PROCEEDINGS OF SCIENCE

Efficient Management of System Logs using a Cloud

Radoslav Bodó*,
CESNET z.s.p.o.,Zikova 4, 160 00 Praha 6, Czech Republic and
University of West Bohemia,Univerzitní 8, 306 14 Pilsen, Czech Republic
E-mail: bodik@civ.zcu.cz

Daniel Kouřil,
CESNET z.s.p.o.,Zikova 4, 160 00 Praha 6, Czech Republic and
Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic
E-mail: kouril@ics.muni.cz

Applications and operating systems describe important events using *logs*. The log record contain information that are very important to detect and fix issues in the infrastructure. Logs are especially crucial for security operations. In the paper we describe our experience with deployment of a central logging service. We also describe the solutions we deployed to process large volumes of data collected on the central server. These solutions operate in a cloud, which make it possible to us to tune the performance just according to current needs and loads of the logs.

The International Symposium on Grids and Clouds (ISGC) 2013,
March 17-22, 2013
Academia Sinica, Taipei, Taiwan

*Speaker.

PoS(ISGC 2013)009

1. Introduction

Centralized management of logs produced by computer systems is an important piece of operations. Having log records collected at a single point simplifies handling of various incidents since the operator has all the information collected centrally. For example a common task is to detect what steps a particular user has performed recently, which is needed during an investigation of a security incident. Another example is ongoing checks of the logs, e.g. to detect an issue immediately when it appears. Without centralized log management in place, all relevant machines in the infrastructure need to perform these checks independently, which is hard to establish and maintain. Being able to evaluate logs quickly and work with them on the fly contributes to solid system management and makes it more efficient.

Despite the basics of such deployment is well understood, we identified several areas that either are not covered at all or whose scope exceeds the possibilities of current tools. In the paper we will present the deployment of a central logging service in the Czech national grid infrastructure provider – METACentrum. We will describe the infrastructure to collect logs from machines that are distributed all over the country to a single place. Such an arrangement differs from usual deployments of remote syslog servers and we describe our experience which might be interesting for other resource infrastructure providers, too.

Another principal problem concerns with the way how data is processed once it is collected and stored on the central service. The traditional way is setting up series of filters that detect some known patterns in the incoming data. This mechanism can only be employed for patterns that are known in advance and can only be applied to new data. Sometimes it is necessary to analyze the data collected. There are several common tools that can be utilized for that, like grep and other standard Unix commands. However, processing of a large amount of data with these tools is very intensive, which makes them unusable for interactive work.

Based on these limitations we decided to explore alternative ways how logged data can be processed and examined. We combined together several open source solutions and built up an infrastructure to index log records with high throughput and manipulate them in an easy and quick way. The solution is based on an internal cloud that runs the indexing tools, which continually process received logs. Being provided in a cloud, the indexing service can be re-scaled on demand, based on current amount of data and the requirements of the operators. On top of the pre-processed results we run visualization tools that are used to access and manipulated the data via web interfaces.

More details on the deployment and utilization of the centrally managed logs as well as details on particular configurations can be found in our technical report [1] that is an extended version of this paper.

2. Collecting Logs

Applications generate log records describing important events in their processing. In order to log an event the application use either an own mechanisms to store it in a file or database or use a dedicated library. For the latter case the syslog POSIX calls are often utilized, which usually pass the log records to the syslog daemon running on the system. In the default configuration the

syslog daemon stores log messages on a disk but it can also be configured to send them to a remote server over a network, and since the log messages may carry sensitive information, it is reasonable to ensure that they are send through protected channel.

To provide transport security of the log messages, the TLS protocol can be used[2], which is supported by the main open-source implementations. However, we were deploying the solution in the environment of project METACentrum[3] which relies heavily on the Kerberos security system. Deploying a parallel PKI infrastructure would have been an unacceptable overhead, so we required to secure the messaging using Kerberos. Another requirement stem from the fact that the infrastructure is distributed all over the whole country, which increases chances of network outages. Therefore, a basic requirement for the whole messaging was that it provides sufficient fault-tolerance so records do not get lost when they cannot be delivered for a while.

