Structlog extensions

Release 1.0.0

Niels Albers

Sep 15, 2019

CONTENTS

Structlog extensions are a set of structlog processors and utility functions to add new logging options to structlog. The primary purpose is to supply tools to convert existing structlog logging output into Elastic Common Schema json output so users can easily plug their application output into centralised logging solutions such as ELK stack.

At present the extensions consist of a `CombinedLogParser` (which for example can be used to convert gunicorn access log output into ECS fields), and the `NestedDictJSONRenderer`, which can be used to convert the output of the CombinedLogParser into ECS json format output.

CHAPTER

ONE

USER GUIDE

1.1 Basic usage

1.1.1 Installation

Install package with pip

```
pip install structlog-extensions-nralbers
```

1.1.2 `CombinedLogParser`

This processor will parse events formatted in Apache Combined log format into Elastic common schema fields.

Example

This is an example for configuring gunicorn to emit json logs.

`gunicorn.conf.py`

```
import structlog
import structlog_extensions

# --- Structlog logging initialisation code

pre_chain = [
    # Add the log level and a timestamp to the event_dict if the log entry
    # is not from structlog.
    structlog.stdlib.add_log_level,
    structlog.stdlib.add_logger_name,
    structlog_extensions.processors.CombinedLogParser("gunicorn.access")
]

logconfig_dict = {
            "version": 1,
            "disable_existing_loggers": False,
            "formatters": {
                "json_formatter": {
                    "()": structlog.stdlib.ProcessorFormatter,
                    "processor": structlog.processors.JSONRenderer(),
                    "foreign_pre_chain": pre_chain,
                }
```

(continues on next page)

(continued from previous page)

```
            },
            "handlers": {
                "error_console": {
                    "class": "logging.StreamHandler",
                    "formatter": "json_formatter",
                },
                "console": {
                    "class": "logging.StreamHandler",
                    "formatter": "json_formatter",
                }
            },
        }
```

1.1.3 `NestedDictJSONRenderer`

This processor will convert key names using a specified separator into nested dictionaries prior to rendering the output as JSON using the structlog JSONRenderer. This processor can for example convert Elastic Common Schema namespaced keynames produced by the `CombinedLogParser` into the nested JSON format that ECS specifies. This processor must be the final processor in a chain because it renders the output as a string instead of passing along an event dictionary.

Example

When using this logging initialisation:

```
# --- std logging initialisation code using structlog rendering
import structlog
import structlog_extensions

pre_chain = [
    # Add the log level and a timestamp to the event_dict if the log entry
    # is not from structlog.
    structlog.stdlib.add_log_level,
    structlog.stdlib.add_logger_name,
    structlog_extensions.processors.CombinedLogParser("gunicorn.access")
]

logging.dict_config( {
            "version": 1,
            "disable_existing_loggers": False,
            "formatters": {
                "json_formatter": {
                    "()": structlog.stdlib.ProcessorFormatter,
                    "processor": structlog_extensions.processors.
↪NestedDictJSONRenderer('.'),
                    "foreign_pre_chain": pre_chain,
                }
            },
            "handlers": {
                "error_console": {
                    "class": "logging.StreamHandler",
                    "formatter": "json_formatter",
                },
```

(continues on next page)

(continued from previous page)

```
                "console": {
                    "class": "logging.StreamHandler",
                    "formatter": "json_formatter",
                }
            },
        })
```

These entries (produced by `structlog_extensions.processors.CombinedLogParser`):

```
{ 'http.request.method': 'get', 'http:.request.referrer': 'http://www.
↪example.com', 'http.version': '1.0'}`
```

will be transformed into the following nested json structure:

```
{ 'http': { 'version': '1.0',
            'request': { 'method': 'get',
                         'referrer': 'http://www.example.com'}
            }
}
```

CHAPTER

TWO

API REFERENCE

2.1 API Reference

2.1.1 `processors` Module

structlog_extensions.processors

This module contains processors for structlog.

class structlog_extensions.processors.**CombinedLogParser**(*target_logger*)

Parses Apache combined log style entries in the event field into separate fields using Elastic Common Schema field names.

target_logger

Name of the logger object that is logging combined log output.

