Logs go LLD

Giedrius Stasiulionis

Me

The one who loves monitoring, programming and automation

Logs

Depending on your environment, logs might be one of the richest data sources for monitoring

Applications, webservers, OS, periodical processes… All of them log something

Zabbix & logs out-of-the-box

log[]

log.count[]

Perfect items for many use cases!

Zabbix & logs out-of-the-box

log[]

log.count[]

But is this enough?

Zabbix & logs out-of-the-box

log[]

log.count[]

For many use cases (or with lots of manual work) – yes, that is enough

Zabbix & logs out-of-the-box

log[]

log.count[]

For more complex needs – no, that is not enough

Solution?

Logs go LLD!

Concept: high level overview

LLD relies on several ideas

- KISS (UserParameter + bash script)
- regexps \S+ are your friends!
- capturing groups (\S+) are your best friends!
- Perl regexp flavor – to be most flexible

Concept: the amount of data

Since discovery is typically being run way less frequently than data collection, it would be too painful (performance wise) to analyze full slices of logs between each discovery run

Imagine log file which is updated with ~100k lines per minute. If you would run your discovery once an hour for full data window between two runs, that would be 6 million lines to process. Depending on particular regexp, it might take too long

Concept

So using this approach relies on empirical knowledge about your logs, finding the best balance for given log file on:

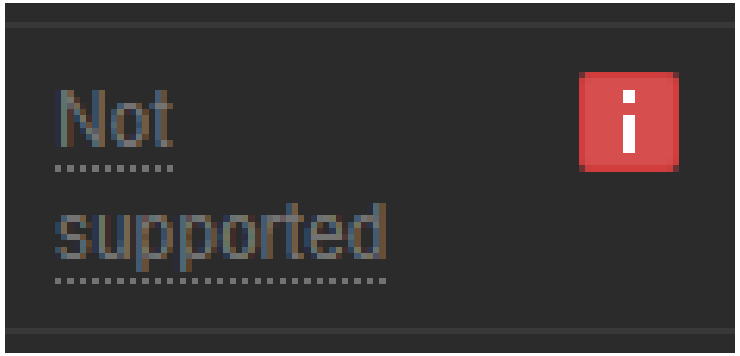
- the amount of data (lines) being processed
- frequency of running the script
- best regexps for specific case

Implementation

```
UnsafeUserParameters=1
UserParameter=log.discovery[*],/etc/zabbix/zabbix_agentd.d/zbx_scripts/log_discovery.sh '$1' '$2' '$3' '$4'
```

```
log_file="${1}"
lines="${2}"
entity_key="${3}"
pattern="${4}"
```

Implementation: UnsafeUserParameters



Implementation: essence

```
while read line; do

  result="${result}$(get_json_body_line ${line})"

done <<< "$(tail -${lines} "${log_file}" | grep -Po "${pattern}" | perl -ne 'while ($_ =~
/'"${pattern}"'/g) { print join(" ", map { $_ // "" } ($1, $2, $3, $4, $5, $6, $7, $8, $9)), "\n"; }' |
sort -u)"
```

Example no. 1

You are interested in counting all requests that appear in your webserver log, grouped by HTTP status codes

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" \"%{Host}i%U%q\" %D" combined
```

```
78.58.57.233 - - [13/Jul/2023:20:54:59 +0300] "GET /sk HTTP/1.1" 302 209 "-" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36" "stelmuze.lt/sk" 150
```

Example no. 1: configuration

```
[root@559953 zbx_scripts]# ./log_discovery.sh /var/log/httpd/access_log 1000 CODE "HTTP\/\d\.\d.\s(\d{3})"
| jq '.'
[
  {
    "{#CODE}": "200"
  },
  {
    "{#CODE}": "301"
  },
  {
    "{#CODE}": "302"
  },
  {
    "{#CODE}": "304"
  },
  {
    "{#CODE}": "404"
  }
]
[root@559953 zbx_scripts]#
```

Example no. 1: configuration

* Name	Discovery HTTP status codes
Type	Zabbix agent (active)
* Key	log.discovery[{$ACCESS_LOG},1000,CODE,"HTTP\/\d\.\d.\s(\d{3})"]
* Update interval	10m

* Name	Count of {#CODE} in {$ACCESS_LOG} (per 1 minute)
Type	Zabbix agent (active)
* Key	log.count[{$ACCESS_LOG},HTTP\/\d\.\d.\s{#CODE},,10000,skip] Select
Type of information	Numeric (unsigned)
Units	
* Update interval	1m

Example no. 1: result

Host	Name	Last check	Last value	Change	
Zabbix server	- other - (5 Items)				
	Count of 200 in /var/log/httpd/access_log (per 1 minute)	2023-07-11 18:32:39	61	+35	Graph
	Count of 301 in /var/log/httpd/access_log (per 1 minute)	2023-07-11 18:32:39	0		Graph
	Count of 302 in /var/log/httpd/access_log (per 1 minute)	2023-07-11 18:32:39	5	+5	Graph
	Count of 304 in /var/log/httpd/access_log (per 1 minute)	2023-07-11 18:32:39	26	+12	Graph
	Count of 404 in /var/log/httpd/access_log (per 1 minute)	2023-07-11 18:32:39	1	+1	Graph

Displaying 5 of 5 found

Example no. 2

Same log, you want durations of all requests based on HTTP method, HTTP status code and specific domain

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" \"%{Host}i%U%q\" %D" combined
```

```
78.58.57.233 - - [13/Jul/2023:20:54:59 +0300] "GET /sk HTTP/1.1" 302 209 "-" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36" "stelmuze.lt/sk" 150
```

