Technische Hochschule Rosenheim
Technical University of Applied Sciences

Microservices

Monitoring and Logging

Content

Monitoring

- Monitoring has 2 objectives
  - Detect errors as soon as possible
  - Avoid occuring errors (that's even better!)
- You've to distinguish between classical monitoring systems (targeting "classical" infrastructure like physical and virtual servers) ant the "new shit"
- Depending on your environment you might need both
- Classical systems:
  - Nagios/Icinga
  - Zabbix
  - PandoraFMS
- "New shit":
  - Prometheus & Grafana
  - Graphite & Grafana
  - ELK/EFK

Monitoring principles

- Almost all classical systems are supporting polling of metrics
- A few of them also support pushing of metrics e.g. with a local agent on every server
- Both have advantages and disadvantages (discussed later on)
- None of them is a perfect solution. You'll want to achieve a "hybrid" solution by combining both principles to get all the information you need to determine if your system is healthy

Active Monitoring

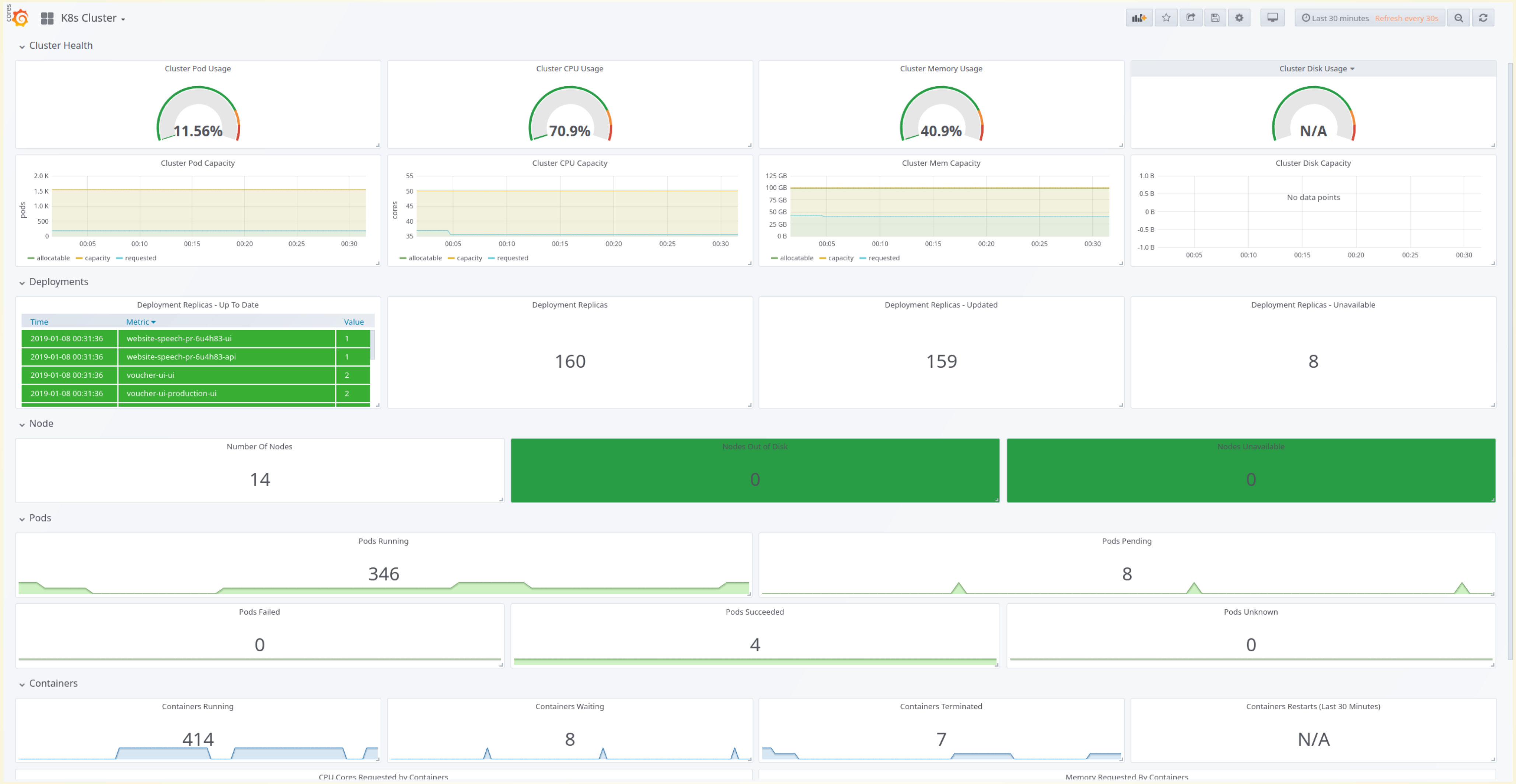
- Active monitoring (polling) has an extrinsic view of the system to monitoring (good for availability and service integrity checks e.g. is the DNS server or web server running correctly)
- It can detect if a server or a service is available and is behaving correctly
- It can't monitor resources like CPU usage, RAM usage, disk usage, disk I/O and so on (actually a few active systems do this by connecting via WMI or SSH but that’s more a hybrid solution)
- Active monitoring is agentless so you don’t have to deploy and configure any additional component to your servers