Type str

Example

Creating and using a parser instance with structlog:

```
import structlog_extensions
import structlog
import logging



structlog.configure(
    processors=[
        structlog_extensions.processors.CombinedLogParser('gunicorn.access'),
        structlog.stdlib.ProcessorFormatter.wrap_for_formatter,
    ],
    logger_factory=structlog.stdlib.LoggerFactory(),
)

formatter = structlog.stdlib.ProcessorFormatter(
    processor=structlog.dev.ConsoleRenderer(colors=False),
)

handler = logging.StreamHandler()
handler.setFormatter(formatter)
root_logger = logging.getLogger()
```

(continues on next page)

(continued from previous page)

```
root_logger.addHandler(handler)
root_logger.setLevel(logging.INFO)

logger = structlog.get_logger("access")
logger.warning(
    '127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0"␣
↪200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"
↪')
```

class structlog_extensions.processors.**NestedDictJSONRenderer**(**args*, *clean_keys=None*, *separator='_'*, ***kwargs*)

Structlog processor for creating JSON output including nested key output. It assumes that any key name using the specified separator is actually multiple nested keys, and will transform the log event dictionary to reflect this. The primary use case is for turning 'flat' key-value pairs into the sort of structured json seen in (for example) elastic common schema.

Notes

Must be the last processor on a chain (when using structlog as a renderer), or the processor of a `structlog.stdlib.ProcessorFormatter` object when using logging as a renderer.

Example

When using this logging initialisation:

```
# --- std logging initialisation code using structlog rendering

pre_chain = [
    # Add the log level and a timestamp to the event_dict if the log entry
    # is not from structlog.
    structlog.stdlib.add_log_level,
    structlog.stdlib.add_logger_name,
    structlog_extensions.processors.CombinedLogParser("gunicorn.access")
]

logging.dict_config( {
            "version": 1,
            "disable_existing_loggers": False,
            "formatters": {
                "json_formatter": {
                    "()": structlog.stdlib.ProcessorFormatter,
                    "processor": structlog_extensions.processors.
↪NestedDictJSONRenderer('.'),
                    "foreign_pre_chain": pre_chain,
                }
            },
            "handlers": {
                "error_console": {
                    "class": "logging.StreamHandler",
                    "formatter": "json_formatter",
                },
                "console": {
```

(continues on next page)

(continued from previous page)

```
                    "class": "logging.StreamHandler",
                    "formatter": "json_formatter",
                }
            },
        })
```

These entries (produced by `structlog_extensions.processors.CombinedLogParser`):

```
{ 'http.request.method': 'get', 'http:.request.referrer': 'http://www.example.com
↪', 'http.version': '1.0'}`
```

will be transformed into the following nested json structure:

```
{ 'http': { 'version': '1.0',
            'request': { 'method': 'get',
                         'referrer': 'http://www.example.com'}
            }
}
```

Notes

In cases where a root key has a value assigned and potential subkeys exist, the behaviour of this processor could be difficult to predict since it uses a deep merge that will perform overwriting in the order the keys are supplied by the event_dict. To prevent this happening, use the *clean_fields* attribute to specify the conflicting keys that should be removed from the event_dict prior to expansion.

separator
Namespace separator. Default = '_'

Type str, optional

clean_keys
List of keys to remove from log event prior to expansion. Intended for use when the original log event has key names that might overlap with the root names of nested keys.

Type list, optional

2.1.2 `utils` Module

`structlog_extensions.utils.`**`convert_combined_log_to_ecs`**(*log_line*, *dataset*, *severity=0*)
Converts a combined log entry into a dict containing the log entry key/values with the key names using Elastic Common schema element names.

Parameters

- **log_line** (*str*) – Combined log entry
- **dataset** (*str*) – source of the log entry (for example 'apache.access')
- **severity** (*int, optional*) – severity of the source log event

Returns Dictionary of key/value pairs with the key names using ECS namespaced names.

Return type dict

`structlog_extensions.utils.`**`unflatten_dict`**(*flat_dict*, *separator='.'*)
Turns a dict with key names defining a namespace into a nested dictionary

Parameters

- **flat_dict** (*dict*) – The dict cont
- **separator** (*str, optional*) – The separator used to split name elements. Default '.'

Returns A nested dictionary structure created from the original flat dict.

Return type dict

structlog_extensions.utils.**combined_log_timestring_to_iso**(*timestamp*)
converts a combined log format date/time entry into an iso standard datetime string :param timestamp: Datetime log entry to convert :type timestamp: str

Returns iso standard datetime string

Return type str

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX