ElasticLogger Documentation

Eduardo Aguilar

Sep 24, 2021

CONTENTS:

CHAPTER

ONE

QUICK START

1.1 Basic config

ElasticLogger use the standard python logging package and the python_json_logger package to get a standardized logger that can be compatible with elastic search and Sentry.

The way to create a simple logger is by following the next example:

```
import logging
from elasticlogger import Logger

logger = Logger("test-logger", level=logging.DEBUG)
```

This will create a logger instance with DEBUG level and you can simply log like this:

```
logger.debug("test logger message")
# {"asctime": "2021-09-23 20:35:21,261", "levelname": "DEBUG", "name": "test-logger",
↪"message": "test logger message"}
```

CHAPTER

TWO

CONTEXT DATA

Context data is used to log some common data across all log outputs to make traceability or correlate outputs of the application.

2.1 Using Logger Context

Use Logger context is similar to use the **fields** or **field** methods of the Logger class, the main difference is that context data is not cleared when the logs are prompted, it keeps alive through all the calls.

To configure some context data we need to call the next functions

```
import logging
from elasticlogger import Logger

logger = Logger("test-logger", logging.DEBUG)

# Add single context value
logger.context.field("context_key", "value")

# Add multiple context values
logger.context.fields({
    "context_key_m1": "value",
    "context_key_m2": "value"
})
```

Context values will override any current extra value with the same name of a context value, giving priority to any context data across all logs.

CHAPTER

THREE

LOGGER HOOKS

Logger hooks are functions or callable classes that receive all the logging information that should be logged by the instance in any of the calls of the debug, info, warning, error, fatal or critical methods.

Hooks can be used to modify logging message or extra data fields before being logged, stream log records to any other platform or trigger any other feature on certain log data conditions.

- Example:

```
def hook_function(context: HookContext) -> NoReturn:
    if 'foo' in context.extra_data:
        del context.extra_data['foo']

logger.add_hook(hook_function)
```

- Example 2:

```
from elasticlogger import Logger
from elasticlogger.hooks import HookContext


class HookClass:
    def __call__(self, context: HookContext) -> NoReturn:
        if 'bar' in context.extra_data:
            context.extra_data['bar'] = context.extra_data['foo']

logger = Logger('test')
logger.add_hook(HookClass())
```

3.1 Elasticsearch

3.1.1 Quick Start with Elasticsearch integration

3.1.1.1 Introduction

Elasticlogger has a native implementation of the Elasticsearch driver to stream all your cluster in a simply and effective way.

To start using this integration is as easy that you just need to call one method:

```
from elasticlogger import Logger
from elasticlogger.hooks.elasticsearch import ElasticSearch

logger = Logger('test', hooks=[ElasticSearch()])
```

This will enable the elastic search integration by searching configs on the env vars *ELASTICSEARCH_URL* and *ELASTICSEARCH_INDEX*.

3.1.1.2 Document keys

Whe you configure the elastic search integration you need to consider the keys that your objects will contain.

For local logging the keys that are prompted are compliant with the standard key words of the python logger:

- asctime
- name
- levelname
- message

For that reason this keys are treat as reserved keywords and are omitted at the display time.

When Logger stream the logs to Elasticsearch some keywords was changed to be more complaint with Elasticsearch naming. This is how the keys are converted and streamed to the cluster.

- asctime => @timestamp
- message => @message
- levelname => level

3.1.1.3 Reserved log key names

- asctime
- name
- levelname
- message
- @timestamp
- @message
- level

3.1.2 Config with AWS Elasticsearch service

To use an Elasticsearch service hosted on AWS yo need to make the following configurations:

```
import os
import logging

from elasticsearch import RequestsHttpConnection
from requests_aws4auth import AWS4Auth
```

(continues on next page)

(continued from previous page)

```
from boto3.session import Session

from elasticlogger import Logger
from elasticlogger.hooks.elasticsearch import ElasticSearch

region = 'us-east-1' # Change with your specific region
service = 'es'
credentials = Session().get_credentials()

aws_auth = AWS4Auth(
    credentials.access_key,
    credentials.secret_key,
    region,
    service,
    session_token=credentials.token
)

es_hook = ElasticSearch(
    url=os.getenv("ELASTICSEARCH_URL"),
    index=os.getenv("ELASTICSEARCH_INDEX"),
    http_auth=aws_auth,
    use_ssl=True,
    verify_certs=True,
    connection_class=RequestsHttpConnection
)

logger = Logger("test-logger", hooks=[es_hook])
```

3.1.3 Introduction

Elasticlogger has a native implementation of the Elasticsearch driver to stream all your cluster in a simply and effective way.

