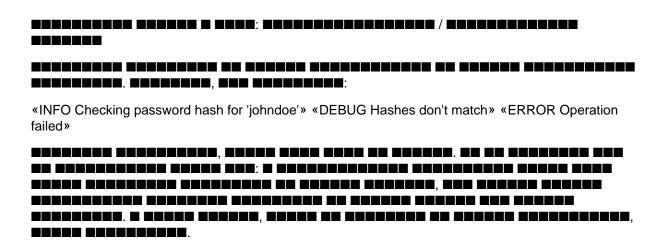
Logging

92.63.107.227 - - [04/Nov/2020:06:30:48 +0000] "GET /ru/hosted-open-vpn-server/ HTTP/1.1" 301 169 "-" "python-requests/2.11.1" "-" 92.63.107.227 - - [04/Nov/2020:06:30:49 +0000] "GET /ru/data-engineering-course/ HTTP/1.1" 301 169 "-" "python-requests/2.11.1" "-" 213.180.203.50 - - [04/Nov/2020:06:36:07 +0000] "GET / HTTP/1.1" 301 169 "-" "Mozilla/5.0 (compatible; YandexMetrika/2.0; +http://yandex.com/bots yabs01)" "-" 114.119.160.75 - - [04/Nov/2020:06:36:41 +0000] "GET /robots.txt HTTP/1.1" 301 169 "-" "(compatible; PetalBot; +https://aspiegel.com/petalbot)" "10.179.80.67" 90.180.35.207 - - [04/Nov/2020:06:47:11 +0000] "GET / HTTP/1.0" 301 169 "-" "-" "-" 46.246.122.77 - - [04/Nov/2020:06:53:22 +0000] "GET / HTTP/1.1" 301 169 "" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36" "-" 66.249.76.16 - - [04/Nov/2020:06:53:30 +0000] "GET / HTTP/1.1" 301 169 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" "-"

ERRENDE BURNE / DE API DE BURNE BURNES,
■■■■■■■■■■■■ email ■■■

2023-08-12 17:49:37 User 'johndoe' successfully logged in

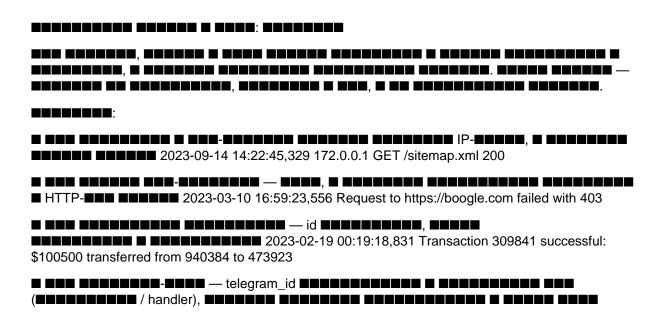
«Step initiate» «OK Transaction failed»
«User 'johndoe' successfully logged in from IP 1.2.3.4» «Transaction 1287786 reverted due to lost connection to DB»



INN NE ENGLE ENGLESSES — ENGLESSES. As simple as that.
Il func DB OK

Connection to database successfully established

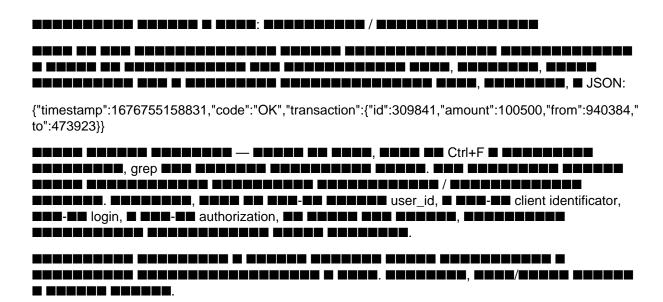
INCOMES AND ADDRESS OF THE PROPERTY OF THE PR
2022-10-02 11:41:42,612



2023-05-15 05:59:46,102 Division by zero at mymodule.divide by(0):18 ------

1885 - 2006 - 20	
;	_,
,	
2023-02-19 00:19:18 Transaction 309841 successful: \$100500 transferred from 940384 to 473923	
1888, 88888 888888 88888888, 888 88888 888888	_
I. S., SESSE SESS SESSES SESSES SESSES SESSES S	_

2023-02-19 00:19:18 Transaction successful. Transaction_id: 309841, amount: 100500, From: 940384, To: 473923



