# Resource Monitoring and Analysis of Mars Relay Operations Service (MaROS) using Splunk

Kevin H. Evans[1]

Mentors: Joanna Liu[2], Brandon Sauer[2]

[1]Washington State University
[2]Jet Propulsion Laboratory/California Institute of Technology

July 29, 2019

**Abstract**

The Mars Relay Operations Service (MaROS) is a software service used to coordinate relay communication between Mars orbiters and landers. The software has several interconnected interfaces and a growing number of active users, both from NASA and ESA. In order to gain a deeper understanding of how the service is accessed, analysis tools can be applied to the data currently collected. Previously, there was no standardized method of accessing performance metrics and log data for diagnostics or analysis: traceability of issues was tedious. To gain more insight into MaROS, Splunk is an apt tool chosen for its robustness and ease of use for viewers, and the stringent security requirements imposed. Data and performance metrics are collected and streamed into Splunk real-time. Several Python scripts were created to fit the diverse range of software utilized by MaROS. Alerts were created for fault conditions and linear models are used to alert of potential future issues. This tool allows administrators to quickly view Splunk for the status of the system in its entirety and are notified of faults prior to end-users. With Splunk, developers can effectively understand how MaROS is accessed and plan development decisions based around the collected data.

# 1 Background

The Mars Relay Operations Service (MaROS) is a service used to plan the execution of relay communication between Earth, the Mars orbiters, and the Mars landers. Relay services through orbiters are typically used rather than direct transmission to landers, as orbiters provide significantly higher data transmission rates to Earth at a much lower power cost to the landers. MaROS provides a standardized interface for planning these relay transmissions, but also stores and displays the performance metrics from past transactions [1]. This software service has been developed for over a decade and has grown significantly since its beginning, both in its codebase and number of active users. The software has several interfaces, but is accessed primarily through a web-based user interface and a command-line interface (CLI).

With potential for additional growth [2, 3], it has become critical to leverage data analysis tools and gain a deeper understanding of how the service is accessed. With this, developers can effectively understand how MaROS is used and steer development decisions around collected data. Prior to the implementation of any data analytics tool, there had been no standardized method of tracing issues throughout the complex system nor an analytics tool to gather metrics on users. Several application logs are collected and typically, text searches on these logs were performed to track errors—often simply using `grep`. This method is tedious, especially among distributed environments with several application layers and servers.

An implementation of a data analytics tool or application performance management tool (APM) can significantly reduce the time needed to trace issues [4], simplifying the workflow of system administrators and developers. These tools allow for an easily-inspectable application, reducing some burden of system administration and assisting in pinpointing errors through the application pipeline. Hundreds of these tools exist with many focused on using collected data to understand users and improve error traceability. For the applicability of these tools onto MaROS, several tools were compared using methods outlined in Section 4. This project focused primarily on tools which are fully compatible with MaROS and required minimal code changes.

# 2 Results

Splunk was found to be the most suitable and was integrated alongside MaROS. Data is streamed into Splunk indexes from several applications within three separate environments and several dashboards were created, providing an overview of the entirety of MaROS. Alerts were created to warn of current and future issues. These alerts are streamed into the chat system (Mattermost) used by the MaROS team and has shown potential to reduce response time significantly: developers and administrators are notified immediately of errors occurring on the system, including those shown in Table 2. However, results of the effectiveness of this system are still awaiting real-world tests.

Preliminary suggestions were drawn from the data collected and analyzed through Splunk. Data collected during July 2019 shows about two-thirds (65.0%) of page hits originate from the CLI, with the remaining hits from the web interface. This was collected using queries defined within Appendix A and Appendix B. However, a vast majority of CLI usage likely originates from automated scripts: as these hits typically occur on a set schedule ran daily. With the likely-automated page hits removed, 97.1% of the page hits originate from the web interface. Of the web users, Firefox and Chrome each account for a third of hits (35.8% and 34.5% respectively), the rest divided among Safari (26.0%) and other browsers ($< 3.7\%$). This is significantly different ($p < 0.01$, $N = 38452$) than the global usage of web browsers [5], with just 5.0% of the population using Firefox.

