

1. Documentation	2
1.1 StrongOps	2
1.1.1 Setting up StrongOps monitoring	3
1.1.2 Using the StrongOps console	5
1.1.2.1 Using the metrics dashboard	7
1.1.2.2 Profiling	13
1.1.2.3 Error reporting	15
1.1.2.4 Configuring alerts	15
1.1.2.5 Controlling an application cluster	16
1.1.3 Using Strong Supervisor	18
1.1.4 Understanding strong agent	19
1.1.5 Open data API	20

Documentation

StrongLoop Suite documentation

StrongLoop Suite includes:

- [LoopBack](#), an open-source mobile backend framework for Node.js.
- [StrongNode](#), professional support for Node.js, plus cluster management and other powerful modules.
- [StrongOps](#), a built-in monitoring and management console.

To get up and running quickly, see [Getting started](#).

What's new

LoopBack now enables you to send push notifications to mobile apps. See [Creating push notifications](#) for details. Client SDKs have been updated to support push notifications:

- [Android SDK](#) (version 1.2)
- [iOS SDK](#) (version 1.2)

StrongLoop now supports the [Digital Ocean](#) cloud platform.

See [What's new](#) for a complete list.

StrongOps

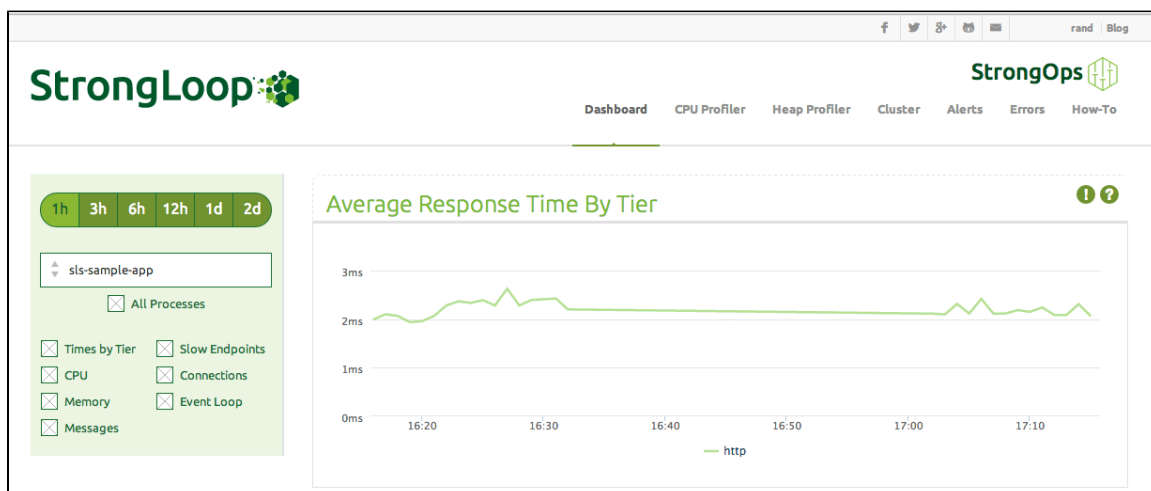
StrongOps is a comprehensive application performance management platform that provides key performance metrics for Node.js applications. StrongOps helps you visualize and optimize Node application performance. It bridges applications from development to production through configuration and deployment capabilities.



Follow the steps in [Getting started](#) to create an application you can monitor with StrongOps. If you already have an application, then see [Setting up StrongOps monitoring](#).

StrongOps console

For instructions on logging in to the StrongOps console, see [Using the StrongOps console](#).



Setting up StrongOps monitoring

- [Prerequisites](#)
- [Setting up an application for monitoring](#)
 - [Providing an API key](#)
- [Instructions for older versions of slc](#)
 - [Adding strong-agent as a dependency](#)
 - [Calling profile\(\)](#)
 - [Circumventing firewall restrictions](#)

Prerequisites

Follow the instructions in [Getting started](#) to:

- Install Node.
- Install the `slc` command-line tool (optional, but strongly recommended).
- Register a user account with StrongOps and [strongloop.com](#) with the *same* username and password.
- Create an application that you want to monitor with StrongOps. It can be any Node application.

The remainder of this article assumes you have performed the above steps.

Setting up an application for monitoring



If you created your application with `slc` command-line tool (the `slc create`, `slc lb project`, or `slc example` commands), then you only need to follow the instructions below enable StrongOps monitoring.

Follow these steps to set up an application for StrongOps monitoring:

1. **Provide a StrongOps API key.** See [Providing an API key](#).
2. Run your application as follows:

```
$ sl-run
```



The `sl-run` command is provided with version 2.2.2 of `strong-cli`. For instructions on getting the latest version, see [Updating to the latest version](#).

Providing an API key

To be monitored, an application must provide a StrongOps API key.

Recommended way

Use the following command to generate an API key and save it to `strongloop.json` in the current directory:

```
$ slc strongops
```

This command prompts for your email address and password then saves the API key to `strongloop.json`. If you need to call this command from a script, there are additional options; see [slc strongops](#) for details.

Alternative ways to provide API key

Alternatively, you can provide default values in configuration files or environment variables. See [Understanding strong agent](#) for details.

You can also provide the API key as an argument to the `profile()` function. See [Calling profile\(\)](#) for more information.

Instructions for older versions of slc

If you have an old version of `slc`, you can enable StrongOps monitoring as follows:

1. **Provide a StrongOps API key.** See [Providing an API key](#) above.
2. **Add strong-agent as a dependency** in your application's `package.json` file. See [Adding strong-agent as a dependency](#).

3. Call `require('strong-agent').profile()` in the application code, at the beginning of your module. See [Calling profile\(\)](#).

Adding strong-agent as a dependency



This step is not required for applications created with the `slc create`, `slc lb project`, or `slc example` commands. Such applications' `package.json` will already list the dependency.

You must ensure that the application's `package.json` file declares `strong-agent` as a dependency. The easiest way to do this is to use the following command in the application's main directory:

```
$ npm install --save strong-agent
```

This command adds `strong-agent` as a dependency to the application's `package.json` file.

You can also edit your `package.json` file and add the dependency manually, but this is not recommended.

Calling profile()



Applications created with an `slc` command (`slc create`, `slc lb project`, or `slc example`), will automatically contain this function call, using the API key in a configuration file.

To enable StrongOps monitoring, call the `strong-agent profile()` function before any code that needs profiling.

If you created the application manually and provided an API key as described above, you must add the following call at the beginning of your module:

```
require('strong-agent').profile();
```



The call to `profile()` must come at the top of your module before any other `require()` calls.

Calling profile() with an API key

If you didn't use `slc strongops` to save your API key for the application, you can provide the key in the code like this:

```
var APPLICATION_NAME = 'My Great App';
var API_KEY = '<your API key>';
require('strong-agent').profile(API_KEY, APPLICATION_NAME);
```

Circumventing firewall restrictions

By default, StrongOps uses port 4567. This may be a problem if you want to use StrongOps from within a firewall that restricts that port number.

To configure the port StrongOps uses, replace this call:


```
require('strong-agent').profile();
```

with the following:

```
require('strong-agent').profile( undefined, undefined, { port: 8080 } );
```

This will enable you to connect to StrongOps from inside your corporate firewall if port 8080 is allowed on your network.

StrongOps will get the API key and application name from environment variables (if defined) or the application's configuration files, as described in [Understanding strong agent](#).

 StrongOps can use only port numbers 4567 and 8080.

Using the StrongOps console

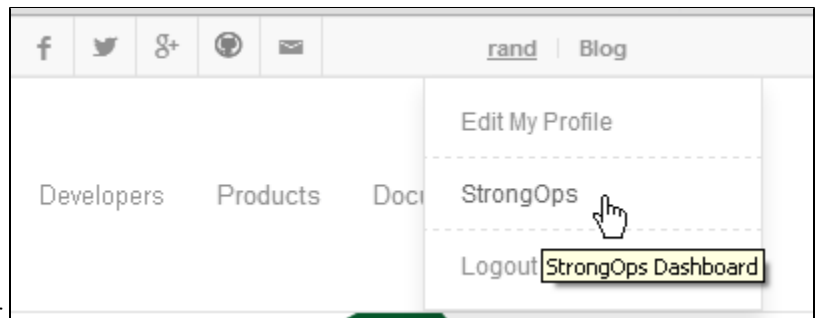
- [Logging in](#)
- [Application selectors and filtering](#)
- [Viewing the dashboard and profiler](#)
 - [Selecting metrics to view on the dashboard](#)
 - [Time frame](#)
 - [Running profilers](#)
- [Account information](#)

Logging in

Log in with your username and password at <https://strongloop.com/login>. The StrongOps console is then available at <http://strongloop.com/ops>.