Example no. 2

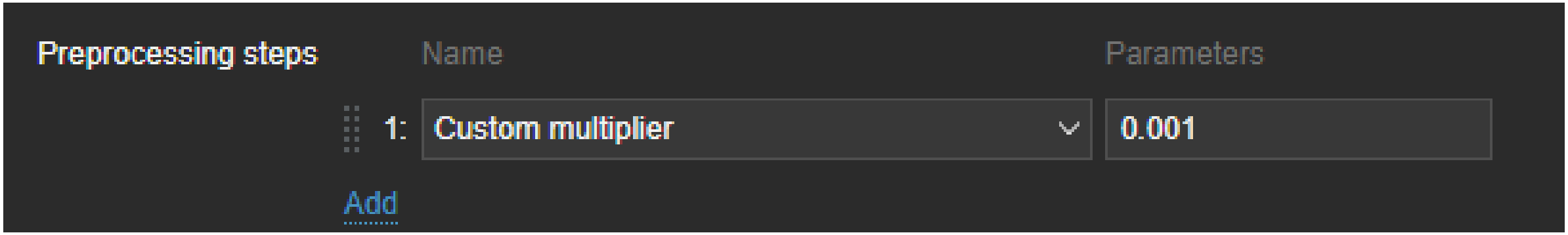
```
[root@559953 zbx_scripts]# ./log_discovery.sh /var/log/httpd/access_log 10000 COMPONENT ":\d{2}\s.*]\s.
(\S+)\s\/.*\sHTTP\/\d\.\d.\s(\d{3})\s.*\"(\S+?)\/.*\"\s(?:\d+)$" | jq '.'
[
  {
    "{#COMPONENT_1}": "GET",
    "{#COMPONENT_2}": "200",
    "{#COMPONENT_3}": "*******.******.lt"
  },
  {
    "{#COMPONENT_1}": "GET",
    "{#COMPONENT_2}": "200",
    "{#COMPONENT_3}": "**********.lt"
  },
  {
    "{#COMPONENT_1}": “POST",
    "{#COMPONENT_2}": “404",
    "{#COMPONENT_3}": "****.lt"
  },
…
```

Example no. 2: configuration

```
log.discovery[{$ACCESS_LOG},10000,COMPONENT,":\d{2}\s.*]\s.(\S+)\s\/.*\sHTTP\/\d\.\d.\s(\d{3})\s.*\"
(\S+?)\/.*\"\s(?:\d+)$"]
```

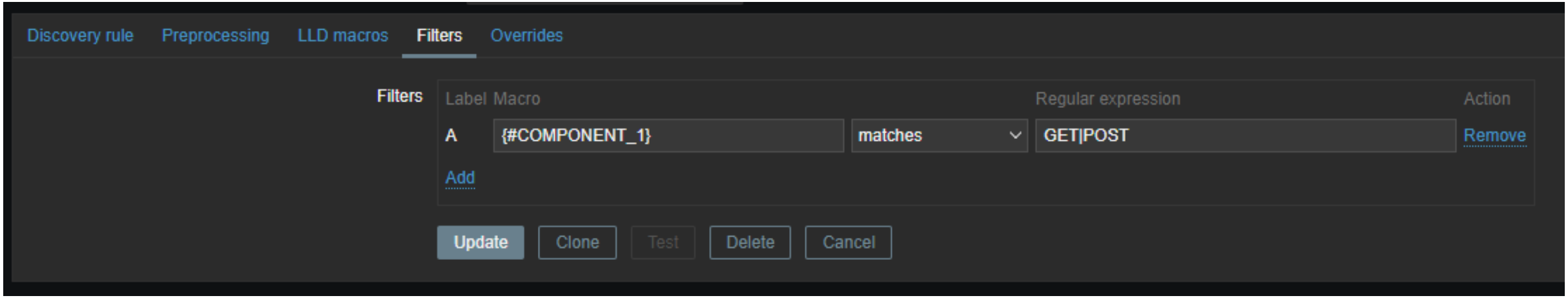
```
log[{$ACCESS_LOG},":\d{2}\s.*]\s.{#COMPONENT_1}\s\/.*\sHTTP\/\d\.\d.\s{#COMPONENT_2}\s.*\"
{#COMPONENT_3}\/.*\"\s(\d+)$",,1000,skip,\1]
```

