LOG ANALYZER

A Project Report

Submitted by

Prajwal S. Bhusari 111603005

in partial fulfillment for the award of the degree

of

B.Tech Computer Engineering/

Information Technology

Under the guidance of

Prof. S. N. Ghotkar

College of Engineering, Pune

AND

Mr. Rajendra Sansare

VMware, Inc.

DEPARTMENT OF COMPUTER ENGINEERING

AND

INFORMATION TECHNOLOGY,

COLLEGE OF ENGINEERING, PUNE-5

May, 2020

DEPARTMENT OF COMPUTER ENGINEERING

AND

INFORMATION TECHNOLOGY,

COLLEGE OF ENGINEERING, PUNE

CERTIFICATE

Certified that this project, titled "Log Analyzer" has been successfully completed by

Prajwal Bhusari 111603005

and is approved for the partial fulfillment of the requirements for the degree of "B.Tech. Computer Engineering".

SIGNATURE
Prof. S. N. Ghotkar
Project Guide
Department of Computer Engineering and Information Technology, College of Engineering Pune, Shivajinagar, Pune - 5.

SIGNATURE
Dr. Vahida Attar
Head
Department of Computer Engineering and Information Technology, College of Engineering Pune, Shivajinagar, Pune - 5.

Abstract

It takes a lot of manual effort and time to analyze logs and support bundles when an issue is encountered. An automated tool can reduce the time and effort to a considerable extent. The need for a tool that automatically goes through the log files and produces the high-level analysis was felt. LogAnalyzer is one such tool that will make log analysis easier for everyone by following a flow through each component for a given unique id. This will provide debuggers to get an overview of what's happening in the log bundle. It takes support bundles and/or log files as input and produces the correlation between the logs and display logs where the error is most likely happen by comparing it with the ideal flow.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

"Logs are imperative in the development and maintenance process of many software systems. They record detailed runtime information during system operation that allows developers and support engineers to monitor their systems and track abnormal behaviors and errors. LogAnalyzer provides a toolkit that implements a number of log analysis techniques that narrow down to the error to a particular component. The project is based on the Log Analysis of logs generated by the NSX support bundle. In order to understand the scale and depth of the system, it becomes crucial to have a quick overview of NSX Data Center a.k.a. NSX-T."[7]

1.1.1 Overview of NSX-T

"In much the same way that server virtualization programmatically creates, snapshots, deletes and restores software-based virtual machines (VMs), NSX-T network virtualization programmatically creates, deletes, and restores software-based virtual networks. With network virtualization, the functional equivalent of a network hypervisor reproduces the complete set of Layer 2 through Layer 7 networking services (for example, switching, routing, access

