Streamlining Infrastructure Monitoring and Metrics in IT-DB-IMS

August 2015

Author:
Charles Callum Newey

Supervisors:
Giacomo Tenaglia
Artur Wiecek

CERN openlab Summer Student Report

Project Specification

Basic Systems Monitoring

This is now done with a CERN-made custom collector which sends data in the form of "notifications" via Apache Flume to HDFS for long-term storage. The notifications are also sent to ElasticSearch and displayed with Kibana (à la Splunk). Due to limitations of the architecture, the system data is collected every 5 minutes which is not ideal.

The idea is to implement a solution which allows more fine-grained sampling of system metrics. Possible ideas are OpenTSDB (to leverage the existing Hadoop infrastructure) or prometheus.io, which should be simpler to setup but it only scales out by sharding. OpenTSDB initially seems like a more promising solution, so investigate the various collection and display alternatives.

Logs Management and Centralisation.

This is now done only for `syslog` with Apache Flume - shipping to HDFS (and kept "forever") and to Elasticsearch/Kibana, with a 1 month retention time. There are two issues: flexibility of the collection process, and authorisation.

On the flexibility side, collected messages need to be split into different fields before being stored in HDFS/Elasticsearch, in order to ease the data mining process. Logstash and `grok` are potentially promising solutions.

On the authorisation side, the idea here is to specifically target the Weblogic installations in order to expose to clients their application logs in a convenient way, and as we host very different applications with different confidentiality levels (amazing what can be found in some application logs!) we need to put an authorisation layer on top of HDFS and ElasticSearch (Kibana just being JS querying ElasticSearch directly). For this there are some methods that could be implemented on HDFS, and for ElasticSearch there is a FOSS plugin to be checked. This part would probably involve setting up an ElasticSearch cluster first.

Abstract

There are a number of problems with the current monitoring infrastructure in IT-DB which currently make it difficult to diagnose certain system issues. These include lack of support for certain log formats, latency in messages, and inflexibility of architecture. Several changes to the log monitoring service architecture were evaluated over the period of several weeks, focusing on improving performance, verbosity, and reducing latency. Additionally, several changes to the metric monitoring architecture were evaluated, including deploying a new time-series database to test performance and latency. There were also a number of extra tests conducted on third-party visualisation dashboards and Elasticsearch security software. The proposed log monitoring architecture was found to be useful, and will be the subject of further development and evaluation. The proposed metric monitoring architecture was also found to be useful, but needs further investigation to decide if it is a significant enough improvement over the current architecture to be of value. There were also several software packages found to be unsuitable or unstable, and these are not used in the proposed system architectures. The evaluation was largely successful as much of the work was either found to be unsuitable or will be the subject of further development.

Table of Contents

1 Introduction

1.1 Introduction to Monitoring

Monitoring (within the context of information technology and system administration) describes a set of processes and activities related to the supervision and long-term management of operating systems, services, and server health. There are several smaller topics that fit neatly within the remit of IT monitoring, and the most relevant of these will be summarised below.

1.1.1 Log Monitoring

Log monitoring is the process of collecting, analysing and reacting to information stored within log files. This includes analysis and long-term storage of various log files, containing different sorts of information – ranging from basic system messages (i.e. `/var/log/messages`) to various application logs (Apache HTTPD, Oracle WebLogic, and other services).

To extract useful, actionable information from large amounts of log files, it is often necessary to search, filter, and visualise data. A group of machines may produce a very large amount of data in a given day. For example, at CERN, a group of approximately 50 machines may produce in excess of 25 million log events per day, translating to over 12GB of data every 24 hours. When one considers that this rate of incoming data is from only 50 machines (and that CERN manages over 10,000 virtual machines), one can begin to see the scale of the problem.

It is in these situations where "big data" tools begin to become useful. Analysing gigabytes (or indeed, terabytes) of unannotated textual data is an arduous task, and this is where tools such as ELK (Elasticsearch, Logstash, and Kibana) begin to show their value - Logstash can extract fields from log files and transform them into semantically useful JSON documents, and Elasticsearch is a high-performance NoSQL data store and search engine, allowing JSON documents to be searched and queried extremely quickly. Kibana simply serves as a useful interface to visually search and analyse data stored inside Elasticsearch.

1.1.2 System Metrics

Metrics are various statistics related to the current state of a given system. For example, common metrics to monitor on production services are statistics like the following; CPU temperature, amount of free memory, latency between nodes (using ICMP pings), load averages, and much more.

Storing these metrics on a long-term basis allows system administrators to analyse service changes or downtime, and also to ensure the consistent operation of various aspects of the machine. This gives administrators the capability to, for example, check that the performance characteristics of a given server stay consistent after an upgrade.

