Spylunking Documentation

Release 1.0.0

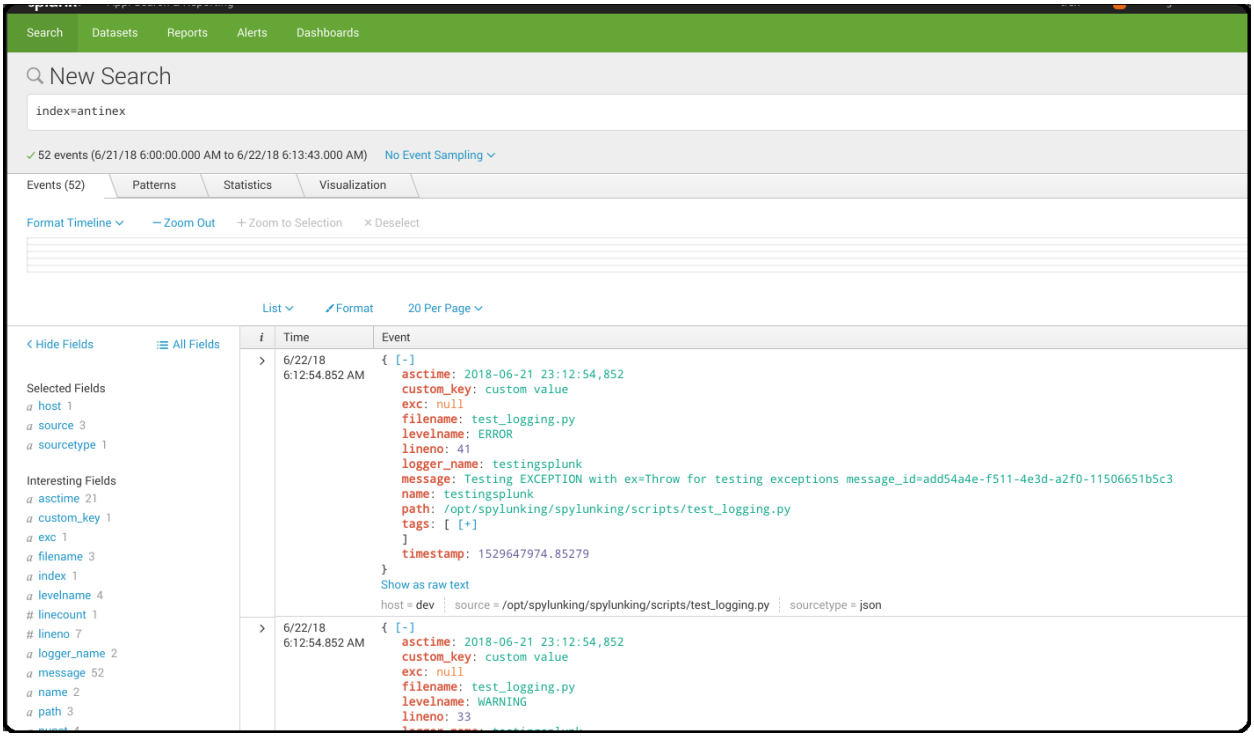
Jay Johnson

Jan 27, 2019

Contents:

Drill down into your logs with an integrated, colorized logger and search tools set up with the included Splunk docker sandbox.

This repository creates Splunk-ready, colorized Python loggers that work with a Splunk TCP Port or the Splunk HEC REST API. Both of these endpoints are automatically set up for use with the included docker container.



CHAPTER 1

Sample Log Handlers

Depending on your application's use case you can use one of the included Python logging handlers:

- TCP Splunk Publisher
- Threaded Splunk Publisher
- Multiprocessing Splunk Publisher

The log publishing and search tools support using existing Splunk tokens or logging in using the configured user and password arguments or from environment variables.

CHAPTER 2

Sample Log Config JSON Files

Here are the sample logging config JSON files:

- TCP Splunk Publisher Log Config
- Threaded Splunk Publisher Log Config
- Multiprocessing Splunk Publisher Log Config

Travis Build	Read the Docs

CHAPTER 3

Getting Started

1. Clone the repo

```
git clone https://github.com/jay-johnson/spylunking.git spylunking
cd spylunking
```

2. Install the pip

```
pip install spylunking
```

   If you want to develop use this command:

```
pip install -e .
```

3. Start the Splunk docker container

```
./run-splunk-in-docker.sh
```

CHAPTER 4

Get a Splunk User Token

By default the container creates a user with the credentials:

username: **trex** password: **123321**

```
get_splunk_token.py
955324da-742b-43d4-9746-bcbedf6ae7f4
```

Set the Splunk Environment Variables

```
export SPLUNK_INDEX=antinex
export SPLUNK_TOKEN=955324da-742b-43d4-9746-bcbedf6ae7f4
```

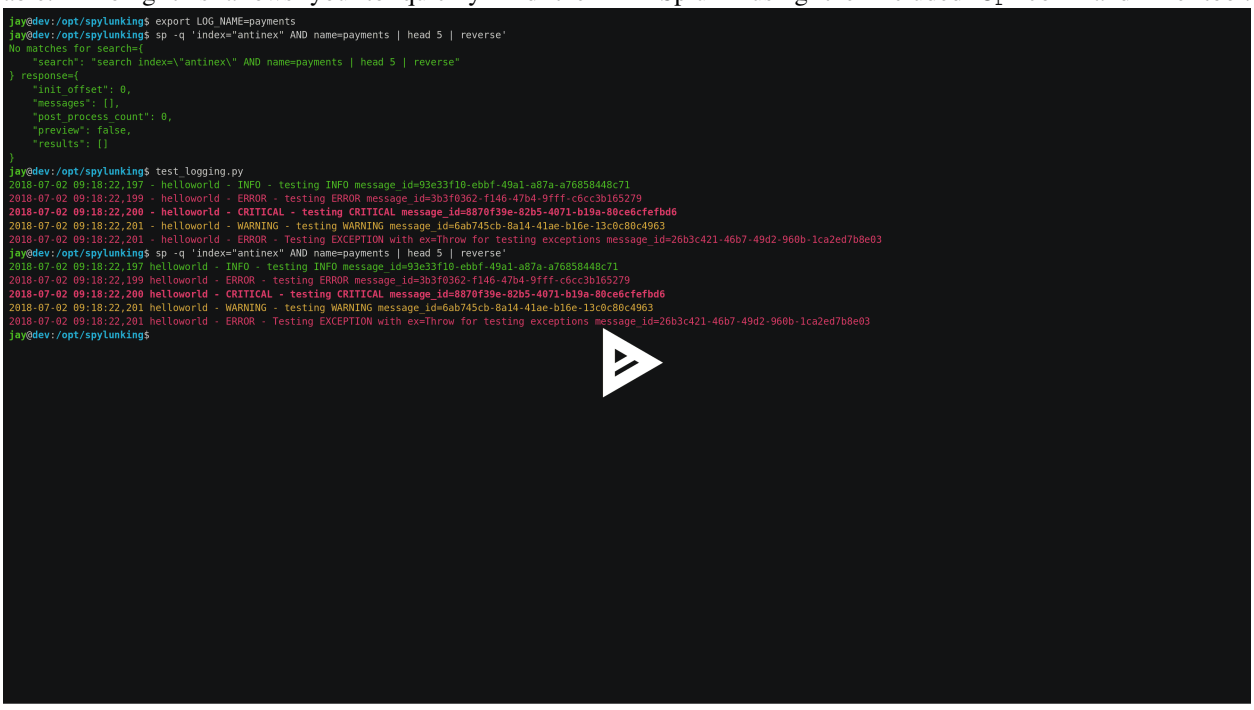
Please wait at least 30 seconds while the container is getting ready. You may see output like this when the `splunk` container is not ready yet or stops running:

```
get_splunk_token.py
Traceback (most recent call last):
File "<redacted path for doc>", line 171, in _new_conn
    (self._dns_host, self.port), self.timeout, **extra_kw)
File "<redacted path for doc>", line 79, in create_connection
    raise err
File "<redacted path for doc>", line 69, in create_connection
    sock.connect(sa)
ConnectionRefusedError: [Errno 111] Connection refused
```

CHAPTER 5

Publishing Logs to Splunk using the Spylunking Logger

Below is a video showing how to tag your application's logs using the `LOG_NAME` environment variable. Doing this allows you to quickly find them in Splunk using the included `sp` command line tool.