Based on these requirements we selected `rsyslog` as the implementation of the syslog messaging and server. It has a plugin-based architecture supporting various ways of authentication and channel protection. One of the plugins supports GSSAPI, which would allow us to leverage the existing security infrastructure of METACentrum. Early at the beginning of our project we found out that the module from the stable release of `rsyslog` (5.8 at the moment of writing) was crashing from time to time and initiated a fix, which was introduced in 5.8.x and onward. However, even after the fix the plugin is not an ideal solution for a huge distributed environment. Having done quite extensive stress testing, we found out that the client side of the plugin is not able to reconnect correctly on server failures, which yields a huge traffic triggered by TCP reconnection attempts. Therefore, we have implemented a new GSSAPI module (`omgssapi`), which is based on the code of output plugins for the 5.8 family which merged the GSSAPI functionality from the existing module. The new module has been contributed to the community and is available from [4].

It is easy to instruct the rsyslog daemon on a node to pass every logs it receives to a remote host. However, in order to provide a robust delivery, a few additional options are needed and must be tuned properly. Unfortunatelly those options are not mentioned by the basic rsyslog documentation, that might be a big source of problems for a new adopters. By default every rsyslog action invoked to process a message is handled directly in the context of the main rsyslog message queue and the corresponding threads. If the action is long running (more than few milliseconds) or it is failing and retrying (possibly forever), it could block processing of the other messages. To address this issue, it's advised to use one of the available action queue implementations to de-couple the processing of the action to a separate queue/thread and to process actions asynchronously so the main processing is not blocked.

When using disk assisted queue implementation for TCP or GSSAPI (or other actions), the buffering can lead to abnormal resource consumption (memory or disk space), which is not desired in production environment. In order to prevent this, the queue must be restricted to use only defined maximum space. It is also inevitable to configure additional enqueueing timeout on the queue so it starts dropping messages that cannot be inserted and protect rsyslog from blocking indefinitely. Exact details and reasoning are provided in the technical report[1].

The configuration of a server collecting logs is quite straightforward and is also well described in the rsyslog documentation.

3. Log processing

Once all system logs are stored on one place, it is possible to analyze them. For example, the Torque batch system logs information about resource utilization (CPU, memory) of the jobs, which can be used to extract information about the actual utilization of the resources. This information can be communicated to the users whose jobs exceed granted capacities and possibly used to warn them before applying hard enforcement of the limits. The central storage makes it much easier to collect the information about activities of a particular user in the whole environment. Being able to access this information on demand it is crucial for efficient function of both user support and security operational staff.

The common way how logged data is processed nowadays is to use general text handling tools like `grep`, `awk` or `perl`. While this approach works for simple cases, the task is getting complicated by the ever increasing size of the logs (50GB per month and more) and also by the fact that subsequent queries requires grepping the whole set all over again. Additional analytic tasks requires an efficient aggregation queries, scanning through the whole dataset and also some persistent storage for internal state and metadata.

The traditional solution to handle large volumes of data is to utilize relational databases. However, before taking the straightforward approach we decided to consider a few things:

The size In the grid environment it might not be sufficient to rely on a single node database system. A scaling solution must be used by design. Probably an advanced partitioning or sharding features of traditional RDBMS like MySQL or PostgreSQL can be used.

The structure In the traditional RDBMS it is necessary to know in advance and work with the fixed structure of the data stored in the tables. Every change of the structure must be reflected not just by application using that data, but also by the corresponding database model and configuration. This requirement does not quite fit to the way how the logs are processed since it is often not known in advance what kind of data will be processed. For instance logs generated by Apache HTTPD differ a lot from logs produced by Kerberos KDC.

Those characteristics led us further into exploring new emerging technologies like text indexing and NoSQL databases, in particular we focused on ElasticSearch and MongoDB. Both solutions are designed to work in a cloud-like environment and both employ a structureless approach to work with the data – JSON documents. The first aspect is very likely to fit into grid computing environment possibly to address size and scaling issues. The second is likely to be more than helpful for fast prototyping or research development of the applications working with logs.

