Enterprise Logging

DAVID LANG

David Lang is a Staff IT Engineer at Intuit, where he has spent more than a decade working in the Security Department for the Banking Division. He was introduced to Linux in 1993 and has been making his living with Linux since 1996. He is an Amateur Extra Class Radio Operator and served on the communications staff of the Civil Air Patrol California Wing, where his duties included managing the statewide digital wireless network. He was awarded the 2012 Chuck Yerkes award for his participation on various open source mailing lists.
david@lang.hm

When the topic of logging comes up, logs are generally recognized to be useful and that having a centralized log system is "industry best practice," and it's even required by most regulatory oversight plans (PCI, HIPAA). But figuring out how to get started in setting up a good logging system is hard, especially if you are already a good size organization when the topic is raised. If you start off by talking to vendors, getting quotes in the seven figure range is easy. This article is an introduction to logging, outlining an inexpensive architecture that can scale up to large log volumes, and providing pointers to a few basic tools to quickly and cheaply get value out of the logs beyond just satisfying audit requirements. Future articles will dive deeper into specific aspects of logging.

Benefits of an Enterprise Logging Plan

Getting started with logging in an enterprise can be as much a political/management issue over the effort and equipment involved as it is a technical issue of deciding what to do, so it's worth starting the discussion by reviewing the basic business benefits.

Logs record what happened. This seems like a trivial statement, but it's easy to get confused and think that logs mean more. Many things can go wrong, and having logs available helps you figure out what so that you can decide how to recover and prevent it from happening again. Examples of problems that you may need to investigate are misbehaving software, outside attacks, insider tampering, hitting system performance limits, and hardware failures (disk/memory/network errors).

Logs let you figure out how frequently things have happened, and this information can be used for utilization reports and capacity planning. Logs can also be analyzed to produce reports that show user behavior , which can then be used for marketing, product development, and detecting "odd" behavior that may indicate attacks. Logs can satisfy audit requirements by indicating who did what and when they did it (for both internal and external users).

Logs are invaluable for monitoring. Nothing can replace what the apps report about their own operations. If an app logs a message at 3:02 a.m. "unable to create file X No space left on device," saying that there's no problem does you no good because Nagios reported lots of disk space available at 3:00 and 3:10.

Collect Log Messages in a Single, Centralized Infrastructure

You most care about logs when something is (or has gone) wrong on the box where they were generated. Having a copy of the logs elsewhere lets you still see the logs. With cloud computing, this is even more critical than in a normal datacenter because a system is far more likely to go down, and when it does, you may never be able to get at its file system again.

By combining all the logs, you gain the ability to see what's happening across systems, to offload the log analysis from the systems that are serving your users, and to implement your tools and policies consistently across the enterprise. Protecting the logs from tampering if they're in one place rather than on every system is far easier.

Most compliance programs (PCI, HIPAA, etc.) require that you collect your logs in some central location. They don't say why, but the underlying reasons boil down to the advantages mentioned above.

You Should Try to Gather ALL Possible Log Messages

Because your logs are a record of what happened on your systems at some time in the past, you are usually not going to have a chance to tweak the logs to support the current problem you are dealing with. Some logs are more important than others, but you can always throw away or ignore logs that you have gathered, whereas you cannot go back in time and collect something that you didn't gather.

You should start with the premise that you will gather every log generated by every device, system, and application and only trim back if you find that you cannot support this. You cannot and should not process every log message the same way. Today's system performance is such that everyone except the largest companies can gather their logs into a single feed at a surprisingly low cost. Analyzing logs can be very expensive, so you will want to filter the logs as they go into your analysis tools, but different types of analysis will want different logs, so start off planning to gather everything.

Getting Started

Once you have decided to build an Enterprise-wide centralized logging system, you must determine the requirements you need it to satisfy.

