MSc in Computing (Communications Software)

A Lightweight System for Just-In-Time Aggregation of Machine Generated Data in a Distributed Network

By
Michael
Dreeling

Waterford Institute of Technology
Department of Computing, Mathematics and Physics

May 25, 2013

Table of Contents

1 Introduction 3

2 Literature Review 3

2.1 Overview of Push Data Collection Systems 3

2.2 Scope Change for JVM Metrics Analysis 5

3 System Design 6

3.1 General Architecture and Problem Domain Model 6

3.2 Data Siphon Server Component 8

3.2.1 multi-ssh-client 8

3.2.2 input-filter 9

3.2.3 output-filter 10

3.2.4 queuing-system 10

3.2.5 profile-manager 10

3.2.6 ui-dashboard and search 10

4 Research Questions Re-visited 11

5 Research Methodology 11

6 Achievements To Date 13

1 Introduction

The purpose of this document is to provide an update on the progress of the dissertation, and to serve as an update to the dissertation proposal. As this dissertation was deferred for one year, I will also be addressing any significant changes in the data ingestion and technology landscape that may affect the direction of the dissertation. The dissertation will attempt to investigate the possibilities of ingesting data in real-time from a distributed system at scale without the need for collectors on each node.

2 Literature Review

The following section describes a detailed literature review of the application area. This dissertation will use the comparison of distributed aggregation systems with a much simpler centralized approach, for the specific use case of crisis management diagnostics in large distributed systems. Each of the existing approaches are analyzed in terms of their

2.1 Overview of Push Data Collection Systems

GrayLog2 [1] is an open sourced centralized logging system intended for use by organizations' in order to provide a window onto real-time and historical log data. The data is pushed to Graylog's server (graylog-server) using syslog from each registered node in data pipelines known as 'streams'. The nodes themselves must be registered within GrayLog using its UI interface. Gray log records the Date, Host, and Level of each of the incoming messages. It reports the number of message per second which are coming into the system in the UI. It has advanced searching features using Elastic Search as it's backend, which even allows some natural language to be used when searching records (i.e you can search a Timeframe such as 'from yesterday'). This UI also provides advanced graphing and tracking of multiple sources at once. It does not support a centralized Multi-SSH server siphon as outlined in the dissertation.

LogStash [5] is also an open sourced tool for managing events and logs. It is written in Ruby and runs on a Java runtime using JRuby. Again with LogStash it uses a push model in order to 'Ship' events from each node to 'Brokers' which cache the information for the 'Indexers' so that they do not become overloaded, this is then finally redirected to the 'Storage and Search' server that is accessible via a Web UI. LogStash has a highly configurable UI (utilizing Kibana) which allows you to create your own dashboards for your application. LogStash needs to be installed on the nodes which contain the logs if they are not being presented over the standard Syslog port 514. [19]

OpenTSDB [2] provides another way to push log and metrics events to a central location. It is a highly scalable system which is in use by many companies [3] at the time of writing. The base components for the system is the 'Time Series Daemon' or 'TSD', several of which is installed on the network being monitored. Each TSD uses HBase [4] to store and retrieve data. 'Collectors', which are basically scripts that are scheduled to run on each monitored node, send data to each TSD which in turn writes the data to HBase. Each monitored node must be aware of the TSD's on the network so that it can send data to them. Again, OpenTSDB is mainly is focused on aggregation and storage of large amounts of historical data and at massive scale, so it utilizes the common push method to achieve this. From a UI perspective, OpenTSDB provides only basic functionality [6] when compared to Kibana based solutions.

Apache Flume [20] is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of streaming event data. Everything in Flume is defined in the content of an event. Events are transported, routed and stored by Flume 'Agents'. Flume Agents are a collection of Flume 'Sources', 'Sinks' and 'Channels', Sources can poll or wait for events, Sink's allow the data to be streamed to a destination (i.e HDFS), and channels provide the conduit between the Source and the Sink (i.e the Source sends the events to a Channel and the Sink drains the Channel).