Passive Monitoring

- Passive monitoring (pushing) has an intrinsic view of the system to be monitored
- It can detect if the actual service (in terms of systemd or Windows services) is running (e.g. Nginx, bind, IIS,...)
- It can detect if the service is behaving correctly
- It is able to monitor CPU usage, RAM usage, disk usage, disk I/O...
- Passive monitoring requires a component on every system you want to observe
- If the whole server fails the central system has to recognize that the system was lost (kind of active monitoring if you like)

Application metrics

- Beside the classical monitoring metrics mentioned earlier we are required to collect metrics of our application/services
- These metrics might include:
  - Success/error response rate
  - Response time statistics (max, min, median, average)
  - memory usage
  - CPU times
  - Message bus throughput
  - ...
- Nor active neither passive monitoring is by default able to collect these metrics, you have to implement custom endpoints to serve these metrics, custom middleware to collect these metrics, ...
- In addition you might want to add domain specific metrics (e.g. how often is a user viewing his profile page, how long does it take to execute a search and so on) as this can help you to decide which features are more important than others and which features are never used



Cluster Health

Cluster Pod Usage	Cluster CPU Usage	Cluster Memory Usage	Cluster Disk Usage
11.56%	70.9%	40.9%	N/A

Deployments

Deployment Replicas - Up To Date

Time	Metric	Value
2019-01-08 00:31:36	website-speech-pr-6u4h83-ui	1
2019-01-08 00:31:36	website-speech-pr-6u4h83-api	1
2019-01-08 00:31:36	voucher-ui-ui	2
2019-01-08 00:31:36	voucher-ui-production-ui	2

Deployment Replicas	Deployment Replicas - Updated	Deployment Replicas - Unavailable
160	159	8

Node

Number Of Nodes	Nodes Out of Disk	Nodes Unavailable
14	0	0

Pods

Pods Running	Pods Pending
346	8

Pods Failed	Pods Succeeded	Pods Unknown
0	4	0

Containers

Containers Running	Containers Waiting	Containers Terminated	Containers Restarts (Last 30 Minutes)
414	8	7	N/A

CPU Cores Requested by Containers

Memory Requested By Containers

Caveats in Microservice applications

- Equivalent to load balancers (see chapter 4) systems have to picked up by the monitoring when they’re started
- Depending on the scale of your application the monitoring system has to keep track of many more systems
- To get a high perspective, your system has to aggregate the collected information to determine if a service is healthy (in term of a component of your application architecture) as it might not matter if a single instance has a high load or is unavailable for a moment if there are other instances to keep your application up and running
- To be able to analyze a certain error it’s also necessary to be able to go down the rabbit hole and inspect a single node
- As you might have many servers and services the amount of collected could be huge. Your system has to be able to handle this amount of data or aggregate it to meaningful snapshots

