# LineRate logging, JSON and LogStash

**Brian Talley, 2014-06-11**

Data collections and data mining are big business all over the web. Whether you are trying to monetize the data or simply track data points for statistical analysis, data storage, formatting and display are essential. Most web-facing applications have logging capabilities built in and work pretty well for storing this data on direct- or network-attached storage. But when you have tens, hundreds and even thousands of applications logging data, storing, aggregating, parsing and, ultimately, visualizing this data can get pretty difficult. Some large web properties can generate 10's or even 100's of GB's of log data per day. One way to simplify this process is to log everything to a central point in the network. Think syslog on steroids.


Since all your traffic is passing through it anyway, the load balancer is a great place to do this. Not only can you record "normal" HTTP statistics, like request method and URI, but logging from the load balancer also allows you to include virtual server and real server data for each request.

In this article, I'll show you how you can gather various data points for HTTP requests using the LineRate proxy and the embedded Node.js scripting engine. You can then format and ship this data to any number of data/log collection services. In this example, I'm going to send JSON formatted data to logstash - "a tool for managing events and logs". (By default, logstash includes ElasticSearch for it's data store and the Kibana web interface for data visualization.)

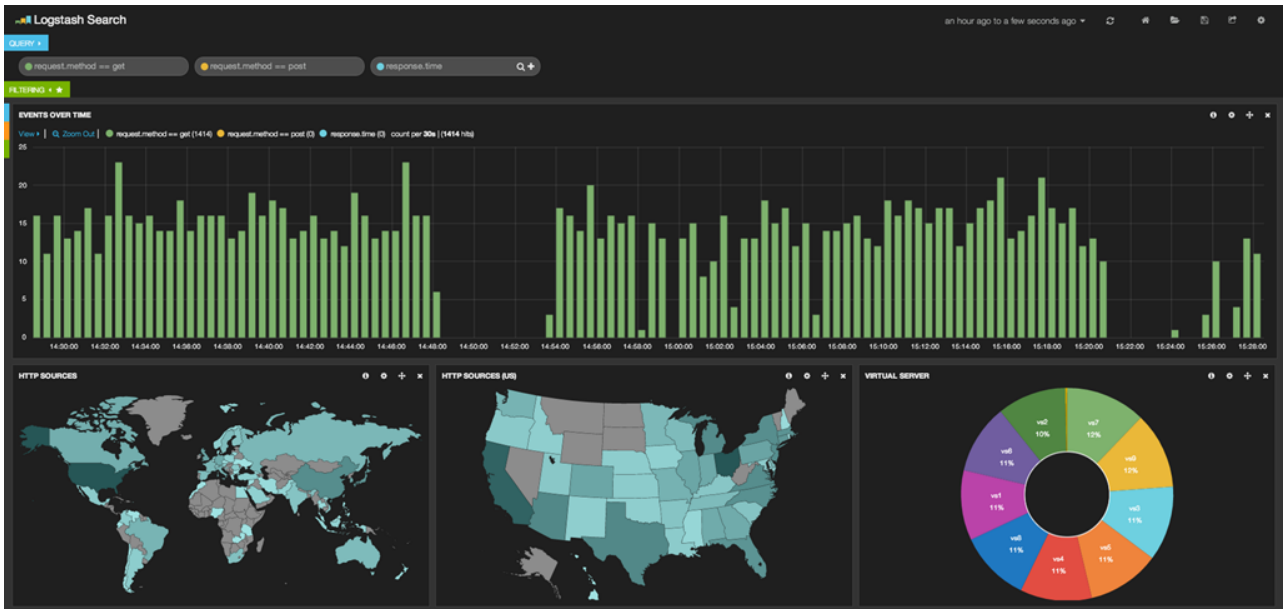
logstash is an open source project and installs easily on Linux. The logstash 10 minute walkthrough should get you started. We're going to configure logstash to ingest JSON formatted data by listening on a TCP port. On the LineRate side, we'll build a JSON object with the data we're interested in and use a TCP stream to transmit the data.

I'll use the TCP input in logstash to ingest the data and then the JSON filter to convert the incoming JSON messages to a logstash event. logstash adds a few fields to the data, but essentially leaves all the original JSON in it's original structure, so this filter is perfect if you're already working with JSON. Here's a simple logstash config file for this setup:

```
input {
  tcp {
        mode => server
        port => 9999
    }
}
filter {
    json {
        source => "message"
    }
    geoip {
        source => "[request][source_ip]"
    }
}
output {
    elasticsearch { host => localhost }
    stdout { codec => rubydebug }
}
```

You might notice that I also added the 'geoip' filter. This is another great built-in feature of logstash. I simply enable the filter and tell logstash which of my input data fields contains the IP address that I want geo data for and it takes care of the rest. It uses its built in GeoIP database (via the GeoLiteCity database from MaxMind) to get data about that IP and adds the data to the message. By default, this filter adds a lot of geo data to the message. You might want to trim some of the fields if it's more than you need. (For example, you might not want latitude and longitude.)