The MaROS interface previously used a Flash-based legacy interface. This interface received zero page views during July. Among all hits to the API, `/fetch/MarosPassKey/` is the post popular endpoint, receiving over a thousand requests per day. The API endpoints `/api/v2/overflightinfo/` and `/fetch/DatabaseExecuteQuery/` each receive around eight hundred requests per day. Of the overflight lookups, MAVEN (MVN) is the most popular choice, accounting for 53.1% of lookups, next being Odyssey (ODY) with 28.3% of lookups. From MAVEN, Curiosity (MSL) is its most popular lander with 75.1% of lookups and from Odyssey, InSight (NSY) accounts for 99.5% of its lookups.

# 3 Conclusions

The addition of Splunk to the tool set of the MaROS developers and administrators is warranted, aiding in the discoverability of issues and understanding of how MaROS is accessed. Splunk is in use by other organizations, both on- and off-lab, with support readily available. Splunk has shown promise by correctly pinpointing causes of errors in MaROS and the effect of those errors on the system. Data collected on the users show a significant difference when compared globally: special care should be taken when testing web applications, ensuring compatibility with users' browsers. As the legacy interface is no longer receiving page views, it can safely be removed from upcoming versions.

# 4 Data analytics tool considerations

Several factors were considered when determining the ideal candidates for the analytics tool. There are hundreds of tools which fit this use-case for MaROS, but only a select handful were considered and examined closely. Shown partly in Table 1 below, the primary factors were: compatibility with the existing MaROS software stack, contract requirements and cost, and security of the data stored. The current software stack is relatively common and as such, many tools support reading existing MaROS logs with little adjustments required. For any future additions or changes, the wide support for other applications and log formats were noted.

Another factor was the potential requirement of code changes to MaROS. Both New Relic and Azure Application Insights require integration with the Java-based server. Due to the limitations of the short ten-week internship timeline and the strict release cycle of MaROS, it would not be feasible to change any compiled code within MaROS. However, some configuration file changes were acceptable, as they would not require a new release version to be created and only a reboot of the system. This greatly affected the choices of potential analytics tools.

Several of these applications rely on cloud storage which poses some risk for sensitive data—though this issue is mitigated when selecting an government-ready storage container, often at a higher cost. Costly solutions were avoided as it would require additional work and time to start a new contract and additionally, while MaROS is growing, many of these solutions are aimed at high-traffic and high-volume applications.

Table 1: Brief overview of data analytics tools considered for MaROS

| Tool | Contract | Storage |
|---|---|---|
| Splunk | Free* | Internal |
| ELK stack | Free** | Internal |
| New Relic | Monthly | Cloud |
| Microsoft Azure App. Insights | Monthly | Cloud |
| Datadog | Monthly | Cloud |

* Free through JPL Cybersecurity group `https://jplsplunk.jpl.nasa.gov/`

** All software within the ELK stack is open-source, however hosting would be required.

Both Splunk and ELK (an acronym for *Elastic, Logstash, Kibana*) stack were outliers in several ways: both are more free-form tools and do not have a singular purpose, but rather are broad tools for data analysis and machine-generated log monitoring, whereas New Relic and Application Insights are full-

fledged APMs. Splunk and ELK ingest various types of logs and provide an interface for querying against the data, plus provide visualizations and alerting tools. These applications also do not rely on cloud storage and store data locally.

Ultimately, Splunk became the clear choice. Although Application Insights or New Relic provide much of the required functionality out-of-box, the free-form nature of Splunk and ELK stack could be used to examine other metrics later on. Creating ELK instances would incur costs if the JPL Elastic Search as a Service[1] (JESSi) is used, or if an additional hosted VM is needed for a self-hosted Elastic Search instance. The JPL Cybersecurity group provides a free[2] hosted service for Splunk, where data is minimally processed locally and forwarded to their indexing instance. This removes the burden of administrating an additional server and lowers resource usage on MaROS servers.