Commands from the video:

1. Set an Application Log Name

```
export LOG_NAME=payments
```

2. Search for Logs in Splunk

```
sp -q 'index="antinex" AND name=payments | head 5 | reverse'
No matches for search={
    "search": "search index=\"antinex\" AND name=payments | head 5 | reverse"
} response={
    "init_offset": 0,
    "messages": [],
    "post_process_count": 0,
    "preview": false,
    "results": []
}
```

3. Send Test Logs to Splunk

```
test_logging.py
2018-07-02 09:18:22,197 - helloworld - INFO - testing INFO message_id=93e33f10-
↪ebbf-49a1-a87a-a76858448c71
2018-07-02 09:18:22,199 - helloworld - ERROR - testing ERROR message_id=3b3f0362-
↪f146-47b4-9fff-c6cc3b165279
2018-07-02 09:18:22,200 - helloworld - CRITICAL - testing CRITICAL message_
↪id=8870f39e-82b5-4071-b19a-80ce6cfefbd6
2018-07-02 09:18:22,201 - helloworld - WARNING - testing WARNING message_
↪id=6ab745cb-8a14-41ae-b16e-13c0c80c4963
2018-07-02 09:18:22,201 - helloworld - ERROR - Testing EXCEPTION with ex=Throw␣
↪for testing exceptions message_id=26b3c421-46b7-49d2-960b-1ca2ed7b8e03
```

4. Search for Test Logs in Splunk

```
sp -q 'index="antinex" AND name=payments | head 5 | reverse'
2018-07-02 09:18:22,197 helloworld - INFO - testing INFO message_id=93e33f10-ebbf-
↪49a1-a87a-a76858448c71
2018-07-02 09:18:22,199 helloworld - ERROR - testing ERROR message_id=3b3f0362-
↪f146-47b4-9fff-c6cc3b165279
2018-07-02 09:18:22,200 helloworld - CRITICAL - testing CRITICAL message_
↪id=8870f39e-82b5-4071-b19a-80ce6cfefbd6
2018-07-02 09:18:22,201 helloworld - WARNING - testing WARNING message_
↪id=6ab745cb-8a14-41ae-b16e-13c0c80c4963
2018-07-02 09:18:22,201 helloworld - ERROR - Testing EXCEPTION with ex=Throw for␣
↪testing exceptions message_id=26b3c421-46b7-49d2-960b-1ca2ed7b8e03
```

CHAPTER 6

Get Splunk Logs from the Command Line Tool

Use the command line tool: **sp** to search for recent logs.

1. Set environment variables:

```
export SPLUNK_ADDRESS="splunkenterprise:8088"
export SPLUNK_API_ADDRESS="splunkenterprise:8089"
export SPLUNK_PASSWORD="123321"
export SPLUNK_USER="trex"
```

Note: The remainder of this guide was recorded by running the splunk container on a remote vm and then setting the environment variables for the search tool `sp` and the spylunking logger to work. If you are running the container locally, either add `splunkenterprise` to `/etc/hosts` at the end of the `127.0.0.1` line or export these environment variables to work with the local splunk container: `export SPLUNK_ADDRESS:localhost:8088` and `export SPLUNK_API_ADDRESS=localhost:8089`.

2. Run the tool:

```
sp
```

Which will log something like:

```
sp - INFO - No matches for search={
    "search": "search index=\"antinex\" | head 10"
}
sp - INFO - done
```

CHAPTER 7

Write Splunk Logs

By default, the container creates a Splunk index called: **antinex** with a user token for the user **trex** to search the index. Once the Splunk container is running, you can use the included **test_logging.py** script to create sample logs to verify the Splunk logging integration is working. The default logger will send logs over TCP using the TCP Splunk Publisher. To change this, you can export the optional environment variable `SHARED_LOG_CFG` to the absolute path of another logging config JSON file like:

```
export SHARED_LOG_CFG=<absolute path to logging config JSON file>
```

Send logs using the command: `test_logging.py`

```
test_logging.py
2018-06-24 01:07:36,378 - testingsplunk - INFO - testing INFO message_id=ce9c91dc-
↪3af9-484d-aeb0-fc09194bb42e
2018-06-24 01:07:36,379 - testingsplunk - ERROR - testing ERROR message_id=9227cc2f-
↪f734-4b99-8448-117776ef6bff
2018-06-24 01:07:36,379 - testingsplunk - CRITICAL - testing CRITICAL message_
↪id=7271a65d-d563-4231-b24a-b17364044818
2018-06-24 01:07:36,379 - testingsplunk - WARNING - testing WARN message_id=54063058-
↪dba1-47ee-a0ab-d654b3140e55
2018-06-24 01:07:36,379 - testingsplunk - ERROR - Testing EXCEPTION with ex=Throw for␣
↪testing exceptions message_id=c1e100f4-202d-48ac-9803-91c4f02c9a92
```

CHAPTER 8

Get the Test Splunk Logs using the Command Line Tool

The command line tool called `sp` is included with the pip on install. When you run it, it will return the most recent logs from the index (`antinex` by default) and print them to stdout.

```
sp
```

If you want to pull logs from splunk with user credentials (`SPLUNK_USER` and `SPLUNK_PASSWORD` as environment variables works too):

```
sp -u trex -p 123321 -a splunkenterprise:8089
```

Running `sp` should return something like these test logs:

```
sp -u trex -p 123321 -a splunkenterprise:8089

sp - ERROR - testingsplunk.testingsplunk 2018-06-24 01:07:36,379 - Testing EXCEPTION␣
↪with ex=Throw for testing exceptions message_id=c1e100f4-202d-48ac-9803-
↪91c4f02c9a92 dc= env= source=/opt/spylunking/spylunking/scripts/test_logging.py␣
↪line=41 ex=None
sp - CRITICAL - testingsplunk.testingsplunk 2018-06-24 01:07:36,379 - testing␣
↪CRITICAL message_id=7271a65d-d563-4231-b24a-b17364044818 dc= env= source=/opt/
↪spylunking/spylunking/scripts/test_logging.py line=31 ex=None
sp - ERROR - testingsplunk.testingsplunk 2018-06-24 01:07:36,379 - testing ERROR␣
↪message_id=9227cc2f-f734-4b99-8448-117776ef6bff dc= env= source=/opt/spylunking/
↪spylunking/scripts/test_logging.py line=29 ex=None
sp - INFO - testingsplunk.testingsplunk 2018-06-24 01:07:36,378 - testing INFO␣
↪message_id=ce9c91dc-3af9-484d-aeb0-fc09194bb42e dc= env= source=/opt/spylunking/
↪spylunking/scripts/test_logging.py line=27 ex=None
sp - INFO - done
```

CHAPTER 9

Examples

9.1 Pull Logs with a Query on the Command Line

```
sp -q 'index="antinex" AND levelname=INFO | head 10' \
    -u trex -p 123321 -a splunkenterprise:8089
sp - INFO - testingsplunk.testingsplunk 2018-06-24 01:40:18,313 - testing INFO␣
↪message_id=74b8fe93-ce07-4b8f-a700-dcf4665416d3 dc= env= source=/opt/spylunking/
↪spylunking/scripts/test_logging.py line=27 ex=None
sp - INFO - testingsplunk.testingsplunk 2018-06-24 01:25:19,162 - testing INFO␣
↪message_id=766e1408-1252-47e2-99db-e3154f5b915a dc= env= source=/opt/spylunking/
↪spylunking/scripts/test_logging.py line=27 ex=None
sp - INFO - testingsplunk.testingsplunk 2018-06-24 01:07:36,378 - testing INFO␣
↪message_id=ce9c91dc-3af9-484d-aeb0-fc09194bb42e dc= env= source=/opt/spylunking/
↪spylunking/scripts/test_logging.py line=27 ex=None
sp - INFO - done
```

9.2 Pull Logs with a Query on the Command Line

9.3 Get CRITICAL logs

```
sp -q 'index="antinex" AND levelname="CRITICAL"'
```

9.4 Get First 10 ERROR logs

```
sp -q 'index="antinex" AND levelname="ERROR" | head 10' \
    -u trex -p 123321 -a splunkenterprise:8089
```

Running `sp` also works if you want to view the full json fields:

```
sp -j -u trex -p 123321 -a splunkenterprise:8089

sp - ERROR - {
    "asctime": "2018-06-24 01:07:36,379",
    "custom_key": "custom value",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "ERROR",
    "lineno": 41,
    "logger_name": "testingsplunk",
    "message": "Testing EXCEPTION with ex=Throw for testing exceptions message_
↪id=c1e100f4-202d-48ac-9803-91c4f02c9a92",
    "name": "testingsplunk",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529827656.3798487
}
sp - CRITICAL - {
    "asctime": "2018-06-24 01:07:36,379",
    "custom_key": "custom value",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "CRITICAL",
    "lineno": 31,
    "logger_name": "testingsplunk",
    "message": "testing CRITICAL message_id=7271a65d-d563-4231-b24a-b17364044818",
    "name": "testingsplunk",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529827656.3794894
}
sp - ERROR - {
    "asctime": "2018-06-24 01:07:36,379",
    "custom_key": "custom value",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "ERROR",
    "lineno": 29,
    "logger_name": "testingsplunk",
    "message": "testing ERROR message_id=9227cc2f-f734-4b99-8448-117776ef6bff",
    "name": "testingsplunk",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529827656.3792682
}
sp - INFO - {
    "asctime": "2018-06-24 01:07:36,378",
    "custom_key": "custom value",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "INFO",
    "lineno": 27,
    "logger_name": "testingsplunk",
    "message": "testing INFO message_id=ce9c91dc-3af9-484d-aeb0-fc09194bb42e",
    "name": "testingsplunk",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
```

(continues on next page)

(continued from previous page)

```
    "timestamp": 1529827656.3789432
}
sp - INFO - done
```

CHAPTER 10

Running Stats Commands like Counting Log Matches

After running a few million logs through the Splunk container you can count the number of matches using `sp`:

```
sp -q 'index="antinex" | stats count'
{
    "count": "9261227"
}
```

CHAPTER 11

Splunk Client Load Testing

If you are looking to tune your Splunk client logging performance, then please check out the included load tester to validate the deployed configuration will not fail to publish log messages (if that is required for your client).

Before using this in production, please note it is possible to overflow the current python queues during something like an extended Splunk maintenance window or if the client is publishing logs over an unreliable network connection. The default configuration is only going to queue up to 1 million log messages before starting to drop new logs. Another way to test this is if your application is writing logs faster than the Splunk REST API can keep up, then eventually it will overflow the queue's default depth. If you are concerned about not losing log messages, then the logger should set a flush interval of `0` to disable the asynchronous, threaded queue support. This will put the client logger into a blocking mode and ensure there are no missed log messages. Please consider that this change will only create blocking log publishers where the `retry_count` and `timeout` values should be tuned to your application's needs to prevent slow application performance while waiting on the client's HTTP requests to acknowledge each log was received.