[TABLE CONTENT]

Numerical Code | Severity

- 0 | Emergency: system is unusable
- 1 | Alert: action must be taken immediately
- 2 | Critical: critical conditions
- 3 | Error: error conditions
- 4 | Warning: warning conditions
- 5 | Notice: normal but significant condition
- 6 | Informational: informational messages
- 7 | Debug: debug-level messages

THE STATE OF THE S

######################################
######################################
======================================
######################################
[TABLE CONTENT]
DEBUG 10 ***********************************

WARNING | 30 | MARKER, MAR MARKERS MARKERS MARKERS, MARKERS MA

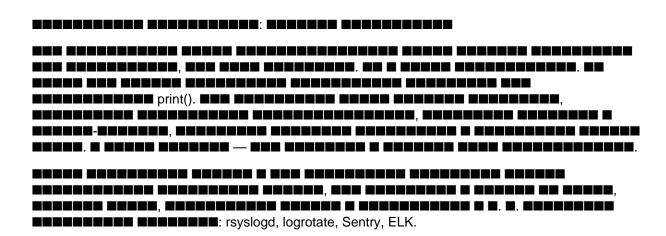
ERROR | 40 | BE-BE BERBERE BERBERE BERBERE BE BERBERE

CRITICAL | 50 | BERNESSEES BERNESSEES, BE-NESSEES BERNESSEES BERNESSEES

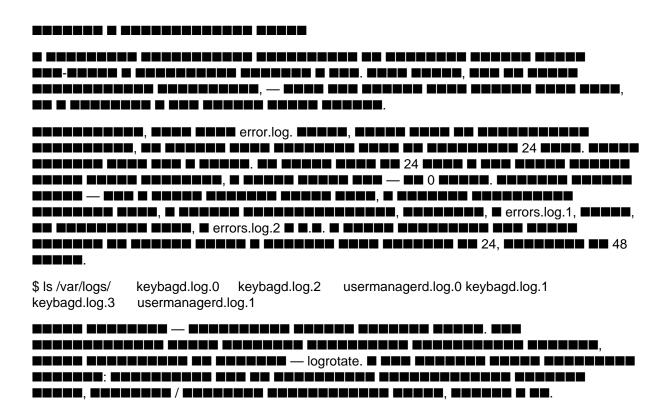
[SLIDE NOTES]

DEBUG — BEREIRE BEREIRE BEREIRE BEREIRE BEREIRE BEREIRE BER

WARNING, ERROR, CRITICAL BURNESS BURNE ENDERDO DE DESENTA DE DEBUG E INFO ENDEDDE DE DE DESENTA DE DESENT -----



,



■■■■■■ ■ ■■■■■■■■■■ : logrotate
/etc/logrotate.d/myapp:
/var/log/myapp/error.log { rotate 7 # ■■■■■■■ 7 ■■■■■■■■■■■■■■■■■■■■■■■■

Python / IIII logging
Python — SESSES Print SESSES SESSES SESSES SESSES SESSES SESSES
Den. Den. Den. Den. Den
Python Description debug(), info(), warning(), error(), critical(),
DESCRIPTION OF THE STREET OF T
WARNING. WE WARNING WARNING WARNING WARNING.
import logging logging.info('For your information') # ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
WARNING:root:I warn you!
ELECTRIC DESCRIPTION OF THE STATE OF THE STA

import logging logging.basicConfig(filename='example.log', level=logging.DEBUG) logging.debug("Let's see if it works") logging.info('For your information') logging.warning('I warn you!') logging.error('Something bad happened pls help!')
IN INDIA DEBUTE DE DEBUTE filename, I DEBUTE level INDIA DEBUTE DE DESERVIR DE DEBUTE DE DESERVIR DE DEBUTE DE DESERVIR DE DEBUTE DE DESERVIR DE DEBUTE DE DESERVIR DE DEBUTE DE DESERVIR DE DESERVIR DE DE DE DE DE DE DEBUTE DE DESERVIR DE
basicConfig()

	■■■■■■■ logging	
		
	ogging — IIIIIIIIII I I	
	•	
(logger	rs) — — — — — — — — — — — — — — — — — — —	
	, (handlers) — ■■■■■■■■	
		,
	formatters) — ■■■■■■■■	
	IDDEED BEIDE BEIDE	
_		

Logging Flow

======================================
import logging counter = '3rd' logging.warning('I am warning you for the %s time!', counter)
→ WARNING:root:I am warning you for the 3rd time!
logging.basicConfig(format='%(asctime)s %(levelname)s %(message)s') logging.warning('Right about

→ 2023-08-21 13:44:38,417 WARNING Right about time!

time!')