Example no. 2: configuration



Example no. 2: configuration

Don’t forget about further possible data transformations!



Example no. 2: configuration

Discovery HTTP request durations: Request duration - GET - 404

Discovery HTTP request durations: Request duration - GET - 412

Discovery HTTP request durations: Request duration - HEAD - 200

Discovery HTTP request durations: Request duration - HEAD - 301

Discovery HTTP request durations: Request duration - HEAD - 302

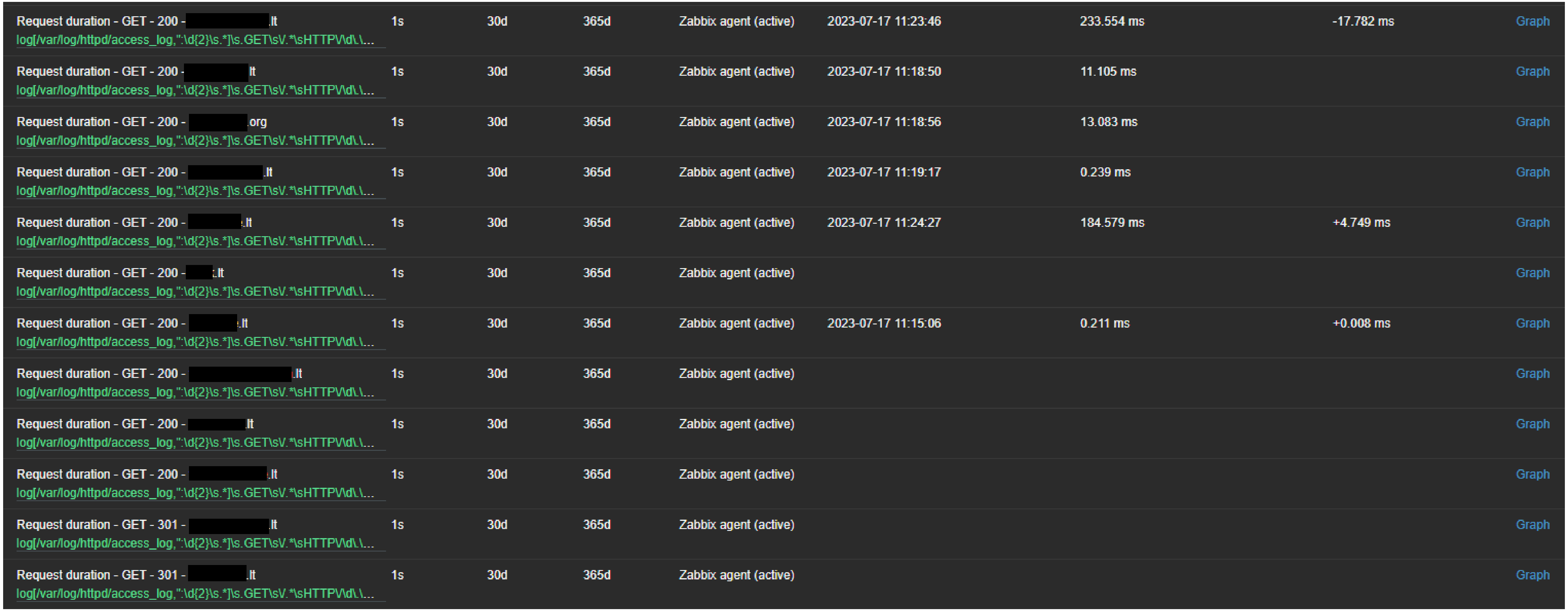
Discovery HTTP request durations: Request duration - HEAD - 404

