



## Getting Started with DevOps for Node.js Applications

*An introduction to using StrongOps for the effective development  
and management of Node applications*

# Table of Contents

<b>Introduction.....</b>	<b>3</b>
<b>What is DevOps? .....</b>	<b>3</b>
<b>Introducing StrongOps .....</b>	<b>3</b>
<b>Development.....</b>	<b>4</b>
<b>QA &amp; Test.....</b>	<b>6</b>
<b>Performance Monitoring.....</b>	<b>11</b>
<b>Process Management &amp; Clustered Scaling .....</b>	<b>15</b>
<b>Reporting and Analytics .....</b>	<b>20</b>
<b>Conclusion.....</b>	<b>22</b>

## Introduction

In this paper we will explore at a high-level the considerations you should have when developing a DevOps strategy for Node applications. We'll also focus on how StrongLoop's StrongOps performance monitoring and DevOps service can help you achieve some of your DevOps goals.

## What is DevOps?

According to Wikipedia, DevOps is, "a software development method that stresses communication, collaboration and integration between software developers and IT professionals. DevOps is a response to the interdependence of software development and IT operations. It aims to help an organization rapidly produce software products and services."

### *Translation*

StrongLoop's perspective is that DevOps should target six core competencies. Specifically, as it relates to Node.js applications – they should be development, quality assurance and testing, performance tuning, monitoring, process management and scaling plus, reporting and analytics.

## Introducing StrongOps

StrongLoop's [StrongOps](#) is a performance monitoring and DevOps solution specifically designed for Node applications. StrongOps is a subscription-based and offered in a software-as-a-service model. It requires a simple sign up and the installation of a small, low-overhead agent, packaged as a Node module that can be installed via npm.

## Installing StrongOps

Getting started with StrongOps is fast and easy. Installation instructions are available on the [strongloop.com](#) [Get Started](#) page. By following these instructions you'll get

StrongOps installed and monitoring a sample Node application running on top of the LoopBack API server with a running load generator to populate the graphs.

## Development

To get started on building node.js applications, developers have to first download and install the Node runtime and core modules. Today, this is distributed on behalf of the entire Node community from [npmjs.org](https://npmjs.org) website. The distribution also contains a client for the package manager called Node Package Manager (NPM). Developers use the NPM client to connect to an NPM service or global public registry running on a cloud and pull down the modules required for their projects. Conversely, developers who wish to commit their private modules back to the community can do so by publishing to projects on NPM. The modules on NPM are segmented as “Core” and “Community” modules. Almost anyone can publish modules to “Community”. However, contributions to “Core” projects have to be approved by project leads.

Enterprises who want to maintain source code and version control are concerned about the high availability and security of the modules. StrongOps helps organizations during this phase by enabling them to setup and maintain a private registry for their Node modules, as well as, provide support for popular and StrongLoop npm modules.

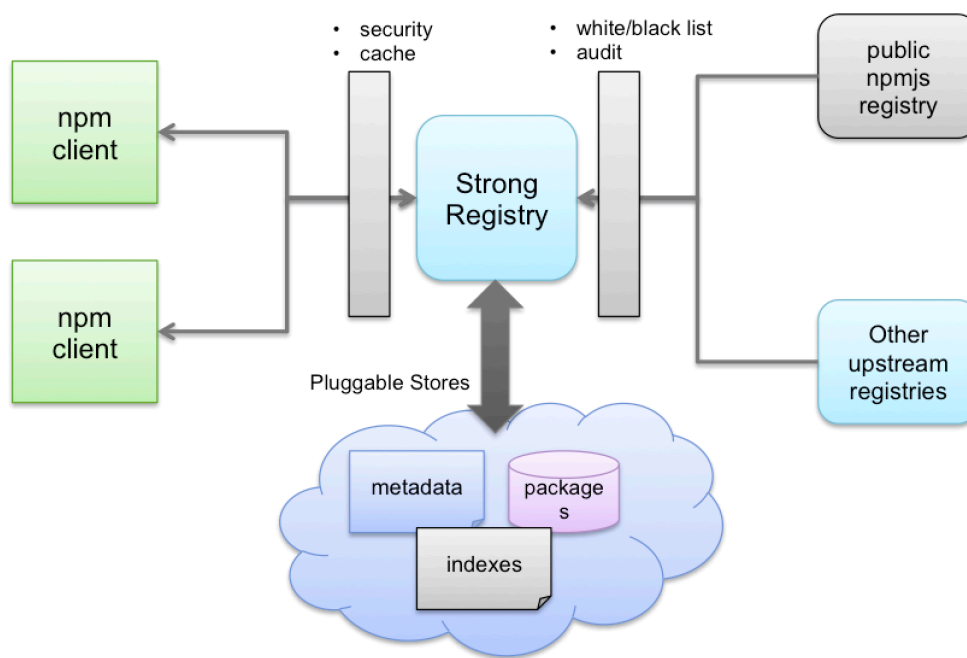
## Private Registry

A private registry helps organizations simplify the management of their code, provide consistency in how applications are built and deployed, plus help those that require some degree of compliance. The benefits of a private registry include:

- Public and private modules secured in a single place
- Version management and enforcement of various modules
- Simplified user management by eliminating multiple npm accounts
- Increased availability and performance of accessing modules

StrongLoop provides a lightweight private registry solution based on [node-reggie](#). To learn how to setup a private registry, visit the [StrongLoop documentation](#). StrongLoop also provides consulting services to build customized private registries. Capabilities include:

- Allowing the publishing of private modules to on-premise registries
- Use standard npm install workflow to install both public (npmjs.org) and private modules
- Promotion of modules from Dev-Test-Prod
- Improved reliability
- Preventing problems due to unpublished packages by localization (lazy mirroring)
- Pluggable/swappable backend stores (Artifactory and others)
- Authentication and authorization
- Whitelist and black-list, Audits
- ACLs



The above diagram shows generic architecture for private registry solution StrongLoop helps implement for enterprises.

## Supported Node Core and Modules

StrongLoop provides bug fixes and technical support for Node core, libuv, a curated list of common npm modules, plus StrongLoop maintained modules for debugging, clustering and enterprise database connectors. Check out the documentation for a complete [list of support modules](#).

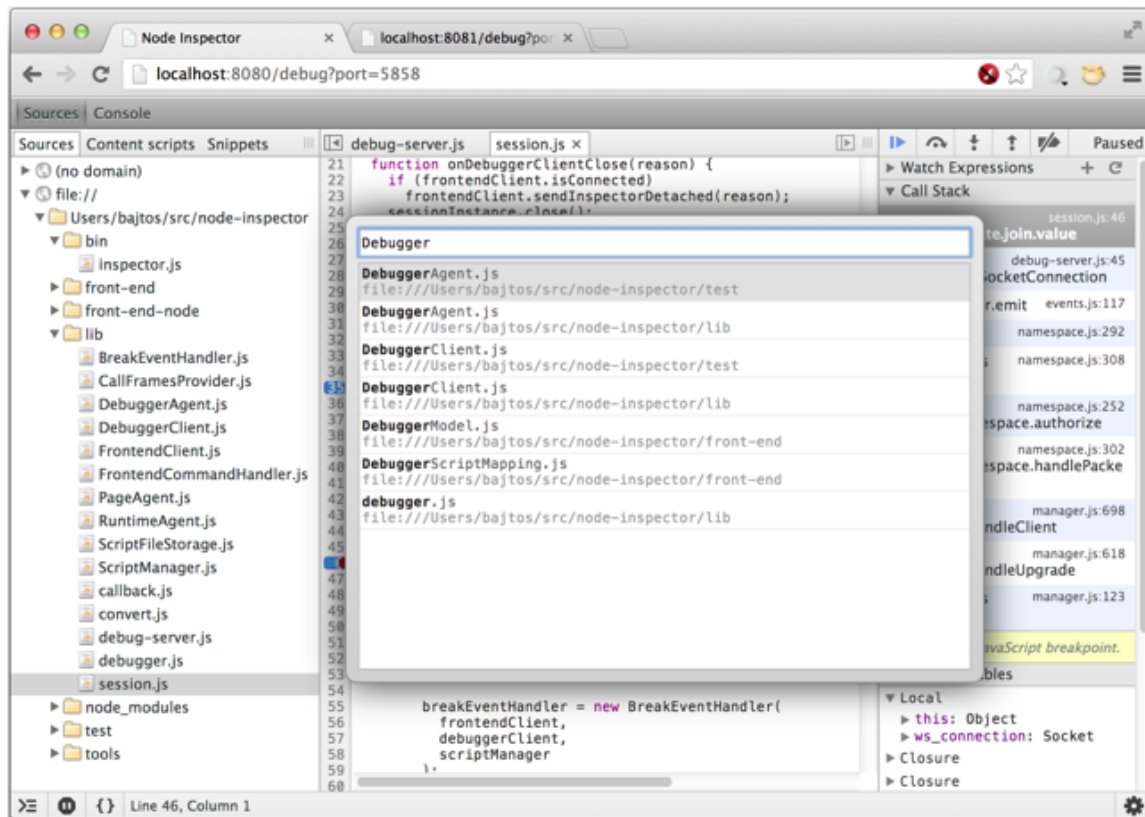
## QA & Test

Similar to applications built in any technology, quality assurance and testing is critical to building defect free Node applications. While being supported at some level by some IDEs like WebStorm, Sublime Text, Node-Eclipse, etc, Node lacks dedicated IDEs. Unit testing frameworks like Mocha and Jasmine are popular, while not being prescriptive. During the coding and testing phase, in-depth debugging and bottleneck detection in the application's performance helps developers write better code. StrongOps helps organizations during this phase by enabling them to debug, profile and detect errors in their code whether still in development or production.

## Debugging

Debugging Node applications is very difficult due to the asynchronous nature of calls and flow through its declared and anonymous functions. This makes it difficult in Node to correlate error events, inspect JavaScript and visualize call-stacks. Having a good debugger reduces development and QA effort by providing quick resolution to errors.

StrongLoop is the maintainer of the popular [Node Inspector](#) module. Node Inspector is a debugging interface for Node applications that leverages Google's Blink Developer Tools.



Key features of Node Inspector include:

- Navigate within your source files
- Set breakpoints and specify trigger conditions
- Step over, step in, step out and resume
- Capture uncaught exceptions
- Inspect scopes, variables and object properties
- Edit variables and object properties
- Break on exceptions
- Remote debugging
- Set breakpoints in files not yet loaded into V8
- Integration with unit testing frameworks like "Mocha"
- Re-execute functions with "Restart Frame"
- Support for source maps

- Auto-load files during debug session
- Open source and built 100% in JavaScript

For more information, visit the [Node Inspector](#) project home page.

## Code Profiling

Node Inspector also provides detailed code profiling. Call stacks along with test framework integration, provide code coverage during unit tests or live debug sessions.

The screenshot displays the Node Inspector interface with two main panels:

**Call Stack:**

Function	File
Module.compile	module.js:456
Module.extensions.js	module.js:474
Module.load	module.js:356
Module._load	module.js:312
Module.runMain	module.js:497
listOnTimeout	timers.js:110

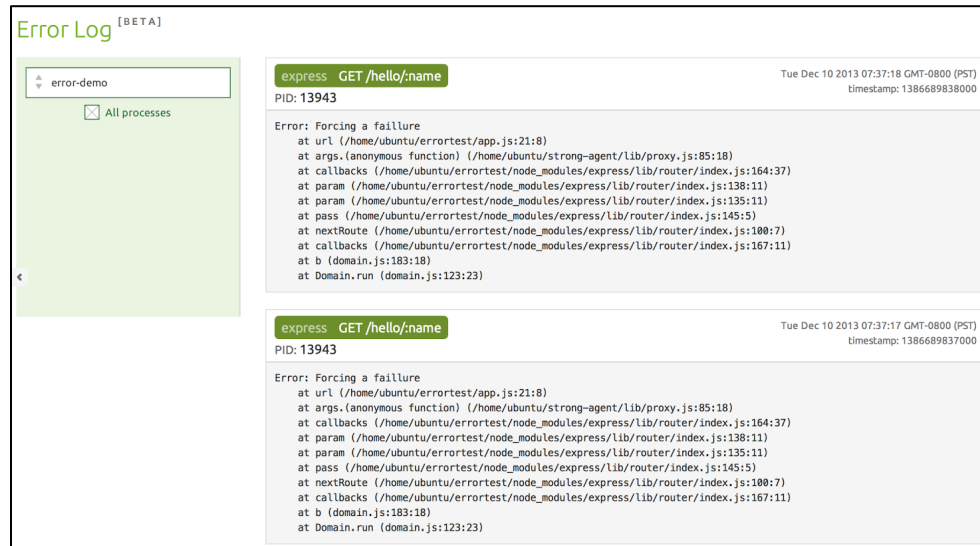
**Local:**

```

▼ Local
  ▼ args: Array[5]
    ▼ 0: Object
      ► __proto__: Object
    ▼ 1: function require(path) {
      arguments: null
      ► cache: Object
      caller: null
      ► extensions: Object
      length: 1
      ► main: Module
      name: "require"
      paths: undefined
      ► prototype: require
      ► registerExtension: function () {
      ► resolve: function (request) {
      ► __proto__: function Empty() {}
      ► <function scope>
    ▼ 2: Module
      ► children: Array[0]
      ▼ exports: Object
        ► __proto__: Object
      filename: "/usr/local/lib/node_modules/node-inspector/bin/run-repl.js"
      id: "."
      loaded: false
      parent: null
      ▼ paths: Array[6]
        0: "/usr/local/lib/node_modules/node-inspector/bin/node_modules"
        1: "/usr/local/lib/node_modules/node-inspector/node_modules"
        2: "/usr/local/lib/node_modules"
        3: "/usr/local/node_modules"
        4: "/usr/node_modules"
        5: "/node_modules"
        length: 6
        ► proto: Array[0]
  
```



## Error Detection

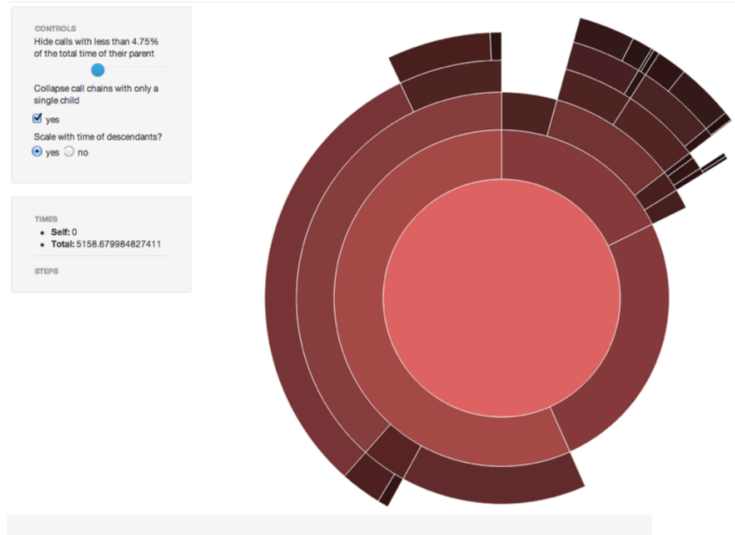


StrongOps error detection capabilities allow you to capture errors from [Express.js](#) applications and correlate them to individual requests or transactions. You can also aggregate errors by application and filter by individual PIDs. This is a very critical capability in production as there is no other way to attach a live error to a transaction. Log scraping does not provide any correlation due to asynchronous nature of Node. Live errors are captured in a running application and then StrongOps inspects and correlates all the function calls that are part of the error stack within a domain. Individual request IDs are then tracked to capture separate error stacks for each request.