There are no SSH connectors for Flume (although open source attempts [7] have been contributed), the reason for this is that the default solution for reading data is to install agents on the Application Server obtaining the logs and to send them to a Sink. The output of this dissertation project is to build a completely passive SSH spooling service that can be connected to a data source, so it is possible that it could be made compatible with Flume.

Splunk [21] is a closed-source commercial offering which is in use by thousands of customers worldwide. It is available in both free and enterprise edition(s). Splunk uses an architecture whereby a 'Forwarder' sends data to a Splunk 'Indexer', a separate server known as the Splunk 'Search Head' provides a UI by which to search the Indexer(s) for event data. It follows paradigms which are very similar to that of OpenTSDB and Flume but with a much richer UI and a simplified configuration. Splunk also scales horizontally extrmnely well Splunk does not provide a feature to read remote files via SSH, if you wish to do so, a script needs to be written to connect to the SSH server and then forward the data.

Kafka [22] is a newer data collection system which was not mentioned in the original proposal, at the time it was at viewed as being so similar to Flume that only one of them merited deeper analysis. It was also at version 0.7.0 having not been released from the Apache Incubator after it was contributed in 2011. Today, Kafka is one of the leading real time open source data pipelines. LinkedIn also released a companion paper [9] which describes the entire system in detail. Kafka is also horizontally scalable and supports up to 40 billion events per day at LinkedIn. [10]

Kafka uses the notion of a Producer and Consumer which send a retrieve data respectively from a Kafka cluster of 'Brokers'. Kafka also has the concept of a channel (similar to Flume) on which you send data to Brokers which is called a 'Topic'. Kafka is very much the just mechanism for achieving data collection at Scale, it does not provide any default Producer's or Consumer's, these must be written. An example of a simple tail Producer [11] is available on GitHub. Apache Storm [12] is frequently used with Kafka as a distributed computation engine which operates on data streamed from a Kafka cluster.

Suro is a data pipeline in use by both NetFlix and Riot Games [14, 13]. Suro is a collection and storage mechanism for streaming data. The version of Suro in use by Riot Games is closed source and known as Honu, but is similar in operation to the version of Suro recently open sourced by NetFlix, they were both developed by the same author independently for both companies. Suro uses a pure collect and forward architecture, requiring 'Collectors' to be installed close to the application being monitored. Agents are not required to be installed on the monitored nodes, however any application which needs to send data to Suro needs to implement SuroSDK, which effectively generates a client side dependency. You cannot use Suro to SSH or retrieve data from nodes remotely. However, it would be possible to implement Suro support into the Multi-SSH server such that it provided its output to Suro. Suro can forward raw data to either S3 or Kafka for further processing. It does not have a real-time consume library or mechanism, hence the use of Kafka.

2.2 Scope Change for JVM Metrics Analysis

For the purposes of this dissertation, It has been decided that the JVM metrics analysis mentioned in the proposal will be secondary to the primary goal of designing and building a server which can stream data from multiple SSH servers. This is to say that a JVM collector may not be designed or built, however, existing JVM collectors may be able to have their logs introspected by the SSH Multi-Client.

3 System Design

The Multi-SSH client system involves design and development of the following components

- Passive MGD Collector (or DSS)
- DSS Data Model
- DSS UI

The purposes of this document will be to describe the technical design proposal for the components above.

3.1 General Architecture and Problem Domain Model

The basic operation of the system is to download machine generated data from several hosts at once via SSH and feed them onto a queue where they can be consumed and presented in a UI.

The MGD being retrieved should normally represent a holistic view of an application or system (such as all of the security logs across an entire server farm)

Before this can be achieved, the hosts, applications and logs must all be predefined in a database so that they can be retrieved quickly and siphoned into the UI on a just in time basis.