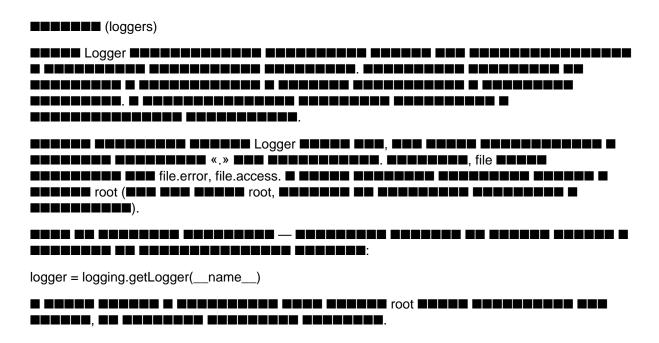


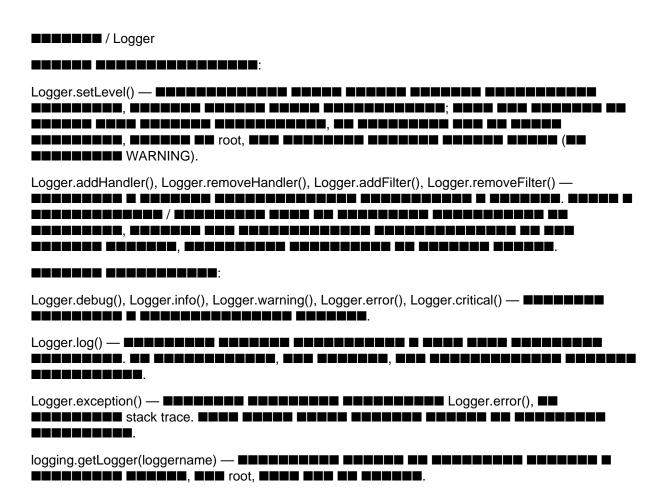
logging.basicConfig(format='%(asctime)s %(message)s', datefmt='%d.%m.%Y %H:%M:%S') logging.warning('is when this event was logged.')

 \rightarrow 21.08.2023 13:54:51 is when this event was logged.

logging.basicConfig() ■ ■■■■■ logging ■ Python ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■ ■ ■■■■■■■■■■ LogRecord attributes.
I logging.basicConfig(),

[TABLE CONTENT]





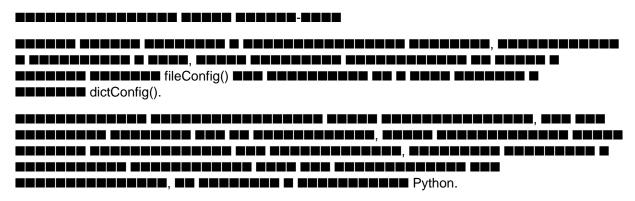
■■■■■■■■■ (handlers)
HAND, SMTPHandler HAND HANDS HANDS HANDS HANDS HANDS. HANDS
======================================
setFormatter() — INCOME
[TABLE CONTENT]
StreamHandler ###################################
FileHandler
RotatingFile- Handler
TimedRotating-FileHandler ###################################
CMTDHandlar MANAGEMENT Compiler

SysLogHandler |

■■■■■■■■ (formatters)
logging.Formatter(fmt=None, datefmt=None, style='%')
BERNE BERNESS fmt;
■■■■■ ■■■■ datefmt;
LogRecord:
'%' — ••• %-••••••••••••••••••••••••••••••••
'{' — ■■■ str.format() ■■■
'\$' — ■■■ string.Templates.
#
\rightarrow [2020-06-08 07:42:59] [logger] [DEBUG] > debug message \rightarrow [2020-06-08 07:42:59] [logger] [INFO] > info message

			 	
■■■■■■ fileC	Config().			