1.1.3 Alerts and Notifications

Alerts and notifications are essentially human-readable messages sent to responsible parties, containing actionable information that has been automatically extracted from the metric-gathering and log monitoring processes. For example, a notification may be sent to a system administrator if a particular machine is reaching its maximum allocated disk quota.

Notifications can take many forms, but are most commonly sent in the form of SMS messages, email messages, or perhaps something more unusual - for example, messages sent into a Slack channel or SNOW tickets.

1.2 Current Technology Stack and Architecture

1.2.1 Apache Flume

Currently, Apache Flume is used as a log transport mechanism to the Elasticsearch cluster. There currently exist a number of Puppet recipes at CERN to deploy and configure Apache Flume on a client node, with a Puppet module that can also configure Flume "sinks" (a "sink" is just Apache Flume terminology for a data store or endpoint) to send to Elasticsearch.

1.2.2 Elasticsearch and Kibana

Elasticsearch is used as the standard data store for almost all log data at CERN. Elasticsearch is simply an advanced, high-performance text search engine and NoSQL data store. It's often used in the context of monitoring in conjunction with Logstash (a semantic log-parsing Java service) and Kibana (an HTML5 visualisation and data exploration dashboard). Logstash and Kibana are both technologies that are explored during the project.

1.2.3 LEMON

For metric collection and reporting, CERN uses a piece of software known as LEMON. The LEMON daemon runs on each node that is monitored, and extracts a given set of metrics (called "sensors") and report to a central server (usually an Oracle database or an Elasticsearch cluster), as well as reporting some events to a log file.

1.3 Problems with Current Monitoring Architecture

1.3.1 Metric and Log Longevity

Currently in CERN's ELK monitoring infrastructure, log data and metric events are only held within Elasticsearch for 30 days. This is long enough to be useful when diagnosing short-term issues or seeing short-term trends, but isn't quite sufficient for longer-term trend monitoring or historic data analysis. For example, if a system administrator wanted to compare time-series data over the course of 3 months to diagnose a degenerative performance issue introduced by a given component in the operating system (perhaps a bug in the kernel), there would currently be no way of doing that.

1.3.2 Granularity and Artificial Latency

Another problem with CERN's monitoring infrastructure is the latency for the appearance of certain metrics in Elasticsearch. Currently, there is a five minute delay between metrics being collected by LEMON and the data being shipped to a database and displayed on a dashboard. This delay is obviously not optimal, as a lot of significant events can happen in five minutes; particularly if 10,000 machines are being monitored.

If events are only collected every five minutes, this will introduce an artificial latency when indexing metrics into Elasticsearch - if Logstash indexes events into Elasticsearch using the timestamp that events were *received*, rather than the timestamp that events were *generated*, then valuable time information will be lost, and it will be more difficult to retrospectively analyse an event.

1.3.3 Lack of WebLogic Support

Oracle WebLogic are enterprise-class business application servers used at CERN for several purposes - largely for purposes such as running HR tools and other business web applications. Currently CERN's ELK stack doesn't parse WebLogic logs, meaning that WebLogic application exceptions can occasionally go unnoticed, and application slowdowns can't be visualised and diagnosed as easily. This is a potential problem as there are a large number of WebLogic nodes at CERN, totalling perhaps several hundred machines.

WebLogic application logs are also extremely verbose, producing tens to hundreds of megabytes of logs per node per day. This makes WebLogic logs extremely difficult to analyse effectively, and also makes it difficult to diagnose problems with the WebLogic software. The ability to diagnose bugs in the WebLogic software is useful to the CERN because some of the IT-DB administrators are in close contact with Oracle developers, regularly contacting them to request bug fixes and performance enhancements.

1.4 Improvements to Current Technology Stack and Architecture

As part of the project, various changes to the monitoring architecture were proposed with the help of reducing or eliminating some of the shortcomings with the existing architecture. In the following section of this document, the changes will be explored and summarised.

1.4.1 Logstash

1.4.1.1 Addition of New Patterns

As mentioned previously, the Logstash instances currently used at CERN do not support messages sent from Oracle WebLogic servers. This presents a problem to the administrators in the IT-DB section because they have a large number of WebLogic server instances, running a fairly diverse array of web applications.

Part of the project was to develop a logging dashboard for clusters of WebLogic machines, so a large portion of time was spent trying to accurately parse the logs from WebLogic machines using Logstash. This was not a simple task due to the logging architecture within the WebLogic applications – and this presented a number of challenges while developing the necessary configuration directives and regular expressions to parse WebLogic messages. Part of the problem was that the formats of the log messages were extremelyinconsistent - six or seven different log message formats are commonly used in each log file, as well as several different localised time formats. Trying to extract this information in a useful and semantic manner posed a problem.