0
1
2
3
4
5
6
7
8
9
10

Valuation - hints

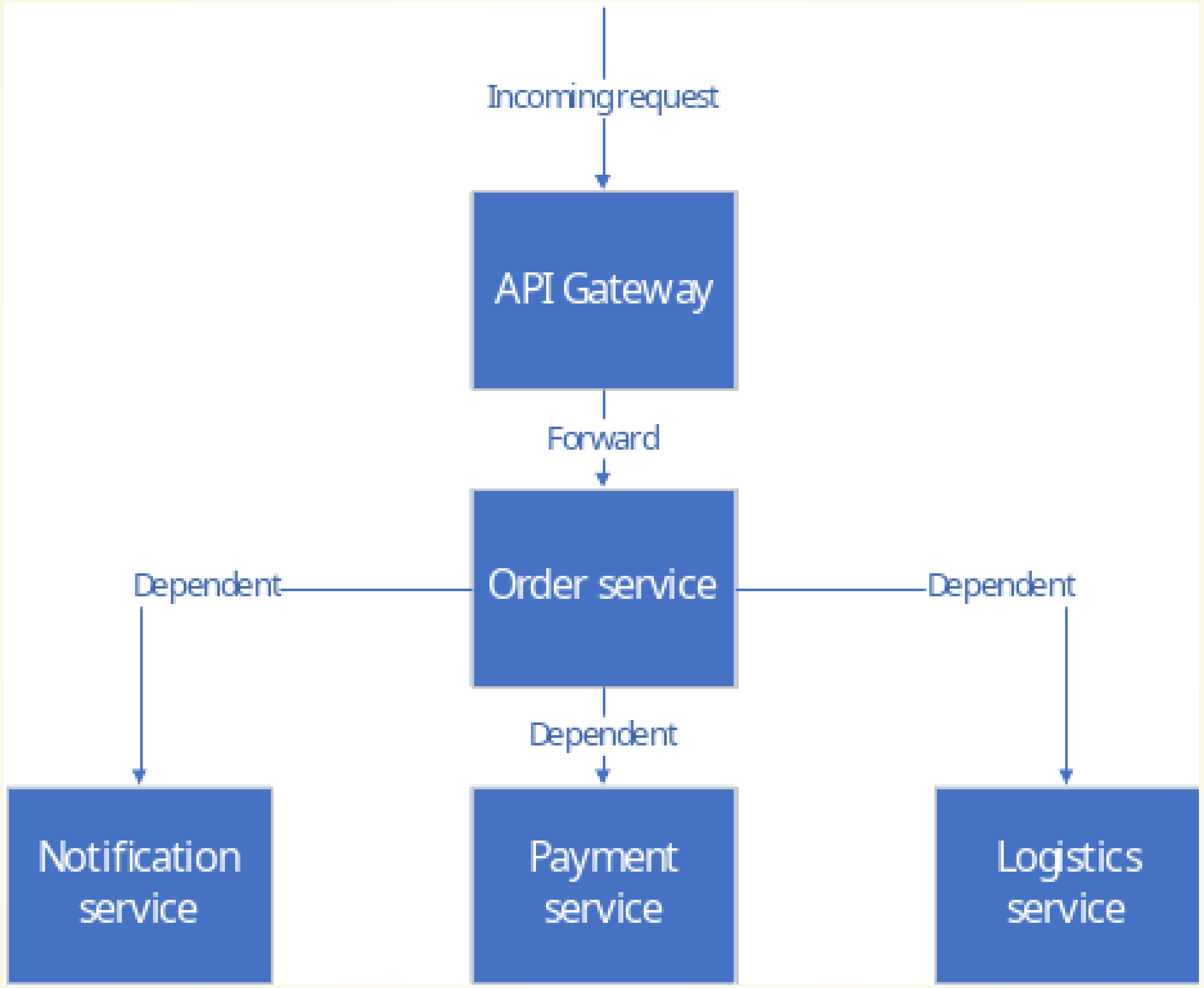
- To decide if the current value of a metric is good, bad or just okay you have to have data to compare it to
- Single datapoints might not be representative. It might be a better choice to trigger an alert only after several values.

Synthetic Monitoring

- Synthetic monitoring is about is the system working
- The advantage of synthetic monitoring is that it tests the whole system at once by executing regular tasks a regular users also executes
- This sounds a lot like behavior driven development and E2E tests, doesn’t it?
- Well, if it sounds like a duck, and smells like a duck and behaves like a duck…it is a duck! Actually you could re-use (more or less) your already existing E2E tests (you implemented E2E tests, right?) for synthetic monitoring!
- When it comes to implementing synthetic monitoring and you want to re-use your E2E tests you have to pay attention that you don’t trigger any side-effects by manipulating real user accounts, orders or anything else.
- To avoid these side effects you could have test accounts, data, … in your production environment. This way you also have a discrete set of data you know when you implement the tests. This is also helpful when it comes to response validation.