Suggested Requirements for Enterprise Logging System

Vendor-Neutral Infrastructure

A good logging system will end up being used by just about every part of your organization. Any system you deploy is going to need to be changed at some point. If you build your logging infrastructure around a single vendor, changing it will be extremely painful. If you build it around standards, you can switch out portions of it at a time. While you are in the middle of migrating, you may not be able to take advantage of some features that only exist in one software package, but this will just temporarily degrade the system, not split it into two parallel systems.

Gather/Deliver the Logs in Near-Real Time

Many uses of logs require that you act on the logs shortly after they are generated. Any scheme that gathers logs nightly or hourly will not work for those uses, but if you gather the logs in near-real time you can support all the uses that will work with the batched gathering.

Run All Systems on the Same Time Zone

Running all your systems on UTC time is best, but even if you just pick the time zone of your main datacenter or office and use that everywhere, you are far better off than if each datacenter has systems running in its local time zone.

In theory this isn't a problem because all timestamps should include time zone information, but in practice, time zone information is frequently dropped; having timestamps from different time zones will confuse analysis of logs (including manual analysis).

The reason UTC is better than local time is that when rolling logs, storing them with filenames that have the timestamp as part of the filename is common; backwards adjustments due to daylight savings will cause you to overwrite and lose log files.

Fix Malformed Logs

Many devices (for example, Cisco Routers) have errors in logs that they send out. Fixing these errors early in the logging infrastructure makes it much easier to make use of the logs.

Add/Correct Log Metadata

Examples of metadata that can be useful to add/fix in log messages are timestamps, sources, and the office a log comes from. If you are in an enterprise large enough to have hostnames and IP addresses reused in different areas (e.g., think of how many workers who are telecommuting from home offices will be using 192.168.1.x IP addresses), adding additional information to the log message to be able to differentiate the duplicates can be extremely valuable.

No Modification Is Possible on Network Equipment and Appliances

This is less a requirement than a recognition of the reality that you cannot change how some devices send logs, so any scheme that requires that you run specific software on the system that's generating the log message cannot work as an enterprise-wide approach, no matter how well it works in a narrower deployment.

Minimize Configuration, Non-Default Software, and Load on the Systems Generating the Logs

While logs are valuable, if logging or administration of logging interferes significantly with the primary purpose of a device, odds are that the logging is going to suffer. The more work you have to do on each system, the higher the odds that the work isn't going to happen consistently, and you will end up with a gap in your logs that you are not aware of. Knowing that you are not receiving something that you would like to get is hard.

"Best Effort" Delivery of Logs

When you first think of the question "under what conditions is it OK to lose a log message," the normal reaction is "never." The problem with this is that the alternative to losing a log message when something goes wrong is to have the system stop. So the real question you must ask is, "Is this log message so critical that I would rather have the process/system stop working than have any possibility of losing the message?"

The answer to this is almost always "No, but I really would like to avoid losing logs if I can"; the rest of this article assumes that this is the case. There are ways to use modern logging daemons to deal with the ultra-reliable logging requirement (what I call "audit grade" logs), but it complicates the system and has horrible effects on performance (I have run tests in which I have measured greater than 1000x difference between "audit grade" and "best effort" performance).

Syslog, the De Facto Standard for Log Processing

The traditional UNIX tool for logs is syslog. Any log processing tool that has any pretension of being a general purpose tool is able to handle syslog messages. This makes syslog an obvious starting point; however, syslog has a poor reputation as a serious log tool because the versions of syslog that were the default on UNIX systems for the first couple of decades of syslog's existence have had a combination of ultra-safe and ultra-unsafe defaults that have limited log rates from tens to low hundreds of logs per second, truncated messages at 1k characters, and either blocked system operation or silently dropped logs beyond these rates. Additionally, filtering in traditional syslog was complex and dependent on the originator of the log messages properly tagging each message; however, current logging daemons bear about as much resemblance to the traditional syslog that Eric Allman created as a quick hack for dealing with Sendmail logs as the cars in a Barrett-Jackson auction have with the cars that were on a Ford dealer's lot in the heyday of the Model T. Most people, including many who deal with logs, do not realize that this has changed. There are now several additional logging implementations available for use, all of which are drastic improvements in performance and capability compared to the traditional syslog software, while still retaining software and network compatibility with traditional syslog. Since 2007, most Linux distros have switched to rsyslog as their default syslog daemon, and the rate of change over the past five years is staggering. Red Hat Enterprise 5.x ships with rsyslog3.22.2, but rsyslog 7.4 rolled out in June 2013. Additionally, syslog-ng, nxlog, and logstash are all free tools to consider if you dislike rsyslog. (Commercial syslog daemons, including a commercial version of syslog-ng, are also available.) Both rsyslog and syslog-ng now can handle more than one million logs per second, and all of these tools support a wide range of filtering and communication options. Combined with the fact that these all support the traditional syslog protocols means that you can choose whichever one you want, and switch from one to the other on your core infrastructure without having to change anything on your systems that are generating the logs.