Another difficulty with parsing WebLogic logs was that several processes wrote to the same log file. This made it difficult to trace the origin of each individual message, and as such caused some confusion when messages from unrelated parts of the WebLogic subsystem started appearing in a single log file. Additionally to this, it was difficult to parse multiline log messages (Java stack traces) with Logstash. Because Logstash (by default) will process each individual log message with multiple worker threads (and each message isassumed to be a single line - as is the standard), stack traces would not get processed correctly because multiple threads would try and process what was effectively a single message. To mitigate this undesired effect, Logstash has to be limited to using a single worker thread per log file. This means that the input rate of messages in Logstash is somewhat limited.

1.4.1.2 Logstash-Forwarder

Logstash-Forwarder is an important component in the experimental logging architecture within IT-DB. Instead of a traditional ELK stack where an instance of Logstash is deployed on every client node, the experimental CERN architecture uses Logstash-Forwarder as a client to ship logs to a central Logstash instance (using transport encrypted with SSL/TLS).

This difference is important for a number of reasons. First of all, the logs are not processed on the client. This means that there is very little CPU and memory overhead related to the log monitoring processes on each client node - only a single lightweight process is run on each client to ship logs to the central Logstash server. The lower resource usage means that log monitoring can now take place on smaller instances than before - Logstash is a fairly resource-intensive Java application and can regularly occupy 500MB of physical memory, whereas Logstash-Forwarder consistently occupies less than 80MB.

The second important distinction to make is that there is now a separation between log *shipping* and log *processing*. Making changes to the Logstash configuration (for example, adding support for new log formats, adding tags, or adding new data transformations for specific log categories) is considerably faster, simpler, and more reliable. Instead of distributing the configuration changes across the *n* monitoring clients, configuration changes now only need to be pushed to a single server, where they will be active immediately.

The Lumberjack protocol is another important aspect of the new architecture. Typically, Logstash transports log data to an Elasticsearch cluster - and this transport is unencrypted. However, the transport used by Logstash-Forwarder is, by default, encrypted and authenticated with SSL/TLS certificates. This ensures that log files containing potentially sensitive application data cannot be accidentally leaked across the internal network, and is less susceptible to man-in-the-middle attacks.