Technische Hochschule Rosenheim
Technical University of Applied Sciences

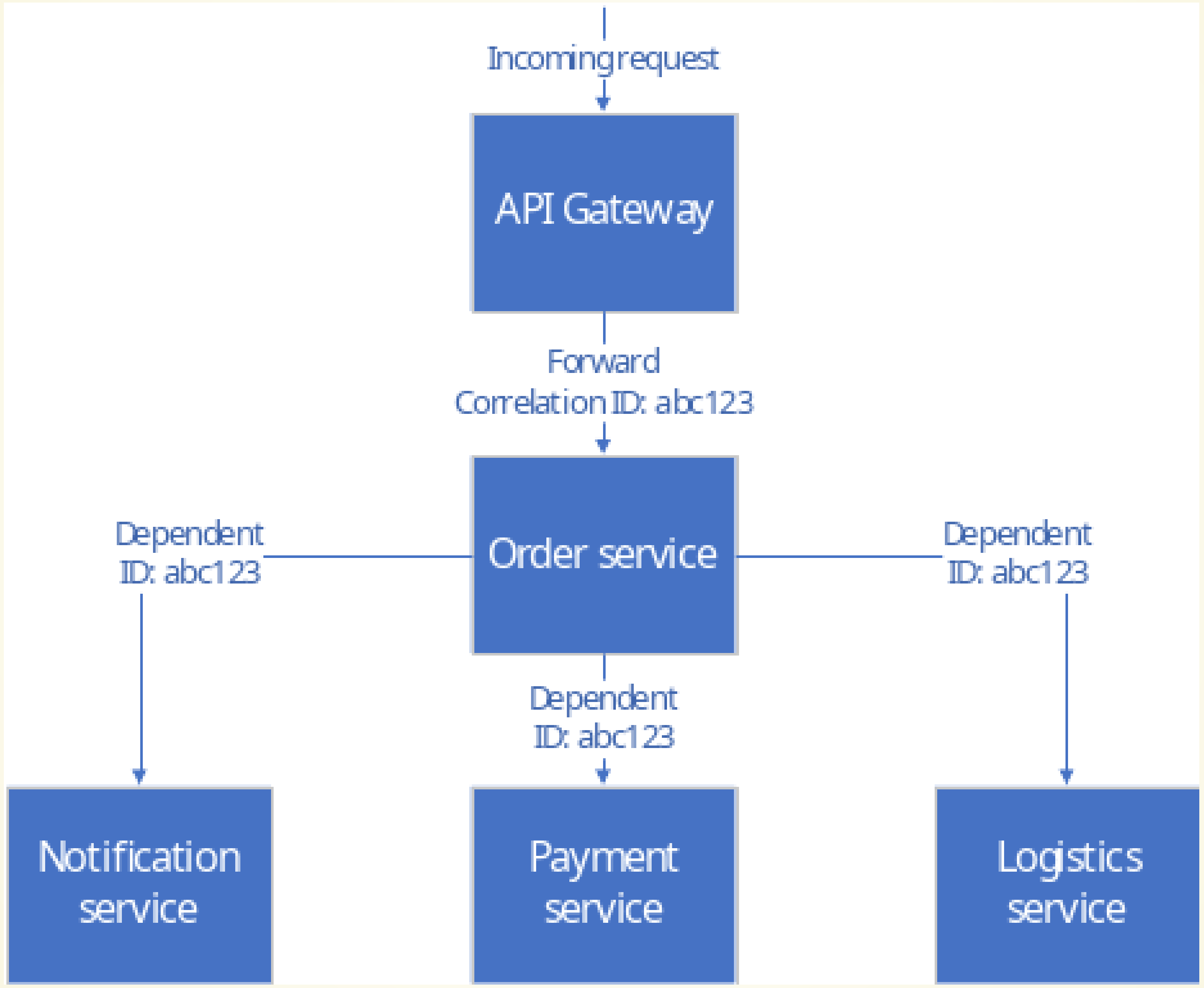
Logging


Incoming request
API Gateway
Forward
Order service
Dependent
Dependent
Dependent
Notification service
Payment service
Logistics service

Correlation IDs

- From the moment when you have more than one service in your microservice application it might be the case that an interaction of a user might result in a cascade of events in multiple services (even if you avoided direct coupling)
- When it comes to an error and you have to analyze how this error occurred you have to dig through your logs and trace the request from the beginning to the error site (think of a distributed stack trace)
- If you don’t have any indicator which entries belong to the initial request you have to analyze all logs of a certain timespan and guess which entries are belonging together

CORRELATE ALL THE THINGS
CORRELATION ID FTW
imgflip.com


Incoming request
API Gateway
Forward
Correlation ID: abc123
Order service
Dependent
ID: abc123
Dependent
ID: abc123
Dependent
ID: abc123
Notification service
Payment service
Logistics service

Correlation IDs - conclusion

- A correlation id helps to aggregate logs belonging together
- Whenever a service triggers an action of another service (directly or indirectly e.g. by enqueuing a message to a bus) it has to include the correlation id
- Tools like Kong are providing plugins or mechanisms to add correlation ids
- To avoid that a team mate forgets to include the correlation id it might be a good idea to handle the inclusion in a middleware component, a custom HTTP client wrapper, a custom event bus wrapper,…
- Tools like ZipKin help to trace requests in distributed systems (but with extra costs!)