## Performance Testing & Tuning

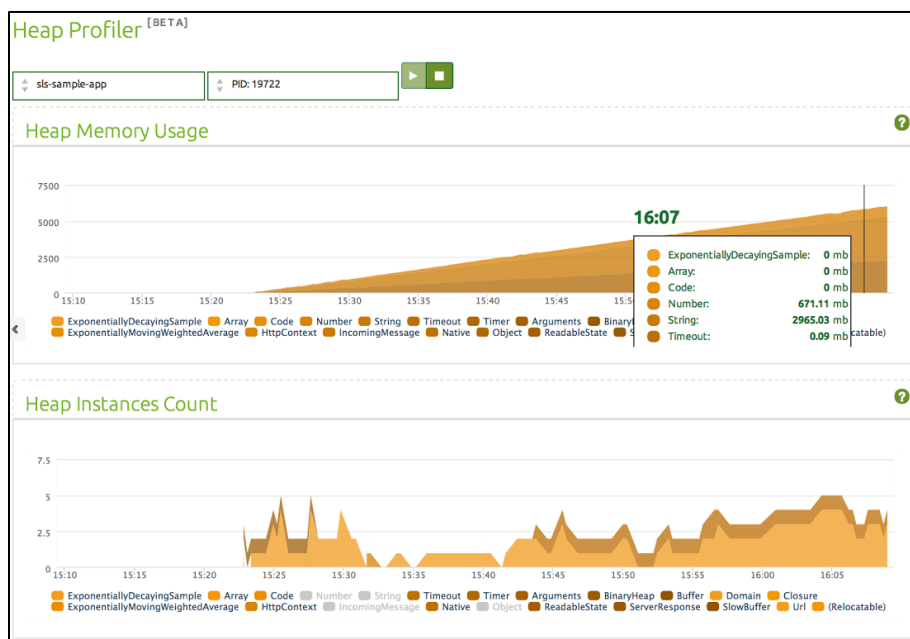
During performance testing, load tests are run against applications in staged environments similar to production. The benchmarking is conducted under different load patterns. Bottlenecks in performance are investigated and then fixed. Thus code is tuned and optimized for the best performance possible before promoting the application to production. Resource profiling is a key aspect of the tuning exercise. StrongOps helps organizations during this phase by enabling them to profile CPU, memory and heap usage, plus chase down application memory leaks both in pre-production and production.

## CPU Profiling



With the CPU profiling view you can visualize the CPU footprint of user and system space. You can also identify functions and lines of code within the application that may be indicative of CPU hotspots.

## Memory and Heap Profiling



The memory profiling view enables you to monitor the count and memory usage of all instances of the constructor type. (In JavaScript, new objects are created using a constructor.) You can also identify when heavy data types are constructed too frequently or not freed from memory quickly enough. StrongOps lets you narrow down up to object level memory allocation as well as instance counts to capture leaking segments of application over multiple GC iterations.

## Memory Leak Detection

Distance	Objects Count	Shallow Size	Retained Size
2	220 1%	6 472 0%	74 085 912 99%
3	1 172 8%	84 384 0%	73 084 992 98%
4	3 267 22%	72 788 360 98%	72 788 360 98%
5	"some big file with some big \$	540 312 1%	540 312 1%
6	"some 0 file with some 0 str	482 424 1%	482 424 1%
6	"some 0 file with some 0 str	482 424 1%	482 424 1%
6	"some 0 file with some 0 str	482 424 1%	482 424 1%
6	"some 0 file with some 0 str	482 424 1%	482 424 1%
6	"some 0 file with some 0 str	482 424 1%	482 424 1%
6	"some 0 file with some 0 str	482 424 1%	482 424 1%
6	"some 0 file with some 0 str	482 424 1%	482 424 1%
6	"some 0 file with some 0 str	482 424 1%	482 424 1%

Object	Shallow Size	Retained Size	Distance
▼ bigstr in function() @23339	72 0%	72 0%	4
▼ [19] in (Isolate) @15	0 0%	496 0%	3
[4] in (GC roots) @3	0 0%	516 800 1%	2
▶ [10] in (Isolate) @15	0 0%	496 0%	3
▶ callback in function wrapper() @23333	72 0%	72 0%	4
▶ bigstr in function() @19255	72 0%	72 429 624 97%	7

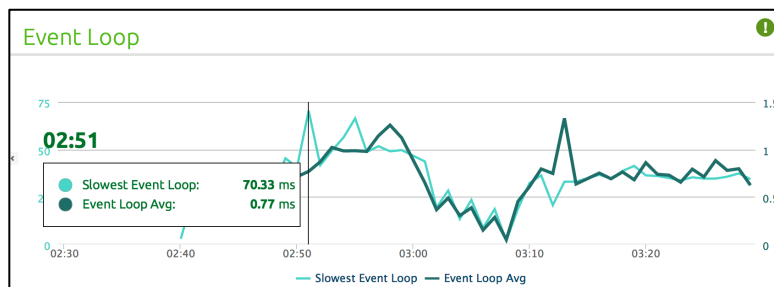
StrongOps enables you to capture V8 heap snapshots at runtime and inspect object sizes, groups, retaining paths and dominators. You can also program automatic snapshots at timed intervals. This is provided using a module called as [node-heapdump](#) and memory analysis is provided using Blink Toolkit integration. Memory leaks can be pinpointed down to the line of code. GC roots can also be diagnosed using StrongOps.

## Performance Monitoring

Application performance monitoring provides visibility into the customer experience and application behavior during production operations. This visibility enables the proactive notification and triage of incidents enabling operations teams to determine root causes. And then take corrective actions to reduce Mean Time To Restore (MTTR).