3.1 ElasticSearch

ElasticSearch[5] is a full-text engine to index and search text data, which is built on the top of the Lucene library[6]. ElasticSearch was designed for a cloud, allowing for dynamic adding and removing of nodes, and thus a distributing the workload, based on current requirements. Every instance of the ElasticSearch node can play one or more roles according to it's configuration (*client*, *master*, *data*) which is used for forming a cluster with layered architecture. In order to search the

indexed data users can utilize available interfaces (REST, native, thrift) or a web GUI provided by Kibana[7].

However, current interfaces do not support encryption or authentication so anyone who is able to connect to the data nodes will be able to work with the data stored in the index. To address this limitation it is advised to make sure the cluster of ElasticSearch nodes is private and only available to authorized people or services. In METACentrum we leverage the virtualization framework to setup virtual clusters overlaying physical resources. Using the framework it is possible to instantiate virtual workernodes connected by a dedicated private VLAN. The VLAN itself is accessible from all main grid sites through the CESNET backbone network so the cluster members can be randomly allocated on all major sites[8, 9]. Proxy servers are used to provide access to the cloud services running inside such network.

Establishing a cloud with multiple ElasticSearch nodes is as easy as starting an ElasticSearch instance on every node of the cloud. Utilizing the autodiscovery feature and automatic sharding, the nodes will establish themselves the topology and are ready to accept data to index and answer queries. No additional configuration of the nodes is necessary, especially no data schema needs to be specified. The data is just inserted and ElasticSearch handles it properly.

3.2 Turning logs into structured documents

Logs are not just text lines, but should rather be considered as structured data. There is at least a timestamp in every log along with some text data, and the data itself has a given structure[10, 11]. The structure of the log messages varies from product to product. ES is able to accept and parse JSON structures and index them so that queries can be made based on the provided fields. Using created index provides much better experience and response time in comparsion with the common tools like `grep`.

There are several ways how logs can be passed to ES for processing. The simplest way is to use the ElasticSearch plugins available from the latest development release of rsyslog. However, in order to keep a better control over the parsing process we decided to use Logstash[12] for several reasons:

Grok – a library which is designed to parse text lines with regular expressions and turn them into structured data. Regular expressions are written in enhanced language which allows pattern reuse[13].

Logstash flexible architecture – the concept of event/message processing pipeline consists from the set of input, filtering and output plugins working in the separate threads and interconnected with the sized queue. Logstash supports a large scale of input and outputs which can be used to create flexible event processing services.

Logstash provides an entry point used to insert the syslog data to ElasticSearch. The basic arrangement is depicted in Fig. 1. There are ElasticSearch data nodes running within the virtual cloud and special services running on the proxy host are used to push (Logstash parser) and query (Kibana along with ElasticSearch client) the data into/from the cloud.

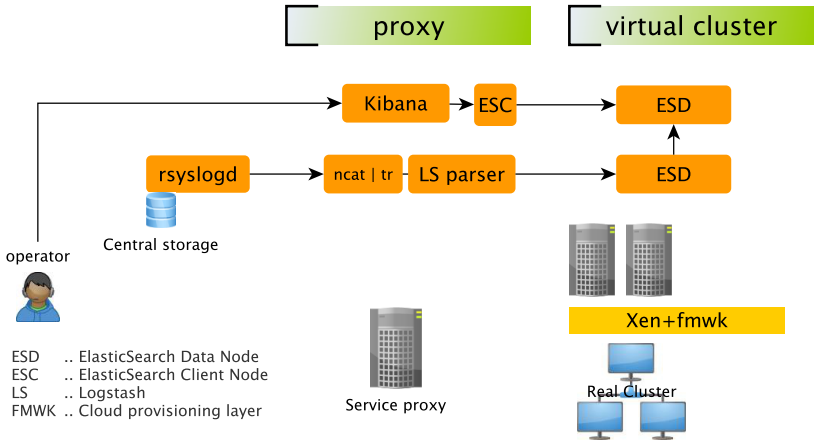

Figure 1: Proxy parser

3.3 Performance tuning

Establishment of the whole infrastructure is quite easy and can be done quickly. While the infrastructure works well for small datasets, it has performance limits, which need to be addressed before bigger data (>1GB) is processed .

When more indexing or searching power is needed than what the existing deployment provides, a larger ElasticSearch cloud can be instantiated, but in that case the single LS parser becomes a bottleneck. That could be resolved by moving the parser from the proxy node to the cloud and running multiple instances to increase parallelism. For that, some form of data spooling or other messaging queue is required, so indexing services running in the cloud can be configured to pop data from a single place. For example the Redis[14] server can be used in this manner. Redis is populated with logs sent from the syslog server and provides the data to parsers running in the cloud.

The default behavior of most Logstash input and output plugins is to process only one message/event at a time. That may be fine for a low traffic environment but when bootstrapping the ElasticSearch cluster, indexing historical dataset or recovering after total cluster failure, it is necessary to enhance the uploading speed by introducing support for batch processing at several places.

At Redis as a message queue, batching can be done using pipelining of redis commands or using scripts in the LUA language. Scripts can result in less network utilization while pipelining leads in a smoother client requests interleaving on the server side. The latest Logstash implementation uses LUA scripting.

There are two main options for the final delivery of structured data from Logstash to ES cluster. One can either use the native API plugin or an HTTP output plugin but neither is ideal. The native API plugin does not support any bulk indexing API yet. The HTTP plugin feeds the data by separate

POST requests yielding significant overhead for standard syslog messages[1]. In order to improve the performance of bulk indexing we have produced a customized output plugin that implements an enhanced batching system with synchronized internal buffer queue and time based flush.

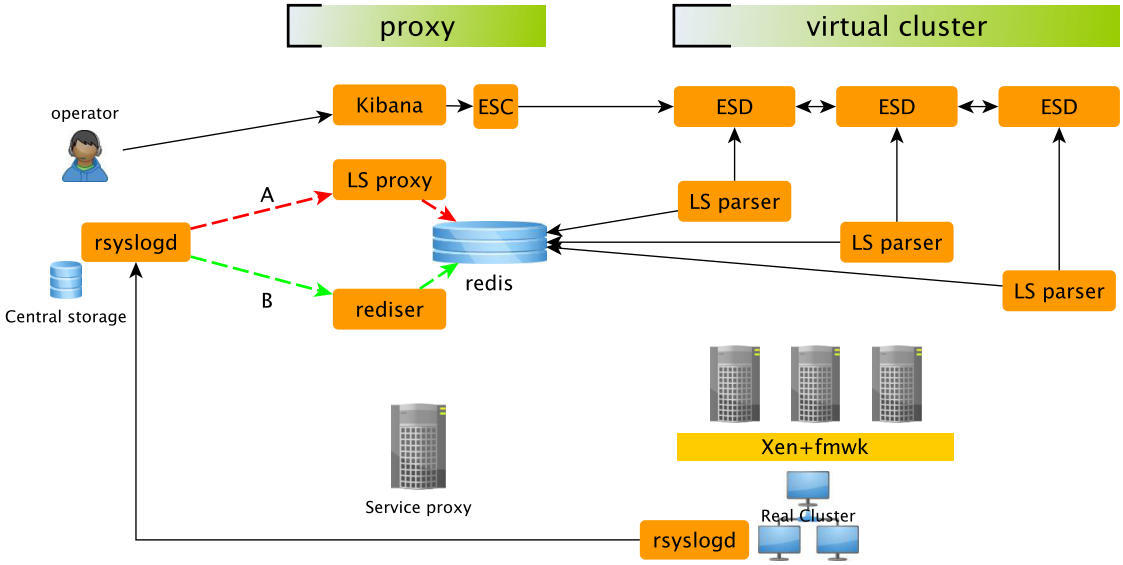

Figure 2: Cloud parser

Another bottleneck was revealed in the way how the Redis server is populated. By default, Logstash adds a lot of metadata information to every log record providing more information about the origin of the data. In our case, however, all data is arriving from the syslog server and it all relates to a syslog messages. Therefore the metadata only generates unnecessary overhead that increases the requirements for the processing of the records.

In order to produce a feeder for the Redis server, which does not add overhead, we created a separate component – `rediser`. Rediser is built on the Logstash source code and it makes possible to push incoming messages into redis queue without any modification.

Whole arrangement of a cloud tuned to better performance can be seen in Fig. 2.

3.4 The speed of ES upload and search

The speed of upload generally depends on many factors, such as the size of grokked data and final documents, batch/flush size of input and output processing, etc.

A real dataset from January 2013 in the size of 105GB (∼ 800M events) was used for main testing. The time for indexing is approximately 4 hours using 8 ElasticSearch data nodes (ESD) with 16 shared indexers (LS parser installed on all ESD nodes) and speed averages about 50 000 messages per second (eps). During regular operations an incoming rate 100 - 5000 eps was observed. A single or two node cluster is able to handle those rates.

Search speed was evaluated on basic level and is limited by IO throughput, but even when using a small cluster it is much faster than the conventional tools. While a grep query over the data

[1] version 1.1.10

on the local filesystem required approximately 18 minutes, the same query over indexed data in ElasticSearch took only 35 seconds.

4. Advanced data analysis

Logs contain a lot of interesting information that can reveal security incidents or attacks very quickly. The logs can also be correlated with other sources of information provide combined knowledge, which can point to a particular issue. While ElasticSearch makes it possible to quickly search the logs, it is not suitable to provide advanced analysis and datamining features. Therefore we decided to utilize a database to store parts of the logs and provide additional processing of them. We choose the MongoDB[15] nosql database since we aggregate data from multiple sources, and those vary in the structure of provided information.

The database is populated from the central syslog server in a way that is similar to feeding data to ElasticSearch. Unlike the feed to ElasticSearch however, only a fraction of data is stored in MongoDB. For a pilot solution we focus mainly on information logged by the SSH servers and we used the Logstash and grok to parse authentication logs.

Statistical analysis and other analytic tasks can be done on the index/collection produced by such a grok filter. For example, the aggregation framework of MongoDB and its map-reduce techniques can be used to generate time-based hierarchical aggregations. Using these techniques it is simple to group events originated in the same period of time. The refined data can be further used for reporting or help with investigation of security events. All users and/or services interacting with the monitored domain can be profiled with the collected data. For instance it is easy to detect SSH attacks by correlating the number of unsuccessful login attempts.

It is also possible to include information from other sources. Our solution currently consumes data from IDS Warden[16] operated by CESNET-CERTS within the CESNET2 network. Warden itself is a messaging platform used for sharing data about security incidents and attacks that have been detected by the CESNET member institutions and also by external partners. Data received from Warden is stored in MongoDB and used to detect attempts to access from malicious addresses, etc.

Logstash mongodb output does not use bulk inserts, neither we tried to develop this feature. Also only a subset of sharding/clustering capabilities[2] of MongoDB was used during our experimentations, mainly because MongoDB does not yet include any autodiscovery or autoconfiguration features.

A real dataset from January 2013 in the size of 20GB ($\sim$ 33M events) was used for testing. Time for inserting all data is approximately 8 hours using one MongoDB server with 20 shards and one dedicated indexer with six LS instances. The average time of all incremental aggregations during normal operations is 10 seconds.

5. Deployment and scaling

The described system has been in production for about a year and in our current deployment we

[2] single host, 1 shard per CPU, 0 replica set

have 90% coverage of worker and service nodes connected to the central logging service, yielding approximately 750 machines. The amount of stored logs is approximately 100 GB per month.

Although we have not done extensive scaling testing and evaluation, the proposed system scales because of the design of the used technologies. Databases scales because of the sharding/partitioning stored data and log processing components scales due to spooling of a incoming data and paralellization of the processing. During our tests we were able to upload test dataset (100 GB) in 8 hours using 4 ElasticSearch nodes with 8 shared parsers, while using 8 nodes and 16 parsers it took 4 hours.

The overall schema is depicted in Fig. 3.