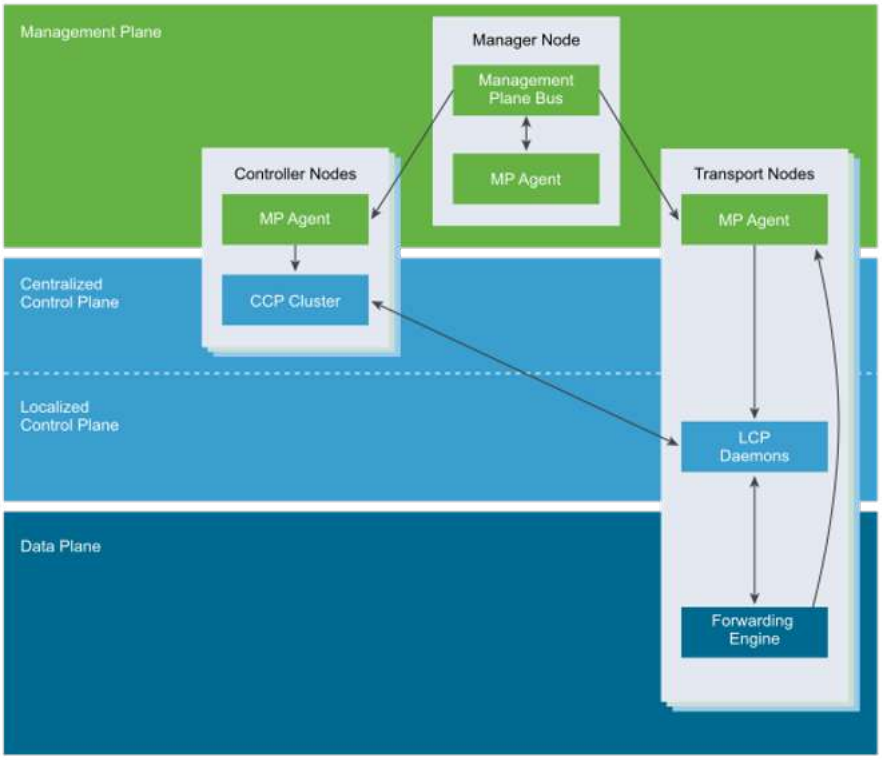

Figure 1.1: Architecture of NSX-T

control, firewalling, QoS) in software. As a result, these services can be programmatically assembled in any arbitrary combination, to produce unique, isolated virtual networks in a matter of seconds. NSX-T works by implementing three separate but integrated planes: management, control, and data. The three planes are implemented as a set of processes, modules, and agents residing on three types of nodes: manager, controller, and transport nodes.

Every node hosts a management plane agent.

The NSX Manager node hosts API services. Each NSX-T installation supports a single NSX Manager node.

NSX Controller nodes host the central control plane cluster daemons.

NSX Manager and NSX Controller nodes may be co-hosted on the same physical server.

Transport nodes host local control plane daemons and forwarding engines."[1]

1.1.2 Data Plane

"Performs stateless forwarding/transformation of packets based on tables populated by the control plane and reports topology information to the control plane, and maintains packet level statistics. The data plane is the source of truth for the physical topology and status for example, VIF location, tunnel status, and so on. If you are dealing with moving packets from one place to another, you are in the data plane. The data plane also maintains status of and handles failover between multiple links/tunnels. Per-packet performance is paramount with very strict latency or jitter requirements. Data plane is not necessarily fully contained in kernel, drivers, userspace, or even specific userspace processes. The Data plane is constrained to totally stateless forwarding based on tables/rules populated by the control plane. The data plane also may have components that maintain some amount of state for features such as TCP termination. This is different from the control plane managed state such as MAC:IP tunnel mappings, because the state managed by the control plane is about how to forward the packets, whereas state managed by the data plane is limited to how to manipulate payload."[2]

1.1.3 Control Plane

"Computes all ephemeral runtime state based on configuration from the management plane, disseminates topology information reported by the data plane elements, and pushes stateless configuration to forwarding engines. The control plane is sometimes described as the signaling for the network. If you are dealing with processing messages in order to maintain the data plane in

the presence of static user configuration, you are in the control plane (for example, responding to a vMotion of a virtual machine (VM) is a control plane responsibility, but connecting the VM to the logical network is a management plane responsibility) Often the control plane is acting as a reflector for topological info from the data plane elements to one another for example, MAC/Tunnel mappings for VTEPs. In other cases, the control plane is acting on data received from some data plane elements to (re)configure some data plane elements such as, using VIF locators to compute and establish the correct subset mesh of tunnels. The set of objects that the control plane deals with include VIFs, logical networks, logical ports, logical routers, IP addresses, and so on. The control plane is split into two parts in NSX-T, the central control plane (CCP), which runs on the NSX Controller cluster nodes, and the local control plane (LCP), which runs on the transport nodes, adjacent to the data plane it controls. The Central Control Plane computes some ephemeral runtime state based on configuration from the management plane and disseminates information reported by the data plane elements via the local control plane. The Local Control Plane monitors local link status, computes most ephemeral runtime state based on updates from the data plane and CCP, and pushes the stateless configuration to forwarding engines. The LCP shares fate with the data plane element which hosts it"[2]

1.1.4 Management Plane

"The management plane provides a single API entry point to the system, persists user configuration, handles user queries, and performs operational tasks on all management, control, and data plane nodes in the system. For NSX-T anything dealing with querying, modifying, and persisting user configuration is a management plane responsibility, while dissemination of that

configuration down to the correct subset of data plane elements is a control plane responsibility. This means that some data belongs to multiple planes depending on what stage of its existence it is in. The management plane also handles querying recent status and statistics from the control plane, and sometimes directly from the data plane. The management plane is the only source-of-truth for the configured (logical) system, as managed by the user via configuration. Changes are made using either a RESTful API or the NSX-T UI.

In NSX there is also a management plane agent (MPA) running on all cluster and transport nodes. Example use cases are bootstrapping configurations such as central management node address(es) credentials, packages, statistics, and status. The MPA can run relatively independently of the control plane and data plane, and to be restarted independently if its process crashes or wedges, however, there are scenarios where fate is shared because they run on the same host. The MPA is both locally accessible and remotely accessible. MPA runs on transport nodes, control nodes, and management nodes for node management. On transport nodes, it may perform data plane related tasks as well.