The Architecture consists of the following components

- Data Siphon Server :: input-filter
- Data Siphon Server :: multi-ssh-client
- Data Siphon Server :: output-filter
- Data Siphon Server :: queueing-system
- Data Siphon Server :: app-database
- Data Siphon Server :: profile-manager

- UI Server :: rest-data-access
- UI Server :: queue-reader
- UI Server :: ui-dashboard

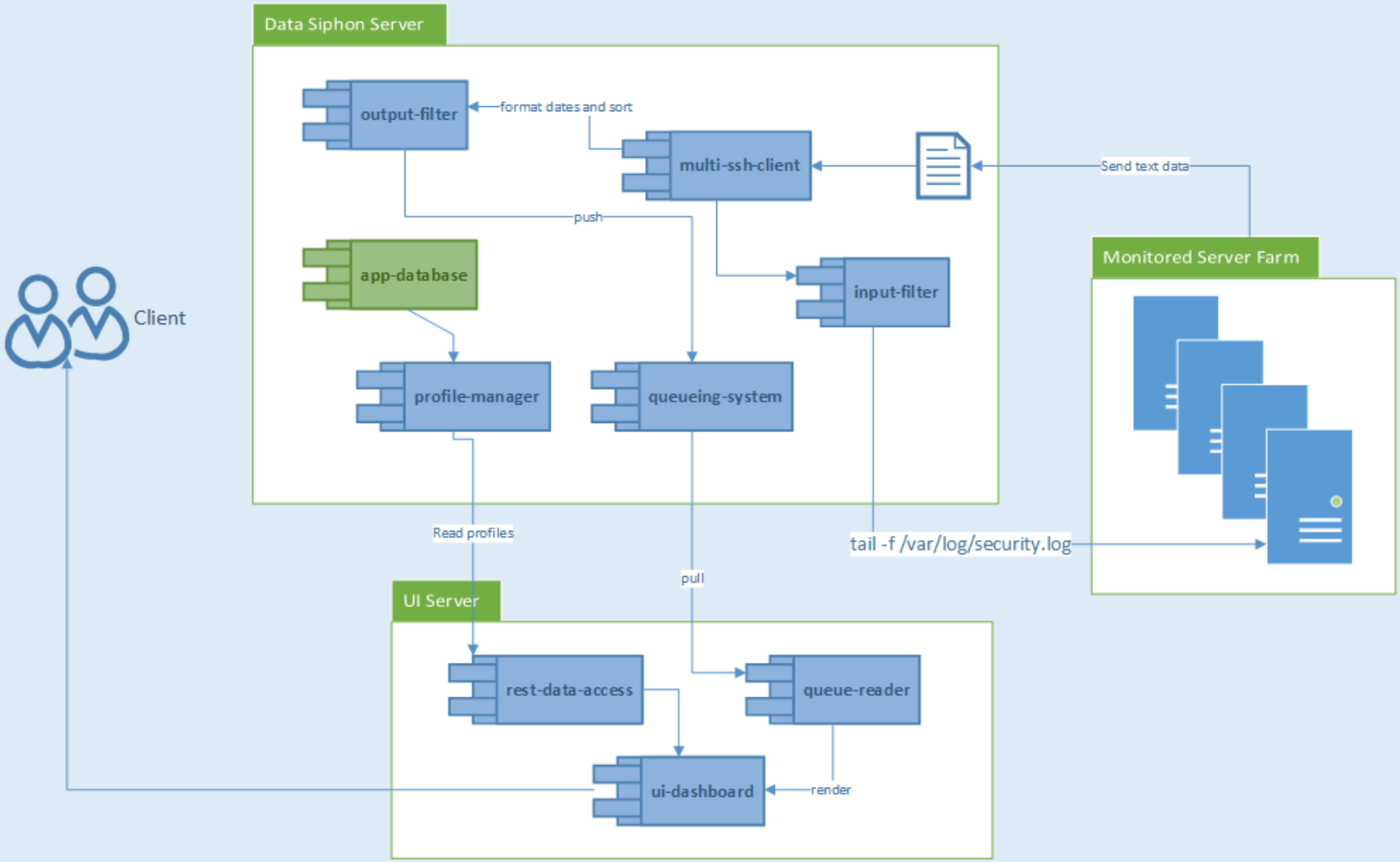
The problem domain consists of the following entities.

