Jason Wilder's Blog

Software developer and architect interested in scalability, performance and distributed systems.

Centralized Logging Architecture

Jul 16, 2013 · 5 minute read · 14 Comments

logging fluentd logstash architecture

In Centralized Logging, I covered a few tools that help with the problem of centralized logging. Many of these tools address only a portion of the problem which means you need to use several of them together to build a robust solution.

The main aspects you will need to address are: *collection, transport, storage*, and *analysis*. In some special cases, you may also want to have an *alerting* capability as well.

Collection

Applications create logs in different ways, some log through syslog, others log directly to files. If you consider a typical web application running on a linux hosts, there will be a dozen or more log files in /var/log as well as a few application specific logs in home directories or other locations.

If you are supporting a web based application and your developers or operations staff need access to log data quickly in order to troubleshoot live issues, you need a solution that is able to monitor changes to log files in near real-time. If you are using a file replication based approach where files are replicated to a central server on a fixed schedule, then you can only inspect logs as frequently as the replication runs. A one minute rsync cron job might not be fast enough when your site is down and you are waiting for the relevant log data to be replicated.

On the other hand, if you need to analyze log data offline for calculating metrics or

other batch related work, a file replication strategy might be a good fit.

Transport

Log data can accumulate quickly on multiple hosts. Transporting it reliably and quickly to your centralized location may need additional tooling in order to effectively transmit it and ensure data is not lost.

Frameworks such as Scribe, Flume, Heka, Logstash, Chukwa, fluentd, nsq and Kafka are designed for transporting large volumes of data from one host to another reliably. Although each of these frameworks addresses the transport problem, they do so quite differently.

For example, Scribe, nsq and Kafka, require clients to log data via their API. Typically, application code is written to log directly to these sources which allows them to reduce latency and improve reliability. If you want to centralize typical log file data, you would need something to tail and stream the logs via their respective APIs. If you control the app that is logging the data you want to collect, these can be much more efficient.

Logstash, Heka, fluentd and Flume provide a number of input sources but also support natively tailing files and transporting them reliably. These are a better fit for more general log collection.

While rsyslog and Syslog-ng are typically thought of as the defacto log collector, not all applications use syslog.

Storage

Now that your log data is being transfered, it needs a destination. Your centralized storage system needs to be able to handle the growth in data over time. Each day will add a certain amount of storage that is relative to the number of hosts and processes that are generating log data.

How you store things depends on a few things:

- *How long should it be stored* - If the logs are for long-term, archival purposes and do not require immediate analysis, S3, AWS Glacier, or tape backup might be a suitable option since they provide relatively low cost for large volumes of data. If you only need a few days or months worth of logs, storing them on some form distributed storage systems such as HDFS, Cassandara, MongoDB or ElasticSearch also works well. If you only need a few hours worth of retention for real-time analysis, Redis might work as well.

- *Your environments data volume*. - A days worth of logs for Google is much different than a days worth of logs for ACME Fishing Supplies. The storage system you chose should allow you to scale-out horizontally if your data volume will be large.

- *How will you need to access the logs* - Some storage is not suitable for real-time or even batch analysis. AWS Glacier or tape backup can take hours to load a file. These don't work if you need log access for production troubleshooting. If you plan to do more interactive data analysis, storing log data in ElasticSearch or HDFS may allow you work with the raw data more effectively. Some log data is so large that it can only be analyzed in more batch oriented frameworks. The defacto standard is this case is Apache Hadoop along with HDFS.

Analysis

Once your logs are stored on a centralized storage platform, you need a way to analyze them. The most common approach is a batch oriented process that runs periodically. If you are storing log data in HDFS, Hive or Pig might help analyzing the data easier than writing native MapReduce jobs.

If you need a UI for analysis, you can store parsed log data in ElasticSearch and use a front-end such as Kibana or Graylog2 to query and inspect the data. The log parsing can be handled by Logstash, Heka or applications logging with JSON directly. This approach allows more real-time, interactive access to the data but is not really suited for a mass batch processing.

Alerting

The last component that is sometimes nice to have is the ability to alert on log patterns or calculated metrics based on log data. Two common uses for this are error reporting and monitoring.

Most log data is not interesting but errors almost always indicate a problem. It's much more effective to have the logging system email or notify respective parties when errors ocurr instead of having someone repeatedly watch for the events. There are several services that solely provide application error logging such as Sentry or HoneyBadger. These can also aggregate repetitve exceptions which can give you and idea of how frequently an error is occuring.

Another use case is monitoring. For example, you may have hundreds of web servers and want to know if they start returning 500 status codes. If you can parse your web log files and record a metric on the status code, you can then trigger

alerts when that metric crosses a certain threshold. Riemann is designed for detecting scenarios just like this.

Hopefully this helps provide a basic model for designing a centralized logging solution for your environment.

Patrick Woll • a year ago

NxLog is great for collecting and transporting of logs by the way: http://www.nxlog.org/products/...

^ | v • Reply • Share ›

Swapnil Sonawane • a year ago

Really helpful this post, Jason. Helps to get started and explore different tools. It saves a lot of google searches.

^ | v • Reply • Share ›

ROCKY MAJUMDAR • a year ago

Thanks Jason for this great article.

^ | v • Reply • Share ›

Guest • 2 years ago

Thanks Jason for enlightenment! Every company with high volumes needs such a solution.

^ | v • Reply • Share ›

OpenOpsIQ • 2 years ago

Jason - thanks for this article. As you have pointed out, there is a rich collection of mature open source tools to help with monitoring and logging. It is interesting to see LinkedIn and Netflix release some of their own centralized logging and monitoring architectures. What is particularly interesting is the real-time event collection and processing aspect enabled by stream processing technologies like Apache Storm etc. We have referenced your blog in one of our posts here.

http://openopsiq.com/2014/01/1...

^ | v • Reply • Share ›

Javier Alba • 2 years ago

Clear and concise, thanks.

Just a couple of comments/questions:

Don't you think that HDFS is also suitable for long-term storage?
Would you say that real-time oriented tools (like Storm) are also suitable for the "alerting" part?

^ | v • Reply • Share ›

Jason Wilder Mod ↗ Javier Alba • 2 years ago

Yes, HDFS could be used for long-term storage as well. HDFS is more complicated to operate than something like S3 though. If you never intend to look at that data and only need it for archival purposes, it might be more work then necessary. Certainly a valid option though.

Storm could be a really good fit for the alerting use case too. A common approach I've seen is to have events flow through storm for real-time

approach I've seen is to have events flow through storm for real-time analysis and, in parallel, to HDFS for larger batch processing.

• Reply • Share ›

Madhumitha • 2 years ago

Amazing article!

• Reply • Share ›

Anat • 2 years ago

This is a great overview of the topic, thanks!

• Reply • Share ›