Tasks that happen on the management plan include:

1. Configuration persistence (desired logical state)
2. Input validation
3. User management – role assignments
4. Policy management
5. Background task tracking"[2]

1.2 Overview of Platform MP team

The MP team is responsible for providing the development for new functionalities and bugs and also analyzing issues reported in customer environment. Planning and discussion for features to be done for releases.

A typical day of any teammate is spent either in working on new features for the next release and/or debugging issues and providing fixes to the same. A lot of time is utilized to get to the root cause and hence we need to automate this repetitive task and get a clearer picture of the happenings in the system. A high-level view of the tasks can be generated based on the logs.

1.3 Log Analysis

Log Analysis is a fundamental step in the debugging and bug fixing. It's important to root cause the issue which is present somewhere in millions of log lines. Some issues are sequence dependent hence it becomes a kind of routine task to find the missing piece of log which in turn means the missing piece of code execution.

1.3.1 Log Format

The log format for the logs is:

<Timestamp><Severity><Thread><Class><Component><Message>

1.3.2 Sequence of Logs

The logs form a sequence like A->C->J->P->Z. We need to track this sequence in the logs. If after C we don't find J we say that an error has occurred.

1.3.3 Instances of Sequences

To make matters more complex we don't have only one sequence in the logs. An activity may be triggered many times which are different from each other and can be distinguished using IDs in the Message and also in the Thread Name at times. These multiple sequences form the instances of the sequences.

Chapter 2

Literature Survey

2.1 Existing Methods

The developers currently go through the logs using the command line applications like vim and commands like grep and awk.

2.2 Drawbacks of Existing Methods

The existing methods are efficient and are in place since the early inception of computers. The biggest drawback of the method is that it is slow when there are log files of huge scaled setups. Going through all the files manually is a painful task and doing it repetitively is a much bigger challenging task. The manual verification becomes difficult at times and regression problems are missed. This leads to jumping of the bug to multiple teams until somebody figures out the actual root cause.

2.3 Problem Definition

1. **Analysis** - Ability to narrow down the issue to specific Component and Module. Within a component, toolkit traces the flow and identifies modules where the issue is likely to have been caused.

2. **Database Query** - Pluggable Framework that allows other components to plug in their toolkit, provide input (logs, patterns, parameters). Easy to access/run Component Toolkit.

2.4 Requirement Specification

2.4.1 Functional Requirements

Parse the logs provided by the user as per Regex provided by user in config file.

Provide a high-level summary to understand the happenings on the system.

After parsing logs, dump it in ElasticSearch for allowing queries.

2.4.2 Non-Functional Requirements

Provide a user interface to upload support bundles or log files

Untar the support bundles which are relevant according to arguments given by user and get the log files in consideration.

Provide a way such that it is pluggable anywhere.

Chapter 3

Proposed System

3.1 Abstract Workflow

3.1.1 Analysis

Log Collection

Provide interface such that user will be able to put many mangers and edge bundles with a given unique id. Add various functionalities for developers to debug the issue efficiently. Add help function to show user how to use Log Analyzer.

Log Analysis

Analyze the logs by grepping uuid through each component, create some output files to show the analysis. Show the workflow for the given uuid through each and every component of support archive

Log Summary

Filter the log bundle and output it in different files according to the analysis.

1. *Analysis.txt*
2. *MatchingLogs.txt*

3. *ErrorLogs.txt*

4. *UUIDErrors.txt*

3.1.2 Database Query

Log Collection

Provide interface such that user will be able to give gunzip support bundles with particular component, timestamp and command string. Also provide interface to take only manager bundles instead of whole support bundle.

Log Extraction

Extract only relevant logs from the bundle based on the components given by user. Also extract logs which only matches the criteria given by user.

Pluggable Environment

Provide a pluggable environment such that anyone can be able to use this tool by providing the Regex of logs. Configuration file ensures that this tool is pluggable.

Log Parsing

Parse the logs based on the Regex provided in Config file. Make a structured data for each log and put it in a file. Format that file according to the input taken by ElasticSearch.

Dump in Elasticsearch

Dump all those logs in Elasticsearch after splitting the files which are greater than 100 mb. As elasticsearch only take 100mb input at a time.