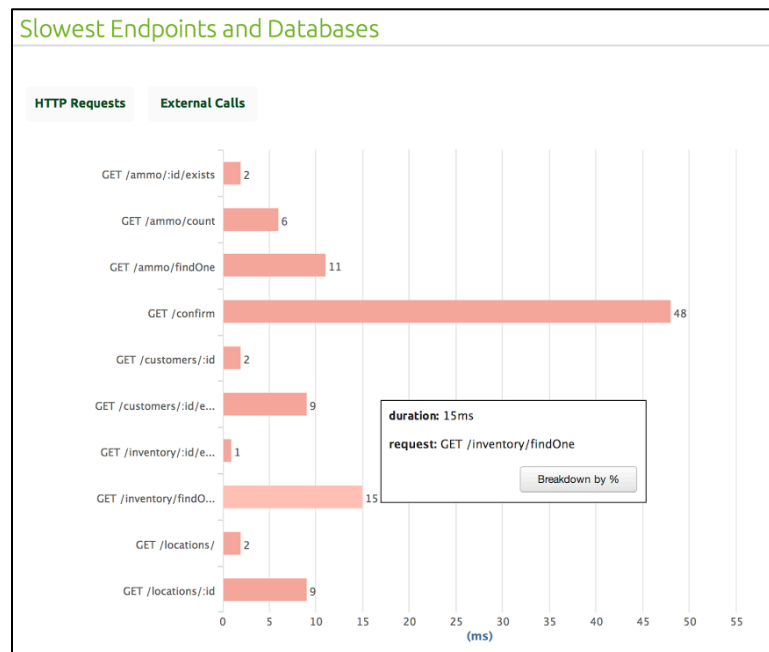
StrongOps helps organizations during this phase by enabling them to monitor the performance characteristics of their Node applications in real-time, while in production. Capabilities include event loop monitoring, slowest endpoints and databases, transaction profiling and messaging.

## Event Loop Monitoring



This view shows you event loop cycle times. The graph presents both the slowest event loop and the average time of all event loops.

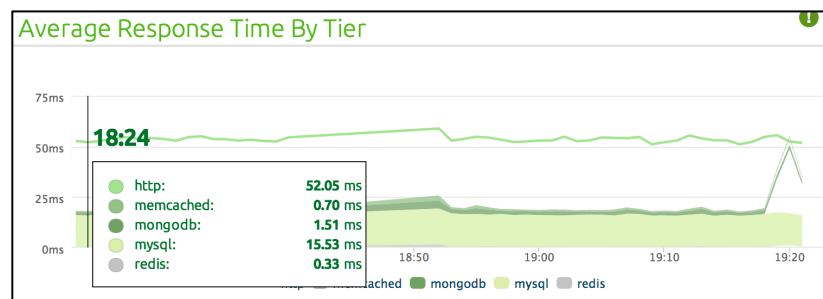
## Slowest Endpoints and Databases



This view gives you visibility into web services, relational, NoSQL and in-memory datastores. You can trace individual SQL queries or service request calls across the multiple tiers the applications interact with. Including:

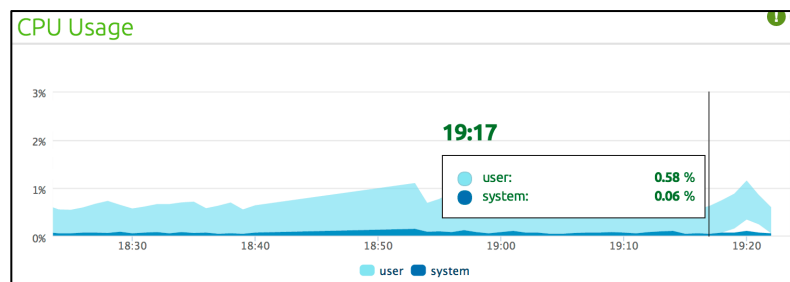
- HTTP requests to see the slowest URLs or routes
- External calls that show the slowest URLs requests
- Slow queries for the various databases supported

## Average Response Time by Tier



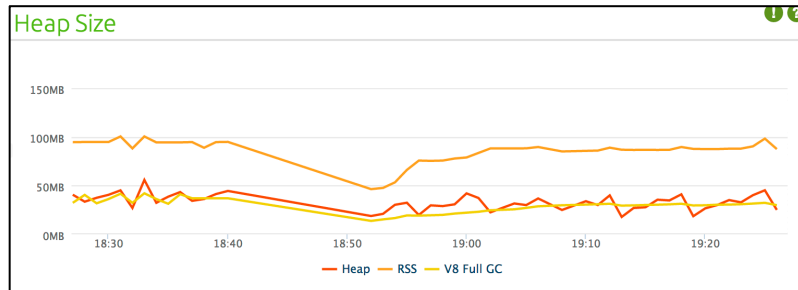
This view shows the average total time for any incoming HTTP request.