- Applications (Web, Daemons)
- Nodes (Hosts, Servers)
- Machine Generated Data Points (Logs)

These components are described in more detail in the diagram below and also in the next section.

Data Siphon Server Architecture

March 31, 2013



3.2 Data Siphon Server Component

In this section we will analyze the different components of the system and determine suitable software designs and libraries to be used.

3.2.1 multi-ssh-client

In order to connect and retrieve log information, across multiple servers, a client which can log into and maintain a very high number of SSH connections will need to be developed or obtained.

This client should have the following characteristics

- Lightweight
- SSH2 support
- Supports compression
- Wide cipher support

Due to the high number of required connections, SSH libraries which have various compression methods will be preferred. There are several libraries which can be used across multiple different languages.

Libraries available and being considered are listed below

Java Client Libraries

Name	*License*	*Link*
Ganymed	BSD	http://www.ganymed.ethz.ch/ssh2
JCraft	BSD	http://www.jcraft.com/jsch/
SSHj	Apache 2.0	https://github.com/shikhar/sshj
J2SSH Maverick	Commercial	https://www.javassh.com
Jaramiko	MIT Style	http://www.lag.net/paramiko/java/

Ruby Client Libraries

Name	*License*	*Link*
Net:SSH	MIT Style	http://net-ssh.rubyforge.org/
Rye	MIT Style	https://github.com/delano/rye

C Libraries

Name	*License*	*Link*
Dancers Shell	GNU	http://www.netfort.gr.jp/~dancer/software/dsh.html.en
LibSSH	LGPLV2	http://www.libssh.org/
FLowSsh	Commercial	http://www.bitvise.com/flowssh

Python Libraries

Name	*License*	*Link*
Paramiko	MIT Style	http://www.lag.net/paramiko/
Spur	MIT Style	https://pypi.python.org/pypi/spur
Fabric	MIT Style	http://docs.fabfile.org/en/0.9.1/
PXSsh (part of PexSpect)	MIT Style	http://pexpect.sourceforge.net/pxssh.html

Most libraries do not inherent cater for the management and retrieval of data across multiple connections at once, although Dancers Shell, Fabric and Rye do.

Library Selection

The selected library for the implementation will be JSch (Java Secure Channel) from JCraft. This library is pure Java and includes key features such as

- High Performance Enabled SSH/SCP [1]
- JZlib compression
- Variety of Cipher and Encryption options

As such this library has the largest majority of features from other available libraries. The only feature which JSch does not implement by default is the ability to maintain several concurrent connections and execute identical commands on each one. However, the ease of use of the library makes it much straight forward to implement such functionality.

3.2.2 input-filter

The purpose of the input filter is to protect the downstream operating systems from restricted commands that are not required for the operation of the data siphoning operation. The only system known to sanitize these commands at the time of writing is Rye, which disables the usage of

- File globs. i.e ls*.rb
- Environment variables as arguments. i.e echo $HOME
- Pipes and operators i.e |, &&, >, <, ||, ~
- Backticks, i.e procs=`ps aux`

Any implementation of the input-filter should at least cover the scenarios covered by Rye.

3.2.3 output-filter

The purpose of the output filter is to discard erroneous data which would otherwise be queued and displayed on the UI. All data passes through this filter and as such there may need to be many instances of this component.

The primary function of the output filter is twofold and should be configurable as follows

- Allow the removal any data which the user has specified in their profile. (i.e The removal of lines containing the word INFO)
- Allow only specific data to pass through (i.e lines only containing the word ERROR)

3.2.4 queuing-system

A queuing system is required in order to architecturally de-couple ingest of data from its consumption. The UI should be able to read and display events independently of the system receiving the. The queuing system which is most likely to be used will be a simple in memory solution which will spill over to disk when full. In order to preserve messages and queue them to the UI, a persistent ActiveMQ solution may also be used. Access to the queue will be provided via an API.