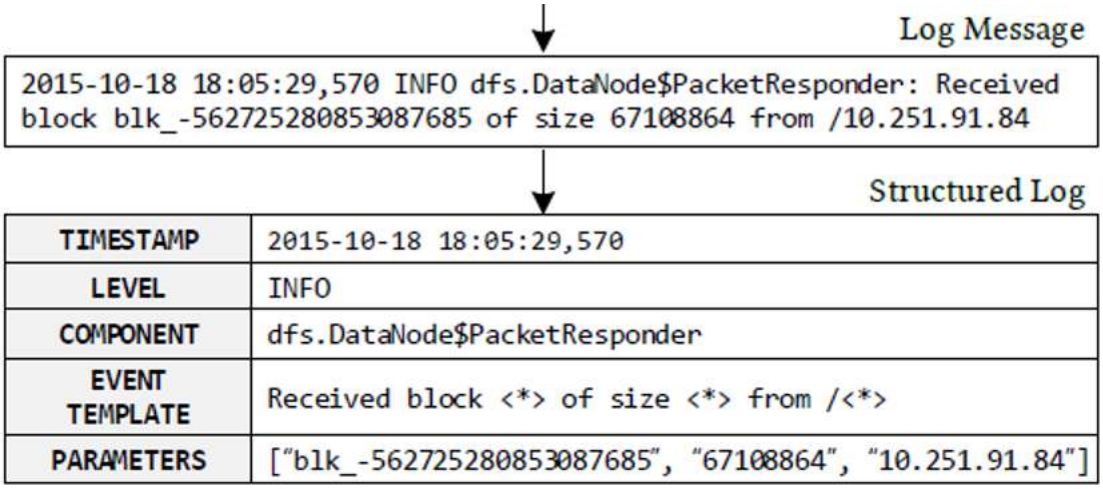

Figure 3.1: Unstructured to Structured Data

View Logs

View those logs using Kibana.

3.2 Implementation

3.2.1 Analysis

Log Analysis

Analysis part of the log bundle is done by Bash scripts and bash functions.

1. Start with a particular component, check if a call of given uuid in next component has been failed or not. If failed get the logs of that uuid's thread from that component and backtrack till previous class.Do this for all four components and put thread stack of each log in *Analysis.txt*. The unique id given will work only in one or two components, for other components we have to check for correlation happened in a log file where correlation is mentioned. Picking up that correlation we will be able to check the other component's log too. In some components we might have to convert it into decimal form and search. All this correlation,

thread stack and conversion of uuid will be put in *Analysis.txt*

2. All the Matching Logs for that uuid will be put in *MatchingLogs.txt.* Five logs above and below of matching log will also be included in *MatchingLogs.txt*

3. All the error, warning and failure logs irrespective of unique id in both manager and edge bundle will be output in *UUIDErrors.txt*

4. All the error, warning and failure logs of bundle irrespective of UUID given by user in both manager and edge bundle will be output in *ErrorsLogs.txt*

```
psbhusari$ ./Log_Analyzer.sh
Not enough arguments
Try 'Log_Analyzer.sh -help' for help
psbhusari$ ./Log_Analyzer.sh -help

Usage: Log_Analyzer.sh [OPTION] [DIRECTORY]...
Analyze Manager or edge bundle by giving a particular UUID.
Example: ./Log_Analyzer.sh -m nsx_manager_.../ -e  nsx_edge_.../ -u 123e4567-e89b-12d3-a456-426614174000

Options :

   -m               |         put before giving any manager bundle directory.
   -e               |         put before giving any edge bundle director.
   -u               |         put before giving any UUID
   -error           |         Include this option to get ErrorLogs.txt

Output  :

   Analysis.txt     |         Thread stack and deep analysis on that UUID across given bundle.
   MatchingLogs.txt |         Matching Logs for that UUID across given bundle.
   UUIDErrors.txt   |         Errors, warning and failure on that UUID across given bundle
   ErrorLogs.txt    |         Errors, warnings and failures across given bundle irrespective of given UUID

psbhusari$
```

Figure 3.2: Log Analyzer functions

3.2.2 Database Query

1. Process the given inputs from user using Configuration file. Identify if it's .tar bundle or .tgz bundle. and extract relevant files according to the yaml file specification.