## CPU Usage



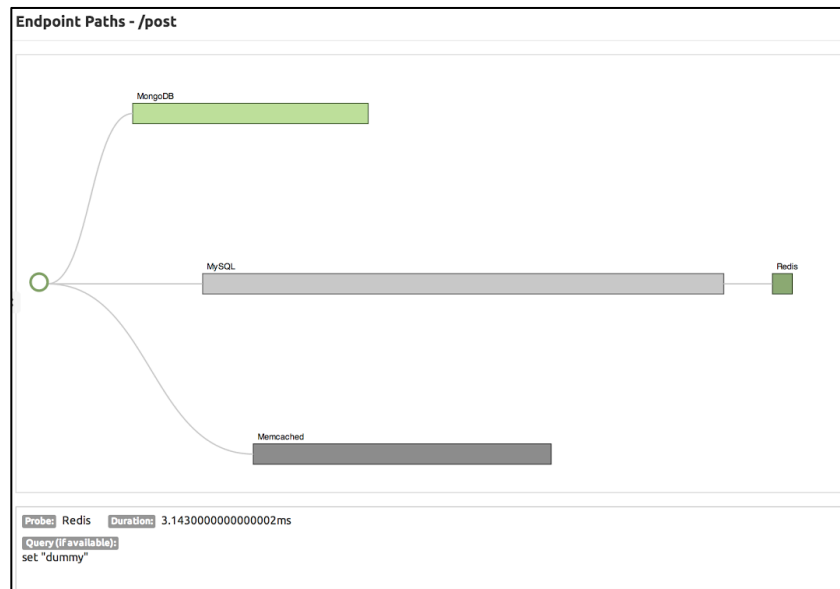
This view shows the time this process has spent in user and system space.

## Heap Size



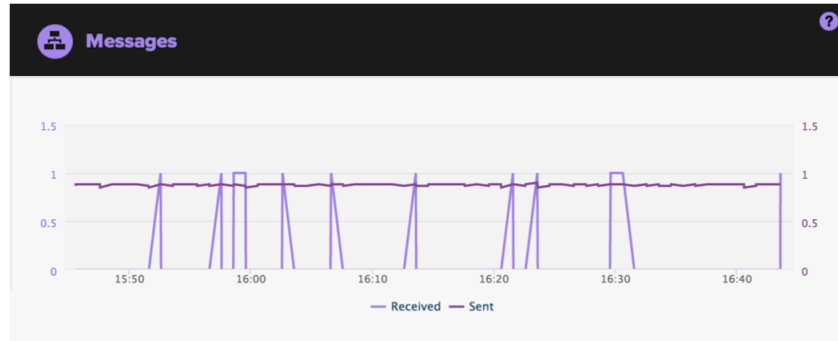
This view shows the current heap size, resident set size and the V8 GC which is sampled immediately after a full garbage collection.

## Transaction Profiling



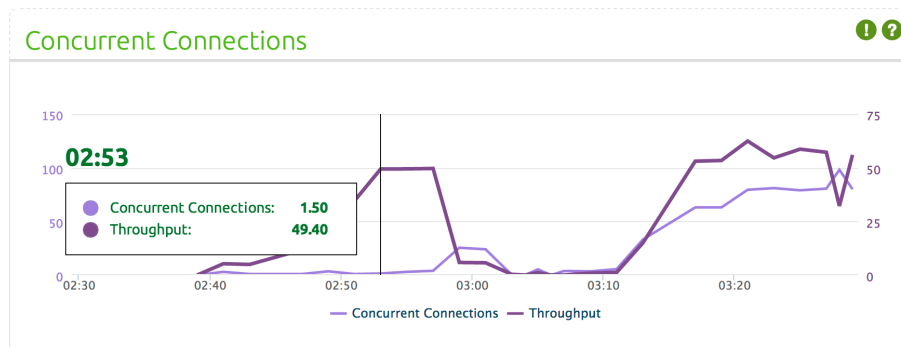
This view gives you visibility into the type of incoming HTTP request calls, call paths, backend systems being accessed, web-service calls and outbound HTTP responses. You can also drilldown and find out how much time is spent on each layer for an end-to-end view.

## Messaging



This view shows the number of messages sent and received when using the [strong-mq](#) messaging queue module.

## Throughput and Concurrency



This view shows the number of concurrent connections being handled by the node.js application process as well as the responses per interval or throughput of the application.

## Process Management & Clustered Scaling

Node applications are single threaded and each application instance runs as an independent process attached to one (1) CPU core of the server host. Out of the box, Node has poor support for the management of these runtime processes. Scaling is mostly achieved at the infrastructure level by running multiple machine hosts, each with a Node process and then horizontal scaling of machines. Network or web load

balancers like F5, Apache or Nginx are used for workload management. However, application level scaling and load-balancing is found wanting. Capabilities like high availability, hot deployments and rolling restarts are not as mature in Node compared to Java/J2EE or .Net based application servers.

StrongOps helps organizations during this phase by enabling them to manage detached processes and clusters, scale on demand plus hot deployments and rolling restarts. This is packaged as [strong-supervisor](#).