3.2.5 profile-manager

A profile management system is necessary in order to store the applications which a user wishes to monitor. This will be implemented as part of the MySQL database for the SSH Multi-Client application. A user will be able to store profiles containing information about the following entities

- Environments (i.e Production, Test)
- Nodes (i,e Application Hosts such as an Application Server, Db Server)
- Applications (The software being monitored)

3.2.6 ui-dashboard and search

Currently Kibana [15] is being investigated for use as a dashboard and ElasticSearch [16] for search capabilities of the data. These technologies are current in use by other streaming collection mechanisms such as GrayLog2.

4 Research Questions Re-visited

Based on feedback received in 2013 on the dissertation proposal I felt the need to modify some of the wording in my initial research questions in order to provide more detail

1. Is it possible to build a completely passive machine generated data collector which can scale on all resources (cpu, network and memory) when centrally connected to, and reading data from a number of servers?
2. What is the impact such a system would have on the monitoring and monitored servers (cpu, memory) and on the network bandwidth between them.

5 Research Methodology

I have made some slight changes to the research methodology, updated dates based on my deferral and also update assumptions based on knowledge attained since last year, especially around the fact that Kibana [15] has become a very viable UI for this project.

5.1 WP1: Analysis. Expected Completion: 201301

1. Choose and validate transfer protocol as SSH2 (expected duration: 2 days).
2. Choose UI technology (expected duration: 4 days).
3. Choose server side development language and database system (expected duration: 4 days).

5.2 WP2: Design. Expected Completion: 201303

1. Design multi-client SSH data siphoning server. (expected duration: 7 days).
2. Design database schema to store application profiles (expected duration: 7 days).

5.3 WP3: Implementation. Expected Completion: 201407

1. Implement multi-client SSH data siphoning server (expected duration: 30 days)
2. Integrate ElasticSearch with Kibana UI (expected duration: 15 days)

5.4 WP4: Data Analysis. Expected Completion: 201407

1. Run load tests on multi-node network on Amazon EC2 (>10 nodes) (expected duration: 5 days)
2. Test and Analyze effect of SSH2 compression on data delivery (expected duration: 3 days)
3. Load Test UI (expected duration: 2 days)

5.5 WP5: Final Write Up. Expected Completion: 201308

1. Dissertation write-up. (Expected duration 30 days)

6 Achievements To Date

Since last year, several accomplishments have been made for this research project. Currently the project is on schedule for completion in August 2014.

6.1 Research Milestones Achieved

Extensive research has been made into the various data aggregation systems available, and their operation. With this research in mind, the following decisions have been made:

- Elastic search will be used as the indexing solution within which data from the SSH servers will be stored.
- Kibana [15] will be used as the UI for the system, it provides integration with ElasticSearch [16] and is commonly used as a search mechanism in other data aggregation implementations.
- JCraft JSch [23] will be used as the SH library, it is written in pure Java and provides the best balance between ease of use, performance and compression.

6.2 Software Milestones Achieved

- The MVP of the Multi-SSH Server is implemented, it connects to a single EC2 server in Amazon and streams the data from a large file using the unix tail command. [17]
- A grails application has been developed to administer the Profiles section of the application, this is still under development. [18]

[1] Anon, 2014. *Graylog2 - Open source log management and data analytics*. [online] Available at: <http://graylog2.org/> [Accessed 25 May 2014].

[2] Anon, 2014. *OpenTSDB - A Distributed, Scalable Monitoring System*. [online] Available at: <http://opentsdb.net/overview.html> [Accessed 25 May 2014].

[3] Anon, 2014. *Companies using OpenTSDB in production · OpenTSDB/opentsdb Wiki*. [online] Available at: <https://github.com/OpenTSDB/opentsdb/wiki/Companies-using-OpenTSDB-in-production> [Accessed 25 May 2014].