As a logging protocol, syslog has the (dis)advantage that historically it has been poorly defined. Syslog has been around since the '80s with an RFC written for it in 2001 (and a follow-up in 2009), but the reality remains that you can throw just about anything at syslog and it will handle it in some form. This leads to the natural result that a lot of equipment (including from top name vendors) and software is generating syslog messages that don't comply with any RFC, but the modern logging daemons are all flexible enough to be able to deal with the messages in a (relatively) sane manner. This great flexibility means that syslog is easy to get data into, and can deal with just about anything.

A recent development in the syslog world is the support across the many different logging daemons for JSON-structured logs. While this is primarily being driven by people who dream that all logs will be formatted to some standard, making it trivial for any application to parse and understand the log contents, this capability is absolutely wonderful for enterprise logging even if no such standard ever emerges [1]. This is because it makes it possible to take the original unstructured syslog message, wrap it in JSON and then add additional fields to hold information to the log message that is sent upstream. This maintains the separation between the original message and the new fields, allowing you to hand the original message to an analysis tool that doesn't know about the new fields or format. This added information can include, for example, the environment the log was generated in, so that you can alert differently depending on whether the log was generated from a development machine or a production machine without needing separate logging systems.

Architecture

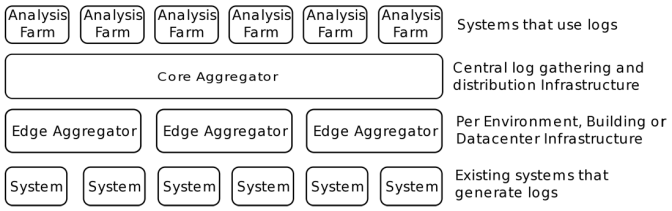

Figure 1: The best design for an enterprise logging infrastructure is divided into four main layers, which serve as clear boundaries between the logging responsibilities of the different systems in your enterprise.

The architecture should be able to handle a large enterprise with hundreds of thousands of systems across multiple datacenters. Smaller organizations can collapse the different layers in Figure 1 if appropriate. All of this infrastructure can be virtualized or cloud based, but performance or data sensitivity concerns may cause your organization to decide that parts of it should be kept in-house.

The Log Originators

Log Originators are all your normal servers, appliances, storage devices, switches, routers, firewalls, etc. These systems all send their logs to the closest Edge Aggregation systems, usually via UDP syslog.

For applications that cannot send their logs directly to syslog, you have several options to watch and scrape the logs from files

to send them upstream. The syslog daemons mentioned above all have some capability to gather logs from local files, plus there are other, simpler tools that can do the job.

Noting that all the systems that you use for the rest of your logging infrastructure are also Log Originators and should be sending their locally generated log messages off to an Edge Aggregation system is important.

The Edge Aggregators

Edge Aggregation systems perform many different tasks: gather logs from local machines, fix malformed logs and add metadata, and queue logs as needed for reliable delivery to Core Aggregation systems.

Gather Logs From All Local Machines

There are many edge systems, distributed around the organization. The number of Edge Aggregation systems you deploy is a balancing act involving cost, complexity (the number of systems to manage), load on each system, and the reliability of delivering logs to the Edge Aggregators from your other systems.