Here is how to start a single process load tester:

```
./spylunking/scripts/start_logging_loader.py
2018-06-28 22:01:47,702 - load-test-2018_06_29_05_01_47 - INFO - INFO message_
↪id=acdbfd0a-6349-4c2e-959c-f49572fc94ca
2018-06-28 22:01:47,702 - load-test-2018_06_29_05_01_47 - ERROR - ERROR message_
↪id=7daf8a8e-0d8d-4aa8-9ed1-313cd5dfb421
2018-06-28 22:01:47,702 - load-test-2018_06_29_05_01_47 - CRITICAL - CRITICAL message_
↪id=a27e7778-94be-4a35-9ce2-279403b7cf60
2018-06-28 22:01:47,703 - load-test-2018_06_29_05_01_47 - WARNING - WARN message_
↪id=d4f39765-5812-4e2e-b7ce-857b231f79d4
```

CHAPTER 12

Logging to Splunk from a Python Shell

Here are python commands to build a colorized, splunk-ready python logger. On startup, the logger will authenticate with splunk using the provided credentials. Once authenticated you can use it like a normal logger.

Note: The `build_colorized_logger` and `search` method also support authentication using a pre-existing `splunk_token=<token string>` or by setting a `SPLUNK_TOKEN` environment key

```
python -c '\
    import json;\
    from spylunking.log.setup_logging import build_colorized_logger;\
    import spylunking.search as sp;\
    from spylunking.ppj import ppj;\
    print("build the logger");\
    log = build_colorized_logger(\
        name="spylunking-in-a-shell",\
        splunk_user="trex", \
        splunk_password="123321");\
    print("import the search wrapper");\
    res = sp.search(\
        user="trex",\
        password="123321",\
        address="splunkenterprise:8089",\
        query_dict={\
            "search": "search index=\"antinex\" | head 1"\
        });\
    print("pretty print the first record in the result list");\
    log.critical("found search results={}".format(ppj(json.loads(res["record"][
↪"results"][0]["_raw"]))))'
```

Here is sample output from running this command:

```
build the logger
import the search wrapper
```

(continues on next page)

(continued from previous page)

```
pretty print the first record in the result list
2018-06-21 22:38:38,475 - spylunking-in-a-shell - CRITICAL - found search results={
    "asctime": "2018-06-21 22:13:36,279",
    "custom_key": "custom value",
    "exc": null,
    "filename": "<stdin>",
    "levelname": "INFO",
    "lineno": 1,
    "logger_name": "spylunking-in-a-shell",
    "message": "testing from a python shell",
    "name": "spylunking-in-a-shell",
    "path": "<stdin>",
    "tags": [],
    "timestamp": 1529644416.2790444
}
```

Here it is from a python shell:

```
python
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from spylunking.log.setup_logging import build_colorized_logger
>>> log = build_colorized_logger(
        name='spylunking-in-a-shell',
        splunk_user='trex',
        splunk_password='123321')
>>> import spylunking.search as sp
>>> res = sp.search(
        user='trex',
        password='123321',
        address="splunkenterprise:8089",
        query_dict={
            'search': 'search index="antinex" | head 1'
        })
>>> from spylunking.ppj import ppj
>>> log.critical('found search results={}'.format(ppj(json.loads(res['record'][
↪'results'][0]['_raw']))))
2018-06-21 22:31:04,231 - spylunking-in-a-shell - CRITICAL - found search results={
    "asctime": "2018-06-21 22:13:36,279",
    "custom_key": "custom value",
    "exc": null,
    "filename": "<stdin>",
    "levelname": "INFO",
    "lineno": 1,
    "logger_name": "spylunking-in-a-shell",
    "message": "testing from a python shell",
    "name": "spylunking-in-a-shell",
    "path": "<stdin>",
    "tags": [],
    "timestamp": 1529644416.2790444
}
```

CHAPTER 13

Publishing Logs to a Remote Splunk Server

Set up the environment variables:

```
export SPLUNK_API_ADDRESS="splunkenterprise:8089"
export SPLUNK_ADDRESS="splunkenterprise:8088"
export SPLUNK_USER="trex"
export SPLUNK_PASSWORD="123321"
```

Run the test tool to verify logs are published:

```
test_logging.py
2018-06-24 01:07:36,378 - testingsplunk - INFO - testing INFO message_id=ce9c91dc-
↪3af9-484d-aeb0-fc09194bb42e
2018-06-24 01:07:36,379 - testingsplunk - ERROR - testing ERROR message_id=9227cc2f-
↪f734-4b99-8448-117776ef6bff
2018-06-24 01:07:36,379 - testingsplunk - CRITICAL - testing CRITICAL message_
↪id=7271a65d-d563-4231-b24a-b17364044818
2018-06-24 01:07:36,379 - testingsplunk - WARNING - testing WARN message_id=54063058-
↪dba1-47ee-a0ab-d654b3140e55
2018-06-24 01:07:36,379 - testingsplunk - ERROR - Testing EXCEPTION with ex=Throw for␣
↪testing exceptions message_id=c1e100f4-202d-48ac-9803-91c4f02c9a92
```

Get the logs with `sp`

```
sp -a splunkenterprise:8089
```

Which should return the newly published logs:

```
sp - ERROR - testingsplunk.testingsplunk 2018-06-24 01:07:36,379 - Testing EXCEPTION␣
↪with ex=Throw for testing exceptions message_id=c1e100f4-202d-48ac-9803-
↪91c4f02c9a92 dc= env= source=/opt/spylunking/spylunking/scripts/test_logging.py␣
↪line=41 ex=None
sp - CRITICAL - testingsplunk.testingsplunk 2018-06-24 01:07:36,379 - testing␣
↪CRITICAL message_id=7271a65d-d563-4231-b24a-b17364044818 dc= env= source=/opt/
↪spylunking/spylunking/scripts/test_logging.py line=31 ex=None
```

(continues on next page)

(continued from previous page)

```
sp - ERROR - testingsplunk.testingsplunk 2018-06-24 01:07:36,379 - testing ERROR␣
↪message_id=9227cc2f-f734-4b99-8448-117776ef6bff dc= env= source=/opt/spylunking/
↪spylunking/scripts/test_logging.py line=29 ex=None
sp - INFO - testingsplunk.testingsplunk 2018-06-24 01:07:36,378 - testing INFO␣
↪message_id=ce9c91dc-3af9-484d-aeb0-fc09194bb42e dc= env= source=/opt/spylunking/
↪spylunking/scripts/test_logging.py line=27 ex=None
sp - INFO - done
```

CHAPTER 14

Set up a Logger

There are multiple loggers avaiable depending on the type of logger that is needed.

CHAPTER 15

Simple Logger

Build a simple, no dates colorized logger that prints just the message in colors and does not publish logs to Splunk using:

```
from spylunking.log.setup_logging import simple_logger
log = simple_logger()
log.info('simple logger example')
simple logger example
```

CHAPTER 16

No Date Colorized Logger

Build a colorized logger that preserves the parent application name and log level without a date field and does not publish logs to Splunk using:

```
from spylunking.log.setup_logging import no_date_colors_logger
log = no_date_colors_logger(name='app-name')
log.info('no date with colors logger example')
app-name - INFO - no date with colors logger example
```

CHAPTER 17

Test Logger

The test logger is for unittests and does not publish to Splunk.

```
from spylunking.log.setup_logging import test_logger
log = test_logger(name='unittest logger')
log.info('unittest log line')
2018-06-25 16:01:50,118 - using-a-colorized-logger - INFO - colorized logger example
```

CHAPTER 18

Console Logger

The console logger is the same as the `build_colorized_logger` which can be created with authenticated Splunk-ready logging using:

```
from spylunking.log.setup_logging import build_colorized_logger
log = build_colorized_logger(name='using-a-colorized-logger')
log.info('colorized logger example')
2018-06-25 16:47:54,053 - unittest logger - INFO - unittest log line
```

CHAPTER 19

Define Custom Fields for Splunk

You can export a custom JSON dictionary to send as JSON fields for helping drill down on log lines using this environment variable.

```
export LOG_FIELDS_DICT='{"name":"hello-world","dc":"k8-splunk","env":"development"}'
```

Or you can export the following environment variables if you just want a couple set in the logs:

```
export LOG_NAME=<application log name>
export DEPLOY_CONFIG=<PaaS/CaaS deployment config name>
export ENV_NAME<deployed environment name>
```

Log some new test messages to Splunk:

```
test_logging.py
2018-06-25 20:48:51,367 - testingsplunk - INFO - testing INFO message_id=0c5e2a2c-
↪9553-4c8a-8fff-8d77de2be78a
2018-06-25 20:48:51,368 - testingsplunk - ERROR - testing ERROR message_id=0dc1086d-
↪4fe4-4062-9882-e822f9256d6f
2018-06-25 20:48:51,368 - testingsplunk - CRITICAL - testing CRITICAL message_
↪id=0c0f56f2-e87f-41a0-babb-b71e2b9d5d5a
2018-06-25 20:48:51,368 - testingsplunk - WARNING - testing WARN message_id=59b099eb-
↪8c0d-40d0-9d3a-7dfa13fefc90
2018-06-25 20:48:51,368 - testingsplunk - ERROR - Testing EXCEPTION with ex=Throw for␣
↪testing exceptions message_id=70fc422d-d33b-4a9e-bb51-ed86aa0a02f9
```

