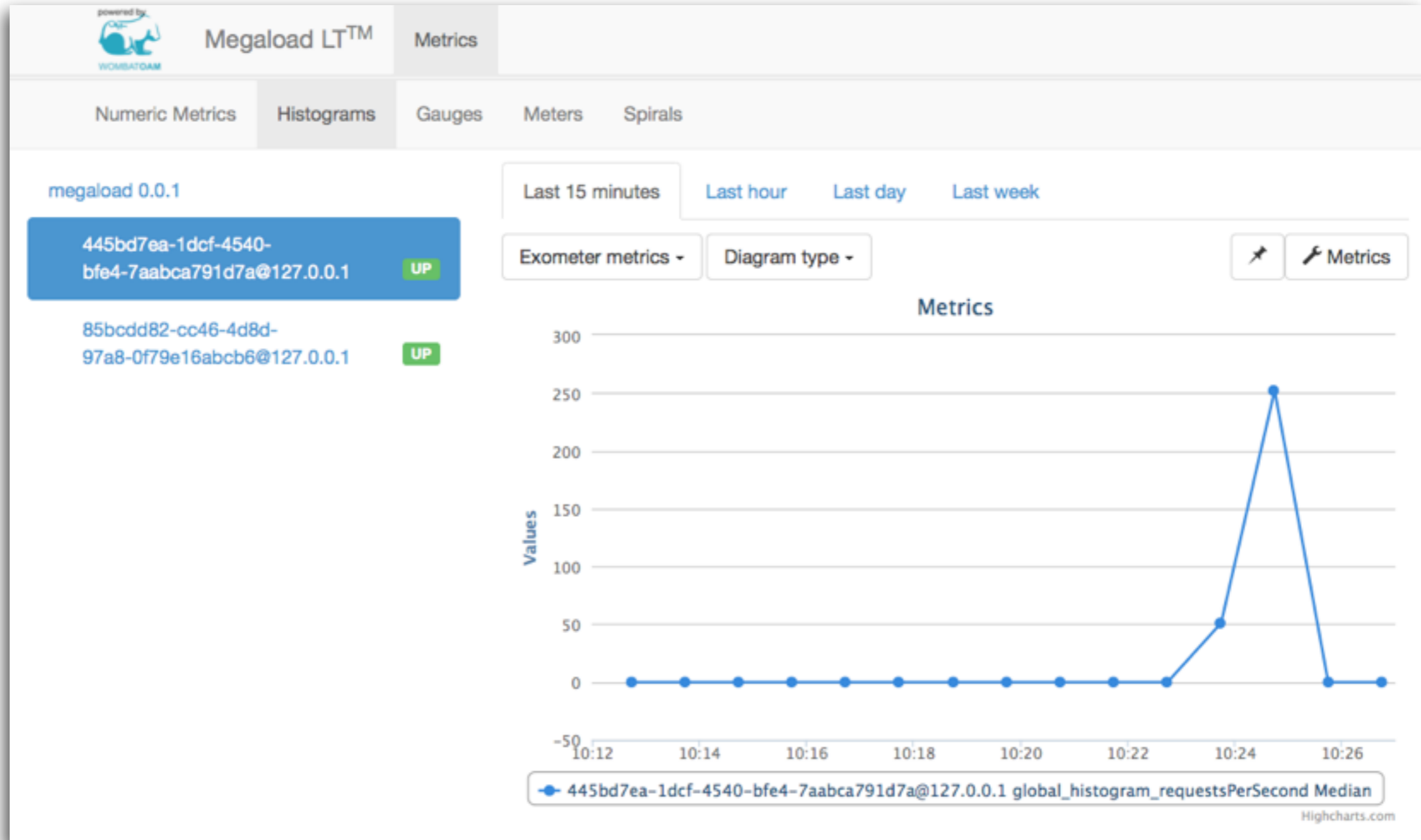
Metrics

Aggregated counters - cluster



Metrics

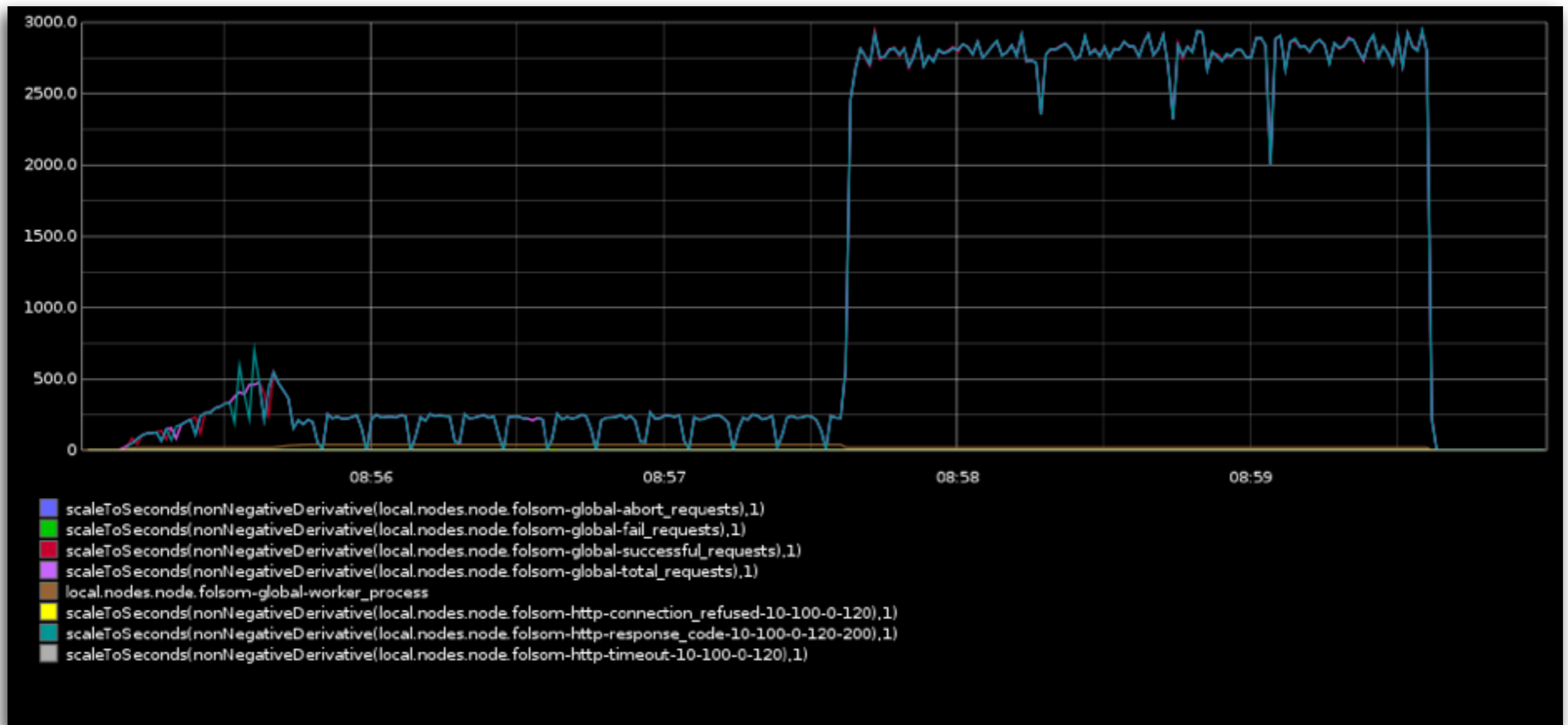
Histograms - node



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

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

Summary

Test case **SUT-test** has **Failed**

Overview

Test Identifier	Status
SUT-test	Failed
Phase Identifier	Status
1 SUT-low	Passed

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	<code>{"assert-histogram": {"id": "http_histogram_responseTime", "statistic": "mean", "filter": {"lt": 150000}}}</code>	165649
true	<code>{"assert-counter": {"id": "global_counter_failedRequests", "metric": "count", "filter": {"eq": 0}}}</code>	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"
  }
}
```

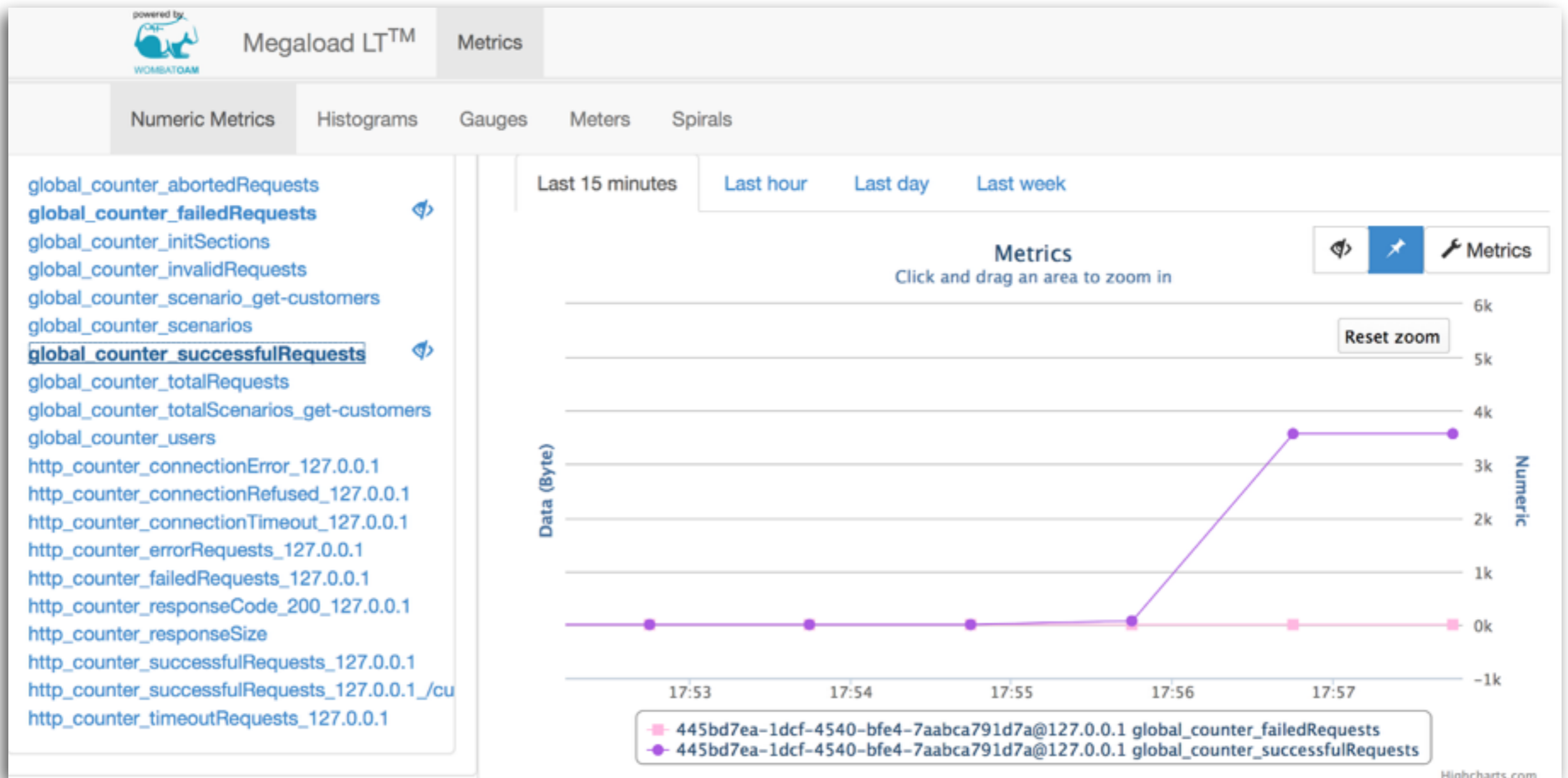
Request assertions

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