import logging # INFO) # INFO # INFO



import logging import logging.config # **INTERPORT OF STATE OF STAT**

logging.conf

[loggers] keys=root,exampleLogger [handlers] keys=consoleHandler [formatters] keys=simpleFormatter [logger_root] level=DEBUG handlers=consoleHandler [logger_simpleExample] level=DEBUG handlers=consoleHandler qualname=simpleExample propagate=0 [handler_consoleHandler] class=StreamHandler level=DEBUG formatter=simpleFormatter args=(sys.stdout,) [formatter_simpleFormatter] format=%(asctime)s - %(name)s - %(levelname)s - %(message)s

logging.config.dictConfig
HINDER DE HUNDE, DECENDA DE BURDE DE BURDE (BURDEDE, JSON DES YAML)
HENNE BROKE HENNES YAML:

handlers: console: class : logging.StreamHandler formatter: brief level : INFO filters: [allow_foo] stream : ext://sys.stdout file: class : logging.handlers.RotatingFileHandler formatter: precise filename: logconfig.log maxBytes: 1024 backupCount: 3

```
THE STATE OF THE S
{
"version": 1,
"disable_existing_loggers": false,
"formatters": {
"simple": {
"format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
},
"extra": {
"format": "%(asctime)s %(name)s %(filename)s %(lineno)s %(funcName)s %(levelname)s
%(message)s"
}
},
"handlers": {
"console_handler": {
"class": "logging.StreamHandler",
"level": "DEBUG",
"formatter": "simple",
"stream": "ext://sys.stdout"
},
"info_file_handler": {
"class": "logging.FileHandler",
"level": "INFO",
"formatter": "simple",
"filename": "info.log",
"encoding": "utf8"
},
"error_file_handler": {
"class": "logging.FileHandler",
```

```
"level": "ERROR",
"formatter": "extra",
"filename": "errors.log",
"encoding": "utf8"
}
},
"loggers": {
"MyApp": {
"level": "WARNING",
"handlers": ["console_handler"],
"propagate": false
},
"MyApp.MyClass1": {
"level": "DEBUG",
"handlers": ["console_handler", "info_file_handler", "error_file_handler"],
"propagate": false
},
"MyApp.MyClass2": {
"level": "ERROR",
"handlers": ["error_file_handler"],
"propagate": false
}
},
"root": {
"level": "WARNING",
"handlers": ["console_handler"]
}
}
```

test(MyClass1()) test(MyClass2())

2023-08-21 17:09:40,562 - MyApp - WARNING - warn message

2023-08-21 17:09:40,563 - MyApp - ERROR - error message

2023-08-21 17:09:40,563 - MyApp - CRITICAL - critical message

2023-08-21 17:09:40,564 - MyApp.MyClass1 - DEBUG - debug message

2023-08-21 17:09:40,565 - MyApp.MyClass1 - INFO - info message

2023-08-21 17:09:40,565 - MyApp.MyClass1 - WARNING - warn message

2023-08-21 17:09:40,566 - MyApp.MyClass1 - ERROR - error message

2023-08-21 17:09:40,566 - MyApp.MyClass1 - CRITICAL - critical message

info.log:

2023-08-21 17:16:19,753 - MyApp.MyClass1 - INFO - info message

2023-08-21 17:16:19,756 - MyApp.MyClass1 - WARNING - warn message

2023-08-21 17:16:19,756 - MyApp.MyClass1 - ERROR - error message 2023-08-21 17:16:19,756 - MyApp.MyClass1 - CRITICAL - critical message

errors.log:

2023-08-21 17:17:18,172 MyApp.MyClass1 main.py 93 test ERROR error message

2023-08-21 17:17:18,172 MyApp.MyClass1 main.py 94 test CRITICAL critical message

2023-08-21 17:17:18,173 MyApp.MyClass2 main.py 93 test ERROR error message

2023-08-21 17:17:18,173 MyApp.MyClass2 main.py 94 test CRITICAL critical message

■ MyApp (■ ■■■■■ MyAppLogger) ■■■■■ ■■■■■■ ■■■■ console_handler, ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
MyApp — WARNING, ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ MyApp.MyClass1 (■ ■■■■■ MyClass1) ■■■■■ ■■■ ■■■■■: console_handler, info_file_handler, error_file_handler. ■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ MyApp.MyClass2 (■ ■■■■■■ MyClass2) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