Splunk is used by several organizations outside of JPL, including a similar use-case by the National Ignition Facility [6]. Support is readily available through the Splunk documentation, as well as their online Q&A-style forum[3], and through Splunk administrators and users at JPL.

### 4.1 MaROS architecture

The MaROS system uses an architecture similar to many websites, implementing a Java-based backend sever being proxied through a load-balanced Apache instance. Both the web interface and CLI make use of a RESTful API through this layer. Additionally, a MySQL server is used to store persistent data. There are additional data sources used, including external mail servers and APIs, but data is not collected directly from these and some log data is exposed through the Java server logs. The Java server and web server (with some application code) are each housed within a Docker container.

There are three environments used by MaROS: development ("dev"), testing ("testbeds"), and a production ("ops") environment are implemented. Each of these environments consist of one or more load-balanced MaROS system instances.

## 5 Splunk add-ons and configuration

Splunk is extensible through the use of add-ons ("apps"). These apps are typically Python scripts which are spawned by the Splunk daemon and communicate through `stdin` and `stdout`, TCP sockets, or other interfaces. *Modular inputs* are scripts which use the Splunk SDK[4] to input data on-the-fly. Several modular inputs were created for MaROS.

[1] JESSi, `https://jessi.jpl.nasa.gov/`
[2] JPL Splunk is free as of July, 2019.
[3] Splunk Answers, `https://answers.splunk.com/`
[4] Splunk SDK, `https://github.com/splunk/splunk-sdk-python`

## 5.1 REST/Jolokia modular input

A modular input was created to poll REST[5] endpoints. Initially, this add-on was used to poll the Java server's Jolokia endpoint. This endpoint exposed JMX (Java Management Extensions) data to provide metrics on the Java Runtime Environment (JRE) and its host server. While free Splunk add-ons exist for retrieving JMX data, none were compatible with this Jolokia interface. This data includes performance metrics, as well as developer-definable counters: various queue lengths are included in this data and also processor time consumed by garbage collection in the JRE. This data is used to detect for out-of-memory conditions, where the JRE halts the main thread due to excess time consumed by garbage collection [7]. Using the same data, the JRE's processor time is monitored as well, as it was noticed that when the JRE is unresponsive, it uses zero processor time. These can both be monitored and alerts were created to raise flags during fault conditions.

Though this add-on was initially created to poll REST data, it was later extended to record response times and HTTP response statuses. Using this data collected, the availability of each server and uptime percentage can be determined. These are each displayed on a dashboard and alerts were created, monitoring the trends in these values.

## 5.2 MySQL modular input

Another modular input was created to query against MySQL servers. Several add-ons exist for querying MySQL, but all depended on DB Connect[6]. DB Connect is a Java-based add-on which is incompatible with Universal Forwarders and can only be ran on a full instance of Splunk, requiring an additional Splunk license. Using a lightweight Python script, a new add-on was created to poll MySQL with arbitrary queries and intervals. For performance metrics, data is queried against tables within `PERFORMANCE_SCHEMA` and `INFORMATION_SCHEMA`. These schemata contain statistics on all MySQL tables, including various performance counters. All MySQL processes are polled frequently and queries with long runtimes are recorded. Later, this may be changed to take advantage of the native slow and general query log, once enabled on the servers. These additional log tables are currently disabled as the MySQL is not managed directly by the MaROS team.

## 5.3 Docker status modular input

Data from the Docker daemon is collected as a modular input. This modular input polls the Docker API at a set interval and collects data relating to running containers. The API is exposed through a REST interface over a Unix socket. Data collected includes both the status of containers and resource usage of

[5] REST (Representational State Transfer) is a standardized method of accessing data over web services.