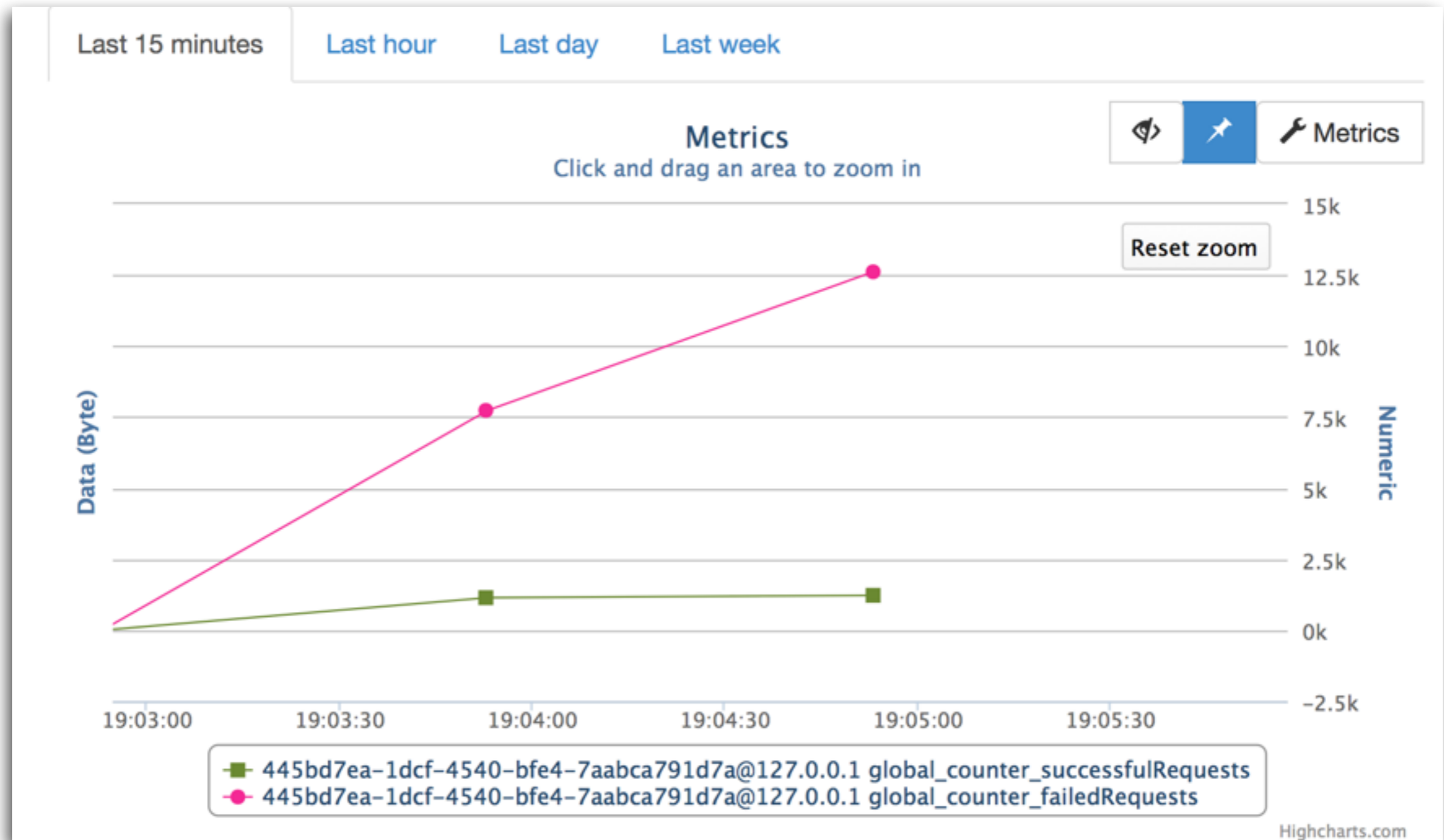
Now we have failing requests

The screenshot shows the Megaload LT™ interface. The top navigation bar includes 'Megaload LT™', 'Load generators', and 'Load test'. Below this, a secondary navigation bar contains 'Load test »', 'Upload test cases', 'Edit test cases', 'Runner', 'Property-based testing', 'PBT live results', and 'Load report'. The main content area displays a test case