«
datetime
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ uppercase: USER_ADDED, DOMAIN_DELETED, MAIL_SENT, PASSWORD_RECOVERY_REQUEST, SMS_SEND_ERROR ■ ■.■.
service / script name
hostname
user_id / service_id — IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
DATA
2023-08-27 12:12:12 USER_ADDED mobappbackend cl23.mycloud.ru 23455 0 {"Name":"Ivan Ivanov","email":"ivanoff@gmail.com","regip":"1.2.3.4"}
2023-08-27 12:12:13 MAIL_SENT userspam.py cl2.mycloud.ru 11455 249499 {"email":"prtrff@gmail.com","title":"

import logging default extra fields = { 'user id': 0, 'key id': 0, 'ext params': " } class CustAdapter(logging.LoggerAdapter): def process(self, msg, kwargs): extra fields = {} for key in ['user_id', 'key_id', 'ext_params']: extra_fields[key] = kwargs.pop(key, self.extra[key]) kwargs['extra'] = extra_fields return msg, kwargs logger = logging.getLogger('actions') logger.setLevel(logging.INFO) handler = logging.StreamHandler() # handler = logging.FileHandler('/var/log/myapp.log') formatter = logging.Formatter(fmt = '%(asctime)s\t%(message)s\t%(user_id)s\t%(key_id)s\t%(ext_params)s', datefmt = '%d.%m.%Y %H:%M:%S',) handler.setFormatter(formatter) logger.addHandler(handler) logger = CustAdapter(logger, default_extra_fields) # logger.info('SOME_ACTION') logger.info('ANOTHER_ACTION', key_id = 777) logger.info('ACTION1', user_id = '123', ext_params = {'param1': 123, 'param2': ['A','B',456]}) logger.info('ACTION1', user_id = '123', key id = 77, ext params = {'par1': 123, 'par2': ['A','B',45]})

user id

key_id

ext_params

Loguru

to use out of the box without boilerplate code.»

DESCRIPTION OF THE PROPERTY OF
BERNARD. BERNARD BERNARD loguru BERNARD, BERNARD.
Zero-cost logger
Indiana Indian
Airbrake.
LESS SECURITY Loguru

Loguru

✓ Ready to use out of the box without boilerplate

from loguru import logger logger.debug("That's it, beautiful and simple logging!")

```
2023-11-13 20:29:32.531 | DEBUG | \_main\_::1 - That's it, beautiful and simple logging!
```

✓ No Handler, no Formatter, no Filter: one function to rule them all

How to add a handler? How to set up logs formatting? How to filter messages? How to set level?

logger.add(sys.stderr, format="{time} {level} {message}", filter="my_module", level="INFO")

✓ Easier file logging with rotation / retention / compression

logger.add("file_{time}.log") logger.add("file_1.log", rotation="500 MB") # Automatically rotate too big file logger.add("file_2.log", rotation="12:00") # New file is created each day at noon logger.add("file_3.log", rotation="1 week") # Once the file is too old, it's rotated logger.add("file_X.log", retention="10 days") # Cleanup after some time logger.add("file_Y.log", compression="zip") # Save some loved space

Loguru

✓ Modern string formatting using braces style

logger.info("If you're using Python {}, prefer {feature} of course!", 3.6, feature="f-strings")

- ✓ Exceptions catching within threads or main
- @logger.catch def my_function(x, y, z): # An error? It's caught anyway! return 1 / (x + y + z)
- ✓ Pretty logging with colors

logger.add(sys.stdout, colorize=True, format="{time} {message}")

```
logger.add("out.log", backtrace=True, diagnose=True) def func(a, b): return a / b def nested(c): try:
                                               logger.exception("What?!") nested(0)
    func(5, c) except ZeroDivisionError:
LINE Loguru

✓ Fully descriptive exceptions

        2018-07-17 01:38:43 | ERROR | __main__:nested:10 What?
Traceback (most recent call last):
File "test.py", line 12, in
nested(0)
> File "test.py", line 8, in nested
func(5, c)
■ ■ 0
File "test.py", line 4, in func
return a / b
■ ■ 0
5
ZeroDivisionError: division by zero
```

Loguru

✓ Lazy evaluation of expensive functions

Sometime you would like to log verbose information without performance penalty in production, you can use the opt() method to achieve this.

logger.opt(lazy=True).debug("If sink level <= DEBUG: {x}", x=lambda: expensive_function(2**64)) # By the way, "opt()" serves many usages logger.opt(exception=True).info("Error stacktrace added to the log message (tuple accepted too)") logger.opt(colors=True).info("Per message colors") logger.opt(record=True).info("Display values from the record (eg. {record[thread]})") logger.opt(raw=True).info("Bypass sink formatting\n") logger.opt(depth=1).info("Use parent stack context (useful within wrapped functions)") logger.opt(capture=False).info("Keyword arguments not added to {dest} dict", dest="extra")