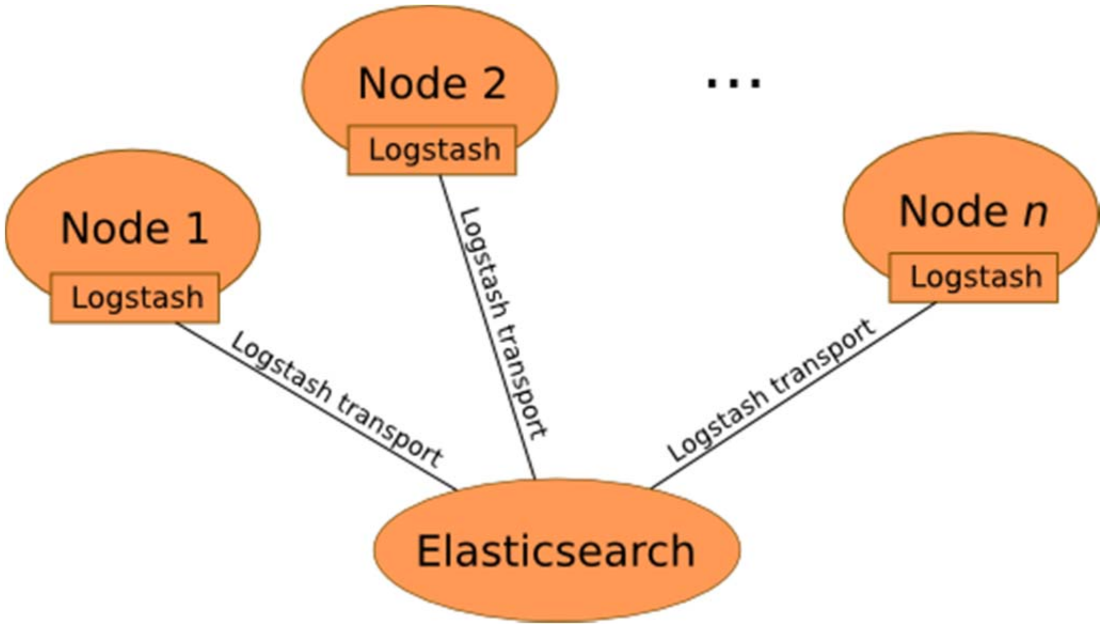


Figure 1: A simplified representation of a traditional ELK architecture

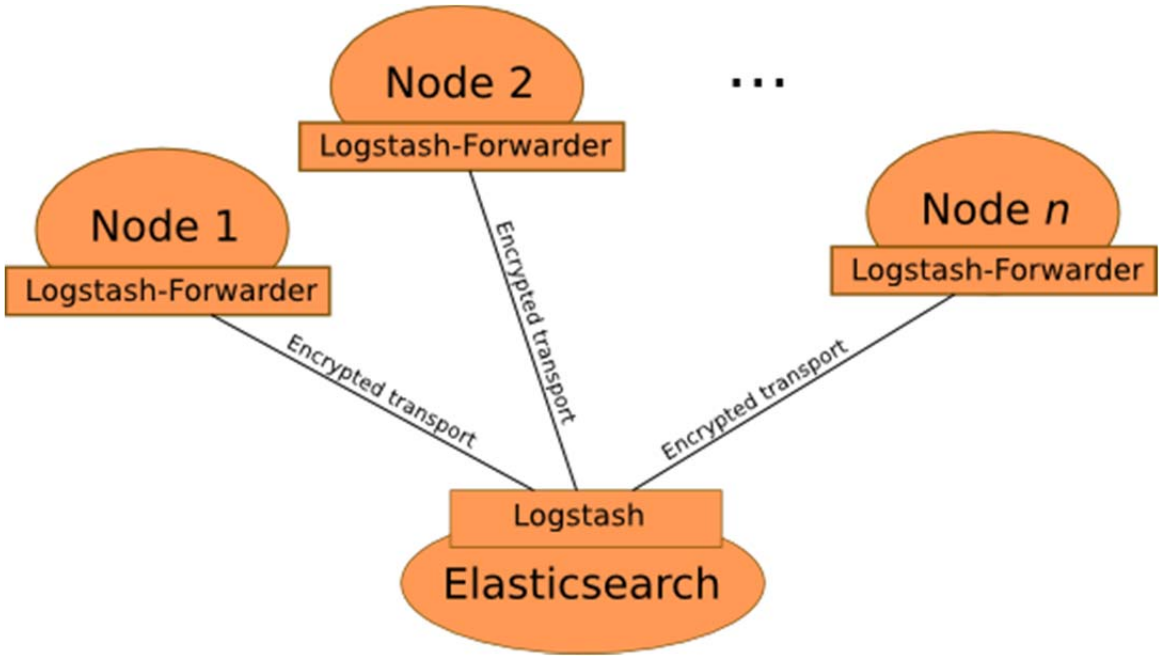


Figure 2: A simplified representation of the proposed (more efficient) ELK architecture

1.4.2 Elasticsearch, Kibana, and Search-Guard

The Elasticsearch installation is a fairly standard setup - essentially just a single large Elasticsearch node (because the logging architecture is still being tested, there is no need for a clustered setup) with several plugins installed. The key plugins are Elasticsearch-HQ (a simple web interface to interact with the REST API within Elasticsearch) and Kibana (a polished web-based visualisation dashboard for Elasticsearch data).

Another plugin was also trialled during the project - one of the key problems with Elasticsearch (and therefore with the existing log-monitoring architecture) is that communication with Elasticsearch is not access-controlled by default. This is a problem that Search-Guard attempts to solve. Search-Guard is a plugin for Elasticsearch that adds several useful security features to otherwise-unsecured Elasticsearch clusters. Amongst other features, it provides the ability to add access control lists to Elasticsearch clusters - restricting access to authenticated users. Search-Guard also includes support for a number of common authentication protocols including LDAP and Kerberos.

1.4.2.1 Problems with Search-Guard

Unfortunately, as became eminently clear when deploying and configuring Search-Guard, it is still very much beta software. Firstly, it is still an unstable piece of software - application errors are not gracefully handled, simply terminating the Elasticsearch instance when an exception is generated. Secondly, the plugin required running a very new version of Elasticsearch (at the time of writing, 1.6 or newer), which is problematic in a production environment as it hasn't been proven to work reliably over a long period of time, as some older releases have been.

Additionally, there were a number of application-critical bugs within the plugin that caused any Elasticsearch to exit - regularly and without a graceful shutdown, meaning that data integrity became a concern. Finally, it also became clear that Kerberos authentication was only properly supported in a version of the plugin that was designed for a much newer version of Elasticsearch that (at the time of writing) only had a beta release. This was obviously not suitable for a production environment.

For these reasons, it was decided that Search-Guard was too new a project to safely use in a production environment and that a more traditional alternative (such as a simple authentication layer using `nginx` as a reverse proxy) would be more stable.

2 Improving Granularity of Systems Monitoring

One of the key problems previously mentioned with the current metric infrastructure at CERN is the latency between receiving data from LEMON and indexing it into Elasticsearch - this delay can be up to five minutes. This latency can delay diagnosis of critical system issues and reduce visibility of problems in a running system.

2.1 OpenTSDB

OpenTSDB is a time-series database and visualisation layer built on top of Apache's Hadoop and HBase (a columnar NoSQL database that uses the Hadoop Distributed File System). As OpenTSDB is designed specifically with the objective of storing and searching very large amounts of time-series data, it is ideal for using as a data store for large-scale system metrics.