named 'megaload 0.0.1' with a 'FINISHED' status. Two test instances are listed with their respective IDs and target IP addresses. The 'Assertion details' tab is active, showing a table for 'SUT-test' with columns for 'Status', 'Assertion', and 'Value'. The table contains two rows: one with a 'true' status and a value of 677, and another with a 'false' status and a value of 12427. The value 12427 is circled in orange, indicating a failed assertion.

Status	Assertion	Value
true	{"assert-histogram": {"id": "http_histogram_responseTime", "statistic": "mean", "filter": {"lt": 150000}}}	677
false	{"assert-counter": {"id": "global_counter_failedRequests", "metric": "count", "filter": {"eq": 0}}}	12427

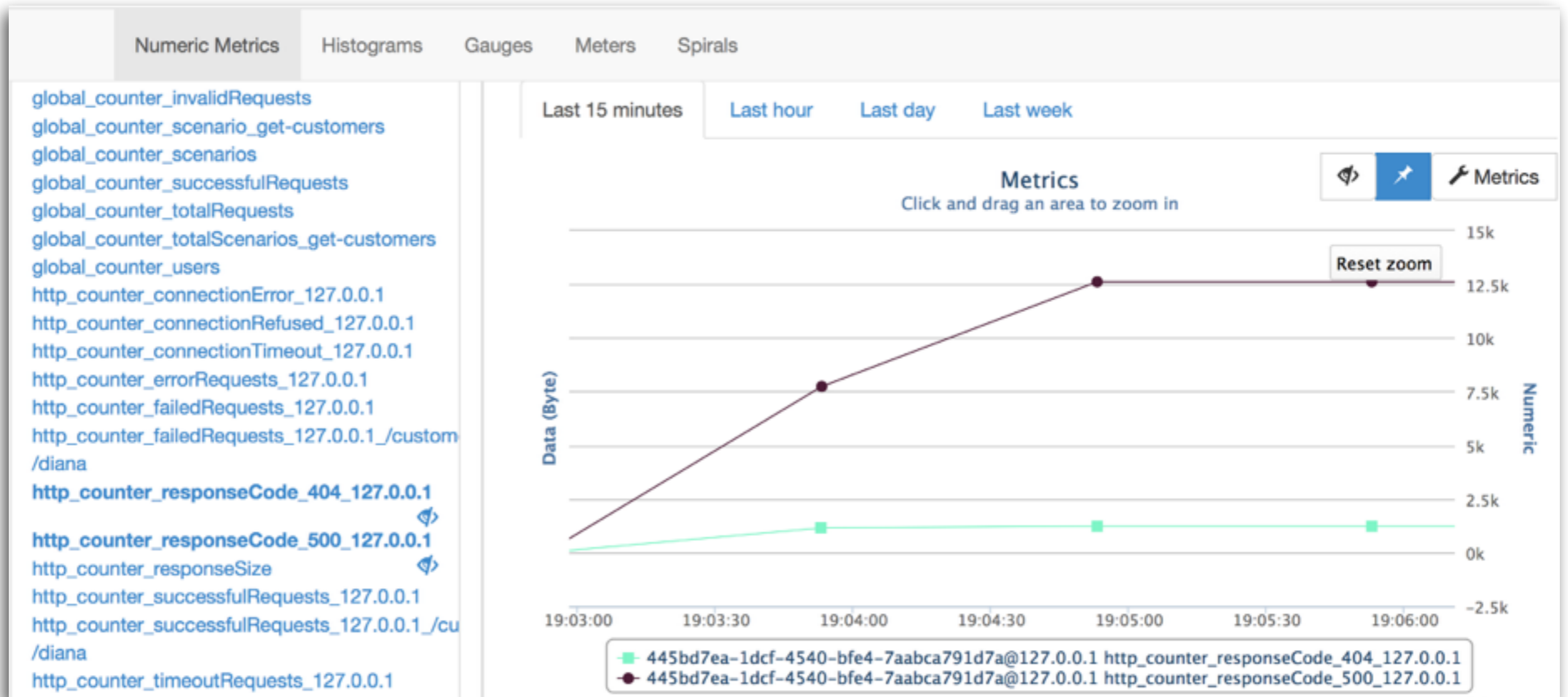
Request assertions

Not all of them failed



Request assertions

We got 505 response codes!



Questions?

Thanks for coming!