Or, to get to the StrongOps console:

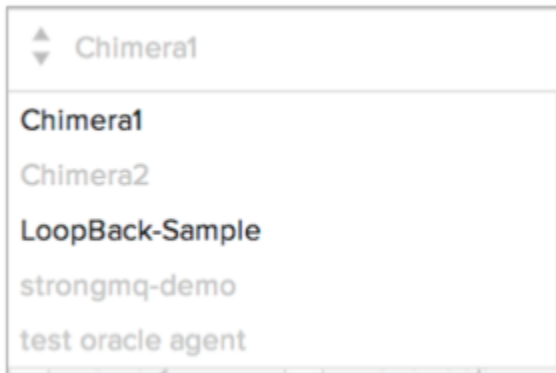
1. Hover your mouse pointer over your username at the top right of the window.
2. In the menu that appears, choose **StrongOps**.



Application selectors and filtering

While viewing performance dashboards and profiling metrics, StrongOps provides the ability to select all hosts and processes which are part of an node application or filter based on individual host and/or process. Filtering for individual host is enabled by selecting a particular hostname from a drop down menu on the left of the operations console. Similarly filtering for a individual process is enabled by selecting a ProcessIdentifier (PID) of interest from the dropdown of process identifiers listed. Both hosts and processes are automatically detected by the StrongOps agent.

All processes with the same application name are treated as part of a cluster. The StrongOps dashboard shows aggregate data for those processes and enables you to display information for individual process IDs.



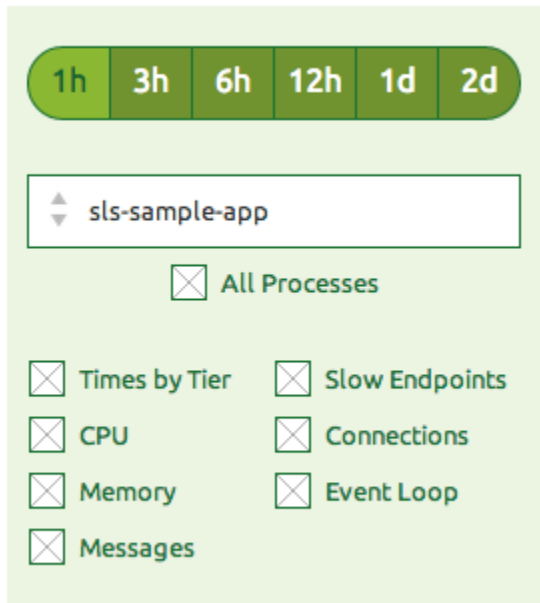
Viewing the dashboard and profiler

The StrongOps operational console has two views:

- **Dashboard:** provides graphs of key performance metrics like cross-tier application response time, slow endpoints (web services, HTTP requests, SQL and NoSQL databases, Memcached, Redis, and so on), CPU use, heap use, connection concurrency, event loop and messages monitoring.
- **Profiler:** displays CPU and memory profiles of an application under load and view related analytics in the profiler view.

Selecting metrics to view on the dashboard

On the left side panel of the dashboard, there are a set of check boxes. These check boxes indicate which metrics have been selected for display on the dashboard. You can select or un-select the metrics based on interest area.



Once you select a metric to view on the checkbox selection on the left panel, the selected metrics are displayed in the selected or default timeframe on the right side dashboards.

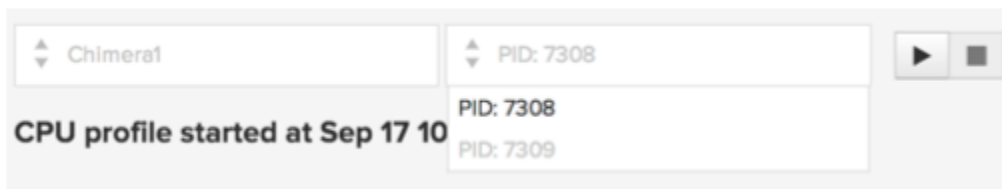
See [Using the metrics dashboard](#) for more information.

Time frame

You can select data periods for viewing collected metrics from the menu above the Application selection dropdowns. The dashboards metrics then display data for the selected timeframe.

Running profilers

Once you navigate to the Profiler page by selecting the Profiler link from the top menu you can run either the CPU or the memory profiler. First, the hostname and process ID have to be selected from the available drop-down and the small "run" icon shaped like an horizontal arrow needs to be pressed. Usually 5 second duration profiles are advisable to take in systems under production load. After the time of profiling, click the "stop" button shaped like a square to collect the display of the CPU or memory profile.



Some of the the metric graphs and profiler views have further fine-grained visual help components which let us highlight relevant metrics, detect patterns or visualize in aggregations.

See [Profiling](#) for more information on using profilers.

Account information

Hover your mouse over your user name shown the top right of the page and click **Account** to display the **StrongOps Account** page, where you can change your password and find your API key, as shown below.

StrongOps Account

Password

This password is for your StrongOps command via StrongLoop Suite's slc command line tool. This will **not** change your main strongloop.com password

Update

API KEY

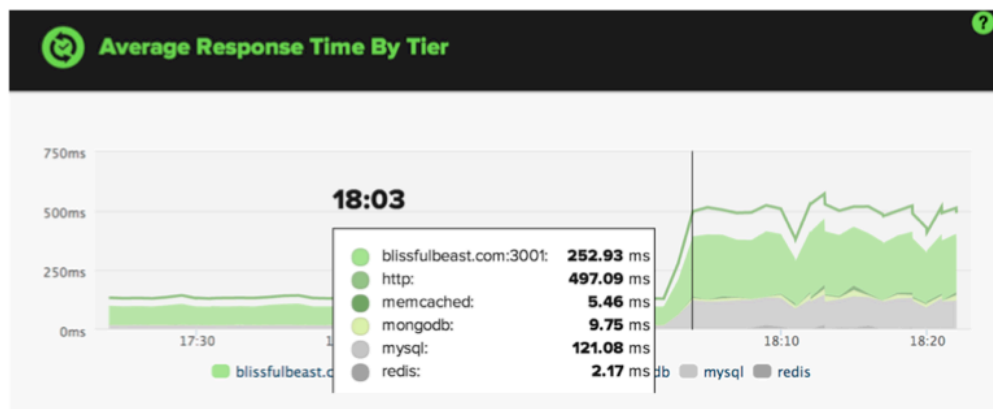
1c16889f53658141a3ac7fe6aea843f0

Using the metrics dashboard

- Cross-tier application response time
- Slow endpoints and databases
 - HTTP requests
- Application tracing
- Heap use
- CPU use
- Concurrency
- Event loop
- Messages

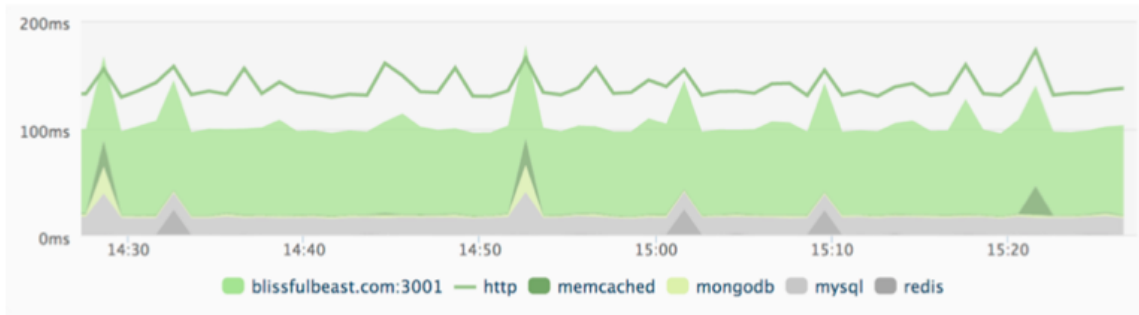
Cross-tier application response time

Cross-tier application response time is the first graph that is displayed on the StrongOps dashboard view. As transactions flow through the various front-end, middleware and back-end tiers of a Node.js application, StrongOps provides visibility into the time spent on each tier. In case of an incident, the cross tier application response time dashboard quickly identifies the application component that is causing the bottleneck.

