

---

# **Flask-Track-Usage Documentation**

***Release 2.0.0***

**Steve Milner**

**Jun 03, 2018**



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Requirements . . . . .	3
1.2	Via pip . . . . .	3
1.3	Via source . . . . .	3
<b>2</b>	<b>Usage</b>	<b>5</b>
<b>3</b>	<b>Upgrading Schema</b>	<b>7</b>
3.1	SQL . . . . .	7
<b>4</b>	<b>Blueprint Support</b>	<b>9</b>
4.1	Include . . . . .	9
4.2	Exclude . . . . .	9
<b>5</b>	<b>Configuration</b>	<b>11</b>
5.1	TRACK_USAGE_USE_FREEGEOIP . . . . .	11
5.2	TRACK_USAGE_FREEGEOIP_ENDPOINT . . . . .	11
5.3	TRACK_USAGE_INCLUDE_OR_EXCLUDE_VIEWS . . . . .	12
5.4	TRACK_USAGE_COOKIE . . . . .	12
<b>6</b>	<b>Storage</b>	<b>13</b>
6.1	printer.PrintWriter . . . . .	13
6.2	couchdb.CouchDBStorage . . . . .	14
6.3	mongo.MongoPiggybackStorage . . . . .	14
6.4	mongo.MongoStorage . . . . .	14
6.5	mongo.MongoEngineStorage . . . . .	14
6.6	output.OutputWriter . . . . .	14
6.7	redis_db.RedisStorage . . . . .	14
6.8	sql.SQLStorage . . . . .	14
<b>7</b>	<b>Retrieving Log Data</b>	<b>15</b>
<b>8</b>	<b>Hooks</b>	<b>17</b>
<b>9</b>	<b>Flask-Track-Usage Hooks</b>	<b>19</b>
9.1	How To Use . . . . .	19
9.2	Standard Summary Hooks . . . . .	19



Basic metrics tracking for your [Flask](#) application. The core of library is very light and focuses more on storing basic metrics such as remote ip address and user agent. No extra cookies or javascript are used for usage tracking.

- Simple. It's a Flask extension.
- Supports either include or exempt for views.
- Provides lite abstraction for data retrieval.
- Multiple storage options available.
- Multiple storage options can be used.
- Pluggable functionality for storage instances.
- Supports Python 2.7 and 3+.

The following is optional:

- [freegeoip.net](#) integration for storing geography of the visitor.
- Unique visitor tracking if you are wanting to use Flask's cookie storage.
- Summation hooks for live count of common web analysis statistics such as hit counts.



### 1.1 Requirements

- Flask: <http://flask.pocoo.org/>
- A storage object to save the metrics data with

### 1.2 Via pip

```
$ pip install Flask-Track-Usage
```

### 1.3 Via source

```
$ python setup.py install
```





## CHAPTER 2

---

### Usage

---

```
# Create the Flask 'app'
from flask import Flask, g
app = Flask(__name__)

# Set the configuration items manually for the example
app.config['TRACK_USAGE_USE_FREEGEOIP'] = False
# You can use a different instance of freegeoip like so
# app.config['TRACK_USAGE_FREEGEOIP_ENDPOINT'] = 'https://example.org/api/'
app.config['TRACK_USAGE_INCLUDE_OR_EXCLUDE_VIEWS'] = 'include'

# We will just print out the data for the example
from flask.ext.track_usage import TrackUsage

# We will just print out the data for the example
from flask_track_usage.storage.printer import PrintWriter
from flask_track_usage.storage.output import OutputWriter

# Make an instance of the extension and put two writers
t = TrackUsage(app, [
    PrintWriter(),
    OutputWriter(transform=lambda s: "OUTPUT: " + str(s))
])

# Include the view in the metrics
@t.include
@app.route('/')
def index():
    g.track_var["optional"] = "something"
    return "Hello"

# Run the application!
app.run(debug=True)
```



### 3.1 SQL

#### 3.1.1 1.x -> 2.0.0

1. Edit alembic.ini setting `sqlalchemy.url` to the database that you want to upgrade to 2.0.0.
2. Run the alembic upgrade like so:

```
$ alembic upgrade head INFO [alembic.runtime.migration] Context impl SQLiteImpl. INFO [alembic.runtime.migration] Will assume non-transactional DDL. INFO [alembic.runtime.migration] Running upgrade <base> -> 07c46d368ba4, Initial empty db INFO [alembic.runtime.migration] Running upgrade 07c46d368ba4 -> 0aedc36acb3f, Upgrade to 2.0.0
```

#### 3.1.2 MongoDB

MongoDB should not need modification

#### 3.1.3 Redis

Redis should not need modification

#### 3.1.4 CouchDB

CouchDB should not need modification



Blueprints can be included or excluded from Flask-TrackUsage in their entirety.