Once published, you can search for these new logs using those new JSON fields with the `sp` search tool. Here is an example of searching for the logs with the application log name `hello-world`:

```
sp -q 'index="antinex" AND name=hello-world'
2018-06-25 20:48:51,368 testingsplunk - ERROR - Testing EXCEPTION with ex=Throw for␣
↪testing exceptions message_id=70fc422d-d33b-4a9e-bb51-ed86aa0a02f9
2018-06-25 20:48:51,368 testingsplunk - CRITICAL - testing CRITICAL message_
↪id=0c0f56f2-e87f-41a0-babb-b71e2b9d5d5a
2018-06-25 20:48:51,368 testingsplunk - ERROR - testing ERROR message_id=0dc1086d-
↪4fe4-4062-9882-e822f9256d6f
```

(continues on next page)

(continued from previous page)

```
2018-06-25 20:48:51,367 testingsplunk - INFO - testing INFO message_id=0c5e2a2c-9553-
↪4c8a-8fff-8d77de2be78a
done
```

And you can view log the full JSON dictionaries using the -j argument on the sp command:

```
sp -q 'index="antinex" AND name=hello-world' -j
{
    "asctime": "2018-06-25 20:48:51,368",
    "custom_key": "custom value",
    "dc": "k8-deploy",
    "env": "development",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "ERROR",
    "lineno": 41,
    "logger_name": "testingsplunk",
    "message": "Testing EXCEPTION with ex=Throw for testing exceptions message_
↪id=70fc422d-d33b-4a9e-bb51-ed86aa0a02f9",
    "name": "hello-world",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529984931.3688767
}
{
    "asctime": "2018-06-25 20:48:51,368",
    "custom_key": "custom value",
    "dc": "k8-deploy",
    "env": "development",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "CRITICAL",
    "lineno": 31,
    "logger_name": "testingsplunk",
    "message": "testing CRITICAL message_id=0c0f56f2-e87f-41a0-babb-b71e2b9d5d5a",
    "name": "hello-world",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529984931.3684626
}
{
    "asctime": "2018-06-25 20:48:51,368",
    "custom_key": "custom value",
    "dc": "k8-deploy",
    "env": "development",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "ERROR",
    "lineno": 29,
    "logger_name": "testingsplunk",
    "message": "testing ERROR message_id=0dc1086d-4fe4-4062-9882-e822f9256d6f",
    "name": "hello-world",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529984931.3682773
}
{
```

(continues on next page)

(continued from previous page)

```
    "asctime": "2018-06-25 20:48:51,367",
    "custom_key": "custom value",
    "dc": "k8-deploy",
    "env": "development",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "INFO",
    "lineno": 27,
    "logger_name": "testingsplunk",
    "message": "testing INFO message_id=0c5e2a2c-9553-4c8a-8fff-8d77de2be78a",
    "name": "hello-world",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529984931.3679354
}
done
```

CHAPTER 20

Available Environment Variables

20.1 Drill down fields

Splunk drill down fields with environment variables:

```
export LOG_NAME="<application log name>"
export DEPLOY_CONFIG="<application deployed config like k8 filename>"
export ENV_NAME="<environment name for this application>"
```

20.2 Common Environment Variables

```
export SPLUNK_USER="<splunk host>"
export SPLUNK_PASSWORD="<splunk host>"
export SPLUNK_HOST="<splunk host>"
export SPLUNK_PORT="<splunk port: 8088>"
export SPLUNK_API_PORT="<splunk port: 8089>"
export SPLUNK_ADDRESS="<splunk address host:port>"
export SPLUNK_API_ADDRESS="<splunk api address host:port>"
export SPLUNK_TOKEN="<splunk token>"
export SPLUNK_INDEX="<splunk index>"
export SPLUNK_SOURCE="<splunk source>"
export SPLUNK_SOURCETYPE="<splunk sourcetype>"
export SPLUNK_VERIFY="<verify certs on HTTP POST>"
export SPLUNK_TIMEOUT="<timeout in seconds>"
export SPLUNK_QUEUE_SIZE="<num msgs allowed in queue - 0=infinite>"
export SPLUNK_SLEEP_INTERVAL="<sleep in seconds per batch>"
export SPLUNK_RETRY_COUNT="<attempts per log to retry publishing>"
export SPLUNK_RETRY_BACKOFF="<cooldown in seconds per failed POST>"
export SPLUNK_DEBUG="<debug the publisher - 1 enable debug|0 off>"
export SPLUNK_VERBOSE="<debug the sp command line tool - 1 enable|0 off>"
```

20.3 Debug the Publishers

Export this variable before creating a logger to see the publisher logs:

```
export SPLUNK_DEBUG=1
```

CHAPTER 21

Login to Splunk from a Browser

Open this url in a browser to view the **splunk** container's web application:

http://127.0.0.1:8000

Login with the credentials:

username: **trex** password: **123321**

CHAPTER 22

Troubleshooting

22.1 Splunk Handler Dropping Logs

If the splunk handler is dropping log messages you can use these values to tune the handler's worker thread:

```
export SPLUNK_RETRY_COUNT="<number of attempts to send logs>"
export SPLUNK_TIMEOUT="<timeout in seconds per attempt>"
export SPLUNK_QUEUE_SIZE="<integer value or 0 for infinite>"
export SPLUNK_SLEEP_INTERVAL="<seconds to sleep between publishes>"
export SPLUNK_DEBUG="<debug the Splunk Publisher by setting to 1>"
```

22.2 Testing in a Python Shell

Here is a debugging python shell session for showing some common errors you can expect to see as you start to play around with `spylunking`.

```
python
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from spylunking.log.setup_logging import build_colorized_logger
>>> log = build_colorized_logger(
        name='spylunking-in-a-shell',
        splunk_user='trex',
        splunk_password='123321')
>>> log.info("testing from a python shell")
2018-06-21 22:13:36,279 - spylunking-in-a-shell - INFO - testing from a python shell
>>> import spylunking.search as sp
>>> res = sp.search(
        user='trex',
        password='123321',
```

(continues on next page)

(continued from previous page)

```
        query_dict={
                'search': 'index="antinex" | head 1'
        },
        verify=False)
>>> log.info('job status={}'.format(res['status']))
2018-06-21 22:16:22,158 - spylunking-in-a-shell - INFO - job status=2
>>> log.info('job err={}'.format(res['err']))
2018-06-21 22:16:28,945 - spylunking-in-a-shell - INFO - job err=Failed to get splunk␣
↪token for user=trex url=https://None ex=HTTPSConnectionPool(host='none', port=443):␣
↪Max retries exceeded with url: /services/auth/login (Caused by NewConnectionError('
↪<urllib3.connection.VerifiedHTTPSConnection object at 0x7f869c2f2cc0>: Failed to␣
↪establish a new connection: [Errno -2] Name or service not known',))
>>> print("now search with the url set")
now search with the url set
>>> res = sp.search(
        user='trex',
        password='123321',
        query_dict={
                'search': 'index="antinex" | head 1'
        },
        address="splunkenterprise:8089")
2018-06-21 22:18:15,380 - spylunking.search - ERROR - Failed searching splunk␣
↪response=<?xml version="1.0" encoding="UTF-8"?>
<response>
<messages>
    <msg type="ERROR">Search Factory: Unknown search command 'index'.</msg>
</messages>
</response>
for query={
    "search": "index=\"antinex\" | head 1"
} url=https://splunkenterprise:8089/services/search/jobs ex=list index out of range
>>> print("now nest the search correctly")
now nest the search correctly
>>> res = sp.search(
        user='trex',
        password='123321',
        address="splunkenterprise:8089",
        query_dict={
                'search': 'search index="antinex" | head 1'
        })
>>> log.info('job status={}'.format(res['status']))
2018-06-21 22:20:10,142 - spylunking-in-a-shell - INFO - job status=0
>>> log.info('job err={}'.format(res['err']))
2018-06-21 22:20:14,667 - spylunking-in-a-shell - INFO - job err=
>>> from spylunking.ppj import ppj
>>> log.critical('found search results={}'.format(ppj(res['record'])))
2018-06-21 22:21:25,977 - spylunking-in-a-shell - CRITICAL - found search results={
    "fields": [
        {
            "name": "_bkt"
        },
        {
            "name": "_cd"
        },
        {
            "name": "_indextime"
        },
```

(continues on next page)

(continued from previous page)


```
        {
            "name": "_raw"
        },
        {
            "name": "_serial"
        },
        {
            "name": "_si"
        },
        {
            "name": "_sourcetype"
        },
        {
            "name": "_subsecond"
        },
        {
            "name": "_time"
        },
        {
            "name": "host"
        },
        {
            "name": "index"
        },
        {
            "name": "linecount"
        },
        {
            "name": "source"
        },
        {
            "name": "sourcetype"
        },
        {
            "name": "splunk_server"
        }
    ],
    "highlighted": {},
    "init_offset": 0,
    "messages": [],
    "preview": false,
    "results": [
        {
            "_bkt": "antinex~0~791398E7-6A0B-4640-B8D5-5D25E7EF3D02",
            "_cd": "0:3",
            "_indextime": "1529644419",
            "_raw": "{\"asctime\": \"2018-06-21 22:13:36,279\", \"name\": \
↪"spylunking-in-a-shell\", \"levelname\": \"INFO\", \"message\": \"testing from a␣
↪python shell\", \"filename\": \"<stdin>\", \"lineno\": 1, \"timestamp\": 1529644416.
↪2790444, \"path\": \"<stdin>\", \"custom_key\": \"custom value\", \"tags\": [], \
↪"exc\": null, \"logger_name\": \"spylunking-in-a-shell\"}",
            "_serial": "0",
            "_si": [
                "splunkenterprise",
                "antinex"
            ],
            "_sourcetype": "json",
```