2.1.1 Configuring OpenTSDB with HBase and Zookeeper

Configuring OpenTSDB was surprisingly straightforward, even with Puppet. The relevant database schema had to be created in HBase, and the configuration file had to be templated with the Zookeeper endpoint. It was a very simple task to configure OpenTSDB with Zookeeper, although due to limitations with Puppet, it's not possible to automate database creation in HBase (at least, not in an idempotent manner).

2.1.2 Scalability and Performance of OpenTSDB

With regards to performing at scale, OpenTSDB works extremely well. It has been tested with an increasing number of client nodes, and has been shown to perform extremely well (approx. 5% CPU usage) with over 50 client nodes per OpenTSDB server. This is mostly because OpenTSDB doesn't actually do much processing - it essentially acts as a consistent and convenient API layer on top of HBase to provide a convenient method for data transport.

2.1.3 Data Integrity and Replication in HBase and HDFS

Data stored in Hadoop's distributed filesystem (HDFS) is, by default, replicated 3 times on separate data nodes. Not only is this good archival practice, but it allows OpenTSDB and Zookeeper to structure their queries in a way such that, if a large amount of data is requested, the data can be transferred from multiple nodes. This is a useful tool in ensuring high performance from an OpenTSDB installation.

2.2 Metric Collection

Once OpenTSDB was deployed and verified to work correctly, it was pertinent to consider which software would be used to send client metrics to it. A small amount of research led to the conclusion that a Python package called `tcollector` would be an appropriate choice.

2.2.1 TCollector

Upon first inspection, `tcollector` seemed like an ideal candidate for a metric collection agent. It collected a wide range of metrics at sensible intervals, and was very simple to configure - even if the default method of installation was a little unusual (there was no official system package, or even a `Makefile` to install the application). This meant that distributing the `tcollector` code using Puppet was a challenge.

2.2.1.1 Packaging and Distributing TCollector

After several attempts trying to distribute the `tcollector` code using Puppet and Git, it became clear that a simpler method of distributing the software was necessary. At CERN, there is a Koji build system for building and distributing RPM packages through `yum` repos. After some experimentation and some time reading the documentation for Koji and the RPM-building toolchain, it became possible to build and distribute `tcollector` using `yum` repositories. This made it considerably easier to deploy `tcollector` onto client nodes using Puppet.

2.2.1.2 Resource Usage

While `tcollector` was being tested as a viable candidate for metric collection, Artur brought it to my attention that `tcollector` forked large numbers of processes (one process for each metric), and incidentally consumed large amounts of resident memory in each system it was deployed on. This was obviously far from an ideal situation, as a metric collection agent should use very few resources - a collection agent only needs to consume statistics (that the system has already generated) and ship them over the network. Shortly after the discovery of `tcollector`'s poor performance, other collection agents were explored.

2.2.2 SCollector

After some research, it became apparent that `scollector` was a viable candidate to replace `tcollector`. An open-source project developed by staff at StackExchange, `scollector` aims to be a comprehensive and resource-efficient metric collection agent. It is more efficient partly due to better software design, and partly due to the choice of language - `scollector` is built in Go, a systems programming language by Google that compiles down to native machine code.

2.2.2.1 Packaging and Distributing SCollector with Koji and Puppet

Distributing `scollector` was a little different to `tcollector`. Because the `scollector` code compiles directly to an architecture-specific binary, some changes had to be made to the RPM Specfile to ensure that it was only available on the `x86_64` architecture. This made little impact on the availability of the `scollector` binary in the CERN `yum` repositories, because the vast majority of CERN machines are *x86_64* commodity servers.

2.2.2.2 Bespoke "Init" Script

Because `scollector` is somewhat bespoke software (and is not commonly used outside of large organisations), the source code is simply distributed as a Go library. This means that, even when compiled, `scollector` does not ship with an `init` script. Because of this limitation a custom `init` script had to be written so that `scollector` could be run as a system service, and could be managed properly with Puppet.

2.2.3 Metrics Collected

As there are several areas of each system that it's important to monitor, `scollector` and `tcollector` both support a wide range of metrics. Various metrics are collected from each client system, including a number of useful statistics from the `/proc` filesystem - for example, load average (1, 5, 15), free memory, free disk space, and so on. Both `scollector` and `tcollector` support adding custom metrics (the structure of each software package is modular to allow for this), and both packages can monitor more obscure services (such as Elasticsearch) by default.

2.3 Visualisation

2.3.1 OpenTSDB

The default web interface in OpenTSDB is governed by a lightweight server process that makes use of the common Unix plotting utility `gnuplot` to generate charts. This has the useful effect of being extremely fast, and the charts are usually of a high enough quality to provide useful information. However, the user interface of OpenTSDB isn't particularly user-friendly and the charts lack interactivity, which is something that many other HTML5 visualisation dashboards can provide.