To start using this integration is as easy that you just need to call one method:

```
from elasticlogger import Logger
from elasticlogger.hooks.elasticsearch import ElasticSearch

logger = Logger('test', hooks=[ElasticSearch()])

# Also can be added after Logger initialization
logger.add_hook(ElasticSearch())
```

This will enable the elastic search integration by searching configs on the env vars *ELASTICSEARCH_URL* and *ELASTICSEARCH_INDEX*.

CHAPTER

FOUR

MODULE CLASSES

4.1 Logger

CHAPTER

FIVE

ELASTICLOGGER PACKAGE API

5.1 Submodules

5.2 elasticlogger.hooks.elasticsearch.elasticsearch module

5.3 elasticlogger.hooks.hook_context module

Hook context that keeps data to be modified.

class elasticlogger.hooks.hook_context.**HookContext**(*level: int*, *logger_level: int*, *logger_name: AnyStr*, *message: AnyStr*, *extra_data: Dict[AnyStr, Any]*)

Bases: object

Context information for hook execution.

Parameters

- **level** (*int*) – Current log record level
- **logger_level** (*int*) – Global logger level
- **logger_name** (*str*) – Global logger name
- **message** (*str*) – Current log record message
- **extra_data** (*dict*) – Current log record extra data

property level: int
Getter for level property.

property logger_level: int
Global logger level.

property logger_name: AnyStr
Global logger name.

5.4 elasticlogger.hooks.hooks module

Definition of a hook.

Logger hooks are functions or callable classes that receive all the logging information that should be logged by the instance in any of the calls of the debug, info, warning, error, fatal or critical methods.

Example:

```
def hook_function(context: HookContext) -> NoReturn:
    if 'foo' in context.extra_data: del context.extra_data['foo']
logger.add_hook(hook_function)
```

Example2:

```
class HookClass:
    def __call__(self, context: HookContext) -> NoReturn:
        if 'bar' in context.extra_data: context.extra_data['bar'] = context.extra_data['foo']
logger.add_hook(HookClass())
```

5.5 elasticlogger.ports.elasticsearch.elasticsearch module

5.6 elasticlogger.types.linked_list module

5.7 elasticlogger.types.context module

Logger context data store

class elasticlogger.types.context.**Context**

Bases: contextlib.ContextDecorator

Context manager for global and persistent logger data.

clear()

Delete all previous context data

property data

Return stored data fo the context.

field(*name: AnyStr*, *value: Any*)

Add single field to context data. Context data will be logged in all logs and never is auto cleaned

Parameters

- **name** – New key name for the field
- **value** – Value of the field (if it's an object needs to be json serializable)

Return Context Self instance

Raises *ContextKeyError* – When the given context key is not a str

fields(*fields: dict*)

Add fields to context data. Context data will be logged in all logs and never is auto cleaned

Parameters **fields** – (dict) Extra fields to add in the json log

Return Context Self instance

5.8 elasticlogger.types.json_encoder module

5.9 elasticlogger.utils.utils module

Utils that can be used across all files.

elasticlogger.utils.utils.**get_error_info**(*error: Any*) → Dict[AnyStr, Any]
Extract error information from a given error object.

Parameters **error** – Error information

Return Dict[AnyStr, Any] Extracted error information

elasticlogger.utils.utils.**get_level_name**(*level: int*)
Return level name from logging package levels

Parameters **level** – Logging level

Raises *InvalidLogLevel* – For an invalid level value

Return int String name

elasticlogger.utils.utils.**get_logging_method**(*level: int*, *logger: logging.Logger*) → Callable
Return logging method from an instance of a logger

Parameters

- **level** – Logging level
- **logger** – Current instance of a Logger object

Raises *InvalidLogLevel* – For an invalid level value

Return int String name

5.10 elasticlogger.base_logger module

5.11 elasticlogger.errors module

Custom module errors

exception elasticlogger.errors.**ContextKeyError**
Bases: Exception

Error for invalid type of a Context key

exception elasticlogger.errors.**InvalidLogLevel**
Bases: Exception

Error raised when an invalid log level is detected

5.12 elasticlogger.logger module

5.13 Module contents

PYTHON MODULE INDEX

e

INDEX