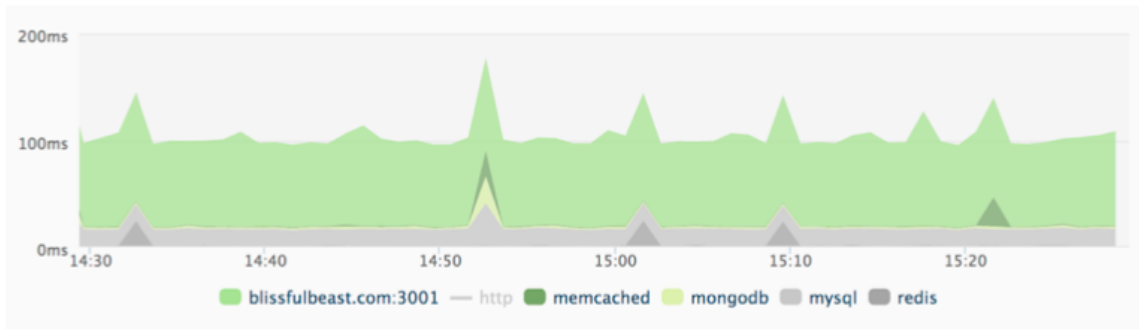


This metric is displayed by selecting the "Times by Tier" check box on the metric selection menu on the left panel of the dashboard

Metrics on this graph can be selected or unselected by clicking the tiers and components listed below the graph



Graph with "HTTP" metric unselected The HTTP metrics in the graph is removed on un-selection.



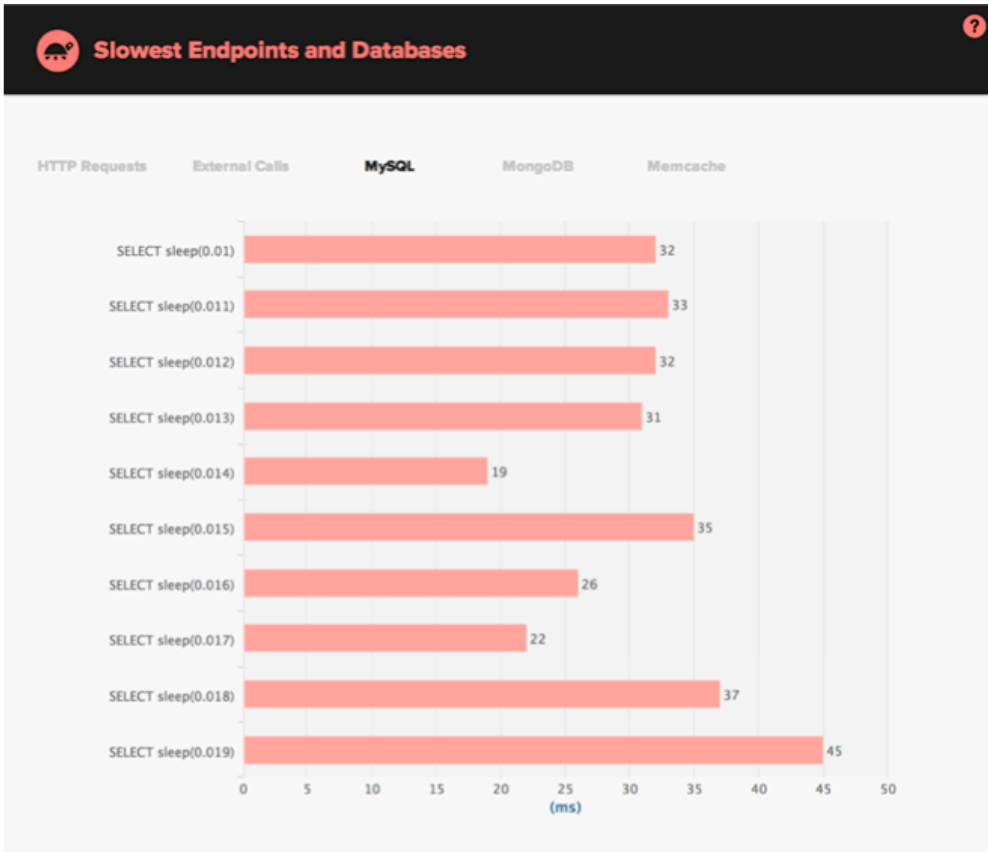
Slow endpoints and databases

During an incident, the single most common cause of failure and performance bottlenecks are backend systems. StrongOps provides visibility into key slow endpoints like incoming frontend requests, the main logic tier and the backends the node application interacts with. This metric is displayed by selecting the "Slow Endpoints" check box on the metric selection menu on the left panel of the dashboard. Key backend transactions and endpoints monitored by StrongOps are displayed and can be selected from the menu bar on top of the graph, as shown here:



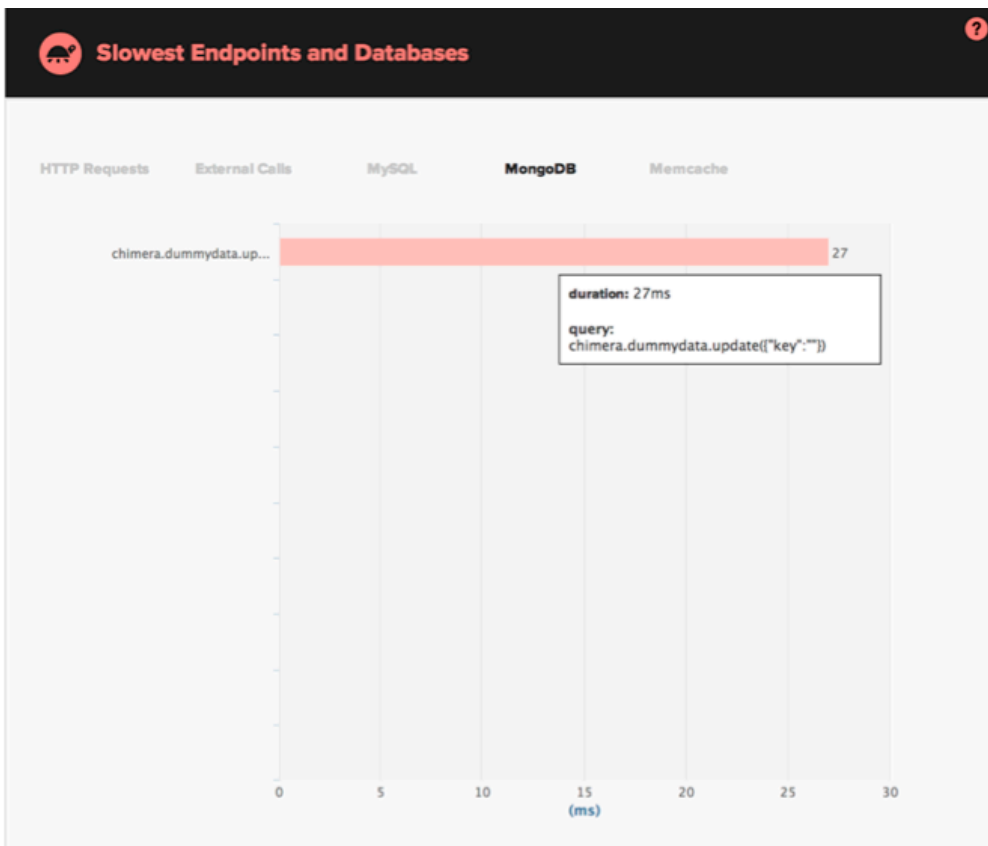
Each item in the menu enables you to display a category of slow endpoints in the app.

SQL calls to relational databases like MySQL and Oracle database server.



Hovering the mouse over the bars in the bar charts gives additional display for the SQL statements.

Calls to NoSQL databases like MongoDB



Hovering the mouse over the bars in the bar charts gives additional dispalys for the outbound calls.

Web services calls over SOAP, HTTP and REST protocols



Hovering the mouse over the bars in the bar charts gives additional dispalys for the outbound calls.

Additionally, Memcached and Redis calls made by the application as well as HTTP GET/POST requests to the application are aslo monitored

HTTP requests

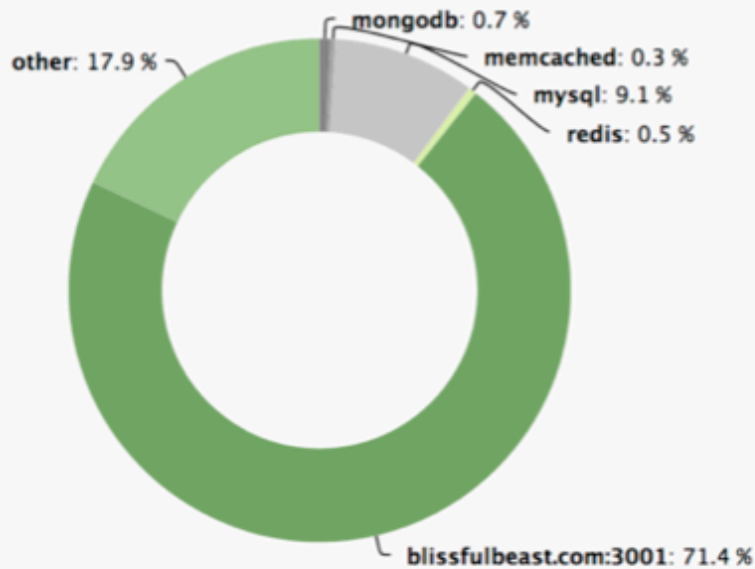
HTTP requests such as GET and POST are further aligned into a time distribution graph with related called components based on the time spent in each component and can be visually represented by a breakdown by "%" and transaction path.

To investigate these breakdown views, hover the mouse over a detected HTTP request type. This hovering brings up a selection box with 2 options, one for % breakdown and the other for path tracing.



Breakdown of Average Response Time by %

This breakdown is the result of selection of the % breakdown of average response time of the application transactions across multiple application components. These transactions are executed by end users in form of HTTP Requests and get rendered after being processed by the application's logic and backend components.

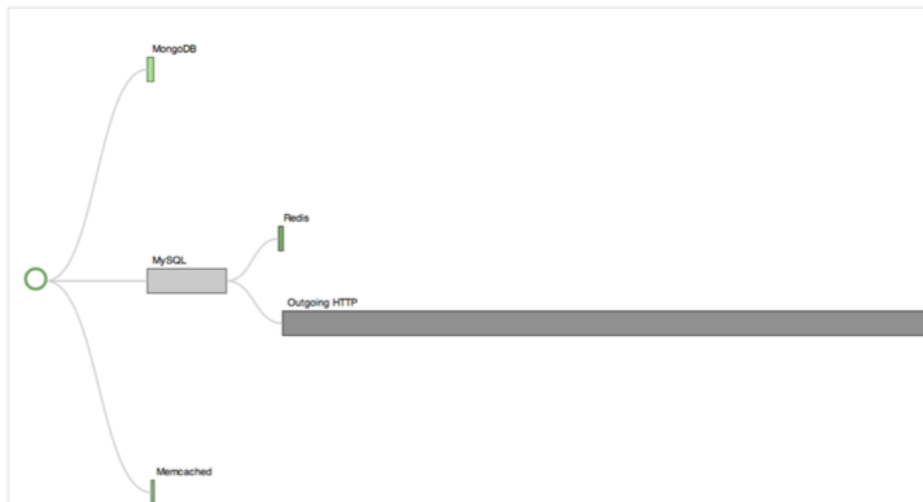


Application tracing

Application tracing is the feature in StrongOps which displays the path, sequence and time end user transactions spend on the application components and endpoints. This view is launched by selecting the path tracing option of the HTTP requests.

Application tracing provides visibility into the type of incoming http request calls, Call path, backend systems being accessed, web-service calls and outbound http responses the application makes to complete delivery of transactions. Time spent on each layer is also stitched together to provide an end to end view.

Endpoint Paths - /post



Probe: (Hover over chart for details)

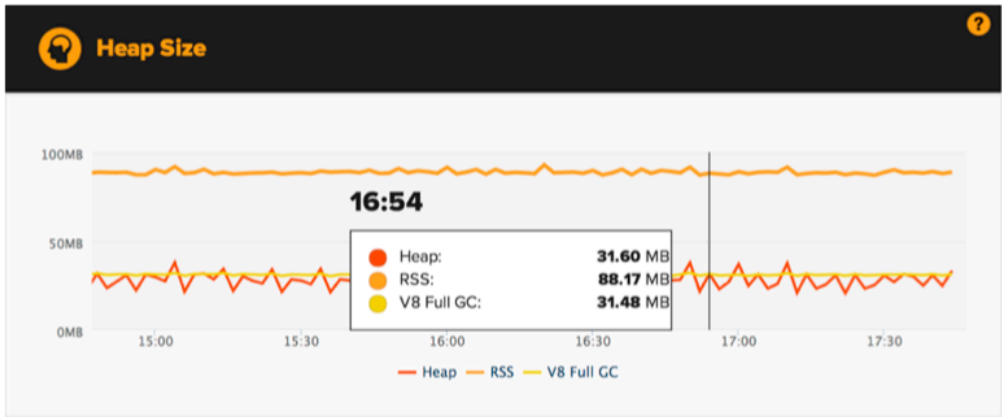
Duration:

Query (if available):

This tracing is provided at the component level and aggregate transaction levels. We are actively working on deep-dive single transaction tracing across javascript, event and native levels for individual transaction instances.

Heap use

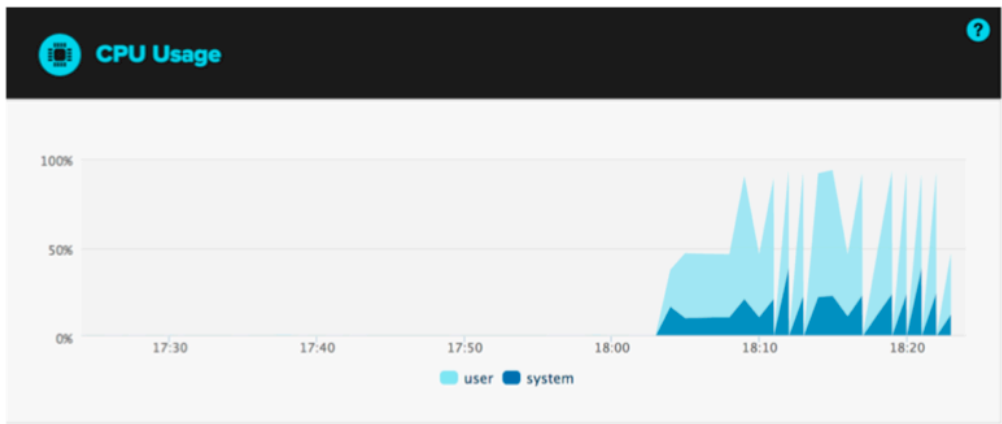
Node applications run on memory and run inside the chrome v8 engine. Efficient memory management and garbage collection are key to high performing applications. StrongOps provides visibility into to total allocated memory heap, the memory currently in use by the application and the frequency of garbage collection. These metrics help us identify memory management bottlenecks and possible memory leaks.



This metric is displayed by selecting the "Memory" check box on the metric selection menu on the left panel of the dashboard. Key metrics plotted are heap size, RSS, V8 full GC and frequencies

CPU use

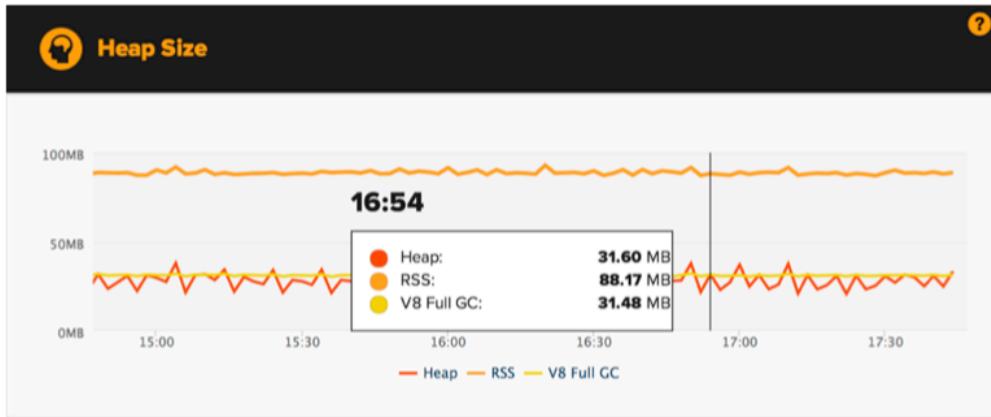
Node applications are single-threaded and make maximum use of the CPU cores available to them. High CPU usage can result in lesser CPU cycles being available to the running application, thereby slowing it down. It is hence very important to track the CPU usage on multiple cores that the application is deployed on.



This metric is displayed by selecting the "CPU" check box on the metric selection menu on the left panel of the dashboard

Concurrency

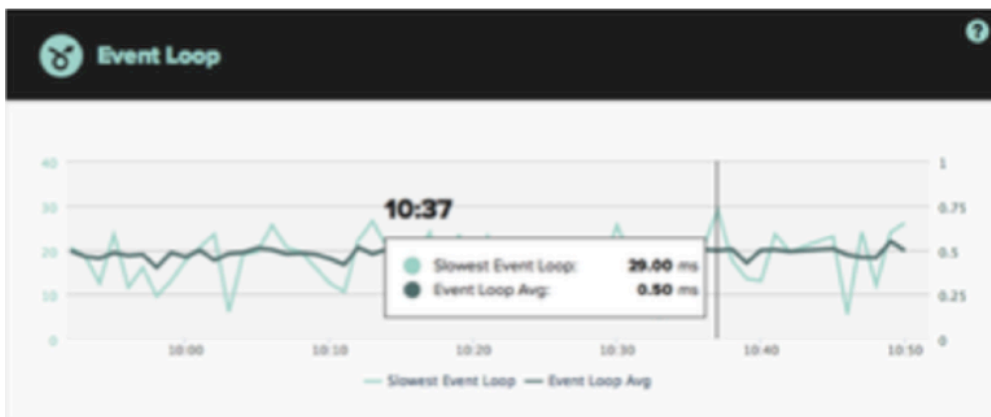
Concurrency is a key indicator of the incoming queue of requests in-flight or yet to be processed by the application. In case of bottlenecks in the application, requests start to backup resulting in higher than normal concurrency on the application.



This metric is displayed by selecting the "Connections" check box on the metric selection menu on the left panel of the dashboard

Event loop

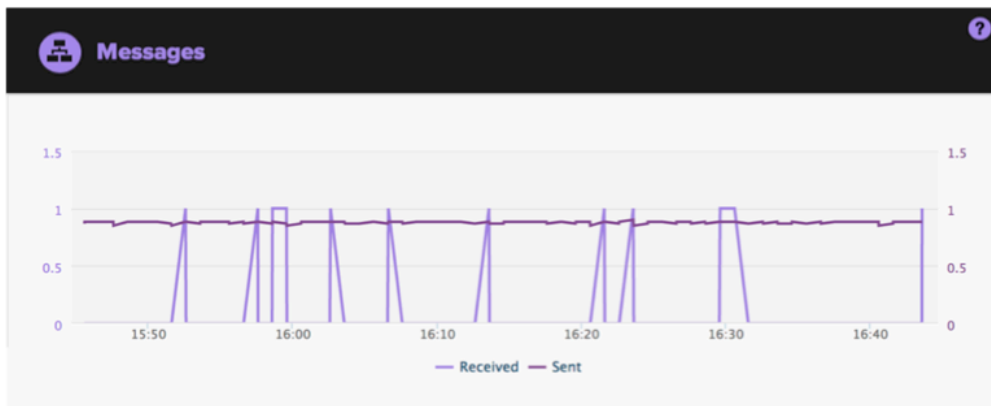
Node applications are event driven on the Server side. Event loops are usually extremely fast. However even the smallest amount of latency or degradation in the event loop response time can cause the transaction response times to jump abnormally. StrongOps provides visibility into event loops that are used by the application on the server.



This metric is displayed by selecting the "Event Loop" check box on the metric selection menu on the left panel of the dashboard

Messages

Node applications can interact with one or many messaging systems on the backend to put or get messages. Generally the message functions from the client applications side are very fast. However even the smallest amount of latency or degradation in the message handling response time can cause the transaction response times to jump abnormally. StrongOps provides visibility into messaging services i.e the messages sent and received by the Node application. It currently supports only messages handled through the strong-mq modules of the strongloop distribution.



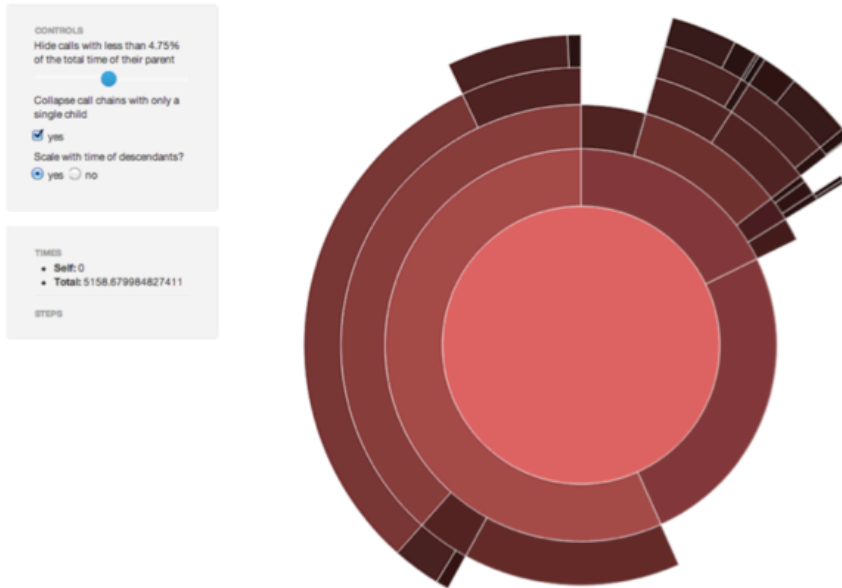
This metric is displayed by selecting the "Messages" check box on the metric selection menu on the left panel of the dashboard

Profiling

- CPU profiler
- Memory profiler

CPU profiler

CPU profiling is a feature of StrongOps, whereby you can collect the CPU footprints, modules, functions and lines of code from the application. A profiling session for a host and process can be kicked off from the StrongOps console and we recommend running it for about 5 seconds to get a comprehensive snapshot of the CPU footprint under production load.



Once the profile is collected, the following analytics are provided:

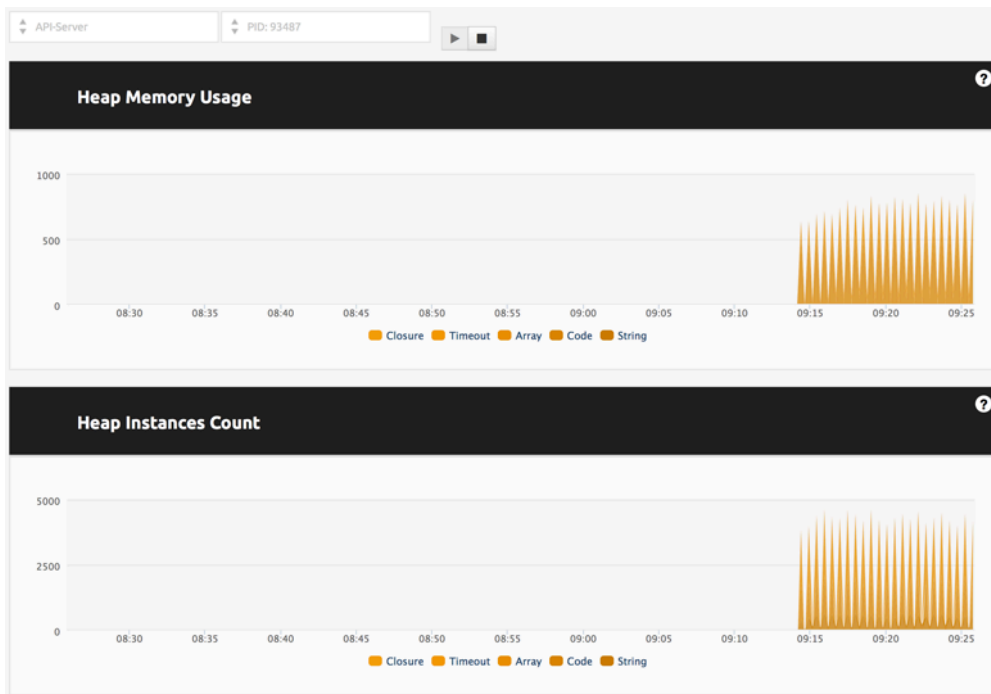
- Process specific cpu utilization profile
- Time distribution of code calls in perspective of CPU footprint over the collection period
- Call chains with insight into multiple child elements
- Ability to hide or explore elements in a call chain taking less than 5% time of parent
- Ability to hide or explore elements with single child elements
- Mouse-over and printing of line of code as we browse through the CPU footprint distribution for hotspots

All the above features are highlighted in the picture.

Visual aid: The sliding ball drives the control and granularity of the profile view. Additional check boxes and radio buttons are provided for collapsing the call chains with single elements and scaling the plot with time of descendent objects.

Memory profiler

Memory profiling (also called heap profiling) enables you to monitor the count and memory use of all instances of a constructor type. By tracking these metrics, you can identify when heavy data types are constructed too frequently or not freed from memory quickly enough.



To use it, go to the Memory Profiler tab and click **Start**. You should start to see lines for several constructor types on the chart. Usually the most useful thing to watch for is rapid growth in the memory footprint or instance count of a particular constructor type. This usually indicates a memory leak.

Some generic constructor types such as Array or Object, can be difficult to debug. Usually they are properties of another constructor, so you can try looking for constructor types with high instance count growth, but low memory usage.

Error reporting

The StrongOps console can display an error log for Express applications, including LoopBack applications.

To view an application's error log:

1. Click **Errors** in the StrongOps console main menu.
2. Choose the application from the drop-down list shown on the left.
3. By default, the dashboard shows errors for all processes. To view errors for a specific process, click **All processes** then select the desired process ID.

error-demo

☐ All processes

PID: 1129

express GET /hello:name
Wed Dec 04 2013 21:55:31 GMT-0800 (PST)
timestamp: 138622931000

PID: 1129

Error: Forcing a failure

```

at url (/home/ubuntu/errortest/app.js:21:8)
at args.(anonymous function) (/home/ubuntu/strong-agent/lib/proxy.js:85:18)
at callbacks (/home/ubuntu/errortest/node_modules/express/lib/router/index.js:164:37)
at param (/home/ubuntu/errortest/node_modules/express/lib/router/index.js:138:11)
at param (/home/ubuntu/errortest/node_modules/express/lib/router/index.js:135:11)
at pass (/home/ubuntu/errortest/node_modules/express/lib/router/index.js:145:5)
at nextRoute (/home/ubuntu/errortest/node_modules/express/lib/router/index.js:100:7)
at callbacks (/home/ubuntu/errortest/node_modules/express/lib/router/index.js:167:11)
at b (domain.js:183:18)
at Domain.run (domain.js:123:23)
          
```

Where applicable, the metadata will include details of the transaction or command where an error occurred.



Strong-agent does not provide a mechanism for your app to safely continue running after an error. If your app would have normally crashed and exited, it will continue to do so.


Configuring alerts

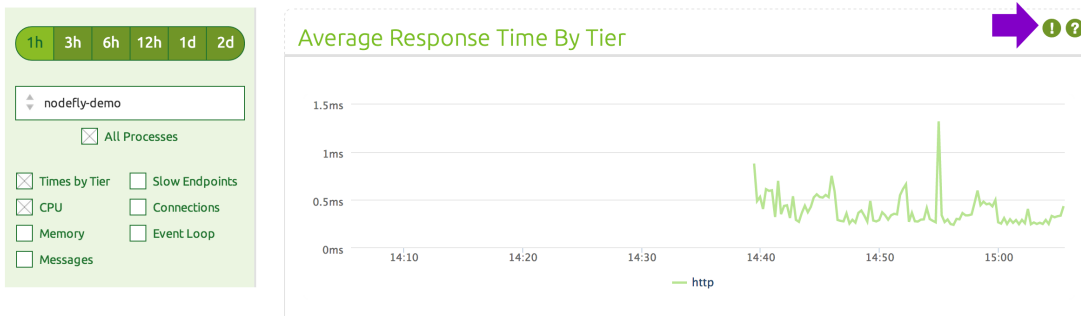
Overview

Create an *alert* to have StrongOps send a notification when a particular metric matches a predefined condition. For example, you could create an "excessive CPU" alert to email appropriate people when an app exceeds 75% CPU usage.

Creating an alert

To configure an alert with the StrongOps dashboard:

1. View the chart for the metric on which you want to set an alert.
2. click the alert button  shown at the upper right of the chart, as illustrated below.



3. In the dialog that appears, enter:

- A name for the alert. This is just an identifier.

REVIEW COMMENT
Where is this used?

- The metric on which to set the alert, for example incoming HTTP requests.
- Comparison operator, for example ">" for greater than.
- Threshold value: level of the specified metric that will trigger the alert.
StrongOps compares the specified metric to the threshold value using the comparison operator. If the result is true, then the alert is triggered.
For example: HTTP response time > 5000ms
- Duration in minutes

REVIEW COMMENT
What is this? Is it how long the alert will be active?

- Email address to send the alert notification; use comma-separated addresses to send to multiple recipients.

4. Click Add to create the alert.

When the metric in question meets the specified threshold comparison, StrongOps will send an email to the specified addresses.

Viewing and modifying alerts

Click **Alerts** in the StrongOps console main menu to review configured and active alerts. For each active alert, you'll see an entry for each time the alert's criteria are met. This enables you to see a history of the alert status.

Alert Name: slow request		config: tiers_http > 6000 for 0.001mins			edit config	delete config
AppName: Chimera1						
	Date	Alert Status	PID	Is Process Running?		
clear	Dec 04, 2:12PM	active	30751	Yes		

To clear a specific alert instance, click **clear**. This removes the entry from the alert table.

To modify an existing alert, click **edit config**.

To delete an alert and stop any future notifications for that configuration, click the red **delete config** button.

Controlling an application cluster

- [Overview](#)
- [Prerequisites](#)
- [Managing clusters with the StrongOps dashboard](#)
 - [Select application](#)
 - [Scale to size](#)
 - [Control workers](#)

Overview

A *application cluster* uses [Node's cluster feature](#) to create child processes that all share the same network port. This enables applications to take advantage of multi-core systems.

The StrongOps console enables you to view and change the number of worker processes in a cluster, once you have enabled cluster control for the application.



Do not make changes to clusters using the StrongOps dashboard and the `slc clusterctl` CLI command simultaneously.

For more information see:

- [Clustering applications](#).
- Node's [cluster documentation](#).

Prerequisites

An app must call the strong-agent `profile()` function for you to be able to monitor and manage it with the StrongOps console. See [Setting Up StrongOps](#) for more information.

To enable cluster control, the app must also call `control.start()`.

To use cluster controls, the app must use:

- strong-agent version 0.2.19 or later
- strong-cluster-control version 0.2.0 or later

Managing clusters with the StrongOps dashboard

To control a cluster, click **Cluster** in the main navigation. You'll see a page with three panes: **Select Application**, **Scale to Size**, and **Control Workers**.

Select application

Click the drop-down list under Select Application to choose the application for which you want to control clustering. Choose an application by name, and if it is using strong-cluster control you will be able to choose the master process ID from another drop-down.

Scale to size

The middle pane displays the current count of workers and the number of CPUs available. Enter a value in the text field and click **Set Size** to set the desired

Scale to Size:

4

4 Current Size

1 # CPUs

Cluster Control

Select Application:

cluster-test0



Master PID: 11249

number

of workers. The StrongOps console will send a message to the master process to adjust the number of workers accordingly. The display will update as the number of workers changes.

Control workers

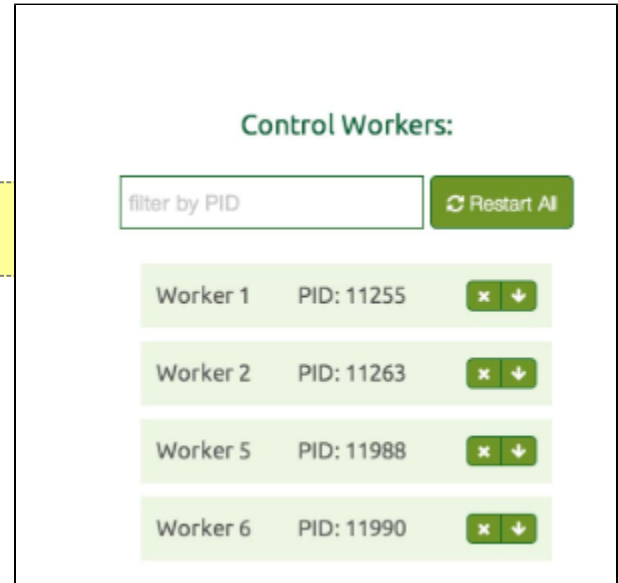
The right pane displays a list of active workers and their process IDs (PIDs). To filter the list by process ID, enter a partial or full PID in the text field.

Click  corresponding to a worker to terminate the process. Click  to shutdown the process

REVIEW COMMENT

What's the difference between terminate and shutdown?

Click **Restart All** to perform a rolling restart of **all** the workers (not just the filtered ones).



Using Strong Supervisor

Overview

[Strong Supervisor](#) enables you to run applications with Node clustering and control the cluster using the StrongOps console, without changing application code.

Strong Supervisor enables you to:

- Control application clusters through the StrongOps console. Supervisor will run the application clustered by default, maintaining one worker per CPU. It does this using [strong-cluster-control](#). You can configure this behavior (see `loadOptions()` in the [strong-cluster-control](#) documentation).
- Run your Node application as a daemon (a background process). Supervisor can detach the master from the controlling terminal, allowing to run as a daemon. This behaviour is optional, see the `--detach` option.
- Log output of a detached daemon to a file. Supervisor can optionally direct the output of a detached daemon to a file.
- Create a process identifier (PID) file. Supervisor can optionally write a PID file with the master process ID. This could be useful to send signals to a detached process from within system startup scripts as used by `init` or `upstart`.
- Perform signal handling. Supervisor will attempt a clean shutdown of the cluster before exiting if it is signalled with `SIGINT` or `SIGTERM`, see [control.stop\(\)](#). If the supervisor is detached, it will attempt a restart of the cluster if it is signalled with `SIGHUP`, see [control.restart\(\)](#).

Running an application with Strong Supervisor

Follow the steps in [Getting started](#) to install the StrongLoop command-line tool (`slc`), create a new LoopBack application using `slc lb` project and register for StrongOps. Then run the application as follows:

```
$ sl-run
```

Now go to the [StrongOps dashboard](#). If you haven't already done so, register with Strongloop.

Now you can:

- Increase or decrease the cluster size.

- Restart the cluster.
- Change the cluster size from the command line (`clusterctl set-size 20`) while the above are still in progress.
- Observe the PIDs on the panel, go to the command line, and kill it! (`kill -KILL PID`) It will get restarted.
- Kill another worker (kill a bunch!), using the X icon to it's right in the panel.
- Start a second instance of the application, on another machine if you have one, on the same machine but different port if you are just trying things out: `PORT=30001 sl-run`, note the second master in the Panel.

Understanding strong agent

- Configuration
 - The `package.json` file
 - The `strongloop.json` file
 - Environment
- Enabling profiling

Strong agent enables StrongOps to monitor and profile your Node processes and clusters.

Configuration

Obtain an API key by registering at <http://www.strongloop.com/ops> or by using the `slc` command:

```
$ slc strongops
```

Pass configurations directly to `profile(key, appName)`, or go in your `package.json`, `strongloop.json` config or environment variables.

Strong agent searches for the key and app name in this order:

- Arguments to the `slc strongops` command
- Environment variables
 - `SL_APP_NAME= Application name`
 - `SL_KEY= API key`
- The `strongloop.json` file in app directory.
- The `package.json` in app directory.
- The `strongloop.json` file in your home directory (`~/strongloop.json`)

The package.json file

Enable StrongOps monitoring by adding the following to `package.json` in the app's root directory.

```
{
  ...
  "name": <application name>,
  "strongAgentKey": <api key>,
  ...
}
```

The strongloop.json file

Enable StrongOps monitoring by adding the following to `strongloop.json` in the app's root directory or your home directory.

```
{
  "key": <api key>,
  "appName": <application name>
}
```

Environment

Enable StrongOps monitoring by setting the following environment variables:

SL_APP_NAME=<application name>
SL_KEY=<api key>

Enabling profiling

Once you have an API key, you can enable profiling by simply running your app as follows:

```
$ sl-run
```



The sl-run command is provided in version 2.2.2 of strong-cli. If you have an older version, see [Updating to the latest version](#) for instructions on updating.

If you have an old version of the CLI tools, you can enable monitoring by adding the following at the beginning of your code:

```
require('strong-agent').profile();
```

Open data API

StrongOps provides performance data via the open data API for use by proprietary reporting tools and integration into broader operations systems.

Open data API is built on top of [JSONP](#) and enables you to pull metrics and graph data hosted on the StrongOps console. This data extract is based on the user profile of registered users and their applications. The API uses the same authentication credentials as the StrongOps console and shares the same contexts of connected sessions, cluster keys, and PIDs.

The API is available at <http://strongops-api.strongloop.com/>.

- [Register](#)
- [Login](#)
- [Logout](#)
- [Get user information](#)
- [Change password](#)
- [Delete user](#)
- [Get list of clusters](#)
- [Get cluster information](#)
- [Delete cluster](#)
- [Get session](#)
- [Get errors](#)
- [Get graph data](#)

Quick reference

URI Pattern	HTTP Verb	Action	Parameters
/v2/register	POST	Register user.	Form data: email, password, and name.
/v2/login	POST	Log in user	Form data: email and password.
/v2/logout	POST	Log out user	
/v2/users/me	GET	Get user information	
/v2/users/me	UPDATE	Change user password	
/v2/users/me	DELETE	Delete user	
/v2/clusters?sessions=[true false]	GET	Get list of clusters	
/v2/clusters/clusterId?sessions=[true false]	GET	Get cluster information	Cluster ID
/v2/clusters/clusterId	DELETE	Delete cluster	Cluster ID
/clusters/clusterId/sessionId	GET	Get session information	Cluster ID, session ID
/clusters/clusterId/errors	GET	Returns any errors reported in specified timeframe	Cluster ID
/v2/clusters/clusterId/graphData	GET	Returns an graph data for a given timeframe	Cluster ID

Register

POST /v2/register

Parameters

Form data: email, password, and name.

Return value

If successful, returns 200 OK with a session cookie.

Login

POST /v2/login

Parameters

Form data: email and password.

Return value

Returns a session cookie. Include the session cookie with all subsequent requests.

Logout

Logs out the user and invalidates the session cookie.

POST /v2/logout

Get user information

Get information on the user.

GET /v2/users/me

Return value

This resource returns user information in JSON format; for example:

```
{ id: '2dd04f9aa1bf44907139dd5be98e26af',  
  userId: 43,  
  name: 'Test User',  
  userKey: '2dd04f9aa1bf44907139dd5be98e26af',  
  email: 'test@strongloop.com',  
  created: '2013-11-15T19:45:17.000Z',  
  apps: []  
}
```

Change password

Updates the user's password.

UPDATE /v2/users/me

Parameters

Form data email and password.

Delete user

Deletes your account from StrongOps, so use it carefully.

```
DELETE /v2/users/me
```

Parameters

Form data email and password.

Get list of clusters

```
GET /v2/clusters
```

```
GET /v2/clusters?sessions=true
```

Return value

Returns an array of cluster informatino in the following format (*sessions* is omitted if *sessions=false*):

```
[ { clusterId: '3dc033eb201f78d59996ffed9fela10a',
  appId: 1,
  appName: 'API Test 1',
  userId: 2,
  ts: '2013-10-15T17:17:00.000Z',
  procCount: 0,
  sessions:
    [ { sessionId: 'RE684CltzC4DRtSo0Ij7hsGB',
      appId: 1,
      hostname: 'Hildisvini.local',
      appName: 'API Test 1',
      agentVersion: '0.2.15',
      pid: 99135 },
      { sessionId: '4uzxFrAekcmYtc9VrSXlriDv',
        appId: 1,
        hostname: 'Hildisvini.local',
        appName: 'API Test 1',
        agentVersion: '0.2.15',
        pid: 4369 },
      { sessionId: 'Uwepu78fsbXoNcOr0Gjvsgkj',
        appId: 1,
        hostname: 'Hildisvini.local',
        appName: 'API Test 1',
        agentVersion: '0.2.15',
        pid: 7123 } ]
} ]
```

Get cluster information

```
GET /v2/clusters/clusterId
```

```
GET /v2/clusters/clusterId?sessions=true
```

Parameters

Path parameter *clusterId*, the Cluster ID.

Example

```
GET /v2/clusters/3dc033eb201f78d59996ffed9fe1a10a?sessions=false
```

Sample output (without sessions):

```
{ clusterId: '3dc033eb201f78d59996ffed9fe1a10a',  
  appId: 1,  
  appName: 'API Test 1',  
  userId: 2,  
  ts: '2013-10-15T17:17:00.000Z',  
  procCount: 0 }
```

Delete cluster

Delete the specified cluster

```
DELETE /v2/clusters/clusterId
```

Parameters

Path parameter *clusterId*, the Cluster ID.

Example

```
DELETE /v2/clusters/3dc033eb201f78d59996ffed9fe1a10a
```

Get session

```
GET /clusters/clusterId/sessionId
```

Parameters

Path parameters *sessionId*, the session ID and *clusterId*, cluster ID.

Example

```
GET /v2/clusters/3dc033eb201f78d59996ffed9fe1a10a/Uwepu78fsbXoNcOr0Gjvsgkj
```

Sample output:

```
{ sessionId: 'Uwepu78fsbXoNcOr0Gjvsgkj',  
  appId: 1,  
  hostname: 'Hildisvini.local',  
  appName: 'API Test 1',  
  agentVersion: '0.2.15',  
  pid: 7123 }
```

Get errors

GET /clusters/*clusterId*/errors

Parameters

Path parameter *clusterId*, the Cluster ID.

Query parameters: *start*, *end* (both UNIX timestamps) and *range* (number of seconds).

Return value

Returns any errors (stack traces) that have been reported in a given timeframe.

Example: GET /v2/clusters/3dc033eb201f78d59996ffed9fela10a/errors?end=1384546713&range=3600&start=1384543113

This can also have a *sessionId* specified as in "Get specific session" above, for example: GET

/v2/clusters/3dc033eb201f78d59996ffed9fela10a/dYAev2kaeWCVp6jtp7cK6MQw/errors?end=1384546713&range=3600&start=1384543113

Sample output:

```
[ { sessionId: 'dYAev2kaeWCVp6jtp7cK6MQw',
  type: 'express',
  command: 'GET /throwError',
  stack: 'Error: express go boom now\n    at
/Users/andrew/nf/example/example.js:47:9\n    at args.(anonymous function)
(/Users/andrew/nf/strong-agent/lib/proxy.js:85:18)\n    at callbacks
(/Users/andrew/nf/example/node_modules/express/lib/router/index.js:161:37)\n    at
param (/Users/andrew/nf/example/node_modules/express/lib/router/index.js:135:11)\n
at pass (/Users/andrew/nf/example/node_modules/express/lib/router/index.js:142:5)\n
at nextRoute
(/Users/andrew/nf/example/node_modules/express/lib/router/index.js:100:7)\n    at
callbacks (/Users/andrew/nf/example/node_modules/express/lib/router/index.js:164:11)\n
at b (domain.js:183:18)\n    at Domain.run (domain.js:123:23)\n    at
/Users/andrew/nf/strong-agent/lib/errors.js:63:9',
  ts: '2013-11-15T20:18:25.000Z',
  clusterId: '3dc033eb201f78d59996ffed9fela10a' },
  { sessionId: 'dYAev2kaeWCVp6jtp7cK6MQw',
  type: 'express',
  command: 'GET /throwError',
  stack: 'Error: express go boom now\n    at
/Users/andrew/nf/example/example.js:47:9\n    at args.(anonymous function)
(/Users/andrew/nf/strong-agent/lib/proxy.js:85:18)\n    at callbacks
(/Users/andrew/nf/example/node_modules/express/lib/router/index.js:161:37)\n    at
param (/Users/andrew/nf/example/node_modules/express/lib/router/index.js:135:11)\n
at pass (/Users/andrew/nf/example/node_modules/express/lib/router/index.js:142:5)\n
at nextRoute
(/Users/andrew/nf/example/node_modules/express/lib/router/index.js:100:7)\n    at
callbacks (/Users/andrew/nf/example/node_modules/express/lib/router/index.js:164:11)\n
at b (domain.js:183:18)\n    at Domain.run (domain.js:123:23)\n    at
/Users/andrew/nf/strong-agent/lib/errors.js:63:9',
  ts: '2013-11-15T20:17:35.000Z',
  clusterId: '3dc033eb201f78d59996ffed9fela10a' } ]
```

Get graph data

GET /v2/clusters/*clusterId*/graphData

Parameters

Query parameters `start,end` (both UNIX timestamps) and `range` (number of seconds).

Additionally, one can specify a `bucket` to use. Data is rolled up into various buckets containing aggregated data appropriate for each time scale. Acceptable values for `bucket` would be one of `[1,2,5,10,15,30,60,120,180,360,720,1440]`.

One also needs to specify the `graphs` for specific data. Acceptable values for `graphs` include TBD

Return value

Returns an object containing data for a given timeframe.

Example

```
/v2/clusters/3dc033eb201f78d59996ffed9fe1a10a/graphData?bucket=1&start=1384548361&range=3600&graphs=tiers,top.cpu,conn,mem,eventloop,strongl
```

The data that you are most likely interested in are stored as `[timestamp,value]`

```
{ "update":
  { "externalCalls": [],
    "top_functions": [],
    "mysqlCalls": [] ,
    "mongoCalls": [
      { "q": "chimera.dummydata.update({\"key\":\"\"})",
        "duration": 39,
        "pie": null,
        "path": null
      } ],
    "redisCalls": [],
    "memcacheCalls": [
      { "q": "set hello",
        "duration": 38,
        "pie": null,
        "path": null
      } ],
    "oracleCalls": [],
    "connections": {
      "concurrent": [[1384547521,1],[1384547581,1],[1384547700,1],[1384547760,1]],
      "throughput":
[[1384547521,0.01666667],[1384547581,0.01666667],[1384547700,0.01666667],[1384547760,0
]]
    },
    "cpu": {
      "stime":
[[1384547521,0],[1384547581,0],[1384547640,0],[1384547700,0.0167308],[1384547760,0]],
      "utime":
[[1384547521,0.10041841],[1384547581,0.05025967],[1384547640,0.05031869],[1384547700,0
.0501924],[1384547760,0.03347841]]
    },
    "times_by_tier": {
      "http":
[[1384547521,0],[1384547581,0],[1384547640,0],[1384547700,0],[1384547760,0]],
      "memcached_in":
[[1384547521,0],[1384547581,1.753],[1384547640,0],[1384547700,2.005],[1384547760,0]],
      "mongodb_in":
[[1384547521,0],[1384547581,1.946],[1384547640,0],[1384547700,2.468],[1384547760,0]],
      "mysql_in":
[[1384547521,0],[1384547581,0],[1384547640,0],[1384547700,0],[1384547760,0]],
```

```
    "redis_in":  
[[1384547521,0],[1384547581,0],[1384547640,0],[1384547700,0],[1384547760,0]]  
    },  
    "heap": {  
  
"heap": [[1384547521,43.266976],[1384547581,44.487584],[1384547700,45.680736],[13845477  
60,46.15592]],  
  
"rss": [[1384547521,82.8416],[1384547581,82.8416],[1384547700,82.8416],[1384547760,82.8  
416]],  
    "v8gc": [[1384545240,31.98568291],[1384546980,32.71478691]]  
    },  
    "queue": {  
  
"blockfrequency": [[1384547521,0],[1384547581,0],[1384547640,0],[1384547700,0],[1384547  
760,0]],  
  
"blocktime": [[1384547521,0],[1384547581,0],[1384547640,0],[1384547700,0],[1384547760,0  
]]  
    },  
    "mq": {}  
}
```

```
} ,  
"end":1384547760  
}
```