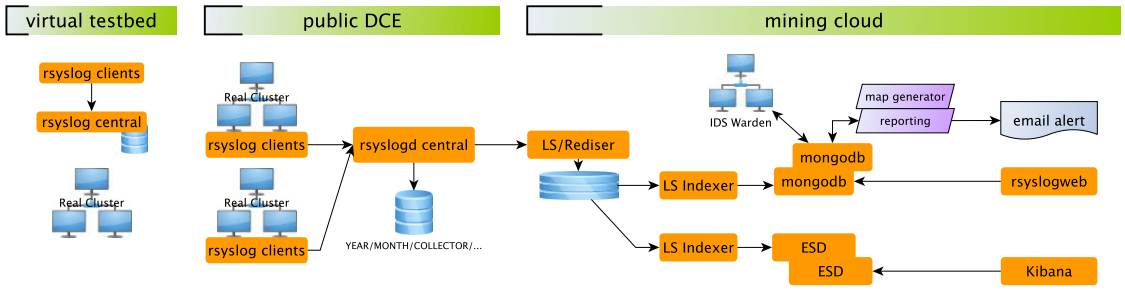

Figure 3: Overall schema of mining architecture

6. Conclusions

In this paper we have presented a system designed for collection and analysis of a logs from distributed environment. The system is based on open source projects and is backed up by cloud solutions provided within the METACentrum. Whole solution is designed to ease daily operations and security tasks and due to its cloud nature it is capable of scaling up and also down depending on actual needs.

All custom components and enhancements developed on the top of the used tools are published and available to the public domain in [4]. To ease with adoption or testing of the described tools, we created and published a virtual machine image with all necessary components[17].

The system was developed mainly towards intrusion detection and incident response usecases. Experience gathered so far shows that it is much easier to collect all information with the new set of services. The operator can access relevant information in 10 minutes using the presented indexing and searching services in contrast of using traditional log handling processors like grep or perl, which would take seriously much more time.

Acknowledgments

The work has been supported by the research programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005). We highly appreciate the access to computing and storage facilities provided by the National Grid Infrastructure METACentrum, co-funded by the programme

We also want to thank to the user groups of rsyslog (`rsyslog@lists.adiscon.com`) and Logstash (`irc.freenode.org#logstash`) for their consultancy support provided during our project.

References

[1] Radoslav Bodó, Daniel Kouřil, Jiří Sitera, Miloš Mulač, Pavel Vondruška. "Crunching Logs for Fun and Profit". CESNET Technical Report. `http://home.zcu.cz/~bodik/metasw/doc-harvesting-logs/harvesting-logs.pdf`

[2] F. Miao, Y. Ma, J. Salowey: "Transport Layer Security (TLS) Transport Mapping for Syslog". IETF RFC 5425. 2009.

[3] `http://www.metacentrum.cz`

[4] `http://home.zcu.cz/~bodik/metasw/doc-harvesting-logs/pubdata`

[5] `http://www.elasticssearch.org/`

[6] Shay Banon. "Road to a Distributed Search Engine". `http://www.elasticsearch.org/videos/2011/08/09/road-to-a-distributed-searchengine-berlinbuzzwords.html`

[7] `http://www.kibana.org/`

[8] Václav Novák, Pavel Šmrha, Josef Verich. "Deployment of CESNET2+ E2E Services in 2007". *Networking Studies II, Selected technical Reports*. 2008. `http://www.cesnet.cz/vyzkum-a-vyvoj/dosazene-vysledky/networking-studies/networking-studies-2008/`

[9] David Antoš, Luděk Matyska, Petr Holub, Jiří Sitera. "VirtCloud: Virtualizing Network for Grid Environments". *In Advanced Information Networking and Applications—AINA 2009*. 2009.

[10] Common Event Expression: Unified Event language for Interoperability `http://cee.mitre.org/`

[11] Jordan Sissel. "logging, logstash and other things". *PuppetConf 2012*. `http://www.youtube.com/watch?v=RuUFnog29M4`

[12] `http://www.logstash.net`

[13] `http://code.google.com/p/semicomplete/wiki/Grok`

[14] `http://www.redis.io`

[15] `http://www.mongodb.org/`

[16] `https://warden.cesnet.cz/`

[17] `http://home.zcu.cz/~bodik/metasw/egiesbtf`