## **Detached Processes and Logging**

Running Node as a detached process needs a graceful control mechanism other than running a live process in the background of the terminal with an “&”. StrongOps provides a run time process manager (strong-supervisor) to provide this functionality. Controlled logging is included in this feature.



```
Shubhras-MacBook-Pro:sls-sample-app shubhrakar$ slc run --help
usage: slc run [options] [app [app-options...]]

Run an app, allowing it to be profiled (using StrongOps) and supervised.

`app` can be a node file to run or a package directory. The default
value is ".", the current working directory. Packages will be run by
requiring the first that is found of:
  1. server.js
  2. app.js
  3. `main` property of package.json
  4. index.js

Runner options:
  -h,--help          Print this message and exit.
  -v,--version        Print runner version and exit.
  -d,--detach         Detach master from terminal to run as a daemon (default
                      is to not detach).
  -l,--log FILE       When detaching, redirect terminal output to FILE, in
                      the app's working directory if FILE path is not
                      absolute (default is supervisor.log)
  -p,--pid FILE       Write supervisor's pid to FILE, failing if FILE
                      already has a valid pid in it (default is not to)
  --cluster N         Set the cluster size (default is off, but see below).
  --no-profile        Do not profile with StrongOps (default is to profile
                      if registration data is found).

Cluster size N is one of:
  - A number of workers to run
  - A string containing "cpu" to run a worker per CPU
  - The string "off" to run unclustered, in which case the app
    will *NOT* be supervisable or controllable, but will be monitored.

Clustering defaults to off unless NODE_ENV is production, in which case it
defaults to CPUs.
```

The graphic above shows all the runtime options for running detached processes (with `-d` option), directing terminal output to log files (with `-l` option), generating PID file (with `-p` option), et all.

## Cluster Control for On-Demand Scaling

At the core of StrongOps cluster management capabilities is support for the [strong-cluster-control](#) module in strong-supervisor. This module allows for the sizing of workers and monitors them for uptime. It also allows for soft shutdowns, hard terminations and can throttle worker restart rates if they exit abnormally.

Cluster Control [BETA]

Select Application:

sls-sample-app

Master PID: 70784

Scale to Size:

6

↑ Set Size

6

Current Size

8

# CPUS

Control Workers:

filter by PID

Restart All

Worker 1	PID: 70786	✕ ↓
Worker 2	PID: 70787	✕ ↓
Worker 3	PID: 70788	✕ ↓
Worker 4	PID: 70790	✕ ↓
Worker 5	PID: 70792	✕ ↓
Worker 6	PID: 70794	✕ ↓

```

Shubhras-MacBook-Pro:sls-sample-app shubhrakar$ slc clusterctl #
master pid: 70784
worker count: 6
worker id 7: { pid: 70891 }
worker id 8: { pid: 70892 }
worker id 9: { pid: 70893 }
worker id 10: { pid: 70895 }
worker id 11: { pid: 70897 }
worker id 12: { pid: 70899 }
Shubhras-MacBook-Pro:sls-sample-app shubhrakar$ slc clusterctl set-size 2
supervisor size set to 2
Shubhras-MacBook-Pro:sls-sample-app shubhrakar$ supervisor stopped worker 12 (pid 70899)
supervisor worker id 12 (pid 70899) expected exit with 143
supervisor stopped worker 11 (pid 70897)
supervisor worker id 11 (pid 70897) expected exit with 143
supervisor stopped worker 10 (pid 70895)
supervisor worker id 10 (pid 70895) expected exit with 143
supervisor stopped worker 9 (pid 70893)
supervisor resized to 2
supervisor worker id 9 (pid 70893) expected exit with 143
supervisor resized to 2
slc clusterctl #
master pid: 70784
worker count: 2
worker id 7: { pid: 70891 }
worker id 8: { pid: 70892 }

```

The graphics above show the ability to scale up or down on demand using the StrongOps UI console or through the packaged command line utility. Additional processes can be added or removed from the cluster without having to restart the main application process.

## Hot Deploy & Rolling Restarts

This capability allows runtime changes to an active application process being managed by strong-supervisor. This includes any code changes made on the application files (for example: app.js) while the application is still running. When strong-supervisor is used

to restart the entire cluster, it sequentially restarts one process at a time within the cluster and similarly persists changes, one process at a time within the live cluster. allows the cluster to operate at full capacity minus one (1) process where changes are being applied, even in production. The code changes have to be applied using the terminal, but rolling restarts can be triggered using the console UI or the command line (CLI). Hot deployment is done automatically under the covers and is transparent to the end user.

## Fault Isolation and High Availability

Fault isolation is also provided by strong-supervisor. If a faulty deployment package having syntax or errors in application files is pushed to a set of live processes of a cluster, strong-supervisor will first deploy and try to restart all the processes in the cluster one by one. If the first process in sequence with the faulty package fails to restart and crashes, strong-supervisor will not attempt to deploy and restart the other processes in cluster, thus preventing the fault to spread across the full cluster. High availability within a clustered application is provided by the strong-supervisor's embedded [strong-cluster-control](#) module. If one or multiple processes in a managed cluster are terminated due to machine or human intervention, strong-supervisor will automatically try to restart the terminated Node application instance, albeit with a different process ID. Additionally, due to capability of dynamic scaling, additional processes get added to the cluster and get managed by the master without having to restart the main application process.

## Clustered State Sharing

When using a cluster, you have multiple processes with each handling requests. However, sometimes you need to keep state information locally across all the Node processes. This is when you use [Strong Store](#) for Cluster. Strong Store for Cluster enables you to share information across clustered Node processes in a key/value collection. The most common use is to implement sessions.

Transport Layer Security (TLS) provides encrypted streams using sockets. [Strong Cluster TLS Store](#) implements a TLS session store using Node's native cluster messaging to improve the performance of Node's TLS/HTTPS server running in a cluster. By adding hooks for the events 'newSession' and 'resumeSession' on the `tls.Server` object, Strong Cluster TLS Store enables clients to resume previous TLS sessions.