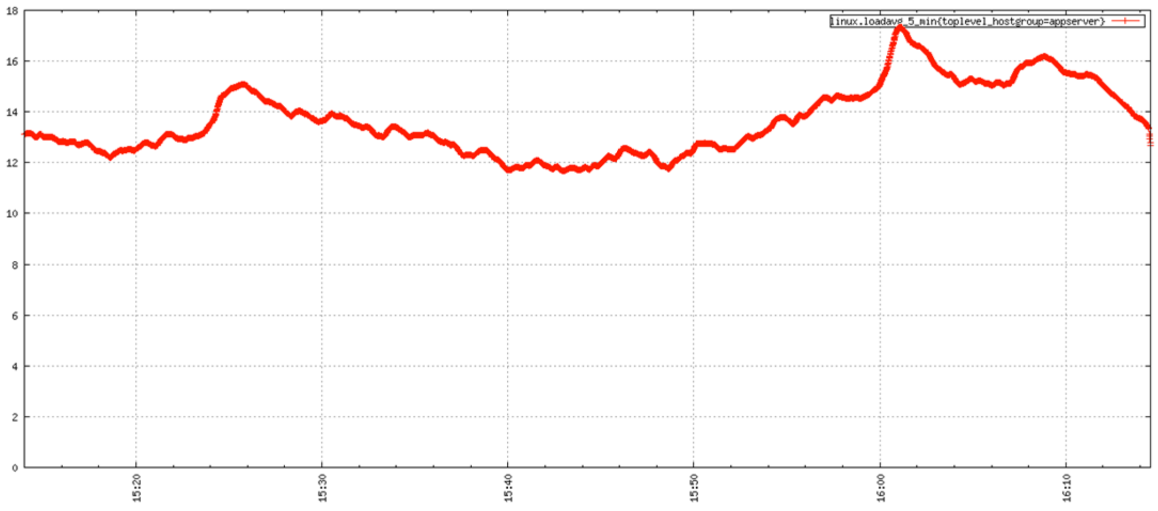


Figure 3: A sample OpenTSDB chart, showing load average for a Puppet hostgroup

2.3.2 Grafana

Grafana is an HTML5 data visualisation dashboard that has been developed as a fork of the original Kibana code base. Originally designed for Graphite, Grafana has also evolved support for a number of other data sources, including OpenTSDB. The visualisations that Grafana produces are very effective, and most allow the user to interactively explore the data in OpenTSDB. Unfortunately, Grafana also requires an additional Java process to be run on the server - and Grafana requires a large amount of resources to run, so it wasn't a viable solution to deploy on the OpenTSDB server.

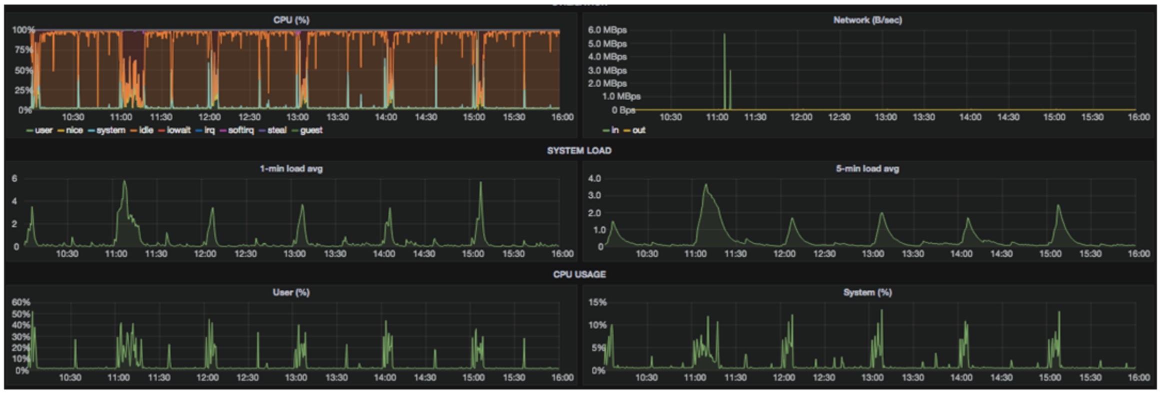


Figure 4: A sample Grafana dashboard showing various metrics

2.3.3 Performance

Regarding performance, both Grafana and the OpenTSDB interface perform extremely well, even when generating comparatively large plots. With OpenTSDB, a plot with approximately 40,000 points (data from 1 metric, from approx. 50 machines over an hour) takes about 800ms to generate. Grafana is a little different as the graphs are interactive (HTML5 and JavaScript), but it seems to perform reasonably well for smaller datasets. Grafana's performance did degrade when plotting extremely large datasets, but to some extent any performance issues can be mitigated by downsampling the time-series data.