2. Dump the extracted files in one directory which is created by the name of bug_number given by user.

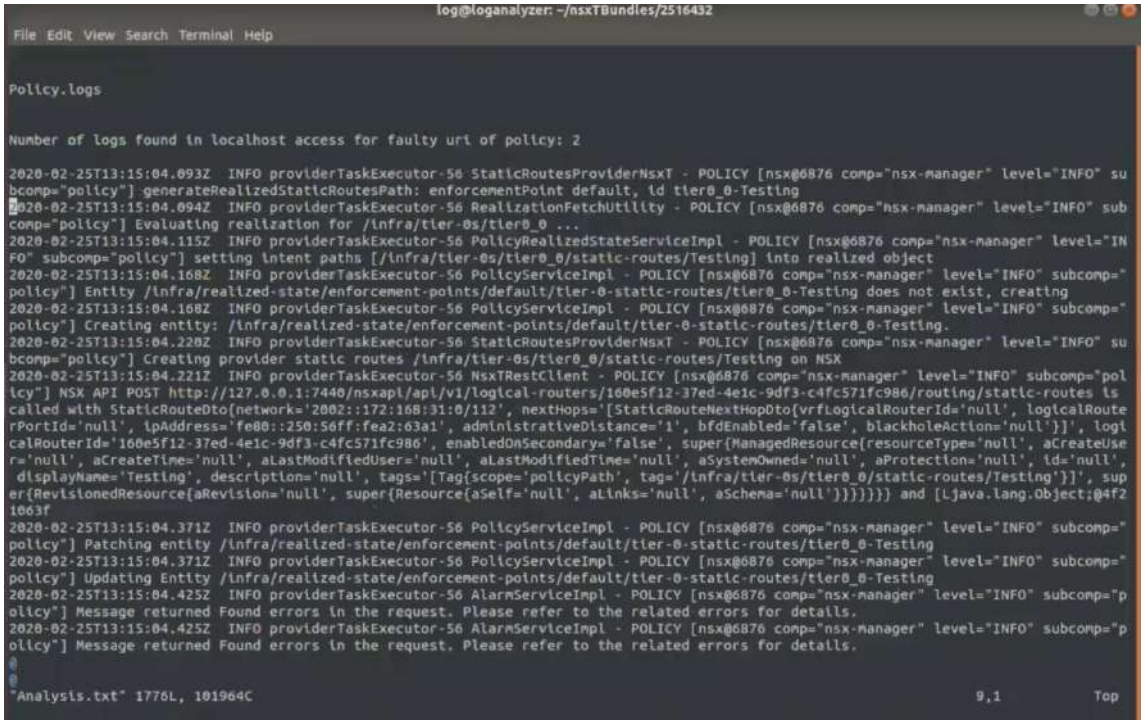

Figure 3.3: Analysis.txt overview

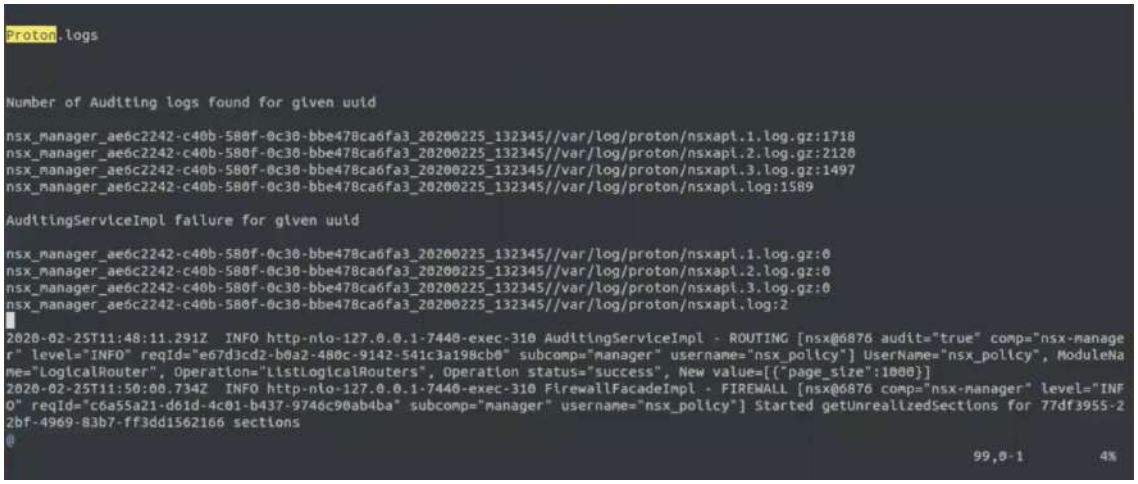

Figure 3.4: Error shown by Analysis.txt