### 4.1 Include

```
# ...
app.config['TRACK_USAGE_INCLUDE_OR_EXCLUDE_VIEWS'] = 'include'

# Make an instance of the extension
t = TrackUsage(app, [PrintWriter()])

from my_blueprints import a_blueprint

# Now ALL of a_blueprint's views will be in the include list
t.include_blueprint(a_blueprint)
```

### 4.2 Exclude

```
# ...
app.config['TRACK_USAGE_INCLUDE_OR_EXCLUDE_VIEWS'] = 'exclude'

# Make an instance of the extension
t = TrackUsage(app, [PrintWriter()])

from my_blueprints import a_blueprint

# Now ALL of different_blueprints will be in the exclude list
t.exclude_blueprint(a_blueprint)
```



### 5.1 TRACK\_USAGE\_USE\_FREEGEOIP

**Values:** True, False

**Default:** False

Turn FreeGeoIP integration on or off. If set to true, then geography information is also stored in the usage logs.

Changed in version 1.1.: The default server for using geoip integration changed to extreme-ip-lookup.com

### 5.2 TRACK\_USAGE\_FREEGEOIP\_ENDPOINT

**Values:** URL for RESTful JSON query

**Default:** “<http://extreme-ip-lookup.com/json/{ip}>”

If TRACK\_USAGE\_USE\_FREEGEOIP is True, then this field must be set. Mark the location for the IP address with “{ip}”. For example:

“<http://example.com/json/{ip}/>”

would resolve (with an IP of 1.2.3.4) to:

“<http://example.com/json/1.2.3.4/>”

If using SQLStorage, the returned JSON is converted to a string. You will likely want to pass a field list in the URL to avoid exceeding the 128 character limit of the field.

Set the URL prefix used to map the remote IP address of each request to a geography. The service must return a JSON response.

## 5.3 TRACK\_USAGE\_INCLUDE\_OR\_EXCLUDE\_VIEWS

**Values:** include, exclude

**Default:** exclude

If views should be included or excluded by default.

- When set to *exclude* each routed view must be explicitly included via decorator or blueprint include method. If a routed view is not included it will not be tracked.
- When set to *include* each routed view must be explicitly excluded via decorator or blueprint exclude method. If a routed view is not excluded it will be tracked.

## 5.4 TRACK\_USAGE\_COOKIE

**Values:** True, False

**Default:** False

Turn on unique visitor tracking via cookie on or off. If True, then the unique visitor ID (a quasi-random number) is also stored in the usage logs.



## CHAPTER 6

---

### Storage

---

The following are built-in, ready-to-use storage backends.

---

**Note:** Inputs for `set_up` should be passed in `__init__` when creating a storage instance

---

#### 6.1 `printer.PrintWriter`

---

**Note:** This storage backend is only for testing!

---

## 6.2 couchdb.CouchDBStorage

## 6.3 mongo.MongoPiggybackStorage

## 6.4 mongo.MongoStorage

## 6.5 mongo.MongoEngineStorage

## 6.6 output.OutputWriter

## 6.7 redis\_db.RedisStorage

## 6.8 sql.SQLStorage

<b>Warning:</b> This storage is not backwards compatible with sql.SQLStorage 1.0.x
--

---

## Retrieving Log Data

---

All storage backends, other than `printer.PrintStorage`, provide `get_usage`.

Results that are returned from all instances of `get_usage` should **always** look like this:

```
[
  {
    'url': str,
    'user_agent': {
      'browser': str,
      'language': str,
      'platform': str,
      'version': str,
    },
    'blueprint': str,
    'view_args': str(dict) or None,
    'status': int,
    'remote_addr': str,
    'xforwardedfor': str,
    'authorization': bool
    'ip_info': str or None,
    'path': str,
    'speed': float,
    'date': datetime,
    'username': str,
    'track_var': str(dict) or None,
  },
  {
    ....
  }
]
```

Changed in version 1.1.0: `xforwardedfor` item added directly after `remote_addr`



## CHAPTER 8

---

### Hooks

---

The basic function of the library simply logs on unit of information per request received. This keeps it simple and light.