(continues on next page)

(continued from previous page)

```
            "_subsecond": ".2792356",
            "_time": "2018-06-22T05:13:36.279+00:00",
            "host": "dev",
            "index": "antinex",
            "linecount": "1",
            "source": "<stdin>",
            "sourcetype": "json",
            "splunk_server": "splunkenterprise"
        }
    ]
}
>>> exit()
```

Please refer to the command line tool's updated usage prompt for help searching for logs:

```
usage: sp [-h] [-u USER] [-p PASSWORD] [-f DATAFILE] [-i INDEX_NAME]
      [-a ADDRESS] [-e EARLIEST_TIME_MINUTES] [-l LATEST_TIME_MINUTES]
      [-q [QUERY_ARGS [QUERY_ARGS ...]]] [-j] [-m] [-v] [-b]

Search Splunk

optional arguments:
-h, --help            show this help message and exit
-u USER               username
-p PASSWORD           user password
-f DATAFILE           splunk-ready request in a json file
-i INDEX_NAME         index to search
-a ADDRESS            host address: <fqdn:port>
-e EARLIEST_TIME_MINUTES
                        (Optional) earliest_time minutes back
-l LATEST_TIME_MINUTES
                        (Optional) latest_time minutes back
-q [QUERY_ARGS [QUERY_ARGS ...]], --queryargs [QUERY_ARGS [QUERY_ARGS ...]]
                        query string for searching splunk: search
                        index="antinex" AND levelname="ERROR"
-j                    (Optional) view as json dictionary logs
-m                    (Optional) verbose message when getting logs
-v                    (Optional) verify certs - disabled by default
-b                    verbose
```

For trying the host-only compose file, you may see errors like:

`unable to resolve host splunkenterprise`

Please add `splunkenterprise` to the end of the line for `127.0.0.1` in your `/etc/hosts`

CHAPTER 23

Cleanup

Remove the docker container with the commands:

```
docker stop splunk
docker rm splunk
```

23.1 Manual Splunk Commands

Create Token

```
curl -k -u admin:changeme https://splunkenterprise:8089/servicesNS/admin/splunk_
↪httpinput/data/inputs/http -d name=antinex-token
```

List Token

```
curl -k -u admin:changeme https://splunkenterprise:8089/servicesNS/admin/splunk_
↪httpinput/data/inputs/http
```

23.2 Using Splunk CLI

List Tokens

```
./bin/splunk http-event-collector list -uri 'https://splunkenterprise:8089' -auth
↪'admin:changeme'
```

Add Index

```
./bin/splunk add index antinex -auth 'admin:changeme'
```

Create Token

```
./bin/splunk \
    http-event-collector create  \
    antinex-token 'antinex logging token'  \
    -index antinex \
    -uri 'https://splunkenterprise:8089' \
    -auth 'admin:changeme'
```

CHAPTER 24

Development

Setting up your development environment (right now this demo is using virtualenv):

```
virtualenv -p python3 ~/.venvs/spylunk && source ~/.venvs/spylunk/bin/activate && pip␣
↪install -e .
```

CHAPTER 25

Testing

Run all

```
py.test
```

CHAPTER 26

Linting

flake8 .

pycodestyle .

CHAPTER 27

License

Apache 2.0 - Please refer to the LICENSE for more details

27.1 Examples and Scripts

27.1.1 Environment Variables

Please use these environment variables to publish logs and run searches with a local or remote splunk server:

```
export SPLUNK_ADDRESS="splunkenterprise:8088"
export SPLUNK_API_ADDRESS="splunkenterprise:8089"
export SPLUNK_PASSWORD="123321"
export SPLUNK_USER="trex"
export SPLUNK_TOKEN="<Optional pre-existing Splunk token>"
```

27.1.2 Search Splunk with a Dictionary

The command line client `sp` is actually a copy of the `search_splunk.py` script. Note, this will likely change in the future, but for now this makes the docs easy to host on RTD.

A tool for searching splunk with python - spylunking

Examples

Please use these environment variables to publish logs and run searches with a local or remote splunk server:

```
export SPLUNK_ADDRESS="splunkenterprise:8088"
export SPLUNK_API_ADDRESS="splunkenterprise:8089"
export SPLUNK_PASSWORD="123321"
export SPLUNK_USER="trex"
```

(continues on next page)

(continued from previous page)

```
export SPLUNK_TOKEN="<Optional pre-existing Splunk token>"
export SPLUNK_INDEX="<splunk index>"
```

Pull Logs with a Query on the Command Line

```
sp -q 'index="antinex" AND levelname=INFO | head 10 | reverse'          -u trex -p␣
↪123321 -a splunkenterprise:8089
```

Pull Logs with a Query on the Command Line

Get CRITICAL logs

```
sp -q 'index="antinex" AND levelname="CRITICAL" | reverse'
```

Get First 10 ERROR logs

```
sp -q 'index="antinex" AND levelname="ERROR" | head 10 | reverse'          -u trex -p␣
↪123321 -a splunkenterprise:8089
```

spylunking.scripts.search_splunk.**run_main**()
Search Splunk

27.1.3 Publish Logs to Splunk

Publish functional testing logs to splunk using the logger

spylunking.scripts.test_logging.**run_main**()

27.1.4 Load Test Splunk

Splunk client load tester for determining how many messages can this client send over splunk. By default, this tester sends a batch of 1000 messages and then sleeps to let the client catch up.

Splunk client load tester for determining how many messages can this client send over splunk. By default, this tester sends a batch of messages and then sleeps to let the client catch up.

spylunking.scripts.start_logging_loader.**run_main**()

27.1.5 Get a Splunk User Token

Get a Splunk User Token

spylunking.scripts.get_splunk_token.**run_main**()

27.1.6 Get Splunk Service Token (Session Key)

A tool for getting splunk service tokens

spylunking.scripts.show_service_token.**run_main**()
Get Splunk Service Token

27.1.7 Splunk Functional Test Publisher

spylunking.scripts.test_publish_to_splunk.**format_record**(*msg*, *token=None*)

Parameters **token** – existing splunk token

spylunking.scripts.test_publish_to_splunk.**run_main**(*token=None*, *address=None*)
Publish logs to Splunk over a TCP data input with the sourcetype set to _json

Parameters

- **token** – splunk token to use
- **address** – splunk TCP endpoint address <fqdn:port>

27.2 Spylunking API Reference

27.3 Using TCP to Publish to Splunk

Here is the code for the Splunk Publisher that uses a TCP logging.SocketHandler to send logs to the configured Splunk server.

This class was built for writing JSON log messages to Splunk over a TCP Port.

Available environment variables:

```
export SPLUNK_TCP_ADDRESS="<splunk port: 1514>"
export SPLUNK_INDEX="<splunk index>"
export SPLUNK_SOURCE="<splunk source>"
export SPLUNK_SOURCETYPE="<splunk sourcetype>"
export SPLUNK_DEBUG="<1 enable debug|0 off>"
export SPLUNK_LOG_TOKEN="<splunk log token>"
```

class spylunking.tcp_splunk_publisher.**TCPSplunkPublisher**(*address=None*, *index=None*, *hostname=None*, *source=None*, *sourcetype='json'*, *name=None*, *dc=None*, *env=None*, *custom_dict=None*, *debug=False*, ***kwargs*)
A logging handler to send logs to a Splunk Enterprise instance with a Splunk TCP input set up for json with:

The TCP Splunk Publisher requires having a JSON-ready entry in the /opt/splunk/etc/system/default/props.conf config file.

Tokens are not required for this to work, but can be included for authenticated TCP logging.

If you want to send include a token in on the tcp log body, then you can export:

```
export SPLUNK_LOG_TOKEN=<Optional Log Token>
```

Here are the additional lines added to the splunk props.conf for validation:

```
[usejson]
SHOULD_LINEMERGE = false
KV_MODE = json
TIME_FORMAT = %Y-%m-%dT%H:%M:%S.%6N%:z
```

build_into_splunk_tcp_message(*body*)
Format a message for a Splunk JSON sourcetype

Parameters **body** – splunk JSON dictionary msg body

debug_log(*log_message*)
Write logs that only show up in debug mode. To turn on debugging with environment variables please set this environment variable:

```
export SPLUNK_DEBUG="1"
```

Parameters **log_message** – message to log

makePickle(*record*)
Convert a log record into a JSON dictionary packaged into a socket-ready message for a Splunk TCP Port configured that is set up with a JSON sourcetype

Parameters **record** – log record to format as a socket-ready message

set_fields(*custom_dict*)
Set the formatter's fields as needed even after the logger has been created.

Parameters **custom_dict** – new fields to patch in

write_log(*log_message*)
Write logs to stdout

Parameters **log_message** – message to log

27.4 Using Threads to Publish to Splunk

Here is the code for the Splunk Publisher that uses a thread to send logs to the configured Splunk server.

Note: Each created logger will spawn a separate thread with its own queue for log messages. When the parent process exits, please consider that the queue may hold log messages which can be lost due to the parent process exiting before they get a chance to be pushed to Splunk.

Including a handler derived from the original repository: https://github.com/zach-taylor/splunk_handler

This version was built to fix issues seen with multiple Celery worker processes.