[6] Splunk DB Connect, `https://splunkbase.splunk.com/app/2686/`

the host system. Process data can additionally be collected, but is currently disabled. The process data would generally be unused and is simply excessive for monitoring status. The Docker status does not return resource utilization as percents, but rather uses several counters. A simple formula is used to determine processor utilization and uncached/cached memory utilization from these counters. Proportional processor usage is determined by calculating two differences in processor time, $\Delta t$, and the number of available processors $N$,

$$p_{\text{cpu}} = \begin{cases} \frac{\Delta t_{\text{container}}}{\Delta t_{\text{system}}} * N & \Delta t_{\text{container}}, \Delta t_{\text{system}} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The proportional memory usage is found using bytes of memory $m$,

$$p_{\text{mem}} = \frac{m_{\text{used}} - m_{\text{cached}}}{m_{\text{total}}}$$

Once this data is collected, the data was plotted in Splunk in several dashboards, visible in Figure 1 and 2. Alerts were initially created to warn of memory and processor usage beyond 50%. After the deployment to ops, it was soon changed to warn if the three-minute running average was beyond 75%. Other tests included using a Wilcoxon signed-rank test to determine if current resource usage was significantly higher than earlier values, but it was found that simpler tests were more reliable at detecting problems.

## 5.4 Installation and initial configuration of Splunk

The JPL Splunk team provides a preconfigured installer script for the Splunk Universal Forwarder (UF). As the MaROS environment uses Docker for server applications, it was decided to use Docker for housing and managing the Splunk UF instance. A Dockerfile was created based on a minimal CentOS 7 image, which uses the JPL Splunk installer and configures the UF instance. Several Splunk apps and configuration files are copied over during build-time. This method greatly reduces complexity when installing or upgrading a Splunk instance: a Makefile handles all complex commands and a deploy requires only a single command. In future versions, this may be simplified more using a hypervisor or orchestration tool for Docker, along with a registry for Docker image upgrades.

The Docker image contains several scripts, including `entrypoint.sh` and `healthcheck.sh`. `entrypoint.sh` is used as a main entry point for Docker and manages the `splunkd` process and abstracts away some permissions-related. `healthcheck.sh` is polled periodically by the Docker daemon and returns data of the health of `splunkd`.

Splunk relies on configuration files to read logs using the native file monitor. Primarily, the `props.conf` defines a sourcetype, indicating how files are read and parsed, and `inputs.conf` defines the paths of the log files. The sourcetypes may be defined in the web interface as well. Splunk has a built-in file parser which breaks events on predefined rules. Regular expressions are used to break

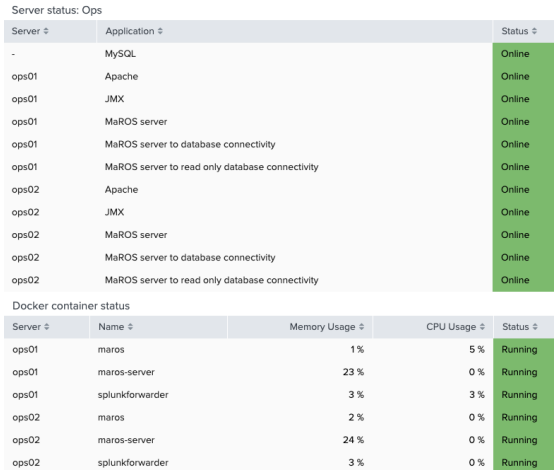
individual log entries into smaller components usable by the search engine. The regular expressions used to break the `log4j`-style events are noted in Appendix C, using the positive lookahead group-naming syntax (`?P<...>`) to name such attributes. These properties may be used in search queries directly, exemplified in Appendix B. This feature of Splunk is valuable, allowing free-form text extraction without the need for a strict-typing system, like those used in many traditional databases.

# 6 Interfaces for administrators

The Splunk web interface provides an easy-to-use and intuitive method of accessing and visualizing logs, as well as creating summaries of data and creating real-time alerts. Permissions are managed by LDAP groups and require no additional configuration. Dashboards can be created by any user and require little knowledge of the Splunk search processing language (SPL).

## 6.1 System and performance monitoring

A typical use-case of Splunk is providing a summary of a system and the ability to view performance metrics. Tables in dashboards are generated in real-time, displaying the health of the system, as shown in Figure 1. For the web services and MySQL server, an online/offline status is assigned based on the status of the last test check. The Docker containers are similar and check the time of the last-online time. These items are color-coded to give an update at a brief glance, with green representing zero issues. When problems arise, the color changes to a yellow-red scale, depending on the severity of the problem. Other metrics can easily be added by modifying the query used to display the table and using a union to append new data. When a user clicks on the table, a drilldown action may be specified. This is shown in Figure 2, in which a user selects a Docker container, navigating to a new dashboard. This new screen displays the current status and historical resource usage of the selected container.



