MSc in Computing (Communications Software)

Design Document

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

March 25, 2013

Table of Contents

1 Introduction

This document servers to describe the general high level design of a passive data collection server using SSH2 protocols. The document will include comparisons of tools and technologies required to build and test the system.

2 Server Side Design

The server side of the 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.

2.1 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 next section.

2.2 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.

2.2.1 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 at the time of writing 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.

2.2.2 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 (if Rye is not used) should at least cover the scenarios covered by Rye.

2.2.3 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)

2.2.4 queuing-system

2.2.5 profile-manager

2.2.6 rest-data-access

2.2.7 queue-reader

2.2.8 ui-dashboard