Available environment variables:

```
export SPLUNK_HOST="<splunk host>"
export SPLUNK_PORT="<splunk port: 8088>"
export SPLUNK_API_PORT="<splunk port: 8089>"
```

(continues on next page)

(continued from previous page)

```
export SPLUNK_ADDRESS="<splunk address host:port>"
export SPLUNK_API_ADDRESS="<splunk api address host:port>"
export SPLUNK_TOKEN="<splunk token>"
export SPLUNK_INDEX="<splunk index>"
export SPLUNK_SOURCE="<splunk source>"
export SPLUNK_SOURCETYPE="<splunk sourcetype>"
export SPLUNK_VERIFY="<verify certs on HTTP POST>"
export SPLUNK_TIMEOUT="<timeout in seconds>"
export SPLUNK_QUEUE_SIZE="<num msgs allowed in queue - 0=infinite>"
export SPLUNK_SLEEP_INTERVAL="<sleep in seconds per batch>"
export SPLUNK_RETRY_COUNT="<attempts per log to retry publishing>"
export SPLUNK_RETRY_BACKOFF="<cooldown in seconds per failed POST>"
export SPLUNK_DEBUG="<1 enable debug|0 off>"
```

class spylunking.splunk_publisher.**SplunkPublisher**(*host=None*, *port=None*, *address=None*, *token=None*, *index=None*, *hostname=None*, *source=None*, *sourcetype='text'*, *verify=True*, *timeout=60*, *sleep_interval=2.0*, *queue_size=0*, *debug=False*, *retry_count=20*, *retry_backoff=2.0*, *run_once=False*)

A logging handler to send logs to a Splunk Enterprise instance running the Splunk HTTP Event Collector. Originally inspired from the repository: https://github.com/zach-taylor/splunk_handler This class allows multiple processes like Celery workers to reliably publish logs to Splunk from inside of a Celery task

build_payload_from_queued_messages(*use_queue*, *shutdown_event*, *triggered_by_shutdown=False*)

Empty the queued messages by building a large `self.log_payload`

Parameters

- **use_queue** – queue holding the messages
- **shutdown_event** – shutdown event
- **triggered_by_shutdown** – called during shutdown

close()

debug_log(*log_message*)

Write logs that only show up in debug mode. To turn on debugging with environment variables please set this environment variable:

```
export SPLUNK_DEBUG="1"
```

Parameters **log_message** – message to log

emit(*record*)

Emit handler for queue-ing message for the helper thread to send to Splunk on the `sleep_interval`

Parameters **record** – LogRecord to send to Splunk https://docs.python.org/3/library/logging.html

force_flush()

Flush the queue and publish everything to Splunk

format_record(*record*)
Convert a log record into a Splunk-ready format

Parameters **record** – message to format

is_shutting_down(*shutdown_event*)
Determine if the parent is shutting down or this was triggered to shutdown

Parameters **shutdown_event** – shutdown event

perform_work()
Process handler function for processing messages found in the `multiprocessing.Manager.queue`

Build the `self.log_payload` from the queued log messages and POST it to the Splunk endpoint

publish_to_splunk(*payload=None*)
Build the `self.log_payload` from the queued log messages and POST it to the Splunk endpoint

Parameters **payload** – string message to send to Splunk

queue_empty(*use_queue*)

Parameters **use_queue** – queue to test

shutdown()

start_worker_thread(*sleep_interval=1.0*)
Start the helper worker thread to publish queued messages to Splunk

Parameters **sleep_interval** – sleep in seconds before reading from the queue again

write_log(*log_message*)
Write logs to stdout

Parameters **log_message** – message to log

27.5 Using Multiprocesing to Publish to Splunk

Here is the code for the Splunk Publisher that uses a multiprocessing to send logs to the configured Splunk server.

Note: Each created logger will fork a separate process with its own queue for log messages. When the parent process exits, please consider that the queue may hold log messages which can take some time to finish publishing before the process exits.

Publish to Splunk using a `multiprocessing.Process` worker

Including a handler derived from the original repository: https://github.com/zach-taylor/splunk_handler

Please note this will not work with Celery. Please use the `spylunking.splunk_publisher.SplunkPublisher` if you want to see logs inside a Celery task.

Supported environment variables:

```
export SPLUNK_HOST="<splunk host>"
export SPLUNK_PORT="<splunk port: 8088>"
export SPLUNK_API_PORT="<splunk port: 8089>"
export SPLUNK_ADDRESS="<splunk address host:port>"
export SPLUNK_API_ADDRESS="<splunk api address host:port>"
export SPLUNK_TOKEN="<splunk token>"
export SPLUNK_INDEX="<splunk index>"
```

(continues on next page)

(continued from previous page)

```
export SPLUNK_SOURCE="<splunk source>"
export SPLUNK_SOURCETYPE="<splunk sourcetype>"
export SPLUNK_VERIFY="<verify certs on HTTP POST>"
export SPLUNK_TIMEOUT="<timeout in seconds>"
export SPLUNK_QUEUE_SIZE="<num msgs allowed in queue - 0=infinite>"
export SPLUNK_SLEEP_INTERVAL="<sleep in seconds per batch>"
export SPLUNK_RETRY_COUNT="<attempts per log to retry publishing>"
export SPLUNK_RETRY_BACKOFF="<cooldown in seconds per failed POST>"
export SPLUNK_DEBUG="<1 enable debug|0 off>"
```

class spylunking.mp_splunk_publisher.**MPSplunkPublisher**(*host=None*, *port=None*, *address=None*, *token=None*, *index=None*, *hostname=None*, *source=None*, *sourcetype='text'*, *verify=True*, *timeout=60*, *sleep_interval=None*, *queue_size=0*, *debug=False*, *retry_count=20*, *run_once=False*, *retry_backoff=2.0*)

A logging handler to send logs to a Splunk Enterprise instance running the Splunk HTTP Event Collector.

Originally inspired from the repository but written for python's multiprocessing framework: https://github.com/zach-taylor/splunk_handler

This class allows multiple processes like Celery workers to reliably publish logs to Splunk from inside of a Celery task

build_payload_from_queued_messages(*use_queue*, *shutdown_event*)

Empty the queued messages by building a large `self.log_payload`

Parameters

- **use_queue** – queue holding the messages
- **shutdown_event** – shutdown event

close()

debug_log(*log_message*)

Write logs that only show up in debug mode. To turn on debugging with environment variables please set this environment variable:

```
export SPLUNK_DEBUG="1"
```

Parameters **log_message** – message to log

emit(*record*)

Emit handler for queue-ing message for the helper thread to send to Splunk on the `self.sleep_interval`

Parameters **record** – LogRecord to send to Splunk https://docs.python.org/3/library/logging.html

force_flush()

Flush the queue and publish everything to Splunk

format_record(*record*)
Convert a log record into a Splunk-ready format

Parameters **record** – message to format

is_shutting_down(*shutdown_event*)
Determine if the parent is shutting down or this was triggered to shutdown

Parameters **shutdown_event** – shutdown event

perform_work(*use_queue*, *shutdown_event*, *shutdown_ack_event*, *already_done_event*)
Process handler function for processing messages found in the `multiprocessing.Manager.queue`

Build the `self.log_payload` from the queued log messages and POST it to the Splunk endpoint

Parameters

- **use_queue** – multiprocessing.Queue - queue holding the messages
- **shutdown_event** – multiprocessing.Event - shutdown event
- **shutdown_ack_event** – multiprocessing.Event - acknowledge shutdown is in progress
- **already_done_event** – multiprocessing.Event - already shutting down

publish_to_splunk(*payload=None*, *shutdown_event=None*, *shutdown_ack_event=None*, *already_done_event=None*)
Publish the queued messages to Splunk

Parameters

- **payload** – optional string log message to send to Splunk
- **shutdown_event** – multiprocessing.Event - shutdown event
- **shutdown_ack_event** – multiprocessing.Event - acknowledge shutdown is in progress
- **already_done_event** – multiprocessing.Event - already shutting down

queue_empty(*use_queue*)

Parameters **use_queue** – queue to test

shutdown()

start_worker()
Start the helper worker process to package queued messages and send them to Splunk

write_log(*log_message*)

Parameters **log_message** – message to log

27.5.1 Get a Splunk Service Session Key

Get a Splunk User Session Key - for running searches

spylunking.get_session_key.**get_session_key**(*user*, *password*, *url='https://localhost:8089'*, *verify=False*, *ssl_options=None*)
This will get a user session key and throw for any errors

Parameters

- **user** – username

- **password** – password
- **url** – splunk auth url
- **verify** – verify cert
- **ssl_options** – ssl options dictionary

27.5.2 Get a Splunk User Token

Get a Splunk User Token

spylunking.get_token.**get_token**(*user=None*, *password=None*, *url='https://localhost:8089'*, *verify=False*, *ssl_options=None*, *version='7.0.3'*, *debug=False*)

This will get a user token and throw for any errors

Parameters

- **user** – username - defaults to env var: SPLUNK_ADMIN_USER
- **password** – password - defaults to env var: SPLUNK_ADMIN_PASSWORD
- **url** – splunk auth url
- **verify** – verify cert
- **ssl_options** – ssl options dictionary
- **version** – splunk version string
- **debug** – debug xml response from splunk (for versioning)

27.5.3 Search Splunk

Search wrapper for authenticating with Splunk and running a search query.

spylunking.search.**search**(*user=None*, *password=None*, *token=None*, *address=None*, *query_dict=None*, *verify=False*, *debug=False*)

Search Splunk with a pre-built query dictionary and wait until it finishes.