3 Reproducibility

To ensure that the proposed logging infrastructure remains easy to understand and easy to replicate, the entire infrastructure has been written as Puppet code. Not only is Puppet easy to read (almost to the point of being self-documenting), but it is easy to distribute at CERN due to the prevalence of Puppet.

The Puppet classes are simple parameterised Puppet classes - simply include the classes and provide the relevant variables for each one, and the required software packages will be deployed on the specified machines. Configuration is also handled automatically - application configuration files are template with the relevant information provided inside the Puppet classes.

3.1 SCollector

As mentioned previously, the `scollector` metric agent has been packaged as an RPM and is now available in CERN's `yum` repositories - it is slowly transitioning through CERN's package testing hierarchy, and will hopefully be accepted into CERN's production environment in the near future. After being tested during the course of several weeks on a representative cluster of machines, it has proven to be stable, lightweight and resilient – performing well and coping gracefully with network issues and service downtime.

4 WebLogic Kibana Dashboards

One of the key problems that was introduced early on in the project was the lack of WebLogic monitoring support in the current ELK installation. As a consequence of this, one of the key tasks during the project was to add support for monitoring and shipping WebLogic log files. What initially sounded like a simple task actually posed several significant challenges due to the flawed logging architecture within the WebLogic application server.

4.1 Logstash Patterns

To be able to parse a log file effectively, Logstash must know the format of each log file - the position of each field, the data type of each field, and so on. This process is necessary to allow Elasticsearch to index and search each log entry in a useful and meaningful way. The process of configuring Logstash to parse log files involves writing a large number of regular expressions to be able to parse each entry - a challenge made more difficult by the inconsistent and complex nature of the logging infrastructure within the WebLogic application.

4.1.1 Inconsistent Log Message Formats

There were several log files that had to be monitored, with each file containing messages from different parts of the system. Unfortunately in addition to this, multiple application components were also writing to the same file, and all with different log formats. In all, there were in excess of 7 different log message formats in some files, and this posed a difficult problem when it came to extracting useful information from each log file.

4.1.2 Inconsistent Timestamp Formats

The timestamp formats were also wildly inconsistent - some files used 5 or more different formats, each with idiosynchrasies - some with random characters mistakenly left in the timestamp format string from development, some with localised time zone strings (i.e. "CEST", not the standard "+0200"), some with halfday markers instead of 24-hour clocks (i.e. "4:30:20 PM", not the standard "16:30:20"). This caused a number of difficulties with the Java time parsing library (Joda) within Logstash, as it did not recognise all of these items. Special Logstash "mutate" filters had to be developed to replace parts of incompatible timestamps, just so that they could be parsed properly.