✓ Better datetime handling

The standard logging is bloated with arguments like datefmt or msecs, %(asctime)s and %(created)s, naive datetimes without timezone information, not intuitive formatting, etc. Loguru fixes it:

```
logger.add("file.log", format="{time:YYYY-MM-DD at HH:mm:ss} | {level} | {message}")
```

✓ Suitable for scripts and libraries

For scripts config = { "handlers": [{"sink": sys.stdout, "format": "{time} - {message}"}, {"sink": "file.log", "serialize": True},], "extra": {"user": "someone"} } logger.configure(**config) # For libraries, should be your library's `__name__` logger.disable("my_library") logger.info("No matter added sinks, this message is not displayed") # In your application, enable the logger in the library logger.enable("my_library") logger.info("This message however is propagated to the sinks")

✓ Entirely compatible with standard logging

handler = logging.handlers.SysLogHandler(address=('localhost', 514)) logger.add(handler)

✓ Personalizable defaults through environment variables

```
\# Linux / OSX export LOGURU_FORMAT="\{time\}\ |\ \{message\}" \# Windows setx LOGURU_DEBUG_COLOR ""
```

✓ Convenient parser

pattern = r"(?P.*) - (?P[0-9]+) - (?P.*)" # Regex / named groups caster_dict = dict(time=dateutil.parser.parse, level=int) # Transform matching groups for groups in logger.parse("file.log", pattern, cast=caster_dict): print("Parsed:", groups) # {"level": 30, "message": "Log example", "time": datetime(2018, 12, 09, 11, 23, 55)}

✓ Exhaustive notifier

Loguru can easily be combined with the great notifiers library (must be installed separately) to receive an e-mail when your program fail unexpectedly or to send many other kind of notifications: Pushover, SimplePush, Slack, Gmail, Telegram, Gitter, Pushbullet, Join, Zulip, Twilio, Pagerduty, Mailgun, PopcornNotify, StatusPage.io, iCloud, VictorOps (Splunk)

import notifiers params = { "username": "you@gmail.com", "password": "abc123", "to": "dest@gmail.com" } # Send a single notification notifier = notifiers.get_notifier("gmail") notifier.notify(message="The application is running!", **params) # Be alerted on each error message from notifiers.logging import NotificationHandler handler = NotificationHandler("gmail", defaults=params) logger.add(handler, level="ERROR")

√ 10x faster than built-in logging

Zero-cost logger.

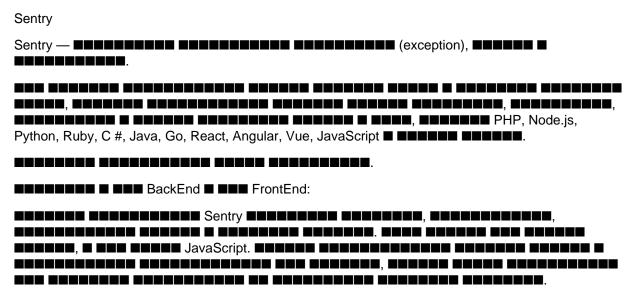
 , " , " ,
■ ■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■ ■■■■ ■■■■■■ : ■■■■■ ■■■■■■■ column-based ■■■■ (clickhouse ■ ■.■.):

Clickhouse + web logs + ■■■■■■

ENDERDED ENDERDE ENDERDE ELK
ELK – NEW
E = Elasticsearch — Apache Lucene.
L = Logstash — ***********************************
■■■■■ ■■■■■ ■■■■■■■■■ . K = Kibana Kibana — ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
ELK?
Logstash MANANAN, MANANANANAN MANANANANANANANANANANANANAN
Elasticsearch
Kibana

Graylog ------**■■■■** Graylog

BE BEREERE BELK BEREERE BEREERE BEREERE BEREE.



import logging, sentry_sdk from sentry_sdk.integrations.logging import LoggingIntegration sentry_logging = LoggingIntegration(level=logging.INFO, event_level=logging.ERROR) sentry_sdk.init(dsn="https://exampleKey@o0.ingest.sentry.io/0", integrations=[sentry_logging,],)

https://github.com/xtrueman/prog_instruments/blob/main/Logging.md

logging

Logging HOWTO

Loggint Cookbook