Server status: Ops

| Server | Application | Status |
|---|---|---|
| - | MySQL | Online |
| ops01 | Apache | Online |
| ops01 | JMX | Online |
| ops01 | MaROS server | Online |
| ops01 | MaROS server to database connectivity | Online |
| ops01 | MaROS server to read only database connectivity | Online |
| ops02 | Apache | Online |
| ops02 | JMX | Online |
| ops02 | MaROS server | Online |
| ops02 | MaROS server to database connectivity | Online |
| ops02 | MaROS server to read only database connectivity | Online |

Docker container status

| Server | Name | Memory Usage | CPU Usage | Status |
|---|---|---|---|---|
| ops01 | maros | 1 % | 5 % | Running |
| ops01 | maros-server | 23 % | 0 % | Running |
| ops01 | splunkforwarder | 3 % | 3 % | Running |
| ops02 | maros | 2 % | 0 % | Running |
| ops02 | maros-server | 24 % | 0 % | Running |
| ops02 | splunkforwarder | 3 % | 0 % | Running |

FIGURE 1: Screenshot of a table displaying MaROS system health for a single environment.


FIGURE 2: Screenshot of charts displaying the status and resource usage by a Docker container.

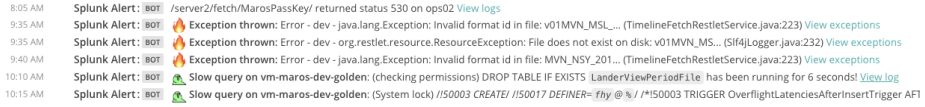



FIGURE 3: Screenshot of alerts within Mattermost, using emojis to categorize alerts.

Alerts in Splunk were created to notify the team of potential issues, such as: high resource usage, specified exceptions and errors, abnormal behavior. Splunk has several options for notifications, in particular: text message, email, and Mattermost[7]. Though most at JPL use Slack for instant messaging, Mattermost or Jabber is preferred as it is hosted internally and some sensitive data can be sent, whereas only public information may be sent over Slack. Real-time Mattermost notifications are fairly useful as they provide an outlet for low- to medium-priority alerts which do not clutter an email inbox. Figure 3 shows some example notifications within Mattermost. Emojis are used to categorize alerts and potentially severity, similar to use of colors and gradients to note severity within the system overview tables in Figure 1.

Table 2: Excerpt of alerts created in Splunk/Mattermost.

| Name | Conditions | Outcome |
|---|---|---|
| Server errors | 50X HTTP status errors | Discovery of issue related to redirection in cURL during API calls. |
| Database status | Connection lost to database | – |
| Docker status | Docker container falls offline | – |
| Docker high CPU | High average CPU usage | – |
| Java frozen | CPU time is zero or high GC time | Discovery that some large queries cause garbage collection to use significant amounts of processor time. |
| SPS failures | MPX API call failure | – |
| Slow queries | Query time greater than 3 sec. | Database backups take significant amounts of time and cause table locks. |
| Exceptions | High severity exceptions | Email parser fails completely on development environment. |

## 6.2 Java exceptions and tracing

Tracing Java exceptions can be done through Splunk. Several dashboards were created to view exceptions, as well as alerts. These dashboards allow users to view exceptions and their relative frequencies. When exceptions are selected, additional details are provided, including a stack trace and exception messages, and allows the user to view across all logs within a short time span prior to the

[7]Mattermost is an internally-hosted alternative to Slack, `https://mm.jpl.nasa.gov/`

exception thrown. Analyzing the nearby log events can aid in determining the root cause of the exception. Exceptions are additionally assigned severity values at search-time. A lookup table exists within Splunk and provides a mapping between exceptions to an integer value. This severity value is used to determine which exceptions spawn alerts and which exceptions are filtered on within the dashboard.

