Docker Engine Log ETL

These tools provide ways to perform ETL (Extract, Transform, Load) Docker Engine Logs

Overview

For enterprise administrators overseeing development teams, understanding the nuances of developer interactions with Docker is crucial. Docker generates a vast array of logs that chronicle developer activities. The tools in this repository are designed to help parse through these log files in order to provide actionable data, such as most frequently pulled images, preferred registries, container lifecycle events, and more. These tools attempt to be as agnostic as possible so they can be used for the development of more robust tooling.

The overall goal is to equip administrators with actionable insights, enabling them to identify potential bottlenecks, streamline workflows, and optimize the developer experience at scale. This data-driven approach is a valuable asset for any admin aiming to enhance efficiency and productivity, especially in large development environments.

Existing Documentation

Please see the Understanding Docker Logs walkthrough for more detailed information on the location and contents of the Docker Logs along with links to the official Docker log documentation.

Important Notes

Be aware that Docker rotates the logfiles *frequently*, so any usage of tooling that relies on keeping a file descriptor (fd) open, such as `tail -f` will stop working when the logfile is rotated.

Python Example

The Python directory contains three python files:

- `parse_log.py` will parse the Docker log file and produce human or syslog readable output.
- `sample_data.py` provides sample data to test the python program with.
- `test_parse_log.py` runs a test on the program to ensure it is parsing the data correctly.

Windows Examples

Sending Data to the Windows Event Log

Windows administrators are invariably going to ask for important logs to be imported into the Windows Event Viewer in order to better integrate with their tooling and processes. This requires reading the existing text files, parsing them, and then loading them into the Windows Event system.

While there are tools that can do this, the best tooling is commercial (NXLog and Eventlog Explorer) or difficult to setup, run, and administer (Logstash and Filebeat).

Two potential ways of accomplishing this are by using a Windows Powershell Script or by writing a C# program. These can then be run as a scheduled service or as a service managed by Windows Service Manager.

Powershell

This PowerShell script is tailored for administrators aiming to integrate Docker logs into the Windows Event Log system. Here's a step-by-step breakdown of its functionality:

1. Parameters Initialization:

- `$logFile` : Specifies the path to the Docker log file you wish to process.
- `$logName` : Defines the name of the Event Log (default is "**Application**").
- `$source` : Sets the source identifier for the log entries, which is "**DockerLogParser**" by default.

2. Source Verification:

- The script checks if the specified `$source` exists within the Event Log. If it doesn't, the script creates a new event source using the provided `$logName` and `$source` .

3. Log Processing:

- The script reads the Docker log file line-by-line using `Get-Content` .
- For each line, it employs a regular expression match to extract key details like timestamp, source, severity, and the actual message.
- The severity from the Docker log is then mapped to a corresponding Windows Event Log `EntryType` (Information, Warning, or Error).

4. Event Log Writing:

- A formatted log message is constructed, combining the extracted timestamp, source, and message.
- This message is then written to the Windows Event Log using `Write-EventLog` , with the specified log name, source, and determined entry type.
-

Source Code Location

The code can be found in the windows directory as LogsToEventlog.ps1.

C#

The DockerLogToEventLog program is a C# utility designed to facilitate the integration of Docker logs into the Windows Event Log system. Here's a detailed breakdown of its functionality:

1. Parameter Verification:

- The program expects three command-line arguments: the path to the Docker log file, the name of the Event Log, and the source identifier for the log entries.
- If the correct number of arguments isn't provided, the program outputs a usage message and exits.

2. Event Source Initialization:

- The program checks if the specified source identifier exists within the Event Log. If it doesn't, a new event source is created using the provided log name and source identifier.

3. Log File Reading:

- The program attempts to open the specified Docker log file for reading. It uses a `FileStream` in combination with a `StreamReader` to read the file, ensuring compatibility with files that might still be written to (using `FileShare.ReadWrite`).

- If any IO exceptions occur during this process, they are caught, and a corresponding error message is displayed. The program then exits.

4. Log Parsing:

- The program employs a regular expression to parse each line of the log file. This regex extracts key details like timestamp, source, severity, and the actual message.
- The severity from the Docker log is mapped to a corresponding Windows Event Log `EventLogEntryType` (Information, Warning, or Error).

5. Event Log Writing:

- For each successfully parsed log line, a formatted message is constructed, combining the extracted timestamp, source, and message.
- This message is then written to the Windows Event Log using `EventLog.WriteEntry`, with the specified source identifier and determined entry type.

Source Code Location

The code can be found in the windows directory as LogsToEventlog.cs.

Caveats and Warnings

- The Windows Event Log is difficult to manipulate in terms of the timestamp that is logged; because of this both of these utilities put the actual timestamp in the body of the message. Specifically, the `System.Diagnostics.EventLogEntry` class does not have a public constructor or a writable `TimeGenerated` property, which means you can't instantiate an object of this class directly nor set its timestamp.
- The C# program uses the `FileShare.ReadWrite` option, which allows you to read a file even if another process is writing to it. Be cautious: reading a file that's being written to can sometimes lead to reading incomplete data.

SOFTWARE DISCLAIMER

This software is provided as a "Proof of Concept" (PoC) and is not intended for production use.

NO WARRANTIES: The author expressly disclaims any warranty for this software. The software and any related documentation is provided "as is" without warranty of any kind, either express or implied, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. The entire risk arising out of use or performance of the software remains with the user.

NO LIABILITY FOR DAMAGES: In no event shall the author be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use or inability to use this product, even if the author has been advised of the possibility of such damages.

USE AT YOUR OWN RISK: This software is intended for educational or demonstration purposes only. Users are strongly cautioned against using it in production or mission-critical environments. If you choose to use the software, it is at your own discretion and responsibility to ensure that it does not cause any harm or issues to your systems or data.

MODIFICATIONS: Users are free to modify the software for their own use, but redistribution should include this disclaimer.

Always take a backup of your data and test any software in a controlled environment before any widespread use.

Citations and Helpful Information

- Logrus Logging
- Where you can find more information on the steps covered above
- EventLog Explorer
- NXLog
- Powershell Scripting
- C#
- Logstash / Filebeat