Parameters

- **user** – splunk username
- **password** – splunk password
- **token** – splunk token
- **address** – splunk HEC address: localhost:8089
- **query_dict** – query dictionary to search
- **verify** – ssl verify
- **debug** – debug flag

27.6 Set up a Logger

There are multiple loggers avaiable depending on the type of logger that is needed.

27.7 Simple Logger

Build a simple, no dates colorized logger that prints just the message in colors and does not publish logs to Splunk using:

```
from spylunking.log.setup_logging import simple_logger
log = simple_logger()
log.info('simple logger example')
simple logger example
```

27.8 No Date Colorized Logger

Build a colorized logger that preserves the parent application name and log level without a date field and does not publish logs to Splunk using:

```
from spylunking.log.setup_logging import no_date_colors_logger
log = no_date_colors_logger(name='app-name')
log.info('no date with colors logger example')
app-name - INFO - no date with colors logger example
```

27.9 Test Logger

The test logger is for unittests and does not publish to Splunk.

```
from spylunking.log.setup_logging import test_logger
log = test_logger(name='unittest logger')
log.info('unittest log line')
2018-06-25 16:01:50,118 - using-a-colorized-logger - INFO - colorized logger example
```

27.10 Console Logger

The console logger is the same as the `build_colorized_logger` which can be created with authenticated Splunk-ready logging using:

```
from spylunking.log.setup_logging import build_colorized_logger
log = build_colorized_logger(name='using-a-colorized-logger')
log.info('colorized logger example')
2018-06-25 16:47:54,053 - unittest logger - INFO - unittest log line
```

27.11 Define Custom Fields for Splunk

You can export a custom JSON dictionary to send as JSON fields for helping drill down on log lines using this environment variable.

```
export LOG_FIELDS_DICT='{"name":"hello-world","dc":"k8-splunk","env":"development"}'
```

Or you can export the following environment variables if you just want a couple set in the logs:

```
export LOG_NAME=<application log name>
export DEPLOY_CONFIG=<PaaS/CaaS deployment config name>
export ENV_NAME<deployed environment name>
```

Log some new test messages to Splunk:

```
test_logging.py
2018-06-25 20:48:51,367 - testingsplunk - INFO - testing INFO message_id=0c5e2a2c-
↪9553-4c8a-8fff-8d77de2be78a
2018-06-25 20:48:51,368 - testingsplunk - ERROR - testing ERROR message_id=0dc1086d-
↪4fe4-4062-9882-e822f9256d6f
2018-06-25 20:48:51,368 - testingsplunk - CRITICAL - testing CRITICAL message_
↪id=0c0f56f2-e87f-41a0-babb-b71e2b9d5d5a
2018-06-25 20:48:51,368 - testingsplunk - WARNING - testing WARN message_id=59b099eb-
↪8c0d-40d0-9d3a-7dfa13fefc90
2018-06-25 20:48:51,368 - testingsplunk - ERROR - Testing EXCEPTION with ex=Throw for␣
↪testing exceptions message_id=70fc422d-d33b-4a9e-bb51-ed86aa0a02f9
```

Once published, you can search for these new logs using those new JSON fields with the `sp` search tool. Here is an example of searching for the logs with the application log name `hello-world`:

```
sp -q 'index="antinex" AND name=hello-world'
creating client user=trex address=splunk:8089
connecting trex@splunk:8089
2018-06-25 20:48:51,368 testingsplunk - ERROR - Testing EXCEPTION with ex=Throw for␣
↪testing exceptions message_id=70fc422d-d33b-4a9e-bb51-ed86aa0a02f9
2018-06-25 20:48:51,368 testingsplunk - CRITICAL - testing CRITICAL message_
↪id=0c0f56f2-e87f-41a0-babb-b71e2b9d5d5a
2018-06-25 20:48:51,368 testingsplunk - ERROR - testing ERROR message_id=0dc1086d-
↪4fe4-4062-9882-e822f9256d6f
2018-06-25 20:48:51,367 testingsplunk - INFO - testing INFO message_id=0c5e2a2c-9553-
↪4c8a-8fff-8d77de2be78a
done
```

And you can view log the full JSON dictionaries using the `-j` argument on the `sp` command:

```
sp -q 'index="antinex" AND name=hello-world' -j
creating client user=trex address=splunk:8089
connecting trex@splunk:8089
{
    "asctime": "2018-06-25 20:48:51,368",
    "custom_key": "custom value",
    "dc": "k8-deploy",
    "env": "development",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "ERROR",
    "lineno": 41,
    "logger_name": "testingsplunk",
    "message": "Testing EXCEPTION with ex=Throw for testing exceptions message_
↪id=70fc422d-d33b-4a9e-bb51-ed86aa0a02f9",
    "name": "hello-world",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529984931.3688767
}
{
```

(continues on next page)

(continued from previous page)

```
    "asctime": "2018-06-25 20:48:51,368",
    "custom_key": "custom value",
    "dc": "k8-deploy",
    "env": "development",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "CRITICAL",
    "lineno": 31,
    "logger_name": "testingsplunk",
    "message": "testing CRITICAL message_id=0c0f56f2-e87f-41a0-babb-b71e2b9d5d5a",
    "name": "hello-world",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529984931.3684626
}
{
    "asctime": "2018-06-25 20:48:51,368",
    "custom_key": "custom value",
    "dc": "k8-deploy",
    "env": "development",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "ERROR",
    "lineno": 29,
    "logger_name": "testingsplunk",
    "message": "testing ERROR message_id=0dc1086d-4fe4-4062-9882-e822f9256d6f",
    "name": "hello-world",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529984931.3682773
}
{
    "asctime": "2018-06-25 20:48:51,367",
    "custom_key": "custom value",
    "dc": "k8-deploy",
    "env": "development",
    "exc": null,
    "filename": "test_logging.py",
    "levelname": "INFO",
    "lineno": 27,
    "logger_name": "testingsplunk",
    "message": "testing INFO message_id=0c5e2a2c-9553-4c8a-8fff-8d77de2be78a",
    "name": "hello-world",
    "path": "/opt/spylunking/spylunking/scripts/test_logging.py",
    "tags": [],
    "timestamp": 1529984931.3679354
}
done
```

27.12 Debug the Logger

Export this variable before creating a logger.

```
export SPLUNK_DEBUG=1
```

27.13 Full Console Logger with Splunk

To build a logger please use the `build_colorized_logger` method. Under the hood, this method will authenticate with Splunk and if it gets a valid token it will enable the `splunk` handlers by default and install the token into the logging configuration dictionary before starting up the python logger.

The `build_colorized_logger` calls the `setup_logging` method that builds the formatters, handlers and the python logging system.

27.13.1 Spylunking

Splunk-ready python logging functions, classes and tools

Please use these environment variables to publish logs and run searches with a local or remote splunk server:

```
export SPLUNK_ADDRESS="splunkenterprise:8088"
export SPLUNK_API_ADDRESS="splunkenterprise:8089"
export SPLUNK_PASSWORD="123321"
export SPLUNK_USER="trex"
export SPLUNK_TOKEN="<Optional pre-existing Splunk token>"
```

Splunk drill down fields with environment variables:

```
export LOG_NAME="<application log name>"
export DEPLOY_CONFIG="<application deployed config like k8 filename>"
export ENV_NAME="<environment name for this application>"
```

Splunk optional tuning environment variables:

```
export SPLUNK_INDEX="<splunk index>"
export SPLUNK_SOURCE="<splunk source>"
export SPLUNK_SOURCETYPE="<splunk sourcetype>"
export SPLUNK_VERIFY="<verify certs on HTTP POST>"
export SPLUNK_TIMEOUT="<timeout in seconds>"
export SPLUNK_QUEUE_SIZE="<num msgs allowed in queue - 0=infinite>"
export SPLUNK_SLEEP_INTERVAL="<sleep in seconds per batch>"
export SPLUNK_RETRY_COUNT="<attempts per log to retry publishing>"
export SPLUNK_RETRY_BACKOFF="<cooldown in seconds per failed POST>"
export SPLUNK_DEBUG="<debug the publisher - 1 enable debug|0 off>"
export SPLUNK_VERBOSE="<debug the sp command line tool - 1 enable|0 off>"
```

Change the absolute path to the logging config JSON file:

```
export SHARED_LOG_CFG=<absolute path to logging config JSON file>
```

spylunking.log.setup_logging.**build_colorized_logger**(*name='lg'*, *config='shared-logging.json'*, *log_level=20*, *log_config_path=None*, *handler_name='console'*, *handlers_dict=None*, *enable_splunk=True*, *splunk_user=None*, *splunk_password=None*, *splunk_address=None*, *splunk_api_address=None*, *splunk_index=None*, *splunk_token=None*, *splunk_handler_name='splunk'*, *splunk_sleep_interval=-1*, *splunk_verify=None*, *splunk_debug=False*)

Build a colorized logger using function arguments and environment variables.

Parameters

- **name** – name that shows in the logger
- **config** – name of the config file
- **log_level** – level to log
- **log_config_path** – path to log config file
- **handler_name** – handler name in the config
- **handlers_dict** – handlers dict
- **enable_splunk** – Turn off splunk even if the env keys are set True by default - all processes that have the SPLUNK_* env keys will publish logs to splunk
- **splunk_user** – splunk username - defaults to environment variable: SPLUNK_USER
- **splunk_password** – splunk password - defaults to environment variable: SPLUNK_PASSWORD
- **splunk_address** – splunk address - defaults to environment variable: SPLUNK_ADDRESS which is localhost:8088
- **splunk_api_address** – splunk api address - defaults to environment variable: SPLUNK_API_ADDRESS which is localhost:8089
- **splunk_index** – splunk index - defaults to environment variable: SPLUNK_INDEX
- **splunk_token** – splunk token - defaults to environment variable: SPLUNK_TOKEN
- **splunk_handler_name** – splunk log config handler name - defaults to : SPLUNK_HANDLER_NAME
- **splunk_handler_name** – splunk log config handler name - defaults to : SPLUNK_HANDLER_NAME
- **splunk_sleep_interval** – optional splunk sleep interval
- **splunk_verify** – splunk verify - defaults to environment variable: SPLUNK_VERIFY=<1|0>
- **splunk_debug** – print out the connection attempt for debugging Please Avoid on production…

spylunking.log.setup_logging.**setup_logging**(*default_level=20*, *default_path=None*, *env_key='LOG_CFG'*, *handler_name='console'*, *handlers_dict=None*, *log_dict=None*, *config_name=None*, *splunk_host=None*, *splunk_port=None*, *splunk_index=None*, *splunk_token=None*, *splunk_verify=False*, *splunk_handler_name='splunk'*, *splunk_sleep_interval=-1*, *splunk_debug=False*)

Setup logging configuration

Parameters

- **default_level** – level to log
- **default_path** – path to config (optional)
- **env_key** – path to config in this env var
- **handler_name** – handler name in the config
- **handlers_dict** – handlers dict
- **log_dict** – full log dictionary config
- **config_name** – filename for config
- **splunk_host** – optional splunk host
- **splunk_port** – optional splunk port
- **splunk_index** – optional splunk index
- **splunk_token** – optional splunk token
- **splunk_verify** – optional splunk verify - default to False
- **splunk_handler_name** – optional splunk handler name
- **splunk_sleep_interval** – optional splunk sleep interval
- **splunk_debug** – optional splunk debug - default to False

class spylunking.log.setup_logging.**SplunkFormatter**(**args*, ***kwargs*)

add_fields(*log_record*, *record*, *message_dict*)

Parameters

- **log_record** – log record
- **record** – log message
- **message_dict** – message dict

format(*record*, *datefmt='%Y:%m:%d %H:%M:%S.%f'*)

Parameters **record** – message object to format

get_current_fields()

set_fields(*new_fields*)

Change the fields that will be added in on a log

Parameters **new_fields** – new fields to patch in

updated_current_fields(*update_fields*)

Parameters **update_fields** – dict with values for updating fields_to_add

spylunking.log.setup_logging.**console_logger**(*name='cl'*, *config='shared-logging.json'*, *log_level=20*, *log_config_path=None*, *handler_name='console'*, *handlers_dict=None*, *enable_splunk=False*, *splunk_user=None*, *splunk_password=None*, *splunk_address=None*, *splunk_api_address=None*, *splunk_index=None*, *splunk_token=None*, *splunk_handler_name=None*, *splunk_sleep_interval=-1*, *splunk_verify=None*, *splunk_debug=False*)

Build the full console logger

Parameters

- **name** – name that shows in the logger
- **config** – name of the config file
- **log_level** – level to log
- **log_config_path** – path to log config file
- **handler_name** – handler name in the config
- **handlers_dict** – handlers dict
- **enable_splunk** – Turn off splunk even if the env keys are set False by default - all processes that have the SPLUNK_* env keys will publish logs to splunk
- **splunk_user** – splunk username - defaults to environment variable: SPLUNK_USER
- **splunk_password** – splunk password - defaults to environment variable: SPLUNK_PASSWORD
- **splunk_address** – splunk address - defaults to environment variable: SPLUNK_ADDRESS which is localhost:8088
- **splunk_api_address** – splunk api address - defaults to environment variable: SPLUNK_API_ADDRESS which is localhost:8089
- **splunk_index** – splunk index - defaults to environment variable: SPLUNK_INDEX
- **splunk_token** – splunk token - defaults to environment variable: SPLUNK_TOKEN
- **splunk_handler_name** – splunk log config handler name - defaults to : SPLUNK_HANDLER_NAME
- **splunk_sleep_interval** – optional splunk sleep interval
- **splunk_verify** – splunk verify - defaults to environment variable: SPLUNK_VERIFY=<1|0>
- **splunk_debug** – print out the connection attempt for debugging Please Avoid on production...

spylunking.log.setup_logging.**no_date_colors_logger**(*name='nd'*, *config='shared-logging.json'*, *log_level=20*, *log_config_path=None*, *handler_name='no_date_colors'*, *handlers_dict=None*, *enable_splunk=False*, *splunk_user=None*, *splunk_password=None*, *splunk_address=None*, *splunk_api_address=None*, *splunk_index=None*, *splunk_token=None*, *splunk_handler_name=None*, *splunk_sleep_interval=-1*, *splunk_verify=None*, *splunk_debug=False*)

Build a colorized logger without dates

Parameters

- **name** – name that shows in the logger
- **config** – name of the config file
- **log_level** – level to log
- **log_config_path** – path to log config file
- **handler_name** – handler name in the config
- **handlers_dict** – handlers dict
- **enable_splunk** – Turn off splunk even if the env keys are set False by default - all processes that have the SPLUNK_* env keys will publish logs to splunk
- **splunk_user** – splunk username - defaults to environment variable: SPLUNK_USER
- **splunk_password** – splunk password - defaults to environment variable: SPLUNK_PASSWORD
- **splunk_address** – splunk address - defaults to environment variable: SPLUNK_ADDRESS which is localhost:8088
- **splunk_api_address** – splunk api address - defaults to environment variable: SPLUNK_API_ADDRESS which is localhost:8089
- **splunk_index** – splunk index - defaults to environment variable: SPLUNK_INDEX
- **splunk_token** – splunk token - defaults to environment variable: SPLUNK_TOKEN
- **splunk_handler_name** – splunk log config handler name - defaults to : SPLUNK_HANDLER_NAME
- **splunk_sleep_interval** – optional splunk sleep interval
- **splunk_verify** – splunk verify - defaults to environment variable: SPLUNK_VERIFY=<1|0>
- **splunk_debug** – print out the connection attempt for debugging Please Avoid on production…

spylunking.log.setup_logging.**simple_logger**(*name=''*, *config='shared-logging.json'*, *log_level=20*, *log_config_path=None*, *handler_name='simple'*, *handlers_dict=None*, *enable_splunk=False*, *splunk_user=None*, *splunk_password=None*, *splunk_address=None*, *splunk_api_address=None*, *splunk_index=None*, *splunk_token=None*, *splunk_handler_name=None*, *splunk_sleep_interval=-1*, *splunk_verify=None*, *splunk_debug=False*)

Build a colorized logger for just the message - Used by command line tools.

Parameters

- **name** – name that shows in the logger
- **config** – name of the config file
- **log_level** – level to log
- **log_config_path** – path to log config file
- **handler_name** – handler name in the config
- **handlers_dict** – handlers dict
- **enable_splunk** – Turn off splunk even if the env keys are set False by default - all processes that have the SPLUNK_* env keys will publish logs to splunk
- **splunk_user** – splunk username - defaults to environment variable: SPLUNK_USER
- **splunk_password** – splunk password - defaults to environment variable: SPLUNK_PASSWORD
- **splunk_address** – splunk address - defaults to environment variable: SPLUNK_ADDRESS which is localhost:8088
- **splunk_api_address** – splunk api address - defaults to environment variable: SPLUNK_API_ADDRESS which is localhost:8089
- **splunk_index** – splunk index - defaults to environment variable: SPLUNK_INDEX
- **splunk_token** – splunk token - defaults to environment variable: SPLUNK_TOKEN
- **splunk_handler_name** – splunk log config handler name - defaults to : SPLUNK_HANDLER_NAME
- **splunk_sleep_interval** – optional splunk sleep interval
- **splunk_verify** – splunk verify - defaults to environment variable: SPLUNK_VERIFY=<1|0>
- **splunk_debug** – print out the connection attempt for debugging Please Avoid on production...

27.13.2 Exit Case Handling for the Thread and Multiprocessing Publishers

spylunking.wait_for_exit.**wait_for_exit**(*log*, *debug=False*)

Sleep to allow the thread to pick up final messages before exiting and stopping the Splunk HTTP publisher.

You can decrease this delay (in seconds) by reducing the splunk_sleep_interval or by exporting the env var: export SPLUNK_SLEEP_INTERVAL=0.5

If you set the timer to 0 then it will be a blocking HTTP POST sent to Splunk for each log message. This creates a blocking logger in your application that will wait until each log's HTTP POST was received before continuing.

Note: Reducing this Splunk sleep timer could result in losing messages that were stuck in the queue when the parent process exits. The multiprocessing Splunk Publisher was built to do this, but will not work in certain frameworks like Celery as it requires access to spawn daemon processes to prevent this 'message loss' case during exiting. Applications using this library should ensure there's no critical log messages stuck in a queue when stopping a long-running process.

Parameters

- **log** – created logger
- **debug** – bool to debug with prints

Utilities

Pretty Print a JSON dictionary

Utility - pretty print a dictionary

spylunking.ppj.**ppj**(*json_data*)

Pretty print a json dictionary and return it as a string

Parameters **json_data** – dictionary to print

Get Environment Key

Utility - get environment key

spylunking.ev.**ev**(*k*, *v*)

Get environment key and strip

Parameters

- **k** – environment variable key
- **v** – environment variable value

27.14 Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

N

P

Q

R

S

T

U

W