[Strong Cluster Connect Store](#) provides an easy way to use sessions in a clustered application. Each clustered worker is a separate Node process with its own memory space, however, Strong Cluster Connect Store enables you to store sessions in a single location with low-latency access from all members of a cluster.

[Socket IO Store](#) for Clusters is an implementation of socket.IO store using Node's native cluster messaging. It provides an easy solution for running a socket.IO server in a cluster. Key features include no dependencies on external services and the ability to use your version of socket.io.

## Load Balancing

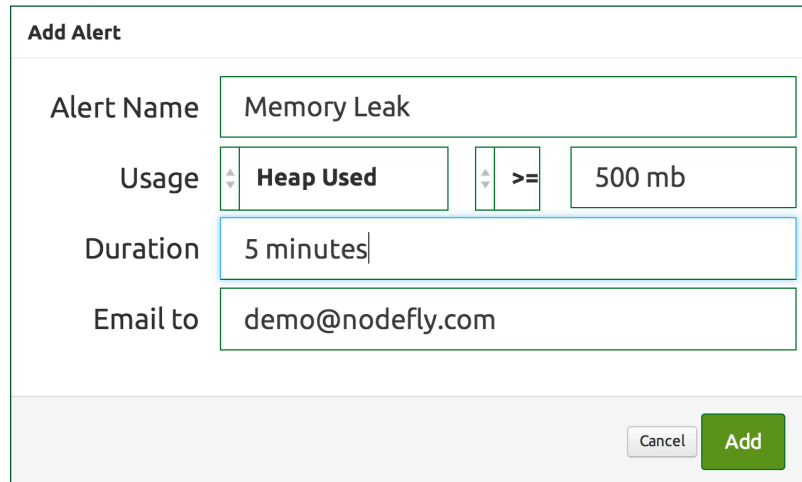
StrongLoop provides [Round-Robin-Load-Balancing](#) as part of the cluster management capabilities. This uses a master-child configuration and provides application level workload management capabilities.

## Reporting and Analytics

Reporting is a key ingredient of application monitoring and management solutions and the same applies for Node. Alert notifications help DevOps teams to quickly jump to investigate and triage outages or issues in production. Additionally, KPI metrics are often required for reporting purposes.

StrongOps helps organizations during this phase by enabling them get alerts on errors and performance problems, plus the ability to visualize the data in external reporting or analytics tools.

## Alerts

A screenshot of the 'Add Alert' form in StrongOps. The form has a title 'Add Alert' at the top. It contains four input fields: 'Alert Name' with the value 'Memory Leak', 'Usage' with a dropdown menu showing 'Heap Used', a comparison operator dropdown showing '>=', and a value field showing '500 mb', 'Duration' with the value '5 minutes', and 'Email to' with the value 'demo@nodefly.com'. At the bottom right, there are two buttons: 'Cancel' and 'Add'.

With StrongOps you can create alerts to have a notification sent when a particular metric matches a predefined condition. For example, you could create an “excessive CPU” alert to email the appropriate people when an app exceeded 75% CPU usage.

## Open Data API for Metrics

StrongOps provide performance data via the open data API for use by other reporting tools. The open data API is built on top of JSONP and enables you to pull metrics and graph data normally visualized in StrongOps. For more information on how the API works, visit the [official documentation](#).

```

{ "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],[1384547760,46.15592]],
      "rss": [[1384547521,82.8416],[1384547581,82.8416],[1384547700,82.8416],[1384547760,82.8416]],
      "v8gc": [[1384545240,31.98568291],[1384546980,32.71478691]]
    },
    "queue": {
      "blockfrequency": [[1384547521,0],[1384547581,0],[1384547640,0],[1384547700,0],[1384547760,0]],
      "blocktime": [[1384547521,0],[1384547581,0],[1384547640,0],[1384547700,0],[1384547760,0]]
    },
    "mq": {}
  },
  "end": 1384547760
}

```

Above diagram shows the output of API calls that provide metrics collected by StrongOps, as output to the calling system.

## Conclusion

In this paper we looked at some of the considerations you should have when developing a DevOps strategy for Node applications, including:

- Development
- Quality assurance and testing
- Performance tuning

- Monitoring
- Process management
- Scaling
- Reporting and analytics

We showed how many of the features of StrongOps can help you achieve some of your DevOps goals. Ready to try StrongOps for yourself? Visit the [StrongOps product page](#) to learn more.

## Resources

- StrongLoop [website](#)
- StrongLoop [technical blog](#)
- [StrongOps documentation](#)
- [Technical videos](#)

## About StrongLoop

StrongLoop is the leading contributor to Node.js v0.12. Launched in 2013 and based in Silicon Valley, StrongLoop was founded by engineers who have been contributing to Node.js since 2011. The company is funded by Ignition Partners and Shasta Ventures, plus advised by Marten Mickos, CEO of Eucalyptus.

Node.js is used to create “front edge” APIs that mobile applications use to connect to backend data. For developers creating these APIs, StrongLoop offers an API framework and mobile services such as push and geolocation that can be leveraged via iOS, Android and HTML5 SDKs with a variety of connectors including Oracle.

StrongLoop also offers the leading DevOps tooling for clustering, monitoring and optimizing Node applications. StrongLoop runs on all major operating systems and eight clouds including Amazon, Heroku, Red Hat’s OpenShift and Rackspace. For more information, visit <http://strongloop.com> or drop us a line: [callback@strongloop.com](mailto:callback@strongloop.com).