Here's a sample screenshot of logstash/kibana with data logged from a LineRate proxy:




Here's the Node.js script for LineRate.

```
'use strict';

var vsm = require('lrs/virtualServerModule');
var net = require('net');
var os = require('os');
var cm = require('connman');

// change these variables to match your env
var virtual_server = 'vs_http';
var logging_server = {
    'host': '10.190.5.134',
    'port': 9999
};

var hostname = os.hostname();
var pid = process.pid;
var sock;

function onData() {
    // noop
}

function onReset() {
    // noop
}

function onReconnect() {
    // noop
}

function onConnect(socket) {
```

```
    console.log("Socket connected, pid = ", process.pid);
    sock = socket;
}

function log_message(message) {
    try {
        sock.write(JSON.stringify(message) + '\r\n');
    }
    catch (ex) {
        // if you want an exception logged
        // when a write failure occurs, uncomment
        // the next line.  Note this could have
        // very negative effects if you are logging
        // at a high request rate

        //console.log(ex);
    }
}

cm.connect(logging_server.port, logging_server.host, onConnect, onReset, onData);

var processRequest = function(servReq, servResp, cliReq) {

    // object to store stats
    var message = {};

    // record request start time
    var time_start = process.hrtime();

    message['request'] = {
        'method': servReq.method,
        'url': servReq.url,
        'version': servReq.httpVersion,
        'source_ip': servReq.connection.remoteAddress,
        'source_port': servReq.connection.remotePort,
        'headers': servReq.headers
    };

    message['vip'] = {
        'virtual_ip': servReq.connection.address().address,
        'virtual_port': servReq.connection.address().port,
        'virtual_family': servReq.connection.address().family
    };

    message['lb'] = {
        'virtual_server': virtual_server,
        'hostname': hostname,
        'process_id': pid
    };

    servReq.on('response', function onResp(cliResp) {

        message['real'] = {
            'ip': cliResp.connection.remoteAddress,
            'port': cliResp.connection.remotePort
        };

        cliResp.bindHeaders(servResp);
```

```
        cliResp.pipe(servResp);

        // get total request time
        var time_diff = process.hrtime(time_start);
        //console.log('setting response time: ' + time_diff[1]);

        message['response'] = {
            'headers': cliResp.headers,
            'status_code': cliResp.statusCode,
            'version': cliResp.httpVersion,
            'time_s': time_diff[0] + (time_diff[1] / 1e9)
        };
        log_message(message);

    });

    /* continue with request processing, send the request to real-server */
    cliReq();
};

var createCallback = function(virtualServerObject) {
    console.log('Logging script installed on Virtual Server: ' +
         virtualServerObject.id);
    virtualServerObject.on('request',processRequest);
};

vsm.on('exist', virtual_server, createCallback);
```

Finally, here's a sample message in raw JSON format retrieved from the elasticsearch datastore:

```
{
  "_index": "logstash-2014.11.03",
  "_type": "logs",
  "_id": "PZTeM0KWRfiju9qUifmDhQ",
  "_score": null,
  "_source": {
    "message": "(redacted)",
    "@version": "1",
    "@timestamp": "2014-11-03T22:32:22.930Z",
    "host": "192.168.88.155:59625",
    "geoip": {
      "area_code": 614,
      "city_name": "Columbus",
      "continent_code": "NA",
      "country_code2": "US",
      "country_code3": "USA",
      "country_name": "United States",
      "dma_code": 535, "ip": "172.16.87.1",
      "latitude": 39.96119999999999,
      "location": [ -82.9988, 39.96119999999999 ],
      "longitude": -82.9988,
      "postal_code": "43218",
      "real_region_name": "Ohio",
      "region_name": "OH",
      "timezone": "America/New_York"
    },
    "request": {
```

```
      "method": "GET",
      "url": "/test/page",
      "version": "1.1",
      "source_ip": "172.16.87.1",
      "source_port": 59721,
      "headers": {
        "User-Agent": "curl/7.30.0",
        "Connection": "close",
        "Accept": "*/*",
        "Host": "lrosvm",
        "X-Fake-Header": "fake-header-value"
      }
    },
    "vip": {
      "virtual_ip": "172.16.87.153",
      "virtual_port": 80,
      "virtual_family": "IPv4"
    },
    "lb": {
      "virtual_server": "vs_http",
      "hostname": "lros01",
      "process_id": 8177
    },
    "real": {
      "ip": "192.168.233.1",
      "port": 15000
    },
    "response": {
      "headers": {
        "Content-Length": "36",
        "Date": "Mon, 03 Nov 2014 22:32:22 GMT",
        "Content-Type": "text/plain"
      },
      "status_code": 200,
      "version": "1.1",
      "time_s": 0.00395572
    },
    "_source": {}
  },
  "sort": [
    1415053942930,
    1415053942930
  ]
}
```

Please leave a comment or reach out to us with any questions or suggestions and if you're not a LineRate user yet, remember you can try it out for free.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com