django-throttle-requests Documentation

Release 0.5.0

Lewis Sobotkiewicz

Oct 03, 2017

Contents

Contents:

CHAPTER 1

Introduction

In the context of web applications, limiting the number of requests a host or user makes solves two problems:

- withstanding Denial-of-service attacks (rate-limiting)
- ensuring that a user doesn't consume too many resources (throttling)

Rate-limiting is often accomplished with firewall rules on a device, `iptables`, or web server. They are enforced at the network or transport layer before the request is delivered to the application. For example, a rule such as "An IP address may make no more than 20 reqs/sec" would queue, or simply drop any requests that exceeded the maximum rate, and the application will not receive the request.

Throttling can be thought of as application middleware that maintains a count of users' requests during a specific time period. If an incoming request exceeds the maximum for the time period, the user receives a response (e.g. HTTP 403) containing a helpful error message.

A good example of throttling is Twitter's controversial API rate-limiting. Twitter enforces several types of limits depending on the type of access token used and the API function used. An example of a rule is "a user may make no more than 150 requests per 15-minute window".

Note: Although Twitter uses the term `rate limiting`, I find it helpful to distinguish the concepts of network-layer rate limiting versus application-specific request limiting (throttling).

CHAPTER 2

Installation

1. Install the library with pip:

```
sudo pip install django-throttle-requests
```

2. Add the directory `throttle` to your project's `PYTHONPATH`.
3. Insert the following configuration into your project's settings:

```
THROTTLE_ZONES = {
    'default': {
        'VARY':'throttle.zones.RemoteIP',
        'NUM_BUCKETS':2,  # Number of buckets worth of history to keep. Must be␣
↪at least 2
        'BUCKET_INTERVAL':15 * 60  # Period of time to enforce limits.
        'BUCKET_CAPACITY':50,  # Maximum number of requests allowed within BUCKET_
↪INTERVAL
    },
}

# Where to store request counts.
THROTTLE_BACKEND = 'throttle.backends.cache.CacheBackend'

# Force throttling when DEBUG=True
THROTTLE_ENABLED = True
```

4. Use the `@throttle` decorator to enforce throttling rules on a view:

```
from throttle.decorators import throttle

@throttle(zone='default')
def myview(request):
    ...
```

CHAPTER 3

Configuration

django.conf.settings.**THROTTLE_ENABLED**

Default not `settings.DEBUG`

Optional boolean value that is used to control whether or not throttling is enforced. To test throttling when `DEBUG` is `True`, you must also explicitly set `THROTTLE_ENABLED = True`.

django.conf.settings.**THROTTLE_BACKEND**

The path to the class that implements the backend storage mechanism for per-user request counts.

django.conf.settings.**THROTTLE_ZONES**

A dictionary that contains definitions of the rate limiting rules for your application.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

T