Discovery HTTP request durations: Request duration - OPTIONS - 302

Discovery HTTP request durations: Request duration - POST - 200

Discovery HTTP request durations: Request duration - POST - 301

Example no. 2: result



Request duration - GET - 200 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)	2023-07-17 11:23:46	233.554 ms	-17.782 ms	Graph
Request duration - GET - 200 - [redacted]lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)	2023-07-17 11:18:50	11.105 ms		Graph
Request duration - GET - 200 - [redacted].org log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)	2023-07-17 11:18:56	13.083 ms		Graph
Request duration - GET - 200 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)	2023-07-17 11:19:17	0.239 ms		Graph
Request duration - GET - 200 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)	2023-07-17 11:24:27	184.579 ms	+4.749 ms	Graph
Request duration - GET - 200 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)				Graph
Request duration - GET - 200 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)	2023-07-17 11:15:06	0.211 ms	+0.008 ms	Graph
Request duration - GET - 200 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)				Graph
Request duration - GET - 200 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)				Graph
Request duration - GET - 200 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)				Graph
Request duration - GET - 301 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)				Graph
Request duration - GET - 301 - [redacted].lt log[/var/log/httpd/access_log,":\d{2}\s.*]\s.GET\s\/.*\sHTTP\/\d\.\...	1s	30d	365d	Zabbix agent (active)				Graph

Example no. 2: result

Explaining the pipe logic

We create discovery json based on this:

```
"$(tail -${lines} "${log_file}" | grep -Po "${pattern}" | perl -ne 'while ($_ =~ /'"${pattern}"'/g) { print
join(" ", map { $_ // "" } ($1, $2, $3, $4, $5, $6, $7, $8, $9)), "\n"; }' | sort -u)"
```

Explaining the pipe logic: first step

First of all, you tail desired number of lines from your log file:

```
"$(tail -${lines} "${log_file}" | grep -Po "${pattern}" | perl -ne 'while ($_ =~ /'"${pattern}"'/g) { print
join(" ", map { $_ // "" } ($1, $2, $3, $4, $5, $6, $7, $8, $9)), "\n"; }' | sort -u)"
```

Explaining the pipe logic: second step

Next, simplify things for further processing by grepping only pattern matching ones:

```
"$(tail -${lines} "${log_file}" | grep -Po "${pattern}" | perl -ne 'while ($_ =~ /'"${pattern}"'/g) { print
join(" ", map { $_ // "" } ($1, $2, $3, $4, $5, $6, $7, $8, $9)), "\n"; }' | sort -u)"
```

Explaining the pipe logic: third step

Most important step – print only what is matched by capturing groups:

```
"$(tail -${lines} "${log_file}" | grep -Po "${pattern}" | perl -ne 'while ($_ =~ /'"${pattern}"'/g) { print
join(" ", map { $_ // "" } ($1, $2, $3, $4, $5, $6, $7, $8, $9)), "\n"; }' | sort -u)"
```

Explaining the pipe logic: fourth step

Final step – make sets of caught entities unique:

```
"$(tail -${lines} "${log_file}" | grep -Po "${pattern}" | perl -ne 'while ($_ =~ /'"${pattern}"'/g) { print
join(" ", map { $_ // "" } ($1, $2, $3, $4, $5, $6, $7, $8, $9)), "\n"; }' | sort -u)"
```

Explaining the pipe logic

So in this pipeline, line is transformed from:

```
78.58.57.233 - - [13/Jul/2023:20:54:59 +0300] "GET /sk HTTP/1.1" 302 209 "-" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36" "stelmuze.lt/sk" 150
```

into just:

```
GET 302 stelmuze.lt
```

Explaining the pipe logic

This transformation ensures the speed

Anything highly repetitive in your slice of data (like HTTP 200 for some domain) but having something dynamic and different in between of capturing groups (like user agent in my example) doesn’t matter at all!

What matters is only the output of capturing groups!

Explaining the pipe logic

I have 44184 “unique” lines after initial "grep -Po", since I left time in the beginning and user agent is in between, all for the sake of demonstration:

```
[root@559953 zbx_scripts]# grep -Po ":\d{2}\s.*]\s.(\S+)\s\/.*\sHTTP\/\d\.\d.\s(\d{3})\s.*\"
(\S+?)\/.*\"\s(?:\d+)$" /var/log/httpd/access_log | sort | uniq | wc -l
44184
```

At this ("grep -Po") point, those “unique” lines will look like:

```
:59 +0300] "GET /sk HTTP/1.1" 302 209 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36" "stelmuze.lt/sk" 150
```

Explaining the pipe logic

After printing just needed data (output of capturing groups) and sorting unique entries from it, we get the true uniqueness we want – just 68 sets of data:

```
[root@559953 zbx_scripts]# ./log_discovery.sh /var/log/httpd/access_log 50000 COMPONENT ":\d{2}.*]\s.
(\S+)\s\/.*\sHTTP\/\d\.\d.\s(\d{3})\s.*\"(\S+?)\/.*\"\s(?:\d+)$" | jq '. | length'
68
```