Logging formats

- NGinx logs
- Apache2 logs
- MySQL server logs
- JSON
  - GELF
  - FluentD

Technische Hochschule Rosenheim
Technical University of Applied Sciences

NGinx logs

```
192.168.96.2 - - [02/Oct/2018:09:58:04 +0200] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0
192.168.96.2 - - [02/Oct/2018:09:58:05 +0200] "GET /favicon.ico HTTP/1.1" 404 162 "-" "
192.168.96.2 - - [02/Oct/2018:09:58:05 +0200] "GET /favicon.ico HTTP/1.1" 404 162 "-" "
192.168.97.4 - - [02/Nov/2018:10:59:32 +0100] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0
192.168.97.4 - - [02/Nov/2018:10:59:32 +0100] "GET /nice%20ports%2C/Tri%6Eity.txt%2ebak
192.168.97.4 - - [02/Nov/2018:10:59:33 +0100] "GET / HTTP/1.0" 200 612 "-" "-" "-"
```

Apache2 logs

```
127.0.0.1 - - [05/Feb/2012:17:11:55 +0000] "GET / HTTP/1.1" 200 140 "-" "Mozilla/5.0 (W
```

MySQL server logs

```
2018-05-25T01:54:33.376807Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprec
2018-05-25T01:54:33.378175Z 0 [Note] /usr/sbin/mysqld (mysqld 5.7.22) starting as proce
2018-05-25T01:54:33.380657Z 0 [Note] InnoDB: PUNCH HOLE support available
2018-05-25T01:54:33.380687Z 0 [Note] InnoDB: Mutexes and rw_locks use GCC atomic builti
2018-05-25T01:54:33.380692Z 0 [Note] InnoDB: Uses event mutexes
2018-05-25T01:54:33.380695Z 0 [Note] InnoDB: GCC builtin __atomic_thread_fence() is use
2018-05-25T01:54:33.380697Z 0 [Note] InnoDB: Compressed tables use zlib 1.2.3
2018-05-25T01:54:33.380701Z 0 [Note] InnoDB: Using Linux native AIO
2018-05-25T01:54:33.380911Z 0 [Note] InnoDB: Number of pools: 1
2018-05-25T01:54:33.380997Z 0 [Note] InnoDB: Using CPU crc32 instructions
2018-05-25T01:54:33.382234Z 0 [Note] InnoDB: Initializing buffer pool, total size = 128
2018-05-25T01:54:33.388204Z 0 [Note] InnoDB: Completed initialization of buffer pool
2018-05-25T01:54:33.389840Z 0 [Note] InnoDB: If the mysqld execution user is authorized
2018-05-25T01:54:33.399911Z 0 [ERROR] InnoDB: The innodb_system data file 'ibdata1' mus
2018-05-25T01:54:33.399937Z 0 [ERROR] InnoDB: The innodb_system data file 'ibdata1' mus
2018-05-25T01:54:33.399944Z 0 [ERROR] InnoDB: Plugin initialization aborted with error
2018-05-25T01:54:34.000567Z 0 [ERROR] Plugin 'InnoDB' init function returned error.
2018-05-25T01:54:34.000617Z 0 [ERROR] Plugin 'InnoDB' registration as a STORAGE ENGINE
2018-05-25T01:54:34.000627Z 0 [ERROR] Failed to initialize builtin plugins.
```

Graylog Extended Log Format - GELF

```json
{
  "version": "1.1",
  "host": "example.org",
  "short_message": "A short message that helps you identify what is going on",
  "full_message": "Backtrace here\n\nmore stuff",
  "timestamp": 1385053862.3072,
  "level": 1,
  "_user_id": 9001,
  "_some_info": "foo",
  "_some_env_var": "bar"
}
```

GELF - background

- Graylog supports receiving log messages via TCP and UDP
- Developed to avoid problems with classic Syslog format (limited length, no compression, too many different dialects)
- GELF spec

FluentD

```
{
    "time": 1362020400,
    "host": "192.168.0.1",
    "size": 777,
    "method": "PUT"
}
```

The only mandatory parameter in FluentD is `time` to order the gathered logs in a timely manor.