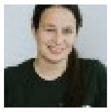
How to save hours of debugging with logs



Maya Gilad

Nov 12, 2018

Agood logging mechanism helps us in our time of need.

When we're handling a production failure or trying to understand an unexpected response, logs can be our best friend or our worst enemy.

Their importance for our ability to handle failures is enormous. When it comes to our day to day work, when we design our new production service/feature, we sometimes overlook their importance. We neglect to give them proper attention.

When I started developing, I made a few logging mistakes that cost me many sleepless nights. Now, I know better, and I can share with you a few practices I've learned over the years.

Not enough disk space

When developing on our local machine, we usually don't mind using a file handler for logging. Our local disk is quite large and the amount of log entries being written is very small.

That is not the case in our production machines. Their local disk usually has limited free disk space. In time the disk space won't be able to store log entries of a production service. Therefore, using a file handler will eventually result in losing all new log entries.

If you want your logs to be available on the service's local disk, **don't forget to use a rotating file handler.** Thiscan limit the max space that your logs will consume. The rotating file handler will handle overriding old log entries to make space for new ones.

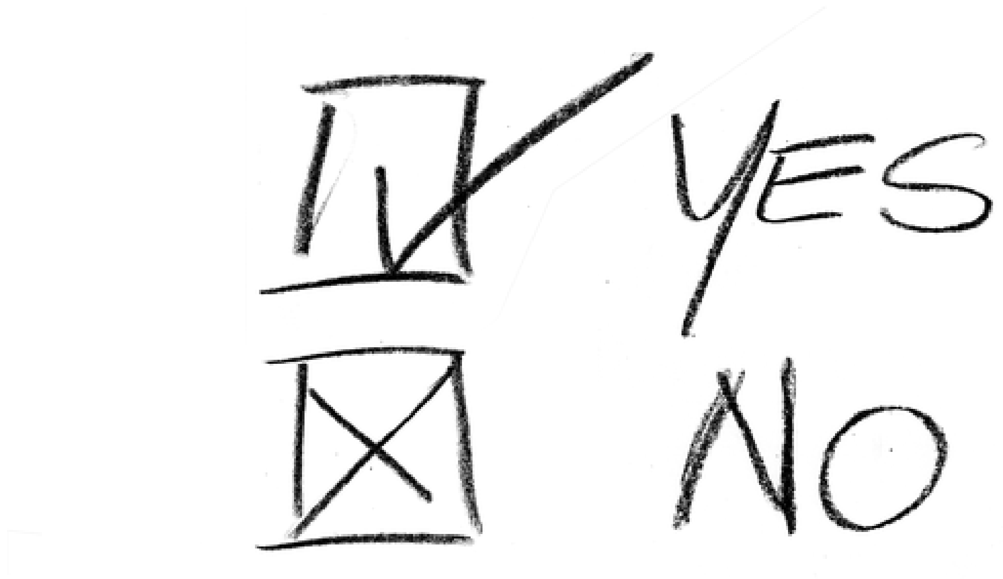
Eeny, meeny, miny, moe

Our production service is usually spread across multiple machines. Searching a specific log entry will require investigating all them. When we're in a hurry to fix our service, there's no time to waste on trying to figure out where exactly did the error occur.

Instead of saving logs on local disk, **stream them into a centralized logging system.** This allows you to search all them at the same time.

If you're using AWS or GCP—you can use their logging agent. The agent will take care of streaming the logs into their logging search engine.

To log or not log? this is the question...



There is a thin line between too few and too many logs. In my opinion, log entries should be meaningful and only serve the purpose of investigating issues on our production environment. When you're about to add a new log entry, you should think about how you will use it in the future. Try to answer this question: **What information does the log message provide the developer who will read it?**

Too many times I see logs being used for user analysis. Yes, it is much easier to write "user watermelon2018 has clicked the button" to a log entry than to develop a new events infrastructure. This is not the what logs are meant for (and parsing log entries is not fun either, so extracting insights will take time).

A needle in a haystack

In the following screenshot we see three requests which were processed by our service.