- The closer the Edge systems are to the systems generating the logs, the more reliable your logs are going to be. While UDP syslog is extremely reliable over a local LAN switch, once you start sending it through links that can be bottlenecks (routers, firewalls, WANs, etc.), the chance of losing logs due to congestion, equipment failure, routing errors, ACL errors, etc. starts climbing rapidly.

In theory the answer is to use a more reliable transport than UDP syslog; however, many systems and appliances can only talk UDP syslog, so even if you change all your servers to a more reliable transport, you have only solved part of the problem. Deploying the Edge Aggregation systems close to the sources of the logs in HA pairs will let you survive system and network failures and congestion with the minimum loss of logs. The Heartbeat project [2] provides the tools to make implementing HA on a pair of Linux systems trivial.

At the very least, you should have Edge Aggregators before any WAN hops. I try hard to have a set of Edge Aggregation systems connected so that logs never have to go through a router or firewall before they will hit an Edge Aggregation system. In some extreme cases, I have used Edge Aggregation systems that have as many as 22 Ethernet ports on them (5x 4-port cards plus 2 on the motherboard) to allow me to connect directly to the different networks.

Fix Malformed Logs/Add Metadata

Fixing the logs and adding metadata should be done as close to the source of the logs as possible. There are several reasons for this:

- It limits the scope of one-off fixes that you may need to do for particular devices.
- Testing every message to see whether it needs to be fixed is expensive. Modifying a message is less expensive, but still not free. Doing this on the Edge Aggregators scales well.
- The Edge Aggregators know the actual source IP of the Log Originators, while systems further on only know what's in the message.
- The Edge Aggregators can have hard-coded values based on where they are in the network.

Queue Logs for Reliable Delivery to Core Aggregation Systems

Because there are relatively few Edge Aggregation systems, any effort you spend on them has a much higher cost-benefit ratio than work done on the Log Origination systems. Because these systems do not have to be used for anything else, you can replace your OS defaults with newer or different versions of logging software, and they can afford to expend more effort to deliver messages. If the messages are being delivered over a WAN link that goes down, these systems are perfectly positioned to queue messages to be delivered later. This is not mandating full "audit grade" reliability, but simply using one of the network protocols that will detect outages such as TCP syslog or Reliable Event Logging Protocol (RELP). Think about the safety of the links you are sending the logs over; you may want to encrypt the data before it is sent.

The Core Aggregation System

This farm of Edge Aggregator systems handles your entire log feed. Its purpose is to provide a single logical destination to which all the Edge Aggregation systems can deliver their messages, and a single logical source for distributing the logs out to the various Analysis Farms.

Logically, this is a single system (implemented as a load-balanced cluster of boxes as needed). If you only have one datacenter, you can easily collapse this functionality into your Edge Aggregators by multi-homing them with one leg on a network you use for your Analysis Farms. If you have multiple datacenters with a disaster recovery set of Analysis Farms, you will want to spread it across two datacenters that have Analysis Farms. The logs from each half of the Core Aggregator cluster should be sent to the other half so that both sets of Analysis Farms will see the full set of logs. The other option is to have multiple Core Aggregation clusters and have your Edge Aggregators send (and queue) logs to every Core cluster.

Because this farm of systems is handling the full log feed, a large enterprise will need to have these systems doing as little work as possible. Ideally, they should not be doing any processing of the log messages other than aggregating and delivering them.

When delivering a large volume of logs to many destinations (many different Analysis Farms), the resulting traffic can strain your network (as well as the systems doing the sending). One good way of dealing with this problem is to use Multicast MAC as I described in a 2012 LISA paper [3].

The Analysis Farms

Analysis Farms are the systems that do something with the logs, and because that includes lots of different things (especially in a large organization), doing this analysis can take many systems' worth of resources. So it's a good idea to think of the different sets of functionality as separate farms, even if you start off implementing multiple sets of functionality on one box (or an HA pair of boxes). This approach makes it much easier to split things apart as your needs grow.