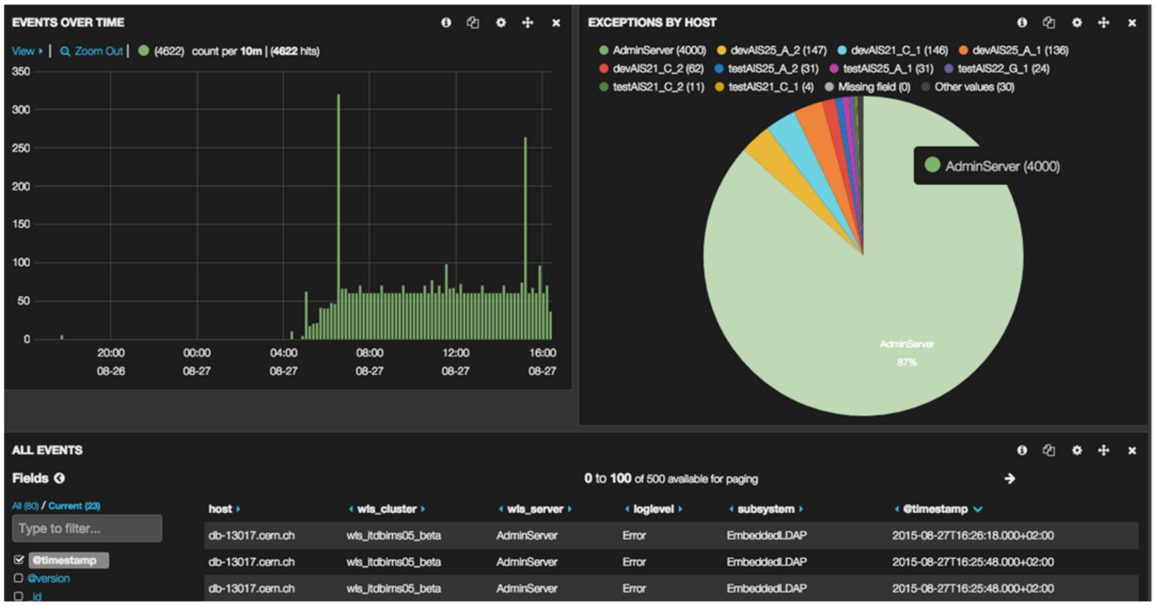
4.1.3 Multiline Messages

Java applications often flush full stack traces to log files, and this was also the case with WebLogic. This is problematic when parsing with Logstash for a number of reasons. Firstly, trying to write a Logstash configuration for multiline log messages is a challenge itself - extra Logstash filters have to be employed so that Logstash recognises the beginning of each individual message. Luckily, this was reasonably straightforward as Oracle prepends each new log message with an easily-recognisable string (in this case, "####").

However, this wasn't the only difficulty with multiline log messages. Multithreading is also a consideration that must be made when processing multiline log messages. Due to the line-oriented nature of the vast majority of log files (i.e. one message on each line), most log analysis tools are designed to process each line as an individual message. For this reason, Logstash usually processes each line as an individual task (often using a separate thread). This means that if log messages are permitted to span multiple lines (as is the case with WebLogic and many other Java applications), Logstash must be limited to processing these log files using only a single thread. This limits the potential message throughput of a given Logstash instance.

4.2 Sample WebLogic Dashboard

As well as adding WebLogic log support to Logstash, another task was to create an analytics dashboard to pinpoint and visualise exceptions and errors occurring within WebLogic applications. Due to the clustered nature of the WebLogic application server (and the way that the workload is distributed among client nodes), often an application error can occur on one of several nodes within a cluster. The new Kibana dashboard allows administrators to search for exceptions, filter by time and date, log message content, subsystem (the WebLogic component that generated the given error), and more. This is a useful tool in allowing system administrators to communicate more effectively with Oracle developers when getting application support or diagnosing bugs.


EVENTS OVER TIME
View ▸ | Zoom Out | ● (4622) count per 10m | (4622 hits)
350
300
250
200
150
100
50
0
20:00 08-26
00:00 08-27
04:00 08-27
08:00 08-27
12:00 08-27
16:00 08-27
EXCEPTIONS BY HOST
● AdminServer (4000) ● devAIS25_A_2 (147) ● devAIS21_C_1 (146) ● devAIS25_A_1 (136)
● devAIS21_C_2 (62) ● testAIS25_A_2 (31) ● testAIS25_A_1 (31) ● testAIS22_G_1 (24)
● testAIS21_C_2 (11) ● testAIS21_C_1 (4) ● Missing field (0) ● Other values (30)
AdminServer (4000)
AdminServer
87%
ALL EVENTS
Fields
All (80) / Current (23)
Type to filter...
@timestamp
@version
_id
0 to 100 of 500 available for paging

host	wls_cluster	wls_server	loglevel	subsystem	@timestamp
db-13017.cern.ch	wls_itdbims05_beta	AdminServer	Error	EmbeddedLDAP	2015-08-27T16:26:18.000+02:00
db-13017.cern.ch	wls_itdbims05_beta	AdminServer	Error	EmbeddedLDAP	2015-08-27T16:25:48.000+02:00
db-13017.cern.ch	wls_itdbims05_beta	AdminServer	Error	EmbeddedLDAP	2015-08-27T16:25:48.000+02:00

Figure 5: A screenshot of the sample WebLogic exceptions dashboard showing the various visualisations

5 Overall Project Summary

5.1 Conclusions and Impact

To conclude, the project was largely successful. The existing logging infrastructure had several improvements made to it - Logstash-Forwarder is now used in place of Logstash to ship logs, meaning that log monitoring is more resource-efficient, and therefore can also run on less powerful machines. WebLogic logs are now able to be parsed, indexed into Elasticsearch and analysed with Kibana - which means that IT administrators can now pinpoint and analyse the causes of exceptions within the WebLogic core application server, which will help improve their communication with Oracle developers. As CERN uses Puppet for automated deployment, all of the infrastructure is written in Puppet and is easy to reproduce.

A number of additional project goals have been achieved, including the evaluation of several experimental software tools. OpenTSDB and `scollector` have proved to be reliable, useful and resource-efficient tools for visualising and archiving metric data. Less fortunately, the Search-Guard Elasticsearch plugin and Grafana have been shown to be less suitable for usage at CERN currently.

5.2 Future work

Regarding the future plan for the work undertaken in this project, the testing of the proposed ELK infrastructure will continue. An intra-departmental review has showed positive feedback for the new infrastructure, and the WebLogic dashboards were also praised as being useful. Regarding the future of OpenTSDB, an intra-departmental review showed that it was indeed useful, but that it may not provide a great deal of value over the current metric infrastructure (i.e. LEMON monitoring), and so evaluation will continue.