However, you can also add post-storage “hooks” that are called after the individual log is stored. In theory, anything could be triggered after the storage.

```
# ...
def helloWorld(*kwargs):
    print "hello world!"

# Make an instance of the extension
t = TrackUsage(app, [PrintWriter(hooks=[helloWorld])])
```

In this example, the `helloWorld` function would be called once each time `PrintWriters` output is invoked. The keyword parameters are those found in the [Retrieving Log Data](#) function. (see above) Some Storages/Writers also add more keys.

This library has a list of standardized hooks that are used for log summarizing. They are documented in detail here:

#### **hooks** Standard Summarization Hooks

Not all Stores support all of these hooks. See the details for more information. Usage is fairly straightforward:

```
from flask.ext.track_usage import TrackUsage
from flask.ext.track_usage.storage.mongo import MongoEngineStorage
from flask.ext.track_usage.summarization import sumUrl

t = TrackUsage(app, [MongoEngineStorage(hooks=[sumUrl])])
```



---

## Flask-Track-Usage Hooks

---

The library supports post-storage functions that can optionally do more after the request itself is stored.

### 9.1 How To Use

To use, simply add the list of functions you wish to call from the context of call to `TrackUsage`.

For example, to add `sumRemotes` and `sumLanguages` to a `MongoEngineStorage` storage:

```
from flask.ext.track_usage import TrackUsage
from flask.ext.track_usage.storage.mongo import MongoEngineStorage
from flask.ext.track_usage.summarization import sumRemotes, sumLanguages

t = TrackUsage(app, [MongoEngineStorage(hooks=[sumRemotes, sumLanguages])])
```

### 9.2 Standard Summary Hooks

#### 9.2.1 Time Periods for ALL Summary Hooks

When keeping live metrics for each of the summaries, the following time periods are used:

**Hour** one common unit of “storage” is kept to keep track of the hourly traffic for each hour of a particular date.

**Date** one common unit of “storage” is kept to keep track of the daily traffic for a particular date.

**Month** one common unit of “storage” is kept to keep track of the monthly traffic for a particular month. The month stored is the first day of the month. For example, the summary for March 2017 would be stored under the date of 2017-03-01.

Please note that this library DOES NOT handle expiration of old data. If you wish to delete, say, hourly data that is over 60 days old, you will need to create a separate process to handle this. This library merely adds or updates new data and presumes limitless storage.

## 9.2.2 Summary Targets for ALL Summary Hooks

Currently, the following two data targets are summarized for each of the Time Periods.

**Hits** The total number of requests seen.

**Transfer** The total number of bytes transfered in response to all requests seen.

## 9.2.3 sumUrls – URLs

Traffic is summarized for each URL requested of the Flask server.

## 9.2.4 sumRemotes – remote IPs

Traffic is summarized for each remote IP address seen by the Flask server.

## 9.2.5 sumUserAgents – user agent clients

Traffic is summarized for each client (aka web browser) seen by the Flask server.

## 9.2.6 sumLanguages – languages

Traffic is summarized for each language seen in the requests sent to the Flask server.

## 9.2.7 sumServer – site-wide server hits/traffic

Traffic is summarized for all requests sent to the Flask server. This metric is mostly useful for diagnosing performance.

## 9.2.8 sumVisitors – unique visitors (as tracked by cookies)

Traffic is summarized for each unique visitor of the Flask server. For this to function, the optional `TRACK_USAGE_COOKIE` function must be enabled in config.

This metric is limited by the cookie technology. User behavior such as switching browsers or turning on “anonymous mode” on a browser will make them appear to be multiple users.

## 9.2.9 sumGeo – physical country of remote IPs

Traffic is summarized for the tracked geographies of remote IPs seen by the Flask server. For this to properly function, the optional `TRACK_USAGE_FREEGEOIP` config must be enabled. While the geography function provides a great deal of information, only the country is used for this summarization.