Explaining the pipe logic

And all of this is done in just around a second!

```
[root@559953 zbx_scripts]# time ./log_discovery.sh /var/log/httpd/access_log 50000 COMPONENT ":\d{2}.*]\s.
(\S+)\s\/.*\sHTTP\/\d\.\d.\s(\d{3})\s.*\"(\S+?)\/.*\"\s(?:\d+)$" | jq '. | length'
68

real    0m0.937s
user    0m1.245s
sys     0m0.081s
```

Customizing this custom LLD

Given idea / LLD script can be used “as is” but it can be customized further for specific needs or use cases

What if…

What if out of discovered entities we don’t need actual matches, but rather we want to group it all somehow…

For instance, what if in previous example we wouldn’t need each and every HTTP status code, but we would like to have only 2XX and “the rest”, at the same time, two other capturing groups should give all possible matches?

Modified version

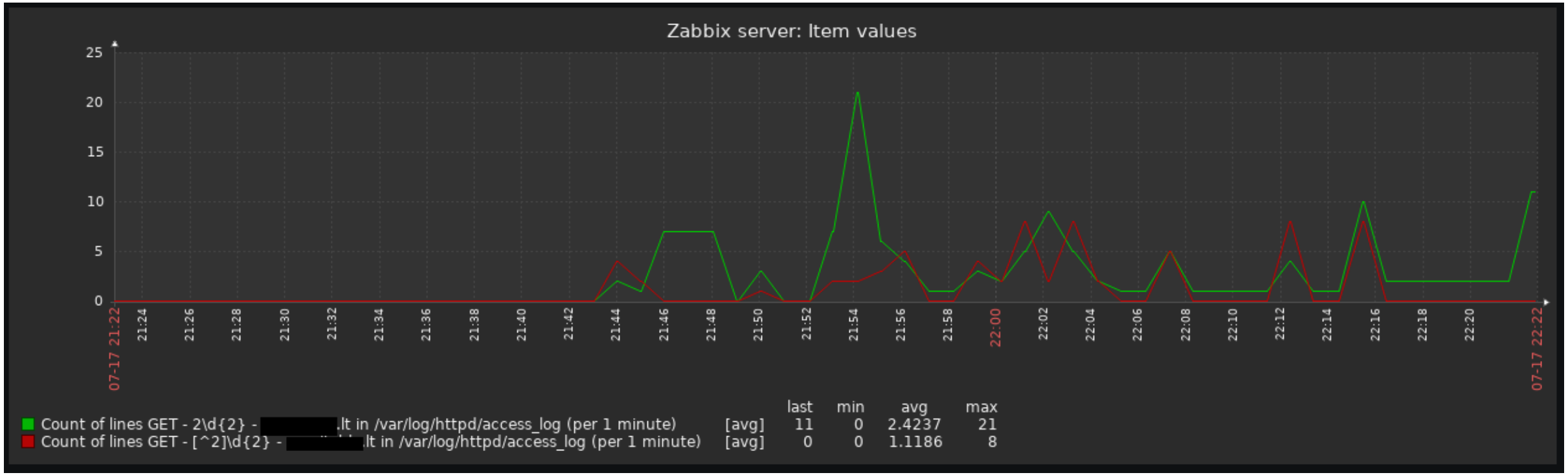
Additional parameter will solve this and similar tasks. That parameter will have some static or regexp alternatives for each capturing group (or none if actual matches are needed), as in our example:

```
;2\d{2}|[^2]\d{2};
```

Modified version: result

```
[root@559953 zbx_scripts]# ./log_discovery_groups.sh /var/log/httpd/access_log 10000 COMPONENT
":\d{2}\s.*]\s.(\S+)\s\/.*\sHTTP\/\d\.\d.\s(\d{3})\s.*\"(\S+?)\/.*\"\s(?:\d+)$" ";2\d{2}|[^2]\d{2};" | jq
'.'
[
  {
    "{#COMPONENT_1}": "GET",
    "{#COMPONENT_2}": "2\\d{2}",
    "{#COMPONENT_3}": "*******.******.lt"
  },
  {
    "{#COMPONENT_1}": "GET",
    "{#COMPONENT_2}": "[^2]\\d{2}",
    "{#COMPONENT_3}": "*******.******.lt"
  },
  {
    "{#COMPONENT_1}": "POST",
    "{#COMPONENT_2}": "2\\d{2}",
    "{#COMPONENT_3}": "**********.lt"
  },
…
```

Modified version: result



https://github.com/b1nary1/zabbix

Thanks!