# 7 Issues encountered

## 7.1 Full traceability of requests

Unlike full-fledged APMs, there is no direct method of tracing events throughout the pipeline. New Relic and Azure Application Insights, and many other APMs, allow events to be broken down to a deeper level, similar to a stack trace. These APMs have this feature as an agent runs within the server, storing additional data as requests flow through the pipeline. This typically allows visibility into the exact queries ran by a request, the functions used by an application, and other details can be seen. Splunk is unable to do this as it is completely independent of the server application.

With several code changes, such as including a unique session identifier or request identifier in all log entries, across all applications, could allow searches in Splunk to associate log entries and coalesce them as one. As code changes were specifically avoided during this project, it was not feasible to do. Instead, events are loosely correlated with time and some log entries are coalesced using this method. A simple method of implementing this request identifier could be a cookie generated by the server on the first request, or a static cookie generated at the start of each request. A custom Logback handler can be implemented by extending the `ClassicConverter` class[8], which the identifier is printed during each log entry. In the Apache log, cookies can be appended to logs using a cookie specifier, like `%{MyCustomIdent}C` onto the `LogFormat` format specifier list.

## 7.2 Splunk container: failure to poll

Several days after the deployment of Splunk to the production environment, its host server was updated. The other containers were restarted manually, but the Splunk Universal Forwarder container was not started. There was some data lost: the polling scripts were unable to poll, and expected, no data could be collected for real-time metrics. Since the server applications log to files, there was no log data loss by those, as Splunk maintains a pointer to last-uploaded log entries. This was mitigated with the use of the `--restart=always` argument in the `docker run` command. This parameter to the Docker container will restart

[8] A similar implementation can be found:
`https://github.jpl.nasa.gov/gist/khevans/98be18c1a39f301ed71704a06c0054c5`

the application during failures and after server restarts. This is not the ideal solution, but rather a hypervisor should be used to manage Docker containers.

## 7.3 Add-on maintainability

Custom add-ons can be problematic for maintainability. These scripts may need to be changed if the APIs of other applications are updated to a newer version or format.

However, this is not an issue for the add-ons created for this project. The REST add-on is free-form enough to allow major changes to the Jolokia JMX API and still log metrics. The Docker API is subject to change in later versions, but the API version currently used will be supported later on. In Docker API requests, with the target API version specified, data will be returned in the target version's format. The MySQL add-on would not require updates, unless MySQL is upgraded beyond version 5.7—the API would likely remain compatible, but 8.0 drops support for several performance counters used by the add-on.

## Acknowledgements

I would like to express my appreciation to all those involved in MaROS, especially my mentors Joanna Liu and Brandon Sauer, as well as the rest of the MaROS development team, including Frank Hy and Jared Call. I would also like to thank Cathy Moroney for the incredibly helpful advice regarding presentations.

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by JPL Summer Internship program (session Summer 2019) and National Aeronautics and Space Administration (80NM0018D004).

# References


[1] Dan Allard and Roy Gladden. Mars Relay Operations Service (MaROS): Managing strategic and tactical relay for the evolving Mars network. In *2012 IEEE Aerospace Conference*, pages 1–11. IEEE, 2012.

[2] C. D. Edwards, P. R. Barela, R. E. Gladden, C. H. Lee, and R. De Paula. Replenishing the Mars relay network. In *2014 IEEE Aerospace Conference*, pages 1–13. IEEE, March 2014.

[3] C. D. Edwards, R. Gladden, C. H. Lee, and D. Wenkert. Assessment of potential Mars relay network enhancements. In *2018 IEEE Aerospace Conference*, pages 1–8, March 2018.

[4] Tarek M. Ahmed, Cor-Paul Bezemer, Tse-Hsun Chen, Ahmed E. Hassan, and Weiyi Shang. Studying the effectiveness of application performance management (apm) tools for detecting performance regressions for web applications: An experience report. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 1–12, New York, NY, USA, 2016. ACM.

[5] Wikimedia Foundation. User Agent Breakdowns, Browser Family Timeseries, July 2019.

> This contains data on the current usage of browsers that view Wikipedia. It is used to give a broad representation of global web browser usage.
> `https://analytics.wikimedia.org/dashboards/browsers/`

[6] Mikhail Fedorov, Phillip Adams, Gordon Brunton, Barry Fishler, Michael Flegel, Karl Wilhelmsen, and Eric Wilson. Leveraging Splunk for Control System Monitoring and Management. In *Proceedings, 16th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2017): Barcelona, Spain, October 8-13, 2017*, page TUCPA02, 2018.

[7] Oracle Corporation. *Java Platform, Standard Edition Troubleshooting Guide*, March 2015. Release 8.

# Appendices

## Appendix A: Retrieving browsers by family using user agents

```
eventtype=maros_web_logs_valid useragent=* host=vm-maros-ops*-golden
| eval http_user_agent=useragent
| lookup user_agents http_user_agent
| eval ua_family = IF(useragent == "Python-urllib/2.7"
                      or ua_family == "Python Requests",
                      "Python",
                      ua_family
                )
| stats count by ua_family
| where ua_family != "Python"
```

## Appendix B: Bucketing users to CLI/browser usage

```
| multisearch
        [ search eventtype = maros_web_logs
                      uri != *heartbeat*
                      uri != *MarosPassKey*
                      useragent = *
                      host = vm-maros-ops*-golden
                      file = *.php
                      status = 200
                | fields _time user uri status method host useragent
                | eval type = "web"
        ],
        [ search eventtype = maros_server_access api_uri != *heartbeat*
                      host = vm-maros-ops*-golden (NOT api_referrer=*legacy*)
                      api_uri != *MarosPassKey*
                | rename api_useragent as useragent,
                      api_method as method,
                      api_status as status
                      api_uri as uri
                      api_user as user
                | fields _time user uri status method host useragent
                | eval type = "api"
        ]
| table type _time user uri status method host useragent
| eval http_user_agent = useragent
| lookup user_agents http_user_agent
| eval is_cli = CASE(
                type == "api" AND isnull(useragent), 0,
                match(useragent, "[Pp]ython"), 1,
                match(useragent, "Mozilla"), 0,
                type == "web" AND user == "-", 1,
```

```
                type == "api" AND isnotnull(useragent), 1,
                true(), -1
        )
| eval Type = IF(is_cli == 1, "CLI hits", "Web hits")
| stats count by Type
```

## Appendix C: Regexes used for field extraction

Listing 1: Multiline `log4j`-style format, used by `logback`

```
(?m)^(?P<timestamp>\d{4}-\d{2}-\d{2}[ T]\d{2}:\d{2}:\d{2}[\.\d]{4})\s(?P<
    ↪ severity>[A-Z]+)\s\[(?P<sourceFile>[\w.]+):(?P<sourceLine>[\d]+)
    ↪ \]\s(?P<logMessage>[^\r\n]+)?(?(?=\s+?\w+\.[\w\.]+\:)(\s+?(?P<
    ↪ exception>(\w+\.[\w\.]+)):(\s+)?(?P<exceptionMessage>(?!(\s+at.+\
    ↪ s))[^\t]+)?(?P<stacktrace>(\s+at.+\s?)+)?)|[\r\n]+?(?P<
    ↪ logMessageAdditional>[\S\s]+))?)
```

Listing 2: Extraction for Restlet/API access

```
^.*\[[\w]+\.java:\d+\]\s(?P<api_clientip>[\d\.]+)\s(?P<api_user>[\w]+)?\s
    ↪ (?P<api_server_address>[^\s:]+)(:?(?P<api_server_port>\d+)\s|\s)(
    ↪ ?P<api_method>\w+)\s(?P<api_uri>[^\s]+)\s(?P<api_uri_query>[^\s]+
    ↪ )?\s(?P<api_status>\d+)\s(?P<api_bytes_sent>\d+)?\s(?P<
    ↪ api_bytes_received>\d+)?\s(?P<api_host>[^\s]+)?[ \t](?P<
    ↪ api_useragent>.+)?[ \t](?P<api_referrer>.+)?$
```