In a large enterprise, it may not be reasonable for everyone who needs to see some logs to be able to do so. Depending on the capabilities of the tools that you use, you may opt to implement such restrictions in each tool, or you may choose to have multiple Analysis Farms of the same type, but filter the logs so that a given farm only contains the subset of logs that the users of that farm are allowed to have access to.

The following are a handful of basic functions that you need to have as part of your Analysis Farms.

Log Archiving

This can be as trivial as an HA pair of systems that just receives the logs and writes them to disk in simple gzip files for long-term data retention. In a more demanding environment, you could have these systems digitally sign the log files to make them tamper-resistant, encrypt the archives, and store the archives off-site.

Log Message Alerts

You can get started with Simple Event Correlator (SEC) on an HA pair of systems, and as your load climbs you can split the logs across different machines along the lines described in this LISA 2010 paper [4]. It's a very good idea to feed the alerts that are generated back into the system as new log messages that all other Analysis Farms can then see.

Reporting on Log Contents

Start by using rsyslog filtering to split logs into different files per application and then have simple scripts crunch these smaller files periodically. (I do hourly and daily reports this way.) SEC can also be useful for reporting. You can use a combination of Calendar rules and simple content matching rules to count occurrences of matches and output the counts at regular intervals.

Searching Logs

At low volumes, you can get by with zgrep, but as log volumes increase, this becomes unwieldy as a general purpose search tool; however, it's still great when looking at a small time window for specific data, especially when combined with rsyslog filtering to create files that only contain a given type of log. This is where Hadoop, Cassandra, ElasticSearch (all free), and Splunk (commercial) come into play.

Other Uses

Beyond the basic functions outlined above, Analysis Farms provide endless possibilities for other uses, among them:

- Artificial ignorance reporting
- Machine learning
- Predictive modeling
- Automated reactions

The really nice feature of this architecture is that you can add/remove Analysis Farms without having to reconfigure anything beyond the Core Aggregators (and if you use the Multicast MAC approach to distribute the data, you don't even have to reconfigure those). This lets you experiment freely with different tools without disrupting anything else. It also makes it hard for someone to generate a new set of logs and only send it to the analysis tool that they care about without it going to other groups (an app team forgetting to send the logs to the security team, for example).

Here's an example of a simple trick that you can implement to get a lot of value immediately. I like to add vmstat and iostat data to the logs. This both produces a tremendously dense set of performance related data with little impact to the systems and provides a heartbeat that you can use to detect if anything (including system failure) interrupts the logs. Doing this can be as simple as adding

```
nohup vmstat 60 |logger -t vmstat 2>&1 >/dev/null &
nohup iostat -xk 60 |logger -t iostat 2>&1 >/dev/null &
```

to your startup scripts. And a simple config to SEC similar to:

```
type=Single
ptype=perlfunc
pattern=sub {@test=split(' ', substr($_[0],16)); if ($test[1] =~
   /vmstat/ ) { return $test[0];} }
desc=vmstat_$1
action=create vmstat_heartbeat_$1 180 ( shellcmd sendmessage
   "$1" )
continue=takenext
```

Conclusion

The value that you get out of a logging system is related far more to the effort that you put into the system than the amount of money you spend on the system. You can get a lot of value quickly without spending a significant amount of money. I hope that this article helps provide a road map that you can use to get started dealing with your logs regardless of how much data you end up dealing with as your system grows.

References

[1] http://json-ld.org/, http://cee.mitre.org/, https://fedorahosted.org/lumberjack/.

[2] http://linux-ha.org/wiki/Heartbeat.

[3] David Lang, " Building a 100K log/sec Logging Infrastructure": https://www.usenix.org/conference/lisa12/building-100k-logsec-logging-infrastructure.

[4] Paul Krizak, "Log Analysis and Event Correlation Using Variable Temporal Event Correlator (VTEC)": http://static.usenix.org/events/lisa10/tech/full_papers/Krizak.pdf.

xkcd