How long did it take to process the second request? Is it 1ms, 4ms or 6ms?

```
2018-10-21 22:39:07,051 - simple_example - INFO - entered request 2018-10-21 22:39:07,053 -
simple_example - INFO - entered request 2018-10-21 22:39:07,054 - simple_example - INFO - ended request
2018-10-21 22:39:07,056 - simple_example - INFO - entered request 2018-10-21 22:39:07,057 -
simple_example - INFO - ended request 2018-10-21 22:39:07,059 - simple_example - INFO - ended request
```

Since we don't have any additional information on each log entry, we cannot be sure which is the correct answer. Having the request id in each log entry could have reduced the number of possible answers to one. Moreover, **having metadata inside each log entry can help us filter the logs**and focus on the relevant entries.

Let's add some metadata to our log entry:

```
2018-10-21 23:17:09,139 - INFO - entered request 1 - simple_example
2018-10-21 23:17:09,141 - INFO - entered request 2 - simple_example
2018-10-21 23:17:09,142 - INFO - ended request id 2 - simple_example
2018-10-21 23:17:09,143 - INFO - req 1 invalid request structure - simple_example
2018-10-21 23:17:09,144 - INFO - entered request 3 - simple_example
2018-10-21 23:17:09,145 - INFO - ended request id 1 - simple_example
2018-10-21 23:17:09,147 - INFO - ended request id 3 - simple_example
```

The metadata is placed as part of the free text section of the entry. Therefore, each developer can enforce his/her own standards and style. This will result in a complicated search.

Our metadata should be defined as part of the entry's fixed structure.

```
2018-10-21 22:45:38,325 - simple_example - INFO - user/create - req 1 - entered request
2018-10-21 22:45:38,328 - simple_example - INFO - user/login - req 2 - entered request
2018-10-21 22:45:38,329 - simple_example - INFO - user/login - req 2 - ended request
2018-10-21 22:45:38,331 - simple_example - INFO - user/create - req 3 - entered request
2018-10-21 22:45:38,333 - simple_example - INFO - user/create - req 1 - ended request
2018-10-21 22:45:38,335 - simple_example - INFO - user/create - req 3 - ended request
```

Each message in the log was pushed aside by our metadata. Since we read from left to right, we should place the message as close as possible to the beginning of the line. In addition, placing the message in the beginning "breaks" the line's structure. This helps us with identifying the message faster.

```
2018-10-21 23:10:02,097 - INFO - entered request [user/create] [req: 1] - simple_example
2018-10-21 23:10:02,099 - INFO - entered request [user/login] [req: 2] - simple_example
2018-10-21 23:10:02,101 - INFO - ended request [user/login] [req: 2] - simple_example
2018-10-21 23:10:02,102 - INFO - entered request [user/create] [req: 3] - simple_example
2018-10-21 23:10:02,104 - INFO - ended request [user/create [req: 1] - simple_example
2018-10-21 23:10:02,107 - INFO - ended request [user/create] [req: 3] - simple_example
```

Placing the timestamp and log level prior to the message can assist us in understanding the flow of events. The rest of the metadata is mainly used for filtering. At this stage it is no longer necessary and can be placed at the end of the line.

An error which is logged under INFO will be lost between all normal log entries. **Using the entire range of logging levels (ERROR, DEBUG, etc.) can reduce search time significantly**. If you want to read more about log levels, you can continue reading here.

```
2018-10-21 23:12:39,497 - INFO - entered request [user/create] [req: 1] - simple_example
2018-10-21 23:12:39,500 - INFO - entered request [user/login] [req: 2] - simple_example
2018-10-21 23:12:39,502 - INFO - ended request [user/login] [req: 2] - simple_example
2018-10-21 23:12:39,504 - ERROR - invalid request structure [user/login] [req: 1] - simple_example
2018-10-21 23:12:39,506 - INFO - entered request [user/create] [req: 3] - simple_example
2018-10-21 23:12:39,507 - INFO - ended request [user/create [req: 1] - simple_example
2018-10-21 23:12:39,509 - INFO - ended request [user/create] [req: 3] - simple_example
```

Logs analysis

Searching files for log entries is a long and frustrating process. It usually requires us to process very large files and sometimes even to use regular expressions.

Nowadays, we can **take advantage of fast search engines** such as Elastic Search and index our log entries in it. Using ELK stack will also provide you the ability to analyze your logs and answer questions such as:

1. Is the error localized to one machine? or does it occur in all the environment?
2. When did the error started? What is the error's occurrence rate?

Being able to perform aggregations on log entries can provide hints for possible failure's causes that will not be noticed just by reading a few log entries.

In conclusion, do not take logging for granted. On each new feature you develop, think about your future self and which log entry will help you and which will just distract you.

Remember: your logs will help you solve production issues only if you let them.