[4] Anon, 2014. *HBase - Apache HBase™ Home*. [online] Available at: <https://hbase.apache.org/> [Accessed 25 May 2014].

[5] Anon, 2014. *logstash - open source log management*. [online] Available at: <http://logstash.net/> [Accessed 25 May 2014].

[6] Anon, 2014. *GUI — OpenTDSB 2.0 documentation*. [online] Available at: <http://opentsdb.net/docs/build/html/user_guide/guis/index.html> [Accessed 25 May 2014].

[7] Anon, 2014. *prateek/ssh-spool-source*. [online] Available at: <https://github.com/prateek/ssh-spool-source> [Accessed 26 May 2014].

[8] Anon, 2014. *Welcome to Apache Flume — Apache Flume*. [online] Available at: <http://flume.apache.org/> [Accessed 26 May 2014]

[9] Ken Goodhope, Joel Koshy, Jay Kreps, Neha Narkhede, Richard Park, Jun Rao, Victor Yang Ye, 2012. *Building LinkedIn;s Real-time Activity Data Pipeline*. [online] Available at: <http://sites.computer.org/debull/A12june/pipeline.pdf/> [Accessed 26 May 2014]

[10] Anon, 2014. *A Few Notes on Kafka and Jepsen - Jay Kreps*. [online] Available at: <http://blog.empathybox.com/post/62279088548/a-few-notes-on-kafka-and-jepsen> [Accessed 26 May 2014].

[11] Anon, 2014. *harelba/tail2kafka*. [online] Available at: <https://github.com/harelba/tail2kafka> [Accessed 26 May 2014].

[12] Anon, 2014. *Storm (event processor) - Wikipedia, the free encyclopedia*. [online] Available at: <http://en.wikipedia.org/wiki/Storm_(event_processor)> [Accessed 26 May 2014].

[13] Jerome Boulon, 2014. *Honu/Big Data @ Riot Games*. [online] Available at: <http://www.slideshare.net/jboulon/big-dataatriotgames-sdvfinal> [Accessed 26 May 2014].

[14] Anon, 2014. *The Netflix Tech Blog: Announcing Suro: Backbone of Netflix's Data Pipeline*. [online] Available at: <http://techblog.netflix.com/2013/12/announcing-suro-backbone-of-netflixs.html> [Accessed 26 May 2014].

[15] Anon, 2014. *Elasticsearch.org Kibana | Overview | Elasticsearch*. [online] Available at: <http://www.elasticsearch.org/overview/kibana/> [Accessed 26 May 2014].

[16] Anon, 2014. *Elasticsearch.org Open Source Distributed Real Time Search & Analytics | Elasticsearch*. [online] Available at: <http://www.elasticsearch.org/> [Accessed 26 May 2014].

[17] Anon, 2014. *mdreeling/ocelli*. [online] Available at: <https://github.com/mdreeling/ocelli> [Accessed 26 May 2014].

[18] Anon, 2014. *ocelli/ocelli-grails at master · mdreeling/ocelli*. [online] Available at: <https://github.com/mdreeling/ocelli/tree/master/ocelli-grails> [Accessed 26 May 2014].

[19] Anon, 2014. *logstash and kibana for adhoc log analysis | OpenNomad*. [online] Available at: <http://opennomad.com/content/logstash-and-kibana-adhoc-log-analysis> [Accessed 26 May 2014].

[20] Anon, 2014. *Welcome to Apache Flume — Apache Flume*. [online] Available at: <http://flume.apache.org/> [Accessed 26 May 2014].

[21] Anon, 2014. *Operational Intelligence, Log Management, Application Management, Enterprise Security and Compliance | Splunk*. [online] Available at: <http://www.splunk.com/> [Accessed 26 May 2014].

[22] Anon, 2014. *Apache Kafka*. [online] Available at: <http://kafka.apache.org/> [Accessed 26 May 2014].

[23] Anon, 2014. *JSch - Java Secure Channel*. [online] Available at: <http://www.jcraft.com/jsch/> [Accessed 26 May 2014].